lecture20: Priority Queue ADT

Sean Massung

Largely based on slides by Cinda Heeren
CS 225 UIUC

15th July, 2013
Announcements

- mp5 due tonight
- lab_avl tomorrow, due Thursday night (7/18)
- mp6 out tonight, mp6.1 extra credit due Friday night (7/19)
The wealthy owner of an amusement park has many connections. His acquaintances (both close and distant) use this to their advantage when they are waiting in line there.

Each patron at the amusement park has some level of familiarity with the rich owner, measured as $f \in [0, 1]$. When new spots are available on the rides, those waiting in line with the highest familiarity get to go on first.

The software used to implement this queueing system makes use of a new ADT: the *PriorityQueue*
The amusement park’s queueing system

```java
PriorityQueue<Person> line;

Person a("Bob", .34);          // familiarity = .34
Person b("Sally", .76);        // familiarity = .76
Person c("Fred", .66);         // familiarity = .66

line.push(a);
line.push(b);
line.push(c);

line.pop();                    // returns Sally

Person d("Jim", .50);          // familiarity = .50
line.push(d);

line.pop();                    // returns Fred
line.pop();                    // returns Jim
line.pop();                    // returns Bob
```
The PriorityQueue ADT

```cpp
/** @file priority_queue.h */

template <class T>
class PriorityQueue
{
  public:
    PriorityQueue(); // + big 3
    void push(const T & elem);
    T peek() const;
    T pop();
    size_t size() const;
    bool empty() const;

  private:
    // ???
};
```

Let’s assume we can compare type T objects with `operator<`. That is, if T == Person, `operator<` would work with respect to each person’s familiarity.
How would you implement a PriorityQueue?

Implementation options:

1. Unsorted or sorted array?
2. Unsorted or sorted linked list?
3. Balanced or unbalanced BST?

Remember, we need to support these PriorityQueue ADT functions:

1. push (any element)
2. pop (the highest priority element)
3. peek (show the next element that will be popped)
## PQ: A sorted array

<table>
<thead>
<tr>
<th>Operation</th>
<th>How?</th>
<th>Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>push</td>
<td>search + insert</td>
<td></td>
</tr>
<tr>
<td>pop</td>
<td>search + remove</td>
<td></td>
</tr>
<tr>
<td>peek</td>
<td>search</td>
<td></td>
</tr>
</tbody>
</table>
PQ: A sorted LL

<table>
<thead>
<tr>
<th>Operation</th>
<th>How?</th>
<th>Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>push</td>
<td>search + insert</td>
<td></td>
</tr>
<tr>
<td>pop</td>
<td>search + remove</td>
<td></td>
</tr>
<tr>
<td>peek</td>
<td>search</td>
<td></td>
</tr>
</tbody>
</table>
## PQ: A balanced BST

<table>
<thead>
<tr>
<th>Operation</th>
<th>How?</th>
<th>Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>push</td>
<td>search + insert</td>
<td></td>
</tr>
<tr>
<td>pop</td>
<td>search + remove</td>
<td></td>
</tr>
<tr>
<td>peek</td>
<td>search</td>
<td></td>
</tr>
</tbody>
</table>
PQ: A binary heap

<table>
<thead>
<tr>
<th>Operation</th>
<th>How?</th>
<th>Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>push</td>
<td>?</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>pop</td>
<td>?</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>peek</td>
<td>?</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>
Binary Heaps

- What kind of tree structure is this?
- Can you describe any properties?
- Imagine this is a max-heap; that is, higher numbers have a higher priority
Binary Heaps

Recursive definition: a heap $T$ is a complete binary tree and

- $T = \{\}$ is a heap
- $T = \{r\}$ is a heap
- $T = \{r, T_{left}, T_{right}\}$ and
  - $T_{left, right}$ are not both empty
  - $\text{key}(r) > \text{key}(T_{left})$
  - $\text{key}(r) > \text{key}(T_{right})$
  - $T_{left, right}$ are both heaps
A heap is usually stored as an array, *not* as a pointer-based tree! We can do this efficiently because we know the heap is a complete tree. (Why is this important?)
Heap Implementation

Given the index of a subtree, how can you calculate the indices of its children and parent?
Sometimes it makes the calculations easier to have the heap start at index 1:
Could you write peek? What is the running time?
Let's implement a PriorityQueue with a heap, and implement the heap with a Vector (which is a dynamically resizing array).

Let's assume in our heap class that the function `root()` returns 0 or 1 depending on how we set up our Vector to hold our elements.

In that case, `peek` would simply be

```cpp
template <class T>
T Heap<T>::peek() const
{
}
```
Could you write pop? What is the running time?
template <class T>
T Heap<T>::pop()
{

}
HeapifyDown

- While we are not at a leaf...
  - Swap the current value with the value of the higher priority child
  - call HeapifyDown on the child index you swapped with

This sifts elements down into the heap until the heap property is restored. You will write this function in lab_heaps.
Could you write push? What is the running time?
Writing push

```cpp
template <class T>
void Heap<T>::push(const T & elem)
{
    // Implementation...
}
```
HeapifyUp

- While we are not at the root...
  - Swap the current value with the value of the parent if your priority is higher
  - call HeapifyUp on the parent index if you swapped with it

This sifts elements up into the heap until the heap property is restored. You will write this function in lab_heaps.