Priority Queues: 
Binary Min Heaps

CSE 373 
Data Structures and Algorithms

Today’s Outline

• Announcements
  – Assignment #2 due Fri, Oct 12 at the BEGINNING of lecture
  – Midterm #1, Fri, Oct 19.
  – Assignment #3 coming soon, due Thurs, Oct 25.

• Today’s Topics:
  – Dictionary
    • Balanced Binary Search Trees - (AVL Trees)
  – Priority Queues
    • Binary Min Heap

Priority Queue ADT

1. PQueue data: collection of data with priority

2. PQueue operations
   – insert
   – deleteMin
   (also: create, destroy, is_empty)

3. PQueue property: for two elements in the queue, x and y, if x has a lower priority value than y, x will be deleted before y

Applications of the Priority Q

• Select print jobs in order of decreasing length
• Forward packets on network routers in order of urgency
• Select most frequent symbols for compression
• Sort numbers, picking minimum first

• Anything greedy

Implementations of Priority Queue ADT

<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>

Representing Complete Binary Trees in an Array

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

implicit (array) implementation:
Why better than tree with pointers?

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.

![Heap order diagram]

Heap Operations

- **findMin:**
- **insert(val):** percolate up.
- **deleteMin:** percolate down.

![Heap operations diagram]

Heap – Insert(val)

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

![Heap insert diagram]

Insert pseudo Code (optimized)

```java
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)
Heap – Delegate

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 pseudo Code (Optimized)

```java
int PercolateDown(int hole, Object value) {
    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] < value) {
            Heap[hole] = Heap[target];
            hole = target;
        } else {
            break;
        }
    }
    return hole;
}
```

DeleteMin: percolate down

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

  Solution: change priority and __________________________

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

Other Heap Operations

- **decreaseKey**
  - objectPtr, amount): raise the priority of a object, percolate up

- **increaseKey**
  - objectPtr, amount): lower the priority of a object, percolate down

- **remove**
  - objectPtr): remove a object, move to top, then delete.

  1) decreaseKey(objectPtr, ∞)

  2) deleteMin()

Worst case Running time for all of these:

FindMax?

ExpandHeap – when heap fills, copy into new space.
Binary Min Heaps (summary)
- **insert**: percolate up. $\Theta(\log N)$ time.
- **deleteMin**: percolate down. $\Theta(\log N)$ time.
- **Build Heap?**

**BuildHeap: Floyd’s Method**

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

**Buildheap pseudocode**

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

**Finally…**