

ejgreedy.pdf



Anónimo



Algorítmica



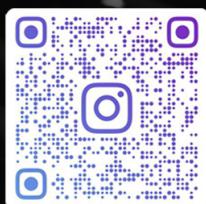
2º Grado en Ingeniería Informática



Escuela Técnica Superior de Ingenierías Informática y de
Telecomunicación
Universidad de Granada



LA PRIMERA RESIDENCIA GAMING
EN EL MUNDO ABRE EN MADRID
ESCANEA Y PARTICIPA EN EL
SORTEO DE UN ALIENWARE



gamingresidences.com

info@gamingresidences.com

ESCANEÁ Y PARTICIPA EN EL
SORTEO DE UN ALIENWARE



LA PRIMERA RESIDENCIA GAMING
EN EL MUNDO ABRE EN MADRID



GAMING RESIDENCES

gamingresidences.com

info@gamingresidences.com



LA PRIMERA RESIDENCIA GAMING EN
EL MUNDO ABRE EN MADRID
ESCANEAR EL CÓDIGO QR Y PARTICIPA
EN EL SORTEO DE UN ALIENWARE



gamingresidences.com
info@gamingresidences.com

El problema del barco de mercancías

Se tiene un barco de mercancías cuya capacidad de carga es de K toneladas y un conjunto de contenedores C_1, C_2, \dots, C_n cuyos pesos respectivos (en toneladas) son: P_1, P_2, \dots, P_n . La capacidad del barco es inferior a la suma total de los pesos de los contenedores.

Discretar un algoritmo greedy que maximice el número de contenedores cargados.

Lista de candidatos C : conjunto de contenedores.

Lista de seleccionados S : solución parcial con los contenedores seleccionados hasta el momento.

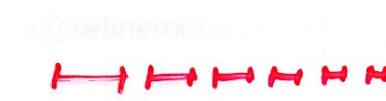
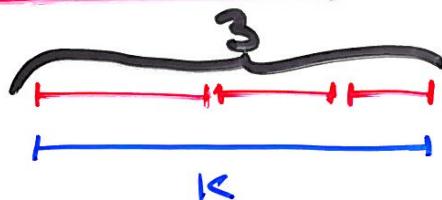
Función solución: cuando no quedan elementos / candidatos factibles

Función objetivo: n : elementos en S

Función selección: Posibilidades:

1. Lograr el siguiente contenedor de menor peso
2. Lograr el siguiente contenedor de mayor peso

Selección 2



cabrían mas !!

Optaremos por la selección 1

Algoritmo

MaxWut (C, S)

$S = \emptyset$

$P = \emptyset$

Mientras ($C \neq \emptyset$) hacer u vemos

\downarrow

$x = \text{buscar mínimo } (C) \quad 14, 4-1, 4-2$

$C = C - \{x\}$

 Si ($P + \text{peso}(x) \leq k$)

\downarrow

$S = S \cup \{x\}$

$\{$ O(1)

\downarrow

$P = P + \text{peso}(x)$

\downarrow

$O(n^2)$

para podría ser logu si se hace una ordenación previa de los contenidos por peso
& ir seleccionandolos mientras queden

¿y si hubiera que diseñar un algoritmo greedy para maximizar el número de tocaduras conseguidas?

Encontrar contrejemplos que prueben que en este caso el algoritmo greedy no es óptimo.

El problema del fontanero

Un fontanero necesita hacer n reparaciones urgentes, y sabe de antemano el tiempo que le va a llevar cada una de ellas: en la tarea i -ésima tardará t_i minutos.

Se trata de decidir el orden en el que atenderá los avisos para minimizar el tiempo medio de espera de los clientes.

Si E_i es el tiempo de espera del cliente i -ésimo hasta ver reparada su avería por completo, se haría de minimizar la expresión

$$E(n) = \sum_{i=1}^n E_i$$

El fontanero siempre tardará el mismo tiempo global $T = t_1 + t_2 + \dots + t_n$ en realizar todas las reparaciones independientemente de cómo las ordene. Sin embargo, los tiempos de espera de los clientes si dependen de esta ordenación.



LA PRIMERA RESIDENCIA GAMING EN
EL MUNDO ABRE EN MADRID

ESCANEA EL CÓDIGO QR Y PARTICIPA
EN EL SORTEO DE UN ALIENWARE



Si se mantiene la ordenación original de las tareas $(1, 2, \dots, n)$ la expresión de los tiempos de espera de los clientes viene dada por:

$$E_1 = t_1$$

$$E_2 = t_1 + t_2$$

$$\vdots \vdots \vdots \vdots$$

$$E_n = t_1 + t_2 + \dots + t_n$$

Tenemos que encontrar una permutación de las tareas donde se minimice $E(n)$, es decir minimizar:

$$E(n) = \sum_{i=1}^n E_i = \sum_{i=1}^n t_i + \cancel{t_1 + t_2 + \dots + t_n}$$

Solución greedy

La permutación óptima es aquella en la que los avisos se atienden en orden creciente de sus tiempos de reparación.

$$\begin{aligned} \sum_{i=1}^n E_i &= t_1 + (t_1 + t_2) + (t_1 + t_2 + t_3) + \dots + \\ &+ (t_1 + t_2 + \dots + t_n) = n t_1 + (n-1) t_2 + \\ &\dots + \dots = \sum_{k=1}^n (n-k+1) t_k \end{aligned}$$

PRODUCTO DE MATRICES

$$\left. \begin{array}{l} M_1 \rightarrow 10 \times 20 \\ M_2 \rightarrow 20 \times 50 \\ M_3 \rightarrow 50 \times 1 \\ M_4 \rightarrow 1 \times 100 \end{array} \right\}$$

$$M_1 \times M_2 \times M_3 \times M_4$$

a)

Reservados todos los derechos.
Lo se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

Multiplicación óptima de matrices

$$M_1 [30 \times 1] \times M_2 [1 \times 40] \times M_3 [40 \times 10] \times M_4 [10 \times 25]$$

$$((M_1 M_2) M_3) M_4 = 30 \cdot 1 \cdot 40 + 30 \cdot 40 \cdot 10 + 30 \cdot 10 \cdot 25 = 20.700$$

$$(M_1 (M_2 (M_3 M_4))) = 40 \cdot 10 \cdot 25 + 1 \cdot 40 \cdot 25 + 30 \cdot 1 \cdot 25 = 11.750$$

$$(M_1 M_2) (M_3 M_4) = 30 \cdot 1 \cdot 40 + 40 \cdot 10 \cdot 25 + 30 \cdot 40 \cdot 25 = 41.200$$

$$\underline{M_1 ((M_2 M_3) M_4)} = 1 \cdot 40 \cdot 10 + 1 \cdot 10 \cdot 25 + 30 \cdot 1 \cdot 25 = 1.400$$

$$(M_1 (M_2 M_3)) M_4 = 1 \cdot 40 \cdot 10 + 30 \cdot 1 \cdot 10 + 30 \cdot 10 \cdot 25 = 8.200$$

Possibles estrategias

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

$(M_1 M_2) (M_3 M_4)$ Es peor resultado

2. Igual que 1 cambiando menor por mayor

$M_1 ((M_2 M_3) M_4)$ óptimo en el ej. prop.

Ejemplo: $M_1 [2 \times 5] \times M_2 [5 \times 4] \times M_3 [4 \times 1]$

Hasta $(M_1 M_2) M_3$ ($2 \cdot 5 \cdot 4 + 2 \cdot 4 \cdot 1 = 48$) es el producto $M_1 (M_2 M_3)$ ($5 \cdot 4 \cdot 1 + 2 \cdot 5 \cdot 1 = 30$) es menor

3. Realizar primero la multiplicación de las matrices $M_i M_{i+1}$ que requiere menor número de operaciones $[d_{i-1} d_i d_{i+1}]$ y repetir el proceso

$$M_1 ((M_2 M_3) M_4) \quad \left. \begin{array}{l} \\ M_1 (M_2 M_3) \end{array} \right\} \text{óptimo en los 2 ej.}$$

pero:

$$M_1 [3 \times 1] \times M_2 [1 \times 100] \times M_3 [100 \times 5]$$

$$\text{haria: } (M_1 M_2) M_3 \quad (3 \cdot 1 \cdot 100 + 3 \cdot 100 \cdot 5 = 1800)$$

$$\text{pero: } M_1 (M_2 M_3) \quad (1 \cdot 100 \cdot 5 + 3 \cdot 1 \cdot 5 = 515) \text{ es mejor}$$

4. Realizar primero la multiplicación de las matrices $M_i M_{i+1}$ que requiere mayor número de operaciones $[d_{i-1} d_i d_{i+1}]$ y repetir el proceso

óptimo en el último ejemplo pasó lo en los 2 anteriores

Estrategia greedy mala: la 3



LA PRIMERA RESIDENCIA GAMING EN
EL MUNDO ABRE EN MADRID
ESCANEAR EL CÓDIGO QR Y PARTICIPA
EN EL SORTEO DE UN ALIENWARE



Problema de ejecución de tareas

Tenemos que ejecutar un conjunto de n tareas, cada una de las cuales requiere un tiempo unitario. En un instante $T = 1, 2, \dots$ podemos ejecutar únicamente una tarea. La tarea i produce un beneficio $g_i (> 0)$ solo en el caso de que sea ejecutada en un instante anterior o igual a d_i .

Utilizando la técnica greedy, encontrar un algoritmo que nos permita seleccionar el conjunto de tareas a realizar de forma que nos aseguremos que tenemos la mayor ganancia posible.

Aplicarlo al siguiente ejemplo:

i	1	2	3	4
g_i	50	40	15	30
d_i	2	1	2	1

- * Lista de candidatos C : conjunto de tareas
- * Lista de seleccionados S : las tareas seleccionadas hasta el momento
- * Función solución: cuando no quedan más tareas factibles
- * Función objetivo: Beneficios acumulados por las tareas en S :
$$B = \sum_{i \in S} g_i$$
- * Función de factibilidad: $(S \cup \{x_i\})$ es factible si todas las tareas $i \in (S \cup \{x_i\})$ pueden ser ejecutadas en un instante anterior o igual a d_i
- * Función de selección:
Cojer las tareas con mayor beneficio

Si pensamos en otras: P. ej: escoger primero aquellas que deben ser ejecutadas antes, tenemos:

i	1	2	3	4	
g_i	10	5	50	30	
d_i	2	2	3	3	

→ No entra y

sin embargo es mejor que las 2 primarias

Algoritmo

Lo complicado está en poder comprobar de forma eficiente si incluir una nueva tarea genera conflicto. Para ello se puede mantener el vector solución S ordenado (según d_i) tratando de insertar la nueva tarea en su posición correspondiente. Es fácil comprobar después si alguna de las tareas desplazadas o la propia tarea insertada se encuentra en una posición mayor a su correspondiente d_i , en cuyo caso añadir la tarea NO es factible (se numera de desorden)

```

void factible_insertar (datos* s, int & elems, datosx)
{
    // datos es una estructura con los valores
    int factible=1, v, pos
    di, gi

    for (i=elems; i>0 && x.d < s[i-1].d
        && factible; i--)
        if (s[i-1] > i) factible=0;

    pos=i;
    if (factible && x.d <= pos) // comprobacion
    {
        factible // factibilidad
        for (i=elems; i>pos; i--)
            s[i]=s[i-1];
        s[pos]=x;
        elems++;
    }
    return y;
}

```

```

void MaxBeneficio (datos* c, int n, datos* s,
                    int & elems)

```

```

{
    int i;
    ordenarporbeneficios (c, n);
    elems=0;
    for (i=0; i<n; i++)
        factible_insertar (s, elems, c[i]);
    return factible_insertar (s, elems, c[i]);
}

```



LA PRIMERA RESIDENCIA GAMING EN
EL MUNDO ABRE EN MADRID
ESCANEAR EL CÓDIGO QR Y PARTICIPA
EN EL SORTEO DE UN ALIENWARE



i	0	1	2	3
gi	50	10	15	30
di	1	0	1	0

(se adaptan los
valores para que crezcan
de 0)

↓ ordenamos por beneficio

gi	50	30	15	10
di	1	0	1	0

Iteración 0

s	g	50	0
d	1		

elements = 1

Iteración 1

s	g	30	0	1
d	0		1	

elements = 2

Iteración 2 y 3

Ningún otro elemento puede ser intercambiado

s	g	30	50	0
d	0		1	

elements = 2

Solución óptima con beneficio 80

Ejercicio: Demostrar la optimalidad