Breadth-First Search

First we'll overview the graph representations.

Then we'll look at problem of finding a shortest path in a directed unweighted graph.
Adjacency List

Vertices \( \{a, b, c, d\} \)

Outedges
- \(a \rightarrow \{e_3, e_1, e_2\}\)
- \(b \rightarrow \{e_4\}\)
- \(c \rightarrow \{e_5\}\)
- \(d \rightarrow \{\}\)

Inedges
- \(a \rightarrow \{\}\)
- \(b \rightarrow \{e_3\}\)
- \(c \rightarrow \{e_1\}\)
- \(d \rightarrow \{e_2, e_4, e_5\}\)

We've called Augmented Adj List.
<table>
<thead>
<tr>
<th>Data Structure</th>
<th>storeIncomingEdges</th>
<th>TaggedBucketCollection type</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdjacencyList</td>
<td>false</td>
<td>$V \rightarrow \text{List}&lt;E&gt;$</td>
</tr>
<tr>
<td>AugmentedAdjacencyList</td>
<td>true</td>
<td>$V \rightarrow \text{List}&lt;E&gt;$</td>
</tr>
<tr>
<td>Adjacency Set (no multi-edges)</td>
<td>false</td>
<td>$V \rightarrow \text{Set}&lt;E&gt;$</td>
</tr>
<tr>
<td>Augmented Adjacency Set (no multi-edges)</td>
<td>true</td>
<td>$V \rightarrow \text{Set}&lt;E&gt;$</td>
</tr>
<tr>
<td>Adjacency Set (with multi-edges)</td>
<td>false</td>
<td>$V \rightarrow \text{BucketMapping}&lt;V, \text{List}&lt;E&gt;&gt;$</td>
</tr>
<tr>
<td>Augmented Adjacency Set (with multi-edges)</td>
<td>true</td>
<td>$V \rightarrow \text{BucketMapping}&lt;V, \text{List}&lt;E&gt;&gt;$</td>
</tr>
</tbody>
</table>
Adjacency matrix

\[
\begin{array}{c}
\text{id}s \\
\begin{array}{c}
a ightarrow 0 \\
c ightarrow 2 \\
d ightarrow 3 \\
b ightarrow 1 \\
\end{array} \\
\end{array}
\]

\[
\begin{array}{cccc}
\text{edges} & 0 & 1 & 2 & 3 \\
\begin{array}{cccc}
\emptyset & \emptyset & e_1 & e_2 \\
\emptyset & \emptyset & \emptyset & \emptyset \\
\emptyset & \emptyset & \emptyset & e_5 \\
\emptyset & \emptyset & \emptyset & \emptyset \\
\end{array}
\end{array}
\]

Diagram:

- Vertices: a, b, c, d
- Edges: e_1, e_2, e_3, e_4, e_5
Analysis (directed graph, no multi-edges)

<table>
<thead>
<tr>
<th></th>
<th>Adj List</th>
<th>Adj Set</th>
<th>Adj Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>(#\text{vertices}) \text{m edges}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>\textit{Contains edge } v_i v_j</td>
<td>$O(\text{# edges out of } v_i)$</td>
<td>$O(\text{# vertices})$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>\textit{Iterate over out edges } V</td>
<td>$O(\text{# edges out of } V)$</td>
<td>$O(\text{# edges out of } V)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>\textit{Iterate over all edges}</td>
<td>$O(n+m)$</td>
<td>$O(n+m)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>\textit{Space}</td>
<td>$O(n+m)$</td>
<td>$O(n+m)$</td>
<td>$O(n^2)$</td>
</tr>
</tbody>
</table>
$V_1 \xrightarrow{1} \square \xrightarrow{2} \square \xrightarrow{3} \text{null}$

$V_2 \rightarrow \square$

$\vdots$

$V_n \rightarrow \square \rightarrow \square \rightarrow \square$

\[m + n\]

\[m \text{ total edges}\]
Finding Shortest Paths in Unweighted Directed Graph
Breadth first search $O(n+m)$

Each vertex store

Parent (Edge) - how you were reached from parent

Maybe boolean to know if discovered

Dist (length of shortest path)

Use a queue to maintain order to process
`bfs(V source)`

For each `u ∈ V`
- `u.discovered = false`
- `u.dist = ∞`
- `u.parentEdge = null`

Source.discovered = true
Source.dist = 0
`queue.enqueue(source)`

While `(!queue.isEmpty())`
- `u = q.dequeue()`

For each edge `e` in outgoing edges from `u`
- `v = e.dest`

If `(!v.discovered)`
- `v.discovered = true`
- `v.dist = u.dist + 1`
- `v.parentEdge = e`
- `queue.enqueue(v)}`

Initialization

Source -> dest

U -> e -> V
What if the edges are weighted?

Assume:
No negative weight edges
\[ \log (a \times b \times c \times d) = \log a + \log b + \log c + \log d \]