PROGRAMACIÓN Y ESTRUCTURAS DE DATOS AVANZADAS

PED1

Codigo asignatura: 71902019

1.- ENUNCIADO DE LA PRÁCTICA: Minimización del tiempo en el sistema

Un servidor (por ejemplo, un procesador, un surtidor de gasolina o un cajero automático) tiene que atender n clientes que llegan todos juntos al sistema. El tiempo que requerirá dar servicio a cada cliente es conocido, siendo ti el tiempo de servicio requerido por el cliente i-ésimo, y siendo $1 \leq i \leq n$. El objetivo es minimizar el tiempo medio de estancia de los clientes en el sistema. Como el valor de n es conocido, minimizar el tiempo medio de estancia equivale a minimizar el tiempo total que están en el sistema todos los clientes; es decir, se pretende minimizar la expresión:

$$T = \Sigma_{i=1}^{n} \text{ tiempo en el sistema del cliente i}$$

Para ello, se pide implementar un algoritmo voraz que construya la secuencia ordenada óptima de servicio a los distintos clientes.

- **Descripción del esquema algorítmico utilizado y cómo se aplica al problema.**

Tal como indica el enunciado el algoritmo utilizado será voraz. Para explicar su funcionamiento primero explicaremos lo que es:

Una aproximación voraz consiste en que cada elemento a considerar se evalúa una única vez, siendo descartado o seleccionado, de tal forma que si es seleccionado forma parte de la solución, y si es descartado, no forma parte de la solución ni volverá a ser considerado para la misma.

Conociendo lo que hace el algoritmo lo aplicamos a la practica de tal forma que se elige el valor mínimo y lo añade a la solución. Seguiremos el ejemplo de una gasolinera para explicar el funcionamiento. Cuando llegan varios vehículos a la vez y conocemos el tiempo que requerirá dar servicio a cada uno de ellos, elegimos primero el tenga menor tiempo de servicio, le atendemos y volvemos a repetir el mismo proceso, buscando el que tenga menor tiempo. Con esto conseguiremos minimizar tanto el coste medio de espera como el coste total de espera.

La aplicación del algoritmo se ha llevado a cabo mediante la ordenación de los clientes según su tiempo de servicio, de menor a mayor y después atendiendo los en ese orden. Así se insertan todos los tiempos de espera en un vector y se ordena garantizando asi el cumplimiento del algoritmo.

- **Demostración de optamibilidad.**

Para la demostración de optimibilidad procederemos de la siguiente forma:
Suponiendo que tenemos una cantidad n de clientes, tendremos un conjunto de clientes C=$(i_1, i_2, ..., i_n)$.
Calculamos el tiempo total de espera:

$$T(C) = t_{i1} + (t_{i1} + t_{i2}) + ((t_{i1} + t_{i2}) + t_{i3}) + ...$$
$$= nt_{i1} + (n-1)t_{i2} + (n-2)t_{i3} + ... + 2t_{in-1} + t_{in}$$
$$= \sum_{k=1}^{n} = (n-k+1)t_{ik}$$

Buscamos ahora un cliente cuyo tiempo de servicio es mayor que el de otro atendido después de el, de tal modo que $a < b$ y $t_{ia} > t_{ib}$. De esta manera, si invertimos sus posiciones y los ponemos en el orden esperado obtendríamos una nueva secuencia C'. Entonces el nuevo orden es el siguiente:

$$T(C') = (n-a+1)t_{ib} + (n-b+1)t_{ia} + \sum_{\substack{k=1 \\ k \neq a,b}}^{n} (n-k+1)t_{ik}$$

$$T(C) - T(C') = (n-a+1)(t_{ia} - t_{ib}) + (n-b+1)(t_{ib} - t_{ia})$$
$$= (b-a)(t_{ia} - t_{ib}) > 0$$

Por lo que llegamos a la conclusión de que cualquier secuencia en la que un cliente que necesita mas tiempos es atendido antes que otro que necesita menos se puede mejorar. Las únicas secuencias que no pueden ser mejoradas, y por tanto son óptimas, son aquellas en las que los clientes son atendidos según su tiempo de servicio creciente.

- **Analisis el coste computacional del algoritmo.**

Para analizar el coste computacional de la practica veremos los distintos costes de los bucles utilizados (PED1_1).
-Para la inserción de clientes tenemos un coste lineal $O(n)$

```
for(String numero:cadena){
    mont.add(Integer.parseInt(numero));
    aux.add(Integer.parseInt(numero));
}
```

-Para la ordenación del vector se ha utilizado quicksort existente en las bibliotecas de java, cuyo coste es $O(nlog(n))$

-Para el cálculo del tiempo total de espera se utiliza el siguiente bucle con coste lineal $O(n)$:

```
//Se calcula el tiempo total de espera de los clientes con coste lineal: n
for(int n=0;n<num;n++){total=total+aux.get(n)*(num-n);}
```

-Se insertan as posiciones de salida utilizando un bucle con coste cuadrático

```
        //Se rellena la cadena de salida utilizando el orden de llegada con coste: n^2
        for(int j=0; j<num; j++){
            for(int i=1; i<num+1; i++){
                if(mont.get(i-1)==aux.get(j)){
                    cadena+=(i+" ");
                }
            }
        }
```

$O(n^2)$

Comprobamos por tanto que a pesar de que el coste de entrada y ordenación de los clientes no es elevado, el coste de obtener la posición en la que entran si lo es.

- **Alternativas al esquema utilizado si las hay, y comparacion del coste entre las distintas soluciones.**

Cabe destacar que debido al ultimo bucle presentado el coste computacional aumenta considerablemente. Como solución se propone la realización de la practica siguiendo la programación orientada a objetos, y por tanto tratando los clientes entrantes como objetos. Para ello crearemos una clase Cliente. Cada cliente tendrá su posición y el valor del tiempo de espera. Las modificaciones realizadas al código se incluyen en (PED1_2).

Así para el coste computacional tenemos:
- Para el coste de inserción seguimos teniendo un coste lineal $O(n)$:

```
//Se rellena el ArrayList con los clientes existentes
for(int i=0; i<num; i++){
   Cliente cl = new Cliente(i+1,Integer.parseInt(cadena[i]));
   listaClientes.add(cl);
}
```

- Para la ordenación del vector seguimos teniendo un coste $O(nlog(n))$

- La diferencia la encontramos en la creación de la cadena que vamos a imprimir, habiendo mejorado el coste a $O(n)$, y a la vez calculamos el tiempo total de espera y realizamos la traza:

```
//Se calcula el tiempo total de espera de los clientes con coste lineal: n
//Se rellena la cadena de salida utilizando el orden de llegada con coste: nlog(n)
int suma=0;
int total=0;
for(int i=0;i<num;i++){
        cadena+=(listaClientes.get(i).getPosicion()+" ");
        suma=suma+listaClientes.get(i).getValor();
        total=total+suma;
        miTraza=miTraza+"Paso: "+(i+1)+"\n"+"Datos: "+cadena+"\n"+"Tiempo de espera: "
        +total+"\n--------------------\n";
}
```

Vemos por tanto la diferencia de utilizar "fuerza bruta" y planificación orientada a objetos. Se ha obviado la opción de traza en PED1_1 debido a que aumenta todavía mas el coste y tiene una gran complejidad de código. Asi de un $O(n^2)$ hemos pasado a un $O(nlog(n))$ mejorando la eficiencia del programa con poco esfuerzo e incluso facilitando la comprensión del código.

- **Descripción de los datos de prueba utilizados y los resultados obtenidos con ellos.**

Se han realizado varias pruebas sobre el código, introduciendo diferentes parámetros. Se han comprobado las distintas ejecuciones del programa:

Código	Función
"","",""	No se realiza ninguna funcion y se imprime mensaje de error
"-h","",""	Imprime el menu ayuda
"f_entrada","",""	Imprime por pantalla el tiempo total y el orden
"-t","f_entrada",""	Imprime por pantalla el tiempo total, el orden y la traza
"f_entrada","f_salida",""	Guarda el tiempo total y el orden en el fichero de salida
"-t","f_entrada","f_salida"	Guarda el tiempo total,el orden y la traza en el fichero de salida

Table 1: Funcionamiento del programa.

A continuación se presentan varios ejemplos:
Menu ayuda:

Función:

("-h","doc_e","")

```
SINTAXIS:
servicio [-t] [-h] [fichero_entrada] [fichero_salida]
-t                      Traza la selección de clientes
-h                      Muestra esta ayuda
fichero_entrada         Nombre del fiechero de entrada
fichero_salida          Nombre del fichero de salida
```

Traza con 6 elementos:

Entrada:

```
6
1 9 7 5 3 3
```

Función:

"-t","doc_e",""

Salida:

```
71
1 5 6 4 3 2

Traza:
--------------------
Paso: 1
Datos: 1
Tiempo de espera: 1
--------------------
Paso: 2
Datos: 1 5
Tiempo de espera: 5
--------------------
Paso: 3
Datos: 1 5 6
Tiempo de espera: 12
--------------------
Paso: 4
Datos: 1 5 6 4
Tiempo de espera: 24
--------------------
Paso: 5
Datos: 1 5 6 4 3
Tiempo de espera: 43
--------------------
Paso: 6
Datos: 1 5 6 4 3 2
Tiempo de espera: 71
--------------------
```

Traza con 15 elementos:

Entrada	Salida
15 11 9 17 5 13 3 8 2 6 4 7 3 1 5 8 Función: "-t", "doc_e", ""	554 13 8 6 12 10 4 14 9 11 7 15 2 1 5 3 Traza: -------------------- Paso: 1 Datos: 13 Tiempo de espera: 1 -------------------- Paso: 2 Datos: 13 8 Tiempo de espera: 4 -------------------- Paso: 3 Datos: 13 8 6 Tiempo de espera: 10 -------------------- Paso: 4 Datos: 13 8 6 12 Tiempo de espera: 19 -------------------- Paso: 5 Datos: 13 8 6 12 10 Tiempo de espera: 32 -------------------- Paso: 6 Datos: 13 8 6 12 10 4 Tiempo de espera: 50 -------------------- Paso: 7 Datos: 13 8 6 12 10 4 14 Tiempo de espera: 73 -------------------- Paso: 8 Datos: 13 8 6 12 10 4 14 9 Tiempo de espera: 102 -------------------- Paso: 9 Datos: 13 8 6 12 10 4 14 9 11 Tiempo de espera: 138 -------------------- Paso: 10 Datos: 13 8 6 12 10 4 14 9 11 7 Tiempo de espera: 182 -------------------- Paso: 11 Datos: 13 8 6 12 10 4 14 9 11 7 15 Tiempo de espera: 234 -------------------- Paso: 12 Datos: 13 8 6 12 10 4 14 9 11 7 15 2 Tiempo de espera: 295 -------------------- Paso: 13 Datos: 13 8 6 12 10 4 14 9 11 7 15 2 1 Tiempo de espera: 367 -------------------- Paso: 14 Datos: 13 8 6 12 10 4 14 9 11 7 15 2 1 5 Tiempo de espera: 452 -------------------- Paso: 15 Datos: 13 8 6 12 10 4 14 9 11 7 15 2 1 5 3 Tiempo de espera: 554 --------------------

- **Listado COMPLETO del codigo fuente:**

Cliente.java | *PED1_1.java | servicio.java

```java
public class Cliente implements Comparable<Cliente> {
    int posicion,valor;
    public Cliente(int pos, int val){
        this.posicion=pos;
        this.valor=val;
    }
    public int getPosicion(){
        return posicion;
    }
    public int getValor(){
        return valor;
    }
    @Override
    public int compareTo(Cliente cl) {
        if (valor < cl.valor) {
            return -1;
        }
        if (valor > cl.valor) {
            return 1;
        }
        return 0;
    }

}

```

Cliente.java | *PED1_1.java | servicio.java

```java
/*
 *
 *
 *
 *
 */

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.util.ArrayList;
import java.util.Collections;

public class servicio{
    int num = 0; // Variable utilizada para el numero de clientes
    int total=0; // Variable utilizada para almacenar el tiempo total
    String cadena=""; // Variable utilizada para guardar el contenido del vector e imprimirlo
    String miTraza=""; // Variable utilizada para guardar la traza

    public servicio(String select, String f_entrada, String f_salida){
        boolean traza=(select.trim().equals("-t") || select.trim().equals("-T"));
        boolean help=(select.trim().equals("-h") || select.trim().equals("-H"));
        if(!traza && !help){
            f_salida=f_entrada;
            f_entrada=select;
            }
        if(f_entrada.isEmpty())System.out.println("Seleccione archivo de entrada");
        //Solo se permitiran archivos con formato *.txt, en caso de que no se ponga expresamente
        if(!f_entrada.endsWith(".txt"))f_entrada=f_entrada+".txt";
        if(!f_salida.isEmpty() && !f_salida.endsWith(".txt"))f_salida=f_salida+".txt";

        //(-h)OPCION HELP o AYUDA
        if (help){
            System.out.println("SINTAXIS:");
            System.out.println("servicio [-t] [-h] [fichero_entrada] [fichero_salida]");
            System.out.println("-t\t\t\t Traza la selección de clientes");
            System.out.println("-h\t\t\t Muestra esta ayuda");
            System.out.println("fichero_entrada\t\t Nombre del fiechero de entrada");
            System.out.println("fichero_salida\t\t Nombre del fichero de salida");
        }

            //Caso con archivo de entrada
        else if(!f_entrada.isEmpty()){
                //Se utiliza 1 ArrayListde Clientes
                ArrayList<Cliente> listaClientes = new ArrayList<Cliente>();
                try {
                    //Lectura del fichero de entrada
                     FileReader entrada = new FileReader(f_entrada);
                     BufferedReader lector = new BufferedReader(entrada);
                     String linea = lector.readLine();
                     num=Integer.parseInt(linea.trim());
                     listaClientes = new ArrayList<Cliente>(num);
                     linea = lector.readLine();
                     String[] cadena = linea.split(" ");
                     //Se rellena el ArrayList con los clientes existentes
                     for(int i=0; i<num; i++){
                        Cliente cl = new Cliente(i+1,Integer.parseInt(cadena[i]));
                        listaClientes.add(cl);
```

Cliente.java | *PED1_1.java | servicio.java

```
58                          Cliente cl = new Cliente(i+1,Integer.parseInt(cadena[i]));
59                          listaClientes.add(cl);
60                      }
61                      lector.close();
62                      //Se ordena el vector auxiliar utilizando algoritmo de ordenacion quicksort
63                      Collections.sort(listaClientes);
64                  }
65                 catch (Exception e){System.out.println("Ha ocurrido un error no previsto");}
66                  //Se calcula el tiempo total de espera de los clientes con coste lineal: n
67                  //Se rellena la cadena de salida utilizando el orden de llegada con coste: nlog(
68                  int suma=0;
69                  int total=0;
70                  for(int i=0;i<num;i++){
71                          cadena+=(listaClientes.get(i).getPosicion()+" ");
72                          suma=suma+listaClientes.get(i).getValor();
73                          total=total+suma;
74                          miTraza=miTraza+"\nPaso: "+(i+1)+"\n"+"Datos: "+cadena+"\n"+"Tiempo de e
75                          +total+"\n--------------------";
76                  }
77                  //Casos de salida de fichero
78                  try {
79                      //Sin archivo de salida, imprimimos por pantalla
80                      if(f_salida.isEmpty()){
81                          System.out.println(total);
82                          System.out.println(cadena);
83                        //(-t)OPCION TRAZA
84                          if (traza){
85                              System.out.println("\nTraza:\n--------------------\n"+miTraza);
86                                      }
87                      }
88                      //Guardamos el contenido en el archivo de salida
89                      else if(f_salida!=null || !f_salida.isEmpty()) {
90                          File nuevo=new File(f_salida);
91                          String ruta=nuevo.getAbsolutePath();
92                          File archivo=new File(ruta);
93                          if(archivo.exists()){
94                              System.out.println("Error, no se permite sobreescribrir.");
95                          }
96                          else if(!archivo.exists()){
97                              FileWriter escribir=new FileWriter(archivo,true);
98                              escribir.write(total+"");
99                              escribir.write("\r\n");
100                             escribir.write(cadena);
101                             if(traza){escribir.write(miTraza);}
102                             escribir.close();
103                         }
104                     }
105                 }
106                 catch(Exception e){System.out.println("Ha ocurrido un error no previsto");}
107             }
108             else{System.out.println("Comando incorrecto");}
109
110         }
111
112     public static void main(String[] args) {
113         //Main m = new Main(args[0],args[1],args[2]);
114         new servicio("-t","doc_e","");
115     }
116 }
```

Cliente.java | *PED1_1.java | servicio.java

```
 2  *
 7
 8  import java.io.BufferedReader;
14
15  public class PED1_1{
16      int num = 0; // Variable utilizada para el numero de clientes
17      int total=0; // Variable utilizada para almacenar el tiempo total
18      String cadena=""; // Variable utilizada para guardar el contenido del vector e imprimirlo
19      String miTraza=""; // Variable utilizada para la traza
20      private int sumatorio;
21
22      public PED1_1(String select, String f_entrada, String f_salida){
23          if(!select.trim().equals("-t") && !select.trim().equals("-T") && !select.trim().equals("
24              f_salida=f_entrada;
25              f_entrada=select;
26              }
27          else if(f_entrada.isEmpty())System.out.println("Seleccione archivo de entrada");
28          //Solo se permitiran archivos con formato *.txt, en caso de que no se ponga expresamente
29          else if(!f_entrada.endsWith(".txt"))f_entrada=f_entrada+".txt";
30          else if(!f_salida.isEmpty() && !f_salida.endsWith(".txt"))f_salida=f_salida+".txt";
31
32          //(-h)OPCION HELP o AYUDA
33          if (select.trim().equals("-h") || select.trim().equals("-H")){
34              System.out.println("SINTAXIS:");
35              System.out.println("servicio [-t] [-h] [fichero_entrada] [fichero_salida]");
36              System.out.println("-t\t\t\t Traza la selección de clientes");
37              System.out.println("-h\t\t\t Muestra esta ayuda");
38              System.out.println("fichero_entrada\t\t Nombre del fiechero de entrada");
39              System.out.println("fichero_salida\t\t Nombre del fichero de salida");
40          }
41
42              //Caso con archivo de entrada
43              if(!f_entrada.isEmpty()){
44                  //Se utilizan 2 ArrayList, uno para el vector original y otro para el auxiliar q
45                  ArrayList<Integer> mont = new ArrayList<Integer>();
46                  ArrayList<Integer> aux = new ArrayList<Integer>();
47                  try {
48                      //Lectura del fichero de entrada
49                       FileReader entrada = new FileReader(f_entrada);
50                       BufferedReader lector = new BufferedReader(entrada);
51                       String linea = lector.readLine();
52                       num=Integer.parseInt(linea.trim());
53                       mont = new ArrayList<Integer>(num);
54                       linea = lector.readLine();
55                       String[] cadena = linea.split(" ");
56                       //Se rellenan los ArrayList con los valores de entrada
57                       for(String numero:cadena){
58                           mont.add(Integer.parseInt(numero));
59                           aux.add(Integer.parseInt(numero));
60                       }
61                       lector.close();
62                       //Se ordena el vector auxiliar utilizando algoritmo de ordenacion quicksort
63                       Collections.sort(aux);
64                  }
65                  catch (Exception e){System.out.println("Ha ocurrido un error no previsto");}
66                  //Se calcula el tiempo total de espera de los clientes con coste lineal: n
67                  //for(int n=0;n<num;n++){total=total+aux.get(n)*(num-n);}
68                  //Se rellena la cadena de salida utilizando el orden de llegada con coste: n^2
69                  ArrayList<Integer> suma = new ArrayList<Integer>();
```

Cliente.java | *PED1_1.java | servicio.java

```
67                  //for(int n=0;n<num;n++){total=total+aux.get(n)*(num-n);}
68                  //Se rellena la cadena de salida utilizando el orden de llegada con coste: n^2
69                  ArrayList<Integer> suma = new ArrayList<Integer>();
70                  for(int j=0; j<num; j++){
71                      total=total+aux.get(j)*(num-j);
72                      if(j==0){suma.add(aux.get(j));}
73                      if (j>0){
74                          sumatorio = 0;
75                          for(int k=0;k<suma.size();k++){
76                              sumatorio+=suma.get(k);}
77                          suma.add(suma.get(j-1)+sumatorio+aux.get(j));
78                      }
79                      for(int i=1; i<num+1; i++){
80                          if(mont.get(i-1)==aux.get(j)){
81                              cadena+=(i+" ");
82                              System.out.println(suma);
83                              miTraza=miTraza+"Paso: "+(j+1)+"\n"+"Datos: "+cadena+"\n"+"Tiempo: "
84                          }
85                      }
86                  }
87                  //Casos de salida de fichero
88                  try {
89                      //Sin archivo de salida, imprimimos por pantalla
90                      if(f_salida.isEmpty()){
91                          System.out.println(total);
92                          System.out.println(cadena);
93                      }
94                      //Guardamos el contenido en el archivo de salida
95                      else if(f_salida!=null || !f_salida.isEmpty()) {
96                          File nuevo=new File(f_salida);
97                          String ruta=nuevo.getAbsolutePath();
98                          File archivo=new File(ruta);
99                          if(archivo.exists()){
100                             System.out.println("Error, no se permite sobreescribrir.");
101                         }
102                         else if(!archivo.exists()){
103                             FileWriter escribir=new FileWriter(archivo,true);
104                             escribir.write(total+"");
105                             escribir.write("\r\n");
106                             escribir.write(cadena);
107                             escribir.close();
108                         }
109                     }
110                 }
111                 catch(Exception e){System.out.println("Ha ocurrido un error no previsto");}
112             }
113             else{System.out.println("Comando incorrecto");}
114           //(-t)OPCION TRAZA
115         if (select.trim().equals("-t") || select.trim().equals("-T")){
116             System.out.println("\nTraza:\n--------------------\n"+miTraza);
117                     }
118     }
119
120
121     public static void main(String[] args) {
122         //Main m = new Main(args[0],args[1],args[2]);
123         new PED1_1("-t", "doc_e","");
124     }
125 }
```