Documento anónimo

tema-15.pdf

Apuntes completos

2º Algorítmica

Grado en Ingeniería Informática

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

Programación Dinámica

Algoritmos

Teoría de Algoritmos

Capítulo 5: Programación Dinámica

Tema 15: Algoritmos basados en P.D.

- El Problema de la Mochila
- El Problema del Camino Mínimo: Algoritmo de Floyd
- El Problema del Viajante de Comercio
- Otros ejemplos.

BARCELÓ DESALIA

LIVE AHORA

5 DÍAS DE FIESTA.
MÁS DE 1000 INVITADOS.
PLAYOTE, PERREO Y RON BARCELÓ.

EL MOMENTO PARA VIVIR DESALIA ES AHORA

Participa en VIVEAHORADESALIA.COM

www.disfruta-de-un-consumo-responsable.com 37,5º. Promoción válida en España para mayores de 18 años hasta el 10 de mayo de 2019

Aplicación de la P.D. al Diseño de Algoritmos

PD se aplica en cuatro fases

- Naturaleza n-etápica del problema
- Verificación del POB
- Planteamiento de una recurrencia
- Cálculo de la solución (enfoque adelantado o atrasado)

Las desarrollaremos sobre

- Problema del Camino Mínimo
- Problema de la Mochila
- El Problema del Viajante

Programación Dinámica

Algoritmos

El Problema de la Mochila

1. Naturaleza n-etápica: El Problema de la Mochila

- El Problema de la Mochila es un ejemplo clásico de problema n-etápico, y por tanto de PD
- En el PM su solución puede verse como el resultado de una sucesión de decisiones:
  - Tenemos que decidir los valores de x_i, $1 \leq i \leq n$.
- Así, primero tomaríamos una decisión sobre x_1, luego sobre x_2, después sobre x_3, etc.
- Una sucesión optimal de decisiones, verificando las restricciones del problema, será aquella que maximice la función objetivo.

2. Comprobación del P.O.B.: Problema de la Mochila 0-1

- Notamos M(1,j,Y) al siguiente problema,

$$\text{Max: } \sum_{1\le i\le j} p_i x_i$$
$$\text{Sujeto a: } \sum_{1\le i\le j} w_i x_i \le Y$$
$$x_i = 0, 1;\ 1\le i\le j$$

el problema de la mochila 0-1 se representa por M(1,n,M).

- Sea $y_1, y_2, \ldots, y_n$ una sucesión optimal de valores 0-1 para $x_1, x_2, \ldots, x_n$.
- Si $y_1 = 0$, entonces $y_2, \ldots, y_n$ debe ser una sucesión optimal para el problema M(2,n,M).
- Si no lo es: $y_1, y_2, \ldots, y_n$ no es una sucesión optimal de M(1,n,M).

2. Comprobación del P.O.B.: Problema de la Mochila 0-1

- Si $y_1 = 1$, entonces $y_2, \ldots, y_n$ debe ser una sucesión optimal para $M(2,n, M-w_1)$.
- Si no lo fuera, habría otra sucesión 0-1, $z_2, z_3, \ldots, z_n$ tal que

$$\sum_{2\le i\le n} w_i z_i \le M - w_1 \text{ y } \sum_{2\le i\le n} p_i z_i > \sum_{2\le i\le n} p_i y_i$$

y por tanto la sucesión $y_1, z_2, z_3, \ldots, z_n$ es una sucesión para el problema de partida con mayor valor.

- Por tanto puede aplicarse el POB

3. Construcción de una ecuación recurrente: Problema de la Mochila

- Consideremos ahora el Problema de la Mochila 0-1.
- Sea $f_j(y)$ el valor de una solución optimal del problema Mochila (j+1, n, y).
- Claramente $f_0(M)$ es el valor de una solución optimal de Mochila (1,n,M).
- Las posibles decisiones para x_1 son 0 o 1 ($D_1 = \{0,1\}$).
- A partir del POB se sigue que

$$f_0(M) = \text{Max}\{f_1(M), f_1(M-w_1) + p_1\}$$

Existen muchisimos algoritmos para resolver este problema, pero es NP completo

Los algoritmos conocidos que lo resuelven son exponenciales. No se conocen algoritmos polinomiales que lo resuelvan

Un enfoque de solución para el Problema de la Mochila

- Podemos obtener una solución para el problema de la mochila tomando una sucesión de decisiones sobre las variables $x_1, \ldots, x_n$.
- Una decisión sobre la variable x_i supone decidir que valor (0 o 1) ha de tomar.
- Supongamos que las decisiones sobre las x_i se hacen en el orden $x_n, \ldots, x_1$.
- Tras una decisión sobre x_n nos podemos encontrar ante una de estas dos posibilidades:
  - la capacidad restante de la mochila es M, y no se ha producido incremento de beneficio, o bien
  - la capacidad restante es $M - w_n$ y hemos aumentado el beneficio en p_n.

Un enfoque de solución para el Problema de la Mochila

- Sea $f_j(X)$ el valor de una solución optimal del problema Mochila(1,j,X). Como se verifica el Principio de Optimalidad, obtenemos,

$$f_n(M) = \text{Max}\{f_{n-1}(M), f_{n-1}(M-w_n) + p_n\}$$

que para i > 0 arbitrario, se generaliza a,

$$f_i(X) = \text{Max}\{f_{i-1}(X), f_{i-1}(X-w_i) + p_i\}$$

- Esta ecuación puede resolverse para $f_n(M)$ teniendo en cuenta que $f_0(X) = 0$ para todo X, y que $f_i(x) = \infty$, $x < 0$. Entonces se pueden calcular sucesivamente $f_1, f_2, \ldots, f_n$.

- Ejemplo:

  n=3 C=15

  (b_1,b_2,b_3)=(38,40,24)

  (p_1,p_2,p_3)=(9,6,5)

- Recordar la estrategia voraz:
  - Tomar siempre el objeto que proporcione mayor beneficio por unidad de peso.
  - Se obtiene la solución:

    (x_1,x_2,x_3)=(0,1,1), con beneficio 64
  - Sin embargo, la solución óptima es:

    (x_1,x_2,x_3)=(1,1,0), con beneficio 78

- Por tanto, la estrategia voraz no calcula la solución óptima del problema de la mochila 0-1.

- Para evitar la repetición de cálculos, las soluciones de los subproblemas se deben almacenan en una tabla.
  - Matriz $n \times C$ cuyo elemento (j,c) almacena $\bar{g}_j(c)$
  - Para el ejemplo anterior:

    $n=3$ $\quad C=15$

    $(b_1,b_2,b_3)=(38,40,24)$

    $(p_1,p_2,p_3)=(9,6,5)$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$p_1 = 9$	0	0	0	0	0	0	0	0	0	38	38	38	38	38	38	38
$p_2 = 6$	0	0	0	0	0	0	40	40	40	40	40	40	40	40	40	78
$p_3 = 5$	0	0	0	0	0	24	40	40	40	40	40	64	64	64	64	78

$$\bar{g}_j(c) = \max\left\{\bar{g}_{j-1}(c),\ \bar{g}_{j-1}(c-p_j)+b_j\right\}$$

Programación Dinámica

Algoritmos

El Problema del Camino Mínimo

Algoritmo de Floyd

1. Naturaleza n-etápica: El Problema del Camino Mínimo

- El Problema del Camino Mínimo es otro ejemplo clásico de problema de PD
- En este caso, para encontrar ese camino desde un vértice i a otro j en un grafo G, veríamos que vértice debe ser el segundo, cual el tercero, etc. hasta alcanzar el j.
- Una sucesión optimal de decisiones proporcionará entonces el camino de longitud mínima.

Por tanto, ambos problemas tienen una clara naturaleza n-etápica

2. Comprobación del P.O.B.: Problema del Camino mínimo

- Sea $i, i_1, \ldots i_k, j$ el camino mínimo desde i hasta j.
- Comenzando con el vértice inicial i, se ha tomado la decisión de ir al vértice i_1.
- Como resultado, ahora el estado del problema esta definido por el vértice i_1, y lo que se necesita es encontrar un camino desde i_1 hasta j.
- Está claro que la sucesión $i_1, i_2, \ldots i_k, j$ debe constituir un camino mínimo entre i_1 y j. Si no:
- Sea $i_1, r_1, \ldots r_q, j$ un camino más corto entre i_1 y j
- Entonces $i, i_1, r_1, r_2, \ldots r_q, j$ es un camino entre i y j que es más corto que el camino $i, i_1, i_2, \ldots i_k, j$.
- Por tanto el POB también puede aplicarse a este problema.

3. Construcción de una ecuación recurrente: Caminos Mínimos

- Sea A_i el conjunto de los vértices adyacentes al vértice i.
- Para cada vértice $k \in A_i$ sea Γ_k el camino mínimo desde k hasta j.
- Entonces el camino más corto desde i hasta j es el más corto de los caminos del conjunto $\{i, \Gamma_k / k \in A_i\}$
- La recurrencia es trivial en este caso:

$$D_k(i,j) = \text{Min}\{D_{k-1}(i,j), D_{k-1}(i,k) + D_{k-1}(k,j)\}$$

4. Caminos mínimos: Algoritmo de Floyd

- Sea $G = (N,A)$ un grafo dirigido en el que N es su conjunto de nodos y A el de sus arcos. Cada arco tiene asociada una longitud, no negativa.
- El problema consiste en determinar el camino de longitud mínima que una **cualquier** par de nodos del grafo.
- Supondremos que los nodos están numerados de 1 a n, $N = \{1,2,\ldots,n\}$ y que la matriz L da la longitud de cada arco, de modo que $L(i,i) = 0$, $L(i,j) \geq 0$ si i es distinto de j, y $L(i,j) = \infty$ si no existe el arco (i,j).
- El POB se aplica del siguiente modo:
  - Si k es un nodo en el camino mínimo que une i con j, entonces la parte de ese camino que va de i hasta k, y la del que va de k hasta j, es también optimal.

4. Caminos mínimos: Algoritmo de Floyd

- El Algoritmo consiste en lo siguiente.
- Construimos una matriz D que da la longitud del camino mínimo entre cada par de nodos.
- El algoritmo comienza asignando a D, L y, entonces, realiza n iteraciones.
- Tras la iteración k, D da la longitud de los caminos mínimos que solo usan como nodos intermedios los del conjunto {1,2,...,k}.
- Después de n iteraciones tendremos, por tanto la solución buscada.

4. Caminos mínimos: Algoritmo de Floyd

- En la iteración k, el algoritmo tiene que chequear, para cada par de nodos (i,j), si existe o no un camino que pase a través de k que sea mejor que el actual camino minimal que solo pasa a través de los nodos {1,2,...,k-1}.
- Sea D la matriz después de la k-ésima iteración. El chequeo puede expresarse como,

$$D_k(i,j) = \text{Min}\{D_{k-1}(i,j), D_{k-1}(i,k) + D_{k-1}(k,j)\}$$

- donde hemos hecho uso del POB para calcular la longitud del camino más corto que pasa a través de k.

4. Caminos mínimos: Algoritmo de Floyd

```
Procedimiento Floyd
Begin
        For i := 1 to n do
            For j := 1 to n do
                D[i,j] := L[i,j];
        For i := 1 to n do
                D[i,i] := 0;
        For k := 1 to n do
          For i := 1 to n do
              For j := 1 to n do
                        If D[i,k] + D[k,j] < D[i,j]
                        Then D[i,j] := D[i,k] + D[k,j]
 End;
```

4. Caminos mínimos: Algoritmo de Floyd

- El algoritmo consume un tiempo $O(n^3)$.
- También podemos usar el algoritmo de Dijkstra, entonces aplicaríamos ese algoritmo n veces, eligiendo un nodo diferente como origen cada vez.
- Si queremos usar la versión de Dijkstra que trabaja con una matriz de distancias, el tiempo de cálculo total está en n x $O(n^2)$, es decir en $O(n^3)$.
- El orden es el mismo que para el algoritmo de Floyd, pero la simplicidad de este supone que, en la práctica, probablemente sea más rápido.

4. Caminos mínimos: Algoritmo de Floyd

- Si queremos saber por donde va el camino más corto

```
Procedimiento Floyd-Warshall
Begin
    For i := 1 to n do
          For j := 1 to n do begin
              D[i,j] := L[i,j];
              P[i,j] := 0
          End;
    For i := 1 to n do
          D[i,i] := 0;
    For k := 1 to n do
        For i := 1 to n do
          For j := 1 to n do
              If D[i,k] + D[k,j] < D[i,j] then begin
                  D[i,j] := D[i,k] + D[k,j];
                 P[i,j] := k
              End
End;
```

```
Procedimiento Camino
Begin
   k := P[i,j];
   If k = 0 then Return;
   Path (i,k);
   Writeln (k);
   Path (k,j)
End;
```

Programación Dinámica

Algoritmos

El Problema del Viajante de Comercio

El Problema del Viajante de Comercio

- Dado un grafo con longitudes no negativas asociadas a sus arcos, queremos encontrar el circuito más corto posible que comience y termine en un mismo nodo, es decir, un camino cerrado que recorra todos los nodos una y solo una vez y que tenga longitud minimal (el circuito hamiltoniano minimal)
- Sea $G = (N,A)$ un grafo dirigido, $N = \{1,2,\ldots,n\}$, y L_{ij} la matriz de distancia,
  - $L(i,i) = 0$,
  - $L(i,j) \geq 0$ si i es distinto de j, y
  - $L(i,j) = \infty$ si no existe el arco (i,j).

El Problema del Viajante de Comercio

- Suponemos, sin pérdida de generalidad, que el circuito comienza y termina en el nodo 1. El problema es claramente n-etápico
- El circuito, está constituido por un arco (1, j), seguido de un camino de j a 1 que pasa exactamente una vez a través de cada nodo de N - {1, ,j}.
- Si el circuito es optimal, es decir, de longitud mínima, entonces ese es el camino de j a 1 y vale el principio de Optimalidad.

El Problema del Viajante de Comercio

- Sea $S \subseteq N - \{1\}$ un conjunto de nodos y consideremos un nodo más $i \in N - S$
- Está permitido que $i = 1$ solo si $S = N - \{1\}$.
- Definimos el valor $g(i,S)$ para cada índice i, como la longitud del camino más corto desde el nodo i al nodo 1 que pasa exactamente una vez a través de cada nodo de S.
- Usando esta definición,

$$g(1, N - \{1\})$$

- es la longitud de un circuito optimal.

El Problema del Viajante de Comercio

- Por el POB vemos que

$$g(1, N-\{1\}) = \text{Min}_{2 \le j \le n} [L_{1j} + g(j, N - \{1,j\})]$$

- Mas generalmente, si i no es igual a 1, el conjunto S no es vacío y además es distinto de N - {1} e $i \notin S$,

$$g(i,S) = \text{Min}_{j \in S} [L_{ij} + g(j, S - \{j\})]$$

- Además,
- $g(i,\varnothing) = L_{i1}$, i = 2,3,...n
- Por tanto, los valores de g(i,S) se conocen cuando S es vacío

El Problema del Viajante de Comercio: Solución operativa

- Podemos aplicar

$$g(i,S) = \text{Min}_{j \in S} [L_{ij} + g(j, S - \{j\})]$$

para calcular g en todos los conjuntos S que contienen exactamente un nodo (que no es el 1).

- Luego aplicamos la misma fórmula para calcular g en todos los conjuntos S que contienen dos nodos (distintos del 1), y así sucesivamente.
- Cuando se conoce el valor de $g[j,N - \{1,j\}]$ para todos los nodos j, excepto para 1, utilizamos

$$g(1, N-\{1\}) = \text{Min}_{2 \le j \le n} [L_{1j} + g(j, N - \{1,,j\})]$$

- para calcular $g(1,N - \{1\})$, y definitivamente resolver el problema

El Problema del Viajante de Comercio: Tiempo de ejecución

- El tiempo que consumirá este algoritmo se hallará a partir de las expresiones anteriores.
- Para calcular $g(j,\varnothing)$ hay que hacer n-1 consultas de una tabla.
- Para calcular $g(i,S)$

$$g(i,S) = \text{Min}_{j \in S} [L_{ij} + g(j, S - \{j\})]$$

- Hay qye calcular todas las $g(i,S)$ tales que $1 \leq |S| = k \leq n-2$, lo que supone realizar,

$$(n-1) \times C_{n-2,k} \times k$$

adiciones.

De ellas,

El Problema del Viajante de Comercio: Tiempo de ejecución

- De esas $(n-1) \times C_{n-2,k} \times k$ adiciones,

  - n-1 corresponden a los posibles valores que puede tomar la variable i,
  - k provienen de los valores que puede tomar la variable j,
  - y las combinaciones restantes son todos los conjuntos que podemos formar de n-2 elementos tomados de k en k.

- Calcular g(1, N-{1}) implica n-1 adiciones.
- Estas operaciones pueden usarse como referencia para calcular la eficiencia del algoritmo., y así el tiempo que se lleva el algoritmo en cálculos es,

$$O[2(n-1) + \sum_{k=1..(n-2)} (n-1)\times k\times C_{n-2,k}\ ] = O(n^2 2^n)$$

ya que

$$\sum_{k=1..r} k\times C_{r,k} = r2^{r-1}$$

El Problema del Viajante de Comercio: Tiempo de ejecución

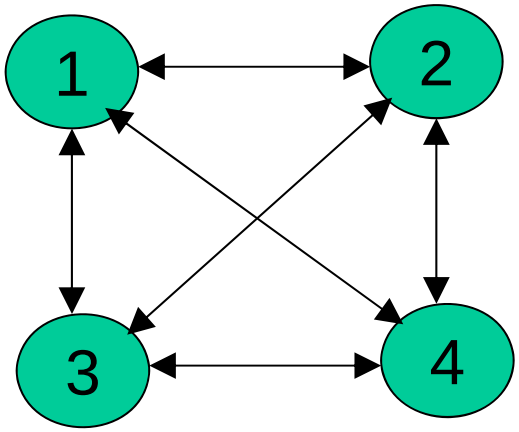
Ese tiempo es bastante considerable, pero mejor que O(n!) que es el que proporciona la fuerza bruta.

	Tiempo	Tiempo
N	Método directo	PD
	n!	$n^2 2^n$
5	120	800
10	3.628.800	102.400
15	1.31×10^{12}	7.372.800
20	2.43×10^{18}	419.430.400

Por ejemplo, $20^2 2^{20}$ microsegundos es menos de siete minutos, mientras que 20! microsegundos supera las 77 mil años.

El Problema del Viajante de Comercio: Ejemplo

- Consideremos el grafo



$$\begin{bmatrix} 0 & 10 & 15 & 20 \\ 5 & 0 & 9 & 10 \\ 6 & 13 & 0 & 12 \\ 8 & 8 & 9 & 0 \end{bmatrix}$$

$g(2, \varnothing) = c_{21} = 5$; $g(3, \varnothing) = c_{31} = 6$; $g(4, \varnothing) = c_{41} = 8$

Y luego,

$g(2,\{3\}) = c_{23} + g(3, \varnothing) = 15$; $g(2,\{4\}) = 18$

$g(3,\{2\}) = 18$; $g(3,\{4\}) = 20$

$g(4,\{2\}) = 13$; $g(4,\{3\}) = 15$

El Problema del Viajante de Comercio: Ejemplo

- Calculamos $g(i,S)$ para conjuntos S de cardinal 2, $i \neq 1$, $1 \notin S$ e $i \notin S$:

$$g(2,\{3,4\}) = \text{Min}\,[c_{23} + g(3,\{4\}),\ c_{24} + g(4,\{3\})] = 25$$

$$g(3,\{2,4\}) = \text{Min}\,[c_{32} + g(2,\{4\}),\ c_{34} + g(4,\{2\})] = 25$$

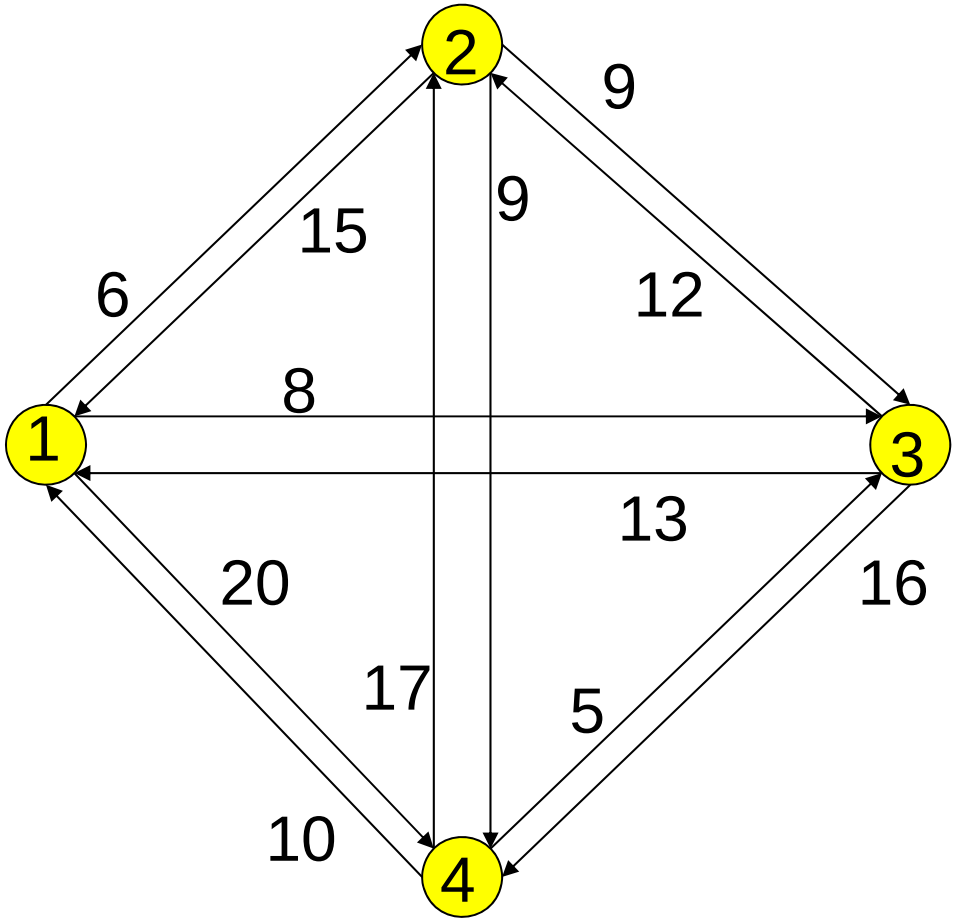
$$g(4,\{2,3\}) = \text{Min}\,[c_{42} + g(2,\{3\}),\ c_{43} + g(3,\{2\})] = 23$$

- y finalmente,

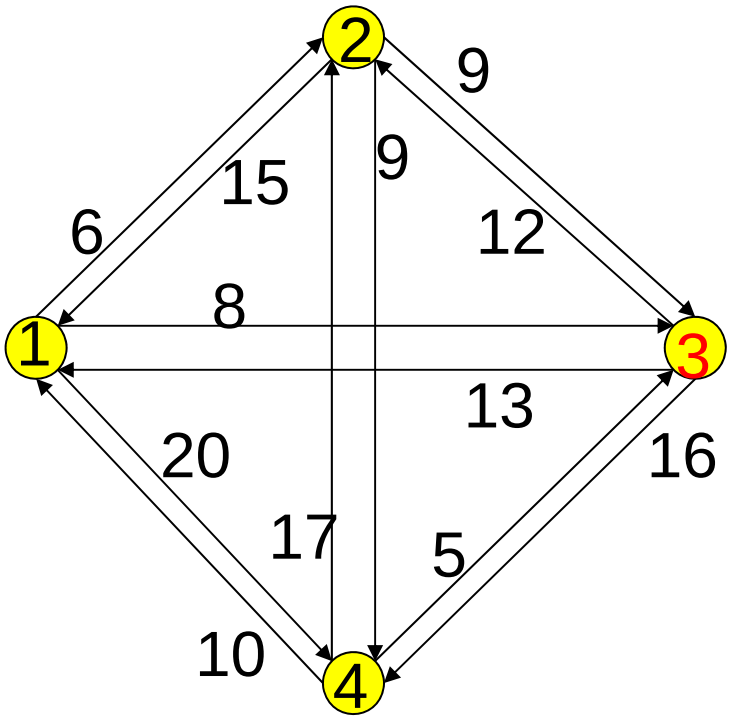
$$g(1,\{2,3,4\}) = \text{Min}\,[c_{12} + g(2,\{3,4\}),\ c_{13} + g(3,\{2,4\}),\ c_{21} + g(4,\{2,3\})] =$$
$$= \text{Min}\,\{35,40,43\} = 35$$

que es la longitud del circuito buscado.

El Problema del Viajante de Comercio: otro ejemplo



El Problema del Viajante de Comercio: otro ejemplo



Sea k = 3.

g(3,{1,2,3,4}-{3,1}) es el camino más corto de 3 a 1 que pasa por 2 y 4

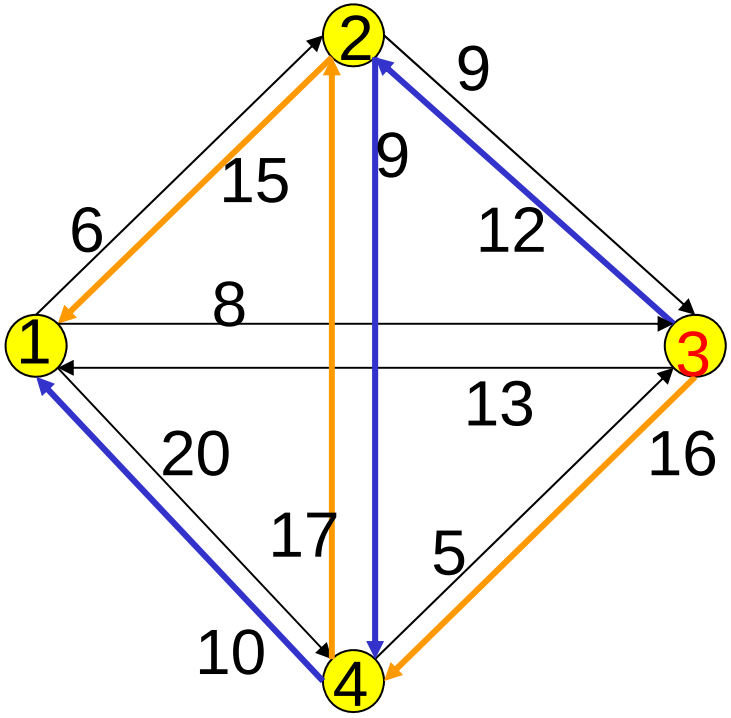
Y cual es ese camino?

Simplemente es el más corto de los posibles:

$$c_{3i}+g(i,V-\{3,1,i\})$$

con i = 2 y 4.

El Problema del Viajante de Comercio: otro ejemplo



Así,

$c_{32} + g(2,\{4\}) =$

$= 12 + (9+10)$

$= 31$

o

$c_{34} + g(4,\{2\}) =$

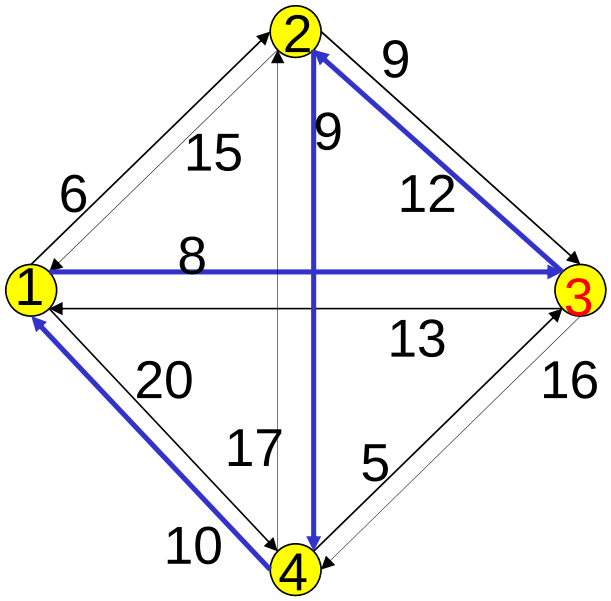
$= 16 + (17+15)$

$= 48$

Luego

$g(3,\{2,4\}) = 31.$

El Problema del Viajante de Comercio: otro ejemplo



Así $g(1,\{1,2,3,4\}-1)$ podría ser

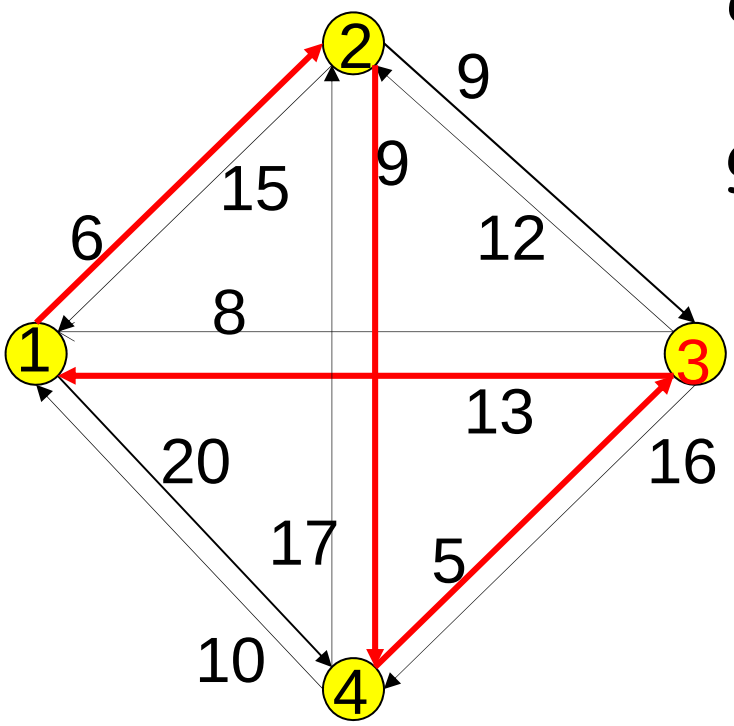
$c_{13} + g(3,\{2,4\}) =$

$= 8 + 31$

$= 39.$

Pero para resolver el problema necesitamos encontrar tambien $g(2,\{3,4\})$ y $g(4,\{2,3\})$ y escoger el menor.

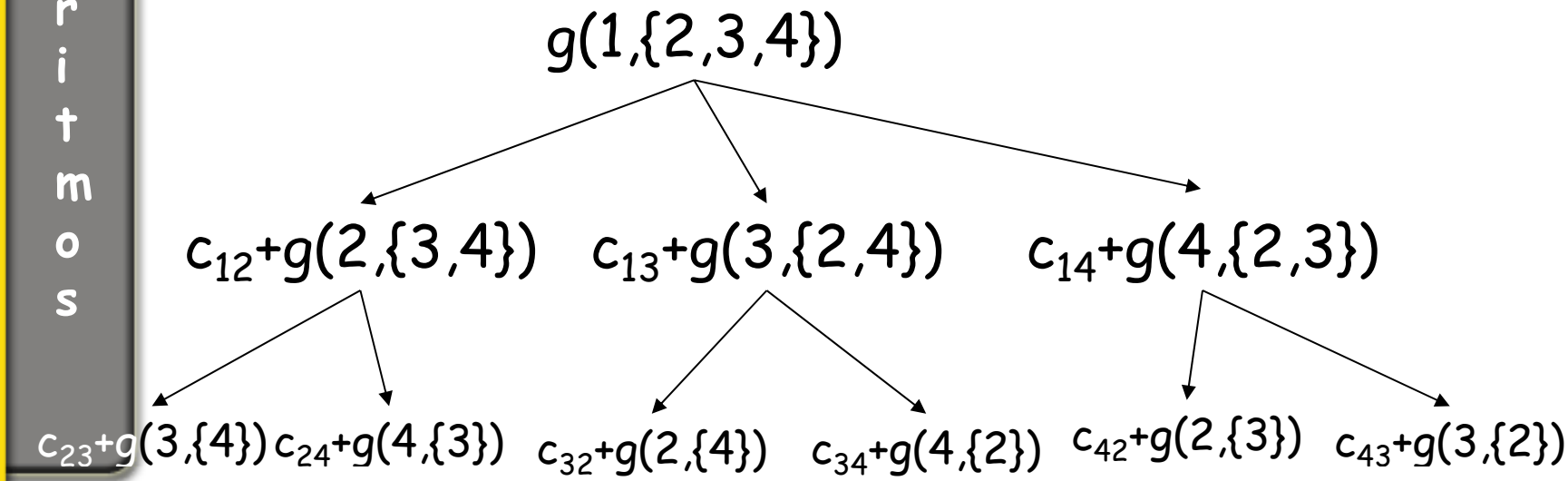
El Problema del Viajante de Comercio: otro ejemplo



Definitivamente, la solución es:

$$g(1,\{2,3,4\}) = c_{12} + g(2,\{3,4\})$$
$$= 6 + 27$$
$$= 33.$$

El Problema del Viajante de Comercio: otro ejemplo



Un ejemplo de inversiones

Un ejemplo de inversiones

Se quieren asignar 12 personas a 3 tareas. La asignación de m personas a la tarea i produce $f_i(m)$ según la siguiente tabla,

m	$f_1(m)$	$f_2(m)$	$f_3(m)$
0	0	0	0
1	3	2	2
2	5	4	3
3	6	5	5
4	7	7	6
5 ó más	7	8	6

¿Como debería realizarse la asignación para que lo producido sea máximo?.

Un ejemplo de inversiones

Tenemos que,

$F_3(12) = \text{Max}_{y \leq 12} \{f_3(y) + F_2(12-y)\}$

$F_2(n) = \text{Max}_{y \leq n} \{f_2(y) + f_1(n-y)\}$

donde n puede valer 12, 11, 10, etc.

Así, por ejemplo,

$F_2(8) = \text{Max}_{y \leq 8} \{2+7, 4+7, 5+8, 7+7, 8+6, 7+5, \ldots\}$

Con lo que,

N	12	11	10	9	8	7	6
$F_2(n)$	15	15	15	15	14	13	12

Y

$F_3(12) = \text{Max} \{15, 15+2, 15+3, 15+5, 14+6, 13+6, \ldots\} = 20$

Así la solución optimal se alcanza al asignar 4,5,3 o 3,5,4 o 4,4,4; produciéndose en todos los casos un beneficio de 20.