Memoria Práctica 2

Algoritmia básica

David Jiménez Omeñaca (825068)
Carlos Giralt Fuixench (831274)

Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

Ingeniería Informática
Universidad de Zaragoza
Zaragoza,España
2023-2024

Índice

1. Introducción

La presente memoria detalla el planteamiento, la metodología, la implementación y el análisis de resultados obtenidos durante la realización de la segunda práctica de la asignatura Algoritmia Básica. Su objetivo es la implementación de un algoritmo de búsqueda con retroceso que determine la disposición óptima de un conjunto de artículos en una página de un periódico o revista de modo que se emplee el mayor área disponible posible. Estos artículos, así como sus dimensiones y demás características se detallan en un fichero de entrada. Del mismo modo, los resultados obtenidos por el algoritmo se almacenan en un fichero de salida. Tanto el formato del fichero de entrada como el de salida pueden consultarse en A.

2. Metodología y recursos empleados

En primer lugar, se analizó en profundidad el problema planteado, prestando especial atención a las variables que intervienen en él para, posteriormente, definir las **clases** necesarias para resolverlo. A continuación, nos centramos en el diseño de una buena **cota de poda**, pues este es un aspecto fundamental en cualquier algoritmo de búsqueda con retroceso ya que permite reducir considerablemente, si está bien elegida, el número de nodos visitados en la construcción de la solución óptima.

Una vez familiarizados con el problema, planteamos una primera aproximación en pseudocódigo (ver B). Satisfechos con nuestro planteamiento, pasamos a la implementación del mismo. Escogimos, como lenguaje de programación, **Python**, pues es un lenguaje fácilmente manejable y legible y que cuenta con una amplia variedad de librerías de gran utilidad y que se adaptan de maravilla a la complejidad y dimensión del problema a resolver. En particular, se hizo uso de la clase **Polygon**, implementada en la librería `shapely.geometry`. Esta clase permite definir polígonos a partir de las coordenadas de sus vértices y ofrece métodos para determinar si dos instancias de dicha clase se intersecan, generar la unión de varios polígonos, y calcular su área, entre otras.

Finalmente, diseñamos una batería de pruebas para comprobar el correcto funcionamiento de nuestra solución. Dichas pruebas, así como una breve explicación de las mismas, pueden consultarse en C.

3. Diseño e Implementación

Comenzamos con un conjunto de artículos $\mathbf{A} = \{A_1, \ldots, A_n\}$ y una página $P \subset R^2$. Cada artículo ocupa una superficie $\mathbf{S} = \{S_1, \ldots, S_n\}$ satisfaciendo que $S_i \subset P$. Definimos también la función área dada por Area : $R^2 \to R^+$ que devuelve el área de un subconjunto del plano. Para nuestro problema, queremos encontrar una tupla $(x_1, \ldots, x_k)$ con $x_k \in \{1, \ldots, n\}$ $\forall k$ tales que

$$\text{Area}((x_1, \ldots, x_k)) := \sum_{i=1}^{k} \text{Area}(A_{x_i})$$

sea máxima. Para ello vamos a recurrir a un proceso recursivo donde vamos a ir formando las tuplas sucesivamente. Empezamos definiendo las restricciones **explícitas del problema**:

- El **espacio de búsqueda** es $S = C \times \cdots \times C$ donde $C = \{1, \ldots, n\}$, es decir, $x_i \in \{1, \ldots, n\}$ $\forall i$

Para las **restricciones implícitas**:

- Las intersecciones dos a dos de los artículos de índices $(x_1, \ldots, x_k)$ deben ser vacías, es decir:

$$S_{x_i} \cap S_{x_k} = \emptyset \quad \forall i, k \in \{1, \ldots, n\}$$

- Las tuplas deben cumplir que $x_j < x_{j+1},\ 1 \leq j \leq i-1$.

Una vez definidos el espacio de búsqueda y las restricciones explícitas e implícitas, definimos los siguientes predicados acotadores:

Para el predicado acotador, necesitamos primero una función de cota, que, dada una tupla parcial, indique el mayor área que se puede construir a partir de dicha solución. En nuestro caso la calculamos tal que

$$\begin{aligned} U = U(\{x_1, \ldots, x_k\}) &:= \bigcup_{i=1}^{k} A_{x_i} \\ V &= \{i \in \{1, \ldots, n\} \mid A_i \cap U = \emptyset\} \\ C(\{x_1, \ldots, x_k\}) &= \text{Area}\left(\bigcup_{i \in V} A_i\right) \end{aligned}$$

Sea $Z = \{z_1, \ldots, z_r\}$ la mejor solución factible encontrada hasta el momento. Esta solución sólo es modificada en caso de hallar otra mejor, en cuyo caso almacenamos la última.

Podemos entonces definir nuestro predicado acotador como:

$$P(\mathbf{x}, i+1) = C(\mathbf{x}) + \text{Area}(\mathbf{x}) > \text{Area}(\mathbf{Z})$$

De no cumplirse lo anterior, sería imposible hallar por esta rama una solución mejor a la ya encontrada añadiendo el artículo que se está considerando. Por lo tanto, se poda.

Como ya se ha comentado en las restricciones implícitas, para saber si una tupla es solución, tenemos que comprobar que no hay intersecciones:

$$R(\mathbf{x}, i+1) = (A_{x_i} \cap A_{x_j} = \emptyset)\ \forall i, j$$

Por otro lado, el conjunto de los valores posibles de x_{i+1} tales que $\{x_1, \ldots, x_{i+1}\}$ es un camino hasta un nodo de un árbol, son los artículos $\{A_j,\ j > i+1\}$.

$$G(\mathbf{x}, i+1) = \{i+2, \ldots, n\}$$

Nótese que el tamaño del espacio de búsqueda es: 2^n.

4. Análisis temporal

Se han realizado tests con un número de artículos creciente para ver el crecimiento exponencial del problema. Notar que estos tests son muy exhaustivos y que la cota no poda demasiadas ramas debido al elevado número de intersecciones entre los artículos que componen los tests, de ahí los elevados tiempos. Los `.svg` se pueden ver en 1 y las gráficas de resultados en 2.

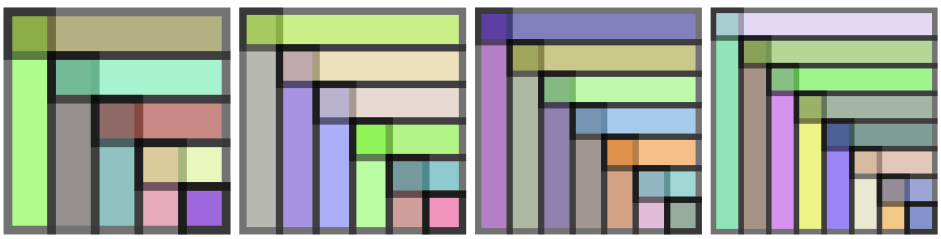

Figura 1: Svgs de los tests de tiempos.

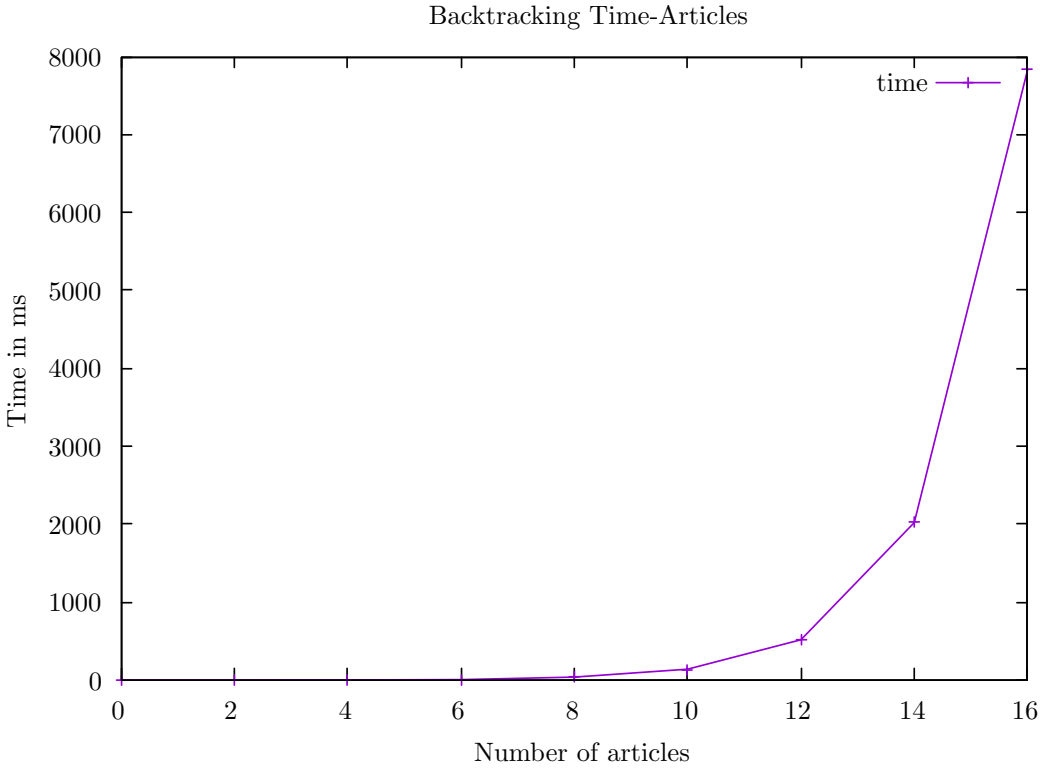

Figura 2: Relación entre el número de artículos y el tiempo (en **ms**) que tarda el programa en hallar la solución

5. Resultados y conclusiones

En esta práctica se ha experimentado con la técnica algorítmica de **backtracking** para resolver un problema con un espacio de búsqueda de un tamaño considerable. Si bien es cierto que los resultados obtenidos en la mayoría de pruebas realizadas son más que prometedores, la cota puede resultar de poca utilidad en casos en los que se produzcan muchas intersecciones entre los artículos considerados. En cualquier caso, es evidente que el uso de este tipo de técnicas es necesario y recomendado para problemas de este estilo, pues reducen en cierto grado el número de nodos, o soluciones posibles, a considerar en la resolución del problema, a diferencia de una aproximación iterativa o recursiva.

Estamos muy satisfechos con el trabajo realizado y su posterior análisis de resultados, así como con la documentación aportada.

A. Formato de los ficheros de entrada y salida

A.1. Formato de los ficheros de entrada

Para garantizar la correcta ejecución de la solución propuesta, se debe asegurar que los ficheros de entrada siguen el siguiente formato: el fichero de entrada está organizado en bloques. Cada bloque es una instancia diferente del problema. La primera línea de cada bloque consta de 3 números reales: el número de artículos n, la anchura W y altura H de la página. Las n líneas siguientes representan los artículos a colocar en la página. Para cada artículo hay cuatro números: anchura w_i, altura h_i, y coordenadas cartesianas x_i,y_i de su extremo superior izquierdo. Por último, los ficheros de entrada deben contener una línea en blanco después del último bloque de datos. A continuación se muestra un ejemplo de fichero de entrada que especifica dos instancias del problema.

```
1   3 280 400
2   140 200 0 0
3   140 200 140 0
4   200 400 0 0
5   4 280 400
6   120 120 0 0
7   130 130 0 0
8   210 210 20 10
9   260 380 20 20
10
```

Figura 3: Ejemplo de fichero de entrada

Ambas páginas tienen la misma dimensión `(280 x 400 mm)`. En la primera hay $n = 3$ artículos a colocar, mientras que en la segunda hay $n = 4$.

A.2. Formato del fichero de salida

La primera línea del fichero de salida indica el fichero de entrada a partir del cual se han obtenido los resultados. A continuación, hay tantas líneas como número de bloques tiene el fichero de entrada además de una última línea en blanco. Cada línea contiene, en primer lugar, el número de artículos del problema, seguido por el área (en mm) total ocupada por los artículos elegidos, el número de nodos visitados en el proceso de construcción de la solución y, finalmente, el tiempo (en ms) empleado por el algoritmo en encontrar la solución óptima. La siguiente figura contiene, a modo de ejemplo, el fichero de salida correspondiente a la ejecución de nuestra solución, usando el fichero **test0.txt** como fichero de entrada.

```
1   Results from test0.txt
2   112000.0 1 0.0008008480072021484
3   112000.0 2 0.005618095397949219
4   80000.0 6 0.007769584655761719
5   98800.0 10 0.016277551651000977
6
```

Figura 4: Ejemplo de fichero de salida

B. Pseudocódigo de la solución propuesta

Se muestra, a continuación, el pseudocódigo correspondiente a la solución propuesta. En primer lugar, se detalla la función encargada de obtener los datos del fichero de entrada:

```
1  funcion ParsearFichero(F:fichero) devuelve (V:Variables)
2  variables V:vector de Variables, linea:cadena, u,v:vector de real,
       n:natural, page:Articulo, i:entero, valor:real,
       listaArticulos:vector de Articulos
3  principio
4      V := crearVacio(V)
5      mientasQue no finFichero(F) hacer
6          leerLinea(linea)
7          linea := leerLinea(linea)
8          v = [valor para valor en trocear(linea, " ")]
9          page := crearArticulo((0,0), v[2], v[1])
10         n := v[0]
11
12         listaArticulos := crearVacio(listaArticulos)
13
14         para i := 0 hasta n-1 hacer
15             linea := leerLinea(linea)
16             linea := quitarUltimoCaracter(linea)
17             u := [valor para valor in trocear(linea, " ")]
18             anadir(listaArticulos, crearArticle((u[2],u[3]),u[1],u[0]))
19         fpara
20
21         anadir(V,Variables(listaArticulos,n,page))
22         leerLinea(linea)
23     fmq
24     devuelve V
25 fin
```

Las siguientes funciones contienen la esencia del algoritmo de búsqueda con retroceso diseñado para solucionar el problema planteado en esta practica.

```
1  funcion Backtracking(V:Variables,IgnorarCota:booleano) devuelve (real,
       natural)
2  variables mejorSolucion,solucionInicial:Solution, nodos:natural
3  principio
4      mejorSolucion := crearVacia(mejorSolucion)
5      solucionInicial := crearVacia(solucionInicial)
6      nodos := 0
7      Backtracking_R(V,solucionInicial,IgnorarCota,V.AreaPagina(),
           mejorSolucion, nodos)
8      devuelve (mejorSolucion.totalArea, nodos)
9  fin
```

```
funcion Backtracking_R(V:Variables,Sol:Solution,IgnorarCota:booleano,
cota:real,M:Solution,nodos:natural) devuelve (real, natural)
variables nuevaSol:Solution, i:natural
principio
    para todo i en (Sol.siguiente, V.n) hacer
        nodos := nodos + 1
        si Cabe(i,Sol) entonces
            nuevaSol := CrearSolucion(Sol.indices+[i],
                        Sol.totalArea+V.areaArticulo(i))
            si nuevaSol > Sol entonces
                M := nuevaSol
            fsi
            si i < V.n-1 entonces
                siguienteCota := V.cota(nuevaSol)
                si IgnorarCota OR cota > M.totalArea entonces
                    Backtracking_R(V,nuevaSol,IgnorarCota,siguienteCota)
                    si cota <= M.totalArea entonces
                        devuelve (M.totalArea, nodos)
                    fsi
                fsi
            fsi
        fsi
    fpara
fin
```

C. Pruebas

A fin de comprobar el correcto funcionamiento de nuestro código, se diseñó una batería de pruebas. Dichas pruebas están recogidas en un total de 6 ficheros de texto, cuya contenido se detalla a continuación:

1. **test0.txt**: contiene 4 bloques:
   a) Consta de un único artículo, cuya dimensión coincide con la de la propia página.
   b) Contiene dos artículos, ambos con la misma anchura (la mitad de la página) y altura (la de la página), colocados uno al lado del otro.
   c) Contiene 3 artículos. Dos de ellos tienen la misma anchura y altura (la mitad de la de la pagina en ambos casos) y se encuentran uno al lado del otro. El último tiene como altura la de la página y como anchura un valor algo inferior al de la página. Además, este último artículo se sobrepone a los dos anteriores.
   d) Contiene un total de 4 artículos. Cada uno de ellos se solapa con los demás. Uno de los artículos es el de máxima área y, por lo tanto, el que deberá ser elegido por el algoritmo.
2. **test1.txt**: Contiene 2 bloques:
   a) Consta de 5 artículos. Dos de ellos comparten, únicamente, el vértice inferior derecho. Uno de los restantes artículos tiene como origen, el centro de uno de los anteriores. De los dos restantes, uno comparte lado con el anterior, mientras que el último se sobrepone a los dos anteriores.
   b) Consta de 6 artículos. De los cuales, sólo uno de ellos no se sobrepone a los demás.
3. **test2.txt**: Conjunto de pruebas de volumen. Son un total de 4 bloques:
   a) Consta de 36 artículos, todos con el mismo área (100 mm) que recubren la página sin sobreponerse unos a otros.
   b) Consta de 6 artículos, todos ellos comparten el extremo superior izquierdo y, por lo tanto, se sobreponen los unos a los otros. Uno de ellos tiene las mismas dimensiones que la página.
   c) Consta de 5 artículos, todos ellos de las mismas dimensiones. Cada uno de los artículos tiene como extremo superior izquierdo el centro del artículo anterior, excepto el primero, que comparte su extremo superior izquierdo con la página.
   d) Consta de 12 artículos. Los artículos están agrupados por parejas. Cada pareja comparte su extremo superior izquierdo. En cada pareja, las dimensiones de uno son la permutación de las del otro.
4. **test3.txt**: Consta de 8 artículos dispuestos de forma que se generen muchos solapes. El objetivo de esta prueba es comprobar la corrección de nuestra solución a la hora de escoger con qué artículos se queda.
5. **test4.txt**: Consta de 5 artículos dispuestos de forma que se generen muchos solapes. El objetivo de esta prueba es comprobar la corrección de nuestra solución a la hora de escoger con qué artículos se queda.
6. **test5.txt**: La finalidad de este test es la obtención de tiempos de ejecución al escalar el problema para poder realizar un buen análisis temporal de nuestra solución.

D. SVG

Para hacer más fácil el análisis de los resultados obtenidos en las pruebas realizadas, así como comprobar el correcto funcionamiento de nuestra solución, especialmente el tratamiento de los objetos **Polygon**, creamos una carpeta, de nombre `/svg`, en la que, para cada test, almacenar una representación visual, en formato `.svg`, de la disposición de sus artículos. A modo de ejemplo, se proporciona la imagen `.svg` correspondiente al primer bloque de artículos del test **test2.txt**.

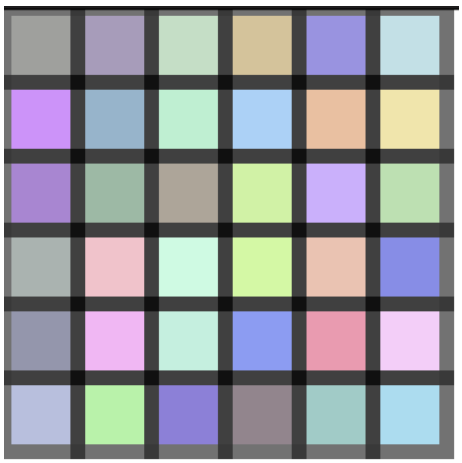

Figura 5: Imagen `.svg` correspondiente a uno de los tests en **test2.txt**