

S

EXPLORE

TRACKS

MY COURSES

EDPRESSO

REFER A FRIEND

educative

From Python to Numpy

0% COMPLETED

Search Course

Coding Example: Find shortest path in a maze

Coding Example: Find shortest path in a maze (Breadth-First approach)

Coding Example: Find shortest path in a maze (Bellman-Ford approach)

Coding Example: Fluid Dynamics

Coding Example: Blue Noise Sampling

Coding Example: Blue Noise Sampling using DART method

Coding Example: Blue Noise Sampling using Bridson method

Conclusion

Coding Example: Find shortest path in a maze (Breadth-First approach)

In this lesson, we will implement the solution of finding shortest path in the maze that we build in the previous lesson. To find the shortest path, we will use the BFS strategy.

We'll cover the following

- Breadth-First Algorithm
 - Step 1: Implement a Graph Class
 - Step 2: Implement Breadth-First Algorithm
- Complete Solution

Breadth-First Algorithm

The breadth-first (as well as depth-first) search algorithm addresses the problem of finding a path between two nodes by examining all possibilities starting from the root node and stopping as soon as a solution has been found (destination node has been reached). This algorithm runs in linear time with complexity in $O(|V| + |E|)$ (where V is the number of vertices, and E is the number of edges).

Writing such an algorithm is not especially difficult, provided you have the right data structure. In our case, the array representation of the maze is not the most well-suited and we need to transform it into an actual graph as proposed by [Valentin Bryukhanov](#).

Step 1: Implement a Graph Class

```
1 def build_graph(maze):
2     height, width = maze.shape
3     graph = {(i, j): [] for j in range(width)
4             for i in range(height) if not maze[i][j]}
5     for row, col in graph.keys():
6         if row < height - 1 and not maze[row + 1][col]:
7             graph[(row, col)].append(("S", (row + 1, col)))
8             graph[(row + 1, col)].append(("N", (row, col)))
9         if col < width - 1 and not maze[row][col + 1]:
10            graph[(row, col)].append(("E", (row, col + 1)))
11            graph[(row, col + 1)].append(("W", (row, col)))
12     return graph
```

Note: If we had used the depth-first algorithm, there is no guarantee to find the shortest path, only to find a path (if it exists).

Step 2: Implement Breadth-First Algorithm

Once this is done, writing the breadth-first algorithm is straightforward. We start from the starting node and we visit nodes at the current depth only (breadth-first, remember?) and we iterate the process until reaching the final node, if possible. The question is then: do we get the shortest path exploring the graph this way? In this specific case, “yes”, because we don’t have an edge-weighted graph, i.e. all the edges have the same weight (or cost).

```
1 def breadth_first(maze, start, goal):
2     queue = deque([(start, start)])
3     visited = set()
4     graph = build_graph(maze)
5
6     while queue:
7         path, current = queue.popleft()
8         if current == goal:
9             return np.array(path)
10        if current in visited:
11            continue
12        visited.add(current)
13        for direction, neighbour in graph[current]:
14            p = list(path)
15            p.append(neighbour)
16            queue.append((p, neighbour))
17     return None
```

Complete Solution

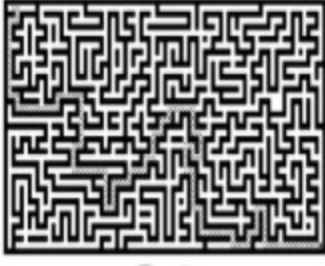
Let’s merge all this logic into one code and visualize the maze, gradient, and the shortest path! Run the following code and once the output is generated, zoom-in to have a clearer view at the shortest path.

```
1 # -----
2 # From Numpy to Python
3 # Copyright (2017) Nicolas P. Rougier - BSD license
4 # More information at https://github.com/rougier/numpy-book
5 # -----
6 import numpy as np
7 from collections import deque
8 import matplotlib.pyplot as plt
9 from scipy.ndimage import generic_filter
10
11
12 def build_maze(shape=(65,65), complexity=0.75, density = 0.50):
13     """
14     Build a maze using given complexity and density
15
16     Parameters
17     =====
18
19     shape : (rows,cols)
20     | Size of the maze
21
22     complexity: float
23     | Mean length of islands (as a ratio of maze size)
24
25     density: float
26     | Mean numbers of highland (as a ratio of maze surface)
27
28     """
```

RUN

SAVE

RESET



In the next lesson, we will try to solve this same problem using the Bellman-Ford approach!

Mark as Completed

← Back

Next →

Coding Example: Find shortest path in...Coding Example: Find shortest path in...

Stuck? Get help on

DISCUSS

Send feedback

Recommend