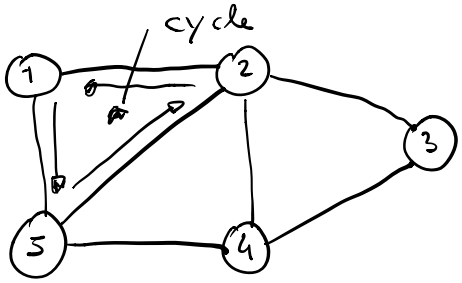
Graphs

Cormen 22.1 22.2 22.3 (no proofs)

$G = (V, E)$ $\quad |V| = n$

V = set of nodes

E = set of edges



$V = \{1, \ldots, n\}$ $\quad V = \{1, 2, 3, 4, 5\}$

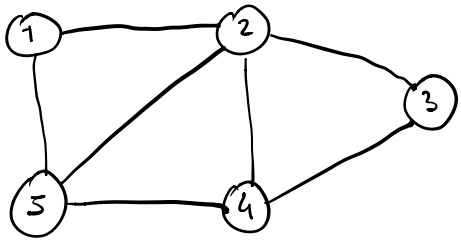
$E \subseteq V \times V$ $\quad E = \{(1,2), (2,3), (1,5), (2,4), (2,5), (3,4), (4,5)\}$

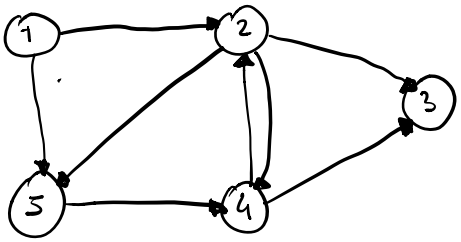
$(u, v) \in E$

edges may have a direction.

Direct Graphs

Undirected graph





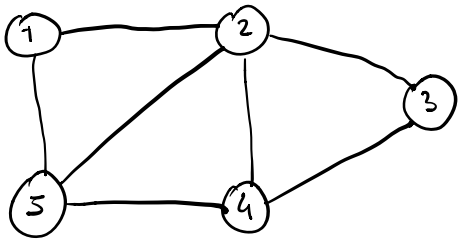
Representations of Graphs

0. List of edges, each edge is a pair (u, v)
   any op. takes $\Theta(|E|)$ time

A graph is either
- sparse — $|E|$ is much less $|V|^2$ eg $|E| = \Theta(n)$
- dense — $|E|$ is close to $|V|^2$ eg $|E| = \Theta(n^2)$

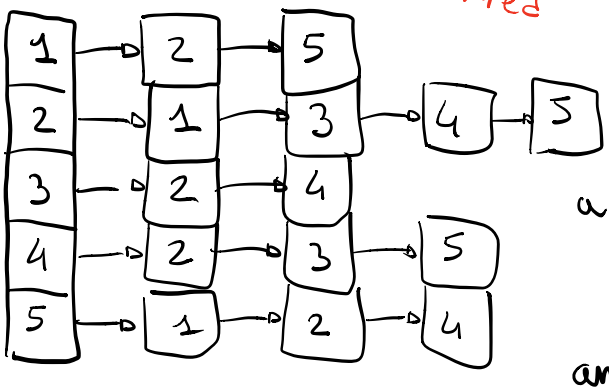
Two possible representations
- Adj List
- Adj Matrix

Adjacency Lists representation



For any node u, store all node adjacent to u,
i.e. undirected
any v, $(u, v) \in E$ or $(v, u) \in E$

Store lists sorted



directed
any v, $(u, v) \in E$

any edge outgoing from u

every edge is represented twice in undirected graphs

Total number of elements in our rep.

$$\sum_{v \in V} d(v) = \begin{cases} |E| & \text{directed graphs} \\ 2|E| & \text{undirected graphs} \end{cases}$$

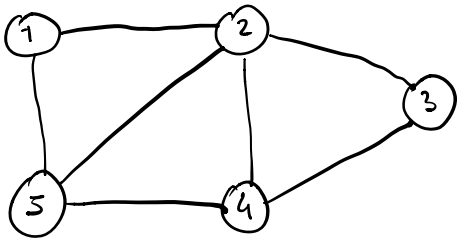
$d(v)$: degree of v — number of outgoing (from v) edges

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

Adjancy 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 graphs)



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

Adj lists and Adj Matrix

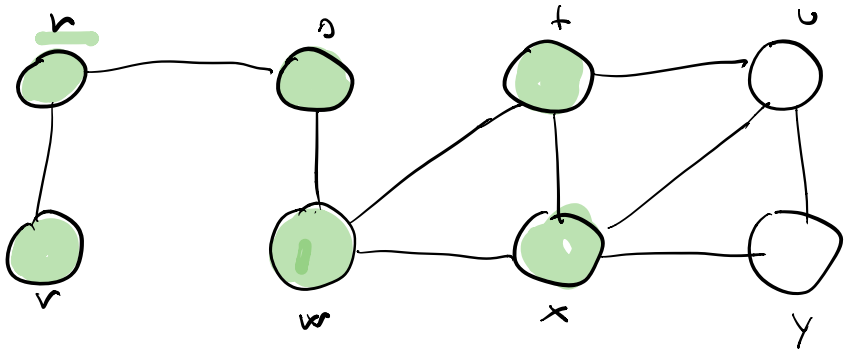
Space $\Theta((|E|\log|V|)$ bits vs $\Theta(|V|^2)$ bits

Adj Matrix is better only for very dense graphs where $|E| = \Omega\left(\frac{|V|^2}{\log|V|}\right)$

Operations	Adj List	Adj Matrix
$(u,v) \in E$?	$\Theta(\log d(u))$ bin search (or $O(1)$ expected with Hashing)	$O(1)$ WC
List neighbours of u	$\Theta(d(u))$	$\Theta(\|V\|)$

Breadth-first Search (BFS)

We start from a vertex s, (called source)
BFS discover every vertex v reachable from s $\xrightarrow[\text{path}]{}$ v (find the shortest path)



We need a QUEUE FIFO
to implement the strategy

```
BFS(G, s)
1  for each vertex u ∈ V \ {s}
2      u.color = white      / white: undiscovered
                            / gray: discovered but unvisited
                            / black: both discovered and visited
3      u.d = ∞              / shortest path distance from s
4      u.π = NIL            / we build a shortest path tree from s to any other vertex
                            / u.π is the parent of u in the tree
5  s.color = gray
6  s.d = 0
7  s.π = NIL
```

```
8  Q = ∅
9  ENQUEUE(Q, s)
10 while Q ≠ ∅
11         u = DEQUEUE(Q) // pop head of Q
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 OF BFS

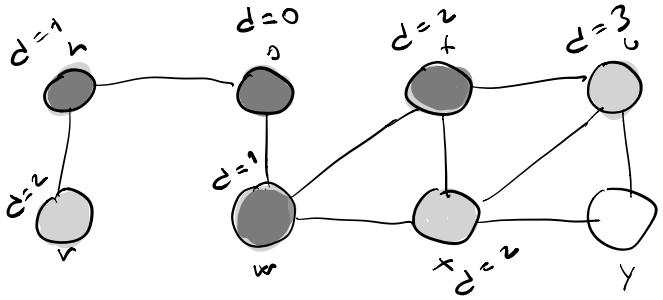
visit of node u costs $\Theta(1 + d(u))$

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

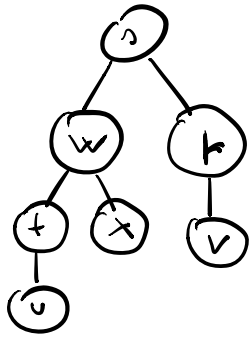
Q w r

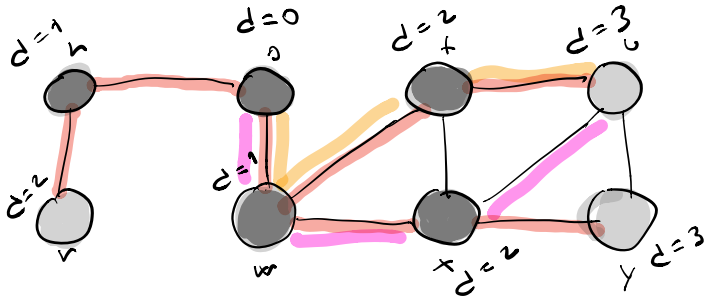
Q ~~w~~ r t x

Q ~~w~~ ~~r~~ t x v

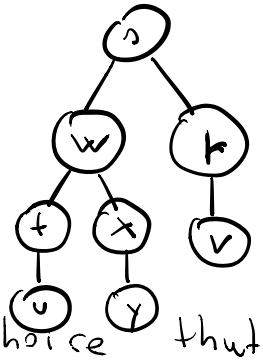


Q ~~w~~ ~~r~~ ~~t~~ x v u

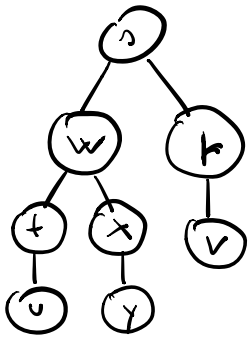


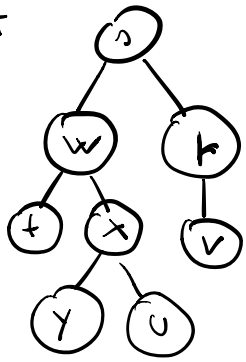


Q ~~w~~ ~~r~~ ~~t~~ ~~x~~ v u y



Question: find a different choice that changes the tree before t





Depth-first search (DFS)

opposite strategy: go deeper in the graph

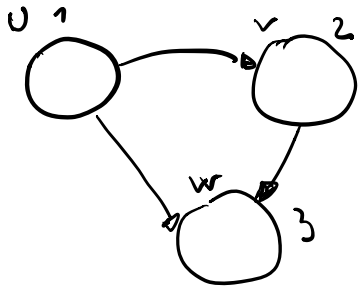
just replace the queue with a stack

- for applications, the source from which we start is not so important
- we build a forest of trees, not just a tree

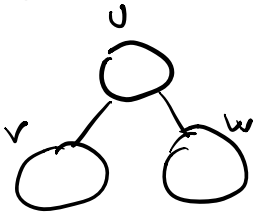
```
DFS(G)
1  for each vertex u ∈ V
2      u.color = white
3      u.π = NIL
4  time = 0          // remember time we start visiting or finish the visit of a vertex
5  for each vertex u ∈ V
6      if u.color == white
7          DFS-visit(G, u)
```

```
DFS-Visit(G, u)
1  time = time + 1   // white vertex u is just discovered
2  u.d = time        // starting time
3  u.color = gray
4  for each v ∈ G.Adj[u]
5      if v.color == white
6          v.π = u
```

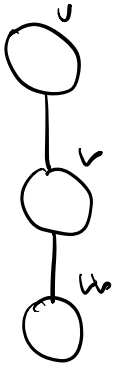
```
7                DFS-Visit ( G, v)
8   u. color = BLACK
9   time = time + 1
10  u. f = time     // ending time visit of u
```



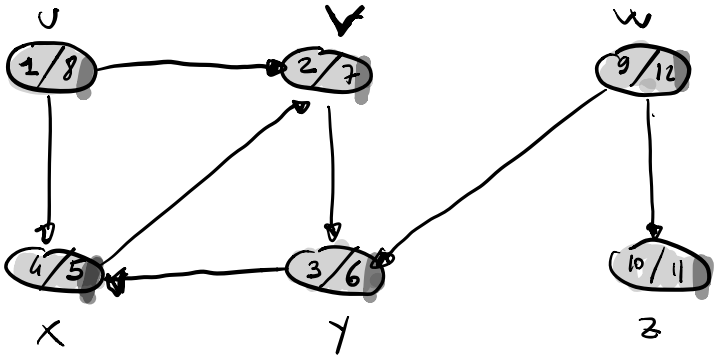
BFS

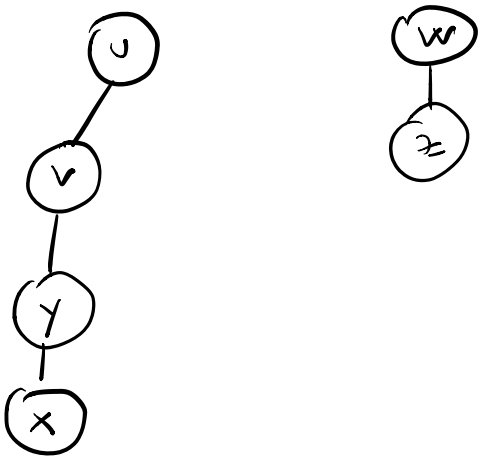


DFS



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




u
v
y
x
w
z