Fundamentos de Algoritmia

Grados en Ingeniería Informática

Convocatoria ordinaria, 20 de enero de 2022. Grupos B, D y G

1. (3 puntos) Diseña un algoritmo iterativo que, dado un vector de números reales, indique cuántos de ellos son mayores o iguales que un número A y menores o iguales que otro número B ($A \leq B$).

Debes, asimismo, justificar la corrección (precondición, postcondición, invariante, cota) y el orden de complejidad del algoritmo.

La implementación deberá ir acompañada de un programa de prueba, que lea desde la entrada estandar casos de prueba, los ejecute, e imprima por la salida estándar el resultado. Cada caso de prueba consiste en dos líneas:

- La primera línea indica el tamaño `n` del vector (`n` ≤ 10000), seguido de los números A y B.
- La segunda línea contiene los n valores reales del vector.

La lista de casos de prueba termina con un `-1`.

Por cada caso de prueba se escribirá el número de elementos del vector en el rango $[A..B]$.

A continuación se muestra un ejemplo de entrada/salida:

Entrada	Salida
5 2.0 3.5 6 2.53 2.0 10 3.5 2 2 2 2 2.0001 -1	3 1

2. (3 puntos) Una *intromisión* en un vector de enteros **sin elementos repetidos** consiste en insertar en este vector un nuevo valor **que no aparece en el mismo** (el *valor intruso*).

Debes desarrollar un algoritmo "divide y venceras" eficiente que, dados dos vectores `int a[n]` e `int b[n+1]`, donde el vector `b` representa el resultado de llevar a cabo una *intromisión* en el vector `a`, devuelva el *valor intruso*. Debes, así mismo, determinar justificadamente el coste de este algoritmo, planteando y resolviendo las recurrencias apropiadas.

La implementación deberá ir acompañada de un programa de prueba, que lea desde la entrada estandar casos de prueba, los ejecute, e imprima por la salida estándar el resultado. Cada caso de prueba consiste en tres líneas:

- La primera línea indica el tamaño `n` del vector `a` (`n` ≤ 10000).
- La segunda línea contiene los valores del vector `a` (`n` valores).
- La tercera línea contiene los valores del vector `b` (`n + 1` valores).

La lista de casos de prueba termina con un `-1`.

Para cada caso de prueba leído se escribirá el *valor intruso* devuelto por el algoritmo.

A continuación se muestra un ejemplo de entrada/salida:

Entrada	Salida
5 2 1 3 5 4 2 10 1 3 5 4 4 1 3 2 4 1 3 2 4 5 -1	10 5

3. (4 puntos) Dado un conjunto de N procesos, donde cada uno de ellos requiere un tiempo de cómputo t_i y proporciona un beneficio b_i tras su ejecución, y dos procesadores p_1 y p_2 que podemos utilizar durante T unidades de tiempo cada uno, implementa un algoritmo de "vuelta atrás" que encuentre el beneficio máximo que podemos llegar a obtener ejecutando procesos en los procesadores. Para ello se deberán tener en cuenta las siguientes restricciones:

- Los procesos se pueden ejecutar en cualquier orden y en paralelo entre sí (ejecutando uno en un procesador, y el otro en el otro), pero un proceso, en caso de que se ejecute, se ejecutará secuencialmente, en un solo procesador.

- Los procesos se deben ejecutar de manera completa, desde el principio hasta el final (no es posible fragmentar su ejecución).
- Los procesadores son *nononucleo*: en cada instante de tiempo, en un procesador estará ejecutándose, a lo sumo, un único proceso.
- No es necesario ejecutar todos los procesos: se obtendrá el beneficio de aquellos que se ejecuten.
- La suma de los tiempos de cómputo de todos los procesos ejecutados en un procesador no prodrá exceder las T unidades de tiempo de las que se dispone en el procesador.

A modo de ejemplo, supongamos que podemos utilizar cada procesador durante 2 unidades de tiempo ($T = 2$), que tenemos 6 procesos ($N = 6$), que los tiempos de cómputo de cada proceso son $t_1 = 2$, $t_2 = 1$, $t_3 = 1$, $t_4 = 1$, $t_5 = 1$, y $t_6 = 2$, y que los correspondiente beneficios son $b_1 = 5$, $b_2 = 2$, $b_3 = 2$, $b_4 = 2$, $b_5 = 2$ y $b_6 = 1$. Si ejecutamos en el procesador p_1 el primer proceso, y en el procesador p_2 el sexto proceso, obtendremos un beneficio total de 6. Sin embargo, si ejecutamos en p_1 el primer proceso, y en p_2 el segundo y el tercero, obtendremos un beneficio total de 9 (en este caso, este es el beneficio máximo que podemos obtener).

En el desarrollo de este algoritmo, se valorará el uso de una función de poda que disminuya el número de soluciones exploradas para resolver el problema.

La implementación deberá ir acompañada de un programa de prueba, que lea desde la entrada estandar casos de prueba, los ejecute, e imprima por la salida estándar el resultado. Cada caso de prueba se describe en 3 líneas:

- La primera línea contiene dos números enteros que representan las unidades de tiempo T de cómputo disponibles para cada procesador y el número $N \leq 100$ de procesos disponibles.
- Las siguientes dos líneas contienen N números enteros positivos cada una que representan, respectivamente, los tiempos de cómputo t_i y beneficios b_i asociados a cada uno de los procesos.

De esta forma, el caso ejemplificado anteriormente se describirá como:

```
2 6
2 1 1 1 1 2
5 2 2 2 2 1
```

La entrada termina con dos ceros que no se deben procesar.

Para cada caso de prueba se escribirá en una línea diferente el beneficio máximo que se puede obtener.

A continuación se muestra un ejemplo de entrada/salida:

Entrada	**Salida**
2 3 2 1 1 2 2 2 2 5 1 1 1 1 1 1 2 2 1 2 2 4 3 4 1 7 3 5 1 9 6 7 5 3 3 4 5 2 2 9 5 5 4 5 8 3 6 19 5 9 1 8 6 1 2 2 8 7 5 9 5 3 7 4 1 1 9 3 9 8 8 7 8 4 8 6 5 1 7 1 5 3 2 4 3 9 0 0	6 7 1 22 40