Assignment 4 - Surfin' U.S.A.

Chucheng Xie

Report Draft

CSE 13S – Spring 2023

Purpose

In this assignment, the task is to use graph theory to get to each city using the smallest amount of gas. The program will then use recursive depth-first searching in order to find the shortest path within the matrix. A path will visit each vertex only once and end with an edge that leads back to the origin vertex. After the search is completed it will print the length of the shortest path, the path itself, and the number of calls to dfs().

Program Design

Pseudocode:

graph.c:

include graph.h, vertices.h, stdbool.h, stdint.h, stdlib.h

uint32_t vertices;
boolean directed;
boolean visited[VERTICES];
uint32_t matrix[VERTICES] [VERTICES];

use calloc to dynamically allocate memory for a graph *g
set the graph's number of vertices to vertices
set the graph's directed boolean to directed
return the graph

free the graph
set the graph to NULL
return

```
return the graph's number of vertices

if i and j are both less than VERTICES:
    location [i] [j] in the graph's matrix = k
    if the graph is directed:
        location [j] [i] in the graph's matrix = k
    return true
else:
    return false

if i and j are both less than VERTICES:
    if graph_add_edge(g, i, j) is false:
        return 0
    else:
        return the weight of the edge at [i] [j] in the matrix
return 0

if the graph has visited v:
    return ture
else:
    retrun false

if v is less than VERTICES:
    mark the graph to have unvisited v
return

print something to debug
return
```

stack.c:

include stack.h, stdbool.h, stdint.h, stdio.h, stdlib.h

```
uint32_t top;
uint32_t capacity;
uint32_t *items;

use malloc to dynamically allocate memory for a stack *s
if s:
    set the top of the stack to 0
    set the stack's capacity to capacity
    use calloc to dynamically allocate memory for stack items
    if there are no items:
        free the stack
        set the stack to NULL
return the stack

if the stack exists and has items:
    free the memory allocated to the items
    free the memory allocated to the stack
    set the pointer to the stack to NULL
return

if the top of the stack is 0:
    return true
else:
    return false

if the top of the stack is at capacity:
    return true
else:
    return false

return the top of the stack

if the top is at capacity:
    return false
```

set the value in the items array at [top] to x
increment the top
return true

if the top is 0:
    return false
decrement the top
dereference x to change the value it points to as the popped item
return true

if the top is at 0:
    return false
dereference x to change the value it points to as the top item - 1
return true

for loop begins with i = 0, iterates while i < capacity, increment i:
    set value in dst items[i] = src items[i]
    set the top of dst to equal the top of src
return

for loop begins with i = 0, iterates while i < top of s, increase i:
    print the city name to outfile
    if the next i is not the top:
        print an -> to outfile
print a newline to outfile
return

path.c:

include path.h, graph.h, stack.h, vertices.h, stdbool.h, stdio.h, stdlib.h

Stack *vertices;
uint32_t length;

use malloc to dynamically allocate memory for a path *p
set vertices as a new stack with VERTICES capacity
set the length of the path to 0
return the path

delete the vertices stack
free the path
set the path to NULL
return the path

create a variable to store the vertex on top of the stack
if there is a vertex on top of the stack:
    increase path length by edge weight between top of stack and v
else:
    increase path length by edge weight between origin and v
if the vertex is successfully pushed:
    return true
else:
    return false

return the size of the path's vertices stack

return the length of the path

set the length of dst equal to the length of src
copy the vertices stack of src to the vertices stack of dst
return

print the first city to outfile
print the vertices stack
return

Result

From this assignment, I learned a lot, such as using the depth-first algorithm to find the shortest path. At the same time, I am more proficient in debugging. If the loop is not written correctly, it can be decomposed one by one for debugging, which is very helpful for us to find and correct errors.

Program simple output: