Jeffrey Lansford

10/16/2019

Lab 5

Introduction

We are task with designing a DFS search algorithm with our Graph class we created earlier in Lab 4. We had to create a recursive and iterative algorithm and compare how they differ.

Methods

For the recursive algorithm, I followed the pseudocode that was discussed in class:

Procedure dfs(G)

For all v ∈ V:

    Visted(v) = false

For all v ∈ V:

    If not visted(v):

        ExploreR(G,v)

And the explore function was also discussed in class with the pseudocode:

Procedure exploreR(G,v)

Visted(v) = true

Previst(v)

For each edge (v,u) ∈ E:

    If not visited(u): explore(G,u)

Postvist(v)

For previst, postvist, and visited, I added values pre, post, and visted into the Node class and had mutators and accessors in the Graph functions to see and change this values.

For the iterative algorithm, I mostly used the BFS algorithm as a guideline, but it is not the same. I used a stack to keep track of nodes that were discovered to have it move through the list and keep track of visited nodes and change their post values accordingly.

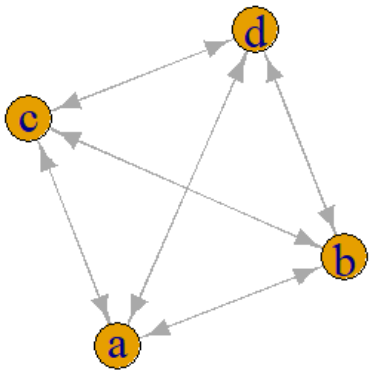
```
Procedure DFS(g)
Procedure dfs(G)
For all v ∈ V:
    Visted(v) = false
For all v ∈ V:
    If not visted(v):
        ExploreI(G,v)

Procedure exploreI(G,v)
D : stack of discovered nodes
Push v into D
While ( D is not empty)
    If (D.top is not visted )
        Visted(D.top())
        Previst(D.top())
    For each edge in (v,u) ∈ E:
        If (u is not visted)
            D.push(u)
    If ( D.top() is visted )
        If ( D.top() post is 0 )
            Postvist(D.top())
        D.pop()
```

I tested these two algorithms with five different graphs with answers of pre and post I did by hand.

Results

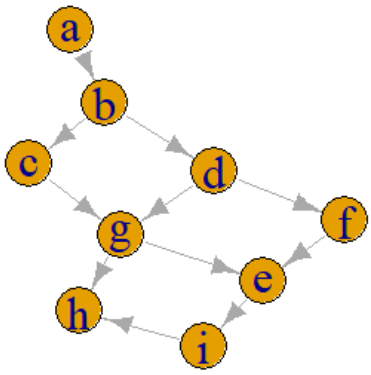
With my five different test graphs when testing it with C, I graph they in R to show the different types.

For a cyclic graph, I graphed:



The double arrows mean it is undirected graph, but this graph has cycles in it and the two algorithms can correctly get the pre and post visits.

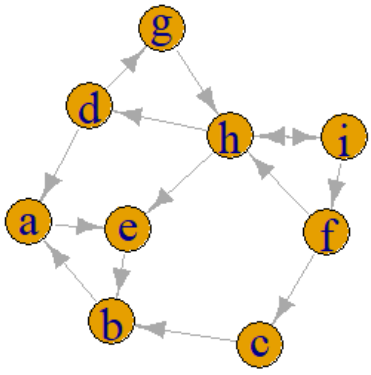
For an acyclic graph, I graphed:



This is a simple dag with not cycles.

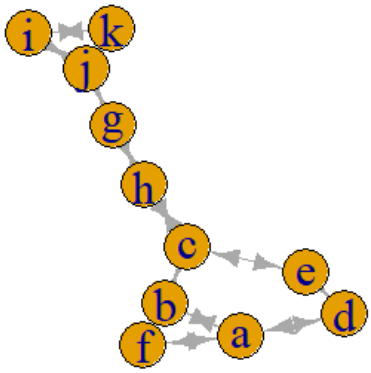
For a directed graph, I graphed:

With



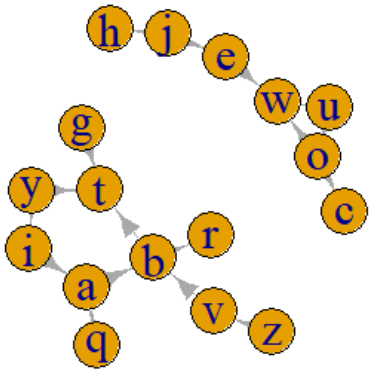
With this graph, I have cycles in it to help further test, but still a directed graph.

For a undirected graph, I graphed:



It is hard to see the double arrows, but it is an undirected graph

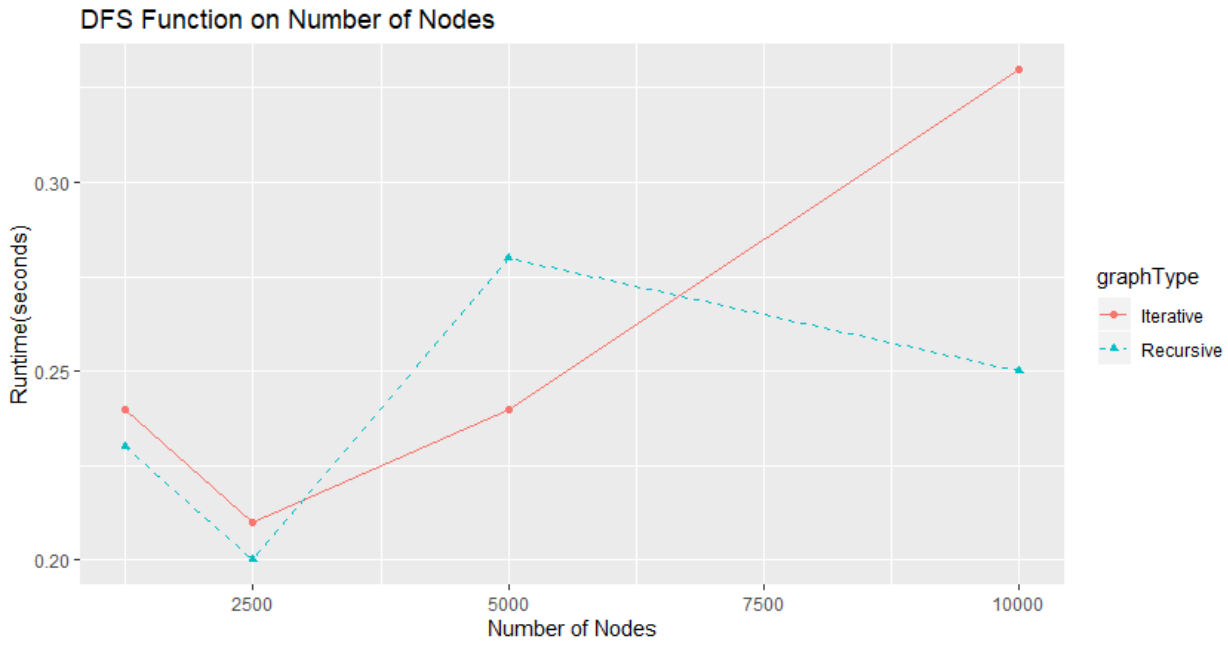
For a connected components graph, I graphed:



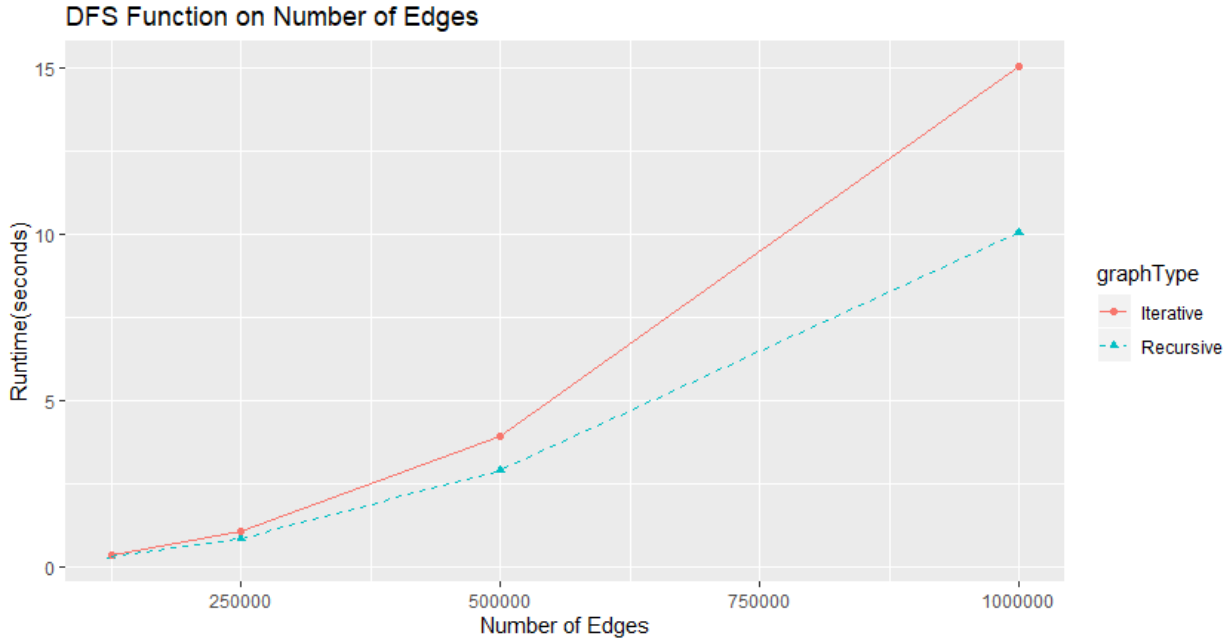
It is a directed graph with many connected components. DFS creates separate search trees with this graph.

For the four figures, I combine the iterative and recursive graphs to one chart to make the differences appear better. Also, the runtime analysis may not be the most accurate as this includes the runtime of creating graphs with DFS algorithms calls. I could not separate the creation of graph from how the DFS is designed.

Here is the graph with respect with number of Nodes:



And here is the graph with respect to number of edges:



These two graphs appear linear-ish. The Nodes graph is what surprised me the most with its shape as it slopes down and rises back up. I believe this is because of how we do the adjacent list and checking through it, if we have less nodes and way more edges, that list will be huge and a pain to go through and adds overhead to the algorithm. The same can be applied to edges as the number of edges increases and not the number nodes, it creates more over-head leading to the spike at higher number of edges.

Discussions

My number one issue is how sensitive the recursive algorithm is with heap overflow. If I go over 100,000 nodes, I get Segmentation faults from a stack overflow issue. This is because the recursive algorithm creates too many recursive calls from the huge number of nodes. I do believe there is no way around this without extending the heap or not using recursion. I do also wish there was a way to split the graph creation and using the DFS algorithms in R, but I see no possible way. It would provide a more accurate reading. Looking at the two graphs, I see how recursive can be faster than a loop solution, but after running into memory problems, I can see its downfall with larger datasets. Maybe you can fix this with having more memory in your computer, but I do have 16GB which is becoming a standard for all modern computers and laptops. Perhaps if I had 128GB I would not run into this issue.

Conclusions

We created DFS algorithms two different ways and showed about the difference between running them on functions of nodes and edges. The iterative solution takes longer on my machine, but can handle bigger data sets. The recursive solution can run faster but has limit of nodes and possible edges based on the system.