CS 170

Lecture 5

- use iClicker

Sanjam Garg.

join.iclicker.com/RGWR

Graphs:

$G = (V, E)$ $\quad |V| = n$ $\quad |E| = m < n^2$

Undirected Graphs

Directed Graphs

Connectivity?

Can we go from node $u \rightsquigarrow v$

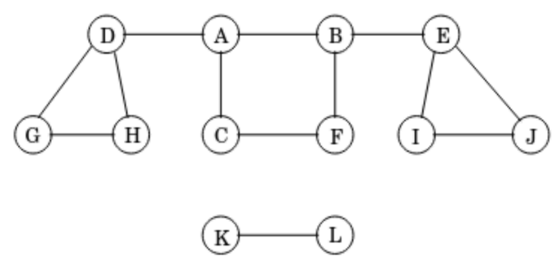
explore(G, v)

Input : $G = (V, E)$ & a vertex v

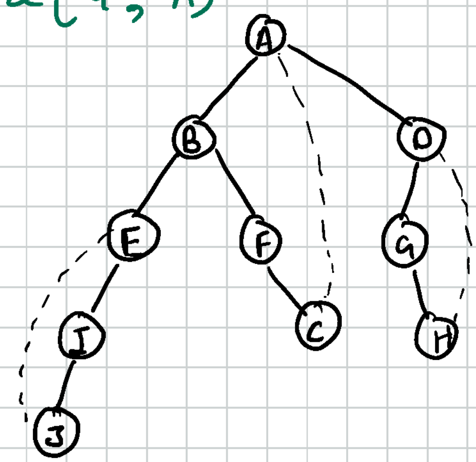
Output : visited(u) is set to T
if u is reachable from v

visited(v) = True

for each $(v, u) \in E$
  if not visited(u) then explore(u)



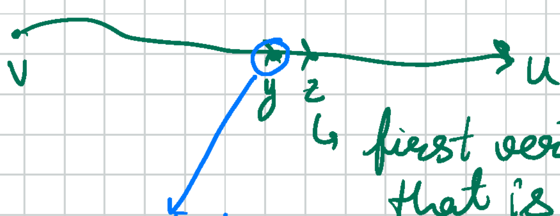
explore(G, A)



Theorem :# Explore(v) visits exactly the vertices u that have a path $v \rightsquigarrow u$ in G

* vertex u is visited then $\exists$ a path from $v \to u$

* $\exists$ a path $v \rightsquigarrow u$ then u is visited



↳ first vertex of this path that is not visited.

y is visited

→ Thus a contradiction

dFS(v)

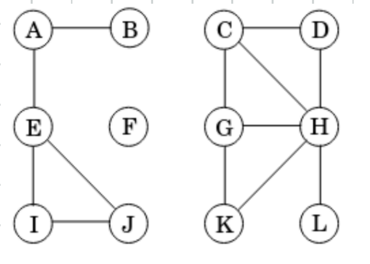
$\forall\ u \in V$

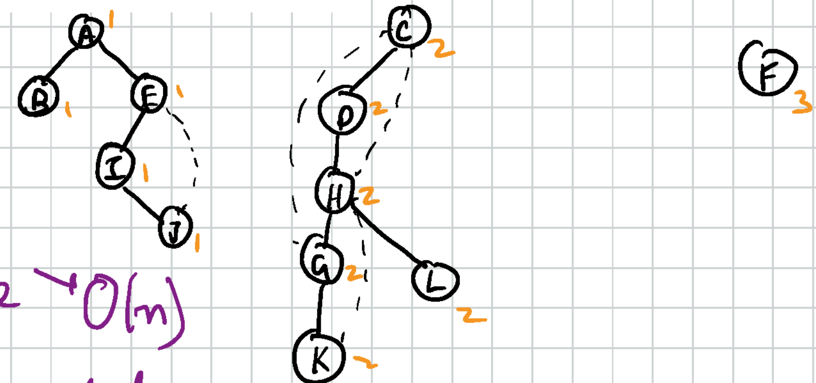
visited(u) = false

$\forall\ v \in V$

if not visited(v)

explore(v)





Runtime

1) explore on each vertex v is invoked exactly once $\rightarrow O(n)$

2) runtime of explore on input v

$O(1+\deg(v))$ $\rightarrow$ $\sum_v (1+\deg(v)) = O(n+2m) = O(n+m)$

or $O(|V|+|E|)$

How many connected components are there? Which connected component does a node v belong to?

explore(G, v)

Input : $G=(V,E)$ & a vertex v

Output : visited(u) is set to T if u is reachable from v

visited[v] = true

ccnum[v] = cc

$\forall\ (v,u) \in E$

if non visited(u)

explore(u)

dFS(G)

$\forall\ v \in V$ visited(v)=false

init ccnum[1...n] = $\perp$

cc = 1

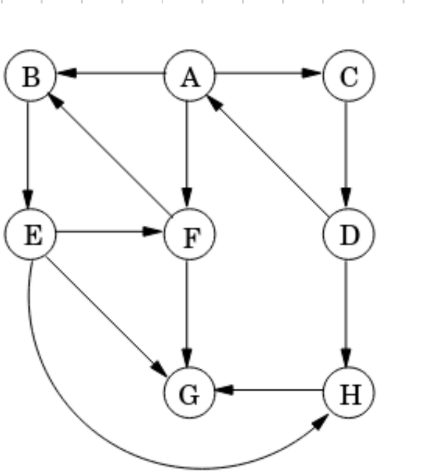
$\forall\ v \in V$

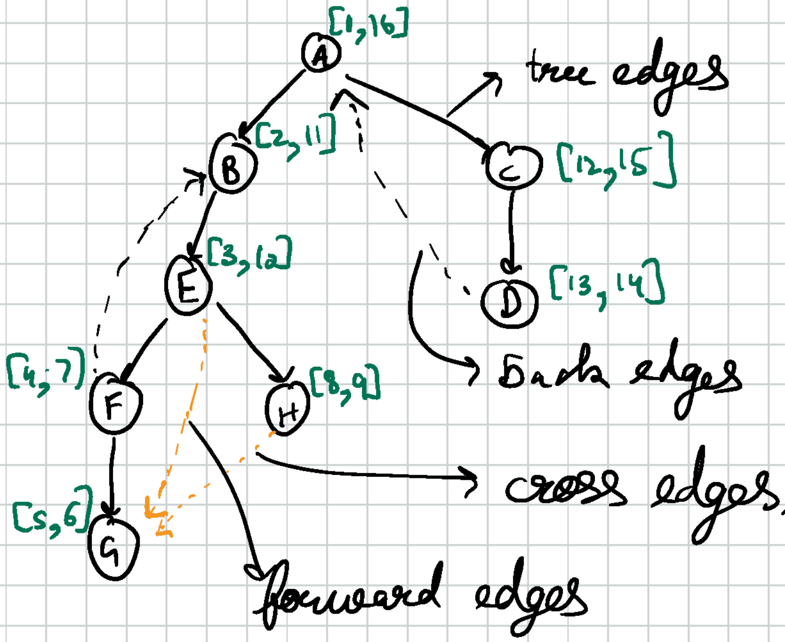
if not visited(v)

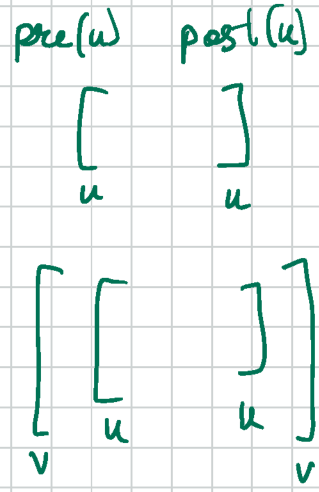
explore(v)

cc = cc+1

DFS over directed graphs







explore(G, v)

Input : G = (V, E) & a vertex v

Output : visited(u) is set to T

if u is reachable from v

visited[v] = true

previsit (v)

∀ (v, u) ∈ E

if non visited (u)

explore (u)

postvisit (v)

init clock = 1

previsit (v)

pre (v) = clock

clock = clock + 1

postvisit (v)

post (v) = clock

clock = clock + 1

Observation ①: $\forall\, u, v \in V$, the time interval $[pre(u), post(u)]$ and $[pre(v), post(v)]$ are such that

either (i) they are disjoint

or (ii) one is contained in the other.

Property ②: G has a cycle iff it has a back edge.

Proof: * $u \to v$ is a back edge.

$\to$ a cycle.

* $v_0 \to v_1 \to v_{i-1} \to v_i - - \to v_k$

$\hookrightarrow$ the first vertex to be visited.

Observation ③: (u, v) is a back edge iff $post(u) < post(v)$

$u \to v$

① $[_u \; [_v \quad]_v \;]_u$ $\longrightarrow$ tree edge or a forward edge

② $[_v \; [_u \quad]_u \;]_v$ $\longrightarrow$ back edge. $post(u) < post(v)$

③ $[_v \quad]_v \; [_u \quad]_u$ $\longrightarrow$ cross edge.

$post(v) < post(u)$

$[_u \quad]_u \; [_v \quad]_v$ $\longrightarrow$ not possible

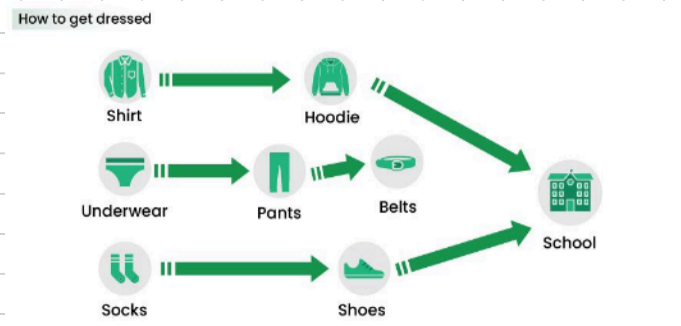
Can we test if a directed graph has a cycle?

Directed Acyclic Graphs (DAGs)

↳ Directed graphs with no cycles.

Examples

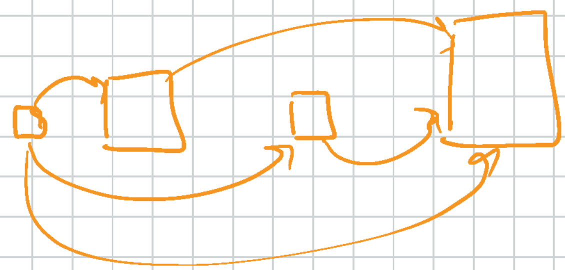
①



② course prerequisites

③ software compilation

④



Topological Sort (Linearization)

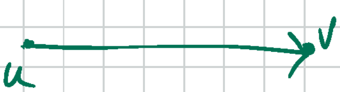
reducing post time

Input : DAG $G=(V,E)$

Output : An ordering of vertices so that all edges go left to right

① Naive: inefficient.

② observation $u \longrightarrow v$ $post(u) > post(v)$



u v

post(u) > post(v)

③ TS reduces to DFS.