CSE 326: Data Structures

Topic #5: Skew Heaps and Binomial Qs

Ashish Sabharwal

Autumn, 2003

Today's Outline

- Binary Heaps: average runtime of *insert*
- Leftist Heaps: re-do proof of property #1

- **Amortized Runtime**
- **Skew Heaps**
- **Binomial Queues**

- Comparing Implementations of Priority Qs

Binary Heaps: Average runtime of *Insert*

Recall: Insert-in-Binary-Heap(x) {
    Put x in the next available position
    percolateUp(last node)
}

How long does this percolateUp(last node) take?

- Worst case: Θ(tree height), i.e. $\Theta(\log n)$
- Average case: $\Theta(1)$ Why??

Average runtime of insert in binary heap = $\Theta(1)$

Right Path in a Leftist Tree is Short (#1)

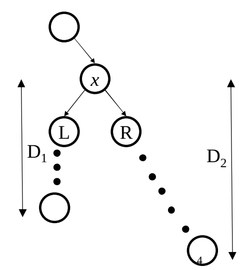
Claim: The right path is as short as *any* in the tree.

Proof: (By contradiction)

Pick a shorter path: $D_1 < D_2$
Say it diverges from right path at x

$npl(L) \le D_1-1$ because of the path of length D_1-1 to null

$npl(R) \ge D_2-1$ because every node on right path is leftist



Leftist property at x violated!

A Twist in Complexity Analysis: The Amortized Case

If a sequence of M operations takes O(M f(n)) time, we say the amortized runtime is O(f(n)).

- Worst case time *per operation* can still be large, say O(n)
- Worst case time for *any* sequence of M operations is O(M f(n))

- Average time *per operation* for *any* sequence is O(f(n))

Is this the same as average time?

Skew Heaps

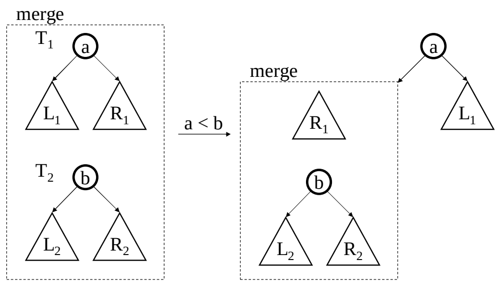
Problems with leftist heaps

- extra storage for npl
- extra complexity/logic to maintain and check npl
- two pass iterative merge (requires stack!)
- right side is "often" heavy and requires a switch

Solution: skew heaps

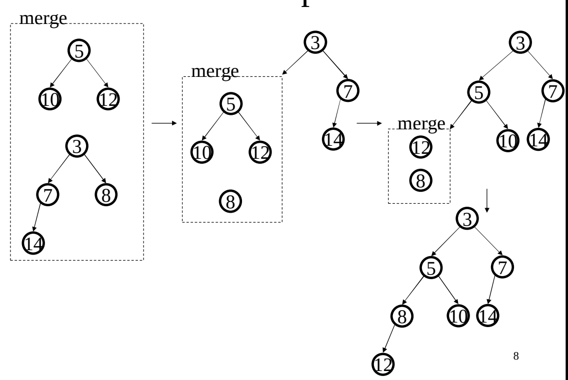
- blind adjusting version of leftist heaps
- merge *always* switches children when fixing right path
- iterative method has only one pass
- amortized time for merge, insert, and deleteMin is $\Theta(\log n)$
- however, worst case time for all three is $\Theta(n)$

Merging Two Skew Heaps



Only one step per iteration, with children *always* switched

Example



Skew Heap Code

```
void merge(heap1, heap2) {
  case {
      heap1 == NULL: return heap2;
      heap2 == NULL: return heap1;
      heap1.findMin() < heap2.findMin():
             temp = heap1.right;
             heap1.right = heap1.left;
             heap1.left = merge(heap2, temp);
             return heap1;
      otherwise:
             return merge(heap2, heap1);
  }
}
```


Runtime Analysis: Worst-case and Amortized

- No worst case guarantee on right path length!
- All operations rely on merge

  ⇒ worst case complexity of all ops =

- Will do amortized analysis later in the course (see chapter 11 if curious)
- Result: M merges take time $M \log n$

  ⇒ amortized complexity of all ops =

ATaB: Comparing Heaps

- Binary Heaps
- Leftist Heaps
- d-Heaps
- Skew Heaps

Still scope for improvement!

Yet Another Data Structure: Binomial Queues

- Structural property
  - Forest of binomial trees with at most one tree of any height

What's a forest?

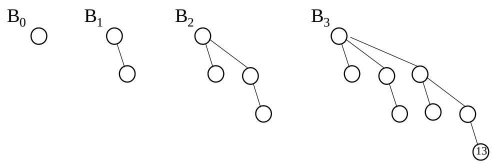
What's a binomial tree?

- Order property
  - Each binomial tree has the heap-order property

My opinion: Beautiful and elegant!

The Binomial Tree, B_h

- B_h has height h and exactly 2^h nodes
- B_h is formed by making B_{h-1} a child of another B_{h-1}
- Root has exactly h children
- Number of nodes at depth d is binomial coeff. $\binom{h}{d}$
  - Hence the name; we will *not* use this last property



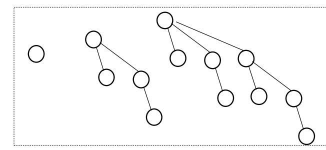
13

Binomial Q with n elements

Binomial Q with n elements has a *unique* structural representation in terms of binomial trees!

Write n in binary: $n = 1101_{\text{(base 2)}} = 13_{\text{(base 10)}}$

1 B_3 1 B_2 No B_1 1 B_0



14

Properties of Binomial Q

- At most one binomial tree of any height
- n nodes $\Rightarrow$ binary representation is of size ?
  $\Rightarrow$ deepest tree has height ?
  $\Rightarrow$ number of trees is ?

Define: height(forest F) = $\max_{\text{tree T in F}}$ { height(T) }

Binomial Q with n nodes has height $\Theta(\log n)$

15

Operations on Binomial Q

- Will again define *merge* as the base operation
  - insert, deleteMin, buildBinomialQ will use merge
- Can we do increaseKey efficiently? decreaseKey?
- What about findMin?

16

Merging Two Binomial Qs

Essentially like adding two binary numbers!

1. Combine the two forests
2. For k from 1 to maxheight {
   a. $m \leftarrow$ total number of B_k's in the two BQs
   b. if m=0: continue;
   c. if m=1: continue;
   d. if m=2: combine the two B_k's to form a B_{k+1}
   e. if m=3: retain one B_k and combine the other two to form a B_{k+1}

}

of 1's
0+0 = 0
1+0 = 1
1+1 = 1+c
1+1+c = 1+c

Claim: When this process ends, the forest has at most one tree of any height

17

Complexity of Merge

Constant time for each height
Max height is log n

$\Rightarrow$ worst case running time = $\Theta($ $)$

18

Insert in a Binomial Q

Insert(x): Similar to leftist or skew heap

runtime

Worst case complexity: same as merge

$\Theta($ $)$

Average case complexity: $\Theta(1)$

Why?? *Hint: Think of adding 1 to 1101*

19

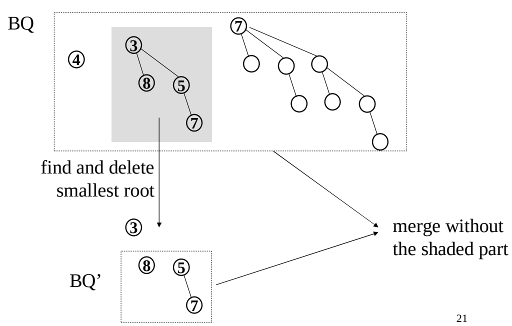
deleteMin in Binomial Q

deleteMin: Similar to leftist and skew heaps

A tiny bit more complicated

20

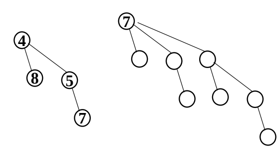
deleteMin: Example



21

deleteMin: Example

Result:



runtime:

22

buildBinomialQ

Call insert n times on an initially empty BQ

runtime: naïve O(n log n)

careful analysis $\Theta(n)$

idea: count the number of times one needs to combine trees

23

To Do

- Project #1 due tonight!
  - Bring printout to section tomorrow
- Written homework #1
  - will be out later today; I'll send an email
- Revise binary search tree basics
- Begin reading chapter 4 in the book

24