Lecture 8: Priority Queue
CSC2100 Data Structure

Yufei Tao
CSE department, CUHK

March 12, 2011
In this lecture, we will learn a structure called the priority queue, which is also known as the heap. It is an implementation of the binary tree, which is an important general structure that also underlies many other concrete structures (some of which will be discussed later in this course).
1 Problem

2 Binary tree

3 Priority queue
   - Definition
   - Insertion
   - Delete-min

4 Time analysis
Problem definition

Problem (Heap)

Let $S$ be a set of numbers. We want to support an arbitrary sequence of the following operations:

1. **Insert**: Add a new number into $S$.
2. **Delete-min**: Remove from $S$ the smallest number.

Our goal is to perform every operation (of each type) in $O(\log n)$ time, where $n$ is the size of $S$. 
Try to solve the heap problem using arrays and linked lists. What obstacles would you run into?
A binary tree is a hierarchy of nodes, where every parent node has at most two child nodes. There is a unique node, called the root, that does not have a parent.

Node $u_1$ is the parent of $u_2$ and $u_3$, which, therefore, are the child nodes of $u_1$. Node $u_1$ is the root.
In a binary tree, a node is a **leaf**, if it does not have any child. Otherwise, it is a **non-leaf** node.

**Example**

\[u_7, u_8, u_9, u_6\] are leaf nodes, while the other nodes are non-leaf nodes.
The **level** of a node $u$ is the number of edges that need to be traversed from the root to $u$ (as a corollary, the root has level 0). The **height** of the tree is the maximum level of all nodes.

**Example**

Node $u_4$ is at level 2, while $u_9$ at level 3. The tree has height 3.
A **complete binary tree** is a binary tree where

- every level is completely filled, except possibly the lowest one;
- at the lowest level, all nodes are as far left as possible.

**Example**

Only the first tree is a complete binary tree.

Note that if the $i$-th ($i > 0$) level is filled, then there are $2^i$ nodes at this level.
A priority queue on a set $S$ of numbers is a complete binary tree where

- each node stores a distinct number in $S$, which is called the node’s key;
- the key of a node is always greater than that of its parent.

Note that the first condition implies that the priority queue has $n = |S|$ nodes. Since each node occupies $O(1)$ space, the overall space cost of a priority queue is $O(n)$. 
There can be multiple priority queues for a set $S$ of numbers. Assume, for example, $S = \{1, 8, 23, 26, 39, 54, 79, 93\}$. The following are both legal priority queues:
Inserting a number

Let us first see an example.

Denote by $H$ the above priority queue. Assume that we want to insert a number 15.
Example (cont.)

First, put 15 as the right most node at the bottom level of \( H \) (so that the resulting \( H \) is still a complete binary tree).

Note that 15 causes a violation because it is smaller than its parent. This is fixed by swapping it with its parent, as shown in the next slide.
Inserting a number (cont.)

Example (cont.)

15 still causes a violation, necessitating another swap, as shown next.
Inserting a number (cont.)

Example (cont.)

No more violation. This is the final $H$ after the insertion.
**Insertion algorithm**

```
algorithm Insert(v)
/* insert a number v into the priority queue H */
1. create a node u such that u.key = v
2. place u as the right most node at the bottom level of H
   /* we will discuss in the tutorial how to do this efficiently */
3. while u has a parent p
4. if u.key > p.key
5. return /* no violation */
6. swap the keys of u and p
7. u = p
```
Again, let us get the idea of the algorithm from an example.

Assume that we want to maintain $H$ to be a legal priority queue after removing the root 1 (which is returned as the minimum).
Delete-min (cont.)

Example (cont.)

First, find the right most node at the bottom level of $H$, namely, node 79.

![Binary tree diagram]

Note that $H$ is still a complete binary tree after removing this node.
Delete-min (cont.)

Example (cont.)

Remove node 79, but place the value 79 in the root.

Note that node 79 causes a violation because it is greater than its children. This is fixed by swapping it with node 8, which is the child of node 79 with a smaller key. See the next slide.
Node 79 still has a violation, causing another swap as shown next.
Delete-min (cont.)

Example (cont.)

The final $H$ after the delete-min.
algorithm $\text{Delete-min}(v)$

/* remove the smallest key in $H$ */

1. if $H$ is empty, then return $\emptyset$
2. $m =$ the value in the root of $H$
3. $u =$ the root of $H$
4. $u' =$ the right most node at the bottom level of $H$
   /* we will discuss in the tutorial how to find $u'$ efficiently */
5. set the key of $u$ to that of $u'$, and remove $u'$ from $H$
6. while $u$ has a child
7. if $u.key$ is smaller than the keys of its child nodes
8. return /* no violation */
9. $c =$ the child of $u$ having a smaller key (if $u$ has only one child, then $c$ is that child)
10. swap the keys of $u$ and $c$
11. $u = c$
12. return $m$
A general result

Lemma

A complete binary tree with \( n \) nodes has height \( O(\log n) \).

Proof.

Assume that the tree has height \( h \). In other words, levels 0, 1, ..., \( h-1 \) are full, namely, for each \( 0 \leq i \leq h-1 \), the \( i \)-th level has \( 2^i \) nodes. Hence:

\[
n \geq \sum_{i=0}^{h-1} 2^i = 2^h - 1
\]

Hence, \( h \leq \log_2(n + 1) = O(\log n) \). \( \square \)
A priority queue with $n$ nodes has height $O(\log n)$.

We can decide where to place the new node in $O(\log n)$ time (tutorial).

Fixing violation requires traversing at most one leaf-to-root path, which also takes $O(\log n)$ time.

Therefore, the total insertion cost is $O(\log n)$. 
The right most node at the bottom level can be found in $O(\log n)$ time (tutorial).

Fixing violation then requires traversing at most one root-to-leaf path, which takes $O(\log n)$ time.

Therefore, the total delete-min time is $O(\log n)$. 
Remark 1

The priority queue described earlier is also called a *min-heap*, because it is designed to remove the minimum of a set $S$ efficiently.

Likewise, we can also easily design an alternative priority queue to remove the maximum of $S$ efficiently. That priority queue is thus called a *max-heap*. 
Using a min-heap, we can design another optimal algorithm to sort a set $S$ of $n$ numbers in ascending order. The algorithm involves 2 simple steps:

1. Insert the $n$ numbers in $S$ into a min-heap $H$.
2. Perform delete-min $n$ times, and output the numbers in the order they are removed from the heap.

Running time $= O(n \log n)$. The above algorithm is called heap sort.
Playback of this lecture:

- Priority queue (a.k.a. heap).
- $O(n)$ space, where $n$ is the number of nodes in the priority queue.
- $O(\log n)$ insertion time.
- $O(\log n)$ delete-min time.