Solving Mazes Computationally in Python

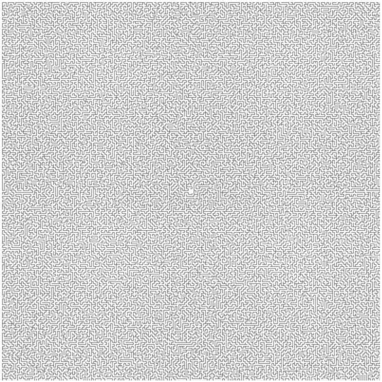
Sanjay Seshan and Bill Wang

MIT Splash

20 November 2022

The Problem

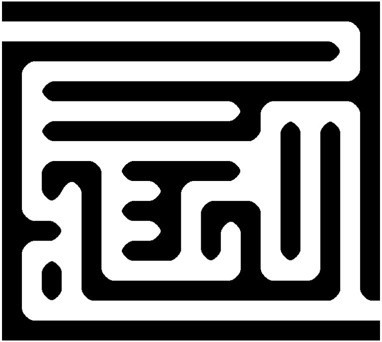
- We consider a maze to be a series of paths from a start position to an end position
- Mazes are both traditional puzzles in newspapers and much larger
- The concept of solving a maze is simple, but often hard by hand
- How can we scale this on a computer?



Basic Algorithm

Let's consider the following maze.

- Solve this using a right hand rule method/follow the wall
- This is a trivial approach – follow a single wall until the end.



Basic Algorithm

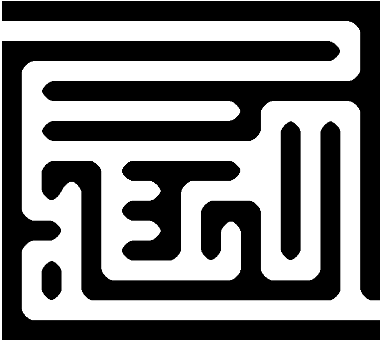
Let's consider the following maze.

- Solve this using a right hand rule method/follow the wall
- This is a trivial approach – follow a single wall until the end.

DFS Algorithm

Let's consider the following maze.

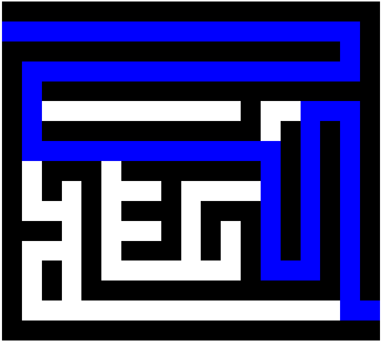
- Solve this by considering all outgoing paths at each intersection
- Select one and continue, unless a dead-end is found
- Backtrack and try again until the exit is reached
- This is called Depth First Search (DFS)



DFS Algorithm

Let's consider the following maze.

- What patterns do you notice? Does this give you the "best" path?
- Is there a better path?
- Which is the "shortest" path?



DFS Algorithm

```
procedure DFS(G, v) is
    label v as discovered
    for all directed edges from v to w that are in G.adjacentEdges(v) do
        if vertex w is not labeled as discovered then
            recursively call DFS(G, w)
```

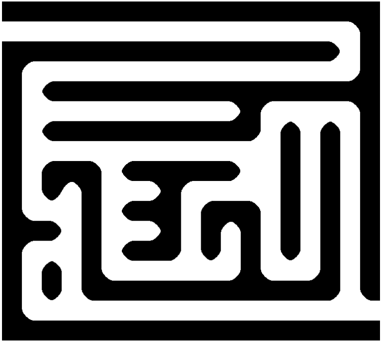
DFS Algorithm

```python
def run_dfs(pixels,curr,visited):
    for dr, dc in [(1,0),(-1,0),(0,1),(0,-1)]:
        visited.add(curr)
        new = curr[0] + dr, curr[1] + dc
        if new[0] < rdim and new[1] < cdim and new[0] >=0 and new[1] >=0:
            if pixels[new[0],new[1]] == red:
                return (curr,new)
            elif pixels[new[0],new[1]] == white:
                if new not in visited:
                    res = run_dfs(pixels, new, visited)
                    if res is not None:
                        return (curr,)+res
                    else:
                        visited.remove(curr)
    return None
pos = run_dfs(pixels,(0,1),set())
```

BFS Algorithm

Let's consider the following maze.

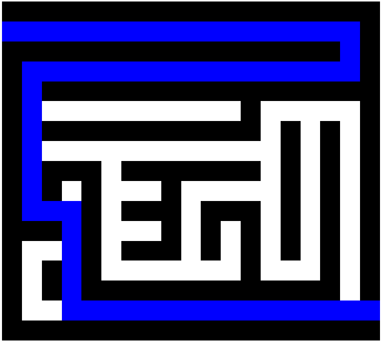
- Solve this by considering all outgoing paths at each intersection
- Try each path
- Cancel paths that do not work
- This is called Breath First Search (BFS)
- This is hard to do by hand so we will do it computationally



BFS Algorithm

Let's consider the following maze.

- This looks much better



BFS Algorithm

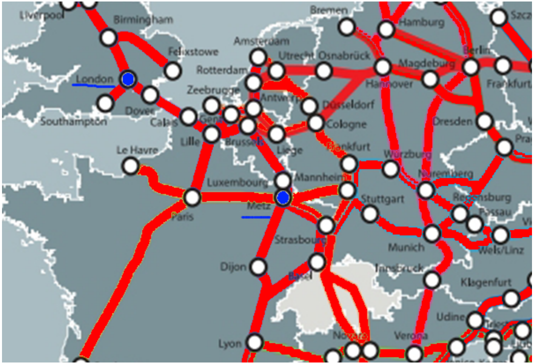
```
procedure BFS(G, root) is
    let Q be a queue
    label root as explored
    Q.enqueue(root)
    while Q is not empty do
        v := Q.dequeue()
        if v is the goal then
            return v
        for all edges from v to w in G.adjacentEdges(v) do
            if w is not labeled as explored then
                label w as explored
                w.parent := v
                Q.enqueue(w)
```

BFS Algorithm

```
def run_bfs(pixels,rdim, cdim):
    Q = [((0,1),)]
    while Q != []:
    path = Q.pop(0)
    curr = path[-1]
    for dr, dc in [(1,0),(0,1),(-1,0),(0,-1)]:
        new = curr[0] + dr, curr[1] + dc
        if  new[0] < rdim and new[1] < cdim and new[0] >=0 and new[1] >=0:
        if pixels[new[0],new[1]] == red:
            return path+(new,)
        elif pixels[new[0],new[1]] == white:
            if new not in path:
                new_path = path+(new,)
                Q.append(new_path)
```

Expanded Problem – Train routes in Europe

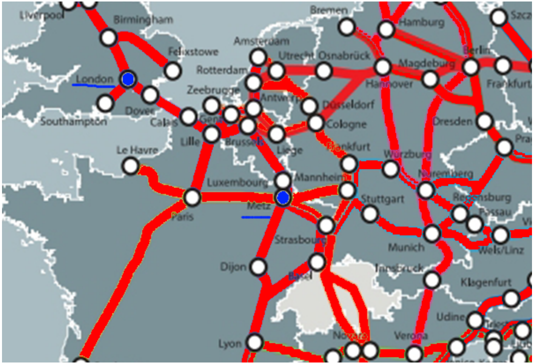
- What is the shortest path from London to Metz? Consider this as a maze with different distances between cities.
- In terms of stops?
- In terms of distance?
- Assume all trains are the same speed and there is no time lost between stops.



- Are all stops evenly spaced? What is “shortest” here?

Expanded Problem – Train routes in Europe

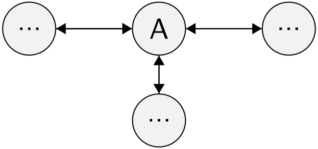
- What is the shortest path from London to Metz?
- In terms of stops? (BFS)
  London to Dover to Calais to Lille to Paris to Metz
- In terms of distance?
  London to Dover to Calais to Lille to Brussels to Luxembourg to Metz



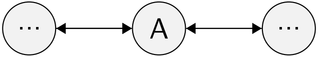
Mazes to Graphs

- Finding a path for trains is similar to solving a maze – there are possible routes and forks that must be followed
- How can we convert a maze to a graph? What is an intersection? What is a path?
- Each node is a ”vertex” and each path is an ”edge”, edges can be directed. Solving paths in graphs is similar to solving mazes. The same algorithms apply.

Intersection:

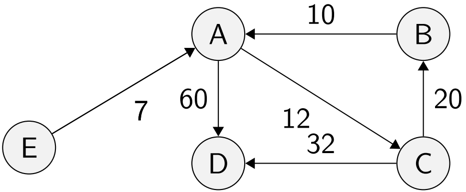


Standard path:



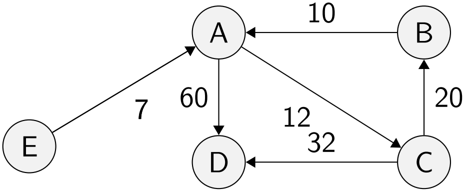
Dijkstra's Algorithm

- Consider a reduced representation – what is the shortest path from E to D?
- Let the number on the arrow be the distance from each node or city.
- The direction of the arrow maps the direction of travel permitted.
- What pattern do you see?



Dijkstra's Algorithm

- Consider a reduced representation – what is the shortest path from E to D?
  E to A to C to D



- Instead of considering each branch equally like in BFS, we consider the edge weight by taking the lowest one so far at each branch
- This is called Dijkstra's algorithm.
- Can we represent the above map, or a maze, like this?

Dijkstra's Algorithm

```
function Dijkstra(Graph, source):
    while Q is not empty:
            u <- vertex in Q with min dist[u]
            remove u from Q

            for each neighbor v of u still in Q:
                alt <- dist[u] + Graph.Edges(u, v)
                push (prev[v] <- u with dist[v] <- alt) to Q

        if u is target:
            return path prev[]
```

Dijkstra's Algorithm

```python
def run_dijkstra(nodes):
    Q = [((nodes[0],),0)]
    while Q != []:
        imin = 0
        vmin = None
        for i in range(len(Q)):
            if vmin is None or Q[i][1] < vmin:
                vmin = Q[i][1]
                imin = i
        path, dist = Q.pop(imin)
        curr = path[-1]
        for child, next_dist in curr.get_children():
            if child == nodes[-1]:
                return path+(child,)
            elif child not in path:
                new_path = path+(child,)
                Q.append((new_path,dist+next_dist))
```

Demos

(1) Simple maze using DFS
(2) Simple maze using BFS
(3) Large maze using BFS
(4) Simple graph using Dijkstra's algorithm
(5) Map of European trains using Dijkstra's algorithm
(6) Map of Cambridge using Dijkstra's algorithm

Discussion

(1) How can we use a graph to represent a maze?
(2) How can we scale graphs to solve large problems?
(3) How does solving mazes apply to the real world?
(4) How fast do these run? Can we make them faster?

Discussion

(1) How can we use a graph to represent a maze?
Consider each position as a vertex and connecting positions as vertices

(2) How can we scale graphs to solve large problems?
Represent large data as a graph that can be solved – perhaps reduce to small problems and expand

(3) How does solving mazes apply to the real world?
Finding routes uses graph algorithms in effect. Think Google Maps.

(4) How fast do these run? Can we make them faster?
For those curious, our examples are inefficient, but we can make these run in $O(|V| + |E|)$ *time for BFS and DFS and* $O(|E| + |V| \log |V|)$ *time for Dijkstra's (E is the number of edges and V vertices). We can improve our array operations by using more advanced algorithms and data structures. We can create a better representation of grids. This is more for another class.*

Reference

```python
class Node(object):
    def __init__(self, name):
        self.children = []
        self.name = name
    def add_connection(self, child, distance):
        self.children.append((child,distance))
    def get_children(self):
        return self.children
    def __repr__(self):
        return self.name
E = Node("E")
A = Node("A")
D = Node("D")
C = Node("C")
B = Node("B")
mini = [E,A,C,B,D]
E.add_connection(A,7)
D.add_connection(A,60)
A.add_connection(C,12)
C.add_connection(B,20)
B.add_connection(A,10)
C.add_connection(D,32)
```

Reference

```python
from PIL import Image

im = Image.open("maze_bfs.png")

pixels = im.load()

rdim, cdim = im.size

red = (255,0,0, 255)
blue = (0,0,255, 255)
white = (255,255,255, 255)
black = (0,0,0, 255)

```

https://github.com/sanjayseshan/mit-splash-mazes