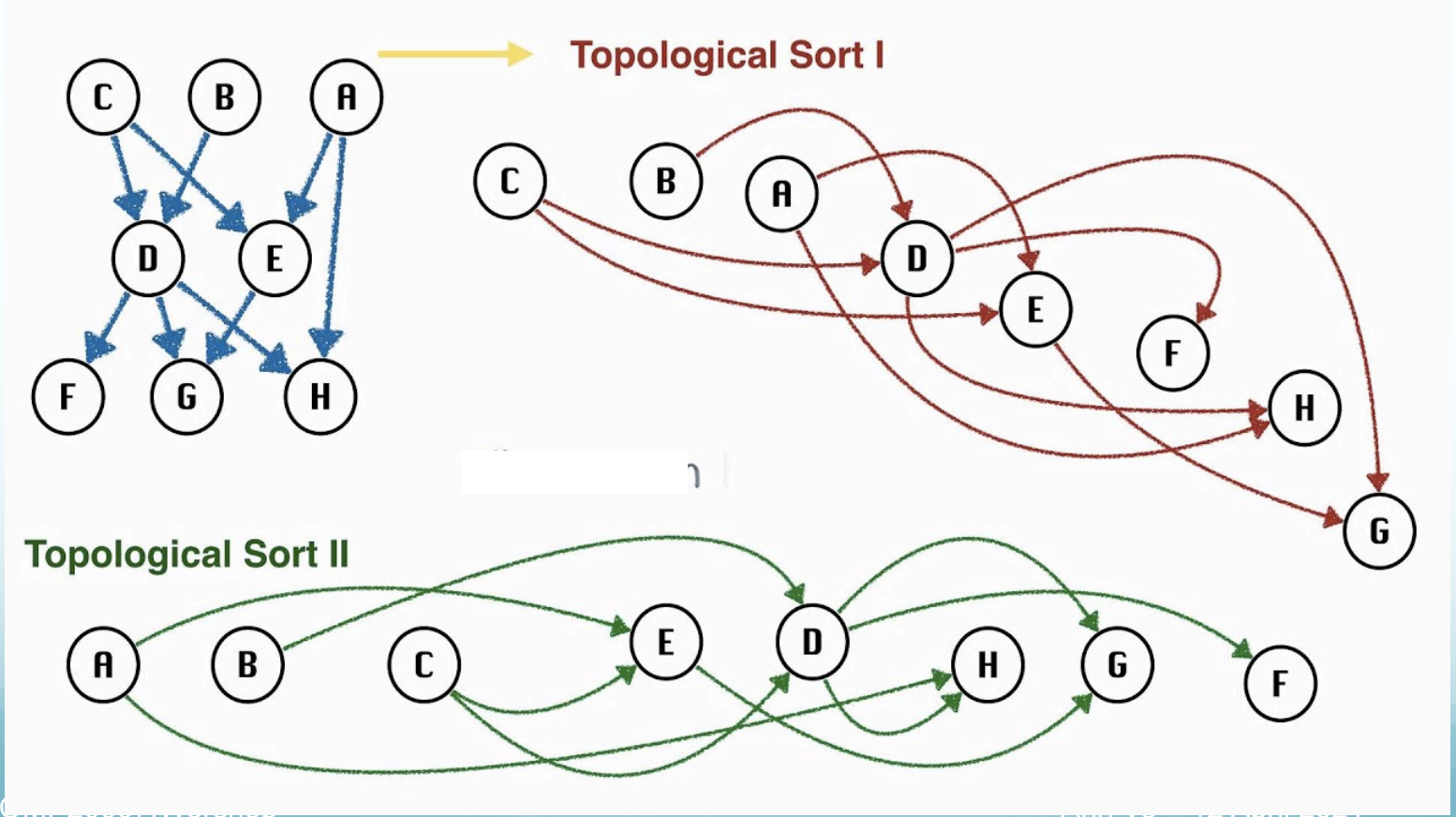
COMP20007 Workshop Week 7

	Preparation: download ppt (or pdf) from `github.com/anhvir/c207`
1	**Topic 1:** Topological Sorting ***Group Work:*** Problems T1 (toposort)
2	**Topic 2:** Binary Trees & BST ***Group Work:*** Problems T3, T4, T5
	Lab: • Finish ass1 if not yet done, or • Finish lab works of previous weeks

Topological Sorting (for DAG only!)

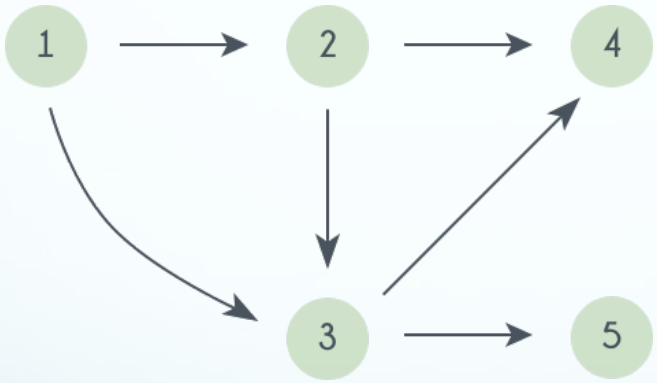
A *topological ordering:* sorting the nodes of the graph such that all edges point in one direction, to nodes later in the ordering.



Topological Sorting

Method 1: Select a source (node with no incoming edges), then remove this source and all of its incidents (ie. its outgoing edges). Repeat this process until all nodes have been selected.

What's the complexity of this algorithm if using adjacency matrix? list?



Complexity for

Adjacency matrix:

$O()?$ $\theta()?$

Adjacency list:

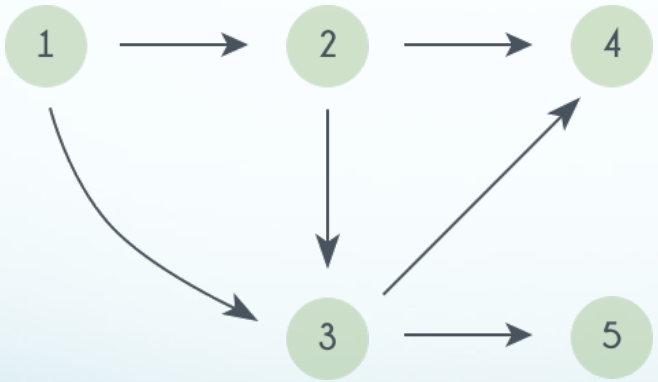
$O()?$ $\theta()?$

Topological Sorting

Another Algorithm (discussed in lectures) involves running a DFS on the DAG and keeping track of the order in which the vertices are popped from the stack. The topological ordering will be the reverse of this order.

What's the complexity if using adjacency lists for graphs? adj matrix?

How to have the pop-order?



Complexity for

Adjacency matrix:

O( ) ? θ( ) ?

why?

Adjacency list:

O( ) ? θ( ) ?

why?

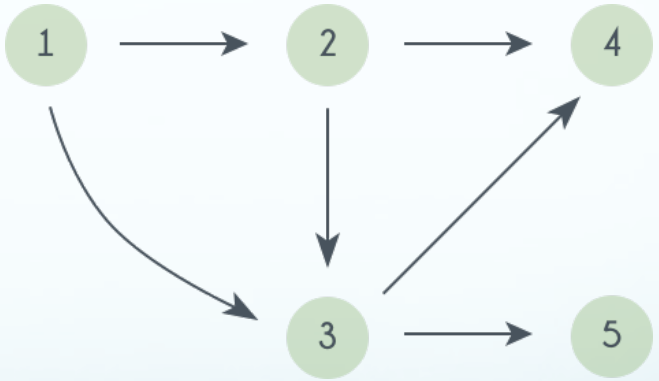
DFS: push- and pop-order (pre- and post-order)

```
function Dfs(<V,E>)
  mark each u in V as unvisited
  for each u in V do
    if u is unvisited then
      BfsExplore(u)
```

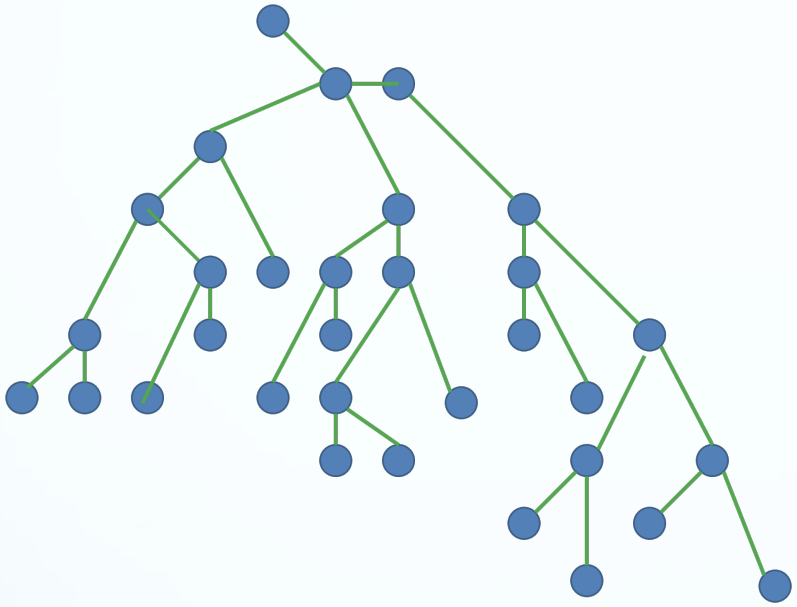
```
function DfsExplore(u)
  mark u as visited
  for each edge (u,v) do
    if v is unvisited then
      DfsExplore(v)
```

Problem:

- *For the graph below, write the push and pop order for DFS, starting from node 1*
- *Modify the DFS algorithm so that it also builds the arrays* `pre[V]` *and* `post[V]` *to store the push- and the pop-order of the vertices.*



Trees as Special Graphs



If a graph (V,E) is a tree, then:

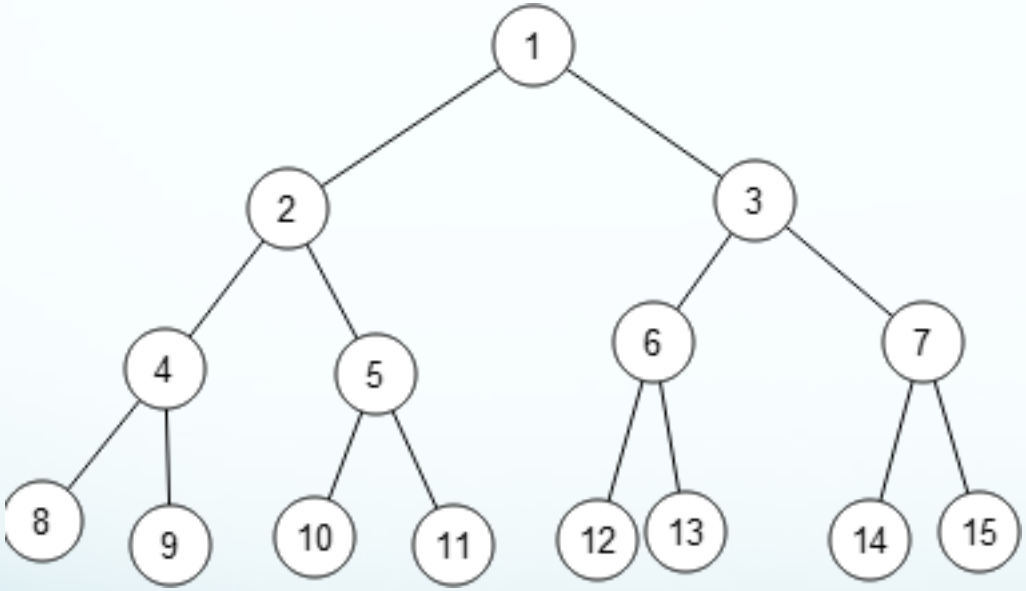
- digraph?
- cyclic?
- connected?
- sparse/dense?
- relationship between n=|V| and m=|E| :

- best representation?

Binary Tree: Recursive Definition

Binary tree traversal

What is *inorder*, *preorder*, *postorder,* and *level-order* traversal? Are they BFS or DFS?

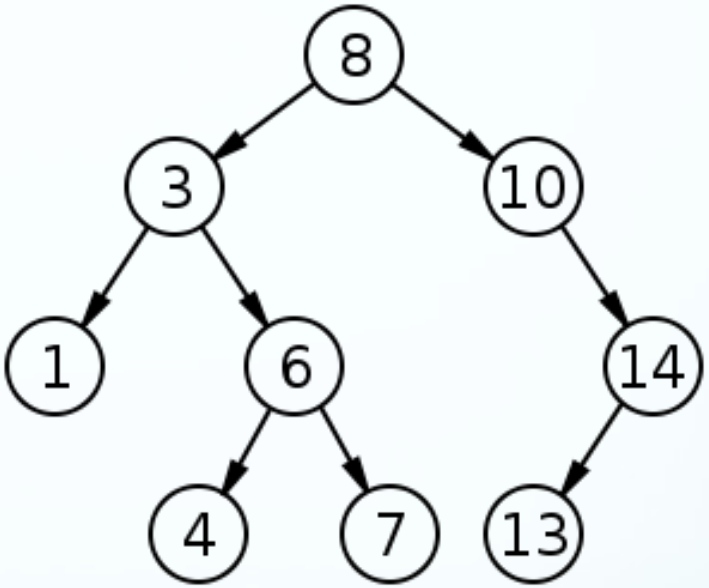


Group Work: Problems T3, T4, T5.

Binary (Search) Tree

How to:

1. print the number in increasing order?
2. print in decreasing order?
3. copy the tree?
4. free the tree?

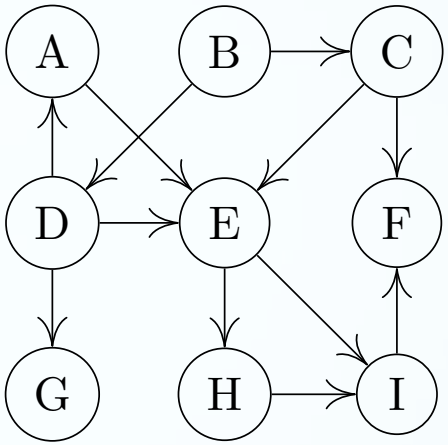


Your notes:

1.
2.
3.
4.

Group Work: problem 1

T1: *Finding a topological order for the graph by running a DFS.*



YOUR ANSWER:

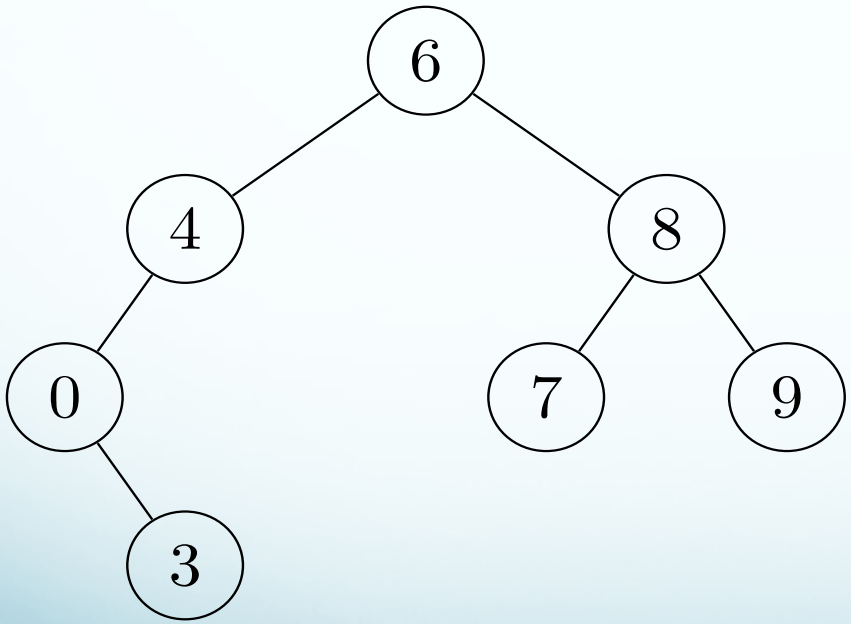
note: it would be better first to write down the operations with and the content of stack like:

operation	content	operation	content
init stack	$		
push A	$A		

(you can also just draw graph and write down push and pop order on the left and right of each node)

T3: conventional traversal

Write the *inorder*, *preorder* and *postorder* traversals of the following binary tree:



YOUR ANSWER:

In-order:

Pre-order:

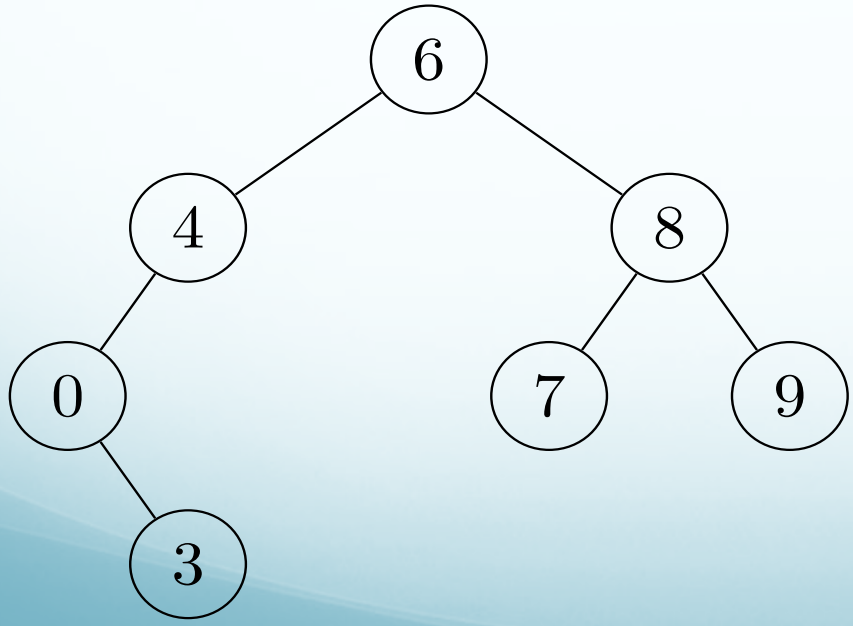
Post-order:

T4: level-order traversal

Level-order: visit level-by-level, left-to-right, starting from the root (which is in 0-th level).

a) For the tree below, what's the visited order?

b) Write the level-order pseudo-code.



YOUR ANSWER:

Level-order:

T4: level-order traversal

Level-order: visit level-by-level, left-to-right, starting from the root (which is in 0-th level).

a) For the tree below, what's the visited order?

b) Write the level-order pseudo-code.

YOUR ANSWER: The pseudocode:

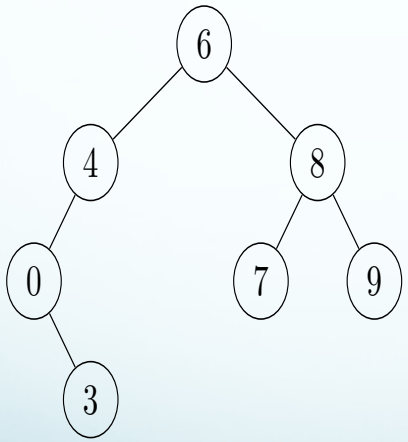
```
function LevelOrder( T )
```

T5: Binary Tree Sum

Write a recursive algorithm to calculate the sum of a binary tree where each node contains a number.

YOUR ANSWER: The pseudocode:

```
function Sum( T )
```



LAB

Assignment 1:

- check & make sure that you can submit **right now**
- ask questions
- note that you can discuss general problems with your friends but please do not reveal or show your solution and code

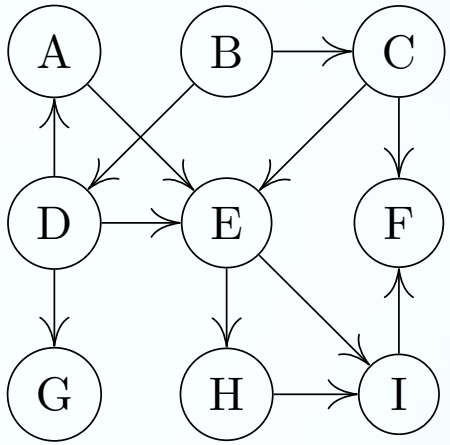
OR:

- do not-yet-done problems (if any) of previous workshops/lab/lectures

Group Work: moved to Week 7

T2: *Dijkstra's algorithm, unmodified, can't handle some graphs with negative edge weights. Your friend has come up with a modified algorithm for finding shortest paths in a graph with negative edge weights:*

1. Find the largest negative edge weight, call this weight **−w**.
2. Add **w** to the weight of all edges in the graph. Now, all edges have non-negative weights.
3. Run Dijkstra's algorithm on the resulting non-negative-edge-weighted graph.
4. For each path found by Dijkstra's algorithm, compute its true cost by subtracting **w** from the weight of each of its edges.



a) Give an example showing that DA can't handle negative weights.

b) Will your friend's algorithm work? Give an example.