Maze path search

Mustafa Sadiq

May 17, 2021

Introduction

Given a maze with obstacles, the goal is to find a path from top left corner to bottom right corner using different search algorithms:

- Depth first search
- Breadth first search
- A* search

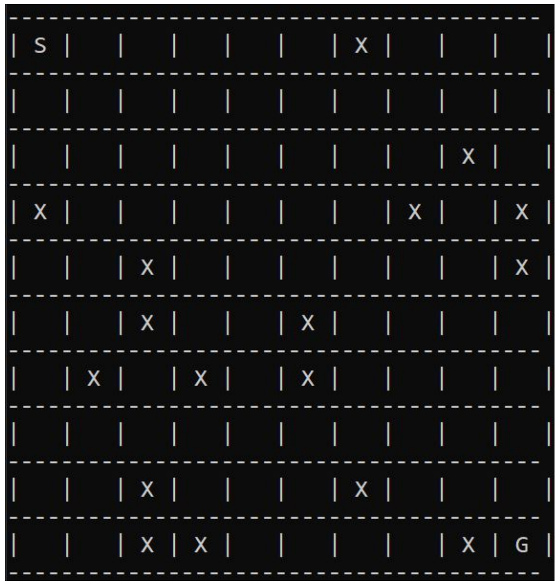
A maze is generated given a dimension and obstacle density p as follows:

```
maze = np.full([dim, dim], ' ', dtype=str)

for x in range(dim):

    for y in range(dim):
        if (rand.random() <= p): maze[x][y] = 'X' # obstacle
```

Libraries required: *numpy*, *random*, *deque from collections*, *heapq*, *sqrt from math*

Maze

For the purpose of this report, we will demonstrate all type of searches on the following maze with dimension 10×10 and obstacle density $p = 0.3$.



S						X			
								X	
X							X		X
		X							X
		X			X				
	X		X		X				
		X				X			
		X	X					X	G

Figure 1: Original maze, $dim = 10 \times 10$, $p = 0.3$

S : start

G : goal

X : obstacle

Depth first search

```
fringe = [start]
tree = dict()
tree[start] = None

while fringe:
    current = fringe.pop()
    if current == goal:
        current = goal
        path = []
        while current != start:
            path.append(current)
            current = tree[current]
        path.append(start)
        path.reverse()
        return path
    for neighbour in get_neighbours(maze, current):
        if neighbour not in tree:
            fringe.append(neighbour)
            tree[neighbour] = current

return None
```

Breadth first search

```
fringe = deque()
fringe.append(start)
tree = dict()
tree[start] = None

while fringe:
    current = fringe.popleft()
    if current == goal:
        current = goal
        path = []
        while current != start:
            path.append(current)
            current = tree[current]
        path.append(start)
        path.reverse()
        return path
    for neighbour in get_neighbours(maze, current):
        if neighbour not in tree:
            fringe.append(neighbour)
            tree[neighbour] = current

return None
```

A* search

```
fringe = []
heapq.heappush(fringe, (0,start))
tree = dict()
cost_tree = dict()
tree[start] = None
cost_tree[start] = 0

while  fringe:
    current = heapq.heappop(fringe)[1]
    if current == goal:
        current = goal
        path = []
        while current != start:
            path.append(current)
            current = tree[current]
        path.append(start)
        path.reverse()
        return path
    for neighbour in get_neighbours(maze, current):
        new_cost = cost_tree[current] + 1
        if neighbour not in cost_tree or new_cost < cost_tree[neighbour]:
            cost_tree[neighbour] = new_cost
            priority = new_cost + distance(goal, neighbour)
            heapq.heappush(fringe, (priority, neighbour))
            tree[neighbour] = current

return None
```

Results

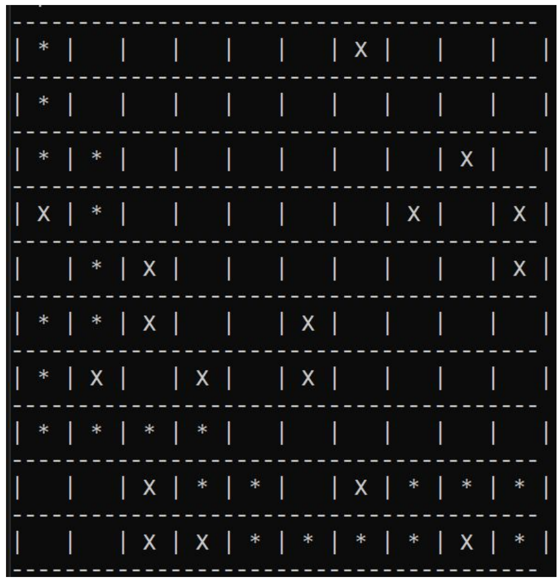


Figure 2: Path returned by Depth first search

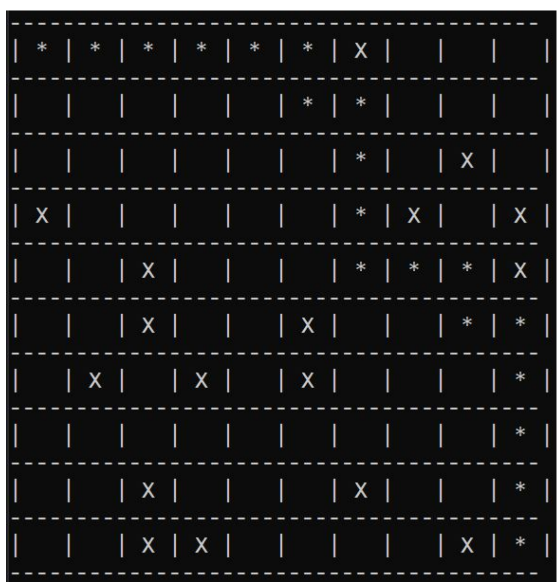


Figure 3: Path returned by Breadth first search

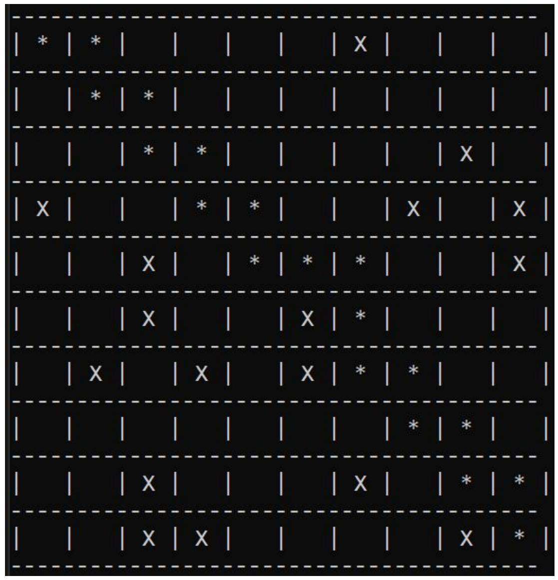


Figure 4: Path returned by A* search

- The path returned by Depth first search is not shorter than Breadth first search
- The path returned by Breadth first search is shorter than Depth first search
- The path returned by A* search is not shorter than Breadth first search