Control 2 – 24 de abril de 2017

Apellidos, nombre _____ NIF: _____

Pregunta 1 (4 p.)

Tenemos información privilegiada sobre diferentes inversiones que pueden ofrecernos muy buenos beneficios. Sabiendo que el coste de las inversiones es alto (no nos dejan invertir sólo una parte de su coste), queremos crear un algoritmo que nos permita decidir qué inversión realizar, teniendo en cuenta que sólo tenemos 10000€ para invertir y hay varias opciones que pueden resultarnos interesantes. A continuación, se muestra una tabla con los costes en euros de las inversiones y los posibles beneficios que obtendremos.

	Inversión A	Inversión B	Inversión C	Inversión D
Coste	3000	4000	6000	2000
Posible beneficio	1000	2000	3000	1000

a) (2 puntos) Diseña un algoritmo mediante programación dinámica (sin utilizar ningún lenguaje de programación ni pseudocódigo) e indica cuál sería el beneficio máximo que podríamos obtener con nuestros 10000€ ¿Y con 5000€? ¿Y con 8000€?
b) (0,5 puntos) ¿Cuál sería la complejidad del algoritmo suponiendo que queremos implementarlo de forma genérica?
c) (0,5 puntos) ¿Será la solución a este problema (para otros casos) siempre óptima utilizando programación dinámica?
d) (0,5 puntos) Si, en lugar de utilizar programación dinámica, utilizásemos un algoritmo voraz, ¿tendríamos siempre una solución óptima?
e) (0,5 puntos) Si, en lugar de utilizar programación dinámica, utilizásemos Backtracking, ¿tendríamos siempre una solución óptima?

Pregunta 2 (3 p.)

Las nuevas instalaciones olímpicas de Tokio 2020 necesitan un nuevo tendido eléctrico que les proporcione la energía suficiente para su funcionamiento. Para optimizar la longitud de éste tendido los ingenieros informáticos japoneses han desarrollado una aplicación con el algoritmo de Kruskal. El cual, dado un grafo conexo no dirigido, donde cada arista posee una longitud no negativa, permite calcular el árbol de recubrimiento mínimo.

En el algoritmo de Kruskal se elige una arista del grafo de entre las de menor peso, se añaden paulatinamente las aristas con menor peso siempre que estas no formen un ciclo con las otras ya incorporadas, el proceso termina cuando se han seleccionado n-1 aristas.

a) (1 punto) Explicar, en función de sus características, qué tipo de algoritmo es este.
b) (0,5 puntos) Escribir pseudocódigo para la función heurística.
c) (1,5 puntos) Completar el código Java del método que permita obtener la solución a este problema mediante el algoritmo descrito. Para simplificar se proporcionan varios

métodos auxiliares en clase Grafo para trabajar con los nodos del grafo. La puntación máxima se obtendrá si se consigue una complejidad de O(nlogn), suponiendo que las operaciones auxiliares tienen como complejidad máxima O(log n).

```
public class Grafo {
	class Arista implements Comparable<Arista> {
		int origen, destino;		// nodos de origen y destino
		int peso;			// peso de la arista
		public Arista(int origen, int destino, int peso) {…}
		@Override
		public int compareTo(Arista a2) {…}
	}

	private int numNodos;			// nodos 0 .. numNodos -1
	private int [][] pesos;			// matriz pesos simétrica

	public Grafo() {…	}	// constructor
	// devuelve el número de nodos del grafo
	public int getNumNodos() {…}
	// devuelve todas las aristas del grafo
	public ArrayList<Arista> getTodasAristas() {… }
	// devuelve true si añadiendo Arista a al árobl forma algún ciclo
	public boolean formaCiclo(Arista a, ArrayList<Arista> arbol) {…	}


	public ArrayList<Arista> kruskal(Grafo grafo)
	{
		int n= getNumNodos();
		ArrayList<Arista> arbolRecubrimiento;









		return arbolRecubrimiento;

	}
}
```

Apellidos, nombre ______________________________ NIF: ______________

Pregunta 3 (3 p.)

Tenemos un laberinto representado por una matriz cuadrada (*n x n*), de enteros donde 0 (camino) significa que se puede pasar y 1 (muro) que no se puede pasar.

El punto de *inicio* en el laberinto estará situado siempre en una de las filas de la primera columna y estará representado por una coordenada (Ini_x, Ini_y); y el punto de destino estará situado en la última columna en una de sus filas y estará representado por (des_x, des_y).

Los movimientos posibles son: arriba, abajo, izquierda y derecha. Debemos marcar el camino que se sigue desde el inicio al destino en las casillas. Al final, debe aparecer si se encontró solución o no, y si se encuentra mostrar el número de pasos realizados desde origen a destino.

Escribir el programa en Java, que implemente mediante backtracking la solución al problema. Rellenando los huecos en el siguiente código dado:

```
public class LaberintoUna {
    static int n;              //tamaño del laberinto (n*n)
    static int[][] lab;        //representación de caminos y muros
    static boolean haySolucion;

    static int[] movx= {0,  0,  -1, 1};     // desplazamientos x
    static int[] movy= {-1, 1,  0,  0};     // desplazamientos y

    static int inix;           //coordenada x de la posición inicial
    static int iniy;           //coordenada y de la posición inicial

    static int desx;           //coordenada x de la posición destino
    static int desy;           //coordenada y de la posición destino

    static void backtracking(______________________________) {
        if (______________________________) {
            ______________________________
            ______________________________
            ______________________________
        }
        else
        {
            for (______________________________)
            {
                int u= x+movx[k];
                int v= y+movy[k];

                if (______________________________
                    ______________________________)
                {
                    ______________________________
                    backtracking(________________________);
                    ______________________________
        }
            }
        }
}
```

```
public static void main(String arg[]) {
    // n y lab inicializadas previamente
     inix= 0; iniy= 0; desx= n-1; desy= n-1;
     haySolucion = false;
     lab[inix][iniy] = 2;

     backtracking(                                          );
     if (!haySolucion) System.out.println("NO HAY SOLUCIм");
}
```

b) (1 punto) Qué cambios habría que realizar para devolver el camino más corto para alcanzar la salida. Indícalo modificando el código Java del apartado anterior.