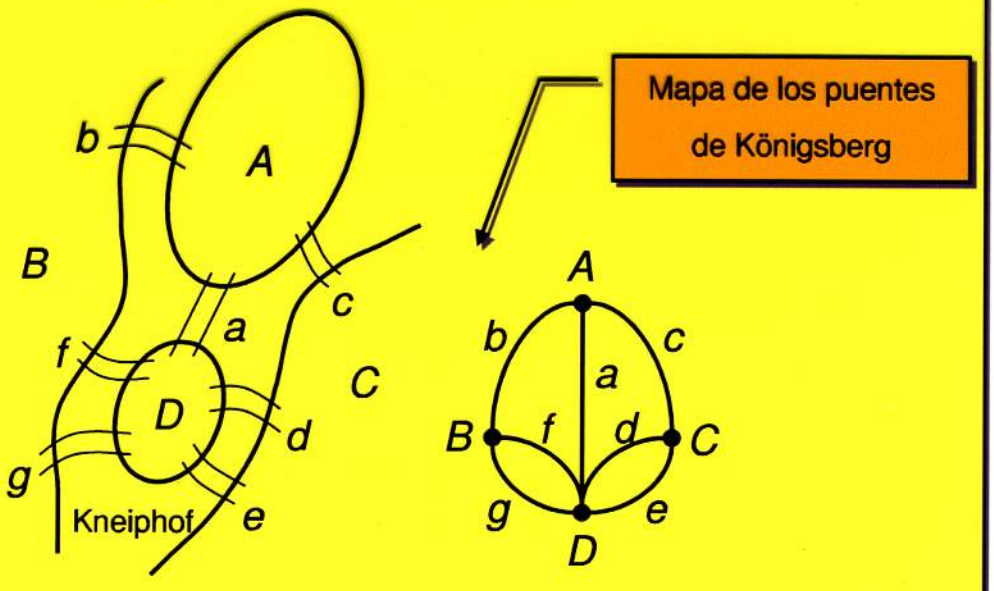
Grafos

1. Conceptos sobre grafos
2. Representaciones de grafos
   - *Matriz de adyacencias*
   - *Listas de adyacencias*
3. Recorrido en grafos
   - *Búsqueda en profundidad*
   - *Búsqueda en anchura*
4. Aplicaciones
   - *Ordenación topológica*
   - *Camino de coste mínimo*
   - *Árbol de extensión de coste mínimo*

En los problemas originados en ciencias de la computación, matemáticas, ingeniería y muchas otras disciplinas, a menudo es necesario representar relaciones arbitrarias entre objetos de datos. Los grafos dirigidos y los no dirigidos son modelos naturales de tales relaciones.

El lenguaje de grafos permite permite abstraerse de una situación dada por relaciones arbitrarias entre objetos, mediante el dibujo de un gráfico en el que los puntos (*vértices)* representan objetos individuales, posiciones, componentes químicos, etc., conectados por líneas (*ejes*) o flechas (*arcos*) que simbolizan las relaciones.



Algunas aplicaciones de los grafos

- Representación del conocimiento
- Problemas de búsqueda
- Análisis y verificación de programas
- Redes eléctricas
- Redes de ordenadores y comunicaciones
- Redes de distribución
- Planificación de vuelos
- Planificación de exámenes
- Encontrar secuencias óptimas de procesos

Conceptos sobre grafos

Grafo

Un grafo es un par $G=(V, A)$ formado por dos conjuntos, V (de vértices o nodos) y A (de arcos o ejes).

Cada arco es un par (v, w) con $u,v \in V$.

El grafo es simple si sólo hay un arco (v,w).

Los vértices y los arcos pueden estar etiquetados.

Grafo Dirigido

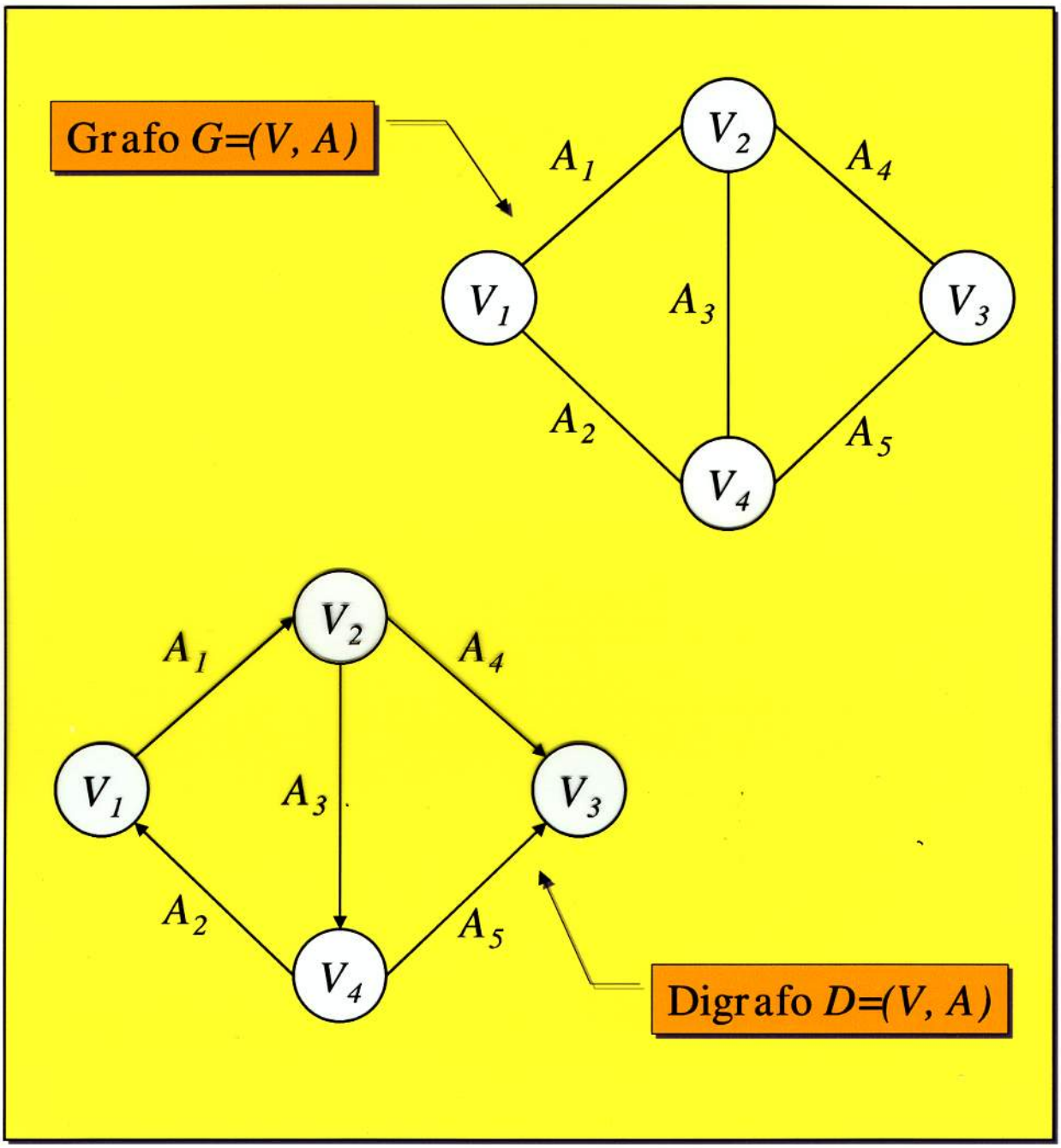
Si el par (v, w) es ordenado, el grafo es un grafo dirigido o digrafo.

Se dice que el arco $(v,w) \in A$ va de v a w y que w es adyacente a v.

Grafo No Dirigido

Si el par (v, w) no es ordenado, el grafo es un grafo no dirigido o simplemente grafo.

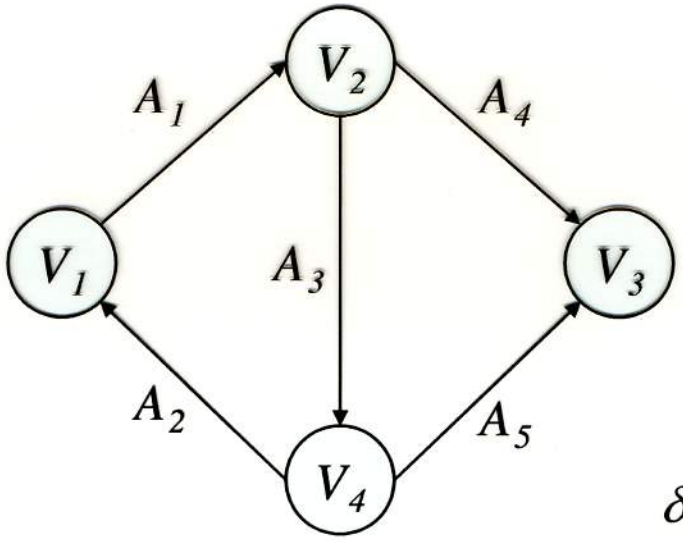
En este caso $(v, w) = (w, v)$ y se dice que v y w son adyacentes entre sí.


Grafo G=(V, A)
V1
V2
V3
V4
A1
A2
A3
A4
A5
Digrafo D=(V, A)

Grado de un vértice

Sea $G=(V, A)$ un grafo y $v \in V$ un vértice, se define el grado de v, $\delta(v)$, cómo el número de arcos incidentes en él; es decir, el número de arcos de los que v es extremo.

En digrafos se definen el grado de entrada $\delta_e(v)$ y el grado de salida $\delta_s(v)$.



$\delta_e(v_4)=1$

$\delta_s(v_4)=2$

Camino de un grafo

Sea G un grafo (o digrafo). Un camino en G es una sucesión de vértices y arcos

$$v_0\, a_1\, v_1\, a_2\, v_2 \ldots\, a_n v_n$$

tal que los extremos (origen y destino si es un digrafo) del arco a_i son v_{i-1} y v_i $(0 \leq i \leq n)$.

Al número de arcos del camino se le denomina longitud (orden) de éste; v_0 y v_n son los extremos (origen y destino para un digrafo) del camino y los vértices v_1, v_2, …, v_{n-1} se denominan vértices interiores.

- Un camino es cerrado si $v_0 \equiv v_n$
- Un camino es simple si sus arcos son distintos dos a dos
- Un camino es elemental si sus vértices son distintos dos a dos exceptuando, a lo sumo, sus extremos.
- Un camino de longitud cero es el que va de un vértice a sí mismo.

Ciclo de un grafo

Es un camino cerrado, elemental y simple.

Vértices conectados

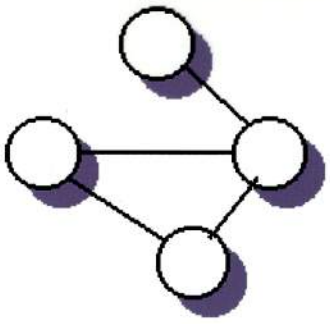
Dos vértices u y v de un grafo G se dice que están conectados cuando existe un camino en G de extremos u y v.

Grafo conexo (conectado)

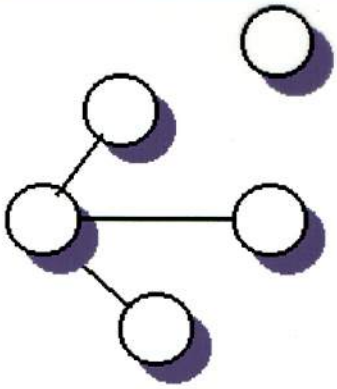
Un grafo G es conexo si hay un camino entre cada par de vértices del mismo.

Grafo completo

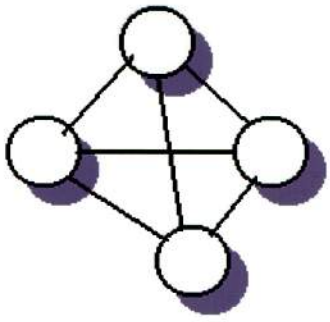
Un grafo G es completo si es conexo y además cada par de vértices están unidos por un arco.



conexo



inconexo



completo

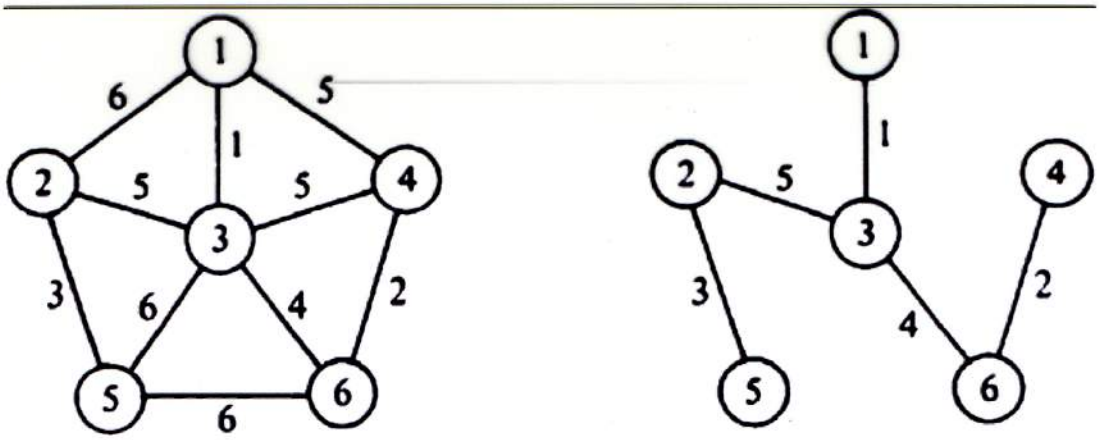
Árbol de expansión (árbol generador)

Un árbol de expansión (*spanning tree*) de un grafo G es un árbol que conecta todos los vértices de V.

Coste de un árbol de expansión

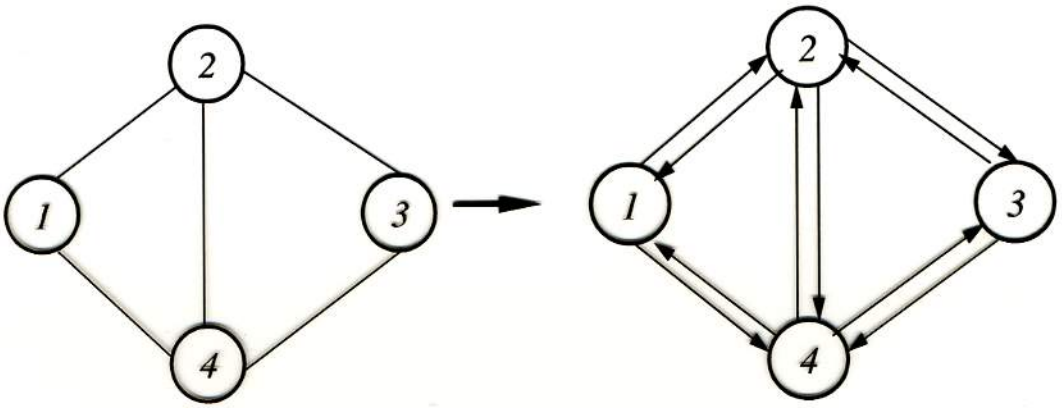
El coste de un árbol de expansión es la suma de los costes de los arcos del árbol.

Un problema típico consiste en la búsqueda del árbol de expansión de mínimo coste.



Representaciones de grafos

Desde el punto de vista de la representación no se establecerá ninguna diferencia entre grafos no dirigidos y grafos dirigidos dada la equivalencia que existe entre ambos:

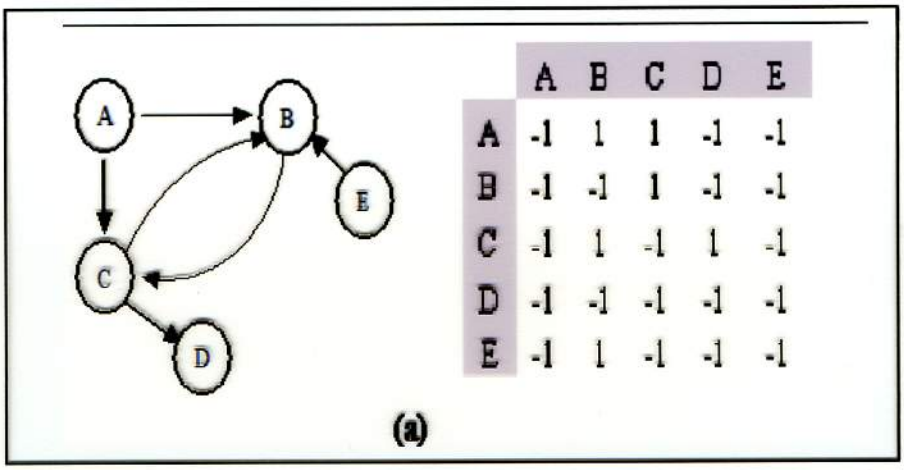


Las dos representaciones de grafos más extendidas son las que se basan en:

- *Matriz de adyacencias*
- *Lista de adyacencias*

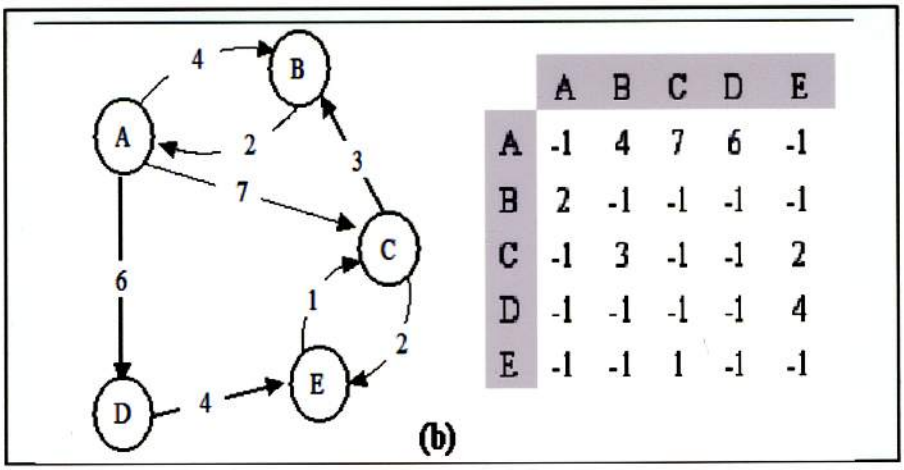
Matriz de adyacencias

Dado un grafo G con *n* vértices en V, la matriz de adyacencia para G es una matriz de dimensión *nxn* donde a[i,j] tiene un valor que indica si existe o no el arco (i,j).



	A	B	C	D	E
A	-1	1	1	-1	-1
B	-1	-1	1	-1	-1
C	-1	1	-1	1	-1
D	-1	-1	-1	-1	-1
E	-1	1	-1	-1	-1

(a)



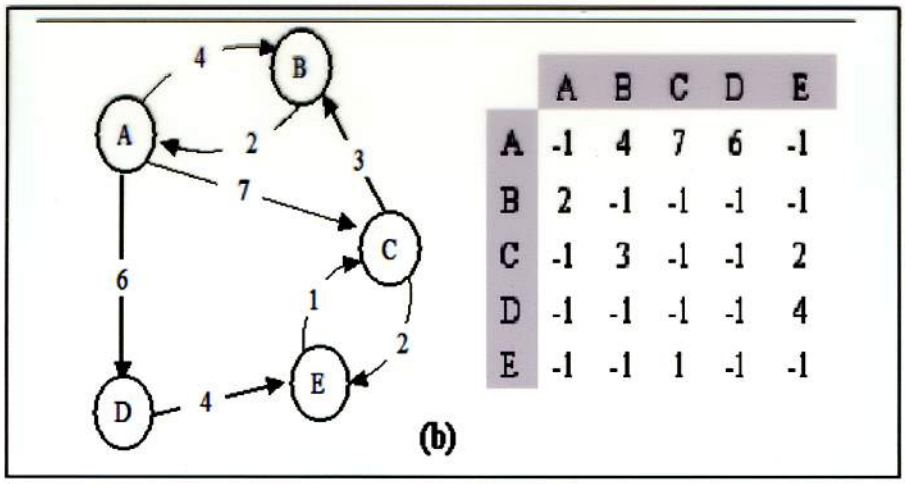
	A	B	C	D	E
A	-1	4	7	6	-1
B	2	-1	-1	-1	-1
C	-1	3	-1	-1	2
D	-1	-1	-1	-1	4
E	-1	-1	1	-1	-1

(b)

Esta representación es útil en algoritmos para grafos en los que se necesite conocer si un arco dado está presente, operación que se realizaría en tiempo constante.

Por contra el principal inconveniente es que requiere un espacio de $O(n^2)$ aun si el grafo tiene menos de n^2 arcos.

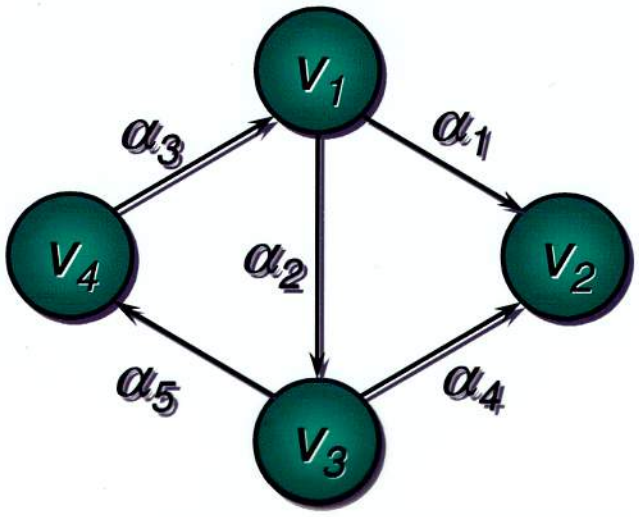
En grafos poco densos (con pocos arcos) aumenta la complejidad de ciertas operaciones en las que únicamente se requiere examinar los arcos del grafo.



	A	B	C	D	E
A	-1	4	7	6	-1
B	2	-1	-1	-1	-1
C	-1	3	-1	-1	2
D	-1	-1	-1	-1	4
E	-1	-1	1	-1	-1

(b)

Matriz de adyacencias



	v_1	v_2	v_3	v_4
v_1		α_1	α_2	
v_2				
v_3		α_4		α_5
v_4	α_3			

Requiere representar un valor característico: el **arco nulo**

Requiere numerar los vértices

```
// área de datos
matrix<A> data;
```

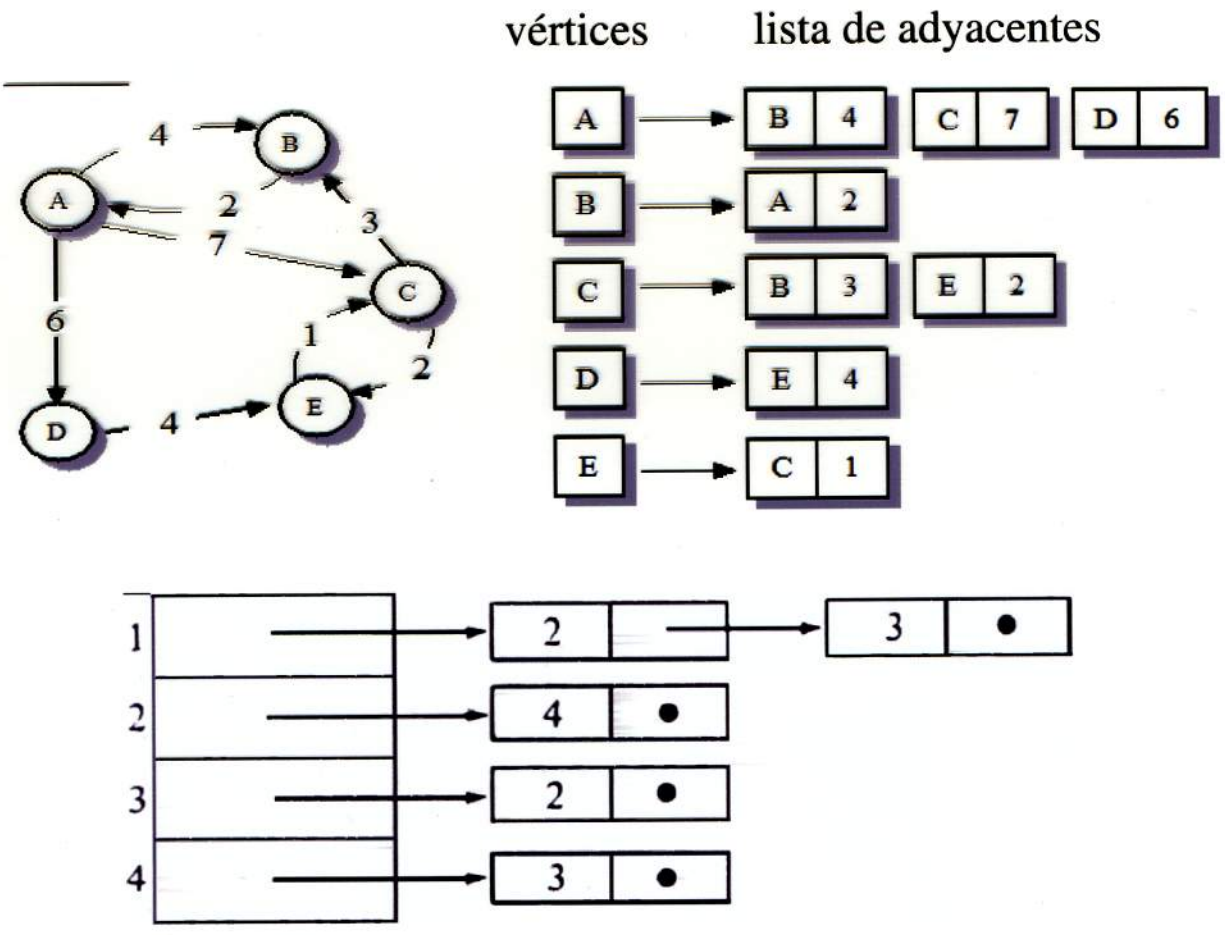
No se utiliza por dar lugar a operaciones ineficientes

```
// área de datos
dictList<V, dictList<V, A> > data;
```

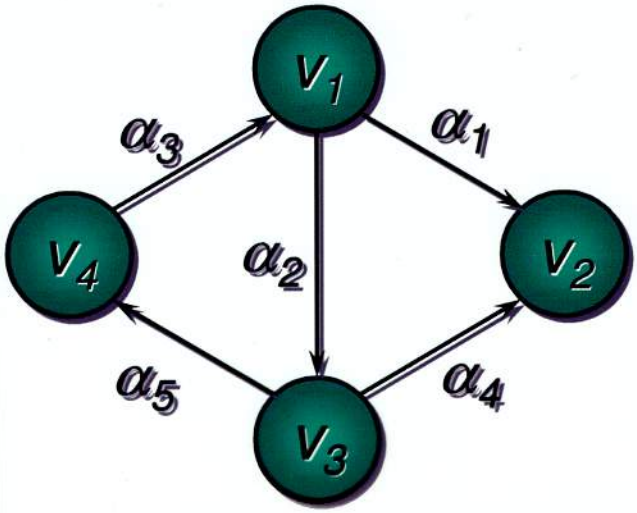
Lista de adyacencias

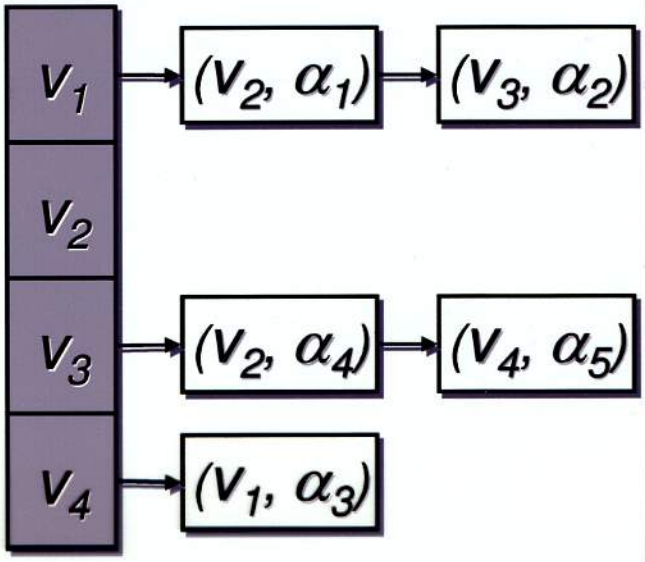
Dado un grafo G con *n* vértices en V, la lista de adyacencia para un vértice i es una lista, en algún orden, de todos los vértices adyacentes a i.

Cada vértice de la lista de adyacencias puede llevar asociado el peso del arco correspondiente.



Lista de adyacencias





Requiere numerar los vértices

```
// área de datos
vector<dictList<V, A> > data;
vector<LList<V> > data;          //sin pesos
```

No requiere numerar los vértices

```
// área de datos
dictList<V, dictList<V, A> > data;
```

Recorrido en grafos

Hay un gran número de casos en que la solución a un problema en el que es necesario utilizar grafos pasa por recorrer todos los vértices y arcos del mismo de forma sistemática. Para ello se utilizan dos formas de búsqueda:

- La búsqueda en profundidad

  Generalización del recorrido en preorden en un árbol

- La búsqueda en anchura

  Generalización del recorrido por niveles en un árbol

El recorrido del grafo se realiza garantizando que se visitan todos sus vértices mediante una de las técnicas de búsqueda indicadas.

Búsqueda en profundidad en un grafo

```
acción BPF (v:Vértice)
var w:Vértice;
inicio
  marcar(v);
  para cada vértice w adyacente a v hacer
    si ¬ marcado(w)
    entonces
      predecesor(w, v);
      BPF(w)
    fsi
  fpara
facción
```

Recorrido en profundidad de un grafo

```
acción REP (g:Grafo)
inicio
  para cada vértice v de g hacer
    desmarcar(v);
    predecesor(v, NULO)
  fpara;
  para cada vértice v de g hacer
    si ¬ marcado(v)
    entonces
      BPF(v)
    fsi
  fpara
facción
```

Búsqueda en anchura en un grafo

```
acción BEA (v:Vértice)
var v, w, u:Vértice;
    pendientes:ColaNodos;
inicio
añadir(q,v); marcar(v);
 mientras ¬ vacía(q) hacer
   u ← borrar(q);
   para cada vértice w adyacente a u hacer
    si ¬ marcado(w)
    entonces
      marcar(w);
      predecesor(w, u)
      añadir(q, w)
    fsi
   fpara
 fmientras
facción
```

```
acción REA (g:Grafo)
inicio
 para cada vértice v de g hacer
   desmarcar(v);
   predecesor(v, NULO)
 fpara;
 para cada vértice v de g hacer
   si ¬ marcado(v)
   entonces
     BEA(v)
   fsi
 fpara
facción
```

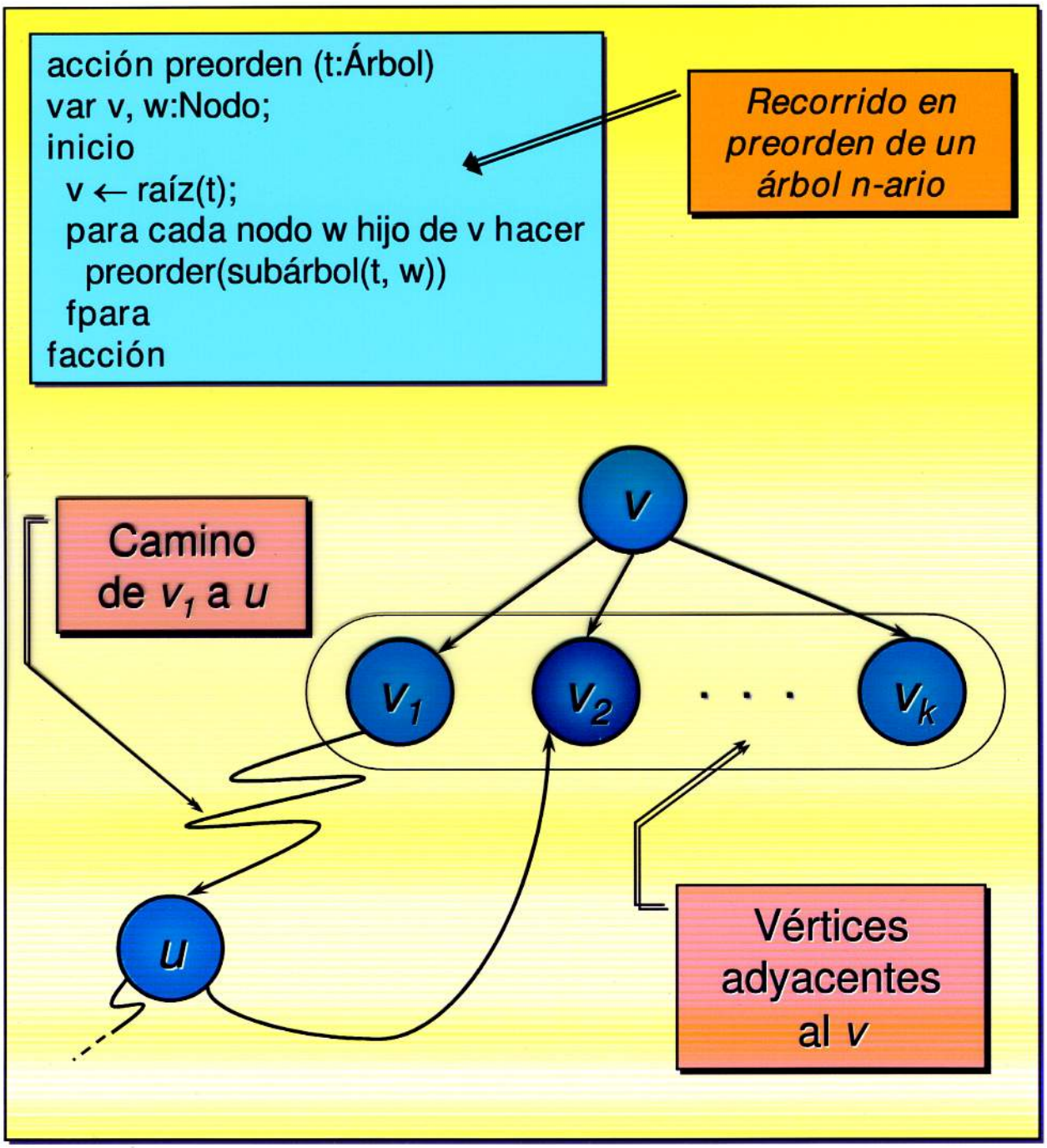
Recorrido en anchura de un grafo

La complejidad del algoritmo de recorrido en profundidad del grafo es el siguiente:

- Si el grafo se representa mediante listas de adyacencias se recorrerían todas las listas una vez en el peor de los casos, $O(|V|+|A|)$.
- Si el grafo se representa mediante la matriz de adyacencias, deben buscarse los adyacentes para cada vértice , $O(|V|^2)$.

Para que la complejidad sea la indicada la operación *marcado(v)* debe de ser de tiempo constante.

Búsqueda en profundidad



Búsqueda en anchura

```
acción niveles (t:Árbol)
var v, w, u:Nodo;
    pendientes:ColaNodos;
inicio
 v ← raíz(t);
 añadir(pendientes, v);
 mientras ¬ vacía(pendientes) hacer
  u ← borrar(q);
  para cada nodo w hijo de u hacer
   añadir(q, w)
  fpara
 fmientras
facción
```

Recorrido por niveles de un árbol n-ario

El algoritmo de búsqueda en anchura de un grafo es la generalización del algoritmo previo pero, al igual que en el caso de la búsqueda en profundidad, marcando los vértices que se visitan.