PROGRAMACIÓN Y ESTRUCTURAS DE DATOS AVANZADAS

PRÁCTICA 1

Luis Daniel Guardado Garcia
77175422A
Lguardado7@alumno.uned.es

CUESTIONES TEÓRICAS DE LA PRÁCTICA

1) Analiza el coste computacional del algoritmo.

La resolución del problema usa un esquema de divide y vencerás.

Al tratarse de algoritmos recursivos, es necesario plantear la ecuación de recurrencia. Al estar dividiendo el problema, normalmente se podrá aplicar la formula para reducciones por división:

$$T(n) = \begin{cases} cn^k, & si\ 1 \leq n < b \\ aT\left(\frac{n}{b}\right) + cn^k, & si\ n \geq b \end{cases}$$

$$f(n) = \begin{cases} \theta(n^k), & si\ a < b^k \\ \theta(n^k \log n), & si\ a = b^k \\ \theta(n^{log_b a}), & si\ a > b^k \end{cases}$$

Por lo tanto, hay que plantear la ecuación de recurrencia para $T(n)$ e identificar en ella las constantes a, b y k.

Para este problema, dividimos la entrada de edificios E en dos conjuntos. Calculamos la línea del horizonte para cada uno de los conjuntos y luego combinamos ambas líneas utilizando siempre el punto con mayor altura. Por lo descrito, la estructura utilizada es parecida a la que se utiliza en el algoritmo mergesort. Para fusionar ambas líneas utilizaremos variables auxiliares para tener registradas la altura previa de cada conjunto, la altura actual y las nuevas coordenadas.
Nuestra línea de horizontes, tal y como está detallado en el enunciado de la práctica, son una lista de pares que representan la posición y la altura donde cambia la línea de horizonte. Según lo detallado en el libro, dividimos el problema en subproblemas de tamaño n/2, suponiendo que n = 2k
La fusión de dos líneas de horizonte de tamaño n/2 requiere O(n), es decir, un tiempo lineal. A su vez, la descomposición de E en dos conjuntos requiere un tiempo lineal O(n).

Por lo tanto, T(n) = 2T(n/2) + O(n).

Es decir, T(n) ∈ Θ (n log n). Podemos observar que esta es la complejidad que tiene un algoritmo mergesort. Puesto que estamos utilizando una estructura similar tiene sentido que la complejidad sea parecida o igual.

2) Describe alternativas al esquema utilizado, si las hay, y compara su coste con el de la solución realizada.

La única alternativa que se me ocurre es utilizando el algoritmo programación dinámica ya que es el único que puede descomponer el problema en subproblemas de tamaño menor del mismo tipo para llegar a la solución. Su coste es el siguiente:

$$M(i,j,k) = \begin{cases} 0, & si\ k = 0\ y\ i = j \\ A[i,j], & si\ k = 0\ y\ i \neq j \\ \min\ (M(i,j,k-1), & si\ k > 0 \\ M(i,k,k-1) + M(k,j,j-1)) & \end{cases}$$

El coste temporal de este algoritmo, con los tres bucles anidados, está en $\theta(N^3)$, mientras el coste espacial está en $\theta(N^2)$.

Diferencia entre ambos:

1. Divide y vencerás a menudo utiliza cálculos recursivos, cálculos de arriba hacia abajo y la programación dinámica calcula directamente de abajo hacia arriba

2. Los pequeños problemas de divide y vencerás pueden calcularse repetidamente en el proceso de recursión. Los pequeños problemas en la programación dinámica se almacenan después del cálculo y se llamarán directamente cuando se encuentren la próxima vez.

3. La solución al pequeño problema del método de divide y vencerás se usa solo una vez, y la solución del pequeño problema de la programación dinámica se almacena para llamadas repetidas al resolver el gran problema

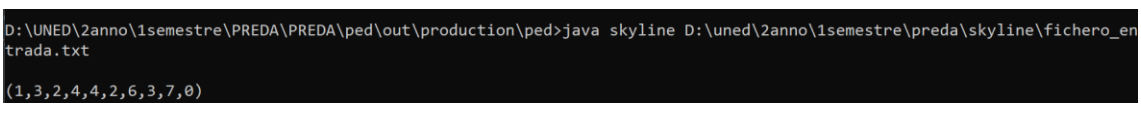
Condiciones generales de construcción de la programación dinámica:

1. Existe una subestructura óptima: la solución óptima en el siguiente nivel puede usarse como base para la solución óptima en el nivel anterior

2. Hay subproblemas duplicados: muchos subproblemas se llamarán varias veces (eficiencia)

UN EJEMPLO DE EJECUCIÓN PARA DISTINTOS TAMAÑOS DEL PROBLEMA

Ejemplo nº1: Sólo como argumento el fichero de entrada.



Ejemplo nº2: Como argumentos el fichero de entrada y salida (sin crear).

```
D:\UNED\2anno\1semestre\PREDA\PREDA\ped\out\production\ped>java skyline D:\uned\2anno\1semestre\preda\skyline\fichero_en
trada.txt D:\uned\2anno\1semestre\preda\floyd\salida.txt
 Archivo de salida creada. Si no se ha creado es porque has introducido mal la ubicación o el formato de salida (txt)
```

Ejemplo nº3: Como argumentos el fichero de entrada y salida (creado).

```
D:\UNED\2anno\1semestre\PREDA\PREDA\ped\out\production\ped>java skyline D:\uned\2anno\1semestre\preda\skyline\fichero_en
trada.txt D:\uned\2anno\1semestre\preda\floyd\salidaa.txt
 El archivo de salida ya existe.
```

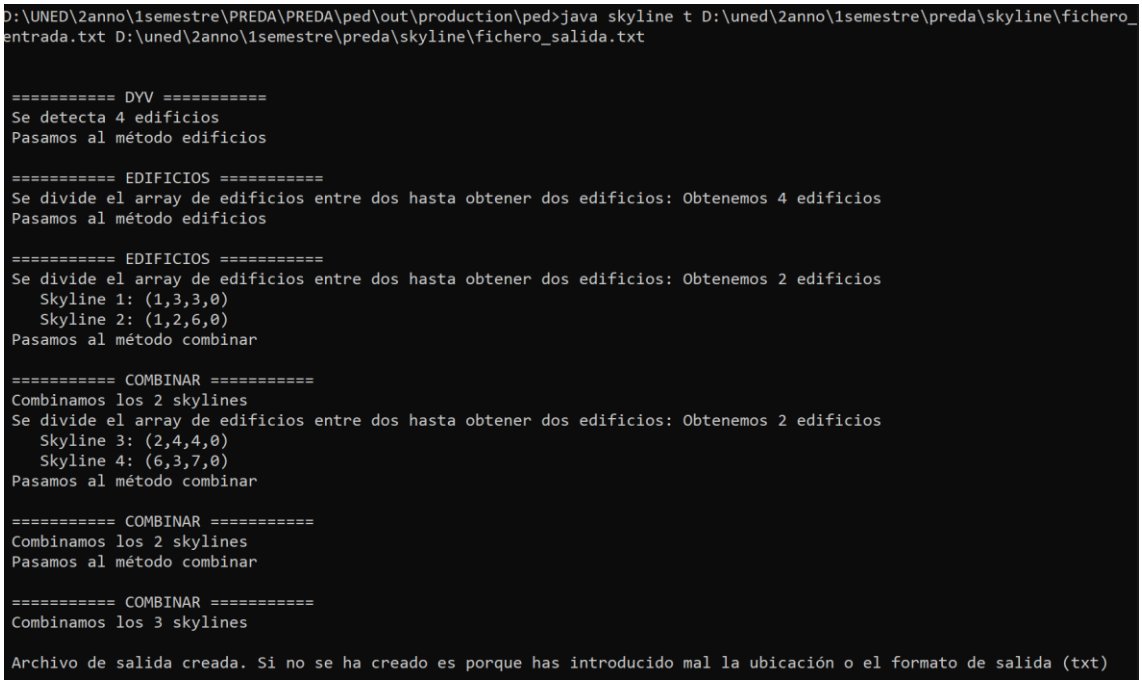
Ejemplo nº4: Como argumentos, la tecla t y fichero de entrada.

```
D:\UNED\2anno\1semestre\PREDA\PREDA\ped\out\production\ped>java skyline t D:\uned\2anno\1semestre\preda\skyline\fichero_
entrada.txt

(1,3,2,4,4,2,6,3,7,0)

 ========== DYV ==========
 Se detecta 4 edificios
 Pasamos al método edificios

 ========== EDIFICIOS ==========
 Se divide el array de edificios entre dos hasta obtener dos edificios: Obtenemos 4 edificios
 Pasamos al método edificios

 ========== EDIFICIOS ==========
 Se divide el array de edificios entre dos hasta obtener dos edificios: Obtenemos 2 edificios
    Skyline 1: (1,3,3,0)
    Skyline 2: (1,2,6,0)
 Pasamos al método combinar

 ========== COMBINAR ==========
 Combinamos los 2 skylines
 Se divide el array de edificios entre dos hasta obtener dos edificios: Obtenemos 2 edificios
    Skyline 3: (2,4,4,0)
    Skyline 4: (6,3,7,0)
 Pasamos al método combinar

 ========== COMBINAR ==========
 Combinamos los 2 skylines
 Pasamos al método combinar

 ========== COMBINAR ==========
 Combinamos los 3 skylines
```

Ejemplo nº5: Sólo como argumento, la letra h.

```
D:\UNED\2anno\1semestre\PREDA\PREDA\ped\out\production\ped>java skyline h

 SINTAXIS: skyline [-t] [-h]  [fichero entrada]  [fichero salida]
      -t          Traza cada llamada recursiva y sus parámetros
      -h          Muestra esta ayuda
      [fichero entrada]   Conjunto de edificios de la ciudad
      [ficharo salida]   Secuencia que representan el skyline de la ciudad
```

Ejemplo nº6: Como argumentos, la letra t, fichero de entrada y fichero de salida.



CÓDIGO COMPLETO

```
import java.io.*;
import java.util.*;

public class skyline {

    public static String resumen;
    public static Boolean activar = false;
    public static int num = 1;

    // Divide y Vencerás
    public List<int[]> dyv(int[][] problema) {
// O(1)
        resumen = "\n\n ========== DYV ==========\n";
// 1
        resumen += " Se detecta " + problema.length + " edificios\n";
// 1

        if (problema.length == 0)
// 1
            return new ArrayList<>();
        activar = true;

        return edificios(problema, 0, problema.length - 1);
// 1
    }
```

```
    // Edificios
    private ArrayList<int[]> edificios(int[][] C, int i, int j) {
// O(1)
        int m, n;
// 1
        m = (i + j - 1) / 2;
// 1
        n = j - i + 1;
// 1

        if (n == 1) {
// 1
            ArrayList<int[]> s = new ArrayList<>();
            s.add(new int[]{C[i][0], C[i][2]});
            s.add(new int[]{C[i][1], 0});
            activar = false;
            resumen += "    Skyline " + (i + 1) + ": " + "(" + C[i][0]
+ "," + C[i][2] + "," + C[i][1] + "," + 0 + ")\n";
            return s;
        } else {
// 1
            if (activar && num == 1) {
                resumen += " Pasamos al método edificios\n";
                resumen += "\n ========== EDIFICIOS ==========\n";
            }

            resumen += " Se divide el array de edificios entre dos
hasta obtener dos edificios: Obtenemos " + n + " edificios\n";
            ArrayList<int[]> sa = edificios(C, i, m);
            ArrayList<int[]> sb = edificios(C, m + 1, j);
            activar = false;
            num++;
            return combinar(sa, sb);
        }
    }


    // Combinar
    private ArrayList<int[]> combinar(ArrayList<int[]> sa,
ArrayList<int[]> sb) {   // O(n)
        resumen += " Pasamos al método combinar\n";
// 1
        resumen += "\n ========== COMBINAR ==========\n";
// 1
        ArrayList<int[]> s = new ArrayList<>();
// 1
        int ha = 0, hb = 0, uh = 0;
// 1
        int[] a, b;
```

```
// 1
        int ia, ib, nx, nh;
// 1

        ia = 0;
// 1
        ib = 0;
// 1

        resumen += " Combinamos los " + sa.size() + " skylines\n";
// 1

        while (ia < sa.size() && ib < sb.size()) {
// n
            a = sa.get(ia);
            b = sb.get(ib);

            if (a[0] == b[0]) {
                nx = a[0];
                nh = Math.max(a[1], b[1]);
                ha = a[1];
                hb = b[1];
                ia = ia + 1;
                ib = ib + 1;
            } else {
                if (a[0] < b[0]) {
                    nx = a[0];
                    nh = Math.max(a[1], hb);
                    ha = a[1];
                    ia = ia + 1;
                } else {
                    nx = b[0];
                    nh = Math.max(b[1], ha);
                    hb = b[1];
                    ib = ib + 1;
                }
            }
            if (uh != nh) {
                s.add(new int[]{nx, nh});
                uh = nh;
            }
        }
        while (ia < sa.size()) {
// n
            s.add(sa.get(ia++));
        }
        while (ib < sb.size()) {
// n
            s.add(sb.get(ib++));
        }
```

```
        return s;
// 1
    }

    // Menú de ayuda
    public static void ayuda() {
// O(1)
        System.out.println();
        System.out.println(" SINTAXIS: skyline [-t] [-h]  [fichero
entrada]  [fichero salida]");
        System.out.println("      -t          Traza cada llamada
recursiva y sus parámetros");
        System.out.println("      -h          Muestra esta ayuda");
        System.out.println("      [fichero entrada]   Conjunto de
edificios de la ciudad");
        System.out.println("      [ficharo salida]   Secuencia que
representan el skyline de la ciudad");
        System.out.println();
    }

    // En esta función lo que se hace básicamente es obtener los
edificios del txt
    public static void programa(String t, String h, String entrada,
String salida) {// O(n^3)
        try {
// 1
            String prueba;  // Es la entrada formateada para que
nuestro programa pueda leerla.
            File fentrada = new File(entrada);
            File fsalida = new File(salida);

            if (fentrada.exists()) {
// 1
                BufferedReader bufferedReader = new BufferedReader(new
FileReader(fentrada));
                // El formato de entrada es {(x,x),(x,x)} así como nos
lo indica el libro.
                while ((prueba = bufferedReader.readLine()) != null) {
// n
                    prueba = prueba.replace("}", "");
                    prueba = prueba.replace('{', ' ');
                    prueba = prueba.replaceAll("\\(", "");
                    prueba = prueba.replaceAll("\\),", "&");
                    prueba = prueba.replaceAll("\\)", "");
                    prueba = prueba.trim();

                    String[] row = prueba.split("&");
                    int[][] matrix = new int[row.length][];
                    for (int i = 0; i < row.length; i++) {
// n^2
```

```
                        String currentline = row[i].trim();
                        int[] temp = new int[currentline.length() -
2];
                        int cont = 0;
                        for (int j = 0; j < currentline.length(); j =
j + 2) {      // n^3
                            temp[cont] =
Character.getNumericValue(currentline.charAt(j));
                            cont++;
                        }
                        matrix[i] = temp;
                    }

                    StringBuilder cadena;  // Contiene la salida del
skyline
                    skyline ejemplo = new skyline();
                    cadena = new StringBuilder("(");

                    for (int[] p : ejemplo.dyv(matrix)) {
// n^2
                        for (int j : p) {
// n^3
                            if (j != 0) {
                                cadena.append(j).append(",");
                            } else {
                                cadena.append(j);
                            }
                        }
                    }

                    // Esto es un miniprograma que consiste en
comprobar si es necesario crear una salida.txt o imprimirla por
pantalla.
                    if (salida.equals("0")) {
// n
                        System.out.println();
                        System.out.print(cadena + ")");
                    } else {
// n
                        if (fsalida.exists()) {
                            System.out.println();
                            System.out.println(cadena + ")");
                        } else {
                            BufferedWriter bufferedWriter = new
BufferedWriter(new FileWriter(fsalida));
                            bufferedWriter.write(cadena + ")");
                            bufferedWriter.newLine();
                            bufferedWriter.close();
                        }
                    }
```

```
                }
            } else {
                System.out.println(" El fichero de entrada no
existe");
            }
        } catch (IOException e) {
            System.out.println(" Error E/S: " + e);
        }
    }

    public static void main(String[] args) {
// O(1)

        String h = "h";
// 1
        String t = "t";
// 1

        // Argumentos
        switch (args.length) {
// 1
            case 1 : {
                File entrada = new File(args[0]);
                if (args[0].equals(h)) {                          // Si
el argumento 0 es igual a "h", que salga el menú de ayuda.
                    ayuda();
                } else if (args[0].equals(t)) {                   // Si
el argumento 0 es igual a "t", nos salta un error indicándonos que
tenemos que introducir una entrada.
                    System.out.println(" Debe introducir una
entrada\n");
                } else if (entrada.exists()) {                    // Si
el argumento 0 contiene una entrada, que nos muestre la salida por
consola.
                    String txtEntrada = entrada.toString();
                    programa(h, t, txtEntrada, "0");
                } else {                                          // Si
el argumento 0 contiene otro tipo que no sea los anteriores indicados,
que nos lance un error.
                    System.out.println(" Los parámetros introducidos
son erróneos.\n");
                }
                break;
            }
            case 2: {
// 1
                File entrada2 = new File(args[0]);
                File salida2 = new File(args[1]);
                File entrada2_ = new File(args[1]);
                if (args[0].equals(t) && args[1].equals(h) ||
```

```
args[0].equals(h) && args[1].equals(t)) {  // Si los argumentos 0 y 1
es igual a h y t o viceversa, muestra el menú de ayuda y lanza un
error.
                    ayuda();
                    System.out.println(" Para que muestre la traza
debe introducir una entrada.\n");
                } else if (args[0].equals(t) && entrada2_.exists()) {
// Si el argumento 0 contiene t y el segundo contiene una entrada, que
nos muestre la salida y su traza.
                    String txtEntrada = entrada2_.toString();
                    programa(t, h, txtEntrada, "0");
                    System.out.println(resumen);
                } else if (args[0].equals(h) && entrada2_.exists()) {
// Si el argumento 0 contiene h y el segundo contiene una entrada, que
nos muestre la salida y el menú de ayuda.
                    String txtEntrada = entrada2_.toString();
                    ayuda();
                    programa(t, h, txtEntrada, "0");
                } else if (entrada2.exists() && !salida2.exists()) {
// Si el argumento 0 contiene una entrada y el segundo una salida.txt
no creada, que nos cree la salida.txt
                    String txtEntrada = entrada2.toString();
                    String txtSalida = salida2.toString();
                    programa(t, h, txtEntrada, txtSalida);
                    System.out.println(" Archivo de salida creada. Si
no se ha creado es porque has introducido mal la ubicación o el
formato de salida (txt)");
                } else if (entrada2.exists() && salida2.exists()) {
// Si el argumento 0 contiene una entrada y el segundo una salida.txt
creada, que nos lance un error.
                    System.out.println(" El archivo de salida ya
existe.\n");
                }
                break;
            }
            case 3: {
// 1
                File entrada3 = new File(args[1]);
                File salida3 = new File(args[2]);
                if (args[0].equals(t) && entrada3.exists() &&
!salida3.exists()) {  // Si el argumento 0 contiene "t", argumento 1
contiene una entrada y argumento 2 contiene una salida.txt no creada,
que nos muestre su traza y nos cree una salida.txt
                    String txtEntrada2 = entrada3.toString();
                    String txtSalida2 = salida3.toString();
                    programa(t, h, txtEntrada2, txtSalida2);
                    System.out.println(resumen);
                    System.out.println(" Archivo de salida creada. Si
no se ha creado es porque has introducido mal la ubicación o el
formato de salida (txt)");
```

```
                } else if (args[0].equals(h) && entrada3.exists() &&
!salida3.exists()) {  // Si el argumento 0 contiene "h", arguemento 1
contiene una entrada y argumento 2 contiene una salida.txt no creada,
que nos muestre el menú de ayuda y nos cree una salida.txt
                    String txtEntrada2 = entrada3.toString();
                    String txtSalida2 = salida3.toString();
                    ayuda();
                    programa(t, h, txtEntrada2, txtSalida2);
                    System.out.println(" Archivo de salida creada. Si
no se ha creado es porque has introducido mal la ubicación o el
formato de salida (txt)");
                }  else if (args[0].equals(t) && entrada3.exists() &&
salida3.exists()) {  // Si el argumento 0 contiene "t", arguemento 1
contiene una entrada y argumento 2 contiene una salida.txtcreada, que
nos muestre error de salida.
                    System.out.println(" El archivo de salida ya
existe.\n");
                }
                else {    // Si los argumentos contiene algún dato no
especificado anteriormente, nos lanzará un error.
                    System.out.println(" Los parámetros introducidos
son erróneos.\n");
                }
                break;
            }
            case 4: {
// 1
                File entrada4 = new File(args[2]);
                File salida4 = new File(args[3]);  // Si el argumento
0 contiene t, argumento 1 contiene h, argumento 2 contiene una entrada
y argumento 3 contiene una salida.txt no creada, que nos muestre su
trazado y nos cree una salida.txt
                if (args[0].equals(t) && args[1].equals(h) &&
entrada4.exists() && !salida4.exists() || args[0].equals(h) &&
args[1].equals(t) && entrada4.exists() && !salida4.exists()) {
                    String txtEntrada3 = entrada4.toString();
                    String txtSalida3 = salida4.toString();
                    ayuda();
                    programa(t, h, txtEntrada3, txtSalida3);
                }
                break;
            }
        }
    }
}
```