Graph Traversal
Depth-First Search
Breadth-First Search
Graph traversal - Idea

Problem:
- you visit each node in a graph, but all you have to start with is:
  - One vertex A
  - A method getNeighbors(vertex v) that returns the set of vertices adjacent to v
Graph traversal - Motivations

Applications

- Exploration of graph not known in advance, or too big to be stored:
  - Web crawling
  - Exploration of a maze

- Graph may be computed as you go. Example: game strategy:
  - Vertices = set of all configurations of a Rubik's cube
  - Edges connect pairs of configuration that are one rotation away.
Depth-First Search

Diamond: Idea: Go Deep!

- Intuition: Adventurous web browsing: always click the first unvisited link available. Click "back" when you hit a deadend.

- Start at some vertex $v$
- Let $w$ be the first neighbor of $v$ that is not yet visited. Move to $w$.
- If no such unvisited neighbor exists, move back to the vertex that lead to $v$
Example

- A: unexplored vertex
- A: visited vertex
- unexplored edge
- discovery edge

Depth-First Search
Example (cont.)
DFS Algorithm

Algorithm \textit{DFS}(G, v)

\textbf{Input:} graph \textit{G} with no parallel edges and a start vertex \textit{v} of \textit{G}

\textbf{Output:} Visits each vertex once (as long as \textit{G} is connected)

\begin{verbatim}
print \textit{v}  // or do some kind of processing on \textit{v}
\textit{v}.setLabel(VISITED)
for all \textit{u} \in \textit{v}.getNeighbors()
    if ( \textit{u}.getLabel() != VISITED ) then \textit{DFS}(G, \textit{u})
\end{verbatim}
**DFS and Maze Traversal**

- The DFS algorithm is similar to a classic strategy for exploring a maze.
  - We mark each intersection, corner and dead end (vertex) visited.
  - We mark each corridor (edge) traversed.
  - We keep track of the path back to the entrance (start vertex) by means of a rope (recursion stack).
DFS and Rubik’s cube

Rubik’s cube game can be represented as a graph:
- Vertices: Set of all possible configurations of the cube
- Edges: Connect configurations that are just one rotation away from each other

Given a starting configuration $S$, find a path to the “perfect” configuration $P$

Depth-first search could in principle be used:
- start at $S$ and making rotations until $P$ is reached, avoiding configurations already visited

Problem: The graph is huge:
43,252,003,274,489,856,000 vertices
Running time of DFS

- DFS(G, v) is called once for every vertex v (if G is connected)
- When visiting node v, the number of iterations of the for loop is deg(v).
- Conclusion: The total number of iterations of all for loops is: $\sum_v \text{deg}(v) = ?$

- Thus, the total running time is $O(|E|)$
Applications of variants of DFS

DFS can be used to:

- Determine if a graph is connected
- Determine if a graph contains cycles
- Solve games single-player games like Rubik’s cube
Breadth-First Search

**Idea:**
- Explore graph layers by layers
- Start at some vertex $v$
- Then explore all the neighbors of $v$
- Then explore all the unvisited neighbors of the neighbors of $v$
- Then explore all the unvisited neighbors of the neighbors of the neighbors of $v$
- until no more unvisited vertices remain
Example

unexplored vertex
visited vertex
unexplored edge
discovery edge

Depth-First Search
Example (cont.)
Example (cont.)

Depth-First Search
Iterative BFS

Idea: use a queue to remember the set of vertices on the frontier

Algorithm iterativeBFS(G, v)

Input graph G with no parallel edges and a start vertex v of G
Output Visits each vertex once (as long as G is connected)

q ← new Queue()
v.setLabel(VISITED)
q.enqueue(v)

while (! q.empty()) do
    w ← q.dequeue()
    print w // or do some kind of processing on w
    for all u ∈ w.getNeighbors() do
        if (u.getLabel() != VISITED) then
            u.setLabel(VISITED)
s.enqueue(u)
Running time and applications

- Running time of BFS: Same as DFS, $O(|E|)$
- BFS can be used to:
  - Find a shortest path between two vertices
    - Rubik’s cube’s fastest solution
  - Determine if a graph is connected
  - Determine if a graph contains cycles
  - Get out of an infinite maze...
Iterative DFS

Use a stack to remember your path so far

Algorithm iterativeDFS($G$, $v$)

Input graph $G$ with no parallel edges and a start vertex $v$ of $G$

Output Visits each vertex once (as long as $G$ is connected)

$s \leftarrow \text{new Stack}()$

$v$.setLabel(VISITED)

$s$.push($v$)

while (! $s$.empty()) do

$w \leftarrow s$.pop()

print $w$

for all $u \in w$.getNeighbors() do

if $(u$.setLabel() \neq VISITED$) then

$u$.setLabel(VISITED)

$s$.push($u$)

Notice: Code is identical to BFS, but with a stack instead of a queue