

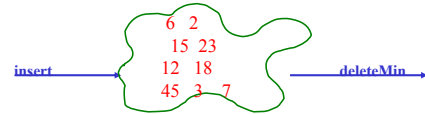
Priority Queues (Heaps)

Chapter 6 in Weiss

Reading: pp.211-222, pp. 222-225 examples.

Priority Queue ADT

- Checkout line at the supermarket
- Printer queues
- operations: insert, deleteMin



Implementations of Priority Queue ADT

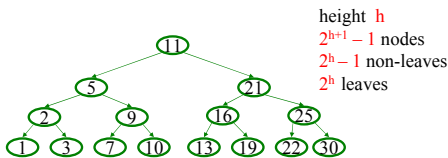
	insert	deleteMin
Unsorted list (Array)		
Unsorted list (Linked-List)		
Sorted list (Array)		
Sorted list (Linked-List)		
BST		
AVL tree		
Hash Table		

Binary Heap Properties

1. Structure Property
2. Ordering Property

Brief interlude: Some Definitions:

A **Perfect** binary tree – A binary tree with all leaf nodes at the same depth. All internal nodes have 2 children.



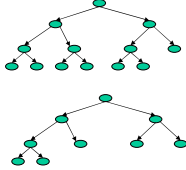
Full Binary Tree

- A binary tree in which each node has exactly zero or two children.
- (also known as a proper binary tree)
- (we will use this later for Huffman trees)

Heap Structure Property

- A binary heap is a **complete** binary tree.
- Complete binary tree** – binary tree that is completely filled, with the possible exception of the bottom level, which is filled left to right.

Examples:



3/28/2004

CS 216, Spring 2004

7

Complete binary tree of height h

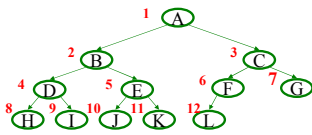
- For $h = 0$, just a single node.
- For $h = 1$, left child or two children.
- For $h \geq 2$, **either**
 - the left subtree of the root is full with height $h-1$ and the right is complete with height $h-1$, **OR**
 - the left is complete with height $h-1$ and the right is full with height $h-2$.

3/28/2004

CS 216, Spring 2004

8

Representing Complete Binary Trees in an Array



From node i :

left child:
right child:
parent:

implicit (array) implementation:

	A	B	C	D	E	F	G	H	I	J	K	L	
0	1	2	3	4	5	6	7	8	9	10	11	12	13

3/28/2004

CS 216, Spring 2004

9

Why better than pointers?

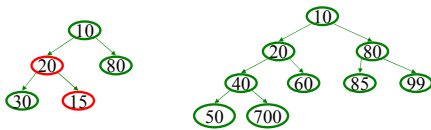
3/28/2004

CS 216, Spring 2004

10

Heap Order Property

- Heap order property:** For every non-root node X , the key in the parent of X is less than (or equal to) the key in X .



not a heap

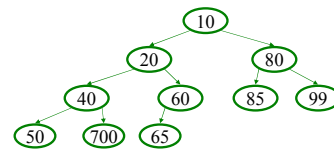
3/28/2004

CS 216, Spring 2004

11

Heap Operations

- findMin:
- insert(val): percolate up.
- deleteMin: percolate down.



3/28/2004

CS 216, Spring 2004

12

Heap – Insert(val)

Basic Idea:

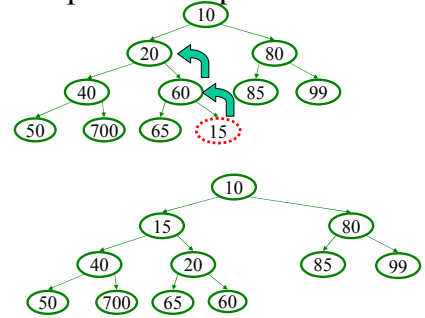
1. Put val at “next” leaf position
2. Repeatedly exchange node with its parent if needed

3/28/2004

CS 216, Spring 2004

13

Insert: percolate up



3/28/2004

CS 216, Spring 2004

14

Heap – Deletemin

Basic Idea:

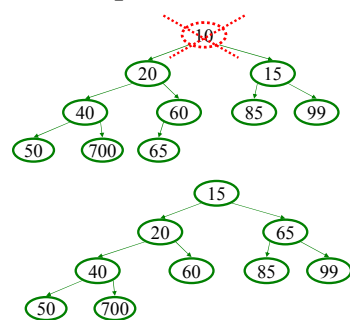
1. Remove root (that is always the min!)
2. Put “last” leaf node at root
3. Find smallest child
4. Swap node with smallest child if needed.
5. Repeat steps 3 & 4 until no swaps needed.

3/28/2004

CS 216, Spring 2004

15

DeleteMin: percolate down

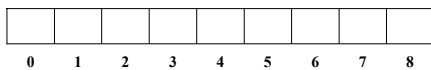


3/28/2004

CS 216, Spring 2004

16

Insert: 16, 32, 4, 69, 105, 43, 2



3/28/2004

CS 216, Spring 2004

17

Other Heap Operations

decreaseKey(process, amount): raise the priority of a process, percolate up

increaseKey(processID, amount): lower the priority of a process, percolate down

remove(processID): remove a process, move to top, then delete.
 1) decreaseKey(processID, ∞)
 2) deleteMin()

Worst case Running time for all of these: $O(\log N)$

FindMax?

ExpandHeap – when heap fills, copy into new space.

3/28/2004

CS 216, Spring 2004

18

Heaps (summary)

- insert: percolate up. $O(\log N)$ time.
- deleteMin: percolate down. $O(\log N)$ time.

- Heapsort?