1. Ejercicio resuelto en clase en semana 5

1.1. Letra

Este es otro problema de planificación en el que hay que asignar un recurso a n tareas que lo solicitan. Cada tarea tiene una duración y un valor, ambos mayores que 0. El recurso debe ser asignado a todas las tareas pero cuando el es asignado a una tarea esa tarea lo usa de manera exclusiva durante toda su duración. Por ejemplo, el recurso puede ser una sala de conferencias y las tareas sesiones de un congreso, cada una de las cuales tiene una duración y una lista de participantes. La cantidad de participantes es lo que corresponde al valor de la tarea. Tanto el recurso como las tareas están disponibles desde el tiempo 0. Para la tarea i denotamos su duración con d_i y su valor con v_i. El resultado de la planificación debe ser los conjuntos de tiempos de inicio, s_i, y de finalización, f_i de cada tarea. La planificación debe minimizar la suma de los productos de los tiempos de finalización por el valor de cada tarea. O sea, debe minimizar $\sum_{i=1}^{n} v_i f_i$. La intuición de este objetivo en el ejemplo de la sala de conferencia es minimizar la suma global, teniendo en cuenta a todos los particicpantes, de los tiempos de espera y duración de la sesión.

(a) Dé una representación matemática del problema, que incluya los datos de entrada, el resultado y las restricciones.

(b) Resuelva el problema para los siguientes casos particulares: (1) todas las duraciones son iguales; (2) todos los valores son iguales.

(c) En el contexto de algoritmos greedy, encuentre dos o más reglas de selección que solucionen correctamente el problema en los dos casos anteriores.

(d) Diseñe ejemplos que permitan descartar algunas de esas reglas.

(e) Dé un algoritmo que resuelve el problema para una instancia genérica.

(f) Demuestre la corrección del algoritmo.

(g) Determine el orden de tiempo de ejecución del algoritmo.

1.2. Formalización

Entrada:
n tareas n >0
T = (. . . , ($d_{i,vi}$), . . .) secuencia de n pares, con d_i >0, v_i >0
1 recurso
todas las tareas y el recurso habilitados desde tiempo 0
Salida:
A = (. . . , ($s_{i,fi}$),. . .) secuencia de n pares

Recurso disponible en 0: $s_i \geq 0$

asignación continua: $f_i = s_i + d_i$

asignación exclusiva: P(i) <P(j) entonces $f_i \leq s_j$

P es una permutación de {1,. . . ,n} y P(i) es la posición de la tarea i en la salida

Las planificaciones que cumplen lo anterior se denominan *planificaciones válidas*. El conjunto de ellas es el *espacio de soluciones*.

La *función objetivo* de una planificación es $\text{Obj(A)} = \sum_{i=1}^{n} f_i v_i$

Optimización: min obj(A)

Encontrar una planificación en el espacio de soluciones para la cual la función objetivo sea mínima

Tiempos ociosos: Dadas dos tareas que se agendan de manera consecutiva, P(j) = P(i) + 1, ¿algo impide $s_j > f_i$, o sea, que el recurso quede con tiempos ociosos? Sería una planificación válida. Se puede demostrar fácilmente que una planiificación con tiempos ocisosos no es óptima.

Asumimos $s_j = f_i$. Transformamos el problema en hallar una permutación, reordenar la secuencia de entrada. De ahí calcularemos los s_i y f_i. Para no sobrecargar la notación suponemos que la permutación es la identidad.

De esta manera podemos expresar la función objetivo en función de los parámetros de entrada. La expresamos de dos maneras diferentes, una de las cuales muestra por que duraciones se multiplica cada valor y la otra muestra por que valores se multiplica cada duración.

d1 v1 + d1 v2 + d2 v2 + d1 v3 + d2 v3 + d3 v3

$\sum_{i=1}^{n} \sum_{j=1}^{i} d_j v_i = \sum_{j=1}^{n} \sum_{i=j}^{n} v_i d_j$

1.3. Casos particulares

Vamos a considerar dos casos particulares para ayudar a diseña la función de selección.

1. Mismas duraciones

   $\sum_{i=1}^{n} i v_i$ Para minimizar esta expresión los v_i deben ser decrecientes

1. Mismos valores

$\sum_{j=1}^{n}(n+1-j)d_j = (n+1)\sum_{j=1}^{n} d_j - \sum_{j=1}^{n} j d_j$ Para minimizar esta expresión los d_i deben ser crecientes

1.4. Función de selección

Diseñamos una función de selección:
$f(d_i, v_i) \geq f(d_j, v_j)$ significa que i va antes que j
Debe satistacer los casos particulares:
Si $d_i = d_j$ entonces $v_i \geq v_j$
Si $v_i = v_j$ entonces $d_i \leq d_j$
Consideramos dos funciones que cumplen con ello:
fr $(d_i, v_i) = v_i - d_i$
fq $(d_i, v_i) = v_i / d_i$

1.5. Ejemplos

Buscamos un ejemplo que permita descartar una de las soluciones
Dados (d_i, v_i) y (d_j, v_j)
¿sirve $d_i > d_j$ y $v_i < v_j$? No, porque ambas eligen lo mismo. Queremos que las planificaciones generados usando una función sea diferente a la que se genera con la otra.
Si $d_i > d_j$ se debe elegir $v_i > v_j$
$(d_i, v_i) = (5, 10)$, (d_j, v_j) $(1, 5)$

	i	j
fr	5	4
fq	2	5

Según fr la planificación es (i,j) y según fq es (j,i)
El resultado de la función objetivo usando cada una de las funciones es:
fr: $5 \cdot 10 + 6 \cdot 5 = 80$
fq: $1 \cdot 5 + 6 \cdot 10 = 65$
Descartamos fr
Nos queda fq(d,v) = v / d

1.6. Algoritmo

```
Calcular fq para cada tarea
t = 0
A = secuencia vacía
Mientras T no es vacío
  sea i en T que maximice fq(d_i, v_i)
  agregar i al final de A
  remover i de T
```

```
  s_i = t
  f_i = t + d_i
  t = f_i
La salida es A y los (s_i, t_i)
```

1.7. Corrección

Suponemos que renombramos las tareas para que A sea A = (1, 2, . . . , n)

Suponemos que O es otra planificación válida y óptima, diferente a A y que A no es óptima

(1 5 3 4 2 6) tiene inversiones. Una *inversión* es un par (i,j), i >j e i aparece antes que j en la planificación.

O tiene inversiones porque es diferente a A.

Si hay inversiones hay una inversión (i,j) en la que j aparece inmediatamente después de i. Se puede demostrar fácilmente por inducción.

Construimos O' igual a O excepto que invierte esa inversión Es igual a O desde la posición 1 hasta la anterior a i y desde la posterior a j hasta la posiciónn. Queda j antes que i.

Calcular Obj para cada una. Solo vemos las diferencias.

O: $(s_i + d_i)\ v_i + (s_i + d_i + d_j)\ v_j$

O': $(s_i + d_j)\ v_j + (s_i + d_i + d_j)\ v_i$

O: $d_i\ v_j$

O': $d_j\ v_i$

Según la regla elegida la inversión implica $fq(d_i, v_i) <= fq(d_j, v_j)$

1. Simplificación en la que asumimos que no hay dos tareas cuyas evaluaciones de fq sean iguales

   $v_i / d_i < v_j / d_j$, o sea

   $d_j\ v_i < d_i\ v_j$

   Por lo que resulta que obj(O') <obj(O) (O' es "mejor"que O) aunque habíamos supuesto que O es óptima. Esto es un absurdo que proviene de suponer que A no es óptima.

2. Caso general

   $v_i / d_i \leq v_j / d_j$

   $d_j\ v_i \leq d_i\ v_j$

   Como pueden ser iguales O' podría no ser "mejor"que O y por lo tanto no sería una contradicción haber supuesto que O es óptima.

Pero O' tiene una inversión menos. O' es "más cercana.ª A de lo que era O.

obj(O') no disminuyó pero tampoco aumentó. Como la cantidad de inversiones es finita, máximo n(n+1)/2, sucesivos cambios convierten la solución en A, por lo que A es óptima.

1.8. Complejidad

Adaptamos el algoritmo, preprocesando, para que en cada paso la elección de la tarea sea O(1)

Calcular los fq // O(n)

Ordenar // O(n log n)

Calcular los s_i, f_i // O(n)

Como hay una cantidad fija de pasos el tiempo queda determinado por el paso de mayor costo

O sea O(n log n).