csc263/W18

cs-toronto.edu

~sam/teaching/263

CSC263 Jan 10, 2018

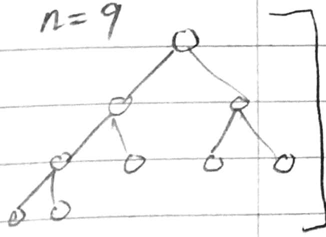
- abstract data type (ADT): object & its operations
- data structure: specific implementation of some ADT

- Example of an ADT is a Priority Queue
  - object: maintains a set S of elements with keys that can be compared
  - operations: insert(S, X); $S \leftarrow S \cup \{x\}$
    - max(S); return an element in S with max priority
    - extract-max(S); find element $x \leftarrow$ Max(S) and remove it from set S
  - PQ might be used by an operating system to organize processes that need to be run

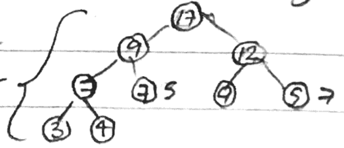
- Some data structures for ADT Priority Queue:

DS.	WC Insert	Extract Max
unsorted linked list	O(1)	O(n)
sorted linked list	O(n)	O(1)
heaps	O(log n)	O(log n)

- heaps: store n-element set S into an n-node CBT



  - complete binary tree: start at the leftmost node, every node has at most two children, fill each level before moving to the next
  - height of binary tree T: length of longest path from root of T to any leaf of T; for CBT of n nodes, height is $\lfloor \log_2 n \rfloor$
  - elements in heap stored s.t. priority of node > priority of children
  - heaps for set S are not unique
  - Ex S={3 4 5 7 7 9 9 12 17}



- how do you represent a heap in a computer?

1	2	3	4	5	6	7	8	9
17	9	12	7	7	9	5	3	4

use array to store CBT elements top-to-bottom, left-to-right

1-indexed!
- right child index = 2(parent index)
- left child index = 2(parent index)+1
- parent index = $\lfloor$ child index/2 $\rfloor$

- OUR HEAP A[ ] = HEAPSIZE=9

17	9	12	7	7	9	5	3	9

- INSERT(13): add element to next space; to SIZE = 10

							3	4	13

  get parent index, compare values, swap if parent smaller;
  par_ind = 5, par = 7, 13 > 7, swap; par_ind = 2, par = 9, 13 > 9, swap; etc.

  Worst case time of insert: compare & swap in constant time, height is max number of swaps ∴ $\Theta(\log n)$

  - $O(\log n)$: bcuz every insert takes at most $c \cdot \log(n)$ for constant c
  - $\Omega(\log n)$: bcuz ∃ insert that takes at least $c \log(n)$ steps (ie $x >$ root)
- MAX(A[ ]): $\theta(1)$; get first element of array
- Extract MAX(A[ ]): swap first & last (can return last as max), shrink heap size by 1, compare parent to both children, swap parent with larger of two children, continue until max order property holds or next child index greater than size of heap (no children)
  = $\theta \log n)$
- SORTING: extract max until all elements extracted, $\theta(n \log n$