Heaps

c++,heaps,lecture15

Priority Queue

- supports three operations
    - insert a new item into the queue
    - get the value of the highest priority item
    - remove the highest priority item from the queue
- when you define a priority queue, you must specify how to determine the priority of each item in the queue
    - e.g. **priority = amount of blood lost + number of cuts**
- priority queues with many priority levels can be implemented using a **HEAP**

Heap

- uses a special type of binary tree to hold its data
    - **However, it is NOT a binary search tree**
- All heaps use a “complete” binary tree

Complete Binary Tree Requirements

- the top N-1 levels of the tree are completely filled with nodes
- all nodes on the bottom-most level must be as far left as possible (with no empty slots between nodes)

Two Types of Heaps

Maxheap

- quickly insert a new item into the heap
- quickly retrieve the **largest** item from the heap

Minheap

- quickly insert a new item into the heap
- quickly retrieve the **smallest** item from the heap

The Maxheap

Requirements

- the tree is a complete binary tree
- the value contained by a node is **always greater than or equal to** the values of the node's children
    - this means the top item is **ALWAYS** the max
- the rules for a minheap are the same but replace "greater" with "less"

Extracting the Biggest Item

1. If the tree is empty, return error
2. Otherwise, save the top item for later since it's the max.
3. If the heap has only one node, then delete it and return the saved value.
4. Copy the value from the right-most node in the bottom-most row to the root node.
5. Delete the right-most node in the bottom-most row.
6. Repeatedly swap the just-moved value with the larger of its two children until the value is greater than or equal to both of its children (**sifting down**).
7. Return the value saved in step 2 to the user.

Adding a Node to a Maxheap (reheapification)

1. If the tree is empty, create a new root node & return.
2. Otherwise, insert the new node in the bottom-most, left-most position of the tree (so it's still a complete tree).
3. Compare the new value with its parent's value.
4. If the new value is greater than its parent's value, then swap them.
5. Repeat steps 3-4 until the new value rises to its proper place (**sifting up**).

Implementing a Heap

- heaps can easily be implemented using an array
- root of the heap goes in array[0]

Properties

- We can always find the root value in heap[0]
- We can always find the bottom-most right-most node in heap[n-1]
- We can always find the bottom-most left-most empty spot (to add new value) in heap[n]
- We can add or remove a node by simply setting heap[n] = value and/or updating our count

Locating a Node's Children in a Heap

- leftChild(parent) = (2 • parent) + 1
- rightChild(parent) = (2 • parent) + 1
- parent(child) = (child - 1) / 2

```
class HeapHelper {
   HeapHelper()                { num = 0;  }
   int GetRootIndex()          { return(0); }
   int LeftChildLoc(int i)     { return(2*i+1); }
   int RightChildLoc(int i)    { return(2*i+2); }
   int ParentLoc(int i)        { return((i-1)/2); }
   int PrintVal(int i)         { cout << a[i]; }
   void AddNode(int v)         { a[num] = v; ++num;}
private:
  int a[MAX_ITEMS];
  int num;
};
```

Big-O

Insertion: $O(\log_2 N)$
Deletion: $O(\log_2 N)$

Naive Heapsort

Algorithm

Given an array of N numbers that we want to sort:
Insert all N numbers into a new maxheap.
Whle there are numbers left in the heap,
  Remove the biggest value from the heap.
  Place it in the last open slot of the array.

Big-O: $O(N \bullet \log_2 N)$

Effecient (Official) Heapsort

Algorithm

Given an array of N numbers that we want to sort:
Convert our input array into a maxheap.
Whle there are numbers left in the heap,
  Remove the biggest value from the heap.
  Place it in the last open slot of the array.

Convert input array to maxheap algorithm (BIG-O: O(N))

For the nodes from N/2 -1 through the rootNode,
  Focus on the subtree rooted at the current node.
  Think of this subtree as a maxheap.
  Keep shifting the top value down until your subtree becomes a valid maxheap.

Big-O: $O(N \bullet \log_2 N)$