Lecture 10 - Oct 6

Graphs

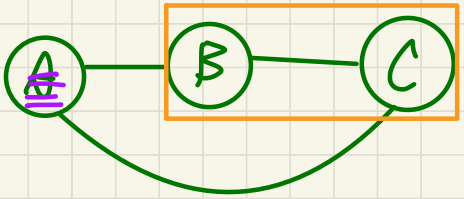
Forest vs. Tree vs. Spanning Tree
Graph Traversal: Depth-First Search (DFS)
DFS on a Tree vs. Pre-Order Traversal

Simple cycle:

A closed path that starts and ends at the same vertex and does not repeat any vertex or edge except for the starting/ending vertex.

simple path.

Example: A-B-C-A



Non-simple cycle:

A closed path that starts and ends at the same vertex but may repeat vertices or edges along the way.

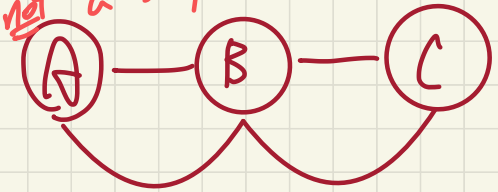
repeats.

Example: A-B-C-B-A

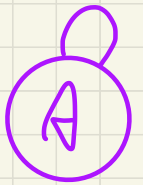
cycle

not a simple path

not a simple cycle.



Self Loop

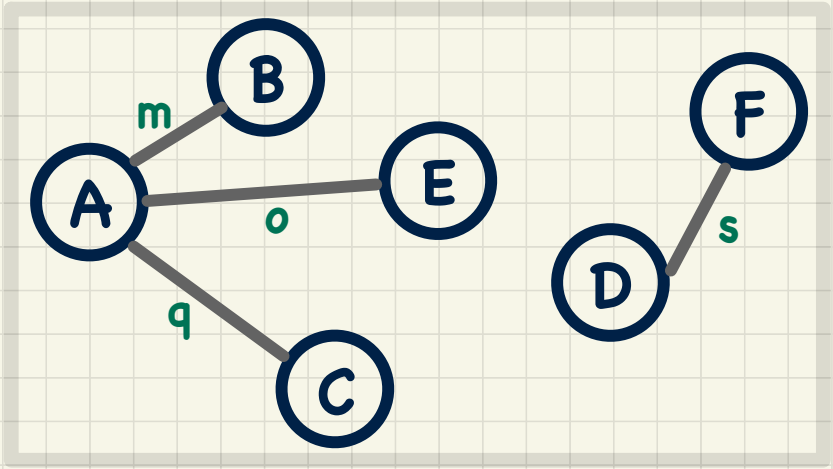


empty path

simple path

cycle

Graph: Forests and Trees

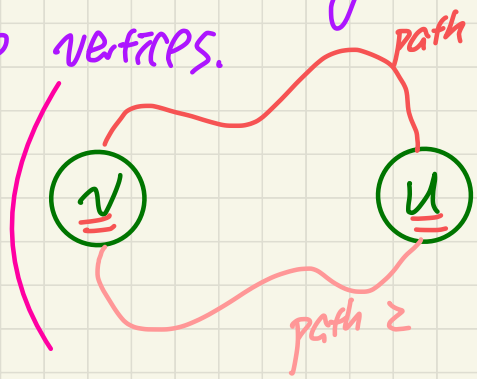


- undirected
- no cycle (acyclic)

Any two vertices v, u are connected via * at most one path.

≤ 1 | > 1

What if ≥ 2 edges connecting two vertices.



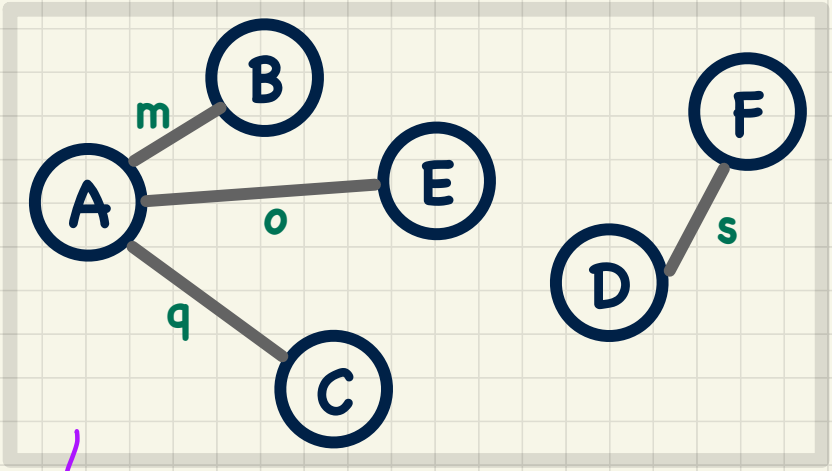
there's a cycle from v to v.

* Special case:
if u and v are connected by edge $\rightsquigarrow$ the graph is not connected.

special case of ≤ 1 (0)

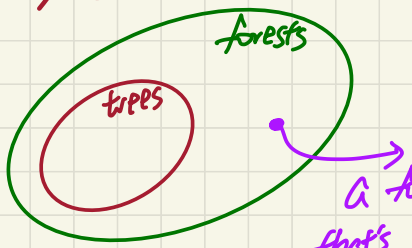
A forest may or may not be connected.

Graph: Forests and Trees

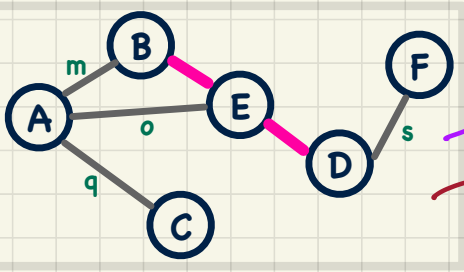


↳ forest but not a tree.

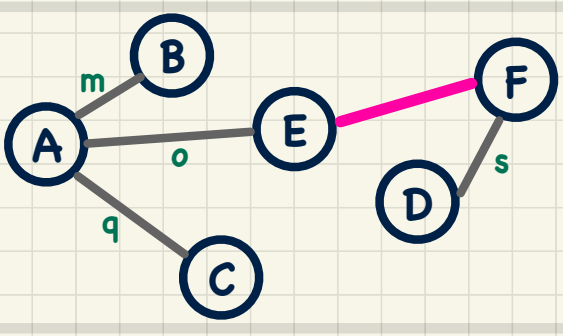
a connected forest.



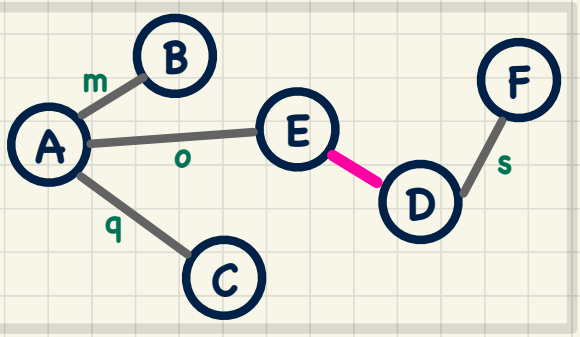
a forest that's not connected.



↳ connected
↳ cyclic
↳ x forest
↳ x tree.



tree!



tree.

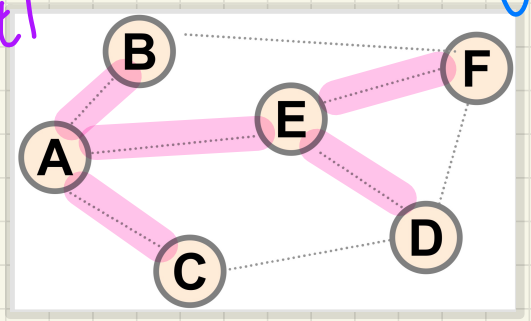
Graph: Spanning Trees

G is a spanning tree $\Rightarrow |E| = |V|-1$

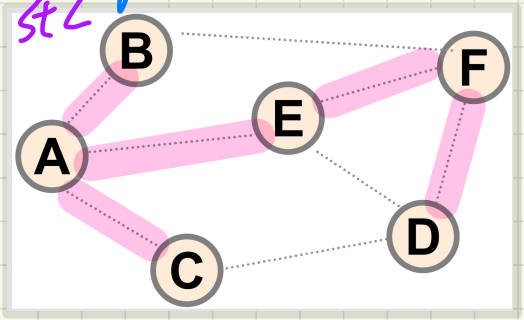
↳ a connected spanning subgraph that has no cycle

↳ a spanning subgraph that is also a tree.

st1



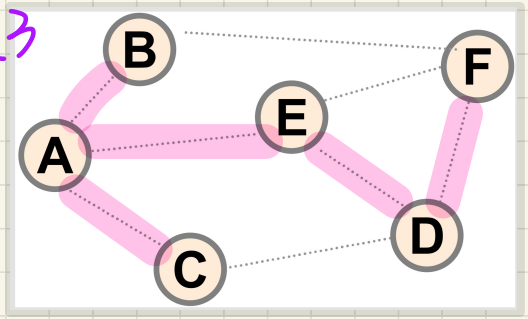
st2



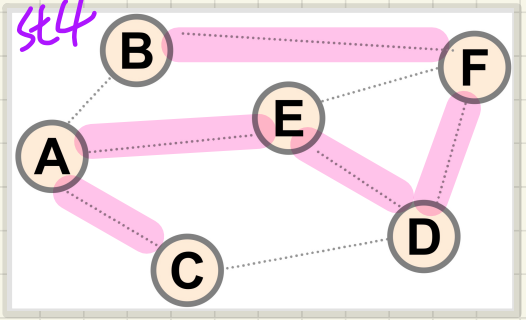
$|V| = 6$

$|E| = 5 \ (= |V|-1)$

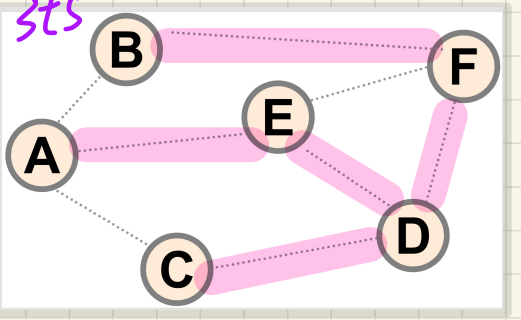
st3



st4



st5

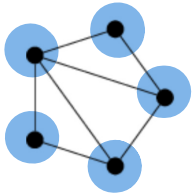


Summary of Terms

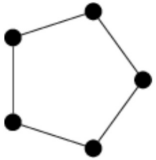
	subgraph	spanning subgraph	forest	tree	spanning tree
undirected?	✓	✓	✓	✓	✓
acyclic?	? maybe maybe not	?	✓	✓	✓
connected?	?	?	?	✓	✓
spanning?	?	✓	?	?	✓

Graph: Exercises

Given a graph

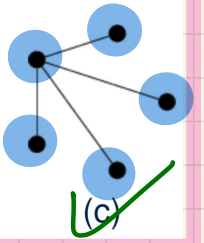


Which one of the following is a ***spanning tree***?



(a)

(b)



(c)

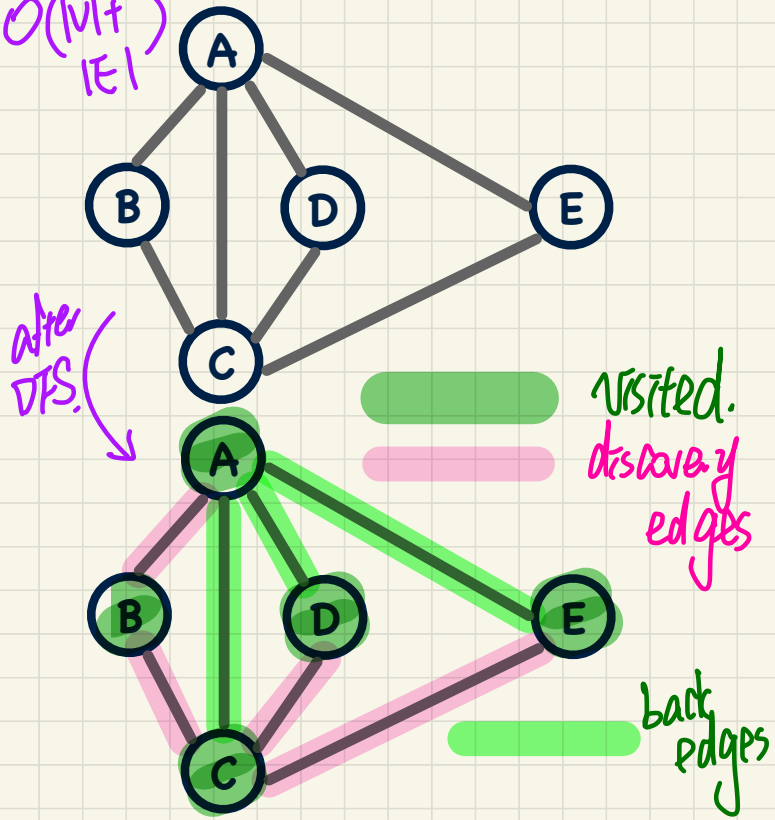
spanning but cyclic.

acyclic but not spanning.

Graph Traversals: Definition & Applications

Efficient Traversal of Graph G:

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



Applications: (visit later).

- **Reachable** Vertices from $v \in V$
- A **path** between $\{u, v\} \subseteq V$
- The **minimum path** between $\{u, v\} \subseteq V$
- Is G **connected**?
- Compute a **spanning tree** of a connected G.
- Compute the **connected components** of G.
- If G is cyclic, return a **cycle**.

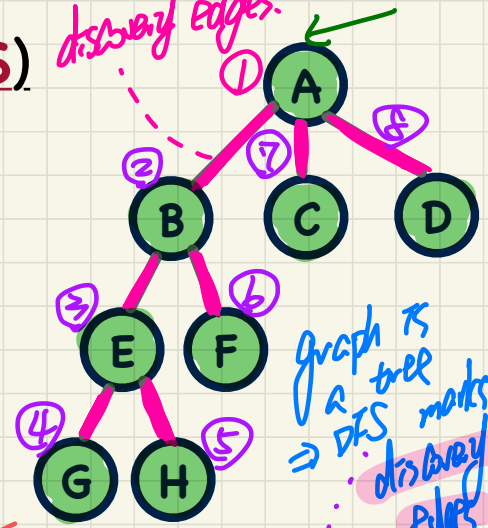
Graph Traversal: Depth-First Search (DFS)

arbitrary.

A ***Depth-First Search*** (***DFS***) of graph G = (V, E), starting from some vertex $v \in V$, proceeds along a **path** from v.

- The **path** is constructed by following **an** ***incident edge***.
- The **path** is extended **as far as possible** until **all** ***incident edges*** lead to vertices that have already been ***visited***.
- Once the **path** originated from v **cannot be extended further**, ***backtrack*** to the **latest** vertex whose ***incident edges*** lead to some ***unvisited*** vertices.

e.g. A



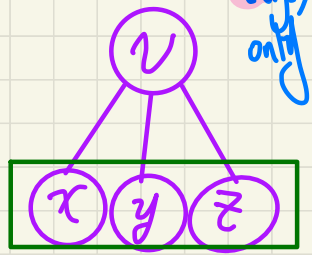
Assumption: iterate through neighbours alphabetically.

Q. When a **graph** is a **tree**, what kind of **tree traversal** does it correspond to?

pre-order (parent first, children next).

Q. What data structure should be used to keep track of the visited nodes?

↓ stack. (LIFO).



say all these neighbours have been visited.
↳ back track!

Depth-First Search (DFS): Marking Vertices & Edges

Before the ***DFS*** starts:

- All vertices are ***unvisited***.
- All edges are **unexplored/unmarked**.

Over the course of a ***DFS***, we **mark** vertices and edges:

- A vertex *v* is marked ***visited*** when it is **first** encountered.
- Then, we iterate through each of *v*'s **incident edges**, say *e*:
  - If edge *e* is already **marked**, then skip it.
  - Otherwise, mark edge *e* as:
    - A ***discovery*** edge if it leads to an ***unvisited*** vertex
    - A ***back*** edge if it leads to a ***visited*** vertex (i.e., an ancestor vertex)

the graph is cyclic

from C to A leading to vertex A that's already visited

back edge