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 \((v, w)\)
  - ordered pair \( \Rightarrow \) directed edge & graph
  - unordered pair \( \Rightarrow \) undirected

\[
\begin{align*}
\text{e.g.} & \quad \begin{array}{c}
\text{UNDIRECTED} \\
\text{DIRECTED}
\end{array} \\
V &= \{a, b, c, d\} \\
E &= \{\{a, b\}, \{a, c\}, \{b, c\}, \{b, d\}, \{c, d\}\} \\
V &= \{a, b, c\} \\
E &= \{\{a, c\}, \{b, c\}, \{c, b\}, \{b, a\}\}
\end{align*}
\]

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: 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

11 for 2x2x2
20 for 3x3x3
O(n^2/log n) for nxnxn

\(8! \cdot 3^8 = 264,539,520\)

11 for 2x2x2
20 for 3x3x3
\(O(n^2/log n)\) for nxnxn

\(8! \cdot 3^8 = 264,539,520\)

solution

possible first moves

reachable in two steps but not one

breadth-first tree

hardest configs
Graph representation: (data structures)

**Adjacency lists:** array 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

- **e.g.**
  - \(\text{Adj}\)

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

**Implicit graphs:** \(\text{Adj}(u)\) 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. \(\text{Adj}[u]\)
  - (or method for implicit graphs)

**"Incidence lists:"
- can also make edges objects
- \(u\).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

$\text{BFS}(s, \text{Adj})$:

```
level = \{s: \emptyset\}  # previous level, i-1
parent = \{s: None\}
i = 1
frontier = [s]           # next level, i
while frontier:
    next = []
    for u in frontier:
        for v in Adj[u]:
            if v not in level:
                level[v] = i  # not yet seen
                parent[v] = u
                next.append(v)
    frontier = next
    i += 1
```

[see CLRS for queue-based implementation]
Example:

Analysis:
- vertex \( v \) enters next (\& then frontier) only once (because level[v] then set)
  - base case: \( v = s \)
  \[ \Rightarrow \text{Adj}[v] \text{ looped through only once} \]
- \( \text{time} = \sum_{v \in V} |\text{Adj}[v]| \leq |E| \) for directed graphs
  \( \leq 2|E| \) for undirected graphs
  \[ \Rightarrow O(E) \text{ time} \]
- \( O(V+E) \) 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 \( v \)
  \[ \Rightarrow \text{to find shortest path, take } v, \text{parent}[v], \text{parent}[\text{parent}[v]], \ldots \text{, until } s \text{ (or None)} \]

frontier_0 = \{ s \}
frontier_1 = \{ a, x \}
frontier_2 = \{ z, d, c \}
frontier_3 = \{ f, v \}
(not x, c, d)
6.006 Introduction to Algorithms
Fall 2011

For information about citing these materials or our Terms of Use, visit: http://ocw.mit.edu/terms.