Tree Data Structure

Vivek K. S., Deepak G.

Information Systems Decision Sciences (ISDS)
MUMA College of Business
University of South Florida
Tampa, Florida

2017

Introduction

- Queue is a first-in-first-out data structure and it has a very important variation called a priority queue.
- A priority queue acts like a queue, except that the dequeueing happens from the front of the queue.
- The logical ordering of items insider a priority queue is also determined by their priority.
- The classic way to implement a priority queue is by using a data structure called a binary heap.
- The binary heap is implemented with a single list as an internal representation and has two variations, the "min heap" and the "max heap".
- In the min heap, the smallest key is at the front and its the opposite in the max key.

Essential Operations in a Binary Heap

- Ability to create a new empty Binary Heap.
- Ability to insert a new item.
- Finding the item with the minimum key value in the heap.
- Removing the item with the minimum key value.
- Find out if the heap is empty.
- Find out the size of the heap.
- Building a new heap from a list of keys.

Properties of a Binary Heap

Lets discuss the properties of a Binary heap before we go ahead with the implementation. Structure Property:

- For the heap to work efficiently, it must preserve the logarithmic nature of the binary tree.
- The tree must remain balanced, to ensure logarithmic performance. A balanced binary tree has roughly the same number of nodes in the left and right subtrees.
- A complete binary tree is a tree in which each level has all its nodes except the bottom level of the entire tree.
- An interesting property of a complete tree is that we can represent it using a single list.
- In such an implementation, the node at index p is the parent of the node found at position 2p which is actually its left child.
- Therefore, intuitively, the node at 2p+1 becomes the right child.

Contd..

- Using this property, we can easily identify the parent of any node using simple integer division.
- For a node at position p, its parent node will be located at p//2. Remember that integer division only returns the quotient part of the division.
- Therefore, this method works for both nodes (left or right child node).
- Thus, in a complete binary tree, where the structural property is preserved, the nodes can be traversed easily using simple mathematical operations.

Heap Order Property

- Along with the structure property, the Heap order property is very important in building a Binary Heap.
- Its defined as follows: For every node n with parent p, the key in p is smaller than or equal to the key in n.
- These two properties are absolutely necessary to be in place for us.

Python Implementation

- As discussed before, the Binary heap can be represented by a single list. The class definition will be as follows:
- We will initialize a list and also add another attribute to keep track of the size of the list.
- For ease of doing mathematical operations, we will add a single element to the list (just as a placeholder). We will add a zero for this purpose.
- Note that this zero has no significance in this implementation whatsoever. It is added just to make the math easier.

```
class Binary_Heap:
    def __init__(self):
        self.h_list = [0]
        self.size = 0
```

Adding an Item

- Insertion can be done by using the append method of lists.
- But it is imperative that we maintain the heap order property while doing it.
- Hence, when the item added is less than its parent, we need to move it up the tree.
- We call this percolate up.
- We may have to percolate the newly added item up till we find that the parents and child nodes are all in order.
- Here is the implementation of this logic.

```
def percolate_up(self, i):
    while i//2 > 0:
        if self.h_list[i] < self.h_list[i//2]:
            self.h_list[i], self.h_list[i//2] =
            self.h_list[i//2], self.h_list[i]
        i = i//2
```

Contd..

We are now ready to write the insert method. Most of the work in the insert method is really done by percolate_up.
Once a new item is appended to the tree, percolate_up takes over and positions the new item properly.

```
def insert(self, item):
    self.h_list.append(item)
    self.size = self.size + 1
    self.percolate_up(self.size)
```

As you can see, once the item is appended, the size is incremented by 1 and then the control goes to the percolate_up method.

Delete Minimum

Deleting the minimum key in the heap is again going to violate the heap structure and order property.

- Finding the min item is easy but restoring the order and structure of the heap is the hard part.
- We'll do that in two steps. First, after removal of the minimum element, we'll move the last item in the list to the root position.
- Moving the last item restores the structure of the heap. But we need to restore the order and for that we need to move the new root down the tree to its appropriate position.
- We call this percolate down.
- Using the method, we swap the parent and child nodes until the order is achieved.

Implementation

The percolate down method

```
def percolate_down(self, i):
    while (i * 2) <= self.current_size:
        mc = self.min_child(i)
        if self.h_list[i] > self.h_list[mc]:
            self.h_list[i], self.h_list[mc] =
             self.h_list[mc], self.h_list[i]
        i = mc
```

Method to identify the min child.

```
def min_child(self, i):
    if i * 2 + 1 > self.size: return i * 2
    else:
        if self.h_list[i*2] < self.h_list[i*2 +1]:
            return i * 2
        else: return i * 2 + 1
```

Implementation

Now that we have implemented all the supporting methods, we will implement the delete method as follows:

```
def del_min(self):
    ret_val = self.h_list[1]
    self.h_list[1] = self.h_list[self.size]
    self.size = self.size - 1
    self.h_list.pop()
    self.percolate_down(1)
    return ret_val
```

Summary

- Binary Heaps is an important tree data structure wherein the properties of a queue and a tree are combined together.
- The properties of a binary heap dictates that the structure property and the order property have to be maintained to get the best performance from the data structure.
- It could be implemented using a single list.