

# OPTIMIZA!CIÓN

## Actividad III: Caminos mínimos en un digrafo.

Profesor Responsable: Sergio Alonso

Dificultad: baja

Presentación del plan de trabajo: lunes 11 de mayo

Fecha límite para la entrega de la práctica: jueves 21 de mayo de 2020, 23:59

### Objetivo

El objetivo de esta actividad es añadir nuevas funcionalidades al menú de utilidades sobre grafos de las prácticas anteriores. En este caso, para el caso de grafos dirigidos o *digrafos* estudiaremos un algoritmo para construir los caminos mínimos de un nodo del grafo a todos los demás: **el algoritmo de Dijkstra**.

### Algoritmo de Dijkstra

La estructura del programa a implementar será también la de un menú de opciones, que añadirá a las ya vistas en la actividad anterior: la utilidad de resolución de los problemas de caminos mínimos con el algoritmo ya mencionado. De nuevo el grafo se lee del fichero, siendo los grafos de esta actividad todos dirigidos y con costes o pesos asociados a los arcos.

La clase GRAFO incluye, además, el nuevo método para la resolución de los problemas de caminos mínimos.

```
class GRAFO
{
    unsigned n;           // número de nodos o vértices
    unsigned m;           // número de arcos o aristas
    unsigned dirigido;     // si el grafo es dirigido, 1
    vector<LA_nodo> LS;    // lista de sucesores o de adyacencia
    vector<LA_nodo> LP;    // lista de predecesores
public:
    unsigned Es_dirigido(); // 0 si el grafo es no dirigido, 1 si es dirigido
    GRAFO(char nombrefichero[], int &errorapertura);
    void actualizar(char nombrefichero[], int &errorapertura);
    void Info_Grafo();
    void Mostrar_Listas(int l);
    void ListaPredecesores();
    void ComponentesConexas();
    void dfs(unsigned i, vector<bool> &visitado);
    void Kruskal();
    void Dijkstra();
    ~GRAFO();
}
```

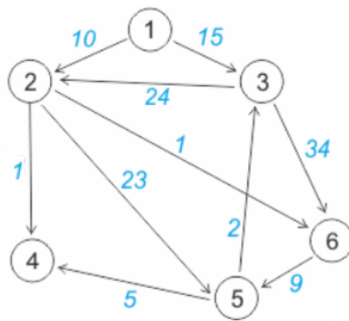
Las definiciones de las estructuras de datos necesarias, las de la clase GRAFO y otras comunes, se almacenarán en el fichero de cabeceras **grafo.h** que se aporta en la carpeta de ficheros de la actividad en el campus virtual.

Es muy conveniente revisar los métodos de carga de datos del fichero de texto, la construcción de las estructuras de listas de los grafos y la salida por pantalla de esta información, para que tengan en cuenta el coste asociado a los arcos o aristas.

### Cómo guardar los caminos mínimos

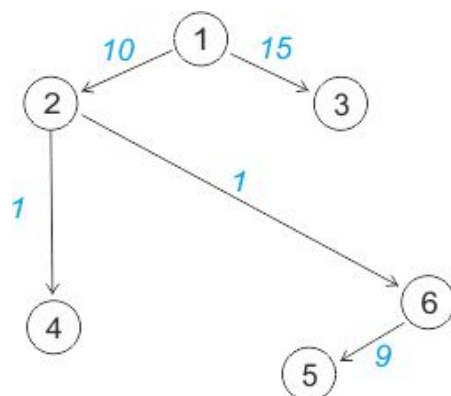
Usaremos el grafo siguiente para ilustrar los elementos de básicos de los problemas de caminos mínimos, en lo que afecta al desarrollo de la presente actividad. Se encuentra codificado como `grafo2.gr` en la biblioteca de recursos del campus virtual de la asignatura.

```
6 10 1
1 2 10
1 3 15
2 4 1
2 5 23
2 6 34
3 2 24
3 6 34
5 3 2
5 4 5
6 5 9
```



Los caminos mínimos entre pares de nodos de un grafo dirigido, han de cumplir la siguiente condición necesaria y suficiente *los subcaminos que contiene deben ser también mínimos*. Por ello, realmente, al construir un camino mínimo entre un par de nodos, estamos construyendo los caminos mínimos de los nodos que atraviesa. Tal es así, que si usamos lo

arcos que participan en los caminos mínimos de un nodo al resto, éstos forman una arborescencia con raíz el nodo origen, es decir, el nodo 1. En el caso del grafo que nos ocupa, sería esta imagen solución.



Usemos esta ventaja para facilitar el almacenamiento de la información sobre los caminos mínimos, pues, si bien comprobamos que en la arborescencia, un nodo puede tener más de un sucesor, tiene un predecesor único. Por ello, el vector de nodos `pred`, es suficiente para codificar la arborescencia. De igual forma, hemos de almacenar la distancia acumulada a cada nodo desde el nodo origen, usando también un vector denominado *etiqueta distancia* denominado `d`. La codificación de los valores de la solución del grafo que nos ocupa serían:

nodo	1	2	3	4	5	6
<b>pred</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>2</b>	<b>6</b>	<b>2</b>
<b>d</b>	<b>0</b>	<b>10</b>	<b>15</b>	<b>11</b>	<b>20</b>	<b>11</b>

Por ello, el siguiente procedimiento recursivo, recorre el vector `pred`, mostrando los caminos mínimos almacenados:

```
void MostrarCamino(unsigned s, unsigned i, vector<unsigned> pred)
{
    if (i != s)
    {
        MostrarCamino(s, pred[i], pred);
        cout << pred[i]+1 << " -> ";
    }
}
```

## Algoritmo

Los algoritmos para la construcción de caminos mínimos en un grafo desde un nodo a todos los demás, se basan en la búsqueda ordenada de mejoras de los valores de la etiqueta distancia, *d*, o lo que es lo mismo, la búsqueda de *atajos* a los caminos ya establecidos.

**El algoritmo de [Dijkstra](#)**, resuelve el problema de forma óptima cuando no hay costes negativos en los arcos del grafo. El método a incorporar en el fichero **grafo.cpp** tendría el esquema siguiente:

```
void GRAFO::Dijkstra()
{
    vector<bool> PermanentementeEtiquetado;
    vector<int> d;
    vector<unsigned> pred;
    int min;
    unsigned s, candidato;

    //Inicialmente no hay ningun nodo permanentemente etiquetado
    PermanentementeEtiquetado.resize(n, false);
    //Inicialmente todas las etiquetas distancias son infinito
    d.resize(n, maxint);
    //Inicialmente el pred es null
    pred.resize(n, UERROR);

    //Solicitamos al usuario nodo origen
    cout << endl;
    cout << "Caminos minimos: Dijkstra" << endl;
    cout << "Nodo de partida? [1-" << n << "]: ";
    cin >> (unsigned &) s;
    //La etiqueta distancia del nodo origen es 0, y es su propio pred
    d[--s]=0; pred[s]=s;
    do
    {
        - Buscamos un nodo candidato a ser permanentemente etiquetado: aquel de
          entre los no permanentemente etiquetados con menor etiqueta distancia no
          infinita.
        - Si existe ese candidato, lo etiquetamos permanentemente y usamos los arcos
          de la lista de sucesores para buscar atajos.
        - Esto lo hacemos mientras haya candidatos

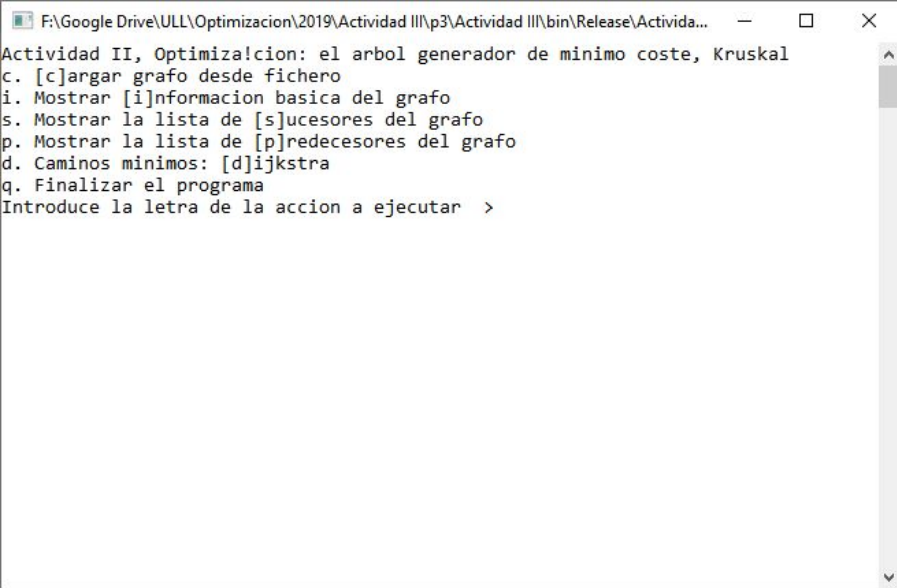
    }
    while (min < maxint);

    cout << "Soluciones:" << endl;
    //En esta parte del código, mostramos los caminos mínimos, si los hay
}
```

Es decir, en la dinámica de uso de la opción d del menú, se le solicita al usuario el nodo origen de los caminos, el nodo distinguido s, y a partir de ahí se ejecuta el algoritmo de Dijkstra. La solución, no sólo debe indicar el camino mínimo, los nodos que atraviesa y su coste, sino, en caso de no existir el camino entre s y ese nodo en particular, advertirlo.

## Ficheros

De igual forma que en la actividad anterior, además del fichero de cabecera `grafo.h` y con el código de desarrollo de sus métodos y procedimientos, **grafo.cpp**, se trabajará con **pg3.cpp** que incluye el programa principal main, que será un simple gestor tipo menú. Las pantallas con las opciones dependerán del tipo de grafo que sea, así, para un grafo dirigido, mostrará ahora:



```
F:\Google Drive\ULL\Optimizacion\2019\Actividad III\p3\Actividad III\bin\Release\Activa...
Actividad II, Optimización: el árbol generador de mínimo coste, Kruskal
c. [c]argar grafo desde fichero
i. Mostrar [i]nformación básica del grafo
s. Mostrar la lista de [s]ucesores del grafo
p. Mostrar la lista de [p]redecesores del grafo
d. Caminos mínimos: [d]ijkstra
q. Finalizar el programa
Introduce la letra de la acción a ejecutar >
```

Mientras que, para un grafo no dirigido, no cambiará respecto a la actividad anterior.

Para el grafo 2 estudiado, la solución mediante el **algoritmo de Dijkstra** sería, desde el nodo 1:

```
F:\Google Drive\ULL\Optimizacion\2019\Actividad III\p3\Actividad III\bin\Release\Activida...
Camino minimo: Dijkstra
Nodo de partida? [1-6]: 1
Soluciones:
El camino desde 1 al nodo 2 es : 1 -> 2 de longitud 10
El camino desde 1 al nodo 3 es : 1 -> 3 de longitud 15
El camino desde 1 al nodo 4 es : 1 -> 2 -> 4 de longitud 11
El camino desde 1 al nodo 5 es : 1 -> 2 -> 6 -> 5 de longitud 20
El camino desde 1 al nodo 6 es : 1 -> 2 -> 6 de longitud 11
Presione una tecla para continuar . . .
```

## Evaluación

Para superar esta actividad, los métodos para los algoritmos deberán estar correctamente implementados y deberán funcionar correctamente las opciones del menú. En ese caso, el alumno o la alumna obtendrá un apto+. Los errores que se manifiesten irán penalizando la calificación, hasta el apto -.