Algoritmos y Estructuras de Datos (VJ1215) - Universitat Jaume I
Examen final - 2022/2023 - Segunda parte

24 de enero de 2023

Nombre:

Esta prueba consta de dos partes con una duración de dos horas cada una. Este es el enunciado de la segunda parte, que suma 5 puntos. La prueba es individual. No puedes consultar libros, apuntes ni dispositivos electrónicos. Al finalizar entrega tu solución junto con el enunciado (no es necesario que entregues todas las hojas). Pon tu nombre en cada hoja que entregues.

En los ejercicios 6, 7 y 9, escribe tus soluciones empleando el lenguaje C++.

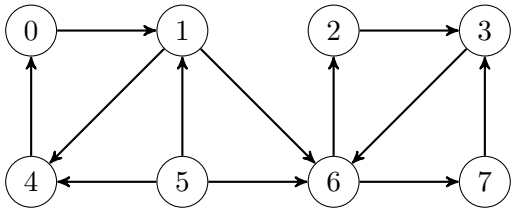
En los ejercicios 6 y 7 debes decidir qué algoritmo utilizar para resolver eficientemente el problema planteado, e implementarlo añadiendo el método que se pide a la siguiente clase utilizada para representar grafos dirigidos en los ejercicios de la asignatura. No puedes añadir ahí otros atributos. Puedes hacer uso de los Tipos Abstractos de Datos *Pila*, *Cola* y *Cola de Prioridad*, con las operaciones que necesites, sin tener que implementarlas (puedes poner sus nombres en inglés o en castellano). No puedes hacer uso de otros métodos o funciones si no los implementas.

```
class GrafoDirigido {
   struct Arco {
      int vecino;
      float peso;
      Arco * siguiente;
      Arco(int, float, Arco *);
   };
   struct Vertice {
      Arco * primerArcoDeEntrada;
      Arco * primerArcoDeSalida;
      int gradoDeEntrada;
      int gradoDeSalida;
      Vertice();
   };
   vector<Vertice> vertices;
 public:
   ...
};
```

Ejercicio 6 — 1,5 puntos

Sea $G = (V, E)$ un grafo dirigido que representa posibles desplazamientos en un juego. Queremos elegir un subconjunto de vértices para situar un jugador (y solo uno) en cada uno de los vértices elegidos de modo tal que se cumplan dos condiciones: (i) todo jugador sea alcanzable desde todos los demás jugadores, y (ii) la cantidad de jugadores sea la máxima posible.

Por ejemplo, con el siguiente grafo situaríamos 4 jugadores en los vértices 2, 3, 6 y 7:



Implementa un método eficiente

```
vector<int> elegirPosicionesJugadores() const
```

que resuelva ese problema y devuelva como resultado los vértices elegidos.

¿Cuál es el coste temporal de tu solución en el peor caso en función de $|V|$ y $|E|$? No es necesario que lo justifiques.

Ejercicio 7 — 1,5 puntos

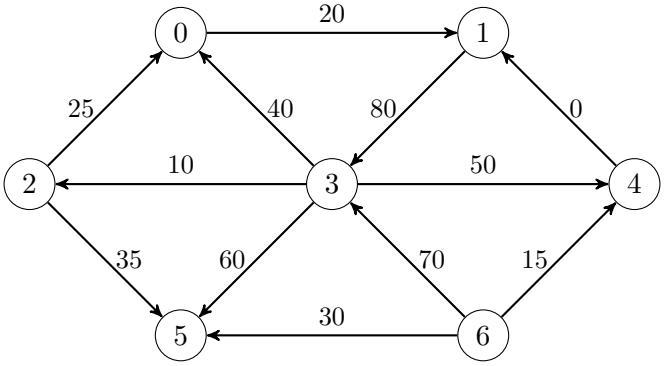
Sea $G = (V, E)$ un grafo dirigido ponderado que representa posibles desplazamientos en un juego. El grafo puede tener ciclos. El peso de cada arco $(u,\ v)$ es una cantidad no negativa que se corresponde con la energía que consume el jugador al moverse del vértice u al vértice v utilizando ese arco. Al desplazarse desde un vértice hasta otro siguiendo un camino en el grafo se consume una energía dada por la suma de los pesos de los arcos utilizados.

Implementa un método eficiente

```
int elegirDestino(int s, float e, const vector<float> & puntos) const
```

que recibe un vértice origen s, una cantidad de energía $e > 0$ y un vector *puntos* de talla $|V|$ que contiene una puntuación asociada a cada vértice del grafo. El método debe devolver un vértice con máxima puntuación entre los que se pueden alcanzar desde s consumiendo una energía total menor o igual que e. Como caso particular, la energía que se consume para llegar a s desde s es 0, y podría suceder que el vértice alzanzable de máxima puntuación sea el propio s.

Por ejemplo, con el siguiente grafo



con $s = 3$, $e = 45$ y este vector *puntos*:

0	1	2	3	4	5	6
200	700	400	100	300	500	600

el resultado sería 5, porque los vértices alcanzables desde el 3 consumiendo una cantidad de energía menor o igual que 45 son 0, 2, 3 y 5, y el vértice 5 es el de mayor puntuación de esos cuatro.

Se valorará que la solución no siga recorriendo el grafo si, con lo visto hasta ese momento, ya se puede determinar el resultado.

Ejercicio 8 — 0,5 puntos

Aplica al siguiente vector el algoritmo que permite construir un montículo binario de mínimos (*min-heap*) a partir de n datos iniciales dados con coste temporal $O(n)$. Dibuja, en el orden en que sucede, el árbol que se obtiene tras cada paso del algoritmo (considerando que intercambiar las posiciones de dos datos es un paso).

7	5	9	8	4	6	3	1	2

Ejercicio 9 — 1,5 puntos

El escenario de un juego está formado por una matriz rectangular de celdas de tamaño $f \times c$. El juego tiene las siguientes reglas:

1. El jugador debe empezar en la celda de la esquina superior izquierda, que es la celda $(0, 0)$, y debe terminar en la celda de la esquina inferior derecha, que es la celda $(f - 1, c - 1)$.

2. Solo se permiten dos tipos de movimientos para desplazarse de una celda a otra: saltar en línea recta hacia abajo, o saltar en línea recta hacia la derecha.

3. Cada celda (i, j), para i desde 0 hasta $f - 1$ y para j desde 0 hasta $c - 1$, tiene asociado un valor entero $saltoMáximo[i][j] \geq 0$ que indica la distancia máxima a la que puede saltar el jugador cuando se encuentra en esa celda: desde la celda (i, j) puede saltar en línea recta hacia abajo hasta cualquier celda $(i + k, j)$, o en línea recta hacia la derecha hasta cualquier celda $(i, j + k)$, siempre que $k \leq saltoMáximo[i][j]$ y que la celda exista.

Necesitamos una función que calcule la mínima cantidad de saltos necesarios para llegar desde la celda inicial hasta la celda final:

```
int minimosSaltos(const vector< vector<int> > & saltoMaximo)
```

Por ejemplo, con $f = 7$ filas, $c = 6$ columnas y esta matriz *saltoMáximo*:

fila \ **columna**	0	1	2	3	4	5
0	3	1	4	0	2	1
1	5	5	3	4	4	1
2	4	4	0	4	1	2
3	5	4	1	2	0	1
4	2	1	3	0	2	1
5	4	0	1	1	1	1
6	1	3	2	2	1	0

el resultado sería 4 saltos, y correspondería a hacer un salto hacia abajo de distancia 2 desde la celda $(0, 0)$ hasta la $(2, 0)$, seguido de un salto hacia la derecha de distancia 3 hasta la $(2, 3)$, a continuación un salto hacia abajo de distancia 4 hasta la $(6, 3)$ y finalmente un salto de distancia 2 hacia la derecha hasta la $(6, 5)$:

fila \ **columna**	0	1	2	3	4	5
0	**3**	1	4	0	2	1
1	5	5	3	4	4	1
2	**4**	4	0	**4**	1	2
3	5	4	1	2	0	1
4	2	1	3	0	2	1
5	4	0	1	1	1	1
6	1	3	2	**2**	1	**0**

Empleando Programación Dinámica, diseña e implementa un algoritmo eficiente para resolver el problema a) de forma recursiva y b) de forma no recursiva.

Indica el coste temporal y espacial de cada una de tus dos soluciones en el peor caso, en función de f y c.

	Recursiva	**No recursiva**
Coste temporal	$O($ $)$	$O($ $)$
Coste espacial	$O($ $)$	$O($ $)$