Exercises:

s-t path:

- $\forall v \in V$ add a field $L_V[v]$.parent
- modify DFS(G, v) s.t. when a DISCOVERY EDGE (v, w) is labeled then $L_V[w]$.parent $= v$
- Run DFS(G, s): check if t has been visited → NO: return "No path"

  YES: starting from t, follow the "parent" labels so as to build a path from t to s

cycle:

- $\forall v \in V$ add a field $L_V[v]$.parent
- $\forall e \in E$ add a field $L_E[e]$.ancestor
- (v, w) is a DISC. EDGE then $L_V[w]$.parent $= v$
- (v, w) is a BACK EDGE then $L_E[e]$.ancestor $= w$

→ w is an ancestor of v in the DFS tree

- run DFS on each connected 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 EDGE is found then return "No cycles"

Complexity : $\Theta(n + m)$

More applications of DFS :

graph connectivity : return whether the graph is connected or not

connected components : return a labeling of all the vertices of G s.t. 2 vertices have the same label if and only if they are in the same connected component

idea:
modify DFS(G,v): ~~$L_V[v].ID \leftarrow 1$~~
$L_V[v].ID \leftarrow k$ integer, label of the k-th component

for $v = 1$ to n do
    $L_V[v].ID = 0$

$k = 0$

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

if $k = 1$ then return YES
return NO

connected components

graph connectivity

Complexity: $\Theta(n + m)$

Summary:

Given a graph G the following problems can be solved in $\Theta(n+m)$ time 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 "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

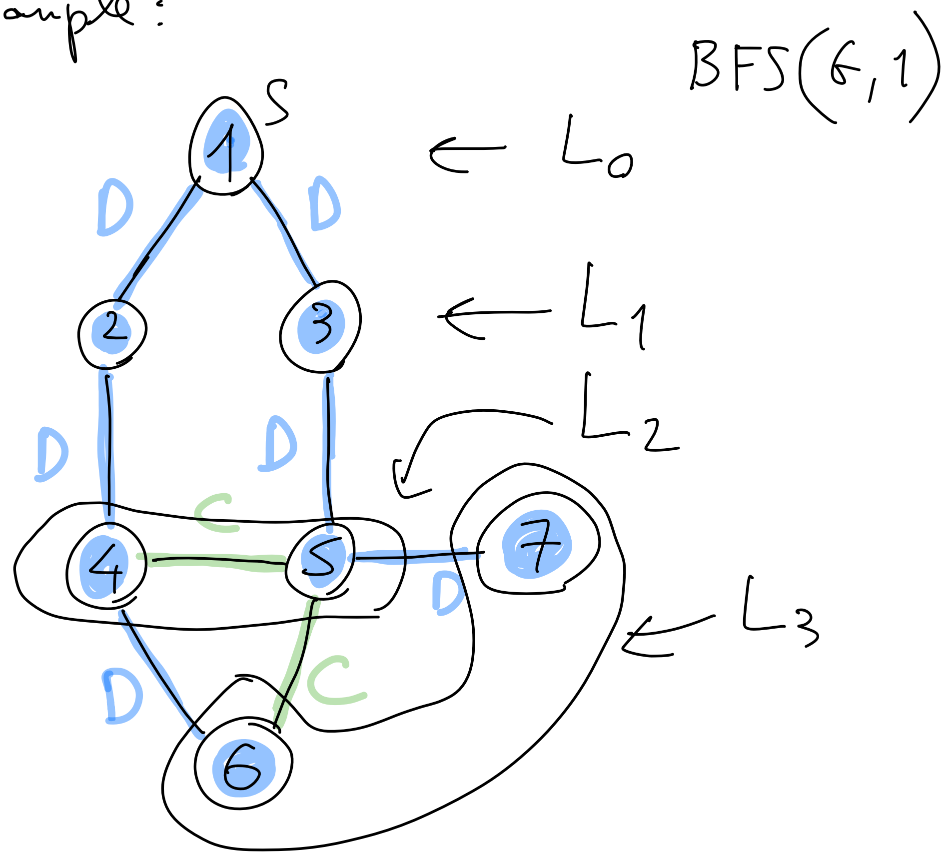
$$L_V[v].ID = \begin{cases} 0 & \text{not visited} \\ 1 & \text{visited} \end{cases}$$

$$L_E[e].label = \begin{cases} \text{null} & \text{if } e \text{ has no label} \\ \text{DISCOVERY EDGE} \\ \text{CROSS EDGE} \end{cases}$$

BFS (G, s)

```
visit s : Lv[s].ID = 1
create a set L0 containing s
i = 0
while (!Li.isEmpty()) do
    create a set of vertices Li+1, empty
    for all v ∈ Li do
        for all e ∈ G.incidentEdges(v) do
            if LE[e].label = null then
                w = G.opposite(v, e)
                if Lv[w].ID = 0 then
                    LE[e].label = DISCOVERY EDGE
                    visit w
                    Lv[w].ID = 1
                    add w to Li+1
                else LE[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 and all the edges are labelled DISC/CROSS EDGE

2) the set of DISCOVERY EDGES are a spanning tree T of C_s $\rightarrow$ called "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 1) and 2) : as for DFS

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} $\forall j$

$\Rightarrow (v_{j-1}, v_j)$ is a DISC. EDGE

$\Rightarrow$ P is a path in T

By contradiction, assume $\exists$ a path

P': $s = z_0 \to z_1 \to \dots \to z_t = v$ with $\underline{t<i}$

$s = z_0 \in L_0$

$z_1 \in L_1$

$z_2 \in L_2$ or L_1

$\vdots$

$z_t \in L_t$ or ... or L_2 or L_1

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

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

$$\Longrightarrow \Theta(m_s)$$

$(\Theta(m)$ if 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 DISCOVERY EDGE then $L_V[u]$.parent $= v$
- run BFS and return the set of child-parent edges

Complexity: $\Theta(m_s)$