CSE 373: Heaps (Priority Queues)

Chapter 6

Motivation

We’d like a data structure that stores the programs currently running on a computer

- a queue provides a “fair” data structure since it has FIFO ordering
- but, sometimes things shouldn’t be exactly fair
  - system administrator may need to run something of high priority
  - user may have job that isn’t urgent
  - interactive applications should perhaps run more often than long numerical computations
  - run short applications first to get them out of the way
One Approach

Use an array of queues

<table>
<thead>
<tr>
<th>high priority</th>
<th>backup</th>
<th>dir</th>
<th>mon</th>
<th>...</th>
<th>...</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>medium priority</td>
<td>quake</td>
<td>ppt</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>low priority</td>
<td>hist</td>
<td>word</td>
<td>clock</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

But what if there were 100 priority levels rather than just three?

Priority Queue Goals

- We’d like a data structure that allows us to find its lowest (highest) stored value quickly
- Inserts should also be fast
- Current Approaches:
  - simple list
  - sorted list
  - binary search tree
  - hash table
(Binary) Heap Structure

Heaps will always be stored as a complete binary tree:

```
                  A
                 / \
                B   C
               / \  / \  
              D  E F  G
             /   \  /   
            H    I  J   K
```

Note that a complete tree’s bottom level need not be completely full – but it must fill left to right.

Heap Order

Each node must be smaller than its descendants:

```
                  2
                 / \  
               4   5
              / \  / \  
             7  6 10  8
        11 9 12 13 20

                  2
                 /  
               4   5
              / \  / 
             7  6 1
Binary Heap: Array Implementation

More on Array Implementation

left(i) = 2i
right(i) = 2i + 1
parent(i) = ⌊i/2⌋
Heap Implementation

typedef struct _HeapStruct {
    HeapType* data;
    int capacity;
    int size;
} HeapStruct;

typedef HeapStruct* Heap;

Heap Operations

- Main Operations
  - void Insert(Heap, HeapType);
  - HeapType DeleteMin(Heap);
  - HeapType FindMin(Heap);

- Normal Creation/Deletion operations
- No iteration
- Other Operations:
  - void DecreaseKey(Heap, Position, int);
  - void IncreaseKey(Heap, Position, int);
  - Heap BuildHeap(HeapType []);
  - void Delete(Heap, Position);
**FindMin()**

• Trivial...

```
HeapType FindMin(Heap H) {
    // Implementation...
}
```

**DeleteMin()**

```
// Implementation...
```
DeleteMin() – Continued

Insert()
**Insert () – Continued**

**Heap Operator Summary**

- problem size
- space
- `FindMin()`
- `DeleteMin()`
- `Insert()`