

# Optimiza!ción

## Práctica 4: Árboles generadores de mínimo coste, el algoritmo de Kruskal

Profesor responsable:

Dificultad: media.

Tutorización: semana del 8 de abril.

Corrección: semana del 15 de abril.

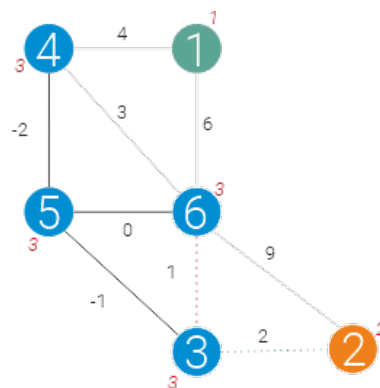
### Objetivo

El objetivo de esta práctica es la implementación de un algoritmo capaz de construir el árbol generador de mínimo coste, MST, de un grafo no dirigido con costes o pesos en sus aristas. Para resolver el problema MST usaremos el algoritmo de Kruskal (1956) que lo construye, por lo que debemos inducir un orden entre las aristas según sus costes, para examinarlas en ese orden y decidir si entran o no en la solución, siempre evitando los ciclos.

### El Algoritmo de Kruskal

El algoritmo de Kruskal selecciona las aristas que entran en la solución de forma que no construyan un ciclo con las previamente seleccionadas, además de analizar las candidaturas por orden no decreciente de su peso o coste.

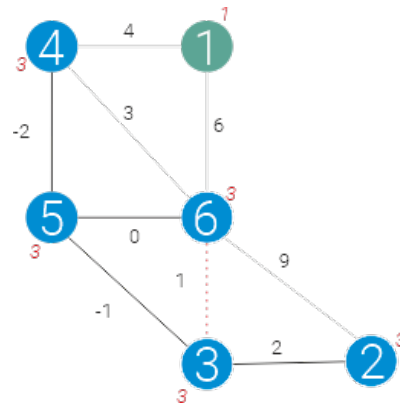
En el algoritmo de Kruskal, se controla de forma muy sencilla cuando una arista  $(i, j)$ , formará un ciclo o no con las aristas previamente seleccionadas para la solución. Usa un sencillo sistema de etiquetado de los nodos que están en cada componente conexa. En el siguiente ejemplo, se ilustra cómo, pensemos que ya se han añadido algunas aristas de coste bajo a una solución parcial:



- se han añadido por ahora:  $(4, 5)$ , de coste -2;  $(3, 5)$  de coste -1;  $(5, 6)$  de coste 0; (sí, tienen coste negativo, pero eso no es problema en el MST)
- las componentes conexas con las aristas añadidas serían: componente conexa 1, verde,  $\{1\}$ ; componente conexa 2, naranja,  $\{2\}$ ; componente conexa 3, azul,  $\{3, 4, 5, 6\}$ .

## Optimiza!ción

- siguiente arista candidata, (3, 6), forma ciclo, pues los dos n odos están en la misma componente conexas: se rechaza.
- siguiente arista candidata, (2, 3), no forma ciclo, el nodo 2 está en la componente conexas etiquetada con el 2; el nodo 3 está en la componente conexas etiquetada con el 3: se acepta.
- una vez aceptada, se actualizan las componentes conexas y sus etiquetas:



En pseudocódigo, sin mucho detalle, sería:

Ordenar las aristas en orden no decreciente según sus costes

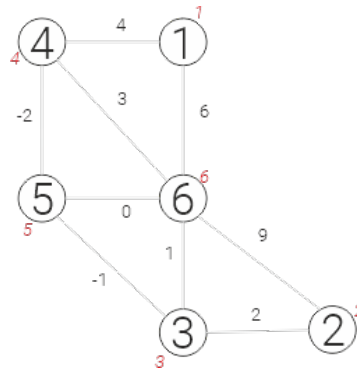
$T = \emptyset$ ;

Para todo nodo  $i$  de  $V$  hacer  $\text{raiz}[i] = i$ ;

Mientras en  $T$  no haya  $n-1$  aristas hacer

```
{
  Sea  $e=(i,j)$  la siguiente arista ordenada;
  Si  $\text{raiz}[i] \neq \text{raiz}[j]$  entonces
    {
       $T = T \cup \{e\}$ ;
       $\text{kill} = \text{raiz}[i]$ ;
      Para todo nodo  $k$  de  $V$  hacer
        {
          Si  $\text{raiz}[k] = \text{kill}$  entonces  $\text{raiz}[k] = \text{raiz}[j]$ 
        }
    }
}
```

Por tanto, sólo añadir que la posición de etiquetado inicial de las componentes conexas, cuando no hay ninguna arista aún en la solución, sería el que reflejara que cada nodo está en su propia componente conexas unitaria:



## Plan de trabajo

Para implementar el algoritmo de Kruskal para la construcción del MST, hay que resolver dos obstáculos:

- ordenación de las aristas según sus costes de forma lo más eficiente posible.
- controlar las componentes conexas que se van conformando a medida de que se van añadiendo ala solución parcial.

## Ordenación de las aristas según sus costes

La implementación del algoritmo de Kruskal **necesita** que las aristas sean analizadas en orden no decreciente de costes. Para ello, proponemos construir un vector con las aristas y sus costes, *manteniendo un orden parcial de interés* como veremos. Por ello, ha de incluirse en el fichero grafo.h, la siguiente estructura de datos:

```
typedef struct {
    unsigned extremo1, extremo2;
    int peso;
} AristaPesada;
```

Y así, cargar la información de la lista de adyacencia del grafo, en LS, a este vector:  
vector <AristaPesada> Aristas;

```
/*Cargamos todas las aristas de la lista de adyacencia*/
Aristas.resize(m);
unsigned k = 0;
for (unsigned i = 0; i<n; i++)
{
    for (unsigned j=0; j<LS[i].size();j++)
    {
        if (i < LS[i][j].j)
        {
            Aristas[k].extremo1 = i;
            Aristas[k].extremo2 = LS[i][j].j;
            Aristas[k++].peso = LS[i][j].c;
        }
    }
};
```

Por tanto, es en el vector **Aristas** en el que buscaremos esa arista de menor coste candidata a entrar en la solución. Estrategias para ello:

- ordenar todo el vector por orden no decreciente de costes (puede usarse cualquier método: *heapsort*, *mergesort*, *quicksort*)
- ir situando en las posiciones de cabeza las aristas con menos coste, a medida que el algoritmo lo necesite.

En ese caso, sería algo así:

- Vector **Aristas** cargado de la lista de adyacencia en LS: cada fila contiene la posición 0..m-1, los dos extremos (recordemos que el nodo  $i$  aparece como  $i-1$ ), y el coste  $c$ :

índice	extremo	extremo	coste	head
0	0	3	4	
1	0	5	6	
2	1	2	2	
3	1	5	9	
4	2	4	-1	
5	2	5	1	
6	3	4	-2	
7	3	5	3	
8	4	5	0	

- Posicionamos el valor de la cabeza **head** a 0, y la arista de menor coste es buscada para que ocupe esa posición, y por tanto, queda:

índice	extremo	extremo	coste	head
0	3	4	-2	0
1	0	5	6	
2	0	3	4	
3	1	5	9	
4	1	2	2	
5	2	5	1	
6	2	4	-1	
7	3	5	3	
8	4	5	0	

**¿Cómo se ha hecho?** Hemos recorrido las posiciones desde la 1 ( $\text{head}+1$ ) en adelante, comparando el coste de la arista de la posición **head** con las siguientes; si encontramos una arista de menor coste que en **head**, la intercambiamos; así, al terminar, tendremos la arista de menor coste en la posición **head**. Al ser la de menor coste, es la arista candidata para poder entrar en el MST.

- Posicionamos el valor de la cabeza **head** a 1, repitiendo: la arista de menor coste entre las posiciones 1 a m-1 es buscada para que ocupe esa posición, y por tanto, queda:

índice	extremo	extremo	coste	head
0	3	4	-2	1
1	2	4	-1	
2	0	5	6	
3	1	5	9	
4	0	3	4	
5	1	2	2	
6	2	5	1	
7	3	5	3	
8	4	5	0	

**¿Cómo se ha hecho?** De nuevo hemos recorrido las posiciones desde la 2 ( head+1) en adelante, comparando el coste de la arista de la posición head con las siguientes; si encontramos una arista de menor coste que en head, la intercambiamos; así, al terminar, tendremos la arista de menor coste en la posición head. Al ser la de menor coste, es la arista candidata para poder entrar en el MST.

- iv. Ahora posicionamos el valor de la cabeza head a 2, repitiendo: la arista de menor coste entre las posiciones 2 a m-1 es buscada para que ocupe esa posición, y por tanto, queda:

índice	extremo	extremo	coste	head
0	3	4	-2	
1	2	4	-1	
2	4	5	0	2
3	1	5	9	
4	0	5	6	
5	2	4	4	
6	1	2	2	
7	3	5	3	
8	2	5	1	

**¿Cómo se ha hecho?** De nuevo hemos recorrido las posiciones desde la 3 (head+1) en adelante, comparando el coste de la arista de la posición head con las siguientes; si encontramos una arista de menor coste que en head, la intercambiamos; así, al terminar, tendremos la arista de menor coste en la posición head. Al ser la de menor coste, es la arista candidata para poder entrar en el MST.

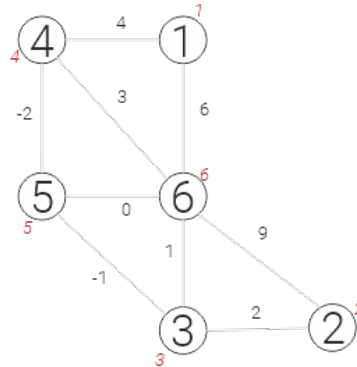
**¿Qué estamos haciendo?** Estamos ordenando el vector de aristas, sí, pero poco a poco, de una forma que deja siempre el menor de las posiciones restantes al principio, identificando la arista que pasaría a ser candidata para el algoritmo de Kruskal. Date cuenta que si el algoritmo de Kruskal finaliza mucho antes de que se recorran todas las aristas, no hará falta ordenarlo completamente.

Controlando las componentes conexas que se van conformando a medida de que se van añadiendo a la solución parcial

# Optimiza!ción

Para controlar que no se formen ciclos, el algoritmo de Kruskal etiqueta a cada nodo con la componente conexas a la que pertenece. Inicialmente, dado que no hay aristas en la solución, cada nodo está en su propia componente conexas que lo tiene a él como único elemento.

Sería el caso de esta imagen que ya hemos usado:



Con lo que tendríamos, inicialmente que:

```
/*Inicializamos el registro de componentes conexas: cada nodo está en su componente conexas*/
```

```
vector <unsigned> Raiz;
```

```
Raiz.resize(n);
```

```
for (unsigned q = 0; q < n; q++)
```

```
{
```

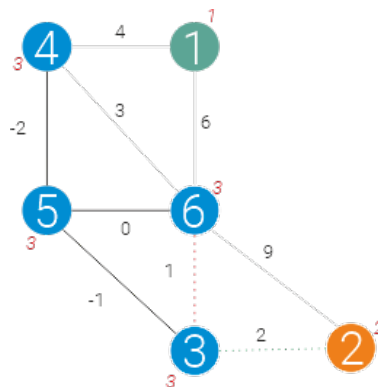
```
    Raiz[q]=q;
```

```
};
```

Y, por tanto, en el vector Raiz, para cada nodo sería, inicialmente:

posición	0	1	2	3	4	5
valor	0	1	2	3	4	5

Añadiendo tres aristas, las de menor coste por orden, tendríamos esta solución parcial:

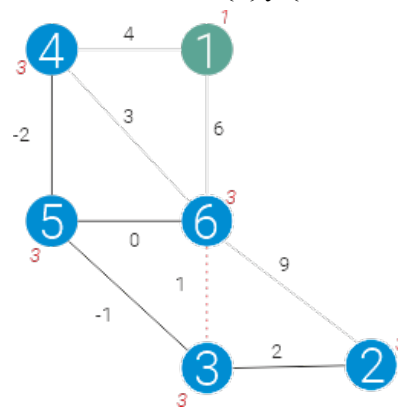


En donde las componentes conexas son, {3, 4, 5, 6}, {2}, y {1}. Por tanto, el vector Raiz sería algo como este:

posición	0	1	2	3	4	5
valor	0	1	2	2	2	2

La arista candidata, (3, 6), tiene sus nodos en las componentes conexas  $\text{Raiz}[3-1]$  y  $\text{Raiz}[6-1]$ , que tienen como valor, 2, ambas. Por tanto, sus nodos están en la misma componente conexas, y no se puede añadir a la solución parcial, pues crearía un ciclo.

La siguiente arista candidata, (2, 3), tiene sus nodos en las componentes conexas,  $\text{Raiz}[2-1]$  y  $\text{Raiz}[3-1]$ , que tienen como valor, 1 y 2, respectivamente. En este caso, sus nodos están en componente conexas distintas, y sí se puede añadir a la solución parcial, pues no crearía un ciclo. La añadimos. En ese caso, hay que actualizar  $\text{Raiz}$  para que refleje que las componentes conexas  $\{2\}$  y  $\{3, 4, 5, 6\}$  se han unido.



Y el vector  $\text{Raiz}$ , sería:

posición	0	1	2	3	4	5
valor	0	2	2	2	2	2

Sólo habría que reemplazar cualquier valor de  $\text{Raiz}[2-1]$  por  $\text{Raiz}[3-1]$ .

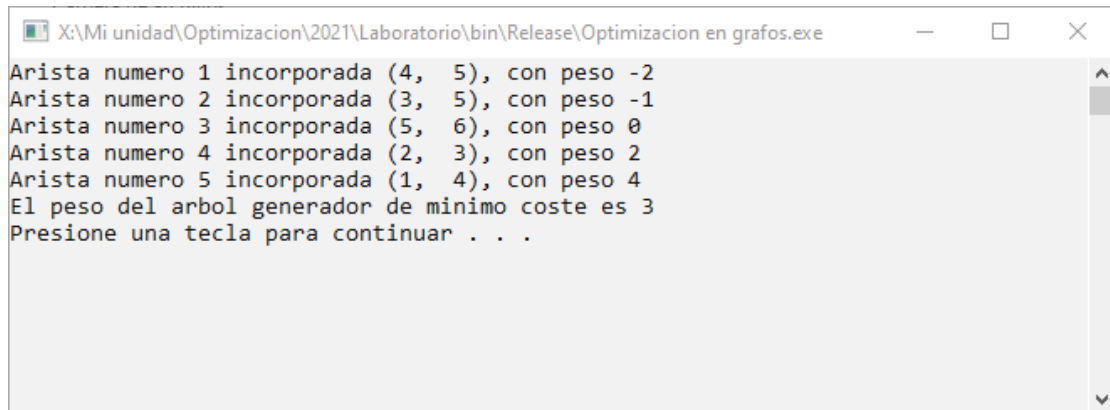
## Métodos a incluir

El único método que hay que implementar es el algoritmo de Kruskal,

```
void GRAFO::Kruskal();
```

incluyendo la opción en el menú de grafos no dirigidos para que se construya el MST mediante este algoritmo.

La resolución del problema se muestra por pantalla: tanto las aristas que forman parte del MST como el peso total del mismo, como se muestra en el siguiente pantallazo:



```
X:\Mi unidad\Optimizacion\2021\Laboratorio\bin\Release\Optimizacion en grafos.exe
Arista numero 1 incorporada (4, 5), con peso -2
Arista numero 2 incorporada (3, 5), con peso -1
Arista numero 3 incorporada (5, 6), con peso 0
Arista numero 4 incorporada (2, 3), con peso 2
Arista numero 5 incorporada (1, 4), con peso 4
El peso del arbol generador de minimo coste es 3
Presione una tecla para continuar . . .
```

## Evaluaci3n

Para superar esta pr3ctica en el laboratorio debe implementarse el algoritmo de Kruskal para la construcci3n del MST, que debe funcionar correctamente.

Para poder acceder al apto+ se evaluar3 la defensa de la pr3ctica por parte del alumno o alumna, las respuestas a las preguntas durante la correcci3n, el c3digo de la pr3ctica y se podr3 plantear una modificaci3n para que sea resuelta durante la correcci3n.

Asimismo, al finalizar la evaluaci3n de la pr3ctica, si es apto-, apto o apto +, el alumno responder3 a un corto cuestionario para evaluar sus conocimientos sobre la pr3ctica y los contenidos te3ricos que la sustentan.