Priority Queues

(Today: Binary Min Heaps)

Chapter 6 in Weiss

CSE 326
Data Structures
Ruth Anderson
Winter 2010

Today’s Outline

• Announcements
  – Project #1, due 11pm Wed Jan 13.
  – Written Assignment #1 posted, due at the beginning of class Friday Jan 15.

• Today’s Topics:
  – Asymptotic Analysis
  – Priority Queues
    • Binary Min Heap

The One Page Cheat Sheet

• Calculating series:
  e.g. \( \sum_{j=1}^{n} j = \frac{n(n+1)}{2} \)
  1. Brute force (Section 1.2.3)
  2. Induction (Section 1.2.5)
  3. Memorize simple ones!

• General proofs (Section 1.2.5)
  e.g. How many edges in a tree with \( n \) nodes?
  1. Counterexample
  2. Induction
  3. Contradiction

Simplifying Recurrences

Given a recursive equation for the running time, can sometimes simplify it for analysis.

• For an upper-bound analysis, can optionally simplify to something larger, e.g.
  \( T(n) = T(\text{floor}(n/2)) + 1 \) to \( T(n) \leq T(n/2) + 1 \)

• For a lower-bound analysis, can optionally simplify to something smaller, e.g.
  \( T(n) = 2T(n/2 + 5) + 1 \) to \( T(n) \geq 2T(n/2) + 1 \)

Set Notation

“\( O(f(n)) \) is a set of functions”

\[ O(n^3) \]

So we say both
\( 100n^2 \log n = O(n^3) \)
and
\( 100n^2 \log n \in \( O(n^3) \) \)

Set notation allows us to formalize our intuition

\( O(n^3) \subset O(2^n) \)
Processor Scheduling

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>

Tree Review

- **root(T)**:
- **leaves(T)**:
- **children(B)**:
- **parent(H)**:
- **siblings(E)**:
- **ancestors(F)**:
- **descendents(G)**:
- **subtree(C)**:
**More Tree Terminology**

- **Depth** ($d(T)$): 
- **Height** ($h(G)$): 
- **Degree** ($d(B)$): 
- **Branching Factor** ($b(T)$): 

**Some More Tree Terminology**

- $T$ is **binary** if …
- $T$ is **$n$-ary** if …
- $T$ is **complete** if …

**Brief Interlude: Some Definitions:**

A **Perfect** binary tree – A binary tree with all leaf nodes at the same depth. All internal nodes have 2 children.

**Full** Binary Tree

- A binary tree in which each node has **exactly zero or two children**.
- (also known as a proper binary tree)
- (we will use this later for Huffman trees)

**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:**
Representing Complete Binary Trees in an Array

From node i:
left child: [4 5 6 7]
right child: [8 9 10 11 12 13]
parent: [1 2 3 4 5 6]

implicit (array) implementation:

<table>
<thead>
<tr>
<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>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++;
    newPos = percolateUp(size, o);
    Heap[newPos] = o;
}
```

runtime:

(Java code in book)
DeleteMin pseudo Code (Optimized)

```java
int percolateDown(int hole, Object val) {
    assert(!isEmpty());
    returnVal = Heap[1];
    size--;
    newPos = percolateDown(1, Heap[size+1]);
    Heap[newPos] = Heap[size + 1];
    return returnVal;
}
```

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

876543210

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

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 an object, percolate up
increaseKey(objPtr, amount): lower the priority of an object, percolate down
remove(objPtr): remove an object, move to top, then delete

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

BuildHeap pseudocode

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

Finally…