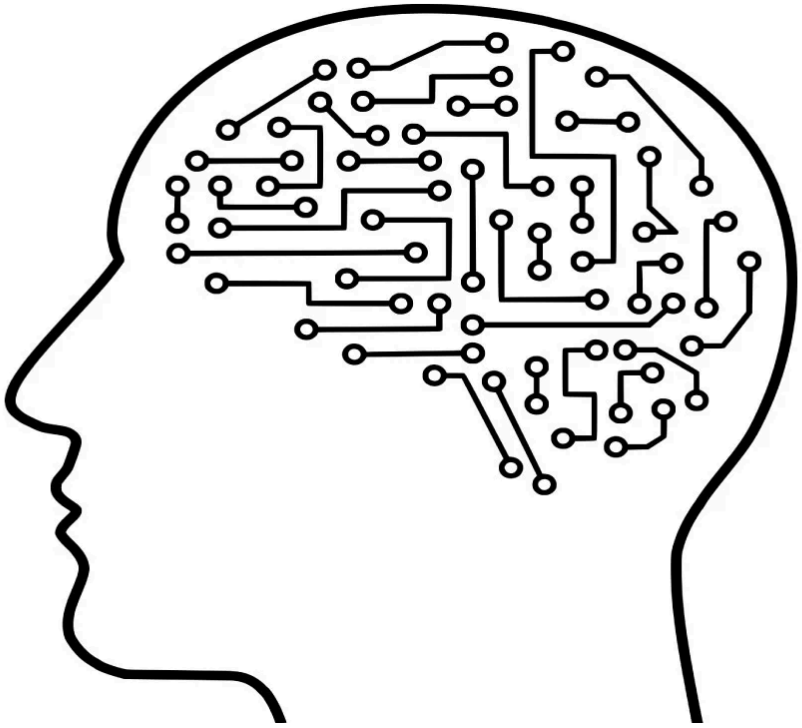
ALGORITMIA BÁSICA



PRÁCTICA 3

Jesús López Ansón - 839922
Javier Sin Pelayo - 843442

Análisis de los resultados

Como bien se indica en *LEEME.md*, en el script *ejecutar.sh*, para las pruebas, hemos hecho que por defecto aplique el algoritmo a 4 ficheros de interés:

- *1_prueba.txt:* es un fichero cuyo contenido es idéntico al del ejemplo presentado en el guión.
- 2_singleArticle.txt: contiene un bloque con un único artículo.
- 3_moreArticles.txt: es una versión del fichero inicial *1_prueba.txt,* pero al cual se le han añadido algunos artículos a cada uno de los dos bloques.
- 4_tricky.txt: caso de prueba con trampas para el algoritmo.

A continuación se muestra un ejemplo de la salida en pantalla de *ejecutar.sh:*

```
########## Ejecutando pruebas con 1_prueba.txt ##########
        Recursiva
400 0.106900
20 10 25 15
10 10 0 0
10 10 15 15
8000 0.106800
90 80 70 60
20 30 40 50
10 20 30 40
        Iterativa
400 0.307000
10 10 15 15
10 10 0 0
20 10 25 15
8000 0.555000
10 20 30 40
20 30 40 50
90 80 70 60
        Voraz
400 0.058500
20 10 20 20
10 10 10 10
10 10 0 0
8000 0.050300
90 80 70 60
20 30 40 50
10 20 30 40
########## Ejecutando pruebas con 2_singleArticle.txt ##########
        Recursiva
100 0.023800
10 10 15 15
        Iterativa
100 0.043300
10 10 15 15
        Voraz
100 0.018000
10 10 15 15
########## Ejecutando pruebas con 3_moreArticles.txt ##########
        Recursiva
425 0.280000
20 10 25 15
10 10 0 0
10 10 15 15
5 5 15 25
8300 0.581800
90 80 70 60
20 30 40 50
10 20 30 40
20 10 20 20
```

```
10 10 10 10
        Iterativa
425 1.716100
5 5 15 25
10 10 15 15
10 10 0 0
20 10 25 15
8300 8.359900
10 10 10 10
20 10 20 20
10 20 30 40
20 30 40 50
90 80 70 60
        Voraz
425 0.091200
20 10 20 20
10 10 10 10
10 10 0 0
5 5 15 25
8300 0.106200
90 80 70 60
20 30 40 50
20 10 20 20
10 20 30 40
10 10 10 10
########## Ejecutando pruebas con 4_tricky.txt ##########
        Recursiva
128 0.071100
8 8 0 0
8 8 8 8
128 0.037600
8 8 0 0
8 8 8 8
        Iterativa
128 0.078700
8 8 8 8
8 8 0 0
128 0.055600
8 8 8 8
8 8 0 0
        Voraz
100 0.059100
10 10 0 0
100 0.032400
10 10 7 7
```

Podemos concluir que los algoritmos tienen unas complejidades:

- **recursivo**: La complejidad temporal de este algoritmo es O (2 ^ n), donde n es el número de artículos. Esto se debe a que, en el peor de los casos, el algoritmo explora todos los subconjuntos posibles de artículos, que es 2^n. Sin embargo, la complejidad del tiempo real puede ser menor si el algoritmo puede podar algunas ramas del árbol de recursividad debido a superposiciones.
- **iterativo**: La complejidad temporal de este algoritmo es O (2 ^ n * n), donde n es el número de artículos. Esto se debe a que el algoritmo itera sobre los 2^n subconjuntos posibles de artículos y, para cada subconjunto, es posible que necesite iterar sobre los n artículos para verificar si un artículo está en el subconjunto y si se superpone con cualquier otro artículo en el subconjunto.
- **voraz**: La complejidad temporal de este algoritmo es O (n ^ 2), donde n es el número de artículos. Esto se debe a que el algoritmo primero calcula el número de superposiciones para cada artículo, lo que requiere iterar sobre todos los pares de artículos, y luego itera sobre los artículos ordenados para agregarlos a la solución si no se superponen con ningún otro artículo en el solución.

Por norma general, el algoritmo iterativo es más eficiente que el recursivo. Sin embargo, en algún caso especial como en el del viajero (visto en clases de teoría) o en este, se obtienen valores peores para el algoritmo iterativo ya que resulta más costoso el cálculo de la tabla de memoización, que la búsqueda en recursivo basado en las ecuaciones de recurrencia. En estos casos, lo correcto es calcular solamente la parte de interés de la tabla, la que se va a utilizar, y no su totalidad.

Aún considerando esta puntualización, se han obtenido valores más eficaces en recurrencia que en iteratividad.

Exposición de los algoritmos

Recursivo

Está basado en la ecuación en recurrencias.

1. Ordena los artículos por área (ancho * alto) en orden descendente.
2. Llama a función recursiva con el número de artículos y la solución parcial vacía
3. Llamada recursiva:
   3.1. Si llega al caso base (se queda sin artículos), devuelve una solución vacía.
   3.2. Si no, compara la solución de excluir el artículo actual del bloque, o incluirlo en la solución quedándose con la solución más óptima. Si el artículo se superpone, se queda con la solución excluyente directamente.

Iterativo

Está basado en la tabla *memoization*.

1. Ordena los artículos por área (ancho * alto) en orden descendente.
2. Itera sobre todos los posibles subconjuntos de artículos.
3. En cada iteración realiza lo siguiente:
   a. Si el artículo no se encuentra en el subconjunto que se está verificando, se pasa a la siguiente iteración.
   b. Si el artículo se superpone al resto, se continúa a la siguiente iteración.
   c. Si no ocurre ninguna de estas condiciones previas, se calcula la solución añadiendo el artículo, y se guarda en la tabla el máximo entre esta y la solución previa.

Voraz

1. Una heurística voraz simple e inteligente es ordenar los artículos por área dividida por el número de superposiciones con otros artículos.
2. Tras realizar esta ordenación, se itera en los artículos y se van añadiendo si no se superponen con los que ya se encuentran en el resto de la solución.

Además, en los tres algoritmos se han hecho dos suposiciones:

- Hay al menos un artículo en cada bloque.
- Cada artículo por individual cabe en su página.