Assignment 4
Surfin' USA

By Jiancheng (Jason) Xiong
CSE 13S Spring 2023

Due May 21st 2023, 11:59 PM

1 Purpose

The purpose of this assignment is to create a program that uses stacks and graphs in order to create a path ADT that we will use to solve the traveling salesman problem; i.e finding the shortest path that travels through all points (vertices) and ends up at the beginning point. We will be reading the input information from a text file and output it to another. After the program has run, if the graph given has a hamiltonian cycle the one with shortest total distance will be given, and if not an error message will be printed explaining the lack of possible solutions. We will also create our own makefile in order to compile all necessary files within our project.

2 Using the Program

In order to use the program, compile and run the file tsp.c using makefile. In order to run the program, run ./tsp and specify the commands in which you want to run. The available options are;

-i (name of input file); changes the file read from to the input file given, if not specified input is stdin

-o (name of output file); changes the file read from to the output file given, if not specified input is stdout

-d; changes the edges from undirected to directed.

-h; displays a help message detailing program usage.

3 Program Design

The main program will be in tsp.c, while the separate ADTs will be in

-stack.c and stack.h for the stack ADT

-graph.c and graph.h for the graph ADT

-path.c and path.h for the path ADT

Additional input files can be found in the maps folder.

4 Data Structures

There are 3 main data structures we will be using in this project; that being the stack ADT, the graph ADT, and the path ADT.

The stack will have three properties; capacity, top, and an array of the items. The capacity will be the max number of items the stack can hold, the top will be a pointer to the next available position in the stack, and the array will be an array of every item within the stack. In addition, the stack will have the property of LIFO, and will include helper functions such as stack_push and stack_pop that will enable us to manipulate the stack.

The graph will be created as an n by n adjacency matrix we will use to store the connections between nodes and the weights between them. The graph will store the input vertices it is given, as well as the weights for edges between vertices. It will also store whether the graph is directed or not, as well as if a certain node has been visited previously.

The path will be used in conjunction with the graph ADT to produce and store possible solutions to the traveling salesman problem. The path structure will store the different nodes it has visited as well as the total combined weight of the path it has taken.

We will also be using the DFS search algorithm in order to find all possible paths that the salesman can take, then eliminating paths that do not visit every beach or ends at a beach that does not have a suitable path back to the starting vertex.

In addition, we will be using file operations to read inputs from suitable .graph files, and output the solution to a text file if it exists.

5 Algorithms

Pseudocode for stack:

Stack *stack_create(capacity):

    Sets capacity of the stack to the input capacity
    Sets top of the stack to 0
    Allocates storage for capacity
    Return a pointer to the stack

Void stack_free(stack):

    Free all memory used in the stack
    Set all pointers to NULL

Free memory for the stack

Bool stack_push(stack, val):
Add the val to the top of stack
Increment the top counter
Return true if successful, otherwise return false

Bool stack_pop(stack, val):
Sets val to the top item of the stack
Decrement the top counter
Return true if successful, otherwise return false

Bool stack_peep(stack, val):
Sets val to the top item of the stack
Return true if successful, otherwise return false

Bool stack_full(stack):
Returns true if the number of elements in stack is equal to capacity, otherwise false

Bool stack_empty(stack):
Returns true if the number of elements in stack is 0, otherwise false

Int stack_size(stack):
Returns the number of elements in the stack

Void stack_copy(stack1, stack2):
Copies all properties of stack 2 into stack 1

Void stack_print(stack):
Prints the stack as a list of elements starting with the bottom of the stack

Pseudocode for graph:

Graph *graph_create (int, directed)
Creates an empty 2d array with 0s for the adjacency matrix, and set another 2d array for “visited” with false in all spots.

Allocates memory for the graph
Allocates memory for the visited vertices
Allocates memory for the names
Allocates memory for the weights
Return a pointer to the graph

Void graph_free(graph):
Free all memory used within the graph
Set all pointers within the graph to NULL
Free memory used for the graph
Set the graph pointer to NULL

Int graph_vertices(graph):
return the number of vertices in a graph

Void graph_add_vertex(graph, name, v):
Adds the input name at vertex v, making a copy of the name and storing it in the names object.

Const char* graph_get_vertex_name(graph, v):
Gets the name of the city with vertex v from the array and returns it.

Char **graph_get_names(graph):
Gets the names of every city in an array

Void graph_add_edge(graph, start, end, weight):
Adds an edge between start and end with weight weight to the adjacency matrix of the graph

Int graph_get_weight(graph, start, end):
Returns the weight of the edge between start and end

Void graph_visit_vertex(graph,v):
Adds the vertex v to the list of visited vertices (sets its index to true)

Void graph_unvisit_vertex(graph,v):

Removes the vertex v from the list of visited vertices (sets its index to false)

Bool graph_visited(graph, v):

Returns true if vertex v is visited in graph g, otherwise returns false

Pseudocode for path

Path *path_create(capacity):

Creates a path containing a stack for the vertices and a weight of zero

Void path_free(path):

Frees allocated memory in path
Sets pointers in stack to NULL
Frees allocated memory for path
Sets path pointer to NULL

Int path_vertices(path):

Finds the number of vertices in a path

Int path_distance(path):

Finds the distance covered by a path

Void path_add(path, val, graph):

Adds a vertex val from graph to the path, updating the length of the path

Void path_remove(path, graph):

Removes the most recently added vertex from the path, updating the length of the path

Void path_copy(path1,path2):

Copies the path path2 to path1

Void path_print(path, outputfile, graph):

Prints the path stored using the vertex names from g into the outputfile

6 Function Descriptions

Main function: takes its arguments from the command line, using getopt to know what sorting functions to run. Takes one (command line) input and outputs the optimal solution for the traveling salesman problem (if it exists).

The inputs and outputs of the various functions for the ADTS can be found above in the pseudo code.

7 Results

TBA

8 References

"Official Gnuplot Documentation." *Gnuplot Documentation*, Feb. 2022, www.gnuplot.info/documentation.html.

"Getopt() Function in C to Parse Command Line Arguments." *GeeksforGeeks*, 10 Sept. 2018, www.geeksforgeeks.org/getopt-function-in-c-to-parse-command-line-arguments/.