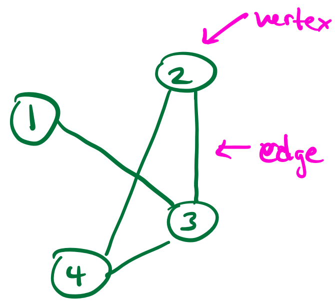
Graphs

Terminology

V - vertices (nodes)

E - edges

G - graph = (V, E)



Undirected graph: edges are bidirectional

→ Degree: number of edges touching a node

Directed graph: edges are unidirectional

→ In-degree: number of edges <u>to</u> a node

→ Out-degree: number of edges <u>from</u> a node

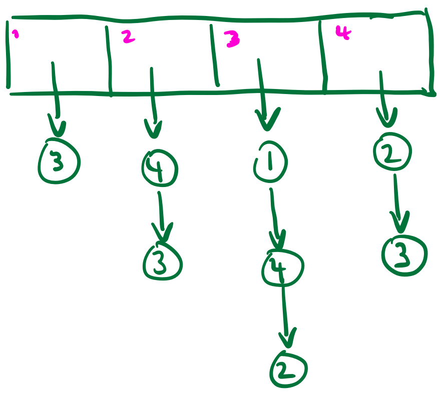
Representation

Adjacency Matrix

$$
\begin{array}{cc}
 & \text{dst} \\
 & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\
\text{src} \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}
\end{array}
$$

(does not need to be symmetric for directed graphs)

Adjacency lists (using array or linked list design)



(also doesn't need to be symmetric)

Note: vertices can store other info

(i.e., attributes - name, IP addr, etc)

+ helper info for graph algs

Goal: perform operations

→ Checking for edge membership

→ querying for node neighbors

Tradeoffs:

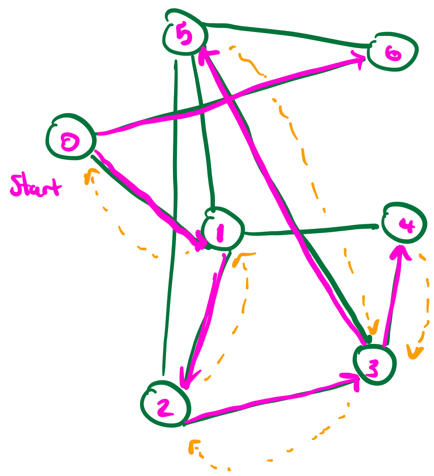
	Adj. mat.	Adj. list	
Edge membership	$O(1)$	$O(\lvert V \rvert)$	← $O(deg(x))$ for a node x
Neighbor query	$O(\lvert V \rvert)$	$O(1)$	← $O(deg(x))$ if need to iterate
Space requirements	$O(\lvert V \rvert^2)$	$O(\lvert V \rvert + \lvert E \rvert)$	

Depth-First Search (DFS)

Generic algorithm for graph traversal.

Idea: explore graph w/ "string and chalk"

Example



Pseudocode

```
Algorithm DFS(v)
    v.visited = True
    do something with v   ← generalizable!
    for w in v.neighbors:
        if !v.visited:
            DFS(w)
```

note: can also do this iteratively with a stack

(more efficient for larger data)

Runtime

→ visit every vertex once, doing $O(1)$ work at each step

→ loop over neighbors and check visited in $O(1)$: $O(\deg(v))$

→ $\sum_{v \in V} (O(\deg(v)) + O(1)) = O(|V| + |E|) = O(|E|)$

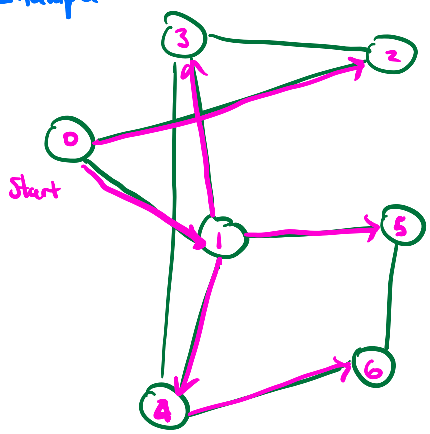
($|V| \leq |E| + 1$ for any graph)

Breadth-First Search (DFS)

Explore graphs in order of steps to reach each node

Main application: finding shortest paths.

Example



Pseudocode

```
Algorithm BFS (V, E)
    Let Q be a queue
    Q.push(v)    // start node
    while !Q.empty:
        w = Q.pop()
        Mark w as visited
        do something with w
        for x in w.neighbors:
            if !x.visited:
                Q.push(x)
```

Runtime

For finding shortest path, $O(|E|)$