DESIGN.pdf ASGN4

Nathan Ong

October 21, 2021

1 Description

This collection of files contains a pathfinder that operates based off of Depth-first Search (DFS). The DFS searches a graph and finds the shortest possible path that visits all of the graph's vertices (Hamiltonian path). The main file used to initiate the pathfinder and its supporting files has the following command line options:

- -h: Prints out help message then exits the program.
- -v: Enables verbose printing; prints out all Hamiltonian paths that were found as well as the total number of recursive calls to dfs().
- -u: Specifies the graph to be undirected.
- -i infile: Specifies the input file path containing the graph, edges, and cities within it. The default input is stdin.
- -o outfile: Specifies the output file path to print the output to. The default output is stdout.

1.1 Sample Input Graph

```
4 /* number of vertices */
Asgard /* names of vertices */
Elysium
Olympus
Shangri-La
0 3 5 /* vertices and edge weights */
3 2 4
2 1 10
1 0 2
```

2 Files Included in the Directory

1. graph.h
   (a) This file is a header file that contains the function syntax for graph.c and the graph ADT.
2. graph.c
   (a) This file contains the source code for the functions that implement the graph ADT.
3. path.h
   (a) This file is a header file that contains the function syntax for graph.c and the path ADT.
4. path.c
   (a) This file contains the source code for the functions that implement the path ADT.

5. stack.h
    (a) This file is a header file that contains the function syntax for stack.c and the stack ADT.
6. stack.c
    (a) This file contains the source code for the functions that implement the stack ADT.
7. tsp.c
    (a) This file contains the source code for the main function that includes the command prompt parser and the DFS algorithm to find the shortest Hamiltonian path through the graph.
8. vertices.h
    (a) This file is a header file that contains the limit of the number of vertices present in a graph and an array that stores the vertices of a given graph.

3 Pseudocode and Structure

3.1 graph.c

1. graph vertices
   return number of vertices

2. graph add edge
   make edge weight between two vertices be number in argument
   if graph is undirected, make vertex at [column][row] the same weight

3. graph has edge
   return true if vertices in bounds and edge isnt 0
   return false if vertices out of bounds or if edge is 0

4. graph edge weight
   return weight of edge between vertices if they are within bounds
   return 0 if vertices not in bounds or no edge

5. graph visited
   return true if vertex is in visited array
   return false if not

6. graph mark visited
   if vertex in bounds, add to visited array

7. graph mark unvisited
   if vertex in bounds, remove from visited array

8. graph print
   print adjacency matrix

3.2 stack.c

1. stack empty
   ```
   return true if top equals 0
   return false if not
   ```

2. stack full
   ```
   return true if top equals maximum capacity
   return false if not
   ```

3. stack size
   ```
   return top value (number of elements currently in stack)
   ```

4. stack push
   ```
   if stack isnt full
   append element to top of stack
   increment top by 1
   return true
   otherwise return false
   ```

5. stack pop
   ```
   if stack isnt empty
   remove element from top of stack
   decrement top by 1
   return true
   otherwise return false
   ```

6. stack peek
   ```
   if stack isnt empty
   return top element of stack
   if not return false
   ```

7. stack copy
   ```
   for all elements in stack
   newstack[n] = stack[n]
   ```

8. stack print
   ```
   print all items in stack
   ```

3.3 path.c

1. path create
   ```
   allocate memory for Path
   create stack for vertices
   length = 0
   return path array
   ```

2. path delete
   ```
   if pointer s and pointer s points to items
   free pointer to items
   free pointer
   ```

```
null pointer
```

3. path push vertex
   ```
   if stack is full
   if pushing vertex to stack is successful
   increment length by edge weight of vertex
   return true
   return false if push is unsuccessful
   ```

4. path pop vertex
   ```
   if popping stack is successful
   get vertex from stack
   decrement edge weight from length
   return true
   if pop unsuccessful return false
   ```

5. path vertices
   ```
   return the vertices in the path
   ```

6. path length
   ```
   return length of path
   ```

7. path copy
   ```
   call stack copy
   make pointers equal
   ```

8. path print
   ```
   call stack print
   print vertices and cities
   ```

3.4 tsp.c

```
while opt isnt -1
    indice through arguments
    check for required arguments if necessary
    execute arguments
execute depth first search algorithm
    label v as visited
    for all edges connected to vertex
        if conncted vertex is labelled as visited
            call DFS algorithm for connected vertex
    label vertex as unvisited
```

4 Additional Credits

1. Sloan's section on October 5th provided the Makefile's format.
2. The code for Stack and Graph create, along with delete, and the data structs were provided in the asgn4.pdf document.
3. The pseudocode for the depth-first search algorithm were provided in the asgn4.pdf document.