Assignment 4 – Surfin' U.S.A.

Mateo Garcia

CSE 13S – Spring 2023

Purpose

The purpose of this program is to simulate the Travelling Salesman Problem. "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?". In our program, Jessie's friend, Alissa, wants to visit all of the places mentioned in the Beach Boys' 1963 song, *Surfin U.S.A.*. The program uses graph theory in order to get Alissa to each city in the shortest distance. It uses an algorithm called Depth First Search in order to find the most efficient/fastest route.

How to Use the Program

First you must go into the cse13s/asgn4 folder where all your files are located. Then run the command **make** and then you can run the main command. By running **./tsp *-[flag1] -[flag2]...*** you can add 4 different flags and they are listed below.

- -i : Sets the file to read from (input file). Requires a filename as an argument. The default file to read from is **stdin**
- -o : Sets the file to write to (output file). Requires a filename as an argument. The default file to write to is **stdout**
- -d : Treats all graphs as *directed*. The default is to assume an *undirected* graph, which means that any edge (i, j) that is specified should be added as both (i, j) and (j, i). So if -d is specified, then (i, j) will be added, but (j, i) won't.
- -h : Prints a help message to **stdio**

Program Design

Data Structures

There are only three big abstract data types used for this assignment. Graphs, Stack, Path. The graphs consists of vertices and edges, you can travel the edges to reach vertices and that is how you can traverse the graph. The graph will be used to map out the locations of cities and the distance between them. The stack is a LIFO data structure and will be used to store information in the program. A path will be used to track the cities that Alissa visits, and it is made of a stack data structure. The program will also use getopt() to pass in the flags.

Algorithms

The main algorithm we will use is Depth First Search.

```
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

These are the functions of the three main ADT's.

This is the stack's constructor:

```
typedef struct stack
    uint32_t capacity;
    uint32_t top;
    uint32_t *items;
```

The stack module will have these functions:

- **Stack *stack_create(uint32_t capacity)** Creates a stack, dynamically allocates space for it, and returns pointer for it. are equal to 0.
- **void stack_free(Stack **sp)** Frees all the space used by the stack given, and sets the pointer to NULL.
- **bool stack_push(Stack *s, uint32_t val)** Adds val to the top of the stack s, and increments the counter (the size value). Returns true if successful, false otherwise. (ex: the stack is full).
- **bool stack_pop(Stack *s, uint32_t *val)** Since val here is a pointer, it sets the integer pointed by val to the last item on the stack s, and removes that item from the stack as well.
- **bool stack_peek(const Stack *s, uint32_t *val)** This function sets the integer pointed by val to the last item on the stack but does not modify the stack. Returns true if successful, false otherwise.
- **bool stack_empty(const Stack *s)** Returns true if the stack is empty, otherwise false.
- **bool stack_full(const Stack *s)** Returns true if the stack is full, otherwise false.
- **uint32_t stack_size(const Stack *s)** Returns the number of elements in stack s.
- **void stack_copy(Stack *dst, const Stack *src)** Stack dst will be overwritten with all the items in Stack src.
- **void stack_print(const Stack* s, FILE *outfile, char *cities[])** This function will print out the list of elements in the stack, starting with the bottom of the stack.

Results

Have not started yet

Error Handling

Numeric results