

**Script** generated by TTT

Title: Lammich: FDS (22.06.2018)

Date: Fri Jun 22 08:37:24 CEST 2018

Duration: 99:55 min

Pages: 80

15 Priority Queues

16 Leftist Heap

17 Priority Queue via Braun Tree

18 Binomial Heap

19 Skew Binomial Heap

173

## Chapter 9

### Priority Queues

171

#### Priority queue informally

Collection of elements with priorities

Operations:

- empty

174

## Priority queues are multisets

The same element can be contained **multiple times** in a priority queue

175

## Priority queues are multisets

The same element can be contained **multiple times** in a priority queue

⇒

The abstract view of a priority queue is a **multiset**

175

## Interface of implementation

The type of elements (= priorities)  $'a$  is a linear order

176

## Interface of implementation

The type of elements (= priorities)  $'a$  is a linear order

An implementation of a priority queue of elements of type  $'a$  must provide

176

## Interface of implementation

The type of elements (= priorities)  $'a$  is a linear order

An implementation of a priority queue of elements of type  $'a$  must provide

- An implementation type  $'q$
- $empty :: 'q$

176

## Interface of implementation

The type of elements (= priorities)  $'a$  is a linear order

An implementation of a priority queue of elements of type  $'a$  must provide

- An implementation type  $'q$
- $insert :: 'a \Rightarrow 'q \Rightarrow 'q$
- $del\_min :: 'q \Rightarrow 'q$

176

## More operations

- $merge :: 'q \Rightarrow 'q \Rightarrow 'q$   
Often provided
- decrease key/priority  
Not easy in functional setting

177

## Correctness of implementation

A priority queue represents a **multiset** of priorities.  
Correctness proof requires:

Abstraction function:  $mset :: 'q \Rightarrow 'a \text{ multiset}$

178

## Correctness of implementation

A priority queue represents a **multiset** of priorities.  
Correctness proof requires:

Abstraction function:  $mset :: 'q \Rightarrow 'a \text{ multiset}$   
Invariant:  $invar :: 'q \Rightarrow bool$

178

## Correctness of implementation

Must prove  $invar\ q \implies$   
 $mset\ empty = \{\#\}$

179

## Correctness of implementation

Must prove  $invar\ q \implies$

179

## Correctness of implementation

Must prove  $invar\ q \implies$   
 $mset\ empty = \{\#\}$   
 $is\_empty\ q = (mset\ q = \{\#\})$

179

## Correctness of implementation

Must prove  $invar\ q \implies$

$mset\ empty = \{\#\}$

$is\_empty\ q = (mset\ q = \{\#\})$

$mset\ (insert\ x\ q) = mset\ q + \{\#x\#\}$

179

## Correctness of implementation

Must prove  $invar\ q \implies$

$mset\ empty = \{\#\}$

$is\_empty\ q = (mset\ q = \{\#\})$

$mset\ (insert\ x\ q) = mset\ q + \{\#x\#\}$

$mset\ q \neq \{\#\} \implies get\_min\ q = Min\_mset\ (mset\ q)$

179

## Correctness of implementation

Must prove  $invar\ q \implies$

$mset\ empty = \{\#\}$

$is\_empty\ q = (mset\ q = \{\#\})$

$mset\ (insert\ x\ q) = mset\ q + \{\#x\#\}$

$mset\ q \neq \{\#\} \implies get\_min\ q = Min\_mset\ (mset\ q)$

$mset\ q \neq \{\#\} \implies$

$mset\ (del\_min\ q) = mset\ q - \{\#get\_min\ q\#\}$

179

## Correctness of implementation

Must prove  $invar\ q \implies$

$mset\ empty = \{\#\}$

$is\_empty\ q = (mset\ q = \{\#\})$

$mset\ (insert\ x\ q) = mset\ q + \{\#x\#\}$

$mset\ q \neq \{\#\} \implies get\_min\ q = Min\_mset\ (mset\ q)$

$mset\ q \neq \{\#\} \implies$

$mset\ (del\_min\ q) = mset\ q - \{\#get\_min\ q\#\}$

$invar\ empty$

$invar\ (insert\ x\ q)$

$invar\ (del\_min\ q)$

179

## Terminology

A binary tree is a *heap* if for every subtree the root is  $\leq$  all elements in that subtree.

180

## Terminology

A binary tree is a *heap* if for every subtree the root is  $\leq$  all elements in that subtree.

$heap \langle \rangle = True$   
 $heap \langle l, m, r \rangle =$   
 $(heap \ l \wedge heap \ r \wedge (\forall x \in set\_tree \ l \cup set\_tree \ r. m \leq x))$

180

## Terminology

A binary tree is a *heap* if for every subtree the root is  $\leq$  all elements in that subtree.

$heap \langle \rangle = True$   
 $heap \langle l, m, r \rangle =$   
 $(heap \ l \wedge heap \ r \wedge (\forall x \in set\_tree \ l \cup set\_tree \ r. m \leq x))$

The term “heap” is frequently used synonymously with “priority queue”.

180

## Priority queue via heap

181

## Priority queue via heap

- $empty = \langle \rangle$
- $is\_empty\ h = (h = \langle \rangle)$
- $get\_min\ \langle -, a, - \rangle = a$
- Assume we have *merge*
- $insert\ a\ t = merge\ \langle \langle \rangle, a, \langle \rangle \rangle\ t$

181

## Priority queue via heap

- $empty = \langle \rangle$
- $is\_empty\ h = (h = \langle \rangle)$
- $get\_min\ \langle -, a, - \rangle = a$
- Assume we have *merge*
- $insert\ a\ t = merge\ \langle \langle \rangle, a, \langle \rangle \rangle\ t$
- $del\_min\ \langle l, a, r \rangle = merge\ l\ r$

181

## Priority queue via heap

A naive merge:

182

## Priority queue via heap

A naive merge:

$merge\ t_1\ t_2 = (\text{case } (t_1, t_2) \text{ of}$   
 $\langle \rangle, - \Rightarrow t_2 \mid$   
 $-, \langle \rangle \Rightarrow t_1 \mid$

182

## Priority queue via heap

A naive merge:

$merge\ t_1\ t_2 = (\text{case } (t_1, t_2) \text{ of}$   
   $(\langle \rangle, -) \Rightarrow t_2 \mid$   
   $(-, \langle \rangle) \Rightarrow t_1 \mid$   
   $(\langle l_1, a_1, r_1 \rangle, \langle l_2, a_2, r_2 \rangle) \Rightarrow$   
    if  $a_1 \leq a_2$  then  $\langle merge\ l_1\ r_1, a_1, t_2 \rangle$   
    else  $\langle t_1, a_2, merge\ l_2\ r_2 \rangle$

**Challenge:** how to maintain some kind of balance

182

15 Priority Queues

16 Leftist Heap

17 Priority Queue via Braun Tree

18 Binomial Heap

19 Skew Binomial Heap

183

## Leftist tree informally

The *rank* of a tree is the depth of the rightmost leaf.

In a *leftist tree*, the rank of every left child is  $\geq$  the rank of its right sibling.

185

## Leftist tree informally

The *rank* of a tree is the depth of the rightmost leaf.

In a *leftist tree*, the rank of every left child is  $\geq$  the rank of its right sibling.

Merge descends along the right spine.  
Thus rank bounds number of steps.

185



## Leftist tree informally

The *rank* of a tree is the depth of the rightmost leaf.

In a *leftist tree*, the rank of every left child is  $\geq$  the rank of its right sibling.

Merge descends along the right spine.

Thus rank bounds number of steps.

If rank of right child gets too large: swap with left child.

185

## Implementation type

**datatype**

*'a lheap = Leaf | Node nat ('a tree) 'a ('a tree)*

Abbreviations  $\langle \rangle$  and  $\langle h, l, a, r \rangle$  as usual

186

## Implementation type

**datatype**

*'a lheap = Leaf | Node nat ('a tree) 'a ('a tree)*

186

## Leftist tree

*rank :: 'a lheap  $\Rightarrow$  nat*

187

## Leftist tree

$rank :: 'a\ lheap \Rightarrow nat$   
 $rank \langle \rangle = 0$   
 $rank \langle -, -, -, r \rangle = rank\ r + 1$

187

## Leftist tree

$rank :: 'a\ lheap \Rightarrow nat$   
 $rank \langle \rangle = 0$   
 $rank \langle -, -, -, r \rangle = rank\ r + 1$

Node  $\langle n, l, a, r \rangle$ :  $n = \text{rank of node}$

$ltree :: 'a\ lheap \Rightarrow bool$

187

## Leftist tree

$rank :: 'a\ lheap \Rightarrow nat$   
 $rank \langle \rangle = 0$   
 $rank \langle -, -, -, r \rangle = rank\ r + 1$

Node  $\langle n, l, a, r \rangle$ :  $n = \text{rank of node}$

$ltree :: 'a\ lheap \Rightarrow bool$

$ltree \langle \rangle = True$

$ltree \langle n, l, -, r \rangle =$

$(n = rank\ r + 1 \wedge rank\ r \leq rank\ l \wedge ltree\ l \wedge ltree\ r)$

187

Principle: descend on the right

*merge*

189

## merge

Principle: descend on the right

$merge \langle \rangle t_2 = t_2$   
 $merge t_1 \langle \rangle = t_1$

189

## merge

Principle: descend on the right

$merge \langle \rangle t_2 = t_2$   
 $merge t_1 \langle \rangle = t_1$   
 $merge (\langle n_1, l_1, a_1, r_1 \rangle =: t_1) (\langle n_2, l_2, a_2, r_2 \rangle =: t_2) =$   
 $(if\ a_1 \leq a_2\ then\ node\ l_1\ a_1\ (merge\ r_1\ t_2)$   
 $\ else\ node\ l_2\ a_2\ (merge\ r_2\ t_1))$

$node :: 'a\ lheap \Rightarrow 'a \Rightarrow 'a\ lheap \Rightarrow 'a\ lheap$

189

## merge

Principle: descend on the right

$merge \langle \rangle t_2 = t_2$   
 $merge t_1 \langle \rangle = t_1$   
 $merge (\langle n_1, l_1, a_1, r_1 \rangle =: t_1) (\langle n_2, l_2, a_2, r_2 \rangle =: t_2) =$   
 $(if\ a_1 \leq a_2\ then\ node\ l_1\ a_1\ (merge\ r_1\ t_2)$   
 $\ else\ node\ l_2\ a_2\ (merge\ r_2\ t_1))$

$node :: 'a\ lheap \Rightarrow 'a \Rightarrow 'a\ lheap \Rightarrow 'a\ lheap$

$node\ l\ a\ r =$

$(let\ rl = rk\ l;\ rr = rk\ r$

$\ in\ if\ rr \leq rl\ then\ \langle rr + 1, l, a, r \rangle\ else\ \langle rl + 1, r, a, l \rangle)$

where  $rk\ \langle n, -, - \rangle = n$

189

## merge

Principle: descend on the right

$merge \langle \rangle t_2 = t_2$   
 $merge t_1 \langle \rangle = t_1$   
 $merge (\langle n_1, l_1, a_1, r_1 \rangle =: t_1) (\langle n_2, l_2, a_2, r_2 \rangle =: t_2) =$   
 $(if\ a_1 \leq a_2\ then\ node\ l_1\ a_1\ (merge\ r_1\ t_2)$   
 $\ else\ node\ l_2\ a_2\ (merge\ r_2\ t_1))$

$node :: 'a\ lheap \Rightarrow 'a \Rightarrow 'a\ lheap \Rightarrow 'a\ lheap$

189

## merge

$merge (\langle n_1, l_1, a_1, r_1 \rangle =: t_1) (\langle n_2, l_2, a_2, r_2 \rangle =: t_2) =$   
(if  $a_1 \leq a_2$  then  $node\ l_1\ a_1\ (merge\ r_1\ t_2)$   
else  $node\ l_2\ a_2\ (merge\ r_2\ t_1)$ )

190

## merge

$merge (\langle n_1, l_1, a_1, r_1 \rangle =: t_1) (\langle n_2, l_2, a_2, r_2 \rangle =: t_2) =$   
(if  $a_1 \leq a_2$  then  $node\ l_1\ a_1\ (merge\ r_1\ t_2)$   
else  $node\ l_2\ a_2\ (merge\ r_2\ t_1)$ )

Function *merge* terminates because  $size\ t_1 + size\ t_2$   
decreases with every recursive call.

190

## Functional correctness proofs

including preservation of *invar*

Straightforward

191

## Logarithmic complexity

Correlation of rank and size:

**Lemma**  $l_{tree}\ t \implies 2^{rank\ t} \leq |t|_1$

Complexity measures  $t\_merge$ ,  $t\_insert$   $t\_del\_min$ :  
count calls of *merge*.

192

## Logarithmic complexity

Correlation of rank and size:

**Lemma**  $ltree\ t \implies 2^{\text{rank}\ t} \leq |t|_1$

Complexity measures  $t\_merge$ ,  $t\_insert$   $t\_del\_min$ :  
count calls of  $merge$ .

**Lemma**  $t\_merge\ l\ r \leq \text{rank}\ l + \text{rank}\ r + 1$

192

## Logarithmic complexity

Correlation of rank and size:

**Lemma**  $ltree\ t \implies 2^{\text{rank}\ t} \leq |t|_1$

Complexity measures  $t\_merge$ ,  $t\_insert$   $t\_del\_min$ :  
count calls of  $merge$ .

**Lemma**  $t\_merge\ l\ r \leq \text{rank}\ l + \text{rank}\ r + 1$

**Corollary**  $[[ltree\ l; ltree\ r]]$

$\implies t\_merge\ l\ r \leq \log_2 |l|_1 + \log_2 |r|_1 + 1$

192

## Logarithmic complexity

Correlation of rank and size:

**Lemma**  $ltree\ t \implies 2^{\text{rank}\ t} \leq |t|_1$

Complexity measures  $t\_merge$ ,  $t\_insert$   $t\_del\_min$ :  
count calls of  $merge$ .

**Lemma**  $t\_merge\ l\ r \leq \text{rank}\ l + \text{rank}\ r + 1$

**Corollary**  $[[ltree\ l; ltree\ r]]$

$\implies t\_merge\ l\ r \leq \log_2 |l|_1 + \log_2 |r|_1 + 1$

**Corollary**

$ltree\ t \implies t\_insert\ x\ t \leq \log_2 |t|_1 + 2$

192

The screenshot shows a presentation window titled 'slides-fds.pdf'. The table of contents on the left lists the following items and their slide numbers:

Index	
Sorting	3
Binary Trees	30
Search Trees	44
Abstract Data Types	64
2-3 Trees	81
Union, Intersection, Difference on BSTs	126
Tries and Patricia Tries	142
Priority Queues	171

The main content area of the slide contains the text: "Can we avoid the rank info in each node?"

192

```
Isabelle2017 - Leftist_Heap.thy (modified)
File Edit Search Markers Folding View Utilities Macros Plugins Help

Leftist_Heap.thy ($ISABELLE_HOME/src/HOL/Data_Structures)
44 fun get_min :: "'a heap => 'a" where
45 "get_min(Node n l a r) = a"
46
47 text <For function <merge>:>
48 unbundle pattern_aliases
49 (*declare size_prod_measure[measure_function]*)
50
51 fun merge :: "'a::ord heap => 'a heap => 'a heap" where
52 "merge t1 Leaf = t1" |
53 "merge t1 Leaf = t1" |
54 "merge (Node n1 l1 a1 r1) (Node n2 l2 a2 r2) =>
55  (if a1 <= a2 then node l1 a1 (merge r1 t2)
56   else node l2 a2 (merge r2 t1))"
57
58 Lemma merge_code: "merge t1 t2 = (case (t1,t2) of
59  (Leaf, _) => t2 |
60  (_, Leaf) => t1 |
61  (Node n1 l1 a1 r1, Node n2 l2 a2 r2) =>
62   if a1 <= a2 then node l1 a1 (merge r1 t2) else node l2 a2 (merge r2 t1))"
63 by(induction t1 t2 rule: merge.induct) (simp_all split: tree.split)
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

Output | Query | Sledgehammer | Symbols
52.23 (1.240/7571) Input/output complete (isabelle.isabelle.UTF-8-isabelle)tmr o UG 993/1.004MB 9:34 AM
debian 1 2 3 4 iamlich@lapnikow10: ~/opt/isabelle20... Isabelle2017 - Leftist_Heap.thy (modified) Isabelle2017 - Leftist_Heap.thy (modified) 09:34:23
```

```
Isabelle2017 - Leftist_Heap.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help

Leftist_Heap.thy ($ISABELLE_HOME/src/HOL/Data_Structures)
44 fun get_min :: "'a heap => 'a" where
45 "get_min(Node n l a r) = a"
46
47 text <For function <merge>:>
48 unbundle pattern_aliases
49 declare size_prod_measure[measure_function]
50
51 fun merge :: "'a::ord heap => 'a heap => 'a heap" where
52 "merge Leaf t2 = t2" |
53 "merge t1 Leaf = t1" |
54 "merge (Node n1 l1 a1 r1 = t1) (Node n2 l2 a2 r2 = t2) =
55  (if a1 <= a2 then node l1 a1 (merge r1 t2)
56   else node l2 a2 (merge r2 t1))"
57
58 Lemma merge_code: "merge t1 t2 = (case (t1,t2) of
59  (Leaf, _) => t2 |
60  (_, Leaf) => t1 |
61  (Node n1 l1 a1 r1, Node n2 l2 a2 r2) =>
62   if a1 <= a2 then node l1 a1 (merge r1 t2) else node l2 a2 (merge r2 t1))"
63 by(induction t1 t2 rule: merge.induct) (simp_all split: tree.split)
64

Output | Query | Sledgehammer | Symbols
49.9 (1.120/7567) null parsing complete, 0 error(s) (isabelle.isabelle.UTF-8-isabelle)tmr o UG 1.44/1.208MB 9:35 AM
debian 1 2 3 4 iamlich@lapnikow10: ~/opt/isabelle20... Isabelle2017 - Leftist_Heap.thy Isabelle2017 - Leftist_Heap.thy 09:35:34
```

```
Isabelle2017 - Leftist_Heap.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help

Leftist_Heap.thy ($ISABELLE_HOME/src/HOL/Data_Structures)
44 fun get_min :: "'a heap => 'a" where
45 "get_min(Node n l a r) = a"
46
47 text <For function <merge>:>
48 unbundle pattern_aliases
49 declare size_prod_measure[measure_function]
50
51 fun merge :: "'a::ord heap => 'a heap => 'a heap" where
52 "merge Leaf t2 = t2" |
53 "merge t1 Leaf = t1" |
54 "merge (Node n1 l1 a1 r1 = t1) (Node n2 l2 a2 r2 = t2) =
55  (if a1 <= a2 then node l1 a1 (merge r1 t2)
56   else node l2 a2 (merge r2 t1))"
57
58 Lemma merge_code: "merge t1 t2 = (case (t1,t2) of
59  (Leaf, _) => t2 |
60  (_, Leaf) => t1 |
61  (Node n1 l1 a1 r1, Node n2 l2 a2 r2) =>
62   if a1 <= a2 then node l1 a1 (merge r1 t2) else node l2 a2 (merge r2 t1))"
63 by(induction t1 t2 rule: merge.induct) (simp_all split: tree.split)
64

Output | Query | Sledgehammer | Symbols
52.1 (1.214/7567) (isabelle.isabelle.UTF-8-isabelle)tmr o UG 97/1.208MB 9:35 AM
debian 1 2 3 4 iamlich@lapnikow10: ~/opt/isabelle20... Isabelle2017 - Leftist_Heap.thy Isabelle2017 - Leftist_Heap.thy 09:35:50
```

```
Isabelle2017 - Leftist_Heap.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help

Leftist_Heap.thy ($ISABELLE_HOME/src/HOL/Data_Structures)
44 fun get_min :: "'a heap => 'a" where
45 "get_min(Node n l a r) = a"
46
47 text <For function <merge>:>
48 unbundle pattern_aliases
49 declare size_prod_measure[measure_function]
50
51 fun merge :: "'a::ord heap => 'a heap => 'a heap" where
52 "merge Leaf t2 = t2" |
53 "merge t1 Leaf = t1" |
54 "merge (Node n1 l1 a1 r1 = t1) (Node n2 l2 a2 r2 = t2) =
55  (if a1 <= a2 then node l1 a1 (merge r1 t2)
56   else node l2 a2 (merge r2 t1))"
57
58 Lemma merge_code: "merge t1 t2 = (case (t1,t2) of
59  (Leaf, _) => t2 |
60  (_, Leaf) => t1 |
61  (Node n1 l1 a1 r1, Node n2 l2 a2 r2) =>
62   if a1 <= a2 then node l1 a1 (merge r1 t2) else node l2 a2 (merge r2 t1))"
63 by(induction t1 t2 rule: merge.induct) (simp_all split: tree.split)
64

Output | Query | Sledgehammer | Symbols
54.30 (1.289/7567) (isabelle.isabelle.UTF-8-isabelle)tmr o UG 85/1.208MB 9:36 AM
debian 1 2 3 4 iamlich@lapnikow10: ~/opt/isabelle20... Isabelle2017 - Leftist_Heap.thy Isabelle2017 - Leftist_Heap.thy 09:36:53
```

```

169 subsection "Complexity"
170
171 Lemma pow2_rank_size1: "ltree t  $\implies$  2 ^ rank t  $\leq$  size1 t"
172 proof(induction t)
173 case Leaf show ?case by simp
174 next
175 case (Node n l a r)
176 hence "rank r  $\leq$  rank l" by simp
177 hence *: "(2::nat) ^ rank r  $\leq$  2 ^ rank l" by simp
178 have "(2::nat) ^ rank (n, l, a, r) = 2 ^ rank r + 2 ^ rank r"
179 by(simp add: mult_2)
180 also have "...  $\leq$  size1 l + size1 r"
181 using Node * by (simp del: power_increasing_iff)
182 also have "... = size1 (n, l, a, r)" by simp
183 finally show ?case .
184 qed
185
186 text<Explicit termination argument: sum of sizes>
187
188 fun t_merge :: "'a::ord lheap  $\implies$  'a lheap  $\implies$  nat" where
189

```

slides-fds.pdf

193 | (612 of 716)

219,989

Index

- Sorting 3
- Binary Trees 30
- Search Trees 44
  - Abstract Data Types 64
  - 2-3 Trees 81
  - Union, Intersection, Difference on BSTs 126
  - Trees and Patricia Trees 142
  - Priority Queues 171

Can we avoid the rank info in each node?

182.1 (5393/7567) (isabelle.isabelle.UTF-8+isabelle)lmm - UG - 31/1287MB 9:39 AM

debian | 1 2 3 4 | lamlich@lapnikow10: ~/opt/isabelle20... | Isabelle2017 - Leftist\_Heap.thy | Isabelle2017 - Leftist\_Heap.thy | 09:39:39

slides-fds.pdf

193 | (612 of 716)

219,989

Index

- Sorting 3
- Binary Trees 30
- Search Trees 44
  - Abstract Data Types 64
  - 2-3 Trees 81
  - Union, Intersection, Difference on BSTs 126
  - Trees and Patricia Trees 142
  - Priority Queues 171

Can we avoid the rank info in each node?

193

debian | 1 2 3 4 | lamlich@lapnikow10: ~/lehre/FDS/SS1... | slides-fds.pdf | 09:41:49

## What is a Braun tree?

$braun :: 'a\ tree \implies bool$

$braun \langle \rangle = True$

$braun \langle l, x, r \rangle = (|r| \leq |l| \wedge |l| \leq |r| + 1 \wedge braun\ l \wedge braun\ r)$

1

196

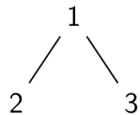
## What is a Braun tree?

$braun :: 'a\ tree \Rightarrow bool$

$braun \langle \rangle = True$

$braun \langle l, x, r \rangle =$

$(|r| \leq |l| \wedge |l| \leq |r| + 1 \wedge braun\ l \wedge braun\ r)$



196

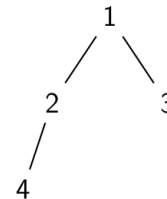
## What is a Braun tree?

$braun :: 'a\ tree \Rightarrow bool$

$braun \langle \rangle = True$

$braun \langle l, x, r \rangle =$

$(|r| \leq |l| \wedge |l| \leq |r| + 1 \wedge braun\ l \wedge braun\ r)$



196

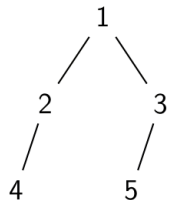
## What is a Braun tree?

$braun :: 'a\ tree \Rightarrow bool$

$braun \langle \rangle = True$

$braun \langle l, x, r \rangle =$

$(|r| \leq |l| \wedge |l| \leq |r| + 1 \wedge braun\ l \wedge braun\ r)$



196

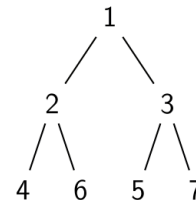
## What is a Braun tree?

$braun :: 'a\ tree \Rightarrow bool$

$braun \langle \rangle = True$

$braun \langle l, x, r \rangle =$

$(|r| \leq |l| \wedge |l| \leq |r| + 1 \wedge braun\ l \wedge braun\ r)$



**Lemma**  $braun\ t \Rightarrow 2^{h(t)} \leq 2 * |t| + 1$

196



## Idea of invariant maintenance

$braun \langle \rangle = True$   
 $braun \langle l, x, r \rangle =$   
 $(|r| \leq |l| \wedge |l| \leq |r| + 1 \wedge braun l \wedge braun r)$

197

## Idea of invariant maintenance

$braun \langle \rangle = True$   
 $braun \langle l, x, r \rangle =$   
 $(|r| \leq |l| \wedge |l| \leq |r| + 1 \wedge braun l \wedge braun r)$

Add element:

197

## Idea of invariant maintenance

$braun \langle \rangle = True$   
 $braun \langle l, x, r \rangle =$   
 $(|r| \leq |l| \wedge |l| \leq |r| + 1 \wedge braun l \wedge braun r)$

Add element: to right subtree, then swap subtrees

**Goal:**  $|l| \leq |r| + 1 \wedge |r| + 1 \leq |l| + 1$

197

## Idea of invariant maintenance

$braun \langle \rangle = True$   
 $braun \langle l, x, r \rangle =$   
 $(|r| \leq |l| \wedge |l| \leq |r| + 1 \wedge braun l \wedge braun r)$

Add element: to right subtree, then swap subtrees

**Goal:**  $|l| \leq |r| + 1 \wedge |r| + 1 \leq |l| + 1$   $\square$

Remove element:

197

## Idea of invariant maintenance

$braun \langle \rangle = True$   
 $braun \langle l, x, r \rangle =$   
 $(|r| \leq |l| \wedge |l| \leq |r| + 1 \wedge braun\ l \wedge braun\ r)$

Add element: to right subtree, then swap subtrees

**Goal:**  $|l| \leq |r| + 1 \wedge |r| + 1 \leq |l| + 1$

Remove element: from left subtree, then swap subtrees

**Goal:**  $|l| - 1 \leq |r| \wedge |r| \leq |l|$

197

## Idea of invariant maintenance

$braun \langle \rangle = True$   
 $braun \langle l, x, r \rangle =$   
 $(|r| \leq |l| \wedge |l| \leq |r| + 1 \wedge braun\ l \wedge braun\ r)$

Add element: to right subtree, then swap subtrees

**Goal:**  $|l| \leq |r| + 1 \wedge |r| + 1 \leq |l| + 1$

Remove element: from left subtree, then swap subtrees

**Goal:**  $|l| - 1 \leq |r| \wedge |r| \leq |l|$

197

## Idea of invariant maintenance

$braun \langle \rangle = True$   
 $braun \langle l, x, r \rangle =$   
 $(|r| \leq |l| \wedge |l| \leq |r| + 1 \wedge braun\ l \wedge braun\ r)$

Add element: to right subtree, then swap subtrees

**Goal:**  $|l| \leq |r| + 1 \wedge |r| + 1 \leq |l| + 1$

Remove element: from left subtree, then swap subtrees

**Goal:**  $|l| - 1 \leq |r| \wedge |r| \leq |l|$

197

## *insert*

$insert :: 'a \Rightarrow 'a\ tree \Rightarrow 'a\ tree$

$insert\ a\ \langle \rangle = \langle \langle \rangle, a, \langle \rangle \rangle$

$insert\ a\ \langle l, x, r \rangle =$

$(if\ a < x\ then\ \langle insert\ x\ r, a, l \rangle\ else\ \langle insert\ a\ r, x, l \rangle)$

199

## *insert*

```
insert :: 'a ⇒ 'a tree ⇒ 'a tree
insert a ⟨⟩ = ⟨⟨⟩, a, ⟨⟩⟩
insert a ⟨l, x, r⟩ =
  (if a < x then ⟨insert x r, a, l⟩ else ⟨insert a r, x, l⟩)
```

Correctness and preservation of invariant straightforward.

199

## *del\_min*

```
del_min :: 'a tree ⇒ 'a tree
del_min ⟨⟩ = ⟨⟩
del_min ⟨⟨⟩, x, r⟩ = ⟨⟩
del_min ⟨l, x, r⟩ =
  (let (y, l') = del_left l in sift_down r y l')
```

200

## *del\_min*

```
del_min :: 'a tree ⇒ 'a tree
del_min ⟨⟩ = ⟨⟩
del_min ⟨⟨⟩, x, r⟩ = ⟨⟩
del_min ⟨l, x, r⟩ =
  (let (y, l') = del_left l in sift_down r y l')
```

- 1 Delete leftmost element  $y$
- 2 Sift  $y$  from the root down

200

## *sift\_down*

```
sift_down :: 'a tree ⇒ 'a ⇒ 'a tree ⇒ 'a tree
```

202

## sift\_down

$sift\_down :: 'a\ tree \Rightarrow 'a \Rightarrow 'a\ tree \Rightarrow 'a\ tree$   
 $sift\_down \langle \rangle a \langle \rangle = \langle \langle \rangle, a, \langle \rangle \rangle$   
 $sift\_down \langle \langle \rangle, x, \langle \rangle \rangle a \langle \rangle =$   
if  $a \leq x$  then  $\langle \langle \langle \rangle, x, \langle \rangle \rangle, a, \langle \rangle \rangle$   
else  $\langle \langle \langle \rangle, a, \langle \rangle \rangle, x, \langle \rangle \rangle$   
 $sift\_down \langle \langle l_1, x_1, r_1 \rangle =: t_1 \rangle a \langle \langle l_2, x_2, r_2 \rangle =: t_2 \rangle =$   
if  $a \leq x_1 \wedge a \leq x_2$  then  $\langle t_1, a, t_2 \rangle$   
else if  $x_1 \leq x_2$  then  $\langle sift\_down\ l_1\ a\ r_1, x_1, t_2 \rangle$   
else  $\langle t_1, x_2, sift\_down\ l_2\ a\ r_2 \rangle$

Maintains *braun*

202

## Logarithmic complexity

Running time of *insert*, *del\_left* and *sift\_down* (and therefore *del\_min*) bounded by height

204

Archive of Formal Proofs - Mozilla Firefox  
https://www.isa-afp.org/topics.html

INDEX BY TOPIC

**Computer Science**

**Automata and Formal Languages**  
Posix-Lexing LocalLexing KBPs Regular-Sets Regex\_Equivalence  
MSO\_Regex\_Equivalence Formula\_Derivatives Myhill-Nerode CYK Presburger-Automata Functional-Automata Statecharts Stuttering\_Equivalence  
Coinductive\_Languages Tree-Automata Kleene\_Algebra KAT\_and\_DRA KAD  
Regular\_Algebras Markov\_Models Probabilistic\_System\_Zoo CAVA\_Automata LTL  
LTL\_to\_GBA CAVA\_LTL\_Modelchecker Probabilistic\_Timed\_Automata  
Finite\_Automata\_HF\_LTL\_to\_DRA Timed\_Automata Stochastic\_Matrices  
Buchi\_Complementation Transition\_Systems\_and\_Automata

**Algorithms**  
Knuth\_Morris\_Pratt Probabilistic\_While Comparison\_Sort\_Lower\_Bound  
Quick\_Sort\_Cost DFS\_Framework Prpu\_Maxflow TortoiseHare Floyd\_Warshall  
Roy\_Floyd\_Warshall Selection\_Heap\_Sort Dijkstra\_Shortest\_Path  
EdmondsKarp\_Maxflow VerifyThis2018 CYK\_Boolean\_Expression\_Checkers\_Depth-  
First-Search FFT\_Gauss-Jordan-Elim-Fun UpDown\_Scheme GraphMarkingIBP  
Efficient-Mergesort SATSolverVerification Transitive-Closure Transitive-Closure-II

Home  
About  
Submission  
Updating Entries  
Using Entries  
Search  
Statistics  
Index  
Download

Archive of Formal Proofs - Mozilla Firefox  
https://www.isa-afp.org/index.html

**Computer Science**

**Automata and Formal Languages**  
Posix-Lexing LocalLexing KBPs Regular-Sets Regex\_Equivalence  
MSO\_Regex\_Equivalence Formula\_Derivatives Myhill-Nerode CYK Presburger-Automata Functional-Automata Statecharts Stuttering\_Equivalence  
Coinductive\_Languages Tree-Automata Kleene\_Algebra KAT\_and\_DRA KAD  
Regular\_Algebras Markov\_Models Probabilistic\_System\_Zoo CAVA\_Automata LTL  
LTL\_to\_GBA CAVA\_LTL\_Modelchecker Probabilistic\_Timed\_Automata  
Finite\_Automata\_HF\_LTL\_to\_DRA Timed\_Automata Stochastic\_Matrices  
Buchi\_Complementation Transition\_Systems\_and\_Automata

**Algorithms**  
Knuth\_Morris\_Pratt Probabilistic\_While Comparison\_Sort\_Lower\_Bound  
Quick\_Sort\_Cost DFS\_Framework Prpu\_Maxflow TortoiseHare Floyd\_Warshall  
Roy\_Floyd\_Warshall Selection\_Heap\_Sort Dijkstra\_Shortest\_Path  
EdmondsKarp\_Maxflow VerifyThis2018 CYK\_Boolean\_Expression\_Checkers\_Depth-  
First-Search FFT\_Gauss-Jordan-Elim-Fun UpDown\_Scheme GraphMarkingIBP  
Efficient-Mergesort SATSolverVerification Transitive-Closure Transitive-Closure-II  
MuchAdoAboutTwo First\_Order\_Terms Polynomials Gabow\_SCC Monad\_Memo\_DP  
Hidden\_Markov\_Models Gauss\_Jordan Echelon\_Form QR\_Decomposition Hermite  
Imperative\_Insertion\_Sort Deep\_Learning Formal\_SSA ROBDD Groebner\_Bases  
Median\_Of\_Medians\_Selection IMAP-CRDT Fisher\_Yates Diophantine\_Eqns\_Lin\_Hom  
Taylor\_Models LLL\_Basis\_Reduction Optimal\_BST **Distributed:** DiskPaxos GenClock  
ClockSynchronst Heard\_Of Consensus Refined Abortable Linearizable Modules CRDT

Home  
About  
Submission  
Updating Entries  
Using Entries  
Search  
Statistics  
Index  
Download

Session Priority\_Queue\_Braun (Isabelle2017: October 2017) - Mozilla Firefox

Session Priority\_Queue\_Braun X

https://www.isa-afp.org/browser\_info/current/AFP/Priority\_Queue\_Braun

Most Visited Latest Headlines Inbox - Ipeter1@vt... Kletter-TODO Tourenplaner • Dein... KOMPASS.Maps reload with eAccess C+ASM semantics VT

# Session Priority\_Queue\_Braun



View [theory dependencies](#)  
View [document](#)  
View [outline](#)

## Theories

- [Priority\\_Queue\\_Braun](#)

debian | 1 2 3 4 | tammich@lapnikow10: ~/lehre/FDS/SS1... | slides-fds.pdf | Session Priority\_Queue\_Braun (Isabelle20... | 10:03:37