Indalecia
www.wuolah.com/student/Indalecia

57054

Examen FP - Extraordinario 2018.pdf

FEBRERO 2018 (EXTRAORDINARIA) + SOLUCIÓN

1° Fundamentos de Programación

Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

UNIVERSIDAD DE GRANADA

Departamento de Ciencias de la Computación e Inteligencia Artificial

Fundamentos de Programación (2017/18)
1º GII / GII-M / GII-ADE
Conv. Ordinaria - 19 de Enero de 2018

Normas para la realización del examen: **Duración: 2.25 horas**

- **Debe disponer de un documento oficial que acredite su identidad a disposición del profesor.**
- **El único material permitido durante la realización del examen es un bolígrafo o lápiz azul o negro.**
- **No olvide escribir su nombre completo y grupo en todos y cada uno de los folios que entregue.**
- **Dispone de 30 minutos para decidir si no se presenta al examen (y no le cuenta convocatoria)**

◁ Ejercicio 1 ▷ El problema del viajante de comercio [3.5 puntos]

Un diligente agente de comercio tiene que visitar periódicamente a sus n clientes. Cada uno de ellos vive en una ciudad distinta. Su jefe le ordena que prepare el plan de viaje del mes siguiente según las siguientes reglas:

- Tiene que visitar a todos los clientes y una sola vez a cada uno.
- El recorrido se hará de forma que el coste total sea el menor posible.
- No importa por qué ciudad comience, pero debe finalizar en la misma ciudad en la que comenzó. Por tanto, el recorrido forma un ciclo. Y, obviamente, no puede volver a la ciudad inicial hasta haber visitado todas las demás.

Como datos de entrada se facilitan el número de clientes, n, y el coste de viaje que hay entre cada par de ciudades, expresada como una matriz, C, de tamaño $n \times n$, donde C_{ij} es el coste de viajar desde la ciudad i a la j. Obviamente, estos valores son todos positivos, $C_{ij} > 0$.

Este es un problema muy difícil de resolver para el que existen distintas soluciones aproximadas. Uno de esos algoritmos, llamado *heurística del vecino más cercano*, funciona así: se selecciona una ciudad de partida, a, y se busca la ciudad más económica para llegar, b. A continuación, se busca la ciudad *aún no visitada* más económica a b. Y así hasta completar la visita a todas las ciudades. El coste del recorrido es la suma de todos los costes intermedios.

Implemente el algoritmo descrito anteriormente. Suponga que tanto el valor de n como la matriz C están declarados y tiene valores correctos. El programa pedirá primero la ciudad inicial. Luego calculará el recorrido y posteriormente su coste. Finalmente mostrará ambos resultados por la salida estándar.

Por ejemplo, supongamos un problema con $n = 4$ cuya matriz de costes es:

$$C = \begin{bmatrix} - & 10 & 15 & 9 \\ 11 & - & 13 & 8 \\ 17 & 21 & - & 15 \\ 26 & 7 & 14 & - \end{bmatrix}$$

Por simplicidad, asumimos que los nombres de las ciudades son `0, 1, ..., n`. Si partimos de la ciudad 0, entonces la solución encontrada es: $0, 3, 1, 2$ con un coste de $9 + 7 + 13 + 17 = 46$. Si el viaje comienza por la ciudad 2, el recorrido final sería $2, 3, 1, 0$ con un coste de $15 + 7 + 11 + 15 = 48$.

◁ Ejercicio 2 ▷ Buenas Palabras [3 puntos]

El profesor $Newton$ estaba leyendo un libro sobre el clima y los océanos cuando se encontró la palabra ‘‘`glaciological`’’. Pensó que era una palabra curiosa puesto que tiene la siguiente propiedad: si calculamos la diferencia entre las ocurrencias de las letras más y menos frecuentes (en este caso la 'l' y la 'g') el resultado es 1.

$Newton$ decidió llamar a palabras de este tipo como "palabras buenas con grado 1". Luego generalizó el concepto a palabras buenas de grado k: sea x el número de ocurrencias de la letra más frecuente y sea y el de la menos frecuente, entonces $x - y = k$.

Ejemplos:
palabra: `visibilidad`, valor de $k = 3$
palabra: `internet`, valor de $k = 1$
palabra: `marzo`, valor de $k = 0$

Suponga que una palabra se representa utilizando la clase `SecuenciaMinusculas`, la cual es similar a `SecuenciaCaracteres` pero solo permite almacenar letras minúsculas del alfabeto inglés (desde la 'a' a la 'z'). La especificación aparece más abajo.

A partir de dicha especificación se pide:

- implemente un método que indique cuál es el grado de la palabra almacenada. La solución debe evitar la repetición de cálculos.
- implemente un método que permita saber si el grado de una palabra es mayor que el grado de otra.

SecuenciaMinusculas
privados:

`static const int TAMANIO = 100`
`char vector_privado[TAMANIO]`
`int total_utilizados`

Métodos públicos que NO hay que implementar y se pueden usar:

`SecuenciaCaracteres()`
`int TotalUtilizados()`
`int Capacidad()`
`void Aniade(char c)`
`char Elemento(int i)`

Puede añadir los métodos adicionales que considere oportunos. Justifique si deberían ser públicos o privados.

◁ Ejercicio 3 ▷ Homogeneidad [3.5 puntos]

Se dispone de las clases `SecuenciaEnteros` y `TablaRectangularEnteros` descritas como se indica a continuación:

SecuenciaEnteros
Datos miembros privados:

`static const int TAMANIO = ...`
`int vector_privado[TAMANIO]`
`int total_utilizados`

Métodos públicos que NO hay que implementar y se pueden usar:

`SecuenciaEnteros()`
`int TotalUtilizados()`
`int Capacidad()`
`void Aniade(int nuevo)`
`int Elemento(int indice)`
`void EliminaTodos()`

TablaRectangularEnteros
Datos miembros privados:

`static const int NUM_FILS = ...`
`static const int NUM_COLS = ...`
`int matriz_privada[NUM_FILS][NUM_COLS]`
`int filas_utilizadas`
`int cols_utilizadas`

Métodos públicos que NO hay que implementar y se pueden usar:

`TablaRectangularEnteros(int num_columnas)`
`TablaRectangularEnteros(SecuenciaEnteros primera_fila)`
`int FilasUtilizadas()`
`int ColumnasUtilizadas()`
`int CapacidadFilas()`
`int CapacidadColumnas()`
`int Elemento(int indice_fila, int indice_columna)`
`void Aniade(SecuenciaEnteros fila_nueva)`

Suponemos que los valores que puede contener un objeto de la clase `TablaRectangularEnteros` están en el conjunto $[0, 255]$.

Añadir a la clase `TablaRectangularEnteros` el método `CalculaHomogeneidad` que permite detectar zonas homogéneas.

Para esta tarea se dispone también del método público de la clase `TablaRectangularEnteros` (no es necesario implementarlo):

```
bool RegionHomogenea (int fila, int col, int tam_vecindad, double umbral);
```

> Devuelve `true` si la casilla (`fila`, `col`) es el centro de una región homogénea de "ancho" `tam_vecindad`. El criterio de homogeneidad (mayor o menor exigencia) puede ajustarse según el valor de `umbral`.
>
> Si `tam_vecindad` es 1 la vecindad de la casilla (`fila`, `col`) está formada por ella misma y las 8 casillas que la rodean. Si `tam_vecindad` es 2, la vecindad está formada por la vecindad de tamaño 1 y las 16 casillas que la rodean. Si `tam_vecindad` es 3, la vecindad está formada por su vecindad de tamaño 2 y las 24 casillas que la rodean, ...

El método `CalculaHomogeneidad` devolverá una `TablaRectangularEnteros` en la que las casillas que no tengan la vecindad completa tendrán el valor `NO_TRATADO`. En otro caso, se tendrá en cuenta el criterio de homogeneidad implementado en el método `RegionHomogenea()`: aquellas casillas que sean centro de una zona homogénea de "ancho" `tam_vecindad` tendrán el valor `HOMOGENEA` y las que no lo sean tendrán el valor `NO_HOMOGENEA`.

Nota: **No está permitido acceder a los datos privados. Usad únicamente los métodos públicos de la clase.**