



DATA STRUCTURES & ALGORITHMS *in* PYTHON



JOHN CANNING ALAN BRODER
ROBERT LAFORE

FREE SAMPLE CHAPTER |



جان كيننگ
ايلن بروڈر
رابرٹ لافور

ڈیٹا

ڈھانچے اور

ازگر میں الگورتھم

◆ Addison-Wesley

بوسٹن • کولمبس • نیویارک • سان فرانسسکو • ایمسٹرڈیم • کیپ ٹاؤن
دہلی • لندن • میڈرڈ • میلان • میونخ • پیرس • مونٹریال • ٹورنٹو • دہلی • میکسیکو سٹی
ساؤ پالو • سڈنی • ہانگ کانگ • سیول • سنگاپور • تائپے • ٹوکیو

مینوفیکچررز اور بیچنے والے اپنی مصنوعات کو الگ کرنے کے لیے استعمال کیے گئے بہت سے عہدوں کا دعویٰ بطور ٹریڈ مارک کیا جاتا ہے۔ جہاں وہ عہدہ اس کتاب میں نظر آتا ہے، اور پبلشر کو ٹریڈ مارک کے دعوے کا علم تھا، وہ عہدہ ابتدائی بڑے حروف کے ساتھ یا تمام بڑے حروف میں پرنٹ کیا گیا ہے۔

مصنفین اور پبلشر نے اس کتاب کی تیاری میں احتیاط برتی ہے، لیکن کسی قسم کی کوئی ظاہری یا مضمحل وارنٹی نہیں دیں گے اور غلطیوں یا کوتاہی کے لیے کوئی ذمہ داری قبول نہیں کریں گے۔ یہاں موجود معلومات یا پروگراموں کے استعمال سے یا اس کے نتیجے میں ہونے والے واقعاتی یا نتیجہ خیز نقصانات کے لیے کوئی ذمہ داری قبول نہیں کی جاتی ہے۔

اس ٹائٹل کو بڑی مقدار میں خریدنے کے بارے میں معلومات کے لیے، یا خاص فروخت کے مواقع کے لیے (جس میں الیکٹرانک ورژن شامل ہو سکتے ہیں؛ حسب ضرورت کور ڈیزائن؛ اور آپ کے کاروبار کے لیے مواد، تربیتی امداد، مارکیٹنگ کی توجہ، یا برانڈنگ کی دلچسپیاں)، براہ کرم ہمارے کارپوریٹ سے رابطہ کریں۔ licensing@pearsoned.com یا 382-3419 (800) پر سیلز ڈیپارٹمنٹ۔

سرکاری فروخت سے متعلق پوچھ گچھ کے لیے، براہ کرم governmentsales@pearsoned.com سے رابطہ کریں۔

امریکہ سے باہر فروخت کے بارے میں سوالات کے لیے، براہ کرم intlcs@pearson.com پر رابطہ کریں۔

ہمیں ویب پر دیکھیں: informit.com/aw آف کانگریس کنٹرول نمبر: Copyright © 2023 Pearson Education, Inc. 2022910068

جملہ حقوق محفوظ ہیں۔ یہ اشاعت کاہی رائٹ کے ذریعہ محفوظ ہے، اور کسی بھی ممنوعہ تولید، بازیافت کے نظام میں ذخیرہ کرنے، یا کسی بھی شکل میں یا کسی بھی ذریعہ، الیکٹرانک، میکینیکل، تصویری کاہی، ریکارڈنگ، یا اسی طرح سے نشر کرنے سے پہلے ناشر سے اجازت حاصل کی جانی چاہیے۔ Pearson Education Global Rights & Permissions Department کے اندر اجازتوں، درخواست فارموں اور مناسب رابطوں کے بارے میں معلومات کے لیے، براہ کرم www.pearson.com/permissions پر جائیں۔

یہاں موجود معلومات کے استعمال کے سلسلے میں کوئی پینٹ ذمہ داری قبول نہیں کی جاتی ہے۔ اگرچہ اس کتاب کی تیاری میں تمام احتیاط برتی گئی ہے، لیکن ناشر اور مصنف غلطیوں یا کوتاہی کی ذمہ داری قبول نہیں کرتے۔ نہ ہی یہاں موجود معلومات کے استعمال کے نتیجے میں ہونے والے نقصانات کے لیے کوئی ذمہ داری قبول نہیں کی جاتی۔

ISBN-13: 978-0-13-485568-4

ISBN-10: 0-13-485568-X

سکاؤٹ آٹومیٹڈ پرنٹ کوڈ

چیف ایڈیٹر

مارک ٹاؤب

ڈائریکٹر، آئی ٹی پی پروڈکٹ

انتظام

بریٹ ہارٹو

حصول ایڈیٹر

کم اسپینسلی

ترقیاتی ایڈیٹر

کرس زین

مینجنگ ایڈیٹر

سینڈرا شووڈر

پروجیکٹ ایڈیٹر

مینڈی فرینک

کاہی ایڈیٹر

چک بچسن

اشاریہ ساز

چیرل لینسر

پروف ریڈر

باربرا میک

ادارتی معاون

سنڈی ٹیڈرز

ڈیزائنر

چوٹی پرسرٹسٹھ

کمپوزٹر کوڈ منتر

ایک نظر میں مشمولات

1	جائزہ
2	صفیں 29 3 سادہ چھانٹ رہا
103 75 4 ڈھیر اور قطاریں
157 103
157 157 لنک شدہ فہرستیں
229 229 تکرار
285 8 285 8 بائری
335 9 2-3-4 335 9 2-3-4 ٹریس
401 401 ذخیرہ
463 463 10 وی وی ایل اور سرخ سیاہ درخت
525 525 11 پیش ٹیبلز
597 13 597 13 12 مقامی اعداد و شمار کے ڈھانچے
665 14 665 14 ڈھیر
705 15 705 15 گراف
767 16 767 16 گراف
813 813 کیوں
اپینڈکس	
833 B 833 B A رنگ دی ویڈیو لائزیشن
841 C 841 C پڑھنا
845 845 جوابات
859 859 اشاریہ

فہرست کا خانہ

1 جائزہ

1

ڈیٹا سٹرکچر اور الگورتھم کیا ہیں؟.....	1
ڈیٹا سٹرکچرز کا جائزہ.....	4
الگورتھم کا جائزہ.....	6
تعریفیں.....	6
ڈیٹا بیس.....	6
ریکارڈ.....	6
میدان.....	7
کلیکچرز.....	7
ڈیٹا بیس بمقابلہ ڈیٹا	7
Python میں پروگرامنگ.....	8
ترجمان.....	8
متحرک.....	8
ٹائپنگ.....	12
تسلسل.....	13
لوپنگ اور	13
تکرار.....	15
کنٹر الثانوی	15
تفویض.....	17
درآمد کرنا	17
ماڈیولز.....	18
فنکشنز اور سب	18
روٹینز.....	19
فہرست فہمیاں.....	20
پیوٹنگ.....	22
آبجیکٹ اور اینڈ	22
خلاصہ.....	27
تجربیات.....	28

2 صفیں

29

اری ویژولائزیشن ٹول.....	30
تلاش کر رہا	30
.....	31
نقل کا	31
مسئلہ.....	35
ارے کلاس کو لاگو کرنے کے لیے ازگر کی فہرستوں کا	35
استعمال.....	37
ایک صف بنانا.....	37
فہرست عناصر تک	37
رسائی.....	38
ایک بہتر صف کے طبقے کا نفاذ.....	38
.....	43
The OrderedArray Visualization Tool.....	43
.....	47
.....	48
لکیری تلاش.....	48
.....	48
ثنائی تلاش.....	48

آرڈر شدہ اررے کلاس کے لیے ازگر کوڈ	52
تلاش (طریقہ کے ساتھ بائری تلاش	52
کریں..... The OrderedArray Class	52
ترتیب	
شده صفوں کے 53 فوائد	57
لوگارتھمز	58
مساوات	59
مخالف 2 کو ایک طاقت میں بڑھانے	
کا	60
ذخیرہ کرنا اشیاء	
The OrderedRecordArray کلاس	60
نوٹیشن	61
نوٹیشن	65
غیر ترتیب شدہ صف میں داخل کرنا:	
مسلسل	66
66 لکیری تلاش: N کے متناسب	
66 نٹائی تلاش: لاگ (N) کے	
متناسب	67
67 مسلسل کی ضرورت نہیں	

67

بر چیز کے لیے Arrays کا استعمال کیوں نہیں کرتے؟

69 خلاصہ

69 سوالات 70

72 جزیات

73 پروگرامنگ پروجیکٹس

سادہ ترتیب

آپ بہ کیسے کریں گے؟

76

بالے کی ترتیب

77 بال کے کھلاڑیوں پر بلبل چھانٹنا

سادہ ترتیب دینے والا ویژولائزیشن ٹول

79 ازگر کوڈ برائے ایک بلبلے کی

ترتیب

81

غیر متزلزل

82

بلبلے کی ترتیب کی کارکردگی

82 انتخاب کی

ترتیب

83

فٹ بال پلیئرز پر سلیکشن کی ترتیب

83 اے مختصر

تفصیل

83 مزید تفصیل

83 سادہ ترتیب دینے والے ویژولائزیشن ٹول میں سلیکشن کی ترتیب

85 سلیکشن کی

ترتیب کے لیے ازگر کوڈ

85

متغیر

86

انتخاب کی ترتیب کی کارکردگی

86

اندراج کی ترتیب

87

فٹ بال کے کھلاڑیوں پر اندراج کی چھانٹی

87 جزوی

چھانٹنا

87 نشان زد

کھلاڑی

87 سادہ ترتیب دینے والے ویژولائزیشن ٹول میں

اندراج کی ترتیب

90 Python Code for Insertion Sort

89

اندراج کی ترتیب میں متغیرات	91
اندراج کی ترتیب کی کارکردگی	91
کا کوڈ	91
استحکام	96
موازنہ کرنا	96
خلاصہ	98
سوالات	98
تجربات	100
پروجیکٹس	101

104 ڈھیر اور قطار

مختلف استعمال کے معاملات کے لیے مختلف ڈھانچے	103
سٹوریج اور بازیافت کا نمونہ	103
رسائی	104
مزید خلاصہ	104
ڈھیر	104
پوسٹل کی تشبیہ	105
ٹول	106
اسٹیک کے لیے ازگر کوڈ	108
اسٹیک مثال: ایک لفظ کو	108
کارکردگی	112
اسٹیک مثال: ڈیلیمیٹر میچنگ	113
ڈھیروں کی	113
قطاریں	116
ایک ڈھمبیر مسئلہ	118
قطار، ویزو لائیکس، ٹول	118
قطار کے لیے 119 ازگر کا کوڈ	120
قطاروں کی	120
کارکردگی	125
ٹیک	125
ترجیحی قطاریں	126
ترجیحی قطار، ویزو لائیکس، ٹول	127
ازگر، کوڈ، ترجیحی قطار، سفر کے بارے میں، کیا، خیال	129
؟	132

ریاضی کے تاثرات کو پارس کرنا	132
پوسٹ فکس	132
نوٹیشن	133
انفکس کا پوسٹ فکس میں ترجمہ کرنا	134
انفکس کیلکولیٹر	134
ٹول	142
پوسٹ فکس ایکسپریشنز کا جائزہ لینا	148
خلاصہ	151
سوالات	152
تجربات	154
پروجیکٹس	155

فہرست فہرستیں۔

..... لنکس	158
..... حوالہ جات اور بنیادی اقسام	160
..... 164 لنکڈ لسٹ ویژولائزیشن ٹول	164
..... تلاش کا بٹن	166
..... ڈیلیٹ بٹن	166
..... نیا بٹن	167
..... دوسرے بٹن	167
..... ایک سادہ لنک کی فہرست	167
..... طریقہ	168
..... لنک شدہ فہرستوں کو عبور کرنا	169
..... تلاش کریں	170
..... لنک شدہ فہرستوں میں حذف کرنا	174
..... دوبری ختم شدہ فہرستیں	177
..... لنک شدہ فہرست کی کارکردگی	183
..... آجیکٹ	184
..... 184 ایک قطار ایک لنک شدہ فہرست کے ذریعہ لاگو کی گئی ہے	187
..... خلاصہ	189
..... ADT	189
..... 191	
..... ADTs بطور ڈیرائن ٹول	191
..... فہرستیں	192
..... آرڈر شدہ فہرستوں کے لیے ازگرو کوڈ	193
..... کارکردگی	198
..... دوبری منسلک فہرستیں	198
..... کرنا	201
..... وسط میں داخل کرنا اور حذف کرنا	204
..... ڈیک کی بنیاد کے طور پر دوبری منسلک فہرست	208
..... لسٹ	209
..... تکرار کرنے والے	211
..... ایٹریٹر کے بنیادی طریقے	212
..... تکرار کرنے کے دیگر طریقے	216
..... ازگرو میں تکرار کرنے والے	217
..... تجویزات	226
..... پروجیکٹس	227
..... 229	
..... 6 تکرار	
..... مثلث نمبر	230
..... کرنا	231

تکرار کا استعمال کرتے ہوئے نوبی اصطلاح کا پتہ لگانا	232
واقعی کیا ہو رہا ہے؟	234
تکرار موثر ہے؟	236
ریاضی کی شمولیت	237
جزئیات	237
اینگرامس	239
ایک تکراری بائری تلاش	242
تکرار لوپ کی جگہ لے لیتا ہے۔	243
تقسیم اور فتح الگورتھم	245
بنوئی کا ٹاور	246
ویژولائزیشن ٹول کا ٹاور	246
ہرکت پذیر، ابرام	250
انسضمام کے ساتھ تکراری عمل چھانٹنا	255
دو ترتیب شدہ صفوں کو ضم کرنا	255
ضم کر کے چھانٹنا	255
ذیلی ریجز کو ضم کرنا	260
کوڈ کی جانچ کرنا	263
انسضمام کی کارکردگی	264
تکرار کو ختم کرنا	264
تکرار اور اسٹیک	267
ایک تکراری فعل کی نقل کرنا : مثلث	270
کچھ دلچسپ تکراری درخواستیں	275
طاقات کے لیے نمبر بڑھانا	275
کنپٹی کا مسئلہ	277
مجموعے: ٹیم چننا	277
خلاصہ	280
سوالات	281
تجربات	283
پروگرامنگ پروجیکٹس	283
لی درجے کی ترتیب	285
شیلسورٹ	285
اندراج کی ترتیب: بہت زیادہ کاپیاں	286
N-	286
چھانٹنا	286
کم ہونے والے فرق	288
ایڈوانسڈ سورٹنگ ویژولائزیشن ٹول	289

.....	شیلسورٹ کے لیے ازگر کوڈ	291 دوسرے وقفہ کے
.....	سلسلے	293 شیلسورٹ کی کارکردگی
.....	تقسیم	294 تقسیم کا
.....	عمل	295
.....	جنرل پارٹیشننگ الگورتھم	297 کی کارکردگی پارٹیشن
.....	الگورتھم	302
.....	Quickort	301 بنیادی Quickort
.....	الگورتھم	302 ایک محور قدر کا انتخاب
.....	304 A پہلی فوری ترتیب کا نفاذ	310
.....		
.....	SO(N2) کارکردگی میں تنزلی	312 میڈین آف تھری
.....	پارٹیشننگ	313 چھوٹے پارٹیشنز کو بینڈل کرنا
.....	315 مکمل فوری ترتیب کا نفاذ	315 تکرار کو
.....	بٹانا	318 Quickort کی
.....	کارکردگی	318 ریڈکس
.....	ترتیب	320
.....	ریڈکس ترتیب کے لیے الگورتھم	321 ریڈکس ترتیب پروگرام ڈیزائن
.....	کرنا	321 ریڈکس ترتیب کی کارکردگی
.....	322 ریڈکس کی ترتیب کو عام کرنا	323
.....		
.....	ٹمسورٹ	324
.....	Timsort کی کارکردگی	327
.....	خلاصہ	327 سوالات
.....		
.....	تجربات	331 پروگرامنگ
.....	پروجیکٹس	332
.....		
.....	8 ہائٹری درخت	335
.....	ہائٹری درخت کیوں استعمال کریں؟	335 ایک ترتیب شدہ صف
.....	میں آہستہ اندراج	335 لنک شدہ فہرست میں سست تلاش
.....	کرنا	336 درخت ریسکیو کے لیے
.....		
.....	درخت کیا ہے؟	336
.....	درخت کی اصطلاحات	337
.....	جڑ	338
.....	راستہ	338
.....	والدین	338

بچہ.....	338
بہن بھائی.....	339
پتی.....	339
ذیلی درخت.....	339
دورہ.....	339
گزرنا.....	339
درجے.....	339
چایاں.....	339
ٹریز.....	339
بائٹری تلاش کے.....	339
درخت.....	340
تشبیہ.....	340
کیسے کام کرنے ہیں؟.....	341
ٹول.....	341
تلاش کرنا.....	344
تلاش کرنا.....	344
کارکردگی.....	348
نوڈ.....	350
استعمال کرتے ہوئے.....	351
گزرنا.....	353
ٹراورسل.....	353
پری آرڈر اور پوسٹ آرڈر ٹراورسل.....	355
ٹراورسل کے لیے ازگر کا کوڈ.....	361
آرڈر.....	363
کم سے کم اور زیادہ سے زیادہ کلیدی اقدار کا پتہ لگانا.....	365
کرنا.....	366
نہیں ہے.....	367
کیس: 2: جس نوڈ کو حذف کیا جانا ہے اس کا ایک بچہ ہے.....	367
کیس: 3: جس نوڈ کو حذف کیا جانا ہے اس کے دو بچے ہیں.....	370
ثنائی تلاش کے درختوں کی کارکردگی.....	375
گیا.....	377
درختوں کی طباعت.....	379
کیز.....	381
پروگرام.....	382
386.....	386
کریکٹر کوڈز.....	386
بف مین ٹری کے ساتھ ڈی کوڈنگ.....	388
تخلیق.....	389

پیغام کوڈ کرنا	391
خلاصہ	393 سوالات 394
....._تجربات
.....	396 پروگرامنگ پروجیکٹس
.....	397
.....	401
.....	401 2-3-4 درختوں کا تعارف
.....	402 نام میں کیا ہے؟
.....	403 2-3-4 درختوں کی اصطلاحات
.....	403 2-3-4 درخت کی تلاش
.....	404 تنظیم
.....	404 اندراج
.....	405 نوڈ سپلٹس
.....	406 نوڈ سپلٹس پر تقسیم
.....	408 The Tree234 Visualization Tool
.....	409 رینڈم فل اور نئے ٹری بن
.....	409 تلاش کا ہٹن
.....	409 داخل کرنے کا ہٹن
.....	410 تجربات
.....	411 ازگر کوڈ برائے
.....	412 نوڈ کلاس
.....	415
.....	421 سفر
.....	423 حذف کرنا
.....	430 2-3-4 درختوں کی کارکردگی
.....	431 رفتار
.....	432 تقاضے
.....	432 2-3 درخت
.....	433 اندرونی نوڈس میں تقسیم کو فروغ
.....	435 عمل درآمد
.....	438 ایکسٹرنل 437 درختوں کی افادیت
.....	438 بیرونی ڈیٹا تک
.....	439 ترتیب وار ترتیب
.....	442 رسائی
.....	444 بی درخت
.....	450 پیچیدہ تلاش کے
.....	452 بیرونی چھاتنا فائلیں
.....	453

نیچے کے راستے پر گردشیں	505 حذف
کرنا.....	508 ..
سرخ سیاہ درختوں کی افادیت	509 2-3-4 درخت اور سرخ سیاہ
درخت	510
2-3-4 سے سرخ سیاہ میں تبدیلی	510
آپریشنل مساوات	512 سرخ سیاہ درخت کا نفاذ 514
خلاصہ	515 سوالات 517
.....	تجربات
.....	520
.....	521 پروگرامنگ پروجیکٹس
.....	525
.....	11 پیش ٹیبلز
.....	پیشنگ کا تعارف
.....	526 اینک اکاؤنٹ نمبرز بطور
.....	کلید
.....	526 ایک لغت
.....	530
.....	527 پیشنگ
.....	تصادم.....
.....	533
.....	اوپن ایڈریسنگ
.....	536 لکیری
.....	تحقیقات
.....	536 ازگر کوڈ برائے اوپن ایڈریسنگ پیش
.....	ٹیبلز.....
.....	544 کوآڈرائٹک پروینگ
.....	554 ڈیل
.....	پیشنگ
.....	559 علیحدہ زنجیریں.....
.....	565 پیش ٹیبل چیننگ ویژولائزیشن ٹول
.....	566 ازگر کوڈ برائے علیحدہ زنجیر
.....	569 پیش
.....	فنکشنز.....
.....	575
.....	فوری حساب کتاب
.....	575 نان ریڈم کیز
.....	576 پیشنگ
.....	578
.....	سٹرنگز
.....	580 پیشنگ کی
.....	کارکردگی
.....	581 اوپن
.....	ایڈریسنگ
.....	581 علیحدہ زنجیریں
.....	583 اوپن ایڈریسنگ بمقابلہ علیحدہ زنجیر.....
.....	587 پیشنگ اور ایکسٹرنل
.....	سٹوریج
.....	588 فائل پوائنٹرز کا جدول.....
.....	588
.....	نان فل بلاکس
.....	588
.....	مکمل بلاکس
.....	589

.....	590 سوالات	592
.....	تجربات	594 پروگرامنگ
.....	پروجیکٹس	595
.....	12 مقامی ڈیٹا سٹرکچرز	597
.....	مقامی ڈیٹا	597 کارٹیشین
.....	کوآرڈینیٹس	597
.....	جغرافیائی نقاط	598 پوائنٹس کے درمیان فاصلوں کا حساب
.....	لگانا	599 کارٹیشین کوآرڈینیٹس کے درمیان فاصلہ
.....	599	
.....	دائرے اور ہائونڈنگ بکس	601 فاصلوں اور دائروں کو واضح
.....	کرنا	601 ہائونڈنگ باکسز
.....	603	
.....	کا ہائونڈنگ باکس کارٹیشین کوآرڈینیٹس میں ایک سوال کا دائرہ	603 جغرافیائی نقاط میں ایک سوال کے دائرے کا
.....	ہائونڈنگ باکس	604
.....	ہائونڈنگ باکسز کو لاگو کرنا	605 سرکل ہائونڈنگ ذیلی
.....	طریقے	607
.....	اس بات کا تعین کرنا کہ آیا دو ہائونڈ آجیکٹ آپس میں ملتے ہیں	609 اس بات کا تعین کرنا کہ آیا ایک
.....	ہائونڈ آجیکٹ مکمل طور پر دوسرے کے اندر موجود ہے	
.....	610	
.....	مقامی ڈیٹا کی تلاش	611 نکات کی
.....	فہرستیں	612
.....	پوائنٹ لسٹ کلاس کی ایک مثال بنانا	612 پوائنٹس داخل
.....	کرنا	613 ایک عین مطابق مماثلت تلاش
.....	کرنا	614 ایک نقطہ کو حذف
.....	کرنا	614 پوائنٹس کو عبور
.....	کرنا	615 قریب ترین میچ تلاش
.....	615	
.....	گرڈز	617
.....	ازگر میں گرڈ کا نفاذ	618 گرڈ کلاس کی ایک مثال بنانا
.....	619 داخل کرنے والے پوائنٹس	620 ایک عین مطابق مماثلت
.....	تلاش کرنا	621 ہٹاؤ اور عملی غور و
.....	فکر	622 حذف کرنا اور عبور کرنا
.....	623 قریب ترین میچ تلاش کرنا	624 کیا سوال کا دائرہ ایک تہہ کے اندر آتا
.....	ہے؟	625 کیا سوال کا دائرہ ایک گرڈ سیل کو آپس میں جوڑتا ہے؟
.....	628	
.....	وزٹ کرنے کے لیے پڑوسی سیلوں کی ترتیب پیدا کرنا	629

یہ سب ایک ساتھ کھینچنا: گڑ کے قریب ترین () کو نافذ کرنا 630
 کوآڈٹریز..... 633 کوآڈ ٹری کلاس کی ایک مثال
 بنانا 635 داخل کرنے والے نکات: ایک تصوراتی جائزہ..... 636 ابہام سے
 بچنا 638 کوآڈ ٹری ویژولائزیشن ٹول 639 کوآڈٹریز کو
 نافذ کرنا: نوڈ کلاس 640 داخل کرنے کا طریقہ..... 641

داخل کرنے کی کارکردگی 644 عین مطابق مماثلت تلاش
 کرنا 644 درست تلاش کی کارکردگی
 645 پوائنٹس کو عبور کرنا .. 645 ایک نقطہ کو حذف
 کرنا 646 قریب ترین میچ تلاش کرنا
 647 تلاش کرنا امیدوار نوڈ 648 قریب ترین نوڈ تلاش
 کرنا 655 .. _ _ نظریاتی کارکردگی اور
 اصلاح 656 عملی غور و فکر .
 656

مزید توسیعات 658
 دیگر آپریشنز 658 اعلیٰ چہتیں
 659 خلاصہ
 سوالات 661
 تجربات 662 پروگرامنگ
 پروجیکٹس 663

13 ڈھیر 665

ہیپس کا تعارف 666 ترجیحی قطاریں، ہیپس، اور 667
 ADTs جزوی طور پر حکم دیا گیا 668 _
 اندراج..... 669
 ہٹانا 670
 دیگر آپریشنز 674 ہیپ ویژولائزیشن ٹول
 674 داخل کرنے کا ہٹن 675

رینڈم ہیپ ہٹن بنائیں 676 مٹانے اور رینڈم فل ہٹن 676

جھانکنے والا ہٹن 677

ہٹائیں زیادہ سے زیادہ ہٹن 677

Heapify ہٹن 677

ٹراورس ہٹن	677
ڈھیروں کے لیے ازگر کوڈ	677
اندراج.....	679
ہٹانا	680
سفر	682
ہیپ آپریشنز کی کارکردگی	683
ڈھیر	684
686 اوپر کی بجائے نیچے چھاننا	686
اسی صف کا استعمال کرتے ہوئے.....	688
688 ہیپسورٹ (اسب روٹین)	691
ہیپسورٹ کی کارکردگی	693
آرڈر کے اعداد و شمار	694
جزوی ترتیب انتہائی قدروں کو تلاش کرنے میں معاون ہے	695
K Highest	700
کارکردگی	696
خلاصہ	701
سوالات	703
تجربیات	703
پروجیکٹس	703
گراف 14	705
گراف کا تعارف	705
تعریفیں	706
گراف کے پہلے استعمال	708
کسی پروگرام میں گراف کی نمائندگی	713
کرنا	709
گراف میں عمودی اور کناروں کو شامل کرنا	713
کلاس	715
سفر اور تلاش	718
گہرائی سب سے پہلے	719
چوڑائی	729
کم سے کم پھیلے ہوئے درخت	734
گراف ویژولائزیشن ٹول میں	734
کم سے کم پھیلے ہوئے درخت	736
از کم پھیلے ہوئے درخت	737
Code	740
Topological Sorting	741
ڈائریکٹڈ	741
Python انحصاری رشتے	742
گرافس	742
ڈائریکٹڈ گراف کی چھانٹی	742
گراف ویژولائزیشن ٹول	743
ٹاپولوجیکل چھانٹی الگورتھم	744
ٹاپولوجیکل اور درخت	746
بنیادی ٹاپولوجیکل ترتیب کے لیے	745
ازگر کا	745

..... 748 ڈائریکٹڈ گرافس میں 753 عبوری
..... 753 وارشل کے الگورتھم کا نفاذ 758
..... 759 760
..... 762 پروگرامنگ 763
..... 767 15 وزنی گراف
..... 767 ایک مثال: جنگل میں نیٹ ورکنگ 768 وزنی گراف ویژولائزیشن ٹول
..... 768 کم سے کم پھیلے ہوئے درخت کی تعمیر: سروے کرنے والوں کو 770 الگورتھم بنانا
..... 775 مختصر ترین راستہ کا مسئلہ 783
..... 784 784
..... 784 ویژولائزیشن ٹول کا استعمال 790 الگورتھم کو نافذ کرنا
..... 794 795 تمام جوڑوں کا مختصر ترین راستہ کا
..... 797 800 ناقابل علاج
..... 801 802
..... 802 803
..... 805 806
..... 808 809
..... 813 814
..... 814 815
..... 816 817
..... 818 818

رفتار اور الگورتھم	819	لائبریریاں _	820	_
صفیں	820	لنک شدہ	821
فہرستیں	821	822
ثنائی تلاش کے درخت	822	متوازن تلاش کے	822
درخت	822	823
بیش ٹیلز	823	824
عام مقصد کے ذخیرے کے ڈھانچے کا موازنہ کرنا	824	خصوصی ترتیب دینے والے ڈیٹا	824
سٹرکچرز	824	اسٹیک	825
قطار	825	ترجیحی	826
قطار	826	خصوصی ترتیب دینے والے ڈھانچے کا	826
موازنہ	826	چھانٹنا	828
826 خصوصیت ڈیٹا سٹرکچر	828	کوآڈٹریز اور	828
گرڈز	828	گراف	829
828 ایکسٹرنل سٹوریج	829	ترتیب وار	829
ذخیرہ	829	انڈیکسڈ فائلیں	830
829	830
بی درخت	830	830
بیشنگ	830	_ ورچوئل	830
میموری	830	831
آگے	831
اپینڈکس					
A رنگ دی ویژولائزیشنز	833				
ڈویلپرز کے لیے: تصورات کو چلانا اور تبدیل کرنا	834	حاصل			
کرنا	834	گٹ حاصل			
کرنا	835	تصورات حاصل کرنا			
835 مینیجرز کے لیے: ڈاؤن لوڈ کرنا اور تصورات کو چلانا	836	دوسروں کے لیے: انٹرنیٹ پر تصورات			
دیکھنا	837	تصورات کا استعمال کرتے ہوئے			
838			
ایڈیٹ پڑھنا					
ڈیٹا کے ڈھانچے اور الگورتھم	841				

آجیکٹ اورینٹڈ پروگرامنگ لینگویجز 842 آجیکٹ اورینٹڈ ڈیزائن (OOD) اور سافٹ
ویئر انجینئرنگ 842

C سوالات کے جوابات 845

باب 1، "جائزہ" 845 باب 2،
"آرے" 846 باب 3، "سادہ
ترتیب" 847 باب 4، "ڈھیر اور قطاریں"
848 باب 5، "منسلک فہرستیں" 848 باب 6،
"دوبارہ" 849 باب 7، "اعلیٰ درجے کی ترتیب"
850 باب 8، "ثنائی درخت" 851 باب 4-3-2، "9 درخت اور بیرونی ذخیرہ 851
" باب " 10، "ای وی ایل اور سرخ سیاہ درخت 852
باب 11، "پیش ٹیبلز" 853 باب 12، "مقامی ڈیٹا
سٹرکچرز" 853 باب 13، "ڈھیر"
854 باب 14، "گرافس" 855 باب 15، "وزن والے
گراف" 856

تنوع کے لیے پیئرسن کا عزم، ایکوٹیٹی، اور شمولیت

Pearson تعصب سے پاک مواد تخلیق کرنے کے لیے وقف ہے جو تمام سیکھنے والوں کے تنوع کی عکاسی کرتا ہے۔ ہم تنوع کی بہت سی جہتوں کو قبول کرتے ہیں، جن میں نسل، نسل، جنس، سماجی اقتصادی حیثیت، قابلیت، عمر، جنسی رجحان، اور مذہبی یا سیاسی عقائد شامل ہیں لیکن ان تک محدود نہیں۔

تعلیم ہماری دنیا میں مساوات اور تبدیلی کے لیے ایک طاقتور قوت ہے۔ اس میں ایسے مواقع فراہم کرنے کی صلاحیت ہے جو زندگیوں کو بہتر بناتے ہیں اور معاشی نقل و حرکت کو قابل بناتے ہیں۔ جیسا کہ ہم ہر پروڈکٹ اور سروس کے لیے مواد تخلیق کرنے کے لیے مصنفین کے ساتھ کام کرتے ہیں، ہم شمولیت کا مظاہرہ کرنے اور متنوع اسکالرشپ کو شامل کرنے کی اپنی ذمہ داری کو تسلیم کرتے ہیں تاکہ ہر کوئی سیکھنے کے ذریعے اپنی صلاحیتوں کو حاصل کر سکے۔ دنیا کی سرکردہ لرننگ کمپنی کے طور پر، ہمارا فرض ہے کہ ہم تبدیلی کو آگے بڑھانے میں مدد کریں اور اپنے مقصد کے مطابق زندگی گزاریں تاکہ زیادہ سے زیادہ لوگوں کو اپنے لیے ایک بہتر زندگی بنانے اور ایک بہتر دنیا بنانے میں مدد ملے۔

ہماری عزائم جان بوجھ کر ایک ایسی دنیا میں تعاون کرنا ہے جہاں

□□ ہر کسی کے پاس سیکھنے کے ذریعے کامیاب ہونے کا مساوی اور زندگی بھر کا موقع ہے۔

□□ ہماری تعلیمی مصنوعات اور خدمات جامع ہیں اور بھرپور تنوع کی نمائندگی کرتی ہیں۔
سیکھنے والوں کی

□□ ہمارا تعلیمی مواد درست طریقے سے ان سیکھنے والوں کی تاریخوں اور تجربات کی عکاسی کرتا ہے جن کی ہم خدمت کرتے ہیں

□□ ہمارا تعلیمی مواد سیکھنے والوں کے ساتھ گہری بات چیت کا اشارہ کرتا ہے اور انہیں اپنی تعلیم (اور عالمی نظریہ) کو بڑھانے کے لیے تحریک دیتا ہے۔

جب کہ ہم غیرجانبدارانہ مواد پیش کرنے کے لیے سخت محنت کرتے ہیں، ہم اس پیئرسن پروڈکٹ سے متعلق کسی بھی قسم کے خدشات یا ضروریات کے بارے میں آپ سے سننا چاہتے ہیں تاکہ ہم ان کی چھان بین اور ان کا ازالہ کر سکیں۔

براہ کرم کسی بھی ممکنہ تعصب کے بارے میں خدشات کے ساتھ <https://www.pearson.com/report-bias.html> پر رابطہ کریں۔

میری ماں کو، جس نے مجھے علم کی پیاس دی،
میرے والد کو، جنہوں نے مجھے انجینئرنگ کی خوشیاں سکھائیں،
اور جون تک، جس نے دونوں کا پیچھا کرنا ممکن بنایا۔

جان کیننگ

میرے والد سول پروڈر کے لیے، کمپیوٹر سائنس کے علمبردار، راہنمائی
کے لیے۔
میری والدہ مارلن پروڈر کو، ماسٹر ایجوکیٹر، مجھے پڑھانے کی
ترغیب دینے کے لیے۔
فران کو، میری زندگی کو مکمل کرنے کے لیے۔

ایلن پروڈر

ڈاؤن لوڈز، اپ ڈیٹس، اور تصحیحات کے دستیاب ہوتے ہی ان تک رسائی حاصل کرنے کے لیے Python میں ڈیٹا اسٹرکچرز اور الگورتھم کی اپنی کاپی informit.com پر رجسٹر کریں۔ رجسٹریشن کا عمل شروع کرنے کے لیے، informit.com/register پر جائیں اور لاگ ان کریں یا اکاؤنٹ بنائیں۔ پروڈکٹ ISBN 9780134855684 درج کریں اور جمع کروائیں پر کلک کریں۔ عمل مکمل ہونے کے بعد، آپ کو "رجسٹرڈ پروڈکٹس" کے تحت کوئی بھی دستیاب بونس مواد مل جائے گا۔

کتاب کے بارے میں دوسرے قارئین کے ساتھ گفتگو میں شامل ہونے کے لیے کتاب کی ویب سائٹ <https://datastructures.live> ملاحظہ کریں ، اور ان تصورات کے بارے میں مزید جانیں جو ڈیٹا کے ڈھانچے کو زندہ کرتے ہیں۔

اعترافات

جان کیننگ اور ایلن بروڈر سے

اس کتاب کا رابرٹ لافور کا جاوا پر مبنی ورژن کئی سالوں سے دنیا بھر میں ڈیٹا سٹرکچر کورسز اور پیشہ ور افراد کے حوالہ جات میں ایک اہم مقام رہا ہے۔ جب یسہوا یونیورسٹی کے اسٹرن کالج فار ویمن میں ایلن کا ڈیٹا سٹرکچر کورس پائتھون پر چلا گیا تو کورس میں لافور کی کتاب کا استعمال نہ کرنا ایک حقیقی نقصان تھا۔ اس طرح ہم خاص طور پر اس نئے اور نظر ثانی شدہ ایڈیشن کو Python پروگرامرز اور طالب علموں کی دنیا میں لانے میں خوش ہیں۔

ہم سٹرن کے بہت سے طلباء کا شکریہ ادا کرنا چاہتے ہیں جنہوں نے پچھلے کئی سالوں میں براہ راست یا بالواسطہ اس کتاب میں تعاون کیا۔ Lafore کے جاوا کے نفاذ کے ابتدائی Python ورژن ایلن کے Python پر مبنی کورسز میں مرکزی حیثیت رکھتے تھے، اور Stern طالب علم کے تاثرات نے کوڈ کی وضاحت کو بہتر بنانے، اس کی کارکردگی کو بہتر بنانے، اور بعض اوقات کیڑے کی نشاندہی کرنے اور درست کرنے میں بھی مدد کی!

اس نئے ایڈیشن کے ابتدائی مسودوں پر ان کی قیمتی آراء اور سفارشات کے لیے، ہم ایلن کے ڈیٹا سٹرکچر کورسز میں بہت سے طلباء کے شکر گزار ہیں، جن میں ایسٹی بروکس، اڈینا بروس، جولیا چیس، ہانا فشر، لیمر کوہنم، ایلیشیوا کوہن، شیرا اورلین، شیرا پھم شامل ہیں۔ ، Jennie Peled, Alexandra Roffe, Avigail Royzenberg, Batia Segal, Penina Waghalter, and Esther Werblowsky۔ اگر ہم نے کسی کا نام چھوڑ دیا ہے تو معذرت خواہ ہیں۔

اس کتاب کے آپ کے مطالعہ کو بڑھانے کے لیے ڈیٹا سٹرکچر ویژولائزیشن کا ایک اوپن سورس پیکیج دستیاب ہے، اور اسٹرن طلباء نے بصری سازی سافٹ ویئر کی ترقی میں ایک فعال کردار ادا کیا۔ جان اور ایلن اس پراجیکٹ کے اسٹرن اسٹوڈنٹ کے علمبرداروں اور رہنماؤں کا بہت شکریہ ادا کرتے ہیں، بشمول الانا ریڈنسکی، ایلانا اوفیلہام، ایلانا ٹیٹیلہام، اور للی پولوٹسکی، نیز مندرجہ ذیل ماضی اور حال کے اسٹرن طلباء کے تعاون کرنے والوں اور سرپرستوں: زو ایبودی، آلیٹ ابرون، لارا امر، نطانیہ برنہام، اڈینا بروس، چانی ڈوبن، سارہ اینجل، سارہ گراف، ایوبیگیل بیلمین، میکل کافمین، سرینا کوفمین، ریچل لیڈر، تالیہ لیڈر، شانی لیوس، رینا میلنکوف، اٹارا نیوگوشل، شیرا پامر، میریم، میریم، ریپ، شیرا ساسون، شیرا شنائیڈر، مزال شوئن والڈ، شیرا اسمتھ، ریوا ٹراپ، الیگزینڈرا وولچیک، اور ایستھر وربلوسکی۔ اس کے علاوہ، اسٹرن فیکلٹی کا بہت شکریہ جنہوں نے طلباء کے شرکاء کی رہنمائی کی: پروفیسر ایری شماش، پروفیسر لارنس ٹائل مین، اور پروفیسر جوشوا ویکسمین۔ اگر ہم نے اس فہرست میں سے کسی کو چھوڑ دیا ہے تو ہم معذرت خواہ ہیں۔

یونیورسٹی آف پنسلوانیا کے پروفیسر ڈیوڈ میٹسزیک کا بہت شکریہ ان کے خیالات اور پاورپوائنٹ سلائیڈز کی ابتدائی شراکت کے لیے جب ایلن نے پہلی بار سٹرن میں ڈیٹا سٹرکچر پڑھانا شروع کیا۔ انسٹرکٹرز ریسورسز کے سیکشن میں دستیاب بہت سی سلائیڈز کی اصل اس کی واضح اور اچھی طرح سے ڈیزائن کردہ سلائیڈوں میں ہے۔ نیز، ہم یسہوا یونیورسٹی کے شعبہ ریاضی کے پروفیسر ماریان گیڈیا کے شکر گزار ہیں کہ انہوں نے کروی مثلثیات میں بصیرت کی۔

آخر میں، ہم پیٹرسن کے باصلاحیت ایڈیٹرز کا بہت بڑا قرض دار ہیں جنہوں نے اس کتاب کو حقیقت بنایا: مارک ٹیبر، کم اسپینسلی، مینڈی فرینک، چوٹی پراسرستہ، اور کرس زین۔

ان کی بہت سی صلاحیتوں اور مریضی کی مدد کے بغیر، یہ پروجیکٹ ٹیکسٹ فائلوں، ڈرائنگ اور سورس کوڈ کا محض ایک عجیب مجموعہ ہوگا۔

کتاب کے جاوا پر مبنی ورژن کے لیے رابرٹ لافور سے

پہلے ایڈیشن کا اعتراف، ڈیٹا کے ڈھانچے اور الگورتھم جاوا

اس مختصر اعتراف میں درج ذیل لوگوں (اور بہت سے دوسرے) کے لیے میرا شکریہ ادا نہیں کیا جا سکتا۔ ہمیشہ کی طرح، مچ ویٹ نے جاوا کی چیز کسی اور سے پہلے سمجھ لی تھی۔ اس نے مجھے ایپلٹس کو اس وقت تک اچھالنے دیا جب تک کہ وہ کام نہ کر لیں، اور قیاس آرائیوں کی بد حالی سے اس منصوبے کی مجموعی شکل نکال لی۔ میرے ایڈیٹر، کرٹ سٹیفن، نے بہت اچھے مبصرین کو تلاش کیا، اس بات کو یقینی بنایا کہ ہر کوئی ایک ہی صفحے پر ہے، گیند کو رول کرتا رہے، اور نرمی سے لیکن مضبوطی سے اس بات کو یقینی بنایا کہ میں نے وہی کیا جو مجھے کرنا تھا۔ بیری بینڈرسن نے بہت سے قیمتی تجاویز کے ساتھ پہلے مسودے کا ماہرانہ جائزہ فراہم کیا۔ رچرڈ ایس رائٹ، جونینٹر، تکنیکی ایڈیٹر کے طور پر، تفصیل کے لیے اپنی گہری نظر سے متعدد مسائل کو درست کیا۔ Jaime Niño اور لینز یونیورسٹی کے پی ایچ ڈی، نے مجھے اپنے آپ سے بچانے کی کوشش کی اور کبھی کبھار کامیاب ہو گئے، لیکن میرے نقطہ نظر یا کوڈنگ کی تفصیلات کی کوئی ذمہ داری نہیں اٹھانی چاہیے۔ سوسن والٹن اس منصوبے کے جوہر کو غیر تکنیکی تک پہنچانے میں مدد کرنے میں ایک مضبوط اور قابل تعریف حامی رہی ہے۔ کارمیلا کارواجل علمی دنیا کے ساتھ اپنے روابط بڑھانے میں انمول تھی۔ ڈین شیرف نے نہ صرف CD-ROM کو اکٹھا کیا، بلکہ تیزی سے تیار ہونے والی سافٹ ویئر کی تبدیلیوں پر مجھے اپ ٹو ڈیٹ رکھنے میں انتھک محنت کی۔ آخر میں، سیسیل کامین نے کتاب کو تدوین سے لے کر پیداواری عمل تک منتقلی کے ذریعے اس کی بھرپور طریقے سے مدد کی۔

دوم کو تسلیمات ایڈیشن

میں سامس پبلشنگ میں درج ذیل لوگوں کا شکریہ ادا کرتا ہوں کہ ان کی قابلیت، کوشش اور صبر کے ساتھ اس دوسرے ایڈیشن کی تیاری میں۔ ایکوزیشن ایڈیٹر کیرول ایکرمین اور ڈیولپمنٹ ایڈیٹر سونگلن کیو نے پیچیدہ پیداواری عمل کے ذریعے اس ایڈیشن کی بھرپور رہنمائی کی۔ پروجیکٹ ایڈیٹر Matt Purcell نے نیم لامحدود گرام کی غلطیوں کو درست کیا اور اس بات کو یقینی بنایا کہ ہر چیز صحیح ہے۔ ٹیک ایڈیٹر مائیک کوپک نے پروگراموں کا جائزہ لیا اور مجھے کئی مسائل سے بچایا۔ آخری لیکن کم از کم، ڈین شیرف، پچھلے دور کا ایک پرانا دوست، سامس ویب سائٹ پر میرے کوڈ اور ایپلٹس کا ہنر مند انتظام فراہم کرتا ہے۔

مصنفین کے بارے میں

ڈاکٹر جان کیننگ ایک انجینئر، کمپیوٹر سائنسدان اور محقق ہیں۔ انہوں نے میساچوسٹس انسٹی ٹیوٹ آف ٹیکنالوجی سے الیکٹریکل انجینئرنگ میں ایس بی کی ڈگری حاصل کی اور پی ایچ ڈی کی۔ کالج پارک میں یونیورسٹی آف میری لینڈ سے کمپیوٹر سائنس میں۔ ان کے مختلف پیشوں میں کمپیوٹر سائنس کا پروفیسر، صنعت میں ایک محقق اور سافٹ ویئر انجینئر، اور کمپنی کا نائب صدر شامل ہے۔ اب وہ شکمنت سافٹ ویئر کے صدر ہیں۔

ایلن بروڈر کلینیکل پروفیسر ہیں اور نیو یارک شہر میں یشیوا یونیورسٹی کے اسٹرن کالج برائے خواتین میں کمپیوٹر سائنس کے شعبہ کی چیئر ہیں۔ وہ Python پروگرامنگ، ڈیٹا ڈھانچے، اور ڈیٹا سائنس میں تعارفی اور جدید کورسز پڑھاتا ہے۔ سٹرن کالج میں شامل ہونے سے پہلے، وہ ایک سافٹ ویئر انجینئر تھا، بڑے پیمانے پر ڈیٹا کے تجزیہ کے نظام کو ڈیزائن اور بنا رہا تھا۔ اس نے واٹ اوک ٹیکنالوجیز، انکارپوریٹڈ کو اس کے سی ای او کے طور پر قائم کیا اور اس کی قیادت کی، اور بعد میں فیئر فیکس، ورجینیا میں اس کی جانشین کمپنی، نوویٹا کے چیئرمین اور ساتھی کے طور پر خدمات انجام دیں۔

رابرٹ لافور نے الیکٹریکل انجینئرنگ اور ریاضی میں ڈگریاں حاصل کی ہیں، لارنس برکلے لیبارٹری کے لیے ایک نظام تجزیہ کار کے طور پر کام کیا ہے، اپنی سافٹ ویئر کمپنی کی بنیاد رکھی ہے، اور کمپیوٹر پروگرامنگ کے شعبے میں سب سے زیادہ فروخت ہونے والے مصنف ہیں۔ اس کے کچھ عنوانات ++C میں آجیکٹ اورینٹڈ پروگرامنگ اور جاوا میں ڈیٹا سٹرکچرز اور الگورتھم ہیں۔

تعارف

اس کتاب میں کیا ہے؟ یہ کتاب ان طلباء کے لیے ڈیٹا ڈھانچے اور الگورتھم کے عملی تعارف کے لیے بنائی گئی ہے جنہوں نے ابھی کمپیوٹر پروگرام لکھنا شروع کیا ہے۔ یہ تعارفی عبارت آپ کو کتاب کے بارے میں مزید بتائے گی، یہ کیسے ترتیب دی گئی ہے، ہم امید کرتے ہیں کہ کتاب شروع کرنے سے پہلے قارئین کو کیا تجربہ حاصل ہوگا، اور اسے پڑھنے اور مشقیں کرنے سے آپ کو کیا علم حاصل ہوگا۔

یہ کتاب کس کے لیے ہے۔

ڈیٹا ڈھانچے اور الگورتھم کمپیوٹر سائنس کا مرکز ہیں۔ اگر آپ نے کبھی یہ سمجھنا چاہا ہے کہ کمپیوٹر کیا کر سکتے ہیں، وہ کیسے کرتے ہیں، اور وہ کیا نہیں کر سکتے، تو آپ کو دونوں کے بارے میں گہری سمجھ کی ضرورت ہے (شاید یہ کہنا بہتر ہے کہ "کمپیوٹر کو کیا کرنے میں دشواری ہوتی ہے" کے بجائے وہ نہیں کر سکتے)۔ اس کتاب کو ڈیٹا سٹرکچرز اور/یا الگورتھم کورس میں بطور متن استعمال کیا جا سکتا ہے، جو کثرت سے یونیورسٹی کے کمپیوٹر سائنس کے نصاب کے دوسرے سال میں پڑھایا جاتا ہے۔ تاہم، متن پیشہ ور پروگرامرز، ہائی اسکول کے طلباء، اور کسی اور کے لیے بھی ڈیزائن کیا گیا ہے جسے محض پروگرامنگ زبان جاننے سے اگلا قدم اٹھانے کی ضرورت ہے۔ چونکہ یہ سمجھنا آسان ہے، اس لیے یہ زیادہ رسمی کورس کے ضمنی متن کے طور پر بھی مناسب ہے۔ یہ مثالوں، مشقوں، اور اضافی مواد سے بھری ہوئی ہے، لہذا اسے کلاس روم کی ترتیب سے باہر خود مطالعہ کے لیے استعمال کیا جا سکتا ہے۔

اس کتاب کو لکھنے میں ہمارا نقطہ نظر یہ ہے کہ قارئین کے لیے یہ سمجھنا آسان ہو کہ ڈیٹا ڈھانچے کیسے کام کرتے ہیں اور انہیں عملی طور پر کیسے لاگو کیا جائے۔ یہ کچھ دوسرے متن سے مختلف ہے جو ریاضی کے نظریہ پر زور دیتے ہیں، یا ان ڈھانچے کو کسی خاص زبان یا سافٹ ویئر لائبریری میں کیسے لاگو کیا جاتا ہے۔ ہم نے حقیقی دنیا کے ایپلی کیشنز کے ساتھ مثالوں کا انتخاب کیا ہے اور صرف ریاضی یا غیر واضح مثالوں کے استعمال سے گریز کیا ہے۔ ہم اہم خیالات کو بات چیت کرنے میں مدد کے لیے اعداد و شمار اور تصوراتی پروگرام استعمال کرتے ہیں۔ ہم اب بھی الگورتھم کی پیچیدگی کا احاطہ کرتے ہیں اور یہ بتانے کے لیے ریاضی کی ضرورت ہے کہ کس طرح پیچیدگی کارکردگی کو متاثر کرتی ہے۔

اس کتاب کو پڑھنے سے پہلے آپ کو کیا جاننے کی ضرورت ہے۔

اس کتاب کو استعمال کرنے کے لیے ضروری شرائط ہیں: کچھ پروگرامنگ زبان اور کچھ ریاضی کا علم۔ اگرچہ نمونہ کا کوڈ Python میں لکھا گیا ہے، لیکن آپ کو Python کو جاننے کی ضرورت نہیں ہے کہ کیا ہو رہا ہے۔ اگر آپ نے کچھ طریقہ کار اور/یا آبیجیکٹ پر مبنی پروگرامنگ کی ہے تو ازگزر کو سمجھنا مشکل نہیں ہے۔ ہم نے مثالوں میں نحو کو ہر ممکن حد تک عام رکھا ہے،

مزید خاص طور پر، ہم Python ورژن 3 نحو استعمال کرتے ہیں۔ یہ ورژن Python 2 سے کچھ مختلف ہے، لیکن بہت زیادہ نہیں۔ Python ایک بھرپور زبان ہے جس میں بہت سے بلٹ ان ڈیٹا کی اقسام اور لائبریری ہیں جو اس کی صلاحیتوں کو بڑھاتی ہیں۔ تاہم، ہماری مثالیں دو وجوہات کی بنا پر زیادہ بنیادی تعمیرات کا استعمال کرتی ہیں: یہ دوسری زبانوں سے واقف پروگرامرز کے لیے ان کو سمجھنا آسان بناتا ہے، اور یہ ڈیٹا ڈھانچے کی تفصیلات کو زیادہ واضح طور پر واضح کرتا ہے۔ بعد کے ابواب میں، ہم Python کی کچھ خصوصیات کا استعمال کرتے ہیں جو دوسری زبانوں میں نہیں ملتی ہیں جیسے جنریٹر اور فہرست کی تفہیم۔ ہم وضاحت کرتے ہیں کہ یہ کیا ہیں اور ان سے پروگرامر کو کیسے فائدہ ہوتا ہے۔

یقیناً، اگر آپ ازگر (ورژن 2 یا 3) سے پہلے ہی واقف ہیں تو اس سے مدد ملے گی۔ شاید آپ نے Python کے بہت سے ڈیٹا ڈھانچے میں سے کچھ استعمال کیے ہوں گے اور آپ کو اس بارے میں تجسس ہے کہ ان کو کیسے لاگو کیا جاتا ہے۔ ہم باب 1 میں Python نحو کا جائزہ لیتے ہیں، "جائزہ" ان لوگوں کے لیے جنہیں تعارف یا ریفریشر کی ضرورت ہے۔ اگر آپ نے جاوا، JavaScript، C#، ++C یا پرل جیسی زبانوں میں پروگرام کیا ہے، تو بہت سی تعمیرات سے واقف ہونا چاہیے۔ اگر آپ نے صرف فنکشنل یا ڈومین کے لیے مخصوص زبانوں کا استعمال کرتے ہوئے پروگرام کیا ہے، تو آپ کو ازگر کے بنیادی عناصر سے واقف ہونے کے لیے زیادہ وقت گزارنا پڑ سکتا ہے۔ اس متن کے علاوہ، نوآموز Python پروگرامرز کے لیے بہت سے وسائل دستیاب ہیں، بشمول انٹرنیٹ پر بہت سے ٹیوٹوریلز۔

پروگرامنگ زبان کے علاوہ، ہر پروگرامر کو کیا جاننا چاہیے؟ ریاضی سے الجبرا کے ذریعے ریاضی کا ایک اچھا علم ضروری ہے۔ کمپیوٹر پروگرامنگ علامتی بیرا پھیری ہے۔ بالکل الجبرا کی طرح، اصطلاحات کو دوبارہ ترتیب دینے، انہیں مختلف شکلوں میں رکھنے، اور کچھ حصوں کو زیادہ نمایاں کرنے کے لیے اظہار کو تبدیل کرنے کے طریقے ہیں، یہ سب کچھ ایک ہی معنی کو برقرار رکھتے ہوئے ہے۔ ریاضی میں استفادہ کو سمجھنا بھی ضروری ہے۔ کمپیوٹر سائنس کا زیادہ تر یہ جاننے پر مبنی ہے کہ ایک نمبر کو دوسرے نمبر کی طاقت تک بڑھانے کا کیا مطلب ہے۔ ریاضی کے علاوہ، تنظیم کا ایک اچھا احساس بھی تمام پروگرامنگ کے لیے فائدہ مند ہے۔ یہ جاننا کہ آئٹمز کو مختلف طریقوں سے کیسے ترتیب دیا جائے (وقت کے لحاظ سے، فنکشن کے لحاظ سے، سائز کے لحاظ سے، پیچیدگی کے لحاظ سے، وغیرہ) پروگراموں کو موثر اور برقرار رکھنے کے لیے بہت ضروری ہے۔ جب ہم کارکردگی اور برقرار رکھنے کے بارے میں بات کرتے ہیں، تو کمپیوٹر سائنس میں ان کے خاص معنی ہوتے ہیں۔ کارکردگی زیادہ تر اس بات پر ہوتی ہے کہ چیزوں کی گنتی میں کتنا وقت لگتا ہے لیکن اس میں لگنے والی جگہ کی مقدار بھی ہو سکتی ہے۔ مینٹینینس سے مراد دوسرے پروگرامرز کے ساتھ ساتھ آپ کے پروگراموں کو سمجھنے اور ان میں ترمیم کرنے میں آسانی ہے۔

آپ کو انٹرنیٹ پر چیزوں کو تلاش کرنے، سافٹ ویئر ڈاؤن لوڈ اور انسٹال کرنے اور انہیں کمپیوٹر پر چلانے کے بارے میں بھی علم کی ضرورت ہوگی۔ ویژولائزیشن پروگراموں کو ڈاؤن لوڈ کرنے اور چلانے کی ہدایات اس کتاب کے ضمیمہ A میں دیکھی جا سکتی ہیں۔ انٹرنیٹ نے پروگرامنگ اور کمپیوٹر سائنس سیکھنے کے ٹولز سمیت ٹولز کے کارنو کوپیا تک رسائی کو بہت آسان بنا دیا ہے۔ ہم توقع کرتے ہیں کہ قارئین پہلے ہی جان لیں گے کہ مفید وسائل کیسے تلاش کیے جائیں اور ایسے ذرائع سے بچیں جو نقصان دہ سافٹ ویئر فراہم کر سکتے ہیں۔

آپ اس کتاب سے کیا سیکھ سکتے ہیں۔

جیسا کہ آپ اس کے عنوان سے توقع کر سکتے ہیں، یہ کتاب آپ کو یہ سکھا سکتی ہے کہ ڈیٹا سٹرکچر کس طرح پروگراموں (اور پروگرامرز) کو اپنے کام میں زیادہ موثر بناتے ہیں۔ آپ جان سکتے ہیں کہ کس طرح ڈیٹا آرگنائزیشن اور مناسب الگورتھم کے ساتھ اس کا جوڑا بہت زیادہ اثر انداز ہوتا ہے کہ کمپیوٹنگ وسائل کی دی گئی مقدار کے ساتھ کیا شمار کیا جا سکتا ہے۔ یہ کتاب آپ کو اعداد و شمار کے ڈھانچے کو لاگو کرنے کے طریقے کے بارے میں مکمل طور پر وضاحت دے سکتی ہے، اور یہ آپ کو کسی بھی پروگرامنگ زبان میں ان کو نافذ کرنے کے قابل بنائے گی۔ آپ یہ فیصلہ کرنے کا عمل سیکھ سکتے ہیں کہ کسی خاص پروگرامنگ کی درخواست کو پورا کرنے کے لیے کون سا ڈیٹا ڈھانچہ اور الگورتھم سب سے زیادہ موزوں ہیں۔ شاید سب سے اہم بات، آپ یہ جان سکتے ہیں کہ ایک الگورتھم اور یا ڈیٹا کا ڈھانچہ کسی مخصوص استعمال کے معاملے میں کب ناکام ہو جائے گا۔ ڈیٹا ڈھانچے اور الگورتھم کو سمجھنا کمپیوٹر سائنس کا بنیادی حصہ ہے، جو کہ ایک Python (یا دوسری زبان) پروگرامر ہونے سے مختلف ہے۔

کتاب بنیادی ڈیٹا ڈھانچے کو سکھاتی ہے جو ہر پروگرامر کو جاننا چاہیے۔

قارئین کو سمجھ لینا چاہیے کہ اور بھی بہت ہیں۔ ڈیٹا کے یہ بنیادی ڈھانچے مختلف قسم کے حالات میں کام کرتے ہیں۔ اس کتاب میں آپ جو مہارتیں تیار کرتے ہیں، آپ کو قابل ہونا چاہیے۔

کسی دوسرے ڈیٹا ڈھانچے یا الگورتھم کی تفصیل کو پڑھنے کے لیے اور تجزیہ کرنا شروع کریں کہ آیا یہ اس سے بہتر کارکردگی کا مظاہرہ کرے گا یا اس سے بدتر کارکردگی کا مظاہرہ کرے گا جو آپ نے پہلے سے ہی مخصوص استعمال کے معاملات میں سیکھا ہے۔

یہ کتاب Python کے کچھ نحو اور ساخت کی وضاحت کرتی ہے، لیکن یہ آپ کو اس کی تمام صلاحیتیں نہیں سکھائے گی۔ کتاب میں Python کی مکمل صلاحیتوں کے ذیلی سیٹ کا استعمال کیا گیا ہے تاکہ یہ واضح کیا جا سکے کہ ڈیٹا کے مزید پیچیدہ ڈھانچے کس طرح آسان تعمیرات سے بنائے جاتے ہیں۔ یہ کسی ایسے شخص کو پروگرامنگ کی بنیادی باتیں سکھانے کے لیے نہیں بنایا گیا ہے جس نے کبھی پروگرام نہیں کیا ہے۔ ازگر ایک بہت ہی اعلیٰ سطحی زبان ہے جس میں بہت سے بلٹ ان ڈیٹا ڈھانچے ہیں۔ کچھ زیادہ ابتدائی اقسام کا استعمال کرنا جیسے کہ انٹیجرز کی صفوں یا ریکارڈ ڈھانچے کا استعمال کرنا، جیسا کہ آپ کو C++ میں مل سکتا ہے، ازگر میں کچھ زیادہ مشکل ہے۔ چونکہ کتاب کا فوکس ڈیٹا ڈھانچے کا نفاذ اور تجزیہ ہے، اس لیے ہماری مثالیں ان قدیم قسموں کے لیے قربت کا استعمال کرتی ہیں۔

کچھ Python پروگرامز معیاری لائبریریوں میں زبان کے ساتھ فراہم کردہ مزید خوبصورت تعمیرات کے بارے میں جانتے ہوئے، یہ مثالیں غیر ضروری طور پر پیچیدہ محسوس کر سکتے ہیں۔ اگر آپ کمپیوٹر سائنس، اور خاص طور پر الگورتھم کی پیچیدگی کو سمجھنا چاہتے ہیں، تو آپ کو پرائمیٹوز پر بنیادی کارروائیوں کو سمجھنا چاہیے۔ جب آپ کسی پروگرامنگ لینگویج میں فراہم کردہ ڈیٹا سٹرکچر یا اس کے کسی ایڈ آن ماڈیول سے استعمال کرتے ہیں، تو آپ کو اکثر یہ جاننے کے لیے اس کی پیچیدگی کو جاننا پڑے گا کہ آیا یہ آپ کے استعمال کے معاملے میں اچھی طرح سے کام کرے گا۔ ڈیٹا کے بنیادی ڈھانچے، ان کی پیچیدگیوں، اور تجارتی معاہدوں کو سمجھنے سے آپ کو ان کے اوپر بنائے گئے ڈھانچے کو سمجھنے میں مدد ملے گی۔

تمام ڈیٹا ڈھانچے کو آبجیکٹ اور اینڈ پروگرامنگ (OOP) کا استعمال کرتے ہوئے تیار کیا گیا ہے۔ اگر یہ آپ کے لیے ایک نیا تصور ہے تو، باب 1 کا جائزہ اس بات کا کہ Python میں کلاسز کی تعریف اور استعمال کیسے کیا جاتا ہے OOP کا بنیادی تعارف فراہم کرتا ہے۔ آپ کو اس متن سے OOP کی مکمل طاقت اور فوائد سیکھنے کی امید نہیں رکھنی چاہیے۔ اس کے بجائے، آپ ہر ڈیٹا ڈھانچے کو بطور کلاس لاگو کرنا سیکھیں گے۔ یہ کلاسز OOP میں اشیاء کی قسمیں ہیں اور یہ سافٹ ویئر تیار کرنا آسان بناتی ہیں جنہیں بہت سے مختلف ایپلی کیشنز قابل اعتماد طریقے سے دوبارہ استعمال کر سکتے ہیں۔

کتاب میں بہت سی مثالیں استعمال کی گئی ہیں، لیکن یہ کمپیوٹر سائنس کے کسی خاص اطلاقی علاقے جیسے ڈیٹا بیس، یوزر انٹرفیس، یا مصنوعی ذہانت کے بارے میں کتاب نہیں ہے۔ مثالوں کا انتخاب پروگراموں کی عام ایپلی کیشنز کو واضح کرنے کے لیے کیا جاتا ہے، لیکن تمام پروگرام ایک خاص سیاق و سباق میں لکھے جاتے ہیں، اور یہ وقت کے ساتھ ساتھ تبدیل ہوتے رہتے ہیں۔ 1970 میں لکھا گیا ڈیٹا بیس پروگرام اس وقت بہت ترقی یافتہ دکھائی دے سکتا ہے، لیکن آج یہ بہت معمولی لگتا ہے۔ اس متن میں پیش کردہ امتحانات کو یہ سکھانے کے لیے ڈیزائن کیا گیا ہے کہ ڈیٹا ڈھانچے کو کس طرح لاگو کیا جاتا ہے، وہ کیسے کارکردگی کا مظاہرہ کرتے ہیں، اور نیا پروگرام ڈیزائن کرتے وقت ان کا موازنہ کیسے کیا جاتا ہے۔ امتحانات کو ہر ڈیٹا ڈھانچے کے سب سے زیادہ جامع یا بہترین نفاذ کے طور پر نہیں لیا جانا چاہیے، اور نہ ہی ان تمام ممکنہ ڈیٹا ڈھانچے کے مکمل جائزہ کے طور پر لیا جانا چاہیے جو کسی مخصوص درخواست کے علاقے کے لیے موزوں ہو سکتے ہیں۔

ساخت

ہر باب ڈیٹا ڈھانچے اور متعلقہ الگورتھم کا ایک خاص گروپ پیش کرتا ہے۔

ابواب کے اختتام پر، ہر باب کے اہم نکات اور بعض اوقات پچھلے ابواب کے ساتھ تعلقات کا احاطہ کرنے والے جائزہ سوالات فراہم کرتے ہیں۔ ان کے جوابات ضمیمہ سی، "سوالات کے جوابات" میں مل سکتے ہیں۔ ان سوالات کا مقصد قارئین کے لیے خود امتحان ہے، تاکہ یہ یقینی بنایا جا سکے کہ آپ تمام مواد کو سمجھ گئے ہیں۔

بہت سے ابواب قارئین کو آزمانے کے لیے تجربات تجویز کرتے ہیں۔ یہ انفرادی سوچ کے تجربات، ٹیم اسائنمنٹس، یا کتاب کے ساتھ فراہم کردہ سافٹ ویئر ٹولز کے ساتھ مشقیں ہو سکتی ہیں۔ یہ ابھی سیکھے گئے علم کو کسی اور شعبے میں لاگو کرنے اور آپ کی سمجھ کو گہرا کرنے میں مدد کرنے کے لیے بنائے گئے ہیں۔

پروگرامنگ کے منصوبے طویل، زیادہ چیلنجنگ پروگرامنگ مشقیں ہیں۔ ہم مشکل کی مختلف سطحوں کے پروجیکٹس فراہم کرتے ہیں۔ ان منصوبوں کو کلاس روم کی ترتیبات میں ہوم ورک اسائنمنٹ کے طور پر استعمال کیا جا سکتا ہے۔ پروگرامنگ پروجیکٹس کے نمونے حل پبلشر اور ویب سائٹ <https://datastructures.live> سے اہل اساتذہ کے لیے دستیاب ہیں۔

تاریخ مچل ویٹ اور رابرٹ لافور نے اس کتاب کا پہلا ورژن تیار کیا اور اسے جاوا میں ڈیٹا سٹرکچرز اور الگورتھم کا عنوان دیا۔ پہلا ایڈیشن 1998 میں شائع ہوا تھا، اور دوسرا ایڈیشن، رابرٹ کا، 2002 میں سامنے آیا تھا۔ جان کیننگ اور ایلن بروڈر نے اس ورژن کو ایجوکیشن اور تجارتی اور غیر تجارتی سوفٹ ویئر کی ترقی میں مقبولیت کی وجہ سے پائنتھون کا استعمال کرتے ہوئے تیار کیا۔ جاوا بڑے پیمانے پر استعمال کیا جاتا ہے اور کمپیوٹر سائنس دانوں کے لیے جاننے کے لیے ایک اہم زبان ہے۔ بہت سے اسکولوں نے Python کو پہلی پروگرامنگ لینگویج کے طور پر اپنانے کے ساتھ، نصابی کتب کی ضرورت جو پہلے سے مانوس زبان میں نئے تصورات متعارف کراتے ہیں، اس کتاب کی ترقی کا باعث بنے۔ ہم نے ڈیٹا ڈھانچے کی کوریج کو بڑھایا اور بہت سی مثالوں کو اپ ڈیٹ کیا۔

ہم نے سیکھنے کے عمل کو ہر ممکن حد تک تکلیف دہ بنانے کی کوشش کی ہے۔ ہم امید کرتے ہیں کہ یہ متن بنیادی، اور واضح طور پر، کمپیوٹر سائنس کی خوبصورتی کو سب کے لیے قابل رسائی بنائے گا۔ کھڑے ہونے کے علاوہ، ہم امید کرتے ہیں کہ آپ کو ان خیالات کو سیکھنے میں مزہ آئے گا۔ اپنے آپ سے لطف اندوز!

فگر کریڈٹس

اعداد و شمار

کریڈٹ/انتساب

اعداد و شمار، 2.1-2.4، 2.6، 3.5، 3.7، 3.9، 4.2، 4.6، 4.10، 4.11، 5.3، 5.4،

شکمنت سافت ویئر

5.9، 6.10، 6.18، 7.3، 7.4، 7.6، 7.10، 7.11، 8.5، 8.8، 8.9، 8.11،

8.14، 8.15، 9.7-9.9، 10.09-10.11(ac)، 10.14-10.16، 10.29(ab)،

11.06-11.10، 11.14، 11.15، 11.18، 11.19، 12.24، 13.7، 13.8،

13.9، 13.13، 14.6، 14.12، 14.17، 15.02، 15.9، 15.14-15.18،

A.01-A.04

شکل 6.1

بشکریہ Droste BV

شکل 13.15

www.wordclouds.com کے ذریعہ تیار کردہ لفظ کلاؤڈ۔

ڈھانپنا

آرکی / 13 شٹر اسٹاک

اس باب میں

باب 8

ہائری درخت

ہائری درخت کیوں استعمال کریں؟

ہائری درخت کی اصطلاحات

ہائری درخت کی تشبیہ

ہائری درخت کی تلاش کے درخت کیسے کام کرتے ہیں؟

ہائری درخت کی تلاش کرنا

ہائری درخت کی تلاش کرنا

ہائری درخت کو عبور کرنا

ہائری درخت سے کم تلاش کرنا اور زیادہ سے زیادہ کلیدی اقدار

ہائری درخت کو حذف کرنا

ہائری درخت کی کارکردگی

ہائری درخت کو صفوں کے طور پر دکھایا گیا ہے۔

ہائری درخت کی پریشانی

ہائری درخت کی چابیاں

BinarySearchTreeTester پر پروگرام

ہائری درخت میں کوڈ

اس باب کو ہم الگورتھم سے تبدیل کرتے ہیں، باب 7 کا فوکس، "اعلیٰ درجے کی ترتیب"، ڈیٹا ڈھانچے پر۔ ہائری درخت پروگرامنگ میں استعمال ہونے والے بنیادی ڈیٹا اسٹوریج ڈھانچے میں سے ایک ہیں۔ وہ ایسے فوائد فراہم کرتے ہیں جو آپ نے اب تک دیکھے ہوئے ڈیٹا ڈھانچے نہیں کر سکتے۔ اس باب میں آپ سیکھتے ہیں کہ آپ درختوں کو کیوں استعمال کرنا چاہیں گے، وہ کیسے کام کرتے ہیں، اور انہیں کیسے بنانا ہے۔

ہائری درخت کیوں استعمال کریں؟

آپ درخت کیوں استعمال کرنا چاہتے ہیں؟ عام طور پر، کیونکہ یہ com دو دیگر ڈھانچے کے فوائد کو جوڑتا ہے: ایک ترتیب شدہ صف اور ایک منسلک فہرست۔ آپ ایک درخت کو تیزی سے تلاش کر سکتے ہیں، جیسا کہ آپ آرڈر کی گئی صف کر سکتے ہیں، اور آپ آٹمز کو جلدی سے داخل اور حذف بھی کر سکتے ہیں، جیسا کہ آپ لنک کردہ فہرست کے ساتھ کر سکتے ہیں۔ آئیے درختوں کی تفصیلات میں جانے سے پہلے ان موضوعات کو تھوڑا سا دریافت کرتے ہیں۔

ایک ترتیب شدہ صف میں آہستہ اندراج ایک ایسی صف کا تصور کریں جس میں تمام عناصر کو ترتیب سے ترتیب دیا گیا ہو — یعنی ایک ترتیب شدہ صف — جیسا کہ آپ نے باب 2 میں دیکھا تھا۔ جیسا کہ آپ نے سیکھا، آپ ہائری تلاش کا استعمال کرتے ہوئے کسی خاص قدر کے لیے اس طرح کی صف کو تیزی سے تلاش کر سکتے ہیں۔

آپ صف کے مرکز میں چیک کریں؛ اگر آپ جس چیز کی تلاش کر رہے ہیں وہ اس سے زیادہ ہے جو آپ کو وہاں ملتی ہے، تو آپ اپنی تلاش کو صف کے اوپری نصف تک محدود کر دیتے ہیں۔ اگر یہ کم ہے تو، آپ اپنی تلاش کو نیچے کے نصف تک محدود کر دیتے ہیں۔ اس عمل کو بار بار لگانے سے $O(\log N)$ وقت میں آجیکٹ مل جاتا ہے۔ آپ ترتیب شدہ ترتیب میں ہر آجیکٹ کا دورہ کرتے ہوئے، ترتیب دی گئی صف کو بھی تیزی سے عبور کر سکتے ہیں۔

دوسری طرف، اگر آپ کسی نئے آجیکٹ کو ترتیب شدہ صف میں داخل کرنا چاہتے ہیں، تو آپ کو پہلے یہ معلوم کرنا ہوگا کہ آجیکٹ کہاں جائے گا اور پھر اس کے لیے جگہ بنانے کے لیے تمام اشیاء کو زیادہ کلیدوں کے ساتھ صف میں ایک جگہ پر منتقل کریں۔ یہ متعدد

چالیں وقت طلب ہیں، اوسطاً، نصف اشیاء کو منتقل کرنے کی ضرورت ہوتی ہے ($N/2$ حرکتیں)۔
حذف کرنے میں ایک ہی متعدد حرکتیں شامل ہیں اور اس طرح اتنی ہی سست ہے۔
اگر آپ بہت زیادہ اندراجات اور حذف کرنے جا رہے ہیں، تو آرڈر شدہ صف ایک برا انتخاب ہے۔

لنکڈ لسٹ میں سست تلاش جیسا کہ آپ نے باب 5 میں دیکھا، "لنکڈ لسٹ"، آپ لنکڈ لسٹ پر تیزی سے اندراج اور حذف کر سکتے ہیں۔
آپ صرف چند حوالوں کو تبدیل کر کے ان کارروائیوں کو پورا کر سکتے ہیں۔

ان دو آپریشنز کے لیے ($O(1)$ وقت (تیز ترین بگ O ٹائم) درکار ہوتا ہے۔

بدقسمتی سے، منسلک فہرست میں ایک مخصوص عنصر تلاش کرنا اتنا تیز نہیں ہے۔ آپ کو فہرست کے آغاز سے شروع کرنا چاہیے اور ہر ایک عنصر کو اس وقت تک ملاحظہ کرنا چاہیے جب تک کہ آپ کو وہ چیز نہ مل جائے جس کی آپ تلاش کر رہے ہیں۔ اس طرح، آپ کو اوسطاً $N/2$ اشیاء کا دورہ کرنے کی ضرورت ہے، ہر ایک کی کلید کا مطلوبہ قدر کے ساتھ موازنہ کرنا۔ یہ عمل سست ہے، $O(N)$ وقت درکار ہے۔ (نوٹ کریں کہ ایک ترتیب کے لیے تیز سمجھے جانے والے اوقات داخل کرنے، حذف کرنے اور تلاش کے بنیادی ڈیٹا ڈھانچے کے کاموں کے لیے سست ہیں۔)

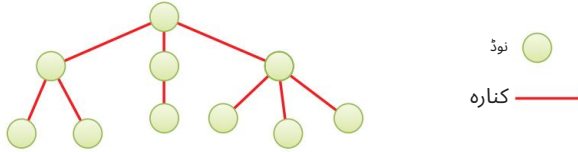
آپ سوچ سکتے ہیں کہ آپ آرڈر شدہ لنکڈ لسٹ کا استعمال کر کے چیزوں کو تیز کر سکتے ہیں، جس میں عناصر کو ترتیب سے ترتیب دیا گیا ہے، لیکن اس سے کوئی فائدہ نہیں ہوتا ہے۔ آپ کو اب بھی شروع سے شروع کرنا چاہیے اور عناصر کو ترتیب سے دیکھنا چاہیے کیونکہ کسی بھی عنصر کے حوالہ جات کی پیروی کے بغیر اس تک رسائی کا کوئی طریقہ نہیں ہے۔ آپ ترتیب شدہ ترتیب میں ایک خلا کو تلاش کرنے کے بعد عنصر کی تلاش کو ترک کر سکتے ہیں جہاں اسے ہونا چاہیے تھا، لہذا اس سے گمشدہ اشیاء کی شناخت میں تھوڑا وقت بچ جائے گا۔ آرڈر شدہ فہرست کا استعمال صرف نوڈس کو تیز تر ترتیب دینے میں مدد کرتا ہے اور کسی صوابدیدی چیز کو تلاش کرنے میں مدد نہیں کرتا ہے۔

بچاؤ کے لیے درخت

یہ اچھا ہو گا کہ اگر کسی لنک شدہ فہرست کے فوری اندراج اور حذف کرنے کے ساتھ ایک ترتیب شدہ صف کی فوری تلاش کے ساتھ ڈیٹا کا ڈھانچہ موجود ہو۔ درخت ان دونوں خصوصیات کو فراہم کرتے ہیں اور ڈیٹا کے سب سے دلچسپ ڈھانچے میں سے ایک ہیں۔

درخت کیا ہے؟

ایک درخت کناروں سے جڑے ہوئے نوڈس پر مشتمل ہوتا ہے۔ شکل 1-8 ایک درخت کو دکھاتی ہے۔ درخت کی ایسی تصویر میں نوڈس کو دائروں کے طور پر اور کناروں کو دائروں کو جوڑنے والی لکیریوں کے طور پر دکھایا جاتا ہے۔



شکل 1-8 ایک عام (غیر بائنری) درخت

درختوں کا بڑے پیمانے پر تجریدی ریاضیاتی اداروں کے طور پر مطالعہ کیا گیا ہے، لہذا ان کے بارے میں نظریاتی علم کی ایک بڑی مقدار موجود ہے۔ ایک درخت درحقیقت ایک زیادہ عمومی زمرہ کی مثال ہے جسے گراف کہتے ہیں۔ نوڈس کو جوڑنے والے کناروں کی اقسام اور ترتیب درختوں اور گرافوں میں فرق کرتے ہیں، لیکن آپ کو گرافس کے اضافی مسائل کے بارے میں فکر کرنے کی ضرورت نہیں ہے۔ ہم باب 14 "گرافس"، اور باب 15 "وزن والے گرافس" میں گرافس پر بحث کرتے ہیں۔

کمپیوٹر پروگراموں میں، نوڈس اکثر اداروں کی نمائندگی کرتے ہیں جیسے فائل فولڈرز، فائلز، ڈیپارٹمنٹس، لوگ وغیرہ۔ دوسرے لفظوں میں، کسی بھی قسم کے ڈیٹا ڈھانچے میں محفوظ کردہ عام ریکارڈ اور آئٹمز۔ آجیکٹ اور اینڈ پروگرامنگ لینگویج میں، نوڈس ایسی اشیاء ہوتی ہیں جو کبھی کبھی حقیقی دنیا میں ہستیوں کی نمائندگی کرتی ہیں۔

نوڈس کے درمیان لکیریں (کنارے) نوڈس کے متعلق ہونے کے طریقے کی نمائندگی کرتی ہیں۔ موٹے طور پر، لکیریں سہولت کی نمائندگی کرتی ہیں: ایک پروگرام کے لیے ایک نوڈ سے دوسرے نوڈ تک جانا آسان (اور تیز) ہے اگر کوئی لائن ان کو جوڑتی ہے۔ درحقیقت، نوڈ سے نوڈ تک جانے کا واحد راستہ لائنوں کے ساتھ ایک راستے پر چلنا ہے۔ یہ بنیادی طور پر وہی ہیں جو آپ نے منسلک فہرستوں میں دیکھے تھے۔ ہر نوڈ میں دوسرے نوڈس کے کچھ حوالہ جات ہو سکتے ہیں۔ الگورتھم کناروں کے ساتھ ایک سمت جانے تک محدود ہیں: نوڈ سے کسی دوسرے نوڈ کے حوالے سے۔ دوبرے منسلک نوڈس کا استعمال بعض اوقات دونوں سمتوں میں جانے کے لیے کیا جاتا ہے۔

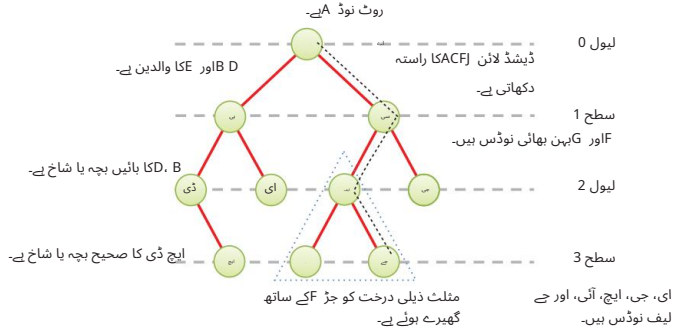
عام طور پر، ایک نوڈ کو درخت کی جڑ کے طور پر نامزد کیا جاتا ہے۔ بالکل لنکڈ لسٹ کے بیڈ کی طرح، باقی تمام نوڈس جڑ سے کناروں کی پیروی کرتے ہوئے پہنچ جاتے ہیں۔ روٹ نوڈ ڈائیکرام کے اوپری حصے میں ٹائپی کالی تیار کیا گیا ہے، جیسا کہ شکل 1-8 میں ہے۔ دوسرے نوڈس اس کے نیچے دکھائے گئے ہیں، اور خاکہ میں جتنا نیچے، دوسرے نوڈ تک جانے کے لیے اتنے ہی کناروں کو فالو کرنے کی ضرورت ہے۔ اس طرح، درختوں کے خاکے اوپر سے چھوٹے اور نیچے بڑے ہوتے ہیں۔ یہ ترتیب حقیقی درختوں کے مقابلے میں الٹا لگ سکتی ہے، کم از کم زمین کے اوپر حقیقی درختوں کے حصوں کے مقابلے میں۔ خاکے بصری معنوں میں درختوں کے جڑ کے نظام کی طرح ہیں۔ یہ ترتیب انہیں چارٹس کی طرح بناتا ہے جو خاندانی درختوں کو اوپر والے آباؤ اجداد کے ساتھ اور نیچے کی اولاد کو دکھانے کے لیے استعمال کیا جاتا ہے۔ عام طور پر، پروگرام درخت کے چھوٹے حصے، جڑ سے آپریشن شروع کرتے ہیں، اور کناروں کو وسیع تر کنارے تک لے جاتے ہیں۔ اوپر سے نیچے تک جانے کے بارے میں سوچنا زیادہ فطری ہے، جیسا کہ متن کو پڑھتے ہیں، اس لیے جڑ کے نیچے دوسرے نوڈس رکھنے سے نوڈس کی نسبتی ترتیب کو ظاہر کرنے میں مدد ملتی ہے۔

درختوں کی مختلف قسمیں ہیں، کناروں کی تعداد اور قسم سے ممتاز ہیں۔ شکل 1-8 میں دکھائے گئے درخت میں فی نوڈ دو سے زیادہ بچے ہیں۔ (ہم ایک لمحے میں بتاتے ہیں کہ "بچوں" کا کیا مطلب ہے۔) اس باب میں ہم درخت کی ایک مخصوص شکل پر بحث کرتے ہیں جسے بائنری ٹری کہتے ہیں۔ بائنری درخت میں ہر نوڈ میں زیادہ سے زیادہ دو بچے ہوتے ہیں۔ زیادہ عام درخت، جن میں نوڈس میں دو سے زیادہ بچے ہو سکتے ہیں، ملٹی وے ٹری کہلاتے ہیں۔ ہم باب 3-4، 9، درخت اور بیرونی ذخیرہ" میں ملٹی وے درختوں کی مثالیں دکھاتے ہیں۔

درخت کی اصطلاحات

درختوں کے مخصوص پہلوؤں کو بیان کرنے کے لیے بہت سی اصطلاحات استعمال کی جاتی ہیں۔ آپ کو ان کو جاننے کی ضرورت ہے تاکہ یہ بحث قابل فہم ہو۔ خوش قسمتی سے، ان میں سے زیادہ تر اصطلاحات حقیقی دنیا کے درختوں یا خاندانی تعلقات سے متعلق ہیں، اس لیے انہیں یاد رکھنا مشکل نہیں ہے۔ شکل 2-18 میں سے بہت سی اصطلاحات کو دکھاتا ہے جو بائنری درخت پر لاگو ہوتے ہیں۔

338 باب 8 بائنری درخت

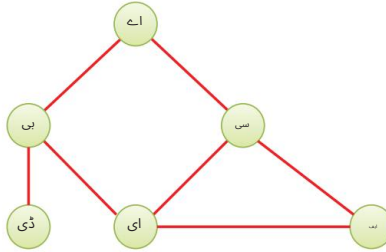


شکل 2-8 درخت کی اصطلاحات

جڑ درخت کے اوپری حصے میں موجود نوڈ کو جڑ کہا جاتا ہے۔ ایک درخت میں صرف ایک جڑ ہے، جس پر تصویر میں A کا لیبل لگا ہوا ہے۔

راستہ کسی ایسے شخص کے بارے میں سوچیں جو ان کو جوڑنے ہوئے کناروں کے ساتھ نوڈ سے نوڈ تک چلتے ہیں۔ نوڈس کے نتیجے میں ترتیب کو راستہ کہا جاتا ہے۔ نوڈس اور کناروں کے مجموعے کو درخت کے طور پر بیان کرنے کے لیے، جڑ سے کسی دوسرے نوڈ تک ایک (اور صرف ایک!) راستہ ہونا چاہیے۔

شکل 3-8 ایک غیر درخت کو ظاہر کرتی ہے۔ آپ دیکھ سکتے ہیں کہ یہ اس اصول کی خلاف ورزی کرتا ہے کیونکہ A سے نوڈس E اور F تک متعدد راستے ہیں۔ یہ گراف کی ایک مثال ہے جو درخت نہیں ہے۔



تصویر 3-8 ایک غیر درخت

والدین

کسی بھی نوڈ (جڑ کے علاوہ) کا بالکل ایک کنارہ ہوتا ہے جو اسے اوپر والے نوڈ سے جوڑتا ہے۔ اس کے اوپر والے نوڈ کو نوڈ کا پیرنٹ کہا جاتا ہے۔ روٹ نوڈ کا پیرنٹ نہیں ہونا چاہیے۔

بچہ

کسی بھی نوڈ میں ایک یا زیادہ کنارے ہو سکتے ہیں جو اسے نیچے نوڈس سے جوڑتے ہیں۔ دیئے گئے نوڈ کے نیچے ان نوڈس کو اس کے بچے، یا بعض اوقات شاخیں کہتے ہیں۔

Sibling روٹ نوڈ کے علاوہ کسی بھی نوڈ میں بہن بھائی نوڈس ہو سکتے ہیں۔ ان نوڈس میں مشترکہ پیرنٹ نوڈ ہوتا ہے۔

لیف ایک نوڈ جس کی کوئی اولاد نہیں ہوتی اسے لیف نوڈ یا محض ایک پتی کہا جاتا ہے۔ ایک درخت میں صرف ایک جڑ ہوسکتی ہے، لیکن بہت سے پتے ہوسکتے ہیں۔ اس کے برعکس، ایک نوڈ جس میں بچے ہوتے ہیں ایک اندرونی نوڈ ہوتا ہے۔

سب ٹری کسی بھی نوڈ (جڑ کے علاوہ) کو ذیلی درخت کی جڑ سمجھا جا سکتا ہے، جس میں اس کے بچے، اور اس کے بچوں کے بچے وغیرہ شامل ہیں۔ اگر آپ خاندانی جھوٹ کے لحاظ سے سوچتے ہیں تو، نوڈ کے ذیلی درخت میں اس کی تمام اولادیں ہوتی ہیں۔

نوڈ کا دورہ اس وقت کیا جاتا ہے جب پروگرام کنٹرول نوڈ پر آتا ہے، عام طور پر نوڈ پر کچھ آپریشن کرنے کے مقصد کے لیے، جیسے کہ اس کے ڈیٹا فیلڈ میں سے کسی کی قدر کی جانچ کرنا یا اسے ڈسپلے کرنا۔ ایک نوڈ سے دوسرے نوڈ کے راستے پر محض ایک نوڈ کے اوپر سے گزرنا نوڈ پر جانا نہیں سمجھا جاتا ہے۔

ایک درخت کو عبور کرنے کا مطلب ہے تمام نوڈس کو کچھ مخصوص ترتیب میں جانا۔ مثال کے طور پر، آپ چڑھتے ہوئے کلیدی قدر کی ترتیب میں تمام نوڈس ملاحظہ کر سکتے ہیں۔ درخت کو عبور کرنے کے اور بھی طریقے ہیں، جیسا کہ ہم بعد میں بیان کریں گے۔

سطحیں کسی خاص نوڈ کی سطح سے مراد یہ ہے کہ نوڈ جڑ سے کتنی نسلوں پر مشتمل ہے۔

اگر آپ فرض کریں کہ جڑ لیول 0 ہے، تو اس کے بچے لیول 1 پر ہیں، اس کے پوتے پوتیاں لیول 2 پر ہیں، وغیرہ۔ اسے بعض اوقات نوڈ کی گہرائی بھی کہا جاتا ہے۔

کیز آپ نے دیکھا ہے کہ کسی چیز میں ایک ڈیٹا فیلڈ کو عام طور پر کلیدی قدر کے طور پر نامزد کیا جاتا ہے، یا ایک کلید کو سم پلائی کیا جاتا ہے۔ یہ قدر آئٹم کو تلاش کرنے یا اس پر دیگر کارروائیاں کرنے کے لیے استعمال ہوتی ہے۔ درختوں کے خاکوں میں، جب ایک دائرہ ڈیٹا آئٹم کے حامل نوڈ کی نمائندگی کرتا ہے، تو آئٹم کی کلیدی قدر عام طور پر دائرے میں دکھائی جاتی ہے۔

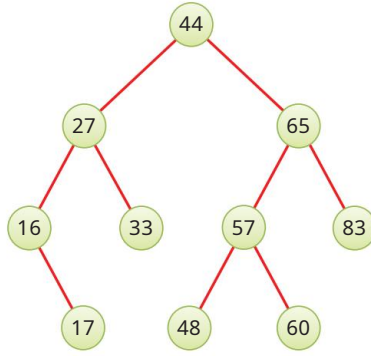
Binary Trees اگر درخت کے ہر نوڈ میں زیادہ سے زیادہ دو بچے ہوں تو درخت کو بائنری ٹری کہا جاتا ہے۔ اس باب میں ہم بائنری درختوں پر توجہ مرکوز کرتے ہیں کیونکہ وہ سب سے آسان اور عام ہیں۔

بائنری ٹری میں ہر نوڈ کے دو بچوں کو لیفٹ چائلڈ اور رائٹ چائلڈ کہا جاتا ہے، ان کی پوزیشنوں کے مطابق جب آپ درخت کی تصویر کھینچتے ہیں، جیسا کہ شکل 2-8 میں دکھایا گیا ہے۔ بائنری ٹری میں ایک نوڈ ضروری نہیں کہ زیادہ سے زیادہ دو ہوں۔

بچے! اس کا صرف بائیں بچہ ہو سکتا ہے یا صرف دائیں بچہ ہو سکتا ہے، یا اس کا کوئی بچہ نہیں ہو سکتا (اس صورت میں یہ ایک پتی ہے)۔

بائنری سرچ ٹری جس قسم کے بائنری ٹری پر ہم اس باب کے شروع میں بحث کر رہے ہیں اسے تکنیکی طور پر بائنری سرچ ٹری کہا جاتا ہے۔ نوڈس کی چابیاں تلاش کے درختوں میں ایک خاص ترتیب رکھتی ہیں۔

شکل 4-8 ایک بائنری تلاش کا درخت دکھاتا ہے۔



تصویر 4-8 ایک بائنری تلاش کا درخت

نوٹ

بائنری سرچ ٹری کی وضاحتی خصوصیت یہ ہے: نوڈ کے بائیں بچے کی کلید اس کے والدین کی کلید سے کم ہونی چاہیے، اور نوڈ کے دائیں بچے کے پاس اس کے والدین کی کلید سے زیادہ یا اس کے برابر کلید ہونی چاہیے۔

ایک تشبیہ

ایک عام درخت کا سامنا ڈیسک ٹاپ کمپیوٹرز پر درجہ بندی کا فائل سسٹم ہے۔

یہ سسٹم بیسویں صدی میں کاروباری اداروں کے ذریعہ استعمال ہونے والی دستاویزی ذخیرہ کرنے کی مروجہ ٹیکنالوجی پر وضع کیا گیا تھا: فائلنگ کینٹ جس میں فولڈرز ہوتے ہیں جن میں ذیلی فولڈرز ہوتے ہیں، انفرادی دستاویزات تک۔ کمپیوٹر آپریٹنگ سسٹم فائلوں کو درجہ بندی میں محفوظ کر کے اس کی نقل کرتے ہیں۔ درجہ بندی کے اوپری حصے میں روٹ ڈائریکٹری ہے۔ اس ڈائریکٹری میں "فولڈرز" شامل ہیں جو کہ ذیلی ڈائریکٹریز ہیں، اور فائلیں، جو کاغذی دستاویزات کی طرح ہیں۔ ہر ذیلی ڈائریکٹری میں اس کی اپنی اور مزید فائلوں کی سب ڈائریکٹریاں ہوسکتی ہیں۔ ان سب کی درخت میں مشابہتیں ہیں: روٹ ڈائریکٹری روٹ نوڈ ہے، سب ڈائریکٹریاں بچوں کے ساتھ نوڈز ہیں، اور فائلیں لیف نوڈ ہیں۔

فائل سسٹم میں کسی خاص فائل کی وضاحت کرنے کے لیے، آپ روٹ ڈائریکٹری سے فائل تک مکمل راستہ استعمال کرتے ہیں۔ یہ درخت کے نوڈ کے راستے کے برابر ہے۔ یونیکارم ریسورس لوکاٹورز (URLs) انٹرنیٹ پر کسی وسائل کا راستہ دکھانے کے لیے اسی طرح کی تعمیر کا استعمال کرتے ہیں۔ فائل سسٹم کے ہاتھ نام اور یو آر ایل دونوں ذیلی ڈائریکٹریز کی کئی سطحوں کی اجازت دیتے ہیں۔ فائل سسٹم ہاتھ میں آخری نام یا تو سب ڈائریکٹری یا فائل ہے۔ فائلیں پتوں کی نمائندگی کرتی ہیں۔ ان کی اپنی کوئی اولاد نہیں ہے۔

واضح طور پر، ایک درجہ بندی کا فائل سسٹم بائنری ٹری نہیں ہے کیونکہ ایک ڈائریکٹری میں بہت سے بچے ہوسکتے ہیں۔ ایک درجہ بندی کا فائل سسٹم ان درختوں سے ایک اور اہم طریقے سے مختلف ہے جن پر ہم یہاں بحث کرتے ہیں۔ فائل سسٹم میں، سب ڈائریکٹریاں ان کے نام جیسی صفات کے علاوہ کوئی ڈیٹا نہیں رکھتی ہیں۔ ان میں صرف دیگر ذیلی ڈائریکٹریوں یا فائلوں کے حوالے ہوتے ہیں۔ صرف فائلوں میں ڈیٹا ہوتا ہے۔

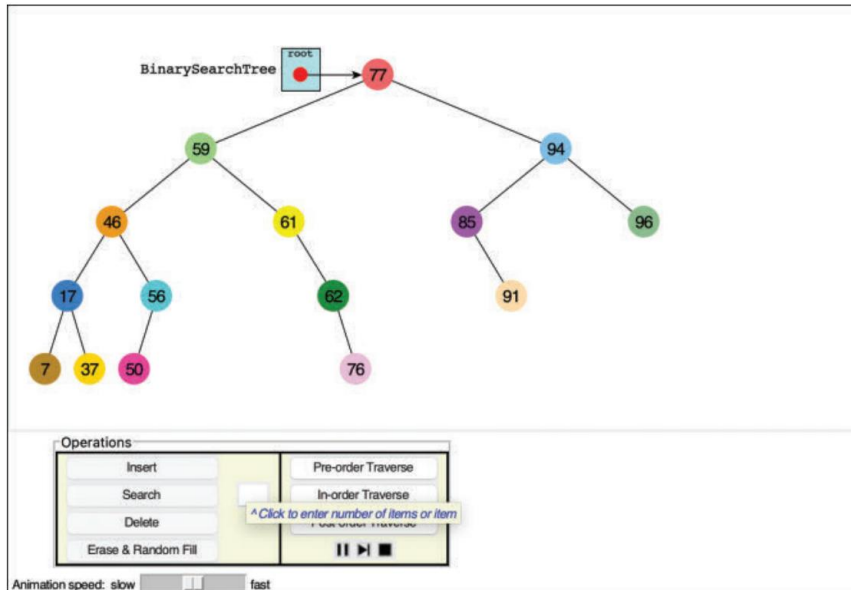
ایک درخت میں، ہر نوڈ میں ڈیٹا ہوتا ہے۔ ڈیٹا کی صحیح قسم کا انحصار اس بات پر ہوتا ہے کہ کیا بھیجا جا رہا ہے: اہلکاروں کے بارے میں ریکارڈ، گاڑی کی تعمیر میں استعمال ہونے والے اجزاء کے بارے میں ریکارڈ، وغیرہ۔ اعداد و شمار کے علاوہ، پتوں کے علاوہ تمام نوڈس دیگر نوڈس کے حوالے پر مشتمل ہوتے ہیں۔

درجہ بندی کے فائل سسٹم دوسرے پہلوؤں میں بھی بائنری سرچ ٹری سے مختلف ہیں۔ فائل سسٹم کا مقصد فائلوں کو منظم کرنا ہے۔ بائنری سرچ ٹری کا مقصد زیادہ عمومی اور خلاصہ ہے۔ یہ ایک ڈیٹا ڈھانچہ ہے جو آئٹمز کے مجموعے پر داخل کرنے، حذف کرنے، تلاش کرنے، اور ٹراورسل کی عام کارروائیاں فراہم کرتا ہے، ان کو اپنی کلیدوں کے ذریعے منظم کر کے کارروائیوں کو تیز کرتا ہے۔ دونوں کے درمیان مشابہت کا مقصد ایک اور مانوس نظام کو ظاہر کرنا ہے جو کچھ اہم خصوصیات کا اشتراک کرتا ہے، لیکن تمام نہیں۔

بائنری تلاش کے درخت کیسے کام کرتے ہیں؟

آئیے دیکھتے ہیں کہ دی گئی کلید کے ساتھ نوڈ تلاش کرنے، نیا نوڈ ڈالنے، درخت کو عبور کرنے اور نوڈ کو حذف کرنے کے عام بائنری ٹری آپریشنز کو کیسے انجام دیا جائے۔ ان میں سے ہر ایک آپریشن کے لیے، ہم سب سے پہلے یہ دکھاتے ہیں کہ اسے انجام دینے کے لیے بائنری سرچ ٹری ویژولائزیشن ٹول کا استعمال کیسے کیا جائے، پھر ہم متعلقہ Python کوڈ کو دیکھتے ہیں۔

بائنری سرچ ٹری ویژولائزیشن ٹول اس مثال کے لیے، بائنری سرچ ٹری ویژولائزیشن ٹول شروع کریں (پروگرام کو BinaryTree.py کہا جاتا ہے)۔ آپ کو ایک ایسی سکرین نظر آتی چاہیے جو کہ شکل 8-5 میں دکھائی گئی ہے۔



تصویر 8-5 بائنری سرچ ٹری ویژولائزیشن ٹول

ویژولائزیشن ٹول کا استعمال نوڈس میں دکھائی جانے والی کلیدی اقدار 0 سے 99 تک ہوتی ہیں۔ بلاشبہ، ایک حقیقی درخت میں، کلیدی اقدار کی ایک بڑی رینج ہو سکتی ہے۔ مثال کے طور پر، اگر ٹیلی فون نمبرز کو کلیدی اقدار کے لیے استعمال کیا جاتا ہے، تو وہ 999,999,999,999,999,999 تک ہو سکتے ہیں (15 ہندسے بشمول بین الاقوامی ٹیلی کمیونیکیشن یونین کے معیار میں ملک کے کوڈز)۔ ہم ممکنہ چابیاں کے ایک آسان سیٹ پر توجہ مرکوز کرتے ہیں۔

ویژولائزیشن ٹول اور حقیقی درخت کے درمیان ایک اور فرق یہ ہے کہ ٹول اپنے درخت کو پانچ کی گہرائی تک محدود کرتا ہے۔ یعنی، جڑ سے نیچے تک پانچ سے زیادہ سطحیں نہیں ہو سکتیں (لیول 0 سے لیول 4)۔ (یہ پابندی یقینی بناتی ہے کہ درخت میں موجود تمام نوڈس اسکرین پر نظر آئیں گے۔ ایک حقیقی درخت میں سطحوں کی تعداد لامحدود ہوتی ہے (جب تک کہ کام پیوٹر میموری ختم نہ ہو جائے)۔

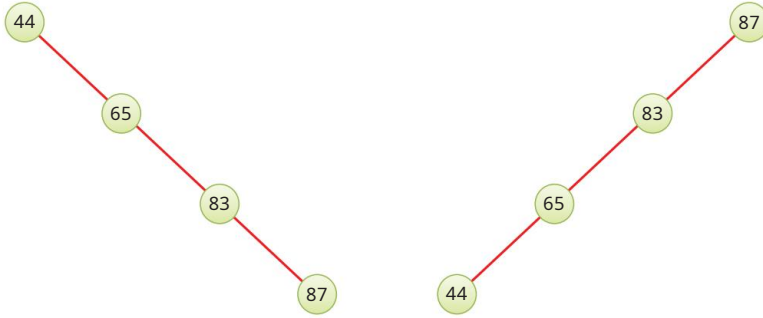
ویژولائزیشن ٹول کا استعمال کرتے ہوئے، آپ جب چاہیں ایک نیا درخت بنا سکتے ہیں۔ ایسا کرنے کے لیے، متعدد آئٹمز درج کریں اور Fill Ease & Random بٹن پر کلک کریں۔ آپ 0 سے 99 آئٹمز بھرنے کے لیے کہہ سکتے ہیں۔ اگر آپ 0 کا انتخاب کرتے ہیں، تو آپ ایک خالی درخت بنائیں گے۔ بڑی تعداد کے استعمال سے مزید نوڈس بھر جائیں گے، لیکن کچھ درخواست کردہ نوڈس ظاہر نہیں ہو سکتے ہیں۔ یہ درخت کی گہرائی کی حد اور آئٹمز کو داخل کیے جانے کے لیے ترتیب ترتیب کی وجہ سے ہے۔ آپ بے ترتیب ترتیب سے نکلنے والے درختوں کی مختلف قسموں کو دیکھنے کے لیے نوڈس کی مختلف تعداد کے ساتھ درخت بنا کر تجربہ کر سکتے ہیں۔

نوڈس مختلف رنگوں کے ساتھ بنائے گئے ہیں۔ رنگ کلید کے ساتھ ذخیرہ شدہ ڈیٹا کی نمائندگی کرتا ہے۔ ہم تھوڑی دیر بعد دکھاتے ہیں کہ کچھ آپریشنز میں اس ڈیٹا کو کیسے اپ ڈیٹ کیا جاتا ہے۔

درختوں کی تعمیر جیسا کہ ویژولائزیشن ٹول میں دکھایا گیا ہے، درخت کی شکل دونوں چیزوں پر منحصر ہوتی ہے جو اس میں شامل ہیں اور ساتھ ہی ان اشیاء کو درخت میں کس ترتیب سے داخل کیا جاتا ہے۔ یہ پہلے تو عجیب لگ سکتا ہے۔ اگر آئٹمز کو ترتیب شدہ صف میں داخل کیا جاتا ہے، تو وہ ہمیشہ ایک ہی ترتیب میں ختم ہوتے ہیں، قطع نظر ان کی ترتیب سے۔ ہائری تلاش کے درخت مختلف کیوں ہیں؟

ہائری سرچ ٹری کی ایک اہم خصوصیت یہ ہے کہ اس میں آئٹمز کو داخل ہونے کے ساتھ ہی انہیں مکمل طور پر آرڈر کرنے کی ضرورت نہیں ہے۔ جب یہ کسی موجودہ درخت میں کوئی نئی چیز شامل کرتا ہے، تو یہ فیصلہ کرتا ہے کہ نئے لیف نوڈ کو کہاں رکھنا ہے اس کی کلید کا درخت میں پہلے سے ذخیرہ شدہ نوڈس سے موازنہ کر کے۔ یہ جڑ سے ایک گمشدہ بجے تک جانے والے راستے کی پیروی کرتا ہے جہاں نیا نوڈ "تعلق رکھتا ہے۔" ہائری چائلڈ کو منتخب کرنے سے جب نئے نوڈ کی کلید اندرونی نوڈ کی کلید سے کم ہو اور دوسری اقدار کے لیے صحیح چائلڈ، ہمیشہ نئے نوڈ کے لیے ایک منفرد راستہ ہوگا۔ اس منفرد راستے کا مطلب ہے کہ آپ اس نوڈ کو بعد میں اس کی کلید کے ذریعے آسانی سے تلاش کر سکتے ہیں، لیکن اس کا مطلب یہ بھی ہے کہ پہلے داخل کردہ آئٹمز کسی بھی نئی آئٹم کے راستے کو متاثر کرتی ہیں۔

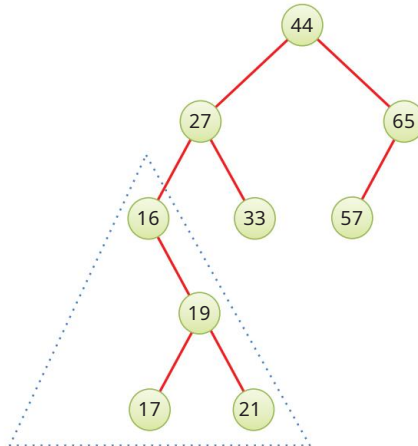
مثال کے طور پر، اگر آپ خالی ہائری سرچ ٹری سے شروع کرتے ہیں اور کلیدی ترتیب میں اضافہ کرتے ہوئے نوڈس داخل کرتے ہیں، تو ہر ایک کے لیے منفرد راستہ ہمیشہ صحیح ترین راستہ ہوگا۔ ہر اندراج نیچے دائیں جانب ایک اور نوڈ کا اضافہ کرتا ہے۔ اگر آپ نوڈس کی ترتیب کو ریورس کرتے ہیں اور انہیں خالی درخت میں داخل کرتے ہیں، تو ہر اندراج نیچے بائیں طرف نوڈ کو جوڑ دے گا کیونکہ کلید اب تک درخت میں کسی دوسرے سے کم ہے۔ شکل 6-8 دکھاتا ہے کہ اگر آپ 83، 65، 44 اور 87 کلیدوں کے ساتھ نوڈس کو آگے یا ریورس ترتیب میں داخل کرتے ہیں تو کیا ہوتا ہے۔



شکل 8-6 درخت ترتیب دے گئے نوڈس ڈال کر بنائے گئے ہیں۔

غیر متوازن درخت وہ درخت جو شکل 8-6 میں دکھائے گئے ہیں، درختوں کی طرح نظر نہیں آتے۔ درحقیقت، وہ لنکڈ فہرستوں کی طرح نظر آتے ہیں۔ ہائری سرچ ٹری کے اہداف میں سے ایک خاص نوڈ کی تلاش کو تیز کرنا ہے، لہذا نوڈ کو تلاش کرنے کے لیے منسلک فہرست کے ذریعے قدم اٹھانا کوئی بہتری نہیں ہوگی۔ درخت کا فائدہ حاصل کرنے کے لیے، جڑ کے دونوں طرف نوڈس کو تقریباً متوازن ہونا چاہیے۔ مثالی طور پر، نوڈ کو تلاش کرنے کے راستے پر ہر قدم کو تلاش کرنے کے لیے نوڈس کی تعداد کو آدھے حصے میں کاٹنا چاہیے، بالکل اسی طرح جیسے صفوں کی ہائری تلاشوں اور باب 2 میں بیان کردہ guess-a-number گیم میں۔

ہم کچھ درختوں کو غیر متوازن کہہ سکتے ہیں۔ یعنی، ان کے زیادہ تر نوڈس جڑ کے ایک طرف یا دوسری طرف ہوتے ہیں، جیسا کہ شکل 8-7 میں دکھایا گیا ہے۔ کوئی بھی ذیلی درخت بھی غیر متوازن ہو سکتا ہے جیسا کہ اعداد و شمار کے نچلے بائیں طرف بیان کردہ۔ بلاشبہ، صرف ایک مکمل متوازن درخت کے بائیں اور دائیں ذیلی درختوں پر مساوی تعداد میں نوڈس ہوں گے (اور مکمل طور پر متوازن ہونے کی وجہ سے، ہر نوڈ کا ذیلی درخت بھی مکمل طور پر متوازن ہوگا)۔ تصادفی طور پر منتخب کردہ آٹمز پر ایک وقت میں ایک نوڈس داخل کرنے کا مطلب ہے کہ زیادہ تر درخت ایک یا زیادہ نوڈس سے غیر متوازن ہوں گے کیونکہ وہ تعمیر ہوتے ہیں، لہذا آپ عام طور پر مکمل طور پر متوازن درخت تلاش کرنے کی توقع نہیں کر سکتے۔ مندرجہ ذیل ابواب میں، ہم ان میں توازن پیدا کرنے کے طریقوں کو زیادہ احتیاط سے دیکھتے ہیں کیونکہ نوڈس ڈالے اور حذف کیے جاتے ہیں۔



تصویر 8-7 ایک غیر متوازن درخت (ایک غیر متوازن ذیلی درخت کے ساتھ)

درخت اس ترتیب کی وجہ سے غیر متوازن ہو جائے ہیں جس میں ڈیٹا آئٹمز ڈالے جاتے ہیں۔ اگر ان کلیدی اقدار کو تصادفی طور پر داخل کیا جائے تو درخت کم و بیش متوازن ہو جائے گا۔ جب ایک صعودی ترتیب (جیسے 65، 42، 33، 18، 11 یا نزولی ترتیب کا سامنا ہوتا ہے، تو تمام اقدار دائیں بچے (اگر چڑھتے ہیں) یا بائیں بچے (اگر اتارنے ہیں) ہوں گے، اور درخت غیر متوازن ہوگا۔ ویژولائزیشن ٹول میں کلیدی قدریں تصادفی طور پر پیدا ہوتی ہیں، لیکن یقیناً کچھ مختصر صعودی یا نزولی ترتیب بہرحال تخلیق کی جائے گی، جو مقامی عدم توازن کا باعث بنے گی۔

اگر ایک درخت ڈیٹا آئٹمز کے ذریعہ بنایا گیا ہے جس کی کلیدی قدریں بے ترتیب ترتیب میں آتی ہیں، تو غیر متوازن درختوں کا مسئلہ بڑے درختوں کے لیے بہت زیادہ مسئلہ نہیں ہوسکتا ہے کیونکہ ایک ترتیب میں نمبروں کے طویل چلنے کے امکانات کم ہیں۔ بعض اوقات، تاہم، کلیدی قدریں سخت ترتیب میں آئیں گی۔ مثال کے طور پر، جب کوئی ڈیٹا انٹری کرنے والا ڈیٹا داخل کرنے سے پہلے فارموں کے اسٹیک کو حروف تہجی کے حساب سے ترتیب دیتا ہے۔ جب ایسا ہوتا ہے تو، درخت کی کارکردگی کو سنجیدگی سے گرایا جا سکتا ہے۔ ہم ابواب 9 اور 10 میں غیر متوازن درختوں اور ان کے بارے میں کیا کرنے کے بارے میں بات کرتے ہیں۔

پائتھون کوڈ میں درخت کی نمائندگی آئیے ازگر میں بائنری سرچ ٹری کو لاگو کرنا شروع کریں۔ دوسرے ڈیٹا ڈھانچے کی طرح، کمپیوٹر کی میموری میں درخت کی نمائندگی کرنے کے کئی طریقے ہیں۔ سب سے عام یہ ہے کہ نوڈس کو میموری میں (غیر متعلقہ) جگہوں پر اسٹور کیا جائے اور ہر نوڈ میں حوالہ جات کا استعمال کرتے ہوئے ان کو جوڑ دیا جائے جو اس کے بچوں کی طرف اشارہ کرتے ہیں۔

آپ میموری میں ایک درخت کی بھی ایک صف کے طور پر نمائندگی کر سکتے ہیں، مخصوص پوزیشنوں میں نوڈس کے ساتھ صف میں متعلقہ پوزیشنوں میں محفوظ ہیں۔ ہم اس باب کے آخر میں اس امکان کی طرف لوٹتے ہیں۔ ہمارے نمونے کے Python کوڈ کے لیے ہم حوالہ جات کا استعمال کرتے ہوئے نوڈس کو جوڑنے کا طریقہ استعمال کریں گے، جیسا کہ باب 5 میں منسلک فہرستوں کو لاگو کیا گیا تھا۔

BinarySearchTree کلاس ہمیں ٹری آبجیکٹ کے لیے ایک کلاس کی ضرورت ہے: وہ آبجیکٹ جو تمام نوڈس کو رکھتا ہے، یا کم از کم اس کی طرف لے جاتا ہے۔ ہم اس کلاس کو BinarySearchTree کہیں گے۔ اس میں صرف ایک فیلڈ ہے، `_root` جو روٹ نوڈ کا حوالہ رکھتا ہے، جیسا کہ فہرست 1-8 میں دکھایا گیا ہے۔ یہ لنکڈ لسٹ کلاس سے بہت ملتا جلتا ہے جو باب 5 میں منسلک فہرستوں کی نمائندگی کے لیے استعمال کیا گیا تھا۔ BinarySearchTree کلاس کو دوسرے نوڈس کے لیے فیلڈز کی ضرورت نہیں ہے کیونکہ وہ سہی روٹ نوڈ سے دوسرے حوالوں کی پیروی کرتے ہوئے حاصل کیے جاتے ہیں۔

فہرست سازی BinarySearchTree 1-8 کلاس کے لیے کنسٹرکٹر

کلاس بائنری سرچ ٹری (آبجیکٹ): # ایک بائنری سرچ ٹری کلاس

```
def __init__(self):
    # درخت اپنی #چابیاں کے ذریعہ نوڈس کو منظم کرتا ہے۔ ابتدائی طور
    پر، یہ خالی ہے۔
    self._root = کوئی نہیں۔
```

کنسٹرکٹر روٹ نوڈ کے حوالے کو شروع کرتا ہے جیسے خالی درخت سے شروع کرنے کے لیے کوئی نہیں۔ جب پہلا نوڈ داخل کیا جاتا ہے، تو `_root` اس کی طرف اشارہ کرے گا جیسا کہ شکل 5-8 کے ویژولائزیشن ٹول کی مثال میں دکھایا گیا ہے۔ یقیناً بہت سے طریقے ہیں جو BinarySearchTree آبجیکٹ پر کام کرتے ہیں، لیکن پہلے، آپ کو ان کے اندر موجود نوڈس کی وضاحت کرنے کی ضرورت ہے۔

نوڈ کلاس درخت کے نوڈس میں ذخیرہ کی جانے والی اشیاء کی نمائندگی کرنے والا ڈیٹا ہوتا ہے (مثال کے طور پر ایڈریس بک میں رابطے کی معلومات)، ان اشیاء کی شناخت کے لیے ایک کلید (اور انہیں ترتیب دینے کے لیے)، اور ہر نوڈ کے دو کے حوالے ہجے۔ ہمیں یاد دلانے کے لیے کہ BinarySearchTree آبیجیکٹ بنانے والے کالرز کو براہ راست نوڈس کو تبدیل کرنے کی اجازت نہیں ہونی چاہیے، ہم اس کلاس کے اندر ایک نجی Node کلاس بناتے ہیں۔ 2-8 کی فہرست بتاتی ہے کہ BinarySearchTree کلاس کے اندر اندرونی کلاس کی وضاحت کیسے کی جا سکتی ہے۔

فہرست سازی Node 2-8 اور BinarySearchTree کلاسز کے کنسٹرکٹرز

کلاس ہائری سرچ ٹری (آبیجیکٹ): # ایک ہائری سرچ ٹری کلاس

...

```

class Node:
    def __init__(self, key):
        self.data = key
        self.leftChild = None
        self.rightChild = None

    def __str__(self):
        return str(self.data) + " (" + str(self.key) + ")"

    def isEmpty(self):
        return self.data is None

    def root(self):
        if self.isEmpty():
            return None
        return self.data, self.key

```

ہائری سرچ ٹری میں ایک نوڈ

کنسٹرکٹر ایک کلید لیتا ہے جو # تلاش کے درخت کے اندر پوزیشن کا تعین کرنے کے لیے استعمال کیا جاتا ہے، # کلید سے وابستہ ڈیٹا اور بائیں اور دائیں چائلڈ نوڈ

اگر معلوم ہو۔

مثال کے طور پر پیرامیٹرز کا پی کریں # آبیجیکٹ کی خصوصیات

self.data = ڈیٹا

self.leftChild = بائیں

self.rightChild = حق

نوڈ کو سٹرنگ ریٹرن کے طور پر پیش کریں " (" + str(self.data) + " (" + str(self.key) + ")")

درخت اپنی # چاہاں کے ذریعہ نوڈس کو منظم کرتا ہے۔ ابتدائی طور پر، یہ خالی ہے۔

def isEmpty(self):

خود کو واپس کریں۔ - جو کوئی نہیں ہے۔

def root(self): if self.isEmpty():

روٹ نوڈ کا ڈیٹا اور کلید حاصل کریں۔

اگر درخت خالی ہے تو استثنیٰ بڑھائیں استثنیٰ ("خالی درخت میں کوئی جڑ نوڈ نہیں")

بصورت دیگر روٹ ڈیٹا اور اس کی کلید واپس کریں۔

self.__root.data, self.__root.key)

Node آبیجیکٹ BinarySearchTree کے طریقوں سے بنائے جاتے ہیں اور جوڑ توڑ کرتے ہیں۔ Node کے اندر فیلڈز کو عوامی اوصاف کے طور پر شروع کیا جا سکتا ہے کیونکہ BinarySearchTree کے طریقے اس بات کا خیال رکھتے ہیں کہ کبھی بھی Node آبیجیکٹ کو واپس نہ کریں۔ یہ اعلامیہ getKey() یا setData() جیسے رسائی کے طریقے بنائے بغیر براہ راست پڑھنے اور لکھنے کی اجازت دیتا ہے۔ نوڈ کنسٹرکٹر صرف فراہم کردہ دلائل سے فیلڈز کو آباد کرتا ہے۔ اگر چائلڈ نوڈس فراہم نہیں کیے جاتے ہیں، تو ان کے حوالہ جات کے لیے فیلڈز None سے بھرے جاتے ہیں۔

ہم ڈیبگ کرتے وقت مواد کو ظاہر کرنے میں مدد کے لیے `_Node` آبجیکٹ کے لیے ایک `_(str)` طریقہ شامل کرتے ہیں۔ خاص طور پر، یہ چائلڈ نوڈس نہیں دکھاتا ہے۔ ہم تھوڑی دیر بعد مکمل درختوں کو ظاہر کرنے کے طریقہ پر تبادلہ خیال کرتے ہیں۔ یہ وہ تمام طریقے ہیں جن کی ضرورت `_Node` آبجیکٹ کے لیے ہے۔ باقی تمام طریقے جن کی آپ وضاحت کرنے ہیں وہ `BinarySearchTree` آبجیکٹ کے لیے ہیں۔

2-8 کی فہرست `BinarySearchTree` آبجیکٹ کے لیے `isEmpty` طریقہ دکھاتی ہے جو یہ چیک کرتی ہے کہ آیا درخت میں کوئی نوڈس ہیں یا نہیں۔ روٹ (`root`) طریقہ روٹ نوڈ کا ڈیٹا اور کلید نکالتا ہے۔ یہ قطار کے لیے `peek()` کی طرح ہے اور اگر درخت خالی ہے تو استثناء پیدا کرتا ہے۔

کچھ پروگرامرز `_Node` کلاس میں نوڈ کے والدین کا حوالہ بھی شامل کرتے ہیں۔ ایسا کرنا کچھ کاموں کو آسان بناتا ہے لیکن دوسروں کو پیچیدہ بناتا ہے، لہذا ہم اسے یہاں شامل نہیں کرتے ہیں۔ والدین کا حوالہ شامل کرنے سے باب 5 میں بیان کردہ `DoublyLinkedList` کلاس کی طرح کچھ حاصل ہوتا ہے، "لنکڈ فہرستیں"؛ یہ بعض سیاق و سباق میں مفید ہے لیکن پیچیدگی کا اضافہ کرتا ہے۔

ہم نے ہر نوڈ کے لیے کلید کو اس کے اپنے فیلڈ میں اسٹور کر کے ایک اور ڈیزائن کا انتخاب کیا ہے۔ صفوں پر مبنی ڈیٹا سٹرکچرز کے لیے، ہم نے ایک کلیدی فنکشن استعمال کرنے کا انتخاب کیا ہے جو ہر صف کے آئٹم سے کلید نکالتا ہے۔ یہ نقطہ نظر صفوں کے لیے زیادہ آسان تھا کیونکہ کیز کو ڈیٹا سے الگ ذخیرہ کرنے کے لیے ڈیٹا سرنی کے ساتھ کلیدی سرنی کے برابر کی ضرورت ہوگی۔ نامزد فیلڈز کے ساتھ نوڈ کلاس کے معاملے میں، ایک کلیدی فیلڈ شامل کرنے سے کوڈ کو شاید زیادہ پڑھنے کے قابل اور کچھ فنکشن کالز سے گریز کر کے کچھ زیادہ موثر بناتا ہے۔ یہ کلید کو ڈیٹا سے زیادہ آزاد بھی بناتا ہے، جس سے لچک میں اضافہ ہوتا ہے اور ڈیٹا میں تبدیلی کے باوجود بھی غیر تبدیل شدہ کلیدوں جیسی رکاوٹوں کو نافذ کرنے کے لیے استعمال کیا جا سکتا ہے۔

`BinarySearchTree` کلاس کے کئی طریقے ہیں۔ وہ نوڈس کو تلاش کرنے، داخل کرنے، حذف کرنے اور عبور کرنے کے لیے استعمال ہوتے ہیں۔ اور درخت کی نمائش کے لیے۔ ہم ان میں سے ہر ایک کی الگ الگ تحقیقات کرتے ہیں۔

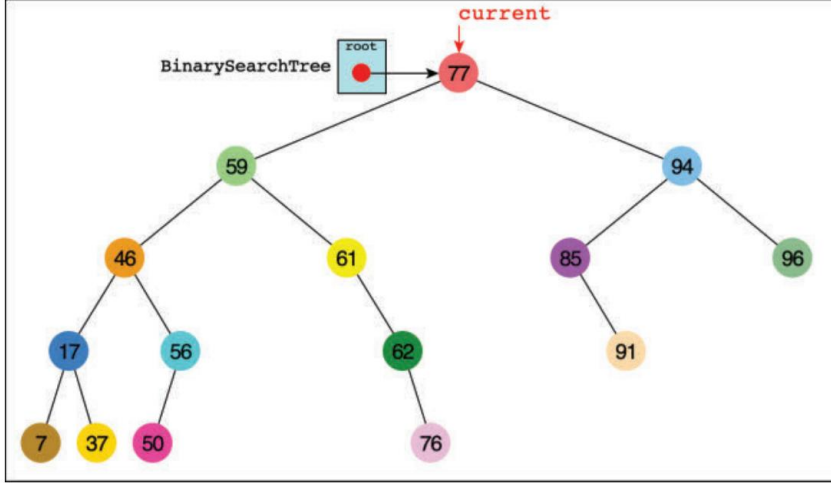
نوڈ تلاش کرنا

ایک مخصوص کلید کے ساتھ نوڈ تلاش کرنا درخت کے بڑے آپریشنز میں سب سے آسان ہے۔ یہ سب سے اہم بھی ہے کیونکہ یہ بائنری سرچ ٹری کے مقصد کے لیے ضروری ہے۔

ویژولائزیشن ٹول ہر نوڈ کے لیے صرف کلید اور اس کے ڈیٹا کے لیے ایک رنگ دکھاتا ہے۔ ذہن میں رکھیں کہ ڈیٹا ڈھانچہ کا مقصد صرف کلید یا سادہ رنگ نہیں بلکہ ریکارڈز کا مجموعہ ذخیرہ کرنا ہے۔ چابیاں سادہ عدد سے زیادہ ہو سکتی ہیں۔ کسی بھی ڈیٹا کی قسم جس کا آرڈر دیا جا سکتا ہے استعمال کیا جا سکتا ہے۔ یہاں دکھایا گیا تصور اور مثالیں اختصار کے لیے عدد کا استعمال کرتی ہیں۔ ایک نوڈ کو اس کی کلید سے دریافت کرنے کے بعد، یہ وہ ڈیٹا ہے جو کال کرنے والے کو واپس کیا جاتا ہے، خود نوڈ کو نہیں۔

نوڈ تلاش کرنے کے لیے ویژولائزیشن ٹول کا استعمال کرتے ہوئے ویژولائزیشن ٹول کو دیکھیں اور ایک نوڈ چنیں، ترجیحاً درخت کے نیچے (جت تک ممکن ہو جڑ سے)۔ اس نوڈ میں دکھایا گیا نمبر اس کی کلیدی قدر ہے۔ ہم یہ ظاہر کرنے جا رہے ہیں کہ کلیدی قدر کے پیش نظر ویژولائزیشن ٹول نوڈ کو کیسے تلاش کرتا ہے۔

اس بحث کے مقاصد کے لیے، ہم کلیدی قدر 50 کے ساتھ آئٹم کے حامل نوڈ کو تلاش کرنے کا انتخاب کرتے ہیں، جیسا کہ شکل 8-8 میں دکھایا گیا ہے۔ بلاشبہ، جب آپ ویڈیو لائزیشن ٹول چلاتے ہیں، تو آپ کو ایک مختلف درخت مل سکتا ہے اور آپ کو ایک مختلف کلیدی قیمت لینے کی ضرورت پڑ سکتی ہے۔

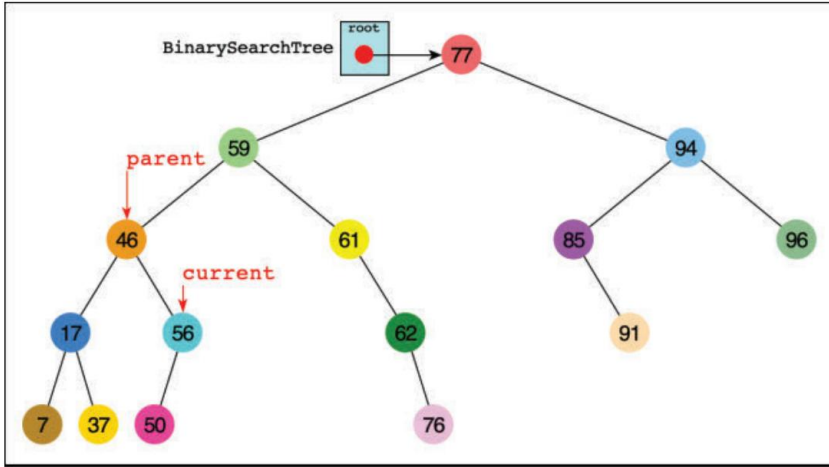


تصویر 8-8 کلید 50 کے ساتھ نوڈ تلاش کرنا

ٹیکسٹ اٹری باکس میں کلیدی قدر درج کریں، شفٹ کی کو دبائے رکھیں، اور سرچ بٹن کو منتخب کریں، اور پھر اسٹیپ بٹن۔ اسٹیپ بٹن کو بار بار دبانے سے، آپ کلید 50 کو تلاش کرنے کے لیے اٹھائے گئے تمام انفرادی اقدامات کو دیکھ سکتے ہیں۔ دوسری پریس پر، موجودہ پوائنٹر درخت کی جڑ میں ظاہر ہوتا ہے، جیسا کہ شکل 8-8 میں دیکھا گیا ہے۔ اگلے کلک پر، ایک پیرنٹ پوائنٹر ظاہر ہوتا ہے جو موجودہ پوائنٹر کی پیروی کرے گا۔ ایک لمحے کے لیے اس پوائنٹر اور کوڈ ڈسپلے کو نظر انداز کریں؛ ہم ان کو جلد ہی تفصیل سے بیان کرتے ہیں۔

جیسا کہ ویڈیو لائزیشن ٹول مخصوص نوڈ کو تلاش کرتا ہے، یہ موجودہ نوڈ پر فیصلہ کرتا ہے۔ یہ مطلوبہ کلید کا موازنہ موجودہ نوڈ پر موجود کلید سے کرتا ہے۔ اگر یہ ایک جیسا ہے تو اسے مطلوبہ نوڈ مل گیا ہے اور وہ چھوڑ سکتا ہے۔ اگر نہیں، تو اسے فیصلہ کرنا ہوگا کہ آگے کہاں دیکھنا ہے۔

شکل 8-8 میں موجودہ تیر جڑ سے شروع ہوتا ہے۔ پروگرام گول کلید کی قدر 50 کا جڑ کی قدر سے موازنہ کرتا ہے، جو کہ 77 ہے۔ گول کلید کم ہے، اس لیے پروگرام جانتا ہے کہ مطلوبہ نوڈ درخت کے بائیں جانب ہونا چاہیے — یا تو جڑ کے بائیں بچے یا ان میں سے ایک اس بچے کی اولاد روٹ کے بائیں چائلڈ کی ویلیو 59 ہے، لہذا اور 50 کا com پیریڈن ظاہر کرے گا کہ مطلوبہ نوڈ 59 کے بائیں ذیلی درخت میں ہے۔ کرنٹ کرنٹ ایرو 46 پر جاتا ہے، اس ذیلی درخت کی جڑ۔ اس بار، 50 نوڈ سے بڑا ہے، تو یہ دائیں طرف جاتا ہے، نوڈ 56 پر، جیسا کہ شکل 8-9 میں دکھایا گیا ہے۔ چند قدم بعد، 50 کا 56 سے موازنہ پروگرام کو بائیں بچے کی طرف لے جاتا ہے۔ اس لیف نوڈ کا موازنہ ظاہر کرتا ہے کہ 50 نوڈ کی کلیدی قدر کے برابر ہے، اس لیے اسے وہ نوڈ مل گیا ہے جس کی ہم نے تلاش کی تھی۔



تصویر 8-9 کلید 50 کو تلاش کرنے کا دوسرا آخری مرحلہ

مطلوبہ نوڈ تلاش کرنے کے بعد ویژولائزیشن ٹول تھوڑا بدل جاتا ہے۔ موجودہ تیر نوڈ تیر میں بدل جاتا ہے (اور پیرنٹ پی میں تبدیل ہوتا ہے)۔ یہ اس وجہ سے ہے کہ کوڈ میں variables نام رکھا گیا ہے، جسے ہم اگلے حصے میں دکھاتے ہیں۔ ٹول نوڈ کو ڈھونڈنے کے بعد اس کے ساتھ کچھ نہیں کرتا، سوائے اس کے گھیرے میں لینے اور ایک پیغام ظاہر کرنے کے کہ یہ مل گیا ہے۔ ایک سنجیدہ پروگرام پایا نوڈ پر کچھ آپریشن کرے گا، جیسے کہ اس کے مواد کو ظاہر کرنا یا اس کے کسی ایک فیلڈ کو تبدیل کرنا۔

نوڈ لسٹنگ 3-8 تلاش کرنے کے لیے ارگر کوڈ find() اور search() طریقوں کا کوڈ دکھاتا ہے۔ find() طریقہ نجی ہے کیونکہ یہ نوڈ آبیچٹ کو واپس کرسکتا ہے۔ BinarySearchTree کلاس کے کال کرنے والے ایک نوڈ میں ذخیرہ شدہ ڈیٹا حاصل کرنے کے لیے search() طریقہ استعمال کرتے ہیں۔

فہرست 3-8 اس کی کلید کی بنیاد پر بائری سرچ ٹری نوڈ تلاش کرنے کے طریقے

کلیڈیں (TheTree) ee (آبیچٹ):

...

def dnif (خود، مقصد):

موجودہ = خود۔جڑ

والدین = خود جبکہ (موجودہ اور

ایک اندرونی نوڈ تلاش کریں جس کی کلید # گول اور اس کے والدین سے مماثل ہو۔
روٹ سے شروع کریں اور موجودہ نوڈ کے والدین کو ٹریک کریں # جب کہ دریافت
کرنے کے لیے ایک درخت باقی ہے اور موجودہ کلید مقصد نہیں ہے # ایک سطح نیچے جانے
کے لیے تیار ہوں

گول ! (current.key = والدین = موجودہ موجودہ = current.leftChild) = اگر گول # else < current.key

موجودہ ہے، رائٹ چائلڈ)

ایڈوانس کرنا کو بائیں سب ٹری جب

موجودہ کلید سے کم، ورنہ ٹھیک ہے۔

اگر لوپ کسی نوڈ پر ختم ہوتا ہے، تو اس میں گول کلید کی واپسی ہونی چاہیے (موجودہ، والدین)

نوڈ یا کوئی نہیں اور والدین کو واپس کریں۔

مقصد کی کلید کے (find(goal)) node ہے self if node else None # w/ واپس node.data if node else None اور کوئی سب سے پہلے، نوڈ تلاش کیجیں کریں

(find()) کی واحد دلیل گول ہے، کلیدی قدر جس کو تلاش کرنا ہے۔ یہ معمول متغیر کرنٹ بناتا ہے تاکہ اس نوڈ کو بولڈ کیا جا سکے جس کی فی الحال جانچ کی جا رہی ہے۔ روٹین جڈ سے شروع ہوتی ہے - واحد نوڈ جس تک یہ براہ راست رسائی حاصل کر سکتا ہے۔ یعنی یہ روٹ پر کرنٹ سیٹ کرتا ہے۔ یہ خود کے لیے ایک متغیر متغیر بھی متعین کرتا ہے، جو کہ درخت کی چیز ہے۔ ویژولائزیشن ٹول میں، پیرنٹ ٹری آبجیکٹ کی طرف اشارہ کرنا شروع کرتا ہے۔ چونکہ پیرنٹ لنکس نوڈس میں محفوظ نہیں ہوتے ہیں، اس لیے (find()) طریقہ کرنٹ کے پیرنٹ نوڈ کو ٹریک کرتا ہے تاکہ وہ اسے گول نوڈ کے ساتھ کالر کو واپس کر سکے۔ یہ صلاحیت دوسرے طریقوں میں بہت کارآمد ثابت ہوگی۔ پیرنٹ متغیر ہمیشہ یا تو BinarySearchTree کو تلاش کیا جاتا ہے یا اس کے Node آبجیکٹ میں سے ایک ہوتا ہے۔

while لوپ میں، (find()) پہلے تصدیق کرتا ہے کہ کرنٹ None نہیں ہے اور کچھ موجودہ نوڈ کا حوالہ دیتا ہے۔ اگر ایسا نہیں ہوتا ہے تو، تلاش لیف نوڈ سے آگے نکل گئی ہے (یا خالی درخت سے شروع ہوئی ہے)، اور گول نوڈ درخت میں نہیں ہے۔ while test com کا دوسرا حصہ موجودہ نوڈ کے کلیدی فیڈ کی قدر کے ساتھ، تلاش کی جانے والی قدر، گول کا موازنہ کرتا ہے۔ اگر کلید ملتی ہے، تو لوپ کیا جاتا ہے۔ اگر ایسا نہیں ہوتا ہے، تو موجودہ کو مناسب ذیلی درخت کی طرف بڑھنے کی ضرورت ہے۔ سب سے پہلے، یہ پیرنٹ کو موجودہ نوڈ بننے کے لیے اپ ڈیٹ کرتا ہے اور پھر کرنٹ کو اپ ڈیٹ کرتا ہے۔ اگر بدف کرنٹ کی کلید سے کم ہے تو کرنٹ اپنے بائیں بچے کی طرف بڑھتا ہے۔ اگر بدف کرنٹ کی کلید سے بڑا ہے، تو کرنٹ اپنے صحیح بچے کی طرف بڑھتا ہے۔

نوڈ نہیں مل سکتا

اگر کرنٹ None کے برابر ہو جاتا ہے، تو آپ جس نوڈ کو تلاش کر رہے تھے اسے تلاش کیے بغیر لائن کے آخر تک پہنچ گئے ہیں، لہذا یہ درخت میں نہیں ہو سکتا۔ ایسا ہو سکتا ہے اگر روٹ نوڈ کوئی نہ ہو یا اگر چائلڈ لنکس کو فالو کرنے سے بچے کے بغیر نوڈ بن جائے (اس طرف جہاں گول کی جائے گی)۔ موجودہ نوڈ (کوئی نہیں) اور اس کے والدین دونوں کو کال کرنے والے کو واپس کر دیا جاتا ہے تاکہ نتیجہ کی نشاندہی کی جا سکے۔ ویژولائزیشن ٹول میں، ایک کلید داخل کرنے کی کوشش کریں جو درخت میں ظاہر نہیں ہوتی ہے اور تلاش کو منتخب کریں۔ اس کے بعد آپ دیکھیں گے کہ موجودہ پوائنٹر موجودہ نوڈس کے ذریعے اترتا ہے اور اس جگہ پر اترتا ہے جہاں کلید ملنی چاہیے لیکن کوئی نوڈ موجود نہیں ہے۔

"خالی جگہ" کی طرف اشارہ کرتا ہے کہ متغیر کی قدر کوئی نہیں ہے۔

نوڈ ملا

اگر موجودہ وقت کے دوران لوپ کی حالت مطمئن نہیں ہوتی ہے جبکہ موجودہ درخت میں کچھ نوڈ کا حوالہ دیتا ہے، تو لوپ باہر نکل جاتا ہے، اور موجودہ کلید کا مقصد ہونا چاہیے۔ یعنی، اس نے نوڈ کو تلاش کیا ہے اور اس کا موجودہ حوالہ دیا ہے۔ یہ پیرنٹ ریفرنس کے ساتھ نوڈ کا حوالہ واپس کرتا ہے تاکہ روٹین جسے (find()) کہا جاتا ہے نوڈ (یا اس کے والدین کے) ڈیٹا میں سے کسی تک رسائی حاصل کر سکے۔ نوٹ کریں کہ یہ کلید تلاش کرنے میں کامیابی اور ناکامی دونوں کے لیے کرنٹ کی قدر لوٹاتا ہے۔ جب مقصد نہیں ملتا ہے تو یہ کوئی نہیں ہے۔

سرج () طریقہ اپنے نوڈ اور پیرنٹ (p) متغیرات کو سیٹ کرنے کے لیے (find()) طریقہ کو کال کرتا ہے۔

(find()) طریقہ واپس آنے کے بعد آپ ویژولائزیشن ٹول میں یہی دیکھتے ہیں۔ اگر کوئی غیر کا حوالہ نہیں ملا تو تلاش () اس نوڈ کا ڈیٹا لوٹاتا ہے۔ اس صورت میں، طریقہ یہ مانتا ہے کہ نوڈس میں ذخیرہ شدہ ڈیٹا آئٹمز کبھی بھی کوئی نہیں ہو سکتے۔ بصورت دیگر، کال کرنے والا ان میں تمیز نہیں کر سکے گا۔

درخت کی کارکردگی جیسا کہ آپ دیکھ سکتے ہیں، نوڈ کو تلاش کرنے کے لیے درکار وقت کا انحصار درخت میں اس کی گہرائی، جڑ سے نیچے کی سطحوں کی تعداد پر ہوتا ہے۔ اگر درخت متوازن ہے، تو یہ $O(\log N)$ وقت ہے، یا اس سے زیادہ خاص طور پر $O(\log 2N)$ وقت ہے، لاگرتھم سے بیس 2، جہاں N نوڈس کی تعداد ہے۔

یہ بالکل بائری تلاش کی طرح ہے جو صفوں میں کی جاتی ہے جہاں ہر موازنہ کے بعد آدھے نوڈس کو ختم کر دیا جاتا ہے۔ ایک مکمل متوازن درخت بہترین صورت ہے۔ بدترین صورت میں، درخت مکمل طور پر غیر متوازن ہے، جیسا کہ شکل 6-8 میں دکھایا گیا ہے، اور مطلوبہ وقت $O(N)$ ہے۔

ہم اس باب کے اختتام کی طرف `find()` اور دیگر کارروائیوں کی کارکردگی پر تبادلہ خیال کرتے ہیں۔

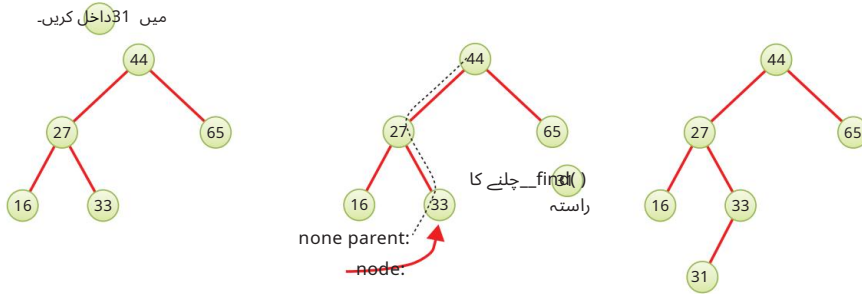
نوڈ داخل کرنا

نوڈ داخل کرنے کے لیے، آپ کو پہلے اسے داخل کرنے کے لیے جگہ تلاش کرنا ہوگی۔ یہ وہی عمل ہے جو کسی ایسے نوڈ کو تلاش کرنے کی کوشش کرتا ہے جو کہ موجود نہیں ہے، جیسا کہ پہلے "نوڈ کو تلاش نہیں کر سکتے" سیکشن میں بیان کیا گیا ہے۔ آپ جڑ سے مناسب نوڈ کی طرف راستے پر چلتے ہیں۔ یہ یا تو ایک نوڈ ہے جس کی کلید نوڈ ڈالی جائے گی یا کوئی نہیں، اگر یہ نئی کلید ہے۔ اگر یہ ایک ہی کلید ہے، تو آپ اسے صحیح ذیلی درخت میں ڈالنے کی کوشش کر سکتے ہیں، لیکن ایسا کرنے سے کچھ پیچیدگیاں بڑھ جاتی ہیں۔ دوسرا آپشن اس نوڈ کے ڈیٹا کو نئے ڈیٹا سے تبدیل کرنا ہے۔ ابھی کے لیے، ہم صرف منفرد کلیدوں کو داخل کرنے کی اجازت دیتے ہیں۔ ہم بعد میں ڈپلیکیٹ کیڑ پر بات کرتے ہیں۔

اگر داخل کرنے کی کلید درخت میں نہیں ہے، تو پھر `find()` نوڈ کے حوالے کے لیے پرنٹ ریفرنس کے ساتھ `None` واپس کرتا ہے۔ نیا نوڈ والدین کے بائیں یا دائیں بچے کے طور پر جڑا ہوا ہے، اس بات پر منحصر ہے کہ آیا نئے نوڈ کی کلید والدین کی کلید سے کم ہے یا زیادہ۔ اگر `find()` کے ذریعہ واپس کیا گیا پرنٹ حوالہ خود ہے، خود `BinarySearchTree` نوڈ روٹ نوڈ بن جاتا ہے۔

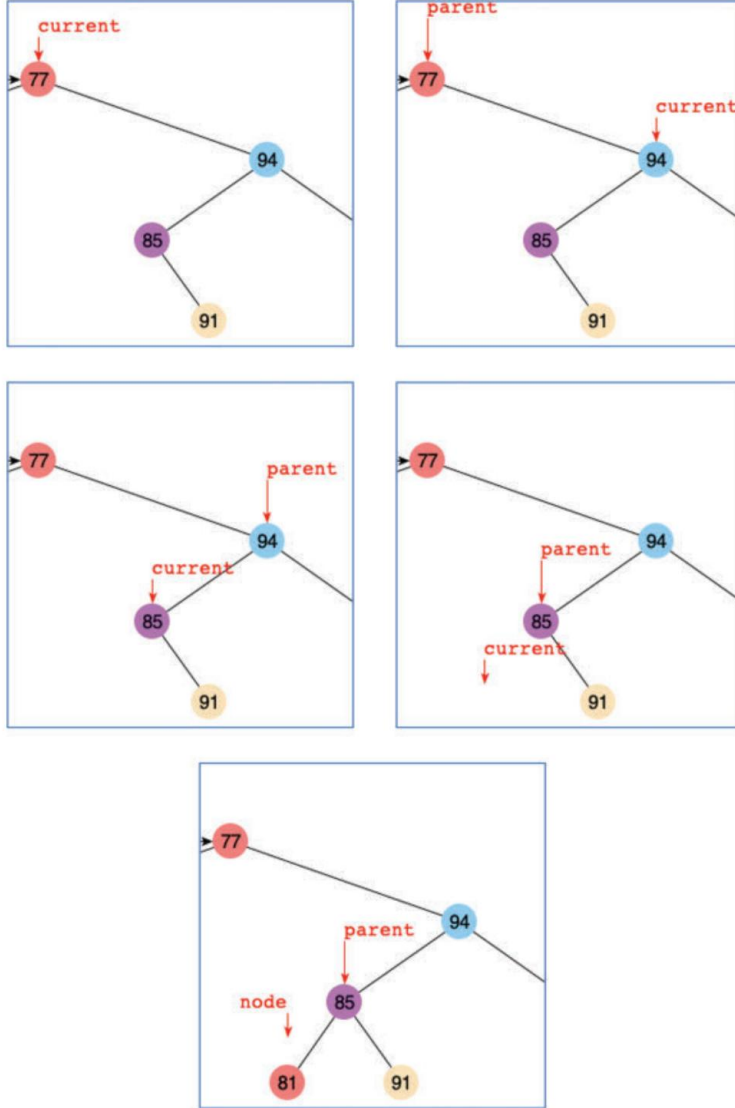
شکل 10-8 ایک نوڈ داخل کرنے کے عمل کی وضاحت کرتا ہے، کلید 31 کے ساتھ، درخت میں۔ `find()` طریقہ روٹ نوڈ سے راستے پر چلنا شروع کرتا ہے۔ کیونکہ 31 روٹ نوڈ کلید 44 سے کم ہے، یہ بائیں چائلڈ لنک کی پیروی کرتا ہے۔ اس بچے کی کلید 27 ہے، لہذا یہ اس بچے کے دائیں چائلڈ لنک کی پیروی کرتی ہے۔ وہاں یہ کلید 33 کا سامنا کرتا ہے، لہذا یہ دوبارہ بائیں چائلڈ لنک کی پیروی کرتا ہے۔

یہ کوئی نہیں ہے، لہذا `find(31)` کلید 33 کے ساتھ لیف نوڈ کی طرف اشارہ کرنے والے پرنٹ کے ساتھ رک جاتا ہے۔ کلید 31 کے ساتھ نیا لیف نوڈ بن جاتا ہے، اور والدین کے بائیں چائلڈ لنک کو اس کا حوالہ دینے کے لیے سیٹ کیا جاتا ہے۔



تصویر 8-10 بائری سرچ ٹری میں نوڈ ڈالنا

نوڈ داخل کرنے کے لیے ویژولائزیشن ٹول کا استعمال کرتے ہوئے ویژولائزیشن ٹول کے ساتھ نیا نوڈ داخل کرنے کے لیے، ایک کلیدی قدر درج کریں جو درخت میں نہیں ہے اور داخل کریں بٹن کو منتخب کریں۔ پروگرام کے لیے پہلا قدم یہ معلوم کرنا ہے کہ اسے کہاں داخل کیا جانا چاہیے۔ مثال کے طور پر، پہلے کی مثال سے درخت میں 81 ڈالنا `find()` فہرست 3-8 کا طریقہ کہتا ہے، جس کی وجہ سے تلاش تصویر 8-11 میں دکھائے گئے راستے پر چلتی ہے۔



تصویر 8-11 ویژولائزیشن ٹول کا استعمال کرتے ہوئے کلید 81 کے ساتھ نوڈ داخل کرنے کے اقدامات

موجودہ پوائنٹر روٹ نوڈ سے کلید 77 سے شروع ہوتا ہے۔ 81 کو بڑا ہونے کی تلاش میں، یہ دائیں چائلڈ، نوڈ 94 پر جاتا ہے۔

نوڈ 85 تک۔ پیرنٹ پوائنٹر ان میں سے ہر ایک مرحلے پر موجودہ پوائنٹر کی پیروی کرتا ہے۔ جب کرنٹ نوڈ 85 تک پہنچتا ہے، تو یہ اپنے بائیں بچے کے پاس جاتا ہے اور اسے غائب پایا جاتا ہے۔ `find()` پر کال کوئی نہیں اور پیرنٹ پوائنٹر واپس کرتی ہے۔

خالی بچے کے ساتھ پیرنٹ نوڈ کا پتہ لگانے کے بعد جہاں نئی کلید کو جانا چاہیے، ویڈیولائزیشن ٹول کلید 81 کے ساتھ ایک نیا نوڈ بناتا ہے، جس میں کچھ ڈیٹا رنگ سے ظاہر ہوتا ہے، اور نوڈ 85 کے بائیں چائلڈ پوائنٹر، پیرنٹ، کو پوائنٹ کرنے کے لیے سیٹ کرتا ہے۔ اس پر `find()` کے ذریعہ لوٹا ہوا نوڈ پوائنٹر ہٹا دیا گیا ہے کیونکہ یہ اب بھی کوئی نہیں ہے۔

نوڈ داخل کرنے کے لیے پائنتھون کوڈ `insert()` طریقہ کلید اور ڈیٹا داخل کرنے کے لیے پیرامیٹرز لیتا ہے، جیسا کہ فہرست 8-4 میں دکھایا گیا ہے۔ یہ نئے نوڈ کی کلید کے ساتھ `find()` طریقہ کو کال کرتا ہے تاکہ یہ معلوم کیا جا سکے کہ آیا وہ کلید پہلے سے موجود ہے اور اس کا پیرنٹ نوڈ کہاں ہونا چاہیے۔ یہ نفاذ درخت میں صرف منفرد کلیدوں کی اجازت دیتا ہے، لہذا اگر اسے اسی کلید کے ساتھ کوئی نوڈ ملتا ہے، تو `insert()` اس کلید کے لیے ڈیٹا کو اپ ڈیٹ کرتا ہے اور `False` لوٹاتا ہے تاکہ یہ ظاہر کیا جا سکے کہ کوئی نیا نوڈ نہیں بنایا گیا تھا۔

فہرست 8-4 BinarySearchTree کا `insert()` طریقہ

کاپی رائٹ © TheCrazyCyanide (انجیکٹ):

...

```
key, data): node, parent = self._find(key) if node:
    def insert(self,
```

بائنری میں ایک نیا نوڈ داخل کریں # تلاش کے درخت کو تلاش کریں کہ
اس کی کلید اسے کہاں رکھتی ہے اور اس کا ڈیٹا اسٹور کرتی ہے

درخت میں کلید تلاش کرنے کی کوشش کریں # اور اس کا بنیادی نوڈ
حاصل کریں۔

اگر ہمیں اس کلید کے ساتھ کوئی نوڈ ملتا ہے، # پھر نوڈ کے ڈیٹا کو اپ
ڈیٹ کریں # اور بغیر اندراج کے جھنڈا واپس کریں۔

node.data =

غلط واپسی

اگر والدین خود ہیں:

```
data, right=node) # تو (key, data) = self._root = self._Node(
parent.leftChild = self._Node( key,
```

بائیں کے طور پر نیا نوڈ داخل کریں۔

والدین کا بچہ

بصورت دیگر والدین کے دائیں # بچے کے طور پر نیا نوڈ داخل کریں۔

دوسری:

```
parent.rightChild = self._Node( key, data, right=node)
```

واپس سچ

درست اندراج کے لیے جھنڈا واپس کریں۔

اگر کوئی مماثل نوڈ نہیں ملا، تو اندراج اس بات پر منحصر ہے کہ `find()` سے کس قسم کا پیرنٹ حوالہ واپس کیا گیا تھا۔ اگر یہ خود ہے، `BinarySearchTree` خالی ہونا ضروری ہے، لہذا نیا نوڈ درخت کا جڑ نوڈ بن جاتا ہے۔ بصورت دیگر، والدین ایک نوڈ ہے، لہذا `insert()` نئے نوڈ کی کلید کا والدین کی کلید سے موازنہ کر کے فیصلہ کرتا ہے کہ کون سا بچہ نیا نوڈ حاصل کرے گا۔ اگر نئی کلید کم ہے، تو نیا نوڈ بائیں چائلڈ بن جاتا ہے۔ دوسری صورت میں، یہ صحیح بچہ بن جاتا ہے۔ آخر میں، `insert()` واپسی `True` اس بات کی نشاندہی کرنے کے لیے کہ داخل کرنا کامیاب ہو گیا۔

جب insert() نوڈ بنانا ہے، تو یہ نئے نوڈ کے رائٹ چائلڈ کو find() سے واپس آنے والے نوڈ پر سیٹ کرتا ہے۔ آپ حیران ہوں گے کہ یہ وہاں کیوں ہے، خاص طور پر اس لیے کہ اس مقام پر نوڈ صرف None ہو سکتا ہے (اگر یہ None نہ ہوتا تو insert() اس مقام تک پہنچنے سے پہلے False واپس کر دیتا)۔ وجہ ڈپلیکیٹ چابیاں کے ساتھ کیا کرنا ہے واپس چلا جاتا ہے۔ اگر آپ ڈپلیکیٹ چابیاں والے نوڈس کی اجازت دیتے ہیں، تو آپ کو انہیں کہیں رکھنا چاہیے۔ بائٹری سرچ ٹری ڈیف انیشن کا کہنا ہے کہ نوڈ کی کلید اس کے دائیں بچے کی کلید سے کم یا اس کے برابر ہوتی ہے۔ لہذا، اگر آپ ڈپلیکیٹ چابیاں کی اجازت دیتے ہیں، تو ڈپلیکیٹ نوڈ بائیں بچے میں نہیں جا سکتا۔ نئے نوڈ کے رائٹ چائلڈ کے طور پر None کے علاوہ کسی اور چیز کی وضاحت کر کے، اس کی کلید کے ساتھ دوسرے نوڈس کو برقرار رکھا جا سکتا ہے۔ ہم ایک مشق کے طور پر چھوڑتے ہیں کہ ڈپلیکیٹ کیز کے ساتھ نوڈس کیسے داخل کریں (اور حذف کریں)۔

درخت کو عبور کرنا

درخت کو عبور کرنے کا مطلب ہے ہر نوڈ کو ایک مخصوص ترتیب میں جانا۔ درخت کو عبور کرنا کچھ حالات میں مفید ہے جیسے کہ تمام ریکارڈز کو تلاش کرنے کے لیے ان کو تلاش کرنا جن کے لیے کچھ کارروائی کی ضرورت ہے (مثال کے طور پر، گاڑی کے پرزے جو کسی خاص ملک سے حاصل کیے گئے ہوں)۔

یہ عمل عام طور پر نوڈس کو تلاش کرنے، داخل کرنے اور حذف کرنے کے طور پر استعمال نہیں ہو سکتا ہے لیکن اس کے باوجود یہ اہم ہے۔

آپ ایک درخت کو تین آسان طریقوں سے عبور کر سکتے ہیں۔ انہیں پری آرڈر، ان آرڈر اور پوسٹ آرڈر کہا جاتا ہے۔ بائٹری سرچ ٹریز کے لیے سب سے زیادہ استعمال ہونے والا آرڈر ان آرڈر ہے، تو آئیے پہلے اسے دیکھتے ہیں اور پھر دوسرے دو پر مختصر طور پر واپس آئے ہیں۔

ان آرڈر ٹراورسل

بائٹری سرچ ٹری کا ان آرڈر ٹراورسل تمام نوڈس کو ان کی کلیدی اقدار کے صعودی ترتیب میں دیکھنے کا سبب بنتا ہے۔ اگر آپ بائٹری ٹری میں ڈیٹا کی ایک فہرست بنانا چاہتے ہیں جو ان کی چابیاں کے مطابق ترتیب دی گئی ہے، تو یہ ایسا کرنے کا ایک طریقہ ہے۔

ٹراورسل کرنے کا سب سے آسان طریقہ تکرار کا استعمال ہے (جس پر باب 6 میں بحث کی گئی ہے)۔ پورے درخت کو عبور کرنے کے لیے ایک تکراری طریقہ کو نوڈ کے ساتھ دلیل کے طور پر کہا جاتا ہے۔ ابتدائی طور پر، یہ نوڈ جڑ ہے، طریقہ کار کو صرف تین چیزیں کرنے کی ضرورت ہے:

1. نوڈ کے بائیں سب ٹری کو عبور کرنے کے لیے خود کو کال کریں۔

2. نوڈ پر جائیں۔

3. نوڈ کے دائیں ذیلی درخت کو عبور کرنے کے لیے خود کو کال کریں۔

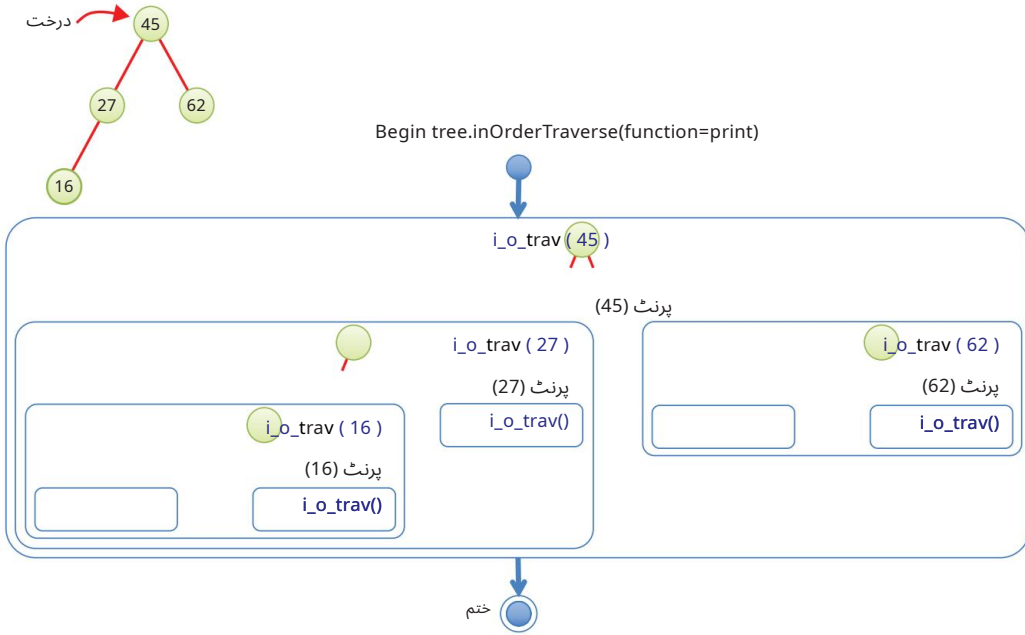
یاد رکھیں کہ نوڈ پر جانے کا مطلب ہے اس کے لیے کچھ کرنا: اسے ڈسپلے کرنا، کسی فیلڈ کو اپ ڈیٹ کرنا، اسے قطار میں شامل کرنا، اسے فائل میں لکھنا، یا کچھ بھی۔

تین ٹراورسل آرڈرز کسی بھی بائٹری ٹری کے ساتھ کام کرتے ہیں، نہ صرف بائٹری سرچ ٹری کے ساتھ۔

ٹراورسل میکانزم نوڈس کی کلیدی اقدار پر کوئی توجہ نہیں دیتا ہے۔ یہ صرف نوڈ کے بچوں اور ڈیٹا سے متعلق ہے۔ دوسرے لفظوں میں، ان آرڈر ٹراورسل کا مطلب ہے "کلیدی اقدار کو بڑھانے کی ترتیب میں" صرف اس صورت میں جب بائٹری سرچ ٹری کے معیار کو درخت میں نوڈس رکھنے کے لیے استعمال کیا جاتا ہے۔ ان آف آرڈر سے مراد بائیں اور دائیں ذیلی درختوں کے درمیان ایک نوڈ کا دورہ کیا جاتا ہے۔ پری آرڈر کا مطلب ہے بچوں سے ملنے سے پہلے نوڈ کا دورہ کرنا، اور پوسٹ آرڈر بچوں سے ملنے کے بعد نوڈ کا دورہ کرنا۔ یہ فرق ایسا ہے۔

باب 4 میں بیان کردہ ریاضی کے تاثرات کے لیے انفکس اور پوسٹ فکس اشارے کے درمیان فرق ، "اسٹیک اور قطاریں۔"

یہ دیکھنے کے لیے کہ یہ بار بار چلنے والا ٹراورسل کیسے کام کرتا ہے، شکل 12-8 ان کالوں کو دکھاتی ہے جو ایک چھوٹے بائنری ٹری کے ان آرڈر ٹراورسل کے دوران ہوتی ہیں۔ درخت کا متغیر چار نوڈ بائنری سرچ ٹری کا حوالہ دیتا ہے۔ اعداد و شمار درخت پر ایک `inOrderTraverse()` طریقہ کی درخواست کو ظاہر کرتا ہے جو اس کے ہر نوڈس پر پرنٹ فنکشن کو کال کرے گا۔



شکل 8-12 ایک چھوٹے سے درخت کی ترتیب سے گزرنا

نیلے گول مستطیل ہر ذیلی درخت پر دوبارہ آنے والی کالیں دکھاتے ہیں۔ اعداد و شمار میں تمام کالوں کو فٹ کرنے کے لئے تکراری طریقہ کا نام `i_o_trav()` کے طور پر مختص کیا گیا ہے۔ پہلی (بابر ترین) کال روٹ نوڈ (کلید 45) پر ہوتی ہے۔ ہر تکراری کال پہلے بیان کردہ تین مراحل کو انجام دیتی ہے۔ سب سے پہلے، یہ بائیں ذیلی درخت پر ایک تکراری کال کرتا ہے، جس کی جڑ کلید 27 پر ہوتی ہے۔ یہ شکل کے بائیں جانب ایک اور نیلے گول مستطیل کے طور پر ظاہر ہوتا ہے۔

کلید 27 پر جڑی ہوئی سب ٹری کو پروسیس کرنا اس کے بائیں سب ٹری پر ایک بار کال کرنے سے شروع ہوتا ہے، جو کلید 16 پر جڑی ہوئی ہے۔ ایک اور مستطیل اس کال کو نیچے بائیں طرف دکھاتا ہے۔ پہلے کی طرح، اس کا پہلا کام اس کے بائیں سب ٹری پر بار بار آنے والی کال کرنا ہے۔ وہ ذیلی درخت خالی ہے کیونکہ یہ ایک لیف نوڈ ہے اور تصویر میں بغیر کسی دلیل کے `i_o_trav()` کو کال کے طور پر دکھایا گیا ہے۔ کیونکہ سب ٹری خالی ہے، کچھ نہیں ہوتا اور بار بار آنے والی کال واپس آتی ہے۔

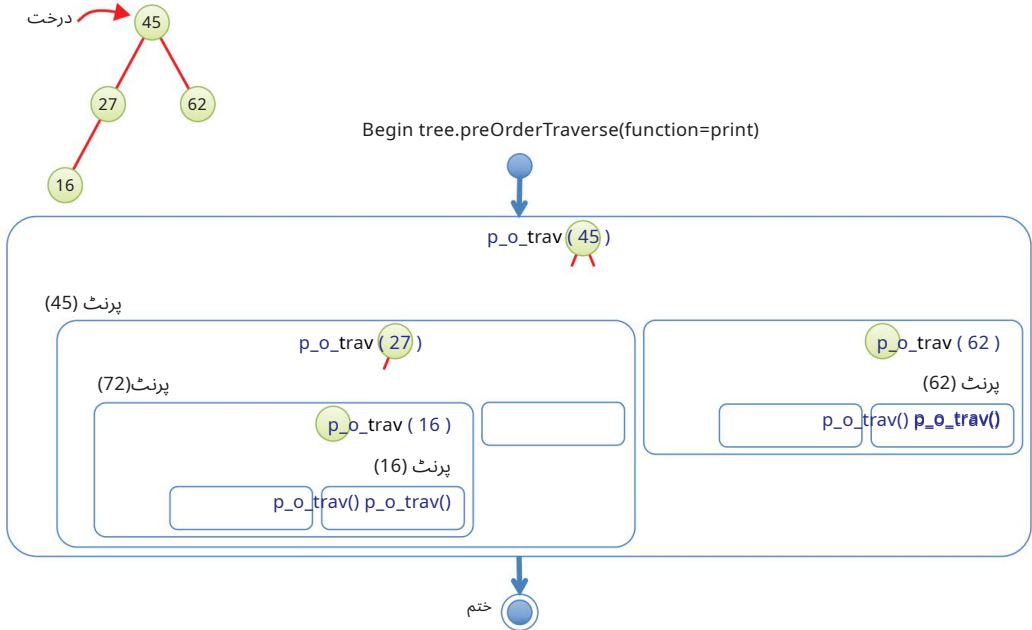
`i_o_trav(16)` کو کال میں واپس ، یہ اب مرحلہ 2 تک پہنچ جاتا ہے اور نوڈ پر ہی فنکشن، پرنٹ کو انجام دے کر نوڈ کو "وزٹ" کرتا ہے۔ یہ تصویر میں بطور پرنٹ (16) سیاہ میں دکھایا گیا ہے۔ عام طور پر، نوڈ پر جانا صرف نوڈ کی کلید پرنٹ کرنے سے زیادہ کام کرتا ہے۔ یہ نوڈ میں ذخیرہ شدہ ڈیٹا پر کچھ کارروائی کرے گا۔ اعداد و شمار اس عمل کو نہیں دکھاتا ہے، لیکن یہ اس وقت ہوگا جب پرنٹ (16) کو انجام دیا جائے گا۔

کلید 16 کے ساتھ نوڈ پر جانے کے بعد، یہ مرحلہ 3 کا وقت ہے: خود کو دائیں ذیلی درخت پر کال کریں۔ کلید 16 والے نوڈ کا کوئی صحیح بچہ نہیں ہے، جو سب سے چھوٹے سائز کے مستطیل کے طور پر ظاہر ہوتا ہے کیونکہ یہ خالی ذیلی درخت پر کال ہے۔ یہ کلید 16 پر جڑے ہوئے ذیلی درخت کے لیے عمل کو مکمل کرتا ہے۔ کنٹرول واپس کالر کے پاس جاتا ہے، سب ٹری پر کال کلید 27 پر جڑ جاتی ہے۔

باقی پروسیسنگ اسی طرح ترقی کرتی ہے۔ کلید 27 کے ساتھ نوڈ کا دورہ پرنٹ (27) کو انجام دیتا ہے اور پھر اس کے خالی دائیں ذیلی درخت پر کال کرتا ہے۔ اس سے نوڈ 27 پر کال ختم ہو جاتی ہے اور کنٹرول درخت کی جڑ، نوڈ 45 پر کال پر واپس چلا جاتا ہے۔ پرنٹ (45) پر عمل کرنے کے بعد، یہ اپنے دائیں (غیر خالی) ذیلی درخت کو عبور کرنے کے لیے کال کرتا ہے۔ یہ درخت کا چوتھا اور آخری نوڈ ہے، نوڈ 62۔ یہ اپنے خالی بائیں سب ٹری پر کال کرتا ہے، پرنٹ (62) پر عمل کرتا ہے، اور اپنے خالی دائیں سب ٹری پر کال کے ساتھ ختم کرتا ہے۔ کنٹرول روٹ نوڈ، 45 پر کال کے ذریعے بیک اپ جاتا ہے، اور اس سے درخت کا مکمل راستہ ختم ہو جاتا ہے۔

پری آرڈر اور پوسٹ آرڈر ٹراورسلز

دوسرے دو ٹراورسل آرڈرز ایک جیسے ہیں: صرف نوڈ پر جانے کا سلسلہ تبدیل ہوتا ہے۔ پری آرڈر ٹراورسل کے لیے، نوڈ کو پہلے وزٹ کیا جاتا ہے، اور پوسٹ آرڈر کے لیے، اسے آخری بار دیکھا جاتا ہے۔ دو ذیلی درختوں کو ہمیشہ ایک ہی ترتیب میں دیکھا جاتا ہے: بائیں اور پھر دائیں شکل 8-13 اسی چار نوڈ درخت پر پری آرڈر ٹراورسل کے عمل کو ظاہر کرتا ہے جیسا کہ شکل 8-12 میں ہے۔ پرنٹ (فنکشن کا عمل دو ذیلی درختوں کو دیکھنے سے پہلے ہوتا ہے۔ اس کا مطلب ہے کہ پری آرڈر ٹراورسل 62، 45، 27، 16، 62 کے مقابلے میں 45، 27، 16، 62 پر پرنٹ کرے گا۔ بڑا ہے۔



شکل 8-13 ایک چھوٹے سے درخت کا پری آرڈر کریں۔

356 باب 8 بائری درخت

پیتھون کوڈ فار ٹراورسنگ آئیے اب ان آرڈر ٹراورسل کو لاگو کرنے کا ایک آسان طریقہ دیکھتے ہیں۔ جیسا کہ آپ نے ڈھیروں، قطاروں، منسلک فہرستوں، اور دیگر ڈیٹا ڈھانچے میں دیکھا ہے، traversal کی وضاحت ایک دلیل کے طور پر پاس کردہ فنکشن کا استعمال کرتے ہوئے کرنا سیدھا ہے جو ڈھانچے میں محفوظ ہر آئٹم پر لاگو ہوتا ہے۔ درختوں کے ساتھ دلچسپ فرق یہ ہے کہ تکرار اسے بہت کمپیٹ بناتی ہے۔

چونکہ ان درختوں کی نمائندگی دو کلاسز، BinarySearchTree اور Node کے ذریعے کی گئی ہے، آپ کو ایسے طریقوں کی ضرورت ہے جو دونوں قسم کی اشیاء پر کام کر سکیں۔ فہرست 5-8 میں، BinarySearchTree کا inOrderTraverse() طریقہ BinarySearchTree آجیکٹ پر ٹراورسل کو بینڈل کرتا ہے۔ یہ ٹراورسل کے لیے عوامی انٹرفیس کے طور پر کام کرتا ہے اور ذیلی درختوں پر اصل کام کرنے کے لیے ایک نجی طریقہ inOrderTraverse() کو کال کرتا ہے۔ یہ روٹ نوڈ کو پرائیویٹ طریقہ میں منتقل کرتا ہے اور واپس آتا ہے۔

فہرست سازی (inOrderTraverse) کا 8-5 بار بار عمل درآمد

کامیابی سے inOrderTraverse() (آجیکٹ):

...

```
def inOrderTraverse(
    خود، فنکشن = پرپرنٹ):
    self._inOrderTraverse(
        __root, function=function) # root node
```

```
def _inOrderTraverse(
    خود، نوڈ، فنکشن):
    اگر نوڈ:
        self._inOrderTraverse(
            function)
        self._inOrderTraverse(node.rightChild,
            function) function(node)
        node.leftChild,
```

نجی طریقہ اپنے نوڈ پیرامیٹر کے لیے ایک Node__آجیکٹ (یا None کی توقع کرتا ہے اور نوڈ پر جڑی ذیلی درخت پر تین مراحل انجام دیتا ہے۔ سب سے پہلے، یہ نوڈ کو چیک کرتا ہے اور فوری طور پر واپس آتا ہے اگر یہ None نہیں ہے کیونکہ یہ خالی ذیلی درخت کی نشاندہی کرتا ہے۔ جائز نوڈس کے لیے، یہ نوڈ کے بائیں چائلڈ پر کارروائی کرنے کے لیے سب سے پہلے اپنے آپ کو ایک بار کال کرتا ہے۔ دوسرا، یہ نوڈ پر فنکشن کو پکار کر اس کا دورہ کرتا ہے۔ تیسرا، یہ نوڈ کے دائیں بچے پر کارروائی کرنے کے لیے ایک بار کال کرتا ہے۔ بس اتنا ہی ہے۔

ٹراورسل کے لیے جنریٹر کا استعمال (inOrderTraverse) طریقہ سیدھا ہے، لیکن اس میں کم از کم تین خامیاں ہیں۔

سب سے پہلے، دوسرے آرڈرنگ کو لاگو کرنے کے لیے، آپ کو یا تو مزید طریقہ لکھنے ہوں گے یا ایک ایسا پیرامیٹر شامل کرنے کی ضرورت ہوگی جو انجام دینے کے لیے ترتیب کی وضاحت کرے۔

دوسرا، ہر نوڈ کو "وزٹ" کرنے کے لیے ایک دلیل کے طور پر منظور ہونے والے فنکشن کو Node__آجیکٹ کو بطور دلیل لینے کی ضرورت ہے۔ یہ BinarySearchTree کے اندر ایک نجی کلاس ہے جو نوڈس کو کال کرنے والے کے ذریعے بھرا پھیری سے بچاتی ہے۔ ایک متبادل جو Node__آجیکٹ کا حوالہ دینے سے گریز کرتا ہے وہ صرف ڈیٹا فیلڈ (اور شاید کلیدی فیلڈ) میں گزرنا ہوگا۔

وزٹ فنکشن کے دلائل کے طور پر ہر نوڈ کا۔ اس نقطہ نظر سے یہ کم ہو جائے گا کہ کال کرنے والا نوڈ کے ساتھ کیا کر سکتا ہے اور اسے دوسرے نوڈ حوالوں کو تبدیل کرنے سے روک دے گا۔

تیسرا، ہر نوڈ پر انجام دینے کے عمل کو بیان کرنے کے لیے فنکشن کا استعمال کرنے کی اپنی حدود ہیں۔ عام طور پر، فنکشنز ہر بار ایک ہی آپریشن کرتے ہیں جب ان کو پکارا جاتا ہے اور پچھلی کالوں کی تاریخ کے بارے میں نہیں جانتے ہیں۔ درخت کی طرح ڈیٹا سٹرکچر کے ٹراورسل کے دوران، پچھلے نوڈ وزٹ کے نتائج کو استعمال کرنے کے قابل ہونا ممکنہ کارروائیوں کو ڈرامائی طور پر بڑھا دیتا ہے۔ یہاں کچھ مثالیں ہیں جو آپ کرنا چاہتے ہیں:

□□ ہر نوڈ پر ایک مخصوص فیلڈ میں تمام اقدار کو شامل کریں۔

□□ ہر نوڈ سے فیلڈ میں تمام منفرد تاروں کی فہرست حاصل کریں۔

□□ نوڈ کی کلید کو کچھ فہرست میں شامل کریں اگر پہلے ملاحظہ کیے گئے نوڈس میں سے کسی کی بھی کسی فیلڈ میں قدر زیادہ نہیں ہے۔

ان تمام ٹراورسلز میں، ٹراورسل کے دوران کہیں بھی نتائج جمع کرنے کے قابل ہونا بہت آسان ہے۔ فنکشنز کے ساتھ ایسا کرنا ممکن ہے، لیکن جنریٹر اسے آسان بناتے ہیں۔ ہم نے باب 5 میں جنریٹرز متعارف کرائے ہیں، اور چونکہ درخت ان ڈھانچوں کے ساتھ بہت سی مماثلت رکھتے ہیں، اس لیے وہ درختوں کو عبور کرنے کے لیے بہت مفید ہیں۔

ہم ان کوتاہیوں کو ٹراورس میتھڈ، `traverse_rec()` کے ایک بار بار آنے والے جنریٹر ورژن میں دور کرتے ہیں، جو فہرست 6-8 میں دکھایا گیا ہے۔ یہ ورژن کوڈ میں کچھ پیچیدگی کا اضافہ کرتا ہے لیکن ٹراورسل کا استعمال بہت آسان بنا دیتا ہے۔ سب سے پہلے، ہم ایک پیرامیٹر، `traverse_rec()`، `traverseType` طریقہ میں شامل کرتے ہیں تاکہ ہمیں تین الگ الگ ٹراورس روٹینز کی ضرورت نہ ہو۔ پہلا اگر بیان اس بات کی تصدیق کرتا ہے کہ یہ پیرامیٹر تین معاون ترتیبوں میں سے ایک ہے: پری، ان، اور پوسٹ۔ اگر نہیں، تو یہ ایک استثناء پیدا کرتا ہے۔ بصورت دیگر، یہ روٹ نوڈ سے شروع ہونے والے ریکرسیو پرائیویٹ طریقہ، `traverse()` کا آغاز کرتا ہے، بالکل اسی طرح جیسے `inOrderTraverse()` کرتا ہے۔

`traverse()` طریقہ کو کال کرنے میں ایک اہم لیکن لطیف نکتہ نوٹ کرنا ہے۔ عوامی `traverse_rec()` طریقہ نجی `traverse()` طریقہ کو کال کرنے کا نتیجہ واپس کرتا ہے اور اسے صرف ذیلی روٹین کے طور پر نہیں کہتا ہے۔ وجہ یہ ہے کہ `traverse()` طریقہ خود جنریٹر نہیں ہے۔ اس میں کوئی پیداواری بیانات نہیں ہیں۔ اسے `traverse()` پر کال کے ذریعہ تیار کردہ ایٹریٹر کو واپس کرنا ہوگا، جسے `traverse_rec()` نوڈس پر عادی کرنے کے لیے استعمال کرے گا۔

`traverse()` طریقہ کے اندر، اگر بیانات کا ایک سلسلہ ہے۔ پہلا بیس کیس کی جانچ کرتا ہے۔ اگر نوڈ کوئی نہیں ہے، تو یہ ایک خالی درخت (یا ذیلی درخت) ہے۔ یہ اس بات کی نشاندہی کرنے کے لیے واپس آتا ہے کہ تکرار کرنے والا اختتام کو پہنچ گیا ہے (جسے ازگر `StopIteration` استثناء میں تبدیل کرتا ہے)۔ اگلا اگر بیان چیک کرتا ہے کہ آیا ٹراورسل قسم پہلے سے آرڈر ہے، اور اگر یہ ہے، تو اس سے نوڈ کی کلید اور ڈیٹا حاصل ہوتا ہے۔ یاد رکھیں کہ ایٹریٹر کو اس وقت روک دیا جائے گا جب کہ کنٹرول واپس اس کے کالر کے پاس جاتا ہے۔ اسی جگہ نوڈ کو "وزٹ کیا جائے گا" پروسیسنگ مکمل ہونے کے بعد، کالر کا لوپ اگلا نوڈ حاصل کرنے کے لیے اس تکرار کرنے والے کو طلب کرتا ہے۔ تکرار کنندہ تمام سیاق و سباق کو یاد رکھتے ہوئے، پیداوار کے بیان کے فوراً بعد دوبارہ کارروائی شروع کرتا ہے۔

جب تکرار کرنے والا دوبارہ شروع ہوتا ہے (یا اگر آرڈر پری آرڈر کے علاوہ کچھ تھا) تو اگلا مرحلہ لوپ کے لیے ہوتا ہے۔ یہ بائیں سب ٹری کے ٹراورسل کو انجام دینے کے لیے ایک بار بار چلنے والا جنریٹر ہے۔

یہ اسی `traverseType` کا استعمال کرتے ہوئے نوڈ کے لیفت چائلڈ پر `traverse()` طریقہ کو کال کرتا ہے۔ یہ اس ذیلی درخت میں نوڈس پر کارروائی کرنے کے لیے اپنا ایٹریٹر بناتا ہے۔ جیسا کہ نوڈس حاصل کیے جاتے ہیں

واپس کلید کے طور پر، ڈیٹا کے جوڑے، یہ اعلیٰ سطحی تکرار کرنے والا انہیں واپس اپنے کالر کو دیتا ہے۔ یہ لوپ کنسٹرکشن ایئرٹروں کا ایک نیسٹڈ اسٹیک تیار کرتا ہے، جیسا کہ شکل 8-12 میں دکھایا گیا ہے (traverse_o_ کے نیسٹڈ انوکیشنز کی طرح ہے۔ جب ہر تکرار کرنے والا اپنے کام کے اختتام پر واپس آتا ہے، تو یہ ایک StopIteration پیدا کرتا ہے۔ منسلک تکرار کرنے والا ہر استثنا کو پکڑتا ہے، لہذا مختلف سطحیں ایک دوسرے میں مداخلت نہیں کرتی ہیں۔

فہرست سازی 6-8 ٹراورسل کے لیے تکراری جنریٹر

```

#بانئری سرچ ٹری کلاس
class BTreeSyranier(ABC):
    ...

#پری، ان، یا پوسٹ آرڈر میں بار بار درخت کو عبور کریں۔
def traverse(self, node, traverseType="in"):
    #ترaverseType in [ 'pre', 'in', 'post' ] خود واپسی
    #ترaverseType کی قسم ایک قبول شدہ قدر ہے اور #درخت پر چلنے کے لیے جنریٹر کا
    #استعمال کریں #حاصل کرنے والے (کلید، ڈیٹا) جوڑے #جڑ سے شروع

    #انگریزی: "نامعلوم ٹراورسل قسم"
    + str(traverseType)

#بار بار چلنے والا جنریٹر عبور کرنے کے لیے #ذیلی درخت کی جڑیں نوڈ پر پری، ان، یا #
#پوسٹ آرڈر میں
#اگر ذیلی درخت خالی ہے تو #ٹراورسل کیا جاتا ہے۔

#پری آرڈر کے لیے، موجودہ #نوڈ کو #traverseType "جاری" کریں۔
#پیداوار (node.key, node.data)
#چائلڈ کی کے لیے، خود میں چائلڈ ڈیٹا
#node.leftChild, traverseType): #
#اس کے نوڈس حاصل کرنا
#اگر ترتیب میں ہے، تو اب موجودہ #نوڈ حاصل کریں۔

#چائلڈ کی کے لیے، خود میں چائلڈ ڈیٹا
#node.rightChild, traverseType):# traverse right subtree # yielding its nodes #
#پیداوار (چائلڈ کی، چائلڈ ڈیٹا)
#اگر traverseType == "post":
#پیداوار (node.key, node.data)

```

باقی traverse() طریقہ سیدھا ہے۔ بائیں سب ٹری میں تمام نوڈس پر لوپ ختم کرنے کے بعد، اگلا اگر اسٹیٹمنٹ ان آرڈر ٹراورسل قسم کی جانچ کرتا ہے اور نوڈ کی کلید اور ڈیٹا حاصل کرتا ہے، اگر یہ ترتیب ہے۔ نوڈ کو ان آرڈر ٹراورسل کے لیے بائیں اور دائیں ذیلی درختوں کے درمیان پروسیس کیا جاتا ہے۔ اس کے بعد، دائیں سب ٹری کو اس کے اپنے لوپ میں پروسیس کیا جاتا ہے، جس سے ہر وزٹ کیے گئے نوڈس واپس اس کے کالر کو ملتے ہیں۔ صحیح ذیلی درخت کے مکمل ہونے کے بعد، پوسٹ آرڈر ٹراورسل کے لیے ایک چیک اس بات کا تعین کرتا ہے کہ آیا اس مرحلے پر نوڈ کو حاصل کیا جانا چاہیے یا نہیں۔ اس کے بعد، traverse() جنریٹر ہو جاتا ہے، اس کے کالر کا لوپ ختم ہوتا ہے۔

جنریٹر کو موثر بنانا

تکراری جنریٹر میں ساختی سادگی کا فائدہ ہے۔ بیس کیسز اور بار بار آنے والی کالیں درخت کے نوڈ اور چائلڈ سٹرکچر کی پیروی کرتی ہیں۔ پروٹو ٹائپ تیار کرنا اور اس ڈھانچے سے قدرتی طور پر اس کے صحیح رویے کے بہاؤ کو ثابت کرنا۔

جنریٹر، تاہم، عملدرآمد میں کچھ ناکارہ ہے۔ $traverse()$ طریقہ کی ہر درخواست میں دو لوپس شامل ہوتے ہیں: ایک بائیں کے لیے اور ایک دائیں کے لیے۔ ان میں سے ہر ایک لوپ ایک نیا ایٹریٹر بناتا ہے تاکہ آئٹمز کو ان کے ذیلی درختوں سے واپس حاصل کرنے کے لیے $traverse()$ طریقہ کی اس درخواست کے ذریعے تخلیق کیا گیا ہو۔ تکرار کرنے والوں کی یہ تہہ جڑ سے نیچے ہر لیف نوڈ تک پھیلی ہوئی ہے۔

درخت میں N اشیاء کو عبور کرنے میں $O(N)$ وقت لگنا چاہئے، لیکن ہر پتے تک جڑ سے نیچے تک تکرار کرنے والوں کا ایک ڈھیر بنانا پیچیدگی کو بڑھاتا ہے جو پتوں کی گہرائی کے متناسب ہے۔ پتے $O(\log N)$ کی گہرائی میں ہیں، بہترین صورت میں۔ اس کا مطلب ہے کہ N اشیاء کے مجموعی طور پر گزرنے میں $O(N \times \log N)$ وقت لگے گا۔

$O(N)$ وقت حاصل کرنے کے لیے، آپ کو باب 6 کے آخر میں زیر بحث طریقہ کو لاگو کرنے کی ضرورت ہے اور کارروائی کی جا رہی اشیاء کو رکھنے کے لیے اسٹیک کا استعمال کرنا ہوگا۔ آئٹمز میں نوڈ کے ڈھانچے اور (کلید، ڈیٹا) جوڑے شامل ہوتے ہیں جو نوڈس پر محفوظ کیے جاتے ہیں تاکہ ایک خاص ترتیب میں گزرے جائیں۔
7-8 کی فہرست کوڈ دکھاتا ہے۔

غیر تکراری طریقہ تکراری نقطہ نظر کے دو حصوں کو ایک واحد $traverse()$ طریقہ میں جوڑتا ہے۔ ٹراورسل قسم کی درستگی کے لیے وہی چیک شروع میں ہوتا ہے۔ اگلا مرحلہ باب (LinkStack) 5 ماڈیول میں بیان کردہ) سے منسلک فہرست پر بنائے گئے Stack کلاس کا استعمال کرتے ہوئے ایک اسٹیک بناتا ہے۔

ابتدائی طور پر، طریقہ درخت کے جڑ نوڈ کو اسٹیک پر دھکیلتا ہے۔ اس کا مطلب ہے کہ باقی کام پورے درخت کو جڑ سے شروع کرنا ہے۔ اس کے بعد آنے والا $while$ لوپ باقی کام کے ذریعے اس وقت تک کام کرتا ہے جب تک کہ اسٹیک خالی نہ ہو جائے۔

جب لوپ سے گزرتے ہیں تو، اسٹیک کی سب سے اوپر والی چیز پاپ آف ہوجاتی ہے۔ اسٹیک پر تین قسم کی اشیاء ہوسکتی ہیں: ایک نوڈ آئٹم، ایک (کلید، ڈیٹا) ٹوپل، یا کوئی نہیں۔ مؤخر الذکر اس صورت میں ہوتا ہے جب درخت خالی ہو اور جب وہ لیف نوڈس پر کارروائی کرتا ہے (اور ان کے بچوں کو کوئی نہیں ہوتا ہے)۔

اگر اسٹیک کا اوپری حصہ نوڈ آئٹم ہے تو، $traverse()$ طریقہ اس بات کا تعین کرتا ہے کہ درخواست کردہ ٹراورسل آرڈر کی بنیاد پر نوڈ کے ڈیٹا اور اس کے بچوں کو کیسے پروسیس کیا جائے۔ یہ آئٹمز کو اسٹیک پر دھکیلتا ہے تاکہ اس کے بعد کے گزرنے والے گزرگاہوں پر عمل کیا جا سکے۔ چونکہ آئٹمز اسٹیک سے الٹے آرڈر میں پاپ ہو جائیں گے جس طرح سے انہیں اس پر دھکیل دیا گیا تھا، یہ پوسٹ آرڈر ٹراورسل کے کیس کو سنبھالنے سے شروع ہوتا ہے۔

پوسٹ آرڈر میں، پہلا آئٹم جو دھکیلا گیا ہے وہ نوڈز (کلید، ڈیٹا) ٹیپل ہے۔ چونکہ اسے پہلے دھکیل دیا گیا ہے، اس پر مجموعی طور پر آخری کارروائی ہوگی۔ اگلی چیز جو دھکیل دی گئی وہ نوڈ کا دائیں بچہ ہے۔ پوسٹ آرڈر میں، یہ نوڈ کے ڈیٹا پر کارروائی کرنے سے پہلے گزر جاتا ہے۔ دوسرے آرڈرز کے لیے، صحیح بچہ ہمیشہ آخری نوڈ ہوتا ہے۔

دائیں بچے کو دبانے کے بعد، اگلا اگر بیان چیک کرتا ہے کہ آیا ان آرڈر ٹراورسل کی درخواست کی گئی تھی۔ اگر ایسا ہے تو، یہ دو چائلڈ نوڈس کے درمیان پروسیسڈ ہونے کے لیے اسٹیک پر نوڈز (کی، ڈیٹا) ٹیپل کو دھکیل دیتا ہے۔ اس کے بعد بائیں بچے کو پروسیسنگ کے لیے اسٹیک پر دھکیلتا ہے۔

360 باب 8 بائنری درخت

فہرست سازی 7-8 غیر تکراری ٹراورس (جنریٹر)

* LinkStack درآمد سے

بائنری سرچ ٹری کلاس

class eerThcraeSyranib (نچیکٹ):

...

def traverse (self, traverseType='in'):

درخت کو عبور کرنے کے لیے # غیر تکراری جنریٹر پہلے، اندر یا پوسٹ آرڈر میں

traverseType میں نہیں ہے: ValueError کو بڑھاؤ (

تصدیق کریں کہ ٹراورسل قسم ایک # قبول شدہ قدر ہے۔

+ str(traverseType))

= Stack() stack.push(self.__root)
stack

ایک اسٹیک بنائیں

روٹ نوڈ کو اسٹیک میں رکھیں

stack.isEmpty(): جبکہ

جبکہ اسٹیک میں کام ہے۔

if isinstance(item, self.__Node): # اگر item = stack.pop() ہے تو نوڈ

if traverseType == 'post': #

stack.push((item.key, item.data))

stack.push(item.rightChild) # traverse right child if traverseType == 'in': #

2nd stack.push((item.key,

stack.push(item.leftChild) #

traverseType == 'pre': # اگر کو عبور کریں

پری آرڈر کے لیے، آئٹم 1st stack.push((item.key, item.data)) رکھیں

ایلیف آئٹم:

دوسری غیر کوئی چیز ایک (کلیدی، قدر) جوڑی ہے جسے حاصل کیا

پیداوار آئٹم

جائے گا۔

آخر میں، آخری if اسٹیٹمنٹ چیک کرتا ہے کہ آیا پری آرڈر ٹراورسل کی درخواست کی گئی تھی اور پھر نوڈ کے ڈیٹا کو اسٹیک پر بائیں اور دائیں بچوں سے پہلے پروسیسنگ کے لیے دھکیلتا ہے۔

یہ وائل لوپ کے ذریعے اگلے پاس کے دوران پاپ آف ہو جائے گا۔ یہ نوڈ آئٹم کے تمام کام کو مکمل کرتا ہے۔

حتمی ایلیف اسٹیٹمنٹ اسٹیک پر ایک غیر None آئٹم کے لیے چیک کرتا ہے، جو ایک (کلیدی، ڈیٹا) ٹیپل ہونا چاہیے۔ جب لوپ کو ایسا ٹیپل مل جاتا ہے، تو یہ اسے کال کرنے والے کو واپس دیتا ہے۔ پیداوار کا بیان اس بات کو یقینی بناتا ہے کہ () traverse طریقہ جنریٹر بن جائے، فنکشن نہیں۔

لوپ میں None قدروں کا کوئی واضح بینڈنگ نہیں ہے جو خالی روٹ اور چائلڈ لنکس کے لیے اسٹیک پر دھکیل دیا جاتا ہے۔ وجہ یہ ہے کہ ان کے لیے کرنے کے لیے کچھ نہیں ہے: بس انہیں اسٹیک سے باہر کریں اور باقی کام پر جاری رکھیں۔

اسٹیک کا استعمال کرتے ہوئے، آپ نے اب ایک O(N) جنریٹر بنایا ہے۔ درخت کے ہر نوڈ کو بالکل ایک بار دیکھا جاتا ہے، اسٹیک پر دھکیل دیا جاتا ہے، اور بعد میں پاپ آف ہوجاتا ہے۔ اس کے کلیدی ڈیٹا کے جوڑے اور چائلڈ لنکس کو بھی دھکیل دیا جاتا ہے اور بالکل ایک بار پاپ آف کیا جاتا ہے۔ نوڈ وزٹ اور چائلڈ لنکس کی ترتیب درخواست کردہ ٹراورسل آرڈرنگ کی پیروی کرتی ہے۔ اسٹیک کا استعمال کرتے ہوئے اور احتیاط سے ریورس کرنا

اس پر ڈالی گئی اشیاء کوڈ کو سمجھنے کے لیے قدرے پیچیدہ بناتی ہیں لیکن کارکردگی کو بہتر بناتی ہیں۔

ٹراورسنگ کے لیے جنریٹر کا استعمال

جنریٹر ایروج (دونوں تکراری اور اسٹیک پر مبنی) کالر کے لوپس کو آسان بناتا ہے۔ مثال کے طور پر، اگر آپ کسی درخت میں موجود تمام اشیاء کو جمع کرنا چاہتے ہیں جن کا ڈیٹا اوسط ڈیٹا ویلیو سے کم ہے، تو آپ دو لوپس استعمال کر سکتے ہیں:

کل، شمار، 0 = کلید کے لیے `random_tree.traverse('pre')` میں ڈیٹا:

کل += ڈیٹا

شمار += 1

اوسط = کل / گنتی = [] `below_average.append((key, data))` : اوسط: `random_tree.traverse('in')` اگر ڈیٹا <= اوسط:

پہلا لوپ `random_tree` میں آئٹمز کی تعداد کو شمار کرتا ہے اور ان کے ڈیٹا کی قدروں کو جمع کرتا ہے۔ دوسرا لوپ ان تمام آئٹمز کو تلاش کرتا ہے جن کا ڈیٹا اوسط سے کم ہے اور کلید اور ڈیٹا کے جوڑے کو نیچے کی اوسط فہرست میں شامل کرتا ہے۔ چونکہ دوسرا لوپ ان آرڈر میں ہوتا ہے، نیچے اوسط میں کیز صعودی ترتیب میں ہوتی ہیں۔ کسی فنکشن ہاڈی سے باہر کچھ عالمی (یا غیر مقامی) متغیرات کی وضاحت کیے بغیر نتائج جمع کرنے والے متغیرات کا حوالہ دینے کے قابل ہونا—کل، شمار، اور نیچے اوسط—جنریٹر کا استعمال ٹراورسل کے لیے بہت آسان بناتا ہے۔

ہائری سرچ ٹری ویژولائزیشن ٹول کے ساتھ ٹریورسنگ آپ کو جنریٹرز کا استعمال کرنے ہوئے ٹراورسل کی تفصیلات دریافت کرنے کی اجازت دیتا ہے۔ آپ پری آرڈر ٹراورس، ان آرڈر ٹریورس، یا پوسٹ آرڈر ٹریورس ہٹن کو منتخب کر کے تین قسم کے ٹراورسلز میں سے کوئی بھی لانچ کر سکتے ہیں۔ ہر صورت میں، ٹول فارم کا ایک سادہ لوپ چلاتا ہے۔

کلید کے لیے، ٹری ٹراورس میں ڈیٹا ("پری"): پرنٹ (کلید)

تفصیلات دیکھنے کے لیے، اسٹیک ہٹن کا استعمال کریں (ہٹن کو منتخب کرتے وقت آپ شفٹ کی کو دبا کر اسٹیک موڈ میں آپریشن شروع کر سکتے ہیں)۔ کوڈ ونڈو میں، آپ کو پہلے مختصر ٹراورسل لوپ نظر آتا ہے۔ مثال (`traverse`) طریقہ کو کال کرتی ہے تاکہ ایک لوپ میں تمام کیز اور ڈیٹا کو دیکھیں جیسے کہ پری جیسے آرڈرز میں سے ایک۔

شکل 8-14 پری آرڈر ٹراورسل کے آغاز کے قریب ایک سنیپ شاٹ دکھاتا ہے۔ (`traverse`) طریقہ کا کوڈ نیچے دائیں طرف ظاہر ہوتا ہے۔ کوڈ کے اوپر درخت کے دائیں طرف، اسٹیک دکھایا گیا ہے۔ 59 اور 94 کیز پر مشتمل نوڈس اسٹیک پر ہیں۔ اسٹیک کے اوپری حصے کو پہلے ہی پاپ آف کر دیا گیا تھا اور آئٹم لیبل کے نیچے اوپر دائیں طرف منتقل کر دیا گیا تھا۔ یہ کلید، 77 کوما کے ساتھ اس کے رنگین مستطیل سے الگ کرتا ہے (کلید، ڈیٹا) ٹیبل کی نمائندگی کرنے کے لیے جیسے اسٹیک پر دھکیل دیا گیا تھا۔ پیداوار کے بیان پر روشنی ڈالی گئی ہے، جس سے ظاہر ہوتا ہے کہ (`traverse`) iterator اور ڈیٹا کالر کو واپس دینے والا ہے۔ لوپ جیسے (`traverse`) کہا جاتا ہے کوڈ ڈسپلے کو بند کر دیا گیا ہے لیکن اگلے مرحلے پر دکھایا جائے گا۔

traverseType: 'pre' item 77

BinarySearchTree

stack

Operations

Insert		Pre-order Traversal
Search	50	In-order Traversal
Delete		Post-order Traversal
Erase & Random Fill		

```

item = stack.pop()
if isinstance(item, self._Node):
    if traverseType == 'post':
        stack.push(item.key, item.data)
    if traverseType == 'in':
        stack.push(item.key, item.data)
    stack.push(item.leftChild)
    if traverseType == 'pre':
        stack.push(item.key, item.data)
    elif item:
        yield item

```

Animation speed: slow fast

شکل 8-14 traverse() iterator استعمال کرتے ہوئے پری آرڈر میں درخت کو عبور کرنا

جب کنٹرول کالنگ لوپ پر واپس آجاتا ہے، traverse() کوڈ ونڈو سے غائب ہوجاتا ہے اور اسی طرح اسٹیک بھی، جیسا کہ شکل 8-15 میں دکھایا گیا ہے۔ کلید اور ڈیٹا متغیر اب 77 اور روٹ نوڈ کے ڈیٹا کے پابند ہیں۔ پرنٹ اسٹیٹمنٹ کو بائی لائنٹ کیا گیا ہے کیونکہ پروگرام درخت کے نیچے والے آؤٹ پٹ باکس میں کلید کو پرنٹ کرنے والا ہے۔ اگلا مرحلہ ظاہر کرتا ہے کہ کلید 77 کو آؤٹ پٹ باکس میں کاپی کیا جا رہا ہے۔

traverseType: 'pre' key, data 77

BinarySearchTree

stack

Operations

Insert		Pre-order Traversal
Search	50	In-order Traversal
Delete		Post-order Traversal
Erase & Random Fill		

```

for key, data in tree.traverse('pre'):
    print(key)

```

Animation speed: slow fast

تصویر 8-15 لوپ جو ٹراورس (ایٹریشنر کو کال کرتا ہے۔

پرنٹ کرنے کے بعد، ٹری، ٹریورس ('پری') لوپ میں ڈیٹا فار کلید پر کنٹرول واپس آجاتا ہے۔ یہ `traverse() iterator` کو ڈسپلے پر واپس دھکیلتا ہے، اس کے ساتھ اس کے اسٹیک بھی شکل 14-8 سے ملتا چلتا ہے۔ `iterator` میں `while loop` سے پتہ چلتا ہے کہ اسٹیک خالی نہیں ہے، اس لیے یہ ٹاپ آئٹم کو پاپ کرتا ہے۔ وہ آئٹم نوڈ 59 ہے، نوڈ 77 کا بائیں چائلڈ۔ یہ عمل نوڈ 59 کے بچوں اور نوڈ کی کلید، اسٹیک پر ڈیٹا پیئر کو دبائے سے دہرایا جاتا ہے۔ اگلے لوپ کی تکرار پر، وہ ٹوپل اسٹیک سے باہر نکل جاتا ہے، اور اسے پرنٹ لوپ پر واپس لایا جاتا ہے۔

تکرار کرنے والوں کی پروسیسنگ بیان کرنے کے لیے پیچیدہ ہے، اور ویژولائزیشن ٹول تحریری تفصیل کو پڑھنے کے بجائے مختلف سطحوں اور مراحل کی پیروی کرنا آسان بنا دیتا ہے۔ متعدد نوڈس کی پروسیسنگ کے ذریعے قدم اٹھانے کی کوشش کریں، بشمول جب تکرار کنندہ کسی لیف نوڈ تک پہنچتا ہے اور اسٹیک پر `None` کو دھکیلتا ہے۔ اسٹیک تکرار کرنے والے کو نوڈس پر واپس جانے کے لیے رہنمائی کرتا ہے جن پر کارروائی ہونا باقی ہے۔

ٹراورسل آرڈر

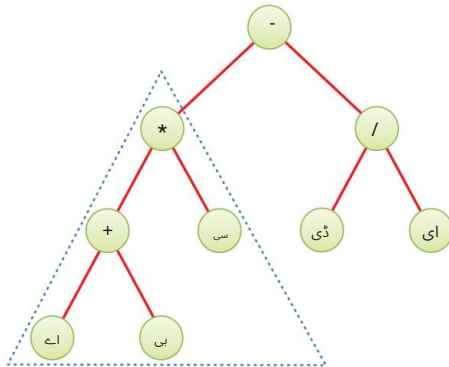
تین ٹراورسل آرڈرز کا کیا فائدہ؟ ایک فائدہ یہ ہے کہ ان آرڈر ٹراورسل بائری تلاش کے درختوں میں چاہا جانے والے صغودی ترتیب کی ضمانت دیتا ہے۔ آرڈر سے پہلے اور بعد کے سفر کے لیے الگ الگ موٹی ویشن ہے۔ یہ بہت مفید ہیں اگر آپ ایسے پروگرام لکھ رہے ہیں جو الجبری تاثرات کو پارس یا تجزیہ کرتے ہیں۔ آئیے دیکھتے ہیں کہ ایسا کیوں ہے۔

ایک بائری ٹری (بائری سرچ ٹری نہیں) کو ایک الجبری اظہار کی نمائندگی کرنے کے لیے استعمال کیا جا سکتا ہے جس میں بائری ریاضی کے آپریٹرز جیسے `+`، `-`، `*` اور `/` شامل ہوتے ہیں۔ روٹ نوڈ اور ہر نون لیف نوڈ میں ایک آپریٹر ہوتا ہے۔ لیف نوڈس یا تو متغیر نام (جیسے `A`، `B`، یا `C`) یا نمبر رکھتے ہیں۔ ہر ذیلی درخت ایک درست الجبری اظہار ہے۔

مثال کے طور پر، شکل 16-8 میں دکھایا گیا بائری درخت الجبری اظہار کی نمائندگی کرتا ہے۔

$$(A+B) * C \square D/E$$

اسے انفکس نوٹیشن کہتے ہیں۔ یہ وہ اشارے ہے جو عام طور پر الجبرا میں استعمال ہوتا ہے۔ (انفکس اور پوسٹ فکس کے بارے میں مزید جاننے کے لیے باب 4 میں سیکنشن "آرٹھمیٹک ایکسپریشنز کو پارس کرنا" دیکھیں۔) درخت کو ترتیب سے ٹریورس کرنے سے ترتیب میں درست ترتیب `A+B*C-D/E` پیدا ہوتی ہے، لیکن آپ کو داخل کرنے کی ضرورت ہے۔ آپریشن کی متوقع ترتیب حاصل کرنے کے لیے اپنے آپ کو قوسین کرتا ہے۔ نوٹ کریں کہ ذیلی درخت اپنے ذیلی تاثرات بناتے ہیں جیسے `(A+B) * C` شکل میں بیان کیا گیا ہے۔



تصویر 16-8 بائری ٹری ایک الجبری اظہار کی نمائندگی کرتا ہے۔

اس کے بائیں جانب دو ذیلی اظہارات کی تشریح کریں، جو آپریٹر کے آپرینڈ بن جاتے ہیں۔ اگر یہ ایک حرف ہے، تو یہ ایک سادہ متغیر ہے، اور اگر یہ ایک عدد ہے، تو یہ ایک مستقل ہے۔ متغیر ایبلز اور نمبرز دونوں کے لیے، آپ انہیں اظہار کے دائیں جانب سے "پاپ" کرتے ہیں اور انہیں انکلوزنگ ایکسپریشن کے عمل میں واپس کردیتے ہیں۔

ہم یہاں تفصیلات نہیں دکھاتے ہیں، لیکن آپ تصویر 16-8 میں پوسٹ فکس ایکسپریشن کو بطور ان پٹ استعمال کر کے آسانی سے درخت بنا سکتے ہیں۔ نقطہ نظر پوسٹ فکس ایکسپریشن کا جائزہ لینے کے مترادف ہے، جسے آپ نے باب 4 میں PostfixTranslate.py پروگرام اور اس سے متعلقہ InfixCalculator Visualization ٹول میں دیکھا ہے۔ اسٹیک پر آپرینڈز کو ذخیرہ کرنے کے بجائے، تاہم، آپ پورے ذیلی درختوں کو ذخیرہ کرتے ہیں۔ آپ پوسٹ فکس سٹرنگ کے ساتھ بائیں سے دائیں پڑھتے ہیں جیسا کہ آپ نے PostfixEvaluate() طریقہ میں کیا تھا۔ یہاں وہ اقدامات ہیں جب آپ کو آپرینڈ (متغیر یا نمبر) کا سامنا کرنا پڑتا ہے:

1. ایک نوڈ کے ساتھ ایک درخت بنائیں جس میں آپرینڈ ہو۔

2. اس درخت کو اسٹیک پر دھکیلیں۔

جب آپ کسی آپریٹر کا سامنا کرتے ہیں تو یہ اقدامات ہیں، O:

1. اوپرینڈ درخت R اور L کو اسٹیک سے ہٹا دیں (اسٹیک کے اوپری حصے میں سب سے دائیں طرف ہے آپرینڈ، آر)۔

2. آپریٹر، O کے ساتھ اس کی جڑ میں ایک نیا درخت T بنائیں۔

3. T کے صحیح بچے کے طور پر R کو جوڑیں۔

4. T کے بائیں بچے کے طور پر L کو جوڑیں۔

5. نتیجے میں آنے والے درخت، T کو اسٹیک پر واپس دھکیلیں۔

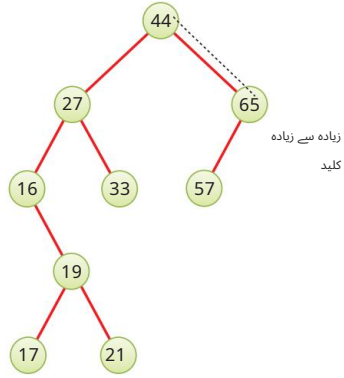
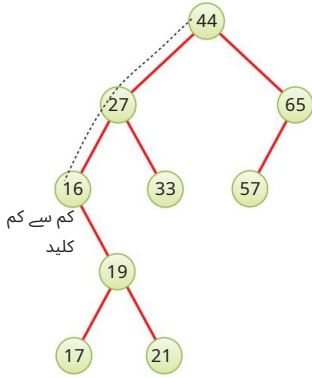
جب آپ پوسٹ فکس سٹرنگ کی جانچ کر لیتے ہیں، تو آپ ایک باقی آئٹم کو اسٹیک سے باہر کر دیتے ہیں۔ کسی حد تک حیرت انگیز طور پر، یہ آئٹم ایک مکمل درخت ہے جو الجبری اظہار کو ظاہر کرتا ہے۔ اس کے بعد آپ ہمارے بیان کردہ تین ترتیبوں میں سے کسی ایک میں درخت کو عبور کرتے ہوئے اصل پوسٹ فکس اشارے (اور پوسٹ فکس اظہار کی بازیافت) کے سابقہ اور انفکس کی نمائندگی دیکھ سکتے ہیں۔ ہم اس عمل کے نفاذ کو ایک مشق کے طور پر چھوڑتے ہیں۔

کم از کم اور زیادہ سے زیادہ کلیدی اقدار تلاش کرنا

اتفاق سے، آپ کو نوٹ کرنا چاہیے کہ بائیں سرچ ٹری میں کم از کم اور زیادہ سے زیادہ کلیدی اقدار کو تلاش کرنا کتنا آسان ہے۔ درحقیقت، یہ عمل اتنا آسان ہے کہ ہم اسے ویٹولائزیشن ٹول میں بطور آپشن شامل نہیں کرتے ہیں۔ پھر بھی، یہ سمجھنا ضروری ہے کہ یہ کیسے کام کرتا ہے۔

کم از کم، جڑ کے بائیں بچے پر جائیں؛ پھر اس بچے کے بائیں بچے کے پاس جائیں، اور اسی طرح، جب تک کہ آپ اس نوڈ پر نہ آجائیں جس میں کوئی بچہ نہیں ہے۔ یہ نوڈ کم از کم ہے۔

اسی طرح، زیادہ سے زیادہ، جڑ سے شروع کریں اور صحیح چائلڈ لنکس کی پیروی کریں جب تک کہ وہ ختم نہ ہوں۔ یہ درخت میں زیادہ سے زیادہ کلید ہوگی، جیسا کہ شکل 17-8 میں دکھایا گیا ہے۔



تصویر 8-17 بائری سرچ ٹری کی کم از کم اور زیادہ سے زیادہ کلیدی اقدار

یہاں کچھ کوڈ ہے جو کم از کم نوڈ کا ڈیٹا اور کلیدی اقدار واپس کرتا ہے:

```
# کم از کم کلید کے Node(self) اور واپس کریں۔
def findMin(self):
    if self.isEmpty():
        return None
    node = self._root
    while node.leftChild:
        node = node.leftChild
    return node
```

اگر درخت خالی ہے تو اسٹینٹن بڑھائیں اسٹینٹن ("خالی درخت میں کوئی کم از کم نوڈ نہیں")

جڑ سے شروع کریں۔

جبکہ نوڈ میں بائیں بچہ ہے،

بچے کے بائیں حوالہ کی پیروی کریں۔

واپسی # (node.key, node.data) واپسی حتمی نوڈ کلید اور ڈیٹا

زیادہ سے زیادہ تلاش کرنا اسی طرح ہے؛ صرف بائیں بچے کے لئے دائیں کو تبدیل کریں۔ آپ نوڈس کو حذف کرنے کے بارے میں اگلے حصے میں کم از کم قدر تلاش کرنے کے ایک اہم استعمال کے بارے میں سیکھتے ہیں۔

نوڈ کو حذف کرنا

نوڈ کو حذف کرنا بائری تلاش کے درختوں کے لیے درکار سب سے پیچیدہ کام ہے۔ تاہم، حذف کرنے کے بنیادی عمل کو نظر انداز نہیں کیا جا سکتا، اور تفصیلات کا مطالعہ کردار کی تعمیر کرنا ہے۔ اگر آپ کردار سازی کے موڈ میں نہیں ہیں، تو بلا جھجک Binary Search Trees سیکشن کی Efficiency پر جائیں۔

آپ اس بات کی تصدیق کرتے ہوئے شروع کرتے ہیں کہ درخت خالی نہیں ہے اور پھر جس نوڈ کو آپ حذف کرنا چاہتے ہیں اسے ڈھونڈتے ہیں، اسی نقطہ نظر کو استعمال کرتے ہوئے جو آپ نے `insert()` اور `find()` میں دیکھا تھا۔ اگر نوڈ نہیں ملا، تو آپ کا کام ہو گیا۔ جب آپ کو نوڈ اور اس کے والدین مل جاتے ہیں، تو غور کرنے کے لیے تین صورتیں ہیں:

1. جس نوڈ کو حذف کیا جانا ہے وہ ایک پتی ہے (اس کی کوئی اولاد نہیں ہے)۔

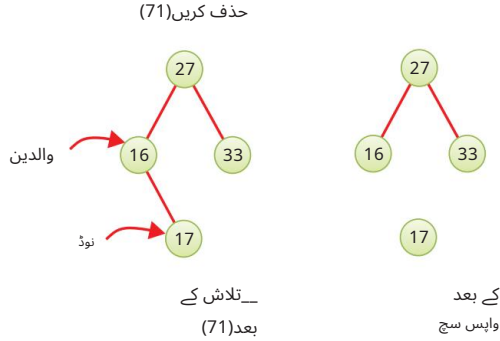
2. حذف کیے جانے والے نوڈ کا ایک بچہ ہے۔

3. حذف کیے جانے والے نوڈ کے دو بچے ہیں۔

آئیے ان تینوں صورتوں کو باری باری دیکھتے ہیں۔ پہلا آسان ہے؛ دوسرا، تقریباً اتنا ہی آسان؛ اور تیسرا، کافی پیچیدہ۔

کیس 1: جس نوڈ کو حذف کیا جائے اس کا کوئی بچہ نہیں ہے۔

لیف نوڈ کو حذف کرنے کے لیے، آپ نوڈ کے والدین میں مناسب چائلڈ فیلڈ کو نوڈ کے بجائے None میں تبدیل کر دیتے ہیں۔ نوڈ آبجیکٹ اب بھی موجود ہے، لیکن اب یہ درخت کا حصہ نہیں ہے، جیسا کہ شکل 8-18 میں نوڈ 17 کو حذف کرتے وقت دکھایا گیا ہے۔



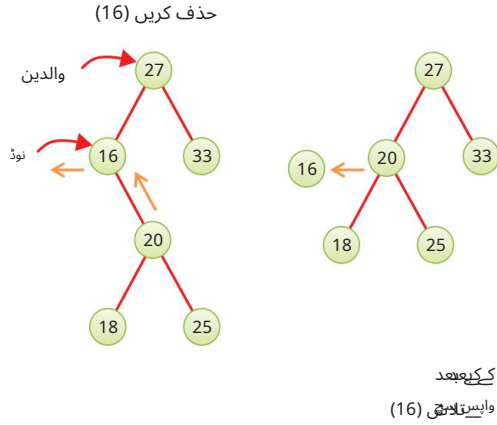
اگر آپ ازگر یا جاوا جیسی زبان استعمال کر رہے ہیں جس میں کوڑا کرکٹ جمع ہے، تو حذف شدہ نوڈ کی میموری کو دوسرے استعمال کے لیے دوبارہ حاصل کیا جائے گا (اگر آپ پروگرام میں اس کے تمام حوالوں کو ختم کر دیتے ہیں)۔ ایسی زبانوں میں جن کے لیے میموری کی واضح مختص اور ڈی ایلوکیشن کی ضرورت ہوتی ہے، حذف شدہ نوڈ کو دوبارہ استعمال کے لیے جاری کیا جانا چاہیے۔

بغیر بچوں کے نوڈ کو حذف کرنے کے لیے ویڈولائزیشن ٹول کا استعمال ہائیری سرچ ٹری ویڈولائزیشن ٹول کا استعمال کرنے کے لیے نوڈ کو حذف کرنے کی کوشش کریں۔ آپ ٹیکسٹ انٹری باکس میں نوڈ کی کلید ٹائپ کر سکتے ہیں یا اپنے پوائنٹر ڈیوائس کے ساتھ ایک لیف منتخب کر سکتے ہیں اور پھر ڈیلیٹ کو منتخب کر سکتے ہیں۔ آپ دیکھتے ہیں کہ پروگرام find() کو اس کی کلید کے ذریعے نوڈ کو تلاش کرنے کے لیے استعمال کرتا ہے، اسے ایک عارضی متغیر میں کاپی کرتا ہے، پیرنٹ لنک کو None پر سیٹ کرتا ہے، اور پھر حذف شدہ کلید اور ڈیٹا کو (اس کے رنگین پس منظر کی شکل میں) "واپس" کرتا ہے۔

کیس 2: حذف کیے جانے والے نوڈ میں ایک بچہ ہے۔

یہ دوسرا معاملہ بھی زیادہ مشکل نہیں ہے۔ نوڈ کے صرف دو کنارے ہیں: ایک اس کے والدین کی طرف اور دوسرا اس کے اکلوتے بچے کی طرف۔ آپ نوڈ کو اس کے والدین کو براہ راست اس کے بچے سے جوڑ کر اس ترتیب سے "کاٹ" کرنا چاہتے ہیں۔ اس عمل میں حذف شدہ نوڈ کے بچے کی طرف اشارہ کرنے کے لیے والدین (بائیں چائلڈ یا رائٹ چائلڈ یا _روٹ) میں مناسب حوالہ تبدیل کرنا شامل ہے۔

شکل 8-19 میں نوڈ 16 کو حذف کیا گیا ہے، جس کا صرف ایک بچہ ہے۔



تصویر 19-8 ایک بچے کے ساتھ نوڈ کو حذف کرنا

نوڈ اور اس کے والدین کو تلاش کرنے کے بعد، حذف کرنے کے طریقہ کار کو صرف ایک حوالہ تبدیل کرنا ہوگا۔ حذف شدہ نوڈ، تصویر میں کلید 16، درخت سے منقطع ہو جاتا ہے (حالانکہ اس میں اب بھی اس نوڈ کے لیے چائلڈ پوائنٹر ہو سکتا ہے جس کو پرموٹ کیا گیا تھا (کلید 20 کوڑا اٹھانے والے یہ جاننے کے لیے کافی نفیس ہیں کہ وہ حذف کیے گئے نوڈ کا دوبارہ دعویٰ کر سکتے ہیں۔ دوسرے نوڈس کے لنکس کی پیروی کیے بغیر جن کی اب بھی ضرورت ہو سکتی ہے۔

اب آئیے بغیر بچوں کے نوڈ کو حذف کرنے کے معاملے پر واپس جائیں۔ اس صورت میں، حذف کرنے کے طریقہ کار نے والدین کے چائلڈ پوائنٹر میں سے ایک کو تبدیل کرنے کے لیے ایک تبدیلی بھی کی ہے۔

اس پوائنٹر کو None پر سیٹ کیا گیا تھا کیونکہ کوئی متبادل چائلڈ نوڈ نہیں تھا۔ یہ کیس 2 کا سمی لار آپریشن ہے، لہذا آپ یہ کہہ کر کیس 1 اور کیس 2 کا ایک ساتھ علاج کر سکتے ہیں، "اگر نوڈ کو حذف کرنا ہے، D کے 0 یا 1 بچے ہیں، تو اس کے والدین میں مناسب لنک کو ہائیں بچے کے ساتھ تبدیل کریں۔ D کا، اگر یہ خالی نہیں ہے، یا D کا صحیح بچہ ہے۔" اگر D سے دونوں چائلڈ لنکس کوئی نہیں ہیں، تو آپ نے کیس 1 کا احاطہ کیا ہے۔ اگر D کے چائلڈ لنکس میں سے صرف ایک ہی غیر کوئی نہیں ہے، تو مناسب بچے کو والدین کے نئے بچے کے طور پر منتخب کیا جائے گا، جس میں کیس 2 کا احاطہ کیا جائے گا۔

آپ والدین کے بچے (یا ممکنہ طور پر root__حوالہ میں واحد بچے یا None کو فروغ دیتے ہیں۔

ایک بچے کے ساتھ نوڈ کو حذف کرنے کے لیے ویڈیو لائزیشن ٹول کا استعمال آئیے فرض کریں کہ آپ شکل 5-8 میں درخت پر ویڈیو لائزیشن ٹول استعمال کر رہے ہیں اور نوڈ 61 کو حذف کر رہے ہیں، جس کا دائیں بچہ ہے لیکن بائیں بچہ نہیں۔ نوڈ 61 پر کلک کریں اور ڈیلیٹ بٹن کو فعال کرنے ہوئے کلید ٹیکسٹ انٹری ایریا میں ظاہر ہونی چاہیے۔ بٹن کو منتخب کرنے سے find() پر ایک اور کال شروع ہوتی ہے جو کرنٹ کی طرف نوڈ اور پیرنٹ پوائنٹ نوڈ 59 کی طرف اشارہ کرنے کے ساتھ رک جاتی ہے۔

نوڈ 61 کی ایک کاپی بنانے کے بعد، اینیمیشن دکھاتی ہے کہ نوڈ 59 سے صحیح چائلڈ لنک نوڈ 61 کے رائٹ چائلڈ، نوڈ 62 پر سیٹ کیا جا رہا ہے۔ نوڈ 61 کی اصل کاپی ختم ہو جاتی ہے، اور درخت کو ایڈجسٹ کیا جاتا ہے تاکہ نوڈ پر جڑی ہوئی ذیلی درخت رکھی جا سکے۔ 62 اپنی نئی پوزیشن میں۔ آخر میں، نوڈ 61 کی کاپی نیچے والے آؤٹ پٹ باکس میں منتقل ہو جاتی ہے۔

سنگل چائلڈ نوڈس کے ساتھ نئے درخت بنانے کے لیے ویژولائزیشن ٹول کا استعمال کریں اور دیکھیں کہ جب آپ انہیں حذف کرتے ہیں تو کیا ہوتا ہے۔ ذیلی درخت کو تلاش کریں جس کی جڑ حذف شدہ نوڈ کا بچہ ہے۔ اس سے کوئی فرق نہیں پڑتا ہے کہ یہ ذیلی درخت کتنا ہی پیچیدہ ہے، اسے آسانی سے اوپر منتقل کیا جاتا ہے اور حذف شدہ نوڈ کے والدین کے نئے بچے کے طور پر پلگ ان ہوتا ہے۔

نوڈ کو حذف کرنے کے لیے پانٹھون کوڈ اب کم از کم کیسز 1 اور 2 کے کوڈ کو دیکھتے ہیں۔ 8-8 کی فہرست ڈیلیٹ() طریقہ کار کے لیے کوڈ کو ظاہر کرتی ہے، جس میں ایک دلیل لی جاتی ہے، نوڈ کو حذف کرنے کی کلید۔ یہ یا تو نوڈ کا ڈیٹا لوٹاتا ہے جسے حذف کیا گیا تھا یا کوئی نہیں، اس بات کی نشاندہی کرنے کے لیے کہ نوڈ نہیں ملا تھا۔

اس سے یہ کچھ اس طرح برتاؤ کرتا ہے جیسے کسی آئٹم کو اسٹیک سے ہٹانے یا قطار سے کسی آئٹم کو حذف کرنے کے طریقوں کی طرح۔ فرق یہ ہے کہ اعداد و شمار کے ڈھانچے میں معلوم مقام پر ہونے کے بجائے نوڈ کو درخت کے اندر پایا جانا چاہیے۔

فہرست 8-8 بائیری سرچ ٹری کا ڈیلیٹ () طریقہ

کلائنر JhFagsvranBee (بجیکٹ):

...

```
# ایک نوڈ کو حذف کریں جس کی کلید گول سے مماثل ہو۔
def حذف (خود، مقصد):
    node, parent = self._find(goal) # گول اور اس کے والدین کو تلاش کریں اگر نوڈ کوئی نہیں ہے:
    # اگر نوڈ پایا گیا تھا، # پھر پیرنٹ کے نیچے نوڈ # پر حذف کریں۔
    خود کو واپس کریں۔_حذف کریں(والدین، نوڈ)
```

```
# درخت میں مخصوص نوڈ کو حذف کریں # پیرنٹ نوڈ/ٹری میں ترمیم کرنا
def _delete (خود، والدین، نوڈ):
```

```
    # وہ ڈیٹا محفوظ کریں جسے حذف کرنا ہے۔
    # ذیلی درختوں کی تعداد کا تعین کریں۔
    # اگر دونوں ذیلی درخت موجود ہیں،
    # پھر جانشین کو # ڈیلیٹ شدہ نوڈ کو تبدیل کرنے کے لیے پرموٹ کریں۔
    # اگر صحیح بچہ نہیں ہے تو بائیں بچے کو اوپر لے جائیں۔
    # اگر والدین باخود ہیں:
    # اگر نوڈ والدین کا بائیں ہے،
    # باقی بائیں کو اپ ڈیٹ کریں:
    # دائیں بچے کو اپ ڈیٹ کریں
    دوسری:
```

```
parent.rightChild = node.leftChild
```

دوسری:

کوئی بچہ نہیں چھوڑا! لہذا صحیح بچے کو فروغ دیں

اگر والدین خود ہیں:

اگر ماں باپ پورا درخت ہے،

```
self._root = node.rightChild # update root elif parent.leftChild ہے:
    # اگر نوڈ والدین کا بائیں ہے
```

```
parent.leftChild = node.rightChild # پھر اپ ڈیٹ کریں: # بائیں چائلڈ لنک اور اپ ڈیٹ
```

```
parent.rightChild = node.rightChild # right child # حذف شدہ نوڈ کا ڈیٹا واپس کریں
```

واپس کو حذف کر دیا گیا۔

بالکل اسی طرح جیسے داخل کرنے کے لیے، پہلا قدم حذف کرنے کے لیے نوڈ اور اس کے والدین کو تلاش کرنا ہے۔ اگر اس تلاش کو گول نوڈ نہیں ملتا ہے، تو درخت سے حذف کرنے کے لیے کچھ نہیں ہے، اور حذف () کوئی نہیں لوٹاتا ہے۔ اگر حذف کرنے کے لیے نوڈ مل جاتا ہے، تو نوڈ اور اس کے والدین کو درخت میں موجود نوڈس میں ترمیم کرنے کے لیے نجی delete() طریقہ کار میں منتقل کیا جاتا ہے۔

(delete() طریقہ کے اندر، پہلا قدم یہ ہے کہ حذف کیے جانے والے نوڈ ڈیٹا کا حوالہ محفوظ کیا جائے۔ یہ مرحلہ درخت سے حوالہ جات ہٹانے کے بعد نوڈ کے ڈیٹا کی بازیافت کے قابل بناتا ہے۔ اگلا مرحلہ چیک کرتا ہے کہ نوڈ میں کتنے ذیلی درخت ہیں۔ یہ طے کرتا ہے کہ کس کیس پر کارروائی ہو رہی ہے۔ اگر بائیں اور دائیں دونوں بچے موجود ہیں، تو یہ کیس 3 ہے، اور یہ حذف کرنے کو ایک اور نجی طریقہ، promote_successor() کے حوالے کرتا ہے، جسے ہم تھوڑی دیر بعد بیان کرتے ہیں۔

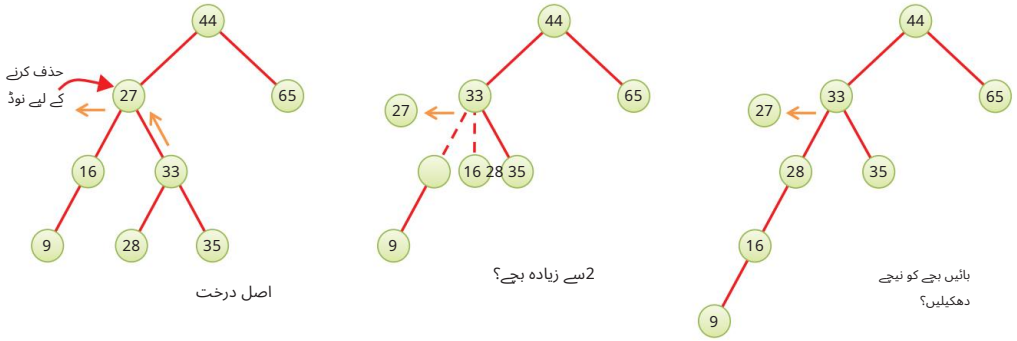
اگر حذف کرنے کے لیے نوڈ کا صرف ایک بائیں ذیلی درخت ہے، تو اگلی چیز اس کا پیرنٹ نوڈ ہے۔ اگر والدین BinarySearchTree آبیجیکٹ (self) ہے، تو حذف کرنے کے لیے نوڈ کا روٹ نوڈ ہونا چاہیے، اس لیے بائیں بچے کو روٹ نوڈ سلاٹ میں ترقی دی جاتی ہے۔ اگر پیرنٹ کا باپا بچہ حذف کرنے کے لیے نوڈ ہے، تو نوڈ کو ہٹانے کے لیے والدین کے بائیں چائلڈ کے لنک کو نوڈ کے بائیں چائلڈ سے بدل دیا جاتا ہے۔ بصورت دیگر، نوڈ کو ہٹانے کے لیے والدین کے دائیں چائلڈ لنک کو اپ ڈیٹ کیا جاتا ہے۔

یاد رکھیں کہ حوالہ جات کے ساتھ کام کرنے سے پورے ذیلی درخت کو منتقل کرنا آسان ہوجاتا ہے۔ جب نوڈ کے لیے والدین کا حوالہ آپ ڈیٹ کیا جاتا ہے، تو ترقی پانے والا بچہ واحد نوڈ یا بہت بڑا ذیلی درخت ہوسکتا ہے۔ صرف ایک حوالہ تبدیل کرنے کی ضرورت ہے۔ اگرچہ ذیلی درخت میں بہت سے نوڈس ہو سکتے ہیں، آپ کو ان کو انفرادی اتحادی میں منتقل کرنے کے بارے میں فکر کرنے کی ضرورت نہیں ہے۔ درحقیقت، وہ صرف دوسرے نوڈس کے مقابلے میں تصوراتی طور پر مختلف پوزیشنوں میں ہونے کے معنی میں "حرکت" کرتے ہیں۔ جہاں تک پروگرام کا تعلق ہے، سب ٹری کی جڑ کے لیے صرف والدین کا حوالہ تبدیل ہوا ہے، اور میموری میں باقی مواد وہی رہتا ہے۔

(delete() طریقہ کار کی حتمی دوسری شق اس معاملے سے نمٹتی ہے جب نوڈ میں کوئی بچہ نہیں ہوتا ہے۔ چاہے نوڈ کا صحیح بچہ ہے یا نہیں، delete() کو نوڈ کے دائیں بچے کی طرف اشارہ کرنے کے لیے صرف والدین کے حوالے کو اپ ڈیٹ کرنے کی ضرورت ہے۔ یہ کیس 1 اور کیس 2 دونوں کو بینڈل کرتا ہے۔ اسے اب بھی یہ طے کرنا ہوگا کہ پیرنٹ آبیجیکٹ کے کون سے فیڈ کو نوڈ کے دائیں بچے کا حوالہ ملتا ہے، بالکل اسی طرح جیسے پہلے لائنوں میں جب صرف بائیں بچہ موجود تھا۔ اس کے مطابق، یہ node.rightChild کو والدین کے root، leftChild یا rightChild فیڈ میں رکھتا ہے۔ آخر میں، یہ نوڈ کا ڈیٹا لوٹاتا ہے جسے حذف کر دیا گیا تھا۔

کیس 3: حذف کیے جانے والے نوڈ کے دو بچے ہیں۔

اب مزہ شروع ہوتا ہے۔ اگر حذف شدہ نوڈ کے دو بچے ہیں، تو آپ اسے صرف ان بچوں میں سے کسی ایک سے نہیں بدل سکتے، کم از کم اگر اس بچے کے اپنے (بڑے) بچے ہوں۔ کیوں نہیں؟ شکل 8-20 میں جانچیں اور تصور کریں کہ نوڈ 27 کو حذف کریں اور اسے اس کے دائیں ذیلی درخت سے تبدیل کریں، جس کی جڑ 33 ہے۔ آپ صحیح ذیلی درخت کو فروغ دے رہے ہیں، لیکن اس کے اپنے بچے ہیں۔ نوڈ 33 کی نئی پوزیشن میں کون سا بائیں بچہ ہوگا، حذف شدہ نوڈ کا باپا بچہ، 16 یا نوڈ 33 کا بائیں بچہ، 28؟ اور آپ دوسرے بچے کے ساتھ کیا کرتے ہیں؟ آپ اسے صرف پھینک نہیں سکتے۔

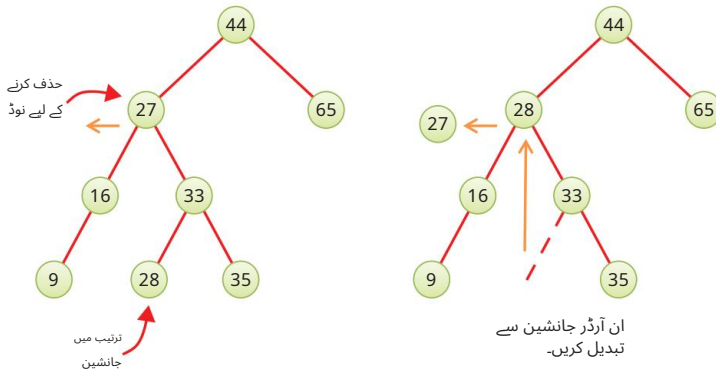


شکل 20-8 دو ذیلی درختوں کے ساتھ نوڈ کو حذف کرنے کے اختیارات

شکل 20-8 میں درمیانی آپشن تین بچوں کو ممکنہ طور پر اجازت دیتا ہے۔ اس سے بہت ساری دیگر پریشانیوں پیدا ہوں گی کیونکہ درخت اب بائٹری نہیں ہے (اس خیال پر مزید کے لیے باب 9 دیکھیں)۔ اعداد و شمار میں دائیں ہاتھ کا اختیار حذف شدہ نوڈ کے بائیں بچے، 16، کو نیچے دھکیلتے ہوئے اور نئے نوڈ کے بائیں بچے، 28، کو اس کے اوپر دھکیلتے ہوئے دکھاتا ہے۔ یہ نقطہ نظر قابل فہم لگتا ہے۔ درخت اب بھی ایک بائٹری تلاش درخت ہے، کم از کم، مسئلہ، تاہم، یہ ہے کہ کیا کرنا ہے اگر ترقی یافتہ نوڈ کے بائیں بچے کا اپنا ایک پیچیدہ ذیلی درخت ہے (مثال کے طور پر، اگر اعداد و شمار میں نوڈ 28 کے نیچے ایک پورا ذیلی درخت تھا)۔ اس کا مطلب یہ ہو سکتا ہے کہ بائیں ذیلی درختوں کو آپس میں کہاں تقسیم کرنا ہے یہ جاننے کے لیے ایک لمبا راستہ چھوڑنا ہے۔

بمیں ایک اور نقطہ نظر کی ضرورت ہے۔ اچھی خبر یہ ہے کہ ایک چال ہے۔ بری خبر یہ ہے کہ، یہاں تک کہ چال کے ساتھ، غور کرنے کے لئے خاص معاملات ہیں۔ یاد رکھیں، بائٹری سرچ ٹری میں، نوڈس کو چڑھتی ہوئی چابیاں کی ترتیب سے ترتیب دیا جاتا ہے۔ ہر نوڈ کے لیے، اگلی اعلیٰ ترین کلید کے ساتھ نوڈ کو اس کا ان آرڈر جانشین، یا محض اس کا جانشین کہا جاتا ہے۔ شکل 20-8 کے اصل درخت میں، نوڈ 28 نوڈ 27 کا ان آرڈر جانشین ہے۔

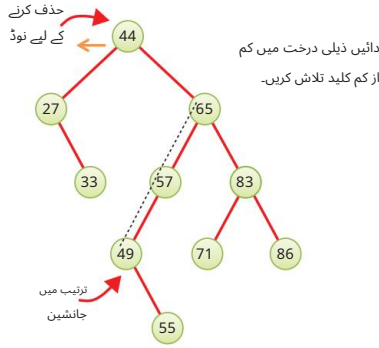
چال یہ ہے: دو بچوں والے نوڈ کو حذف کرنے کے لیے، نوڈ کو اس کے ان آرڈر جانشین سے بدل دیں۔ شکل 21-8 ایک حذف شدہ نوڈ کو اس کے جانشین سے تبدیل کر رہا ہے۔ نوٹ کریں کہ نوڈس اب بھی ترتیب میں ہیں۔ اس میں صرف ایک سادہ متبادل تھا۔ اگر جانشین کے خود بچے ہوں تو یہ تھوڑا زیادہ پیچیدہ ہو جائے گا۔ ہم ایک لمحے میں اس امکان کو دیکھتے ہیں۔



تصویر 21-8 نوڈ کو اس کے جانشین سے تبدیل کیا ہے۔

جانشین کی تلاش آپ نوڈ کے جانشین کو کیسے تلاش کرتے ہیں؟ انسان یہ کام جلدی کر سکتا ہے (چھوٹے درختوں کے لیے، ویسے بھی)۔ بس درخت پر ایک سرسری نظر ڈالیں اور حذف کیے جانے والے نوڈ کی کلید کو کم کرنے والی اگلی سب سے بڑی تعداد تلاش کریں۔ تصویر 8-21 میں یہ دیکھنے میں زیادہ وقت نہیں لگتا کہ 27 کا جانشین 28 ہے، یا یہ کہ 35 کا جانشین 44 ہے۔ تاہم، کمپیوٹر "ایک نظر میں" چیزیں نہیں کر سکتا۔ اسے الگورتھم کی ضرورت ہے۔

کم سے کم یا زیادہ سے زیادہ کلید کے ساتھ نوڈ کو تلاش کرنا یاد ہے؟ اس صورت میں آپ حذف کی جانے والی کلید سے بڑی کم از کم کلید تلاش کر رہے ہیں۔ جس نوڈ کو حذف کیا جانا ہے اس میں بائیں اور دائیں دونوں ذیلی درخت ہیں کیونکہ آپ کیس 3 پر کام کر رہے ہیں۔ لہذا، آپ صرف دائیں ذیلی درخت میں کم از کم کلید تلاش کر سکتے ہیں، جیسا کہ شکل 8-22 میں دکھایا گیا ہے۔ آپ کو صرف اس وقت تک بائیں چائلڈ لنکس پر عمل کرنے کی ضرورت ہے جب تک کہ آپ کو کوئی نوڈ نہ ملے جس میں بائیں بچے کے ساتھ کوئی بچہ نہ ہو۔

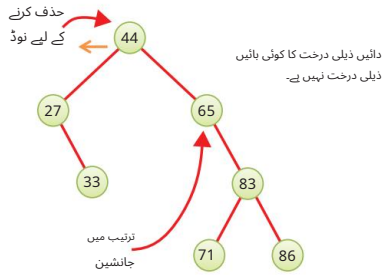


تصویر 8-22 جانشین کی تلاش

حذف کیے جانے والے نوڈ کے اوپر جڑے درختوں میں ممکنہ نوڈس کے بارے میں کیا خیال ہے؟ کیا جانشین اس میں کہیں نہیں ہو سکتا؟ آئیے اس کے ذریعے سوچتے ہیں۔ تصور کریں کہ آپ شکل 8-22 میں نوڈ 27 کے کامیاب سب سے بڑی تلاش کر رہے ہیں۔ جانشین کی عمر 27 سے زیادہ اور 33 سال سے کم ہونی چاہیے، جو اس کے صحیح بچے کی کلید ہے۔ ان دونوں اقدار کے درمیان کلید کے ساتھ کوئی بھی نوڈ نوڈ 33 کے بائیں ذیلی درخت میں کہیں داخل کیا جائے گا۔ یاد رکھیں کہ آپ ہمیشہ ہائیری سرچ ٹری کو تلاش کرتے ہیں اور درخت میں پہلے سے موجود کلیدوں کے لیے کلید کے رشتہ دار ترتیب کی بنیاد پر راستہ منتخب کرتے ہیں۔ مزید برآں، نوڈ 33 کو نوڈ 27 کے دائیں چائلڈ کے طور پر رکھا گیا تھا کیونکہ یہ روٹ نوڈ، 44 سے کم تھا۔ کسی بھی نوڈ کی رائٹ چائلڈ کلید اس کے والدین کی کلید سے کم ہونی چاہیے اگر یہ اس والدین کا بائیں بچہ ہے۔ لہذا والدین، دادا دادی، یا اس سے آگے جانا (بائیں چائلڈ لنکس کے بعد) صرف بڑی کلیدوں کی طرف جاتا ہے، اور وہ چابیاں نہیں ہوسکتی ہیں

جانشین

جانشین کے بارے میں نوٹ کرنے کے لئے ایک دو چیزیں ہیں۔ اگر حذف کرنے کے لیے اصل نوڈ کے دائیں بچے کے بائیں بچے نہیں ہیں، تو یہ دائیں بچہ خود جانشین ہے، جیسا کہ شکل 8-23 کی مثال میں دکھایا گیا ہے۔ چونکہ جانشین کے پاس ہمیشہ ایک خالی بائیں چائلڈ لنک ہوتا ہے، اس لیے اس میں زیادہ سے زیادہ ایک بچہ ہوتا ہے۔



تصویر 23-8 صحیح بچہ جانشین ہے۔

جانشین کے ساتھ تبدیل کرنا جانشین کو تلاش کرنے کے بعد، آپ آسانی سے اس کی کلید اور ڈیٹا کی قدروں کو حذف کرنے کے لیے نوڈ میں کاپی کر سکتے ہیں، لیکن جانشین نوڈ پر جڑی ذیلی درخت کے ساتھ آپ کیا کریں گے؟ آپ جانشین نوڈ کی کاپی وہاں درخت میں نہیں چھوڑ سکتے کیونکہ ڈیٹا کو دو جگہوں پر ذخیرہ کیا جائے گا، ڈپلیکیٹ کیز بنائیں گے، اور جانشین کو حذف کرنے کو مستقبل میں ایک مسئلہ بنا دیں گے۔

تو، اسے درخت سے نکالنے کا سب سے آسان طریقہ کیا ہے؟

امید ہے کہ، باب 6 پڑھنے سے جواب درست ہو جائے گا۔ اب آپ بار بار آنے والی کال کا استعمال کر کے جانشین کو درخت سے حذف کر سکتے ہیں۔ آپ جانشین پر وہی آپریشن کرنا چاہتے ہیں جو آپ اصل نوڈ پر حذف کرنے کے لیے کر رہے ہیں — جس میں گول کلید ہے۔ کیا فرق ہے کہ آپ کو صرف ایک چھوٹے درخت میں حذف کرنے کی ضرورت ہے، صحیح ذیلی درخت جہاں آپ کو جانشین ملا ہے۔ اگر آپ نے گول نوڈ کو تبدیل کرنے کے بعد درخت کی جڑ سے شروع کرنے کی کوشش کی تو، `find()` طریقہ اسی راستے پر چلے گا اور اس نوڈ پر ختم ہوگا جسے آپ نے ابھی تبدیل کیا ہے۔ جانشین کو حذف کرنے تک آپ کلید کو تبدیل کرنے میں تاخیر کر کے اس مسئلے کو حل کر سکتے ہیں، لیکن یہ بہت آسان ہے — اور زیادہ اہم بات، تیز — اگر آپ صحیح ذیلی درخت میں ایک نیا حذف آپریشن شروع کرتے ہیں۔ تلاش کرنے کے لیے بہت کم درخت ہوں گے، اور آپ غلطی سے پچھلے گول نوڈ پر نہیں پہنچ سکتے۔

درحقیقت، جب آپ نے جانشین کو تلاش کیا، تو آپ نے راستے کا تعین کرنے کے لیے چائلڈ لنکس کی پیروی کی، اور اس سے آپ کو جانشین اور جانشین کا پیرنٹ نوڈ دونوں مل گئے۔ ان دو حوالوں کی دستیابی کے ساتھ، اب آپ کے پاس پرائیویٹ `delete()` طریقہ کو کال کرنے کے لیے درکار ہر چیز موجود ہے جو فہرست 8-8 میں دکھائے گئے ہیں۔ اب آپ `promote_successor()` طریقہ کی وضاحت کر سکتے ہیں، جیسا کہ فہرست 8-9 میں دکھایا گیا ہے۔ یاد رکھیں، یہ وہ طریقہ ہے جو کیس 3 کو ہینڈل کرنے کے لیے استعمال کیا جاتا ہے — جب حذف کرنے والے نوڈ کے دو بچے ہوں۔

`promote_successor()` طریقہ اپنے واحد پیرامیٹر کے طور پر نوڈ کو حذف کرتا ہے۔

چونکہ یہ اس نوڈ کے ڈیٹا اور کلید کو تبدیل کرنے جا رہا ہے اور پھر جانشین کو حذف کر دے گا، اس تناظر میں اسے تبدیل کرنے کے لیے نوڈ کے طور پر حوالہ دینا آسان ہے۔ شروع کرنے کے لیے، یہ تبدیل کیے جانے والے نوڈ کے دائیں چائلڈ پر ایک `successor` متغیر کی نشاندہی کرتا ہے۔ بالکل `find()` طریقہ کی طرح، یہ جانشین نوڈ کے پیرنٹ کو ٹریک کرتا ہے، جسے تبدیل کرنے کے لیے نوڈ بننے کے لیے شروع کیا جاتا ہے۔

یہ `minNode()` طریقہ کی طرح کام کرتا ہے، اگر بائیں بچے ہے تو اس کے بائیں بچے کے ساتھ جانشین کو اپ ڈیٹ کرنے کے لیے ایک `while` لوپ کا استعمال کرتا ہے۔ جب لوپ باہر نکلتا ہے، جانشین نوڈ اور پیرنٹ کو اس کے پیرنٹ نوڈ کی طرف اشارہ کرتا ہے۔

فہرست سازی BinarySearchTree 9-8 کا () promote_successor__ طریقہ

کتابیں: Jhonny Syran (جیکٹ):

...

```

#دونوں ذیلی درختوں کے ساتھ نوڈ کو حذف کرنے وقت، #دائیں سب ٹری پر جانشین تلاش کریں،
#اس کا ڈیٹا اس نوڈ میں ڈالیں، اور #جانشین کو دائیں سب ٹری سے حذف کریں۔

def __promote_successor( خود،
    نوڈ): جانشین = node.rightChild
    والدین = نوڈ جیکہ: successor.leftChild

#میں جانشین کی تلاش شروع کریں۔
#دائیں ذیلی درخت اور اس کے والدین کو ٹریک کریں۔
#نیچے بائیں چائلڈ لنکس تک
#مزید بائیں لنک والے بائیں جانشین، بائیں جانشین
= successor.leftChild

node.key = successor.key
node.data = successor.data

leteled__files (والدین، جانشین) #جانشین نوڈ کو ہٹا دیں۔

```

بس اتنا کرنا باقی ہے کہ نوڈ کی کلید اور ڈیٹا کو اپ ڈیٹ کرنا ہے جسے تبدیل کیا جانا ہے اور () delete__ پر دوبارہ آنے والی کال کا استعمال کرتے ہوئے جانشین نوڈ کو حذف کرنا ہے۔ آپ کے دیکھے ہوئے پچھلے تکراری طریقوں کے برعکس، یہ اسی روٹین کے لیے کال نہیں ہے جہاں کال ہوتی ہے۔ اس صورت میں، () promote_successor__ طریقہ () delete__ کو کال کرتا ہے، جو بدلے میں () promote_suc cessor() کو کال کر سکتا ہے۔ اسے باہمی تکرار کہتے ہیں — جہاں دو یا زیادہ معمولات ایک دوسرے کو کال کرتے ہیں۔

آپ کے حواس اب کانپ رہے ہوں گے۔ آپ کیسے جانتے ہیں کہ یہ باہمی تکرار ختم ہو جائے گی؟ بنیادی کیس کہاں ہے جو آپ نے "سادہ" تکرار کے معمولات کے ساتھ دیکھا؟ کیا آپ باہمی تکراری کالوں کے لامحدود لوپ میں داخل ہو سکتے ہیں؟ اس کے بارے میں فکر کرنا اچھی بات ہے، لیکن یہاں ایسا نہیں ہونے والا ہے۔ یاد رکھیں کہ نوڈ کو حذف کرنا تین صورتوں میں ٹوٹ جاتا ہے۔ کیس 1 اور 2 ایک بچے کے ساتھ لیف نوڈس اور نوڈس کو حذف کرنے کے تھے۔ ان دو کیسز سے () promote_successor__ کالز نہیں ہوتیں، اس لیے وہ بنیادی کیسز ہیں۔ جب آپ کیس 3 کے لیے () promote_successor__ کال کرتے ہیں، تو یہ حذف کرنے کے لیے نوڈ پر جڑی ذیلی درخت پر کام کرتا ہے، اس لیے واحد موقع یہ ہے کہ درخت کو بار بار پروسیس کیا جا رہا ہے اصل سے چھوٹا نہیں ہے اگر حذف کرنے والا نوڈ روٹ نوڈ ہے۔ تاہم، کلینچر یہ ہے کہ () delete__ () promote_successor__ کو صرف جانشین نوڈس پر کال کرتا ہے — ایسے نوڈس جن پر زیادہ سے زیادہ ایک بچہ ہونے کی ضمانت دی جاتی ہے اور درخت میں کم از کم ایک سطح اس نوڈ سے کم ہوتی ہے جس پر انہوں نے شروع کیا تھا۔ وہ ہمیشہ ایک بنیادی کیس کی طرف لے جاتے ہیں اور کبھی بھی لامحدود تکرار نہیں کرتے ہیں۔

دو بچوں کے ساتھ نوڈ کو حذف کرنے کے لیے ویڑولائزیشن ٹول کا استعمال کرتے ہوئے ویڑولائزیشن ٹول کے ساتھ ایک درخت بنائیں اور دو بچوں کے ساتھ ایک نوڈ چنیں۔ اب مرد یہ پتہ لگاتے ہیں کہ کون سا نوڈ اس کا جانشین ہے، اس کے دائیں بچے کے پاس جا کر اور پھر اس دائیں بچے کے بائیں بچوں کی لائن پر عمل کرتے ہوئے (اگر کوئی ہے)۔ اپنی پہلی کوشش کے لیے، آپ یہ یقینی بنانا چاہیں گے کہ جانشین کی اپنی کوئی اولاد نہیں ہے۔ بعد کی کوششوں پر، زیادہ پیچیدہ صورتحال کو دیکھنے کی کوشش کریں جہاں ایک نوڈ کے بجائے جانشین کے پورے ذیلی درختوں کو ادھر ادھر منتقل کیا جاتا ہے۔

حذف کرنے کے لیے نوڈ کا انتخاب کرنے کے بعد، حذف کریں بٹن پر کلک کریں۔ آپ انفرادی مراحل کو ٹریک کرنے کے لیے قدم یا توقف/یلے بٹن استعمال کرنا چاہتے ہیں۔ ہم نے جو طریقہ بیان کیا ہے ان میں سے ہر ایک کوڈ ونڈو میں ظاہر ہوگا، لہذا آپ دیکھ سکتے ہیں کہ یہ کس طرح نوڈ کو حذف کرنے کا فیصلہ کرتا ہے جس کے دو بچے ہیں، جانشین کا پتہ لگاتا ہے، جانشین کلید اور ڈیٹا کو کاپی کرتا ہے، اور پھر جانشین نوڈ کو حذف کرتا ہے۔

کیا حذف کرنا ضروری ہے؟

اگر آپ یہاں تک پہنچے ہیں، تو آپ دیکھ سکتے ہیں کہ حذف کرنا کافی حد تک شامل ہے۔ درحقیقت، یہ اتنا پیچیدہ ہے کہ کچھ پروگرامرز اسے مکمل طور پر چھوڑنے کی کوشش کرتے ہیں۔ وہ Node_کلاس میں ایک نیا بولین فیلڈ شامل کرتے ہیں، جسے Deleted کہا جاتا ہے۔ نوڈ کو حذف کرنے کے لیے، انہوں نے صرف اس فیلڈ کو True پر سیٹ کیا۔ یہ ایک طرح کا "نرم" حذف ہے، جیسے کسی فائل کو صحیح معنوں میں حذف کیے بغیر کوڑے دان کے فولڈر میں منتقل کرنا۔ پھر دوسرے آپریشنز، جیسے find()، اس فیلڈ کو چیک کریں تاکہ یہ یقینی بنایا جا سکے کہ نوڈ کے ساتھ کام کرنے سے پہلے اسے حذف شدہ کے طور پر نشان زد نہیں کیا گیا ہے۔ اس طرح، نوڈ کو حذف کرنے سے درخت کی ساخت تبدیل نہیں ہوتی ہے۔ یقیناً، اس کا مطلب یہ بھی ہے کہ میموری پہلے "حذف شدہ" نوڈس سے بھر سکتی ہے۔

یہ نقطہ نظر تھوڑا سا پولیس آؤٹ ہے، لیکن یہ مناسب ہوسکتا ہے جہاں درخت میں بہت زیادہ حذف نہیں ہوں گے۔ بہت محتاط رہیں۔ اس طرح کے مفروضے آپ کو پریشان کرنے کے لئے واپس آتے ہیں۔ مثال کے طور پر، یہ فرض کرنا کہ کمپنی کے عملے کے ریکارڈ کے لیے حذف کرنا متواتر نہیں ہو سکتا ہے، پروگرامر کو Deleted فیلڈ استعمال کرنے کی ترغیب دے سکتا ہے۔ اگر کمپنی سیکڑوں سالوں تک قائم رہتی ہے تو مستقبل میں کسی وقت فعال ملازمین سے زیادہ حذف ہونے کا امکان ہے۔ یہی بات درست ہے اگر کمپنی کو ٹرن اوور کی اعلیٰ شرحوں کا سامنا کرنا پڑتا ہے، یہاں تک کہ مختصر وقت کے اندر بھی۔ یہ درختوں کی کارروائیوں کی کارکردگی کو نمایاں طور پر متاثر کرے گا۔

ثنائی تلاش کے درختوں کی کارکردگی جیسا کہ آپ نے دیکھا ہے، درختوں کے ساتھ زیادہ تر کارروائیوں میں کسی خاص نوڈ کو تلاش کرنے کے لیے درخت کو سطح سے دوسری سطح تک اتارنا شامل ہے۔ اس آپریشن میں کتنا وقت لگتا ہے؟ ہم نے پہلے ذکر کیا ہے کہ نوڈ کو تلاش کرنے کی کارکردگی $O(\log N)$ سے $O(N)$ تک ہو سکتی ہے، لیکن آئیے تفصیلات دیکھیں۔

ایک مکمل، متوازن درخت میں، تقریباً نصف نوڈس نیچے کی سطح پر ہوتے ہیں۔ زیادہ درست طور پر، ایک مکمل، متوازن درخت میں، باقی درخت کے مقابلے میں نیچے کی قطار میں بالکل ایک اور نوڈ ہوتا ہے۔ اس طرح، تمام تلاشوں یا اندراجات یا حذفوں میں سے تقریباً نصف کے لیے نچلی سطح پر نوڈ تلاش کرنے کی ضرورت ہوتی ہے۔ (تمام تلاشی کارروائیوں کے ایک چوتھائی کے لیے اگلے سے نچلی سطح پر نوڈ تلاش کرنے کی ضرورت ہوتی ہے، وغیرہ۔)

تلاش کے دوران، آپ کو ہر سطح پر ایک نوڈ کا دورہ کرنے کی ضرورت ہے۔ آپ یہ جان کر اچھی طرح اندازہ لگا سکتے ہیں کہ ان کارروائیوں کو انجام دینے میں کتنا وقت لگتا ہے یہ جان کر کہ کتنے درجے ہیں۔ ایک مکمل، متوازن درخت کو فرض کریں، جدول 1-8 دکھاتا ہے کہ دی گئی تعداد میں نوڈس کو رکھنے کے لیے کتنی سطحیں ضروری ہیں۔

نمبر بہت زیادہ ایسے ہی ہیں جیسے باب ٹیر 2 میں زیر بحث ترتیب شدہ سرنی کو تلاش کرنے کے لیے۔ اس صورت میں، ہائیری تلاش کے لیے موازنہ کی تعداد تقریباً آرے میں خلیوں کی تعداد کے بیس 2 لاگرتھم کے برابر تھی۔ یہاں، اگر آپ پہلے کالم N میں نوڈس کی تعداد، اور دوسرے کالم L میں لیولز کی تعداد کو کال کرتے ہیں، تو آپ کہہ سکتے ہیں کہ N پاور L میں 2 سے 1 کم ہے، یا

$$N = 2L - 1$$

مساوات کے دونوں اطراف میں 1 کا اضافہ کرنا، آپ کے پاس ہے۔

$$N + 1 = 2L$$

باب 2 میں جو کچھ آپ نے سیکھا ہے اس کا استعمال کرتے ہوئے لاگرتھم ایک عدد کو طاقت تک بڑھانے کے الٹا ہونے کے بارے میں، آپ دونوں اطراف کا لوگارتھم لے سکتے ہیں اور شرائط کو دوبارہ ترتیب دے سکتے ہیں۔

$$\log_2(N + 1) = \log_2(2L) = L$$

$$L = \log_2(N + 1)$$

376 باب 8 ہائری درخت

اس طرح، عام درختوں کی کارروائیوں کو انجام دینے کے لیے درکار وقت N کے بیس 2 لاگ کے متناسب ہے۔ بگ O اشارے میں، آپ کہتے ہیں کہ اس طرح کی کارروائیوں میں $O(\log N)$ وقت لگتا ہے۔

جدول 1-8 نوڈس کی مخصوص تعداد کے لیے سطحوں کی تعداد

نوٹ: چوٹی کی تعداد	
1	1
2	3
3	7
4	15
5	31
...	...
1,023	10
...	...
32,767	15
...	...
1,048,575	20
...	...
33,554,431	25
...	...
1,073,741,823	30

اگر درخت مکمل یا متوازن نہیں ہے، تو تجزیہ مشکل ہے۔ آپ کہہ سکتے ہیں کہ درجات کی دی گئی تعداد والے درخت کے لیے، غیر مکمل درخت کے لیے اوسط تلاش کا وقت پورے درخت کے مقابلے میں کم ہوگا کیونکہ کم تلاشیں نچلی سطح پر جائیں گی۔

درخت کا موازنہ دوسرے ڈیٹا اسٹوریج ڈھانچے سے کریں جن پر ہم نے اب تک بات کی ہے۔ 1,000,000 آئٹمز پر مشتمل ایک غیر یا ڈیرڈ صف میں یا ایک لنک شدہ فہرست میں، جس چیز کو آپ چاہتے ہیں اسے تلاش کرنے کے لیے، اوسطاً، 500,000 موازنہ، بنیادی طور پر 1,000,000 $O(N)$ اشیاء کے متوازن درخت میں، صرف 20 (یا اس سے کم) موازنہ کی ضرورت ہے کیونکہ یہ $O(\log N)$ ہے۔

ترتیب شدہ صف میں، آپ اتنی ہی تیزی سے ایک آئٹم تلاش کر سکتے ہیں، لیکن کسی آئٹم کو داخل کرنے کے لیے، اوسطاً، 500,000 آئٹمز کو منتقل کرنے کی ضرورت ہوتی ہے۔ 1,000,000 آئٹمز والے درخت میں کسی آئٹم کو داخل کرنے کے لیے 20 یا اس سے کم موازنہ درکار ہوتے ہیں، نیز آئٹم کو جوڑنے کے لیے تھوڑا سا وقت درکار ہوتا ہے۔ اضافی وقت مستقل ہے اور اشیاء کی تعداد پر منحصر نہیں ہے۔

اسی طرح، 1,000,000 آئٹم سرنی سے کسی آئٹم کو حذف کرنے کے لیے اوسطاً 500,000 آئٹمز کو منتقل کرنے کی ضرورت ہوتی ہے، جب کہ 1,000,000 نوڈ ٹری سے کسی آئٹم کو حذف کرنے کے لیے 20 یا اس سے کم کی ضرورت ہوتی ہے۔

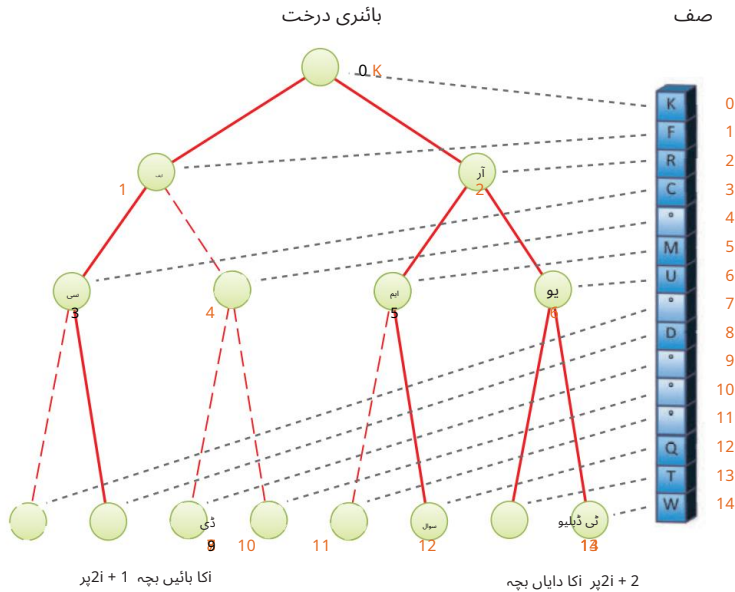
آئٹم کو تلاش کرنے کے لیے موازنہ، نیز اس کے جانشین کو تلاش کرنے کے لیے کچھ اور موازنہ، نیز اس چیز کو منقطع کرنے اور اس کے جانشین کو جوڑنے کے لیے مختصر وقت۔ چونکہ جانشین حذف کرنے کے لیے نوڈ کے مقابلے میں درخت میں کہیں کم ہے، نوڈ اور اس کے جانشین دونوں کو تلاش کرنے کے لیے موازنہ کی کل تعداد 20 یا اس سے کم ہوگی۔

اس طرح، ایک درخت ڈیٹا ذخیرہ کرنے کے تمام کاموں کے لیے اعلیٰ کارکردگی فراہم کرتا ہے: تلاش، اندراج، اور حذف کرنا۔ ٹراورسنگ دیگر آپریشنز کی طرح تیز نہیں ہے، لیکن تعریف کے مطابق، تمام N آئٹمز کا احاطہ کرنے کے لیے اسے $O(N)$ ہونا چاہیے۔ آپ نے جو بھی ڈیٹا ڈھانچہ دیکھا ہے، ان میں یہ $O(N)$ رہا ہے، لیکن ہم بعد میں کچھ دوسرے ڈیٹا ڈھانچے دکھاتے ہیں جہاں یہ زیادہ ہو سکتا ہے۔ صفوں یا فہرستوں کے مقابلے درخت کو عبور کرنے کے لیے تھوڑی زیادہ میموری کی ضرورت ہوتی ہے کیونکہ آپ کو بار بار آنے والی کالز کو اسٹور کرنے یا اسٹیک استعمال کرنے کی ضرورت ہوتی ہے۔ وہ میموری $O(\log N)$ ہوگی۔ یہ ان صفوں اور فہرستوں سے متصادم ہے جنہیں ٹراورسل کے دوران صرف $O(1)$ میموری کی ضرورت ہوتی ہے۔

درختوں کو صفوں کے طور پر دکھایا گیا ہے۔

اب تک، ہم نے بائیں اور دائیں بچوں کے حوالہ جات کے ساتھ آبیچٹ کا استعمال کرتے ہوئے بائینری ٹری نوڈس کی نمائندگی کی ہے۔ درخت کی نمائندگی کرنے کا ایک بالکل مختلف طریقہ ہے: ایک صف کے ساتھ۔

صف کے نقطہ نظر میں، نوڈس ایک صف میں محفوظ ہوتے ہیں اور حوالہ جات سے منسلک نہیں ہوتے ہیں۔ صف میں نوڈ کی پوزیشن درخت میں اس کی پوزیشن کے مساوی ہے۔ ہم روٹ نوڈ کو انڈیکس 0 پر رکھتے ہیں۔ جڑ کے بائیں بچے کو انڈیکس 1 پر اور اس کے دائیں بچے کو انڈیکس 2 پر رکھا جاتا ہے، اور اسی طرح درخت کی ہر سطح کے ساتھ ساتھ بائیں سے دائیں آگے بڑھتا ہے۔ یہ نقطہ نظر تصویر 8-24 میں دکھایا گیا ہے، جو ایک بائینری سرچ ٹری ہے جس میں کلید کے حروف ہیں۔



تصویر 8-24 ایک بائینری درخت جس کی نمائندگی ایک صف سے ہوتی ہے۔

درخت کی ہر پوزیشن، چاہے وہ موجودہ نوڈ کی نمائندگی کرے یا نہ کرے، صف میں موجود سیل سے مساوی ہے۔ درخت میں دی گئی پوزیشن پر نوڈ کو شامل کرنے کا مطلب ہے نوڈ کو صف میں موجود مساوی سیل میں داخل کرنا۔ بغیر نوڈس کے درخت کی پوزیشنوں کی نمائندگی کرنے والے سیل None، 0 یا کچھ اور خاص قدر سے بھرے ہوئے ہیں جنہیں نوڈ کے ساتھ الجھایا نہیں جا سکتا۔

تصویر میں، خالی نوڈس کے لیے صف میں ° علامت استعمال کی گئی ہے۔

اس اسکیم کے ساتھ، ایک نوڈ کے بچوں اور والدین کو صف میں نوڈ کے انڈیکس نمبر پر کچھ آسان ریاضی لگا کر تلاش کیا جا سکتا ہے۔ اگر نوڈ کا انڈیکس نمبر n ہے، تو اس نوڈ کا بائیں بچہ ہے۔

2 انڈیکس + 1

اس کا صحیح بچہ ہے

2 انڈیکس + 2

اور اس کے والدین ہیں

(انڈیکس 2 // 2) 1

(جہاں // بغیر کسی باقی کے عددی تقسیم کی نشاندہی کرتا ہے)۔ آپ شکل 24-8 میں انڈیکس کو دیکھ کر ان فارمولوں کے کام کی تصدیق کر سکتے ہیں۔ کوئی بھی الگورتھم جو نوڈس کے درمیان روابط کی پیروی کرتا ہے آسانی سے اس بات کا تعین کر سکتا ہے کہ اگلے نوڈ کو کہاں چیک کرنا ہے۔ یہ اسکیم کسی بھی بائنری درخت کے لیے کام کرتی ہے، نہ کہ صرف بائنری تلاش کے درختوں کے لیے۔ اس میں اچھی خصوصیت ہے کہ نوڈس کے درمیان کناروں / لنکس کا سفر کرنا اتنا ہی آسان ہے جتنا کہ وہ نیچے جا رہے ہیں (فہرستوں کے لئے ڈبل لنکنگ کی ضرورت کے بغیر)۔ اس سے بھی بہتر، یہ کسی بھی درخت پر عام کیا جا سکتا ہے جس میں بچوں کی ایک مقررہ تعداد ہے۔

تاہم، زیادہ تر حالات میں، ایک صف کے ساتھ درخت کی نمائندگی کرنا زیادہ کارگر نہیں ہوتا ہے۔ بھرے ہوئے نوڈس صف میں سوراخ چھوڑ دیتے ہیں، میموری کو ضائع کر دیتے ہیں۔ اس سے بھی بدتر، جب نوڈ کو حذف کرنے میں ذیلی درختوں کو منتقل کرنا شامل ہوتا ہے، سب ٹری میں موجود ہر نوڈ کو صف میں اس کے نئے مقام پر منتقل کیا جانا چاہیے، جو بڑے درختوں میں وقت طلب ہے۔ ان انسریشنز کے لیے جو نوڈس کو درخت کی موجودہ زیادہ سے زیادہ گہرائی سے باہر داخل کرتے ہیں، سرنی کا سائز تبدیل کرنے کی ضرورت پڑ سکتی ہے۔

اگر حذف کرنے کی اجازت نہیں ہے یا بہت ناپاب ہیں اور درخت کی زیادہ سے زیادہ گہرائی کا اندازہ لگایا جا سکتا ہے، تو صف کی نمائندگی مفید ہو سکتی ہے، خاص طور پر اگر ہر نوڈ کے لیے متحرک طور پر میموری حاصل کرنا، کسی وجہ سے، بہت زیادہ وقت طلب ہے۔ ایسا ہو سکتا ہے جب اسمبلی لینگویج میں پروگرامنگ ہو یا ایک بہت ہی محدود آپریٹنگ سسٹم، یا ایسا سسٹم جس میں کوڑا کرکٹ جمع نہ ہو۔

درخت کی سطح اور سائز

جب درختوں کو صفوں کے طور پر پیش کیا جاتا ہے، تو زیادہ سے زیادہ سطح اور نوڈس کی تعداد سرنی کے سائز کے لحاظ سے تنگ ہوتی ہے۔ منسلک درختوں کے لیے، کوئی خاص زیادہ سے زیادہ نہیں ہے۔ دونوں نمائندگیوں کے لیے، موجودہ زیادہ سے زیادہ سطح اور نوڈس کی تعداد کا تعین صرف درخت سے گزر کر کیا جا سکتا ہے۔ اگر ان میٹرکس کی درخواست کرنے کے لیے متواتر کالیں آتی ہیں، تو Binary Search Tree آپیکٹ ان کے لیے اقدار کو برقرار رکھ سکتا ہے، لیکن insert() اور delete() طریقوں کو قدروں کو اپ ڈیٹ کرنے کے لیے تبدیل کیا جانا چاہیے کیونکہ نوڈس کو شامل اور ہٹا دیا جاتا ہے۔

منسلک درخت میں نوڈس کو گنتی کے لیے، آپ تمام نوڈس پر اعادہ کرنے کے لیے (`traverse`) طریقہ استعمال کر سکتے ہیں اور گنتی میں اضافہ کر سکتے ہیں، جیسا کہ پہلے مثال میں اوسط کلیدی قدر تلاش کرنے کے لیے اور دوبارہ فہرست کے 8 کے (`nodes`) طریقہ میں دکھایا گیا ہے۔ 10- زیادہ سے زیادہ سطح تلاش کرنے کے لیے، آپ ایک ہی تکنیک کا استعمال نہیں کر سکتے ہیں کیونکہ ٹراورسل کے دوران ہر نوڈ کی سطح فراہم نہیں کی جاتی ہے (حالانکہ اسے جنریٹر میں ترمیم کر کے شامل کیا جا سکتا ہے)۔ اس کے بجائے، لسٹنگ 8-10 میں دکھائی گئی تکراری تعریف کام کو کوڈ کی چند سطروں میں انجام دیتی ہے۔

فہرست سازی BinarySearchTree 8-10 کے درجے (اور نوڈس) طریقے

کامیابی `treeTraverse` (بیجیکٹ):

ڈیپ ٹریوں کو پیمائش کریں۔

واپسی # `self._levels(self._root)` سے شروع ہونے والی گنتی

`def _levels(self, node):` if node:

بار بار ذیلی درخت میں سطحوں کی گنتی کریں۔

اگر نوڈ فراہم کیا جاتا ہے، تو لیول 1 ہے۔

واپسی # `max child + max(self._levels(node.leftChild), self._levels(node.rightChild))` # خالی ذیلی درخت کی

کوئی سطح نہیں ہے

دوسری: واپسی 0

ڈیف نوڈس (خود):

شمار 0 =

ایٹریٹر کا استعمال کرتے ہوئے ٹری نوڈس کو شمار کریں۔

ایک خالی درخت فرض کریں۔

کلید کے لیے، `self.traverse` میں ڈیٹا # (کسی بھی میں تمام کلیدوں پر اعادہ کریں۔

آرڈر اور انکریمنٹ شمار

شمار 1 +=

واپسی کی گنتی

ذیلی درخت کی سطحوں کو شمار کرنا اس سے کچھ مختلف ہے جو آپ نے پہلے دیکھا ہے کہ ہر نوڈ اپنے ہر ذیلی درخت کی زیادہ سے زیادہ سطح لیتا ہے اور نوڈ کے لیے اس میں ایک کا اضافہ کرتا ہے۔ ایسا لگتا ہے جیسے کم سے کم یا زیادہ سے زیادہ کلید کی گہرائی کو دیکھ کر کوئی شارٹ کٹ ہونا چاہئے تاکہ آپ کو ہر نوڈ پر جانے کی ضرورت نہ ہو۔ اگر آپ اس کے بارے میں سوچتے ہیں، تاہم، یہاں تک کہ کم سے کم اور زیادہ سے زیادہ چابیاں تلاش کرنے سے بھی درخت کے بائیں اور دائیں "فلانکس" پر گہرائی ظاہر ہوتی ہے۔ درمیان میں کہیں طویل راستے ہوسکتے ہیں، اور انہیں تلاش کرنے کا واحد طریقہ تمام نوڈس کا دورہ کرنا ہے۔

پرٹنگ درخت

آپ نے دیکھا ہے کہ درختوں کو مختلف ترتیبوں میں کیسے عبور کرنا ہے۔ آپ درخت کے تمام نوڈس کو پرٹ کرنے کے لیے ہمیشہ ٹراورسل طریقہ استعمال کر سکتے ہیں، جیسا کہ ویڈیو لائزیشن ٹول میں دکھایا گیا ہے۔ ان آرڈر ٹراورسل کا استعمال اشیاء کو ان کی چابیاں کے بڑھتے ہوئے ترتیب میں دکھانے گا۔ دو جتی آؤٹ پٹ پر، آپ افقی محور کے ساتھ نوڈس اور ہر نوڈ کی سطح کو اس کی عمودی پوزیشن کا تعین کرنے کے لیے ترتیب ترتیب کا استعمال کر سکتے ہیں۔ یہ پچھلے اعداد و شمار میں دکھائے گئے درختوں کے خاکے تیار کر سکتا ہے۔

ایک سادہ کمانڈ لائن آؤٹ پٹ پر، فی لائن ایک نوڈ پرٹ کرنا آسان ہے۔ پھر مسئلہ درخت کی شکل کی نشاندہی کرنے کے لیے لائن پر نوڈ کی پوزیشننگ بن جاتا ہے۔ اگر آپ سب سے اوپر روٹ نوڈ چاہتے ہیں، تو آپ کو پورے درخت اور جگہ کی چورائی کا حساب لگانا ہوگا۔

380 باب 8 بائری درخت

وہ نوڈ پوری چوڑائی کے وسط میں ہے۔ زیادہ درست طریقے سے، آپ کو بائیں اور دائیں ذیلی درختوں کی چوڑائی کا حساب لگانا ہوگا اور متوازن اور غیر متوازن درختوں کو درست طریقے سے دکھانے کے لیے جڑ کو پوزیشن میں رکھنے کے لیے اس کا استعمال کرنا ہوگا۔

دوسری طرف، اگر آپ روٹ کو آؤٹ پٹ لائن کے بائیں جانب رکھتے ہیں اور نوڈس کی سطح کو سب سے بائیں کالم سے انڈینٹیشن کے طور پر دکھاتے ہیں، تو درخت کو ٹرمینل پر پرنٹ کرنا آسان ہے۔ ایسا کرنا بنیادی طور پر درخت کو 90° بائیں گھماتا ہے۔ درخت کا ہر نوڈ آؤٹ پٹ کی اپنی لائن پر ظاہر ہوتا ہے۔ یہ آپ کو ذیلی درختوں کی چوڑائی کا تعین کرنے کے بارے میں بھولنے اور ایک سادہ تکراری طریقہ لکھنے کی اجازت دیتا ہے، جیسا کہ فہرست 8-11 میں دکھایا گیا ہے۔

ایک نوڈ فی لائن کے ساتھ درختوں کو پرنٹ کرنے کے 8-11 طریقوں کی فہرست

کتابخانہ JhtraSyran (بجیکٹ):

...
`void printTree(self, Node node, int type, int indent, int indentBy=4) {
 cout << " " << node->data << "\n";
 if (node->left) printTree(self, node->left, type, indent+indentBy, indentBy);
 if (node->right) printTree(self, node->right, type, indent+indentBy, indentBy);
}`

`void printTree(self, Node node, int type, int indent, int indentBy=4) {`

`cout << " " << node->data << "\n";`
 ٹائپ اپنے # والدین سے تعلق کو ظاہر کرتا ہے اور انڈینٹ اس کی سطح کو ظاہر کرتا ہے

ذیلی درختوں کے لیے انڈینٹ لیول میں اضافہ کریں۔

صرف اس صورت میں پرنٹ کریں جب کوئی نوڈ ہو۔

اگر نوڈ:

`self->printTree(node->rightChild, "RIGHT:", #`

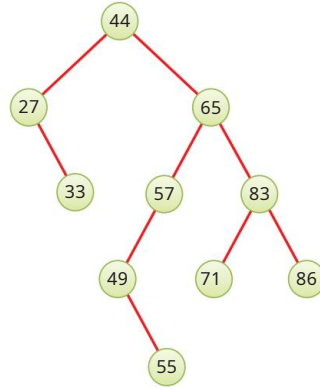
`indentBy) # subtree print(indent + node->type, node) #`

`self->printTree(node->leftChild, "LEFT:", # "اس نوڈ کو خود پرنٹ کریں۔", # indentBy,`

`#`

`# "ذیلی درخت" * indentBy, indentBy) #`

پبلک پرنٹ () طریقہ روٹ نوڈ سے شروع ہونے والے نوڈس کو بار بار پرنٹ کرنے کے لیے نجی `printTree()` طریقہ کو کہتے ہیں۔ یہ ایک پیرامیٹر لیتا ہے، `indentBy` یہ کنٹرول کرنے کے لیے کہ درخت کی ہر سطح کو انڈینٹ کرنے کے لیے کتنی جگہیں استعمال کی جاتی ہیں۔ یہ نوڈس پر لیبل لگاتا ہے تاکہ ان کے والدین کے ساتھ ان کا رشتہ ظاہر کیا جا سکے (اگر یہ ان کے انڈینٹیشن اور رشتہ دار پوزیشن سے پہلے ہی واضح نہیں تھا)۔ تکراری طریقہ پر عمل درآمد بیس کیس، ایک خالی نوڈ کو چیک کر کے شروع ہوتا ہے، اس صورت میں کچھ بھی پرنٹ کرنے کی ضرورت نہیں ہے۔ ہر دوسرے نوڈ کے لیے، یہ سب سے پہلے دائیں سب ٹری کو بار بار پرنٹ کرتا ہے کیونکہ یہ پرنٹ شدہ ورژن کا سب سے اوپر ہے۔ یہ انڈینٹ میں خالی جگہیں شامل کرتا ہے تاکہ ذیلی درخت کو مزید دائیں طرف پرنٹ کیا جائے۔ پھر یہ موجودہ نوڈ کو اس کے انڈینٹیشن اور نوڈ ٹائپ لیبل کے ساتھ پہلے سے پرنٹ کرتا ہے۔ آخر میں، یہ توسیع شدہ انڈینٹیشن کے ساتھ بائیں ذیلی درخت کو چھپی ہوئی چھاپتا ہے۔ یہ ایک آؤٹ پٹ تیار کرتا ہے جیسا کہ شکل 8-25 میں دکھایا گیا ہے۔ نوڈس کو {key, data} جوڑوں کے بطور پرنٹ کیا جاتا ہے اور اعداد و شمار کی مثال میں اس کے ساتھ کوئی ڈیٹا محفوظ نہیں ہوتا ہے۔



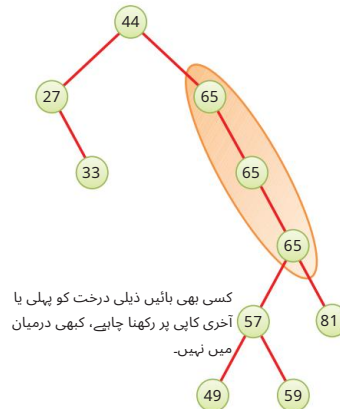
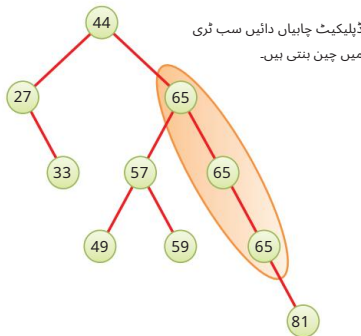
شکل 8-25 درخت نوڈ کی گہرائی کے لیے انڈینٹیشن کے ساتھ پرنٹ کیا گیا ہے۔

درخت کو اس طرح پرنٹ کرنے میں، آپ تین معیاری ترتیب سے مختلف ٹراورسل آرڈر استعمال کرتے ہیں۔ پرنٹ آرڈر درخت کے الٹ ان آرڈر ٹراورسل کا استعمال کرتا ہے۔

ڈپلیکیٹ چابیاں

دوسرے ڈیٹا ڈھانچے کی طرح، ڈپلیکیٹ کیز کے مسئلے کو حل کرنا ضروری ہے۔ insert() کے لیے دکھائے گئے کوڈ میں اور ویژولائزیشن ٹول میں، ڈپلیکیٹ کلید والا نوڈ داخل نہیں کیا جائے گا۔ ٹول نوڈ کو بھرنے کے لیے ایک نئے رنگ کے دائرے کو حرکت دے کر اپ ڈیٹ کیے جانے والے نوڈ کا ڈیٹا دکھاتا ہے۔

ڈپلیکیٹ کلیدوں کی اجازت دینے کے لیے، آپ کو کئی انتخاب کرنا ہوں گے۔ ڈپلیکیٹس بنیادی بائنری سرچ ٹری اصول کی بنیاد پر صحیح ذیلی درخت میں جاتے ہیں۔ وہ صرف صحیح چائلڈ لنکس کے ساتھ نوڈس کا ایک سلسلہ بناتے ہیں، جیسا کہ شکل 8-26 میں دکھایا گیا ہے۔ ڈیزائن کے انتخاب میں سے ایک یہ ہے کہ کسی بھی بائیں چائلڈ لنک کو کہاں رکھنا ہے۔ اسے صرف سلسلہ میں پہلے یا آخری ڈپلیکیٹ پر جانا چاہئے تاکہ الگورتھم کو معلوم ہو کہ اسے کہاں تلاش کرنا ہے۔ اعداد و شمار دو انتخاب کی وضاحت کرتا ہے۔ نئی ڈپلیکیٹ چابیاں سلسلہ کے مخالف سرے پر ڈالی جانی چاہئیں۔



تصویر 8-26 بائنری سرچ ٹریز میں ڈپلیکیٹ کیز

ایک اور انتخاب یہ ہے کہ ڈپلیکیٹس والی کلید کے لیے (`find()` اور `search()` طریقوں سے کیا لوٹنا ہے۔ کیا اسے پہلی یا آخری واپس کرنی چاہئے؟ انتخاب اس کے مطابق بھی ہونا چاہئے کہ کس نوڈ کو حذف کیا جاتا ہے اور حذف (طریقہ سے واپس کیا جاتا ہے۔ اگر انہیں پہلے داخل کیا جاتا ہے اور پہلے سے ہٹا دیا جاتا ہے، تو حذف) ڈوبلی کیٹ نوڈس کے لیے منی اسٹیک کی طرح کام کرے گا۔

حذف کرنے کا عمل اس حقیقت سے پیچیدہ ہے کہ ہر ڈپلیکیٹ نوڈس پر مختلف ڈیٹا ویلیوز کو محفوظ کیا جا سکتا ہے۔ کال کرنے والے کو ڈپلیکیٹ کلید والے کسی نوڈ کے بجائے مخصوص ڈیٹا والے نوڈ کو حذف کرنے کی ضرورت پڑسکتی ہے۔ جو بھی اسکیم منتخب کی گئی ہے، حذف کرنے کے معمول کو اس بات کو یقینی بنانے کی ضرورت ہوگی کہ بائیں ذیلی درخت، اگر کوئی ہے، مناسب جگہ سے منسلک رہے۔

کسی بھی قسم کی ڈپلیکیٹ چابیاں کے ساتھ، درخت کو متوازن کرنا مشکل یا ناممکن ہو جاتا ہے۔ ڈپلیکیٹس کی زنجیریں اضافی سطحیں شامل کرتی ہیں جنہیں توازن میں مدد کے لیے دوبارہ ترتیب نہیں دیا جا سکتا۔ اس کا مطلب ہے کہ آٹم کو تلاش کرنے کی کارکردگی $O(\log N)$ کے بہترین کیس سے $O(N)$ کی طرف بڑھ جاتی ہے۔

جیسا کہ آپ دیکھ سکتے ہیں، ڈپلیکیٹ کیز کو اجازت دینا ڈیٹا ڈھانچے میں کوئی آسان اضافہ نہیں ہے۔ ڈیٹا کے دوسرے ڈھانچے میں، ڈپلیکیٹ کیز چینجز پیش کرتی ہیں، لیکن یہ سب بانٹری سرچ ٹری کی طرح مشکل نہیں ہیں۔

BinarySearchTreeTester.py پروگرام

یہ ہمیشہ ایک اچھا خیال ہے کہ کوڈ ماڈیول کے کام کاج کو ٹیسٹ لکھ کر جانچیں جو ہر آپریشن کو استعمال کرتے ہیں۔ ٹیسٹوں کا ایک جامع سیٹ لکھنا اپنے آپ میں ایک فن ہے۔ ایک اور کارآمد حکمت عملی ایک انٹرایکٹو ٹیسٹ پروگرام لکھنا ہے جو آپ کو مختلف ترتیبوں اور مختلف دلائل کے ساتھ آپریشنز کی ایک سیریز کو آزمانے کی اجازت دیتا ہے۔ دکھائے گئے تمام `BinarySearchTree` کلاس میتھ `ods` جانچنے کے لیے، آپ `BinarySearchTreeTester.py` جیسا پروگرام استعمال کر سکتے ہیں جو فہرست 8-12 میں دکھایا گیا ہے۔

فہرست سازی `BinarySearchTreeTester.py` 8-12 پروگرام

`BinarySearchTree` #امورٹ سے انٹرایکٹو طریقے سے `BinarySearchTree` کلاس کی جانچ کریں *

```
theTree = BinarySearchTree()
```

#خالی درخت سے شروع کریں۔

```
"2006 2") theTree.insert("Tim", "2016 1")
"2006 1") theTree.insert("Vint",
" 2002 3") theTree.insert("Fran",
"2002 3") theTree.insert("Ron",
"1996 1") theTree.insert("Adi",
"1994 1") theTree.insert("Amir",
"1988 1 " ") theTree.insert("Raj",
"1979 1") theTree.insert("Ivan",
"1975 2") theTree.insert("Ken",
"1974 1") theTree.insert("Herb",
theTree.insert("Don",
```

#کچھ ڈیٹا داخل کریں۔

```

def clearTree():
    #درخت کے تمام نوڈس کو ہٹا دیں۔
    جبکہ theTree.isEmpty():
        = theTree.root() theTree.delete(key) کلید

def traverseTree(traverseType="in"): # Tree.traverse(traverseType)
    میں کلید، ڈیٹا کے لیے تمام نوڈس کو عبور اور پرنٹ کریں:
    پرنٹ کریں

    کمانڈز # = کمانڈ کے نام، افعال، اور ان کے پیرامیٹرز
    theTree.insert( ('key', 'data'), ['delete', theTree.delete, ('key', )],
        [], ['quit', none, []], ['تلاش', 'print', theTree.print, []], ['insert',
        clearTree, []], ['help', print_commands, []], ['?', print_commands,
        theTree.search, ('key', )], ['traverse', traverseTree, ('type', )], ['clear',

]

# ایک فہرست میں کمانڈ کے تمام نام جمع کریں۔
". ".join(c[0] for c in commands) for i in range(len(commands)):
    command_names =
        #دلیل کی فہرست میں کمانڈ کے نام رکھیں
        اگر کمانڈ [1] == [1] پرنٹ کمانڈز: پرنٹ کمانڈز کا #
        [2] = [command_names][i] کمانڈز

# کمانڈ کے نام کے پہلے حرف کو # کمانڈ کی تفصیلات (نام، فنکشن، پیرامیٹرز/آرگس) کمانڈ ڈکٹ c = dict((c[0][0], c) کے لیے ایک لغت بنائیں

# انٹرایکٹو لوپ کے لیے معلومات پرنٹ کریں۔
print_commands(command_names)
theTree.print()
    = جواب

# صارف سے کمانڈ حاصل کرنے کے لیے لوپ کریں اور اس پر عمل کرنے بونے جواب دیں: 'q' != [0]
    پرنٹ 'درخت ہے'، theTree.nodes(), 'نوڈس پار'،
    theTree.levels(), 'levels')
    if len(ans) == 0: (rewol("کمانڈ کا پہلا حرف درج کریں:")):
        =
        = جواب

    اگر جواب [0] کمانڈ ڈکٹ میں:
        نام، فنکشن، پیرامیٹرز = کمانڈ ڈکٹ [جواب][0]
        اگر فنکشن کوئی نہیں ہے:

```


384 باب 8 بائری درخت

```
= if print(name)ecnatsnisiprint(پیرامیٹر، فہرست):
    argumentsپیرامیٹرز
```

دوسری:

```
param arguments = [] میں پیرامیٹرز
arguments.append(arg) کے لیے:
arg = input("Enter " "command:") " +
```

کوشش کریں: نتیجہ = فنکشن (*دلائل) پرنٹ (نتیجہ):
نتیجہ

استثناء کے علاوہ بطور ای: پرنٹ (استثنیٰ واقع ہوا) پرنٹ
(ای)

دوسری:

پرنٹ ("غلط کمانڈ: ", "جواب: ")

یہ پروگرام صارفین کو ٹرمینل انٹرفیس میں ٹائپ کر کے کمانڈ داخل کرنے کی اجازت دیتا ہے۔ یہ سب سے پہلے BinarySearchTree ماڈیول درآمد کرتا ہے اور اس کے ساتھ ایک خالی درخت بناتا ہے۔ پھر یہ کچھ تاروں کے ساتھ ناموں کو جوڑنے کے لیے insert() کا استعمال کرتے ہوئے اس میں کچھ ڈیٹا رکھتا ہے۔ نام وہ چابیاں ہیں جو درخت کے اندر نوڈس رکھنے کے لیے استعمال ہوتی ہیں۔

ٹیسٹر تمام ممکنہ کمانڈز کو پرنٹ کرنے، درخت سے تمام نوڈس کو صاف کرنے، اور ہر نوڈ کو پرنٹ کرنے کے لیے درخت کو عبور کرنے کے لیے متعدد افادیت کے افعال کی وضاحت کرتا ہے۔ یہ فنکشن نیچے کمانڈ لوپ میں کمانڈز کو بینڈل کرنے ہیں۔

ٹیسٹر پروگرام کا اگلا حصہ کمانڈز کی فہرست کی وضاحت کرتا ہے۔ ہر ایک کے لیے، اس کا ایک نام، کمانڈ پر عمل کرنے کے لیے ایک فنکشن، اور دلائل یا پیرامیٹرز کی فہرست یا ٹوپل ہے۔

یہ ہم نے اب تک دکھایا ہے اس سے کہیں زیادہ جدید Python کوڈ ہے، لہذا یہ تھوڑا سا عجیب لگ سکتا ہے۔ نام وہ ہیں جو صارف ٹائپ کرے گا (یا کم از کم ان کا پہلا حرف)، اور فنکشنز یا تو درخت کے طریقے ہیں یا ٹیسٹر میں بیان کردہ یوٹیلیٹی فنکشنز۔ صارف کے کمانڈ کا انتخاب کرنے کے بعد دلائل اور پیرامیٹرز پر کارروائی کی جائے گی۔

تھوڑی کمانڈ لائن مدد فراہم کرنے کے لیے، ٹیسٹر کمانڈ کے ناموں کی فہرست کو ایک سٹرنگ میں جوڑتا ہے، انہیں کوما سے الگ کرتا ہے۔ یہ آپریشن جوائن () سٹرنگ کے طریقہ سے مکمل کیا جاتا ہے۔ ہر کمانڈ کے نام کے درمیان جو متن رکھنا ہے وہ سٹرنگ ہے (کوما اور اسپیس)، اور جوائن کرنے کی دلیل ناموں کی فہرست ہے۔ پروگرام کمانڈز میں کمانڈ کی وضاحتوں کے ذریعے اعادہ کرنے کے لیے فہرست کی سمجھ کا استعمال کرتا ہے اور پہلا عنصر نکالتا ہے، جو کہ کمانڈ کا نام ہے: .join(c[0] for c in commands) ، "نتیجہ کمانڈ نام متغیر میں محفوظ ہے۔"

پھر کمانڈ کے ناموں کی مربوط سٹرنگ کو print_commands فنکشن کے لیے دلیل کی فہرست میں داخل کرنے کی ضرورت ہے۔ یہ لوپ کے لئے میں کیا گیا ہے۔ دو اندراجات میں print_commands فنکشن ہے: مدد اور ؟ احکامات

کمانڈ لوپ کے لیے تیاری کا آخری حصہ ایک لغت، کمانڈ ڈکٹ بناتا ہے، جو ہر کمانڈ کے پہلے کردار کو کمانڈ کی تفصیلات میں نقشہ بناتا ہے۔ آپ کے پاس نہیں ہے۔

ابھی تک اس Python ڈیٹا کا ڈھانچہ استعمال کیا ہے۔ باب 11 میں، "بیش ٹیبلز"، آپ دیکھتے ہیں کہ وہ کیسے کام کرتے ہیں، لہذا اگر آپ ان سے واقف نہیں ہیں، تو ان کے بارے میں سوچیں کہ انٹیجر کے بجائے ایک سٹرنگ کے ذریعہ ترتیب کردہ ایک سرنی۔ آپ صف میں قدریں تفویض کر سکتے ہیں اور پھر انہیں تیزی سے تلاش کر سکتے ہیں۔ ٹیسٹر پروگرام میں، `command_dict['p']` کا جائزہ لینے سے پرنٹ کمانڈ، یعنی `[]` کے لیے `'print', theTree.print,` تفصیلات واپس آ جائیں گی۔ وہ تصریحات کومپیکٹ (لیکن خفیہ) فہم کا استعمال کرتے ہوئے لغت میں محفوظ ہو جاتی ہیں: `dict(c[0][0], c)` کمانڈز میں `c` کے لیے۔

باقی ٹیسٹر کمانڈ لوپ کو نافذ کرتا ہے۔ یہ پہلے درخت کو ٹر منل پر پرنٹ کرتا ہے، اس کے بعد کمانڈز کی فہرست۔ جواب متغیر صارف کے ذریعہ ٹائپ کردہ ان پٹ رکھتا ہے۔ یہ ایک اسپیس میں شروع ہو جاتا ہے تاکہ کمانڈ لوپ شروع ہو اور ایک نئی کمانڈ کا اشارہ کرے۔

کمانڈ لوپ اس وقت تک جاری رہتا ہے جب تک کہ صارف `quit` کمانڈ کو طلب نہیں کرتا، جو `q` سے شروع ہوتا ہے۔ لوپ باڈی کے اندر، درخت میں نوڈس اور لیولز کی تعداد پرنٹ کی جاتی ہے، اور پھر صارف سے کمانڈ کے لیے کہا جاتا ہے۔ `input()` کے ذریعے واپس آنے والی سٹرنگ کو کمانڈ کی تلاش کو آسان بنانے کے لیے چھوٹے حروف میں تبدیل کیا جاتا ہے۔ اگر صارف نے صرف واپسی کو دیا، تو اسٹرنگ میں کوئی پہلا حرف نہیں ہوگا، لہذا آپ ایک کو بھریں گے؟ پہلے سے طے شدہ جواب بنانے کے لیے تمام کمانڈ کے ناموں کو دوبارہ پرنٹ کرنا ہے۔

اگلے بیان میں — `if ans[0] in command_dict:` — ٹیسٹر چیک کرتا ہے کہ آیا صارف کے جواب میں پہلا کردار معلوم کمانڈز میں سے ایک ہے۔ اگر کریکٹر کو `gnized rec` کیا جاتا ہے، تو یہ کمانڈ ڈکٹ میں محفوظ کردہ تفصیلات سے نام، فنکشن اور پیرامیٹرز نکالتا ہے۔ اگر عمل کرنے کے لیے کوئی فنکشن ہے، تو اس پر کارروائی کی جائے گی۔ اگر نہیں، تو صارف نے چھوٹے کو کہا، اور `while` لوپ باہر نکل جائے گا۔ جب صارف کے جواب کا پہلا کردار کسی کمانڈ سے میل نہیں کھاتا ہے، تو ایک ایبر میسج پرنٹ کیا جاتا ہے، اور لوپ ایک نئی کمانڈ کا اشارہ کرتا ہے۔

کمانڈ کی تفصیلات ملنے کے بعد، اسے یا تو صارف کو فنکشن کو کال کرتے وقت استعمال کرنے کے لیے دلائل کے لیے اشارہ کرنے کی ضرورت ہوتی ہے یا انہیں تفصیلات سے حاصل کرنا ہوتا ہے۔ یہ انتخاب اس بات پر مبنی ہے کہ آیا پیرامیٹرز کو `Python tuple` یا فہرست کے طور پر بیان کیا گیا تھا۔ اگر یہ ٹوپل ہے، تو ٹیبل کے عناصر پیرامیٹرز کے نام ہیں۔ اگر یہ فہرست ہے، تو فہرست میں فنکشن کے دلائل ہوتے ہیں۔ ٹیبلز کے لیے، صارف کو ہر دلیل کو نام کے ساتھ درج کرنے کے لیے کہا جاتا ہے، اور جوابات دلائل کی فہرست میں محفوظ کیے جاتے ہیں۔ دلائل کے تعین کے بعد، کمانڈ لوپ رزلٹ = فنکشن(*دلائل) کا استعمال کرتے ہوئے آرگومینٹس کی فہرست کے ساتھ فنکشن کو کال کرنے کی کوشش کرتا ہے۔ دلائل سے پہلے ستارہ (*ضرب کیشن آپریٹر نہیں ہے۔ اس کا مطلب ہے کہ دلائل کی فہرست کو فنکشن کے لیے پوزیشن دلائل کی فہرست کے طور پر استعمال کیا جانا چاہیے۔ اگر فنکشن کوئی مستثنیات اٹھاتا ہے، تو وہ پکڑے جاتے ہیں اور دکھائے جاتے ہیں۔ دوسری صورت میں، فنکشن کا نتیجہ دوسری کمانڈ حاصل کرنے کے لیے لوپ کرنے سے پہلے پرنٹ کیا جاتا ہے۔

ٹیسٹر کو استعمال کرنے کی کوشش کریں چار اہم آپریشنز: تلاش کریں، داخل کریں، عبور کریں، اور حذف کریں۔ حذف کرنے کے لیے، اثر دیکھنے کے لیے، 1، 0 اور 2 چائلڈ نوڈس والے نوڈس کو حذف کرنے کی کوشش کریں۔ جب آپ 2 بچوں کے ساتھ نوڈ کو حذف کرتے ہیں، تو اندازہ لگائیں کہ کون سا جانشین نوڈ حذف شدہ نوڈ کی جگہ لے گا اور دیکھیں کہ آیا آپ صحیح ہیں۔

بف مین کوڈ

آپ کو یہ خیال نہیں آنا چاہئے کہ ہائیری درخت ہمیشہ تلاش کے درخت ہوتے ہیں۔ بہت سے ہائیری درخت دوسرے طریقوں سے استعمال ہوتے ہیں۔ شکل 8-16 ایک مثال دکھاتی ہے جہاں ایک ہائیری درخت ایک الجی بریک اظہار کی نمائندگی کرتا ہے۔ اب ہم ایک الگورتھم پر بات کرتے ہیں جو ڈیٹا کو کمپریس کرنے کے لیے حیران کن انداز میں ہائیری ٹری کا استعمال کرتا ہے۔ اسے بف مین کوڈ کہتے ہیں، ڈیوڈ بف مین کے بعد، جس نے اسے 1952 میں دریافت کیا تھا۔ ڈیٹا کمپریشن بہت سے حالات میں اہم ہوتا ہے۔ ایک مثال انٹرنیٹ پر یا ڈیجیٹل نشریات کے ذریعے ڈیٹا بھیجنا ہے، جہاں معلومات کو اس کی مختصر ترین شکل میں بھیجنا ضروری ہے۔ ڈیٹا کو کمپریس کرنے کا مطلب ہے کہ ایک ہی وقت میں بینڈوڈنھ کی حد کے تحت مزید ڈیٹا بھیجا جا سکتا ہے۔

کریکٹر کوڈز ایک غیر کمپریسڈ ٹیکسٹ فائل میں ہر کریکٹر کو کمپیوٹر میں ایک سے چار بائٹس کے ذریعے دکھایا جاتا ہے، اس پر منحصر ہے کہ حروف کو انکوڈ کیا جاتا ہے۔ قابل احترام ASCII کوڈ کے لیے، صرف ایک بائٹ استعمال کیا جاتا ہے، لیکن یہ حروف کی حد کو محدود کر دیتا ہے جن کا اظہار 128 سے کم ہو سکتا ہے۔ دنیا کی تمام زبانوں کے علاوہ دیگر علامتوں جیسے ایموجیز کا حساب کتاب کرنے کے لیے، مختلف یونی کوڈ معیارات تک استعمال کرتے ہیں۔ چار بائٹس فی کریکٹر۔ اس بحث کے لیے، ہم فرض کرتے ہیں کہ صرف ASCII حروف کی ضرورت ہے، اور ہر کردار ایک بائٹ (یا آٹھ بٹس) لیتا ہے۔

جدول 8-2 دکھاتا ہے کہ ASCII کوڈ کا استعمال کرنے ہوئے ہائیری میں کچھ حروف کی نمائندگی کیسے کی جاتی ہے۔

جدول 8-2 کچھ ASCII کوڈز

گولڈناریہ	بائیری
0	01000000
اے	01000001
بی	01000010
...	...
Y	01011001
Z	01011010
...	...
a	01100001
ب	01100010

ڈیٹا کو کمپریس کرنے کے کئی طریقے ہیں۔ متن کے لیے، سب سے عام نقطہ نظر ہٹس کی تعداد کو کم کرنا ہے جو سب سے زیادہ استعمال ہونے والے حروف کی نمائندگی کرتے ہیں۔ نتیجے کے طور پر، ہر کردار ہٹس کے "سٹریم" میں ہٹس کی متغیر تعداد لیتا ہے جو مکمل متن کی نمائندگی کرتا ہے۔

انگریزی میں، E اور T بہت عام حروف ہیں، جب نثر اور دوسرے شخص سے فرد کے مواصلات کا جائزہ لیتے ہیں اور خالی جگہوں اور اوقاف جیسی چیزوں کو نظر انداز کرتے ہیں۔ اگر آپ ایسی اسکیم کا انتخاب کرتے ہیں جس میں E، T، اور دوسرے عام حروف کو لکھنے کے لیے صرف چند بٹس استعمال کیے جاتے ہیں، تو یہ اس سے زیادہ کمپیٹ بونا چاہیے اگر آپ ہر حرف کے لیے یکساں بٹس استعمال کرتے ہیں۔ سپیکٹرم کے دوسرے سرے پر، Q اور Z شاذ و نادر ہی ظاہر ہوتے ہیں، اس لیے ان حروف کے لیے کبھی کبھار بڑی تعداد میں بٹس استعمال کرنا اتنا برا نہیں ہے۔

فرض کریں کہ آپ E کے لیے صرف دو بٹس استعمال کرتے ہیں — کہئے۔ 01 آپ انگریزی حروف تہجی کے ہر حرف کو دو بٹس میں انکوڈ نہیں کر سکتے کیونکہ صرف چار 2 بٹ کے امتزاج ہیں: 00، 01، 10، اور 11۔ کیا آپ ان چار مجموعوں کو سب سے زیادہ استعمال ہونے والے چار حروف کے لیے استعمال کر سکتے ہیں؟ ٹھیک ہے، اگر آپ نے ایسا کیا، اور آپ اب بھی کم استعمال شدہ حروف کے لیے کچھ انکوڈنگ کرنا چاہتے ہیں، تو آپ کو پریشانی ہوگی۔ بٹس کی تشریح کرنے والے الگورتھم کو کسی نہ کسی طرح یہ اندازہ لگانا ہوگا کہ بٹس کا جوڑا ایک کریکٹر ہے یا کچھ لمے کریکٹر کوڈ کا حصہ ہے۔

انکوڈنگ میں اہم خیالات میں سے ایک یہ ہے کہ ہمیں کوڈ کی قدروں میں سے کچھ کو اشارے کے طور پر ایک طرف رکھ دینا چاہیے جو کم استعمال شدہ کردار کو انکوڈ کرنے کے لیے تھوڑی لمبی سٹرنگ کی پیروی کرتی ہے۔ الگورتھم کو کسی خاص لمبائی کی تھوڑی سی تار کو دیکھنے اور اس بات کا تعین کرنے کے لیے ایک طریقہ کی ضرورت ہوتی ہے کہ آیا یہ حروف میں سے کسی ایک کے لیے مکمل کوڈ ہے یا لمبے کوڈ کی قدر کے لیے صرف ایک سابقہ ہے۔ آپ کو محتاط رہنا چاہیے کہ کسی بھی کردار کی نمائندگی اسی بٹ کے امتزاج سے نہیں ہوتی ہے جو کسی دوسرے کردار کے لیے استعمال ہونے والے لمبے کوڈ کے شروع میں ظاہر ہوتا ہے۔ مثال کے طور پر، اگر E 01 ہے، اور 01011000 Z ہے، تو 01011000 کو ڈی کوڈنگ کرنے والا الگورتھم یہ نہیں جانتا کہ ابتدائی E 01 کی نمائندگی کرتا ہے یا Z کا آغاز۔ یہ ایک اصول کی طرف جاتا ہے: کوئی کوڈ کسی کا سابقہ نہیں ہو سکتا۔ دوسرے کوڈ۔

اس بات پر بھی غور کریں کہ کچھ پیغامات میں، ممکن ہے E سب سے زیادہ استعمال ہونے والا حرف نہ ہو۔ اگر متن ایک پروگرام سورس فائل ہے، مثال کے طور پر، رموز اوقاف کے حروف جیسے بڑی آنت (:)، سیمی کالون (:)، اور انڈر سکور (E) کی نسبت زیادہ کثرت سے ظاہر ہو سکتے ہیں۔ اس مسئلے کا حل یہ ہے: ہر پیغام کے لیے، آپ اس مخصوص پیغام کے مطابق ایک نیا کوڈ بناتے ہیں۔ فرض کریں کہ آپ SPAM EGG + SPAM SPAM پیغام بھیجنا چاہتے ہیں۔ حرف S بہت زیادہ ظاہر ہوتا ہے، اور اسی طرح خلائی کردار بھی۔ ہو سکتا ہے آپ ایک ٹیبل بنانا چاہیں جس میں دکھایا جائے کہ ہر حرف کتنی بار ظاہر ہوتا ہے۔ اسے فریکوئنسی ٹیبل کہا جاتا ہے، جیسا کہ جدول 8-3 میں دکھایا گیا ہے۔

سپیم پیغام کے لیے جدول 8-3 فریکوئنسی ٹیبل

شمار	کیریکٹر		
4	ے		
4	ئی		
5	جلا		
1	4	+	

سب سے زیادہ گنتی والے حروف کو تھوڑی تعداد میں بٹس کے ساتھ کوڈ کیا جانا چاہیے۔ جدول 8-4 ایک طریقہ دکھاتا ہے کہ آپ سپام پیغام میں حروف کو کیسے انکوڈ کر سکتے ہیں۔

آپ اسپیس کے لیے 01 استعمال کر سکتے ہیں کیونکہ یہ سب سے زیادہ بار بار آتا ہے۔ اگلے سب سے زیادہ متواتر کردار ادا کرنے والے S، P، A، اور M ہیں، ہر ایک چار بار ظاہر ہوتا ہے۔ آپ آخری کوڈ 100 استعمال کرتے ہیں، M باقی کوڈز 00 یا 01 سے شروع نہیں ہو سکتے کیونکہ اس سے اصول ٹوٹ جائے گا۔

کہ کوئی کوڈ کسی دوسرے کوڈ کا سابقہ نہیں ہو سکتا۔ یہ 10 اور 11 کو دوسرے حروف کے لیے بطور سابقہ استعمال کرنے کے لیے چھوڑ دیتا ہے۔

سیم پیغام کے لیے جدول 4-8 بف مین کوڈ

کوڈ	شمار	کردار	کوڈ	بف مین	بف مین
110	4	بی	111		
101	4	اس	10000	1	ای
01	5	خلا	1001	2	جی
10001	1	+	00	4	ایم

3بٹ کوڈ کے امتزاج کے بارے میں کیا خیال ہے؟ آٹھ امکانات ہیں: 110، 101، 100، 011، 010، 001، 000 اور 111 لیکن آپ پہلے ہی جانتے ہیں کہ آپ 00 یا 01 سے شروع ہونے والی کوئی بھی چیز استعمال نہیں کر سکتے۔ اس سے چار امکانات ختم ہو جاتے ہیں۔ آپ ان میں سے کچھ 3بٹ کوڈز کو اگلے اکثر آنے والے حروف کو تفویض کر سکتے ہیں، S بطور P، 101 بطور A اور 111 یہ باقی حروف کے لیے استعمال کرنے کے لیے سابقہ 100 چھوڑ دیتا ہے۔ آپ 4بٹ کوڈ، 1001 اگلے سب سے زیادہ بار بار آنے والے کردار، جی کے لیے استعمال کرتے ہیں، جو دو بار ظاہر ہوتا ہے۔ دو حروف ہیں جو صرف ایک بار ظاہر ہوتے ہیں، E اور + ان کو 5بٹ کوڈز، 10000 اور 10001 کے ساتھ انکوڈ کیا گیا ہے۔

اس طرح، پورے پیغام کو بطور کوڈ کیا جاتا ہے۔

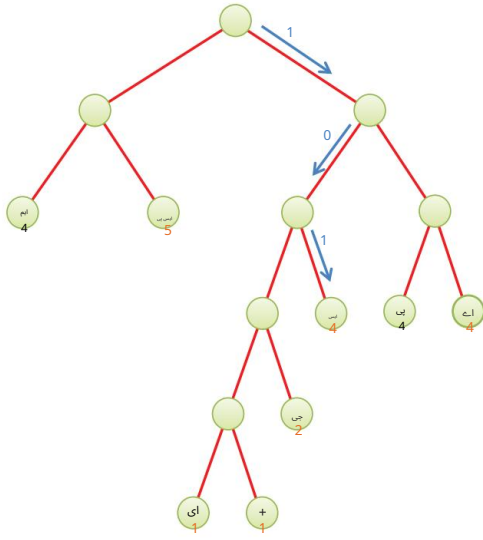
101 110 111 00 01 101 110 111 00 01 101 110 111 00 01 10000 1001 1001 01 10001 01
101 110 111 00

واضح ہونے کے لیے، ہم اس پیغام کو انفرادی حروف کے کوڈز میں توڑ کر دکھاتے ہیں۔ یقیناً، تمام ہٹس ایک ساتھ چلیں گے کیونکہ بائری میس سیج میں کوئی اسپیس کریکٹر نہیں ہے، صرف 0s اور 1s اس سے یہ معلوم کرنا زیادہ مشکل ہو جاتا ہے کہ کون سے ہٹس کسی کردار سے مطابقت رکھتے ہیں۔ تاہم، ہم نکتہ یہ ہے کہ ان ہٹ پیغام میں 25 حروف، جو عام طور پر میموری میں 200 ہٹس (8 × 25) میں محفوظ ہوں گے، بف مین کوڈنگ میں صرف 72 ہٹس کی ضرورت ہوتی ہے۔

بف مین ٹری کے ساتھ ڈی کوڈنگ ہم بعد میں دکھاتے ہیں کہ بف مین کوڈز کیسے بنتے ہیں۔ پہلے، آئیے ڈی کوڈنگ کے کسی حد تک آسان پروسیس کا جائزہ لیں۔ فرض کریں کہ آپ کو پچھلے حصے میں دکھائے گئے ہٹس کی تار موصول ہوئی ہے۔

آپ اسے دوبارہ کرداروں میں کیسے تبدیل کریں گے؟ آپ ایک قسم کا بائری درخت استعمال کر سکتے ہیں جسے بف مین ٹری کہتے ہیں۔ تصویر SPAM 8-27 پیغام کے لیے بف مین ٹری کو دکھاتا ہے جس پر ابھی ابھی بات ہوئی ہے۔

پیغام کے حروف درخت میں لیف نوڈس کے طور پر ظاہر ہوتے ہیں۔ پیغام میں ان کی فریکوئنسی جتنی زیادہ ہوگی، درخت میں وہ اتنے ہی اوپر دکھائی دیں گے۔ ہر لیف نوڈ کے باہر کا نمبر اس کی فریکوئنسی ہے۔ یہ خلائی کردار (sp) کو دوسرے درجے پر رکھتا ہے، اور S، P، A، اور M حروف کو دوسرے یا تیسرے درجے پر رکھتا ہے۔ سب سے کم بار بار، E اور + سب سے نچلی سطح پر ہیں، 5۔



پیغام: SPAM SPAM SPAM EGG + SPAM

	کاوڈز کاؤنٹ	کاوڈز کاؤنٹ	کاوڈز کاؤنٹ
اے	4 111	ہی	4 110
ای	10000	س	4 101
ج	2 1001	خلا	5 01
م	4 00	+	1 10001

تصویر 27-8 سہیم پیغام کے لیے بف مین ٹری

پیغام کو ڈی کوڈ کرنے کے لیے آپ اس درخت کو کیسے استعمال کرتے ہیں؟ آپ پیغام کے پہلے بٹ کو دیکھ کر شروع کرتے ہیں اور درخت کے جڑ نوڈ پر ایک پوائنٹر سیٹ کرتے ہیں۔ اگر آپ کو 0 بٹ نظر آتا ہے، تو آپ پوائنٹر کو نوڈ کے بائیں چائلڈ کی طرف لے جاتے ہیں، اور اگر آپ کو 1 بٹ نظر آتا ہے، تو آپ اسے دائیں منتقل کرتے ہیں۔ اگر identified میں کوئی منسلک کریکٹر نہیں ہے، تو آپ پیغام میں اگلے بٹ پر جائیں گے۔ اسے 5 کے کوڈ کے ساتھ آزمائیں، جو کہ 101 ہے۔ آپ دائیں، بائیں، پھر دائیں پھر جائیں، اور 'voila' آپ اپنے آپ کو 5 نوڈ پر پائیں گے۔ یہ شکل 27-8 میں نیلے تیروں سے دکھایا گیا ہے۔

آپ دوسرے کرداروں کے ساتھ بھی ایسا ہی کر سکتے ہیں۔ لیف نوڈ پر پہنچنے کے بعد، آپ اس کے کریکٹر کو ڈی کوڈ شدہ سٹرنگ میں شامل کر سکتے ہیں اور پوائنٹر کو روٹ نوڈ پر واپس لے جا سکتے ہیں۔
اگر آپ میں صبر ہے تو، آپ اس طرح پوری بٹ سٹرنگ کو ڈی کوڈ کر سکتے ہیں۔

بف مین ٹری بنانا آپ نے دیکھا ہے کہ ڈی کوڈنگ کے لیے بف مین ٹری کا استعمال کیسے کیا جاتا ہے، لیکن آپ اس درخت کو کیسے بناتے ہیں؟

اس مسئلے سے نمٹنے کے بہت سے طریقے ہیں۔ آپ کو بف مین ٹری آجیکٹ کی ضرورت ہے، اور یہ کچھ اس طرح ہے جیسا کہ پہلے بیان کیا گیا ہے کہ اس میں نوڈس ہیں جن میں دو چائلڈ نوڈس ہیں۔ تاہم، یہ بالکل مختلف ہے، کیونکہ معمولات جو تلاش کے درختوں میں کلیدوں کے لیے مخصوص ہیں، جیسے (find(), insert(), delete()) متعلقہ نہیں ہیں۔ یہ پابندی کہ نوڈ کی کلید اس کے بائیں بچے کی کسی بھی کلید سے بڑی ہو اور اس کے دائیں بچے کی کسی بھی کلید کے برابر یا اس سے کم ہو، ہفمین درخت پر لاگو نہیں ہوتا ہے۔ آئیے نئی کلاس HuffmanTree کو کال کریں، اور سرچ ٹری کی طرح، ہر نوڈ پر ایک کلید اور ایک قدر ذخیرہ کریں۔ کلید ڈی کوڈ شدہ میسج کریکٹر جیسے S یا G کو رکھے گی۔ یہ اسپیس کریکٹر ہو سکتا ہے، جیسا کہ آپ نے دیکھا ہے، اور اسے "کوئی کریکٹر" کے لیے ایک خاص قدر کی ضرورت ہے۔

میسیج سٹرنگ سے ہف مین ٹری بنانے کا الگورتھم یہ ہے:

تیاری

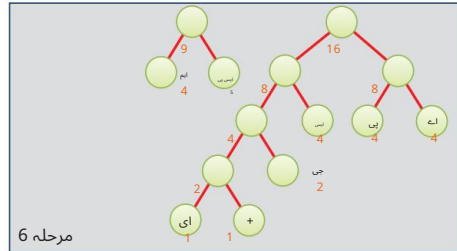
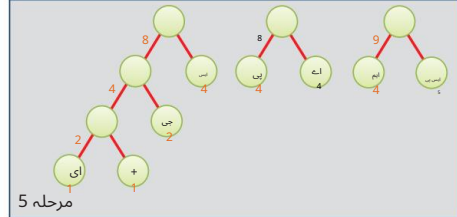
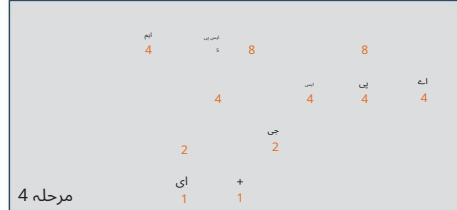
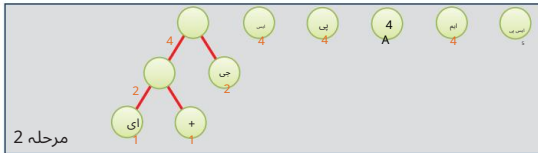
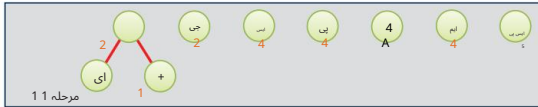
1. شمار کریں کہ پیغام کی تار میں ہر حرف کتنی بار ظاہر ہوتا ہے۔

2. پیغام میں استعمال ہونے والے ہر کردار کے لیے Huffman Tree آبجیکٹ بنائیں۔ سپیم کے لیے

پیغام کی مثال، وہ آٹھ درخت ہوں گے۔ ہر درخت میں ایک نوڈ ہوتا ہے جس کی کلید ایک کریکٹر ہے اور جس کی قیمت پیغام میں اس کریکٹر کی فریکوئنسی ہے۔ وہ values 3 یا 8 یا ٹیبل 8-4 میں اسپام پیغام کے لیے مل سکتے ہیں۔

3. ان درختوں کو ترجیحی قطار میں داخل کریں (جیسا کہ باب 4 میں بیان کیا گیا ہے)۔ انہیں فریکوئنسی (ہر روٹ نوڈ کی قدر کے طور پر ذخیرہ کیا جاتا ہے) اور درخت میں سطحوں کی تعداد کے حساب سے ترتیب دیا جاتا ہے۔ سب سے چھوٹی فریکوئنسی والا درخت سب سے زیادہ ترجیح رکھتا ہے۔ مساوی تعدد والے درختوں میں، زیادہ درجے والے درخت کو سب سے زیادہ ترجیح دی جاتی ہے۔ دوسرے لفظوں میں، جب آپ ترجیحی قطار سے کسی درخت کو ہٹاتے ہیں، تو یہ ہمیشہ سب سے کم استعمال شدہ کردار کا سب سے گہرا درخت ہوتا ہے۔ (درخت کی گہرائی کا استعمال کرتے ہوئے تعلقات کو توڑنا، حتمی Huffman درخت کے توازن کو بہتر بناتا ہے۔)

یہ تیاری مکمل کرتا ہے، جیسا کہ شکل 8-28 کے مرحلہ 0 میں دکھایا گیا ہے۔ ہر ایک نوڈ ہف مین ٹریز میں نوڈ کے بیچ میں ایک کریکٹر دکھایا گیا ہے اور نوڈ کے نیچے اور بائیں طرف ایک فریکوئنسی ویلیو دکھائی گئی ہے۔

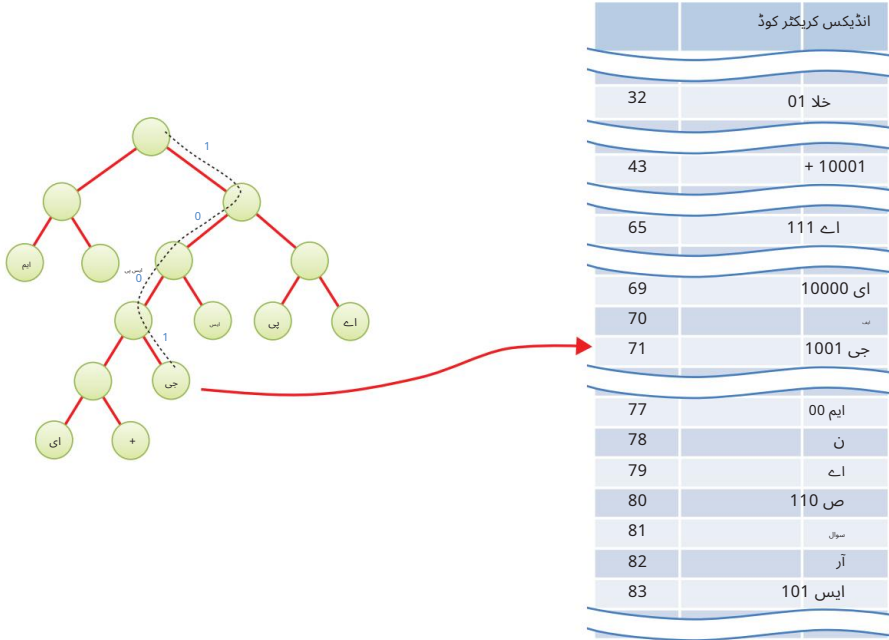


تصویر 8-28 ہف مین درخت کو اگانا، پہلے چھ مراحل

ٹیبل میں کوڈز کو بھرنے کے لیے، آپ ہف مین ٹری کو عبور کرتے ہیں، ہر نوڈ کے راستے پر نظر رکھتے ہوئے جب اس کا دورہ کیا جاتا ہے۔ جب آپ کسی لیف نوڈ پر جاتے ہیں، تو آپ اس نوڈ کی کلید کو ٹیبل کے اشاریہ کے طور پر استعمال کرتے ہیں اور سیل کی قدر میں ہائیری سٹرنگ کے طور پر راستہ داخل کرتے ہیں۔ ہر سیل میں کوڈ نہیں ہوتا ہے۔ صرف وہی جو پیغام میں ظاہر ہوتے ہیں۔ شکل 30-8 دکھاتا ہے کہ یہ سپیم پیغام کے لیے کیسا لگتا ہے۔ ٹیبل کو صرف اہم قطاروں کو دکھانے کے لیے مختصر کیا گیا ہے۔

کریکٹر G کے لیے لیف نوڈ کا راستہ دکھایا گیا ہے کیونکہ درخت کو عبور کیا جا رہا ہے۔

مکمل کوڈ ٹیبل ایک ایسے طریقہ کو کال کر کے بنایا جا سکتا ہے جو جڑ سے شروع ہوتا ہے اور پھر ہر جگہ کے لیے بار بار کال کرتا ہے۔ آخر کار، تمام لیف نوڈس کے راستے تلاش کیے جائیں گے، اور کوڈ ٹیبل مکمل ہو جائے گا۔



تصویر 30-8 کوڈ ٹیبل بنانا

غور کرنے کے لیے ایک اور چیز: اگر آپ کو ایک ہائیری پیغام موصول ہوتا ہے جسے ہف مین کوڈ کے ساتھ کمپریس کیا گیا ہے، تو آپ کو کیسے معلوم ہوگا کہ اسے ڈی کوڈ کرنے کے لیے ہف مین ٹری کو کس طرح استعمال کرنا ہے؟ جواب یہ ہے کہ ہف مین ٹری کو پہلے، ہائیری پیغام سے پہلے، کسی ایسے فارمیٹ میں بھیجا جانا چاہیے جس کے لیے پیغام کے مواد کے علم کی ضرورت نہیں ہے۔ یاد رکھیں کہ ہف مین کوڈز ڈیٹا کو کمپریس کرنے کے لیے ہیں، اسے خفیہ کرنے کے لیے نہیں۔ ہف مین ٹری کی مختصر تفصیل بھیجنے کے بعد لمبے پیغام کا کمپریسڈ ورژن بہت سے بٹس بچاتا ہے۔

خلاصہ

- درخت کناروں سے جڑے ہوئے نوڈس پر مشتمل ہوتے ہیں۔
- جڑ درخت میں سب سے اوپر کی نوڈ ہے؛ اس کا کوئی والدین نہیں ہے۔
- تمام نوڈس کے علاوہ درخت میں جڑ کے بالکل ایک پیرنٹ ہوتے ہیں۔
- ایک بائنری درخت میں، ایک نوڈ کے زیادہ سے زیادہ دو بچے ہوتے ہیں۔
- درخت میں لیف نوڈس میں چائلڈ نوڈس نہیں ہوتے اور جڑ تک بالکل ایک راستہ ہوتا ہے۔
- غیر متوازن درخت وہ ہوتا ہے جس کی جڑ میں دائیں سے زیادہ بائیں نسل کی اولاد ہوتی ہے۔ اولاد، یا اس کے برعکس۔
- درخت کا ہر نوڈ کچھ ڈیٹا محفوظ کرنا ہے۔ ڈیٹا میں عام طور پر ایک کلیدی قدر ہوتی ہے جسے شناخت کرنے کے لیے استعمال کیا جاتا ہے۔ اسے بتانا۔
- کناروں کو عام طور پر نوڈ کے بچوں کے حوالے سے ظاہر کیا جاتا ہے۔ کم com
- ایک نوڈ سے اس کے والدین کے حوالہ جات ہیں۔
- درخت کو عبور کرنے کا مطلب ہے اس کے تمام نوڈس کو کچھ پہلے سے طے شدہ ترتیب میں دیکھنا۔
- سب سے آسان ٹراورسلز پری آرڈر، ان آرڈر اور پوسٹ آرڈر ہیں۔
- پری آرڈر اور پوسٹ آرڈر ٹراورسلز الجبری تاثرات کو پارس کرنے کے لیے مفید ہیں۔
- بائنری تلاش کے درخت
- ایک بائنری سرچ ٹری میں، تمام نوڈس جو نوڈ A کی اولاد چھوڑے گئے ہیں
- کلیدی قدریں A سے کم؛ تمام نوڈس جو A کی صحیح اولاد ہیں ان کی کلیدی قدریں A سے زیادہ (یا اس کے برابر) ہیں۔
- ثنائی تلاش کے درخت $O(\log N)$ میں تلاش، اندراج، اور حذف کرتے ہیں
- وقت
- بائنری سرچ ٹری میں نوڈ کی تلاش میں ایک نوڈ کی کلیدی قدر سے ملنے والی گول کلید کا موازنہ کرنا اور اگر گول کی کم ہے تو اس نوڈ کے بائیں چائلڈ کے پاس جانا یا اگر گول کلید ہے تو نوڈ کے دائیں بچے کی طرف جانا شامل ہے۔ زیادہ
- داخل کرنے میں نیا نوڈ داخل کرنے کے لیے جگہ تلاش کرنا اور پھر اس کا حوالہ دینے کے لیے چائلڈ فییلڈ کو اس کے نئے پیرنٹ (یا درخت کی جڑ) میں تبدیل کرنا شامل ہے۔
- ایک ان آرڈر ٹراورسل وزٹ نوڈس کو چڑھنے والی کلیدوں کی ترتیب میں۔
- جب نوڈ کے کوئی بچے نہیں ہوتے ہیں، تو آپ چائلڈ فییلڈ کو صاف کر کے اسے حذف کر سکتے ہیں۔
- اس کے والدین (مثال کے طور پر، اسے Python میں None پر سیٹ کرنا)۔
- جب نوڈ میں ایک بچہ ہوتا ہے، تو آپ اس میں چائلڈ فییلڈ سیٹ کر کے اسے حذف کر سکتے ہیں۔
- والدین اپنے بچے کی طرف اشارہ کریں۔
- جب کسی نوڈ کے دو بچے ہوتے ہیں، تو آپ اسے اس کے successor سے تبدیل کر کے اور subtree سے جانشین کو حذف کر کے اسے حذف کر سکتے ہیں۔
- آپ A میں کم از کم نوڈ تلاش کر کے نوڈ A کے جانشین کو تلاش کر سکتے ہیں۔
- دائیں ذیلی درخت۔

□□ ڈپلیکیٹ کلیدی اقدار والے نوڈس کو اضافی کوڈنگ کی ضرورت ہوتی ہے کیونکہ عام طور پر صرف ان میں سے ایک (پہلا) ایک تلاش میں پایا جاتا ہے، اور اپنے بچوں کو منظم کرنے سے اندراج اور حذف کرنے کی سازش ہوتی ہے۔

□□ درختوں کو کمپیوٹر کی میموری میں ایک صف کے طور پر پیش کیا جا سکتا ہے، حالانکہ حوالہ پر مبنی نقطہ نظر زیادہ عام اور میموری کو موثر ہے۔

□□ برف مین ٹری ایک ہائیری ٹری ہے (لیکن سرچ ٹری نہیں) جسے ڈیٹا کمپریشن الگورتھم میں استعمال کیا جاتا ہے جسے برف مین کوڈنگ کہتے ہیں۔

□□ برف مین کوڈ میں، جو حروف اکثر نظر آتے ہیں ان کو سب سے کم ہٹس کے ساتھ کوڈ کیا جاتا ہے، اور جو شاذ و نادر ہی نظر آتے ہیں ان کو سب سے زیادہ ہٹس کے ساتھ کوڈ کیا جاتا ہے۔

□□ برف مین ٹری میں راستے ہر ایک لیف نوڈس کے لیے کوڈ فراہم کرتے ہیں۔

□□ لیف نوڈ کی سطح اس کی کلید کے لیے کوڈ میں استعمال ہونے والے ہٹس کی تعداد کی نشاندہی کرتی ہے۔

□□ برف مین کوڈڈ پیغام میں کم سے کم کثرت سے ظاہر ہونے والے حروف کو برف مین درخت کی گہری سطح پر لیف نوڈس میں رکھا جاتا ہے۔

سوالات ان سوالات کا مقصد قارئین کے لیے خود امتحان ہے۔ جوابات ضمیمہ سی میں مل سکتے ہیں۔

1. ہائیری سرچ ٹری میں اندراج اور حذف کرنے کے لیے کیا بگ O وقت درکار ہوتا ہے؟

2. ایک ہائیری درخت ایک تلاش درخت ہے اگر

a ہر نان لیف نوڈ میں ایسے بچے ہوتے ہیں جن کی کلیدی قدریں اس سے کم یا برابر ہوتی ہیں۔
والدین

b ہر نون لیف نوڈ کی کلیدی اقدار کی کلیدوں کا مجموعہ یا مجموعہ ہے۔
اس کے بچے

c ہر ہائیری بچے کی کلید اس کے والدین سے کم ہوتی ہے اور ہر دائیں بچے کے پاس اس کے والدین سے بڑی یا اس کے برابر کی ہوتی ہے۔

d جڑ سے لے کر ہر لیف نوڈ تک کے راستے میں، ہر نوڈ کی کلید اس سے بڑی ہوتی ہے۔
یا اس کے والدین کی کلید کے برابر۔

3. صحیح یا غلط: اگر آپ درخت کو عبور کرتے ہیں اور ہر نوڈ کے راستے کو ایک سلسلہ کے طور پر پرنٹ کرتے ہیں حروف اور R اس بات کے لیے کہ آیا راستہ ہر قدم پر بائیں یا دائیں بچے کی پیروی کرتا ہے، کچھ ڈپلیکیٹ راستے ہو سکتے ہیں۔

4. جب ترتیب شدہ صف میں ڈیٹا کو ذخیرہ کرنے کے مقابلے میں، اس میں ذخیرہ کرنے کا بنیادی فائدہ ایک ہائیری تلاش درخت ہے

a بگ O نوٹیشن میں ٹراورسل ٹائم جیسا ہی سرچ ٹائم ہونا۔

b ہر آئٹمز ڈالنے یا حذف کرتے وقت ڈیٹا کا پی کرنے کی ضرورت نہیں ہے۔

c O(log N) وقت میں کسی چیز کو تلاش کرنے کے قابل ہونا۔

d ایک کلید ہونا جو کلید کے ذریعہ شناخت شدہ قدر سے الگ ہے۔

5.20 نوڈس کے ساتھ ایک مکمل، متوازن بائنری درخت میں، اور جڑ کو سمجھا جاتا ہے۔
سطح، 0 سطح پر کتنے نوڈس ہیں؟

6. بائنری درخت کے ذیلی درخت میں ہمیشہ ہوتا ہے۔

a ایک جڑ جو اہم درخت کی جڑ کا بچہ ہے۔

b ایک جڑ جو مرکزی درخت کی جڑ سے جڑی ہوئی ہے۔

c مرکزی درخت سے کم نوڈس۔

d نوڈس کی مساوی یا بڑی تعداد کے ساتھ ایک بہن بھائی۔

افہام طور پر sepa ہیں _____ جاتی ہے۔
7. درختوں کو بطور آبجیکٹ لاگو کرتے وقت، درجہ بندی کی

8. بائنری سرچ ٹری میں نوڈ تلاش کرنے میں نوڈ سے نوڈ تک جانا، پوچھنا شامل ہے

a تلاش کی کلید کے سلسلے میں نوڈ کی کلید کتنی بڑی ہے۔

b نوڈ کی کلید اس کے دائیں یا بائیں بچے کی کلید سے کتنی بڑی ہے۔

c آپ کس لیف نوڈ تک پہنچنا چاہتے ہیں۔

d چاہے آپ جس سطح پر ہیں وہ سرچ کلید کے اوپر ہے یا نیچے۔

9. ایک غیر متوازن درخت ایک ہے۔

a جس میں زیادہ تر کیڑ کی قدریں اوسط سے زیادہ ہوتی ہیں۔

b جہاں مرکزی نوڈ کے اوپر نیچے سے زیادہ نوڈس ہیں۔

c جہاں لیف نوڈس اپنے برابر کے بائیں بچے کے طور پر زیادہ کثرت سے ظاہر ہوتے ہیں۔

d صحیح بچے کے طور پر، یا اس کے برعکس۔

d جس میں روٹ یا کسی اور نوڈ میں دائیں اولاد کے مقابلے میں بہت زیادہ بائیں اولاد ہوتی ہے، یا اس کے برعکس۔

10. صحیح یا غلط: ایک درجہ بندی کا فائل سسٹم بنیادی طور پر ایک بائنری سرچ ٹری ہے، حالانکہ یہ
غیر متوازن ہو سکتا ہے۔

11. نوڈ داخل کرنا انہی مراحل سے شروع ہونے لگتا ہے نوڈ _____

12. درختوں کے ڈیٹا ڈھانچے کو عبور کرنا

a مختلف ٹراورسل آرڈرز کو سنبھالنے کے لیے متعدد طریقوں کی ضرورت ہوتی ہے۔

b تکراری افعال یا جنریٹرز کا استعمال کرتے ہوئے لاگو کیا جا سکتا ہے۔

c سرنی ڈیٹا ڈھانچے کو عبور کرنے سے کہیں زیادہ تیز ہے۔

d اشیاء کے نرم حذف کو عملی بنانے کا ایک طریقہ ہے۔

13. جب ایک درخت انتہائی غیر متوازن ہوتا ہے تو وہ اس طرح برتاؤ کرنے لگتا ہے۔
ڈیٹا _____

ساخت

14. فرض کریں کہ ایک نوڈ A کے پاس ہائری سرچ ٹری میں جانشین نوڈ S کے جس کا کوئی ڈپلیکیٹ نہیں ہے لیکن اس سے چھوٹا یا اس کے برابر ہے۔ _____ چابیاں پھر S کے پاس ایک کلید ہونی چاہیے جو اس سے بڑی ہو۔

15. ہائری سرچ ٹری میں نوڈس کو حذف کرنا پیچیدہ ہے کیونکہ

a جانشین کے نیچے ذیلی درختوں کو کاپی کرنے کے لیے ایک اور ٹراورسل کی ضرورت ہوتی ہے۔

ب جانشین تلاش کرنا مشکل ہے، خاص طور پر جب درخت غیر متوازن ہو۔

c درخت ایک سے زیادہ درختوں، ایک جنگل میں تقسیم ہو سکتا ہے، اگر یہ صحیح طریقے سے نہیں کیا جاتا ہے۔

d نوڈ کے حذف کیے جانے والے چائلڈ نوڈس کی مختلف تعداد کے لیے آپریشن بہت مختلف ہے، 1، 0 یا 2۔

16. ایک ہائری درخت میں جو ریاضی کے اظہار کی نمائندگی کرتا ہے،

a آپریٹر نوڈ کے دونوں بچے آپرینڈز ہونے چاہئیں۔

ب پوسٹ آرڈر ٹراورسل کے بعد، قوسین کو شامل کرنا ضروری ہے۔

c پری آرڈر ٹراورسل کے بعد، قوسین کو شامل کرنا ضروری ہے۔

d پری آرڈر ٹراورسل میں، ایک نوڈ کو اس کے بچوں میں سے کسی سے پہلے دیکھا جاتا ہے۔

17. جب ایک درخت کو ایک صف سے ظاہر کیا جاتا ہے، انڈیکس n میں نوڈ کے دائیں بچے کے پاس ایک ہوتا ہے

کی انڈیکس _____

18. صحیح یا غلط: ہائری سرچ ٹری سے ایک بچے کے ساتھ نوڈ کو حذف کرنے میں اس نوڈ کے جانشین کو تلاش کرنا شامل ہے۔

19. ایک Huffman درخت عام طور پر استعمال کیا جاتا ہے۔ _____

20. مندرجہ ذیل میں سے کون سا ہف مین درخت کے بارے میں درست نہیں ہے؟

a سب سے زیادہ استعمال ہونے والے حروف ہمیشہ درخت کی چوٹی کے قریب ظاہر ہوتے ہیں۔

ب عام طور پر، کسی پیغام کو ڈی کوڈ کرنے میں بار بار جڑ سے کسی راستے پر چلنا شامل ہوتا ہے۔
ایک پتی کو۔

c ایک کردار کو کوڈنگ میں، آپ عام طور پر ایک پتی سے شروع کرتے ہیں اور اوپر کی طرف کام کرتے ہیں۔

d درخت کو ترجیحی بنیاد پر ہٹانے اور داخل کرنے کی کارروائیوں سے پیدا کیا جا سکتا ہے۔

چھوٹے درختوں کی قطار۔

تجربات

ان تجربات کو انجام دینے سے باب میں شامل موضوعات کے بارے میں بصیرت فراہم کرنے میں مدد ملے گی۔ کوئی پروگرامنگ شامل نہیں ہے۔

8-A ہائری سرچ ٹری ویڈیو لائزیشن ٹول کا استعمال کر کے 20 بے ترتیب درختوں کو 20 اشیاء کی مطلوبہ تعداد کے طور پر استعمال کریں۔ آپ کتنے فیصد کو سنجیدگی سے غیر متوازن کہیں گے؟

8-B فہرست 12-8 اور پرو میں دکھایا گیا BinarySearchTreeTester.py پروگرام استعمال کریں

درج ذیل تجربات کرنے کے لیے پبلشر کی ویب سائٹ سے کوڈ کی مثالوں کے ساتھ: vided:

a ایک نوڈ کو حذف کریں جس کے کوئی بچے نہیں ہیں۔

b ایک نوڈ کو حذف کریں جس میں 1 چائلڈ نوڈ ہو۔

c ایک نوڈ کو حذف کریں جس میں 2 چائلڈ نوڈس ہوں۔

داخل کرنے کے لیے نئے نوڈ کے لیے ایک کلید چنیں۔ اس بات کا تعین کریں کہ آپ کے خیال میں اسے درخت میں کہاں ڈالا جائے گا، اور پھر اسے پروگرام کے ساتھ داخل کریں۔ کیا یہ طے کرنا آسان ہے کہ یہ کہاں جائے گا؟

e پچھلے مرحلے کو کسی اور کلید کے ساتھ دہرائیں لیکن اسے دوسرے بچے میں ڈالنے کی کوشش کریں۔

شاخ مثال کے طور پر، اگر آپ کا پہلا نوڈ بائیں بچے کے طور پر ڈالا گیا تھا، تو کوشش کریں کہ ایک کو دائیں بچے کے طور پر یا دائیں ذیلی درخت میں ڈالیں۔

8-C فہرست 12-8 میں دکھایا گیا BinarySearchTreeTester.py پروگرام آئٹمز کے داخل کرنے کے آرڈر کی بنیاد پر 7 سطحوں پر 11 نوڈس کے ابتدائی درخت کو پرنٹ کرتا ہے۔ درخت کے مکمل طور پر متوازن ورژن میں 4 سطحوں پر ایک جیسے نوڈس محفوظ ہوں گے۔ درخت کو صاف کرنے کے لیے پروگرام کا استعمال کریں، اور پھر اس بات کا تعین کریں کہ بال اینسڈ ٹری بنانے کے لیے وہی چابیاں کس ترتیب میں ڈالنی ہیں۔ اپنا آرڈر کرنے کی کوشش کریں اور دیکھیں کہ کیا درخت متوازن طور پر نکلتا ہے۔ اگر نہیں، تو ایک اور آرڈر کرنے کی کوشش کریں۔ کیا آپ چند جملوں میں داخل کرنے کی ترتیب کو بیان کر سکتے ہیں جو ہمیشہ چابیاں کے مخصوص سیٹ سے ایک متوازن بائٹری سرچ ٹری بنائے گا؟

8-D بائٹری سرچ ٹری ویژولائزیشن ٹول استعمال کریں تاکہ ہر ممکن طریقے سے نوڈ کو حذف کیا جا سکے۔

صورت حال

پروگرامنگ پروجیکٹس

پروگرامنگ پروجیکٹس کو حل کرنے کے لیے پروگرام لکھنا مواد کے بارے میں آپ کی سمجھ کو مستحکم کرنے میں مدد کرتا ہے اور یہ ظاہر کرتا ہے کہ باب کے تصورات کو کس طرح لاگو کیا جاتا ہے۔ (جیسا کہ تعارف میں بتایا گیا ہے، اہل اساتذہ پبلشر کی ویب سائٹ پر پروگرامنگ پروجیکٹس کے مکمل حل حاصل کر سکتے ہیں۔)

8.1 اس باب میں بیان کردہ BinarySearchTree کلاس کو تبدیل کریں تاکہ ڈوپلی کیٹ کیز کے ساتھ نوڈس کی اجازت دی جا سکے۔ تین طریقے متاثر ہوئے ہیں: insert(), find() اور delete()۔ ڈوپلی کیٹ کیز کو مساوی کلیدوں کے درمیان سب سے کم سطح پر داخل کرنے کا انتخاب کریں، جیسا کہ شکل 8-26 کے بائیں جانب دکھایا گیا ہے، اور ہمیشہ برابر کیز کے درمیان سب سے گہری تلاش کریں اور حذف کریں۔ مزید خاص طور پر، find() اور search() طریقوں کو مساوی کلیدوں کے درمیان گہری test کو واپس کرنا چاہئے جس کا سامنا کرنا پڑتا ہے لیکن ایک اختیاری پیرامیٹر کو سب سے کم تلاش کرنے کی وضاحت کرنے کی اجازت دینی چاہئے۔ insert() طریقہ کار کو اس صورت میں تبدیل کرنا چاہیے جب داخل کی جانے والی آئٹم موجودہ نوڈ کو ڈوپلی کیٹ بناتی ہے، ایک نیا نوڈ ڈال کر گہری ڈوپلی کیٹ کلید کے نیچے خالی بائیں بچے کے ساتھ۔ ڈیلیٹ (طریقہ کو ڈوپلی کیٹ کیز کے درمیان سب سے گہرے نوڈ کو حذف کرنا چاہیے، اس طرح ڈوپلی کیٹ کیز کے درمیان LIFO یا اسٹیک جیسا برتاؤ فراہم کرتا ہے۔ حذف کرنے کے معاملات کے بارے میں احتیاط سے سوچیں اور آیا

398 باب 8 بانٹری درخت

جانشین نوڈس کا انتخاب مختلف اقدار سے وابستہ کئی ڈپلیکیٹ کلیدیوں داخل کرنے والے درخت پر آپ کا نفاذ کیسے کام کرتا ہے اس کا مظاہرہ کریں۔ پھر ان کیز کو حذف کریں اور ان کی قدریں دکھائیں تاکہ یہ واضح ہو جائے کہ آخری ڈپلیکیٹ داخل کیا گیا پہلا ڈپلیکیٹ حذف کیا گیا ہے۔

8.2 ایک پروگرام لکھیں جو پوسٹ فکس ایکسپریشن پر مشتمل سٹرنگ لے اور الجبری ایکسپریشن کی نمائندگی کرنے کے لیے ایک بانٹری ٹری بنائے جیسا کہ شکل 8-16 میں دکھایا گیا ہے۔

آپ کو BinaryTree کلاس کی ضرورت ہے، جیسے BinarySearchTree کی، لیکن بغیر کسی کلید یا نوڈس کی ترتیب کے۔ find(), insert(), اور delete() طریقوں کی بجائے، آپ کو سنگل نوڈ BinaryTrees بنانے کی اہلیت کی ضرورت ہے جس میں ایک واحد آپرینڈ ہو اور دو بانٹری ٹریوں کو ملا کر تیسرا آپریٹر کے ساتھ روٹ نوڈ کے طور پر بنایا جائے۔ آپریٹرز اور آپرینڈز کا نحو وہی ہے جو باب 4 کے PostfixTranslate.py ماڈیول میں استعمال کیا گیا تھا۔ آپ ان پٹ سٹرنگ کو آپریٹر اور آپرینڈ ٹوکنز میں پارس کرنے کے لیے اس ماڈیول میں NextToken() فنکشن کا استعمال کر سکتے ہیں۔ آپ کو حد بندی کے طور پر قوسین کی ضرورت نہیں ہے کیونکہ پوسٹ فکس اظہار ان کا استعمال نہیں کرتے ہیں۔

تصدیق کریں کہ ان پٹ ایکسپریشن ایک ہی الجبری ایکسپریشن تیار کرتا ہے اور اگر ایسا نہیں ہوتا ہے تو ایک استثناء بڑھائیں۔ درست الجبری بانٹری درختوں کے لیے، آؤٹ پٹ فارمز میں ان پٹ کا ترجمہ کرنے کے لیے درخت کے پری، ان- اور پوسٹ آرڈر ٹراورسلز کا استعمال کریں۔ آؤٹ پٹ ترجمہ میں آپریٹر کی ترجیح کو واضح کرنے کے لیے ان آرڈر ٹراورسل کے لیے قوسین کو شامل کریں۔ اپنے پروگرام کو کم از کم درج ذیل تاثرات پر چلائیں:

a 91 95 + 15 + 19 + 4 *

b بی بی * اے سی * 4

c 42

A 57 اس میں ایک استثناء پیدا ہونا چاہئے۔

+ / اس سے ایک استثناء پیدا ہونا چاہئے۔

8.3 ہف مین کوڈنگ اور ڈی کوڈنگ کو لاگو کرنے کے لیے ایک پروگرام لکھیں۔ اسے کرنا چاہیے۔

درج ذیل:

□ ٹیکسٹ میسج (سٹرنگ) کو قبول کریں۔

□ اس پیغام کے لیے ہف مین ٹری بنائیں۔

□ کوڈ ٹیبل بنائیں۔

□ ٹیکسٹ میسج کو بانٹری میں انکوڈ کریں۔

□ بانٹری پیغام کو واپس متن میں ڈی کوڈ کریں۔

□ بانٹری پیغام میں ہس کی تعداد اور ان پٹ پیغام میں حروف کی تعداد دکھائیں۔

اگر پیغام مختصر ہے، تو پروگرام کو تخلیق کرنے کے بعد ہف مین ٹری کو ظاہر کرنے کے قابل ہونا چاہیے۔ آپ بانٹری پیغامات کو ذخیرہ کرنے کے لیے Python سٹرنگ متغیرات کو حروف 1 اور 0 کی ترتیب کے طور پر استعمال کر سکتے ہیں۔ ہانٹ رے کا استعمال کرتے ہوئے اصل ہٹ بیرا پھیری کرنے کی فکر نہ کریں جب تک کہ آپ واقعی ایسا نہ کریں۔ کوڈ ٹیبل بنانے کا آسان ترین طریقہ

Python میں ڈکشنری (dict) ڈیٹا ٹائپ استعمال کرنا ہے۔ اگر یہ ناواقف ہے تو، یہ بنیادی طور پر ایک صف ہے جسے سٹرنگ یا ایک کریکٹر کے ذریعے ترتیب دیا جا سکتا ہے۔ یہ BinarySearchTreeTester.py ماڈیول میں استعمال ہوتا ہے جو لسٹنگ 8-12 میں نقشہ کمانڈ کرنے کے لیے دکھائے گئے ہیں۔ اگر آپ انٹیجر انڈیکسڈ صف کو استعمال کرنے کا انتخاب کرتے ہیں، تو آپ ایک حرف کو عدد میں تبدیل کرنے کے لیے Python کے ord() فنکشن کا استعمال کر سکتے ہیں لیکن آگاہ رہیں کہ اگر آپ پیغام میں صوابدیدی یونیکوڈ حروف جیسے ایموجیز (☺) کی اجازت دیتے ہیں تو آپ کو ایک بڑی صف کی ضرورت ہوگی۔

8.4 درخت کے توازن کی پیمائش مشکل ہو سکتی ہے۔ آپ دو آسان اقدامات کر سکتے ہیں: نوڈ بیلنس اور لیول (یا اونچائی) بیلنس۔ جیسا کہ پہلے ذکر کیا گیا ہے، متوازن درختوں کے بائیں اور دائیں ذیلی درختوں میں تقریباً مساوی تعداد میں نوڈس ہوتے ہیں۔ اسی طرح، بائیں اور دائیں ذیلی درختوں میں سطحوں (یا اونچائی) کی تقریباً برابر تعداد ہونی چاہیے۔ درج ذیل طریقے لکھ کر BinarySearchTree کلاس کو بڑھائیں:

نوڈ بیلنس - (دائیں ذیلی درخت میں نوڈس کی تعداد کو بائیں ذیلی درخت میں نوڈس کی تعداد کو کم کرتا ہے

ب لیول بیلنس() — دائیں سب ٹری میں لیولز کی تعداد کا حساب لگانا ہے مائنس بائیں سب ٹری میں لیولز کی تعداد

c غیر متوازن نوڈس (بائی) — (1 = نوڈ کیز کی ایک فہرست لوٹاتا ہے جہاں بیلنس میٹرکس میں سے کسی ایک کی مطلق قدر حد سے تجاوز کر جاتی ہے، جو پہلے سے طے شدہ 1 ہوتی ہے۔

ان تینوں طریقوں کے لیے (بار بار آنے والے) مددگار طریقوں کی ضرورت ہوتی ہے جو درخت کے اندر نوڈس پر چڑے ذیلی درختوں کو عبور کرتے ہیں۔ ایک متوازن درخت میں، غیر متوازن نوڈس کی فہرست خالی ہوگی۔ ایک خالی BinarySearchTree میں کلیدوں کی درج ذیل چار فہرستیں ڈال کر (ترتیب میں، بائیں سے دائیں)، نتیجے میں آنے والے 15-نوڈ ٹری کو پرنٹ کر کے، نتیجے میں آنے والے روٹ نوڈ کے نوڈ اور لیول بیلنس کو پرنٹ کر کے، اور پھر فہرست کو پرنٹ کر کے اپنے اقدامات کو آزمائیں۔ 1 by اور 2 by= ساتھ غیر متوازن کلیدوں کا۔

13, 15], [8, 4], 2, 3, 1, 6, 5, 7, 12, 10, 9, 11, 14, 13, 8.5]
9, 11, 14, 13, 15], [8, 4, 2, 3, 1, 6, 5, 7, 12, 10, 9, 11, 14,
8, 12, 10, 9, 11, 14, 13, 15], [8, 4, 5, 6, 7, 3, 2, 1, 12], 10,
[7, 6, 5, 4, 3, 2, 1,

8.5 ہر بائیں درخت کو ایک صف کے طور پر دکھایا جا سکتا ہے، جیسا کہ عنوان والے حصے میں بیان کیا گیا ہے۔ "درختوں کو صفوں کے طور پر پیش کیا گیا ہے۔" ایک صف کو درخت کے طور پر پیش کرنے کا الٹ، تاہم، صرف کچھ صفوں کے لیے کام کرتا ہے۔ درخت کے گم شدہ نوڈس کو صف کے خلیوں میں کچھ پہلے سے طے شدہ قدر کے طور پر پیش کیا جاتا ہے — جیسے کہ کوئی نہیں — جو کہ درخت کے نوڈ میں ذخیرہ شدہ قدر نہیں ہو سکتی۔ اگر صف میں روٹ نوڈ غائب ہے، تو متعلقہ درخت نہیں بنایا جا سکتا۔ ایک فنکشن لکھیں جو ایک صف کو بطور ان پٹ لیتا ہے اور اس کے مواد سے بائیں ٹری بنانے کی کوشش کرتا ہے۔ ہر وہ سیل جو None نہیں ہے ایک ٹری نوڈ پر ذخیرہ کرنے کی قدر ہے۔

جب آپ پیرنٹ نوڈ کے بغیر کسی نوڈ کے سامنے آتے ہیں (روٹ نوڈ کے علاوہ)، فنکشن کو ایک استثناء اٹھانا چاہئے جس سے یہ ظاہر ہوتا ہے کہ درخت نہیں بنایا جا سکتا۔ نوٹ کریں کہ نتیجہ ضروری نہیں کہ بائیں سرچ ٹری ہو، صرف ایک بائیں ٹری۔ اشارہ: لیف نوڈس سے جڑ تک کام کرنا آسان ہے، ہر اس سیل کے لیے نوڈس بنانا جو کوئی نہیں ہے اور نتیجے میں آنے والے نوڈ کو دوبارہ ان پٹ ارے کے اسی سیل میں اسٹور کرنا

انڈیکس

نمبرز

2-3 درخت

بیان کیا گیا، 432-433

نوڈ سپلٹس، 433-434

اندراج اور، 437-438

اندرونی نوڈس کو فروغ دینا، 438 کی 435-437 رفتار

2-3-4 درخت

فائدے/نقصانات، 5

تعریف، 401-403

اندراج، 404-405

نوڈ تقسیم، 407-408، 405-406 تنظیم، 404

403-

سرخ سیاہ درخت اور 508-510

آپریشنل مساوات، 510-514

509-510 کے درمیان تبدیلی

جڑوں کی تقسیم، 407-406 تلاش، 404

کی رفتار، 431 اسٹوریج کی ضروریات،

432 اصطلاحات، 403

درخت 234 کلاس، 415

نوڈس کو حذف کرنا، 423-430

نوڈ کلاس، 412-415

نوڈ تقسیم، 418-421 تلاش،

417-415 گزرنا، 421-423

234 Tree ویژولائزیشن ٹول، 408-411

اے

	anagrams بیان کیا گیا، 239-242 مسئلہ کا
	تجزیہ، 814-818
خلاصہ ڈیٹا کی اقسام ADTs دیکھیں (خلاصہ ڈیٹا اقسام)	ڈیٹا کی مقدار، 815-816
خلاصہ، بیان کیا گیا، 190-191	آپریشنز کی فریکوئنسی، 816-817 سافٹ ویئر کی دیکھ بھال کی ذمہ داریاں، 817-818
ملحقہ، وضاحت شدہ، 707	
ملحقہ فہرستیں	ڈیٹا کی اقسام، 814-815
ماڈلنگ، 712-713	دلائل، وضاحت شدہ، 20 ریاضی کے تاثرات، تجزیہ
کناروں کو ذخیرہ کرنا، 714-716	
ملحقہ میٹرکس	بیان کیا گیا، 132-133
بطور پیش ٹیبل، 712	پوسٹ فکس ایکسپریشنز کا جائزہ لینا، 148-151
دو جہتی صف کے طور پر، 710-711	انفکس کیلکولیٹر ٹول، 142-148
ADT فہرستیں، 191	پوسٹ فکس اشارے، 133-134 انفکس کو پوسٹ
ADTs (خلاصہ ڈیٹا کی اقسام)	فکس میں ترجمہ کرنا، 134-142
بیان کردہ، 184، 190-191	ارے کلاس تک رسائی کے عناصر، 38-39
بطور ڈیزائن ٹول، 191-192	
انٹرفیس کے طور پر، 191	بلبلے کی قسمیں، 81-82
ترجیحی قطاریں اور، 667-668	صفوں کی تخلیق، 37-38
کب استعمال کریں، 824-826	حذف کرنا، 42
ایڈوانسڈ سورٹنگ ویژولائزیشن ٹول	انکیسولیشن، 42-43 کی مثال، 39-42
295-297 کے ساتھ تقسیم	بہتر مثال، 43-47
309-310، 318 کے ساتھ فوری ترتیب	
289-291 کے ساتھ شیل سورتس	ابتداء، 39
الگورتھم	داخل کرنا، 42
تعریف، 1	اندراج کی قسمیں، 90
متغیرات، 82	تلاش، 42
کا مقصد، 40-47	انتخاب کی قسمیں، 85-86
ترکیب مشابہت، 1-3	ٹراورسل، 42
کی رفتار	اری ویژولائزیشن ٹول، 30-37
بگ اے نوٹیشن، 65-68	حذف کرنا، 37، 34-35
عام مقصد کے ڈیٹا ڈھانچے، 824، 819-820	نقلیں، 35-37
	داخل کرنا، 33
الگورتھم چھانٹنا، 828	تلاش، 31-33
خصوصی ترتیب دینے والے ڈیٹا ڈھانچے، 826	الگورتھم کی رفتار، 37
صفوں کی تقسیم، 39	ٹراورسل، 35
تمام جوڑوں کا مختصر ترین راستہ کا مسئلہ، 796-798	صفوں بیش ٹیبلز بھی دیکھیں۔ بیشتر
تکرار کے دوران ڈیٹا ڈھانچے کو تبدیل کرنا، 216-217	فوائد/نقصانات، 5، 69، 820-821

ارے کلاس تک رسائی کے عناصر، 38-39 تخلیق،	ملحقہ میٹرکس، ازگر میں 710-711، 713-714
37-38	
حذف کرنا، 42	اسٹیک/قطار کے ساتھ کیس کا موازنہ استعمال کریں، 104
انکیسولیشن، 42-43 کی مثال، 39-42	103-
کی بہتر مثال، 43-47	آمد کی ترتیب، وضاحت شدہ، 116-117
	ASCII کوڈز، 386
ابتداء، 39	اسائنمنٹ سٹیٹمنٹس، ازگر میں کثیر قدر کی تفویض،
داخل کرنا، 42	17-18
تلاش، 42	صفات
ٹراورسل، 42	تعریف، 7
اری ویژولائزیشن ٹول، 30-37	ازگر کا نام مینگلنگ، 44
حذف کرنا، 34-35، 37	AVL درخت
نقلیں، 35-37	اے وی ایل ٹری کلاس
داخل کرنا، 33	نوڈس کو حذف کرنا، 479-484 نوڈس
تلاش، 31-33	داخل کرنا، 474-478
الگورتھم کی رفتار، 37	نوڈ کلاس، 472-474
ٹراورسل، 35	AVL Tree ویژولائزیشن ٹول، 470
بائٹری تلاش کے درخت کے طور پر	نوڈس داخل کرنا، 470-472
بیان کیا گیا، 377-378	گردشوں میں کراس اوور ذیلی درخت، 478-479
سطح اور سائز، 378-379	بیان کردہ، 469-471، 463
اندراج، رفتار، 66	484-485 کی رفتار
منسلک فہرستیں بمقابلہ 164	اے وی ایل ٹری کلاس
کے طور پر فہرست، 37	نوڈس کو حذف کرنا، 479-484 نوڈس
ترتیب شدہ اری ویژولائزیشن ٹول، 47-51 بائٹری تلاشیں، 49-51	داخل کرنا، 474-478
ڈپلیکیٹس، 51	نوڈ کلاس، 472-474
	AVL Tree ویژولائزیشن ٹول، 470
اندازہ ایک-نمبر گیم کی مثال، 48-49	نوڈس داخل کرنا، 470-472
OrderedArray کلاس کے فوائد، 57-58 مثال کے	
طور پر، find() 53-57 طریقہ،	بی
OrderedRecordArray 52-53 کلاس، 61-65	متوازن درخت۔ AVL درخت بھی دیکھیں۔ سرخ سیاہ
بیپسورٹ میں دوبارہ استعمال کرنا، 688-691	درخت
بطور ترتیب، 13-15 چھانٹنا۔ چھانٹی دیکھیں	تعریف، 463
	تنزلی، 463-464 پیمائشی توازن، 469-464
	سرخ سیاہ قوانین اور، 495
دو جہتی	کب استعمال کریں، 822-823

233	بیس کیس، وضاحت شدہ،	508	سرخ سیاہ درخت،
266	بہترین صورت، بیان کردہ،	86-87	انتخاب کی قسمیں،
65-68	بگ اے نوٹیشن،	294	شلیسورٹس،
2-3	درخت،	658	الگورتھم چھانٹنا، 828 مقامی ڈیٹا کی تلاش،
431-432	درخت،	656-	
484-485	درخت، AVL	826	خصوصی ترتیب دینے والے ڈیٹا ڈھانچے،
350، 375-377	ہائری تلاش کے درخت،	116	ڈھیر،
67		327	ٹمسورٹس،
82	بلبلے کی قسمیں،	751	ٹاپولوجیکل چھانٹنا،
96-97	چھانٹنے کے طریقوں کا موازنہ،	341-344	ہائری سرچ ٹری ویڈیولائزیشن ٹول،
67-68	مستقل،	374	ڈبل چائلڈ نوڈس کو حذف کرنا،
463-464	گنتی کی قسمیں، 324 ڈیجنریٹس،	367	لیف نوڈس کو حذف کرنا،
622-623	پوائنٹ میچز، عام مقصد کے اعداد و شمار کے	368-369	سنگل چائلڈ نوڈس کو حذف کرنا،
581	ڈھانچے، 824 گرافس، 798 پیشنگ،	351-352	نوڈس، 346-348 داخل کرنے والے نوڈس،
583-587	583-587 بیس-6839 بیس،	363-363	کے ساتھ ٹراورسل
66	6839 بیس غیر ترتیب شدہ صفوں میں،		ہائری تلاش کے درخت، AVL درخت بھی دیکھیں۔ نوڈس سرخ سیاہ درخت
			صفوں کے طور پر
			بیان کیا گیا، 377-378
			سطح اور سائز، 378-379
			ہائری سرچ ٹری ویڈیولائزیشن ٹول، 341-344
91	داخل کرنے کی قسمیں،	374	ڈبل چائلڈ نوڈس کو حذف کرنا،
696-700	K سب سے زیادہ،	367	لیف نوڈس کو حذف کرنا،
66-67	لکیری تلاشیں،	368-369	سنگل چائلڈ نوڈس کو حذف کرنا،
183-184	منسلک فہرستیں،	346-348	نوڈس تلاش کرنا،
264-267، 456	انضمام،	351-352	کے ساتھ ٹراورسل،
198	آرڈر شدہ فہرستیں،	361-363	
301-302	تقسیم کرنے کا الگورتھم،	344	BinarySearchTree کلاس،
132	قطاریں،	369-370	سنگل چائلڈ نوڈس کو حذف کرنا،
quadtrees		348-349	نوڈس تلاش کرنا،
645	عین مطابق میچز،	352-353	والے نوڈس،
644	اندراج،	345-346	نوڈ کلاس،
655	قریب ترین میچز،	382-385	ٹیسٹنگ کوڈ،
125	قطاریں،	356-361	کے ساتھ ٹراورسل
318-		340	تعریف،
ریڈکس کی قسمیں، 322		381-382	میں ڈپلیکیٹ کیز
236	تکرار،		

درجہ بندی کے فائل سسٹم کی مشابہت، 341-340 کم از کم/زیادہ	سیاہ اونچائی
سے زیادہ کلیدی اقدار، 366-365 پرتنگ، 381-379 کے ساتھ علیحدہ	رنگ بدلے میں، 500-499
زنخیر، 377-375 کی 585 رفتار	تعریف، 488
	B-trees میں فی نوڈ بلاکس، 445-444
	نیچے سے اوپر کا اندراج، 487-486
822 کب استعمال کریں،	سوالات کے دائروں کے پابند خانے، 603
	باؤنڈڈ کلاس، 606-605
	مکمل طور پر دوسری حدود کے اندر، 611-610
اندازہ-ایک-نمبر گیم کی مثال، 49-48 لوگارٹھمز، 60-58	کارٹیشین کوآرڈینیٹ، 604-603
	سرکل باؤنڈڈ سب کلاس، 609-607
ترتیب شدہ آری ویژولائزیشن ٹول، 51-49	جغرافیائی نقاط، 605-604
53-57 کی 57-58 مثال کے طور پر، find() 57-53 طریقہ،	باؤنڈڈ بکس کا چوراہا، 610-609
-52 کے آرڈرڈ ارے کلاس فوائد	گڈ سیلز کے ساتھ انٹرسیکشن، 629-628
	تہوں کے اندر، 628-625
	باؤنڈڈ کلاس، 606-605
242-244 میں تکرار	شاخیں، بیان کردہ، 338
67 کی رفتار	چوڑائی کا پہلا راستہ
باؤنڈری درخت، باؤنڈری تلاش کے درختوں کے فوائد/نقصانات، 5 متوازن/	
غیر متوازن بھی دیکھیں	731-727 کی مثال
	گراف کلاس، 733-731
	گراف ویژولائزیشن ٹول، 731
463، تعریف	بی درخت
464-463 پیمائشی توازن، 469-464	بلاکس فی نوڈ، 445-444
	تعریف، 444
337، 339-340 بیان کردہ،	نوڈس داخل کرنا، 449-446 تلاش کرنا،
ڈھیر جیسا کہ، 685-684، 666-667	445-446 کی رفتار، 449-446
بف مین کے درخت	
تخلیق کرنا، 391-389 کے ساتھ ضابطہ	
388-389 کشائی کرنا،	830 کب استعمال کریں،
388، تعریف	بلبلے کی قسمیں، 82-77
391-392 کے ساتھ انکوڈنگ	ارے کلاس میں، 82-81
ریاضی کے تاثرات کی نمائندگی کرنا، 365-363	چھانٹنے کے طریقوں کا موازنہ، 97-96
BinarySearchTree کلاس، 344 سنگل چائلڈ نوڈس کو حذف کرنا، 370-369	بیان کیا گیا، 79-77
فائڈنگ نوڈس، 349-348 داخل کرنے والے نوڈس، 353-352	متغیرات، 82
	سادہ ترتیب دینے والا ویژولائزیشن ٹول، 81-79
	82 کی رفتار
___ نوڈ کلاس، 346-345	بالٹیاں، بیان کردہ، 569
ٹیسٹنگ کوڈ، 385-382	بفرز، وضاحت شدہ، 442
356-361 کے ساتھ ٹراورسل	بانٹ کوڈ، وضاحت شدہ، 8

سی

کارٹیشن کوآرڈینیٹ	323-
سوالات کے دائروں کے پابند خانے، 603-604	بگ O اشارے میں مستقل، 67-68 گنتی کی ترتیب، 324
تعریف، 597-598	کراس اوور ذیلی درخت
599 کے درمیان فاصلہ	تعریف، 478
خلیات، وضاحت شدہ، 38	گردشوں میں، 478-479
کریکٹر کوڈز، 386-388	کٹ آف پوائنٹس، ڈیفائنڈ، ڈائریکٹڈ گرافس میں 315
درخت نوڈس کے بچے	سائیکل، 743-744
تعریف، 338	بیملائونین، 800-802
ڈبل چائلڈ نوڈس، ڈیلیٹ کرنا، 370-375	وارشل کا الگورتھم اور، 755-758
سرخ سیاہ درختوں میں صفر بچے، 495-496	
سنگل چائلڈ نوڈس، ڈیلیٹ کرنا، 367-370	
سرکل ہاؤنڈز سب کلاس، 607-609	
حلقے استفسار کے حلقے دیکھیں	
سرکلر فہرستیں، 209-210	
سرکلر قطاریں، بیان کردہ، 118	
طبقاتی صفات، بیان کردہ، 25	
کلاسز، وضاحت شدہ، 23	
قریبی میچز قریب ترین میچز دیکھیں	
بیش ٹیبلز میں کلسٹرنگ	
بیش ٹیبل اوپن ایڈریسنگ ویژولائزیشن ٹول کے ساتھ، 540-543	
پرائمری اور سیکنڈری کلسٹرنگ، 558-559	
تصادم	
تعریف، 533	
بیشنگ اور، 533-536	
امتزاج، تکرار اور، 278-280	
ازگر میں تبصرے، بیان کیا گیا، 12	
پیچیدہ اعداد، بیان کردہ، 26	
ڈیٹا سکیڑ رہا ہے۔ بگ مین کوڈ دیکھیں	
کمپیوٹیشنل پیچیدگی، وضاحت شدہ، 68	
مربوط ترتیب، 15	
مشروط اظہار، وضاحت، 20	
منسلک گراف، وضاحت شدہ، 708	
ڈائریکٹڈ گرافس میں کنیکٹوٹی، 751	
کنیکٹیویٹی میٹرکس، 753	
عبوری بندش، 751-756	
	ڈیٹا کمپریشن، بگ مین کوڈ ڈیٹا تنظیمیں دیکھیں
	تعریف، 1
	ترکیب مشابہت، 3-1
	ڈیٹا ڈھانچے، مخصوص ڈیٹا کی اقسام بھی دیکھیں
	ڈھانچے
	تکرار کے دوران تبدیل کرنا، 217-216 کا انتخاب
	کرنا کہ کیا استعمال کرنا ہے۔
	بنیادی اعداد و شمار کے ڈھانچے، 818-824
	مسئلہ کا تجزیہ، 814-818 خصوصی ترتیب دینے والے
	ڈیٹا ڈھانچے، 824-826
	خصوصی ڈیٹا ڈھانچے، 828-829
	ڈیٹا بیس بمقابلہ 7
	تعریف، 1
	فہرست، 4-5
	آپریشنز، 4 مقصد، 3-4 ڈیٹا کی اقسام
	ADTs (خلاصہ ڈیٹا کی اقسام)
	بیان کردہ، 184، 190-191
	بطور ڈیزائن ٹول، 191-192
	انٹرفیس کے طور پر، 191
	بیان کیا گیا، 189-190

ڈی

متحرک ٹائپنگ، 13-12 حوالہ جات کی اقسام،
ازگر میں 163-160 ترتیب، 15-13

انحصاری تعلقات کی مثال (ٹیپولوجیکل جھانٹنا)، 739 گہرائی کی پہلی ٹراورسل
مثال، 722-720 گیم سمولیشن، 727

ڈیٹا بیس

ڈیٹا سٹرکچر بمقابلہ 7

تعریف، 6

کلاؤڈ میں تقسیم کردہ ڈیٹا سیٹس، وضاحت شدہ، 438

بف میں درختوں کے ساتھ ضابطہ کشائی، 389-388 نرنلی درخت،

وضاحت شدہ، 464-463

حذف کرنا ہٹانا بھی دیکھیں

صفوں میں

ارے کلاس، 42

اری ویژولائزیشن ٹول، 37، 35-34

تعریف، 4

دوبری منسلک فہرستوں میں

اختتام پر، 204-201

وسط میں، 208-204

ڈپلیکیٹس کے ساتھ، گڑ میں

36، 623

بیش ٹیبلز میں

بیش ٹیبل کلاس کے ساتھ، 553-552

بیش ٹیبل چیننگ ویژولائزیشن ٹول کے ساتھ، 568

HashTableOpenAddressing کے ساتھ

تصور کا آلہ، 542

علیحدہ زنجیر کے لیے، 574

منسلک فہرستوں میں، 177-174، 167-166

نوڈس کے

2-3-4 درختوں میں، 430-423

AVL درختوں میں، 484-479

ڈبل چائلڈ نوڈس، 375-370

ہائری درختوں میں لیف نوڈس، 367 کا عمل، 367

366-

سرخ سیاہ درختوں میں، 508، 491

سنگل چائلڈ نوڈس، پوائنٹ لسٹوں میں 370،

367-366 کوآڈریز میں 647-646، 615-614

گراف کلاس، 727-724

گراف ویژولائزیشن ٹول، 723-722 ہولولیبیا کی مشابہت، 722 ڈیکس

بیان کردہ، 126-125

208 کے لیے دوبری منسلک فہرستیں۔

اولاد، وضاحت شدہ، 339

لغت کی مثال (بیشنگ)، 530-527

ڈکسٹرا کے الگورتھم کا نفاذ، 792-791 ریل سفر کی مثال، 788-782

ویٹڈ گراف ویژولائزیشن ٹول، 791-788 ڈائریکٹڈ گرافس کنیکٹیویٹی ان،

751 کنیکٹیویٹی میٹرکس، 753

عیوری بندش، 756-751

743-744 میں سائیکل

بیان کردہ، 709-708

بیان کیا گیا، 740-739

گراف ویژولائزیشن ٹول میں، 742-741 ٹاپولوجیکل جھانٹنا،

742-743

پوائنٹس کے درمیان فاصلہ

کارٹیشن کوآرڈینیٹس، 599

جغرافیائی نقاط، 601-599 استفسار کے حلقے، 603-601

تقسیم اور فتح الگورتھم

تعریف، 245

ضم کریں بطور، 260-257 ڈبل چائلڈ نوڈس، ڈیلیٹ کرنا، 375-370 ڈبل

بیشنگ، 565-555 کی مثال، 564-562

بیش ٹیبل اوپن ایڈریسنگ ویژولائزیشن ٹول، 562-561

ڈھیروں کے لیے حد بندی مماثل مثال، 116-113

نفاذ، 561-559 کی رفتار، 583

میز کا سائز، 564-565

دوبری فہرستیں، 183-177

دوبری جڑی بوئی فہرستیں، 201-198 ڈیکس کے لیے، 208 سرے پر اندراج/حذف

کرنا، 204-201 بیچ میں اندراج/حذف کرنا، 208-204

بانٹری تلاش کے درختوں میں ڈبلیکیٹ کیڑ، صفوں میں 382-381 نقلیں

اری ویژولائزیشن ٹول، 37-35

ترتیب شدہ اری ویژولائزیشن ٹول، 51

بیش ٹیبلز میں

بیش ٹیبل چیننگ ویژولائزیشن ٹول کے ساتھ، 568

HashTableOpenAddressing کے ساتھ

تصور کا آلہ، 542

متحرک ٹائپنگ، بیان کردہ، 13-12

ای

گرافس میں اضافہ کرنے والے کنارے، 716-713

بیان کردہ، 706، 336

مائلنگ، ملحقہ فہرستوں میں 710 ذخیرہ کرنا، ملحقہ

میٹرکس میں 716-714 وزنی گراف کے لیے 776-774، 712

-710 کارکردگی۔ رفتار کے عناصر (فہرستوں کی) دیکھیں

رسائی، 39-38

تعریف، 38

تکرار کو ختم کرنا، 267

انضمام میں، 275-270 کوٹکس سورٹس

میں، 318

ڈھیروں کے ساتھ، 270-267

encapsulation وضاحت شدہ، 43-42 انکوڈنگ

تعریف، 532

بف مین درختوں کے ساتھ، 392-391

گنتی کے سلسلے، 17-15 ڈھیروں میں غلطی سے نمٹنے،

111-112 غلطیاں۔ مستثنیات دیکھیں

یوکلیدین فاصلہ، بیان کردہ، 599

ایلر، لیون ہارڈ، 709

پوسٹ فکس ایکسپریشنز کا جائزہ لینا، 151-148

عین مطابق مماثلتیں

گرڈز میں، پوائنٹ لسٹوں میں

quadrees، 623-621 میں 644-645

614،

مستثنیات

بیان کیا گیا، 23-22

تکرار ختم کرنا، 215-213

بیرونی اسٹوریج تک رسائی، 442-439

بی درخت

بلاکس فی نوڈ، 445-444

تعریف، 444

نوڈس داخل کرنا، 449-446 تلاش کرنا، 446

-445 کی رفتار، 450-449 منتخب کرنا کہ کیا

استعمال کرنا ہے، 831-829

تعریف، 438

فائل انڈیکس

پیچیدہ تلاش کے معیار، 453-452

بیان کردہ، 451-450

بیشنگ اور، 590-588 داخل کرنا، میموری میں

451، 452-451 متعدد، 452-451 تلاش، 451

ترتیب وار ترتیب، 443-442 انضمام کے ساتھ

ترتیب دینا، 456-453 انتہائی اقدار، تلاش کرنا

695-700	بائری تلاش کے درختوں میں، ڈھیروں میں	818-824	بنیادی ڈیٹا ڈھانچے، کب استعمال کرنا ہے،
365-366			
		19-20	Python میں افعال، بیان کیا گیا،
			امتزاز
		429-430	نزل پر درخواست دینا،
		427	تعریف،
		428-429	توسیع،
			جی
			فرق کی ترتیب
		288-289	بیان کردہ،
		293	منتخب کرنا،
		818-824	عام مقصد کے ڈیٹا ڈھانچے، کب استعمال کرنا ہے،
			جنریٹرز
		421-423	2-3-4 درختوں کے لیے،
		724-727	ملحقہ ورٹیکس ٹراورسل کے لیے،
		218-222	بیان کیا گیا،
			doubleHashProbe()، 559-565
			گراف ٹراورسل
			کے لیے، 731-733
			گرڈ آف سیٹس کے لیے، 629-630
			گرڈ ٹراورسل کے لیے، 623-624
			بیش ٹیبل ٹراورسل کے لیے، 553-554
			ہیپ ٹراورسل کے لیے، 682-683
			linearProbe()، 552
			اپوائنٹ ٹراورسل
			کے لیے، 615
			quadraticProbe()، 554-559
			کوآڈٹری ٹراورسل کے لیے، 645-646
			Timsorts میں، 326
			درختوں کو عبور کرنے کے لیے، 356-361
			جغرافیائی نقاط
			سوالات کے دائروں کے پابند خانے، 604-605
			تعریف، 598
			کے درمیان فاصلہ، 599-601
			بائری تلاش کے درختوں میں، ڈھیروں میں
			365-366،
			237-238
			فیکٹوریلز، بیان کیا گیا،
			fencepost loops وضاحت شدہ، 145
			فیونیکس ترتیب، 218-222
			فیلڈز، وضاحت شدہ، 7
			فائل انڈیکس
			پیچیدہ تلاش کے معیار، 452-453
			بیان کردہ، 450-451
			پیشنگ اور، 588-590 داخل کرنا، 452
			-451 میموری میں، 452، 450 متعدد،
			452 تلاش، 451
			کب استعمال کریں، 829-830
			فائلیں، وضاحت شدہ، 438
			بیش ٹیبلز میں بھرے ہوئے سلسلے، () find() 542-540 طریقہ، بائری تلاش کے
			ساتھ، 53-52 تلاش۔ ڈھیروں میں انتہائی قدریں تلاش کرنا بھی دیکھیں،
			695-700 کم از کم/زیادہ سے زیادہ کلیدی اقدار، 365-366
			نوڈس
			بائری سرچ ٹری ویژولائزیشن ٹول کے ساتھ، 348-346
			BinarySearchTree کلاس کے ساتھ، 348-349
			جانشین، 372-373
			تکرار ختم کرنا
			استثنیٰ بینڈنگ، 215-213 مارکر/سینٹینل اقدار،
			213
			برطرفی کے ٹیسٹ، 213
			فلائڈز، رابرٹ، 798
			Floyd-Warshall الگورتھم، 798 فولڈنگ، ڈیفائنڈ،
			580-581
			فولڈز، ڈیفائنڈ، 531

گراف کلاس، 715-718	کنارے، 710
چوڑائی پہلا ٹراورسل، 731-733	عمودی، 709-710
گہرائی کا پہلا سفر، 724-727	کا مقصد، الگورتھم کی 706 رفتار، 798
735-739 میں کم سے کم پھیلے ہوئے درخت	اصطلاحات، 707-709
ٹاپولوجیکل چھانٹنا، 746-747	
گراف ویژولائزیشن ٹول	ٹراورسل، 718-719
چوڑائی-پہلی ٹراورسل، 731	چوڑائی-پہلی، 727-733
ڈیپتھ فرسٹ ٹراورسل، 722-723	گہرائی پہلے، 719-727
741-742 میں گرافس کو ہدایت کی گئی۔	کب استعمال کریں، 828-829
733 میں کم سے کم پھیلے ہوئے درخت	عظیم دائرہ، وضاحت شدہ، 599-600
ٹاپولوجیکل چھانٹنا الگورتھم، 742	گرڈ کلاس، 619-620
گراف ٹاپولوجیکل چھانٹنا بھی دیکھیں۔ وزنی گراف	گرڈز، 617
عمودی / کناروں کو شامل کرنا، 713-716	حذف کرنے والے پوائنٹس، 623
فائدے/نقصانات، 5	عین مطابق مماثلتیں، 621-623
تعریف، 337	گرڈ کلاس کی مثالیں، 619-620
ہدایت کردہ گراف	Python میں نفاذ، 618-619 داخل کرنے والے پوائنٹس، 621
کنیکٹوٹی، 751-756 میں	1620 استفسار کے حلقوں کے ساتھ، 628-629
743-744 میں سائیکل	قریب ترین میچز، 630-633، 624-625
بیان کیا گیا، 739-740	بمسابہ سیل کی ترتیب، تہوں کے اندر 629-630 استفسار
گراف ویژولائزیشن ٹول میں، 741-742	کے دائرے، 628-625 ٹریورسنگ پوائنٹس، 623-624
ٹاپولوجیکل چھانٹنا، 742-743	
گراف کلاس، 715-718	کب استعمال کریں، 828
709 کی تاریخ	بڑھتی ہوئی بیش میز
ناقابل اعتماد مسائل	بیش ٹیبل کلاس کے ساتھ، 550-551
وضاحت، 798	585-587 کی رفتار
بیملٹن کے راستے / سائیکل، 800-802	اندازہ-ایک-نمبر گیم کی مثال، 48-49
نائٹ ٹور، 798	
ٹریولنگ سیلز پرسن، 799-800	
کم سے کم پھیلے ہوئے درخت	
بیان کیا گیا، 733	
گراف کلاس میں، 735-739	بیملٹونین پاتھ/سائیکل کا انٹری ایبل مسئلہ، 800-802
گراف ویژولائزیشن ٹول میں، 733	بیش ایڈریسز، ڈیفائنڈ، 531
ذیلی گراف کے طور پر، 733-737	بیش کے افعال
ماڈلنگ	حساب کی رفتار، 575
ملحقہ فہرست، 712-713	بیان کردہ، 526، 531
ملحقہ میٹرکس، 710-712	فولڈنگ، 580-581

نان رینڈم کیز، 576-577 رینڈم کیز، simpleHash()
546، 578-580 کے لیے سٹرنگز کے لیے
545-

بیش میزیں

ملحقہ میٹرکس بطور، 712 فوائد/نقصانات، 5، 525

بیان کردہ، 531، 525

بیرونی اسٹوریج کے لیے، 590-588

بیش ٹیبل کلاس، 545-544

ڈیٹا ڈیلیٹ کرنا، 553-552 بڑھتے ہوئے بیش ٹیبلز،

551-550 ڈیٹا داخل کرنا، 549-548 ڈیٹا کو ری بیش

کرنا، 551 ڈیٹا سرچ کرنا، simpleHash() 548-546

فنکشن، 546-545

ٹراورسل، 554-553

بیش ٹیبل چیننگ ویژولائزیشن ٹول، 569-566

بالٹیاں، 569

ڈیٹا کو حذف کرنا، 568 نقلیں،

568

بوجھ کے عوامل، 568

میز کا سائز، 569

بیش ٹیبل اوپن ایڈریسنگ ویژولائزیشن ٹول، 543-536

کلسترنگ، 543-540 ڈیٹا کو حذف کرنا، 542

ڈبل بیشنگ، 562-561 ڈپلیکیٹس، 542 ڈیٹا

داخل کرنا، 540-537 چوکور تحقیقات، 558

555-ڈیٹا تلاش کرنا، 540

ٹراورسل، 554

کب استعمال کریں، 823

hashingf

تصادم اور، 536-533

بیرونی اسٹوریج اور 590-588

کلیدوں کی لغت کی مثال، 530-527 نان رینڈم کیز،
576-578

نمبرز جیسا کہ، 527-526

بے ترتیب چابیاں، 576-575

اوپن ایڈریسنگ ڈبل بیشنگ، 565-559

بیش ٹیبل کلاس، 554-544

لکیری تحقیقات، 543-536 چوکور تحقیقات،

559-554 علیحدہ زنجیر بمقابلہ 588-587 عمل،

533-530 علیحدہ زنجیر

تعریف، 565

بیش ٹیبل کلاس، 574-569

بیش ٹیبل چیننگ ویژولائزیشن ٹول، 569-566

KeyValuelist کلاس، 572-571 اوپن

ایڈریسنگ بمقابلہ 588-587 اقسام استعمال

کرنے کے لیے، 575-574

سادہ بیش () فنکشن، 546-545 کی رفتار، 581 اوپن

ایڈریسنگ، 583-581 الگ چیننگ، 587-583 تار،

580-578

کب استعمال کریں، 830

بیش ٹیبل کلاس

اوپن ایڈریسنگ، 545-544 ڈیٹا ڈیلیٹ کرنا، 553-552

بڑھتے ہوئے بیش ٹیبلز، 551-550 ڈیٹا داخل کرنا،

linearProbe() 548-549 جنریٹر، 552 ری بیشنگ

ڈیٹا، 551 ڈیٹا سرچنگ، 548-546

ٹراورسل، 554-553

علیحدہ زنجیری، 574-569

بیش ٹیبل چیننگ ویژولائزیشن ٹول، 569-566

بالٹیاں، 569

ٹیٹا کو حذف کرنا، 568 نقلیں،	بیپ کلاس کے ساتھ، 682-680
568	زیادہ سے زیادہ، 674 کو تبدیل کرنا
بوجھ کے عوامل، 568	بیپ ویژولائزیشن ٹول کے ساتھ، 677
میز کا سائز، 569	اوپر/نیچے چھاننا، 674-670
بیش ٹیبل اوپن ایڈریسنگ ویژولائزیشن ٹول، 543-536	چھاننا 684-683 کی بیاپسورٹ کی
	رفتار دیکھیں
کلسترنگ، 543-540 ڈیٹا کو حذف کرنا، 542	عبور
ڈبل بیشنگ، 562-561 ڈپلیکیٹس، 542 ڈیٹا	بیپ کلاس کے ساتھ، 683-682
داخل کرنا، 540-537 چوکور تحقیقات، 558	بیپ ویژولائزیشن ٹول کے ساتھ، 677
-555 ڈیٹا تلاش کرنا، 540	heapsort
	heapify() سب روٹین، 693-691
	heapsort() سب روٹین، 693-691
	686 کا عمل
ٹراورسل، 554	688-691 کے لیے سرئی کو دوبارہ استعمال کرنا
hasrsine فارمولا، وضاحت، 600	اوپر/نیچے چھاننا، 693 کی 688-686 رفتار
بیپ کلاس، 683-677	
بیپ ویژولائزیشن ٹول، heapify() 677-674 سبروٹین، 693-691	کب استعمال کریں، 827
بیس کے فوائد/نقصانات، 5 بائری ٹریز کے طور پر، 685-684	heapsort() سب روٹین، 693-691
667-666 ترجیحات کو تبدیل کرنا، 674	ذیلی درختوں کی اونچائی، بیان کردہ، 467
	درجہ بندی فائل سسٹم کی مشابہت، 341-340
	سوراخ، وضاحت شدہ، 35-34
بیان کردہ، 666، 104	بف مین، ڈیوڈ، 386
انتہائی قدریں تلاش کرنا، 700-695	بف مین کوڈ
بیپ کلاس، 683-677	کریکٹر کوڈز، 388-386
بیپ ویژولائزیشن ٹول، 677-674	تعریف، 386
مثانہ اور بے ترتیب طور پر بھرنا، 676	بف مین کے درخت
اندراج، 670-669	تخلیق کرنا، 391-389 کے ساتھ ضابطہ
بیپ کلاس کے ساتھ، بیپ ویژولائزیشن ٹول کے ساتھ	کشائی کرنا، 389-388
675، 680-679	تعریف، 388
آرڈر کے اعدادوشمار کے لیے، 695-694	392-391 کے ساتھ انکوڈنگ
جیسا کہ جزوی طور پر حکم دیا گیا ہے، 668 جھانکنا،	
674 بیپ ویژولائزیشن ٹول کے ساتھ، 677	
ترجیحی قطاریں اور، 668-667 کا مقصد، 665	اڑگے ماڈیولز درآمد کرنا، اڑگے میں 19-18 انڈینٹیشن، بیان کیا
	گیا، 12-9
674-670 میں بتانا	اشارہ جات فائل انڈیکس بھی دیکھیں

بیش ٹیبلز بطور، 588	ڈھیروں میں، 669-670
کب استعمال کریں، 829-830	بیپ کلاس کے ساتھ، 679-680
شامل کرنا، وضاحت شدہ، 237	بیپ ویژولائزیشن ٹول کے ساتھ، 675
infix انویٹیشن	منسلک فہرستوں میں، 170-174
پوسٹ فکس اشارے کے ساتھ موازنہ، 133	نوڈس کے
بیان کردہ، 133، 363	AVL tree کلاس کے ساتھ، 474-478
انفکس کیلکولیٹر ٹول، 142-148	AVL Tree ویژولائزیشن ٹول کے ساتھ، 470-472
پوسٹ فکس میں ترجمہ، 134-142	ہائری سرچ ٹری ویژولائزیشن ٹول کے ساتھ، 351-352
انفکس کیلکولیٹر ٹول، 142-148	Binary Search Tree کلاس کے ساتھ، 352-353
وراثت، وضاحت، 23	بی درختوں میں، 446-449
صفوں کی ابتداء، 39	متعدد سرخ نوڈس، 492 کا عمل
ترتیب میں جانشین	
تعریف، 371	
تلاش کرنا، 372-373 نوڈس کی جگہ 373-374	
	سرخ سیاہ درختوں میں، 492، 498-499
ان آرڈر ٹراورسل، 353-355	پوائنٹ لسٹوں میں، 613-614
اندراج	ترجیحی قطاروں میں، 127-128 کوآڈٹریز میں، 644
2-3 درختوں میں، 437-438	641-638-636
2-3-4 درختوں میں، 404-405	قطاروں میں، 117، 119
Tree 234 ویژولائزیشن ٹول کے ساتھ، 409-410	ترتیب وار ترتیب دی گئی فائلوں میں، 443
صفوں میں	ڈھیروں میں دھکا (ڈھیر) دیکھیں
ارے کلاس، 42	درختوں میں اوپر سے نیچے، 486
اری ویژولائزیشن ٹول، 66 کی رفتار 33	اندراج کی قسمیں، 87-91
	ارے کلاس میں، 90
درختوں میں نیچے سے اوپر، 486-487	چھانٹنے کے طریقوں کا موازنہ، 96-97
تعریف، 4	بیان کیا گیا، 87-89
دوبری منسلک فہرستوں میں	نقصانات، 286
اختتام پر، 201-204	متغیرات، 91
وسط میں، 204-208	فہرست اندراج کی قسمیں، 198
نقل کے ساتھ، 36	سادہ ترتیب دینے والا ویژولائزیشن ٹول، چھوٹے پارٹیشنز کے اندر
فائل انڈیکس میں، 451-452	89-90، 315 کی رفتار، 91
گڈز میں، 621-620	
بیش ٹیبلز میں	
بیش ٹیبل کلاس کے ساتھ، 548-549	کب استعمال کریں، 827
HashTable Open Addressing کے ساتھ	مثال کی خصوصیات، بیان کردہ، 25
ویژولائزیشن ٹول، 537-540	مثالی، بیان کردہ، 23
584 کی رفتار	انٹیجر انڈیکس، وضاحت شدہ، 38

اندرونی نوڈس
 تعریف، 339
 2-3-4 درختوں میں حذف کرنا، 424-425 تقسیم کو
 فروغ دینا، 435-437 ترجمان (Python) بیان کیا گیا،
 8-12
 چورایا
 ہائونڈنگ خانوں کا، گڈ سیلز کا 610، 628-629
 609-وقفہ کی ترتیب
 بیان کردہ، 288-289
 منتخب کرنا، 293 پیچیدہ
 مسائل
 وضاحت، 798
 ہملٹن کے راستے / سائیکل، 800-802
 نائٹ ٹور، 798
 ٹریولنگ سیلز پرسن، 799-800
 invariants
 بلبوں کی قسموں میں، 82
 تعریف، 82
 اندراج کی قسموں میں، 91
 انتخاب کی قسموں میں، 86
 isomorphic وضاحت شدہ، 508
 اشیاء، وضاحت، 38
 تکرار
 216-217 کے دوران ڈیٹا ڈھانچے کو تبدیل کرنا
 بیان کیا گیا، 15-17
 ختم کرنا
 اسٹینن ہینڈلنگ، 213-215 مارکر/سینٹینل
 اقدار، 213
 برطرفی کے ٹیسٹ، 213
 تکرار کرنے والے
 بیان کیا گیا، 211-212
 212-217 میں طریقے
 ازگر میں، 217-222

کے

K سب سے زیادہ
 انتہائی قدریں تلاش کرنا، 695-696
 696-700 کی رفتار
 kd درخت، وضاحت شدہ، 659
 اہم اقدار
 تعریف، 346
 درختوں میں کم از کم/زیادہ سے زیادہ تلاش کرنا، 365-366
 چابیاں
 بیان کردہ، 61، 339، 7
 ہائری تلاش کے درختوں میں نقلیں، 381-382
 ہیشنگ کے لیے
 لغت کی مثال، 527-530
 نان رینڈم کیز، 576-578
 نمبرز جیسا کہ، 526-527
 بے ترتیب چابیاں، 575-576
 ثانوی ترتیب کی چابیاں، 96
 کلیدی قدر کی فہرست کی کلاس، 571-572
 knapsack مسئلہ، 277-278
 نائٹ ٹور کا انٹری ایبل مسئلہ، 798
 کونگسبرگ پل کا مسئلہ، 709
 عرض البلد، وضاحت شدہ، 598
 تہوں، سوالات کے دائرے کے اندر، 625-628
 بٹے
 تعریف، 339
 حذف کرنا
 2-3-4 درختوں میں، 424
 ہائری تلاش کے درختوں میں، 367

ایل

339-340 بائیں بچہ، وضاحت شدہ،	لنک اور لنکڈ لسٹ کلاسز
476 بائیں بھاری، وضاحت شدہ،	حذف کرنا، 174-177
سطحیں (نوڈس کی)	اندراج، 170-174
تعریف، 339	167-169 میں طریقے
درختوں میں صفوں کے طور پر، 378-379	تلاش، 170-174
لائبریریاں، کب استعمال کریں، 820	ٹراورسل، 169-170
لکیری تحقیقات، 536-543	لنکڈ لسٹ ویژولائزیشن ٹول، 164-167
تعریف، 536	لنکس، 158-160
بیش ٹیبل کلاس، 552	فہرستوں کا حکم دیا
بیش ٹیبل اوپن ایڈریسنگ ویژولائزیشن ٹول، 536-543	بیان کیا گیا، 192
کلسٹرنگ، 543-540 ڈیٹا کو حذف	فہرست اندراج کی ترتیب، 198 کے ساتھ
کرنا، 542 نقلیں، 542 ڈیٹا داخل کرنا،	آرڈرڈ لسٹ کلاس، 193-198
540-537 ڈیٹا تلاش کرنا، 540	آرڈرڈ لسٹ ویژولائزیشن کلاس، 192-193
	198 کی رفتار
	187-189 تک قطار کا نفاذ
ٹراورسل، 554	حوالہ کی اقسام اور، 160-163
582-583 کی رفتار	183-184 کی رفتار
لکیری تلاشیں	184-187 کے ذریعے اسٹیک نفاذ
تعریف، 48	جڑے ہوئے درخت، نوڈس، 378-379
کی رفتار، 66-67	لنکڈ لسٹ کلاس، 159
لنک کلاس، 159	حذف کرنا، 174-177
حذف کرنا، 174-177	اندراج، 170-174
اندراج، 170-174	167-169 میں طریقے
167-169 میں طریقے	تلاش، 170-174
تلاش، 170-174	ٹراورسل، 169-170
ٹراورسل، 169-170	لنکڈ لسٹ ویژولائزیشن ٹول، 164-167
منسلک فہرستیں	منسلک فہرستوں میں لنکس، 158-160
ملحقہ فہرست، ماڈلنگ، 712-713 فوائد/نقصانات، 821	فہرست فہم، بیان کردہ، 20-22
5، 336، صفیں بمقابلہ 164	پوائنٹس کی فہرست، 612
	حذف کرنے والے پوائنٹس، 614-615
سرکلر فہرستیں، 209-210	عین مطابق میچز، 614
دوبری فہرستیں، 177-183	پوائنٹس داخل کرنا، 613-614
دوبری جڑی ہوئی فہرستیں، 201-198 ڈیکس کے لیے، 208 سرے	قریب ترین میچز، 615-616
پر اندراج/حذف کرنا، 204-201 بیچ میں اندراج/حذف کرنا، 208	پوائنٹ لسٹ کلاس، 612
204-	عبور پوائنٹس، 615

فہرستیں (ازگر میں ڈیٹا کی قسم). ADT فہرستیں بھی دیکھیں۔ منسلک فہرستیں

صفوں کے طور پر، 37 رسائی والے عناصر، 39
38-تخلیق، 37-38

حذف کرنا، 42

ابتداء، 39

داخل کرنا، 42

تلاش، 42

ٹراورسل، 42

ترتیب کے طور پر، 13-15

سلائسنگ، 39

ڈھیر کے طور پر، 108-112

حد بندی مماثل مثال، 113-116 غلطی سے نمٹنے، 112

111-لفظ الثنی کی مثال، 113-112

بوجھ کے عوامل

تعریف، 548

علیحدہ زنجیر میں، 568

مقامی متغیرات، وضاحت شدہ، 25

بائنری تلاشوں میں لوگارتھمز، ازگر میں 58-60 منطقی

لکیریں، وضاحت شدہ، 10 طول البلد، وضاحت شدہ،

598 لوہنگ۔ تکرار کرنے والے بھی دیکھیں

بیان کیا گیا، 15-17

فہرست فہمیاں، 20-22

ایم

نقشہ سازی، بیان کردہ، 21 مارکر، ختم ہونے والی تکرار، 213

مماثل حد بندیوں کی مثال اسٹیک کے لیے، 113-116

ریاضی کی شمولیت، وضاحت شدہ، 237

زیادہ سے زیادہ، ڈھیروں میں بدلنا، 677، 674، بھولیلیا کی

مشابہت (گہرائی سے پہلے ٹراورسل)، 722 مابینے والے درخت

کے توازن، 464-469

تین تقسیم کا درمیان، 313-315

ضم کرنا

فوائد/نقصانات، 827، 255، بطور تقسیم اور فتح الگورتھم، 260

257-میں تکرار کو ختم کرنا، 275-270

بیرونی فائلوں کے لیے، 453-456

ضم کرنے کا ویژولائزیشن ٹول، ترتیب شدہ صفوں کے ساتھ

255-264، 263-264 کی رفتار، 267-264 سب رینجز کے

ساتھ، 262-260 ٹیسٹنگ کوڈ، 263-262

مرجسورٹ ویژولائزیشن ٹول، 264-263

طریقے، وضاحت، 23

کم سے کم پھیلے ہوئے درخت

بیان کیا گیا، 733

گراف کلاس میں، 739-735 گراف ویژولائزیشن ٹول

میں، 733 بطور ذیلی گراف، 737-733 وزنی گراف بنانے

کے ساتھ، 774-770 الگورتھم بنانا، 780-774 نیٹ

ورکنگ مثال، 768

ویٹڈ گراف کلاس، 779-776

ویٹڈ گراف ویژولائزیشن ٹول، 770-768

ماڈلنگ گراف

ملحقہ فہرست، 713-712 ملحقہ

میترکس، 712-710 کنارے، 710

عمودی، 710-709

ماڈیولز، امپورٹنگ، 18-19 ایک سے زیادہ فائل انڈیکس، 452 ایک

سے زیادہ ریڈ نوڈس داخل کرنے کے دوران، 492 ضرب ترتیب، 15

ملٹی ویلیو اسائمنٹ، بیان کردہ، 18-17 ملٹی وے ٹریز، ڈیفینڈ،

2-3-337 درخت بھی دیکھیں؛ 4-3-2 درخت؛ بی درخت

بایمی تکرار، وضاحت شدہ، 374

ن

Python میں نام ملنا، وضاحت شدہ، 44	BinarySearchTree کلاس کے ساتھ، 352-353
Python میں نام کی جگہیں، وضاحت شدہ، 19	بی درختوں میں، 446-449
قریب ترین میچز	350 کا عمل
گڈز میں، 630-633، 624-625	سرخ سیاہ درختوں میں، 498-499، 491-492
پوائنٹ لسٹوں میں، 616-615	جڑے ہوئے درختوں میں، 379-378
quadtrees میں، 655-647	جانشینوں سے بدلنا، 374-373 سرخ سیاہ درختوں میں
پڑوسی، بیان کردہ، 707	گھومتے ہوئے، 491-490
نیٹ ورکنگ کی مثال (کم سے کم پھیلے ہوئے درخت)، 768	492-493، 500-507
	تقسیم
الگورتھم، 780-774	2-3 درختوں میں، 438-437، 434-433
کم از کم پھیلے ہوئے درخت کی تعمیر، 774-770	2-3-4 درختوں میں، 408-407، 406-405
__نوڈ کلاس	کلر سوپیس اور، 512 اندرونی نوڈس کو فروغ دینا، 437-435
AVLtree کلاس، 474-472	گردشیں اور، 514-512
BinarySearchTree کلاس، 346-345	234 Tree کلاس کے ساتھ، 421-418
درخت 234 کلاس، 415-412	بیشنگ کے لیے نان رینڈم کیز، 578-576 نان ولیٹائل ڈینا
نوڈ کلاس (کوآڈریز)، 641-640	اسٹوریج، ڈیفائنڈ، 439
نوڈس	سرخ سیاہ درختوں میں صفر بجے، 496-495
B-trees میں فی نوڈ بلاکس، 445-444	نمبرز
تعریف، 336	الفاظ کو تبدیل کرنا، 530-527 بطور پیش کیز،
حذف کرنا	527-526 کو اختیارات میں بڑھانا، 276-275
2-3-4 درختوں میں، 430-423	
AVL درختوں میں، 484-479	
ڈبل چائلڈ نوڈس، 375-370	
لیف نوڈس، 367	
367-366 کا عمل	
سرخ سیاہ درختوں میں، 508، 491	
سنگل چائلڈ نوڈس، 370-367	
تلاش کرنا	آجیکٹ پر مبنی پروگرامنگ، بیان کردہ، 23-26 آجیکٹ
ہائری سرچ ٹری ویژولائزیشن ٹول کے ساتھ، 348-346	23، تعریف
BinarySearchTree کلاس کے ساتھ، 349-348	ذخیرہ کرنا، 65-60
ہلٹے ہوئے رنگ، 500-499، 494-493، 489-490	octrees, defined, 659
	اوپن ایڈریسنگ
داخل کرنا	تعریف، 535
AVLtree کلاس کے ساتھ، 478-474	ڈبل بیشنگ، 565-559 کی مثال، 564
AVLTree ویژولائزیشن ٹول کے ساتھ، 472-470	562-

اے

بیش ٹیبل اوپن ایڈریسنگ ویڈیو لائزیشن ٹول، 561-562

نفاذ، 559-561

میز کا سائز، 564-565

بیش ٹیبل کلاس، 544-545

ڈیٹا کو حذف کرنا، 553-552 بڑھتے ہوئے بیش

ٹیبلز، 551-550 ڈیٹا داخل کرنا، linearProbe()

548-549 جنریٹر، 552 ڈیٹا کو ری بیش کرنا،

546-548 ڈیٹا تلاش کرنا،

ٹراورسل، 553-554

لکیری تحقیقات، 536-543

تعریف، 536

بیش ٹیبل اوپن ایڈریسنگ ویڈیو لائزیشن ٹول، 554، 543-536

چوکور تحقیقات، 559-554 الگ چیننگ بمقابلہ

581-583 کی رفتار، 588

آپریٹرز، وضاحت شدہ، 364، 133

آپریٹرز

تعریف، 133

ترجیح، 135 اسٹیک پر بچت، 140-139

فنکشن کی ترتیب، بیان کردہ، 68

آرڈر کے اعداد و شمار، ڈھیر برائے، 695-694 آرڈر شدہ صفوں کے

فوائد/نقصانات، 826، 336-335، 58-57، 5

تعریف، 47

اندازہ-ایک-نمبر گیم کی مثال، 48-49

58-57 کی 57-58 مثال کے طور پر، find() 53-57 طریقہ،

52-53 کے آرڈرڈ اررے کلاس فوائد

OrderedArray ویڈیو لائزیشن ٹول، 51-47 بائنری سرچز، 51

49-ڈپلیکیٹس، 51

OrderedRecordArray کلاس، 61-65

فہرستوں کا حکم دیا

بیان کیا گیا، 192

فہرست اندراج کی ترتیب، 198 کے ساتھ

آرڈرڈ لسٹ کلاس، 193-198

آرڈرڈ لسٹ ویڈیو لائزیشن کلاس، 193-192

198 کی رفتار

آرڈر شدہ آری کلاس

57-58 کے فوائد

مثال کے طور پر، 53-57

تلاش () طریقہ، 52-53

OrderedArray ویڈیو لائزیشن ٹول، 47-51

بائنری تلاشیں، 49-51

نقلیں، 51

اندازہ-ایک-نمبر گیم کی مثال، 48-49

آرڈرڈ لسٹ کلاس، 193-198

آرڈرڈ لسٹ ویڈیو لائزیشن کلاس، 193-192

OrderedRecordArray کلاس، 61-65

طول و عرض کے احکامات، بیان کردہ، 815

پیرامیٹرز، وضاحت شدہ، 20

پیرنٹ نوڈس، وضاحت شدہ، 338

ریاضی کے تاثرات کو پارس کرنا

بیان کیا گیا، 133-132

پوسٹ فکس ایکسپریشنز کا جائزہ لینا، 148-151

انفکس کیلکولیٹر ٹول، 142-148

پوسٹ فکس نوٹیشن، 133-134

انفکس کو پوسٹ فکس میں ترجمہ کرنا، 134-142

ٹراورسل آرڈر اور، 363-365

جزوی چھانٹی

تعریف، 87

انتہائی قدری تلاش کرنا، 695-700 بیبیس اور، 668

تقسیم	جغرافیائی نقاط، 601-599 استفسار کے حلقے، 603
ایڈوانسڈ سورٹنگ ویژولائزیشن ٹول کے ساتھ، 295-297	601-
	گرڈز، 617
الگورتھم برائے، 301-297	حذف کرنے والے پوائنٹس، 623
بیان کردہ، 295-294	عین مطابق مماثلتیں، 623-621
Quicksort الگورتھم میں، 304-302	گرڈ کلاس کی مثالیں، 620-619
تفصیلی وضاحت، 313-310	Python میں نفاذ، 619-618 داخل کرنے والے پوائنٹس، 621
318 مکمل نفاذ، 318-315 ابتدائی نفاذ، 309-306	620-استفسار کے حلقوں کے ساتھ، 629-628
میں تکرار کو ختم کرنا	
	قریب ترین میچز، 633-630، 625-624
تین تقسیم کا درمیانی، 315-313	بمساویہ سیل کی ترتیب، تہوں کے اندر 630-629 استفسار
چھوٹے پارٹیشنز، 315	کے دائرے، 628-625 ٹریورسنگ پوائنٹس، 624-623
رفتار، 320-318	
رفتار، 302-301	فہرستیں، 612
راستے	حذف کرنے والے پوائنٹس، 615-614
بیان کردہ، 708-707، 338	عین مطابق میچز، 614
بیملٹونین، 802-800	پوائنٹس داخل کرنا، 614-613
ڈھیروں میں جھانکنا، 674	قریب ترین میچز، 616-615
	پوائنٹ لسٹ کلاس، 612
بیپ ویژولائزیشن ٹول کے ساتھ، 677	عبور پوائنٹس، 615
ترجیحی قطاروں میں، 128	quadtrees
قطاروں میں، 120	ابہام، 639-638 ڈیلیٹنگ پوائنٹس، 647
ڈھیروں میں	646-
تعریف، 106	بیان کیا گیا، 635-633
اسٹیک ویژولائزیشن ٹول میں، 108	عین مطابق مماثلتیں، 645-644
کامل بیش فنکشنز، ڈیفائنڈ، 576-575	پوائنٹس داخل کرنا، 644-641، 638-636
ترتیب، وضاحت، 239	قریب ترین میچز، 655-647
پیٹرز، ٹم، 325	نوڈ کلاس، 641-640
محور اقدار	کوڈ ٹری کلاس، 636-635
بیان کردہ، 296-295	کوڈ ٹری ویژولائزیشن ٹول، 640-639 ٹراورسنگ پوائنٹس،
مساوی چابیاں، 301-300	646-645 پاپ (اسٹیک)
تین کا درمیانی، 315-313	
منتخب کرنا، 306-304	تعریف، 104
پوائنٹ لسٹ کلاس، 612	اسٹیک ویژولائزیشن ٹول میں، 108
پوائنٹس	انفکس نوٹیشن کے ساتھ پوسٹ فکس اشارے کا موازنہ، 133
کے درمیان فاصلہ	
کارٹیشن کوآرڈینیٹس، 599	بیان کردہ، 364، 133

بیان کیا گیا، 133-134
 تاثرات کی تشخیص، 148-151
 انفکس کیلکولیٹر ٹول، 142-148
 انفکس کا ترجمہ، 134-142 پوسٹ آرڈر ٹراورسل، 355
 پاورز، نمبرز کو بڑھانا، 275-276 پریزیڈنٹس (آپریٹرز کی)،
 ڈیفائنڈ، 135 پریفکس نوٹیشن، ڈیفائنڈ، 364، 134 پری آرڈر
 ٹراورسل، 355 پرائمری ڈیفائنڈ 558 پرائمری سرچ ٹریز،
 381-379 تریج، ڈھیروں میں تبدیل، 674 تریجی قطاریں

تعریف، 106
 بیان کیا گیا، 126-127
 بیہس اور، 667-668
 تریجی قطار کی کلاس، 129-132
 تریجی قطار کے تصور کا آلہ، 127-129
 تلاش اور ٹراورسل، 132
 کی رفتار، 132 ارے کے ساتھ کیس کا موازنہ، 104-103
 کب استعمال کریں، 826
 تریجی قطار کی کلاس، 129-132
 تریجی قطار کے تصور کا آلہ، 127-129 جانچ، وضاحت، 541
 مسئلے کا تجزیہ، 818-814
 ڈیٹا کی مقدار، 815-816
 آپریشنز کی فریکوئنسی، 816-817 سافٹ ویئر کی دیکھ
 بھال کی ذمہ داریاں، 818-817
 ڈیٹا کی اقسام، 815-814
 سیڈوکوڈ، ڈیفائنڈ، 140 پش (اسٹیک)
 تعریف، 104
 اسٹیک ویژولائزیشن ٹول میں، 107
 ارگر
 تبصرے، 12
 متحرک ٹائپنگ، 13-12

مستثنیات، 23-22 فنکشنز/سبروٹائز، 20-19
 بطور ترجمانی زبان، 8-12
 تکرار، 15-17
 فہرست کی تفہیم، 22-20 ماڈیولز، امپورٹنگ، 19-18 ملٹی
 ویلیو اسائنمنٹ، 18-17 بطور آجیکٹ اور اینڈ پروگرامنگ،
 26-23 تسلسل، 15-13 وائٹ اسپیس، 12-9، 8

سوال

چوکور تحقیقات، 559-554 کی رفتار، 583
 کواڈ ٹری کلاس، 636-635
 کواڈ ٹری ویژولائزیشن ٹول، 640-639 کواڈ ٹریز
 فوائد/نقصانات، 828، 5 میں ابہام، 639-638 ڈیلیٹنگ
 پوائنٹس، 647-646
 بیان کیا گیا، 635-633
 عین مطابق مماثلتیں، 645-644
 پوائنٹس داخل کرنا، 644-641، 638-636
 قریب ترین میچز، 655-647
 نوڈ کلاس، 641-640
 کواڈ ٹری کلاس، 636-635
 کواڈ ٹری ویژولائزیشن ٹول، 640-639 ٹراورسنگ پوائنٹس،
 646-645 استفہار کے دائرے ہاؤنڈنگ باکسز، 603
 ہاؤنڈز کلاس، 606-605
 مکمل طور پر دوسری حدود کے اندر، 611-610
 کارٹیشن کوآرڈینیٹ، 604-603
 سرکل ہاؤنڈز سب کلاس، 609-607
 جغرافیائی نقاط، 605-604

باؤنڈنگ خانوں کا انٹرسیکشن، گڈ سیلز کے ساتھ 609-610
 انٹرسیکشن، تہوں کے اندر 629-628 پوائنٹس کے درمیان 628
 -625 فاصلہ، 603-601 قطار کی کلاس، 125-120 قطار
 ویژولائزیشن ٹول، 120-119 قطاریں۔ ترجیحی قطاریں بھی دیکھیں

تعریف، 321
 ڈیزائننگ، 322-321 کو عام کرنا، 323-322 کی رفتار، 322 ریل
 سفر کی مثال (سب سے مختصر راستہ کا مسئلہ)، 782-780

تمام جوڑوں کا مختصر ترین راستہ کا مسئلہ، 796-798
 Dijkstra کا الگورتھم، 788-782 الگورتھم کا نفاذ، 792-791

فوائد/نقصانات، 825، 5

سرکلر، 118

تعریف، 106

deques، 125-126

بیان کیا گیا، 117-116

189-187 کے منسلک فہرست پر عمل درآمد

قطار کی کلاس، 125-120

قطار کے تصور کا آلہ، 120-119

تلاش اور ٹراورسل، 132

شفٹنگ، 118-117 کی رفتار، 125 استعمال کیس کی صفوں کے

ساتھ موازنہ، 104-103

ویڈیو گراف کلاس، 796-792

ویڈیو گراف ویژولائزیشن ٹول، 791-788

طاقاتوں تک نمبر بڑھانا، 276-275 بے ترتیب ڈھیر،

بیشنگ کے لیے 676 بے ترتیب کلیدیوں، 576-575

ریکارڈز

بیان کردہ، 61-60، 6

OrderedRecordArray کلاس، 65-61

تکرار

anagrams کے لیے 242-239

درخواستیں۔

مجموعے، 280-278

knap sack مسئلہ، 278-277 طاقتوں کی تعداد میں

اضافہ، 276-275

ہائری تلاشیں، 244-242

236-235 کی خصوصیات

بیان کردہ، 229، 6

تقسیم اور فتح الگورتھم، 245 ختم کرنا، 267

انضمام میں، 275-270

ڈھیروں کے ساتھ، 270-267

فیکٹوریلز، 238-237

235-232 کے ساتھ n ویں اصطلاحات تلاش کرنا

ریاضی کی شمولیت اور، 237

ضم کرنا

فوائد/نقصانات، 255 بطور تقسیم اور فتح الگورتھم، 260-257

میں تکرار کو ختم کرنا، 275-270

فوری ترتیب

ایڈوانسڈ سورٹنگ ویژولائزیشن ٹول، 318، 310-309

304-302 کے لیے الگورتھم

تعریف، 302

تفصیلی وضاحت، 313-310 میں تکرار کو ختم کرنا، 318

مکمل نفاذ، 318-315 ابتدائی نفاذ، 309-306 میڈین آف

تھری پارٹیشن، 315-313 پیوٹ ویلیوز، 306-304 چھوٹی

پارٹیشنز، 318-315 کی رفتار، 318-315

آر

ریڈکس، وضاحت شدہ، 321

ریڈکس ترتیب، 323-320

مرجسورٹ ویژولائزیشن ٹول، 264-263

255-257 ترتیب شدہ صفوں کے ساتھ،	488 خلاف ورزیوں کو درست کرنا،
264-267 کی رفتار	495-496 null بچے،
260-262 سب رینجز کے ساتھ،	496-497 گردشیں اور،
262-263 ٹیسٹنگ کوڈ،	488-489 ریڈ بلیک ٹری ویڈیولائزیشن ٹول،
297-301 تقسیم کرنے کے الگورتھم میں،	نوڈس کو حذف کرنا، 508، 491 مٹانا اور بے ترتیب طور پر بھرنا،
302-304 Quicksort الگورتھم میں،	491 فلپنگ نوڈ کے رنگ، 499-500، 493-494، 489-490
310-313 تفصیلی وضاحت،	
309-318 مکمل نفاذ، 315-318 ابتدائی نفاذ،	نوڈس داخل کرنا، 498-499، 492-491
306-309 میں تکرار کو ختم کرنا	ایک سے زیادہ ریڈ نوڈس داخل کرنے کا تجربہ، 492
313-315 تین تقسیم کا درمیان،	490-491، 492-493، 500-507 گھومنے والے نوڈس،
315 چھوٹے پارٹیشنز،	
318-320 رفتار،	491 غیر متوازن درخت کا تجربہ، 494-496 کی
236 کی رفتار	508 اوپر سے نیچے داخل کرنا، 486
بنوئی کا ٹاور	
245-246 بیان کیا گیا،	488-489 ریڈ بلیک ٹری ویڈیولائزیشن ٹول،
250-255 حل کا نفاذ،	نوڈس کو حذف کرنا، 508، 491 مٹانا اور بے ترتیب طور پر بھرنا،
247-249 TowerOfHanoi ویڈیولائزیشن ٹول،	491 فلپنگ نوڈ کے رنگ، 499-500، 493-494، 489-490
246-247 تکراری گہرائی، بیان کردہ، 255	
487 سرخ سیاہ درست، وضاحت شدہ،	
سرخ سیاہ قوانین	نوڈس داخل کرنا، 498-499، 492-491
495 متوازن درخت اور	اندراج کے تجربے کے دوران متعدد سرخ نوڈس، 492
487-488 بیان کیا گیا،	
488 خلاف ورزیوں کو درست کرنا،	491 گھومنے والے نوڈس، 500-507، 492-493، 490-491 تلاش،
495-496 null بچے،	163 غیر متوازن درختوں کا تجربہ، 496-494 حوالہ جات کی اقسام،
496-497 گردشیں اور،	160-
485 سرخ سیاہ درخت،	
508-510 2-3-4 درخت اور،	551 بیش ٹیبل کلاس کے ساتھ بیش ٹیبلز کو ری بیش کرنا،
510-514 آپریشنل مساوات،	سے ہٹانا، حذف کرنا بھی دیکھیں
509-510 درمیان تبدیلی	بیس میں، بیپ کلاس کے ساتھ، 674-670
فائدے/نقصانات، 5	ترجیحی قطاروں میں، 682-680 قطاروں میں
486-487 نیچے سے اوپر کا اندراج،	128، 117، 119-120
487 کی خصوصیات	
513-515 نفاذ،	ڈھیروں میں پاپ (ڈھیر) دیکھیں
سرخ سیاہ قوانین	ڈھیروں کے لیے معکوس الفاظ کی مثال، 113-112 رائٹ چائلڈ، ڈیفائنڈ،
495 متوازن درخت اور	339-340 رائٹ بیوی، ڈیفائنڈ، 476
487-488 بیان کیا گیا،	

رنگ بفرز، وضاحت شدہ، 118 جڑ (درختوں کی)	سرخ سیاہ درخت، 491 سلسلے، 15
بیان کردہ، 337، 338	ترتیب وار ترتیب دی گئی فائلیں، 443-442
تقسیم، 406-407	مقامی ڈیٹا، 611-612
درخت نوڈس کی گردش	گرڈز، 617-633
نزول پر درخواست دینا، 429-430	پوائنٹس کی فہرست، 612-616
کراس اوور سب ٹریس، 479-478 میں	کارکردگی کی اصلاح، 658-656
بیان کردہ، 426-427	quadtrees، 633-655
نوڈ تقسیم اور، 514-512	اسٹیک اور قطاریں، 132 سیکنڈری کلسٹرنگ، ڈیفائنڈ، 558 سیکنڈری سورٹ کیز، ڈیفائنڈ، 96
سرخ سیاہ قوانین اور، 497-496	
سرخ سیاہ درختوں میں، 493-492، 491-490 500-507	
	انتخاب کی قسمیں، 87-83
	ارے کلاس میں، 86-85
	چھانٹنے کے طریقوں کا موازنہ، 97-96
ایس	بیان کیا گیا، 85-83
	متغیرات، 86
ڈھیروں پر آپریٹرز کو بچانا، Tree234 ویژولائزیشن ٹول میں 140-139	سادہ ترتیب دینے والا ویژولائزیشن ٹول، 85 کی رفتار، 87
سکروولنگ، 411-410	86-
تلاش کریں	سینٹیپیل اقدار
ہائٹری تلاش ہائٹری تلاشیں دیکھیں	تکرار ختم کرنا، 213
تعریف، 4	تین تقسیم کا میڈین، 314
لکیری تلاشیں، 67-66، 48	علیحدہ زنجیر
تلاش کی چابیاں، چابیاں تلاش کرنے ہوئے	بالٹیوں کے ساتھ، 569
دیکھیں۔ تلاش بھی دیکھیں	بیان کردہ، 565، 535
3-4-2 درخت، 404	بیش ٹیبل کلاس، 574-569
Tree234 کلاس کے ساتھ، 417-415	بیش ٹیبل چیننگ ویژولائزیشن ٹول، 569-566 ڈیٹا ڈیلیٹ کرنا، 568
Tree234 ویژولائزیشن ٹول کے ساتھ، 409	ڈپلیکیٹس، 568
صفوں	
ارے کلاس، 42	بوجھ کے عوامل، 568
اری ویژولائزیشن ٹول، 33-31	میز کا سائز، 569
بی درخت، 446-445	کلیدی قدر کی فہرست کی کلاس، 572-571
نقل کے ساتھ، 36-35	اوپن ایڈریسنگ بمقابلہ 588-587
فائل اشاریہ جات، 453-452، 451	583-587 کی رفتار
بیش میزیں	استعمال کرنے کی اقسام، 575-574
بیش ٹیبل کلاس کے ساتھ، 548-546	تسلسل
HashTableOpenAddressing کے ساتھ	بیان کیا گیا، 15-13
ویژولائزیشن ٹول، 540	
584 کی رفتار	شمار کرنا، 17-15 کثیر الوقوع تفویض، 18-17
منسلک فہرستیں، 174-170، 166	

ترتیب وار ترتیب، 442-443 ترتیب وار ذخیرہ،	چھانٹنا
829	گنتی کی ترتیب کے ساتھ، 323-324
شیل، ڈونلڈ L.285	تعریف، 6
شیلسورٹ	ہیپسورٹ کے ساتھ
ایڈوانسڈ سورٹنگ ویژولائزیشن ٹول، 291-289 فوائد/نقصانات، 286	heapify() سب روٹین، 693-691
285-	heapsort() سب روٹین، 693-691
بیان کیا گیا، 286-288	686 کا عمل
اندراج کی ترتیب کے نقصانات، 286 وقفہ کی ترتیب،	688-691 کے لیے سرئی کو دوبارہ استعمال کرنا
288-289، 293	اوپر/نیچے چھانٹنا، 693 کی 686-688
شیل سورٹ کلاس، 291-293	رفتار
294 کی رفتار	انضمام کے ساتھ
کب استعمال کریں، 827	فوائد/نقصانات، 255
شیل سورٹ کلاس، 291-293	بطور تقسیم اور فتح الگورتھم، 257-260
قطاریں بدلنا، 117-118 مختصر ترین	270-275 میں تکرار کو ختم کرنا
راستہ کا مسئلہ	بیرونی فائلوں کے لیے، 453-456
تمام جوڑوں کا مختصر ترین راستہ کا مسئلہ، 796-798	مرجسورٹ ویژولائزیشن ٹول، 263-264
Dijkstra کا الگورتھم، 788-782 نفاذ، 791-792 ریل سفر کی	ترتیب شدہ صفوں کے ساتھ، 257-255 کی
مثال، 780-782	رفتار، 267-264
ویڈ گراف کلاس، 796-792	سب رینجز کے ساتھ، 262-260
ویڈ گراف ویژولائزیشن ٹول، 791-788	ٹیسٹنگ کوڈ، 263-262
بہن بھائی، وضاحت شدہ، 339 ڈھیروں میں اوپر/نیچے چھانٹنا، 674	تقسیم AdvancedSorting Visualization Tool
670-ہیپسورٹ میں، 688-686	Quicksort کے ساتھ پارٹیشننگ دیکھیں، 318، 310-309
سادہ چھانٹی بصری آلہ	302-304 کے لیے الگورتھم
بلبلے کی قسمیں، 81-79	تعریف، 302
اندراج کی قسمیں، 90-89	تفصیلی وضاحت، 313-310
انتخاب کی قسمیں، 85	318 میں تکرار کو ختم کرنا
simpleHash() فنکشن، 546-545 سنگل چائلڈ نوڈس،	مکمل نفاذ، 318-315
ڈیلیٹ کرنا، ازگر میں 370-367 سلائسنگ لسٹ، 39	ابتدائی نفاذ، 309-306
تسلسل، 14	تین کا درمیانی تقسیم، 315-313 پیوٹ ویلیوز، 306-304
سافٹ ویئر بلوٹ، ڈیفائنڈ، 819	چھوٹے پارٹیشنز، 315
چابیاں ترتیب دیں کلیدیں دیکھیں	رفتار، 320-318
SortArray کلاس، 96-91	ریڈکس ترتیب کے ساتھ، 323-320
	شیلسورٹ کے ساتھ

ایڈوانسڈ سورٹنگ ویڈیو لائزیشن ٹول، 289-291	603 کے خانوں کو ہائونڈنگ کرنے والے
فوائد/نقصانات، 285-286	حلقوں کے سوالات
بیان کیا گیا، 286-288	ہائونڈز کلاس، 605-606
اندراج کی ترتیب کے نقصانات، 286 وقفہ کی ترتیب، 293	مکمل طور پر دوسری حدود کے اندر، 610-611
288-289،	سرکل ہائونڈز سب کلاس، 607-609
شیل سورٹ کلاس، 291-293	پوائنٹس کے درمیان فاصلہ، 601-603
کی رفتار، 294 سادہ ترتیب دینے والے	ہائونڈنگ خانوں کا چوراہا، تہوں کے اندر 609-610، 625-628، تلاش،
الگورتھم	612-611، 633-617
بلبلے کی قسمیں، 77-82	
96-97 کا موازنہ	
اندراج کی قسمیں، 87-91	پوائنٹس کی فہرست، 612-616
فہرست اندراج کی قسمیں، 198	کارکردگی کی اصلاح، 656-658
انتخاب کی قسمیں، 83-87	quadrees، 633-655
SortArray کلاس، 96-91 کا	خصوصی ترتیب دینے والے ڈیٹا ڈھانچے، کب استعمال کریں، 826، 828-829
استحکام	824-
Timsort کے ساتھ، 324-327	رفتار
ٹاپولوجیکل چھاننی	3-2 درخت، 438
کے لیے الگورتھم، 742 انحصاری رشتوں کی مثال، 739 ہدایت شدہ	3-4 درخت، 431
گراف میں، 743-742 گراف کلاس میں، 747-746 میں بہتری،	الگورتھم
751-747 کی رفتار، 751	بگ اے نوٹیشن، 65-68
	عام مقصد کے ڈیٹا ڈھانچے، 824، 820-819
	الگورتھم چھاننا، 828
کب استعمال کریں، 826-828	خصوصی ترتیب دینے والے ڈیٹا ڈھانچے، 826
مقامی ڈیٹا	AVL درخت، 485-484
کارٹیشن کوآرڈینیٹ	ہائری تلاش کے درخت، 377-375، 350
سوالات کے دائروں کے پابند خانے، 603-604	بی درخت، 450-449
تعریف، 597-598	بلبلے کی قسمیں، 82
پوائنٹس کے درمیان فاصلہ، 599	گنتی کی ترتیب، 324
تعریف، 597	گراف الگورتھم، 798
جغرافیائی نقاط	بیش فنکشن کمپوٹیشن، 575
سوالات کے دائروں کے پابند خانے، 604-605	بیشنگ، 581
تعریف، 598	اوپن ایڈریسنگ، 583-581 علیحدہ زنجیر، 587
پوائنٹس کے درمیان فاصلہ، 601-599	583-بیپس، 684-683 بیپسورٹ، 693
658-659 آپریشنز، 658 کے لیے اعلیٰ طول و عرض	

Kسب سے زیادہ، 696-700	پاپ بٹن، 108
منسلک فہرستیں، 183-184	پش بٹن، 107
ضم کریں، 264-267، 456	اسٹیک کا سائز، 108
آرڈر شدہ فہرستیں، 198	ڈھیر
تقسیم کاری، 301-302ترجیحی	فوائد/نقصانات، 825، 5
قطاریں، 132کواڈٹریز	بیان کردہ، 104-105
عین مطابق میچز، 645	267-270لنکڈ لسٹ پر عمل درآمد، 184-187پوسٹل
اندراج، 644	اینالوجی، 105-106سیونگ آپریٹرز کے ساتھ تکرار کو ختم
قریب ترین میچز، 655	کرنا، 139-140
قطاریں، 125فوری ترتیب، 320	تلاش اور ٹراورسل، 132
318-	116کی رفتار
ریڈکس ترتیب، 322	اسٹیک کلاس، 108-112
تکرار، 236	حد بندی مماثل مثال، 113-116غلطی سے نمٹنے، 112
سرخ سیاہ درخت، 508	111-لفظ لٹنے کی مثال، 112-113
انتخاب کی قسمیں، 86-87	اسٹیک ویژولائزیشن ٹول، 106-108
شیلسورٹس، 294	نیا بٹن، 107-108
مقامی ڈیٹا کی تلاش، 656-658	جھانکنے والا بٹن، 108
ڈھیر، 116	پاپ بٹن، 108
ٹمسورٹ، 327	پش بٹن، 107
ٹاپولوجیکل چھانٹنا، 751تقسیم کرنا	اسٹیک کا سائز، 108
نوڈس	ارے کے ساتھ کیس کا موازنہ استعمال کریں، 103-104اسٹوریج کی
2-3درختوں میں، 437-438، 433-434	ضروریات۔ بیرونی اسٹوریج بھی دیکھیں
3-4درختوں میں، 407-408، 405-406	2-3درخت، 438
گردشیں اور، 512-514	کلر سویپس اور AVLدرخت، 433-434، 435-437
234Treeکلاس کے ساتھ، 418-421	484-485
جڑ (درختوں کی)، 406-407	ہائٹری تلاش کے درخت، 350، 375-377
الگورتھم چھانٹنے کا استحکام، 96	بی درخت، 449-450
اسٹیک کلاس، 108-112	بلبلے کی قسمیں، 82
حد بندی مماثل مثال، 113-116غلطی سے نمٹنے، 112	گنتی کی ترتیب، 324گراف
111-لفظ لٹنے کی مثال، 112-113	الگورتھم، 798
اسٹیک ویژولائزیشن ٹول، 106-108	بیش ٹیلز، 581
نیا بٹن، 107-108	اوپن ایڈریسنگ، 581-583علیحدہ زنجیر،
جھانکنے والا بٹن، 108	583-587بیپس، 683-684بیپسورٹ،
	693
	داخل کرنے کی قسمیں، 91

696-700، K سب سے زیادہ،	جانشین
منسلک فہرستیں، 183-184	تعریف، 371
ضم کریں، 264-267، 456	تلاش کرنا، 373-372 نوڈس کو تبدیل کرنا، 374-373 نوڈ کے
آرڈر شدہ فہرستیں، 198	رنگ تبدیل کرنا، 499-500، 493-494، 489-490
تقسیم کاری، 301-302 ترحیحی	
قطاریں، 132 کوآڈٹریز	نوڈ سپلٹس اور، 512 سمیل ٹیبلز،
عین مطابق میچز، 645	ڈیفائنڈ، 527
اندراج، 644	
قریب ترین میچز، 655	ٹی
قطاریں، 125 فوری ترتیب، 320	ٹرمینیشن ٹیسٹ، ختم ہونے والی تکرار، 213 ٹیسٹنگ ارے، 47
318-	40-
ریڈکس ترتیب، 322	
تکرار، 236	
سرخ سیاہ درخت، 508	BinarySearchTree کلاس، 382-385
انتخاب کی قسمیں، 86-87	دوبری فہرستیں، 183-180
شیلسورٹس، 294	دوبری منسلک فہرستیں، 208-207 ضم، 263
مقامی ڈیٹا کی تلاش، 658-656	262-ترتیب شدہ صفیں، 65-61، 57-56
ڈھیر، 116	
ٹمسورٹ، 327	ترتیب شدہ فہرستیں، 198-196
ٹاپولوجیکل چھانٹنا، ملحقہ فہرستوں میں 751 ذخیرہ کرنے والے کنارے،	ترحیحی قطاریں، 132-131 قطاریں، 189
ملحقہ میٹرکس میں، 716-714 وزنی گراف کے لیے 776-774، 712-710	188، 123-125
آجیکٹ، آرڈرڈ ریکارڈ آرے کلاس، 65-61 سٹرنگ بیشنگ، 578-578 ذیلی	شیلسورٹس، 293-292
گراف، 578-578 ذیلی گراف جیسا کہ، 737-733 ذیلی بیس، 688-686	سادہ ترتیب دینے والے الگورتھم، 96-94
ذیلی رینجز، انضمام، 262-260	اسٹیکس، 187-186، 112-110
	ٹمسورٹ، 827، 327-324
	ٹوکن، وضاحت شدہ، 145
	اوپر سے نیچے اندراج، 486 ٹاپولوجیکل
	چھانٹنا
	کے لیے الگورتھم، 742 انحصاری تعلقات کی مثال، 739
	ڈائریکٹڈ گرافس، 743-742 گراف کلاس میں، 747-746 میں
	بہتری، 751-747 کی رفتار، 751
ذیلی روٹینز، بیان کردہ، 20-19	
ذیلی درخت	
تعریف، 339	بنوئی کا ٹاور
اونچائی، 467	بیان کیا گیا، 246-245
گردش، 497-496	حل کا نفاذ، 255-250

247-249 کا حل	339, 353 بیان کردہ،
TowerOfHanoi ویڈیولائزیشن ٹول، 246-247	آرڈر، 363-365
TowerOfHanoi ویڈیولائزیشن ٹول، 246-247	ان آرڈر ٹراورسل، 353-355
عبوری بندش، 751-756	پوسٹ آرڈر ٹراورسل، 355 پری آرڈر
انفکس کو پوسٹ فکس نوٹیشن میں ترجمہ کرنا، 134-142	ٹراورسل، 355
ٹریولنگ سیلز پرسن کا ناقابل قابل مسئلہ، 799-800	درخت 234 کلاس، 415
عبور	نوڈس کو حذف کرنا، 423-430
صفوں میں	نوڈ کلاس، 412-415
ارے کلاس، 42	نوڈ سیٹس، 418-421
اری ویڈیولائزیشن ٹول، 35	تلاش کرنا، 415-417
تعریف، 4	گزرنا، 421-423
نقل کے ساتھ، 36-37	Tree234 ویڈیولائزیشن ٹول، 408-411
گراف کا، 718-719	درختوں پر مبنی ڈھیر، 684-685 درختوں کے فوائد/
چوڑائی پہلی، 727-733	نقصانات کی اقسام بھی دیکھیں، 335-336
گہرائی پہلی، 719-727	متوازن/غیر متوازن
گرڈز میں، 623-624	تعریف، 463
بیش ٹیبلز میں	تنزلی، 463-464 پیمائشی توازن، 464-469
بیش ٹیبل کلاس کے ساتھ، 553-554	سائیکل اور، 743-744
HashTableOpenAddressing کے ساتھ	بیان کردہ، 336-337
تصور کا آلہ، 554	اصطلاحات، 337-340
آرڈر، 573-574	کا عبور
ڈھیروں میں	3-4-2 درخت، 421-423
بیپ کلاس کے ساتھ، 682-683	بانئری سرچ ٹری ویڈیولائزیشن ٹول کے ساتھ، 361-363
بیپ ویڈیولائزیشن ٹول کے ساتھ، 677	BinarySearchTree کلاس کے ساتھ، 356-361
تکرار کرنے والے	بیان کردہ، 339, 353
بیان کیا گیا، 211-212	آرڈر، 363-365
212-217 میں طریقے	ان آرڈر ٹراورسل، 353-355
ازگر میں، 217-222	پوسٹ آرڈر ٹراورسل، 355 پری آرڈر
منسلک فہرستوں میں، 169-170	ٹراورسل، 355
پوائنٹ لسٹوں میں، 615	مثلث نمبر
quadtrees میں، 645-646	بیان کردہ، 230-231
ڈھیروں اور قطاروں میں، 132	میں تکرار کو ختم کرنا، 268-270 لوپس کے ساتھ
درختوں کی	نووی اصطلاح کی تلاش، 231-232
3-4-2 درخت، 421-423	تکرار کے ساتھ، 232-235
بانئری سرچ ٹری ویڈیولائزیشن ٹول کے ساتھ، 361-363	BinarySearchTree کلاس کے ساتھ، 356-361

ٹپلز، وضاحت شدہ، 715، 17، دو جہتی صفیں	الگورتھم کی رفتار، 37 ٹراورسل، 35
ملحقہ میٹرکس بطور، 710-711 ارگر میں، 714-713	اے وی ایل درخت، 470 نوڈس داخل کرنا، 472-470 بائری تلاش کے درخت، 344-341
	ڈبل چائلڈ نوڈس کو حذف کرنا، 374 لیف نوڈس کو حذف کرنا، 367-368 سنگل چائلڈ نوڈس کو حذف کرنا، 369-368 فائنڈنگ نوڈس، 348-346 داخل کرنے والے نوڈس، 352-351

یو

غیر متوازن درخت بیان کردہ، 344-343 تنزلی، 464-463 494-494 کی مثال کی تشکیل، 463 میزان کی پیمائش، 469-464 غیر وزنی گراف، گراف دیکھیے ڈیٹا ڈھانچے کے لیے کیسز کا استعمال کریں، 104-103	363-361 کے ساتھ ٹراورسل گراف چوڑائی-پہلی ٹراورسل، 731 ڈیپتھ فرسٹ ٹراورسل، 723-722 ڈائریکٹڈ گرافس، 742-741 میں کم سے کم پھیلے ہوئے درخت، 733 ٹاپولوجیکل چھاننی الگورتھم، 742
---	--

وی

چوٹی گراف میں اضافہ کرنا، 716-713 وضاحت، 706 ماڈلنگ، 710-709 ورچوئل میموری، 831-830 دورہ کرنا، وضاحت کی گئی، 353، 339 تصور کے اوزار 3-4-2 درخت، 411-408 اعلیٰ درجے کی ترتیب کی تقسیم، 297-295 Quicksorts، 310-309، 318 شیلسورٹس، 291-289 صفیں، 37-30 حذف کرنا، 37، 35-34 نقلیں، 37-35 داخل کرنا، 33 تلاش، 33-31	بیش میزیں ڈبل بیشنگ، 562-561 اوپن ایڈریسنگ، 554، 543 536-535 کوآڈریٹک پروینگ، 558-555 الگ چیننگ، 569 566- بیس، 677-674 منسلک فہرستیں، 167-164 انضمام، 264-263 ترتیب شدہ صفیں، 51-47 بائری تلاشیوں، 51-49 نقلیں، 51 اندازہ-ایک-نمبر گیم کی مثال، 49-48 ترتیب شدہ فہرستیں، 193-192 ترجیحی قطاریں، 129-127 کوآڈریٹک، 640-639 قطاریں، 120-119 سرخ سیاہ درخت، 489-488 نوڈس کو حذف کرنا، 508، 491 مٹانا اور بے ترتیب طور پر بھرنا، 491-490 فلہنگ نوڈ کے رنگ، 500-499، 494-493، 490-489
---	---

نوڈس داخل کرنا، 498-499، 492-491
ایک سے زیادہ ریڈ نوڈس داخل کرنے کا تجربہ،
492
گھومنے والے نوڈس، 500-507، 493-492، 490-491
تلاش کرنا، 491 غیر متوازن درخت کا تجربہ،
494-496
سادہ چھانٹی
بلبلے کی قسمیں، 81-79
اندراج کی قسمیں، 90-89
انتخاب کی قسمیں، 85
ڈھیر، 108-106
نیا بٹن، 108-107
جھانکنے والا بٹن، 108
پاپ بٹن، 108
پش بٹن، 107
اسٹیک کا سائز، 108
ٹاور آف بنوئی پھیلی، 247-246 وزنی گرافس کم از کم پھیلے
ہوئے درخت، 770-768 مختصر ترین راستے کا مسئلہ، 791
788-

ڈبلو

وارشل، اسٹیفن، 798، 752

وارشل کا الگورتھم

نفاذ، 756

عبوری بندش اور، 758-751

وزنی گراف

تمام جوڑوں کا مختصر ترین راستہ کا مسئلہ، 798-796 کی

وضاحت، 709-708

کے ساتھ کم سے کم پھیلے ہوئے درخت

عمارت، 774-770

الگورتھم بنانا، 780-774

نیٹ ورکنگ کی مثال، 768

ویٹڈ گراف کلاس، 779-776

ویٹڈ گراف ویژولائزیشن ٹول، 770-768

مختصر ترین راستہ کا مسئلہ

ڈکسٹرا کا الگورتھم، 788-782

نفاذ، 792-791

ریل سفر کی مثال، 782-780

ویٹڈ گراف کلاس، 796-792

ویٹڈ گراف ویژولائزیشن ٹول، 791-788

ویٹڈ گراف کلاس کم از کم پھیلے ہوئے درخت، 779-776

مختصر ترین راستہ کا مسئلہ، 796-792

ویٹڈ گراف ویژولائزیشن ٹول

کم سے کم پھیلے ہوئے درخت، 770-768 مختصر

ترین راستے کا مسئلہ، 791-788

Python میں وائٹ اسپیس، بیان کیا گیا، 12-9، لفظ بادلوں کی

مثال (ڈھیر)، 695-694 لفظ التے کی مثال اسٹیک کے لیے، 113

112-

بدترین صورت، بیان کردہ، 266

Z

2-3-4 درختوں میں زومنگ، 411-410