Section 3.1: Exploring an Undirected Graph

Let $G = (V, E)$ be an undirected graph.

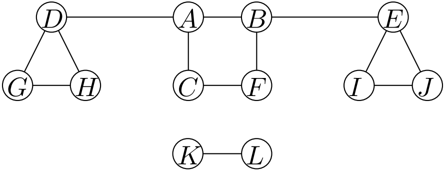
Task: Find all vertices that can be reached from a given vertex $v \in V$.

Algorithm *explore*(v)

```
visited(v) = TRUE
previsit(v)                                // See later
for each edge {u, v} ∈ E do
    if visited(u) = FALSE then
        call explore(u)
    end if
end for
postvisit(v)                               // See later
```

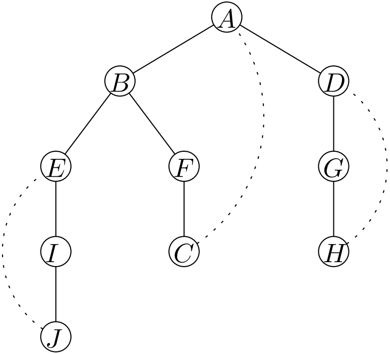
Algorithm *explore*(v)

```
visited(v) = TRUE
previsit(v)                                   // See later
for each edge {u, v} ∈ E do
    if visited(u) = FALSE then
        call explore(u)
    end if
end for
postvisit(v)                                  // See later
```



Run *explore*(A). In the for-loop, use alphabetical order (i.e., adjacency lists are sorted alphabetically). Each time an edge $\{u, v\}$ is traversed (because $visited(u) = \text{FALSE}$): u is discovered for the first time.

- Draw $\{u, v\}$ as a solid edge.
- All other edges: dotted.



The solid edges form a *tree* (connected, no cycle). These edges are called *tree edges*. The dotted edges are called *back edges*.

Why is algorithm *explore*(v) correct?

First, how can we explain that it always terminates?

Algorithm *explore*(v)

```
visited(v) = TRUE
previsit(v)                                  // See later
for each edge {u, v} ∈ E do
    if visited(u) = FALSE then
        call explore(u)
    end if
end for
postvisit(v)                                 // See later
```

Why is algorithm *explore*(v) correct?

First, how can we explain that it always terminates?

The number of vertices u such that "$\mathit{visited}(u) = \text{FALSE}$" decreases in each recursive call. Since there is a finite number of vertices, the algorithm eventually terminates.

Algorithm *explore*(v)

$\mathit{visited}(v) = \text{TRUE}$
$\mathit{previsit}(v)$ // See later
for each edge $\{u, v\} \in E$ **do**
  if $\mathit{visited}(u) = \text{FALSE}$ **then**
    call $\mathit{explore}(u)$
  end if
end for
$\mathit{postvisit}(v)$ // See later

How can we explain that it does visit all vertices that are reachable from v?

Algorithm *explore*(v)

```
visited(v) = TRUE
previsit(v)                                   // See later
for each edge {u, v} ∈ E do
    if visited(u) = FALSE then
        call explore(u)
    end if
end for
postvisit(v)                                  // See later
```

How can we explain that it does visit all vertices that are reachable from v?

Lemma

Assume that, initially, $visited(u) = \text{FALSE}$. *After* $explore(v)$ *has terminated,*

$$visited(u) = \text{TRUE}$$

$$\iff$$

there is a path from v *to* u.

Algorithm $explore(v)$

1: $visited(v) = \text{TRUE}$
2: $previsit(v)$ // See later
3: **for** each edge $\{u, v\} \in E$ **do**
4:     **if** $visited(u) = \text{FALSE}$ **then**
5:         call $explore(u)$
6:     **end if**
7: **end for**
8: $postvisit(v)$ // See later

Solid edges form a tree
(connected, no cycles)
These edges are called: <u>tree edges</u>

Dotted edges: <u>back edges</u>

Why is algorithm explore(v) correct?

Why does it terminate: number of vertices u with visited(u) = false decreases in each recursive call.

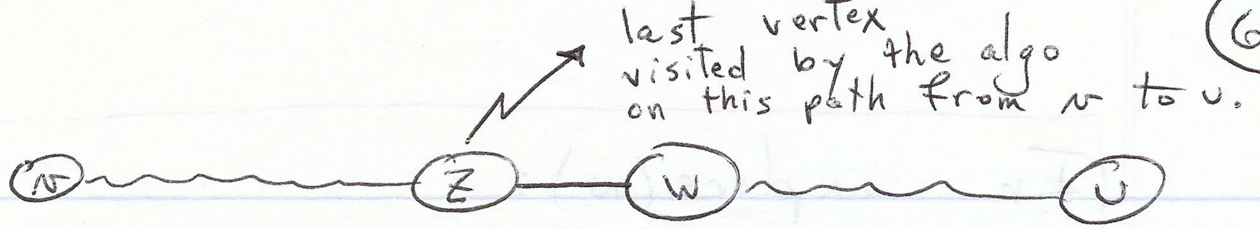
Assume that, initially, visited(u) = false.

<u>Claim</u>: After explore(v) has terminated:

visited(u) = true $\Leftrightarrow$ there is a path from v to u.

<u>proof</u>: [$\Rightarrow$] Follows from the algorithm: the algorithm "walks" from a vertex to a neighboring vertex

[$\Leftarrow$] By contradiction: Assume there is a path from v to u, and assume that, after termination, visited(u) = false. Consider any path from v to u:



So z was visited, but w was not. This is a contradiction. When visiting z, the algorithm notices that visited (w) = false and then visits w.

$\square$

<u>Connected components</u> of $G = (V, E)$:

number the connected components as $1, 2, 3, \ldots$

for each vertex v: ccnumber(v) = number of the connected component that v belongs to.

Algo DFS(G): // depth-first search

```
for all v ∈ V:
        visited(v) = false
cc = 0
for all v ∈ V:
        if visited(v) = false
                cc = cc+1
                explore(v)
```

Connected Components of $G = (V, E)$

The goal is to number the connected components as $1, 2, 3, \ldots$ such that for each vertex v,

$$ccnumber(v) = \# \text{ of the connected component that } v \text{ belongs to}$$

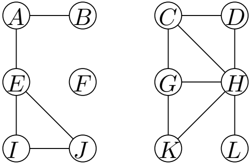
Connected Components of $G = (V, E)$

Algorithm $DFS(G)$

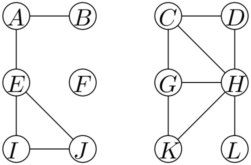
for all $v \in V$ **do**
    $visited(v) = false$
end for
$cc = 0$
for all $v \in V$ **do**
    if $visited(v) = false$ **then**
        $cc = cc + 1$
        $explore(v)$
    end if
end for

In $exlore(v)$,

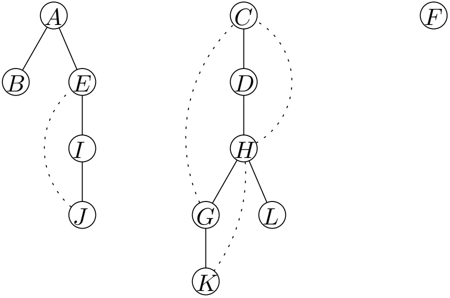
- $previsit(v) \equiv$ "$ccnumber(v) = cc$"
- $postvisit(v) \equiv$ " nil "



As usual, assume that the adjacency lists are sorted in alphabetical order.



As usual, assume that the adjacency lists are sorted in alphabetical order. We get the following DFS-forest.



$ccnumber = 1$ DFS-tree 1 | $ccnumber = 2$ DFS-tree 2 | $ccnumber = 3$ DFS-tree 3

Running Time of Depth-First-Search (DFS)

First for-loop : $O(|V|)$ time

Second for-loop :

Running Time of Depth-First-Search (DFS)

First for-loop : $O(|V|)$ time

Second for-loop :

- $\rightarrow$ *explore*(u) is called exactly once for each vertex u (this may be part of a recursive call)
- $\rightarrow$ time spent for *explore*(u), excluding recursive calls, is $O(1 + \mathit{degree}(u))$

Running Time of Depth-First-Search (DFS)

First for-loop : $O(|V|)$ time

Second for-loop :

- $\rightarrow$ *explore*(u) is called exactly once for each vertex u (this may be part of a recursive call)
- $\rightarrow$ time spent for *explore*(u), excluding recursive calls, is $O(1 + \mathit{degree}(u))$

Total time:

$$O\left(|V| + \sum_{u \in V} (1 + \mathit{degree}(u))\right)$$

Running Time of Depth-First-Search (DFS)

First for-loop : $O(|V|)$ time

Second for-loop :

$\rightarrow$ *explore*(u) is called exactly once for each vertex u (this may be part of a recursive call)

$\rightarrow$ time spent for *explore*(u), excluding recursive calls, is $O(1 + \mathit{degree}(u))$

Total time:

$$O\left(|V| + \sum_{u \in V} (1 + \mathit{degree}(u))\right) = O\left(|V| + |V| + 2|E|\right) = O(|V| + |E|)$$