Proofs:

1) in the Σ, every edge is counted twice
2) in a simple graph there are $\binom{n}{2}$ possible pairs of vertices
3) fix a root. Then E represents father-child relationships, which are $n-1$
4) G is a tree that may have cycles $\Rightarrow$ it can only have more edges than a tree
5) G is a tree that may not be connected $\Rightarrow$ it can only have less edges than a tree

Representing a graph

How to encode a graph for use in an algorithm?

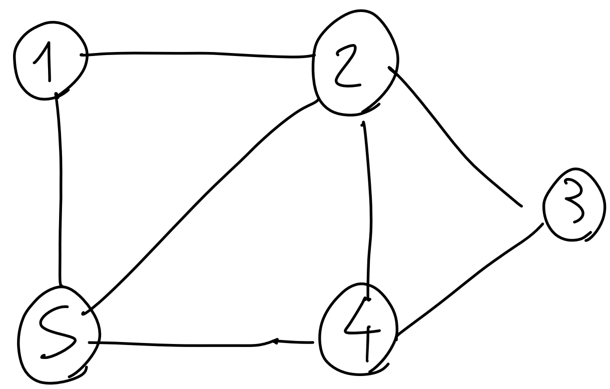
List of vertices L_V & list of edges L_E with pointers to $L_V(v)$, $L_V(v)$

Let's assume vertices are called $1, 2, \dots, n$

To allow for direct access to edges, one of the following data structures are used, in addition to L_V, L_E:

- <u>Adjacency list</u>: an array A of n lists, one $\forall$ vertex $v \in V$, each containing all the vertices adjacent to v

example:



$A =$

1	2 5
2	1 3 4 5
3	2 4
4	2 5 3
5	4 1 2

what if directed? only vertices pointed from that vertex

Pro: space usage: $\Theta(n+m)$ i.e <u>linear</u>

Con: no quick way to determine if a given edge is in the graph

- **Adjacency matrix**: a $n \times n$ matrix A s.t.

$$A[i,j] = \begin{cases} 1 & \text{if edge } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

$$A = \begin{array}{c c} & \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \left[\begin{array}{ccccc} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{array}\right] \end{array} \longrightarrow \text{symmetric}$$

what if graph is directed? asymmetric

weighted? w and $-$/null

Pro: quick to determine if a given is in the graph

Con: space required is $\Theta(n^2)$ $\longrightarrow$ can be superlinear in the input size

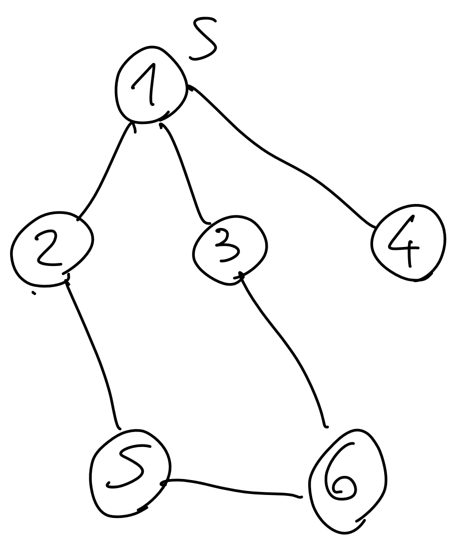
Graph Search and its applications

↳ a.k.a. traversal / exploration

A systematic way to <u>explore</u> a graph starting from a vertex $s \in V$ visiting all the vertices

- Depth-First Search (DFS)
- Breadth-First Search (BFS)

example:



DFS: $1 \to 2 \to 5 \to 6 \to 3 \to 4$

BFS: $1 \to 2 \to 3 \to 4 \to 5 \to 6$

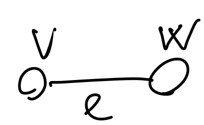
DFS algorithm:

- recursive algorithm that starting from $s \in V$ "visits" all the vertices of the connected component $C_s \subseteq G$ containing s
- adjacency list
- every vertex v has a field $L_V[v].ID$ — 1 if visited / 0 otherwise
- every edge e has a field

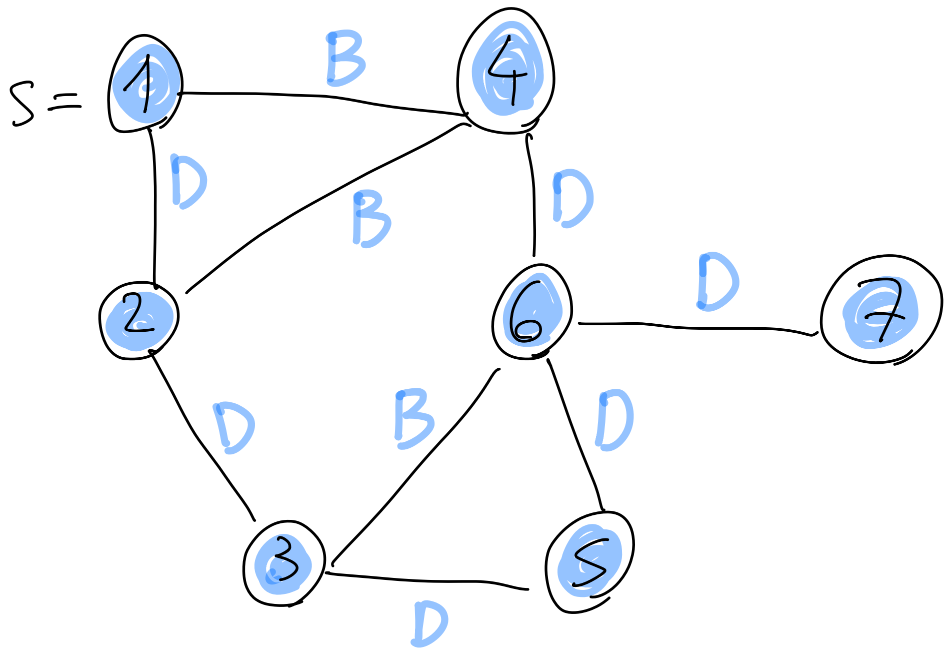
$L_E[e].label$ — null initially / DISCOVERY EDGE or BACK EDGE

DFS(G, v) (first invoke: $v = s$)

```
visit v ; L_V[v].ID = 1
for all e ∈ G.incidentEdges(v) do
    if L_E[e].label = null then
        w = G.opposite(v, e)
        if L_V[w].ID = 0 then
            L_E[e].label = DISCOVERY EDGE
            DFS(G, w)
        else L_E[e].label = BACK EDGE
```



Example:



Correctness:

At the end of the algorithm

1) all vertices of C_s have been visited, and all the edges in C_s are labeled either DISC/BACK edge

2) the set of DISC. EDGES is a spanning tree T of C_s $\longrightarrow$ called "DFS tree"

Proof:

1) (short: by construction)
by contradiction, assume $\exists\ v \in C_s$ not visited

$\exists$ path from s to v

$$s = v_0 \to v_1 \to v_2 \to \dots \to v_\ell = v$$

$v_j \in C_s$

v_i : first vertex not visited

contradiction: $DFS(G, v_{i-1})$ must have been executed $\longrightarrow$ find v_i not visited $\Longrightarrow DFS(G, v_i)$

a vertex v is visited only when $DFS(G, v)$ is invoked $\Longrightarrow$ DFS is called $\forall\ v \in C_s$
$\Longrightarrow$ all incident (on v) edges are labeled, by construction

2) DFS is called $\forall v \in C_s$, once, and $\forall v \neq s$ $\exists$ a vertex u s.t. (u,v) $\exists$ and is labeled DISCOVERY EDGE and DFS(G, v) is invoked from DFS(G, u).

We say that v gets "discovered" by u, and let's call u "father" of v

$\Rightarrow$ $\forall v \in C_s$ $v \neq s$

- $\exists!$ father
- going back father to father eventually s is reached

$\Rightarrow$ the set of DISC. EDGES is a rooted tree with all the vertices of C_s

$\rightarrow$ it's a spanning tree of C_s

Complexity of DFS(G, s)

n_s = n° of vertices of C_s

m_s = " edges "

$$\Theta\left(\sum_{v \in C_s} d(v)\right) = \Theta(m_s)$$

obs.: C_s is connected $\Rightarrow m_s \geq n_s - 1$

$\Rightarrow m_s = \Omega(n_s)$

Application: visit all the graph:

```
for v = 1 to n do
    if L_V[v].ID = 0 then
        DFS(G, v)
```

Complexity: $\Theta(n + m)$

Problem: Given a graph G and 2 vertices s, t return, if it exists, a path from s to t

Problem: Given a graph G return a cycle (if any)