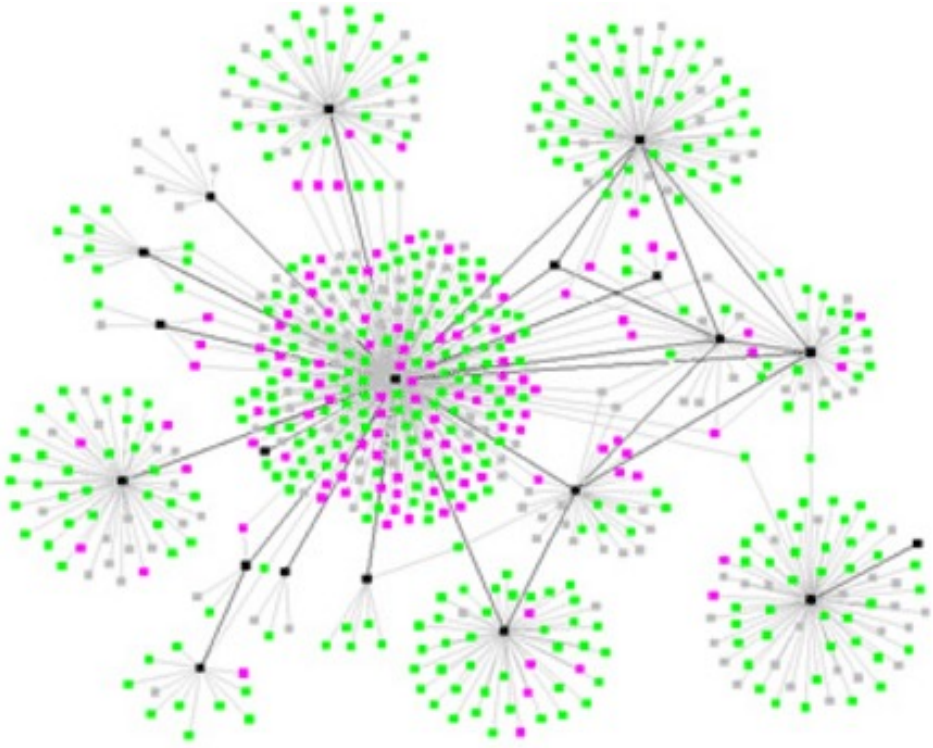
COMP20003 Workshop Week 10

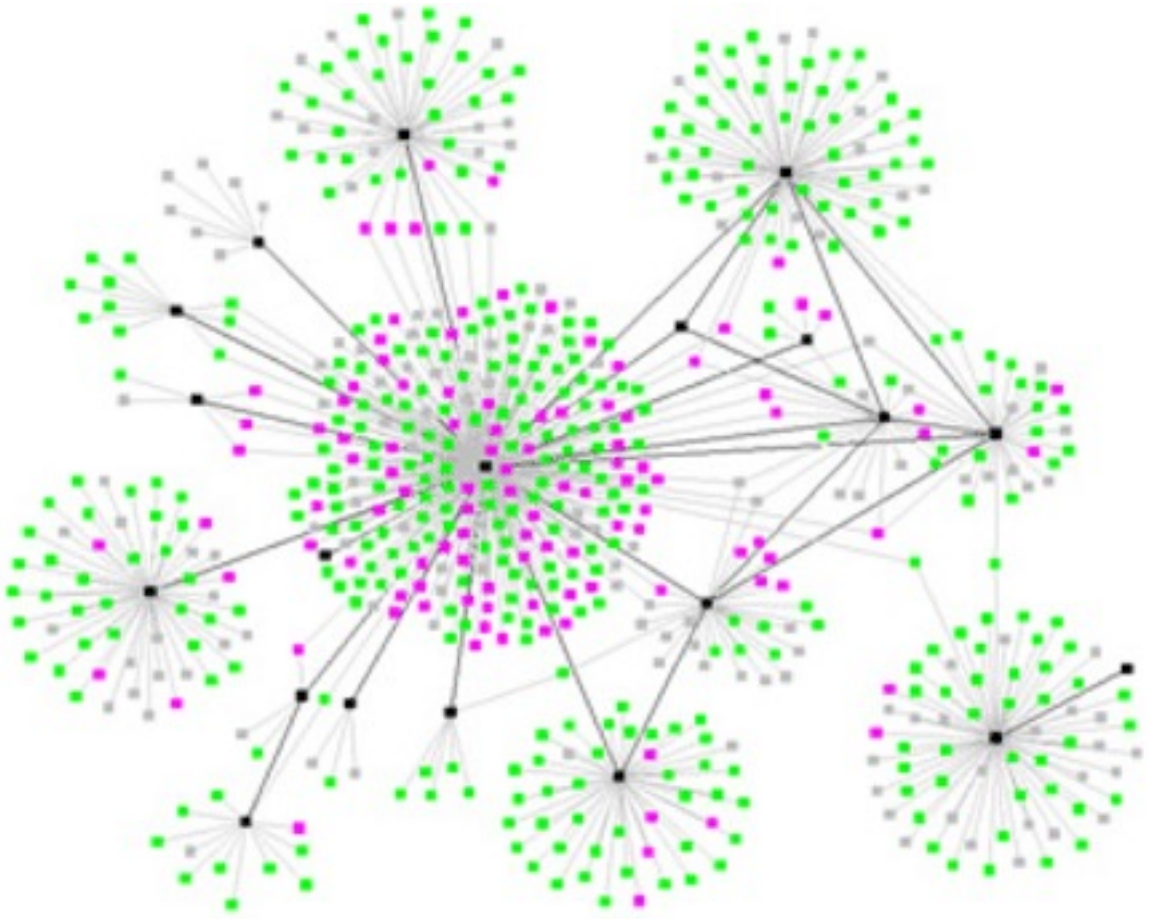
1	Graphs: concepts
2	Graph representation
3	DFS (and when to use?)
4	BFS
5	Dijkstra's Algorithm Lab: Implement Dijkstra's Algorithm Exam date= ?

Graphs: example



example

An example of the type of graph produced by contact tracing. Each node represents an infected individual, and each line connects two individuals who have been in contact



Graphs: Concepts

Formal definition: $\mathbf{G = (V, E)}$ where

$V = \{\mathbf{v_i}\}$: set of *vertices*, or *nodes*

$E = \{\mathbf{(v_i, v_j)} \mid v_i \in V, v_j \in V\}$: set of *edges*, or *links*;

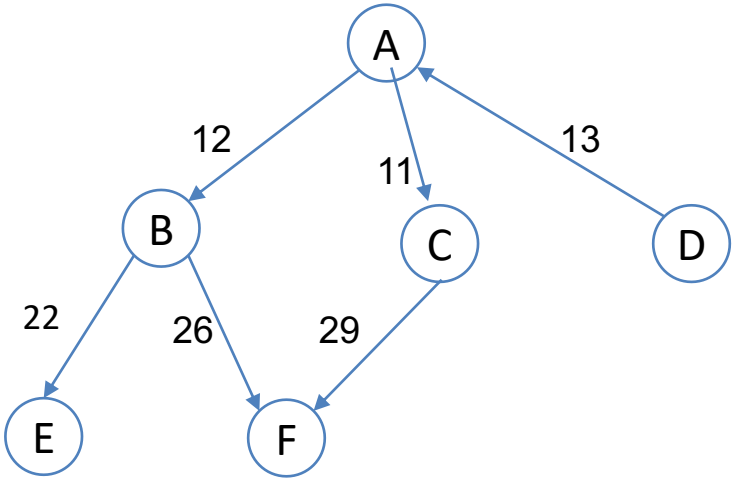
$|V|$ is called the *order* of the graph

$|E|$ is called the *size* of the graph

Understanding concepts:

- *dense* and *sparse* graphs
- *directed*, *di-graph*, *undirected*,
- *cyclic, acyclic*, *DAG*
- *connected and unconnected graph, connected component*
- *weakly and strongly connected components*

weighted di-graph representation: Using Adjacency Lists



input data

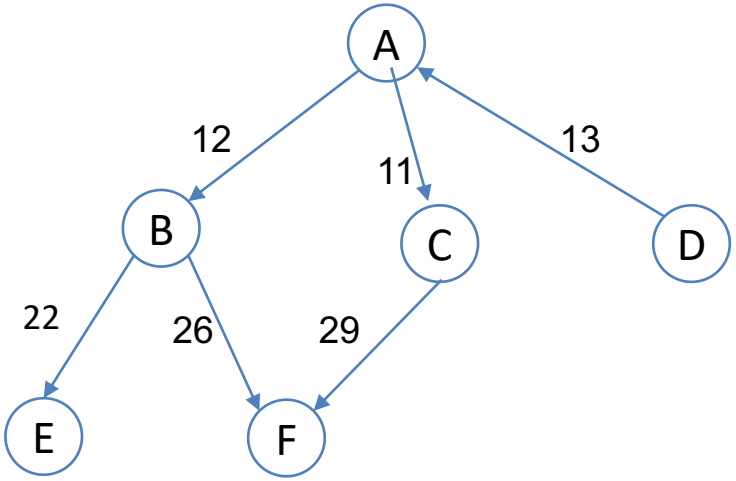
```
"A","B",12
"B","F",26
"B","E",22
"D","A",13
"A","C",11
"C","F",29
```

array indices as vertex/node ids

id	name	adj list
0	"A"	→ 1,12
1	"B"	→ 2,26
2	"F"	

node 0 has a link to node 1 with weight 12
(if the graph is undirected, element (0,12) must be added to the adjacency list of node 1

weighted di-graph representation: Using Adjacency Lists

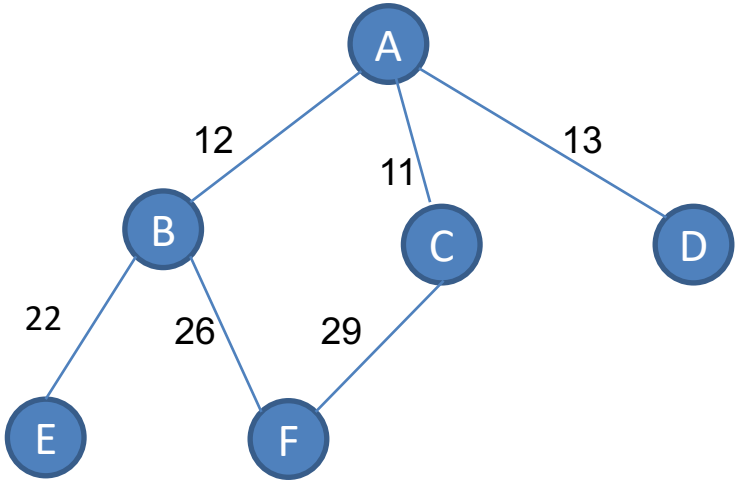


input data

```
"A","B",12
"B","F",26
"B","E",22
"D","A",13
"A","C",11
"C","F",29
```

id	name	adj list
0	"A"	→ 1,12 → 5,11
1	"B"	→ 2,26 → 3,22
2	"F"	
3	"E"	
4	"D"	→ 0,13
5	"C"	→ 2,29

Using Adjacency Lists: what if graph is undirected? unweighted?

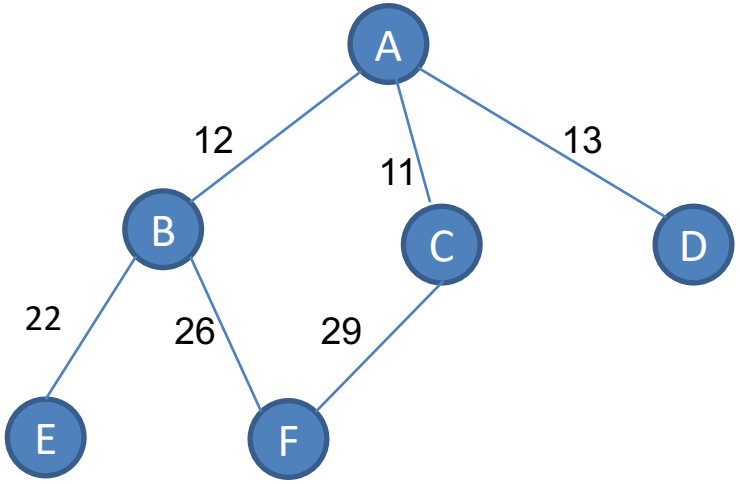


input data

```
"A","B",12
"B","F",26
"B","E",22
"D","A",13
"A","C",11
"C","F",29
```

id	name	adj list
0	"A"	→ 1,12 → 4,13 → 5,11
1	"B"	→ 0,12 → 2,26 → 3,22
2	"F"	→ 1,26 → 5,29
3	"E"	→ 1,22
4	"D"	→ 0,13
5	"C"	→ 0,11 → 2,29

Using Adjacency Lists: Complexity



Complexity of graph algorithms is represented as a function of:

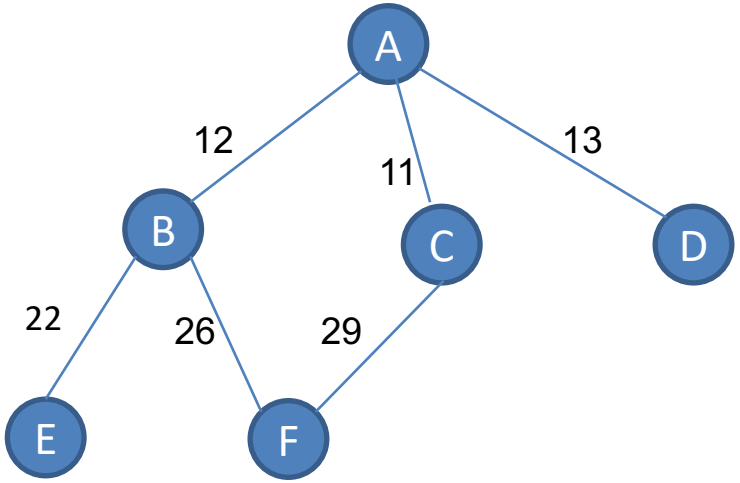
- $|V|$ *(or just* V*): number of vertices*
- $|E|$ *(or just* E*): number of edges*

For adj list representation, what is:

- space complexity?: $O(\ ?\ )$, $\theta(\ ?\ )$
- time complexity of retrieving an edge (for example: is there an edge between C and F)? : $O(\ ?\ )$, $\theta(\ ?\ )$

id	name	adj list
0	"A"	1,12 → 4,13 → 5,11
1	"B"	0,12 → 2,26 → 3,22
2	"F"	1,26 → 5,29
3	"E"	1,22
4	"D"	0,13
5	"C"	0,11 → 2,29

Check your answers: Using Adjacency Lists: Complexity

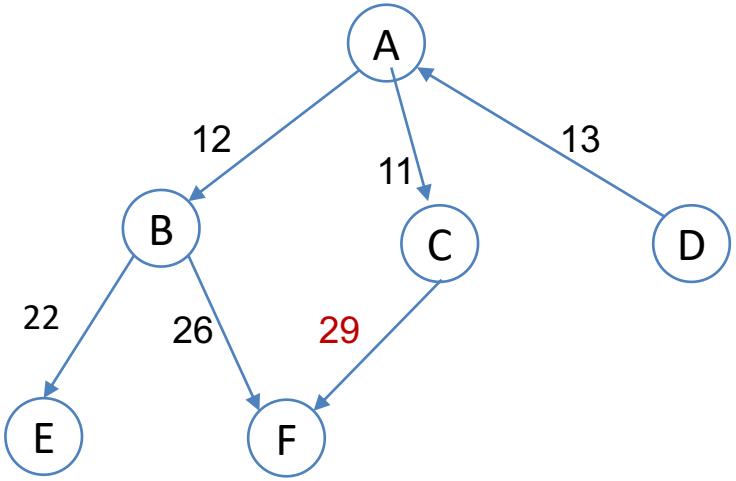


For adj list representation, what is:

- space complexity?: $\theta(V+E)$
- time complexity of retrieving an edge (for example: is there an edge between C and F)? : O(V)

id	name	adj list
0	"A"	→ 1,12 → 4,13 → 5,11
1	"B"	→ 0,12 → 2,26 → 3,22
2	"F"	→ 1,26 → 5,29
3	"E"	→ 1,22
4	"D"	→ 0,13
5	"C"	→ 0,11 → 2,29

weighted di-graph representation: Using Adjacency Matrix



input data

```
"A","B",12
"B","F",26
"B","E",22
"D","A",13
"A","C",11
"C","F",29
```

id	name
0	"A"
1	"B"
2	"F"
3	"E"
4	"D"
5	"C"

A is a matrix of V x V		TO 0	1	2	3	4	5
FROM	0		12				11
	1			26	22		
	2						
	3						
	4	13					
	5				29		

NOTES

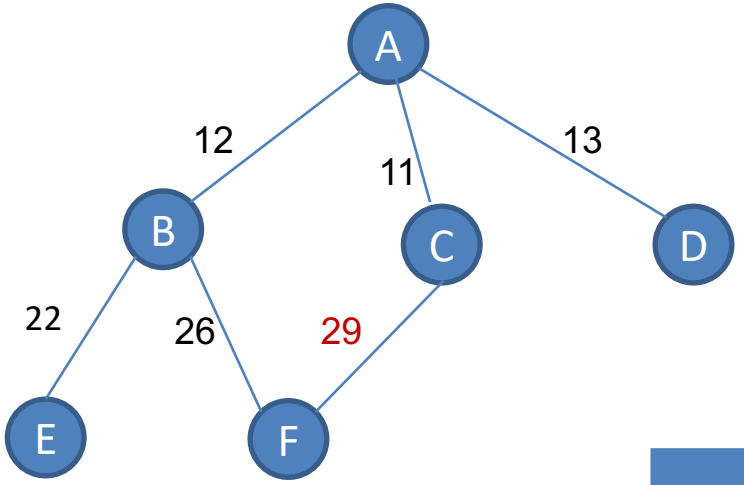
Aij =

- weight of edge i→j
- 1 or 0 if graph is unweighted

For weighted graph, empty cells:

- are "undefined"
- depending on situations, could bet set as 0 or ∞

Using Adjacency Matrix: for undirected graph, the matrix is symmetric



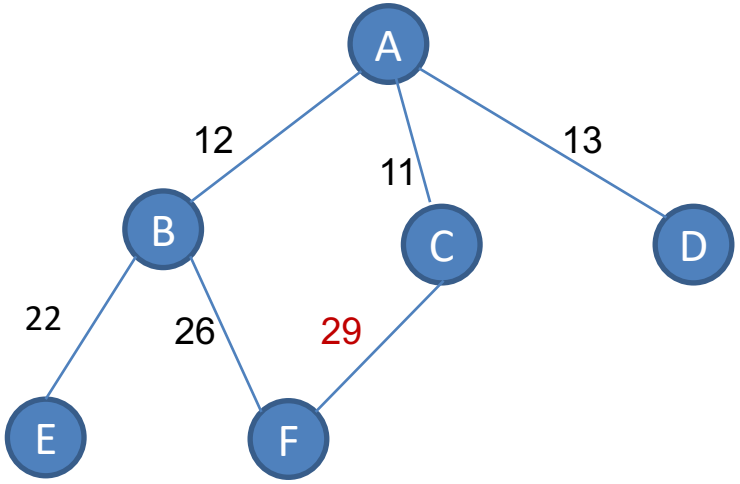
input data

```
"A","B",12
"B","F",26
"B","E",22
"D","A",13
"A","C",11
"C","F",29
```

id	name
0	"A"
1	"B"
2	"F"
3	"E"
4	"D"
5	"C"

		TO					
		0	1	2	3	4	5
FROM	0		12			13	11
	1	12		26	22		
	2		26				
	3		22				29
	4	13					
	5	11			29		

Using Adjacency Matrix - Complexity



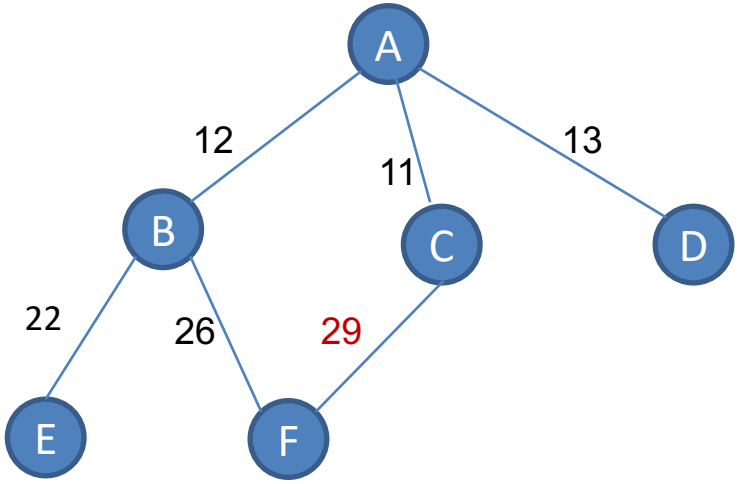
id	name
0	"A"
1	"B"
2	"F"
3	"E"
4	"D"
5	"C"

		TO					
		0	1	2	3	4	5
FROM	0		12			13	11
	1	12		26	22		
	2		26				
	3		22				29
	4	13					
	5	11			29		

For adj matrix representation, what is:

- space complexity?:
  $O(?), \theta(?)$
- time complexity of retrieving an edge (for example: is there an edge between C and F)? :
  $O(?), \theta(?)$

Check your answer: Using Adjacency Matrix - Complexity



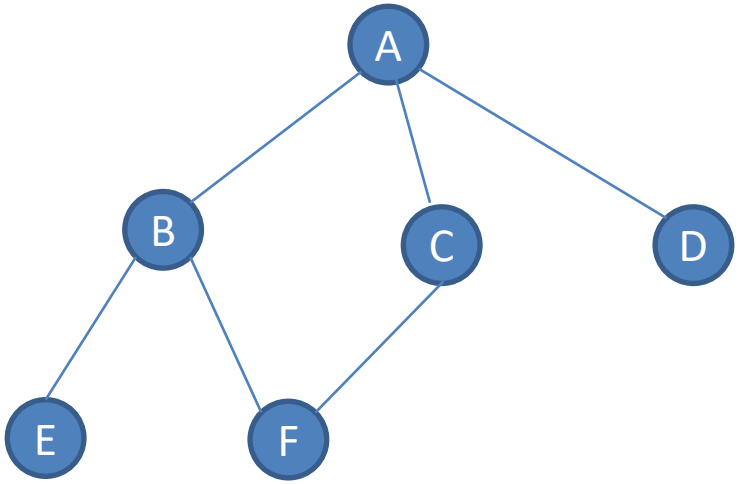
id	name
0	"A"
1	"B"
2	"F"
3	"E"
4	"D"
5	"C"

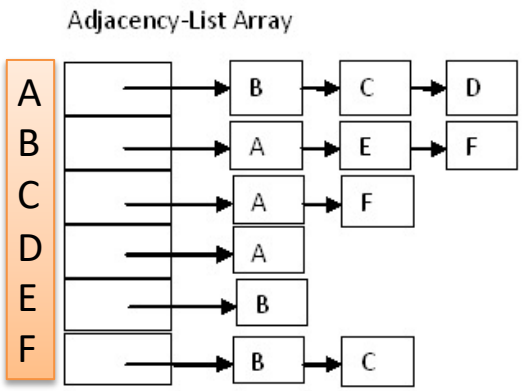
		TO					
		0	1	2	3	4	5
FROM	0		12			13	11
	1	12		26	22		
	2		26				
	3		22				29
	4	13					
	5	11			29		

For adj matrix representation, what is:

- space complexity?:
  $\theta(V^2)$
- time complexity of retrieving an edge (for example: is there an edge between C and F)? :
  $O(1)$

Example for unweighted, undirected graph





	A	B	C	D	E	F
A	0	1	1	1	0	0
B	1	0	0	0	1	1
C	1	0	0	0	0	1
D	1	0	0	0	0	0
E	0	1	0	0	0	0
F	0	1	1	0	0	0

Graphs: Representation - sparse & dense graphs

What is a suitable representation method for:

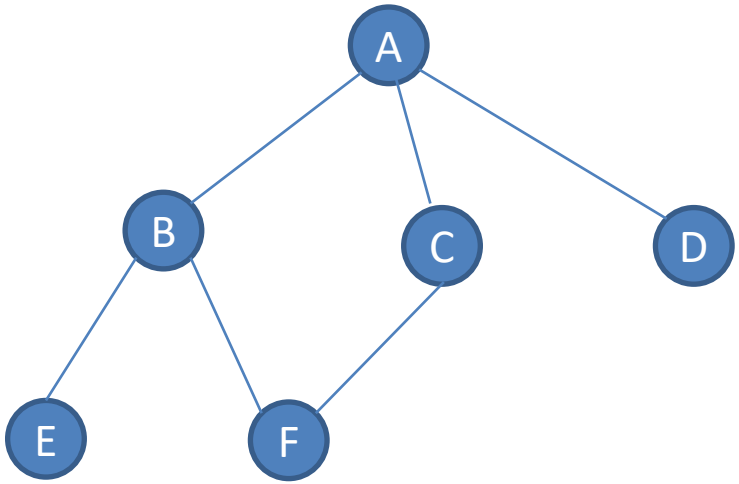
- a graph of this on-line class, where nodes represent students, edge (a,b) means “a knows b”,
- a graph of this would-be-face-to-face class, where nodes represent students, edge (a,b) means “a knows b”
- the webgraph?

Is each graph dense/sparse?

What is the criteria for:

- *Sparse graphs:*
- *Dense graphs:*

Graph Traversal = What? How?

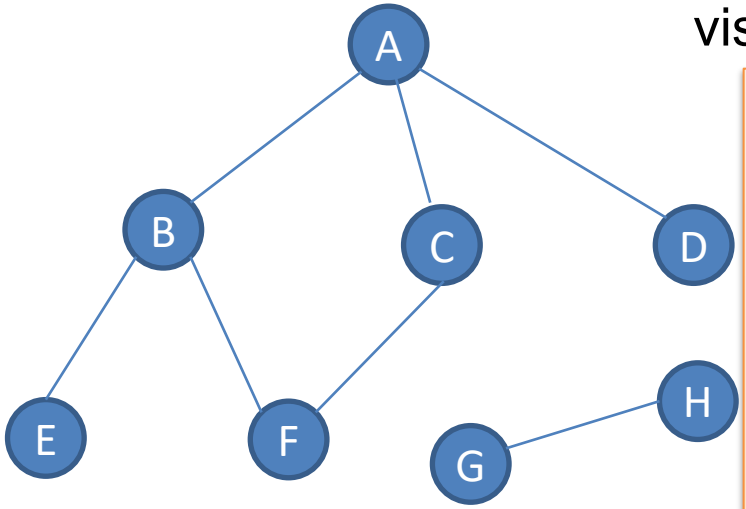


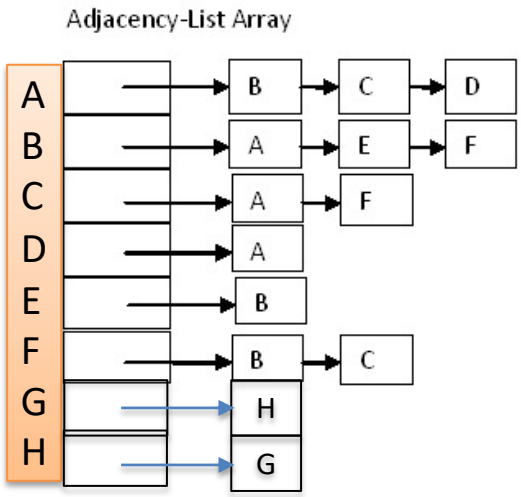
Traversal: visit nodes of a graph in a systematic way in order to perform some search. For example:

- find a path from A to F

Graph Traversal = Graph Search

Graph Traversal = How?



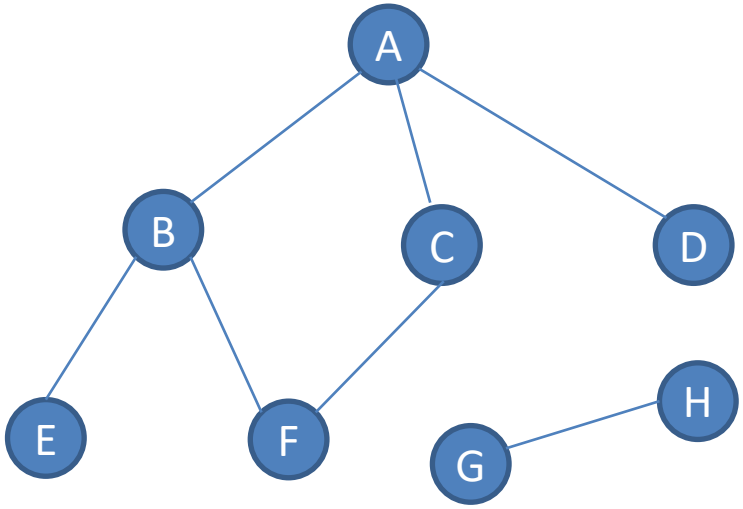


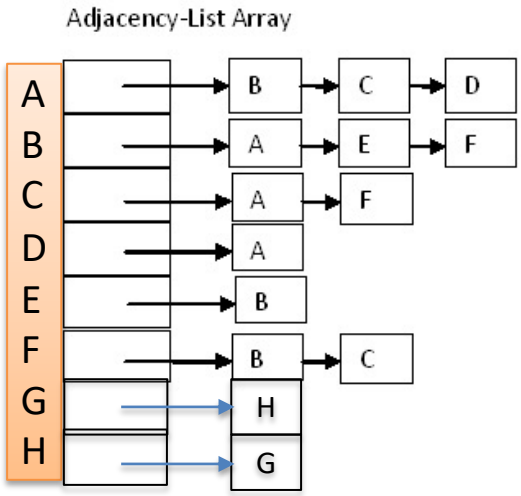
visit each node exactly once, in a systematic way

```
//mark all nodes as "unvisited":
visited[u]= 0 for each u= 0..|V|-1
order= 0;
for each node u {
  if (!visited(u)) visit(u);
}

function visit(int u) {
  S= empty data structure
  insert u into S
  while (S not empty) {
    u = remove from S
    if (!visited[u]) {
      visited[i]= ++order;
      visit u;
      insert unvisited neighbours of u to S
    }
  }
}
```

Q: using the algorithm, how to find out the number of components in the graph?

Breath-First Search = level-by-level using queues





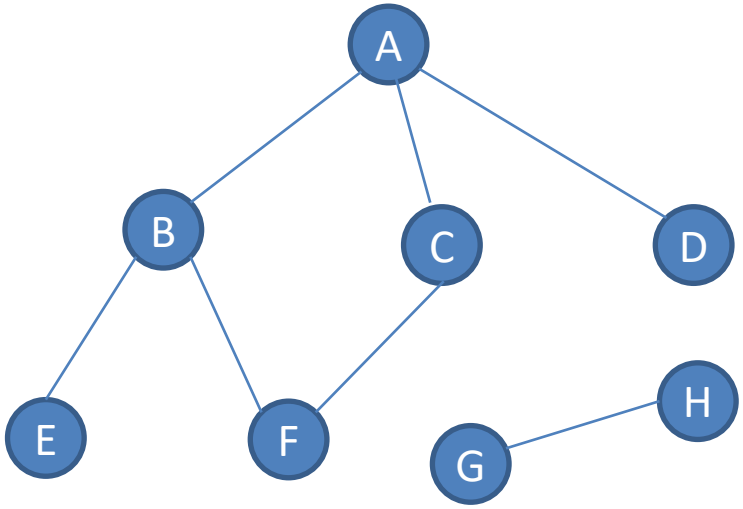

```
//mark all nodes as "unvisited":
visited[u]= 0 for each u= 0..|V|-1
order= 0;
for each node u {
  if (!visited(u)) visit(u);
}

function visit(int u) {
  Q= empty queue
  enQ(u, Q)
  while (Q not empty) {
    u = deQ(Q)
    if (!visited[u]) {
       visited[i]= ++order;
       visit u;
       for each (u,v)
          if (!visited[v]) enQ(v, Q)
    }
  }
}
```

Q: List the nodes in order of visited by BFS. *On tie choose the smallest element.*

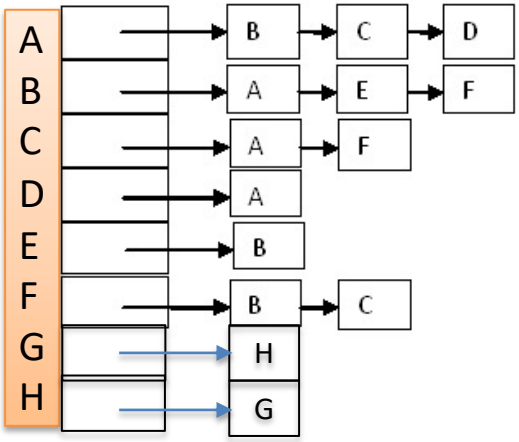
?

Complexity= O(?) θ(?)

Breath-First Search = level-by-level using queues



Adjacency-List Array




```
function visit(int u) {
  Q= empty queue
  enQ(u, Q)
  while (Q not empty) {
    u = deQ(Q)
    if (!visited[u]) {
       visited[i]= ++order;
       visit u;
       for each (u,v)
           if (!visited[v]) enQ(v, Q)
    }
  }
}
```

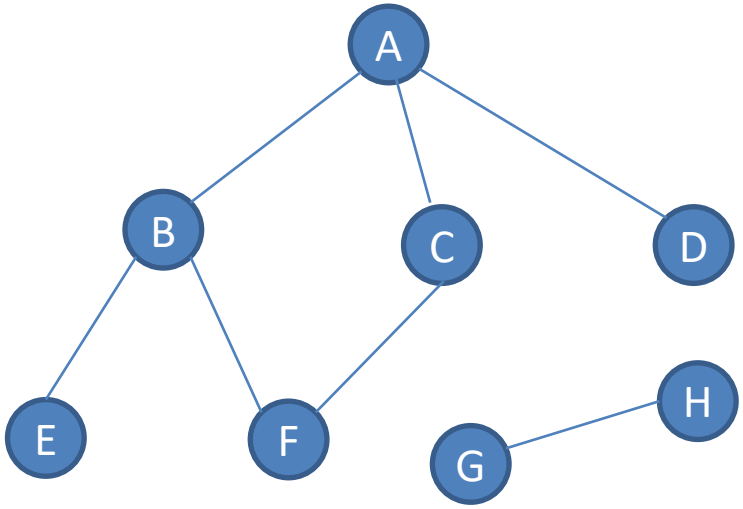
Q: List the nodes in order of visited by BFS. *On tie choose the smallest element.*

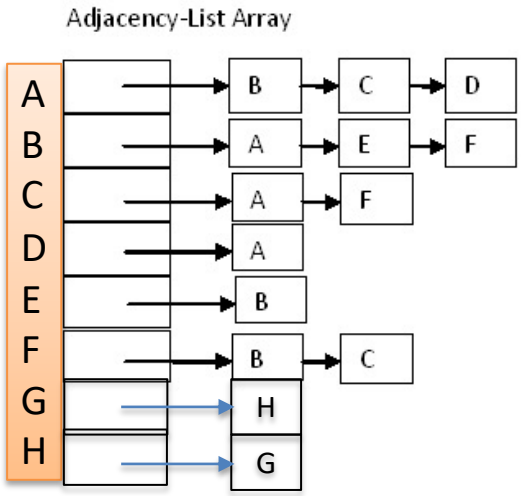
?

Complexity=

- using Adj Lists: O(?) θ(?)
- using Adj matrix: O(?) θ(?)
- what if graph is sparse/dense?

Breath-First Search = level-by-level using queues





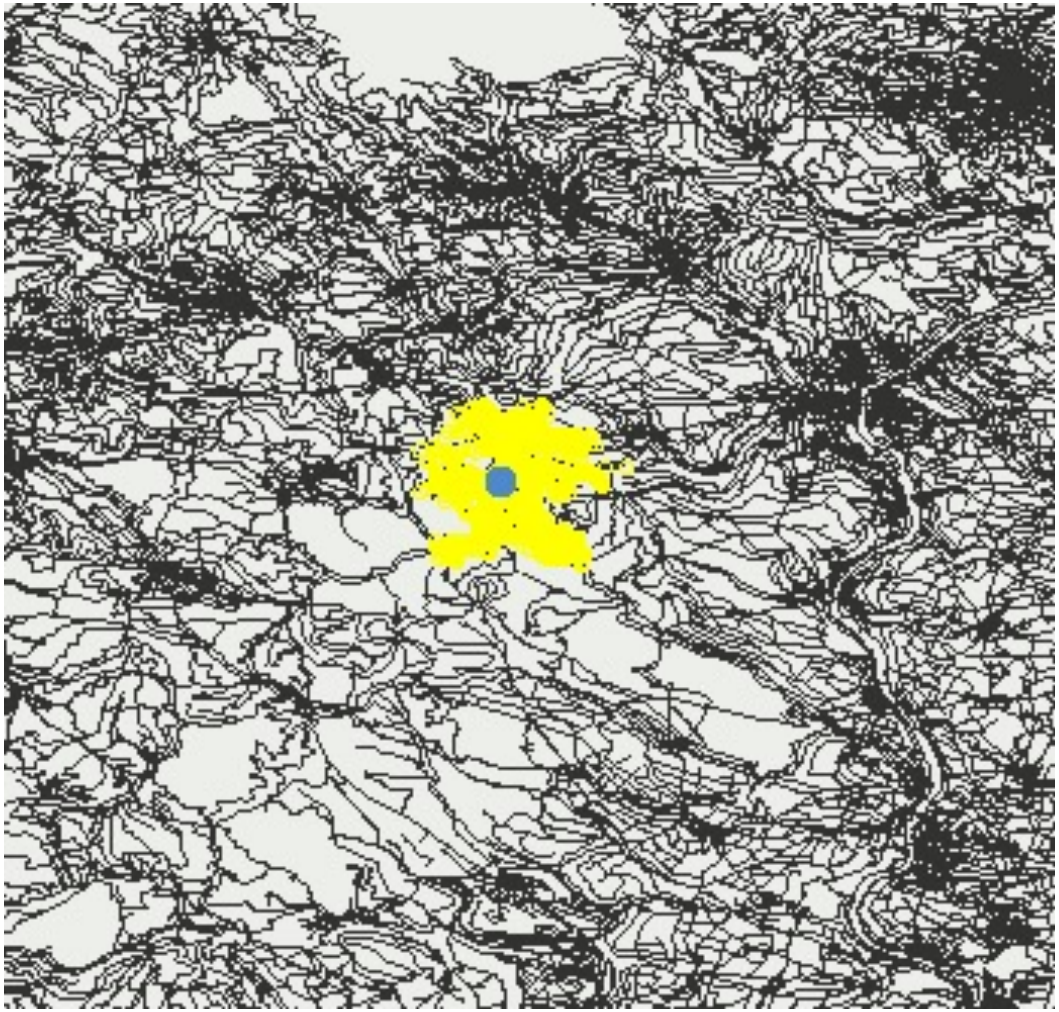

```
function visit(int u) {
  Q= empty queue
  enQ(u, Q)
  while (Q not empty) {
    u = deQ(Q)
    if (!visited[u]) {
       visited[i]= ++order;
       visit u;
       for each (u,v)
          if (!visited[v]) enQ(v, Q)
    }
  }
}
```

Complexity=

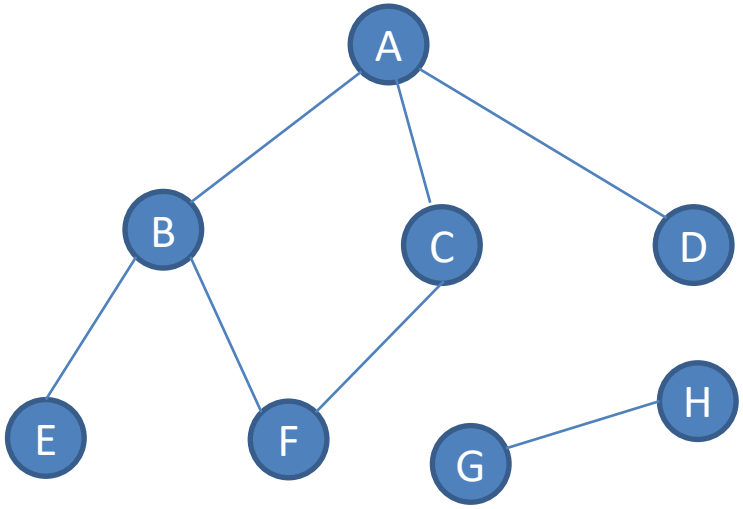
- using Adj Lists: $\theta(V+E)$
- using Adj matrix: $\theta(\ V^2\ )$
- what if graph is sparse/dense?

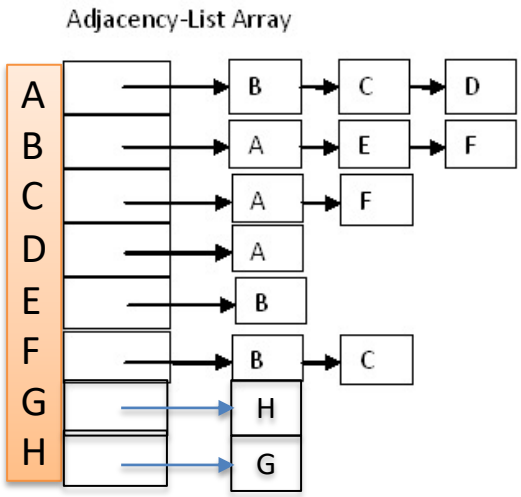
	adj list	adj matrix
sparse	$\theta(\ \ )$	$\theta(\ \ )$
dense	$\theta(\ \ )$	$\theta(\ \ )$

for shortest path: use Breath-first Search (BFS)



Depth-First Search = using stackss





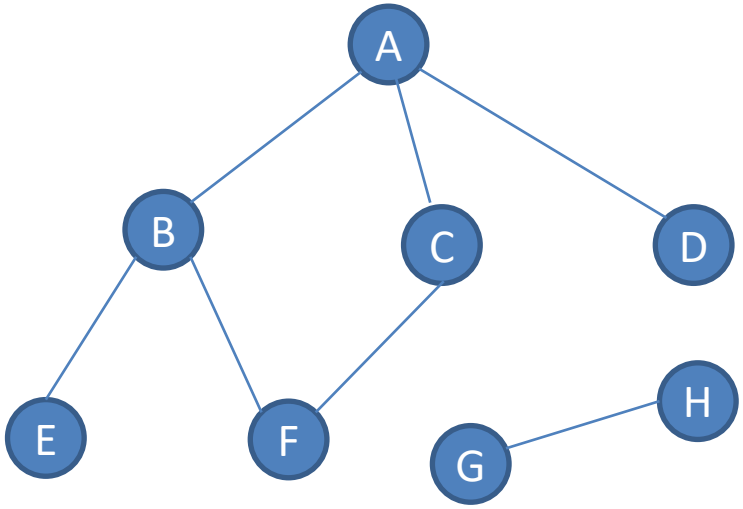

```
//mark all nodes as "unvisited":
visited[u]= 0 for each u= 0..|V|-1
order= 0;
for each node u {
  if (!visited(u)) visit(u);
}

function visit(int u) {
  S= empty stack
  push(u, S)
  while (S not empty) {
    u = pop(S)
    if (!visited[u]) {
       visited[i]= ++order;
       visit u;
       for each (u,v)
          if (!visited[v]) push(v, S)
    }
  }
}
```

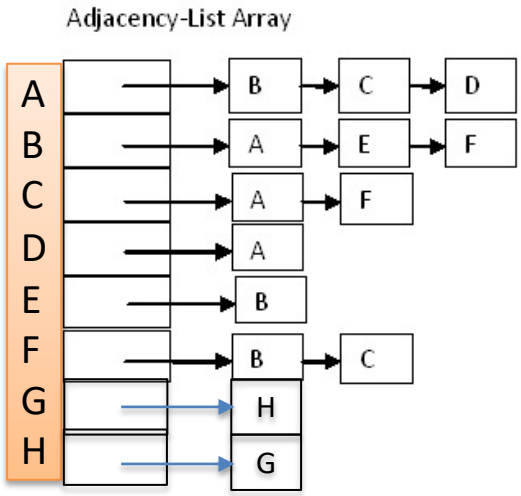
NOTE: instead of using stack, function `visit` is normally implemented as a recursive function.

Q: List the nodes in order of visited by DFS.

?

Depth-First Search = using stackss






```
function visit(int u) {
  S= empty stack
  push(u, S)
  while (S not empty) {
    u = pop(S)
    if (!visited[u]) {
       visited[i]= ++order;
       visit u;
       for each (u,v)
          if (!visited[v]) push(v, S)
    }
  }
}
```

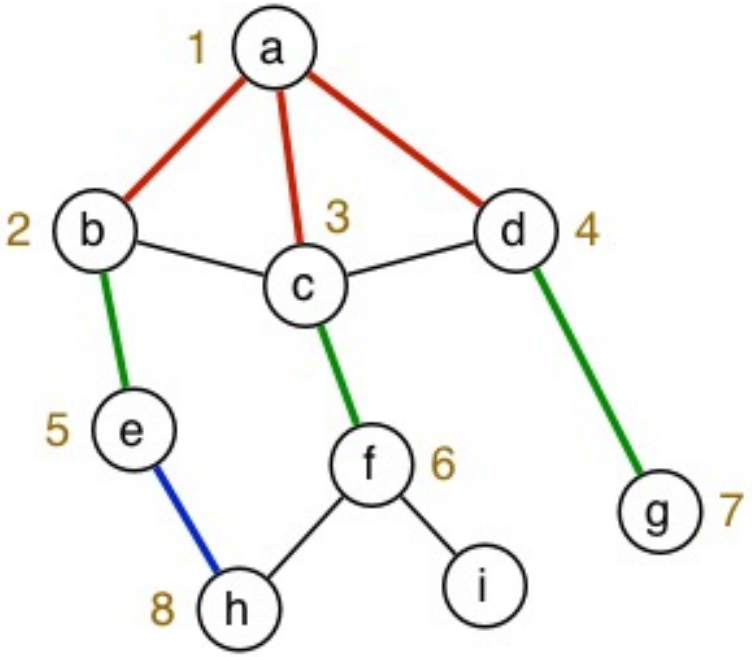
Q: List the nodes in order of visited by DFS.

? A B E F C D G H

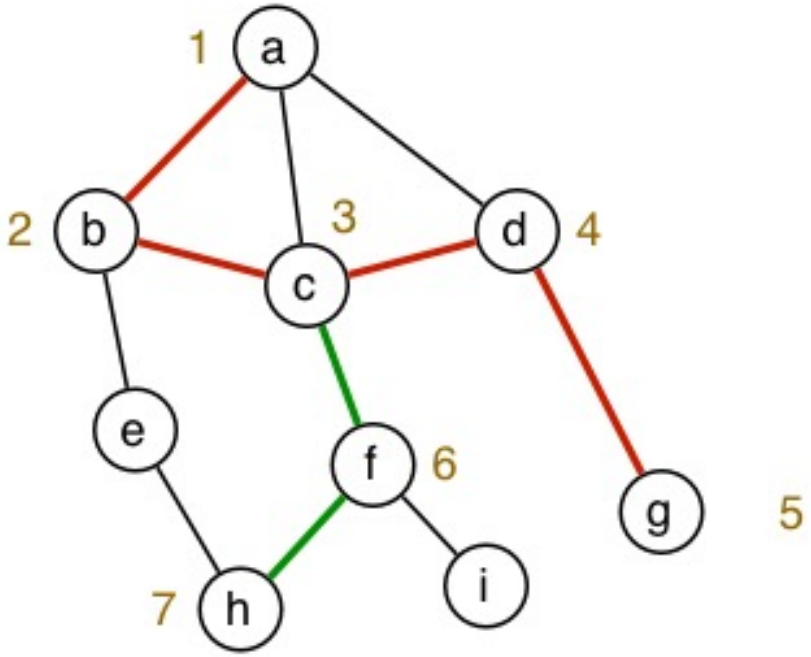
Complexity=

- using Adj Lists: $\theta(V+E)$
- using Adj matrix: $\theta(V^2)$
- what if graph is sparse/dense?

Graphs: DFS & BFS

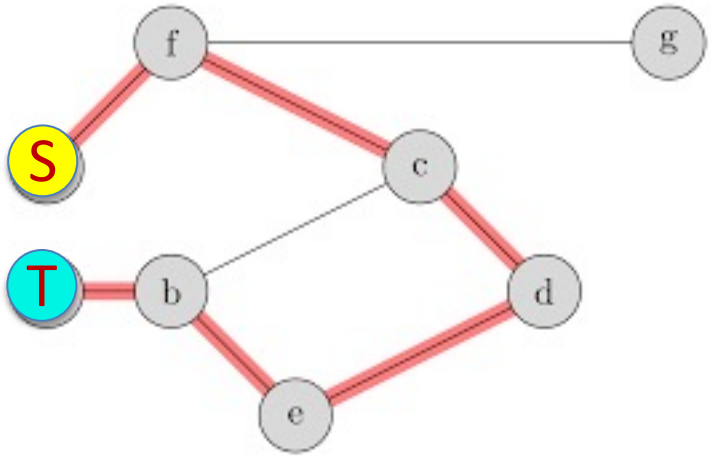


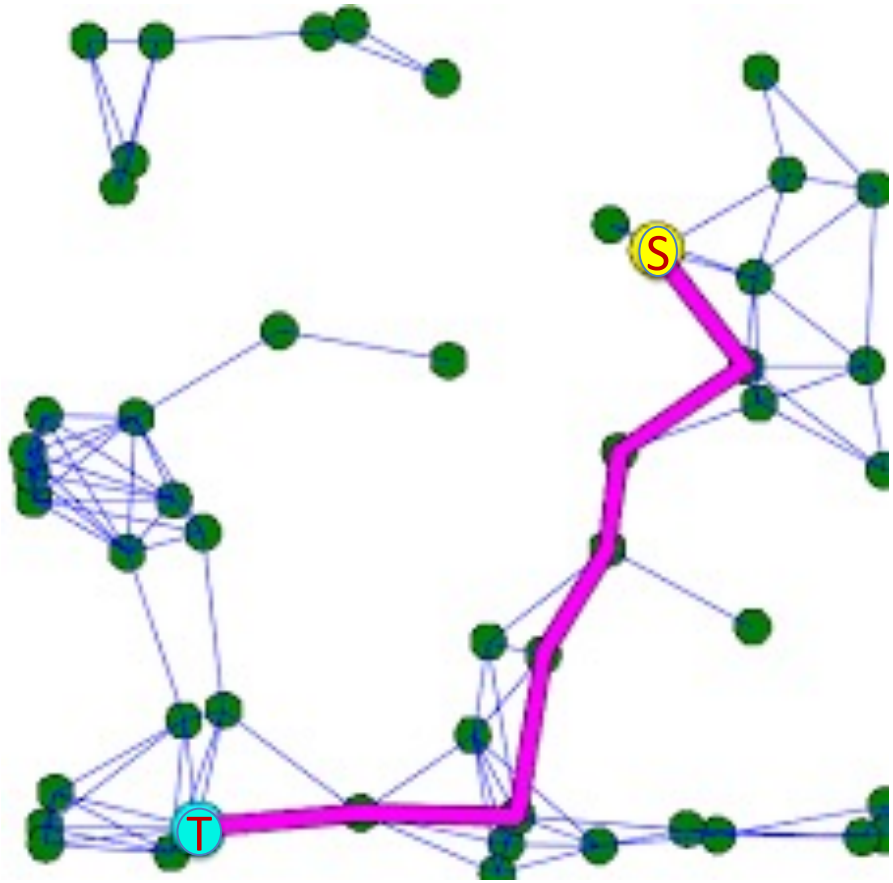
Breadth-first Search

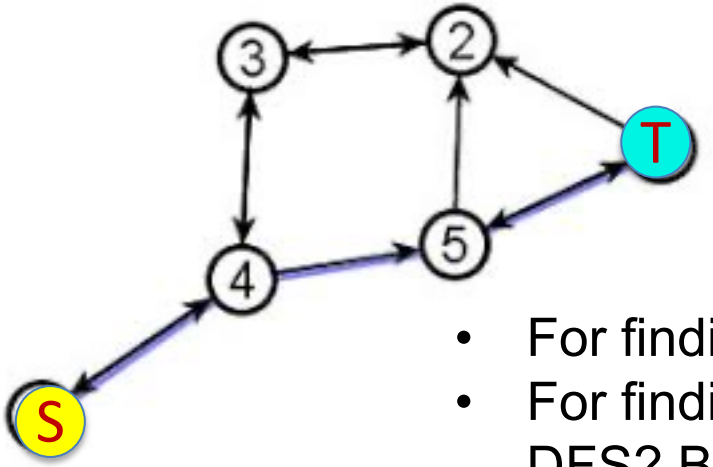


Depth-first Search

Paths in unweighted graphs: path length, shortest path

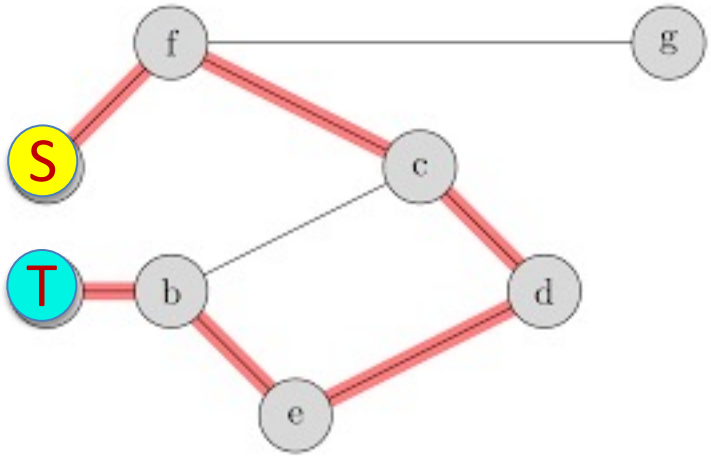


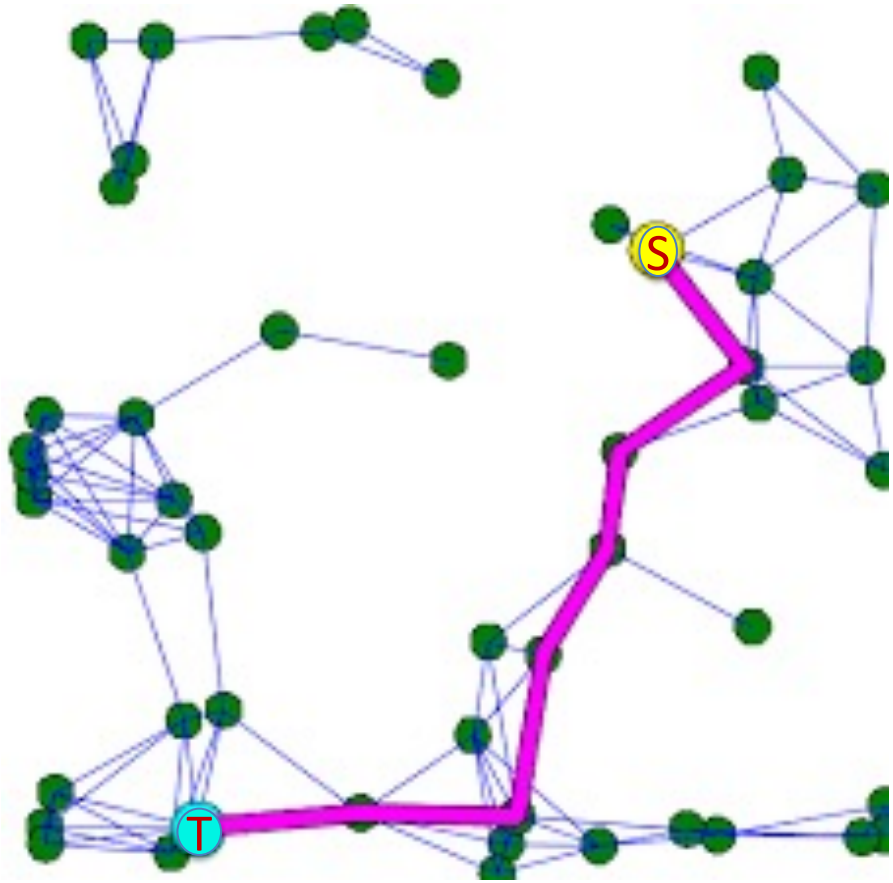


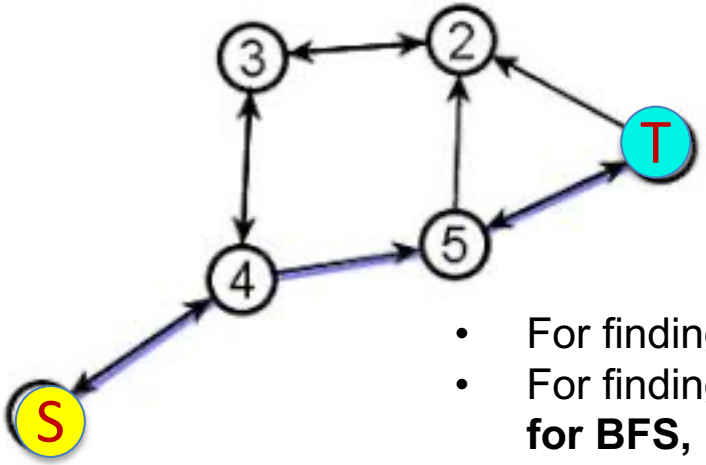


- For finding a path from S to T can we use DFS? BFS?
- For finding a shortest path from S to T can we use DFS? BFS?
- Applied to directed graphs? cyclic graphs?

Paths in unweighted graphs: path length, shortest path



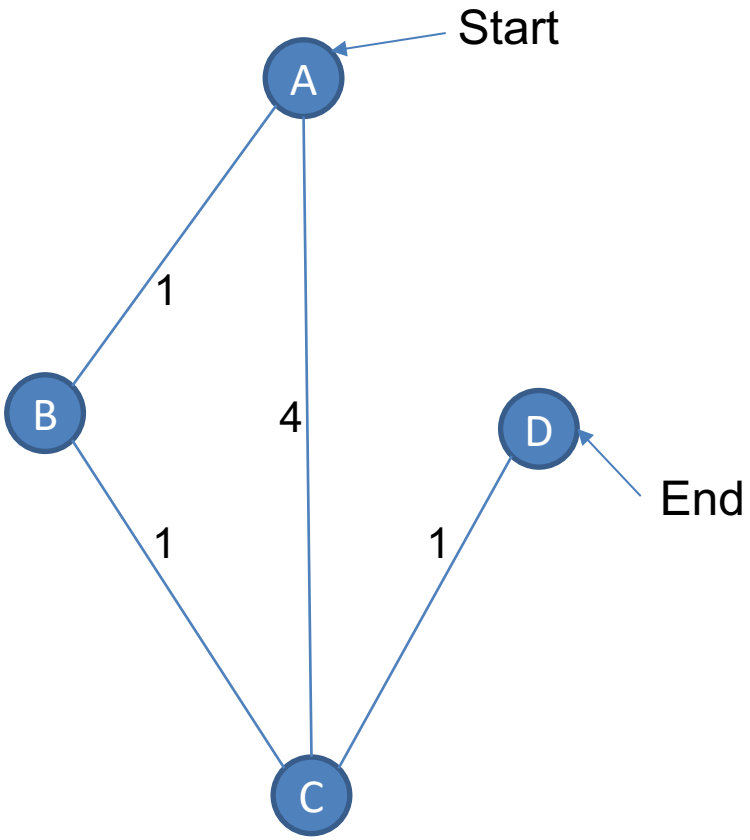




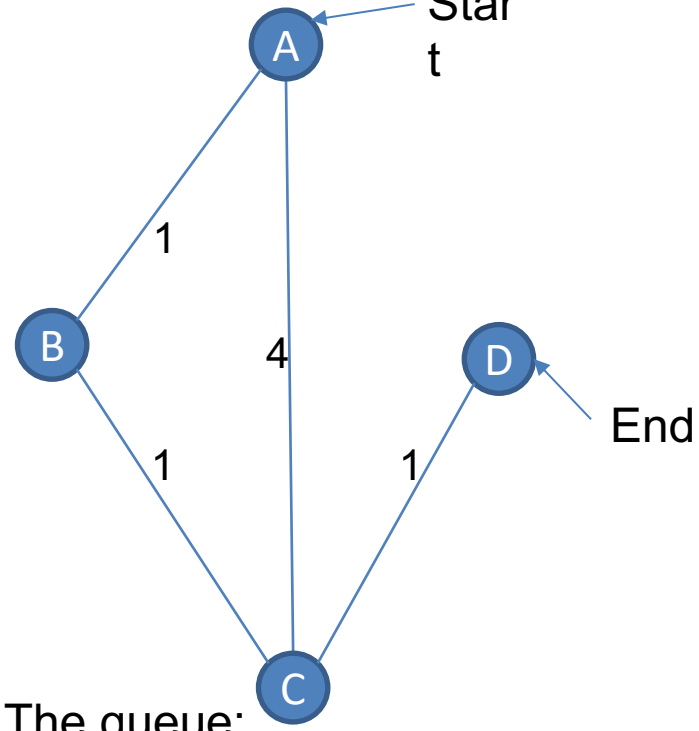
- For finding a path from S to T can we use DFS? BFS? **Yes to both.**
- For finding a shortest path from S to T can we use DFS? BFS? **Yes for BFS, No for DFS.**
- Applied to directed graphs? cyclic graphs? **Yes, Yes.**

Exercise: Using BFS to find shortest paths in weighted graphs?

How to use BFS for finding shortest paths in weighted graphs, supposing weight are positive integers? What's the shortest path from A to D?

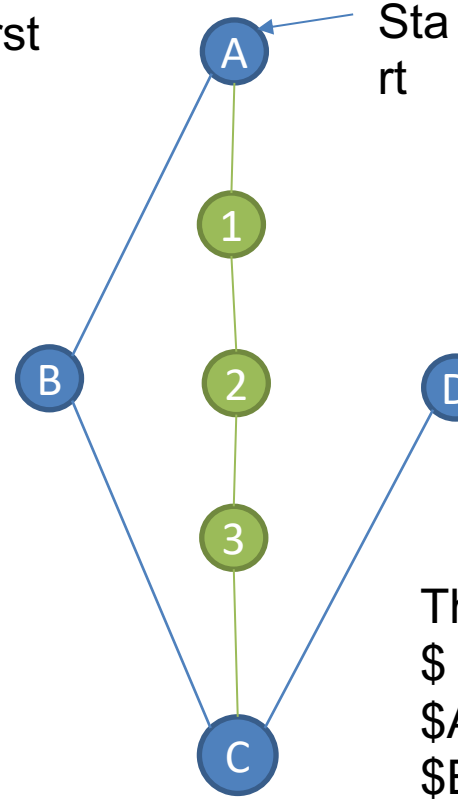


Exercise…



The queue:
$
$A
$BC
$C
$D
$

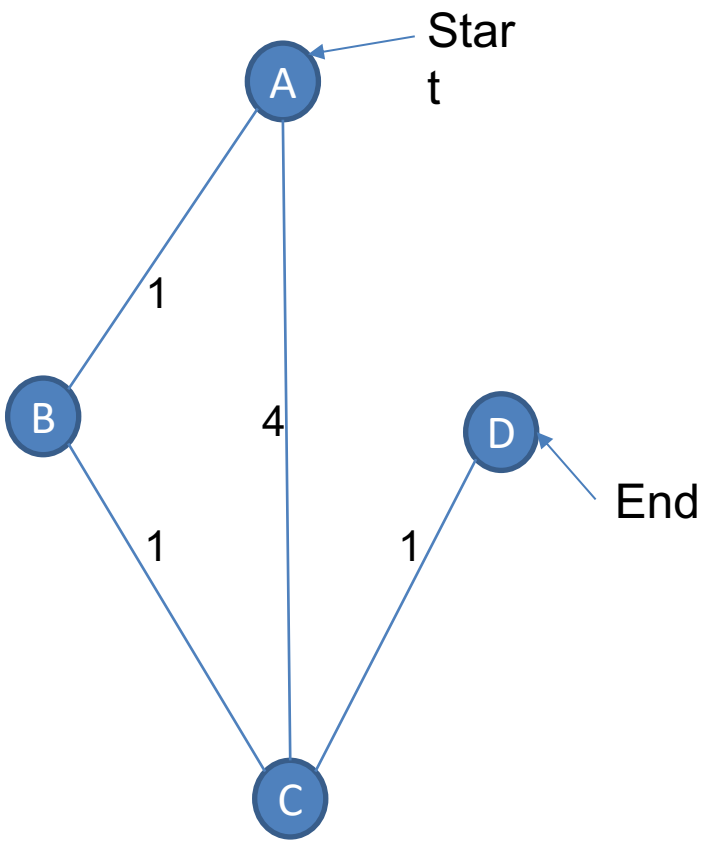
Adapting Breadth-First Search to work on weighted graph



The queue:
$
$A
$B1
$1C
$C2
$2D3
$D3
$3

Is there a better way?

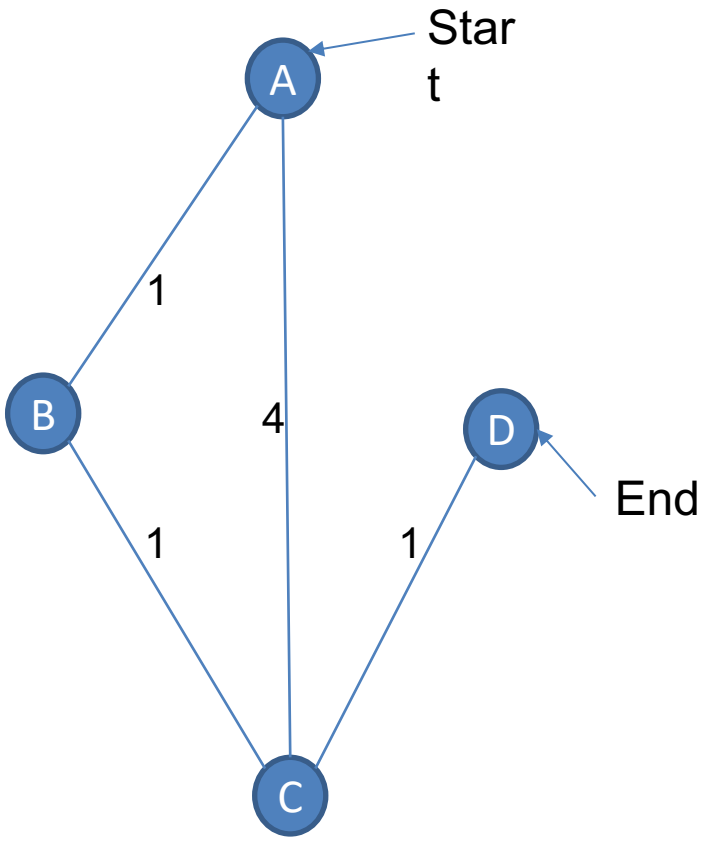
Finding shortest paths using priority queues



content of PQ
$
$ (A,0)
$ (B,1), (C,4)

insert A with distance A→A = 0

Finding shortest paths using priority queues



content of PQ
$
$ (A,0)

$ (B,1), (C,4)

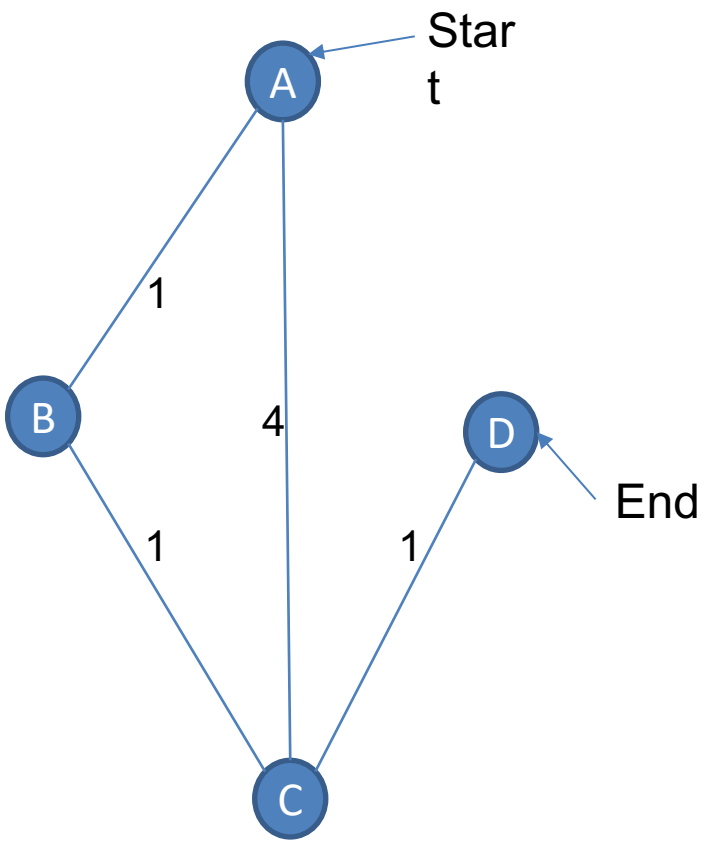
$ (C,2)

insert A with distance A→A = 0
done with A, path A→A = 0

done with B, path A→B = 1
would insert C with distance=
dist(A,B) + dist(B,C) =2
But …
we should replace (C,4) with (C,2)

…

Finding shortest paths using priority queues



content of PQ

$ (A,0),(B,∞), (C,∞), (D,∞)

populate queue with all nodes
then remove min-dist A
Done= { (A,0) }
and update distance for the neighbours of A

$ (B,1), (C,4), (D,∞)

remove B that has min dist
Done= {(A,0), (B,1)}
update…

$ (C,2), (D,∞)

Done= {(A,0}, {B,1}, (C,2)}

$ (D,3)

Done= {(A,0}, {B,1}, (C,2),(D,3)}

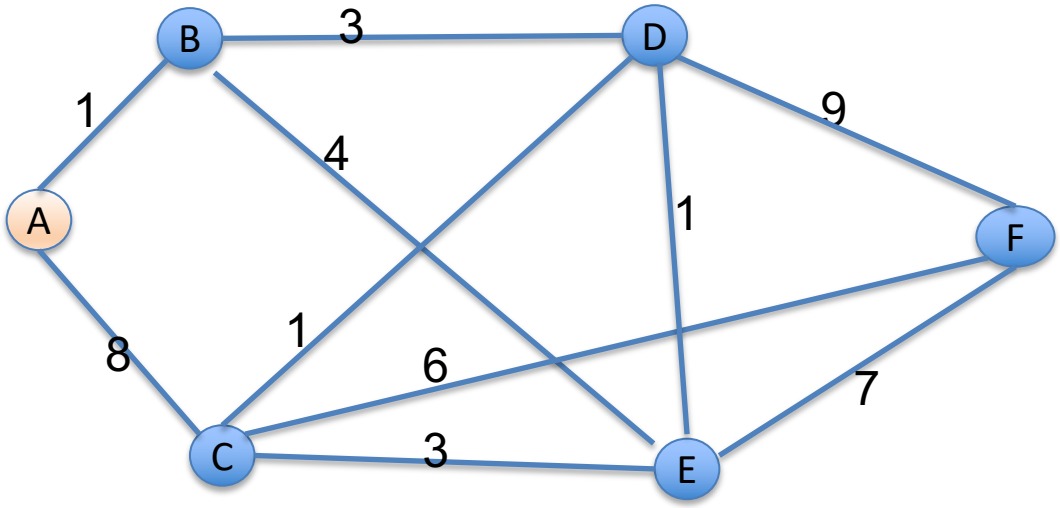
$

So the shortest path A→D has length 3, but what is the path?

Dijkstra's Algorithm: Single Source Shortest Path SSSP

The task:

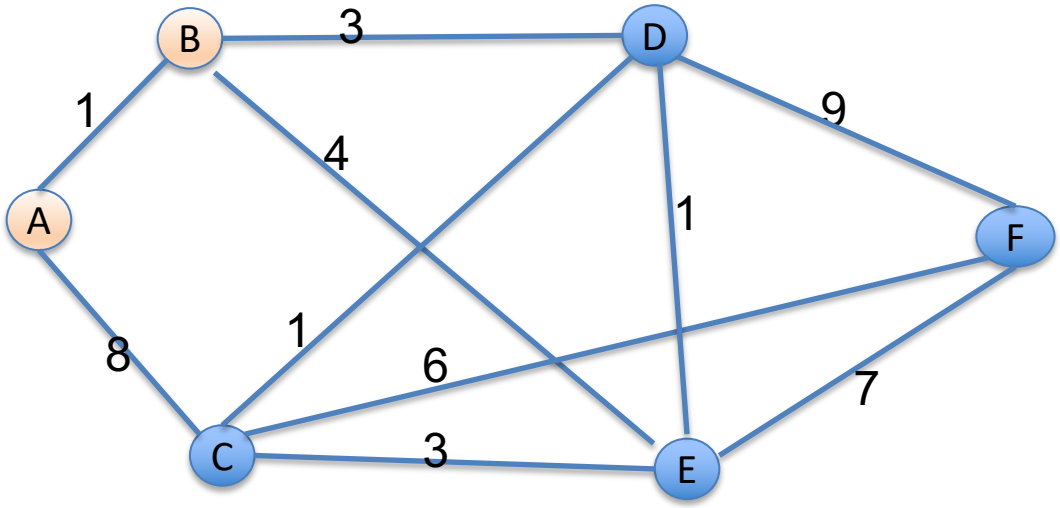
- Given a weighted graph $G=(V,E,w(E))$, and $s \in V$, and supposing that *all weights are positive*.
- Find shortest path (path with min total weight) from s to all other vertices.



Find a shortest path:

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

We know shortest path A->A costs 0: dist[A]= 0, and dist[*]=∞
Use A to update: dist[B]= 1, dist(C)= 8 dist={A:0, B:1, C:8, *: ∞}
done with A

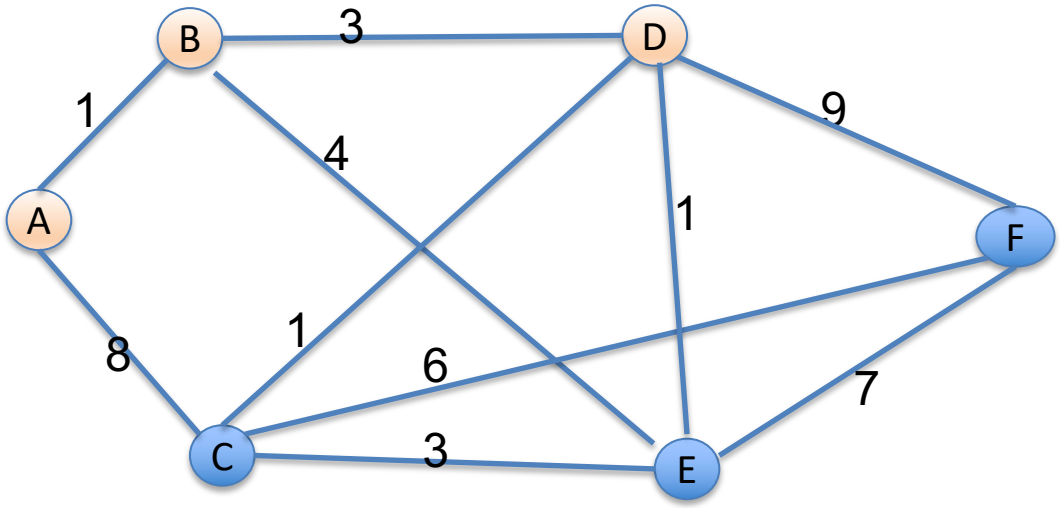


Find a shortest path:

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

We know shortest path A->A costs 0: dist[A]= 0, and dist[*]=∞
Use A to update: dist[B]= 1, dist(C)= 8 dist={A:0, B:1, C:8, *: ∞}
done with A

Shortest path A->B should cost 1, why?
Use B to update: dist={A:0, B:1, C:8, D:, E:, *: ∞}



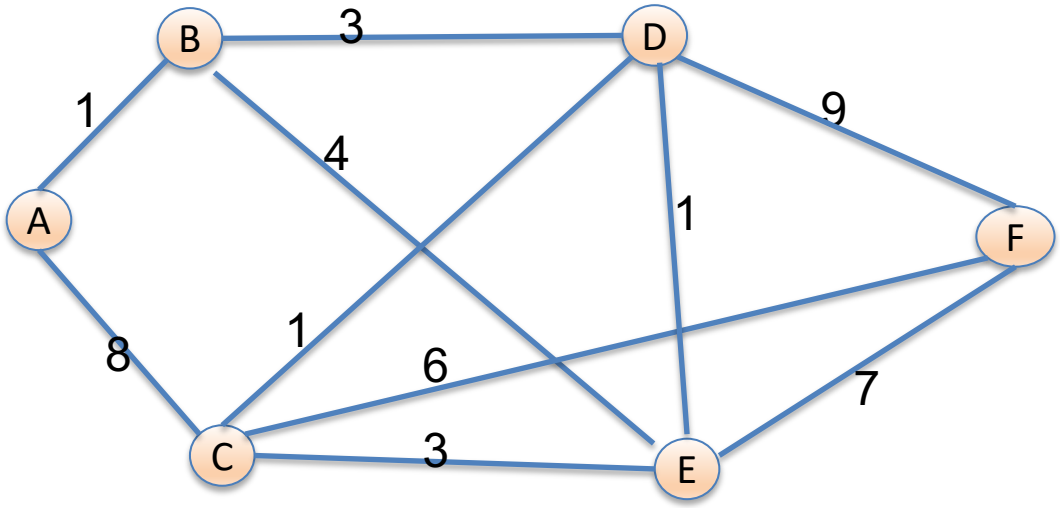
Find a shortest path:

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

We know shortest path A->A costs 0: dist[A]= 0, and dist[*]=∞
Use A to update: dist[B]= 1, dist(C)= 8 dist={A:0, B:1, C:8, *: ∞}
done with A

Shortest path A->B should cost 1, why?
Use B to update: dist={A:0, B:1, C:8, D:4, E:5, *: ∞}
Shortest path A->D should cost 4, why?

Use D to update: dist={A:0, B:1, D:4, C: 8 vs 4+1, E:5 vs 4+1, F: 13}
dist={A:0, B:1, D:4, C: 5, E:5, F: 13}



Find a shortest path:

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

We know shortest path A->A costs 0: dist[A]= 0, and dist[*]=∞
Use A to update: dist[B]= 1, dist(C)= 8 dist={A:0, B:1, C:8, *: ∞}
done with A

Shortest path A->B should cost 1, why?
Use B to update: dist={A:0, B:1, C:8, D:4, E:5, *: ∞}
Shortest path A->D should cost 4, why?

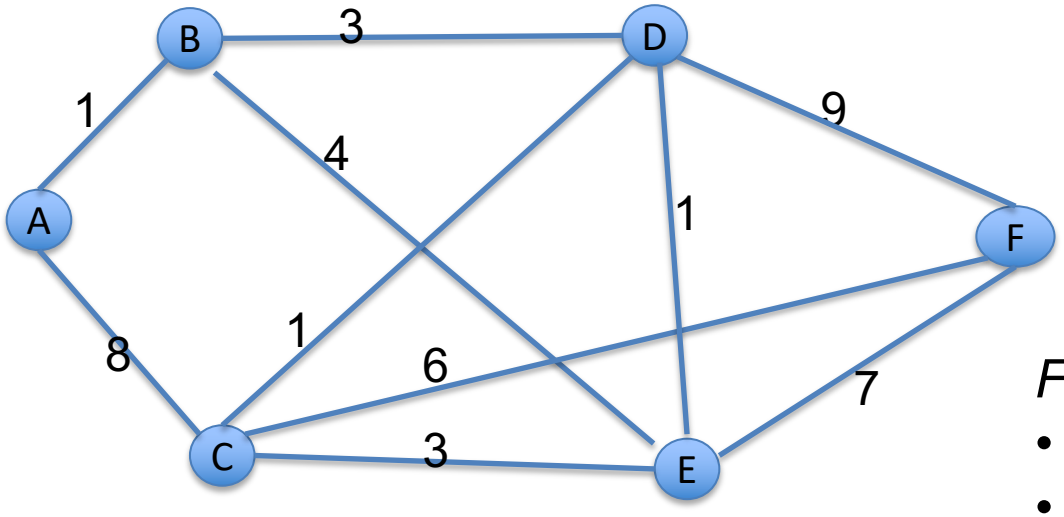
Use D to update: dist={A:0, B:1, D:4, C: 8 vs 4+1, E:5 vs 4+1, F: 13}
dist={A:0, B:1, D:4, C: 5, E:5, F: 13}
Continue:
dist={A:0, B:1, D:4, C: 5, E:5, F: 11}
dist={A:0, B:1, D:4, C: 5, E:5, F: 11}
dist={A:0, B:1, D:4, C: 5, E:5, F: 11} *What's the shortest path from A to C?*

Dijkstra's algorithm [conceptual only]

```
set dist[u]= ∞, pred[u]=nil for all u
set dist[s]= 0
set PQ= makePQ(V)
while (PQ not empty)
    u= deleteMin(PQ)
    visit  u                  // practice: just mark u as visited
    for all (u,v) in G:
        if (dist[v] > dist[u]+w(u,v)):
            update dist[v] and pred[v]
```

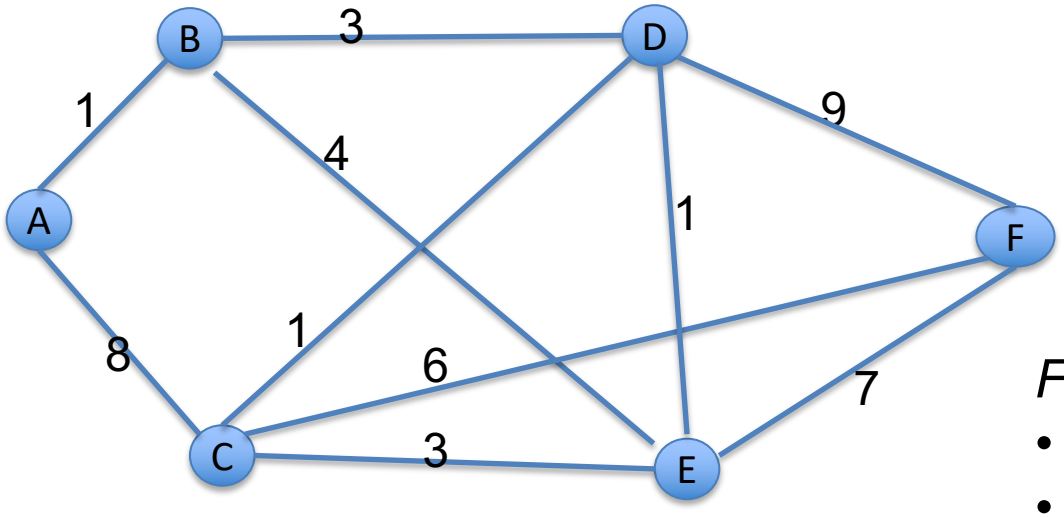
Programming note: "update `dist[v]` and `pred[v]`" means

1. `dist[v]= dist[u]+w(u,v),  pred[v]= u`
2. decrease weight of `v` in `PQ` to `dist[v]`, hence, need to locate `v` in `PQ`, change weight and `upheap`. There is a way to do that operation in `O(log n)` [not for this course]



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:

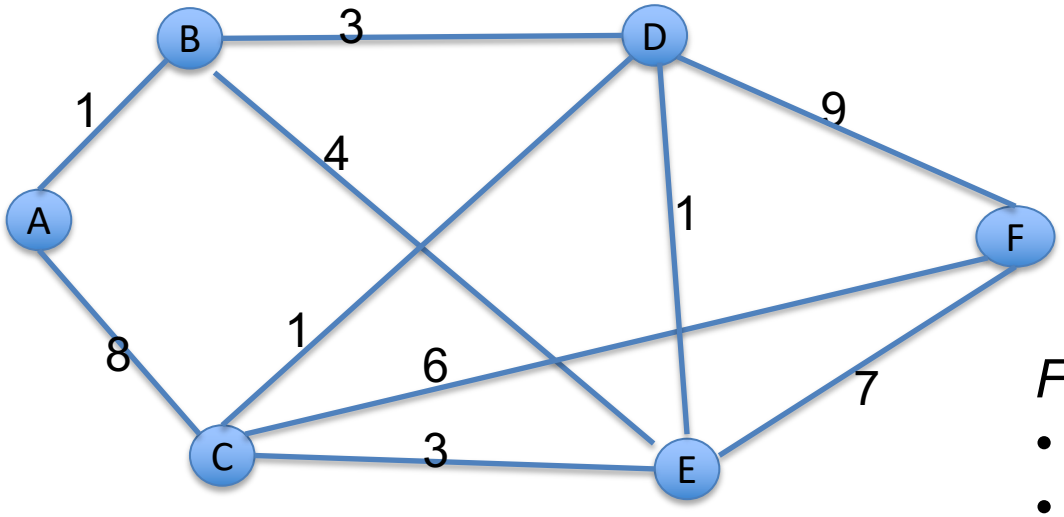
- From A to B
- From A to C
- From A to F
- From A to any other node

min heap	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

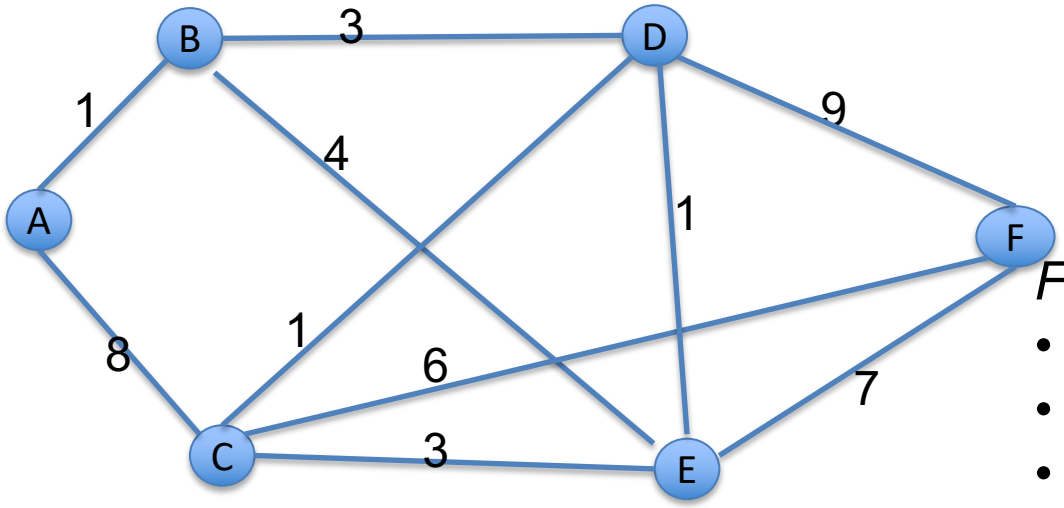
prev[D]: node that precedes D in the path A→D



Find a shortest path:

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

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



Find a shortest path:

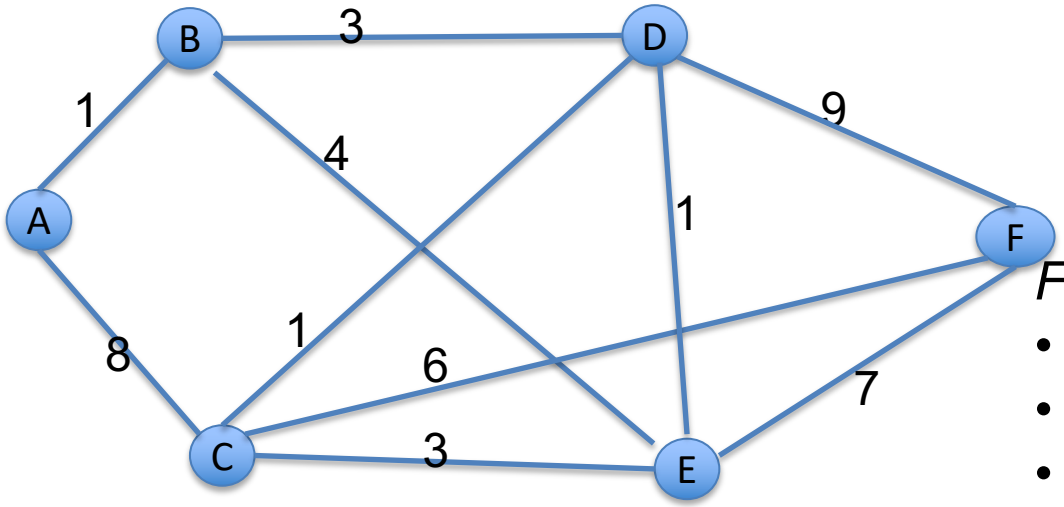
- From A to B
- From A to C
- From A to F
- SP A->F=

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 ∞

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**
C						

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	∞,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**
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

Exercise or/and Lab Work

Data	For a directed graph with the edges listed in LHS:
a b 3 **a d 7** **b d 2** **c e 6** **d b 2** **d c 5** **d e 4** **e d 2**	1. Draw a weighted directed graph that reflects these edges and weights (logical representation). 2. Construct an adjacency matrix for the weighted digraph you have just drawn, including the weights. Be explicit about how you are going to handle matrix cells for which there is no information in the data. 3. Run through Dijkstra's Algorithm starting from the vertex a.

Lab: Peer Programming Exercises

In breakout rooms:

- review Question 10.1, make sure you can do/redo it
- do Programming 9.1 (read: Programming 10.1)

Notes on Programming 10.1:

- You need to fill in for dijkstra.c
- PQ was implemented for you, but using unsorted array with O(n) complexity for deQ