Universidad de Guadalajara

Centro Universitario de Ciencias Exactas e Ingenierías

Seminario de Algoritmia

REPORTE DE PRÁCTICA

IDENTIFICACIÓN DE LA PRÁCTICA

Práctica	**8**	**Nombre de la práctica**	**Algoritmo de Kruskal**
Fecha	**28/10/2021**	**Nombre del profesor**	**Alma Nayeli Rodríguez Vázquez**
Nombre de los integrantes del equipo		**1. Cárdenas Pérez Calvin Cristopher**	
		2. Farfán de León José Osvaldo	
		3. García Martínez Noe Aaron	

OBJETIVO

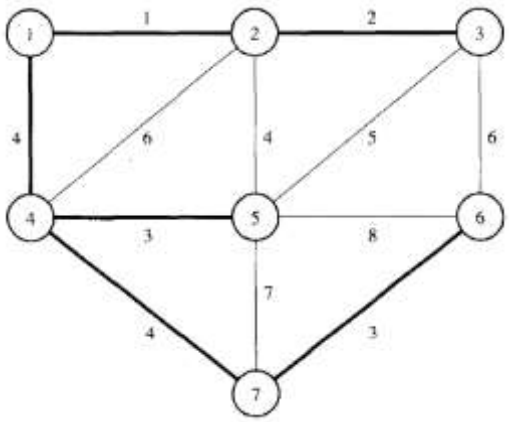
El objetivo de esta práctica consiste en implementar el algoritmo de Kruskal para encontrar el árbol de recubrimiento mínimo de un grafo.

PROCEDIMIENTO

Realiza la implementación siguiendo estas instrucciones.

Implementa el algoritmo de Kruskal utilizando Matlab y C++ / Python. Para la implementación, apóyate del siguiente algoritmo y del ejemplo:

```
function Kruskal(G = ⟨N, A⟩: graph; length: A → ℝ⁺): set of edges
    {initialization}
    Sort A by increasing length
    n ← the number of nodes in N
    T ← ∅ {will contain the edges of the minimum spanning tree}
    Initialize n sets, each containing a different element of N
    {greedy loop}
    repeat
        e ← {u, v} ← shortest edge not yet considered
        ucomp ← find(u)
        vcomp ← find(v)
        if ucomp ≠ vcompthen
            merge(ucomp, vcomp)
            T ← T ∪ {e}
    until T contains n − 1 edges
    return T
```



Step	Edge considered	Connected components
Initialization	—	{1} {2} {3} {4} {5} {6} {7}
1	{1, 2}	{1, 2} {3} {4} {5} {6} {7}
2	{2, 3}	{1, 2, 3} {4} {5} {6} {7}
3	{4, 5}	{1, 2, 3} {4, 5} {6} {7}
4	{6, 7}	{1, 2, 3} {4, 5} {6, 7}
5	{1, 4}	{1, 2, 3, 4, 5} {6, 7}
6	{2, 5}	rejected
7	{4, 7}	{1, 2, 3, 4, 5, 6, 7}

IMPLEMENTACIÓN

Agrega el código de tu implementación aquí.

```
no_nodos-7;
no_conjuntos=no_nodos;
grafo=[1 2 1;
             1 4 4;
              2 3 2;
              2 4 6;
              2 5 4;
              3 5 5;
              3 6 6;
              4 5 3;
              4 7 4;
              5 6 8;
              5 7 7;
              6 7 3];
pesos=grafo(: , 3) ;
aristas=grafo(: , 1 : 2) ;
[pesos, ind]=sort(pesos);
aristas=aristas(ind, : );
no_aristas=size(aristas, 1);
T=[];
conjuntos= { } ;
for i=1 : no_nodos
     conjuntos { i } = i ;
end
for i=1 : no_aristas
     aristas_i=aristas ( i , : ) ;
for j=1 : no_conjuntos

if i smember (arista_i(1) , conjuntos{ j }
conjunto1= j;
break;
       end
end
for j=1 :no_conjuntos
     if ismember(arista_i (2) ,conjuntos { j })
               conjunto2=j;
```

```
end
end
if conjunto1~=conjunto2
     conjuntos{conjunto1} = [conjuntos{conjunto1 } ; conjuntos{
conjuntos2} ];
     conjuntos{conjuntos2} = [];
     T= [ T; arista_i];
End
sizeT=size(T, 1);
if sizeT==no_nodos-1
     break;
end
end
T(: , 1 : 2)
pesoTotal=sum(T(: , 3) )
```

Código de Matlab

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <string.h>
using namespace std;

class Arista
{
	int vertice1, vertice2, peso;

public:

	Arista(int v1, int v2, int peso)
	{
		vertice1 = v1;
		vertice2 = v2;
		this->peso = peso;
	}

	int obtenerVertice1()
	{
		return vertice1;
	}

	int obtenerVertice2()
	{
		return vertice2;
	}

	int obtenerPeso()
	{
		return peso;
	}
```

```
	bool operator < (const Arista& arista2) const
	{
		return (peso < arista2.peso);
	}
};

class Grafo
{
	int V;
	vector<Arista> aristas;

public:

	Grafo(int V)
	{
		this->V = V;
	}

	void adicionarArista(int v1, int v2, int peso)
	{
		Arista arista(v1, v2, peso);
		aristas.push_back(arista);
	}

	int buscar(int subset[], int i)
	{
		if(subset[i] == -1)
			return i;
		return buscar(subset, subset[i]);
	}

	void unir(int subset[], int v1, int v2)
	{
		int v1_set = buscar(subset, v1);
		int v2_set = buscar(subset, v2);
		subset[v1_set] = v2_set;
	}

	void kruskal()
	{
		vector<Arista> arbol;
		int size_aristas = aristas.size();

		sort(aristas.begin(), aristas.end());

		int * subset = new int[V];
```

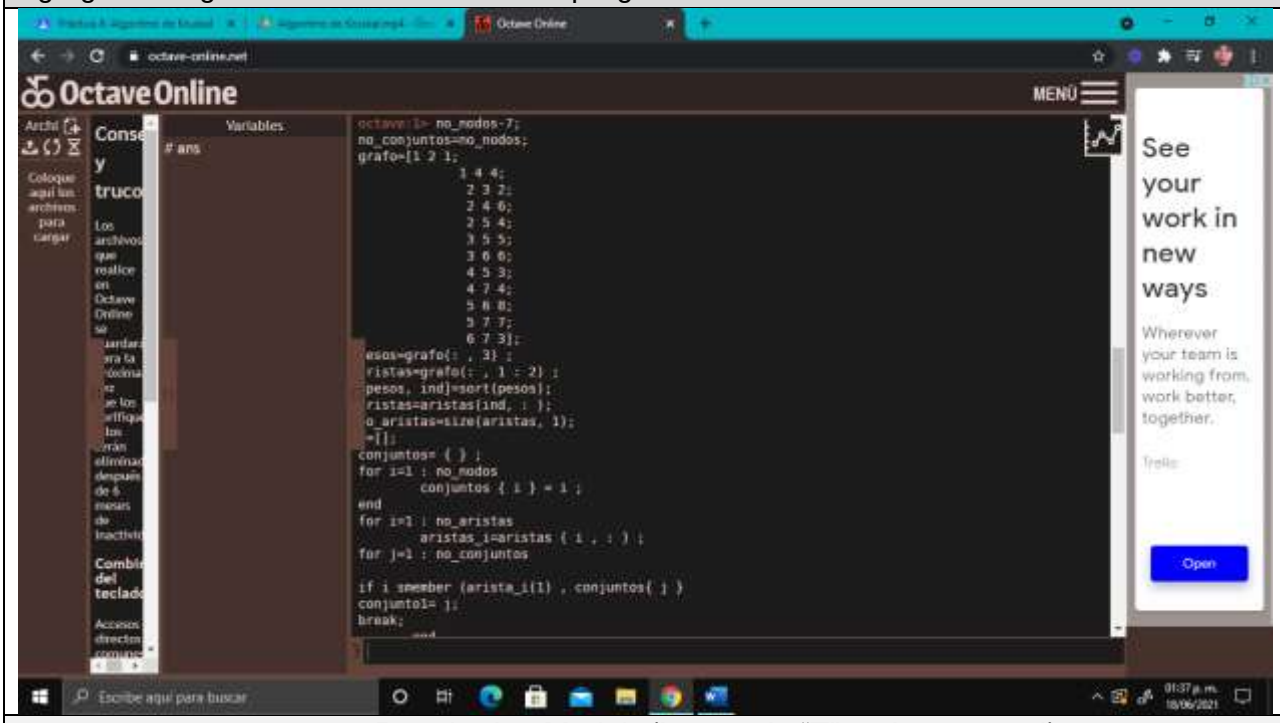
```
                memset(subset, -1, sizeof(int) * V);

                for(int i = 0; i < size_aristas; i++)
                {
                        int v1 = buscar(subset, aristas[i].obtenerVertice1());
                        int v2 = buscar(subset, aristas[i].obtenerVertice2());

                        if(v1 != v2)
                        {
                                arbol.push_back(aristas[i]);
                                unir(subset, v1, v2);
                        }
                }

                int size_arbol = arbol.size();

                for(int i = 0; i < size_arbol; i++)
                {
                        char v1 = '1' + arbol[i].obtenerVertice1();
                        char v2 = '1' + arbol[i].obtenerVertice2();
                        cout << "(" << v1 << ", " << v2 << ") = " << arbol[i].obtenerPeso() << endl;
                }
                cout << "peso minimo :" <<peso;
        }
};

int main(int argc, char *argv[])
{
        Grafo g(7); // grafo

        // agregar las aristas
        g.adicionarArista(0, 1, 1);
        g.adicionarArista(0, 3, 4);
        g.adicionarArista(1, 2, 2);
        g.adicionarArista(1, 3, 6);
        g.adicionarArista(1, 4, 4);
        g.adicionarArista(2, 4, 5);
        g.adicionarArista(2, 5, 6);
        g.adicionarArista(3, 4, 3);
        g.adicionarArista(3, 6, 4);
        g.adicionarArista(4, 5, 8);
        g.adicionarArista(4, 6, 7);
        g.adicionarArista(5, 6, 3);

        g.kruskal(); // ejecutar el algoritmo de Kruskal

        return 0;
}
```

Seminario de Algoritmia

Código en C++/Python

RESULTADOS

Agrega la imagen de la consola con el despliegue de los resultados obtenidos.



Resultados Matlab (La matriz “T” y el peso total)

Resultados C++/Python (La matriz “T” y el peso total)

CONCLUSIONES

Escribe tus observaciones y conclusiones.
El algoritmo de kruskal es un algoritmo que trabaja con grafos conexos y busca generar arboles mediante seleccionar solamente las aristas que no ciclen alguna parte del árbol, ya que esto es una característica importante para diferenciar un árbol de un grafo conexo. Otra cualidad del algoritmo de Kruskal es que no solo se requiere de tener un grafo conexo, sino que además este ponderado. Esto se refiera a que a cada arista del grafo se le asigna un peso, este peso puede ser arbitrario o estar relacionado a algún aspecto de la arista, como puede ser su longitud. Con esto también descubrimos otra función del algoritmo de Kruskal, y es que este decide por las aristas cuyo peso sea el menor a comparación con las demás aristas que se conectan al vértice que se está analizando actualmente. Esto logra que la suma de los pesos de las aristas seleccionadas sea el menor peso total posible (excepto en algunas pocas excepciones). Este algoritmo puede tener como utilidad, por ejemplo, en una red de trenes: se tienen varias rutas que el tren puede tomar, pero queremos que se toma la ruta más corta a nuestro destino, sin causar alguna especie de ciclo en la ruta. Entonces podemos utilizar el algoritmo para eliminar los ciclos en la ruta, seleccionamos la estación de destino y podemos tomar una ruta marcada por el algoritmo que será la más corta para llegar a la estación.