COMP20007 Workshop Week 5

1	**Questions on graph representation?**
2	**Topic 1: Path Finding and Graph Traversal** Q5.4, Q5.6, Q5.7 Homework: Q 5.5, Q 5.7 ?
3	**Topic 2:** MST, Paths & Shortest Paths Dijkstra's (Greedy) Algorithm, Q5.9 ***Probably Delayed to Next Week:*** Prim's Algorithm Q5.8 and some others?
4	**A1 : Q&A**

Coming Soon

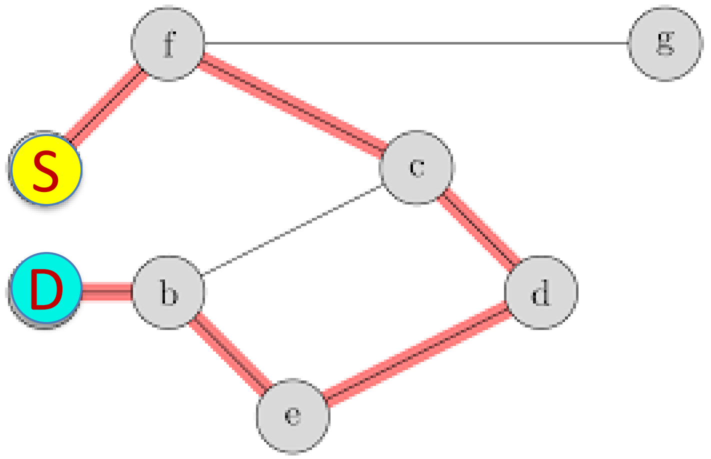
Assignment 1:
- due Mon Week 6: it might take heap of time to finish!

MST:
- Tuesday Week 7 ???
- Review & bring questions to the next workshop

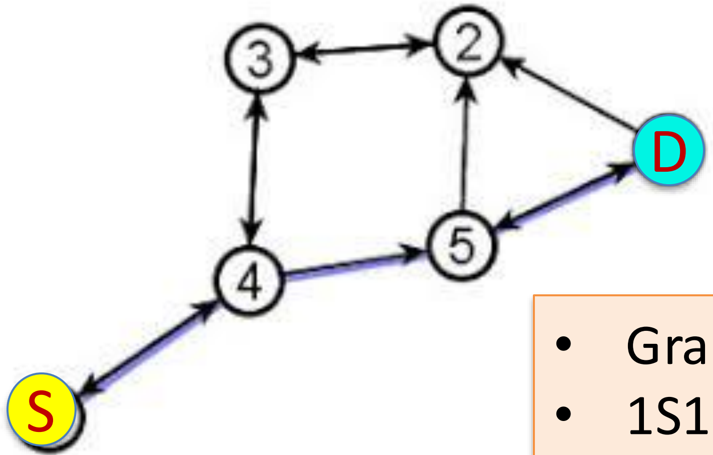
Paths in unweighted graphs: path length, shortest path

Path= S → f → c → b → D length = 4

S → f → c → d → e → b → D length = 6

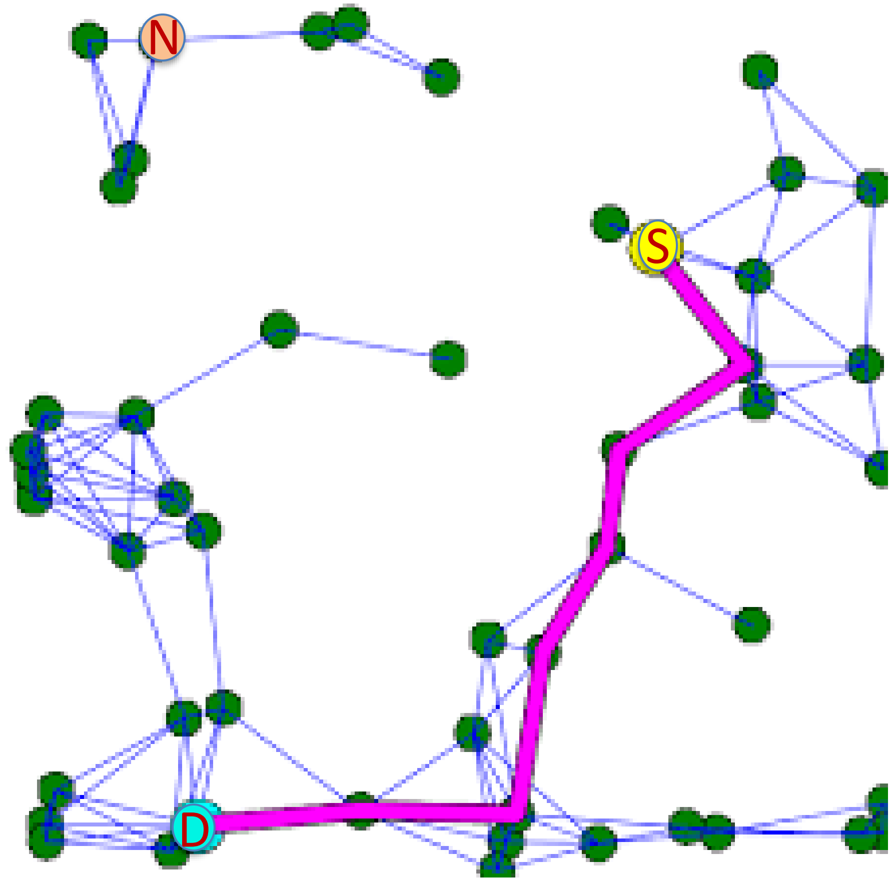


Path= S → 4 → 5 → T, length= 3



The pink path is a shortest path from S to D

There is no path from S to N



- Graph Search = systematic exploration of the vertices and edges of a graph.
- 1S1D = using graph search to find a path from 1 vertex S to 1 vertex D
- 1S*D = using graph search to find a path from 1 vertex S to all other vertices

DFS and BFS

Graph Search = Graph Traversal
= systematic exploration of the vertices and edges of a graph.

How?

When visiting (e.g. exploring) a vertex we store its unvistited neigbours in a data structure so that we can visit (or try) all the neigbours later in a systematic way.

Two basic ways:

DFS = using a stack as the data structure for storing unvisited vertices

BFS = using a queue instead

DFS for 1S1D (in unweighted graphs)

```
function DFS_1S1D(G=(V,E), S, D)
  mark each node in V as unvisited
  stack := make_empty_stack
  push S into stack
  while stack is not empty
    v := pop from stack
    if v = D
      return "path S→D found”
    if v is unvisited
      mark v as visited
      push all unvisited neighbours of v into stack
  return PATHNOTFOUND
```

BFS for 1S1D

```
function BFS_1S1D...
```

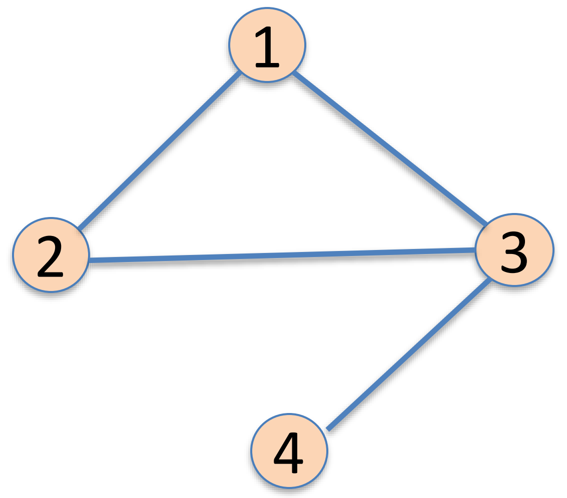
same as

DFS_1S1D(G=(V,E), S, D)

just need to replace:

- stack with queue
- push with enqueue
- pop with dequeue

G=



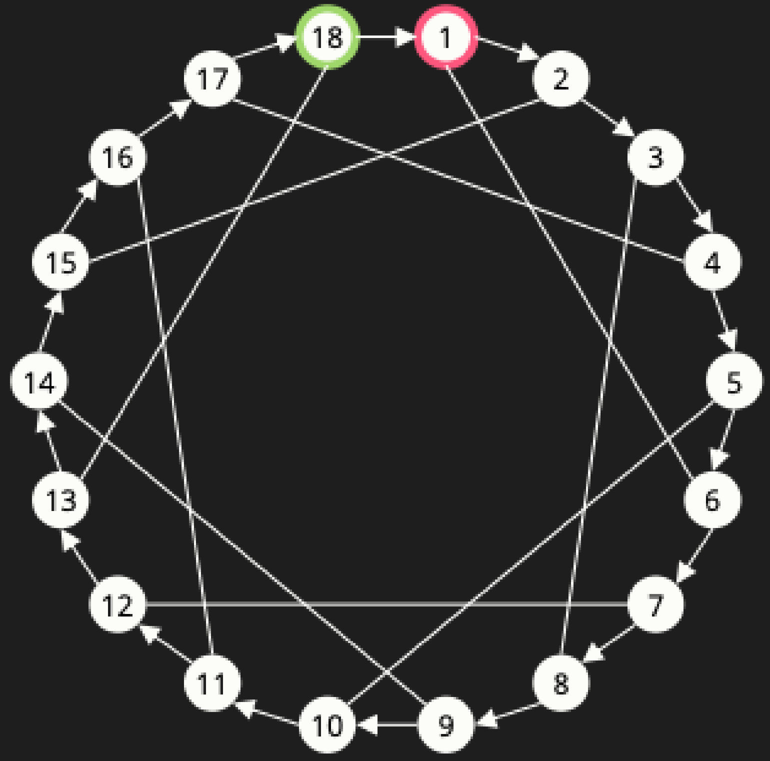
Example: Compare the executions of
DFS_1S1D(G, 1, 4)
and
BFS_1S1D(G, 1, 4)

Question: *How to return the path from S to D?*
*How to adapt for the 1S*D task?*

Peer Activity: Shortest Path Search

Which search algorithm guarantees the shortest path between the nodes? Why?

a. Breadth-first search
b. Depth-first search



DFS for Graph Traversal

DFS algorithm – as in lecture

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

function DFsExplore(v)
  count := count + 1
  mark v with count
  for each edge (v,w) do
    if w is marked with 0 then
      DfsExplore(w)
```

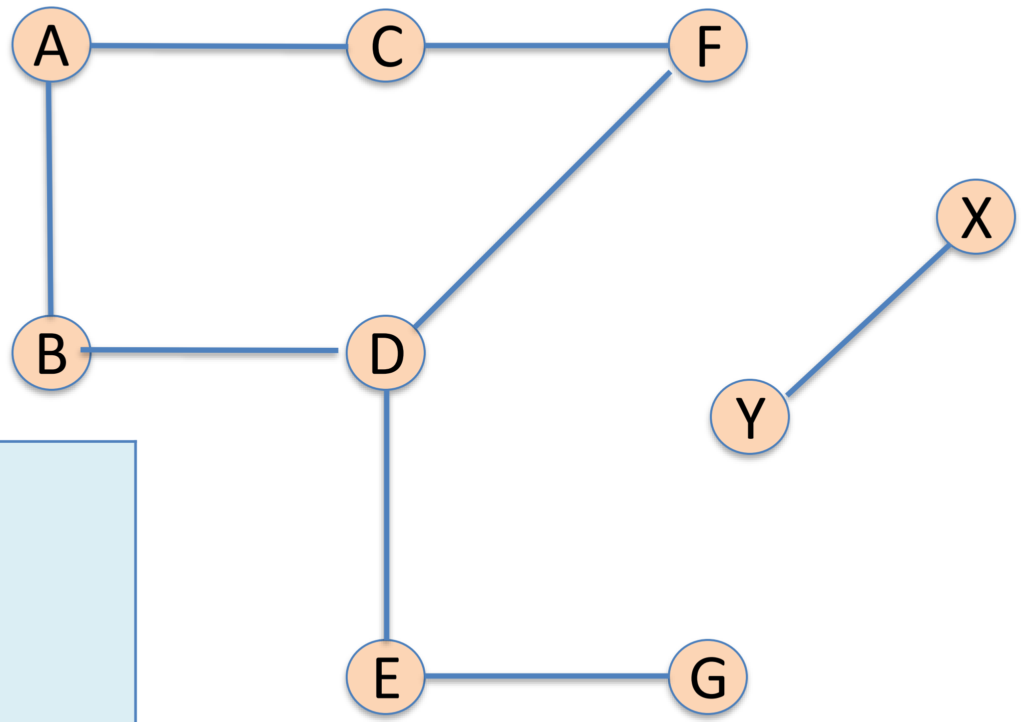
The loop in DFS is to ensure that all connected components of a graph are explored. Note that DFSExplore(v) explores and can only explore all nodes that can be reached from v by a path.

DFsExplore(v) is similar to DFS_1S1D but:

- recursion is employed instead of using a stack
- NO early termination when v=D
- Here, "count" is used as a timestamp for the push-order

Q5.4 (modified): DFS & BFS

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



YOUR ANSWER:

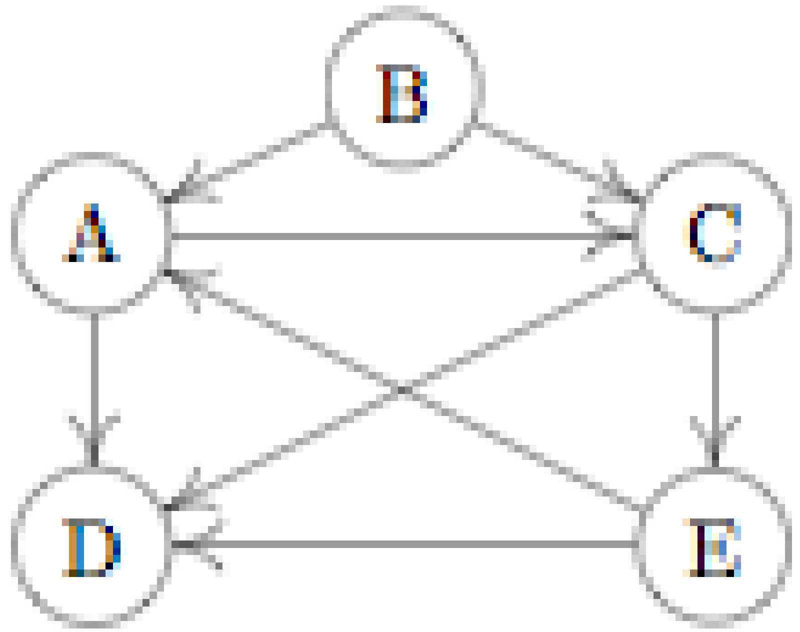
a) The order of the nodes visited by DFS is:

A

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

A

Q5.5: [Homework] 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 DFS to see whether an undirected graph is cyclic.

Transform function DFS to:

isCyclic(G=(V,E))

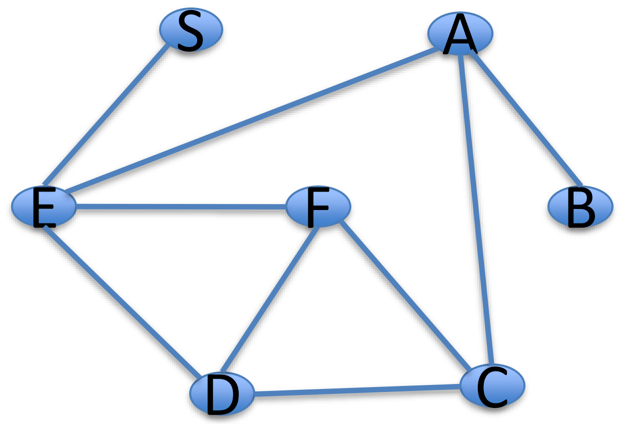
b) Can we use BFS for the task?

Which one is better: DFS or BFS?

DFS algorithm – as in lecture
but only mark node with 1 if visited

```
// Aim: transform this to isCyclic
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?
```

Q5.7: 2-Colourability – possibly homework



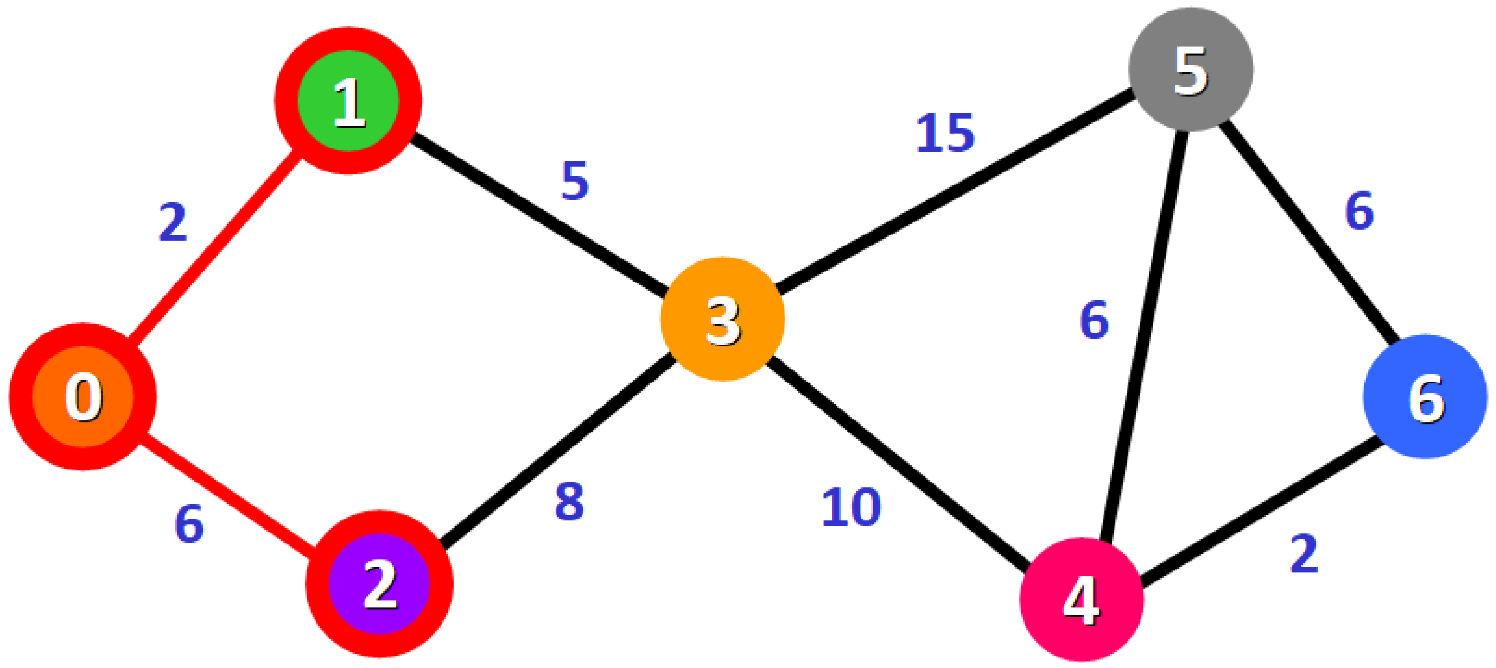
Design an algorithm to check whether an undirected graph is 2-colourable, that is, whether its nodes can be coloured with just 2 colours in such a way that no edge connects two nodes of the same colour.

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?

```
// Aim: 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?
```

Shortest path on weighted graphs



Path from 0 to 3:

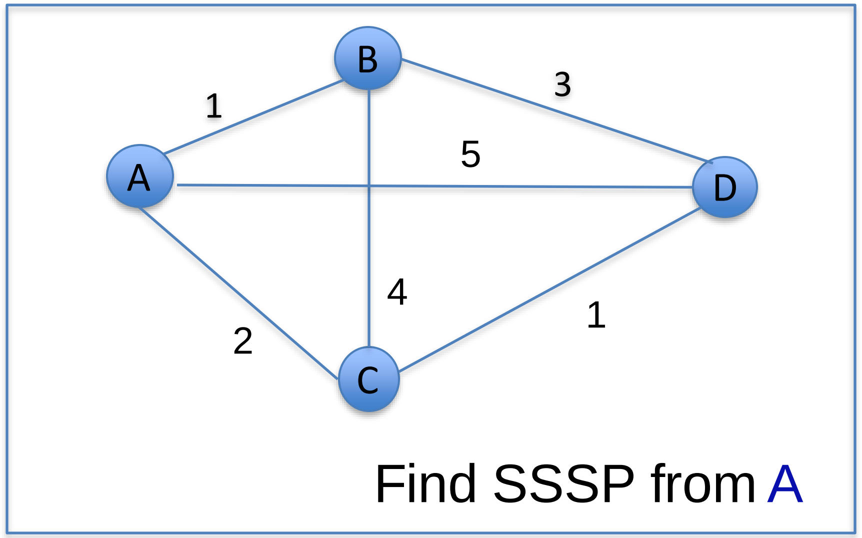
- *possible paths*:
  - 0→1→3,
  - 0→2→3
- *shortest path*: 0→1→3 with total weight= 7

Dijkstra’s Algorithm: 1S*D, e.g. Single Source Shortest Path (SSSP)

The task:

- Given a weighted graph G=(V,E,w(E)), and s∈V, and supposing that *all weights are positive*.
- Find shortest path (path with min total weight / min distance) from s to all other vertices.

Input



Output

Destination	Shortest Path Path	 Length
A	A→A	0
B	A→B	1
C	A→C	2
D	A→C→D	3

- Solution: using Dijkstra’s Algorithm!
- Q: Can the Dijkstra’s Algorithm be used for the SSSP in unweighted graphs?

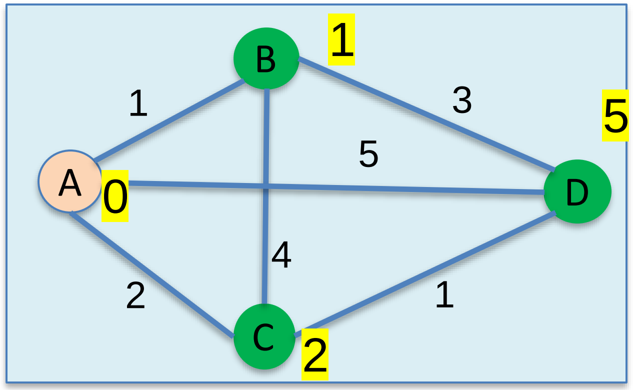
Dijkstra's Algorithm is similar to BFS, but it uses a priority queue instead of a FIFO queue.
We start from A (node 0), and shortest-distance-so-far to all other nodes is ∞

dist[]= {0, ∞, ∞, ∞}

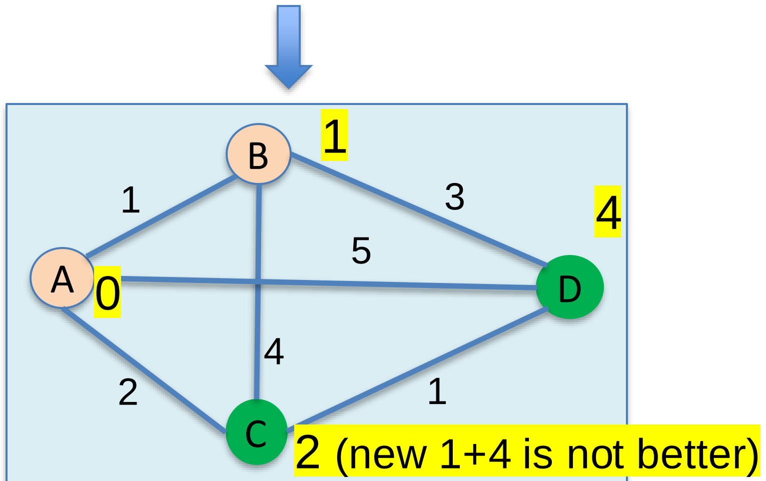
We use the edges from A to update the shortest-distance-so-far

dist[]= {0, 1, 2, 5}

image: https://medium.com/analytics-vidhya/what-is-the-greedy-algorithm-5ed71f9a7b3a



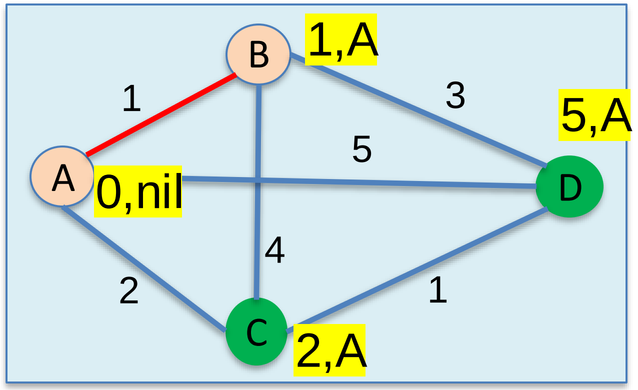
What's next?
explore C, B, or D?



- ➔ Using this greedy policy, shortest path from A to B found with dist[B]= 1.
- ➔ For retrieving the actual path, we also need to keep prev[B]=A.
- ➔ continue this way to update dist[] for other nodes

Keep track of our steps: for each v maintain

- dist[v]: shortest distance-so-far from A to v
- prev[v]: node that precedes v in the path



this column: nodes with found shortest path

	A	B	C	D
	0, nil	∞,nil	∞,nil	∞,nil
A		**1,A**	2,A	5,A
B				

prev[D]= A node that precedes D in the path A→D

dist[D]= 5 shortest-so-far distance from A

queue {B,C,D} includes not-yet-done elements at this stage.
From the queue we remove B - the one with lowest dist, so shortest path A→B found

From the queue, we always remove the element with min value ➔ for better efficiency, we use a priority queue instead of a normal FIFO queue.

Dijkstra's algorithm

```
set dist[u]:= ∞, prev[u]:=nil for all u
dist[s]:= 0
PQ:= makePQ(V) using dist[] as the weight
while (PQ not empty)
        u:= deleteMin(PQ)
        visit  u
        for all (u,v) in G:
    if (dist[u]+w(u,v)<dist[v]):
        update dist[v] and pred[v]
```

A	B	C	D
0,nil	∞,nil	∞,nil	∞,nil
	1,A	2,A	5,A
		2,A	4,B
			3,C

Programming note:
Here PQ is a priority queue.
"update dist[v] and pred[v]" means:

1. dist[v]:= dist[u]+w(u,v), pred[v]:= u
2. decrease the weight of v in PQ to the new dist[v].

For complexity, we will see:

- makePQ of n elems: $O(n)$
- delemin(PQ): $O(\log n)$
- change a weight in PQ: $O(\log n)$

So:

Complexity of DA is: $O(\ ?\ \ )$

Dijkstra's algorithm: a variation for finding shortest path for 1S1D

```
set dist[u]:= ∞, prev[u]:=nil, visit[u]:=0 for all u
dist[S]:= 0

~~set PQ= makePQ(V) using dist[] as the weight~~
PQ:= new empty PQ
enPQ(S, dist[S])

while (PQ not empty)
      u:= deleteMin(PQ)
      if visit[u]=1 continue
  visit[u]= 1
  if u=D break;   #path S→D found

      for all (u,v) in G:
    if (dist[u]+w(u,v)<dist[v]):
      update dist[v] and pred[v]
      PQ := enPQ(v, dist[v])
cost := dist[D]
path := < use prev[] to work out >
return (cost, path)
```

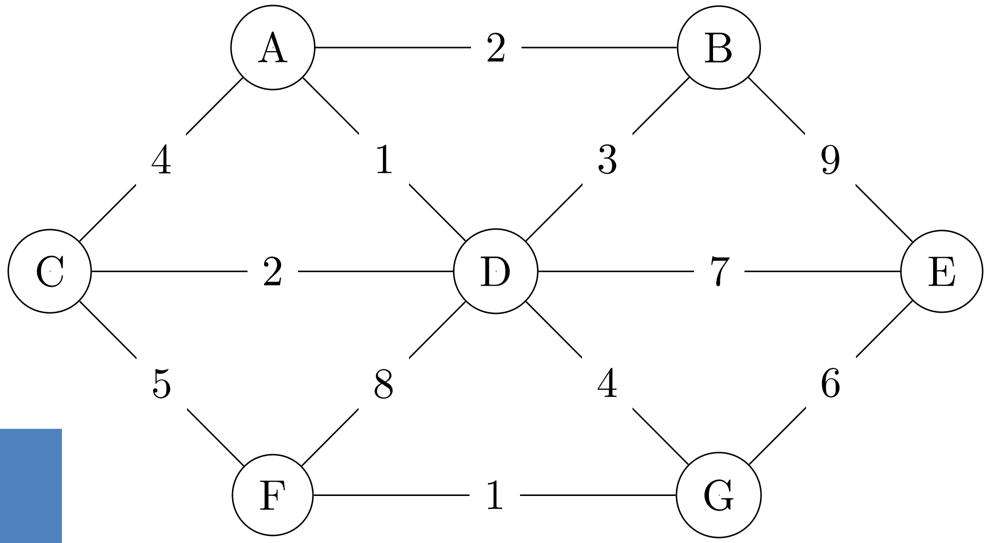
SIDE NOTES

Complexity of DA is O((V+E)log V), which is O(V log V) for sparse graphs

- DA does not work if there are edges with negative weight
- Bellman-Ford algorithm can be used if there are negative weights and no negative cycle, but the complexity is higher than that of DA
- DA cannot be adapted for finding a longest path

Q5.9 a): SSSP with Dijkstra's Algorithm

Trace Dijkstra's algorithm on the following graph, with node E as the source.
How long, and what, is the shortest path from E to A?



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

LAB

- question/work on assignment 1
- or, work on not-yet-done this week's workshop problems

Notes:

Before the next workshop

- Review all topics in workshops week 2 – week 5,
- Try sample MST papers

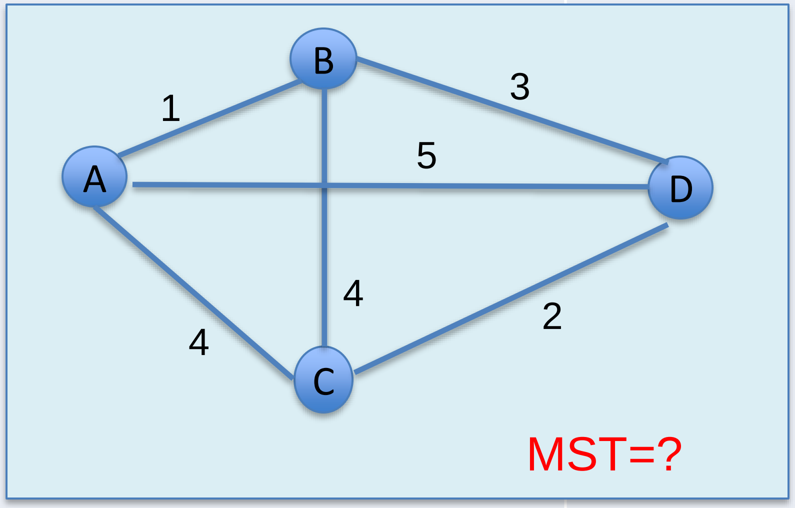
Bring questions to the workshop.

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?



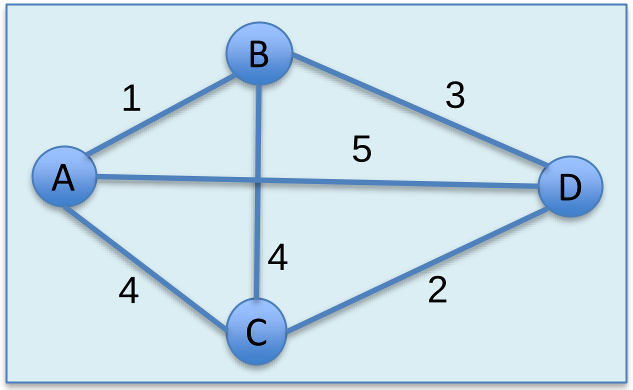
Dijkstra’s and Prim’s are similar

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**.

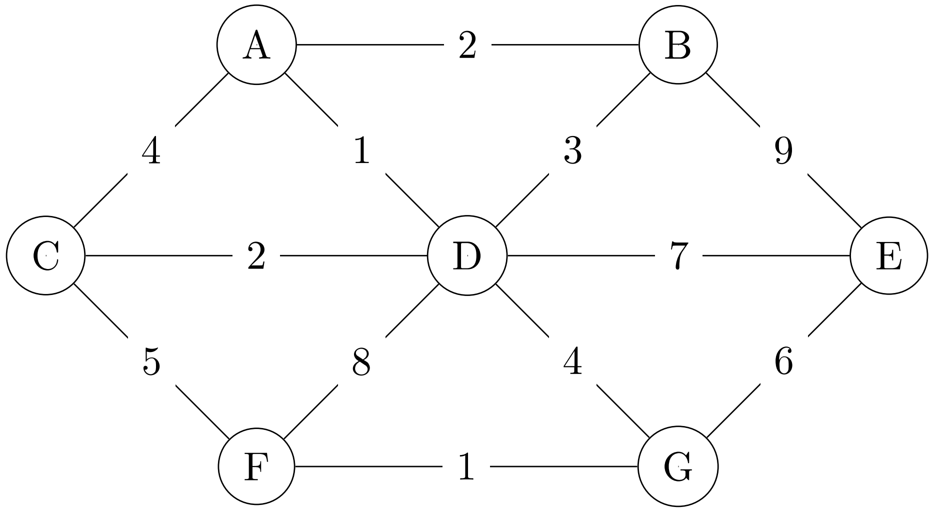
We start with A according to the alphabetical order.

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?



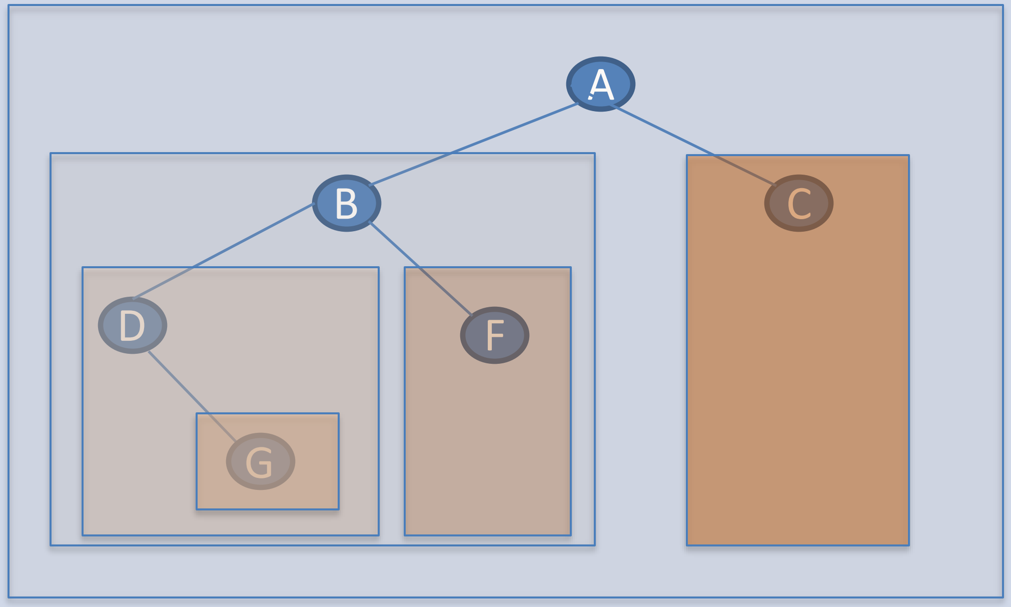
step	node done	A	B	C	D	E	F	G
0		0/nil	∞/nil	∞/nil	∞/nil	∞/nil	∞/nil	∞/nil
1								
2								
3								
4								
5								
6								
7								

Additional Slides

DFS vs BFS

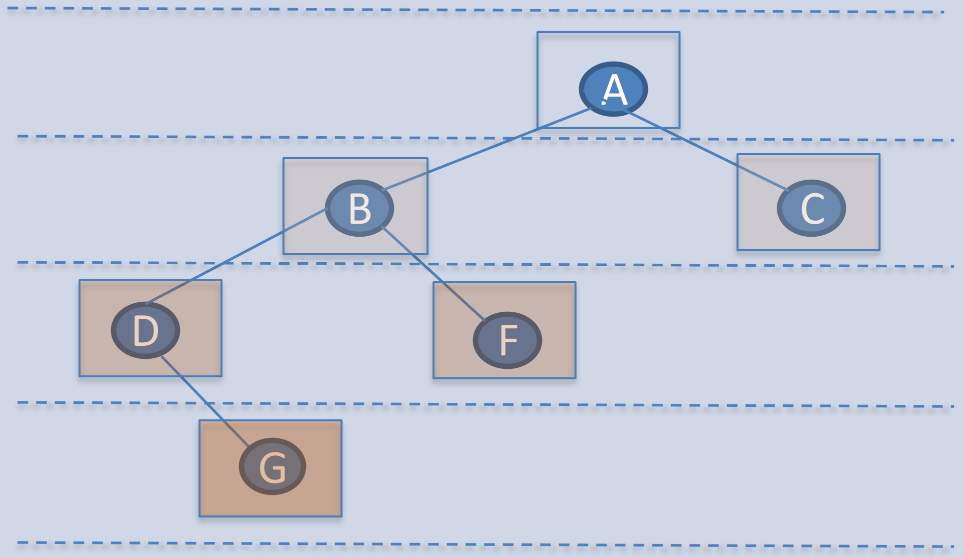
DfsExplore(v)

```
#start visit v
visit v
mark v as visited


for each neighbor w of v
  if !visited(w)
    DfsExplore(w)

#end visit v
```



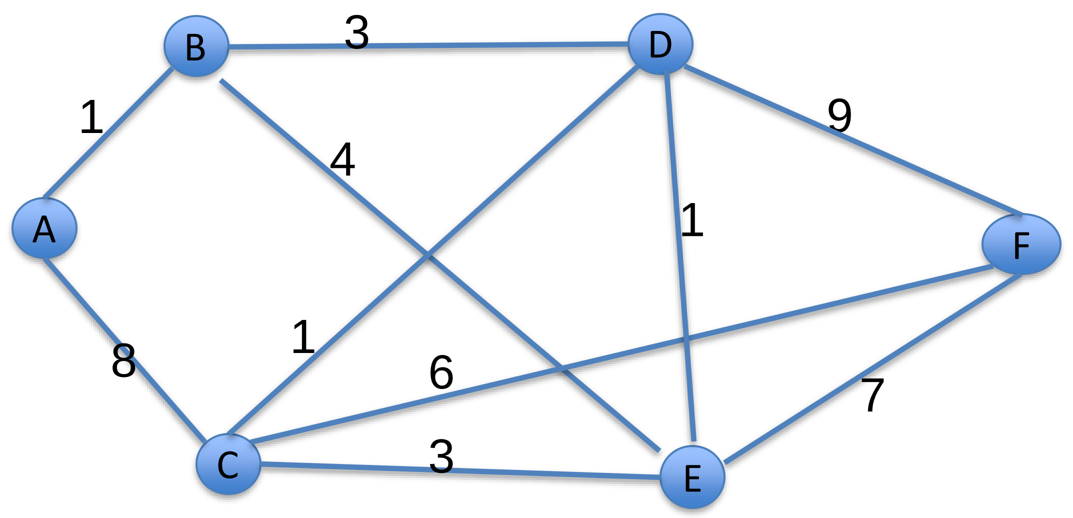
BfsExplore(v)

```
visit v #including start...end
mark v as visited
Q := empty queue
enqueue(Q,v)
while Q is not empty
   v := dequeue
   for each neighbor w of v
     if !visited(w)
       visit w
       mark w as visited
       enqueue(Q,w)
```



Each visit is represented by a box. In BFS, a visit ends before a next visit starts. In DFS, a visit includes other visits in itself.

Tracing Dijkstra’s Algorithm



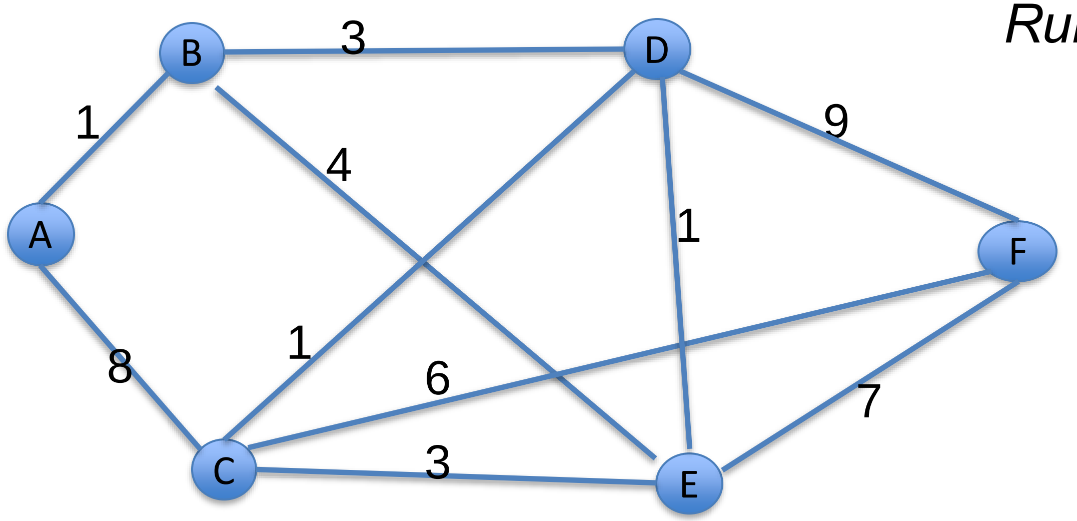
Still confused on how to trace the Dijkstra’s Algorithm?

Try to do with the above graph then compare with the solution in the next 3 slides. The task:

Given the above graph. Find a shortest path:

- From A to B
- From A to C
- From A to F
- From A to any other node

by tracing the Dijkstra’s Algorithm.



Run Dijkstra's Algorithm from A

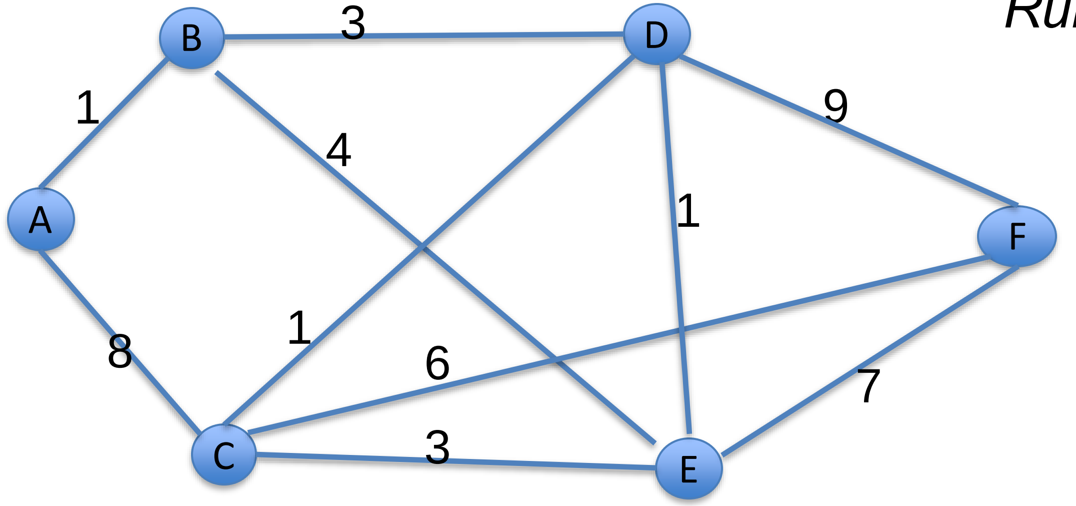
done	A	B	C	D	E	F
	0, nil	∞,nil	∞,nil	∞,nil	∞,nil	∞,nil
A						

this column: nodes with shortest path found

`dist[B]`: shortest-so-far distance from A

`pred[D]`: node that precedes D in the path A→D

Run Dijkstra's Algorithm from A

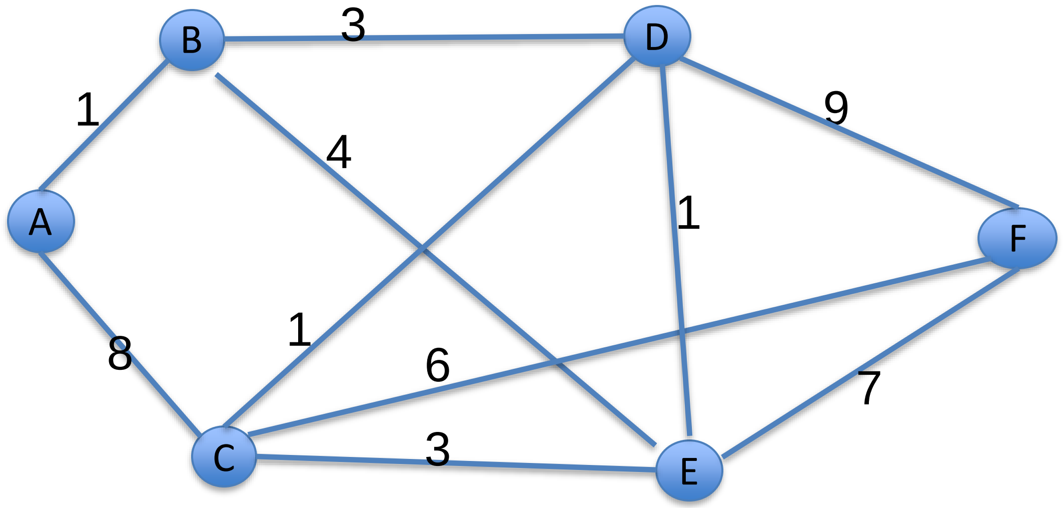


The dist at A is 0, there is an edge A->C with length 8, so we can reach C from A with distance 0+8, and 8 is better than previuosly-found distance of ∞

done	A	B	C	D	E	F
	0, nil	∞,nil	∞,nil	∞,nil	∞,nil	∞,nil
A		**1,A**	8,A	∞,nil	∞,nil	∞,nil
B			8,A	**4,B**	5,B	∞,nil
D			**5,D**		5,B	13,D
C					**5,B**	11,C
E						**11,C**
F						

Update this cell because now we can reach C from D with distance 4 (of D) + 1 (of edge D→C), and 5 is **better** than 8

At this pointy, we can reach E from D with distance 4 (of D) + 1 (of edge D→E), but new distance 5 is **not better** than the previously found 5, so no update!



Find a shortest path A→F:

- *the shortest path has weight 11 (last row of column F)*
- *the path is*

C→F

D→C

B→D

A→B

path= A→B→D→C→F

What's the found shortest path from A to F?
distance= 11, path=A→B→D→ C→F

done	A	B	C	D	E	F
	0, nil	∞,nil	∞,nil	∞,nil	∞,nil	∞,nil
A		**1,A**	8,A	-	-	-
B			8,A	**4,B**	5,B	-
D			**5,D**		5,B	13,D
C					**5,B**	11,C
E						**11,C**
C						

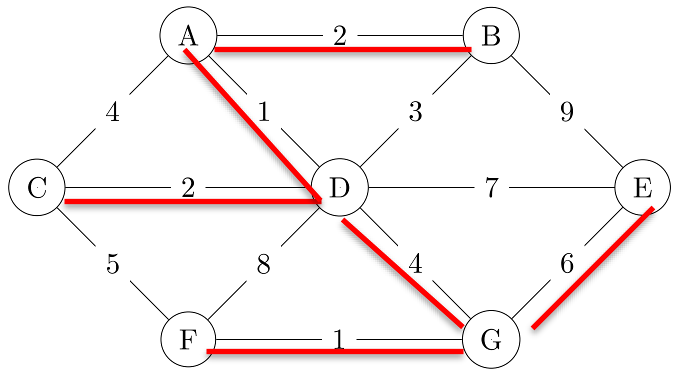
pred[B]= A:
A→B→D→ C→F

pred[D]= B:
B→D→ C→F

pred[C]= D:
D→ C→F

pred[F]= C, that is we came to F from C: C→F

the shortest distance from A to F is 11



Check: Soln to Q5.8 : Tracing Prim's Alg

What's the resulted MST: the red-linked

What's the cost of that MST?

cost= 0+2+2+1+6+1+4= 16

step	node ejected	A	B	C	D	E	F	G
0		**0/nil**	∞/nil	∞/nil	∞/nil	∞/nil	∞/nil	∞/nil
1	A		2,A	4,A	**1,A**	∞/nil	∞/nil	∞/nil
2	D		**2,A**	2,D		7,D	8,D	4,D
3	B			**2,D**		7,D	8,D	4,D
4	C					7,D	5,C	**4,D**
5	G					6,G	**1,G**	
6	F					**6,G**		
7	G							

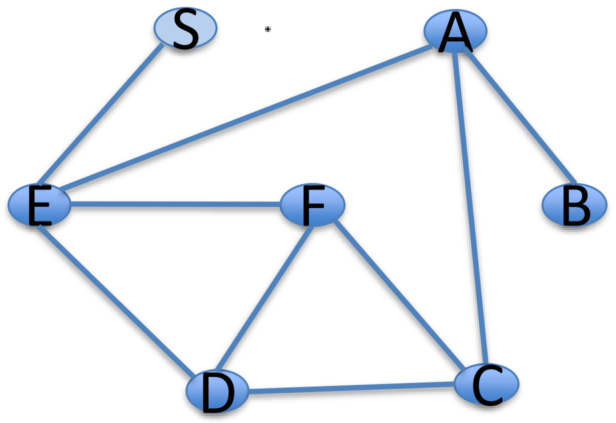
Q5.6 (Additional Homework) Finding Cycles with BFS

Explain how one can also use DFS to see whether an undirected graph is cyclic.

```
// adapt this to isCyclic
//   if you haven't done with DFS

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 (Additional Homework)

```
// adapt this to is2Colorable
//   if you haven't done with DFS

function is2Colorable(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)
        ?
```



b) 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.

Perhaps by editing BFS or DFS. BFS attached here, DFS in the next page.

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