Assignment 4

Design Document

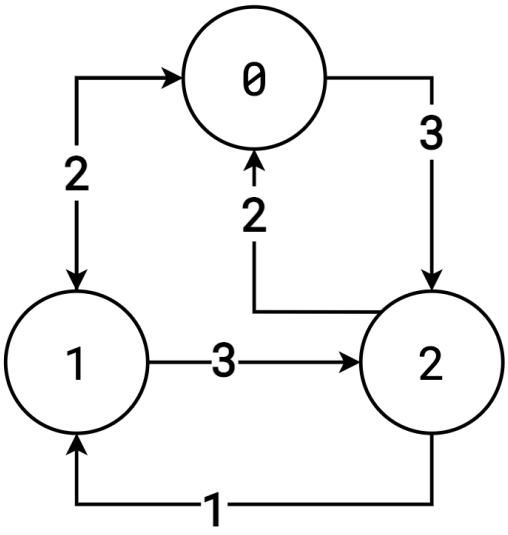
Purpose

This program will use data from an infile to construct an adjacency graph, find all hamiltonian paths using depth-first search, identify the shortest hamiltonian path, and write the path and number of recursive calls to to an outfile.

Layout/Structure

Adjacency Matrix visualization:

```
[ 0  2  3 ]
[ 2  0  3 ]
[ 2  1  0 ]
```



Assistive code/abstract data types:

graph.c	path.c	stack.c
Graph ADT	**Path ADT**	**Stack ADT**

```
int main( ... )
```

Description/Explanation

In this program `main` will parse command-line options, read data from infile, construct graph, and call `dfs`

Pseudocode

```
Parse command-line options
Graph is directed by default
Verbose printing is off by default
The infile/outfile is stdin/stdout by default
For every option given ...
       If h then print help message
       If u then make the graph undirected
       If v then enable verbose printing
       If i then update infile to path provided
       If o then update outfile to path provided
       If other then invalid option so print help message

Get the number of cities (vertices) from the infile
If the first line of the infile could not be read as an integer …
Or if it was an integer greater than 26 …
Or if it was a negative integer …
       Print “Error: malformed number of vertices."
       Exit
If the number of vertices is 2 or 1 …
       Print “There's nowhere to go.”
       Exit

Get the city names from the infile
Create array for storing cities
Create a buffer to temporarily store cities
For the next [vertices] number of lines of the infile …
       Put the city name into buffer
       Copy each city from the buffer to the array of cities
       Remove the newline at the end

Create a graph

Get edges from infile
Read the remaining lines of the infile
       If line can not be read as a triplet of integers …
              Print “malformed edge”
              Exit
       Otherwise …
              Add edge to graph

Create path for tracking current path

Create path for tracking shortest path

Use depth-first search to find the shortest Hamiltonian path – call dfs

Print the results of the depth-first search
If the length of the shortest path is zero ...
       No Hamiltonian path was found
       So print "No Hamiltonian path found."
Otherwise …
       Print the shortest Hamiltonian path to outfile
Print the total number of recursive calls to outfile
```

```
void dfs( ... )
```

Description/Explanation

`dfs` will use recursion to find find all hamiltonian paths. As it goes, it will compute and record path length, compute and record total number of recursive calls, select the shortest path, and write to file or stdout accordingly.

Pseudocode

```
Increment the total number of recursive calls
Mark current vertex v as visited
Push v to the current path
We are now at v and searching from v

If every vertex has been visited once and the origin vertex is reachable ...
	A Hamiltonian can be completed from here
	Push the origin vertex to current path
	Set the current vertex v to the origin vertex
	The current path is now a Hamiltonian path
	If the current Hamiltonian path is shorter than the previous Hamiltonian path ...
	Or if the current Hamiltonian is the first ever Hamiltonian path ...
		A new shortest Hamiltonian path has been found!
		If verbose printing is enabled and there was a previous Hamiltonian path ...
			Print the old Hamiltonian path to the outfile
		Now update the shortest Hamiltonian path
	Now go back two vertices to search for more Hamiltonian paths
	First, pop the origin vertex off the current path
	Second, pop the vertex that lead to the origin vertex
	Mark the vertex that lead to the origin vertex as unvisited
	Go back up the call stack to keep searching for other Hamiltonian paths

Otherwise, for every vertex w that could be accessible from the current vertex v ...
	As long as w is unvisited, w is not the same as v, and there is an edge from v to w ...
		Continue to w if path will still be shorter than shortest Hamiltonian path
		Or if a Hamiltonian path has not been found
			Go search from vertex w (Recursively call dfs from w)

Since all outgoing edges from w have been exhausted ...
Mark v as unvisited
Pop v from the current path
Go back up the call stack
```