Assignment 4 – Surfin' USA

Rahul Amudhasagaran

CSE 13S – Spring 2023

Purpose

This program serves to transverse a pre-made graph to find the shortest path to visit all of the graph's nodes.

How to Use the Program

To use this program, the user must run the program and then select an option to run the program with. The options of programs are listed below:

```
Options:
-i: sets the file to read from FILE (default stdin)
-o: sets the file to write to as FILE (default stdout)
-d: initializes the graph as a directed graph (default undirected)
-h: help message
```

All of these options can be run at once.

These descriptions were given by the Lab Document[1].

Program Design

There are 4 major parts of this program: the section that handles the stack data structure, the section that handles the graph data structure, the part that handles path finding, and the input handler.

Data Structures

There are three main data structures in this program. The stack data structure contains the vertices that have been already visited. The path data structure contains the stack and the distance traveled so far. The graph data structure contains all of the nodes that the path and stack data structure use to record their data.

Each of these data structures have their own functions, which are briefly listed in the Function Descriptions section.

Algorithms

This program utilizes one algorithm: Depth First Search (DFS). The pseudo code is provided below:

```
PROTOTYPE:
void dfs (Node n, Graph g):
    n -> visited = 1
    for (e = 0; e < g -> stack size; ++e)
```

[1]Assignment 4 Surfin' USA by Jess Srinivas edited by Kerry Veenstra[1]

```
        if (!e):
        dfs (e, g)
    n -> visited = 0

WITH PATH APPLICATION:
void dfs (Graph *g, Path *p, uint32_t v, Path *best_path)
    graph_visit_vertex (g, v)
    path_add (p, v, g)
    for uint32_t adj: 0 <-> graph_vertices (g)
        if graph_get_weight (g, v, adj) && !graph_visited (g, adj)
            dfs (g, p, adj, best_path)
    if graph_get_weight (g, v, START_VERTEX) && path_vertices (p) == graph_vertices (g)
        path_add(p, START_VERTEX, g)
        if path_distance (p) < path_distance (best_path) || !path_distance (best_path)
            path_copy (best_path, p)
        path_remove (p, g)
    path_remove (p, g)
    graph_unvisit_vertex (g, v)
```

Function Descriptions

There are many functions used in this program. I will sort them out based on which section they are part of.

Stack functions

```
Stack *stack_create (uint32_t capacity)
Inputs: capacity of the stack
Outputs: created stack
Purpose: This function creates a stack.

void stack_free (Stack **sp)
Inputs: pointer to the pointer of the Stack
Outputs: None
Purpose: This function frees the Stack and sets the pointer to NULL.

bool stack_push (Stack *s, uint32_t val)
Inputs: pointer to stack, value that will be pushed
Outputs: boolean possible to run
Purpose: This function pushes item val into the stack. This
function will also return true if it is able to push the item and false otherwise.

bool stack_pop (Stack *s, uint32_t *val)
Inputs: pointer to stack, pointer to value
Outputs: boolean possible to run
Purpose: This function pops the last item from the stack. This function will also return
true if it is able to pop the item and false otherwise.

bool stack_peek (const Stack *s, uint32_t *val)
Inputs: pointer to stack, pointer to value
Outputs: boolean possible to run
Purpose: This function sets val to the value of the last item in the stack. This
function will also return true if it is able to peek and false otherwise.

bool stack_empty (const Stack *s)
```

```
Inputs: pointer to stack
Outputs: boolean empty
Purpose: This function checks if the stack is empty.

bool stack_full (const Stack *s)
Inputs: pointer to stack
Outputs: boolean empty
Purpose: This function checks if the stack is full.

uint32_t stack_size (const Stack *s)
Inputs: pointer to stack
Outputs: int size of stack
Purpose: This function returns the size of the stack.

uint32_t stack_size (const Stack *s)
Inputs: pointer to stack
Outputs: int size of stack
Purpose: This function returns the size of the stack.

void stack_copy (Stack *dst, const Stack *src)
Inputs: two pointers to stacks
Outputs: None
Purpose: This function copies one stack to another.

void stack_print (const Stack* s, FILE *outfile, char *cities[])
Inputs: pointer to stack, output file, array of city names
Outputs: None
Purpose: This function prints out the contents of the stack.
```

Graph Functions

```
Graph *graph_create (uint32_t vertices, bool directed)
Inputs: number of vertices, if graph is directed
Outputs: pointer to created graph
Purpose: This function constructs a graph data structure.

void graph_free (Graph **gp)
Inputs: pointer to the pointer of the graph
Outputs: None
Purpose: This function frees the graph and sets the pointer to NULL.

uint32_t graph_vertices (const Graph *g)
Inputs: pointer to graph
Outputs: number of vertices
Purpose: This function finds the number of vertices.

void graph_add_vertex (Graph *g, const char *name, uint32_t v)
Inputs: pointer to graph, pointer to the name, vertex index
Outputs: None
Purpose: This function adds a vertex to the graph.

const char* graph_get_vertex_name (const Graph *g, uint32_t v)
Inputs: pointer to graph, vertex index
Outputs: the name of the vertex
```

```
Purpose: This function returns the name of the vertex.

char **graph_get_names (const Graph *g)
Inputs: pointer to graph
Outputs: an array of strings
Purpose: This function returns an array of city names.

void graph_add_edge (Graph *g, uint32_t start, uint32_t end, uint32_t weight)
Inputs: pointer to graph, starting node, ending node, weight of edge
Outputs: None
Purpose: This function adds an edge between two vertices with a weight.

uint32_t graph_get_weight (const Graph *g, uint32_t start, uint32_t end)
Inputs: pointer to graph, starting node, ending node
Outputs: weight of edge
Purpose: This function returns the weight of the edge between the two nodes.

void graph_visit_vertex (const Graph *g, uint32_t v)
Inputs: pointer to graph, vertex index
Outputs: None
Purpose: This function marks this vertex as visited.

void graph_unvisit_vertex (Graph *g, uint32_t v)
Inputs: pointer to graph, vertex index
Outputs: None
Purpose: This function marks this vertex as unvisited.

bool graph_visited (Graph *g, uint32_t v)
Inputs: pointer to graph, vertex index
Outputs: if visited
Purpose: This function checks if this vertex has been visited.

void graph_print (const Graph *g)
Inputs: pointer to graph
Outputs: None
Purpose: This function prints the graph.
```

Path Functions

```
Path *path_create (uint32_t capacity)
Inputs: capacity of path
Outputs: Created path
Purpose: This function creates a new Path.

void path_free (Path **pp)
Inputs: pointer to the pointer of the path
Outputs: None
Purpose: This function frees the Path and sets the pointer to NULL.

uint32_t path_vertices (const Path *p)
Inputs: pointer to the path
Outputs: number of vertices in path
Purpose: This function returns the number of vertices in the path.

uint32_t path_distance (const Path *p)
```

```
Inputs: pointer to the path
Outputs: distance of path
Purpose: This function returns the total distance made by a path.

void path_add (Path *p, uint32_t val, const Graph *g)
Inputs: pointer to path, vertex index, pointer to graph
Outputs: None
Purpose: This function adds an index from the graph to the path.

uint32_t path_remove (Path *p, const Graph *g)
Inputs: pointer to path, pointer to graph
Outputs: index of path that was removed
Purpose: This function removes a path from the graph.

void path_copy (Path *dst, const Path *src)
Inputs: pointer to two paths
Outputs: None
Purpose: This function copies one path to another.

void path_print (const Path *p, FILE *outfile, const Graph *g)
Inputs: pointer to path, output file, point to graph
Outputs: None
Purpose: This function prints the path.
```

main Functions

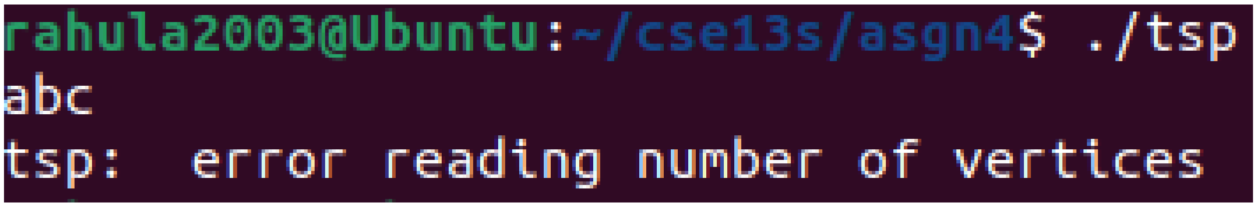
```
int main (int args, char **argv)
Inputs: number of arguments, strings of arguments
Outputs: Exit
Purpose: This function runs the program

void dfs (Graph *g, Path *p, uint32_t v, Path *best_path)
Inputs: graph, current path, vertex number, best path
Outputs: None
Purpose: This function runs the DFS algorithm.

void print_help (File *f)
Inputs: Output file
Outputs: None
Purpose: This function prints out the help function.
```

Results

This program ran with no major issues. It was able to respond to all of the given inputs. This code could have better if some of the surrounding functions had run with better efficiency. My dfs algorithm could have also been improved.

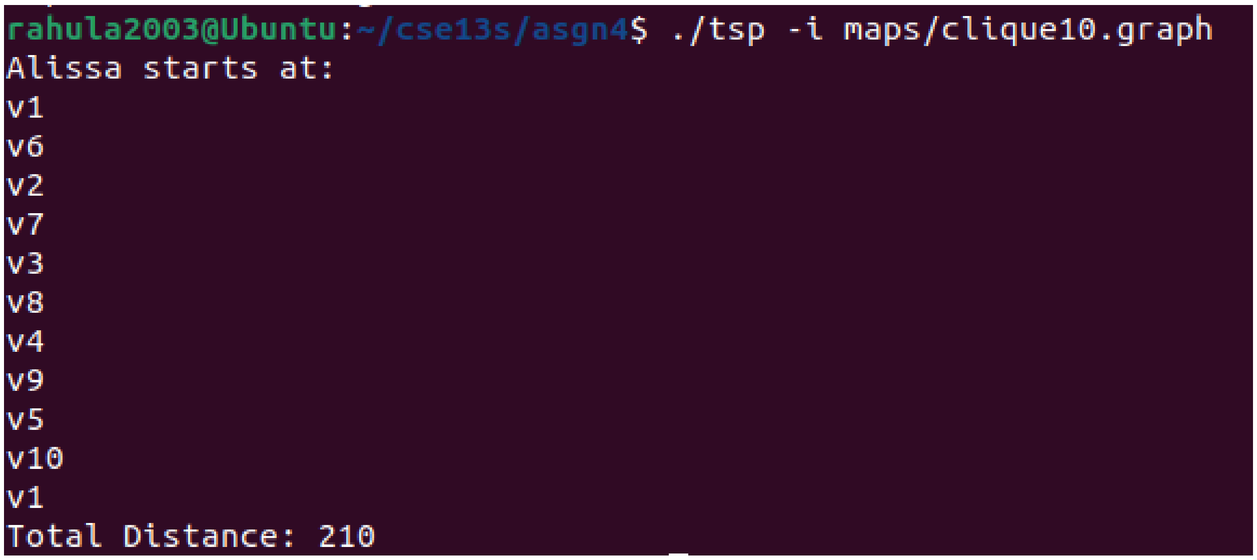
Error Handling

This program is very good at handling errors. There are many cases where the input handler can catch whether the inputs are incorrect. Duplicate options are caught by default in the switch statement and the file input errors are caught by the file input readers themselves.



```
rahula2003@Ubuntu:~/cse13s/asgn4$ ./tsp
abc
tsp:  error reading number of vertices
```

Figure 1: Screenshot of Input Error Handling

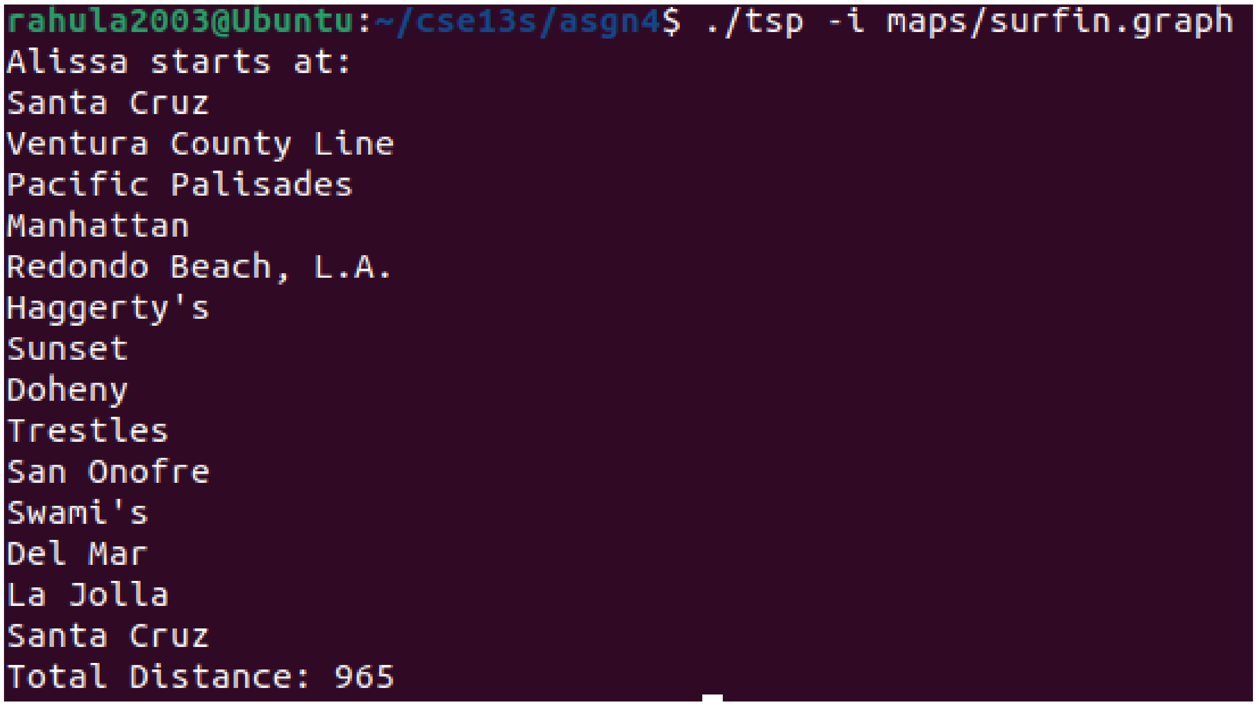


```
rahula2003@Ubuntu:~/cse13s/asgn4$ ./tsp -i maps/clique10.graph
Alissa starts at:
v1
v6
v2
v7
v3
v8
v4
v9
v5
v10
v1
Total Distance: 210
```

Figure 2: Screenshot of Code Running clique10.graph

References

[1] Kerry Veenstra, Jess Srinivas. Assignment 4 surfin' usa. https://git.ucsc.edu/cse13s/spring2023/resources/-/blob/master/asgn4/asgn4.pdf, 2023. [Online; accessed 17-May-2023].



```
rahula2003@Ubuntu:~/cse13s/asgn4$ ./tsp -i maps/surfin.graph
Alissa starts at:
Santa Cruz
Ventura County Line
Pacific Palisades
Manhattan
Redondo Beach, L.A.
Haggerty's
Sunset
Doheny
Trestles
San Onofre
Swami's
Del Mar
La Jolla
Santa Cruz
Total Distance: 965
```

Figure 3: Screenshot of Code Running surfin.graph