COMP20007 Workshop Week 5

0	**Preparation:** - download this slide.pptx (or pdf) from github.com/anhvir/c207 - (optional) open `wokshop6.pdf` (from `LMS`)
1	**Topic 1:** DFS and BFS: Problems 3, 5
2	**Topic 2:** Dijkstra's and Prim's Algorithms; Problem 9 ***Group Work:*** Problems 9, 8, 4, 6, 7 (preferably in that order)
LAB	**Q&A on Practice MST** **Assignment 1 Q&A**, and -make sure that you can scp, ssh, use dimefox to test/submit -make sure you understand how to start the programming part -do Assignment 1 or review for MST & ask questions

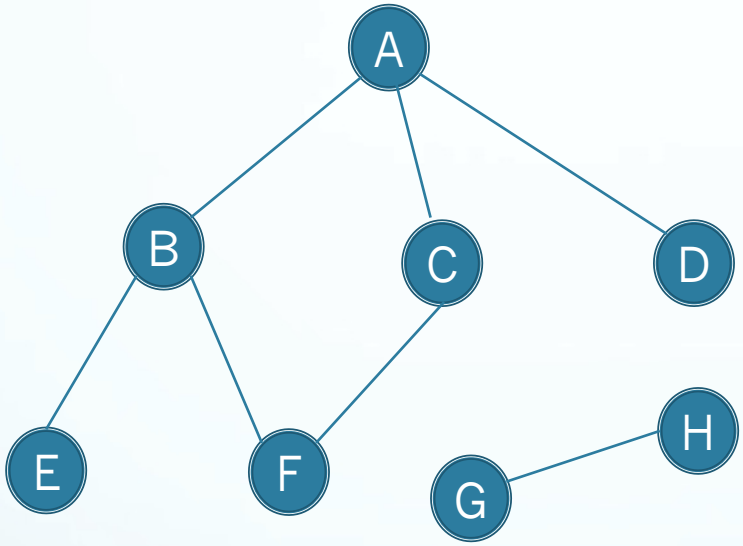
DFS & BFS

Example: Perform DFS & BFS in the tree:

DFS and BFS: what & how?

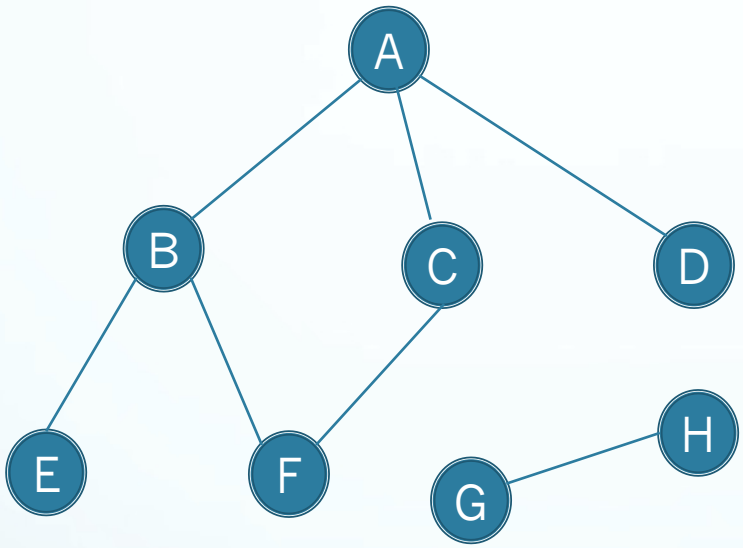
- Notes: In exercises, ties should always be broken in alphabetical/ascending order

Compare: `DfsExplore(A)` and `BfsExplore(A)`



DfsExplore/ BfsExplore?

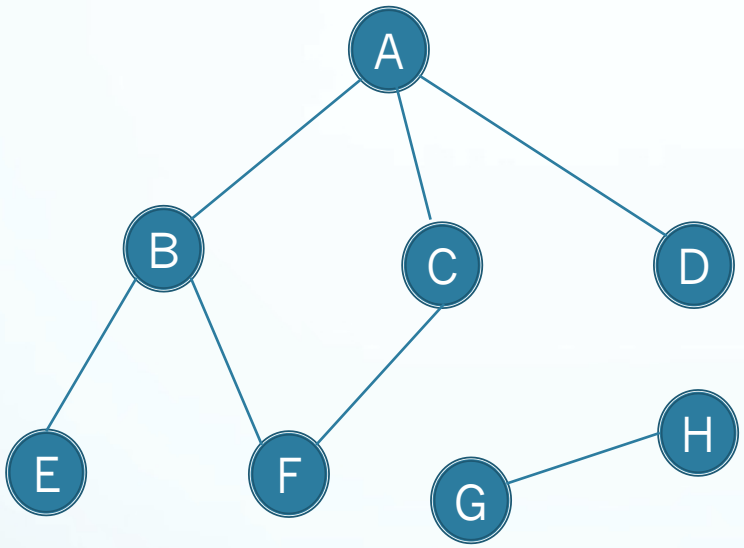
- Notes: In exercises, ties should always be broken in alphabetical/ascending order



functiom `DfsExplore((V,E), u)`
  mark u as visited
  ?

DFS and BFS: what & how?

- Notes: In exercises, ties should always be broken in alphabetical/ascending order

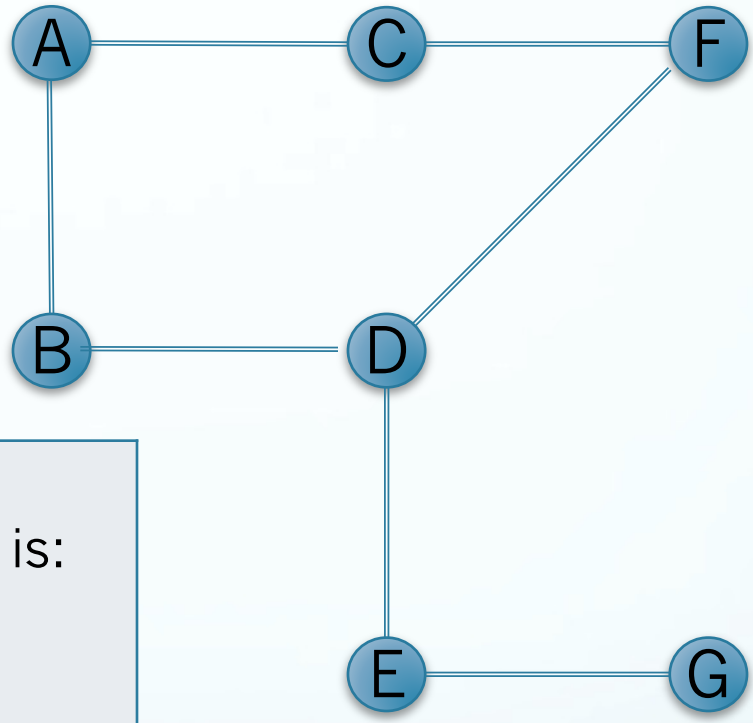



```
function DFS((V,E))
  for each v ∈ V do mark v as unvisited
  for each v ∈ V do
     if (v is unvisited) then
       DfsExplore(v)
```

function BFS((V,E))

Problem 4: 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

Problem 6: 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 for proof).
Note: skip part b) if it takes you more than 2 minutes, you can do it later!

USE THE BFS TO WRITE A PSEUDOCODE FOR PART a):

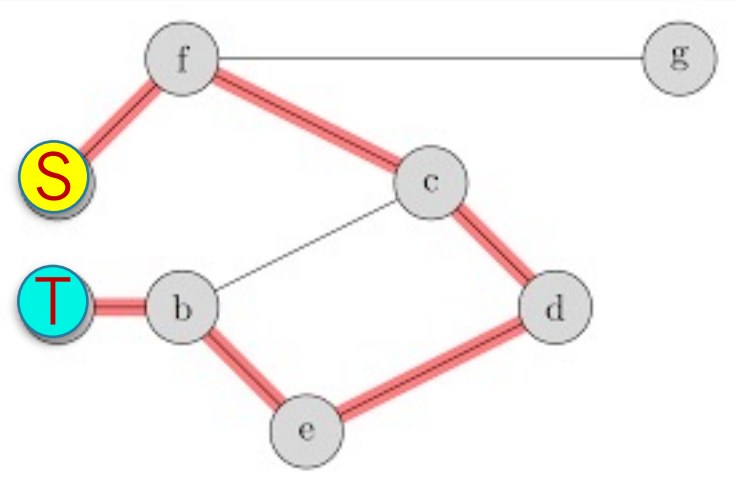
function isCyclic((V,E))
…

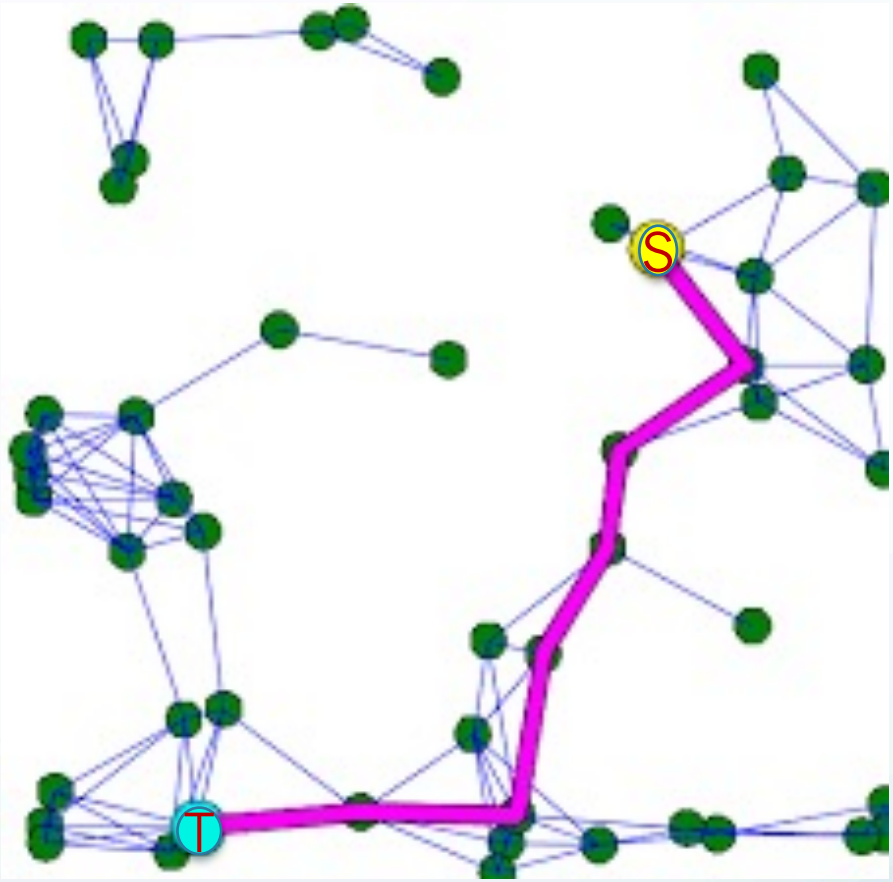
BFS algorithm from lecture

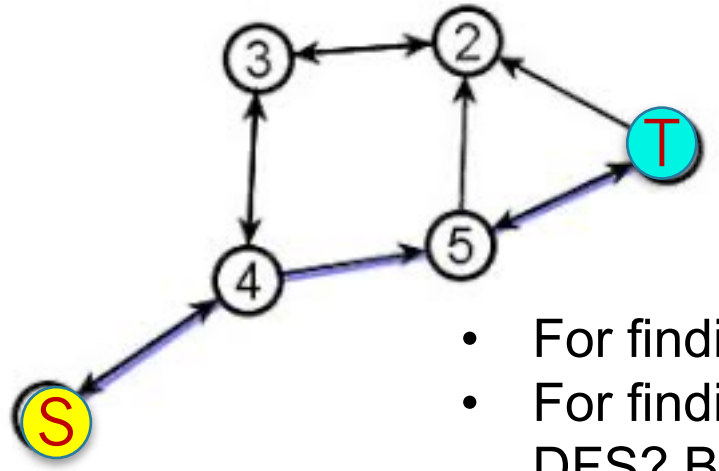
```
function BFS(⟨V, E⟩)
    mark each node in V with 0
    count ← 0, init(queue)
    for each v in V do
        if v is marked 0 then
            BfsExplore(v)
```

```
function BfsExplore(v)
    count ← count + 1
    mark v with count
    inject(queue, v)                    ▷ queue
    while queue is non-empty do
        u ← eject(queue)
        for each edge (u, w) adjacent to u do
            if w is marked with 0 then
                count ← count + 1
                mark w with count
                inject(queue, w)
```

Paths in unweighted graphs: path length, shortest path

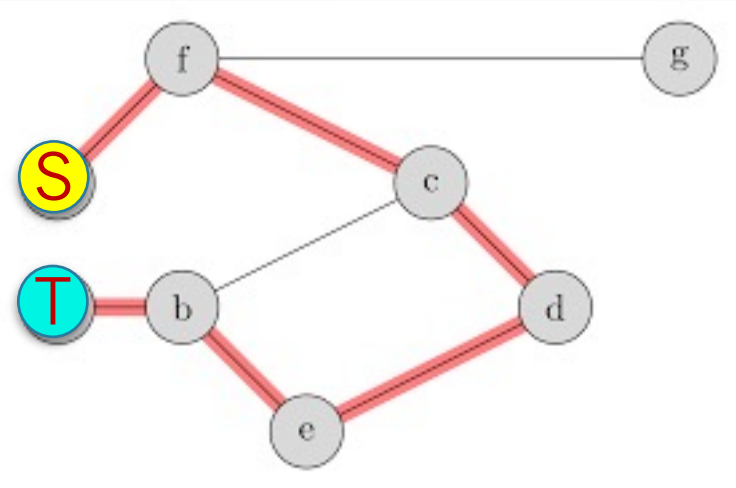






- For finding a path from S to T can we use DFS? BFS?
- For finding a shortest path from S to T can we use DFS? BFS?
- Applied to directed graphs? cyclic graphs?

Paths in unweighted graphs: path length, shortest path

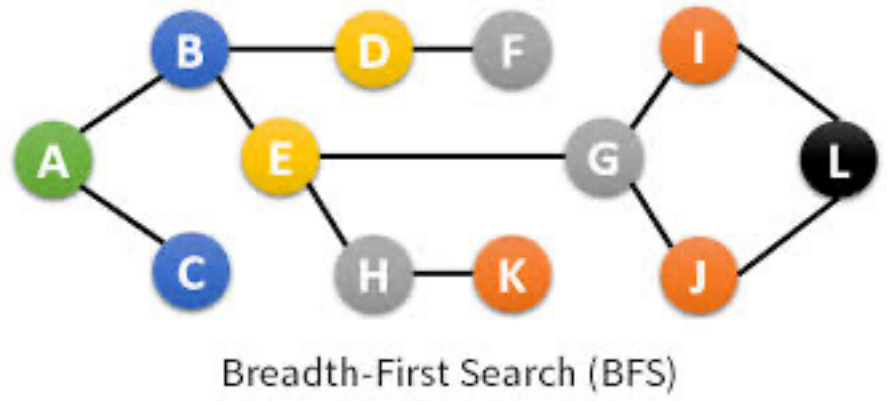


- For finding a path from S to T can we use DFS? BFS? **Yes to both.**
- For finding a shortest path from S to T can we use DFS? BFS? **Yes for BFS, No for DFS.**
- Applied to directed graphs? cyclic graphs? **Yes, Yes.**

How to retrieve the path with DFS?

How to retrieve the path with BFS?

How about shortest path in weighted graphs?



Breadth-First Search (BFS)

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) from s to all other vertices.

Dijkstra's Algorithm as a (special) BFS

- Basic idea

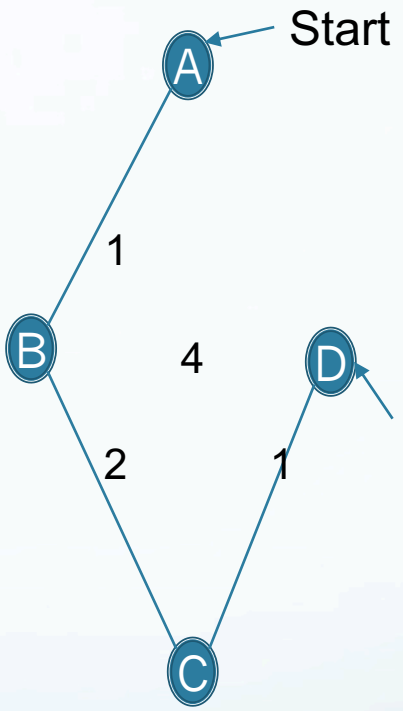
  if `s` → `A` → `B` is a shortest path then `s` → `A` is a shortest path.

- Init:
  - start with `dist[s]= 0`, and `dist[*]=` , set `notfound_set= V`

Round 1:

- choose node with min `dist[],` which is `s,` remove it from `notfound_set`;
- visit all nodes `u` adjacent to `s` and update `dist[u]`;

- Round 2:
  - choose the node with min `dist[]` from `notfound_set`
  - do the other steps as in Round 1

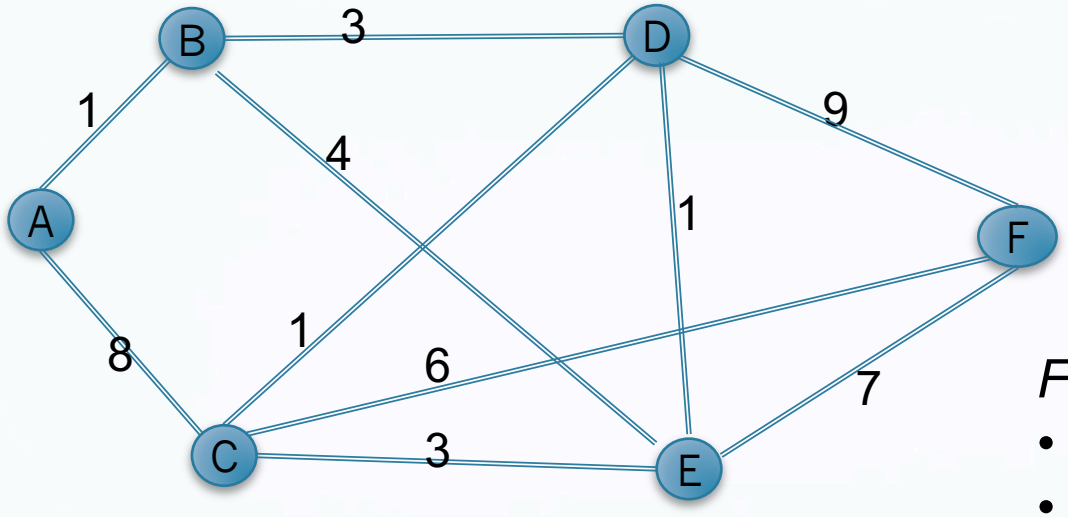


Dijkstra's algorithm [conceptual only]

Purpose: Find shortest path from vertex s

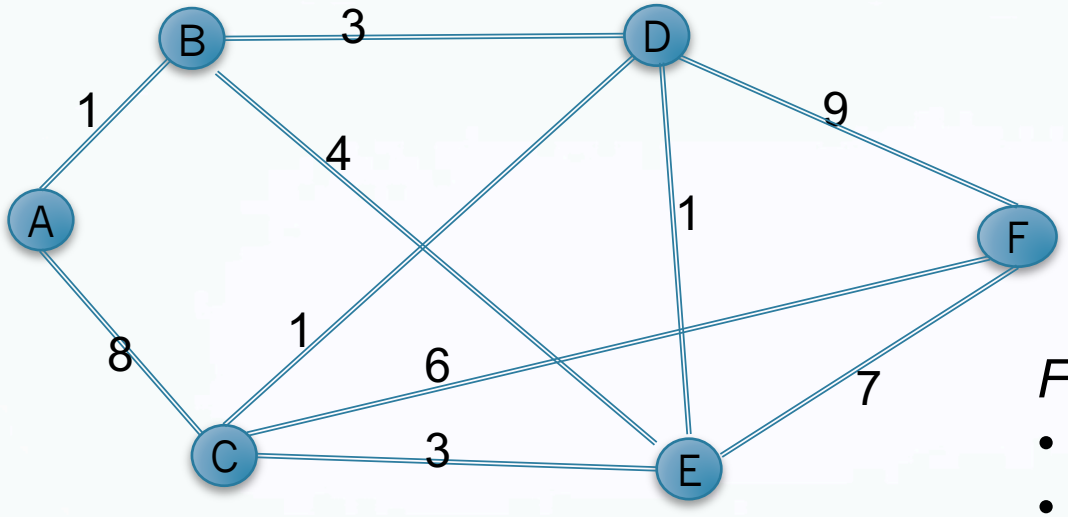
- set `dist[u] = ∞`, `pred[u]=nil` for all `u`,
- set `dist[s]= 0`;
- Insert all pair (dist[u], pred[u]) into a priority queue PQ

```
while (PQ is not empty):
   remove u from PQ (so that dist[u] is smallest)
   mark: found shortest path for u
   for all (u,v) in G:
     if (dist[v] > dist[u]+w(u,v):
           update (dist[v],pred[v]) in PQ
```



Find a shortest path:

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



Find a shortest path:

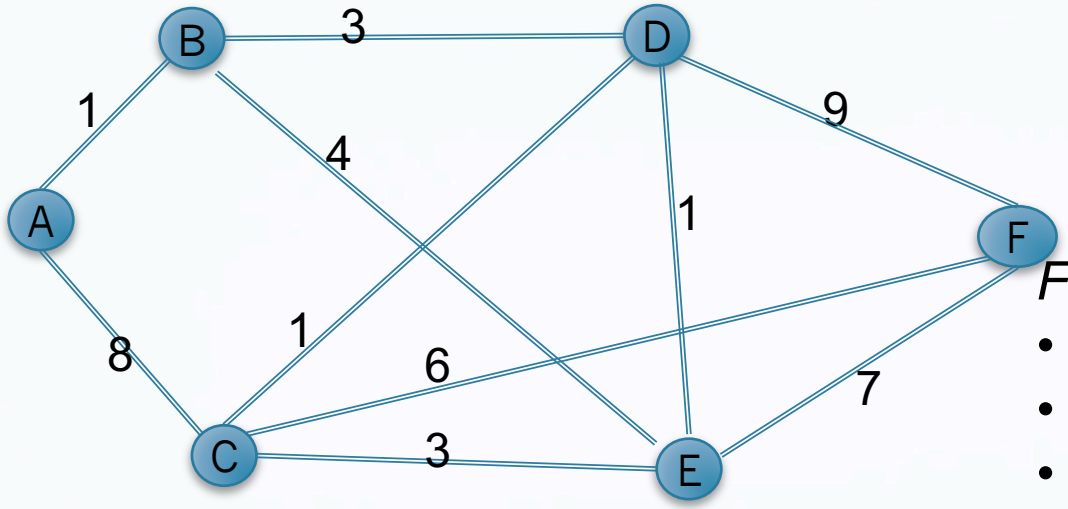
- From A to B
- From A to C
- From A to F
- From A to any other node

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



Find a shortest path:

- From A to B
- From A to C
- From A to F
- SP A->F=

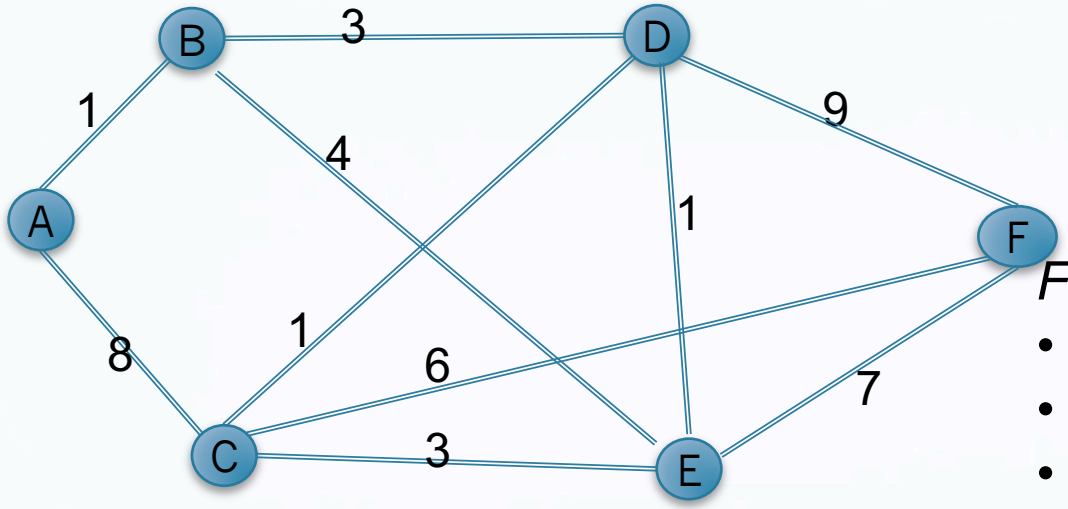
4 The dist at SA is 0, there is an edge A->C with length 8, so we can reach C from A with distance 0+8, and 8 is better than previuosly-found distance of ∞

The "-" is shorthand for "∞,nil"

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,**

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:

- From A to B
- From A to C
- From A to F
- SP A->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

Dijkstra's algorithm: Q&A

- Purpose: Single Source Shortest Path SSSP

Prim's Algorithm

- Purpose: MST
- Concepts:
  - ST,
  - MST,
  - is MST unique?

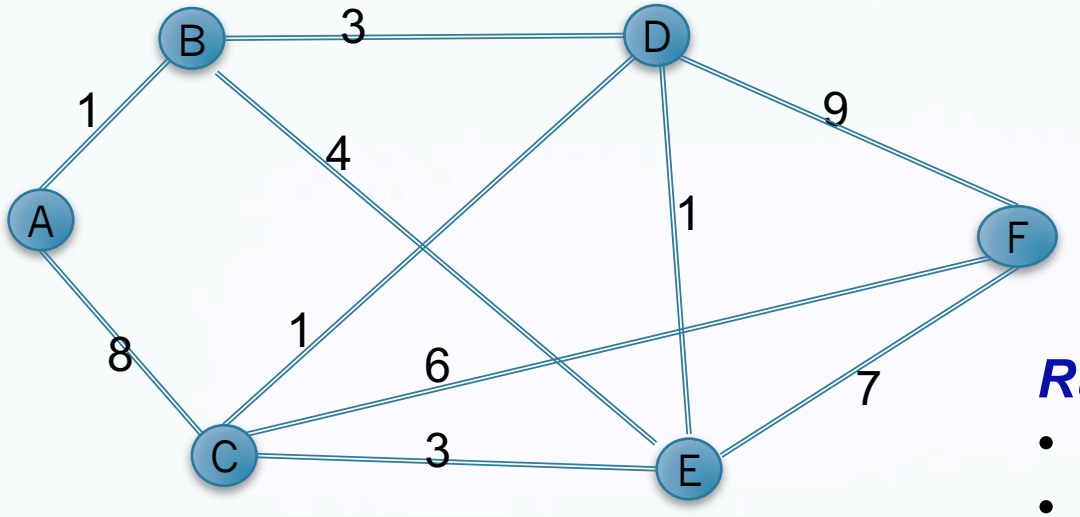
Dijkstra's and Prim's are similar

Dijkstra(G,S)	Prim(G)
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)`
	`for each v ∈V do`
	`  dist[v]:= ∞`
	`  prev[v]:= nil`
	pick initial s
	`dist[S]:=0`
	`PQ= create_priority_queue(V)`

`while (PQ is not empty) do` `  u := eject node with minimal dist[u] from PQ`	`while (PQ is not empty) do` `  u := eject node with minimal dist[u] from PQ`

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



Run Prim to find MST:

- start from any node
- supposing start from A

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

this column: nodes with shortest path found

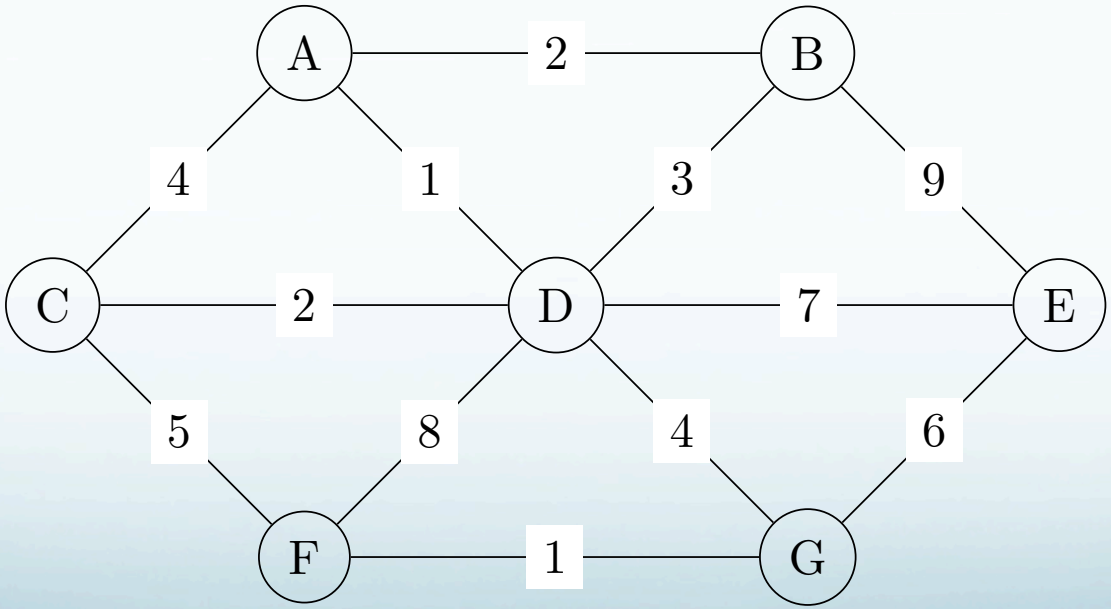
`dist[B]`: weight contributed to MST when adding A

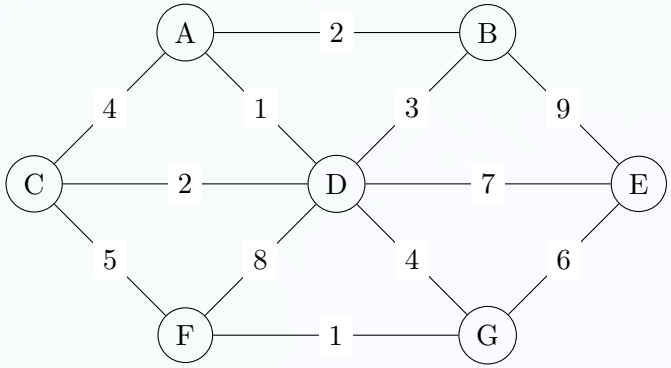
`pred[D]`: node that precedes D in the path A→D

Problem 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

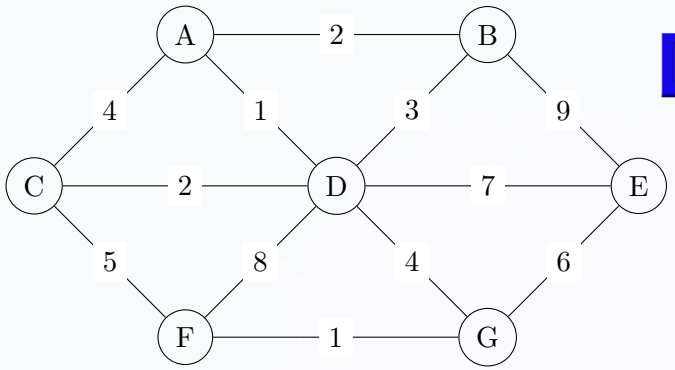




DA from A

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

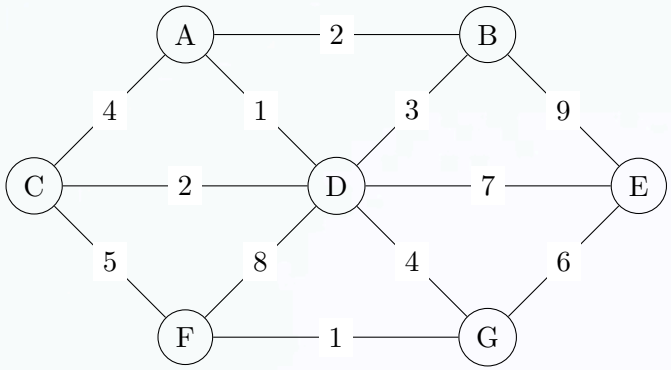
step	node visited	A	B	C	D	E	F	G
0		0/nil	∞/nil	∞/nil	∞/nil	∞/nil	∞/nil	∞/nil
1								
2								
3								
4								
5								
6								
7								



Dijkstra's Algorithm from E

dist[E] prev[E]

step	node ejected	A	B	C	D	E	F	G
0		∞/nil	∞/nil	∞/nil	∞/nil	**0**/nil	∞/nil	∞/nil
	E	∞/nil	**9,E**	∞/nil	**7,E**		∞/nil	**6,E**
	G	∞/nil	**9,E**	∞/nil	**7,E**		**7,G**	
	D	**8,D**	**9,E**	**9,D**			**7,G**	
	F	**8,D**	**9,E**	**9,D**				
	A		**9,E**	**9,D**				
	B							



DA from E

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

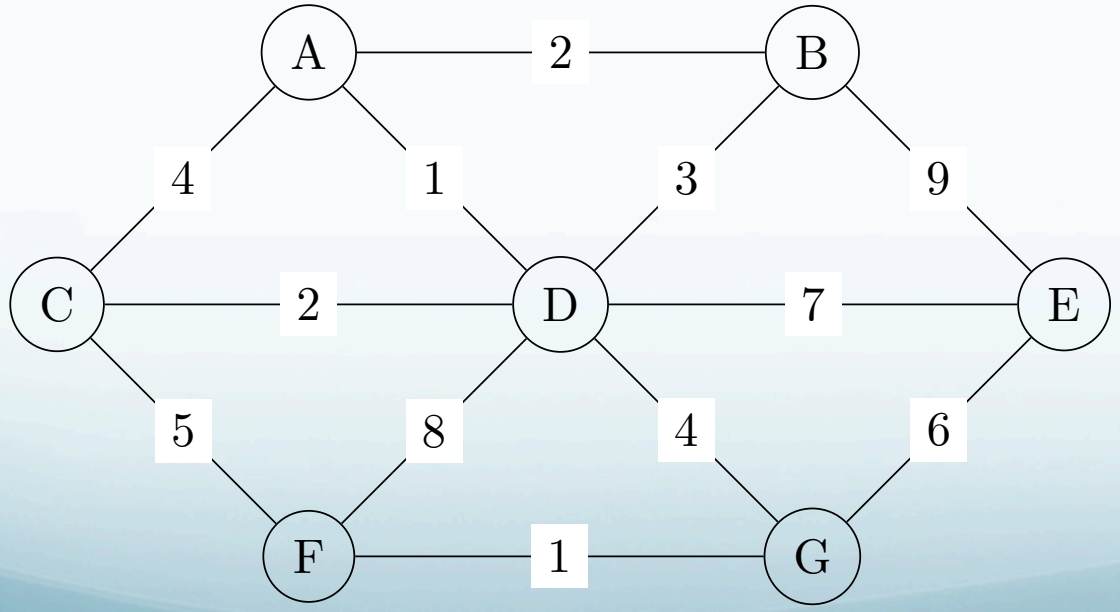
path= ?

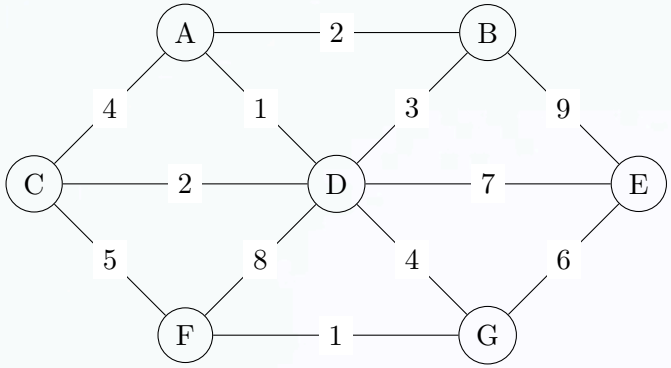
step	node ejected	A	B	C	D	E	F	G
0		∞/nil	∞/nil	∞/nil	∞/nil	**0**/nil	∞/nil	∞/nil
1	E	~	9/E	~	7/E		~	**6**/E
2	G(6)		**9/E**		**7/E**		**7,G**	
3	D (7)	**8,D**	**9/E**	9,D			**7,G**	
4	F(7)	**8,D**	**9/E**	9.D				
5	A(8)		**9/E**	9/D				
6	B(9)			9/D				
7	D							

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?

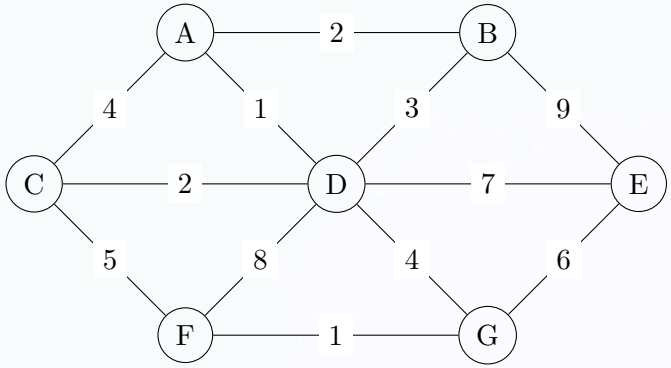




Run Prim's Alg

Break ties using alphabetic order.
Draw the resuklted MST

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



Prim's Alg from A

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

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	-	-	-
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					**6G**		
7	G							

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 hates 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.

github.com/anhvir/c207/Lab_Notes.pdf (how to work with dimefox)

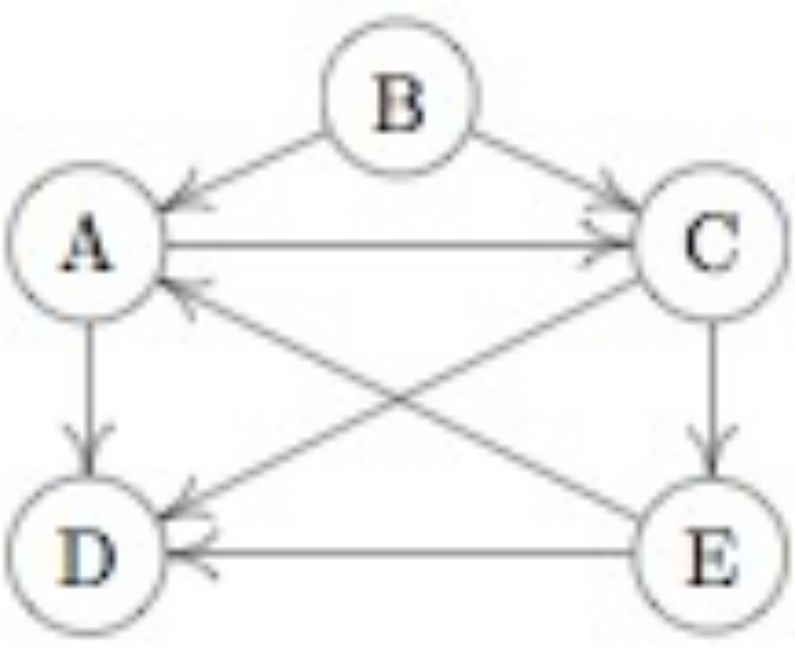
LAB:

- work on not-yet-done this week's workshop problems: problems 9, 8 (if not finished) and 5, 6, 7 (see further slides) note that the solution for this week already posted on LMS

- prepare for MST: **see sample MST test and solution from LMS**
- work on ass1

Lab: do assmt1 or your choice

Problem 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 ;-).

Problem 6: 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 for proof).
Note: skip part b) if it takes you more than 2 minutes, you can do it later!

YOUR BRIEF ANSWER:

BFS algorithm from lecture

function BFS($\langle V, E \rangle$)
    mark each node in V with 0
    $count \leftarrow 0$, *init*(queue)
    for each v in V **do**
        if v is marked 0 **then**
            BfsExplore(v)

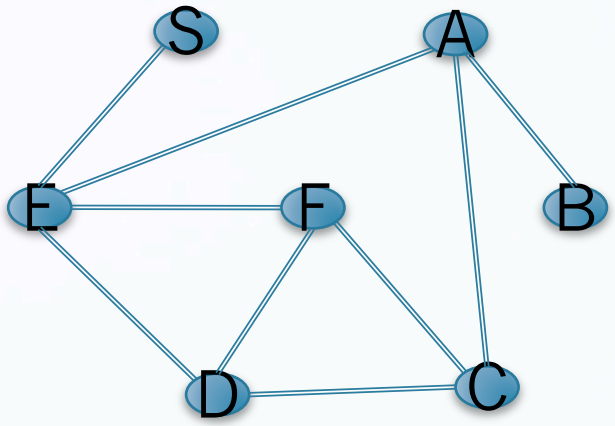
function BfsExplore(v)
    $count \leftarrow count + 1$
    mark v with *count*
    inject(*queue*, v) ▷ queue
    while *queue* is non-empty **do**
        $u \leftarrow$ *eject*(*queue*)
        for each edge (u, w) adjacent to u **do**
            if w is marked with 0 **then**
                $count \leftarrow count + 1$
                mark w with *count*
                inject(*queue*, w)

Problem 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 and 2
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?

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