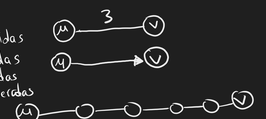
Grafos

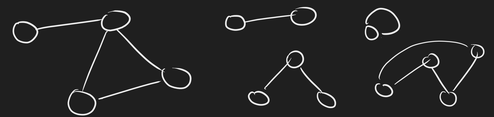
* Nodo o vértice
* Aristas
  - no dirigidas
  - dirigidas
  - ponderadas
  - no ponderadas



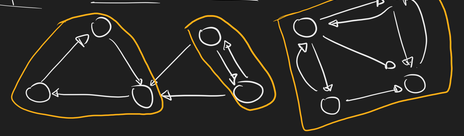
* Camino

* Conexión: Un nodo u está conectado a v si existe un camino de u a v

* Componente Conexa: Conjunto de nodos $V = \{v_1, v_2, \dots, v_n\}$ tal que v_i está conectado a v_j, $\forall i,j$



* Componente Fuertemente Conexa



* Multiaristas

* Ciclos

* Self-loop

Representación de grafo

N = # nodos, M = # aristas

u_1 v_1

* Almacenando aristas

vector<pair<int, int>> edges(m)

* Matriz de Adyacencia

$M_{ij} = 0$: No existe arista de $i \to j$

$M_{ij} = 1$: Existe arista de $i \to j$

Ventaja: Representación Matricial

Determinar si v es adyacente a v en $O(1)$.

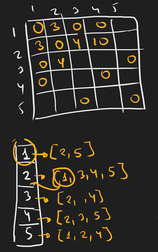
Desventaja: $O(n^2)$ en memoria

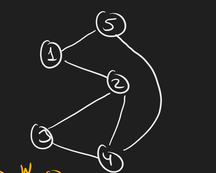
* Lista de Adyacencia

$O(N+M)$

Espacial — Temporal


```
5 7
1 2 3
3 4 5
5 1 2
3 2 4
4 5 7
4 2 10
5 2 8
```



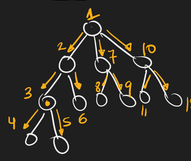


adj[u].push_back(v, w)

adj[v].push_back(u, w)

* Algoritmos para recorrer un grafo

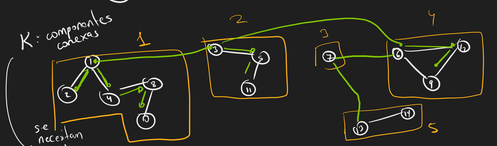
1) DFS: (Depth - First - Search)




```
vector<bool> vis(N+1)
DFS(u):
  vis[u] = true;
  for(int v: adj[u])
    if(vis[v]) continue
    DFS(v)
```

↳ O(N+M)

K: componentes conexas



se necesitan K-1 aristas

vis = true true true true true true true true true true true true

=

vector<vector<bool>> vis

DFS

grid[x][y] = '.'

(x,y)

↳ (x-1, y)
(x, y-1)
(x+1, y)
(x, y+1)

```
5 8
########
#..#...#
####.#.#
#..#...#
########
```