



Algoritmos Codiciosos

Estructuras de Datos y Algoritmos – IIC2133

Actividades que usan un mismo recurso

Demostramos que hay una solución óptima que comienza con la elección codiciosa de la actividad 1 (la que termina más temprano):

sea $A \subseteq S$ una solución óptima

ordenemos las actividades en A por hora de término

sea k la primera actividad en A

si $k = 1$, entonces A comienza con una elección codiciosa (y queda demostrado)

...


...

si $k \neq 1$, probamos que hay otra solución óptima B que sí empieza con la actividad 1:

sea $B = A - \{k\} \cup \{1\}$

... entonces como $f_1 \leq f_k$, las actividades en B son compatibles

... y como B tiene el mismo número de actividades que A , B también es una solución óptima (pero que incluye a la actividad 1)



Elegida la actividad 1, el problema se reduce a encontrar una solución óptima al mismo problema,

... pero sobre las actividades en S que son compatibles con la actividad 1

Demostramos por contradicción que si A es una solución óptima a S ,

... entonces $A' = A - \{1\}$ es una solución óptima a $S' = \{ i \in S : s_i \geq f_1 \}$:

si hubiera una solución B' a S' con más actividades que A' ,

... entonces agregando la actividad 1 a B' daría una solución B a S con más actividades que A ,

... contradiciendo que A es óptima

Pasos para diseñar algoritmos codiciosos como el anterior

1. Formular el problema como uno en el cual hacemos una elección y nos quedamos con un subproblema para resolver
2. Demostrar que hay una solución óptima al problema original que hace la elección codiciosa —la elección codiciosa es segura
3. Demostrar que, habiendo hecho la elección codiciosa, lo que queda es un subproblema tal que si combinamos una solución óptima al subproblema con la elección codiciosa hecha en 2, obtenemos una solución óptima al problema original (subestructura óptima)

Programación de tareas con plazos y ganancias

Sean:

(p_i, d_i) , $1 \leq i \leq n$, una instancia del problema

J el conjunto de tareas seleccionadas por nuestra estrategia

T el conjunto de tareas en una solución óptima

Demostramos que J y T tienen el mismo valor y por lo tanto J es óptimo



Suponemos

$J \neq T$ (de lo contrario, no hay nada que demostrar)

$T \not\subset J$ (de lo contrario, T no puede ser óptimo)

$J \not\subset T$ (la estrategia codiciosa prohíbe que $J \subset T$)



Por lo tanto, existen tareas a y b tales que

$$a \in J, \quad a \notin T, \quad b \in T, \quad b \notin J$$

Sea a la tarea de mayor ganancia tal que $a \in J$ y $a \notin T$:

de la estrategia codiciosa deducimos que $p_a \geq p_b$, para todas las tareas b tal que $b \in T$ y $b \notin J$

... de lo contrario, la estrategia codiciosa habría considerado b antes que a y la habría incluido en J

Tomemos las secuencias factibles S_J y S_T , para J y T ;

... sea i una tarea tal que

$$i \in J, i \in T$$

i está programada de t a $t+1$ en S_J , y de t' a $t'+1$ en S_T

Si $t < t'$, intercambiamos la tarea programada en $[t', t'+1]$ en S_J con i ; la nueva secuencia también es factible

... si $t' < t$, hacemos una transformación similar en S_T

Obtenemos las secuencias S_J' y S_T' en que todas las tareas comunes a J y T están programadas al mismo tiempo

Tomemos ahora el intervalo $[t_a, t_a+1]$ en S_J' en donde está programada la tarea a definida anteriormente

sea b la tarea programada en S_T' en ese intervalo:

de la elección codiciosa de a , $p_a \geq p_b$

si programamos a desde t_a hasta t_a+1 en S_T' y sacamos b ,
obtenemos una secuencia factible para el conjunto de tareas
 $T' = T - \{b\} \cup \{a\}$

Cómo demostrar la corrección de algoritmos codiciosos

1. Comparar la solución producida por la estrategia codiciosa con una solución óptima
2. Si difieren, entonces encontrar el primer x_k en el cual difieren
3. Luego, mostrar cómo hacer que el x_k de la solución óptima sea igual al de la solución codiciosa *sin reducir el valor total de la solución óptima*

El uso repetido de esta transformación demuestra que la solución codiciosa es óptima

Programación de tareas con duraciones

Tenemos n tareas para ejecutar en una única máquina

Cada tarea i , $1 \leq i \leq n$, tiene una duración de ejecución t_i

Queremos ejecutar todas las tareas de modo de *minimizar el promedio de los tiempos de finalización*



P.ej., supongamos que las tareas 1, 2, 3 y 4 duran 15, 8, 3 y 10:

- si las ejecutamos en orden 1, 2, 3 y 4,
... entonces los tiempos de finalización son 15, 23, 26 y 36,
... y el promedio es 25.00
- si las ejecutamos en orden 3, 2, 4 y 1,
... entonces los tiempos de finalización son 3, 1, 21 y 36,
... y el promedio es 17.75

Es decir, el orden de ejecución influye en el promedio que queremos minimizar

Minimizar el promedio de los tiempos de finalización es equivalente a minimizar la suma de los tiempos de finalización, ya que el número de tareas es siempre el mismo

Si las tareas se ejecutan en orden j_1, j_2, \dots, j_n

... entonces los tiempos son $t_{j_1}, t_{j_1}+t_{j_2}, t_{j_1}+t_{j_2}+t_{j_3}, \dots$

La suma S de estos tiempos es

$$S = \sum_{k=1 \dots n} (n - k + 1) t_{j_k}$$

La suma S de estos tiempos es

$$S = (n + 1) \sum_{k=1 \dots n} t_{jk} - \sum_{k=1 \dots n} k t_{jk}$$

El primer término es independiente del orden de ejecución

Si existieran $x > y$ tal que $t_{j_x} < t_{j_y}$,

... vemos que intercambiando j_x con j_y , el segundo término aumenta, disminuyendo S

Es decir, cualquier orden de ejecución en que las duraciones no sean monótonamente no decrecientes es subóptimo

Por lo tanto, las tareas deben ser ejecutadas según el orden menor duración primero