Teoría de Grafos

Un ***grafo no dirigido*** G=(V,E) se compone de dos conjuntos finitos[1]:

- el conjunto V={v_1, v_2,...}, que es el conjunto de ***nodos*** de G y
- el conjunto de aristas E={e_1, e_2,...} que es un conjunto de pares no ordenados de nodos diferentes de G.

Cada elemento del conjunto de aristas e=(u,v) se conoce como una ***arista*** y se dice que une los nodos u y v.

Si e=(u,v) es una arista de G, entonces los nodos u y v son ***adyacentes.*** Dado que estamos tratando con pares no ordenados, (u,v) y (v,u) representan la misma arista.

El ***grado*** de un nodo en un grafo no dirigido está dado por el número de aristas en las que participa dicho nodo.

Un ***camino P de longitud n***, desde un nodo *u* a un nodo *v*, se define como la secuencia de n+1 nodos : P=(v_0, v_1, ...,v_n) donde u=v_0, v=v_n, v_i es adyacente a v_{i-1}, 1<=i<=n.

Un ***camino P es cerrado*** si v_0=v_n.

Un ***camino P es simple*** si todos los nodos son distintos, a excepción de v_0 que puede ser igual a v_n.

Un ***ciclo*** es un camino simple cerrado de longitud 3 o mas. Un ciclo de longitud k se llama k-ciclo.

Un ***grafo acíclico***, también llamado ***bosque***, es aquel grafo que no contiene ningún ciclo.

Si hay un camino desde el nodo *u* al nodo *v*, entonces diremos que *v* es ***accesible*** desde *u*.

Un ***grafo*** G se dice que es ***conexo***, si al menos existe un camino entre cada par de nodos del grafo.

Árbol es un grafo acíclico conexo.

Un grafo G está etiquetado, si sus aristas tienen datos asignados. En particular, el grafo G tiene peso, también llamado ***grafo valorado*** o ***ponderado o con peso***, si cada arista *e* de G tiene asignado un valor numérico, *w(e)*, llamado peso o longitud de *e*.
Una asignación de pesos a un grafo G=(V,E), viene dada por una aplicación $w:\{(u,v)\in E\}\rightarrow R^{+}\cup\{0\}$

En los grafos ponderados el peso de un camino P de un nodo *u* a un nodo *v*, es la suma de los pesos de las aristas que unen los nodos que forman el camino. El camino de menor peso, entre dos nodos, se denomina ***camino mínimo***. Si el grafo no es valorado, el camino mínimo entre dos nodos es aquel que contiene el menor número de aristas.

[1] Heileman, Gregory L. **Estructuras de Datos, Algoritmos y Programación Orientada a Objetos.** McGraw Hill. Año 1998. 306 páginas. Pág.280.

Un ***grafo dirigido*** o ***digrafo***, G=(V,E), es tal que cada arista *e* de G tiene una dirección asignada; es decir, cada arista *e* está identificada por un *par ordenado* (u,v) de nodos de G.

Consideremos el arco *e=(u,v)* del digrafo G, entonces:
- *e* empieza en *u* y termina en *v*.
- *u* es el origen o punto inicial de *e*, y *v* es el destino o punto terminal de *e*.
- *u* es un predecesor de *v* y *v* es sucesor de *u*.

En un digrafo G, el ***grado de salida*** de un nodo *u*, también llamado gredo de emisión, grad_sal(u), es el número de aristas que empiezan en u.

En un digrafo G, el ***grado de entrada*** de un nodo *u*, también llamado grado de recepción, grad_ent(u), es el número de aristas que terminan en u.

En un digrafo G, un nodo *u* es ***nodo fuente***, si grad_sal(u)>0 y grad_ent(u)=0.

En un digrafo G, un nodo *u* es ***nodo sumidero***, si grad_ent(u)>0 y grad_sal(u)=0.

Camino entre los nodos u y v, es una secuencia de nodos P=(v_0, v_1,...,v_n), u=v_0 y v=v_n tal que (v_0,v_1) $\in$ E, (v_1,v_2) $\in$ E,..., (v_{n-1},v_n) $\in$ E. Las nociones camino y camino mínimo presentadas en este documento, son reconocidas como trayectoria y pista, respectivamente, en Matemática Discreta.

Un ***digrafo* G *es fuertemente conexo***, si cualquiera dos nodos del grafo dirigido son accesibles el uno desde el otro. Es decir, para cada par de nodos *u,v* $\in$V, existe un camino de u a *v* y un camino de *v* a *u*.

Un ***dígrafo G es simple conexo***, si para cada par de nodos *u,v* $\in$V, existe un camino de u a *v* ó un camino de *v* a *u*.

Situaciones reales modelables por medio de Grafos

Para modelar una red de comunicaciones como un grafo, representamos a cada miembro de la colección - computadoras, aeropuertos, ciudades, terminales, depósitos u otras entidades que pretendan comunicarse- como un nodo, y generamos una arista –dirigida o no dirigida dependiendo de si la comunicación es bidireccional o no- entre los miembros que puedan comunicarse de manera directa. Las capacidades conocidas, de los canales de comunicación en una red de transporte -ancho de banda, distancias, tiempos por ejemplo-, pueden ser modeladas en el grafo como pesos de las aristas. El grafo resultante será valorado, siempre que se necesite considerar la cantidad de carga que pueda ser transportada entre los distintos sitios, o las distancias existentes entre ellos, entre otros requerimientos.
Hipertextos o multimedias también pueden ser modelados por medio de grafos, en los que los nodos representan a cada una de sus páginas y las aristas a los vínculos existentes entre ellas.
Las relaciones de precedencia que pueden existir entre las tareas que deben realizarse para completar un trabajo, también pueden modelarse por medio de grafos. En este caso,

cada nodo representa a una tarea, y existirá una arista dirigida del nodo v_i al nodo v_j, si la tarea asociada con v_i debe ser completada antes que la tarea asociada con v_j. Grafos de precedencia son usados para modelar las tareas involucradas en una construcción edilicia o en computaciones. En este caso los nodos representan las operaciones que deben ejecutarse, y las aristas las dependencias entre dichas operaciones. Un requerimiento frecuente relacionado con estos grafos, es encontrar una planificación posible de tareas.
La ventaja de esta perspectiva radica en que una vez modelada una situación particular por medio de un grafo, la teoría de grafos provee el marco teórico en el que se resuelve el problema.
Como con otros TADs, el grafo o dígrafo que modele un problema particular, puede coincidir con los resultantes de otros problemas sin conexión inicial aparente.

T.A.D. GRAFO

a) Especificaciones

NOMBRE	*ENCABEZADO*	*FUNCION*	*ENTRADA*	*SALIDA*
Adyacentes	Adyacentes(G,u)	Determina los nodos adyacentes de u	G y u	Reporta los nodos adyacentes a u.
Camino	Camino(G,u,v)	Determina el camino de u a v	G, u y v	Reporta el camino de u a v, si v es alcanzable desde u; Error en caso contrario
Conexo	Conexo(G)	Evalúa si G es conexo	G	V si G es conexo, F en caso contrario
Acíclico	Acíclico(G)	Evalúa si G es acíclico	G	V si G es acíclico, F en caso contrario
BEA*	BEA(G)	Procesa todos los elementos de G en anchura	G	Está sujeta al proceso que se realice sobre los elementos de G
BEP**	BEP(G)	Procesa todos los elementos de G en profundidad	G	Está sujeta al proceso que se realice sobre los elementos de G

* BEA: Recorrido en Anchura
**BEP: Recorrido en Profundidad

Actividad
Realice las especificaciones de las operaciones del TAD Dígrafo.

b) Representación

- *Representación secuencial*: En este caso el grafo G es representado por medio de una ***matriz de adyacencia***. Una matriz de adyacencia de un grafo G=(V,E) se almacena en un arreglo bidimensional A de |V| x |V| componentes, donde

$$A[i,j] = \begin{cases} 1 & si\ (i,j) \in E \\ 0 & si\ (i,j) \notin E \end{cases}$$

Si G es un grafo con peso, entonces puede ser representado por medio de una matriz de pesos que se almacena en un arreglo bidimensional W de |V| x |V| componentes donde:

$$W[i,j] = \begin{cases} w & si\ (i,j) \in E \\ 0 & si\ (i,j) \notin E \end{cases}$$

Actividad

1- Represente por medio de una matriz de adyacencia el grafo G=(V,E), en el que $V=\{v_1,v_2,v_3,v_4,v_5\}$ y $E=\{(v_1,v_2),(v_1,v_4),((v_2,v_5),(v_3,v_5),(v_4,v_5)\}$
2- ¿Qué propiedades posee la matriz adyacencia de un grafo G? Justifique su respuesta.

- *Representación encadenada:* Para esta situación proponemos la representación de ***listas de adyacencia***, que consiste de un arreglo de |V| listas de nodos adyacentes, donde cada v_i es una lista enlazada de nodos adyacentes al nodo v_i .

Actividad

1- Represente por medio de listas de adyacencia el grafo G del ítem 1 de la actividad precedente.
2- Analice cuidadosamente las dos representaciones:
   a- ¿Qué cantidad de memoria se requiere para la representación de un grafo G por medio de matriz de adyacencia?
   b- ¿Qué cantidad de memoria se requiere para la representación de un grafo G por medio de listas de adyacencia?

3- Si G es un grafo dirigido, ¿qué cantidad de memoria requiere cada una de las representaciones?

c) Construcción de las operaciones abstractas

Las operaciones responden a los problemas más comunes de grafos. Entre las mas importantes se encuentran: Conexo, Camino, Camino mínimo y Ordenación Topológica.

- Conexo : Si un grafo particular es conexo, podrá responderse afirmativamente a la pregunta ¿es posible desplazarse de un sitio a cualquier otro sitio?, si el grafo

modela una red de transporte, o a la pregunta ¿es posible transmitir información desde una computadora a cualquier otra?, en el caso de que el grafo modele una red de computadoras.

- Accesibilidad: Determinar si un nodo puede ser accedido desde otro nodo es importante cuando debemos responder a preguntas como la siguiente: ¿es posible desplazarse de un sitio a otro sitio particular?, si el grafo modela una red de transportes. Específicamente, a partir de un grafo G=(V,E), un nodo de origen *s* y un nodo destino *d*, *s,d* $\in$ V, el problema de accesibilidad se traduce en determinar si existe camino desde *s* hasta *d*.
- Caminos mínimos: No solo es importante determinar si existe camino entre dos nodos, sino que en ocasiones también resulta conveniente encontrar el camino con el menor número de aristas, si el gafo es un grafo no valorado o el camino con el menor peso total si el grafo es valorado.
- Ordenación Topológica: A partir de un grafo dirigido acíclico, la ordenación topológica permite obtener una planificación posible de tareas. Esta ordenación produce un orden lineal de V, tal que si (u,v) $\in$ E, entonces *u* aparece antes que *v*.

Algoritmos

A continuación presentamos un conjunto de algoritmos que pueden ser usados en la implementación de las operaciones propuestas.

En general, examinar las propiedades de un grafo G requiere analizar los nodos y las aristas de dicho grafo. Se proponen dos criterios de análisis a partir de un nodo $s \in V$. El primer criterio, contemplado en el algoritmo BEA, sostiene que todos los nodos adyacentes a *s* serán examinados primero, después serán examinados los nodos adyacentes a estos y así sucesivamente. Para ello el algoritmo hace uso de una cola. El otro criterio, presentado en BEP, hace uso de una pila, por lo que los nodos de V serán examinados desde *s* en profundidad.

- Búsqueda en Anchura (BEA)[2]: Este algoritmo utiliza un arreglo d[1..|V|], para almacenar los nodos marcados y los nodos no marcados de V. Al terminar el algoritmo, para cada nodo marcado *u* $\in$ V, d [u] contendrá el mínimo número de aristas de cualquier camino desde el origen *s* hasta *u*. Los nodos no marcados no son accesibles, por lo que este algoritmo puede ser usado para resolver el problema de accesibilidad y de camino mínimo con origen único para grafos no valorados.

```
BEA (G, s)        s es el origen
   Para cada v ∈ V hacer
        d[ v ] ←  ∞            todos los nodos están no marcados
   d[s] ←  0                   marcar el origen
   Insertar ( Cola , s)
   Mientras no Vacía ( Cola)  hacer
         Suprimir (Cola , v)
          Para Cada u Adyacente a v  hacer
                Si  d [ u ] = ∞
                      entonces    d [ u ] ←  d [ v ] + 1      marcar el nodo u
                                  Insertar (Cola, u)
```

[2] Heileman, Gregory L. Op. cit. pág. 254.

Este algoritmo permite detectar los nodos no accesibles, por lo que favorece el análisis de Conexidad.

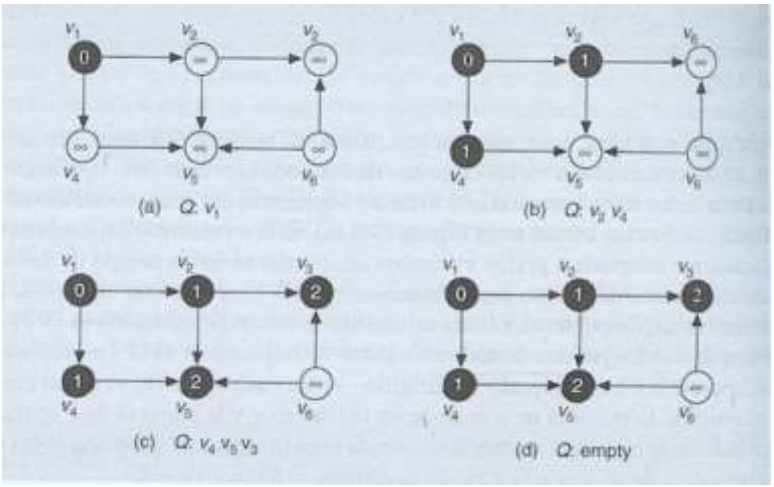


Figura extraída de Estructuras de Datos, Algoritmos y Programación Orientada a Objetos. Pag. 254.
Una ejecución ejemplo de BEA() sobre un grafo dirigido con nodo origen v_1. En cada paso del algoritmo, los nodos actualmente no marcados están en blanco y los marcados en negro. Dentro de cada nodo marcado, tal y como se calcula en la línea 9 de BEA(), está el número mínimo de aristas de cualquier camino desde el origen. (a) Inicialmente sólo el origen está marcado y la cola almacena v_1. (b) El resultado de procesar el nodo v_1. Los nodos v_2 y v_4 se marcan y añaden a la cola. (c) El resultado de procesar el nodo v_2 . Los nodos v_5 y v_3 se marcan y añaden a la cola. (d) El resultado final, después de procesar los nodos v_4 , v_5 y v_3 . Este procesamiento no provoca que se marque ningún nodo adicional. Obsérvese que v_6 no es accesible desde v_1 y que no está marcado en el algoritmo.

Actividad

Modifique el algoritmo BEA, para facilitar la reconstrucción del camino desde el origen s, a cualquier nodo alcanzable v.

- Búsqueda en Profundidad (BEP) [3]: Este algoritmo utiliza la recursión en lugar de emplear una pila. BEP usa un arreglo d que contendrá los momentos en que los nodos son descubiertos. También usa un arreglo f que informa los momentos en que es completado el tratamiento de los nodos.

```
BEP (G)
    Para cada v ∈ V hacer
          d[v] ←  0          d[v]=0   Todos los nodos están sin descubrir
    tiempo ←  0              Tiempo es una variable global
    Para cada s ∈ V hacer
          Si d[s] = 0
                entonces
                  BEP-Visita (G,s)
BEP- Visita (G, s)
```

[3] Heileman, Gregory L. op.cit. pág 255.

```
tiempo ⟵ tiempo + 1
d[ s] ⟵ tiempo                    el nodo s es descubierto y marcado
Para Cada u Adyacente a s  hacer
      Si d[ u ] = 0
                entonces
                    BEP-Visita (G,u)
tiempo ⟵ tiempo + 1
f [ s ] ⟵ tiempo                  finaliza el procesamiento del nodo s
```

Cada vez que un nodo *s* es seleccionado en BEP, el algoritmo busca en profundidad hasta encontrar un nodo no marcado, es decir, no descubierto. El algoritmo BEP garantiza que todo nodo del grafo será eventualmente marcado, aún si el grafo no es conexo.

Heileman sugiere que BEP puede ser usado para determinar propiedades de un grafo – por ejemplo si un grafo es acíclico o no-, a partir de la clasificación de sus aristas. La clasificación sugerida contempla, entre otros, los siguientes tipos de aristas:

- Aristas de árbol: Estas son cualquier colección de aristas de un grafo que formen un bosque-grafo acíclico. Todo nodo del grafo es un árbol con un único nodo con respecto a esta colección, o es parte de algún árbol mas grande por medio de su conexión a otro nodo vía una arista de árbol. Téngase en cuenta que esta colección no es única.
- Aristas hacia atrás: Dada una colección de aristas de árbol, las aristas hacia atrás de un grafo son aquellas aristas que conectan algún nodo descendiente de un árbol con un nodo antepasado del mismo árbol.

En un grafo no dirigido, una búsqueda en profundidad solo producirá aristas de árbol y aristas hacia atrás.

Un grafo no dirigido es acíclico si no tiene aristas hacia atrás, pues una arista hacia atrás conecta un nodo descendiente con un nodo antepasado, lo que formaría un ciclo. Esta afirmación también es cierta para grafos dirigidos.

Seguidamente presentamos las reglas que propone Heileman para clasificar las aristas de un grafo usando BEP:

- Cuando un nodo descubierto encuentra un nodo no descubierto, etiquetar la arista entre ellos como arista de árbol.
- Cuando un nodo descubierto encuentra otro nodo descubierto pero no terminado, etiquetar la arista entre ellos como arista hacia atrás.

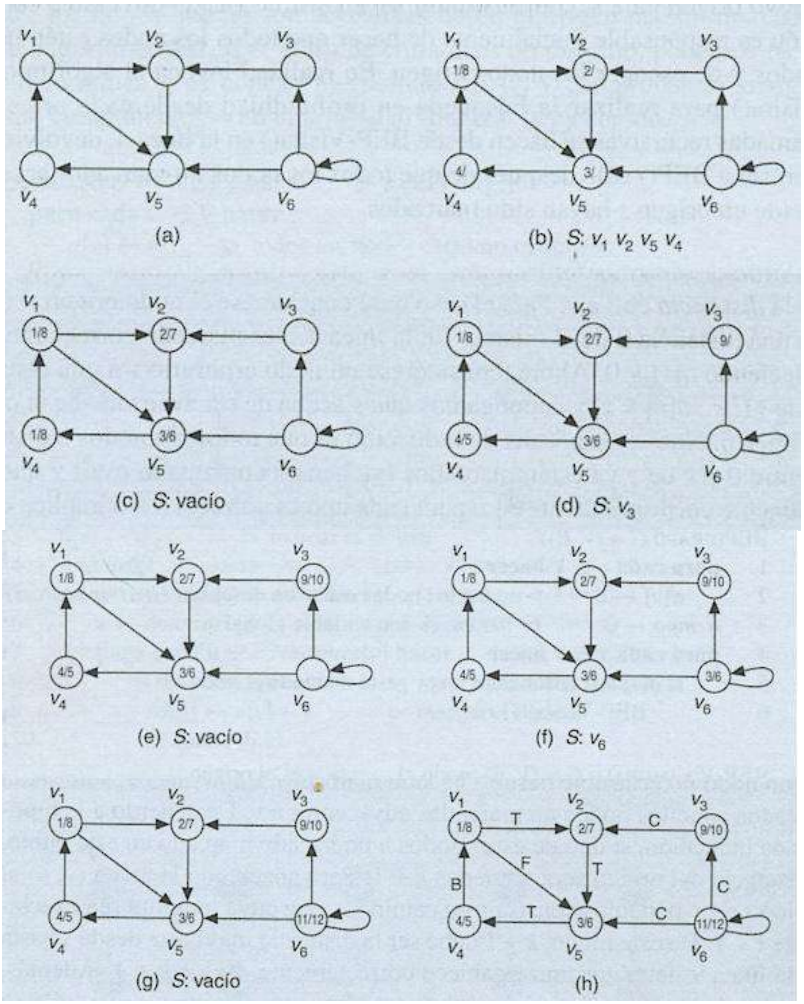


Figura extraída de Estructuras de Datos, Algoritmos y Programación Orientada a Objetos. Pag.256.

Una ejecución ejemplo de BEP() sobre un grafo dirigido. El tratamiento de los nodos comienza en v_1 (también podría haber empezado desde cualquier otro nodo del grafo). Los momentos de descubrimiento y de terminación para cada nodo v se muestran dentro de él usando el formato d[v]/f[v]. Si el interior de un nodo está vacío, entonces es un nodo no descubierto. El conjunto S será utilizado para denotar la colección de nodos marcados actualmente mantenidos en la pila de ejecución. (a) El grafo de entrada. (b) El estado después de que los cuatro primeros nodos han sido descubiertos. Comenzando desde el origen v_1, no pueden descubrirse nodos no marcados adicionales. (c) El estado después de que ha terminado el tratamiento de los primeros cuatro nodos descubiertos. (d) La búsqueda continúa desde el siguiente nodo origen v_3. (e) Todos los nodos adyacentes a v_3 ya han sido descubiertos; así su tratamiento se completa en el siguiente paso. (f) La búsqueda desde el último nodo origen v_6 comienza en el 11. ° intervalo de tiempo. (g) El estado después de que todos los nodos han sido procesados. (h) La clasificación de aristas ofrecida por BEP() , donde T es una arista de árbol, B es una arista hacia atrás....

Actividad

Modifique el algoritmo BEP, para que clasifique las aristas de un grafo.

- Ordenación Topológica[4] : El algoritmo BEP puede ser utilizado para realizar una ordenación topológica sobre un grafo dirigido acíclico recibido como entrada.

```
Ordenación - Topológica ( G )
    Ejecutar BEP ( G ) insertando nodos a la cabeza de la lista L conforme
        son terminados
    Retornar  L                    L contiene la Ordenación Topológica de G
```

- Algoritmo de Dijkstra: A continuación se presenta una versión adaptada, correspondiente al pseudocódigo Pascal del algoritmo de Dijkstra[5]. Este algoritmo resuelve el problema de camino mínimo, en un grafo valorado que contiene aristas con pesos no negativos:

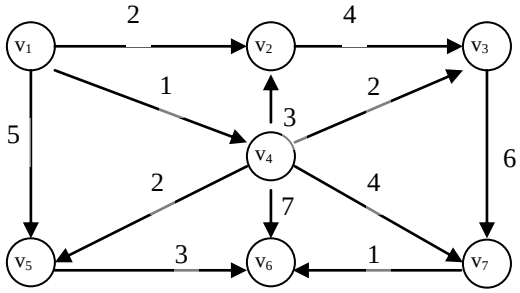
```
Dijkstra(T: Tabla);

Para i desde 1 hasta |V| hacer
        v ⟵  vertice con la distancia mas corta y desconocido
        T[v].conocido ⟵  True
        Para   cada w adyacente a v  hacer
              Si T[w].conocido = False
                    entonces

                  Si  T[v].distancia + w(v,w) < T[w].distancia
                      entonces
                      Reducir ( T[w].distancia  a   T[v].distancia + w(v,w))
                      T[w].camino ⟵   v
                  FinSi
             FinSi
           FinPara
FinPara
```

Cada componente de la tabla T tiene, al menos, los siguientes campos: conocido- de tipo booleano-, camino y distancia. Además, podría contener la lista de adyacentes, si se elige la representación enlazada del objeto de datos.

Previo a la ejecución del algoritmo, los campos conocido, camino y distancia deben ser inicializados con los valores False, 0- si los nodos se representan numéricamente-, y Maxint respectivamente. Tratamiento especial debe tener el nodo inicial, para el cual el campo distancia debe ser iniciado en 0.

A continuación presentamos un ejemplo de aplicación del algoritmo de Dijkstra.

[4] Heileman, Gregory L. op. cit.. pág. 257
[5] Weiss, Mark Allen. **Estructuras de Datos y Algoritmos.** Addison-Wesley Iberoamericana. Año 1995. 490 páginas. Pag. 314



Valores iniciales de la tabla T

Vértices	Conocido	Distancia	Camino
v_1	F	0	
v_2	F	∝	
v_3	F	∝	
v_4	F	∝	
v_5	F	∝	
v_6	F	∝	
v_7	F	∝	

Instancias intermedias de T

- i=1, v=v_1, w=v_2,v_4,v_5

Vértices	Conocido	Distancia	Camino
v_1	T	0	
v_2	F	**2**	**v_1**
v_3	F	∝	
v_4	F	**1**	**v_1**
v_5	F	**5**	**v_1**
v_6	F	∝	
v_7	F	∝	

- i=2, v=v_4, w=v_2,v_3,v_5,v_6,v_7

Vértices	Conocido	Distancia	Camino
v_1	T	0	
v_2	F	2	v_1
v_3	F	**3**	**v_4**
v_4	**T**	1	v_1
v_5	F	**3**	**v_4**
v_6	F	**8**	**v_4**
v_7	F	**5**	**v_4**

- i=3, v=v_2, w=v_3

Vértices	Conocido	Distancia	Camino
v_1	T	0	
v_2	**T**	2	v_1
v_3	F	3	v_4
v_4	T	1	v_1
v_5	F	3	v_4
v_6	F	8	v_4
v_7	F	5	v_4

- i=4, v=v_3, w=v_7

Vértices	Conocido	Distancia	Camino
v_1	T	0	
v_2	T	2	v_1
v_3	**T**	3	v_4
v_4	T	1	v_1
v_5	F	3	v_4
v_6	F	8	v_4
v_7	F	5	v_4

- i=5,v=v_5, w=v6

Vértices	Conocido	Distancia	Camino
v_1	T	0	
v_2	T	2	v_1
v_3	T	3	v_4
v_4	T	1	v_1
v_5	**T**	3	v_4
v_6	F	8	v_4
v_7	F	5	v_4

- i=6,v=v_7, w=v_6

Vértices	Conocido	Distancia	Camino
v_1	T	0	
v_2	T	2	v_1
v_3	T	3	v_4
v_4	T	1	v_1
v_5	T	3	v_4
v_6	F	**6**	**v_7**
v_7	**T**	5	v_4

- i=7,v=v_6, w=

Vértices	Conocido	Distancia	Camino
v_1	T	0	
v_2	T	2	v_1
v_3	T	3	v_4
v_4	T	1	v_1
v_5	T	3	v_4
v_6	**T**	6	v_7
v_7	T	5	v_4

Actividad

Diseñe el algoritmo que reporte el peso correspondiente al camino mínimo asociado con cada vértice.

A continuación se presentan otros algoritmos que se apoyan en la representación secuencial de grafos.

- Matriz Potencia: A partir de la matriz de adyacencia A que representa a un grafo G, se pueden generar las matrices potencia A^2, A^3, A^4,....., A^k. En general, a_k (i , j) - entrada i, j en la matriz A^k – contiene el numero de caminos de longitud k desde v_i a v_j.

- Algoritmo de Warshall: Algoritmo mas eficiente que el presentado en el punto anterior para encontrar la matriz de caminos P a partir de la matriz de adyacencia A de G.

$P_k [i , j]$ {

1 Si existe un camino simple de v_i a v_j que no usa otros nodos aparte de $v_1, v_2, ..., v_k$

0 En otro caso

Es decir : $P_0 [i,j] = 1$ Si hay arista de v_i a v_j

$P_1 [i,j] = 1$ Si hay un camino simple de v_i a v_j que no usa otros nodos excepto posiblemente v_1.

.....

En general: $P_0 = A$ y $P_n = P$ matriz de caminos de G

$P_k [i , j] = 1$ si {

1) Existe un camino simple de *vi* a *vj* que no usa otros nodos excepto $v_1, v_2, ..., v_{k-1}$:

$P_{k-1} [i, j] = 1$

ó

2) Existe un camino simple de v_i a v_k y otro camino simple de v_k a v_j que no usan otros nodos excepto posiblemente $v_1, v_2, ..., v_{k-1}$:

$P_{k-1} [i , k] = 1$ *y* $P_{k-1} [k , j] = 1$

Los elementos de la matriz P_k pueden obtenerse por:

$$P_k [i , j] = P_{k-1} [i, j] \vee (P_{k-1} [i , k] \wedge P_{k-1} [k , j])$$

```
WARSHALL ( A, P)
    P=A
    Para  1<= k <=n hacer
        Para  1<= i <=n hacer
            Para  1<= j <=n hacer
              P [ i , j] = P [ i, j ]  ∨ (P [ i , k ]  ∧ P  [ k , j ])
```

- Si partimos de un grafo con peso, este se puede representar a partir de su matriz de pesos $W = w_{ij}$ definida como

w_{ij} {

w (e) si hay una arista *e* de v_i a v_j

0 si no hay arista de v_i a v_j

A partir de la matriz de pesos, se desea encontrar la matriz de caminos mínimos Q , donde q_{ij} contiene la longitud del camino mínimo de v_i a v_j.

Ahora en lugar de encontrar las matrices P_0, P_1,....., P_k , encontraremos las matrices Q_0, Q_1,....., Q_k donde

q_k [i , j] = La menor de las longitudes de los anteriores caminos de v_i a v_j

ó

La suma de las longitudes de los anteriores caminos v_i a v_k y v_k a v_j

Es decir: q_k [i , j] = Mínimo (q_{k-1} [i , j] , q_{k-1} [i , k] + q_{k-1} [k , j])

```
CAMINO_MINIMO ( W, Q)
        Q = W
        Para   1<= k <=n hacer
                Para   1<= i <=n hacer
                        Para   1<= j <=n hacer
                          Q[ i , j] = Mínimo (Q [ i , j] , Q [ i , k] + Q [ k , j] )
```

Actividad

Compare los algoritmos de Warshall y de Camino Mínimo.