Proof:

1) in the Σ, every edge is counted twice
2) in a simple there are $\binom{n}{2}$ possible pair 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, thus it can only have more edges
5) G is a tree that may not be connected, thus it can only have ~~more~~ less edges

Representing a graph

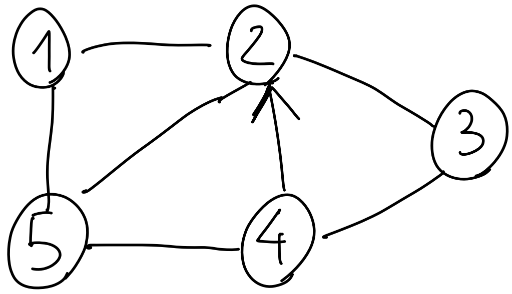
how to encode a graph for use in an algorithm?

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

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

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

example:



$$A = \begin{array}{c|l} 1 & 2\ 5 \\ 2 & 1\ 3\ 4\ 5 \\ 3 & 2\ 4 \\ 4 & 2\ 5\ 3 \\ 5 & 4\ 1\ 2 \end{array}$$

what if directed? only vertices pointed from that vertex

Pro: space required: $\Theta(n+m)$ i.e. <u>linear</u>
Con: no quick way to determine if a given edge is present in the graph

— Adjacency Matrix: a $n \times n$ matrix A s.t.

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

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 1 & 0 & 0 & 1 \\ 2 & 1 & 0 & 1 & 1 & 1 \\ 3 & 0 & 1 & 0 & 1 & 0 \\ 4 & 0 & 1 & 1 & 0 & 1 \\ 5 & 1 & 1 & 0 & 1 & 0 \end{array}$$

symmetric

what if graph is directed? asymmetric
weighted? w and —/null
can have parallel edges? number of edges

Pro : quick to determine if a given edge is present
Con : space required : $\Theta(n^2)$ $\rightarrow$ can be superlinear in the input size

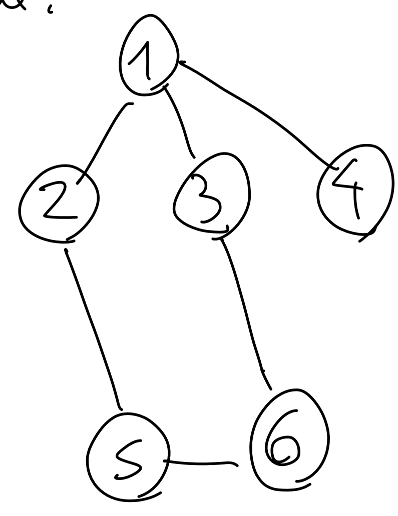
Graph Search and its applications

↳ aka traversal/exploration

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

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

example :



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

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

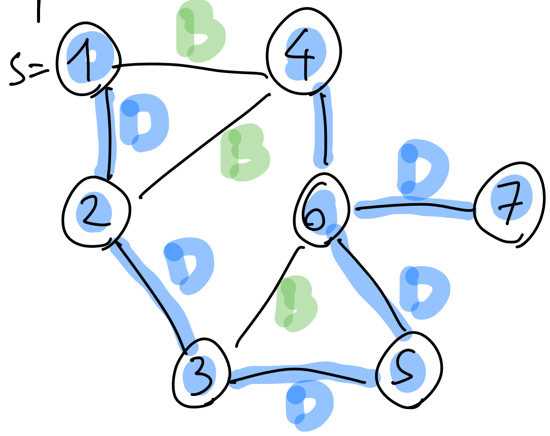
DFS algorithm :

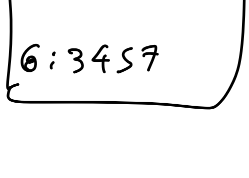
- recursive algorithm that starting from $s \in V$ "visits" all the vertices of the connected component $C_s \subseteq G$ containing s
- adjacency lists
- every vertex v has a field $L_V[v].ID$ — 1 if visited / 0 otherwise
- every edge e " $L_E[e].label$ — null initially / DISCOVERY EDGE or BACK EDGE

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

visit v; $L_V[v].ID \leftarrow 1$
for all $e \in G.\mathrm{incidentEdges}(v)$ do
  if $L_E[e].label = $ null then
    $w \leftarrow G.\mathrm{opposite}(v, e)$
    if $L_V[w].ID = 0$ then
      $L_E[e].label \leftarrow$ DISCOVERY EDGE
      DFS(G, w)
    else $L_E[e].label \leftarrow$ BACK EDGE

example:





Correctness :

At the end of the algorithm

1) all the 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 $\longrightarrow$ DFS tree

proof :

1) (short : by construction)

by contradiction, assume $\exists\ v \in C_S$ not visited

$\exists$ path from s to v

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

$v_j \in C_S$

v_i

first vertex not visited

contradiction : DFS(G, v_{i-1}) must have been executed

$\to$ find v_i not visited $\Longrightarrow$ DFS(G, v_i)

DFS is ~~called~~ called $\forall\ v \in C_S \Longrightarrow$ all incident edge on v 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

$\forall\, v \in C_s \quad v \neq s$

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

$\Rightarrow$ the set of DISC. EDGES is a rooted tree touching all the vertices di 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

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

obs.: C_s is connected $\Rightarrow m_s \geq n_s - 1 \Rightarrow m_s = \Omega(n_s)$

Visit all the graph:

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

Complexity: $\underline{\Theta(n + m)}$

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

Problem: Given a graph determine a cycle (if any)