Algoritmos y Estructuras de Datos II
Clase 11

Carreras:

Licenciatura en Informática

Ingeniería en Informática

2024

Unidad III

Técnicas de diseño de algoritmos

Algoritmos greedy (3)

Planificación con plazo y beneficio

Problema: Se tienen ***n*** trabajos cuya ejecución lleva una unidad de tiempo y que van a ser ejecutados en ***un solo servidor***, de manera secuencial, sin poder compartirlo en simultáneo.

Cada trabajo ***i*** , con $i=1,n$ tiene asignado:

- un **plazo *ti***
- y un **beneficio *bi*** solo en el caso que sea ejecutada dentro de su plazo ***ti*** .

Solución: Una solución será una ordenación de los trabajos que se comiencen a ejecutar dentro de los plazos.

Solución optima:

Entre las posibles soluciones se quiere encontrar la que tenga ***más beneficio***, esto es que maximice:

$$\sum_{i=1}^{n} b_i$$

Planificación con plazo y beneficio

Una forma de lograr una primera solución consiste en realizar las siguientes etapas:

- resolver el problema para todas las posibles soluciones,
- calcular el beneficio asociado a cada solución
- elegir la de mayor beneficio.

En principio las soluciones posibles son **n!** que serían las permutaciones de los n trabajos.

Por lo tanto este método sería muy costoso: O(n!)

Planificación con plazo y beneficio

Ejemplo:

n=4

b=(100,10,15,27)

t=(2,1,2,1)

posibles soluciones: n!=24

Como el plazo máximo es 2, se reducen a 12:

Secuencia	Es solución?	Beneficio
1,2	No	-
1,3	Si	115
1,4	No	-
2,1	Si	110
2,3	Si	25
2,4	No	-
3,1	Si	115
3,2	No	-
3,4	No	-
4,1	**Si**	**127**
4,2	No	-
4,3	Si	42

Planificación con plazo y beneficio

Se puede pensar en un método Greedy que construya la solución paso a paso:

- En cada paso agregue el trabajo de mayor beneficio bi de entre los que no se han usado, siempre y cuando con ese trabajo la solución sea factible.

Una solución será factible si existe alguna ordenación de los trabajos de modo que todos los trabajos se puedan ejecutar dentro de sus respectivos plazos.

- **Este método obtiene la solución óptima.**

(Ver algoritmo y demostración en la bibliografía:
"***Fundamentos de Algoritmia***". Brassard G., P. Bratley.
Prentice Hall, 1998, Seccion 6.6)

Planificación con plazo y beneficio

Solución Greedy para ejemplo anterior:

n=4 b=(100,10,15,27) t=(2,1,2,1)

- Elige tarea 1 (de mayor beneficio), la solución {1} es factible.
- Entre las tareas restantes elige tarea 4 (de mayor beneficio), la solución {1,4} es factible si se ejecuta en el orden (4,1)
- Entre las tareas restantes elige tarea 3 (de mayor beneficio), la solución {1,3,4} no es factible en ningún orden.
- Por último elige tarea 2, la solución {1,3,2} tampoco es factible en ningún orden.

Solución óptima {1,4} que sólo se puede ejecutar en el orden (4,1) con beneficio total de 127.

Asignación de Tareas

Problema:

Suponga que se dispone de n trabajadores y n tareas que ellos tienen que realizar.
Se conoce el costo de que cada trabajador realice cada una de las tareas, están guardados en una matriz B donde:

- $b_{ij} > 0$ es el costo de asignarle al trabajador i la tarea j.

Este es un problema que aparece con mucha frecuencia, en donde los costos son o bien salarios (que los trabajadores cobran por cada tarea) o bien tiempos (que demoran en realizarlas).

Asignación de Tareas

Solución:

Una *asignacion de tareas* puede ser expresada con una matriz $\boldsymbol{X}$ de $\boldsymbol{n}$ filas por $\boldsymbol{n}$ columnas con una asignación de los valores 0 o 1 a las componentes x_{ij}:

- $x_{ij} = 0$ si al trabajador i no se le ha asignado la tarea j,
- $x_{ij} = 1$ si al trabajador i le han asignado la tarea j.

Asignación de Tareas

Asignación válida:

Una *asignación válida* es aquella en la que a cada trabajador sólo le corresponde una tarea y cada tarea está asignada a un trabajador.

Se define el *costo de una asignación* válida como:

$$\text{costo} = \sum_{i=1}^{n}\sum_{j=1}^{n} x_{ij} b_{ij}$$

Solución óptima:

Se dice que una ***asignación es óptima*** si es de ***mínimo costo***.

$$\text{asignacion optima} = \text{minimo} \sum_{i=1}^{n}\sum_{j=1}^{n} x_{ij} b_{ij}$$

Asignación de Tareas

Para diseñar un algoritmo Greedy que resuelva este problema se puede pensar en dos estrategias distintas:

1) Asignar cada trabajador la *mejor tarea posible*.

2) Asignar cada tarea *al mejor trabajador* disponible.

Estas dos estrategias encuentran soluciones óptimas siempre?

Asignación de Tareas

Ejemplo 1 :

$$b = \begin{bmatrix} 16 & 20 & 18 \\ 11 & 15 & 17 \\ 17 & 1 & 20 \end{bmatrix}$$

Usar la estrategia greedy en que cada trabajador ejecuta la mejor tarea:

$$solucion\ greedy = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

costo=16+15+20=51

$$solucion\ \ optima = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

costo=18+11+1=30

CONCLUSION: No funciona la estrategia greedy a la mejor tarea

Asignación de Tareas

Ejemplo 2 :

$$b = \begin{bmatrix} 16 & 11 & 17 \\ 20 & 15 & 1 \\ 18 & 17 & 20 \end{bmatrix}$$

Usar la estrategia greedy en que cada tarea se asigna al mejor trabajador:

$$solucion\ greedy = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$ costo=16+15+20=51

$$solucion\ optima = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$ costo=18+11+1=30

CONCLUSION: No funciona la estrategia greedy al mejor trabajador

Problema de grabación de archivos

Problema 1: Archivos Y Almacenamiento

Suponga que dispone de:

n archivos $f1, f2, ..., fn$

con tamaños $l1, l2, ..., ln$

y una unidad de almacenamiento de capacidad $d < l1 + l2 + ... + ln$.

Objetivo: Se quiere *maximizar el número de archivos* que se puedan almacenar.

Algoritmo Greedy: se ordenan los archivos por orden creciente de su tamaño y se van grabando archivos hasta que no se puede grabar más.

Se puede demostrar que ***este algoritmo greedy encuentra siempre la solución óptima***, en todos los casos.

Problema de grabación de archivos

Demostración:

Suponga los archivos $f1$, $f2$, ..., fn ya se encuentran ordenados respecto a su tamaño:

$$l1 \leq l2 \leq ... \leq ln.$$

Si se llama L a la función que devuelve la longitud de un archivo dado, lo que tiene es que:

$$L(f1) \leq L(f2) \leq ... \leq L(fn).$$

El algoritmo Greedy indicado sugiere ir grabando los archivos según están ordenados hasta que no quepa ninguno más.

Se va a demostrar que el número de archivos que caben de esta forma es el óptimo (máximo).

Problema de grabación de archivos

Sea $\boldsymbol{m}$ el número de archivos que determina el algoritmo Greedy que caben en una unidad de almacenamiento de capacidad $\boldsymbol{d}$.

Se van a considerar 2 casos:

1) Es el caso en que todos los archivos caben en la unidad de almacenamiento.

Si $d \geq \sum_{i=1}^{n} L(\ fi\ )$ entonces: $m = n$

Esto es, la cantidad de archivos grabados m coincide con el total de archivos n

Problema de grabación de archivos

Sea ***m*** el número de archivos que determina el algoritmo Greedy que caben en una unidad de almacenamiento de capacidad ***d***.

2) Es el caso en que no todos los archivos caben en la unidad de almacenamiento.

Si no hay capacidad para almacenar todos los archivos: $d < \sum_{i=1}^{n} L(\ fi\ )$

Por la forma en la que el algoritmo construye la solución se sabe que se verifica la siguiente relación:

$$\sum_{i=1}^{m} L(\ fi\ ) \leq d < \sum_{i=1}^{m+1} L(\ fi\ ) \qquad \textbf{(*)}$$

Problema de grabación de archivos

Sea entonces $g1$, $g2$, ..., gs otro subconjunto de s archivos elegidos entre nos n disponibles que caben también, es decir, tal que:

$$\sum_{i=1}^{s} L(\ gi\ ) \leq d$$

Se va a demostrar que: $s \leq m$

En primer lugar, se puede suponer sin pérdida de generalidad que el conjunto de los archivos gi también está ordenado en orden creciente de tamaño:

$$L(g1) \leq L(g2) \leq \ldots \leq L(gs)$$

Como ambos subconjuntos son distintos, sea k el primer índice tal que: $fk \neq gk$.

Se puede suponer sin perder generalidad que $k = 1$, puesto que si hasta $fk-1$ los archivos son iguales se puede eliminarlos y restar la suma de los tamaños de tales archivos a la capacidad de la unidad de almacenamiento.

Problema de grabación de archivos

Por la forma en que funciona el algoritmo:

Si $f1 \neq g1$ entonces $L(f1) \leq L(g1)$ ya que $f1$ era el archivo de menor tamaño.

Además, $g1$ corresponderá a un archivo fa en la ordenación inicial, con $a > 1$.

Análogamente, $g2$ corresponderá a un archivo fb en la ordenación inicial, con $b > a > 1$, y por tanto $b > 2$, por lo que $L(g2) \geq L(f2)$.

Repitiendo el razonamiento, los archivos gi se corresponderán con archivos de la ordenación inicial, pero siempre cumpliendo que:

$$L(gi) \geq L(fi) \qquad (1 \leq i \leq s)$$

Problema de grabación de archivos

Por las relaciones anteriores: $\sum_{i=1}^{s} L(\ gi\ ) \le d$ y $L(gi) \ge L(fi),\ i=1,s$

se obtiene:

(**) $$d \ge \sum_{i=1}^{s} L(\ gi\ ) \ge \sum_{i=1}^{s} L(\ fi\ )$$

$$\sum_{i=1}^{m} L(\ fi\ ) \le d < \sum_{i=1}^{m+1} L(\ fi\ )$$

Usando también la relación **(*)** que implica que:

(***) $$\sum_{i=1}^{m+1} L(\ fi\ ) > d$$

De (**) y (***) se puede escribir entonces que:

$$\sum_{i=1}^{m+1} L(\ fi\ ) > d \ge \sum_{i=1}^{s} L(\ fi\ )$$

De esto se concluye que s tiene que ser estrictamente menor que $m+1$, $s < m+1$ y por tanto $\boldsymbol{s \le m}$, como se quería demostrar.

Problema de grabación de archivos

Problema 2: Archivos Y Almacenamiento

Suponga que dispone de:

n archivos $f1, f2, ..., fn$

con tamaños $l1, l2, ..., ln$

y un dispositivo de capacidad $d < l1 + l2 + ... + ln$.

Objetivo: Se quiere ***utilizar la mayor capacidad posible*** de la unidad de almacenamiento, esto es, llenarla tanto como se pueda.

Algoritmo Greedy: ordenar los archivos por orden decreciente de su tamaño y se van grabando archivos hasta que no se pueda grabar más.

Determinar si este algoritmo greedy encuentra solución óptima en todos los casos.-

Problema de carga de combustible

Problema:

- Un viajante recorre una ruta entre dos ciudades ***A*** y ***B*** separadas ***N*** kilómetros.
- Parte de ***A*** con el tanque lleno de combustible.
- Tiene autonomía de ***M*** kilómetros que puede recorrer sin detenerse.
- Tiene un mapa de rutas con la información de todas las distancias parciales entre las estaciones de servicio que hay en su recorrido entre las dos ciudades.
- Para minimizar el tiempo empleado en recorrer su ruta, el conductor quiere *parar a cargar combustible el menor número posible de veces*.

Problema de carga de combustible

Algoritmo:

- Un algoritmo greedy que determine en qué estaciones tiene que parar a cargar combustible consiste, recorrer el mayor número posible de kilómetros sin cargar, según lo determine su autonomía, tratando de ir desde cada estación en donde se pare a cargar combustible a la siguiente lo más alejada posible, así siguiendo hasta llegar al destino.

- Demostrar que este algoritmo greedy encuentra siempre una solución óptima.

Problema de carga de combustible

Datos del problema:

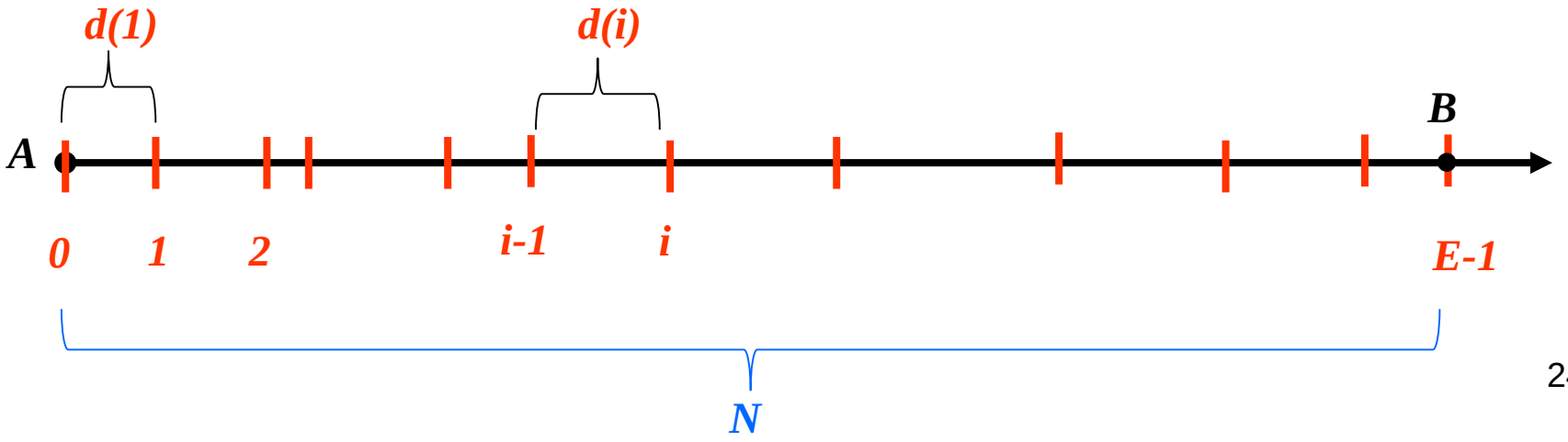
Separación de las 2 ciudades: ***N*** kilómetros .

Autonomía del vehículo: ***M*** kilómetros

Cantidad de estaciones de servicio: ***E***, numeradas de 0 (ciudad de partida) a ***E-1*** (ciudad de destino)

Distancia entre las estaciones de servicio: vector ***d(1..E-1),*** donde ***di*** es la distancia en kilómetros entre la estación ***i-1*** y la estación ***i***.

En un gráfico:

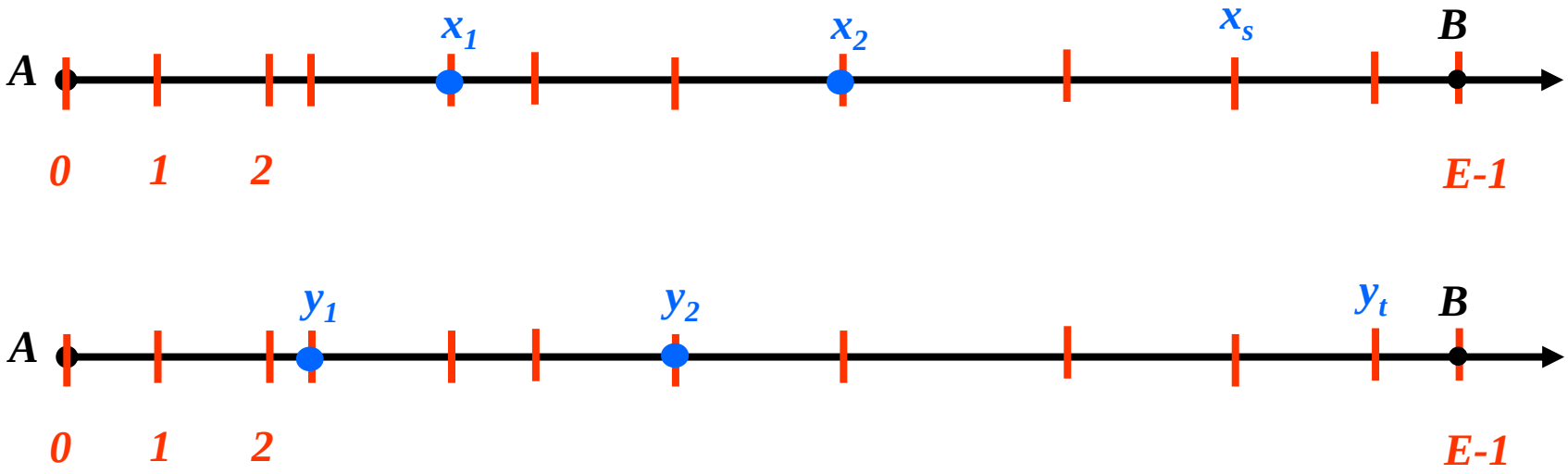


Problema de carga de combustible

Sea $X = (x_1, x_2, \ldots, x_s)$ la solución encontrada por la técnica greedy, que enumera *s* las estaciones *xi* , *i=1,s* donde va a parar a cargar.

Sea $Y = (y_1, y_2, \ldots, y_t)$ la solución encontrada por otra técnica, que enumera *s* las estaciones *yi* , *i=1,t* donde va a parar a cargar.

Se debe demostrar que **$s \leq t$** ya que la solución greedy hace menos paradas.



Problema de carga de combustible

Se puede armar el vector D, tal que $D(i)$ es la distancia total recorrida hasta la estación i:

$$D(i) = \sum_{k=1}^{i} d(k) \qquad D(E-1) = N$$

Como las soluciones X e Y son distintas, sea k el primer índice tal que $x_k \neq y_k$. Se puede suponer sin perdida de generalidad que $k=1$.

Ya que si no fuera así, se podría restar del camino todas las partes que fueran coincidentes en la elección de las estaciones.

Si $k=1$, entonces: $x_1 \neq y_1$, por la construcción de la solución X que es greedy, se concluye que:

$x_1 > y_1$ y además se cumple que: $D(x_1) > D(y_1)$

Problema de carga de combustible

También se cumple que: $x_2 \geq y_2$ ya que la estrategia greedy supone la elección de la estación mas alejada.

Si se supone por el contrario que: $y_2 > x_2$ y la solución Y puede ir de y_1 a y_2 hay menos de M kilómetros entre ellas:

$$D(y_2) - D(y_1) < M$$

y dado que $D(y_1) < D(x_1)$, también desde x_1 hay menos de M kilómetros a y_2 entonces:

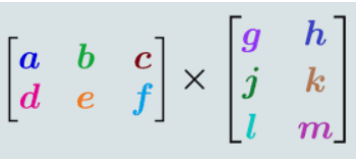
$$D(y_2) - D(x_1) < M$$

Entonces el algoritmo greedy no hubiera elegido a x_2 como estación siguiente a x_1 sino a la estación y_2 , porque siempre elige la más alejada que puede alcanzar de la autonomía. De este modo, la suposición $y_2 > x_2$ es falsa.

Repitiendo el razonamiento se puede probar que en cada paso:

$x_k \geq y_k$ para todo k

Multiplicación de Matrices



Producto de A_{mxq} por una matriz B_{qxn} es una matriz $C_{mxn} = A_{mxq}$ x B_{qxn} cuyas componentes son:

$$c_{i,j} = \sum_{k=1}^{q} a_{i,k} \cdot b_{k,j} \qquad 1 \le i \le m \quad 1 \le j \le n$$

Algorítmicamente se tiene un lazo triple anidado de la forma:

```
Para i = 1, m hacer
    Para j = 1, n hacer
        c(i,j) ← 0
        Para k = 1, q hacer
            c(i,j) ← c(i,j)+a(i,k)*b(k,j)
```

T(*m.n.q*) ϵ O(*m.n.q*)

Multiplicación Óptima De Matrices

Se necesita calcular la matriz producto M de n matrices dadas:

$$M=M_1M_2...M_n.$$

Por ser asociativa la multiplicación de matrices, existen muchas formas posibles de realizar esa operación, cada una con un costo asociado (en términos del número de multiplicaciones escalares).
En general, el costo asociado a las distintas formas de multiplicar las n matrices puede ser bastante diferente en unas y otras.

Si M_i es de dimensión d_{i-1} x d_i ($1 \leq i \leq n$), entonces multiplicar M_i x M_{i+1} requiere d_{i-1} x d_i x d_{i+1} operaciones.

El problema consiste en: ***encontrar el mínimo número de operaciones*** necesario para calcular el producto M.

Multiplicación Óptima De Matrices

Ejemplo 1: n = 4 y las matrices M_1, M_2, M_3 y M_4 cuyos órdenes son: $M_1(30\times1)$, $M_2(1\times40)$, $M_3(40\times10)$, $M_4(10\times25)$.

Existen cinco formas distintas de multiplicarlas. Sus costos (en términos de las multiplicaciones) son:

- $\mathbf{((M_1M_2)M_3)M_4}$ = $30\cdot1\cdot40 + 30\cdot40\cdot10 + 30\cdot10\cdot25$ = **20700**
- $\mathbf{M_1(M_2(M_3M_4))}$ = $40\cdot10\cdot25 + 1\cdot40\cdot25 + 30\cdot1\cdot25$ = **11750**
- $\mathbf{(M_1M_2)(M_3M_4)}$ = $30\cdot1\cdot40 + 40\cdot10\cdot25 + 30\cdot40\cdot25$ = **41200**
- $\mathbf{M_1((M_2M_3)M_4)}$ = $1\cdot40\cdot10 + 1\cdot10\cdot25 + 30\cdot1\cdot25$ = **1400**
- $\mathbf{(M_1(M_2M_3))M_4}$ = $1\cdot40\cdot10 + 30\cdot1\cdot10 + 30\cdot10\cdot25$ = **8200**

Como puede observarse, la 4ª. forma necesita casi treinta veces menos multiplicaciones que la 3ª., por lo cual es importante elegir una buena asociación.

Multiplicación Óptima De Matrices

Se podría pensar en una solución que:

- Calcule el costo de cada una de las opciones posibles.
- Elija en entre ellas la mejor de entre ellas antes de multiplicar.

Sin embargo, para valores grandes de n esta estrategia es inútil, pues el número de opciones crece exponencialmente con n.

De hecho, el número de opciones posible sigue la sucesión de los números de Catalán:

$$T(n) = \sum_{i=1}^{n-1} T(i).T(n-i) = \frac{1}{n}\binom{2n-2}{n-1} \in \Theta\left(\frac{4^n}{\sqrt{n}}\right)$$

$$T(0)=1$$

Multiplicación Óptima De Matrices

Parece entonces muy útil la búsqueda de un algoritmo greedy que resuelva este problema.

Se presentan cuatro estrategias diferentes:

a) Multiplicar primero las matrices $M_i M_{i+1}$ cuya dimensión común d_i sea la ***menor*** entre todas, y repetir el proceso.

b) Multiplicar primero las matrices $M_i M_{i+1}$ cuya dimensión común d_i sea la ***mayor*** entre todas, y repetir el proceso.

c) Realizar primero la multiplicación de las matrices $M_i M_{i+1}$ que requiera ***menor*** número de operaciones ($d_{i-1} d_i d_{i+1}$), y repetir el proceso.

d) Realizar primero la multiplicación de las matrices $M_i M_{i+1}$ que requiera ***mayor*** número de operaciones ($d_{i-1} d_i d_{i+1}$), y repetir el proceso.

Multiplicación Óptima De Matrices

Alguna de las estrategias propuestas encuentra siempre solución óptima?

Como es habitual en los algoritmos greedy para comprobar su funcionamiento, sería necesario una demostración formal o bien dar un contraejemplo que justifique la respuesta.

Multiplicación Óptima De Matrices

a) La primera estrategia consiste en multiplicar siempre primero las matrices $M_i\ M_{i+1}$ cuya dimensión común d_i es la menor entre todas.

Ejemplo 1: n = 4 y las matrices M_1, M_2, M_3 y M_4 cuyos órdenes son: $M_1(30\times1)$, $M_2(1\times40)$, $M_3(40\times10)$, $M_4(10\times25)$

- $\boldsymbol{(M_1M_2)(M_3M_4) = 30\cdot1\cdot40 + 40\cdot10\cdot25 + 30\cdot40\cdot25 = 41200}$

Observando el ejemplo la elección greedy corresponde al producto $(M_1M_2)(M_3M_4)$ que resulta ser el peor de todos, esta estrategia no da resultado óptimo (que es $\boldsymbol{M_1((M_2M_3)M_4)}$ y requiere 1400).

La estrategia greedy no da la solución óptima.

Multiplicación Óptima De Matrices

b) Si se multiplica siempre primero las matrices $M_i M_{i+1}$ cuya dimensión común d_i es la mayor entre todas se encuentra la solución óptima para el ejemplo anterior, pero existen otros ejemplos donde falla esta estrategia, como el siguiente.

Ejemplo 2: n = 3 y las matrices M_1, M_2, M_3 cuyos órdenes son: $M_1(2\times5)$, $M_2(5\times4)$ y $M_3(4\times1)$.

Según esta estrategia, el producto elegido como mejor sería:

- $(M_1M_2)M_3$ = 2·5·4 + 2·4·1 = 48

Sin embargo, el producto:

- $M_1(M_2M_3)$ = 5·4·1 + 2·5·1 = 30, es menor que el anterior.

La estrategia greedy no da la solución óptima.

Multiplicación Óptima De Matrices

c) Si se decide realizar siempre primero la multiplicación de las matrices M_i . M_{i+1} que requiera menor número de operaciones, se encontraría la solución óptima para los dos ejemplos anteriores, pero no para el siguiente:

Ejemplo 3: n = 3 y las matrices M_1, M_2, M_3 cuyos órdenes son: $M_1(3\times1)$, $M_2(1\times100)$ y $M_3(100\times5)$.

Según esta estrategia, el producto elegido como mejor sería:

- $(M_1M_2)M_3$ = 3·1·100 + 3·100·5 = 1800

Sin embargo, el costo del producto:

- $M_1(M_2M_3)$ = 1·100·5 + 3·1·5 = 515, que es menor que el anterior.

La estrategia greedy no da la solución óptima.

Multiplicación Óptima De Matrices

d) Análogamente, realizando siempre primero la multiplicación de las matrices $M_i\ M_{i+1}$ que requiera mayor número de operaciones se puede encontrar la solución óptima para este último ejemplo (Ejemplo 3), pero no para los dos primeros (Ejemplo 1 y 2).
La estrategia greedy no da la solución óptima.

Lamentablemente, ninguna de las estrategias presentadas encuentra la solución óptima.

Por tanto, para todas es posible dar un contraejemplo en donde el algoritmo falla.

CONCLUSION: ninguna de las 4 estrategias Greedy planteadas encuentra la solución optima del problema.-

Algoritmos Greedy

Trabajo Práctico no. 5