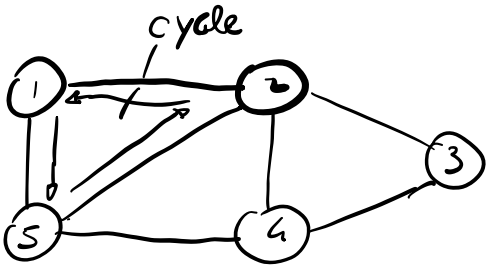
Graphs Cormen 22.1 22.2 22.3 (no proofs)

$G = (V, E)$ $|V| = n$ $|E| = m$

V = set of nodes

E = set of edges



← undirected

$V = \{1, \ldots, n\}$

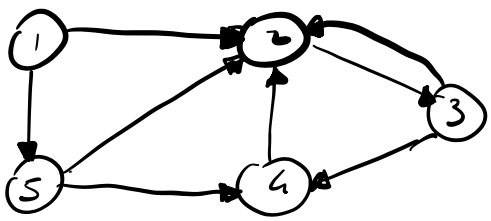
$E \subseteq V \times V$

(u, v)

directed graph

$V = \{1, \ldots, 5\}$

$E = \{(1,2), (2,3), (1,5), (2,4), (2,5), (3,4), (4,5)\}$



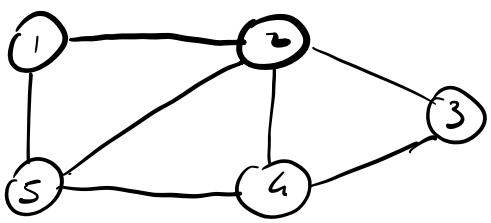
Representations of Graphs

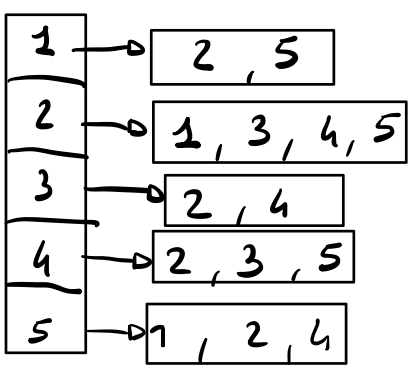
① A list of edges (i.e. pairs of nodes)

any op takes $\Theta(|E|)$ time

A graph is either { **sparse** $|E|$ is much less than $|V|^2$ eg $|E| = \Theta(n)$; **dense** $|E|$ is close to $|V|^2$ eg $|E| = \Theta(n^2)$

Adjacency List representation





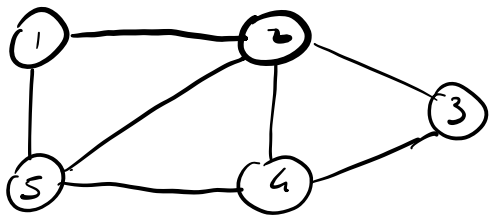
we store $2|E|$ numbers as every edge is represented twice in a undirected graph

Overall space usage $\Theta(|E| \log n)$ bits

Adjacency Matrix

Store a binary matrix M of size $|V| \times |V|$

$M[u,v] = 1$ iff $(u,v) \in E$, 0 otherwise
(or $(v,u) \in E$ for undirected)



M	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

If G is undirected, M is symetric

Space usage is $|M| = |V|^2$ bits
$= \Theta(n^2)$

Adj lists vs Adj Matrix

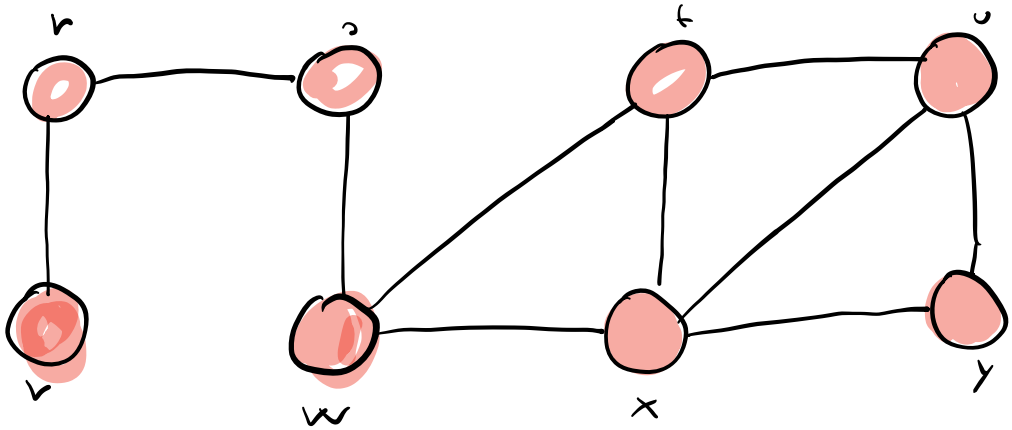
Space $\Theta(|E| \log |V|)$ vs $\Theta(|V|^2)$ bits

first one is better for sparse graphs

Operations	Adj Lists	Adj Matrix
$(u,v) \in E$?	$\Theta(\log(d(u)))$ time (binary search)	$O(1)$
	($d(u)$ = degree of u = size of u's list)	
	(or $O(1)$ expected hashing)	
List neighbours of u	$\Theta(d(u))$	$\Theta(\|V\|)$

Breadth - first Search (BFS)

We start from vertex s (called source)
BFS discovers every vertex v reachable from s $s \rightsquigarrow v$ (find the shortest path)
path



we need a Queue to implement this strategy

BFS(G, s)

1 for each vertex $u \in V \setminus \{s\}$
2 $u.color = white$ ← white: undiscovered; gray: discovered but unvisited; black: both discovered and visited
3 $u.d = +\infty$ // shortest path distance from s
4 $u.\pi = NIL$ // $u.\pi$ is the parent of u in the BFS tree
5 $s.color = gray$
6 $s.d = 0$
7 $s.\pi = NIL$

```
8   Q = ∅
9   ENQUEUE(Q, s)
10  while Q ≠ ∅
11      u = DEQUEUE(Q)  // pop head
12      for v in G.adj[u]
13          if v.color == white
14              v.color = gray
15              v.d = u.d + 1
16              v.π = u
17              ENQUEUE(Q, v)
18      u.color = black
```

time complexity

lines 1 - 4 $\Theta(|V|)$ time

lines 5 - 9 $O(1)$ time

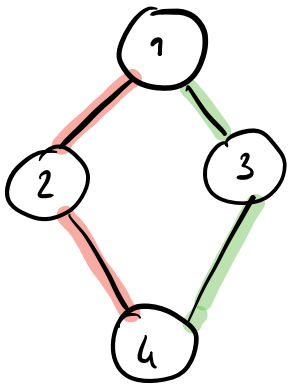
lines 10 - 18

- every node u is visited at most once
- cost of visiting u

$\Theta\left(\sum_{u \in V} (1 + d(u))\right)$ time

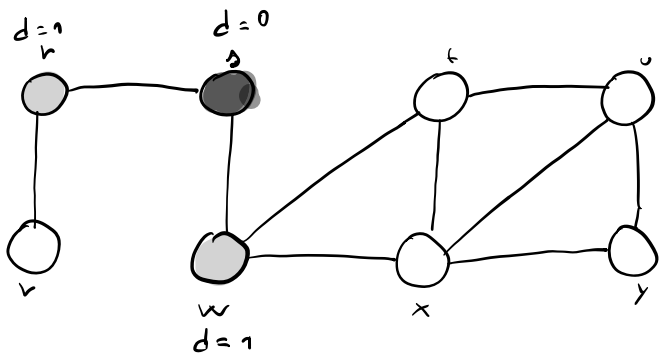
$$\Theta(|V|) + \Theta\left(\sum_{v \in V} (1 + d(v))\right) \text{ time}$$

$$= \Theta(|V| + |E|) \text{ time}$$

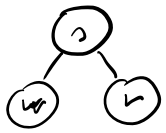


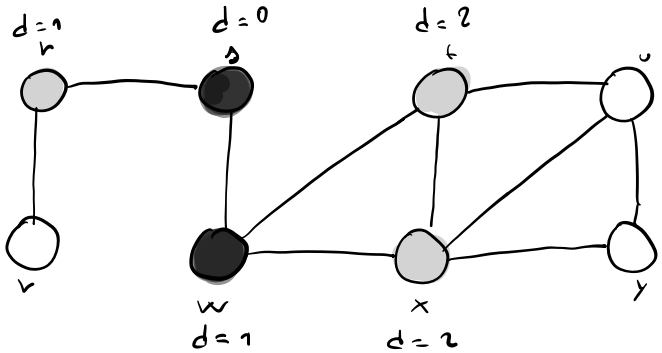
Shortest path from (1) to (4) has length 2

there are 2 paths

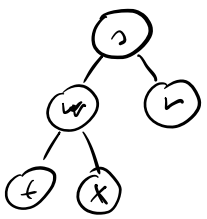


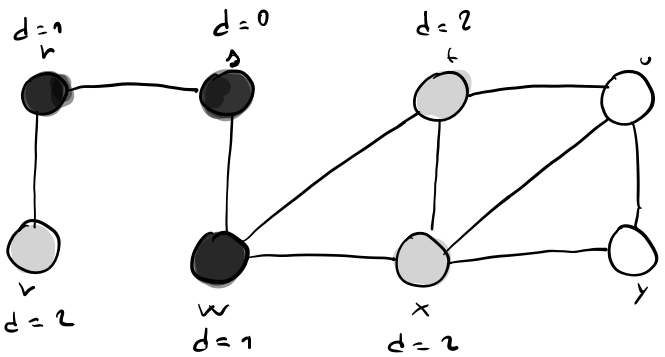
$Q = s\ w\ r$



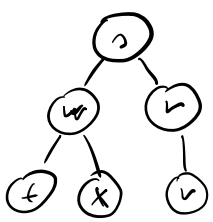


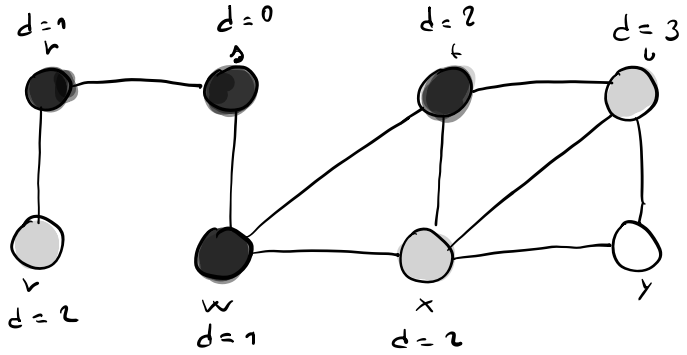
$Q = s\ w\ r\ t\ x$



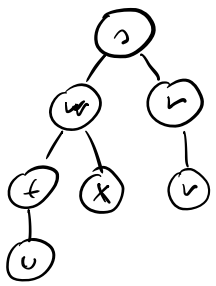


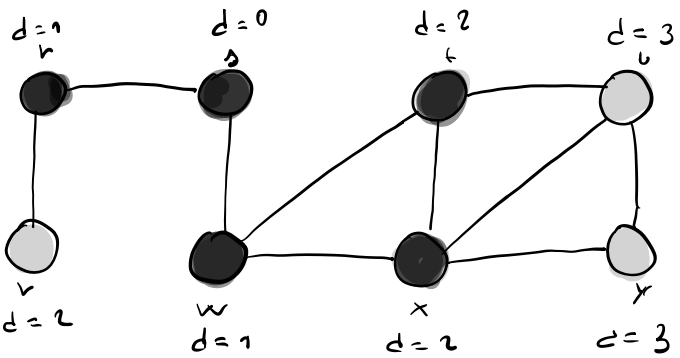
$Q = s\ w\ r\ t\ x\ v$



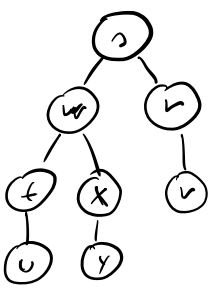


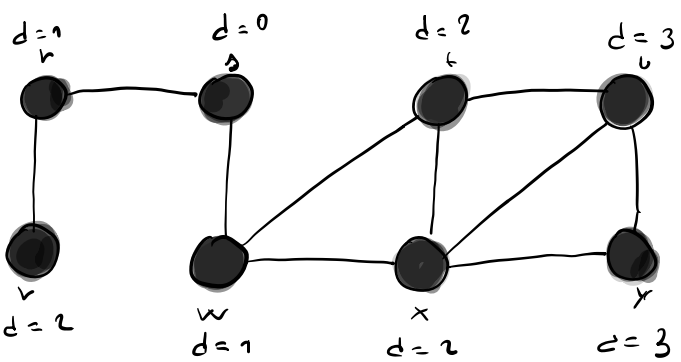
Q = s w r t x v u



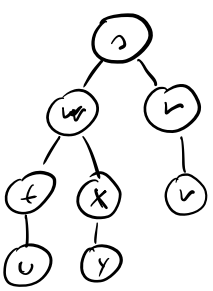


Q = s w r t x v u y

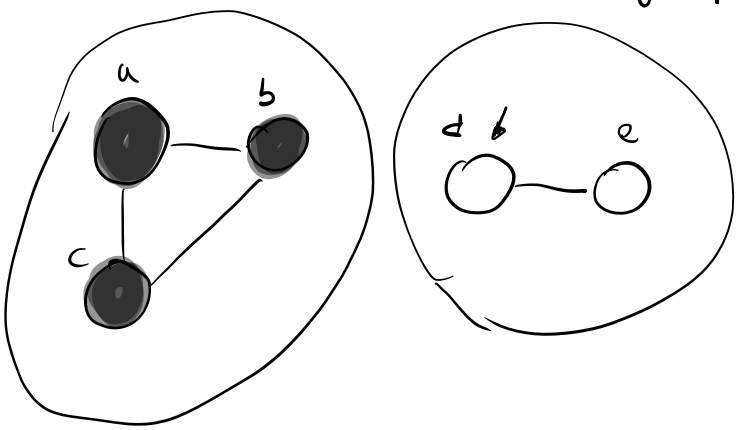




Q = s w r t x v u y



Is G connected? (undirected graphs only)



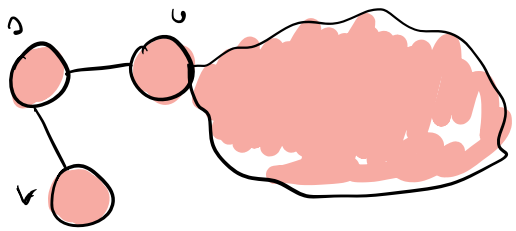
Cost of discovering all the components is

$$\Theta(d(|V| + |E|)) \quad \text{or} \quad \Theta(|V| + |E|)$$

where d is # of components

Depth-first search (DFS)

Opposite strategy: go deeper in the graph

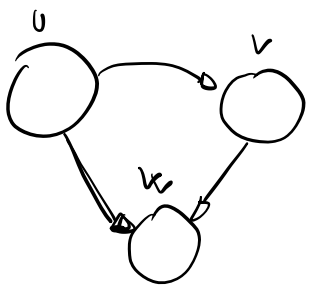


Just replace the queue with a stack

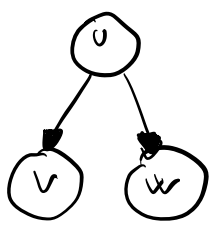
- For applications the starting source is not important
- we build a forest of trees and every node is in exactly one tree

```
DFS(G)
1  for each vertex u ∈ V
2      u.color = white
3      u.π = NIL
4  time = 0    // remember the time at which we start or end a visit
5  for each vertex u ∈ V
6      if u.color == white
7          DFS-visit(G, u)
```

```
DFS-visit (G, u)
1  time = time + 1          ← u's visit starts
2  u.d = time               // remember starting time
3  u.color = gray
4  for each v ∈ G.Adj[u]
5        if v.color == White
6              v.π = u
7              DFS-visit (G, v)
8  u.color = black
9  time = time + 1
10 u.f = time               // ending time
```

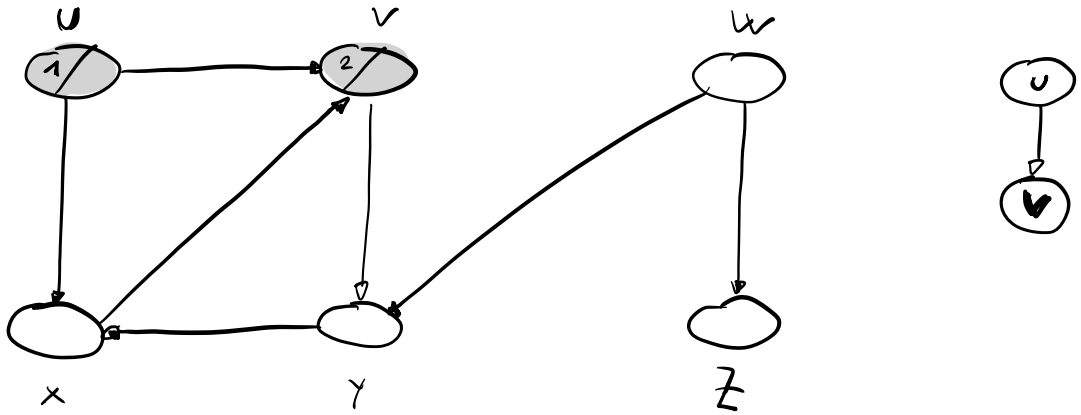


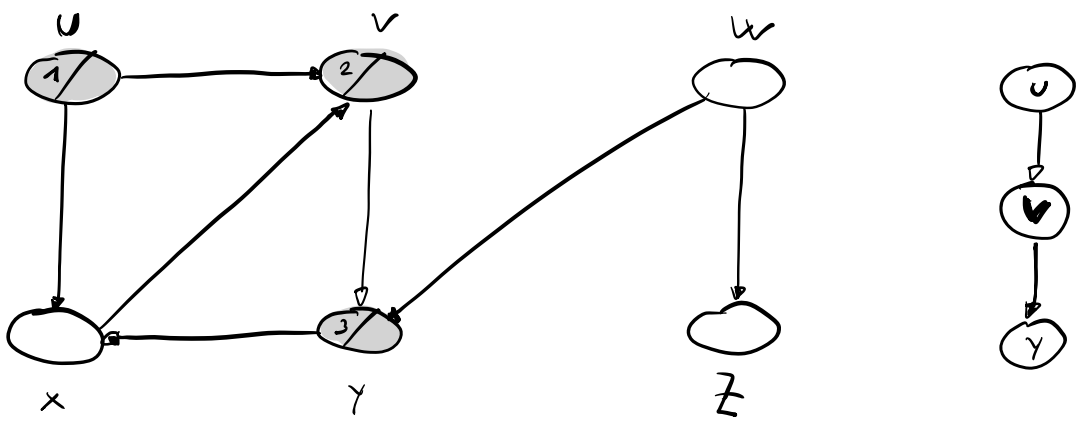
BFS

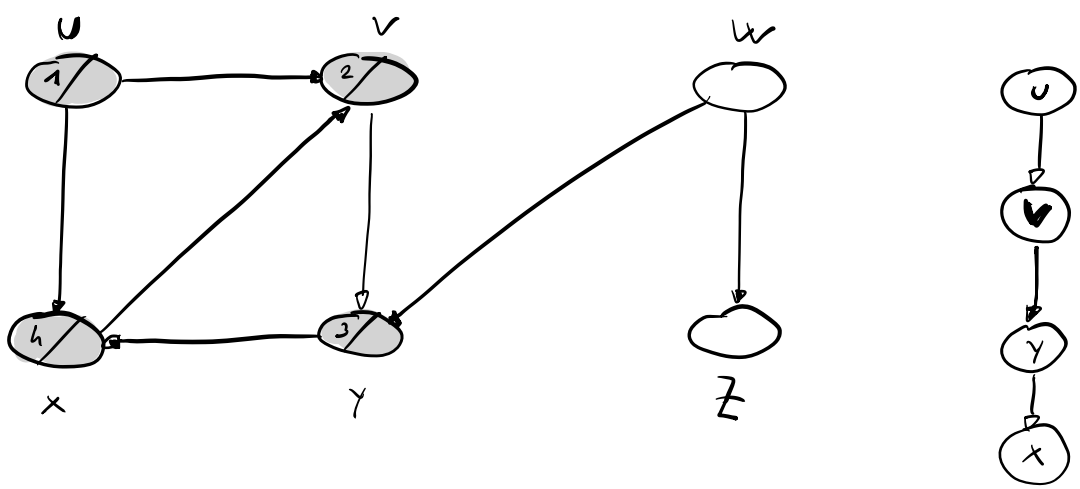


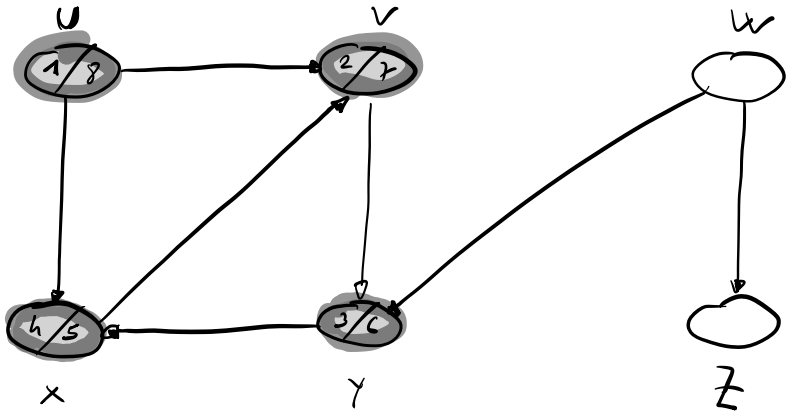
DFS

Complexity $\Theta(|V| + |E|)$ time

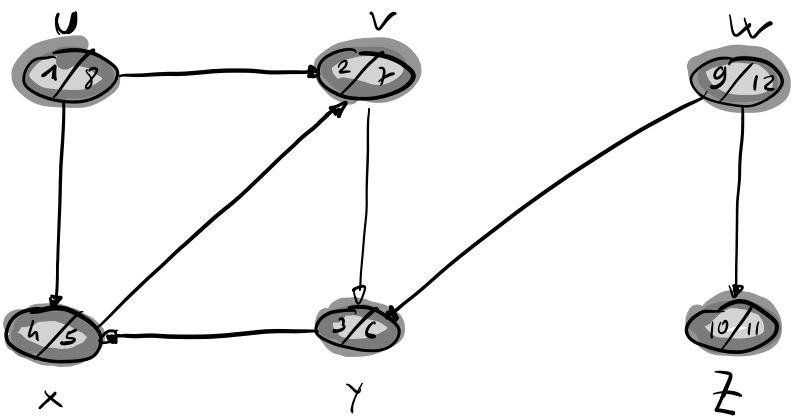

u
v
w
1
2
x
y
z
u
v

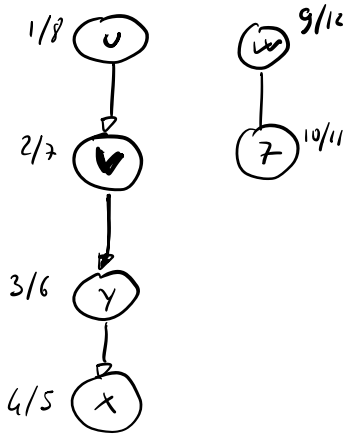

u
v
w
1
2
3
x
y
z
u
v
y


u
v
w
1
2
4
3
x
y
z
u
v
y
x


u
v
w
1/8
2/7
4/5
3/6
x
y
z


u
v
y
x


u
v
w
1/8
2/7
9/12
4/5
3/6
10/11
x
y
z


1/8
u
9/12
w
2/7
v
10/11
z
3/6
y
4/5
x