NSYSU-MATH Data Structure – Spring 2024

Homework 5

Design: Maze Solver Design Using Graph Searching Algorithms

Data Preparation

For this assignment, you will be provided with a zip file named `HW5.zip`, which contains template files and public test data. Your primary objective is to implement the `search_from_graph_bfs()` and `search_from_graph_dijkstra()` functions using the `Maze` class, using either Python or C++. Below is an overview of the directory structure and the contents in the zip file:

1. Python Implementation (`Py`/ directory):
   - ✓ `maze_graph.py`: This is where you will implement your `search_from_graph_bfs()` and `search_from_graph_dijkstra()` functions.
   - ✓ `*.txt`: Includes test data files for mazes and paths.
2. C++ Implementation (`Cpp/` directory):
   - ✓ `maze_graph.cpp:` This is where you will implement your `search_from_graph_bfs()` and `search_from_graph_dijkstra()` functions.
   - ✓ `*.txt:` Test data files for mazes and paths.
   - ✓ *.h: Header files used for graphical representations and additional functionality.

Description

In our recent lessons, we've explored how to model problems using graphs and utilizing existing graph-based algorithms. This assignment requires you to apply this concept by designing a maze solver that employs breadth-first search (BFS) and the Dijkstra algorithm, similar to the one used in our textbook. In addition, we have provided the `build_graph()` and the reference implementation for DFS on this graph in `search_from_graph_dfs()`. The assignment is structured into three main parts as follows:

1. Read the implementation of `build_graph()` and `search_from_graph_dfs()` in `maze_graph.py` (for Python) or `maze_graph.cpp` (for C++). Explain how these two functions work in your own words in the report.
2. Function Implementation:
   - ✓ Implement the function named `search_from_graph_bfs()` and

`search_from_graph_dijkstra()` in the provided template file. For Python, use `maze_graph.py`. For C++, use `maze_graph.cpp`.

3. Discussion:
   - ✓ Discuss the difference between the three algorithms and their pros and cons. Discuss whether they yield the same solutions.

Specifications

1. Function name: `search_from_graph_bfs()` and `search_from_graph_dijkstra()`
2. Input: The function takes three parameters: `maze`, which is a custom class we define representing the maze; `graph,` which is the graph representation of the maze and `start_row, start_col,` which are the starting coordinates.
3. Output: The function should return a Boolean value to indicate whether a path has been found. It should also return the found path from start to exit as a list or vector of tuples/pairs, representing the row and column coordinates. **In addition, the `distance` variable should be correctly handled.**
4. **The `maze` is stored as a matrix, as detailed in our textbook.**
5. **Use the `list` (in Python) or a `vector` (in C++) to store the found path. The path should be a list of tuples that store the row and column coordinates as tuples. In C++, you should use the `vector` of pairs. (e.g. [(3,4), (3,5)….])**
6. The breadth-first search and Dijkstra's algorithm should be used for the implementations. You need to use the graph built by the `build_graph`() function, where the key of the vertex is in the format of "r-c" (1-1, 3-4, 3-5, …).
7. You are allowed to use only the standard libraries of Python or C++. In addition, use the `Queue`() and `PriorityQueue()` classes from our provided code base.
8. For this assignment, do not focus on visualization/color of node or closing/discovering times; your primary goal is to ensure the correct path and Boolean value are returned. However, if you are interested, feel free to explore the GUI code or display your path.

Usage of the programs

1. Use `python .\maze.py .\maze1.txt Algorithm` to execute the program and run the specific maze solver with the specified maze file.
2. Use `python .\maze.py .\maze1.txt .\correct_path1.txt Algorithm` to compare the solution generated by your program against a correct path file, verifying the implementation's accuracy.
3. Use `python .\maze.py .\maze1.txt Algorithm -nogui` if you wish to run the program without the graphical user interface, suitable for environments where GUI support is unavailable or unnecessary.

4. Ensure to add `-O2 -lgdi32` flags when compiling your C++ program (pass them to the linker) on Windows to optimize the execution and include necessary libraries for graphics support. Ensure your GCC compiler version is above 7.0 to guarantee compatibility with C++14 and newer versions of the STL that the CTurtle library requires. For compilation assistance, feel free to reach out to us.

Deliverables

1. Deadline: 2024/6/09 (Sun.), 11:59 PM. Hand in the following two items to the cyber universities. Please see our Facebook group for the late policy and rules.
2. Report:
   - ✓ Read the implementation of `build_graph()` and `search_from_graph_dfs()` in `maze_graph.py` (for Python) or `maze_graph.cpp` (for C++). Explain how these two functions work in your own words and explain it in the report.
   - ✓ Describe the design of your program and the data structures you utilized. Discuss what you have learned from completing this homework.
   - ✓ Analyze the differences between difference between the three algorithms and their pros and cons. Discuss whether they yield the same solutions.
3. Program Source Files:
   - ✓ Submit your source files and report according to the instructions stated here. **Ensure that you follow the provided template files**.
   - ✓ Source File Comments: Each file must begin with three lines of comments indicating the Author, Date, and Purpose of the program. Include appropriate comments throughout your code for clarity.

Grading Policy

- Function Correctness: 60% (36% for public test cases and 24% for hidden test cases).
- Report and discussion: 40%.

Reference

1. https://runestone.academy/ns/books/published/pythonds3/Recursion/ExploringaMaze.html
2. https://medium.com/@saverio3107/solving-mazes-with-breadth-first-search-bfs-or-depth-first-search-dfs-ee4d10861288
3. https://sqlpad.io/tutorial/python-maze-solver/
4. https://hackmd.io/@rd2865OAQZSLjri24DYcow/B1zalLE6u