New Unit: Graph Algorithms

Many different-seeming problems may be represented graphically, and a common set of algorithms can be applied to solving them efficiently.

- Vertices connected by edges (directed or undirected)

"Objects represented with their relationships"

V: web pages, individuals, map locations, network nodes, game states
E: web links, "friends", elem. routes, direct moves, automated players (Chess, Rubik's cube, etc.)

Application:
- page ranking, web crawling
- friend finder, mapping route finder, internet routing
Today

- Graphs
- Graph representations
- Algorithm: Breadth-first search (BFS)

Motivating Problem

Given flights between airports, find the route with the fewest hops connecting any two airports (e.g., Boston → Miami).

Graph Definition

\[ G = (V, E) \]

- Set of vertices \((V)\)
- Set of edges \((E)\)

- Undirected graph (edges have no arrows)
- Directed graph (edges have direction)

\[ V = \{A, B, C\} \]
\[ E = \{(A, B)\} \]
Representing Graphs (in Computers)

Need to somehow list the vertices and their edges in a useful way.

2 common representations:

1. Adjacency Lists - For each vertex list its neighbors

   - array of $|V|$ linked lists
   - For $v \in V$, list $\text{Adj}[v]$ stores neighbors $\{u \mid (v, u) \in E\}$
   - directed stores edge once
   - undirected stores each edge in 2 places

2. Adjacency Matrix

   - assume $V = \{1, ..., n\}$
   - Matrix $A = (a_{ij})$ is $n \times n$ with row $i$ and column $j$
   - $a_{ij} = 1$ if $(i, j) \in E$
   - $a_{ij} = 0$ otherwise

\[ a = \begin{bmatrix}
    0 & 1 & 0 \\
    1 & 0 & 1 \\
    0 & 1 & 0 \\
\end{bmatrix} \]
Operation
- Add edge
- Check existence of edge
- Visit all neighbors of \( V \)

Space usage

<table>
<thead>
<tr>
<th>Adjacency Lists</th>
<th>Adjacency Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathcal{O}(1) )</td>
<td>( \mathcal{O}(1) )</td>
</tr>
<tr>
<td>( \mathcal{O}(\text{longest list}) )</td>
<td>( \mathcal{O}(#\text{neighbors}) )</td>
</tr>
<tr>
<td>( \mathcal{O}(#\text{neighbors}) )</td>
<td>( \mathcal{O}(n) )</td>
</tr>
</tbody>
</table>

\( \mathcal{O}(m) \) | \( \mathcal{O}(n^2) \)

"Searching a Graph"
- find a path from start vertex \( s \) to target \( t \)
- visit all vertices (or edges) of graph (or only those reachable from vertex \( s \))
- count hops \( s \rightarrow t \) or find path with fewest hops
- etc.

Breadth-First Search

S

level | level | level
\( \emptyset \) | 1 | 2

- define a "frontier" of exploration (initially just \( s \))
- "explore" each neighbor of \( s \) by adding to frontier; then remove \( s \) from frontier
- final level
- until nothing further to explore
**BFS** (V, Adj, s)

**Init:**

\[
\begin{align*}
\text{level}[s] &= 0 \\
\text{for every } u \neq s, \text{ level}[u] &= \infty & \leftarrow \text{not yet visited} \\
\text{for every } u \in V, \text{ parent}[u] &= \text{null} \\
\text{frontier} &= [s] & \leftarrow \text{frontier is a queue}
\end{align*}
\]

\(\Theta(V)\)

- While frontier \(\neq \emptyset\)
  - Remove node \(u\) from front of frontier list
  - For \(v\) in \(\text{Adj}[u]\)
    - If \(\text{level}[v] = \infty\) \(\leftarrow\) vertex is unvisited
      - \(\text{level}[v] = \text{level}[u] + 1\)
      - \(\text{parent}[v] = u\)
      - \(\text{frontier}.\text{append}[v]\)
  - Else do nothing \(\leftarrow\) vertex previously visited

\(\Theta(V + E)\)

**Note:**

- Every vertex is in frontier at most once
- Every edge in adjacency lists is processed at most once

**Breadth-First Tree Result:**

- Boston - Chicago
- Newark - Pittsburgh
- Miami - Miami
- San Francisco - Dallas
- Seattle - Portland
Note:

1. BFS systematically explores G to find all vertices reachable from s.
2. Vertices unreachable from s have level[u] = \infty.

\[ S \]

3. Level[u] contains distance (number of edges) from s to u along shortest route (fewest edges) for reachable vertices.

4. BFS produces breadth-first tree of shortest paths:

\[ u \leftarrow \text{parent}[u] \leftarrow \text{parent}[\text{parent}[u]] \leftarrow \ldots \leftarrow s \]

5. If shorter path existed, then u would have been discovered at an earlier level of BFS.