Pseudocódigo: Programación de Trabajos con Algoritmo Greedy

Algoritmo Greedy

25 de septiembre de 2025

1. Descripción del Problema

El problema de programación de trabajos con deadline (Job Scheduling with Deadlines) consiste en seleccionar y programar un subconjunto de trabajos de manera que se maximice la ganancia total, respetando las restricciones de deadline. Cada trabajo tiene:

- Un identificador único
- Un deadline (tiempo límite para completar el trabajo)
- Una ganancia asociada

El objetivo es encontrar la secuencia de trabajos que maximice la ganancia total, donde cada trabajo debe completarse antes de su deadline y cada trabajo toma exactamente una unidad de tiempo.

El algoritmo greedy resuelve este problema ordenando los trabajos por ganancia descendente y asignando cada trabajo al slot más tardío posible dentro de su deadline.

2. Pseudocódigo

Algorithm 1 Programación de Trabajos con Algoritmo Greedy

Require: Lista de trabajos $jobs$ donde cada trabajo tiene $(id, deadline, profit)$
Ensure: Lista de trabajos seleccionados y ganancia total

Ordenar $jobs$ por ganancia en orden descendente
$max\_deadline \leftarrow$ máx$\{job.deadline$ para todo job en $jobs\}$
$slots \leftarrow$ arreglo de tamaño $max\_deadline + 1$ inicializado en -1
$total\_profit \leftarrow 0$
$scheduled\_jobs \leftarrow$ lista vacía
for cada job en $jobs$ (ordenados por ganancia descendente) **do**
    for $t \leftarrow job.deadline;\ t \geq 1;\ t \leftarrow t - 1$ **do**
        if $slots[t] = -1$ **then**
            $slots[t] \leftarrow job.id$
            $total\_profit \leftarrow total\_profit + job.profit$
            $scheduled\_jobs.append(job.id)$
            break
        end if
    end for
end forreturn $scheduled\_jobs, total\_profit$

3. Explicación del Algoritmo

3.1. Estrategia Greedy

El algoritmo greedy para programación de trabajos utiliza la siguiente estrategia:

- **Ordenamiento por ganancia:** Se ordenan los trabajos en orden descendente de ganancia
- **Asignación tardía:** Para cada trabajo, se busca el slot más tardío disponible dentro de su deadline
- **Justificación:** Al asignar trabajos de alta ganancia primero y al slot más tardío posible, se maximiza la flexibilidad para trabajos posteriores

La elección greedy de procesar trabajos por ganancia descendente es una heurística que, aunque no garantiza optimalidad en todos los casos, tiende a producir buenas soluciones en la práctica.

3.2. Subestructura Óptima

El problema de programación de trabajos exhibe subestructura óptima:
Si S es una solución óptima para el conjunto de trabajos J, y j es el trabajo con mayor ganancia en S, entonces:

- $S - \{j\}$ debe ser una solución óptima para $J - \{j\}$ con el slot ocupado por j liberado
- Esto permite construir la solución óptima de manera incremental

Sin embargo, el algoritmo greedy no siempre encuentra la solución óptima debido a la dependencia del orden de procesamiento.

3.3. Complejidad

- **Tiempo:** $O(n \log n + n \cdot d)$
  - $O(n \log n)$ para ordenar los trabajos por ganancia
  - $O(n \cdot d)$ para asignar trabajos, donde d es el deadline máximo
- **Espacio:** $O(d)$ para almacenar los slots, donde d es el deadline máximo

La complejidad es eficiente para problemas prácticos donde el deadline máximo no es muy grande.

4. Resolución Paso a Paso

4.1. Ejemplo: Trabajos = [A(2,100), B(1,19), C(2,27), D(1,25), E(3,15)]

Trabajos de entrada:

ID	Deadline	Ganancia
A	2	100
B	1	19
C	2	27
D	1	25
E	3	15

Paso 1: Ordenar trabajos por ganancia descendente

ID	Deadline	Ganancia
A	2	100
C	2	27
D	1	25
B	1	19
E	3	15

Paso 2: Inicialización

- $max\_deadline = \text{máx}\{2, 1, 2, 1, 3\} = 3$
- $slots = [-1, -1, -1, -1]$ (índices 0, 1, 2, 3)
- $total\_profit = 0$
- $scheduled\_jobs = []$

Paso 3: Procesar trabajos en orden de ganancia
Trabajo A (ganancia = 100, deadline = 2):

- Buscar slot desde $t = 2$ hacia atrás hasta $t = 1$
- $slots[2] = -1$ (disponible)
- Asignar: $slots[2] = A$, $total\_profit = 100$, $scheduled\_jobs = [A]$
- Estado: $slots = [-1, -1, A, -1]$

Trabajo C (ganancia = 27, deadline = 2):

- Buscar slot desde $t = 2$ hacia atrás hasta $t = 1$
- $slots[2] = A$ (ocupado)
- $slots[1] = -1$ (disponible)
- Asignar: $slots[1] = C$, $total\_profit = 127$, $scheduled\_jobs = [A, C]$
- Estado: $slots = [-1, C, A, -1]$

Trabajo D (ganancia = 25, deadline = 1):

- Buscar slot desde $t = 1$ hacia atrás
- $slots[1] = C$ (ocupado)
- No hay slots disponibles dentro del deadline
- **NO se puede asignar**
- Estado: $slots = [-1, C, A, -1]$ (sin cambios)

Trabajo B (ganancia = 19, deadline = 1):

- Buscar slot desde $t = 1$ hacia atrás
- $slots[1] = C$ (ocupado)
- No hay slots disponibles dentro del deadline
- **NO se puede asignar**
- Estado: $slots = [-1, C, A, -1]$ (sin cambios)

Trabajo E (ganancia = 15, deadline = 3):

- Buscar slot desde $t = 3$ hacia atrás hasta $t = 1$
- $slots[3] = -1$ (disponible)
- Asignar: $slots[3] = E$, $total\_profit = 142$, $scheduled\_jobs = [A, C, E]$
- Estado: $slots = [-1, C, A, E]$

Resultado Final:

- **Trabajos seleccionados:** [A, C, E]
- **Ganancia total:** 142
- **Programación:**
  - Tiempo 1: Trabajo C (deadline=2, ganancia=27)
  - Tiempo 2: Trabajo A (deadline=2, ganancia=100)
  - Tiempo 3: Trabajo E (deadline=3, ganancia=15)

4.2. Visualización de la Programación

Tiempo	Trabajo	Deadline	Ganancia
1	C	2	27
2	A	2	100
3	E	3	15
Total			**142**

4.3. Análisis de la Solución

- **Trabajos rechazados:** D y B (ambos con deadline=1)
- **Razón del rechazo:** El slot 1 ya está ocupado por C
- **Decisión greedy:** Se priorizó C (ganancia=27) sobre D (ganancia=25) y B (ganancia=19)
- **Eficiencia:** Se maximizó la ganancia total respetando las restricciones de deadline

5. Análisis del Algoritmo

5.1. Ventajas del Algoritmo Greedy

- **Eficiencia:** Complejidad $O(n \log n + n \cdot d)$
- **Simplicidad:** Fácil de implementar y entender
- **Práctico:** Funciona bien en la mayoría de casos reales
- **Rápido:** Tiempo polinomial

5.2. Limitaciones

- **No optimalidad:** No garantiza encontrar la solución óptima en todos los casos
- **Dependencia del orden:** El resultado puede variar según el orden de trabajos con igual ganancia
- **Casos patológicos:** Puede fallar en configuraciones específicas de deadlines y ganancias

Ejemplo de no optimalidad: Para trabajos con deadlines muy restrictivos, el algoritmo greedy puede rechazar trabajos de alta ganancia que podrían incluirse en una solución óptima.

5.3. Comparación con Fuerza Bruta

- **Complejidad:**
  - Greedy: $O(n \log n + n \cdot d)$
  - Fuerza Bruta: $O(n! \cdot n)$
- **Ventajas del Greedy:**
  - Mucho más eficiente para problemas grandes
  - Tiempo polinomial vs. factorial
  - Práctico para aplicaciones reales
- **Desventajas:**
  - No garantiza optimalidad
  - Puede producir soluciones subóptimas

6. Conclusiones

El algoritmo greedy para programación de trabajos es una solución eficiente y práctica que produce buenos resultados en la mayoría de casos. Aunque no garantiza optimalidad, su simplicidad y eficiencia lo hacen adecuado para muchas aplicaciones prácticas donde se requiere una solución razonable rápidamente.

Para casos donde se necesita garantizar la solución óptima, se requieren algoritmos más sofisticados como programación dinámica o técnicas de optimización, pero el algoritmo greedy proporciona una excelente base y punto de partida para el análisis de problemas de programación de trabajos.