# CC184 - Complejidad Algorítmica

| | |
|---|---|
| Tema: | Algoritmos de Fuerza Bruta(Brute Force) y Backtracking |
| Formato: | Esquema de Aprendizaje |
| Elaborado por: | Robert Zubieta |
| Fuente: | Propia |

# ALGORITMOS DE FUERZA BRUTA Y BACKTRACKING

## I. Fuerza Bruta (Brute-Force): Alcance

**CARACTERÍSTICAS**

- Probar todas las posibles soluciones => Obtiene la mejor solucion, pero puede tomar demasiado tiempo.
- Busca sistemáticamente en todo el espacio de búsqueda (no sofisticación).
- Se toma la ruta más directa, sin ningún intento de minimizar el número de operaciones necesarias para calcular la solución
- Prueba todas las posibles soluciones (dentro de ellas, está la más optima).
- Los algoritmos son lentos, pero fáciles de implementar.

**VENTAJAS**

- Siempre se encuentra una solucion.
- Facil de implementar

**DESVENTAJA**

- Puede tomar demasiado tiempo.

**ESTRATEGIA DE USO**

- Combinaciones con o sin repetición: Si el orden no importa, es una combinación.
- Permutaciones con o sin repetición: Si el orden sí importa es una permutación.

Una permutación es una combinación ordenada.

## II. Fuerza Bruta (Brute-Force): Ejemplo

**Encontrar el DIVISOR de un Número Natural**

**Divisores de 4**

| | | | | |
|---|---|---|---|---|
| 4 | / | 1 | = | 4 |
| 4 | / | 2 | = | 2 |
| 4 | / | 3 | = | ~~1.33~~ |
| 4 | / | 4 | = | 1 |

**Divisores de 99**

| | | | | |
|---|---|---|---|---|
| 99 | / | 1 | = | 99 |
| 99 | / | 2 | = | ~~49.5~~ |
| 99 | / | 3 | = | 33 |
| 99 | / | 4 | = | ~~24.8~~ |
| | ... | | = | ... |
| | ... | | = | ... |
| 99 | / | 99 | = | 1 |

**Divisores de 999**

| | | | | |
|---|---|---|---|---|
| 999 | / | 1 | = | 999 |
| 999 | / | 2 | | ~~500~~ |
| 999 | / | 3 | = | 333 |
| 999 | / | 4 | | ~~250~~ |
| 999 | / | 5 | = | ~~200~~ |
| 999 | / | 6 | | ~~167~~ |
| 999 | / | 7 | = | ~~143~~ |
| | ... | | | ... |
| | ... | | | ... |
| 999 | / | 999 | | 1 |

n=1 => Complejidad: $O(10^1)$ n=2 => Complejidad: $O(10^2)$ n=3 => Complejidad: $O(10^3)$

n: Cantidad de dígitos del Número Complejidad: $O(10^n)$

## III. Fuerza Bruta (Brute-Force): String Matching

**PROBLEMA: Buscar en la cadena el patron "NOT"**

| N | O | B | O | D | Y | - | N | O | T | I | C | E | D | - | H | I | M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | |
| N | O | T | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | |
| | N | O | T | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | |
| | | N | O | T | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | |
| | | | N | O | T | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | |
| | | | | N | O | T | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | |
| | | | | | N | O | T | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | |
| | | | | | | N | O | T | | | | | | | | | |
| | | | | | | | | | | | | | | | | | |
| | | | | | | | N | O | T | | | | | | | | |
| | | | | | | | | | | | | | | | | | |

**DESAFÍO**:
Buscar el patrón "SHE" en la cadena:
SHE SELLS SEASHELLS ON THE SEASHORE

# String Matching

**ALGORITHM** *BruteForceStringMatch*$(T[0..n-1], P[0..m-1])$

```
//Implements brute-force string matching
//Input: An array T[0..n − 1] of n characters representing a text and
//       an array P[0..m − 1] of m characters representing a pattern
//Output: The index of the first character in the text that starts a
//        matching substring or −1 if the search is unsuccessful
for i ← 0 to n − m do
    j ← 0
    while j < m and P[j] = T[i + j] do
        j ← j + 1
    if j = m return i
return −1
```

## IV. Problema de las 8 Reinas

**Problema de 8 Reinas**

**En que consiste ?**

Debemos poner todas las reinas de tal forma que ninguna de ellas pueda atacar a otra

**Como lo solucionamos?**

Probar cada uno de los casos hasta verificar que se cumpla

Iteraciones $\binom{64}{8} = \frac{64!}{8!}$

## V. BackTracking: Alcance

**CARACTERISTICAS**

Backtracking es Fuerza Bruta pero más eficiente
Usa proceso de PODA (no termina recorrido)
Toma decisiones sistemáticas: basado en Reglas o Restricciones del problema
Reglas Explicitas | Reglas Implicitas
Si no cumple la Reglas => Regresa atrás (BACKTRAKING)
Un Algoritmo de Backtraing puede ser representado por un ARBOL de Busqueda

**CUANDO APLICAR (sugerencias)**

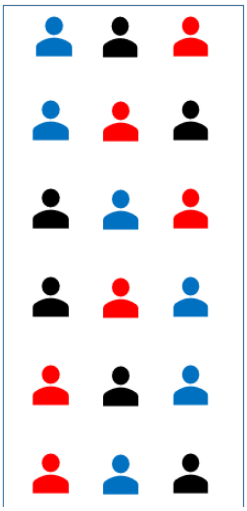
- Cuando hay multiples caminos. Ejm: Problema del Laberinto: Mover hacia derecha, hacia izquierda, hacia arriba, hacia abajo.
- Existen multiples soluciones. Ejm: 8 Reinas, tiene 92 soluciones.
- Cuando no se usen en problemas de Optimización.
- Cuando no hay suficiente información. Ejm: Problema del Laberinto.
- Cuando cada decisión afecta a la siguiente. Ejm: Si ingresamos la univ. => Escogemos CC. Si escogemos CC => Llevamos 20 creditos, etc.
- Cuando hay Reglas y/o Restricciones.

**3 Keys of BACKTRACKING**
- Choices
- Constraints
- Goal

## VI. Brute-Force vs BackTracking: Caso Aplicado sin Restricciones

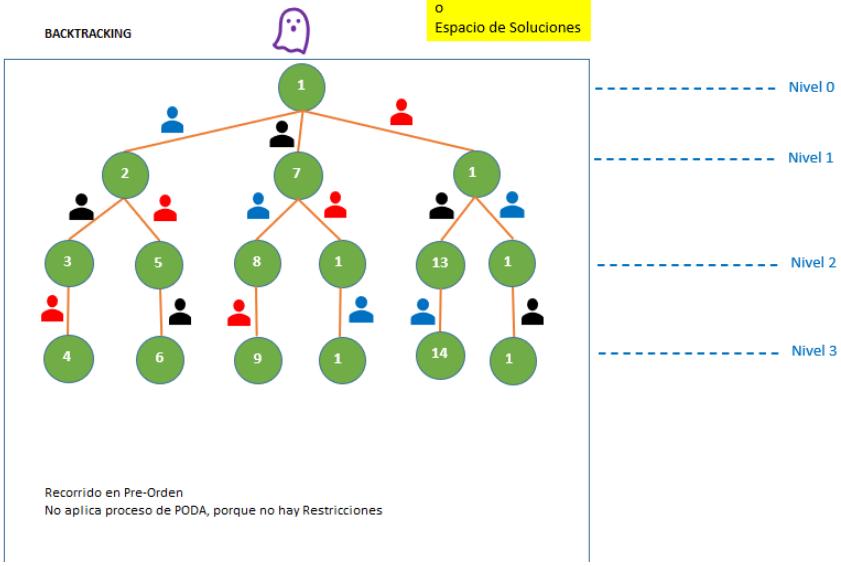
**PROBLEMA: De cuantas formas se pueden sentar 3 personas en 3 asientos ?**

FUERZA BRUTA



BACKTRACKING

Arbol de Soluciones
o
Espacio de Soluciones

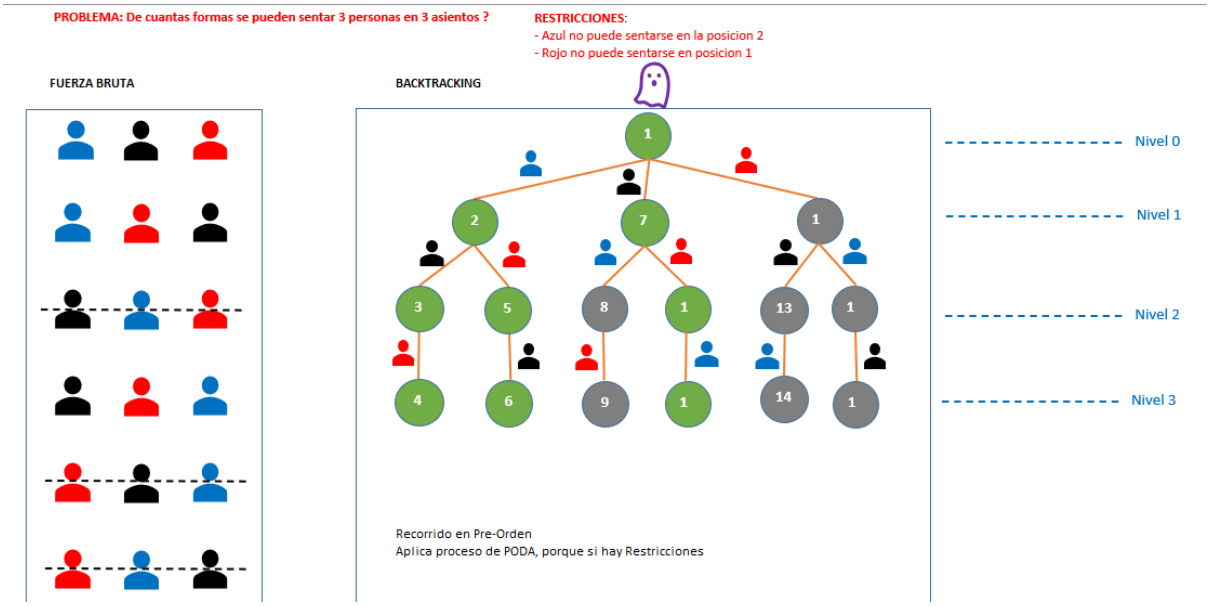


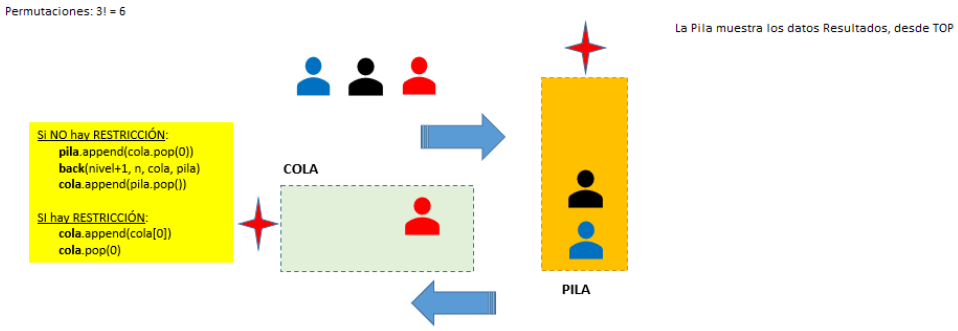

Recorrido en Pre-Orden
No aplica proceso de PODA, porque no hay Restricciones

Permutaciones: 3! = 6

```
import itertools

def main():
  lista = ["a", "n", "r"]

  permutations = list(itertools.permutations(lista))
  print(permutations)

main()
```

## VII. Brute-Force vs BackTracking: Caso Aplicado con Restricciones



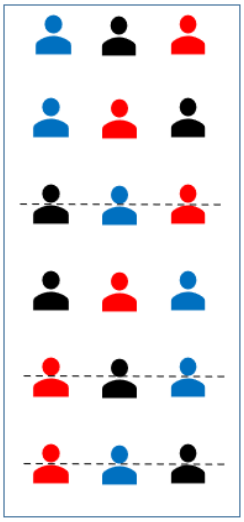



## VIII. Brute-Force vs BackTracking: Caso Aplicado con Restricciones. Escenario Final

PROBLEMA: De cuantas formas se pueden sentar 3 personas en 3 asientos ?

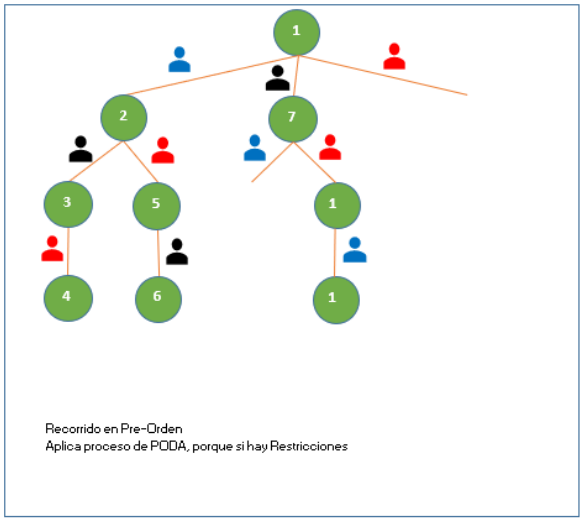
RESTRICCIONES:
- Azul no puede sentarse en la posicion 2
- Rojo no puede sentarse en posicion 1

FUERZA BRUTA



Permutaciones: 3! = 6

BACKTRACKING




Recorrido en Pre-Orden
Aplica proceso de PODA, porque si hay Restricciones