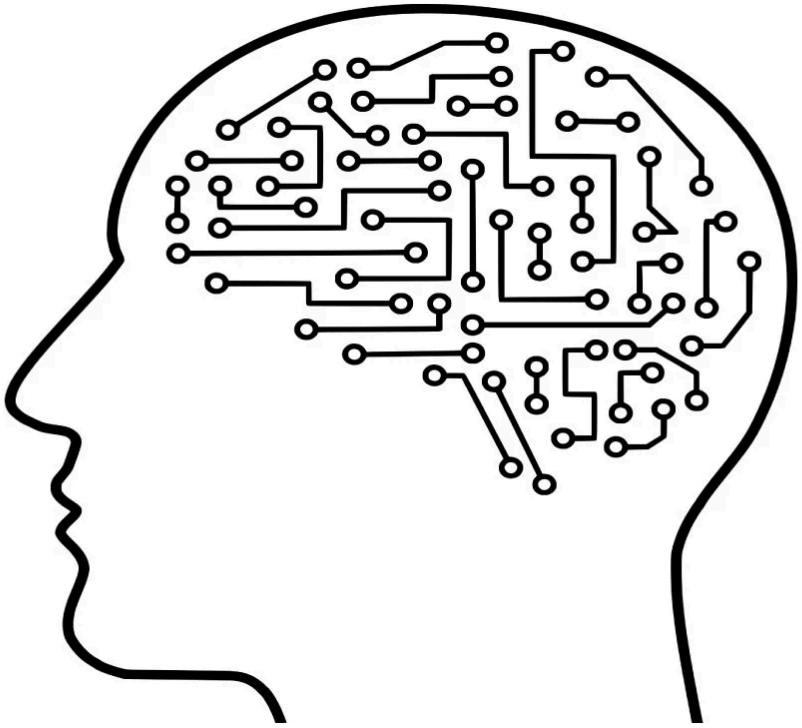
# ALGORITMIA BÁSICA



# PRÁCTICA 2

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 3 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 su vez, ejecuta un banco de pruebas auxiliares con otros casos de estudio de interés, comparando 3 problemas distintos resueltos con **backtracking** frente a su resolución mediante **fuerza bruta**. Se comparan tanto los tiempos de ejecución de ambos algoritmos, como los nodos generados por estos, para cada uno de los tests realizados.

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

```
$> . ejecutar.sh

Pruebas con los archivos de prueba
Probando con 1_prueba.txt
400 0.534800
20 10 20 20
10 10 0 0
10 10 10 10
8000 0.822600
90 80 70 60
20 30 40 50
10 20 30 40
Probando con 2_singleArticle.txt
100 0.038500
10 10 0 0
Probando con 3_moreArticles.txt
425 1.950200
20 10 20 20
10 10 0 0
10 10 10 10
5 5 15 25
8300 13.440300
90 80 70 60
20 30 40 50
10 20 30 40
20 10 20 20
10 10 10 10
Probando con 4_tricky.txt
128 0.128000
8 8 0 0
8 8 8 8
128 0.087000
8 8 0 0
8 8 8 8

Tests:

Block: n: 9, W: 200, H: 200, articles:
Article with values--> w: 10, h: 10, x: 45, y: 45
Article with values--> w: 10, h: 10, x: 15, y: 15
Article with values--> w: 10, h: 10, x: 75, y: 75
Article with values--> w: 10, h: 10, x: 30, y: 30
Article with values--> w: 10, h: 10, x: 30, y: 30
```

```
Article with values--> w: 10, h: 10, x: 100, y: 100
Article with values--> w: 10, h: 10, x: 5, y: 5
Article with values--> w: 10, h: 10, x: 0, y: 0
Article with values--> w: 10, h: 10, x: 60, y: 60

Backtracking: 58.616700ms
Brute force:  13.794900ms
Nodes generated in backtracking: 2503
Nodes generated in brute force: 511

Block: n: 5, W: 280, H: 400, articles:
Article with values--> w: 20, h: 10, x: 25, y: 15
Article with values--> w: 20, h: 10, x: 20, y: 20
Article with values--> w: 10, h: 10, x: 0, y: 0
Article with values--> w: 10, h: 10, x: 10, y: 10
Article with values--> w: 10, h: 10, x: 15, y: 15

Backtracking:  0.379800ms
Brute force:   0.201300ms
Nodes generated in backtracking: 25
Nodes generated in brute force: 31

Block: n: 6, W: 280, H: 400, articles:
Article with values--> w: 90, h: 80, x: 70, y: 60
Article with values--> w: 80, h: 70, x: 60, y: 50
Article with values--> w: 50, h: 60, x: 70, y: 80
Article with values--> w: 50, h: 50, x: 40, y: 40
Article with values--> w: 20, h: 30, x: 40, y: 50
Article with values--> w: 10, h: 20, x: 30, y: 40

Backtracking:  0.738200ms
Brute force:   0.270600ms
Nodes generated in backtracking: 33
Nodes generated in brute force: 63

Block: n: 7, W: 280, H: 400, articles:
Article with values--> w: 20, h: 10, x: 25, y: 15
Article with values--> w: 20, h: 10, x: 20, y: 20
Article with values--> w: 10, h: 10, x: 0, y: 0
Article with values--> w: 10, h: 10, x: 10, y: 10
Article with values--> w: 10, h: 10, x: 15, y: 15
Article with values--> w: 7, h: 7, x: 12, y: 12
Article with values--> w: 5, h: 5, x: 15, y: 25

Backtracking:  2.873100ms
Brute force:   1.521700ms
Nodes generated in backtracking: 115
Nodes generated in brute force: 127

Block: n: 9, W: 280, H: 400, articles:
Article with values--> w: 90, h: 80, x: 70, y: 60
Article with values--> w: 80, h: 70, x: 60, y: 50
Article with values--> w: 50, h: 60, x: 70, y: 80
Article with values--> w: 50, h: 50, x: 40, y: 40
Article with values--> w: 20, h: 30, x: 40, y: 50
Article with values--> w: 20, h: 10, x: 20, y: 20
Article with values--> w: 10, h: 20, x: 30, y: 40
Article with values--> w: 10, h: 10, x: 15, y: 15
Article with values--> w: 10, h: 10, x: 10, y: 10

Backtracking: 29.742600ms
Brute force:   9.965400ms
Nodes generated in backtracking: 581
Nodes generated in brute force: 511

Block: n: 3, W: 100, H: 100, articles:
Article with values--> w: 10, h: 10, x: 0, y: 0
```

```
Article with values--> w: 8, h: 8, x: 8, y: 8
Article with values--> w: 8, h: 8, x: 0, y: 0

Backtracking:  0.134100ms
Brute force:   0.081200ms
Nodes generated in backtracking: 6
Nodes generated in brute force: 7

Block: n: 3, W: 100, H: 100, articles:
Article with values--> w: 10, h: 10, x: 7, y: 7
Article with values--> w: 8, h: 8, x: 0, y: 0
Article with values--> w: 8, h: 8, x: 8, y: 8

Backtracking:  0.466200ms
Brute force:   0.087600ms
Nodes generated in backtracking: 6
Nodes generated in brute force: 7
.
----------------------------------------------------------------------
Ran 2 tests in 0.140s

OK
```

De esta manera, podemos concluir que el algoritmo diseñado tiene una complejidad en el caso peor de: **O(p(n)n!)**. Esto se puede ver fácilmente en el siguiente ejemplo utilizado en los tests:

```
articles = [Article(10, 10, i * 15, i * 15) for i in range(6)]
articles.append(Article(10, 10, 5, 5))
articles.append(Article(10, 10, 30, 30))
articles.append(Article(10, 10, 100, 100))
block = Block(len(articles), 200, 200, articles)
```

Este bloque de código genera un bloque donde hay muy poco overlap por lo tanto el predicado acotador no ejerce su función de poda y de este modo se generan una gran cantidad de nodos.

Se aprecia, cómo dependiendo del caso, funciona mejor la fuerza bruta, ya que si hay mucho overlap entre artículos backtracking funciona mejor porque el predicado acotador poda más ramas del árbol de búsqueda.

# Exposición del algoritmo

- Ordena los artículos por área (ancho * alto) en orden descendente.
- Función recursiva
  Si ya se han explorado todos los artículos (profundidad máxima), se devuelve

1. Para cada nodo hijo:
    1.1. Si el artículo no se superpone con su artículos ancestros del árbol, entonces:
        1.1.1. Se llama recursivamente con el nodo hijo actual.
        1.1.2. Si la nueva solución es mejor que la solución parcial hasta el momento:
            1.1.2.1. Actualiza la mejor solución a esta nueva.
2. Devuelve la mejor solución encontrada entre sus nodos hijos

Además, se han hecho dos suposiciones:

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