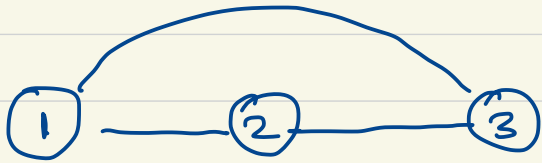


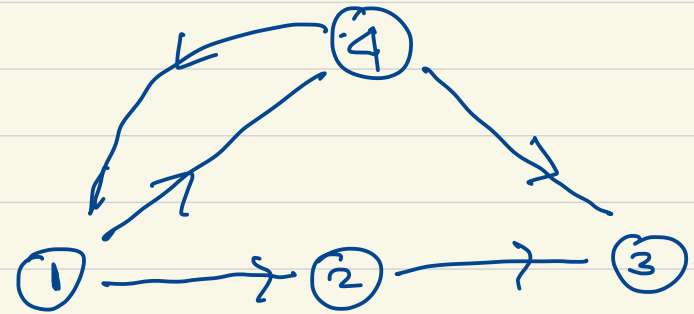
LECTURE 5

GRAPHS



$$G = (V, E)$$

(UNDIRECTED)



$$(u, v) \in E \text{ if } u \rightarrow v$$

(DIRECTED)

Parameters:

$$n = |V|$$
$$m = |E| \leq n^2$$

Example: Facebook friend graph

* $n \approx 2.91$ billion users

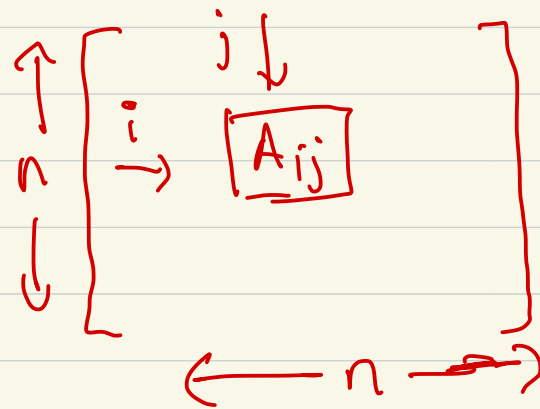
* avg user has 338 friends

$m \approx 500$ billion edges

Representing Graphs on Computers

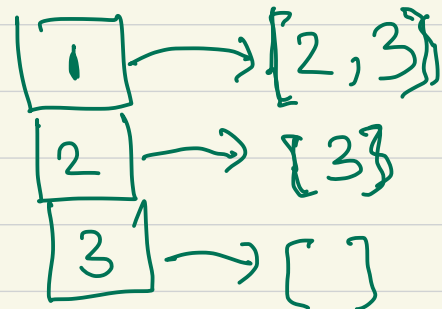
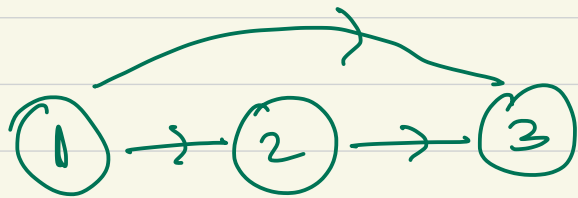
$$V = \{1, \dots, n\}$$

(1) Adjacency Matrix Representation

$$A_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$$


(2) Adjacency List Representation

= list of neighbors for every vertex



	Adjacency Matrix	Adjacency Lists
size	$\Theta(n^2)$ space	$\Theta(n+m)$ space
Is $(u,v) \in E$?	$\Theta(1)$ time	$\Theta(\deg(u))$ time
Enumerate all neighbors of u	$\Theta(n)$ time	$\Theta(\deg(u))$ time

Obs: For large sparse graphs, adjacency matrix is too space inefficient

Example: Facebook graph $n^2 = (2.91 \text{ billion})^2 = 8.5 \times 10^{18} = 6 \text{ million TB}$

CONNECTIVITY IN GRAPHS

* Is there a path from u to v ?

* Is G connected?

* What are connected components?

DEPTH FIRST SEARCH

$visited[u] = FALSE \quad \forall u \in W$

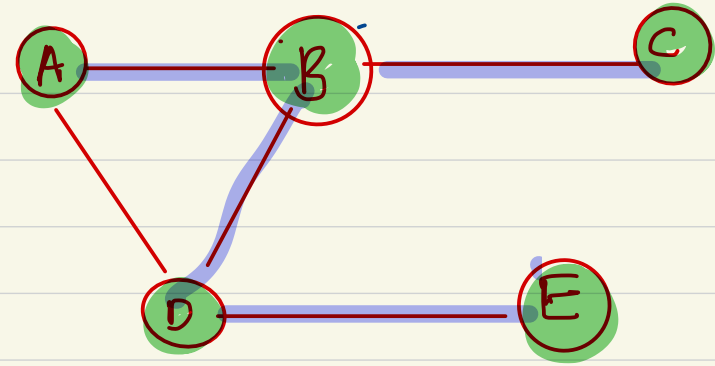
EXPLORE(vertex v)

$visited[v] = TRUE$

for each edge $(v, w) \in E$

if NOT $visited[w]$

EXPLORE(w)



DEPTH - FIRST SEARCH

$visited[u] = FALSE \quad \forall u \in W$

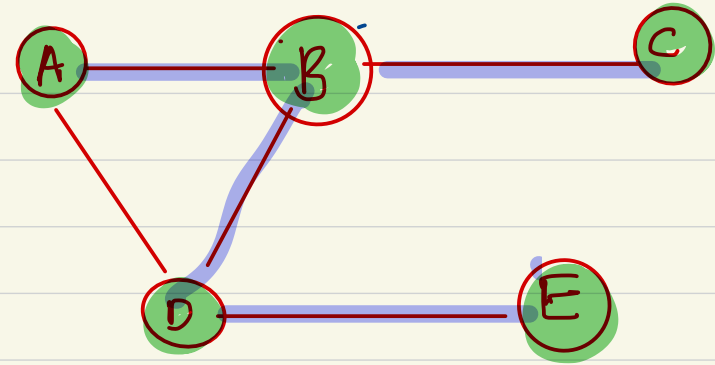
EXPLORE(vertex v)

$visited[v] = TRUE$

for each edge $(v, w) \in E$

if NOT visited $[w]$

EXPLORE(w)



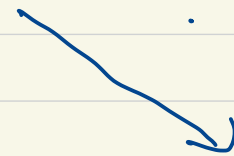
explore(A) (AB, AD)



explore(B) (BA, BC, BD)



explore(C) (CB)



explore(D) (DA, DB, DE)

DEPTH FIRST SEARCH

$visited[u] = FALSE \quad \forall u \in V$

EXPLORE(vertex v)

$visited[v] = TRUE$

for each edge $(v, w) \in E$

if NOT $visited[w]$

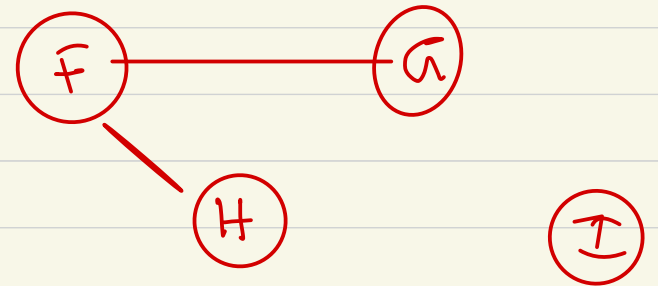
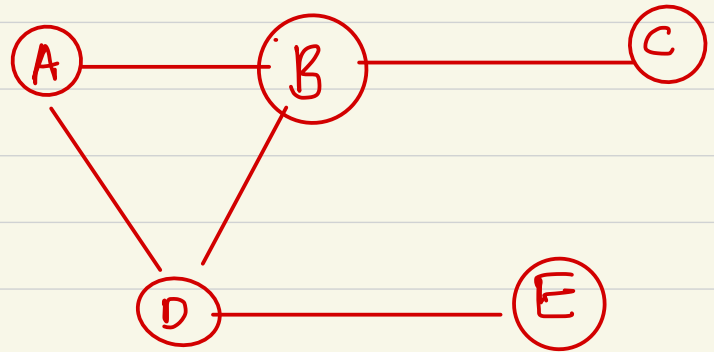
EXPLORE(w)

DFS (GRAPH G)

$visited[u] = FALSE \quad \forall u \in V$

for each vertex v

IF NOT $visited[v] \vdash EXPLORE(v)$



DEPTH FIRST SEARCH

$visited[u] = FALSE \quad \forall u \in V$

EXPLORE(vertex v)

$visited[v] = TRUE$

for each edge $(v, w) \in E$

if NOT $visited[w]$

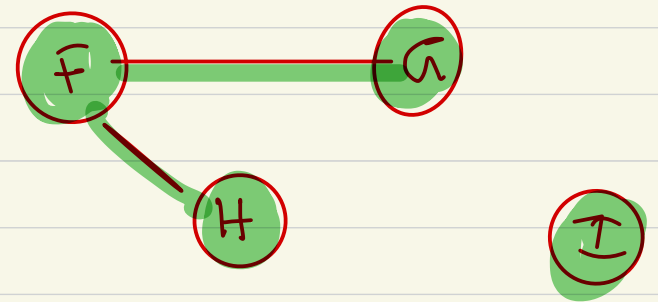
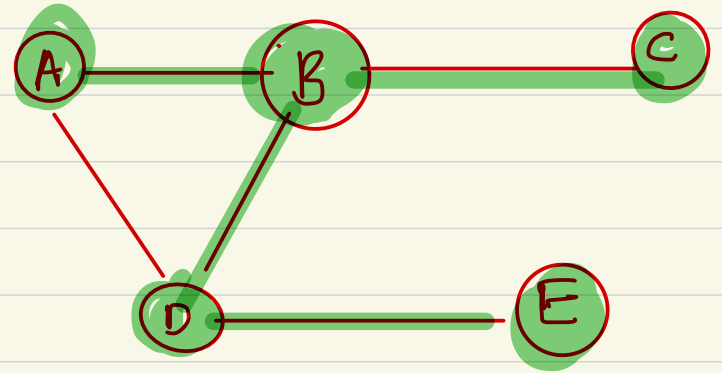
EXPLORE(w)

DFS (GRAPH G)

$visited[u] = FALSE \quad \forall u \in V$

for each vertex v

IF NOT $visited[v] : EXPLORE(v)$



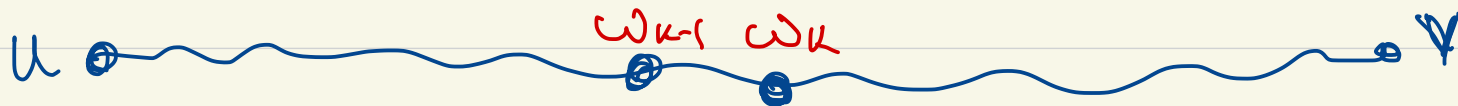
PROPERTY: $\text{Explore}(u)$ visits exactly the vertices v such that graph G has a path from u to v .

PROOF: 1) Vertex v is visited
 $\Rightarrow \exists$ a path from u to v .

2) $\exists$ a path from u to v
 $\Rightarrow$ Vertex v is visited

SUPPOSE NOT Assume v is NOT visited

Consider the path from $u \rightsquigarrow v$



let $w_k =$ first vertex along path P that

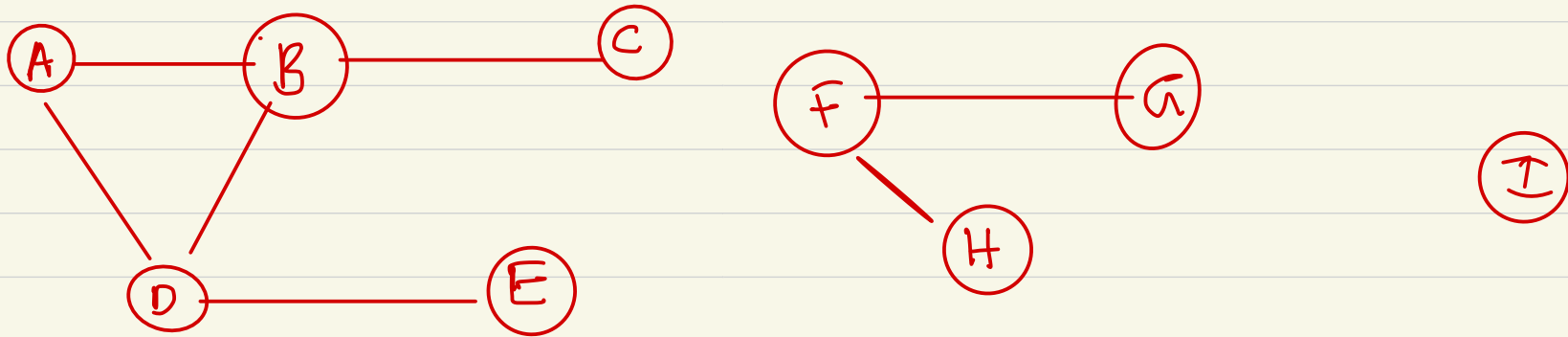
is NOT visited

w_{k-1} is visited (i.e. $\text{Explore}(w_{k-1})$ is called)

(w_{k-1}, w_k) is an edge $\Rightarrow \text{Explore}(w_k)$ is called

A CONTRADICTION!

CONNECTED COMPONENTS IN AN UNDIRECTED GRAPH



EXPLORE (vertex v)

visited[v] = TRUE
ccnum[v] = count

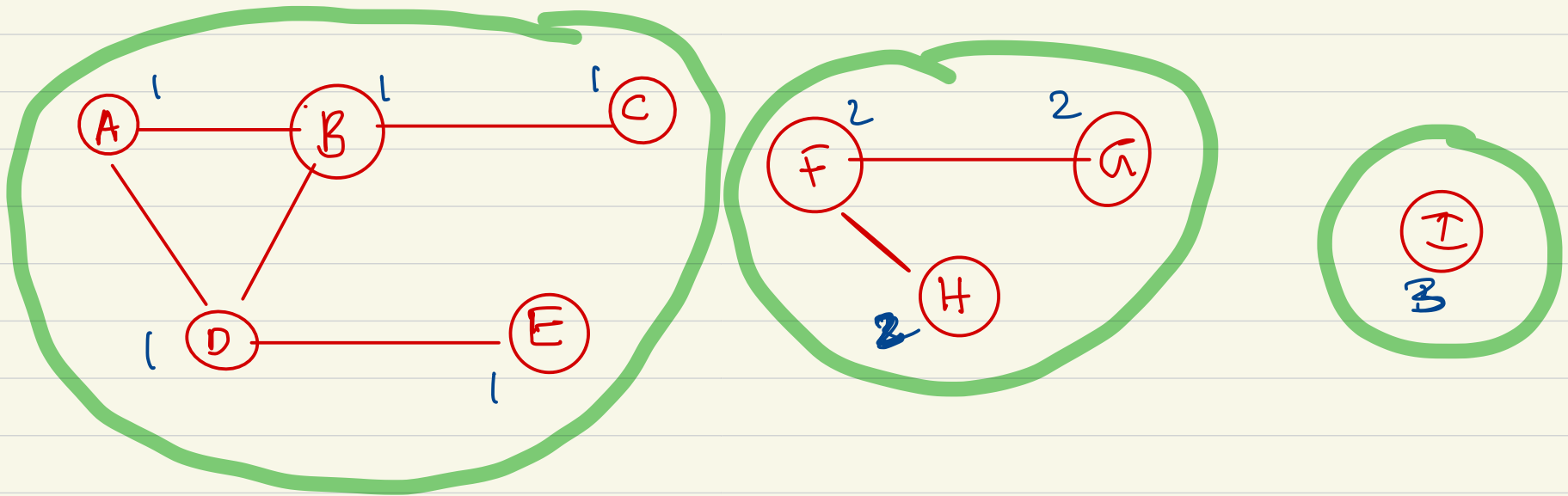
for each edge $(v, w) \in E$
if NOT visited[w]
EXPLORE(w)

DFS(Graph G)

visited[u] = FALSE $\forall u \in V$
count = 1
int array ccnum[n].

for $v \in V$
if NOT visited[v]
EXPLORE(v)
count = count + 1

CONNECTED COMPONENTS IN AN UNDIRECTED GRAPH



EXPLORE (vertex v)

visited [v] = TRUE
ccnum [v] = count

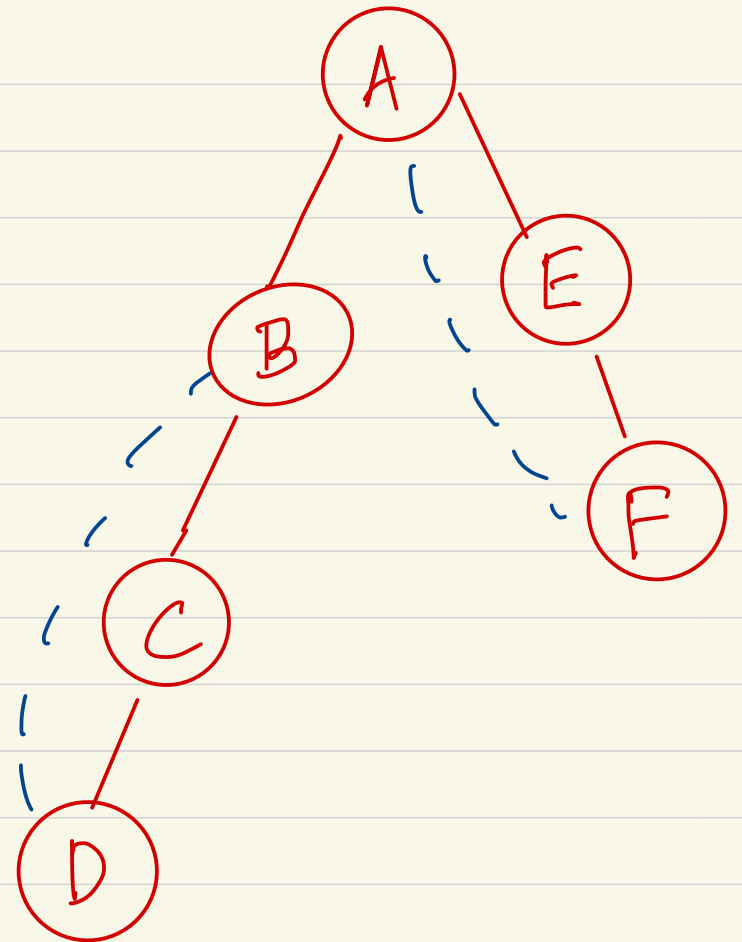
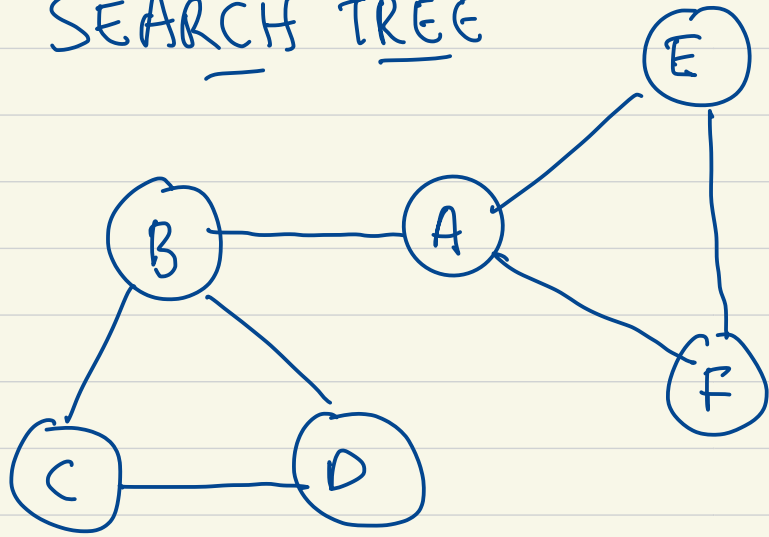
for each edge $(v, w) \in E$
if NOT visited [w]
EXPLORE (w)

DFS (Graph G)

visited [u] = FALSE $\forall u \in V$
count = 1
int array ccnum [n].

for $v \in V$
if NOT visited [v]
EXPLORE (v)
count = count + 1

DFS SEARCH TREE



RUNTIME OF DFS

1) Only call $\text{EXPLORE}(v)$ once per vertex

2) Runtime of $\text{EXPLORE}(v)$

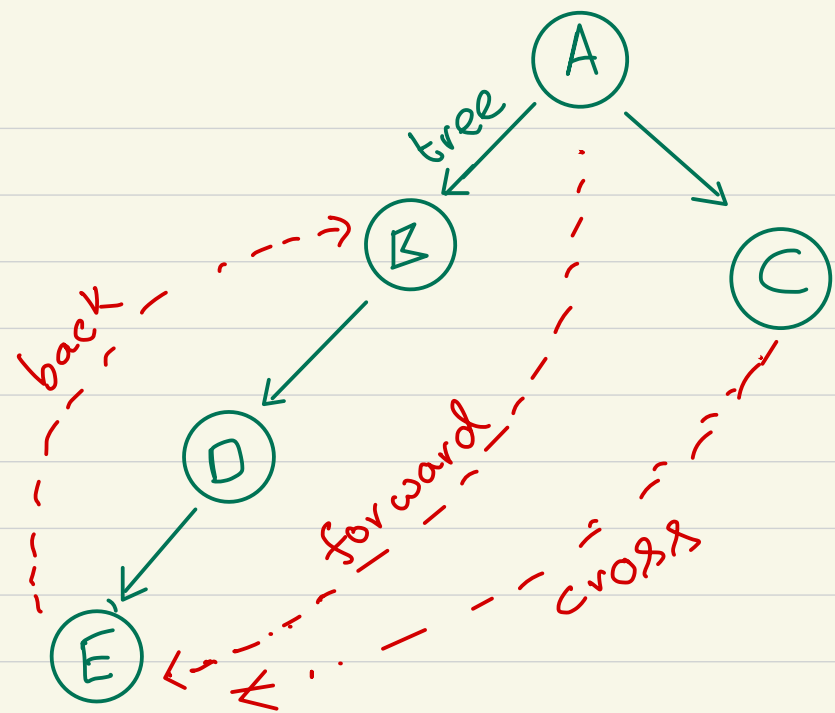
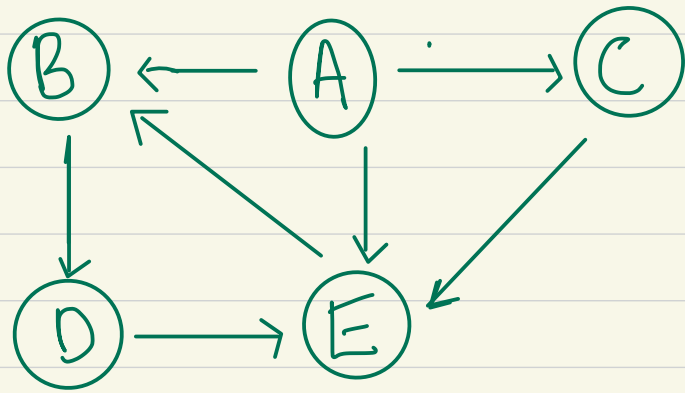
* Set $\text{visited}[v] = \text{TRUE}$ $O(1)$ time

* enumerate v 's neighbors

$O(\deg(v))$ time

$$\text{total} = \sum_v O(1 + \deg(v)) = O(n + m) \text{ time}$$

DFS in DIRECTED GRAPHS



EXPLORE (vertex v)

$visited[v] = TRUE$

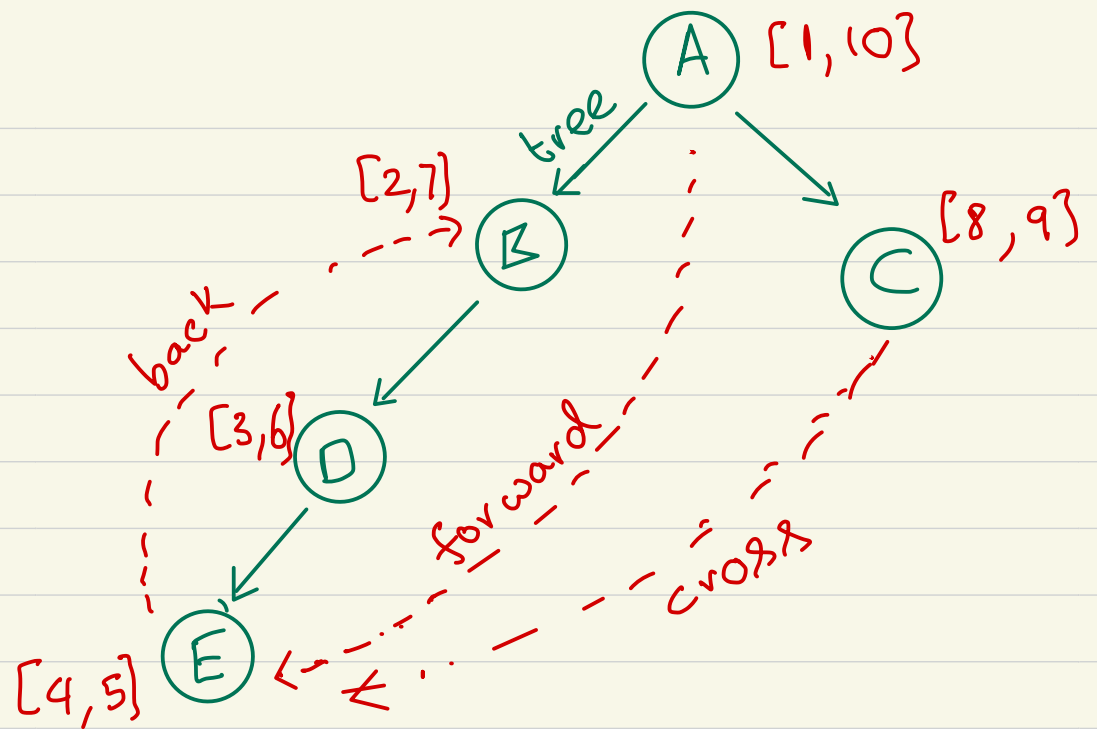
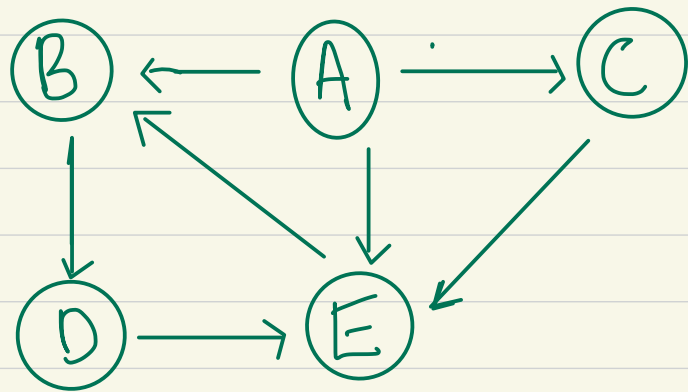
for each edge $(v, w) \in E$
if NOT $visited[w]$
EXPLORE(w)

DFS(Graph G)

$visited[u] = FALSE \quad \forall u \in V$

for $v \in V$
if NOT $visited[v]$
EXPLORE(v)

DFS in DIRECTED GRAPHS



EXPLORE (vertex v)

$visited[v] = TRUE$

$pre[v] = clock$

$clock = clock + 1$

for each edge $(v, w) \in E$
if NOT $visited[w]$
EXPLORE(w)

$post[v] = clock$

$clock = clock + 1$

DFS(Graph G)

$visited[u] = FALSE \forall u \in V$

$clock = 0$

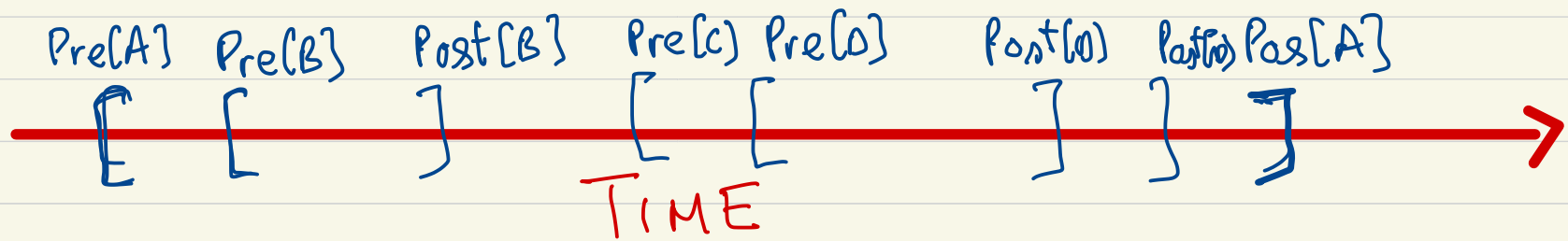
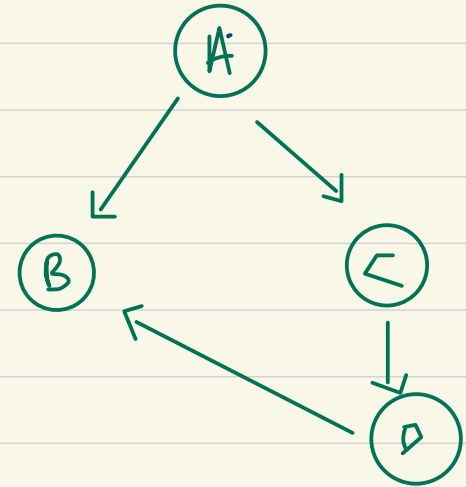
int array $pre[n]$ $post[n]$

for $v \in V$

if NOT $visited[v]$

EXPLORE(v)

PRE & POST NUMBERS



Edge: $u \rightarrow v$

$\left[\left[\begin{array}{c} u \\ v \end{array} \right] \right]_v = \text{Tree OR Forward Edge}$

$\left[\left[\begin{array}{c} u \\ v \end{array} \right] \right]_u = \text{Back Edge}$

$\left[\begin{array}{c} u \\ u \end{array} \right] \left[\begin{array}{c} v \\ v \end{array} \right] = \text{IMPOSSIBLE}$

$$\left[\begin{array}{c} \text{ } \\ v \end{array} \right] \left[\begin{array}{c} \text{ } \\ v \end{array} \right] \left[\begin{array}{c} \text{ } \\ u \end{array} \right] \left[\begin{array}{c} \text{ } \\ u \end{array} \right] = \text{CROSS EDGE}$$

$$\left[\begin{array}{c} \text{ } \\ u \end{array} \right] \left[\begin{array}{c} \text{ } \\ v \end{array} \right] \left[\begin{array}{c} \text{ } \\ u \end{array} \right] \left[\begin{array}{c} \text{ } \\ v \end{array} \right] = \text{IMPOSSIBLE}$$

Observation: For all edges $u \rightarrow v$

$\text{post}[u] < \text{post}[v]$ if and only if

$u \rightarrow v$ is a back edge

DIRECTED ACYCLIC GRAPHS

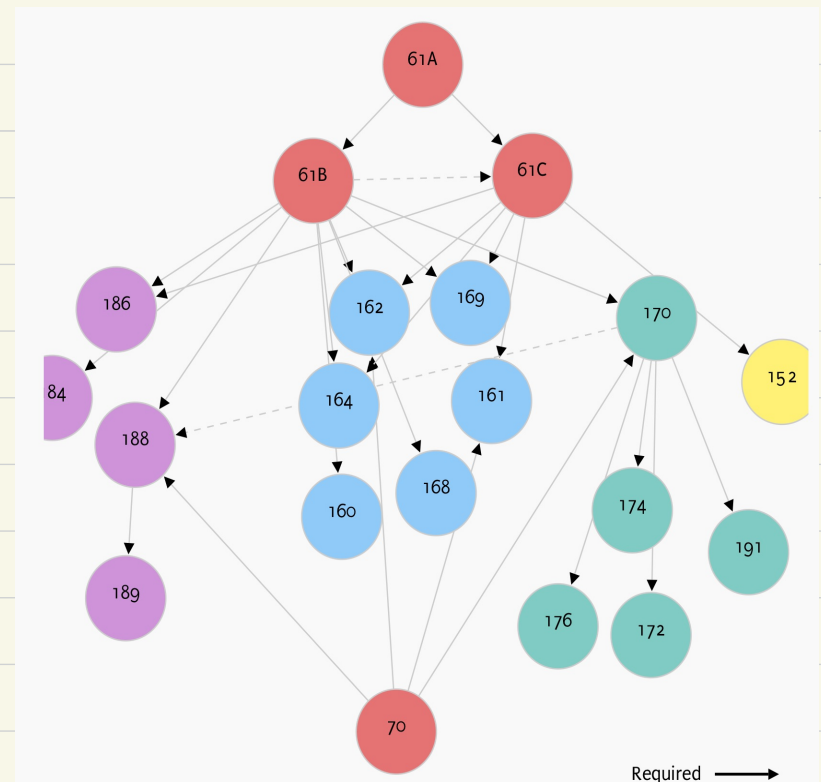
"Directed graphs with NO directed cycles"

Application:

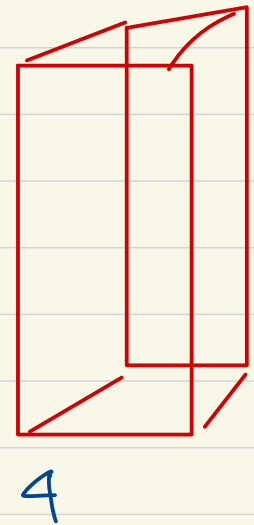
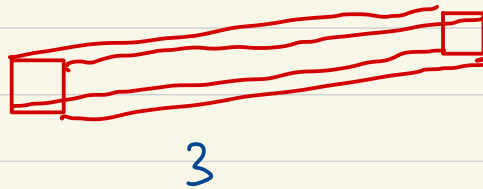
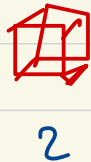
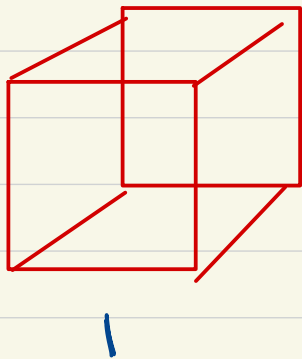
1) Modelling Dependencies / Pre-requisites

Example 1: Course prerequisites

Example 2: Source files for
Compilation



APPLICATION 2: Partially Ordered Sets.



Box $A < \text{Box } B$
if A fits inside B .

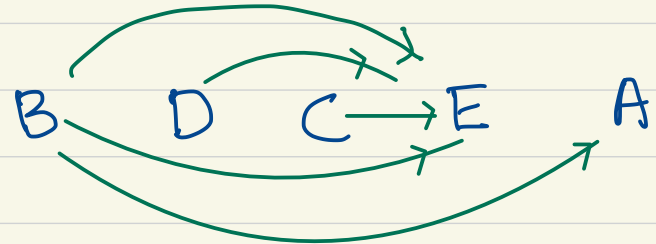
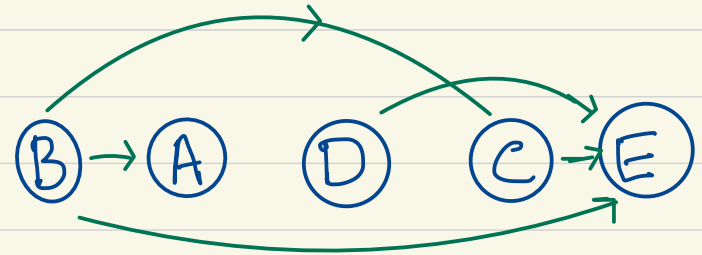
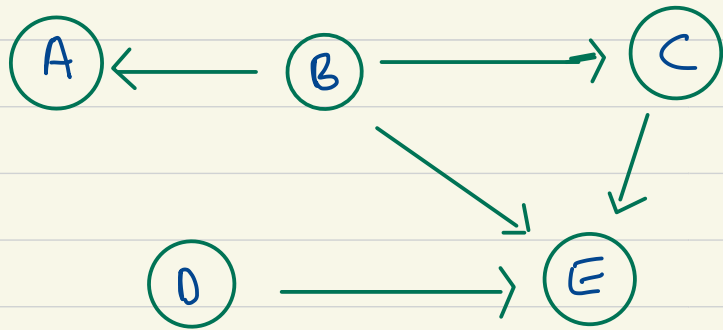
A partial order on $\{a_1, \dots, a_n\}$ is a relation $\preceq$,
where if we create a directed graph
 $a_i \rightarrow a_j$ whenever $a_i \preceq a_j$
then it's a DAG.

TOPOLOGICAL SORT [LINEARIZATION]

INPUT : A directed acyclic graph $G = (V, E)$

OUTPUT : An ordering of vertices so ALL edges
go from LEFT to RIGHT

[a.k.a. "LINEARIZE" the DAG]



⋮

TOPOLOGICAL SORT

INPUT : A directed acyclic graph $G = (V, E)$

OUTPUT : An ordering of vertices so ALL edges
go from LEFT to RIGHT

ALGORITHM:

1) Run DFS to compute pre & post values

2) Sort vertices in decreasing post values

PROOF OF CORRECTNESS:

1) By Observation earlier

$\text{post}[u] < \text{post}[v]$ if and only if $u \rightarrow v$
is a back edge

2) DAG has NO back edges

$\Rightarrow \forall \text{ edges } u \rightarrow v \quad \text{post}[u] > \text{post}[v]$

$\Rightarrow$ All edges are from LEFT to RIGHT when
vertices are ordered in decreasing post values