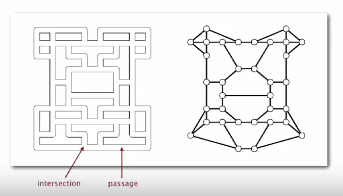
Depth-First Search

Consider similar to a maze… model with a graph. 

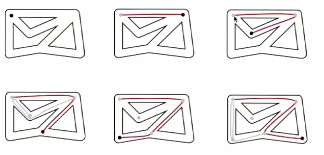
Maze graph:

* Vertex = intersection
* Edge = passage

Goal: Explore every intersection in the maze

Tremaux maze exploration

* Unroll a ball of string behind you
* Mark each visited intersection and each visited passage
* Retrace steps when no unvisited options



First use: when Theseus entered Labyrinth to kill the Minotaur (fun fact)

DFS

Goal: systematically search through a graph

Idea: mimic maze exploration

---

DFS(to visit a vertex *V*)

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

*---*

Typical applications

* Find all vertices connected to a given source vertex
* Find a path between two vertices

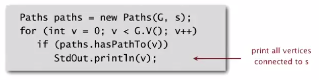
Design pattern: decouple graph data from graph processing

* Create a Graph object
* Pass the Graph to a graph-processing routine
* Query the graph-processing routine for information

Potential API:

Public class Paths  
Paths(Graph G, int s) : find all 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

Implementation



DFS DEMO:

To visit a vertex *v:*

* Mark vertex *v* as visited
* Recursively visit all unmarked vertices adjacent to *v*

Summary of DFS:

Goal: Find all vertices connected to *s* (and a corresponding path)

Idea: Mimic maze exploration

Algorithm:

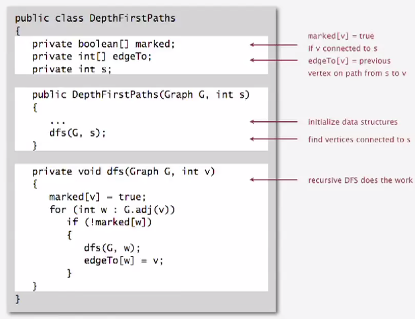
* Use recursion (ball of string()
* Mark each visited vertex (and keep track of edge taken to visit it)
* Return (retrace steps) when no unvisited options

Data structures:

* Boolean[] marked : mark visited vertices
* Int[] edgeTo : keep tree of paths

(edgeTo[w] == v) means that edge v-w taken to visit w for first time

Implementation of Depth-first search:



Depth-first search properties

**Proposition**: DFS marks all vertices connected to *s* in time proportion to the sum of their degrees

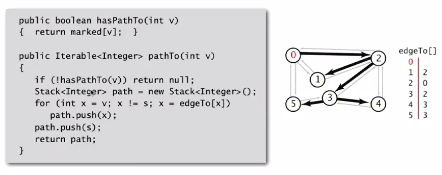
Proof:

* If you mark the vertex w, there has to be a way to get to the vertex from s (through a series of recursive calls, each of which corresponds to an edge on a path from s to w )
* If w is connected to s, then w is marked  
  (if w is unmarked, then consider last edge on a path from s to w that goes from a marked vertex to an unmarked one. The last edge is the furthest link that can be made from s)
* Running time: each vertex connected to s is visited once (linear)

**Proposition**: After DFS completes, can find vertices connected to s in constant time and can find a path to s (if one exists) in time proportional to its length)

Proof: edgeTo[] is a parent-link representation (if a vertex is connected to s, its edgeTo is its parent in the tree) of a tree rooted at s.

Implementation:



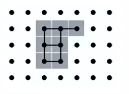
DFS Applications

Flood fill in Photoshop



Assumption: pictures have billions of pixels

Solution: build a grid graph



* Vertex: pixel
* Edge: between two adjacent gray pixels
* Blob: all pixels connected to a given pixel