

Optimiza!ción

Práctica 5, Caminos mínimos en un digrafo, la estrategia de Dijkstra

Profesor Responsable: Sergio Alonso

Dificultad: baja

Tutorización: semana del 1 de mayo

Corrección: semana del 8 de mayo

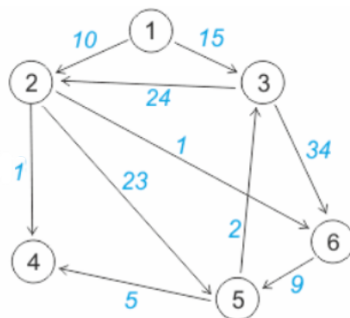
1. Objetivo

En esta actividad es añadiremos nuevas funcionalidades al menú de utilidades sobre grafos de las prácticas anteriores. En este caso, para el caso de grafos dirigidos o *digrafos* el objetivo es plantear una comparativa de ejecución entre **el algoritmo de Dijkstra** que construye los caminos mínimos de un nodo al resto en un grafo con costes no negativos, y del **algoritmo de Bellman, Ford y End**, que resuelve el mismo problema pero aplicando la esencia de su condición de optimalidad.

2. Sobre cómo guardar caminos mínimos

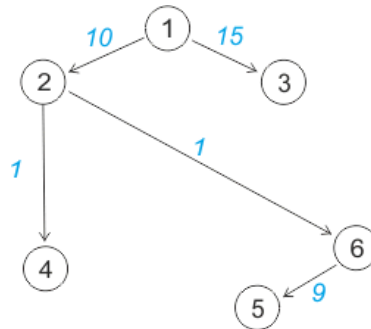
Usaremos el grafo siguiente para ilustrar los elementos básicos de los problemas de caminos mínimos de un nodo a todos los demás, en lo que afecta al desarrollo de la presente actividad. Su codificación sería la siguiente:

```
6 10 1
1 2 10
1 3 15
2 4 1
2 5 23
2 6 1
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, cumplen la siguiente condición necesaria y suficiente: los subcaminos que contiene deben ser también mínimos.

Con ello, realmente, al construir un camino m3nimo entre un par de nodos, estamos construyendo todos los caminos m3nimos de los nodos que atraviesa. Tal es as3, que si usamos s3lo los arcos que participan en los caminos m3nimos de un nodo al resto, 3stos forman una arborescencia con ra3z el nodo origen, en este caso, el nodo 1. En el caso del grafo que nos ocupa, ser3a esta imagen soluci3n.



Usemos esta ventaja para facilitar la codificaci3n de la informaci3n necesaria sobre los caminos, pues, si bien comprobamos que en la arborescencia, un nodo puede tener m3s de un sucesor, tiene un predecesor 3nico. 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 tambi3n un vector donde guardamos la *etiqueta distancia* denominado d, tal que, d[i] almacena el coste del mejor camino encontrado del nodo origen al nodo i.

La codificaci3n de los valores de la soluci3n del grafo que nos ocupa ser3an:

nodo	1	2	3	4	5	6
pred	1	1	1	2	6	2
d	0	10	15	11	20	11

El siguiente procedimiento recursivo usa la propiedad del vector pred recorri3ndolo y, mostrando los caminos m3nimos almacenados:

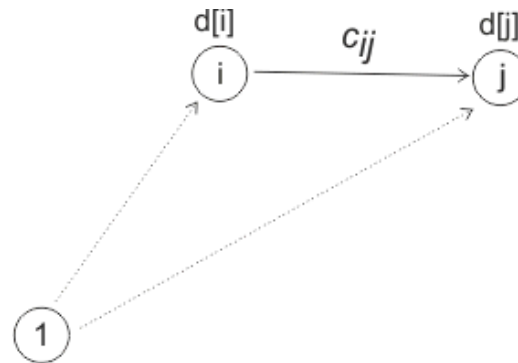
```

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

3. Condici3n de optimalidad de los problemas de caminos m3nimos

Los algoritmos para la construcci3n de caminos m3nimos en un grafo desde un nodo a todos los dem3s, se basan en la b3squeda 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. En la imagen siguiente se muestra la base de las mejoras de la etiqueta distancia:



El razonamiento que hacemos es el siguiente:

- desde el nodo 1 puedo llegar al nodo i y me supone un coste de $d[i]$.
- desde el nodo 1 puedo llegar al nodo j y me supone un coste de $d[j]$.
- decimos que el arco (i, j) con coste c_{ij} es un atajo si $d[i] + c_{ij} < d[j]$, esto es, el coste de llegar a j a través de i es mejor que el camino que tenía construido hasta el momento.

Por todo ello, la condición de optimalidad del problema del camino mínimo de un nodo al resto se cumple cuando no hay arcos que puedan ser atajo, esto es, que mejoren las etiquetas distancia calculadas hasta ese momento.

Los algoritmos de caminos mínimos plantean distintas estrategias para, usando la comparativa de los atajos, llegar a la condición de optimalidad, esto es, no hay más atajos. A continuación detallamos los dos algoritmos que vamos a comparar.

3.1. El algoritmo de Dijkstra (1956)

Partiendo de la conclusión previa, insistimos en que los algoritmos para la construcción de caminos mínimos en un grafo, se basan en la búsqueda ordenada de mejoras de los valores de la etiqueta distancia d , esto es, la búsqueda de atajos a los caminos ya establecidos.

El algoritmo de Dijkstra, resuelve el problema de construir los caminos mínimos de un nodo al resto 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_(double &comparaciones, unsigned s)
{
    vector<bool> PermanentementeEtiquetado;
    vector<int> d;
    vector<unsigned> pred;
    int min;
    unsigned 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);

    //La etiqueta distancia del nodo origen es 0, y es su propio pred
    d[s]=0; pred[s]=s; comparaciones = 0;
    do
    {
        - Buscamos un nodo candidato a ser permanentemente etiquetado: aquel
          de entre los no permanentemente etiquetados, es decir, en el almacén 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. Por cada
          comparación realizada para buscar atajos, incrementamos el contador de
          comparaciones.
        - Esto lo hacemos mientras haya candidatos

    }
    while condición de parada;

    cout << "Soluciones:" << endl;
    En esta parte del código, mostramos los caminos mínimos para cada nodo si
    los hay.

}

```

3.2. El algoritmo de Bellman Ford End (1959)

Este algoritmo fue desarrollado de forma independiente, por Richard Bellman, Lester Ford y Samuel End. El **algoritmo de BellmanFordEnd** usa otra estrategia que deriva en mayor número de comparaciones que el algoritmo anterior, pero permite dar soluciones cuando hay costes negativos en los arcos, y es, además, capaz de detectar cuando el grafo contiene un ciclo de coste negativo. El método a incorporar en el fichero grafo.cpp tendría el esquema siguiente:

```

void GRAFO::BellmanFordEnd_(double &comparaciones, unsigned s)
{
    vector<int> d;
    vector<unsigned> pred;
    unsigned numeromejoras = 0;
    bool mejora;

    //Idem que en el algoritmo de Dijkstra
    d.resize(n,maxint);
    pred.resize(n,UERROR);

    d[s]=0; pred[s]=s; comparaciones = 0;

do
{
    // recorremos todos los arcos, y para cada (i, j), buscamos si d[j] > d[i]
    + cij, y actualizamos d y pred, incrementando el contador comparaciones
    cuando comparamos, independientemente de si mejoramos o no.
    //si al menos en una ocasion ha mejorado una etiqueta distancia, no hemos
    terminado; contabilizamos los bucles en los que ha habido mejoras
}
while ((numeromejoras < n) && (mejora == true));
//para salir del bucle, si mejora es true, pues hay un ciclo, pues hemos
realizado n+1 veces la relajacion con mejora; si mejora es false, pues
tenemos solucion

//Mostramos los caminos mínimos que se puedan haber encontrado, o
advertimos de la existencia de un ciclo de coste negativo.

}

```

3.3. Resultados a mostrar: un ejemplo

Una vez codificados ambos métodos, a situar en la parte privada de la clase, debe incorporarse en la sección pública de la clase el método siguiente:

```
void GRAFO::ComparativaCM ()
```

que será encargado de solicitar al usuario el nodo de partida, ejecutar ambos algoritmos usando los métodos anteriores, y establecer la relación entre el número de comparaciones, esto es, búsqueda de atajos, que ambos hacen para resolver el problema, mostrando el resultado.

Por tanto, 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 la modificación del programa principal main, que, al igual que en las prácticas anteriores, será un simple gestor tipo menú con una opción más en el menú de grafos dirigidos.

```

"X:\Mi unidad\Optimizacion\2022\Laboratorio\bin\Debug\Optimizacion en grafos.exe"
Laboratorio del segundo bloque: Optimización en Grafos
c. [c]argar grafo desde fichero
i. Mostrar [i]nformacion básica del grafo
s. Mostrar la lista de [s]ucesores del grafo
e. Mostrar la lista de [p]redecessores del grafo
y. Mostrar la matriz de ad[y]acencia del grafo
m. Realizar un recorrido en a[m]plitud del grafo desde un nodo por sucesores
r. Realizar un recorrido en p[r]ofundidad del grafo desde un nodo por sucesores
d. Caminos mínimos: [d]ijkstra
v. Caminos mínimos: Comparamos Dijkstra [v]s BellmanFordEnd
f. Caminos mínimos: [f]loyd-warshall
q. Finalizar el programa
Introduce la letra de la accion a ejecutar >

```

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

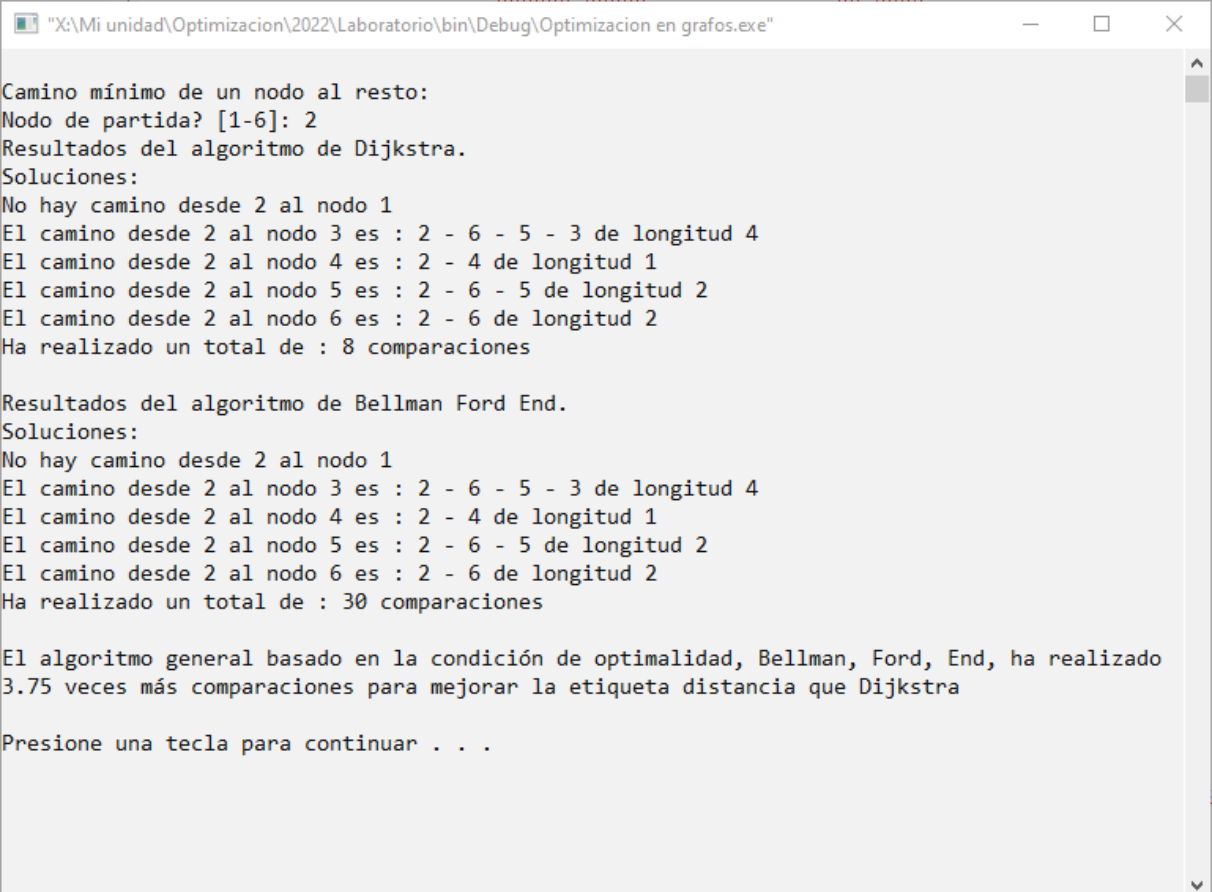
Para el grafo siguiente

```

6 10 1
1 2 3
1 3 2
2 4 1
2 5 3
2 6 2
3 2 1
3 6 3
5 3 2
5 4 1
6 5 0

```

buscamos los caminos mínimos desde el nodo 2 al resto, comparando ambos algoritmos, y la pantalla como el resultado es la siguiente:



```
"X:\Mi unidad\Optimizacion\2022\Laboratorio\bin\Debug\Optimizacion en grafos.exe"

Camino mínimo de un nodo al resto:
Nodo de partida? [1-6]: 2
Resultados del algoritmo de Dijkstra.
Soluciones:
No hay camino desde 2 al nodo 1
El camino desde 2 al nodo 3 es : 2 - 6 - 5 - 3 de longitud 4
El camino desde 2 al nodo 4 es : 2 - 4 de longitud 1
El camino desde 2 al nodo 5 es : 2 - 6 - 5 de longitud 2
El camino desde 2 al nodo 6 es : 2 - 6 de longitud 2
Ha realizado un total de : 8 comparaciones

Resultados del algoritmo de Bellman Ford End.
Soluciones:
No hay camino desde 2 al nodo 1
El camino desde 2 al nodo 3 es : 2 - 6 - 5 - 3 de longitud 4
El camino desde 2 al nodo 4 es : 2 - 4 de longitud 1
El camino desde 2 al nodo 5 es : 2 - 6 - 5 de longitud 2
El camino desde 2 al nodo 6 es : 2 - 6 de longitud 2
Ha realizado un total de : 30 comparaciones

El algoritmo general basado en la condición de optimalidad, Bellman, Ford, End, ha realizado
3.75 veces más comparaciones para mejorar la etiqueta distancia que Dijkstra

Presione una tecla para continuar . . .
```

4. Evaluación

Para superar esta práctica en el laboratorio, los métodos para los algoritmos deberán estar correctamente implementados y deberá funcionar la opción del menú. Para poder acceder al apto+ se evaluará la defensa de la práctica por parte del alumno o alumna, la respuestas a las preguntas durante la corrección, el código de la práctica y se podrá plantear una modificación para que sea resuelta durante la corrección.