Heap
4th Recitation for CS1501

Huichao Xue

Department of Computer Science
Universities of Pittsburgh

Algorithm Implementation, Spring, 2009
Outline

1 Concepts
   - Priority Queue
   - Operations on Priority Queues
   - Heap

2 Algorithms
   - Initializing
   - Storage
   - Inserting
   - Removing
Outline

1 Concepts
   - Priority Queue
   - Operations on Priority Queues
   - Heap

2 Algorithms
   - Initializing
   - Storage
   - Inserting
   - Removing
Definition

A structure in which elements are processed in order from the largest (or from smallest) but not necessarily all at once.

This means we don’t have to sort the elements up front – so we may be able to do better than $O(N \log N)$.
Definition

A structure in which elements are processed in order from the largest (or from smallest) but not necessarily all at once.

This means we don’t have to sort the elements up front – so we may be able to do better than $O(N \log N)$.
Outline

1. Concepts
   - Priority Queue
   - Operations on Priority Queues
   - Heap

2. Algorithms
   - Initializing
   - Storage
   - Inserting
   - Removing
Operations

- *construct* a priority queue from $N$ items
- *insert* a new item into the priority queue
- *remove* the largest item from the priority queue
- *change* the priority of an item

Read chapter 11 in the text. We will implement a priority queue as a heap data structure.
Operations

- **construct** a priority queue from $N$ items
- **insert** a new item into the priority queue
- **remove** the largest item from the priority queue
- **change** the priority of an item

Read chapter 11 in the text. We will implement a priority queue as a heap data structure.
Outline

1. Concepts
   - Priority Queue
   - Operations on Priority Queues
   - Heap

2. Algorithms
   - Initializing
   - Storage
   - Inserting
   - Removing
Definition

A full binary tree is one that is filled in at every level from the root to the leaf-level.
A full binary tree is one that is filled in at every level from the root to the leaf-level.
Full binary tree

Definition

A full binary tree is one that is filled in at every level from the root to the leaf-level.

This tree is full!
Complete binary tree

**Definition**

A complete binary tree is full at every level except possibly the last but the nodes at the bottom-level are filled in with no gaps from left to right.
A complete binary tree is full at every level except possibly the last but the nodes at the bottom-level are filled in with no gaps from left to right.
Complete binary tree

Definition

A complete binary tree is full at every level except possibly the last but the nodes at the bottom-level are filled in with no gaps from left to right.

This tree is not a complete binary tree.
Example

[Diagram of a heap structure]

Huichao Xue
Heap
Example

This tree is a complete binary tree.
A heap is a complete binary tree in which each node has priority greater or equal to each of its children.

Assign priority values to the nodes so that every node has priority greater or equal to its children, and we get a heap (Max Heap).
Definition

A heap is a complete binary tree in which each node has priority greater or equal to each of its children. Assign priority values to the nodes so that every node has priority greater or equal to its children, and we get a heap (Max Heap).
Outline

1. Concepts
   - Priority Queue
   - Operations on Priority Queues
   - Heap

2. Algorithms
   - Initializing
   - Storage
   - Inserting
   - Removing
Initializing a heap

```cpp
PQ::PQ(int max)
{
    a = new int[max];  //store for the priorities
    N = 0;            //number of priorities
}
```
Outline

1. Concepts
   - Priority Queue
   - Operations on Priority Queues
   - Heap

2. Algorithms
   - Initializing
   - Storage
   - Inserting
   - Removing
An example

Here is how it is stored:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>59</td>
<td>45</td>
<td>32</td>
<td>17</td>
<td>37</td>
<td>25</td>
<td>2</td>
<td>15</td>
<td>12</td>
<td>20</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Here is how it is stored:

<p>| | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>59</td>
<td>45</td>
<td>32</td>
<td>17</td>
<td>37</td>
<td>25</td>
<td>2</td>
<td>15</td>
<td>12</td>
<td>20</td>
<td></td>
</tr>
</tbody>
</table>
Outline

1. **Concepts**
   - Priority Queue
   - Operations on Priority Queues
   - Heap

2. **Algorithms**
   - Initializing
   - Storage
   - Inserting
   - Removing
```cpp
void PQ::upheap(int v)
{
    a[++N] = v;
    place = N;
    parent = place/2;

    while(parent>0 && (a[place] > a[parent])) {
        swap(a[place], a[parent]);
        place = parent;
        parent = place/2;
    }
}
```
An Example

Let’s add $v = 48$ to the heap above. A call to `upheap(48)` is made:

1. $N$ is incremented to 11.
2. $a[11] = 48$, so the array now contains:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>59</td>
<td>45</td>
<td>32</td>
<td>17</td>
<td>37</td>
<td>25</td>
<td>2</td>
<td>15</td>
<td>12</td>
<td>20</td>
<td>48</td>
</tr>
</tbody>
</table>
Let’s add $v = 48$ to the heap above.

A call to upheap(48) is made:

1. $N$ is incremented to 11.
2. $a[11] = 48$, so the array now contains:

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>59</td>
<td>45</td>
<td>32</td>
<td>17</td>
<td>37</td>
<td>25</td>
<td>2</td>
<td>15</td>
<td>12</td>
<td>20</td>
<td>48</td>
</tr>
</tbody>
</table>
An Example

Let’s add $v = 48$ to the heap above. A call to `upheap(48)` is made:

1. $N$ is incremented to 11.
2. $a[11] = 48$, so the array now contains:

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>59</td>
<td>45</td>
<td>32</td>
<td>17</td>
<td>37</td>
<td>25</td>
<td>2</td>
<td>15</td>
<td>12</td>
<td>20</td>
<td>48</td>
</tr>
</tbody>
</table>
An Example (cont.)
An Example (cont.)

Heap
An Example (cont.)

A heap data structure is represented by the following tree:

59

48

17

45

15 12 20 37

25

2

Heap
Outline

1 Concepts
   - Priority Queue
   - Operations on Priority Queues
   - Heap

2 Algorithms
   - Initializing
   - Storage
   - Inserting
   - Removing
Remove the max item

```cpp
int PQ::remove()
{
    v = a[1];
    a[1] = a[N--];
    downHeap(1);
    return v;
}
```
void PQ::downHeap(int root)
{
    if (root is not a leaf)
    {
        //root must have a left child
        child = 2*root;

        if (root has a right child)
        {
            rightChild = child + 1;
            if (a[rightChild] > a[child])
            {
                child = rightChild;  //index of larger child
            }
        }

        if (a[root] < a[child])
        {
            swap(a[root], a[child]);
            downHeap(child);
        }
    }
}
Example

```
59
  48
  17
  15 12 20 37
   45
     25
    2
  32
```

Huichao Xue

Heap
Example
Example

Heap

Concepts
Algorithms

Initializing
Storage
Inserting
Removing

Huichao Xue
Example

Heap tree:

- 48
  - 37
    - 17
    - 45
    - 15
  - 32
    - 25
    - 2
Example

Concepts
Algorithms
Initializing Storage Inserting Removing

Heap

Huichao Xue

48

37

17

15

32

25

2

20

12

45

17

15
Example

Heap

Concepts
Algorithms
Initializing
Storage
Inserting
Removing

48
45
32
17
37
25
2
15
12
20