

Proyecto 3 Árbol de expansión mínimo

Manual de Usuario

El objetivo de este manual es explicar la implementación de los algoritmos Prim y Kruskal para la obtención del árbol de expansión mínimo (en inglés MinimumSpanningTree (MST)) , con su costo total. Se explicará el código sección por sección y se mostrará un ejemplo. El código se encuentra guardado como .h que es un archivo de cabecera, esto permite que se puede incluir en otros códigos con include. El código “MST.h” contiene una clase llamada MST donde se encuentran los siguientes métodos que serán explicados a lo largo de este manual:

```
// Prim
```

```
float prim(string archivo);
```

```
//Deteccion de ciclos por dfs
```

```
dfs(int s , int padre);
```

```
// Kruskal con DFS para detección de ciclos
```

```
float kruskalDFS(string archivo);
```

```
//Funcion Find
```

```
int Find(int p);
```

```
//Funcion Union
```

```
void Union(int i , int j);
```

```
// Kruskal con Union-Find para detección de ciclos
```

```
float kruskalUF(string archivo);
```

Consideraciones importantes:

- Puedes ocupar el IDE de tu preferencia, en este manual se muestran ejemplos utilizando DEV-C++.

- Los elementos a ordenar están guardados en un archivo.txt (Ej. Prueba.txt) que debe estar guardado en la misma carpeta del código. El archivo que recibe la función de lectura describe un grafo NO dirigido con costos enteros de sus aristas. La manera en la están guardados los datos es la siguiente:

[Número de nodos] [Número de aristas]

[Nodo 1 de la arista 1] [Nodo 2 de la arista 1] [costo de la arista 1]

[Nodo 1 de la arista 2] [Nodo 2 de la arista 2] [costo de la arista 2]

.....

- Puede haber datos iguales y negativos.

- El archivo que mandara a llamar a MST.h (Ej.main.cpp) debe contener el #include "MST.h".

```
#include "MST.h"
```

- Los archivos MST.h, Prueba.txt y el main.cpp deben de estar guardados en la misma carpeta.

- Dentro del programa se metió código para calcular el tiempo de ejecución de cada algoritmo (desde la leída del grafo), este lo regresa el tiempo en ms

- Este manual contiene dos ejemplos uno con un grafo con pocos nodos(4) y otro con varios nodos(500).

Esta es una vista en general del código fuente de "MST.h":

```
MST.h  main.cpp
12  #include <iostream> // Para el manejo de datos cin y cout
13  #include <fstream> //Para el manejo de archivos
14  #include <vector> //Para el manejo de vectores
15  #include <ctime> //Para el manejo del tiempo de ejecución [1]
16  #define INF 9999999 //Para la representacion del Infinito
17  using namespace std;
18  unsigned t0, t1; //Variables para medir el tiempo de ejecucion
19  vector<int> parent; //Vector para guardar Los papás
20  class MST{
21  private:
22      vector<int> visited; //Vector que se ocupara para ver Los nodos visitados
23      vector<int> pesos; //Vector que se ocupara para guardar los pesos de las aristas
24      bool flag; //Este bool solo puede ser true o false y se ocupara la detección de ciclos
25      int nodes; // Variable que contiene el numero de nodos del grafo
26      int **Grafo; // Inicialización de una matriz dinamica [3]
27      int **Grafo2;
28
29  public:
30      // Prim con key-value
31      /*Este metodo recibe el nombre del archivo que contiene el grafo y devuelve el MST con su costo total*/
32      float prim(string archivo){
116      //Metodos utilizamos para Union-Find
117      //Encuentra el set del vertice p
118      int Find(int p)
119      {
125      /*Union de i y j , devuelve false si i y j ya estan en el mismo set*/
126      void Union(int i, int j)
127      {
133      // Kruskal con Union-Find para detección de ciclos
134      /*Este metodo recibe el nombre del archivo que contiene el grafo y devuelve el MST con su costo total*/
135      float kruskalUF(string archivo)
136      {
210
211      /*Detección de ciclos por dfs , devuelve true->si hay ciclo y false -> no ciclo
212      -1 -> No visitados (Blanco)
213      0 -> Siendo Visitados (Gris)
214      1 -> Completamente visitados (Negro)
215      Recibe el nodo a analizar y su padre*/
216      void dfs(int s, int padre){
232
233      // Kruskal con DFS para detección de ciclos
234      /*Este metodo recibe el nombre del archivo que contiene el grafo y devuelve el MST con su costo total*/
235      float kruskalDFS(string archivo){
333  }; //Así se tiene que cerrar una clase
```

*Es importante terminar la clase con); un punto y coma

Explicación método Prim:

El método Prim recibe como entrada un string que es el nombre de un archivo que contiene un grafo NO DIRIGIDO, y lo lee colocándolo en una lista de adyacencia, en este caso la lista de adyacencia es una matriz de adyacencia. Se marca un vértice cualquiera que será el vértice con el que se comenzará. Se selecciona la arista de menor peso incidente en el vértice seleccionado anteriormente y se selecciona el otro vértice en el que incide dicha arista. Repetir el paso 2 siempre que la arista elegida enlace un vértice seleccionado y otro que no lo esté. Es decir, siempre que la arista elegida no cree ningún ciclo. El árbol de expansión mínima será encontrado cuando hayan sido seleccionados todos los vértices del grafo.

```
MST.h main.cpp
36 // Prim con key-value
37 /*Este metodo recibe el nombre del archivo que contiene el grafo y regresa el MST con su costo total*/
38 float prim(string archivo){
39     t0=clock(); //Se pone al principio para medir el tiempo de ejecucion
40     int n , a, c , d, z ; // n-> no. nodos a->aristas
41     int suma; //Para el costo total del MST
42
43     ifstream miArchivo(archivo); //Obtener el string del nombre del archivo que se desea abrir
44     miArchivo >> n >> a; //En la primera linea del txt se encuentra el numero de nodos y de aristas
45     //que se guardaran en estas variables respectivamente
46     int V = n; //Paso el numero de nodos a la variable V
47     long GrafoP [V][V]; // Declaro una Matriz de tamaño NodosxNodos , donde estara el Grafo[2]
48     // Inicializo todos los espacios de la matriz en cero*Esto se hace para que no se le meten valores basura a la matriz
49     for(int i = 0; i < V; i++) {
50         for(int j = 0; j < V; j++) {
51             GrafoP[i][j]=0;
52         }
53     }
54     if (miArchivo.is_open())
55     {
56         for(int i = 0; i<a ; i++) // Este for va desde 0 hasta el numero de aristas para leer los datos
57             //siguientes de archivo.txt
58             {
59                 miArchivo >> c >> d>> z; // c-> nodo 1 , d -> nodo 2 , z -> peso
60                 GrafoP[c][d] = z; //Guardo los pesos de las aristas en las coordenadas correspondientes
61                 GrafoP[d][c] = z; // a sus nodos , como es no dirigido tambien lo hago de nodo 2 a nodo 1
62             }
63     }
64     miArchivo.close(); //Cerrar el archivo
65 }
66 else
67 {
68     cout<< "Error al abrir el archivo"<<endl; //Si el archivo no se encuentra en la misma o hay
69     //Un error en el archivo se mostrara este mensaje
70 }
71 /*Este comentario sirve para imprimir tu matriz si deseas
```

...

```
MST.h main.cpp
71  /*Este parte comentada sirve para imprimir tu matriz si deseas
72  para comprobar los valores que se ingresaron*/
73  /* for(int i = 0; i < V; i++) {
74      for(int j = 0; j < V; j++) {
75          cout << GrafoP[i][j] << " ";
76      }
77      cout << endl;
78  }*/
79  int no_a= 0; //Variable que va ayudar a controlar el numero de aristas encontrada
80  //Vector para el control de los nodos seleccionados
81  //Seleccionado -> true , los inicializamos todos en false
82  vector<bool> selected(V, false); //Recordar que V contiene el numero de nodos del grafo
83  //Aqui se establece en que nodo se desea empezar la búsqueda
84  //empezaremos por el cero asi que lo haremos true para que ya esta visitado
85  selected[0] = true;
86  int x; // numero fila
87  int y; // numero columna
88
89  /* Para cada vértice en el conjunto del seleccionando se encuentran todos los vértices adyacentes
90  , se calcula el peso desde el vértice seleccionado si el vértice ya habia sido seleccionado
91  se descarta de lo contrario se elige otro vértice más cercano al vértice seleccionado al principio.*/
92
93  while (no_a< V - 1) { /*El numero de aristas de un MST siempre sera (V-1)*/
94      int min = INF; //Key value a infinito
95      x = 0;
96      y = 0;
97      for (int i = 0; i < V; i++) {
98          if (selected[i]) {
99              for (int j = 0; j < V; j++) {
100                  if (!selected[j] && GrafoP[i][j]) { // Que no este seleccionado y que si haya arista
101                      if (min > GrafoP[i][j]) {
102                          min = GrafoP[i][j]; //Ahora al key value se le pasa el numero que esta en el Grafo la posicion [i][j]
103                          x = i;
104                          y = j;
105                      }
106                  }
107              }
108          }
109      }
```

...continúa

...

```
while (no_a < V - 1) { /*El numero de aristas de un MST siempre sera (V-1)*/
    int min = INF; //Key value a infinito
    x = 0;
    y = 0;
    for (int i = 0; i < V; i++) {
        if (selected[i]) {
            for (int j = 0; j < V; j++) {
                if (!selected[j] && GrafoP[i][j]) { // Que no este seleccionado y que si haya arista
                    if (min > GrafoP[i][j]) {
                        min = GrafoP[i][j]; //Ahora al key value se le pasa el numero que esta en el Grafo la posicion [i][j]
                        x = i;
                        y = j;
                    }
                }
            }
        }
    }
    cout << "(" << x << ", " << y << ", " << GrafoP[x][y] << ")"; //Se imprime la fila la columna de la matriz para que ver de que nodo a que nodo
    cout << endl; //tambien su peso, que es la interseccion
    selected[y] = true; //Ahora ese nodo es marcado como seleccionado
    no_a++; //Se aumenta uno al contador de las aristas
    suma = suma + GrafoP[x][y]; //Para ir sumando los pesos de las aristas seleccionadas
}
t1 = clock(); //Para decir que aqui debe de parar de contar el tiempo de ejecucion
double time = (double(t1-t0)/CLOCKS_PER_SEC); // [1]
cout << "TIEMPO:" << time*1000 << "ms" << endl;
return (float)suma; //Regresar el costo total
}
```

/* ... */

Explicación método kruskalDFS:

En este método se manda a llamar a dfs(int s , int padre); para detención de ciclos y esto se hace en base a que el nodo cambia de estados previamente establecidos.

```
MST.h  main.cpp  MST2.cpp
233 |
234 | // Kruskal con DFS para detección de ciclos
235 | /*Este metodo recibe el nombre del archivo que contiene el grafo y regresa el MST con su costo total*/
236 | float kruskalDFS(string archivo){
237 |     int n , aristas, c , d, z ; //n-> nodos aristas->aristas
238 |     int mincost = 0; //Par guardar el Costo total del MST
239 |     ifstream miArchivo(archivo); //Leer la primera linea del txt donde se encuentra el numero de nodos y aristas
240 |     miArchivo >> n >> aristas;
241 |     nodes = n; //La variables nodes ahora tiene el valor de n
242 |     pesos.resize(aristas); //Establecer el tamaño del vector de pesos como el numero de aristas
243 |     //Establacer el tamaño de mis matrices
244 |     Grafo= new int*[nodes];
245 |     for(int i = 0; i < nodes; i++){
246 |         Grafo[i] = new int[nodes];
247 |     }
248 |     Grafo2= new int*[nodes]; //El Grafo 2 es donde ire guardando el MST
249 |     for(int f = 0; f < nodes; f++){
250 |         Grafo2[f] = new int[nodes];
251 |     }
252 |     //Inicializar mis dos matrices en cero . Esto ayuda que no les entre algun valor basura
253 |     for(int i = 0; i < nodes; i++) {
254 |         for(int j = 0; j < nodes; j++) {
255 |             Grafo[i][j]=0;
256 |             Grafo2[i][j]=0;
257 |         }
258 |     }
259 |     if (miArchivo.is_open())
260 |     {
261 |         for(int i = 0; i<aristas ; i++) // Este for va desde 0 hasta el numero de aristas para Leer Los datos
262 |             //siguientes de archivo.txt
263 |             {
264 |                 miArchivo >> c >> d>> z; // c-> nodo 1 , d -> nodo 2 , z -> peso
265 |                 Grafo[c][d] = z; //Guardo los pesos de las aristas en las coordenadas correspondientes
266 |                 Grafo[d][c] = z; // a sus nodos , como es un Grafo No dirigido tambien lo hago de nodo 2 a nodo 1
267 |                 if(Grafo[c][d]!=0){ //Si el peso no es cero meto el peso al vector de pesos
268 |                     pesos.at(i) = Grafo[c][d];
269 |                 }
270 |             }
271 |         miArchivo.close(); //Cierro el archivo
272 |     }
273 |     else
274 |     {
275 |         cout<< "Error al abrir el archivo"<<endl; ////Si el archivo no se encuentra en la misma o hay un error en el archivo se mostrara este mensaje
276 |     }
```

...continua

MST.h

main.cpp

```

276 visited.resize(nodes); //Establecer el tamaño del vector para los nodos visitados al de los nodos
277 //Ordeno el vector de mis pesos de manera creciente usando BubbleSort y los guardo en el vector de pesos
278 int temp;
279 for(int i=1; i<aristas; ++i){
280     for(int j=0; j<(aristas-i); ++j){
281         if(pesos[j]>pesos[j+1])
282             {
283                 temp=pesos[j];
284                 pesos[j]=pesos[j+1];
285                 pesos[j+1]=temp;
286             }
287     }
288 }
289 int min = INF, a = 0, b = 0; //Inicializo variables que se utilizaran dentro del while
290 for(int i = 0 ; i<aristas; i++){ //Este for es para que se pase por todos los pesos de nuestro vector de pesos
291     for (int j = 0; j < nodes; j++) { //Marco todos los nodos como no visitados
292         visited[j] = -1;
293     }
294     for ( int k = 0; k < nodes; k++) {
295         for (int l = 0; l < nodes; l++) {
296             if (Grafo[k][l]!=0 && pesos.at(i)==Grafo[k][l] ) { //Para que entre a este if que a y b tenga en valor de los nodos
297                 a=k; //el peso del grafo tiene que ser igual al primer peso de nuestro
298                 b=l; //vector de pesos que esta ordenado de manera creciente
299             }
300         }
301     }
302     min = Grafo[a][b]; //Se guarda el valor del peso de la arista
303     Grafo2[a][b] = Grafo[a][b]; //Guardo los valores de los nodos en el Grafo2
304     Grafo2[b][a] = Grafo[b][a];
305     flag = false;
306     dfs(a,a); //Mando los nodos a dfs para ver si generan ciclo en el MST
307     if(!flag) //Entra a este if solo si NO se genera ciclo, lo tengo que negar por que es false cuando no hay ciclo
308     {
309         min = Grafo[a][b];
310         cout << "(" << a << ", " << b << ", " << min << ")" << endl;
311         mincost += min; //Llevo la sumatoria del costo total
312         Grafo[a][b] = 0; //Hago el peso cero para que ya no tome en cuenta esa arista
313         Grafo[b][a] = 0; //Tambien la coordenada al reves por ser un grafo no dirigido
314     }
315     else
316     {
317         Grafo2[a][b] = 0; //Si hay ciclo elimino esa arista de mi MST, que esta guardado en el Grafo2
318         Grafo2[b][a] = 0;
319     }

```

...continua


```

315     }
316     else
317     {
318         Grafo2[a][b] = 0; //Si hay ciclo elimino esa arista de mi MST , que esta guardado el Grafo2
319         Grafo2[b][a] = 0;
320     }
321 }
322 t1 = clock(); //Para que termine el tiempo de ejecucion
323 double time = (double(t1-t0)/CLOCKS_PER_SEC);
324 cout << "TIEMPO:" << time * 1000 << "ms" << endl;
325 cout.precision(3);
326 return (float)mincost; //Regresar el costo
327 }
328 }; //Así se tiene que cerrar una clase
329
330

```

Explicación método dfs(int s , int padre); Este es un método recursivo que recibe el nodo a analizar y su papá

```

211
212 /*Detencion de ciclos por dfs , regresa true->si hay ciclo y false -> no ciclo
213 -1 -> No visitados (Blanco)
214 0 -> Siendo Visitados (Gris)
215 1 -> Completamente visitados (Negro)
216 Recibe el nodo a analizar y su padre*/
217 void dfs(int s, int padre){
218     if(visited[s] == -1){ //Hacer el nodo que entra como visitado
219         visited[s] = 0;
220     }
221     for(int i = 0; i < nodes; i++){
222         if(Grafo2[s][i] != 0 && i != padre){ //Si el peso no es cero y el nodo no es igual a papa
223             if(visited[i] == -1){ //Si no ha sido visitado
224                 dfs(i,s); //Llamo al dfs recursivamete
225             }
226             if(visited[i] == 0){ //Si quieres analizar un nodo que esta siendo visitado
227                 flag = true; //Si hay ciclos se regresa un true
228             }
229         }
230     }
231     visited[s] = 1; // Hasta que se termina el proceso del for , maraco el nodo como completamente visitado
232 }
233
234 // Kruskal con DFS para detección de ciclos
235 /*Este metodo recibe el nombre del archivo que contiene el grafo y regresa el MST con su costo total*/
236 float kruskalDFS(string archivo){
237     //Así se tiene que cerrar una clase

```

Explicación método kruskalUF:

Este método hace uso de otros dos métodos Flin y Union mostrados posteriormente pero básicamente lo que hace cada uno es lo siguiente Find(A): Determina a cual conjunto pertenece el elemento A. Esta operación puede ser usada para determinar si 2 elementos están o no en el mismo conjunto. Union(A, B): Une todo el conjunto al que pertenece A con todo el conjunto al que pertenece B, dando como resultado un nuevo conjunto basado en los elementos tanto de A como de B. Cabe mencionar que en esta implementación representamos el grafo como un árbol, donde cada nodo mantiene la información de su nodo padre, la raíz del árbol será el elemento representativo de todo el conjunto.

MST.hmain.cpp

```
264 // Kruskal con Union-Find para detección de ciclos
265 /*Este metodo recibe el nombre del archivo que contiene el grafo y regresa el MST con su costo total*/
266 float kruskalUF(string archivo)
267 {
268     t0=clock();
269     int n , aristas, c , d, z ; // n-> no. nodos a->aristas
270
271     int costo = 0; // Para el costo total del MST
272
273     ifstream miArchivo(archivo);
274     miArchivo >> n >> aristas;
275     int V = n; //Paso el numero de nodos a la variable V
276     int GrafoU [V][V]; // Declaro una Matriz de tamaño NodosxNodos , donde estara el Grafo[2]
277     // Inicializo todos los espacios de la matriz en cero
278     for(int i = 0; i < V; i++) {
279         for(int j = 0; j < V; j++) {
280             GrafoU[i][j]=0;
281         }
282     }
283     if (miArchivo.is_open())
284     {
285         for(int i = 0; i<aristas ; i++) // Este for va desde 0 hasta el numero de aristas para leer los datos
286             //siguientes de archivo.txt
287             {
288                 miArchivo >> c >> d>> z; // c-> nodo 1 , d -> nodo 2 , z -> peso
289                 GrafoU[c][d] = z; //Guardo los pesos de las aristas en las coordenadas correspondientes
290                 GrafoU[d][c] = z; // a sus nodos , como es no dirigido tambien lo hago de nodo 2 a nodo 1
291             }
292         miArchivo.close(); //Cierro el archivo
293     }
294     else
295     {
296         cout<< "Error al abrir el archivo"<<endl;
297     }
298 }
```

MST.h

main.cpp

```

299  /*Este parte comentada sirve para imprimir tu matriz si deseas para comprobar los valores que se ingresaron*/
300  /*for(int i = 0; i < V; i++) {
301      for(int j = 0; j < V; j++) {
302          cout << GrafoU[i][j] << " ";
303      }
304      cout << endl;
305  }*/
306  parent.resize(V); //Inicializo el vector para ver a los papas de los nodos
307                      //con el tamaño de nodos que hay en el grafo
308  //Este for le asigna el padre a los nodos , al principio
309  // cada uno es su propio padre
310  for (int i = 0; i < V; i++) {
311      parent[i] = i;
312  }
313  int no_aristas = 0; //Para el control de las aristas encontradas
314  while (no_aristas < V - 1) { /*El numero de aristas de un MST siempre sera (V-1)*/
315      int min = INF, a = 0, b = 0; //Inicializamos variables que utilizaremos
316      for (int k = 0; k < V; k++) { //Estos for hacen que pases por todos los nodos de
317          for (int l = 0; l < V; l++) { //la matriz
318              if (Find(k) != Find(l) && GrafoU[k][l] < min && GrafoU[k][l] != 0) { //Ve si los nodos no tienen el mismo padre en el metodo
319                  min = GrafoU[k][l]; //Find(p); tambien no entra a este if si el peso es cero
320                  a = k; //pasa los valores de los nodos encontrados a //por que eso significa que no hay arista
321                  b = l; //a y b para ver si se pueden usar dependiendo de lo que regrese el Union
322              }
323          }
324      }
325      Union(a, b); //Si esto regresa true se hace lo de abajo si no ,no
326      cout << "(" << a << ", " << b << ", " << min << ")"; //Se imprime el nodo1 y el nodo2 y su peso
327      cout << endl;
328      no_aristas++; //Se aumenta uno al contador de las aristas
329      costo += min; //Llevar la sumatoria de costo final
330  }
331  t1 = clock(); //Para que deje de correr el tiempo de ejecucion
332  double time = (double(t1-t0)/CLOCKS_PER_SEC); //[1]
333  cout << "TIEMPO:" << time*1000 << "ms" << endl;
334  return (float)costo; //Regreso el costo total del MST
335

```

Explicación métodos Find y Union

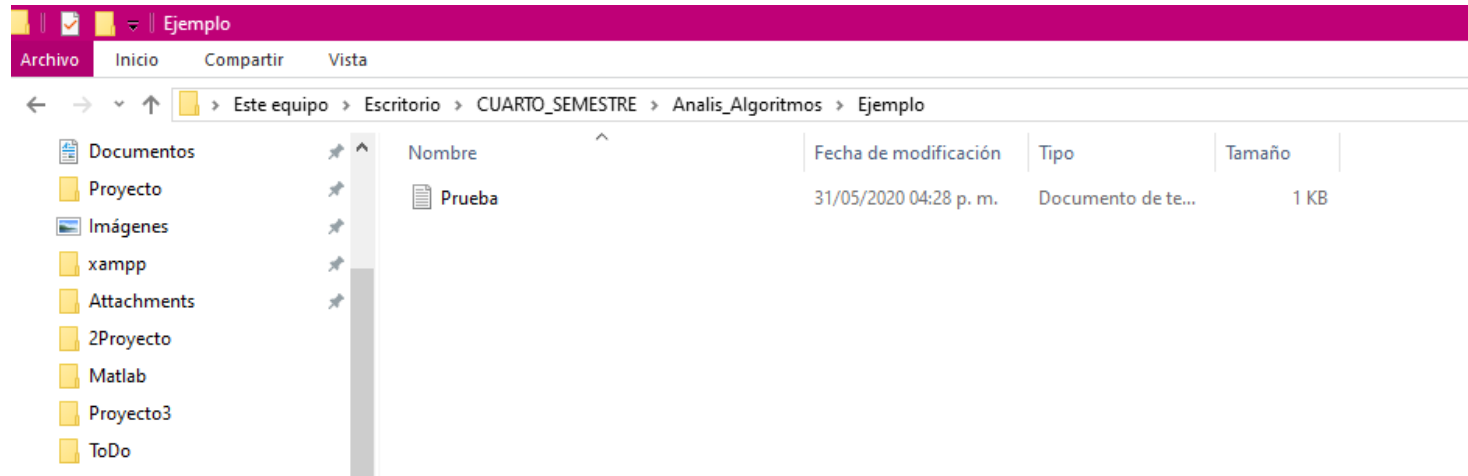
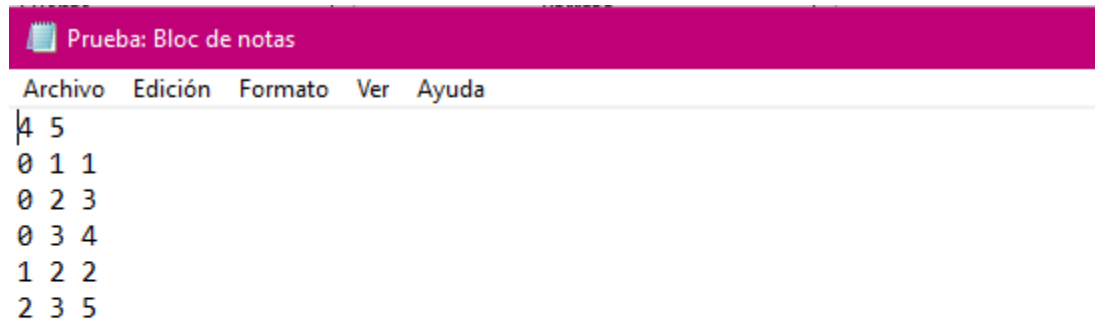
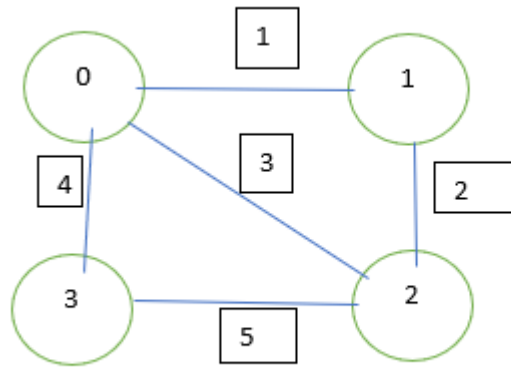
Find :Si estoy en la raíz regreso la raíz si no busco el padre del vértice actual, hasta llegar a la raíz.

Unión: Obtenemos la raíz del vértice x.->Obtenemos la raíz del vértice y.->Actualizamos el padre de alguna de las raíces, asignándole como nuevo padre la otra raíz.

```
MST.h  main.cpp
245  //Metodos utilizamos para Union-Find
246
247  //Encuentra el set del vertice p
248  int Find(int p)
249  {
250      if( p == parent[p] ){ //Si estoy en la raiz, retorno la raiz
251          return p;
252      }
253      else return Find( parent[p] );
254  }
255  /*Union de i y j , regresa false si i y j ya estan en el mismo set*/
256  void Union(int i, int j)
257  {
258      int a = Find(i); //Obtengo la raiz de la componente del vértice i
259      int b = Find(j); //Obtengo la raiz de la componente del vértice j
260
261      parent[a] = b; //Mezclo ambos arboles , actualizando su padre de alguno de ellos como la raiz de otro
262  }
```

Ejemplo de cómo se usa el programa:

Primero debes crear un archivo.txt que contenga los nodos y los pesos del grafo y que cumpla con las especificaciones que se dijeron al principio . En este manual trabajaremos con este grafo: *Recordar que en la primera línea se encuentra el número de nodos y de aristas del grafo.



Segundo deberás crear un archivo.cpp, en este ejemplo lo llamaremos main. Debes poner el #include "MST.h" y definir el objeto de tipo MST. Este programa debe de estar guardado en la misma carpeta del archivo.txt

MST.hmain.cpp

1

2

3

4

5

6

7

8

#include "MST.h"

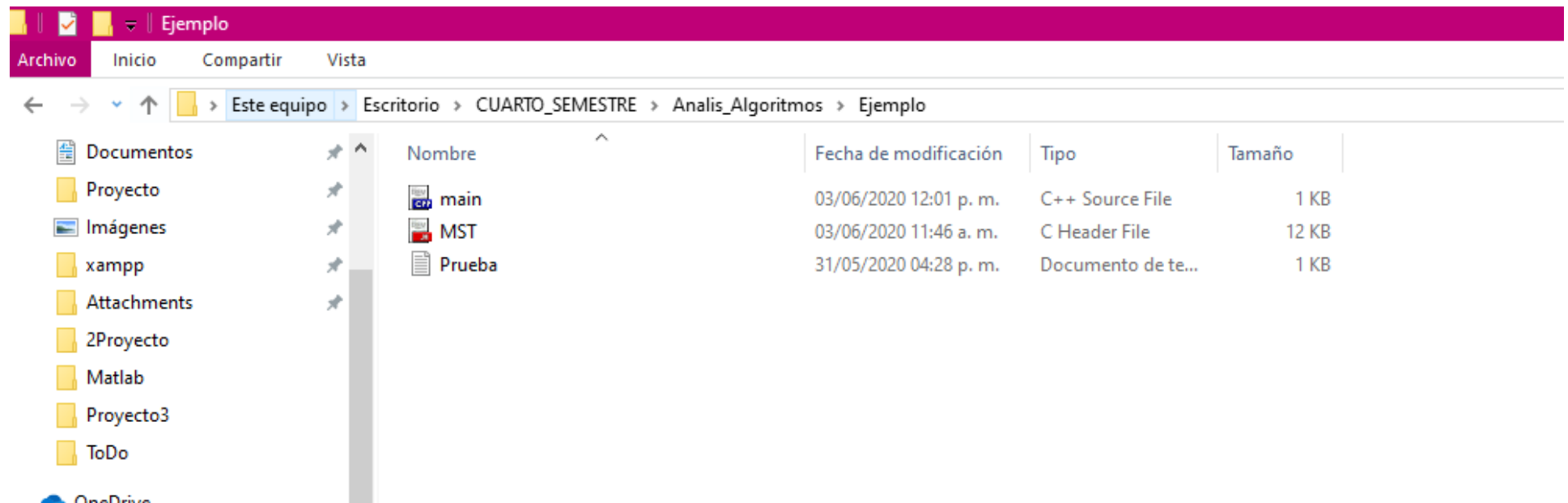
int main(){

MST miMST;

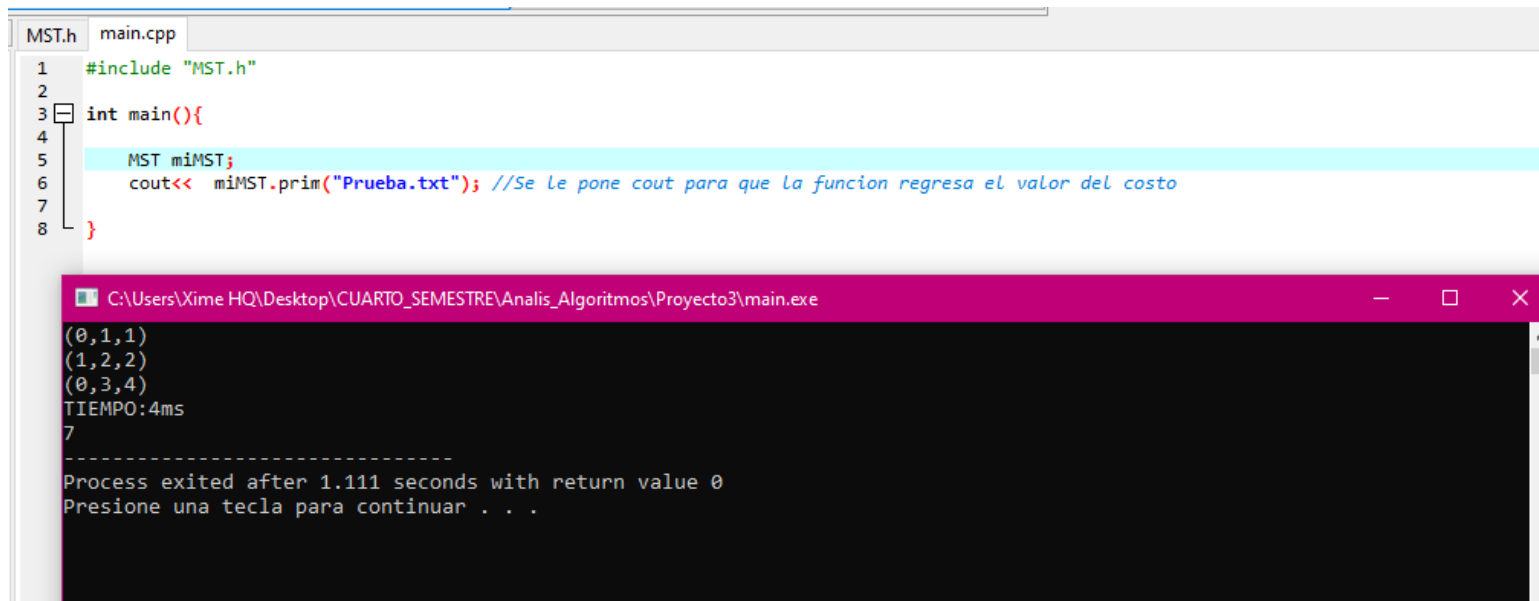
}

Esto se declara para poder llamar a los métodos de la clase MST

La carpeta se debe ver así con tus 3 archivos :



Ahora trabajaremos en el archivo main.cpp, primero llamaremos al método prim



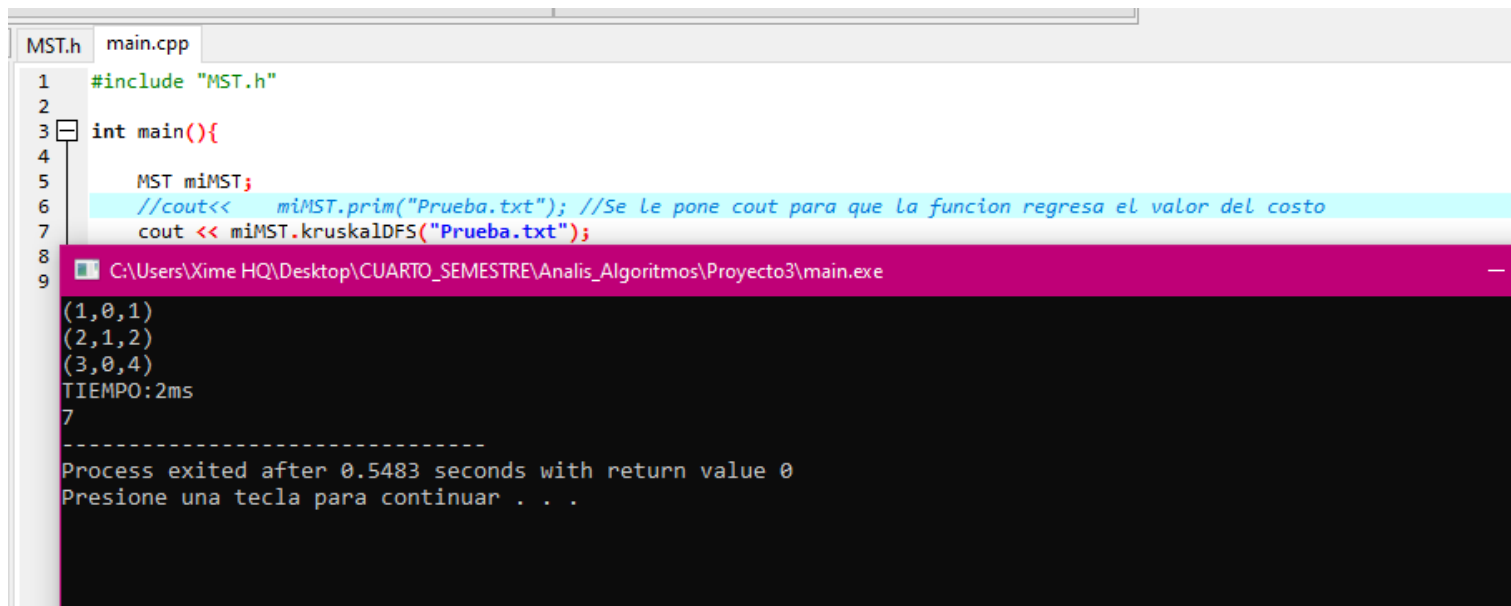
The screenshot shows a C++ IDE with two tabs: 'MST.h' and 'main.cpp'. The 'main.cpp' tab is active, displaying the following code:

```
1 #include "MST.h"
2
3 int main(){
4
5     MST miMST;
6     cout<< miMST.prim("Prueba.txt"); //Se le pone cout para que la funcion regresa el valor del costo
7
8 }
```

Below the code editor, a terminal window titled 'C:\Users\Xime HQ\Desktop\CUARTO_SEMESTRE\Analis_Algoritmos\Proyecto3\main.exe' shows the output of the program:

```
(0,1,1)
(1,2,2)
(0,3,4)
TIEMPO:4ms
7
-----
Process exited after 1.111 seconds with return value 0
Presione una tecla para continuar . . .
```

Posteriormente podemos llamar al método de kruskalDFS



The screenshot shows the same C++ IDE with the 'main.cpp' tab active. The code has been updated to call the 'kruskalDFS' method instead of 'prim':

```
1 #include "MST.h"
2
3 int main(){
4
5     MST miMST;
6     //cout<< miMST.prim("Prueba.txt"); //Se le pone cout para que la funcion regresa el valor del costo
7     cout << miMST.kruskalDFS("Prueba.txt");
8
9 }
```

The terminal window below shows the updated output:

```
(1,0,1)
(2,1,2)
(3,0,4)
TIEMPO:2ms
7
-----
Process exited after 0.5483 seconds with return value 0
Presione una tecla para continuar . . .
```

Finalmente podemos llamar al método kruskalUF mandándole el mismo .txt Nota que puedes comentar la llamada a los otros métodos con //

MST.h main.cpp

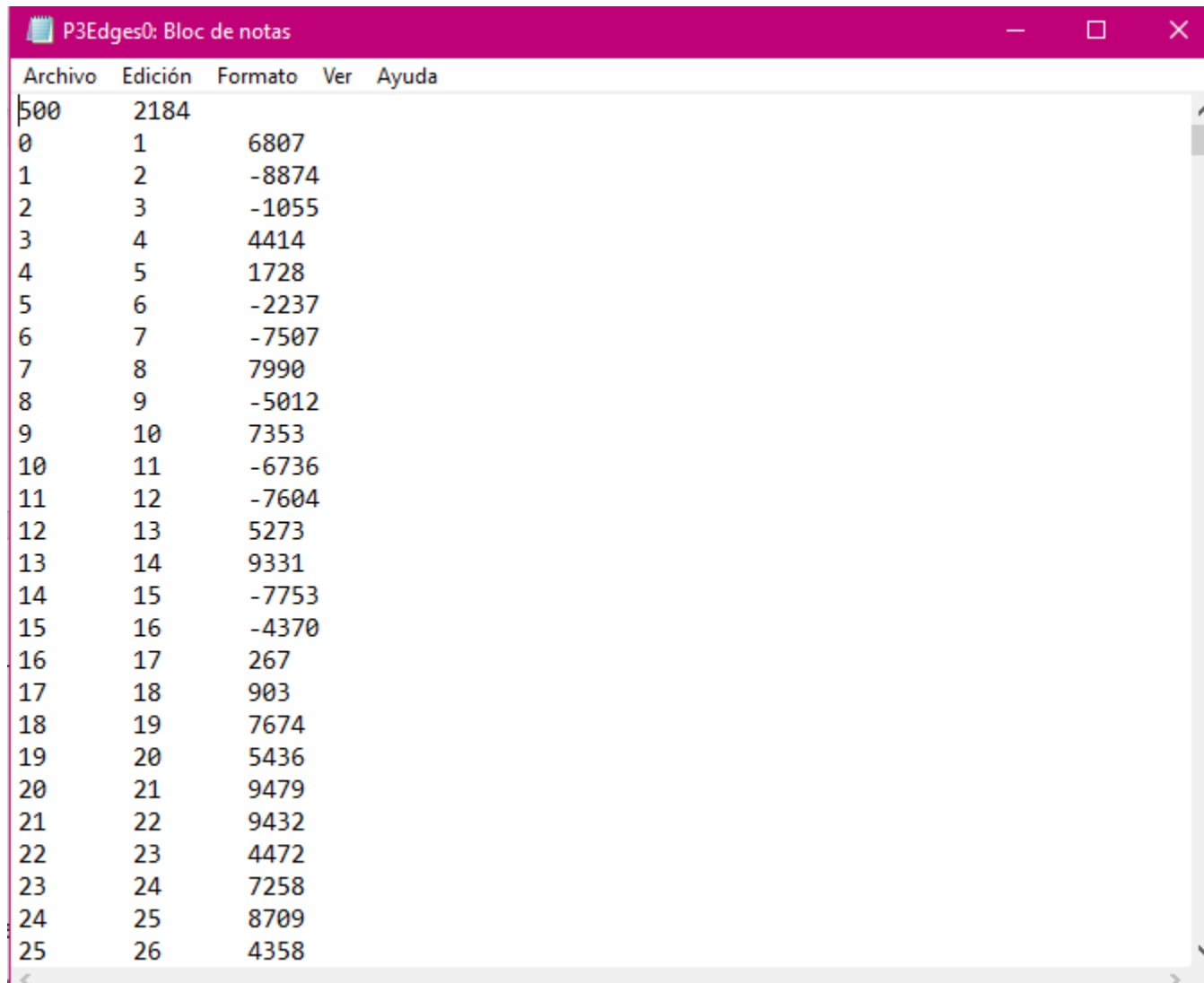
```
1  #include "MST.h"
2
3  int main(){
4
5      MST miMST;
6      //cout<<  miMST.prim("Prueba.txt"); //Se le pone cout para que la funcion regresa el valor del costo
7      //cout <<  miMST.kruskalDFS("Prueba.txt");
8      cout <<  miMST.kruskalUF("Prueba.txt");
9  }
```

C:\Users\Xime HQ\Desktop\CUARTO_SEMESTRE\Analisis_Algoritmos\Proyecto3\main.exe

```
(0,1,1)
(1,2,2)
(0,3,4)
TIEMPO: 2ms
7
```

```
-----
Process exited after 0.6063 seconds with return value 0
Presione una tecla para continuar . . .
```

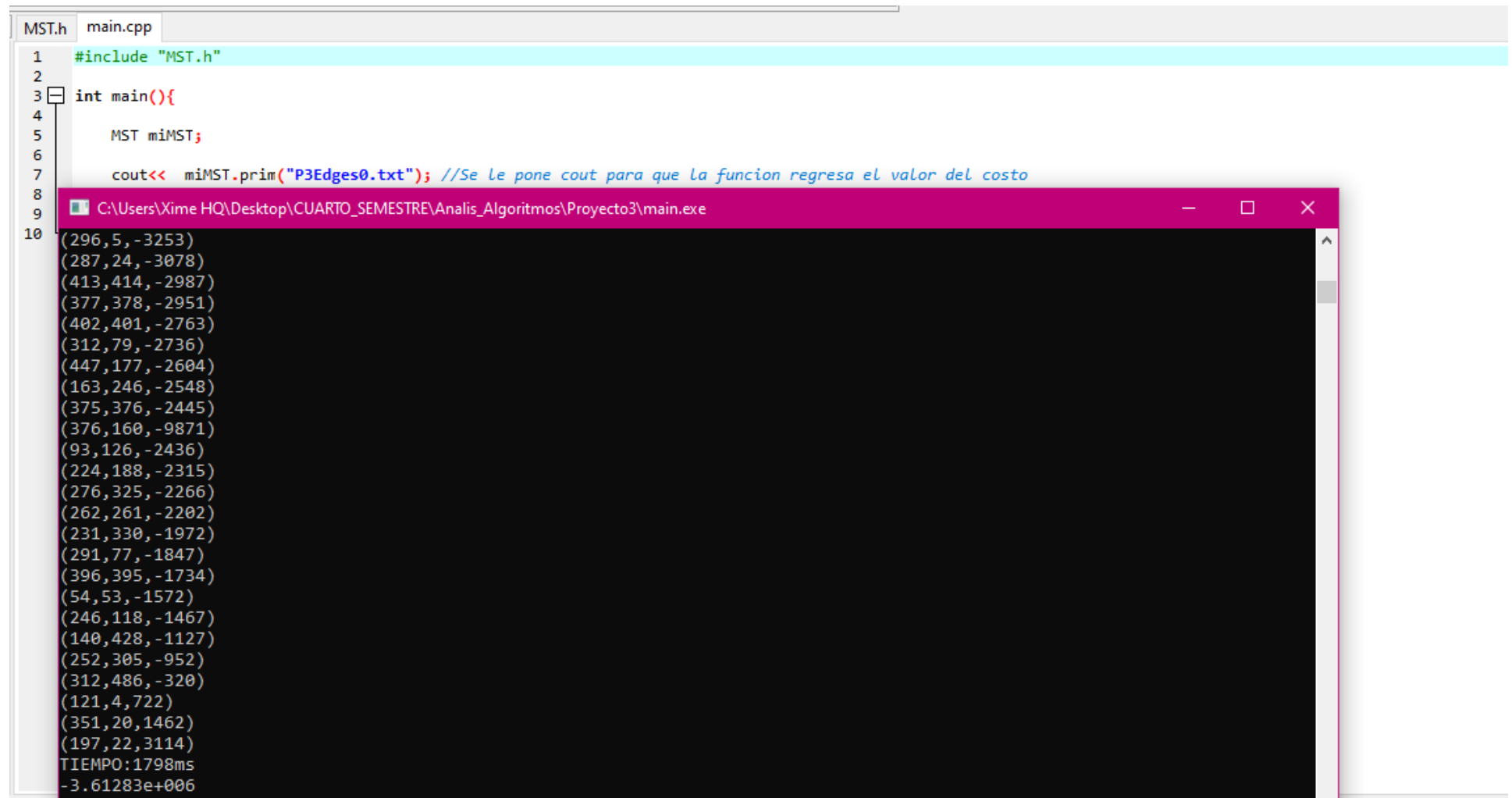

Se mostrará la **Solución Final** ahora con un grafo de 500 nodos que se guardan en un archivo llamada P3Edges0



The image shows a Notepad window titled "P3Edges0: Bloc de notas". The window contains a list of 500 nodes, each with a weight. The list is organized into three columns: the first column contains the node index (0 to 25), the second column contains the node weight (1 to 26), and the third column contains the node weight (6807 to 4358). The list is displayed in a table format with a header row and 26 data rows. The window has a standard Windows interface with a title bar, menu bar, and scrollbars.

Archivo	Edición	Formato	Ver	Ayuda
500	2184			
0	1	6807		
1	2	-8874		
2	3	-1055		
3	4	4414		
4	5	1728		
5	6	-2237		
6	7	-7507		
7	8	7990		
8	9	-5012		
9	10	7353		
10	11	-6736		
11	12	-7604		
12	13	5273		
13	14	9331		
14	15	-7753		
15	16	-4370		
16	17	267		
17	18	903		
18	19	7674		
19	20	5436		
20	21	9479		
21	22	9432		
22	23	4472		
23	24	7258		
24	25	8709		
25	26	4358		

Solución con Prim:



```
MST.h main.cpp
1 #include "MST.h"
2
3 int main(){
4
5     MST miMST;
6
7     cout<< miMST.prim("P3Edges0.txt"); //Se le pone cout para que la funcion regresa el valor del costo
8
9
10
```

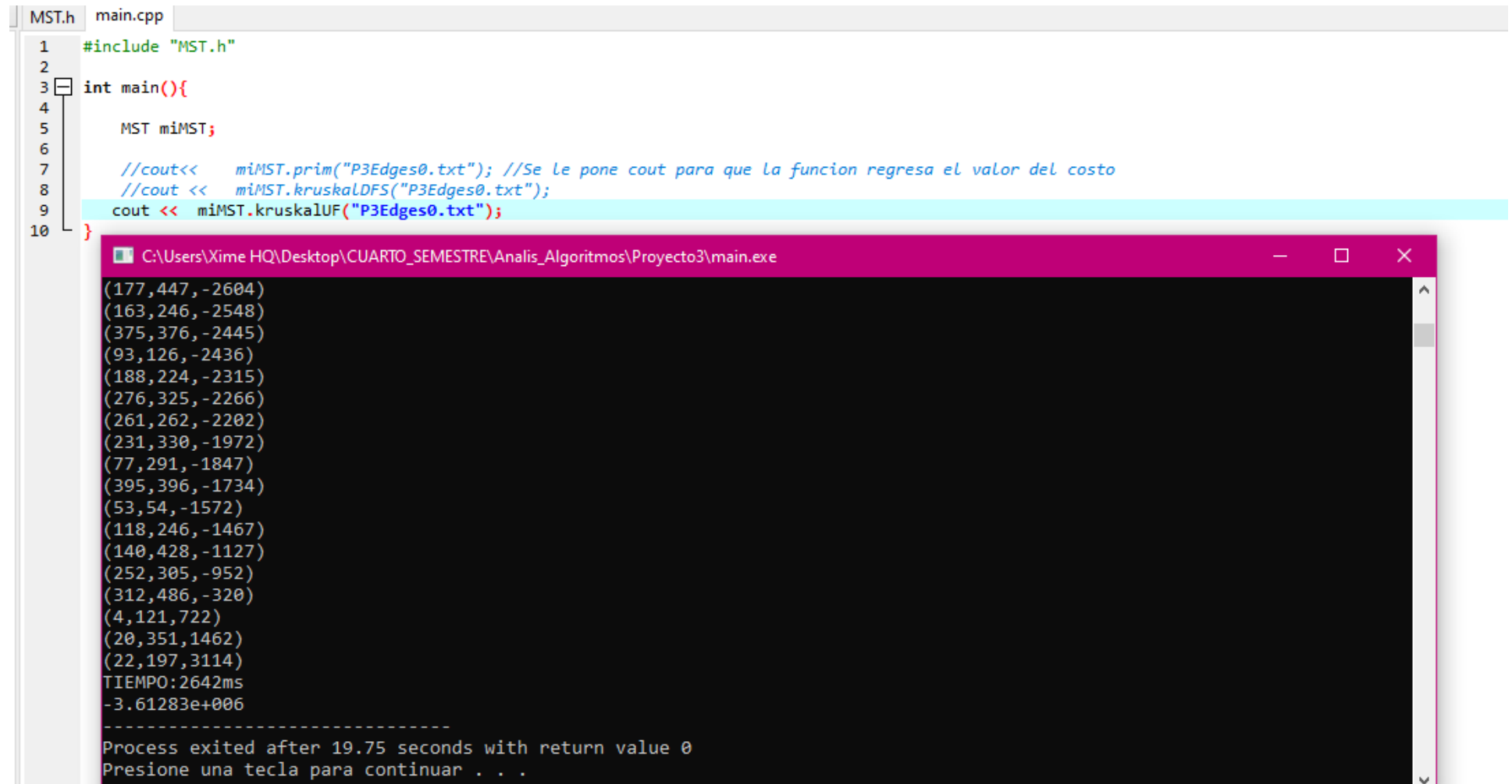
C:\Users\Xime HQ\Desktop\CUARTO_SEMESTRE\Analis_Algoritmos\Proyecto3\main.exe

(296,5,-3253)
(287,24,-3078)
(413,414,-2987)
(377,378,-2951)
(402,401,-2763)
(312,79,-2736)
(447,177,-2604)
(163,246,-2548)
(375,376,-2445)
(376,160,-9871)
(93,126,-2436)
(224,188,-2315)
(276,325,-2266)
(262,261,-2202)
(231,330,-1972)
(291,77,-1847)
(396,395,-1734)
(54,53,-1572)
(246,118,-1467)
(140,428,-1127)
(252,305,-952)
(312,486,-320)
(121,4,722)
(351,20,1462)
(197,22,3114)
TIEMPO:1798ms
-3.61283e+006

Solución con Kruscal DFS para la detención de ciclos:

```
MST.h  main.cpp
1  #include "MST.h"
2
3  int main(){
4
5      MST miMST;
6      //cout<<  miMST.prim("P3Edges0.txt"); //Se le pone cout para que la funcion regresa el valor del costo
7      cout << miMST.kruskalDFS("P3Edges0.txt");
8
9  C:\Users\Xime HQ\Desktop\CUARTO_SEMESTRE\Analis_Algoritmos\Proyecto3\main.exe
(246,163,-2548)
(376,375,-2445)
(126,93,-2436)
(224,188,-2315)
(325,276,-2266)
(262,261,-2202)
(330,231,-1972)
(401,103,-1853)
(291,77,-1847)
(396,395,-1734)
(54,53,-1572)
(246,118,-1467)
(428,140,-1127)
(305,252,-952)
(486,312,-320)
(121,4,722)
(351,20,1462)
(197,22,3114)
TIEMPO:2796ms
-3.61e+006
-----
Process exited after 5.648 seconds with return value 0
Presione una tecla para continuar . . .
```

Solución con Kruscal Union -Find para la detención de ciclos:



The image shows a C++ IDE with two tabs: 'MST.h' and 'main.cpp'. The 'main.cpp' tab is active, displaying the following code:

```
1 #include "MST.h"
2
3 int main(){
4     MST miMST;
5
6     //cout<< miMST.prim("P3Edges0.txt"); //Se le pone cout para que la funcion regresa el valor del costo
7     //cout << miMST.kruskalDFS("P3Edges0.txt");
8     cout << miMST.kruskalUF("P3Edges0.txt");
9 }
10
```

Below the code editor, a console window titled 'C:\Users\Xime HQ\Desktop\CUARTO_SEMESTRE\Analisis_Algoritmos\Proyecto3\main.exe' is open. It displays the output of the program, which consists of a list of coordinates, the execution time, and the total cost:

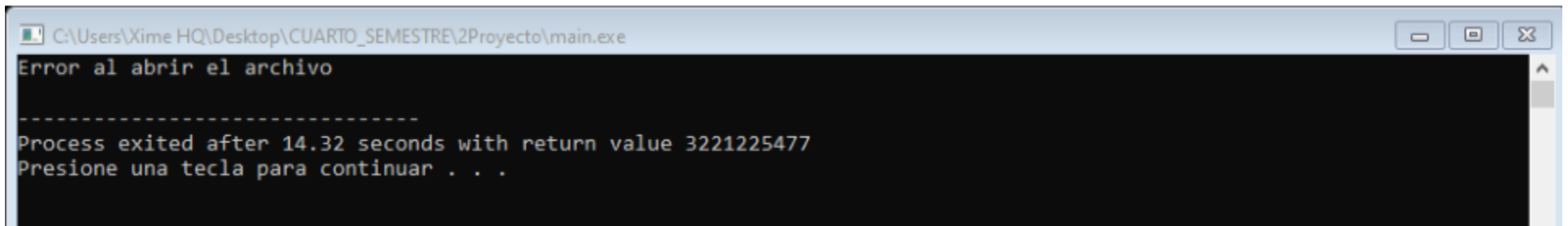
```
(177,447,-2604)
(163,246,-2548)
(375,376,-2445)
(93,126,-2436)
(188,224,-2315)
(276,325,-2266)
(261,262,-2202)
(231,330,-1972)
(77,291,-1847)
(395,396,-1734)
(53,54,-1572)
(118,246,-1467)
(140,428,-1127)
(252,305,-952)
(312,486,-320)
(4,121,722)
(20,351,1462)
(22,197,3114)
TIEMPO:2642ms
-3.61283e+006
-----
Process exited after 19.75 seconds with return value 0
Presione una tecla para continuar . . .
```

Conclusiones de los resultados obtenidos:

Como podemos observar en las capturas de pantalla, el costo del MST es el mismo para los tres algoritmos. El orden de las aristas o de los nodos puede ser diferente, pero al final el costo es el mismo y es importante resaltar que el mínimo costo con que todos los nodos pueden estar conectados. Hablando del tiempo de ejecución realice varias pruebas y el tiempo de ejecución cambia ligeramente dependiendo de cuantas cosas tengas abiertas en tu computadora. Como podemos observar los tiempos de los algoritmos de Kruscal son similares. Cabe mencionar que tiempo se medio desde que se leía el grafo y se metía a la matriz de adyacencia hasta que se despegaban todos las aristas del MST. La complejidad del algoritmo de Prim es $O(n^2)$. Siendo n el número de vértices del grafo y la complejidad del algoritmo de Kruscal es $O(m)$ o $O(m \log n)$ donde m es el número de aristas y n es el número de vértices(nodos) , ya que estamos trabajando con un nodo grande (500 nodos y 2184 aristas) es norma que se tarde mas con Prim o con Kruscal.

Posibles Errores:

-Si el método leerGrafo no encuentra el archivo.txt por que esta mal el nombre o no se encuentra en la misma carpeta que el main.cpp y MST.h o el archivo se encuentra vacio desplegara esto:

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Users\Xime HQ\Desktop\CUARTO_SEMESTRE\2Proyecto\main.exe'. The command prompt displays the following text: 'Error al abrir el archivo', followed by a line of dashes '-----', then 'Process exited after 14.32 seconds with return value 3221225477', and finally 'Presione una tecla para continuar . . .'.

```
C:\Users\Xime HQ\Desktop\CUARTO_SEMESTRE\2Proyecto\main.exe
Error al abrir el archivo
-----
Process exited after 14.32 seconds with return value 3221225477
Presione una tecla para continuar . . .
```

-El archivo que contiene el grafo tiene que tener los nodos empezando en el 0 , si no no hará nada el programa.

-Si el grafo contiene ciclos , no desplegará ninguna arista y el costo será 0

Referencias:

[1] <https://www.lawebdelprogramador.com/foros/Dev-C/707658-medir-tiempo-de-ejecucion.html>

[2]<https://www.tutorialspoint.com/cplusplus-program-to-implement-adjacency-matrix>

[3]<http://mauricioavilesc.blogspot.com/2015/08/matriz-dinamica-en-c.html>