CSE 373: Heaps (Priority Queues)

Pete Morcos
University of Washington
4/10/00

http://www.cs.washington.edu/education/courses/cse373/00sp

The Problem

- In some situations, we want to quickly get the smallest (or largest) item from a group
  - Emergency room patients, rated by severity
  - Simulation events, ranked by when they start
- So we want an ADT that can efficiently perform:
  - FindMin (or FindMax)
  - DeleteMin
  - and of course Insert
- ADTs in this class are called *priority queues*
  - Like a queue, but not FIFO anymore

What we know so far

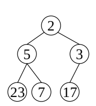
- Lists
  - If kept sorted, Insert O(N), DeleteMin O(1)
  - If not, Insert O(1), DeleteMin O(N)
- Binary Search Trees
  - Insert O(log N), FindMin O(log N), DeleteMin O(N)
    - if we assume a previous FindMin, DeleteMin is O(1). why?
- Hash Tables
  - Insert O(1), FindMin ?, DeleteMin ?
  - for answer, see work by _________, April 2000.
    YOUR NAME HERE

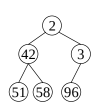
Why not use BSTs?

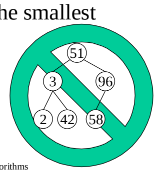
- Binary search trees look pretty good
- We can do slightly better
- As always, we should look at the assumptions and requirements to see how
  - BSTs maintain a strong left/right ordering
  - BSTs provide efficient Find, not just FindMin
  - We only need FindMin/DeleteMin
- We can relax the BST requirements to get a slightly faster data structure for our purpose

Heaps

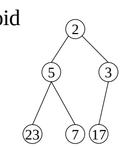
- A (binary) heap is a **complete** binary tree that satisfies the *heap order property*.
- Every node is smaller than its children
  - BST: every node bigger than left child, smaller than right
- Thus, the top node is always the smallest



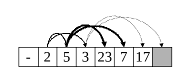




Array Implementation



- By requiring tree to be complete, can avoid use of pointers
- Recall trick mentioned before:
  - Children of A[i] are A[2i], A[2i + 1]
  - Keep track of size (in this case, 6)
- Unlike BSTs, very easy to maintain completeness property
  - Restriction on new node placement is softer—either side OK

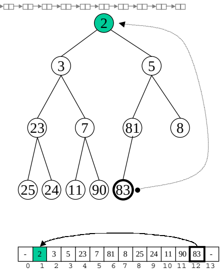


Heap ADT Operations

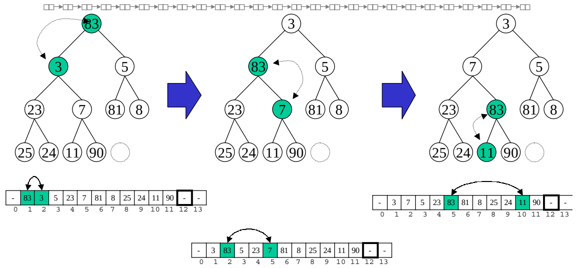
- Insert(Heap H, ElementType N)
- ElementType FindMin(Heap H)
- ElementType DeleteMin(Heap H)
- note: no ops to scan through heap; only min

DeleteMin

- Remove top node
- We don't just replace with smallest child—could violate completeness
  - Consider shifting up 3, then 7, then 11
- Heap will be 1 node smaller, so we know that last slot will empty out
- So, steps are:
  - move last item to top (guarantees that heap property is maintained)
  - then allow it to *percolate down* to its natural position

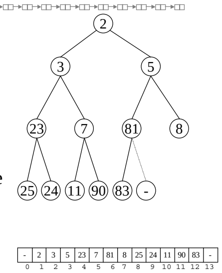


DeleteMin cont'd



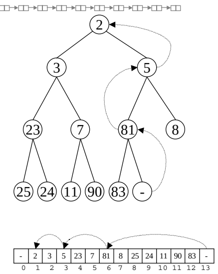
Insert

- As with DeleteMin, can't start from top and work our way down
- Steps:
  - Create new hole at end of array
  - If item can legally go there, done
  - Else, slide parent down
  - Repeat checks recursively



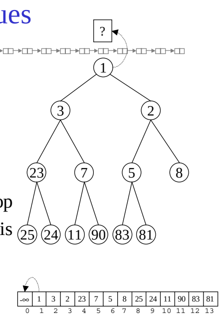
Insert cont'd

- Consider inserting 1
- Can't go in array slot #13, so slide down 81 (from slot #6)
- Can't go in slot #6, so slide down 5 (from slot #3)
- Can't go in slot #3, so slide down 2 (from slot #1)
- Done
  - need a test for top of array...



Sentinel Values

- At every step, have to do 2 tests
  - Are we at top?
  - Is inserted item > than parent?
- If we put a very negative number in slot 0, then can skip first test
- Second test automatically stops at top
- A false data value used as a marker is known as a *sentinel*
  - Usually used for efficiency
  - Subtle, so comment your code!



Heap ADT Analysis

- Space: O(N)
  - (sort of): need to over-allocate space (typical for arrays)
  - Just need one extra variable to keep track of size
  - Efficient: really only uses N + 2 space (pointers would be 3N + 1)
- Insert: O(log N)
- DeleteMin: O(log N)
- FindMin: O(1)
- BuildHeap (i.e. from N inputs)
  - Since Insert is log N, might expect O(N log N)
  - Actually only takes O(N) [see book]
    - Treat input array as a heap, then “fixup” by percolating down non-leaves

Optional Operations

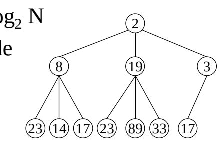
- Although heaps are defined to hide all values but the minimum, sometimes useful to be able to modify any node.
- DecreaseKey – lower value of (any) node
  - Need a PercolateUp routine to slide node to right place
- IncreaseKey – raise value of (any) node
  - PercolateDown
- Delete – delete (any) node
  - DecreaseKey by infinity, then do DeleteMin

MaxHeaps

- Our heap definition has been defined around minimum values—a MinHeap
- Can trivially change to use maximums instead
  - DeleteMax instead of DeleteMin
  - Heap order property: parent *greater* than children
- Can’t easily support both DeleteMin and DeleteMax
  - How long would DeleteMax take on a MinHeap?

d-Heaps

- The heap we’ve been talking about is a *binary heap*, and is the most common
- A *d*-heap has *d* children per node
- 3-heap shown: children of *i* are 3*i*-1, 3*i*, 3*i*+1
- Shallower: $\log_d$ N instead of $\log_2$ N
- But, more children at each node
  - *d*-1 compares to find smallest



Priority Queues

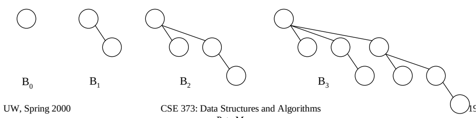
- A (min/max/*d*-)heap is just one kind of priority queue
- Recall that a PQ is simply an ADT that provides a DeleteMin operation (and Insert of course)
- Can design other data structurs besides heaps to do this efficiently. Book presents 3 alternatives
- Big advantage is that they can efficiently perform a Merge of two PQs, unlike heaps
- We will briefly discuss one, *binomial queues*

Binomial Queues

- In a BQ, a set of data of size N is divided into a forest of *binomial trees*
- Each binomial tree has the heap order property
  - Thus, overall minimum is at the top of one of the trees
- Merging two BQs is broken down into individual mergers of the binomial trees, which turns out to be easy due to their structure

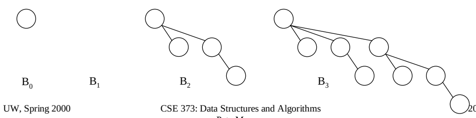
Binomial Trees

- Can have various binomial trees B_i
  - B_i has 2^i nodes
  - Restricted structure means there is only one possible B_i
- Defined recursively
  - B_0 is a single node
  - B_k is a B_{k-1} with another B_{k-1} attached to root



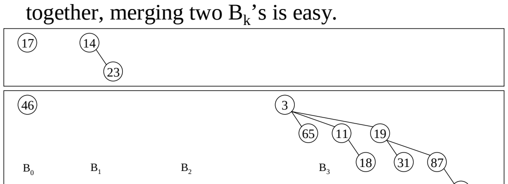
A Binomial Queue is a Forest

- Any number N can be written as a sum of powers of two
  - That's the whole idea of binary numbers
- Each binomial tree has size 2^i, so a unique set of trees is necessary to hold N nodes
- Example: N = 13 (binary 1011)



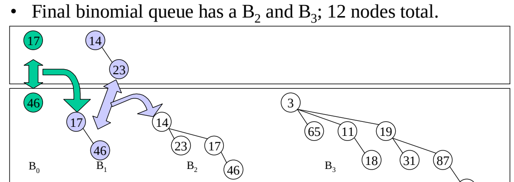
Operations

- FindMin scans all trees—at most log N of them.
- Merge: simply add the trees of the same size in each forest. Since B_{k+1} is just two B_k's attached together, merging two B_k's is easy.



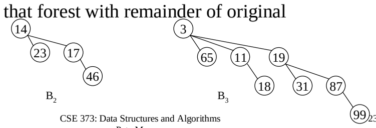
Merge

- To merge, larger root becomes child of smaller root.
- Merging the two B_0's creates a new B_1.
- Merging original B_1 with new one creates a B_2
- Merging the B_2 and B_3 groups is trivial
- Final binomial queue has a B_2 and B_3; 12 nodes total.



Insert, DeleteMin

- To insert, treat new node as a 1-node BQ, then merge with existing BQ
- For DeleteMin:
  - First find smallest root, tree B_k
  - Remove B_k from the BQ
  - Remove root of B_k, creating forest $B_{k-1}, B_{k-2}, \ldots, B_1, B_0$
  - Merge that forest with remainder of original



Priority Queue Summary

- If the minimum node in a set is the only one we care about, can use simpler ADT than a BST
- Binary heap is most common
- All ops are O(log N) worst case
- Merging heaps is not efficient, so alternatives like binomial queues can be devised
- Priority Queues useful when things like priority, time order, or repeated minimum searches are needed