Fundamentos de Algoritmia

Examen de enero Curso 2022/2023

NOMBRE: .....

Normas de realización del examen

1. Debes programar soluciones para cada uno de los tres ejercicios, probarlas y entregarlas en el juez automático accesible en la dirección `http://exacrc`. Para la entrega el juez sólo tiene los datos de prueba del enunciado del problema.

2. Escribe comentarios que expliquen tu solución, justifiquen por qué se ha hecho así y ayuden a entenderla. Calcula la complejidad de todas las funciones que implementes.

3. En el juez te identificarás con el nombre de usuario y contraseña que has recibido al comienzo del examen. El nombre de usuario y contraseña que has estado utilizando durante la evaluación continua **no** son válidos.

4. Escribe tu **nombre y apellidos** en un comentario en la primera línea de cada fichero que subas al juez.

5. Tus soluciones serán evaluadas por el profesor independientemente del veredicto del juez automático. Para ello, el profesor tendrá en cuenta **exclusivamente** el último envío que hayas realizado de cada ejercicio.

Ejercicio 1

El dueño de la frutería de mi barrio está muy preocupado porque el margen de beneficio del negocio es cada vez más pequeño. Lleva meses apuntando cada día lo que gasta en Mercamadrid para adquirir la fruta por la mañana y el importe de lo que vende a lo largo del día. Sabiendo el beneficio que tiene cada día (diferencia entre lo que gana y lo que gasta), ¿podrías calcular la longitud de la secuencia más larga de días consecutivos en los que ha estado perdiendo más de una cierta cantidad? También queremos conocer cuándo comenzó esa *mala racha* y cuál es la máxima pérdida producida durante esos días.

Se pide:

1. Especifica una función que reciba un vector con el beneficio de cada día y calcule la longitud de la secuencia más larga de días consecutivos en los que ha estado perdiendo más de una cierta cantidad.
2. Implementa una función que resuelva el problema pedido, devolviendo los tres datos que se indican en el enunciado. El coste de la implementación debe ser lineal en el número de días.
3. Indica un invariante que permita probar la corrección del algoritmo implementado y justifica que el coste del algoritmo es lineal en el número de días.

Entrada

La entrada consiste en una serie de casos de prueba. Cada caso comienza con una línea en que se indica el número de días de los que se tiene apuntado el beneficio y el límite de la pérdida que se quiere permitir. En la línea siguiente aparece el beneficio de cada día. La entrada termina con un valor 0 que no debe procesarse.

El número de días es mayor que 0 y menor que 100 000. El límite de la pérdida que se quiere permitir es un número negativo en el intervalo [-1000,-1]. El beneficio es un número entero en el intervalo $[-1000, 1000]$.

Salida

Para cada caso de entrada se escribirá el máximo número de días consecutivos en los que se ha perdido más de la cantidad dada, seguido del día en que se comenzó a perder más de esa cantidad y del máximo que se ha perdido durante esos días. Si existen varias rachas con el número máximo de días se elegirá aquella que tenga una pérdida mayor y si existen varias con el número máximo de días y la misma pérdida máxima se elegirá aquella que ocurra en último lugar. Los días comienzan a numerarse en el cero.

Si no existe ningún día en que se hayan producido pérdidas mayores al valor indicado se escribirá `SIN PERDIDAS`.

Entrada de ejemplo

```
9 -5
-7 -8 3 5 -2 -10 -7 -6 -5
4 -4
-4 -4 -4 -4
3 -2
3 -1 3
8 -1
2 -3 -5 -2 -1 -4 -10 -2
8 -1
2 -3 -5 -2 -1 -5 -2 -2
0
```

Salida de ejemplo

```
3 5 -10
SIN PERDIDAS
SIN PERDIDAS
3 5 -10
3 5 -5
```

Ejercicio 2

Los polinomios de Hermite son un ejemplo de polinomios ortogonales que encuentran su principal ámbito de aplicaciones en la mecánica cuántica. Se definen mediante la ecuación de recurrencia:

$$H_n(y) = \begin{cases} 1 & n = 0 \\ 2y & n = 1 \\ 2yH_{n-1}(y) - 2(n-1)H_{n-2}(y) & n > 1 \end{cases}$$

Se pide implementar una función recursiva eficiente que calcule el valor del n-ésimo polinomio de Hermite para un cierto valor de y.

- Explica el algoritmo empleado e impleméntalo.
- Plantea la recurrencia de la solución implementada e indica su coste.

Entrada

La entrada consta de una serie de casos. Cada caso se escribe en una línea y consiste en un número n $(0 \leq n \leq 10)$ seguido del valor de y en el que se quiere $(1 \leq y \leq 20)$. La entrada termina con el valor -1.

Salida

Para cada valor de entrada mostrar en una línea el valor del polinomio pedido.

Entrada de ejemplo

```
0 3
1 6
2 4
3 2
10 20
-1
```

Salida de ejemplo

```
1
12
62
40
9906193528929760
```

Ejercicio 3

Dado un grafo valorado no dirigido $G = (V, A)$ y un entero r, escribir un algoritmo de vuelta atrás que encuentre, si existe, un clique de tamaño r y de *mínimo* coste en G. Un clique es un subgrafo completo, es decir, un subconjunto $V' \subseteq V$ tal que existe una arista entre cada par de vértices en V'. El tamaño de un clique es su número de vértices y su coste es la suma de los costes de sus aristas.

El algoritmo debe seleccionar entre los vértices V del grafo los r que van a pertenecer al clique. Un vértice puede pertenecer al clique si está conectado con todos los otros vértices del clique. Ten en cuenta que, durante la exploración, aquellas ramas en las que el número de vértices sin explorar no sea suficiente para completar el clique deberán ser podadas.

Se pide:

1. Describir el árbol de exploración y la forma del vector solución que se emplean en la solución del problema.
2. Implementar el algoritmo de vuelta atrás que resuelve el problema.
3. Piensa también en una estimación del coste que se pueda utilizar: aunque no la implementes, explícala.

Entrada

La entrada comienza con una línea que contiene el número de casos de prueba. Cada caso de prueba contendrá inicialmente el número de vértices n del grafo ($0 < n < 10$) y el tamaño r del clique ($0 < r \leq n$). A continuación, en una matriz simétrica de dimensión n por n se indica el coste (estrictamente positivo) de la arista entre los vértices correspondientes, o bien un 0 si no son adyacentes.

Salida

Para cada caso de prueba, si existe un clique del tamaño pedido el programa escribirá el coste mínimo y a continuación, separados por blancos, los vértices que lo componen. En caso contrario se escribirá `NO EXISTE`. Estudia el orden en que se deben explorar las ramas del árbol de exploración para que si dos cliques tienen el mismo coste el programa muestre el que tenga el vértice con menor número en primer lugar; si es el mismo, se quedará con el de menor valor en la segunda posición, etc.

Entrada de ejemplo

```
3
3 2
0 1 2
1 0 3
2 3 0
4 4
0 1 2 3
1 0 4 5
2 4 0 6
3 5 6 0
4 3
0 1 0 0
1 0 0 0
0 0 0 2
0 0 2 0
```

Salida de ejemplo

```
1 0 1
21 0 1 2 3
NO EXISTE
```

Se podría obtener una estimación que nos ayude a poder ramas del árbol a partir del valor mínimo de todas las aristas del grafo. Así, si se han escogido ya vértices (es decir, el valor de usados es), lo mejor que podría ocurrir es que el coste de cada una de las aristas que faltan por añadir sea . Este número se obtiene restando a todas las aristas que existen entre r vértices las que existen entre i, con lo que

$$estimacion = coste - actual + (r * (r + 1)/2 - i * (i + 1)/2) * m$$