CS171 Introduction to Computer Science II

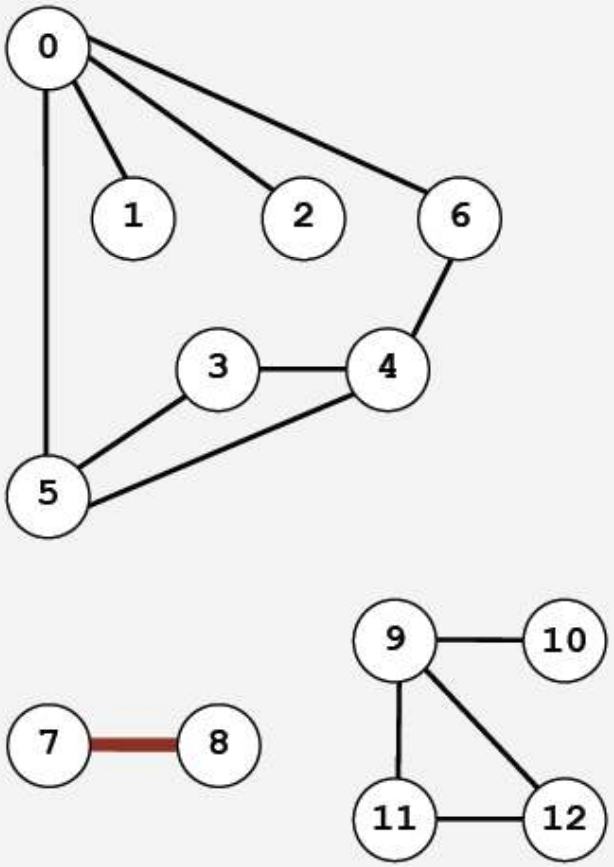
Graphs

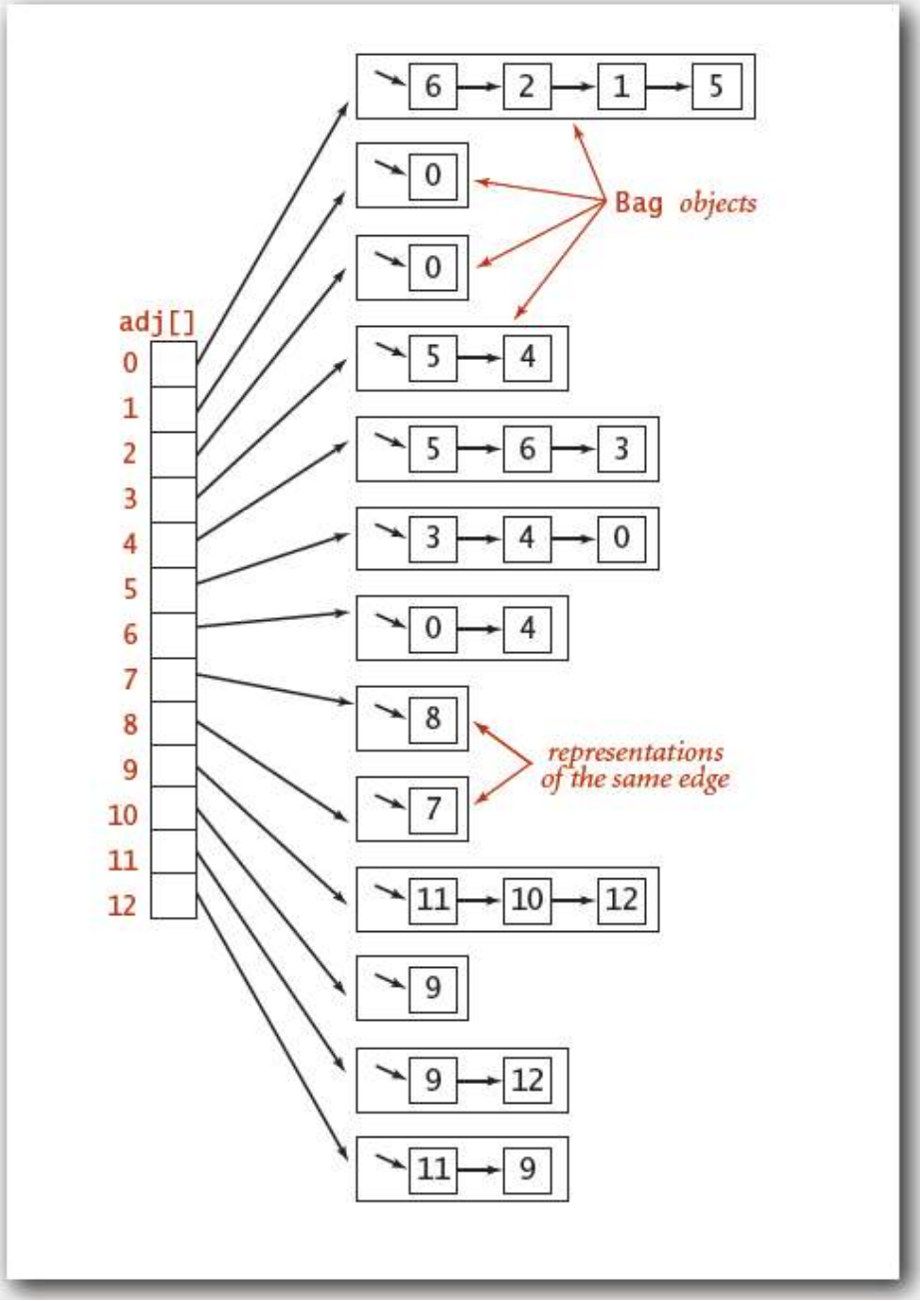
Graphs

- Definitions
- Implementation/Representation of graphs
- Search

Adjacency-list graph representation

Maintain vertex-indexed array of lists.





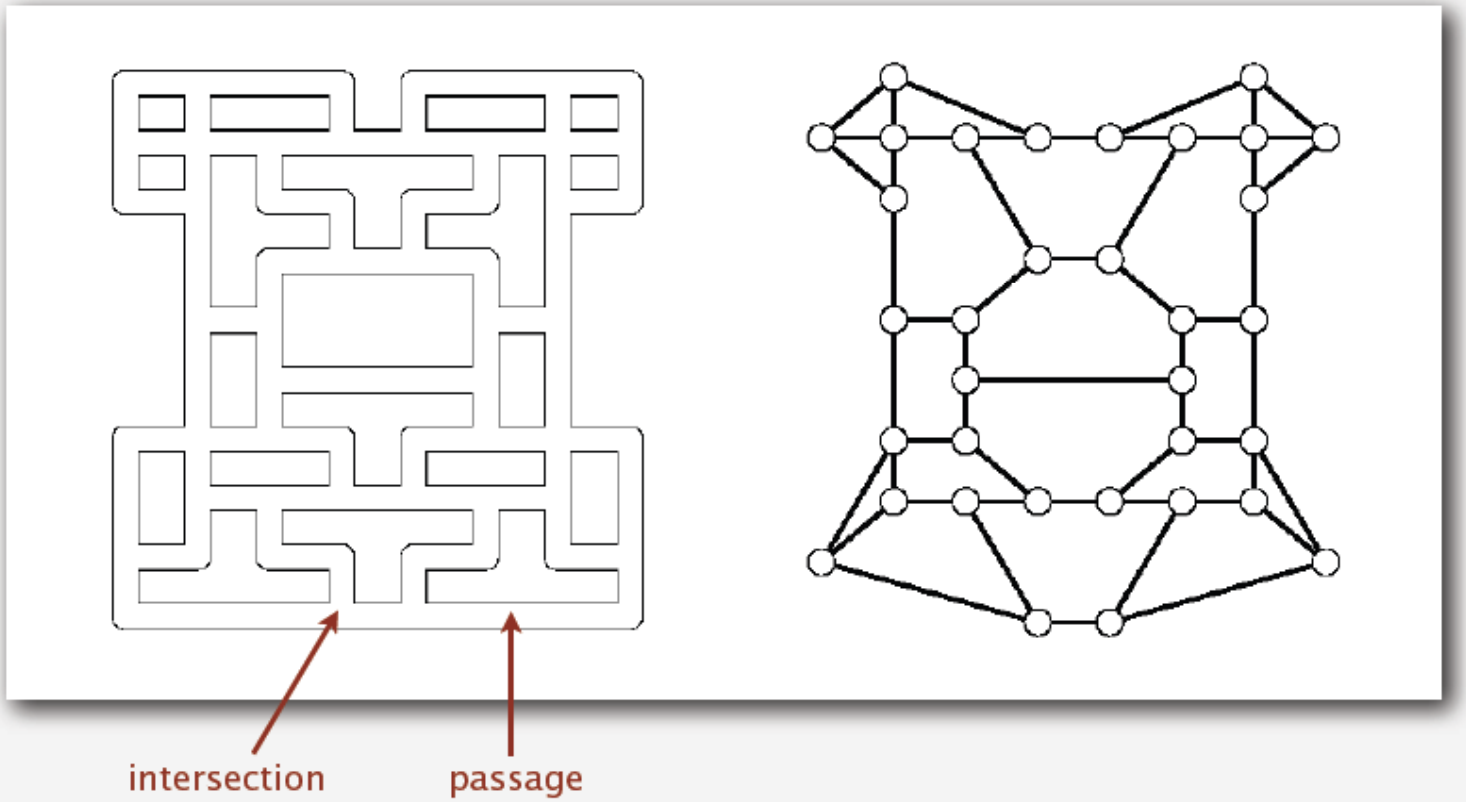
Traversing graphs

- 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.

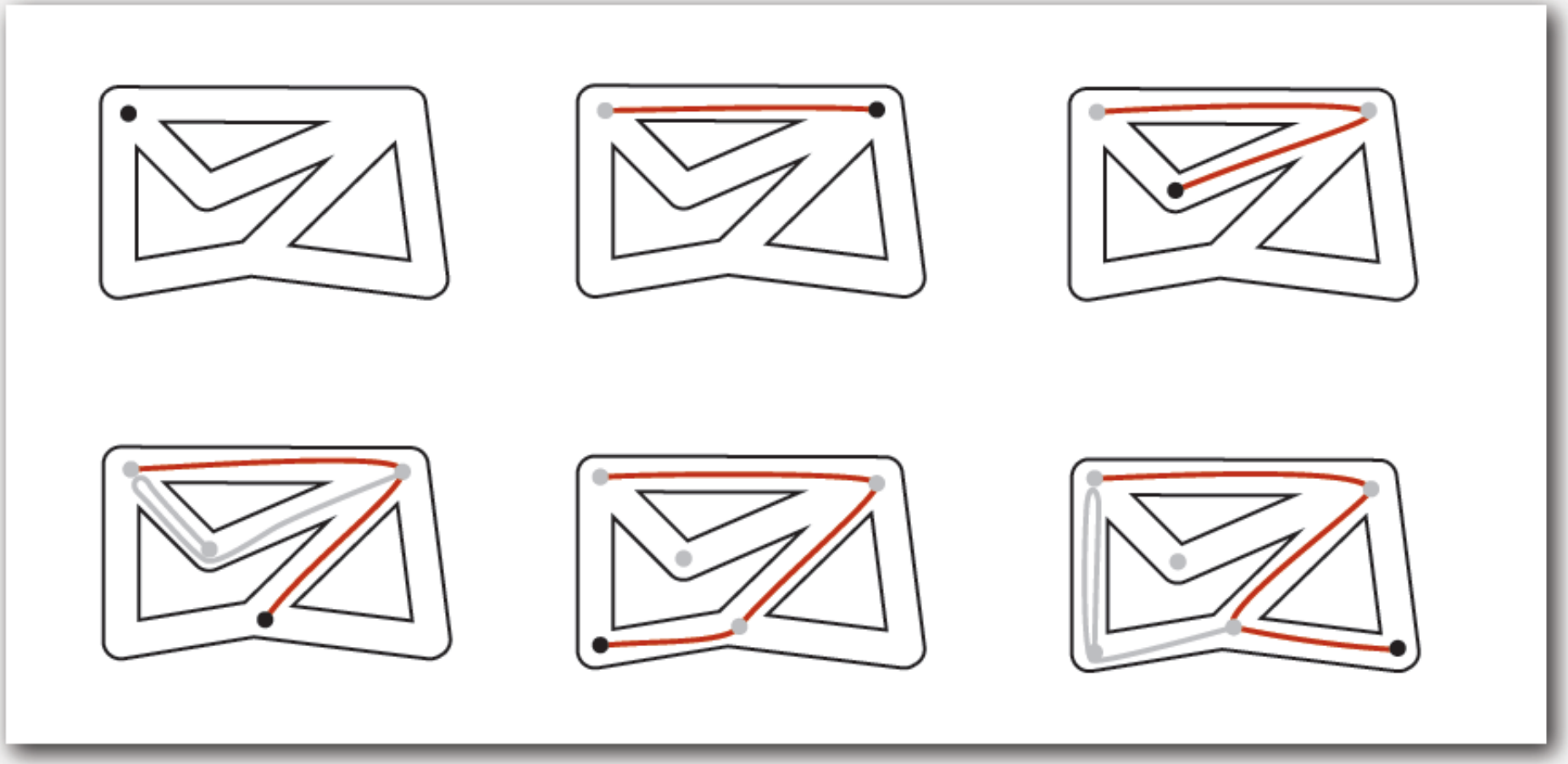


Goal. Explore every intersection in the maze.

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.

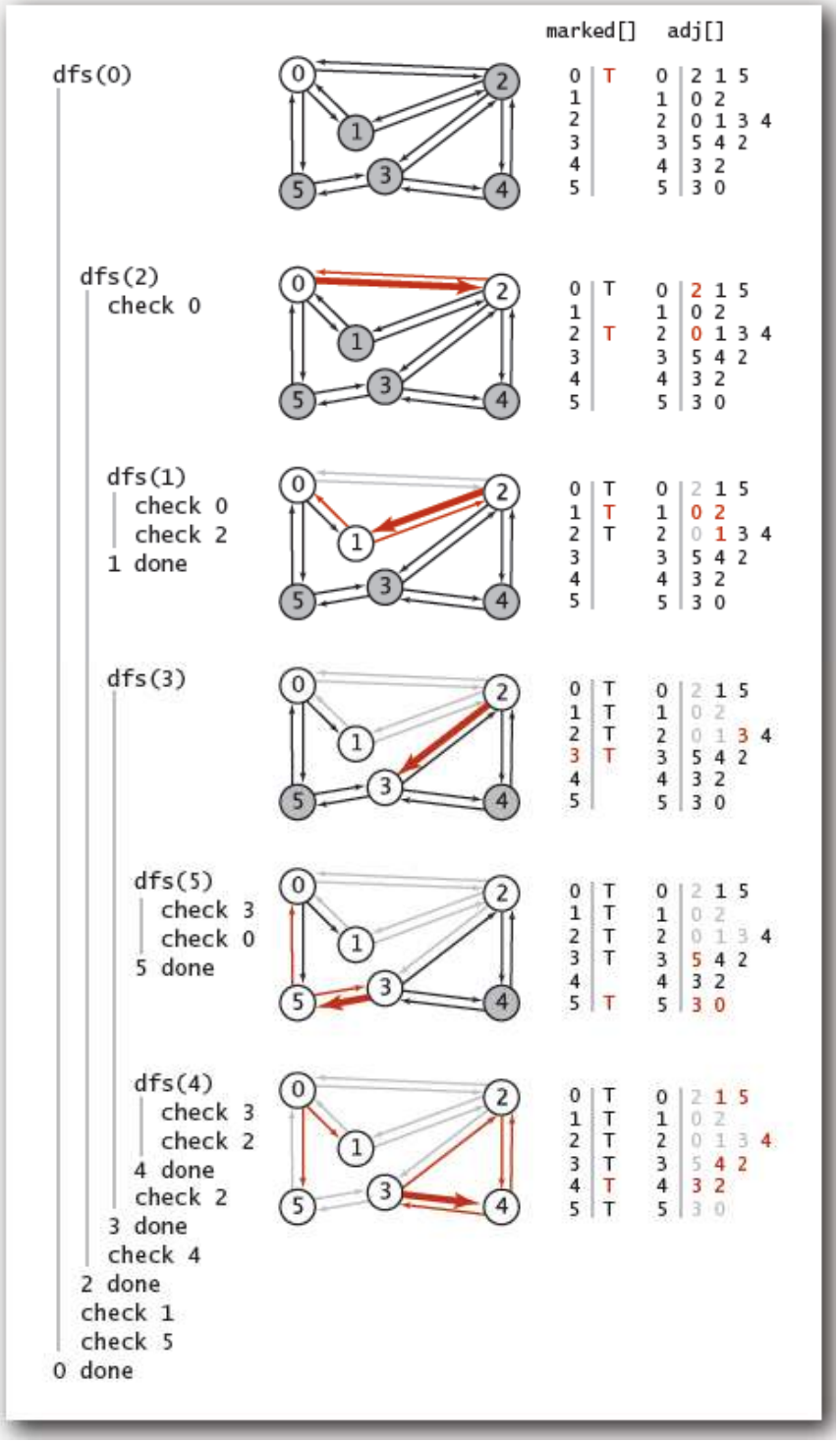
Idea. Mimic maze exploration.

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)

```
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];  }
}
```

marked[v] = true
if v connected to s

constructor marks
vertices connected to s

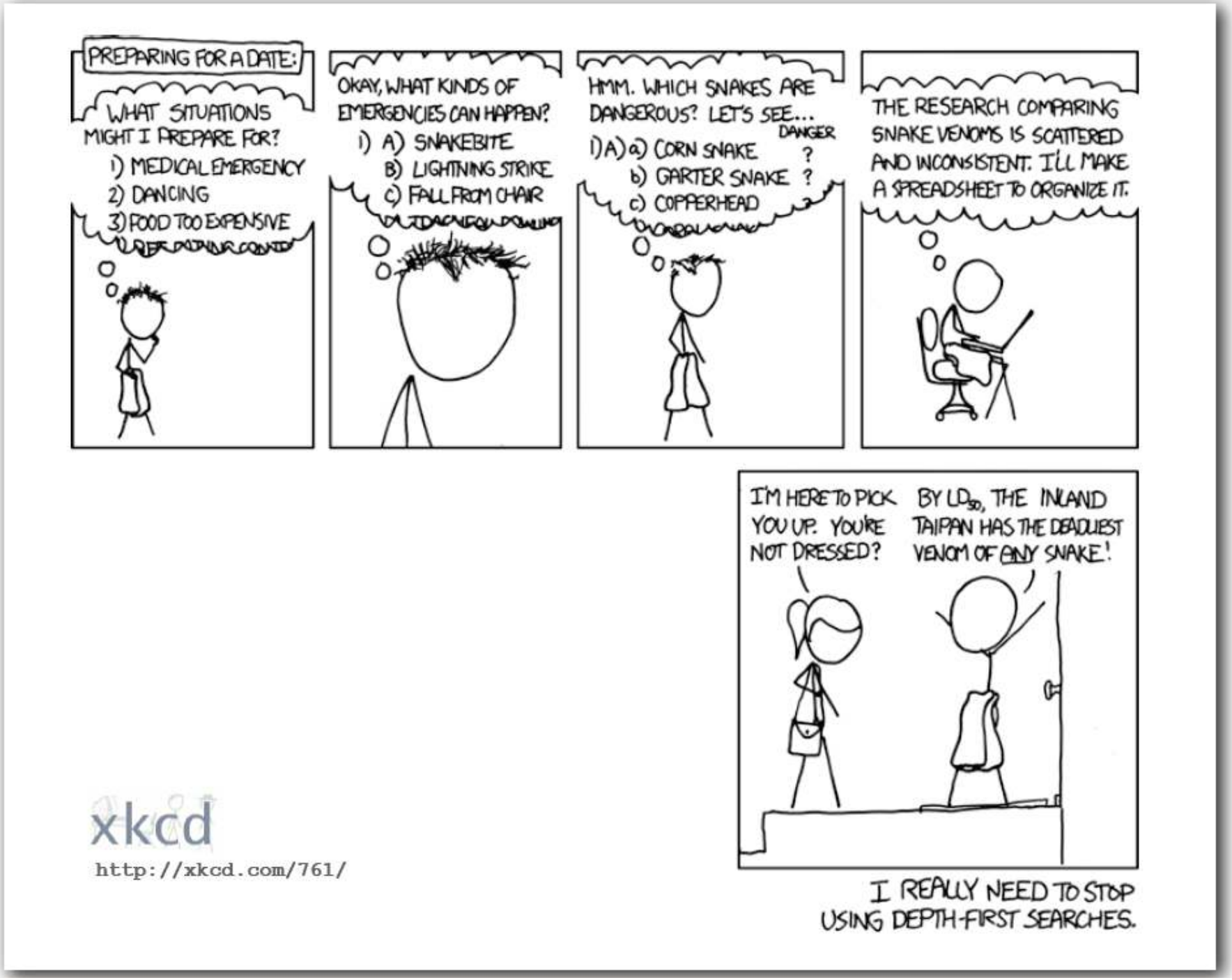
recursive DFS does the work

client can ask whether
vertex v is connected to s

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



Pathfinding in graphs

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

public class Paths		
	`Paths(Graph G, int s)`	*find paths in G from source s*
`boolean`	`hasPathTo(int v)`	*is there a path from s to v?*
`Iterable<Integer>`	`pathTo(int v)`	*path from s to v; null if no such path*

Depth-first search (pathfinding)

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

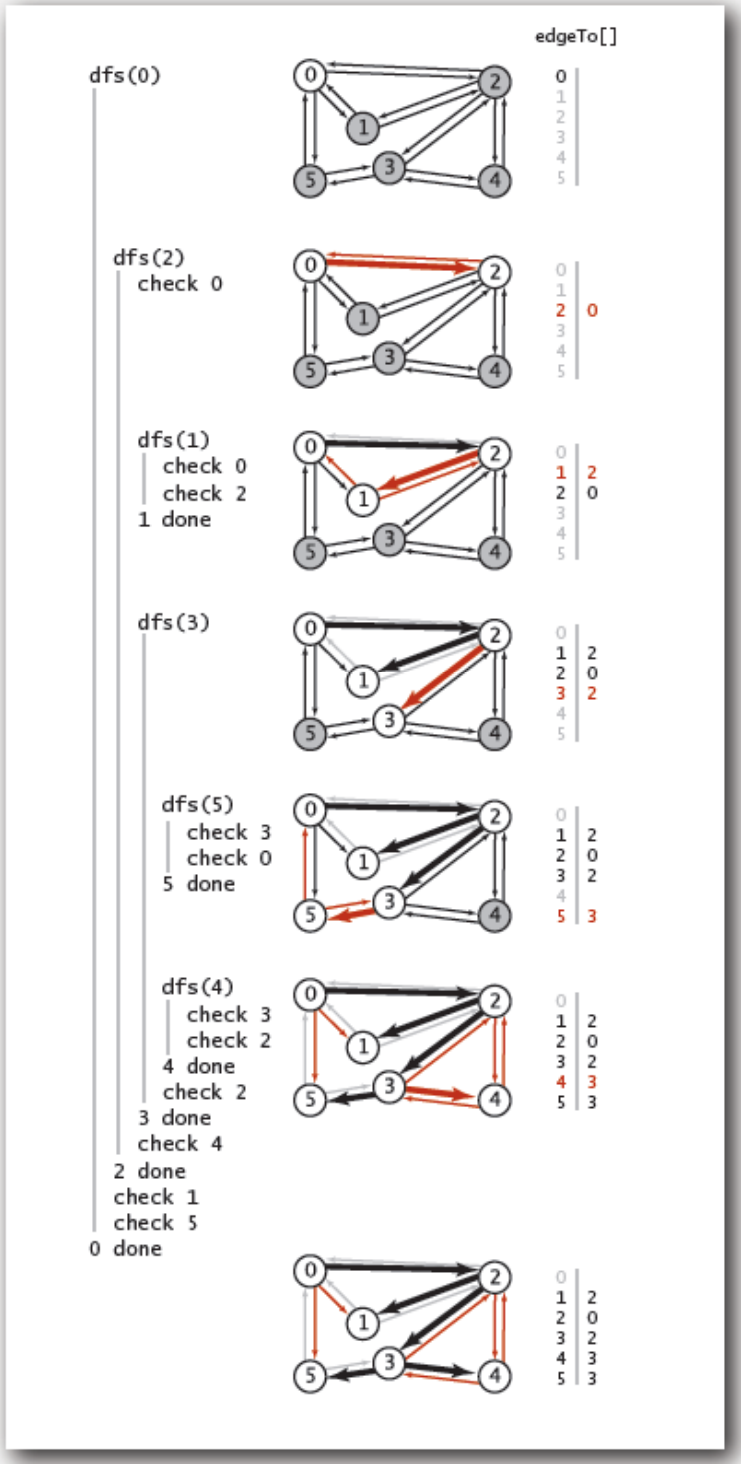
Idea. Mimic maze exploration.

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



Depth-first search (pathfinding)

```
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)
   public Iterable<Integer> pathTo(int v)
}
```

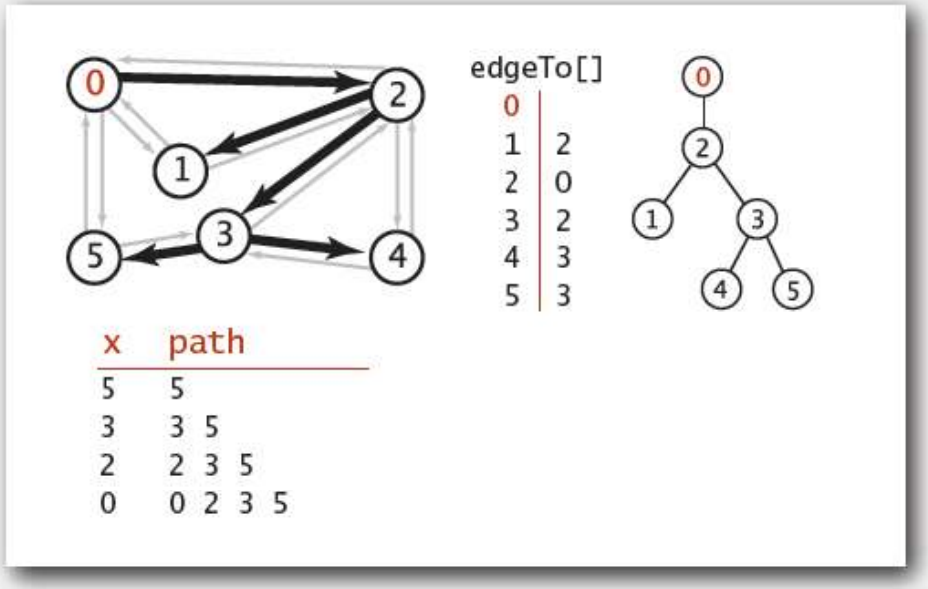
parent-link representation of DFS tree

set parent link

ahead

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