EN843407 - 2568/2 - Programming Assignment #1

Search Problems

In a spiritual realm torn by war, the legendary Captain Commander of the Gotei 128, **Genrūsaion Shonekuni Yumyummoto (**ゲンルーサイオンショネクニヤムヤムモト**)**, embarked on a critical mission. This was no ordinary treasure hunt, but a race to secure a Reishi collection point vital for saving his world from an invading army. A battle-hardened veteran with thousands of years of experience, he navigated through a dark, magical forest created by the enemy to trap him. The twisting illusions and heavy pressure of the battlefield challenged his resolve, but with eyes of steel, the Commander pressed on. As his fiery aura illuminated the dark canopy, his gaze hardened, for the objective awaited him, hidden deep within the enemy lines.

Shonekuni wielded an ancient sword capable of summoning the ultimate fire. With a mere swing, its flames could turn the thickest magical barriers and towering trees to ash in an instant, clearing a direct path through the blocked terrain. However, this destructive power placed a heavy burden on his energy reserves. He could not burn down the whole forest, as he needed to save his strength for the final battle against the enemy leader. Therefore, he may use his **Bankai: Zanka no Tachi (**ザンカノタチ**)** to burn down any obstacle or terrain in its path, but it can be used **only once** (**YOBO**: You Only Burn Once).

In this high-stakes race, Shonekuni faces stiff competition from the Sternritter who are also rushing toward the Reishi collection point. Seeking the most efficient path is no longer a luxury but a necessity for survival. He requires your assistance to implement search algorithms to outmaneuver the enemy forces and secure the objective before the Soul Society falls.

The map of the forest can be represented as a M x N grid, where M is the number of rows and N is the number of columns. The grid consists of four types of symbols:

- “S” : The starting location
- “G” : The location of the treasure
- “X” : Trees that cannot be walked through
- “1”, “2”, … “9” : The empty path where Shonekuni can walk through, but the number shows the stickiness of the ground indicating how many minutes he needs to walk through that location.

Here is an example of the map of the forest of size 6 x 7 (M = 6, N = 7):

X	X	X	X	X	X	X
X	S	5	1	2	6	X
X	1	X	1	X	2	X
X	2	X	X	X	G	X
X	3	2	4	2	1	X
X	X	X	X	X	X	X

Figure 1

Unfortunately, you do not have access to the map of the forest. The only way you can know about the type of a cell is to call the function ***explore(i, j)*** , and the function will return the value of the cell at the ***i*** th row and ***j*** th column in a string format. For example, in the map above,

- explore(3 , 1) returns “2”
- explore(2 , 4) returns “X”
- explore(3 , 5) returns “G”

You can assume that every cell at the border of the grid is always a tree “X”, (i = 0, j = 0, i = N-1, j = M-1), and travelling outside the map is not permitted.

To ensure your code integrates with the Soul Society's mainframe, all search functions must be implemented as methods inside a single class named ***Search***.

Task #1: DFS (150 points)

In this task, you will implement a depth-first **graph search** algorithm to find a solution path from the starting point to the goal. Your job is to implement two functions: ***initialize_dfs()*** and ***dfs().***

- The function ***initialize_dfs(M, N, Sr, Sc, Gr, Gc)*** will be called once at the beginning to provide information about the size of the grid, M x N, the location of the starting point, (Sr, Sc), and the location of the goal, (Gr, Gc)
- Then, the function **dfs()** will be called to start searching using the DFS algorithm. Your function must call the function ***explore(i, j)*** **in the correct order**.
    - When expanding a state, the order of the cells to be explored must be **the right cell, the top cell, the left cell, and then the bottom cell**.
    - The same cell must not be explored twice.
    - travelling outside the map is not permitted.
    - The starting cell must not be explored.
    - The goal cell must be explored once, without making any more calls after that, before your program terminates.

Task #2: BFS (150 points)

In this task, you will implement a breadth-first **graph search** algorithm to find a solution path from the starting point to the goal. Your job is to implement two functions: ***initialize_bfs()*** and ***bfs().*** All the conditions are the same as specified in Task #1, but now the function ***explore(i, j)*** must be called **in the correct order** using the BFS algorithm.

Task #3: A* Search (700 points)

In this task, you will implement an **A*** search algorithm to find an **optimal** path from the starting point to the goal. You will design and implement your own heuristic function. Your job is to implement two functions: ***initialize_Astar()*** and ***Astar().***

- The function ***initialize_Astar(M, N, Sr, Sc, Gr, Gc, K)*** will be called once at the beginning to provide information about the size of the grid, M x N, the location of the starting point, (Sr, Sc), the location of the goal, (Gr, Gc), and the number of grids that Zanka no Tachi can burn down, K.
- The function ***Astar()*** will be called to start searching using the A* algorithm. Unlike the previous tasks, your function must return **the shortest distance** from starting location to the goal. Your function does not have to call the function ***explore(i, j)*** in any specific order, but **the fewer number of times you call the function, the higher score you will receive**.

Starting Code:

You are provided with several Python files, but **the ONLY file you can modify is *search.py*** . The other files are only for testing your code, and will not be used during the grading.

Examples:

Example #1 - test_example_bfs.py

The BFS algorithm is used on the example maze in Figure 1. The order of the explored nodes are shown below

X	X	X	X	X	X	X
X	**S**	a	c	f	j	X
X	b	d	g	k	n	X
X	e	h	l		**G**	X
X	i	m	o			X
X	X	X	X	X	X	X

Example #2 - test_example_dfs.py

The DFS algorithm is used on the example maze in Figure 1. The order of the explored nodes are shown below

X	X	X	X	X	X	X
X	**S**	a	c	e	g	X
X	b	d	f	h	i	X
X					**G**	X
X						X
X	X	X	X	X	X	X

Example #3 - test_example_Astar1.py

The A* algorithm is used on the example maze in Figure 1. The length of Zanka no Tachi is zero. The shortest path is shown below; the distance is 15. The order of the explored nodes depends on your heuristic function, and is not shown here.

X	X	X	X	X	X	X
X	**S**	5	1	2	6	X
X	**1**	X	1	X	2	X
X	**2**	X	X	X	**G**	X
X	**3**	**2**	**4**	**2**	**1**	X
X	X	X	X	X	X	X

Example #4 - test_example_Astar2.py

The A* algorithm is used on the example maze in Figure 1. The length of Zanka no Tachi is one. The shortest path is shown below; the distance is 9 by cutting the tree in the green cell. The order of the explored nodes depends on your heuristic function, and is not shown here.

X	X	X	X	X	X	X
X	**S**	**5**	**1**	2	6	X
X	1	X	**1**	**0**	**2**	X
X	2	X	X	X	**G**	X
X	3	2	4	2	1	X
X	X	X	X	X	X	X

Example #5 - test_example_Astar3.py

The A* algorithm is used on the example maze in Figure 1. The length of Zanka no Tachi is three. The shortest path is shown below; the distance is 3 by cutting trees in the green cells. The order of the explored nodes depends on your heuristic function, and is not shown here.

X	X	X	X	X	X	X
X	S	5	1	2	6	X
X	1	X	1	X	2	X
X	2	0	0	0	G	X
X	3	2	4	2	1	X
X	X	X	X	X	X	X

Example #6 - test_example_Astar4.py

The A* algorithm is used on the left maze below. The length of Zanka no Tachi is two. The shortest path is shown below; the distance is 7 by cutting the two spaces in the green cells. (Zanka no Tachi can cut literally anything; hence, they do not have to be trees). The order of the explored nodes depends on your heuristic function, and is not shown here.

X	X	X	X	X	X	X
X	2	6	8	5	G	X
X	S	X	1	X	3	X
X	X	X	X	X	X	X

X	X	X	X	X	X	X
X	2	0	0	5	G	X
X	S	X	1	X	3	X
X	X	X	X	X	X	X

Grading Criteria:

For each task, your program will be tested on different test cases with a variety of grids, and you will receive a score based on the correctness and the efficiency of your program. **Your program will be tested with a time limit of 15 seconds per test case and a memory limit of 256 MB.**

For Task #1 (DFS) and Task #2 (BFS), you will receive a score on each test case only if your program calls the function ***explore(i, j)*** in the correct order.

- (20 points) 20% of the test cases will be small grids (5 <= M,N <= 15)
- (40 points) 40% of the test cases will be medium-sized grids (50 <= M,N <= 150)
- (40 points) 40% of the test cases will be large grids (500 <= M,N <= 1000)

For Task #3, you will receive a score on each test case only if your function returns the correct shortest distance between the starting location and the goal. Then, your function will be evaluated on the efficiency, where the baseline is the manhattan distance as the heuristic function.

- You will receive 20% of the total points of the test case if your function returns the correct shortest distance but make too many calls to the function **explore(i,j)**
- You will receive 50% of the total points of the test case if your function returns the correct shortest distance and make roughly the same number of calls to the function **explore(i,j)** as the baseline
- You will receive up to 100% of the total points of the test case if your function returns the correct shortest distance and makes a significantly less number of calls to the function **explore(i,j)** compared to the baseline.

Here are the details of the test cases for Task #3:

- (150 points) 300 <= M,N <= 500, K = 0
- (200 points) 500 <= M,N <= 800, K = 1
- (250 points) 500 <= M,N <= 1000, 2 <= K <= 5
- (100 points) 800 <= M,N <= 1000, 20 <= K <= 50

Submission:

As automatic scripts will be used for grading, to ensure that your submission will be graded, please follow the instructions below:

- Compile the following files into **one .zip file**
    - Your ***search.py*** file (do not rename the file), with your name and ID at the top
    - README file explaining about the heuristic function that you use, with your name and ID at the top
- Name your zipped file as ***<ID>.zip*** (without a hyphen)***.*** For example, ***1234567890.zip***

Important Notes:

- You are encouraged to discuss with your friends about how to solve this assignment, but please do NOT share your code, or tell others what to type.
- You may not use a code generator tool, such as ChatGPT, GitHub Copilot, or other similar software. The purpose of this assignment is to develop your understanding and implementation skills in search algorithms; relying on automated code generation undermines the learning process and creative problem-solving skills.

If you have any questions, please feel free to contact the instructor sarunpa@kku.ac.th