CSE 332: Data Structures
Priority Queues – Binary Heaps
Richard Anderson, Steve Seitz
Winter 2014

Administrative

• P1A due tonight (Monday) by 11:59pm
  – via catalyst
• HW1 due beginning of class Wednesday
• Reading for this week: Chapter 6.1-6.5

Recall Queues

• FIFO: First-In, First-Out
  – Print jobs
  – File serving
  – Phone calls and operators
  – Lines at the Department of Licensing…

Priority Queues

Prioritize who goes first – a priority queue:
  – treat ER patients in order of severity
  – route network packets in order of urgency
  – operating system can favor jobs of shorter duration or those tagged as having higher importance
  – Greedy optimization: “best first” problem solving

Priority Queue ADT

• Need a new ADT
• Operations: Insert an Item, Remove the “Best” Item

1. PQueue data: collection of data with priority
2. PQueue operations
   – insert
   – deleteMin
     (also: create, destroy, is_empty)
3. PQueue property: if x has lower priority than y, x will be deleted before y
### Potential implementations

<table>
<thead>
<tr>
<th></th>
<th>insert</th>
<th>deleteMin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsorted list (Array)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unsorted list (Linked-List)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sorted list (Array)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sorted list (Linked-List)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Binary Search Tree (BST)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Binary Heap data structure

- **binary heap** (a kind of binary tree) for priority queues:
  - $O(\log n)$ worst case for both insert and deleteMin
  - $O(1)$ average insert

- It's optimized for priority queues. Lousy for other types of operations (e.g., searching, sorting)

### Binary Heap Properties

A binary heap is a binary tree with two important properties that make it a good choice for priority queues:

1. **Completeness**
2. **Heap Order**

**Note:** we will sometimes refer to a binary heap as simply a "heap".

### Completeness Property

- A binary heap is a **complete** binary tree:
  - a binary tree with all levels full, except possibly the bottom level, which is filled left to right

**Examples:**

Height of a complete binary tree with $n$ nodes?
**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.

Which of these is a heap?

---

**Heap Operations**

- Main ops: insert, deleteMin
- Key is to maintain
  - Completeness
  - Heap Order
- Basic idea is to propagate changes up/down the tree, fixing order as we go

---

**Heap – insert(val)**

Basic Idea:
1. Put val at last leaf position
2. Percolate up by repeatedly exchanging node with parent as long as needed

**Insert: percolate up**

---

**Heap – deleteMin**

Basic Idea:
1. Remove min element
2. Put “last” leaf node value 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: percolate down

Representing Complete Binary Trees in an Array

Why use an array?

DeleteMin Code

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;
}

runtime:

(Java code in book)

Insert Code

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])
        return hole;
    }

runtime:

(Java code in book)

Insert: 16, 32, 4, 69, 105, 43, 2

runtime:

(Java code in book)
More Priority Queue Operations

decreaseKey(nodePtr, amount):
given a pointer to a node in the queue, reduce its priority

Binary heap: change priority of node and ________________

increaseKey(nodePtr, amount):
given a pointer to a node in the queue, increase its priority

Binary heap: change priority of node and ________________

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

Worst case running times?

More Binary Heap Operations

expandHeap():
If heap has used up array, copy to new, larger array.
• Running time:

buildHeap(objList):  
Given list of objects with priorities, fill the heap.
• Running time:

We do better with buildHeap...

Building a Heap: Take 1

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

BuildHeap: Floyd’s Method

Add elements arbitrarily to form a complete tree.
Pretend it’s a heap and fix the heap-order property!

Red nodes need to percolate down

Key idea: fix red nodes from bottom-up
Finally…

```
private void buildHeap() {
    for (int i = currentSize/2; i > 0; i--)
        percolateDown(i);
}
```

Buildheap Analysis

- n/4 nodes percolate at most 1 level
- n/8 percolate at most 2 levels
- n/16 percolate at most 3 levels
- ...

runtime: