Exercises:

s-t path: — $\forall\, v \in V$ add a field $L_V[v]$.parent

— modify DFS(G,v) s.t. when a DISC. EDGE (v,w) is labeled then $L_V[w]$.parent $= v$

— Run DFS(G,s). Check if t has been visited $\longrightarrow$ NO: then return "No path"

$\searrow$ YES: starting from t, follow the "parent" label so as to build a path from t to s

cycle: — $\forall\, v \in V$ add a field $L_V[v]$.parent

$\forall\, e \in E$ " $L_E[e]$.ancestor

— (v,w) is a DISC. EDGE then $L_V[w]$.parent $= v$

(v,w) is a BACK EDGE then $L_E[e]$.ancestor $= w$

$\longrightarrow$ w is an ancestor of v in the DFS tree

— Run DFS on each conn. component

— Check all the edges. As soon as an edge $e = (v,w)$ is found as BACK EDGE and $L_E[e]$.ancestor $= w$ then return a cycle adding to e all the edges found in the path from v to w. If no BACK EDGES then return "No cycle"

Complexity: $\Theta(n+m)$

More applications of DFS:

Connected components / connectivity

↓ labeling all the vertices of G s.t. 2 vertices have the same label $\iff$ they are in the same conn. component

↓ return whether the graph is connected or not

$$Lv[v].ID \begin{cases} 0 & \text{if not visited} \\ 1 & \text{if visited} \end{cases}$$

```
for v ← 1 to n do
    Lv[v].ID = 0
k = 0
for v ← 1 to n do
    if Lv[v].ID = 0 then
        k ← k+1
        DFS(G, v, k)
if k = 1 then return YES
return NO
```

Conn. comp.

Connectivity

→ modify DFS(G,v):

~~Lv[v].ID ← 1~~

Lv[v].ID ← k

Complexity: $\Theta(n+m)$

Summarizing:

Given a graph $G = (V, E)$ the following problems can be solved in $\Theta(n+m)$ using the DFS

- test if G is connected
- find the connected components of G
- find a spanning tree of G (if G is connected)
- find a path between two vertices (if any)
- find a cycle (if any)

Breadth-First Search (BFS)

An iterative algorithm that starting from a source vertex s "visits" all the vertices in the same connected component of s, and partitioning the vertices in levels L_i depending on their distance i from s.

We'll use adjacency list to represent G

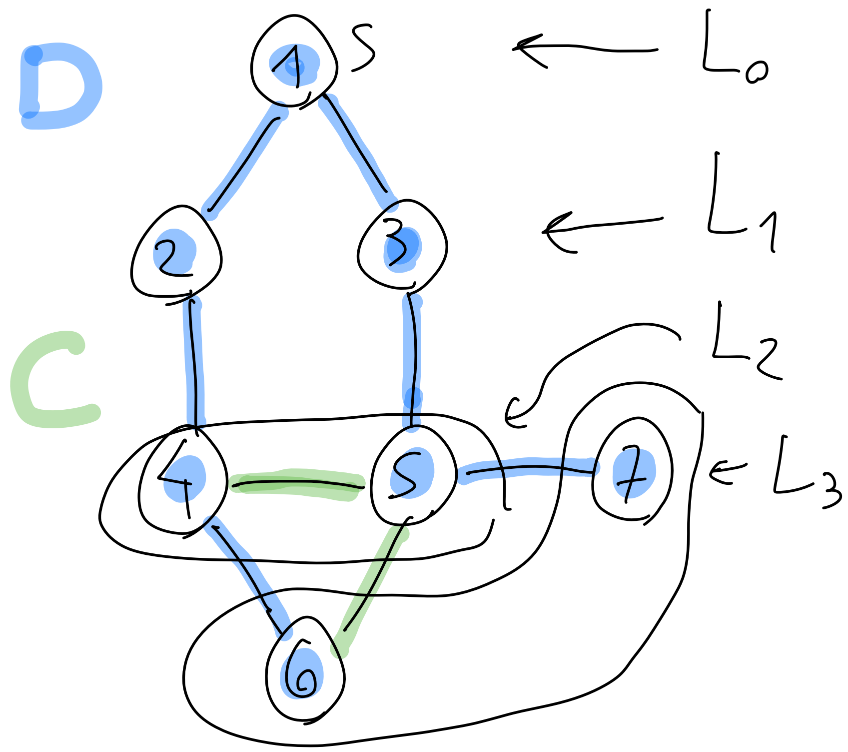
$L_V[v].ID$ — 0 not visited; 1 visited

$L_E[e].label$ — null if e has no label; DISCOVERY EDGE; CROSS EDGE

BFS (G, s)

```
visit s
L_V[s].ID = 1
create a set L_0 containing s
i ← 0
while (!L_i.isEmty()) do
    create a set of vertices L_{i+1}, empty
    for all v ∈ L_i do
        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
                    visit w
                    L_V[w].ID ← 1
                    add w in L_{i+1}
                else L_E[e].label ← CROSS EDGE
    i = i + 1
```

Example:



Correctness:

At the end of BFS(G,s) we have:

1. all vertices in C_s are visited & all the edges are labeled DISC./CROSS EDGE
2. the set of DISC. EDGES are a spanning tree T of C_s → BFS tree
3. $\forall v \in L_i$ the path in T from s to v has i edges and every other path from s to v has $\geq i$ edges

proof of 3) :

$P: s = v_0 \to v_1 \to \dots \to v_i = v$ where
$v_j \in L_j$ is "discovered" from v_{j-1} $\forall j \implies$
(v_{j-1}, v_j) is a DISC. EDGE

By contradiction, assume $\exists$ a path
$P': s = \underline{z_0 \to z_1 \to \dots \to z_t} = v$ with $t < i$

$s = z_0 \in L_0$
$z_1 \in L_1$
$z_2 \in L_2$ or L_1
$\vdots$
$z_t \in L_1$ or L_2 or ... L_t

$z_t = v \to v \notin L_i$: contradiction

Complexity:
$\forall v \in C_s$ 1 iteration of the 1st forall and
$d(v)$ iterations of the 2nd for all

$\implies \Theta(m_s) \to \Theta(m)$ is G is connected

Applications: same as for DFS, in $\Theta(n+m)$ time

Given $G=(V,E)$, $s,t \in V$, return (if any) a shortest path from s to t

- $\forall v \in V$ $L_V[v]$.parent
- modify BFS(G,s) s.t. when (v,u) is labeled DISC. EDGE then $L_V[u]$.parent $= v$
- Run BFS return the set of child–parent edges

$\Theta(m_s)$

<u>Minimum</u> <u>Spanning</u> <u>Trees</u>

Application: a set of objects that I want to <u>interconnect</u> in the cheapest possible way

example: computers — cable

a golden problem, studied since '20

Def.:

Input: a graph $G = (V, E)$ undirected, connected, and weighted

$\hookrightarrow w : E \to \mathbb{R}$

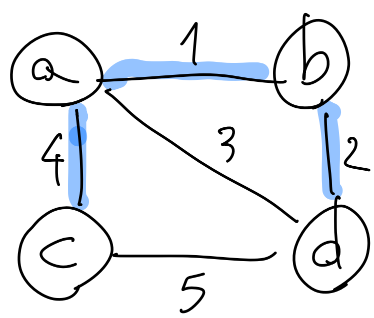
$w(u,v)$ = cost of edge (u,v)

Output: a spanning tree $T \subseteq E$ of G s.t.

$$w(T) = \sum_{(u,v) \in T} w(u,v) \text{ is minimized}$$

minimum-~~weight~~ spanning tree

example:



$MST(G) = ?$