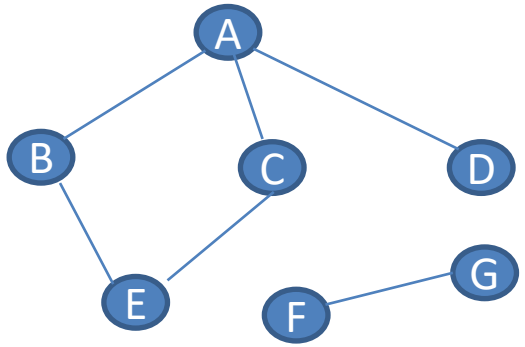
COMP20007 Workshop Week 6

1	**Topic 1:** DFS & Topological Sorting ***Exercise:*** Q6.1 (toposort)
2	**Topic 2:** Binary Trees & BST ***Group/Individual Exercises:*** Q 6.2, 6.4, 6.3
	Exercises from last week Lab from previous weeks
	ASSIGNMENT 1 • done • what if not?

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 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 at the end
```



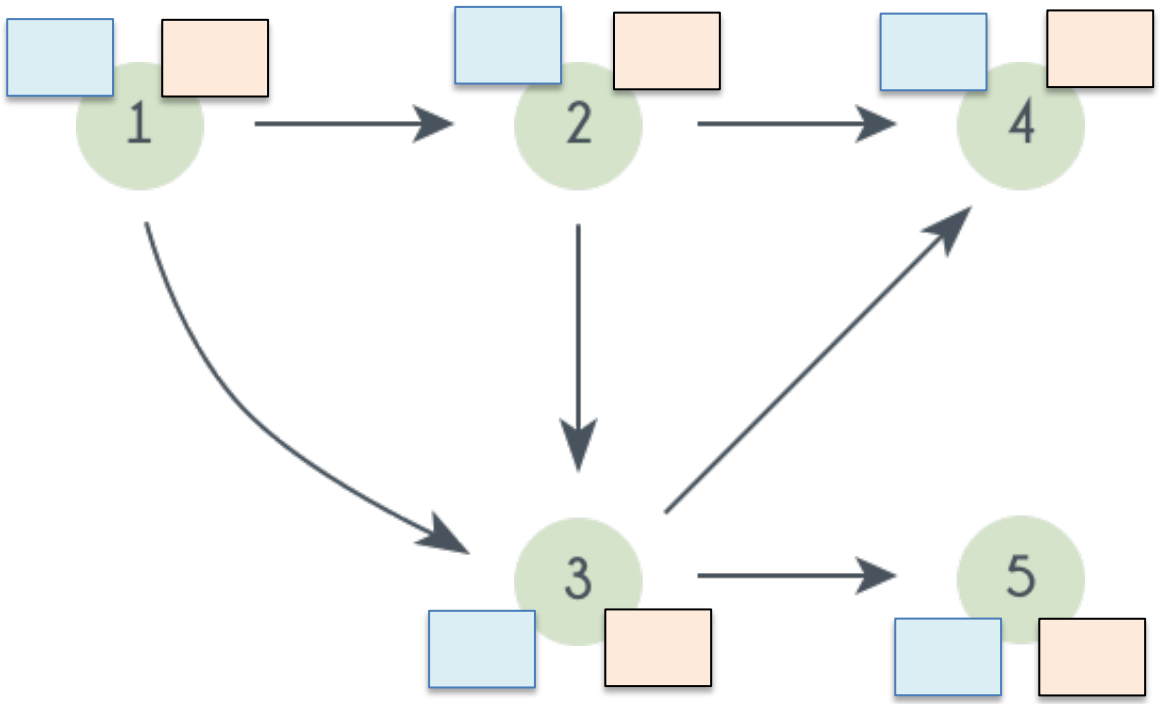
Work out Time Complexity

- for graphs with adjacency lists
- for graphs with 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.*

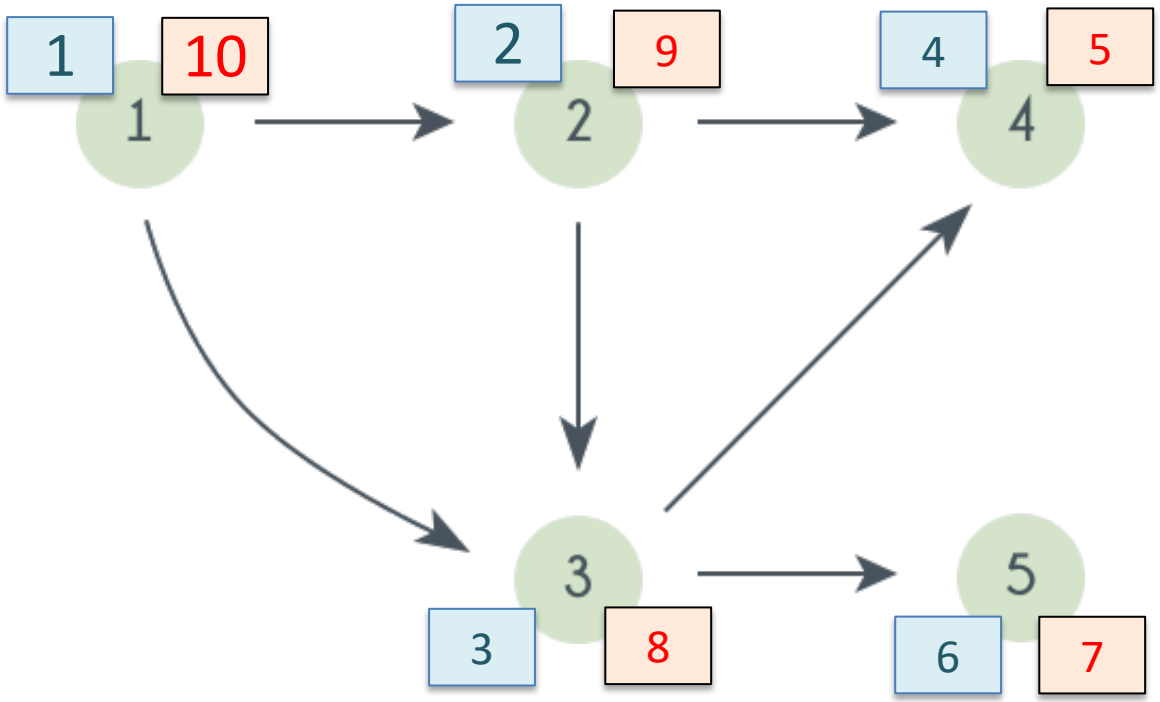



```
The PUSH and POP, using $
for the bottom of stacks.

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.*



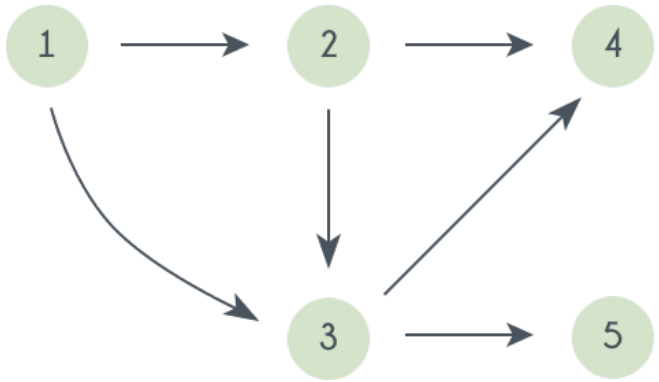
The PUSH and POP, using $ for the bottom of stacks.

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: push- and pop-order, DFS complexity

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.*

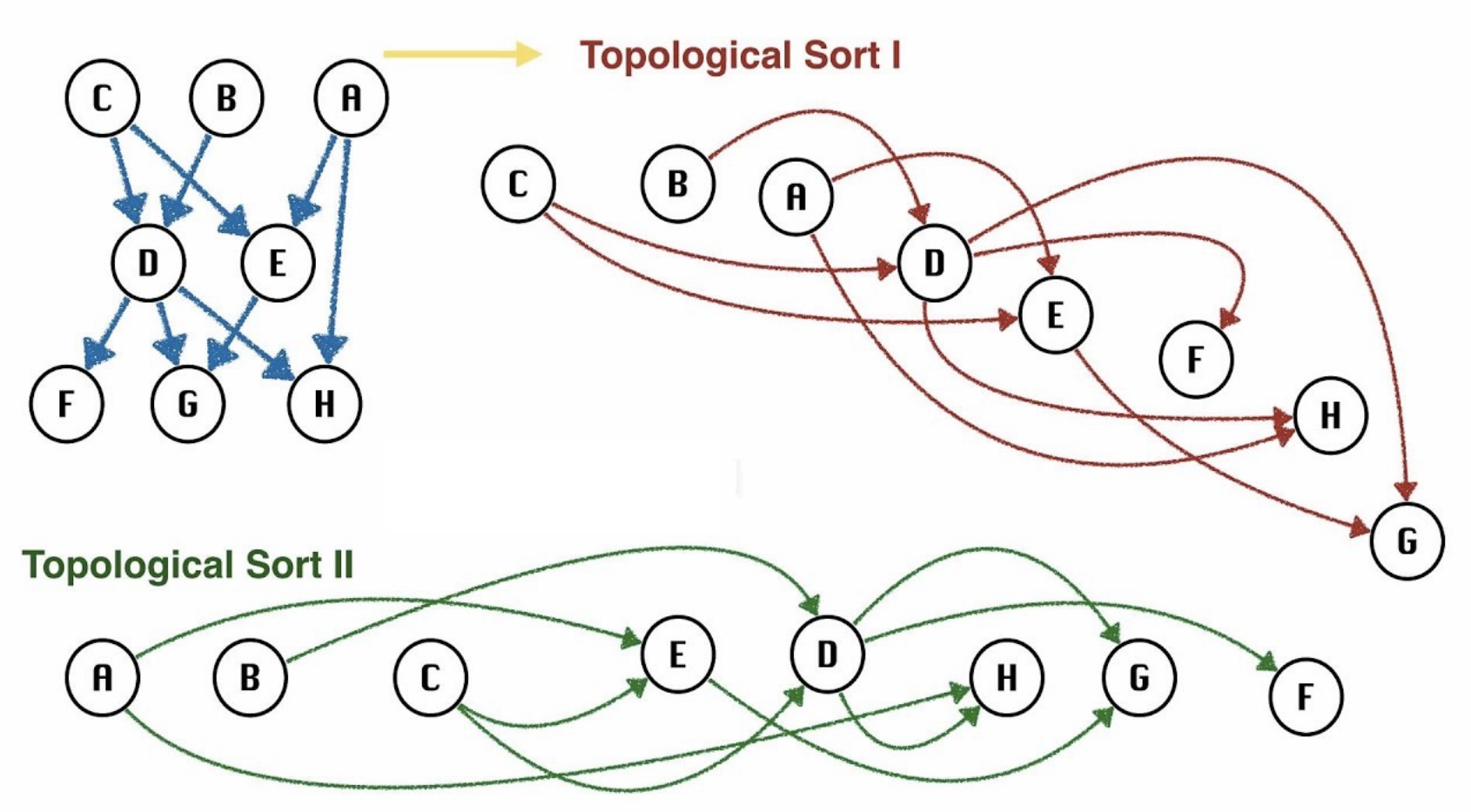
Example graph:




```
// 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.

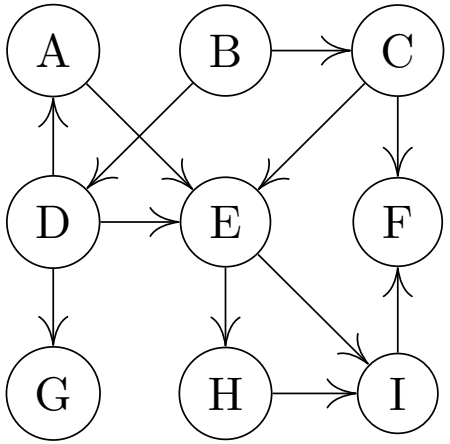


Group Work: Q 6.1

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.*

The reversed post-order is a topological order!



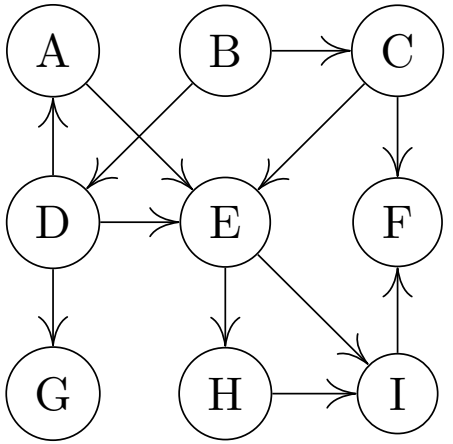
operation	content	operation	content
init stack	$		
push(A)	$A		

The topological order resulted from the above DFS run:
???

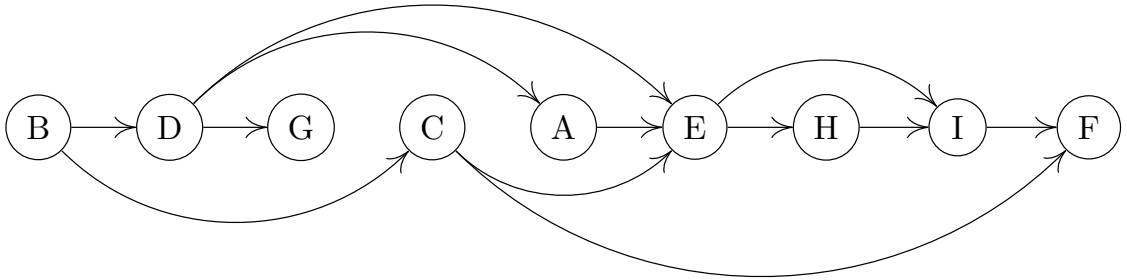
Check Soln: Q6.1

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.

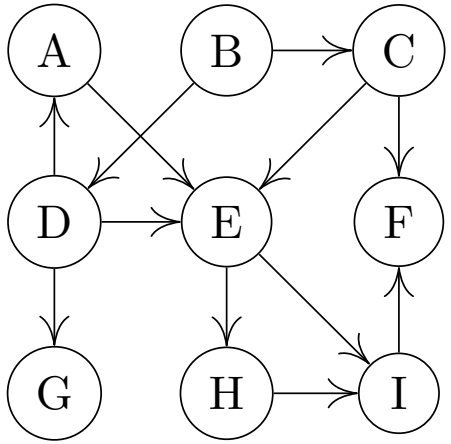


Q 6.1: Discuss other methods for toposort?

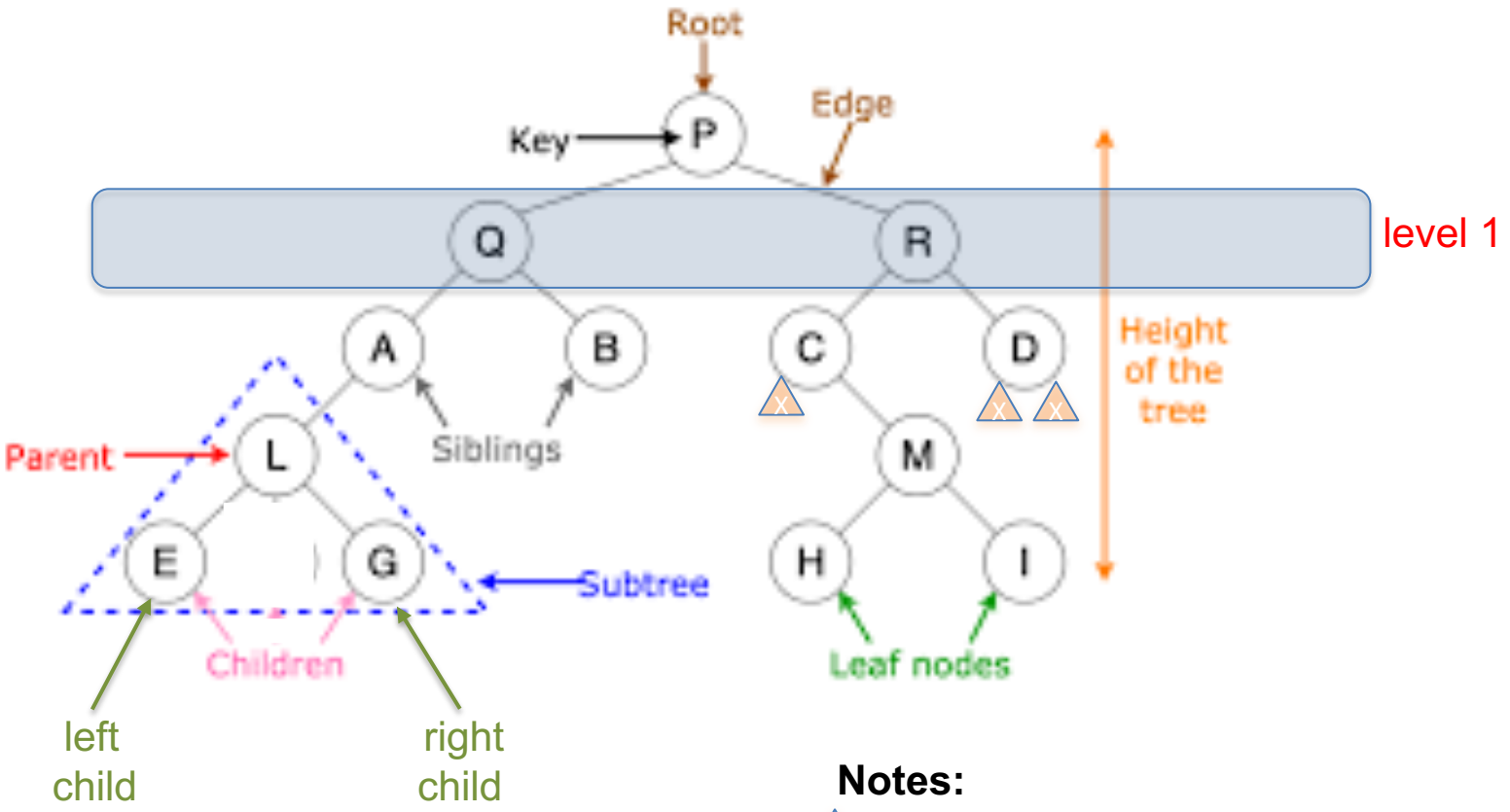
Alternative method *for Finding a topological order.*

Def: a source node is a node that have no incoming edge.

Algorithm based on source nodes= ?



Binary Trees as Special Graphs



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

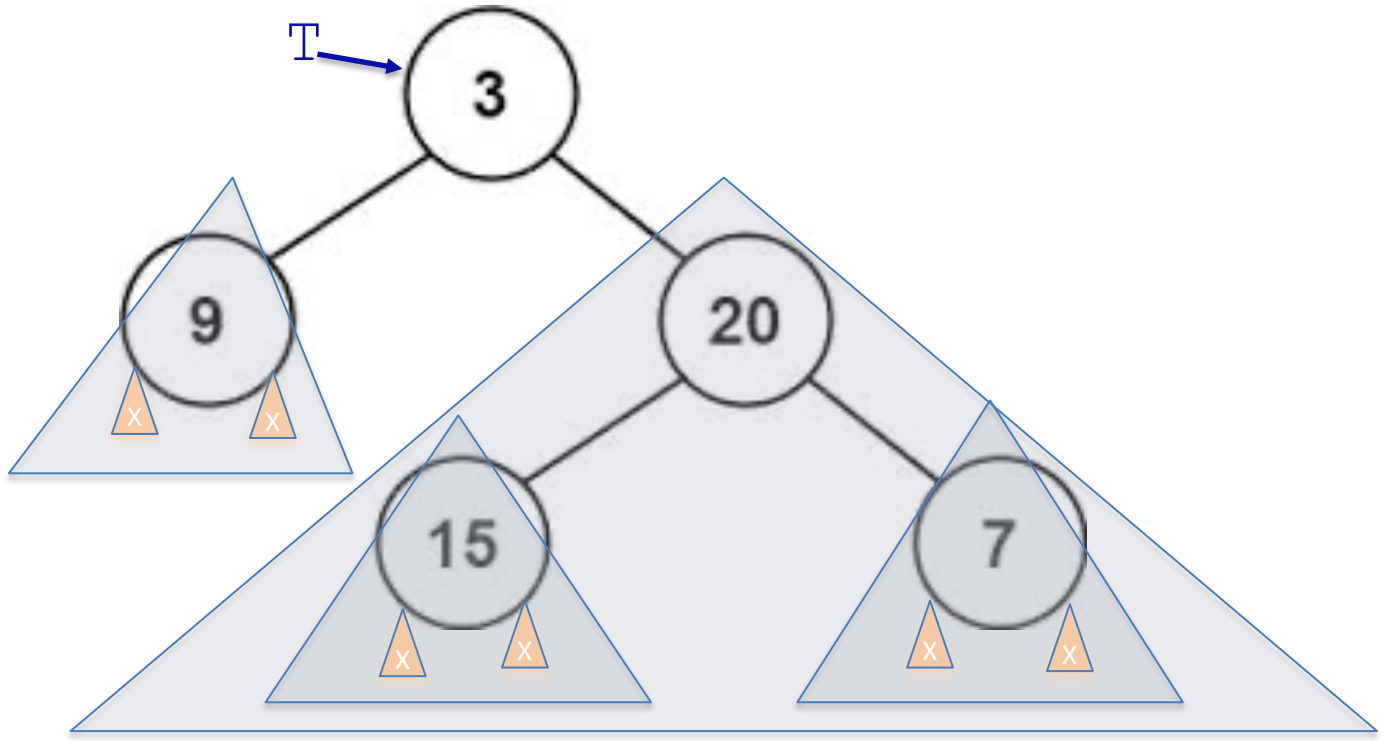
A binary tree is:

- NULL, or
- a node, called the tree's *root node*, that contains:
  - some data, normally including a key,
  - 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.



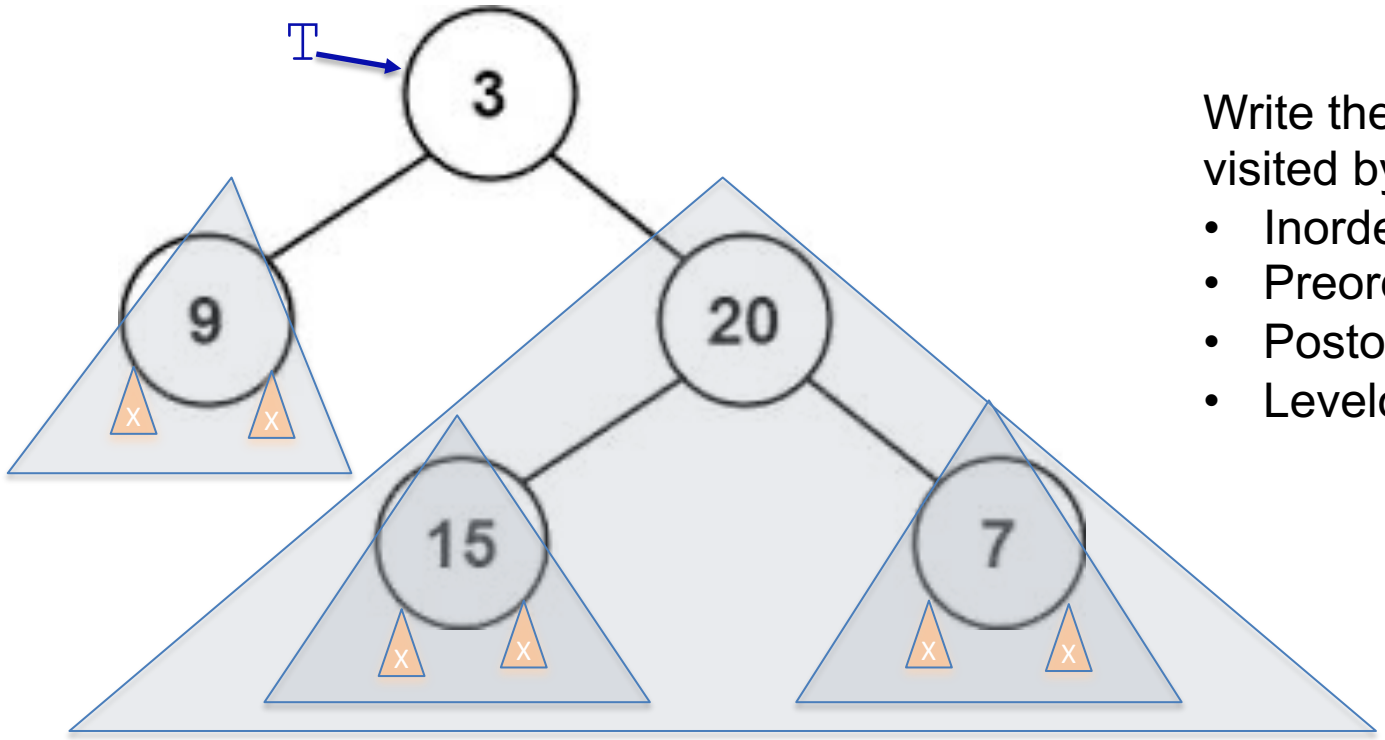
Binary tree traversal

Understand:

tree traversal?

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

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

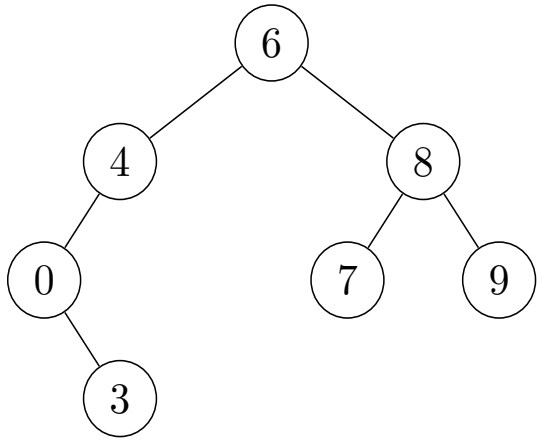


Write the nodes in order visited by:

- 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:

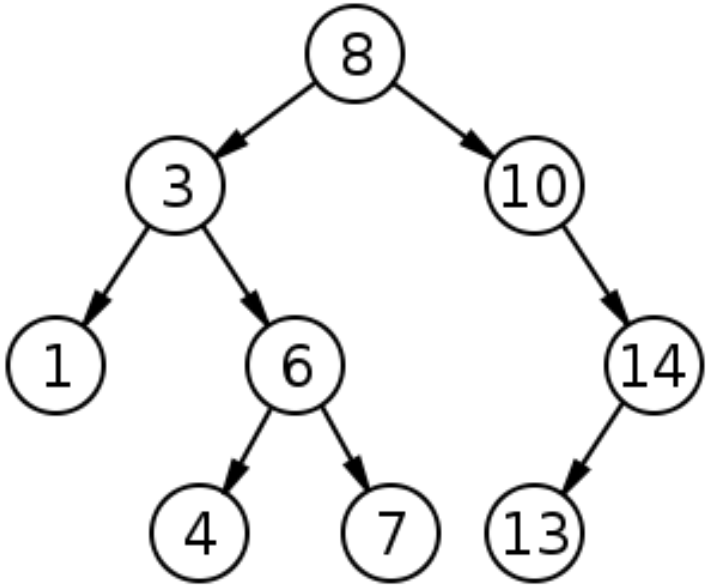
```
function ???
```

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?



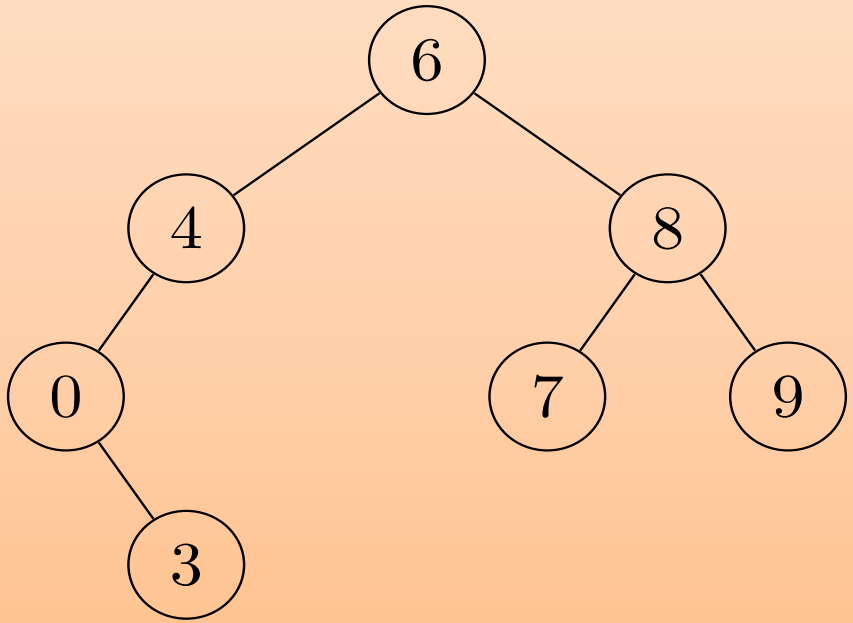
Your answers:

1.
2.
3.
4.

Group/Individual work

Q6.2:

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



Q6.3:

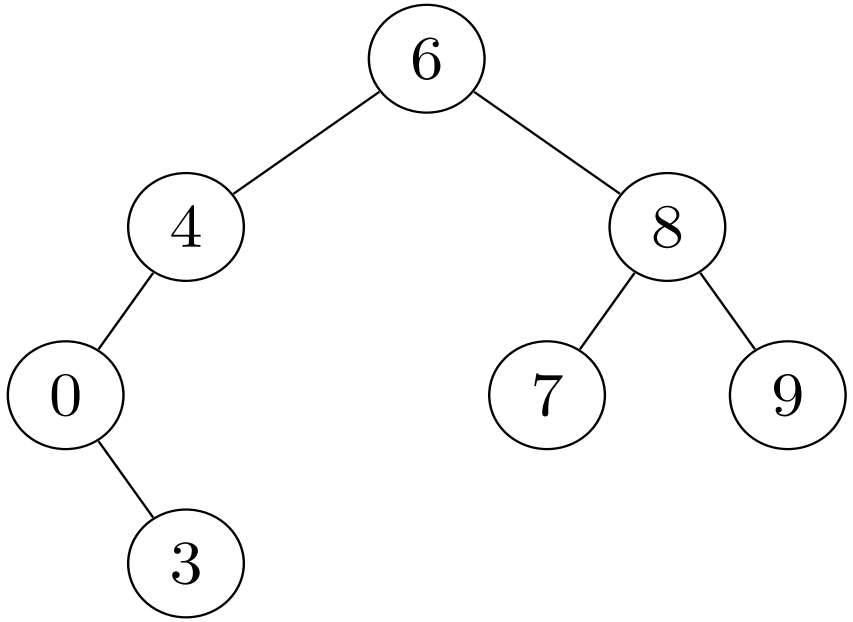
Level-order: visit level-by-level, left-to-right, starting from the root (which is in 0-th level).

a) For the tree of Q6.2, what's the visited order?

b) Write the level-order pseudo-code

Q 6.2: conventional traversal

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



YOUR ANSWER:

In-order:
?

Pre-order:
?

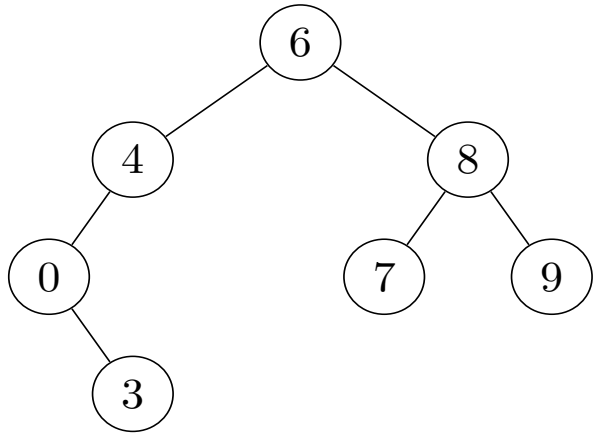
Post-order:
?

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: ???

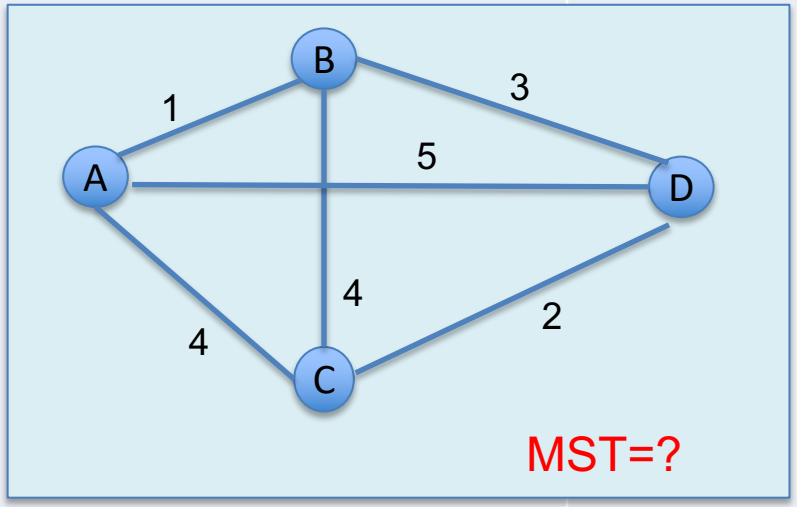
```
// level order traversal for binary tree T
function LevelOrder(T)
  ?
```

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

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

Related concepts for Prim's

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



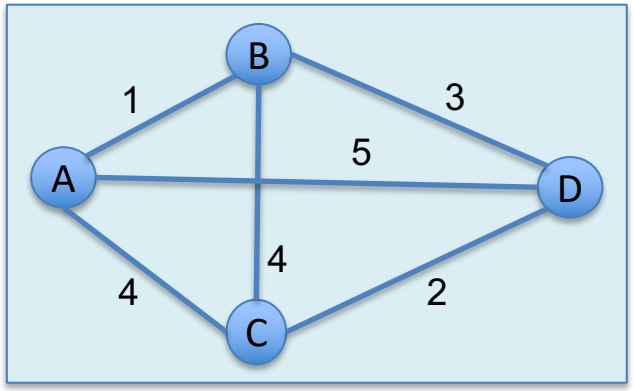
Dijkstra's and Prim's are similar

Dijkstra(G=(V,E),S)	Prim(G=(V,E))
Task: Find SSSP from S (that involves all nodes of a *connected* graph)	Task: MST (that involves all nodes of a *connected* graph)

`for each v ∈V do`
`  cost[v]:= ∞`
`  prev[v]:= nil`
 	
`cost[S]= 0`
`PQ:= create_priority_queue(V,cost) with`
`cost[v] as priority of v∈V`
	`for each v ∈V do`
	`  cost[v]:= ∞`
	`  prev[v]:= nil`
	`pick initial S`
	`cost[S]:=0`
	`PQ:= create_priority_queue(V, cost)`
	`with cost[v] as priority of v∈V`

`while (PQ is not empty) do` `  u := ejectMin(PQ)`	`while (PQ is not empty) do` `  u := ejectMin(PQ)`

`  for each neighbour v of u do`
 	
`  if  dist[u]+w(u,v) < cost[v] then`
`  cost[v]:= cost[u]+w(u,v)`
`  update (v, cost[v]) in PQ`
`  prev[v]:= u`
	`  for each neighbour v of u do`
	
	`  if  w(u,v) < cost[v] then`
	`  cost[v]:= w(u,v)`
	`  update (v, cost[v]) in PQ`
	`  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**.

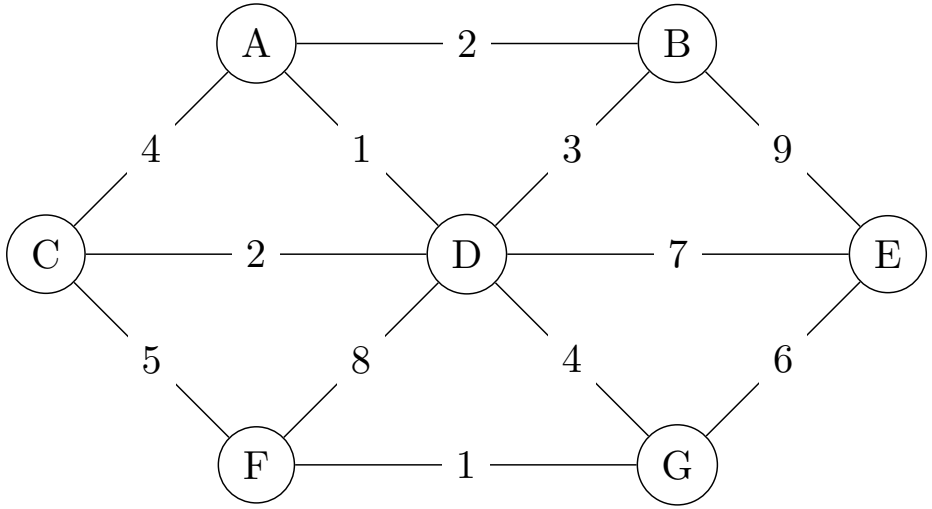
We start with A according to the alphabetical order.

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								

Q5.6: 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:

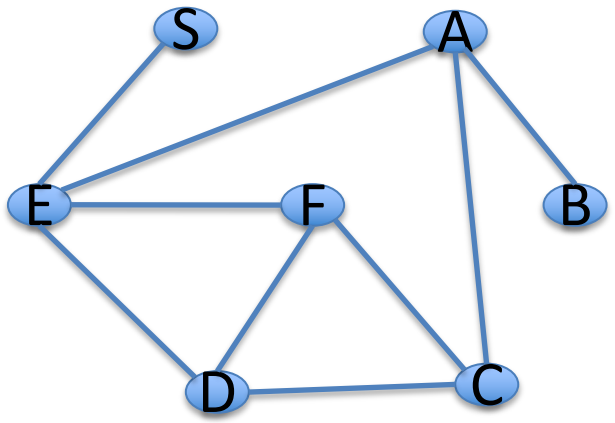
```
isCyclic(G=(V,E))
```

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 ≠ Ø  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



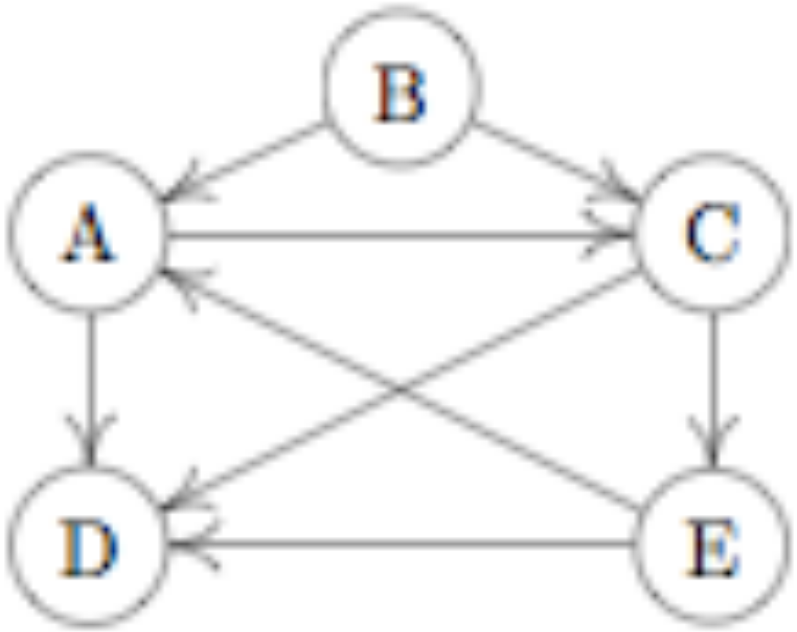
Design an algorithm to check whether an undirected graph is 2-colourable, that is, whether its nodes can be coloured with just 2 colours in such a way that no edge connects two nodes of the same colour.

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?
```

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 wont 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.

How to use valgrind in Ed to check for possible bugs?

When opening a programming terminal. Now, suppose you want to run `problem2a` with input data `test_cases/2a-1.txt`, you normally do with:

```
make problem2a
./problem2a <test_cases/2a-1.txt
```

Now, if you want to test with `valgrind`, just run:

```
valgrind –leak-check=full  ./problem2a <test_cases/2a-1.txt
```

`valgrind` will give some output. If at the end of the output, you see lines:

```
==??== All heap blocks were freed -- no leaks are possible
==??== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

then your probably doesn't have bugs (well, you need to check for correctness yourself). Otherwise, you need to scroll up and look at the start of the `valgrind` output, you will see `valgrind` reports some line numbers from some `.c` files that might cause problems. **Only solve the first error, then try `valgrind` again.**

Short intro to `valgrind`:

- https://epitech-2022-technical-documentation.readthedocs.io/en/latest/valgrind.html
- https://zoo.cs.yale.edu/classes/cs323/doc/Valgrind

LAB

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

Happy Easter Break