Design Document for Assignment 4: The Permutations of Denver Long

Purpose:
The purpose of this assignment is to answer the age old computer science problem, The Traveling Salesman Problem. Imagine you are a salesman that needs to visit a set of cities in the shortest amount of time, returning back to where you started? You would want to find which path that visits all cities and returns back to the beginning has the shortest distance. These paths are called Hamiltonian Paths. In this assignment we will implement 3 ADTs, a Stack, a Path, and a Graph, as well as a Depth First Search algorithm in a main function to figure out which path will be the shortest out of all possible paths.

Implementation:
Graph
graph_create
    creates the graph structure, allocates memory and sets undirected to pos/negative and sets vertices
graph_delete
    deletes graph, frees memory
graph_vertices
    returns number of vertices in graph
graph_add_edge
    adds and edge from i to j with weight k, returns bool if successful
graph_has_edge
    returns bool if there is an edge between imputed i and j
graph_visited
    returns bool if specified vertex has been visited
graph_mark_visited
    mark specified vertex as visited in visited array
graph_mark_unvisited
    mark specified vertex as unvisited in visited array
graph_print
    optional function to help with debugging

Stack
stack_create
    creates a stack, allocates memory for struct and array. Does not allocate anything if it cant allocate for both
stack_delete
    deletes stack by freeing memory
stack_empty
    returns bool if stack is empty
stack_full
    return bool if stack is full
stack_size
    returns stack size using top
stack_push

pushes value onto stack

stack_pop

pops top value from stack

stack_peek

puts top value in stack in specified variable x

stack_print

prints stack to outfile in a specific format

Path

path_create

creates path and underlying stack, allocates memory for both

path_delete

deletes path and stack, frees memory

path_push_vertex

pushes vertex onto path, adds weight to total length of path

path_pop_vertex

pops vertex from path, removes weight from total length of path

path_vertices

returns number of vertices in path

path_length

returns length of path using weights of vertices in path

path_copy

copies path from one path to another

path_print

prints number of vertices and the stack using stack_print

The file that we will create an executable and run will include the main function as well as the dfs function on top of it. The main function has an opt arg, which sees what has been imputed. It reads which file will be used as infile and outfile, and if none are specified use the standard infile and out file. open both the infile and outfile, read the values from infile and store them in their respectful space, like cities go into the cities array, edges go into the graph, and vertices go into the variable vertices. the infile is closed, and then dfs is run.

Depth First Search:

This function will be used as the algorithm to find the shortest path in our graph. It will go through Hamiltonian Paths, and when it finds a shorter one, it stores it as the shortest path. We mark the current vertex as visited and add it to our current path. Now we check if the current path we have is full and there is an edge from the last vertex to the first one, and if it is, add the first path to the end to make a complete loop. We then check to see if it is the shortest path. If it is, change the smallest length to the current path's length, and copy the current path to the shortest path. Remove the first vertex from the end of the path. If the path instead is not full or there is no edge between the latest vertex and the start, we check to see if our current path is too big to be the shortest path anyway. If it is still possible for it to be the smallest path, cycle through neighboring nodes that havent been visited yet and recursively call each of them. After all is checked and done, remove the current vertex from the path.

If h is triggered, print Synopsis, close the outfile and terminate the program. Print the shortest path into the outfile, and then close the outfile. Free all of the allocated memory.

Pseudocode:

DFS:

```
dfs(graph, current vertex, current path, shortest path, cities array, outfile, verbose printing)
    recursive counter +1
    mark v visited
    push v to curr path
    if #vertices in path = #vertices in graph and graph has edge from v to 0
        push 0 vertex to path
        if(verbose printing)
            print path to outfile
        if curr path length is less than smallest length
            smallest length = current path length
            copy path from current to shortest
        temp = 0
        pop top vertex from curr path to temp
    else if length of curr path is less than smallest length
        for w from 0 to number of vertices in G w++
            if there is an edge from v to w in graph
                if w is NOT visited
                    dfs(graph, w, curr path, short path, cities array, outfile, verbose)
    mark v in graph unvisited
    pop vertex from current path into v
```