Roman Vasilyev

Prof Ross

CISP 430

4/8/2024

Graph Stack Implementation and Sketching

//Part 1

Just your code

```
/*
Graph Stack Traversal
Dan Ross
Nov 2017
*/
#include <iostream>
#include <fstream>
using namespace std;

// the graph adjacency matrix
int graph[8][8] = {
0, 1, 1, 0, 0, 0, 0, 0, //A
1, 0, 1, 0, 0, 0, 0, 0, //B
1, 1, 0, 1, 0, 1, 0, 1, //C
0, 0, 1, 0, 1, 0, 0, 0, //D
0, 0, 0, 1, 0, 1, 0, 0, //E
0, 0, 1, 0, 1, 0, 1, 1, //F
0, 0, 0, 0, 0, 1, 0, 0, //G
0, 0, 1, 0, 0, 1, 0, 0 //H
};
//A B C D E F G H

// where I've been
bool visited[] = {false, false, false, false, false, false, false, false};

// the resulting tree. Each node's parent is stored
int tree[] = {-1, -1, -1, -1, -1, -1, -1, -1};


void printnode(int nodelist)
{
  char ch = nodelist + 'A';
cout << ch << endl;
}


void printtree()
{
cout << "\nThe resulting tree:";
cout << "\nNode Parent\n";

for (int i = 0; i < 8; i++)
{
cout << ((char)(i + 'A')) << " " << ((char) (tree[i] + 'A')) << endl;
//Console.WriteLine("{0} {1}", (char)(65 + i), (char)(65 + tree[i]));
}
}


// traverse each nodelist (row in the matrix)
void traverse(int nodelist)
{
```

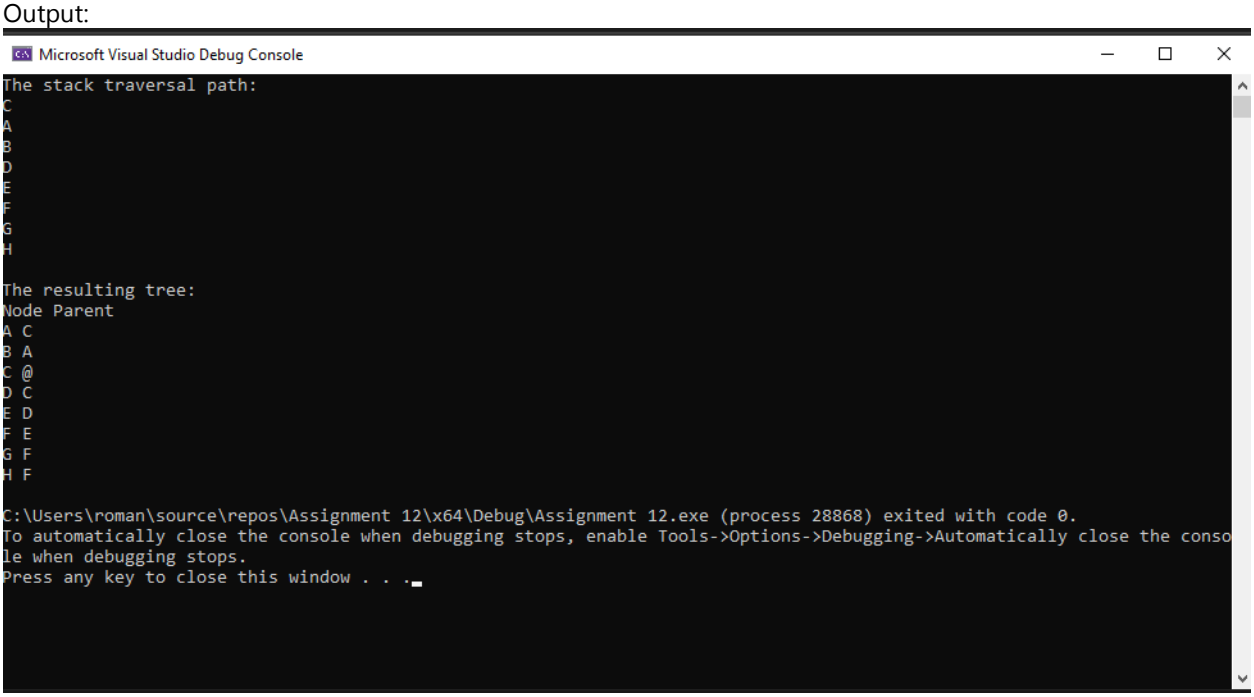
```
visited[nodelist] = true; // been there done that
printnode(nodelist);

// find an unvisted node to select
int i = 0;
while(i < 8)
{
if(!visited[i] && graph[nodelist][i] == 1)
{
tree[i] = nodelist; // who's your daddy?
traverse(i); // "push" this node
}
i++;
}
}


// A function to fill a table from a text file
int fill_O_file()
{
char buffer[10];

// open source file
ifstream fin("graph1.txt");
if (!fin) { cerr << "Input file could not be opened\n"; exit(1); }

// loop through strings in file & spit em' out
int row = 0;
int col = 0;
while (fin >> buffer) {
//cout << buffer << endl;
// parse this row into the table
for (int col = 0; col < 8; col++)
graph[row][col] = buffer[col] - '0';
row++;
}

// close file
fin.close();

}


void main(void)
{
cout << "The stack traversal path:\n";

fill_O_file();

// "Push" C
traverse(2);

printtree();

fill_O_file();
}
```

Part 2

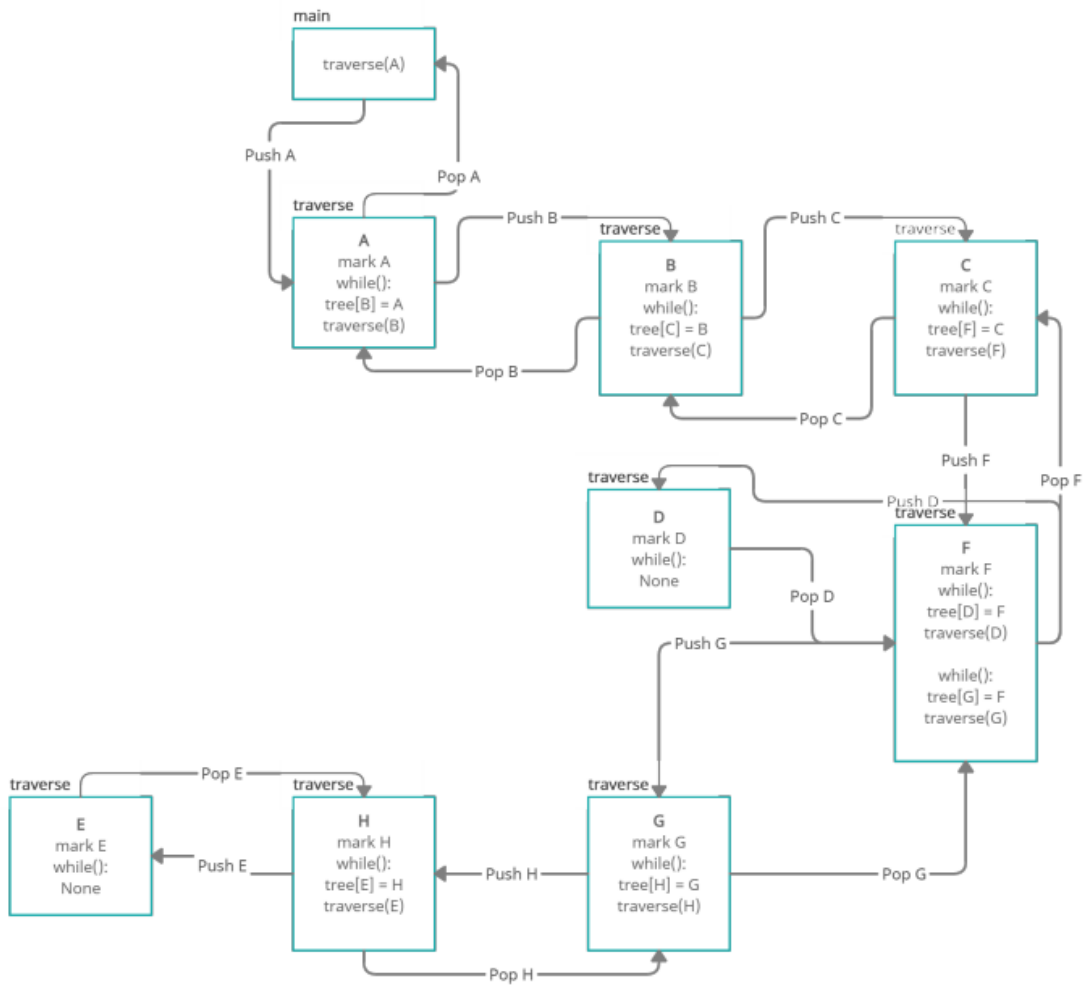
```
#include <iostream>
#include <fstream>

using namespace std;

int graph[8][8] = {
    0, 1, 1, 0, 0, 0, 0, 0,      //A
   1, 0, 1, 0, 0, 0, 0, 0,       //B
   1, 1, 0, 1, 0, 1, 0, 1,       //C
   0, 0, 1, 0, 1, 0, 0, 0,       //D
   0, 0, 0, 1, 0, 1, 0, 0,       //E
   0, 0, 1, 0, 1, 0, 1, 1,       //F
   0, 0, 0, 0, 0, 1, 0, 0,       //G
   0, 0, 1, 0, 0, 1, 0, 0        //H

};

bool visited[] = { false, false, false, false, false, false, false, false };

int tree[] = { -1, -1, -1, -1, -1, -1, -1, -1 };

void printnode(int nodelist) {
   char ch = nodelist + 'A';
   cout << ch << endl;
}

void printtree() {
   cout << "\nThe resulting tree:";
   cout << "\nNode Parent\n";

   for (int i = 0; i < 8; i++) {
      cout << ((char)(i + 'A')) << " " << ((char)(tree[i] + 'A')) << endl;
   }
}

void traverse(int nodelist) {
   visited[nodelist] = true;
   printnode(nodelist);

   int i = 0;
   while (i < 8) {
      if (!visited[i] && graph[nodelist][i] == 1) {
         tree[i] = nodelist;
         traverse(i);
      }
      i++;
   }
}
```

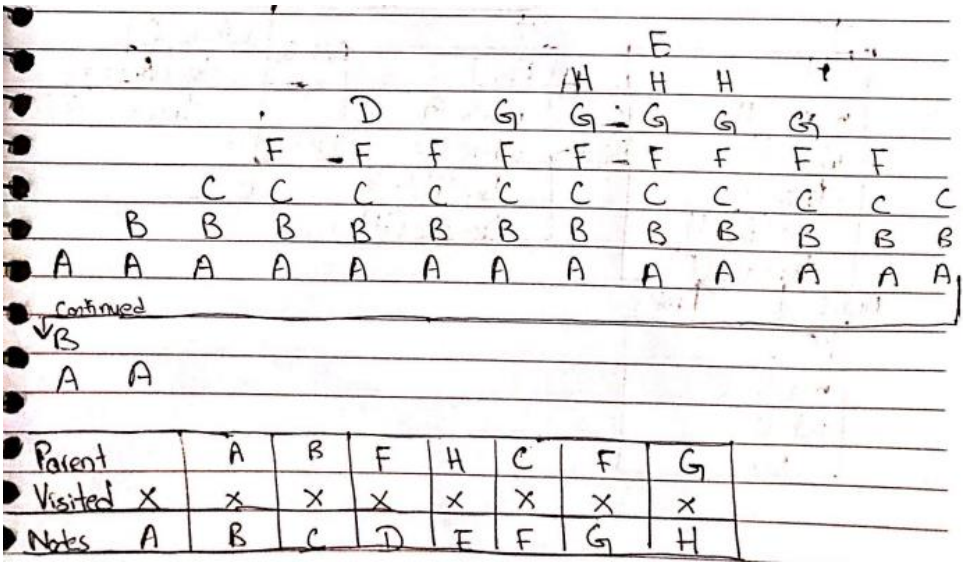
```
int main(void) {
    cout << "The stack traversal path:\n";

    // "Push" C
    traverse(2);

    printtree();

    return 0;
}
```

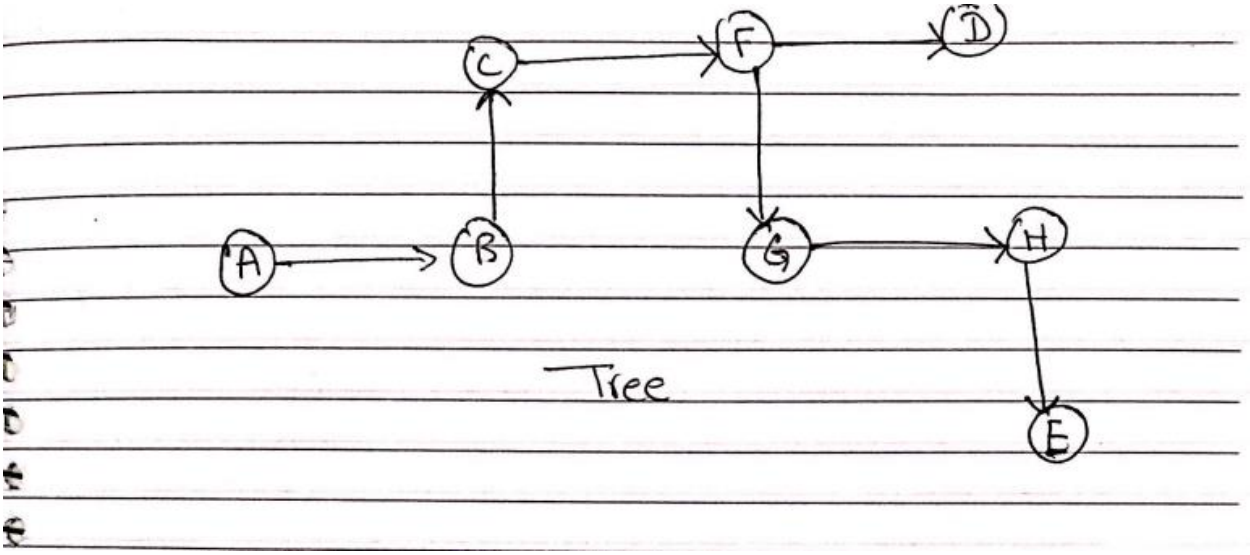
Output:



Part 3


main
traverse(A)
Push A
Pop A
traverse
A
mark A
while():
tree[B] = A
traverse(B)
Push B
Pop B
traverse
B
mark B
while():
tree[C] = B
traverse(C)
Push C
Pop C
traverse
C
mark C
while():
tree[F] = C
traverse(F)
Push F
Pop F
traverse
D
mark D
while():
None
Push D
Pop D
traverse
F
mark F
while():
tree[D] = F
traverse(D)
while():
tree[G] = F
traverse(G)
Push G
Pop G
traverse
G
mark G
while():
tree[H] = G
traverse(H)
Push H
Pop H
traverse
H
mark H
while():
tree[E] = H
traverse(E)
Push E
Pop E
traverse
E
mark E
while():
None


Continued
Parent A B F H C F G
Visited X X X X X X X
Notes A B C D E F G H



Part 4

```
#include <iostream>
#include <string>
#include <fstream>
#include <stack>
#include <vector>
using namespace std;

int counter = 0;
int globalcounter = 0;
int max_counter = 0;
stack<char> global_stack;
stack<char> max_stack;

// the graph adjacency matrix
int graph[100][100] = {};

// where I've been
bool visited[100] = {};

// the resulting tree. Each node's parent is stored
int tree[100] = {};

void printnode(int nodelist) {
   counter++;
   char ch = (nodelist % 26) + 'A'; // Loop back to 'A' if exceeding 'Z'
   cout << ch << " ";
   if (counter % 10 == 0)
      cout << endl;
}

void printtree() {
   cout << "\nThe resulting tree:\n";
```

```
    cout << "Node    Parent\n";
    for (int i = 0; i < 100; i++) {
        if (tree[i] != -1) {
            char node = (i % 26) + 'A';
            char parent = (tree[i] % 26) + 'A';
            cout << node << "       " << parent << endl;
        }
    }
}

void traverse(int nodelist) {
    globalcounter++;
    char ch = nodelist + 'A';
    global_stack.push(ch);
    visited[nodelist] = true; // been there done that
    printnode(nodelist);

    // find an unvisited node to select
    for (int i = 0; i < 100; i++) {
        if (!visited[i] && graph[nodelist][i] == 1) {
            tree[i] = nodelist; // who's your daddy?
            traverse(i);        // "push" this node
        }
    }

    if (globalcounter > max_counter) {
        max_counter = globalcounter;
        max_stack = global_stack;
    }

    globalcounter--;
    global_stack.pop();
}

// A function to fill a table from a text file
void fill_O_file() {
    // Open source file
    ifstream fin("C:/Users/roman/OneDrive/Desktop/BiggieGraph.txt");
    if (!fin) {
        cerr << "Input file could not be opened\n";
        exit(1);
    }

    // Loop through lines in file
    int row = 0;
    string line;
    while (getline(fin, line)) {
        // Copy values into graph
        for (int col = 0; col < line.length(); col++) {
            graph[row][col] = line[col] - '0';
        }
```

```
        row++;
    }

    // Close file
    fin.close();
}

int main() {
    cout << "The stack traversal path:\n";
    fill_O_file();

    // "Push" A
    traverse(0);

    printtree();

    cout << "Sequence of the longest branch: ";
    stack<char> max_stack_copy = max_stack; // Create a copy of max_stack
    while (!max_stack_copy.empty()) {
        char node = max_stack_copy.top();
        char letter = (node % 26) + 'A'; // Convert index to letter
        cout << letter << " ";
        max_stack_copy.pop();
    }

}
```

Output

```
The stack traversal path:
A B E C G D H I K J
O F M Q P L T R U N
W S V A Z B C X Y D
G I F H J E M K N O
R L P Q S T U Z W V
A C X Y B D F E R I
G H M N J K L P S Q
O T V U X Y W A D Z
B E C F G I J K H N
L M O P R Q S V U T

The resulting tree:
Node    Parent
A        A
B        A
C        E
D        G
E        B
F        O
G        C
H        D
I        H
J        K
K        I
L        P
M        F
N        U
O        J
P        Q
Q        M
R        T
S        W
T        L
U        R
V        S
W        N
X        C
Y        X
Z        A
A        V
B        Z
C        B
D        Y
E        J
F        I
G        D
H        F
I        G
J        H
K        M
L        R
M        E
N        K
O        N
P        L
Q        P
R        O
S        Q
T        S
U        T
V        W
W        Z
```

```
A       W
B       Z
C       E
D       A
E       B
F       C
G       F
H       K
I       G
J       I
K       J
L       N
M       L
N       H
O       M
P       O
Q       R
R       P
S       Q
T       R
U       Q
V       S

Length of the longest branch: 98
Sequence of the longest branch: V S Q R P O M L N H K J I G F C E B Z D A W Y X U V T O Q S P L K J N M H G I R E F D B Y X C A V W Z U T S Q P L R O N K M E J H F I G D Y X C B Z A V S W N U R T L P Q M F O J K I H D G C E B A
C:\Users\roman\source\repos\Assignment 12\x64\Debug\Assignment 12.exe (process 32468) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```