Algoritmos y Estructuras de Datos (VJ1215) - Universitat Jaume I

Examen final - 2024/2025 - Primera parte
Recuperación del tercer examen parcial

21 de enero de 2025

Nombre:

El examen final consta de dos partes, cada una con una duración de dos horas. La primera parte está destinada a aquellos estudiantes que han solicitado intentar mejorar la calificación obtenida en cualquiera de los exámenes parciales, renunciando a la misma. Este tercer ejercicio de la primera parte está dirigido a quienes buscan mejorar la nota del tercer examen parcial, y tiene una duración de 50 minutos.

La prueba es individual. No puedes consultar libros, apuntes ni dispositivos electrónicos. Escribe tu solución empleando el lenguaje C++, y no olvides los costes que se piden. 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.

Ejercicio 3 — 2,5 puntos

En cada apartado de este ejercicio, debes decidir qué algoritmo utilizar e implementarlo en C++, 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.

```
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:
   ...
};
```

Consideremos un grafo dirigido $G = (V, E)$ que representa el escenario de un juego en el que participan varios equipos de jugadores. Cada vértice está asociado a un equipo y solo uno.

Necesitamos añadir a la clase `GrafoDirigido` un método

```
bool componenteMismoEquipo(int jugador, const vector<int> & equipo) const
```

que recibe dos parámetros: *jugador*, el vértice en el que se encuentra el jugador; y *equipo*, un vector de tamaño $|V|$ que, en la posición v, contiene el equipo asociado al vertice v, para cada v desde 0 hasta $|V|-1$. El método debe devolver *true* si, y solo si, todos los vértices que están en la misma componente fuertemente conexa que el jugador son del mismo equipo que el vértice del jugador.

Recuerda que dos vértices u y v están en la misma componente fuertemente conexa si, y solo si, son mutuamente alcanzables, es decir, existe al menos un camino dirigido desde u hasta v y también existe al menos un camino dirigido desde v hasta u.

En cada uno de los siguientes dos apartados, tu solución debe ser eficiente para que pueda recibir puntuación. Además, se valorará especialmente que el algoritmo no siga recorriendo el grafo, y finalice

devolviendo *false*, tan pronto como se pueda determinar que hay un vértice que pertenece a la misma componente fuertemente conexa que el jugador pero no está en el mismo equipo.

a) **[1 punto]** Implementa el método `componenteMismoEquipo` sin utilizar recursión. Si lo necesitas, en este apartado, puedes utilizar las clases `stack`, `queue` y/o `priority_queue` de C++, sin necesidad de implementarlas. Si no recuerdas los nombres de sus operaciones básicas, puedes escribirlos en castellano. Si haces uso de otras funciones, debes implementarlas también sin utilizar recursión.

b) **[1,5 puntos]** Implementa el método `componenteMismoEquipo` utilizando recursión para resolver el problema. Puedes utilizar también bucles. En este apartado, no puedes utilizar las clases `stack`, `queue` ni `priority_queue`, ni implementarlas, ni utilizar, para hacer el papel de esas clases, vectores u otras estructuras de datos.

Indica cuáles son los siguientes costes de tus dos soluciones en función de la cantidad de vértices $|V|$ y la cantidad de arcos $|E|$. No es necesario justificar las respuestas. En el cálculo del coste espacial, no consideres el espacio ocupado por el propio grafo.

	Apartado a	**Apartado b**
Coste temporal en el peor caso	$O($ $)$	$O($ $)$
Coste espacial en el peor caso sin contar el grafo	$O($ $)$	$O($ $)$