ESTRUCTURAS DE DATOS II

Grado en Ingeniería Informática

CURSO 2019/20

Práctica 4

Grafos

Objetivos

- Afianzar los conocimientos adquiridos en teoría en la resolución de problemas usando grafos

Duración

3 sesiones

Observaciones

- Los grafos considerados en esta práctica son dirigidos y etiquetados (mediante valores reales mayores que 0).
- La implementación usada se ha basado en una matriz de adyacencia, de forma que el número máximo de vértices es determinado al principio de la creación del grafo (con su constructor).

Ejercicio 1

Escribir una función que devuelva el vértice de un grafo con el máximo "coste total de salida" (suma de los costes de las aristas que parten de un vértice). Si hubiera varios vértices con dicho máximo coste, devolver cualquiera de ellos:

```
template <typename T>
T verticeMaxCoste(const Grafo<T, float>& G)
```

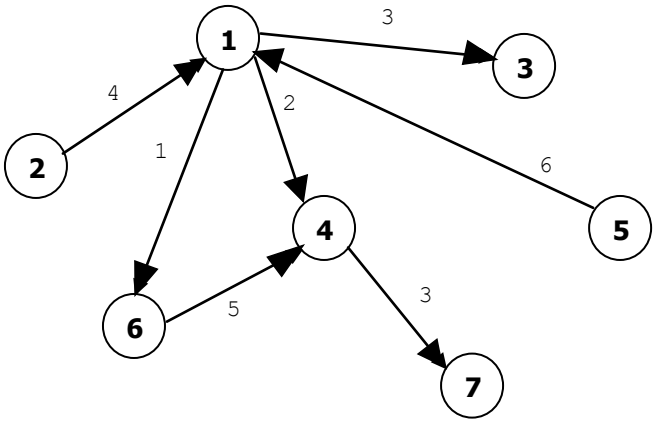
En el grafo representado en la figura del ejercicio 2, la función podría devolver el vértice **1** o el **5**, ya que ambos tienen el mismo coste total de salida (6), que es el máximo.

Ejercicio 2

Dado un grafo, escribe una función que muestre por pantalla los vértices inaccesibles del mismo. Se entiende por *vértice inaccesible* aquél al que no se puede acceder desde ningún otro vértice del grafo. Por tanto, cualquier vértice que no sea destino de ninguna arista es un vértice inaccesible del grafo.

```
template <typename T, typename U>
void inaccesibles(const Grafo<T, U>& G)
```

En el grafo representado en la figura siguiente, los vértices inaccesibles son el **2** y el **5**.



Ejercicio 3

Dado un grafo dirigido (G) y dos vértices (vo, vd), escribe una función que indique si existe o no un camino entre dichos vértices.

```
template <typename T, typename U>
bool caminoEntreDos(const Grafo<T, U>& G, const T& vo, const T& vd)
```

NOTA: Resolver este ejercicio con un algoritmo NO recursivo, y haciendo uso del TAD Cola proporcionado por la STL (***queue***).

Ejercicio 4

Escribir un procedimiento que dados un grafo y un vértice del mismo, muestre por pantalla todos los caminos que partan de dicho vértice y tengan un coste total no superior a un valor dado (maxCoste)

```
template <typename T>
void caminosAcotados(const Grafo<T, float>& G, const T& u, float maxCoste)
```

Nota: Para resolver este ejercicio, se recomienda pensar en un algoritmo **recursivo**

Ejercicio 5

Escribir una función que devuelva un vértice outConectado de un grafo. Se considera que un vértice está outConectado si tiene un mayor número de de conexiones de salida que de entrada. Si hubiera varios vértices outConectados, la función puede devolver cualquiera de ellos.

```
template <typename T, typename U>
T outConectado(const Grafo<T, U>& G)
```

Por ejemplo, en el grafo del ejercicio 2, el vértice **1** estaría outConectado.

Nota: Para resolver este ejercicio, se puede suponer que al menos hay un vértice outconectado

Ejercicio 6

Escribir un procedimiento que muestre por pantalla el *recorrido en profundidad* a partir de un grafo y un vértice dados, pero sin utilizar ninguna estructura auxiliar (pila o cola). Se trata de escribir un algoritmo **recursivo**.

```
template <typename T, typename U>
void recorrido_profundidad(const Grafo<T, U>& G, const T& v)
```

NOTA

Para resolver estos ejercicios se proporcionan los ficheros: **grafo.h**, **grafo.o**, **vertice.h**, **vertice.o**, **arista.h**, **arista.o**, **conjunto.h**, **conjunto.o, excep_grafo.h, excep_conjunto.h, excepción.h** y **practica4.cpp**