Fundamentos de Algoritmos. Grupo F

Grados de la Facultad de Informática (UCM)

21 de Junio de 2023

Normas de realización del examen

1. La parte práctica del examen dura **2 horas y 40 minutos**.
2. Debes desarrollar e implementar soluciones para cada uno de los ejercicios, probarlas y entregarlas en el juez automático accesible en la dirección `http://exacrc.fdi.ucm.es`.
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 **TODOS** los dicheros que subas. Si no incluyes el nombre en el fichero el ejercicio se evaluará con un 0.
5. Dispones de un fichero plantilla para cada ejercicio. Debes utilizarlo atendiendo a las instrucciones que se dan en cada fichero y escribiendo tu solución en los espacios reservados para ello, siempre entre las etiquetas `//@ <answer>` y `//@ </answer>`.
6. 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.
7. A lo largo del examen se te pedirá que te identifiques y rellenes tus datos en una hoja de firma.

Primero resuelve el problema. Entonces, escribe el código.

— John Johnson

Comentar el código es como limpiar el cuarto de baño;
nadie quiere hacerlo, pero el resultado es siempre
una experiencia más agradable para uno mismo y sus invitados.

— Ryan Campbell

Ejercicio 1. El casino de René (3.5 puntos)

René anda obsesionado con hundir a la banca del casino local. Tiene dos armas para ello: un chivatazo y sus conocimientos de algoritmos. Un amigo que trabaja en el casino, le ha pasado la secuencia de ganancias y pérdidas que sacará la ruleta esta noche. La secuencia de números que recibe René indica qué ganaría (si es un valor positivo) o perdería (si es un valor negativo) de hacer una apuesta en ese instante. Su intención es obtener cuál es la racha que le proporciona la ganancia máxima.

Como hay cierto riesgo de que le pillen, René ha decidido que sólo hará una apuesta si, al menos, se asegura un beneficio mínimo. Además, si encuentra más de una racha que genere ese beneficio máximo, sólo le interesa aquella que lo haga antes (es decir, que comience antes).

Se pide:

1. (0.75 puntos) Especifica una función que reciba un vector de enteros y un valor $s > 0$, y devuelva tres valores: el valor de la ganancia máxima (siempre que ésta sea al menos s), el inicio y el final del intervalo en la que esto ocurre (ambos índices incluidos en el intervalo).
2. (2 puntos) Diseña e implementa un algoritmo iterativo eficiente que resuelva el problema propuesto y que cumpla la especificación que has hecho.
3. (0.5 puntos) Escribe el invariante del bucle que permite demostrar la corrección del mismo y proporciona una función de cota.
4. (0.25 puntos) Indica y justifica adecuandamente el coste asintótico del algoritmo en el caso peor.

Entrada

La entrada consta de varios casos de prueba. Cada uno de ellos ocupa dos líneas. La primera línea de cada caso indica la longitud N de la lista de entrada ($0 \leq N \leq 10^8$) y el valor S ($0 \leq S$) del beneficio mínimo esperado; y la segunda línea contiene los N elementos de la lista.

Salida

Para cada caso de prueba debe escribirse una línea con los 3 valores que nos interesan: la ganancia máxima, el inicio del intervalo y el final del intervalo (ambos índices incluídos). Si no hay ningún intervalo con beneficio mayor o igual a S, la salida será `NO COMPENSA`.

Entrada de ejemplo

```
5
10 5
4 3 -4 1 5 2 -12 8 3 -1
5 10
-10 11 10 -100 1
3 2
-1 -2 0
7 7
7 0 1 -6 4 3 -2
7 1
8 -8 7 3 -20 2 5
```

Salida de ejemplo

```
11 0 5
21 1 2
NO COMPENSA
9 0 5
10 0 3
```

Ejercicio 2. Apretándose el cinturón (2.5 puntos)

Lucas se acaba de independizar y necesita manejar su presupuesto con cuidado. Sabe que la mayoría de sus gastos se van en la casa y la comida pero quiere asegurarse de que es capaz de reservar lo suficiente como para poder darse algún capricho. ¡Ahorrar en sus hobbies es tan importante que prefiere gastar menos por otro lado antes que no disponer de dinero para ocio!

Lucas cuenta con su presupuesto total, la cantidad de dinero que quiere ahorrar por cada uno de los días, y una estimación de los gastos acumulados por cada uno de los días. Su objetivo es encontrar el primer día en el que debe apretarse el cinturón y empezar a gastar menos en comida porque no le queda suficiente dinero para ahorrar en sus caprichos.

Se pide:

1. (2 puntos) Escribe un algoritmo recursivo eficiente que permita resolver el problema.
2. (0.5 puntos) Escribe la recurrencia que corresponde al coste de la función recursiva. Indica también a qué orden de complejidad asintótica pertenece dicho coste.

Entrada

La entrada comienza con una línea que contiene el número de casos de prueba. Cada caso de prueba costa de dos líneas.

La primera contiene el valor P del presupuesto total; el valor a con la cantidad que se quiere ahorrar por día; y el número n total de días que va a ahorrar (siempre $P \geq n \cdot a$). La segunda línea contiene una secuencia de tamaño n con la estimación del gasto acumulado día a día (una secuencia estrictamente creciente con saltos de, como poco, el valor a).

Salida

Para cada caso de prueba el programa escribirá el día en que debe comenzar a apretarse el cinturón (siendo el primer día el día cero) o `NO NECESARIO` en caso de que no necesite hacerlo.

Entrada de ejemplo

```
4
50 10 5
0 20 30 40 50
50 7 6
3 10 18 26 35 45
32 2 4
27 29 31 33
14 2 6
1 3 6 8 10 12
```

Salida de ejemplo

```
1
5
0
NO NECESARIO
```

Ejercicio 3. Al acecho del dragón (4 puntos)

La prosperidad del reino de Aliahan está basada en una serie de cristales con mágicas propiedades. Tan valiosos son que el malvado Baramos ha invocado un poderoso dragón contra el pequeño reino para hacerse con los cristales. La única esperanza del reino es recurrir a un grupo de valientes héroes y usar el poder de los cristales contra el dragón.

Los cristales permiten aumentar los puntos de ataque de sus portadores, además no es necesario que cada héroe use un único cristal sino que con un cristal se pueden beneficiar hasta 4 héroes.

Contamos con H héroes, cada uno con un *nivel*, *clase* y *puntos de ataque*. El reino de Aliahan cuenta con N cristales. Como buenos objetos mágicos que son, los grupos que se formen entorno a un cristal, deben cumplir dos condiciones:

- El poder de un cristal está limitado a un nivel M. La suma de los niveles de los héroes de un grupo no puede superar ese valor M.
- Como mucho puede haber 4 héroes entorno al cristal y, además, sólo puede haber 2 héroes con la misma *clase* en cada grupo.

El beneficio del cristal es aumentar la suma de los *puntos de ataque* del grupo. En concreto, por cada héroe que esté, los puntos de ataque del grupo aumenta en un 25 %. Por ejemplo, un grupo de 3 héroes tendrá un 75 % más de puntos de ataque: la suma del poder de los 3 miembros más el 75 % de esa cantidad.

Para vencer al dragón la suma de los puntos de ataque totales de todos los grupos debe ser mayor o igual a los puntos de vida del dragón.

El problema ahora mismo es que el dragón se acerca y los cristales mágicos son un bien escaso y necesario para el reino. ¡El reino necesita que los héroes usen el menor número posible de cristales!

Se pide diseñar e implementar un algoritmo de vuelta atrás que resuelva el problema presentado. Describe claramente el espacio de soluciones, los marcadores utilizados y la poda de optimalidad realizada.

Entrada

La entrada comienza con una línea que contiene el número de casos de prueba.

Cada caso de prueba contendrá inicialmente el número H de héroes; el número máximo N de cristales mágicos; el nivel máximo M de cada cristal; y los puntos de vida PV del dragón. A continuación, tres líneas de longitud H: la clase de cada héroe (un número no negativo), el nivel (un número mayor o igual que 1) y los puntos de ataque de cada héroe (un número mayor o igual que 1).

Salida

La salida es el menor número de cristales necesarios para vencer al dragón. En caso de no ser posible hacer una distribución de los cristales que acabe con el dragón, la salida será `DERROTA`.

Entrada de ejemplo

```
4
6 7 2 150
1 2 1 2 1 2
1 1 1 1 1 1
20 20 20 20 20 20
6 7 1 30
1 2 1 2 1 2
1 1 1 1 1 1
5 5 5 5 5 5
4 3 5 100
0 1 1 0
3 4 2 1
15 26 1 57
5 5 5 100
3 1 5 6 8
2 2 1 1 1
10 10 10 10 10
```

Salida de ejemplo

```
3
6
2
DERROTA
```