

a.)

tree

Forward – a descendent of a node that the edge is not in the DFS Tree

Back – point from a node to one of its ancestors in the DFS tree

Cross Edge- point from a node to a previously visited node that is neither an ancestor nor a descendant

DFS Problem

0 1 2 3 4 6 7 5 8 10 9 11

A D B C E G H F I K J L

DFS

t0

A t1 u.d 1

D t2 u.d 2

B t3 u.d 3

C t4 u.d 4

C t5 u.f 5

B t6 u.f 6

E t7 u.d 7

G t8 u.d 8

H t9 u.d 9

F t10 u.d 10

I t11 u.d 11

I t12 u.f 11

F t13 u.f 13

K t14 u.d 14

J t15 u.d 15

L t16 u.d 16

L t17 u.f 17

J t18 u.f 18

K t19 u.f 19

H t20 u.f 20

G t21 u.f 21

E t22 u.f 22

D t23 u.f 22

A t24 u.f 24

TREE EDGES

A -> D

D -> B

D -> E

B -> C

E -> G

G -> H

H -> F

F -> I

I -> K

K -> J

J -> L

BACK EDGES

D -> A

B -> D

E -> D

I -> F

K -> G

L -> K

CROSS

E -> C

FORWARD

H -> I

b.) Do the DFS Starting from L

L t1 u.d

K t2 u.d

G t3 u.d

H t4 u.d

F t5 u.d

I t6 u.d

I t7 u.f

F t8 u.f

H t9 u.f

G t10 u.f

J t11 u.d

J t12 u.f

K t13 u.f

L t14 u.f

Diagram, schematic

Description automatically generated

c.

G(T) – transform the graph by reversing the direction of all paths

Call DFS again on G(T) referencing the stack above for which nodes to go to when we hit a wall Stack

A t1 u.d

D t2 u.d

B t3 u.d

B t4 u.f

E t5 u.d

E t6 u.f

SCC1 A->D->B->E

G t7 u.d

K t8 u.d

H t9 u.d

H t10 u.f

L t11 u.d

J t12 u.d

J t13 u.f

L t14 u.f

K t15 u.f

G t16 u.f

SCC2 G->K->H->L->J

F t17 u.d

I t18 u.d

I t19 u.f

F t20 u.f

SCC3 F->I

C t21 u.d

C t22 u.f

SCC4 C

D.

Go through the original DFS with the answers found from 1b.

Based on that the Stack is

LKJGHFI

L t0 u.d

J t1 u.d

K t2 u.d

H t3 u.d

G t4 u.d

G t5 u.f

H t6 u.f

K t7 u.f

J t8 u.f

L t9 u.f

SCC1 L->J->K->K->H->G

F t10 u.d

I t11 u.d

I t12 u.f

F t13 u.f

SCC2 F-> I

Diagram, schematic

Description automatically generated

Abstract and Analysis

During this assignment, I was tasked with implementing a graph of the DFS and BFS. By definition, a graph is a data structure that consist of a set of vertices often called nodes, and a set of edges between pairs of vertices. Edges are defined as paths or connection between vertices and both vertices and edges must be finite. In a modern setting, a graph is a very useful data structure and most common is used for items such as a flight map. The example used in class is the most common and was of airports on a flight map and the routes that a plane may take. The running time complexity of both the dfs and bfs is [Θ(V+E)](https://cs.stackexchange.com/questions/109503/running-time-for-breadth-first-search-vs-depth-first-search#:~:text=The%20running%20time%20of%20both,the%20course%20of%20your%20study).). I was given a class declaration in the header file, which was implemented in the graph.cpp file. In this assignment the Adjacency Matrix was given and its representation is used for the graph. We assumed that the notes are represented with a unique ID and that the integer values are starting from 0 to match the indexes in the Adjacency Matrix. Test cases were also provided in the main.cpp file. I ran a few tests to ensure that my outcome was as expected. The final implementation of this assignment included to keep track of the visiting node instead of what is generally done as a color change.

In this assignment we were not task at looking at test that look at the running time complexity, but with running the test cases it became apparent that this is a straightforward algorithm.