Lecture 12: Searching I: Graph Search and Representations

Lecture Overview: Search 1 of 3

- Graph Search
- Applications
- Graph Representations
- Introduction to breadth-first and depth-first search

Readings
CLRS 22.1-22.3, B.4

Graph Search

Explore a graph e.g., find a path from start vertices to a desired vertex

Recall: graph $G = (V, E)$

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

![Graph Examples](image)

Figure 1: Example to illustrate graph terminology
Applications:

There are many.

- web crawling (How Google finds pages)
- social networking (Facebook friend finder)
- computer networks (Routing in the Internet) shortest paths [next unit]
- solving puzzles and games
- checking mathematical conjectures

Pocket Cube:

Consider a $2 \times 2 \times 2$ Rubik’s cube

![Rubik’s Cube](image)

Figure 2: Rubik’s Cube

- Configuration Graph:
  - vertex for each possible state
  - edge for each basic move (e.g., 90 degree turn) from one state to another
  - undirected: moves are reversible
- Puzzle: Given initial state $s$, find a path to the solved state
- $\#$ vertices $= 8! \cdot 3^8 = 264,539,520$ (because there are 8 cubelets in arbitrary positions, and each cubelet has 3 possible twists)

![Symmetry](image)

Figure 3: Illustration of Symmetry
• can factor out 24-fold symmetry of cube: fix one cubelet

\[ 8^{11} \cdot 3 \Rightarrow 7! \cdot 3^7 = 11,022,480 \]

• 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 Breadth-First Tree](image)

Figure 4: Breadth-First Tree

<table>
<thead>
<tr>
<th>distance</th>
<th>90° turns</th>
<th>90° &amp; 180° 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>870,072</td>
</tr>
<tr>
<td>9</td>
<td>360,508</td>
<td>1,887,748</td>
</tr>
<tr>
<td>10</td>
<td>930,588</td>
<td>623,800</td>
</tr>
<tr>
<td>11</td>
<td>1,350,852</td>
<td>2,644 ← diameter</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 ← diameter</td>
<td>3,674,160</td>
</tr>
</tbody>
</table>

Wikipedia Pocket Cube

*Cf.* $3 \times 3 \times 3$ Rubik’s cube: $\approx 1.4$ trillion states; diameter is unknown! $\leq 26$
Representing Graphs: (data structures)

**Adjacency lists:**

Array $Adj$ of $|V|$ linked lists

- for each vertex $u \in V$, $Adj[u]$ stores $u$’s neighbors, i.e., $\{v \in V | (u,v) \in E\}$. $\text{colorBlue}(u, v)$ are just outgoing edges if directed. (See Fig. 5 for an example)

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

- advantage: multiple graphs on same vertices

![Adjacency List Representation](image)

Figure 5: Adjacency List Representation

**Object-oriented variations:**

- object for each vertex $u$

- $u$.neighbors = list of neighbors i.e., $Adj[u]$

**Incidence Lists:**

- can also make edges objects (see Figure [6])

- $u$.edges = list of (outgoing) edges from $u$.

- advantage: storing data with vertices and edges without hashing
Representing Graphs: contd.

The above representations are good for sparse graphs where $|E| \ll (|V|^2)$. This translates to a space requirement $= \Theta(V + E)$ (Don’t bother with $|$’s inside $O/\Theta$).

Adjacency Matrix:

- assume $V = \{1, 2, \ldots, |v|\}$ (number vertices)
- $A = (a_{ij}) = |V| \times |V|$ matrix where $i = \text{row}$ and $j = \text{column}$, and

$$a_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E \\ \phi & \text{otherwise} \end{cases}$$

See Figure 7

- good for dense graphs where $|E| \approx (|V|^2)$
- space requirement $= \Theta(V^2)$
- cool properties like $A^2$ gives length-2 paths and Google PageRank $\approx A^\infty$
- but we’ll rarely use it Google couldn’t; $|V| \approx 20$ billion $\implies (|V|^2) \approx 4.10^{20}$ [50,000 petabytes]

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Figure 7: Matrix Representation
Implicit Graphs:
Adj(u) is a function or u.neighbors/edges is a method $\implies$ “no space” (just what you need now)

High level overview of next two lectures:

Breadth-first search
Levels like “geography”

![Figure 8: Illustrating Breadth-First Search](image)

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

Depth-first search
This is like exploring a maze.

- e.g.: (left-hand rule) - See Figure 9
- follow path until you get stuck
- backtrack along breadcrumbs until you reach an unexplored edge
• recursively explore it

• careful not to repeat a vertex

Figure 9: Illustrating Depth-First Search