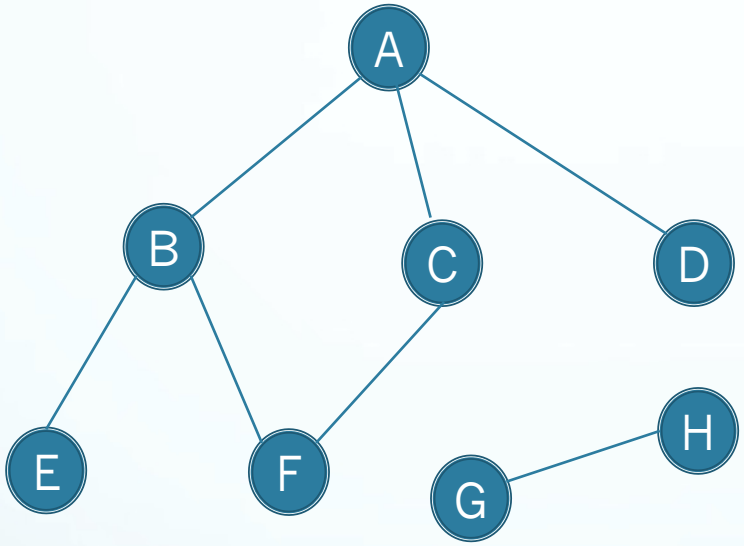
COMP20007 Workshop Week 5

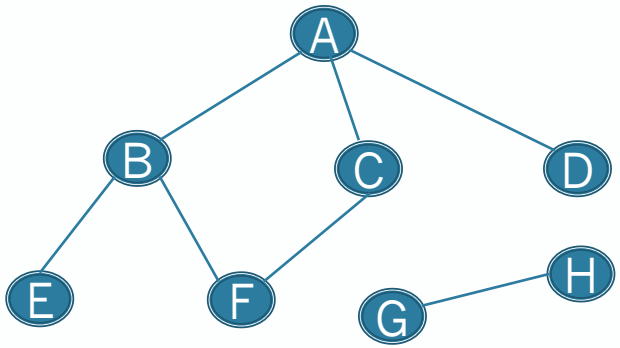
0	**Topic 1:** DFS and BFS: Q 5.4, 5.5, 5.6, 5.7 group work: 5.6, 5.7 homework: 5.5
1	**Topic 2:** Dijkstra's and Prim's Algorithms; Q5.8, Q5.9 ***Group Work:*** Problems 5.8, 5.9
2	**Assignment 1 Q&A (and MST if applicable)**, and -make sure you understand how to start the programming part -
LAB	**review for MST if applicable** **remaining time: do ass1 & ask questions**

DFS/ BFS= ?

- List the nodes in order visited by BFS, then by DFS



DFS and BFS: the algorithms




```
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) in E do
    if w is marked with 0 then
      DfsExplore(w)
```

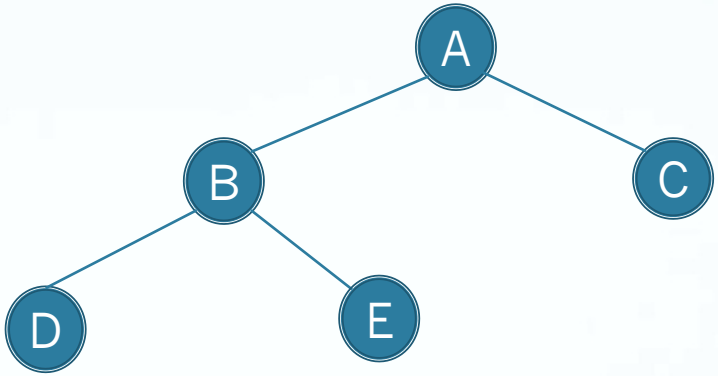
STACK MECHANISM VIA RECURSION

```
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 (u,w) in E do
        if w is marked 0 then
          mark w with 1
          inject(Q, w)
```

BfsExplore(v)

EXPLICIT QUEUE MECHANISM

DFS/ BFS= ?



BFS

- start by visiting A

```
do "visit A" stuffs
```

- add A's neighbours in queue Q of "jobs to do"
- what's the content of the queue now?

DFS:

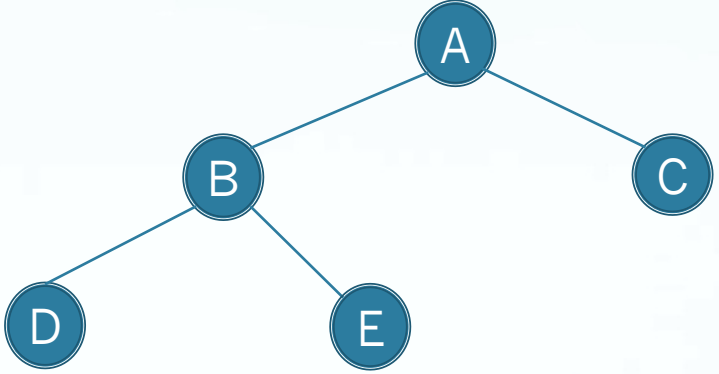
- start "visit A"

```
visit B
```

```
visit C
```

- when doing "visit A" stuffs
- when ending "visit A"

DFS/ BFS= ?

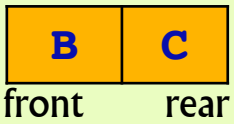


BFS

- start by visiting A

```
do all "visiting A" stuffs
```

- add A's neighbours in queue Q of "jobs to do"
- the queue now is



- remove from front of the queue, get B

```
do all "visiting B" stuffs
```

- add new B's neighbours (D and E) to the queue

- now visit C, D, E in that order, no new nodes added to the queue.

Order of "start visit": A B C D E
Order of "end visit": A B C D E

DFS:

- start "visiting A"

```
visit B
        visit D

        visit E

visit C
```

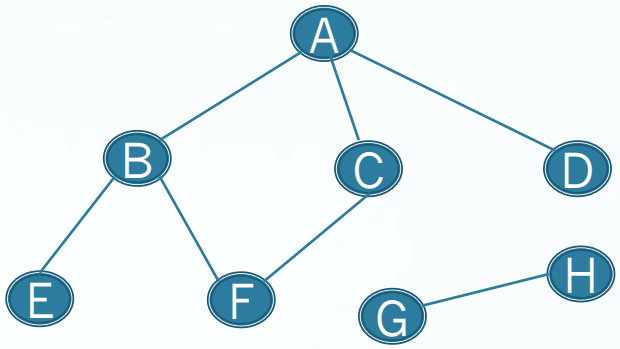
- do "visiting A" stuffs
- end "visiting A"

Order of "start visit": A B D E C (push-order)
Order of "end visit": D E B C A (pop-order)

DFS and BFS: the algorithms

How to:

- compute the number of connected components?
- print out the nodes in the visited order?
- print the nodes in the reverse of being visited order?



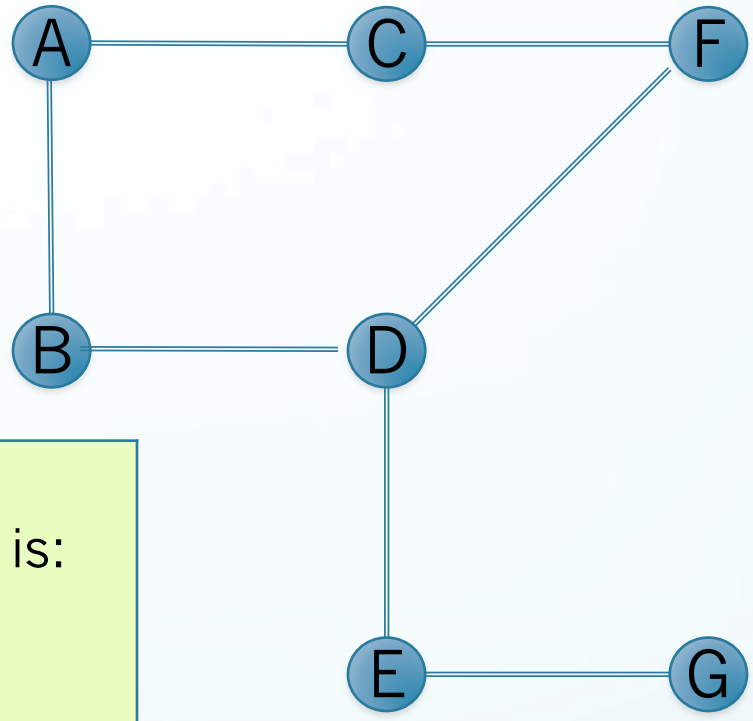

```
function DFS(G=(V,E))
  mark each node in V with 0
  for each v in V do
    if v is marked with 0 then
      DfsExplore(v)

function DfsExplore(v)
  // start visit v, Push is here
  // ... do pre-visiting stuffs
  mark v with 1
  for each edge (v,w) in E do
    if w is marked with 0 then
      DfsExplore(w)
  // ... do post-visiting stuffs
  // end visit v, Pop is here
```

```
function BFS(G=(V,E))
  mark each node in V with 0
  for each v in V do
    if v is marked with 0 then
      //BfsExplore(v)
      Q := empty queue
      mark v with 1
      inject(Q, v)        //=enQueue
      while Q ≠ Ø  do
        u := eject(Q)   //=deQueue
        // start visit u
        // ... do visiting stuffs with u
        // end visit u
        for each (u,w) in E do
          if w is marked 0 then
            mark w with 1
            inject(Q, w)//=enQueue
```

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: is cyclic?

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 for proof).

Note: skip part b) if it takes you more than 2 minutes, you can do it later!

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.

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?

Q5.6: Finding Cycles

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

YOUR ANSWER:

a) How?

Your pseudocode: make change to the BFS. **OR:**

Can we use DFS? You can try it using DFS in next page

function ISCYLIC(G=(V,E)) : just turn BFS to ISCYCLIC

b) Which one is better: DFS or BFS?

BFS algorithm – based on 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: Finding Cycles with DFS

DFS algorithm – based on lecture

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

YOUR ANSWER:

a) How?

Your pseudocode: make change to the DFS if you didn't do with BFS.

```
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)
```

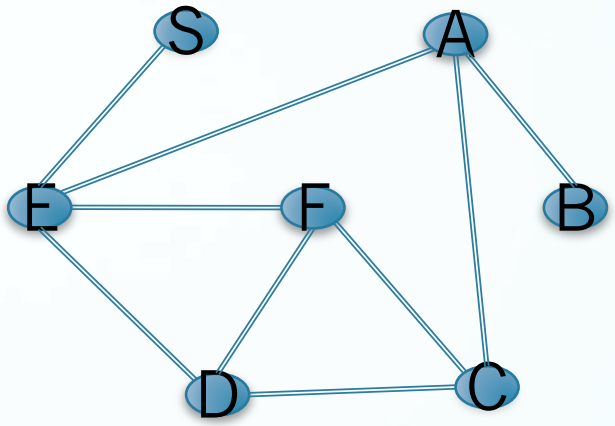
b) Which one is better: DFS or BFS?

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:

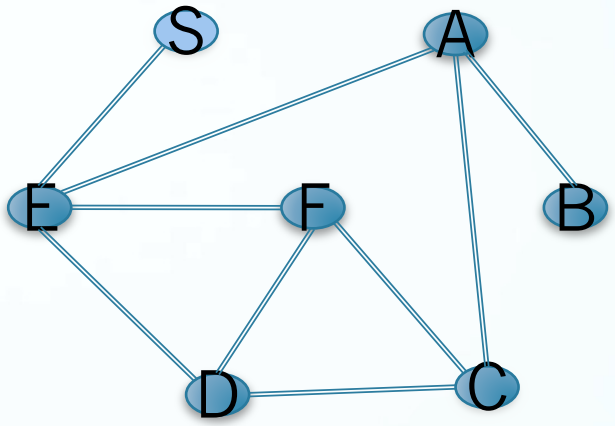
a) try to 2-colour the above graph, starting from **S**, using 2 colours 1=RED and 2=BLUE
hint: what is colour for S? how do we continue?

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

```
// adapt this to IS2COLORABLE
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)
```



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.

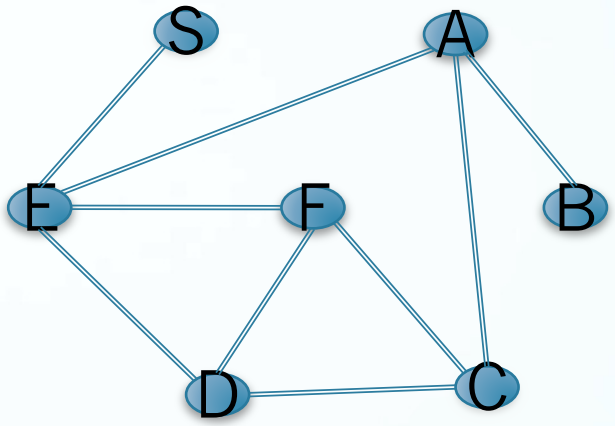
Perhaps by editing BFS or DFS. BFS attached here, DFS in the next page.

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

5.7: 2-Colourability

```
// adapt this to IS2COLORABLE
//    iif you haven't done with BFS
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)
```



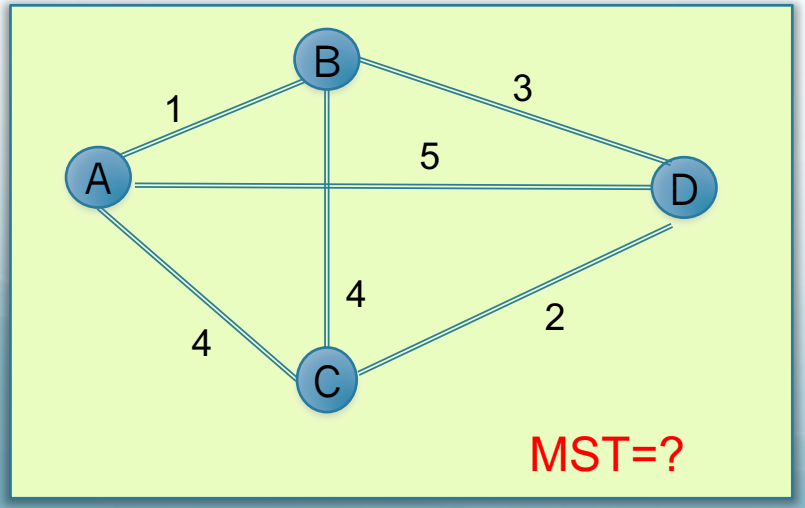
b) Design an algorithm to check whether an undirected graph is 2-colourable, that is, whether its nodes can be coloured with just 2 colours in such a way that no edge connects two nodes of the same colour.

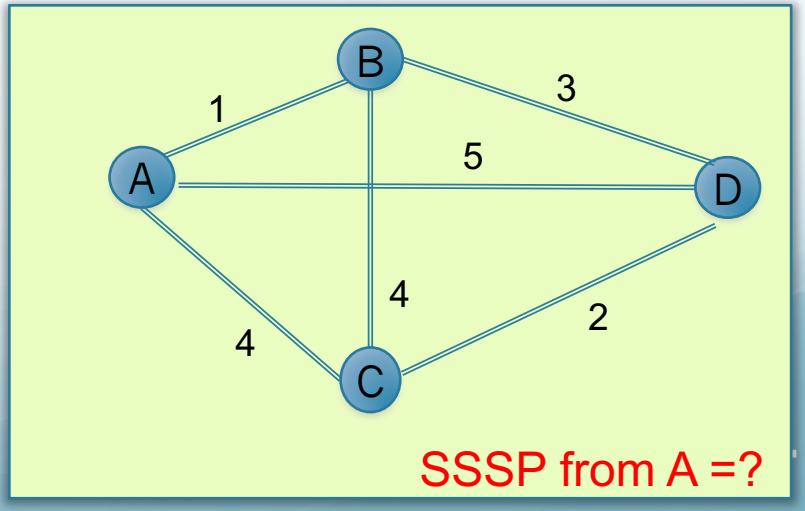
Perhaps by editing BFS or DFS. DFS attached here.

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

Prim's Algorithm and Dijkstra's Algorithm

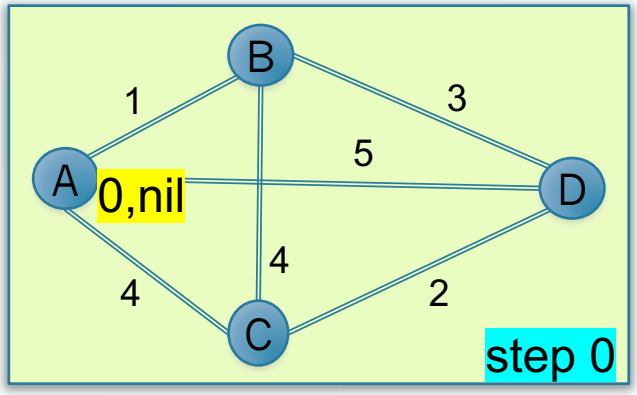
	Prim's	Dijkstra's
Aim	find a MST	find SSSP from a vertex s
Applied to	?	
Works on directed graphs?	?	
Works on unweighted graph?	?	
Related concepts		
spanning trees = ? MST = ? is MST unique?		paths A→D = ? shortest paths A→D = ? is shortest path from s→v unique?





Prim's Algorithm and Dijkstra's Algorithm

	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?		
	Related concepts	
	spanning trees = ? MST = ? is MST unique?	paths = ? shortest paths = ? is shortest path from s→v unique?



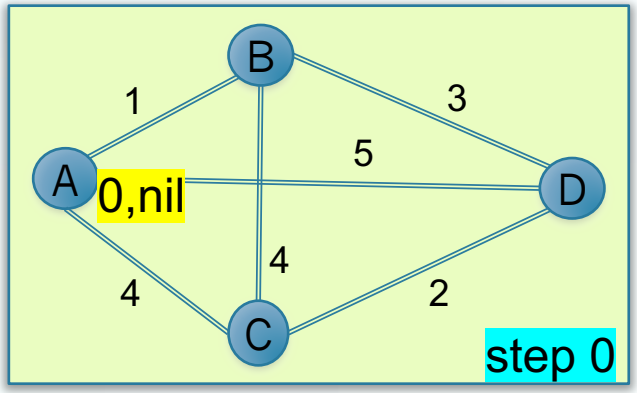
Running Prim's Algorithm to find a MST

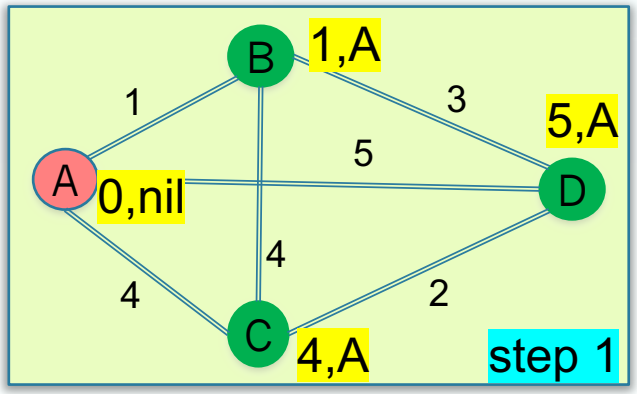
At each step, we add a node to MST.

We choose the node with minimal 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



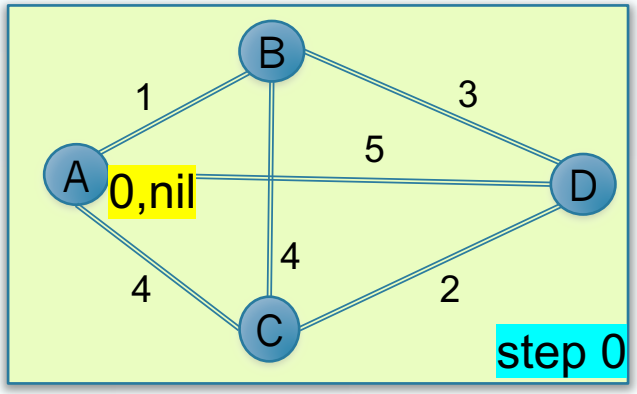


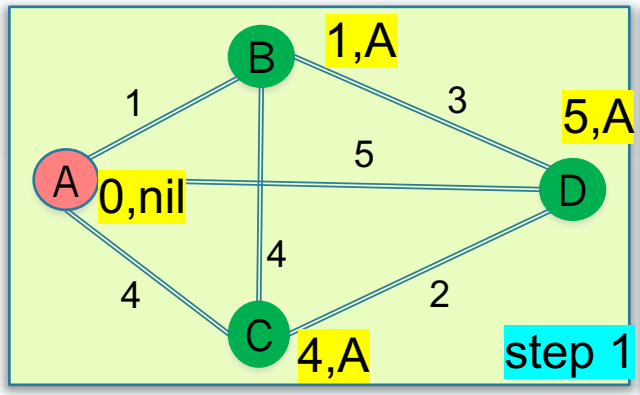
Running Prim's Algorithm to find a MST

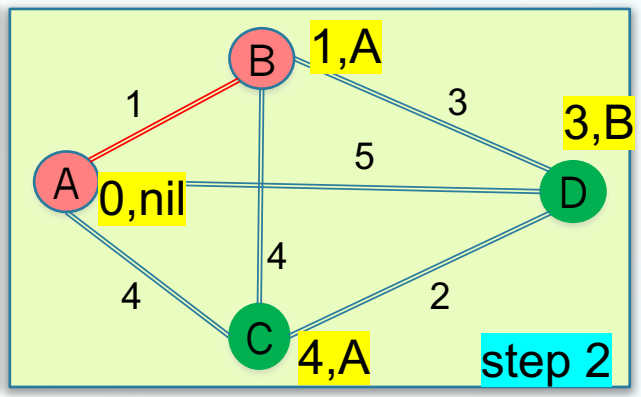
Step 1:

- 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





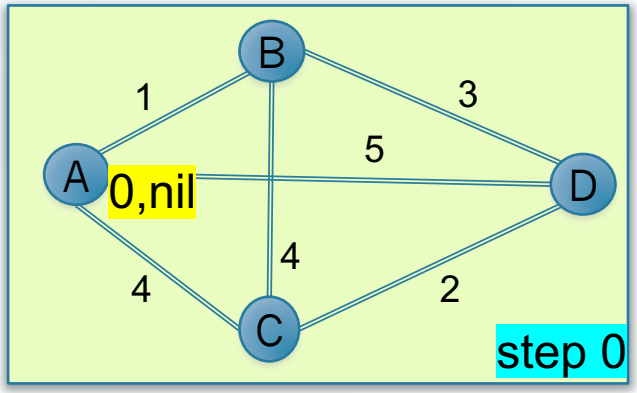


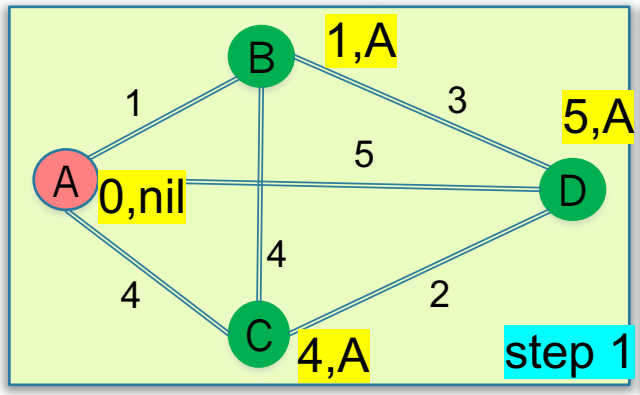
Running Prim's Algorithm to find a MST

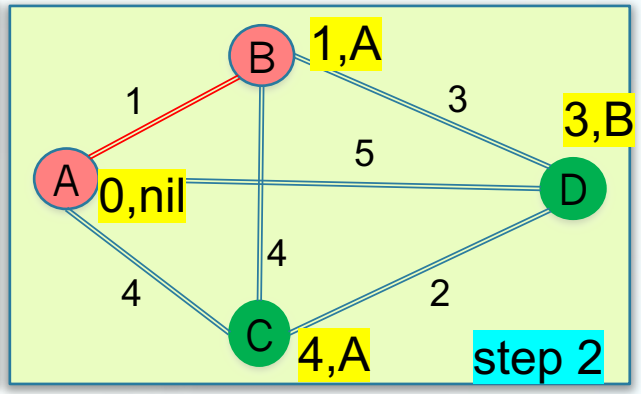
Step 2:

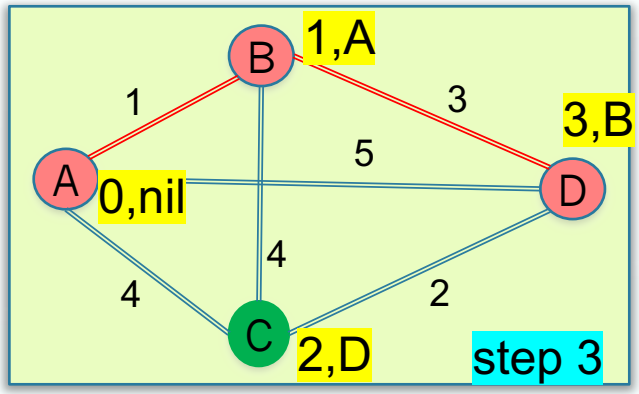
- B is chosen because it has the min cost amongst the green nodes (the green nodes and blue nodes are in the queue).

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

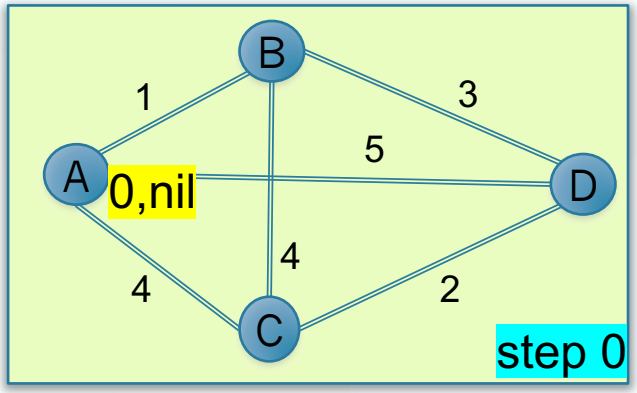


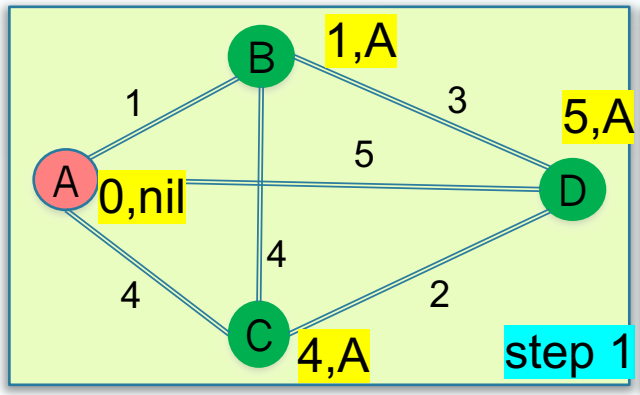


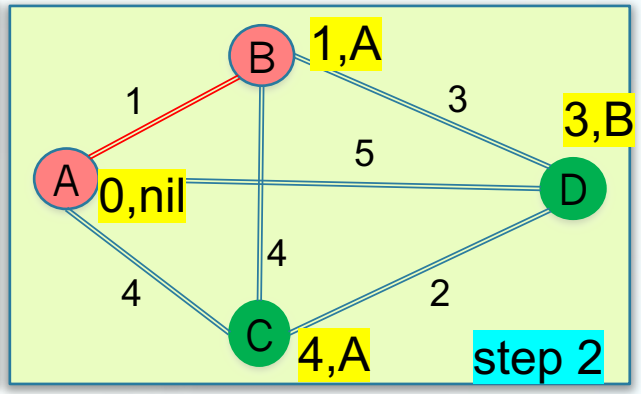




step	node added to MST	A	B	C	D
0		0,nil	∞,nil	∞,nil	∞,nil
1	A		1,A	4,A	5,A
2	B			4,A	3,B
3	D			2,D	

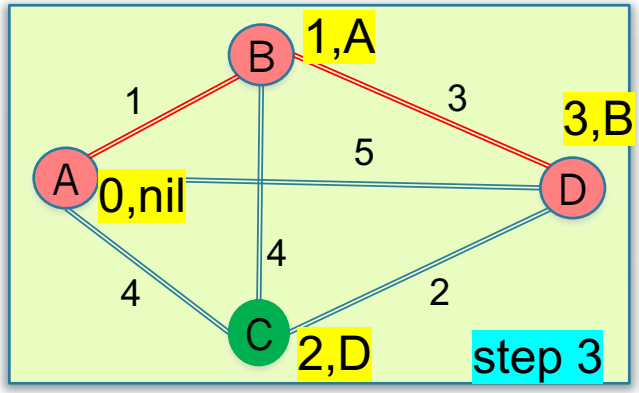


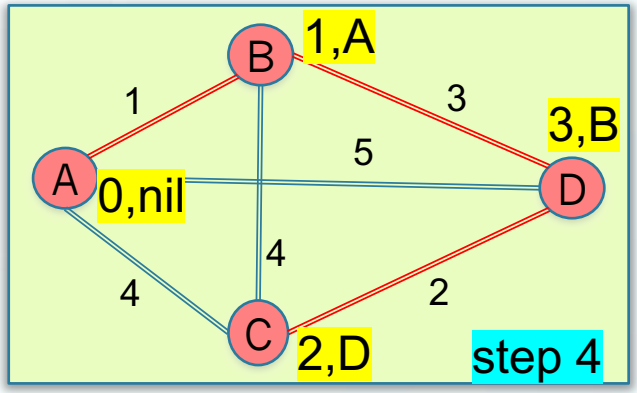




The final MST has cost= 0(A) + 1(B) + 2(C) +3(D)= 6

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.*



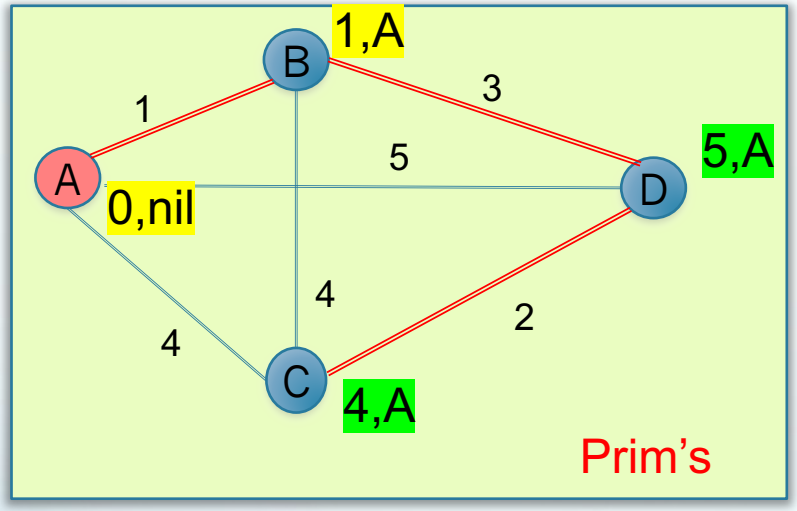


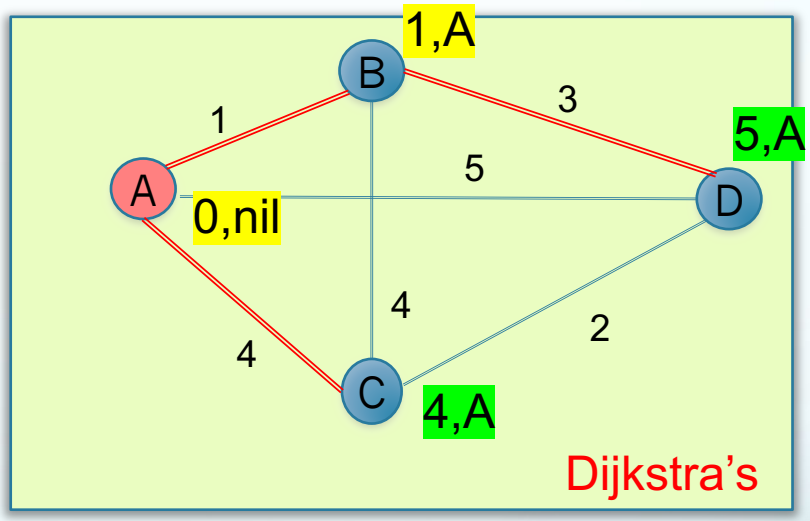
step	node added to MST	A	B	C	D
0		0,nil	∞,nil	∞,nil	∞,nil
1	A		1,A	4,A	5,A
2	B			4,A	3,B
3	D			2,D	
4	C				

Dijkstra's and Prim's are similar, how?

Here are situation after step 1:

- what happens in step 2 for Prim's?
- what should be in step 2 for Dijkstra?





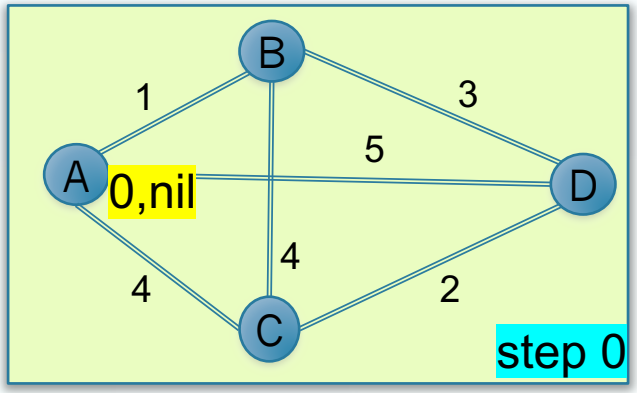
Dijkstra's and Prim's are similar

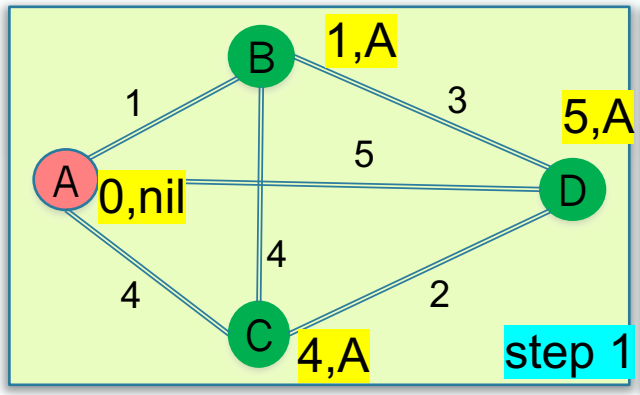
Dijkstra(G=(V,E),S)	Prim(G=(V,E))
SSSP (that involves all nodes of a *connected* graph)	MST (that involves all nodes of a *connected* graph)

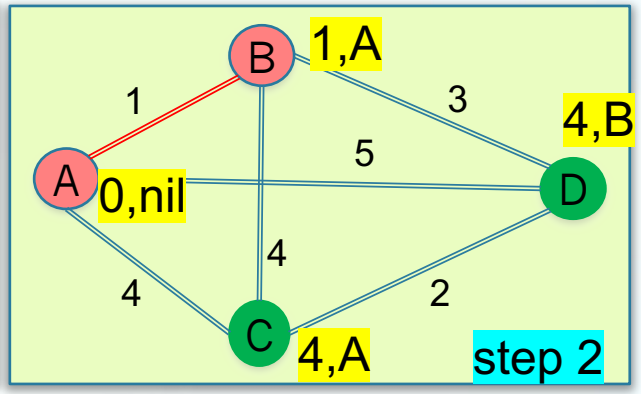
`for each v ∈V do`
`  dist[v]:= ∞`
`  prev[v]:= nil`
 	
`dist[S]= 0`
`PQ:= create_priority_queue(V,dist) with`
`dist[v] as priority of v∈V`
	`for each v ∈V do`
	`  cost[v]:= ∞`
	`  prev[v]:= nil`
	pick initial s
	`cost[S]:=0`
	`PQ:= create_priority_queue(V, cost)`
	`with cost[v] as priority of v∈V`

`while (PQ is not empty) do`
`  u := ejectMin(PQ)  // node with minimal`
`  // dist in PQ`
	`while (PQ is not empty) do`
	`  u := ejectMin(PQ)`

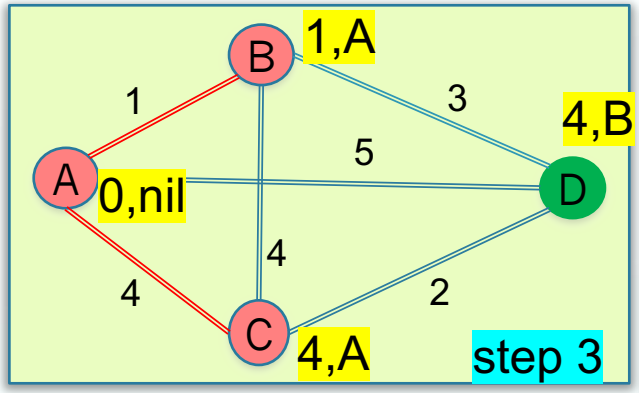
`  for each neighbour v of u do`
 	
`  if v∈PQ & dist[u]+w(u,v) < dist[v] then`
`  dist[v]:= dist[u]+w(u,v)`
`  update (v, dist[v]) in PQ`
`  prev[v]:= u`
	`  for each neighbour v of u do`
	
	`  if v∈PQ & w(u,v) < cost[v] then`
	`  cost[v]:= w(u,v)`
	`  update (v, cost[v]) in PQ`
	`  prev[v]:= u`

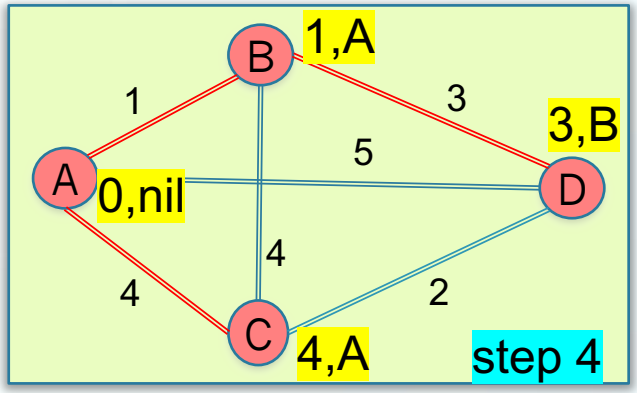






Running Dijkstra from node A, ie. finding shortest paths from A to all nodes.
The number at each **node** is the total distance from **A** to this **node** = distance of *previous node* + weight of the edge (*previous node*, **node**)



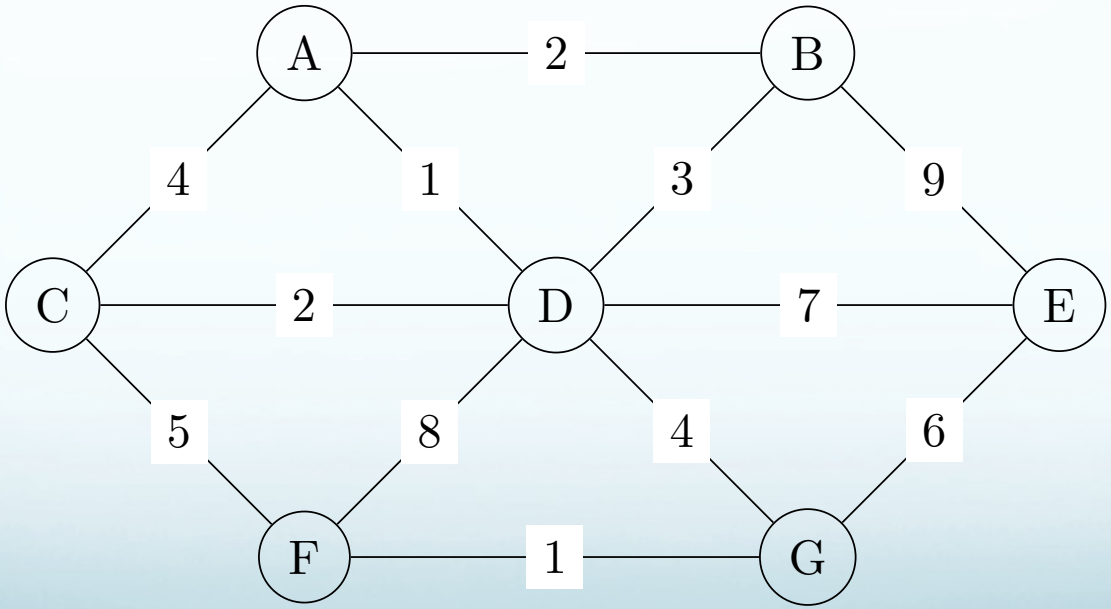


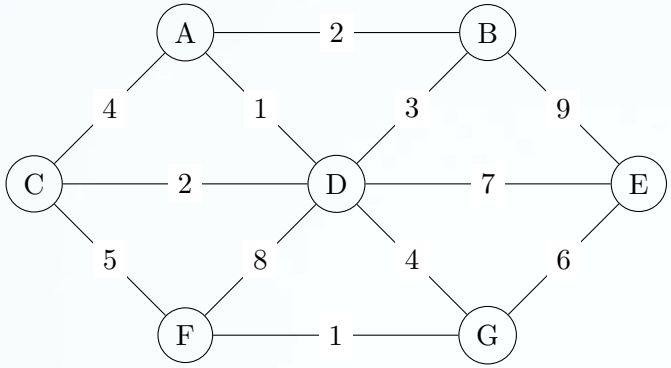
step	node added to MST	A	B	C	D
0		0,nil	∞,nil	∞,nil	∞,nil
1	A		1,A	4,A	5,A
2	B			4,A	4,B
3	C				4,B
4	D				

Q4.9: 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





DA from A

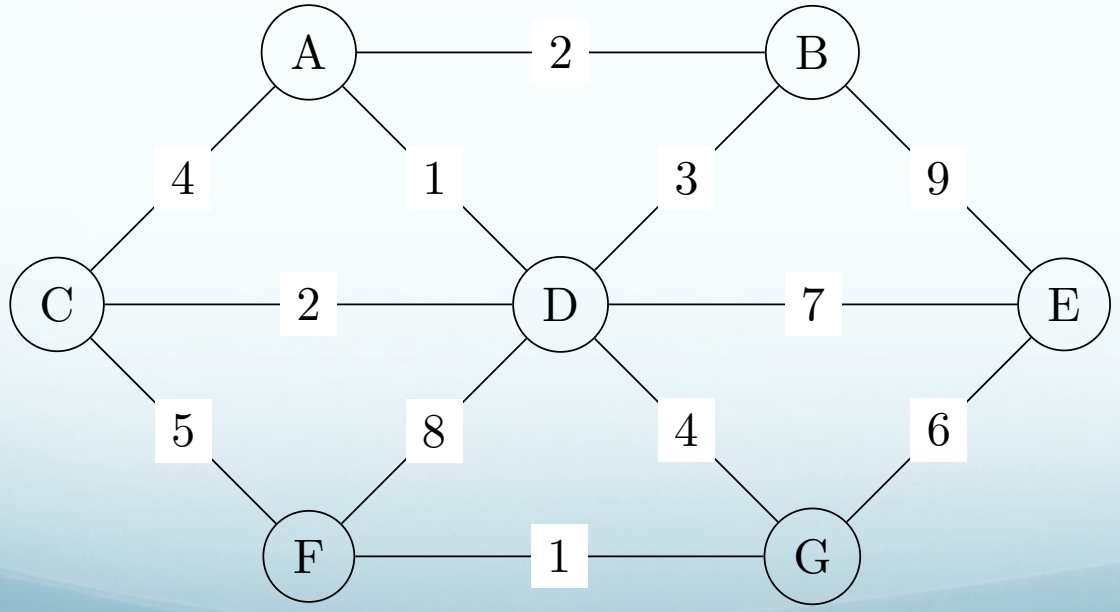
How long, *and what is*, the shortest path from E to A?

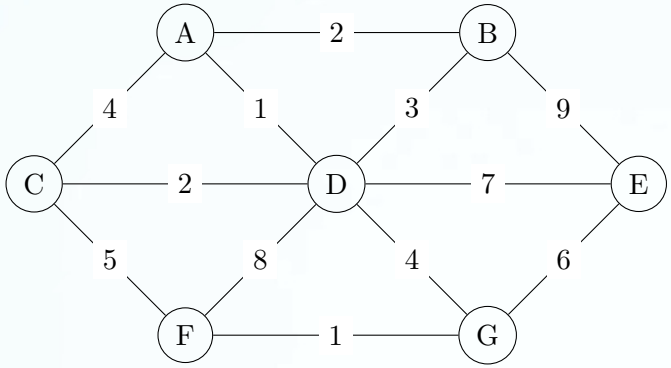
step	node done	A	B	C	D	E	F	G
0		0/nil	∞/nil	∞/nil	∞/nil	∞/nil	∞/nil	∞/nil
1								
2								
3								
4								
5								
6								
7								

Problem 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?





Run Prim's Alg

Break ties using alphabetic order.

Draw the resulted MST

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								

assignment 1

- Do it early! Submit early, resubmit if needed!
- 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.

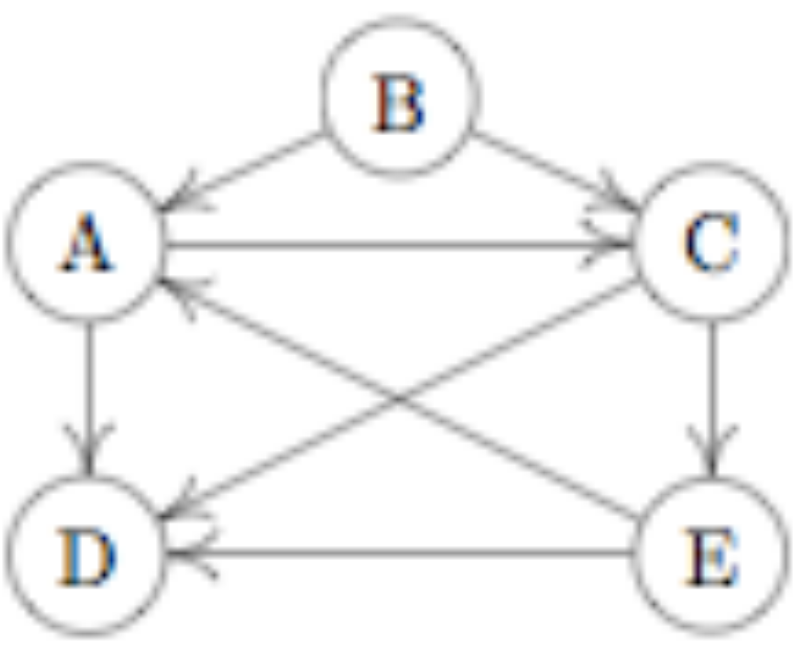
LAB

- 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

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



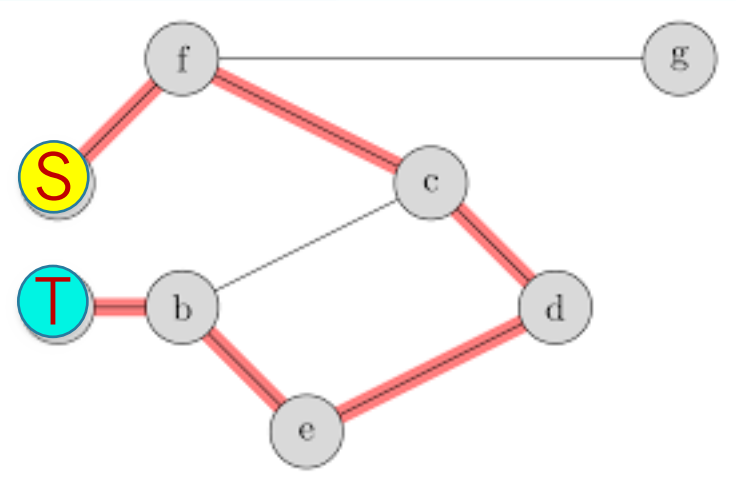
A DFS of a di-graph can be represented as a collection of trees. Each edge of the graph can then be classified as a *tree edge*, a *back edge*, a *forward edge*, or a *cross edge*. A tree edge is an edge to a previously un-visited node, a back edge is an edge from a node to an ancestor, a forward edge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

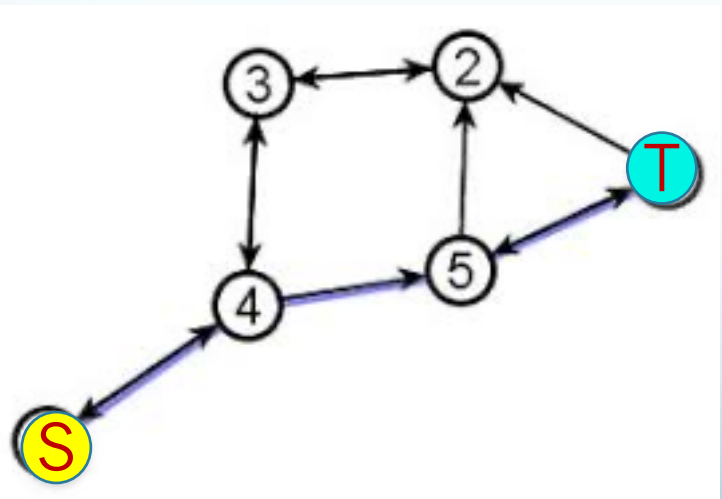
Draw a DFS tree based on the following graph, and classify its edges into these categories.

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.

A gift from Anh: If you are a bit bored or tired, skip this exercise, and instead use `digraph_dfs_demonstration.pdf` from `github` to entertain yourselves ;-).

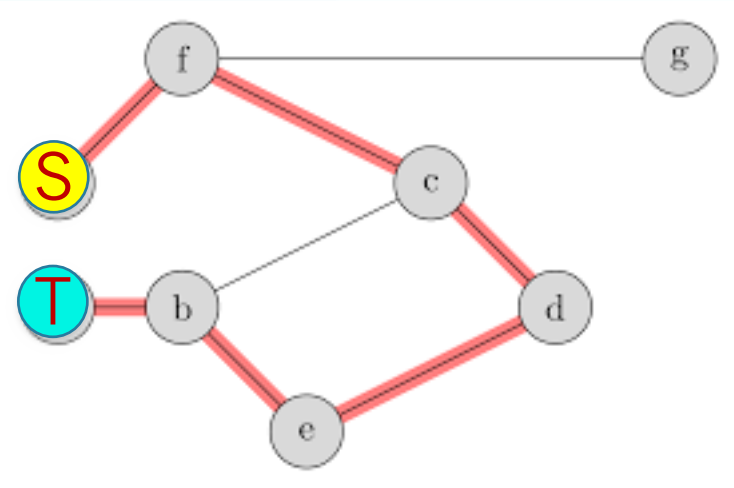
Paths in unweighted graphs: path length, shortest path



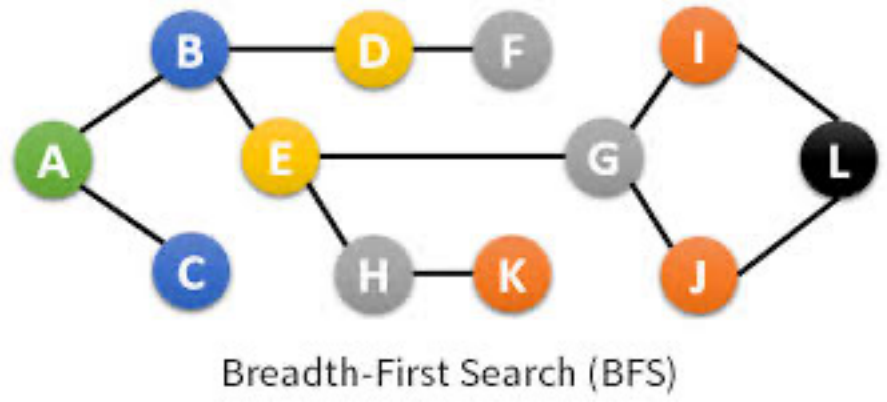


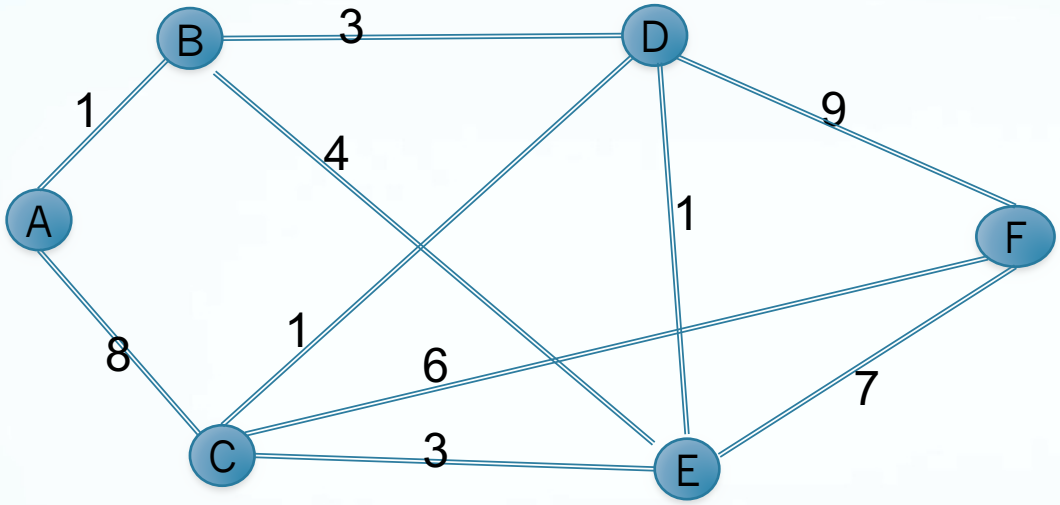
- For finding a path from S to T can we use DFS? BFS?
- For finding a shortest path (path with least number of edges) from S to T can we use DFS? BFS?
- Applied to directed graphs? cyclic graphs?

Paths in unweighted graphs: path length, shortest path



- For finding a path from S to T can we use DFS? BFS? **Yes to both.**
- For finding a shortest path from S to T can we use DFS? BFS? **Yes for BFS, No for DFS.**
- Applied to directed graphs? cyclic graphs? **Yes, Yes.**



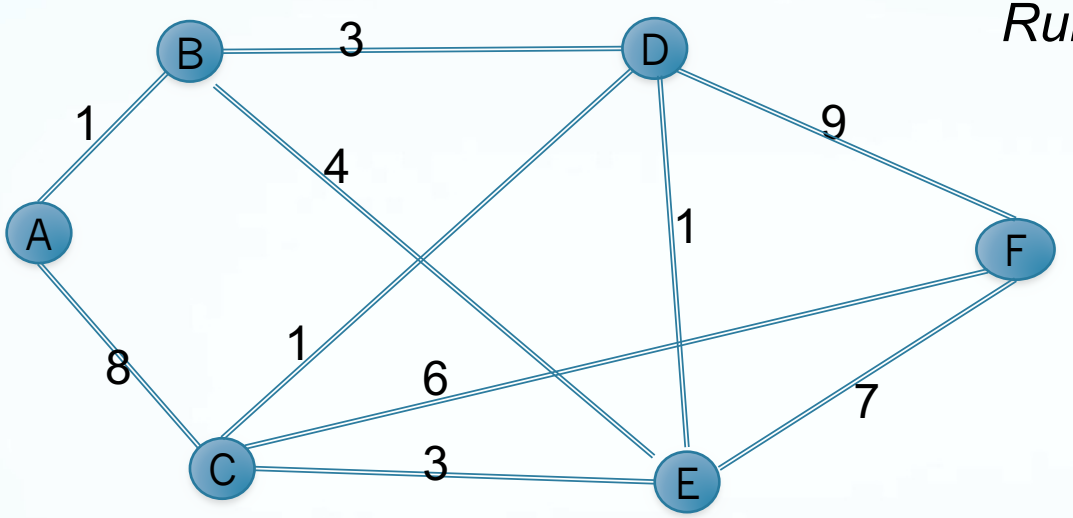


A Task:

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

Run Dijkstra's Algorithm from A



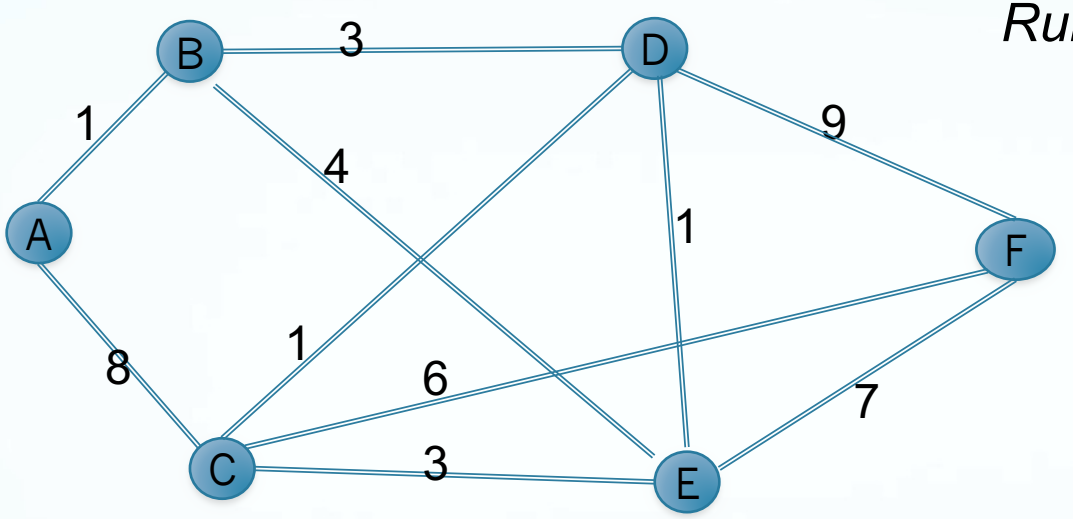
done	A	B	C	D	E	F
	0, nil	∞,nil	∞,nil	∞,nil	∞,nil	∞,nil
A						

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

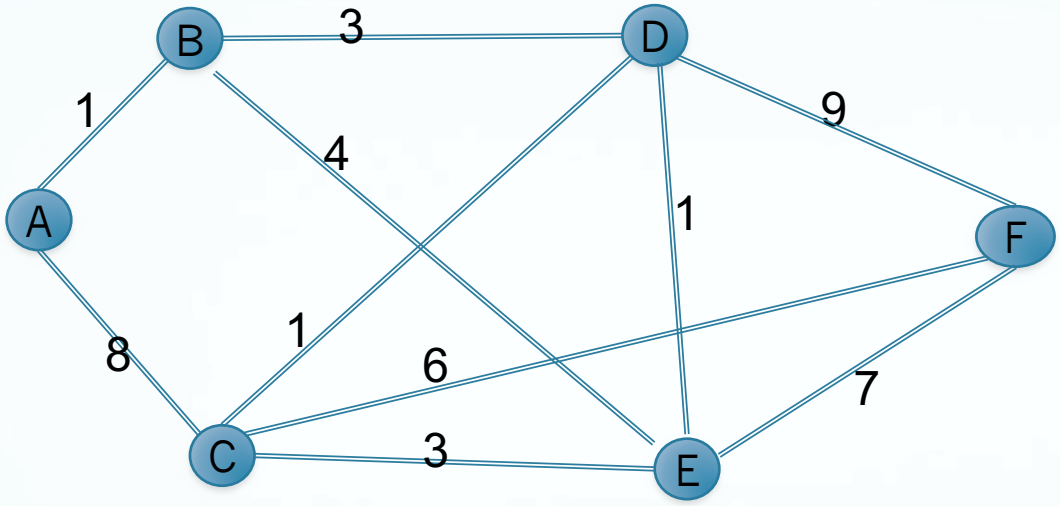


The dist at A is 0, there is an edge A->C with length 8, so we can reach C from A with distance 0+8, and 8 is better than previuosly-found distance of ∞

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

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

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						