Graph Representations:

**Matrix**: |V| = n. Takes n^2 space, so want a matrix when |E| \leq n^2.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Adjacency List**: n = |V| and m = |E|, m \leq n^2. Takes O(n+m) space.

- Not good for backtracking.

- When m < n, space can be saved using other data structures, such as a hash table.

Algorithms:

**Visit**: Walk a graph without visiting a vertex more than once.

\( \forall v \in V \):
- color\( [v] \) = white

\( \forall v \in V \):
- if color\( [v] \) = white then visit (\{v\})

**visit(L)**: Note: Can be DFS or BFS depending on starred lines.

while L ≠ ∅:
- *pick v ∈ L
  - if color\( [v] \) = white
    - color\( [v] \) = black
    - w = succ\( (v) \) n white
    - *L = L + w
**DFS of Visit:**

Do \( L = W \cdot L \) if pick chooses first element from \( L \).

Using a stack:
- \( \text{pick} = \text{pop}(L) \)
- \( L = \text{push}(W, L) \)

**BFS of Visit:**

Do \( L = L \cdot W \) if pick chooses first element from \( L \).

Using a queue:
- \( \text{pick} = \text{head}(L) \), note, must also remove the head of the queue.
- \( L = \text{enqueue}(L, W) \)

**BFS:**
- Finds shortest paths. Keep track of parents and it will keep an optimal path to the root from any node.

\[
\forall v \neq v_0 \\
\text{color}[v] = \text{white} \\
d[v] = \infty \\
\pi[v] = nil \\
\text{color}[v_0] = \text{gray} \\
d[v_0] = 0 \\
\pi[v_0] = nil \\
\text{BFS}(v_0)
\]

\[
\text{BFS}(L) \\
\text{while } L \neq 0 \\
\quad v = \text{head}(L) \\
\quad \forall u \in \text{succ}(v) \\
\quad \text{if color}[u] = \text{white} \\
\quad \quad \text{color}[u] = \text{gray} \\
\quad \quad d[u] = d[v] + 1 \\
\quad \quad \pi[u] = v \\
\quad \quad \text{enqueue}(L, u) \\
\quad \text{color}[v] = \text{black} \\
\]

\( O(n+m) \) using Adjacency List.
DFS:
\[ + = 0 \]
\[ + v \]
\[ \text{color}[v] = \text{white} \]
\[ \uparrow [v] = w + 1 \]
\[ + v \]
\[ \text{if color}[v] = \text{white} \]
\[ \text{DFS}(v) \]

\[
\begin{align*}
\text{DFS}(v) \\
\text{color}[v] &= \text{gray} \\
+ &= + + 1 \\
\text{d}[v] &= + \\
+ v 
\in \text{succ}(v) \\
\text{if color}[v] &= \text{white} \\
\text{f}[v] &= v \\
\text{DFS}(u) \\
\text{color}[v] &= \text{black} \\
+ &= + + 1 \\
f[v] &= +
\end{align*}
\]

Parenthesis Property: \( (CJ), (C) \ [J], \text{Cannot have}: \ (C) \]

Spanning Tree: \( T \) will provide a spanning tree of the graph.

Tree edge: \( T \), to white node

Back edge: \( B \), to grey node (an ancestor), is a loop edge.

Forward edge: \( d(v) < d(u) \), to a descendant

Cross edge: \( d(w) < d(v) \), to a sibling.

Thm: Forward and cross edges are not present in undirected graphs. For any edge \( (u, v) \), \( d(u) < d(v) \) unless it is a back edge.

Topological Sort:
\( \text{In a directed graph there exists at least one node with no incoming edges in a graph without loops} \).

Top Sort:
\( \cdot \text{Do DFS} \)
\( \cdot \text{List nodes in decreasing finish times} \)

\[
\begin{align*}
0 \rightarrow \text{Top Sort} \\
1 \rightarrow \\
2 \rightarrow \\
3 \rightarrow \\
4 \rightarrow \\
5 \rightarrow \\
6 \rightarrow
\end{align*}
\]