Paths and connectivity

pseudocode: * what order to consider the edges?
* obscure way to implement things

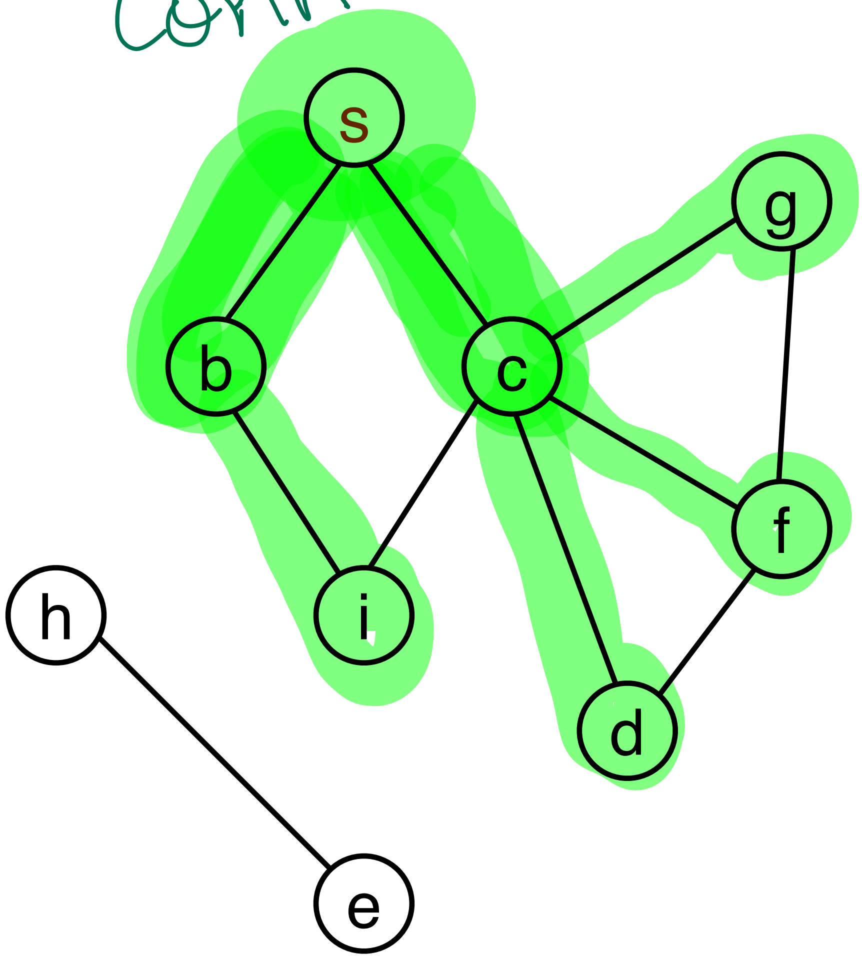
Question. Is there a path from s to e?

Def. An undirected graph is connected, if there is a path between any pair of nodes.

Task: Given a source node s, find all nodes connected to s.

input: a graph
a vertex

conn



Algorithm 1: GraphSearch$(G(V, E), s)$

```
/* $G$ is an undirected graph, $s$ a source node
conn $\leftarrow \{s\}$/* set of nodes connected to $s$
while there is edge $(u, v)$ with $u$ in conn and $v$ not in conn do
    add $v$ to conn;
return conn
```

→ output: set of vertices

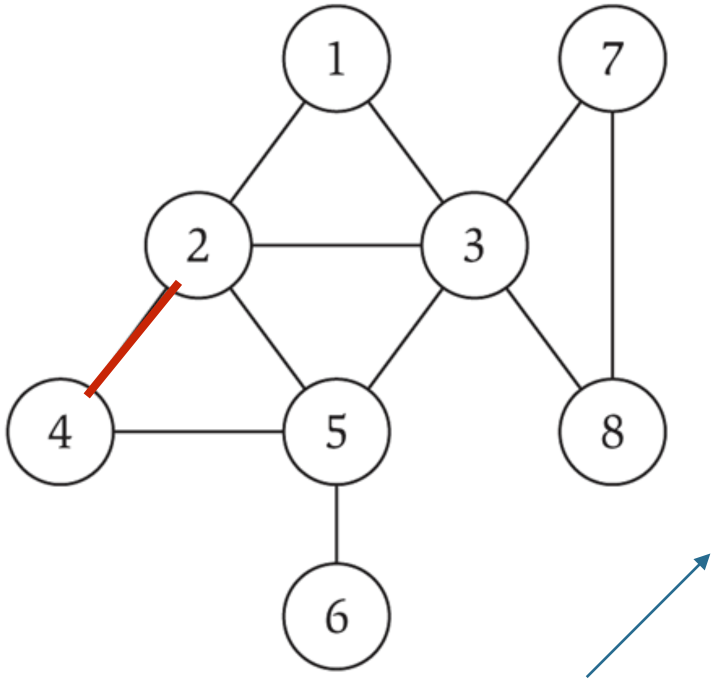
s - c - d (simple path)

s - c - g - f - c - d

(not simple path)

Graph representation: adjacency matrix

Adjacency matrix: nxn binary matrix A, such that A[i,j]=1 iff (i,j) is an edge



$$\begin{array}{c|cccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 3 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 4 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 5 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 7 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 8 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{array}$$

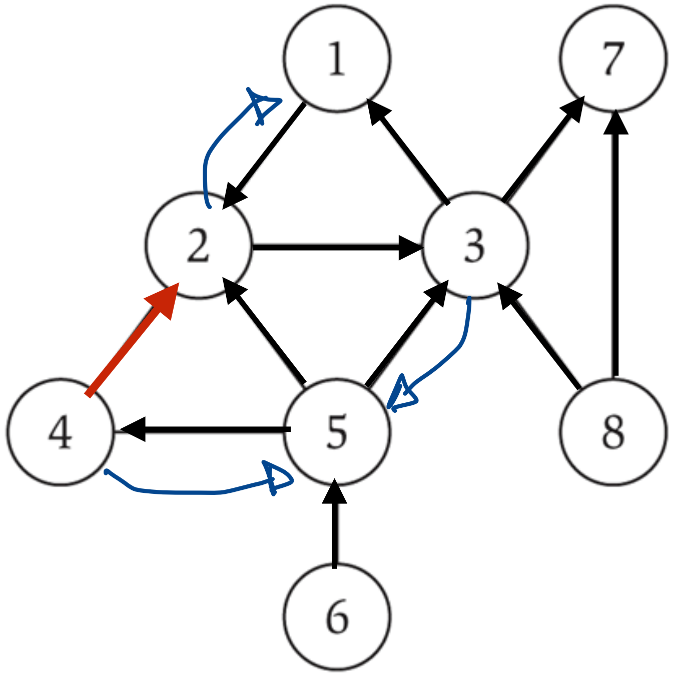
node ids

symmetric

undirected graph

there is no edge connecting 2 to 4

directed graph



$$\begin{array}{c|cccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 4 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 5 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 8 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{array}$$

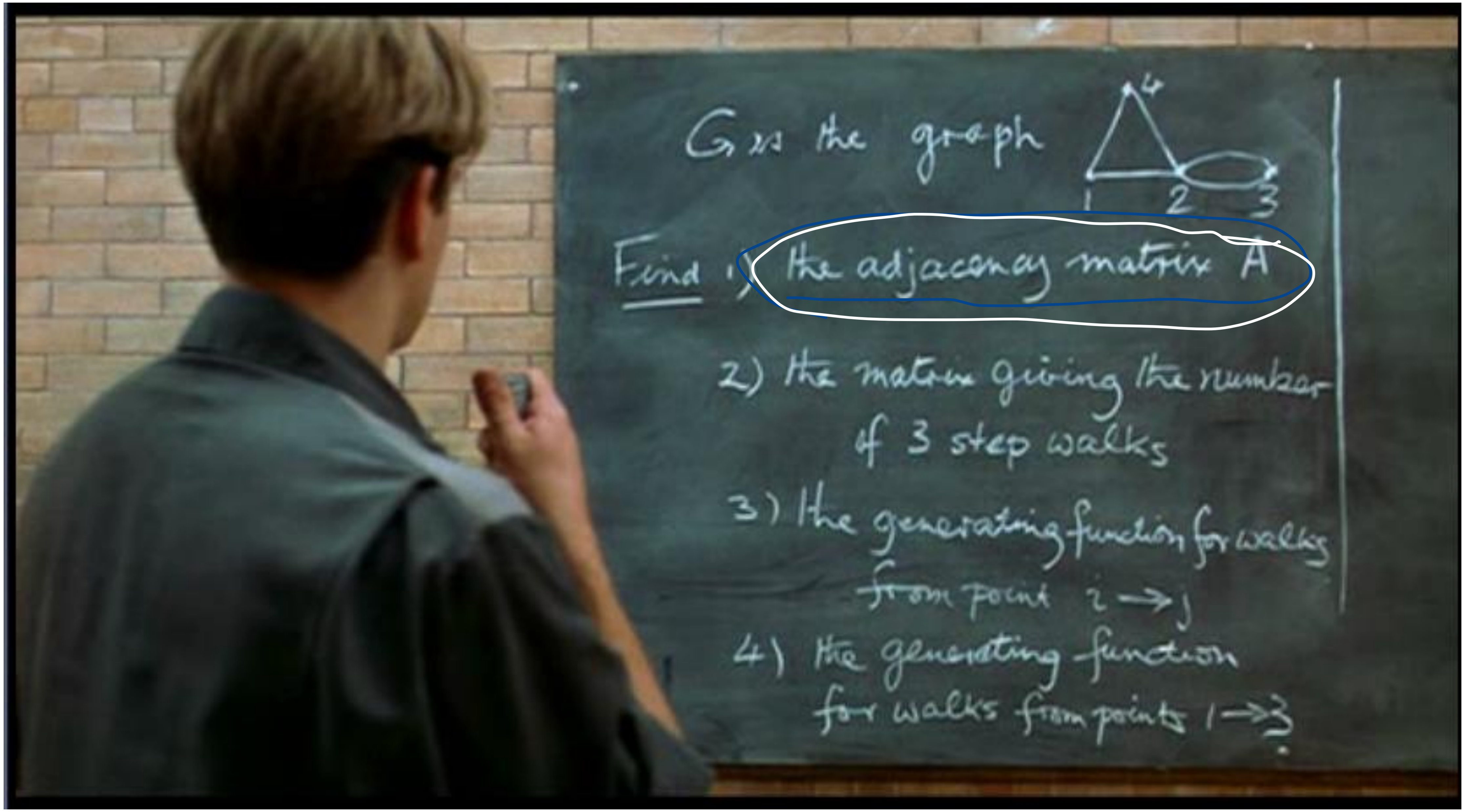
asymmetric

directed adjacency matrix:

rows: node of origin of edge e

cols: destination node

Adjacency matrix in Good Will Hunting.



G is the graph

Find 1) the adjacency matrix A

2) the matrix giving the number of 3 step walks

3) the generating function for walks from point i → j

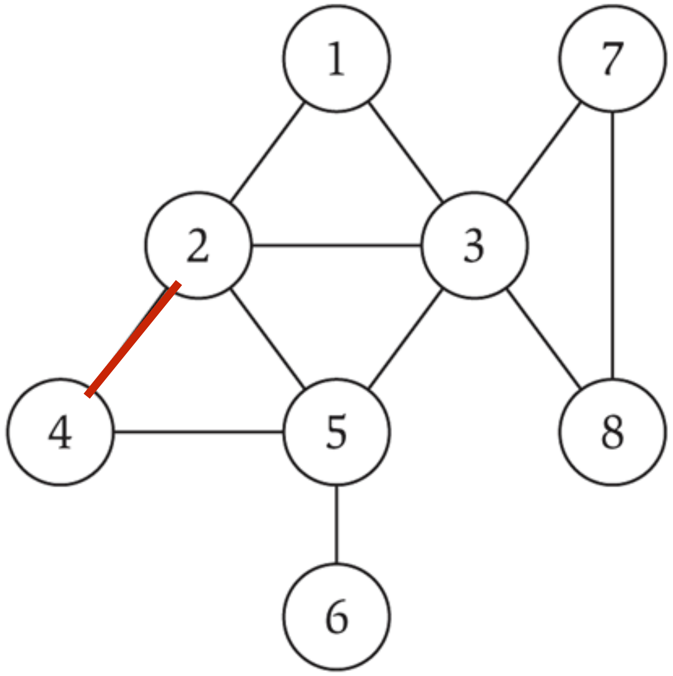
4) the generating function for walks from points 1 → 3

https://www.imdb.com/title/tt0119217/

https://graph-theory.blogspot.com/2008/11/good-will-hunting-problem.html

Operations on an adjacency matrix

Adjacency matrix: nxn binary matrix A, such that A[i,j]=1 iff (i,j) is an edge



$$
\begin{array}{c|cccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\hline
1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
2 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
3 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\
4 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
5 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\
6 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
7 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
8 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0
\end{array}
$$

How do we perform a lookup? e.g. check whether (u,v) is an edge

Check the value of the matrix $A[u,v]$ at the intersection of row u and column v.

if $A[u,v] = 1 \rightarrow$ connected

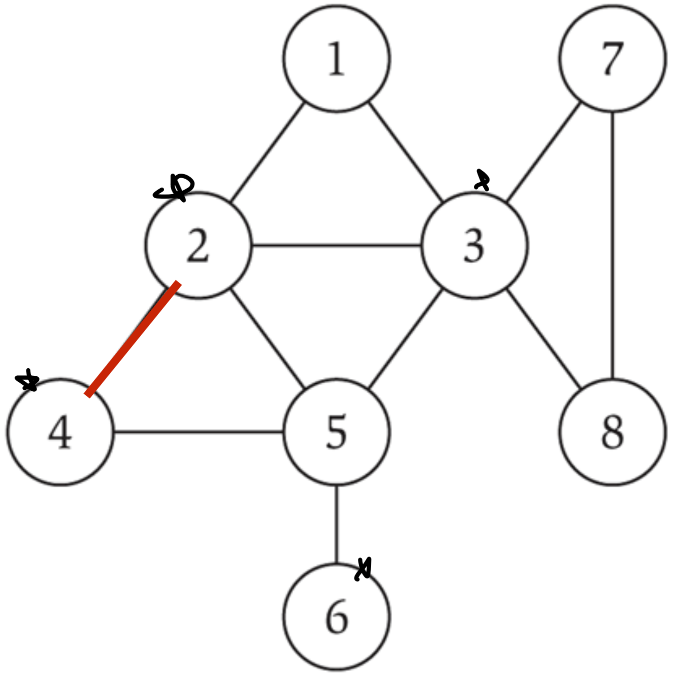
if $A[u,v] = 0 \rightarrow$ disconnected

How many operations is that?

$\Theta(1)$ = constant time

Operations on an adjacency matrix – TopHat

Adjacency matrix: nxn binary matrix A, such that A[i,j]=1 iff (i,j) is an edge



$$\begin{array}{c|cccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 3 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 4 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 5 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 7 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 8 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{array}$$

Question 1.: (multiple choice!)

Complexity of listing all neighbors of vertex v?

A. $\Theta(1)$

[✓ B.] $\Theta(n)$

C. $\Theta(n^2)$

Complexity of listing every edge in G?

D. $\Theta(n)$

[✓ E.] $\Theta(n^2)$

F. $\Theta(m)$

Operations on an adjacency matrix – directed

How would you implement the following operations and what is their complexity?

- Lookup: verify whether the pair (u,v) form a directed edge

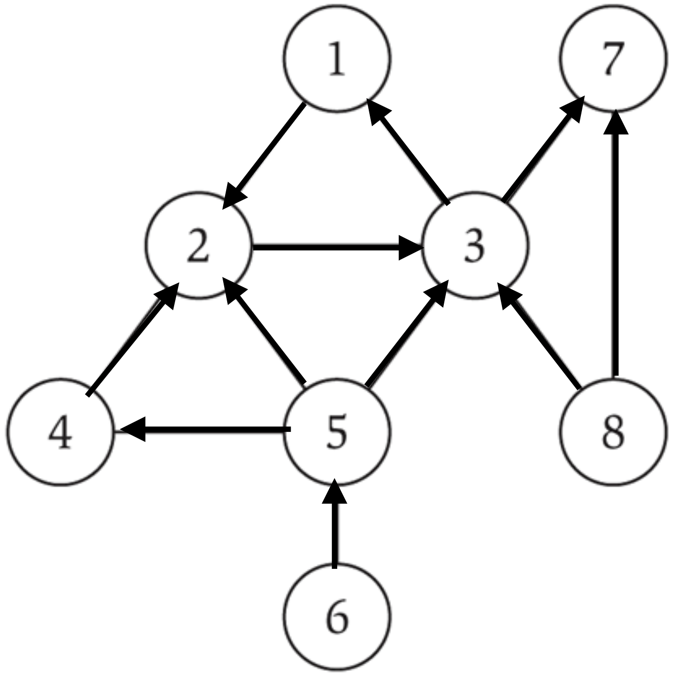
Check if the intersection of row u and column v is 1

- Find the out-neighbors of v, or find the out-degree of v. (degree = number of edges directed away from v) Count the amount of 1 in the row v
- Find the in-neighbors of v

Count the amount of 1's in the column v

- List every directed edge in G

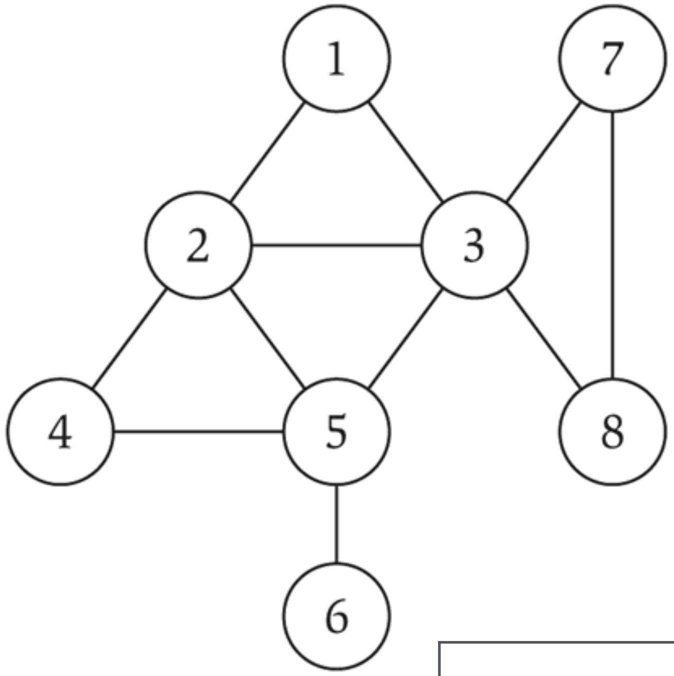
```
count = 0
for i=1 to n do:
    for j=1 to n do:
        if A[i,j]==1 then:
            count += 1
return count
```



$$\begin{array}{c|cccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\hline
1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
3 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
4 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
5 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
6 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
8 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0
\end{array}$$

Graph representation: adjacency matrix

Adjacency matrix: nxn binary matrix A, such that A[i,j]=1 iff (i,j) is an edge

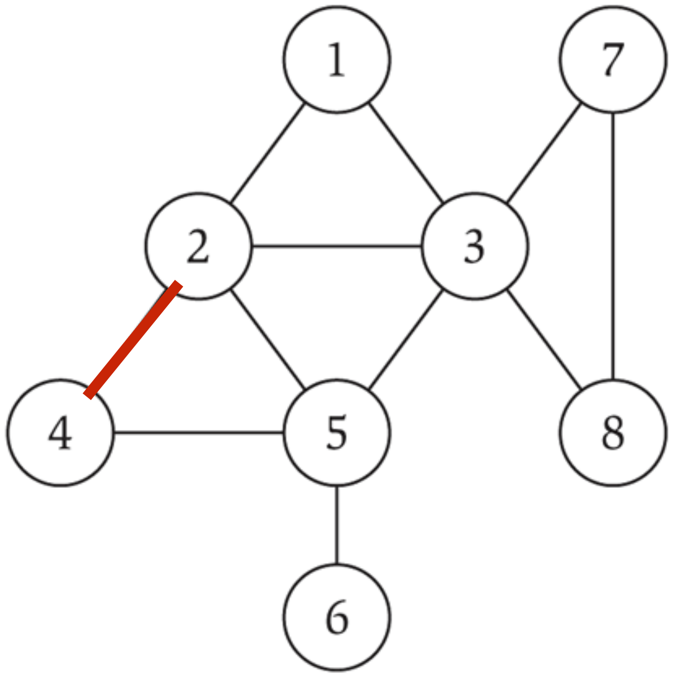


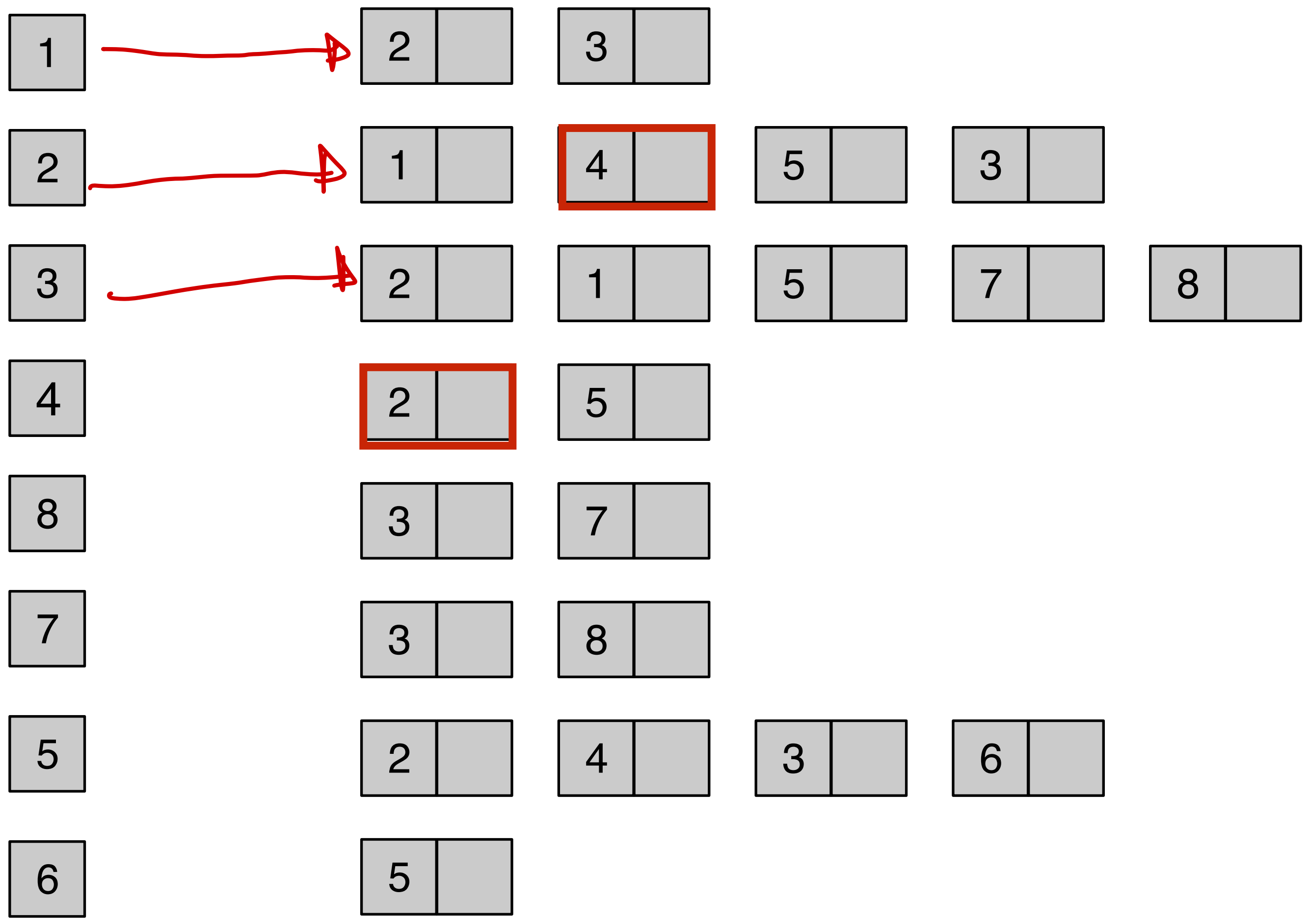
$$
\begin{array}{c|cccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\hline
1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
2 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
3 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\
4 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
5 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\
6 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
7 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
8 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0
\end{array}
$$

space complexity	$\Theta(n^2)$
Operation	**time complexity**
check edge (u,v)	$\Theta(1)$
list all neighbors of u	$\Theta(n)$
list all edges	$\Theta(n^2)$

Graph representation: adjacency lists

Adjacency list: For each node v there is a record listing the nodes to which v is connected. ↳ Uses less space than an adjacency matrix because we only store connected neighbors





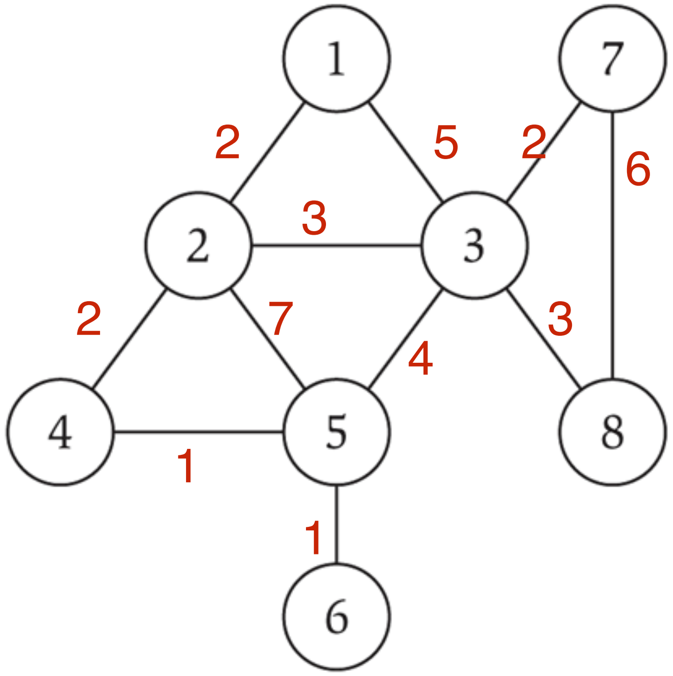
NodeId	Neighbors				
1	2	3			
2	1	4	5	3	
3	2	1	5	7	8
4	2	5			
8	3	7			
7	3	8			
5	2	4	3	6	
6	5				

Note: vertices in an adjacency list need not appear in any particular order.

doesn't need to

Graph representation: adjacency list – weighted graphs

Adjacency list: For each node v there is a record listing the (node, weight) pairs to which v is connected.



neighbor id

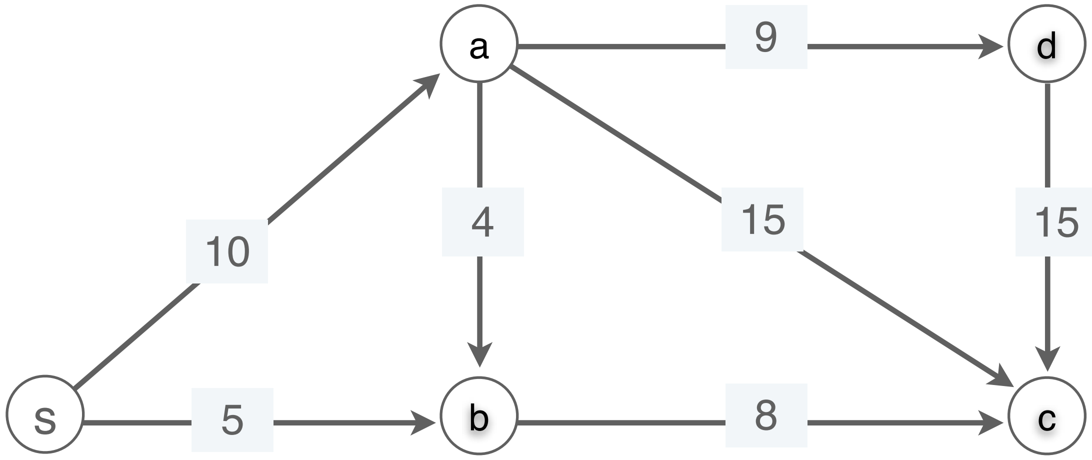
edge weight

NodeId	Neighbors									
1	2	2	3	5						
2	1	2	4	2	5	7	3	3		
3	2	3	1	5	5	4	7	2	8	3
4	2	2	5	1						
8	3	3	7	6						
7	3	2	8	7						
5	2	7	4	1	3	4	6	1		
6	5	1								

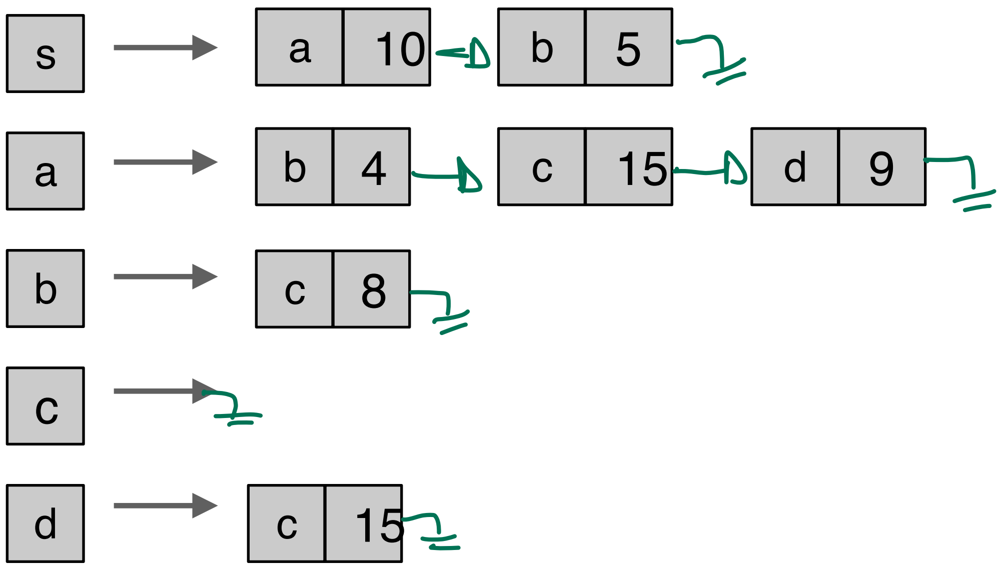
If (u,v) is an edge with weight w, then we store (v,w) instead of (v) in u's neighbor list

Graph representation: adjacency list

Adjacency list: For each node v there is a record listing the nodes *towards* which v has a directed edge.



NodeId Neighbors



s → a 10 → b 5

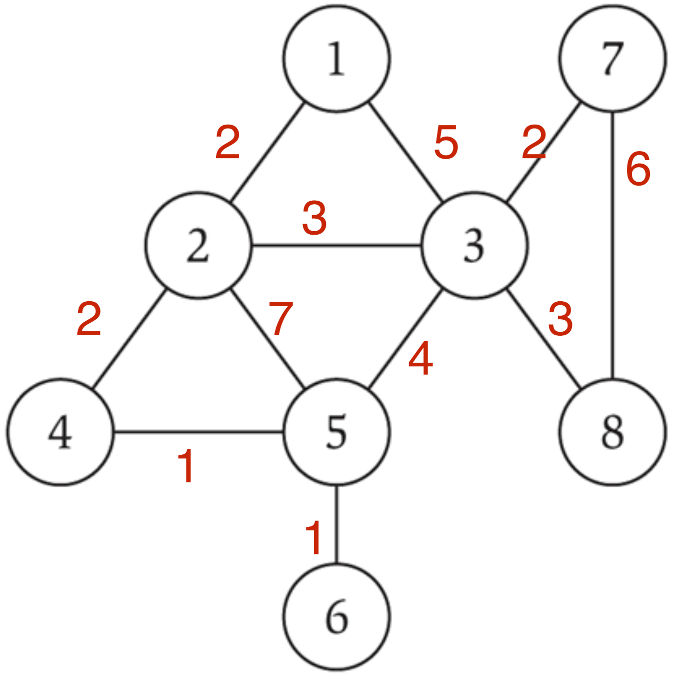
a → b 4 → c 15 → d 9

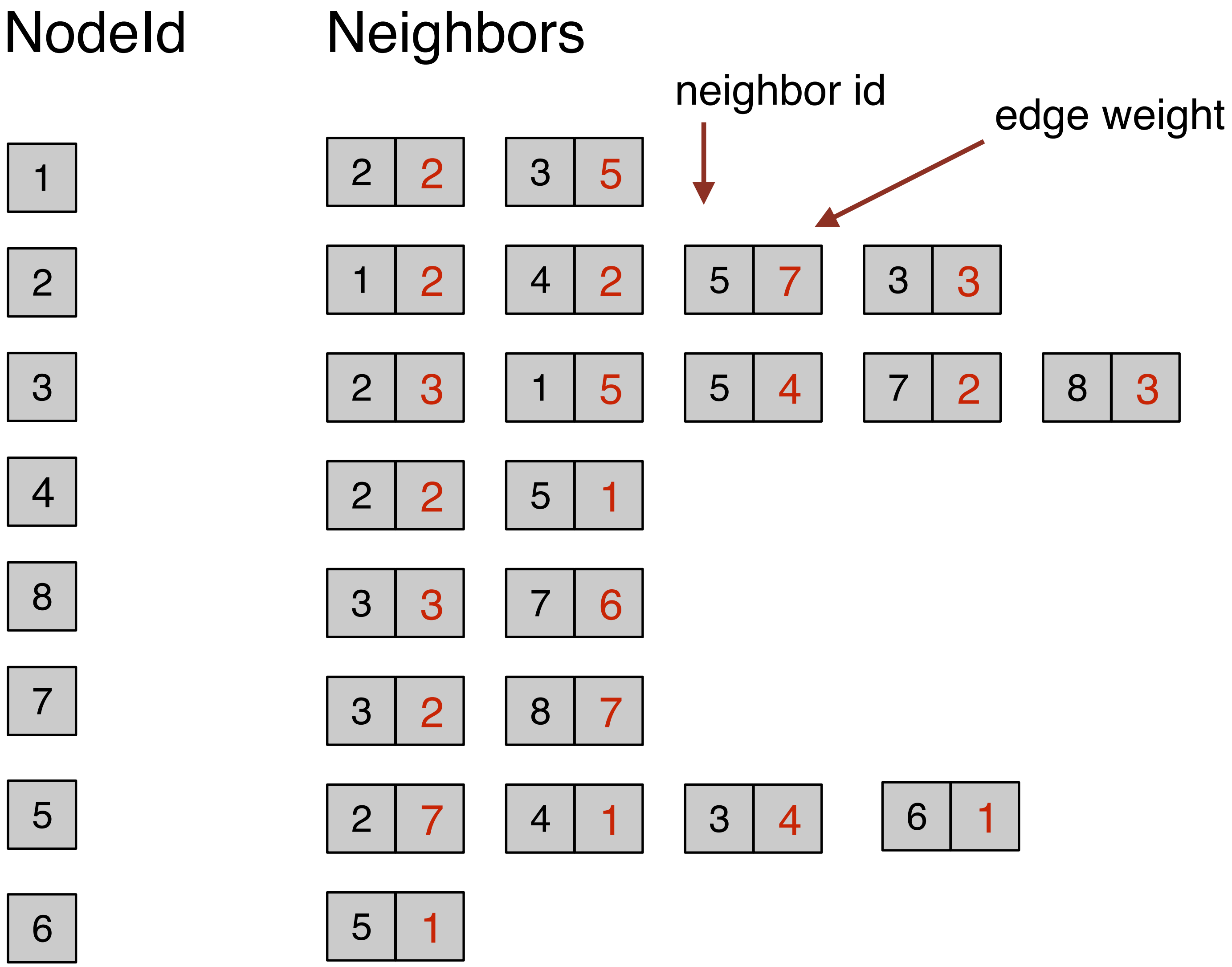
b → c 8

c →

d → c 15

TopHat





NodeId	Neighbors				
1	2 2	3 5			
2	1 2	4 2	5 7	3 3	
3	2 3	1 5	5 4	7 2	8 3
4	2 2	5 1			
8	3 3	7 6			
7	3 2	8 7			
5	2 7	4 1	3 4	6 1	
6	5 1				

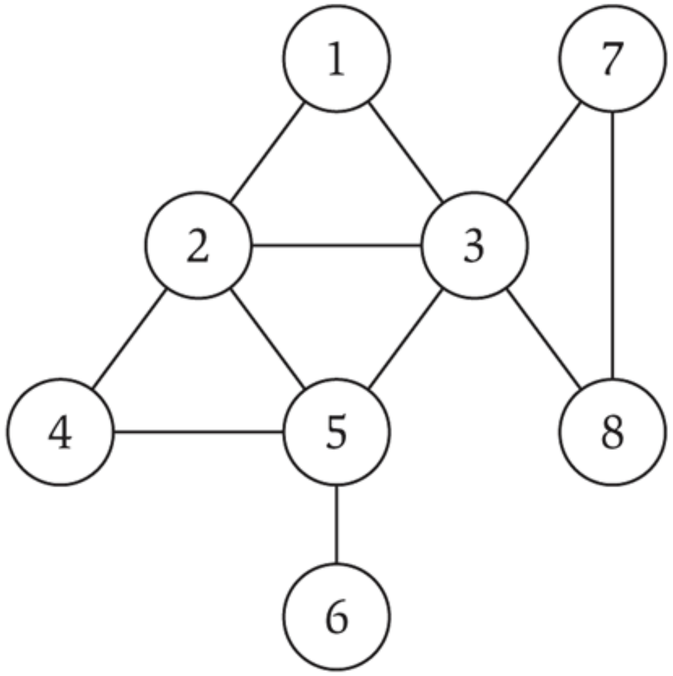
Question:

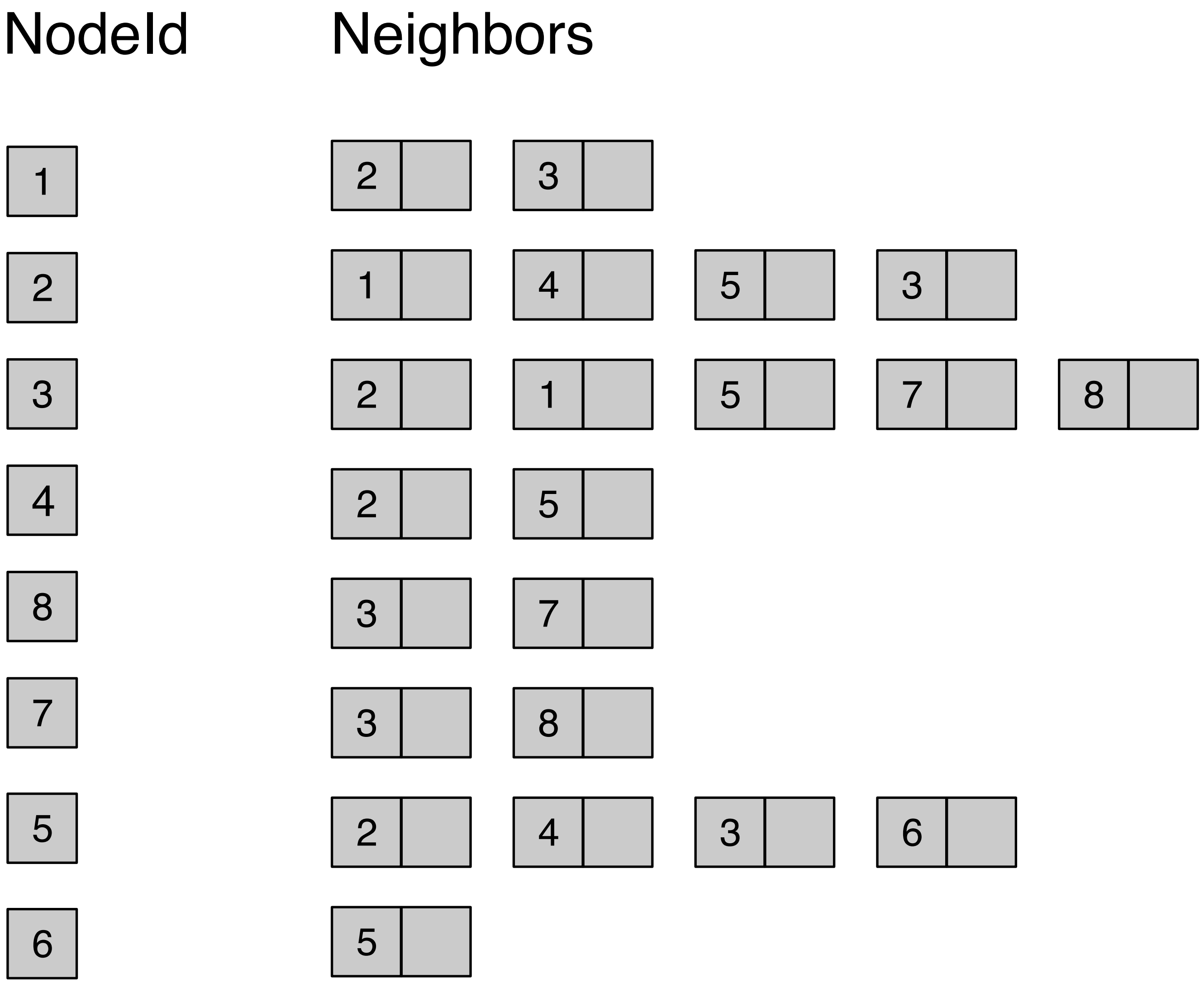
Let A be an adjacency list. What is the total number of keys stored in A?

A. $\Theta(|V|)$ C. $\Theta(|E|)$ E. $\Theta(|V|+|E|)$

B. $\Theta(|V|^2)$ D. $\Theta(\max\{degree(v)\})$

Graph representation: adjacency lists





NodeId	Neighbors				
1	2	3			
2	1	4	5	3	
3	2	1	5	7	8
4	2	5			
8	3	7			
7	3	8			
5	2	4	3	6	
6	5				

space complexity	$\Theta(\lvert V\rvert + \lvert E\rvert)$
Operation	**time complexity**
check edge (u,v)	$\Theta(\delta(u))$
find all neighbors of u	$\Theta(\delta(u))$
find all edges	$\Theta(\lvert V\rvert + \lvert E\rvert)$

Adjacency list – space complexity analysis

$|V| = n$

$|E| = m$

\# of nodes

\# of edges

Claim. The size of the adjacency list is $\Theta(n+m)$

Note. using *n+m* is the preferred notation. $\Theta(n+m) = \Theta(\max\{n,m\})$

$\Theta(n)$ = the size of list containing nodes id

$\Theta(m)$ = each edge in this data structure is represented as a node on the right side of the list of nodes. (counted twice)

We need at least $\Theta(n)$ to represent the nodes

We need at least $\Theta(m)$ to represent the edges

Thus, we need $\Theta(\max\{n,m\})$ to represent the adjency list $\Rightarrow \Theta(\max(n,m)) = \Theta(n+m)$

Adjacency list – space complexity analysis

Claim. The size of the adjacency list is $\Theta(n+m)$

Note. using *n+m* is the preferred notation. $\Theta(n+m) = \Theta(\max\{n,m\})$

proof 1:

$\Theta(n)$: there is a slot in the main list for every vertex v even if their degree is 0

$\Theta(m)$: there is a list of size $\delta(v)$ for every v in the main list

- For each v the size of the list containing v's neighbors is degree(v)
- We know that each edge (v,w) contributes once to the degree of v and once to w

$$\sum_{v\in V} degree(v) = 2m = \Theta(m)$$

proof 2:

$\Theta(n)$: there is a slot in the main list for every vertex v even if their degree is 0

$\Theta(m)$: every neighbor of v is represented on the right side of the main list

- Each edge (v,w) is represented in the table twice; w is in the neighbor list of w and vice verse.

Exercise. Repeat the same proof on directed graphs and the directed adj list.

n^2 vs. m in graphs

Why is the use of adjacency lists, with $\Theta(n+m)$ complexity considered better than the $\Theta(n^2)$ of the adjacency matrix?

Intuitivly most of the time a graph contains more edges then vertices. So, why use $\Theta(n+m)$ instead $\Theta(n^2)$?

$$\Theta(n^2) = \frac{n(n-1)}{2} \leq c \cdot n^2$$ (Because every edge is counted twice)

What is the minimum amount of edges in a conected graph?

$$n-1 \leq m \leq n^2$$

Exercise. Compute the minimum number of edges in a connected graph on n vertices.

Computing the running time in loops

A is the adjacency matrix of a graph, G is the adjacency list of the same graph.

```
**Algorithm 1:** MatrixCount($A$)
$count \leftarrow 0$;
for $i = 1$ to $n$ do
    for $j = 1$ to $n$ do
        if $A[i, j] == 1$ then
            $count += 1$;
return $count/2$
```

```
**Algorithm 2:** ListCount($G$)
$count \leftarrow 0$;
for $v$ in $G$ do
    for $w$ in $G[v]$ do
        $count += 1$;
return $count/2$
```

Same algorithm, but using different data structures.

Both return the number of edges

Computing the running time in loops - TopHat

A is the adjacency matrix of a graph, G is the adjacency list of the same graph.

Algorithm 1: MatrixCount(A)

```
count ← 0;
for $i = 1$ to $n$ do                -> n·times
    for $j = 1$ to $n$ do            -> n times
        if $A[i, j] == 1$ then       } Θ(1)
            $count += 1$;
return $count/2$
```

Algorithm 2: ListCount(G)

```
count ← 0;
for $v$ in $G$ do                    -> n times
    for $w$ in $G[v]$ do
        $count += 1$;
return $count/2$
```

the length of the loop depends on the amount of neighbors of v

(multiple choice!)

What is the running time of MatrixCount?

A. $\Theta(n)$

[✓ B. $\Theta(n^2)$]

What is the running time of ListCount?

[✓ C. $\Theta(n+m)$]

D. $\Theta(nm)$

E. $\Theta(n \cdot deg_{max})$

Computing the running time in loops

A is the adjacency matrix of a graph, G is the adjacency list of the same graph.

Algorithm 1: MatrixCount(A)

```
$count \leftarrow 0$;
for $i = 1$ to $n$ do
    for $j = 1$ to $n$ do
        if $A[i, j] == 1$ then
            $count += 1$;
return $count/2$
```

If the number of iterations in the inner loop is *fixed*, then we can multiply to get the number of iterations.:

n x n iterations, O(1) each results in O(n^2)

Algorithm 2: ListCount(G)

```
$count \leftarrow 0$;
for $v$ in $G$ do
    for $w$ in $G[v]$ do
        $count += 1$;
return $count/2$
```

If the number of iterations is *different* each time then compute the total number of operations across the iterations.:

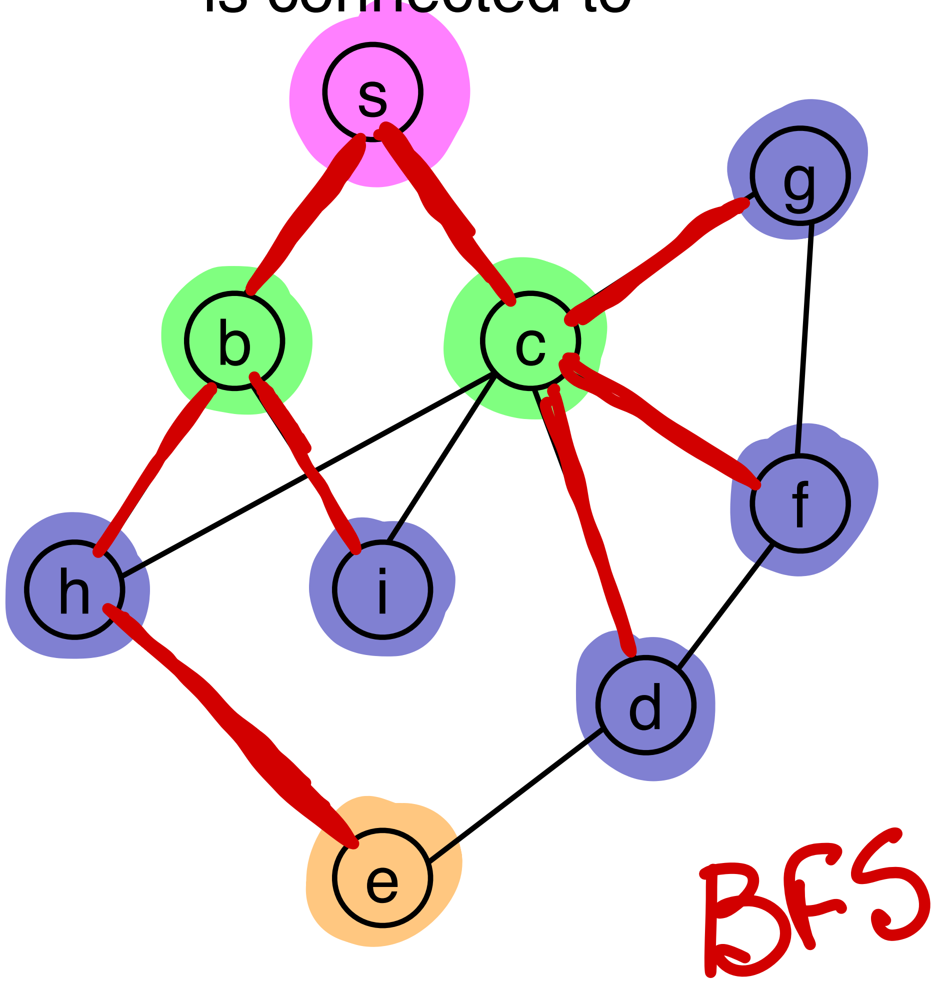
- the outer loop visits each node ones: O(n)
- the inner loop iterates over each edge in G, the counter is increased once for each edge: O(m)
- total: O(n+m)

Breadth First Search (BFS)

🟪 0 steps
🟩 1 step
🟦 2 steps
🟧 3 steps

Breadth first search.

- Task: Given a source node s, find the *shortest* path from s to each node v that it is connected to



1. explore directed neighbors of s
2 - explore all neighbors of b and c
3 - explore all neighbors of h, i, d, f, and g.

BFS Tree (represented by red edges)

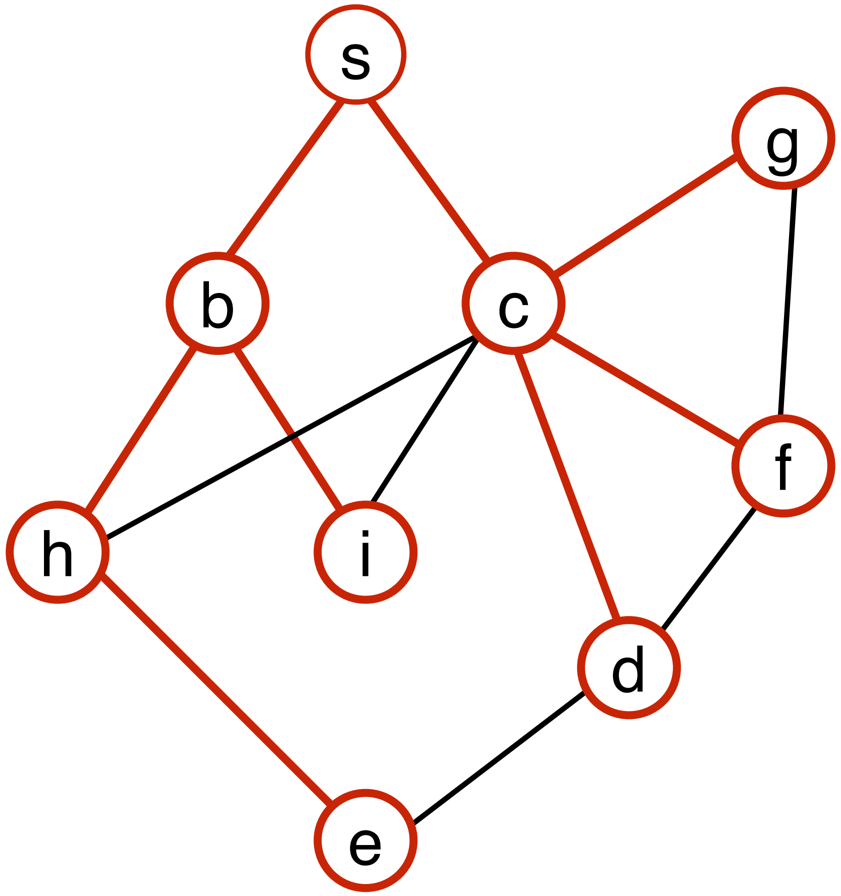
- Contains all nodes of the original graph
- has the minium amount of edges

shortest path between nodes u and v = the connecting path with the least edges

BFS

Breadth first search.

- Task: Given a source node s, find the *shortest* path from s to each node v that it is connected to



First iteration:

start from s.

Second iteration:

find all neighbors of s. first layer

Third iteration:

find all (so far undiscovered) neighbors of nodes in first layer. second layer

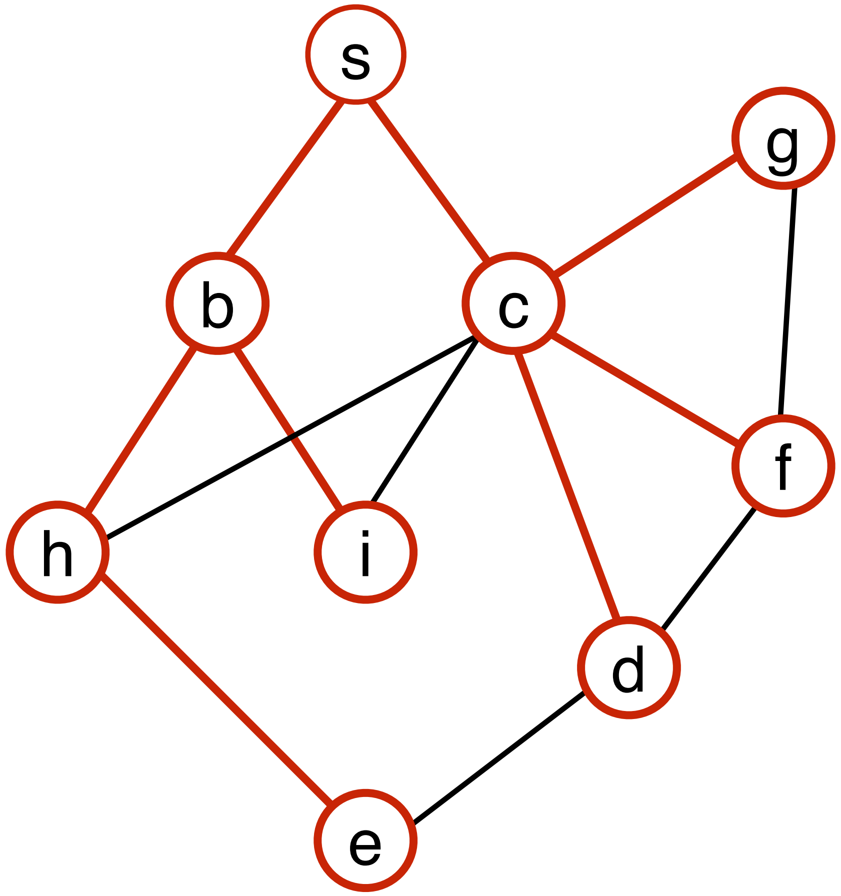
Fourth iteration:

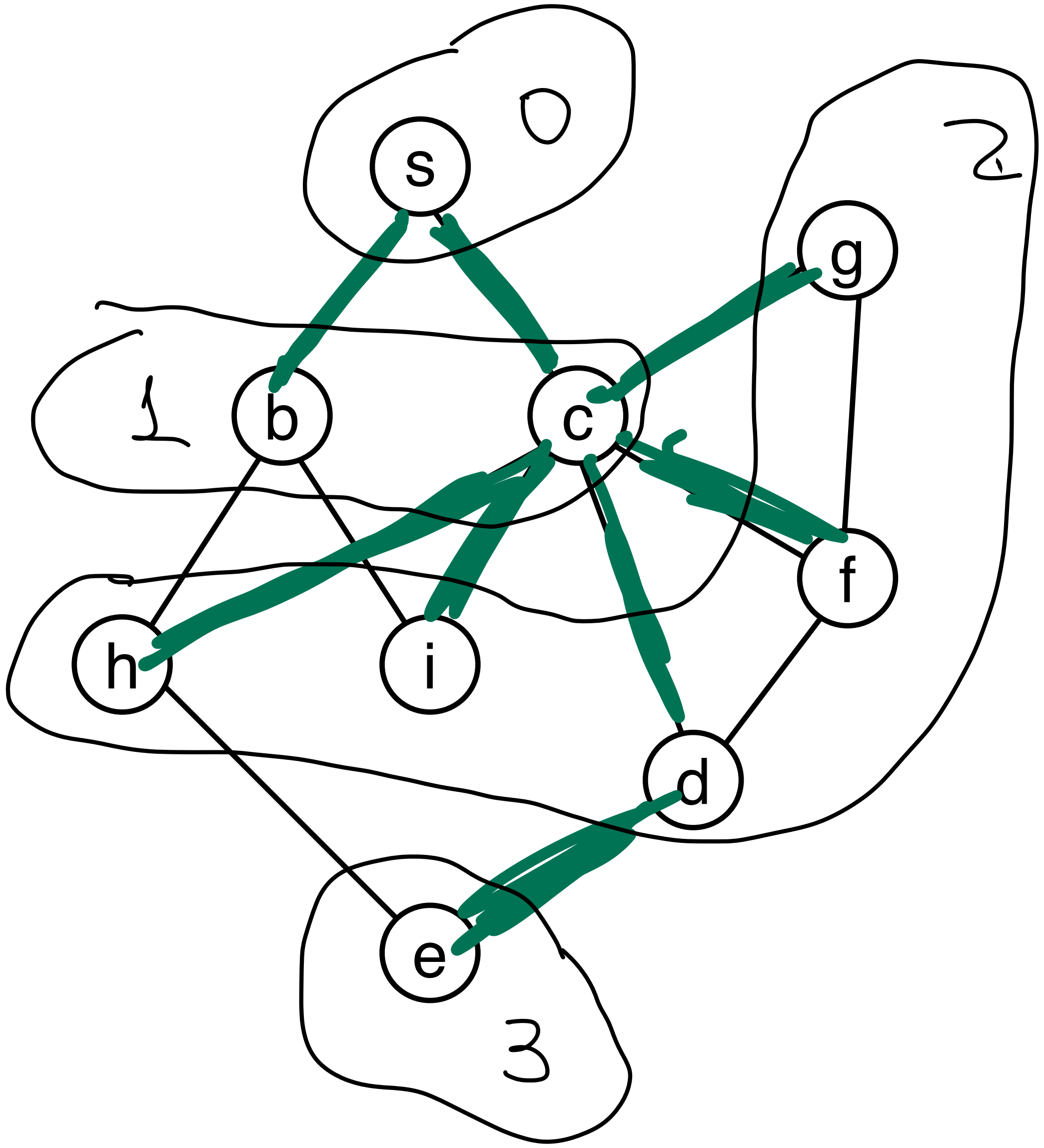
find all neighbors of the second layer. third layer

BFS search tree: an edge connecting each node to the neighbor through which it was discovered.

Breadth first search – order of node processing

Green: alternative order of node processing.





BFS search tree