Outline: Search I (of 3)
- graph search
- applications
- graph representations
- introduction to breadth-first & depth-first search

Reading: CLRS 22.1-22.3, B.4

Graph search: explore a graph
  e.g. find a path from start vertex s to a desired vertex

Recall: graph $G = (V,E)$
- $V =$ set of vertices (arbitrary labels)
- $E =$ set of edges i.e. vertex pairs $(v,w)$
  - ordered pair $\Rightarrow$ directed edge & graph
  - unordered pair $\Rightarrow$ undirected

  e.g. \begin{tikzpicture}
    \node (a) at (0,0) {a};
    \node (b) at (1,0) {b};
    \node (c) at (1,-1) {c};
    \node (d) at (0,-1) {d};
    \draw (a) -- (b);
    \draw (a) -- (c);
    \draw (b) -- (c);
    \draw (c) -- (d);
  \end{tikzpicture}
  \hspace{1cm}
  $V = \{a, b, c, d\}$
  $E = \{ab, ac, bc, cd\}$
  UNDIRECTED

  \begin{tikzpicture}
    \node (a) at (0,0) {a};
    \node (b) at (1,0) {b};
    \node (c) at (1,-1) {c};
    \draw (a) -- (b);
    \draw (b) -- (c);
    \draw (b) -- (c);
  \end{tikzpicture}
  \hspace{1cm}
  $V = \{a, b, c\}$
  $E = \{ac, bc, (b,c), (c,b), (b,a)\}$
  DIRECTED
Applications: many
- web crawling (how Google finds pages)
- social networking (Facebook friend finder)
- computer networks: (routing in the Internet)
  shortest paths [next unit]
- solving puzzles & games
- checking mathematical conjectures

Pocket Cube: 2x2x2 Rubik's cube
- configuration graph:
  - vertex for each possible state
  - edge for each basic move (e.g., 90° turn) from one state to another
  - undirected: moves are reversible
- puzzle: given initial state s, find a path to the solved state
- #vertices = 8! \cdot 3^8 = 264,539,520
  8 cubelet in \uparrow arbitrary positions
  each cubelet has 3 possible twists
- can factor out 24-fold symmetry of cube:
  fix one cubelet \Rightarrow 7! \cdot 3^7 = 11,022,480
- in fact, graph has 3 connected components of equal size \Rightarrow only need to search in one
  \Rightarrow 7! \cdot 3^6 = 3,674,160
"Geography" of configuration graph:

![Diagram of configuration graph]

Possible first moves: \( \cdots \) reachable in two steps but not one

<table>
<thead>
<tr>
<th>distance</th>
<th>(90^\circ) turns</th>
<th>(90^\circ&amp;180^\circ) turns</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>27</td>
<td>54</td>
</tr>
<tr>
<td>3</td>
<td>120</td>
<td>321</td>
</tr>
<tr>
<td>4</td>
<td>534</td>
<td>1,847</td>
</tr>
<tr>
<td>5</td>
<td>2,256</td>
<td>9,992</td>
</tr>
<tr>
<td>6</td>
<td>8,969</td>
<td>50,136</td>
</tr>
<tr>
<td>7</td>
<td>33,058</td>
<td>227,536</td>
</tr>
<tr>
<td>8</td>
<td>114,149</td>
<td>876,092</td>
</tr>
<tr>
<td>9</td>
<td>366,508</td>
<td>1,887,748</td>
</tr>
<tr>
<td>10</td>
<td>936,588</td>
<td>623,800</td>
</tr>
<tr>
<td>11</td>
<td>1,350,852</td>
<td>2,644</td>
</tr>
<tr>
<td>12</td>
<td>782,536</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>90,280</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>276</td>
<td>(\frac{3,674,160}{3,674,160})</td>
</tr>
</tbody>
</table>

 Cf. 3x3x3 Rubik's cube: \(~1.4\) trillion states
\[\text{diameter unknown!} \leq 26\]

http://en.wikipedia.org/wiki/Pocket_Cube
Representing graphs: (data structures)

**Adjacency lists:** array $\text{Adj}$ of $|V|$ linked lists
- for each vertex $u \in V$, $\text{Adj}[u]$ stores $u$'s neighbors, i.e. $\{v \in V \mid (u,v) \in E\}$
  - just outgoing edges if directed

*Example:*

```
\begin{tikzpicture}
  \node (a) at (0,0) {a};
  \node (b) at (-1,-1) {b};
  \node (c) at (1,-1) {c};
  \draw [->] (a) -- (b);
  \draw [->] (a) -- (c);
  \draw [->] (b) -- (c);
\end{tikzpicture}
```

```
\begin{tikzpicture}
  \node (a) at (0,0) {a};
  \node (b) at (0,-1) {b};
  \node (c) at (1,-1) {c};
  \node (d) at (2,0) {d};
  \draw [->] (a) -- (b);
  \draw [->] (a) -- (c);
  \draw [->] (a) -- (d);
\end{tikzpicture}
```

- in Python: $\text{Adj}$ = dictionary of list/set values
- vertex = any hashable object (e.g., int, tuple)
- advantage: multiple graphs on same vertices

**Object-oriented variations:**
- object for each vertex $u$
- $u$.neighbors = list of neighbors i.e. $\text{Adj}[u]$

"**Incidence lists:**"
- can also make edges objects
- $u$.edges = list of (outgoing) edges from $u$
- advantage: storing data with vertices & edges without hashing
Representing graphs: (cont’d)  
above representations are good for sparse graphs where \(|E| \ll |V|^2\)  
- space requirement = \(\Theta(V+E)\)

\[\text{Adjacency matrix:}\]
- assume \(V = \{1, 2, \ldots, |V|\}\) (number vertices)
- \(A = (a_{ij}) = |V| \times |V|\) matrix
  - where \(a_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ \emptyset & \text{otherwise} \end{cases}\)
  - \(i = \text{row}\)
  - \(j = \text{column}\)

\[\begin{align*}
  \text{e.g.} & & A = \begin{pmatrix} 1 & 2 & 3 \\
  \emptyset & \emptyset & 1 \\
  1 & \emptyset & 1 \\
  \emptyset & 1 & \emptyset \end{pmatrix} \\
 1 & 2 & 3 \\
  \emptyset & \emptyset & 1 \\
  1 & \emptyset & 1 \\
  \emptyset & 1 & \emptyset \\
\end{pmatrix}
\]

- good for dense graphs where \(|E| \approx |V|^2\)
- space requirement = \(\Theta(V^2)\)
- cool properties like \(A^2\) gives length-2 paths & Google PageRank \(\approx A^\infty\)
- but we’ll rarely use it
  (Google couldn’t: \(|V| \approx 20\) billion \(\Rightarrow |V|^2 \approx 4 \cdot 10^{20}\) \([50,000\) petabytes])

**Implicit graphs:** \(\text{Adj}(u)\) is a function
or \(u.\text{neighbors}/\text{edges}\) is a method
\(\Rightarrow\) “no space” (just what you need now)
High-level overview of next two lectures:

**Breadth-first search**: levels like "geography"

- \( \text{frontier} = \text{current level} \)
- initially \( \{s\} \)
- repeatedly advance \( \text{frontier} \) to next level, careful not to go backwards to previous level
- actually find shortest paths i.e. fewest possible edges

**Depth-first search**: like exploring a maze

- e.g. (left-hand rule)
- follow path until you get stuck
- backtrack along breadcrumbs until you reach an unexplored edge
- recursively explore it
- careful not to repeat a vertex