Today: Graphs I: BFS (I of 2)
- applications of graph search
- graph representations
- breadth-first search

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

e.g. \[
\begin{array}{cc}
\text{UNDIRECTED} & \text{DIRECTED} \\
V = \{a, b, c, d\} & V = \{a, b, c\} \\
E = \{\{a, b\}, \{a, c\}, \{b, c\}, \{b, d\}, \{c, d\}\} & E = \{(a,c), (b,c), (c,b), (b,a)\}
\end{array}
\]

Graph search: “explore a graph”
e.g. find a path from start vertex $s$
to a desired vertex
e.g. visit all vertices or edges of graph, or only those reachable from $s$
Applications: many
- web crawling (how Google finds pages)
- social networking (Facebook friend finder)
- network broadcast routing
- garbage collection
- model checking (finite state machine)
- checking mathematical conjectures
- solving puzzles & games

Pocket Cube: 2×2×2 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

\[ \text{diameter ("God's Number")} \]

\[ 11 \text{ for 2x2x2} \]
\[ 20 \text{ for 3x3x3} \]
\[ O(n^{3/2} \log n) \text{ for nxnxn} \]

Demaine, Demaine, Eisenstat, Lubiw, Winslow 2011

\[ \# \text{vertices} = 8! \cdot 3^8 = 264,539,520 \]

- First moves
- \# possible first moves

8 cubelet in arbitrary positions each cubelet has 3 possible twists
\[ \times \frac{1}{24} \text{ if we remove cube symmetries} \]
\[ \times \frac{1}{3} \text{ actually reachable (3 connected components)} \]
Graph representation (data structures)

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

  e.g. 
  
  ![Diagram](image)
  
  Adj

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

  - advantage: multiple graphs on same vertices

**Implicit graphs:** Adj(u) is a function
- compute local structure on the fly
  - e.g. Rubik’s Cube

**Object-oriented variations:**
- object for each vertex u
  - u.neighbors = list of neighbors i.e. Adj[u]
  - (or method for implicit graphs)

“Incidence lists:”
- can also make edges objects
  - e.g. e = e.b

- u.edges = list of (outgoing) edges from u
- advantage: store edge data without hashing
**Breadth-first search (BFS):**
- Explore graph level by level from \( s \).
  - Level \( \emptyset = \{s\} \).
  - Level \( i = \) vertices reachable by path of \( i \) edges but not fewer.
  - Build level \( i > \emptyset \) from level \( i-1 \) by trying all outgoing edges, but ignoring vertices from previous levels.

**BFS** \( (s, \text{Adj}) \):
- \( \text{level} = \{s: \emptyset\} \)
- \( \text{parent} = \{s: \text{None}\} \)
- \( i = 1 \)
- \( \text{frontier} = [s] \)

While \( \text{frontier} \):
- \( \text{next} = [] \)
  - \( \text{next level, i} \)
  - For \( u \) in \( \text{frontier} \):
    - For \( v \) in \( \text{Adj}[u] \):
      - If \( v \) not in \( \text{level} \):
        - \( \text{level}[v] = i \)
        - \( \text{parent}[v] = u \)
        - \( \text{next}.\text{append}(v) \)
  - \( \text{frontier} = \text{next} \)
  - \( i += 1 \)

[See CLRS for queue-based implementation]
Example:

```
Example: 
```

```
Analysis:
- vertex v enters next (and then frontier)
  only once (because level[v] then set)
  - base case: v = s
  => Adj[v] looped through only once
- time = \( \sum_{v \in V} |\text{Adj}[v]| = O(E) \) for directed graphs
  \( O(2E) \) for undirected graphs
  => O(E) time
- O(V+E) to also list vertices unreachable
  from v (those still not assigned level)
  "linear time"

Shortest paths: [cf. L15-18]
- for every vertex v, fewest edges to get
  from s to v is \( \{ \text{level}[v] \} \) if v assigned level
  \( \infty \) else (no path)
- parent pointers form shortest-path tree
  = union of such a shortest path for each v
  => to find shortest path, take v, parent[v],
  parent[parent[v]], etc., until s (or None)