capítulo

3

El método codicioso

El método codicioso (greedy) es una estrategia para resolver problemas de optimización. Se supondrá que es posible resolver un problema mediante una secuencia de decisiones. El método codicioso utiliza el siguiente enfoque: en cada etapa, la decisión es óptima. Para algunos problemas, como se verá, estas soluciones localmente óptimas se agregarán para integrar una solución globalmente óptima. Aquí se pone énfasis en que este método codicioso sólo es capaz de resolver algunos problemas de optimización. En casos en que las decisiones localmente óptimas no den por resultado una solución globalmente óptima, el método codicioso podría seguir siendo recomendable porque, como se verá más tarde, por lo menos produce una solución que suele ser aceptable.

A continuación, la característica del método codicioso se describirá con un ejemplo. Considere el caso en que dado un conjunto de n números se solicita escoger k números, de entre todas las formas que hay para elegir los k números del conjunto dado, de modo que la suma de estos k números sea máxima.

Para resolver este problema podrían probarse todas las formas posibles que hay para escoger k números de un conjunto de n números. Por supuesto, ésta es una forma absurda de resolver el problema, ya que simplemente podrían escogerse los k números más grandes, mismos que constituirían la solución. O bien, podría afirmarse que el algoritmo para resolver este problema es como sigue:

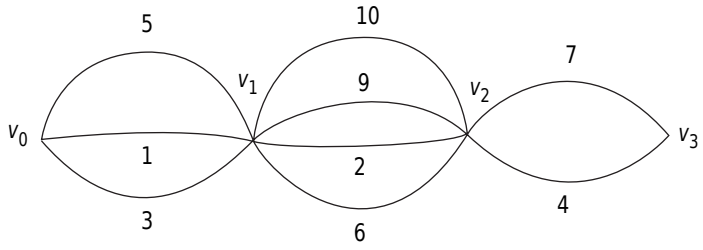
```
For i := 1 to k do
    Escoger el número más grande y eliminarlo de la entrada.
Endfor
```

El algoritmo anterior es un método codicioso típico. En cada etapa se selecciona el número más grande.

Se considerará otro caso en el que también es posible aplicar el método codicioso. En la figura 3-1 se solicita encontrar la ruta más corta de v_0 a v_3. Para esta gráfica particular, el problema puede resolverse encontrando una ruta más corta entre v_i y v_{i+1},

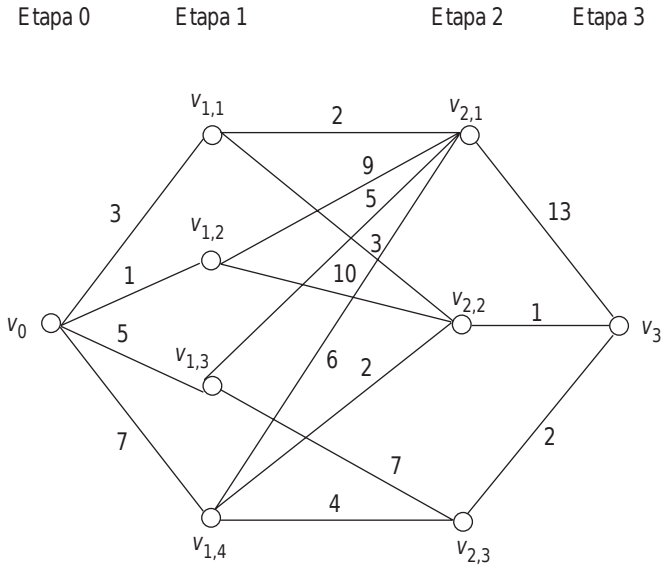
$i = 0$ hasta 2. Es decir, primero se determina la ruta más corta entre v_0 y v_1, luego entre v_1 y v_2, y así sucesivamente. Debe resultar evidente que este procedimiento termina por proporcionar una solución óptima.

FIGURA 3-1 Un caso en que funciona el método codicioso.



Sin embargo, fácilmente puede proporcionarse un ejemplo en que el método codicioso no funciona. Considere la figura 3-2.

FIGURA 3-2 Un caso en que no funciona el método codicioso.



En la figura 3-2 nuevamente se solicita obtener una ruta más corta de v_0 a v_3. Si se utiliza el método codicioso, en la etapa 1 se encontrará una ruta más corta de v_0 a algún

nodo en la etapa 1. Así, se selecciona $v_{1,2}$. En el siguiente movimiento se encontrará la ruta más corta entre $v_{1,2}$ y algún nodo en la etapa 2. Se selecciona $v_{2,1}$. La solución final es

$$v_0 \rightarrow v_{1,2} \rightarrow v_{2,1} \rightarrow v_3.$$

La longitud total de esta ruta es $1 + 9 + 13 = 23$.

Esta solución, aunque se obtuvo de manera rápida, no es una solución óptima. De hecho, la solución óptima es

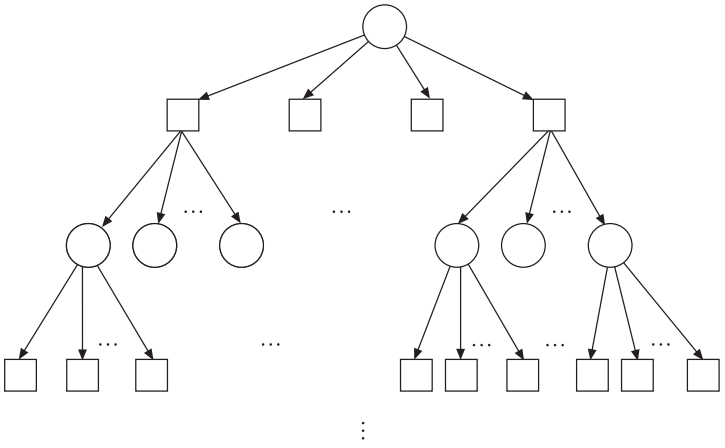
$$v_0 \rightarrow v_{1,1} \rightarrow v_{2,2} \rightarrow v_3$$

cuyo costo es $3 + 3 + 1 = 7$.

En capítulos posteriores se presentarán métodos para resolver el problema.

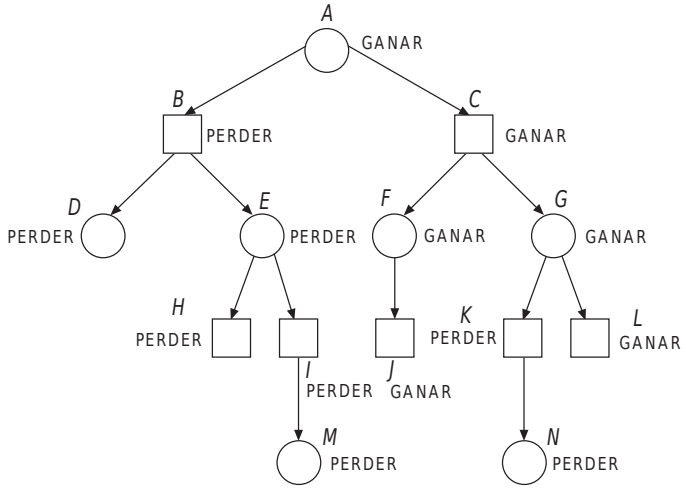
¿Qué falla cuando el método codicioso se aplica para resolver el problema que se muestra en la figura 3-2? Para contestar esta pregunta, por un momento la atención se dirigirá al juego de ajedrez. Un buen jugador de ajedrez no simplemente observa el tablero y hace un movimiento que es el mejor para él en ese momento; más bien mira "hacia delante". Es decir, tiene que imaginar la mayor parte de los movimientos posibles que puede efectuar y luego supone la forma en que podría reaccionar su oponente. Debe entender que su oponente también "mira hacia delante". Así, toda la partida podría verse como el árbol que se muestra en la figura 3-3.

FIGURA 3-3 Árbol de juego.



En la figura 3-3 un círculo representa un jugador y un cuadro representa a su oponente. El primer movimiento inteligente y correcto sólo puede hacerse cuando es posible *podar* (*prune*: eliminar algunas ramas) este árbol de juego. Considere la figura 3-4, que representa un árbol de final de juego ficticio.

FIGURA 3-4 Árbol de final de juego.



Con base en este árbol de final de juego es posible decidir qué movimiento realizar razonando que:

1. Debido a que el oponente siempre quiere que perdamos, *I* y *K* se identifican como PERDER; si se alcanza cualquiera de estos estados, definitivamente perderemos la partida.
2. Debido a que uno siempre quiere ganar, *F* y *G* se identifican como GANAR. Además, *E* debe identificarse como PERDER.
3. De nuevo, como el oponente siempre quiere que perdamos, *B* y *C* se identifican como PERDER y GANAR, respectivamente.
4. Debido a que uno quiere ganar, *A* se identifica como GANAR.

A partir del razonamiento anterior, *se entiende que cuando se toman decisiones, a menudo se mira hacia delante. Sin embargo, el método codicioso jamás hace ningún trabajo en ver hacia delante*. Considere la figura 3-2. Para encontrar una ruta más corta de v_0 a v_3, también es necesario ver hacia delante. Para seleccionar un nodo de los nodos de la etapa 1, es necesario conocer la distancia más corta de cada nodo de la etapa 1 a v_3. Sea $dmin(i, j)$ la distancia mínima entre los nodos i y j. Entonces

$$dmin(v_0, v_3) = \begin{cases} 3 + dmin(v_{1,1}, v_3) \\ 1 + dmin(v_{1,2}, v_3) \\ 5 + dmin(v_{1,3}, v_3) \\ 7 + dmin(v_{1,4}, v_3). \end{cases}$$

Ahora el lector puede ver que como el método codicioso no mira hacia delante, puede fracasar en cuanto a proporcionar una solución óptima. Sin embargo, en el resto de este capítulo se presentarán muchos ejemplos interesantes en los que el método codicioso funciona.

3-1 MÉTODO DE KRUSKAL PARA ENCONTRAR UN ÁRBOL DE EXPANSIÓN MÍNIMA

Uno de los problemas más famosos que pueden resolverse con el método codicioso es el problema del árbol de expansión mínima. En esta sección se presentará el método de Kruskal para encontrar un árbol de expansión mínima. Los árboles de expansión mínima pueden definirse sobre puntos del espacio euclidiano o sobre una gráfica. Para el método de Kruskal, los árboles de expansión mínima se definen sobre gráficas.

Definición

Sea $G = (V, E)$ una gráfica no dirigida conexa ponderada, donde V representa el conjunto de vértices y E representa el conjunto de aristas. Un árbol de expansión de G es un árbol no dirigido $S = (V, T)$ donde T es un subconjunto de E. El peso total de un árbol de expansión es la suma de todos los pesos de T. Un árbol de expansión mínima de G es un árbol de expansión de G cuyo peso total es mínimo.

Ejemplo 3-1 Árbol de expansión mínima

En la figura 3-5 se muestra una gráfica que consta de cinco vértices y ocho aristas. En la figura 3-6 se muestran algunos de los árboles de expansión de esta gráfica. En la figura 3-7 se muestra un árbol de expansión mínima de la gráfica, cuyo peso total es $50 + 80 + 60 + 70 = 260$.

FIGURA 3-5 Gráfica no dirigida conexa ponderada.

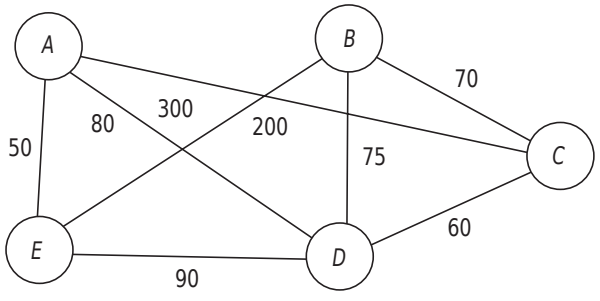
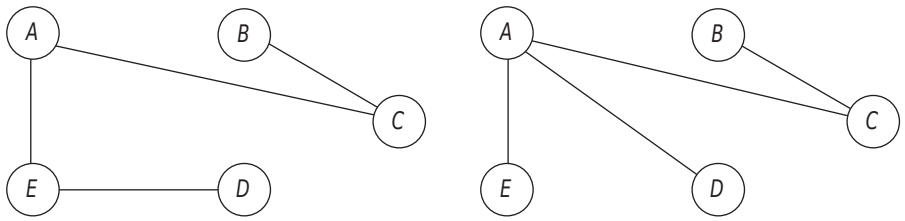
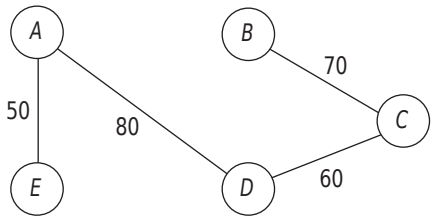


FIGURA 3-6 Algunos árboles de expansión.



a) *b*)

FIGURA 3-7 Árbol de expansión mínima.



El método de Kruskal para construir un árbol de expansión mínima puede describirse brevemente como sigue:

1. Del conjunto de aristas, seleccionar la de menor peso. Ésta constituye la subgráfica inicial construida parcialmente que después será desarrollada en un árbol de expansión mínima.
2. A esta gráfica construida parcialmente súmese la siguiente arista ponderada más pequeña si esto no provoca la formación de un ciclo. En caso contrario, eliminar la arista seleccionada.
3. Terminar si el árbol de expansión contiene $n - 1$ aristas. En caso contrario, ir a 2.

Algoritmo 3-1 □ Algoritmo de Kruskal del árbol de expansión mínima

Input: Una gráfica no dirigida, conexa y ponderada $G = (V, E)$.
Output: Un árbol de expansión mínima para G.

$T := \phi$
Mientras T contiene menos de $n - 1$ aristas, hacer
Begin
    Elegir una arista (v, w) de E del peso más pequeño

```
        Delete (v, w) de E
        Si [al agregar (v, w) a T no se produce ningún ciclo en T] entonces
            Add (v, w) a T
        En otro caso
            Discard (v, w)
End
```

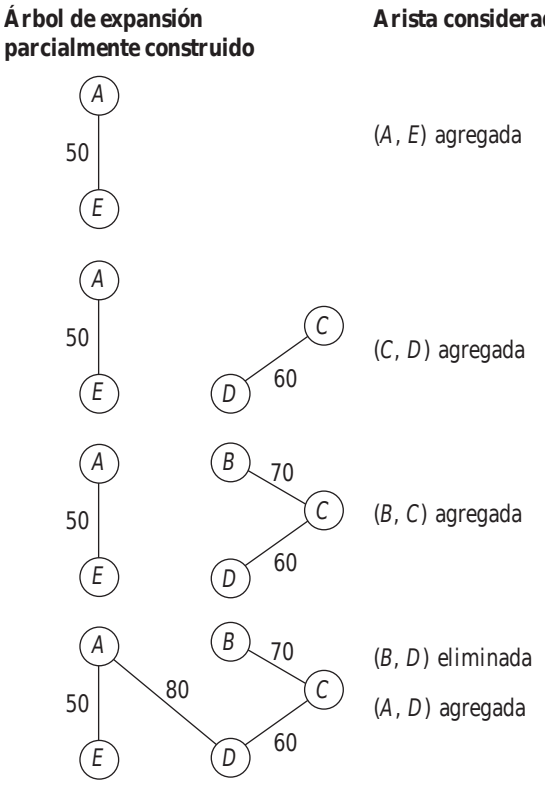
Ejemplo 3-2 Algoritmo de Kruskal

Considere la gráfica de la figura 3-5. Las aristas están ordenadas en la siguiente secuencia:

(*A*, *E*) (*C*, *D*) (*B*, *C*) (*B*, *D*) (*A*, *D*) (*E*, *D*) (*E*, *B*) (*A*, *C*).

Luego se construye el árbol de expansión mínima que se muestra en la figura 3-8.

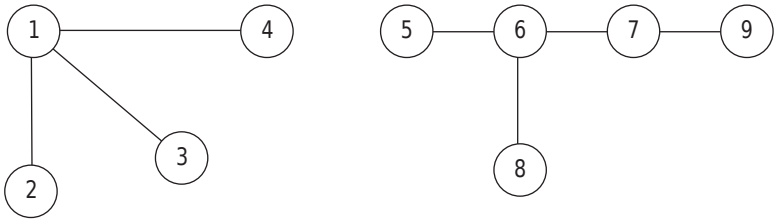
FIGURA 3-8 Determinación de un árbol de expansión mínima aplicando el algoritmo de Kruskal.



El lector debe observar que nunca fue necesario clasificar las aristas. De hecho, el heap considerado en el apartado 2-5 puede usarse para seleccionar la siguiente arista con el peso más pequeño.

Hay otro problema que es necesario resolver: ¿cómo puede determinarse eficientemente si la arista agregada formará un ciclo? La respuesta de esta pregunta es muy fácil. Durante el proceso del algoritmo de Kruskal, la subgráfica parcialmente construida es un bosque generador que consta de muchos árboles. En consecuencia, cada conjunto de vértices de un árbol puede preservarse en un conjunto individual. Considere la figura 3-9. En esta figura se observan dos árboles que pueden representarse como $S_1 = \{1, 2, 3, 4\}$ y $S_2 = \{5, 6, 7, 8, 9\}$. Suponga que la siguiente arista que ha de agregarse es (3, 4). Debido a que ambos vértices 3 y 4 están en S_1, esto originará que se forme un ciclo. Por consiguiente, no es posible agregar (3, 4). De manera semejante, no es posible agregar (8, 9) porque ambos vértices 8 y 9 están en S_2. No obstante, es posible agregar (4, 8).

FIGURA 3-9 Un bosque generador.



Con base en el razonamiento anterior puede verse que el algoritmo de Kruskal es dominado por las siguientes acciones:

1. Ordenamiento, que requiere $O(m \log m)$ pasos, donde m es el número de aristas en la gráfica.
2. Unión de dos conjuntos. Esto es necesario cuando se fusionan dos árboles. Cuando se inserta una arista que vincula dos subárboles, esencialmente se está encontrando la unión de dos conjuntos. Por ejemplo, suponga que la arista (4, 8) se agrega al árbol de expansión en la figura 3-9; al hacer lo anterior, los dos conjuntos {1, 2, 3, 4} y {5, 6, 7, 8, 9} se fusionan en {1, 2, 3, 4, 5, 6, 7, 8, 9}. Así, es necesario realizar una operación; a saber, la unión de dos conjuntos.
3. Determinación de un elemento en un conjunto. Observe que cuando se comprueba si es posible agregar o no una arista, es necesario verificar si dos vértices están o no en un conjunto de vértices. Así, es necesario realizar una operación, denominada operación encontrar, con la cual se determina si un elemento está o no en un conjunto.

En el capítulo 10 se mostrará que para efectuar y para encontrar las operaciones de unión se requieren $O(m)$ pasos. Así, el tiempo total del algoritmo de Kruskal es dominado por el ordenamiento, que es $O(m \log m)$. En el peor caso, $m = n^2$. En consecuencia, la complejidad temporal del algoritmo de Kruskal es $O(n^2 \log n)$.

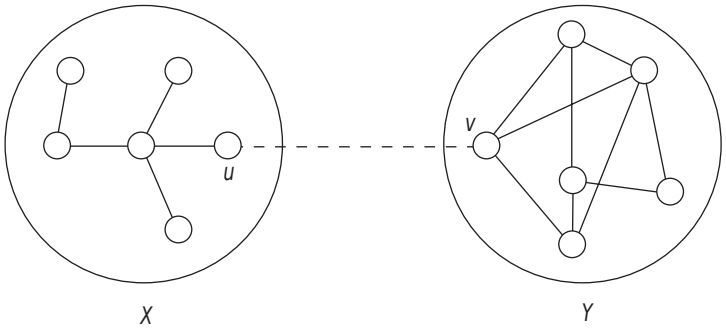
A continuación se demostrará que el algoritmo de Kruskal es correcto. Es decir, que produce un árbol de expansión mínima. Se supondrá que los pesos asociados con todas las aristas son distintos y que $|e_1| < |e_2| < \cdots < |e_m|$, donde $m = |E|$. Sean T el árbol de expansión producido por el algoritmo de Kruskal y T' un árbol de expansión mínima. Se demostrará que $T = T'$. Suponga lo contrario. Así, sea e_i la arista con peso mínimo en T que no aparece en T'. Resulta evidente que $i \neq 1$; e_i se agrega a T'. Esto necesariamente forma un ciclo en T'. Sea e_j una arista en este ciclo que no es una arista en T. Esta e_j debe existir. En caso contrario, todas las aristas en este ciclo pertenecen a T, lo cual significa que en T hay un ciclo imposible. Hay dos casos posibles. Caso 1: el peso de e_j es menor que el peso de e_i; por ejemplo, $j < i$. Sea T_k el árbol producido por el algoritmo de Kruskal después de comprobar si es posible o no agregar e_k. Resulta evidente que T_k, para $k < i$, es un subárbol tanto de T como de T' porque e_i es la arista de menor peso en T que no aparece en T', como se supuso. Debido a que $j < i$ y a que e_j no es una arista en T, entonces e_j no debe ser seleccionada por el algoritmo de Kruskal. La razón de esto es que al agregar e_j a T_{j-1} se forma un ciclo. No obstante, ya que T_{j-1} también es un subárbol de T' y como e_j es una arista de T', al agregar e_j a T_{j-1} no puede formarse un ciclo. Así, este caso es imposible. Caso 2: el peso de e_j es mayor que el de e_i; por ejemplo, $j > i$. En este caso se elimina e_j. De este modo se crea un nuevo árbol de expansión cuyo peso total es menor que el de T'. Así, T' debe ser el mismo que T.

Resulta evidente que el algoritmo de Kruskal utiliza el método codicioso. En cada paso, la siguiente arista que ha de agregarse es localmente óptima. Resulta interesante observar que el resultado final es globalmente óptimo.

3-2 MÉTODO DE PRIM PARA ENCONTRAR UN ÁRBOL DE EXPANSIÓN MÍNIMA

En el apartado 3-1 se presentó el algoritmo de Kruskal para encontrar un árbol de expansión mínima. En esta sección se presenta un algoritmo descubierto de manera independiente por Dijkstra y Prim. El algoritmo de Prim construye paso a paso un árbol de expansión mínima. En cualquier momento, sea X el conjunto de vértices contenidos en el árbol de expansión mínima parcialmente construido. Sea $Y = V - X$. La siguiente arista (u, v) que ha de agregarse es una arista entre X y Y ($u \in X$ y $v \in Y$) con el menor peso. La situación se describe en la figura 3-10. La siguiente arista agregada es (u, v) y después que se agrega esta arista, v se agrega a X y se elimina de Y. Un punto importante en el método de Prim es que puede empezarse en cualquier vértice, lo cual es muy conveniente.

FIGURA 3-10 Ilustración del método de Prim.



A continuación se presentará brevemente el algoritmo de Prim. Después se describirá con más detalle.

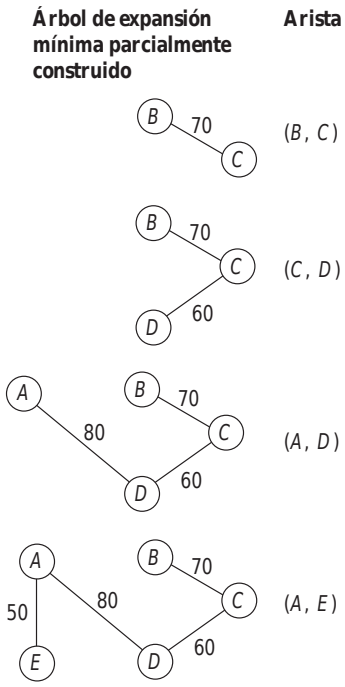
Algoritmo 3-2 □ Algoritmo básico de Prim para encontrar un árbol de expansión mínima

Input: Una gráfica no dirigida, conexa y ponderada $G = (V, E)$.
Output: Un árbol de expansión mínima para G.
Paso 1. Hacer x cualquier vértice en V. Sean $X = \{x\}$ y $Y = V - \{x\}$.
Paso 2. Seleccionar una arista (u, v) de E tal que $u \in X$, $v \in Y$ y (u, v) tiene el menor peso de todas las aristas entre X y Y.
Paso 3. Conectar u con v. Hacer $X = X \cup \{v\}$ y $Y = Y - \{v\}$.
Paso 4. Si Y es vacío, se termina y el árbol resultante es un árbol de expansión mínima. En caso contrario, ir al paso 2.

Ejemplo 3-3 Algoritmo básico de Prim

Considere nuevamente la figura 3-5. A continuación se mostrará cómo el algoritmo de Prim generaría un árbol de expansión mínima. Se supondrá que el vértice B se selecciona como punto inicial. En la figura 3-11 se ilustra este proceso. En cada paso del proceso, la arista que ha de agregarse incrementa al mínimo el costo total. Por ejemplo, cuando el árbol parcialmente construido contiene los vértices B, C y D, el conjunto de vértices restante es $\{A, E\}$. La arista de peso mínimo que une $\{A, E\}$ y $\{B, C, D\}$ es (A, D). Así, se agrega (A, D). Debido a que el vértice A no está contenido en el árbol de expansión mínima parcialmente construido, esta nueva arista no forma ningún ciclo.

FIGURA 3-11 Determinación de un árbol de expansión mínima con el algoritmo de Prim y vértice inicial *B*.



Quizás el lector esté interesado en saber si es posible empezar con algún otro vértice. En efecto, suponga que inicialmente se empieza con el vértice *C*. En la figura 3-12 se ilustra el árbol de expansión mínima que se construye.

FIGURA 3-12 Determinación de un árbol de expansión mínima con el algoritmo básico de Prim con vértice inicial *C*.

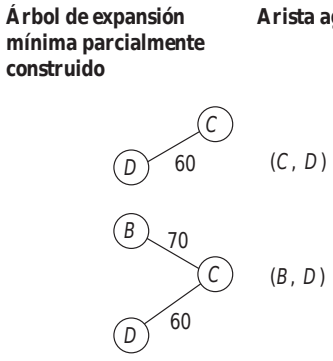
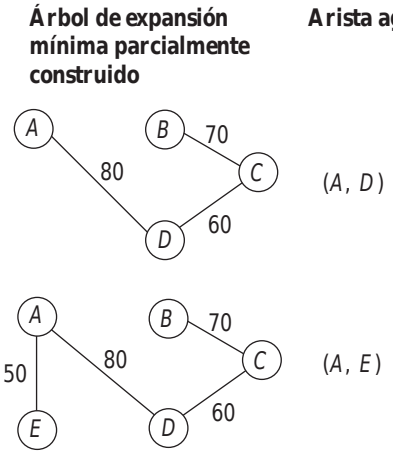
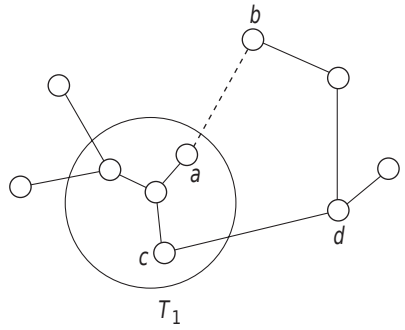


FIGURA 3-12 (*continuación*)



A continuación se demostrará que el algoritmo de Prim es correcto. Es decir, que el árbol de expansión que produce es en efecto un árbol de expansión mínima. Sea $G = (V, E)$ una gráfica conexa ponderada. Sin pérdida de generalidad puede suponerse que los pesos de todas las aristas son distintos. Sea T un árbol de expansión mínima para G. Sea T_1 un subárbol de T, como se muestra en la figura 3-13. Sean V_1 el conjunto de vértices en T_1 y $V_2 = V - V_1$. Sea (a, b) una arista de peso mínimo en E tal que $a \in V_1$ y $b \in V_2$. Se demostrará que (a, b) debe estar en T. Suponga lo contrario. Entonces debe haber una ruta en T de a a b porque un árbol es conexo. Sea (c, d) una arista en esa ruta tal que $c \in V_1$ y $d \in V_2$. El peso de (c, d) debe ser mayor que el peso de (a, b). En consecuencia, es posible crear otro árbol de expansión más pequeño al eliminar (c, d) y agregar (a, b). Esto prueba que T no debe ser un árbol de expansión mínima. Por consiguiente, (a, b) debe estar en T y el algoritmo de Prim es correcto.

FIGURA 3-13 Árbol de expansión mínima para explicar que el algoritmo de Prim es correcto.



El algoritmo que acaba de presentarse es sólo un breve bosquejo del algoritmo de Prim. Se presentó de modo que fuese fácil de comprender. A continuación se empezará desde el principio: $X = \{x\}$ y $Y = V - \{x\}$. Para encontrar la arista de peso mínimo entre X y Y es necesario analizar todas las aristas que inciden en x. En el peor caso, lo anterior se efectúa en n – 1 pasos, donde n es el número de vértices en V. Suponga que y se agrega a X. Es decir, suponga que $X = \{x, y\}$ y $Y = V - \{x, y\}$. Para encontrar la arista de menor peso entre X y Y parece que hay un problema: ¿es necesario analizar de nuevo las aristas que inciden en x? (Por supuesto, no es necesario volver a analizar la arista entre x y y porque ambas están en X.) Prim sugirió una forma inteligente para evitar este problema, preservando dos vectores.

Considere que hay n vértices, identificados como 1, 2, ... , n. Sean dos vectores C_1 y C_2. Sean X el conjunto de vértices de árbol parcialmente construido en el algoritmo de Prim y $Y = V$ – X. Sea i un vértice en Y. De todas las aristas que inciden sobre los vértices de X y el vértice i en Y, sea la arista (i, j), $j \in X$, la arista de menor peso. Los vectores C_1 y C_2 se usan para almacenar esta información. Sea $w(i, j)$ el peso de la arista (i, j). Así, en cualquier paso del algoritmo de Prim,

$$C_1(i) = j$$
y $$C_2(i) = w(i, j).$$

A continuación se demostrará que estos dos vectores pueden usarse para evitar el análisis repetido de las aristas. Sin pérdida de generalidad, inicialmente se supone $X = \{1\}$ y $Y = \{2, 3, \ldots, n\}$. Resulta evidente que para cada vértice i en Y, $C_1(i) = 1$ y $C_2(i) = w(i, 1)$ si la arista $(i, 1)$ existe. El menor $C_2(i)$ determina el siguiente vértice que ha de agregarse a X.

De nuevo puede suponerse que el vértice 2 se selecciona como el punto que ha de agregarse a X. Así, $X = \{1, 2\}$ y $Y = \{3, 4, \ldots, n\}$. El algoritmo de Prim requiere la determinación de la arista de menor peso entre X y Y. Pero, con ayuda de $C_1(i)$ y $C_2(i)$, ya no es necesario analizar las aristas incidentes sobre el vértice i. Suponga que i es un vértice en Y. Si $w(i, 2) < w(i, 1)$, $C_1(i)$ se cambia de 1 a 2 y $C_2(i)$ se cambia de $w(i, 1)$ a $w(i, 2)$. Si $w(i, 2) \geq w(i, 1)$, no se hace nada. Una vez que la actualización se ha completado para todos los vértices en Y, es posible escoger un vértice que ha de agregarse a X mediante el análisis de $C_2(i)$. El menor $C_2(i)$ determina el siguiente vértice que ha de agregarse. Como puede ver el lector, ahora se ha evitado exitosamente el análisis repetido de todas las aristas. Cada arista se examina una sola vez.

A continuación se proporciona el algoritmo de Prim con más detalle.

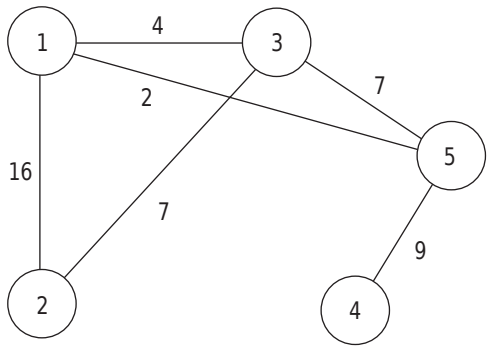
Algoritmo 3-3 □ Algoritmo de Prim para construir un árbol de expansión mínima

Input: Una gráfica no dirigida, conexa y ponderada $G = (V, E)$.
Output: Un árbol de expansión mínima de G.
Paso 1. Hacer $X = \{x\}$ y $Y = V - \{x\}$, donde x es cualquier vértice en V.
Paso 2. Asignar $C_1(y_j) = x$ y $C_2(y_j) = \infty$ para todo vértice y_j en V.
Paso 3. Para cada vértice y_j en V, se analiza si y_j está en Y y si la arista (x, y_j) existe. Si y_j está en Y, entonces la arista (x, y_j) existe y $w(x, y_j) = b < C_2(y_j)$, hacer $C_1(y_j) = x$ y $C_2(y_j) = b$; en caso contrario, no se hace nada.
Paso 4. Hacer y un vértice en Y tal que $C_2(y)$ es mínimo. Hacer $z = C_1(y)$ (z debe estar en X). Conectar y en la arista (y, z) con z en el árbol parcialmente construido T. Hacer $X = X + \{y\}$ y $Y = Y - \{y\}$ y $C_2(y) = \infty$.
Paso 5. Si Y está vacío, se termina y el árbol resultante T es un árbol de expansión mínima; en caso contrario, se hace $x = y$ y se va al paso 3.

Ejemplo 3-4 Algoritmo de Prim

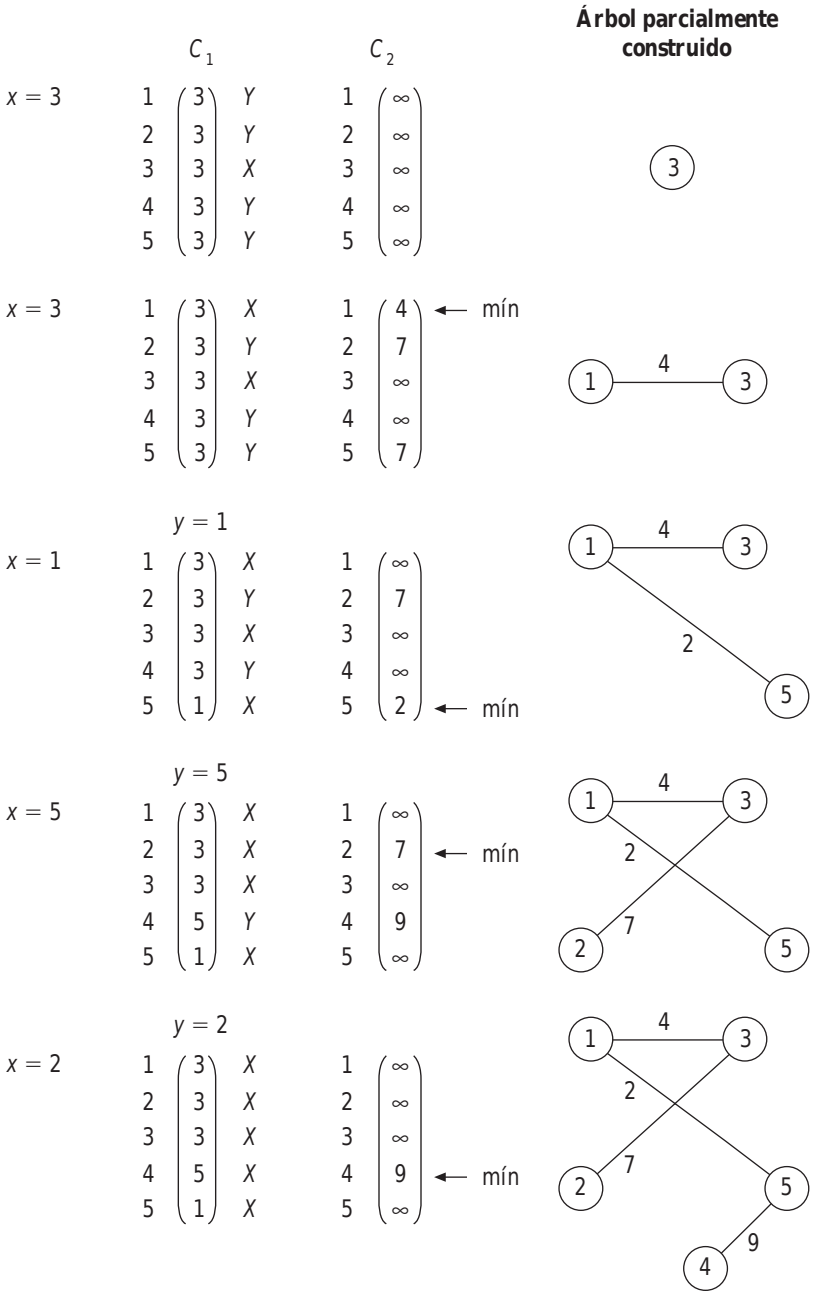
Se considerará la gráfica que se muestra en la figura 3-14.

FIGURA 3-14 Gráfica para demostrar el algoritmo de Prim.



Con la figura 3-15 se demostrará la forma en que funciona el algoritmo de Prim para generar un árbol de expansión mínima. Se supondrá que inicialmente se seleccionó el vértice 3.

FIGURA 3-15 Determinación de un árbol de expansión mínima aplicando el algoritmo de Prim y vértice inicial 3.



Para el algoritmo de Prim, siempre que un vértice se agrega al árbol parcialmente construido es necesario examinar cada elemento de C_1. En consecuencia, la complejidad temporal del algoritmo de Prim, tanto en el peor caso como en el caso promedio, es $O(n^2)$, donde n es el número de vértices en V. Observe que la complejidad temporal del algoritmo de Kruskal es $O(m \log m)$, donde m es el número de aristas en E. En caso de que m sea pequeño, es preferible el método de Kruskal. En el peor caso, ya se mencionó, m puede ser igual a $O(n^2)$ y la complejidad temporal del peor caso del algoritmo de Kruskal se vuelve $O(n^2 \log n)$, que es mayor que el del algoritmo de Prim.

3-3 EL PROBLEMA DE LA RUTA MÁS CORTA DE ORIGEN ÚNICO

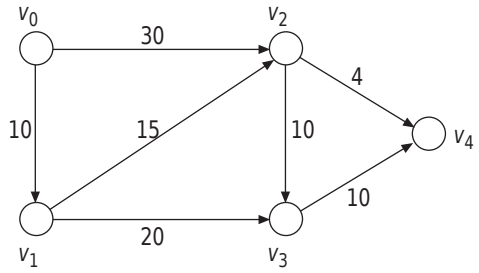
En el problema de la ruta más corta se cuenta con una gráfica dirigida $G = (V, E)$ donde cada arista está asociada con un peso no negativo. Este peso puede considerarse como la longitud de esta arista. La longitud de una ruta en G se define como la suma de las longitudes de las aristas en esta ruta. El problema de la ruta más corta de origen único consiste en encontrar todas las rutas más cortas desde un vértice fuente (o inicial) determinado, que se denota por v_0, a todos los demás vértices en V.

En esta sección se demostrará que dicho problema puede resolverse con el método codicioso. Debido a que este hecho fue indicado por Dijkstra, el algoritmo que se presentará se conoce como algoritmo de Dijkstra. La esencia de este algoritmo es bastante semejante a la del algoritmo del árbol de expansión mínima recientemente presentado. La idea básica es más bien simple: las rutas más cortas desde v_0 hasta todos los demás vértices se encuentran una por una. Primero se encuentra el vecino más próximo de v_0. Luego se encuentra el segundo vecino más próximo de v_0. Este proceso se repite hasta que se encuentra el n-ésimo más próximo de v_0, donde n es el número de vértices distintos a v_0 que hay en la gráfica.

Ejemplo 3-5 Problema de la ruta más corta de origen único

Considere la gráfica dirigida en la figura 3-16. Encuentre todas las rutas más cortas que salen de v_0.

FIGURA 3-16 Gráfica para demostrar el método de Dijkstra.



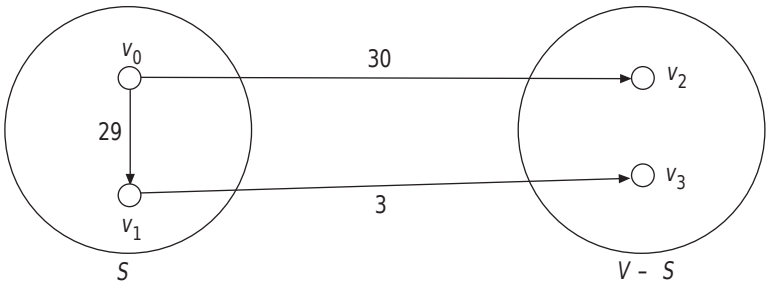
Este algoritmo determina primero que el vecino más próximo de v_0 es v_1. La ruta más corta de v_0 a v_1 es v_0v_1. Luego se determina que el segundo vecino más próximo de v_0 es v_2 y que su ruta correspondiente más corta es $v_0v_1v_2$. Observe que aunque esta ruta consta de sólo dos aristas, sigue siendo más corta que la ruta v_0v_2, que contiene sólo una arista. Se encuentra que los vecinos más próximos tercero y cuarto de v_0 son v_4 y v_3, respectivamente. Todo el proceso puede ilustrarse con la tabla 3-1.

TABLA 3-1 Ilustración para encontrar las rutas más cortas desde v_0.

i	*i*-ésimo vecino más próximo de v_0	Ruta más corta desde v_0 hasta el *i*-ésimo vecino más próximo (longitud)
1	v_1	v_0v_1 (10)
2	v_2	$v_0v_1v_2$ (25)
3	v_4	$v_0v_1v_2v_4$ (29)
4	v_3	$v_0v_1v_3$ (30)

Así como ocurre en el problema del árbol de expansión mínima, en el algoritmo de Dijkstra también se dividen los conjuntos de vértices en dos conjuntos: S y $V - S$, donde S contiene a todos los i vecinos más próximos de v_0 que se han encontrado en los i primeros pasos. Así, en el $(i + 1)$-ésimo paso, la tarea es encontrar el $(i + 1)$-ésimo vecino más próximo de v_0. En este momento, es muy importante no llevar a cabo una acción incorrecta. Considere la figura 3-17. En esta figura se muestra que ya se ha encontrado el primer vecino más próximo de v_0, que es v_1. Puede parecer que como v_1v_3 es el vínculo más corto entre S y $V - S$, debería escogerse a v_1v_3 como la siguiente arista y a v_3 como el segundo vecino más próximo. Esto sería erróneo porque se tiene interés en encontrar las rutas más cortas que salen de v_0. Para este caso, el segundo vecino más próximo es v_2, no v_3.

FIGURA 3-17 Dos conjuntos de vértices, S y $V - S$.



A continuación se explicará un truco importante usado en el algoritmo de Dijkstra para encontrar de manera eficiente el siguiente vecino más próximo de v_0. Esto puede explicarse si se considera la figura 3-16. $L(v_i)$ denota la distancia más corta de v_0 a v_1 que se ha encontrado hasta el momento. Desde el principio, $S = \{v_0\}$ y se tiene

$$L(v_1) = 10$$
y $$L(v_2) = 30$$

ya que v_1 y v_2 están unidos a v_0.

Debido a que $L(v_1)$ es la ruta más corta, v_1 es el primer vecino más próximo de v_0. Sea $S = \{v_0, v_1\}$. Así, sólo v_2 y v_3 están unidos con S. Para v_2, su $L(v_2)$ previo era igual a 30. No obstante, después que v_1 se coloca en S, es posible usar la ruta $v_0v_1v_2$ cuya longitud es $10 + 15 = 25 < 30$. Así, en cuanto corresponde a v_2, su $L(v_2)'$ se calcula como:

$$\begin{aligned} L(v_2)' &= \text{mín}\{L(v_2), L(v_1) + \text{longitud de } v_1v_2\} \\ &= \text{mín}\{30,\ 10 + 15\} \\ &= 25. \end{aligned}$$

El análisis anterior muestra que la distancia más corta de v_0 a v_2 encontrada hasta el momento puede no ser suficientemente corta debido al vértice que acaba de agregarse. Si ocurre esta situación, es necesario actualizar la distancia más corta.

Sea u el último vértice que se ha agregado a S. Sea $L(w)$ la distancia más corta de v_0 a w que se ha encontrado hasta el momento. Sea $c(u, w)$ la longitud de la arista que une u y w. Así, es necesario actualizar $L(w)$ según la siguiente fórmula:

$$L(w) = \text{mín}(L(w), L(u) + c(u,w)).$$

A continuación se resume el algoritmo de Dijkstra para resolver el problema de la ruta más corta de origen único.

Algoritmo 3-4 □ Algoritmo de Dijkstra para generar rutas más cortas de origen único

Input: Una gráfica dirigida $G = (V, E)$ y un vértice fuente v_0. Para cada arista $(u, v) \in E$, hay un número no negativo $c(u, v)$ asociado con ésta. $|V| = n + 1$.

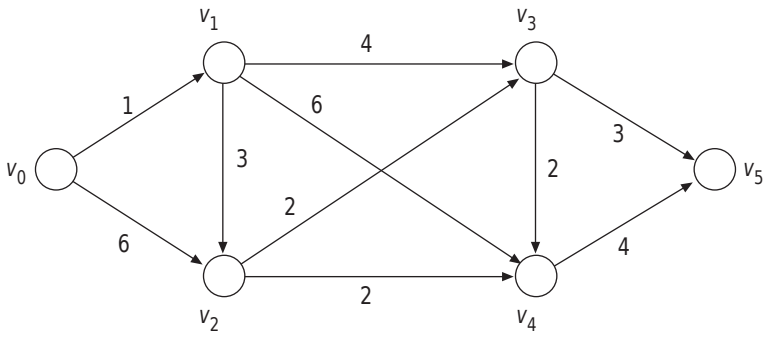
Output: Para cada $v \in V$, la longitud de una ruta más corta de v_0 a v.

```
S := {v0}
For i := 1 to n do
Begin
    If (v0, vi) ∈ E then
        L(vi) := c(v0, vi)
    else
        L(vi) := ∞
End
For i := 1 to n do
Begin
    Elegir u de V – S de modo que L(u) sea el menor
    S := S ∪ {u} (* colocar u en S*)
    Para todo w en V – S do
        L(w) := mín(L(w), L(u) + c(u, w))
End
```

Ejemplo 3-6 Algoritmo de Dijkstra

Considere la gráfica dirigida que se muestra en la figura 3-18.

FIGURA 3-18 Gráfica dirigida ponderada.



El algoritmo de Dijkstra procederá de la siguiente forma:

1. $S = \{v_0\}$

   $L(v_1) = 1$

   $L(v_2) = 6$

   Todas las demás $L(v_i)$ son iguales a ∞.

   $L(v_1)$ es la más pequeña. v_0v_1 es la ruta más corta de v_0 a v_1.

   $S = \{v_0, v_1\}$

2. $L(v_2) = \text{mín}(6, L(v_1) + c(v_1, v_2))$

   $= \text{mín}(6, 1 + 3)$

   $= 4$

   $L(v_3) = \text{mín}(\infty, L(v_1) + c(v_1, v_3))$

   $= 1 + 4$

   $= 5$

   $L(v_4) = \text{mín}(\infty, L(v_1) + c(v_1, v_4))$

   $= 1 + 6$

   $= 7$

   $L(v_2)$ es la más pequeña. $v_0v_1v_2$ es la ruta más corta de v_0 a v_2.

   $S = \{v_0, v_1, v_2\}$

3. $L(v_3) = \text{mín}(5, L(v_2) + c(v_2, v_3))$

   $= \text{mín}(5, 4 + 2)$

   $= 5$

   $L(v_4) = \text{mín}(7, L(v_2) + c(v_2, v_4))$

   $= \text{mín}(7, 4 + 2)$

   $= 6$

   $L(v_3)$ es la más pequeña. $v_0v_1v_3$ es la ruta más corta de v_0 a v_3.

   $S = \{v_0, v_1, v_2, v_3\}$

4. $L(v_4) = \text{mín}(6, L(v_3) + c(v_3, v_4))$

   $= \text{mín}(6, 5 + 2)$

   $= 6$

   $L(v_5) = \text{mín}(\infty, L(v_3)\, c(v_3, v_5))$

   $= 5 + 3$

   $= 8$

$L(v_4)$ es la más pequeña. $v_0v_1v_2v_4$ es la ruta más corta de v_0 a v_4.

$S = \{v_0, v_1, v_2, v_3, v_4\}$

5. $L(v_5) = \text{mín}(8, L(v_4)\ c(v_4, v_5))$
$= \text{mín}(8, 6 + 4)$
$= 8$

$v_0v_1v_3v_5$ es la ruta más corta de v_0 a v_5.

En la tabla 3-2 se resume la salida.

TABLA 3-2 Rutas más cortas desde el vértice v_0.

Vértice	Distancia más corta a v_0 (longitud)
v_1	v_0v_1 (1)
v_2	$v_0v_1v_2$ (1 + 3 = 4)
v_3	$v_0v_1v_3$ (1 + 4 = 5)
v_4	$v_0v_1v_2v_4$ (1 + 3 + 2 = 6)
v_5	$v_0v_1v_3v_5$ (1 + 4 + 3 = 8)

Complejidad temporal del algoritmo de Dijkstra

Es fácil ver que la complejidad temporal del peor caso del algoritmo de Dijkstra es $O(n^2)$ debido a las operaciones repetidas para calcular $L(w)$. Por otro lado, el número mínimo de pasos para resolver el problema de la ruta más corta de fuente única es $\Omega(e)$, donde e es el número de aristas en la gráfica porque es necesario examinar cada arista. En el peor caso, $\Omega(e) = \Omega(n^2)$. Por consiguiente, en este sentido el algoritmo de Dijkstra es óptimo.

3-4 PROBLEMA DE MEZCLAR 2 LISTAS (2-WAY MERGE)

Se cuenta con dos listas ordenadas L_1 y L_2, $L_1 = (a_1, a_2, \ldots, a_{n_1})$ y $L_2 = (b_1, b_2, \ldots, b_{n_2})$. L_1 y L_2 pueden fusionarse en una lista ordenada si se aplica el algoritmo de mezcla lineal que se describe a continuación.

Algoritmo 3-5 □ Algoritmo de mezcla lineal

Input: Dos listas ordenadas, $L_1 = (a_1, a_2, \ldots, a_{n_1})$ y $L_2 = (b_1, b_2, \ldots, b_{n_2})$.

Output: Una lista ordenada que consta de los elementos en L_1 y L_2.

```
Begin
    i := 1
    j := 1
do
    if a_i > b_j then output b_j y j := j + 1
    else output a_j e i := i + 1
while (i ≤ n_1 y j ≤ n_2)
    if i > n_1 then output b_j, b_{j+1}, ..., b_{n_2},
    else output a_i, a_{i+1}, ..., a_{n_1}.
End.
```

Puede verse fácilmente que el número de comparaciones requeridas es $m + n - 1$ en el peor caso. Cuando m y n son iguales, puede demostrarse que el número de comparaciones para el algoritmo de mezcla lineal es óptimo. Si se requiere mezclar más de dos listas ordenadas, el algoritmo de mezcla lineal es óptimo y puede seguir aplicándose, ya que mezcla dos listas ordenadas, repetidamente. Estos procesos de mezcla se denominan mezcla de 2 listas porque cada paso de mezcla sólo mezcla dos listas ordenadas. Suponga que se tienen tres listas ordenadas L_1, L_2 y L_3 que constan de 50, 30 y 10 elementos, respectivamente. Así, es posible mezclar L_1 y L_2 para obtener L_4. Este paso de mezcla requiere $50 + 30 - 1 = 79$ comparaciones en el peor caso. Luego se mezclan L_4 y L_3 usando $80 + 10 - 1 = 89$ comparaciones. El número de comparaciones necesarias en esta secuencia de mezclas es 168. En forma alterna, primero pueden mezclarse L_2 y L_3 y luego L_1. El número de comparaciones necesarias es sólo 128. Hay muchas sucesiones de mezcla diferentes que requieren números de comparaciones distintos. Por ahora el interés lo constituye el siguiente problema: hay m listas ordenadas. Cada una de ellas consta de n_i elementos. ¿Cuál es la sucesión óptima del proceso de mezcla para mezclar estas listas ordenadas aplicando el número mínimo de comparaciones?

En lo que sigue, para simplificar el análisis se usará $n + m$, en vez de $n + m - 1$, para indicar el número de comparaciones necesarias para mezclar dos listas de tamaños n y m, respectivamente, ya que evidentemente esto no afecta el diseño del algoritmo. Se considerará un ejemplo en el que se tiene $(L_1, L_2, L_3, L_4, L_5)$ con tamaños (20, 5, 8, 7, 4). Imagine que estas listas se mezclan como se muestra a continuación:

L_1 se mezcla con L_2 para obtener Z_1 con $20 + 5 = 25$ comparaciones
Z_1 se mezcla con L_3 para obtener Z_2 con $25 + 8 = 33$ comparaciones
Z_2 se mezcla con L_4 para obtener Z_3 con $33 + 7 = 40$ comparaciones
Z_3 se mezcla con L_5 para obtener Z_4 con $40 + 4 = 44$ comparaciones
Total $= 142$ comparaciones.

El patrón de mezcla puede representarse mediante un árbol binario como se muestra en la figura 3-19*a*).

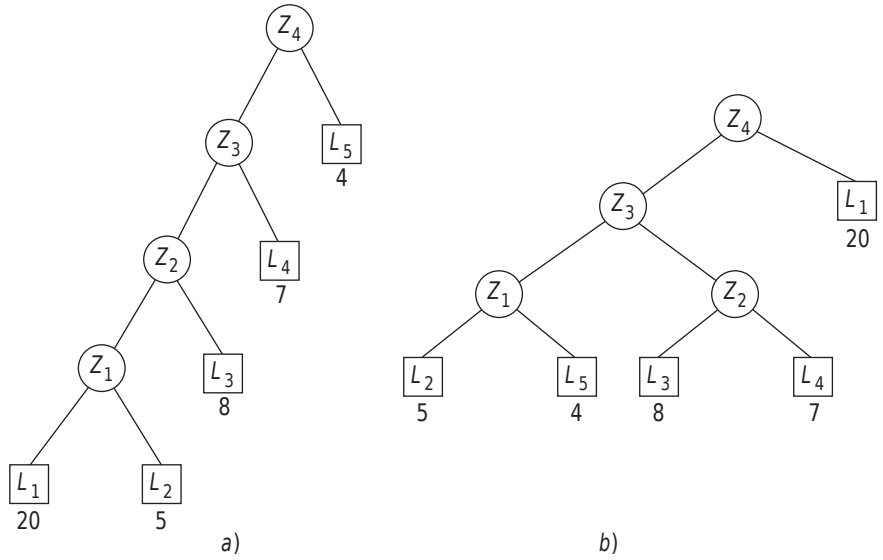
Sea d_i la profundidad de un nodo hoja del árbol binario. Sea n_i el tamaño de la lista L_i asociada con este nodo hoja. Así, puede verse fácilmente que el número total de comparaciones correspondiente a este proceso de mezcla es $\sum_{i=1}^{5} d_i n_i$. En nuestro caso, $d_1 = d_2 = 4$, $d_3 = 3$, $d_4 = 2$ y $d_5 = 1$. Entonces, el número total de comparaciones necesarias puede calcularse como $4 \cdot 20 + 4 \cdot 5 + 3 \cdot 8 + 2 \cdot 7 + 1 \cdot 4 = 80 + 20 + 24 + 14 + 4 = 142$, que es correcto.

Suponga que se usa un método codicioso en el que siempre se mezclan dos listas presentes más cortas. Entonces el patrón de mezcla es

L_2 se mezcla con L_5 para obtener Z_1 con $5 + 4 = 9$ comparaciones
L_3 se mezcla con L_4 para obtener Z_2 con $8 + 7 = 15$ comparaciones
Z_1 se mezcla con Z_2 para obtener Z_3 con $9 + 15 = 24$ comparaciones
Z_3 se mezcla con L_1 para obtener Z_4 con $24 + 20 = 44$ comparaciones
Total $= 92$ comparaciones.

El proceso de mezcla anterior se muestra como un árbol binario en la figura 3-19*b*).

FIGURA 3-19 Secuencias de mezcla distintas.



De nuevo es posible aplicar la fórmula $\sum_{i=1}^{5} d_i n_i$. En este caso, $d_1 = 1$ y $d_2 = d_3 = d_4 = d_5 = 3$. Puede calcularse que el número total de comparaciones es $1 \cdot 20 + 3 \cdot (5 + 4 + 8 + 7) = 20 + 3 \cdot 24 = 92$, que es menor que el cálculo correspondiente a la figura 3-19*a*).

A continuación se presenta un método codicioso para encontrar una mezcla óptima de 2 listas:

Algoritmo 3-6 □ Un algoritmo codicioso para generar un árbol óptimo de mezcla de 2 listas

Input: m listas ordenadas, L_i, $i = 1, 2, \ldots, m$, donde cada lista L_i consta de n_i elementos.

Output: Un árbol de mezcla de 2 listas óptimo.

Paso 1. Generar m árboles, donde cada árbol tiene exactamente un nodo (nodo externo) con peso n_i.

Paso 2. Elegir dos árboles T_1 y T_2 con pesos mínimos.

Paso 3. Crear un nuevo árbol T cuya raíz tenga a T_1 y T_2 como sus subárboles y cuyo peso sea igual a la suma de los pesos de T_1 y T_2.

Paso 4. Reemplazar T_1 y T_2 por T.

Paso 5. Si sólo queda un árbol, stop y return; en caso contrario, ir al paso 2.

Ejemplo 3-7

Se tienen seis listas ordenadas de longitudes 2, 3, 5, 7, 11 y 13. Encontrar un árbol binario extendido con la longitud de ruta ponderada mínima para estas listas.

Primero se mezclan 2 y 3 y se busca la solución del problema para mezclar 5 listas ordenadas con longitudes 5, 5, 7, 11 y 13. Luego se mezclan 5 y 5, y así sucesivamente. La secuencia de mezcla se muestra en la figura 3-20.

FIGURA 3-20 Secuencia de mezcla óptima de 2 listas.

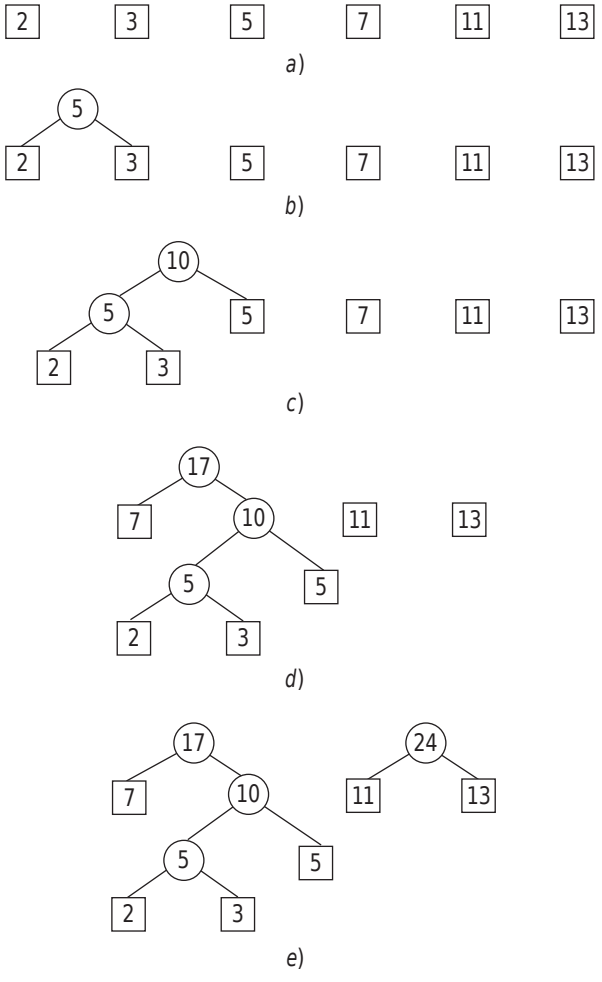
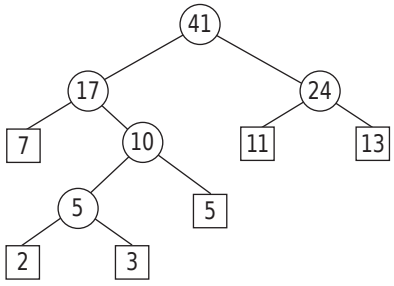


FIGURA 3-20 (*Continuación*).



f)

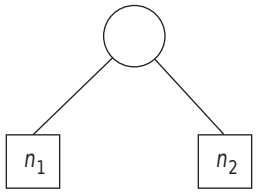
Para demostrar que el algoritmo codicioso anterior es correcto, primero se prueba que existe un árbol de mezcla de 2 listas óptimo donde los dos nodos hoja de tamaños mínimos se asignan a dos hermanos. Sea A un nodo interno situado a la distancia máxima desde la raíz. Por ejemplo, el nodo identificado con 5 en la figura 3-20 es tal nodo. Los hijos de A, por ejemplo L_i y L_j, deben ser nodos hoja. Suponga que los tamaños de los hijos de A son n_i y n_j. Si n_i y n_j no son los más pequeños, es posible intercambiar L_i y L_j con los dos nodos más pequeños sin incrementar los pesos del árbol de mezcla de 2 listas. Así, se obtiene este árbol donde los dos nodos hoja de menor tamaño se asignan como hermanos.

Con base en el análisis anterior, puede suponerse que T es un árbol óptimo de mezcla de 2 listas para $L_1, L_2, \ldots, L_m$ con longitudes $n_1, n_2, \ldots, n_m$, respectivamente, y sin pérdida de generalidad, $n_1 \leq n_2 \ldots \leq n_m$, donde las dos listas de menor longitud, a saber, L_1 y L_2, son hermanos. Sea A el padre de L_1 y L_2. Sea T_1 el árbol en que A se sustituye por una lista de longitud $n_1 + n_2$. Sea $W(X)$ el peso de un árbol de mezcla de 2 listas. Entonces se tiene

$$W(T) = W(T_1) + n_1 + n_2. \qquad \textbf{(3-1)}$$

Ahora es posible demostrar por inducción que el algoritmo codicioso es correcto. Resulta evidente que este algoritmo produce un árbol de mezcla de 2 listas óptimo para $m = 2$. Luego, se supone que el algoritmo produce un árbol de mezcla de 2 listas óptimo para $m - 1$ listas. Para el caso del problema que implique m listas $L_1, L_2, \ldots, L_m$, se combinan las dos primeras listas, a saber, L_1 y L_2. Luego se aplica el algoritmo a este caso del problema con $m - 1$ listas. Sea T_2 el árbol de mezcla de 2 listas óptimo producido por el algoritmo. En T_2 hay un nodo hoja de longitud $n_1 + n_2$. Este nodo se separa de modo que tenga dos hijos, a saber, L_1 y L_2 con longitudes n_1 y n_2, respectivamente, como se muestra en la figura 3-21. Este árbol de nueva creación se denota por T_3. Se tiene

FIGURA 3-21 Un subárbol.



$$W(T_3) = W(T_2) + n_1 + n_2. \quad \textbf{(3-2)}$$

Se afirma que T_3 es un árbol de mezcla de 2 listas óptimo para $L_1, L_2, \ldots, L_m$. Suponga lo contrario. Entonces

$$W(T_3) > W(T),$$

que implica

$$W(T_2) > W(T_1).$$

Esto es imposible, ya que por la hipótesis de inducción T_2 es un árbol de mezcla de 2 listas óptimo para $m - 1$ listas.

Complejidad temporal del algoritmo codicioso para generar un árbol de mezcla de 2 listas óptimo

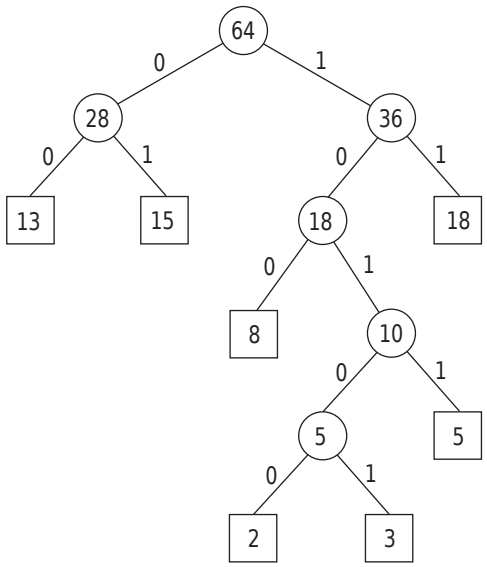
Para los m números dados $n_1, n_2, \ldots, n_m$, es posible construir un mini heap a fin de representar estos números donde el valor de la raíz es menor que los valores de sus hijos. Así, la reconstrucción del árbol después de eliminar la raíz, que tiene el menor valor, puede realizarse en un tiempo $O(\log n)$. Y la inserción de un nuevo nodo en un mini heap también puede efectuarse en un tiempo $O(\log n)$. Debido a que el ciclo principal se lleva a cabo $n - 1$ veces, el tiempo total para generar un árbol binario extendido óptimo es $O(n \log n)$.

Ejemplo 3-8 Códigos de Huffman

Considere un problema de telecomunicaciones donde se quiere representar un conjunto de mensajes por medio de una secuencia de ceros y unos. En consecuencia, para enviar un mensaje simplemente se transmite una secuencia de ceros y unos. Una aplicación del árbol binario extendido con la longitud de ruta externa ponderada óptima es la generación de un conjunto óptimo de códigos; es decir, cadenas binarias, para estos

mensajes. Se supondrá que hay siete mensajes cuyas frecuencias de acceso son 2, 3, 5, 8, 13, 15 y 18. Para minimizar los costos de transmisión y decodificación, pueden usarse cadenas cortas para representar mensajes de uso frecuente. Luego es posible construir un árbol binario óptimo extendido al mezclar primero 2 y 3 y luego 5 y 5, y así sucesivamente. El árbol binario extendido se muestra en la figura 3-22.

FIGURA 3-22 Árbol de código de Huffman.



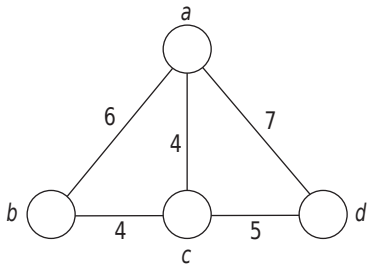
Así, los códigos correspondientes a los mensajes con frecuencias 2, 3, 5, 8, 13, 15 y 18 son 10100, 10101, 1011, 100, 00, 01 y 11, respectivamente. Los mensajes usados más a menudo se codifican mediante códigos cortos.

3-5 EL PROBLEMA DEL CICLO BASE MÍNIMO (MINIMUM CYCLE BASIS) RESUELTO CON EL ALGORITMO CODICIOSO

En esta sección se presentará el problema del ciclo base mínimo y también se demostrará cómo este problema puede resolverse con el algoritmo codicioso.

Considere la figura 3-23. En la gráfica no dirigida de la figura 3-23 hay tres ciclos; específicamente son: {*ab*, *bc*, *ca*}, {*ac*, *cd*, *da*} y {*ab*, *bc*, *cd*, *da*}.

FIGURA 3-23 Gráfica que contiene ciclos.



Mediante una operación idónea, dos ciclos pueden combinarse en otro ciclo. Esta operación es la operación de suma de anillos que se define como sigue: sean A y B dos ciclos. Entonces $C = A \oplus B = (A \cup B) - (A \cap B)$ puede ser un ciclo. Para los ciclos en el caso anterior, sea

$$A_1 = \{ab, bc, ca\}$$
$$A_2 = \{ac, cd, da\}$$
y $$A_3 = \{ab, bc, cd, da\}.$$

Puede demostrarse fácilmente que

$$A_3 = A_1 \oplus A_2$$
$$A_2 = A_1 \oplus A_3$$
y $$A_1 = A_2 \oplus A_3.$$

Este ejemplo muestra que $\{A_1, A_2\}$, $\{A_1, A_3\}$ y $\{A_2, A_3\}$ pueden ser considerados como ciclos base para la gráfica de la figura 3-23 porque para cada uno de ellos, todos los ciclos de esta gráfica pueden ser generados usando ésta. Formalmente, *un ciclo base de una gráfica es un conjunto de ciclos tales que cada ciclo en la gráfica puede generarse aplicando la operación suma de anillos a algunos ciclos de esta base*.

Se asume que cada arista está asociada con un peso. El peso de un ciclo es el peso total de todas las aristas en él. *Éste es el peso total de todos los ciclos del ciclo base*. El problema del ciclo base ponderado se define como sigue: *dada una gráfica, encontrar un ciclo base mínimo para esta gráfica*. Para la gráfica de la figura 3-23, el ciclo base mínimo es $\{A_1, A_2\}$, porque tiene el menor peso.

El método codicioso que resuelve el problema del ciclo base ponderado se apoya en los siguientes conceptos:

1. Es posible determinar el tamaño del ciclo base mínimo. Sea K este tamaño.

2. Suponga que es posible determinar todos los ciclos. (La determinación de todos los ciclos en una gráfica no es nada fácil. Sin embargo, es irrelevante para este problema, por lo que la técnica no se abordará aquí.) Todos los ciclos se clasifican en una sucesión no decreciente según sus pesos.
3. A partir de la sucesión no ordenada de los ciclos, al ciclo base se suman los ciclos uno por uno. Para cada ciclo sumado se comprueba si es una combinación lineal de algunos ciclos que ya existen en el ciclo base parcialmente construido. En caso afirmativo, se elimina el ciclo en cuestión.
4. Detener el proceso si el ciclo base tiene K ciclos.

Nuestro método codicioso tiene un típico enfoque codicioso. Suponga que se quiere encontrar un conjunto de objetos al minimizar algún parámetro. Algunas veces es posible determinar el tamaño mínimo K de este conjunto y después aplicar el método codicioso para agregar objetos uno por uno a este conjunto hasta que el tamaño del conjunto es igual a K. En la siguiente sección se presentará otro ejemplo con el mismo enfoque del método codicioso.

Ahora regresaremos al problema original. Primero se demostrará que es posible determinar el tamaño de un ciclo base mínimo. No se proporcionará una demostración formal del método. En vez de ello, de manera informal el concepto se ilustrará mediante un ejemplo. Considere la figura 3-24. Suponga que se construye un árbol de expansión de esta gráfica como se muestra en la figura 3-25*a*). Este árbol de expansión carece de ciclo. No obstante, si al árbol de expansión se le agrega una arista, como se muestra en la figura 3-25*b*), entonces se forma un ciclo.

De hecho, como se muestra en la figura 3-25, cada adición de una arista crea un nuevo ciclo independiente. El número de ciclos independientes es igual al número de aristas que es posible agregar al árbol de expansión. Ya que el número de aristas en un árbol de expansión es $|V|-1$, entonces el total de aristas que puede agregarse es igual a

$$|E|-(|V|-1) = |E|-|V|+1.$$

FIGURA 3-24 Gráfica que muestra la dimensión de un ciclo base mínimo.

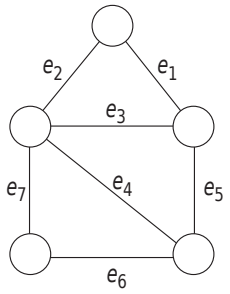
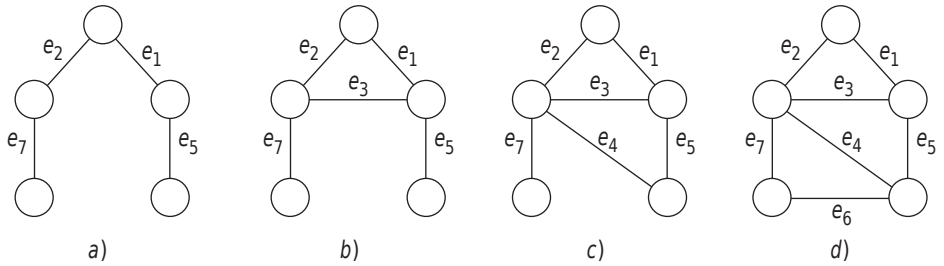


FIGURA 3-25 Relación entre un árbol de expansión y los ciclos.

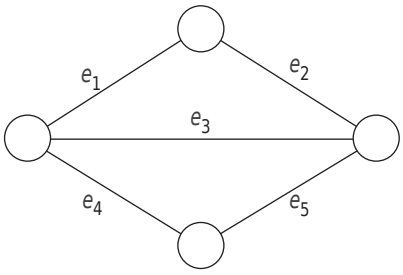


Se ha demostrado la fórmula para encontrar el tamaño de un ciclo base mínimo. Lo único que queda por hacer es demostrar cómo determinar si un ciclo es una combinación lineal de un conjunto de ciclos. Sin embargo, no se pretende describir un método formal. Sólo se describen las técnicas a través de un ejemplo. Para realizar la descripción de las técnicas, los ciclos se representan con una matriz de incidencia. Cada renglón corresponde a un ciclo y cada columna corresponde a una arista. La comprobación de dependencia, o independencia, puede hacerse por eliminación Gaussiana, excepto cuando una operación implica dos renglones en una operación de suma de anillos (u or-exclusivo). A continuación esto se ilustra con un ejemplo. Considere la figura 3-26. Hay tres ciclos: $C_1 = \{e_1, e_2, e_3\}$, $C_2 = \{e_3, e_4, e_5\}$ y $C_3 = \{e_1, e_2, e_5, e_4\}$.

La matriz que representa los dos primeros ciclos se muestra a continuación:

$$\begin{array}{c} \\ C_1 \\ C_2 \end{array} \begin{array}{c} \begin{array}{ccccc} e_1 & e_2 & e_3 & e_4 & e_5 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & & \\ & & 1 & 1 & 1 \end{bmatrix} \end{array}$$

FIGURA 3-26 Gráfica que ilustra la comprobación de independencia de los ciclos.



Si se agrega C_3, la matriz se convierte en:

$$\begin{array}{c} \\ C_1 \\ C_2 \\ C_3 \end{array} \begin{array}{c} \begin{array}{ccccc} e_1 & e_2 & e_3 & e_4 & e_5 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & & \\ & & 1 & 1 & 1 \\ 1 & 1 & & 1 & 1 \end{bmatrix} \end{array}$$

La operación o excluyente sobre el renglón 1 y el renglón 3 produce la siguiente matriz:

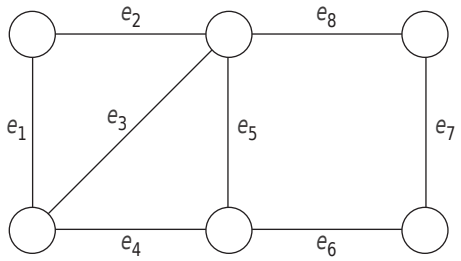
$$\begin{array}{c} \\ C_1 \\ C_2 \\ C_3 \end{array} \begin{array}{c} \begin{array}{ccccc} e_1 & e_2 & e_3 & e_4 & e_5 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & & \\ & & 1 & 1 & 1 \\ & & 1 & 1 & 1 \end{bmatrix} \end{array}$$

La operación excluyente sobre C_2 y C_3 produce un renglón vacío que muestra que C_3 es una combinación lineal de C_1 y C_2.

A continuación se completa el análisis mediante la consideración de la figura 3-27. Se supone que el peso de cada arista es 1. Hay seis ciclos que se muestran a continuación:

$C_1 = \{e_1, e_2, e_3\}$
$C_2 = \{e_3, e_5, e_4\}$
$C_3 = \{e_2, e_5, e_4, e_1\}$
$C_4 = \{e_8, e_7, e_6, e_5\}$
$C_5 = \{e_3, e_8, e_7, e_6, e_4\}$
$C_6 = \{e_2, e_8, e_7, e_6, e_4, e_1\}$.

FIGURA 3-27 Gráfica que ilustra el proceso de cálculo del ciclo base mínimo.



Para este caso, $|E| = 8$ y $|V| = 6$. Así, $K = 8 - 6 + 1 = 3$. Este método codicioso funciona como sigue:

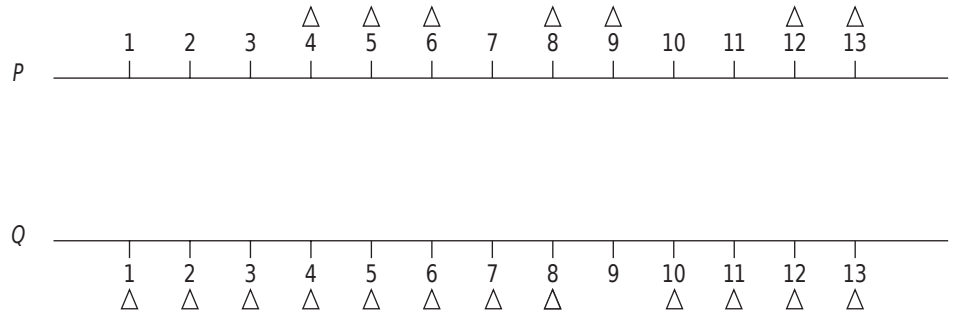
1. Se agrega C_1.
2. Se agrega C_2.
3. Se agrega C_3 y se elimina porque se encuentra que C_3 es una combinación lineal de C_1 y C_2.
4. Se agrega C_4. Debido a que ahora el ciclo base ya tiene tres ciclos, el proceso se detiene cuando $K = 3$. Se encuentra que el ciclo base mínimo es $\{C_1, C_2, C_4\}$.

3-6 SOLUCIÓN POR EL MÉTODO CODICIOSO DEL PROBLEMA DE 2 TERMINALES (UNO A CUALQUIERA/UNO A MUCHOS)

En el diseño VLSI existe un problema de direccionamiento de un canal. Hay muchas versiones de este problema y en el capítulo 5 también se presentará uno de estos problemas que puede resolverse con el método del algoritmo *A**. El problema de direccionamiento de un canal que se analizará en esta sección es una versión muy simplificada del problema general de direccionamiento de un canal. La razón por la que este problema se denomina de 2 terminales-uno a cualquiera se clarificará en el siguiente párrafo.

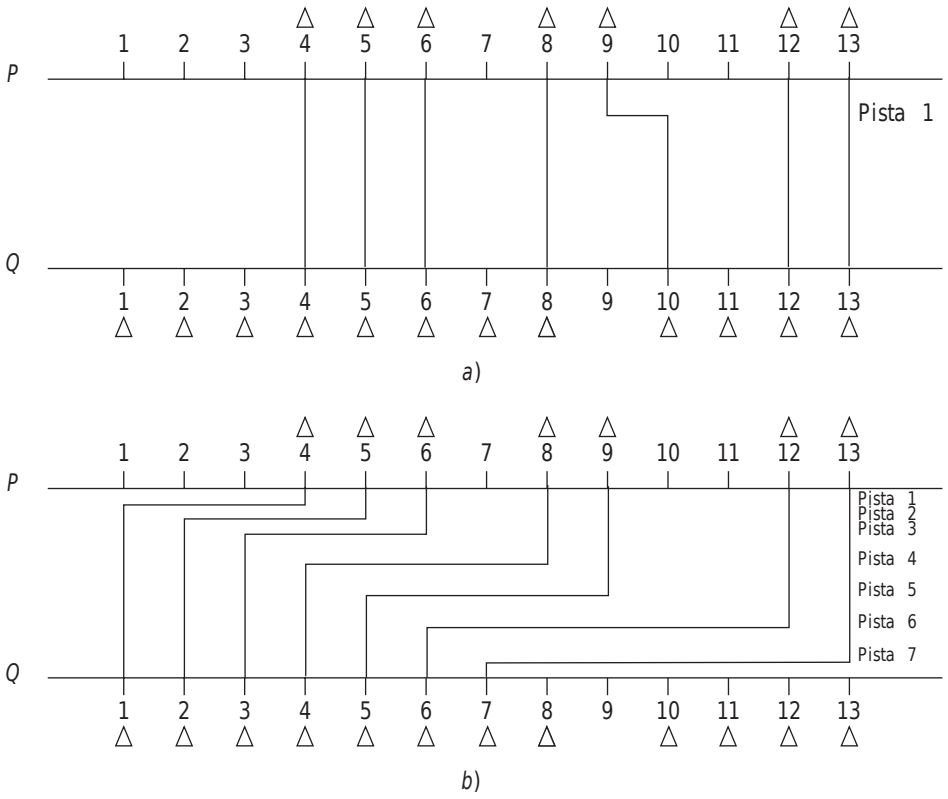
Considere la figura 3-28, donde se han marcado algunas terminales. Cada terminal marcada de un renglón superior debe conectarse o unirse con una terminal de un renglón inferior de manera uno a uno. Se requiere que ningún par de líneas se corte. Además, todas las líneas son verticales u horizontales. Toda línea horizontal corresponde a una pista (track).

FIGURA 3-28 Problema de 2 terminales (uno a cualquiera).



Puede haber muchas soluciones para el mismo problema. En la figura 3-29 se muestran dos soluciones factibles para el caso del problema de la figura 3-28. Puede verse que la solución de la figura 3-29*a*) usa menos pistas que la de la figura 3-28*b*).

FIGURA 3-29 Dos soluciones factibles para el ejemplo del problema en la figura 3-24.

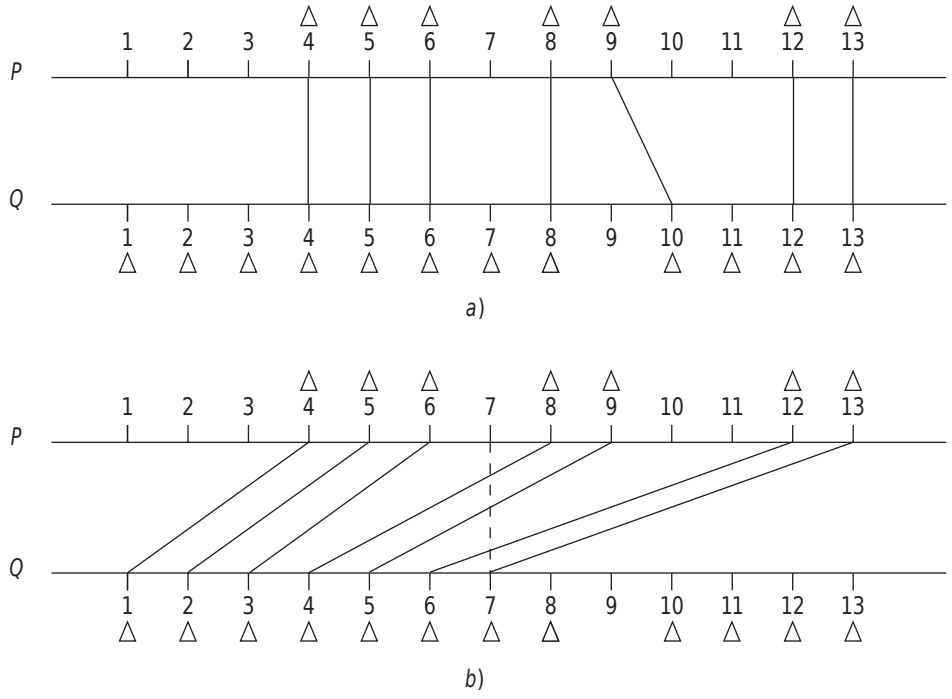


Para un problema real de direccionamiento de un canal, por supuesto que sería deseable minimizar el número de pistas. En esta versión simplificada sólo se intenta minimizar la densidad de una solución.

Para explicar el significado de "densidad", las dos soluciones vuelven a trazarse (figura 3-30) en la figura 3-29. Ahora el lector debe imaginar que para explorar de izquierda a derecha se utiliza una recta vertical. Esta recta vertical "cortará" las líneas de la solución. En cada punto, la densidad local de la solución es el número de líneas que corta la recta vertical.

Por ejemplo, como se muestra en la figura 3-30*b*), la recta vertical (identificada por la línea punteada o discontinua) corta cuatro líneas en la terminal 7. Así, la densidad local en la terminal 7 de la solución en la figura 3-30*b*) es 4. La densidad de una solución es la máxima densidad local. Puede verse fácilmente que las densidades de las soluciones en la figura 3-30*a*) y 3-30*b*) son 1 y 4, respectivamente.

FIGURA 3-30 Figura 3-29 vuelta a trazar incorporando la recta de búsqueda.



¿Por qué se usa la densidad como el parámetro que se intenta minimizar? La respuesta es sencilla: así se simplifica el problema. Si como parámetro se usa el número de pistas, es más difícil resolver el problema. Por otra parte, la densidad es una cota inferior del número de pistas. En consecuencia, es posible usarla como indicador de cuán buena es la solución.

Para encontrar una solución con la densidad mínima se usa el mismo truco que en la última sección, cuando se intentó resolver el problema del ciclo base mínimo. Es decir, se tiene un método para determinar la densidad mínima de un caso de un problema. Una vez que se ha determinado ésta, puede aplicarse el método codicioso para encontrar una solución con esta densidad mínima. No se analizará cómo se determina esta densidad mínima, ya que es demasiado complicado e irrelevante para el análisis del método codicioso. La parte fundamental de este análisis es demostrar que nuestro método codicioso siempre encuentra tal solución con densidad mínima.

A continuación se ilustrará la forma en que funciona el método codicioso. Se nos proporciona un problema ejemplo donde ya se ha determinado la densidad mínima. Para el ejemplo del problema que se muestra en la figura 3-28, la densidad mínima es 1.

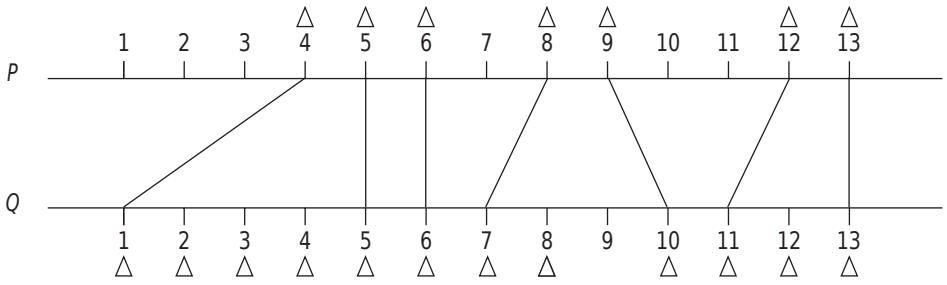
Sean $P_1, P_2, \ldots, P_n$ las terminales del renglón (que representa la pista) superior y $Q_1, Q_2, \ldots, Q_m, m > n$, las terminales del renglón inferior que pueden unirse. También se supone que todas las P_i y las Q_i están identificadas de izquierda a derecha. Es decir, si $j > i$, entonces $P_j(Q_j)$ está a la derecha de $P_i(Q_i)$.

Dadas la densidad mínima d y las notaciones anteriores, nuestro método codicioso procede como sigue:

1. P_1 es conectada con Q_1.
2. Después que P_i se une con, por ejemplo, Q_j, se comprueba si P_{i+1} puede ser conectada a Q_{j+1}. Si la adición de la recta que une P_{i+1} con Q_{j+1} incrementa la densidad a $d + 1$, entonces se intenta conectar P_{i+1} con Q_{j+2}.
3. El paso anterior se repite hasta que todas las P_i están conectadas.

A continuación se ilustrará cómo funciona nuestro método codicioso al aplicarlo al caso del problema en la figura 3-28. Primero se decide que $d = 1$. Con esta información, las terminales se unen como en la figura 3-31. Se observa que P_5 no puede unirse con Q_2, Q_3 y Q_4 porque las uniones incrementarían la densidad a 2, lo cual excede la densidad mínima. De manera semejante y exactamente por la misma razón, no es posible unir P_9 con Q_8.

FIGURA 3-31 Caso del problema de la figura 3-28 resuelto con el método codicioso.



Observe que nuestro algoritmo jamás produce alguna solución cuando los segmentos de recta que unen terminales se cortan entre sí, como en la figura 3-32. En realidad, puede verse fácilmente que jamás se tendrá una unión así porque esta unión puede transformarse en otra unión con una densidad menor o igual, como se muestra en la figura 3-33.

FIGURA 3-32 Intersección cruzada.

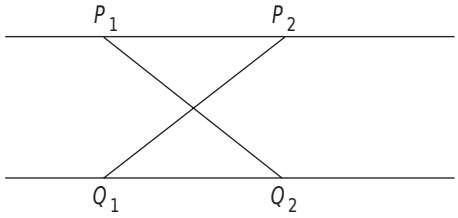
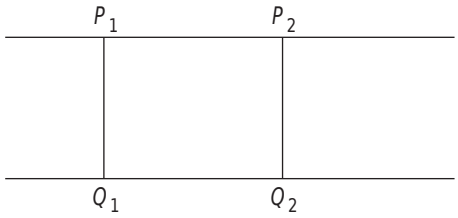


FIGURA 3-33 Interconexión transformada a partir de las intersecciones de la figura 3-33.



Finalmente se mostrará cómo funciona nuestro método codicioso. Debido a que se supone que la densidad mínima es d, entonces existe una solución factible S_1, $((P_1, Q_{j_1}), (P_2, Q_{j_2}), \ldots, (P_n, Q_{j_n}))$, con densidad d. Se demostrará que esta solución puede transformarse en la solución S_2, $((P_2, Q_{i_1}), (P_2, Q_{i_2}), \ldots, (P_n, Q_{i_n}))$, obtenida con nuestro método codicioso. Esto se demostrará por inducción.

Supongamos que las k primeras uniones en S_1 pueden transformarse en las k primeras uniones en S_2 sin violar el requerimiento de la densidad d. Luego se demostrará que esta transformación puede hacerse para $k = k + 1$. Esta hipótesis es trivialmente cierta para $k = 1$. Si esto se cumple para k, entonces se tiene una solución parcial $((P_1, Q_{i_1}), (P_2, Q_{i_2}), \ldots, (P_k, Q_{i_k}))$, sin violar el requerimiento de la densidad d. Considere P_{k+1}. P_{k+1} está unido con $Q_{j_{k+1}}$ en S_1. Suponga que existe una terminal $Q_{j_{k+1}}$ a la izquierda de $Q_{j_{k+1}}$ con la que es posible unir con P_{k+1} sin violar el requerimiento de la densidad mínima, entonces P_{k+1} puede unirse con esta terminal y esto es lo que se obtiene al usar nuestro método codicioso. Si ocurre lo contrario, entonces nuestro método codicioso también une P_{k+1} con $Q_{j_{k+1}}$.

El análisis anterior muestra que cualquier solución factible puede transformarse en una solución obtenida al aplicar el método codicioso y, en consecuencia, éste funciona.

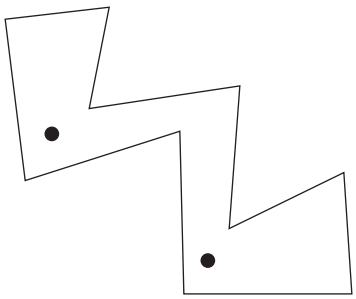
En este último ejemplo en que se utiliza el método del algoritmo codicioso, se determina el valor mínimo del parámetro que se intenta optimizar y luego el método co-

dicioso se aplica directamente para alcanzar esta meta. En el problema del ciclo base, los ciclos se agregaban uno por uno hasta que se obtenía el tamaño mínimo del ciclo base. En este problema primero se calcula la densidad mínima. El algoritmo codicioso une codiciosamente las terminales, y en cualquier instante nuestro algoritmo asegura que la densidad resultante no excede a *d*.

3-7 EL PROBLEMA DEL MÍNIMO DE GUARDIAS COOPERATIVOS PARA POLÍGONOS DE 1-ESPIRAL RESUELTO POR EL MÉTODO CODICIOSO

El problema del mínimo de guardias cooperativos es una variante del problema de la galería de arte, que se define como sigue: se tiene un polígono, que representa la galería de arte, y se requiere colocar el número mínimo de guardias en el polígono de modo que cada punto del polígono sea visible por lo menos para un guardia. Por ejemplo, considere la figura 3-34. Para este polígono, el número mínimo de guardias necesarios es 2. El problema de la galería de arte es un problema NP-difícil (NP-hard).

FIGURA 3-34 Una solución del problema de la galería de arte.

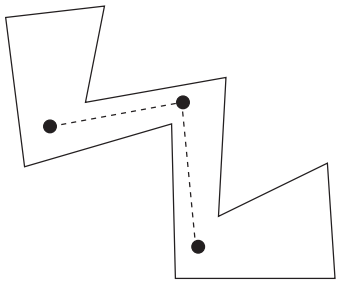


El problema del mínimo de guardias cooperativos agrega más restricciones al problema de la galería de arte. Observe que puede resultar extremadamente peligroso que un guardia esté en una parte de la galería de arte si los otros guardias no pueden verlo. La relación entre los guardias se representa mediante una gráfica de visibilidad *G* de los guardias. En *G*, cada guardia se representa por un vértice y entre dos vértices hay una arista si y sólo si los guardias correspondientes pueden verse mutuamente. Además de encontrar un conjunto de guardias que puedan ver el polígono dado, también se requiere que la gráfica de visibilidad de estos guardias sea conexa. En otras palabras, se requiere que ningún guardia esté aislado y que entre cada par de guardias haya una ruta. Este problema se denomina problema del *mínimo de guardias cooperativos*.

Considere nuevamente la figura 3-34. La gráfica de visibilidad correspondiente a los dos guardias es evidentemente un conjunto de dos vértices que están aislados. Para

cumplir el requerimiento del problema del mínimo de guardias cooperativos es necesario agregar otro guardia, lo cual se muestra en la figura 3-35.

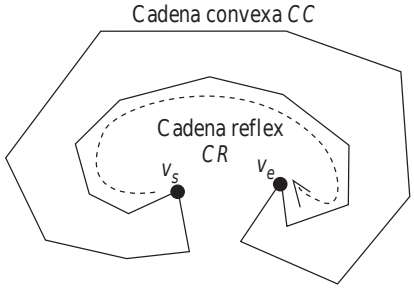
FIGURA 3-35 Una solución del problema del mínimo de guardias cooperativos de la figura 3-34.



De nuevo puede demostrarse que el problema del mínimo de guardias cooperativos es NP-difícil. Por lo tanto, es bastante improbable que este problema del mínimo de guardias cooperativos pueda resolverse con cualquier algoritmo poligonal sobre polígonos generales. Sin embargo, se demostrará que existe un algoritmo codicioso para este problema para polígonos de 1-espiral.

Antes de definir este tipo de polígonos, primero se definen las cadenas reflex (convexas). *Una cadena reflex* (*convexa*), *denotada por CR* (*CC*), *de un polígono simple es una cadena de aristas de este polígono si todos los vértices en esta cadena son reflex* (*convexos*) *con respecto al interior del polígono*.* También se estipula que una cadena reflex se refiere a una cadena reflex *máxima*; es decir, que no está contenida en ninguna otra cadena reflex. Un *polígono P de 1-espiral* es un polígono simple cuya frontera puede partirse en una cadena reflex y una cadena convexa. En la figura 3-36 se muestra un polígono de 1-espiral típico.

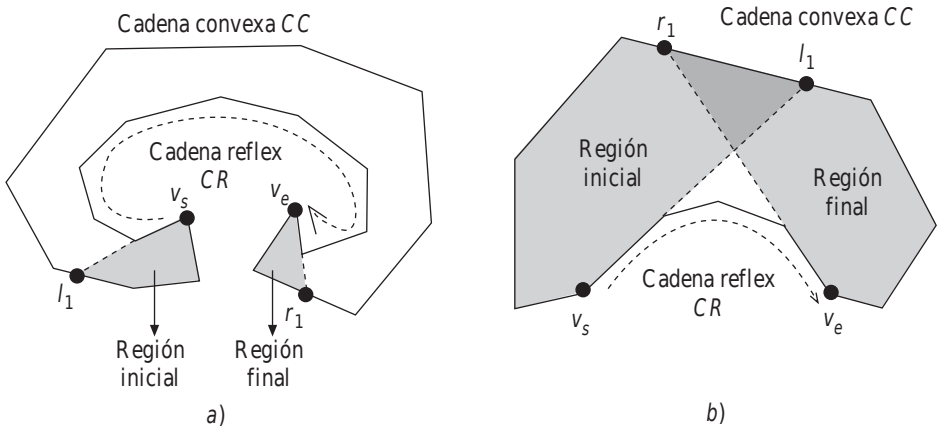
FIGURA 3-36 Un polígono de 1-espiral típico.



* En esta sección los autores utilizan el mismo texto para definir a dos términos (cadena reflex y cadena convexa). *(N. del R.T.)*

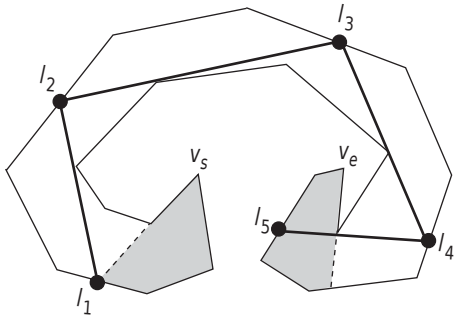
Cuando la frontera de un polígono de 1-espiral se recorre en sentido contrario al movimiento de las manecillas del reloj, el vértice inicial (o final) de la cadena reflex se denomina $v_s(v_e)$. Dados v_s y v_e, ahora es posible definir dos regiones, denominadas región inicial y región final. Se trazará una recta, empezando en $v_s(v_e)$ a lo largo de la primera (última) arista de la cadena reflex hasta que toca la frontera del polígono en $l_1(r_1)$. Este segmento de recta $\overline{v_s l_1}(\overline{v_e r_1})$ y la primera (última) parte de la cadena convexa que empieza en $v_s(v_e)$ forma una región que se denomina *región inicial* (*final*). En la figura 3-37 se muestran dos ejemplos. Observe que las regiones inicial y final pueden traslaparse, por lo que ahí se requieren dos guardias estacionados: uno en la región inicial y otro en la región final.

FIGURA 3-37 Las regiones inicial y final en polígonos de 1-espiral.



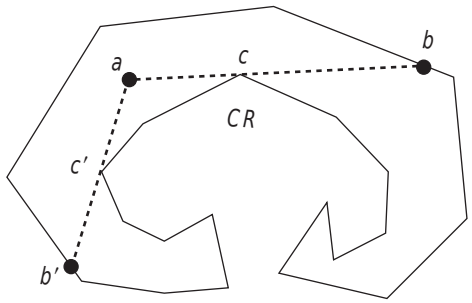
Nuestro algoritmo codicioso resuelve el problema del mínimo de guardias cooperativos para polígonos de 1-espiral como sigue. Primero se ubica un guardia en l_1. Luego se plantea la pregunta: ¿qué área puede observar este guardia? Esto puede contestarse trazando una tangente de CR empezando en l_1 hasta que toque un punto l_2 en CC. Se coloca un guardia en l_2. Esto es bastante razonable. Observe que si a la izquierda de $\overline{l_1 l_2}$ no hay guardia, entonces la gráfica de visibilidad resultante no es conexa. Este proceso se repite hasta que se alcanza la región final. En la figura 3-38 se muestra un caso típico. En esa figura, los guardias estacionados en l_1, l_2, l_3, l_4 y l_5 constituyen una solución óptima del problema del mínimo de guardias cooperativos.

FIGURA 3-38 Un conjunto de guardias $\{l_1, l_2, l_3, l_4, l_5\}$ en un polígono de 1-espiral.



Antes de proporcionar el algoritmo formal es necesario definir algunos nuevos conceptos. Considere la figura 3-39. Sea a un punto en un polígono de 1-espiral P. Es posible trazar dos tangentes con respecto a CR desde a. Si el exterior de CR está completamente a la derecha (izquierda) de una tangente trazada desde a, se denomina la *tangente izquierda* (*derecha*) de a con respecto a CR. A continuación se trazará la tangente izquierda (derecha) de a con respecto a CR hasta tocar la frontera de P en $b(b')$. Así, $\overline{ab}(\overline{ab'})$ se denomina *segmento de recta de apoyo izquierdo* (*derecho*) con respecto a a. Esto se ilustra en la figura 3-39. Si un punto a está en la región inicial (final), entonces el segmento de recta de apoyo derecho (izquierdo) con respecto a a se define como $\overline{av_s}(\overline{av_e})$.

FIGURA 3-39 Los segmentos de recta de apoyo izquierdo y derecho con respecto a a.



A continuación se presenta un algoritmo para resolver el problema del mínimo de guardias cooperativos para polígonos de 1-espiral.

Algoritmo 3-7 □ Un algoritmo para resolver el problema del mínimo de guardias cooperativos para polígonos de 1-espiral

Input: Un polígono de 1-espiral P.

Output: Un conjunto de puntos que es la solución del problema del mínimo de guardias cooperativos.

Paso 1. Buscar la cadena reflex CR y la cadena convexa CC de P.

Paso 2. Buscar los puntos de intersección l_1 y r_1 de CC con las rectas dirigidas que empiezan desde v_s y v_e pasando por las aristas primera y última de CR, respectivamente.

Paso 3. Hacer $k = 1$.

Mientras l_k no está en la región final, hacer (*inicio de ciclo*)

Dibujar la tangente izquierda de l_k con respecto a CR hasta que toca CC en l_{k+1}. ($\overline{l_k l_{k+1}}$ es un segmento de recta de apoyo izquierdo con respecto a l_k.)
Hacer $k = k + 1$.

End While. (*fin de ciclo*)

Paso 4. Report $\{l_1, l_2, \ldots, l_k\}$.

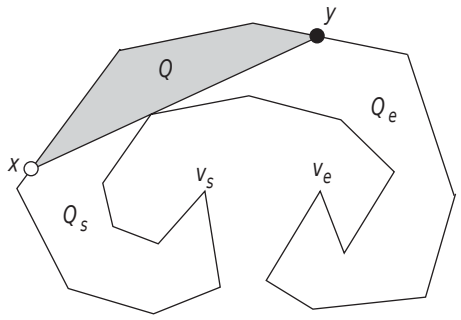
A continuación se establecerá que el algoritmo 3-7 es correcto. Primero se presentan algunas notaciones y términos. Una subcadena de la frontera del polígono P desde el punto a hasta el punto b en sentido contrario al movimiento de las manecillas del reloj se denota por $C[a, b]$.

Suponga que A es un conjunto de guardias cooperativos que pueden ver todo el polígono simple P y que la gráfica de visibilidad de A es conexa, entonces A se denomina solución factible del problema del mínimo de guardias cooperativos para P. Observe que A no necesariamente es mínima.

Sean los puntos x y y que están en la cadena convexa de P y $\overline{xy}$ un segmento de recta de apoyo. $\overline{xy}$ parte a P en tres subpolígonos, Q, Q_s y Q_e. Q es el subpolígono acotado por $C[y, x]$ y $\overline{xy}$, aunque es exclusiva de x o y. Q_s y Q_e son los otros dos subpolígonos que contienen a v_s y v_e, respectivamente. En la figura 3-40 se ilustra esta situación.

Suponga que A es una solución factible del problema del mínimo de guardias cooperativos para P. Entonces, debe haber por lo menos un guardia de A situado dentro de Q; en caso contrario, la gráfica de visibilidad de A no es conexa porque dos guardias cualesquiera ubicados dentro de Q_s y Q_e, respectivamente, no pueden verse el uno al otro.

FIGURA 3-40 Segmento de recta de apoyo $\overline{xy}$ y las regiones Q, Q_s y Q_e.



Sean P un polígono de 1-espiral y $\{l_1, l_2, \ldots, l_k\}$ el conjunto de puntos resultante del algoritmo 3-7 aplicado a P. Sea L_i la región acotada por $C[l_i, l_{i-1}]$ y $\overline{l_{i-1}l_i}$, aunque es exclusiva de l_{i-1}, para $1 \leq i \leq k$. Sea A cualquier solución factible del problema del mínimo de guardias cooperativos para P. Entonces, hay por lo menos un guardia de A ubicado en cada L_i, para $1 \leq i \leq k$. Además, resulta evidente que los L_i son ajenos. Sea N el tamaño de A; entonces, $N \geq k$. Esto significa que k es el número mínimo de guardias en cualquier solución factible del problema del mínimo de guardias cooperativos para P.

Después de establecer la característica de minimalidad de $\{l_1, l_2, \ldots, l_k\}$, a continuación se establecerá la visibilidad de $\{l_1, l_2, \ldots, l_k\}$. Es decir, debe demostrarse que cada punto en P es visible por lo menos desde un guardia en $\{l_1, l_2, \ldots, l_k\}$. Puede verse que una colección observa el polígono de 1-espiral si y sólo si pueden ver las aristas de la cadena reflex. Sea $\overline{l_i l_{i+1}}$ el segmento de recta que entra en contacto con la cadena reflex en c_i, para $1 \leq i \leq k - 1$, y sean $c_0 = v_s$ y $c_k = v_e$. Resulta evidente que l_i puede ver a $C[c_i, c_{i-1}]$, para $1 \leq i \leq k - 1$. Así, $\{l_1, l_2, \ldots, l_k\}$ puede ver toda la cadena reflex $C[v_s, v_e]$ y en consecuencia puede ver todo el polígono de 1-espiral. Además, resulta evidente que la gráfica de visibilidad de $\{l_1, l_2, \ldots, l_k\}$ es conexa según el algoritmo 3-7 en sí. Esto significa que la salida del algoritmo 3-7 es una solución factible del problema del mínimo de guardias cooperativos. Debido a que el número de guardias es mínimo, se concluye que el algoritmo 3-7 produce una solución óptima.

Así como se hizo para el análisis de la complejidad temporal del algoritmo 3-7, sea n el número de vértices de un polígono de 1-espiral. Los pasos 1 y 2 pueden realizarse en tiempo $O(n)$ mediante una exploración de la frontera de P. Para el paso 3 pueden efectuarse una exploración lineal en la cadena reflex en sentido contrario al movimiento de las manecillas del reloj y una exploración lineal en la cadena convexa en el sentido del movimiento de las manecillas del reloj para encontrar todos los segmentos de recta de apoyo izquierdos. El paso 3 también puede llevarse a cabo en tiempo $O(n)$. Así, el algoritmo 3-7 es lineal.

3-8 LOS RESULTADOS EXPERIMENTALES

Para mostrar la potencia de la estrategia del método codicioso, el algoritmo de Prim para árboles generadores mínimos se implementó en una PC IBM con lenguaje C. También se implementó un método directo que generaría todos los conjuntos posibles de $(n - 1)$ aristas de una gráfica $G = (V, E)$. Si estas aristas constituyen un ciclo, se ignoran; en caso contrario, se calcula la longitud total de este árbol de expansión.

El árbol de expansión mínima se encuentra luego que se han analizado todos los árboles generadores. Este método directo también fue implementado en lenguaje C. Cada conjunto de datos corresponde a una gráfica $G = (V, E)$ generada de manera aleatoria. En la tabla 3-3 se resumen los resultados experimentales. Resulta evidente que el problema del árbol de expansión mínima no puede resolverse con el método directo y que el algoritmo de Prim es bastante eficaz.

TABLA 3-3 Resultados experimentales al probar la eficacia del algoritmo de Prim.

Tiempo de ejecución (segundos) (Promedio de 20 veces)							
$\|V\|$	$\|E\|$	Directo	Prim	$\|V\|$	$\|E\|$	Directo	Prim
10	10	0.305	0.014	50	200	–	1.209
10	20	11.464	0.016	50	250	–	1.264
10	30	17.629	0.022	60	120	–	1.648
15	30	9738.883	0.041	60	180	–	1.868
20	40	–	0.077	60	240	–	1.978
20	60	–	0.101	60	300	–	2.033
20	80	–	0.113	70	140	–	2.527
20	100	–	0.118	70	210	–	2.857
30	60	–	0.250	70	280	–	2.967
30	90	–	0.275	70	350	–	3.132
30	120	–	0.319	80	160	–	3.791
30	150	–	0.352	80	240	–	4.176
40	80	–	0.541	80	320	–	4.341
40	120	–	0.607	80	400	–	4.560
40	160	–	0.646	90	180	–	5.275
40	200	–	0.698	90	270	–	5.714
50	100	–	1.030	90	360	–	6.154
50	150	–	1.099	90	450	–	6.264

3-9 NOTAS Y REFERENCIAS

Para más análisis sobre el método codicioso, consulte Horowitz y Sahni (1978) y Papadimitriou y Steiglitz (1982). Korte y Louasz (1984) introdujeron un marco de referencia estructural para el método codicioso.

El algoritmo de Kruskal y el algoritmo de Prim para árboles de expansión pueden consultarse en Kruskal (1956) y Prim (1957), respectivamente. El algoritmo de Dijkstra apareció originalmente en Dijkstra (1959).

El algoritmo codicioso que genera árboles de mezcla óptimos apareció primero en Huffman (1952). Schwartz (1964) proporcionó un algoritmo para producir el conjunto de código Huffman. El algoritmo para encontrar el ciclo base mínimo se debe a Horton (1987).

El método codicioso para resolver el problema de 2-terminales de una a muchas asignaciones de canales fue propuesto por Atallah y Hambrusch (1986). El algoritmo lineal para resolver el problema del mínimo de guardias cooperativos para un polígono de 1-espiral fue proporcionado por Liaw y Lee (1994).

3-10 BIBLIOGRAFÍA ADICIONAL

Aunque el concepto de método codicioso fue descubierto hace mucho tiempo, aún siguen publicándose muchos artículos interesantes para analizarlo. Los siguientes artículos se recomiendan ampliamente para el lector interesado en profundizar la investigación del método codicioso: Bein y Brucker (1986); Bein, Brucker y Tamir (1985); Blot, Fernandez de la Vega, Paschos y Saad (1995); Coffman, Langston y Langston (1984); Cunningham (1985); El-Zahar y Rival (1985); Faigle (1985); Fernandez-Baca y Williams (1991); Frieze, McDiarmid y Reed (1990); Hoffman (1988), y Rival y Zaguia (1987).

El método codicioso, por supuesto, puede usarse para aproximar algoritmos (capítulo 9 de este libro). Los siguientes artículos constituyen buen material de consulta: Gonzalez y Lee (1987); Levcopoulos y Lingas (1987), y Tarhio y Ukkonen (1988).

Para conocer algunos artículos bastante interesantes de reciente aparición, consulte Akutsu, Miyano y Kuhara (2003); Ando, Fujishige y Naitoh (1995); Bekesi, Galambos, Pferschy y Woeginger (1997); Bhagavathi, Grosch y Olariu (1994); Cidon, Kutten, Mansour y Peleg (1995); Coffman, Langston y Langston (1984); Cowureur y Bresler (2000); Csur y Kao (2001); Erlebach y Jansen (1999); Gorodkin, Lyngso y Stormo (2001); Gudmundsson, Levcopoulos y Narasimhan (2002); Hashimoto y Barrera (2003); Iwamura (1993); Jorma y Ukkonen (1988); Krogh, Brown, Mian, Sjolander y Haussler (1994); Maggs y Sitaraman (1999); Petr (1996); Slavik (1997); Tarhio y Ukkonen (1986); Tarhio y Ukkonen (1988); Tomasz (1998); Tsai, Tang y Chen (1994); Tsai, Tang y Chen (1996), y Zhang, Schwartz, Wagner y Miller (2003).