Jeffrey Lansford

10/30/2019

Lab 6

Introduction

We are to implement the algorithms of finding connected components. Both for connected and strongly connected components. Again we used previous labs algorithms, DFS from Lab5 and the graph class from Lab4, to find these components.

Methods

For the Connected Components algorithm, we only had to improve the DFS recursive algorithm to have it count the components

Procedure dfs(G)

For all v ∈ V:

$\text{Visted}(v) = \text{false}$

$\text{For all } v \in V\text{:}$

$\text{If not visted}(v)\text{:}$

$\text{Count Connected Components}$

$\text{ExploreR}(G,v)$

I had to do it a little differently to have the function return it as a vector with each node have it component number with it attached, but what I did was have an vector and integer in the parameters in Explore and have it add it to the index of what node it is. It allows for it to work with the recursion better.

For Strongly Connected Components, we had to produce the reverse graph of G, then do DFS on it, then do DFS on G based on the post Numbers of G^R. For Reverse function, we are provided the solution of the homework question to accomplish this in linear time.

Reverse(G)

1 Input: Graph G = (V, E)

2 Output: Graph GR = (V, ER)

3 for each node v ∈ V :

```
4       Create an empty adjacency list for v in GR: Adj(GR, v) = {}
5 for each node s ∈ V :
6       for each adjacent node k ∈ Adj(G, s):
7               insert s into the adjacency list Adj(GR, k)
8 return GR
```

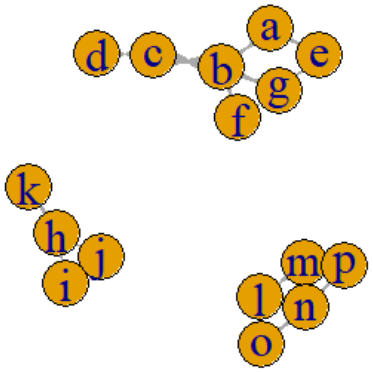
Then for SCC:

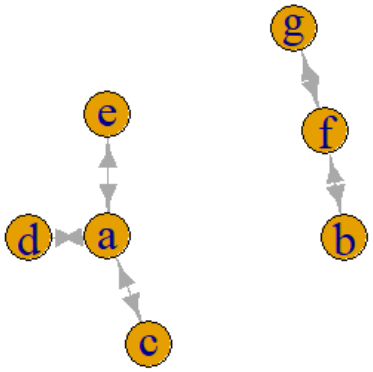
```
SCC(G)
1 Revsere(G)
2 DFS(G^R)
3  For all v ∈ V:
4       Visted(v) = false
5 For all v ∈ V based on post Numbers:
6       If not visted(v):
7               Count Connected Components
8               ExploreR(G,v)
```

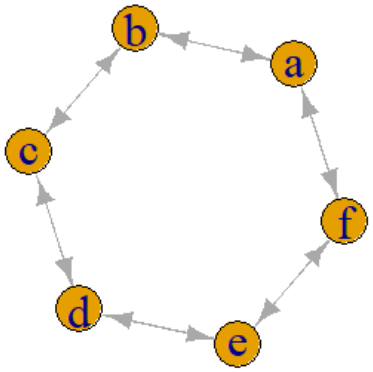
Results

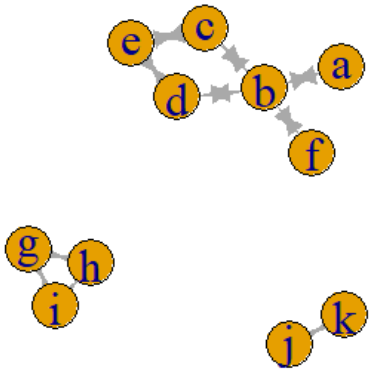
Here are my five graphs for Connected Components

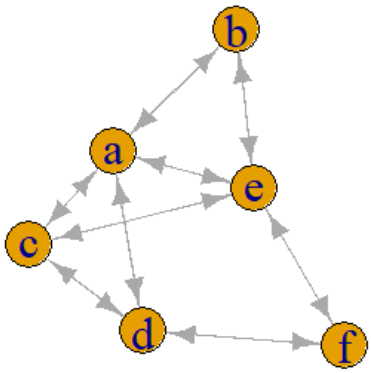
Connected Components :


a
d
c
b
e
g
f
k
h
j
i
m
p
l
n
o

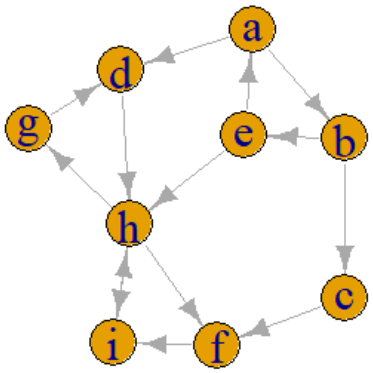

g
e
f
d
a
b
c

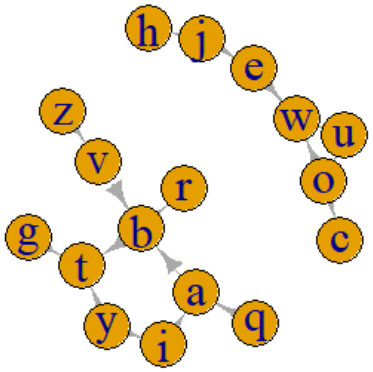

b
a
c
f
d
e

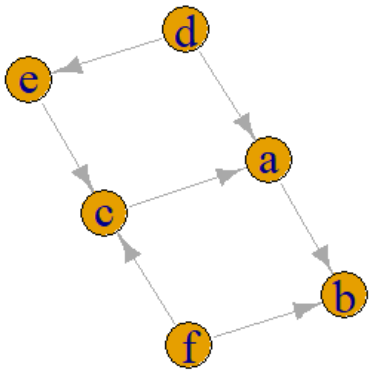

e
c
a
d
b
f
g
h
i
k
j

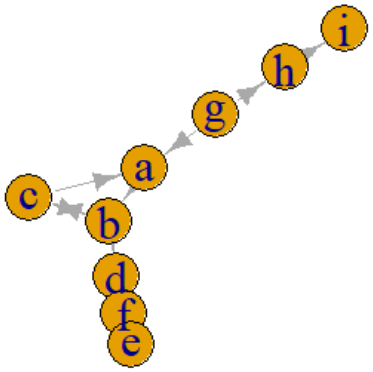


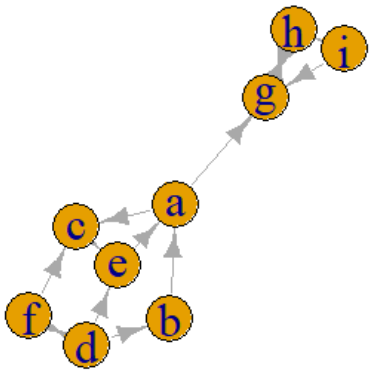
Here is the Strongly Connected Components:


a
d
g
e
b
h
c
i
f

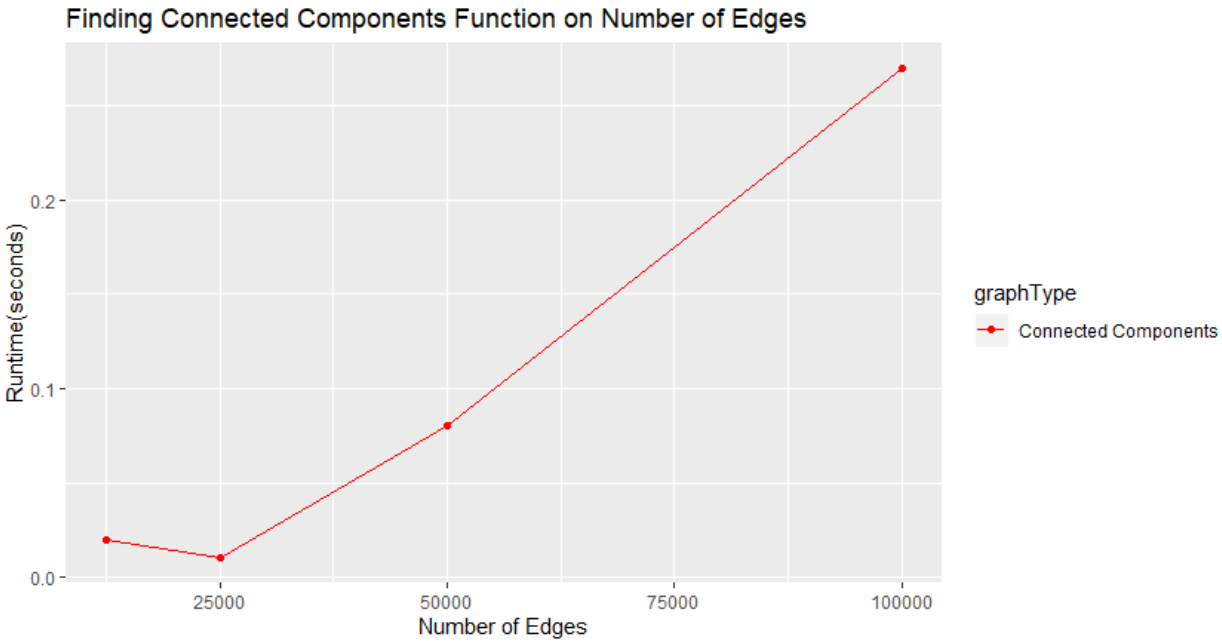

h
j
e
z
w
u
v
r
o
g
b
c
t
a
y
q
i

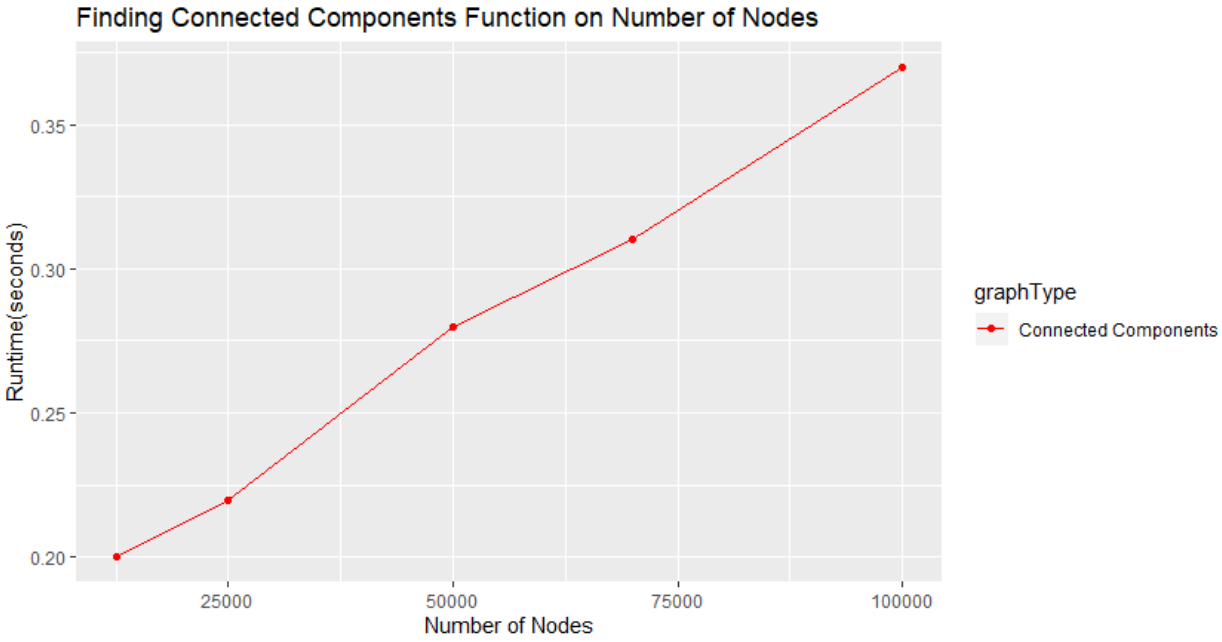

d
e
a
c
b
f

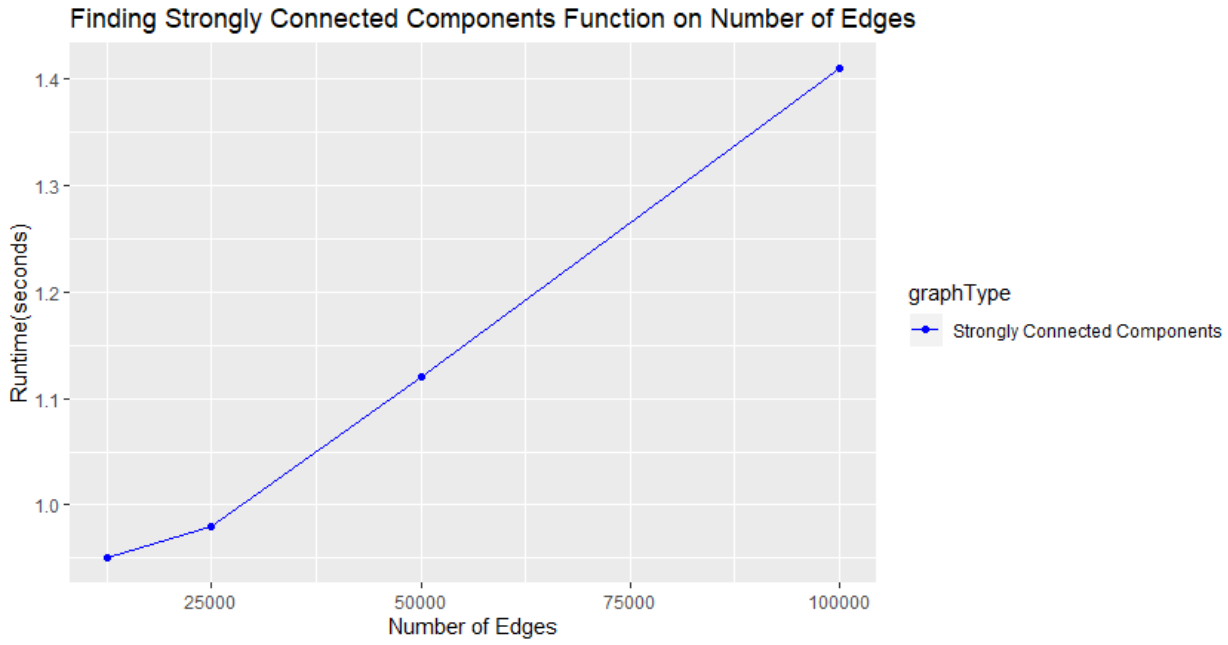

i
h
g
a
c
b
d
f
e

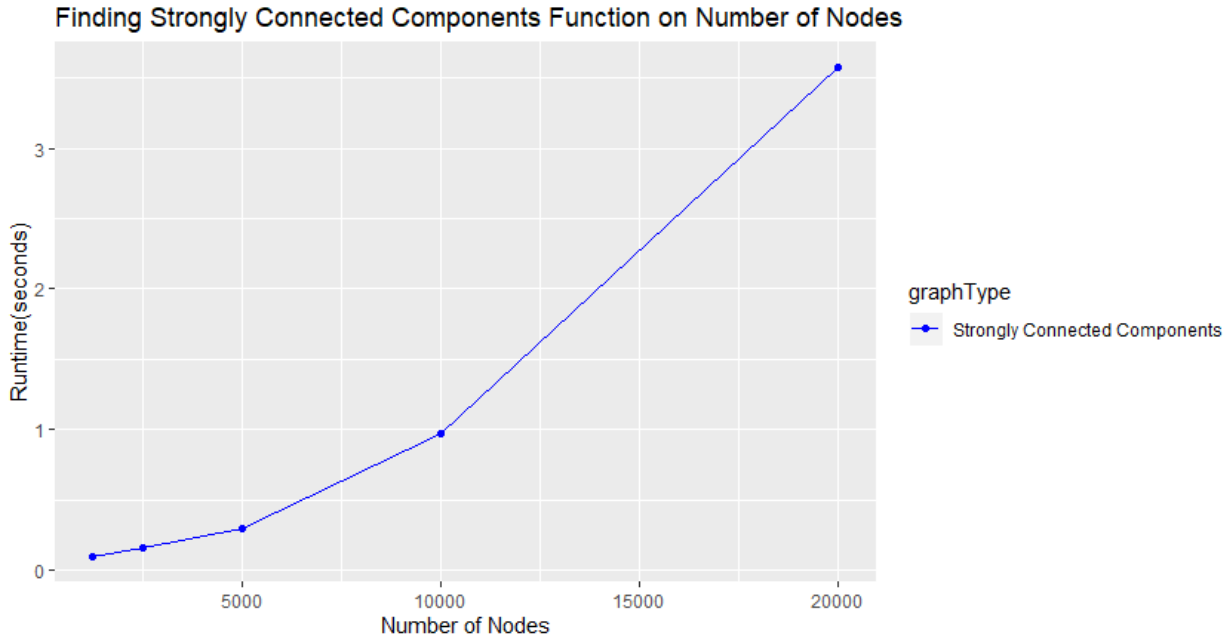


Here is the results from the testing each find algorithm based on functions of Number of edges and nodes:


Finding Connected Components Function on Number of Edges
Runtime(seconds)
0.0
0.1
0.2
25000
50000
75000
100000
Number of Edges
graphType
Connected Components


Finding Connected Components Function on Number of Nodes
Runtime(seconds)
0.20
0.25
0.30
0.35
25000
50000
75000
100000
Number of Nodes
graphType
Connected Components





Connected Components looks to be pretty consent linear graph. With CC, I was a little restricted to get any higher runtimes as I can only go up to 100,000 nodes before I get the Heap Overflow from the recursive algorithm that I discussed last lab. For Strongly Connected Components, the results are not as clean. As a function of Edges, it looks very good almost a perfect linear line for its runtime. The Number of Nodes, not so much. How I can explain for its very quadratic look is maybe how I must arrange the nodes to be in decreasing post number. I must find the max of all the nodes, then add to another vector and delete it. If I could find another way of arranging the nodes in linear time, or even less, this would reduce the runtime.

Discussions

I am using the DFS recursive solution because it is faster than the iterative solution, for me at least. If I probably used the iterative, I would be sitting for awhile waiting for the program to even finish. Also, the recursive solution provides an easier way of coding and not complicating the program. There is a less chance for bugs to happen. But as I have said before, there is a limitation with the recursive solution (on my machine I believe) where it will overflow the Heap. As I stated in my methods, I had to modify explore to manipulate the array instead of doing the way the book describes it. It provides a way of accounting every node has a Component Number associated with it without going back through the nodes. I do wonder if there is a way to modify DFS search on G^R to have to add the post numbers as it finds them. I thought about pushing back on a vector, put it could be disastrous with large graphs for the runtime.

Conclusions

We created the algorithms of finding connected components of undirected and directed graphs. We compared them on functions of nodes and edges to see the impacted they have on the runtime of the different algorithms. This lab brought us a better understanding of how connected components work and how they are found.