Tarea 3: Análisis y Diseño de Algoritmos

Juan Diego Collazos Mejía

1 Greedy strategy for the class scheduling problem

Analizaremos cada estrategia voraz propuesta. Cuando el algoritmo sea correcto, realizaremos una **demostración de la óptima subestructura (optimización global)** y de **la elección voraz (optimización local)**. En caso contrario, proporcionaremos un **contraejemplo** que muestre que el algoritmo no siempre produce una solución óptima.

Análisis de Estrategias

(a) Elegir el curso que termina más tarde

No es óptimo. Este enfoque no garantiza una solución óptima porque seleccionar el curso que termina más tarde puede dejar menos espacio para incluir otros cursos.
Contraejemplo: Considera los siguientes cursos con sus intervalos de tiempo:

- $A = [1, 2]$
- $B = [2, 3]$
- $C = [3, 4]$
- $D = [1, 5]$

Si elegimos el curso que termina más tarde (D), no podemos seleccionar ningún otro curso. Sin embargo, la solución óptima sería elegir A, B y C, que permite seleccionar **tres cursos en lugar de uno**.

(c) Elegir el curso que empieza más tarde

Esta estrategia voraz funciona. La demostración esta en la siguiente pagina.

(e) Elegir el curso que tiene menos conflictos

No es óptimo. Seleccionar el curso que tiene menos conflictos no garantiza poder incluir la mayor cantidad.
Contraejemplo: Considera los siguientes cursos con sus intervalos de tiempo, con sus respectivos números de conflictos:

- $A = [1, 4] \to 3$
- $B = [4, 7] \to 4$
- $C = [7, 10] \to 4$
- $D = [10, 13] \to 3$
- $E = [6, 8] \to 2$
- $F = [1, 6] \to 4$
- $G = [8, 13] \to 4$
- $H = [1, 5] \to 4$
- $I = [2, 6] \to 4$
- $J = [9, 13] \to 4$
- $K = [8, 12] \to 4$

Si elegimos el curso que tiene menos conflictos (E), solo podremos permitirnos elegir 2 cursos más además del E lo cual no es una respuesta óptima. La respuesta óptima es tomar 4 cursos A, B, C y D.

(g) Si no hay conflictos, elegir todos; si los hay, descartar el curso con más conflictos

No es óptimo. La estrategia de si no hay conflictos, elegir todos; si los hay, descartar el curso con más conflictos, no es óptima porque el que tiene más conflictos puede ser parte de todas soluciones óptimas.
Contraejemplo: Considera los siguientes cursos con sus intervalos de tiempo, con sus respectivos números de conflictos:

- $A = [1, 2] \to 1$
- $B = [2, 5] \to 3$
- $C = [5, 8] \to 3$
- $D = [8, 10] \to 1$
- $E = [1, 3] \to 2$

- $F = [3, 7] \rightarrow 3$
- $G = [7, 10] \rightarrow 2$
- $H = [4, 6] \rightarrow 3$

Si se descartara alguno de los cursos con más conflicto (B, C, F). Note que los cursos B y C de la única solución óptima, que es escoger 4 cursos A, B, C y D. Por lo que descartar alguno de ellos sera una error y en consecuencia no se llegaría a la solución optima.

1.1 Demostración estrategia voraz: (c) Elegir el curso que empieza más tarde

Especificación del Problema

Entrada: Arreglos $S[0..N]$, $F[0..N]$, $N \geq 0$ tales que $S[n] < F[n]$ para todo $0 \leq n < N$. Sea $a_n = (S[n], F[n])$.
Salida: Cantidad máxima de actividades en $a_0, a_1, \ldots, a_{n-1}$ sin conflicto entre ellas.

Teorema de Optimización Local

Sea A un conjunto de actividades y sea a_m la actividad con mayor tiempo de inicio en A, entonces a_m forma parte de un conjunto maximal de actividades sin conflicto entre ellas.

***Demostración**

Sea $B \subseteq A$ un conjunto maximal de actividades sin conflicto de A y sea b_m la actividad con mayor tiempo de inicio en B. Hay dos posibilidades:

- Si $a_m = b_m$: En este caso, a_m forma parte de B, que es un conjunto maximal y sin conflictos de actividades de A, cumpliendo el teorema.
- Si $a_m \neq b_m$: Es necesario mostrar que a_m pertenece a un conjunto maximal y sin conflicto de A. Considere $B' = B - \{b_m\} \cup \{a_m\}$. B' no tiene conflictos porque el tiempo de inicio de a_m es máximo en A y, por ende, también en B. Además, $|B'| = |B|$, por lo que B' es óptimo. Dado que a_m pertenece a B', entonces forma parte de un conjunto maximal y sin conflicto de actividades de A.

En consecuencia, en cualquier caso a_m forma parte de un conjunto maximal y sin conflicto de actividades en A.

Función Objetivo

Sea $0 \leq n \leq N$ y $t \leq F[0]$:

$$\phi(n, t) :$$

Máxima cantidad de actividades en $A[n..N]$ que terminan antes de tiempo t y pueden ser agendadas sin conflicto.

Reformulación de la Salida

$$\phi(0, F[n]) :$$

Máxima cantidad de actividades en $A[0..N]$ que terminan antes de un tiempo en el que termina la actividad que comienza mas tarde y pueden ser agendadas sin conflicto.

Planteamiento Recursivo

Sea $0 \leq n \leq N$ y $t \leq F[0]$:

$$\phi(n,t) = \begin{cases} 0 & \text{si } n = N \\ 1 + \phi(n+1, S[n]) & \text{si } n \neq N \text{ y } F[n] \leq t \\ \phi(n+1,t) & \text{si } n \neq N \text{ y } F[n] > t \end{cases}$$

Teorema de Optimización Global

Sea $0 \leq n \leq N$, el llamado $\phi(n,t)$ produce la máxima cantidad de actividades en $A[0..N]$ que empiezan después del tiempo t y pueden ser agendadas sin conflicto.

*Demostración

Procedemos por inducción sobre la diferencia entre N y n.

*Caso Base: $n = N$

En este caso, $A[N..N]$ es un conjunto vacío, por lo tanto, la función produce 0, que es la cantidad óptima.

*Caso Inductivo: $n \neq N$

Supongamos como hipótesis inductiva que $\phi(n+1, t')$ produce la cantidad máxima de actividades que terminan en un tiempo menor o igual a t' y que pueden agendarse sin conflicto para cualquier $t' \geq t$. Consideramos dos casos:

- Si $F[n] \leq t$: Dado que las actividades en $A[0..N]$ están ordenadas por el tiempo de inicio, $A[n] = (S[n], F[n])$ es la actividad en $A[n..N]$ con mayor tiempo de inicio. Por el teorema de optimización local, $A[n]$ forma parte de un agendamiento óptimo y puede ser agendada. En este caso, $t' = S[n]$ y, por hipótesis inductiva, $\phi(n+1, t')$ produce una solución óptima para $A[n+1..N]$. En consecuencia, $1 + \phi(n+1, t')$ es óptimo para $A[n..N]$.

- Si $F[n] > t$: En este caso, la tarea $A[n]$ termina en un tiempo mayor a t y no puede ser agendada. Al hacer $t' = t$, por hipótesis inductiva $\phi(n+1, t')$ produce una solución óptima, que también lo es para $\phi(n,t)$, ya que $A[n]$ no puede ser agendada.

Por lo tanto, en cualquier caso, $\phi(n,t)$ produce la cantidad máxima de actividades en $A[n..N]$ que terminan antes de t y pueden ser agendadas sin conflicto.

*Corolario

El llamado $\phi(0, \infty)$ produce la cantidad máxima de actividades en $A[0..N]$ que terminan en un tiempo menor o igual a infinito y pueden ser agendadas sin conflicto.

Complejidad

La complejidad del algoritmo es $O(n \log(n))$ debido al ordenamiento del arreglo $A[0..N]$.

2 Climbing

2.1 Especificación del Problema

Entrada:

- Un árbol enraizado T con n vértices.
- Un entero positivo k que representa la longitud de los caminos.

Salida:

- El número máximo de caminos disjuntos de longitud exactamente k que avanzan hacia la raíz del árbol.

2.2 Teorema de Optimización Local

En un árbol enraizado T con n vértices, si en cada subárbol seleccionamos localmente la mayor cantidad de caminos disjuntos de longitud k que avanzan hacia la raíz, la suma de estos caminos en todos los subárboles maximiza globalmente el número total de caminos disjuntos de longitud k.

*****Demostración

Procedemos por inducción en el número de vértices del árbol T.

Caso base: Si el árbol tiene un solo vértice ($n = 1$), no hay caminos posibles de longitud $k \geq 1$, por lo tanto, el máximo número de caminos disjuntos es 0. El algoritmo voraz devuelve este valor correctamente.

Paso inductivo: Supongamos que el teorema es válido para todos los árboles con $n \leq m$ vértices. Sea T un árbol con $m + 1$ vértices.

Para cada nodo hijo del nodo raíz, el algoritmo calcula de manera óptima la cantidad de caminos de longitud k que se pueden formar dentro de cada subárbol.

Dado que los caminos no pueden intersectarse, el número total de caminos disjuntos de longitud k en el árbol completo se maximiza al sumar los caminos disjuntos en cada subárbol.

Al combinar localmente las soluciones óptimas de los subárboles, el algoritmo garantiza una solución global óptima debido a la estructura de árbol (donde no hay ciclos que puedan interferir con la solución local).

Por el principio de inducción, el algoritmo voraz maximiza el número total de caminos disjuntos de longitud k en cualquier árbol enraizado.

2.3 Función Objetivo

$$\Phi(T, u, parent, k)$$

Entrada:

- Un árbol enraizado T, representado por una lista de adyacencias.
- Un entero u que representa la raíz actual del árbol.
- Un entero *parent* que representa el nodo padre del nodo u.
- Un natural k que indica la longitud exacta de los caminos.

Salida: Un entero que representa el máximo número de caminos disjuntos de longitud k.

Reformulación de la salida:

$\Phi(T, 0, -1, k')$ devuelve una tupla (p, h) donde:

- p es la cantidad máxima de caminos disjuntos de tamaño k' en el árbol con raíz en 0.
- h es la altura máxima del subárbol sin solapamineto de caminos.

Implementación de la función

```
def max_disjoint_paths(u, parent, tree, k):
    ans = height = 0
    for child in tree[u]:
      if child != parent:
        paths_child, height_child = max_disjoint_paths(child, u, tree, k)
        height = max(height, height_child + 1)
        ans += paths_child
    if height + 1 == k:
      ans += 1
      height = 0
    return ans, height
```

2.4 Teorema de Optimización Global

Si, en cada subárbol del árbol T, se selecciona siempre el máximo número de caminos de longitud k que no se solapan, entonces la suma de estos máximos en todos los subárboles proporciona una solución global óptima.

2.5 Correctitud del Algoritmo

Invariante del algoritmo:

En cada nodo u del árbol durante el recorrido DFS, el algoritmo mantiene las siguientes propiedades:

- 1. *ans* representa el número máximo de caminos disjuntos de longitud k encontrados en el subárbol con raíz en u.
- 2. *height* indica la longitud del camino más largo desde el nodo actual u hasta alguna de sus hojas.

Demostración por inducción estructural:

Caso base: Si el árbol tiene un solo nodo (una hoja), no se puede formar ningún camino de longitud $k \geq 1$. El algoritmo devuelve $(0, 0)$ correctamente.

Hipótesis inductiva: Supongamos que el algoritmo es correcto para cualquier árbol con altura $h \leq m$.

Paso inductivo: Dado un nodo u con hijos $\{v_1, \ldots, v_p\}$, el algoritmo calcula recursivamente los caminos disjuntos y la altura del subárbol de cada hijo.

Se actualizan las variables *ans* y *height* para reflejar el número total de caminos y la altura máxima. Cuando se encuentra un camino completo de longitud k, se incrementa el contador y se reinicia la altura.

Por inducción, el algoritmo es correcto para cualquier árbol de altura $h = m + 1$.

Justificación por el Teorema de Optimización Local:

En cada paso del algoritmo, al recorrer los subárboles de manera recursiva, aplicamos el **Teorema de Optimización Local** al seleccionar el máximo número de caminos disjuntos de longitud k en cada subárbol. Esta estrategia garantiza que:

Localmente Óptimo: Para cada nodo u, el algoritmo maximiza el número de caminos disjuntos de longitud k utilizando la información de los hijos.

Consistencia Global: La combinación de soluciones óptimas en los subárboles asegura que no se omitan caminos que podrían formar una solución global mejor.

Dado que los caminos de cada subárbol son **disjuntos** por construcción y el algoritmo selecciona siempre la máxima cantidad posible, la combinación de estos valores en el nodo padre no puede ser superada por ninguna otra selección.

Relación con el Teorema de Optimización Global:

El teorema establece que si se realiza una optimización local (máximo número de caminos disjuntos en cada subárbol), la suma de estas soluciones locales proporciona una **solución global óptima**. Dado que el algoritmo satisface esta condición en cada paso de la recursión, garantiza que el resultado final es el número máximo global de caminos disjuntos de longitud k.

2.5.1 Corolario

El llamado $\Phi(T, 0, -1, k')$ produce una tupla (p, h) donde p es la cantidad máxima de caminos disjuntos de tamaño k' en el árbol con raíz en 0.

2.6 Complejidad del Algoritmo

El algoritmo realiza un recorrido DFS sobre el árbol, procesando cada nodo una vez, lo que implica una complejidad temporal de $O(n + m)$, donde n es el número de vértices y m el número de aristas del árbol.

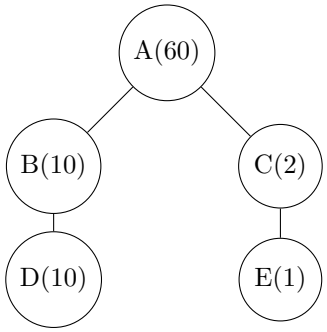
3 Greedy algorithm does not always return the optimal reward

Supongamos que cada vértice del árbol T tiene una recompensa asociada y que el objetivo es encontrar el conjunto de caminos disjuntos de longitud exactamente k que maximicen la suma de las recompensas en los vértices de esos caminos. A continuación, mostraremos mediante un contraejemplo que el algoritmo voraz no siempre proporciona la solución óptima en términos de recompensa total.

Contraejemplo

Consideremos el siguiente árbol enraizado:

- Raíz: vértice A con recompensa 60.
- Dos hijos directos de A:
  - B con recompensa 10.
  - C con recompensa 2.
- El vértice B tiene un hijo:
  - D con recompensa 10.
- El vértice C tiene un hijo:
  - E con recompensa 1.



Supongamos que queremos encontrar caminos disjuntos de longitud $k = 2$ que maximicen la suma de las recompensas.

Ejecución del algoritmo voraz

El algoritmo voraz elegirá los caminos más largos posibles de manera local en cada subárbol:

1. Desde la raíz A, evalúa los subárboles:
   - En el subárbol de B, encuentra el camino $[B \to D]$ con recompensa $10 + 10 = 20$.
   - En el subárbol de C, encuentra el camino $[C \to E]$ con recompensa $2 + 1 = 3$.
2. El algoritmo voraz selecciona estos dos caminos, obteniendo una recompensa total de $20 + 3 = 23$.

Solución óptima global

En cambio, si seleccionamos el camino $[A \to B]$, y el camino $[C \to E]$ obtenemos una recompensa total de:

$$10 + 60 + 2 + 1 = 73$$

La recompensa total óptima es 73, que es mayor que el resultado del algoritmo voraz. Por lo tanto el algoritmo voraz no garantiza la solución óptima al maximizar la recompensa total porque se centra en optimizar cada subárbol de manera independiente sin evaluar combinaciones más rentables a nivel global.