INSTITUTO POLITÉCNICO NACIONAL

"ESCOM" (ESCUELA SUPERIOR DE CÓMPUTO)

DOCENTE: CECILIA ALBORTANTE MORATO

INTEGRANTES:

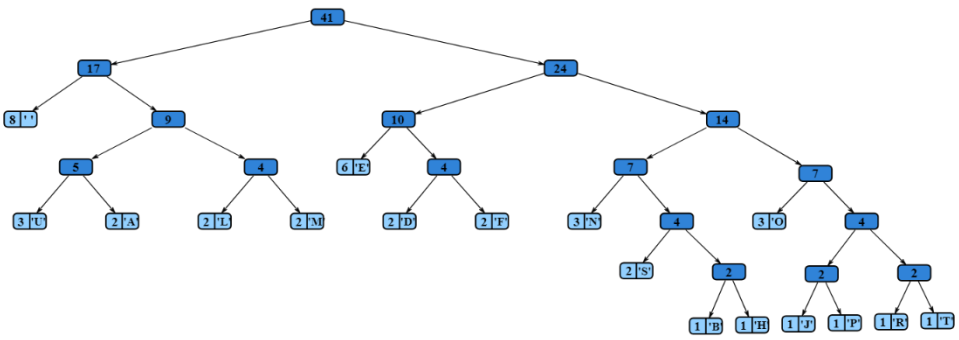
MONTEALEGRE ROSALES DAVID URIEL

VÁZQUEZ BLANCAS CÉSAR SAID

U.A.: ANÁLISIS Y DISEÑO DE ALGORITMOS

GRUPO: 3CM6

PRÁCTICA 6. Árboles de recubrimiento mínimo.



Árboles de recubrimiento mínimo.

Objetivo de la práctica.

Implementar en el lenguaje de programación de su preferencia el algoritmo Kruskal para obtener un árbol de recubrimiento mínimo en un grafo.

Indicaciones solicitadas.

- Probar los grafos de ejemplo vistos en clase o algunos de los grafos adjuntos en esta tarea.
- Tener los grafos precargados en un archivo o en el código del programa.

Introducción.

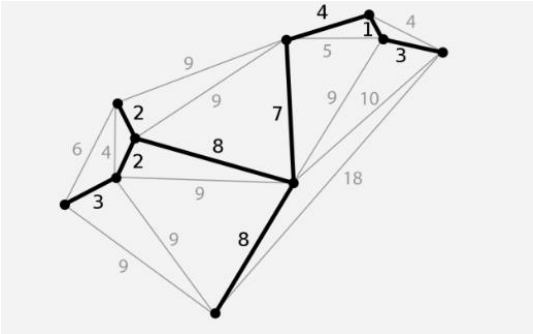
Dado un grafo conexo y no dirigido, un árbol recubridor, árbol de cobertura o árbol de expansión de ese grafo es un subgrafo que tiene que ser un árbol y contener todos los vértices del grafo inicial.

Cada arista tiene asignado un peso proporcional entre ellos, que es un número representativo de algún objeto, distancia, etc.; y se usa para asignar un peso total al árbol recubridor mínimo computando la suma de todos los pesos de las aristas del árbol en cuestión.

Un árbol recubridor mínimo o un árbol de expansión mínimo es un árbol recubridor que pesa menos o igual que todos los otros árboles recubridores. Todo grafo tiene un bosque recubridor mínimo.

Debemos tener en cuenta que:

- Puede haber más de un árbol recubridor posible.
- El árbol recubridor mínimo será el de menos coste.



Ejemplo de árbol recubridor mínimo.

Donde:

- Cada punto representa un vértice.
- Cada arista está etiquetada con su peso (equivale a su longitud).

Método de Kruskal.

Los árboles de expansión mínima pueden definirse sobre puntos del espacio euclidiano o sobre una gráfica. Para el método de Kruskal, los árboles de expansión mínima se definen sobre gráficas.

El algoritmo de Kruskal es un proceso que permite unir todos los nodos de un grafo formando un árbol, tomando en cuenta el peso de las aristas y cuyo coste total es el mínimo posible.

Desarrollo de la práctica.

Para la resolución de la práctica solicitada, fue necesario implementar el algoritmo de Kruskal para encontrar el árbol de recubrimiento mínimo en un grafo ponderado no dirigido. El código utilizado es el siguiente:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX 1000  // Número máximo de vértices

// Estructura para representar una arista
struct Arista {
   int o;     // Vértice origen
   int d;    // Vértice destino
   int w;       // Peso entre el vértice origen y destino
};

int comparador(const void *a, const void *b) {
   return ((struct Arista *)a)->w - ((struct Arista *)b)->w;//devuelve la resta en pesos, si es negativo significa que a esta antes de b, si es positiva,b va antes de a y si son igulaes el orden es relativo
}

int a[MAX]; // Este arreglo contiene el padre del i-ésimo nodo

// inicialización
void inicialización(int n) {
   for (int i = 1; i <= n; ++i)
      a[i] = i;
}
```

```
int Find(int x) {
   return (x == a[x]) ? x : (a[x] = Find(a[x]));//La función verifica si el vértice x es de su propio subconjunto,si si, devuelve x, Si x no es su propia raíz, entonces la función realiza una búsqueda recursiva para encontrar el conjunto a la que pertenece x.
}

void Union(int x, int y) {
   a[Find(x)] = Find(y);//se une el vertice a el conjunto del vertice representado por x
}

int same(int x, int y) {
   return Find(x) == Find(y);//compara y retorna un 1 si ambos son del mismo conjunto o un 0 si no lo son
}



void Kruskal(struct Arista arista[], int V, int A) {
   struct Arista arm[MAX];
   int o, d, w;
   int t = 0;      // Peso total del Arbol de recubrimiento minimo
   int numAristas = 0; // Número de Aristas del arm

   inicialización(V);         // Inicializamos cada vertice
   qsort(arista, A, sizeof(struct Arista), comparador);  // Ordenamos las aristas por su comparador

   for (int i = 0; i < A; ++i) {//recorremos las aristas ordenadas por el peso
      o = arista[i].o;//se guardan los valores en variables auxiliares
      d = arista[i].d;
      w = arista[i].w;

      if (!same(o, d)) {//si ambos vertices estan el el mismo conjunto
         t += w;//el peso de esa arista se suma al total
         arm[numAristas++] = arista[i];// se agrega al arm ,la arista actual y se incrementa para la siguinete arista
         Union(o, d);//se unen y forman un conjunto los vertices involucrados
      }
   }// si no, simplemente se descarta y no se hace nada
   //si no sigue con el programa e imprime todos los datos guardados en el mismo
```

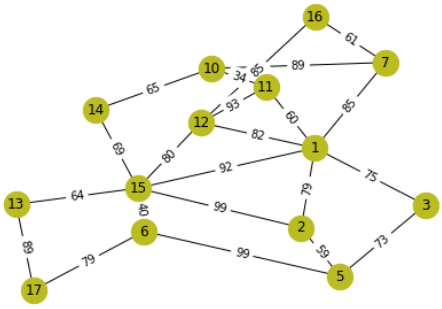
```
   for (int i = 0; i < numAristas; ++i)
      printf("( %d , %d ) : %d\n", arm[i].o, arm[i].d, arm[i].w);

   printf("El costo mínimo de todas las aristas del Arbol de recubrimiento minimo
es : %d\n", t);//imprime el costo de las aristas
}

int main() {
   int V = 14;//declaracion del numero de vertices y aristas
   int E = 22;
   int V2 = 8;
   int E2 = 13;
   int V3 = 11;
   int E3 = 20;
   //estructura de los grafos
   struct Arista arista[] = {
      {1, 2, 79},
      {1, 3, 75},
      {3, 5, 73},
      {2, 5, 59},
      {1, 7, 85},
      {1, 11, 60},
      {7, 10, 89},
      {10, 11, 34},
      {7, 16, 61},
      {12, 16, 85},
      {11, 12, 93},
      {1, 12, 82},
      {10, 14, 65},
      {14, 15, 69},
      {12, 15, 80},
      {1, 15, 92},
      {2, 15, 99},
      {6, 15, 40},
      {5, 6, 99},
      {13, 15, 64},
      {13, 17, 89},
      {6, 17, 79}
   };
   struct Arista arista2[] = {
      {1, 2, 10},
```

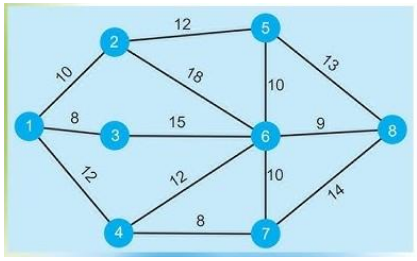
```
    {1, 3, 8},
    {1, 4, 12},
    {2, 6, 18},
    {3, 6, 15},
    {4, 6, 12},
    {2, 5, 12},
    {4, 7, 8},
    {5, 6, 10},
    {6, 7, 10},
    {5, 8, 13},
    {6, 8, 9},
    {7, 8, 14}
  };
  struct Arista arista3[] = {
    {1, 2, 8},
    {1, 11, 3},
    {2, 11, 7},
    {3, 11, 5},
    {2, 3, 10},
    {3, 4, 9},
    {4, 5, 13},
    {2, 5, 2},
    {5, 6, 10},
    {4, 6, 12},
    {6, 7, 8},
    {5, 7, 6},
    {1, 7, 9},
    {7, 8, 7},
    {1, 8, 10},
    {8, 9, 3},
    {1, 9, 6},
    {9, 9, 10},
    {1, 10, 12},
    {10, 11, 8}
  };
  Kruskal(arista, V, E);
  Kruskal(arista2, V2, E2);
  Kruskal(arista3, V3, E3);

  return 0;
}
```

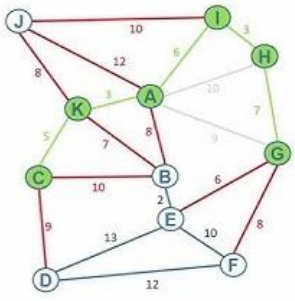
El código utilizado contiene librerías y funciones para su correcto funcionamiento, las cuales son:

- #define MAX 1000 (librería): Nos determina el número máximo de vértices que se pueden utilizar.
- Struct Arista: Se utiliza para representar las aristas de nuestro grafo, esta tendrá tres variables las cuales representan el vértice de origen, vértice de destino y el peso (longitud) que cuesta unir los vértices.
- Función comparador: Nos ayudara a comparar las aristas en orden ascendente (según el peso que utilizan). Devuelve la diferencia entre los pesos de dos aristas (si es negativo significa que a esta antes de b, si es positiva, b va antes de a y si son iguales el orden es relativo).
- Función inicialización: Inicializa cada vértice como su propio padre.
- Función Find: Busca el conjunto al que pertenece un vértice recursivamente para y optimiza futuras búsquedas.
- Función Union: Se encarga de unir los conjuntos a los que pertenecen dos vértices.
- Función same: Se encarga de verificar si dos vértices están en el mismo conjunto.
- Función Kruskal: Se trata del algoritmo de Kruskal. Primero, inicializa la estructura de conjuntos. Luego, ordena las aristas por peso y, en un bucle, selecciona las aristas de menor peso que no formen ciclos. Las aristas seleccionadas se almacenan en el arreglo "arm". Finalmente, imprime las aristas del Árbol de Recubrimiento Mínimo y su peso total.
- Función main: En esta función, se definen los vértices y aristas de tres grafos diferentes y se realiza la llamada a la función Kruskal para cada uno de ellos para así analizar el resultado que nos presenta cada grafo.

Grafos solicitados en la práctica:

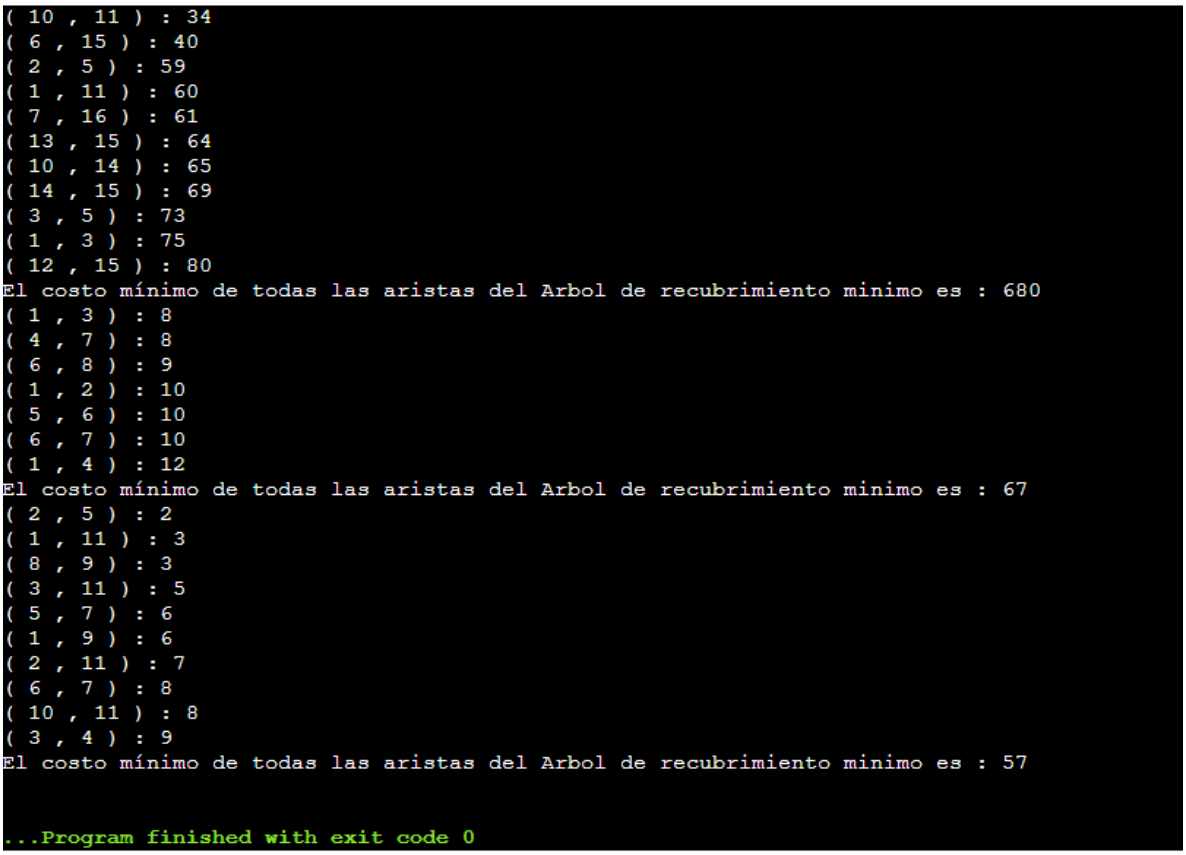






Resultados.

En esta sección podemos observar el correcto funcionamiento del programa:



Conclusión.

En conclusión, la práctica consistió en aplicar el algoritmo de Kruskal para encontrar el árbol de recubrimiento mínimo en un grafo con pesos.

El código diseñado utiliza funciones y estructuras para manejar los diferentes pasos del algoritmo, como la inicialización de conjuntos y la unión de puntos. Además, se realizaron pruebas con varios ejemplos de grafos para asegurarse de que el algoritmo funcione correctamente.

La salida del programa nos da la lista de conexiones que forman el árbol de recubrimiento mínimo, junto con el costo total de esas conexiones. Podemos decir que la implementación del algoritmo de Kruskal nos permite encontrar la manera más eficiente de conectar todos los puntos del grafo sin gastar más de lo necesario.