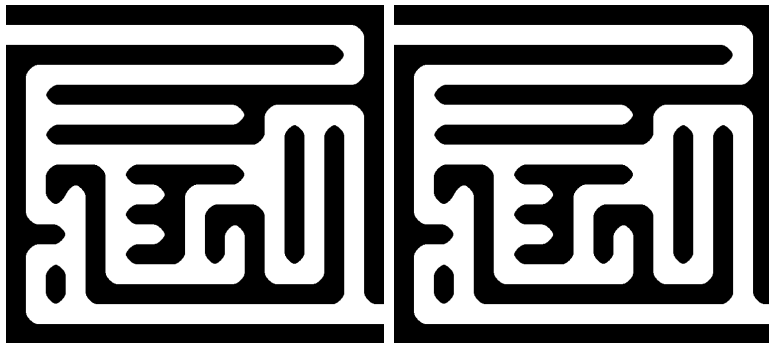


Handout for Solving Mazes Computationally

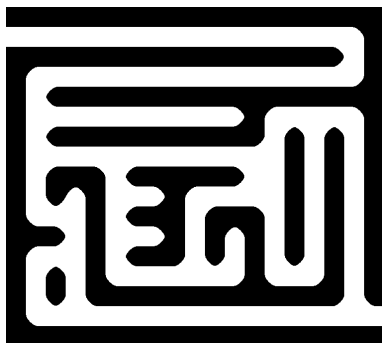
Let's consider the following maze. We will solve this using two approaches:

- (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 Breadth First Search (BFS).

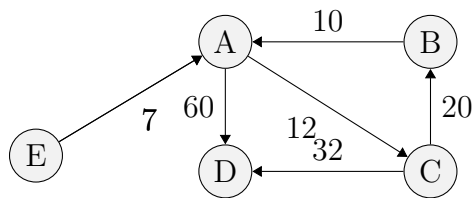
Let's consider an expanded problem – Train routes in Europe What is the shortest path from London to Metz? In terms of stops? In terms of distance?

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?

```

1 def run_dfs(pixels,curr,visited):
2     for dr, dc in [(____,____),(____,____),(____,____),(____,____)]:
3         visited.add(curr)
4         new = curr[0] + dr, curr[1] + dc
5         if new[0] < rdim and new[1] < cdim and new[0] >=0 and new[1] >=0:
6             if pixels[new[0],new[1]] == ____:
7                 return _____
8             elif pixels[new[0],new[1]] == ____:
9                 if new not in visited:
10                    res = _____
11                    if res is not None:
12                        return (curr,)+_____
13                    else:
14                        _____
15 return None
16 pos = run_dfs(pixels,(0,1),set())

1 def run_bfs(pixels,rdim,cdim):
2     Q = [(0,1),]
3     while Q != []:
4         path = Q.pop(____)
5         curr = path[-1]
6         for dr, dc in [(____,____),(____,____),(____,____),(____,____)]:
7             new = curr[0] + dr, curr[1] + dc
8             if new[0] < rdim and new[1]<cdim and new[0]>=0 and new[1]>=0:
9                 if pixels[new[0],new[1]] == ____:
10                    return _____
11                elif pixels[new[0],new[1]] == ____:
12                    if _____:
13                        new_path = path+_____
14                    _____
15 pos = run_bfs(pixels,rdim,cdim)

1 def run_dijkstra(nodes):
2     Q = [(nodes[0],),0]
3     while Q != []:
4         imin = 0
5         vmin = None
6         for i in range(len(Q)):
7             if vmin is None or Q[i][1] < vmin:
8                 vmin = Q[i][1]
9                 imin = i
10        path, dist = Q.pop(_____)
11        curr = path[-1]
12        for child, next_dist in curr.get_children():
13            if child == nodes[-1]:
14                return _____
15            elif child not in path:
16                new_path = _____
17                Q.append(_____)
18 pos = run_dijkstra(mini)

```

References:

```
1 class Node(object):
2     def __init__(self, name):
3         self.children = []
4         self.name = name
5
6     def add_connection(self, child, distance):
7         self.children.append((child,distance))
8
9     def get_children(self):
10        return self.children
11
12    def __repr__(self):
13        return self.name
14
15 E = Node("E")
16 A = Node("A")
17 D = Node("D")
18 C = Node("C")
19 B = Node("B")
20 mini = [E,A,C,B,D]
21 E.add_connection(A,7)
22 D.add_connection(A,60)
23 A.add_connection(C,12)
24 C.add_connection(B,20)
25 B.add_connection(A,10)
26 C.add_connection(D,32)
```

```
1 from PIL import Image
2
3 im = Image.open("maze_bfs.png")
4
5 pixels = im.load()
6
7 rdim, cdim = im.size
8
9 red = (255,0,0, 255)
10 blue = (0,0,255, 255)
11 white = (255,255,255, 255)
12 black = (0,0,0, 255)
```