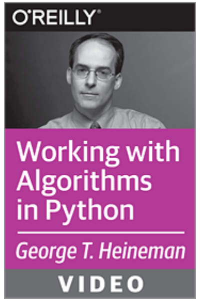
O'REILLY®

Graph Algorithms



O'REILLY

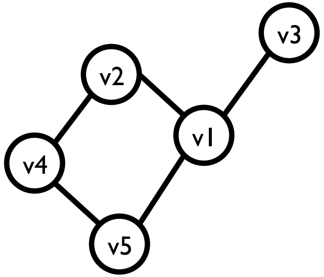
Working with Algorithms in Python

George T. Heineman

VIDEO

Graph Representation

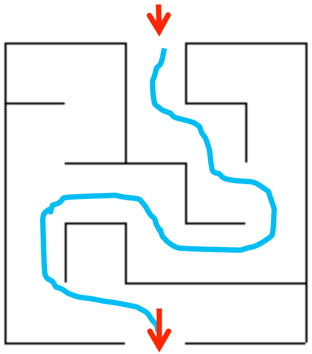
- Useful data structure in many domains
  - Represents information relationships between items
  - Assume simple graphs (no loops, no multiple edges)

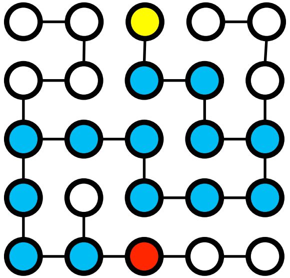


Vertices: v1, v2, v3, v4, v5

Edges: (v1,v2), (v1,v3), (v1,v5),
(v2,v4), (v4,v5)

Casting a Maze As a Graph Problem



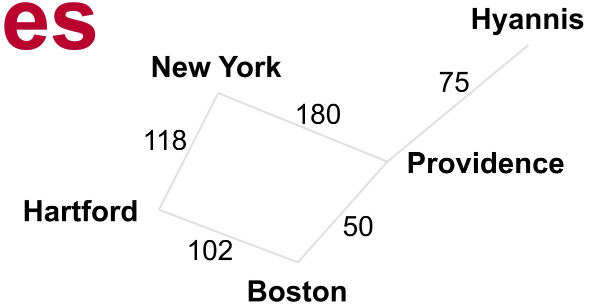


- Find a solution to a rectangular maze
  - Enter at given square and exit at destination
- Represent maze as a **graph**
  - Design traversal algorithm to find path between two vertices in the graph
  - Not concerned about length of path (for now)

Graph Representation

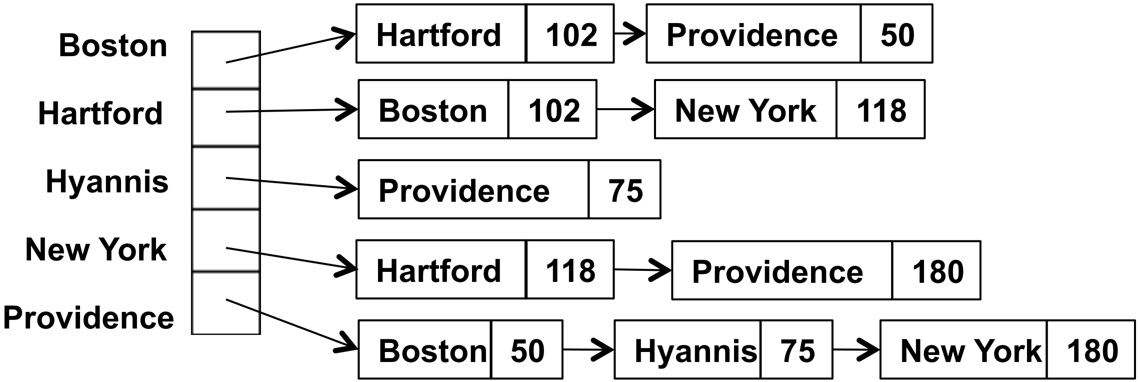
- Graph g = (V, E)
  - Set of vertices and corresponding edges (*u*, *v*)
- Adjacency Matrix Representation
  - Suitable for dense graphs with lots of edges
- Adjacency List Representation
  - Suitable for sparse graphs (such as a Maze)

Graph Representation Choices



	Boston	Hartford	Hyannis	New York	Providence
Boston	0	102	0	0	50
Hartford	102	0	0	118	0
Hyannis	0	0	0	0	75
New York	0	118	0	0	180
Providence	50	0	75	180	0

- Adjacency matrix: O(V^2) space
  - Two dimensional
  - Non-zero represents edge
  - Find edge by matrix[i][j] index
- Adjacency list: O(V+E) space
  - Array of linked lists
  - Find edge requires search



Boston → Hartford | 102 → Providence | 50

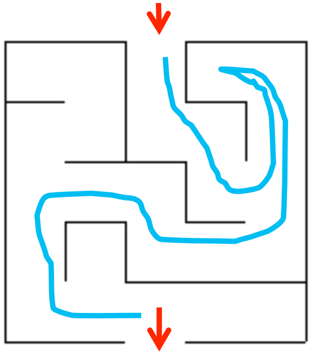
Hartford → Boston | 102 → New York | 118

Hyannis → Providence | 75

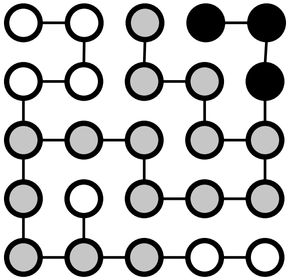
New York → Hartford | 118 → Providence | 180

Providence → Boston | 50 → Hyannis | 75 → New York | 180

Graph Representation

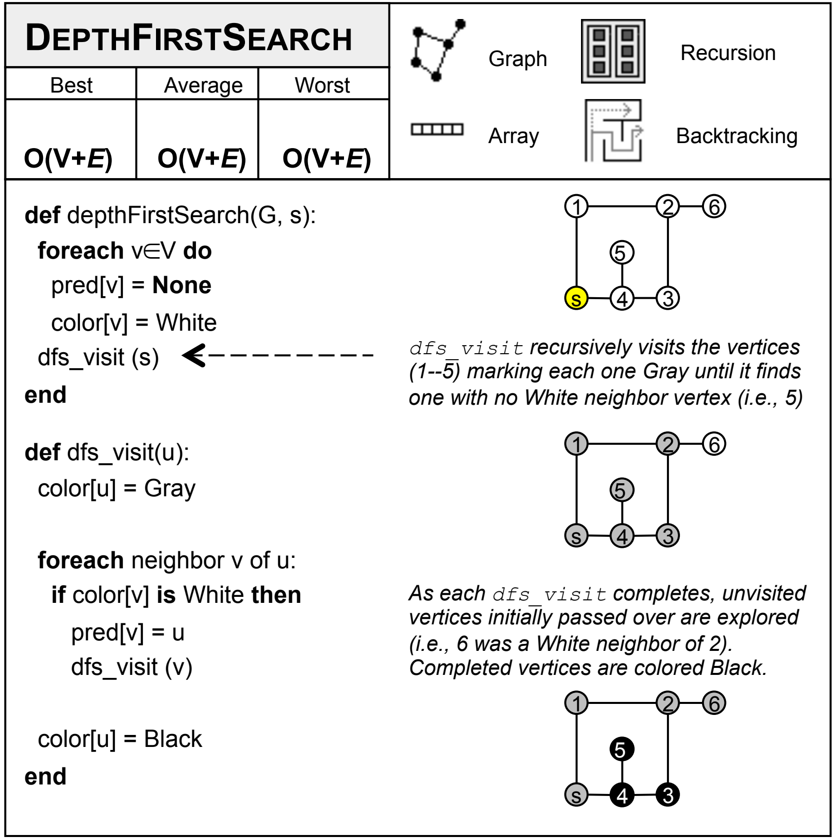


- How to traverse a graph?
  - Record vertices that have been visited
- Record colors with each vertex
  - White means not yet visited
  - Black means visited and leads to dead end
  - Gray means visited and search in progress
- Visit neighbors and backtrack when stuck



DEPTHFIRSTSEARCH Algorithm Structure

- Recursive structure
  - Forward progress reflected in vertex coloring
- Record solution with links
  - pred[u] records path
  - Let’s go to the code



DEPTHFIRSTSEARCH

Best	Average	Worst
O(V+*E*)	O(V+*E*)	O(V+*E*)

Graph
Array
Recursion
Backtracking

```
def depthFirstSearch(G, s):
  foreach v∈V do
    pred[v] = None
    color[v] = White
  dfs_visit (s)
end

def dfs_visit(u):
  color[u] = Gray

  foreach neighbor v of u:
    if color[v] is White then
      pred[v] = u
      dfs_visit (v)

  color[u] = Black
end
```

1
2
6
5
s
4
3

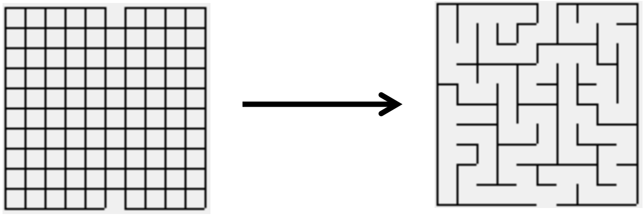
`dfs_visit` recursively visits the vertices (1--5) marking each one Gray until it finds one with no White neighbor vertex (i.e., 5)

1
2
6
5
s
4
3

As each `dfs_visit` completes, unvisited vertices initially passed over are explored (i.e., 6 was a White neighbor of 2). Completed vertices are colored Black.

Graph Project

- Creating rectangular maze
  - In interesting twist use DepthFirstSearch to search grid and remove walls
  - Tkinter Python GUI
  - Let’s go to the code



Graph Algorithms Summary

- DEPTHFIRSTSEARCH is a blind search
  - Not intended to find shortest path
- BREADTHFIRSTSEARCH will find shortest path
  - Visit vertices that are *k* edges away from initial vertex before visiting vertices *k+1* edges away
  - Only visit unmarked vertices and uses same coloring scheme as DEPTHFIRSTSEARCH