# Assignment 4 – Surfin’ USA

Yazmyn Sims

CSE 13S – Spring 2023

## Purpose

This program aims to solve the traveling salesman problem. We will start in Santa Cruz, travel to all the beaches in California and then return back to Santa Cruz. The catch is that you can’t go to any beach more than once (aside from Santa Cruz) and you must find the shortest route.

## How to Use the Program

The user will have to type command line options (-i: sets input file, -o sets output file, -: makes all graphs directed paths, -h: will remind the user how to use the command line options) into vim followed by the graph they want to use. For example user input could look like:

```
./tsp -d -i maps/basic.graph
```

The output will look like this:

```
Alissa starts at:
Home
The beach
Home
Total Distance: 3
```

**If there’s no path then the output will be:**

```
No path found! Alissa is lost!
```

## Program Design

### Data Structures

- Struct: groups several related variables into one place. Like an array that can hold elements of different types.
    - Used at the start of my graph.c, stack.c, and path.c
- Graph: an abstract data type that implements the undirected graph and directed graph concepts from

the field of graph theory within mathematics.
    - Used in graph.c and path.c.
- Stack: a collection of elements, that Push (adds an element) or  Pop (removes the most recently added element)
    - Used in stack.c and path.c
- Path: represents the directory/file relationship
    - Used in path.c

## Algorithms

- Depth-first search algorithm

Pseudocode:

```
def dfs(node n, graph g):
    mark n as visited
    for every one of n's edges:
        if (edge is not visited):
            dfs(edge, g)
    mark n as unvisited
```

## Function Descriptions

Graph Functions

- Graph *graph_create(uint32_t vertices, bool directed)
    - Creates graph struct
    - returns a pointer to it
    - Initialize visited array to false
- void graph_free(Graph **gp)
    - Frees all memory used by graph
    - Sets Graph pointer to NULL
- uint32_t graph_vertices(const Graph *g)
    - Finds number of vertices in graph
- void graph_add_vertex(Graph *g, const char *name, uint32_t v)
    - Gives the city at vertex v the name passed
    - makes a copy of the name
    - stores name in the graph object
    - makes a copy of the string,
    - if we overwrite an existing name, the old one is freed
- const char* graph_get_vertex_name(const Graph *g, uint32_t v)
    - Gets name of city with vertex v from city names' array.
    - returns string stored in Graph
- char **graph_get_names(const Graph *g)
    - Gets names of every city in array
    - Returns a double pointer for an array of strings
- void graph_add_edge(Graph *g, uint32_t start, uint32_t end, uint32_t weight)
    - Adds an edge between start and end with weight to the adjacency matrix
- uint32_t graph_get_weight(const Graph *g, uint32_t start, uint32_t end)
    - Looks up edge weight

    - Returns weight
- void graph_visit_vertex(Graph *g, uint32_t v)
    - Adds vertex v to the list of visited vertices.
- void graph_unvisit_vertex(Graph *g, uint32_t v)
    - Removes vertex v from the list of visited vertices.
- bool graph_visited(Graph *g, uint32_t v) \
    - Returns true if vertex v is visited in graph g
    - Return false otherwise.
- void graph_print(const Graph *g)
    - Prints readable representation of graph
    - Optional

Stack Functions

- Stack *stack_create(uint32_t capacity)
    - Creates a stack
    - dynamically allocates space for the stack
    - Returns a pointer to the stack
- void stack_free(Stack **sp)
    - Frees all space used by stack
    - Sets pointer to NULL
- bool stack_push(Stack *s, uint32_t val)
    - Adds val to the top of stack S
    - Increments counter
    - Returns true if successful
    - Return false if stack is full or operation can't be performed
- bool stack_pop(Stack *s, uint32_t *val)
    - Sets the integer pointed to by val to the last item on the stack
    - removes the last item on the stack
    - Returns true if successful, false otherwise
- bool stack_peek(const Stack *s, uint32_t *val)
    - Sets the integer pointed to by val to the last item on the stack
    - Returns true if successful, false otherwise
- bool stack_empty(const Stack *s)
    - Returns true if the stack is empty
    - Returns false if not empty
- bool stack_full(const Stack *s)
    - Returns true if the stack is full,
    - Returns false if not full
- uint32_t stack_size(const Stack *s)
    - Returns number of elements in stack
- void stack_copy(Stack *dst, const Stack *src)
    - Overwrites dst with all items from src
    - update dst->top
    - Assert that dst has a capacity large enough to store everything from src
- uint32_t stack_print(const Stack* s, FILE *outfile, char *cities[])
    - prints out the stack as a list of elements starting with the bottom

Path Functions

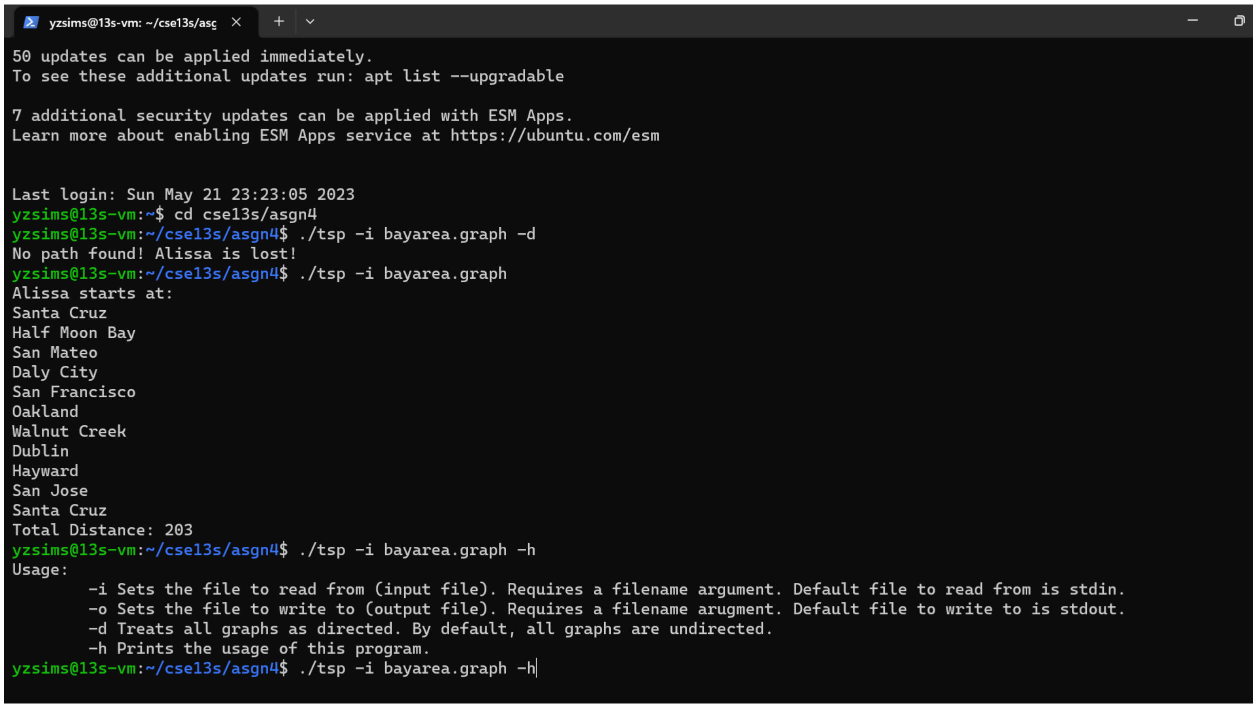
- Path *path_create(uint32_t capacity)
    - Creates path data structure
- void path_free(Path **pp)
    - Frees path and all associated memory
- uint32_t path_vertices(const Path *p)
    - Finds the number of vertices in path
- uint32_t path_distance(const Path *p)
    - Finds distance covered by path
- void path_add(Path *p, uint32_t val, const Graph *g)
    - Adds vertex val from graph g to the path
    - update the distance and length of the path
- uint32_t path_remove(Path *p, const Graph *g)
    - Removes the most recently added vertex from the path.
    - update the distance and length of the path based on the adjacency matrix
- void path_copy(Path *dst, const Path *src)
    - Copies path from src to dst
- void path_print(const Path *p, FILE *outfile, const Graph *g)
    - Prints path to outfile.

## Results

- Everything works as it's supposed to as far as the output. The test cases say I'm missing a newline in path, however, the output is correct and when I fix the newline, everything else fails so I decided to leave it be.

## References

N/A



```
50 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

7 additional security updates can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm


Last login: Sun May 21 23:23:05 2023
yzsims@13s-vm:~$ cd cse13s/asgn4
yzsims@13s-vm:~/cse13s/asgn4$ ./tsp -i bayarea.graph -d
No path found! Alissa is lost!
yzsims@13s-vm:~/cse13s/asgn4$ ./tsp -i bayarea.graph
Alissa starts at:
Santa Cruz
Half Moon Bay
San Mateo
Daly City
San Francisco
Oakland
Walnut Creek
Dublin
Hayward
San Jose
Santa Cruz
Total Distance: 203
yzsims@13s-vm:~/cse13s/asgn4$ ./tsp -i bayarea.graph -h
Usage:
        -i Sets the file to read from (input file). Requires a filename argument. Default file to read from is stdin.
        -o Sets the file to write to (output file). Requires a filename arugment. Default file to write to is stdout.
        -d Treats all graphs as directed. By default, all graphs are undirected.
        -h Prints the usage of this program.
yzsims@13s-vm:~/cse13s/asgn4$ ./tsp -i bayarea.graph -h
```

Figure 1: Shows functional output for bayarea.graph.