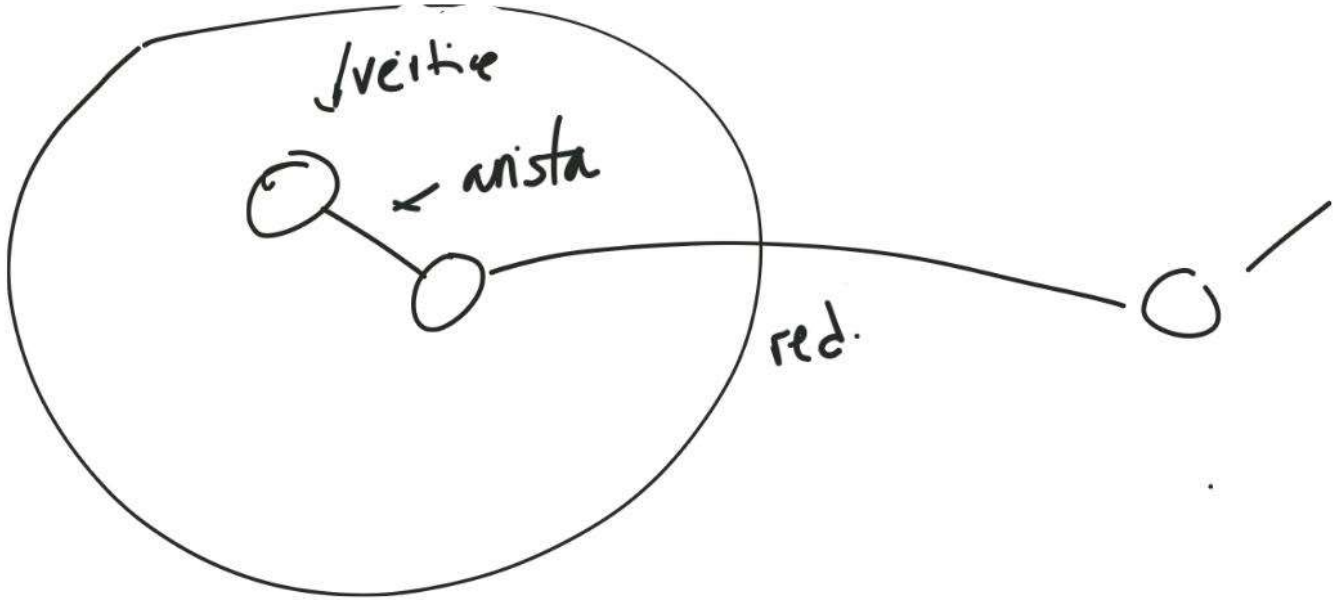
Grafos



Un grafo G es una tupla G=(V, A), donde V es un conjunto no vacío de **vértices** y A es un conjunto de **aristas** o **arcos**.

Cada **arista** es un par (v, w), donde v, w pertenecen a V.

Grafo no dirigido: Las aristas no están ordenadas, es decir:

(v, w) = (w, v)

Grafos dirigidos (o digrafos): Los pares sí están ordenados:

<v, w> ≠ <w, v>

<v, w> ⇒ v = cabeza de la arista, w = cola de la arista

Un vértice **w es adyacente a v** sí y sólo si (v, w) (ó <v, w>) pertenece a A.

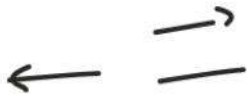
Camino de un vértice w_1 a w_q: es una secuencia $w_1, w_2, ..., w_q \in V$, tal que todas las aristas $(w_1, w_2), ..., (w_{q-1}, w_q) \in A$.

Longitud de un camino: número de aristas del camino = nº de nodos -1

Camino simple: aquel en el que todos los vértices son distintos (excepto el primero y el último que pueden ser iguales).

Ciclo: es un camino en el cual el primer y el último vértice son iguales. Se llama ciclo simple si el camino es simple. En grafos no dirigidos es necesario que las aristas sean diferentes.

Dados dos vértices v, w, se dice que están **conectados** si existen un camino de v a w.



Un grafo es **conexo** (o **conectado**) si hay un camino entre cualquier par de vértices. Si es un grafo dirigido se llama **fuertemente conexo**.

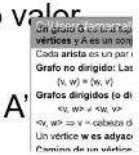
Un grafo es **completo** si existe una arista entre cualquier par de vértices. Para n nodos, ¿cuántas aristas tendrá el grafo?

Grado de un vértice v: número de arcos que inciden en él. Para grafos dirigidos, **grado de entrada:** nº de aristas con <x, v>; **grado de salida:** nº de aristas con <v, x>.

Un grafo está **etiquetado** si asociamos a cada arista un peso o valor.

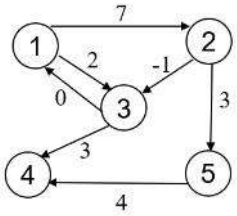
Grafo con pesos: grafo etiquetado con valores numéricos.

Un **subgrafo** de G=(V, A) es un grafo G'=(V', A') tal que V'⊆V y A'



Representación mediante matrices de adyacencia.

- El conjunto de aristas es representado mediante una matriz M[nodo, nodo] de booleanos, donde M[v, w]= 1 sí y sólo si (v, w) $\in$ A.
- Si el grafo está etiquetado, la matriz será de elementos de ese tipo, por ejemplo, caracteres o enteros. Tomará un valor nulo si no existe ese arco.



	1	2	3	4	5
1		7	2		
2			-1		3
3	0			3	
4					
5				4	

Si el grafo es no dirigido, M[v, w] = M[w, v]. La matriz es simétrica.

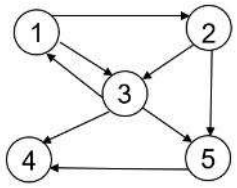
Problemas.

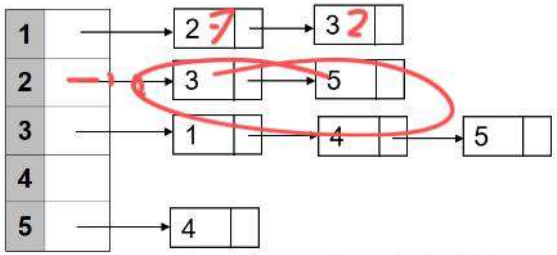
- Si el número de nodos es muy grande y hay poca conectividad (pocos arcos, en relación al máximo posible) se desperdicia memoria.
- Debemos conocer los tamaños aproximados que van a tener los grafos.

¿Cómo se calcularía el grado de entrada o de salida de un nodo?

Representación mediante listas de adyacencia.

- Para cada nodo de V tendremos una lista de aristas que parten de ese nodo. Estas listas están guardadas en un array de nodos cabecera.





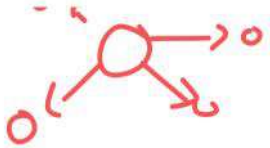
Grafo etiquetado: añadir un nuevo campo a los elementos de la lista.

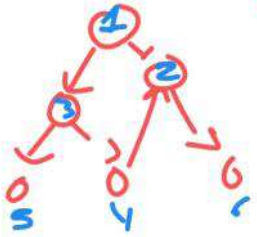
Si el grafo es no dirigido entonces cada arista (v, w) será representada dos veces, en la lista de v y en la de w.

En grafos dirigidos.

- Calcular el grado de salida: recorrer la lista correspondiente. Grado de entrada: recorrer todas las listas.
- Otra posibilidad: tener otra lista con las aristas que llegan a un nodo dado.

La mejor representación dependerá de las características del problema.

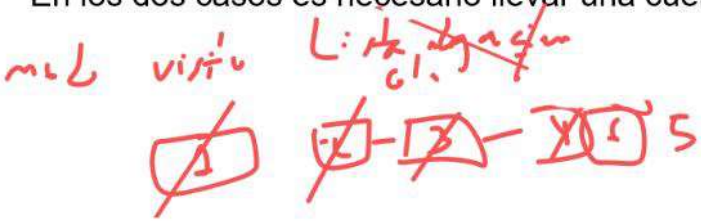


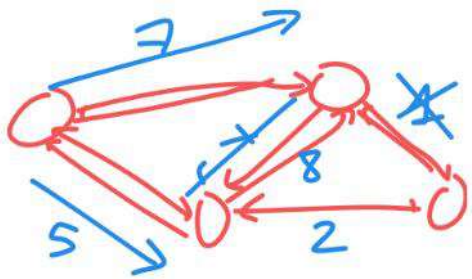


Igual que los recorridos en árboles, se parte de un nodo dado y sirven para visitar los vértices y los arcos de manera sistemática.
Existen dos tipos de recorridos:

- **Búsqueda primero en profundidad.** Es equivalente a un recorrido en preorden de un árbol. Se elige un nodo v de partida. Se marca como visitado y se recorren los nodos no visitados adyacentes a v, usando recursivamente la búsqueda primero en profundidad.
- **Búsqueda primero en amplitud o anchura.** Es equivalente a recorrer un árbol por niveles. Dado un nodo v, se visitan primero todos los nodos adyacentes a v, luego todos los que están a distancia 2 (y no visitados), a distancia 3, y así sucesivamente hasta recorrer todos los nodos.

El recorrido puede ser para grafos dirigidos o no dirigidos.
En los dos casos es necesario llevar una cuenta de los nodos visitados.





Definición: Un **árbol de expansión** de un grafo no dirigido G=(V, A) y conexo es un subgrafo G'=(V, A') no dirigido, conexo y sin ciclos.

Si el grafo es ponderado, el **coste del árbol de expansión** será la suma de los costes de las aristas.

Problema del árbol de expansión de coste mínimo:

Dado un grafo ponderado no dirigido, encontrar el árbol de expansión de menor coste.

Árbol de expansión mínimo

Algoritmo de Prim

Escoger un vértice cualquiera v. El árbol consta sólo del nodo v.

Del resto de vértices, buscar el que esté más próximo a v (con una arista (w, v) de mínimo costo). Añadir w y la arista (w, v) al árbol.

Buscar el vértice más próximo a cualquiera de estos dos, añadir ese vértice y la arista al árbol de expansión.

Así sucesivamente hasta haber añadido los n vértices.

Algoritmo de Kruskal

- Dado un grafo ponderado G=(V, A), el algoritmo parte de un grafo G'= (V, Ø). Cada nodo es una componente conexa en sí misma.
- En cada paso de ejecución se elige la arista de menor costo de A.
  - Si une dos nodos que pertenecen a distintas componentes conexas entonces se añade al árbol de expansión G'.
  - En otro caso no se coge, ya que formaría un ciclo en G'.
- Acabar cuando G' sea conexo: cuando tengamos n-1 aristas.

Definición: Dado un grafo ponderado G= (V, A) (dirigido o no) y un camino w_1, w_2, ..., w_q en G, el **costo del camino** será la suma de los costos asociados a las aristas (w_1, w_2), ..., (w_{q-1}, w_q).

Si el grafo es no ponderado, normalmente el costo se asocia con la longitud del camino.

Problema de los caminos más cortos por un origen:

Encontrar los caminos más cortos entre un nodo origen dado y todos los demás nodos.

Algoritmo de Dijkstra

Supongamos un grafo ponderado G (con pesos ≥ 0) y un nodo origen v.

El algoritmo trabaja con dos conjuntos:

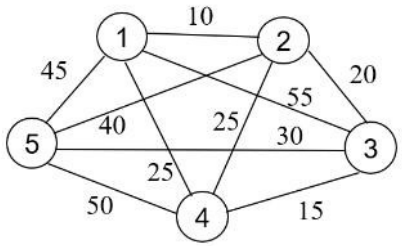
- **S: conjunto de nodos escogidos**, para los cuales se conoce el camino de distancia mínima al origen.
- **C: conjunto de nodos candidatos**, pendientes de calcular el camino mínimo. Conocemos los caminos mínimos al origen pasando por nodos de S.

En cada paso coger del conjunto de candidatos el nodo con distancia mínima al origen. Recalcular los caminos de los demás candidatos pasando por el nodo cogido.

```
DijkstraAlgorithm(weighted simple digraph, vertex first)
    for all vertices v
        currDist(v) = ∞;
    currDist(first) = 0;
    toBeChecked = all vertices;
    while toBeChecked is not empty
        v = a vertex in toBeChecked with minimal currDist(v);
        remove v from toBeChecked;
        for all vertices u adjacent to v and in toBeChecked
            if currDist(u) > currDist(v) + weight(edge(vu))
                currDist(u) = currDist(v) + weight(edge(vu));
                predecessor(u) = v;
```

Problema del viajante (agente viajero)

Dado un grafo no dirigido, completo y ponderado G = (V, A), encontrar un ciclo simple de costo mínimo.



Ejemplos: Un repartidor de determinadas mercancías tiene encargos en varias ciudades. ¿Qué ruta debe seguir para que el costo de desplazamiento sea mínimo?

El problema del viajante es un problema **NP-completo**, con un orden de complejidad exponencial. No existe una solución polinómica.

Podemos aplicar heurísticas, obteniendo soluciones aproximadas, no necesariamente óptimas.