CSE 326 Data Structures

CSE 326 : Dave Bacon

Priority Queues : Binary Min Heap
Logistics

• AA 9:30 section moved to CSE 203
• No class Monday. Holiday!
• Homework 1
  – Hand in at front of class! I’ll pick up the folder about 10 minutes into class!
• Homework 2 – available soon on website
  – Due Fri, Jan 19 in class
• Reading
  – Chapter 6 : Priority Queues [Leftist heaps]
Priority Queue ADT

- Checkout line at the supermarket
- Printer queues
- operations: insert, deleteMin
Binary Heap Properties

1. Structure Property
2. Ordering Property
Heap **Structure** Property

- A binary heap is a **complete** binary tree.

**Complete binary tree** – binary tree that is completely filled, with the possible exception of the bottom level, which is filled left to right.

**Examples:**

```
          1
         / 
        2   3
       /     
      4       5
     /   6   /   7
    /     /     /     
   8   9  10  11  12  13
```

```
          1
         /  
        2    3
       /    /  
      4    5   6
     /  7   /  
    /   8   /   
   9   10  11  12
```
Representing Complete Binary Trees in an Array

From node i:
- left child:
- right child:
- parent:

Implicit (array) implementation:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>K</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<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>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Heap Order Property

Heap order property: For every non-root node $X$, the value in the parent of $X$ is less than (or equal to) the value in $X$.

not a heap
Heap Operations

- `findMin`: 
- `insert(val)`: percolate up.
- `deleteMin`: percolate down.
Heap – Insert(val)

Basic Idea:
1. Put val at “next” leaf position
2. Repeatedly exchange node with its parent if needed
Insert: percolate up

```
(10)
 /   \
|    |
(20)  (80)
 /     /    \
(60)  (85)  (99)
   / 
(40) (700) (65) (15)
 /   \
(50) (700) (65) (60)
```

```
(10)
 /   \
|    |
(15)  (80)
 /     /    \
(20)  (85)  (99)
   / 
(40) (20) (80)
   / 
(50) (40) (85)
   /   \
(700) (20) (99)
```
void insert(Object o) {
    assert(!isFull());
    size++;
    newPos =
        percolateUp(size, o);
    Heap[newPos] = o;
}

int percolateUp(int hole, Object val) {
    while (hole > 1 &&
            val < Heap[hole/2])
        Heap[hole] = Heap[hole/2];
    hole /= 2;
    return hole;
}

runtime:

(Java code in book)
Insert: 16, 32, 4, 69, 105, 43, 2

empty heap

```
(16)     
(32)     
(4)      
(4)      
(32)      
(32)      
(69)      
(105)     
(16)      
(43)      
```

```
(2) -->  (4)      
(32)      
(4)      
(32)      
(69)      
(105)     
(16)      
(43)      
```

del etc
Heap – Deletemin

Basic Idea:
1. Remove root (that is always the min!)
2. Put “last” leaf node at root
3. Find smallest child of node
4. Swap node with its smallest child if needed.
5. Repeat steps 3 & 4 until no swaps needed.
DeleteMin: percolate down
DeleteMin pseudo Code (Optimized)

Object deleteMin() {
    assert(!isEmpty());
    returnVal = Heap[1];
    size--;
    newPos =
        percolateDown(1,
                Heap[size + 1]);
    Heap[newPos] =
        Heap[size + 1];
    return returnVal;
}

int percolateDown(int hole,
        Object val) {
    while (2*hole <= size) {
        left = 2*hole;
        right = left + 1;
        if (right <= size &&
            Heap[right] < Heap[left])
            target = right;
        else
            target = left;

        if (Heap[target] < val) {
            Heap[hole] = Heap[target];
            hole = target;
        } else
            break;
    } return hole;
}
DeleteMin 2 times

```plaintext
[Diagram showing a binary tree with nodes 2, 32, 4, 69, 105, 43, 16]
```
Other Priority Queue Operations

- **decreaseKey**
  - given a pointer to an object in the queue, reduce its priority value

  Solution: change priority and

  

- **increaseKey**
  - given a pointer to an object in the queue, increase its priority value

  Why do we need a *pointer*? Why not simply data value?

  Solution: change priority and

  

More Priority Queue Operations

- **Remove**(objPtr)
  - given a pointer to an object in the queue, remove it

**Solution**: set priority to negative infinity, percolate up to root and deleteMin

Worst case Running time for all of these:
FindMax?
ExpandHeap – when heap fills, copy into new space.
More Priority Queue Operations

- **buildHeap**
  Naïve solution:

  Running time:

  Can we do better?
BuildHeap: Floyd’s Method

Add elements arbitrarily to form a complete tree. Pretend it’s a heap and fix the heap-order property!
private void buildHeap() {
    for (int i = currentSize/2; i > 0; i--)
        percolateDown(i);
}

runtime:
BuildHeap: Floyd’s Method
Finally…

runtime:
Binary Min Heaps (summary)

- **insert**: percolate up. $O(\log N)$ time.
- **deleteMin**: percolate down. $O(\log N)$ time.
- **Build**: Floyd’s method. $O(N)$ time.
- **Next time**: Even more priority queues??