COMP20007 Workshop Week 5

0	**Sparse vs Dense Graph**
1	**Topic 1: Graph Traversal,** Q 5.4, 5.6, 5.7 Traversal Trees: Q 5.5 Paths & Shortest Paths
2	Understanding Task 2 of A1 **Topic 2:** Dijkstra's (Greedy) Algorithm, Q5.9 ***Probably HomeWork:*** Prim's Algorithm Problems 5.8
LAB	**Assignment 1 Q&A (and MST if applicable)**

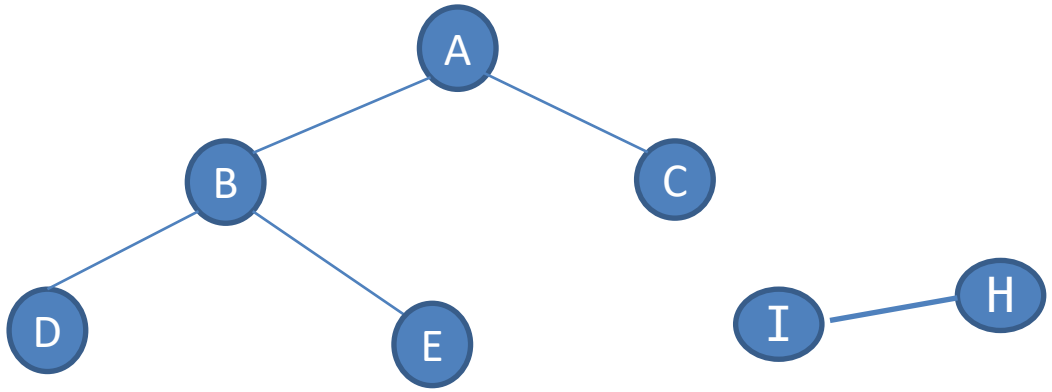
Current Issues

- MST worth 10%, know
  - When, Where

Assignment 1:

- Q&A today
- due Wed Week 6

DFS & BFS

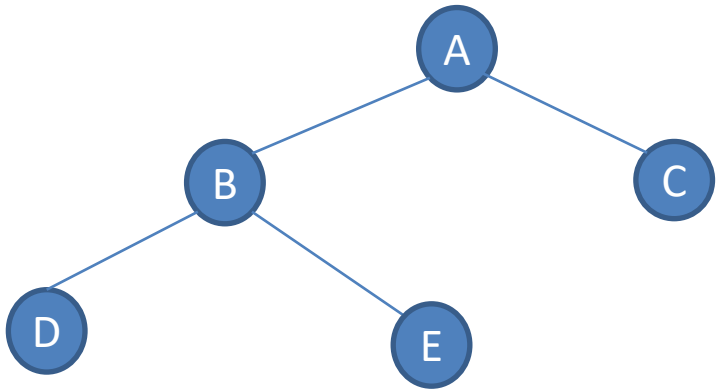


	DFS(G= (V,E))	BFS(G= (V,E))
Purpose	Traverses G in a systematic manner (from node to node using edges)	

Main Procedure	`for each v ∈ V visited[v] := false`
	`for each v ∈ V`
	`  if !visited(v) Explore(v)`
	
	`function Explore(v)`
	`  visit v and all connected-to-v nodes, and mark them as visited`
	`  # DfsExplore and BfsExplore are different`	

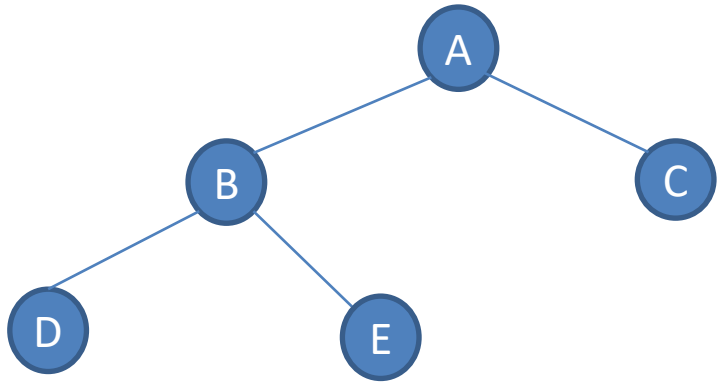
?	From which node we start? How to detect the number of connected components in G?	

DfsExplore




```
DfsExplore(A): should visit A,B,C,D,E
   # start visit A
   mark A as visited
   do pre-visit stuffs

      visit B

      visit C

   do post-visit stuffs
   # end visit A
```

DfsExplore



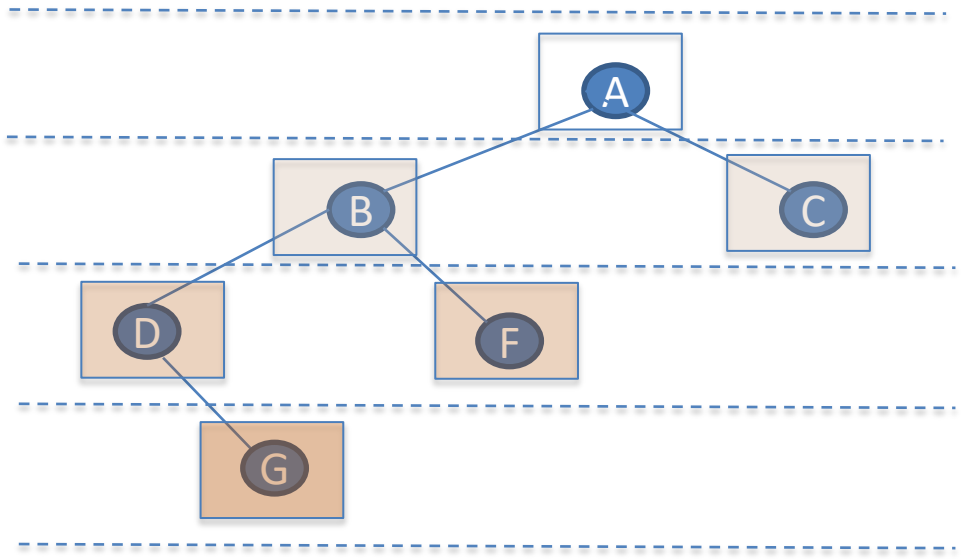
?: in DFS, how to:

- print out the nodes in the visited order (push-order)?
- print the nodes in the reverse of being visited order (pop-order)?

```
DfsExplore(A):
    # start visit A
    mark A as visited
    do pre-visit stuffs

    visit B
        visit D

        visit E

    visit C

    do post-visit stuffs
    # end visit A
```

```
DfsExplore(v):
    # start visit v
    mark u as visited
    do pre-visit stuffs

    for each neighbor w of v
      if w is unvisited
        DfsExplore(w)

    do post-visit stuffs
    # end visit v
```

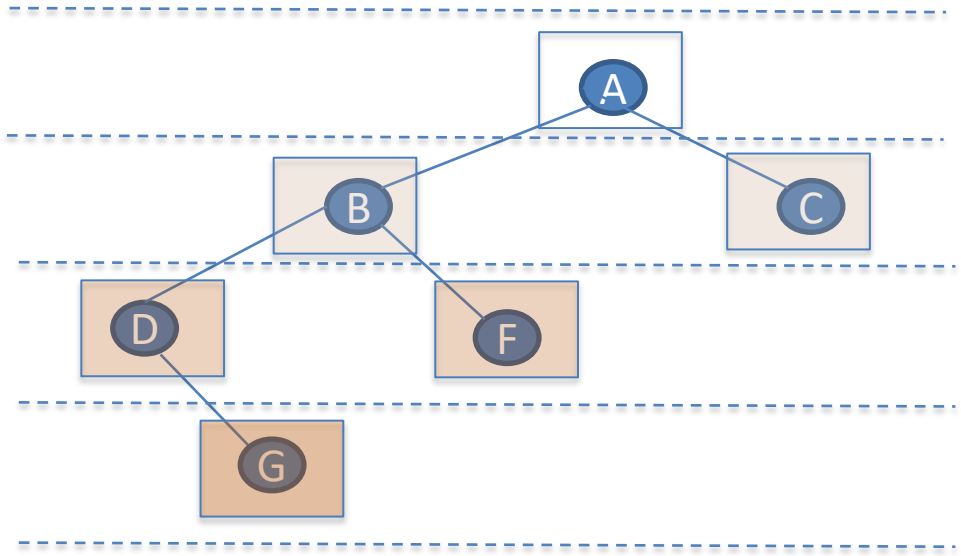
BfsExplore




```
visit A # do all “visit A” stuffs

visit neighbors of A, one by one

visit nodes in distance 2 from A

visit nodes in distance 3 from A
```

How?

How to keep tracks and visit B, C, D, F, ... in that level-order?



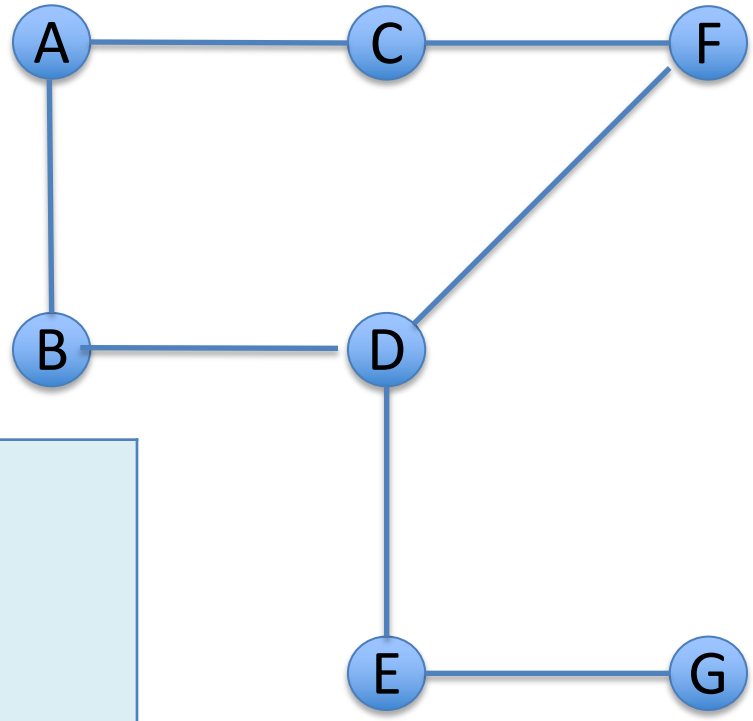

```
visit A # do all “visit A” stuffs

visit neighbors of A, one by one

visit nodes in distance 2 from A

visit nodes in distance 3 from A
```

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)
```

Q5.4: 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

Q5.6, 5.7: Do in groups of 2-3 mates, Discuss How

Q5.6: is cyclic?

a) Explain how one can use BFS and DFS 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 for proof).

```
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)
```

Q5.7: 2-Colourability

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. Start with an example.

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

```
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 ≠ ∅  do
        u := eject(Q)
        for each edge (u,w) do
        if w is marked with 0 then
          mark w with 1
          inject(Q, w)
```

Q5.6: Finding Cycles: detect cycles with BFS

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

Use function BFS to build

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

Can we use DFS?

b) 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 ≠ ∅  do
        u := eject(Q)
        for each edge (u,w) do
        if w is marked with 0 then
          mark w with 1
          inject(Q, w)
```

Q5.6: [homework] Finding Cycles with DFS

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

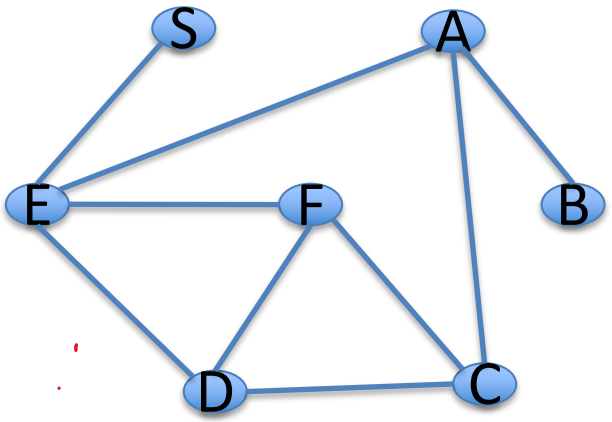
```
function DFS(G=(V,E))
  mark each node in V with 0
  for each v in V do
    if v is marked with 0 then
      DfsExplore(v)

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

5.7: 2-Colourability

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?



YOUR BRIEF ANSWER:

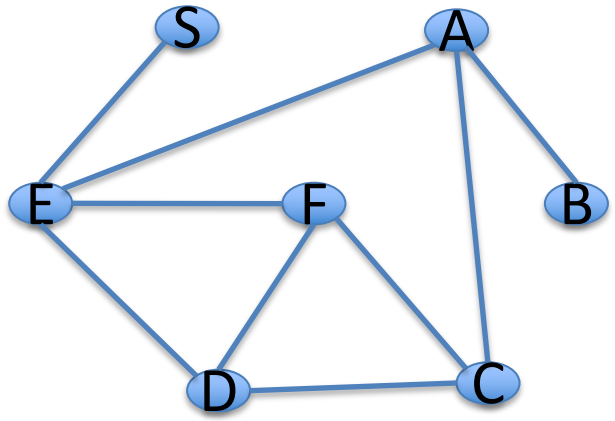
a) try to 2-colour the above graph, starting from **S**, using 2 colours 1=RED and 2=BLUE
hint: what is colour for S? how do we continue?

b) So, how to solve the 2-colourability? The pseudocode? What's the time complexity?
(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?

5.7: 2-Colourability: solving with DFS

```
// adapt 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)
```



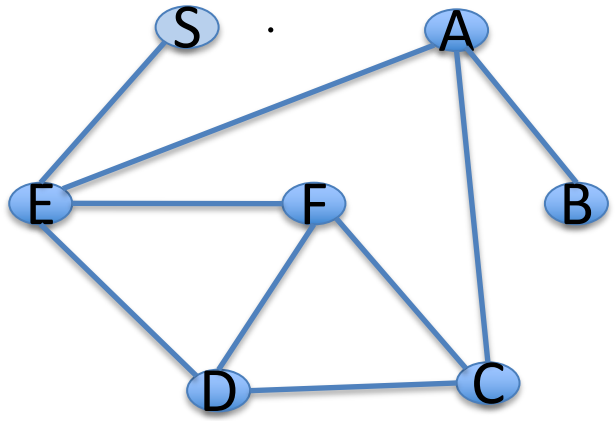
b) Design an algorithm to check whether an undirected graph is 2-colourable, that is, whether its nodes can be coloured with just 2 colours in such a way that no edge connects two nodes of the same colour.

Perhaps by editing BFS or DFS. DFS attached here.

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

5.7: 2-Colourability (Homework: do it by adapting BFS)

```
// adapt this to is2Colorable
//    iif 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 ≠ ∅  do
        u := eject(Q)
        for each edge (u,w) do
        if w is marked with 0 then
          mark w with 1
          inject(Q, w)
        ?
```

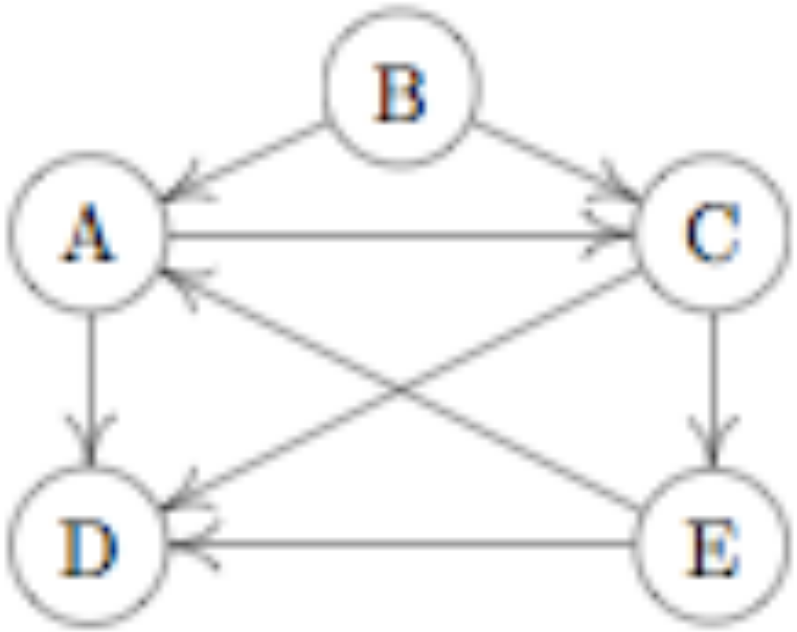


b) Design an algorithm to check whether an undirected graph is 2-colourable, that is, whether its nodes can be coloured with just 2 colours in such a way that no edge connects two nodes of the same colour.

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?

Q5.5: Tree, Back, Forward and Cross Edges (time permits)



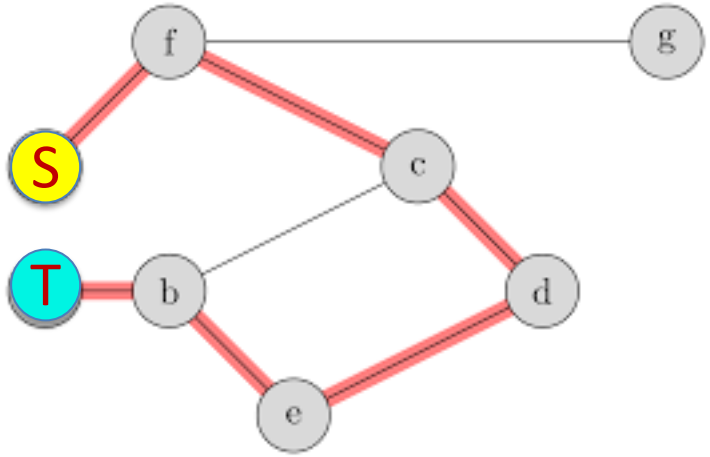
A DFS of a di-graph can be represented as a collection of trees. Each edge of the graph can then be classified as a *tree edge*, a *back edge*, a *forward edge*, or a *cross edge*. A tree edge is an edge to a previously un-visited node, a back edge is an edge from a node to an ancestor, a forward edge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

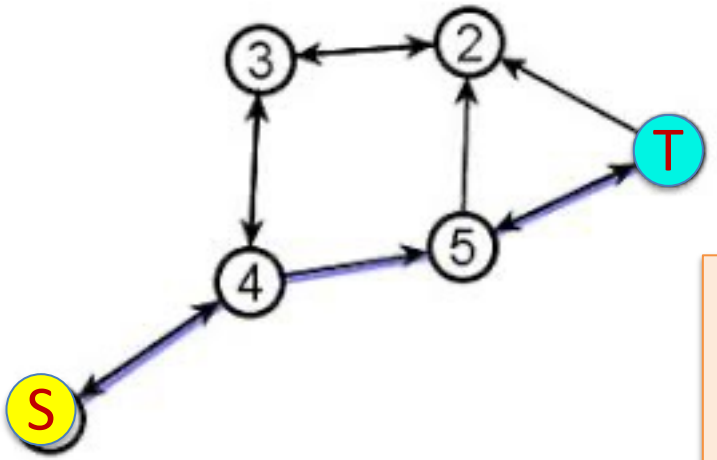
In an undirected graph, you wont find any forward edges or cross edges. Why is this true? You might like to consider the graph above, with each of its edges replaced by undirected edges.

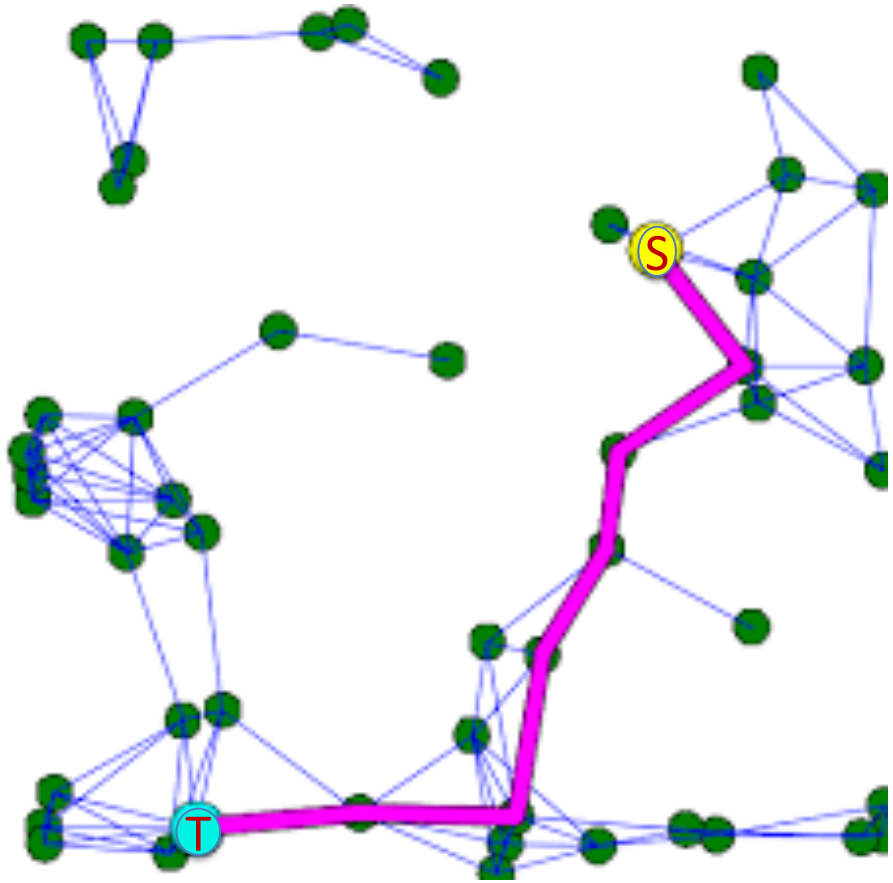
Paths in unweighted graphs: path length, shortest path

Path= S→f→c→d→e→b→T, length=6



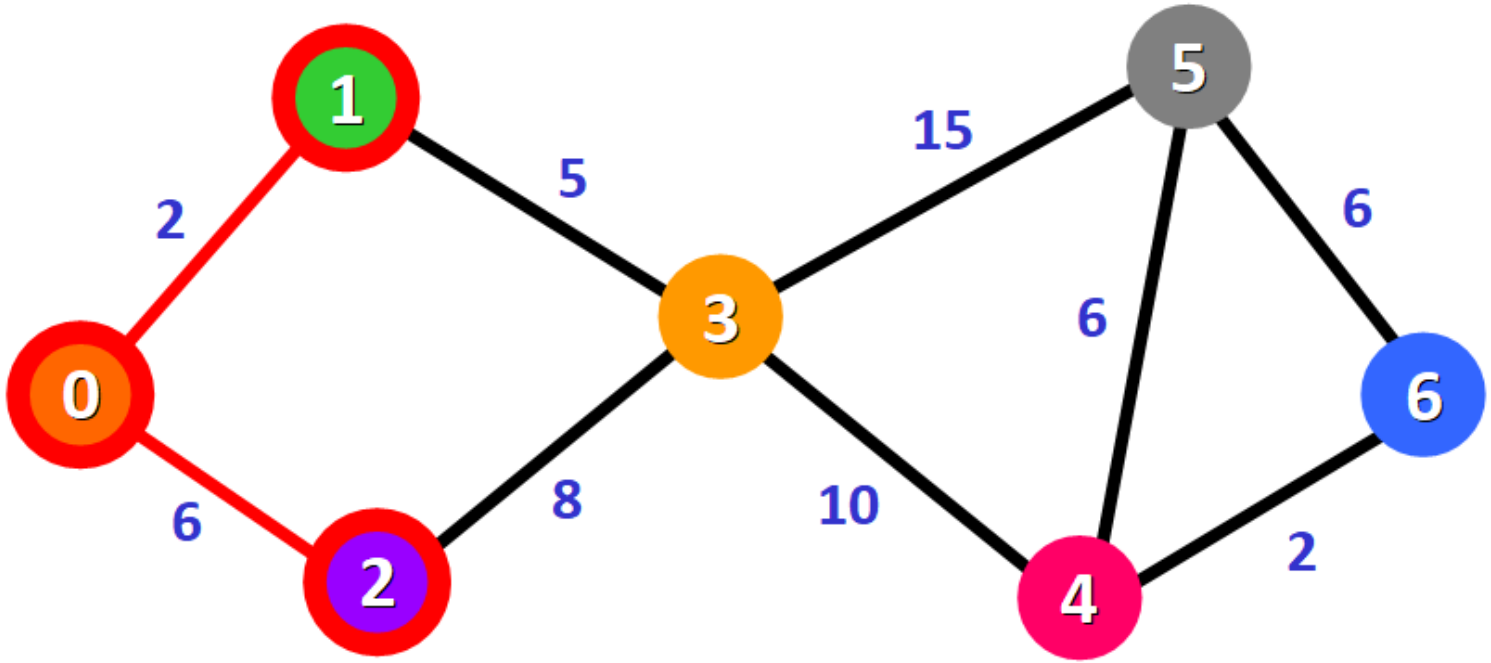
Path= S→4→5→T, length= 3





- For finding a path from S to T can we use DFS? BFS?
- For finding a shortest path from S to T can we use DFS? BFS?
- Applied to directed graphs? cyclic graphs?

How about 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

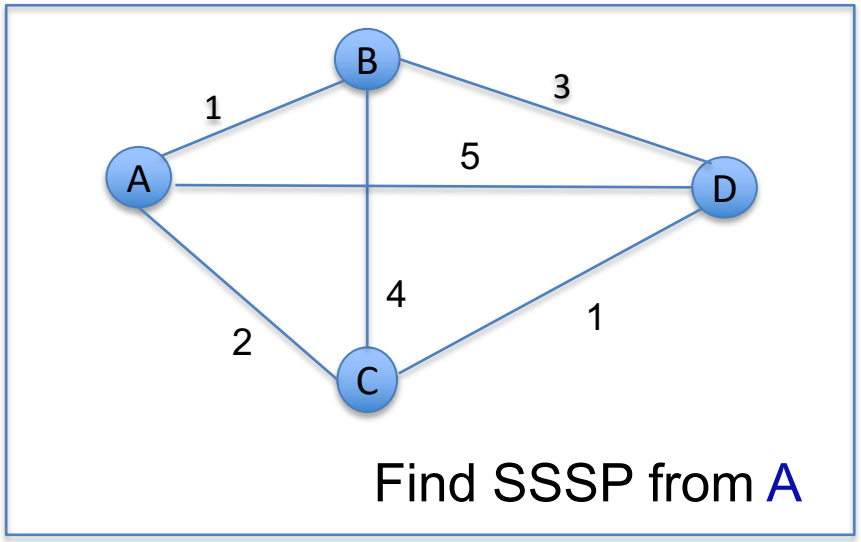
Understanding Task 2 of A1

Dijkstra's Algorithm: Single Source Shortest Path (SSSP)

The task:

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

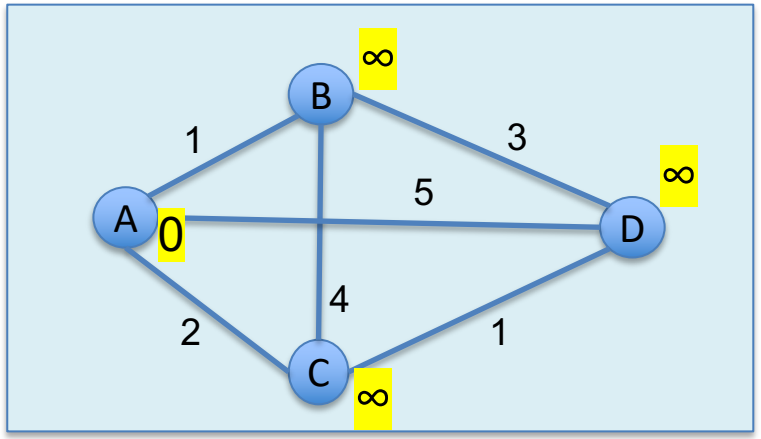
Input



Output

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

How to do SSSP?



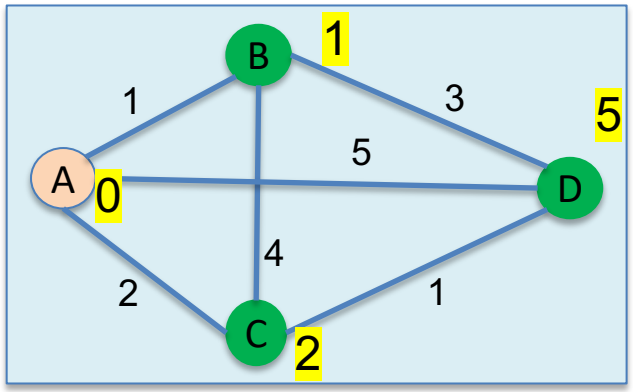
Find a shortest path:

- From A to any other node

Start from the source A:

- shortest path A→A has `dist[A]= 0`
- and distance-so-far `dist[]= {0, ∞, ∞, ∞}` (for {A,B,C,D})

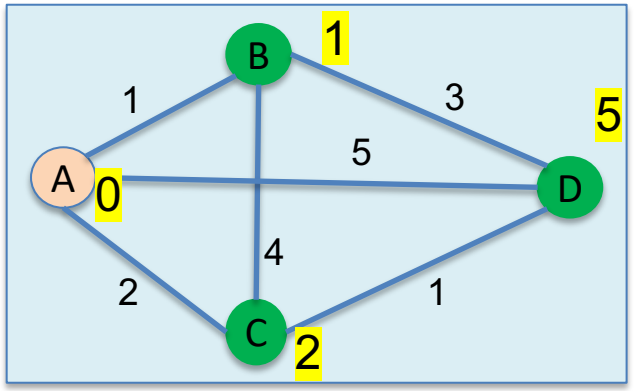
→ Clearly, we've done with A, and can remove it from the job list!
But with that, we should use the information from A to update
`dist[]= {0, 1, 2, 5}`



What's next?

But with that, we should use the information from A to update

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



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

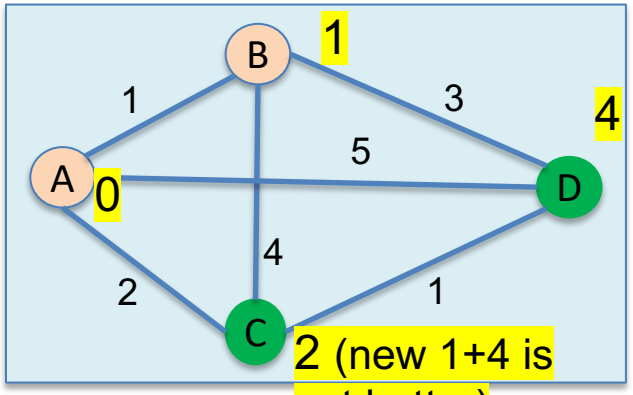
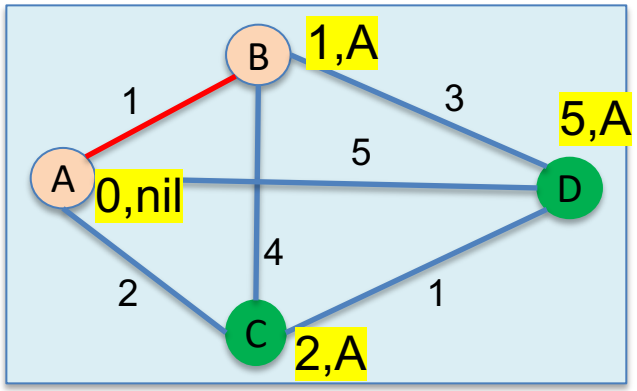


image: https://medium.com/analytics-vidhya/what-is-the-greedy-algorithm-5ed71f9a7b3a

So shortest path from A to B found with `dist[B]= 1`.

But how can we retrieve the detailed path A→B?
need to keep track of ???

HowTo: Keep track of our steps



this column: nodes with found shortest path

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

set {B,C,D} includes not-yet-done elements at this stage.

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

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

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        // practice: just mark u as visited
    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. change the weight of v in PQ to the new dist[v].

Complexity:

Will see:

- makePQ of n elems: $O(n)$
- dequeue = deleMin(PQ): $O(\log n)$
- enqueue = enPQ(PQ,x,w): $O(\log n)$
- change a weight in PQ: $O(\log n)$

So:

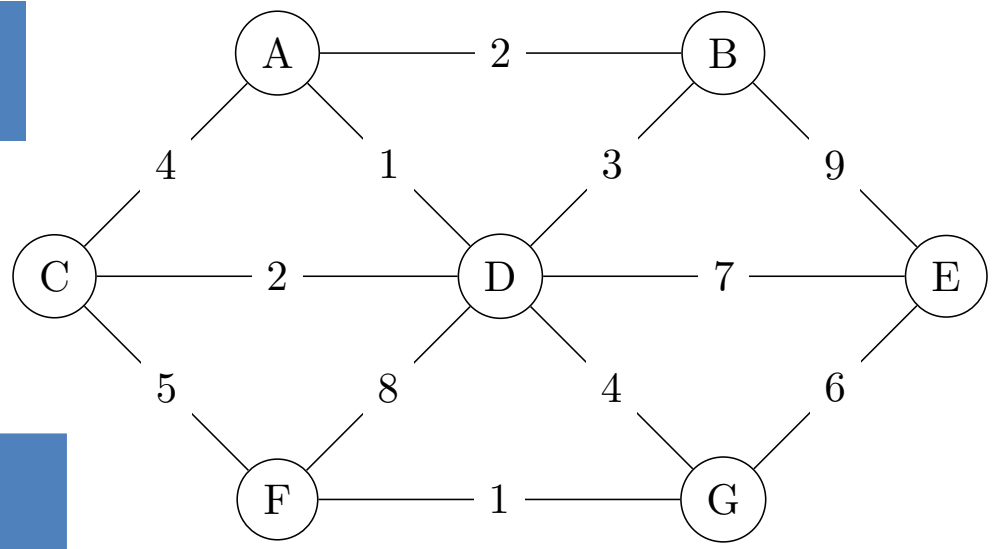
Complexity of DA is: $O(\ ?\ )$

Dijkstra's algorithm: a variation for finding shortest path from s to t

```
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=t break    #path s→t 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])
```

Q5.9: SSSP with Dijkstra's Algorithm: do in groups of 2-3

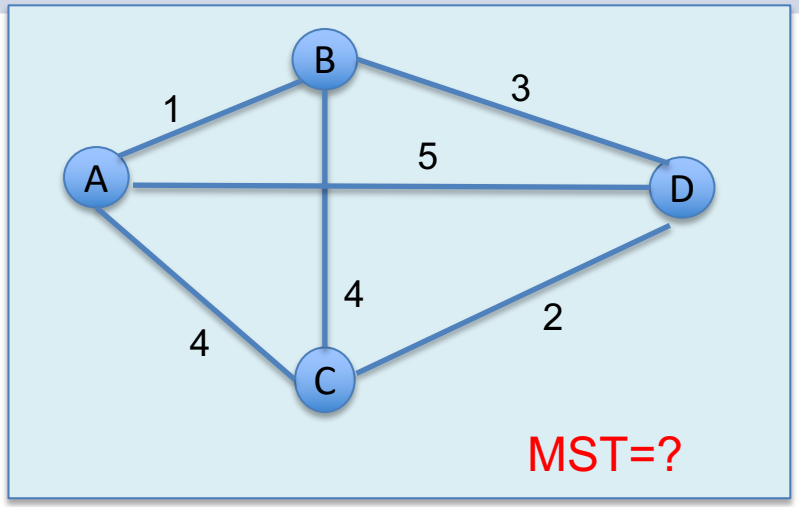
Trace Dijkstra's algorithm on the following graph, with node E as the source.
How long, and what, is the shortest path from E to A?

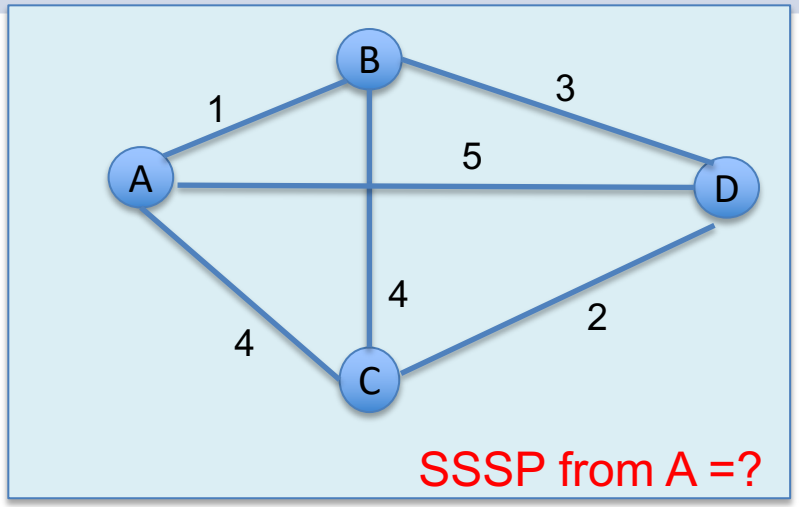


step››	node done	A	B	C	D	E	F	G
0								
1								
2								
3								
4								
5								
6								
7								

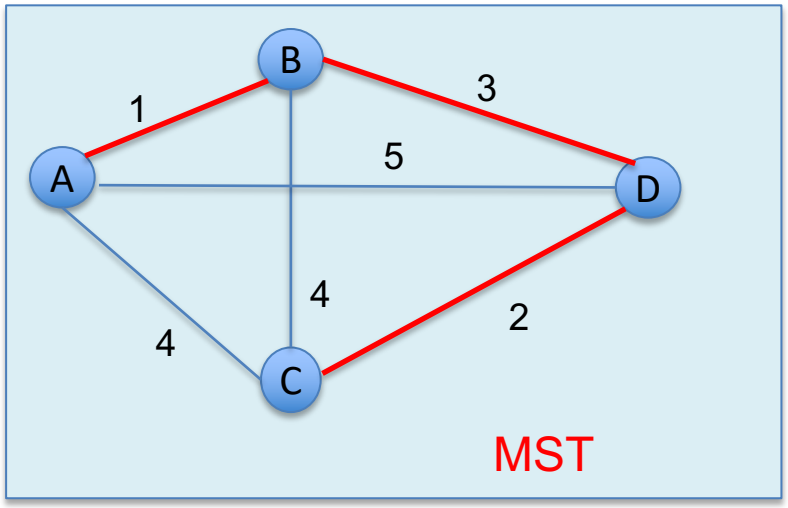
Prim's Algorithm vs Dijkstra's Algorithm (time permits)

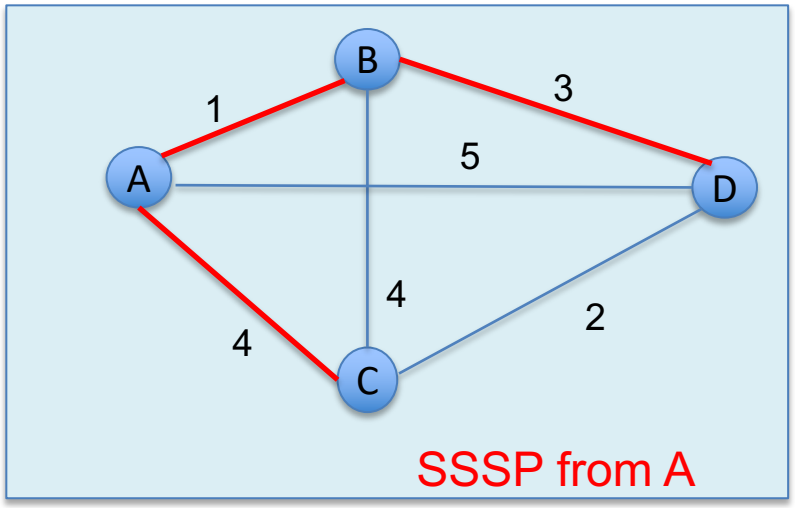
	Prim's	Dijkstra's
Aim	find a MST	find SSSP from a vertex s
Applied to which graphs?	?	
Works on directed graphs?	?	
Works on unweighted graph?	?	
Related concepts		
spanning trees = ? MST = ? is MST unique?		paths A→D = ? shortest paths A→D = ? is shortest path from s→v unique?
Approach:		

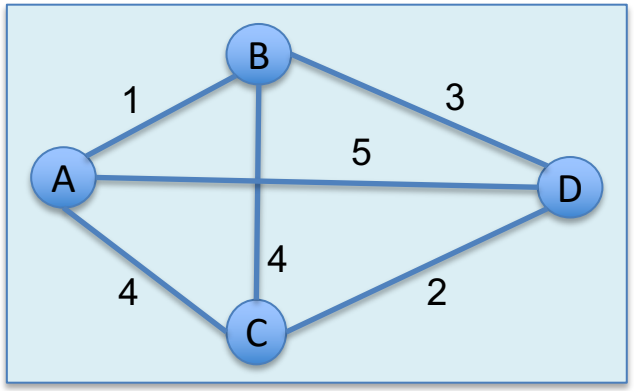




	Prim's	Dijkstra's
Aim	find a MST	find SSSP from a vertex s
Applied to	connected weighted graphs with weights ≥ 0	weighted graphs with weights ≥ 0
Works on directed graphs?	no, a general directed graph is unconnected	yes
Works on unweighted graph?	Y, we just suppose that each edge has weight 1 (or the same weight)	
Related concepts		
spanning trees = any tree that contains all vertices MST = the red-linked tree is MST unique? N		paths A→D: A→B→D, A→D, A→C→D shortest paths A→D = A→B→D is shortest path from s→v unique? N
Same approach: Greedy, at each step pick an edge with the best outcome		







Tracing Prim's Algorithm to find a MST: similar to DA

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					

The rest: at home, do the unfinihsed exercises (if any)

Assignment A2 and MST: Q&A

Task 1: Q?

Task 2: Q?

MST: Q?

assignment 1

- Do it early! Submit early, submit as many times as you want!
- Read & participate in discussion forum!
- Make sure that you follow well the specification.
- Make sure that the theoretical part is presented clearly and concisely

- 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 use valgrind in Ed

LAB

- work on not-yet-done this week's workshop problems: Questions 9, 8, 6, 7 (if not finished) and/or

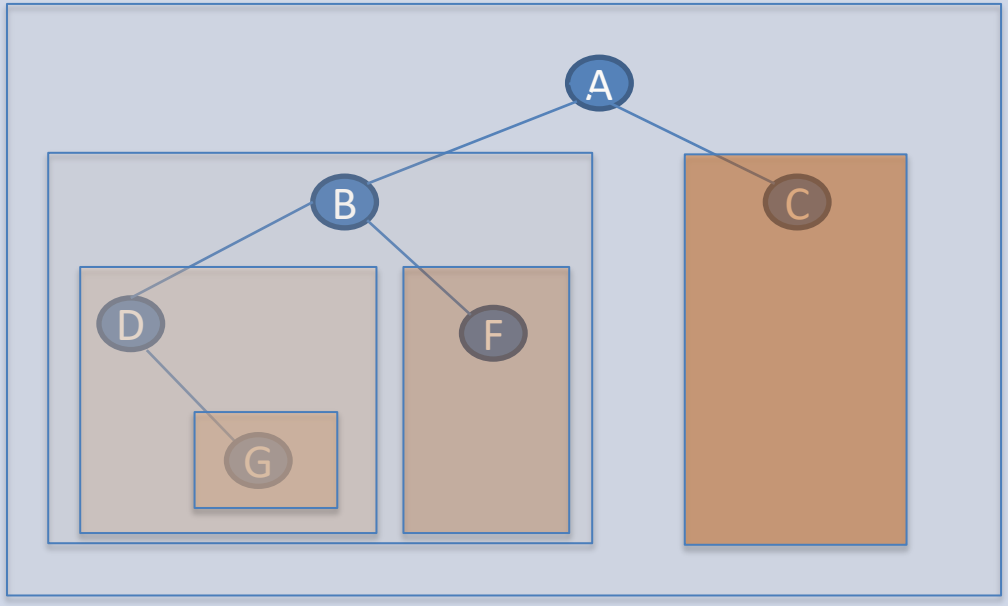
-work on assignment 1

-or, if applicable, prepare for MST: **see sample MST test and solution in Ed**

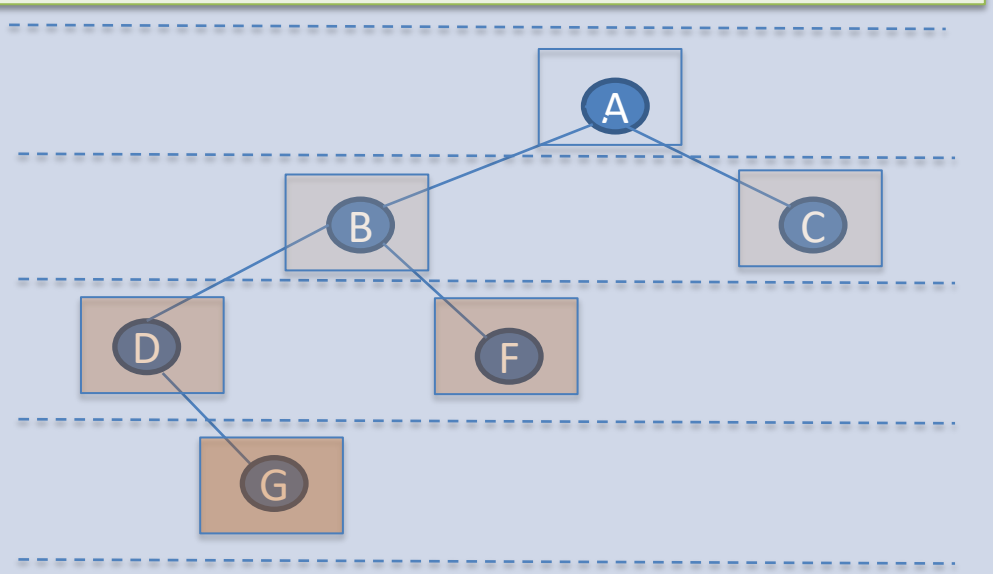
Additional Slides

DFS vs BFS	DfsExplore(v)	BfsExplore(v)

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


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

#end visit v
```

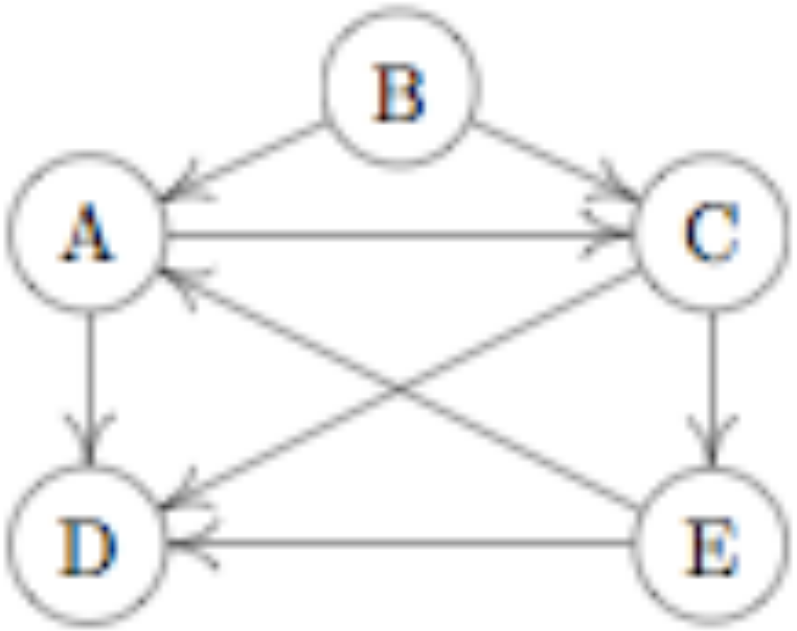



```
visit v #including start...end
mark v as visited
Q := empty queue
enqueue(Q,v)
while Q is not empty
   v := dequeue
   for each neighbor w of v
     if !visited(w)
       visit w
       mark w as visited
       enqueue(Q,w)
```



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.

Q5.5: Tree, Back, Forward and Cross Edges
Do if not yet done!



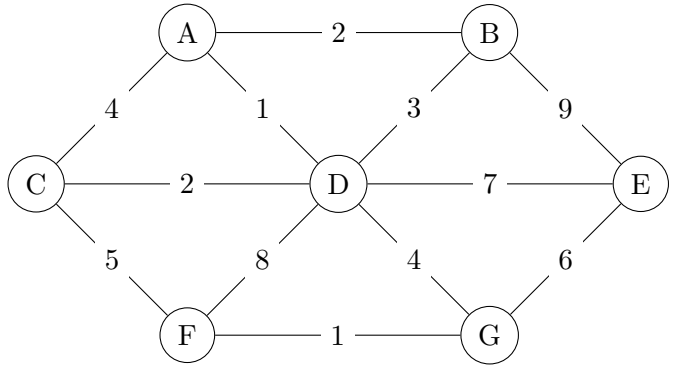
A DFS of a di-graph can be represented as a collection of trees. Each edge of the graph can then be classified as a *tree edge*, a *back edge*, a *forward edge*, or a *cross edge*. A tree edge is an edge to a previously un-visited node, a back edge is an edge from a node to an ancestor, a forward edge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

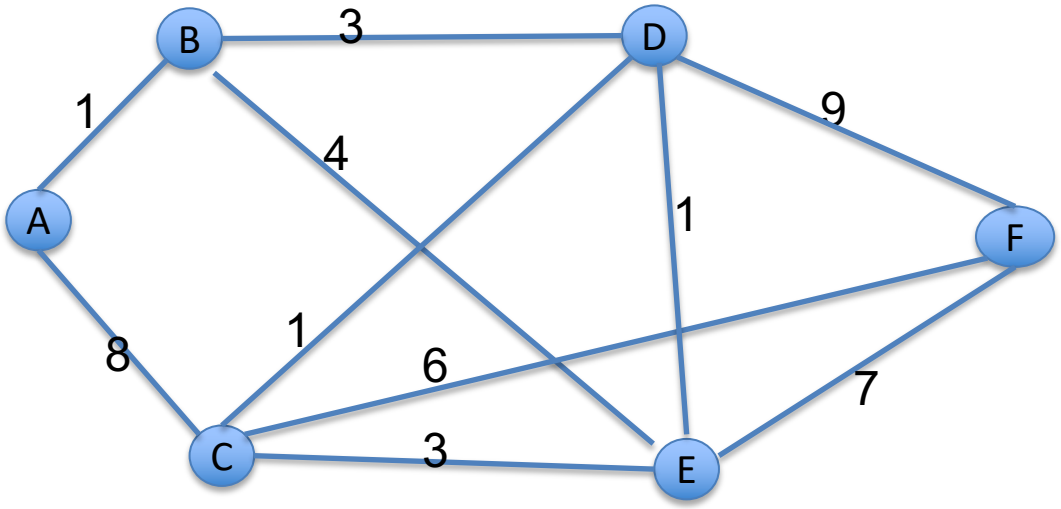
Q 5.9 Part b) : DA from A

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



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								

Tracing Dijkstra's Algorithm



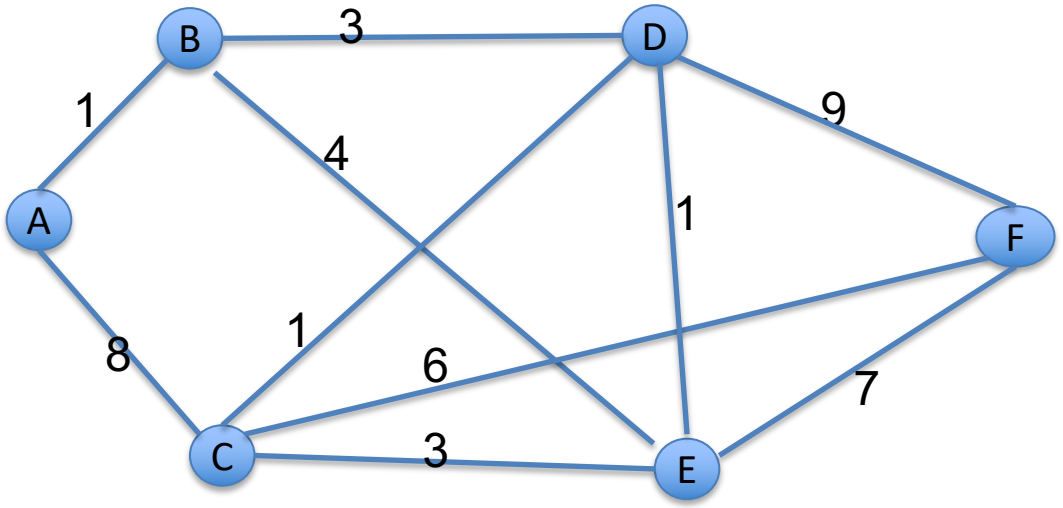
Still confused on how to manually run the Dijkstra's Algorithm?

See next slides for 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*
- *Then compare your soln with the next 2 pages*

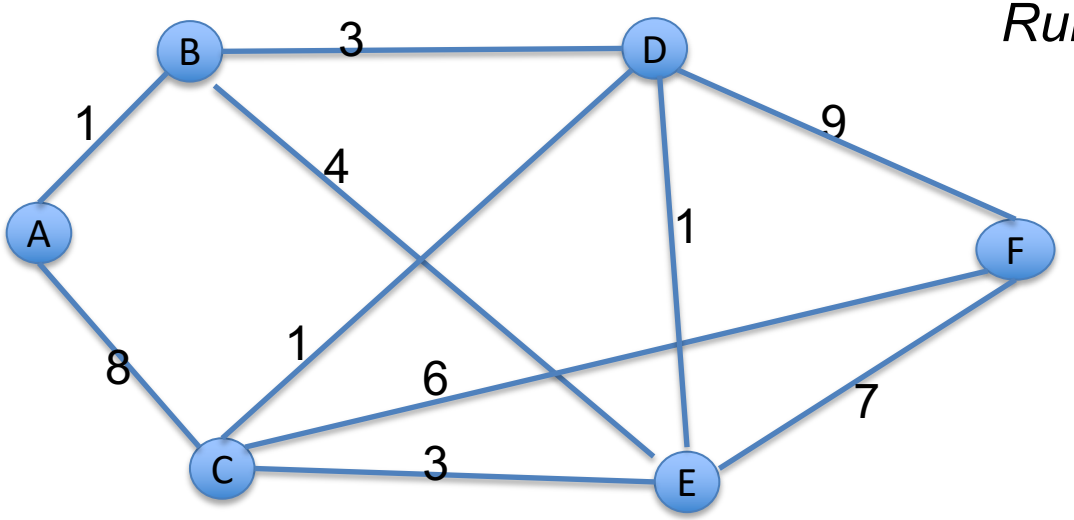
done	A	B	C	D	E	F
	0, nil	∞,nil	∞,nil	∞,nil	∞,nil	∞,nil
A						

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

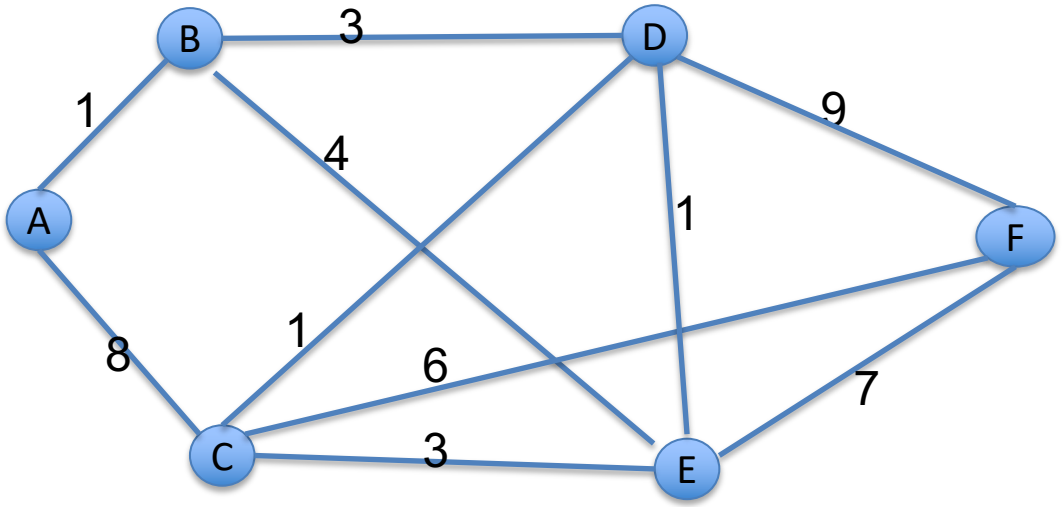


The dist at A is 0, there is an edge A->C with length 8, so we can reach C from A with distance 0+8, and 8 is better than previuosly-found distance of ∞

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

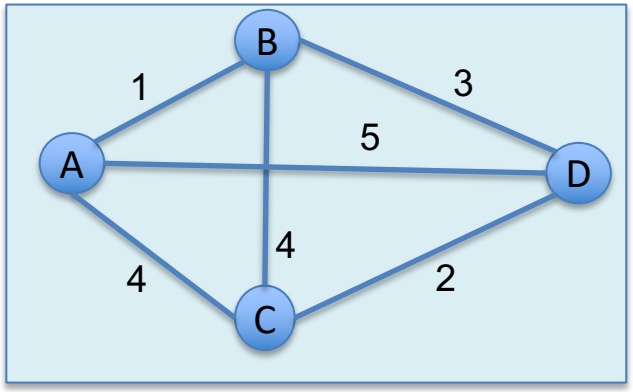
pred[D]= B:
B→D→ C→F

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

Tracing Prim's Algorithm

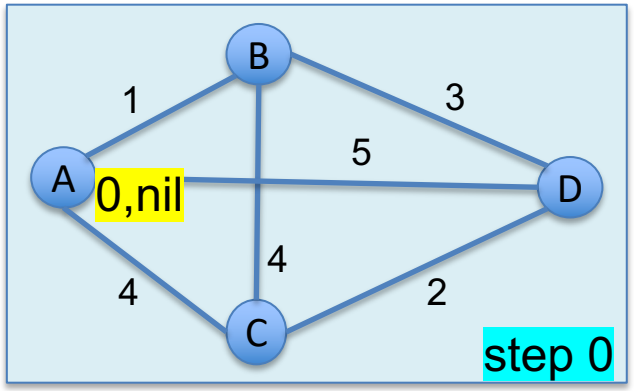


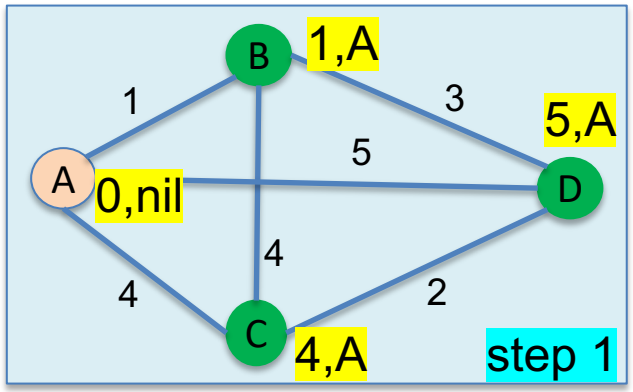
At each step, we add a node to MST.

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					





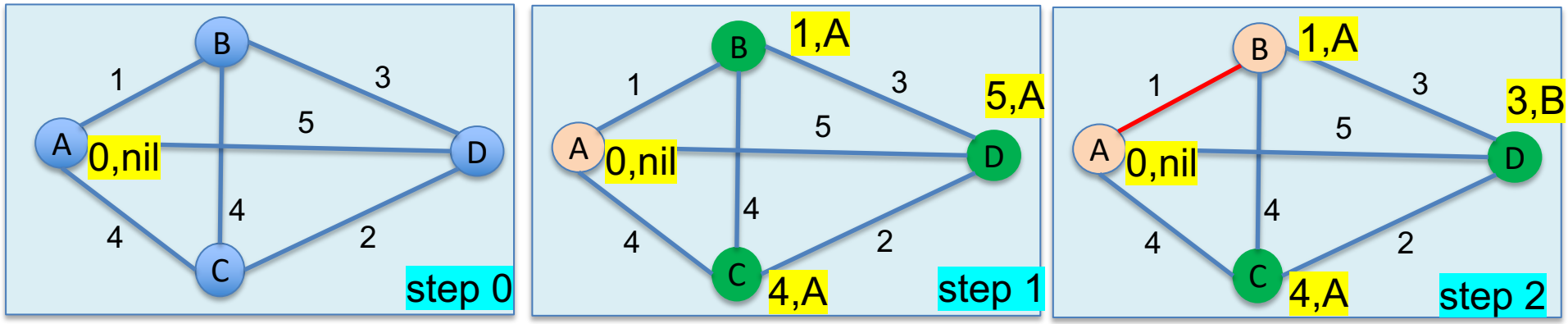
Tracing Prim's Algorithm with graphical illustration

Note:
The graphic illustration is just for fun. Ignore them if you think they are disturbing

Running Prim's Algorithm to find a MST

Step 1:
- add A to MST with cost 0
- use A to update cost for its neighbours

step	node added to MST	A	B	C	D
0		0,nil	∞,nil	∞,nil	∞,nil
1	A		1,A	4,A	5,A



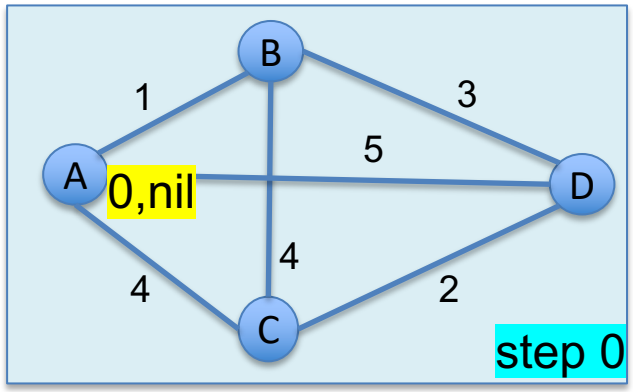
Running Prim's Algorithm to find a MST

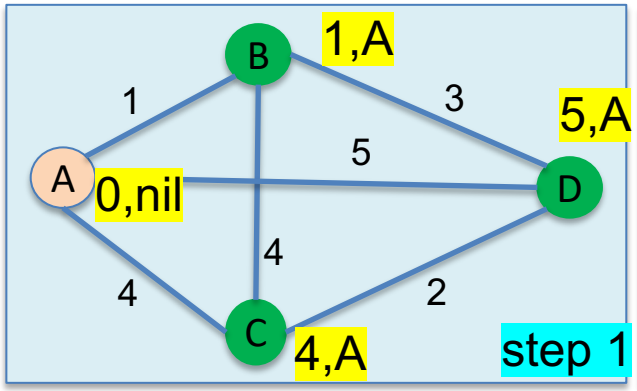
Tracing Prim's Algorithm with graphical illustration

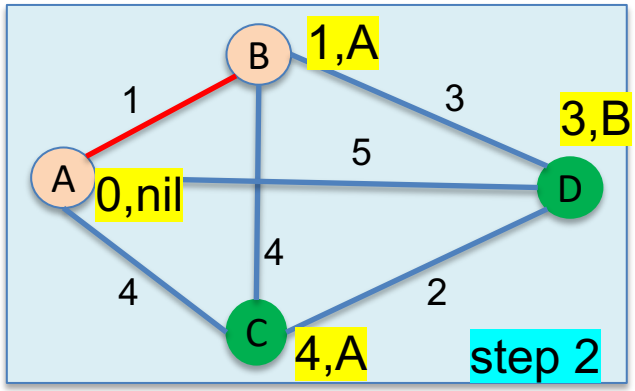
Step 2:

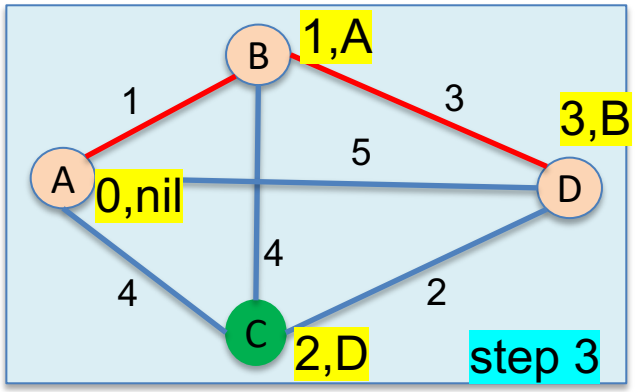
- B is chosen because it has the min cost amongst the green nodes (the green nodes and blue nodes are in the queue).

step	node added to MST	A	B	C	D
0		0,nil	∞,nil	∞,nil	∞,nil
1	A		1,A	4,A	5,A
2	B			4,A	3,B



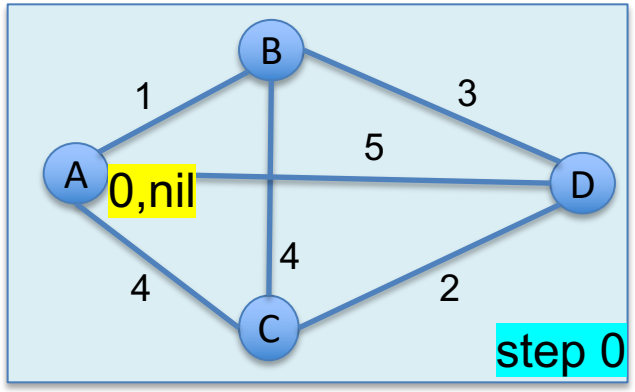


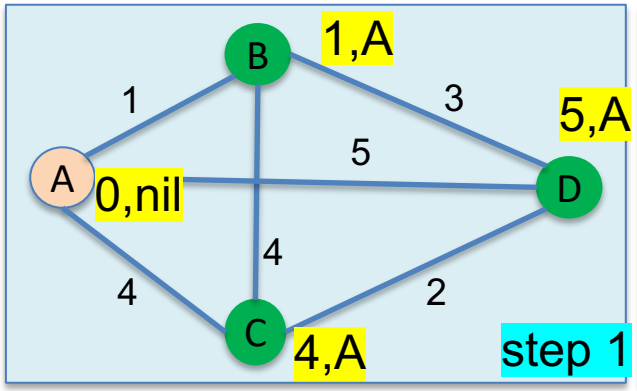


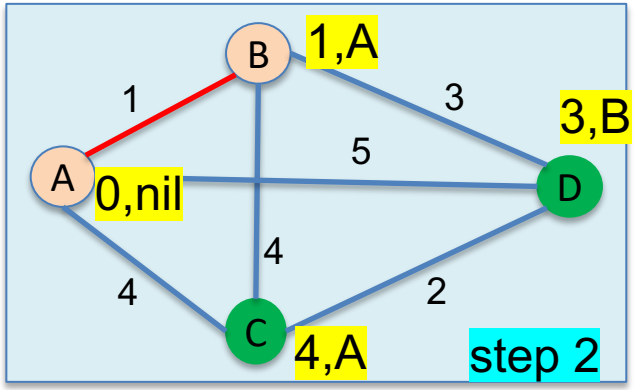


Tracing Prim's Algorithm with graphical illustration

step	node added to MST	A	B	C	D
0		0,nil	∞,nil	∞,nil	∞,nil
1	A		1,A	4,A	5,A
2	B			4,A	3,B
3	D			2,D	

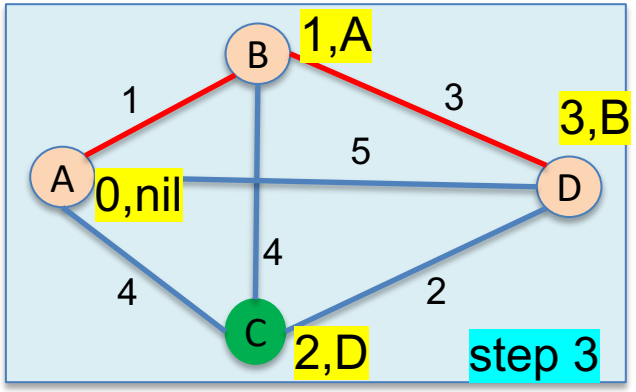


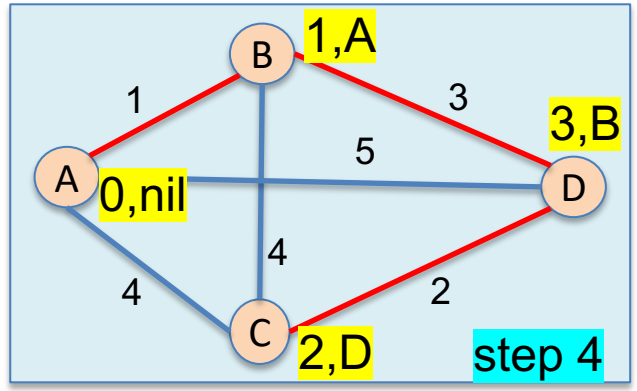




The final MST has cost= 0(A) + 1(B) + 2(C) +3(D)= 6

The MST is the red-linked graph, which can be rebuilt using the prev of the pair cost,prev in each node → *The algorithm can be run just using the table.*





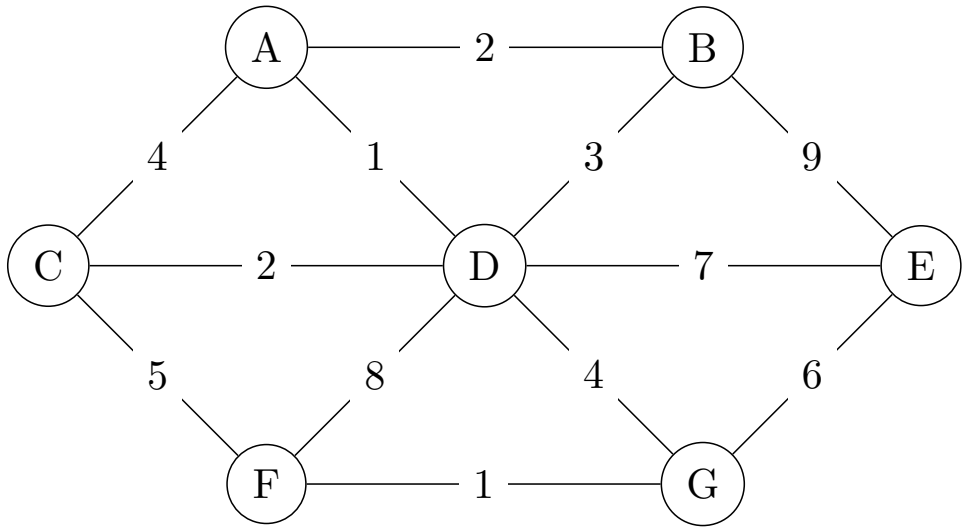
Notes:
The graphic illustration is just for fun. Ignore them if you think they are disturbing

step	node added to MST	A	B	C	D
0		0,nil	∞,nil	∞,nil	∞,nil
1	A		1,A	4,A	5,A
2	B			4,A	3,B
3	D			2,D	
4	C				

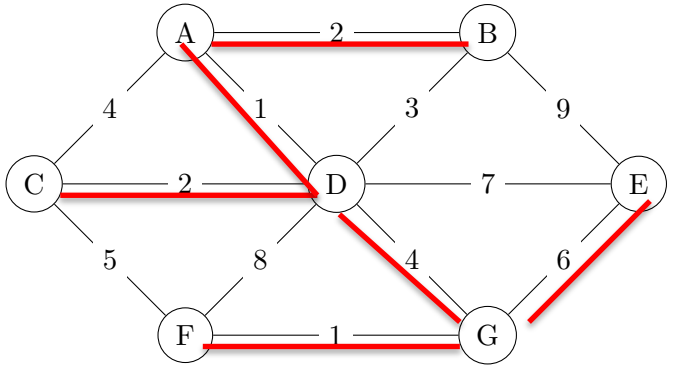
Problem 5.8: MST with Prim's Algorithm – do it, then check with the followed soln

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								



Check: Run 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							