DECSAI
Universidad de Granada

Fundamentos de Programación

Examen de Teoría. Convocatoria de Septiembre. Curso 2013/2014

17 de Septiembre de 2014

Tiempo: 2h 30m

1. (**2 puntos**) Queremos saber si dos círculos intersecan. Para ello, basta ver que la distancia entre sus centros debe ser menor o igual que la suma de sus radios (por tanto, supondremos que dos círculos concéntricos se intersecan). Se pide construir las **clases y métodos** necesarios para resolver este problema, teniendo en cuenta lo siguiente:

   - Debe incluir la definición de los datos miembros y la implementación del constructor de todas las clases que necesite.
   - Debe incluir las cabeceras de los métodos necesarios para resolver el problema, pero no tiene que incluir la implementación de dichos métodos.
   - Debe incluir la implementación del método que comprueba la intersección, pero no tiene que incluir la implementación de los métodos invocados dentro de él.

2. **(2 puntos)** Implemente el algoritmo **Counting Sort**, que trabaja sobre un vector de longitud indeterminada de enteros positivos, y dados dos enteros positivos, n y m, devuelve un nuevo vector con los elementos ordenados en orden no decreciente. Los elementos del vector, que denominamos claves, toman valores en el conjunto $\{n, ..., m\}$.

   El proceso a seguir consiste en 1) calcular un vector de frecuencias sobre el rango de las posibles claves y 2) cargar en el nuevo vector las claves tantas veces como indique su frecuencia.

   La longitud del vector de entrada ha de coincidir con el vector de salida.

   Ejemplo: Suponiendo el vector $18, 15, 13, 11, 11, 16, 14$ y los valores entre $n = 11$ y $m = 18$

Claves	11	12	13	14	15	16	17	18
Frecuencias	2	0	1	1	1	1	0	1

   Salida: $11, 11, 13, 14, 15, 16, 18$.

   Los vectores mostrados son a título ilustrativo, pudiéndose usar los vectores que estime oportuno.

3. (**1 punto**) La **Búsqueda por Interpolación** del valor `buscado` en un vector `v` entre las posiciones `izda` y `dcha` es un método avanzado de búsqueda que recuerda a la *búsqueda binaria* porque: a) requiere que el vector en el que se va a realizar la búsqueda está ordenado, y b) en cada consulta sin éxito se descarta una parte del vector para la siguiente búsqueda. La diferencia fundamental con la búsqueda binaria es la manera en que se calcula el elemento del vector que sirve de referencia en cada consulta (que ocupa la posición `pos`). Ya no es el que ocupa la posición central del subvector en el que se efectúa la búsqueda (el delimitado únicamente por `izda` y `dcha`), sino que depende también del contenido de esas casillas, de manera que `pos` será más cercana a `dcha` si `buscado` es más cercano a `v[dcha]` y más cercana a `izda` si `buscado` es más cercano a `v[izda]`. En definitiva, se cumple la relación:

   $$\frac{\texttt{pos} - \texttt{izda}}{\texttt{dcha} - \texttt{izda}} = \frac{\texttt{buscado} - \texttt{v[izda]}}{\texttt{v[dcha]} - \texttt{v[izda]}}$$

   Escribir una función que implemente la búsqueda por interpolación de manera **recursiva**.

4. (**2 punto**) **¿Cuántos 9 hay entre el 1 y el 100?** Pretendemos diseñar un algoritmo que responda a esta sencilla pregunta, pero de forma suficientemente generalizada. Se deberá implementar una función que reciba una cifra, k (entre 1 y 9), distinta de 0, y dos valores, n y m, con $n \leq m$, y devuelva el número de apariciones de la cifra k en los números contenidos en el intervalo cerrado $[n, m]$.

Fundamentos de Programación

Examen Práctico. Convocatoria de Septiembre. Curso 2013/2014

17 de Septiembre de 2014

Tiempo: 1h

1. (**3 puntos**) Ampliar la clase `MiMatrizRectangularInt` con un **método** que busque la fila de la matriz que más se parezca a un vector de enteros (clase `MiVectorInt`) al que llamaremos `referencia`. El método devolverá el *número* de la fila.

   La similitud entre dos vectores $x = (x_1 \cdots x_p)$ e $y = (y_1 \cdots y_p)$ vendrá dada por la distancia euclídea entre ambos:

   $$dist(x, y) = \sqrt{(x_1 - y_1)^2 + \cdots + (x_p - y_p)^2}$$

   Además, la búsqueda sólo se hará sobre las **filas** de la matriz enumeradas en un segundo vector de enteros (clase `MiVectorInt`) llamado `filas_a_comparar`.

   Por ejemplo, dada la matriz `Matriz` (7×4),

→	3	1	0	8
	4	5	1	5
→	5	7	1	7
	7	9	6	1
→	4	9	5	5
→	2	8	2	2
	7	3	2	5

   y los vectores `referencia` $= \{2, 8, 1, 1\}$ y `filas_a_comparar` $= \{0, 2, 4, 5\}$, el método deberá calcular que la fila más cercana al vector `referencia` es la fila 5 (en el dibujo anterior se han marcado con una flecha las filas indicadas por el vector `filas_a_comparar`).

   Adicionalmente, habrá que realizar un programa para comprobar el correcto funcionamiento del método. Los datos de entrada al programa se deben dar en el siguiente orden:

   a) Número de elementos usados del vector `referencia`.
   b) Elementos del vector `referencia`: una secuencia de números enteros (tantos como elementos usados tenga el vector).
   c) Número de elementos usados del vector `filas_a_comparar`.
   d) Elementos del vector `filas_a_comparar`: una secuencia de números enteros (tantos como elementos usados tenga el vector).
   e) Número de filas usadas de la matriz `Matriz`.
   f) Número de columnas usadas de la matriz `Matriz`.
   g) Los elementos de la matriz (primero se da la fila 0, luego la fila 1, y así sucesivamente): una secuencia de números enteros.

   La salida del programa será la fila (número entero) de `Matriz` más cercana al vector `referencia` según el vector `filas_a_comparar`.

   Ejemplo de fichero de validación:

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

NOTA: En decsai.ugr.es se encuentra una **parte** de la definición de las clases `MiVectorInt` y `MiMatrizRectangularInt` a partir de las cuales se puede resolver el problema.