CS171 Introduction to Computer Science II

Graphs
Graphs

• Definitions
• Implementation/Representation of graphs
• Search
Adjacency-list graph representation

Maintain vertex-indexed array of lists.
Traversing graphs

- Graph traversal: visit each vertex in the graph exactly once

- There are in general two ways to traverse a graph
  - Depth-first search (DFS): Uses a Stack or recursion
    - Begins at a node, explores as far as possible along each branch before backtracking
  - Breath-first search (BFS): uses a Queue
    - Begins at a node, explores all its neighboring nodes. Then for each of those nodes, explores their unexplored neighbor nodes, and so on
Maze exploration

**Maze graphs.**
- Vertex = intersection.
- Edge = passage.

**Goal.** Explore every intersection in the maze.
Trémaux maze exploration

Algorithm.

- Unroll a ball of string behind you.
- Mark each visited intersection and each visited passage.
- Retrace steps when no unvisited options.
Depth-first search

**Goal.** Systematically search through a graph.

**Idea.** Mimic maze exploration.

**DFS (to visit a vertex v)**

Mark v as visited.
Recursively visit all unmarked vertices w adjacent to v.

**Typical applications.** [ahead]

- Find all vertices connected to a given source vertex.
- Find a path between two vertices.
Depth-first search (warmup)

**Goal.** Find all vertices connected to s.

**Idea.** Mimic maze exploration.

**Algorithm.**
- Use recursion (ball of string).
- Mark each visited vertex.
- Return (retrace steps) when no unvisited options.

**Data structure.**
- boolean[] marked to mark visited vertices.
Depth-first search (warmup)

```java
public class DepthFirstSearch {
    private boolean[] marked;

    public DepthFirstSearch(Graph G, int s) {
        marked = new boolean[G.V()];
        dfs(G, s);
    }

    private void dfs(Graph G, int v) {
        marked[v] = true;
        for (int w : G.adj(v))
            if (!marked[w])
                dfs(G, w);
    }

    public boolean marked(int v) {
        return marked[v];
    }
}
```
Depth-First Search (DFS) – Nonrecursive algorithm

- Visit an unvisited neighbor of the current node if possible, push it on the stack
- Pop a node from the stack, make it current node, repeat the above
- Done when the stack is empty
Depth-first search application: preparing for a date

Preparing for a Date:
What situations might I prepare for?
1) Medical emergency
2) Dancing
3) Food too expensive, under donut count

Okay, what kinds of emergencies can happen?
1) Snakebite
2) Lightning strike
3) Fall from chair

Hmm. Which snakes are dangerous? Let's see...
1) Corn snake
2) Garter snake
3) Copperhead

Research comparing snake venoms is scattered and inconsistent. I'll make a spreadsheet to organize it.

I'm here to pick you up. You're not dressed.

By LP, the inland taipan has the deadliest venom of any snake.

I really need to stop using depth-first searches.
Pathfinding in graphs

**Goal.** Does there exist a path from $s$ to $t$? If yes, find any such path.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public class <em>Paths</em></td>
<td></td>
</tr>
<tr>
<td><code>Paths(Graph G, int s)</code></td>
<td>find paths in $G$ from source $s$</td>
</tr>
<tr>
<td>boolean <code>hasPathTo(int v)</code></td>
<td>is there a path from $s$ to $v$?</td>
</tr>
<tr>
<td><code>Iterable&lt;Integer&gt; pathTo(int v)</code></td>
<td>path from $s$ to $v$; null if no such path</td>
</tr>
</tbody>
</table>
Depth-first search (pathfinding)

Goal. Find paths to all vertices connected to a given source $s$.


Algorithm.
- Use recursion (ball of string).
- Mark each visited vertex by keeping track of edge taken to visit it.
- Return (retrace steps) when no unvisited options.

Data structures.
- boolean[] marked to mark visited vertices.
- int[] edgeTo to keep tree of paths.
- (edgeTo[$w$] == $v$) means that edge $v$-$w$ was taken to visit $w$ the first time.
public class DepthFirstPaths {
    private boolean[] marked;
    private int[] edgeTo;
    private final int s;

    public DepthFirstPaths(Graph G, int s) {
        marked = new boolean[G.V()];
        edgeTo = new int[G.V()];
        this.s = s;
        dfs(G, s);
    }

    private void dfs(Graph G, int v) {
        marked[v] = true;
        for (int w : G.adj(v))
            if (!marked[w]) {
                edgeTo[w] = v;
                dfs(G, w);
            }
    }

    public boolean hasPathTo(int v) {
        return marked[v];
    }

    public Iterable<Integer> pathTo(int v) {
        if (!hasPathTo(v)) return null;
        List<Integer> path = new ArrayList<Integer>();
        for (int w = v; w != s; w = edgeTo[w])
            path.add(0, w);
        path.add(0, s);
        return path;
    }
}
Depth-first search (pathfinding iterator)

edgeTo[] is a parent-link representation of a tree rooted at s.

```
public boolean hasPathTo(int v)
{    return marked[v]; }

public Iterable<Integer> pathTo(int v)
{
    if (!hasPathTo(v)) return null;
    Stack<Integer> path = new Stack<Integer>();
    for (int x = v; x != s; x = edgeTo[x])
        path.push(x);
    path.push(s);
    return path;
}
```
Graph Search

• Depth-first search
• Breadth-first search