Project Report

Group Members:

Laiba Zaheer(18k-0311)

Ruba Waqar(18k-1099)

Introduction:

The project consists of implementation of graph data structure and it’s algorithms along with the GUI.

Problem Statement:

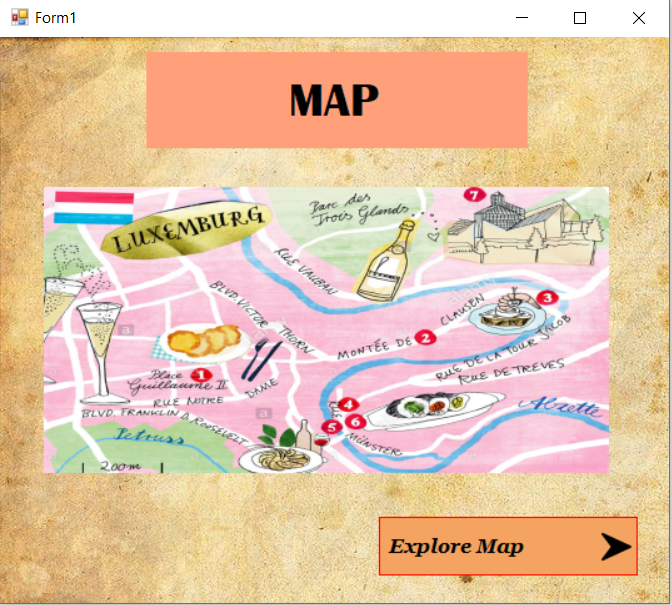
It is often quite difficult to find the route to your destination from your place when the area is new to you. And nowadays everyone wants to pick the shortest route available to spend as less time in the journey as possible. Suppose you are in a new city and you want to visit a nearby restaurant for dinner. In this condition you will definitely require an application which tells you the best nearby restaurant and the shortest route to reach there. Along with that, you’d also like to know the exact distance from your current location to your destination to avoid any problems related to fuel or transport charges.

Working of project:

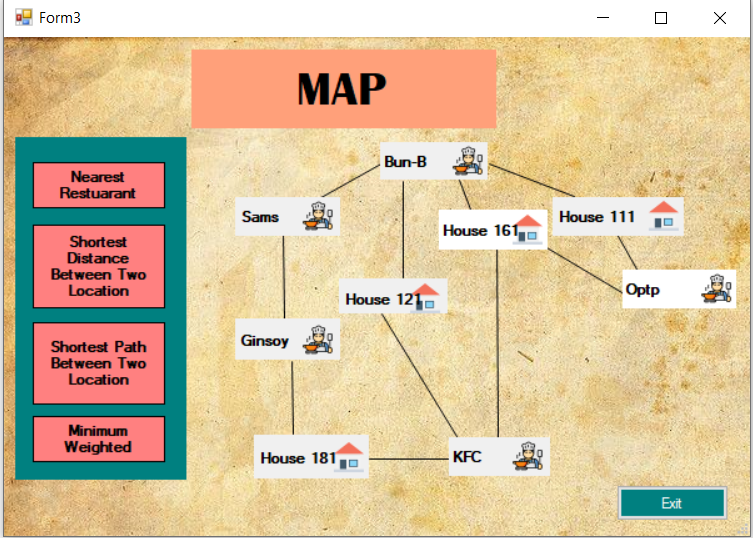
We have tried to make our project as user friendly as possible. Users can enter his/her current and destination locations very easily by just looking at the map and get the desired result in both description form as well as on the map. It is based on dijkstra and kruskal algorithms for shortest distance, path and minimum weight graph. (Minimum weight graph is just an additional functionality to make it more efficient as it eliminates the extra edges and reduces the cost or distance between two vertices) .

To implement graph data structure we have to use an adjacency matrix and in that there are 9 to 10 locations as miss told us to have.Below is the Full flow of the code but **We have also added a video of full working of our project to make it more clear for you.**

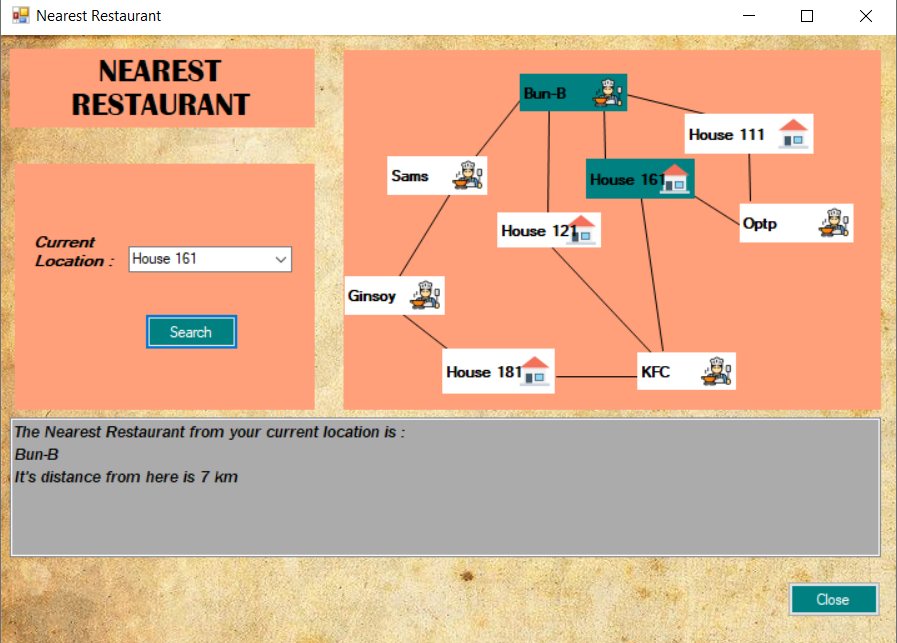
. This is the Starting page of map



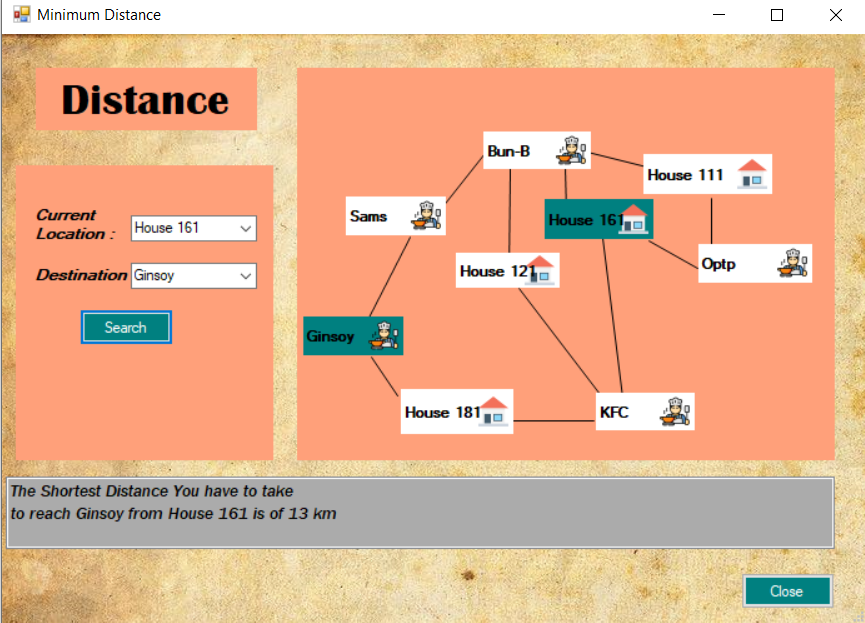
.After clicking Explore Map The Application will direct to this page,here we are showing the MAP or you can say the Graph ,The locations are the Vertices and the lines which are connecting them are Edges beside that you have the options to get the shortest distance,shortest path, nearest Restaurant and to see the Minimum Weighted Graph of This map . In Map we are showing houses and Restaurants .



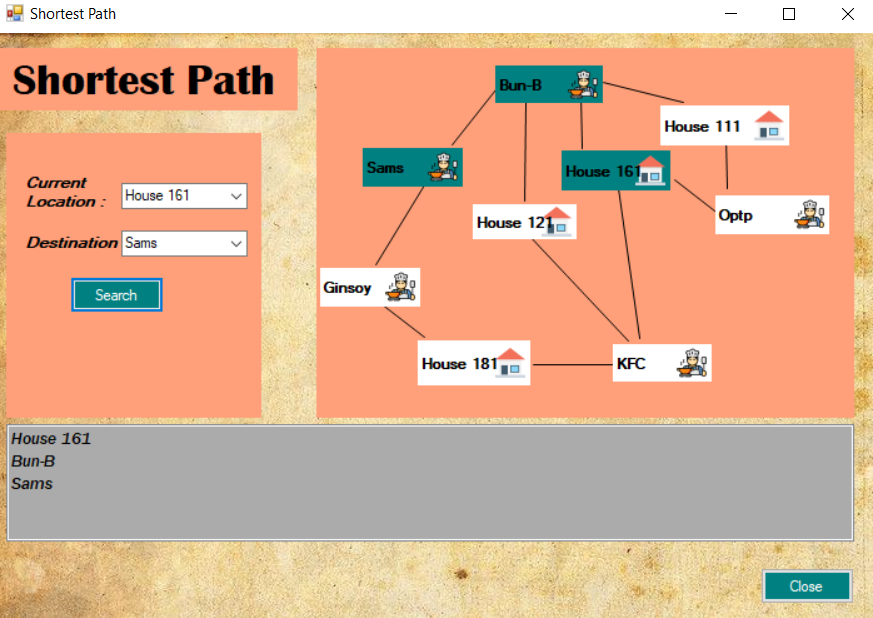
.After Clicking the Nearest Location The new form will appear and their you have to give current location(Which have to House) So that at runtime ,it will tell you the nearest restuarant and it will also show the Restaurant Nearest to you in Map by highlighting the location.We have send video too to show different searches and their result.



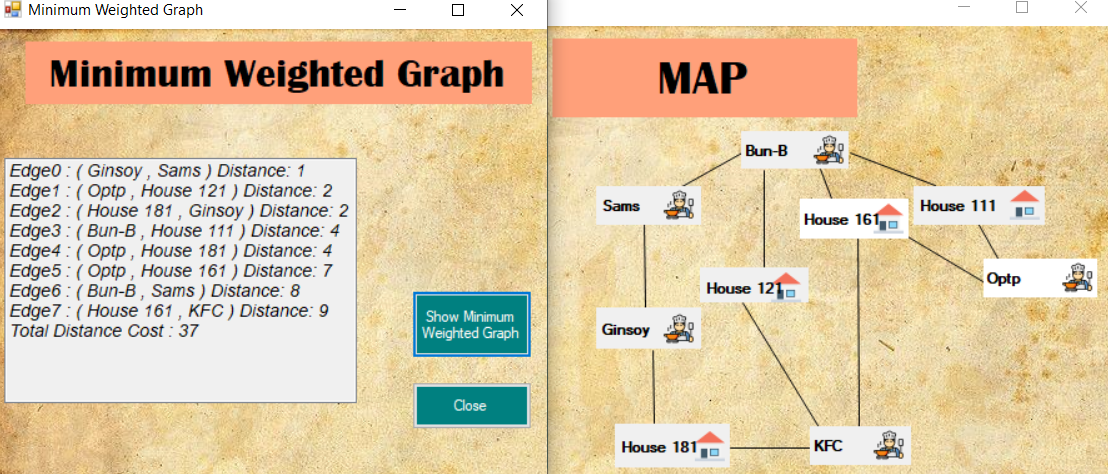
.After Clicking the Shortest Distance Option from Menu the new form will appear, here you have to give the current and destination location and it will tell you the shortest distance even though it has many paths it will choose the shortest path and tells you its distance at runtime .



.After Clicking the Shortest Distance Option from Menu the new form will appear, here you also have to give the current and destination location and it will tell you the shortest route even though it has many paths it will choose the shortest route and tells you at runtime through Map and text.



.After Clicking the Minimum Weighted Graph Option from Menu the new form will appear,here as we can not show this in graph because at runtime we don't know the edges(lines) will come that's why graph can not be made so we show that in text that how much edges and which edge is chosen by displaying its distance between two location.we can compare this with actual graph as you can see that actual graph have 11 edges while minimum weighted graph have 7 .also at last we are telling the total distance by adding all the edges(distance).



Work Division:

We have equally divided our project as it have front-end and back-end both so Shortest distance and Minimum weighted graph is done by laiba and Shortest path and Nearest location is done by ruba .

Conclusion:

With a good GUI and suitable algorithms, we have successfully achieved the project’s goal.

Code:

We have worked on Visual Studio 2010 Windows Form and because of front-end it have a lot of classes and coding so we are showing just the logic many codes of how dynamically graph is changing and showing the location is not here as it will be alot then.

**Form1.h:**

|  |
| --- |
| #pragma once #include "stdAfx.h" #include "Form2.h" #include "Form3.h" #include "time.h" namespace Map {   using namespace System;  using namespace System::ComponentModel;  using namespace System::Collections;  using namespace System::Windows::Forms;  using namespace System::Data;  using namespace System::Drawing;  private: System::Void button2\_Click(System::Object^ sender, System::EventArgs^ e) { this->Hide(); Form3^ f1= gcnew Form3(); f1->ShowDialog(); } }; |

**Form3.h:**

|  |
| --- |
| **#pragma once #include "matrixs.h" #include "Form4.h" #include "Form5.h" #include "Form6.h" #include "Form7.h" namespace Map {   using namespace System;  using namespace System::ComponentModel;  using namespace System::Collections;  using namespace System::Windows::Forms;  using namespace System::Data;  using namespace System::Drawing;   private: System::Void button6\_Click(System::Object^ sender, System::EventArgs^ e) {   // this->Hide();  Form5^ f1= gcnew Form5();  f1->Show();   } private: System::Void button3\_Click\_1(System::Object^ sender, System::EventArgs^ e) {  // this->Hide();  Form6^ f1= gcnew Form6();  f1->ShowDialog();  } private: System::Void close\_Click(System::Object^ sender, System::EventArgs^ e) {  MessageBox::Show("Your Application will now close");  Application::Exit();  } private: System::Void button2\_Click(System::Object^ sender, System::EventArgs^ e) {  Form7^ f=gcnew Form7();  f->ShowDialog();  }};** |

**Form4.h:**

|  |
| --- |
| **#pragma once**  **#include "StdAfx.h"**  **#include "runit.h"**  **#include "Form3.h"**  **#include<String>**  **namespace Map {**  **using namespace System;**  **using namespace System::ComponentModel;**  **using namespace System::Collections;**  **using namespace System::Windows::Forms;**  **using namespace System::Data;**  **using namespace System::Drawing;**  **using namespace std;**  **public: System::Void color(String^m,String ^k){    this->label3->BackColor = System::Drawing::Color::White;  this->label4->BackColor = System::Drawing::Color::White;  this->label10->BackColor = System::Drawing::Color::White;  this->label5->BackColor = System::Drawing::Color::White;  this->label6->BackColor = System::Drawing::Color::White;  this->label11->BackColor = System::Drawing::Color::White;  this->label7->BackColor = System::Drawing::Color::White;  this->label9->BackColor = System::Drawing::Color::White;  this->label8->BackColor = System::Drawing::Color::White;     if(m==label3->Text){  this->label3->BackColor = System::Drawing::Color::Teal;  }  else if(m==label4->Text){  this->label4->BackColor = System::Drawing::Color::Teal;  }  else if(m==label10->Text){  this->label10->BackColor = System::Drawing::Color::Teal;  }  else if(m==label5->Text){  this->label5->BackColor = System::Drawing::Color::Teal;  }  else if(m==label6->Text){  this->label6->BackColor = System::Drawing::Color::Teal;  }  else if(m==label11->Text){  this->label11->BackColor = System::Drawing::Color::Teal;  }  else if(m==label7->Text){  this->label7->BackColor = System::Drawing::Color::Teal;  }  else if(m==label9->Text){  this->label9->BackColor = System::Drawing::Color::Teal;  }  else if(m==label8->Text){  this->label8->BackColor = System::Drawing::Color::Teal;  }  if(k==label3->Text){  this->label3->BackColor = System::Drawing::Color::Teal;  }  else if(k==label4->Text){  this->label4->BackColor = System::Drawing::Color::Teal;  }  else if(k==label10->Text){  this->label10->BackColor = System::Drawing::Color::Teal;  }  else if(k==label5->Text){  this->label5->BackColor = System::Drawing::Color::Teal;  }  else if(k==label6->Text){  this->label6->BackColor = System::Drawing::Color::Teal;  }  else if(k==label11->Text){  this->label11->BackColor = System::Drawing::Color::Teal;  }  else if(k==label7->Text){  this->label7->BackColor = System::Drawing::Color::Teal;  }  else if(k==label9->Text){  this->label9->BackColor = System::Drawing::Color::Teal;  }  else if(k==label8->Text){  this->label8->BackColor = System::Drawing::Color::Teal;  }  }  private: System::Void button2\_Click(System::Object^ sender, System::EventArgs^ e) {     runit^ f1= gcnew runit();  int\* p;    int graph[9][9] = {{0, 4, 0, 0, 0, 0, 0, 8, 0},   {4, 0, 8, 0, 0, 0, 0, 11, 0},   {0, 8, 0, 7, 0, 4, 0, 0, 2},   {0, 0, 7, 0, 9, 14, 0, 0, 0},   {0, 0, 0, 9, 0, 10, 0, 0, 0},   {0, 0, 4, 0, 10, 0, 2, 0, 0},   {0, 0, 0, 14, 0, 2, 0, 1, 6},   {8, 11, 0, 0, 0, 0, 1, 0, 7},   {0, 0, 2, 0, 0, 0, 6, 7, 0},  };    array<String^>^ names = gcnew array<String^>(9);  names[0]="Bun-B";  names[1]="House 111";  names[2]="Optp";  names[3]="House 161";  names[4]="KFC";  names[5]="House 181";  names[6]="Ginsoy";  names[7]="Sams";  names[8]="House 121";    String^ k=this->comboBox1->Text;//current location  String^ m=this->comboBox2->Text;//destination  color(m,k);  if(k=="")  MessageBox::Show("All fields are compulsory");  else if(m=="")  MessageBox::Show("All fields are compulsory");  else if(k==m)  MessageBox::Show("Current and Destination locations cannot be same");   else{  int j;  int h;  for(int i=0;i<9;i++){  if(k==names[i])  {  j=i;  }  if(m==names[i]){  h=i;  }  }  // j->current; h->destination  p=f1->dijkstra\_for\_distance(graph, j);  this->Distance->Items->Clear();  this->Distance->Items->Add("The Shortest Distance You have to take ");  this->Distance->Items->Add("to reach "+ m +" from "+ k +" is of "+ \*(p+h)+" km");   // Form3^ f3=gcnew Form3();  // MessageBox->Show(Form3::label1->Text);  }//end else     }//end function private: System::Void close\_Click(System::Object^ sender, System::EventArgs^ e) {  this->Close();  }};** |

**Form5.h:**

|  |
| --- |
| **#pragma once**  **#include "StdAfx.h"**  **#include "runit.h"**  **#include<String>**  **namespace Map {**  **using namespace System;**  **using namespace System::ComponentModel;**  **using namespace System::Collections;**  **using namespace System::Windows::Forms;**  **using namespace System::Data;**  **using namespace System::Drawing;**  **public: System::Void actualpath(int parent[], int j )  {   if (parent[j]== - 1)   return;    actualpath(parent, parent[j]);   color(names[j]);  this->Distance->Items->Add(names[j]);   }    private: System::Void Form5\_Load(System::Object^ sender, System::EventArgs^ e) {  }   private: System::Void button2\_Click(System::Object^ sender, System::EventArgs^ e) {  this->label3->BackColor = System::Drawing::Color::White;  this->label4->BackColor = System::Drawing::Color::White;  this->label10->BackColor = System::Drawing::Color::White;  this->label5->BackColor = System::Drawing::Color::White;  this->label6->BackColor = System::Drawing::Color::White;  this->label11->BackColor = System::Drawing::Color::White;  this->label7->BackColor = System::Drawing::Color::White;  this->label9->BackColor = System::Drawing::Color::White;  this->label8->BackColor = System::Drawing::Color::White;      runit^ f1= gcnew runit();  int\* p;  int graph[9][9] = {  {0, 4, 0, 0, 0, 0, 0, 8, 10},   {4, 0, 8, 0, 0, 0, 0, 0, 0},   {0, 8, 0, 7, 0, 0, 0, 0, 0},   {8, 0, 7, 0, 9, 0, 0, 0, 0},   {0, 0, 0, 9, 0, 10, 0, 0, 20},   {0, 0, 0, 0, 10, 0, 2, 0, 0},   {0, 0, 0, 0, 0, 2, 0, 1, 0},   {8, 0, 0, 0, 0, 0, 1, 0, 0},   {10, 0, 0, 0, 20, 0, 0, 0, 0},  };    String^ k=this->comboBox1->Text;  String^ m=this->comboBox2->Text;  color(k);  if(k=="")  MessageBox::Show("All fields are compulsory");  else if(m=="")  MessageBox::Show("All fields are compulsory");  else if(k==m)  MessageBox::Show("Current and Destination locations cannot be same");  else{  int j;  int h;  for(int i=0;i<9;i++){  if(k==names[i])  j=i;  if(m==names[i])  h=i;  }  this->Distance->Items->Clear();  this->Distance->Items->Add(names[j]);  p=f1->dijkstra\_for\_path(graph, j);  this->actualpath(p,h);    //while(n != -1){  // this->Distance->Items->Add(n);  // n=p[n];  //}  //this->Distance->Items->Add("The Shortest Distance You have to take ");  //this->Distance->Items->Add("to reach "+ m +" from "+ k +" is of "+ \*(p+h)+" km");   }//end else  }//end funtion private: System::Void close\_Click(System::Object^ sender, System::EventArgs^ e) {  this->Close();  } };** |

**Form6.h:**

|  |
| --- |
| **#pragma once**  **using namespace std;**  **#define V 9**  **#include "StdAfx.h"**  **#include "runit.h"**  **#include<String>**  **namespace Map {**  **using namespace System;**  **using namespace System::ComponentModel;**  **using namespace System::Collections;**  **using namespace System::Windows::Forms;**  **using namespace System::Data;**  **using namespace System::Drawing;**  **/// <summary>**  **/// Summary for Form6**  **/// </summary>**  **public ref class Form6 : public System::Windows::Forms::Form**  **{**  **public:**  **static array<String^>^ names = gcnew array<String^>(9);**  **Form6(void)**  **{**  **InitializeComponent();**  **names[0]="Bun-B";**  **names[1]="House 111";**  **names[2]="Optp";**  **names[3]="House 161";**  **names[4]="KFC";**  **names[5]="House 181";**  **names[6]="Ginsoy";**  **names[7]="Sams";**  **names[8]="House 121";**  **}**  **public: System::Void kruskallogic(int cost[][V]){    int mincost = 0;   int parent[V];   for (int i = 0; i < V; i++)   parent[i] = i;    // Include minimum weight edges one by one   int edge\_count = 0;   while (edge\_count < V - 1) {   int min = 100, a = 0, b = 0;  int i,j;  for (i = 0; i < V; i++) {   for (j = 0; j < V; j++) {     int var1=i;  int var2=j;    while (parent[var1] != var1)   var1 = parent[var1];     while (parent[var2] != var2)   var2 = parent[var2];     if (var1 != var2 && cost[i][j] < min) {   min = cost[i][j];   a = i;   b = j;   }   }   }   int var3=a;  int var4=b;  while (parent[var3] != var3)   var3 = parent[var3];     while (parent[var4] != var4)   var4 = parent[var4];   parent[var3] = var4;   mincost += min;   this->listBox1->Items->Add("Edge"+ (edge\_count) + " : ( "+ names[a] +" , "+names[b]+" )"+ " Distance: "+min);  edge\_count=edge\_count+1;}   this->listBox1->Items->Add("Total Distance Cost : "+ mincost); }   private: System::Void button3\_Click(System::Object^ sender, System::EventArgs^ e) {       int distance[V][V] = {{0, 4, 0, 0, 0, 0, 0, 8, 0},   {4, 0, 8, 0,0,0,0, 11, 0},   {0, 8, 0, 7, 0, 4,0,0, 2},   {0, 0, 7, 0, 9, 14, 0, 0,0},   {0,0,0, 9,0, 10,0,0,0},   {0,0, 4,0, 10,0, 2,0,0},   {0,0,0, 14,0, 2,0, 1, 6},   {8, 11,0,0,0, 0, 1,0, 7},   {0,0, 2,0,0,0, 6, 7,0}   };    // Print the solution   this->listBox1->Items->Clear();  this->kruskallogic(distance); }** |

**Form7.h:**

|  |
| --- |
| **#pragma once**  **#include "StdAfx.h"**  **#include<String>**  **namespace Map {**  **using namespace System;**  **using namespace System::ComponentModel;**  **using namespace System::Collections;**  **using namespace System::Windows::Forms;**  **using namespace System::Data;**  **using namespace System::Drawing;**    **private: System::Void button2\_Click(System::Object^ sender, System::EventArgs^ e) {    int graph[9][9] = {  /\*bun b\*/ {0, 4, 0, 0, 0, 0, 0, 8, 0},   /\*house 111\*/ {4, 0, 8, 0, 0, 0, 0, 11, 0},   /\*optp\*/ {0, 8, 0, 7, 0, 4, 0, 0, 2},   /\*house 161\*/ {0, 0, 7, 0, 9, 14, 0, 0, 0},   /\*kfc\*/ {0, 0, 0, 9, 0, 10, 0, 0, 0},   /\*house 181\*/ {0, 0, 4, 0, 10, 0, 2, 0, 0},   /\*ginsoy\*/ {0, 0, 0, 14, 0, 2, 0, 1, 6},   /\*sams\*/ {8, 11, 0, 0, 0, 0, 1, 0, 7},   /\*house 121\*/ {0, 0, 2, 0, 0, 0, 6, 7, 0},  };   array<String^>^ names = gcnew array<String^>(9);  names[0]="Bun-B";  names[1]="House 111";  names[2]="Optp";  names[3]="House 161";  names[4]="KFC";  names[5]="House 181";  names[6]="Ginsoy";  names[7]="Sams";  names[8]="House 121";    String^ n=this->comboBox3->Text;  if(n=="")  MessageBox::Show("This field is compulsory");  else{  int j;    int r;  String^ d;  for(int i=0;i<9;i++){  if(n==names[i]) //getting the index of current location  j=i;  }  int i;  for( i=1;graph[j][i]==0;i++){  }  int min=graph[j][i];//to get first non zero value  for(i;i<9;i++){  if(graph[j][i]<min){  if(graph[j][i]!=0){  if(names[i][0]!=72){  min=graph[j][i];  r=i;//index of the shortest distance  }//end if  }  }//end if  }//end for    d=names[r];  color(n,d);  this->Nearest\_Location->Items->Clear();  this->Nearest\_Location->Items->Add("The Nearest Restaurant from your current location is : ");  this->Nearest\_Location->Items->Add(d);   this->Nearest\_Location->Items->Add("It's distance from here is "+min+" km");  }//else  }//end function  };** |

**runnit.h:**

|  |
| --- |
| **#pragma once #define V 9  ref class runit { public:    runit(void);  int pak(void);  int runit::minDistance(int dist[],bool sptSet[]);  void runit::printPath(int parent[], int j,int i);  int\* runit::dijkstra\_for\_distance(int graph[V][V], int src) ;  int\* runit::dijkstra\_for\_path(int graph[V][V], int src) ;  int\* runit::matrixsall(int source); };** |

**Runnit.cpp:**

|  |
| --- |
| **#include "StdAfx.h" #include "runit.h" #include <stdio.h>  #include <limits.h>    // Number of vertices  // in the graph  #define V 9  runit::runit(void) {   }   int runit::pak(void) { return 2;  }  int runit::min(int dist[], bool spt[])  {     // Initialize min value   int min = INT\_MAX, min\_index;     for (int v = 0; v < V; v++)   if (spt[v] == false &&   dist[v] <= min)   min = dist[v], min\_index = v;     return min\_index;  }    int\* runit::dijkstra\_for\_distance(int graph[V][V], int src)  {   static int dist[V];   bool spt[V];   int parent[V];    for (int i = 0; i < V; i++)   {   parent[i] = -1;   dist[i] = INT\_MAX;   spt[i] = false;   }   dist[src] = 0;   for (int count = 0; count < V - 1; count++)   {   int u = min(dist, spt);   spt[u] = true;   for (int v = 0; v < V; v++)   if (!spt[v] && graph[u][v] && dist[u] + graph[u][v] < dist[v])   {   parent[v] = u;   dist[v] = dist[u] + graph[u][v];   }   }  return dist;  }    int\* runit::dijkstra\_for\_path(int graph[V][V], int src)  {   static int dist[V];   bool spt[V];   static int parent[V];   for (int i = 0; i < V; i++)   {   parent[i] = -1;   dist[i] = INT\_MAX;   spt[i] = false;   }   dist[src] = 0;   for (int count = 0; count < V - 1; count++)   {   int u = min(dist, spt);   spt[u] = true;   for (int v = 0; v < V; v++)   if (!spt[v] && graph[u][v] && dist[u] + graph[u][v] < dist[v])   {   parent[v] = u;   dist[v] = dist[u] + graph[u][v];   }   }  return parent; }   int\* runit::matrixsall(int source) {  int graph[V][V] = {{0, 4, 0, 0, 0, 0, 0, 8, 0},   {4, 0, 8, 0, 0, 0, 0, 11, 0},   {0, 8, 0, 7, 0, 4, 0, 0, 2},   {0, 0, 7, 0, 9, 14, 0, 0, 0},   {0, 0, 0, 9, 0, 10, 0, 0, 0},   {0, 0, 4, 0, 10, 0, 2, 0, 0},   {0, 0, 0, 14, 0, 2, 0, 1, 6},   {8, 11, 0, 0, 0, 0, 1, 0, 7},   {0, 0, 2, 0, 0, 0, 6, 7, 0},  };    int \*p;  p=this->dijkstra\_for\_path(graph, source);  return p; }** |

**\*\*\*\*\*\* END \*\*\*\*\*\***