CAPITULO V

CAMINOS DE COSTO MÍNIMO Y GRAFOS DE PRECEDENCIA

INTRODUCCIÓN

En este capítulo trataremos el segundo problema que nos ocupa, presentado en las secciones IV.1.1 y IV.2.2: Dado un grafo dirigido G=(V,E) sin arcos múltiples y D el conjunto de vértices alcanzables desde s, encontrar, si existe, un camino P_w de s a cada vértice w en D tal que el camino de s a w satisfaga la condición $C(P_w)=cmin(s,w)$, donde cmin(s,w) es el costo de un camino mínimo de s a w, entendiéndose por costo C(P) de un camino P a la suma de los costos asociados a los arcos del camino.

En este capítulo se presentan varios algoritmos de búsqueda de caminos de costo mínimo basados en el modelo general de algoritmos de etiquetamiento (capítulo IV). Entre estos algoritmos tenemos el de Dijkstra, la técnica de Programación Dinámica y la de ramificación y acotamiento (Branch-and-Bound). Al final del capítulo estudiamos los grafos de precedencia por su importancia y utilidad en los problemas de caminos. Los algoritmos que presentaremos difieren unos de otros en el criterio de elección del próximo abierto a ser cerrado. Para garantizar que el algoritmo termine, cualquiera sea el criterio de elección, debemos tener la siguiente hipótesis:

$\forall v \in D: cmin(s,v) > -\infty$

Esta hipótesis excluye la posibilidad de tener circuitos de costo negativo en el subgrafo inducido por los vértices alcanzables desde s, si hay circuitos negativos entonces el problema no tendría solución ya que para cualquier N negativo, bastaría recorrer esos circuitos un número finito de veces para obtener C(P) menor que N.

El algoritmo de cálculo de caminos mínimos que deduciremos del modelo general de etiquetamiento se fundamenta en el Principio de Optimalidad que daremos a continuación. Este principio permite tener en la lista a lo sumo un camino hasta cada vértice del grafo de forma tal, que el conjunto de los arcos de estos caminos induzca una arborescencia con raíz s.

Proposición V.1 (Principio de Optimalidad):

Si $C(<n_0,e_1,n_1, ...,e_n,n_p>)=cmin(n_0,n_p)$ entonces $C(<n_i,e_{i+1},n_{i+1}, ... ,n_j>)=cmin(n_i,n_j)$ para $0\le i\le j\le p$.

Demostración:

Basta mostrar que $C(<n_i,e_{i+1},n_{i+1}, ... ,n_j>)\le cmin(n_i,n_j)$.

Supongamos que no:

$$C(<n_i,e_{i+1},n_{i+1}, ... ,n_j>) > cmin(n_i,n_j)$$

Por definición de ínfimo existe un camino $P=<m_1,e'_2,m_2, ...,e'_k,m_k>$ con $m_1=n_i$, $m_k=n_j$ y $C(P)< C(<n_i,e_{i+1},n_{i+1}, ... ,n_j>)$

Por lo tanto

$C(<n_0,e_1, ... ,n_{i-1},e_i,m_1,e'_2, ... ,e'_k,m_k,e_{j+1},n_{j+1}, ... ,n_p>)$

$< C(<n_0,e_1,n_1, ... ,e_p,n_p>)=cmin(n_0,n_p)$

lo que contradice la definición de camino mínimo.

¤

El principio de optimalidad asegura que todo trecho de un camino mínimo es mínimo. Una consecuencia inmediata de la proposición anterior es la siguiente:

Corolario V.2

Si $C(<n_0,e_1,n_1, ...,e_n,n_p>)=cmin(n_0,n_p)$ entonces $cmin(n_0,n_k) + cmin(n_k,n_p) = cmin(n_0,n_p)$ para k=0,1,...,p.

V.1 MODELO DE ALGORITMO DE BÚSQUEDA DE CAMINOS DE COSTO MÍNIMO

Presentamos ahora el modelo general de algoritmo de búsqueda de caminos mínimos basado en el modelo de la sección IV.1.2.

Modelo de algoritmo de búsqueda de caminos de costo mínimo (versión 1):

{ Entrada: Un grafo dirigido *G* sin arcos múltiples, una función de costos *c* de *E(G)* a $\Re$ y un vértice *s*.
Salida: Un conjunto de caminos de costo mínimo desde *s* hasta cada vértice alcanzable desde *s* . }
Comienzo
(0) Abra el camino <*s* > con costo cero;
(1) Mientras Existan caminos abiertos hacer:
Comienzo
(1.1) Escoger un camino abierto P_j=<*s* ,...*nj* >;
(1.2) Cerrar P_j;
(1.3) Obtener los sucesores de *nj* : n^1, n^2,...,n^q;
(1.4) Construir los caminos P^i= Exp(P_j,n^i);
(1.5) A cada camino P^i asocie el costo:

$$c^i = c_j + c\ (n_j\ ,n^i\ );$$

(1.6) Ejecutar la rutina de eliminación;
fin
Donde
Rutina de eliminación:
Comienzo
Para cada P^i, $1 \le i \le q$, hacer:
Si existe un camino listado P_k (abierto ó cerrado) con igual vértice terminal que P^i
entonces
Comienzo
Si el costo de P^i es menor que el costo de P_k entonces eliminar a P_k de la lista y abrir P^i
fin
si no Abrir P^i
fin
fin.

Observaciones:

(1) Obtendremos diferentes algoritmos especificando el paso (1.1).

(2) En la lista de caminos no es necesario almacenar por cada camino la secuencia de vértices que lo definen. Podemos representar a un camino P_j por una terna (n_j,c_j,i_j) donde:

- n_j es el vértice terminal del camino P_j.
- c_j es el costo del camino.
- i_j es un apuntador hacia otro elemento listado tal que P_j=Exp(P_{i_j},n_j).

El inconveniente de esta representación recursiva de caminos es la posibilidad que existe de que se pierdan trechos de caminos a medida que el algoritmo avanza pues los cerrados pueden ser eliminados de la lista por la rutina de eliminación, tal como se ilustró en la sección IV.1.2 propiedad (4). Sin embargo, los criterios de elección de los abiertos a ser cerrados, que discutiremos mas adelante, nos garantizarán que en esos casos los caminos cerrados no podrán ser eliminados. Veamos la versión del modelo de algoritmo con esta representación de caminos:

Modelo de algoritmo de búsqueda de caminos de costo mínimo (versión 2):

{ Entrada: Un grafo dirigido *G* sin arcos múltiples, una función de costos *c* de *E(G)* a $\Re$ y un vértice *s*.
Salida: Un conjunto de caminos de costo mínimo desde *s* hasta cada vértice alcanzable desde *s* . }

Comienzo
(0) Abra el elemento $P_0=(s,0,*)$;
(1) Mientras Existan elementos abiertos hacer:
Comienzo
(1.1) Escoger un elemento abierto $P_j=(n_j,c_j,i_j)$;
(1.2) Cerrar P_j;
(1.3) Obtener los sucesores de nj : $n^1, n^2,...,n^q$;
(1.4) Construir los elementos $P^i=(n^i,c^i,j)$ donde:

$$c^i = c_j + c\,(n_j, n^i)\,;$$

(1.5) Ejecutar la rutina de eliminación;
fin
Donde
Rutina de eliminación:
Comienzo
Para cada P^i, $1 \le i \le q$, hacer:
Si existe un elemento listado $P_k=(n_k,c_k,i_k)$ (abierto ó cerrado) con $n_k=n^i$ entonces
Comienzo
Si el costo de P^i es menor que el costo de P_k entonces eliminar a P_k de la lista y abrir P^i
fin
si no Abrir P^i
fin
fin

(3) Daremos varias propiedades del modelo de algoritmo anterior, siendo la más importante el hecho de que el modelo resuelve el problema de la arborescencia de caminos mínimos, es decir, el modelo garantiza que termina y al terminar, los caminos cerrados constituyen caminos de costo mínimo desde s a cada vértice alcanzable desde s y el conjunto de los arcos de estos caminos induce una arborescencia en el grafo. A pesar de que se pueden perder trechos de caminos en la versión 2 del algoritmo, veremos que nunca se perderán trechos de caminos óptimos.

(4) En adelante trabajaremos con la versión 2 del algoritmo.

Proposición V.1.1

Un vértice no puede ser vértice terminal en dos elementos listados.

Demostración:

Supongamos por el absurdo que existen dos elementos listados con un mismo vértice en una iteración cualquiera del algoritmo. Considere la primera iteración al final de la cual un vértice n está en dos elementos listados P y $\overline{P}$ En esta iteración P ó $\overline{P}$ fue obtenido por expansión del camino que se cerró en esta iteración y por lo tanto tuvo que ser comparado con el otro camino en la rutina de eliminación: la contradicción se obtiene de saber que la rutina de eliminación dejará sólo uno de los dos caminos en la lista.

Por lo tanto, la lista siempre contendrá una representación de caminos, ternas, cada uno terminando en un vértice diferente.

¤

Veamos ahora que, aunque se pueden perder trechos de caminos con la representación dada por las ternas, nunca se perderán trechos de caminos óptimos.

Decimos que un elemento listado P=(n,c,p) es mínimo si c=cmin(s,n).

Proposición V.1.2

Un elemento listado mínimo nunca puede ser eliminado y un sucesor mínimo (resultante de la expansión de un camino cerrado) sólo puede ser eliminado por un elemento listado mínimo.

Demostración:

Para eliminar un elemento listado (n,cmin(s,n),.), sería necesario un sucesor (n,c,p) con $c<cmin(s,n)$. Por definición de cmin(s,n), obligatoriamente $c \ge cmin(s,n)$, ya que c es el costo de un camino de s a n. Si c=cmin(s,n)

vemos que el sucesor es mínimo y es eliminado por un elemento listado mínimo.

¤

Proposición V.1.3

Sea $P_j=(n_j,c_j,p)$ mínimo, apuntando hacia $P_p=(n_p,c_p,.)$. Entonces P_p es mínimo.

Demostración:

Por el principio de optimalidad, corolario V.2, se tiene que $cmin(s,n_p) + cmin(n_p,n_j) = cmin(s,n_j)$. Como $c_j=cmin(s,n_j)$, entonces $c(n_p,n_j)=cmin(n_p,n_j)$ y $c_j=c(n_p,n_j) + c_p$ entonces $c_p= cmin(s,n_p)$.

¤

Proposición V.1.4

Si el algoritmo coloca en la lista un elemento mínimo, el camino correspondiente se puede construir siguiendo los apuntadores.

Demostración:

La proposición V.1.3 nos dice que los caminos obtenidos siguiendo apuntadores son mínimos y de acuerdo a la proposición V.1.2 ellos nunca son eliminados.

¤

La proposición siguiente garantiza que se encontrarán caminos mínimos hasta cada vértice alcanzable desde s pues de no haberse encontrado todavía en una iteración cualquiera un camino mínimo hasta un vértice, todos los caminos mínimos hasta ese vértice están siendo construidos por el algoritmo.

Proposición V.1.5

Considere el algoritmo al iniciar el paso (1.1), en una iteración cualquiera. Sea n un vértice cualquiera y $<n^0,e^1,n^1,...,e^p,n^p>$, $n^0=s$, $n^p=n$, un camino mínimo cualquiera de s a n (que suponemos existente). Entonces se tiene el invariante:

Si no existe un cerrado de la forma $(n,cmin(s,n),.)$ entonces existe un abierto de la forma $(n^k,cmin(s,n^k),.)$, con $0\leq k\leq p$.

Demostración:

En la primera iteración (s,0,*) está abierto.

En una iteración cualquiera distinta de la primera, supongamos que no existe un cerrado $(n,cmin(s,n),.)$. Sea k-1 el índice más alto tal que $(n^{k-1},cmin(s,n^{k-1}),.)$ está cerrado (k-1, con $0\leq k-1<p$, siempre existe pues al menos (s,0,*) está cerrado).

Como $(n^{k-1},cmin(s,n^{k-1}),.)$ está cerrado, él fue expandido en una iteración anterior, generando un sucesor mínimo $(n^k,cmin(s,n^k),.)$ pues $cmin(s,n^k)=cmin(s,n^{k-1}) + c(n^{k-1},n^k)$. Si este sucesor no fue eliminado entonces fue abierto y si fue eliminado tuvo que ser eliminado por un abierto mínimo $(n^k,cmin(s,n^k),p^k)$. En cualquier caso fue listado como abierto y no puede ser eliminado según la proposición V.1.2. Por lo tanto permanece abierto completando así la demostración.

¤

Finalmente damos la proposición que garantiza que el algoritmo termina en un número finito de iteraciones. Al terminar habrá en la lista un camino mínimo hasta cada vértice alcanzable desde s y los arcos de estos caminos inducen una arborescencia.

Proposición V.1.6

Si G es un grafo finito y satisface la hipótesis: $\forall$v alcanzable desde s $[cmin(s,v) > -\infty]$, entonces:

(a) El algoritmo termina en un número finito de iteraciones.

(b) Cuando el algoritmo termina, hay un elemento cerrado mínimo para cada vértice alcanzable desde s y por otro lado, los arcos de los caminos obtenidos a partir de los elementos listados, inducen una arborescencia de

caminos mínimos en G, cuya raíz es s.

Demostración:

Primero observamos que un camino correspondiente a un elemento listado es elemental: Sabemos que si hay un camino listado con vértice terminal n, otro camino con vértice terminal n sólo podrá ser abierto (reemplazando al anterior) si tiene costo menor estricto que el del elemento listado. Por lo tanto si el algoritmo expande un camino con vértice terminal n y pasando por n, P=<s,...,n,...,n>, sabemos que el costo c' del primer trozo <s,...,n> del camino P debe ser menor o igual que el costo de P pues <n,...,n> (el segundo trozo de P) no puede tener costo negativo por hipótesis.

(a) Cada elemento listado representa un camino elemental. El mismo camino no puede ser listado dos veces. El número de caminos elementales en un grafo finito es finito ($\leq |V(G)|!$) y por lo tanto el número de iteraciones es finito pues en cada iteración se cierra un abierto y los caminos cerrados no se vuelven a abrir.

(b) Sea n alcanzable desde s. Si (n,cmin(s,n),.) no está cerrado, la proposición V.1.5 garantiza que hay abiertos y que el algoritmo no puede parar. Por lo tanto, cuando el algoritmo termina, cada cerrado corresponde a un camino mínimo que puede ser recuperado por los apuntadores, es decir, se obtiene una arborescencia.

¤

(5) Si en el subgrafo generado por los vértices alcanzables desde s hay circuitos de costo negativo, este hecho puede ser detectado por el algoritmo de la manera siguiente: cuando en la rutina de eliminación conseguimos que un camino expandido elimina a uno listado, en ese momento verificamos si el vértice terminal del camino expandido aparece dos veces en el camino. Si este es el caso, entonces el trozo del camino expandido entre las dos ocurrencias del vértice es un circuito de costo negativo.

V.2 ALGORITMOS PARTICULARES DE BÚSQUEDA DE CAMINOS DE COSTO MÍNIMO

En la sección anterior presentamos un modelo general de algoritmo de búsqueda de caminos mínimos basado en el modelo de etiquetamiento. En esta sección presentamos casos particulares de este modelo. Cada algoritmo estará caracterizado por la regla de elección del próximo abierto a ser cerrado. Entre los algoritmos que veremos están: el de Dijkstra, la técnica Branch and Bound y el algoritmo de Bellman (técnica de la Programación Dinámica) para clases de grafos particulares. En cada caso discutiremos la complejidad en tiempo y consideraciones de espacio de almacenamiento. Por último presentamos el algoritmo de Floyd para el cálculo de los caminos mínimos entre cada par de vértices; este algoritmo es una generalización del algoritmo de Roy-Warshall utilizado en el capítulo III para determinar la matriz de alcance.

Muchas veces interesa determinar el mejor camino desde un vértice s hasta algún vértice del conjunto O que llamaremos *conjunto objetivo*. Si éste es el caso, podríamos agregar una condición de parada a nuestro modelo general, de tal forma que cuando entre los cerrados exista uno con vértice terminal en O entonces termine la ejecución del algoritmo. Podemos suponer que cuando O es vacío, estamos en el problema original de búsqueda de caminos a todos los alcanzables desde el vértice de partida.

Consideraremos que tenemos en esta sección un conjunto objetivo O.

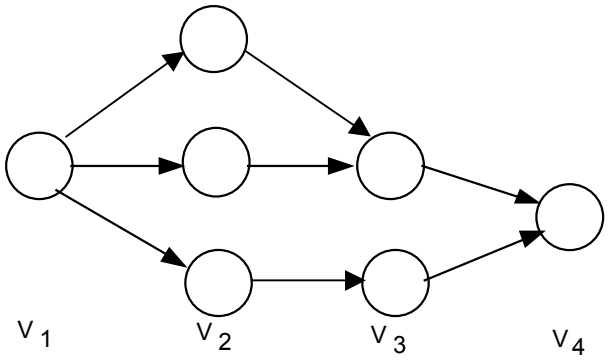
Hablamos de *algoritmos de búsqueda no informados* cuando éstos no utilizan información contenida en los elementos listados con el fin de alcanzar más rápido el objetivo, es decir, ninguna información en cuanto a la proximidad al objetivo es utilizada para guiar la elección de los elementos a expandir (y cerrar) en cada iteración.

Los algoritmos no informados más conocidos son los algoritmos de búsqueda en amplitud (técnica de Programación Dinámica Progresiva), la búsqueda en profundidad (técnica de enumeración implícita no informada o "Backtracking") y el algoritmo de Dijkstra que se utiliza en grafos con costos no negativos.

V.2.1 ALGORITMO DE BÚSQUEDA EN AMPLITUD PARA CAMINOS DE COSTO MÍNIMO

Este algoritmo se obtiene del modelo general aplicando la regla siguiente para escoger el próximo camino a cerrar: Escoja el primer abierto que fue colocado en la lista entre todos los abiertos. Como vimos en la sección IV.2.7, a pesar de que la regla de eliminación varía con respecto a la presentada en esa sección, en la lista los abiertos siguen poseyendo en cada iteración largo k ó k+1 para algún k. Esto se puede interpretar diciendo que el algoritmo explora el grafo por etapas, en una etapa se consideran todos los caminos de un mismo largo k, en la etapa siguiente se consideran los caminos de largo k+1 y así sucesivamente.

Decimos que un grafo G=(V,E) es *en camadas* si V puede ser particionado en conjuntos V_i, i=1,...,q, de modo que si $v \in V_i$ entonces los sucesores de V están en V_{i+1} (ver figura V.1).



Grafo en Camadas

figura V.1

El algoritmo de búsqueda en amplitud aplicado a un grafo en camadas se convierte en una técnica muy conocida de búsqueda de estrategias óptimas en problemas de decisiones secuenciales; esta técnica se denomina Programación Dinámica Progresiva. Las características del algoritmo aplicado a este tipo de grafos, son las siguientes:

- Se cierran todos los vértices de una camada antes de comenzar a cerrar los de la camada siguiente. Al terminar de cerrar los de una camada, están abiertos todos los vértices de la siguiente.

- Los cerrados no son eliminados. Esto garantiza que el número de iteraciones del algoritmo es igual al número de vértices alcanzables desde s. Se podría implementar el algoritmo de manera que en cada iteración el número de operaciones realizables sea de orden $\Delta^+(G)$ (grado máximo exterior de G); bastaría con hacer algunas modificaciones al algoritmo de visita de vértices por búsqueda en amplitud presentado en la sección IV.2.7.

- Se tendrá en memoria sólo los elementos en la camada que se está expandiendo y la camada resultante de la expansión.

V.2.2 ALGORITMO DE BÚSQUEDA EN PROFUNDIDAD PARA CAMINOS DE COSTO MÍNIMO

Este algoritmo se obtiene del modelo general, aplicando la regla siguiente de elección del próximo abierto a ser cerrado: escoger para cerrar y expandir al último camino que se haya abierto (último que entra primero que sale).

Como vimos en las propiedades (8) y (9) de la sección IV.2.1, en cada iteración se cierra un abierto de largo o profundidad máxima. Si se abre alguno de los caminos acabados de expandir en una iteración, en la siguiente uno de estos caminos expandidos será cerrado por el algoritmo, continuando de esta forma hasta alcanzar un vértice que no posee sucesores o que todos los caminos resultantes de la expansión hayan sido eliminados por la rutina de eliminación. En cualquiera de los casos el siguiente camino a ser cerrado será el de largo máximo entre los abiertos.

En cada iteración la lista tiene un formato muy particular. Si P es el camino a expandir en una iteración cualquiera y $P_0,P_1,...,P_p$ es la secuencia de caminos cerrados a partir de P siguiendo los apuntadores, entonces todos los abiertos apuntan a uno de los caminos en la secuencia (ver figura IV.2.1.3). Por lo tanto, si el largo máximo de un camino elemental de G es p, el número de abiertos no supera a $(\Delta^+(G)-1)p+1$.

En problemas con objetivo, si el objetivo es un conjunto numeroso, el algoritmo de búsqueda en profundidad podría ser utilizado para conseguir rápidamente soluciones sub-óptimas. Estas soluciones pueden ofrecer información para la construcción de algoritmos informados. Esta última técnica se conoce en la literatura como algoritmos de "Branch-and-Bound".

V.2.3 ALGORITMO DE DIJKSTRA

Los dos algoritmos estudiados anteriormente tienen la ventaja de utilizar "poca" memoria. Sin embargo son

poco eficientes pues no toman en consideración los costos y exploran indiscriminadamente todos los caminos que parten de s. En el problema de la arborescencia de caminos mínimos, los caminos que más prometen ser óptimos entre los abiertos son aquéllos de costo mínimo. El algoritmo de Dijkstra se obtiene estableciendo el siguiente criterio de elección de abiertos a cerrar: escoger el abierto de menor costo, de esta manera la exploración del grafo será hecha en camadas de caminos de costo similar salvo en el caso de que existan costos negativos.

Cuando los costos son no negativos, este criterio de elección implicará resultados importantes.

Sea G un grafo orientado donde todo arco tiene asociado un costo no negativo.

Proposición V.2.1

Suponga que un algoritmo cualquiera, según el modelo, es aplicado a G a partir de s. Al comienzo de cualquier iteración si $P=(n^*,c^*,p^*)$ es un abierto con $c^*=\min\{c/(n,c,p)$ es un abierto$\}$ entonces $c^*=cmin(s,n^*)$.

Demostración:

Supongamos por el absurdo que $c^* > cmin(s,n^*)$. Sabemos que no existe un cerrado con vértice terminal n^* y, por la proposición V.1.5, existe un abierto (n,cmin(s,n),.) donde n es un vértice intermedio de un camino óptimo de s a n^*.

Por otro lado, $cmin(s,n) \leq cmin(s,n^*)$ pues los costos son no negativos. Así $cmin(s,n) < c^*$ lo que contradice la definición de c^*.

¤

Las consecuencias inmediatas de esta proposición son las siguientes:

(1) Cuando el algoritmo de Dijkstra es utilizado con un grafo de costos no negativos, cualquier cerrado P=(n,c,p) satisface c=cmin(s,n). Esto implica que los cerrados no son eliminados y así el número de iteraciones del algoritmo es igual al número de vértices alcanzables desde s.

(2) Los costos de los elementos cerrados por el algoritmo de Dijkstra, en grafos con costos no negativos, crecen con las iteraciones del algoritmo. Por lo tanto, si el problema incluye un conjunto objetivo O, el primer camino cerrado con vértice terminal en el objetivo es una solución del problema, es decir, es el camino de menor costo desde s a cualquier vértice del objetivo.

Note que el algoritmo de búsqueda en amplitud no es más que un caso particular del algoritmo de Dijkstra, escogiendo para cerrar al abierto de longitud mínima. En este caso el costo de cada arco es 1.

Como dos caminos listados no pueden tener igual vértice terminal, podemos hacer una implementación del algoritmo de Dijkstra de manera que a cada vértice del grafo le asociamos: la etiqueta "cerrado" o "abierto", un apuntador al vértice predecesor en el camino y el costo del camino. La diferencia entre esta implementación y las ternas (n,c,p) es que p ya no apunta a otra terna sino a un vértice al cual podemos tener acceso inmediato, lo cual nos permite reducir el número de operaciones de la rutina de eliminación a $O(\Delta^+(G))$.

A continuación damos la implementación del algoritmo de Dijkstra según los comentarios anteriores.

Algoritmo de Dijkstra:

{ Entrada: Un digrafo *G* sin arcos múltiples. Una función de costos $c: E(G) \rightarrow \Re$, con $c(e) \geq 0 \ \forall e \in E(G)$. Y *s* un vértice de *G*.

Salida: Un apuntador (o referencia a un vértice) y un costo asociados a cada vértice alcanzable desde *s*. El camino de costo mínimo hasta un determinado vértice puede ser reconstruido siguiendo los apuntadores. Los vértices que resulten con costos $+\infty$ son los no alcanzables desde *s* }

Variable *n,m* : vértice;
Comienzo
(1) Inicialmente *s* está abierto, tiene costo cero y apuntador nulo. Todos los demás vértices están abiertos y tienen costo $+\infty$ (un valor muy grande);
(2) Mientras Existan vértices abiertos hacer:
Comienzo
(2.1) Escoger un vértice abierto *n* con el menor costo;
(2.2) Cerrar *n*;
(2.3) Para cada vértice abierto *m* sucesor de *n* hacer:
Si el costo de $n + c(n,m)$ es menor que el costo de *m* entonces

asociar a *m* un apuntador hacia *n* y reemplazar el costo de *m* por el costo de *n* + *c* (*n*,*m*)

fin

fin

Por convención el resultado de las sumas con algún operando igual a $+\infty$ seguirán siendo $+\infty$. Note que el conjunto de vértices abiertos puede ser representado por una cola de prioridades (ver VII.4.1) y el grafo por una lista de adyacencias. De esta forma el paso (2.1) es O(1), el paso (2.2), que significa eliminar el primer vértice de la cola de prioridades, es O($\log_2$ (número de elementos en la cola)), si la cola se implementa como un "heap" binario en un arreglo.

En el paso (2.3) se efectuarán $d^+(n)$ iteraciones. En cada iteración, si el costo de m varía entonces se debe reestructurar la cola de prioridades y esta operación es O($\log_2$(número de elementos en la cola)). El número de iteraciones de (2) es igual al número de vértices del grafo y, en cada iteración, el número de elementos de la cola de prioridades se reduce en una unidad. Por lo tanto la complejidad en tiempo del algoritmo de Dijkstra sería:

$$\sum_{v\in V(G)} (c_1*\log_2(|V(G)|)+d^+(v)*c_2*\log_2(|V(G)|)) \le O(\max(|V(G)|,|E(G)|)*\log_2(|V(G)|))$$

Así conseguimos una implementación O(max(|V|,|E|).$\log_2$|V|) del algoritmo de Dijkstra.

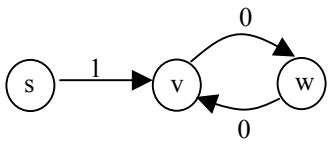
V.2.4 ALGORITMO DE BELLMAN

El algoritmo de Bellman permite calcular todos los caminos mínimos que parten de un vértice s. Se supone que el vértice s es raíz del grafo y que el grafo no posee circuitos. Este algoritmo se puede ver como una generalización del algoritmo de búsqueda en amplitud en grafos en camadas y puede verse como una generalización de la técnica de Programación Dinámica Progresiva para el cálculo de caminos mínimos desde s. Esta técnica está basada en la fórmula recursiva siguiente:

Si C(v) representa el costo de un camino mínimo desde s hasta v:

C(v) = min {C(w) + c(w,v) / w es predecesor de v}

Note que si el grafo tiene circuitos entonces la fórmula recursiva anterior no funcionaría, como se puede comprobar con en el grafo siguiente:



El algoritmo de Bellman es también un caso particular del modelo de algoritmo de búsqueda de caminos mínimos, bajo la suposición de que s sea una raíz de G. La regla de elección del próximo abierto a ser cerrado es : "Escoger un abierto tal que haya un camino listado cerrado hasta cada predecesor del vértice terminal de ese camino abierto". Obsérvese que la hipótesis del modelo general: cmin(s,v) > $-\infty$, $\forall v\in V(G)$, se satisface para grafos sin circuitos. La regla de elección garantiza que un camino cerrado no es eliminado y que en cada iteración exista un abierto con todos sus predecesores como vértices terminales de caminos cerrados. Esto se debe a que si $m_1,...,m_p$ son los predecesores de n entonces todo camino de s a m_i , $\forall i$, no pasa por n ni por ningún ascendiente de n por ser G sin circuitos.

Si s no es raíz de G podremos aplicar el algoritmo de Bellman de la siguiente forma: se determinan los vértices alcanzables desde s y la regla de elección sería “escoger un abierto tal que exista un camino listado cerrado hasta cada predecesor, alcanzable desde s, del vértice terminal del abierto”.

Una implementación elegante y simplificada del algoritmo de Bellman es la siguiente:

Suponga que s es raíz de G. G viene representado por una lista de adyacencias. A cada vértice v asociamos su grado interior o de entrada *grado(v)* =d^-(v), el costo *costo(v)* del camino mínimo de s a v y un apuntador al predecesor en el camino. Inicialmente T={s}, costo(s)=0, costo(v)=$+\infty$ $\forall v\neq s$, *Apuntador(s)* =NULO.

En cada iteración se toma un elemento n de T y se elimina de T. Para cada sucesor m de n se efectúa lo siguiente:

- grado(m)←grado(m)-1.
- Si grado(m)=0 entonces T←T∪{m}
- Si costo(m) > costo(n) + c(n,m) entonces
  costo(m)←costo(n) + c(n,m) y Apuntador(m)←n

Algoritmo de Bellman:
{ Entrada: Un digrafo *G* sin circuitos y una raíz *s*.
Salida: Para cada vértice v de G un apuntador al predecesor en un camino mínimo de s a v y el costo de ese camino hasta v.}
Variable *T* : conjunto de vértices;
s,*v*,*n*,*m* : vértice;
Comienzo
(1) $T \leftarrow \{s\}$;
(2) costo(*s*)←0; Apuntador(*s*)←NULO;
(3) $\forall v \in V(G), v \neq s$: costo(*v*)←$+\infty$;
(4) Mientras $T \neq \emptyset$ hacer:
Comienzo
Tomar un vértice *n* de *T* ;
$T \leftarrow T - \{n\}$;
Para cada sucesor *m* de *n* hacer:
Comienzo
grado(*m*)←grado(*m*)-1;
Si grado(*m*)=0 entonces $T \leftarrow T \cup \{m\}$;
Si costo(*m*)>costo(*n*)+c(*n*,*m*) entonces
Comienzo
costo(*m*)←costo(*n*)+c(*n*,*m*);
Apuntador(*m*)← *n*;
fin
fin
fin
fin.

El cuerpo del paso (4) se realiza una sola vez por cada vértice de G y en cada iteración el número de operaciones es $O(\max(1,d^+(n))$. Por lo tanto este algoritmo es $O(\max(|V(G)|, |E(G)|)$.

Los costos de los arcos pueden ser negativos al aplicar el algoritmo de Bellman, debido a que el grafo no posee circuitos y en estos grafos se cumple la hipótesis $cmin(s,v) > -\infty$,$\forall$ $v \in V(G)$. Esto nos sugiere la posibilidad de hallar con el mismo algoritmo un camino de costo máximo; basta con cambiar el signo a los costos de los arcos.

V.3 CAMINOS MÍNIMOS ENTRE CADA PAR DE VÉRTICES

En esta sección suponemos que todo circuito tiene costo no negativo. Sean $v_1,v_2,...,v_n$ los vértices de un digrafo G. Supongamos que conocemos un camino mínimo desde cualquier vértice v a cualquier vértice w de G entre todos los caminos de v a w cuyos vértices interiores (es decir, vértices por donde pasa el camino y distintos de v y w) están en $\{v_1,...,v_i\}$. Podemos suponer que estos caminos son elementales pues todo circuito tiene costo no negativo. Un camino elemental de costo mínimo P de v a w, entre todos los caminos de v a w cuyos vértices interiores están en $\{v_1,...,v_{i+1}\}$ cumple con una de las dos propiedades siguientes:

(a) Ningún vértice interior de P es igual a v_{i+1}. En cuyo caso P es un camino mínimo de v a w entre todos los caminos de v a w cuyos vértices interiores están en $\{v_1,...,v_i\}$.

(b) Si v_{i+1} es un vértice interior de P, P=<v,...,v_{i+1},...,w>, sean Q=<v,...,v_{i+1}> y R=<v_{i+1},...,w> con P=Q || R. Entonces los vértices interiores de Q y R están en $\{v_1,...,v_i\}$ pues P es elemental. Q (R) es un camino mínimo de v a v_{i+1} (de v_{i+1} a w) entre todos los caminos de v a v_{i+1} (de v_{i+1} a w) cuyos vértices interiores están en $\{v_1,...,v_i\}$. Esto último se debe al principio de optimalidad (Proposición V.1).

Las dos propiedades anteriores indican que un camino mínimo P de v a w, entre todos los caminos de v a w con vertices interiores en $\{v_1,...,v_{i+1}\}$, se puede construir a partir de los caminos mínimos con vértices interiores en $\{v_1,...,v_i\}$. Lo hacemos comparando el costo de un camino mínimo S de v a w con vértices interiores en $\{v_1,...,v_i\}$ y el costo de un camino T de v a w pasando por v_{i+1}. Este último costo es la suma del costo de un camino mínimo Q de v a v_{i+1} con vértices interiores en $\{v_1,...,v_i\}$ más el costo de un camino mínimo R de v_{i+1} a w con vértices interiores en $\{v_1,...,v_i\}$. Si costo(S) < costo(Q)+costo(R) entonces P=S, si no P=Q $\|$ R. Note que costo(P) es igual al costo de un camino mínimo pasando por $\{v_1,...,v_{i+1}\}$.

Hemos así conseguido un método recursivo para calcular un camino mínimo de v a w, $\forall v,w \in V(G)$. Cuando $i=|V(G)|$ habremos conseguido un camino mínimo de v a w, $\forall v,w \in V(G)$. Este algoritmo se conoce como Algoritmo de Floyd. El método es similar al algoritmo de Roy-Warshall para calcular la matriz de alcance (capítulo III).

Inicialmente cuando i=0, es decir el conjunto de vértices interiores es vacío, los caminos iniciales son los arcos del grafo. Por lo tanto podemos partir de una matriz $C=(c_{ij})$, $|V(G)|x|V(G)|$, donde $c_{ij}=c(i,j)$ si $(i,j)\in E(G)$ ó $c_{ij}=+\infty$ si $(i,j)\notin E(G)$.

Haremos una implementación del algoritmo donde sólo calcularemos los costos de los caminos mínimos entre cada par de vértices, sin calcular los caminos en sí. El cálculo de los caminos puede hacerse sin necesidad de guardar para cada par de vértices la secuencia de vértices que lo define, basta tener una matriz A, $|V(G)|x|V(G)|$, de índices, donde el elemento a_{ij} puede contener tres tipos de valores:

- un índice de un vértice intermedio del camino de i a j.
- un indicador de que (i,j) es un arco.
- un indicador de que no hay camino de i a j.

Recursivamente se podrá construir el camino de i a j pues este será:

- (El camino de i a a_{ij}) $\|$ (El camino de a_{ij} a j) ó
- <i,(i,j),j> si (i,j) es un arco ó
- No hay camino de i a j.

Algoritmo de Floyd:

{ Entrada: Un digrafo *G* sin arcos múltiples. Una función de costos no negativos c :E(G)$\rightarrow\Re^{\geq 0}$. El número *n* de vértices del grafo. Una matriz $C=(c_{ij})$ con $c_{ij}=c(i,j)$ =costo del arco *(i,j)*. Si no existe arco de *i* a *j*, c_{ij} es $+\infty$, si $i=j$ entonces $c_{ij}=0$.

Salida: Matriz $C=(c_{ij})$ con c_{ij} =costo de un camino de costo mínimo de *i* a *j* o $+\infty$ si no existe camino.}

Variables *i,j,k* : índice de vértice;
Comienzo
  Para $k \leftarrow 1$ a *n* hacer:
    Para $i \leftarrow 1$ a *n* hacer:
      Si $(C[i,k] < +\infty)$ y $i \neq k$ entonces
        Para $j \leftarrow 1$ a *n* hacer:
          Si $(j \neq k) \wedge (i \neq j) \wedge (C[k,j] < +\infty)$ entonces $C[i,j] \leftarrow \min(C[i,j], C[i,k]+C[k,j])$
fin.

V.4 ALGORITMOS INFORMADOS DE BÚSQUEDA DE CAMINOS MÍNIMOS

En esta sección continuamos tratando el problema de encontrar un camino de costo mínimo desde un vértice s, de un grafo dirigido sin arcos múltiples, hasta un conjunto O de vértices que hemos llamado conjunto objetivo. Cada arco (v,w) tendrá un costo asociado c(v,w).

Definición V.4.1

Si A y B son dos conjuntos de vértices de un grafo orientado G=(V,E), definimos *el costo mínimo para ir de* A *a* B como:

h(A,B) = ínfimo {C(P) / P es un camino con extremo inicial en A, extremo final en B, y cualquier otro

vértice distinto de los extremos de P no está ni en A ni en B}.

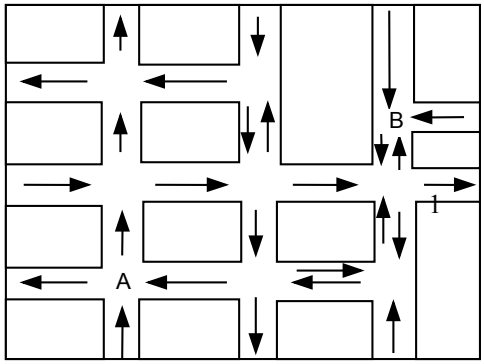
donde C(P) es el costo del camino P. Por convención ínfimo($\varnothing$) = $+\infty$.

Dado un conjunto objetivo O y un conjunto de vértices A, definimos:

h(A) = h(A,O) y g(A) = h({s},A). Si A = {n} escribimos h(n) y g(n) para simplificar la notación..

El problema de búsqueda con conjunto objetivo se formula de la siguiente forma:

Encontrar un vértice $n_0 \in O$ y un camino P de s a n_0 tal que C(P) = h(s) (= g(O)). Si O es vacío supondremos que el problema es determinar la arborescencia de caminos mínimos que parten de A.

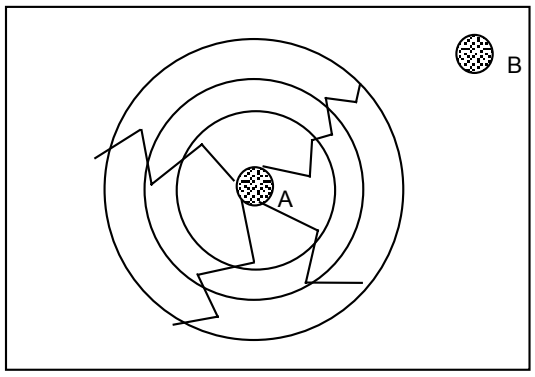


Mapa de la Ciudad

A: lugar de llegada B: lugar de partida

figura V.2

LLamaremos *algoritmos informados* a aquellos que poseen una información adicional a la topología del grafo, que permita alcanzar más rápido el objetivo O con un camino mínimo de s a O. Esta información adicional podría ser un estimado de h(n), $\forall n \in V(G)$. En la sección anterior todos los algoritmos vistos eran no informados pues los criterios de elección de caminos a expandir no toman en cuenta para nada la proximidad del vértice terminal del camino hasta el objetivo.



El algoritmo de Dijkstra considera los caminos
por "ondas" de costo similar

figura V.3

Un algoritmo informado permitiría llegar más rápido al objetivo, como lo muestra el ejemplo siguiente: suponga que usted se encuentra en un lugar A de una ciudad y desea desplazarse a otro lugar B de la ciudad. Usted

cuenta con un plano de la ciudad donde aparece por cada calle y avenida, la cantidad de metros entre intersecciones consecutivas. Además el mapa está cuadriculado de forma que podemos conocer las coordenadas de cada lugar en el mapa. En la figura V.2 se ilustra un mapa. Si usted desea conocer cuál es el camino a menor distancia para llegar a su objetivo podría emplear por ejemplo el algoritmo de Dijkstra. Este algoritmo avanza hacia el objetivo por etapas calculando todos los caminos de costo homogéneo hasta llegar al objetivo (ver figura V.3). Sin embargo, intuitivamente vemos que se puede mejorar la búsqueda tomando en cuenta la distancia euclideana entre el lugar de partida y el objetivo; es natural expandir primero aquellos caminos donde la suma del costo del camino con la distancia euclideana entre el vértice terminal del camino y el objetivo sea mínima. En la figura V.4 es preferible expandir el camino S antes que T pues S es más prometedor que T.

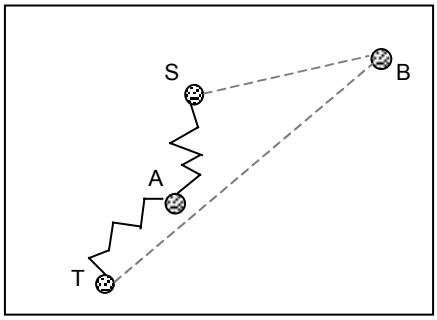


figura V.4

Supongamos que conocemos alguna información sobre h(n). Esta información puede ser utilizada de la manera siguiente:

(1) Si se conoce algún camino C de s a O, su costo puede ser utilizado como una cota superior en la búsqueda de nuevos caminos. Si el costo de un camino listado P con vértice terminal n, sumado a un subestimado(cota inferior) de h(n), supera al costo del camino C, P no puede ser parte del un camino mínimo hasta el objetivo y puede ser eliminado. Este método genera una familia de técnicas conocidas como "Branch-and-Bound" (en [4] se encuentra más información sobre la técnica “brach-and-bound”).

(2) Si conocemos un estimado de h(n), $\forall n \in V(G)$, esta estimativa puede ser utilizada para establecer las prioridades al momento de escoger un abierto a cerrar en el paso (1.1) del modelo general de algoritmo presentado en la sección V.1. El método resultante se conoce como *algoritmo A**.

En lo que sigue describiremos el algoritmo A* y sus propiedades, en función de las propiedades de una estimativa de h(.).

En esta sección suponemos que los grafos no poseen circuitos de costo negativo, es decir, $cmin(s,n) > -\infty$, $\forall n \in V(G)$. Supongamos que al grafo está asociada una función :

$$\hat{h} : V(G) \rightarrow \Re$$

Utilizaremos la versión 2 del modelo general de algoritmo de búsqueda de caminos mínimos que representa los caminos por ternas P=(n,c,p) y definimos:

$$\hat{f}(P) = c + \hat{h}(n)$$

Si $\hat{h}$ es una estimativa de h, entonces $\hat{f}$ es una estimativa del costo de un camino de costo mínimo hasta el objetivo O, restringido a tener su parte inicial igual a P. Así, $\hat{f}(P)$ estima lo que se puede esperar de la expansión del camino P. La regla de elección del próximo abierto a cerrar y expandir: "escoger un abierto P con menor $\hat{f}(P)$", da preferencia a los caminos abiertos más prometedores con la información que se dispone.

A continuación presentamos el algoritmo A*.

Algoritmo A*:

{ Entrada: Un digrafo *G* sin arcos múltiples, con costos en los arcos y sin circuitos de costo negativo. Un vértice *s* de G. $\forall n \in V(G)$ [$cmin(s,n) > -\infty$]. Una función estimativa $\hat{h}$ de h. Un conjunto objetivo *O*.

Salida: Un camino P de *s* a *O* (si existe) que satisface el criterio de parada. Si no existe tal camino, P

contendrá la secuencia vacía.}
Comienzo

(0) Abrir el elemento $P_0=(s,0,*)$ y asociarle $\hat{f}(P_0) = \hat{h}(s)$. Inicialmente no se ha satisfecho el criterio de parada y P es la secuencia vacía;
(1) Mientras Existan elementos abiertos y no se haya satisfecho el criterio de parada hacer:
Comienzo
(1.1) Escoger un elemento abierto $P_j=(n_j,c_j,p_j)$ tal que $\hat{f}(Pj)$ sea mínimo entre los abiertos;`
(1.2) Cerrar P_j;
(1.3) Si $n_j \in O$
entonces Si P_j satisface el criterio de parada entonces asignar a P el camino P_j
si no
Comienzo
- Obtener los sucesores de n_j: $n^1,...,n^q$;
- Construir los caminos expandidos $P^i=(n^i,c^i,j)$, i=1,2,...,q,, de P_j a cada sucesor de n_j donde $c^i=c_j+c(n_j,n^i)$;
- Asociar a cada camino P^i el valor: $\hat{f}(P^i) = c^i + \hat{h}(n^i)$;
- Aplicar la rutina de eliminación a los P^i (la versión 2 del modelo general de la sección V.1);
fin
fin
fin.

Observaciones:

- El criterio de parada debe cumplir con la restricción de que sólo puede ser satisfecho cuando se cierra algún camino (el que permite satisfacer el criterio de parada) cuyo vértice final esté en el objetivo O. Por ejemplo, un criterio de parada podría ser "el primer camino que se cierre y que llegue al objetivo" , otro criterio podría ser "cuando el costo del camino que se cierre y llega al objetivo, sea menor que una cierta cantidad preestablecida".
- Si el conjunto objetivo no es alcanzable desde s entonces el algoritmo A* termina con una arbor4escencia de caminos mínimos hasta los vértices alcanzables desde s y P será la secuencia nula.

La proposición V.1.6 garantiza que A* termina con una arborescencia de caminos mínimos hasta los vértices alcanzables desde s si no hay criterio de parada ni conjunto objetivo.

Definición V.4.2

Una estimativa $\hat{h}$ se llama *admisible* si $\forall n \in V(G)$: $\hat{h}(n) \leq h(n)$, es decir, $\hat{h}$ es una subestimativa de h.

Proposición V.4.1

Supongamos que $\hat{h}$ es admisible y el objetivo es alcanzable desde s.
Si al comienzo de una iteración cualquiera no se ha cerrado todavía un camino con vértice terminal en O entonces el algoritmo A* cierra un P_j con $\hat{f}(P_j) \leq h(s)$.

Demostración:

Supongamos por el absurdo que en una iteración el algoritmo cierra P_j con $\hat{f}(P_j) > h(s)$.
La proposición V.1.5 garantiza la existencia de un abierto mínimo P=(n,cmin(s,n),.). Se tiene que:

$\hat{f}(P) = g(n) + \hat{h}(n) = cmin(s,n) + \hat{h}(n) \leq$
$cmin(s,n) + h(n)$ (pues $\hat{h}$ es admisible)
$= h(s)$ (pues P es mínimo)
$< \hat{f}(P_j)$

lo que contradice la elección de P_j y completa la demostración.

¤

La proposición siguiente nos dice que el primer camino que encuentre el algoritmo con vértice terminal en O,

es un camino mínimo de s a O, si $\hat{h}$ es admisible. De esta forma el criterio de parada será simplemente: $n_j \in O$, en el algoritmo.

Proposición V.4.2

Sea $\hat{h}$ admisible en el algoritmo. Si P_j es el camino escogido en el paso (1.1) la primera vez que $n_j \in O$, entonces P_j es un camino de costo mínimo de s a O, $C(P_j)=h(s)$.

Demostración:

Por la proposición V.4.1:

$\hat{f}(P_j) = c_j + \hat{h}(n_j) \leq h(s)$

Como $\hat{h}(n_j) = 0$ entonces $c_j \leq h(s)$. Por lo que P_j es de costo mínimo de s a O

¤

Trataremos de ver como conseguir propiedades semejantes a las del algoritmo de Dijkstra. En particular, en que casos A* tiene un comportamiento polinomial en función del número de vértices.

Definición V.4.3

Decimos que una estimativa $\hat{h}$ es *consistente* si es admisible y $\forall n,m \in V(G)$: $\hat{h}(n) \leq h(n,m) + \hat{h}(m)$. Esto es, la estimativa tiene un comportamiento similar a la distancia entre vértices de un grafo.

Proposición V.4.3

Sea $\hat{h}$ una estimativa consistente. Entonces en cualquier iteración se tiene que $\hat{f}(P_j) \leq \hat{f}(P^i)$, i=1,...,q, donde los P^i son los elementos expandidos a partir del elemento P_j escogido en el paso (1.1).

Demostración:

Sea $P^i=(n^i,c^i,j)$ un elemento expandido a partir de P_j:

$$\begin{aligned}\hat{f}(P^i) &= c^i + \hat{h}(n^i)\\ &= c_j + c(n_j,n^i) + \hat{h}(n^i)\\ &\geq c_j + h(n_j,n^i) + \hat{h}(n^i) \text{ (por definición de } h(n_j,n^i))\\ &\geq c_j + \hat{h}(n_j) \text{ (pues } \hat{h}(n_j) \leq h(n_j,n^i) + \hat{h}(n^i))\\ &= \hat{f}(P_j)\end{aligned}$$

¤

De la proposición anterior se concluye que $\hat{f}(P_j)$ crece en cada iteración. El algoritmo de Dijkstra es un caso particular de A* tomando $\hat{h} = 0$, la cual es consistente cuando los costos son no negativos.

Proposición V.4.4

Sea $\hat{h}$ consistente. Todo elemento cerrado P=(n,c,p) satisface:

$c=g(n)=cmin(s,n)$

Es decir, P es de costo mínimo.

Demostración:

Suponga que $\bar{P} = (\bar{n}, \bar{c}, \bar{p})$ es cerrado en alguna iteración con $\bar{c} > g(\bar{n})$.

De acuerdo a la proposición V.1.5, al comienzo de esa iteración existe un abierto P=(n,c,p) en esa iteración tal que:

$c=g(n),\ c+h(n,\bar{n}) = g(\bar{n})$

Por lo tanto, $\hat{f}(P) = g(n) + \hat{h}(n)$

$$\begin{aligned}&\leq g(n) + h(n,\bar{n}) + \hat{h}(\bar{n}) \text{ (pues } \hat{h} \text{ es consistente)}\\ &= g(\bar{n}) + \hat{h}(\bar{n})\\ &< \bar{c} + \hat{h}(n)\end{aligned}$$

$$= \hat{f}(\bar{P}) \quad (\text{pues } \bar{c} > g(\bar{n}))$$

lo que contradice la elección de $\bar{P}$ en el paso (1.1).

¤

Las dos últimas proposiciones garantizan que el número de iteraciones del algoritmo A* es de orden lineal en función del número de vértices del grafo cuando la estimativa es consistente. Note que este resultado es independiente del signo de los costos de los arcos.

V.5 GRAFOS DE PRECEDENCIA.

Un *grafo de precedencia* es un grafo dirigido sin circuitos. Muchas son las aplicaciones donde encontramos grafos de precedencia. El concepto de orden entre los elementos de un conjunto aparece con frecuencia en Computación. Veamos algunos ejemplos:

(1) Podemos representar una expresión aritmética:

((a+b)*c + ((a+b)+e) * (e+f)) * ((a+b) * c)

mediante el digrafo sin circuitos de la figura V.5.

(2) El grafo reducido de un grafo dirigido (sección III.3.1).

(3) El grafo de una relación de orden parcial (eliminando los bucles).

(4) El grafo que representa un proceso de decisión secuencial como el de la figura V.6.

(5) Planificación de proyectos: un proyecto se puede dividir en actividades que deben realizarse en un cierto orden. Tal es el caso de un pensum de estudios de una carrera universitaria, donde ciertas materias son requisitos de otras. En este caso podemos representar las materias por vértices de un grafo donde los arcos son los requisitos inmediatos entre materias.

En este capítulo veremos algunas propiedades y algoritmos importantes en grafos de precedencia.

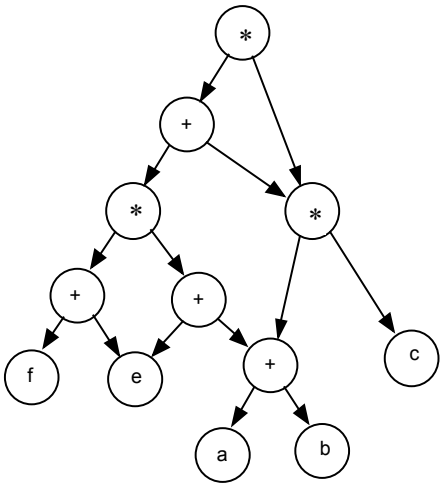


figura V.5

V.5.1 PROPIEDADES

Empezamos por dar dos propiedades sencillas pero importantes de grafos sin circuitos:

(a) Un grafo G no posee circuitos si y sólo sí todo subgrafo de G no posee circuitos.

(b) Principio de dualidad de grafos sin circuitos:

Un grafo G no posee circuitos si y sólo si el *grafo inverso* de G, G^{-1}, obtenido a partir de G invirtiendo la orientación de todos los arcos, no posee circuitos.

El estudio de grafos sin circuitos se fundamenta esencialmente en las propiedades siguientes:

Proposición V.5.1.1

Sea G un digrafo sin bucles. G no posee circuitos si y sólo si toda componente fuertemente conexa es un vértice aislado.

Demostración:

Al haber una componente fuertemente conexa con más de un vértice es evidente que habrá un circuito. Recíprocamente, si G posee un circuito, éste estará en una componente fuertemente conexa.

¤

Proposición V.5.1.2

Un digrafo G es de precedencia si y sólo si todo camino es elemental.

Demostración:

Que exista un camino no elemental es equivalente a decir que existe un camino cerrado. La proposición III.1.3.5 nos dice que existe un camino cerrado si y sólo si existe un circuito.

¤

Un vértice *fuente* de un digrafo es un vértice con grado interior igual a cero, es decir, el vértice es extremo inicial de todos los arcos incidentes en él. Un vértice *sumidero* es un vértice con grado exterior igual a cero, es decir, el vértice es extremo terminal de todo arco incidente en él.

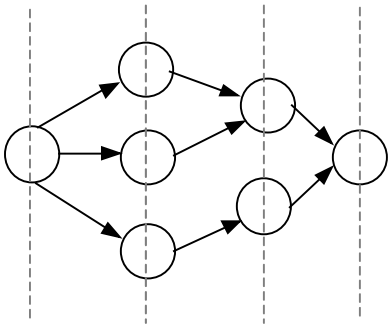


figura V.6

Proposición V.5.1.3

Todo grafo de precedencia G contiene un vértice fuente y un vértice sumidero.

Demostración:

Supongamos que G no contiene un vértice fuente. Entonces $\delta^-(G) \geq 1$ y la proposición III.1.3.11 nos dice que G posee un circuito.

Por otro lado, si G es de precedencia, entonces el grafo inverso de G, G^{-1}, es de precedencia y por lo tanto G^{-1} posee una fuente, la cual es sumidero de G.

¤

A continuación resaltaremos algunas características estructurales de los grafos de precedencia mediante dos números que asociaremos a cada vértice del grafo.

A todo vértice v de un digrafo cualquiera podemos asociar dos números enteros:

– $\eta(v)$ = nivel de v = largo máximo de un camino elemental terminando en v.
- h(v) = altura de v = largo máximo de un camino elemental comenzando en v.

En grafos de precedencia podemos eliminar el término "elemental" de estas definiciones. Note que el nivel de un vértice en G es la altura de ese vértice en G^{-1}, por lo que nivel y altura son conceptos duales.

Proposición V.5.1.4

En un grafo de precedencia, para todo vértice v del grafo, el vértice inicial de un camino de largo máximo terminando en v es una fuente.

Demostración:

Si v es una fuente, el resultado es evidente. Si v no es fuente, existe un camino de largo ≥ 1 comenzando en w ($\neq$v) y terminando en v. Si este camino es de largo máximo entonces cualquier predecesor de w debe estar en el camino. Como el grafo no posee circuitos, el vértice w no puede poseer predecesores, es decir, w es una fuente.

¤

Proposición V.5.1.5

En un grafo de precedencia, para todo vértice v del grafo, el vértice terminal de un camino de largo máximo comenzando en v es un sumidero.

Demostración:

Aplicar el principio de dualidad y la proposición V.5.1.4.

¤

La estructura simple que posee un grafo sin circuitos es puesta de manifiesto por el resultado siguiente:

Proposición V.5.1.6

Un digrafo G=(V,E) es de precedencia si y sólo si V se puede particionar en conjuntos V_0, V_1,..., V_p tal que $\forall$i, $0 \leq i \leq p$, se tiene: $v \in V_i$ si y sólo si i es el largo de un camino más largo terminando en v.

Demostración:

Mostremos la condición necesaria.
Si G no posee circuitos, consideremos la secuencia, bien definida según la proposición V.5.1.3 y la propiedad (a):

V_0 = Conjunto de vértices fuentes de G.
V_1 = Conjunto de vértices fuentes de G_{V-V_0}

....
V_i=Conjunto de los vértices fuentes de $G_{V-V_0-V_1...-V_{i-1}}$

La secuencia $V_0, V_1, \ldots, V_p$ (con V_p tal que G_{V_p} contiene sólo vértices aislados) es una partición de V. Basta mostrar ahora que si $v \in V_i$ entonces el camino más largo terminando en v es de largo i. Razonemos por inducción sobre i. Para i=0 es evidente. Supongamos que para $i \leq k$ se cumple que si $v \in V_i$ entonces el camino más largo terminado en v es de longitud i. Sea $v \in V_{k+1}$. Existe $w \in V_k$ tal que $(w,v) \in E$ y por hipótesis de inducción existe entonces un camino de largo k+1 hasta v. No puede haber un camino hasta v de largo mayor que k+1 pues el vértice w predecesor de v en ese camino está en $V_0 \cup ... \cup V_k$, lo cual implica que un camino más largo hasta w es de largo $\leq$ k, contradiciendo la existencia de tal camino.

Es importante notar que V_i, $0 \leq i \leq p$, es un conjunto estable de G.

Condición suficiente:

De acuerdo a la hipótesis, todo camino en G es de largo finito. Esto significa que todo camino de largo > 0 es elemental. La proposición V.5.1.2 nos dice que G debe ser de precedencia.

¤

De acuerdo a la proposición anterior si G es de precedencia entonces la partición V_o,...,V_p, se llama *partición*

en niveles pues se tiene que $v \in V_i$ si y sólo si $\eta(v)=i$ (ver figura V.7).

V.5.1.1 RELACIONES DE ORDEN Y GRAFOS DE PRECEDENCIA

Un digrafo de precedencia sin arcos múltiples G=(V,E) induce un orden parcial estricto sobre los elementos de V. Basta tomar la clausura transitiva E^+ de la relación binaria E. Decimos que E es una *relación de precedencia inmediata* si $\forall (v,w) \in E$ no existe un camino en G de largo mayor que 1 de v a w; también decimos que G es de precedencia inmediata.

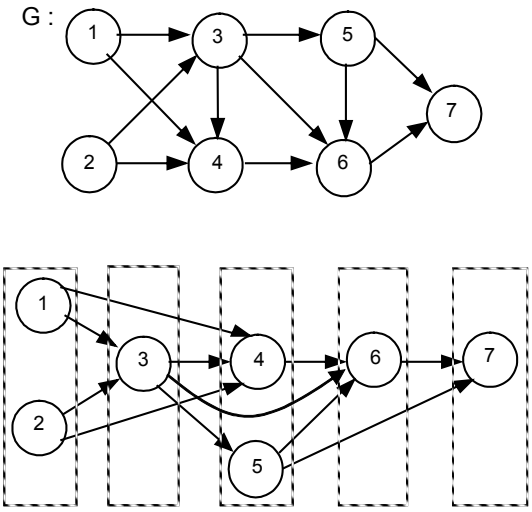
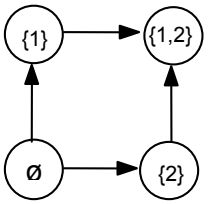


figura V.7

En una relación de orden R al eliminar los bucles y todas las *transitividades* , es decir, eliminar los pares (a,b) tales que existe c ($\neq$a,b) con $(a,c) \in R$ y $(c,b) \in R$, la relación resultante es de precedencia inmediata (el grafo asociado es de precedencia inmediata). Es fácil ver que al eliminar cualquier subconjunto de transitividades de R, la relación resultante posee la misma clausura transitiva de R. Debido a esta propiedad, podemos decir que la relación de precedencia inmediata extraída de R conserva la topología de R y muchas veces es más fácil trabajar con ella pues con menos información (pares) se sabe todo sobre R. Es lo que sucede cuando representamos el orden $(P(A),\subseteq)$, donde P(A) es el conjunto de las partes de un conjunto A, mediante un $|A|$-cubo (ver figura V.8).



2 - cubo asociado a P(A)

A = { 1, 2 }

figura V.8

Mostraremos ahora que cualquiera sea E *una relación de precedencia* sobre un conjunto V, es decir, G=(V,E) es un grafo de precedencia, el grafo G'=(V,E') donde E' es la relación de precedencia inmediata obtenida a partir de E eliminando las transitividades, es un grafo parcial de todo grafo parcial H=(V,F) de G tal que $E^+=F^+$. Esto último indica que cualquier grafo parcial de G cuya clausura transitiva coincide con la de G es obtenido a partir de G' agregándole arcos de G. A G' también se le llama *esqueleto de un grafo de precedencia o diagrama de Hasse*.

Proposición V.5.1.7

Sea G=(V,E) un grafo de precedencia sin arcos múltiples y $T=\{F / F \subseteq E \text{ y } F^+=E^+\}$. Entonces $(T,\subseteq)$ es un

orden parcial con menor elemento E' y mayor elemento E donde E' es la relación de precedencia inmediata de E.

Demostración:

Es evidente que E es el mayor elemento.

Mostremos que $\forall F \in T$ y $(v,w) \in E$ tal que no existe un camino de largo ≥ 2 de v a w, se cumple que $(v,w) \in F$:

Si $(v,w) \notin F$ entonces no existe camino de v a w en F esto último significa que $(v,w) \notin F^+$, pero $F^+=E^+$, lo cual es una contradicción. Así $E' \subseteq F$ y E' es el menor elemento de $(T,\subseteq)$.

¤

Proposición V.5.1.8

Sea $V_0,V_1,...,V_p$ la partición en niveles de V, con G=(V,E) un grafo de precedencia sin arcos múltiples. Si $\exists i$, $1 \leq i \leq p$, tal que $v \in V_{i-1}$, $w \in V_i$ y $(v,w) \in E$ entonces el vértice v precede inmediatamente a w, es decir, no existe en G un camino de longitud mayor que 1 de v a w.

Demostración:

Si $v \in V_{i-1}$ y $w \in V_i$ entonces $\eta(v)+1=\eta(w)$. Si existiera x descendiente de v y ascendiente de w entonces $\eta(v)<\eta(x)<\eta(w)$ lo que implicaría $\eta(w)-\eta(v) \geq 2$, lo cual es contradictorio. Así, v precede inmediatamente a w.

¤

Note que un vértice puede preceder inmediatamente a otro y no estar en niveles consecutivos.

La demostración de la proposición V.5.1.6 nos proporciona un método sencillo para calcular la partición en niveles, aún más, el método permite verificar si el grafo no posee circuitos. En cada iteración se detectan los vértices fuentes del grafo y se eliminan de éste.

Algoritmo de partición en niveles y reconocimiento de grafos de precedencia (versión 1):

{ Entrada: Un digrafo $G=(V,E)$ sin arcos múltiples.
Salida: Una variable booleana *circuito* que será verdadera si el grafo posee circuitos y falsa en caso contrario. Cuando *circuito* sea falsa, el algoritmo proporciona una partición $V_0,V_1,...,V_p$ de V en niveles.}

```
Variables i : entero;
Comienzo
    i ←0; circuito ← falso;
    Mientras G posea vértices y circuito =falso hacer:
    Comienzo
        V_i ← vértices fuentes de G ;
        Si V_i = ∅ entonces circuito ← verdad
            si no
            Comienzo
                Eliminar de G los vértices en V_i ;
                I ← i + 1
            fin
    fin
fin.
```

Una implementación de este algoritmo podría ser la siguiente: supongamos que G viene representado por la lista de adyacencias LA. LA es un arreglo de 1 a $|V|$ donde LA[i] es la lista de los sucesores del vértice i. En lugar de ir eliminando los vértices de G, tendremos asociado a cada vértice v el grado interior $d^-(v)$. Estos valores pueden ser calculados a partir de la lista de adyacencias mediante un algoritmo $O(|V| + |E|)$ y pueden ser almacenados en un arreglo GI de 1 a $|V|$. Sea N un arreglo tal que N[i] contendrá los vértices del nivel i. Cuando un vértice es colocado en N[i], indicaremos su eliminación de G, disminuyendo en una unidad el grado interior de cada sucesor del vértice.

Algoritmo de partición en niveles y reconocimiento de grafos de precedencia (versión 2):

{ Entrada: La lista de adyacencias *LA* de un digrafo *G=(V,E)* sin arcos múltiples. Un arreglo *GI* con el grado interior de cada vértice.

Salida: Una variable booleana *circuito* que será verdadera si existe un circuito en el grafo. Si *circuito* es falsa en el arreglo *N* el elemento *N[i]* contendrá los vértices del nivel i.}

```
Variables i ,nvert,x,y,nivel : entero;
Comienzo
    Para i ← 0 a |V|-1 hacer: N[i] ← ∅;
    nvert ← 0; nivel ← 0; circuito ← falso;
    {Buscar las fuentes de G}
    Para I ← 1 a |V| hacer:
        Si GI[i]=0 entonces
        Comienzo
            Insertar i en el conjunto N[nivel] ;
            nvert ← nvert + 1
        fin
    Mientras nvert < |V| y N[nivel] ≠ ∅ hacer:
    Comienzo
        Para cada x en N[nivel] hacer:
            Para cada y en LA[x] hacer:
            Comienzo
                GI[y] ← GI[y] - 1;
                Si GI[y] = 0 entonces
                Comienzo
                    Insertar y en N[nivel + 1] ;
                    nvert ← nvert + 1
                fin
            fin;
        nivel ← nivel + 1
    fin
    circuito ← (nvert < |V|)
fin.
```

La complejidad en tiempo del algoritmo anterior es la siguiente:

- El cálculo del arreglo GI es $O(|V| + |E|)$.
- La inicialización de N y la determinación de las fuentes de G es $O(|V|)$.
- En el cuerpo del Mientras cada arco es examinado una sola vez, por lo tanto el número de operaciones del Mientras es $O(\max(|V|,|E|))$.

Por lo tanto el algoritmo es $O(\max(|V|,|E|))$.

V.5.2 ORDENAMIENTO TOPOLÓGICO

Un *ordenamiento topológico* de un digrafo G=(V,E) es cualquier función inyectiva f: $V \to \aleph$ tal que:

$$\forall v,w \in V: \text{Si } (v,w) \in E \text{ entonces } f(v) < f(w).$$

Proposición V.5.2.1

Un grafo G es de precedencia si y sólo si G admite un ordenamiento topológico.

Demostración:

Condición suficiente:
Si G admitiera un circuito $C=<x_1,x_2,...,x_p,x_1>$, tendríamos:

$$f(x_1) < f(x_2) <...< f(x_p) < f(x_1).$$

lo cual es una contradicción.

Condición necesaria:
Sea $V_0,V_1,...,V_p$ la partición en niveles de G. Numeremos los vértices de V_0 con los enteros en el intervalo

$[1,|V_0|]$. Este es un ordenamiento topológico de los vértices de V_0 por ser V_0 un estable. Para numerar los elementos de V_i, utilizaremos los enteros en el intervalo $[\sum_{j<i}|V_j|+1,\ \sum_{j\le i}|V_j|]$.

La numeración corresponde a un orden topológico de G pues si $(v,w)\in E(G)$ entonces $\eta(v) < \eta(w)$, por lo tanto v fue numerado con un número menor que el de w.

¤

La proposición anterior proporciona un método para hallar un ordenamiento topológico de un grafo. Incluso, utilizando el algoritmo $O(\max(|V|,|E|))$ para calcular la partición en niveles de G (sección V.5.1,versión 2), podemos determinar un orden topológico de los vértices de G. Basta con asignar el siguiente entero (comenzando en 1) cada vez que un vértice es insertado en el nivel correspondiente. El contador nvert puede ser utilizado para tal fin.

En el capítulo IV, sección IV.2.6.4, dimos un algoritmo basado en el de búsqueda en profundidad para determinar un ordenamiento topológico inverso del grafo reducido de un grafo. El algoritmo consiste en lo siguiente: aplicamos el algoritmo de visita de vértices a G y numeramos sus vértices por orden de terminación de la visita a todos sus descendientes. Llamemos NumComp(v) al número que se asigna a cada vértice v de G.

Proposición V.5.2.2 (similar a la Proposición IV.2.6.4.2)

Sea G un grafo de precedencia sin arcos múltiples y $(v,w)\in E(G)$. Aplicamos el algoritmo de visita de vértices por búsqueda en profundidad al grafo G, asignando a cada vértice v el NumComp(v). Entonces:

NumComp(v) > NumComp(w).

Demostración:

Hay dos casos a considerar:

(a) w es visitado antes que v.

(b) v es visitado antes que w.

(a) Cuando se visita w, ningún descendiente de w en B puede ser v, pues existiría un camino de w a v en G, pero como $(v,w) \in E(G)$, esto implicaría que G posee un circuito. Por lo tanto v no es descendiente de w. Al terminar de visitar los descendientes de w, todavía no se habrá visitado v, así NumComp(v) > NumComp(w).

(b) La proposición IV.2.2.3 nos dice que w debe ser descendiente de v. Por lo tanto, antes de visitar todos los descendientes de v se habrán visitado todos los de w (que también son descendientes de v), así NumComp(v) > NumComp(w).

¤

NumComp es un ordenamiento topológico inverso de G. Un ordenamiento topológico de G sería:

$f(v) = |V|$-NumComp(v)+1.

Algoritmo de ordenamiento topológico:
{ Entrada: Un grafo *G=(V,E)* sin circuitos y sin arcos múltiples.
Salida: Un ordenamiento topológico f de los vértices de *G*. }
Variables *Contador* : entero;
v : vértice;
Comienzo
Inicialmente ningún vértice de *G* ha sido visitado;
Contador $\leftarrow |V| + 1$;
Para cada vértice *v* de G hacer:
Si *v* no ha sido visitado entonces Búsqueda en profundidad(*v*);
Donde
Búsqueda en profundidad(*v* :vértice):
Variables *w* : vértice;
Comienzo
Marcar *v* como visitado;
Para cada sucesor *w* de *v* hacer:
Si *w* no ha sido visitado entonces Búsqueda en profundidad(*w*);

Contador ← *Contador* - 1;
f (*v*) ← *Contador*
fin
fin.

V.5.3 CAMINOS DE COSTO MÍNIMO Y COSTO MÁXIMO

El algoritmo de Bellman (Programación Dinámica Progresiva) presentado en la sección V.2, puede ser utilizado para calcular caminos de costo mínimo y máximo en un grafo sin circuitos. El algoritmo en efecto calcula los caminos de costo mínimo desde un vértice raíz del grafo (por lo tanto fuente) hasta cada vértice alcanzable desde la raíz. Los caminos se van cerrando siguiendo un orden topológico. La implementación que se dio es similar al algoritmo de cálculo de niveles. En ese algoritmo, el cálculo de los caminos mínimos desde un vértice s se hace considerando los vértices según un orden topológico. Al momento de calcular el camino mínimo hasta un vértice v ya tendremos calculados los caminos mínimos hasta cada predecesor de v y el criterio de Bellman nos dice:

$cmin(s,v) = \min \{cmin(s,w) + c(w,v) / w \text{ es predecesor de } v\}$.

Presentaremos a continuación una variante del algoritmo, conocida como *Programación Dinámica Regresiva.* En este caso se calcula el camino mínimo desde un vértice v_i a uno v_j partiendo del vértice destino v_j y aplicando el principio de dualidad de Bellman:

$cmin(v_i,v_j) = \min \{ c(v_i,w) + cmin(w,v_j) / w \text{ es sucesor de } v_i\}$

Avoquémonos a la presentación del algoritmo.

El algoritmo determinará un camino mínimo desde un vértice s a uno t de un grafo de precedencia G=(V,E) con función de costos c en los arcos. Supondremos que si $(v,w) \notin E$ entonces $c(v,w)=+\infty$.

Sea f un ordenamiento topológico de G:

$f(v_1) < f(v_2) < ... < f(v_n)$ con $V=\{v_1, v_2, ... , v_n\}$.

Para todos i,j con i<j se tiene que todos los posibles caminos de v_i a v_j contienen sólo vértices v_k con $i \leq k \leq j$. Por lo tanto si conocemos un camino de menor costo desde v_{k-1} a v_j, éste tiene que ser un camino de menor costo entre los caminos:

$<v_{k-1},v_l> \,||\, P_l$, donde v_l es un sucesor de v_{k-1} con $l \leq j$ y P_l es un camino de menor costo de v_l a v_j.

El algoritmo de visita de vértices por búsqueda en profundidad permite considerar los vértices por orden inverso a un orden topológico. Utilizaremos este algoritmo como esqueleto de nuestro algoritmo.

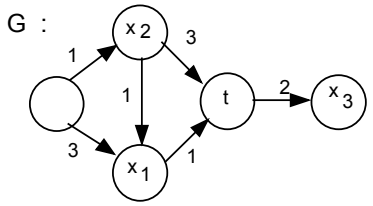
La diferencia del método con los algoritmos hasta ahora presentados es que estos últimos calculan caminos partiendo del vértice origen s, mientras que el algoritmo que presentaremos calcula un camino mínimo desde cada vértice del grafo, presente en un camino de s a t, hasta el vértice sumidero t. Por lo tanto debemos conocer el vértice t al cual se quiere llegar desde s. A cada vértice v del grafo se asocia el valor cmin(v) que representa el costo de un camino mínimo de v a t. En caso de no existir camino se tendrá $cmin(v) = +\infty$; también asociaremos a v un apuntador al vértice sucesor de v en un camino mínimo de v a t y lo denotaremos por siguiente(v). Los comentarios hechos en párrafos anteriores permiten aplicar criterios de dualidad para mostrar la equivalencia entre el algoritmo que presentaremos y el de Bellman dado en la sección V.2. En efecto, el algoritmo que presentamos puede ser ligeramente modificado para calcular los caminos mínimos desde cada vértice de G hasta un vértice t. Si queremos utilizar el algoritmo para calcular caminos mínimos desde un vértice t hasta cada vértice de G, bastaría con invertir la dirección de los arcos de G y aplicar el algoritmo a este nuevo grafo G^{-1}. Obtendremos así un camino mínimo desde cada vértice en G^{-1} hasta t. Ahora bien, C^{-1} es un camino mínimo de v a t en G^{-1} si y sólo si el camino C obtenido de C^{-1} invirtiendo el sentido a los arcos es un camino mínimo de t a v en G. Por lo tanto, conociendo los caminos mínimos desde cada vértice v de G^{-1} a t, los caminos mínimos de t a cada vértice v de G se obtienen de los primeros invirtiendo el sentido a los arcos.

Algoritmo de Programación Dinámica Regresiva para el cálculo de caminos mínimos:

{ Entrada: Un grafo de precedencia *G=(V,E)* sin arcos múltiples. Dos vértices *s* y *t* de *G*. Una función de costos $c : E \rightarrow \Re$.

Salida: Un camino de costo mínimo de *s* a *t*. Si cmin(*s*)=+∞ no hay camino de *s* a *t*. Si no, el camino se podrá construir siguiendo los apuntadores siguiente(s): <*s*,siguiente(*s*), siguiente(siguiente(*s*)),...,*t* >.}

```
Comienzo
    Inicialmente ningún vértice ha sido visitado;
    Inicialmente cmin(v)=+∞, ∀v∈V(G) ;
    Búsqueda en Profundidad(s )
Donde
    Búsqueda en Profundidad(x :vértice):
    Variables y :vértice;
    Comienzo
        Marcar x como visitado;
        { Si ya hemos visitado a t , no hace falta visitar sus
        descendientes }
        Si x =t entonces cmin(x )←0
            si no
            Comienzo
                Para cada y sucesor de x hacer:
                    Si y no ha sido visitado entonces
                        Búsqueda en Profundidad(y );
                { Se visitaron todos los descendientes de x. Así x es el próximo en el orden topológico inverso}
                Para cada y sucesor de x hacer:
                    Si (cmin(y)≠+∞) y (cmin(x) >cmin(y) +c(x,y ))
                        entonces
                        Comienzo
                            cmin(x ) ← cmin(y ) + c(x,y );
                            siguiente(x ) ← y
                        fin
            fin
    fin
fin.
```



Árbol de la búsqueda en profundidad

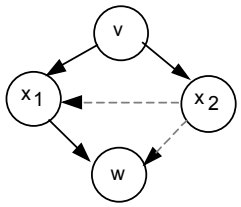


figura V.9

Es importante notar que este algoritmo funciona cualquiera sean s y t. Si al finalizar el algoritmo cmin(s) = +∞, entonces no existirá camino de s a t en G. El símbolo +∞ significa un número muy grande, basta con tomarlo mayor que el costo de cualquier camino en G. Note también que el algoritmo funciona cualquiera sea la función de costos c. Por lo tanto permite calcular el camino de costo máximo de s a t (basta con cambiar de signo a la función de costos c). Finalmente, es evidente que el algoritmo es O(max(|V|,|E|)).

En la figura V.9 vemos el árbol generado por el algoritmo de búsqueda en profundidad aplicado al grafo G de la figura. El primer vértice al cual visitan todos sus descendientes es x_1. Se calcula siguiente(x_1) = t y cmin(x_1) = 1. El siguiente vértice es x_2. El camino mínimo de x_2 a t se calcula entre los dos sucesores de x_2 que son x_1 y t. Como $c(x_2,x_1)+cmin(x_1) < c(x_2,t)+cmin(t)$ entonces siguiente(x_2)=x_1 y cmin(x_2)=2. Finalmente se alcanza a s y como $c(s,x_1)+cmin(x_1) > c(s,x_2)+cmin(x_2)$, se tiene que siguiente(s)=x_2 y cmin(s)=3. Por lo tanto el camino de costo mínimo de s a t es $<s,x_2,x_1,t>$.

V.5.4 PLANIFICACIÓN DE PROYECTOS.

Los grafos de precedencia aparecen en muchas aplicaciones, particularmente en aquéllas donde se pretende determinar caminos de costo máximo entre dos vértices, como es el caso de los grafos que aparecen en Planificación de Proyectos.

Suponga que se quiere realizar un proyecto conformado por un gran número de actividades. Se puede representar cada actividad por un vértice de un grafo G y establecer un arco desde un vértice v_i hasta un vértice v_j para indicar que la actividad i precede a la actividad j. A cada arco asociamos un costo c_{ij} que representa la mínima cantidad de tiempo necesario entre el comienzo de la actividad i y el comienzo de la actividad j; en efecto, es la duración mínima de la actividad i. El grafo resultante es obviamente dirigido sin circuitos, ya que una actividad no puede precederse a si misma.

El problema consiste en determinar el tiempo mínimo necesario para completar el proyecto. Este problema se reduce en términos del grafo a encontrar un camino de costo máximo entre un vértice s que representa el comienzo del proyecto, y un vértice t que representa su terminación. Esto se debe a que el tiempo mínimo para que comience una actividad es el máximo entre los tiempos mínimos para que concluyan las actividades predecesoras.

Un camino de costo máximo entre s y t se llama *camino crítico* debido a que las actividades que en él se encuentran determinan el tiempo global de terminación del proyecto pues cualquier retraso en el comienzo de una actividad en el camino crítico implica un retraso en el tiempo de culminación del proyecto.

Otro de los objetivos de la planificación de proyectos es determinar cuáles son las *actividades críticas* , es decir, aquéllas que no se puede retrasar sin que se retrase la duración de todo el proyecto y para las actividades no críticas determinar la holgura en tiempo que esta cada una posee para ser comenzada sin alterar el tiempo global de duración del proyecto.

Podemos utilizar cualquier algoritmo de los presentados en secciones anteriores para determinar un camino crítico de s a t. Sea cmax(v) el costo del camino máximo de s a v, con $v \in V(G)$. El valor cmax(t) es el costo de un camino crítico. Para conseguir el camino crítico basta con seguir los apuntadores que proporciona el algoritmo de caminos óptimos que utilicemos.

Sea TEC(i) el tiempo más temprano en que puede comenzar la actividad i. Se tiene que TEC(i)=cmax(v_i), donde v_i es el vértice correspondiente a la actividad i en el grafo. Sea TAC(i) el tiempo más tarde en que puede comenzar la actividad i sin afectar la duración cmax(t) del proyecto. Supongamos que las actividades están ordenadas de acuerdo a un orden topológico, es decir, si $(v_i,v_j) \in E(G)$ entonces i<j, con el vértice s numerado 1 y el vértice t numerado n (=$|V(G)|$). El cálculo de TAC(i) se puede efectuar recursivamente de acuerdo a las reglas siguientes:

- TAC(n) = cmax(t).

- TAC(i) = min $\{TAC(j)-c_{ij} / (v_i,v_j) \in E(G)\}$.

La cantidad TAC(i)-TEC(i) representa la holgura en tiempo que posee la actividad i para ser comenzada sin afectar la duración global del proyecto. En otras palabras, si para cada actividad i del proyecto escogemos el tiempo de comienzo de la actividad en el intervalo [TEC(i),TAC(i)] entonces el proyecto no sufriría ningún retraso, su duración seguiría siendo cmax(t). Toda actividad i que esté en un camino crítico cumple con TAC(i)-TEC(i)=0 pues la actividad no puede ser retrasada sin afectar la duración del proyecto.

Es de hacer notar que para reducir los cálculos en la determinación de TAC, como los costos son positivos, podríamos eliminar las transitividades de G y trabajar sobre el grafo de precedencia inmediata.

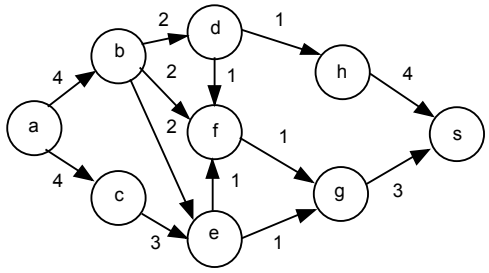


figura V.10

Daremos un ejemplo de cómo sería el cálculo. Supongamos que se tienen las actividades siguientes:

Actividad	Duración	Precede a
a	4	b,c
b	2	d,e,f
c	3	e
d	1	f,h
e	1	f,g
f	1	g
g	3	
h	4	

El grafo de la figura V.10 representa la situación. Note que se ha agregado un vértice s donde convergen los vértices g y h que no poseen actividades sucesoras. No es necesario agregar un vértice fuente pues ya hay una raíz.

Para hallar TEC, aplicamos el algoritmo de Bellman (sección V.2) partiendo de la actividad a, teniendo el cuidado de cambiar el signo a los costos antes de aplicar el algoritmo. Obtendremos el costo de un camino máximo de **a** a cada vértice del grafo. Note que no se puede aplicar el algoritmo presentado en la sección anterior, a menos que invirtamos el sentido a los arcos y tomamos como sumidero al vértice a. Obtenemos:

Actividad	**Costo Máximo**	**Predecesor**
a	0	no tiene
b	4	a
c	4	a
d	6	b
e	7	c
f	8	e
g	9	f
h	7	d
s	12	g

Para hallar TAC, aplicamos el algoritmo de Programación Dinámica Regresiva de la sección V.5.3, modificándolo ligeramente. Comenzamos colocando TAC(s)=TEC(s) y aplicamos recursivamente la fórmula:

$$TAC(i) = \min\{TAC(j)-c_{ij} / (v_i,v_j)\in E(G)\}$$

Seguimos un orden topológico inverso en el cálculo de TAC(i). Este orden resulta inmediato del algoritmo pues el siguiente vértice en el orden inverso es el siguiente vértice al cual se le han visitado todos sus descendientes en el algoritmo. Supongamos que la consideración de los sucesores de cada vértice en el algoritmo de Programación Dinámica Regresiva se hace en orden alfabético de la etiqueta del vértice. El algoritmo genera el árbol de búsqueda

en profundidad presentado en la figura V.11.

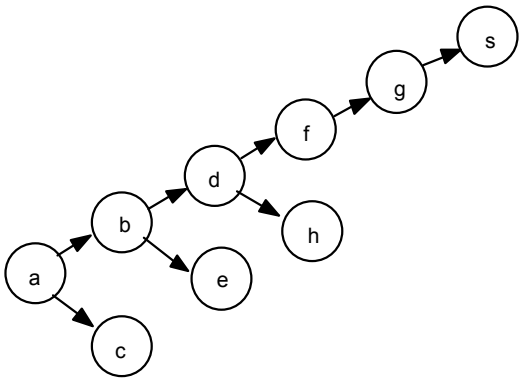


figura V.11

Por lo tanto:

Actividad: s g f h d e b c a

TAC: 12 9 8 8 7 7 5 4 0

Por lo tanto ya tenemos toda la información necesaria sobre el proyecto. Un camino crítico es <a,c,e,f,g,s>. La tabla y diagrama de tiempos siguientes resumen toda la información necesaria:

El diagrama de tiempo indica para cada actividad, lo más temprano que puede comenzar y lo más tarde que puede terminar (ver figura V.12)

Actividad	TEC	TAC	¿Es Crítica?	Holgura (TAC-TEC)	Duración
a	0	0	sí	0	4
b	4	5	no	1	2
c	4	4	sí	0	3
d	6	7	no	1	1
e	7	7	sí	0	1
f	8	8	sí	0	1
g	9	9	sí	0	3
h	7	8	no	1	4

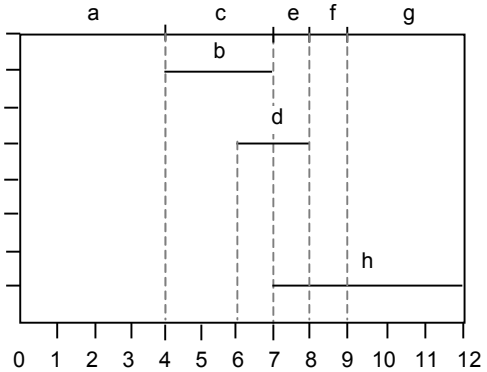


figura V.12

A continuación presentamos el algoritmo para el cálculo de TAC. Es una variante del algoritmo de Programación Dinámica Regresiva.

Cálculo de TAC:

{ Entrada: Un grafo de precedencia $G=(V,E)$ y una función de costos $c:E \rightarrow \Re$. Dos vértices s y t, fuente y sumidero del grafo G, y TEC(t).

Salida: Para cada vértice $v \in V(G)$, TAC(v).}

Comienzo

Inicialmente ningún vértice ha sido visitado;

Inicialmente TAC(v)=$+\infty$, $\forall v \in V(G)$;

Búsqueda en Profundidad(s);

Donde

Búsqueda en Profundidad(x : vértice):

Variables y : vértice;

Comienzo

Marcar x como visitado;

Si $x = t$ entonces TAC(x) $\leftarrow$ TEC(x)

si no

Comienzo

Para cada sucesor y de x hacer:

Si y no ha sido visitado entonces Búsqueda en Profundidad(y);

Para cada sucesor y de x hacer:

Si (TAC(y)$\neq+\infty$) y (TAC(x) > TAC(y) - c(x,y)) entonces TAC(x) $\leftarrow$ TAC(y) - c(x,y);

fin

fin

fin.

Proposición V.5.4.1

Cualquiera sea v vértice de G, se tiene que TAC (v) = cmax(s,t) – cmax(v,t)

Demostración: Inducción sobre la altura de un vértice. Utilizar la hipótesis inductiva: "todos los sucesores de un vértice dado v satisfacen la igualdad" y demuestre para v. La base de la inducción es comprobar la igualdad para el vértice t.

Proposición V.5.4.2

Sabemos que una actividad crítica v es una actividad con holgura cero (TAC(v)-TEC(v) = 0). Una actividad v es una actividad crítica si y sólo si v pertenece a un camino crítico de s a t.

Demostración:

($\Rightarrow$) Si TAC(v)- TEC(v)=0 entonces TAC(v) = TEC(v) = cmax(s,v). Como TAC(v) = cmax(s,t) – cmax(v,t) entonces cmax(s,t) = cmax(s,v) + cmax(v,t). Es decir, un camino máximo de s a v concatenado con un camino máximo de v a t es un camino de costo máximo, por lo que v está en un camino de costo máximo.

($\Leftarrow$) Sea P un camino de costo máximo de s a t que pasa por v. Entonces cmax(s,v)+cmax(v,t) = cmax(s,t) por principio de optimalidad en grafos de precedencia. Por la proposición V.5.4.1 tenemos que TAC(v) = TEC(v).

V.6 EJERCICIOS

1. Use el algoritmo de Dijkstra (tanto la versión del modelo general de algoritmos de costo mínimo, como la versión de Dijkstra propiamente dicha) para encontrar la distancia y el camino más corto del vértice 6 a cualquier vértice del siguiente grafo:

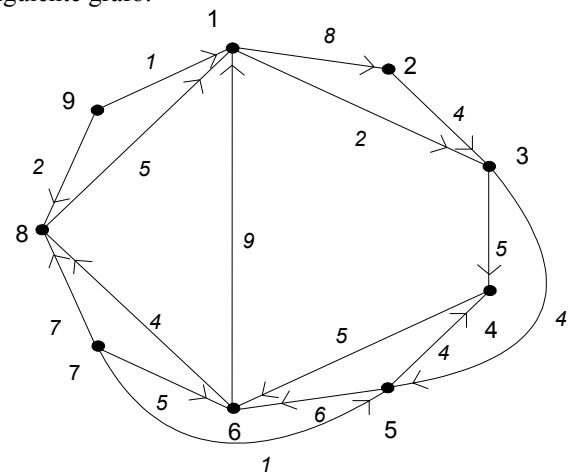


figura V.13

2. Dar tres razones por las cuales la siguiente lista de caminos no puede corresponder a un paso intermedio de un algoritmo de búsqueda de camino de costo mínimo, siguiendo el modelo general, con costos no negativos. *s*, *a*, *b* y *c* son vértices del grafo:

1.-	(s, 0, -)	CERRADO
2.-	(a, 3, 1)	ABIERTO
3.-	(b, 2, 1)	CERRADO
4.-	(b, 5, 2)	ABIERTO
5.-	(c, 1, 3)	ABIERTO

3. Modifique el algoritmo de DFS recursivo para hallar la altura de cada vértice en un grafo dirigido sin circuitos.

4. Muestre que el algoritmo de Dijkstra no funciona si los costos de los arcos son algunos negativos.

5. Seleccione las estructuras de datos apropiadas para lograr que el tiempo de algoritmo de Dijkstra sea $O(e.log(n))$ para un grafo con e aristas y n vértices. ¿En que caso sería conveniente?

6. Analizar cómo varía la eficiencia en BFS y Dijkstra si la lista de caminos abiertos se guarda en un heap en lugar de una lista lineal.

7. Modifique el algoritmo de Dijkstra para que obtenga todos los caminos mínimos de un vértice v dado a cualquier otro vértice w, en un grafo dirigido con costos no negativos en los arcos.

8. Se desea determinar el camino de costo máximo entre un par de vértices de un grafo de precedencia G con costos positivos. Se sugieren dos modificaciones al algoritmo de Dijkstra:

- Se multiplican los costos de G por –1 y se construye con ello un nuevo grafo G'. Se aplica Dijkstra sobre G'. El camino mínimo de v a w en G' constituye el camino máximo en G.

- Sea k el mayor costo de un arco de G. Se calcula la siguiente función de costo: $c'(e)=k-c(e)$, donde $c(e)$ es el costo del lado e en G, para crear el grafo G'. El camino mínimo en G' es el camino máximo en G.

9. Analice si estos algoritmos logran el objetivo deseado.

10. Modifique Dijkstra para detectar circuitos de costo negativo en un digrafo.

11. Muestre si el algoritmo que se da a continuación obtiene el costo de un camino mínimo de v_0 a cualquier vértice v en un grafo arbitrario con aristas que pueden tener costo negativo:

- Comienzo
- $S \leftarrow \{v_0\}$;
- $D[v_0] \leftarrow 0$;
- Para cada v en $V-\{v_0\}$ hacer $D[v] \leftarrow c(v_0, v)$;
- Mientras $S \neq V$ hacer
- Comienzo
- Escoger un vértice w en V-S tal que D[w] sea mínimo;
- $S \leftarrow S \cup \{w\}$;
- Para todo $v \in V$ tal que $D[v] > D[w] + c(w, v)$ hacer
- Comienzo
- $D[v] \leftarrow D[w] + c(w, v)$;
- $S \leftarrow S - \{v\}$;
- Fin
- Fin
- Fin

12. Samuel J, Pettacci A. es miembro fundador y activo de los bomberos de la USB, y es quien maneja el camión de bomberos. Siendo Samuel estudiante de computación, y por lo tanto versado en métodos eficientes para evitar esfuerzos innecesarios, no le hace gracia dejar el agradable ambiente de la casa de su novia en la Urbina por más tiempo del estrictamente necesario. Usualmente, la cantidad de tiempo que les toma llegar a la sede al resto de los bomberos es de 15 minutos.

La pregunta que Samuel debe responder es "¿Cuántos minutos después que suena el buscapersonas puede despedirse de su novia, si quiere llegar no más tarde que el último miembro del grupo?". Samuel construyó de memoria la red de la siguiente figura, fijando para cada tramo un estimado del tiempo necesario para atravesarlo. La casa de la novia de Samuel es el vértice 1 y la sede de los bomberos es el vértice 14. Diga como resolver el problema de Samuel y resuélvalo.

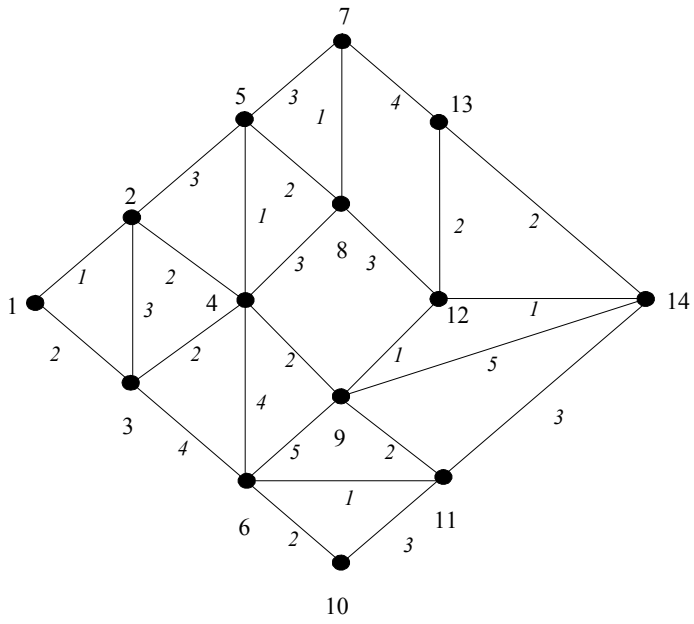


figura V.14

13. Dadas dos coordenadas (x_1,y_1) (x_2, y_2) en un tablero de ajedrez, hallar la ruta mínima que debería recorrer un caballo de ajedrez para ir de (x_1,y_1) a (x_2,y_2). Utilice el algoritmo de Dijkstra y el algoritmo BFS para resolver el problema y compárelos.

14. Escriba un algoritmo para hallar el camino mínimo entre dos vértices *v*, *w* de un grafo *G*, tomando en cuenta una de las siguientes restricciones:

a) El camino de *v* a *w* no debe contener vértices de *V'*, donde *V'* es un subconjunto propio de *V*.

b) El camino de *v* a *w* debe contener todos los vértices de un subconjunto propio *V'* de *V*.

c) El camino no debe contener aristas de un conjunto *E'* que es un subconjunto propio de *E*.

d) El camino debe contener todas las aristas de *E'*, subconjunto propio de *E*.

15. El grafo de la siguiente figura muestra los canales de comunicación (lados) y los tiempos de comunicación en minutos (asociados a los lados) entre 8 centros de comunicaciones (vértices). A las 3:00 pm desde el vértice *a* se envía un mensaje a cada uno de sus vecinos. El resto de los centros de comunicación retransmiten a cada uno de sus vecinos tan pronto lo reciben por primera vez. Se quiere saber a qué hora recibe el mensaje cada vértice de la red. Diga que algoritmo utilizaría y muestre la corrida en frío del mismo.

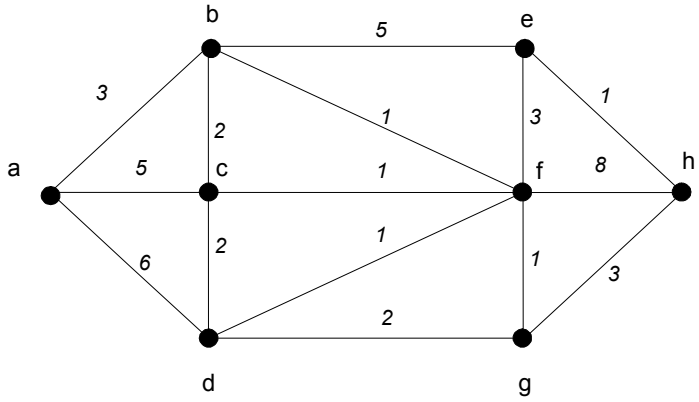


figura V.15

16. Una compañía manufacturera está planificando producir aires acondicionados de 3 componentes: gabinete, ventilador y motor. El precio de fabricación es de 50 dólares para los gabinetes, 75 dólares para los ventiladores y 100 dólares los motores. Sin embargo, una vez que los ventiladores están en producción, el costo de producción de gabinetes y motores se reduce en un 5%. Si los motores se producen primero, el costo de las otras unidades se reduce en un 10%. Además, después de estar 2 unidades en producción, se produce una reducción extra del 5% para la tercera unidad. Defina un grafo que represente el problema y resuélvalo como un problema de camino mínimo.

17. Sea *G* un digrafo sin arcos múltiples. ¿Cómo podrían ser utilizados los algoritmos de Floyd y Dijkstra para calcular la clausura transitiva del grafo *G*?

18. Para el siguiente grafo:

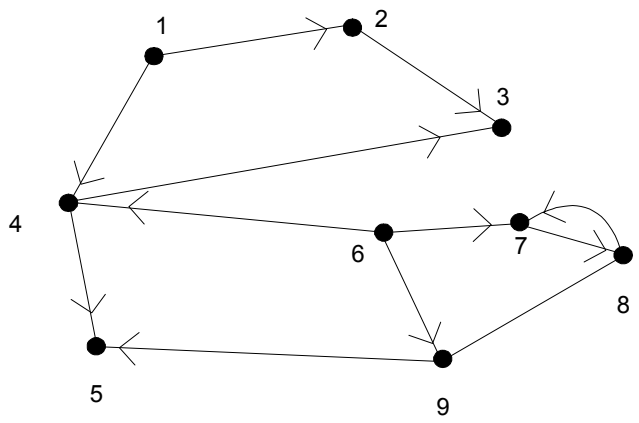


figura V.16

a) Encuentre la longitud del camino más corto entre cada par de vértices.

b) Determinar la altura y el nivel de cada vértice si no existiera el arco (7,8).

19. Mostrar que el siguiente algoritmo, debido a Dantzig, encuentra todas las distancias mínimas en un digrafo sin circuitos negativos, al igual que el algoritmo de Floyd. Sea $C^k(i, j)$ la distancia mínima de i a j, con $1 \le i,j \le k$ sin utilizar ningún vértice mayor que k. Sea $c(i, j) = c(e)$ si $e=\{i, j\} \in E$ y $c(i, j) = \infty$ si $\{i, j\} \notin E$.

- $C^1(1,1) \leftarrow 0$
- $k \leftarrow 1$
- Repetir
- $k \leftarrow k + 1$
- Para $1 \le i \le k$ hacer
- Comienzo
- $C^k(i, k) \leftarrow \text{Min}_{1 \le j \le k-1}\{C^{k-1}(i, j)+c(j, k)\}$
- $C^k(k, i) \leftarrow \text{Min}_{1 \le j \le k-1}\{c(k, j)+C^{k-1}(j, i)\}$
- fin
- Para $1 \le i, j < k$ hacer
- $C^k(i, j) \leftarrow \text{Min}\{C^{k-1}(i, j), C^k(i, k) + C^k(k, j)\}$
- Hasta k=n

20. Dé un ejemplo de una estimativa no consistente para la cual el número de iteraciones del algoritmo A* sea mayor que el número de vértices alcanzables desde s.

21. Determinar la altura y el nivel de cada vértice del grafo siguiente:

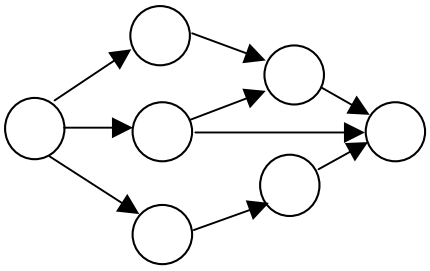


figura V.17

22. Sea *G=(V,E)* un digrafo sin arcos múltiples y *c* una función que asocia un número real a cada arco de *G*. Supongamos que se quiere hallar un camino de costo máximo desde un vértice *s* hasta cada vértice alcanzable desde *s*.

a) ¿Qué condiciones habría que imponer a *G* y *c* para que pueda existir caminos de costo máximo desde *s* a cada vértice alcanzable desde *s*?

b) Suponiendo que se satisfacen las condiciones dadas en a), ¿Para hallar un camino de costo máximo desde *s* hasta cada vértice alcanzable desde *s* bastaría con invertir el signo de los costos de los arcos y aplicar a *G* con estos nuevos costos un algoritmo de búsqueda de caminos mínimos basado en el modelo general de etiquetamiento?

23. Aplicar programación dinámica progresiva y regresiva para hallar un camino de costo máximo de **a** a **g** en el grafo siguiente:

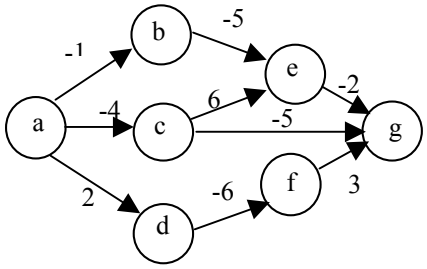


figura V.18

24. Se tiene un proyecto de construcción conformado por varias actividades: A, B, C, D, E, F y G, con duraciones respectivas 1, 2, 5, 3, 4, 3, 2. Algunas actividades preceden a otras en su ejecución, como se indica a continuación:

Actividad	**Precede a**	**Actividad**
A		B, C
B		D, E, F
C		E
D		F

Actividad	**Precede a**	**Actividad**
E		F, G
F		G
G		---

Determine los tiempos más tempranos y tardíos en que debe comenzar una actividad para que el proyecto se ejecute en el menor tiempo posible.

25. La compañía EMPR S.A. tiene planeado para su inmediata ejecución un proyecto que involucra 7 actividades, distinguidas con las letras de la A hasta la G. Existen relaciones de precedencia entre muchas de estas actividades, la cuales se indican mediante el siguiente grafo:

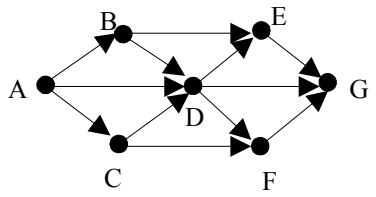


figura V.19

La duración de cada actividad es como sigue: A:2, B:3, C:4, D:1, E:8, F:4, G: 2

a) Encuentre un ordenamiento topológico del grafo.

b) Encuentre el tiempo mínimo requerido para culminar todo el proyecto, así como las actividades críticas.