Priority Queues

CSE 373
Data Structures

Readings

• Reading
  › Goodrich and Tamassia, Chapter 7
Revisiting FindMin

- Application: Find the smallest (or highest priority) item quickly
  - Operating system needs to schedule jobs according to priority instead of FIFO
  - Event simulation (bank customers arriving and departing, ordered according to when the event happened)
  - Find student with highest grade, employee with highest salary etc.

Priority Queue ADT

- Priority Queue can efficiently do:
  - FindMin (and DeleteMin)
  - Insert
- What if we use...
  - Lists: If sorted, what is the run time for Insert and FindMin? Unsorted?
  - Binary Search Trees: What is the run time for Insert and FindMin?
  - Hash Tables: What is the run time for Insert and FindMin?
Less flexibility → More speed

- Lists
  - If sorted: FindMin is $O(1)$ but Insert is $O(N)$
  - If not sorted: Insert is $O(1)$ but FindMin is $O(N)$
- Balanced Binary Search Trees (BSTs)
  - Insert is $O(\log N)$ and FindMin is $O(\log N)$
- Hash Tables
  - Insert $O(1)$ but no hope for FindMin
- BSTs look good but...
  - BSTs are efficient for all Finds, not just FindMin
  - We only need FindMin

Better than a speeding BST

- We can do better than Balanced Binary Search Trees?
  - Very limited requirements: Insert, FindMin, DeleteMin. The goals are:
    - FindMin is $O(1)$
    - Insert is $O(\log N)$
    - DeleteMin is $O(\log N)$
Binary Heaps

- A binary heap is a binary tree (NOT a BST) that is:
  - Complete: the tree is completely filled except possibly the bottom level, which is filled from left to right
  - Satisfies the heap order property
    - every node is less than or equal to its children
    - or every node is greater than or equal to its children
- The root node is always the smallest node
  - or the largest, depending on the heap order

Heap order property

- A heap provides limited ordering information
- Each path is sorted, but the subtrees are not sorted relative to each other
  - A binary heap is NOT a binary search tree

These are all valid binary heaps (minimum)
Binary Heap vs Binary Search Tree

Binary Heap

5
10 94
97 24

Parent is less than both left and right children

Binary Search Tree

94
5 24
10 97

Parent is greater than left child, less than right child

Structure property

• A binary heap is a complete tree
  › All nodes are in use except for possibly the right end of the bottom row
Examples

- Complete tree, heap order is "max"
- Complete tree, heap order is "min"
- Not complete
- Complete tree, but min heap order is broken

Array Implementation of Heaps (Implicit Pointers)

- Root node = A[1]
- Keep track of current size N (number of nodes)

Value: - 2 4 6 7 5
Index: 0 1 2 3 4 5 6 7

N = 5
FindMin and DeleteMin

- **FindMin**: Easy!
  - Return root value $A[1]$
  - Run time = ?

- **DeleteMin**:
  - Delete (and return) value at root node

DeleteMin

- Delete (and return) value at root node
Maintain the Structure Property

- We now have a “Hole” at the root
  - Need to fill the hole with another value
- When we get done, the tree will have one less node and must still be complete

Maintain the Heap Property

- The last value has lost its node
  - We need to find a new place for it
- We can do a simple insertion sort operation to find the correct place for it in the tree
DeleteMin: Percolate Down

- Copy smaller child up and go down one level
- Done if both children are $\geq$ item or reached a leaf node
- What is the run time?

```
Percolate Down

PercDown(i: integer, x: integer): {
    // N is the number elements, i is the hole, x is the value to insert
    Case{
        2i > N : A[i] := x; //at bottom/
        2i = N : if A[2i] < x then
            else A[i] := x;
            else j := 2i+1;
            if A[j] < x then
                A[i] := A[j]; PercDown(j,x);
            else A[i] := x;
    }
}
```
DeleteMin: Run Time Analysis

- Run time is $O(\text{depth of heap})$
- A heap is a complete binary tree
- Depth of a complete binary tree of $N$ nodes?
  - $\text{depth} = \lceil \log_2(N) \rceil$
- Run time of DeleteMin is $O(\log N)$

Insert

- Add a value to the tree
- Structure and heap order properties must still be correct when we are done
Maintain the Structure Property

- The only valid place for a new node in a complete tree is at the end of the array
- We need to decide on the correct value for the new node, and adjust the heap accordingly

Maintain the Heap Property

- The new value goes where?
- We can do a simple insertion sort operation to find the correct place for it in the tree
Insert: Percolate Up

- Start at last node and keep comparing with parent A[i/2]
- If parent larger, copy parent down and go up one level
- Done if parent ≤ item or reached top node A[1]
- Run time?

Insert: Done

- Run time?
PercUp

- Define PercUp which percolates new entry to correct spot.
- Note: the parent of i is i/2

\[
\text{PercUp}(i : \text{integer}, x : \text{integer}) : \{ \\
\text{???
\}
\]

Sentinel Values

- Every iteration of Insert needs to test:
  \(\rightarrow\) if it has reached the top node \(A[1]\)
  \(\rightarrow\) if parent \(\leq\) item
- Can avoid first test if \(A[0]\) contains a very large negative value
  \(\rightarrow\) sentinel \(-\infty <\) item, for all items
- Second test alone always stops at top

<table>
<thead>
<tr>
<th>value</th>
<th>(-\infty)</th>
<th>2</th>
<th>3</th>
<th>8</th>
<th>7</th>
<th>4</th>
<th>10</th>
<th>9</th>
<th>11</th>
<th>9</th>
<th>6</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</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>

4/15/2004 Priority Queues 25

4/15/2004 Priority Queues 26
Binary Heap Analysis

- Space needed for heap of N nodes: \( O(\text{MaxN}) \)
  - An array of size MaxN, plus a variable to store the size N, plus an array slot to hold the sentinel
- Time
  - FindMin: \( O(1) \)
  - DeleteMin and Insert: \( O(\log N) \)
  - BuildHeap from N inputs: \( O(N) \)

How is this possible?

Build Heap

```plaintext
BuildHeap {
    for i = N/2 to 1 by -1 PercDown(i, A[i])
}
```

N=11

![Binary Heap Diagram]
Build Heap

4/15/2004 Priority Queues 29

Build Heap

4/15/2004 Priority Queues 30
Analysis of Build Heap

- Assume \( N = 2^K - 1 \)
  - Level 1: \( k - 1 \) steps for 1 item
  - Level 2: \( k - 2 \) steps for 2 items
  - Level 3: \( k - 3 \) steps for 4 items
  - Level \( i \): \( k - i \) steps for \( 2^{i-1} \) items

\[
\text{Total Steps} = \sum_{i=1}^{k-1} (k-i)2^{i-1} = 2^k - k - 1
\]

\( = O(N) \)

Other Heap Operations

- \textbf{Find}(X, H): Find the element X in heap H of N elements
  - What is the running time? \( O(N) \)
- \textbf{FindMax}(H): Find the maximum element in H
- Where \textbf{FindMin} is \( O(1) \)
  - What is the running time? \( O(N) \)
- \textbf{We} sacrificed performance of these operations in order to get \( O(1) \) performance for \textbf{FindMin}
Other Heap Operations

• DecreaseKey\((P, \Delta, H)\): Decrease the key value of node at position \(P\) by a positive amount \(\Delta\), e.g., to increase priority
  › First, subtract \(\Delta\) from current value at \(P\)
  › Heap order property may be violated
  › so percolate up to fix
  › Running Time: \(O(\log N)\)

• IncreaseKey\((P, \Delta, H)\): Increase the key value of node at position \(P\) by a positive amount \(\Delta\), e.g., to decrease priority
  › First, add \(\Delta\) to current value at \(P\)
  › Heap order property may be violated
  › so percolate down to fix
  › Running Time: \(O(\log N)\)
Other Heap Operations

- Delete(P,H): E.g. Delete a job waiting in queue that has been preemptively terminated by user
  › Use DecreaseKey(P,∞,H) followed by DeleteMin
  › Running Time: O(log N)

Other Heap Operations

- Merge(H1,H2): Merge two heaps H1 and H2 of size O(N). H1 and H2 are stored in two arrays.
  › Can do O(N) Insert operations: O(N log N) time
  › Better: Copy H2 at the end of H1 and use BuildHeap. Running Time: O(N)
PercUp Solution

PercUp(i : integer, x : integer): {
    if i = 1 then A[1] := x
    else if A[i/2] < x then
        A[i] := x;
        Percup(i/2, x);
    else
        A[i] := A[i/2];
        Percup(i/2, x);
}