Name: Phuong Hoang Tran
Id: 010796179
Date: 02/12/2024

Graph Traversal

(DFS & BFS)

Summary

In this assignment we learn how to implement and learn about the fundamental of graph and tree traversal algorithm, DFS and BFS

Details

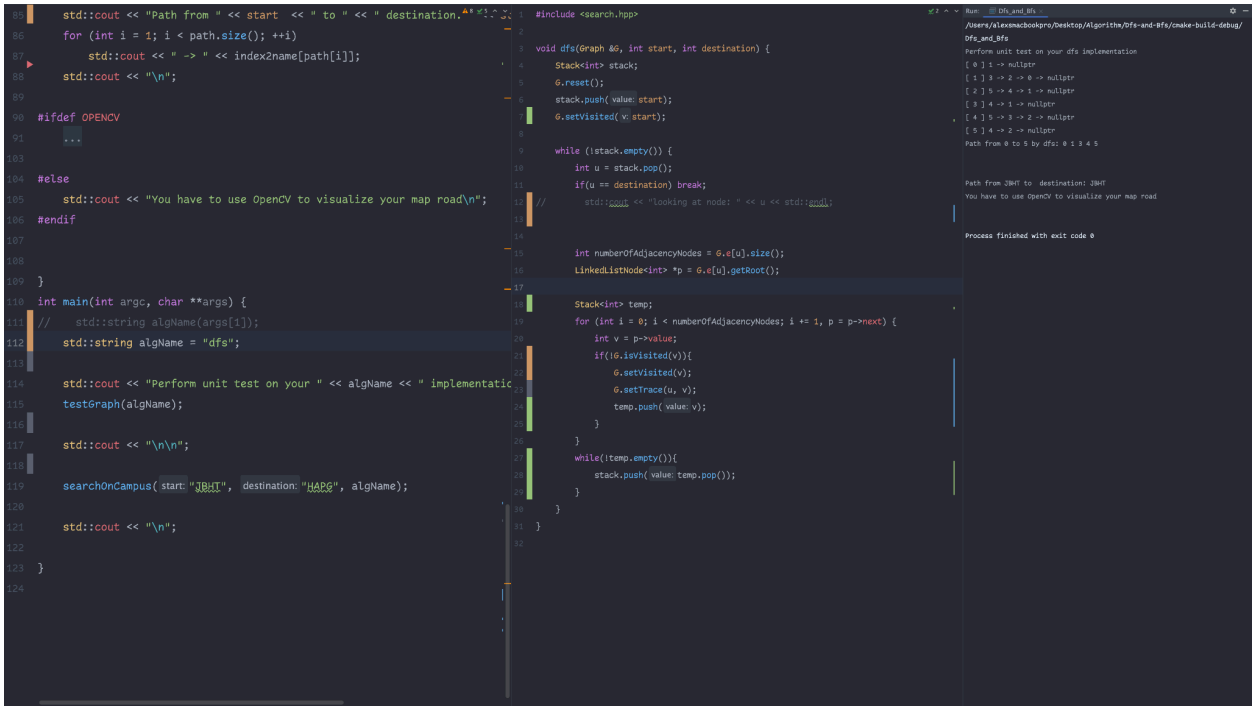
To implement our DFS, BFS and RDFS algorithm, we first need a better test case. Currently the test case doesn't highlight the benefit of using BFS over DFS or RDFS. To allow the search algorithm BFS to find the shortest path (in this context) we must add an additional connection between 2 and 5 inorder to highlight the advantages of BFS. The steps it takes from 0 to 5 should now be 0 -> 1 -> 2 -> 5 which is one step shorter than if I was to use DFS or RDFS to search. The implementation for DFS and BFS is very similar, they both append the starting index to a queue or stack. Both set the starting node to visited before entering the while loop. Both BFS and DFS function iterate through each adjacency node and append the nodes that haven't been visited and mark the appended node as visited. Both utilized the modified version of the setTrace function which prioritizes tracing backward from the destination to the starting point. To make the DFS function behave more like RDFS, I stored all of the current nodes in a temporary array before appending them back to the stack backward, skipping this step will lead to the DFS prioritizing the right end of the adjacency list, I think it's the equivalent of an inorder traversal with a bias toward the right side. To implement RDFS, I first consider the base case that would cause me to return and that was if the starting index is equal to the destination index. Before iterating through all adjacency nodes, I marked the starting or current node as visited and recursively called the RDFS function passing in each adjacency node as the starting index, ensuring that we haven't visited it already.

Design Decision

The reason why I modified the set trace function is because without the modification, with every iteration, the index will be modified and the modified path might not be the right path.

Conclusion

The code works as expected, this is suspiciously easy. I actually got it done this morning but had to double and triple check this evening. Unfortunately I have 3 other assignments due before thursday and a coding assessment due next monday so I will focus on those for now. I will revisit this homework assignment during the weekend.



```
    std::cout << "Path from " << start  << " to " << " destination.";
    for (int i = 1; i < path.size(); ++i)
        std::cout << " -> " << index2name[path[i]];
    std::cout << "\n";

#ifdef OPENCV
    ...

#else
    std::cout << "You have to use OpenCV to visualize your map road\n";
#endif



}
int main(int argc, char **args) {
//    std::string algName(args[1]);
    std::string algName = "dfs";

    std::cout << "Perform unit test on your " << algName << " implementatic
    testGraph(algName);

    std::cout << "\n\n";

    searchOnCampus( start: "JBHT",  destination: "HAPG", algName);

    std::cout << "\n";

}
```

```
#include <search.hpp>

void dfs(Graph &G, int start, int destination) {
    Stack<int> stack;
    G.reset();
    stack.push( value: start);
    G.setVisited( v: start);

    while (!stack.empty()) {
        int u = stack.pop();
        if(u == destination) break;
//        std::cout << "looking at node: " << u << std::endl;


        int numberOfAdjacencyNodes = G.e[u].size();
        LinkedListNode<int> *p = G.e[u].getRoot();

        Stack<int> temp;
        for (int i = 0; i < numberOfAdjacencyNodes; i += 1, p = p->next) {
            int v = p->value;
            if(!G.isVisited(v)){
                G.setVisited(v);
                G.setTrace(u, v);
                temp.push( value: v);
            }
        }
        while(!temp.empty()){
            stack.push( value: temp.pop());
        }
    }
}
```

```
Run:  Dfs_and_Bfs
/Users/alexsmacbookpro/Desktop/Algorithm/Dfs-and-Bfs/cmake-build-debug/
Dfs_and_Bfs
Perform unit test on your dfs implementation
[ 0 ] 1 -> nullptr
[ 1 ] 3 -> 2 -> 0 -> nullptr
[ 2 ] 5 -> 4 -> 1 -> nullptr
[ 3 ] 4 -> 1 -> nullptr
[ 4 ] 5 -> 3 -> 2 -> nullptr
[ 5 ] 4 -> 2 -> nullptr
Path from 0 to 5 by dfs: 0 1 3 4 5


Path from JBHT to  destination: JBHT
You have to use OpenCV to visualize your map road


Process finished with exit code 0
```