

Teoría de Algoritmos

Capítulo 4: Programación Dinámica

Tema 11: Algoritmos basados en P.D.

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

Aplicación de la P.D. al Diseño de Algoritmos

- PD se aplica en cuatro fases
 1. Naturaleza n-etápica del problema
 2. Verificación del POB
 3. Planteamiento de una recurrencia
 4. Cálculo de la solución (enfoque adelantado o atrasado)
- Las desarrollaremos sobre
 - Problema del Camino Mínimo
 - Problema de la Mochila
 - El Problema del Viajante

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

- El Problema de la Mochila es un ejemplo clásico de problema n-etapico, 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.

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 de la Mochila 0-1

- Notamos $M(l,j,Y)$ al siguiente problema,

$$\begin{aligned} \text{Max: } & \sum_{1 \leq i \leq j} p_i x_i \\ \text{Sujeto a: } & \sum_{1 \leq i \leq j} w_i x_i \leq Y \\ & x_i = 0, 1; 1 \leq i \leq j \end{aligned}$$

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

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

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

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

- Por tanto puede aplicarse el POB

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

- Sea i, i_1, \dots, 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 .
- Esta claro que la sucesión i_1, i_2, \dots, i_k, j debe constituir un camino mínimo entre i_1 y j . Si no:
- Sea i_1, r_1, \dots, r_q, j un camino mas corto entre i_1 y j
- Entonces $i, i_1, r_1, r_2, \dots, r_q, j$ es un camino entre i y j que es mas corto que el camino $i, i_1, i_2, \dots, i_k, j$.
- Por tanto el POB tambien puede aplicarse a este problema.

3. Construcción de una ecuación recurrente: Caminos Minimizados

- 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 mas corto desde i hasta j es el mas corto de los caminos del conjunto $\{i, \Gamma_k / k \in A_i\}$
- La recurrencia es trivial en este caso:

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

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

- Consideremos ahora el Problema de la Mochila 0-1.
- Sea $g_j(y)$ el valor de una solución óptima del problema Mochila $(j+1, n, y)$.
- Claramente $g_0(M)$ es el valor de una solución óptima 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
$$g_0(M) = \text{Max} \{g_1(M), g_1(M-w_1) + p_1\}$$

Existen muchísimos 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

4. Caminos minimos: 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 estan numerados de 1 a n , $N = \{1,2,...,n\}$ 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 $\{1, 2, \dots, k\}$.
- Después de n iteraciones tendremos, por tanto la solución buscada.

4. Caminos minimos: 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 mas corto que pasa a través de k .

4. Caminos minimos: 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 minimos: 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 calculo total esta en $n \times O(n^2)$, 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 practica, probablemente sea mas rápido.

4. Caminos minimos: Algoritmo de Floyd

- Si queremos saber por donde va el camino mas 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;

Camino (i,k);

Writeln (k);

Camino (k,j)

End;

El Problema del Viajante de Comercio

- Dado un grafo con longitudes no negativas asociadas a sus arcos, queremos encontrar el circuito mas 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, $N = \{1, 2, \dots, n\}$, 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 mas $i \in N - S$
- 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 mas 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

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

- 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,
- $g(i, \emptyset) = L_{i1}$, $i = 2, 3, \dots, n$
- 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 formula 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
$$g(1, N - \{1\}) = \text{Min}_{2 \leq j \leq n} [L_{1j} + g(j, N - \{1, j\})]$$
- 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 $g(j, \emptyset)$ 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

$$\sum_{k=1..r} k \times C_{r,k} = r 2^{r-1}$$

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.