A graph \( G = (V, E) \) is a set of vertices \( V \) and the set of edges \( E \). If \( G \) is undirected, \( E \) consists of pairs of vertices, unordered. o.w. \( G \) is directed, and \( E \subseteq V \times V \).

If \( uv \in E \), \( u \) and \( v \) are adjacent or neighbors.
For $u \in V$,

Degree of $u$ is $\# \text{neighbors}$.

If $G$ is directed,

- in-degree: $\# \text{edges} \ x \rightarrow u$
- out-degree: $\# \text{edges} \ w \rightarrow y$

For graph algorithms,

$V$ or $E$ might mean $|V|$ or $|E|$

i.e. $O(V + E)$
Representations Adjacency matrix.

$|V|\times|V|$ 2D array/matrix

$A[i,j]=1$ if edge $ij \in E$.

$A[i,j]=0$ otherwise.

$\Theta(1)$ time to check if an edge in $E$

$\Theta(V^2)$ space always

$\Theta(V)$ time to find all neighbors of a vertex
Adjacency list:

An array of length |V|.

Each entry points to a list of adjacent vertices of the entry’s vertex.

If G is undirected, each edge appears twice.

uv is v in u’s list.

If G is directed, u→v appears as v in u’s list only.
\( \Theta (V + E) \) space

\( \Theta (\text{degree}(u)) \) time to list neighbors of \( u \)

\( \Theta (\min\{\text{degree}(u), \text{degree}(v)\}) \) to check if \( uv \) exists

Assume adjacency list unless told otherwise.
A walk is a sequence of edges s.t. each successive pair share a vertex.

![Diagram](image)

It is a path if it repeats no vertices.

A cycle is a path except we do repeat exactly the first & last vertex.
A undirected graph is connected if there is a path from every vertex to every other vertex.

Problem: Given graph $G$ and a vertex $s$. Also given $v$, is $v$ reachable from $s$? i.e. does there exist a path from $s$ to $v$?
Breadth-first search (BFS) :

\[ \text{BFS}(s) : \]

1. put \((\emptyset, s)\) in a queue
2. while queue is not empty:
   1. take \((p, v)\) from queue
   2. if \(v\) is unmarked:
      1. mark \(v\)
      2. parent \((v) \leftarrow p\)
      3. for each edge \(vw\):
         1. put \((v, w)\) in queue

Facts:
1) Marks every vertex reachable from \(s\) exactly once.
2) Edges of the form $\text{parent}(v) v$ form a spanning tree on the component of $G$ containing $s$.

Subgraph $H$ of $G$: Has a subset of $G$'s vertices and edges.

A component of $G$ is a maximal connected subgraph.

A spanning tree is a connected acyclic subgraph containing every vertex with no cycles.
3) The tree contains the shortest path from `s` to every reachable vertex.

Running time: \(O(V + E)\)

(Faster if `s`'s component is small.)

Please use BFS for shortest paths with unit (1) edge weights.
Depth-First Search (DFS):

\[
\text{DFS}(v):
\begin{align*}
&\text{mark } v \\
&\text{PREVISIT}(v) \\
&\text{for each edge } vw \\
&\quad \text{if } w \text{ is unmarked} \\
&\quad \quad \text{parent}(w) \leftarrow v \\
&\quad \text{DFS}(w) \\
&\text{POSTVISIT}(v)
\end{align*}
\]

\[
\text{DFSALL}(G):
\begin{align*}
&\text{PREPROCESS}(G) \\
&\text{for all vertices } v \\
&\quad \text{unmark } v \\
&\text{for all vertices } v \\
&\quad \text{if } v \text{ is unmarked} \\
&\quad \quad \text{DFS}(v)
\end{align*}
\]

\[O(V + E) \text{ Time}\]
Imagine we pass around a "clock" to time events...

* v.pre: starting time of v
* v.post: finishing time
* [v.pre, v.post]: active interval of v

Either two active intervals are disjoint or one contains the other.

**DFSAll(G):**
- $\text{clock} \leftarrow 0$
- for all vertices $v$
  - unmark $v$
- for all vertices $v$
  - if $v$ is unmarked
    - $\text{clock} \leftarrow \text{DFS}(v, \text{clock})$

**DFS(v, clock):**
- mark $v$
- $\text{clock} \leftarrow \text{clock} + 1; \ v.\text{pre} \leftarrow \text{clock}$
- for each edge $v \rightarrow w$
  - if $w$ is unmarked
    - $w.\text{parent} \leftarrow v$
    - $\text{clock} \leftarrow \text{DFS}(w, \text{clock})$
- $\text{clock} \leftarrow \text{clock} + 1; \ v.\text{post} \leftarrow \text{clock}$
- return $\text{clock}$
\[ \{ v \text{.pre}, v \text{.post} \} \subseteq \{ u \text{.pre}, u \text{.post} \} \quad \text{iff} \quad \text{DFS}(u) \text{ (indirectly)} \text{ calls } \text{DFS}(v) \]
implies \( u \) can reach \( v \).

Sort by \( x \text{.pre} \) to get a \underline{preorder}.
Sort by \( x \text{.post} \) to get a \underline{postorder}. 
Say we run DFSAll...

Fix a vertex $v$ and its (future) $v$, pre + $v$, post values.

Consider any moment in the algorithm.

$v$ is **new** if $\text{clock} < v$.pre

$v$ is **active** if $v$.pre $\leq$ $\text{clock}$ $< v$.post

$v$ is **finished** if $v$.post $\leq$ $\text{clock}$

Consider an edge $u$ $\to$ $v$ at the moment $\text{DFS}$(u) begins.
If $v$ is new, a recursive call will mark $v$.

$u$.pre < $v$.pre < $v$.post < $u$.post

$u \geq v$ is a tree edge if

$\text{DFS}(u)$ calls $\text{DFS}(v)$ directly

$u \geq v$ is a forward edge o.w.

If $v$ is active,

$v$.pre < $u$.pre < $u$.post < $v$.post

$u \geq v$ is a back edge

If $v$ is finished,

$v$.post < $u$.pre < $u$.post

$u \geq v$ a cross edge
Thm (for next time):

Graph $G$ has a directed cycle iff $DFSAII(G)$ yield at least one back edge.