capítulo

7

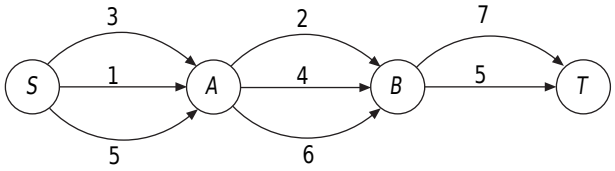
Programación dinámica

La estrategia de programación dinámica es una técnica muy útil para resolver muchos problemas de optimización combinatorios. Antes de presentar el método de programación dinámica, quizá convenga reconsiderar un caso típico donde puede aplicarse el método codicioso (*greedy*). Luego se mostrará un caso para el que el método codicioso no funciona, mientras que el método de programación dinámica sí lo hace.

Se considerará la figura 7-1, que muestra una gráfica de multietapas. Suponga que se quiere encontrar la ruta más corta de *S* a *T*. En este caso, la ruta más corta puede encontrarse mediante la siguiente línea de razonamiento:

1. Debido a que se sabe que la ruta más corta debe pasar por el vértice *A*, es necesario encontrar una ruta más corta de *S* a *A*. El costo de esta ruta es 1.
2. Debido a que se sabe que la ruta más corta debe pasar por *B*, se encuentra una ruta más corta de *A* a *B*. El costo de esta ruta es 2.
3. De manera semejante, se encuentra una ruta más corta de *B* a *T* cuyo costo es 5.

FIGURA 7-1 Un caso en que funciona el método codicioso.



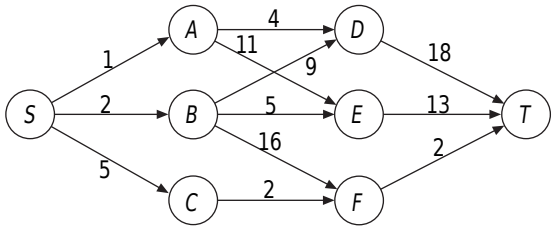
Así, el costo total de la ruta más corta de *S* a *T* es $1 + 2 + 5 = 8$. Además, este problema de ruta más corta en realidad se resolvió aplicando un método codicioso.

¿Por qué es posible usar el método codicioso para resolver ese problema? Esto es posible porque definitivamente se sabe que la solución debe constar de una subruta de *S* a *A*, de una subruta de *A* a *B*, etc. En consecuencia, nuestra estrategia para resolver el

problema primero encuentra una subruta más corta de S a A, luego una subruta más corta de A a B y así sucesivamente.

A continuación se presentará un caso en que no funciona el método codicioso. Considere la figura 7-2. De nuevo, se quiere encontrar una ruta más corta de S a T. Sin embargo, esta vez no se sabe por cuál de los vértices A, B y C debe pasar la ruta más corta. Si para resolver este problema se usa el método codicioso, se escoge el vértice A porque el costo asociado con el borde de S a A es el menor. Después de que se escoge A, se elige D. El costo de esta ruta es $1 + 4 + 18 = 23$. Ésta no es la ruta más corta porque la ruta más corta es $S \rightarrow C \rightarrow F \rightarrow T$, cuyo costo total es $5 + 2 + 2 = 9$.

FIGURA 7-2 Un caso en que no funciona el método codicioso.

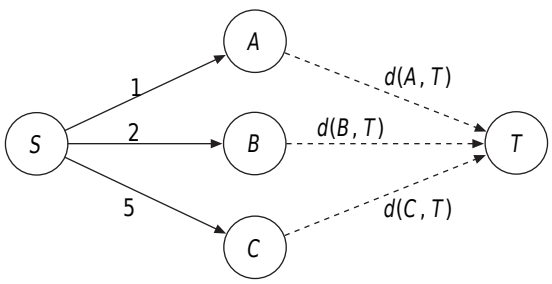


$$S \xrightarrow{5} C \xrightarrow{2} F \xrightarrow{2} T$$

Como ya se indicó, debe seleccionarse C, en vez de A. El método de programación dinámica produce esta solución mediante la siguiente línea de razonamiento.

1. Se sabe que es necesario empezar en S y pasar por A, B o C. Esto se ilustra en la figura 7-3.

FIGURA 7-3 Un paso en el proceso del método de programación dinámica.



En consecuencia, la longitud de la ruta más corta de S a T se determina con la fórmula siguiente:

$$d(S, T) = \text{mín}\{1 + d(A, T), 2 + d(B, T), 5 + d(C, T)\} \quad \textbf{(7.1)}$$

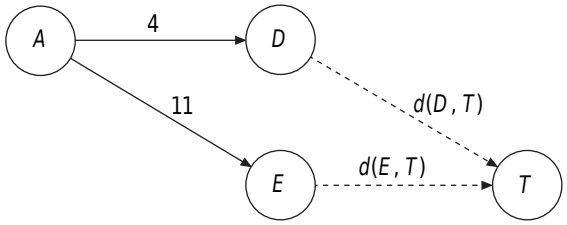
donde $d(X, T)$ denota la longitud de la ruta más corta de X a T. La ruta más corta de S a T se conoce tan pronto como se encuentran las rutas más cortas de A, B y C a T, respectivamente.

2. Para encontrar las rutas más cortas de A, B y C a T, de nuevo puede usarse el método anterior. Para el vértice A, se tiene una subgráfica que se muestra en la figura 7-4. Es decir, $d(A, T) = \text{mín}\{4 + d(D, T), 11 + d(E, T)\}$. Debido a que $d(D, T) = 18$ y $d(E, T)$ 13, se tiene

$$\begin{aligned} d(A, T) &= \text{mín}\{4 + 18, 11 + 13\} \\ &= \text{mín}\{22, 24\} \\ &= 22. \end{aligned}$$

Por supuesto, es necesario registrar el hecho de que esta ruta más corta va de A a D y luego a T.

FIGURA 7-4 Un paso en el proceso del método de programación dinámica.

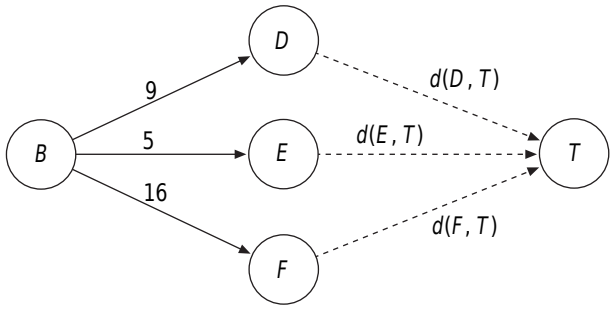


De manera semejante, para B, el paso se muestra en la figura 7-5.

$$\begin{aligned} d(B, T) &= \text{mín}\{9 + d(D, T), 5 + d(E, T), 16 + d(F, T)\} \\ &= \text{mín}\{9 + 18, 5 + 13, 16 + 2\} \\ &= \text{mín}\{27, 18, 18\} \\ &= 18. \end{aligned}$$

Por último, fácilmente se encuentra que $d(C, T)$ es 4.

FIGURA 7-5 Un paso en el proceso del método de programación dinámica.



3. Una vez que se han encontrado $d(A, T)$, $d(B, T)$ y $d(C, T)$ es posible encontrar $d(S, T)$ usando la fórmula (7.1).

$$\begin{aligned} d(S, T) &= \text{mín}\{1 + 22, 2 + 18, 5 + 4\} \\ &= \text{mín}\{23, 20, 9\} \\ &= 9. \end{aligned}$$

El principio fundamental de la programación dinámica consiste en descomponer un problema en subproblemas, cada uno de los cuales se resuelve aplicando de manera recurrente el mismo método. El método puede resumirse como sigue:

1. Para encontrar la ruta más corta de S a T, se encuentran las rutas más cortas de A, B y C a T.
2. *a*) Para encontrar la ruta más corta de A a T, se encuentran las rutas más cortas de D y E a T.
   b) Para encontrar la ruta más corta de B a T, se encuentran las rutas más cortas de D, E y F a T.
   c) Para encontrar la ruta más corta de C a T, se encuentra la ruta más corta de F a T.
3. Las rutas más cortas de D, E y F a T pueden encontrarse fácilmente.
4. Una vez que se han encontrado las rutas más cortas de D, E y F a T, es posible encontrar las rutas más cortas de A, B y C a T.
5. Una vez que se han encontrado las rutas más cortas de A, B y C a T, es posible encontrar la ruta más corta de S a T.

La línea de razonamiento anterior es realmente un *razonamiento hacia atrás* (*backward reasoning*). A continuación se mostrará que el problema también puede resolverse razonando hacia atrás. La ruta más corta se encontrará como sigue:

1. Primero se encuentran $d(S, A)$, $d(S, B)$ y $d(S, C)$.

$$\begin{aligned} d(S, A) &= 1 \\ d(S, B) &= 2 \\ d(S, C) &= 5. \end{aligned}$$

2. Luego se determinan $d(S, D)$, $d(S, E)$ y $d(S, F)$ como sigue:

$$\begin{aligned} d(S, D) &= \text{mín}\{d(A, D) + d(S, A), d(B, D) + d(S, B)\} \\ &= \text{mín}\{4 + 1, 9 + 2\} \\ &= \text{mín}\{5, 11\} \\ &= 5 \\ d(S, E) &= \text{mín}\{d(A, E) + d(S, A), d(B, E) + d(S, B)\} \\ &= \text{mín}\{11 + 1, 5 + 2\} \\ &= \text{mín}\{12, 7\} \\ &= 7 \\ d(S, F) &= \text{mín}\{d(B, F) + d(S, B), d(C, F) + d(S, C)\} \\ &= \text{mín}\{16 + 2, 2 + 5\} \\ &= \text{mín}\{18, 7\} \\ &= 7. \end{aligned}$$

3. Ahora es posible determinar la distancia más corta de S a T como sigue:

$$\begin{aligned} d(S, T) &= \text{mín}\{d(D, T) + d(S, D), d(E, T) + d(S, E), d(F, T) + d(S, F)\} \\ &= \text{mín}\{18 + 5, 13 + 7, 2 + 7\} \\ &= \text{mín}\{23, 20, 9\} \\ &= 9. \end{aligned}$$

El hecho de que el método de programación dinámica ahorra cálculos se explica al considerar nuevamente la figura 7-2. En esta figura, hay una solución como sigue:

$$S \rightarrow B \rightarrow D \rightarrow T.$$

La longitud de esta solución; a saber,

$$d(S, B) + d(B, D) + d(D, T)$$

jamás se calcula si se aplica el método de programación dinámica con razonamiento hacia atrás. Al aplicar el método con razonamiento hacia atrás, se encuentra

$$d(B, E) + d(E, T) < d(B, D) + d(D, T).$$

Es decir, no es necesario considerar la solución:

$$S \rightarrow B \rightarrow D \rightarrow T$$

porque se sabe que la longitud de

$$B \rightarrow E \rightarrow T$$

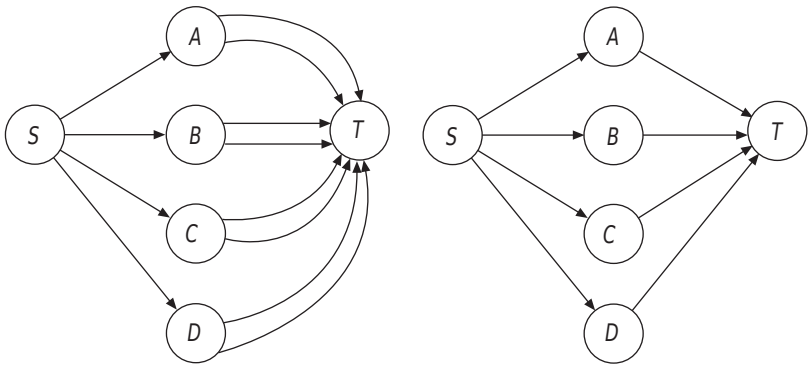
es menor que la longitud de

$$B \rightarrow D \rightarrow T.$$

Así, el método de programación dinámica, como el método *branch-and-bound*, ayuda a evitar una búsqueda exhaustiva en el espacio solución.

Puede afirmarse que el método de programación dinámica es un método de eliminación por etapas porque después de que se considera una etapa, se eliminan muchas subsoluciones. Por ejemplo, considere la figura 7-6*a*). Originalmente hay ocho soluciones. Si se usa el método de programación dinámica, el número de soluciones se reduce a cuatro, que se muestran en la figura 7-6*b*).

FIGURA 7-6 Ejemplo que ilustra la eliminación de la solución en el método de programación dinámica.



a) *b*)

La programación dinámica se basa en un concepto denominado principio de optimalidad. Suponga que al resolver un problema se requiere hacer una serie de decisiones $D_1, D_2, \ldots, D_n$. *Si esta serie es óptima, entonces las k últimas decisiones,* $1 \leq k \leq n$, *deben ser óptimas*. Este principio se ilustrará varias veces en las siguientes secciones.

Al aplicar el método de programación dinámica se tienen dos ventajas. La primera, como ya se explicó, es que pueden eliminarse soluciones y también ahorrarse cálculos. La otra ventaja de la programación dinámica es que ayuda a resolver el problema etapa por etapa de manera sistemática.

Si alguien que resuelve un problema no conoce en absoluto la programación dinámica, quizá tenga que resolver el problema examinando todas las posibles soluciones combinatorias, lo cual puede consumir tiempo excesivamente. Si se aplica el método de programación dinámica, el problema puede repentinamente convertirse en un problema de multietapas, con lo cual puede resolverse de manera muy sistemática. Esto se clarificará después.

7-1 EL PROBLEMA DE ASIGNACIÓN DE RECURSOS

Este problema se define como sigue: Se tienen m recursos y n proyectos. Se obtiene una ganancia $P(i, j)$ si al proyecto i se asignan j, $0 \leq j \leq m$ recursos. El problema consiste en encontrar una asignación de recursos que maximice la ganancia total.

Suponga que hay cuatro proyectos y tres recursos, y que la matriz de ganancias P es la que se muestra en la tabla 7-1 con $P(i, 0) = 0$ para toda i.

TABLA 7-1 Matriz de ganancias.

Proyecto \ Recurso	1	2	3
1	2	8	9
2	5	6	7
3	4	4	4
4	2	4	5

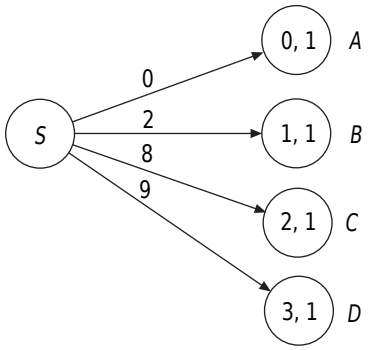
Para resolver este problema se toma una serie de decisiones. En cada decisión, se determina el número de recursos a asignar al proyecto i. Es decir que, sin pérdida de generalidad, es necesario decidir lo siguiente:

1. ¿Cuántos recursos deben asignarse al proyecto 1?
2. ¿Cuántos recursos deben asignarse al proyecto 2?

⋮

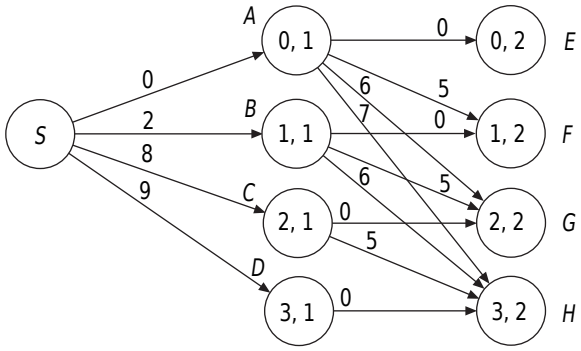
Resulta evidente que el enfoque del método codicioso no funciona en este caso. Sin embargo, es posible resolver el problema usando el método de programación dinámica. Si se aplica este método, es posible suponer que cada decisión origina un nuevo estado. Sea (i, j) el estado alcanzado donde se han asignado i recursos a los proyectos $1, 2, \ldots, j$. Así, inicialmente, sólo se tienen cuatro estados descritos, que se muestran en la figura 7-7.

FIGURA 7-7 Decisiones de la primera etapa de un problema de asignación de recursos.



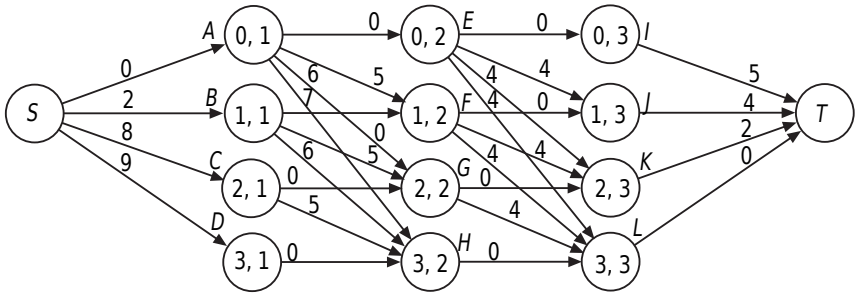
Después de haber asignado i recursos al proyecto 1, cuando mucho es posible asignar $(3 - i)$ recursos al proyecto 2. Así, las decisiones de la segunda etapa están relacionadas con las decisiones de la primera etapa, que se muestran en la figura 7-8.

FIGURA 7-8 Decisiones de las dos primeras etapas de un problema de asignación de recursos.



Finalmente, en la figura 7-9 se muestra cómo es posible describir todo el problema.

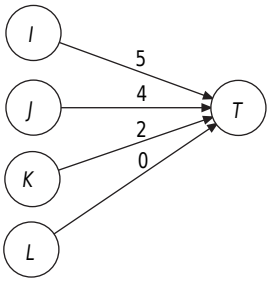
FIGURA 7-9 El problema de asignación de recursos descrito como una gráfica multietapas.



Para resolver el problema de asignación de recursos, basta encontrar la ruta más larga de S a T. El método de razonamiento hacia atrás puede aplicarse como sigue:

1. Las rutas más largas de I, J, K y L a T se muestran en la figura 7-10.

FIGURA 7-10 Las rutas más largas de I, J, K y L a T.

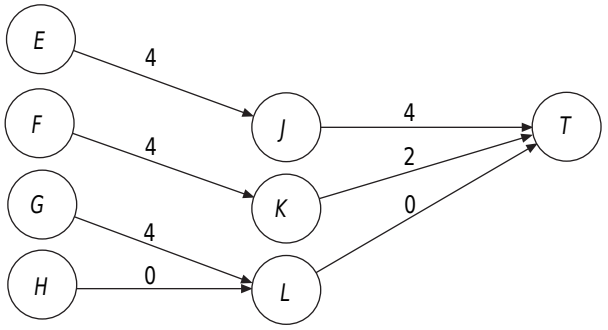


2. Una vez que se han obtenido las rutas más largas de I, J, K y L a T, las rutas más largas de E, F, G y H a T pueden obtenerse fácilmente. Por ejemplo, la ruta más larga de E a T se determina como sigue:

$$\begin{aligned} d(E,T) &= \text{máx}\{d(E,I) + d(I,T), d(E,J) + d(J,T), \\ &\qquad d(E,K) + d(K,T), d(E,L) + d(L,T)\} \\ &= \text{máx}\{0 + 5, 4 + 4, 4 + 2, 4 + 0\} \\ &= \text{máx}\{5, 8, 6, 4\} \\ &= 8. \end{aligned}$$

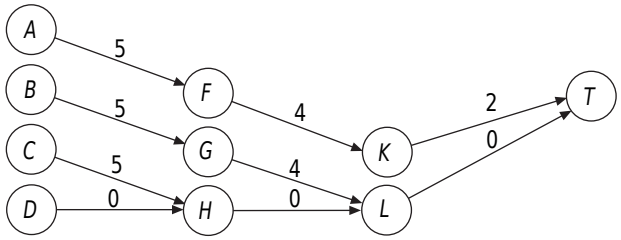
Los resultados se resumen en la figura 7-11.

FIGURA 7-11 Las rutas más largas de E, F, G y H a T.



3. Las rutas más largas de A, B, C y D a T, respectivamente, pueden obtenerse fácilmente aplicando el mismo método y se muestran en la figura 7-12.

FIGURA 7-12 Las rutas más largas de A, B, C y D a T.



4. Finalmente, la ruta más larga de S a T se obtiene como sigue:

$$\begin{aligned} d(S,T) &= \text{máx}\{d(S,A) + d(A,T), d(S,B) + d(B,T), \\ &\qquad d(S,C) + d(C,T), d(S,D) + d(D,T)\} \\ &= \text{máx}\{0 + 11, 2 + 9, 8 + 5, 9 + 0\} \\ &= \text{máx}\{11, 11, 13, 9\} \\ &= 13. \end{aligned}$$

La ruta más larga es

$$S \to C \to H \to L \to T.$$

Lo anterior corresponde a

2 recursos asignados al proyecto 1,
1 recurso asignado al proyecto 2,
0 recursos asignados al proyecto 3,
y 0 recursos asignados al proyecto 4,

7-2 EL PROBLEMA DE LA SUBSECUENCIA COMÚN MÁS LARGA

Considere una cadena $A = a\ b\ a\ a\ d\ e$. Una subsecuencia A (también llamada subcadena) se obtiene eliminando 0 o más símbolos (no necesariamente consecutivos) de A. Por ejemplo, todas las siguientes cadenas son subsecuencias de A:

a
b
d
a b
a a
b d
a e
a b a
a a a
a d e
b a d e
a a a d
a b a d e

Una subsecuencia común entre A y B se define como una subsecuencia de ambas cadenas. Por ejemplo, considere $A = a\ b\ a\ a\ d\ e\ c$ y $B = c\ a\ a\ c\ e\ d\ c$. Todas las siguientes cadenas son subsecuencias comunes de A y B:

a
d
c
a d
d c
a a
a c
a a c
a a d
a a e c

El problema de la subsecuencia común más larga consiste en encontrar una subsecuencia común más larga entre dos cadenas. Muchos problemas son variantes del problema de la subsecuencia común más larga. Por ejemplo, el problema de identificación de lenguaje puede considerarse como un problema de la subsecuencia común más larga.

Para un novato en la solución de problemas, el problema de la subsecuencia común más larga de ninguna manera es fácil de resolver. Un método directo para resolverlo consiste en encontrar todas las subsecuencias comunes mediante una búsqueda exhaustiva. Por supuesto, debe empezarse con la más larga posible. La longitud de la subsecuencia común más larga entre dos cadenas no puede ser mayor que la longitud de la cadena más corta. En consecuencia, debe empezarse con la cadena más corta.

Por ejemplo, sean $A = a\ b\ a\ a\ d\ e\ c$ y $B = b\ a\ f\ c$. Puede intentar determinarse si $b\ a\ f\ c$ es una subsecuencia común. Puede verse que no lo es. Luego se intenta con subsecuencias escogidas de B cuya longitud es tres; a saber, $a\ f\ c$, $b\ f\ c$, $b\ a\ c$ y $b\ a\ f$. Debido a que $b\ a\ c$ es una subsecuencia común, también debe ser una subsecuencia común más larga, ya que se empezó con la más larga posible.

Este método directo consume demasiado tiempo porque un gran número de subsecuencias de B deben corresponder con un gran número de subsecuencias de A. Así, se trata de un procedimiento exponencial.

Afortunadamente, para resolver este problema de la subsecuencia común más larga puede aplicarse el método de programación dinámica. A continuación se modificará ligeramente el problema original. En vez de encontrar la subsecuencia común más larga, se intentará encontrar la *longitud* de la subsecuencia común más larga. En realidad, al rastrear el procedimiento para encontrar la longitud de la subsecuencia común más larga, fácilmente puede encontrarse la subsecuencia común más larga.

Considere dos cadenas $A = a_1a_2 \ldots a_m$ y $B = b_1b_2 \ldots b_n$. Se prestará atención a los dos últimos símbolos: a_m y b_n. Hay dos posibilidades:

Caso 1: $a_m = b_n$. En este caso, la subsecuencia común más larga debe contener a a_m. Simplemente basta encontrar la subsecuencia común más larga de $a_1a_2 \ldots a_{m-1}$ y $b_1b_2 \ldots b_{n-1}$.

Caso 2: $a_m \neq b_n$. En este caso, puede hacerse corresponder $a_1a_2 \ldots a_m$ con $b_1b_2 \ldots b_{n-1}$ y también $a_1a_2 \ldots a_{m-1}$ con $b_1b_2 \ldots b_n$. Con lo se obtiene una subsecuencia común mayor, ésta será la subsecuencia común más larga.

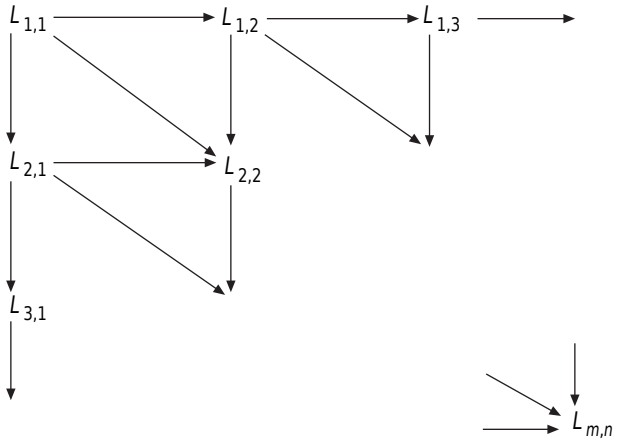
Sea $L_{i,j}$ la longitud de la subsecuencia común más larga de $a_1a_2 \ldots a_i$ con $b_1b_2 \ldots b_j$. $L_{i,j}$ puede encontrarse aplicando la siguiente fórmula recurrente:

$$L_{i,j} = \begin{cases} L_{i-1,j-1} + 1 & \text{si } a_i = b_j \\ \max\{L_{i-1,j}, L_{i,j-1}\} & \text{si } a_i \neq b_j \end{cases}$$

$$L_{0,0} = L_{0,j} = L_{i,0} = 0, \qquad \text{para } 1 \leq i \leq m \text{ y } 1 \leq j \leq n.$$

La fórmula anterior muestra que primero debe encontrarse $L_{1,1}$. Después de encontrar $L_{1,1}$ es posible encontrar $L_{1,2}$ y $L_{2,1}$. Una vez que se obtienen $L_{1,1}$, $L_{1,2}$ y $L_{2,1}$, pueden encontrarse $L_{1,3}$, $L_{2,2}$ y $L_{3,1}$. Todo el procedimiento puede representarse en la figura 7-13.

FIGURA 7-13 El método de programación dinámica para resolver el problema de la subsecuencia común más larga.



Para implementar de manera secuencial el algoritmo es posible calcular el valor de cada $L_{i,j}$ mediante la secuencia: $L_{1,1}, L_{1,2}, L_{1,3}, \ldots, L_{1,n}, L_{2,1}, \ldots, L_{2,n}, L_{3,1}, \ldots, L_{3,n}, \ldots, L_{m,n}$. Debido a que para calcular cada $L_{i,j}$ se requiere tiempo constante, la complejidad temporal del algoritmo es $O(mn)$.

A continuación, el método de programación dinámica se ilustra con un ejemplo. Sean $A = a\ b\ c\ d$ y $B = c\ b\ d$. En este caso se tiene

$a_1 = a$
$a_2 = b$
$a_3 = c$
$a_4 = d$
$b_1 = c$
$b_2 = b$
y $b_3 = d$.

La longitud de la subsecuencia común más larga se encuentra paso por paso, lo cual se ilustra en la tabla 7-2.

TABLA 7-2 Los valores de los $L_{i,j}$

b_j \ a_i			a	b	c	d
		0	**1**	**2**	**3**	**4**
	0	0	0	0	0	0
c	1	0	0	0	1	1
b	2	0	0	1	1	1
d	3	0	0	1	1	2

$L_{i,j}$

7-3 EL PROBLEMA DE ALINEACIÓN DE 2 SECUENCIAS

Sean $A = a_1a_2 \ldots a_m$ y $B = b_1b_2 \ldots b_n$ dos secuencias sobre un conjunto alfabeto Σ. Una alineación de secuencias de A y B es una matriz $2 \times k$ de M $(k \geq m, n)$ de caracteres sobre $\Sigma \cup \{-\}$ tal que ninguna columna de M consta completamente de guiones, y los resultados que se obtienen al eliminar todos los guiones en el primer renglón y en el segundo renglón de M son iguales a A y B, respectivamente. Por ejemplo, si $A = a\ b\ c$ y $B = c\ b\ d$, una alineación posible de éstas sería

$$\begin{matrix} a & b & c & - & d \\ - & - & c & b & d \end{matrix}$$

Otra alineación posible de las dos secuencias es

$$\begin{matrix} a & b & c & d \\ c & b & - & d. \end{matrix}$$

Puede verse que la segunda alineación parece mejor que la primera. Con precisión, es necesario definir una función de puntaje que pueda usarse para medir el desempeño de una alineación. A continuación se presentará una función así. Debe entenderse que es posible definir otros tipos de funciones de puntaje.

Sea $f(x, y)$ el puntaje por alinear x con y. Suponga que tanto x como y son caracteres. Entonces $f(x, y) = 2$ si x y y son iguales y $f(x, y) = 1$ si x y y no son iguales. Si x o y es "$-$", entonces $f(x, y) = -1$.

El puntaje de una alineación es la suma total de los puntajes de las columnas. Así, el puntaje de la siguiente alineación es $-1 - 1 + 2 - 1 + 2 = 1$.

a – b c d
– c b – d

El puntaje de la siguiente alineación es $1 + 2 - 1 + 2 = 4$.

a b c d
c b – d

El problema de alineación de 2 secuencias para A y B consiste en encontrar una alineación óptima con el puntaje máximo. Resulta fácil plantear una fórmula de recurrencia semejante a la que se utilizó para encontrar la secuencia común más larga. Sea $A_{i,j}$ el puntaje de la alineación óptima entre $a_1a_2\ldots\ a_i$ y $b_1b_2\ldots\ b_j$, donde $1 \leq i \leq m$ y $1 \leq j \leq n$. Entonces, $A_{i,j}$ puede expresarse como sigue:

$$A_{0,0} = 0$$
$$A_{i,0} = i \cdot f(a_i, -)$$
$$A_{0,j} = j \cdot f(-, b_j)$$
$$A_{i,j} = \text{máx}\begin{cases} A_{i-1,j} + f(a_i, -) \\ A_{i-1,j-1} + f(a_i, b_j) \\ A_{i,j-1} + f(-, b_j) \end{cases}$$

Quizá sea necesario explicar el significado de la fórmula de recurrencia anterior:

$A_{0,0}$ es para la condición inicial.
$A_{i,0}$ significa que todas las $a_1, a_2, \ldots, a_i$ están alineadas con "–".
$A_{0,j}$ significa que todas las $b_1, b_2, \ldots, b_j$ están alineadas con "–".
$A_{i-1,j} + f(a_i, -)$ significa que a_i está alineada con "–" y se requiere encontrar una alineación óptima entre $a_1, a_2, \ldots, a_{i-1}$ y $b_1, b_2, \ldots, b_j$.
$A_{i-1,j-1} + f(a_i, b_j)$ significa que a_i está alineada con b_j y se requiere encontrar una alineación óptima entre $a_1, a_2, \ldots, a_{i-1}$ y $b_1, b_2, \ldots, b_{j-1}$.
$A_{i,j-1} + f(-, b_j)$ significa que b_j está alineada con "–" y se requiere encontrar una alineación óptima entre $a_1, a_2, \ldots, a_i$ y $b_1, b_2, \ldots, b_{j-1}$.

Las $A_{i,j}$ para $A = a\ b\ d\ a\ d$ y $B = b\ a\ c\ d$ usando la fórmula de recurrencia anterior se enumeran en la tabla 7-3.

TABLA 7-3 Los valores de las $A_{i,j}$ para $A = a\ b\ d\ a\ d$ y $B = b\ a\ c\ d$.

a_i / b_j			*a*	*b*	*d*	*a*	*d*
		0	**1**	**2**	**3**	**4**	**5**
	0	0	−1	−2	−3	−4	−5
b	1	−1	1	1	0	−1	−2
a	2	−2	1	2	2	2	1
c	3	−3	0	2	3	3	3
d	4	−4	−1	1	4	4	5

En la tabla 7-3 se registró cómo se obtuvo cada $A_{i,j}$. Una flecha de (a_i, b_j) a (a_{i-1}, b_{j-1}) significa que a_i corresponde a b_j. Una flecha de (a_i, b_j) a (a_{i-1}, b_j) significa que a_i corresponde a "–", y una flecha de (a_i, b_j) a (a_i, b_{j-1}) significa que b_j corresponde a "–". Con base en las flechas de la tabla, es posible rastrear y encontrar que la alineación óptima es:

a b d a d
– b a c d

Se considerará otro ejemplo. Sean $A = a\ b\ c\ d$ y $B = c\ b\ d$ las dos secuencias. En la tabla 7-4 se muestra cómo se obtuvieron las $A_{i,j}$.

TABLA 7-4 Los valores de las $A_{i,j}$ para $A = a\ b\ c\ d$ y $B = c\ b\ d$.

		1	**2**	**3**	**4**
		–	a	b	c	d
	–	0	-1	-2	-3	-4
1	a	-1	1	0	0	-1
2	b	-2	0	3	2	1
3	d	-3	-1	2	4	4

Con base en las flechas de la tabla 7-4, es posible realizar un rastreo y encontrar que la alineación óptima es

a b c d
c b – d

La alineación de secuencias puede considerarse como un método para medir la semejanza de dos secuencias. A continuación se presentará un concepto, denominado distancia de edición, que también se usa bastante a menudo para medir la semejanza entre dos secuencias.

Se considerarán dos secuencias $A = a_1a_2 \ldots a_m$ y $B = b_1b_2 \ldots b_n$. A puede transformarse en B si se ejecutan las tres siguientes operaciones de edición: eliminar un carácter de A, insertar un carácter en A y sustituir un carácter en A por otro carácter. Por ejemplo, sean $A = GTAAHTY$ y $B = TAHHYC$. A puede transformarse en B si se realizan las siguientes operaciones:

1. Eliminar el primer carácter G de A. La secuencia A se convierte en $A = TAAHTY$.
2. Sustituir el tercer carácter de A; a saber, A por H. La secuencia A se convierte en $A = TAHHTY$.
3. Eliminar el quinto carácter de A; a saber, T. La secuencia A se convierte en $A = TAHHY$.
4. Insertar C después del último carácter de A. La secuencia A se convierte en $A = TAHHYC$, que es idéntica a B.

Con cada operación puede asociarse un costo. La distancia de edición es el costo mínimo asociado con las operaciones de edición necesarias para transformar la secuencia A en la secuencia B. Si para cada operación el costo es uno, la distancia de edición se convierte en el número mínimo de operaciones de edición necesarias para transformar A en B. En el ejemplo anterior, si el costo de cada operación es 1 y la distancia de edición entre A y B es 4, se requieren por lo menos cuatro operaciones de edición.

Resulta evidente que la distancia de edición puede encontrarse con el método de programación dinámica. Fácilmente puede plantearse una fórmula de recurrencia semejante a la que se utilizó para encontrar la secuencia común más larga o una alineación óptima entre dos secuencias. Sean α, β y γ los costos de inserción, eliminación y sustitución, respectivamente. Sea $A_{i,j}$ la distancia de edición entre $a_1a_2 \ldots a_i$ y $b_1b_2 \ldots b_j$. Entonces, $A_{i,j}$ puede expresarse como sigue:

$$A_{0,0} = 0$$

$$A_{i,0} = i\beta$$
$$A_{0,j} = j\alpha$$
$$A_{i,j} = \begin{cases} A_{i-1,j-1} & \text{si } a_i = b_j \\ \text{mín} \begin{cases} A_{i-1,j} + \beta \\ A_{i-1,j-1} + \gamma \\ A_{i,j-1} + \alpha \end{cases} & \text{en caso contrario} \end{cases}$$

En realidad, resulta fácil ver que el problema para encontrar la distancia de edición es equivalente al problema de la alineación óptima. No es la intención presentar una demostración formal aquí, ya que ésta puede percibirse fácilmente a partir de la semejanza de las fórmulas de recurrencia respectivas. En vez de ello, para ilustrar esta cuestión se usará el ejemplo presentado.

Considere otra vez $A = GTAAHTY$ y $B = TAHHYC$. Una alineación óptima produciría lo siguiente:

$$\begin{array}{cccccccc} G & T & A & A & H & T & Y & - \\ - & T & A & H & H & - & Y & C \end{array}$$

Un análisis de la alineación anterior muestra la equivalencia entre las operaciones de edición y las operaciones de alineación como sigue:

1. (a_i, b_j) en la determinación de la alineación es equivalente a la operación de sustitución en la determinación de la distancia de edición. En este caso, a_i se sustituye por b_j.
2. $(a_i, -)$ en la determinación de la alineación es equivalente a eliminar a_i en A en la determinación de la distancia de edición.
3. $(-, b_j)$ en la determinación de la alineación es equivalente a insertar b_j en A en la determinación de la distancia de edición.

El lector puede usar estas reglas y la alineación óptima ya encontrada para producir las cuatro operaciones de edición. Para dos secuencias dadas $A = a_1a_2 \ldots a_m$ y $B = b_1b_2 \ldots b_n$, para registrar $A_{i,j}$ se requiere una tabla con $(n + 1)(m + 1)$ elementos. Es decir, para encontrar una alineación óptima para las dos secuencias $A = a_1a_2 \ldots a_m$ y $B = b_1b_2 \ldots b_n$ se requiere tiempo $O(nm)$.

7-4 PROBLEMA DE APAREAMIENTO DEL MÁXIMO PAR DE BASES DE ARN

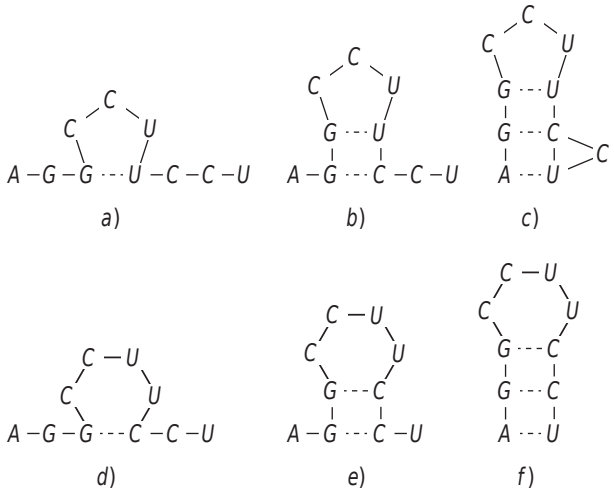
El ácido ribonucleico (ARN) es una cadena simple de nucleótidos (bases), adenina (A), guanina (G), citosina (C) y uracil (U). La secuencia de las bases A, G, C y U se deno-

mina *estructura primaria* de un ARN. En el ARN, *G* y *C* pueden formar un par de bases *G*≡*C* mediante un triple enlace de hidrógeno. *A* y *U* pueden formar un par de bases *A*=*U* mediante un doble enlace de hidrógeno, y *G* y *U* pueden formar una par de bases *G*-*U* mediante un enlace simple de hidrógeno. Debido a estos enlaces de hidrógeno, la estructura primaria de un ARN puede plegarse sobre sí misma para formar su estructura secundaria. Por ejemplo, suponga que se tiene la siguiente secuencia de ARN.

A–*G*–*G*–*C*–*C*–*U*–*U*–*C*–*C*–*U*

Así, esta estructura puede plegarse sobre sí misma para formar muchas estructuras secundarias posibles. En la figura 7-14 se muestran seis estructuras secundarias posibles de esta secuencia. En la naturaleza, no obstante, sólo existe una secuencia secundaria correspondiente a una secuencia de ARN. ¿Cuál es la estructura secundaria verdadera de una secuencia de ARN?

FIGURA 7-14 Seis estructuras secundarias posibles de la secuencia de ARN *A*–*G*–*G*–*C*–*C*–*U*–*U*–*C*–*C*–*U* (las líneas discontinuas indican los enlaces de hidrógeno).



Según la hipótesis de la termodinámica, la estructura secundaria verdadera de una secuencia de ARN es la que tiene energía libre mínima, lo cual se explicará después. En la naturaleza sólo puede existir la estructura estable, y esta estructura debe ser la que tenga energía libre mínima. En una estructura secundaria de ARN, los pares básicos aumentan la estabilidad estructural, y las bases no apareadas la disminuyen. Los pares básicos de los tipos *G*≡*C* y *A*=*U* (denominados pares básicos *Watson-Crick*)

son más estables que los del tipo *G–U* (denominados pares básicos *wobble*). Según estos factores, puede encontrarse que la estructura secundaria de la figura 7-14*f*) es la estructura secundaria verdadera de la secuencia *A–G–G–C–C–U–U–C–C–U*.

Una secuencia de ARN se representa por una cadena de n caracteres $R = r_1 r_2 \ldots r_n$, donde $r_i \in \{A, C, G, U\}$. Típicamente, n puede variar desde 20 hasta 2 000. Una *estructura secundaria* de R es un conjunto S de pares básicos (r_i, r_j) donde $1 \leq i < j \leq n$ tal que se cumplen las condiciones siguientes.

1. $j - i > t$, donde t es una constante positiva pequeña. Típicamente, $t = 3$.
2. Si (r_i, r_j) y (r_k, r_l) son dos pares básicos en S, entonces ocurre una de las siguientes posibilidades
   a) $i = k$ y $j = l$; es decir, (r_i, r_j) y (r_k, r_l) son el mismo par de bases.
   b) $i < j < k < l$; es decir, (r_i, r_j) precede a (r_k, r_l), o bien
   c) $i < k < l < j$; es decir, (r_i, r_j) incluye a (r_k, r_l).

La primera condición implica que la secuencia de ARN no se pliega demasiado acusadamente sobre sí misma. La segunda condición significa que cada nucleótido puede formar parte a lo sumo en un par de bases, y garantiza que la estructura secundaria no contiene un seudonudo. Dos pares básicos (r_i, r_j) y (r_k, r_l) se denominan *seudonudo* si $i < k < j < l$. Los seudonudos ocurren en moléculas de ARN, aunque su exclusión simplifica el problema.

Recuerde que el objetivo de la predicción de la estructura secundaria es encontrar una estructura secundaria que tenga la energía libre mínima. Por lo tanto, es necesario contar con un método para calcular la energía libre de una estructura secundaria S. Debido a que la formación de pares básicos proporciona efectos estabilizadores a la energía libre estructural, el método más simple para medir la energía libre de S consiste en asignar energía a cada par de bases de S y luego la energía libre de S es la suma de las energías de todos los pares básicos. Debido a los diferentes enlaces de hidrógeno, a las energías de los pares básicos suelen asignarse valores distintos. Por ejemplo, los valores razonables para $A \equiv U$, $G \equiv C$ y $G–U$ son -3, -2 y -1, respectivamente. Otros valores posibles podrían ser que todas las energías de todos los pares básicos sean iguales. En este caso, el problema se convierte en encontrar una estructura secundaria con el número máximo de pares básicos. Esta versión del problema de la predicción de la estructura secundaria también se denomina *problema de apareamiento del máximo par de bases de RNA* (maximum base pair matching problem), ya que una estructura secundaria puede considerarse como un apareamiento. A continuación se presentará un algoritmo de programación dinámica para resolver este problema y que se define como sigue: dada una secuencia $R = r_1 r_2 \ldots r_n$ de RNA, encontrar una estructura secundaria de RNA que tenga el número máximo de pares básicos.

Sea $S_{i,j}$ la estructura secundaria del número máximo de pares básicos en la subcadena $R_{i,j} = r_i r_{i+1} \ldots r_j$. El número de pares básicos apareados en $S_{i,j}$ se denota por $M_{i,j}$. Observe que no es posible aparear entre sí dos bases r_i, r_j, donde $1 \leq i < j \leq n$. Los pares básicos admisibles que se consideran aquí son pares básicos Watson-Crick (es decir, $A \equiv U$ y $G \equiv C$) y pares básicos wobble (es decir, G–U). Sea $WW = \{(A, U), (U, A), (G, C), (C, G), (G, U), (U, G)\}$. Luego, se usa una función $\rho(r_i, r_j)$ para indicar si dos bases r_i y r_j cualesquiera pueden constituir un par de bases legal:

$$\rho(r_i, r_j) = \begin{cases} 1 & \text{si } (r_i, r_j) \in WW \\ 0 & \text{en caso contrario} \end{cases}$$

Por definición, se sabe que una secuencia de RNA no se pliega de manera muy pronunciada sobre sí misma. Es decir, si $j - i \leq 3$, entonces r_i y r_j no pueden ser un par de bases de $S_{i,j}$. Entonces, se hace $M_{i,j} = 0$ si $j - i \leq 3$.

Para calcular $M_{i,j}$, donde $j - i > 3$, se consideran los casos siguientes desde el punto de vista de las r_j.

Caso 1: En la solución óptima, r_j no está apareada con ninguna otra base. En este caso, se encuentra una solución óptima para $r_i r_{i+1} \ldots r_{j-1}$ y $M_{i,j} = M_{i,j-1}$ (consulte la figura 7-15).

Caso 2: En la solución óptima, r_j está apareada con r_i. En este caso, se encuentra una solución óptima para $r_{i+1} r_{i+2} \ldots r_{j-1}$ y $M_{i,j} = 1 + M_{i+1,j-1}$ (consulte la figura 7-16).

Caso 3: En la solución óptima, r_j está apareada con alguna r_k, donde $i + 1 \leq k \leq j - 4$. En este caso, se encuentran las soluciones óptimas para $r_i r_{i+1} \ldots r_{k-1}$ y $r_{k+1} r_{k+1} \ldots r_{j-1}$, $M_{i,j} = 1 + M_{i,k-1} + M_{k+1,j-1}$ (consulte la figura 7-17).

Debido a que se quiere encontrar la k entre $i + 1$ y $j + 4$ tal que $M_{i,j}$ sea el máximo, se tiene

FIGURA 7-15 Ilustración del caso 1.

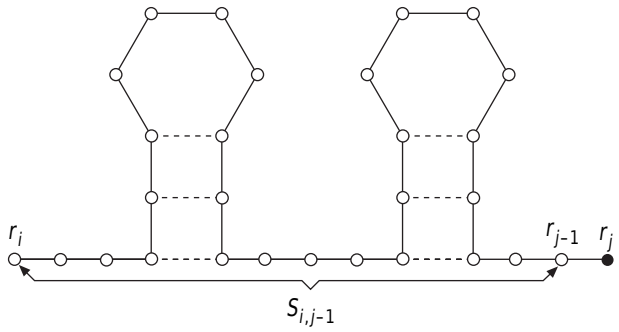


FIGURA 7-16 Ilustración del caso 2.

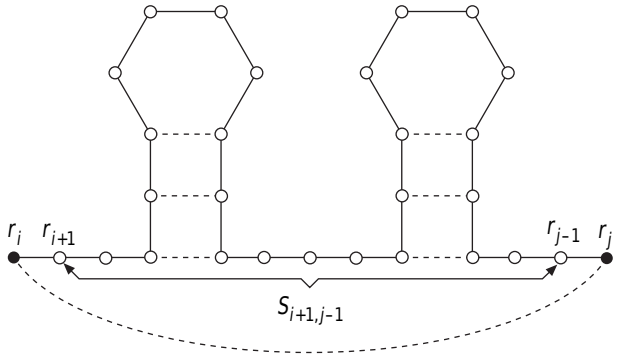
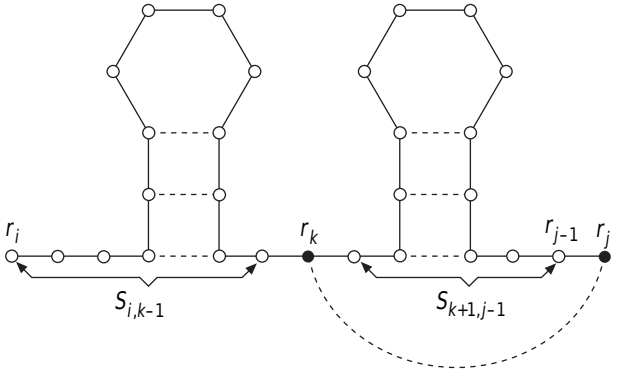


FIGURA 7-17 Ilustración del caso 3.



$$M_{i,j} = \max_{i+1 \le k \le j-4} \{1 + M_{i,k-1} + M_{k+1,j-1}\}$$

En resumen, para calcular $M_{i,j}$ se tiene la siguiente fórmula de recurrencia

Si $j - i \leqq 3$, entonces $M_{i,j} = 0$.

Si $j - i > 3$, entonces $M_{i,j} = \max \begin{cases} M_{i,j-1} \\ (1 + M_{i+1,j-1}) \times \rho(r_i, r_j) \\ \max\limits_{i+1 \le k \le j-4} \{1 + M_{i,k-1} + M_{k+1,j-1}\} \times \rho(r_k, r_j) \end{cases}$

Según la fórmula en la página anterior, el algoritmo 7-1 para calcular $M_{1,n}$ puede diseñarse aplicando la técnica de programación dinámica. En la tabla 7-5 se ilustra el cálculo de $M_{1,n}$, donde $1 \leq i < j \leq 10$, para una secuencia de ARN $R_{1,10} = A\text{–}G\text{–}G\text{–}C\text{–}C\text{–}U\text{–}U\text{–}C\text{–}C\text{–}U$. Como resultado, puede encontrarse que el número máximo de pares básicos en $S_{1,10}$ es 3, ya que $M_{1,10} = 3$.

TABLA 7-5 Cálculo del número máximo de pares básicos de una secuencia de ARN $A\text{–}G\text{–}G\text{–}C\text{–}C\text{–}U\text{–}U\text{–}C\text{–}C\text{–}U$.

i \ j	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	1	2	2	2	3
2	–	0	0	0	0	1	1	2	2	2
3	–	–	0	0	0	0	1	1	1	1
4	–	–	–	0	0	0	0	0	0	0
5	–	–	–	–	0	0	0	0	0	0
6	–	–	–	–	–	0	0	0	0	0
7	–	–	–	–	–	–	0	0	0	0
8	–	–	–	–	–	–	–	0	0	0
9	–	–	–	–	–	–	–	–	0	0
10	–	–	–	–	–	–	–	–	–	0

Algoritmo 7-1 □ Un algoritmo de apareamiento del máximo par de bases de RNA

Input: Una secuencia de ARN $R = r_1 r_2 \ldots r_n$.

Output: Encontrar una estructura secundaria de RNA que tenga el número máximo de pares básicos.

Paso 1: /* Cálculo de la función $\rho(r_i, r_j)$ para $1 \leq i < j \leq n$ */
$WW = \{(A, U), (U, A), (G, C), (C, G), (G, U), (U, G)\}$;
for $i = 1$ **to** n **do**
  for $j = i$ **to** n **do**
    if $(r_i, r_j) \in WW$ **then** $\rho(r_i, r_j) = 1$; **else** $\rho(r_i, r_j) = 0$;

```
        end for
    end for
Paso 2: /* Inicialización de M_{i,j} para j − i ≤ 3 */
    for i = 1 to n do
        for j = i to i + 3 do
            if j ≦ n then M_{i,j} = 0;
        end for
    end for
Paso 3: /* Cálculo de M_{i,j} para j − i > 3 */
    for h = 4 to n − 1 do
        for i = 1 to n − h do
            j = i + h;
            caso1 = M_{i,j−1};
            caso2 = (1 + M_{i+1,j−1}) × ρ(r_i, r_j);
            caso3 = M_{i,j} máx_{i+1≤k≤j−4} {(1 + M_{i,k−1} + M_{k+1,j−1}) × ρ(r_k, r_j)};
            M_{i,j} = máx{caso1, caso2, caso3};
        end for
    end for
```

A continuación se ilustra todo el procedimiento en detalle.

r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}
A	G	G	C	C	U	U	C	C	U

1. $i = 1, j = 5, \rho(r_1, r_5) = \rho(A, C) = 0$

$$M_{1,5} = \text{máx}\begin{cases} M_{1,4} \\ (1 + M_{2,4}) \times \rho(r_1, r_5) \end{cases}$$
$$= \text{máx}\{0, 0\} = 0.$$

2. $i = 2, j = 6, \rho(r_2, r_6) = \rho(G, U) = 1$

$$M_{2,6} = \text{máx}\begin{cases} M_{2,5} \\ (1 + M_{3,5}) \times \rho(r_2, r_6) \end{cases}$$
$$= \text{máx}\{0, (1 + 0) \times 1\} = \text{máx}\{0, 1\} = 1.$$

r_2 coincide con r_6.

3. $i = 3, j = 7, \rho(r_3, r_7) = \rho(G, U) = 1$

$$M_{3,7} = \text{máx}\begin{cases} M_{3,6} \\ (1 + M_{4,6}) \times \rho(r_3, r_7) \end{cases}$$
$$= \text{máx}\{0, (1 + 0) \times 1\} = \text{máx}\{0, 1\} = 1.$$

r_3 coincide con r_7.

4. $i = 4, j = 8, \rho(r_4, r_8) = \rho(C, C) = 0$

$$M_{4,8} = \text{máx}\begin{cases} M_{4,7} \\ (1 + M_{5,7}) \times \rho(r_4, r_8) \end{cases}$$
$$= \text{máx}\{0, (1 + 0) \times 0\} = 0.$$

5. $i = 5, j = 9, \rho(r_5, r_9) = \rho(C, C) = 0$

$$M_{5,9} = \text{máx}\begin{cases} M_{5,8} \\ (1 + M_{6,8}) \times \rho(r_5, r_9) \end{cases}$$
$$= \text{máx}\{0, (1 + 0) \times 0\} = \text{máx}\{0, 0\} = 0.$$

6. $i = 6, j = 10, \rho(r_6, r_{10}) = \rho(U, U) = 0$

$$M_{6,10} = \text{máx}\begin{cases} M_{6,9} \\ (1 + M_{7,9}) \times \rho(r_6, r_{10}) \end{cases}$$
$$= \text{máx}\{0, (1 + 0) \times 0\} = \text{máx}\{0, 0\} = 0.$$

7. $i = 1, j = 6, \rho(r_1, r_6) = \rho(A, U) = 1$

$$M_{1,6} = \text{máx}\begin{cases} M_{1,5} \\ (1 + M_{2,5}) \times \rho(r_1, r_6) \\ (1 + M_{1,1} + M_{3,5}) \times \rho(r_2, r_6) \end{cases}$$
$$= \text{máx}\{0, (1 + 0) \times 1, (1 + 0 + 0) \times 1\}$$
$$= \text{máx}\{0, 1, 1\} = 1.$$

r_1 coincide con r_6.

8. $i = 2, j = 7, \rho(r_2, r_7) = \rho(G, U) = 1$

$$M_{2,7} = \text{máx}\begin{cases} M_{2,6} \\ (1 + M_{3,6}) \times \rho(r_2, r_7) \\ (1 + M_{2,2} + M_{4,6}) \times \rho(r_3, r_7) \end{cases}$$
$$= \text{máx}\{1, (1 + 0) \times 1, (1 + 0 + 0) \times 1\}$$
$$= \text{máx}\{1, 1, 1\} = 1.$$

9. $i = 3, j = 8, \rho(r_3, r_8) = \rho(G, C) = 1$

$$M_{3,8} = \text{máx}\begin{cases} M_{3,7} \\ (1 + M_{4,7}) \times \rho(r_3, r_8) \\ (1 + M_{3,3} + M_{5,7}) \times \rho(r_4, r_8) \end{cases}$$
$$= \text{máx}\{1, (1 + 0) \times 1, (1 + 0 + 0) \times 0\}$$
$$= \text{máx}\{1, 1, 0\} = 1.$$

r_3 coincide con r_8.

10. $i = 4, j = 9, \rho(r_4, r_9) = \rho(C, U) = 0$

$$M_{4,9} = \text{máx}\begin{cases} M_{4,8} \\ (1 + M_{5,8}) \times \rho(r_4, r_9) \\ (1 + M_{4,4} + M_{6,8}) \times \rho(r_5, r_9) \end{cases}$$
$$= \text{máx}\{0, (1 + 0) \times 0, (1 + 0 + 0) \times 0\}$$
$$= \text{máx}\{0, 0, 0\} = 0.$$

11. $i = 5, j = 10, \rho(r_5, r_{10}) = \rho(C, U) = 0$

$$M_{5,10} = \text{máx}\begin{cases} M_{5,9} \\ (1 + M_{6,9}) \times \rho(r_5, r_{10}) \\ (1 + M_{5,5} + M_{7,9}) \times \rho(r_6, r_{10}) \end{cases}$$
$$= \text{máx}\{0, (1 + 0) \times 0, (1 + 0 + 0) \times 0\}$$
$$= \text{máx}\{0, 0, 0\} = 0.$$

12. $i = 1, j = 7, \rho(r_1, r_7) = \rho(A, U) = 1$

$$M_{1,7} = \text{máx}\begin{cases} M_{1,6} \\ (1 + M_{2,6}) \times \rho(r_1, r_7) \\ (1 + M_{1,1} + M_{3,6}) \times \rho(r_2, r_7) \\ (1 + M_{1,2} + M_{4,6}) \times \rho(r_3, r_7) \end{cases}$$

$$= \text{máx}\{1, (1 + 1) \times 1, (1 + 0 + 0) \times 1, (1 + 0 + 0) \times 1\}$$
$$= \text{máx}\{1, 2, 1, 1\} = 2.$$

13. $i = 2, j = 8, \rho(r_2, r_8) = \rho(G, C) = 1$

$$M_{2,8} = \text{máx}\begin{cases} M_{2,7} \\ (1 + M_{3,7}) \times \rho(r_2, r_8) \\ (1 + M_{2,2} + M_{4,7}) \times \rho(r_3, r_8) \\ (1 + M_{2,3} + M_{5,7}) \times \rho(r_4, r_8) \end{cases}$$

$$= \text{máx}\{1, (1 + 1) \times 1, (1 + 0 + 0) \times 1, (1 + 0 + 0) \times 0\}$$
$$= \text{máx}\{1, 2, 1, 0\} = 2.$$

r_2 coincide con r_8; r_3 coincide con r_7.

14. $i = 3, j = 9, \rho(r_3, r_9) = \rho(G, C) = 1$

$$M_{3,9} = \text{máx}\begin{cases} M_{3,8} \\ (1 + M_{4,8}) \times \rho(r_3, r_9) \\ (1 + M_{3,3} + M_{5,8}) \times \rho(r_4, r_9) \\ (1 + M_{3,4} + M_{6,8}) \times \rho(r_5, r_9) \end{cases}$$

$$= \text{máx}\{1, (1 + 0) \times 1, (1 + 0 + 0) \times 0, (1 + 0 + 0) \times 0\}$$
$$= \text{máx}\{1, 1, 0, 0\} = 1.$$

r_3 coincide con r_9.

15. $i = 4, j = 10, \rho(r_4, r_{10}) = \rho(C, U) = 0$

$$M_{4,10} = \text{máx}\begin{cases} M_{4,9} \\ (1 + M_{5,9}) \times \rho(r_4, r_{10}) \\ (1 + M_{4,4} + M_{6,9}) \times \rho(r_5, r_{10}) \\ (1 + M_{4,5} + M_{7,9}) \times \rho(r_6, r_{10}) \end{cases}$$

$$= \text{máx}\{0, (1 + 0) \times 0, (1 + 0 + 0) \times 0, (1 + 0 + 0) \times 0\}$$
$$= \text{máx}\{0, 0, 0, 0\} = 0.$$

16. $i = 1, j = 8, \rho(r_1, r_8) = \rho(A, C) = 0$

$$M_{1,8} = \text{máx}\begin{cases} M_{1,7} \\ (1 + M_{2,7}) \times \rho(r_1, r_8) \\ (1 + M_{1,1} + M_{3,7}) \times \rho(r_2, r_8) \\ (1 + M_{1,2} + M_{4,7}) \times \rho(r_3, r_8) \\ (1 + M_{1,3} + M_{5,7}) \times \rho(r_4, r_8) \end{cases}$$

$$= \text{máx}\{2, (1 + 1) \times 0, (1 + 0 + 1) \times 1, (1 + 0 + 0) \times 1, (1 + 0 + 0) \times 0\}$$

$$= \text{máx}\{2, 0, 2, 1, 0\} = 2.$$

r_1 coincide con r_7; r_2 coincide con r_6.

17. $i = 2, j = 9, \rho(r_2, r_9) = \rho(G, C) = 1$

$$M_{2,9} = \text{máx}\begin{cases} M_{2,8} \\ (1 + M_{3,8}) \times \rho(r_2, r_9) \\ (1 + M_{2,2} + M_{4,8}) \times \rho(r_3, r_9) \\ (1 + M_{2,3} + M_{5,8}) \times \rho(r_4, r_9) \\ (1 + M_{2,4} + M_{6,8}) \times \rho(r_5, r_9) \end{cases}$$

$$= \text{máx}\{2, (1 + 1) \times 1, (1 + 0 + 0) \times 1, (1 + 0 + 0) \times 0, (1 + 0 + 0) \times 0\}$$

$$= \text{máx}\{2, 2, 1, 0, 0\} = 2.$$

r_2 coincide con r_9; r_3 coincide con r_8.

18. $i = 3, j = 10, \rho(r_3, r_{10}) = \rho(G, U) = 1$

$$M_{3,10} = \text{máx}\begin{cases} M_{3,9} \\ (1 + M_{4,9}) \times \rho(r_3, r_{10}) \\ (1 + M_{3,3} + M_{5,9}) \times \rho(r_4, r_{10}) \\ (1 + M_{3,4} + M_{6,9}) \times \rho(r_5, r_{10}) \\ (1 + M_{3,5} + M_{7,9}) \times \rho(r_6, r_{10}) \end{cases}$$

$$= \text{máx}\{1, (1 + 0) \times 1, (1 + 0 + 0) \times 0, (1 + 0 + 0) \times 0, (1 + 0 + 0) \times 0\}$$

$$= \text{máx}\{1, 1, 0, 0, 0\} = 1.$$

r_3 coincide con r_{10}.

19. $i = 1, j = 9, \rho(r_1, r_9) = \rho(A, C) = 0$

$$M_{1,9} = \text{máx}\begin{cases} M_{1,8} \\ (1 + M_{2,8}) \times \rho(r_1, r_9) \\ (1 + M_{1,1} + M_{3,8}) \times \rho(r_2, r_9) \\ (1 + M_{1,2} + M_{4,8}) \times \rho(r_3, r_9) \\ (1 + M_{1,3} + M_{5,8}) \times \rho(r_4, r_9) \\ (1 + M_{1,4} + M_{6,8}) \times \rho(r_5, r_9) \end{cases}$$

$$= \text{máx}\{2, (1 + 2) \times 0, (1 + 0 + 1) \times 1, (1 + 0 + 0) \times 1, (1 + 0 + 0) \times 0, (1 + 0 + 0) \times 0\}$$

$$= \text{máx}\{2, 0, 2, 1, 0, 0\} = 2.$$

r_1 coincide con r_7; r_2 coincide con r_6.

20. $i = 2, j = 10, \rho(r_2, r_{10}) = \rho(G, U) = 1$

$$M_{2,10} = \text{máx}\begin{cases} M_{2,9} \\ (1 + M_{3,9}) \times \rho(r_2, r_{10}) \\ (1 + M_{2,2} + M_{4,9}) \times \rho(r_3, r_{10}) \\ (1 + M_{2,3} + M_{5,9}) \times \rho(r_4, r_{10}) \\ (1 + M_{2,4} + M_{6,9}) \times \rho(r_5, r_{10}) \\ (1 + M_{2,5} + M_{7,9}) \times \rho(r_6, r_{10}) \end{cases}$$

$$= \text{máx}\{2, 2, 1, 0, 0, 0\} = 2.$$

r_2 coincide con r_{10}; r_3 coincide con r_9.

21. $i = 1, j = 10, \rho(r_1, r_{10}) = \rho(A, U) = 1$

$$M_{1,10} = \text{máx}\begin{cases} M_{1,9} \\ (1 + M_{2,9}) \times \rho(r_1, r_{10}) \\ (1 + M_{1,1} + M_{3,9}) \times \rho(r_2, r_{10}) \\ (1 + M_{1,2} + M_{4,9}) \times \rho(r_3, r_{10}) \\ (1 + M_{1,3} + M_{5,9}) \times \rho(r_4, r_{10}) \\ (1 + M_{1,4} + M_{6,9}) \times \rho(r_5, r_{10}) \\ (1 + M_{1,5} + M_{7,9}) \times \rho(r_6, r_{10}) \end{cases}$$

$$= \text{máx}\{2, 3, 2, 1, 0, 0, 0\} = 3.$$

r_1 coincide con r_{10}; r_2 coincide con r_9; r_3 coincide con r_8.

A continuación se analizará la complejidad temporal del algoritmo 7-1. Resulta evidente que el costo del paso 1 es $O(n)$. Para el paso 2, hay cuando mucho $\sum_{i=1}^{n} \sum_{j=i+4}^{n}$ iteraciones y cada iteración cuesta tiempo $O(j - i)$. Por lo tanto, el costo del paso 2 es

$$\sum_{i=1}^{n} \sum_{j=i+4}^{n} O(j-i) = O(n^3)$$

Así, la complejidad temporal del algoritmo 7-1 es $O(n^3)$.

7-5 PROBLEMA 0/1 DE LA MOCHILA

Este problema se analizó en el capítulo 6. En esta sección se demostrará que este problema puede resolverse fácilmente aplicando el método de programación dinámica. El problema 0/1 de la mochila se define como sigue: se tienen n objetos y una mochila. El objeto i pesa W_i y la mochila tiene una capacidad M. Si el objeto i se coloca en la mochila, se obtiene una ganancia P_i. El problema 0/1 de la mochila consiste en maximizar la ganancia total con la restricción de que el peso total de todos los objetos seleccionados sea cuando mucho M.

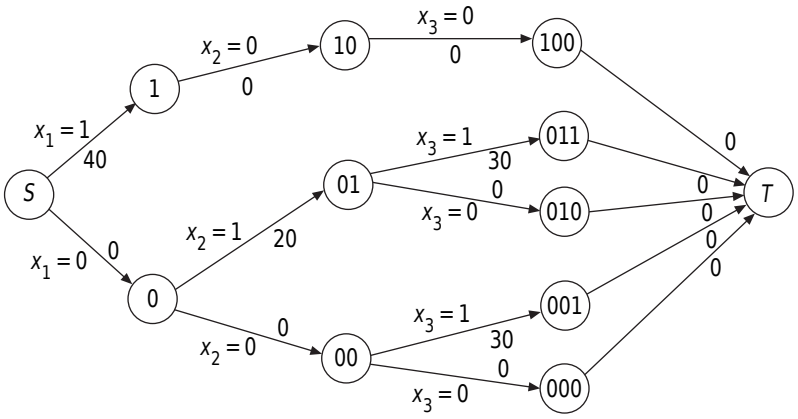
Hay una secuencia de acciones a emprender. Sea X_i la variable que denota si el objeto i se selecciona o no. Es decir, se hace que $X_i = 1$ si el objeto i se escoge y 0 en caso contrario. Si a X_1 se asigna 1 (el objeto 1 es escogido), entonces el problema restante se convierte en un problema 0/1 de la mochila modificado, donde M se vuelve $M - W_1$. En general, después de que se hace una secuencia de decisiones representadas por $X_1, X_2, \ldots, X_i$, el problema se reduce a uno que implica las decisiones $X_{i+1}, X_{i+2}, \ldots, X_n$ y $M' = M - \sum_{j=1}^{i} X_j W_j$. Así, no importa cuáles sean las decisiones $X_1, X_2, \ldots, X_i$, el resto de las decisiones $X_{i+1}, X_{i+2}, \ldots, X_n$ deben ser óptimas con respecto al nuevo problema de la mochila. A continuación se demostrará que el problema 0/1 de la mochila puede representarse mediante un problema de gráfica multietapas. Se supondrá que se tienen tres objetos y una mochila con capacidad 10. Los pesos y las ganancias de estos objetos se muestran en la tabla 7-6.

TABLA 7-6 Pesos y ganancias de tres objetos.

i	W_i	P_i
1	10	40
2	3	20
3	5	30

El método de programación dinámica para resolver este problema 0/1 de la mochila puede ilustrarse en la figura 7-18.

FIGURA 7-18 Método de programación dinámica para resolver el problema 0/1 de la mochila.



En cada nodo se tiene una etiqueta que especifica las decisiones que ya se han tomado hasta este nodo. Por ejemplo, 011 significa $X_1 = 0$, $X_2 = 1$ y $X_3 = 1$. En este caso se tiene interés en la ruta más larga, y es fácil darse cuenta de que ésta es

$$S \rightarrow 0 \rightarrow 01 \rightarrow 011 \rightarrow T$$

que corresponde a

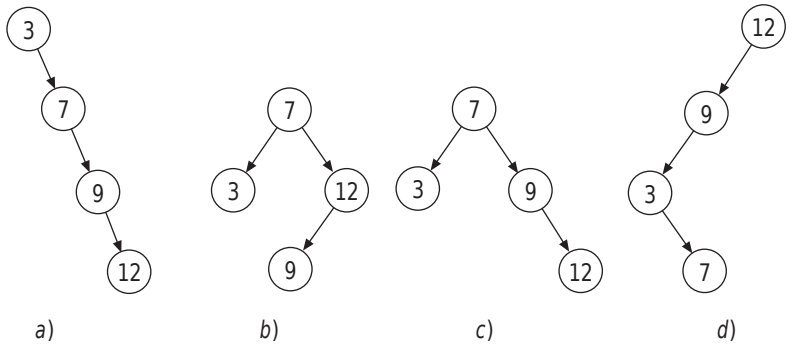
$$X_1 = 0,$$
$$X_2 = 1$$
y $$X_3 = 1$$

con el costo total igual a $20 + 30 = 50$.

7-6 EL PROBLEMA DEL ÁRBOL BINARIO ÓPTIMO

El árbol binario es quizás una de las estructuras de datos más conocidas. Dados n identificadores $a_1 < a_2 < a_3 \cdots < a_n$, pueden tenerse muchos árboles binarios distintos. Por ejemplo, se supondrá que se tienen cuatro identificadores: 3, 7, 9 y 12. En la figura 7-19 se enumeran cuatro árboles binarios distintos para este conjunto de datos.

FIGURA 7-19 Cuatro árboles binarios distintos para el mismo conjunto de datos.



Para un árbol binario dado, los identificadores almacenados cerca de la raíz del árbol pueden buscarse de manera más bien rápida, mientras que para recuperar datos almacenados lejos de la raíz se requieren varios pasos. A cada identificador a_i se asociará una probabilidad P_i, que es la probabilidad de que este identificador sea buscado. Para un identificador que no está almacenado en el árbol, también se supondrá que hay una probabilidad asociada con el identificador. Los datos se separan en $n + 1$ clases de equivalencia. Q_i denota la probabilidad de que X sea buscado donde $a_i < X < a_{i+1}$, en el supuesto de que a_0 es $-\infty$ y a_{n+1} es $+\infty$. Las probabilidades cumplen la siguiente ecuación:

$$\sum_{i=1}^{n} P_i + \sum_{i=0}^{n} Q_i = 1.$$

Resulta fácil determinar el número de pasos necesarios para realizar una búsqueda exitosa. Sea $L(a_i)$ el nivel del árbol binario donde se almacena a_i. Entonces la recuperación de a_i requiere $L(a_i)$ pasos si se hace que la raíz del árbol esté en el nivel 1. Para búsquedas infructuosas, la mejor forma de entenderlas es agregando nodos externos al árbol binario. Se considerará el caso en que se tienen identificadores 4, 5, 8, 10, 11, 12 y 14. Para este conjunto de datos, toda la región se separa como sigue:

	4,	5,	8,	10,	11,	12,	14	
↑	↑	↑	↑	↑	↑	↑	↑	
E_0	E_1	E_2	E_3	E_4	E_5	E_6	E_7	

Suponga que se tiene un árbol binario como se muestra en la figura 7-20. En este árbol binario hay algunos nodos que no tienen dos nodos descendientes. En estos nodos siempre ocurren búsquedas infructuosas. A todos estos nodos es posible agregar nodos ficticios, trazados como cuadrados, como se muestra en la figura 7-21.

FIGURA 7-20 Un árbol binario.

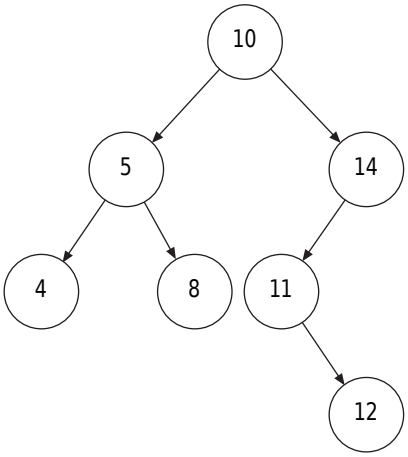
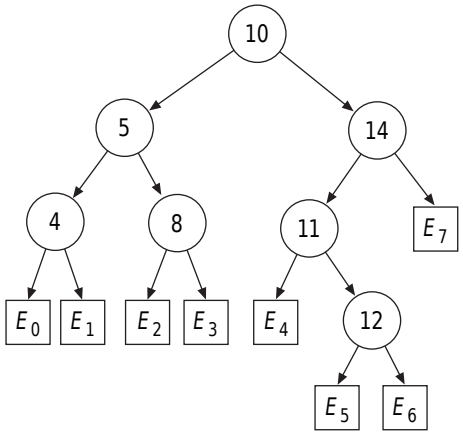


FIGURA 7-21 Un árbol binario con nodos externos agregados.



Todos los demás nodos se denominan nodos internos. Los agregados se denominan nodos externos. Las búsquedas infructuosas siempre terminan en esos nodos externos, mientras que las búsquedas exitosas terminan en nodos internos.

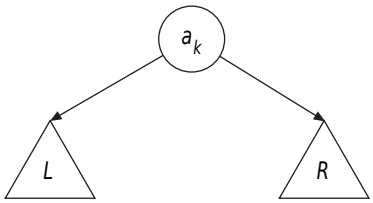
Después de que se agregan los nodos externos, el costo esperado de un árbol binario puede expresarse como sigue:

$$\sum_{i=1}^{n} P_i L(a_i) + \sum_{i=0}^{n} Q_i (L(E_i) - 1).$$

Un árbol binario óptimo es un árbol donde se ha minimizado el costo anterior. Debe observarse que se consumiría un tiempo excesivo si el problema del árbol binario óptimo se resuelve analizando exhaustivamente todos los árboles binarios posibles. Dados n identificadores, el número de árboles binarios distintos que es posible construir a partir de estos n identificadores es $\frac{1}{n+1}\binom{2n}{n}$, que es aproximadamente igual a $O(4^n/n^{3/2})$.

¿Por qué es posible aplicar el método de programación dinámica para resolver el problema del árbol binario óptimo? Un primer paso crítico para encontrar un árbol binario óptimo consiste en seleccionar un identificador, por ejemplo a_k, como raíz del árbol. Después de que se selecciona a_k, todos los identificadores menores que a_k constituyen el descendiente izquierdo de a_k, y todos los identificadores mayores que a_k constituyen el descendiente derecho, lo cual se ilustra en la figura 7-22.

FIGURA 7-22 Un árbol binario después de que a_k se selecciona como la raíz.

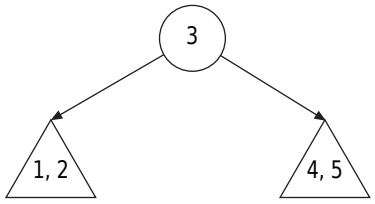


No es fácil determinar cuál a_k debe elegirse, por lo que es necesario analizar todas las a_k posibles. No obstante, una vez que se ha seleccionado a_k, se observa que ambos subárboles L y R también deben ser óptimos. Es decir, una vez que se ha seleccionado a_k, la tarea se reduce a encontrar árboles binarios óptimos para identificadores menores que a_k e identificadores mayores que a_k. El hecho de que este problema puede resolverse de manera recurrente indica que es posible aplicar el método de programación dinámica.

¿Por qué es idóneo el método de programación dinámica para encontrar un árbol binario óptimo? Antes que nada, proporciona una forma de pensamiento sistemático. Para ilustrar este hecho, suponga que hay cinco identificadores: 1, 2, 3, 4 y 5. Suponga que como raíz del árbol se selecciona a 3. Luego, como se muestra en la figura 7-23, el subárbol izquierdo contiene a 1 y 2 y el subárbol derecho contiene a 4 y 5.

El siguiente paso consiste en encontrar árboles binarios óptimos para 1 y 2 y para 4 y 5. Para el árbol binario que contiene a 1 y 2, sólo hay dos opciones para la raíz: 1 o 2. Situaciones semejantes ocurren para el árbol binario que contiene a 4 y 5. En otras palabras, el árbol binario óptimo con 3 como raíz es determinado por los cuatro subárboles siguientes:

FIGURA 7-23 Árbol binario con cierto identificador seleccionado como la raíz.



a) Un subárbol que contiene a 1 y 2 con 1 como la raíz.
b) Un subárbol que contiene a 1 y 2 con 2 como la raíz.
c) Un subárbol que contiene a 4 y 5 con 4 como la raíz.
d) Un subárbol que contiene a 4 y 5 con 5 como la raíz.

El subárbol izquierdo es *a*) o *b*) y el subárbol derecho es *c*) o *d*).

A continuación se analizará el caso en que se selecciona a 2 como la raíz. En este caso, el subárbol izquierdo sólo contiene un identificador; por ejemplo, 1. El subárbol derecho contiene tres identificadores: 3, 4 y 5. Para este subárbol derecho es necesario considerar tres subcasos: seleccionar 3 como la raíz, seleccionar 4 como la raíz y seleccionar 5 como la raíz. Se prestará atención especial al caso en que como la raíz se selecciona a 3. Si se selecciona 3, entonces su subárbol derecho contiene a 4 y 5. Es decir, de nuevo es necesario analizar, en tanto esté implicado un subárbol que contiene a 4 y 5, si debe seleccionarse 4 o 5 como la raíz. Así, se concluye que para encontrar un árbol binario óptimo cuya raíz sea 2, es necesario examinar, entre otros subárboles, los dos árboles siguientes:

e) Un subárbol que contenga a 4 y 5 con 4 como la raíz.
f) Un subárbol que contenga a 4 y 5 con 5 como la raíz.

Resulta obvio que *c*) es equivalente a *e*) y que *d*) es equivalente a *f*). Esto significa que el trabajo realizado para encontrar un árbol binario óptimo con 3 como la raíz también es de utilidad para encontrar un árbol binario óptimo con 2 como la raíz. Con un razonamiento semejante puede verse que en la búsqueda de un árbol binario óptimo con 4 como la raíz es necesario encontrar un árbol binario óptimo que contenga a 1 y 2, con 1 o 2 como la raíz. De nuevo, como ya se indicó, este trabajo es necesario para encontrar un árbol binario óptimo con 3 como la raíz.

En resumen, se observa que dados $a_1, a_2, \ldots, a_n$, el trabajo necesario para encontrar un árbol binario óptimo con, por ejemplo, a_i, como la raíz, quizá también sea necesario para encontrar un árbol binario óptimo con, por ejemplo, a_j, como la raíz. Este principio puede aplicarse a todos los subárboles a todos los niveles. En consecuencia, la programación dinámica debe resolver el problema del árbol binario óptimo trabajando de abajo arriba. Es decir, se empieza por construir árboles binarios óptimos pequeños. Usando estos árboles binarios óptimos pequeños, se construyen árboles binarios óptimos cada vez más grandes. El objetivo se alcanza cuando se encuentra un árbol binario óptimo que contiene a todos los identificadores.

Especificando. Suponga que es necesario encontrar un árbol binario óptimo para cuatro identificadores: 1, 2, 3 y 4. En lo que sigue, se usará la notación $(a_k, a_i \rightarrow a_j)$ para denotar un árbol binario óptimo que contiene los identificadores a_i a a_j y cuya raíz es a_k. También se usará $(a_i \rightarrow a_j)$ para denotar un árbol binario óptimo que contiene los identificadores a_i a a_j. El proceso de programación dinámica para encontrar un árbol binario óptimo que contiene a 1, 2, 3 y 4 se describe a continuación:

1. Se empieza por encontrar lo siguiente:
   $(1, 1 \rightarrow 2)$
   $(2, 1 \rightarrow 2)$
   $(2, 2 \rightarrow 3)$
   $(3, 2 \rightarrow 3)$
   $(3, 3 \rightarrow 4)$
   $(4, 3 \rightarrow 4)$.
2. Al usar los resultados anteriores es posible determinar
   $(1 \rightarrow 2)$ (Determinado por $(1, 1 \rightarrow 2)$ y $(2, 1 \rightarrow 2)$)
   $(2 \rightarrow 3)$
   $(3 \rightarrow 4)$.
3. Luego se encuentra
   $(1, 1 \rightarrow 3)$ (Determinado por $(2 \rightarrow 3)$)
   $(2, 1 \rightarrow 3)$
   $(3, 1 \rightarrow 3)$
   $(2, 2 \rightarrow 4)$
   $(3, 2 \rightarrow 4)$
   $(4, 2 \rightarrow 4)$.
4. Al usar los resultados anteriores es posible determinar
   $(1 \rightarrow 3)$ (Determinado por $(1, 1 \rightarrow 3)$, $(2, 1 \rightarrow 3)$ y $(3, 1 \rightarrow 3)$)
   $(2 \rightarrow 4)$.

5. Luego se encuentra
   $(1, 1 \rightarrow 4)$ (Determinado por $(2 \rightarrow 4)$)
   $(2, 1 \rightarrow 4)$
   $(3, 1 \rightarrow 4)$
   $(4, 1 \rightarrow 4)$.
6. Finalmente, es posible determinar
   $(1 \rightarrow 4)$
   porque está determinado por
   $(1, 1 \rightarrow 4)$
   $(2, 1 \rightarrow 4)$
   $(3, 1 \rightarrow 4)$
   $(4, 1 \rightarrow 4)$.

Cualquiera de los cuatro árboles binarios anteriores con el costo mínimo es óptimo.

El lector puede darse cuenta de que la programación dinámica constituye una manera eficiente para resolver el problema del árbol binario óptimo. No sólo ofrece una forma de pensamiento sistemática, sino que también evita cálculos redundantes.

Una vez que se han descrito los principios básicos para resolver el problema del árbol binario óptimo aplicando el método de programación dinámica, ahora es posible proporcionar el mecanismo preciso. Debido a que la búsqueda de un árbol binario óptimo consiste en encontrar muchos subárboles binarios óptimos, simplemente se considera el caso general de encontrar un subárbol arbitrario. Suponga que se tiene una secuencia de identificadores, $a_1, a_2, \ldots, a_n$ y a_k y que a_k es uno de ellos. Si a_k se selecciona como la raíz del árbol binario, entonces el subárbol izquierdo (derecho) contiene a todos los identificadores $a_1, \ldots, a_{k-1}(a_{k+1}, \ldots, a_n)$. Además, la recuperación de a_k requiere de un paso y todas las otras búsquedas exitosas requieren un paso más que los pasos necesarios para buscar ya sea en el subárbol izquierdo, o derecho. Esto también es cierto para todas las búsquedas infructuosas.

Sea $C(i, j)$ el costo de un árbol binario óptimo que contiene desde a_i hasta a_j. Entonces, el árbol binario óptimo cuya raíz es a_k tiene el costo siguiente

$$C(1,n) = \min_{1 \le k \le n} \left\{ P_k + \left[Q_0 + \sum_{i=1}^{k-1} (P_i + Q_i) + C(1, k-1) \right] + \left[Q_k + \sum_{i=k+1}^{n} (P_i + Q_i) + C(k+1, n) \right] \right\},$$

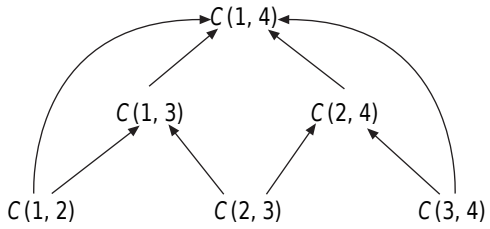
donde P_k es el costo de búsqueda de la raíz y los términos segundo y tercero son los costos de búsqueda para el subárbol izquierdo y el subárbol derecho, respectivamente. La fórmula anterior puede generalizarse para obtener cualquier $C(i, j)$ como sigue:

$$C(i,j) = \min_{i \le k \le j} \left\{ P_k + \left[Q_{i-1} + \sum_{l=i}^{k-1} (P_l + Q_l) + C(i, k-1) \right] + \left[Q_k + \sum_{l=k+1}^{j} (P_l + Q_l) + C(k+1, j) \right] \right\}$$

$$= \min_{i \le k \le j} \left\{ C(i, k-1) + C(k+1, j) + \sum_{l=i}^{j} (P_l + Q_l) + Q_{i-1} \right\}.$$

Por ejemplo, si se tienen cuatro identificadores a_1, a_2, a_3 y a_4, el objetivo es encontrar $C(1, 4)$. Debido a que para la raíz hay cuatro opciones posibles, es necesario calcular el costo de cuatro subárboles óptimos, $C(2, 4)$ (a_1 como la raíz), $C(3, 4)$ (a_2 como la raíz), $C(1, 2)$ (a_3 como la raíz) y $C(1, 3)$ (a_4 como la raíz). Para calcular $C(2, 4)$, se requieren $C(3, 4)$ (a_2 como la raíz) y $C(2, 3)$ (a_4 como la raíz), y para calcular $C(1, 3)$, se requieren $C(2, 3)$ (a_1 como la raíz) y $C(1, 2)$ (a_3 como la raíz). Sus relaciones pueden ilustrarse como se muestra en la figura 7–24.

FIGURA 7-24 Relaciones de cálculo de subárboles.



En general, dados n identificadores para calcular $C(1, n)$, puede procederse calculando primero todos los $C(i, j)$ con $j - i = 1$. Luego, pueden calcularse todos los $C(i, j)$ con $j - i = 2$, luego todos los $C(i, j)$ con $j - i = 3$ y así sucesivamente. A continuación se analiza la complejidad de este procedimiento. Este procedimiento de evaluación requiere el cálculo de $C(i, j)$ para $j - i = 1, 2, \ldots, n - 1$. Cuando $j - i = m$, hay que calcular $(n - m)$ $C(i, j)$. El cálculo de cada uno de estos $C(i, j)$ requiere encontrar el mínimo de m cantidades. Así, cada $C(i, j)$ con $j - i = m$ puede calcularse en tiempo $O(m)$. El tiempo total para calcular todos los $C(i, j)$ con $j - i = m$ es $O(mn - m^2)$. En

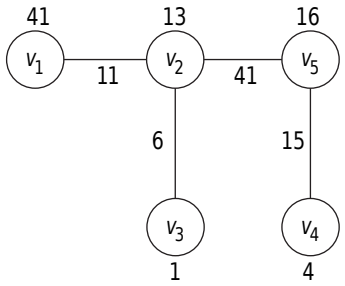
consecuencia, la complejidad temporal total de este método de programación dinámica para resolver el problema del árbol binario óptimo es $O(\sum_{1 \le m \le n} (nm - m^2)) = O(n^3)$.

7-7 EL PROBLEMA PONDERADO DE DOMINACIÓN PERFECTA EN ÁRBOLES

En este problema se tiene una gráfica $G = (V, E)$ donde cada vértice $v \in V$ tiene un costo $c(v)$ y cada arista $e \in E$ también tiene un costo $c(e)$. Un conjunto dominante perfecto de G es un subconjunto D de vértices de V tal que todo vértice que no está en D es adyacente a exactamente un vértice en D. El costo de un conjunto dominante perfecto D incluye todos los costos de los vértices en D y el costo de $c(u, v)$ si v no pertenece a D, u pertenece a D y (u, v) es un arista en E. El problema de dominación perfecta consiste en encontrar un conjunto dominante perfecto con un costo mínimo.

Considere la figura 7-25. Hay muchos conjuntos dominantes perfectos.

FIGURA 7-25 Una gráfica que ilustra el problema ponderado de dominación perfecta.



Por ejemplo, $D_1 = \{v_1, v_2, v_5\}$ es un conjunto dominante perfecto. El costo de D_1 es

$$\begin{aligned} &c(v_1) + c(v_2) + c(v_5) + c(v_3, v_2) + c(v_4, v_5) \\ &= 41 + 13 + 16 + 6 + 15 \\ &= 91. \end{aligned}$$

Otro ejemplo es $D_2 = \{v_2, v_5\}$. El costo de D_2 es

$$\begin{aligned} &c(v_2) + c(v_5) + c(v_1, v_2) + c(v_3, v_2) + c(v_4, v_5) \\ &= 13 + 16 + 11 + 6 + 15 \\ &= 61. \end{aligned}$$

Finalmente, sea D_3 el conjunto $\{v_2, v_3, v_4, v_5\}$. El costo de D_3 es

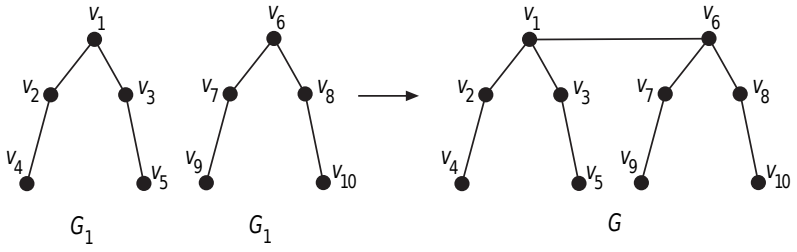
$$c(v_2) + c(v_3) + c(v_4) + c(v_5) + c(v_1, v_2)$$
$$= 13 + 1 + 4 + 16 + 11$$
$$= 45.$$

Puede demostrarse que D_3 es un conjunto dominante perfecto con costo mínimo.

El problema ponderado de dominación perfecta es NP-difícil para gráficas bipartitas y gráficas cordales. Sin embargo, se demostrará que al aplicar el método de programación dinámica es posible resolver el problema ponderado de dominación perfecta en árboles.

La tarea fundamental al aplicar la estrategia de programación dinámica consiste en fusionar dos soluciones factibles en una nueva solución factible. A continuación, la fusión se ilustrará con un ejemplo. Considere la figura 7-27. Ahí hay dos gráficas: G_1 y G_2, que serán fusionadas en G al unir v_1 de G_1 con v_6 de G_2.

FIGURA 7-26 Un ejemplo que ilustra el esquema de fusión al resolver el problema ponderado de dominación perfecta.



Debido a que G_1 y G_2 se combinan al unir v_1 de G_1 con v_6 de G_2, puede considerarse que G_1 está arraigado en v_1 y que G_2 está arraigado en v_6. A continuación se considerarán seis conjuntos dominantes perfectos:

$$D_{11} = \{v_1, v_2, v_3\},\ D_{21} = \{v_6, v_7, v_8\}$$
$$D_{12} = \{v_3, v_4\},\ D_{22} = \{v_7, v_{10}\}$$
$$D_{13} = \{v_4, v_5\},\ D_{23} = \{v_9, v_{10}\}.$$

$D_{11}(D_{21})$ es un conjunto dominante perfecto de $G_1(G_2)$ a condición de que la raíz de $G_1(G_2)$, $v_1(v_6)$ esté incluida en él.

$D_{12}(D_{22})$ es un conjunto dominante perfecto de $G_1(G_2)$ a condición de que la raíz de $G_1(G_2)$, $v_1(v_6)$ no esté incluida en él.

$D_{13}(D_{23})$ es un conjunto dominante perfecto de $G_1 - \{v_1\}(G_2 - \{v_6\})$ a condición de que ninguno de los vecinos v_1 (v_6) esté incluido en él.

A hora es posible producir conjuntos dominantes perfectos para G al combinar los conjuntos dominantes perfectos anteriores para G_1 y G_2.

1. $D_{11} \cup D_{21} = \{v_1, v_2, v_3, v_6, v_7, v_8\}$ es un conjunto dominante perfecto de G. Su costo es la suma de los costos de D_{11} y D_{21}. Este conjunto dominante perfecto incluye tanto a v_1 como a v_6.
2. $D_{11} \cup D_{23} = \{v_1, v_2, v_3, v_9, v_{10}\}$ es un conjunto dominante perfecto de G. Su costo es la suma de los costos de D_{11} y D_{23} más el costo de la arista que une v_1 con v_6. Este conjunto dominante perfecto incluye a v_1 y no incluye a v_6.
3. $D_{12} \cup D_{22} = \{v_3, v_4, v_7, v_{10}\}$ es un conjunto dominante perfecto de G. Su costo es la suma de los costos de D_{12} y D_{22}. Este conjunto dominante perfecto no incluye a v_1 ni a v_6.
4. $D_{13} \cup D_{21} = \{v_4, v_5, v_6, v_7, v_8\}$ es un conjunto dominante perfecto de G. Su costo es la suma de los costos de D_{13} y D_{21} más el costo de la arista que une v_1 con v_6. Este conjunto dominante perfecto no incluye a v_1 pero sí incluye a v_6.

El análisis anterior describe la base de la estrategia de aplicación del método de programación dinámica para resolver el problema ponderado de dominación perfecta.

En lo que sigue, primero se supondrá que un árbol está arraigado en cierto vértice y al combinar las dos gráficas, se unen las dos raíces.

Se define lo siguiente:

$D_1(G, u)$: un conjunto dominante perfecto óptimo para la gráfica G a condición de que el conjunto dominante perfecto incluya a u. El costo de $D_1(G, u)$ se denota por $\delta_1(G, u)$.

$D_2(G, u)$: un conjunto dominante perfecto óptimo para la gráfica G a condición de que el conjunto dominante perfecto no incluya a u. El costo de $D_2(G, u)$ se denota por $\delta_2(G, u)$.

$D_3(G, u)$: un conjunto dominante perfecto óptimo para la gráfica $G - \{u\}$ a condición de que el conjunto dominante perfecto de $G - \{u\}$ no incluya ningún vértice vecino de u. El costo de $D_3(G, u)$ se denota por $\delta_3(G, u)$.

Dadas dos gráficas G_1 y G_2 arraigadas en u y v respectivamente, sea G la gráfica que se obtiene al unir u y v. Así, resulta evidente que se tendrán las reglas siguientes:

REGLA 1: Tanto $D_1(G_1, u) \cup D_1(G_2, v)$ como $D_1(G_1, u) \cup D_3(G_2, v)$ son conjuntos dominantes perfectos de G que incluyen a u.

REGLA 1.1: El costo de $D_1(G_1, u) \cup D_1(G_2, v)$ es $\delta_1(G_1, u) + \delta_1(G_2, v)$.
REGLA 1.2: El costo de $D_1(G_1, u) \cup D_3(G_2, v)$ es $\delta_1(G_1, u) + \delta_3(G_2, v) + c(u, v)$.

REGLA 2: Tanto $D_2(G_1, u) \cup D_2(G_2, v)$ como $D_3(G_1, u) \cup D_1(G_2, v)$ son conjuntos dominantes perfectos de G que no incluyen a u.

REGLA 2.1: El costo de $D_2(G_1, u) \cup D_2(G_2, v)$ es $\delta_2(G_1, u) + \delta_2(G_2, v)$.
REGLA 2.2: El costo de $D_3(G_1, u) \cup D_1(G_2, v)$ es $\delta_3(G_1, u) + \delta_1(G_2, v) + c(u, v)$.

El principio fundamental de aplicar la estrategia de la programación dinámica para encontrar un conjunto dominante perfecto óptimo de una gráfica puede resumirse como sigue:

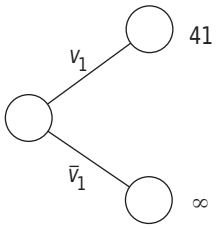
Si una gráfica G puede descomponerse en dos subgráficas G_1 y G_2 de modo que G puede reconstruirse uniendo u de G_1 y v de G_2, entonces un conjunto dominante perfecto de G, denotado por $D(G)$, puede encontrarse como sigue:

1. Si $\delta_1(G_1, u) + \delta_1(G_2, v)$ es menor que $\delta_1(G_1, u) + \delta_3(G_2, v) + c(u, v)$, entonces se hace que $D_1(G, u)$ sea $D_1(G_1, u) \cup D_1(G_2, v)$ y $\delta_1(G, u)$ sea $\delta_1(G_1, u) + \delta_1(G_2, v)$. En caso contrario, se hace que $D_1(G, u)$ sea $D_1(G_1, u) \cup D_3(G_2, v)$ y $\delta_1(G, u)$ sea $\delta_1(G_1, u) + \delta_3(G_2, v) + c(u, v)$.
2. Si $\delta_2(G_1, u) + \delta_2(G_2, v)$ es menor que $\delta_3(G_1, u) + \delta_1(G_2, v) + c(u, v)$, entonces se hace que $D_2(G, u)$ sea $D_2(G_1, u) \cup D_2(G_2, v)$ y $\delta_2(G, u)$ sea $\delta_2(G_1, u) + \delta_2(G_2, v)$. En caso contrario, se hace que $D_2(G, u)$ sea $D_3(G_1, u) \cup D_1(G_2, v)$ y $\delta_2(G, u)$ sea $\delta_3(G_1, u) + \delta_1(G_2, v) + c(u, v)$.
3. $D_3(G, u) = D_3(G_1, u) \cup D_2(G_2, v)$.
   $\delta_3(G, u) = \delta_3(G_1, u) + \delta_2(G_2, v)$.
4. Si $\delta_1(G, u)$ es menor que $\delta_2(G, u)$, se hace que $D(G)$ sea $D_1(G, u)$. En caso contrario, se hace que $D(G)$ sea $D_2(G, u)$.

Las reglas anteriores son para gráficas generales. En los párrafos siguientes, se abordarán los árboles. Para árboles se usará un algoritmo especial que empieza a partir de los nodos hoja y trabaja hacia la parte interna del árbol. Antes de presentar el algoritmo especial para árboles, el algoritmo se ilustrará con un ejemplo completo. Considere nuevamente la figura 7-25. Se empezará sólo desde el nodo hoja v_1. Sólo hay dos

casos: el conjunto dominante perfecto contiene a v_1 o no contiene a v_1, como se muestra en la figura 7-27.

FIGURA 7-27 Cálculo del conjunto dominante perfecto que implica a v_1.



Resulta fácil ver que

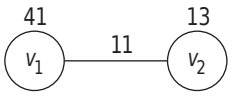
$D_1(\{v_1\}, v_1) = \{v_1\}$,
$D_2(\{v_1\}, v_1)$ no existe,
$D_3(\{v_1\}, v_1) = \phi$,
$\delta_1(\{v_1\}, v_1) = c(v_1) = 41$,
$\delta_2(\{v_1\}, v_1) = \infty$,
y $\delta_3(\{v_1\}, v_1) = 0$.

A continuación se considerará el subárbol que sólo contiene a v_2. De nuevo, es posible ver lo siguiente:

$D_1(\{v_2\}, v_2) = \{v_2\}$,
$D_2(\{v_2\}, v_2)$ no existe,
$D_3(\{v_2\}, v_2) = \phi$,
$\delta_1(\{v_2\}, v_2) = c(v_2) = 13$,
$\delta_2(\{v_2\}, v_2) = \infty$,
y $\delta_3(\{v_2\}, v_2) = 0$.

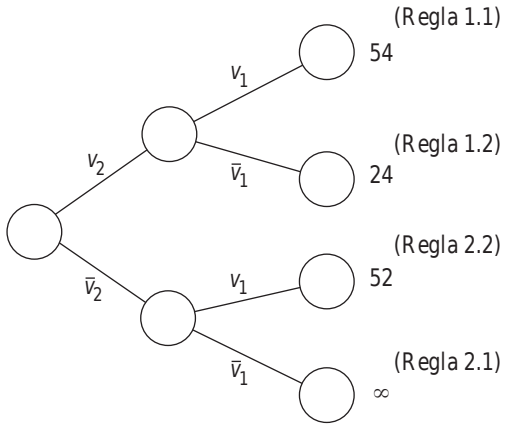
Considere el subárbol que contiene a v_1 y v_2, que se muestra en la figura 7-28.

FIGURA 7-28 Subárbol que contiene a v_1 y v_2.



El cálculo del conjunto dominante perfecto para $\{v_1, v_2\}$, mediante programación dinámica, se ilustra en la figura 7-29.

FIGURA 7-29 Cálculo del conjunto dominante perfecto del subárbol que contiene a v_1 y v_2.



Debido a que mín(54, 24) = 24, se tiene

$D_1(\{v_1, v_2\}, v_2) = \{v_2\}$,
$\delta_1(\{v_1, v_2\}, v_2) = 24$.

Debido a que mín(∞, 52) = 52 se tiene

$D_2(\{v_2, v_1\}, v_2) = \{v_1\}$,
$\delta_2(\{v_2, v_1\}, v_2) = 52$.

Además,

$D_3(\{v_1, v_2\}, v_2)$ no existe,
$\delta_3(\{v_1, v_2\}, v_2) = \infty$.

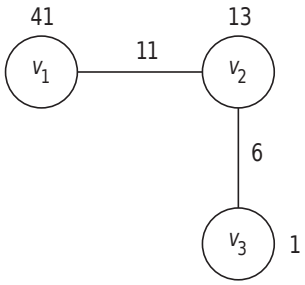
Considere el árbol que sólo contiene a v_3. Resulta evidente que

$D_1(\{v_3\}, v_3) = \{v_3\}$,
$D_2(\{v_3\}, v_3)$ no existe,

$D_3(\{v_3\}, v_3) = \phi$,
$\delta_1(\{v_3\}, v_3) = c(v_3) = 1$,
$\delta_2(\{v_3\}, v_3) = \infty$,
y $\delta_3(\{v_3\}, v_3) = 0$.

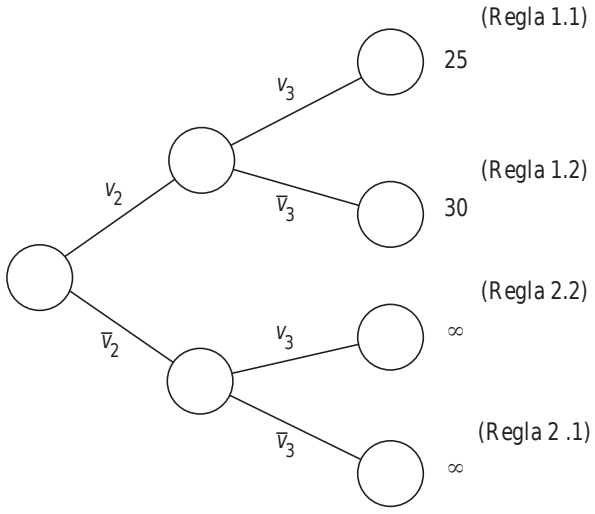
Al subárbol que contiene a v_1 y v_2 se agrega v_3, lo cual se muestra en la figura 7-30.

FIGURA 7-30 Subárbol que contiene a v_1, v_2 y v_3.



El cálculo del conjunto dominante perfecto para este subárbol se ilustra en la figura 7-31.

FIGURA 7-31 Cálculo del conjunto dominante perfecto del subárbol que contiene a v_1, v_2 y v_3.

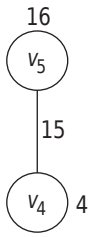


Con base en el cálculo, se tiene

$D_1(\{v_1, v_2, v_3\}, v_2) = \{v_2, v_3\}$,
$D_2(\{v_1, v_2, v_3\}, v_2)$ no existe,
$D_3(\{v_1, v_2, v_3\}, v_2)$ no existe,
$\delta_1(\{v_1, v_2, v_3\}, v_2) = c(v_2) + c(v_3) + c(v_1, v_2) = 13 + 1 + 11 = 25$,
$\delta_2(\{v_1, v_2, v_3\}, v_2) = \infty$,
y $\delta_3(\{v_1, v_2, v_3\}, v_2) = \infty$.

Considere el subárbol que contiene a v_5 y v_4, que se muestra en la figura 7-32.

FIGURA 7-32 Subárbol que contiene a v_5 y v_4.

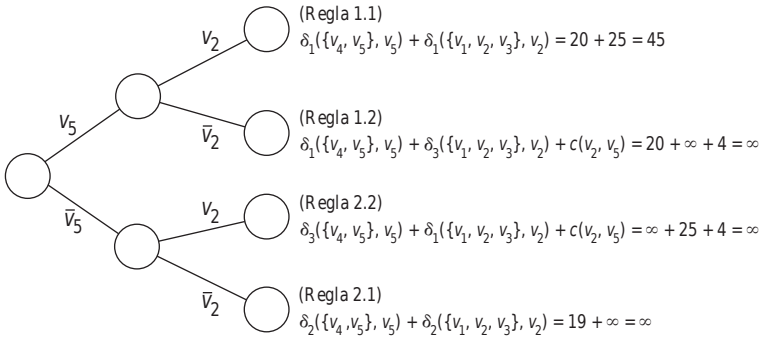


Fácilmente puede verse que lo siguiente es cierto.

$D_1(\{v_5, v_4\}, v_5) = \{v_5, v_4\}$,
$D_2(\{v_5, v_4\}, v_5) = \{v_4\}$,
$D_3(\{v_5, v_4\}, v_5)$ no existe,
$\delta_1(\{v_5, v_4\}, v_5) = c(v_5) + c(v_4) = 16 + 4 = 20$,
$\delta_2(\{v_5, v_4\}, v_5) = c(v_4) + c(v_5, v_4) = 4 + 15 = 19$,
y $\delta_3(\{v_5, v_4\}, v_5) = \infty$.

Ahora se considerará el subárbol que contiene a v_1, v_2 y v_3, así como el subárbol que contiene a v_4 y v_5. Es decir, ahora se considerará todo el árbol. El cálculo de este conjunto dominante perfecto se muestra en la figura 7-33.

FIGURA 7-33 Cálculo del conjunto dominante perfecto de todo el árbol que se muestra en la figura 7-25.



Puede concluirse que el conjunto dominante perfecto con costo mínimo es $\{v_2, v_3, v_4, v_5\}$, cuyo costo es 45.

En el siguiente algoritmo, sea $TP(u)$ el árbol parcialmente construido en el proceso y que está arraigado en u.

Algoritmo 7-2 □ Un algoritmo para resolver el problema ponderado de dominación perfecta en árboles

Input: Un árbol $T = (V, E)$ donde todo vértice $v \in V$ tiene un costo $c(v)$ y toda arista $e \in E$ tiene un costo $c(e)$.

Output: Un conjunto dominante perfecto $D(T)$ de T con costo mínimo.

Paso 1: Para todo vértice $v \in V$, hacer

$TP(v) = \{v\}$

$D_1(TP(v), v) = TP(v)$

$\delta_1(TP(v), v) = c(v)$

$D_2(TP(v), v)$ no existe

$\delta_2(TP(v), v) = \infty$

$D_3(TP(v), v) = \phi$

$\delta_3(TP(v), v) = 0$.

Paso 2: $T' = T$.

Paso 3: Mientras T' tiene más de un vértice, hacer:
escoger una hoja v de T' que sea adyacente a un único vértice u en T'.
Hacer $TP'(u) = TP(u) \cup TP(v) \cup \{u, v\}$.
Si $(\delta_1(TP(u), u) + \delta_1(TP(v), v)) < (\delta_1(TP(u), u) + \delta_3(TP(v), v) + c(u, v))$
$D_1(TP'(u), u) = D_1(TP(u), u) \cup D_1(TP(v), v)$
$\delta_1(TP'(u), u) = \delta_1(TP(u), u) + \delta_1(TP(v), v)$.
En caso contrario,
$D_1(TP'(u), u) = D_1(TP(u), u) \cup D_3(TP(v), v)$
$\delta_1(TP'(u), u) = \delta_1(TP(u), u) + \delta_3(TP(v), v) + c(u, v)$.
Si $(\delta_2(TP(u), u) + \delta_2(TP(v), v)) < (\delta_3(TP(u), u) + \delta_1(TP(v), v) + c(u, v))$
$D_2(TP'(u), u) = D_2(TP(u), u) \cup D_2(TP(v), v)$
$\delta_2(TP'(u), u) = \delta_2(TP(u), u) + \delta_2(TP(v), v)$.
En caso contrario,
$D_2(TP'(u), u) = D_3(TP(u), u) \cup D_1(TP(v), v)$
$\delta_2(TP'(u), u) = \delta_3(TP(u), u) + \delta_1(TP(v), v) + c(u, v)$
$D_3(TP'(u), u) = D_3(TP(u), u) \cup D_2(TP(v), v)$
$\delta_3(TP'(u), u) = \delta_3(TP(u), u) + \delta_2(TP(v), v)$
$TP(u) = TP'(u)$
$T' = T' - v$;
end while.

Paso 4: Si $\delta_1(TP(u), u) < \delta_2(TP(u), u)$
Hacer $D(T) = D_1(TP(u), u)$.
En caso contrario,
hacer $D(T) = D_2(TP(u), u)$.
Regresar $D(T)$ como el conjunto dominante perfecto mínimo de $T(V, E)$.

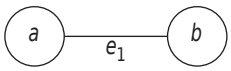
7-8 EL PROBLEMA PONDERADO DE BÚSQUEDA DE BORDES EN UNA GRÁFICA EN UN SOLO PASO

En este problema se tiene una gráfica no dirigida simple $G = (V, E)$ donde cada vértice $v \in V$ está asociado con un peso $wt(v)$. Todos los bordes de G tienen la misma longitud. Se supone que en alguna arista de G está escondido un fugitivo que puede moverse a cualquier velocidad. En cada arista de G, para buscar al fugitivo se asigna un buscador de aristas. El buscador siempre empieza desde un vértice. El costo de recorrer una arista (u, v) se define como $wt(u)$ si el buscador empieza desde u y como $wt(v)$ si empieza desde v. Suponga que todos los buscadores de aristas se desplazan a la misma velocidad. El problema ponderado de búsqueda de aristas en una gráfica simple en un solo paso consiste en disponer las direcciones de búsqueda de los buscadores de aristas de modo que el fugitivo sea atrapado en un paso y se minimice el número de buscadores utilizados.

Resulta evidente que si hay m aristas, se requieren por lo menos m buscadores de aristas. Debido a que el fugitivo puede moverse tan rápido como los buscadores de aristas y cada uno de éstos se desplaza hacia delante, el equipo de buscadores puede capturar al fugitivo en un solo paso si éste no puede escabullirse a través de ningún vértice ya explorado y esconderse detrás de los buscadores de aristas.

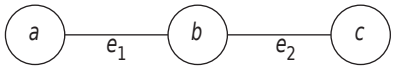
Se considerará la figura 7-34. En este caso sólo se requiere un buscador. No importa dónde esté ubicado inicialmente el buscador, el fugitivo no puede escapar. Suponga que el buscador se encuentra en a y busca en dirección de b. Cuando mucho, el fugitivo puede llegar a b. Pero entonces el buscador también llegará a b y lo capturará. De manera semejante, si un buscador está inicialmente en b, es posible alcanzar el mismo objetivo. Es decir, es posible atrapar al fugitivo en un paso.

FIGURA 7-34 Caso en que no se requieren buscadores adicionales.



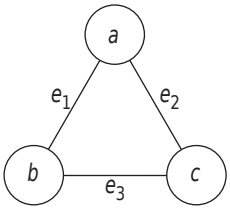
Ahora, considere la figura 7-35. Al colocar buscadores en a y c también es posible atrapar al fugitivo en un paso.

FIGURA 7-35 Otro caso en que no se requieren buscadores adicionales.



A hora considere la figura 7-36.

FIGURA 7-36 Caso en que se requieren buscadores adicionales.



En este caso, si sólo se usan tres buscadores, entonces sin importar cómo se dispongan las direcciones de búsqueda, es posible que el fugitivo evite ser capturado. Por ejemplo, considere el siguiente plan de búsqueda:

1. El buscador en la arista e_1 busca de a a b.
2. El buscador en la arista e_2 busca de a a c.
3. El buscador en la arista e_3 busca de b a c.

Entonces, si el fugitivo está originalmente en e_1, puede pasar desapercibido si se desplaza a e_3.

Suponga que en el vértice b se coloca un buscador adicional y que se aplica el mismo plan de búsqueda ya presentado. Entonces sin importar dónde estaba originalmente el fugitivo, será atrapado. Esto demuestra que en algunos casos se requieren buscadores adicionales.

Un plan de búsqueda en un solo paso consiste en disponer las direcciones de búsqueda de los buscadores de bordes y determinar el número mínimo de buscadores adicionales requeridos. El costo de un plan de búsqueda en un solo paso es la suma de los costos de todos los buscadores. El problema ponderado de búsqueda de bordes en una gráfica en un solo paso consiste en encontrar un plan de búsqueda en un solo paso factible con costo mínimo. Para el ejemplo anterior, el costo de este plan de búsqueda es $2wt(a) + wt(b) + wt(b)$.

El problema ponderado de búsqueda de bordes en una gráfica en un solo paso es NP-difícil para gráficas generales. Sin embargo, se demostrará que al aplicar la estrategia de la programación dinámica, el problema ponderado de búsqueda de bordes en una gráfica en un solo paso en árboles puede resolverse en tiempo polinomial.

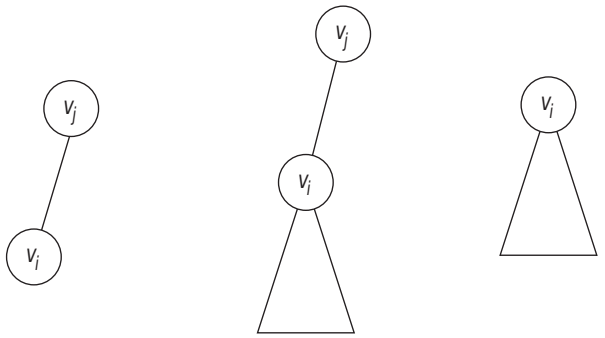
Nuestro método de programación dinámica para resolver este problema se basa en varias reglas básicas que se explicarán a continuación.

Primero se definen algunas notaciones.

Sea v_i el vértice de un árbol.

Caso 1: v_i es un nodo hoja. Entonces $T(v_i)$ denota a v_i y a su nodo padre, lo cual se muestra en la figura 7-37*a*).

FIGURA 7-37 Definición de $T(v_i)$.



Caso 2: v_i es un nodo interno, pero no el nodo raíz. Así, (v_i) denota al árbol que implica a v_i, al nodo padre v_j y a todos los nodos descendientes de v_i, lo cual se muestra en la figura 7-37*b*).

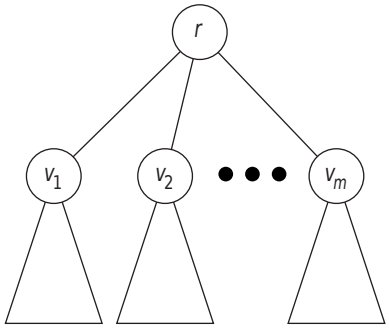
Caso 3: v_i es la raíz de un árbol T. Entonces $T(v_i)$ denota a T, como se muestra en la figura 7-37*c*).

Además, sea v_j el nodo padre de v_i. Entonces $C(T(v_i), v_i, v_j)$ ($C(T(v_i), v_j, v_i)$) denota el costo de un plan de búsqueda óptimo en un solo paso donde la dirección de búsqueda de (v_j, v_i) es de v_i a v_j (de v_j a v_i).

Regla 1: Sea r la raíz de un árbol. Sea $C(T(r), r)(C(T(r), \bar{r}))$ el costo de un plan de búsqueda óptimo en un solo paso para T con (sin) un buscador adicional ubicado en r. Así, el costo de un plan de búsqueda óptimo en un solo paso para T, denotado por $C(T(r))$, es mín$\{C(T(r), r), C(T(r), \bar{r})\}$.

Regla 2: Sea r la raíz de un árbol donde $v_1, v_2, \ldots, v_m$ son descendientes de r (consulte la figura 7-38). Si en r no hay un guardia adicional, entonces todas las direcciones de búsqueda de (r, v_1), (r, v_2), …, (r, v_m) van de r a $v_1, v_2, \ldots, v_m$ o todas van de $v_1, v_2, \ldots, v_m$ a r. Si en r hay un guardia adicional, entonces la dirección de búsqueda de cada (r, v_i) puede determinarse de manera independiente.

FIGURA 7-38 Ilustración de la regla 2.



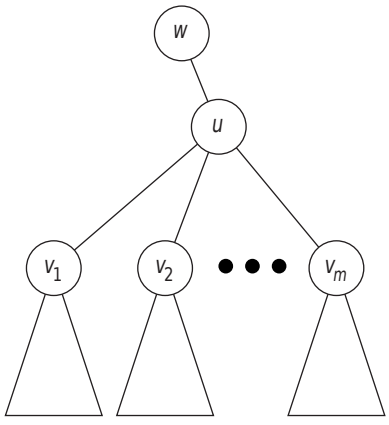
Es decir,

$$C(T(r), \bar{r}) = \text{mín}\left\{\sum_{i=1}^{m} C(T(v_i), r, v_i), \sum_{i=1}^{m} C(T(v_i), v_i, r)\right\}$$

$$C(T(r), r) = wt(r) + \sum_{i=1}^{m} \text{mín}\{C(T(v_i), r, v_i), C(T(v_i), v_i, r)\}.$$

Regla 3: Sean u un nodo interno, w su nodo padre y $v_1, v_2, \ldots, v_m$ sus nodos descendientes (figura 7-39).

FIGURA 7-39 Ilustración de la regla 3.



Si en u no hay un guardia adicional, entonces si la dirección de búsqueda de (w, u) es de w a u (de u a w), todas las direcciones de búsqueda de (u, v_1), (u, v_2), ..., (u, v_m) son de $v_1, v_2, \ldots, v_m$ a u (de u a $v_1, v_2, \ldots, v_m$). Si en u hay un guardia adicional, entonces las direcciones de búsqueda de (u, v_j) pueden determinarse de manera independiente. Sea $C(T(u), w, u, \bar{u})$ $(C(T(u), w, u, u))$ el costo de un plan de búsqueda óptimo en un solo paso donde la dirección de búsqueda de la arista (u, w) es de w a u y en el vértice u no hay un guardia adicional (en el vértice u hay un guardia adicional). De manera semejante, sea $C(T(u), u, w, \bar{u})$ $(C(T(u), u, w, u)$ el costo de un plan de búsqueda óptimo en un solo paso donde la dirección de búsqueda de la arista (u, w) es de u a w y en el vértice u no hay un guardia adicional (en el vértice u hay un guardia adicional). Entonces se tienen las fórmulas siguientes.

$$C(T(u), w, u, \bar{u}) = wt(w) + \sum_{i=1}^{m} C(T(v_i), v_i, u),$$

$$C(T(u), w, u, u) = wt(w) + wt(u) + \sum_{i=1}^{m} \text{mín}\{C(T(v_i), v_i, u), C(T(v_i), u, v_i)\},$$

$$C(T(u), u, w, \bar{u}) = wt(u) + \sum_{i=1}^{m} C(T(v_i), u, v_i),$$

y $$C(T(u), u, w, u) = 2wt(u) + \sum_{i=1}^{m} \text{mín}\{C(T(v_i), v_i, u), C(T(v_i), u, v_i)\}.$$

Entonces,

$$C(T(u), w, u) = \text{mín}\{C(T(u), w, u, \bar{u}), C(T(u), w, u, u)\},$$
$$C(T(u), u, w) = \text{mín}\{C(T(u), u, w, \bar{u}), C(T(u), u, w, u)\}.$$

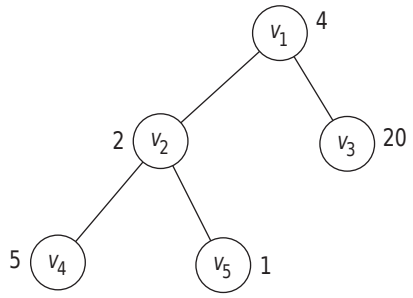
Finalmente, se tiene una regla que concierne a las condiciones en la frontera.

Regla 4: Si u es un nodo hoja y w es su nodo padre, entonces $C(T(u), w, u) = wt(w)$ y $C(T(u), u, w) = wt(u)$.

Nuestro método de programación dinámica resuelve el problema ponderado de búsqueda de bordes en una gráfica en un solo paso al trabajar desde los nodos hoja y gradualmente hacerlo hacia la raíz del árbol. Si un nodo es una hoja, entonces se aplica la regla 4. Si se trata de un nodo interno, pero no es la raíz, entonces se aplica la regla 3. Si es el nodo raíz, entonces primero se aplica la regla 2 y luego la regla 1.

El método se ilustrará con un ejemplo. Considere el árbol ponderado en la figura 7-40.

FIGURA 7-40 Árbol que ilustra el método de programación dinámica.



El método de programación dinámica puede trabajar como sigue:

Paso 1: Se escoge v_3, que es un nodo hoja. Se aplica la regla 4. El nodo padre de v_3 es v_1.

$C(T(v_3), v_1, v_3) = wt(v_1) = 4$

$C(T(v_3), v_3, v_1) = wt(v_3) = 20.$

Paso 2: Se escoge v_4, que es un nodo hoja. Se aplica la regla 4. El nodo padre de v_4 es v_2.

$C(T(v_4), v_2, v_4) = wt(v_2) = 2$

$C(T(v_4), v_4, v_2) = wt(v_4) = 5.$

Paso 3: Se escoge v_5, que es un nodo hoja. Se aplica la regla 4. El nodo padre de v_5 es v_2.

$C(T(v_5), v_2, v_5) = wt(v_2) = 2$

$C(T(v_5), v_5, v_2) = wt(v_5) = 1.$

Paso 4: Se escoge v_2, que es un nodo interno, pero no la raíz. Se aplica la regla 3. El nodo padre de v_2 es v_1. Los nodos descendientes de v_2 son v_4 y v_5.

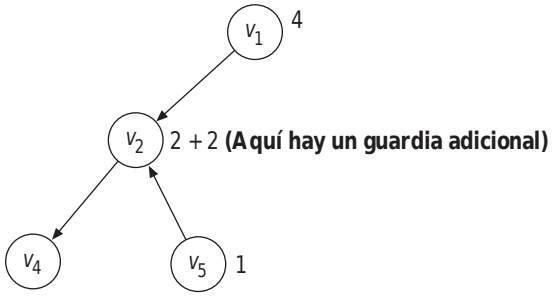
$$C(T(v_2), v_1, v_2) = \text{mín}\{C(T(v_2), v_1, v_2, \overline{v_2}), C(T(v_2), v_1, v_2, v_2)\}$$

$$\begin{aligned} C(T(v_2), v_1, v_2, \overline{v_2}) &= wt(v_1) + C(T(v_4), v_4, v_2) + C(T(v_5), v_5, v_2) \\ &= 4 + 5 + 1 \\ &= 10 \end{aligned}$$

$$
\begin{aligned}
&C(T(v_2), v_1, v_2, \overline{v}_2)\\
&= wt(v_1) + wt(v_2) + \text{mín}\{C(T(v_4), v_2, v_4), C(T(v_4), v_4, v_2)\}\\
&\quad + \text{mín}\{C(T(v_5), v_2, v_5), C(T(v_5), v_5, v_2)\}\\
&= 4 + 2 + \text{mín}\{2, 5\} + \text{mín}\{2, 1\}\\
&= 6 + 2 + 1\\
&= 9\\
&C(T(v_2), v_1, v_2) = \text{mín}\{10, 9\} = 9.
\end{aligned}
$$

Observe que el cálculo anterior indica que si la dirección de búsqueda de (v_1, v_2) es de v_1 a v_2, entonces el plan de búsqueda óptimo es el que se muestra en la figura 7-41. En v_2 hay un guardia adicional.

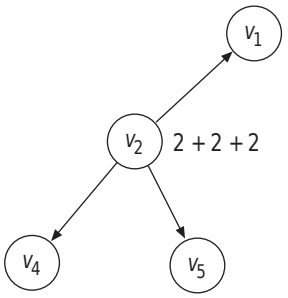
FIGURA 7-41 Plan de búsqueda en un solo paso que implica a v_2.



$$
\begin{aligned}
&C(T(v_2), v_2, v_1) = \text{mín}\{C(T(v_2), v_2, v_1, \overline{v}_2), C(T(v_2), v_2, v_1, v_2)\}\\
&C(T(v_2), v_2, v_1, \overline{v}_2) = wt(v_2) + C(T(v_4), v_2, v_4) + C(T(v_5), v_2, v_5)\\
&\qquad\qquad = 2 + 2 + 2\\
&\qquad\qquad = 6\\
&C(T(v_2), v_2, v_1, v_2)\\
&= wt(v_2) + wt(v_2) + \text{mín}\{C(T(v_4), v_2, v_4), C(T(v_4), v_4, v_2)\}\\
&\quad + \text{mín}\{C(T(v_5), v_2, v_5), C(T(v_5), v_5, v_2)\}\\
&= 2 + 2 + \text{mín}\{2, 5\} + \text{mín}\{2, 1\}\\
&= 4 + 2 + 1\\
&= 7\\
&C(T(v_2), v_2, v_1) = \text{mín}\{6, 7\} = 6.
\end{aligned}
$$

El cálculo anterior indica que si el plan de búsqueda de (v_1, v_2) es de v_2 a v_1, entonces el plan de búsqueda óptimo en un solo paso es el que se muestra en la figura 7-42.

FIGURA 7-42 Otro plan de búsqueda en un solo paso que implica a v_2.

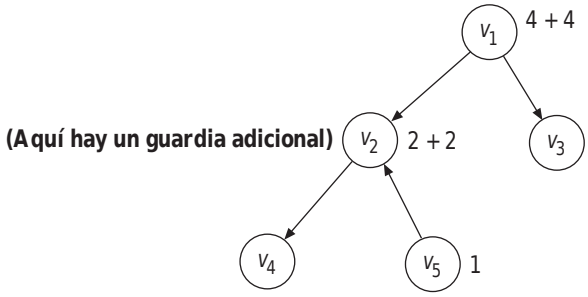


Paso 5. Se escoge v_1. Debido a que v_1 es la raíz, se aplica la regla 2.

$$\begin{aligned} C(T(v_1), \overline{v_1}) &= \text{mín}\{C(T(v_2), v_1, v_2) + C(T(v_3), v_1, v_3), \\ &\qquad C(T(v_2), v_2, v_1) + C(T(v_3), v_3, v_1)\} \\ &= \text{mín}\{9 + 4, 6 + 20\} \\ &= \text{mín}\{13, 26\} \\ &= 13. \end{aligned}$$

Al hacer un rastreo, se comprende que si en v_1 no hay un guardia adicional, entonces el plan de búsqueda óptimo en un solo paso es el que se muestra en la figura 7-43.

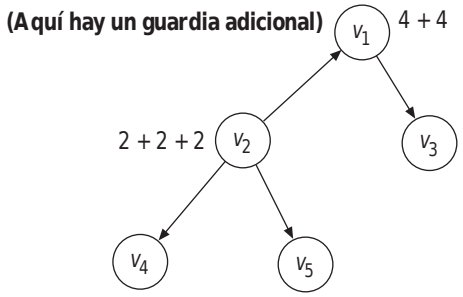
FIGURA 7-43 Plan de búsqueda en un solo paso que implica a v_1.



$$\begin{aligned} C(T(v_1), v_1) &= wt(v_1) + \text{mín}\{C(T(v_2), v_1, v_2), C(T(v_2), v_2, v_1)\} \\ &\quad + \text{mín}\{C(T(v_3), v_1, v_3), C(T(v_3), v_3, v_1)\} \\ &= 4 + \text{mín}\{9, 6\} + \text{mín}\{4, 20\} \\ &= 4 + 6 + 4 \\ &= 14. \end{aligned}$$

Al hacer un rastreo, se comprende que si en v_1 hay un guardia adicional, entonces el plan de búsqueda óptimo en un solo paso es el que se muestra en la figura 7-44.

FIGURA 7-44 Otro plan de búsqueda en un solo paso que implica a v_1.



Finalmente, se aplica la regla 1.

$$\begin{aligned} C(T(v_1)) &= \text{mín}\{C(T(v_1), v_1), C(T(v_1), \overline{v}_1)\} \\ &= \text{mín}\{14, 13\} \\ &= 13. \end{aligned}$$

Esto significa que en v_1 no debe haber ningún guardia adicional y que el plan de búsqueda óptimo es el que se muestra en la figura 7-43.

El número de operaciones sobre cada vértice es dos; una para calcular el costo de búsqueda mínimo y otra para determinar las direcciones de búsqueda en los bordes. Así, el número total de operaciones es $O(n)$, donde n es el número de nodos del árbol. Debido a que para cada operación se requiere tiempo constante, este algoritmo es lineal.

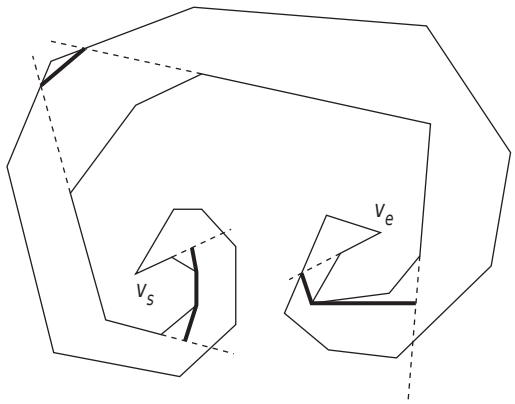
7-9 EL PROBLEMA DE RUTAS DE m-VIGILANTES PARA POLÍGONOS DE 1 ESPIRAL RESUELTO CON EL MÉTODO DE PROGRAMACIÓN DINÁMICA

Este problema se define como sigue. Se tienen un polígono simple y un entero $m \geq 1$ y se requiere encontrar las rutas para m-vigilantes, denominadas las rutas de m-vigilan-

tes, de modo que todo punto del polígono sea visto al menos por un vigilante desde alguna posición en su ruta. El objetivo es minimizar la suma de las longitudes de las rutas. Se ha demostrado que el problema de rutas para *m*-vigilantes para polígonos de 1 espiral es NP-difícil.

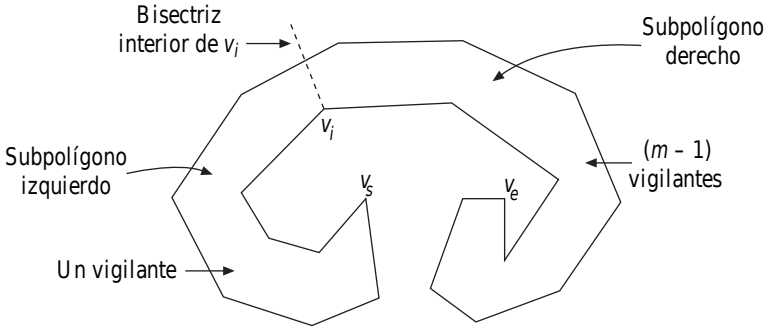
En esta sección se demostrará que el problema de rutas de *m*-vigilantes para polígonos de 1 espiral puede resolverse con el método de programación dinámica. Recuerde que en el capítulo 3 los polígonos de 1 espiral se definieron como un polígono simple cuya frontera puede separarse en una cadena reflex y una cadena convexa. Al atravesar la frontera de un polígono de 1 espiral, los vértices inicial y final de la cadena reflex se denominan v_s y v_e, respectivamente. En la figura 7-45 se muestra una solución del problema de rutas de 3 vigilantes para el polígono de 1 espiral.

FIGURA 7-45 Una solución del problema de rutas de 3 vigilantes para un polígono de 1 espiral.



La idea fundamental es que un polígono de 1 espiral puede separarse en dos partes por medio de una bisectriz interior que parte de un vértice de la cadena reflex. Considere la figura 7-46. Una bisectriz interior que pasa por v_i corta la cadena convexa y separa en dos partes el polígono de 1 espiral, siendo ambas partes de 1 espiral o una de 1 espiral y una convexa. El subpolígono que incluye a v_s se denomina polígono izquierdo y el otro se denomina polígono derecho con respecto a v_i. El vértice v_i se denomina punto de corte. Por conveniencia para el análisis ulterior, la bisectriz interior del primer vértice v_s (el último vértice v_e) en la cadena reflex se define como el primer (último) borde de la cadena convexa.

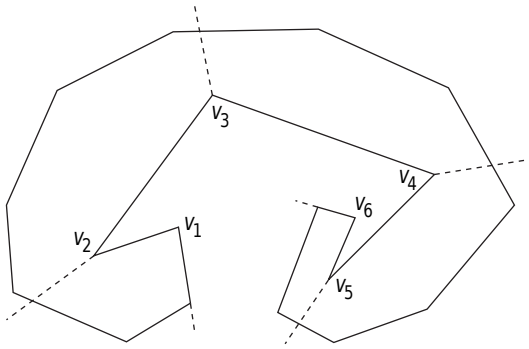
FIGURA 7-46 Idea básica para resolver el problema de rutas de *m*-vigilantes.



Los *m* vigilantes pueden distribuirse en los dos polígonos como sigue: un vigilante para el subpolígono izquierdo y $(m - 1)$ vigilantes para el subpolígono derecho. Se supondrá que se sabe cómo resolver el problema de ruta de 1 vigilante. Entonces el problema de rutas de $(m - 1)$ vigilantes puede resolverse de manera recurrente. Es decir, el subpolígono derecho de nuevo se separa en dos partes y se distribuye un vigilante para la izquierda y $(m - 2)$ vigilantes para la derecha. El problema de rutas de *m* vigilantes puede resolverse al intentar exhaustivamente todos los puntos de corte y escogiendo el que tenga la menor suma de las longitudes de las rutas. A continuación se demostrará por qué este método es de programación dinámica.

Considere la figura 7-47 y suponga que se resolverá un problema de rutas de 3 vigilantes. El método de programación dinámica resuelve este problema de rutas de 3 vigilantes como sigue. Sea $SP(v_i, v_j)$ el subpolígono acotado entre bisectrices interiores de v_i y v_j.

FIGURA 7-47 Polígono de 1 espiral con seis vértices en la cadena reflex.



1. Encontrar las siguientes soluciones de ruta de 1 vigilante.

   Solución 1: La solución de ruta de 1 vigilante para $SP(v_1, v_2)$.
   Solución 2: La solución de ruta de 1 vigilante para $SP(v_1, v_3)$.
   Solución 3: La solución de ruta de 1 vigilante para $SP(v_1, v_4)$.
   Solución 4: La solución de ruta de 1 vigilante para $SP(v_2, v_3)$.
   Solución 5: La solución de ruta de 1 vigilante para $SP(v_2, v_4)$.
   Solución 6: La solución de ruta de 1 vigilante para $SP(v_2, v_5)$.
   Solución 7: La solución de ruta de 1 vigilante para $SP(v_3, v_4)$.
   Solución 8: La solución de ruta de 1 vigilante para $SP(v_3, v_5)$.
   Solución 9: La solución de ruta de 1 vigilante para $SP(v_3, v_6)$.
   Solución 10: La solución de ruta de 1 vigilante para $SP(v_4, v_5)$.
   Solución 11: La solución de ruta de 1 vigilante para $SP(v_4, v_6)$.
   Solución 12: La solución de ruta de 1 vigilante para $SP(v_5, v_6)$.

2. Encontrar las siguientes soluciones de ruta de 2 vigilantes.

   Solución 13: La solución de rutas de 2 vigilantes para $SP(v_2, v_6)$.
      Primero se obtienen las siguientes soluciones:
      Solución 13-1: Combinar la solución 4 y la solución 9.
      Solución 13-2: Combinar la solución 5 y la solución 11.
      Solución 13-3: Combinar la solución 6 y la solución 12.
      Como solución 13, se selecciona la de menor longitud entre la solución 13-1, la solución 13-2 y la solución 13-3.
   Solución 14: La solución de rutas de 2 vigilantes para $SP(v_3, v_6)$.
      Primero se obtienen las siguientes soluciones:
      Solución 14-1: Combinar la solución 7 y la solución 11.
      Solución 14-2: Combinar la solución 8 y la solución 12.
      Como solución 14, se selecciona la de menor longitud entre la solución 14-1 y la solución 14-2.
   Solución 15: La solución de rutas de 2 vigilantes para $SP(v_4, v_6)$.
   Ésta puede obtenerse al combinar la solución 10 y la solución 12.

3. Encontrar la solución de ruta de 3 vigilantes del problema original determinando las siguientes soluciones:

   Solución 16-1: Combinar la solución 1 y la solución 13.
   Solución 16-2: Combinar la solución 2 y la solución 14.
   Solución 16-3: Combinar la solución 3 y la solución 15.

   Se selecciona la de menor longitud entre la solución 16-1, la solución 16-2 o la solución 16-3.

El análisis muestra que el problema de rutas de m vigilantes puede resolverse con el método de programación dinámica. Observe que el espíritu del método de programación dinámica es que empieza resolviendo problemas básicos y gradualmente resuelve problemas cada vez más complicados al combinar los subproblemas ya resueltos. Primero se resuelven todos los problemas relevantes de rutas de 1 vigilante, luego los problemas de rutas de 2 vigilantes y así sucesivamente.

Sea $OWR_k(v_i, v_j)$ la longitud de las rutas óptimas de k vigilantes para el subpolígono $SP(v_i, v_j)$. La ruta óptima de m vigilantes $OWR_m(v_s, v_e)$ puede obtenerse con las siguientes fórmulas:

$$OWR_m(v_s, v_e) = \min_{s+1 \le i \le e-1} \{OWR_1(v_s, v_i) + OWR_{m-1}(v_i, v_e)\},$$

$$OWR_k(v_i, v_e) = \min_{i+1 \le j \le e-1} \{OWR_1(v_i, v_j) + OWR_{k-1}(v_j, v_e)\},$$

para $2 \le k \le m-1, s+1 \le i \le e-1$.

Así como para el problema de ruta de 1 vigilante, aquí tampoco se abordarán los detalles porque el objetivo principal es sólo demostrar que este problema puede resolverse con el método de programación dinámica. En la figura 7-48 se muestra una solución típica de un problema de ruta de 1 vigilante, y en la figura 7-49 se muestran casos especiales.

FIGURA 7-48 Problema típico de ruta de 1 vigilante p, v_a, $C[v_a, v_b]$, v_b, r_1 en un polígono de 1 espiral.

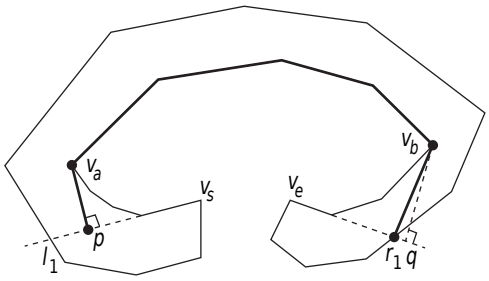
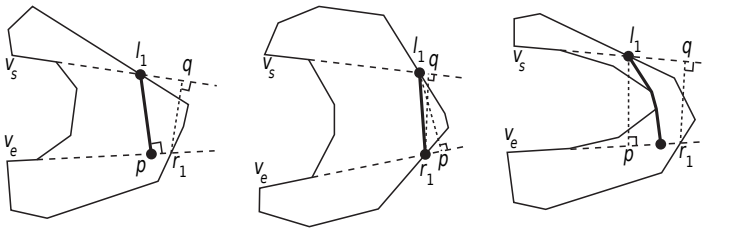


FIGURA 7-49 Casos especiales del problema de ruta de 1 vigilante en un polígono de 1 espiral.



a) Uno de los dos, $\overline{l_1 p}$ o $\overline{r_1 q}$ está dentro del polígono.

b) Ambos p o q están fuera del polígono.

c) Uno de los dos, $\overline{l_1 p}$ o $\overline{r_1 q}$ corta la cadena reflex.

7-10 LOS RESULTADOS EXPERIMENTALES

Para demostrar la potencia del método de programación dinámica, éste se implementó para resolver en una computadora el problema de la subsecuencia común más larga. En la misma computadora también se programó el método directo. Los resultados experimentales se muestran en la tabla 7-7. Los resultados indican claramente la superioridad del método de programación dinámica.

TABLA 7-7 Resultados experimentales.

Longitud de la cadena	Tiempo de CPU en milisegundos	
	Programación dinámica	Método exhaustivo
4	< 1	20
6	< 2	172
8	< 2	2 204
10	< 10	32 952
12	< 14	493 456

7-11 NOTAS Y REFERENCIAS

Puede afirmarse que el término programación dinámica fue acuñado por Bellman (1962). Muchos autores han escrito libros sobre este tema: Nemhauser (1966); Dreyfus y Law (1977), y Denardo (1982).

En 1962, Bellman realizó una revisión de la aplicación de la programación dinámica para resolver problemas combinatorios (1962). El planteamiento del método de programación dinámica para el problema de asignación de recursos y de programación apareció en Lawler y Moore (1969); Sahni (1976), y Horowitz y Sahni (1978). El planteamiento de programación dinámica para el problema del agente viajero se debe a Held y Karp (1962) y Bellman (1962). La aplicación de la programación dinámica al problema de la subsecuencia común más larga fue propuesta por Hirschberg (1975). El método de programación dinámica para resolver el problema 0/1 de la mochila puede encontrarse en Nemhauser y Ullman (1969) y Horowitz y Sahni (1974). La construcción de árboles de búsqueda binarios óptimos usando el método de programación dinámica puede consultarse en Gilbert y Moore (1959), Knuth (1971) y Knuth (1973). La aplicación del método de programación dinámica al problema ponderado del conjunto de dominación perfecta en árboles se encuentra en Yen y Lee (1990), y la aplicación para resolver el problema de búsqueda en un solo paso puede consultarse en Hsiao, Tang y Chang (1993). El algoritmo de alineación de 2 secuencias puede encontrarse en Neddleman y Wunsch (1970), mientras que el algoritmo para la predicción de la estructura secundaria en ARN en Waterman y Smith (1978).

Otras aplicaciones importantes de la programación dinámica, que no se mencionan en este libro, incluyen multiplicación matricial en cadena: Goodbole (1973); Hu y Shing (1982), y Hu y Shing (1984); todas las rutas más cortas: Floyd (1962); análisis sintáctico de lenguajes libre de contexto: Younger (1967), y/o gráficas en serie-paralelo: Simon y Lee (1971) y decodificación de Viterbi: Viterbi (1967) y Omura (1969).

La programación dinámica puede usarse junto con la técnica branch-and-bound. Consulte la obra de Morin y Marsten (1976). También puede aplicarse para resolver un tipo especial de problema de partición. Consulte la sección 4.2 del libro de Garey y Johnson (1979). Este concepto fue usado por Hsu y Nemhauser (1979).

7-12 BIBLIOGRAFÍA ADICIONAL

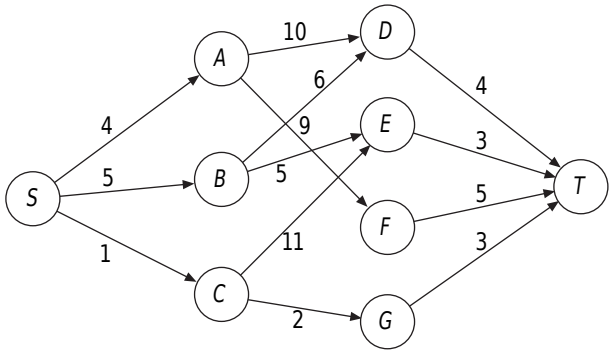
Debido a que la programación dinámica es tan sencilla e ingeniosa, durante mucho tiempo ha constituido un tema de investigación para teóricos y para prácticos en ciencias de computación. Para investigación adicional, recomendamos los artículos siguientes: Akiyoshi y Takeaki (1997); Auletta, Parente y Persiano (1996); Baker (1994);

Bodlaender (1993); Chen, Kuo y Sheu (1988); Chung, Makedon, Sudborough y Turner (1985); Even, Itai y Shamir (1976); Farber y Keil (1985); Fonlupt y Nachef (1993); Gotlieb (1981); Gotlieb y Wood (1981); Hirschberg y Larmore (1987); Horowitz y Sahni (1974); Huo y Chang (1994); Johnson y Burrus (1983); Kantabutra (1994); Kao y Queyranne (1982); Kilpelainen y Mannila (1995); Kryazhimskiy y Savinov (1995); Liang (1994); Meijer y Rappaport (1992); Morin y Marsten (1976); Ozden (1988); Park (1991); Peng, Stephens y Yesha (1993); Perl (1984); Pevzner (1992); Rosenthal (1982); Sekhon (1982); Tidball y Atman (1996); Tsai y Hsu (1993); Tsai y Lee (1997); Yannakakis (1985); Yen y Lee (1990); Yen y Lee (1994), y Yen y Tang (1995).

A continuación se presenta una lista de resultados nuevos e interesantes: Aho, Ganapathi y Tjang (1989); Akutsu (1996); Alpert y Kahng (1995); Amini, Weymouth y Jain (1990); Baker y Giancarlo (2002); Bandelloni, Tucci y Rinaldi (1994); Barbu (1991); Brown y Whitney (1994); Charalambous (1997); Chen, Chern y Jang (1990); Cormen (1999); Culberson y Rudnicki (1989); Delcoigne y Hansen (1975); Eppstein, Galil, Giancarlo e Italiano (1990); Eppstein, Galil, Giancarlo e Italiano (1992(a)); Eppstein, Galil, Giancarlo e Italiano (1992(b)); Erdmann (1993); Farach y Thorup (1997); Fischel-Ghodsian, Mathiowitz y Smith (1990); Geiger, Gupta, Costa y Vlontzos (1995); Gelfand y Roytberg (1993); Galil y Park (1992); Hanson (1991); Haussmann y Suo (1995); Hein (1989); Hell, Shamir y Sharan (2001); Hirosawa, Hoshida, Ishikawa y Toya (1993); Holmes y Durbin (1998); Huang, Liu y Viswanathan (1994); Huang y Waterman (1992); Ibaraki y Nakamura (1994); Karoui y Quenez (1995); Klein (1995); Kostreva y Wiecek (1993); Lewandowski, Condon y Bach (1996); Liao y Shoemaker (1991); Lin, Fan y Lee (1993); Lin, Chen, Jiang y Wen (2002); Littman, Cassandra y Kaelbling (1996); Martin y Talley (1995); Merlet y Zerubia (1996); Miller y Teng (1999); Mohamed y Gader (1996); Moor (1994); Motta y Rampazzo (1996); Myoupo (1992); Ney (1984); Ney (1991); Nuyts, Suetens, Oosterlinck, Roo y Mortelmans (1991); Ohta y Kanade (1985); Ouyang y Shahidehpour (1992); Pearson y Miller (1992); Rivas y Eddy (1999); Sakoe y Chiba (1978); Schmidt (1998); Snyder y Stormo (1993); Sutton (1990); Tataru (1992); Tatman y Shachter (1990); Tatsuya (2000); Vintsyuk (1968); Von Haeselerm, Blum, Simpson, Strum y Waterman (1992); Waterman y Smith (1986); Wu (1996); Xu (1990), y Zuker (1989).

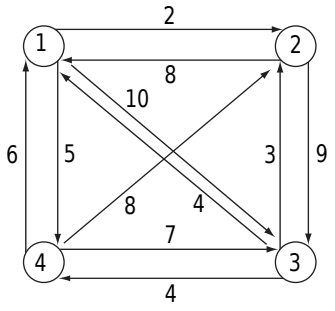
Ejercicios

7.1 Considere la gráfica siguiente. Aplique el método de programación dinámica y encuentre la ruta más corta de *S* a *T*.



7.2 Para la gráfica que se muestra en la figura 7-1, resuelva el mismo problema aplicando el método branch-and-bound. Para este problema, ¿cuál método es mejor (programación dinámica o branch-and-bound)? ¿Por qué?

7.3 Para la gráfica que se muestra a continuación, resuelva el problema del agente viajero aplicando el método branch-and-bound. Compárelo con el método de programación dinámica.



	1	2	3	4
1	∞	2	10	5
2	8	∞	9	∞
3	4	3	∞	4
4	6	8	7	∞

7.4 Para la tabla siguiente, encuentre una asignación de recursos óptima que maximice la ganancia total para estos tres proyectos y cuatro recursos.

recurso / proyecto	1	2	3	4
1	3	7	10	12
2	1	2	6	9
3	2	4	8	9

7.5 Aplique el método de programación dinámica para resolver el siguiente problema de programación lineal.

Maximizar $x_0 = 8x_1 + 7x_2$
sujeta a
$2x_1 + x_2 \leq 8$
$5x_1 + 2x_2 \leq 15$
donde x_1 y x_2 son enteros no negativos.

7.6 Encuentre una subsecuencia común más larga de

$S_1 = a\ a\ b\ c\ d\ a\ e\ f$

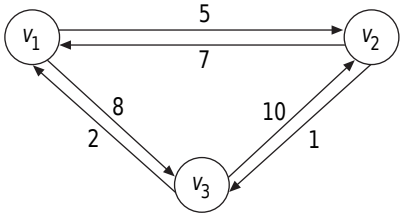
y $S_2 = b\ e\ a\ d\ f$.

7.7 En general, el problema de partición es NP-completo. Sin embargo, bajo algunas restricciones, un tipo especial del problema de partición es un problema polinomial porque puede resolverse con programación dinámica. Consulte la sección 4-2 del libro de Garey y Johnson (1979).

7.8 Encuentre un árbol binario óptimo para $a_1, a_2, \ldots, a_6$, si los identificadores, en orden, tienen probabilidades 0.2, 0.1, 0.15, 0.2, 0.3 y 0.05, respectivamente y todos los demás identificadores tienen probabilidad cero.

7.9 Considere la gráfica siguiente. Resuelva el problema de las rutas más cortas de todos los pares de la gráfica. Este problema consiste en encontrar la ruta más corta que hay entre cada par de vértices. Consulte la

sección 5-3 del libro de Horowitz y Sahni (1978) o la sección 5-4 de la obra de Brassard y Bratley (1988).



7.10 Sean f una función real de x y $y = (y_1, y_2, \ldots, y_k)$. Se dice que f puede descomponerse en f_1 y f_2 si f es separable ($f(x, y) = f_1(x), f_2(y)$) y si, además, la función es monótona no decreciente con respecto a su segundo argumento. Demuestre que si f puede descomponerse con $f(x, y) = (f_1(x), f_2(y))$, entonces

$$\underset{(x,y)}{Ópt}\{f(x,y)\} = Ópt\{f_1(x, Ópt\{f_2(y)\})\} \quad (Ópt = \text{mín o máx})$$

(Consulte la sección 9-2 de [Minoux 1986].)

7.11 El algoritmo de Floyd, que puede encontrarse fácilmente en muchos libros de texto, es para encontrar las rutas más cortas de todos los pares en una gráfica ponderada. Proporcione un ejemplo para explicar el algoritmo.

7.12 Escriba un algoritmo de programación dinámica que resuelva el problema de la subsecuencia creciente más larga.

7.13 Dadas dos secuencias S_1 y S_2 en un conjunto alfabeto Σ y una función de puntaje f: $\Sigma \times \Sigma \rightarrow \Re$, el problema de asignación local consiste en encontrar una subsecuencia S_1' de S_1 y una subsecuencia S_2' de S_2 tales que el puntaje obtenido al alinear S_1' y S_2' sea el más alto, de entre todas las posibles subsecuencias de S_1 y S_2. Aplique la estrategia de programación dinámica para diseñar un algoritmo de tiempo $O(nm)$ o mejor para resolver este problema, donde n y m denotan las longitudes de S_1 y S_2, respectivamente.