**UNIVERSIDAD**

**N**

**ACIONAL**

**AUTONOMA**

**D**

**E M**

**É**

**xico**

**FACULTAD DE ESTUDIOS SUPERIORES**

**PLANTEL ARAGÓN**

**Estructuras Discretas**

:

ALUMNO:

**ZAM**

**ORANO SOLANO GUILLERMO**

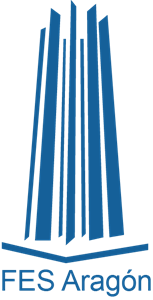
PROFESOR:

ORTIZ CORDERO GABRIEL

NEZAHUALCOYOTL

Estado

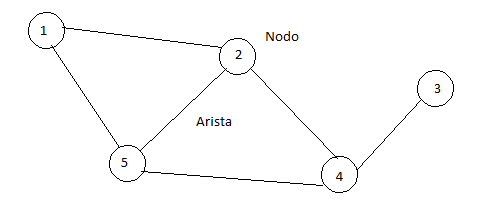
de México.



**Djikstra Routing**

Antes que empezar a describir que es este algoritmo se aclarara que es un grafo :

Un grafo está formado por un conjunto de nodos(o vértices) y un conjunto de arcos. Cada arco en un grafo se especifica por un par de nodos, bueno los nodos ya los hemos visto muchas veces, pero igual es conocido como el vértice en un conjunto de grafos.



El algoritmo de Dijkstra te permite calcular la ruta más corta entre un nodo (tú eliges cuál) y todos los demás nodos en el grafo. También llamado algoritmo de caminos mínimos, es un [algoritmo](http://es.wikipedia.org/wiki/Algoritmo) para la determinación del [camino más corto](http://es.wikipedia.org/wiki/Problema_de_los_caminos_m%C3%A1s_cortos) dado un [vértice](http://es.wikipedia.org/wiki/V%C3%A9rtice_(Teor%C3%ADa_de_grafos)) origen al resto de vértices en un [grafo](http://es.wikipedia.org/wiki/Grafo) con pesos en cada [arista](http://es.wikipedia.org/wiki/Arista_(Teor%C3%ADa_de_grafos)). Su nombre se refiere a [Edsger Dijkstra](http://es.wikipedia.org/wiki/Edsger_Dijkstra" \o "Edsger Dijkstra), quien lo describió por primera vez en 1959.

La Problemática que se intenta resolver en este trabajo esta relacionado con el Mapeo de Redes cual es la mejor Ruta que debe utilizar, que factores influyen en esto.

El algoritmo de Dijkstra proporciona información a los nodos sobre el estado de la red, logrando tomar decisiones de la ruta, a partir de parámetros como la capacidad y el retardo del enlace. En este ejemplo es el cómo se manda un mensaje de un punto a otro utilizando este algoritmo contemplando los costos de cada canal de red que se utiliza:

Imagen que contiene objeto

Descripción generada automáticamente

Supongamos que vas a mandar un mensaje del Nodo A al nodo F, pero no sabes cual es la mejor ruta , no solo por la mas corta si no, la que es menos costosa para nosotros y para lograr esto debemos realizar lo siguiente:

Tenemos que analizar las opciones que tenemos para desplazarnos desde A que sería ir por B o ir a C, entonces vemos los costos de cada uno viendo que si nos vamos a B nos cuesta 2 y si nos vamos a C nos cuesta 1; como buscamos la ruta más económica sabemos que la mejor opción es ir por C.

Repetimos la misma fórmula, pero esta vez como C punto a desplazar que sería para D o E comprobamos los costos y vemos que E es más barato ya que solo vale 1. Finalmente comprobamos el ultimo par de nodos D y F con E y observamos que D es más barata pero como ya llegamos a F en el otro nos desplazamos a ese y terminamos.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| N° de Pasos | Router | Costo/Unión | Costo/Unión | Mas Barata | Costo Total |
| 1 | A | 2A | 1C | AC |  |
| 2 | C | 3D | 1E | ACE |  |
| 3 | E | 1D | 2F | ACEF |  |
|  |  |  | 4 | ACEF | 4 |

Ahora que ya se entendió cómo funciona el algoritmo de Dijkstra aplicado a routing podemos empezar a desarrollar el programa en c++

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <math.h>**

**#include <string.h>**

**typedef int bool;**

**enum {false,**

**true};**

**typedef struct**

**{**

**char name[2];**

**int cost\_from\_start;**

**bool is\_start;**

**bool is\_end;**

**bool visited;**

**char prev\_name[2];**

**char neighbors[100][2];**

**int num\_neighbors;**

**} Node;**

**typedef struct**

**{**

**char name1[2];**

**char name2[2];**

**int cost\_of\_edge;**

**} Edge;**

**bool exister(Node\* arr, int arr\_size, char target[2]);**

**void node\_setter(Node\* n, char name[2], bool start, bool end, bool visited, int inital\_cost, char prev[2]);**

**void edge\_setter(Edge\* e, char name1[2], char name2[2], int edge\_cost);**

**int main()**

**{**

**char choice\_start[2];**

**char choice\_end[2];**

**int num\_visited = 0;**

**int num\_nodes = 0;**

**Node nodes[100];**

**Node last\_obj;**

**int start\_index = -2;**

**int last\_obj\_index = -2;**

**Edge edges[500];**

**Node\* current;**

**printf("Por favor ingresa el nodo Inicial: ");**

**scanf("%s", choice\_start);**

**printf("Por favor ingresa el nodo Final: ");**

**scanf("%s", choice\_end);**

**FILE\* fpointer = fopen("djikstra\_Routing.txt","r");**

**int num\_lines = 0;**

**rewind(fpointer);**

**while(!feof(fpointer))**

**{**

**bool name1\_exists = false;**

**bool name2\_exists = false;**

**char n1[2];**

**char n2[2];**

**int edge\_cost;**

**fscanf(fpointer, "%s %s %d", n1, n2, &edge\_cost);**

**edge\_setter(&edges[num\_lines], n1, n2, edge\_cost);**

**num\_lines++;**

**edge\_setter(&edges[num\_lines], n2, n1, edge\_cost);**

**num\_lines++;**

**name1\_exists = exister(nodes, num\_nodes, n1);**

**name2\_exists = exister(nodes, num\_nodes, n2);**

**if (!name1\_exists)**

**{**

**if (strcmp(choice\_start, n1) == 0)**

**{**

**node\_setter(&nodes[num\_nodes], n1 ,true, false, false, 0, "");**

**}**

**else if (strcmp(choice\_end, n1) == 0)**

**{**

**node\_setter(&nodes[num\_nodes], n1 ,false, true, false, 1000, "");**

**last\_obj\_index = num\_nodes;**

**}**

**else**

**{**

**node\_setter(&nodes[num\_nodes], n1 ,false, false, false, 1000, "");**

**}**

**num\_nodes++;**

**}**

**if (!name2\_exists)**

**{**

**if (strcmp(choice\_start, n2) == 0)**

**{**

**node\_setter(&nodes[num\_nodes], n2 ,true, false, false, 0, "");**

**}**

**else if (strcmp(choice\_end, n2) == 0)**

**{**

**node\_setter(&nodes[num\_nodes], n2 ,false, true, false, 1000, "");**

**last\_obj\_index = num\_nodes;**

**}**

**else**

**{**

**node\_setter(&nodes[num\_nodes], n2 ,false, false, false, 1000, "");**

**}**

**num\_nodes++;**

**}**

**}**

**for(int i = 0; i < num\_nodes; i++)**

**{**

**if(nodes[i].is\_start == true)**

**{**

**start\_index = i;**

**}**

**}**

**rewind(fpointer);**

**char n1[2];**

**char n2[2];**

**while(!feof(fpointer))**

**{**

**Node obj1;**

**Node obj2;**

**int edge\_cost;**

**fscanf(fpointer, "%s %s %d", n1, n2, &edge\_cost);**

**int obj1\_index = -2;**

**int obj2\_index = -2;**

**for (int i = 0; i < num\_nodes; i++)**

**{**

**if (strcmp(nodes[i].name, n1) == 0)**

**{**

**obj1\_index = i;**

**obj1 = nodes[i];**

**}**

**if (strcmp(nodes[i].name, n2) == 0)**

**{**

**obj2\_index = i;**

**obj2 = nodes[i];**

**}**

**}**

**bool has\_neighbor1 = false;**

**bool has\_neighbor2 = false;**

**for(int i = 0; i < obj1.num\_neighbors; i++)**

**{**

**if(strcmp(obj1.neighbors[i], obj2.name) == 0)**

**{**

**has\_neighbor1 = true;**

**}**

**}**

**for(int i = 0; i < obj2.num\_neighbors; i++)**

**{**

**if(strcmp(obj2.neighbors[i], obj1.name) == 0)**

**{**

**has\_neighbor1 = true;**

**}**

**}**

**if(has\_neighbor1 == false)**

**{**

**strcpy(nodes[obj1\_index].neighbors[obj1.num\_neighbors], obj2.name);**

**nodes[obj1\_index].num\_neighbors++;**

**}**

**if(has\_neighbor2 == false)**

**{**

**strcpy(nodes[obj2\_index].neighbors[obj2.num\_neighbors], obj1.name);**

**nodes[obj2\_index].num\_neighbors++;**

**}**

**}**

**fclose(fpointer);**

**Node temp\_val = nodes[0];**

**nodes[0] = nodes[start\_index];**

**nodes[start\_index] = temp\_val;**

**current = &nodes[0];**

**while (num\_visited < num\_nodes)**

**{**

**int temp\_min = 100000;**

**Node\* temp\_min\_node;**

**for (int j = 0; j < (\*current).num\_neighbors; j++)**

**{**

**char current\_neighbor\_name[2];**

**strcpy(current\_neighbor\_name, (\*current).neighbors[j]);**

**int current\_neighbor\_real\_index = -2;**

**for(int i = 0; i < num\_nodes; i++)**

**{**

**if(strcmp(nodes[i].name,(current\_neighbor\_name))== 0)**

**{**

**current\_neighbor\_real\_index = i;**

**}**

**}**

**int curr\_neighbor\_cost = nodes[current\_neighbor\_real\_index].cost\_from\_start;**

**int curr\_node\_cost = (\*current).cost\_from\_start;**

**int edge\_to\_neighbor\_cost;**

**for(int i = 0; i < num\_lines; i++)**

**{**

**if( (strcmp(edges[i].name1, (\*current).name) == 0) && (strcmp(edges[i].name2, current\_neighbor\_name) ==0) )**

**{**

**edge\_to\_neighbor\_cost = edges[i].cost\_of\_edge;**

**}**

**}**

**(\*current).visited = true;**

**if ( curr\_neighbor\_cost > curr\_node\_cost + edge\_to\_neighbor\_cost && ((nodes[current\_neighbor\_real\_index].visited) == false) )**

**{**

**nodes[current\_neighbor\_real\_index].cost\_from\_start = curr\_node\_cost + edge\_to\_neighbor\_cost;**

**strcpy(nodes[current\_neighbor\_real\_index].prev\_name, (\*current).name);**

**}**

**}**

**num\_visited++;**

**for (int i = 1; i < num\_nodes; i++)**

**{**

**if (i != start\_index)**

**{**

**if (nodes[i].cost\_from\_start < temp\_min && (nodes[i].visited == false))**

**{**

**temp\_min = nodes[i].cost\_from\_start;**

**temp\_min\_node = &nodes[i];**

**}**

**}**

**}**

**(current) = temp\_min\_node;**

**start\_index++;**

**}**

**start\_index -= num\_nodes;**

**char str\_result[50];**

**char str\_result\_flipped[50];**

**int str\_result\_max\_index = 0;**

**int result\_cost;**

**for(int i = 0; i < num\_nodes; i++)**

**{**

**if(nodes[i].is\_end)**

**{**

**last\_obj\_index = i;**

**break;**

**}**

**}**

**result\_cost = nodes[last\_obj\_index].cost\_from\_start;**

**if(strcmp(choice\_end, choice\_start)!=0)**

**{**

**char current\_obj\_name[2];**

**Node\* current\_obj = &nodes[last\_obj\_index];**

**Node\* prev;**

**char previous\_name[2];**

**strcpy( previous\_name,(\*current\_obj).prev\_name);**

**for(int i = 0; i < num\_nodes; i++)**

**{**

**if(strcmp(previous\_name,nodes[i].name) == 0)**

**{**

**current\_obj = &nodes[i];**

**}**

**}**

**while( (strcmp((\*current\_obj).name, choice\_start) != 0) )**

**{**

**strcat(str\_result, ">");**

**str\_result\_max\_index++;**

**strcat(str\_result, "-");**

**str\_result\_max\_index++;**

**strcat(str\_result, (\*current\_obj).name);**

**str\_result\_max\_index++;**

**for(int i = 0; i < num\_nodes; i++)**

**{**

**if(strcmp((\*current\_obj).prev\_name,nodes[i].name) == 0)**

**{**

**current\_obj = &nodes[i];**

**break;**

**}**

**}**

**}**

**strcat(str\_result, ">");**

**str\_result\_max\_index++;**

**strcat(str\_result, "-");**

**str\_result\_max\_index++;**

**strcat(str\_result, choice\_start);**

**int index = 0;**

**for(int i = str\_result\_max\_index; i >= 0; i--)**

**{**

**str\_result\_flipped[index] = str\_result[i];**

**index++;**

**}**

**strcat(str\_result\_flipped,nodes[last\_obj\_index].name);**

**}**

**else**

**{**

**result\_cost = 0;**

**strcpy(str\_result\_flipped, choice\_start);**

**strcat(str\_result\_flipped,"->");**

**strcat(str\_result\_flipped, choice\_end);**

**}**

**printf("\n");**

**printf("%s", "Total costo por Routing = ");**

**printf("%d", result\_cost);**

**printf("\n");**

**printf("%s", str\_result\_flipped);**

**printf("\n");**

**return 0;**

**}**

**bool exister(Node\* arr, int arr\_size, char target[2]){**

**bool result = false;**

**for(int i = 0; i < arr\_size; i++)**

**{**

**if(strcmp(arr[i].name,target) == 0)**

**{**

**result = true;**

**}**

**}**

**return result;**

**}**

**void node\_setter(Node\* n, char name[2] ,bool start, bool end, bool visited, int inital\_cost, char prev[2]){**

**strcpy((\*n).name,name);**

**(\*n).is\_start = start;**

**(\*n).is\_end = end;**

**(\*n).visited = visited;**

**(\*n).cost\_from\_start = inital\_cost;**

**strcpy((\*n).prev\_name, prev);**

**}**

**void edge\_setter(Edge\* e, char name1[2], char name2[2], int edge\_cost){**

**strcpy((\*e).name1, name1);**

**strcpy((\*e).name2, name2);**

**(\*e).cost\_of\_edge = edge\_cost;**

**}**

Muy bien ahora que tenemos todo el programa se creó un archivo txt (bloc de notas llamado djikstra\_Routing.txt) donde viene un ejemplo con los nodos del grafo de la red

A B 2

A G 6

B E 2

B C 7

G E 1

E F 2

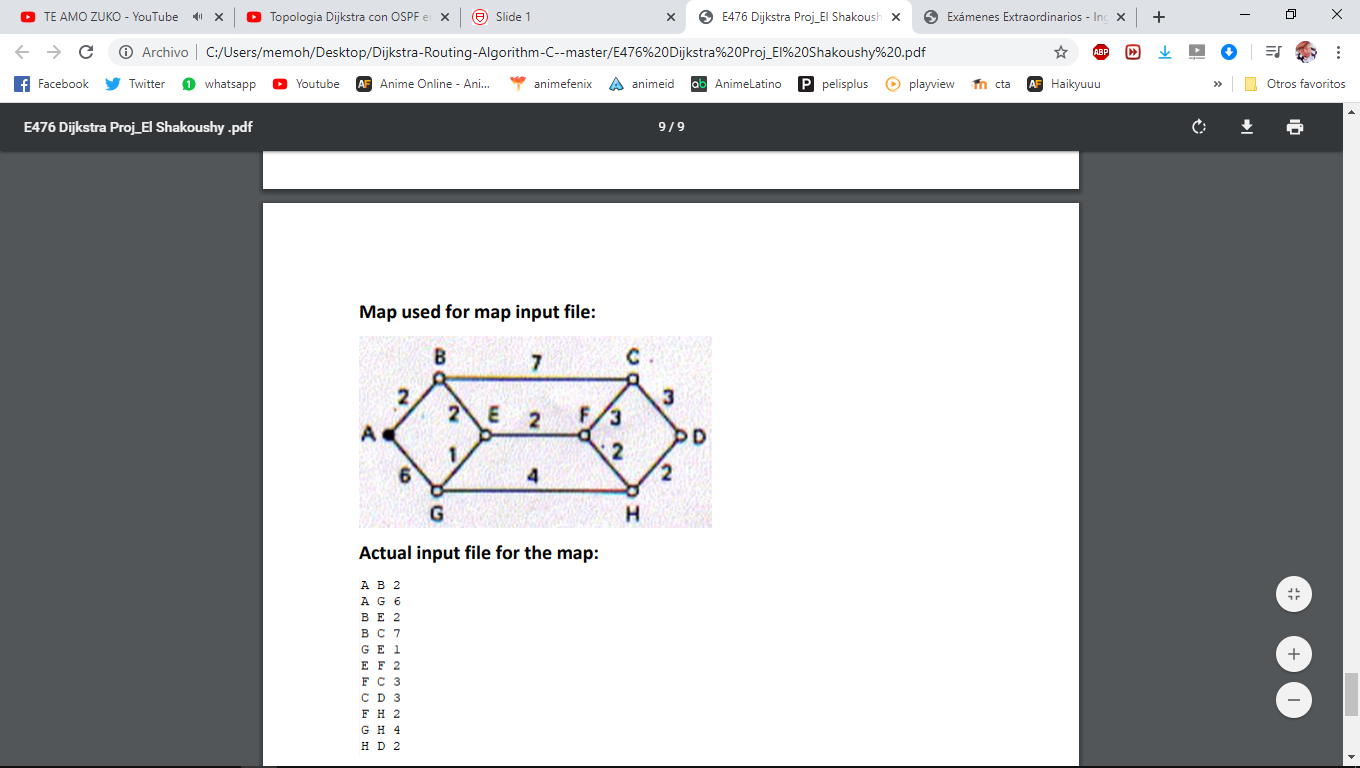
F C 3

C D 3

F H 2

G H 4

H D 2



Son los dos nodos unido-seguidos por el costo de cada uno. Del mismo modo se encuentras otro txt llamado OtrosEjemplos.txt donde está el código del primer ejemplo encontrado arriba

A B 2

A C 1

B D 3

B C 2

C D 3

C E 1

D E 1

E F 2

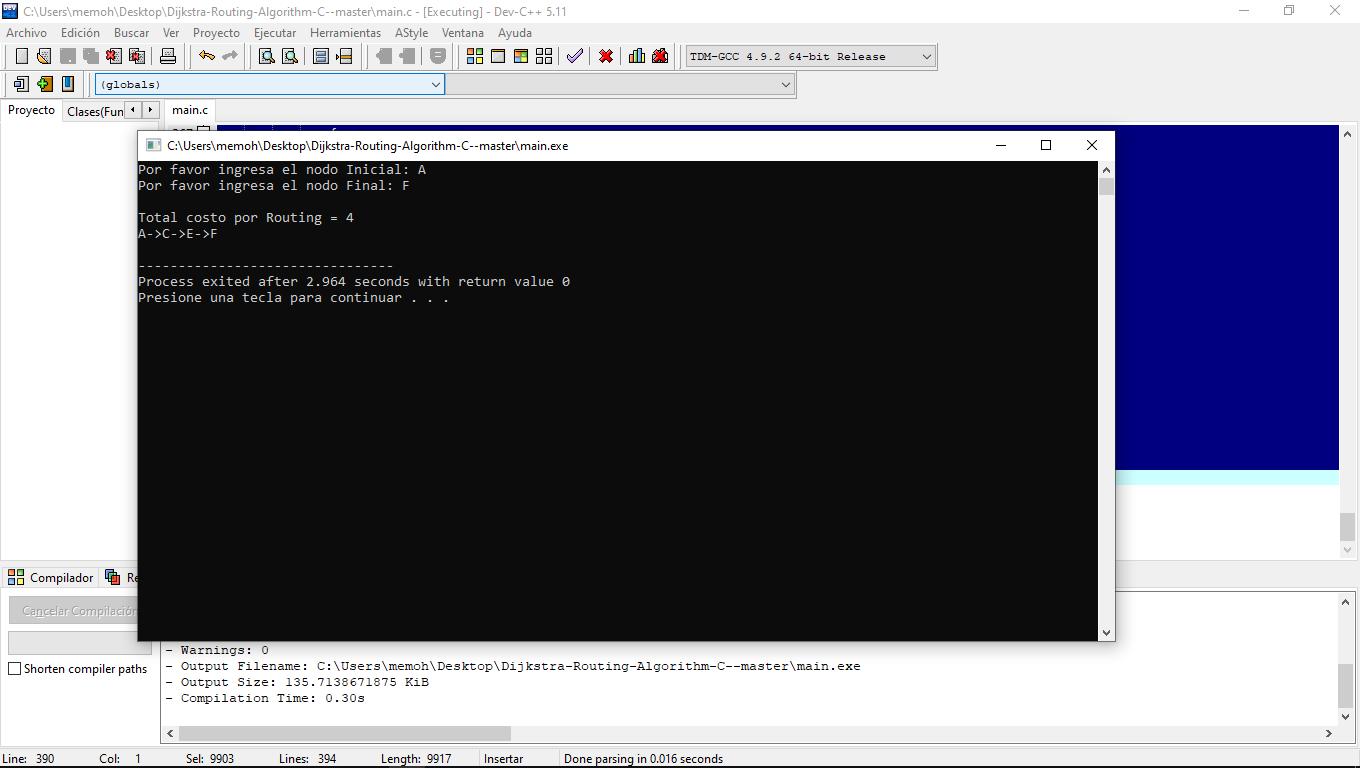
D F 5

Simplemente se deben sustituir estos dígitos por los que estaban para cambiar el Mapeo de redes.

Imagen que contiene objeto

Descripción generada automáticamente

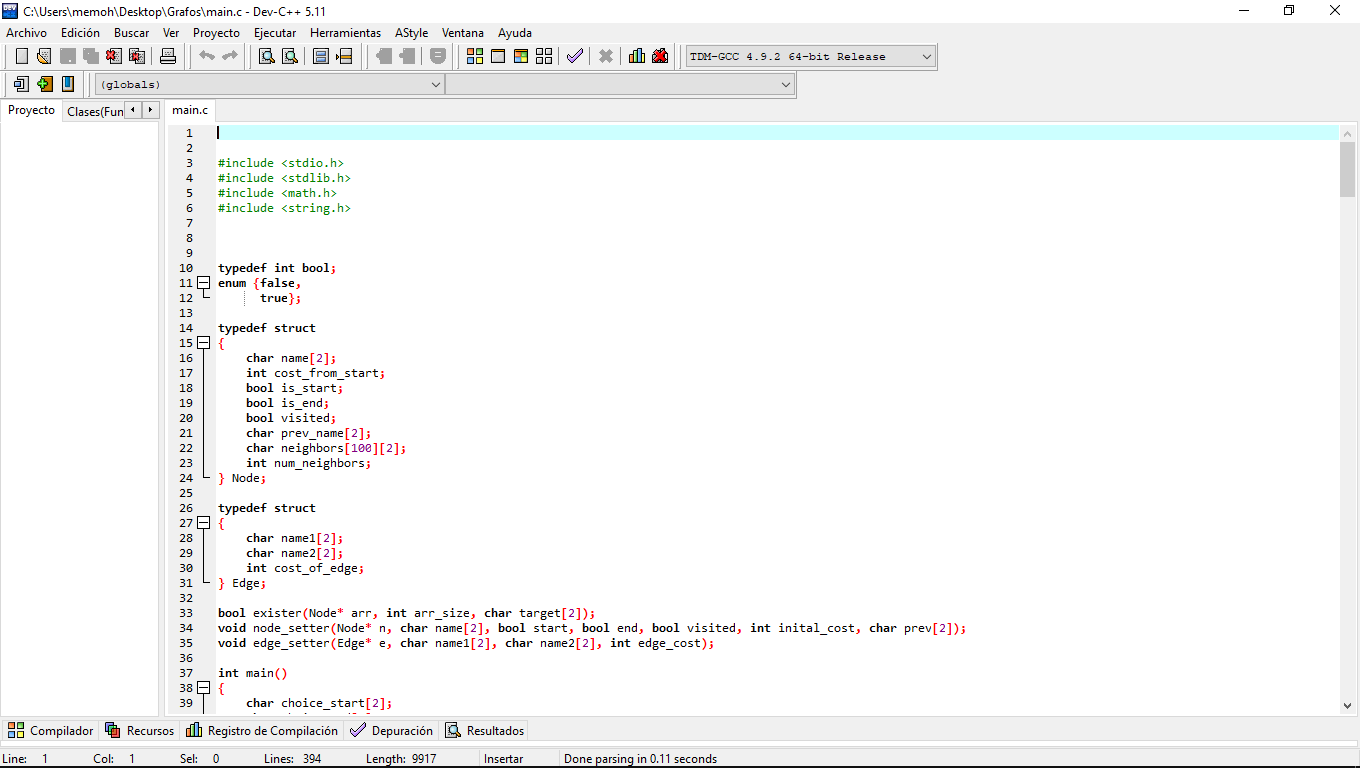
Ahora ejecutaremos el proyecto para ver la interfaz donde nos pedirá que se ingrese el Nodo Principal y el Nodo Final del mapeo de la Red.



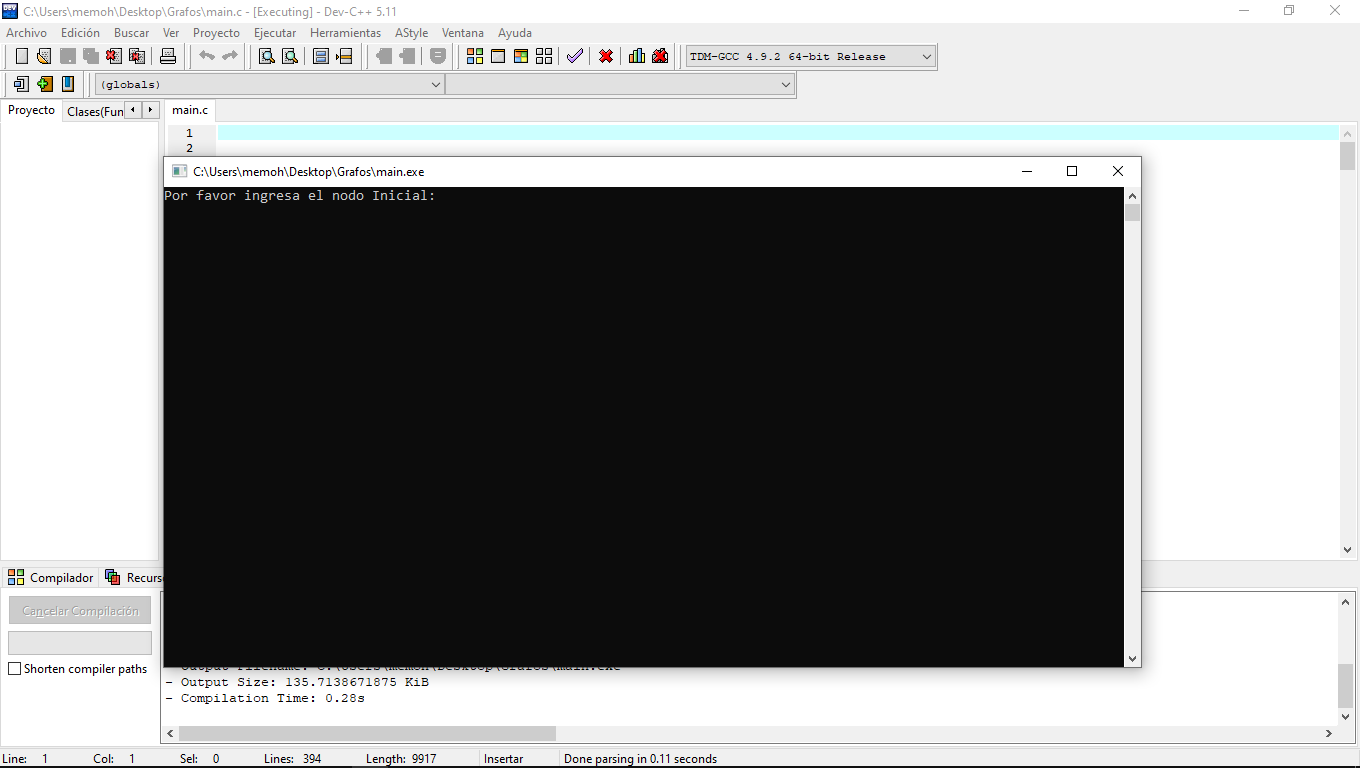
Deben tener cuidado al escribir los nodos ya que estos están en letras MAYUSCULAS si las pones en minúsculas marcara error y se cerrara. Bien el resultado es 4 igual al que obtuvimos en nuestra tabla de arriba por lo tanto el programa esta funcionando.

Bien te marca la ruta mas barata con el resultado marcado en Total costo por Routing.

Para poder cargar el proyecto deberá Descargarlo y descomprimirlo y abrir el archivo llamado main.c con algún software que interprete ejecute y compile c++



En mi caso utilizo Devc++ una vez abierto el archivo main.c simplemente deberá oprimir F11 para compilar el proyecto y ejecutarlo automáticamente.



Con esto se iniciará el proyecto y podrá calcular la mejor ruta a menor costo del Mapa de Red.

Los archivos de la carpeta llamada Grafos deben estar juntos o no funcionara el proyecto

