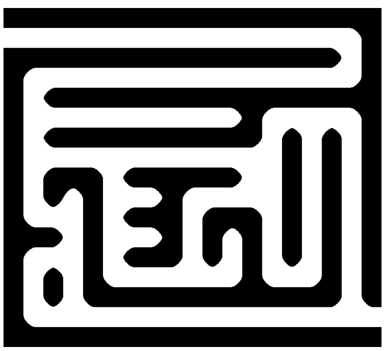
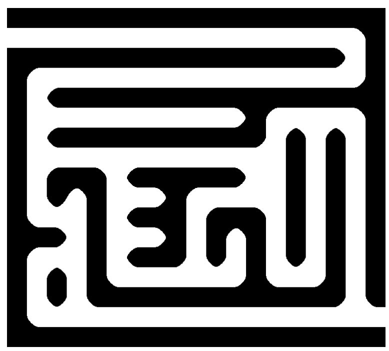
Sanjay Seshan and Bill Wang November 19, 2022

Handout for Solving Mazes Computationally

Let's consider the following maze. We will solve this using two approaches:

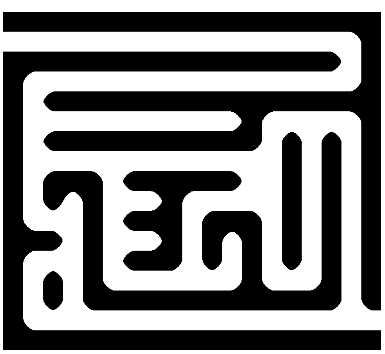
(a) Solve this using a right hand rule method/follow the wall

(b) Solve this by considering all outgoing paths at each branch one at a time (DFS – depth first search)

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

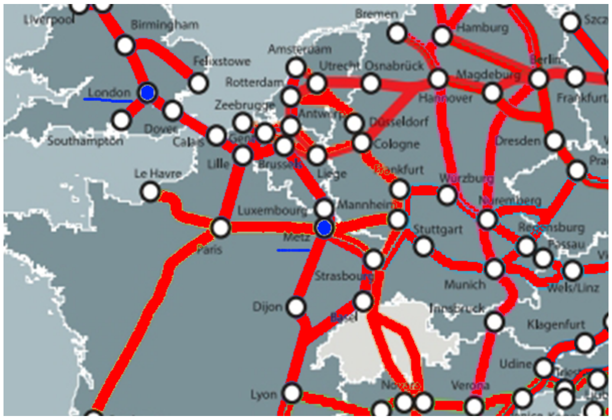
Let's try to find the best path below.



How do you decide which path to take at each intersection? How does it scale to many branches? We are developing the Breath First Search (BFS).

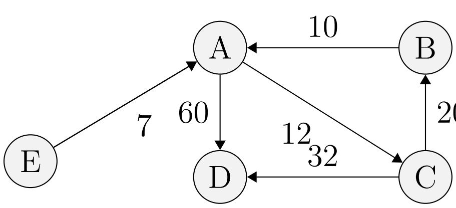
Let's consider an expanded problem – Train routes in Europe What is the shortest path from London to Metz? In terms of stops? In terms of distance?

Assume all trains are the same speed and there is no time lost between stops.



Are all stops evenly spaced? What is "shortest" here?

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?



This is called Dijkstra's algorithm.

Can we represent the above map, or a maze, like this?

```python
def run_dfs(pixels,curr,visited):
    for dr, dc in [(___,___),(___,___),(___,___),(___,___)]:
        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]] == ______:
                return ______,______
            elif pixels[new[0],new[1]] == ______:
                if new not in visited:
                    res = _________________________
                    if res is not None:
                        return (curr,)+______
                    else:
                        _________________________
    return None
pos = run_dfs(pixels,(0,1),set())
```

```python
def run_bfs(pixels,rdim,cdim):
    Q = [((0,1),)]
    while Q != []:
        path = Q.pop(___)
        curr = path[-1]
        for dr, dc in [(___,___),(___,___),(___,___),(___,___)]:
            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]] == ______:
                    return _________________________
                elif pixels[new[0],new[1]] == ______:
                    if _________________:
                        new_path = path+_________________
                        _________________________
pos = run_bfs(pixels,rdim,cdim)
```

```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(_________)
        curr = path[-1]
        for child, next_dist in curr.get_children():
            if child == nodes[-1]:
                return _________________________
            elif child not in path:
                new_path = _________________________
                Q.append(_________________________)
pos = run_dijkstra(mini)
```

References:

```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)
```

```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