

Grado en Ingeniería Informática.  
Universidad de Almería.

Estructura de Datos y Algoritmos II

# Memoria de la práctica III y IV.

**Unidad de Trabajo:**

Marouan Khoullala.

Emilio Cara Garzón

# ÍNDICE:

|   |           |
|---|-----------|
| <b>1. Descripción de la práctica.....</b>                               | <b>3</b>  |
| <b>2. Definición del problema. ....</b>                                 | <b>4</b>  |
| <b>3. Fases de análisis. ....</b>                                       | <b>4</b>  |
| <b>4. Precondiciones y postcondiciones. ....</b>                        | <b>4</b>  |
| <b>5. Diagrama de clases y estructura de datos. ....</b>                | <b>5</b>  |
| <b>6. Explicación y cálculo teórico de algoritmos.....</b>              | <b>9</b>  |
| <b>6.1 Programación Dinámica.....</b>                                   | <b>10</b> |
| <b>6.2 Backtracking. ....</b>   | <b>11</b> |
| <b>7. Demostración experimental y comparación entre algoritmos.....</b> | <b>12</b> |
| <b>8. Generador y Archivos de entrada. ....</b>                         | <b>12</b> |
| <b>9. Archivos de salida. ....</b>                                      | <b>12</b> |
| <b>10. Guía de uso de la aplicación. ....</b>                           | <b>13</b> |
| <b>11. Conclusión. ....</b>   | <b>13</b> |

## 1. Descripción de la práctica.

En este documento, realizaremos la memoria explicativa de las prácticas 3 y 4 cuyo objetivo es el uso y desarrollo de los algoritmos: Programación Dinámica y . Para ello, resumiremos el enunciado de ambas prácticas, empezando por la primera práctica:

Hemos sido contratados por el Departamento de Sanidad (HD) de la expansión urbana de New Almería responsable de hacer un seguimiento de la evolución de la COVID SARS-2. El contrato consiste en dar soporte informático a la gestión de los datos de la pandemia.

Nuestra **ciudad**, New Almeria tiene una estructura en cuadrícula de calles (de oeste a este) y avenidas (de sur a norte). Existen en total “ $m$ ” **avenidas** y “ $n$ ” **calles**.

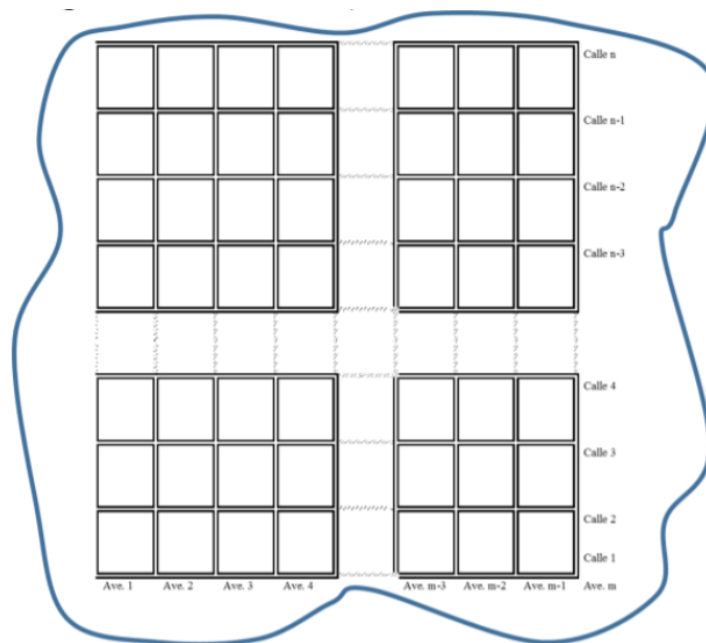


Figura 1 Damero simplificado de New Almeria,  $m$  avenidas y  $n$  calles.

## 2. Definición del problema.

Para realizar este problema vamos a hacer uso de las clases “Distrito”, “Ciudad”, “Main” y “CalcularTiempo”, las cuales explicaremos su funcionamiento más adelante.

En esta práctica se desea específicamente el desarrollo del algoritmo (o algoritmos) que permita/n calcular la selección de distritos a cribar según las tres listas que se piden.

El algoritmo de PD deberá utilizar parámetros para todos los cálculos; en este caso pueden simplificar la entrada de parámetros utilizando datos leídos de una tabla/archivo o leerlos por pantalla. Pueden considerar que los datos están disponibles sin necesidad de conectar con la práctica 2 (histogramas)

## 3. Fases de análisis.

Los datos son las parcelas que hayan superado el consumo de 700% este mes, respecto al anterior. Estos datos estarán dentro de un archivo .txt.

Una vez se reciben los datos, se estudian con los distintos algoritmos (Mochila normal y mochila inversa) para detectar qué clientes son los que obtendrán la reparación de las fugas. Estos datos se presentarán por consola.

## 4. Precondiciones y postcondiciones.

A continuación, se van a enumerar las precondiciones, cualquier incumplimiento de dichas precondiciones no se asegura el correcto funcionamiento del programa.

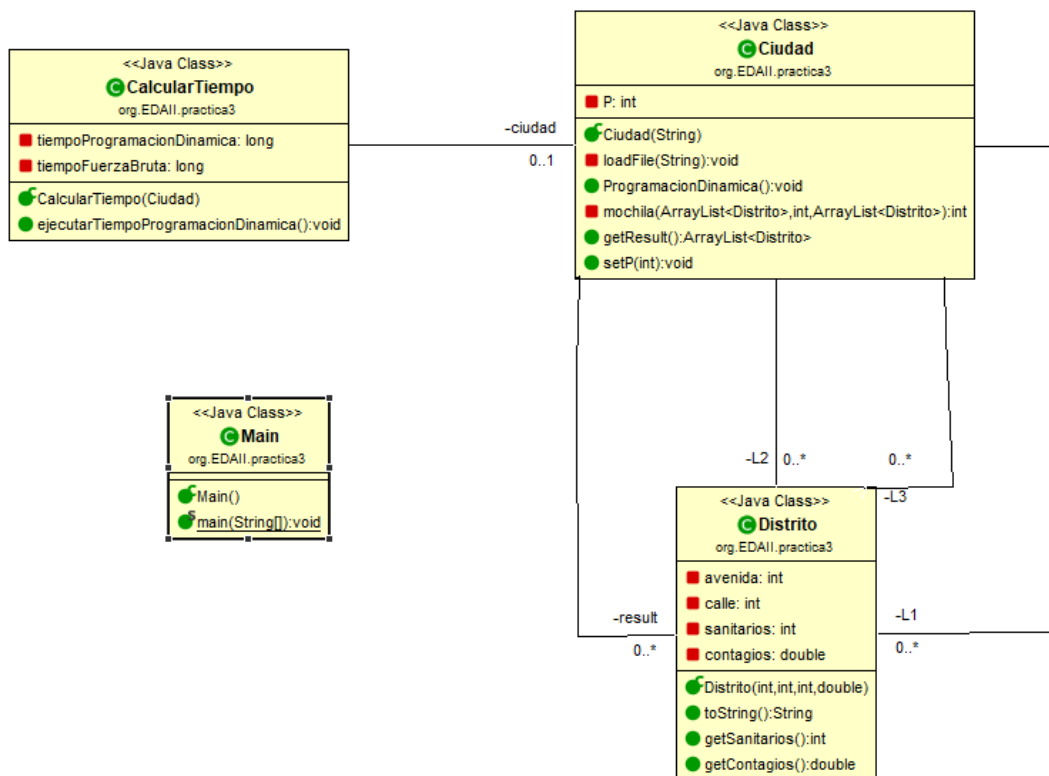
- Los valores de las habilidades de los policías deben ser mayor que 0.
- Los valores de las compatibilidades deben ser mayor que 0.
- Existe un contador al inicio de cada tramo.
- El archivo de entrada debe tener una extensión .txt
- La estructura del archivo de entrada debe de ser como las que se encuentran de ejemplo.










- El programa solo reconoce los números que se indican, en caso de introducir un número no valido u otro carácter se mostrara un aviso de “número no reconocido”.

Las postcondiciones serían:









## 5. Diagrama de clases y estructura de datos.

Hemos distinguido un diagrama de clases para cada práctica. El correspondiente a la Practica 3 sería:






|   |                              |
|---|------------------------------|
| <<Java Class>>  |                              |
|  <b>Distrito</b> |                              |
| org.EDAll.practica3   |                              |
|                  | avenida: int                 |
|                  | calle: int                   |
|                  | sanitarios: int              |
|                  | contagios: double            |
|                  | Distrito(int,int,int,double) |
|                  | toString():String            |
|                  | getSanitarios():int          |
|                  | getContagios():double        |





- **Distrito()**: Constructor de distrito. Se usa para crear los distritos.
- **toString():String**: Devuelve la avenida y la calle seguido de los sanitarios y los contagios.
- **getSanitarios():int**: Devuelve el número de sanitarios.
- **getContagios():double**: Devuelve el número de contagios.

|   |  |
|---|--|
| <<Java Class>>  |  |
|  <b>Ciudad</b> |  |
| org.EDAll.practica3   |  |
|                | P: int   |
|                | Ciudad(String)   |
|                | loadFile(String):void                                    |
|                | ProgramacionDinamica():void                              |
|                | mochila(ArrayList<Distrito>,int,ArrayList<Distrito>):int |
|                | getResult():ArrayList<Distrito>                          |
|              | setP(int):void   |

- **Ciudad()**: Constructor de ciudad. Se usa para crear la ciudad.
- **loadFile(String):void**: Este es el método responsable de la carga del archivo datos.txt
- **ProgramacionDinamica():void**: Este método es en el cuál se asignan la mochila y el distrito
- **mochila():int**: Este es el método que ejecuta la programación dinámica con los datos del archivo y devuelve el peso libre pensando en el siguiente listado
- **getResult()**: Devuelve el resultado
- **setP(int):void**: Establece el peso.

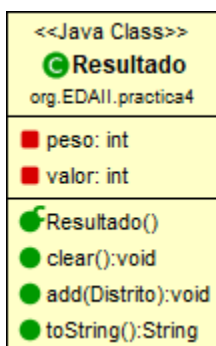
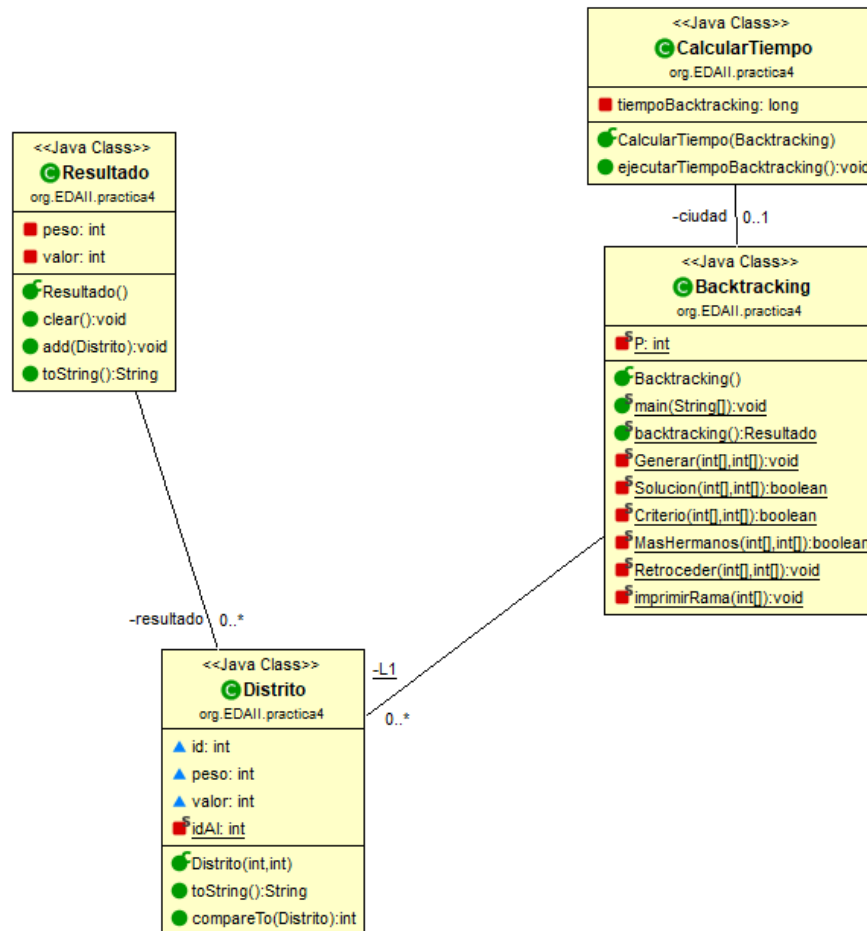
|   |                     |
|---|---------------------|
| <<Java Class>>  |                     |
|  <b>Main</b> |                     |
| org.EDAll.practica3   |                     |
|              | Main()              |
|              | main(String[]):void |

- Aquí se encuentra la dirección del archivo datos.txt y las ejecuciones del algoritmo

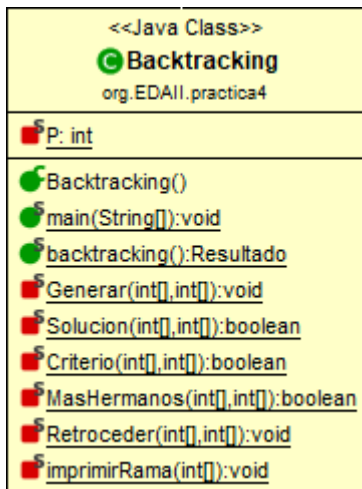
|   |   |
|---|---|
| <<Java Class>>  |   |
|  <b>CalcularTiempo</b> |   |
| org.EDAll.practica3   |   |
|                        | tiempoProgramacionDinamica: long          |
|                        | CalcularTiempo(Ciudad)                    |
|                        | ejecutarTiempoProgramacionDinamica():void |

- Esta clase es necesaria para saber la duración del programa al ejecutar los algoritmos para los diferentes tamaños del programa.

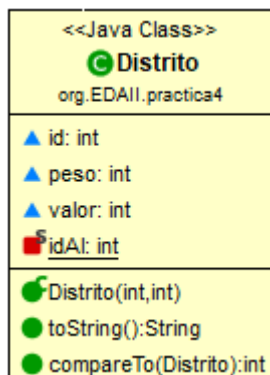
Y el correspondiente a la Practica 4 sería:



- **Resultado():** Devuelve el resultado del algoritmo.
- **clear():** Limpia el árbol.
- **add():** Añade ramas al árbol.
- **toString():** Devuelve el string.







- **Backtracking():** Ejecuta el algoritmo de backtracking que se pide en el problema.
- **main():** El main ejecuta el algoritmo con los requisitos que se piden.
- **Generar():** Generamos el peso y el valor bajo un if.
- **Solucion():** Imprimimos la rama y generamos nueva rama o no, bajo una condición.
- **Criterio():** Aquí escogemos el criterio a seguir para imprimir o no la rama.
- **MasHermanos():** Devuelve la comprobación de si tiene o no, más hermanos.
- **Retroceder():** Retrocede en el árbol si se da la condición.
- **ImprimirRama():** Imprime la rama del árbol.



- **Distrito():** Constructor de distrito. Se usa para crear los distritos.
- **toString():** Devuelve la avenida y la calle seguido de los sanitarios y los contagios.
- **compareTo():** Compara los pesos.



|   |                          |
|---|--------------------------|
| <<Java Class>>  |                          |
|  <b>CalcularTiempo</b>             |                          |
| org.EDAll.practica4   |                          |
|                                    | tiempoBacktracking: long |
|  CalcularTiempo(Backtracking)      |                          |
|  ejecutarTiempoBacktracking():void |                          |

- Esta clase es necesaria para saber la duración del programa al ejecutar los algoritmos para los diferentes tamaños del programa

## 6. Explicación y cálculo teórico de algoritmos.

El algoritmo de programación dinámica establece el peso a 0 con un if, después inicializa el array con el tamaño del listado y ejecutamos un bucle for que se ejecute mientras que i sea menor que la longitud de B, dentro del bucle ejecutamos otro bucle que se ejecute hasta que j sea menor que el listado más el numero de sanitarios. Posteriormente añadimos la variable pesoLibre para ejecutar otro bucle y obtener como return el pesoLibre, que posteriormente servirá para el resto de los listados (Listado 2 y listado 3).

El algoritmo de backtracking funciona de la siguiente manera:

Establecemos el nivel a 0 y un array s al tamaño del listado 1. Inicializamos el array s a -1 y abrimos un bucle "while" que se ejecute mientras el nivel sea mayor que -1. Dentro del bucle ejecutamos el método "generar" para que genere las ramas, después hacemos un "if" para hacer o no un "clear" y un "for" para añadir elementos al árbol. Finalmente establecemos un "if" para comprobar que se respete el criterio y si no se respeta ejecutar el "mas hermano" y "retroceder" en el árbol. Finalmente retorna la solución.

## 6.1 Programación Dinámica.

```
private int mochila(ArrayList<Distrito> L, int P, ArrayList<Distrito>
result) {
    if(P==0) {
        return 0;
    }
    double[][] B = new double[L.size()+1][P+1];
    for (int i = 1; i < B.length; i++) {
        for (int j = 0; j < L.get(i-1).getSanitarios(); j++) {
            B[i][j] = B[i-1][j];
        }
        for (int j = L.get(i-1).getSanitarios(); j <
B[0].length; j++) {
            B[i][j] = Math.max(B[i-1][j], B[i-1][j-L.get(i-
1).getSanitarios()] + L.get(i-1).getContagios());
        }
    }
    int pesoLibre = P;
    int j = B[0].length-1;
    for (int i = B.length-1; i > 0; i--) {
        if(B[i][j] != B[i-1][j]) {
            result.add(L.get(i-1));
            j -= L.get(i-1).getSanitarios();
            pesoLibre=L.get(i-1).getSanitarios();
        }
    }
    return pesoLibre;
}
```

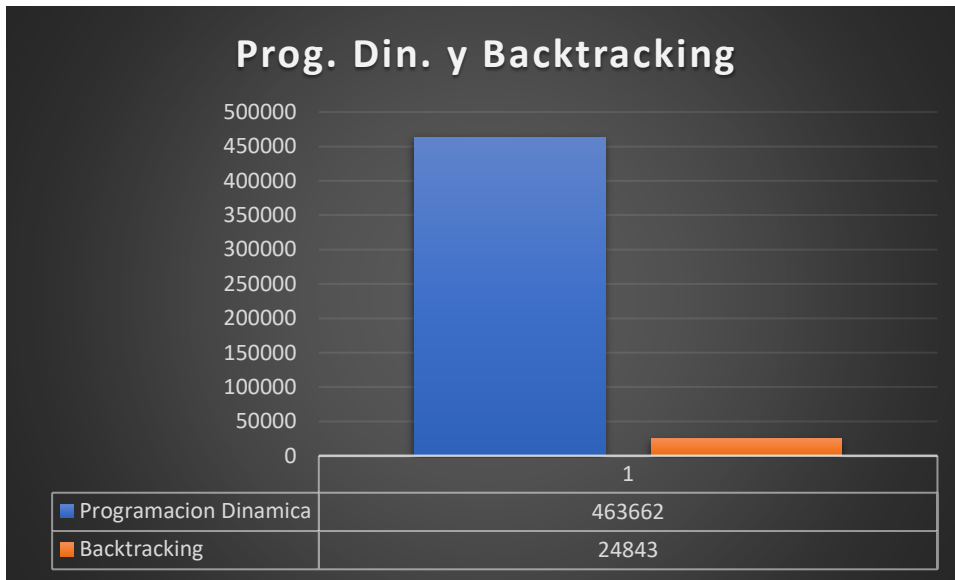
## 6.2 Backtracking.

```

public static Resultado backtracking() {
    int[] a = {0, Integer.MAX_VALUE, 0, 0};
    int[] s = new int[L1.size()];
    Arrays.fill(s, -1);
    Resultado soa = new Resultado();
    while (a[0] > -1) {
        Generar(a, s);
        if(Solucion(a, s)) {
            a[1] = a[3];
            soa.clear();
            for (int i = 0; i < s.length; i++) {
                if(s[i]==1) {
                    soa.add(L1.get(i));
                }
            }
        }
        if(Criterio(a,s)) {
            a[0]++;
        }else {
            while(a[0]>-1 && !MasHermanos(a,s)) {
                Retroceder(a,s);
            }
        }
    }
    return soa;
}

```

## 7. Demostración experimental y comparación entre algoritmos.



Como se puede ver en las gráficas el algoritmo referente al Backtracking es muchísimo más eficiente que el algoritmo de programación dinámica en lo referente al tiempo de ejecución de ambos problemas en ms

## 8. Generador y Archivos de entrada.

Podemos entender como archivos de entrada los casos de prueba o ficheros que escogemos para cargar en el programa y ejecutar el algoritmo. Dichos casos de entrada contienen la puntuación de los policías y su compatibilidad con otros compañeros.

## 9. Archivos de salida.

El programa no genera en si ningún archivo de salida físico(fichero). Como archivo de salida muestra por pantalla el resultado de alguna operación y este se puede acumular en variables para utilizarlo en otras operaciones.

## 10. Guía de uso de la aplicación.

El programa referente a la práctica 3 ejecuta el archivo datos.txt que tu elijas, yendo desde datos.txt hasta datos8.txt pasando por datos2.txt o datos4.txt.

El programa referente a la práctica 4 ejecuta los datos que incluimos directamente en el código del programa, estando claramente diferenciada esa parte del código para poder facilitar así la implementación de varios datos de prueba.

## 11. Conclusión.

Hemos optimizado todo lo que hemos podido y garantizado el correcto funcionamiento del programa no obstante creemos que podría haber una manera más rápida de hacer el algoritmo de backtracking. Lo hemos ajustado para tenerlo listo en el tiempo establecido.