

**Instructor: Mr. Streller**

**Programmer: Sai Sao Kham**

## **TABLE OF CONTENTS**

### **CS232\_Project\_2**

#### **User Manual**

**Page 2-4**

#### **Programmer Manuals**

**Page 5-18**

1	class <b>edgeRep</b>	<b>Page-7</b>
2	class <b>vertex</b>	<b>Page-7</b>
3	class <b>isVertex</b>	<b>Page-8</b>
4	class <b>isUniEdge</b>	<b>Page-9</b>
5	class <b>isBiDirEdge</b>	<b>Page-9</b>
6	class <b>AddVertex</b>	<b>Page-10</b>
7	class <b>DeleteVertex</b>	<b>Page-11</b>
8	class <b>AddUniEdge</b>	<b>Page-11</b>
9	class <b>DeleteUniEdge</b>	<b>Page-12</b>
10	class <b>AddBiDirEdge</b>	<b>Page-13</b>
11	class <b>DeleteBiDirEdge</b>	<b>Page-13</b>
12	class <b>SimplePrintGraph</b>	<b>Page-14</b>
13	class <b>ShortestDistance</b>	<b>Page-15</b>
14	class <b>GetGraph</b>	<b>Page-16</b>
15	class <b>BFTraversal</b>	<b>Page-17</b>
16	class <b>DFTraversal</b>	<b>Page-18</b>

#### **Source Code**

## **User Manual**

### **Project 2: GG**

This program will ask user to enter the name of txt file which include not limit to managing travel routes of customer locations with their associated costs of travels for sales people. The sale person will have ability to access the executable file and type in their desire way to travel from multiple choice of cities. There are 3 types of routes in generally i.e. shortest path, DFT, and BFT. The situations involve tracking of whether or not the path sales person intend to use is the shortest path with low costs depending on the cities sales person located. Conveniently, sales person need not to worry where is his/her locations as he/she will receive the shortest routes from the program as long as valid enters of the name of the cities which are in the program in this case.

In addition, the shortest destinations of sales person desire can be updated by user's typing in the name of txt file. Users will know exactly the shortest path and cost base on the routes information provided in txt file.

## 1. Executing the Program

Turn on and boot your computer to the operating system.

Download and locate the file .exe from your source. Double click the P2.exe file to run it. Alternatively, load the file from a DOS prompt by entering the file path along with the filename (P2) and extension (.exe).

Example: C:\P3.exe A console window will appear (or the program will execute in your console if run from DOS) and ask you for input.

## 2. Input

The program will display to user information about routes from txt file regarding of cities to cities. As routes may vary for sale persons to travel, user will input the name of the city where he/she located and input again the name of the city where the sale person desire to end travels. Inputs into the program will calculate the shortest path and travel cost for them.

By default, information which contain cities cost and routes from txt file will be displayed on screen after user's input file name. The user will then be prompt to enter cities name for shortest path while cities from txt file will be displaced as menu and will be asked to input the alphabetical value associated with the menu command. The default menu options are as follow:

1. Shortest path
  - Where you wanna start
  - Where you wanna stop
  - Please input the graph file name (absolute path):
2. DFT

Where do you wanna start?

Please input the graph file name (absolute path):

### 3. DFT

Where do you wanna start?

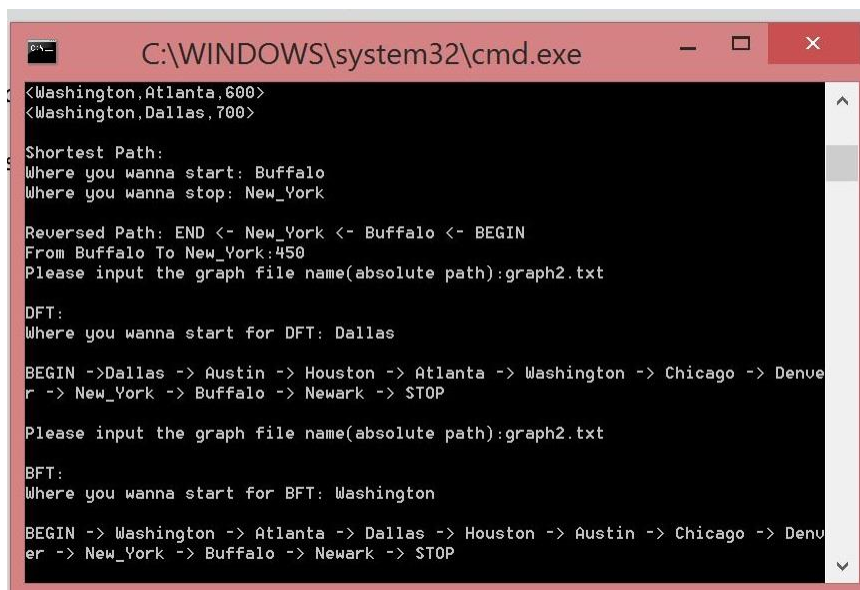
Please input the graph file name (absolute path):

The user will be prompted to enter a selection. After this menu is displayed it will not be displayed again unless the user restart program when prompted for input. The user may enter any city's name or commands as they wish in order to produce a shortest path. The commands may be entered one at a time without spaces. For example, if user enters correct city name it will be print on the screen and ask again for txt file's name otherwise users will need to restart the program again.

### 3. Output

The output of the program will vary depending on the inputs of the user city enter.

For example message will display as in the fig.

A screenshot of a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The window has a black background with white text. The output shows the program's execution flow: it starts with two lines of input, then displays "Shortest Path:" followed by prompts for start and stop cities. It then shows a reversed path and a distance. Next, it prompts for a graph file name, then displays "DFT:" followed by a start city prompt. It then shows a path from BEGIN to STOP. Finally, it prompts for a graph file name again, then displays "BFT:" followed by a start city prompt, and ends with another path from BEGIN to STOP.

```
C:\WINDOWS\system32\cmd.exe
<Washington,Atlanta,600>
<Washington,Dallas,700>

Shortest Path:
Where you wanna start: Buffalo
Where you wanna stop: New_York

Reversed Path: END <- New_York <- Buffalo <- BEGIN
From Buffalo To New_York:450
Please input the graph file name(absolute path):graph2.txt

DFT:
Where you wanna start for DFT: Dallas

BEGIN ->Dallas -> Austin -> Houston -> Atlanta -> Washington -> Chicago -> Denver -> New_York -> Buffalo -> Newark -> STOP

Please input the graph file name(absolute path):graph2.txt

BFT:
Where you wanna start for BFT: Washington

BEGIN -> Washington -> Atlanta -> Dallas -> Houston -> Austin -> Chicago -> Denver -> New_York -> Buffalo -> Newark -> STOP
```

## **Programmer Manual**

### **Project 2: GG**

#### **1. Problem Description**

This program is creating a graph ADT. Graph ADT will consist of a set of vertices and a set of weighted edges. An enhanced user-friendly version of this program could be used for a variety of applications that include: managing travel routes for sales people with vertices representing customer locations and edges representing the travel routes between locations with their associated costs. Managing computer network configurations with vertices representing computer sites and edges representing communication links between computer sites with associated costs. The program uses multiple custom ADTs to accomplish the task of implementing the concept of traveling different cities. As the sales person travels from city to city. More information about the objects used can be found in their respective manuals.

#### **2. Data Types**

The data types used in this program are a combination of basic C and C++ data types as well as 16 class objects. The 16 classes all function some depend and some independently of each other, allowing for modularity of objects. In this program, however, they are used together to create ADT finding the shortest path. Path taken by a sales person will be displayed on screen in this program.

### Class Objects:

- 2.1 class **edgeRep**
- 2.2 class **vertex**
- 2.3 class **isVertex**
- 2.4 class **isUniEdge**
- 2.5 class **isBiDirEdge**
- 2.6 class **AddVertex**
- 2.7 class **DeleteVertex**
- 2.8 class **AddUniEdge**
- 2.9 class **DeleteUniEdge**
- 2.10 class **AddBiDirEdge**
- 2.11 class **DeleteBiDirEdge**
- 2.12 class **SimplePrintGraph**
- 2.13 class **ShortestDistance**
- 2.14 class **GetGraph**
- 2.15 class **BFTraversal**
- 2.16 class **DFTraversal**

## Programmer Manual

### 2.1: Class edgeRep

#### A. Problem Description

This class is an Abstract Data Type (ADT), which represents an array intend to store data which are name and weight of nodes. It is implemented as a dynamic array which can be stored as required information, and allow for resizing of the dimension by changing SIZE, as well as retrieving and changing individual elements (tiles) of the array.

#### B. Data Types

There are only two data types in the **edgeRep** class. These are of template type int and dynamic array.

##### Variables

V name;	- template variable
W weight;	-template variable

### 2.2: Class vertex

#### A. Problem Description

This class is an Abstract Data Type (ADT), which represents an array to store data. It is implemented as a dynamic array which can be stored required information that allow for resizing of the dimension, as well as retrieving and changing individual elements (tiles) of the array.

#### B. Data Types

There are three data types in the vertex class. These are of template type int and dynamic array.

### **Variables**

V name;	- template variable
int visited;	-int data type to store visited nodes
list<edge> edgelist;	-dynamic array to store in edgelist

## **2.3: class isVertex**

### **A. Problem Description**

This class is an Abstract Data Type (ADT), which does calculations to get accurate size for the array and making sure of if the nodes are vertex. It is implemented and variables will be store if nodes follow the rule

### **B. Data Types**

There is only one data types in this **isVertex** class i.e. integer.

### **Variables**

int index = 0;	- A variable to hold as an index for array by using for loop
----------------	--



## 2.4: class **isUniEdge**

### A. Problem Description

This class is an Abstract Data Type (ADT), which represents a direction; and making sure only one direction in the array. It is implemented as a facing integer and an enumeration to correlate the directions with integer.

### B. Data Types

There is only one data type in the **isUniEdge** class besides scratch variables. These are of type int.

#### Variables

int index = 0;      - using index in for loop and vertex function as in the array.

## Programmer Manual

## 2.5: Class **isBiDirEdge**

### A. Problem Description

This class is an Abstract Data Type (ADT), which represents a direction; and making sure both direction in the array pointing. It is implemented as a facing integer and an enumeration to correlate the directions with integer. As reversed of **isUniEdge** class

### B. Data Types

There is only one data type in the Position class besides scratch variables. These are of type int.

## Programmer Manual

### 2.6: Class **AddVertex**

#### A. Problem Description

This class is an Abstract Data Type (ADT), which adding the nodes into the array. This function will loop every single of the array and find the similar name if it is existed in it otherwise the new nodes will be added into the array.

#### B. Data Types

There is one basic integer data type in the **AddVertex** class, which is type int. It also contains three Abstract Data Types.

#### Variables

int index = 0;	-to use as index for the array
vex.name = v;	-assigning into the vex.name of the array
vex.visited = 0;	-tracking down if the nodes are visited or not;

### 2.7: Class **DeleteVertex**

### A. Problem Description

This class is an Abstract Data Type (ADT), which deleting nodes from the array. This function will loop every single set of the array and find the edges if they are connected they the nodes will be deleted from the array.

### B. Data Types

There are basic integer data type in the **DeleteVertex** class, which is type int. It also contains ADT iterator and vertex to delete the nodes from.

#### Variables

int index = 0;	-to use as index for the array
V endName = edg.name;	-using for loop to access to find same vertex name
W endWeight = edg.weight;	-making sure if the weight is work as vertex's deleted.

## 2.8: Class AddUniEdge

### A. Problem Description

This class is an Abstract Data Type (ADT), which adding the edges into the array. This function will loop every single of the array and find the similar name if it is existed in it

otherwise the new nodes will be added into the array. This functions work sort of similar to isUniEdge

## B. Data Types

There is one basic integer data type in the **AddUniEdge** class, which is type int. It also contains three Abstract Data Types.

### Variables

int index = 0;	-to use as index for the array
egInsert.name = v2;	-inserting variable v2
egInsert.weight = wt;	-inserting variable wt

## 2.9: Class DeleteUniEdge

### A. Problem Description

This class is an Abstract Data Type (ADT), which deleting edges but not vertex from the array. This function is using for loop finding the edges to delete it will encounter occurring vertex in the array. However, edges will only be deleted as one direction.

## B. Data Types

There is one basic integer data type in the **DeleteUniEdge** class, which is type int. It also contains three Abstract Data Types.

### Variables

int index = 0;	-to use as index for the array
V endName = edg.name;	- template variable accessing edg.name into endName.
W endWeight = edg.weight;	- template variable accessing edg.weight into endWeight.

## 2.10: Class **AddBiDirEdge**

### A. Problem Description

This class is an Abstract Data Type (ADT), which is adding 2 directions between 2 vertex into the array.

### B. Data Types

There is one basic integer data type in the **AddBiDirEdge** class, which is type int. It also contains three Abstract Data Types.

## 2.11: Class **DeleteBiDirEdge**

### A. Problem Description

This class is an Abstract Data Type (ADT), which is deleting 2 directions between 2 nodes from the array

## B. Data Types

There is one basic integer data type in the **DeleteBiDirEdge** class, which is type int. It also contains three Abstract Data Types.

### Variables

int index = 0;	-to use as index for the array
vex.name = v;	-assigning into the vex.name of the array
vex.visited = 0;	-tracking down if the nodes are visited or not;

## 2.12: Class SimplePrintGraph

### A. Problem Description

This class is an Abstract Data Type (ADT), which is printing vertex from the file by using for loop.

## B. Data Types

There is one basic integer data type in the **SimplePrintGraph** class, which is type int. It also contains three Abstract Data Types.

### Variables

int index = 0;	-to use as index for the array
V endName = edg.name;	-- template variable accessing edg.name into

endName.

W endWeight = edg.weight; -- template variable accessing edg.name into

endWeight..

## 2.13: Class **ShortestDistance**

### A. Problem Description

This class is an Abstract Data Type (ADT), finding shortest distance between vertex.

There are 3 arrays using in this function s[MAX] i.e. to record if the node has been in S set. Dist[Max] i.e to record distance from its node to source node. Prev[MAX] i.e. to record the previous node of ith node in shorted path;

### B. Data Types

There are integer data type in the **ShortestDistance** class, which are type bool s[MAX]; int dist[MAX]; int prev[MAX]; int min; int k;. Of course it also contains Abstract Data Types.

#### Variables

bool s[MAX];            - to record if the node has been in S set;

int dist[MAX];        -to record distance from ith node to source node;

int prev[MAX];	-to record the previous node of ith node in shorted path;
int min;	-to use in the for loop
int k;	-to use in the array of s [k]

## 2.14: Class **GetGraph**

### A. Problem Description

This class is an Abstract Data Type (ADT), which getting vertex from txt file by using ifstream . This function contains vector and will loop every single of the array to print out on scree. User will be asked to enter the name of the file. For example, “graph2.txt” without “”.

### B. Data Types

There is one basic integer data type in the **GetGraph** class, which is type int. It also contains three Abstract Data Types.

#### Variables

string word;	-
string delim(" ");	-using in while loop for ouput purpose.
string textline;	-opening the txt file purpose
int index = 0;	-to use as index for the array
vex.name = v;	-assigning into the vex.name of the array



## 2.15: Class **BFTraversal**

### A. Problem Description

This class is an Abstract Data Type (ADT), which adding the nodes into the array. This function uses loop every single of the array and find the similar name if it is existed in it otherwise the new nodes will be added into the array.

### B. Data Types

There is one basic integer data type in the **BFTraversal** class, which is type int. It also contains three Abstract Data Types.

#### Variables

w.visited = 1;	-marking as visited
G[index] = w;	-printing out vertex as BFT
int index = 0;	-to use as index for the array
vex.name = v;	-assigning into the vex.name of the array
vex.visited = 0;	-tracking down if the nodes are visited or not;

## 2.16: Class **DFTraversal**

### A. Problem Description

This class is an Abstract Data Type (ADT), which accessing the respective nodes from the array. This function use loop every single of the array and printing out vertex as DFT

## B. Data Types

There is one basic integer data type in the **DFTraversal** class, which is type int. It also contains three Abstract Data Types.

### Variables

<code>int index = isVertex(v);</code>	-passing in v into isVertex fun for DFT
<code>w = G[index];</code>	- tracking down into the array for printing out purpose.
<code>int index = 0;</code>	-to use as index for the array
<code>vex.name = v;</code>	-assigning into the vex.name of the array
<code>vex.visited = 0;</code>	-tracking down if the nodes are visited or not;

## C. Limitations and Suggestions

All of the classes of this program could be rewritten to be more robust and dynamic. The executable file will need to run again if user's inputs differ cities name as in txt. The **ShortestDistance** function can be modified as different way for convenient purpose, but most of the classes are sufficient for the purposes of this program. If someone or I were to rewrite this program I would suggest that to modify the **ShortestDistance** function.

```
C:\WINDOWS\system32\cmd.exe

***** Read Map From File *****
Please input the graph file name(absolute path):graph2.txt
```

```
C:\WINDOWS\system32\cmd.exe

***** Read Map From File *****
Please input the graph file name(absolute path):graph2.txt
Print Graph:

<Atlanta,Houston,650>
<Atlanta,Washington,600>
<Austin,Dallas,200>
<Austin,Houston,300>
<Buffalo,New_York,450>
<Buffalo,Newark,500>
<Chicago,Denver,550>
<Chicago,New_York,950>
<Dallas,Austin,200>
<Dallas,Chicago,1500>
<Denver,Atlanta,800>
<Denver,Chicago,550>
<Houston,Atlanta,650>
<New_York,Chicago,950>
<New_York,Buffalo,450>
<Washington,Atlanta,600>
<Washington,Dallas,700>

Shortest Path:
Where you wanna start:
```

```
C:\WINDOWS\system32\cmd.exe

<Atlanta,Houston,650>
<Atlanta,Washington,600>
<Austin,Dallas,200>
<Austin,Houston,300>
<Buffalo,New_York,450>
<Buffalo,Newark,500>
<Chicago,Denver,550>
<Chicago,New_York,950>
<Dallas,Austin,200>
<Dallas,Chicago,1500>
<Denver,Atlanta,800>
<Denver,Chicago,550>
<Houston,Atlanta,650>
<New_York,Chicago,950>
<New_York,Buffalo,450>
<Washington,Atlanta,600>
<Washington,Dallas,700>

Shortest Path:
Where you wanna start: Buffalo
Where you wanna stop: New_York

Reversed Path: END <- New_York <- Buffalo <- BEGIN
From Buffalo To New_York:450
Please input the graph file name(absolute path):
```

```
C:\WINDOWS\system32\cmd.exe

<Dallas,Austin,200>
<Dallas,Chicago,1500>
<Denver,Atlanta,800>
<Denver,Chicago,550>
<Houston,Atlanta,650>
<New_York,Chicago,950>
<New_York,Buffalo,450>
<Washington,Atlanta,600>
<Washington,Dallas,700>

Shortest Path:
Where you wanna start: Buffalo
Where you wanna stop: New_York

Reversed Path: END <- New_York <- Buffalo <- BEGIN
From Buffalo To New_York:450
Please input the graph file name(absolute path):graph2.txt

DFT:
Where you wanna start for DFT: Dallas

BEGIN ->Dallas -> Austin -> Houston -> Atlanta -> Washington -> Chicago -> Denver -> New_York -> Buffalo -> Newark -> STOP
Please input the graph file name(absolute path):
```

```
C:\WINDOWS\system32\cmd.exe

<Washington,Atlanta,600>
<Washington,Dallas,700>

Shortest Path:
Where you wanna start: Buffalo
Where you wanna stop: New_York

Reversed Path: END <- New_York <- Buffalo <- BEGIN
From Buffalo To New_York:450
Please input the graph file name(absolute path):graph2.txt

DFT:
Where you wanna start for DFT: Dallas

BEGIN ->Dallas -> Austin -> Houston -> Atlanta -> Washington -> Chicago -> Denver -> New_York -> Buffalo -> Newark -> STOP

Please input the graph file name(absolute path):graph2.txt

BFT:
Where you wanna start for BFT: Washington

BEGIN -> Washington -> Atlanta -> Dallas -> Houston -> Austin -> Chicago -> Denver -> New_York -> Buffalo -> Newark -> STOP
```