Dmitrius Agoston
Prof. Darrell Long
24 October 2021

CSE 13S Fall 2021
Assignment 4: Perambulations of Denver Long
Design Document

Description of Program:
The purpose of this program is to solve the traveling salesman problem using depth-first search. Given a graph, the program will attempt to find the fastest path through depth-first search. Starting from the vertex the program will be searching for the shortest Hamiltonian path through a recursive function. The program will create two different paths, one for the current path and one for the shortest. At the end the shortest path will be printed with the length and if commanded will also print all of the other paths that it found.

Layout/Pseudocode:

- graph.c
  This file contains the functions necessary for creating and computing different graphs. It includes a structure named Graph to help keep track of all of the variables used for each graph created. The functions allow for the program to create graphs of desired sizes and dynamically allocate memory for those newly created graphs. There is a function for implementing vertices as well that makes use of the vertices header file. Functions for determining the edges and weight of the graph have also been implemented to ensure that a shortest path can actually be found. Also there are functions that are used to help the search that can mark positions as visited as well as unvisiting positions. The file includes a graph print function as well for debugging purposes.

  Structure Graph
      Variables for vertices, undirected, visited, and matrix

  Define graph create with vertices, undirected
      Allocate memory
      Set vertices equal to vertices
      Set undirected equal to undirected
      Return the graph

  Define graph delete with pointer to pointer of graph
      Free memory dedicated to graph

Set graph equal to NULL

Define graph vertices with graph
Return graph vertices

Define graph add edge with graph, i, j, k
If graph has edge
Set current position to desired position
If undirected is on
Mirror current position on graph
Return true
Return false

Define graph has edge with graph, i, j
If i and j are less than the vertices and the current position is > 0
Return true
Return false

Define graph edge weight with graph, i, j
If the the graph has edge
Return the edge weight
Return 0

Define graph visited with graph, v
Return if position is visited or not

Define graph mark visited with graph, v
If v is less than or equal to the vertices
Current position set to true

Define graph mark unvisited with graph, v
If v is less than or equal to the vertices
Current position set to false

Define graph print with graph
Print out graph for debugging purposes

- stack.c
  This file contains the functions involved in creating the stacks that are used in this program. Much like graph.c there is a structure created named Stack used

for keeping track of variables for each stack like the next open slot, the capacity, and the items in the stack. There are functions centered around the creation and deletion of new stacks that make use of the previously defined structure. There are also functions that give values of the stack like if it is full or not and the actual size of the stack. It also has the essential functions of push and pop to allow the stack to truly function as a stack as well as functions for peeking at the top of the stack and copying the stack as a whole. The file also includes a function to print out the current stack starting from the bottom.

```
Structure Stack
        Variables for top, capacity, and items

Define stack create with capacity
        Stack top set to 0
        Stack capacity set to capacity
        Stack items allocated dynamic memory
        If a stack is not available
                Free the stack
                Set stack equal to NULL
        Return stack

Define stack delete with stack
        If there is a stack with items
                Free the stack
                Set stack equal to NUll
        Return

Define stack empty with stack
        If the stack is empty
                Return true
        Return false

Define stack full with stack
        If the stack if full
                Return true
        Return false

Define stack size with stack
        Return stack size
```

Define stack push with stack, x
    If the stack isn't full
        Set the top of the stack equal to x
        Increment top
        Return true
    Return false

Define stack pop with stack and x
    If the stack isn't empty
        Set x = to the top of the stack
        Empty previous top value
        Decrement top
        Return true
    Return false

Define stack peek with stack, x
    Set x equal to element at top of stack if the stack isn't empty
        Return true
    Return false

Define stack copy with dst, src
    Loop through stack that is being copied
        Set each element equal to equivalent element

Define stack print with stack, outfile, cities
    Loop through stack
        Print each element in stack to outfile
    Print new line

- path.c
  This file contains the functions for creating and keeping track of paths of a graph. Like the other two files a new structure is required for these functions named Path that holds the vertices comprising the path as well as the length of the path. The functions in this file share a lot of similarities with stack.c. There are functions for creation and deletion as well as pushing and popping the vertex of the stack. There are also functions for the length of the vertices and length of the path as well as a copy function. Like the other files there is also a print function that prints out a path to the outfile.

Structure Path
    Variables for vertices and length

Define path create
    Dynamically allocate memory for path
    Set vertices as fresh stack that can hold up to VERTICES
    Initialize length of 0

Define path delete with path
    Free stack path vertices
    Free the path
    Set path equal to null

Define path push vertex with path, v, graph
    If the stack is empty
        Push v onto stack
    If v and previous item on stack have edge
        Push v onto stack
        Length of path increased by edge weight
    Return true if successful, false otherwise

Define path pop vertex with path, v, graph
    Pop from stack
    Decrease path by edge weight
    Return true if successful, false otherwise

Define path vertices with path
    Return number of vertices

Define path length with path
    Return path length

Define path copy with dst, src
    If dst is properly initialized
        Loop through src
            Copy elements of src to dst
            Copy vertices stack and length of source path

Define path print with path, outfile, cities
    Print Path length

Print out current path

- tsp.c
  This file contains the main function of the program as well as the function required to do the depth-first search. There is also a helper function for printing the help message. Along with that this file also is capable of receiving files and reading them to use as potential graphs. It reads the command line options and has choices for verbose printing, help message, undirected graphs, and specifying an infile and outfile. If no file is provided, the user will have to provide the graph through the command line.

  Static int for recursive calls

  Defile help
      Print help message

  Define dfs with graph, v, cur, best, cities, outfile, ver
      Increment recursive count
      Mark v as visited
      Push v to stack
      If current path length is greater than best path length
          Mark v unvisited
          Pop v
          Return
      Check if the paths vertices equals the graphs vertices
          If verbose printing is enabled
              Print current path
          Check if current path is shortest path
              Copy current path to best path
          Pop vertex
      For all edges in v to w in the adjacent edges
          If vertex w is not visited
              Recursively call function
      Mark v as unvisited
      Pop v from stack

  Define main
      Variable for opt

Variable for infile set to stdin
Variable for outfile set to stdout
Booleans for verbose, undirected, and user input
Character array set to max size of line
Set vertices equal to zero
Parse through command line
If h entered
    Print help statement
    End program
If v entered
    Enable verbose printing
If u entered
    Enable undirected graph
If i entered
    If opening the file does not equal null
        Set file to infile
    Else
        Print error and end program
If o entered
    If opening the file does not equal null
        Set file to outfile
    else
        Print error and end program
Get the first line of the file and set vertices equal to it
If the first line is greater than the allowed vertices
    Print error and end program
If first line is equal to 1
    Print nowhere to go and end program
Create graph size of vertices
Create character array size of vertices
For all the spaces in the character array
    Read the next line of the file
    Save it to current position of character array
Read each line of the file until EOF is reached
    Check if the two vertices are bigger than allowed or if weight
    Is less than 0 if so
        Print error and end program
    Add edge to graph based on read line
Close infile
Create path for shortest and current path

Run recursive function
Print total recursive calls and best path
Free all space used for names of vertices
Free memory used for the graph
Free memory used for both paths
End program

Error Handling:
When handling errors for this program, the main errors occurred in regards to user/file input. There were a lot of checks that had to be added to make sure that the graph being used and all elements of the graph were valid. Some of the errors that had to be handled were, failing to open infile, failing to open outfile, malformed number of vertices, not enough vertices to travel, malformed edge, and no hamiltonian path found. Some other errors that had to be handled involved how memory was being allocated. The program has to appropriately allocate memory for certain variables and at the end of running has to free up the space it used for those variables.

Credit:
When creating this code I largely used asgn4.pdf as my main source of reference for creating this program.