INSTITUTO POLITÉCNICO NACIONAL

Escuela Superior de Física y Matemáticas

Algoritmos en Gráficas: Árboles de Expansión Mínima y Caminos Cortos

TESIS QUE PARA OBTENER EL TÍTULO DE
Licenciado en Física y Matemáticas
PRESENTA:

José Antonio González Mijangos [1] [2]
Escuela Superior de Física y Matemáticas
Instituto Politécnico Nacional
Edificio No. 9
Unidad Adolfo López Mateos
07300 México D.F.

ASESOR DE TESIS:
Dr. Adrián Alcántar Torres
ESFM-IPN

Junio del 2006

[1]Con apoyo de proyecto de investigación IPN: Reg. 20061075.
[2]Con apoyo de proyecto de investigación CONACYT: No. 49091.

Dedico esta tesis a mis padres y mis hermanos, por el apoyo y la comprensión para este logro.

No podría haber obtenido este logro de no ser por su ayuda, cariño y comprensión. Para ustedes mi más profundo agradecimiento.

Hago un agradecimiento especial al profesor Adrián Alcántar Torres, por sus sugerencias y comentarios, y por el tiempo que dedicó a trabajar conmigo.

Introducción

Los problemas de gráficas o redes surgen en una gran variedad de situaciones. Las redes de transporte, eléctricas y de comunicaciones, predominan en la vida diaria. La representación de gráficas se utiliza ampliamente en áreas tan diversas como producción, distribución, planeación de proyectos, localización de instalaciones, administración de recursos y planeación financiera, por nombrar solo algunos ejemplos. De hecho, una representación de gráficas proporciona un panorama general tan poderoso y una ayuda conceptual para visualizar las relaciones entre las componentes de los sistemas, que se usa casi en todas las áreas científicas, sociales y económicas.

La aparición de algunos algoritmos ha tenido un impacto importante, al igual que las ideas de ciencias de la computación acerca de estructuras de datos y la manipulación eficiente de los mismos. En consecuencia, ahora se disponen de algoritmos y aplicaciones de computadora y se usan en forma rutinaria para resolver problemas muy grandes que no se habrían podido resolver hace algunas décadas.

El objetivo de esta tesis es el estudio detallado del diseño de cuatro algoritmos para gráficas, los primeros dos serán para el estudio del problema del árbol de expansión mínima en una gráfica no dirigida y los otros dos para el problema de caminos cortos con origen fijo en una gráfica dirigida, los cuales al ser aplicados a una gráfica, nos producirán ya sea un árbol de expansión mínima o un árbol de caminos cortos, según sea el caso.

Para poder resolver este tipo de problemas con gráficas de manera eficiente, es necesario analizar y recorrer todos los vértices de la gráfica de manera sistemática.

En el primer capítulo definiremos un algoritmo e introduciremos algunas definiciones básicas sobre la teoría de gráficas. En el capítulo 2, daremos un algoritmo genérico que será la base para el estudio de los algoritmos que se verán en esta tesis, y probaremos dos resultados, que utilizaremos para el estudio de los algoritmos de Kruskal y Prim del capítulo 3, para el problema del árbol de expansión mínima.

En el capítulo 4, abordaremos el problema de caminos cortos con origen fijo en una gráfica dirigida, daremos algunas definiciones y probaremos algunas propiedades sobre caminos cortos, y veremos las consecuencias de aplicar estas propiedades a una gráfica, para que finalmente en el capítulo 5, estu-

diemos los algoritmos de Dijkstra y Bellman-Ford, y probemos que al aplicar estos algoritmos en una gráfica dirigida pesada, se resuelve el problema de caminos cortos con origen fijo.

Finalmente en el capítulo 6, daremos algunos ejemplos y veremos como al ser aplicados estos algoritmos se resuelven los problemas del árbol de expansión mínima y el problema de caminos cortos con origen fijo

Índice general

Capítulo 1

Preliminares

1.1. Algoritmo

Informalmente un *algoritmo* es cualquier procedimiento computacional bien definido que toma algunos valores, o conjunto de valores, como *entrada* y produce algunos valores, o conjunto de valores, como *salida*. Entonces un algoritmo es una secuencia finita de pasos computacionales que transforman la entrada en la salida.

Cada instrucción tiene un significado preciso y puede ejecutarse con una cantidad finita de memoria en un tiempo finito. Las instrucciones de un algoritmo pueden ejecutarse cualquier número de veces, siempre que ellas mismas indiquen la repetición. No obstante, se exige que un algoritmo termine después de ejecutar un número finito de instrucciones, sin importar cuales fueron los valores de entrada.

También un algoritmo se puede ver como una herramienta para resolver un *problema computacional* bien específico. La declaración del problema especifica en términos generales la relación entrada/salida.

Un algoritmo se dice *algoritmo correcto* si, para cada entrada, éste termina con la salida correcta. Decimos que un algoritmo correcto *resuelve* un problema computacional dado. Un algoritmo incorrecto puede no terminar con algunas entradas, o puede terminar con una respuesta no deseada.

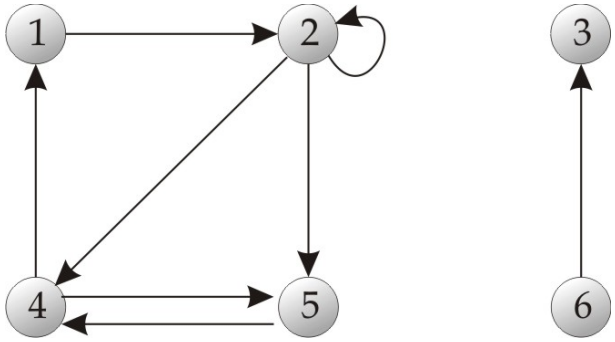


Figura 1. Una gráfica dirigida con 6 vértices y 8 aristas.

1.2. Gráficas

Definición 1.1 Una *gráfica dirigida* G es un par (V, A), donde V es un conjunto finito y A es una relación binaria en V. Los elementos de V son llamados *vértices*, así V es el *conjunto de vértices* de G. El conjunto A es llamado *conjunto de aristas* de G, cuyos elementos son llamados *aristas*.

La figura 1 representa una gráfica dirigida sobre el conjunto de vértices $\{1, 2, 3, 4, 5, 6\}$. Los vértices son representados por círculos en la figura, y las aristas son representadas por flechas. Note que los *lazos*, aristas que unen a un vértice con sí mismo, son posibles.

Definición 1.2 En una *gráfica no dirigida* $G = (V, A)$, el conjunto de aristas A consiste de pares no ordenados de vértices, es decir, una arista es un conjunto $\{u, v\}$, donde $u, v \in V$ y $u \neq v$.

Por convención usamos la notación (u, v) para una arista. En una gráfica no dirigida no existen lazos, pues cada arista consiste de dos vértices distintos. Por una *gráfica* nos referimos a una gráfica dirigida o una gráfica no dirigida.

La figura 2 representa una gráfica no dirigida sobre el conjunto de vértices $\{1, 2, 3, 4, 5, 6\}$. Las aristas son representadas por líneas.

Muchas definiciones para gráficas dirigidas y no dirigidas son las mismas,

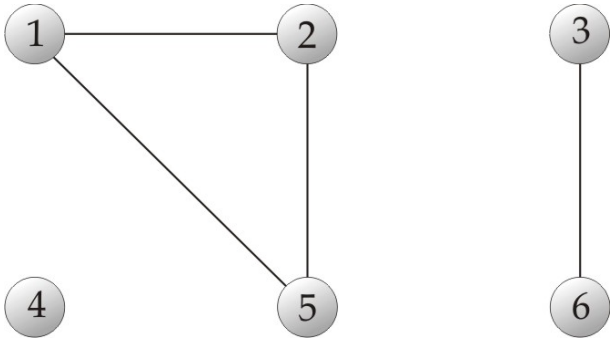


Figura 2. Una gráfica no dirigida con 6 vértices y 4 aristas.

aunque ciertos términos tiene significados ligeramente distintos en los dos contextos.

Definición 1.3 Si (u, v) es una arista en una gráfica dirigida $G = (V, A)$, decimos que (u, v) es *incidente* o *entra* al vértice v y *sale* del vértice u. Si (u, v) es una arista en una gráfica no dirigida $G = (V, A)$, decimos que (u, v) es *incidente sobre* los vértices u y v. Si v es adyacente a u en una gráfica dirigida, escribiremos $u \to v$.

En la figura 1, las aristas que salen del vértice 2 son $(2, 2)$, $(2, 4)$ y $(2, 5)$. Las aristas que entran al vértice 2 son $(1, 2)$ y $(2, 2)$. En la figura 2, las aristas incidentes sobre el vértice 2 son $(1, 2)$ y $(2, 5)$.

Definición 1.4 Si (u, v) es una arista en una gráfica $G = (V, A)$, decimos que el vértice v es *adyacente* al vértice u. Cuando la gráfica es no dirigida, la relación de adyacencia es simétrica.

Si la gráfica es dirigida, la relación de adyacencia no necesariamente es simétrica. Por ejemplo, en las figuras 1 y 2, el vértice 2 es adyacente al vértice 1, puesto que la arista $(1, 2)$ está en ambas gráficas. El vértice 1 no es adyacente al vértice 2 en la figura 1, ya que la arista $(2, 1)$ no está en la gráfica.

Definición 1.5 El *grado* de un vértice en una gráfica no dirigida es el número de aristas incidentes sobre él. En una gráfica dirigida, el *grado de salida* de un vértice es el número de aristas que salen de éste, y el *grado de entrada* de

un vértice es el número de aristas que entran a él. El grado de un vértice en una gráfica dirigida es el grado de salida más el grado de entrada. El grado de un vértice v_i es denotado por $deg\ v_i$.

Definición 1.6 Un *camino* de *longitud* k de un vértice u a un vértice v en una gráfica $G = (V, A)$ es una secuencia $\langle v_o, v_1, \ldots, v_k \rangle$ de vértices, tal que $u = v_o$, $v = v_k$ y $(v_{i-1}, v_i) \in A$ para $i = 1, 2, \ldots, k$. La longitud del camino es el número de aristas que hay en el camino. El camino contiene los vértices $v_o, v_1, \ldots, v_k$ y las aristas $(v_o, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$.

Si hay un camino p de u a v, decimos que v es *alcanzable desde* u vía p; lo cual escribiremos como: $u \overset{p}{\rightsquigarrow} v$, si G es dirigida.

Definición 1.7 Un camino es *simple*, si todos los vértices en el camino son distintos.

Definición 1.8 Un *subcamino* de un camino $p = \langle v_0, v_1, \ldots, v_k \rangle$ es una subsecuencia continua de vértices; esto es, para cualquier $0 \leqslant i \leqslant j \leqslant k$, la subsecuencia de vértices $\langle v_i, v_{i+1}, \ldots, v_j \rangle$ es un subcamino de p.

Definición 1.9 En una gráfica dirigida, un camino $\langle v_o, \ldots, v_k \rangle$ forma un *ciclo* si $v_o = v_k$ y el camino contiene al menos una arista, además si $v_1, \ldots, v_k$ son distintos, diremos que el ciclo es *simple*. Un lazo es un ciclo de longitud 1. Una gráfica dirigida sin lazos es *simple*. Una gráfica es *acíclica* si no contiene ciclos.

Definición 1.10 Dos caminos $\langle v_0, v_1, v_2, \ldots, v_{k-1}, v_0 \rangle$ y $\langle v'_0, v'_1, v'_2, \ldots, v'_{k-1}, v'_0 \rangle$ forman el mismo ciclo, si existe un entero j, talque $v'_i = v_{(i+j) mod k}$ para $i = 0, 1, \ldots, k-1$.

En la figura 1, $\langle 1, 2, 5, 4 \rangle$ es un camino simple de longitud 3, mientras que $\langle 2, 5, 4, 5 \rangle$ es un camino no simple. El camino $\langle 1, 2, 4, 1 \rangle$ forma el mismo ciclo que los caminos $\langle 2, 4, 1, 2 \rangle$ y $\langle 4, 1, 2, 4 \rangle$. El camino $\langle 1, 2, 4, 1 \rangle$ forma un ciclo simple, pero el ciclo $\langle 1, 2, 4, 5, 4, 1 \rangle$ no lo es. El ciclo $\langle 2, 2 \rangle$, formado por la arista $(2, 2)$ es un lazo.

Definición 1.11 En una gráfica no dirigida, un camino $\langle v_o, v_1, \ldots, v_k \rangle$ forma un ciclo si $v_o = v_k$ y $v_1, \ldots, v_k$ son distintos.

En la figura 2, el camino $\langle 1, 2, 5, 1 \rangle$ es un ciclo.

Definición 1.12 Una gráfica no dirigida es *conexa* si cada par de vértices está conectado por un camino.

Las *componentes conexas* de una gráfica son clases de equivalencia bajo la relación "es alcanzable desde".

La figura 2, tiene 3 componentes conexas: $\{1, 2, 5\}$, $\{3, 6\}$ y $\{4\}$. Cada vértice en $\{1, 2, 5\}$ es alcazable desde cualquier otro vértice en $\{1, 2, 5\}$. Una gráfica no dirigida es conexa si tiene exactamente una componente conexa, es decir, si cada vértice es alcanzable desde cualquier otro vértice.

Definición 1.13 Una gráfica dirigida es *fuertemente conexa*, si cada par de vértices u y $v \in V$ son mutuamente alcanzables, es decir, u es alcanzable desde v ($v \rightsquigarrow u$) y v es alcanzable desde u ($u \rightsquigarrow v$). En otras palabras, una gráfica dirigida es fuertemente conexa, si ésta tiene solamente una componente fuertemente conexa.

Las componentes fuertemente conexas de una gráfica son clases de equivalencia bajo la relación "son mutuamente alcanzables".

La gráfica en la figura 1, tiene 3 componentes fuertemente conexas: $\{1, 2, 4, 5\}$, $\{3\}$ y $\{6\}$. Cada par de vértices en $\{1, 2, 4, 5\}$ son mutuamente alcanzables. Los vértices $\{3, 6\}$ no forman una componente fuertemente conexa, porque el vertice 6 no es alcanzable desde el vertice 3.

Definición 1.14 Diremos que una gráfica $G' = (V', A')$ es una *subgráfica* de $G = (V, A)$, si $V' \subseteq V$ y $A' \subseteq A$.

Dado un conjunto $V' \subseteq V$, la subgráfica inducida por V' es la gráfica $G' = (V', A')$, donde

$$A' = \{(u, v) \in A : u, v \in V'\}$$

Dada una gráfica no dirigida $G = (V, A)$, la *versión dirigida* de G es la gráfica dirigida $G' = (V, A')$, donde $(u, v) \in A'$ si, y sólo si, $(u, v) \in A$. Esto es, cada arista no dirigida (u, v) en G es reemplazada en la versión dirigida por dos aristas dirigidas (u, v) y (v, u). Dada una gráfica dirigida $G = (V, A)$, la *version no dirigida* de G es una grafica no dirigida $G' = (V, A')$, donde $(u, v) \in A'$ si, y sólo si, $u \neq v$ y $(u, v) \in A$. Esto es, la versión no dirigida contiene las aristas de G "con sus direcciones removidas"y con los lazos eliminados.

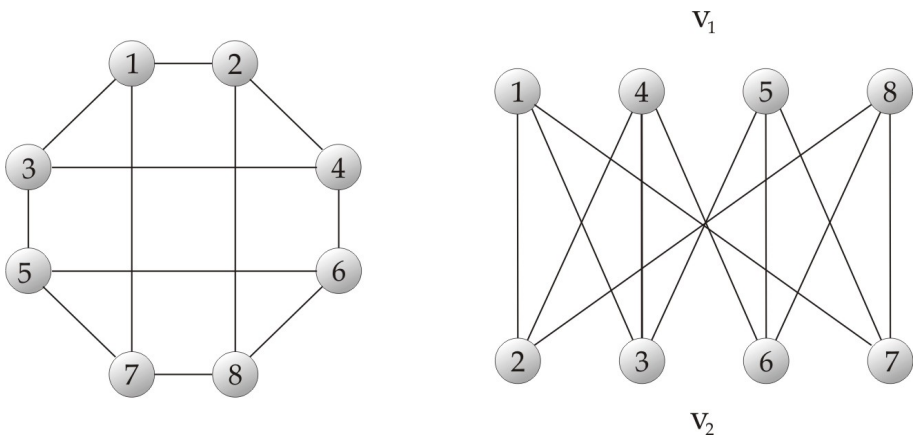


Figura 3. Una gráfica bipartita.

Definición 1.15 La *distancia* $\delta(u, v)$ entre dos vértices u y v es la longitud del camino más corto que los une, si éste existe; de otro modo $\delta(u, v) = \infty$.

Para todos los vértices u, v, w en una gráfica no dirigida conexa tenemos:

1. $\delta(u, v) \geqslant 0$ *con* $\delta(u, v) = 0 \iff u = v$
2. $\delta(u, v) = \delta(v, u)$
3. $\delta(u, v) + \delta(v, w) \geqslant \delta(u, w)$

Definición 1.16 Un vértice v de una gráfica no dirigida G, lo llamaremos *aislado* si $deg\ v = 0$ y será un vértice *final* si $deg\ v = 1$.

Definición 1.17 Una *gráfica bipartita* $G = (V, A)$, es una gráfica no dirigida en la cual V puede ser particionado en dos conjuntos V_1 y V_2, talque si $(u, v) \in A$, se tiene $u \in V_1$ y $v \in V_2$ o $u \in V_2$ y $v \in V_1$, es decir, todas las aristas de A tienen un vertice en V_1 y el otro en V_2.

En la figura 3, la gráfica de la izquierda podemos redibujarla en forma de la derecha para ver claramente que es bipartita.

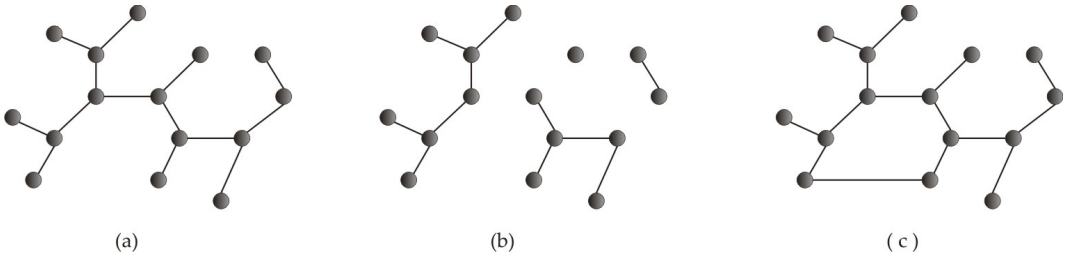


Figura 4. (a) Un árbol. **(b)** Un bosque. **(c)** Una gráfica que contiene un ciclo y por lo tanto no es un árbol ni un bosque.

Proposición 1.1 *Una gráfica es bipartita si, y sólo si, todos sus ciclos son pares.*

Demostración.
$\Rightarrow$) Si G es bipartita, entonces existe una partición $\{V_1, V_2\}$ de V tal que toda arista tenga un vértice en V_1 y el otro en V_2. Entonces para cada ciclo $\langle v_0, v_1, \ldots, v_n\rangle$ en G, podemos suponer $v_0 \in V_1$, así $v_1 \in V_2$ y $v_i \in V_1$ si, y sólo si, i es par, para $0 \leqslant i \leqslant n$, por lo tanto n debe ser par.

$\Leftarrow$) Es suficiente probar que cada componente conexa de G es bipartita, así podemos suponer que G es conexa. Tomemos cualquier vértice $v_1 \in V$ y sea V_1 que consiste de v_1 y todos los vértices a una distancia par de v_1, mientras $V_2 = V - V_1$. Se sigue que cualesquiera dos vértices de V_i no son adyacentes para $i = 1, 2$ de otro modo G contendría un ciclo impar, por lo tanto G es bipartita. ■

Definición 1.18 Un *árbol* es una gráfica no dirigida acíclica conexa. Si una gráfica no dirigida es acíclica pero posiblemente disconexa, diremos que ésta es un bosque; así las componentes de un bosque son árboles, la figura 4 ilustra esto.

Los vértices de grado 1 en un árbol son llamados *hojas*. Cada *árbol no trivial* tiene al menos dos hojas.

Teorema 1.1 *Sea $G = (V, A)$ una gráfica no dirigida. Entonces los siguientes enunciados son equivalentes.*

1. *G es un árbol.*
2. *Cada dos vértices de G están conectados por un único camino simple.*
3. *G es conexa, pero si se descarta una arista de G, el resultado es una gráfica disconexa.*
4. *G es conexa y $|A| = |V| - 1$.*
5. *G es acíclica, pero sí dos vértices no adyacentes de G son conectados por una arista, entonces la gráfica resultante tiene exactamente un ciclo.*

Demostración.
1) $\Rightarrow$ 2)
Como un árbol es conexo, cualesquiera dos vértices en G están conectados al menos por un camino simple. Supongamos que existe un par de vértices conectados por dos caminos distintos, entonces la unión de estos caminos contendrían un ciclo, contradiciendo que G es acíclica, por lo tanto dos vértices en G están conectados por un único camino simple.

2) $\Rightarrow$ 3)
Como cada dos vértices en G están conectados por un único camino simple, entonces G es conexa. Sea (u, v) una arista en A, esta arista es un camino de u a v, y es también el único camino de u a v . Si descartamos a (u, v) de G, ya no hay un camino de u a v, y entonces la gráfica resultante es disconexa.

3) $\Rightarrow$ 4)
Por hipótesis G es conexa. Observemos que G es acíclica ya que si suponemos que existe un ciclo en G, entonces cada dos vértices en el ciclo estarían conectados por dos caminos distintos, y si descartamos una arista del ciclo, G sigue siendo conexa contradiciendo la hipótesis.
Supongamos que G tiene n vértices. Si se descarta una arista de G, por hipótesis se obtiene un bosque de dos árboles. Quitando una segunda arista, resulta un bosque de tres árboles, por la misma razón; y repitiendo la operación $n - 1$ veces, se obtendrá un bosque de n árboles. Pero, por otro

lado, si se quitan todas las aristas de una vez se obtendrían también n árboles triviales: los n vértices aislados, así el total de aristas es precisamente $n-1$.

4)$\Rightarrow$ 5)
Veamos primero que G es acíclica. Supongamos que G tiene un ciclo conteniendo k vértices $v_1, v_2, \ldots, v_k$. Sea $G_k = (V_k, A_k)$ la subgráfica de G que consiste del ciclo. Note que $|V_k| = |A_k| = k$. Si $k < |V|$, existe un vértice $v_{k+1} \in V - V_k$ que es adyacente a algún vértice $v_i \in V_k$ ya que G es conexa. Definimos $G_{k+1} = (V_{k+1}, A_{k+1})$ la subgráfica de G con $V_{k+1} = V_k \cup \{v_{k+1}\}$ y $A_{k+1} = A_k \cup \{(v_i, v_{k+1})\}$. Note que $|V_{k+1}| = |A_{k+1}| = k+1$. Si $k+1 < n$, continuamos definiendo G_{k+2} de la misma manera, y así, hasta obtener $G_n = (V_n, A_n)$, donde $n = |V|$, $V_n = V$ y $|A_n| = |V_n| = |V|$. Puesto que G_n es una subgráfica de G, tenemos $A_n \subseteq A$, así $|A| \geqslant |V|$, lo cual contradice que $|A| = |V| - 1$, por lo tanto G es acíclica.

Como G es conexa y acíclica, entonces por definición G es un árbol. Sean $u, v \in V$ dos vértices no adyacentes en G, y agreguesele una nueva arista a G que conecte a u con v. Como u y v no son adyacentes, según (2) hay un único camino que los conecta, y con la nueva arista el camino se cierra apareciendo en la gráfica un único ciclo.

5) $\Rightarrow$ 1)
Supongamos que G no es conexa. Entonces existirían dos vértices v, $w \in G$ que no son mutuamente alcanzables, es decir que no hay un camino que los conecte. Sin embargo, de acuerdo con la hipótesis, si se agrega una arista con extremos en dichos vértices, aparece un ciclo en la gráfica. Eso implicaría la existencia en G de un camino que conecta a v con w, lo que es una contradicción, por lo tanto G debe ser conexa.■

Definición 1.19 Si la última arista sobre el camino de un vértice r de un árbol T a un vértice x es (y, x), entonces y es el *padre* de x y x es un *hijo* de y.

Definición 1.20 Un *árbol enraizado* tiene un vértice, distinguido de los otros; a dicho vértice se le llama *raíz* del árbol. Nos referimos a un vértice en el árbol enraizado como un *nodo*[1] del árbol.

[1]El término "nodo" es frecuentemente usado en la literatura de la teoría de gráficas como sinónimo de vértice.

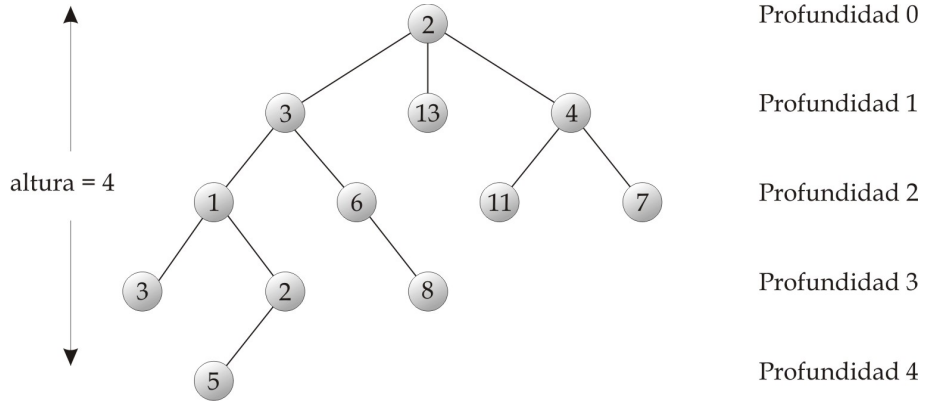


Figura 5. Un árbol enraizado con altura 4.

El vértice raíz es el único nodo en el árbol que no tiene padre. Si dos nodos tienen el mismo padre son *hermanos*. Un nodo que no tiene hijos es un nodo *externo* o una *hoja*. Un nodo no hoja es un nodo *interno*. La figura 5 muestra un árbol enraizado con 12 nodos, donde la raíz es el vértice 2.

Definición 1.21 Considere un nodo x en el árbol enraizado T con raíz r. Cada nodo y sobre el único camino de r a x es llamado *predecesor* de x. Si y es un predecesor de x, entonces x se llamará un *sucesor* de y. El *subárbol enraizado* en x es el árbol inducido por los sucesores de x y raíz en x.

Por ejemplo, el subárbol enraizado en el nodo 1 en la figura 5 contiene los nodos 1, 2, 3 y 5.

Definición 1.22 El número de hijos de un nodo x en un árbol enraizado T es llamado *grado*[2] de x. La longitud del camino de la raíz al nodo x es la *profundidad* de x en T. La profundidad más grande de los nodos en T es la *altura* de T.

[2]Notese que el grado de un nodo depende si T es considerado como un árbol enraizado o un árbol. El grado de un vértice en un árbol es, como en cualquier gráfica no dirigida, el número de aristas incidentes sobre él.

Capítulo 2

Árboles de Expansión Mínima

Consideremos la siguiente situación. Se desea crear una red de carreteras pavimentadas para interconectar un cierto número de comunidades rurales; pero debido a limitaciones presupuestales, el número de kilómetros que serán construidos debe ser el mínimo absoluto que permita la interconexión entre las diferentes comunidades.

Para interconectar un conjunto de n comunidades rurales, podemos usar un arreglo de $n-1$ carreteras, cada una conectando dos comunidades. De todos estos posibles arreglos de carreteras, el que nos va a interesar es el que tenga la menor cantidad de kilómetros.

Podemos modelar este problema de carreteras con una gráfica conexa no dirigida $G = (V, A)$, en donde V es el conjunto de comunidades rurales, A es el conjunto de posibles conexiones entre pares de comunidades, y para cada arista $(u, v) \in A$, se tiene un peso $w(u, v)$ especificando el costo (cantidad que será necesaria para construir la carretera) para conectar u y v. Entonces queremos encontrar un subconjunto acíclico $T \subseteq A$ que interconecte a todos los vértices y para el cual, el peso total

$$w(T) = \sum_{(u,v)\in T} w(u, v)$$

sea mínimo. Como T es acíclico y conecta a todos los vértices, éste debe formar un árbol, al cual llamaremos *árbol de expansión*, ya que éste se "expande"a la gráfica G. Y llamaremos al problema de determinar este árbol T

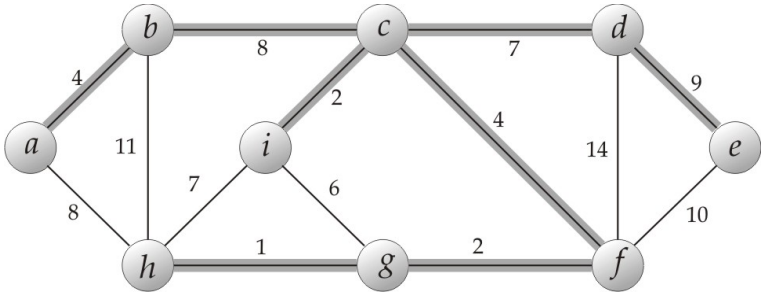


Figura 6. Un árbol de expansión mínima para una gráfica conexa. Cada arista tiene asignado un peso, las aristas en el árbol de expansión mínima son las remarcadas. El peso total del árbol es 37. El árbol no es único, si reemplazamos la arista (b, c) por la arista (a, h), tenemos otro árbol de expansión con peso 37.

como, *el problema del árbol de expansión mínima* [1]. La figura 6 muestra un ejemplo de una gráfica conexa y un árbol de expansión mínima.

La siguiente sección introduce un algoritmo "genérico"de *árbol de expansión mínima*, que hace crecer a un árbol de expansión añadiéndole una arista en cada paso.

2.1. Creciendo un árbol de expansión mínima

Definición 2.1 Diremos que una gráfica $G = (V, A)$ es una gráfica pesada, cuando cada arista de G tenga asociado un valor real, es decir existe una función de peso w tal que $w : A \to \mathbb{R}$.

Supongamos que se tiene una gráfica $G = (V, A)$ conexa no dirigida pesada, y queremos encontrar un árbol de expansión mínima para G. Los dos algoritmos que se verán en el capítulo 3 para resolver el problema del árbol de expansión mínima, usan un enfoque codicioso o avaro, aunque difieren en como se va a aplicar este enfoque en cada algoritmo.

Esta estrategia codiciosa es capturada por el siguiente algoritmo "genérico", el cual hace crecer a un árbol de expansión mínima añadiéndole una

[1]La frase "árbol de expansión mínima"es una forma corta de la frase "árbol de expansión con peso mínimo". No estamos minimizando el número de aristas en T, ya que todos los árboles de expansión tienen exactamente $|V| - 1$ aristas por el teorema 1.1.

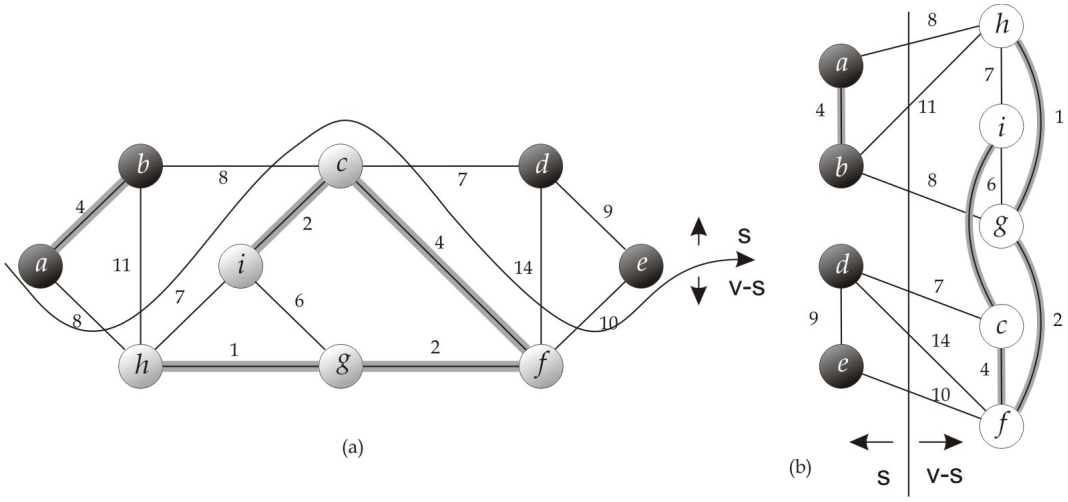


Figura 7. Dos formas de ver al corte $(S, V - S)$ en la gráfica de la figura 6. **(a)** Los vértices en el conjunto S son los vértices negros, los de $V - S$ son los grises. Las aristas que atraviesan el corte son aquellas que conectan vértices grises con vértices negros. La arista (d, c) es la única arista ligera atravesando el corte. Un subconjunto de aristas B, está compuesto por las aristas remarcadas; note que el corte $(S, V - S)$ respeta a B, ya que no hay aristas de B que atraviesen el corte. **(b)** La misma gráfica con los vértices en el conjunto S a la izquierda y los vértices en el conjunto $V - S$ a la derecha. Una arista atraviesa el corte si ésta conecta a un vértice de la izquierda con un vértice de la derecha.

arista en cada paso. El algoritmo utiliza un conjunto B, el cual es siempre un subconjunto de algún árbol de expansión mínima. En cada paso, se determina si una arista (u, v) puede ser añadida a B sin violar la invariante, en el sentido de que $B \cup \{(u, v)\}$ también es un subconjunto de un árbol de expansión mínima. A una arista (u, v), la llamaremos *arista segura* para B, siempre que ésta pueda ser añadida a B, sin destruir esta invariante, es decir, $B \cup \{(u, v)\}$ es un subconjunto de un árbol de expansión mínima.

AEM-GEN(G, w)
1 $B \leftarrow \phi$
2 **mientras** B no forme un árbol de expansión
3 encuentra una arista (u, v) que sea segura para B
4 $B \leftarrow B \cup \{(u, v)\}$
5 **regresa** B

Después de la línea 1, el conjunto B trivialmente satisface la invariante de que éste es un subconjunto de un árbol de expansión mínima. El ciclo de

las líneas 2-4 mantienen la invariante. Por lo tanto, cuando el conjunto B es regresado en la línea 5, éste ya debe ser un árbol de expansión mínima. La parte delicada, será encontrar una arista que sea segura en la línea 3. Alguna debe existir, ya que cuando la línea 3 se ejecuta, la invariante nos dice que hay un árbol de expansión T tal que $B \subseteq T$, y si hay una arista $(u, v) \in T$ talque $(u, v) \notin B$, entonces (u, v) es segura para B.

En esta sección, daremos una regla (Teorema 2.1) para reconocer aristas seguras. En el siguiente capítulo se describen dos algoritmos que usan esta regla para encontrar aristas seguras de forma eficiente.

Primero necesitamos algunas definiciones.

Definición 2.2 Un *corte* $(S, V - S)$ de una gráfica $G = (V, A)$ no dirigida, es una partición de V.

La figura 7 muestra esta noción.

Definición 2.3 Diremos que una arista $(u, v) \in A$ *atraviesa* el corte $(S, V - S)$, si uno de sus puntos finales está en S y el otro en $V - S$.

Definición 2.4 Diremos que un corte *respeta* al conjunto B de aristas, si no hay aristas de B atravesando el corte.

Definición 2.5 Una arista es una *arista ligera* atravesando un corte, si ésta tiene el menor peso de cualquier arista atravesando el corte.

Observe que puede haber más de una arista ligera atravesando un corte. En general, diremos que una arista es una *arista ligera*, satisfaciendo una propiedad dada, si ésta tiene el menor peso de cualquier otra arista satisfaciendo la propiedad. Nuestra regla para reconocer aristas seguras estará dada por el siguiente teorema.

Teorema 2.1 *Sea G=(V,A) una gráfica conexa no dirigida, con función de peso* $w : A \to \mathbb{R}$. *Sea B un subconjunto de A que está incluído en algún árbol de expansión mínima para G; sea (S,V-S) cualquier corte de G que respeta a B, y sea (u,v) una arista ligera atravesando (S,V-S). Entonces, la arista (u,v) es segura para B.*

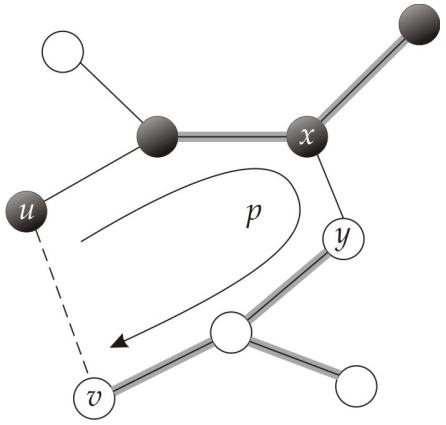


Figura 8. La prueba del teorema 2.1. Los vértices en S son los vértices negros y los de $V - S$ son los blancos. Se muestran las aristas en el árbol de expansión mínima T pero no las aristas en la gráfica G. Las aristas en B son las aristas remarcadas, y (u, v) es una arista ligera atravesando el corte $(S, V - S)$. La arista (x, y) es una arista en el único camino p de u a v en T. Un árbol de expansión mínima T' que contiene a la arista (u, v) se forma por remover la arista (x, y) de T y añadirle la arista (u, v).

Demostración.
Sea T un árbol de expansión mínima que incluya a B, y supongamos que T no contiene a la arista ligera (u, v).

Construyamos otro árbol de expansión mínima T' usando una técnica de cortar y pegar, que incluya a $B \cup \{(u, v)\}$ y probemos que (u, v) es una arista segura para B.

La arista (u, v) forma una ciclo con las aristas que están en el camino p de u a v en T, como se vé en la figura 8. Como los vértices u y v están en lados opuestos del corte $(S, V - S)$, entonces debe haber al menos otra arista en T que está sobre el camino p que también atraviesa este corte. Sea (x, y) tal arista, esta arista no está en B, porque el corte respeta a B. Como (x, y) se encuentra en el único camino de u a v en T, si la removemos, partimos a T en dos componentes, y si añadimos a la arista (u, v), reconectamos estas componentes formando un nuevo árbol de expansión $T' = T - \{(x, y)\} \cup \{(u, v)\}$.

Ahora tenemos que probar que T' es un árbol de expansión mínima.
Como la arista (u, v) es una arista ligera y además atraviesa el corte $(S, V - S)$ y la arista (x, y) también atraviesa el corte, entonces $w(u, v) \leqslant w(x, y)$. Por

lo tanto,

$$\begin{aligned} w(T') &= w(T) - w(x,y) + w(u,v) \\ &\leqslant w(T) \end{aligned}$$

Pero T es un árbol de expansión mínima, entonces también se debe cumplir que, $w(T) \leqslant w(T')$, por lo tanto T' también debe ser un árbol de expansión mínima.

Nos falta probar que la arista (u, v) actualmente es una arista segura para B. Tenemos $B \subseteq T'$, ya que $B \subseteq T$ y $(x, y) \notin B$; entonces $B \cup \{(u, v)\} \subseteq T'$, y en consecuencia, como T' es un árbol de expansión mínima, la arista (u, v) es segura para B.■

El teorema 2.1 nos muestra de una forma más clara como funciona el algoritmo AEM-GEN en una gráfica conexa $G = (V, A)$. Mientras el algoritmo está en ejecución, el conjunto B es siempre acíclico, porque de otra forma, un árbol de expansión mínima que incluya a B tendría un ciclo, lo cual es una contradicción.

En cualquier punto en la ejecución del algoritmo, la gráfica $G_B = (V, B)$ es un bosque, y cada una de las componentes conexas de G_B es un árbol (algunos árboles pueden tener un solo vértice, como en el caso, por ejemplo, cuando el algoritmo inicia: B es vacío y el bosque contiene $|V|$ árboles, uno por cada vértice). Más aún, cualquier arista segura (u, v) para B, conecta componetes distintas de G_B, ya que $B \cup \{(u, v)\}$ debe ser acíclico.

El ciclo en las líneas 2-4 del AEM-GEN se ejecuta $|V| - 1$ veces, mientras cada una de de las $|V| - 1$ aristas del árbol de expansión mínima es determinada de manera sucesiva. Inicialmente, cuando $B = \phi$, hay $|V|$ árboles en G_B, y cada iteración reduce éste número en 1. Cuando el bosque solamente contiene un único árbol, el algoritmo finaliza.

Los dos algoritmos del capítulo 3 usarán el siguiente corolario del teorema 2.1.

Corolario 2.1 *Sea G=(V,A) una gráfica conexa no dirigida pesada, con función de peso $w : A \to \mathbb{R}$. Sea B un subconjunto de A que está incluido en algún árbol de expansión mínima para G, y sea C una componente conexa (árbol) en el bosque $G_B = (V, B)$. Si (u,v) es una arista ligera conectando a C con alguna otra componente en G_B, entonces (u,v) es segura para B.*

Demostración.
Consideremos el corte $(C, V - C)$, este corte respeta a B, y (u, v) es por lo tanto una arista ligera para este corte y entonces segura para B. ■

Capítulo 3

Algoritmos de Kruskal Y Prim

En este capítulo examinaremos dos algoritmos para resolver el *problema del árbol de expansión mínima*: El Algoritmo de Kruskal y el Algoritmo de Prim. Los dos algoritmos utilizan una estrategia "codiciosa"o "avara". En cada paso de cada algoritmo, se hace una elección de varias posibles. La estrategia codiciosa defiende el hecho de que la elección es la mejor en ese momento. Este tipo de estrategia, generalmente no garantiza encontrar las soluciones óptimas globales para el problema. Sin embargo, para el problema del *árbol de expansión mínima*, veremos que esta estrategia codiciosa si produce un árbol de expansión con peso mínimo.

Los dos algoritmos son producto del algoritmo genérico del capítulo anterior. Cada uno usa una regla específica para determinar una arista segura en la línea 3 del AEM-GEN. En el algoritmo de Kruskal, el conjunto B es un bosque. La arista segura que es añadida a B, es siempre la arista con el menor peso en la gráfica que conecta a dos componentes distintas. En el algoritmo de Prim, el conjunto B forma un único árbol. La arista segura añadida a B es siempre la arista de menor peso conectando el árbol con un vértice que no está en el árbol.

3.1. Algoritmo de Kruskal

El algoritmo de Kruskal está basado directamente en el algoritmo *genérico* del *árbol de expansión mínima* del capítulo 2. Éste encuentra una arista segura (u, v) para añadirla al bosque, buscando de entre todas las aristas que conectan cualesquiera dos árboles en el bosque, una arista con el menor peso.

Sean C_1 y C_2 dos árboles que están conectados por la arista (u, v). Como (u, v) debe ser una arista ligera conectando C_1 a algún otro árbol, el corolario 2.1 implica que (u, v) es una arista segura para C_1. El algoritmo de Kruskal se dice un algoritmo codicioso, porque en cada paso, siempre se añade al bosque una arista con el menor peso posible.

La implementación del algoritmo de Kruskal, utiliza una estructura de datos de conjuntos disjuntos para mantener varios conjuntos disjuntos de elementos. Cada conjunto contiene todos los vértices de cada árbol en el bosque. La operación REP-CONJ(u) regresa un elemento representativo del conjunto que contiene a u. De esta manera, podemos determinar si dos vértices u y v pertenecen al mismo árbol, probando si REP-CONJ(u) es igual a REP-CONJ(v), la combinación de los árboles es completada con el procedimiento UNION.

```
AEM-KRUSKAL(G, w)
1  B ← φ
2  para cada vértice v ∈ V[G]
3     CREA-CONJ(v)
4  ordena las aristas de A, de forma creciente con respecto a sus pesos
5  para cada arista (u, v) ∈ A, ordenada de forma creciente
6     si REP-CONJ(u)≠ REP-CONJ(v)
7        entonces B ← B ∪ {(u, v)}
8             UNION(u, v)
9  regresa B
```

El algoritmo de Kruskal funciona como se muestra en la figura 9. Las líneas 1-3 inicializan el conjunto B como el conjunto vacío y crea $|V|$ árboles, cada uno conteniendo un vértice. En la línea 4, todas las aristas de A son ordenadas de forma creciente con respecto a sus pesos. El ciclo **para cada**

en las líneas 5-8 del algoritmo, verifica para cada arista (u, v), si sus puntos finales u y v pertenecen al mismo árbol. Si ésto ocurre, entonces la arista no puede ser añadida al bosque sin crear un ciclo, y entonces la arista es descartada. De otra forma, si los dos vértices pertenecen a árboles distintos, la arista (u, v) es añadida a B en la línea 7 y los vértices en los dos árboles son fusionados en la línea 8. El tiempo de ejecución del algoritmo de Kruskal para una gráfica $G = (V, A)$, va a depender de la implementación que se haga de la estructura de datos de conjuntos disjuntos.

3.2. Algoritmo de Prim

Como el algoritmo Kruskal, el algoritmo de Prim es un caso especial de el algoritmo *genérico del árbol de expansión mínima* del capítulo 2. El algoritmo de Prim tiene la propiedad de que las aristas en el conjunto B siempre forman un único árbol. Como se muestra en la figura 10, el árbol empieza desde un vértice raíz r arbitrario y crece hasta que el árbol se expande a todos los vértices en V. En cada paso, una arista ligera que está conectando un vértice en B con un vértice en $V - B$ es añadida al árbol.

Por el corolario 2.1, ésta regla añade solamente aristas que son seguras para B; por lo tanto, cuando el algoritmo finaliza, las aristas en B forman un árbol de expansión mínima. Esta estrategia, se dice codiciosa porque el árbol es aumentado en cada paso con una arista que contribuye con la mínima cantidad posible al peso del árbol.

La clave para la implementación el algoritmo de Prim de manera eficiente es hacer fácilmente la selección de la nueva arista que será añadida al árbol formado por las aristas de B. En el seudocódigo de adelante, la gráfica conexa G y la raíz r del árbol de expansión mínima que se hará crecer, son datos de entrada para el algoritmo. Durante la ejecución del algoritmo, todos los vértices que no están en el árbol son encolados en Q, en base al campo key.

Para cada vértice v, $key[v]$ es el peso mínimo de cualquier arista conectando el vértice v a un vértice en el árbol; por convención $key[v] = \infty$ si no existe tal arista. El campo $\pi[v]$ es llamado los "predecesores" de v en el árbol. Durante la ejecución del algoritmo, el conjunto B del AEM-GEN se

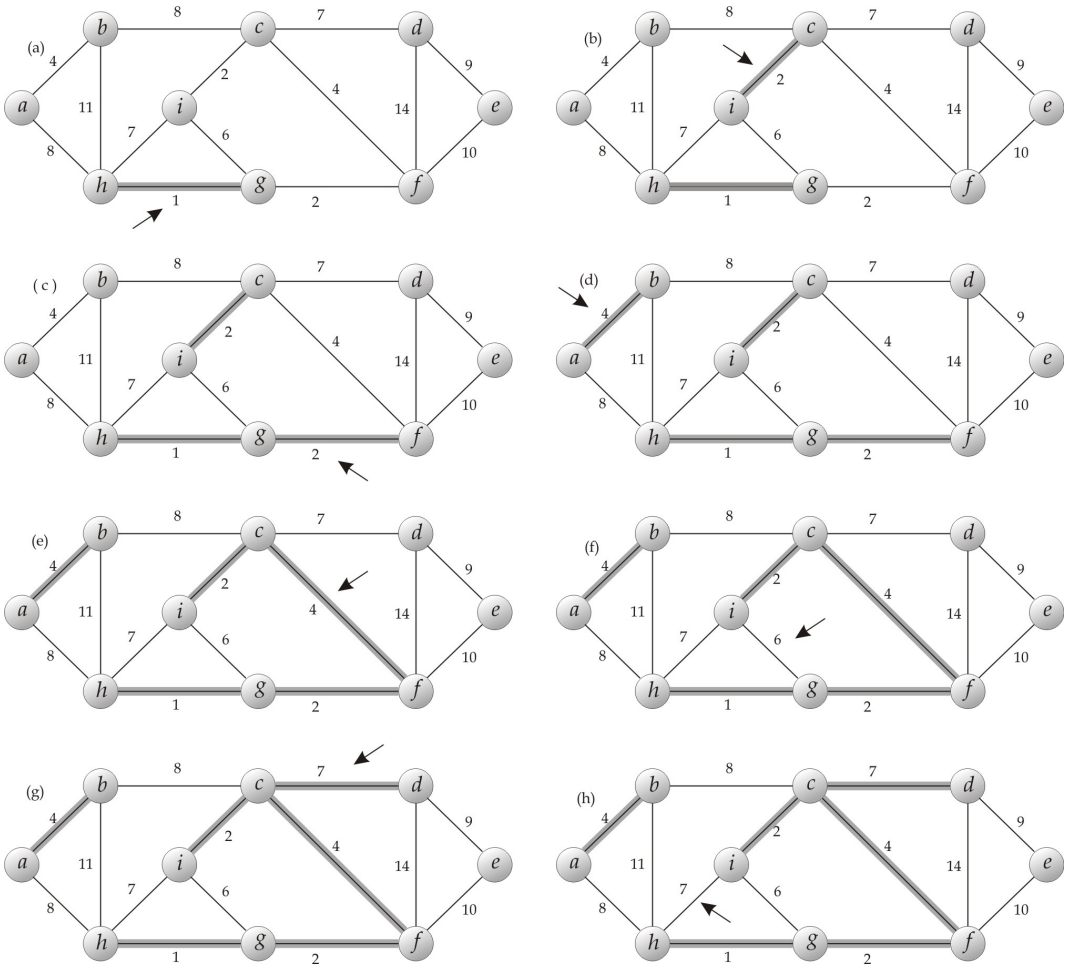
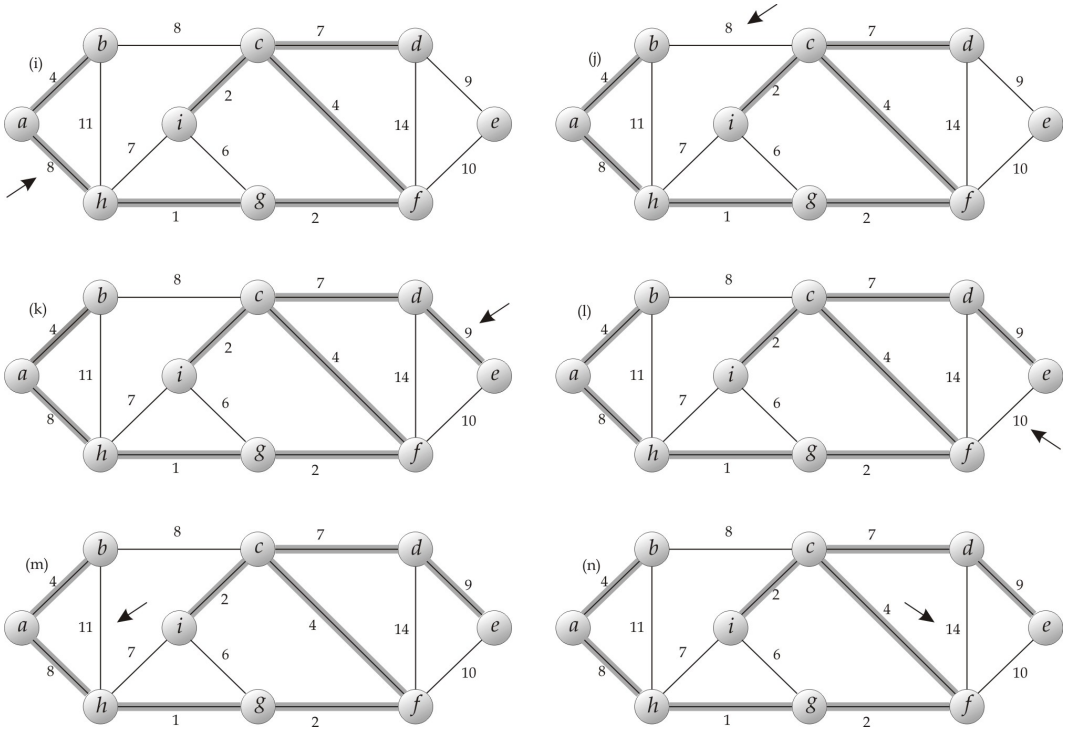


Figura 9. La ejecución del algoritmo de Kruskal en la gráfica de la figura 6. Las aristas remarcadas pertenecen al bosque B que se está haciendo crecer en la gráfica. Las aristas son ordenadas de acuerdo a sus pesos en forma creciente. Una flecha apunta hacia la arista bajo consideración en cada paso del algoritmo. Si la arista une dos árboles distintos en el bosque, ésta es añadida a el bosque, y después son fusionados los dos árboles.


(i)
(j)
(k)
(l)
(m)
(n)
a
b
c
d
e
f
g
h
i
4
8
11
8
7
7
2
4
14
9
6
10
1
2

mantiene implícitamente como

$$B = \{(v, \pi[v]) : v \in V - \{r\} - Q\}$$

Cuando el algoritmo finaliza, la cola Q está vacía; y el árbol de expansión mínima B para G es

$$B = \{(v, \pi[v]) : v \in V - \{r\}\}$$

```
AEM-PRIM(G, w, r)
1  Q ← V[G]
2  para cada vértice u ∈ Q
3      key[u] ← ∞
4  key[r] ← 0
5  π[r] ← φ
6  mientras Q ≠ φ
7      u ← EXT-MIN(Q)
8      para cada v ∈ Ady[u]
9          si v ∈ Q y w(u, v) < key[v]
10             entonces π[v] ← u
11                 key[v] ← w(u, v)
```

El algoritmo de Prim funciona como se muestra en la figura 10. Las líneas 1-4 inicializan la cola Q, que contiene a todos los vértices de G, y se fija el campo key de cada vértice a ∞, excepto para la raíz r, donde key se fija a 0. La línea 5 inicializa $\pi[r]$ como ϕ, porque la raíz r no tiene predecesor.

Durante la ejecución del algoritmo, el conjunto $V-Q$ contiene los vértices del árbol que se está haciendo crecer. La línea 7 identifica un vértice $u \in Q$ en el que incide una arista ligera atravesando el corte $(V-Q, Q)$ (con exepción de la primera iteración, en la cual $u = r$ por la línea 4). Removiendo al vértice u del conjunto Q y entonces añadíendolo al conjunto $V - Q$ de vértices del árbol. Las líneas 8-11 actualizan el campo key y el campo π de cada vértice v adyacente a u, pero no en el árbol. Esta actualización, mantiene las invariantes de que $key[v] = w(v, \pi[v])$ y que $(v, \pi[v])$ es una arista ligera conectando el vértice v a algún vértice en el árbol.

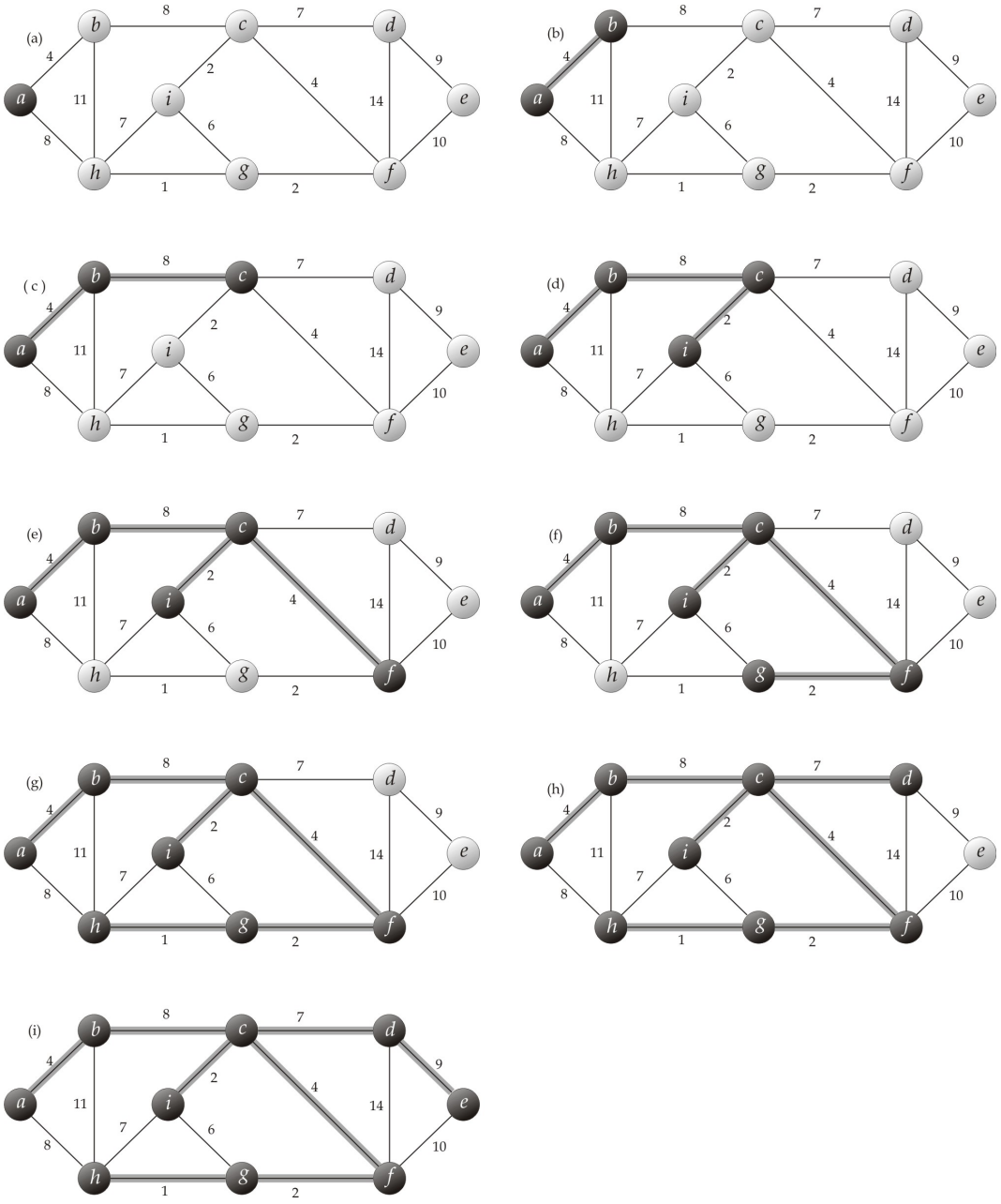


Figura 10. La ejecución del algoritmo de Prim en la gráfica de la figura 6. El vértice raíz es a. Las aristas remarcadas están en el árbol que se está haciendo crecer, y los vértices en el árbol son los vértices negros. En cada paso de el algoritmo, los vértices en el árbol determinan un corte de la gráfica, y una arista ligera atravesando el corte es añadida al árbol. En el segundo paso, por ejemplo, el algoritmo puede elegir añadir a el árbol la arista (b, c) o la arista (a, h), ya que son aristas ligeras atravesando el corte.

Capítulo 4

Caminos Cortos con Origen Fijo

Un automovilista cuenta con un mapa de carreteras y quiere encontrar la ruta más corta posible de Puerto Vallarta a Tampico. Dado el mapa de carreteras, en el cual se da la distancia entre cada par de intersecciones adyacentes, ¿Cómo podemos determinar la ruta más corta?.

Una posibilidad sería enumerar todas las rutas que hay de Puerto Vallarta a Tampico, y con las distancias asignadas a cada ruta en el mapa, seleccionar la más corta. Es fácil ver, que incluso descartando las rutas que contienen ciclos, hay cientos de posibilidades, la mayoria de las cuales simplemente no tendría sentido considerar. Por ejemplo, una ruta de Puerto Vallarta-Acapulco-Tampico sería una mala consideración porque tendría que ir hacia el sur y después hacia el norte y eso lo llevaría a recorrer cientos de kilometros fuera de su objetivo.

En este capítulo, veremos como resolver éste tipo de problemas de forma eficiente. En un *problema de caminos cortos*, donde dada una gráfica $G = (V, A)$ dirigida pesada, con función de peso $w : A \to \mathbb{R}$, mapeando aristas a pesos reales. El *peso* de un camino $p = \langle v_0, v_1, \ldots, v_k \rangle$ es la suma de los pesos de las aristas que lo componen.

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

Definición 4.1 Definimos el *peso de un camino corto* de u a v por

$$\delta_w(u,v) = \begin{cases} min\{w(p) : u \overset{p}{\rightsquigarrow} v\} & \text{si hay un camino } p \text{ de } u \text{ a } v, \\ \infty & \text{de otra forma.} \end{cases}$$

Definición 4.2 Un *camino corto* del vértice u al vértice v, se define como cualquier camino p con peso $w(p) = \delta_w(u,v)$.

En el ejemplo de Puerto Vallarta-Tampico, podemos modelar el mapa de carreteras con una gráfica dirigida: los vértices representan las intersecciones, las aristas los segmentos de carretera y el peso de las aristas representan las distancias de las carreteras. El objetivo es encontrar el camino más corto desde una intersección dada en Puerto Vallarta a una intersección dada en Tampico.

Los pesos de las aristas también pueden ser interpretadas como otro tipo de cantidades; con frecuencia se usa para representar tiempo, costos, penalidades, pérdidas o cualquier otra cantidad acumulada a lo largo del camino, la cual deseamos minimizar.

Variantes

En este capítulo nos enfocaremos en el problema de *caminos cortos con origen fijo*: Dada una gráfica $G = (V, A)$, queremos encontrar el camino más corto desde un vértice $s \in V$ origen dado a cada vértice $v \in V$. Otros problemas se podrían resolver por el algoritmo para el problema del *origen fijo*, incluyendo las siguientes variantes.

Problema de caminos cortos con destino fijo: Encontrar un camino corto a un vértice destino t dado desde cada vértice v. Si invertimos las direcciones de cada arista en la gráfica, podemos reducir este problema a un problema con *origen fijo*.

Problema del camino corto para una pareja de vértices fijos: Encontrar un camino corto de u a v para los vértices dados u y v. Si resolvemos el problema de *origen fijo*, con vértice origen u, podemos resolver este problema también.

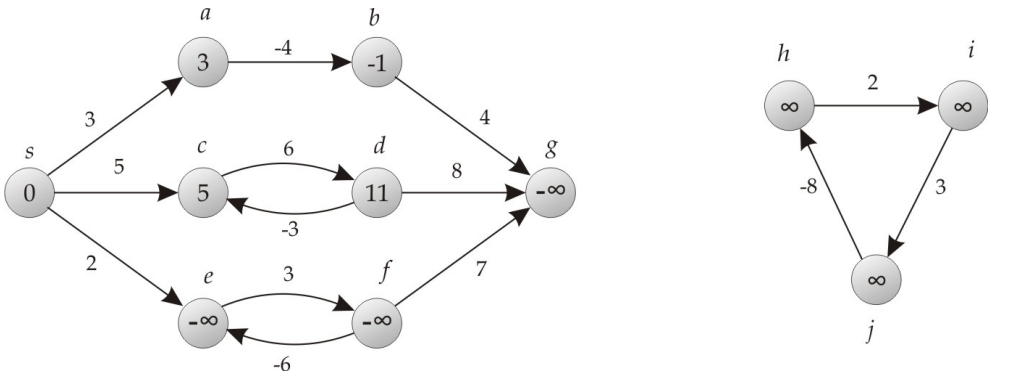


Figura 11. Pesos negativos en aristas de una gráfica dirigida. Dentro de cada vértice se muestra el peso del camino corto desde el origen s. Como los vértices e y f forman un ciclo con peso negativo alcanzable desde s, tiene pesos del camino corto de $-\infty$. Como el vértice g es alcanzable desde un vértice el cual tiene como peso del camino corto $-\infty$, éste también tiene un peso del camino corto de $-\infty$. Los vértices tales como h, i, j no son alcanzables desde s, y así sus pesos del camino corto son ∞, aunque están en un ciclo con peso negativo.

Problema de caminos cortos para cada pareja de vértices: Encontrar un camino corto desde u a v para cada par de vértices u y v. Este problema se puede resolver ejecutando un algoritmo de *origen fijo* una sola vez desde cada vértice.

Aristas con pesos negativos

En algunos casos del problema de *caminos cortos con origen fijo*, pueden haber aristas que tengan pesos negativos. Si la gráfica $G = (V, A)$ no contiene ciclos con peso negativo alcanzables desde el origen s, entonces para todo $v \in V$, el peso del camino corto $\delta_w(s, v)$ está bien definido, incluso si éste tiene un valor negativo. Si se tiene un ciclo con peso negativo alcanzable desde s, entonces el peso del camino corto no está bien definido. Si hay un ciclo con peso negativo en algún camino de s a v, definimos $\delta_w(s, v) = -\infty$.

La figura 11 muestra el efecto que tienen los pesos negativos en los pesos de un *camino corto*. Como solamente hay un camino desde s a a (el camino $\langle s, a\rangle$), $\delta_w(s, a) = w(s, a) = 3$. Similarmente, hay un único camino desde s a b, y así $\delta_w(s, b) = w(s, a) + w(a, b) = 3 + (-4) = -1$. Hay una infinidad de caminos de s a c: $\langle s, c\rangle$, $\langle s, c, d, c\rangle$, $\langle s, c, d, c, d, c\rangle$, ... Como el ciclo $\langle c, d, c\rangle$

tiene peso $6 + (-3) = 3 > 0$, el camino corto desde s a c es $\langle s, c\rangle$, con peso $\delta_w(s, c) = 5$. Similarmente el camino corto de s a d es $\langle s, c, d\rangle$, con peso $\delta_w(s, d) = w(s, c) + w(c, d) = 11$. Análogamente, hay una infinidad de caminos desde s a e; $\langle s, e\rangle$, $\langle s, e, f, e\rangle$, $\langle s, e, f, e, f, e\rangle$ y así sucesivamente. Observe que el ciclo $\langle e, f, e\rangle$ tiene peso $3+(-6) = -3 < 0$, sin embargo no hay un camino corto desde s a e. Recorriendo el ciclo con peso negativo $\langle e, f, e\rangle$ muchas veces arbitrariamente, podemos encontrar caminos de s a e con pesos arbitrariamente chicos, y así $\delta_w(s, e) = -\infty$. Como g es alcanzable desde f, también podemos encontrar caminos con pesos negativos arbitrariamente chicos de s a g, y $\delta_w(s, g) = -\infty$. Los vértices h, i y j también forman un ciclo con peso negativo. Sin embargo, no es alcanzable desde s, y así $\delta_w(s, h) = \delta_w(s, i) = \delta_w(s, j) = \infty$.

Algunos algoritmos de caminos cortos, tal como el algoritmo de Dijkstra, suponen que todos los pesos de las aristas en la gráfica de entrada son positivos, como en el ejemplo del mapa de carreteras. Otros, como el algoritmo de Bellman-Ford, permiten aristas con peso negativo en la gráfica de entrada y producen una respuesta correcta, siempre que no se tengan ciclos con peso negativo que sean alcanzables desde el origen. Si hay un ciclo con peso negativo, el algoritmo puede detectar y reportar esta existencia.

Representando Caminos Cortos

Con frecuencia, no sólo se quiere calcular los pesos de un camino corto, si no también los vértices en los caminos cortos. Dada una gráfica $G = (V, A)$, mantendremos para cada vértice $v \in V$ un *predecesor* $\pi[v]$, que también es otro vértice o el vacío. Los algoritmos de caminos cortos del capítulo 5 fijan los atributos de π y así el cambio de predecesores originan en un vértice v recorrerse hacia atrás a lo largo del camino corto de s a v. De esta forma, dado un vértice v para el cual $\pi[v] \neq \phi$, el siguiente procedimiento se usará para imprimir un camino corto de s a v.

```
IMP-CAM(G, s, v)
1  si v = s
2     entonces imprime s
3  si π[v] = φ
4     entonces imprime "No existe camino de" s "a" v
5  IMP-CAM(G, s, π[v])
6     imprime v
```

Durante la ejecución de un algoritmo de caminos cortos, los valores de π no necesariamente van a indicar los caminos cortos. Estaremos interesados en la subgráfica predecesor $G_\pi = (V_\pi, A_\pi)$ inducida por los valores de π. Definimos el conjunto de vértices V_π, como el conjunto de vértices de G con predecesores no nulos, más el origen s; es decir,

$$V_\pi = \{v \in V : \pi[v] \neq \phi\} \cup \{s\}$$

Así mismo, el conjunto de aristas dirigidas A_π es el conjunto de aristas inducidas por los valores de π para los vértices en V_π,

$$A_\pi = \{(\pi[v], v) \in A : v \in V_\pi - \{s\}\}$$

Probaremos que los valores de π producidos por los algoritmos en el capítulo 5, tienen la propiedad de que al finalizar G_π es un "*arbol de caminos cortos*", un árbol enraizado conteniendo un camino corto desde un vértice origen s a cada vértice que es alcanzable desde s. Un árbol de caminos cortos, contiene caminos cortos desde el origen, que estarán definidos en términos de los pesos de las aristas. Para ser preciso, sea $G = (V, A)$ una gráfica dirigida pesada, con función de peso $w : A \to \mathbb{R}$, y suponga que G no contiene ciclos con peso negativo alcanzables desde el vértice origen $s \in V$, de esta forma los caminos cortos están bien definidos.

Un *árbol de caminos cortos* enraizado en s, es una subgráfica dirigida $G' = (V', A')$, donde $V' \subseteq V$ y $A' \subseteq A$, tal que

1. V' es el conjunto de vértices alcanzables desde s en G,

2. G' forma un árbol enraizado con raíz s, y

3. para todo $v \in V'$, el único camino simple desde s a v en G' es un camino corto desde s a v en G.

Los caminos cortos no necesariamente son únicos, y tampoco lo son los árboles de caminos cortos. Por ejemplo, la figura 12 muestra una gráfica dirigida pesada y dos árboles de caminos cortos con la misma raíz.

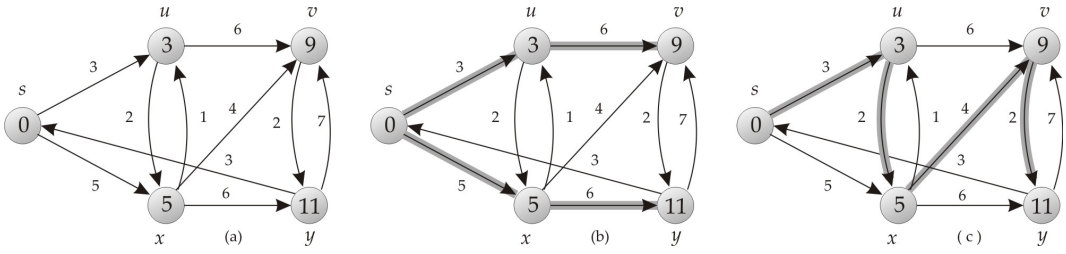


Figura 12. (a) Una gráfica dirigida pesada, con los pesos de los caminos cortos desde el origen. **(b)** Las aristas remarcadas forman un árbol de caminos cortos enraizado en el origen s. **(c)** Otro árbol de caminos cortos con la misma raíz.

4.1. Caminos cortos

Nuestro análisis va a requerir de algunas convenciones para la aritmética con infinitos. Para cualquier número real $a \neq -\infty$, tenemos $a + \infty = \infty + a = \infty$. También, como al realizar las demostraciones estaremos en presencia de ciclos con peso negativo, supondremos que para cualquier número real $a \neq \infty$, se tiene $a + (-\infty) = (-\infty) + a = -\infty$.

Para entender los algoritmos de caminos cortos con origen fijo, nos será útil entender las técnicas que utilizan y las propiedades de caminos cortos que explotan. La técnica principal usada por los algoritmos en el capítulo 5 es la de relajación, la cual veremos en la siguiente sección. Es un método que de manera repetida decrece una cota superior en el peso actual de un camino corto a cada vértice, hasta que esta cota superior es igual que el peso del camino corto. En este capítulo, veremos como funciona la relajación y probaremos varias propiedades que mantiene; el lema 4.5 será la clave para entender los algoritmos del capítulo 5.

Subestructura óptima de un camino corto

Típicamente, los algoritmos de caminos cortos explotan la propiedad de que un camino corto entre dos vértices contiene otro camino corto dentro de éste. El siguiente lema y su corolario nos plantean la propiedad de subestructura óptima de caminos cortos de manera más precisa.

Lema 4.1 (Subcaminos de caminos cortos son caminos cortos) *Dada una gráfica $G = (V, A)$ dirigida pesada, con función de peso $w : A \to \mathbb{R}$, sea $p = \langle v_1, v_2, \ldots, v_k \rangle$ un camino corto del vértice v_1 al vértice v_k, y para cualquier i y j tal que $1 \leqslant i \leqslant j \leqslant k$, sea $p_{i,j} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$ un subcamino de p del vértice v_i al vértice v_j. Entonces $p_{i,j}$ es un camino corto de v_i a v_j.*

Demostración.
Si descomponemos al camino p como $v_1 \overset{p_{1,i}}{\rightsquigarrow} v_i \overset{p_{i,j}}{\rightsquigarrow} v_j \overset{p_{j,k}}{\rightsquigarrow} v_k$, entonces tenemos $w(p) = w(p_{1,i}) + w(p_{i,j}) + w(p_{j,k})$. Supongamos que existe un camino $p'_{i,j}$ de v_i a v_j, con peso $w(p'_{i,j}) < w(p_{i,j})$. Entonces $v_1 \overset{p_{1,i}}{\rightsquigarrow} v_i \overset{p'_{i,j}}{\rightsquigarrow} v_j \overset{p_{j,k}}{\rightsquigarrow} v_k$ es un camino de v_1 a v_k, cuyo peso $w(p_{1,i}) + w(p'_{i,j}) + w(p_{j,k})$ es menor que $w(p)$, lo cual contradice el hecho de que p es un camino corto de v_1 a v_k.■

Corolario 4.1 *Sea $G = (V, A)$ una gráfica dirigida pesada, con función de peso $w : A \to \mathbb{R}$. Suponga que un camino corto p del vértice s al vértice v se puede descomponer como $s \overset{p'}{\rightsquigarrow} u \to v$ para algún vértice u y un camino p'. Entonces, el peso del camino corto de s a v es $\delta_w(s, v) = \delta_w(s, u) + w(u, v)$.*

Demostración.
Por el lema anterior, se tiene que el subcamino p' es un camino corto de el origen s al vértice u. Así que

$$\begin{aligned} \delta_w(s, v) &= w(p) \\ &= w(p') + w(u, v) \\ &= \delta_w(s, u) + w(u, v) \blacksquare \end{aligned}$$

El siguiente lema, nos da una simple, pero muy útil propiedad sobre los pesos de un camino corto.

Lema 4.2 *Sea $G = (V, A)$ una gráfica dirigida pesada, con función de peso $w : A \to \mathbb{R}$ y vértice origen s. Entonces, para toda arista $(u, v) \in A$, se tiene $\delta_w(s, v) \leqslant \delta_w(s, u) + w(u, v)$.*

Demostración.
Un camino corto p del vértice origen s al vértice v, no puede tener mayor peso que cualquier otro camino de s a v. En específico, el camino p no puede tener mayor peso que el camino particular que toma un camino corto del origen s al vértice u y que luego toma a la arista (u, v).■

4.2. Relajación

Los algoritmos del capítulo 5 usan la técnica de *relajación*[1]. Para cada vértice $v \in V$, mantendremos un atributo $d[v]$, el cual será una cota superior del peso de un camino corto desde el origen s al vértice v. Llamaremos a $d[v]$ el *camino corto estimado* al vértice v. Inicializamos los caminos cortos estimados y predecesores con el siguiente procedimiento (IOF fijará al vértice origen)

```
IOF(G, s)
1  para cada vertice v ∈ V[G]
2      d[v] ← ∞
3      π[v] ← φ
4  d[s] ← 0
```

Después de la inicialización, $\pi[v] = \phi$ para todo $v \in V$, $d[v] = 0$ para $v = s$ y $d[v] = \infty$ para todo $v \in V - \{s\}$.

El proceso de relajación de una arista (u, v), consiste en probar si se puede mejorar el camino corto a v, viendo que tan lejos se encuentra de u, y así entonces actualizar $d[v]$ y $\pi[v]$. Un paso de relajación puede o no decrecer el valor del camino corto estimado $d[v]$ y actualizar el campo $\pi[v]$ de predecesores de v. El siguiente código realiza un paso de relajación en una arista (u, v).

```
RELAJA(u, v, w)
1  si d[v] > d[u] + w(u, v)
2      entonces d[v] ← d[u] + w(u, v)
3           π[v] ← u
```

La figura 13 muestra dos ejemplos de relajación en una arista; uno en el cual el camino corto estimado decrece y otro en el cual el camino corto estimado no cambia.

[1]Puede verse extraño que el término relajación sea usado para una operación que restringe a una cota superior. El uso del término es histórico. Viene de que un paso de relajación puede ser visto como una relajación de la restricción $d[v] \leqslant d[u] + w(u, v)$, la cual por el lema 4.2, se debe satisfacer si $d[u] = \delta_w(s, u)$ y $d[v] = \delta_w(s, v)$. Esto es, si $d[v] \leqslant d[u] + w(u, v)$, no hay "prisa"para satisfacer esta restricción, así que la restricción es "relajada".

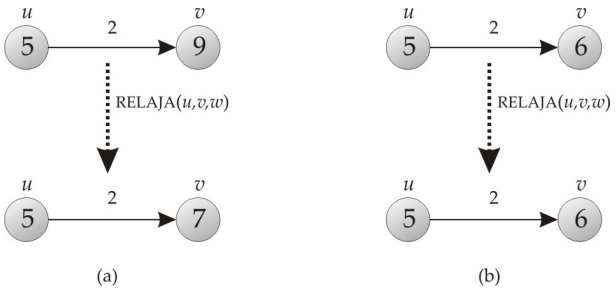


Figura 13. Relajación de una arista (u, v). El camino corto estimado de cada vértice se muestra dentro de cada vértice. (a) Como $d[v] > d[u] + w(u, v)$ antes de la relajación, el valor de $d[v]$ decrece. (b) Aquí $d[v] \leqslant d[u] + w(u, v)$ antes del paso de relajación, así $d[v]$ no cambia por la relajación.

Propiedades de la relajación

La corrección de los algoritmos del capítulo 5 van a depender de propiedades importantes de la relajación, las cuales serán resumidas en los siguientes lemas. La mayoría de los lemas describen el resultado de la ejecución de una secuencia de pasos de relajación en las aristas de una gráfica dirigida pesada, que previamente ha sido inicializada por el procedimiento IOF. Excepto por el lema 4.7, estos lemas se aplican a cualquier secuencia de pasos de relajación, aunque no producen los valores del camino corto.

Lema 4.3 *Sea $G = (V, A)$ una gráfica dirigida pesada, con función de peso $w : A \to \mathbb{R}$, y sea $(u, v) \in A$. Entonces, inmediatamente después de relajar la arista (u, v) por la ejecución de $RELAJA(u, v, w)$, se tiene $d[v] \leqslant d[u] + w(u, v)$.*

Demostración.
Si justo antes de que se relaje la arista (u, v), se tiene $d[v] > d[u] + w(u, v)$; entonces después de la relajación se tendrá $d[v] = d[u] + w(u, v)$.
Por el contrario, si antes de relajar la arista (u, v) se tiene $d[v] \leqslant d[u]+w(u, v)$; entonces los valores de $d[u]$ y $d[v]$ no cambiarán, y después de la relajación se seguirá teniendo $d[v] \leqslant d[u] + w(u, v)$. ■

Lema 4.4 *Sea $G = (V, A)$ una gráfica dirigida pesada, con función de peso $w : A \to \mathbb{R}$, sea $s \in V$ el vértice origen, suponga que la gráfica se inicializa con IOF(G,s). Entonces, $d[v] \geqslant \delta_w(s, v)$ para toda $v \in V$, y ésta invariante se mantiene sobre cualquier secuencia de pasos de relajación en las aristas de G. Más aún, una vez que se cumple $d[v] = \delta_w(s, v)$, esto nunca cambiará.*

Demostración.
La invariante $d[v] \geqslant \delta_w(s, v)$ es cierta después de la inicialización, ya que $d[s] = 0 \geqslant \delta_w(s, s)$ (note que $\delta_w(s, s) = -\infty$ si s está en un ciclo con peso negativo y 0 de otra forma) y $d[v] = \infty$, entonces $d[v] \geqslant \delta_w(s, v)$ para toda $v \in V - \{s\}$. Probaremos por contradicción que la invariante se mantiene sobre cualquier secuencia de pasos de relajación. Sea v el primer vértice para el cual un paso de relajación en la arista (u, v) provoca que $d[v] < \delta_w(s, v)$. Entonces justo después de haber relajado la arista (u, v), se tiene

$$\begin{aligned} d[u] + w(u, v) &= d[v] \\ &< \delta_w(s, v) \\ &\leqslant \delta_w(s, u) + w(u, v) \qquad (\textit{Por el lema 4.2}) \end{aligned}$$

Lo cual implica que $d[u] < \delta_w(s, u)$, pero el valor de $d[u]$ no tiene porque cambiar por la relajación de la arista (u, v), entonces ésta desigualdad también debe ser cierta justo antes de que sea relajada la arista (u, v), pero esto contradice nuestra elección de v como el primer vértice para el cual ocurre que $d[v] < \delta_w(s, u)$. Entonces concluimos que la invariante $d[v] \geqslant \delta_w(s, v)$ se mantiene para toda $v \in V$.

Para ver que los valores de $d[v]$ nunca cambian una vez que $d[v] = \delta_w(s, v)$, note que una vez que se cumple ésto, el valor de $d[v]$ no puede decrecer, por que ya hemos probado que $d[v] \geqslant \delta_w(s, v)$, y $d[v]$ tampoco puede crecer, por que los pasos de relajación no incrementan los valores de d.■

Corolario 4.2 *Suponga que en una gráfica $G = (V, A)$ dirigida pesada, con función de peso $w : A \to \mathbb{R}$, no existe camino que conecte al vértice origen s con un vértice $v \in V$ dado. Entonces, después de que la gráfica es inicializada por IOF(G,s), se tiene $d[v] = \delta_w(s, v)$, y esta igualdad se mantiene como una invariante sobre cualquier secuencia de pasos de relajación en las aristas de G.*

Demostración.
Por el lema 4.4, siempre se cumple que $d[v] \geqslant \delta_w(s, v)$ para toda $v \in V$, como no existe un camino del vértice s al vértice v, entonces $\delta_w(s, v) = \infty$, por lo tanto tenemos $d[v] = \infty = \delta_w(s, v)$■

El siguiente lema es crucial para probar la corrección de los algoritmos de caminos cortos del capítulo 5. Este lema, nos da las condiciones suficientes para que la relajación produzca que el camino corto estimado converja al peso del camino corto.

Lema 4.5 *Sea $G = (V, A)$ una gráfica dirigida pesada, con función de peso $w : A \to \mathbb{R}$, sea $s \in V$ el vértice origen, y sea $s \rightsquigarrow u \to v$ un camino corto en G para algunos vértices $u, v \in V$. Suponga que G es inicializada por IOF(G,s) y entonces una secuencia de pasos de relajación que incluyen la llamada a RELAJA(u,v,w) se ejecuta en las aristas de G. Si se tiene $d[u] = \delta_w(s, u)$ en cualquier momento antes de la llamada a RELAJA, entonces se tendrá $d[v] = \delta_w(s, v)$ en todo momento después de la llamada.*

Demostración.
Por el lema 4.4, si se tiene $d[u] = \delta_w(s, u)$ en algún punto previo a la relajación de la arista (u, v), entonces esta igualdad se mantendrá después de la relajación.
En particular, después de relajar la arista (u, v), se tendrá

$$\begin{aligned} d[v] &\leqslant d[u] + w(u, v) && (\textit{Por el lema 4.3}) \\ &= \delta_w(s, u) + w(u, v) \\ &= \delta_w(s, v) && (\textit{Por el corolario 4.1}) \end{aligned}$$

Por el lema 4.4, se tiene $\delta_w(s, v) \leqslant d[v]$, por lo tanto concluimos que $d[v] = \delta_w(s, v)$, y esta igualdad se mantiene después. ■

Árboles de caminos cortos.

Probaremos que la relajación produce que los caminos cortos estimados desciendan monótonamente hacia los actuales pesos del camino corto. También, probaremos que una vez que una secuencia de relajaciones se ha ejecutado en los pesos actuales del camino corto, la subgráfica predecesor G_π inducida por los valores resultantes de π, es un árbol de caminos cortos para G. Empezaremos con el siguiente lema, el cual muestra que la subgráfica

predecesor siempre forma un árbol enraizado en el cual la raíz es el origen.

Lema 4.6 *Sea $G = (V, A)$ una gráfica dirigida pesada, con función de peso $w : A \to \mathbb{R}$ y vértice origen $s \in V$, y suponga que G no contiene ciclos con peso negativo que son alcanzables desde s. Entonces, después de que la gráfica es inicializada por IOF(G,s), la subgráfica predecesor G_π forma un árbol enraizado con raíz s, y cualquier secuencia de pasos de relajación en las aristas de G mantiene esta propiedad como una invariante.*

Demostración.
Inicialmente, el único vértice en G_π es el vértice origen, y el lema se cumple. Consideremos una subgráfica predecesor G_π que resulta después de una secuencia de pasos de relajación en las aristas de G. Primero probemos que G_π es acíclica.
Por contradicción, supongamos que algún paso de relajación crea un ciclo en la gráfica G_π. Sea $c = \langle v_0, v_1, \ldots, v_k \rangle$ tal ciclo, donde $v_k = v_0$, entonces $\pi[v_i] = v_{i-1}$ para $i = 1, 2, \ldots, k$, y sin perdida de generalidad, podemos suponer que al relajar la arista (v_{k-1}, v_k) es cuando se crea el ciclo en G_π.

Veamos que todos los vértices en el ciclo c son alcanzables desde el origen s. En efecto, cada vértice en c tiene un predecesor no nulo, y así a cada vértice en c le fué asignado un camino corto estimado finito cuando a éste le fué asignado un valor de π no nulo. Por el lema 4.4, cada vértice en el ciclo c tiene un peso del camino corto finito, lo cual implica que es alcanzable desde s.

Ahora analicemos los caminos cortos estimados en c justo antes de llamar a RELAJA(v_{k-1},v_k, w) y probemos que c es un ciclo con peso negativo, y de esta manera contradiciendo el hecho de que G no contiene ciclos con peso negativo que son alcanzables desde el origen. Justo antes de llamar a RELAJA, tenemos $\pi[v_i] = v_{i-1}$ para $i = 1,2, \ldots, k-1$, de esta forma, para $i = 1,2, \ldots, k-1$, la última actualización que se hizo a $d[v_i]$ se dio por la asignación $d[v_i] \leftarrow d[v_{i-1}] + w(v_i, v_{i-1})$. Si el valor de $d[v_{i-1}]$ cambia, entonces éste decrece. Por lo tanto, justo antes de llamar a RELAJA(v_{k-1},v_k, w), tenemos

$$d[v_i] \geqslant d[v_{i-1}] + w(v_{i-1}, v_i) \qquad \textit{para toda } i = 1, 2, \ldots, k-1 \tag{4.1}$$

Como $\pi[v_k]$ es cambiado por la llamada de RELAJA, inmediatamente antes de la llamada también se tiene la desigualdad estricta

$$d[v_k] > d[v_{k-1}] + w(v_{k-1}, v_k)$$

Sumando esta desigualdad estricta con las $k-1$ desigualdades de 4.1, obtenemos la suma de los caminos cortos estimados a lo largo del ciclo c:

$$\begin{aligned} \sum_{i=1}^{k} d[v_i] &> \sum_{i=1}^{k} (d[v_{i-1}] + w(v_{i-1}, v_i)) \\ &= \sum_{i=1}^{k} d[v_{i-1}] + \sum_{i=1}^{k} w(v_{i-1}, v_i) \end{aligned}$$

pero

$$\sum_{i=1}^{k} d[v_i] = \sum_{i=1}^{k} d[v_{i-1}]$$

ya que cada vértice en el ciclo c aparece exactamente una vez en cada sumatoria.
Esto implica

$$0 > \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

Entonces, la suma de los pesos a lo largo del ciclo c es negativo, de esta forma hemos llegado a la contradicción deseada.

Hemos probado que G_π es una gráfica dirigida acíclica. Para probar que ésta forma un árbol enraizado con raíz r, es suficiente probar que para cada vértice $v \in V_\pi$, hay un único camino desde s a v en G_π.

Primero debemos probar que existe un camino de s a cada vértice en V_π. Los vértices en V_π son aquellos con valores de π no nulos, más s. Probaremos por inducción en el número n de relajaciones, que existe un camino de s a todo vértice en V_π.

Por definición se tiene, $V_\pi = \{v \in V : \pi[v] \neq \phi\} \cup \{s\}$ y $A_\pi = \{(\pi[v], v) \in A : v \in V_\pi - \{s\}\}$

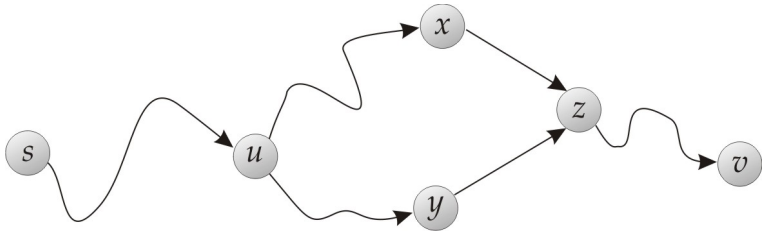


Figura 14. Un camino en G_π desde el origen s al vértice v es único. Si hay dos caminos $p_1(s \rightsquigarrow u \rightsquigarrow x \rightarrow z \rightsquigarrow v)$ y $p_2(s \rightsquigarrow u \rightsquigarrow y \rightarrow z \rightsquigarrow v)$, donde $x \neq y$, entonces $\pi[z] = x$ y $\pi[z] = y$, lo cuál es una contradicción.

Para la base, se tiene $G_\pi = (V_\pi, A_\pi) = (\{s\}, \phi)$ y entonces el resultado es obvio.

Para el paso inductivo, supongamos que después de realizar $n-1$ relajaciones, existe un camino de s a todo vertice $v \in V_\pi$. Supongamos que en la n-ésima relajación se relaja la arista (u, v), si se cumple que $d[v] \leqslant d[u] + w(u, v)$, entonces G_π no cambia y por hipótesis de inducción se tiene el resultado. Si se tiene $d[v] > d[u] + w(u, v)$, entonces la relajación de la arista (u, v) provoca que $\pi[v] = u$, luego $v \in V_\pi$ y $(\pi[v], v) = (u, v) \in A_\pi$, así que G_π sólo tiene un nuevo vértice y una nueva arista; además, el nuevo vértice es alcanzable desde s, pues por hipótesis de inducción u lo es.

Para completar la prueba del lema, ahora debemos probar que para cualquier vértice $v \in V_\pi$, hay al menos un camino de s a v en la gráfica G_π. Supongamos que hay dos caminos simples de s a algún vértice v; p_1, el cual puede descomponerse como $s \rightsquigarrow u \rightsquigarrow x \rightarrow z \rightsquigarrow v$, y p_2, el cual puede descomponerse como $s \rightsquigarrow u \rightsquigarrow y \rightarrow z \rightsquigarrow v$, donde $x \neq y$. (véase figura 14). Pero entonces, $\pi[z] = x$ y $\pi[z] = y$, lo cual implica que $x = y$. Por lo tanto existe un único camino en G_π de s a v, y así G_π forma un árbol enraizado con raíz s.■

Ahora podemos probar que, si después de que se ha ejecutado una secuencia de pasos de relajación, todos los vértices tienen asignados correctamente sus pesos de camino corto, entonces la subgráfica predecesor G_π es un árbol de caminos cortos.

Lema 4.7 *Sea $G = (V, A)$ una gráfica dirigida pesada, con función de peso $w : A \to \mathbb{R}$ y vértice origen $s \in V$, y suponga que G no contiene ciclos con peso negativo que son alcanzables desde s. Llámese a IOF(G,s) y entonces ejecute cualquier secuencia de pasos de relajación en las aristas de G, que produzcan que $d[v] = \delta_w(s, v)$ para todo $v \in V$. Entonces, la subgráfica predecesor G_π es un árbol de caminos cortos enraizado en s.*

Demostración.
Tenemos que probar que las tres propiedades para árboles de caminos cortos se cumplen para G_π. Para la primera propiedad, debemos probar que V_π es un conjunto de vértices alcanzables desde s. Por definición, el peso de un camino corto $\delta_w(s, v)$ es finito si, y sólo si, v es alcanzable desde s, es decir, los vértices que son alcanzables desde s son aquellos que tienen valores de d finitos. Pero un vértice $v \in V - \{s\}$ tiene asignado un valor finito para $d[v]$ si, y sólo si, $\pi[v] \neq \phi$. Así que, los vértices en V_π son exactamente aquellos que son alcanzables desde s.

La segunda propiedad se sigue directamente del lema 4.6.

Nos resta probar la última propiedad de árboles de caminos cortos: para toda $v \in V_\pi$ el único camino simple $s \overset{p}{\rightsquigarrow} v$ en G_π, es un camino corto de s a v en G. Sea $p = \langle v_0, v_1, \ldots, v_k \rangle$, donde $v_0 = s$ y $v_k = v$. Para $i = 1, 2, \ldots, k$, se tiene $d[v_i] = \delta_w(s, v_i)$ y ademas $d[v_i] \geqslant d[v_{i-1}] + w(v_{i-1}, v_i)$, de donde se concluye que $w(v_{i-1}, v_i) \leqslant \delta_w(s, v_i) - \delta_w(s, v_{i-1})$. Sumando los pesos a lo largo del camino p, tenemos

$$\begin{aligned} w(p) &= \sum_{i=1}^{k} w(v_{i-1}, v_i) \\ &\leqslant \sum_{i=1}^{k} (\delta_w(s, v_i) - \delta_w(s, v_{i-1})) \\ &= \delta_w(s, v_k) - \delta_w(s, v_0) \\ &= \delta_w(s, v_k) \end{aligned}$$

La tercera línea viene de la suma telescópica en la segunda línea, y la cuarta línea se sigue de que $\delta_w(s, v_0) = \delta_w(s, s) = 0$. Entonces, $w(p) \leqslant \delta_w(s, v_k)$. Como $\delta_w(s, v_k)$ es una cota mínima en los pesos de cualquier camino de s a v_k, concluimos que $w(p) = \delta_w(s, v_k)$, y así p es un camino corto de s a $v = v_k$ ■

Capítulo 5

Algoritmos de Dijkstra y Bellman-Ford

Los algoritmos de *caminos cortos con origen fijo* en este capítulo están basados en la técnica de relajación. En el capítulo anterior se proporcionaron algunas propiedades importantes de caminos cortos en general y después algunos resultados importantes. En este capítulo se verán el algoritmo de Dijkstra, el cual resuelve el problema de *caminos cortos con origen fijo* donde todas las aristas tienen pesos no negativos, y el algoritmo Bellman-Ford, el cual es usado en casos más generales, en los cuales las aristas pueden tomar valores negativos. Si la gráfica contiene un ciclo con peso negativo alcanzable desde el origen, el algoritmo Bellman-Ford detecta y reporta su existencia.

Los algoritmos de este capítulo llaman al procedimiento IOF y entonces repetidamente relajan todas las aristas. Más aún, la relajación es la única manera por la cual los caminos cortos estimados y predecesores van a cambiar. Estos dos algoritmos difieren en cuantas veces va a ser relaja cada arista y en el orden en el cual serán relajadas. En el algoritmo de Dijkstra, cada arista es relajada exactamente una vez; en el algoritmo de Bellman-Ford, cada arista es relajada varias veces.

5.1. Algoritmo Dijkstra

El algoritmo de Dijkstra resuelve el problema de caminos cortos con origen fijo en una gráfica $G = (V, A)$ dirigida pesada, para el caso en el cual todas las aristas tienen pesos no negativos. Por lo tanto, en esta sección supondremos que $w(u, v) \geqslant 0$, para toda arista $(u, v) \in A$.

El algoritmo de Dijkstra mantiene un conjunto S de vértices de quienes, los pesos finales del camino corto desde el origen s ya han sido determinados. Es decir, para todo vértice $v \in S$, se cumple que $d[v] = \delta_w(s, v)$. El algoritmo repetidamente selecciona un vértice $u \in V - S$ con el camino corto estimado más pequeño, inserta a u en S, y relaja todas las aristas que salen de u. En la siguiente implementación, se mantiene una cola Q que contiene a todos los vértices en $V - S$, clasificados de acuerdo a sus valores de d. Esta implementación asume que la gráfica G es representada por listas de adyacencia.

```
DIJKSTRA(G, w, s)
1  IOF(G, s)
2  S ← φ
3  Q ← V[G]
4  mientras Q ≠ φ
5      u ← EXT-MIN(Q)
6      S ← S ∪ {u}
7      para cada vértice v ∈ Ady[u]
8          RELAJA(u, v, w)
```

El algoritmo de Dijkstra relaja las aristas como se muestra en la figura 15. La línea 1 realiza la inicialización usual de los valores de π y d, y la línea 2 inicializa el conjunto S como el conjunto vacío. La línea 3 inicializa la cola Q, que contiene a todos los vértices en $V - S = V - \phi = V$. Cada vez que se ejecuta el ciclo **mientras** de las líneas 4-8, un vértice u se extrae de $Q = V - S$ y se inserta en el conjunto S. (La primera vez que se ejecuta el ciclo, $u = s$) Por lo tanto, el vértice u tiene el camino corto estimado más pequeño de cualquier vértice en $V - S$. Después, las líneas 7-8 relajan cada arista (u, v) que sale de u, y entonces se actualiza el camino corto estimado $d[v]$ y el predecesor $\pi[v]$ si el camino corto a v puede ser mejorado al pasar por u. Note que los vértices nunca son insertados en Q después de la línea 3

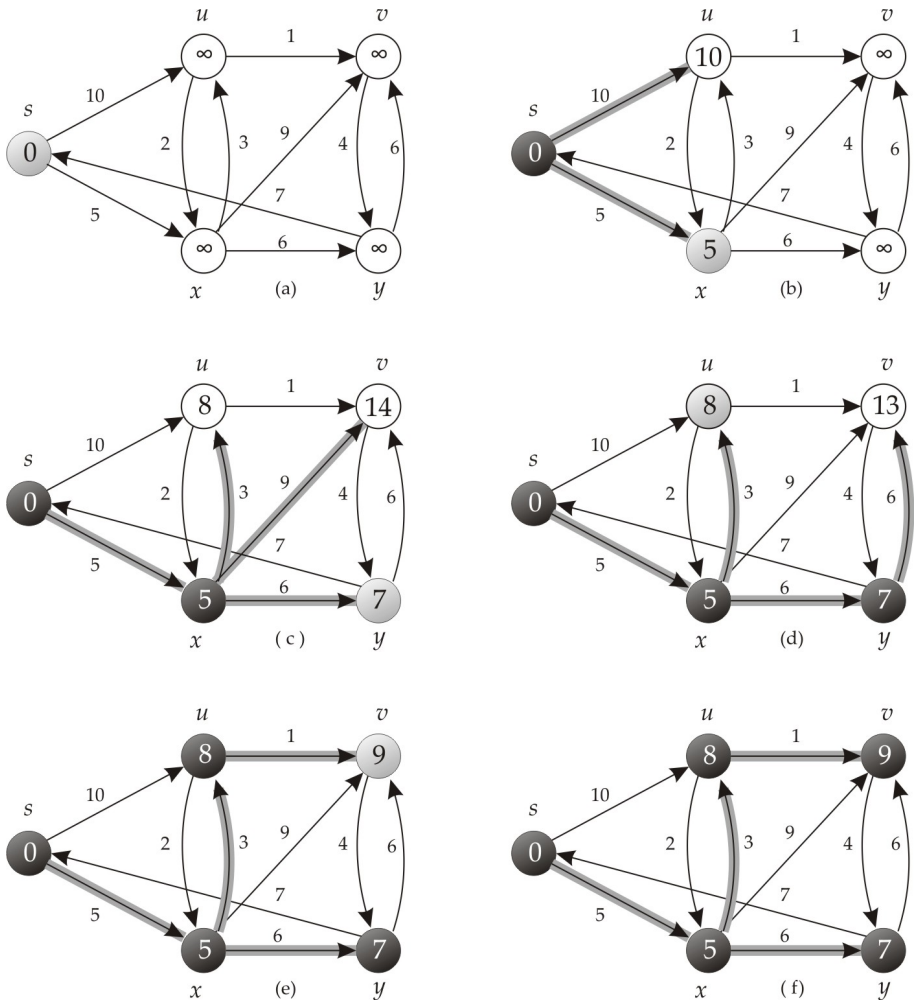


Figura 15. La ejecución del algoritmo de Dijkstra. El origen es el vértice s. Los caminos cortos estimados se muestran dentro de cada vértice, y las aristas remarcadas indican los valores de los predecesores: si la arista (u, v) está remarcada, entonces $\pi[v] = u$. Los vértices negros están en el conjunto S, y los vértices blancos están en la cola $Q = V - S$. (a) Justo antes de la primera iteración de el ciclo **mientras** de las líneas 4-8. El vértice gris tiene el valor mas pequeño de d y se elige como el vértice u en la línea 5. (b)-(f) La situación después de cada iteración sucesiva de el ciclo **mientras**. Los vértices grises en cada parte son elegidos como el vértice u en la línea 5 de la siguiente iteración. Los valores de d y π en la parte (f) son los valores finales.

y que cada vértice es extraído de Q e insertado en S exactamente una vez, así que el ciclo **mientras** de las líneas 4-8 itera exactamente $|V|$ veces.

Como el algoritmo de Dijkstra siempre selecciona el vértice más cercano o barato en $V - S$ para insertarlo en el conjunto S, se dice que éste usa una estrategia "codiciosa". Las estrategias codiciosas en general no siempre producen resultados óptimos; pero en el siguiente teorema y su corolario, se prueba que el algoritmo de Dijkstra realmente produce caminos cortos. La clave es probar que cada vez que un vértice u es insertado en el conjunto S, se tiene $d[u] = \delta_w(s, u)$.

Teorema 5.1 (Corrección del algoritmo de Dijkstra) *Si se ejecuta el algoritmo de Dijkstra en una gráfica dirigida pesada $G = (V, A)$, con función de peso no negativa w y origen s, entonces al finalizar, $d[u] = \delta_w(s, u)$ para todo vértice $u \in V$.*

Demostración.
Debemos probar que para cada vértice $u \in V$, se tiene $d[u] = \delta_w(s, u)$ en el momento en que u es insertado en el conjunto S, y que esta igualdad se mantiene después.
Por contradicción, sea u el primer vértice para el cual ocurre que $d[u] \neq \delta_w(s, u)$ cuando éste es insertado en el conjunto S. Nos centraremos en el momento en que inicia la iteración del ciclo **mientras** del algoritmo, en el cual u es insertado en S. Se debe tener que $u \neq s$, por que s es el primer vértice insertado en el conjunto S y en este momento $d[s] = \delta_w(s, s) = 0$. Como $u \neq s$, también se tiene que $S \neq \phi$, justo antes de que u es insertado en el conjunto S. Tiene que haber algún camino de s a u, porque de otra forma, por el corolario 4.2 se tendría $d[u] = \delta_w(s, u) = \infty$, lo cual contradice el hecho de que $d[u] \neq \delta_w(s, u)$.

Como existe al menos un camino, hay un camino corto p de s a u. El camino p conecta un vértice en S, llamado s, a un vértice en $V - S$, llamado u.

Sea y el primer vértice a lo largo del camino p talque $y \in V - S$, y sea $x \in V$ el predecesor de y. Como se muestra en la figura 16, el camino p se puede descomponer como $s \overset{p_1}{\rightsquigarrow} x \rightarrow y \overset{p_2}{\rightsquigarrow} u$.

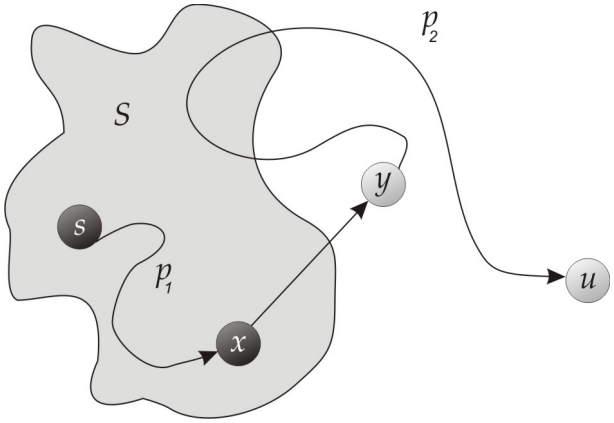


Figura 16. La demostración del teorema 5.1. El conjunto S es no vacío, justo antes de que el vértice u es insertado en él. Un camino corto p desde el origen s a el vértice u, se puede descomponer como $s \overset{p_1}{\rightsquigarrow} x \rightarrow y \overset{p_2}{\rightsquigarrow} u$, donde y es el primer vértice en el camino que no está en $V - S$ y $x \in S$ inmediatamente precede a y. Los vértices x y y son distintos, pero se puede tener $s = x$ o $y = u$. El camino p_2 puede o no volver a entrar al conjunto S.

Afirmamos que $d[y] = \delta_w(s, y)$ cuando u es insertado en S. Para probar ésto, notemos que $x \in S$. Entonces, como u es escogido como el primer vértice para el cual $d[u] \neq \delta_w(s, u)$ cuando éste es insertado en S, se tiene $d[x] = \delta_w(s, x)$ cuando x es insertado en S. La arista (x, y) es relajada en este momento, y así por el lema 4.5 se tiene la afirmación.

Ahora podemos obtener la contradicción para probar el teorema. Como y se encuentra antes de u en un camino corto de s a u y todos los pesos de las aristas son positivos (especialmente aquellos en el camino p_2), se tiene $\delta_w(s, y) \leqslant \delta_w(s, u)$, y así

$$\begin{aligned} d[y] &= \delta_w(s, y) \\ &\leqslant \delta_w(s, u) \\ &\leqslant d[u] \qquad (\textit{Por el lema 4.4}) \end{aligned} \tag{5.1}$$

pero como los vértices u y y están en $V - S$ cuando u es escogido en la línea 5 del algoritmo, tenemos $d[u] \leqslant d[y]$. Así, las dos desigualdades en 5.1 en realidad son igualdades, dándonos

$$d[y] = \delta_w(s, y) = \delta_w(s, u) = d[u]$$

En consecuencia, $d[u] = \delta_w(s, u)$, lo cual contradice nuestra elección de u como el primer vertice para el cual ocurre que $d[u] \neq \delta_w(s, u)$.
Concluimos que en el momento en que cada vértice $u \in V$ es insertado en el conjunto S, se tiene $d[u] = \delta_w(s, u)$, y por el lema 4.4, esta igualdad se mantiene después.■

Corolario 5.1 *Si se ejecuta el algoritmo de Dijkstra en una gráfica $G = (V, A)$ dirigida pesada, con función de peso no negativa w y origen s, entonces al finalizar, la subgráfica predecesor G_π es un árbol de caminos cortos enraizado en s.*

Demostración.
Por el teorema 5.1, al finalizar el algoritmo de Dijkstra se tiene que $d[u] = \delta_w(s, u)$ para todo $u \in V$, y por el lema 4.7, la subgráfica predecesor G_π es un árbol de caminos cortos enraizado en s.■

5.2. Algoritmo Bellman-Ford

El algoritmo Bellman-Ford resuelve el problema de caminos cortos con origen fijo en el caso más general, en el cual los pesos de las aristas pueden tener valores negativos. Dada una gráfica dirigida pesada $G = (V, A)$, con origen s y función de peso $w : A \to \mathbb{R}$, el algoritmo Bellman-Ford regresa un valor booleano indicando si hay o no un ciclo con peso negativo que es alcanzable desde el origen. Si hay tal ciclo, el algoritmo indica que no existe solución. De otra forma, el algoritmo produce los caminos cortos y sus pesos.

Tal como el algoritmo de Dijkstra, el algoritmo Bellman-Ford usa la técnica de relajación, de forma progresiva se hace decrecer el estimado $d[v]$ en el peso de un camino corto desde el origen s a cada vértice $v \in V$, hasta que éste alcance el peso del camino corto, *i.e.*, $d[v] = \delta_w(s, v)$. El algoritmo regresa VERDADERO si, y sólo si, la gráfica no contiene ciclos con peso negativo que son alcanzables desde el origen.

```
BELLMAN-FORD(G, w, s)
1  IOF(G, s)
2  para i ← 1 hasta |V[G]| − 1
3     para cada arista (u, v) ∈ A[G]
4        RELAJA(u, v, w)
5  para cada arista (u, v) ∈ A[G]
6        si d[v] > d[u] + w(u, v)
7           entonces regresa FALSO
8  regresa VERDADERO
```

La figura 17 muestra como funciona el algoritmo Bellman-Ford al ejecutarlo en una gráfica con 5 vértices. Después de realizar la inicialización usual, el algoritmo realiza $|V| - 1$ pasadas sobre las aristas de la gráfica. Cada pasada es una iteración del ciclo **para-hasta** de las líneas 2-4 y consiste de la relajación de cada arista de la gráfica una vez. Las figuras 19(b)-(e) muestran el estado del algoritmo después de cada una de las cuatro pasadas sobre las aristas de G. Después de realizar $|V| - 1$ pasadas, las líneas 5-8 verifican si hay o no un ciclo con peso negativo y regresa el valor booleano apropiado.

Para probar la corrección del algoritmo de Bellman-Ford, empezaremos por probar que si no hay ciclos con peso negativo, el algoritmo calcula correctamente los pesos del camino corto para todos los vértices que son alcanzables desde el origen. La prueba del siguiente lema contiene la intuición detrás del algoritmo.

Lema 5.1 *Sea $G = (V, A)$, una gráfica dirigida pesada, con función de peso $w : A \to \mathbb{R}$, y suponga que G no contiene ciclos con peso negativo que son alcanzables desde el origen s. Entonces, al finalizar BELLMAN-FORD, se tiene $d[v] = \delta_w(s, v)$ para todo vértice v que es alcanzable desde s.*

Demostración.
Sea v un vértice alcanzable desde s, y sea $p = \langle v_0, v_1, \ldots, v_k \rangle$ un camino corto de s a v, donde $v_0 = s$ y $v_k = v$. El camino p es simple, y así $k \leqslant |V| - 1$. Queremos probar por inducción que para $i = 0, 1, \ldots, k$, se tiene $d[v_i] = \delta_w(s, v_i)$ después de la i-ésima pasada sobre las aristas de G y que esta igualdad se mantiene después, y como se realizan $|V| - 1$ pasadas, esto es suficiente para probar el lema.

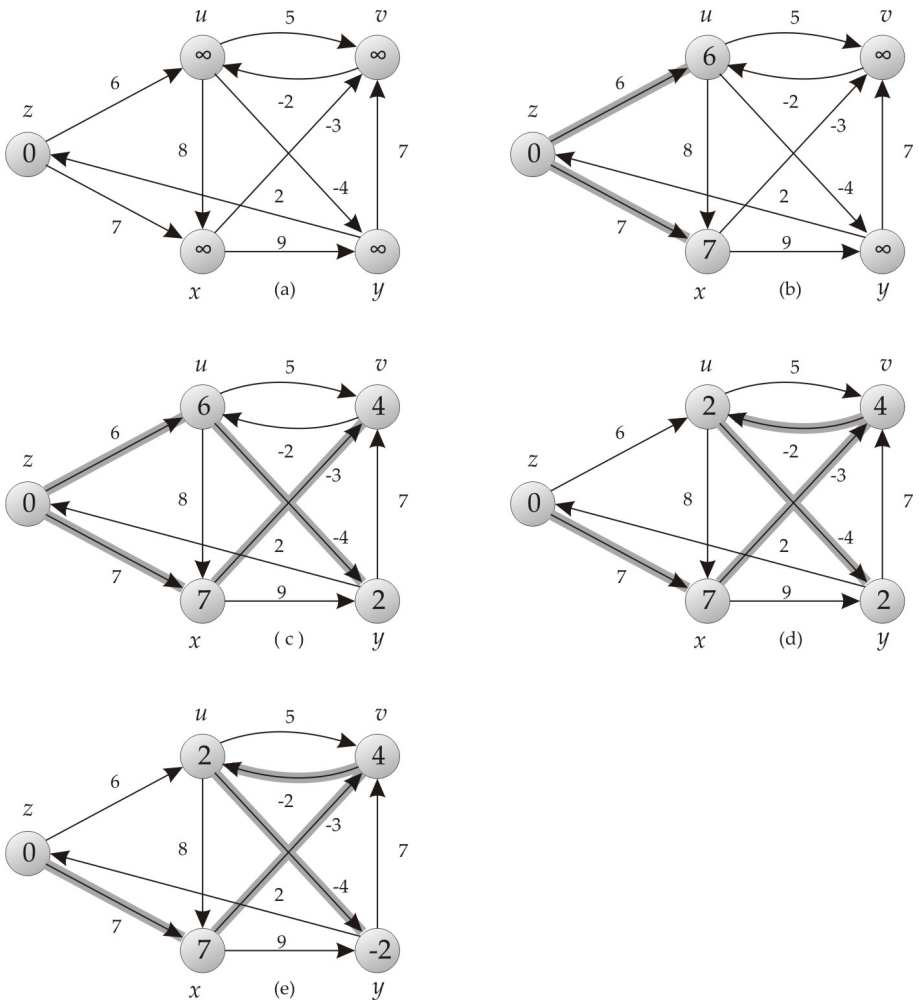


Figura 17. La ejecución del algoritmo Bellman-Ford. El origen es el vértice z. Los valores de d se muestran dentro de los vértices, y las aristas sombreadas indican los valores π. En este ejemplo particular, cada pasada relaja las aristas en orden lexicográfico: $(u, v), (u, x), (u, y), (v, u), (x, v), (x, y), (y, v), (y, z), (z, u), (z, x)$. (a) Justo antes de la primera pasada sobre las aristas. (b)-(e) La situación después de cada pasada sucesiva sobre las aristas . Los valores de d y π en la parte (e) son los valores finales. El algoritmo Bellman-Ford regresa VERDADERO en este ejemplo.

Para la base, tenemos $d[v_0] = \delta_w(s, v_0) = 0$ después de la inicialización, y por el lema 4.4, esta igualdad se mantiene después.

Para el paso inductivo, supongamos que $d[v_{i-1}] = \delta_w(s, v_{i-1})$ después de la $(i-1)$-ésima pasada. Después la arista (v_{i-1}, v_i) es relajada en la i-ésima pasada, y así por el lema 4.5, concluímos que $d[v_i] = \delta_w(s, v_i)$ después de la i-ésima pasada y todas las subsecuentes veces, y de esta forma queda probado el lema.■

Corolario 5.2 *Sea $G = (V, A)$, una gráfica dirigida pesada, con origen s y función de peso $w : A \to \mathbb{R}$. Entonces para cada vértice $v \in V$, hay un camino desde s a v si, y sólo si, BELLMAN-FORD finaliza con $d[v] < \infty$, cuando éste se ejecuta en G.*

Demostración.
La demostración es similar a la del lema 5.1.

Sea v un vértice alcanzable desde s, y sea $p = \langle v_0, v_1, \ldots, v_k \rangle$ un camino corto de s a v, donde $v_0 = s$ y $v_k = v$. El camino p es simple, y así $k \leqslant |V|-1$. Probaremos por inducción que para $i = 0, 1, \ldots, k$, se tiene $d[v_i] < \infty$ después de la i-ésima pasada sobre las aristas de G y que esta desigualdad se mantiene después, y como se realizan $|V| - 1$ pasadas, esto es suficiente para probar el lema.

Para la base, tenemos $d[v_0] = \delta_w(s, v_0) = 0 < \infty$ después de la inicialización, y por el lema 4.4, esto se mantiene después.

Para el paso inductivo, supongamos que $d[v_{i-1}] < \infty$, después de la $(i-1)$-ésima pasada. Después la arista (v_{i-1}, v_i) es relajada en la i-ésima pasada, y tenemos $d[v_i] \leqslant d[v_{i-1}] + w(v_{i-1}, v_i)$, por hipótesis $d[v_{i-1}] < \infty$ y además para toda arista $w(u, v) < \infty$. En consecuencia, $d[v_i] < \infty$ después de la i-ésima pasada y las subsecuentes veces.■

Teorema 5.2 (Corrección del algoritmo Bellman-Ford) *Suponga que se ejecuta Bellman-Ford, en una gráfica $G = (V, A)$ dirigida pesada, con función de peso $w : A \to \mathbb{R}$ y origen s. Si G no contiene ciclos con peso negativo que son alcanzables desde s, entonces el algoritmo regresa VERDADERO, se tiene $d[v] = \delta_w(s, v)$ para todo vértice $v \in V$, y la subgráfica predecesor G_π es un árbol de caminos cortos enraizado en s. Si G contiene un ciclo con peso negativo alcanzable desde s, entonces el algoritmo regresa FALSO.*

Demostración.
Supongamos que la gráfica G no contiene ciclos con peso negativo que son alcanzables desde el origen s. Primero probaremos que al finalizar se tiene, $d[v] = \delta_w(s, v)$ para todo vértice $v \in V$. Si el vértice v es alcanzable desde s, entonces por el lema 5.1, esto queda probado. Si v no es alcanzable desde s, entonces por el corolario 4.2, también se prueba que $d[v] = \delta_w(s, v)$, de esta forma hemos probado que al finalizar, se tiene $d[v] = \delta_w(s, v)$.

Ahora usaremos esto para probar que BELLMAN-FORD regresa VERDADERO. Al finalizar, tenemos para toda arista $(u, v) \in A$,

$$\begin{aligned} d[v] &= \delta_w(s, v) \\ &\leqslant \delta_w(s, u) + w(u, v) \qquad (\textit{Por el lema 4.2}) \\ &= d[u] + w(u, v) \end{aligned}$$

y así ninguna de las comparaciones en la línea 6 provocan que BELLMAN-FORD regrese FALSO; es decir, regresa VERDADERO.

Recíprocamente, supongamos que la gráfica G contiene un ciclo con peso negativo $c = \langle v_0, v_1, \ldots, v_k \rangle$, con $v_0 = v_k$, que es alcanzable desde el vértice origen s. Así, se tiene

$$\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0 \tag{5.2}$$

Por contradicción, supongamos que BELLMAN-FORD regresa VERDADERO, entonces se tiene $d[v_i] \leqslant d[v_{i-1}] + w(v_{i-1}, v_i)$ para $i = 1, 2, \ldots, k$. Sumando las desigualdades a lo largo del ciclo c tenemos

$$\sum_{i=1}^{k} d[v_i] \leqslant \sum_{i=1}^{k} d[v_{i-1}] + \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

al igual que en la demostración del lema 4.6, cada vértice en c aparece exactamente una vez en cada una de las primeras dos sumatorias. Así que,

$$\sum_{i=1}^{k} d[v_i] = \sum_{i=1}^{k} d[v_{i-1}]$$

Más aún, por el corolario 5.2, $d[v_i]$ es finito para $i = 1, 2, \ldots, k$. Entonces, tenemos

$$0 \leqslant \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

lo cual contradice la desigualdad 5.2. Por lo tanto, concluimos que el algoritmo BELLMAN-FORD regresa VERDADERO si la gráfica G no contiene ciclos con peso negativo alcanzables desde el origen, y FALSO de otra forma.■

Capítulo 6

Ejemplos

6.1. Conexión telefónica y ruta para tranvías

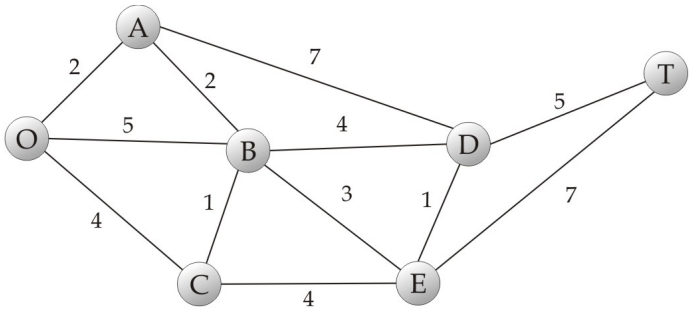
En fecha reciente se reservó el área de SEERVADA PARK para paseos y campamentos. No se permite la entrada de automóviles pero existe un sistema de caminos angostos con curvas para tranvías y Jeeps conducidos solamente por los guardabosques. La figura 18 muestra este sistema de caminos (sin las curvas), en donde O es la entrada al parque; las demás letras representan la localización de las casetas de los guardabosques y otras instalaciones de servicios. Los números son las distancias en millas de estos caminos sinuosos.

El parque contiene un mirador a un hermoso paisaje en la estación T. Unos cuantos tranvías transportan a los visitantes desde la entrada a la estación T y de regreso.

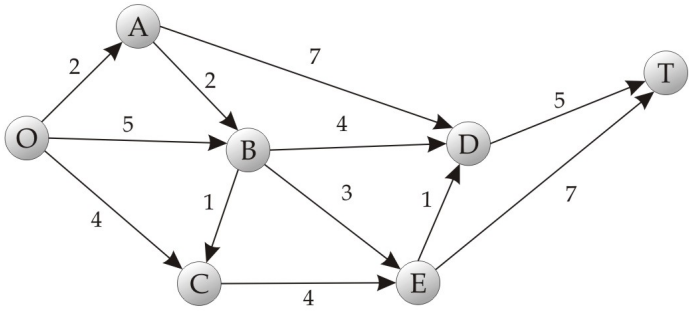
En este momento el administrador del parque se enfrenta a dos problemas. Uno consiste en que se deben de instalar líneas telefónicas subterráneas para establecer comunicación entre todas las estaciones (inclusive la entrada). Como la instalación es costosa y además perturba la ecología, se instalarán líneas que siguen sólo los caminos necesarios para obtener comunicación entre cualquier par de estaciones. La pregunta es por donde deben tenderse las líneas para lograr esto, con el mínimo número total de millas de cable instalado.

El segundo problema reside en determinar qué ruta desde la entrada del parque a la estacion T, es la que tiene la distancia total más corta para la operación de los tranvías.

Para este ejemplo aplicaremos primero los algoritmos de Kruskal y Prim para resolver la primera parte sobre las líneas telefónicas, y después aplicaremos los algoritmos de Dijkstra y Bellman-Ford para resolver la segunda parte de la ruta más corta.

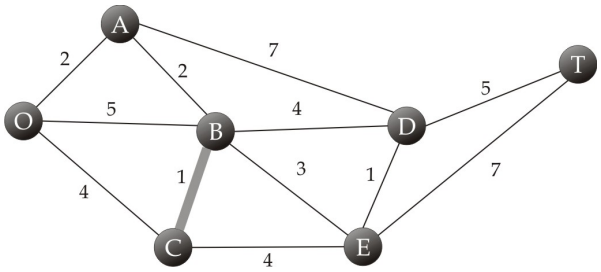


Gráfica para el problema de las líneas telefónicas.

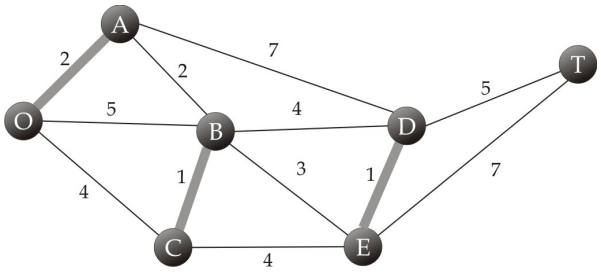


Gráfica para el problema de la operación de los tranvías.

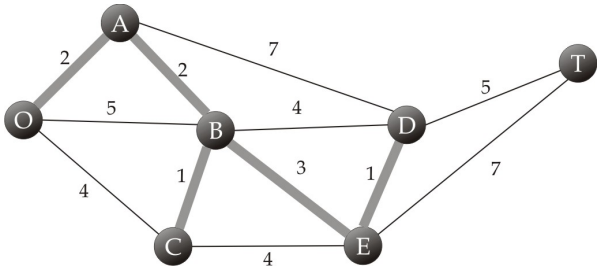
Aplicando el algoritmo de Kruskal para el problema del tendido de las líneas telefónicas en SEERVADA PARK.



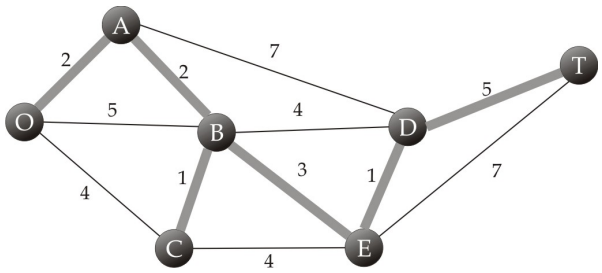
Se crea el bosque, en el cual cada árbol tiene un sólo vértice. Se ordenan las aristas de forma creciente. Se puede elegir la arista ligera (B,C) o (D,E); se eligió a (B,C), y como unía árboles distintos, entonces se añadió a B y se fusionaron los árboles.



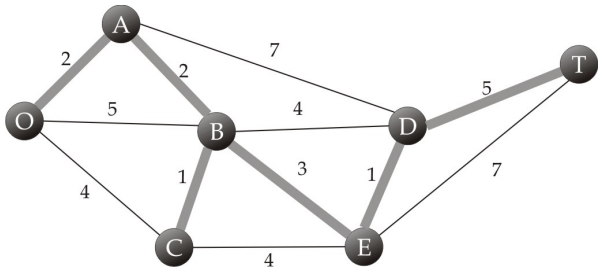
Se eligió a la arista ligera (D,E); como unía árboles distintos, se añadió a B y se fusionaron los árboles. Después, se tenían dos posibilidades, la arista (O,A) o (A,B). Se escogió la (O,A), como unía árboles distintos, se añadió a B, y se fusionaron los árboles.



Elegimos la arista ligera (A,B); como unía árboles distintos, se añadió a B y se fusionaron los árboles. Después se consideró la arista (B,E), como unía árboles distintos, se añadió a B y se fusionaron los árboles.



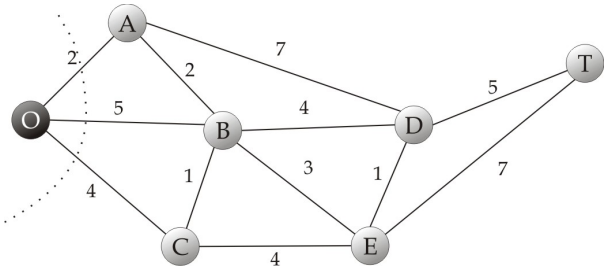
Se tenían tres aristas ligeras: (O,C), (C,E) y (B,D); pero todas unían al mismo árbol y fueron descartadas. Después, se tenían las aristas ligeras (O,B) y (D,T); pero la arista (O,B) unía al mismo árbol y fue descartada; la arista (D,T) unía árboles distintos, se añadió a B y se fusionaron los árboles.



Después se tenían dos posibilidades, la (A,D) y (E,T), pero unían al mismo árbol y entonces las descartamos. Como ya no se tienen más aristas, el algoritmo finaliza y el árbol de expansión mínima queda de ésta forma.

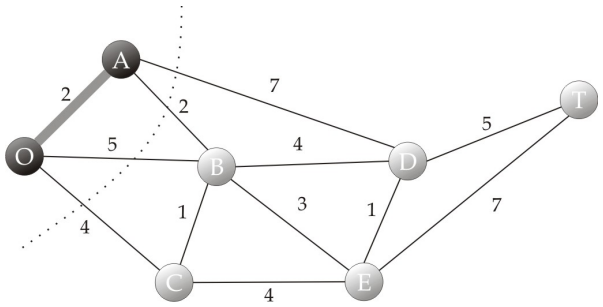
Entonces, para hacer la instalación de las líneas telefónicas para conectar todas las terminales, con un número total mínimo de cable instalado, debemos conectar la entrada con la estación A, la estación A con la B, la B con las estaciones C y E, la estación E con la D, y finalmente la estación D con la estación T; de esta forma queda resuelto el problema aplicando el algoritmo de Kruskal.

Aplicando el algoritmo de Prim para el problema del tendido de las líneas telefónicas en SEERVADA PARK.



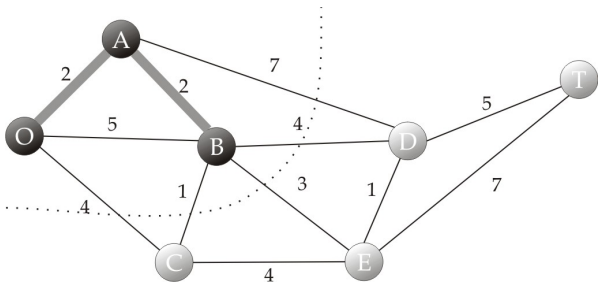
Tenemos, Q={O, A, B, C, D, E, T}, $\forall$ u$\in$Q key[u]=∞ y key[O]=0, π[O]=ϕ. En la primera iteración, quitamos a O de Q y para todo vértice adyacente v$\in$Q, verificamos si w(u,v)<key[v] y reajustamos valores.

π[A]=O, key[A]=2
π[B]=O, key[B]=5
π[C]=O, key[C]=4



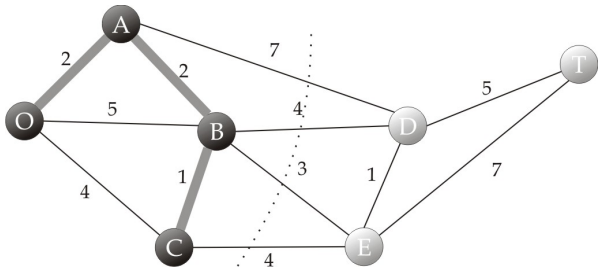
$Q \neq \phi$, en el vértice A incide la arista ligera (O,A), quitamos a A de Q y para cada vértice adyacente v$\in$Q, verificamos si w(u,v)<key[v] y reajustamos.

π[B]=A, key[B]=2
π[D]=A, key[D]=7

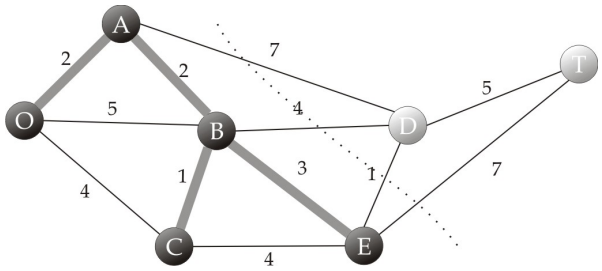


$Q \neq \phi$, en el vértice B incide la arista ligera (A,B), quitamos a B de Q y para todo vértice adyacente v$\in$Q, verificamos si w(u,v)<key[v] y reajustamos.

π[C]=B, key[C]=1
π[E]=B, key[E]=3
π[D]=B, key[D]=4

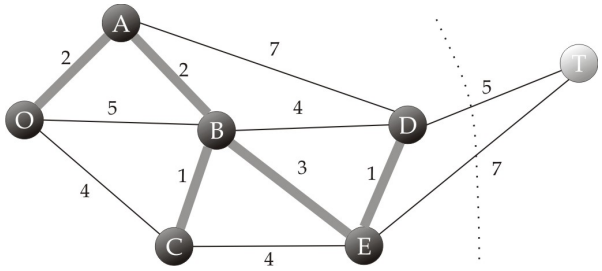


$Q \neq \phi$, en el vértice C incide la arista ligera (B,C), quitamos a C de Q, verificamos la condición para cada vértice adyacente y reajustamos, no se cumplió la condición.



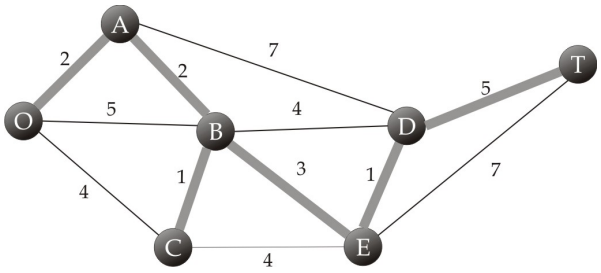
$Q \neq \phi$, en el vértice E incide la arista ligera (B,E), quitamos a E de Q y para cada vértice adyacente v∈Q verificamos lo mismo y reajustamos.

π[D]=E, key[D]=1
π[T]=E, key[T]=7



$Q \neq \phi$, en el vértice D incide la arista ligera (E,D), quitamos a D de Q y para cada vértice adyacente v∈Q, nuevamente verificamos si w(u,v)<key[v], y reajustamos.

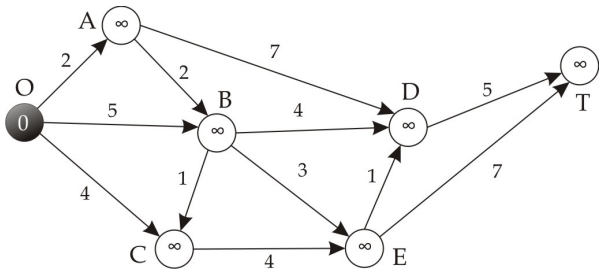
π[T]=D, key[T]=5



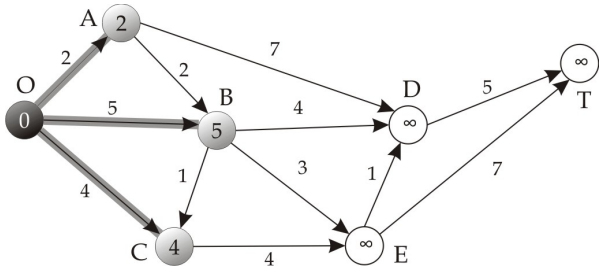
$Q \neq \phi$, en el vértice T incide la arista ligera (D,T) quitamos a T de Q, no se tiene ningún vértice adyacente, no se reajustó nada. Después de esto, $Q = \phi$ y el algoritmo finaliza.

De igual forma que con el algoritmo de Kruskal, para hacer la instalación de las líneas telefónicas para conectar todas las terminales, con un número total mínimo de cable instalado, debemos conectar la entrada con la estación A, la estación A con la B, la B con las estaciones C y E, la estación E con la D, y finalmente la estación D con la estación T; de esta forma queda resuelto el problema aplicando el algoritmo de Prim y generando la misma solución.

Aplicando el algoritmo de Dijkstra para el problema del recorrido de los tranvías en SEERVADA PARK.

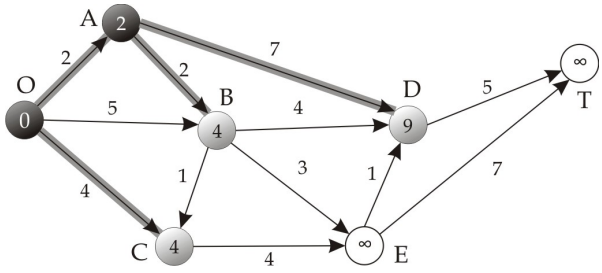


Después de la inicialización se tiene, Q={O, A, B, C, D, E, T}, ∀ v∈Q d[v]=∞, π[v]=0 y d[O]=0.



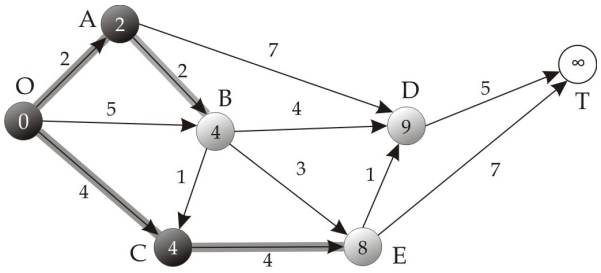
Se quitó a O de Q, se añadió a S y se relajó toda arista que sale de él

d[A]=2, π[A]=O
d[B]=5, π[B]=O
d[C]=4, π[C]=O.



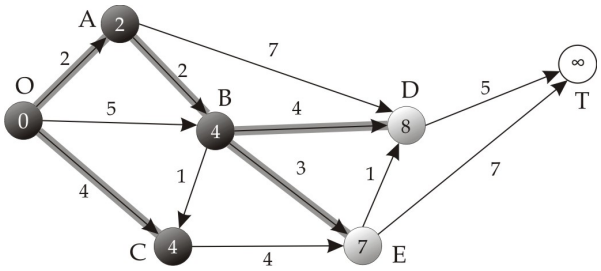
$Q \neq \phi$, el vértice A tiene el camino corto estimado más pequeño, se quitó a A de Q, se añadió a S y se relajó toda arista que sale de él

d[B]=4, π[B]=A
d[D]=9, π[D]=A.



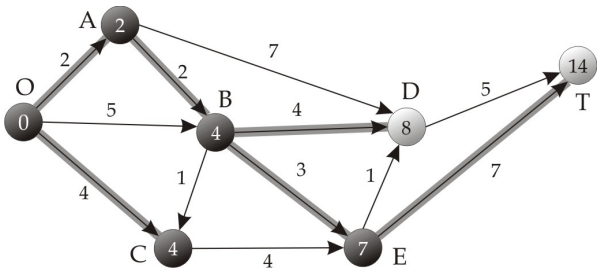
$Q \neq \phi$, los vértices B y C tienen el camino corto estimado más pequeño, se escogió a C, se quitó de Q, se añadió a S y se relajó toda arista que sale de él

$$\text{d[E]}=8, \quad \pi[\text{E}]=\text{C}.$$



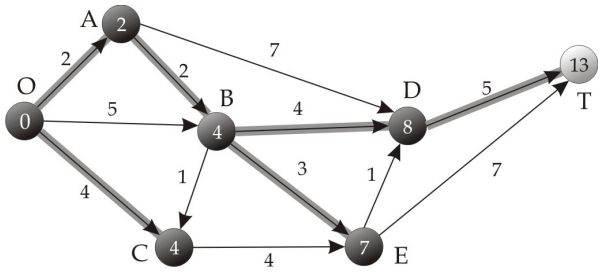
$Q \neq \phi$, el vértice B tiene el camino corto estimado más pequeño, se quitó a B de Q, se añadió a S y se relajó toda arista que sale de él

$$\text{d[D]}=8, \quad \pi[\text{D}]=\text{B}$$
$$\text{d[E]}=7, \quad \pi[\text{E}]=\text{B}.$$



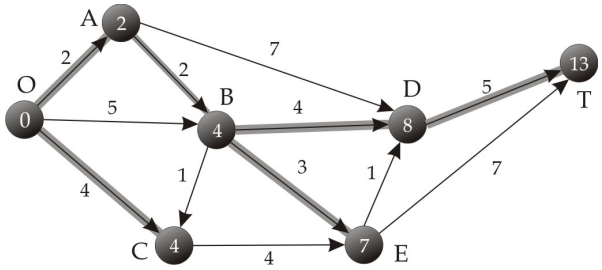
$Q \neq \phi$, el vértice E tiene el camino corto estimado más pequeño, se quitó a E de Q, se añadió a S y se relajó toda arista que sale de él

$$\text{d[T]}=14, \quad \pi[\text{T}]=\text{E}.$$



$Q \neq \phi$, el vértice D tiene el camino corto estimado más pequeño, se quitó a D de Q, se añadió a S y se relajó toda arista que sale de él

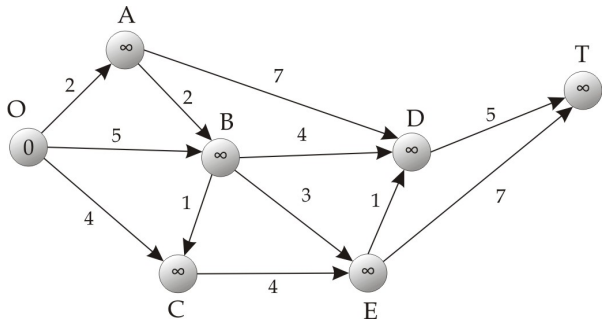
$$\text{d[T]}=13, \quad \pi\text{[T]}=\text{D}.$$



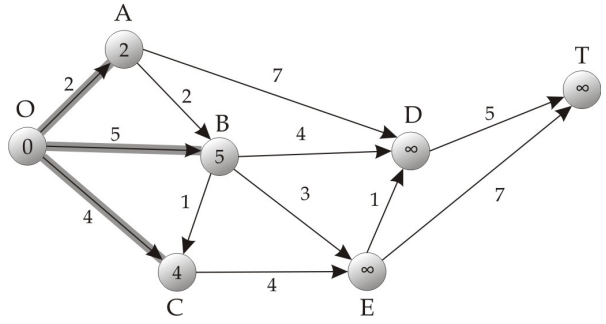
$Q \neq \phi$, el vértice T tiene el camino corto estimado más pequeño, se quitó a T de Q, se añadió a S.
Como Q=ϕ, el algoritmo finaliza.

Finalmente el árbol de caminos cortos queda de esta forma y así resolvemos el problema de la distancia total más corta para la operación de los tranvías en SEERVADA PARK, aplicando el algoritmo de Bellman-Ford.

Aplicando el algoritmo de Bellman-Ford para el problema del recorrido de los tranvías en SEERVADA PARK.
Primero consideremos el siguiente orden en que serán relajadas las aristas: (A, D), (A, B), (B, D), (B, E), (B, C), (C, E), (D, T), (E, D), (E, T), (O, A), (O, B), (O, C).

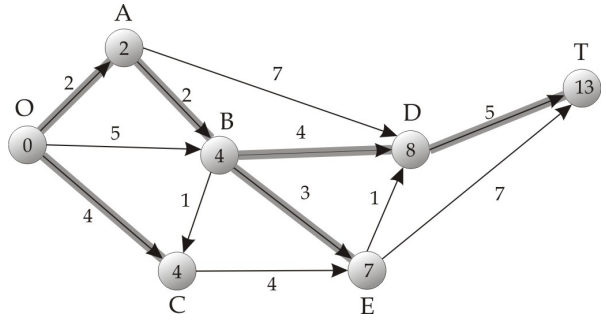


Después de la inicialización se tiene, $\forall$ v$\in$V d[v]=∞, π[v]=ϕ y d[O]=0.



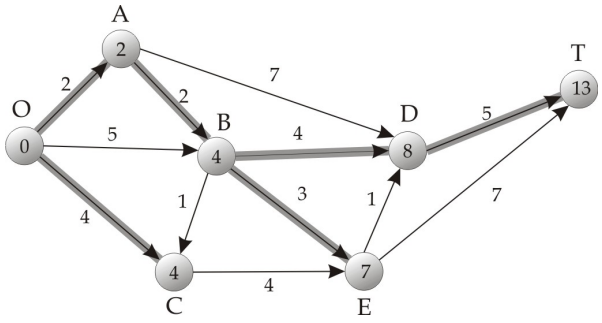
Después de la primera iteración del ciclo **para-hasta**, se tiene

d[A]=2, π[A]=O
d[B]=5, π[B]=O
d[C]=4, π[C]=O.



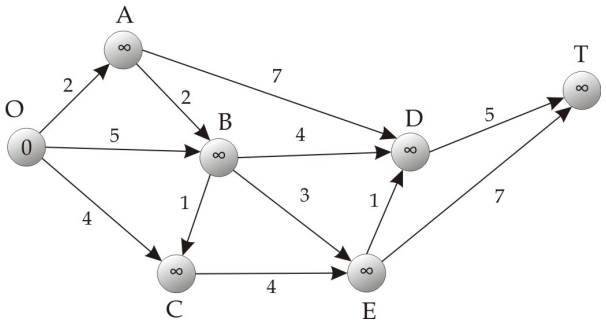
Después de la segunda iteración, se tiene

d[D]=9, π[D]=A
d[B]=4, π[B]=A
d[D]=8, π[D]=B
d[E]=7, π[E]=B
d[T]=13, π[T]=D.

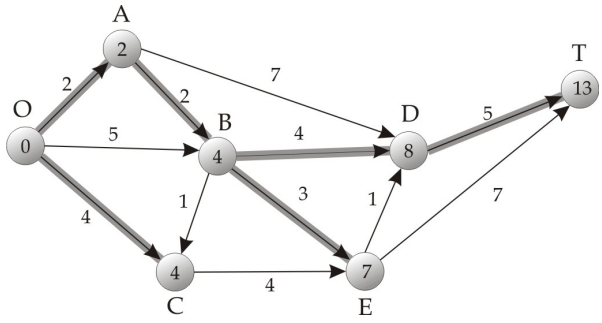


Para la tercera, cuarta, quinta y sexta iteración ya no se tienen mejoras (no hay cambios) y el árbol para la grafica queda de esta forma.

Ahora consideremos el orden de relajación de las aristas: (O, A), (O, B), (O, C), (A, D), (A, B), (B, D), (B, E), (B, C), (C, E), (D, T), (E, D), (E, T).

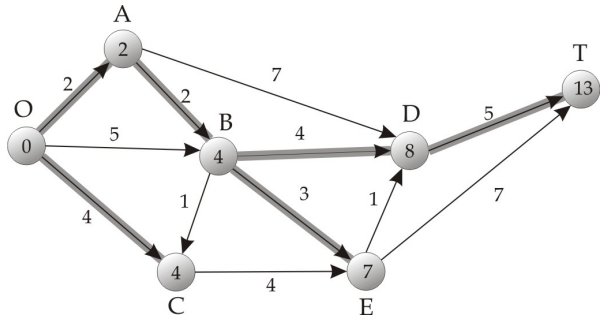


Después de la inicialización se tiene, $\forall$ v$\in$V d[v]=∞, π[v]=ϕ y d[O]=0.



Después de la primera iteración se tiene

d[A]=2,	π[A]=O
d[B]=5,	π[B]=O
d[C]=4,	π[C]=O
d[D]=9,	π[D]=A
d[B]=4,	π[B]=A
d[D]=8,	π[D]=B
d[E]=7,	π[E]=B
d[T]=13,	π[T]=D.



Para la segunda, tercera, cuarta, quinta y sexta iteración, ya no se pueden mejorar los caminos, y el árbol queda de esta forma.

Después de hacer $|V|-1$ pasadas, para cada arista de la gráfica se verifica si d[v]>d[u]+w(u,v); si se cumple, el algoritmo regresa FALSO, lo que nos dice que existe un ciclo con peso negativo alcanzable desde el origen; pero en este caso el algoritmo regresa VERDADERO. Entonces el árbol de caminos cortos queda de esta forma y así resolvemos el problema para la operación de los tranvías en SEERVADA PARK, aplicando el algoritmo de Bellman-Ford.

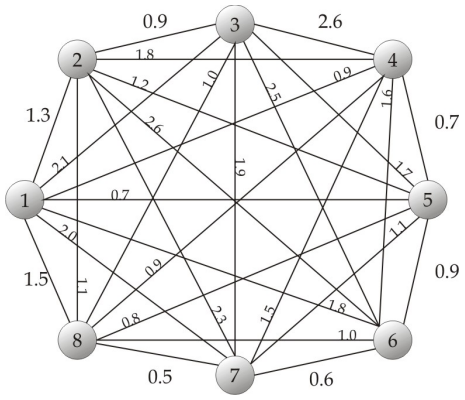
Note que para este último orden, después de la primera iteración del ciclo **para-hasta**, el árbol producido por los valores de π fue el árbol final que resolvía el problema.

6.2. Caminos entre zonas de talado para una maderera

La maderera Wirehouse taladrá árboles en ocho zonas de la misma área. Para esto debe desarrollar un sistema de caminos de tierra para tener acceso a cualquier zona desde cualquier otra. La distancia (en millas) entre cada par de zonas es:

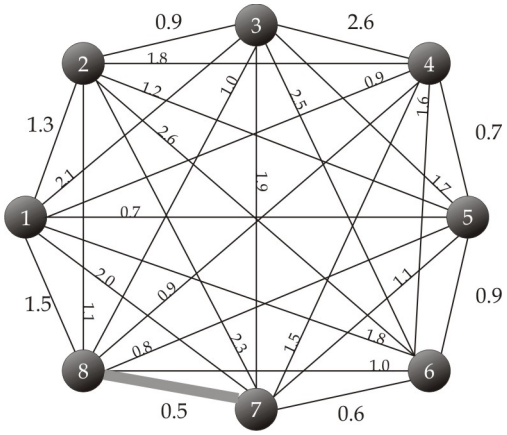
		Distancia entre pares de zonas							
		1	2	3	4	5	6	7	8
	1	-	1.3	2.1	0.9	0.7	1.8	2.0	1.5
	2	1.3	--	0.9	1.8	1.2	2.6	2.3	1.1
Z	3	2.1	0.9	--	2.6	1.7	2.5	1.9	1.0
o	4	0.9	1.8	2.6	--	0.7	1.6	1.5	0.9
n	5	0.7	1.2	1.7	0.7	--	0.9	1.1	0.8
a	6	1.8	2.6	2.5	1.6	0.9	--	0.6	1.0
s	7	2.0	2.3	1.9	1.5	1.1	0.6	--	0.5
	8	1.5	1.1	1.0	0.9	0.8	1.0	0.5	--

El problema es determinar los pares de zonas entre los que deben construírse caminos para conectar todas con una longitud total mínima de caminos.

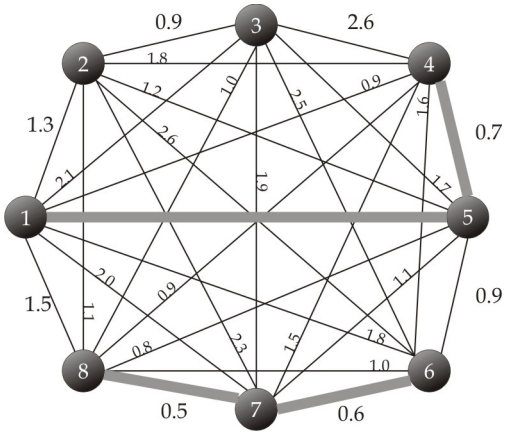


La gráfica para el problema de la maderera.

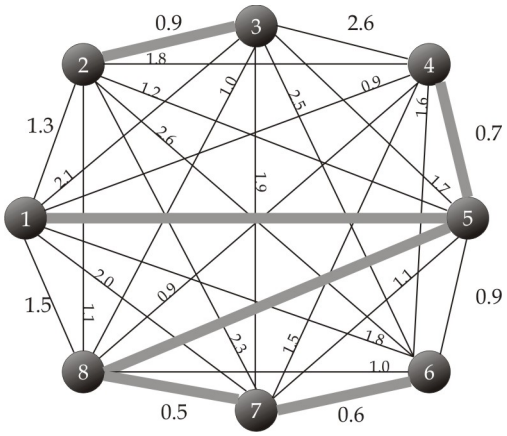
Aplicando el algoritmo de Kruskal para determinar los caminos que debe construir la maderera Wirehouse se tiene:



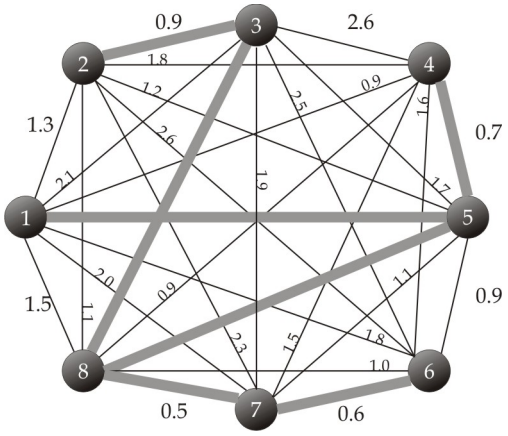
Creamos el bosque, cada árbol tiene solamente un vértice. Se ordenan las aristas de forma creciente con respecto a sus pesos, escogemos la arista ligera (7,8), como unía árboles distintos la añadimos B, y fusionamos los árboles.



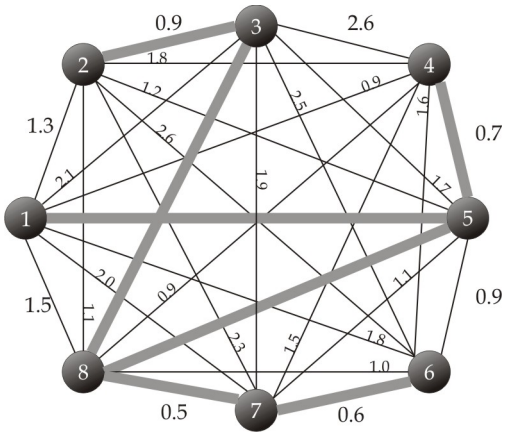
Como la arista ligera (6,7) unía árboles distintos, se añadió a B y se fusionaron los árboles. Después se tenían dos aristas ligeras, (1,5) y (4,5); se escogió la (1,5) y como unía árboles distintos, se añadió a B y se fusionaron los árboles. Después, se consideró la arista ligera (4,5), unía árboles distintos, se añadió a B y se fusionaron los árboles.



La arista ligera (8,5) unía árboles distintos, se añadió a B y se fusionaron los árboles. Después, se tenían cuatro aristas ligeras, (2,3), (4,8), (5,6) y (1,4). Primero se consideró a (2,3), como unía árboles distintos, se añadió a B y se fusionaron los árboles; después a (4,8), como unía al mismo árbol fue descartada; lo mismo ocurrió con (5,6) y (1,4).



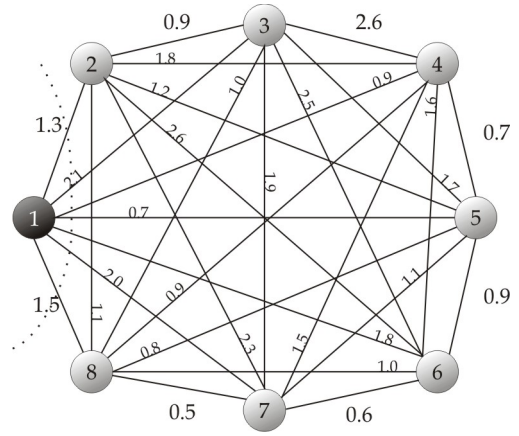
Se tenían dos aristas ligeras, (3,8) y (8,6). Como (8,6) unía al mismo árbol fue descartada, después se consideró a (3,8), como unía árboles distintos, se añadió y fueron fusionados los árboles. Nuevamente se tenían dos aristas ligeras, (2,8) y (5,7), ambas unían al mismo árbol y fueron descartadas.



A partir de ahora todas las aristas unen al mismo árbol y son descartadas.

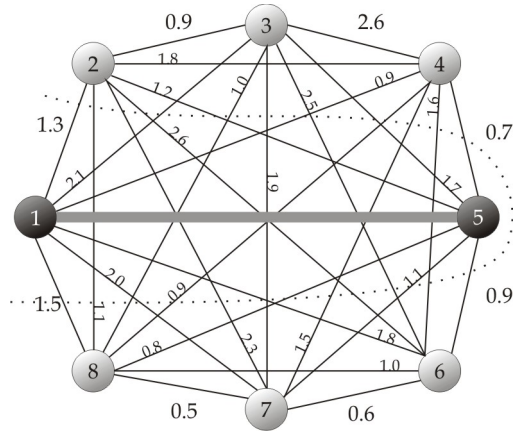
Entonces el árbol de expansión mínima para la gráfica queda así, y de esta forma determinamos qué caminos son los que se deben hacer para tener acceso a cualquier zona desde cualquier otra por la maderera Wirehouse, con una longitud total mínima, aplicando el algoritmo de Kruskal.

Aplicando el algoritmo de Prim para determinar los caminos que debe construir la maderera Wirehouse.



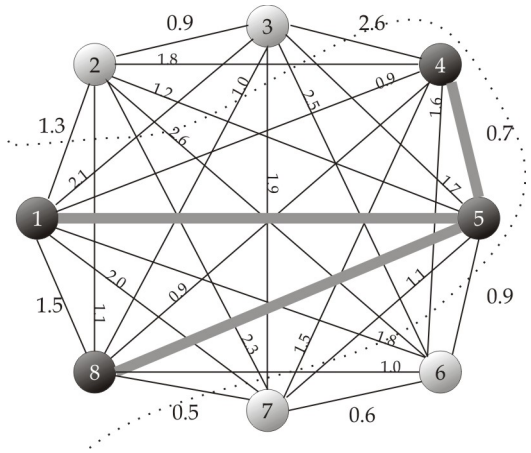
Se tiene, Q={1, 2, 3, 4, 5, 6, 7, 8}, $\forall$ u$\in$Q key[u]=∞ y key[1]=0, π[1]=ϕ.
$Q \neq \phi$, se quitó a 1 de Q y para cada vértice adyacente v$\in$Q, se verificó si w(u,v)<key[v], y se reajustó

key[2]=1.3, key[3]=2.1, key[4]=0.9,
key[5]=0.7, key[6]=1.8, key[7]=2.0,
key[8]=1.5



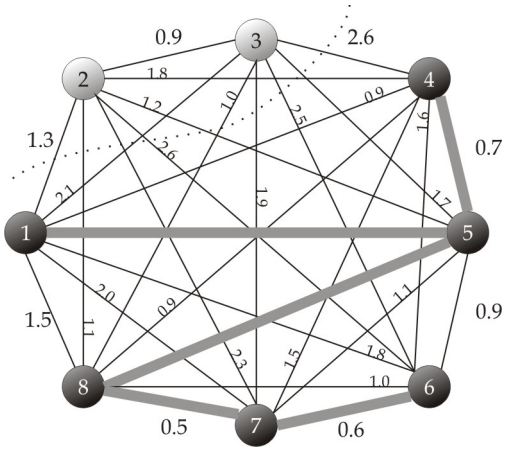
$Q \neq \phi$, la arista (1,5) era una arista ligera atravesando V-Q, quitamos a 5 de Q y para cada vértice adyacente v$\in$Q, se verificó la condición y se reajustó

key[4]=0.7, key[3]=1.7, key[2]=1.2,
key[6]=0.9, key[7]=1.1, key[8]=0.8



$Q \neq \phi$, la arista (5,4) era una arista ligera atravesando V-Q, quitamos a 4 de Q y para cada vértice adyacente v$\in$Q, se verificó si w(u,v)<key[v], no se cumplió.
$Q \neq \phi$, la arista (5,8) era una arista ligera atravesando V-Q, quitamos a 8 de Q y para cada vértice adyacente v$\in$Q, verificó la condición y se reajustó

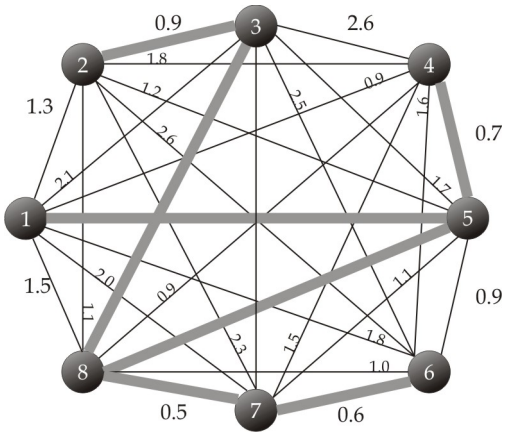
key[2]=1.1, key[3]=1.0, key[7]=0.5



$Q \neq \phi$, la arista (8,7) era una arista ligera atravesando V-Q, quitamos a 7 de Q y para cada vértice adyacente v∈Q, se verificó la condición y se reajustó

key[6]=0.6

$Q \neq \phi$, la arista (7,6) era una arista ligera atravesando V-Q, quitamos a 6 de Q y para cada vértice adyacente v∈Q, se verificó la condición, no se cumplió.



$Q \neq \phi$, la arista (8,3) era una arista ligera atravesando V-Q, quitamos a 3 de Q y para cada vértice adyacente v∈Q, se verificó la condición y se reajustó

key[2]=0.9

$Q \neq \phi$, la arista (3,2) es una arista ligera atravesando V-Q, quitamos a 2 de Q. Como $Q = \phi$, el algoritmo finaliza.

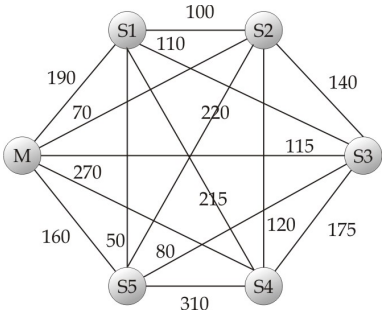
Como $Q = \phi$, entonces el algoritmo finaliza y el árbol de expansión mínima para la gráfica queda así, y de esta forma determinamos qué caminos de tierra son los que se deben hacer para tener acceso a cualquier zona desde cualquier otra en la maderera Wirehouse, con una longitud total mínima, aplicando el algoritmo de Prim.

6.3. Conexión de terminales de una sucursal bancaria

Un banco ha decidido conectar terminales de computadora de cada sucursal a la computadora central de su oficina matriz, mediante líneas telefónicas especiales con dispositivos de telecomunicaciones. No es necesario que la línea telefónica de una sucursal esté conectada directamente con la oficina matriz. La conexión puede ser indirecta a través de otra sucursal que esté conectada (directamente o indirectamente) a la matriz. El único requisito es que exista alguna ruta que conecte a todas las sucursales con la oficina matriz.
El cargo por las líneas telefónicas especiales es directamente proporcional a la distancia cableada, en donde esta distancia (en millas) es:

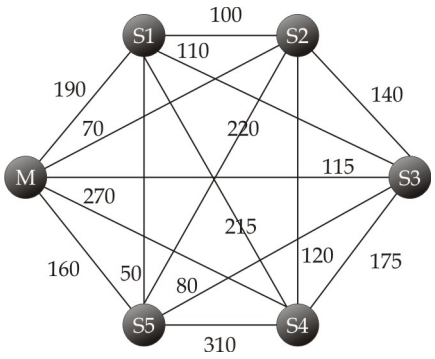
	Distancia entre pares de oficinas					
	Principal	Suc.1	Suc.2	Suc.3	Suc.4	Suc.5
Principal	---	190	70	115	270	160
Sucursal 1	190	---	100	110	215	50
Sucursal 2	70	100	---	140	120	220
Sucursal 3	115	110	140	---	175	80
Sucursal 4	270	245	120	175	---	310
Sucursal 5	160	30	220	80	310	---

La administración desea determinar qué pares de sucursales conectar directamente con las líneas telefónicas especiales para que todas queden conectadas (de modo directo o indirecto) a la oficina matriz con un costo total mínimo.

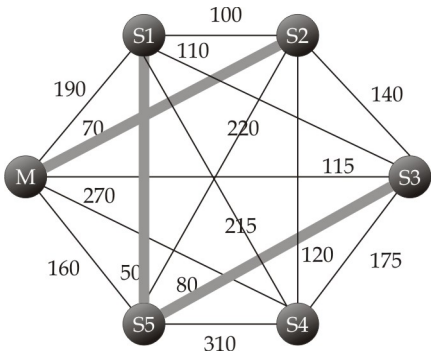


La gráfica para el problema de las sucursales bancarias.

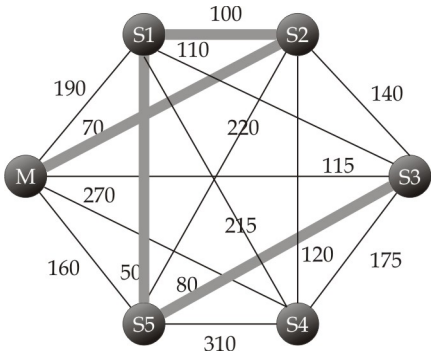
Aplicando el algoritmo de Kruskal para determinar las conexiones entre las sucursales bancarias.



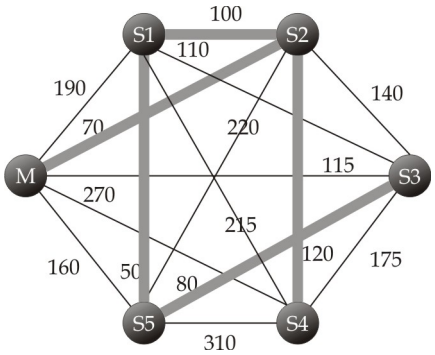
Creamos el bosque, cada árbol tiene solamente un vértice. La arista (S1,S5) es una arista ligera, como une árboles distintos, la añadimos a B y fusionamos los árboles. Consideramos la arista ligera (M,S2), como une árboles distintos la añadimos a B y fusionamos los árboles. La siguiente arista ligera es (S5,S3), como une árboles distintos, entonces la añadimos a B y fusionamos los árboles.



Ahora consideramos la arista ligera (S1,S2) y como une árboles distintos, la añadimos a B y fusionamos los árboles. La siguiente arista ligera es (S1,S3), pero no une árboles distintos y la descartamos. La siguiente arista ligera es la (M,S3), pero une al mismo árbol y la descartamos.



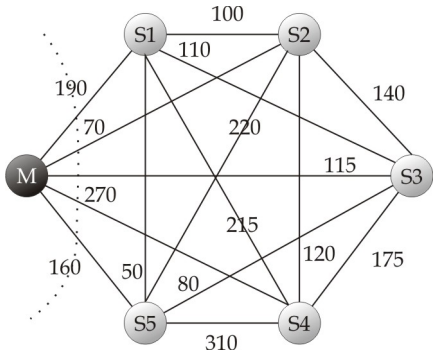
Ahora la arista (S2,S4) es una arista ligera y como une árboles distintos, la añadimos a B y fusionamos los árboles. La siguiente es la (S2,S3) pero esta une al mismo árbol y la descartamos.



De aquí en adelante todas las aristas unen al mismo árbol y entonces son descartadas.

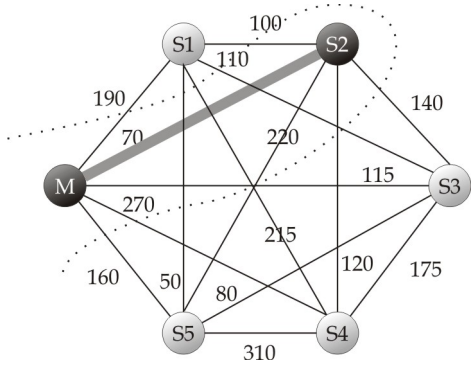
Entonces, el árbol de expansión mínima para la gráfica queda así; de esta forma determinamos qué sucursales son las que se deben conectar entre sí junto con la oficina matriz, con un costo total mínimo, aplicando el algoritmo de Kruskal.

Aplicando el algoritmo de Prim para determinar las conexiones entre las sucursales bancarias.



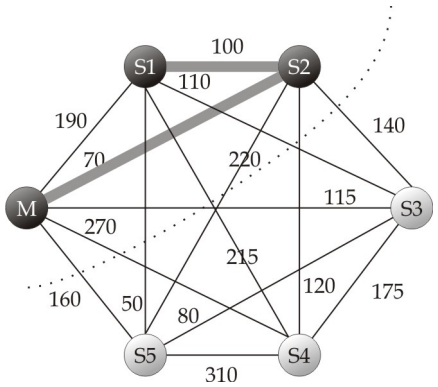
Tenemos, Q={M, S1, S2, S3, S4, S5}, $\forall$ u$\in$Q key[u]=∞ y key[M]=0, π[M]=ϕ.
$Q \neq \phi$, quitamos a M de Q y para cada vértice adyacente v$\in$Q, se verificó si w(u,v)<key[v], y se reajustó.

key[S1]=190, key[S2]=70, key[S3]=115, key[S4]=270, key[S5]=160.



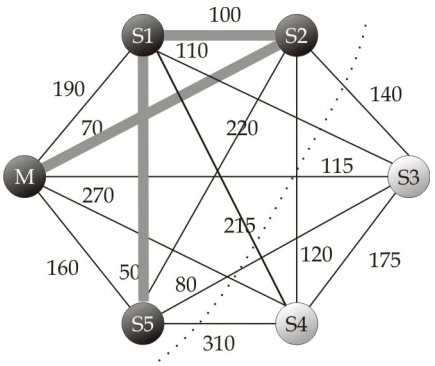
$Q \neq \phi$, la arista (M,S2) es una arista ligera atravesando V-Q, quitamos de a S2 de Q y para cada vértice adyacente v$\in$Q, se verificó si w(u,v)<key[v], y se reajustó

key[S4]=120, key[S1]=100.



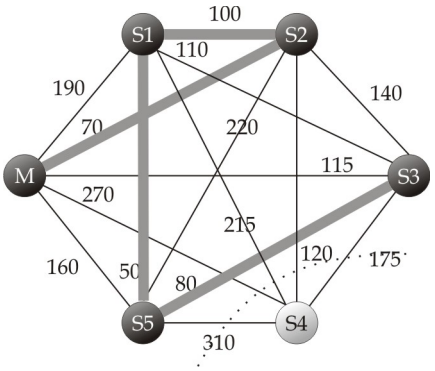
$Q \neq \phi$, la arista (S2,S1) es una arista ligera atravesando V-Q, quitamos a S1 de Q y para cada vértice adyacente v$\in$Q, se verificó si w(u,v)<key[v], y se reajustó

key[S3]=100.

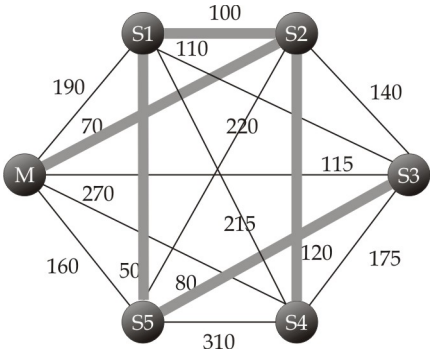


$Q \neq \phi$, la arista (S1,S5), es una arista ligera atravesando V-Q, quitamos a S5 de Q y para cada vértice adyacente v∈Q, se verificó si w(u,v)<key[v], y se reajustó

key[S3]=80.



$Q \neq \phi$, la arista (S5,S3), es una arista ligera atravesando V-Q, quitamos a S3 de Q y para cada vértice adyacente v∈Q, se verificó si w(u,v)<key[v], no se cumplió.



$Q \neq \phi$, la arista (S2,S4) es una arista ligera atravesando V-Q, quitamos a S4 de Q y para cada vértice adyacente v∈Q, se verificó si w(u,v)<key[v], no se hizo nada. Como $Q = \phi$, entonces el algoritmo finaliza.

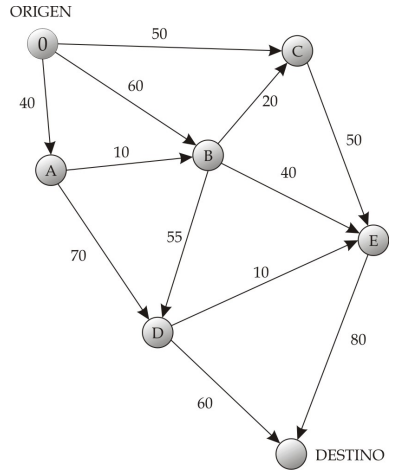
Finalmente, el árbol de expansión mínima para la gráfica del problema de las conexiones entre las sucursales bancarias y la oficina matriz mediante líneas telefónicas especiales con un costo total mínimo, después de aplicar el algoritmo de Prim queda de esta forma.

6.4. La ruta más corta en un viaje en auto

Una persona debe hacer un viaje en auto a otra ciudad que nunca ha visitado, consultando un plano para determinar la ruta más corta a su destino. Según la ruta que elija, hay otras cinco ciudades (llamadas A, B, C, D, E) por las que puede pasar en el camino. El plano muestra las millas de cada carretera que es una conexión directa entre dos ciudades sin que otra intervenga. Estas cifras se resumen en la siguiente tabla, donde un guión indica que no hay conexión directa sin pasar por las otras ciudades.

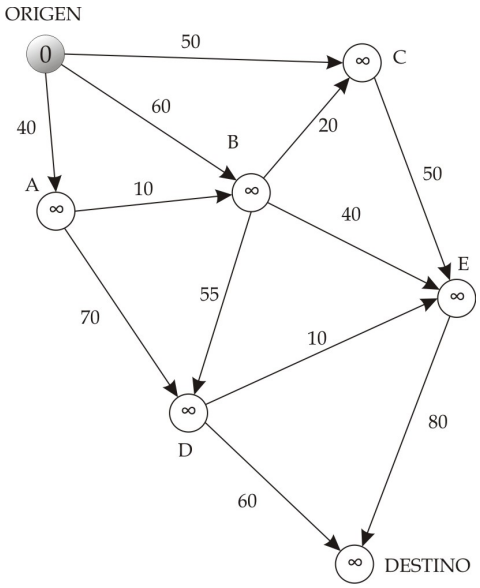
	Millas entre cuidades adyacentes					
Pueblo	A	B	C	D	E	Destino
Origen	40	60	50	--	--	--
A		60	--	70	--	--
B			20	55	40	--
C				--	50	--
D					10	60
E						80

El problema es determinar que ruta debe seguir el automovilista para llegar a su destino, con la mínima distancia total recorrida.

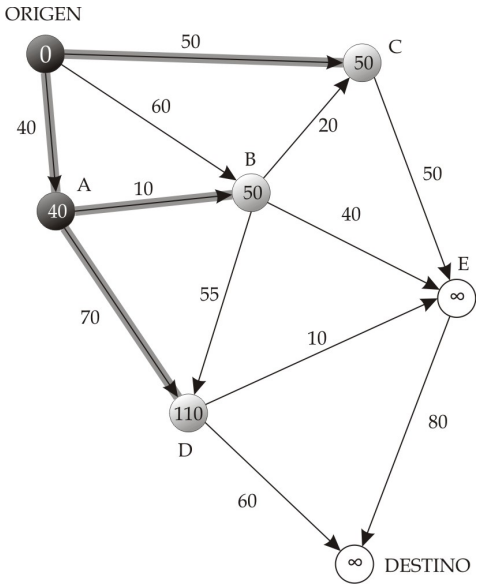


La gráfica para el problema del viaje en auto.

Aplicando el algoritmo de Dijkstra para determinar la ruta que deberá seguir el automovilista.



Después de la inicialización se tiene, S=ϕ, Q={OR, A, B, C, D, E, DE}, $\forall$ u$\in$Q d[u]=∞, π[u]=ϕ y d[OR]=0.

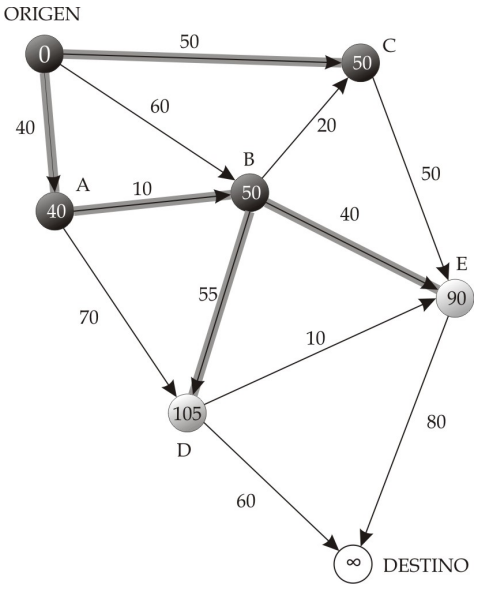


Se quitó al vértice OR (ORIGEN) de Q, se añadió a S y se relajó toda arista que sale de él, se tiene

$$\begin{array}{ll} \text{d[A]=40,} & \pi\text{[A]=OR} \\ \text{d[B]=60,} & \pi\text{[B]=OR} \\ \text{d[C]=50,} & \pi\text{[C]=OR.} \end{array}$$

$Q \neq \phi$, el vértice A tenía el camino corto estimado más pequeño, se quitó de Q, se añadió a S y se relajó toda arista que sale de él, se tiene

$$\begin{array}{ll} \text{d[B]=50,} & \pi\text{[B]=A} \\ \text{d[D]=110,} & \pi\text{[D]=A.} \end{array}$$

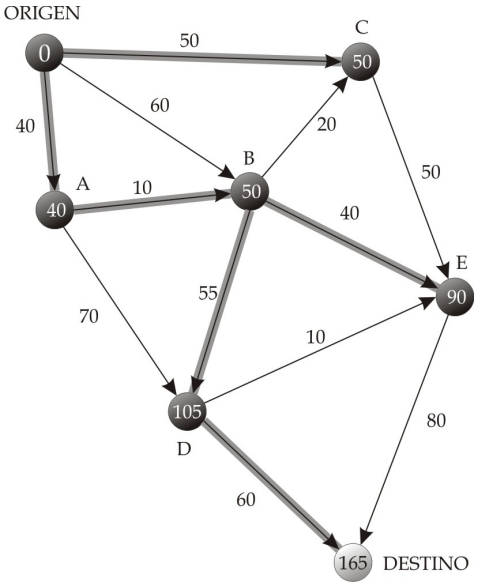


$Q \neq \phi$, los vértices C y B tenían el camino corto estimado más pequeños, elegimos a C, se quitó de Q, se añadió a S, y se relajó toda arista que sale de él, se tiene

$$\text{d[E]}=100, \quad \pi\text{[E]}=\text{C}.$$

$Q \neq \phi$, el vértice B tenía el camino corto estimado más pequeño, se quitó de Q, se añadió a S y se relajó toda arista que sale de él, se tiene

$$\text{d[E]}= 90, \quad \pi\text{[E]}=\text{B}$$
$$\text{d[D]}=105, \quad \pi\text{[D]}=\text{B}.$$

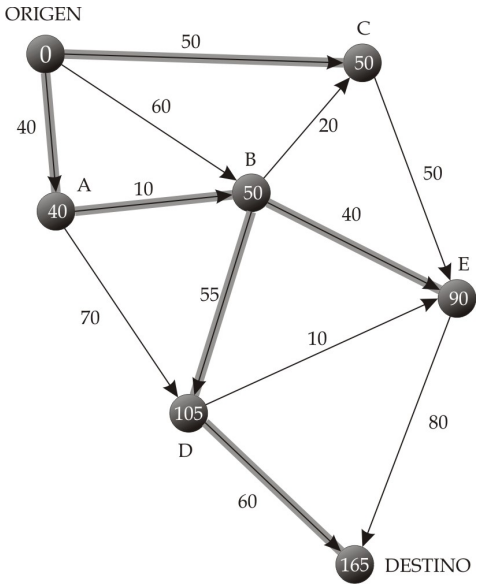


$Q \neq \phi$, el vértice E tenía el camino corto estimado más pequeño, se quitó de Q, se añadió a S, y se relajó toda arista que sale de él, se tiene

$$\text{d[DE]}=170, \quad \pi\text{[DE]}=\text{E}.$$

$Q \neq \phi$, el vértice D tenía el camino corto estimado más pequeño, se quitó de Q, se añadió a S y se relajó toda arista que sale de él, se tiene

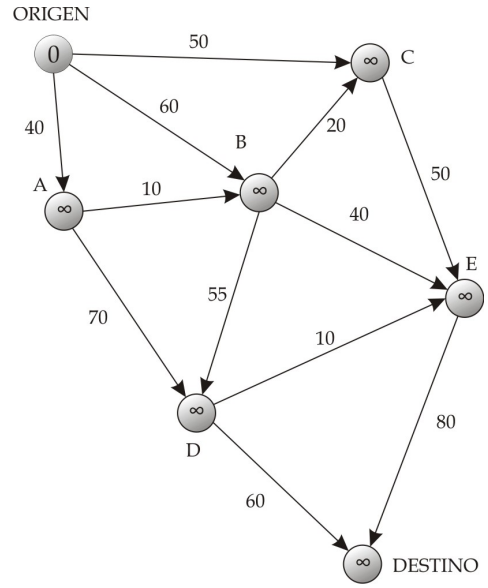
$$\text{d[DE]}=165, \quad \pi\text{[DE]}=\text{D}.$$



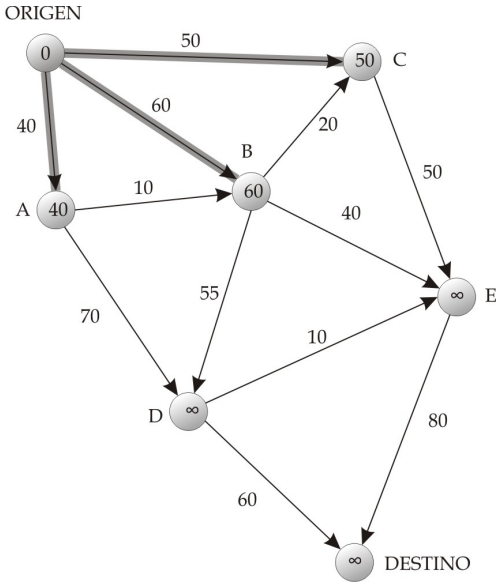
$Q \neq \phi$, el vértice DESTINO (DE) tenía el camino corto estimado más pequeño, se quitó de Q y se añadió a S, no se relaja nada.
Como Q=ϕ, el algoritmo finaliza.

De esta forma resolvemos el problema de determinar la ruta que debe seguir el automovilista para llegar a su destino, con la mínima distancia recorrida; la ruta que debe seguir es: OR→A→B→D→DE, aplicando el algoritmo de Dijkstra.

Aplicando el algoritmo de Bellman-Ford para determinar la ruta que deberá seguir el automovilista.
El orden en que serán relajadas cada arista es: (A, B), (A, D), (B, D), (B, E), (B, C), (D, E), (D, DE), (E, DE), (OR, A), (OR, B), (OR, C).

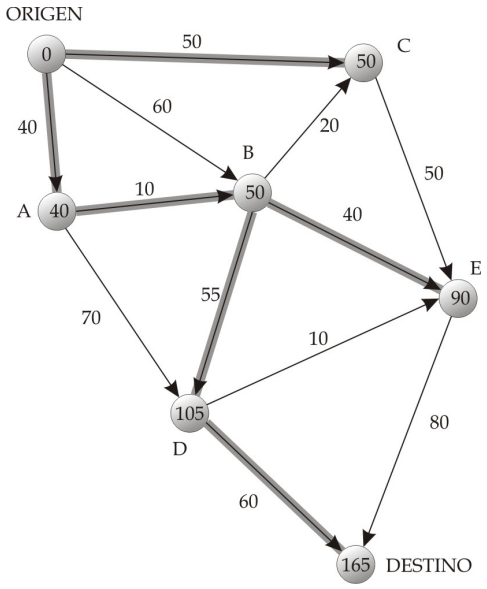


La situación justo antes de la primera iteración, $\forall$ v$\in$V d[v]=∞, π[v]=ϕ y d[OR]=0.



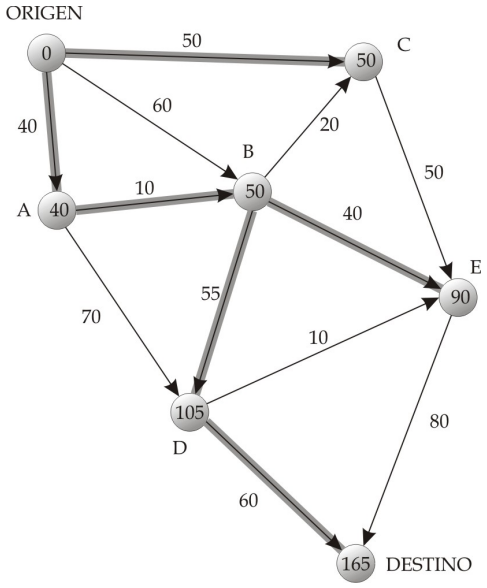
Después de la primera iteración se tiene

d[A]=40, π[A]=OR
d[B]=60, π[B]=OR
d[C]=50, π[C]=OR.



Después de la segunda iteración se tiene

$$\begin{array}{ll} d[D]=110, & \pi[D]=A \\ d[B]=\ 50, & \pi[B]=A \\ d[D]=105, & \pi[D]=B \\ d[E]=\ 90, & \pi[E]=B. \end{array}$$

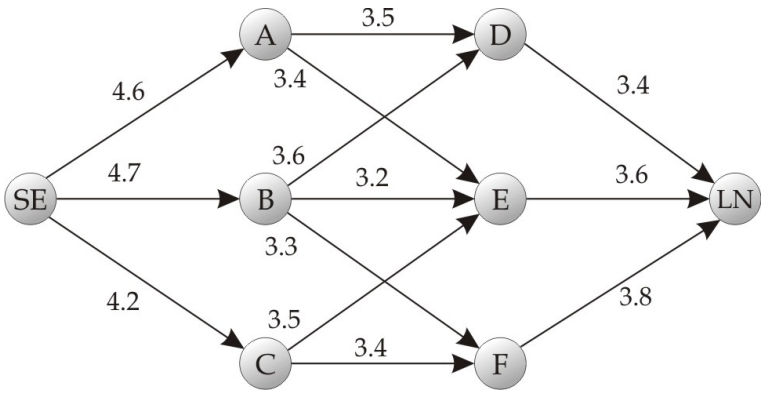


Para la tercera, cuarta, quinta y sexta iteración, ya no hay mejoras y el árbol queda de esta forma.

Después para cada arista de la gráfica, se verifica si d[v]>d[u]+w(u,v), no se cumple y el algoritmo regresa VERDADERO, entonces el árbol de caminos cortos queda de esta forma y así determinamos la ruta que debe seguir el automovilista para llegar a su destino, la cual es: OR→A→B→D→DE, aplicando Bellman-Ford.

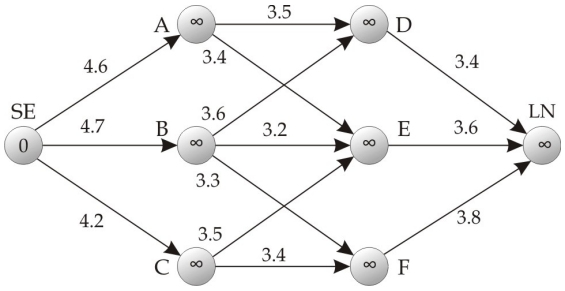
6.5. La ruta más corta de un vuelo en avión

Un vuelo de Speedy Airlines está a punto de despegar de Seattle sin escalas a Londres. Existe cierta flexibilidad para elegir la ruta más precisa, según las condiciones del clima. La siguiente gráfica describe las rutas posibles consideradas, donde SE y LN son Seattle y Londres, respectivamente, y los otros vértices representan varios lugares intermedios. El viento a lo largo de cada arista afecta mucho el tiempo de vuelo (y por ende el consumo de combustible). Con base en el informe meteorológico actual, sobre las aristas se muestran los tiempos de vuelo (en horas). Debido al alto costo del combustible, la administración ha establecido la política de elegir la ruta que minimiza el tiempo total de vuelo. ¿Qué ruta debe elegir el vuelo?.

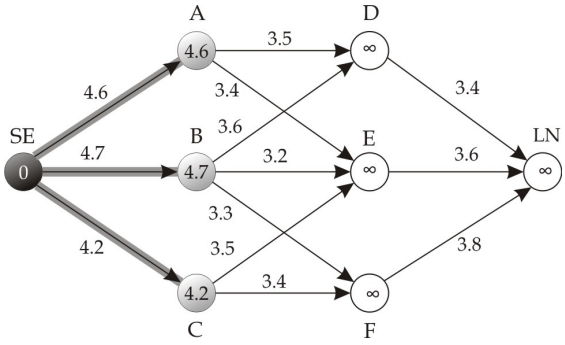


La gráfica para el vuelo de Speedy Airlines.

Aplicando el algoritmo de Dijkstra para determinar la ruta que deberá seguir el vuelo de Speedy Airlines.



La situación justo antes de la primera iteración es Q={SE, A, B, C, D, E, F, LN}, S=ϕ, $\forall$ v$\in$V d[v]=∞, π[v]=ϕ y d[SE]=0.

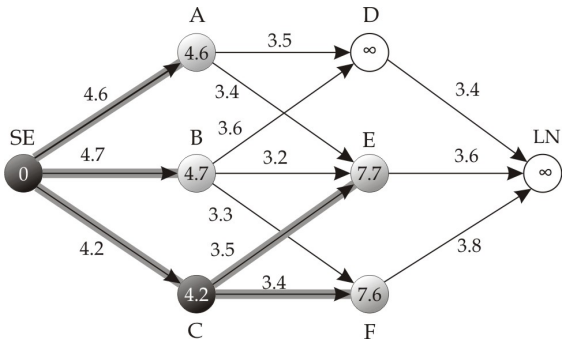


Quitamos a SE de Q, lo añadimos a S y relajamos toda arista que sale de SE

d[A]=4.6, π[A]=SE
d[B]=4.7, π[B]=SE
d[C]=4.2, π[C]=SE.

$Q \neq \phi$, el vértice C tiene el camino corto estimado más pequeño, lo quitamos de Q, lo añadimos a S y relajamos toda arista que sale de C

d[E]=7.7, π[E]=C
d[F]=7.6, π[F]=C.

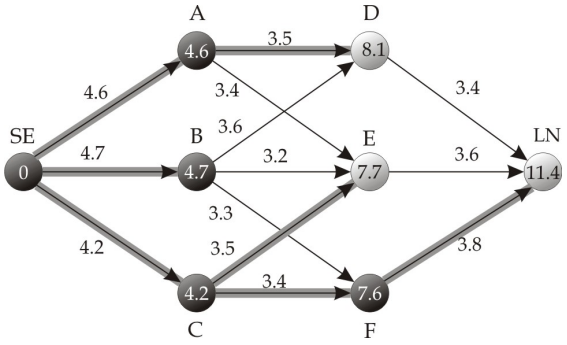


$Q \neq \phi$, el vértice A tiene el camino corto estimado más pequeño, quitamos a A de Q, lo añadimos a S, y relajamos toda arista que sale de A

$$\text{d[D]}=8.1, \quad \pi[\text{D}]=\text{A}.$$

$Q \neq \phi$, el vértice B tiene el camino corto estimado más pequeño, lo quitamos de Q, lo añadimos a S, y relajamos (no hay cambios). Después, el vértice F tiene el camino corto estimado más pequeño, lo quitamos de Q y relajamos cada arista que sale de F

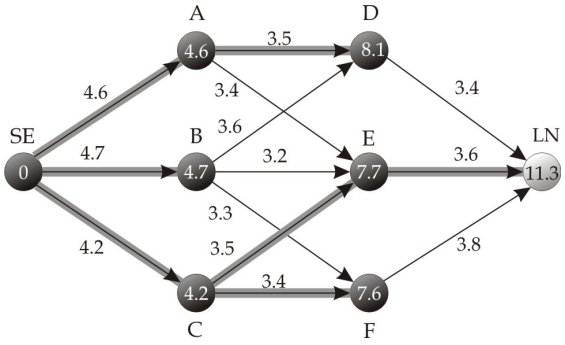
$$\text{d[CN]}=11.4, \quad \pi[\text{CN}]=\text{F}.$$



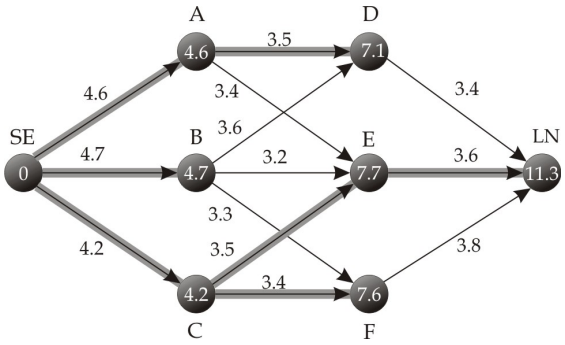
$Q \neq \phi$, el vértice E tiene el camino corto estimado más pequeño, quitamos a E de Q, lo añadimos a S y relajamos las aristas que salen de E

$$\text{d[LN]}=11.3, \quad \pi[\text{LN}]=\text{E}.$$

El vértice D tiene el camino corto estimado mas pequeño, lo quitamos de Q, la añadimos a S y relajamos (no hay cambios).



$Q \neq \phi$, el vértice LN tiene el camino corto estimado más pequeño, quitamos a LN de Q, lo añadimos a S y relajamos (no hay cambios).
Como Q=ϕ, el algoritmo finaliza.

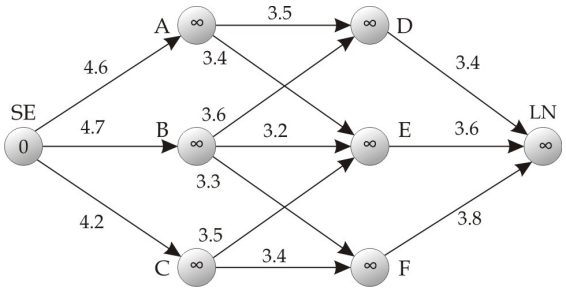


Finalmente el árbol queda de esta forma.

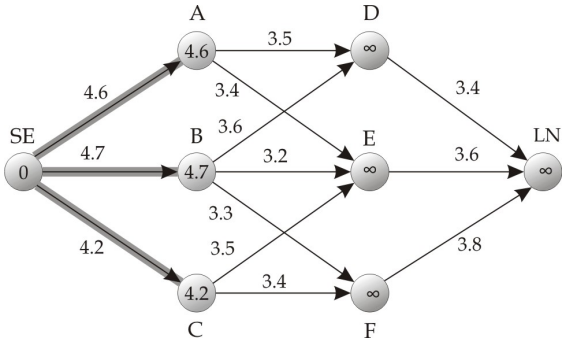
Entonces, de acuerdo con las consideraciones del clima en ese momento, la ruta que deberá hacer el vuelo de Speedy Airlines es, de la terminal de Seattle a la terminal intermedia C, después a la terminal E y finalmente a la terminal de Londres, esto es, (SE→C→E→LN).

Aplicando el algoritmo de Bellman-Ford para determinar la ruta que deberá seguir el vuelo de Speedy Airlines.

Consideremos el orden (A, D), (B, E), (C, F), (E, LN), (SE, B), (D, LN), (A, E), (B, F), (SE, C), (F, LN), (B, D), (C, E), (SE, A).

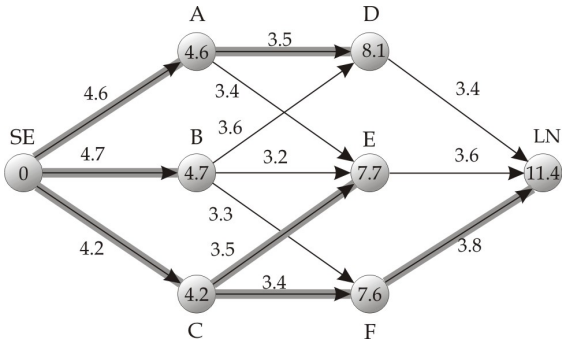


La situación justo antes de la primera iteración, se tiene: $\forall$ v$\in$V d[v]=∞, π[v]=ϕ y d[SE]=0.



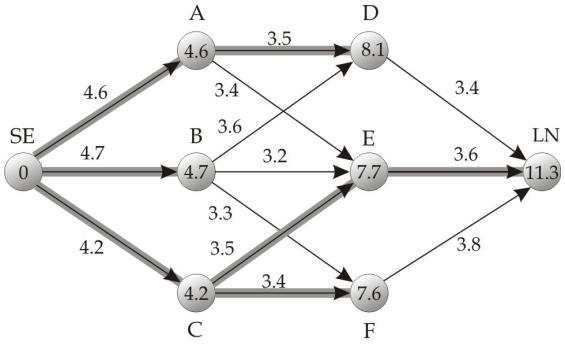
Después de la primera iteración, se tiene

d[B]=4.7, π[B]=SE
d[C]=4.2, π[C]=SE
d[A]=4.6, π[A]=SE.



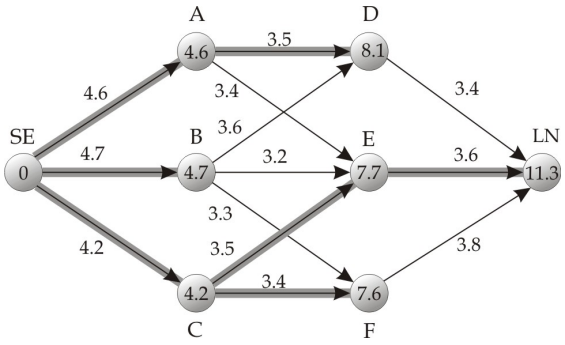
Después de la segunda iteración, se tiene

d[D]=8.1, π[D]=A
d[E]=8.0, π[E]=A
d[F]=8.0, π[F]=B
d[LN]=11.5, π[LN]=E
d[LN]=11.4, π[LN]=F
d[E]=7.7, π[E]=C
d[F]=7.6, π[F]=C.



Después de la tercera iteración, se tiene

$$d[LN]=11.3,\ \pi[LN]=E.$$



Para la cuarta, quinta, sexta y séptima iteración, ya no hay cambios. Después, para cada $(u, v) \in A$ se verifica si d[v]>d[u]+w(u,v). No se cumple, el algoritmo regresa VERDADERO y el árbol queda de ésta forma.

Igualmente que con el algoritmo de Dijkstra, de acuerdo con las consideraciones del clima en ese momento, la ruta que deberá hacer el vuelo de Speedy Airlines es, de la terminal de Seattle a la terminal intermedia C, después a la terminal E y finalmente a la terminal de Londres, esto es, (SE→C→E→LN).

Apéndice A

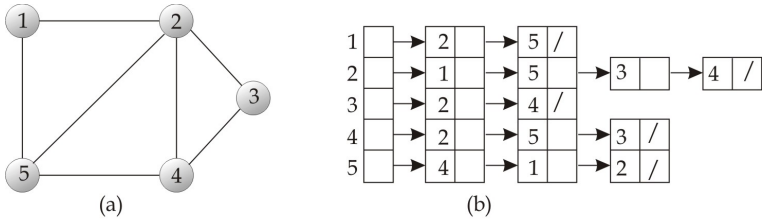
Representación de gráficas

Con el fin de procesar gráficas por medio de un programa de computadoras se necesita decidir como representarlas en la máquina. En esta tesis no se programarán los algoritmos, pero es útil saber como se podrían hacer las representaciones de gráficas para que sean procesadas por un programa de computadora, ya que la eficiencia dependará de la implementación que se haga.

Hay dos formas estándar para representar una gráfica. La representación con una lista de adyacencia es usualmente utilizada porque ésta provee una forma compacta de representar gráficas *dispersas* , cuando $|A|$ es mucho menor que $|V|^2$. La representación con una matriz de adyacencia, se prefiere si la gráfica es *densa*, es decir, $|A|$ es cercana a $|V|^2$ o cuando necesitamos decir rápidamente si existe una arista conectada a dos vértices dados.

La *representación con una lista de adyacencia* de la gráfica $G = (V, A)$ consiste de un arreglo Ady de $|V|$ listas, una para cada vértice en V. Para cada $u \in V$, la lista de adyacencia $Ady[u]$ contiene todos los vértices v, tal que hay una arista $(u, v) \in A$. Esto es, $Ady[u]$ consiste de todos los vértices adyacentes a u en G. Los vértices en cada lista de adyacencia son típicamente almacenados en un orden arbitrario.

La figura 1(b) es una representación con una lista de adyacencia de la gráfica no dirigida en la figura 1(a). Similarmente, la figura 2(b) es una representación con una lista de adyacencia de la gráfica dirigida en la figura 2(a).



	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

(c)

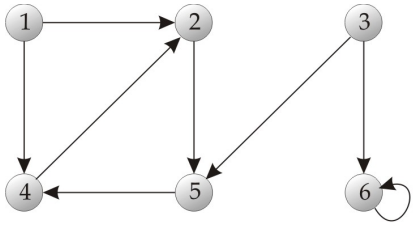
Figura 1. Dos representaciones de una gráfica no dirigida. **(a)** Una gráfica G con cinco vértices y siete aristas. **(b)** Representación con un lista de adyacencia de G. **(c)** Representación con una matriz de adyacencia de G.

Si G es una gráfica dirigida, la suma de las longitudes de todas las listas de adyacencia es $|A|$, entonces una arista de la forma (u, v) es representada si se tiene a v en $Ady[u]$, si G es una gráfica no dirigida, la suma de las longitudes de todas las listas de adyacencia es $2|A|$, puesto que u aparece en la lista de adyacencia de v y viceversa.

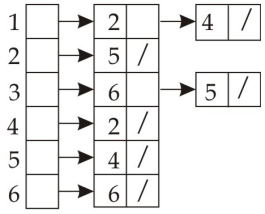
Una desventaja de la lista de adyacencia es que no hay una forma rápida de determinar si una arista dada (u, v) es presentada en la gráfica, es decir, no hay una forma rápida de buscar v en la lista de adyacencia $Ady[u]$; esto se puede modificar usando una matriz de adyacencia, pero se utiliza más memoria.

Para la *representación con una matriz de adyacencia* de la gráfica $G = (V, A)$, suponemos que los vértices son enumerados $1, 2, \ldots, |V|$ en forma arbitraria.

La representación con una matriz de adyacencia de una gráfica $G = (V, A)$ consiste de una matriz $A = (a_{ij})$ con dimensión $|V| \times |V|$ tal que



(a)



(b)

	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

(c)

Figura 2. Dos representaciones de una gráfica dirigida. **(a)** Una gráfica G con seis vértices y ocho aristas. **(b)** Representación con una lista de adyacencia de G. **(c)** Representación con una matriz de adyacencia de G.

$$a_{ij} = \begin{cases} 1 & \text{si } (i,j) \in A \\ 0 & \text{si } (i,j) \notin A \end{cases}$$

La figura 1(c) es una representación con una matriz de adyacencia de la gráfica no dirigida en la figura 1(a). Similarmente, la figura 2(c) es una representación con una matriz de adyacencia de la gráfica dirigida en la figura 2(a).

Obsérvese la simetría a lo largo de la diagonal principal de la matriz de adyacencia en la figura 1(c). Definimos la *transpuesta* de una matriz $A = (a_{ij})$ como $A^t = (a_{ji})$. Como en una gráfica no dirigida, (u, v) y (v, u) representan la misma arista, la matriz de adyacencia A de una gráfica no dirigida es igual a la transpuesta: $A = A^t$.

Bibliografía

[1] Cormen, Leiserson, Rivest, *Introduction to Algorithms*, MIT Press.

[2] Wataru Mayeda, *Graph Theory*, Wiley-Intersciencie, 1972.

[3] Reinhard Diestel, *Graph Theory*, Second Edition, Springer.

[4] Javier Salazar Resines, *Enfoque de sistemas en la educacion: Teoría de Gráficas*, Second Edition, Limusa 1979.

[5] Fausto A. Toranzos, *Introducción a la Teoría de Gráficas.*

[6] Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman, *Estructuras de Datos y Algoritmos*, Addison-Wesley, Reading, MA,1983.

[7] Maribel Martínez Rodríguez, *Tesis Algoritmos de Búsqueda en Amplitud y Profundidad*, México D.F. Diciembre 2001.

[8] Robert Sedgewick, *Algortimos en C++*, Adisson-Wesley

Índice alfabético