VERSIÓN 1.0.0

PREDA 20 / 01 / 2022

PRÁCTICA DE PROGRAMACIÓN Y ESTRUCTURAS DE DATOS AVANZADAS – CURSO 2021-2022

SKYLINE BASADO EN DIVIDE Y VENCERÁS

FERMÍN GORRAIZ GARCÍA -72806064R

fermin.gorraiz@gmail.com

646 612 485

CONTENIDO

SKYLINE BASADO EN DIVIDE Y VENCERÁS

OBJETIVO PRINCIPAL

Sobre una ciudad se alzan los edificios dibujando una línea de horizonte conocida como skyline. El problema consiste en calcular la línea de horizonte de una ciudad en forma de una secuencia de puntos sobre el plano. Se denomina el problema del skyline de una ciudad.

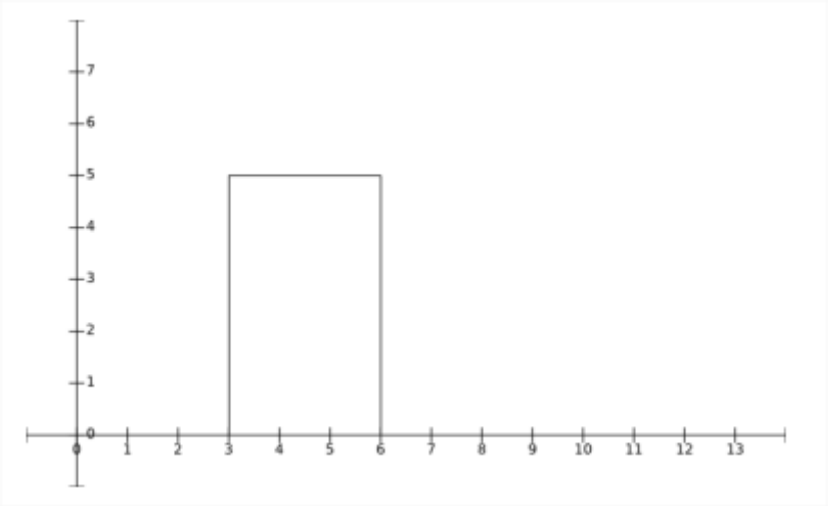
La práctica constará de un programa en java que resuelva el problema aplicando el esquema de Divide y Vencerás.

ANÁLISIS DEL ALGORITMO

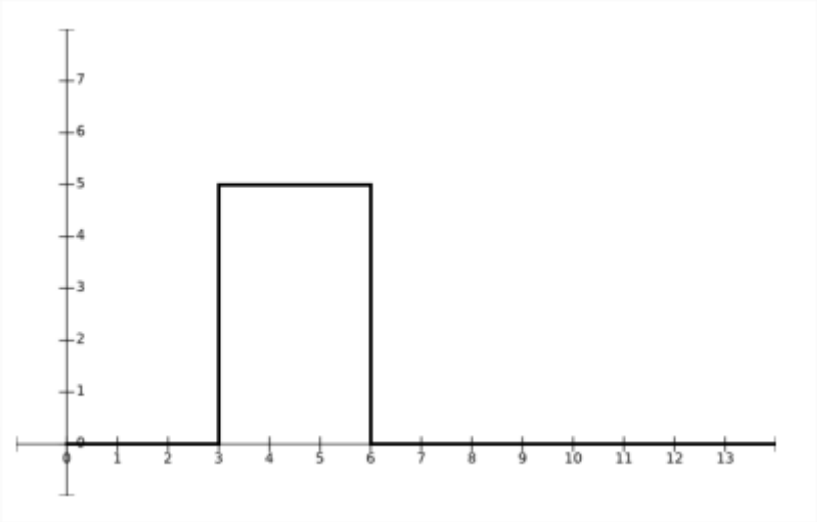
Este algoritmo recibe como entrada un conjunto (o lista de edificios), cada uno de los cuales se supone rectangular. Dichos edificios se representan por una terna de números enteros (x1,x2,h):

- `x1` primera coordenada x del edificio
- `x2` segunda coordenada x del edificio
- `h` altura del edificio

Así, si tenemos un edificio representado por la terna (3,6,5), significará que dicho edificio se extiende desde la coordenada x 3 hasta la coordenada x 6 con una altura de 5:

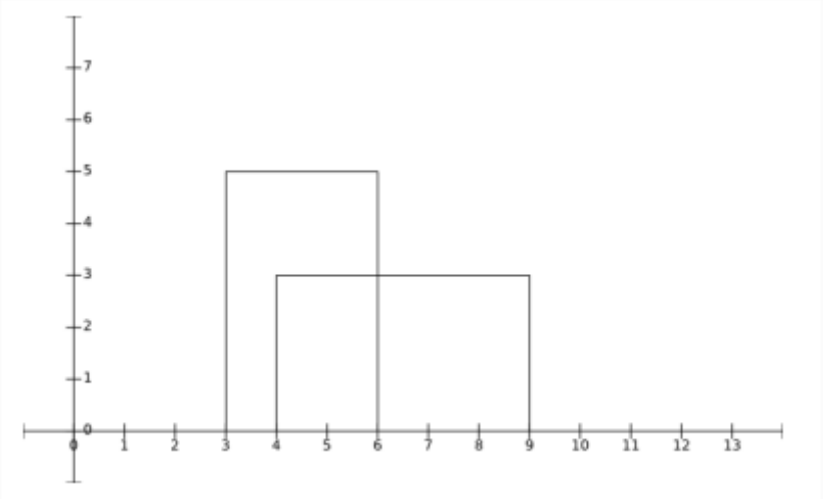


La salida devuelta por el algoritmo consiste en una línea de horizonte, que vendrá representada como una lista ordenada de puntos en el plano donde dicha línea cambia de altura. Así, pues, para el edificio anterior, la línea de horizonte sería [(3,5),(6,0)], puesto que en la coordenada x=3, la línea de horizonte sube desde la altura y=0 a la altura y=5 y en la coordenada x=6 la línea de horizonte baja nuevamente a altura y=0.

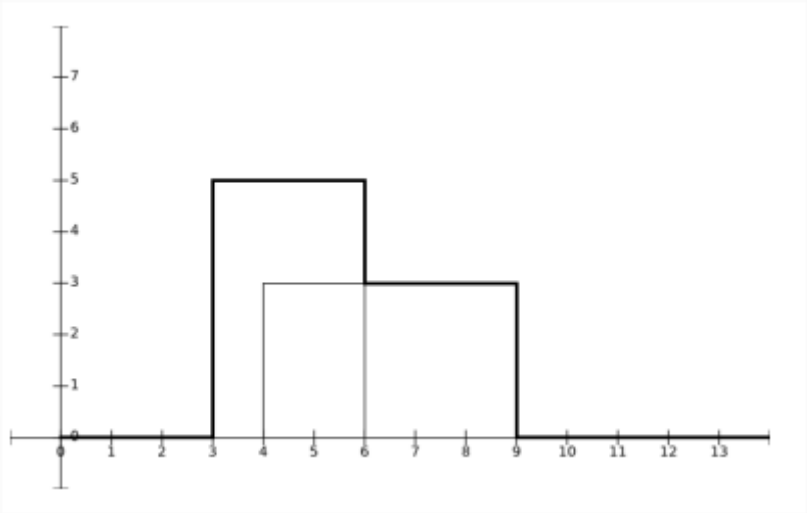


COMBINACIÓN DE DOS LÍNEAS DE HORIZONTE

Cuando hay más de un edificio, la línea de horizonte debe representar todos los cambios de altura. Así, para dos edificios (3,6,5) y (4,9,3):



la línea de horizonte resultante debería ser [(3,5),(6,3),(9,0)]:



A continuación, vamos a ver cómo se ha de realizar esta combinación de dos líneas de horizonte.

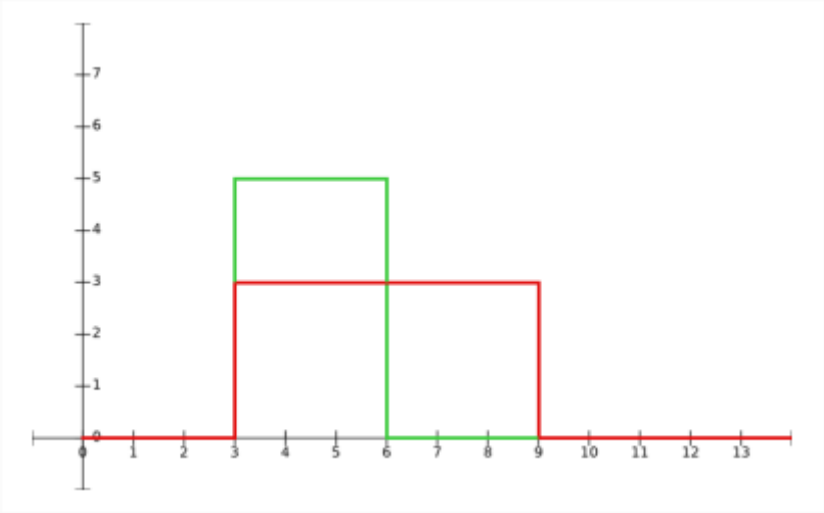
La idea fundamental es comparar el primer punto de ambas líneas de horizonte y obtener la coordenada x y la altura del siguiente punto de la línea de horizonte resultado. El proceso continuará hasta que se hayan consumido todos los puntos de una de las líneas de horizonte.

Distinguiremos dos casos fundamentales según sea el siguiente punto de cada una de las dos líneas de horizonte a combinar:

- Las dos líneas de horizonte comienzan en la misma coordenada x.
- Las dos líneas de horizonte comienzan en distinta coordenada x.

LAS DOS LÍNEAS DE HORIZONTE COMIENZAN EN LA MISMA COORDENADA X

Estaríamos ante un caso como el descrito en la siguiente figura:



Tenemos dos líneas de horizonte:

- La línea verde, representada por la lista [(3,5), (6,0)]
- La línea roja, representada por la lista [(3,3), (9,0)]

El siguiente punto de la línea de horizonte resultado tendrá, obviamente, coordenada x=3 y altura y=5, por ser la mayor de las dos.

Es decir: la coordenada x del siguiente punto será la coordenada x de cualquiera de los dos puntos de la línea de horizonte, mientras que la altura será la mayor de ambas.

Además, ninguno de estos puntos iniciales va a volver a influir en la elección de la coordenada x de los siguientes puntos, por lo que se pueden consumir ambos y continuar con el análisis.

LAS DOS LÍNEAS DE HORIZONTE COMIENZAN EN DISTINTA COORDENADA X

Tras haber consumido los dos primeros puntos de ambas líneas de horizonte en el caso anterior, ahora tendríamos que:

- El resto de la línea verde estaría representado por la lista [(6,0)]
- El resto de la línea roja estaría representado por la lista [(9,0)]

Parece lógico pensar que el siguiente punto de la línea de horizonte resultado tendrá como coordenada x la menor de ambas (6 en este caso). Sin embargo, la altura debería ser 3 según se ve claramente en el dibujo, pero la altura de los primeros puntos del resto de las líneas es 0. ¿Cómo podemos obtener ese 3?

Si nos fijamos, la coordenada x del nuevo punto de la línea de horizonte resultado lo obtenemos de la línea verde, mientras que la altura (3) viene dada por la línea roja, ya que es la altura del último punto de dicha línea de horizonte.

Así pues, la función de combinación debe recordar cuál es la altura del último punto consumido de cada línea de horizonte, para poder así calcular correctamente la altura del siguiente punto de la línea de horizonte resultado. Originalmente todas las líneas de horizonte comienzan en altura 0, por lo que ese será el valor con el que habrá que inicializar esas variables.

El cálculo de la altura se realiza, en general, de la siguiente forma:

1. Elegimos el punto que tenga la coordenada x más pequeña.
2. Comparamos la altura de ese punto con la altura del último punto consumido de la otra línea de horizonte y nos quedamos con el valor mayor.
3. Consumimos el punto elegido.

Aplicando este algoritmo al ejemplo:

1. Elegimos el punto (6,0).
2. Comparamos su altura (0) con la altura del último punto consumido de la otra línea de horizonte (3) y nos quedamos con el valor mayor (3).
3. Consumimos el punto elegido, con lo que el resto de la línea de horizonte verde sería [] (la lista vacía).

Una vez que se ha consumido totalmente una de las dos líneas de horizonte, simplemente habrá que añadir el resto de la otra a la salida sin que haga falta realizar ningún tipo de cálculo adicional.

Es posible que en algunas ocasiones se produzcan dos puntos consecutivos con igual altura. Esto puede solucionarse sin más que comparar la altura del posible nuevo punto a generar con la del último generado y en caso de que sean la misma no generar el nuevo punto.

CONCLUSIONES

- Idea básica Divide y Vencerás: dado un problema, descomponerlo en partes, resolver las partes y juntar las soluciones.
- Idea muy sencilla e intuitiva, pero en los problemas reales de interés:
  - Pueden existir muchas formas de descomponer el problema en subproblemas ⇒ Quedarse con la mejor.
  - Puede que no exista ninguna forma viable, los subproblemas no son independientes ⇒ Descarta la técnica.
- Divide y vencerás requiere la existencia de un método directo de resolución:
  - Tamaños pequeños: solución directa.
  - Tamaños grandes: división y combinación.
  - Establecer el límite pequeño/grande.

ANEXO I

CÓDIGO DE LAS CLASES

SKYLINE

PACKAGE MAIN

SKYLINEMAIN.JAVA

```
1.   package main;
2.
3.  import static java.util.Objects.isNull;
4.
5.  import main.common.CommonApiDate;
6.  import main.common.CommonApiMenu;
7.
8.  public class SkylineMain {
9.
10.     private static final String DEFAULT_OUTPUT = "defaultOutput.txt";
11.
12.     private static final String SHOW_TRACE = "-t";
13.     private static final String SHOW_HELP = "-h";
14.
15.     private static boolean showTrace = false;
16.
17.     public static void main(String[] args) {
18.         System.out.println(CommonApiDate.dateNowToString() + " || Launch Skyline\r\n");
19.         boolean isInput = true;
20.
21.         String inputFile = null;
22.         String outputFile = DEFAULT_OUTPUT;
23.
24.         if (args.length > 4) {
25.             System.out.println("Too much args, see [skyline -h] for help\r\n");
26.             return;
27.         } else if (args.length < 1) {
28.             System.out.println("Not enough arguments, see [skyline -h] for help\r\n");
29.             return;
30.         }
31.
32.         for (String a : args) {
33.             switch (a) {
34.             case SHOW_TRACE:
35.                 showTrace = true;
36.                 continue;
37.             case SHOW_HELP:
38.                 System.out.println(CommonApiMenu.showHelp());
39.                 continue;
40.             }
41.
42.             if (isInput) {
43.                 inputFile = a;
44.                 isInput = false;
45.             } else {
46.                 outputFile = a;
47.             }
48.         }
49.
```

```
50.         if (isNull(inputFile))
51.             return;
52.
53.         DyV.run(inputFile, outputFile, showTrace);
54.         System.out.println(CommonApiDate.dateNowToString() + " || Close Skyline\r\n");
55.     }
56.
57. }
58.
```

DYV.JAVA

```
1.  package main;
2.
3.  import java.util.ArrayList;
4.  import java.util.List;
5.
6.  import main.common.CommonApiDate;
7.  import main.common.CommonApiFile;
8.  import main.domain.Building;
9.  import main.domain.Skyline;
10.
11. public class DyV {
12.
13.     public static void run(String inputFile, String outputFile, boolean showTrace) {
14.         Building[] buildings = CommonApiFile.readFile(inputFile);
15.         List<Skyline> skylines = obtenerSkyLines(buildings, 0, (buildings.length - 1),
    showTrace);
16.
17.         boolean isFirst = true;
18.         StringBuilder sb = new StringBuilder();
19.         sb.append("{");
20.         for (Skyline s : skylines) {
21.             if (isFirst) {
22.                 isFirst = false;
23.             } else {
24.                 sb.append(",");
25.             }
26.
27.             sb.append("(");
28.             sb.append(s.abscisa + "," + s.height);
29.             sb.append(")");
30.
31.         }
32.         sb.append("}");
33.
34.         CommonApiFile.createAndWriteFile(outputFile, sb.toString());
35.     }
36.
37.     private static List<Skyline> obtenerSkyLines(Building[] edificios, int i, int j,
    boolean showTrace) {
38.         if (edificios.length == 0)
39.             return new ArrayList<Skyline>();
40.
41.         if (showTrace)
42.             trace("obtenerSkyLines i : " + i + ", j: " + j);
43.
44.         int n = j - i + 1;
45.         if (n == 1) {
46.             ArrayList<Skyline> s = new ArrayList<Skyline>();
47.             s.add(new Skyline(edificios[i].getLeft(), edificios[i].getHeight()));
48.             s.add(new Skyline(edificios[i].getRight(), 0));
49.
50.             return s;
51.         } else {
52.             int m = (i + j - 1) / 2;
53.             List<Skyline> sa = obtenerSkyLines(edificios, i, m, showTrace);
54.             List<Skyline> sb = obtenerSkyLines(edificios, m + 1, j, showTrace);
```

```
55.             return combinarSkyLines(sa, sb, showTrace);
56.         }
57.
58.     }
59.
60.     private static List<Skyline> combinarSkyLines(List<Skyline> sa, List<Skyline> sb,
    boolean showTrace) {
61.         if (showTrace)
62.             trace("combinarSkyLines sa : " + skylinesToString(sa) + ", sb: " +
    skylinesToString(sb));
63.
64.         ArrayList<Skyline> s = new ArrayList<Skyline>();
65.
66.         Skyline a, b;
67.         int ha = 0, hb = 0, uh = 0;
68.         int ia = 0, ib = 0, nx = 0, nh = 0;
69.
70.         while ((ia < sa.size()) && (ib < sb.size())) {
71.             a = sa.get(ia);
72.             b = sb.get(ib);
73.
74.             if (a.getAbscisa() == b.getAbscisa()) {
75.                 nx = a.getAbscisa();
76.                 nh = Math.max(a.getHeight(), b.getHeight());
77.                 ha = a.getHeight();
78.                 hb = b.getHeight();
79.                 ia = ia + 1;
80.                 ib = ib + 1;
81.             } else if ((a.getAbscisa() < b.getAbscisa())) {
82.                 nx = a.getAbscisa();
83.                 nh = Math.max(a.getHeight(), hb);
84.                 ha = a.getHeight();
85.                 ia = ia + 1;
86.             } else {
87.                 nx = b.getAbscisa();
88.                 nh = Math.max(b.getHeight(), ha);
89.                 hb = b.getHeight();
90.                 ib = ib + 1;
91.             }
92.             if (uh != nh) {
93.                 s.add(new Skyline(nx, nh));
94.                 uh = nh;
95.             }
96.         }
97.         while (ia < sa.size()) {
98.             s.add(sa.get(ia++));
99.         }
100.          while (ib < sb.size()) {
101.              s.add(sb.get(ib++));
102.          }
103.
104.          return s;
105.      }
106.
107.      private static void trace(String trace) {
108.          String date = CommonApiDate.dateNowToString();
109.          System.out.println(date + " || " + trace + "\r\n");
110.      }
111.
112.      private static String skylinesToString(List<Skyline> skylines) {
113.          StringBuilder sb = new StringBuilder();
114.
115.          skylines.forEach(s -> sb.append(s.toString()));
116.
117.          return sb.toString();
118.      }
119.  }
120.
```

PACKAGE MAIN.DOMAIN

BUILDING.JAVA

```
1.  package main.domain;
2.
3.  public class Building {
4.      private int left;
5.      private int right;
6.      private int height;
7.
8.      public Building(int left, int right, int height) {
9.          super();
10.         this.left = left;
11.         this.right = right;
12.         this.height = height;
13.     }
14.
15.     public int getLeft() {
16.         return left;
17.     }
18.
19.     public void setLeft(int left) {
20.         this.left = left;
21.     }
22.
23.     public int getRight() {
24.         return right;
25.     }
26.
27.     public void setRight(int right) {
28.         this.right = right;
29.     }
30.
31.     public int getHeight() {
32.         return height;
33.     }
34.
35.     public void setHeight(int height) {
36.         this.height = height;
37.     }
38.
39.     public String toString() {
40.         StringBuilder sb = new StringBuilder();
41.
42.         sb.append(" Building [ ");
43.         sb.append("left: " + left + ",");
44.         sb.append("right: " + right + ",");
45.         sb.append("height: " + height + ",");
46.         sb.append("] ");
47.
48.         return sb.toString();
49.
50.     }
51. }
52.
```

SKYLINE.JAVA

```
1.  package main.domain;
2.
3.  public class Skyline {
4.
5.      public int abscisa;
6.      public int height;
7.
8.      public Skyline(int abscisa, int altura) {
9.          super();
10.         this.abscisa = abscisa;
11.         this.height = altura;
12.     }
13.
14.     public int getAbscisa() {
15.         return abscisa;
16.     }
17.
18.     public void setAbscisa(int abscisa) {
19.         this.abscisa = abscisa;
20.     }
21.
22.     public int getHeight() {
23.         return height;
24.     }
25.
26.     public void setHeight(int height) {
27.         this.height = height;
28.     }
29.
30.     public String toString() {
31.         StringBuilder sb = new StringBuilder();
32.
33.         sb.append(" Skyline [ ");
34.         sb.append("abscisa: " + abscisa + ",");
35.         sb.append("height: " + height + ",");
36.         sb.append("] ");
37.
38.         return sb.toString();
39.
40.     }
41.
42. }
43.
```

PACKAGE COMMON

COMMONAPIDATE.JAVA

```
1.  package main.common;
2.
3.  import java.text.DateFormat;
4.  import java.text.SimpleDateFormat;
5.  import java.util.Date;
6.
7.  public class CommonApiDate {
8.      public static String dateNowToString() {
9.          Date date = new Date();
10.         DateFormat dateFormat = new SimpleDateFormat("hh:mm:ss.SSS");
11.
12.         return dateFormat.format(date);
13.     }
14. }
15.
```

COMMONAPIFILE.JAVA

```
1.  package main.common;
2.
3.  import java.io.File;
4.  import java.io.FileNotFoundException;
5.  import java.io.FileWriter;
6.  import java.io.IOException;
7.  import java.util.LinkedList;
8.  import java.util.List;
9.  import java.util.Scanner;
10.
11. import main.domain.Building;
12.
13. public class CommonApiFile {
14.     public static Building[] readFile(String filename) {
15.         List<Building> edificios = new LinkedList<Building>();
16.         try {
17.             File myObj = new File(filename);
18.             Scanner myReader = new Scanner(myObj);
19.             while (myReader.hasNextLine()) {
20.                 String data = myReader.nextLine();
21.                 String[] split = data.split(",");
22.                 edificios.add(new Building(Integer.parseInt(split[0]),
    Integer.parseInt(split[1]),
23.                         Integer.parseInt(split[2])));
24.             }
25.             myReader.close();
26.         } catch (FileNotFoundException e) {
27.             System.out.println("An error occurred.\r\n");
28.             e.printStackTrace();
29.         }
30.
31.         return edificios.toArray(new Building[edificios.size()]);
32.     }
33.
34.     public static void createAndWriteFile(String filename, String text) {
35.         createFile(filename);
36.         writeToFile(filename, text);
37.     }
38.
39.     public static void createFile(String filename) {
40.         try {
41.             File myObj = new File(filename);
42.             if (myObj.createNewFile()) {
43.                 System.out.println("File created: " + myObj.getName() + "\r\n");
```

```
            }
        } catch (IOException e) {
            System.out.println("An error occurred.\r\n");
            e.printStackTrace();
        }
    }

    public static void writeToFile(String filename, String text) {
        try {
            FileWriter myWriter = new FileWriter(filename);
            myWriter.write(text);
            myWriter.close();
            System.out.println("Successfully wrote to the file: " + filename + "\r\n");
        } catch (IOException e) {
            System.out.println("An error occurred.\r\n");
            e.printStackTrace();
        }
    }
}

```

COMMONAPIMENU.JAVA

```
package main.common;

public class CommonApiMenu {

    public static String showHelp() {
        StringBuilder sb = new StringBuilder();

        sb.append("SINTAXIS: skyline [-t][-h] [fichero entrada] [fichero salida]\r\n");
        sb.append("-t Traza cada llamada recursiva y sus parámetros\r\n");
        sb.append("-h Muestra esta ayuda\r\n");
        sb.append("[fichero entrada] Conjunto de edificios de la ciudad\r\n");
        sb.append("[fichero salida] Secuencia que representan el skyline de la
    ciudad\r\n");

        return sb.toString();
    }

    public static String showFileNames(String inputFile, String outputFile) {
        StringBuilder sb = new StringBuilder();

        sb.append("fichero entrada: " + inputFile + "\r\n");
        sb.append("fichero salida: " + outputFile + "\r\n");

        return sb.toString();
    }
}

```