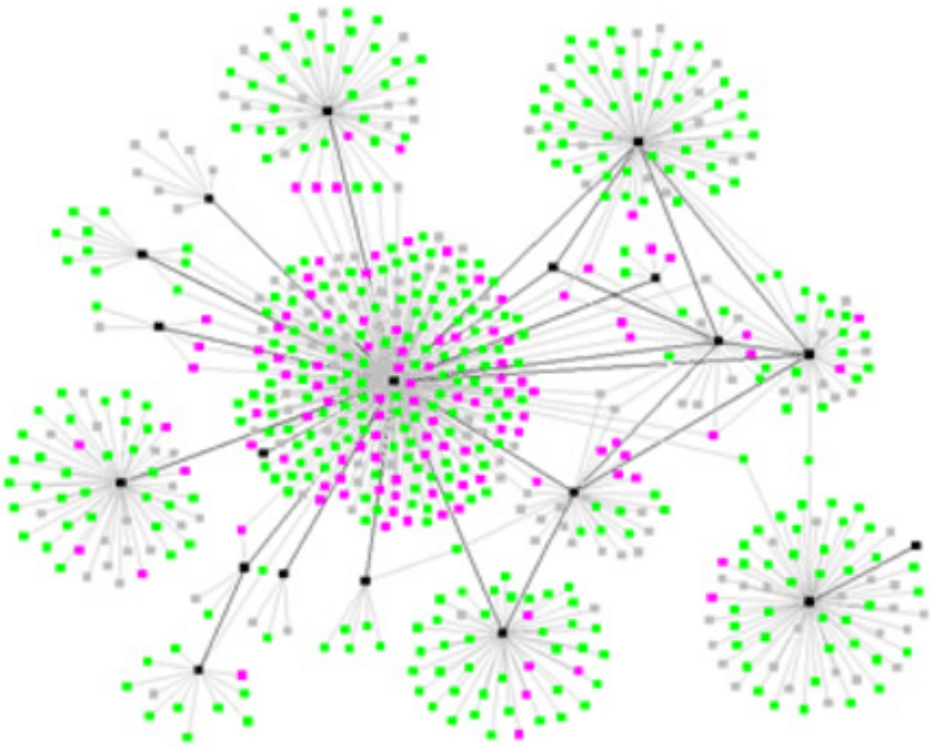
COMP20003 Workshop Week 10

1	Graphs: concepts
2	Graph representation
3	DFS (and when to use?)
4	BFS & Dijkstra's Algorithm
	Lab: Implementation pq

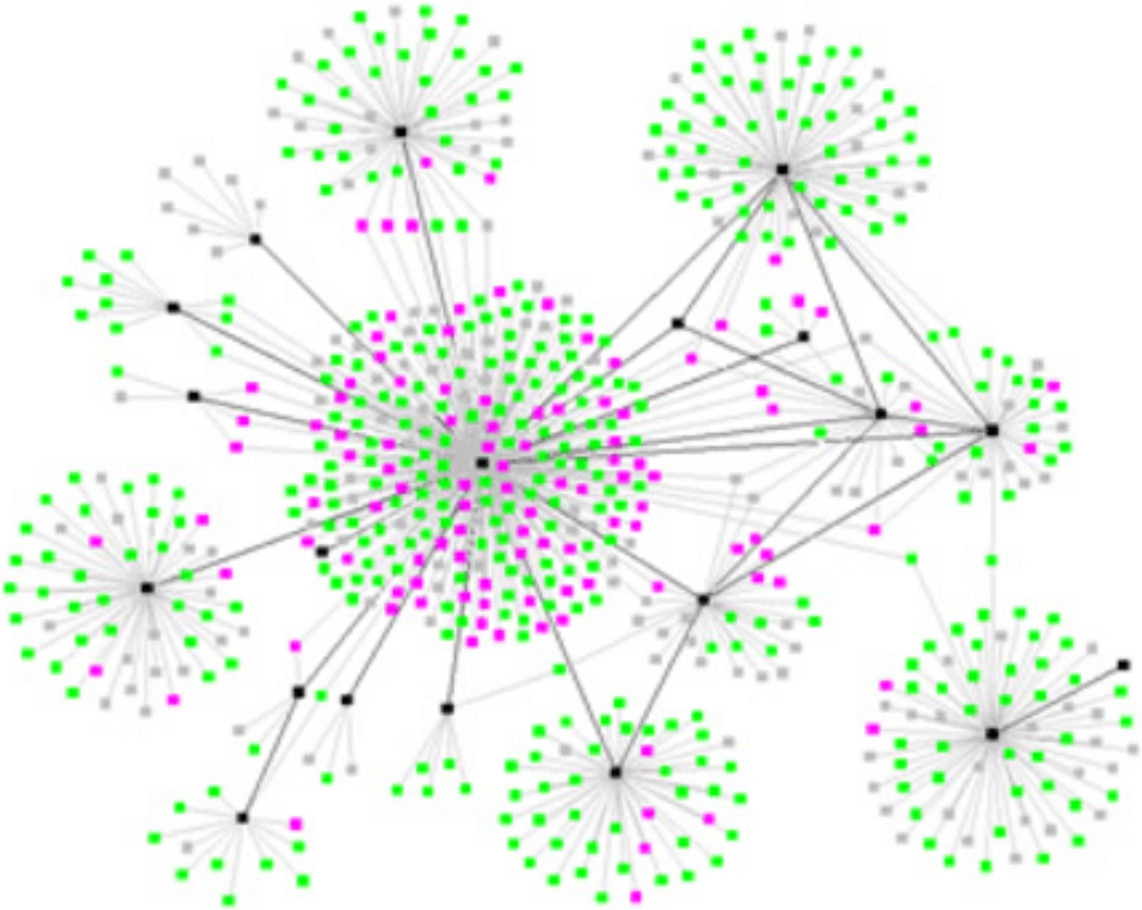
Graphs



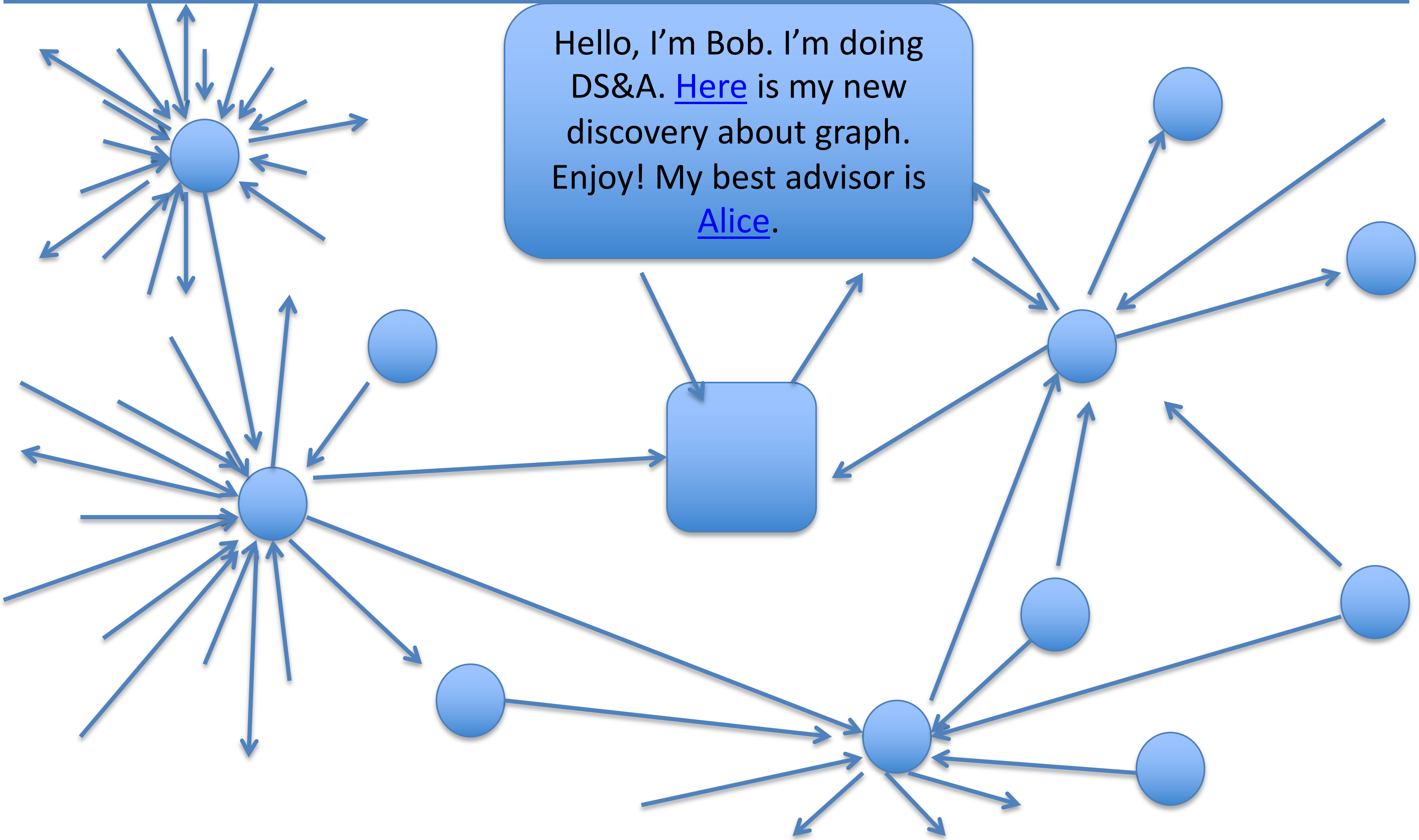
Source: https://www.researchgate.net/figure/An-example-of-the-type-of-graph-produced-by-contact-tracing-Each-node-represents-an_fig2_48693783

a small graph of covid-19…

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



PageRank: Bob becoming famous!



Graphs: Concepts

Formal definition: $G = (V, E)$ where

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

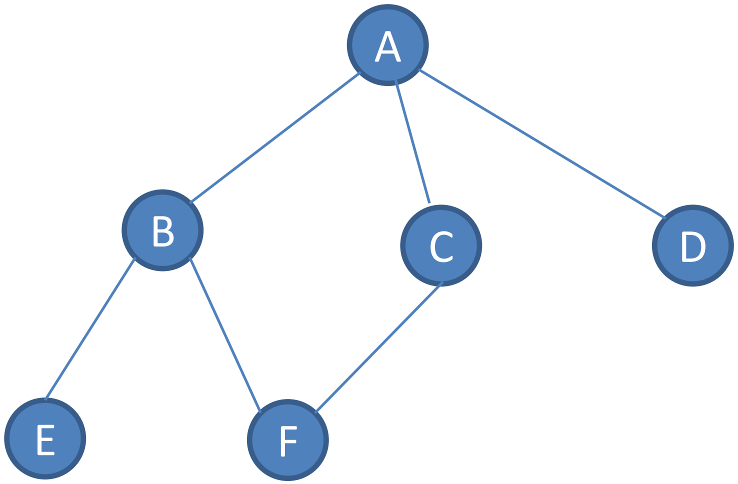
$E = \{(v_i, v_j) | v_i \in V, v_j \in V\}$: set of *edges*, *arcs*, or *links*;

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

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

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

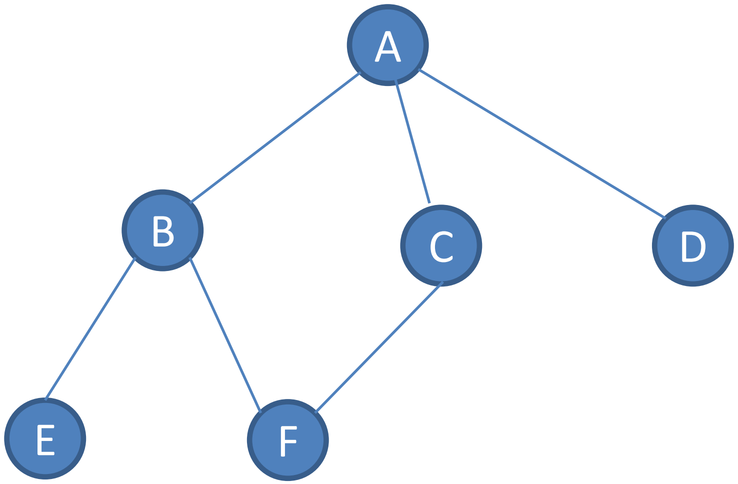
Graph representation: Example for unweighted, undirected graph

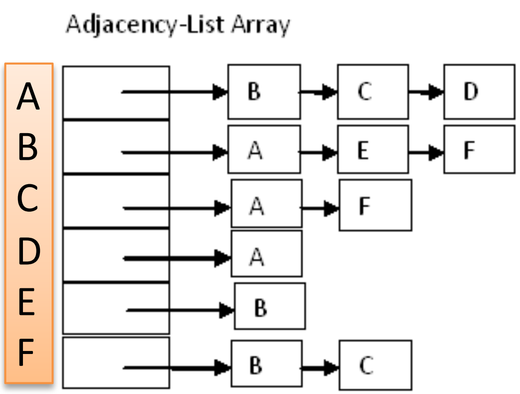


How to represent the graph using:

- Adjacency matrix
- Adjacency lists

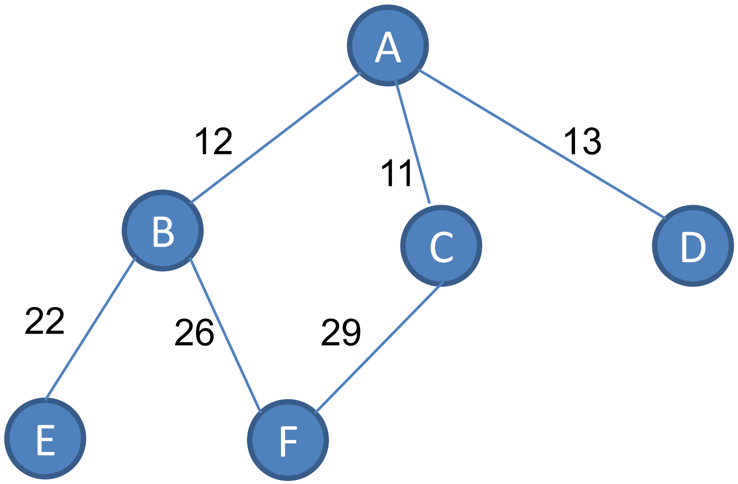
Example for unweighted, undirected graph





	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

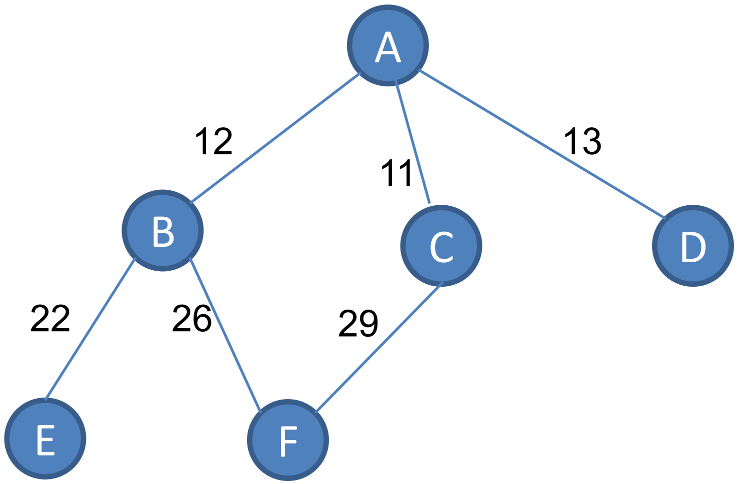
Note: (Weighted) Graph presentation in programs



input data
“A”,“B”,12
“B”,”F”,26
“B”,”E”,22
“A”,”D”,13
“A”,”C”,11
“C”,”F”,29

0 “A” [0] -> (1,12)
1 “B” [1] -> (0,12), (2,26)
2 “F” [2]→ (1,26)

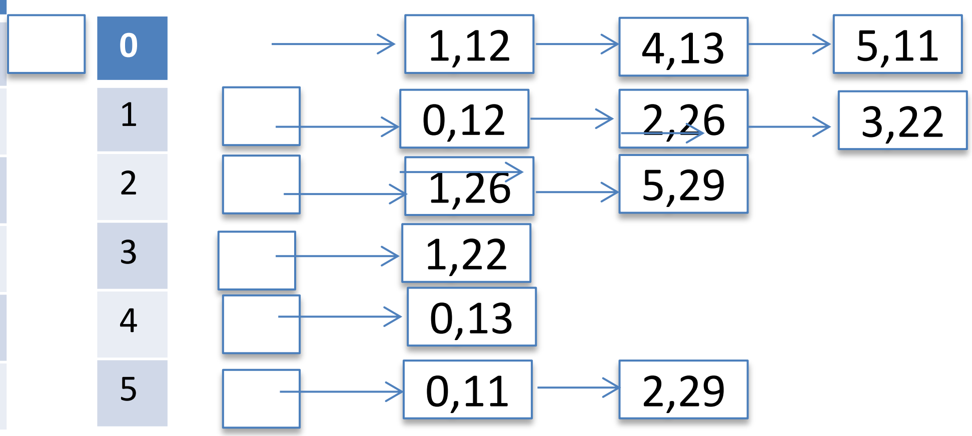
(Weighted) Graph presentation in programs



input data

“A”,“B”,12
“B”,”F”,26
“B”,”E”,22
“A”,”D”,13
“A”,”C”,11
“C”,”F”,29

vertices	
0	“A”
1	“B”
2	“F”
3	“E”
4	“D”
5	“C”



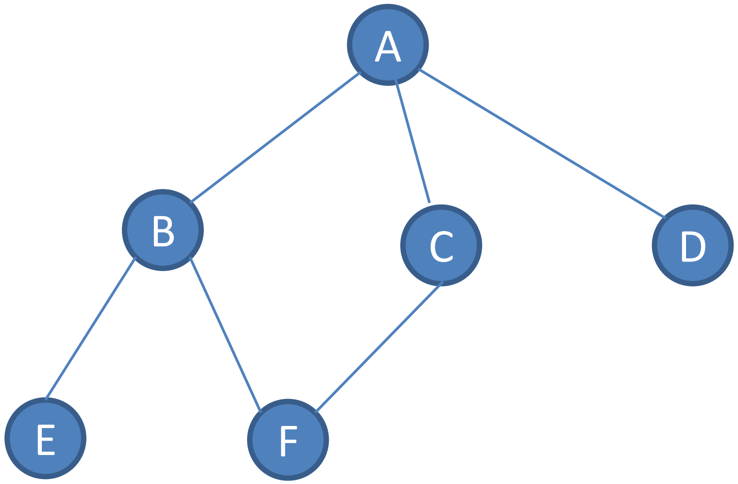
Graphs: Representation

What is a suitable representation method for:

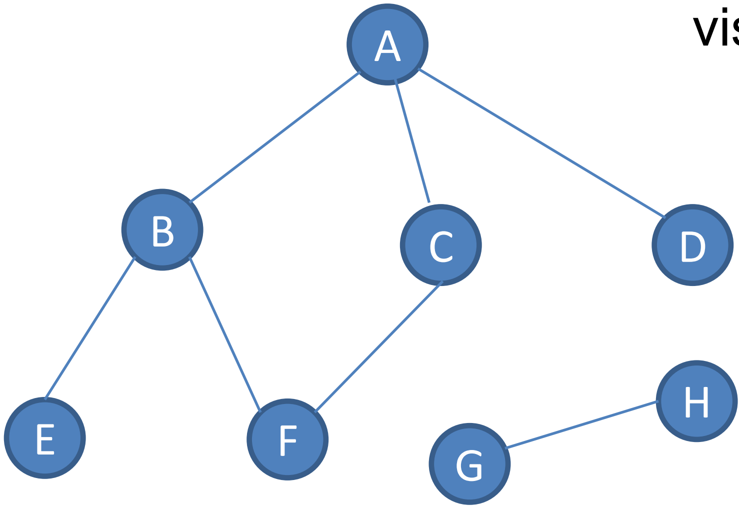
- a graph of this on-line class, where nodes represent students, edge (a,b) means “a knows b”,
- a graph of this would-be-face-to-face class, where nodes represent students, edge (a,b) means “a knows b”
- the webgraph?
- social-distancing graph for people in Australia
- road network between major cities in Australia

Is each graph directed/undirected, weighted/unweighted, cyclic/acyclic, dense/sparse?

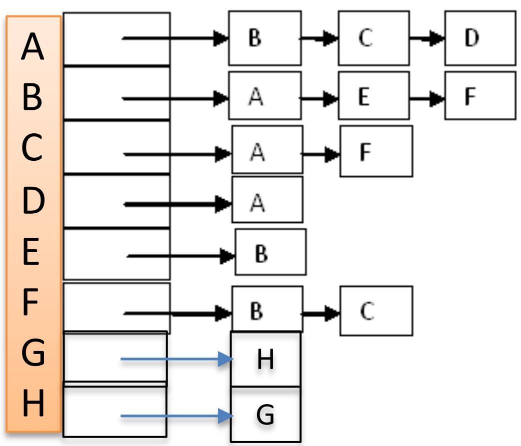
Graph Traversal = What? How?



(Connected) Graph Traversal = How: DFS vs BFS



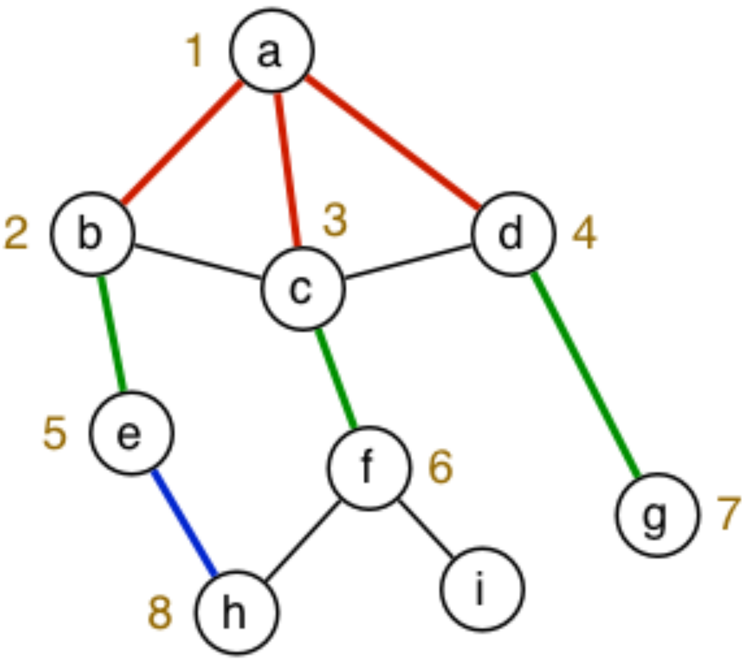
visit each node exactly once, in a systematic way



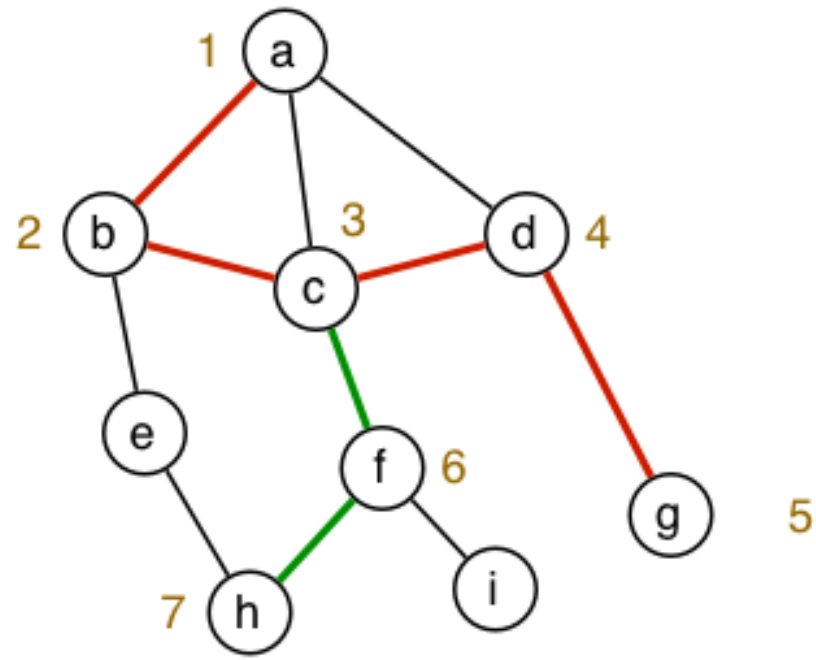

```
//mark all nodes as "unvisited":
visited[u]= 0 for each u= 0..|V|-1
order= 0; c=0;
for each node u {
  if (!visited(u)) {visit(u); c++;}
}

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

Graphs: DFS & BFS



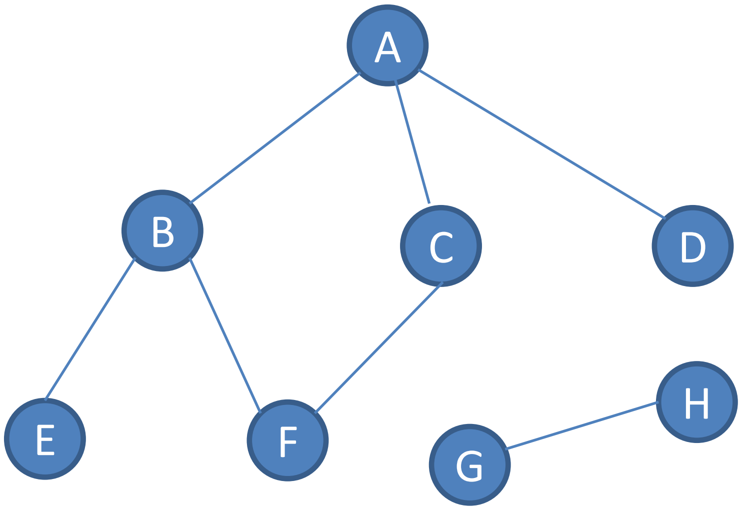
Breadth-first Search



Depth-first Search

Exercise: List nodes in order visited by a)DFS and b)BFS

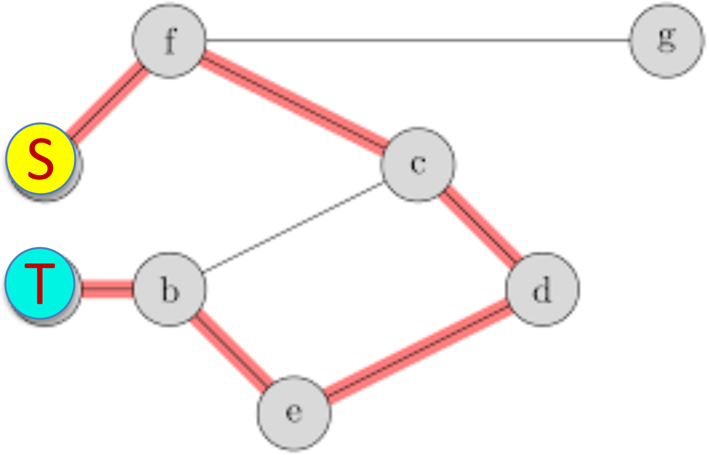
Notes: In exercises, ties should always be broken in alphabetical/ascending order

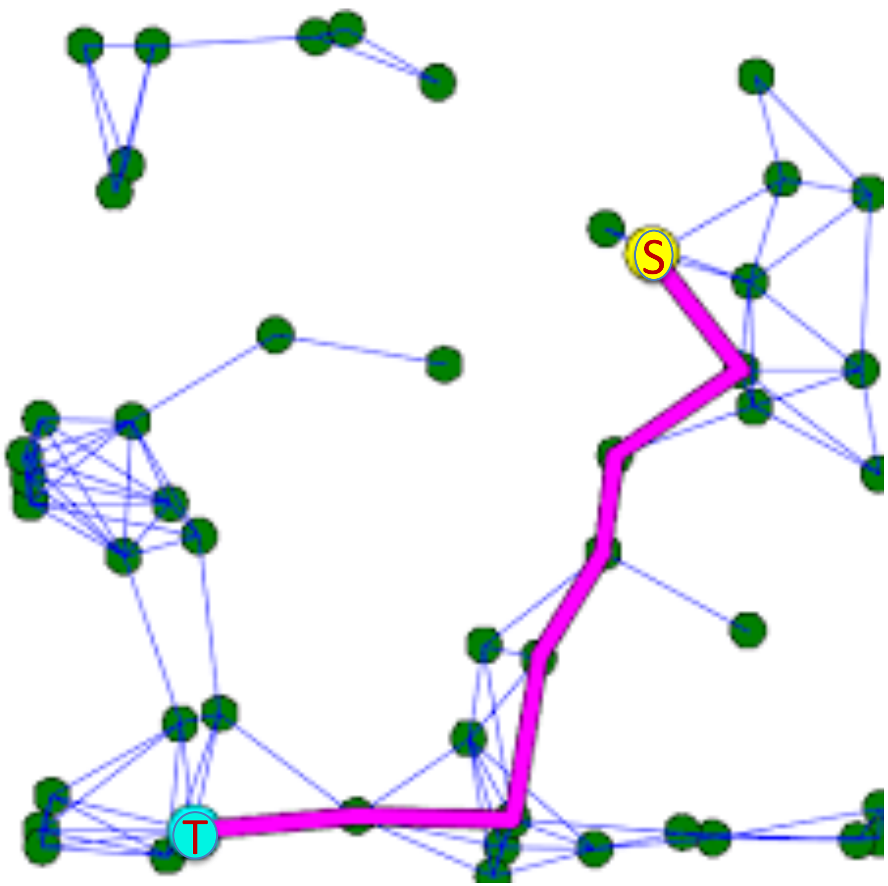


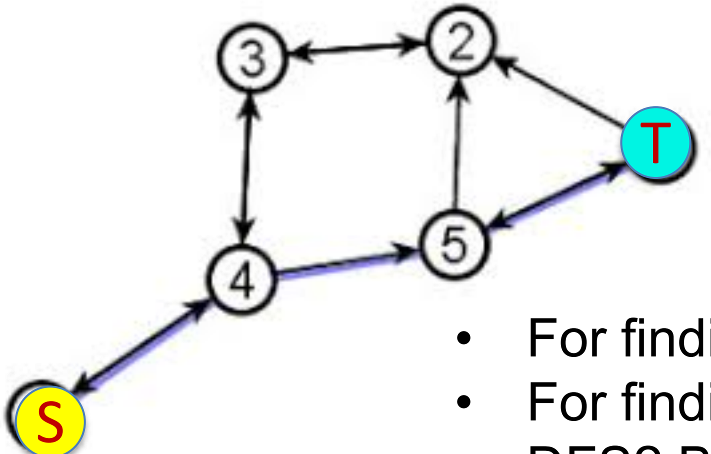
DFS:
A

BFS:
A

Paths in unweighted graphs: path length, shortest path

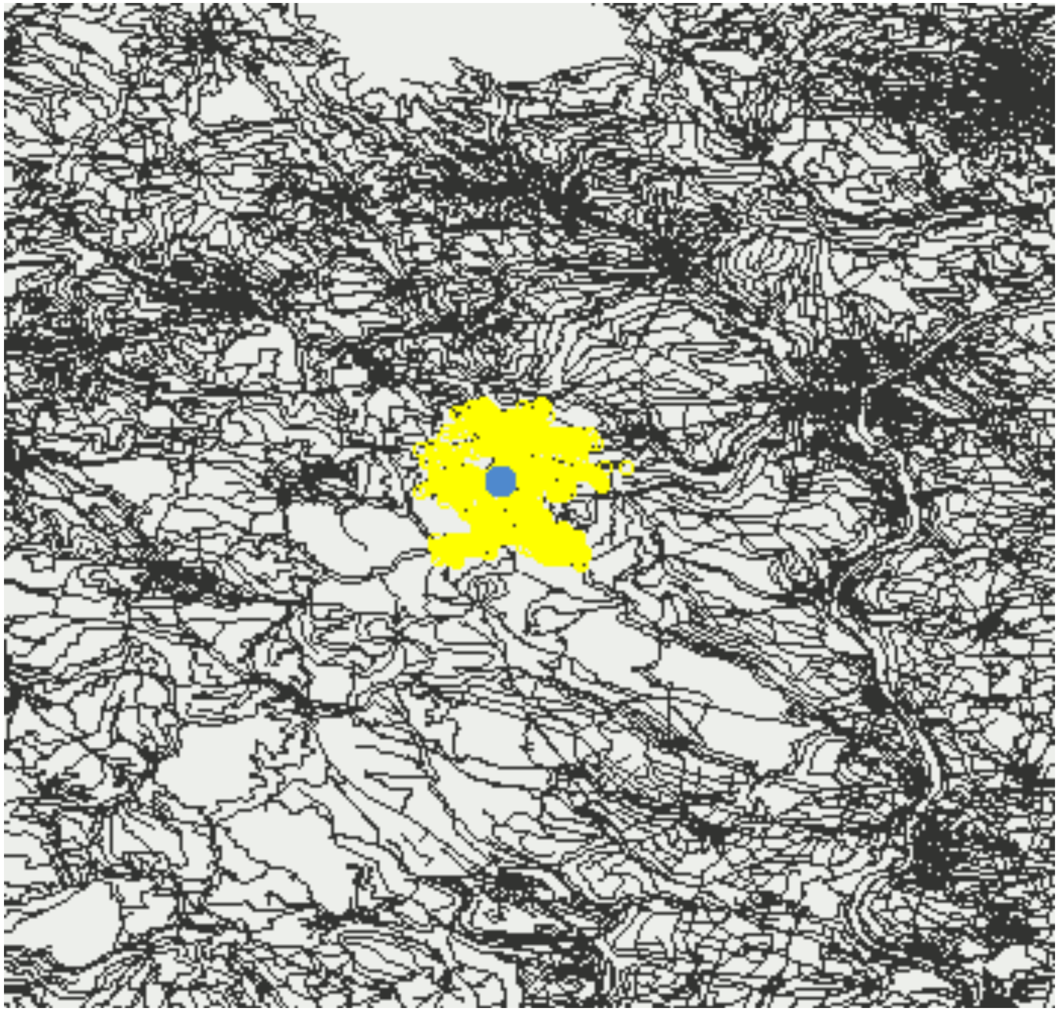




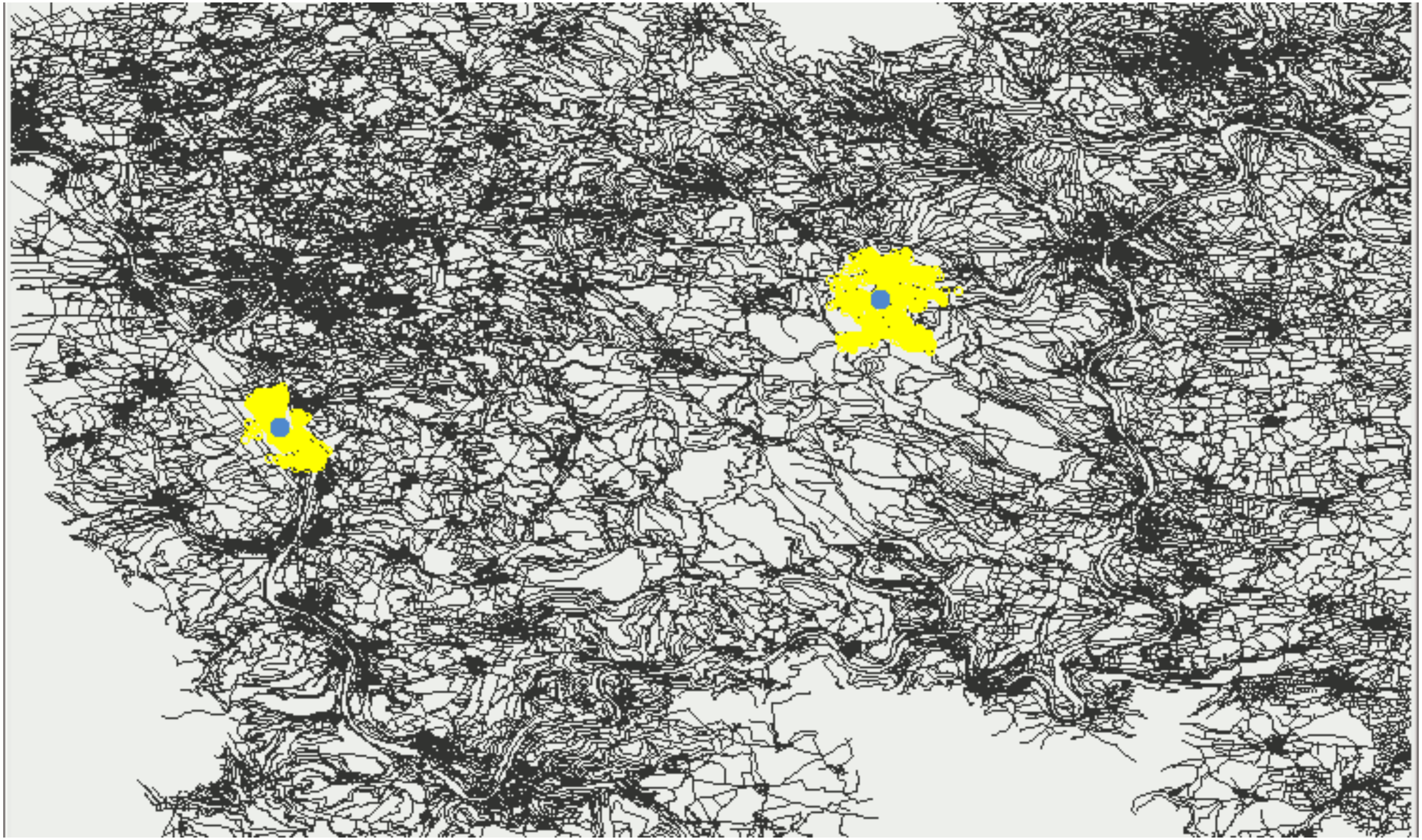


- For finding a path from S to T can we use DFS? BFS?
- For finding a shortest path from S to T can we use DFS? BFS?
- Applied to directed graphs? cyclic graphs?

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



Breath-first Search (BFS): visit all neighbors first



`visitBFS`: BFS from a single vertex

The task:

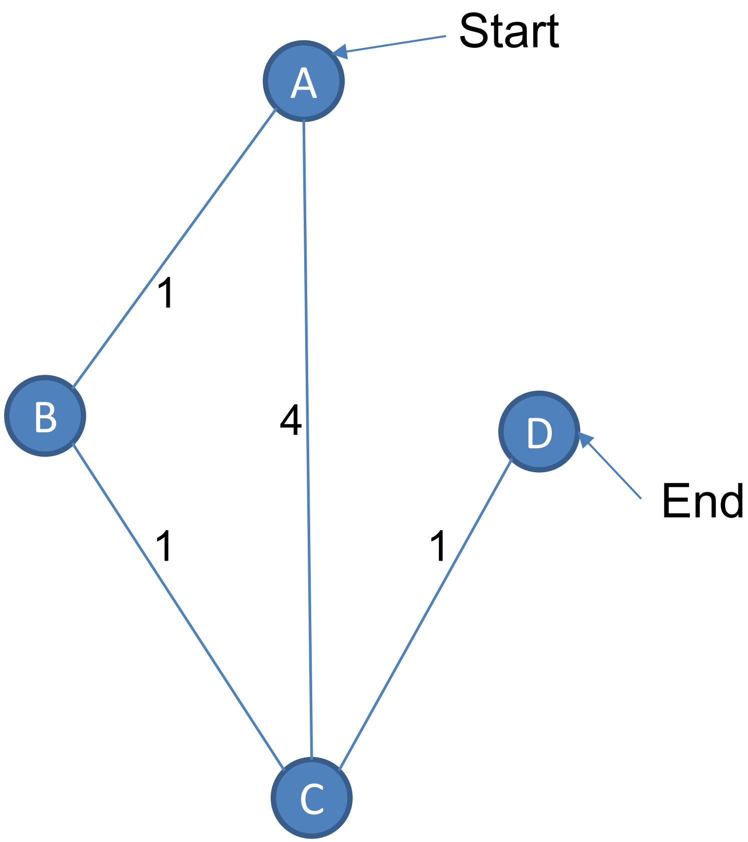
- Given a weighted graph `G=(V,E)`, and `s∈V`
- In the BFS manner, visit all vertices which are reachable from `s`.
- supposing `visited[]` and `order` have been set

The algorithm (using global `visited[|V|]={0}, order= 0;`)

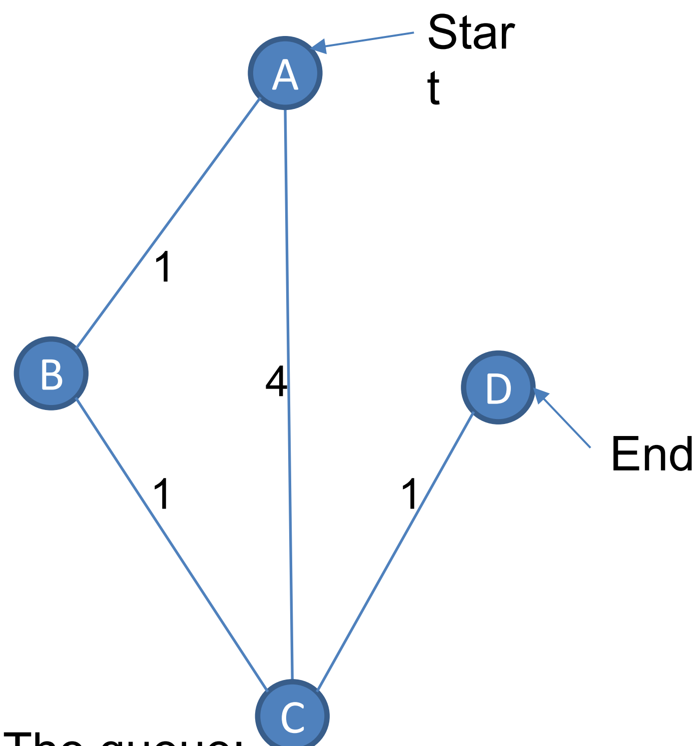
```
visitBFS(s) {    // similar to visit(int u) seen before
    Q= makeEmptyQueue()
    enQ(Q,s)
    while (Q is not empty) {
        u= deQ(Q)
        if (!visited[u]) {
            visited[u]= ++order
            visit(u)  // performs some operations on vertice x
            for all v that (u, v) ∈ E: enQ(Q,v)
        }
    }
}
```

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

How to use BFS for finding shortest paths in weighted graphs, supposing weights are positive integers?

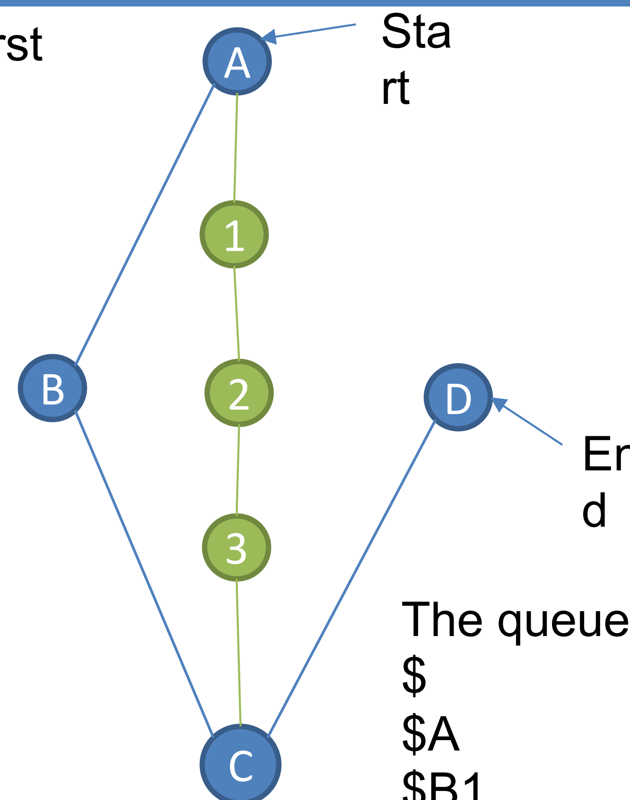


Exercise…



The queue:
$
$A
$BC
$C //path AB found
$D //path ABC found
$ // path ABCD found

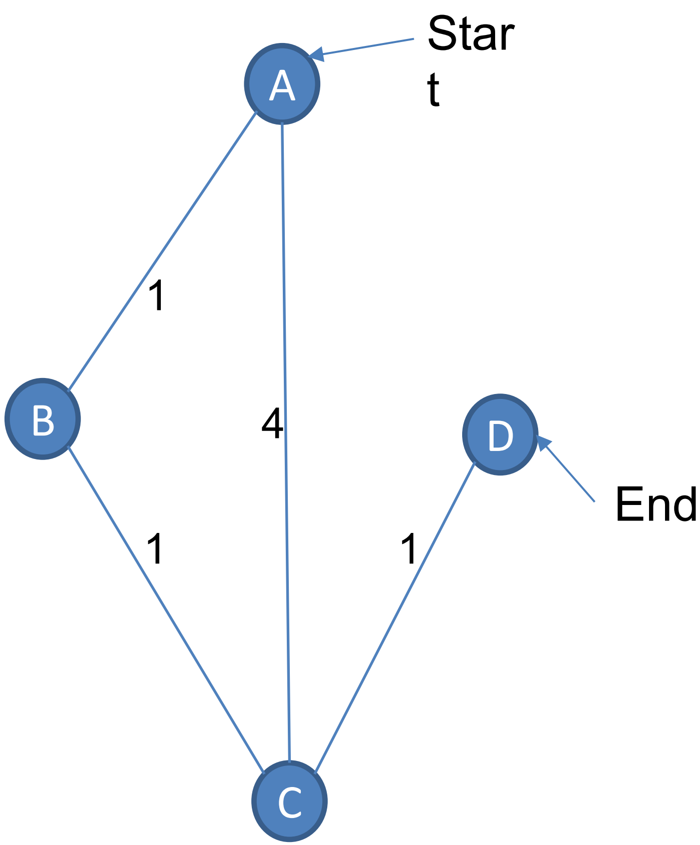
Adapting Breadth-First Search to work on weighted graph



The queue:
$
$A
$B1
$1C //path AB
$C2
$2D3 //path ABC
$D3
$3 //path ABCD
$

Is there a better way?

Finding shortest paths using priority queues



content of PQ

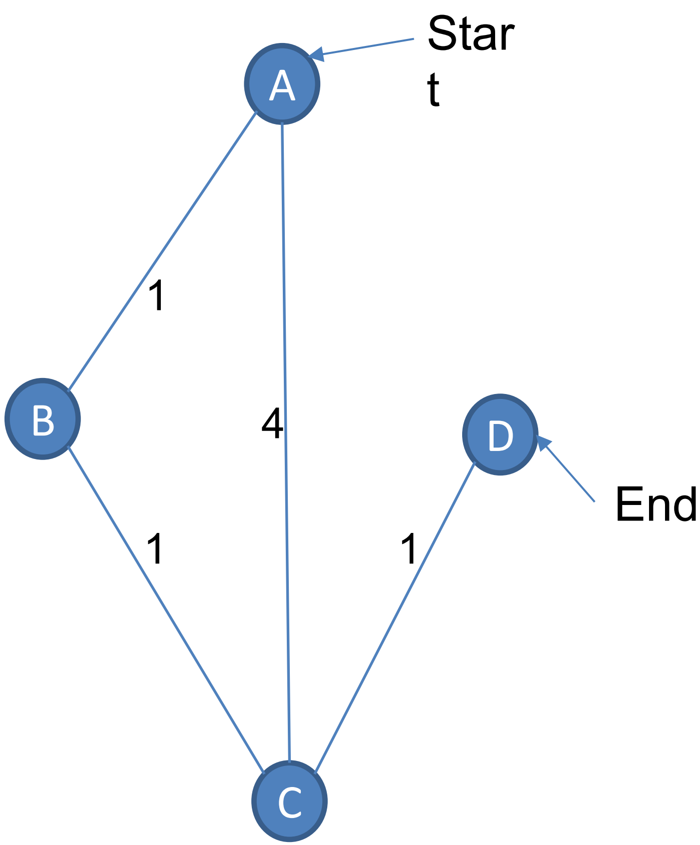
$

$ (A,0)   insert A with distance A→A = 0

$ (B,1), (C,4)   done with B A→B : 1

$

Finding shortest paths using priority queues



content of PQ
$
$ (A,0)

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

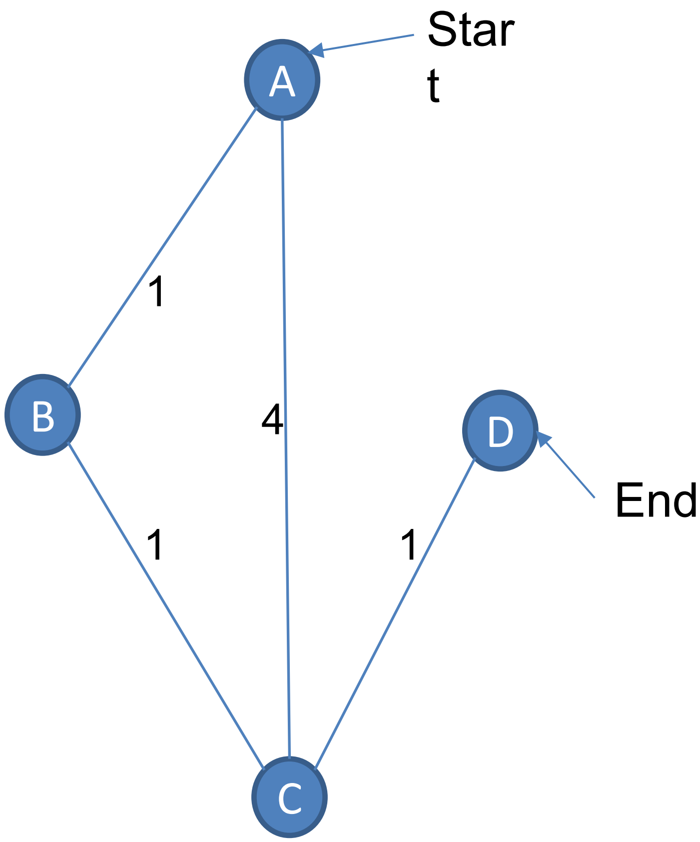
(C,2)
$ (C,2)

$ is front of the PQ, PQ is empty now
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

…
→ it would be easier just to populate the PQ with all vertices from the very start

Finding shortest paths using priority queues



Now we put all vertices into the queue at the start. The content of PQ will be:

$ (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)}
prove that *:* A→B is a shortest from A to B
update…

$ (C,2), (D,∞)

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

$ (D,3)

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

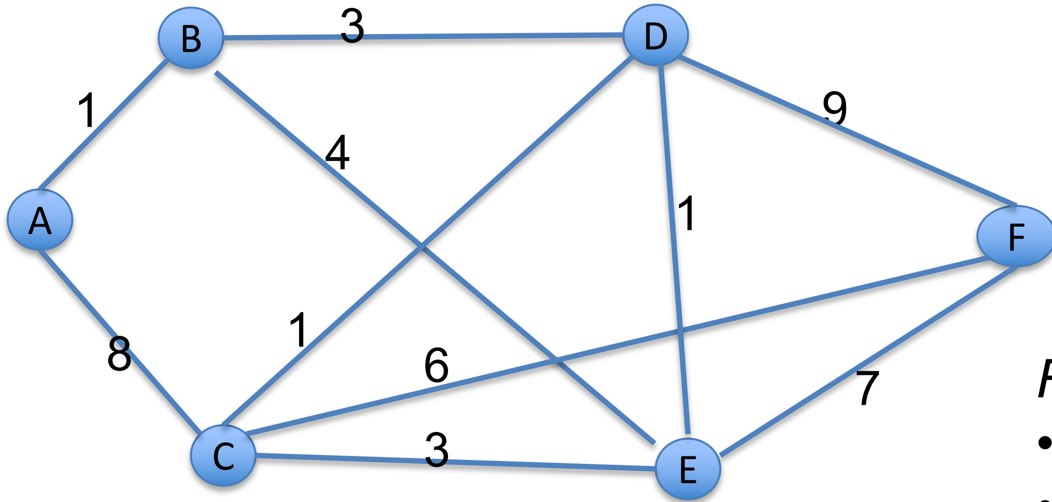
$

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

Dijkstra’s Algorithm: Single Source Shortest Path SSSP

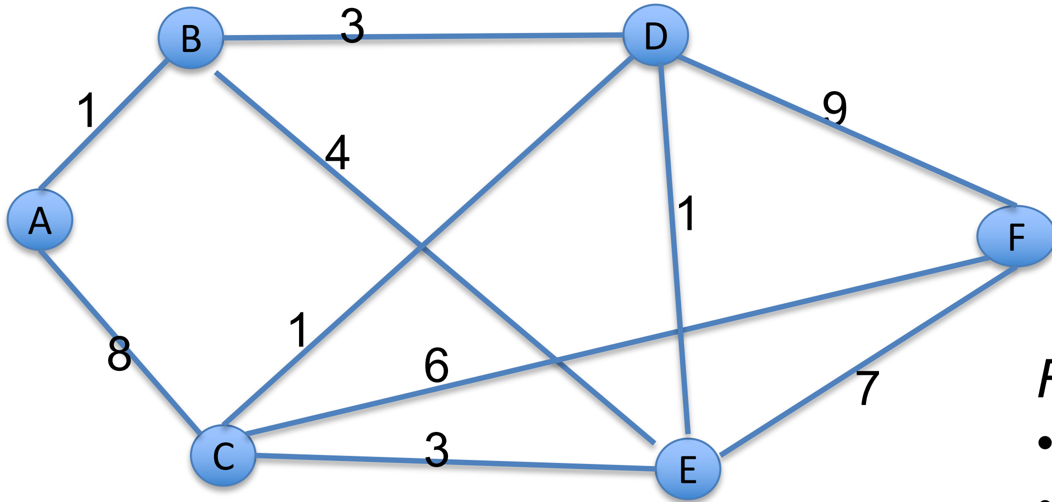
The task:

- Given a weighted graph `G=(V,E,w(E))`, and `s∈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 F
- From A to any other node



Find a shortest path:

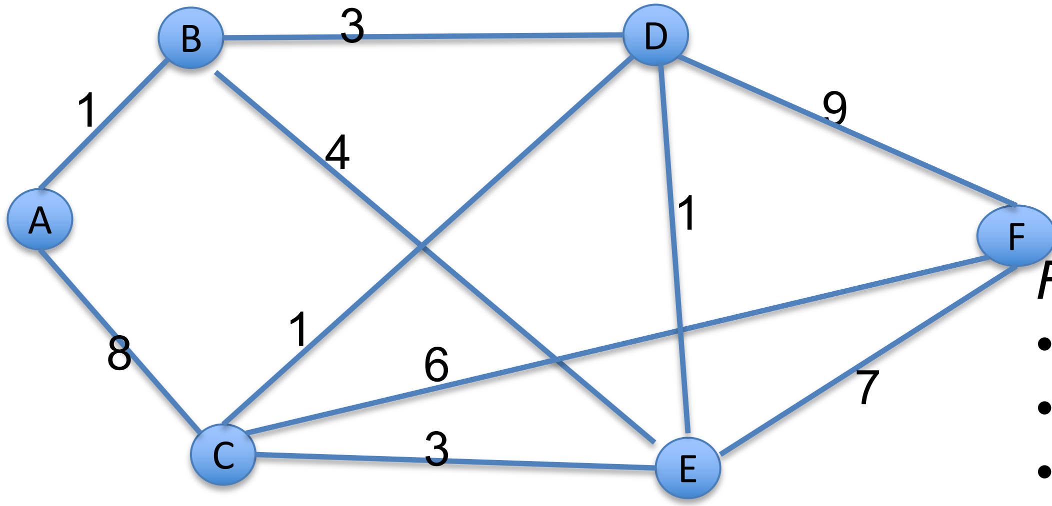
- From A to B
- From A to C
- From A to F
- From A to any other node

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 ∞

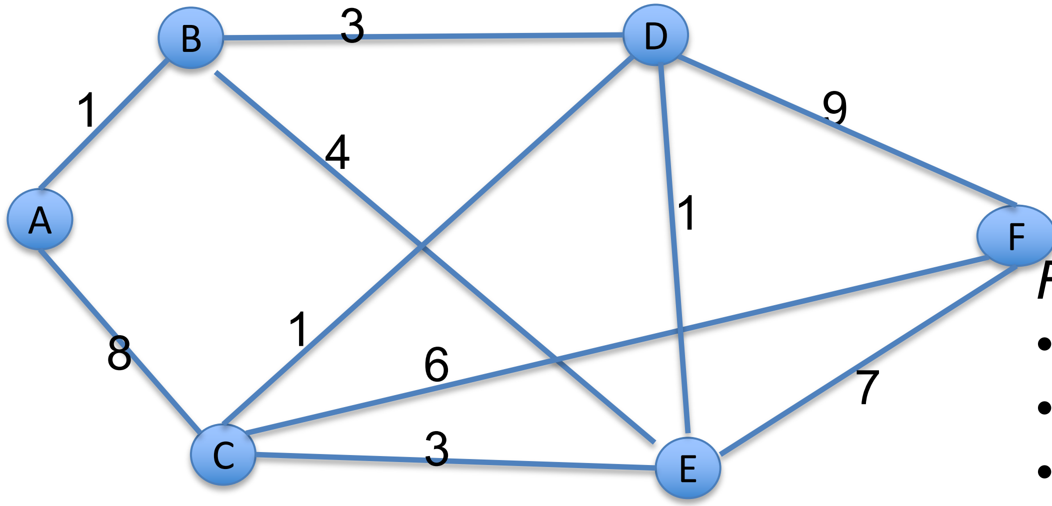
The "-" is shorthand for "∞,nil"

4 non-empty cells means there are 4 elements in the heap

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						

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

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						

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 `unvisited_set= V`

Round 1:

- choose node with min `dist[],` which is `s`;
- visit all nodes `u` adjacent to `s` and update `dist[u]`;
- mark `s` as visited (remove it from the `unvisited_set)`;

Round 2:

- choose the node with min `dist[]` from `unvisited_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 min PQ

```
while (PQ is not empty):
   remove u from PQ ( 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
```

Dijkstra’s Algorithm

Learn more about the algorithm, and its complexity in the lectures

Exercise

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

- Do the exercise in previous page
- If you didn't do the priority queue exercise from the bonus workshop, the sample solution will likely solve you a lot of work
- A couple of sets of practice problems are also provided in week10-practice-problems.pdf – If you're struggling with the Dijkstra task, this may help with the conceptual lead-in
- Grady will also release his .ppt with some detailed slides

Lab:

- Implement priority queue (see LMS):
  - you can use your heap implementation from previous weeks