Memoria Práctica 3

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 tercera práctica de la asignatura *Algoritmia Básica*. Su objetivo es la implementación de dos nuevas soluciones al problema planteado en la anterior práctica. Dicho problema consiste en determinar 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. En particular, se pide una implementación iterativa en programación dinámica y una implementación recursiva que resuelva directamente la ecuación en recurrencias que proporciona la solución al problema. 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 esta ocasión, al tratarse de un problema con el que ya estábamos familiarizados, pasamos directamente a diseñar una primera aproximación al planteamiento **dinámico**. Tras varias horas de análisis y propuestas por parte de ambos miembros del equipo, así como de intercambio de ideas, dudas y sugerencias con otro grupo de trabajo y su posterior exposición al docente, llegamos a la conclusión de que la solución en programación dinámica consistía en una **heurística** y que, en determinadas ocasiones, no sería capaz de hallar la solución óptima e incluso, en los peores casos, ni siquiera hallaría soluciones **"buenas"**. Tras haber llegado a esta conclusión, planteamos las **ecuaciones en recurrencia** que rigen nuestra solución.

Una vez planteada nuestra primera aproximación a la solución del problema mediante programación dinámica, pasamos a plantear una solución **recursiva** que resolviera directamente las ecuaciones en recurrencias. Este paso fue sencillo, pues las ecuaciones en recurrencia de ambas soluciones son muy similares.

Por último, diseñamos una batería de pruebas para comprobar el correcto funcionamiento de nuestras soluciones, así como analizar las diferencias temporales en la ejecución de cada uno de los algoritmos. Estas pruebas pueden consultarse en B.

3. Diseño e implementación

Se exponen a continuación las decisiones de diseño adoptadas en la realización de cada uno de los algoritmos propuestos en nuestra solución. Como rasgos generales, cabe destacar que la implementación de cada una de las soluciones ha sido realizada con el lenguaje de programación **Python** a fin de poder reutilizar algunas de las **clases** definidas en la práctica anterior. Concretamente, las clases **Solution**, **Variables** y **Article**. De nuevo, cabe destacar el uso de la librería `shapely.geometry`.

3.1. Algoritmo con programación dinámica

Se detallan a continuación algunas nociones de notación y conceptos relacionados con el dominio del problema y su resolución, así como las ecuaciones en recurrencia que lo rigen:

- Sea $v^*[1, \ldots, n]$ al vector de artículos ordenados por áreas.
- Sea $A_i^* \subset \mathbb{R}^2$ la posición (como superficie total) que ocupa el i-ésimo artículo.

- Sea $a_i^* := \text{Area}(A_i^*)$ el área del i-ésimo artículo.

Nuestro objetivo es construir una matriz $M = (m_{ij}), 1 \leq i \leq n, 1 \leq j \leq N$, siendo N el área de la página en la que colocar los artículos, que contenga las soluciones óptimas de los subproblemas de tamaño creciente generados a partir del problema original. Más detalladamente, para cada componente de la matriz M, el índice i indica el número de artículos que estamos considerando en el subproblema actual, mientras que el índice j indica el área total de la que disponemos para colocar los i artículos. Por otro lado, cada componente de la matriz almacena el área total ocupada, que denotaremos por $a(i, j)$, y la disposición de los artículos en la página, que denotaremos por $P(i, j)$. Con tal de trabajar más cómodamente con la matriz anterior, se definió una clase, **Cell**, de Python. Finalmente, la matriz se rellena por filas, de manera ascendente, y cada fila es rellenada por columnas, también en sentido ascendente, siguiendo las siguientes ecuaciones en recurrencias:

$$a(i,a) = \begin{cases} a(i-1,a) & \text{si } a_i^* > a \\ \begin{cases} \text{máx}\left\{a\left(i-1,a-a_i^*\right), a_i^*, a(i,a-1)\right\} & \text{si } P(i-1,a-a_i^*) \cap A_i^* \neq \emptyset \\ a\left(i-1,a-a_i^*\right) + a_i^* & \text{si } P(i-1,a-a_i^*) \cap A_i^* = \emptyset \end{cases} & \text{si } a_i^* \leq a \end{cases} \tag{1}$$

$$P(i,a) = \begin{cases} P(i-1,a) & \text{si } a_i^* > a \\ \begin{cases} \text{máx}\left\{P\left(i-1,a-a_i^*\right), A_i^*, P(i,a-1)\right\} & \text{si } P\left(i-1,a-a_i^*\right) \cap A_i^* \neq \emptyset \\ P\left(i-1,a-a_i^*\right) \cup A_i^* & \text{si } P\left(i-1,a-a_i^*\right) \cap A_i^* = \emptyset \end{cases} & \text{si } a_i^* \leq a \end{cases} \tag{2}$$

Cabe destacar que abusamos de la notación máx$(\cdot)$ con superficies, refiriéndonos a la superficie de mayor área.

La solución del problema se obtiene de M_{nN}, i.e., considerando todos los artículos disponibles y el área total de la página.

Entonces el tiempo computacional de este algoritmo es aproximadamente de $O(n \times N)$, que es la dimensión de la tabla.

3.2. Algoritmo recursivo

Se detalla a continuación la implementación del algoritmo recursivo que resuelve el problema de los artículos resolviendo, directamente, las ecuaciones en recurrencia que se detallan más adelante. Como en el caso anterior, empezamos introduciendo algo de notación:

- Sea $\mathbf{P} \subset \mathbb{R}^2$ la página en la que debemos colocar los artículos.
- Sea $\mathbf{A} = \{A_0, \ldots, A_{n-1}\}$ el conjunto de artículos de los que disponemos cumpliendo que $A_i \subset \mathbf{P}$.
- Sea $\mathbf{X} = (x_1, \ldots, x_k)$, con $x_k \in \{0, \ldots, n-1\}\ \forall k$ y $x_k < x_{k+1}\ \forall k$, el conjunto de artículos que conforman una solución parcial del problema.
- Sea $\mathbf{U_X} = U(\{x_1, \ldots, x_k\}) := \bigcup_{i=1}^{k} A_{x_i}$ la superficie total empleada por los artículos que conforman una solución parcial del problema.
- Sea $\mathbf{g}(\mathbf{X}, j)$ con $0 \leq j \leq n$, la ecuación de recurrencias para el paso j con la restricción $x_k < j\ \forall k$.
- Definimos la función máximo de dos conjuntos de índices $\mathbf{X}, \mathbf{Y}$ como aquel cuya área total (área de $\mathbf{U_X}$ y $\mathbf{U_Y}$) sea mayor.

La ecuación en recurrencias que rige el problema y que debe resolver el algoritmo es:

$$\mathbf{g}(\mathbf{X}, j) = \begin{cases} \mathbf{X} & \text{si } j = n \\ \begin{cases} \mathbf{g}(\mathbf{X}, j+1) & \text{si } \mathbf{U}_{\mathbf{X}} \cap S_{x_j} \neq \emptyset \\ \max(\mathbf{g}(\mathbf{X} \cup \{j\}, j+1), \mathbf{g}(\mathbf{X}, j+1)) & \text{si } \mathbf{U}_{\mathbf{X}} \cap S_{x_j} = \emptyset \end{cases} \end{cases} \tag{3}$$

Para obtener la solución del problema, llamamos a la función $\mathbf{g}(\emptyset, 0)$.

Como en cada paso decidimos si meter o no el artículo i-ésimo, el coste computacional de este algoritmo es aproximadamente $O(2^n)$.

3.3. Algoritmo voraz

Como parte opcional de la práctica, se proponía hallar una heurística voraz que resolviera el problema. Empezamos definiendo:

- Sea $\mathbf{X} = \{x_1, \ldots, x_k\}$ una solución parcial del problema.
- Sea $\mathbf{R}_{\mathbf{X}} = \{r_1, \ldots, r_l\}$ los artículos que no se pueden meter ya que colisionan con la solución $\mathbf{X}$.
- Sea $\mathbf{C}_i = \{i_1, \cdots, i_s\}$ el conjunto de índices de los artículos que colisionan con el artículo i-ésimo. Además, definimos
$$\text{Area}(C_i) = \sum_{k=1}^{s} \text{Area}(A_{i_k})$$
la suma de las áreas de todos los artículos que colisionan con A_i.
- Sea $f(X, R_X)$ la función de recurrencias.
- Para este ejercicio requerimos que los artículos estén ordenados de mayor a menor, para obtener mejores resultados.

Entonces, la ecuación en recurrencias queda:

$$f(\mathbf{X}, \mathbf{R}_{\mathbf{X}}) = \begin{cases} \mathbf{X} & \text{si } \mathbf{R}_{\mathbf{X}} = \{1, \ldots, n\} \\ \begin{cases} f(\mathbf{X} \cup \{r_1\}, R_x \cup C_{r_1}) & \text{si Area}(A_{r_1}) > \text{Area}(C_{r_1}) \\ f(\mathbf{X}, R_x \cup \{r_1\}) & \text{si Area}(A_{r_1}) \leq \text{Area}(C_{r_1}) \end{cases} & \text{si no} \end{cases}$$

La idea detrás es: en cada iteración, decidir sobre si meter o no un artículo en la **solución global**. Para ello, comparamos qué área es mayor, la del propio artículo, o la suma de las áreas de los artículos que vamos a dejar de meter al introducir en la solución dicho artículo (los que intersecan con él).

Entonces, el tiempo en el peor caso (solo teniendo en cuenta la recursividad, mas no las operaciones detrás) es $O(n)$.

4. Análisis temporal

Se han realizado tests con un número de artículos creciente para ver el comportamiento temporal de los distintos algoritmos implementados frente al problema. Además, se ha realizado una gráfica para comparar dichos comportamientos. En particular, se comparan los algoritmos **iterativo**, con programación dinámica, **recursivo**, y la versión implementada con **backtracking** y uso de poda de la práctica anterior. Dicha gráfica, junto con otras figuras, se detallan en C.

5. Resultados y conclusiones

En esta práctica se ha experimentado con la técnica algorítmica de **programación dinámica** para resolver un problema con un espacio de búsqueda de un tamaño considerable. Además, la realización de un algoritmo recursivo que resolviera las ecuaciones en recurrencia del mismo problema nos han permitido observar la considerable reducción en coste temporal obtenida al aplicar un algoritmo frente a otro, así como su similar comportamiento en problemas de tamaño reducido.

Por otro lado, la realización de esta práctica ha puesto de manifiesto un principio que suele pasar por alto en el diseño de algoritmos para resolver problemas. A veces, según la técnica que se aplique, es imposible llegar a la solución óptima y los algoritmos se convierten en **heurísticas**, hecho que demuestra no sólo la importancia de escoger el patrón de diseño adecuado para cada problema sino la necesidad de hallar un **equilibrio** entre coste temporal y optimalidad de las soluciones encontradas.

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 1: 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 de los ficheros de salida

El fichero de salida contiene la información necesaria para diseñar la página. Consta de un número de líneas igual al número de bloques del fichero de entrada y tantas líneas adicionales como artículos que se incluyen en la página. Para cada página, la primera línea muestra el área total ocupada por los artículos elegidos y el tiempo de ejecución. A continuación, se muestra un ejemplo del formato de los ficheros de salida:

```
16.0 1.0762214660644531
 4 4 0 0
8.0 1.3277530670166016
 2 2 0 0
 2 2 2 2
```

Figura 2: Ejemplo de fichero de salida

B. 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 3 ficheros de texto, cuya contenido se detalla a continuación:

1. **test0.txt**: Consta de 4 bloques:
   a) 16 artículos repartidos en 4 filas. En cada fila, los artículos se hayan adyacentes unos a otros. Todos tienen el mismo tamaño y recubren la página en su totalidad.
   b) 4 artículos, todos ellos comparten esquina superior izquierda, que coincide con el límite superior izquierdo de la página en la que hay que insertarlos. El área de cada artículo es superior al del anterior en una unidad.
   c) 3 artículos. La esquina inferior derecha de cada uno de ellos coincide con la esquina superior izquierda del anterior. Todos ellos tienen el mismo tamaño.
   d) 8 artículos, agrupados en 4 parejas. Cada pareja de artículos comparte esquina superior izquierda y sus dimensiones, anchura y largura, han sido permutadas.
2. **test1.txt**: Contiene un total de 19 bloques. Cada uno de ellos ha sido diseñado, individualmente, para comprobar el correcto funcionamiento de los algoritmos en situaciones muy particulares. Por ejemplo, el primer bloque no contiene ningún artículo.
3. **test2.txt**: Contiene un único bloque de 8 artículos dispuestos de modo que haya muchas intersecciones entre ellos. Este test ha sido diseñado para comprobar que, bajo estas circunstancias, los algoritmos eligen los artículos que proporcionan una mejor solución.

Tanto el programa dinámico como la solución voraz son soluciones heurísticas. Esto provoca que los algoritmos no alcancen en muchos casos las soluciones óptimas globales y obtengan una solución parcial. Sin embargo, las soluciones obtenidas con los algoritmos expuestos no difieren mucho de la solución óptima. A continuación se exponen las diferentes soluciones.

		Solución Óptima Global	*Heurística*	*Heurística*
		Algoritmo Recursivo	**Algoritmo dinámico**	**Algoritmo Voraz**
Test 0	Variable 1	16	16	16
	Variable 2	16	16	16
	Variable 3	8	8	4
	Variable 4	10	9	10
Test 2	Variable 1	17	15	16
	Desviación	0.0%	4.5%	7.5%

Figura 3: Áreas obtenidas en las pruebas 0 y 2 en cada algoritmo.

C. Gráficas obtenidas en el análisis temporal

Se muestra a continuación la gráfica obtenida en la comparación de las distintas implementaciones del algoritmo al ser ejecutadas con tests de tamaño creciente en cuanto al número de artículos. El eje vertical muestra el tiempo de ejecución (en **ms**), mientras que el eje horizontal muestra el número de artículos del fichero de entrada.

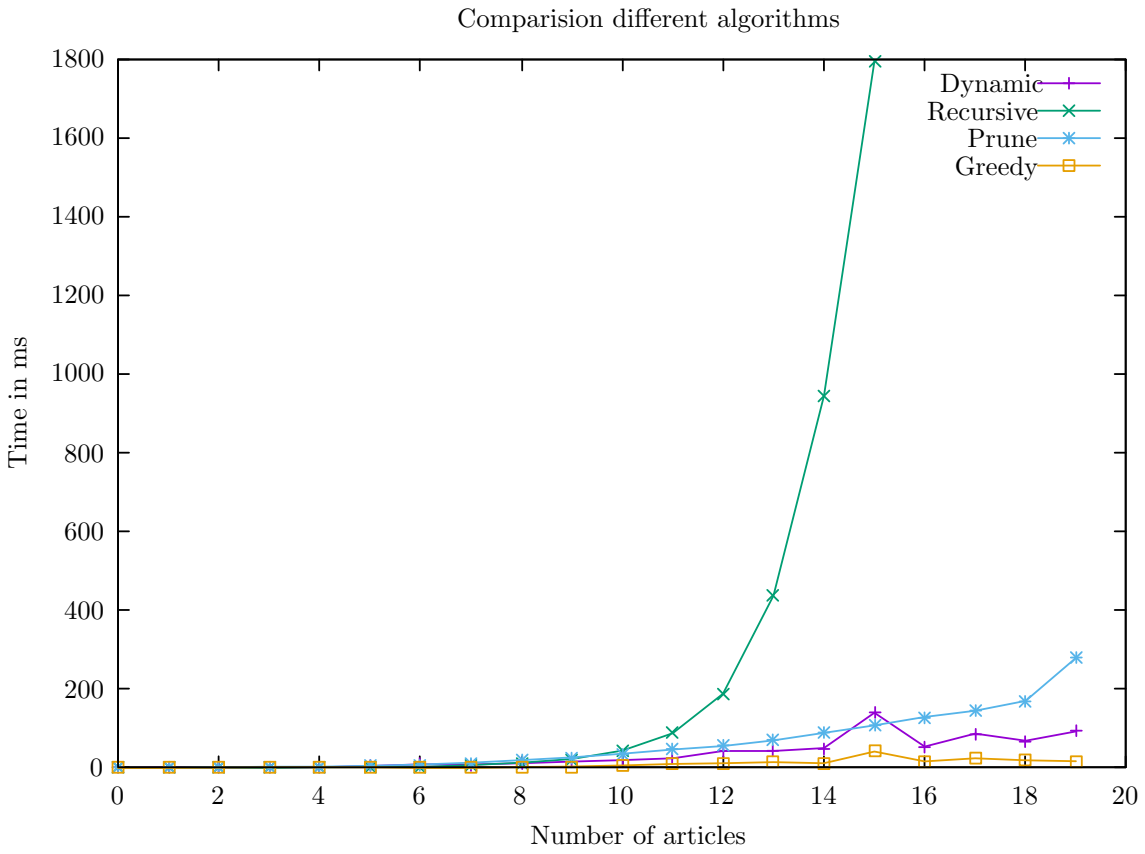

Figura 4: Comparativa temporal de los distintos algoritmos

Las siguientes figuras muestran la disposición de los artículos que constituyen el segundo test de la batería de pruebas (izquierda), así como la matriz obtenida al aplicar el algoritmo iterativo de programación dinámica (derecha).

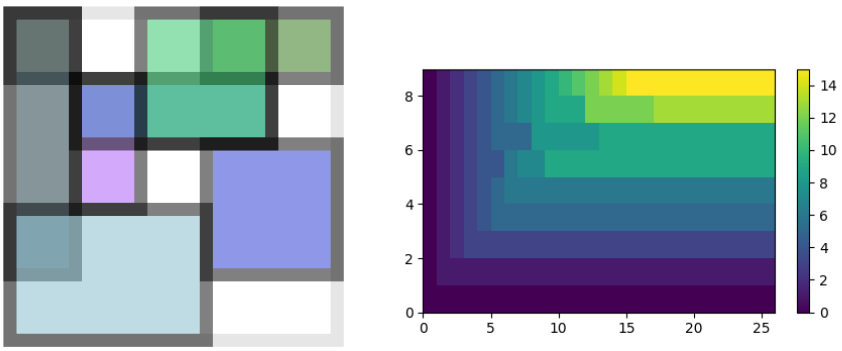

Figura 5: Disposición de artículos y matriz la resolución mediante programación dinámica