Comp 352

Tutorial Session 7

Priority Queues
Quick Overview (1)

- A ***Priority Queue*** is an ADT that supports:
  - Arbitrary Insertion of elements
  - Removal of elements in order of priority

- An entry is a (key, value) pair

- A Comparator is an object that compares 2 keys

Priority Queues
Quick Overview (2)

- A ***Priority Queue P*** supports the following methods:
  - size(): Return the number of entries in *P*.
  - isEmpty(): Test whether *P* is empty.
  - min(): Return (but do not remove) an entry of *P* with smallest key; an error condition occurs if *P* is empty.
  - insert(k,x): Insert into *P* key *k* with value *x* and return the entry storing them; an error condition occurs if *k* is invalid (that is, *k* cannot be compared with other keys.
  - removeMin(): Remove from *P* and return an entry with smallest key; an error condition occurs if *P* is empty.

- Implementation via unsorted vs. sorted list (pp. 344,345)
- Selection vs. Insertion Sort (p. 348)

Heap
Quick Overview (1)

- To overcome the O(n) worst case running time of insertions (using sorted list) and removals (using unsorted lists), entries of a Priority Queue are stored in a binary tree instead of a list.

- The Heap data structure allows us to perform both insertions and removals in logarithmic time.

- ***Heap-Order Property:*** In a heap T, for every node v other than the root, the key stored at v is greater than or equal to the key stored at v's parent.

HEAP
QUICK OVERVIEW (2)

- For efficiency reasons, a heap should have as small a height as possible. A heap T should additionally satisfy a structural property: it must be ***complete***.

- ***Complete Binary Tree Property:*** A heap T with height h is a ***complete*** binary tree if levels 0,1,2,… ,h − 1 of T have the maximum number of nodes possible and in level $h - 1$, all the internal nodes are to the left of the external nodes and there is at most one node with one child, which must be a left child.

- Another important node in a heap T, other than the root, is the ***last node*** of T, which is the right-most, deepest external node of T.

- *A heap T storing n entries has height $h = \lfloor \log n \rfloor$.*

Question 1

Illustrate the execution of a Bottom-Up construction of a heap on the following sequence:

(2,5,16,4,10,23,39,18,26,15,7,9,30,31,40)

Question 2

Illustrate the execution of the heap-sort algorithm on the following sequence:

(2,5,16,4,10,23,39,18,26,15).

Show the contents of both the heap and the sequence at each step of the algorithm.

QUESTION 3

Give an algorithm for changing the value of an arbitrary element from a heap of size N.

Determine worst-case time complexity of your algorithm.

You may describe your algorithm in English.

Question 4

At which nodes of a heap can an entry with the largest key be stored?

Question 5

Bill claims that a preorder traversal of a heap will list its keys in nondecreasing order. Draw an example of a heap that proves him wrong.

QUESTION 6

Let T be a complete binary tree such that node v stores the entry $(p(v), 0)$, where $p(v)$ is the level number of v. Is tree T a heap?

Why or why not?

Question 7

Explain why the case where node *r* has a right child but not a left child was not considered in the description of down-heap bubbling.

QUESTION 8

Is there a heap T storing seven entries with distinct keys such that a pre order traversal of T yields the entries of T in increasing or decreasing order by key?

How about an inorder traversal?

How about a postorder traversal?

If so, give an example; if not, say why.