Assignment 4
The Circumnavigations of Denver Long

DESIGN DOCUMENT

Zack Traczyk
CSE13S - Spring 2021

Due: May 2nd at 11:59 pm

1 Objective

The program (tsp) calculates the shortest route between a given vertices. The program can optionally read and write to a file.

2 Given

- Header files for stack, path, and graph
- Pseudocode for recursive search

3 Test Harness

Command arguments:

- h : Command line options
- v : Verbose printing; prints all Hamiltonian paths found as well as total number of recursive calls to dfs()
- u : Specifes the graph to be undirected
- i : infile; the input file contaiing the cities and edges of a graph (default should be stdin)
- o : outfile; the output file to print to (default is stdout)

3.1 Parse

When parsing command arguments a set is not needed to store flags, since there is only one main function of the program and inputs slightly adapt the functionality. After program arguments are parsed, the input file needs to be parsed. File inputs are parsed by first grabbing the number of nodes, then iterating that many times to store the following strings as names of the nodes. Next, edges are parsed until the end of the file, overwriting edges that have already been defined. Edges are added to a graph datatype as they are parsed.

After parsing the graph is searched, see section *Execute*, then the memory is freed.

3.2 Execute

Execution starts by calling function DFS (depth first search) in search.c, passing the parsed array of string pointers and the graph.

4 Algorithm implementation

4.1 Stack, Path, Graph

Given the pseudocode stack, path, and graph implementation is straight forward.

4.2 Search

Search.c contains two functions: find adjacent edges given a vertex, and recursively find a Hamiltonian path (DFS).

Adjacent edges are found by finding seeing what edges exist in given row (vertex) in graph.

DFS uses the psedocode in the lab doc. First, a global variable is incremented at the beginning of every DFS function call. If it is the first call, the starting vertex is pushed to the stack. Next adjacent edges are found by calling the adjacent edges function and passing an array to store into and returning the number of vertices found. Then the main recursion part of the program is started.

Either the program is at its base case, the path has hit every node and path length and graph length are equal, or the program is searching through more adjacent edges. When the program is at its base case, it checks to see if the current path is shorter than the current shortest array. If it is, it stores it. Here is where the verbose option would print the path and length. If there are still more adjacent edges, the program iterates through these edges to check if they can be searched. With each edge, the program checks if adding it would result in a longer path than the current longest. If so the edge is skipped, if not then DFS is called recursively where the whole process is repeated. After a DFS call, the searched vertex is popped and mark as unvisited. Pseudocode is in *alg.* (1) on the following page

```
increment calls counter;
if first call then
    add vertex to path according to graph;
end
mark vertex as visited in graph;
edge number = adjacent_edges(graph, vertex to check, array to store in);
if path has hit every node then
    push vertex to path;
    if path is shortest then
        if verbose argument set then
            print path;
        end
        copy shortest path;
    end
    pop vertex from path;
else
    for every adjacent vertex do
        if edge not visited then
            push adjacent vertex to path;
            if if path length is not longer than shortest then
                recursive call to DFS;
            end
            pop adjacent vertex from path;
        end
    end
end
add vertex to path according to graph;
```

Algorithm 1: DFS