

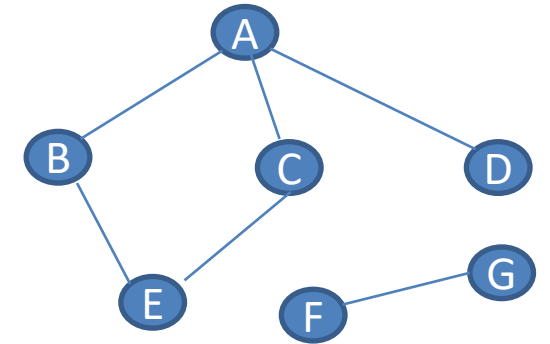
COMP20007 Workshop Week 6

1	Topic 1: More on DFS; Topological Sorting <i>Exercise:</i> Q6.1 (toposort)
2	Graph-related problems & issues
3	Topic 2: Binary Trees & BST <i>Group/Individual Exercises:</i> Q 6.2, 6.4, 6.3 Lab: <ul style="list-style-type: none">• Implement BST• on request: understanding BFS and Prim's, Dijkstra's, other exercises from Week 5 Workshop

DFS revisited: stack mechanism of recursion, time complexity

```
function DFS( $G=(V,E)$ )  
  for each  $v$  in  $V$  do  
    mark  $v$  with 0  
  for each  $v$  in  $V$  do  
    if  $v$  is marked with 0 then  
      DfsExplore( $v$ )
```

```
function DfsExplore( $v$ )  
  // implicit push( $v$ ) at the start  
  // start visiting  $v$   
  mark  $v$  with 1  
  for each edge  $(v,w)$  in  $E$  do  
    if  $w$  is marked with 0 then  
      DfsExplore( $w$ )  
  // end visiting  $v$   
  // implicit pop out  $v$  at the end
```



Work out Time Complexity

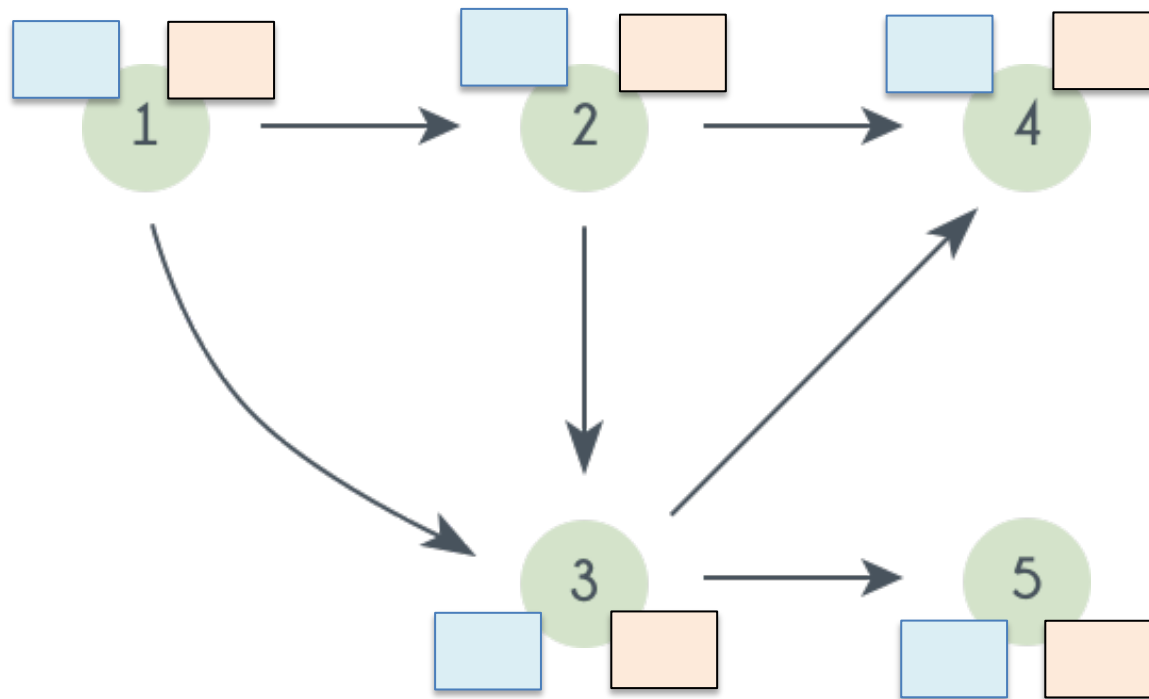
- for adjacency lists
- for adjacency matrix

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

Problem: For the graph below, write the push and pop order for DFS, starting from node 1

Method 1: Fill in the timestamp in yellow boxes for push-orders, pink boxes for pop-orders.

Method 2: Show the stack content and operations.



The PUSH and POP, using \$ to denote the bottom of stacks.

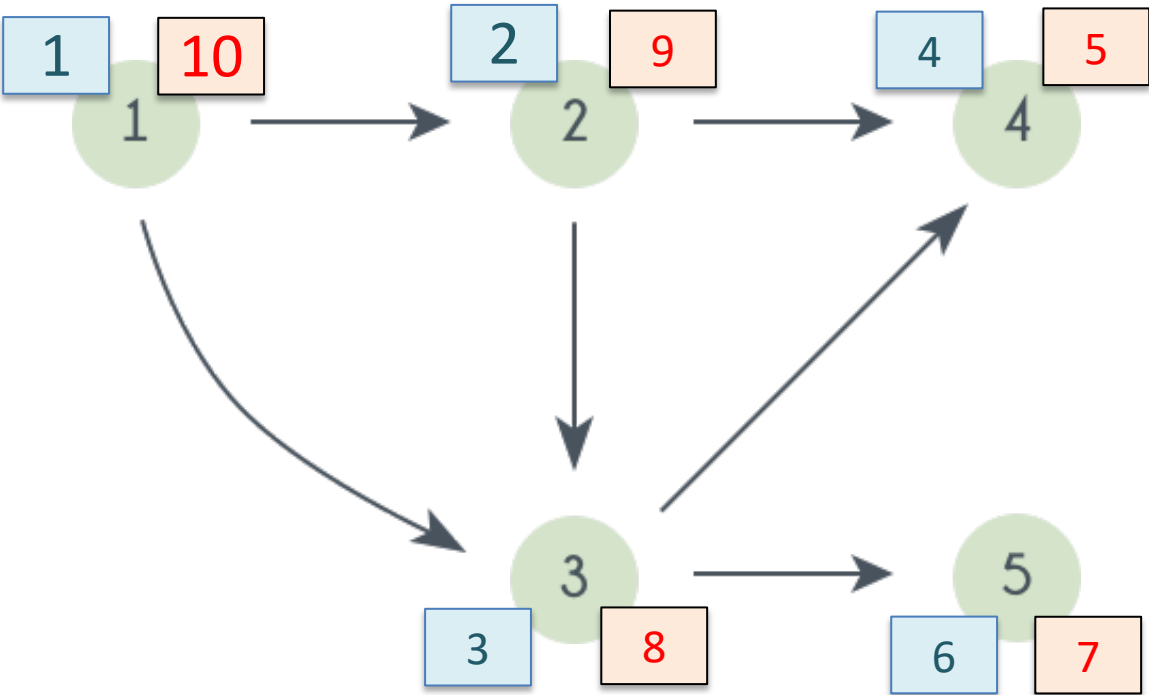
ops	stack content
init	\$
push(1)	\$1
pop 1	\$

Check soln DFS exercise: push- and pop-order (pre- and post-order)

Problem: For the graph below, write the push and pop order for DFS, starting from node 1

Method 1: Fill in the timestamp in yellow boxes for push-orders, pink boxes for pop-orders.

Method 2: Show the stack content and operations.



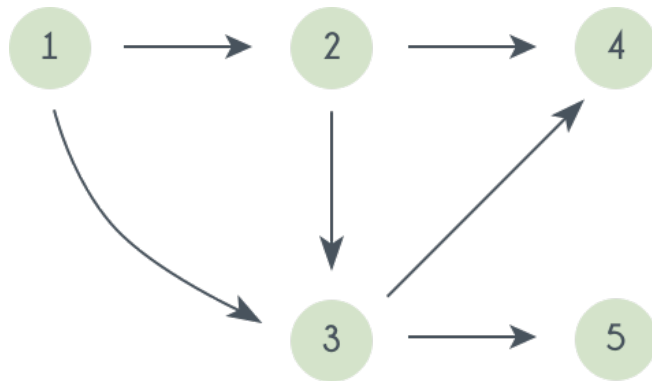
The PUSH and POP, using \$ for the bottom of stacks.

ops	stack content
init	\$
push(1)	\$1
push(2)	\$12
push(3)	\$123
push(4)	\$1234
pop 4	\$123
push(5)	\$1235
pop 5	\$123
pop 3	\$12
pop 2	\$1
pop 1	\$

DFS exercise: generating push- and pop-order

Problem: Modify the DFS algorithm so that it also builds the arrays `push[V]` and `pop[V]` to store the push- and the pop-order of the vertices.

Example graph:

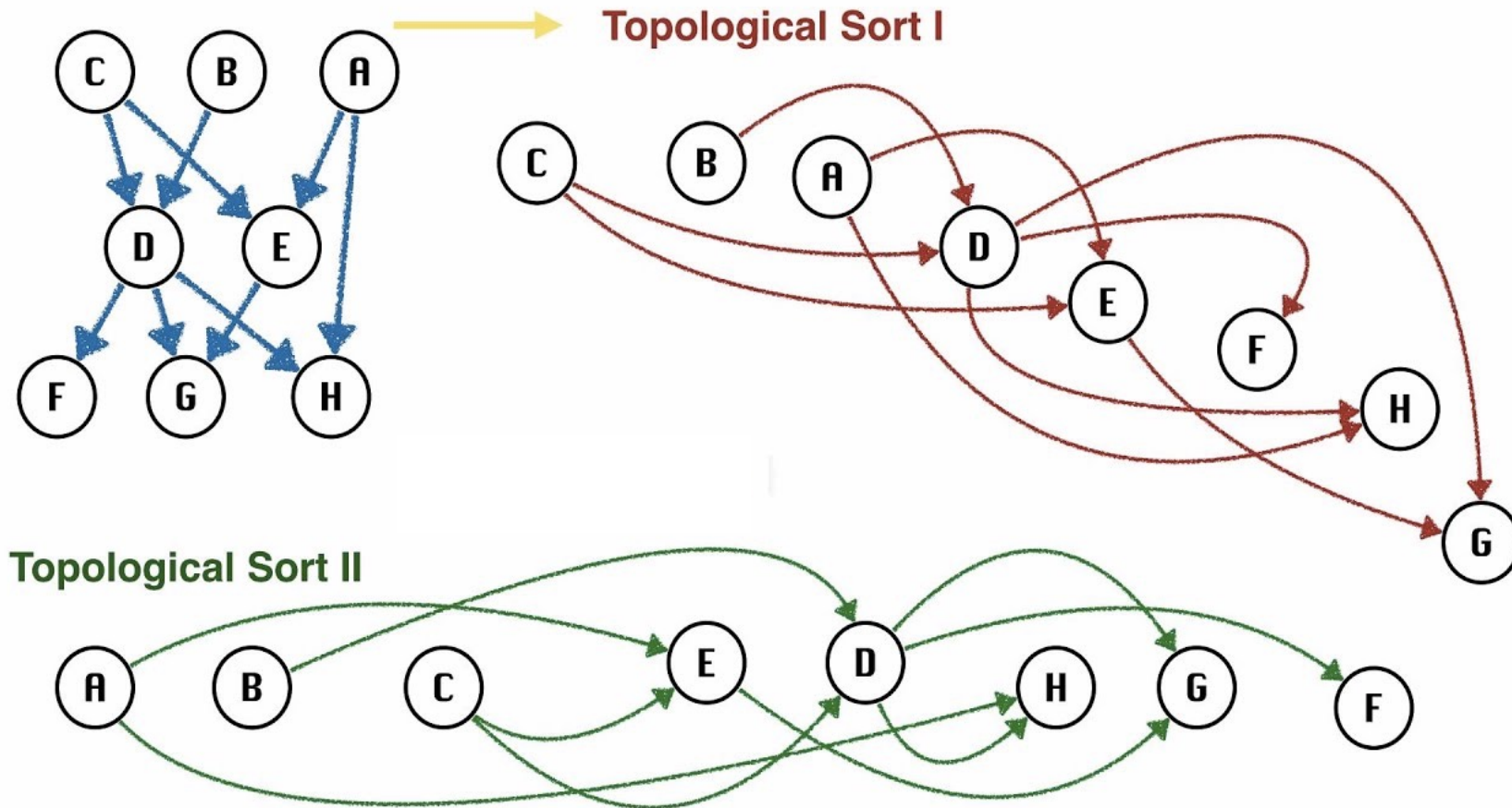


```
// building push[V] and pop[V]
function DFS(G=(V,E))
  for each v in V do
    mark v with 0
  for each v in V do
    if v is marked with 0 then
      DfsExplore(v)

function DfsExplore(v)
  mark v with 1
  for each edge (v,w) in E do
    if w is marked with 0 then
      DfsExplore(w)
```

Topological Sorting (for DAG only!)

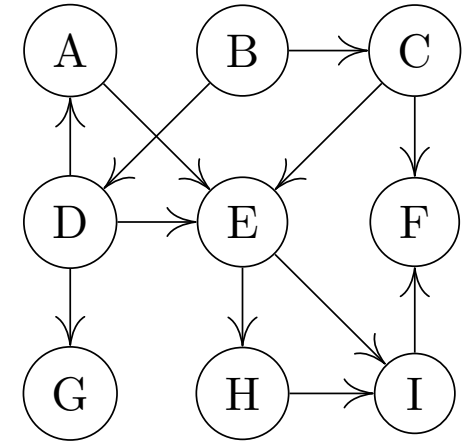
A *topological ordering*: sorting the nodes of the graph such that all edges point in one direction, to nodes later in the ordering.



Group Work: Q 6.1

Q6.1: One of the algorithms 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.

→ Finding a topological order for the graph by running a DFS.



operation	content	operation	content
init stack	\$		
push(A)	\$A		

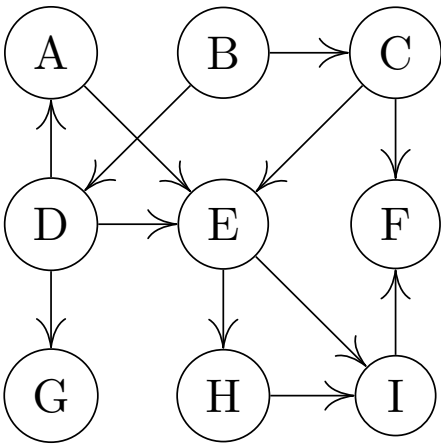
The topological order resulted from the above DFS run:

???

Check Soln: Q6.1

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

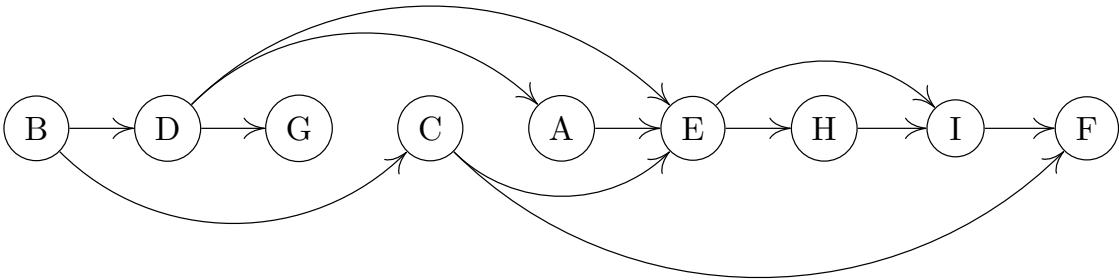
operation	stack_content	operation	stack_content
init stack	\$	push (B)	\$B
push (A)	\$A	push (C)	\$BC
push (E)	\$AE	pop C	\$B
push (H)	\$AEH	push (D)	\$BD
push (I)	\$AEHI	push (G)	\$BDG
push (F)	\$AEHIF	pop G	\$BD
pop F	\$AEHI	pop D	\$B
pop I	\$AEH	pop B	\$
pop H	\$AE		
pop E	\$A		
pop A	\$		



The topological order resulted from the above DFS run (= reversed pop order): B D G C A E H I F , you can check by re-draw the graph in the above linear order.

Notes:

- Alternatively to showing stack content, you can run DFS manually and mark post order with time stamp

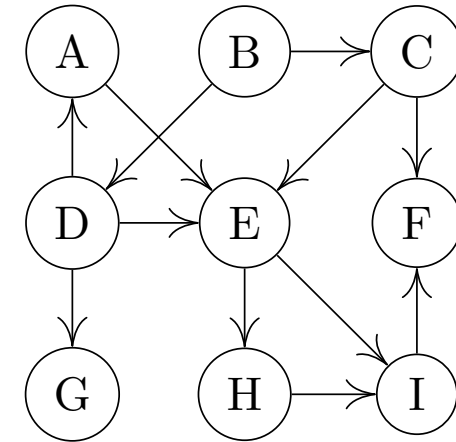


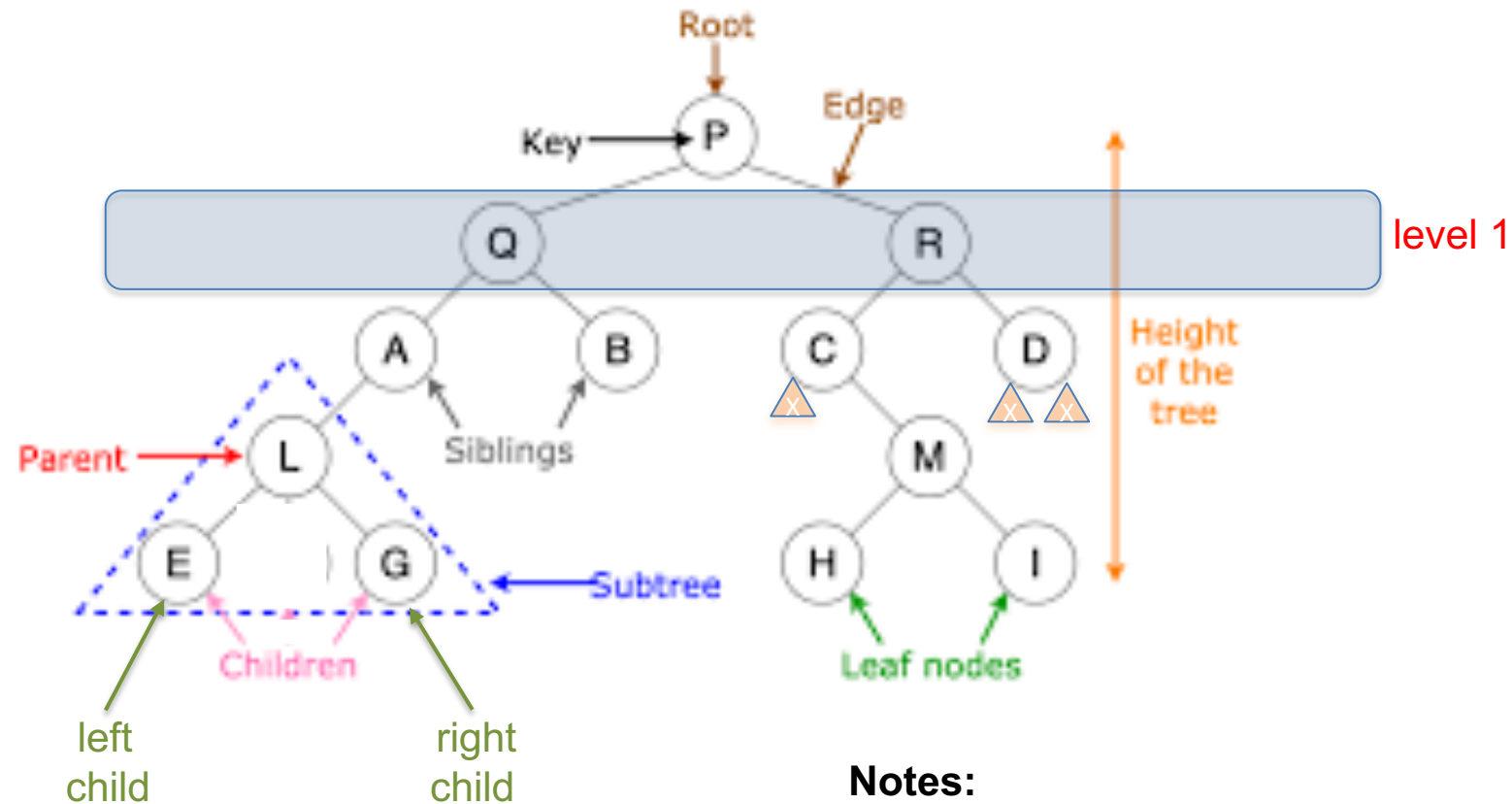
Q 6.1: Discuss other methods for toposort?

Additional Concept for di-graph nodes:

- a *source node* is a node that have no incoming edge,
- a *sink node* is a node that have no outgoing edge.

Alternative method(s) for Finding a topological order.





Notes:

△ denotes a NULL node, aka. *external node*, only a few of them drawn here. If the tree has n internal nodes, it has $n+1$ external nodes

Make differences between:

- *leaf nodes* and *external nodes*
- *none-leaf nodes* and *internal nodes*

Binary Tree: Recursive Definition

A binary tree is:

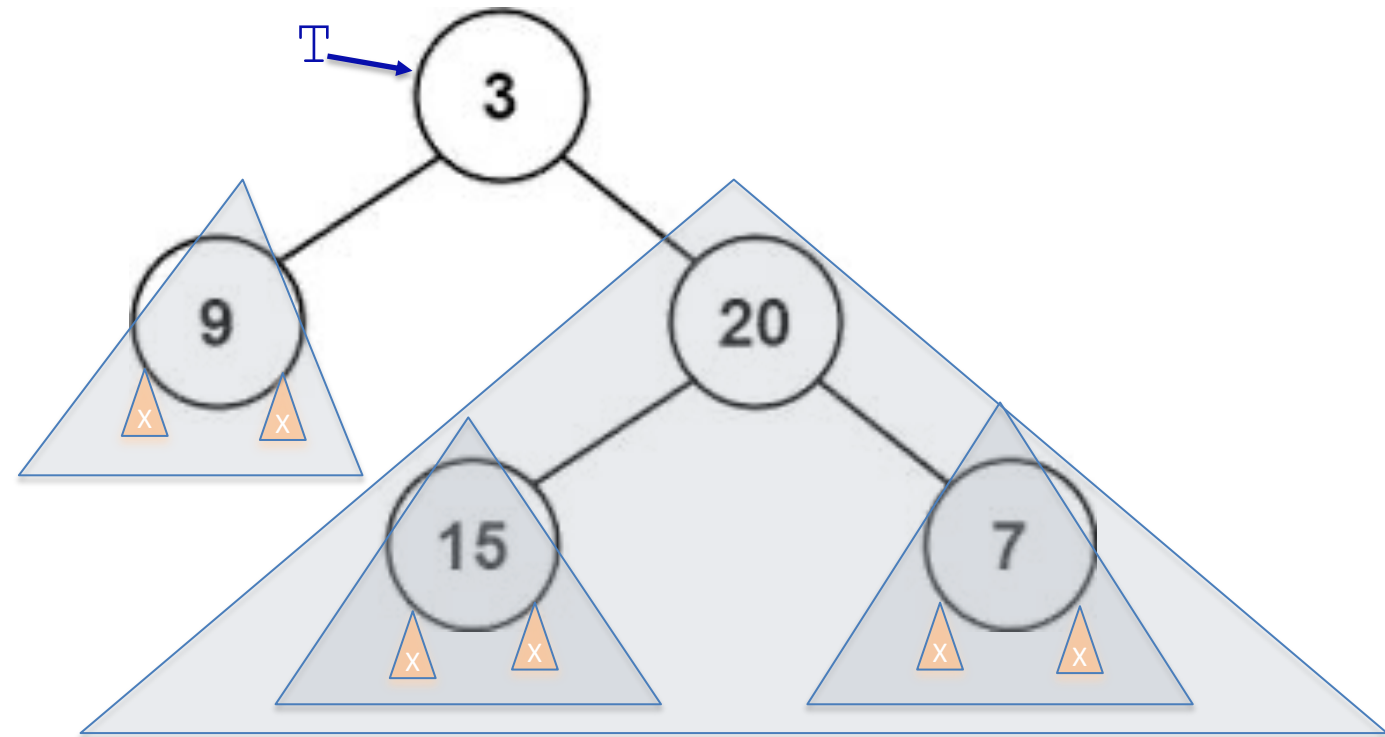
- NULL, or
- a node, called the tree's *root node*, that contains:
 - some data,
 - a link to another binary tree called the root's *left child*, and
 - a link to another binary tree called the root's *right child*

Side note: a non-empty tree is fully defined by its root. For pseudocode simplicity we use:

tree T = its root node

= pointer to its root node.

*For convenience, we also use T.key (or T.data),
T.left, T.right*



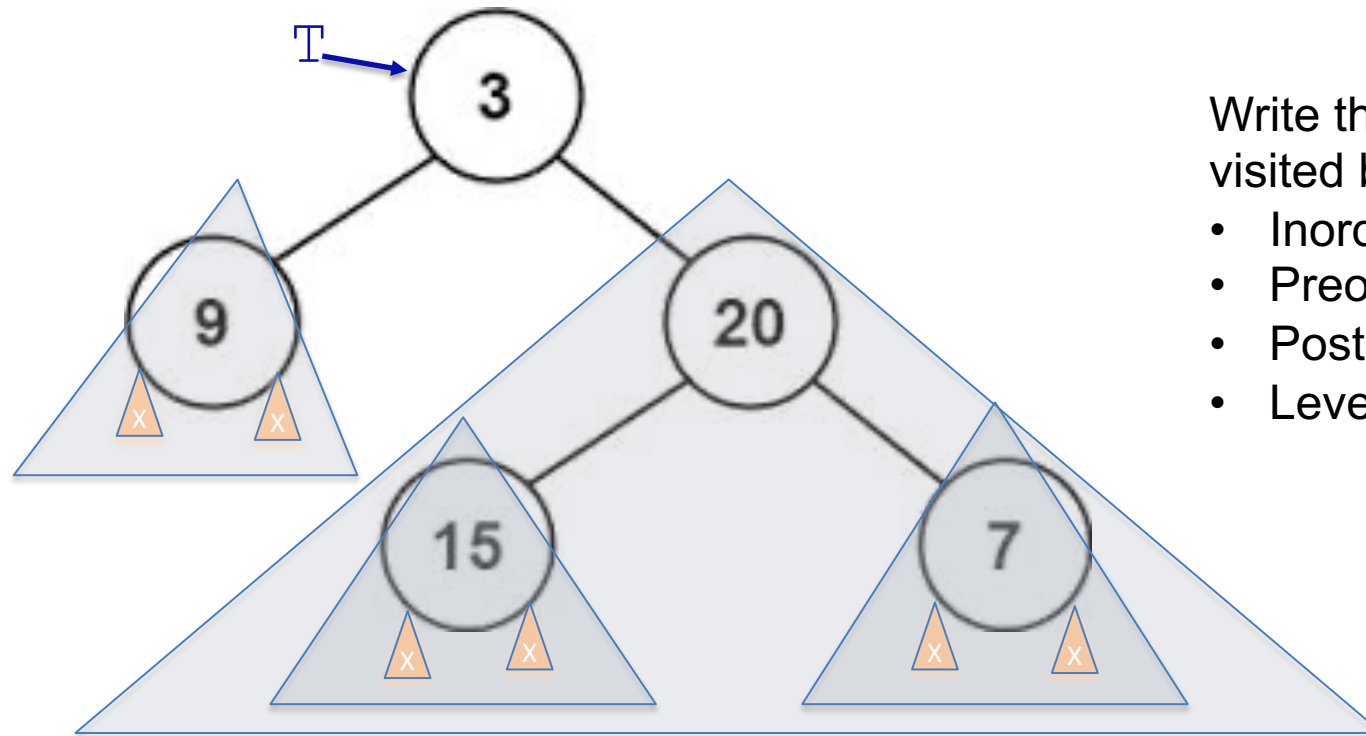
Binary tree traversal

Understand:

tree traversal?

inorder, preorder, postorder traversal? Are they BFS or DFS?

What is *level-order* traversal? Is it BFS or DFS?

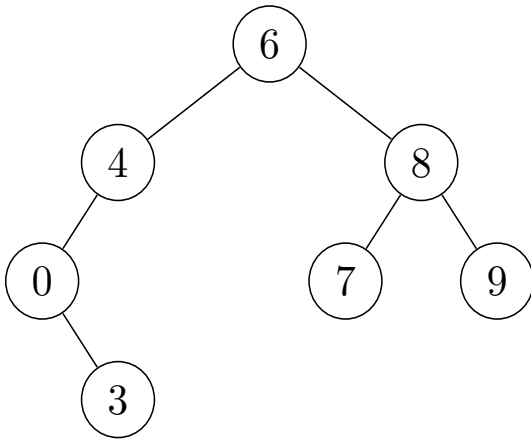


Write the nodes in order visited by:

- Inorder :
- Preorder :
- Postorder :
- Levelorder:

Q 6.4: Binary Tree Sum

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



YOUR ANSWER: The pseudocode:
`function ???`

Binary Search Tree

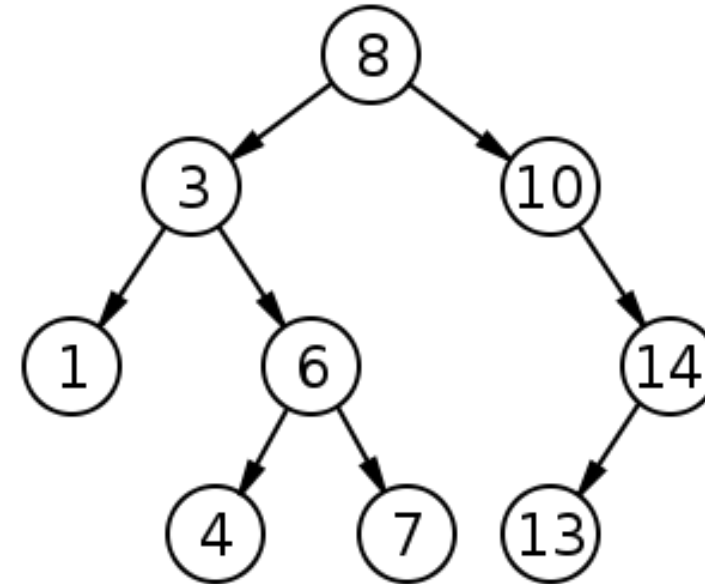
Review: What's a BST?

How to:

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

Your answers:

- 1.
- 2.
- 3.
- 4.



Small Exercise:

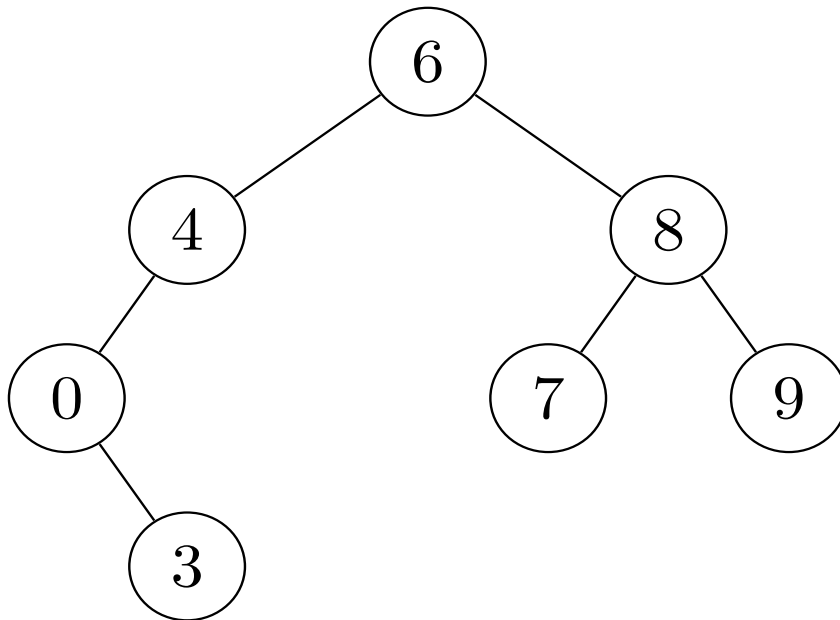
- Write a function for searching in BST
- What's the complexity of that function?

Group/Individual work

Questions 6.2, 6.3.

Q 6.2: conventional traversal

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



YOUR ANSWER:

In-order:

?

Pre-order:

?

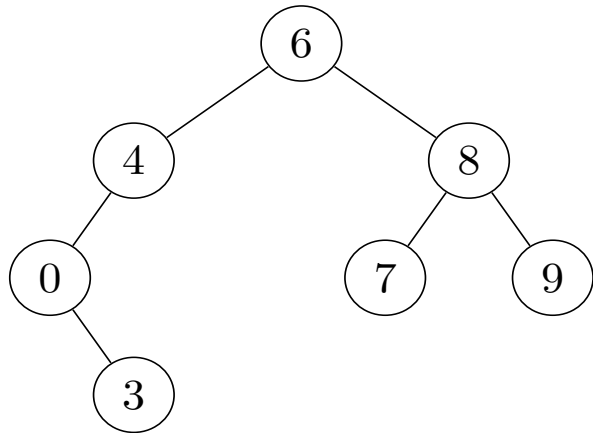
Post-order:

?

Q 6.3: 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:

a) Level-order: ???

```
// level order traversal for binary tree T
function LevelOrder(T)
    ?
```

- Do the BST insert exercise, aim to finish within 30 minutes a runnable version with:
 - data types
 - function insert
 - function printing keys in order
 - main(): a loop to read data and build the tree
- Questions on Dijkstra's and BFS, DFS, graph?

OR:

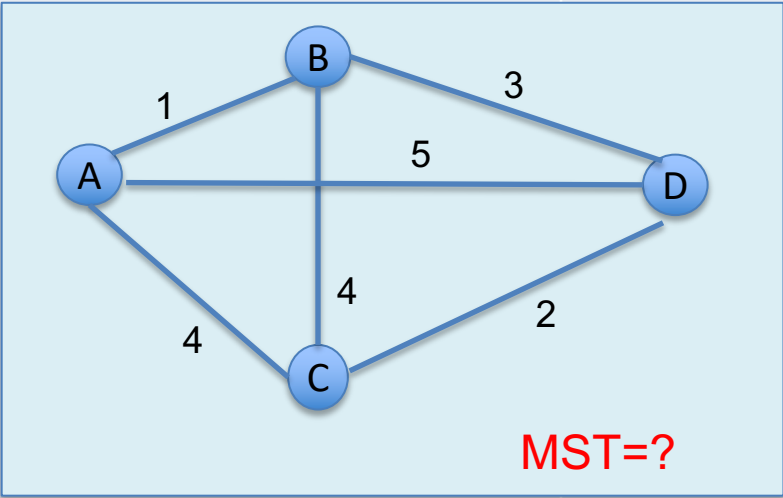
do/ask_questions_on not-yet-done problems (if any) of previous workshops/lab/lectures

Prim's Algorithm vs Dijkstra's Algorithm. Discuss concepts

	Prim's	Dijkstra's
<i>Aim</i>	find a MST	find SSSP from a vertex s
<i>Applied to</i>	connected weighted graphs with weights ≥ 0	weighted graphs with weights ≥ 0
<i>Works on directed graphs?</i>	?	
<i>Works on unweighted graph?</i>	?	

Related concepts for Prim's

- spanning trees = ?
- MST = ?
- is MST unique?



Dijkstra's and Prim's are similar

Dijkstra($G=(V,E),S$)

Task: Find SSSP from S (that involves all nodes of a *connected* graph)

```
for each  $v \in V$  do  
     $\text{cost}[v] := \infty$   
     $\text{prev}[v] := \text{nil}$ 
```

```
 $\text{cost}[S] = 0$   
 $\text{PQ} := \text{create\_priority\_queue}(V, \text{cost})$  with  
     $\text{cost}[v]$  as priority of  $v \in V$ 
```

```
while ( $\text{PQ}$  is not empty) do  
     $u := \text{ejectMin}(\text{PQ})$ 
```

```
    for each neighbour  $v$  of  $u$  do
```

```
        if  $\text{dist}[u] + w(u,v) < \text{cost}[v]$  then  
             $\text{cost}[v] := \text{cost}[u] + w(u,v)$   
            update ( $v, \text{cost}[v]$ ) in  $\text{PQ}$   
             $\text{prev}[v] := u$ 
```

Prim($G=(V,E)$)

Task: MST (that involves all nodes of a *connected* graph)

```
for each  $v \in V$  do  
     $\text{cost}[v] := \infty$   
     $\text{prev}[v] := \text{nil}$ 
```

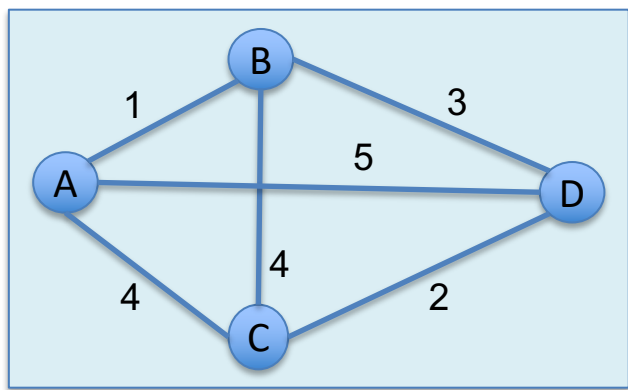
```
pick initial  $S$ 
```

```
 $\text{cost}[S] := 0$   
 $\text{PQ} := \text{create\_priority\_queue}(V, \text{cost})$   
with  $\text{cost}[v]$  as priority of  $v \in V$ 
```

```
while ( $\text{PQ}$  is not empty) do  
     $u := \text{ejectMin}(\text{PQ})$ 
```

```
    for each neighbour  $v$  of  $u$  do
```

```
        if  $w(u,v) < \text{cost}[v]$  then  
             $\text{cost}[v] := w(u,v)$   
            update ( $v, \text{cost}[v]$ ) in  $\text{PQ}$   
             $\text{prev}[v] := u$ 
```



Running Prim's Algorithm to find a MST

At each step, we add a node to MST.

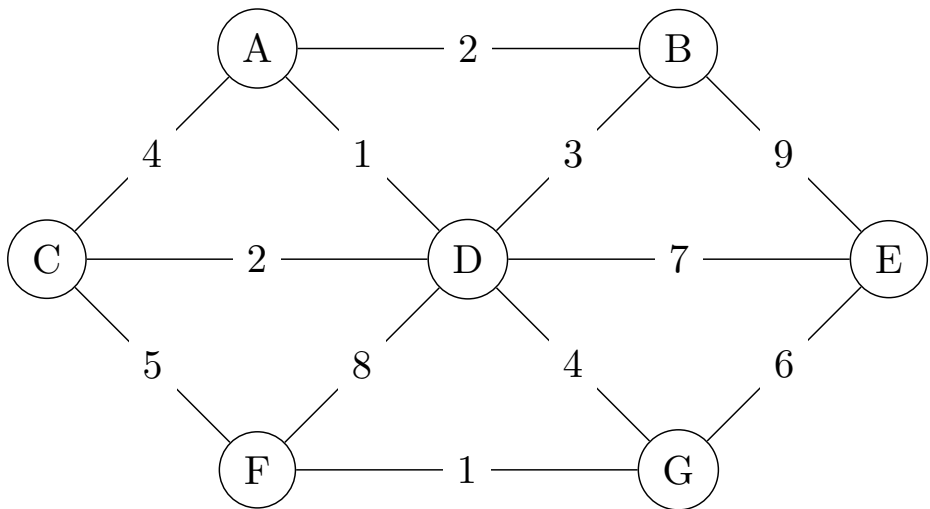
We choose the node with **minimal edge cost**.

We start with A according to the alphabetical order.

step	node added to MST	A	B	C	D
0		0,nil	∞ ,nil	∞ ,nil	∞ ,nil
1					
2					
3					
4					

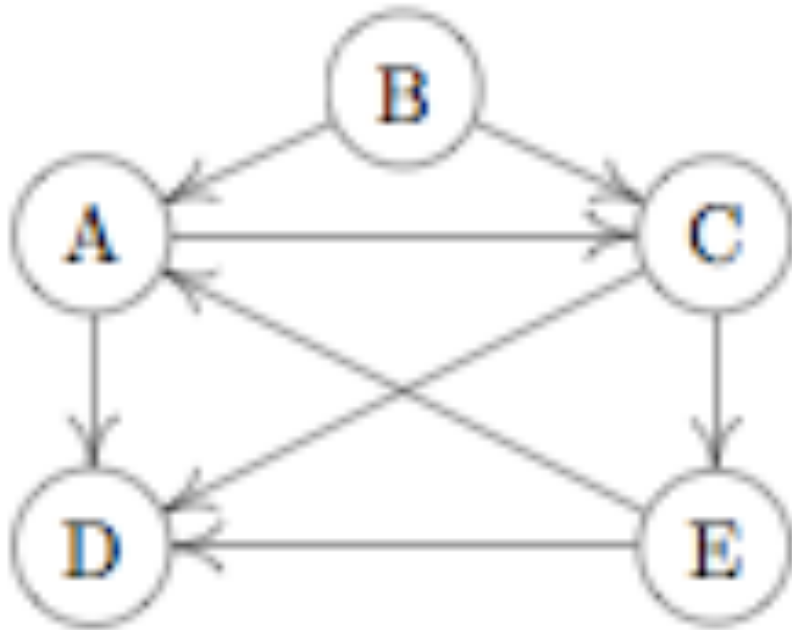
Question 5.8: Minimum Spanning Tree with Prim's Algorithm

Prim's algorithm finds a minimum spanning tree for a weighted graph. Discuss what is meant by the terms 'tree', 'spanning tree', and 'minimum spanning tree'. Run Prim's algorithm on the graph below, using A as the starting node. What is the resulting minimum spanning tree for this graph? What is the cost of this minimum spanning tree?



step	node done	A	B	C	D	E	F	G
0		0/nil	∞ /nil	∞ /nil	∞ /nil	∞ /nil	∞ /nil	∞ /nil
1								
2								
3								
4								
5								
6								
7								

Q5.5: Tree, Back, Forward and Cross Edges



A DFS of a di-graph can be represented as a collection of trees. Each edge of the graph can then be classified as a *tree edge*, a *back edge*, a *forward edge*, or a *cross edge*. A tree edge is an edge to a previously un-visited node, a back edge is an edge from a node to an ancestor, a forward edge is an edge to a non-child descendent and a cross edge is an edge to a node in a different sub-tree (i.e., neither a descendent nor an ancestor)

Draw a DFS tree based on the following graph, and classify its edges into these categories.

In an undirected graph, you won't find any forward edges or cross edges. Why is this true? You might like to consider the graph above, with each of its edges replaced by undirected edges.

Q5.6: Finding Cycles – do in group of 2-3

a) Explain how one can use BFS to see whether an undirected graph is cyclic.

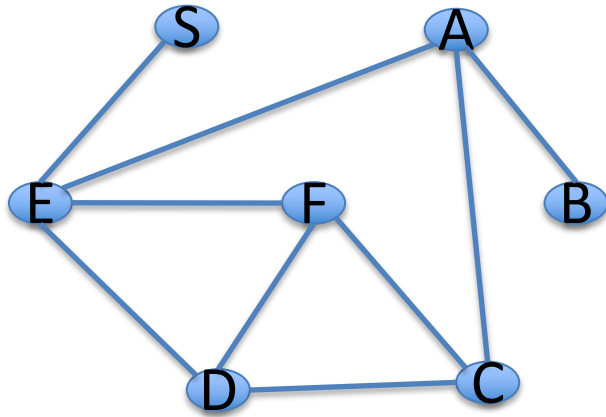
Transform function **BFS** to:
isCyclic($G=(V,E)$)

b) Can we use DFS for the task?
Which one is better: DFS or BFS?

BFS algorithm – as in lecture

```
function BFS ( $G=(V,E)$ )  
  mark each node in  $V$  with 0  
   $Q :=$  empty queue  
  for each  $v$  in  $V$  do  
    if  $v$  is marked with 0 then  
      mark  $v$  with 1  
      INJECT ( $Q, v$ )  
      while  $Q \neq \emptyset$  do  
         $u :=$  EJECT ( $Q$ )  
        for each edge  $(u,w)$  do  
          if  $w$  is marked with 0 then  
            mark  $w$  with 1  
            INJECT ( $Q, w$ )
```

5.7: 2-Colourability – do in group of 2-3



Design an algorithm to check whether an undirected graph is 2-colourable, that is, whether its nodes can be coloured with just 2 colours in such a way that no edge connects two nodes of the same colour.

To get a feel for the problem, try to 2-colour the following graph (start from **S**).

Do you expect we could extend such an algorithm to check if a graph is 3-Colourable, or in general: k -Colourable?

```
// transform this to is2Colorable
function DFS( $G=(V,E)$ )
  mark each node in  $V$  with 0
  for each  $v$  in  $V$  do
    if  $v$  is marked with 0 then
      DfsExplore( $v$ )

function DfsExplore( $v$ )
  mark  $v$  with 1
  for each edge  $(v,w)$  do
    if  $w$  is marked with 0 then
      DfsExplore( $w$ )

// Can we use BFS?
```