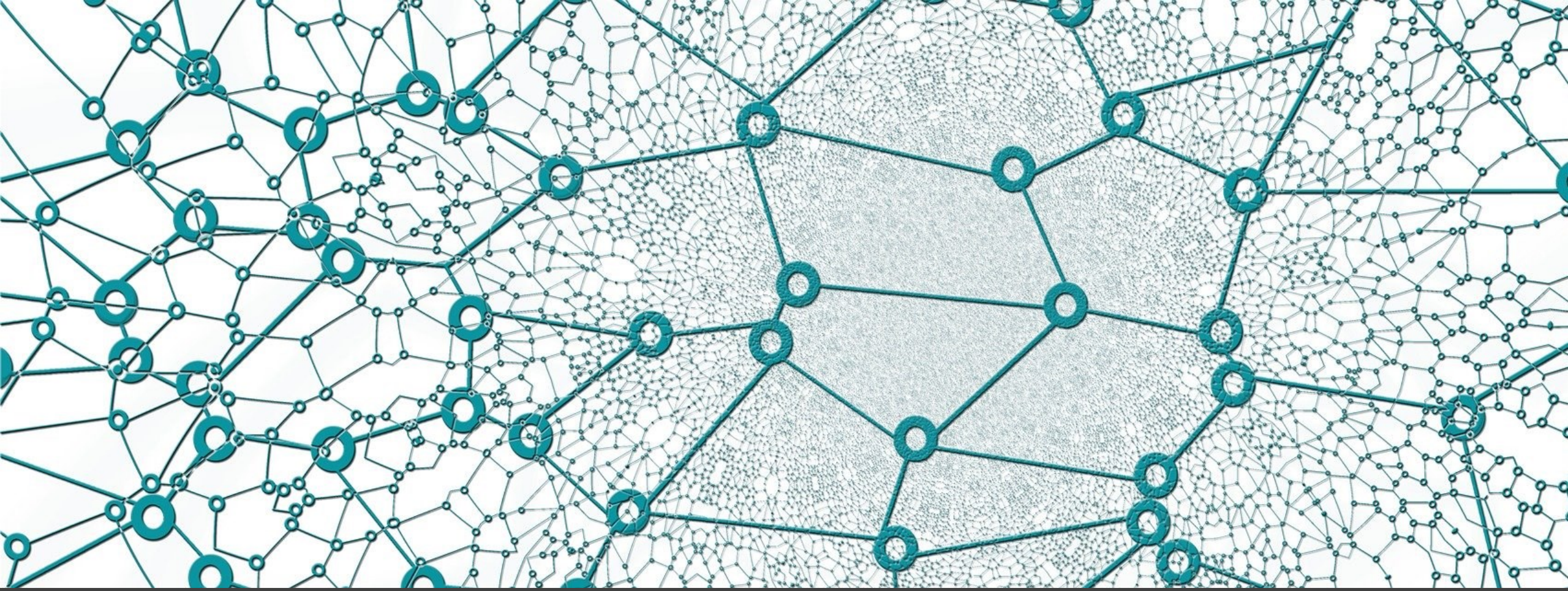


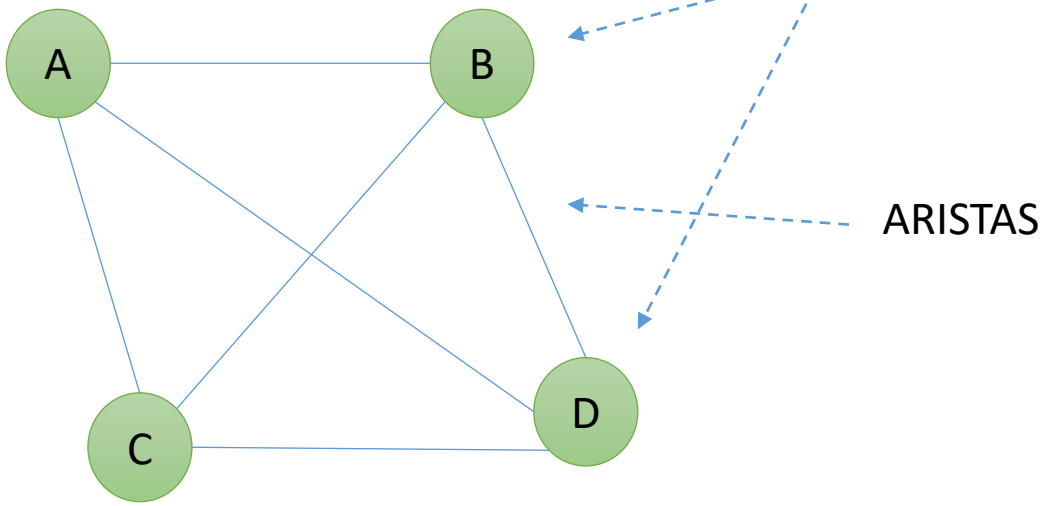
GRAFOS

Estructura de Datos

Sesión 12

GRAFOS

- Los grafos son estructuras de datos no lineales donde cada componente puede tener mas de uno o mas predecesores y sucesores.



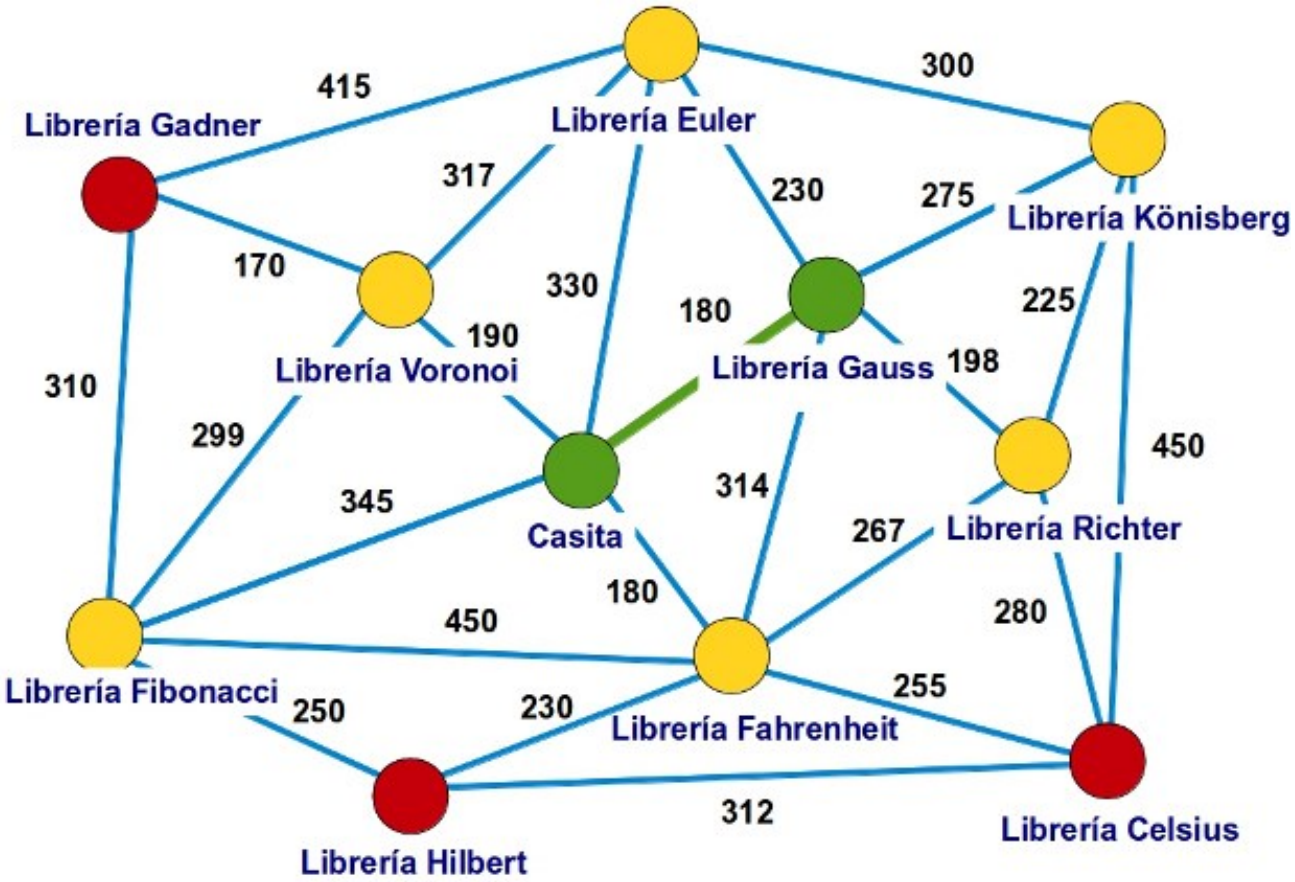
Vértice A
Vértice B
Vértice C
Vértice D

Arista AB (a,b)
Arista AC (a,c)
Arista AD (a,d)
Arista BC (b,c)
Arista BD (b,d)
Arista CD (c,d)

Grafo con 4 VERTICES y 6 ARISTAS

Grafos

- Vértices: Almacenan la Informacion
- Aristas: Representan la relación entre los vertices

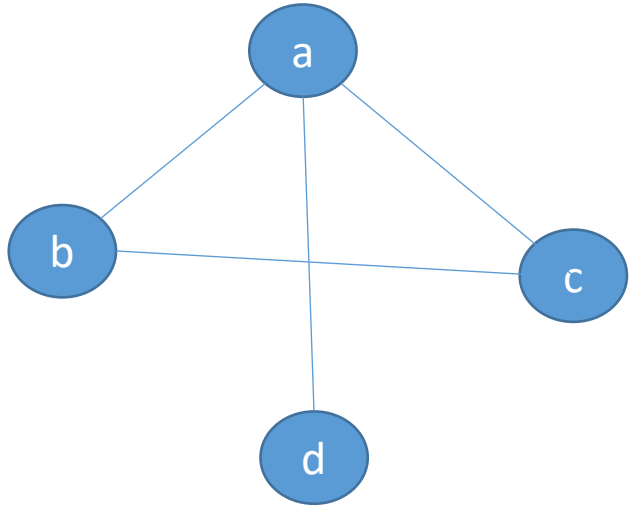


Conceptos Básicos de Grafos

Grado de un vértice:

El grado de un vértice v escrito como grado(v), es el numero de aristas que contienen a v; es decir que contienen a v como extremo

- Si el grado(v)=0 se dice que no tiene aristas y se dice que v es un vértice aislado
- Ejm.

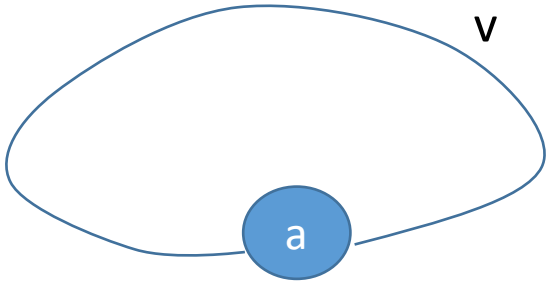


Grado(a)=3
Grado(b)=2
Grado(c)=2
Grado(d)=1

Conceptos Básicos de Grafos

Lazo o Bucle:

- El lazo o bucle es una arista que conecta a un vértice consigo mismo.
- Es decir a=(v,v)



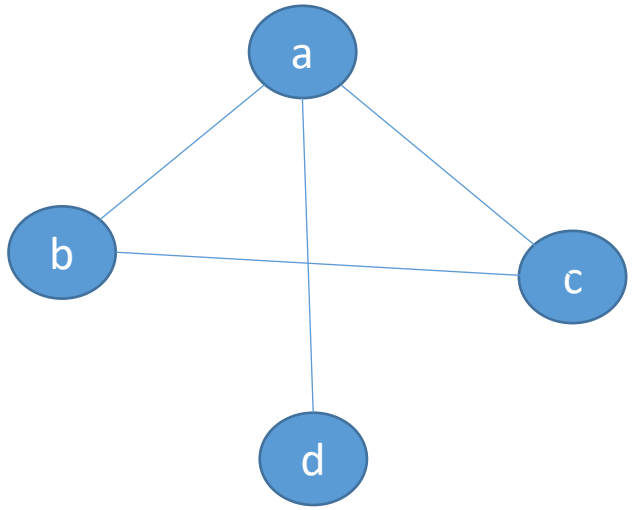
Conceptos Básicos de Grafos

Camino

- Un camino **P** de longitud n se define como la secuencia de **n** vértices que se debe seguir para llegar a llegar del vértice v_1 origen al vértice v_n destino

$$P=(v_1, v_2, \ldots\ldots v_n)$$

De tal modo que v_i es adyacente a v_{i+1} para i={1,2,3,….n}



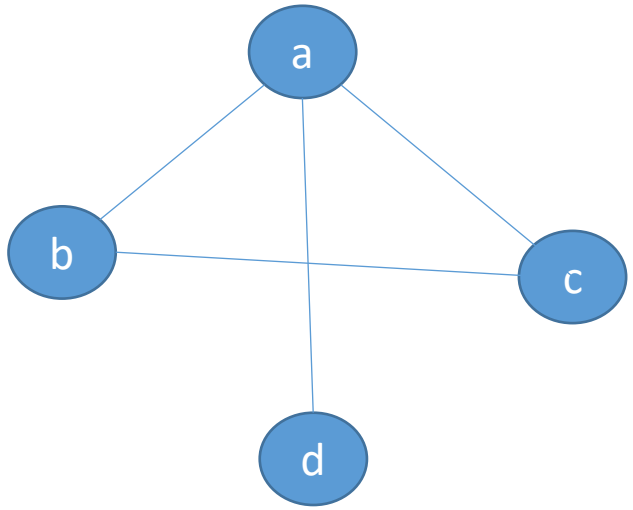
Camino P del vértice b al vértice c

P=(b,a,c)
P=(b,c)

Conceptos Básicos de Grafos

Camino cerrado

- Un camino **P** es cerrado si el primero y el ultimo vértice son iguales.
- Es decir $v_1 = v_n$



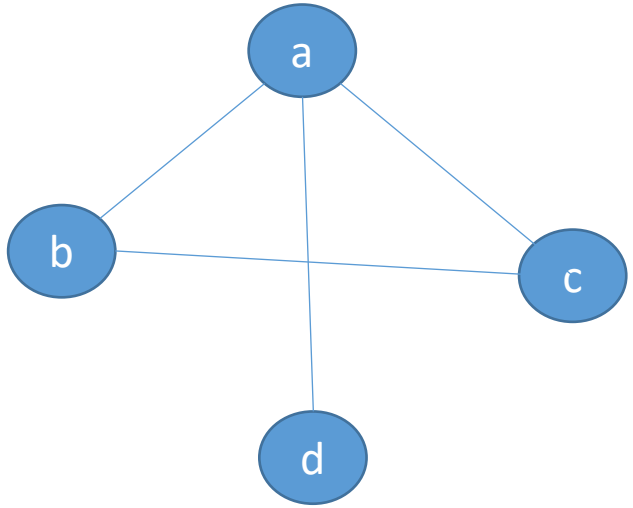
Camino cerrado P del vértice a al vértice a

P=(a,b,c,a)

Conceptos Básicos de Grafos

Camino simple

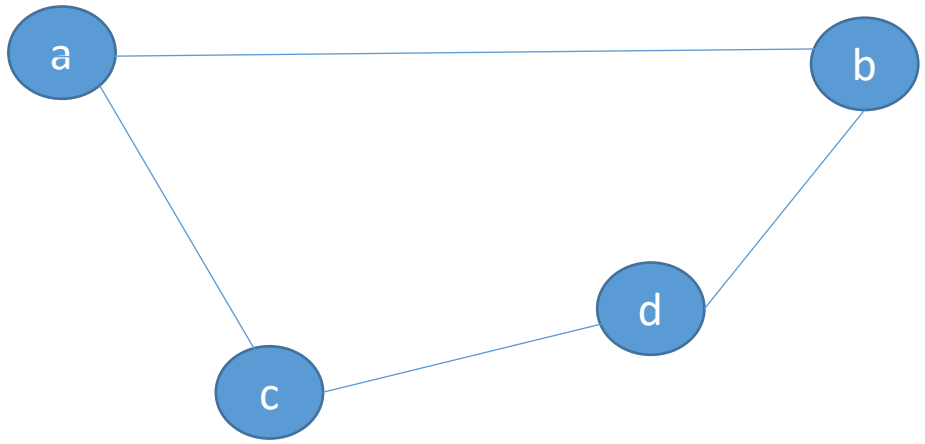
- Un camino **P** es simple si todos los vértices son distintos con excepción del primero y del ultimo que pueden ser iguales.
- Es decir P es simple si v_1 ,v_2 ,… v_{n-1} son distintos



Conceptos Básicos de Grafos

Ciclo

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



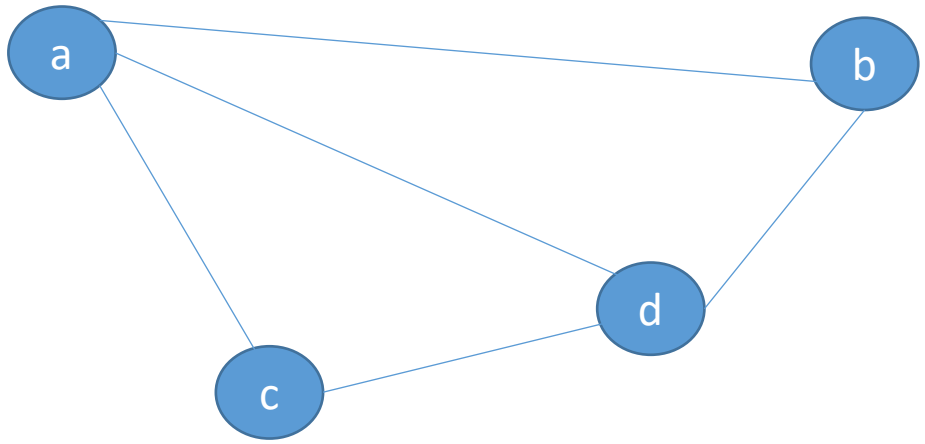
ciclo de longitud **4**

4-ciclo

Conceptos Básicos de Grafos

Grafo conexo

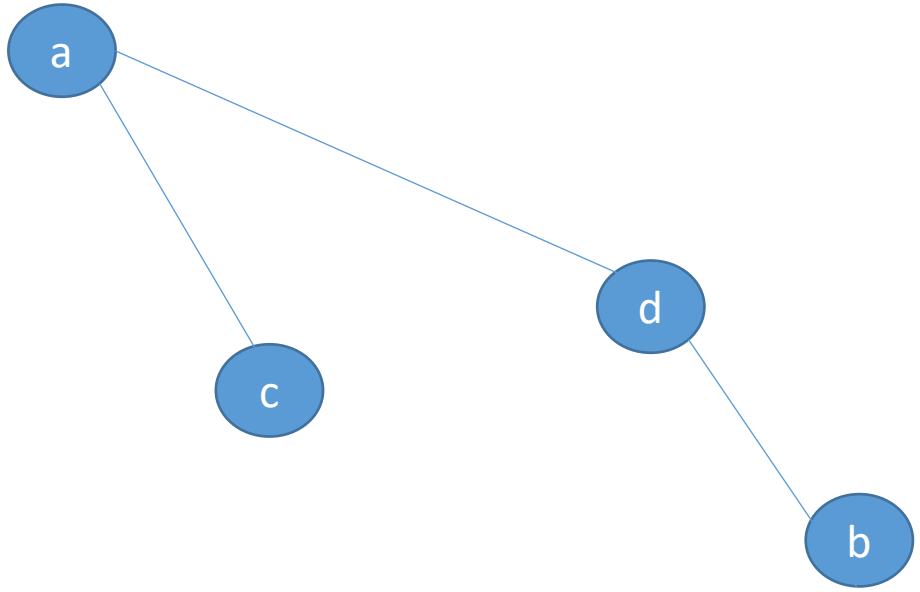
- Se dice que un grafo es conexo si existe un camino simple entre cualesquiera de sus vertices



Conceptos Básicos de Grafos

Grafo árbol

- Se dice que un grafo G es de tipo árbol o árbol libre si G es un grafo conexo sin ciclos

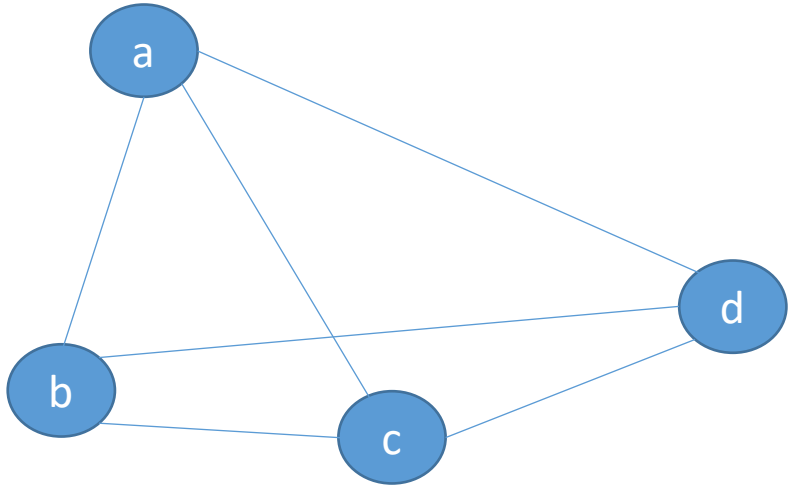


Conceptos Básicos de Grafos

Grafo completo

- Se dice que un grafo G es completo si cada vertice *v* de G es adyacente a todos los demás vértices de G.
- Un grafo completo de n vértices tendrá

$$n(n-1)/2 \text{ aristas}$$



Un grafo completo de 4 vértices tendrá

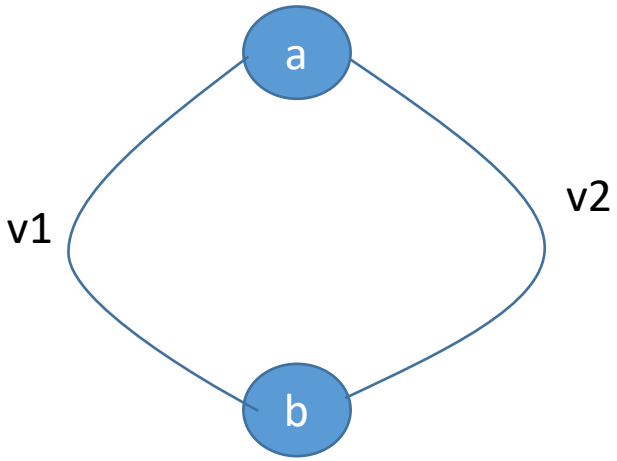
$4(4-1)/2$ aristas

= 6 aristas

Conceptos Básicos de Grafos

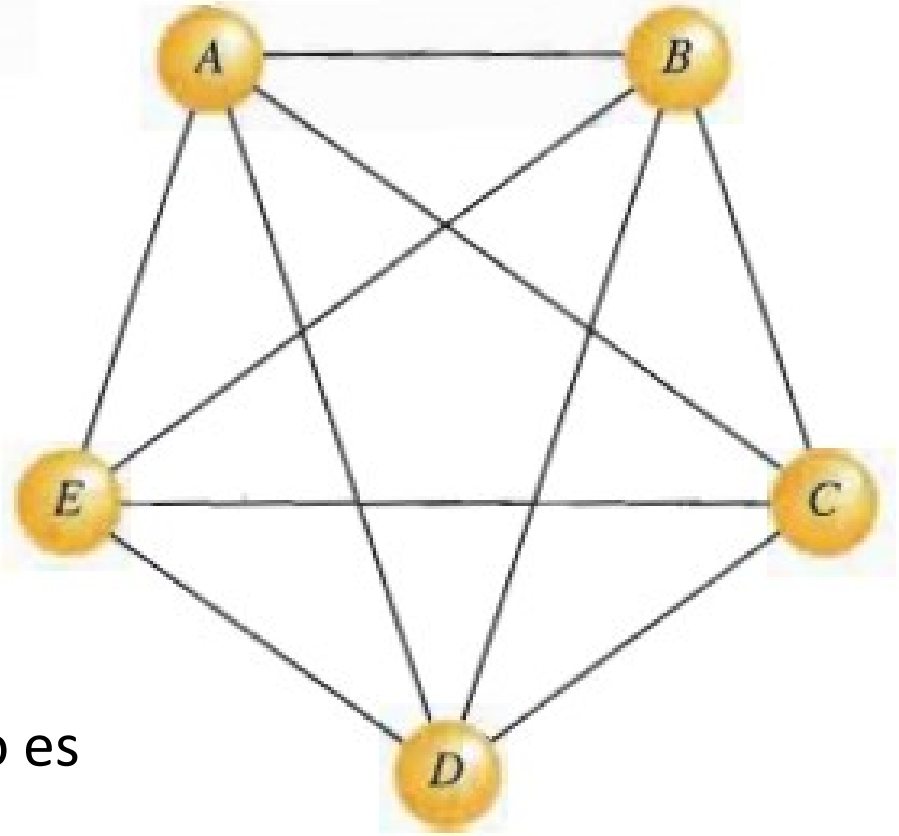
Multigrafo

- Un grafo se denomina multigrafo si al menos dos de sus vértices están conectados entre si por medio de dos aristas.
- En este caso las aristas reciben el nombre de aristas múltiples o paralelas



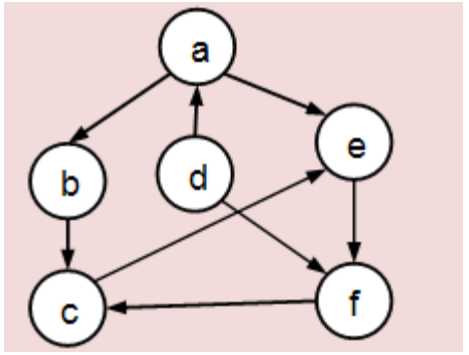
Grafos

- Terminología



- Todos los vértices tienen grado 4
- Camino P para llegar del vértice A al D pueden ser
  A-B-C-D, A-E-D, A-D
- El camino A-C-D-A es camino cerrado, el A-C-D no lo es
- El camino A-C-D-A es un camino simple, el A-C-B-D-C no lo es
- El camino A-C-D-A es un ciclo
- Es un grafo conexo, pues todos los vértices tienen al menos un camino a otro vértice.
- Es un grafo completo, pues todos los vértices se conectan con los otros vertices

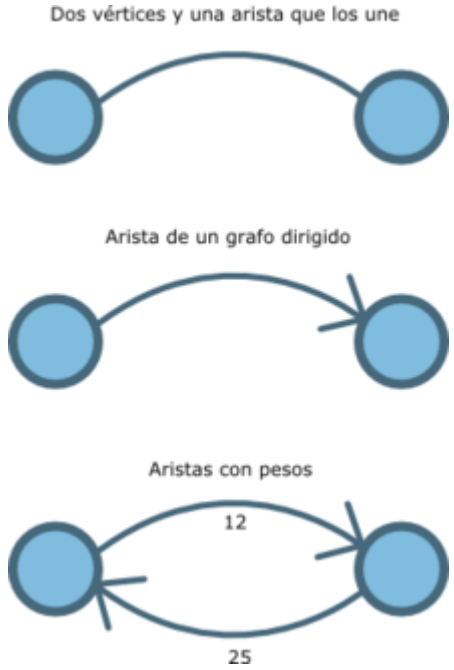
Grafos Dirigidos



- Llamados también DIGRAFOS, se caracterizan porque sus aristas tienen asociados una dirección, cada arista esta asociada a un par ordenado (u,v) de vértices de G.
- Una arista dirigida a=(u,v) se llama arco y generalmente se expresa como u -> v
- Los vértices se utilizan para representar información
- Las aristas representan una relación con dirección o jararquia entre aquellos
- Ejemplo: Duraciones de vuelos entre ciudades

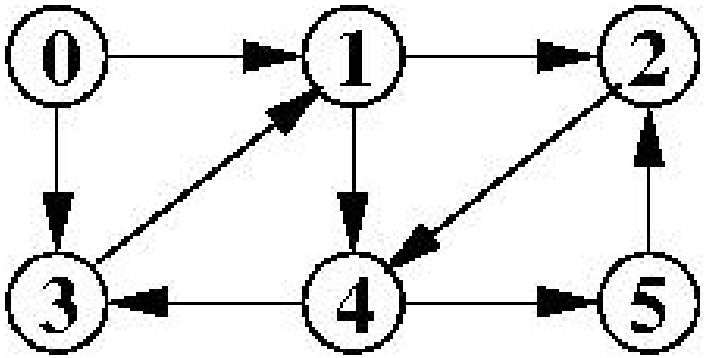
Grafos Dirigidos

- Representación de una arista dirigida



- **a** empieza en **u** y termina en **v**
- **u** es el origen o punto inicial de **a** y **v** es el destino o punto terminal de **a**
- **u** es el predecesor de **v** y **v** es sucesor o vecino de **u**
- **u** es adyacente hacia **v** y **v** es adyacente desde **u**

Representación de Grafos



- Hay tres maneras comunes para representar los grafos; vamos a tomar el siguiente grafo como ejemplo:

1) **Matriz de adyacencia**: las aristas del grafo se guardan en una matriz, indicando si un nodo tiene enlace directo con otro.
2) Listas de Adyacencia

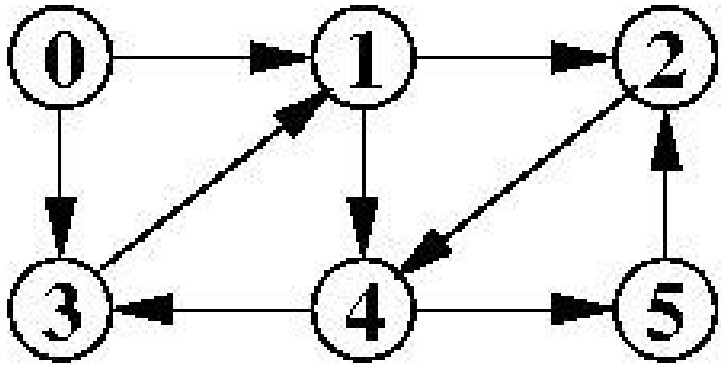
Matriz de Adyacencia

- Matriz de adyacencia es una matriz booleana de orden n, donde n indica el número de vértices de G. Las líneas y columnas de la matriz representan los vértices y el contenido o no de aristas entre ellos
- Cada elemento i,j de la matriz, almacena un 1 ó un 0, dependiendo si existe o no una arista entre los vértices i,j
- Para la generación de la matriz, se da un orden arbitrario a los vértices y se asignan a las filas y columnas en el mismo orden

Matriz de Adyacencia

- Generación de la matriz

Para la generación de la matriz, se da un orden arbitrario a los vértices y se asignan a las filas y columnas en el mismo orden

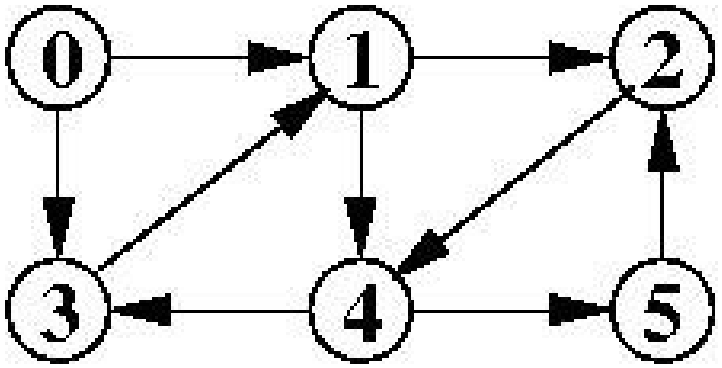


O\D	0	1	2	3	4	5
0	0	1	0	1	0	0
1	0	0	1	0	1	0
2	0	0	0	0	1	0
3	0	1	0	0	0	0
4	0	0	0	1	0	1
5	0	0	1	0	0	0

```
int A[6][6];
//inicializar todo a 0
for(int i=0;i<6;i++)
   for(int j=0;j<6;j++)
      A[i][j] = 0;
//1 si hay enlace origen -> destino
A[0][1] = 1;
A[1][2] = 1;
A[1][4] = 1;
A[2][4] = 1;
A[3][1] = 1;
A[4][3] = 1;
A[4][5] = 1;
A[5][2] = 1;
```

Lista de Adyacencia

Para la generación de la lista, para cada nodo, sólo se guardan sus vecinos.



0 ->	1	3
1 ->	2	4
2 ->	4	
3 ->	1	
4 ->	3	5
5 ->	2	

```
vector<vector<int> > A(6);
//para cada posición origen añadir vecino
A[0].push_back(1);
A[0].push_back(3);
A[1].push_back(2);
A[1].push_back(4);
A[2].push_back(4);
A[3].push_back(1);
A[4].push_back(3);
A[4].push_back(5);
A[5].push_back(2);
```

Guardar los costes del Grafo

Para grafos con costes, en la matriz de adyacencia simplemente se puede guardar el coste de cada arista.

La lista de adyacencia tiene que incluir 2 valores para cada arista: el vecino y el coste, lo que se puede representar con pares de enteros.

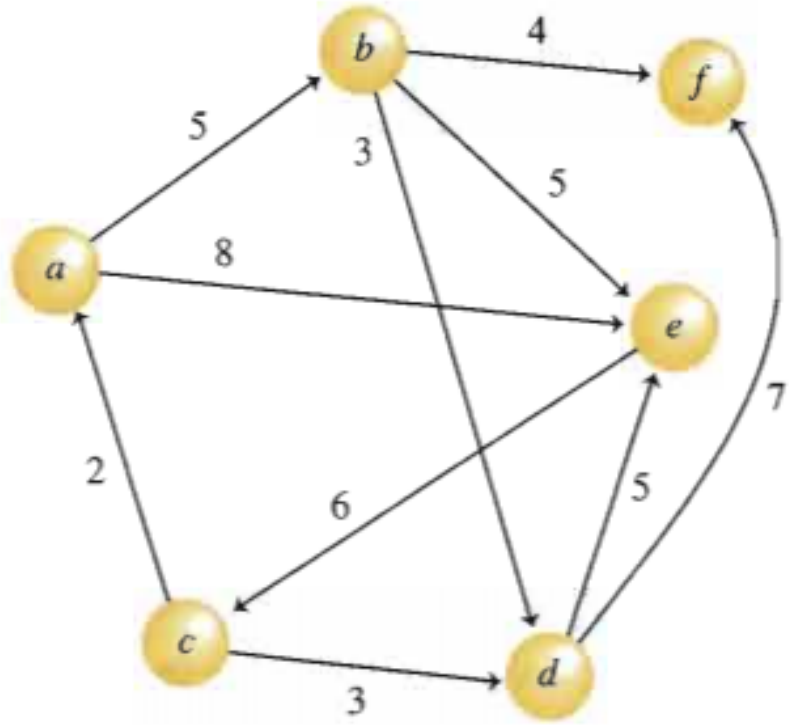
La lista de aristas tiene que incluir 3 valores para cada arista: los dos nodos y el coste. Se puede representar por un struct o combinando pares.

Ventajas y Desventajas

- La ventaja de las matrices de adyacencia es que el tiempo de acceso al elemento requerido es independiente del tamaño de V y de A.
- El tiempo de búsqueda es del orden 0(n). Sin embargo, su principal desventaja es que requiere un espacio de almacenamiento de n^2 posiciones, aunque el numero de arcos de G no sobrepase ese número
- La matriz de adyacencia es útil en los algoritmos diseñados para conocer si existe una arista entre dos vértices dados.

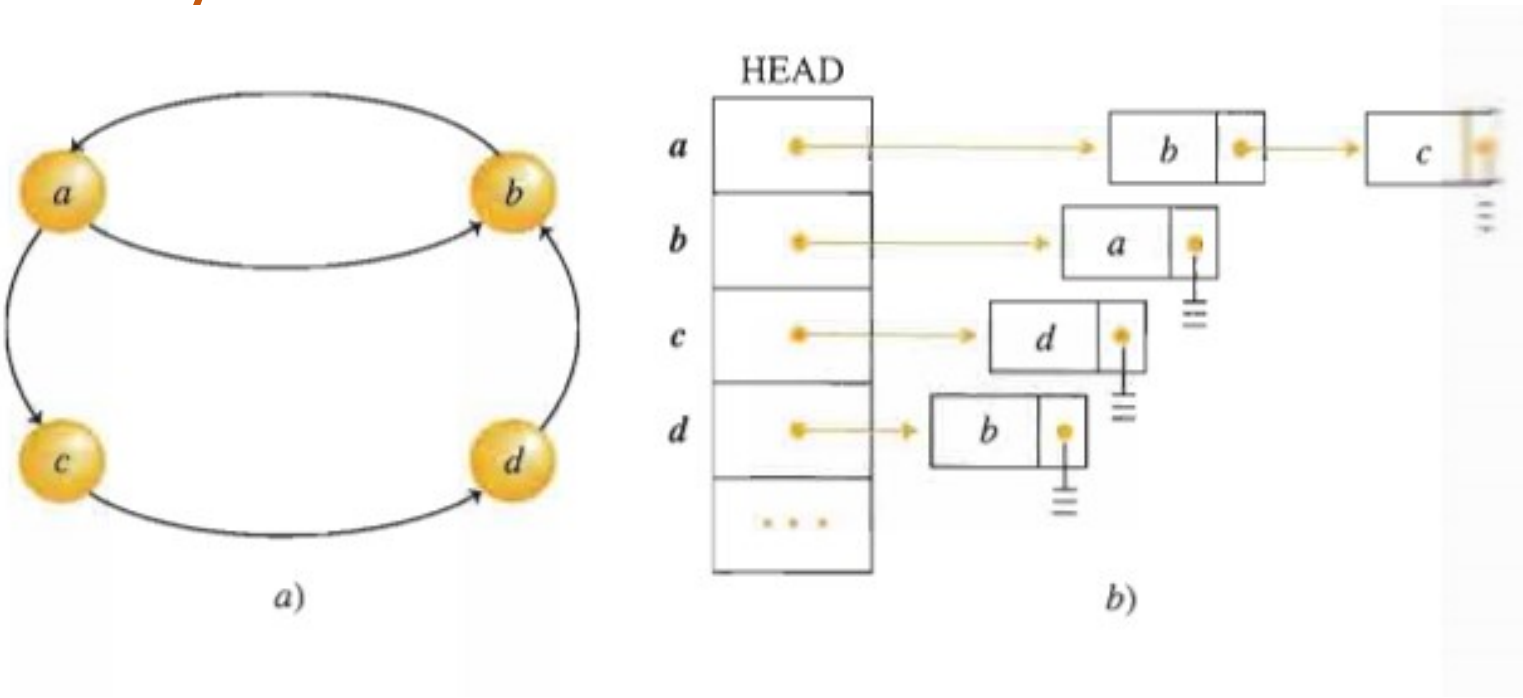
Matriz de Adyacencia Etiquetada

- Se denominan también matrices de costo o de distancias, contienen asociadas a sus aristas los valores de costos o distancias



	a	*b*	*c*	*d*	*e*	*f*
a	0	5	0	0	8	0
b	0	0	0	3	5	4
c	2	0	0	3	0	0
d	0	0	0	0	5	7
e	0	0	6	0	0	0
f	0	0	0	0	0	0

Lista de Adyacencia

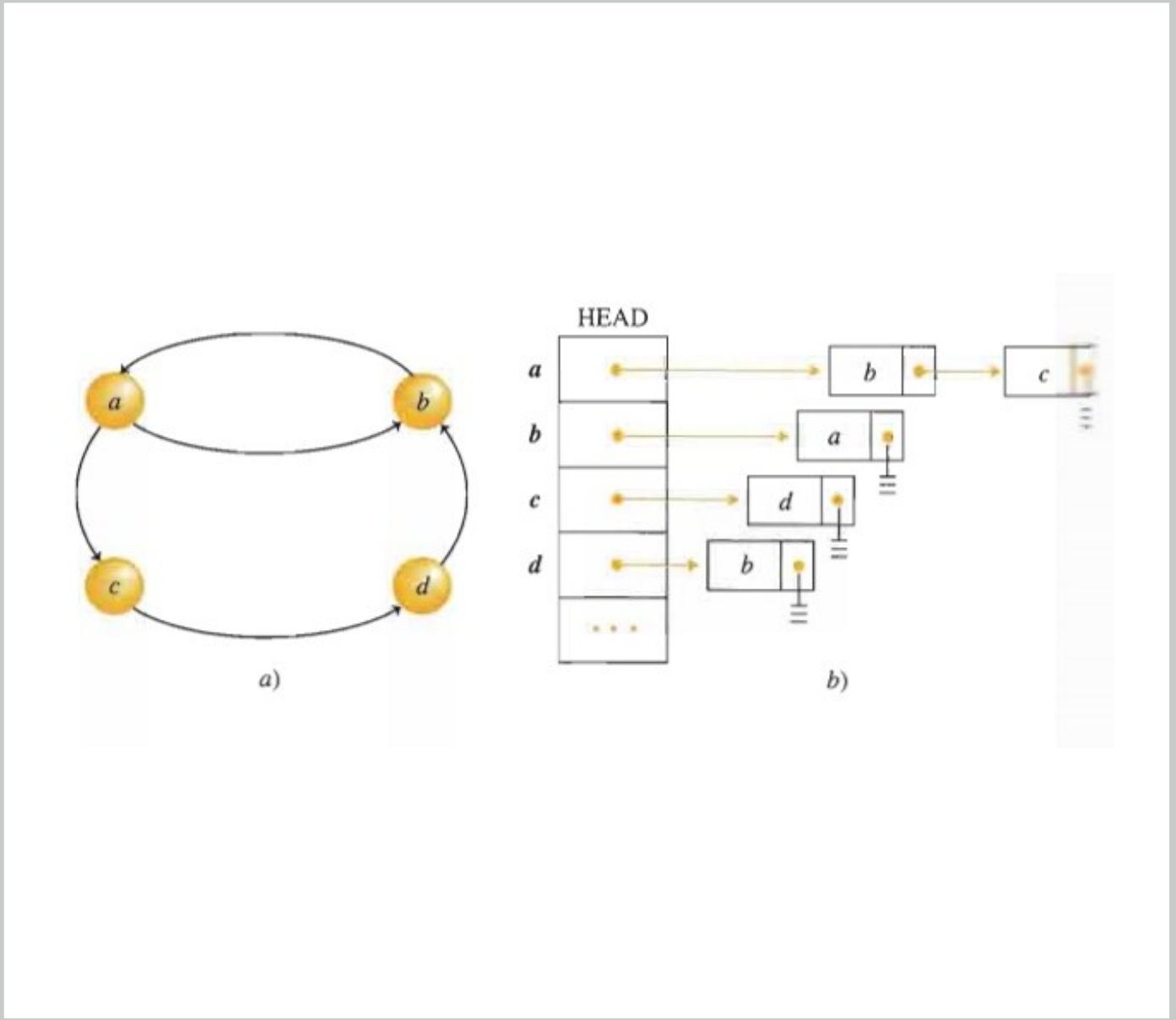


- Una lista de adyacencia para el vértice a es una lista ordenada de todos los vértices adyacentes de a
- Una lista de Adyacencia para representar un Grafo Dirigido, estará formada por tantas listas como vértices tenga G.
- Para guardar los vértices de G se puede usar una lista o un arreglo.
- ***HEAD[i]*** arreglo que es un apuntador a la lista de vértices adyacentes al vértice ***i***
- La lista de adyacencia requiere un espacio proporcional a la suma de números de vértices más el numero de aristas

Lista de Adyacencia Ventaja y desventaja

- Este tipo de representación se recomienda cuando el número de aristas es menor a n^2.
- El uso de la lista de adyacencia permite ahorrar espacio de almacenamiento

- Usar la lista en lugar de una matriz tiene la desventaja de que el tiempo de búsqueda en las aristas puede ser mayor, ya que se pierde el acceso directo que utiliza la matriz



Obtención de caminos dentro de un digrafo

- Al buscar una estructura de datos que se ajuste a las características del problema, se busca también que sobre dicha estructura se puedan realizar operaciones que faciliten el manejo de la información almacenada en ella.
- Para el caso de grafos dirigidos, es de interés encontrar los caminos directos e indirectos entre sus vértices.
- Al trabajar con digrafos etiquetados, se requiere encontrar el camino mas corto entre dos vértices dados o entre todos sus vértices.
- Interesan caminos que permitan llegar desde un vértice origen a un destino recorriendo la menor distancia o con el menor costo.

Algoritmos para Grafos Dirigidos

Los algoritmos más utilizados para este fin son:

- Algoritmo de Dijkstra
- Algoritmo de Floy
- Algoritmo de Warshall

Algoritmo de Dijkstra

- Encuentra el camino mas corto de un vértice elegido a cualquier otro vértice del dígrafo, donde la longitud del camino es la suma de los pesos de las aristas que lo forman (las aristas no deben tener un peso negativo)
- Principales elementos considerados cuando se aplica el algoritmo.
  - **S** es un arreglo formado por los vértices de los cuales ya se conoce la distancia minima entre ellos y el origen. Inicialmente se almacena solo el vértice origen
  - **D** es un arreglo formada por la distancia del vértice origen a cada uno de los otros.
  - **D[i]** almacena la menor distancia o costo entre el origen y el vértice i, este camino se le conoce como Especial
  - Este arreglo se forma en cada paso del algoritmo, al termino, D tiene la distancia mínima entre el origen y cada uno de los vértices del grafo
  - M es una matriz de distancias n x n elementos, tal que M[i,j] almacena la distancia o costo entre los vértices i, j, si entre ambos existe una arista, de lo contrario M[i,j] será un valor muy grande (infinito)

Algoritmo de Dijkstra

- Este algoritmo encuentra la distancia mínima entre un vértice origen y cada uno de los otros vértices de una grafica dirigida. Se considera al vértice 1 como el origen. N es el numero de vértices de la grafica dirigida. S y D son arreglos de N elementos y M es una matriz de n x n elementos

1. Agregar el vértice 1 a S
2. Repetir de 2 hasta N

   Elegir un vértice v en (V - S) tal que D[v] sea el mínimo valor

   Agregar v a S

   2.1 Repetir para cada vértice w en (V - S)

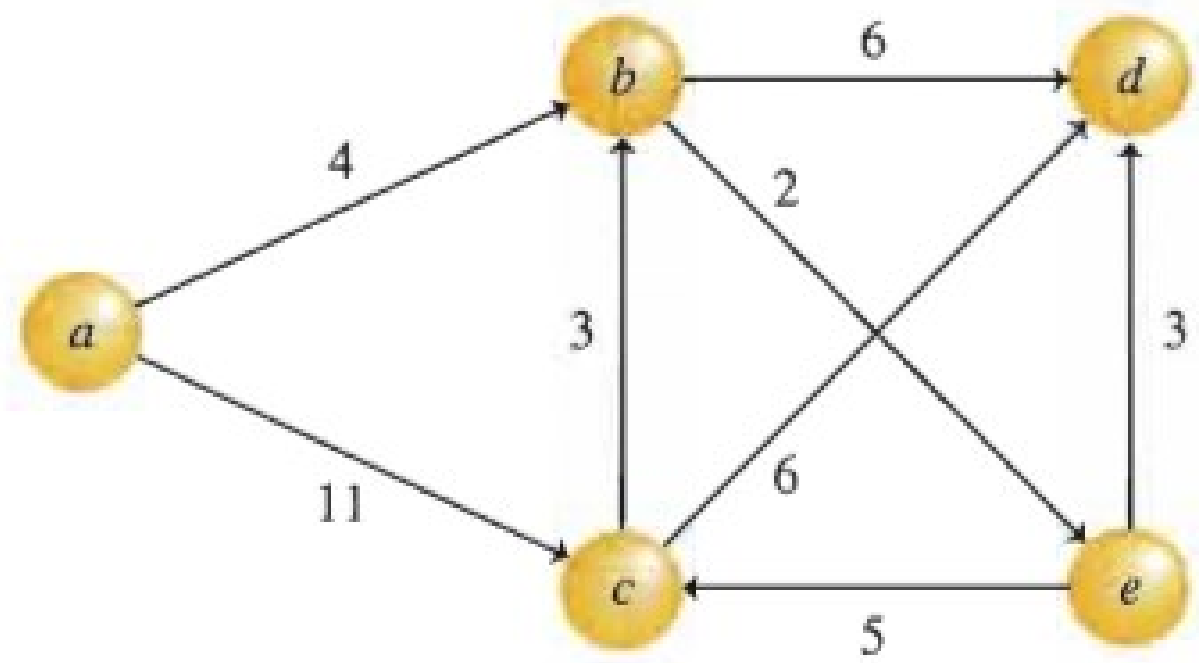
      Hacer D[w] ← minimo(D[w], D[v]+ M[v,w])

   2.2 (fin del ciclo del paso 2.1)

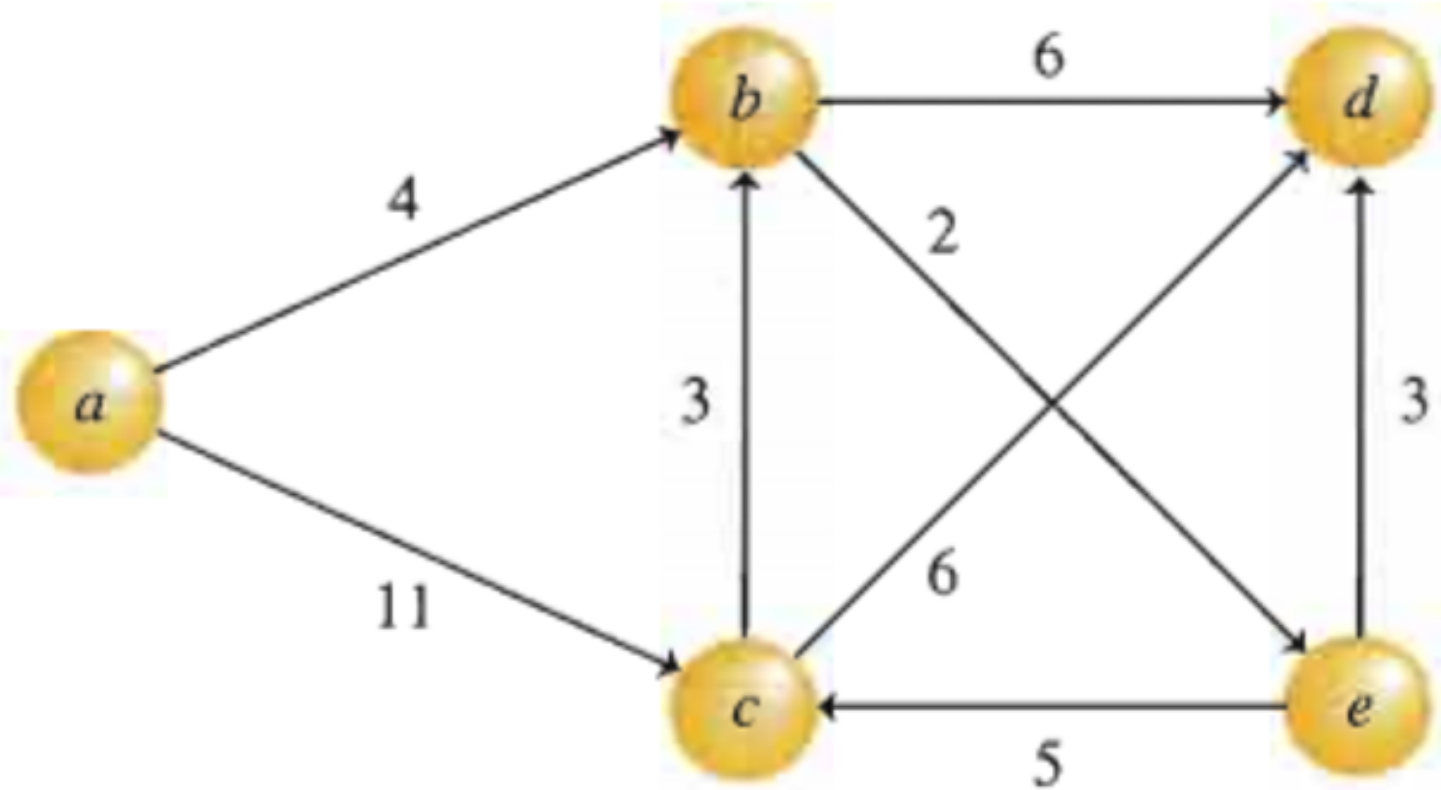
3. (Fin del ciclo del paso 2)

Ejemplo 1

Aplicando el algoritmo de Dijkstra, obtener el camino mas corto desde uno de los vértices a cualquiera de los otros vértices de un grafo dirigido, formado por 5 valores



	paso 1	paso 2	paso 3	paso 4	paso 5
a	(0, -)	*	*	*	*
b	(4,a)	(4,a)	*	*	*
c	(11,a)	(11,a)	(11,e)	(11,e)	(11,e)
d	∞	(10,b)	(9,e)	(9,e)	*
e	∞	(6,b)	(6,b)	*	*



Algoritmo de Floyd

- Encuentra el camino mas corto entre todos los vértices del dígrafo.
- Sea el grafo dirigido G=(V,A), donde cada arista u --> v tiene asociado un peso. El algoritmo de Floyd permite encontrar el camino mas corto entre cada par ordenado u y v
- La matriz de distancia sirve como punto de partida de este algoritmo
- Se realizan k iteraciones sobre la matriz buscando el camino mas corto, por tanto en la k-esima iteración

Algoritmo de Floyd

- M[i , j] tendrá el camino del menor costo para llegar de i a j, pasando por un numero de vértices menor a k, el cual se calcula:

$$M_k[i,j] \text{ min= } \{ M_{k-1}[i,j] \text{ ó } M_{k-1}[i,k] + M_{k-1}[k,j] \}$$

Se obtendrá el camino mas corto entre el valor obtenido en la iteración (k-1) y el que resulta de pasar por el vértice k. En el algoritmo se usa la matriz de costos M donde M[i,j] será igual al costo de ir de i a j, Si no existe camino de i a j, asume un valor muy grande o 0 si i=j

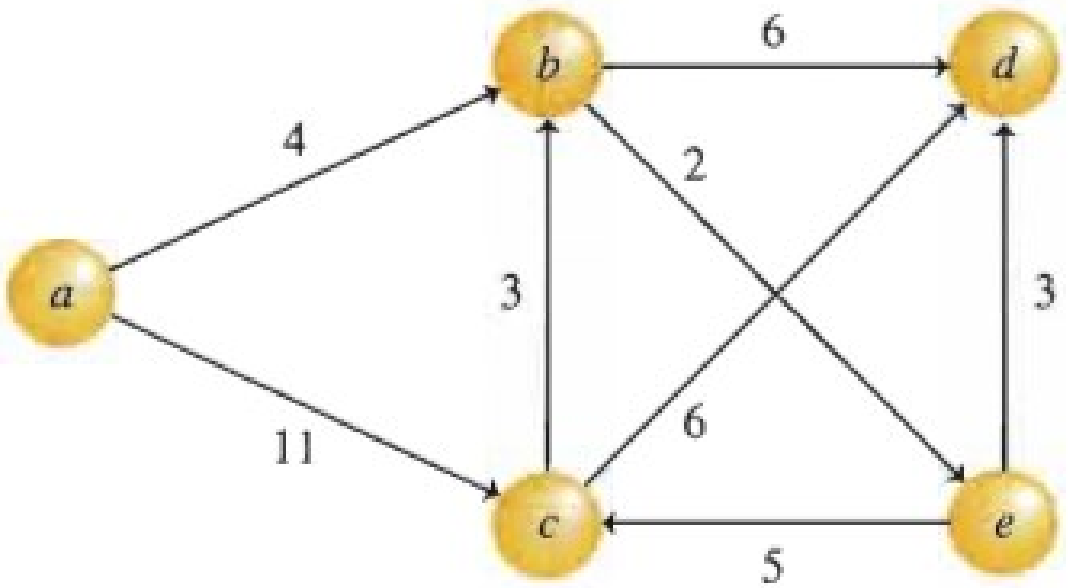
Algoritmo de Floyd

- Este algoritmo encuentra la distancia mínima entre todos los vértices del grafo dirigido. M es una matriz de N x N y se inicia con los costos del dígrafo. k, i, j son variables enteras

1. *Repetir* con K desde 1 hasta N
   1.1 *Repetir* con I desde 1 hasta N
      1.1.1 *Repetir* con J desde 1 hasta N
         1.1.1.1 *Si* $(M_{IK} + M_{KJ} < M_{IJ})$ *entonces*
            Hacer $M_{IJ} \leftarrow M_{IK} + M_{KJ}$
         1.1.1.2 {Fin del condicional del paso 1.1.1.1}
      1.1.2 {Fin del ciclo del paso 1.1.1}
   1.2 {Fin del ciclo del paso 1.1}
2. {Fin del ciclo del paso 1}

Ejemplo de Aplicación

- Encontrar la minima distancia entre todos los vértices de un grafo dirigido

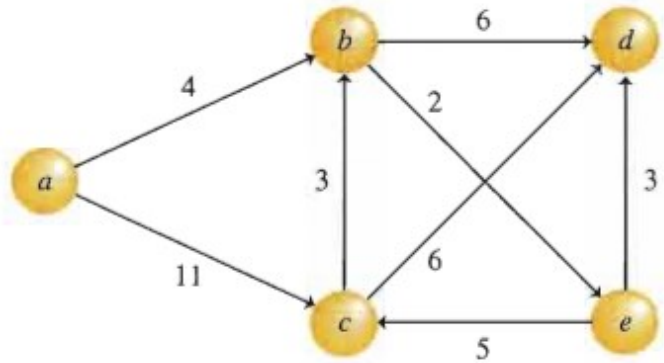


	a	b	c	d	e
a	0	4	11	∞	∞
b	∞	0	∞	6	2
c	∞	3	0	6	∞
d	∞	∞	∞	0	∞
e	∞	∞	5	3	0

Matriz de distancias del dígrafo
(estado inicial)

	a	b	c	d	e
a	0	4	11	10	6
b	∞	0	∞	6	2
c	∞	3	0	6	5
d	∞	∞	∞	0	∞
e	∞	∞	5	3	0

Uso del vértice b como vértice intermedio
K=2, caminos: a b d = 10, a b e = 6, c b e = 5



	a	b	c	d	e
a	0	4	11	10	6
b	∞	0	∞	6	2
c	∞	3	0	6	5
d	∞	∞	∞	0	∞
e	∞	8	5	3	0

Uso del vértice c como vértice intermedio
K=3, caminos: e c b = 8

	a	b	c	d	e
a	0	4	11	9	6
b	∞	0	7	5	2
c	∞	3	0	6	5
d	∞	∞	∞	0	∞
e	∞	8	5	3	0

Uso del vértice e como vértice intermedio
K=5, caminos: a e d= 9, b e c=7, b e d =5

Algoritmo de Warshall

- Encuentra, si es posible, un camino entre cada uno de los vértices del grafo dirigido.
- La solución mostrada por el algoritmo, no presenta la distancia entre los vértices, sólo muestra si hay o no camino entre ellos.
- El algoritmo de Warshall se basa en un concepto llamado **cerradura transitiva de la matriz de adyacencia**

Algoritmo de Warshall

- Sea el grafo dirigido G(V,A) y su matriz de adyacencia M, donde M[i,j]=1 si hay una arista de i a j y 0 si no lo hay.
- La cerradura transitiva de M es la matriz es C, tal que C[i,j]=1 si hay un camino de longitud mayor o igual a 1 de i a j, o 0 en otro caso.
- Para generar la matriz C se establece que existe un camino de vértice i al j que no pasa por un numero de vértices mayor que k, si:

a. Si ya existe un camino de i a j que no pasa por un numero de vértices mayor que -1

b. Hay un camino de i a k que no pasa por un numero de vértices mayor que k-1,

   Hay un camino de k a j que no pasa por un numero de vértices mayor que k-1

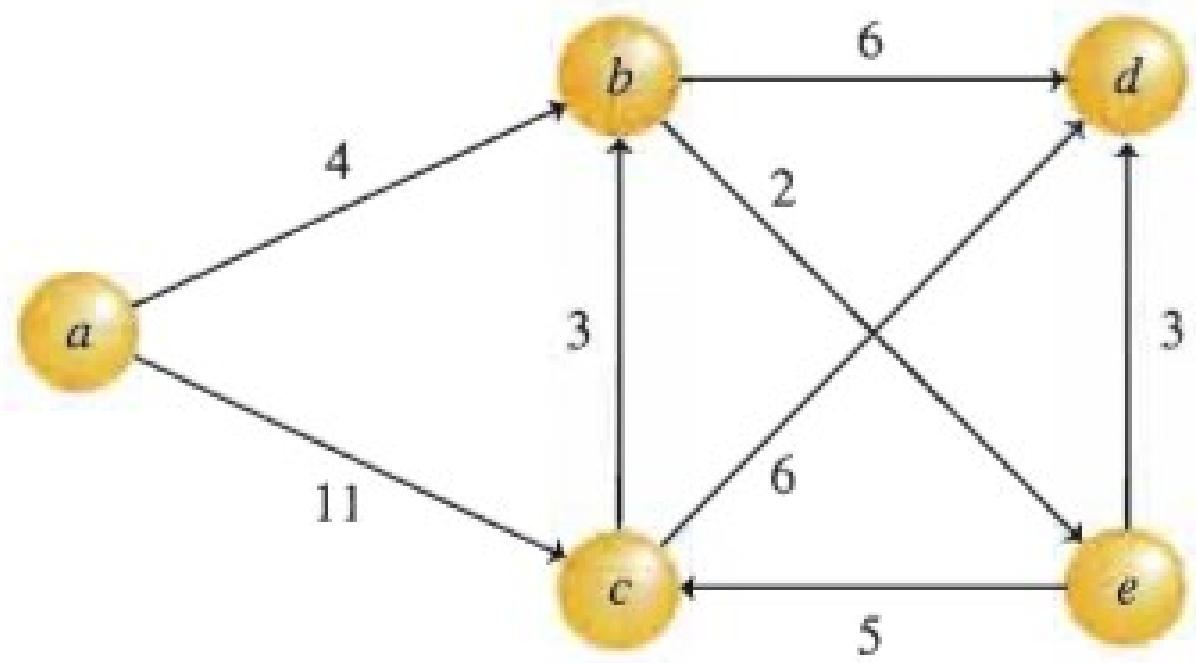
Algoritmo de Warshall

Este algoritmo encuentra, si es posible, un camino de longitud mayor o igual a 1 entre cada uno de los vértices del grafo dirigido. N es el numero de vértices del dígrafo. C es una matriz de N X N elementos. Inicialmente es igual a M. Al terminar el algoritmo contendrá la cerradura transitiva de M. K, I, J son variables enteras

1. *Repetir* con K desde 1 hasta N
   1.1 *Repetir* con I desde 1 hasta N
      1.1.1 *Repetir* con J desde 1 hasta N
         1.1.1.1 *Si* $(A[I, J] = 0)$ *entonces*
            $A[I, J] \leftarrow A[I, K]$ y $A[K, J]$
         1.1.1.2 {Fin del condicional del paso 1.1.1.1}
      1.1.2 {Fin del ciclo del paso 1.1.1}
   1.2 {Fin del ciclo del paso 1.1}
2. {Fin del ciclo del paso 1}

Ejemplo

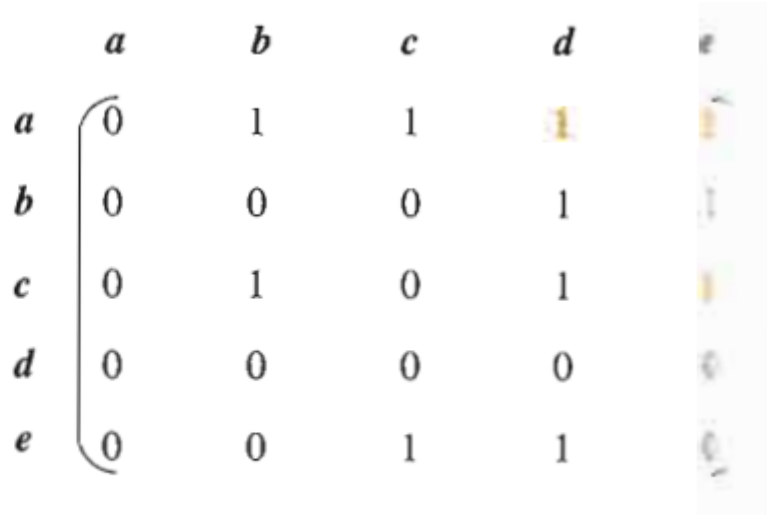
- Se presenta un ejemplo de aplicación para el algoritmo de Warshall para determinar si existe o no un camino entre todos los vértices de un grafo dirigido



Ejemplo

	a	b	c	d	e
a	0	1	1	0	0
b	0	0	0	1	1
c	0	1	0	1	0
d	0	0	0	0	0
e	0	0	1	1	0

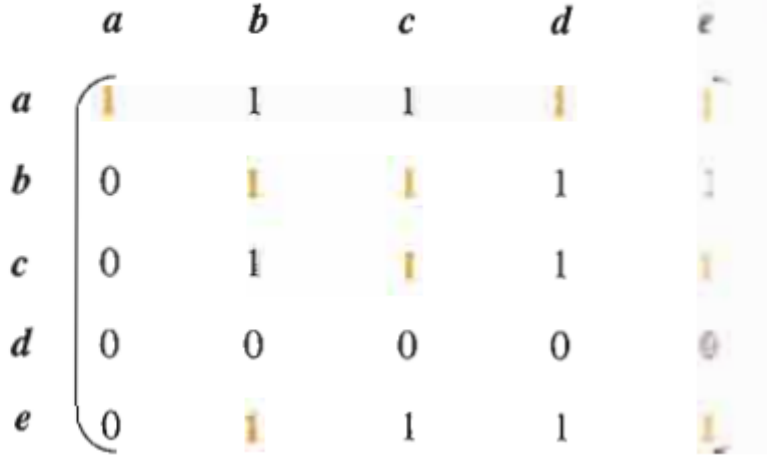
- Estado inicial de la matriz



Resultado de usar el vértice b como intermedio, caminos encontrados: a,b,d, a,b,e, y c,b,e

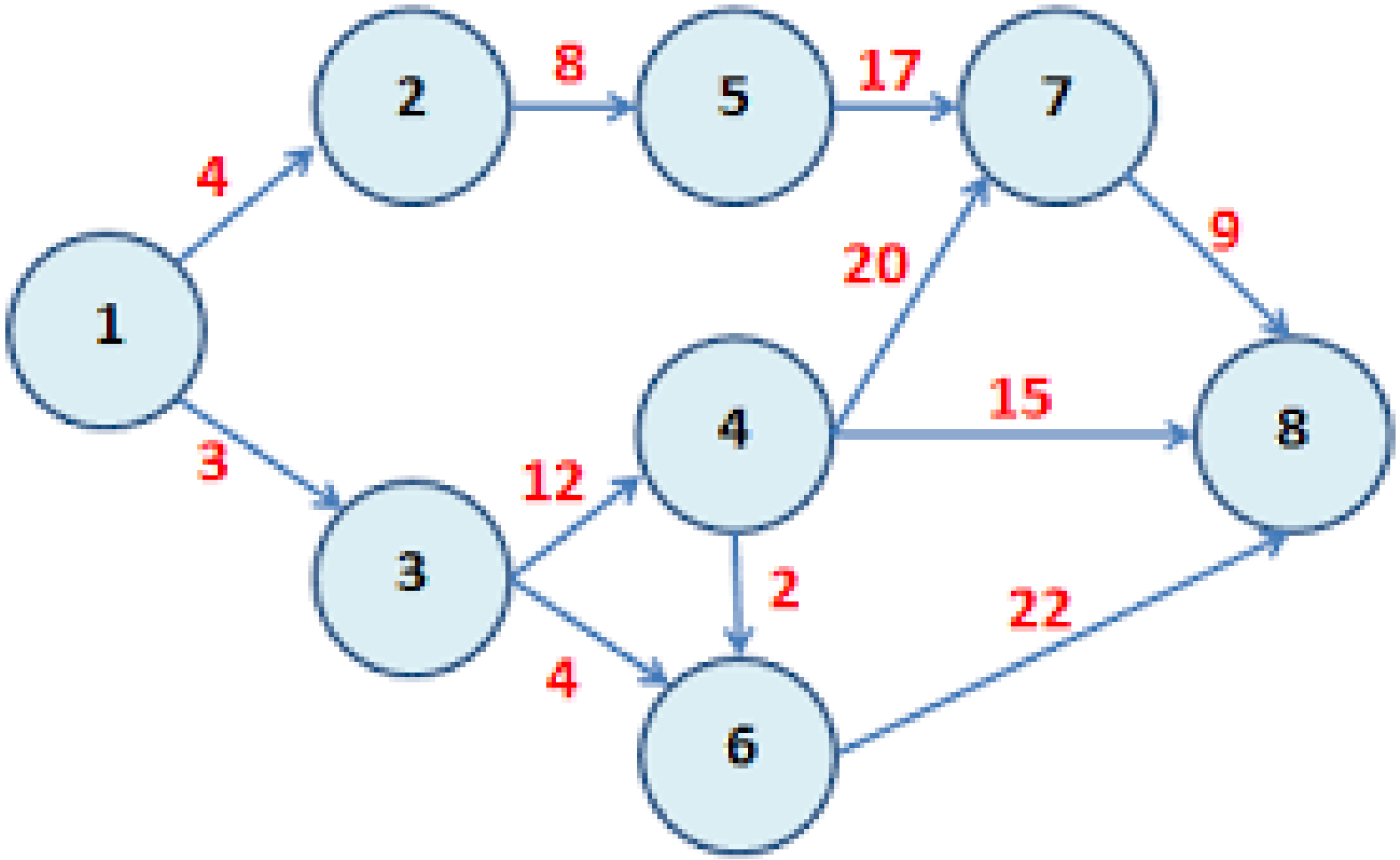
	a	b	c	d	e
a	0	1	1	1	1
b	0	0	0	1	1
c	0	1	0	1	1
d	0	0	0	0	0
e	0	1	1	1	1

Resultado de usar el vértice c como intermedio, caminos encontrados: e,c,b y e,c,e

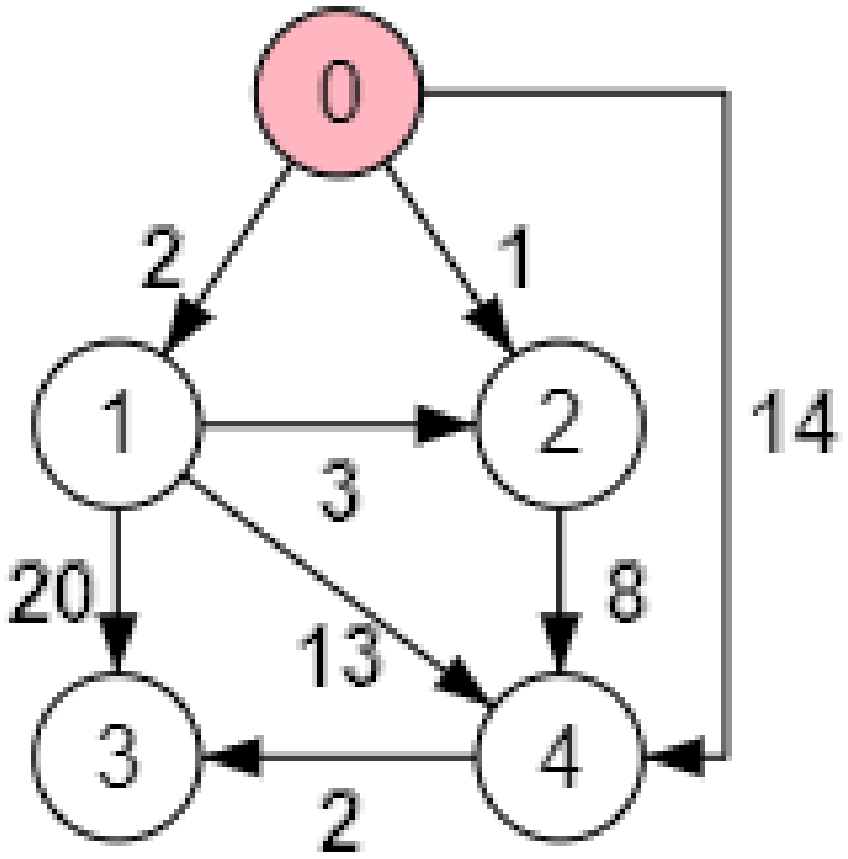


Resultado de usar el vértice e como intermedio, caminos encontrados: a,e,a,b,e,b,b,e,c y c,e,c

EJERCICIO: Aplicar el algoritmo de Dijkstra para obtener el camino mínimo entre el vertice 1 y el vertice 8



EJERCICIO: Aplicar el algoritmo de Floy Warshall para obtener la matriz de caminos mínimos



EJERCICIO: Obtener el camino mínimo entre SJL y La UNMSM en base al grafo dirigido