Pseudocódigo: Problema de Selección de Actividades con Algoritmo Greedy

Algoritmo Greedy

25 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.

El algoritmo greedy resuelve este problema de manera óptima ordenando las actividades por su tiempo de finalización y seleccionando de manera greedy.

2. Pseudocódigo

Algorithm 1 Selección de Actividades con Algoritmo Greedy

Require: Arreglos $start$ y $finish$ de n elementos con tiempos de inicio y finalización
Ensure: El número máximo de actividades mutuamente compatibles

```
activities ← crear lista de tuplas (start[i], finish[i]) para i = 0 a n − 1
Ordenar activities por tiempo de finalización (segundo elemento de cada tupla)
count ← 1 {Al menos una actividad puede realizarse}
j ← 0 {Índice de la última actividad seleccionada}
for i = 1 to n − 1 do
    if activities[i].start > activities[j].finish then
        count ← count + 1
        j ← i {Actualizar última actividad seleccionada}
    end if
end for
return count
```

3. Explicación del Algoritmo

3.1. Estrategia Greedy

El algoritmo greedy para el problema de selección de actividades utiliza la siguiente estrategia:

- **Ordenamiento:** Ordena todas las actividades por su tiempo de finalización en orden creciente
- **Selección Greedy:** Siempre selecciona la primera actividad que no se solape con la última actividad seleccionada

- **Justificación:** Al seleccionar la actividad que termina primero, se maximiza el tiempo disponible para actividades futuras

La elección greedy es óptima porque:

- Si existe una solución óptima que no incluye la actividad que termina primero, se puede reemplazar la primera actividad de esa solución por la que termina primero
- Esto no empeora la solución y mantiene la optimalidad

3.2. Subestructura Óptima

Para el problema de selección de actividades, la subestructura óptima se cumple:
Si S es una solución óptima para el conjunto de actividades A, y a_1 es la actividad que termina primero en A, entonces:

- a_1 debe estar en alguna solución óptima
- El resto de la solución óptima debe ser óptima para el conjunto A' de actividades que no se solapan con a_1

Esto se debe a que seleccionar a_1 no interfiere con las opciones futuras, sino que las maximiza.

3.3. Complejidad

- **Tiempo:** $O(n \log n)$
  - $O(n \log n)$ para ordenar las actividades por tiempo de finalización
  - $O(n)$ para recorrer las actividades y hacer las selecciones
- **Espacio:** $O(n)$ para almacenar las actividades ordenadas

4. Resolución Paso a Paso

4.1. Ejemplo

Actividades = [(1, 2), (3, 4), (0, 6), (5, 7), (8, 9), (5, 9)] Paso 1: Crear tuplas de actividades

- Actividad 0: (1, 2)
- Actividad 1: (3, 4)
- Actividad 2: (0, 6)
- Actividad 3: (5, 7)
- Actividad 4: (8, 9)
- Actividad 5: (5, 9)

Paso 2: Ordenar por tiempo de finalización

- Actividad 0: (1, 2) - termina en 2
- Actividad 1: (3, 4) - termina en 4
- Actividad 2: (0, 6) - termina en 6
- Actividad 3: (5, 7) - termina en 7
- Actividad 4: (8, 9) - termina en 9

- Actividad 5: (5, 9) - termina en 9

Paso 3: Aplicar algoritmo greedy
Inicialización:

- $count = 1$ (siempre se puede hacer al menos una actividad)
- $j = 0$ (primera actividad seleccionada: Actividad 0)

Iteraciones:

- **i = 1:** Actividad 1 (3, 4)
  - ¿$activities[1].start > activities[0].finish$?
  - ¿$3 > 2$? SÍ
  - Seleccionar Actividad 1
  - $count = 2$, $j = 1$
- **i = 2:** Actividad 2 (0, 6)
  - ¿$activities[2].start > activities[1].finish$?
  - ¿$0 > 4$? NO
  - NO seleccionar Actividad 2 (se solapa)
- **i = 3:** Actividad 3 (5, 7)
  - ¿$activities[3].start > activities[1].finish$?
  - ¿$5 > 4$? SÍ
  - Seleccionar Actividad 3
  - $count = 3$, $j = 3$
- **i = 4:** Actividad 4 (8, 9)
  - ¿$activities[4].start > activities[3].finish$?
  - ¿$8 > 7$? SÍ
  - Seleccionar Actividad 4
  - $count = 4$, $j = 4$
- **i = 5:** Actividad 5 (5, 9)
  - ¿$activities[5].start > activities[4].finish$?
  - ¿$5 > 9$? NO
  - NO seleccionar Actividad 5 (se solapa)

Resultado Final: Se pueden realizar 4 actividades: Actividad 0, Actividad 1, Actividad 3, y Actividad 4.

4.2. Visualización Temporal

```
Tiempo: 0  1  2  3  4  5  6  7  8  9  10
Act 0:     |--|                          [OK] Seleccionada
Act 1:        |--|                       [OK] Seleccionada
Act 2:  |----------|                     [X] No seleccionada (se solapa)
Act 3:           |--|                    [OK] Seleccionada
Act 4:                |--|               [OK] Seleccionada
Act 5:           |-------|               [X] No seleccionada (se solapa)

Solución óptima: 4 actividades
```

5. Demostración de Optimalidad

5.1. Teorema

El algoritmo greedy para selección de actividades produce una solución óptima.

Demostración por contradicción:

Supongamos que el algoritmo greedy no produce una solución óptima. Sea G la solución del algoritmo greedy y O una solución óptima con $|O| > |G|$.

Sea a_i la primera actividad en O que no está en G. Como el algoritmo greedy siempre selecciona la actividad que termina primero que no se solapa, y a_i no está en G, debe ser que a_i se solapa con alguna actividad en G que termina antes que a_i.

Pero esto es imposible porque a_i es parte de O, que es una solución válida. Por lo tanto, a_i no puede solaparse con actividades en G que terminen antes.

Esta contradicción demuestra que $|O| \leq |G|$, por lo que G es óptima.

6. Comparación con Fuerza Bruta

- **Complejidad:**
  - Greedy: $O(n \log n)$
  - Fuerza Bruta: $O(2^n \cdot n^2)$
- **Ventajas del Greedy:**
  - Mucho más eficiente para problemas grandes
  - Fácil de implementar
  - Produce solución óptima
- **Cuándo usar cada uno:**
  - Greedy: Para problemas reales y grandes
  - Fuerza Bruta: Para verificación y problemas pequeños

7. Conclusiones

El algoritmo greedy para selección de actividades es una solución elegante y eficiente que produce resultados óptimos en tiempo $O(n \log n)$. Su fortaleza radica en la elección inteligente del criterio de ordenamiento (por tiempo de finalización) y la estrategia de selección greedy, que garantiza la optimalidad de la solución.

Este algoritmo demuestra el poder de las estrategias greedy cuando se aplican correctamente a problemas que exhiben subestructura óptima y la propiedad de elección greedy.