Jeffrey Lansford

12/4/2019

Lab 8

Introduction

In this lab we are tasked with creating Dijkstra Algorithm with two types of data structures for the priority queue, a list and the min-Heap. We must modify our graph class to handle weighted edges and show the runtimes of both implementations.

Methods

I implemented Dijkstra's Algorithm with the pseudocode provide by the book

```
Dijkstra(G, l, s):
    For all u ∈ V:
        $\text{Dist}(u) = \infty$
    $\text{Dist}(s) = 0$
    $H = \text{make-queue}(V)$
    While H is not empty:
        $u = \text{deleteMin}(H)$
        for each edge $(u,v) \in E$ :
            if $\text{dist}(v) > \text{dist}(u) + l(u,v)$ :
                $\text{dist}(v) = \text{dist}(u) + l(u,v)$
                decreaseKey( H, v)
```

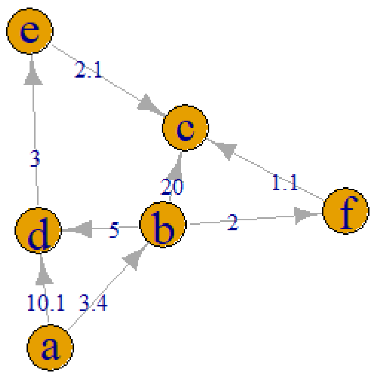
I also implemented the Binary heap-based pseudocode provide by the book also. Some of my solutions differ from the book as the book provide more of an abstracted basis of the code. I had to some operations different to account the index for each element in heap.

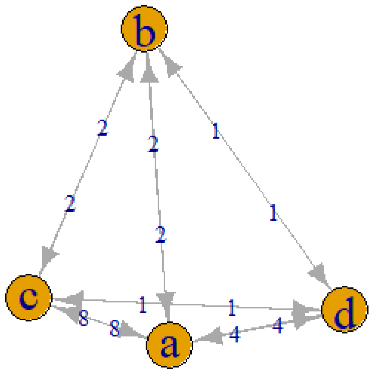
```
Heap:
    Make heap(S):
        H = empty array Of size S
        For x ∈ S:
            H(|H| +1 ) = x
        For i=|S| downto 1:
            Shiftdown(H,H(i),i)
    decreaseKey(h,x):
        bubbleUp(h,x,|h|+1)
    deleteMin(h):
        if |h| = 0
            return null
        else:
            x=h(1)
            shiftdown(h,h(|h|), 1)
    bubbleup(h,x,i):
        p = ⌊i/2⌋
        while i ≠ 1 and key(h(p)) > key(x):
            h(i) = h(c); i=c; c =minChild(h,i)
        h(i) = x
    minChild(h,i:
        if 2i > |h|:
            return 0;
        else:
            return argmin{key(h(j)) : 2i ≤ j ≤ min{|h|, 2i+1}}
```

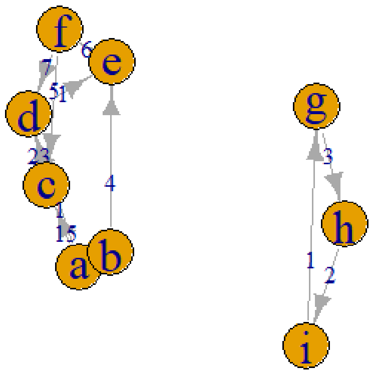
For list, the remove-min is pretty simple where you have to go through the list to find the minimum distance. Decrease-key is not helpful in my program as the list is automatically updated when the distance values are changed in the Graph, i.e. it gets distance by looking at the Graph structure. Remove-min for the heap is also pretty straight forward. You must remove the root as it is the minimum already the minimum in the heap, then shift down the new root you set. The Decrease-key operation sets the key values and does bubble-up on it to have it in the right position in the heap. The runtime of the list will be O(|V|^2) because its deleteMin() function as a runtime of O(|V|), which is called for each element in the while loop, so |V| * |V|. The runtime of the heap is O((|V| + |E|)log(|V|)). Since remove-min and decrease-key take log(|V|), the height of the heap tree, and remove-min is dependent of |V| and decrease-key is dependent on |E|, they would be |V|log(|V|) and |E|log(|V|).

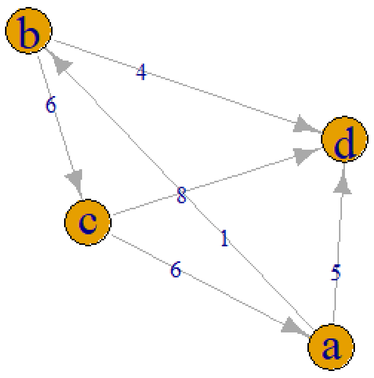
Results

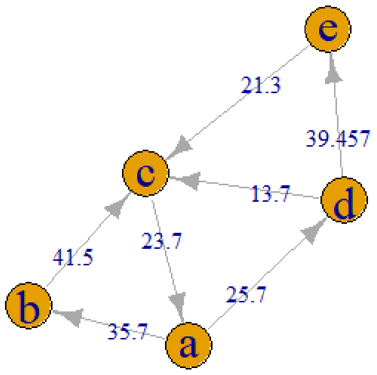
Tests files visualization:




b
c
a
d
2
2
2
2
1
1
1
1
8
8
4
4

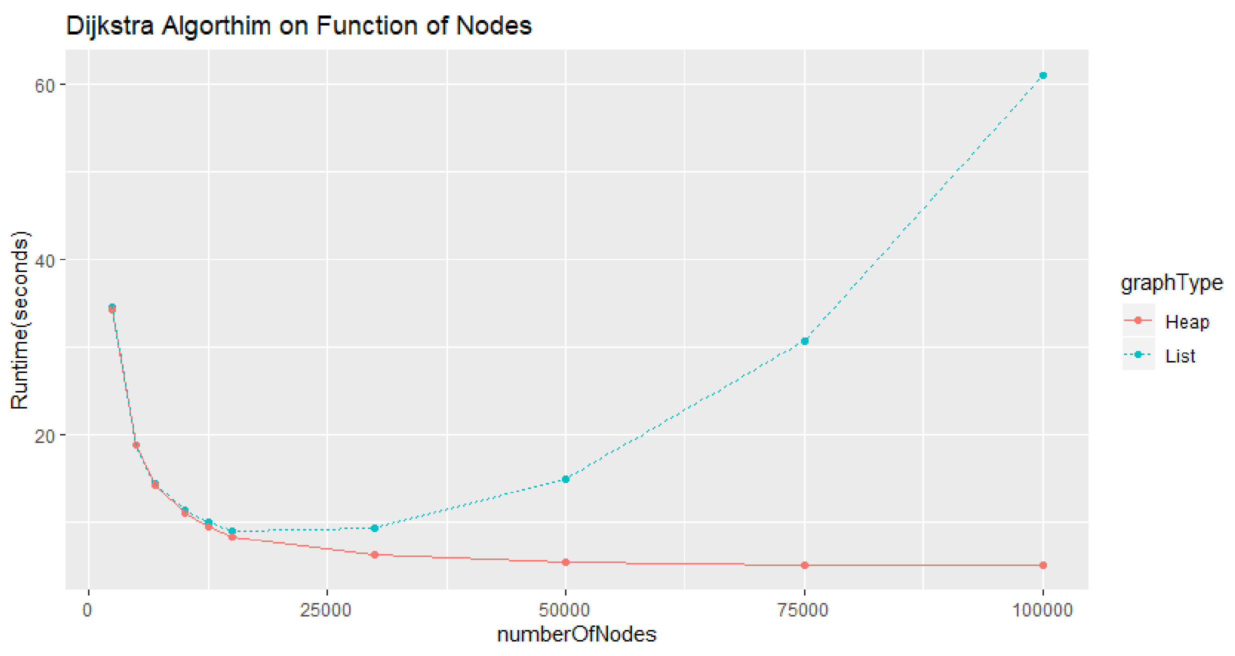

f
e
d
c
a
b
g
h
i
6
7
51
23
4
1
15
3
1
2



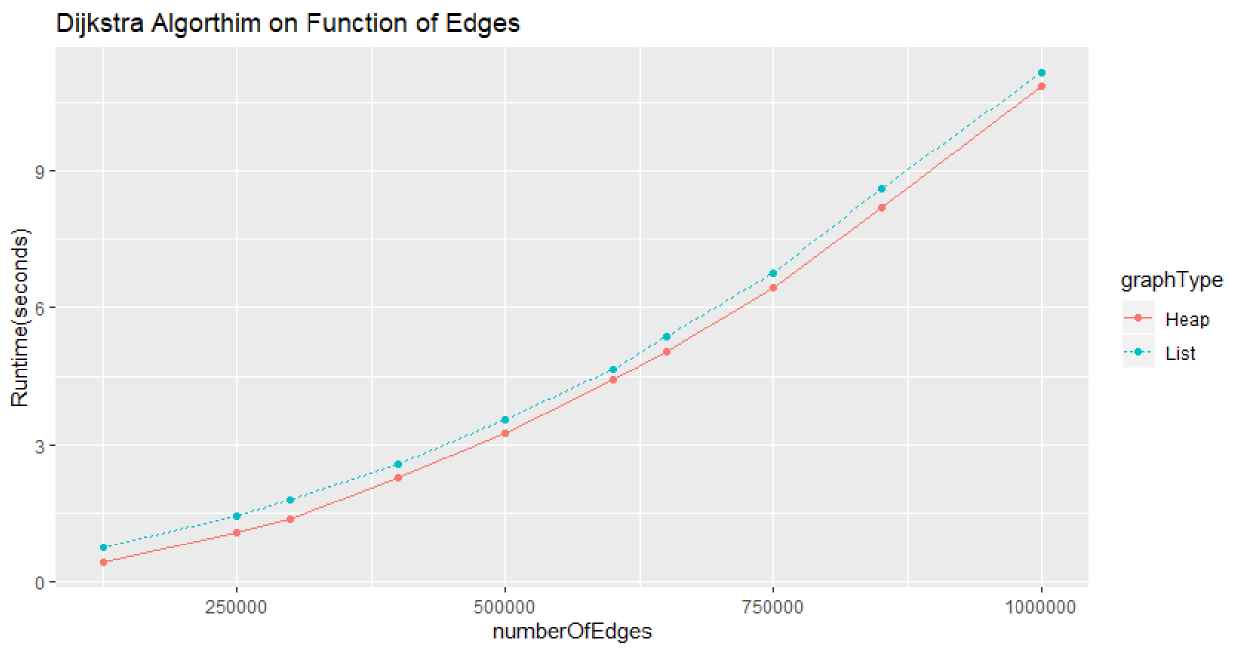


Graphs of comparing the two ways of doing Dijkstra’s algorithm

Function of Nodes:



Function of Edges:



I believe that our results are consent with our exceptions. From the node graph, we can really see the difference between the two implementations. Even though they have high run times at the beginning, they begin to separate from each other. The List graph starts to from a quadratic graph, consistent with our theoretical runtime. Heap continues to decrease from the graph, or at least appears that it does. Maybe with larger test sizes we can see the graph increase in runtime, but it starts increasing way before than the List which is accurate to our theorical runtime. With a function of edges, we can see that edge size does not influence these graphs as much as nodes does. We expected this as both theoretical runtimes because they have very little effect from Edges.

Discussions

I created a little graph with a negative to show how Dijkstra's algorithm does not work with negative weights.

```
    a Pre: 0 Post: 0 Distance: 0
    b Pre: 0 Post: 0 Distance: 5
    c Pre: 0 Post: 0 Distance: -5

    Adjacency list of the graph :
    Node a, ID:0: b: 5, c: 2,
    Node b, ID:1: c: -10,
    Node c, ID:2:
    List is wrong
    Tests Failed
```

The answer should be a:0, b:5, c:2, but the algorithm updates the distance when it sees that the negative value would be smaller than the current distance set for c. Its right, but really thing with the alarm clocks, a->c would go off before a->b even ends.

Overall, I enjoyed this lab. Learning to create a Binary Heap and the way the algorithm works with was cool. In this lab, I used a debug tool within my editor to help ease my debugging and made it way easier to see how certain components work. What I had issues with mostly was implementing an index for the heap and adding weights to the graph without sacrificing so much runtime. Before, my graphs created graphs slower unlike other labs, so I experimented with it to have it not as slow. My current setup is that there is a hash table for each edges weight with the key as their names concatenated into a string. Before when I thought this was the problem, I experimented with an array of arrays to hold each weight. This fixed for smaller graphs, but larger took for ever and took up a lot of space on my RAM. I have 16 GB of RAM and it took 96% - 99% with the larger graphs. Finally, I figured out that it was the way I was scanning the file. I had a double variable where it would just output that to the variable, but it took very long to do that. Instead, I outed it to a string and make a double conversion when I needed it.

Conclusions

This lab we created Dijkstra's algorithm. It showed us that it is important not just to consider using a data structure, but what kind of data structure as some have better applications than others.