9.3 ➜ Árboles de expansión

En esta sección se considera el problema de encontrar una subgráfica T de una gráfica G tal que T es un árbol que contiene todos los vértices de G. Este árbol se conoce como **árbol de expansión**. Se verá que los métodos para encontrar los árboles de expansión también son aplicables y se aplican a otros problemas.

WWW

Definición 9.3.1 ▶ Un árbol T es un *árbol de expansión* de una gráfica G si T es una subgráfica de G que contiene todos los vértices de G. ◀

Ejemplo 9.3.2 ▶ Un árbol de expansión de una gráfica G de la figura 9.3.1 se muestra con línea continua. ◀

Ejemplo 9.3.3 ▶ En general, una gráfica tiene varios árboles de expansión. Otro árbol de expansión de la gráfica G de la figura 9.3.1 se aprecia en la figura 9.3.2.

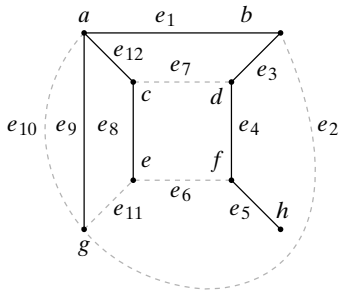


Figura 9.3.1 Una gráfica y un árbol de expansión mostrado por la línea continua.

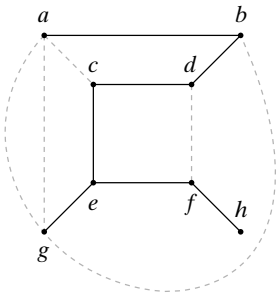


Figura 9.3.2 Otro árbol de expansión (línea continua) de la gráfica de la figura 9.3.1.

◀

Suponga que una gráfica G tiene un árbol de expansión T. Sean a y b vértices de G. Como a y b también son vértices de T y T es un árbol, existe una trayectoria P de a a b. Sin embargo, P también sirve como trayectoria de a a b en G; entonces G es conexa. El inverso también es cierto.

Teorema 9.3.4 *Una gráfica G tiene un árbol de expansión si y sólo si G es conexa.*

Demostración Ya se demostró que si G tiene un árbol de expansión, entonces G es conexa. Suponga que G es conexa. Si G es acíclica, por el Teorema 9.2.3, G es un árbol.

Suponga que G contiene un ciclo. Se elimina una arista (no vértices) de este ciclo. La gráfica producida todavía es conexa. Si es acíclica, nos detenemos. Si contiene un ciclo, se elimina una arista de este ciclo, y así sucesivamente, hasta producir una subgráfica T conexa y acíclica. Por el Teorema 9.2.3, T es un árbol. Como T contiene todos los vértices de G, T es un árbol de expansión de G.

Un algoritmo para encontrar un árbol de expansión basado en la demostración del Teorema 9.3.4 no sería muy eficiente; incluiría el tardado proceso de encontrar ciclos. Es posible hacer algo mejor. Se ilustrará el primer algoritmo para encontrar un árbol de expansión mediante un ejemplo y luego se establecerá el algoritmo.

Ejemplo 9.3.5 ▶ Encuentre el árbol de expansión para la gráfica G de la figura 9.3.1.

Se usará un método llamado **búsqueda a lo ancho** (algoritmo 9.3.6). La idea de la búsqueda a lo ancho es procesar todos los vértices en un nivel dado antes de moverse al nivel más alto que sigue.

WWW

Primero se selecciona un orden, por ejemplo *abcdefgh*, de los vértices de G. Se elige el primer vértice a y se etiqueta como raíz. T consiste en un sólo vértice a y ninguna

arista. Se agregan a T todas las aristas (a, x) y vértices en los cuales inciden, para $x = b$ hasta h, que no produzcan ciclos cuando se agregan a T. Entonces se agregan a T las aristas (a, b), (a, c) y (a, g). (Se puede usar cualquiera de las aristas paralelas que inciden en a y g). Se repite este proceso con los vértices del nivel 1 examinando cada uno en orden:

b: incluir (b, d).
c: incluir (c, e).
g: ninguna

Se repite el proceso con los vértices en el nivel 2:

d: incluir (d, f).
e: ninguna

Se repite el proceso con los vértices del nivel 3:

f: incluir (f, h).

Como no se pueden agregar más aristas al vértice h en el nivel 4, el proceso termina. Se ha encontrado el árbol de expansión de la figura 9.3.1. ◀

El método del ejemplo 9.3.5 se formaliza como el algoritmo 9.3.6.

Algoritmo 9.3.6

Búsqueda a lo ancho de un árbol de expansión

Este algoritmo encuentra un árbol de expansión usando el método de búsqueda a lo ancho.

Entrada: Gráfica conexa G con vértices ordenados

$$v_1, \quad v_2, \quad \ldots, \quad v_n$$

Salida: Árbol de expansión T.

```
bla(V, E) {
    //V = vértices ordenados v1, . . . , vn; E = aristas
    //V' = vértices del árbol de expansión T; E' = aristas del árbol de expansión T
    //v1 es la raíz del árbol de expansión
    //S es una lista ordenada
    S = (v1)
    V' = {v1}
    E' = ∅
    while(verdadero) {
        for x ∈ S, en orden
            for y ∈ V − V', en orden
                if((x, y) es una arista)
                    agregar arista (x, y) a E' y y a V'
        if (no se agregaron aristas)
            return T
        S = hijos de S ordenados siguiendo el orden original
    }
}
```

El ejercicio 16 pide un argumento para demostrar que el algoritmo 9.3.6 encuentra el árbol de expansión correctamente.

La búsqueda a lo ancho se puede usar para probar si una gráfica arbitraria G con n vértices es conexa (vea el ejercicio 26). Se usa el método del algoritmo 9.3.6 para producir un árbol T. Después G es conexa si y sólo si T tiene n vértices.

La búsqueda a lo ancho también resulta útil para encontrar trayectorias de longitud mínima en una gráfica ponderada desde un vértice fijo v a todos los demás vértices (vea el ejercicio 20). Se emplea el método del algoritmo 9.3.6 para generar un árbol de expansión con raíz en v. Se observa que la longitud de una ruta más corta de v al vértice en el nivel i del árbol de expansión es i. El algoritmo de la ruta más corta de Dijkstra para gráficas ponderadas (algoritmo 8.4.1) se puede considerar como una generalización de la búsqueda a lo ancho (vea el ejercicio 21).

WWW

Una alternativa para la búsqueda a lo ancho es la **búsqueda de altura**, que procede a los niveles sucesivos en un árbol en la oportunidad más cercana posible.

Algoritmo 9.3.7

Búsqueda a profundidad de un árbol de expansión

Este algoritmo encuentra un árbol de expansión usando el método de búsqueda a profundidad.

Entrada: Gráfica conexa G con vértices ordenados

$$v_1, \quad v_2, \quad \ldots, \quad v_n$$

Salida: Árbol de expansión T

```
bll(V, E) {
   //V' = vértices del árbol de expansión T; E' = aristas del árbol de expansión T
   //v1 es la raíz del árbol de expansión
   V' = {v1}
   E' = ∅
   w = v1
   while(verdadero) {
      while(hay arista (w, v) que al agregarla a T no crea
         un ciclo en T) {
         elegir la arista (w, vk) con k mínima, que si se
            agrega a T no crea un ciclo en T
         agregar(w, vk) a E'
         agregar vk a V'
         w = vk
      }
      if (w == v1)
         return T
      w = padre de w en T //hacia atrás
   }
}
```

El ejercicio 17 pide un argumento para probar que el algoritmo 9.3.7 encuentra un árbol de expansión correctamente.

Ejemplo 9.3.8 ▶

Use la búsqueda a profundidad (algoritmo 9.3.7) para encontrar un árbol de expansión para la gráfica de la figura 9.3.2 con el orden de vértices *abcdefgh*.

Se selecciona el primer vértice a y se le llama raíz (figura 9.3.2). Después se agrega al árbol la arista (a, x), con x mínima. En este caso se agrega la arista (a, b).

Se repite este proceso. Se agregan las aristas (b, d), (d, c), (c, e), (e, f) y (f, h). En este punto, no se puede agregar una arista de la forma (h, x), de manera que regresamos a f, el padre de h, y tratamos de agregar una arista de la forma (f, x). Una vez más, no es posible agregar una arista de la forma (f, x), así que regresamos a e, el padre de f. Esta vez se puede agregar la arista (e, g). En este punto, no hay más aristas que agregar, de manera que regresamos a la raíz y el procedimiento termina. ◀

En virtud de la línea en el algoritmo 9.3.7 donde se regresa por una arista hacia la raíz elegida el inicio, la búsqueda a profundidad también se llama **de regreso**. En el siguiente ejemplo, se usa el regreso para resolver un juego.

Ejemplo 9.3.9 ▶

Problema de las cuatro reinas

El problema de las cuatro reinas trata de colocar cuatro fichas en una cuadrícula de 4×4 de manera que no haya dos fichas en la misma fila, columna o diagonal. Construya un algoritmo de regreso para resolver el problema de las cuatro reinas. (Para usar la terminología del ajedrez, éste es el problema de colocar cuatro reinas en un tablero de 4×4 de manera que ninguna reina pueda atacar a otra).

WWW

La idea del algoritmo es colocar fichas sucesivamente en las columnas. Cuando es imposible colocar una ficha en una columna, se regresa y ajusta la ficha de la columna anterior. ◀

Algoritmo 9.3.10

Solución al problema de las cuatro reinas usando el regreso

Este algoritmo usa regresos para buscar un arreglo de cuatro fichas en una cuadrícula de 4×4 de manera que no haya dos fichas en la misma fila, columna o diagonal.

```
Entrada:  Un arreglo fila de tamaño 4
Salida:   verdadero, si hay solución
          Falso, si no hay solución
          [Si hay solución, la reina k está en la
          columna k y el renglón fila(k)].
cuatro_reinas(fila) {
   k = 1 //inicia en la columna 1

      //inicia en el renglón 1
   //como fila(k) se incrementa antes de usarla, se hace fila(1) igual a 0
   fila(1) = 0
   while (k > 0) {
      fila(k) = fila(k) + 1
      //se busca un movimiento legal en la columna k
      while (fila(k) ≤ 4 ∧ columna k, fila(k) conflicto)
         //se intenta el siguiente renglón
         fila(k) = fila(k) + 1
      if (fila(k) ≤ 4)
         if(k == 4)
            return verdadero
         else{//siguiente columna
            k = k + 1
            fila(k) = 0
         }
      else //regresar a columna anterior
         k = k − 1
   }
   return falso //no hay solución
}
```

El árbol que genera el algoritmo 9.3.10 se ilustra en la figura 9.3.3. La numeración indica el orden en que se generaron los vértices. La solución se encuentra en el vértice 8.

El problema de las n reinas es colocar las n fichas en una cuadrícula de $n \times n$ de manera que no haya dos fichas en la misma fila, columna o diagonal. Es directo verificar que no hay solución a los problemas de dos o tres reinas (vea el ejercicio 10). Se acaba de ver que el algoritmo 9.3.10 genera una solución para el problema de las cuatro reinas. Se han dado muchos desarrollos para generar una solución al problema de las n reinas para toda $n \geq 4$ (vea, por ejemplo, [Johnsonbaugh]).

La búsqueda a lo largo o de regreso es atractiva, en especial en un problema como el del ejemplo 9.3.9, donde todo lo que se quiere es una solución. Como una solución, si acaso existe, se encuentra en un vértice terminal, moviéndose a los vértices terminales tan pronto como sea posible, en general no se puede evitar generar algunos vértices innecesarios.

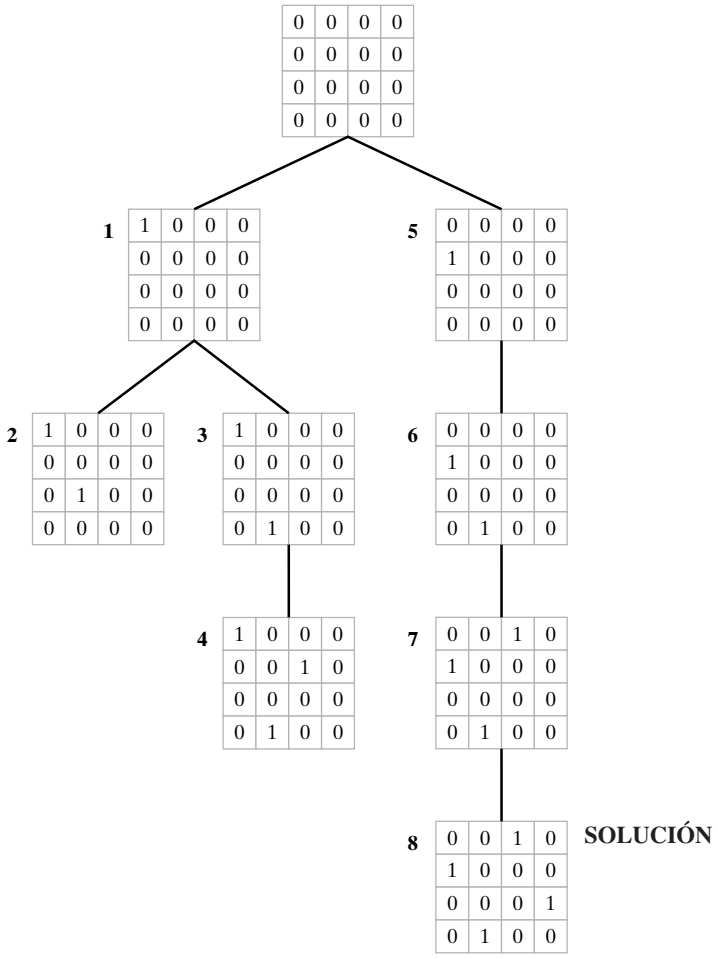


Figura 9.3.3 Árbol generado por el algoritmo de regreso (algoritmo 9.3.10) en la búsqueda de una solución al problema de las cuatro reinas.

Sugerencias para resolver problemas

La búsqueda a profundidad y la búsqueda a lo ancho son la base de muchos algoritmos de gráficas. Por ejemplo, cualquiera de los dos resulta útil para determinar si una gráfica es conexa: Si se puede visitar *cada* vértice en una gráfica a partir de un vértice inicial, la gráfica es conexa; de otra manera no es conexa (vea los ejercicios 26 y 27). La búsqueda a profundidad se puede emplear como algoritmo de búsqueda, en cuyo caso se llama de regreso. En el algoritmo 9.3.10, el regreso se usa para buscar una solución al problema de las 4 reinas; también se puede utilizar para buscar ciclos de Hamilton en una gráfica, para generar permutaciones y para determinar si dos gráficas son isomorfas (vea [Johnsonbaugh]).

Sección de ejercicios de repaso

1. ¿Qué es un árbol de expansión?

2. Establezca una condición necesaria y suficiente para que una gráfica tenga un árbol de expansión.

3. Explique cómo funciona la búsqueda a lo ancho.

4. Explique cómo funciona la búsqueda a profundidad.

5. ¿Qué significa ir de regreso?

Ejercicios

1. Use la búsqueda a lo ancho (algoritmo 9.3.6) con el orden de vértices *hgfedcba* para encontrar un árbol de expansión para la gráfica G de la figura 9.3.1.

2. Use la búsqueda a lo ancho (algoritmo 9.3.6) con el orden de vértices *hfdbgeca* para encontrar un árbol de expansión para la gráfica G de la figura 9.3.1.

3. Use la búsqueda a lo ancho (algoritmo 9.3.6) con el orden de vértices *chbgadfe* para encontrar un árbol de expansión para la gráfica G de la figura 9.3.1.

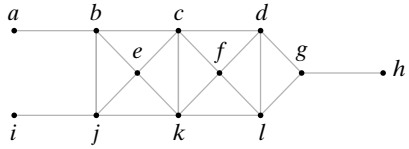
4. Use la búsqueda a profundidad (algoritmo 9.3.7) con el orden de vértices *hgfedcba* para encontrar un árbol de expansión para la gráfica G de la figura 9.3.1.

5. Use la búsqueda a profundidad (algoritmo 9.3.7) con el orden de vértices *hfdbgeca* para encontrar un árbol de expansión para la gráfica G de la figura 9.3.1.

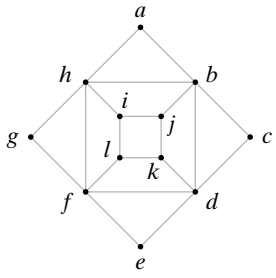
6. Use la búsqueda a profundidad (algoritmo 9.3.7) con el orden de vértices *dhcbefag* para encontrar un árbol de expansión para la gráfica G de la figura 9.3.1.

En los ejercicios 7 al 9, encuentre un árbol de expansión para cada gráfica.

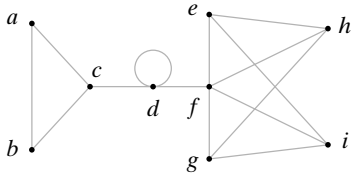
7.



8.



9.



10. Demuestre que no hay solución a los problemas de dos y tres reinas.

11. Encuentre una solución a los problemas de cinco y seis reinas.

12. ¿Falso o verdadero? Si G es una gráfica conexa y T es un árbol de expansión para G, existe un orden de los vértices de G tal que el algoritmo 9.3.6 produce T como una árbol de expansión. Si es verdadero, pruébelo; de otra manera, proporcione un contraejemplo.

13. ¿Falso o verdadero? Si G es una gráfica conexa y T es un árbol de expansión para G, existe un orden de los vértices de G tal que el algoritmo 9.3.7 produce T como un árbol de expansión. Si es verdadero, pruébelo; de otra manera, proporcione un contraejemplo.

14. Muestre, mediante un ejemplo, que el algoritmo 9.3.6 puede producir árboles de expansión idénticos para una gráfica conexa G a partir de dos ordenamientos de los vértices de G.

15. Muestre, mediante un ejemplo, que el algoritmo 9.3.7 puede producir árboles de expansión idénticos para una gráfica conexa G a partir de dos ordenamientos de los vértices de G.

16. Pruebe que el algoritmo 9.3.6 es correcto.

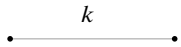
17. Pruebe que el algoritmo 9.3.7 es correcto.

18. ¿En qué condiciones una arista en una gráfica conexa G está contenida en todos los árboles de expansión de G?

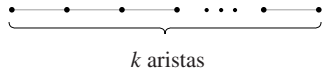
19. Sean T y T' dos árboles de expansión de una gráfica conexa G. Suponga que una arista x está en T pero no en T'. Demuestre que existe una arista y en T' pero no en T tal que $(T - \{x\}) \cup \{y\}$ y $(T' - \{y\}) \cup \{x\}$ son árboles de expansión de G.

20. Escriba un algoritmo basado en la búsqueda a lo ancho que encuentre la longitud mínima de cada trayectoria en una gráfica no ponderada de un vértice fijo v a todos los demás.

21. Sea G una gráfica ponderada en la que el peso de cada arista es un entero positivo. Sea G' la gráfica obtenida de G al sustituir cada arista



en G con peso k por k aristas no ponderadas en serie:



Demuestre que el algoritmo de Dijkstra para encontrar la longitud mínima de cada trayectoria en la gráfica ponderada G desde un vértice fijo v a todos lo demás (algoritmo 8.4.1) y realizar la búsqueda a lo ancho en una gráfica no ponderada G' comenzando en el vértice v son, de hecho, el mismo proceso.

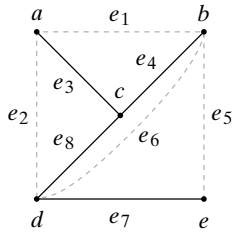
22. Sea T un árbol de expansión para una gráfica G. Demuestre que si una arista en G, pero no en T, se agrega a T, se produce un ciclo único.

Un ciclo como el descrito en el ejercicio 22 se llama ciclo fundamental. La matriz del ciclo fundamental de una gráfica G tiene renglones indexados según los ciclos fundamentales de G respecto a un árbol de expansión T para G y sus columnas indexadas según las aristas de G. El elemento ij es 1 si la arista j está en el i-ésimo ciclo fundamental y 0 de otra manera. Por ejemplo, la matriz del ciclo fundamental de la gráfica G de la figura 9.3.1 relativa al árbol de expansión mostrado en la figura 9.3.1 es

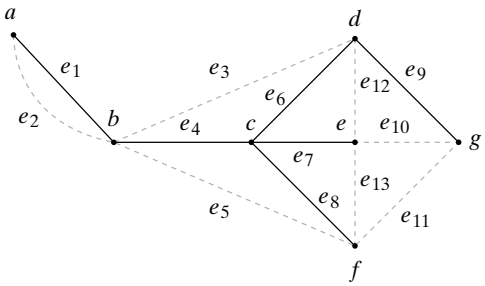
$$\begin{array}{c} \\ (abdca) \\ (efdbace) \\ (ageca) \\ (aga) \\ (abga) \end{array} \begin{array}{c} \begin{array}{cccccccccccc} e_7 & e_6 & e_{11} & e_{10} & e_2 & e_1 & e_3 & e_4 & e_5 & e_8 & e_9 & e_{12} \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array}.$$

Encuentre la matriz del ciclo fundamental de cada gráfica. El árbol de expansión que se usará está dibujado con línea continua.

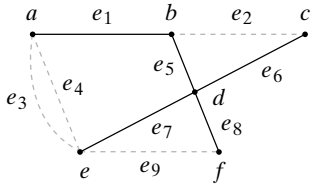
23.



24.



25.



26. Escriba un algoritmo de búsqueda a profundidad para probar si una gráfica es conexa.

27. Escriba un algoritmo de búsqueda a profundidad para probar si una gráfica es conexa.

28. Escriba un algoritmo de búsqueda a lo largo para encontrar todas las soluciones al problema de las cuatro reinas.

9.4 ➜ Árboles de expansión mínima

WWW

La gráfica ponderada G de la figura 9.4.1 muestra seis ciudades y los costos de construir carreteras entre ciertos pares de ciudades. Se desea construir el sistema de carreteras de menor costo que conecte las seis ciudades. La solución habrá de representarse en una subgráfica. Esta subgráfica debe ser un árbol de expansión ya que debe contener a todos los vértices (de manera que todas las ciudades queden comprendidas en el sistema de carreteras), debe ser conexa (para que se pueda llegar a cualquier ciudad desde cualquier otra) y debe tener una trayectoria simple única entre cada par de vértices (puesto que una gráfica que contiene trayectorias simples múltiples entre un par de vértices tal vez no represente un sistema de costo mínimo). Entonces, lo que se necesita es un árbol de expansión en el que la suma de los pesos sea mínima. Este árbol se llama **árbol de expansión mínima**.

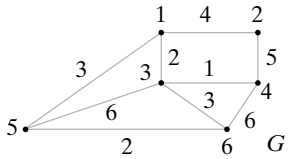


Figura 9.4.1 Seis ciudades (numeradas del 1 al 6) y los costos de construir carreteras entre ciertos pares de ellas.

Definición 9.4.1 ▶ Sea G una gráfica ponderada. Un *árbol de expansión mínima* de G es un árbol de expansión de G con peso mínimo. ◀

Ejemplo 9.4.2 ▶ El árbol T' mostrado en la figura 9.4.2 es un árbol de expansión para la gráfica G de la figura 9.4.1. El peso de T' es 20. Este árbol no es un árbol de expansión mínima, ya que el árbol de expansión T mostrado en la figura 9.4.3 tiene peso 12. Se verá más adelante que T es un árbol de expansión mínima para G.

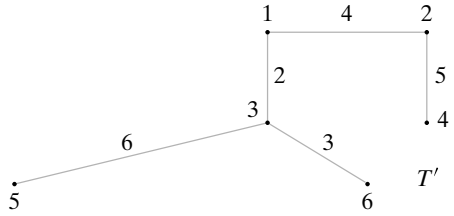


Figura 9.4.2 Árbol de expansión con peso 20 de la gráfica de la figura 9.4.1.

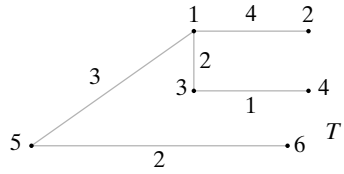


Figura 9.4.3 Árbol de expansión con peso 12 de la gráfica de la figura 9.4.1.

◀

El algoritmo para encontrar un árbol de expansión mínimo que se estudiará se conoce como **algoritmo de Prim** (algoritmo 9.4.3). Este algoritmo construye un árbol agregando aristas de manera iterativa hasta que se obtiene un árbol de expansión mínima. El algoritmo comienza con un solo vértice. Después, en cada iteración, agrega al árbol actual una arista de peso mínimo que no completa un ciclo. Otro algoritmo para encontrar el árbol de expansión mínima, conocido como **algoritmo de Kruskal**, se presenta en los ejercicios 20 al 22.

WWW

Algoritmo 9.4.3

Algoritmo de Prim

Este algoritmo encuentra un árbol de expansión mínima en una gráfica ponderada conexa.

Entrada: Gráfica ponderada conexa con vértices $1, \ldots, n$ y vértice inicial s. Si (i, j) es una arista, $w(i, j)$ es igual al peso de (i, j); si (i, j) no es una arista, $w(i, j)$ es igual a ∞ (un valor mayor que cualquier peso real).

Salida: Conjunto de aristas de E en un árbol de expansión mínima (aem)

```
    prim(w, n, s)  {
        //v(i) = 1 si el vértice i se agrega al aem
        //v(i) = 0 si el vértice i no se agrega al aem
1.      for i = 1 to n
2.          v(i) = 0
        //se agrega el vértice a; aem
3.      v(s) = 1
        //se comienza con un conjunto de aristas vacío
4.      E = ∅
        //se ponen n − 1 aristas en el aem
5.      for i = 1 to n − 1 {
            //se agrega la arista de peso mínimo con un vértice
            //en el aem y un vértice que no esté en el aem
6.          mín = ∞
7.          for j = 1 to n
8.              if (v(j) == 1)//si j es un vértice en el aem
9.                  for k = 1 to n
10.                     if(v(k) == 0 ∧ w(j, k) < mín) {
11.                         agregar_vértice = k
12.                         e = (j, k)
13.                         mín = w(j, k)
14.                     }
            //se pone el vértice y la arista en el aem
15.         v(agregar_vértice) = 1
16.         E = E ∪ {e}
17.     }
18.     return E
19. }
```

Ejemplo 9.4.4 ▶

Demuestre que el algoritmo de Prim encuentra un árbol de expansión mínima para la gráfica de la figura 9.4.1. Suponga que el vértice inicial s es 1.

En la línea 3 se agrega el vértice 1 al árbol de expansión mínima. La primera vez que se ejecuta el ciclo "for" en las líneas 7 a la 14, las aristas con un vértice en el árbol y un vértice fuera del árbol son

Arista	*Peso*
(1, 2)	4
(1, 3)	2
(1, 5)	3

Se selecciona la arista (1, 3) con peso mínimo. En las líneas 15 y 16 se agrega el vértice al árbol de expansión mínima y la arista (1, 3) a *E*.

La siguiente vez que se ejecuta el ciclo "for" en las líneas 7 a la 14, las aristas con un vértice en el árbol y otro fuera del árbol son

Arista	*Peso*
(1, 2)	4
(1, 5)	3
(3, 4)	1
(3, 5)	6
(3, 6)	3

Se selecciona la arista (3, 4) con peso mínimo. En las líneas 15 y 16 se agrega el vértice 4 al árbol de expansión mínima y la arista (3, 4) a *E*.

La siguiente vez que se ejecuta el ciclo "for" en las líneas 7 a la 14, las aristas con un vértice en el árbol y otro fuera del árbol son

Arista	*Peso*
(1, 2)	4
(1, 5)	3
(2, 4)	5
(3, 5)	6
(3, 6)	3
(4, 6)	6

Esta vez, dos aristas tienen peso mínimo de 3. Se construye un árbol de expansión mínima cuando se elige cualquiera de las dos aristas. En esta versión se seleccionó (1, 5). En las líneas 15 y 16 se agrega el vértice 5 al árbol de expansión mínima y la arista (1, 5) a *E*.

La siguiente vez que se ejecuta el ciclo "for" en las líneas 7 a la 14, las aristas con un vértice en el árbol y otro fuera son

Arista	*Peso*
(1, 2)	4
(2, 4)	5
(3, 6)	3
(4, 6)	6
(5, 6)	2

Se selecciona la arista (5, 6) con peso mínimo. En las líneas 15 y 16, se agrega el vértice 6 al árbol de expansión mínima y la arista (5, 6) se agrega a *E*.

La última vez que se ejecuta el ciclo "for" en las líneas 7 a la 14, las aristas con un vértice en el árbol y un vértice fuera son

Arista	*Peso*
(1, 2)	4
(2, 4)	5

Se selecciona la arista (1,2) con peso mínimo. En las líneas 15 y 16 se agrega el vértice 2 al árbol de expansión mínima y la arista (1,2) a *E*. El árbol de expansión mínima construido se muestra en la figura 9.4.3. ◀

El algoritmo de Prim proporciona un ejemplo de un **algoritmo ambicioso**. Un algoritmo ambicioso es uno que optimiza la elección en cada iteración. El principio se resume como "hacer lo mejor localmente". En el algoritmo de Prim, como se desea un árbol de ex-

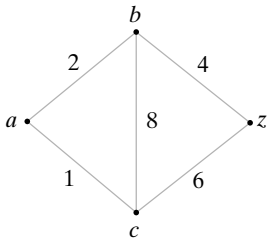


Figura 9.4.4 Gráfica que demuestra que seleccionar una arista con peso mínimo incidente en el último vértice que se agregó *no* necesariamente lleva a la ruta más corta. Comenzando en a, se obtiene (a, c, z), pero la ruta más corta de a a z es (a, b, z).

pansión mínima, en cada iteración simplemente se agrega una arista disponible con peso mínimo.

Optimizar en cada iteración no necesariamente ofrece una solución óptima al problema original. A continuación se mostrará (Teorema 9.4.5) que el algoritmo de Prim es correcto: sí se obtiene un árbol de expansión mínima. Como ejemplo de un algoritmo ambicioso que no lleva a una solución óptima, considere un "algoritmo de la ruta más corta" en el que en cada paso se selecciona una arista disponible con peso mínimo, incidente en el vértice que se agregó más recientemente. Si se aplica este algoritmo a la gráfica ponderada de la figura 9.4.4 para encontrar una ruta más corta de a a z, se seleccionaría la arista (a, c) y después la arista (c, z). Desafortunadamente, ésta no es la ruta más corta de a a z.

A continuación se prueba que el algoritmo de Prim es correcto.

Teorema 9.4.5

El algoritmo de Prim (algoritmo 9.4.3) es correcto; es decir, al terminar el algoritmo 9.4.3, T es un árbol de expansión mínima.

Demostración Sea T_i la gráfica construida por el algoritmo 9.4.3 después de la i-ésima iteración del ciclo "for", líneas 5 a la 17. De manera más precisa, el conjunto de aristas de T_i es el conjunto E construido después de la i-ésima iteración del ciclo "for", líneas 5 a la 17, y el conjunto de vértices de T_i es el conjunto de vértices sobre los cuales inciden las aristas en E. Sea T_0 una gráfica construida por el algoritmo 9.4.3 justo antes de entrar por primera vez al ciclo "for" en la línea 5; T_0 consiste en el único vértice s y ninguna arista. Más adelante en esta prueba, se suprime el conjunto de vértices y se hace referencia a una gráfica especificando su conjunto de aristas.

Por construcción, al terminar el algoritmo 9.4.3, la gráfica que se obtiene, T_{n-1}, es una subgráfica conexa acíclica de la gráfica dada G que contiene todos los vértices de G; así, T_{n-1} es un árbol de expansión de G.

Se usa inducción para demostrar que para toda $i = 0, \ldots, n-1$, T_i está contenida en un árbol de mínima expansión. Entonces, al terminar se deduce que T_{n-1} es un árbol de mínima expansión.

Si $i = 0$, T_0 consiste en un solo vértice. En este caso, T_0 está contenido en todo árbol de expansión mínima. Esto verifica el paso base.

Ahora, suponga que T_i está contenido en un árbol de expansión mínima T'. Sea V el conjunto de vértices en T_i. El algoritmo 9.4.3 selecciona una arista (j, k) de peso mínimo, donde $j \in V$ y $k \notin V$, y la agrega a T_i para producir T_{i+1}. Si (j, k) está en T', entonces T_{i+1} está contenida en el árbol de expansión mínima T'. Si (j, k) no está en T', $T' \cup \{(j, k)\}$ contiene un ciclo C. Elija una arista (x, y) en C, diferente de (j, k), con $x \in V$ y $y \notin V$. Entonces

$$w(x, y) \geq w(j, k). \tag{9.4.1}$$

En virtud de (9.4.1), la gráfica $T'' = [T' \cup \{(j, k)\}] - \{(x, y)\}$ tiene peso menor o igual que el peso de T'. Como T'' es un árbol de expansión, T'' es un árbol de expansión mínima. Como T_{i+1} está contenido en T'', se verifica el paso inductivo. La prueba queda completa.

Nuestra versión del algoritmo de Prim examina $\Theta(n^3)$ aristas en el peor caso (vea el ejercicio 6) para encontrar un árbol de mínima expansión para una gráfica que tiene n vértices. Es posible (vea el ejercicio 8) implementar el algoritmo de Prim de manera que sólo se examinen $\Theta(n^2)$ aristas en el peor caso. Como K_n tiene $\Theta(n^2)$ aristas, esta última versión es óptima.

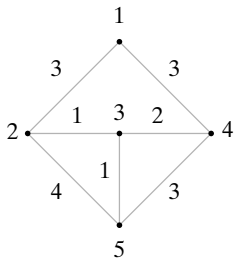
Sección de ejercicios de repaso

1. ¿Qué es un árbol de expansión mínima?
2. Explique cómo encuentra el algoritmo de Prim un árbol de expansión mínima.
3. ¿Qué es un algoritmo ambicioso?

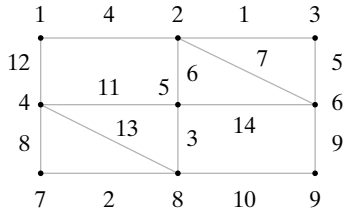
Ejercicios

En los ejercicios 1 al 5, encuentre el árbol de expansión mínima dado por el algoritmo 9.4.3 para cada gráfica.

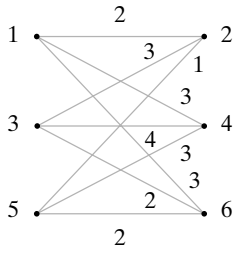
1.



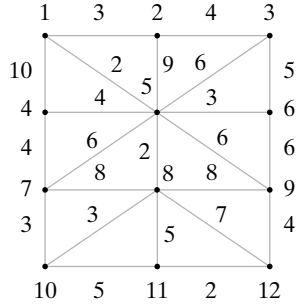
2.



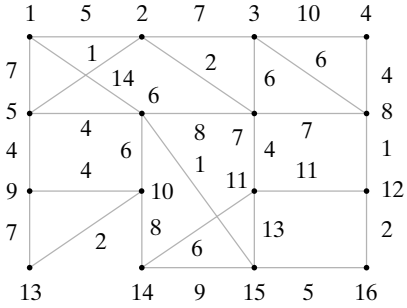
3.



4.



5.



6. Demuestre que el algoritmo 9.4.3 examina $\Theta(n^3)$ aristas en el peor caso.

Los ejercicios 7 al 9 se refieren a la versión alternativa del algoritmo de Prim (algoritmo 9.4.6).

Algoritmo 9.4.6

Versión alternativa del algoritmo de Prim

Este algoritmo encuentra un árbol de expansión mínima en un gráfica G conexa ponderada. En cada paso, algunos vértices tienen etiquetas temporales y otros tienen etiquetas permanentes. La etiqueta de un vértice i se denota por L_i.

Entrada: Gráfica conexa ponderada con vértices $1, \ldots, n$ y vértice inicial s. Si (i, j) es una arista, $w(i, j)$ es igual al peso de (i, j); si (i, j) no es una arista, $w(i, j)$ es igual a ∞ (un valor mayor que cualquier peso real)

Salida: Árbol de expansión mínima T

```
prim_altern(w, n, s) {
    sea T la gráfica con vértice s y ninguna arista
    for j = 1 to n {
        L_j = w(s, j)//estas etiquetas son temporales
        regreso(j) = s
    }
    L_j = 0
    se hace L_s permanente
    while (queden etiquetas temporales) {
        se elige la etiqueta temporal más pequeña L_i
        L_i se hace permanente
        se agrega la arista (i, regreso(i)) a T
        por cada etiqueta temporal L_k
            if (w(i, k) < L_k) {
                L_k = w(i, k)
                regreso (k) = i
            }
        }
        return T
    }
```

7. Muestre la manera en que el algoritmo 9.4.6 encuentra un árbol de expansión mínima para las gráficas de los ejercicios 1 al 5.

8. Demuestre que el algoritmo 9.4.6 examina $\Theta(n^2)$ aristas en el peor caso.

9. Pruebe que el algoritmo 9.4.6 es correcto; es decir, al terminar el algoritmo, T es un árbol de expansión mínima.

10. Sea G una gráfica conexa ponderada, sea v un vértice en G y sea e la arista con peso mínimo incidente en v. Demuestre que e está contenida en algún árbol de expansión mínima.

11. Sea G una gráfica conexa ponderada y sea v un vértice en G. Suponga que los pesos de las aristas incidentes en v son distintos. Sea e la arista con peso mínimo incidente en v. ¿Debe e estar contenida en todo árbol de expansión mínima?

12. Demuestre que cualquier algoritmo que encuentra un árbol de expansión mínima en K_n, cuando todos los pesos son iguales, debe examinar toda arista en K_n.

13. Demuestre que si todos los pesos en una gráfica conexa G son diferentes, G tiene un árbol de expansión mínima.