Pseudocódigo: Problema de Selección de Actividades con Fuerza Bruta

Algoritmo de Fuerza Bruta

24 de septiembre de 2025

1. Descripción del Problema

El problema de selección de actividades consiste en seleccionar el máximo número de actividades mutuamente compatibles de un conjunto dado. Cada actividad tiene un tiempo de inicio y un tiempo de finalización. Dos actividades son compatibles si no se solapan en el tiempo (la finalización de una es menor o igual al inicio de la otra). El objetivo es encontrar el subconjunto de máximo tamaño que no contenga actividades que se solapen.

2. Pseudocódigo

Algorithm 1 Selección de Actividades con Fuerza Bruta

Require: Un arreglo *activities* de n elementos donde cada elemento es una tupla $(start, end)$
Ensure: El subconjunto de máximo tamaño de actividades mutuamente compatibles

```
n ← longitud de activities
best ← lista vacía
{Probar todos los subconjuntos posibles}
for r = 1 to n do
    for cada subset en combinations(activities, r) do
        if is_valid(subset) ∧ |subset| > |best| then
            best ← subset
        end if
    end for
end for
return best
```

3. Explicación del Algoritmo

3.1. Enfoque de Fuerza Bruta

El algoritmo de fuerza bruta para el problema de selección de actividades funciona de la siguiente manera:

- **Generación de subconjuntos:** Se generan todos los posibles subconjuntos de actividades usando la función *combinations*
- **Validación:** Para cada subconjunto generado, se verifica si las actividades son mutuamente compatibles
- **Optimización:** Se mantiene el mejor subconjunto encontrado hasta el momento

La validación de compatibilidad se basa en que dos actividades $(start_i, end_i)$ y $(start_j, end_j)$ son compatibles si:

$$end_i \leq start_j \text{ o } end_j \leq start_i$$

3.2. Complejidad

- **Tiempo:** $O(2^n \cdot n^2)$
  - $O(2^n)$ para generar todos los subconjuntos posibles
  - $O(n^2)$ para validar cada subconjunto (verificar solapamientos)
- **Espacio:** $O(n)$ para almacenar el mejor subconjunto encontrado

4. Resolución Paso a Paso

4.1. Ejemplo: Actividades = [(1, 3), (2, 5), (4, 6)]

Paso 1: Identificar todas las actividades

- Actividad A: (1, 3) - inicia en 1, termina en 3
- Actividad B: (2, 5) - inicia en 2, termina en 5
- Actividad C: (4, 6) - inicia en 4, termina en 6

Paso 2: Generar todos los subconjuntos de tamaño 1

- {A} = {(1, 3)} - Válido (1 actividad)
- {B} = {(2, 5)} - Válido (1 actividad)
- {C} = {(4, 6)} - Válido (1 actividad)

Mejor hasta ahora: cualquier subconjunto de tamaño 1

Paso 3: Generar todos los subconjuntos de tamaño 2

- {A, B} = {(1, 3), (2, 5)}
  - ¿A y B se solapan? A termina en 3, B inicia en 2
  - Como 3 ¿2, se solapan. NO es válido.
- {A, C} = {(1, 3), (4, 6)}
  - ¿A y C se solapan? A termina en 3, C inicia en 4
  - Como $3 \leq 4$, NO se solapan. ES válido.
- {B, C} = {(2, 5), (4, 6)}
  - ¿B y C se solapan? B termina en 5, C inicia en 4
  - Como 5 ¿4, se solapan. NO es válido.

Mejor hasta ahora: {A, C} = {(1, 3), (4, 6)} (tamaño 2)

Paso 4: Generar subconjunto de tamaño 3

- {A, B, C} = {(1, 3), (2, 5), (4, 6)}
  - Verificar solapamientos:
    - A y B: $3 > 2 \Rightarrow$ Se solapan
    - A y C: $3 \leq 4 \Rightarrow$ NO se solapan
    - B y C: $5 > 4 \Rightarrow$ Se solapan
  - Como hay al menos un solapamiento, NO es válido.

Resultado Final: El subconjunto óptimo es {A, C} = {(1, 3), (4, 6)} con 2 actividades.

4.2. Visualización Temporal

```
Tiempo: 0  1  2  3  4  5  6  7
A:      |-----|
B:         |--------|
C:            |-----|

Solución óptima: A y C (no se solapan)
```

5. Análisis de Complejidad

5.1. Comparación con Algoritmo Greedy

- **Fuerza Bruta:** $O(2^n \cdot n^2)$ - Encuentra la solución óptima pero es exponencial
- **Algoritmo Greedy:** $O(n \log n)$ - Solución óptima cuando se ordena por tiempo de finalización

El algoritmo de fuerza bruta es útil para:

- Verificar la corrección de algoritmos más eficientes
- Problemas pequeños donde la complejidad exponencial es manejable
- Casos donde se necesitan múltiples criterios de optimización

6. Conclusiones

El algoritmo de fuerza bruta para el problema de selección de actividades garantiza encontrar la solución óptima pero tiene una complejidad exponencial. Es apropiado para casos de estudio y verificación, pero para problemas reales se recomienda usar el algoritmo greedy que tiene complejidad $O(n \log n)$ y produce resultados óptimos cuando se ordena por tiempo de finalización.