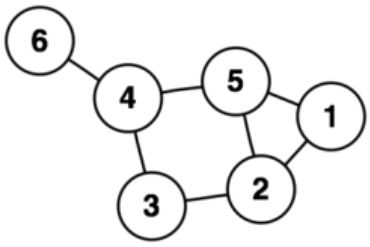
Grafos

Muchos problemas se representan en términos de objetos y como estos se conectan, por ejemplo el mapa de las rutas de sud América, circuitos eléctricos, redes de comunicaciones, etc.

Un grafo consiste en un conjunto de vértices o nodos V y un conjunto de arcos o Aristas A G={V,A}

Un vértice puede tener un nombre y otras propiedades y una arista es la conexión de dos vértices



V={1,2,3,4,5,6}

A={{1,2},{1,5},{2,3},{2,5},{3,4},{4,5},{4,6}}

Un grafo no dirigido es un grafo cuyas aristas están formados por pares de nodos no ordenados o no direccionados u –v

Un grafo dirigido es un grafo cuyas aristas están formados por pares de nodos ordenados o direccionados u →v

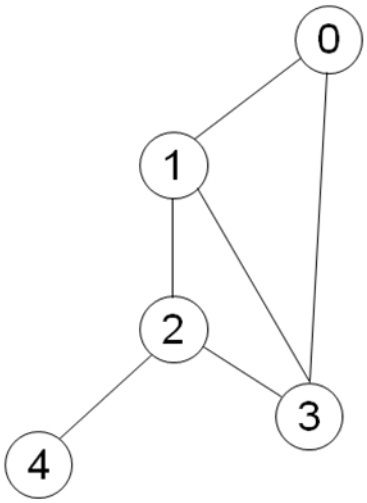
En ocasiones son útiles grafos valorados donde se les asigna un peso a las aristas

- CAMINO: Lista de vértices sucesivos que están conectados por aristas en el grafo
- CAMINO SIMPLE: Camino en que ningún vértice aparece 2 veces
- CICLO: camino simple de al menos 3 vértices donde el primer y último vértice son iguales
- CONEXO: Si existe camino entre cualquier par de nodos Grafos no dirigidos, en Grafos dirigidos sería una cadena que los unió
- CONEXO Fuertemente: Si existe camino entre cualquier par de nodos Grafos dirigidos
- COMPLETO tiene una arista para cualquier par de vértices
- ADYACENTE: Dos vértices se dicen **adyacentes** si existe una arista que los une
- GRAFO sin CICLOS: son llamados árboles

Existen varias formas para representar grafos en el ordenador, las más utilizadas son en lista de adyacencias y en matriz de adyacencias.

REPRESENTACION de GRAFOS

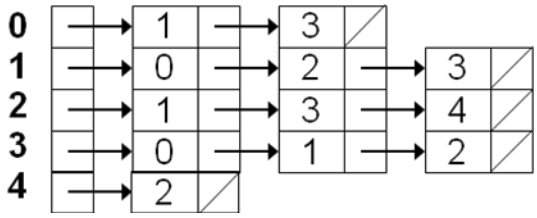
GRAFO NO DIRIGIDO



Matriz de adyacencia

	0	**1**	**2**	**3**	**4**
0	0	1	0	1	0
1	1	0	1	1	0
2	0	1	0	1	1
3	1	1	1	0	0
4	0	0	1	0	0

Lista de adyacencia



MATRIZ de ADYACENCIA

El tamaño de la matriz es de |V|x|V| de elementos, en las posiciones Aij

Aij =1 si Existe arista entre Vi y Vj

Aij =0 si no Existe

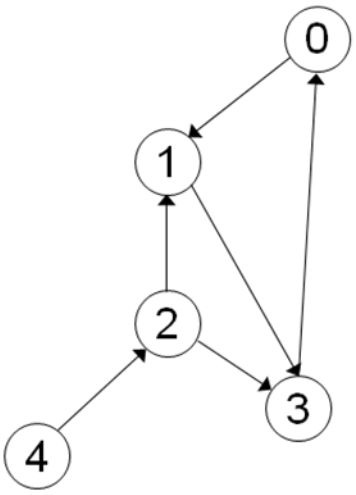
Ineficiente si varían los vértices o hay pocas aristas

LISTA de ADYACENCIA

El tamaño de la Lista de Listas es de |V|x|E| de elementos

No es ideal cuando el grafo es denso

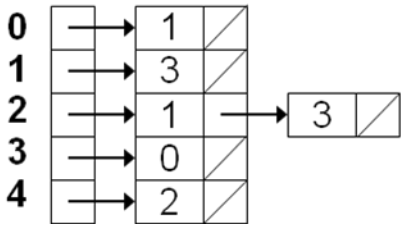
GRAFO DIRIGIDO



Matriz de adyacencia

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

Lista de adyacencia



Algoritmos de Búsqueda principales sobre GRAFOS

En anchura *BFS* - *Breadth First Search*: Permite recorrer el grafo en niveles (de los nodos más cercanos a los más lejanos). Utiliza una COLA para auxiliarse en la tarea

En profundidad *DFS* - *Depth First Search*: Busca caminos que parten desde el nodo de salida hasta que ya no es posible avanzar más, después volver atrás en busca de caminos alternativos inexplorados.

Ambos algoritmos tienen un tiempo de **O(|V|+|A|)** dado que para buscar un camino se recorren en el peor caso todos los Vértices y las Aristas del Grafo.

BFS: Búsqueda en Amplitud

Busca un camino entre el vértice ORIgen y el vértice DESTino y devuelve si existe o no existe, utiliza una **COLA** para realizar la búsqueda, primero entre todos los vértices adyacentes y luego los adyacentes de estos. **No olvidar Inicializar Padre en -1 y Marca en FALSO,** antes de empezar la búsqueda.

BFS (ORI, DEST) →Encontre

```
Encontre ← NO
ARR[ORI].MARCA ← SI
COLA.Poner(ORI)  //Poner al final de la lista

MIENTRAS (COLA <> VACIA y ENCONTRE = NO)
    i_esimo ←1
    VERT ← COLA.Sacar()   //Sacar del principio de la lista
    ADYA ← ARR[VERT].Lista. SacarSiguienteAdyacente (i_esimo)

        MIENTRAS (ADYA <> NULO y ENCONTRE = NO)

                SI (ARR[ADYA].Marca=No)  Entonces
                        ARR[ADYA].Padre← VERT

                        SI (ADYA=Dest)  Entonces
                                Encontre ← SI
                        C/c
                                ARR[ADYA].Marca ← Si
                                COLA.Poner(ADYA) //Poner al final
                        FinSi
                FinSi
            i_ esimo ←i_ esimo +1
            ADYA ← ARR[VERT].Lista. SacarSiguienteAdyacente(i_esimo)
        FinMientras
FinMientras
```

Terminar Devolviendo ENCONTRE

DFS: Búsqueda en Profundidad

Busca un camino recursivamente (se puede usar una PILA en lugar de la recursión) entre el vértice ORI y el vértice DEST y devuelve si existe o no existe un camino, realizando una búsqueda en profundidad uno a uno entre los vértices adyacentes. **Importante inicializar Padre en -1 y Marca en FALSO,** antes de empezar la búsqueda.

DFS(ORI, DEST) →Encontre_cam

```
Encontre_cam ← No
array[ORI].marca←Si
i_esimo ←1
ADYA = array[ORI].Lista. SacarSiguienteAdyacente (i_esimo)

Mientras (ADYA <> Nulo )

    Si ( Encontre_cam=No y  array[ADYA].marca =No ) Entonces

        array[ADYA].padre = ORI
        Si (ADYA = DEST)
            encontre_cam ← Si
        C/c
            encontre_cam ← DFS (ADYA,DEST)
        FinSi

    FinSi
  i_ esimo ←i_ esimo +1
  ADYA = array[ORI].Lista. SacarSiguienteAdyacente (i_esimo)
finMientras
```

Terminar devolviendo encontre_cam