



Universidad Nacional Autónoma de México



Facultad de Ingeniería

Estructuras Discretas

Grupo 06

Tarea 03

Profesor(a): Ing. Orlando Zaldívar Zamorategui

Alumno: David Romero Molina

Martes 14 de Marzo de 2023

Entrega: Martes 28 de Marzo de 2023

Tema 4.5.1: El algoritmo de Dijkstra.

Introducción

Muchos problemas se modelan de forma natural mediante grafos y dígrafos, y las estructuras de datos que implementan el grafo o el dígrafo ADT se utilizan habitualmente en informática. El tema de los algoritmos de grafos es un campo de investigación muy activo en la actualidad. Los grafos y dígrafos también desempeñan un papel importante en Internet y las redes de comunicación, transporte y flujo de mercancías, entre otras.

La estructura subyacente de estas redes se modela de forma natural utilizando un grafo o un dígrafo. Abundan los ejemplos de grafos de redes. El sistema de autopistas interestatales puede modelarse mediante un grafo en el que los nodos representan ciudades (o cruces) y las aristas, autopistas que unen las ciudades. La World Wide Web puede modelarse mediante un grafo dirigido, en el que los nodos corresponden a páginas web y existe una arista dirigida de la página web A a la página web B si la página web A incluye un hiperenlace

Los grafos de la informática paralela sirven de modelo para las redes de interconexión. Y podemos utilizar grafos para representar la estructura de superposición impuesta en Internet para una red entre iguales como Gnutella. Además de sus aplicaciones en redes, los grafos y dígrafos sirven como modelos naturales para otras muchas aplicaciones. Por citar un ejemplo de la informática

Por citar un ejemplo de la informática, el flujo lógico de un programa informático escrito en un lenguaje de alto nivel es naturalmente modelado por un dígrafo de programa (lowchart). Los compiladores de optimización utilizan varias propiedades de este dígrafo, como componentes fuertemente conectados y coloreado de vértices, para ayudar a lograr el objetivo de traducir el código de alto nivel en código máquina que muestre un rendimiento óptimo.

Sin embargo, dos problemas fundamentales para grafos ponderados con amplias aplicaciones son el problema del árbol mínimo y el problema del camino más corto. El problema de encontrar un árbol mínimo en un grafo ponderado es especialmente importante en aplicaciones de redes. Por ejemplo, el problema se plantea en el diseño de cualquier red física que conecte n nodos, donde las conexiones entre nodos están sujetas a restricciones de viabilidad y peso. Ejemplos de este tipo de redes físicas son las redes de comunicación, las redes de transporte, los conductos de energía y los chips VLSI, entre otros. En todos estos ejemplos, el peso de un árbol mínimo proporciona un límite inferior al coste de construcción de la red.

La búsqueda de los caminos más cortos en grafos y dígrafos ponderados también tiene innumerables aplicaciones en redes de diversos tipos. Por ejemplo, en una red informática o de comunicaciones, los datos se transfieren de forma más eficiente entre los nodos de la red a lo largo de un camino más corto entre los nodos, donde

los pesos de los nodos son los mismos y las aristas representan la latencia de la comunicación. Otro ejemplo es un dígrafo que representa una red de vuelos de avión. El vuelo menos costoso (en términos de dinero, distancia o tiempo) de un aeropuerto a otro en la red es el camino directo más corto desde el aeropuerto de salida al aeropuerto de llegada, donde los pesos de las aristas son los costes apropiados. En este capítulo se analizará un solo algoritmo para resolver el árbol del camino más corto, en este caso, el algoritmo de Dijkstra.

Árbol mínimo de expansión

Dado un árbol de expansión T en un grafo G con una ponderación w de las aristas E, el peso de T, denotado peso(T), es la suma de los pesos w de sus aristas. Si T tiene un peso mínimo sobre todos los árboles de expansión de G, entonces llamamos a T árbol de expansión mínima.

En la figura I2.1 se enumeran todos los árboles de expansión de un grafo ponderado de cuatro vértices. El árbol de expansión mínimo se obtiene por inspección.

El número de árboles de expansión, incluso para grafos relativamente pequeños, suele ser enorme, lo que hace inviable una búsqueda enumerativa por fuerza bruta. De hecho, Cayley demostró que el grafo completo K, con n vértices, contiene

$$n^2 \text{ árboles de expansión}$$

Afortunadamente, existen algoritmos eficientes basados en el método codicioso para encontrar árboles mínimos. En esta sección se tratará el problema de determinar la trayectoria más corta entre dos vértices cualesquiera en una gráfica ponderada. Se disponen de muchos algoritmos para determinar la trayectoria más corta en una gráfica ponderada. Se estudiará un solo algoritmo el cual fue descubierto por Edsger Dijkstra.

Algoritmo de Dijktra's

Supóngase que se está planteando un viaje en coche desde alguna ciudad hasta otra, por ejemplo, desde cdmx hasta puebla. Empleando un mapa de carreteras se encontrarán diferentes maneras de llegar a una respuesta. En este caso se podría utilizar el algoritmo de Floyd, que está basado en la representación matricial de un grafo. Dado que este problema implica un grafo ponderado en el cual los costes de las aristas son las distancias entre dos ciudades, las búsquedas en amplitud y la búsqueda en profundidad no son utilizables directamente, porque esas aproximaciones suponen que todos las aristas del grafo tienen un coste de 1.

Esta subsección examina un algoritmo, atribuido a Dijkstra, para hallar la longitud del camino de coste de minio entre dos vértices. Se supone que un grafo dado contiene n nodos v₁, v₂, ..., v_n y que es necesario hallar la longitud del camino de coste mínimo desde v₁ hasta v_n. Aun cuando se desea el camino más corto desde un nodo hasta otro, resulta útil (y eficiente) generar las longitudes de los caminos mínimos desde v₁ hasta todos los demás nodos del grafo. Se asocia un campo de distancia a cada uno de los nodos del grafo. Al comenzar, se selecciona el nodo

inicial v_1 y se da el valor cero a la distancia más corta desde ese nodo hasta sí mismo.

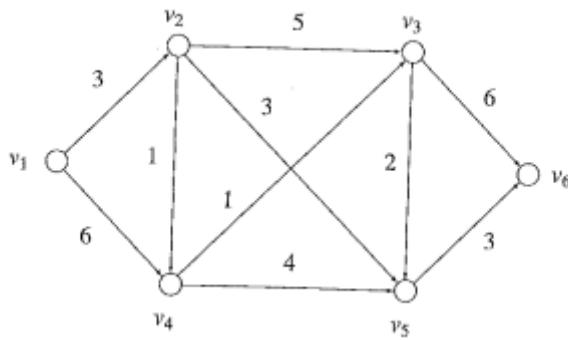
El enfoque consiste en hallar el nodo que se encuentra más próximo a v_n . A continuación se busca el nodo que esté inmediatamente más proximo a v_1 , y así sucesivamente. Eventualmente, el nodo v_n será el siguiente en distancias a v_1 , y en este momento se habrá hallado la longitud del camino mínimo. A medida que se van hallando nodos sucesivamente más alejados de v_1 , se van colocando en un conjunto S y se ignoran en el sentido de que se ha hallado su distancia mínima a v_1 . En cualquier fase dada, $\text{Dist}[v_1]$ es la distancia mínima de un camino que va desde v_1 hasta v_j para aquellos nodos que estén en S .

Como ejemplo de este enfoque, considérese el digrafo ponderado de la figura 7.43. Al hallar las distancias más cortas desde v_1 hasta otros nodos del grafo, se actualiza el vector de distancia actual Dist cada vez que se selecciona un nuevo elemento y se coloca en S . Durante la k -ésima iteración, supongase que S , denota el conjunto de nodos más próximos seleccionados hasta esa iteración, y que u_k denota el nodo siguiente en proximidad. Las distancias más cortas recalculadas para todos los nodos que no están en S_k y son adyacentes a u_k se obtiene mediante el siguiente fragmento de programa:

```
for i >= to n
    if (v2 es adyacente a u2) and (Vi esta en Si))
        then Dist[Vi] := min(Dist[Vi] Dist [Ua] + Coste [Ui, Vi])
```

A cada interacción, antes de hallar el nodo siguiente en proximidad, el vector de distancia actual contendrá la distancia más corta entre v_1 y todos los nodos del grafo, siempre y cuando todos los nodos intermedios de todos los caminos se hayan seleccionando dentro de S . Observe que la distancia actual hasta u_k desde v_1 sigue siendo la misma (esto es, la distancia más corta desde v_1).

Esta observación sigue siendo cierta para todos los nodos que no sean adyacentes a u_k . Si no hay ningún camino desde v_1 hasta un cierto nodo que tenga esta propiedad, entonces se da el valor INF a la distancia actual de este ultimo nodo. Inicialmente, se hace $\text{Dist}[v_1] = 0$, $\text{Dist}[v_2] = 3$, $\text{Dist}[v_3] = 6$ y $\text{Dist}[v_i] = \text{INF}$ para todos los demás nodos, y $S = \{v_1\}$.



En la primera interacción se selecciona el nodo siguiente en proximidad con respecto al nodo v_1 . El nodo más próximo a v_1 es v_2 , y por tanto se pone v_2 en S . A continuación se calcula la distancia más corta de todos los nodos adyacentes a v_2 y que no estén en S . El nuevo cálculo de longitudes desde v_2 hasta v_3 , v_4 y v_5 produce, respectivamente, los valores 8, 4 y 6. Esto completa la primera interacción que se resume en la segunda fila de la figura 7.44. Las distancias actualizadas van entre paréntesis.

Durante la segunda interacción se busca el nodo segundo en proximidad con respecto al nodo v_1 . Hay tres nodos adyacentes a v_2 : v_3 , v_4 y v_5 . Por tanto se selecciona el nodo v_4 como segundo nodo en proximidad a v_1 , y se pone en S . Se vuelven a calcular las distancias actualizadas por los nodos adyacentes a v_4 y que no están en S . El nuevo cálculo de las distancias a v_3 y v_5 produce los valores de 5 y 8, respectivamente.

En la tercera iteración, los nodos v_3 y v_5 son adyacentes a v_4 , y v_3 , es el más próximo. Por lo tanto, el próximo nodo que se seleccione es v_3 . Los nodos v_5 y v_6 son adyacentes a v_3 , y el nuevo cálculo de sus distancias produce, respectivamente, los valores 7 y 11. La distancia 11 es la distancia mínima actual a v_6 ; sin embargo, la distancia 5 no es la distancia mínima a v_5 y por tanto se mantiene la distancia mínima igual a 6.

Las dos iteraciones siguientes seleccionan sucesivamente a v_5 y v_6 . La última iteración produce una distancia más corta entre v_1 y v_6 cuyo valor es 9.

El enfoque anterior se ha formalizado en el procedimiento que se dan en la 7.45. Observe que algunas de las sentencias del procedimiento no son sentencias validas sintácticamente en Pascal. La otra forma exacta de estas estructuras depende de la forma en que esté representada la estructura del grafo, esto es, dependen de si utilizan una matriz ponderada de adyacencias o bien un directorio de tablas de nodos con listas de adyacencia. Este asunto se examinará de nuevo en breve.

Iteración	Dist						u	S
	v_1	v_2	v_3	v_4	v_5	v_6		
0	0	3	INF	6	INF	INF		{ v_1 }
1	0	(3)	INF(8)	6(4)	INF(6)	INF	v_2	{ v_1, v_2 }
2	0	3	8(5)	(4)	6(8)	INF	v_4	{ v_1, v_2, v_4 }
3	0	3	(5)	4	6(7)	INF(11)	v_3	{ v_1, v_2, v_4, v_3 }
4	0	3	5	4	(6)	11(9)	v_5	{ v_1, v_2, v_4, v_3, v_5 }
5	0	3	5	4	6	(9)	v_6	{ $v_1, v_2, v_4, v_3, v_5, v_6$ }

A continuación se considera brevemente el análisis temporal en caso peor del algoritmo de Dijkstra. Puede ser necesario ejecutar el paso 5 para un máximo de $n - 1$ nodos, puesto que el nodo inicialmente seleccionado es Comienzo. Considérese ahora la ejecución de las sentencias if presentes en los pasos 6 y 8, que son rótulos de los dos bucles internos del bucle principal del paso 5. Si se utiliza una representación de matriz de adyacencia, entonces el caso peor puede necesitar $n - 1$ ejecuciones de las sentencias if. Por lo tanto, el análisis temporal del algoritmo en el caso peor es $O(n^2)$. Sin embargo, en el caso medio u puede ser igual a buscar en el bucle repeat...until mucho antes de que sea necesario efectuar las $n - 1$ iteraciones máximas. Por otra parte, si se emplean listas de adyacencia para representar el grafo y el número de aristas del grafo es mucho menor de n^2 (esto es, si el grafo es disperso), digamos con $m = O(n)$ donde m denota el número de aristas, entonces suponiendo que se almacene la matriz de costes, el algoritmo puede ser bastante mejor que $O(n^2)$.

El algoritmo considerado busca la distancia más corta de un nodo a otro. Se puede emplear un enfoque similar al utilizado en el algoritmo de Floyd dado anteriormente para hallar los nodos del camino más corto. Con este objeto, se puede guardar en cada iteración el índice del nodo predecesor dentro del camino más corto hasta el nodo siguiente en proximidad que se coloca en S. Esta información se puede obtener modificando el paso 8 de la figura 7.45. Cuando se produce una disminución de la distancia actual porque hay una arista que va desde el nodo u hasta el nodo i , hasta con registrar el nodo u como predecesor. Eventualmente, cuando el nodo final entre en S, se puede retroceder hasta el nodo especificado. Comienzo empleando la información relativa a los nodos predecesores.

```

procedure Dijkstra (Comienzo, Final : Nodo;
                    n : integer;
                    var Dist : TipoDist);
{Dado un grafo ponderado y su representación, que se considera más corta desde el nodo inicial, Comienzo, hasta el nodo de destino, Final. Se supone que Nodo es un tipo subrango 1..n, por sencillez. Si no existe una distancia actual desde el nodo inicial, se le da el valor INF. El coste de una arista que va desde  $v_i$  hasta  $v_j$  se denota mediante Coste[ $v_i, v_j$ ]. La representación real dependerá de la forma en que se haya representado el grafo. La función min proporciona el más pequeño de sus argumentos.}
var
    i, u : Nodo;
    DistMin : integer;
    S : set of Nodo;
begin
    {Paso 1: Se da el valor infinito al campo distancia de todos los nodos}
    for i := 1 to n do
        Dist[i] := INF;
    {Paso 2: Se selecciona nodo inicial como nodo de partida}
    Dist[Comienzo] := 0;
    S := [Comienzo];
    {Paso 3: Se toma como nodo actual el nodo de partida}
    u := Comienzo;
    {Paso 4: Se calculan las distancias más cortas a todos los nodos adyacentes al nodo inicial}
    for i := 1 to n do
        if not (i in S)
            then Dist[i] := min (Dist[i], Dist[u] + Coste[u, i]);
    {Paso 5: Se selecciona el nodo siguiente en proximidad (u) y se recalculan las distancias más cortas hasta los nodos que no están en S pasando por u}
    repeat
        {Paso 6: Se selecciona el nodo siguiente en proximidad al nodo inicial}
        DistMin := INF;
        for i := 1 to n do
            if not (i in S) and (Dist[i] < DistMin)
                then begin
                    DistMin := Dist[i];
                    u := i
                end;
        {Paso 7: Se añade el nodo siguiente en proximidad al conjunto de nodos más próximos}
        S := S + [u];
    {Paso 8: Se recalculan las distancias más cortas para los nodos que no están en S pasando por u}
    for i := 1 to n do
        if not (i in S)
            then Dist[i] := min (Dist[i], Dist[u] + Coste[u, i]);
    until u = Final;
end;

```

Hay que recordar que el algoritmo de Floyd, que está basado en una representación de matriz de adyacencia del grafo, se emplea para obtener todas las distancias más cortas y todos los caminos de un grafo. El enfoque de Dijkstra también se puede utilizar con este objeto, invocando el procedimiento Dijkstra para todos los nodos del grafo dado. Si el grafo es disperso, este enfoque, desde el punto de vista de un análisis temporal, puede resultar superior al algoritmo de Floyd. Por otra parte, para grafos densos los dos enfoques son del tipo $O(n^3)$, pero es probable que el algoritmo de Floyd resulte más rápido.

Iteraciones	Nodo predecesor en el camino más corto hasta				
	v_2	v_3	v_4	v_5	v_6
0					
1		v_1		v_1	
2			v_2	v_2	v_2
3				v_4	
4					
5					v_5

Igualmente se podría considerar una serie de pasos para determinar la longitud (o peso) de la trayectoria más corta entre dos vértices, digamos a y z , en una gráfica ponderada, el algoritmo asigna etiquetas numéricas a los vértices de la gráfica por medio de un procedimiento iterativo. En cualquier etapa de la iteración, algunos vértices tendrán etiquetas temporales (que no están entre corchetes) y los otros tendrán etiquetas permanentes (que están entre corchetes). Se denotará la etiqueta del vértice v por $L(v)$.

Iteración inicial 0

Deje que V_0 denote el conjunto de todos los vértices v_0 de la gráfica. Al vértice inicial

se le asigna la .etiqueta permanente (O) y a cada uno los otros v

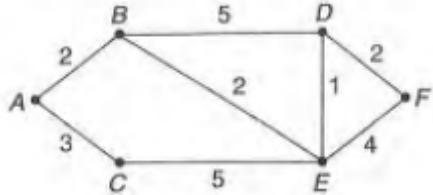
0 la etiqueta temporal oo. Sea $V_1 = V_0 - \{ v_0 \}$, donde v_{c1}' es el vértice inicial al que se le ha asignado una etiqueta permanente.

Iteración 1

Considere que los elementos de V_1 se denotan ahora por v_1 . (Los elementos v_1 son los mismos que los elementos v_0 excluyendo a v_0) Para los elementos de V_1 que son adyacentes a v_{c1}' , las etiquetas temporales se modifican mediante el uso de $L(v_1) = l(v_{c1}') + w(v_{c1}'v_1)$, donde $l(v_{c1}') = 0$, $w(v_{c1}'v_1)$ es el peso de las aristas v_0v_1 y para los otros elementos de V_1 , las etiquetas temporales previas no se alteran. Sea v_i^* el vértice entre las v_1 para las cuales $L(v_1)$ es mínimo. Si hay un empate para la elección de v_i^* , éste se elimina arbitrariamente. Después de esto a $L(v_i^*)$ se le asigna una etiqueta permanente. Sea $V_2 = V_1 - \{ v_i^* \} = \{ v_2 \}$.

Iteración i

Para los elementos de V_i que son adyacentes a v_{i-1}^* , las etiquetas temporales se revisan usando $l(v_i) = L(v_{i-1}^*) + w(v_{i-1}^*, v_i)$ y para los otros elementos de V_i , no se alteran las etiquetas temporales anteriores. Si la etiqueta temporal que se asigna a cualquier vértice en la iteración i -ésima es mayor o igual que la asignada a ella en la iteración $(i-1)$ -ésima la etiqueta previa no se cambia. La iteración se interrumpe cuando al vértice final z se le asigna una etiqueta permanente, aun cuando a algunos vértices podrían no haberseles asignado etiquetas permanentes. La etiqueta permanente de z es la longitud de la trayectoria más corta de a a z . La propia trayectoria más corta se identifica yendo de atrás hacia delante desde z e incluyendo aquellos vértices etiquetados permanentemente a partir de los cuales surgen las etiquetas permanentes subsecuentes. A continuación se considerará un ejemplo y se explicará el algoritmo de Dijkstra paso a paso. Se supondrá que se requiere la trayectoria más corta desde el vértice A hasta el vértice F en la gráfica ponderada que se indica en la figura 7 .58.



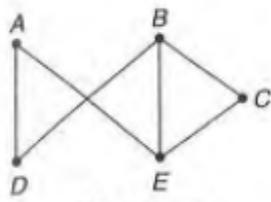
Número de iteración	Detalles de la iteración	Notas
0.	$V_0:$ $A \ B \ C \ D \ E \ F$ $L(v_0): (0) \ \infty \ \infty \ \infty \ \infty \ \infty$	Se suponen las etiquetas iniciales para todos los vértices. A A se le asigna la etiqueta permanente y $L(A^*) = 0$ queda entre paréntesis.
1.	$V_1:$ $A^* \ B \ C \ D \ E \ F$ $L(v_1): \underline{\quad} \ (2) \ 3 \ \infty \ \infty \ \infty$	B y C son vértices adyacentes a A^* . $L(B) = L(A^*) + w(A^*B) = 0 + 2 = 2$ $L(C) = L(A^*) + w(A^*C) = 0 + 3 = 3$ Puesto que $L(B) < L(C)$, a B se le asigna la etiqueta permanente y $L(B^*) = 2$ queda entre paréntesis.
2.	$V_2:$ $A^* \ B^* \ C \ D \ E \ F$ $L(v_2): \underline{\quad} \ (3) \ 7 \ 4 \ \infty$	D y E son vértices adyacentes a B^* . $L(D) = L(B^*) + w(B^*D) = 2 + 5 = 7$ $L(E) = L(B^*) = w(B^*E) = 2 + 2 = 4$ Puesto que C no es adyacente a B^* , $L(C)$ se lleva adelante de la iteración previa como 3. Puesto que $L(C)$ es mínima entre $L(C)$, $L(D)$ y $L(E)$, a C se le asigna la etiqueta permanente y $L(C^*) = 3$ queda entre paréntesis.
3.	$V_3:$ $A^* \ B^* \ C^* \ D \ E \ F$ $L(v_3): \underline{\quad} \ (4) \ \infty$	D y F no son adyacentes a C^* . De modo que $L(D)$ y $L(F)$ se llevan adelante de la iteración (2). $L(E) = L(C^*) + w(C^*E) = 3 + 5 = 8$ Puesto que la $L(E)$ que se revisa la $> L(E)$ anterior $>$ que el valor previo de ahora se retiene el valor previo de $L(E) = 4$. En este caso a E se le asigna la etiqueta permanente y $L(E^*) = 4$ queda entre paréntesis.
4.	$V_4:$ $A^* \ B^* \ C^* \ D \ E^* \ F$ $L(v_4): \underline{\quad} \ (5) \ \underline{\quad} \ \infty$	D y F son adyacentes a E^* . $L(D) = L(E^*) + w(E^*D) = 4 + 1 = 5$ $L(F) = L(E^*) + w(E^*F) = 4 + 4 = 8$ Puesto que $L(D) < L(F)$, a D se le asigna la etiqueta permanente y $L(D^*) = 5$ queda entre paréntesis.
5.	$V_5:$ $A^* \ B^* \ C^* \ D^* \ E^* \ F$ $L(v_5): \underline{\quad} \ (7)$	Puesto que F es el único vértice adyacente a D^* y en vista de que $L(F) = L(D^*) + w(D^*F) = 5 + 2 = 7$, al vértice final F se le asigna la etiqueta permanente y $L(F^*) = 7$ queda entre paréntesis.

Puesto que $L(F^*) = 7$, la longitud de la trayectoria más corta desde A hasta F = 7. Para encontrar la trayectoria más corta, se procede de atrás hacia delante a partir de F de la manera que se explica a continuación: F se convierte en F^* desde D^* en la iteración (5); D se vuelve D^* desde E^* en la iteración (4); E se convierte en E^* desde B (pero no desde C), ya que $L(E) = L(E^*)$ se asumió como la etiqueta 4 en la propia iteración (2); B se vuelve B^* desde A^* en la iteración (1). Por consiguiente, la trayectoria más corta es A - B - E - D - F.

Ejemplos Trabajados

Ejemplo 1.

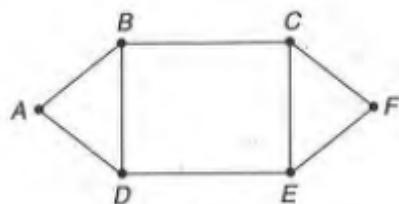
Encuentre cuáles de las siguientes secuencias de vértices son trayectorias simples, trayectorias, trayectorias cerradas (circuitos) y circuitos simples con respecto a la gráfica que se muestra en la siguiente figura:



- a) A - D - E - B - C no es una trayectoria, ya que DE no es una arista de la gráfica dada.
- b) A - D - B - C - E es una trayectoria simple entre los vértices A y E, puesto que los vértices y las aristas implicados son distintos.
- c) A - E - C - B - E - A es una trayectoria cerrada, en vista de que los vértices inicial y final son los mismos y el vértice E aparece dos veces.
- d) C - B - D - A - E - C es un circuito simple, ya que los vértices inicial y final son los mismos y los vértices y las aristas son distintos.
- e) A - D - B - E - C - B es una trayectoria (pero no simple) puesto que el vértice B aparece dos veces.

Ejemplo 2.

Encuentre todas las trayectorias simples de A a F y todos los circuitos en la gráfica dada en la siguiente figura.



Las trayectorias simples de A a F son las siguientes:

1. A - B - C - F;
2. A - D - E - F;
3. A - B - D - E - F;
4. A - D - B - C - F;

5. A - B - C-E - F; E
6. A - D - E - C - F;
7. A - B - D - E - C - F;
8. A - D - B - C - E-F

Los circuitos en la gráfica son los siguientes:

1. A - B - D - A;
2. C - F - E - C;
3. B - C - E - D-B;
4. A - B - C-E-D - A;
5. B - C-F- E - D - B;
6. A - B-C - F - E - D - A.

Ejemplo 3

Determine todas las subgráficas conectadas de la gráfica que se muestra en la siguiente figura que contiene todos los vértices de la gráfica original y que tiene el menor número de aristas posible. ¿En estas subgráficas cuáles son trayectorias y cuáles son trayectorias simples de A a G?

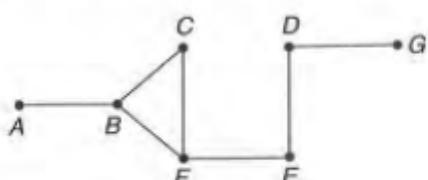


Fig. 7.62

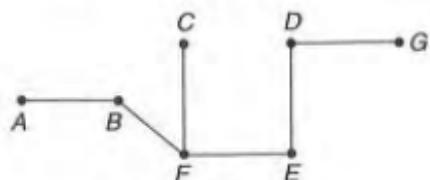


Fig. 7.62a)

Las gráficas en las figuras 7.62a), 7.62b) y 7.62c) son las subgráficas conectadas requeridas. Sin embargo, no hay componentes conectados de la gráfica original en la figura 7.62. En la figura 7.62 a), A - B - F - E - D - G es una trayectoria simple de A a G, en tanto que A - B - F - C - F - E - D - G es una trayectoria de A a G. En la figura 7.62 b), A - B - F - E - D - G es una trayectoria simple, donde A - B - C - B - F - E - D - G es una trayectoria. En la figura 7.62 c), A - B - e -- F- E - D - G es una trayectoria simple que contiene todos los vértices de la gráfica original. No hay trayectorias cerradas y circuitos en las subgráficas, en tanto que éstos están presentes en la gráfica original.

Ejemplo 4. Encuentre analíticamente el número de trayectorias de longitud 4 desde el vértice D hasta el vértice E en la gráfica no dirigida que se muestra en la figura 7.65. Identifique esas trayectorias a partir de las gráficas. La matriz de adyacencia de la gráfica dada es

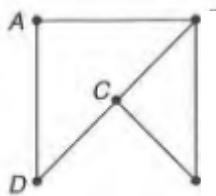


Fig. 7.65

$$A = \begin{bmatrix} A & B & C & D & E \\ 0 & 1 & 0 & 1 & 0 \\ B & 1 & 0 & 1 & 0 & 1 \\ C & 0 & 1 & 0 & 1 & 1 \\ D & 1 & 0 & 1 & 0 & 0 \\ E & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

$$\text{Por multiplicación de matrices, } A^2 = \begin{bmatrix} 2 & 0 & 2 & 0 & 1 \\ 0 & 3 & 1 & 2 & 1 \\ 2 & 1 & 3 & 0 & 1 \\ 0 & 2 & 0 & 2 & 1 \\ 1 & 1 & 1 & 1 & 2 \end{bmatrix}$$

De nuevo, por multiplicación de matrices se obtiene

$$A^4 = \begin{bmatrix} A & B & C & D & E \\ 9 & 3 & 11 & 1 & 6 \\ B & 3 & 15 & 7 & 11 & 8 \\ C & 11 & 7 & 15 & 3 & 8 \\ D & 1 & 11 & 3 & 9 & 6 \\ E & 6 & 8 & 8 & 6 & 8 \end{bmatrix}$$

La entrada en la posición entre paréntesis (4 - 5)-ésima de A^4 es 6. En consecuencia, hay 6 trayectorias cada una de longitud 4 de D a E. Aquellas 6 trayectorias identificadas a partir de las gráficas dadas son como sigue:

1. D - A - D - C - E;
2. D - C - D - C - E;
3. D - A - B - C - E;
4. D - C - E - C - E;
5. D - C - E - B - E;
6. D - C - B - C - E

Ejemplo 5.

Encuentre analíticamente el número de trayectorias de longitud 4 desde el vértice B hasta el vértice D en la gráfica dirigida que se muestra en la figura 7.66. Nombre esas trayectorias utilizando la gráfica.

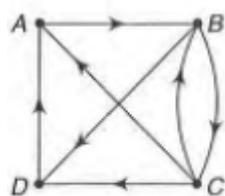


Fig. 7.66

La matriz de adyacencia de la gráfica dada es

$$A = \begin{matrix} & \begin{matrix} A & B & C & D \end{matrix} \\ \begin{matrix} A \\ B \\ C \\ D \end{matrix} & \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \end{matrix}$$

Por multiplicación de matrices se obtiene

$$A^2 = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 2 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

De nuevo, por multiplicación de matrices se obtiene

$$A^4 = \begin{matrix} & \begin{matrix} A & B & C & D \end{matrix} \\ \begin{matrix} A \\ B \\ C \\ D \end{matrix} & \begin{bmatrix} 1 & 2 & 1 & 1 \\ 2 & 2 & 3 & 3 \\ 3 & 3 & 2 & 3 \\ 2 & 1 & 0 & 1 \end{bmatrix} \end{matrix}$$

La entrada en la posición (BD) de A^4 es 3. En consecuencia, hay 3 trayectorias cada una de longitud 4 de B a D.

Hay (1) B - C - B - C - D, (2) B - C - A - B - D y (3) B - D - A - B - D.

Ejemplo 6.

Emplee el algoritmo de Dijkstra para encontrar la trayectoria más corta entre los vértices A y H en la gráfica ponderada que se ilustra en la figura 7.74.

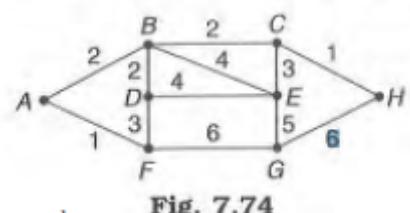


Fig. 7.74

Iteración de Dijkstra									
Número	Detalles de V y $L(v)$								Vértices adyacentes del último v^*
0.	V_0 : A B C D E F G H	—	—	—	—	—	—	—	B y F
	$L(v_0)$: (0)	∞	∞	∞	∞	∞	∞	∞	
1.	V_1 : A* B C D E F G H	—	2	∞	∞	(1)	∞	∞	D y G
	$L(v_1)$: —	—	—	—	—	—	—	—	
2.	V_2 : A* B D E F* G H	—	(2)	4	∞	—	7	∞	C, D y E
	$L(v_2)$: —	—	—	—	—	—	—	—	
3.	V_3 : A* B* C D E F* G H	—	(4)	4	6	—	∞	∞	E y H
	$L(v_3)$: —	—	—	—	—	—	—	—	
4.	V_4 : A* B* C* D E F* G H	—	—	—	∞	7	—	∞	(5)
	$L(v_4)$: —	—	—	—	—	—	—	—	

Puesto que H se alcanza desde C, C se alcanza desde B y B se alcanza a partir de A, la trayectoria más corta es A - B - C - H. Longitud de la trayectoria más corta = $w(AB) + w(BC) + w(CH) = 2 + 2 + 1 = 5$.

Ejemplo 7 Proporcione un ejemplo de una gráfica que contenga i) un circuito euleriano que es también un circuito hamiltoniano ii) un circuito euleriano y un circuito hamiltoniano que son distintos iii) un circuito euleriano, pero no un circuito hamiltoniano iv) un circuito hamiltoniano, pero no un circuito euleriano v) ni un circuito euleriano ni un circuito hamiltoniano.

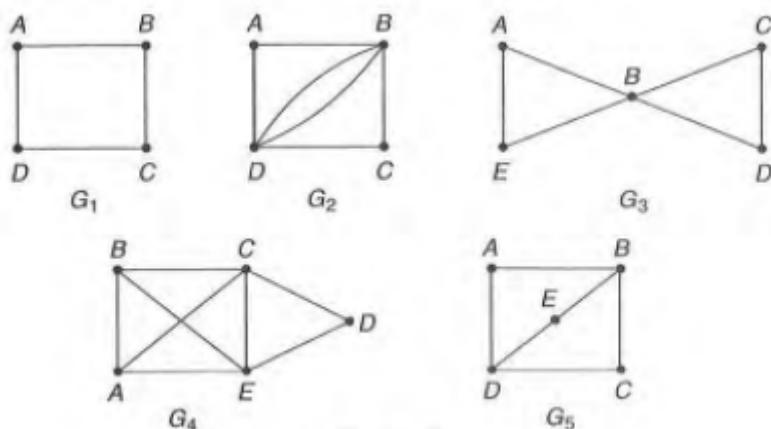


Fig. 7.73

- i) El circuito A - B - C - D - A en G_1 consta de todas las aristas y de todos los vértices, cada uno exactamente una vez. G_1 contiene un circuito que es tanto euleriano como hamiltoniano.
- ii) G_2 contiene el circuito euleriano A - B - D - B - C - D - A y el circuito hamiltoniano A - B - C - D - A, pero los dos circuitos son diferentes.
- iii) G_3 contiene el circuito euleriano A - B - C - D - B - E - A, pero el circuito no es hamiltoniano, ya que el vértice B se repite dos veces.
- iv) G_4 contiene el circuito hamiltoniano A - B - C - D - E - A. Sin embargo, no contiene un circuito euleriano, ya que hay 4 vértices cada uno de grado 3.

v) En G5 el grado de B y el grado de D son iguales a 3. En consecuencia, no hay un circuito de Euler en ella. Además, ningún circuito pasa exactamente una vez a través de cada uno de los vértices.

Ejemplo 8

Encuentre la matriz de la distancia más corta y la correspondiente matriz de la trayectoria más corta para todos los pares de vértices en la gráfica no dirigida ilustrada en la figura 7.75, utilizando el algoritmo de Warshall. La matriz ponderada de la gráfica que se indica está dada por:

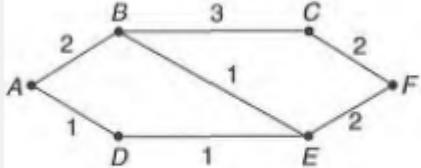


Fig. 7.75

$$W = \begin{matrix} & \begin{matrix} A & B & C & D & E & F \end{matrix} \\ \begin{matrix} A \\ B \\ C \\ D \\ E \\ F \end{matrix} & \begin{pmatrix} 0 & 2 & 0 & 1 & 0 & 0 \\ 2 & 0 & 3 & 0 & 1 & 0 \\ 0 & 3 & 0 & 0 & 0 & 2 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 2 \\ 0 & 0 & 2 & 0 & 2 & 0 \end{pmatrix} \end{matrix}$$

La matriz de la distancia (longitud) inicial L_0 se obtiene de W sustituyendo cada uno de los 0 que no están en la diagonal por ∞ . De tal modo:

$$L_0 = \begin{matrix} & \begin{matrix} A & B & C & D & E & F \end{matrix} \\ \begin{matrix} A \\ B \\ C \\ D \\ E \\ F \end{matrix} & \begin{pmatrix} 0 & 2 & \infty & 1 & \infty & \infty \\ 2 & 0 & 3 & \infty & 1 & \infty \\ \infty & 3 & 0 & \infty & \infty & 2 \\ 1 & \infty & \infty & 0 & 1 & \infty \\ \infty & 1 & \infty & 1 & 0 & 2 \\ \infty & \infty & 2 & \infty & 2 & 0 \end{pmatrix} \end{matrix}$$

En vista de que todas las matrices L_r son simétricas con elementos diagonales cero, sólo es necesario calcular los siguientes elementos en las matrices L_r sucesivas:

$$l_{12}, l_{13}, l_{14}, l_{15}, l_{16}; \quad l_{23}, l_{24}, l_{25}, l_{26}; \quad l_{34}, l_{35}, l_{36}; \quad l_{45}, l_{46} \quad y \quad l_{56}$$

Para la matriz L_1 los elementos anteriores están dados por

$$l_g = \min [l_g; l_{ii} + l_g \text{ de la matriz } L_0]$$

$$\begin{aligned} \text{De tal modo, } l_{12} \text{ de } L_1 &= \min [l_{12}; l_{11} + l_{12} \text{ de } L_0] \\ &= \min [2; 0 + 2] = 2, \text{ y así sucesivamente.} \\ l_{23} \text{ de } L_1 &= \min [l_{23}; l_{21} + l_{13} \text{ de } L_0] \\ &= \min [3; 2 + \infty] = 3, \text{ y así sucesivamente.} \\ l_{34} \text{ de } L_1 &= \min [l_{34}; l_{31} + l_{14} \text{ de } L_0] \\ &= \min [\infty; \infty + 1] = \infty, \text{ y así sucesivamente.} \\ l_{45} \text{ de } L_1 &= \min [l_{45}; l_{41} + l_{15} \text{ de } L_0] \\ &= \min [1; 1 + \infty] = 1, \text{ y así sucesivamente.} \\ l_{56} \text{ de } L_1 &= \min [l_{56}; l_{51} + l_{16} \text{ de } L_0] \\ &= \min [2; \infty + \infty] = 2, \text{ y así sucesivamente.} \end{aligned}$$

$$\text{Por tanto, } L_1 = \begin{pmatrix} 0 & 2 & \infty & 1 & \infty & \infty \\ 2 & 0 & 3 & 3 & 1 & \infty \\ \infty & 3 & 0 & \infty & \infty & 2 \\ 1 & 3 & \infty & 0 & 1 & \infty \\ \infty & 1 & \infty & 1 & 0 & 2 \\ \infty & \infty & 2 & \infty & 2 & 0 \end{pmatrix}.$$

Procediendo de esta manera, los elementos requeridos de la matriz L_r se obtienen usando la regla l_g de $L_r = \min [l_g; l_{ir} + l_{rj} \text{ de } L_{r-1}]$, donde $r = 2, 3, 4, 5, 6$.

De manera correspondiente, las matrices sucesivas están dadas por

$$L_2 = \begin{pmatrix} 0 & 2 & 5 & 1 & 3 & \infty \\ 2 & 0 & 3 & 3 & 1 & \infty \\ 5 & 3 & 0 & 6 & 4 & 2 \\ 1 & 3 & 6 & 0 & 1 & \infty \\ 3 & 1 & 4 & 1 & 0 & 2 \\ \infty & \infty & 2 & \infty & 2 & 0 \end{pmatrix}; \quad L_3 = \begin{pmatrix} 0 & 2 & 5 & 1 & 3 & 7 \\ 2 & 0 & 3 & 3 & 1 & 5 \\ 5 & 3 & 0 & 6 & 4 & 2 \\ 1 & 3 & 6 & 0 & 1 & 8 \\ 3 & 1 & 4 & 1 & 0 & 2 \\ 7 & 5 & 2 & 8 & 2 & 0 \end{pmatrix}$$

$$L_4 = \begin{pmatrix} 0 & 2 & 5 & 1 & 2 & 7 \\ 2 & 0 & 3 & 3 & 1 & 5 \\ 5 & 3 & 0 & 6 & 4 & 2 \\ 1 & 3 & 6 & 0 & 1 & 8 \\ 2 & 1 & 4 & 1 & 0 & 2 \\ 7 & 5 & 2 & 8 & 2 & 0 \end{pmatrix}; \quad L_5 = \begin{pmatrix} 0 & 2 & 5 & 1 & 2 & 4 \\ 2 & 0 & 3 & 2 & 1 & 3 \\ 5 & 3 & 0 & 5 & 4 & 2 \\ 1 & 2 & 5 & 0 & 1 & 3 \\ 2 & 1 & 4 & 1 & 0 & 2 \\ 4 & 3 & 2 & 3 & 2 & 0 \end{pmatrix}$$

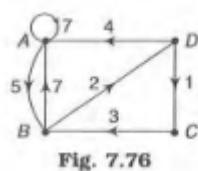
$$L_6 = \begin{matrix} A & B & C & D & E & F \\ A & 0 & 2 & 5 & 1 & 2 & 4 \\ B & 2 & 0 & 3 & 2 & 1 & 3 \\ C & 5 & 3 & 0 & 5 & 4 & 2 \\ D & 1 & 2 & 5 & 0 & 1 & 3 \\ E & 2 & 1 & 4 & 1 & 0 & 2 \\ F & 4 & 3 & 2 & 3 & 2 & 0 \end{matrix}$$

L_6 es la matriz de distancia más corta requerida que produce las distancias más cortas entre todos los pares de vértices de la gráfica dada. La matriz de trayectoria más corta correspondiente es como sigue:

$$\begin{matrix} A & B & C & D & E & F \\ A & — & AB & ABC & AD & ADE & ADEF \\ B & BA & — & BC & BED & BE & BEF \\ C & CBA & CB & — & CFED & CFE & CF \\ D & DA & DEB & DEF C & — & DE & DEF \\ E & EDA & EB & EFC & ED & — & EF \\ F & FEDA & FEB & FC & FED & FE & — \end{matrix}$$

Ejemplo 9

Determine la matriz de distancia más corta y la correspondiente matriz de trayectoria más corta para todos los pares de vértices en la gráfica ponderada dirigida que se presenta en la figura 7.76, utilizando el algoritmo de Warshall. La matriz ponderada de la gráfica que se ilustra es:



$$W = \begin{bmatrix} A & B & C & D \\ A & 7 & 5 & 0 & 0 \\ B & 7 & 0 & 0 & 2 \\ C & 0 & 3 & 0 & 0 \\ D & 4 & 0 & 1 & 0 \end{bmatrix}$$

La matriz de distancia (longitud) inicial L_0 se obtiene de W sustituyendo cada uno de todos los 0 por ∞ .

De tal modo,

$$L_0 = \begin{bmatrix} 7 & 5 & \infty & \infty \\ 7 & \infty & \infty & 2 \\ \infty & 3 & \infty & \infty \\ 4 & \infty & 1 & \infty \end{bmatrix}$$

Utilizando el algoritmo de Warshall y procediendo como en el ejemplo anterior se obtiene

$$L_1 = \begin{bmatrix} 7 & 5 & \infty & \infty \\ 7 & 12 & \infty & 2 \\ \infty & 3 & \infty & \infty \\ 4 & 9 & 1 & \infty \end{bmatrix}$$

uso este libro y lo estudio. Que los dioses lo guarden

$$L_2 = \begin{bmatrix} 7 & 5 & \infty & 7 \\ 7 & 12 & \infty & 2 \\ 10 & 3 & \infty & 5 \\ 4 & 9 & 1 & 11 \end{bmatrix}; L_3 = \begin{bmatrix} 7 & 5 & \infty & 7 \\ 7 & 12 & \infty & 2 \\ 10 & 3 & \infty & 5 \\ 4 & 4 & 1 & 6 \end{bmatrix}$$

$$L_4 = \begin{bmatrix} 7 & 5 & 8 & 7 \\ 7 & 11 & 3 & 2 \\ 9 & 3 & 6 & 5 \\ 4 & 4 & 1 & 6 \end{bmatrix},$$

que es la matriz de distancia más corta requerida que produce las distancias más cortas entre todos los pares de vértices de la gráfica dada. La matriz de trayectoria más corta correspondiente es como sigue:

$$\begin{array}{cccc} & A & B & C & D \\ A & AA & AB & ABDC & ABD \\ B & BA & BDAB & BDC & BD \\ C & CBDA & CB & CBDC & CBD \\ D & DA & DCB & DC & DCBD \end{array}$$

Ejemplo 10

Encuentre una trayectoria hamiltoniana o un circuito hamiltoniano, si éstos existen en cada una de las 3 gráficas de la figura 7.72. Si éstos no existen, explique por qué.

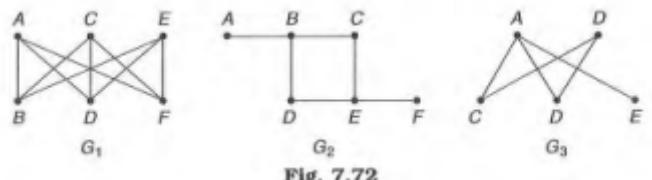


Fig. 7.72

G_1 contiene un circuito hamiltoniano, por ejemplo, $A - B - C - D - E - F - A$. De hecho, hay 5 circuitos hamiltonianos más en G_1 , a saber, $A - B - C - F - E - D - A$, $A - B - E - D - e - F - A$, $A - B - E - F - e - D - A$, $A - D - e - B - E - F - A$ y $A - D - E - B - C - F - A$.

G_2 no contiene ni una trayectoria hamiltoniana ni un circuito hamiltoniano, ya que ninguna trayectoria que contenga todos los vértices debe contener una de las aristas $A - B$ y $E - F$ más de una vez. G_3 contiene 2 trayectorias hamiltonianas de C a E y de D a E , a saber, $C - B - D - A - E$ y $D - B - C - A - E$, pero ningún circuito hamiltoniano.

Libros consultados

CHAPTER 12: Minimum Spanning Tree and Shortest-Path Algorithms ■ 367

The number of spanning trees, even for relatively small graphs, is usually enormous, making an enumerative brute-force search infeasible. Indeed, Cayley proved that the complete graph K_n on n vertices contains

$$n^{n-2} \text{ spanning trees.}$$

Fortunately, efficient algorithms based on the greedy method do exist for finding minimum spanning trees. We discuss two of the more famous algorithms, Kruskal's and Prim's. These algorithms are similar in the sense that their selection functions always choose an edge of minimum weight among the remaining edges. However, Prim's algorithm selection function only considers edges incident to vertices already included in the tree. In particular, the partial solutions built by Prim's algorithm are trees, whereas the partial solutions built by Kruskal's algorithm are forests.

12.1.1 Kruskal's Algorithm

Given a connected graph $G = (V, E)$, with $|V| = n$ and $|E| = m$, and a weight function w defined on the edge set E , Kruskal's algorithm finds a minimum spanning tree T of G by constructing a sequence of n forests F_0, F_1, \dots, F_{n-1} , where F_0 is the empty forest and F_i is obtained from F_{i-1} by adding a single edge e . The edge e is chosen so that it has minimum weight among all the edges not belonging to F_{i-1} and doesn't form a cycle when added to F_{i-1} . Kruskal's algorithm is illustrated for a sample graph in Figure 12.2:

For convenience, we assume that the input graph G to Kruskal's algorithm is connected. However, Kruskal's algorithm is easily modified to accept any graph G (connected or disconnected) and to output a minimum (weight) forest, each of whose trees spans a (connected) component of G .

Using a proof by contradiction, we now show that the spanning tree K generated by Kruskal's algorithm is a minimum spanning tree. For convenience, we will assume that the edge weights are distinct (the proof can be slightly modified to hold for nondistinct edge weights; see Exercise 12.4). Let the edges of K be denoted by x_1, x_2, \dots, x_{n-1} , listed in increasing order of their weights. Assume that K is not a minimum spanning tree, and let T be a minimum spanning tree with edges y_1, y_2, \dots, y_{n-1} , listed in increasing order of their weights. Let j denote the first index i such that $x_i \neq y_i$, so that $x_i = y_j$, $1 \leq i \leq j - 1$. Since x_j was chosen by the greedy strategy, we have $w(x_j) < w(y_j)$.

3

Graphs

3.1 Introduction

Graphs are mathematical discrete structures which have major role in computer science (algorithms and computation), electrical engineering (communication networks and coding theory), operations research (scheduling), and in many fields of engineering and also in sciences such as chemistry, biochemistry (genomics), biology, linguistics, sociology, and other fields. For instance, graphs are encoded to represent the relationship between objects. Many real-world situations can conveniently be described by means of a diagram consisting of a set of points together with lines joining certain pairs of these points. For example, the points could represent people, with lines joining pairs of friends; or the points might be communication centres, with lines representing communication links. Notice that in such diagrams, one is mainly interested in whether or not two given points are joined by a line; the manner in which they are joined is immaterial. A mathematical abstraction of situations of this type gives rise to the concept of a graph.

In this chapter, we focus on the terminology of graphs, its various types, connectivity of graphs, Eulerian path, and Hamiltonian path. Graph theory is introduced as an abstract mathematical system. The most common representation of a graph is by means of a diagram, in which the vertices are represented as points and each edge as a line segment joining its end vertices.

3.2 Graphs and Graph Models

Definition 3.2.1 Graph: A graph $G = (V, E, \phi)$ consists of a non-empty set $V = \{v_1, v_2, \dots\}$ called the set of vertices of the graph and $E = \{e_1, e_2, \dots\}$ called the set of edges of the graph, and ϕ is a mapping from the set of edges E to the set of ordered or unordered pair of elements of vertices.

3

Graphs

3.1 Introduction

Graphs are mathematical discrete structures which have major role in computer science (algorithms and computation), electrical engineering (communication networks and coding theory), operations research (scheduling), and in many fields of engineering and also in sciences such as chemistry, biochemistry (genomics), biology, linguistics, sociology, and other fields. For instance, graphs are encoded to represent the relationship between objects. Many real-world situations can conveniently be described by means of a diagram consisting of a set of points together with lines joining certain pairs of these points. For example, the points could represent people, with lines joining pairs of friends; or the points might be communication centres, with lines representing communication links. Notice that in such diagrams, one is mainly interested in whether or not two given points are joined by a line; the manner in which they are joined is immaterial. A mathematical abstraction of situations of this type gives rise to the concept of a graph.

In this chapter, we focus on the terminology of graphs, its various types, connectivity of graphs, Eulerian path, and Hamiltonian path. Graph theory is introduced as an abstract mathematical system. The most common representation of a graph is by means of a diagram, in which the vertices are represented as points and each edge as a line segment joining its end vertices.

3.2 Graphs and Graph Models

Definition 3.2.1 Graph: A graph $G = (V, E, \phi)$ consists of a non-empty set $V = \{v_1, v_2, \dots\}$ called the set of vertices of the graph, and $E = \{e_1, e_2, \dots\}$ called the set of edges of the graph, and ϕ is a mapping from the set of edges E to the set of ordered or unordered pair of elements of vertices.

los vértices de la gráfica por medio de un procedimiento iterativo. En cualquier etapa de la iteración, algunos vértices tendrán etiquetas temporales (que no están entre corchetes) y los otros tendrán etiquetas permanentes (que están entre corchetes). Se denotará la etiqueta del vértice v por $L(v)$.

Iteración inicial 0

Deje que V_0 denote el conjunto de todos los vértices v_0 de la gráfica. Al vértice inicial se le asigna la etiqueta permanente (0) y a cada uno los otros v_0 la etiqueta temporal ∞ . Sea $V_1 = V_0 - \{v_0^*\}$, donde v_0^* es el vértice inicial al que se le ha asignado una etiqueta permanente.

Iteración 1

Considere que los elementos de V_1 se denotan ahora por v_1 . (Los elementos v_1 son los mismos que los elementos v_0 excluyendo a v_0^* .) Para los elementos de V_1 que son adyacentes a v_0^* , las etiquetas temporales se modifican mediante el uso de $L(v_1) = L(v_0^*) + w(v_0^*v_1)$ donde $L(v_0^*) = 0$, $w(v_0^*v_1)$ es el peso de las aristas $v_0^*v_1$ y para los otros elementos de V_1 , las etiquetas temporales previas no se alteran. Sea v_1^* el vértice entre las v_1 para las cuales $L(v_1)$ es mínimo. Si hay un empate para la elección de v_1^* , éste se elimina arbitrariamente. Después de esto a $L(v_1^*)$ se le asigna una etiqueta permanente. Sea $V_2 = V_1 - \{v_1^*\} \equiv \{v_2\}$.

Iteración i

Para los elementos de V_i que son adyacentes a v_{i-1}^* , las etiquetas temporales se revisan usando $L(v_i) = L(v_{i-1}^*) + w(v_{i-1}^*v_i)$ y para los otros elementos de V_i , no se alteran las etiquetas temporales anteriores. Si la etiqueta temporal que se asigna a cualquier vértice en la iteración i -ésima es mayor o igual que la asignada a ella en la iteración $(i-1)$ -ésima la etiqueta previa no se cambia.

La iteración se interrumpe cuando al vértice final z se le asigna una etiqueta permanente, aun cuando a algunos vértices podrían no haberseles asignado etiquetas permanentes. La etiqueta permanente de z es la longitud de la trayectoria más corta de a a z . La propia trayectoria más corta se identifica yendo de atrás hacia delante desde z e incluyendo aquellos vértices etiquetados permanentemente a partir de los cuales surgen las etiquetas permanentes subsecuentes.

A continuación se considerará un ejemplo y se explicará el algoritmo de Dijkstra paso a paso. Se supondrá que se requiere la trayectoria más corta desde el vértice A hasta el vértice F en la gráfica ponderada que se indica en la figura 7.58.

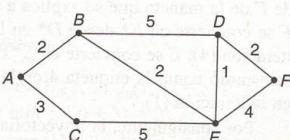


Fig. 7.58

tada unilateralmente. Una digráfica conectada fuertemente también es tanto unilateralmente como débilmente.

Por ejemplo, se considerarán las gráficas que se muestran en las figuras 7.56 y 7.57.

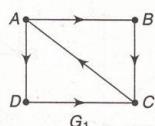


Fig. 7.55

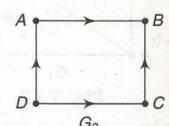


Fig. 7.56

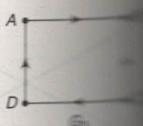


Fig. 7.57

G_1 es una gráfica fuertemente conectada, ya que los posibles pares en G_1 son (A, B) , (A, C) , (A, D) , (B, C) , (B, D) y (C, D) y existe una trayectoria desde el primer vértice hasta el segundo y desde el segundo hasta el tercero para todos los pares.

Por ejemplo, considere el par (A, B) . Claramente la trayectoria de A a B es $A - B$ y la trayectoria de B a A es $B - C - A$.

Similarmente, si se toma el par (B, D) , la trayectoria de B a D es $B - C - D$ y la trayectoria de D a B es $D - C - A - B$.

Claramente G_2 es sólo una gráfica débilmente conectada.

G_3 es unilateralmente conectada, puesto que hay una trayectoria de A a B pero no de B a A . De igual modo, hay una trayectoria de B a C pero no de C a B , pero no de B a D .

Definición

Una subgráfica de una digráfica G que está fuertemente conectada y se encuentra contenida en una subgráfica más grande fuertemente conectada, la subgráfica fuertemente conectada máxima, recibe el nombre de *componente fuertemente conectado de G* [vea el ejemplo (7.8)].

ALGORITMOS DE LA TRAYECTORIA MÁS CORTA

Una gráfica en la cual a cada arista ' e ' se le asigna un número real no negativo se denomina *gráfica ponderada*; $w(e)$ recibe el nombre de *peso de la arista* y puede representar distancia, tiempo, costo, etc., en ciertas unidades.

La *trayectoria más corta* entre dos vértices en una gráfica ponderada es la trayectoria de menor peso. En una gráfica no ponderada, las trayectorias más cortas equivalen a aquella con el menor número de aristas.

En esta sección se tratará el problema de determinar la trayectoria más corta entre dos vértices cualesquiera en una gráfica ponderada. Se dispondrá de algoritmos para determinar la trayectoria más corta en una gráfica ponderada. Los estudiantes estudiarán dos de ellos aquí, uno de los cuales fue descubierto por Dijkstra y el otro por Marshall.

Algoritmo de Dijkstra

Para determinar la longitud (o peso) de la trayectoria más corta entre dos vértices, digamos a y z , en una gráfica ponderada, el algoritmo asigna etiquetas

procedure *Dijkstra*($G, w, r, \text{Parent}[0:n - 1], \text{Dist}$)
Input: G (a connected graph with vertex set V and edge set E)
 r (a vertex of G)
 w (a weight function on E)
Output: $\text{Parent}[0:n - 1]$ (parent array of a shortest-path tree)
 $\text{Dist}[0:n - 1]$ (array of weights of shortest paths from r)
 $\text{InTheTree}[v] \leftarrow \text{false}$.
for $v \leftarrow 1$ to n **do**
 $\text{Dist}[v] \leftarrow \infty$
 $\text{InTheTree}[v] \leftarrow \text{false}$.
endfor
 $\text{Dist}[r] \leftarrow 0$
 $\text{Parent}[r] \leftarrow -1$
for $Stage \leftarrow 1$ to $n - 1$ **do**
 Select vertex u that minimizes $\text{Dist}[u]$ over all u such that $InTheTree[u] = \text{false}$.
 $InTheTree[u] \leftarrow \text{true}$.
 for each vertex v such that $uv \in E$ **do** //update $\text{Dist}[v]$ and $\text{Parent}[v]$
 if .not. $InTheTree[v]$ **then**
 if $\text{Dist}[u] + w(uv) < \text{Dist}[v]$ **then**
 $\text{Dist}[v] \leftarrow \text{Dist}[u] + w(uv)$
 $\text{Parent}[v] \leftarrow u$
 endif
 endif
 endfor
endfor
end Dijkstra

REMARKS

- As with procedure *Prim*, procedure *Dijkstra* terminates after only $n - 1$ stages, even though there are n vertices in the final shortest-path tree. Another iteration of the **for loop** for Stage would result in no change to the arrays $\text{Dist}[0:n - 1]$ and $\text{Parent}[0:n - 1]$.
- Although *Prim's algorithm* and *Dijkstra's algorithm* have some similarity and they both find spanning trees, they solve different problems. Simple examples show that a shortest path can fail to be a minimum spanning tree (see Exercise 12.18).

The shortest-path algorithms presented in this section for undirected graphs generalize naturally to digraphs. In fact, essentially unchanged code for procedure *Dijkstra* applies to digraphs. We merely have to be careful in the associated graphs to note that uv corresponds to a directed edge $uv = (u, v)$ from u to v , as opposed to the undirected edge $uv = \{u, v\}$. We illustrate the *Dijkstra's algorithm* for a sample digraph in Figure 12.10, where the current shortest distance $\text{Dist}[v]$ is shown outside each vertex v .

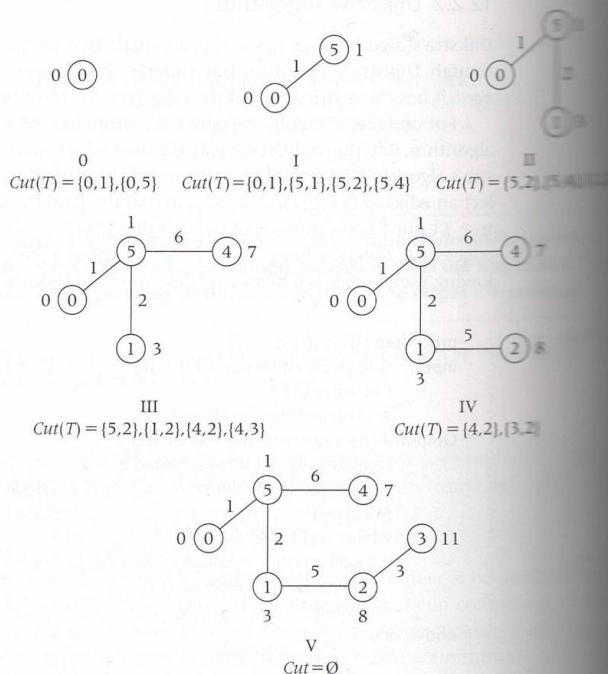
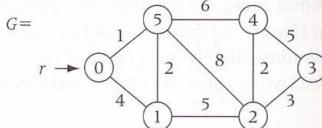
edges $uv \in Cut(T_i)$, where $Dist[u]$ is the weight of $P_u^{(i)}$. By induction assumption, for all $x \in V(T_i)$, $P_x^{(i+1)} = P_x^{(i)}$ is a shortest path in G from r to x . It remains to show that $P_v^{(i+1)}$ is a shortest path in G from r to v . Consider any path Q in G from r to v . Because deleting the edges in $Cut(T_i)$ disconnects G with vertices r and v lying in different components, Q must contain edge xy from $Cut(T_i)$, $x \in V(T_i)$, such that the subpath Q_x of Q from r to x lies entirely in T_i . Then we have

$$\begin{aligned} weight(Q) &\geq weight(Q_x) + w(xy) && (\text{since the weight function is nonnegative}) \\ &\geq weight(P_x^{(i)}) + w(xy) && (\text{since } P_x^{(i)} \text{ is a shortest path in } G \text{ from } r \text{ to } x \text{ in } G) \\ &\geq weight(P_u^{(i)}) + w(uv) && (\text{by the greedy choice Dijkstra makes}) \\ &= weight(P_u^{(i+1)}) \end{aligned}$$

Since Q was chosen to be an arbitrary path in G from r to v , it follows that $P_v^{(i+1)}$ is a shortest path in G from r to v . This completes the induction step and the correctness proof of procedure *Dijkstra*. ■

As with Kruskal's and Prim's algorithms, the complexity of Dijkstra's algorithm depends on specific implementation details. We now implement the procedure *Dijkstra*, which is similar to the procedure *Prim* in that the tree T is dynamically grown using an array $Parent[0:n - 1]$. We also maintain a Boolean array *InTheTree* to keep track of the vertices not in T . Again, rather than having to explicitly maintain the sets $Cut(T_i)$, we can efficiently select the next edge uv to be added to the tree by maintaining an array $Dist[0:n - 1]$, where $Dist[v]$ is the distance from r to v in T and $Dist[v] = \infty$ if v is not in T . Upon completion of *Dijkstra*, the distance from the root r to all the vertices of G is contained in the array $Dist[0:n - 1]$. At each intermediate stage $Dist[v]$ has the following dynamic interpretation. If $v \in V(T)$, then $Dist[v]$ is the weight of a path in T from r to v . If $v \notin V(T)$ and v is adjacent to a vertex of T , then $Dist[v]$ is the minimum value of $Dist[u] + w(uv)$ over all $uv \in Cut(T)$. If $v \notin V(T)$ and v is not adjacent to any vertex of T , then $Dist[v] = \infty$. $Dist[v]$ is initialized to ∞ for each vertex $v \neq r$ and $Dist[r]$ is initialized to 0. Each time a vertex u is added to the tree T , we need only update $Parent[v]$ and $Dist[v]$ for those vertices v that are adjacent to u and do not belong to the tree T . If $Dist[v] > Dist[u] + w(uv)$, then we set $Parent[v] = u$ and $Dist[v] = Dist[u] + w(uv)$.

FIGURE 12.9
Tree T and $Cut(T)$ at each stage in Dijkstra's algorithm



Lemma 12.2.2

For each vertex $v \in V(T_i)$, $P_i^{(v)}$ is a shortest path in G from r to v .

PROOF

Basis step. Clearly, the lemma is true for $i = 0$ because T_0 consists of a single vertex r .

Induction step. Assume that the lemma holds for T_i . The tree T_{i+1} is obtained from T_i by adding a single edge uv that minimizes $Dist[u] + w(uv)$.

When there are no negative cycles, a shortest path contains at most $n - 1$ edges. Hence, the correctness of *BellmanFord* when there are no negative cycles follows immediately from Lemma 12.2.1. We leave the correctness proof of *BellmanFord* when there are negative cycles as an exercise.

12.2.2 Dijkstra's Algorithm

Dijkstra's algorithm grows a shortest-path tree using the greedy method. Although Dijkstra's algorithm has superior worst-case performance to *BellmanFord*, it has the restriction that the edge weights must be nonnegative.

For undirected graphs, Dijkstra's algorithm follows a similar strategy to Prim's algorithm, differing only in the way the next edge is selected. At each stage in Dijkstra's algorithm, instead of selecting an edge of minimum weight in $Cut(T)$, we select an edge $uv \in Cut(T)$, $u \in V(T)$, so that the path from r to v in the augmented tree $T \cup \{uv\}$ is shortest—that is, such that $Dist[u] + w(uv)$ is minimized over all edges $uv \in Cut(T)$, where $Dist[u]$ is the weight of a path from r to u in T . The following pseudocode is a high-level description of Dijkstra's algorithm.

```

procedure Dijkstra( $G, a, w, T$ )
Input:  $G$  (a graph with vertex set  $V = \{0, \dots, n - 1\}$  and edge set  $E$ )
         $r$  (a vertex of  $G$ )
         $w$  (a nonnegative weighting on  $E$ )
Output:  $T$  (a shortest-path tree rooted at  $r$ )
 $Dist[0:n - 1]$  is an array initialized to  $\infty$ 
 $T$  is initialized to be the tree consisting of the single vertex  $r$ 
 $Dist[r] \leftarrow 0$ 
while  $Cut(T) \neq \emptyset$  do
    select an edge  $uv \in Cut(T)$  such that  $Dist[u] + w(uv)$  is minimized
    add vertex  $v$  and edge  $uv$  to  $T$ 
endwhile
end Dijkstra

```

Figure 12.9 shows the tree T at each stage of Dijkstra's algorithm for a sample weighted graph G . In this figure, the label on the outside of each vertex v already included in the growing tree T is the weight of the path from v to a in T .

Let T_i be the tree generated by procedure *Dijkstra* after i iterations of the **while** loop, $i = 0, 1, \dots, t$, where t is the number of iterations of the loop ($t \leq n - 1$). For $v \in V(T_i)$, let $P_i^{(v)}$ denote the path from r to v in T_i , $i = 0, 1, \dots, t$. We prove the correctness of procedure *Dijkstra* by establishing the following lemma.

The number of spanning trees, even for relatively small graphs, is usually enormous, making an enumerative brute-force search infeasible. Indeed, Cayley proved that the complete graph K_n on n vertices contains

$$n^{n-2} \text{ spanning trees.}$$

Fortunately, efficient algorithms based on the greedy method do exist for finding minimum spanning trees. We discuss two of the more famous algorithms, Kruskal's and Prim's. These algorithms are similar in the sense that their selection functions always choose an edge of minimum weight among the remaining edges. However, Prim's algorithm selection function only considers edges incident to vertices already included in the tree. In particular, the partial solutions built by Prim's algorithm are trees, whereas the partial solutions built by Kruskal's algorithm are forests.

12.1.1 Kruskal's Algorithm

Given a connected graph $G = (V, E)$, with $|V| = n$ and $|E| = m$, and a weight function w defined on the edge set E , Kruskal's algorithm finds a minimum spanning tree T of G by constructing a sequence of n forests F_0, F_1, \dots, F_{n-1} , where F_0 is the empty forest and F_i is obtained from F_{i-1} by adding a single edge e . The edge e is chosen so that it has minimum weight among all the edges not belonging to F_{i-1} and doesn't form a cycle when added to F_{i-1} . Kruskal's algorithm is illustrated for a sample graph in Figure 12.2.

For convenience, we assume that the input graph G to Kruskal's algorithm is connected. However, Kruskal's algorithm is easily modified to accept any graph G (connected or disconnected) and to output a minimum (weight) forest, each of whose trees spans a (connected) component of G .

Using a proof by contradiction, we now show that the spanning tree K generated by Kruskal's algorithm is a minimum spanning tree. For convenience, we will assume that the edge weights are distinct (the proof can be slightly modified to hold for nondistinct edge weights; see Exercise 12.4). Let the edges of K be denoted by x_1, x_2, \dots, x_{n-1} , listed in increasing order of their weights. Assume that K is not a minimum spanning tree, and let T be a minimum spanning tree with edges y_1, y_2, \dots, y_{n-1} , listed in increasing order of their weights. Let j denote the first index i such that $x_i \neq y_i$, so that $x_i = y_j$, $1 \leq i \leq j - 1$. Since x_j was chosen by the greedy strategy, we have $w(x_j) < w(y_j)$.

edges represent communication latency. Another example is a diagram showing a network of airplane flights. The least costly flight (in terms of distance, or time) from one airport to another in the network is the shortest directed path from the departure airport to the arrival airport, where the weights are the appropriate costs.

In this chapter, we discuss various algorithms for solving the minimum spanning tree and shortest-path problems.

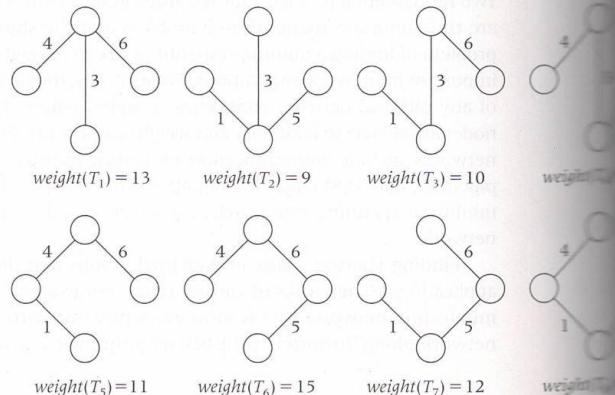
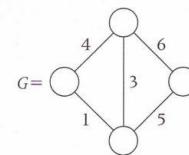
12.1 Minimum Spanning Tree

Given a spanning tree T in a graph G with a weighting w of the edges, the weight of T , denoted $\text{weight}(T)$, is the sum of the w -weights of its edges. The minimum weight over all spanning trees of G , then we call T a minimum spanning tree.

In Figure 12.1, all the spanning trees of a weighted graph on four vertices are enumerated. The minimum spanning tree is then obtained by inspection.

FIGURE 12.1

Enumeration of the spanning trees of G .
The minimum spanning tree is T_2 .



Bibliografía

- VEERARAJAN, T.

Matemáticas discretas con teoría de gráficas y combinatoria
México
McGraw-Hill Interamericana, 2008
- Kolman, B., Busby, R. C., & Ross, S. C. (2008). *Discrete Mathematical Structures* (6a ed.). Pearson.
- Paul, J., & Berman, K. A. (2004). *Algorithms: Sequential, parallel, and distributed*. Course Technology.
- Paul Tremblay, J., & Karl Grassmann, W. (1996). *MATEMÁTICA DISCRETA Y LOGICA: UNA PERSPECTIVA DESDE LA CIENCIA DE LA COMPUTACIÓN* (1.a ed.). PRENTICE-HALL.
- Tremblay, J., & Manohar, R. (2000). *MATEMÁTICAS DISCRETAS CON APLICACIÓN A LAS CIENCIAS DE LA COMPUTACIÓN* (2.a ed.). Compañía Editorial Continental.