Lecture 15: Priority Queues
Collections

Data types that store other objects (elements)

stacks, queues, trees, hash tables, etc.
Collections

Data types that store other objects (elements)

stacks, queues, trees, hash tables, etc.

C++

C++ STL Containers

http://en.cppreference.com/w/cpp/container
Collections

Data types that store other objects (elements)
stacks, queues, trees, hash tables, etc.

C++

C++ STL Containers
http://en.cppreference.com/w/cpp/container

Java

Java Collections Framework
https://docs.oracle.com/javase/8/docs/technote/technote/guides/collections/reference.html
## Container class templates

### Sequence containers:

- **array**
  - Array class (class template)
- **vector**
  - Vector (class template)
- **deque**
  - Double ended queue (class template)
- **forward_list**
  - Forward list (class template)
- **list**
  - List (class template)

### Container adaptors:

- **stack**
  - LIFO stack (class template)
- **queue**
  - FIFO queue (class template)
- **priority_queue**
  - Priority queue (class template)

### Associative containers:

- **set**
  - Set (class template)
- **multiset**
  - Multiple-key set (class template)
- **map**
  - Map (class template)
- **multimap**
  - Multiple-key map (class template)

### Unordered associative containers:

- **unordered_set**
  - Unordered Set (class template)
- **unordered_multiset**
  - Unordered Multiset (class template)
- **unordered_map**
  - Unordered Map (class template)
- **unordered_multimap**
  - Unordered Multimap (class template)
### Container class templates

#### Sequence containers:
- array
- vector
- deque
- forward_list
- list

#### Container adaptors:
- stack
- queue
- priority_queue

#### Associative containers:
- set
- multiset
- map
- multimap

#### Unordered associative containers:
- unordered_set
- unordered_multiset
- unordered_map
- unordered_multimap
## Container class templates

### Sequence containers:
- **array**: Array class (class template)
- **vector**: Vector (class template)
- **deque**: Double ended queue (class template)
- **forward_list**: Forward list (class template)
- **list**: List (class template)

### Container adaptors:
- **stack**: LIFO stack (class template)
- **queue**: FIFO queue (class template)
- **priority_queue**: Priority queue (class template)

### Associative containers:
- **set**: Set (class template)
- **multiset**: Multiple-key set (class template)
- **map**: Map (class template)
- **multimap**: Multiple-key map (class template)

### Unordered associative containers:
- **unordered_set**: Unordered Set (class template)
- **unordered_multiset**: Unordered Multiset (class template)
- **unordered_map**: Unordered Map (class template)
- **unordered_multimaps**: Unordered Multimap (class template)

C++  
STL  
hash tables  
balanced BSTs
### Container class templates

#### Sequence containers:
- **array**
  - Array class (class template)
- **vector**
  - Vector (class template)
- **deque**
  - Double ended queue (class template)
- **forward_list**
  - Forward list (class template)
- **list**
  - List (class template)

#### Container adaptors:
- **stack**
  - LIFO stack (class template)
- **queue**
  - FIFO queue (class template)
- **priority_queue**
  - Priority queue (class template)

#### Associative containers:
- **set**
  - Set (class template)
- **multiset**
  - Multiple-key set (class template)
- **map**
  - Map (class template)
- **multimap**
  - Multiple-key map (class template)

#### Unordered associative containers:
- **unordered_set**
  - Unordered Set (class template)
- **unordered_multiset**
  - Unordered Multiset (class template)
- **unordered_map**
  - Unordered Map (class template)
- **unordered_multimap**
  - Unordered Multimap (class template)
Priority Queues

Queues

basic operations: enqueue, dequeue
always remove the item least recently added
Priority Queues

Queues
basic operations: enqueue, dequeue
always remove the item least recently added

Priority Queues
basic operations: insert, deleteMax
always remove the item with highest (max) priority
Priority Queues

Insert

deleteMax

77 21
10 9
4 35
87
13
10 42
94
## Cost of Priority Queues

<table>
<thead>
<tr>
<th></th>
<th>Insert</th>
<th>deleteMax</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sorted Array</strong></td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>Unsorted Array</strong></td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td><strong>AVL</strong></td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
</tbody>
</table>
Binary Heap

Structure Property

a binary heap is a complete binary tree
Binary Heap

Structure Property

a binary heap is a complete binary tree

Heap-Order Property

for every node $X$, $\text{key}(\text{parent}(X)) \geq \text{key}(X)$
except the root, which has no parent
Complete Binary Tree

Each level is completely filled
possible exception of the bottom level
Complete Binary Tree

Each level is completely filled
possible exception of the bottom level

A complete binary tree has:
- at least $2^h$ nodes
- at most $2^{h+1} - 1$ nodes
Binary Heap

Heap-Order Property

Structure Property
Binary Heap

Heap-Order Property

Structure Property
Binary Heap

Heap-Order Property

Structure Property

👍

👍

👍

👍
50
30
25
 3
 7
15
 8
18
 9
22
 4
 5
45
 2
45
 5
 6
31
 7
 11
 12
 0
1
2
3
4
5
6
7
8
9
10
11
12

parent(i)
floor((i-1)/2)
left_child(i)
i*2 + 1
right_child(i)
parent(i)
floor((i-1)/2)
left_child(i)
i*2 + 1
right_child(i)
i*2 + 2
Insert

Append new element to the end of array
Insert

Append new element to the end of array

Check heap-order property

if violated, **bubble-up** (swap with parent)

**repeat** until heap-order is restored

if not, we are done
insert 47
insert 47
insert 47
insert 42
deleteMax

Max element is the the **first** element of the array

the root of the heap
deleteMax

Max element is the the **first** element of the array
the root of the heap

Copy last element of array to first position
then decrement array’s size by 1
deleteMax

Max element is the first element of the array
the root of the heap

Copy last element of array to first position
then decrement array’s size by 1

Check heap-order property
if violated, bubble-down (swap with larger child)
repeat until heap-order is restored
if not, we are done
deleteMax
deleteMax
deleteMax
Analysis?

Insert?
Analysis?

Insert?  \( O(\log n) \)
Analysis?

Insert? O(log n)

deleteMax?
Analysis?

Insert? \( O(\log n) \)

deleteMax? \( O(\log n) \)
Analysis?

Insert? \( O(\log n) \)

deleteMax? \( O(\log n) \)

tree is always balanced!
Other Heap Operations

decreaseKey
  lowers the value (priority) of a given key
  need to bubble-down
Other Heap Operations

decreaseKey

- lowers the value (priority) of a given key
- need to bubble-down

\( O(\log n) \)
Other Heap Operations

decreaseKey
- lowers the value (priority) of a given key
- need to bubble-down

increaseKey
- increases the value (priority) of a given key
- need to bubble-up

$O(\log n)$
Other Heap Operations

**decreaseKey**
- lowers the value (priority) of a given key
- need to bubble-down

**increaseKey**
- increases the value (priority) of a given key
- need to bubble-up

\[O(\log n)\]
Other Heap Operations

decreaseKey
- lowers the value (priority) of a given key
- need to bubble-down

increaseKey
- increases the value (priority) of a given key
- need to bubble-up

removeKey
- removes a given key
- increaseKey($+\infty$) then deleteMax

$O(\log n)$
Other Heap Operations

decreaseKey
  lowers the value (priority) of a given key
  need to bubble-down
  $O(\log n)$

increaseKey
  increases the value (priority) of a given key
  need to bubble-up
  $O(\log n)$

removeKey
  removes a given key
  $O(\log n)$
  increaseKey($+\infty$) then deleteMax