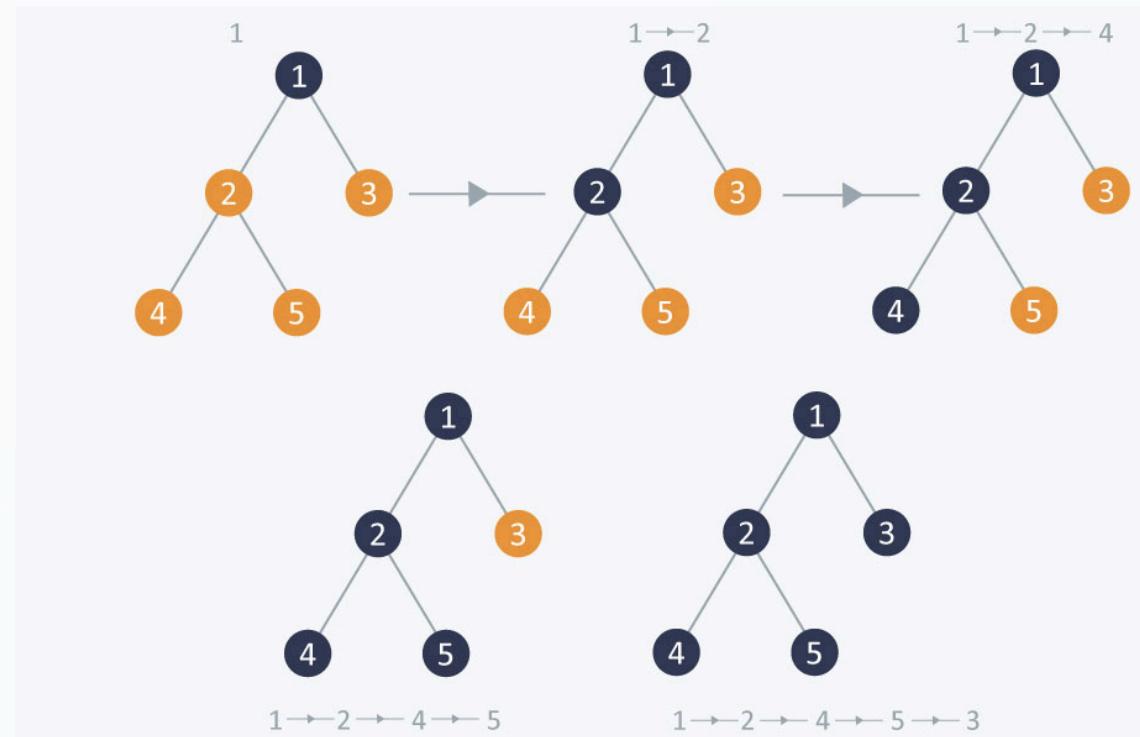


COMP20007 Workshop Week 6

- | | |
|-----|--|
| 1 | Preparation: <ul style="list-style-type: none">- open <code>ws6.pptx</code> from github.com/anhvir/c207- (optional) open <code>wokshop6.pdf</code> (from LMS) |
| 2 | Topic 1: DFS and BFS:
<i>Group Work:</i> Problems T1, T2, T4, T3 (in listed order) |
| 3 | Topic 2: Dijkstra's and Prim's Algorithms
<i>Group Work:</i> Problems T5, T6 |
| LAB | Assignment 1 Q&A, and <ul style="list-style-type: none">- make sure that you can scp, ssh, use dimefox to test/submit- do Assignment 1 or do not-yet-done exercises |
| | Before leaving: <i>Please give comments on the format of today's workshop</i> |

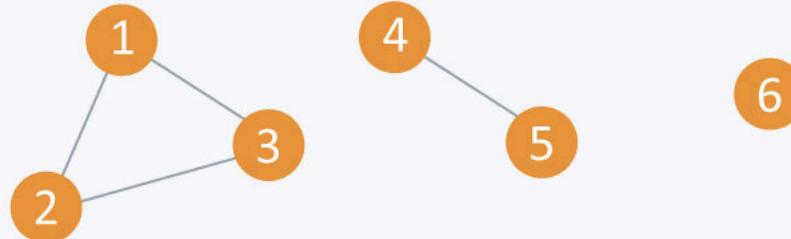
DFSEXPLORE(1): explore a connected component from node 1

```
function DFSEXPLORE( $v$ )
   $count \leftarrow count + 1$ 
  mark  $v$  with  $count$ 
  for each edge  $(v, w)$  do
    if  $w$  is marked with 0 then
      DFSEXPLORE( $w$ )
```



DFS: explore a whole (multi-component) graph

```
function DFS( $\langle V, E \rangle$ )
    mark each node in  $V$  with 0
     $count \leftarrow 0$ 
    for each  $v$  in  $V$  do
        if  $v$  is marked 0 then
            DFSEXPLOR(v)
```



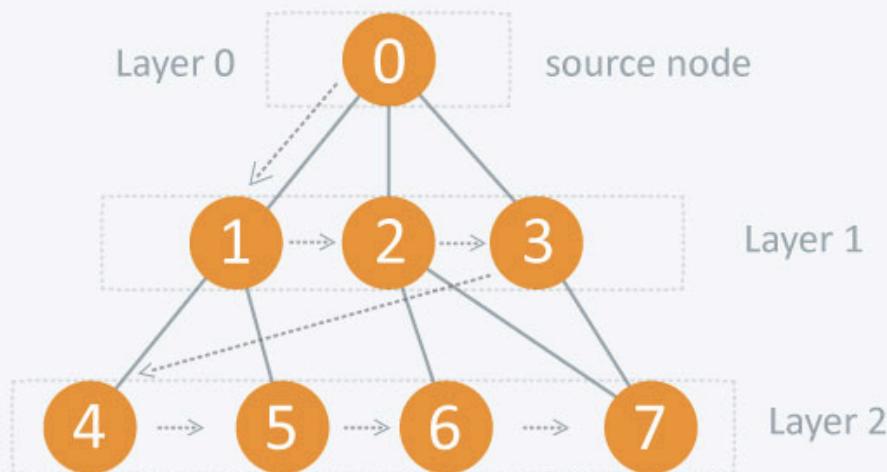
```
function DFSEXPLOR( $v$ )
     $count \leftarrow count + 1$ 
    mark  $v$  with  $count$ 
    for each edge  $(v, w)$  do
        if  $w$  is marked with 0 then
            DFSEXPLOR(w)
```

Using DFS how can we discover that:

- this graph has 3 components?
- this graph is cyclic?

Using DfsExplore, how can we list the length of all possible paths from $1 \rightarrow 3$?

BFSEXPLOR(O): explore a component from node O, in BFS manner



function $\text{BFS}(\langle V, E \rangle)$

mark each node in V with 0

$count \leftarrow 0$, $\text{init}(\text{queue})$

for each v in V **do**

if v is marked 0 **then**

$\text{BFSEXPLOR}(v)$

function $\text{BFSEXPLOR}(v)$

$count \leftarrow count + 1$

 mark v with $count$

$\text{inject}(\text{queue}, v)$

▷ queue c

while $queue$ is non-empty **do**

$u \leftarrow \text{eject}(\text{queue})$

for each edge (u, w) adjacent to u **do**

if w is marked with 0 **then**

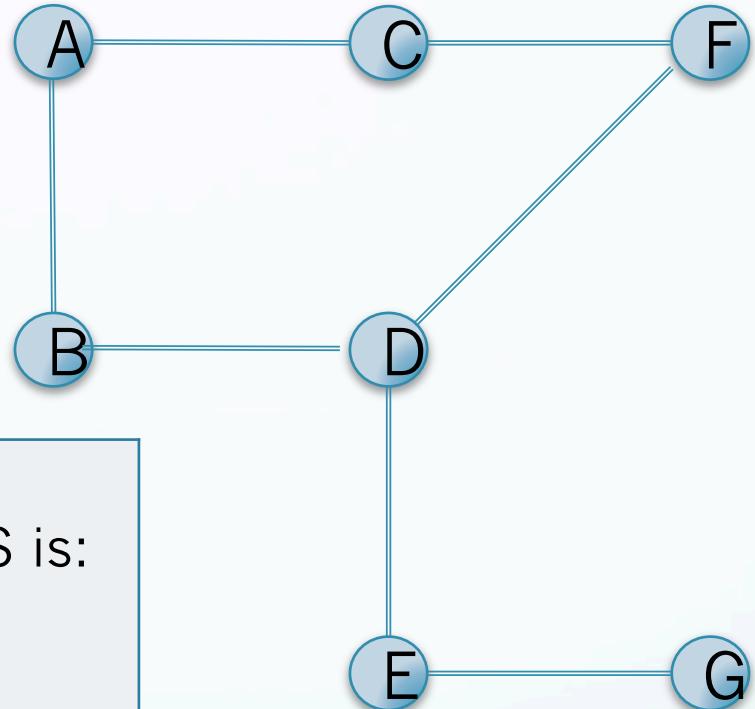
$count \leftarrow count + 1$

 mark w with $count$

$\text{inject}(\text{queue}, w)$

T1: Depth First Search

- List the order of the nodes visited by the a) DFS and b) BFS algorithms



YOUR ANSWER:

a) The order of the nodes visited by DFS is:

A

b) The order of the nodes visited by BFS is:

A

T3: Finding Cycles

a) Explain how one can also use BFS to see whether an undirected graph is cyclic. b) Which of the two traversals, DFS and BFS, will be able to find cycles faster? (If there is no clear winner, give an example where one is better, and another example where the other is better. – but skip this part if it takes more than 1 minute)

YOUR BRIEF ANSWER:

function $\text{BFS}(\langle V, E \rangle)$

mark each node in V with 0

$count \leftarrow 0$, $\text{init}(\text{queue})$

for each v in V **do**

if v is marked 0 **then**
 $\text{BFSEXPLORE}(v)$

function $\text{BFSEXPLORE}(v)$

$count \leftarrow count + 1$

mark v with $count$

$\text{inject}(\text{queue}, v)$

▷ queue \leftarrow

while $queue$ is non-empty **do**

$u \leftarrow \text{eject}(\text{queue})$

for each edge (u, w) adjacent to u **do**

if w is marked with 0 **then**

$count \leftarrow count + 1$

mark w with $count$

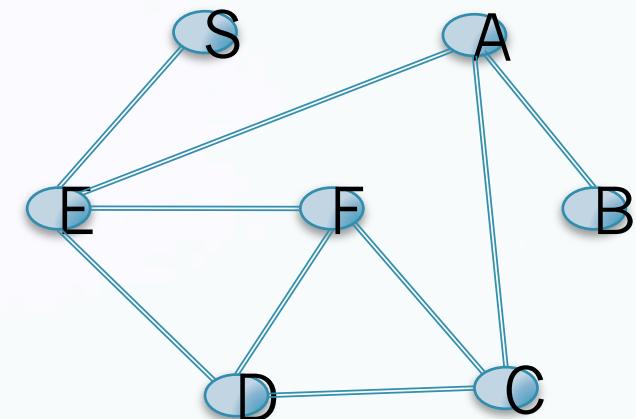
$\text{inject}(\text{queue}, w)$

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?

T4: 2-Colourability



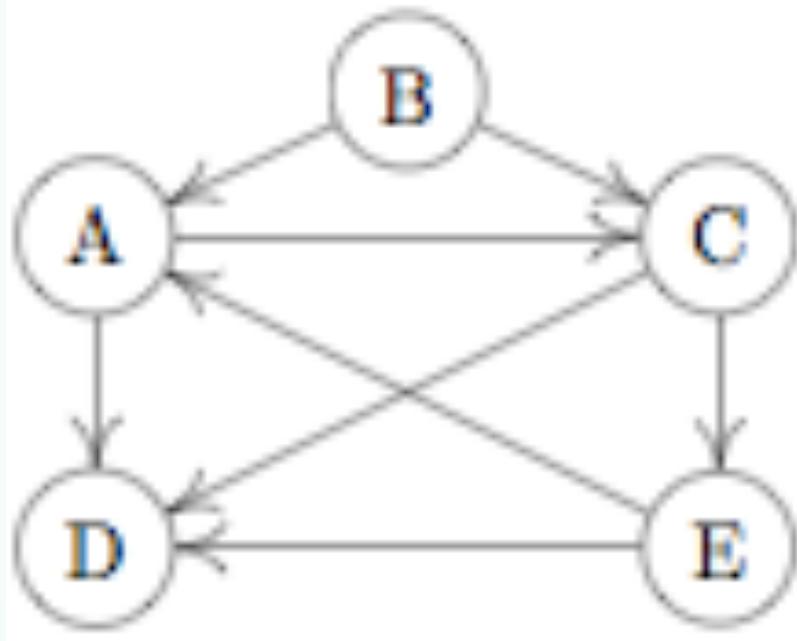
YOUR BRIEF ANSWER:

a) try to 2-colour the above graph, starting from **S**, using 2 colours + and –
hint: what is colour for S? how do we continue?

b) So, how to solve the 2-colourability?

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

T2



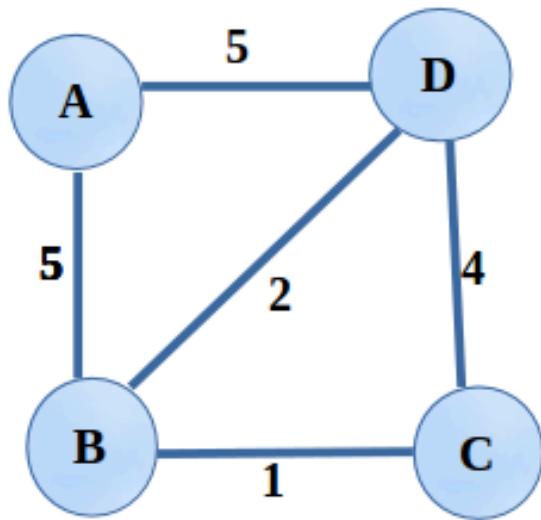
A DFS of a di-graph can be represented as a collection of trees. Each edge of the graph can then be classified as a *tree edge*, a *back edge*, a *forward edge*, or a *cross edge*. A tree edge is an edge to a previously un-visited node, a back edge is an edge from a node to an ancestor, a forward edge 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.

A gift from Anh: If you are a bit bored or tired, skip this exercise, and instead use pages 24-36 (of this file) to entertain yourselves ;-)

Dijkstra's Algorithm: purpose= ?



Dijkstra's Algorithm

purpose	SSSP (that involves all nodes of a connected graph)	
init	<pre>for all v ∈ V : dist[v] = ∞ prev[v] = nil dist[S] = 0</pre>	
basic loop	for all $u \in V$, at each step choose u that $\text{dist}[u]$ is min amongst the unselected	
action after selecting u	<pre>for all unselected v such that $(u, v) \in E$: if $\text{dist}[v] > \text{dist}[u] + w(u, v)$: update $\text{dist}[v]$ update $\text{prev}[v]$</pre>	

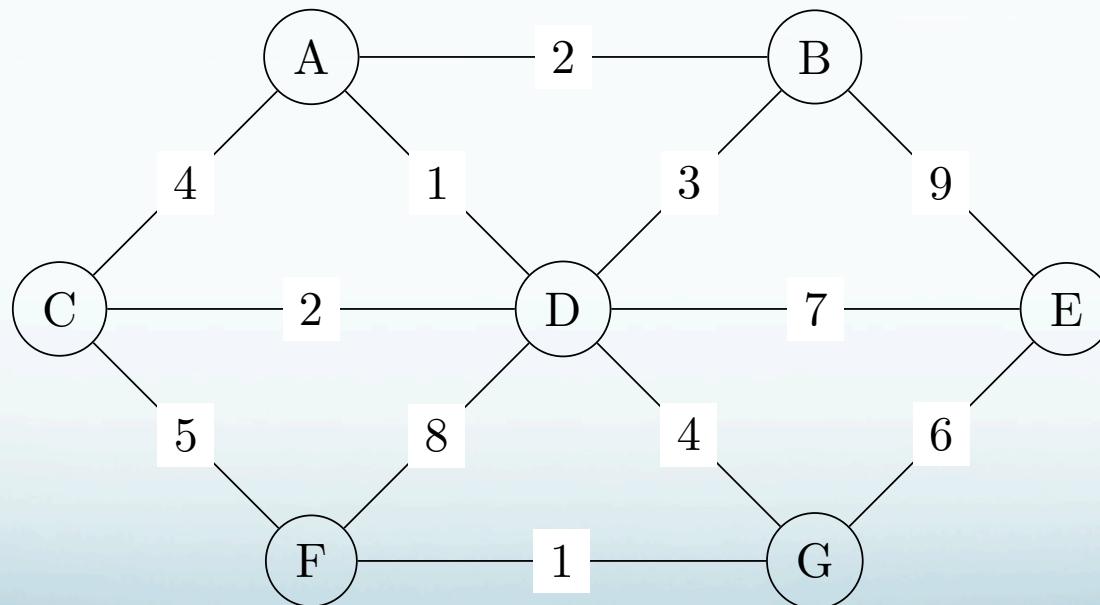
Q: what is update $\text{dist}[v]$? update $\text{prev}[v]$?

why Dijkstra's algorithm is greedy?

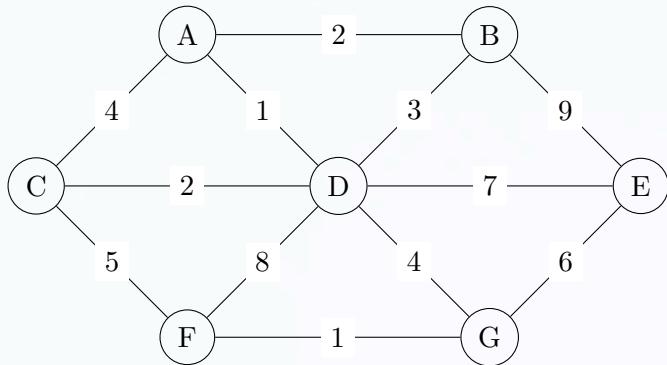
T5: SSSP with Dijkstra's Algorithm (DA)

Dijkstra's algorithm computes the shortest path to each node in a graph from a single starting node (the 'source'). Trace Dijkstra's algorithm on the following graph, with node E as the source

Repeat the algorithm with node A as the source. How long is the shortest path from E to A? How about A to F



Dijkstra's Algorithm from E



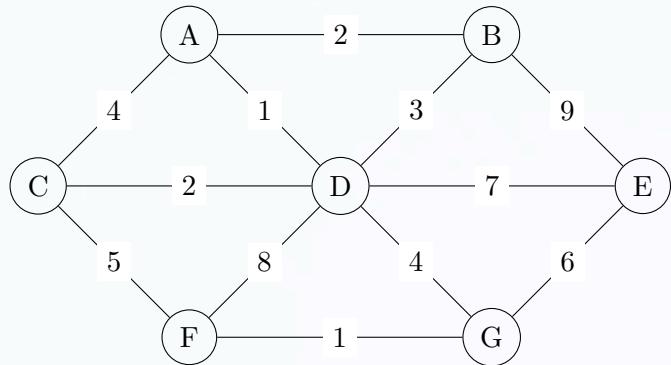
$\text{dist}[E]$

$\text{prev}[E]$
 $(\text{nil}, \text{not shown})$

step	node explored	A	B	C	D	E	F	G
0		∞	∞	∞	∞	0	∞	∞

DA is a BFS. DA is used to find shortest path from a **source** to *all other nodes*. A *practical implementation* is to maintain 3 arrays:

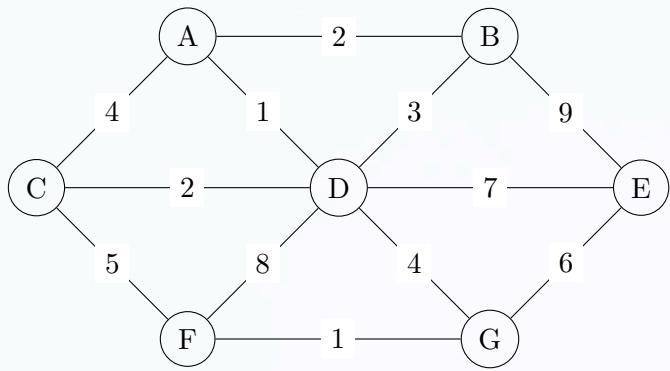
- $\text{dist}[v]$ = shortest-distance-found-so-far from **source** to v
- $\text{prev}[v]$ = id of the node preceding v in the path-found-so-far
- visited**[v] = **Y** if v has been explored/visited, **N** if not yet



DA from E

step	node explored	A	B	C	D	E	F	G
0		∞	∞	∞	∞	0	∞	∞
1	E	∞	9E	∞	7E		∞	6E

At step 1: choose the node with minimal dist ***amongst the unvisited nodes***. This is E. Then, consider to update dist[v] for v= {B, D, G}. And, mark E as visited ???.



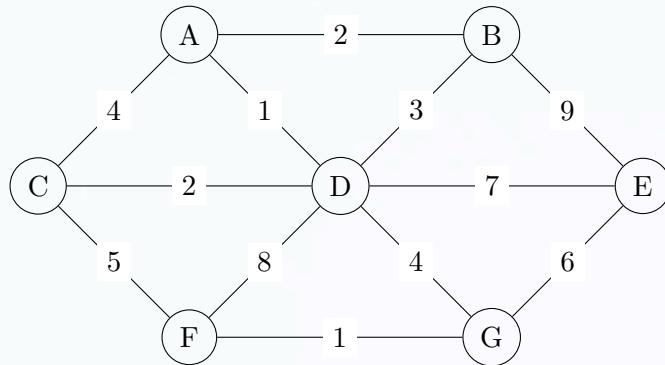
DA from E

How long, and what is, the shortest path from E to A?

cost =

path =

step	node explored	A	B	C	D	E	F	G
0		∞	∞	∞	∞	0	∞	∞
1	E	∞	9E	∞	7E		∞	6E
2	G							

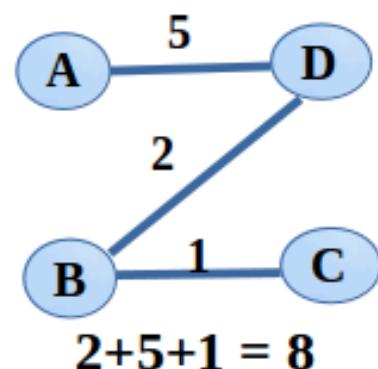
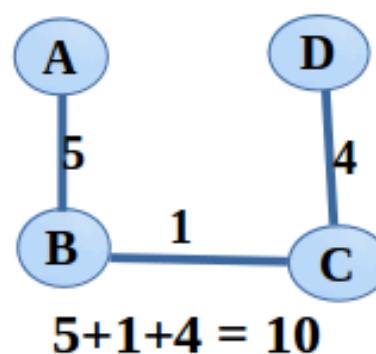
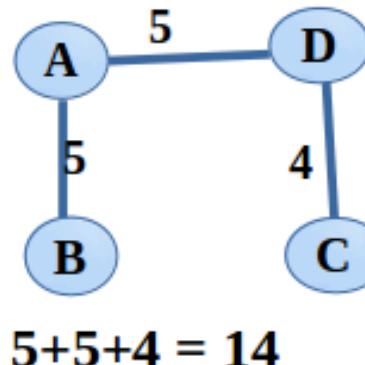
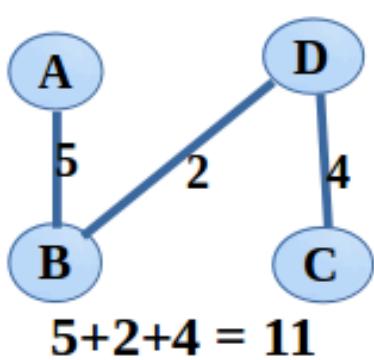
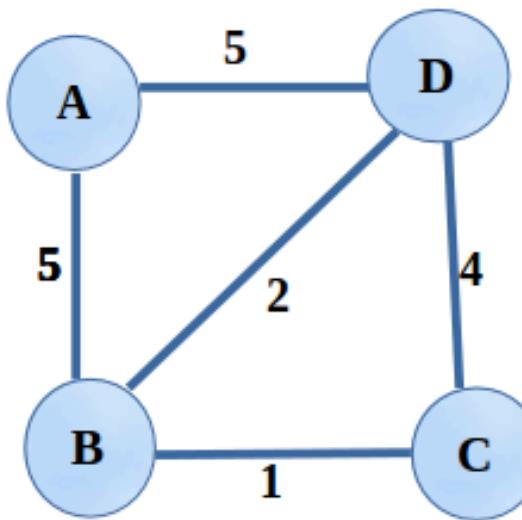


DA from A

How long, and what is, the shortest path from E to A?
 How about from A to F?

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

Prim's Algorithm: Finding a MST of a connected graph



Dijkstra's and Prim's are similar?

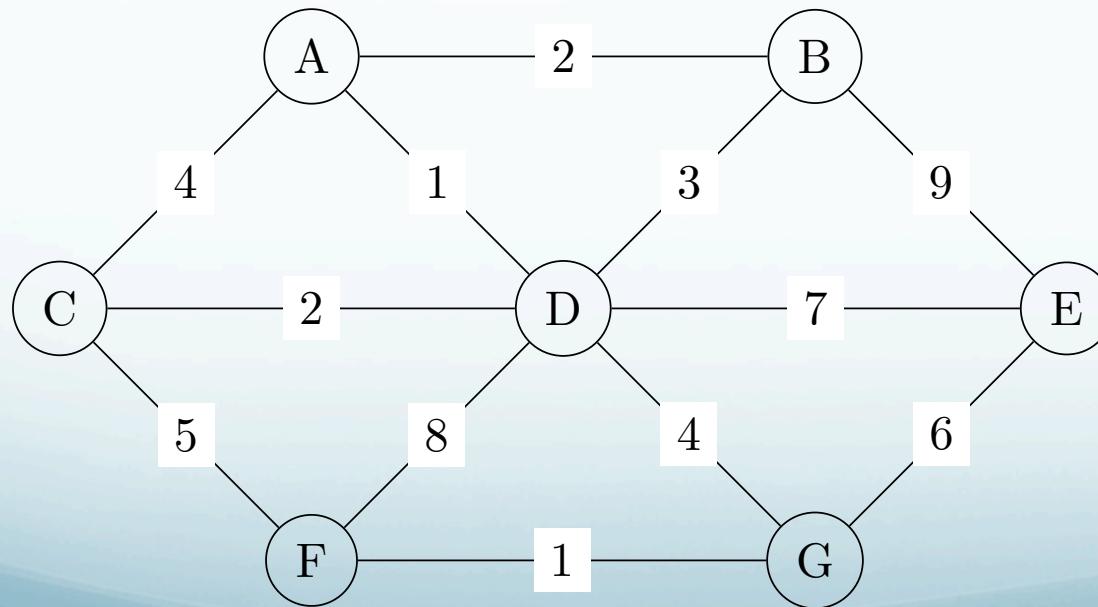
	Dijkstra's	Prim's
purpose	SSSP (that involves all nodes of a connected graph)	MST (that involves all nodes of a connected graph)
init	<pre> for all v ∈ V : dist[v] = ∞ prev[v] = nil dist[S] = 0 </pre>	same
basic loop	for all $u \in V$, at each step choose u that $dist[u]$ is min amongst the unselected	same
action after selecting u	<pre> for all unselected v such that (u,v) ∈ E: if dist[v] > dist[u] + w(u,v): update dist[v] update prev[v] </pre>	same loop to decide whether to include edge (u,v) : <pre> if dist[v] > w(u,v): same </pre>

So, the 2 alg are similar. The only difference is the optimizing functions. There are many other situations when the same algorithm applied, but perhaps with a different optimizing function.

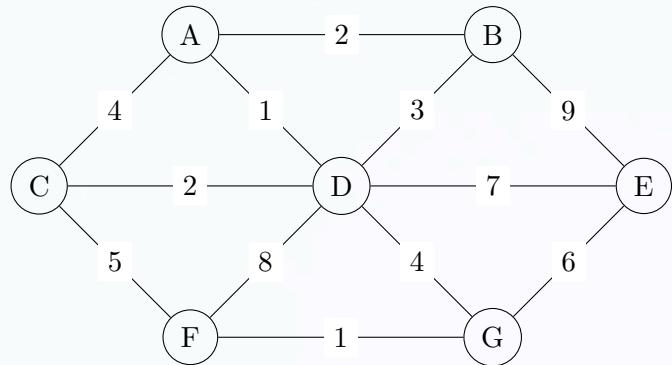
T6: Minimum Spanning Tree with Prim's Algorithm

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?



Prim's Alg from A



What's the resulting MST?
What's the cost of that MST?

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

Food for our brain

- You're organizing your birthday party and inviting n friends. From these friends you know that there are some pairs of people who hate each other. You want to know if it's possible to divide guests into 2 different tables so that in each table there is no such hating pair. Design an algorithm for that.
- You have an unweighted undirected graph. Design an algorithm to find the shortest part (the part with least number of edges) between 2 vertices. What if the graph is weighted?

assignment 1

- Do it early! Submit early, resubmit if needed!
- Read & participate in discussion forum!
- Make sure that you follow well the specification.
- Check your writing part carefully..
- Test your program carefully: remember to test on dimefox
- Make sure you don't have memory leak:
 - check that every execution of `malloc` matches with an execution of `free`, and
 - [optional] use tools like `valgrind` to test for memory leak.

How to test/submit the programming part

Suppose that all all of your files are in directory **A1**. To submit:

- When on the parent directory of **A1**, copy the whole **A1** to **dimefox** using **scp** (or **pscp**, but perhaps you should install **scp**):

```
scp -r A1 username@dimefox.eng.unimelb.edu.au:
```

```
pscp -r A1 username@dimefox.eng.unimelb.edu.au:
```

- Then, login into **dimefox** using **ssh**, after that, in **dimefox**, test your program with **make**, and do the test with sample data and sample outputs

- To test, when on **dimefox** run:

```
cd ~/A1
```

```
make clean
```

```
make
```

```
./a1 p2a < tests/p2a-in-1.txt >my-p2a-out-1.txt
```

```
<compare to see that my-p2a-out-1.txt is the same as tests/p2a-
out-1.txt>
```

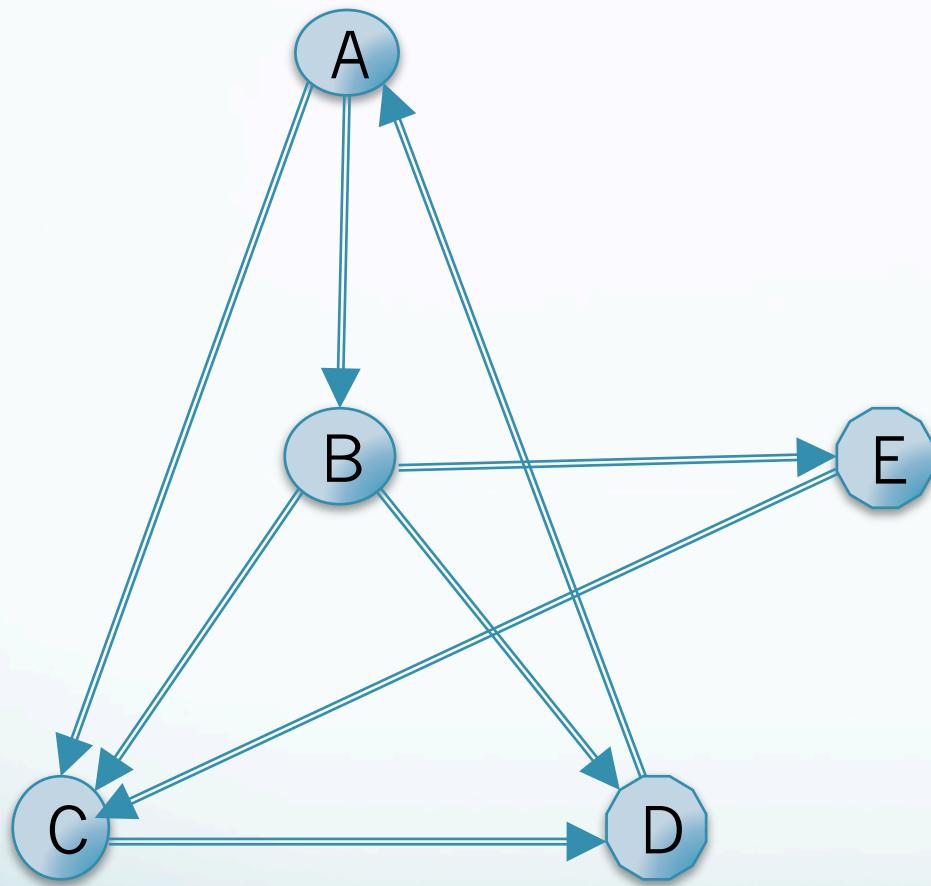
- To submit, when on **dimefox** run:

```
cd ~/A1
```

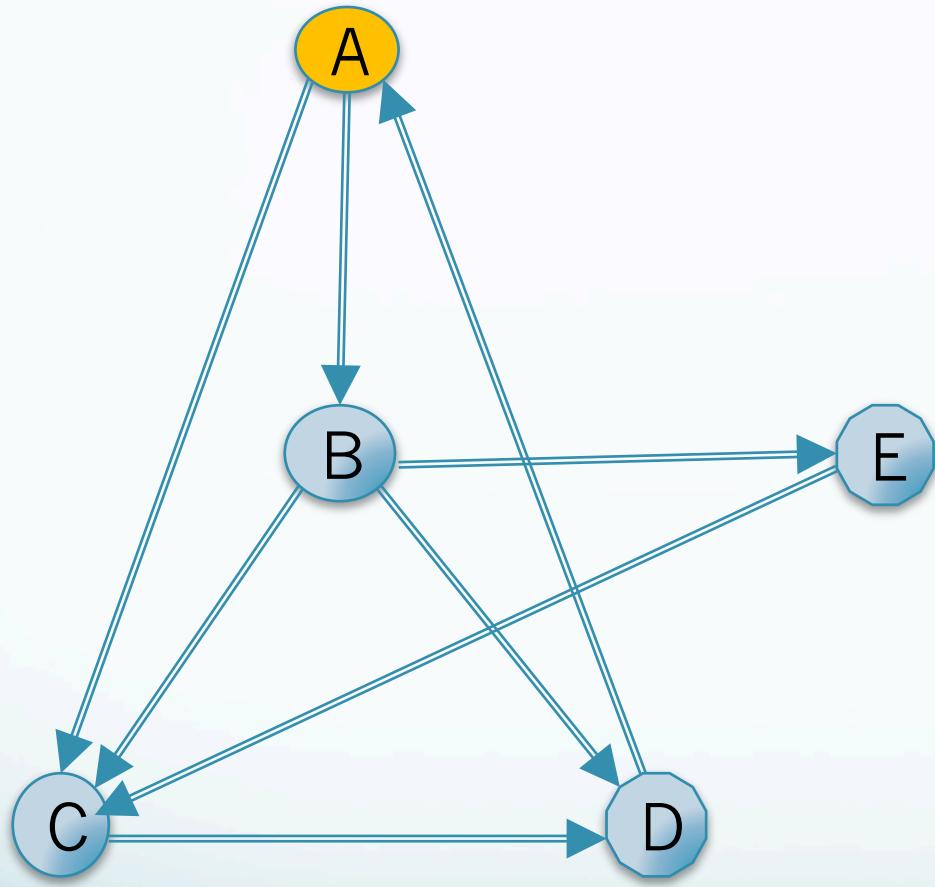
```
submit comp20007 a1 Makefile *.c *.h
```

Lab: do assmt1 or your choice

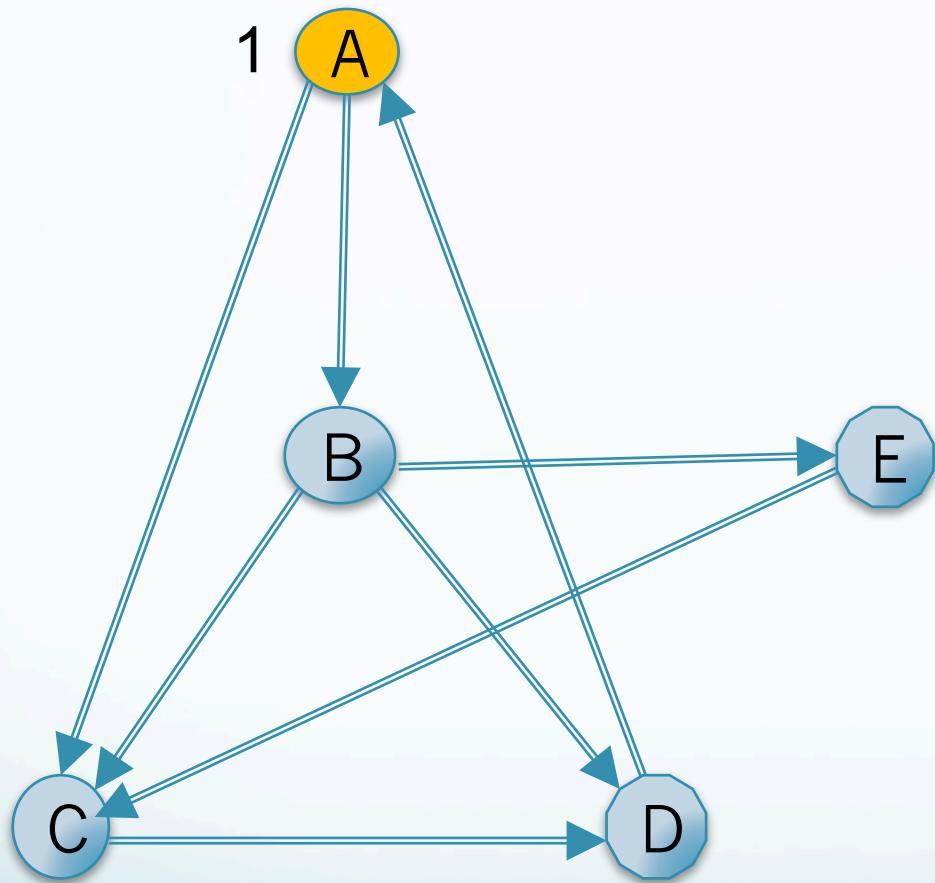
Extra: understanding DFS on di-graphs and the concepts of tree-, back-, forward-, cross-edges



DFS

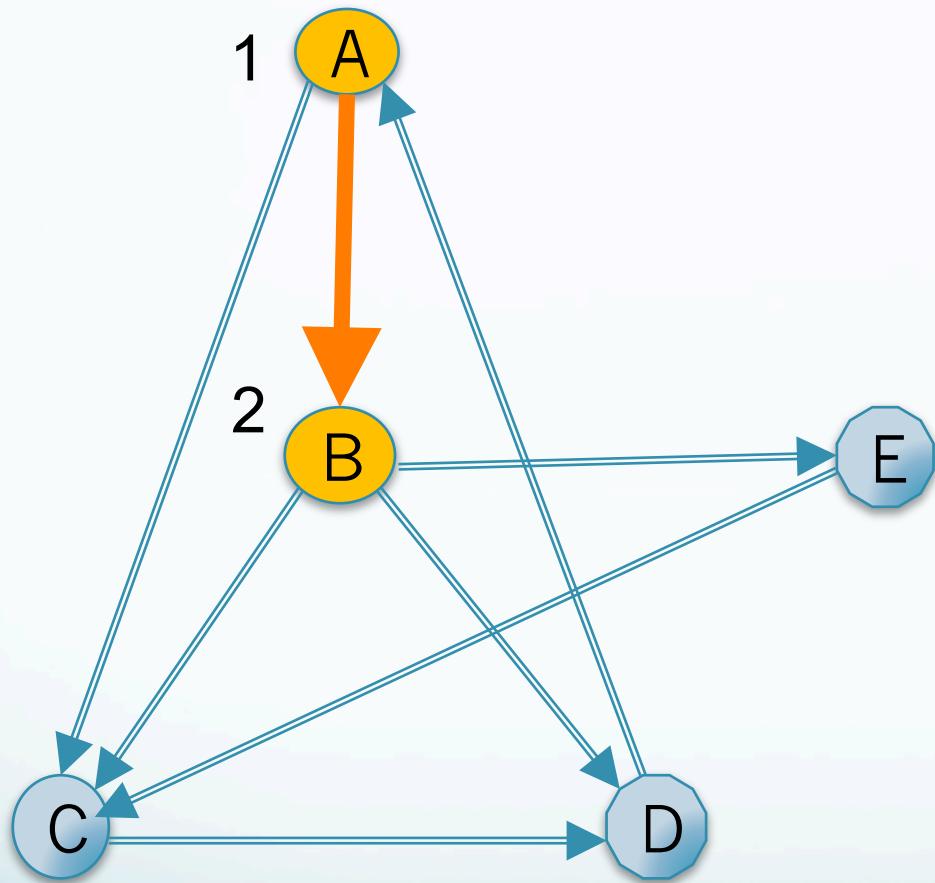


DFS

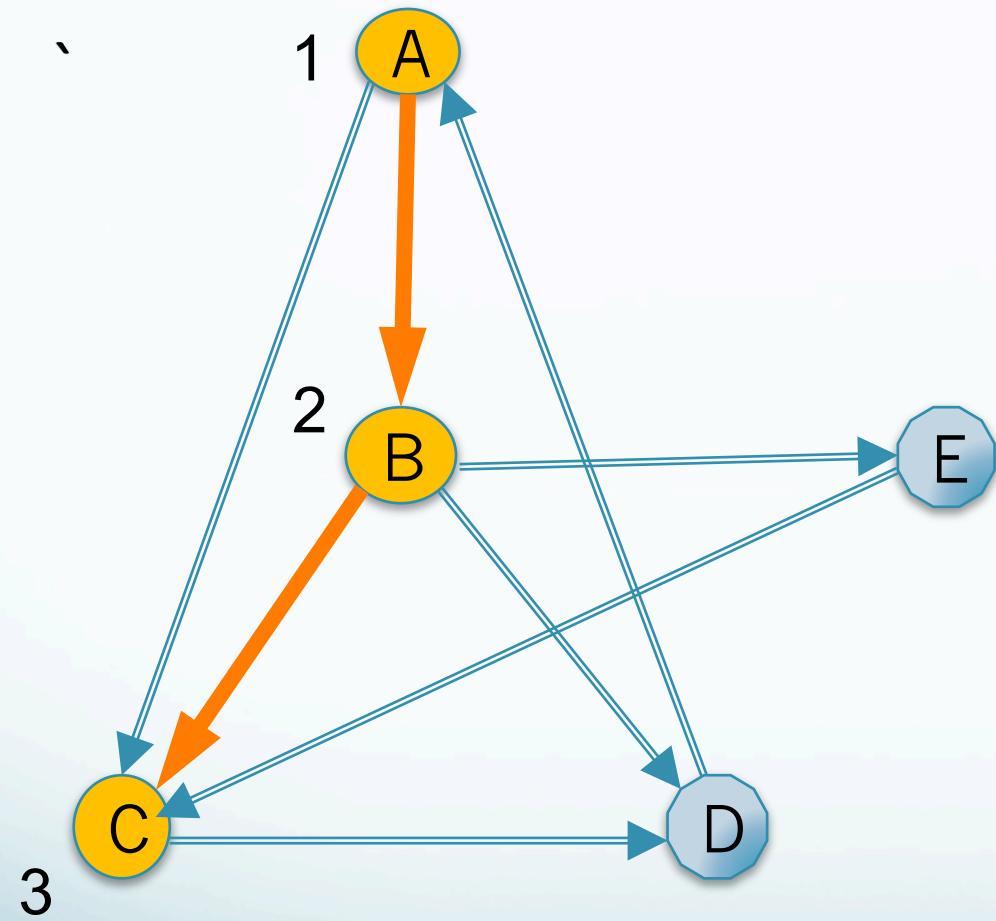


note: black number 1 is the push order

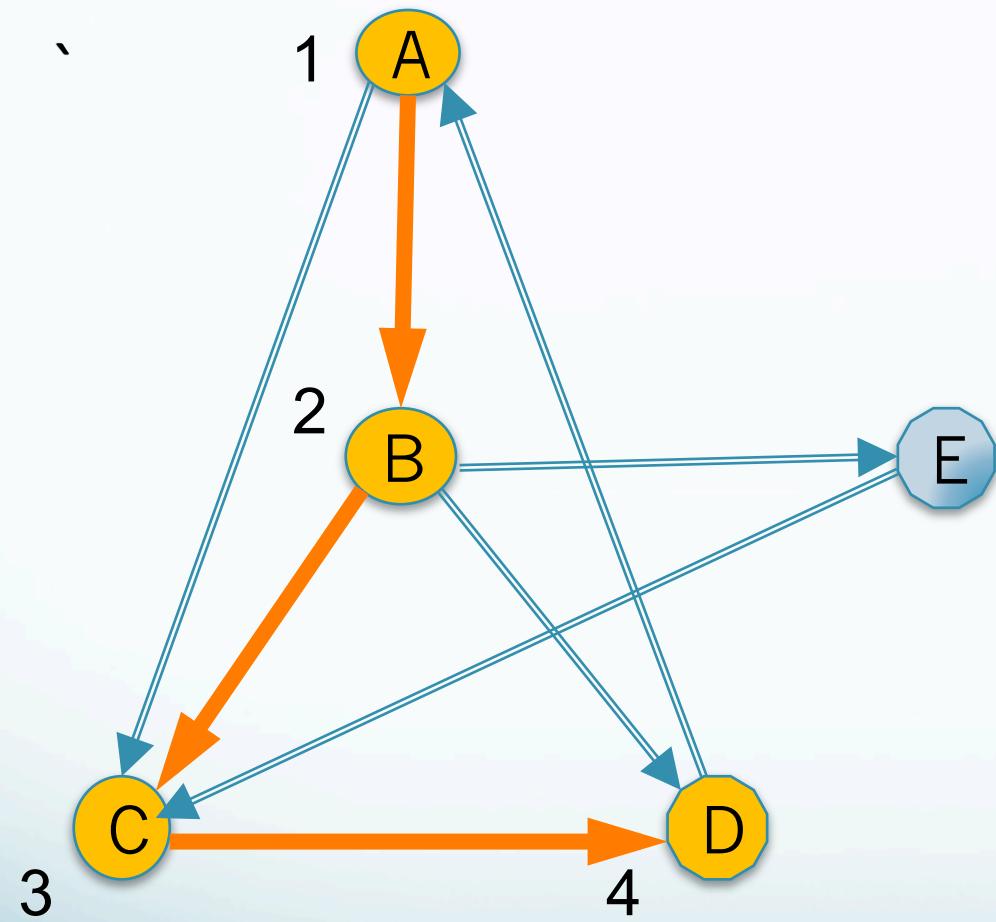
DFS



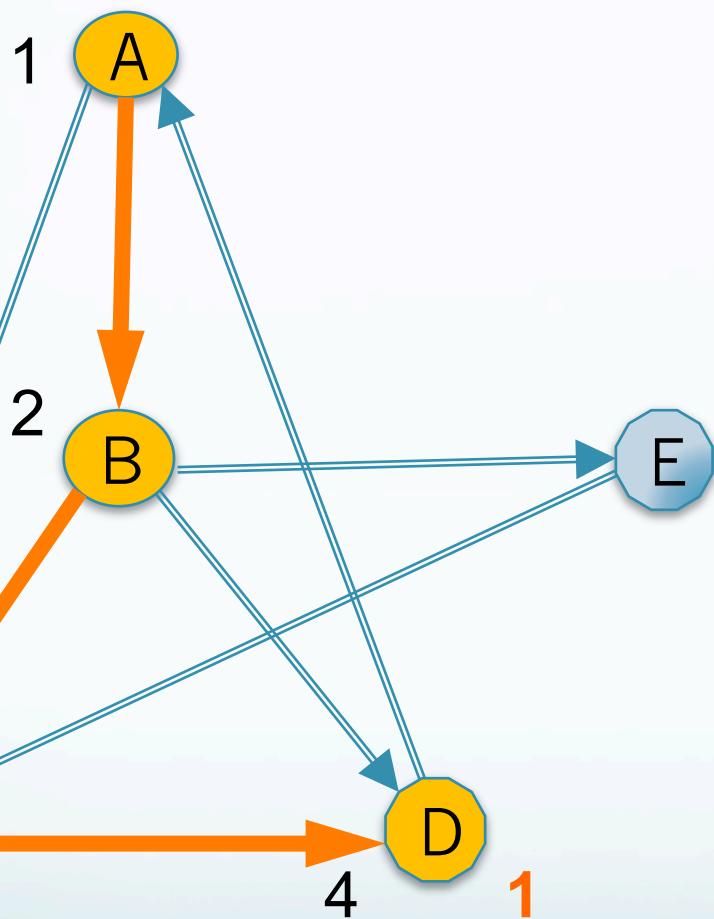
DFS



DFS

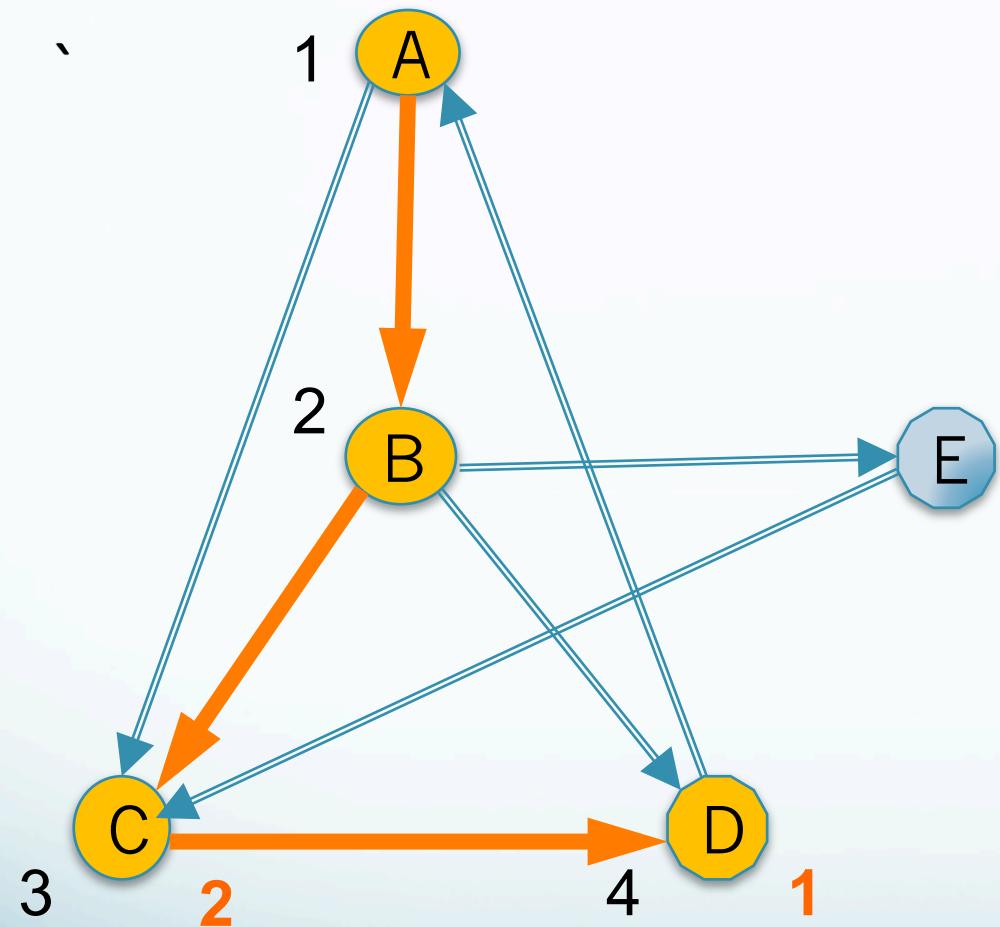


DFS

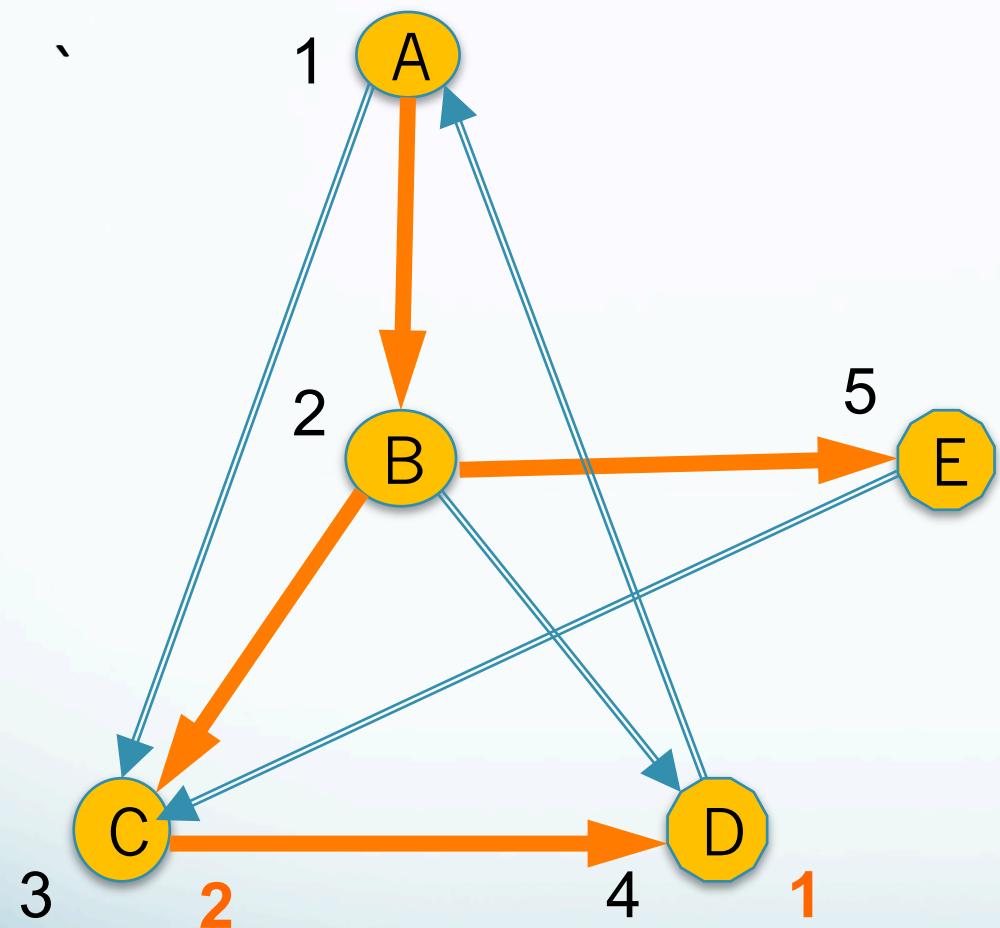


note: orange number **1** represents the pop order

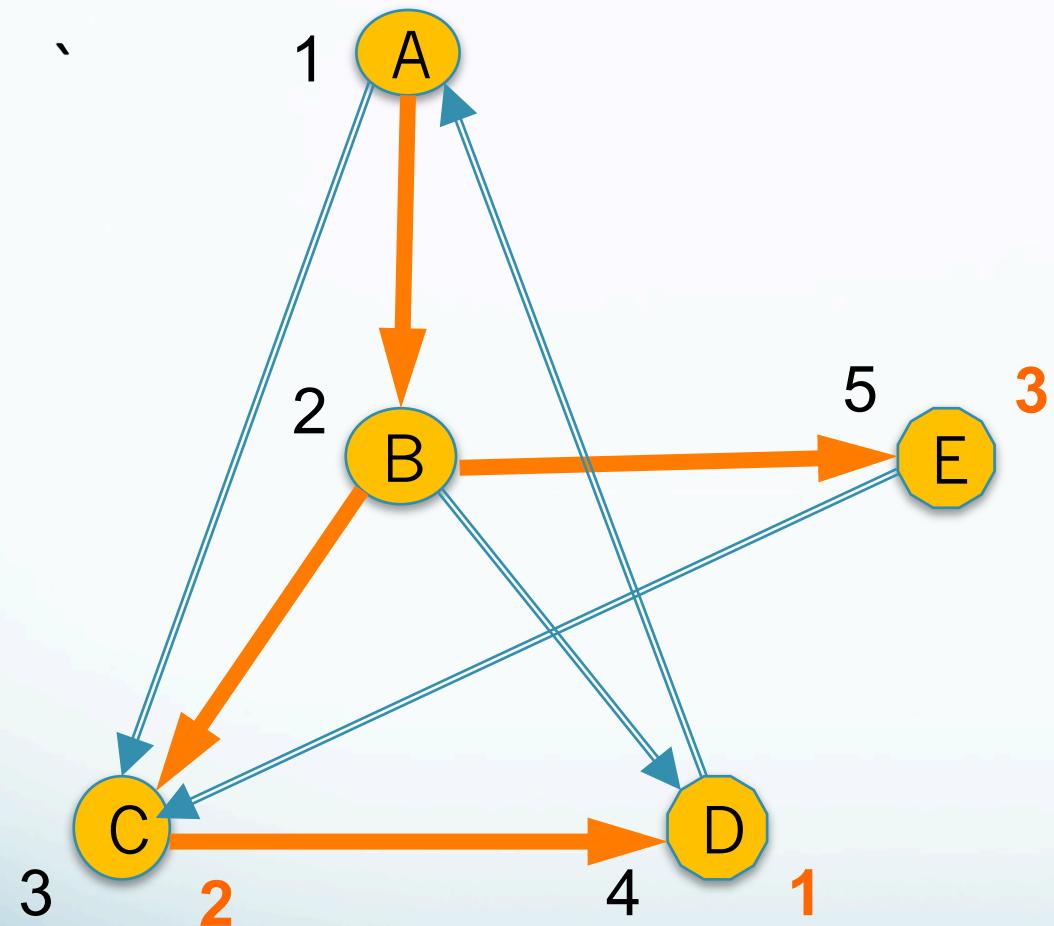
DFS



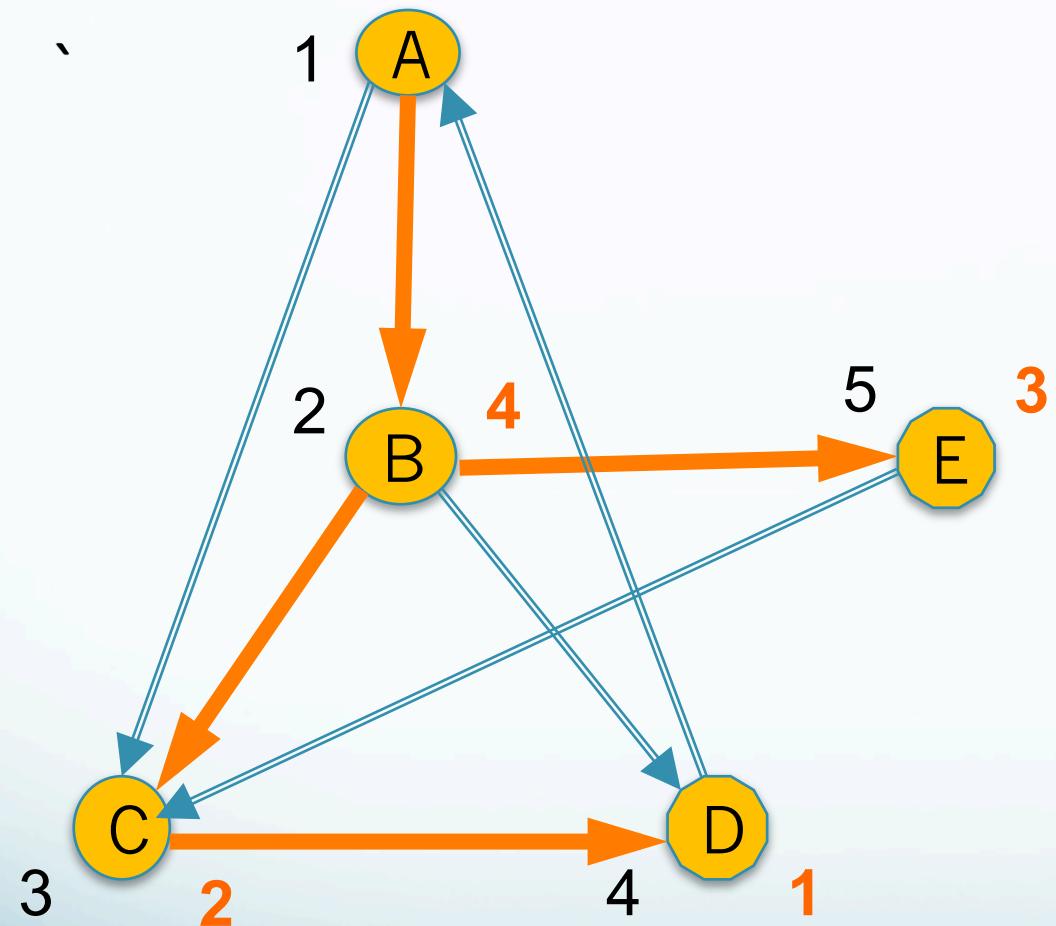
DFS



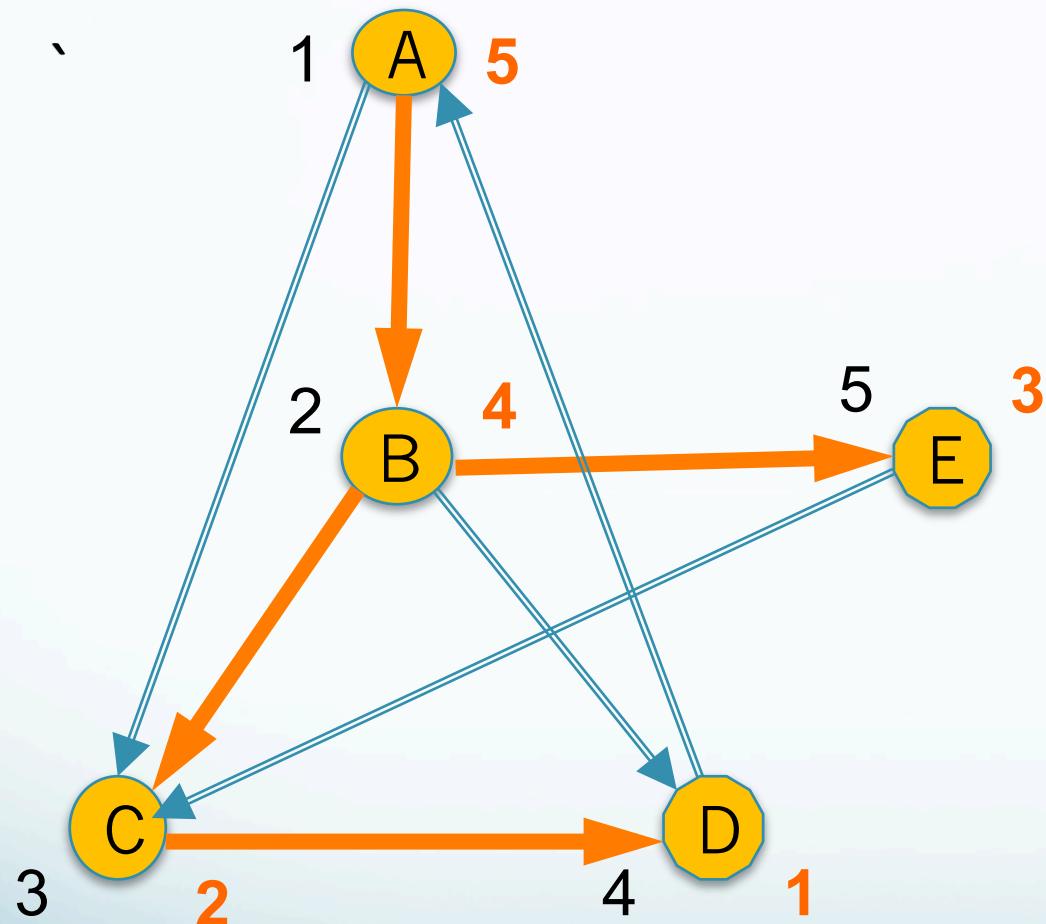
DFS



DFS



DFS

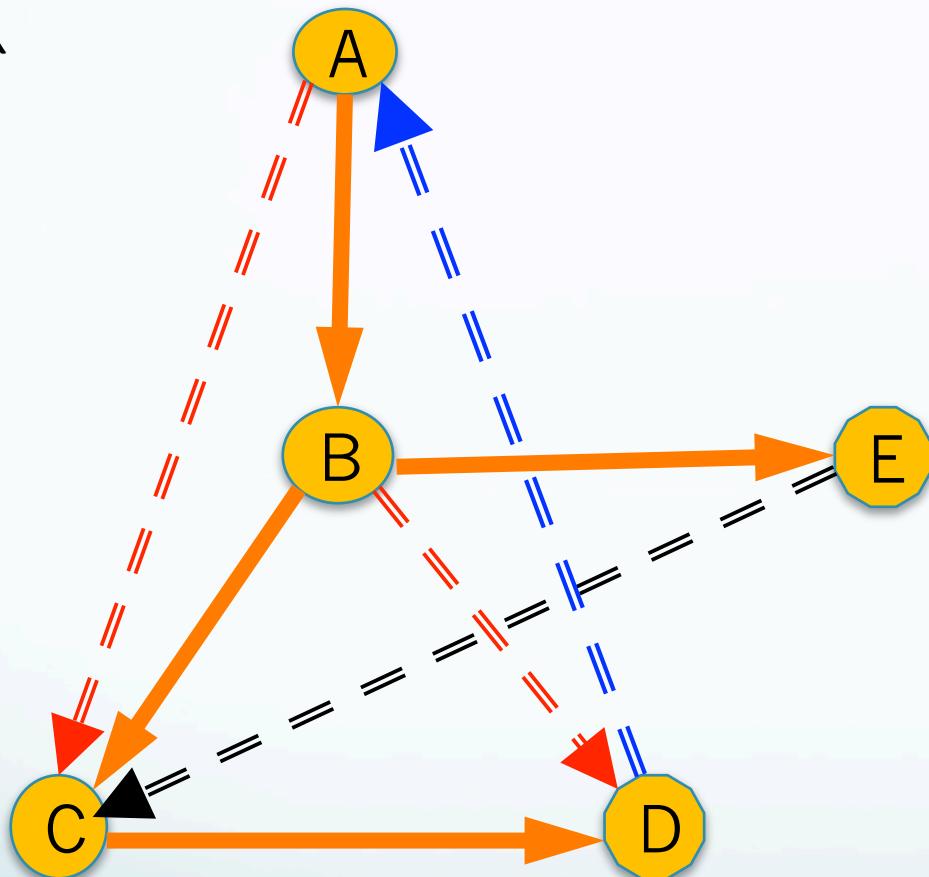


Orange edges are tree edges.

Not used edges:

- all would be back-edges if the graph was un-directed

DFS



Orange edges are tree edges.

Not used edges:

- red are *forward* edges
- blue are *back-edges*
- other dashed are *cross edges*

Why don't we have forward and cross edges in undirected graph?