Heaps

Or priority queues
What Is a Priority Queue? (Heap)

- Like a queue but prioritizes what gets to come out first.
  - No longer FIFO… more like First in Highest Priority Out.
  - Everything that goes into the Heap goes in with a priority.
  - Whatever has highest priority gets out first.

- Like a line, but get to skip ahead if you are “special”
Example 1

• Smart printer queues.
  • Totally sucks when someone tries to print out their 100 page senior thesis right before you try to print your 3 page homework assignment.
  • A priority queue (or heap) would recognize that some 3 page papers should be given higher priority to make more efficient use of printer time.
Example 2

• Smart printer queues again.
  • Or suppose your senior thesis is due in 1 hour and 30 people are already in the queue to print pretty pictures of beaches in Tahiti.
  
• A Heap could assign priority to the senior thesis and make sure it prints next.
Example 3

• Suppose the White House has a finite number of phone lines.

  • All the lines are busy, but the vice-president wants to make a call. High priority!
  • Then president wants to call. Highest priority!
  • Then the red “hot-line” phone needs to ring. Absolute highest priority.
Example 4

- Cafeteria line
  - Assign highest priority to people with classes in 10 minutes
  - Lower Priority to people with classes in next half hour.
  - Lowest priority to everyone else.

- So bubble up to the front of line if your priority is high enough.
- BUT, once in line, your priority may change (time flies), so might need to percolate up the line even further.
Heap Is My Preferred Name

• The name priority queue is a little misleading
  • we will show that binary trees are a better implementation.

• But the Heap ADT is closer to a Queue ADT than to a Tree ADT or…
  • enqueue is now called insert
  • dequeue is now called deleteMin

• And can do a queue implementation.
Heap ADT

- Operations:
  - insert
  - deleteMin

- That’s it! Can add others for convenience, but not required.
  - e.g., isEmpty
Implementation 1

• Queue implementation
  • Assume a fixed number of priority levels: 1 (LOW), 2 (MEDIUM), and 3 (HIGH).
  • Then maintain three queues.
    – everyone with priority 1 is added to that queue
    – etc.
  • Always remove from priority 3 queue unless empty.
    – then remove from priority 2 queue unless empty
    – finally remove from priority 1 queue

• Easily generalized to N priority levels.

Remember: queue insert and delete are O(1).

So what’s the run time for deleteMin? Insert?
Implementation 2

• Binary Search Tree Implementation
  • Assume variable priority levels.
    – e.g., 1.2, 3.8, 10.0
  • “insert” into position based on priority.
  • “deleteMin” by getLeftMost
    – why does this work?

• O(logN) for both operations *on average*.

• But we can do better!
  • Search trees can be unbalanced. Ugh.
  • *And*, we can make a heap with O(logN) as *worst case* for insert and deleteMin.
  • *And*, we’ll have O(1) *on average* for insert. (Sweeeet.)
Binary Heap: A Better Implementation

- Complete binary tree
  - can implement as an array!
    - remember why?

- Assume lowest number is highest priority

- Insist that every node be smaller than its descendants.
  - no left or right requirement
  - so smallest node is at the root
  - so always know where to find it. FAST!
    - except will have to rearrange tree to fill in root position
      » but can do this quickly (stay tuned).
Binary Heap Example
Not a Heap: Why Not?
Not a Heap: Why Not?

uh huh… not a complete binary tree
How to Insert on Heap

1. Create a hole at the next available location.
2. If can put new number there, then done.
3. If too small, then slide parent to last position.
4. Now try to insert at this location.
5. Slide next parent if necessary, etc.

Called “percolating up”.
1. Insert 14 into the tree.

2. The tree after insertion.
Picture: Insert 14 (cont.)

3.

4.

Done!
Insert Growth Rate

• $O(\log N)$ worst case.
  • Must percolate up entire tree.

• $O(1)$ on average!
  • Don’t usually have to percolate too far up the tree.
    – The number will usually be an average value (not too big not too small).
    – For random distribution of numbers, going up one level means that you are smaller than roughly half the values.
      » Why? The values in the lower level are bigger.
      » And going up a level roughly halves the number of nodes. (Going down a level in a binary tree roughly doubles the number of nodes.)
        At level 1: 1 node total in tree
        At level 2: 3 nodes total in tree
        At level 3: 7 nodes total in tree
        At level 4: 15 nodes total in tree
        Etc.
    – So will usually only go up roughly one level!

• Confirmed with numerical experiments.
How to deleteMin From Heap

1. Remove minimum (root) and leave a hole.
2. Also remove the last element, but hang onto it.
3. If last element can be inserted into the hole, then done.
4. Else slide the smaller of the two children up to the root.
5. If can put last element in hole, then done.
6. Else continue sliding…

Called “percolating down”.
1. Picture of deleteMin 1

2. Picture of deleteMin 1
Picture of deleteMin 2

3.

4.
5.

6.
DeleteMin Growth Rate

- So that last example was a worst case
  - percolate through entire tree

- So $O(\log N)$ worst case for deleteMin.
  - why?

- Also $O(\log N)$ on average for deleteMin.
  - The value we are percolating came from the bottom of the tree!
  - So usually will have to percolate all of the way back down!
public class BinaryHeap
{
    //num elements stored in array
    public int currentSize = 0;

    public int[] elements;

    public BinaryHeap(int maxSize)
    {
        elements = new int[maxSize];
    }
}
Heap Insert: Java

```java
public void insert(int insertThis) {
    int i;
    currentSize++;
    for(i = currentSize; i > 0 && elements[(i-1)/2] > insertThis; i = (i-1)/2) {
        elements[i] = elements[(i-1)/2];
    }
    elements[i] = insertThis;
}
```