Weekly Challenge 09: Pathfinding

CS/CE 412/471 Algorithms: Design and Analysis

Spring 2025

Purpose

In this WC, you will explore *graph-based pathfinding* using *breadth-first search (BFS)* and *depth-first search (DFS).* You will implement these algorithms to find paths in a randomly generated maze-like graph, and visualize the search process. This task will deepen your understanding of *graph traversal techniques* and *visualization in Python*, skills that are essential in fields such as *AI, robotics*, and *network analysis.*

Skills

This WC builds your skills in computational problem-solving, algorithmic reasoning, and graph algorithms and efficient search strategies. These align with the university's learning goals, the CS program's learning outcomes, and the course's learning objectives.

Problems similar to the one described in this WC arise commonly in game development, shortest path problems, network routing, and AI-driven decision-making.

Applying computer science concepts and computational thinking to real world problems in this manner will make you a more capable and versatile computer scientist.

Specifically, this WC develops expertise in:

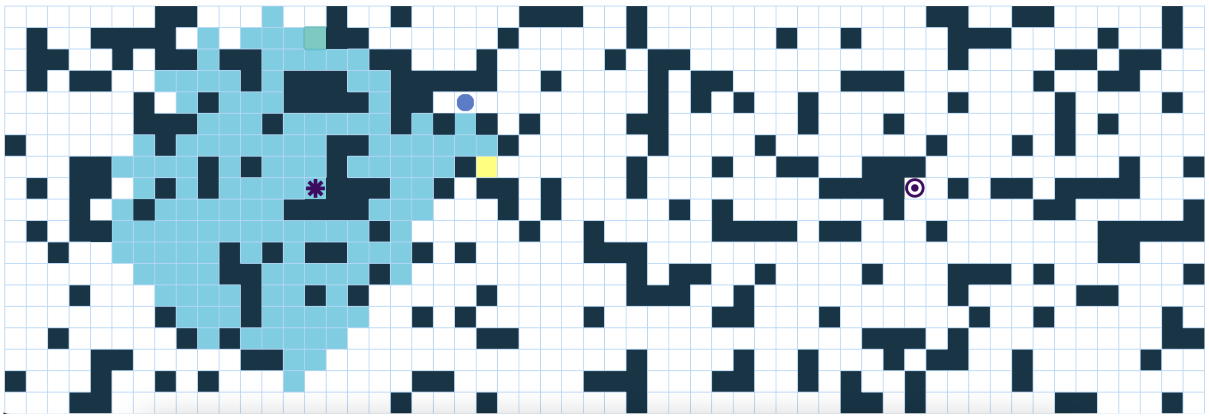
- Algorithm Design and Analysis: implementing and comparing BFS and DFS for pathfinding.
- Graph Representation and Manipulation: using `networkx` to construct and modify graph structures
- Visualization and Animation: utilizing `matplotlib` to visualize search strategies step by step.
- Problem Solving: analyzing and debugging pathfinding behavior in randomly generated mazes.

Background and Requirements

To attempt and submit the WC, you will require:

- Understanding of graph properties such as connectivity and adjacency.
- Knowledge of BFS and DFS algorithms
- The ability to apply algorithmic thinking to solve a problem
- The ability to program in Python and to read and follow technical tutorials
- Experience with the `networkx` and `matplotlib` packages for working with graphs and generating visualizations.
- Ability to structure and modularize code for readability and debugging.

A dead end is just a good place to turn around!



Graphs are a popular structure in computer science and many types of graphs have been defined. For example, a wide variety is provided by the `networkx` package in Python.

We are going to consider a graph as a maze. Nodes represent positions and each egde represents a valid move from one position to another. Randomly assigning one node as the source and another as the destination, we will use two different strategies to find a path from the source to the destination. We will visualize each of the search strategies through an animation by coloring the source and destination nodes, the nodes considered in the search, and, finally, the nodes in the discovered path.

Tasks

You will have to choose a graph to serve as your maze. Visualize different graph types from the `networkx` package in Python in order to finalize one that is to your satisfaction.

Once you have chosen your graph, you have to write a program to perform the following tasks.

1. Initialize your graph. If the graph is connected, then randomly remove some edges, e.g., by generating a random number, z_i, in $[0, 1]$ at each edge, e_i, and deleting e_i if z_i is below some predefined probability, p. The resulting graph must have a reasonable number of nodes (at least 20) and edges (at least 50).

2. In your graph, randomly assign one node as the source and another as the destination. Make sure that the nodes are not isolated.

3. Use breadth-first search to find a path from the source node to the destination node. The path is found and the search terminates when the destination node is visited for the first time. You will have to keep track of the visited nodes and of the discovered path for your visualization which is defined below.

4. Use depth-first search to find a path from the source node to the destination node. The path is found and the search terminates when the destination node is visited for the first time. You will have to keep track of the visited nodes and of the discovered path for your visualization which is defined below.

5. Visualize your search as an animation as follows. Each frame of the animation shows your maze and a step of your search. In the first frame, the source and destination nodes are visualized in different colors and the search is about to commence. In each step, depending on the current strategy, the search visits a new node or backtracks from a visited node. The search terminates when the destination is visited and the search is successful or no further nodes can be visited and the search has failed. In case of a successful search, color the nodes on the path from the source to the destination.

Example

See the Pathfinding Visualizer as an example. The animation has more features than required for this WC. For example, it finds a path on a grid whereas we are using a graph, visited nodes change shape and color whereas ours will retain their shape and only get colored one during the search, etc.

Hints

Use the `matplotlib` package for visualization and saving animations as in previous WCs.

Submission

You will submit

- your python file:
  - Make sure that it only `import`s built-in packages, `networkx`, and `matplotlib`.
  - Ensure a global variable, `STRATEGY`, at the top of the file that can be set to `BFS` or `DFS` to decide the strategy.
  - Ensure a global variable, `SAVE`, at the top of the file. When `True`, running your python script will save the animation to a file `search.mp4`, otherwise the animation is launched on the desktop.
- an MP4 file, `bfs.mp4`, showing a breadth-first search.
- an MP4 file, `dfs.mp4`, showing a depth-first search.