Priority Queues:
Binary Min Heaps

CSE 373
Data Structures and Algorithms

Today’s Outline

• Announcements
  – Assignment #3 coming soon, due Thurs, May 7th.

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

Priority Queue ADT

• Checkout line at the supermarket ???
• Printer queues ???
• operations: insert, deleteMin

1. PQ data : collection of data with priority
2. PQ operations
  – insert
  – deleteMin
    (also: create, destroy, is_empty)
3. PQ 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: 2i
- right child: 2i + 1
- parent: \( \left\lfloor \frac{i - 1}{2} \right\rfloor \)

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></td>
<td>0</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>
</tbody>
</table>

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 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 pseudo Code (optimized)

```java
void insert(Object o) {
    assert(!isFull());
    size++;
    int 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)
DeleteMin pseudo Code (Optimized)

```java
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: 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(objPtr, amount): raise the priority of a object, percolate up
increaseKey(objPtr, amount): lower the priority of a object, percolate down
remove(objPtr): remove a object, move to top, then delete.
   1) decreaseKey(objPtr, ∞)
   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. Θ(log N) time.
- deleteMin: percolate down. Θ(log N) time.
- Build Heap?

BuildHeap: Floyd’s Method

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

Buildheap pseudocode

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

Finally…

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

runtime:
Facts about Binary Min Heaps

Observations:
- finding a child/parent index is a multiply/divide by two
- operations jump widely through the heap
- each percolate step looks at only two new nodes
- inserts are at least as common as deleteMins

Realities:
- division/multiplication by powers of two are equally fast
- looking at only two new pieces of data: bad for cache!
- with huge data sets, disk accesses dominate