Middle East Technical University Department of Computer Engineering

CENG 462

Artificial Intelligence

Fall '2022-2023

Homework 2

Due date: 13 November 2022, Sunday, 23:55

1 Objectives

In the first assignment, you'd been asked to implement uninformed search methods (Breadth-First Search (BFS), Depth-First Search (DFS), Uniform-Cost Search (UCS)) with the `TREE-SEARCH` algorithm. This assignment aims to broaden the set of search methods (so far) with an informed search method, namely A*, along with the `GRAPH-SEARCH` algorithm.

2 Problem Definition

Informed search methods aim to guide a search operation with some additional information to attain the objective of a problem more quickly than uninformed search methods. In order to retain the optimality of solutions, the information incorporated has to conform to some specific requirements.

In this assignment, you are expected to implement Uniform-Cost Search (UCS) and the A* search with the `GRAPH-SEARCH` algorithm in order to solve path problems.

2.1 UCS Part

You are working in a cargo company. Employees of the company build an agent that goes from the start to the exit point in Figure 1 ,and the path should be optimal. So, you need to use UCS again. In other words, the problem's solution is the path with the smallest number of steps(the red path, as shown in Figure 2). The agent cannot use black cells. In other words, the black cells are obstacles. Four possible movements (actions) can be applied to the agent. Left-action, right-action, up-action, and down-action mean the agent moves to the square on the left, right, up, and down, respectively.

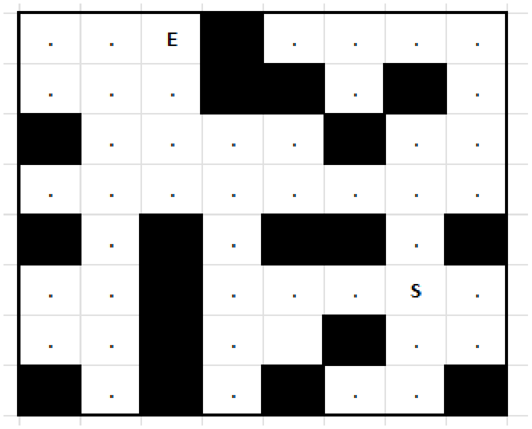


Figure 1: A sample environment of the problem.

- S is the start point and E is the exit point.
- S and E can be anywhere in environment
- Environment always be a square.
- Agent cannot pass through from black cells.
- Agent can move one cell at every turn.

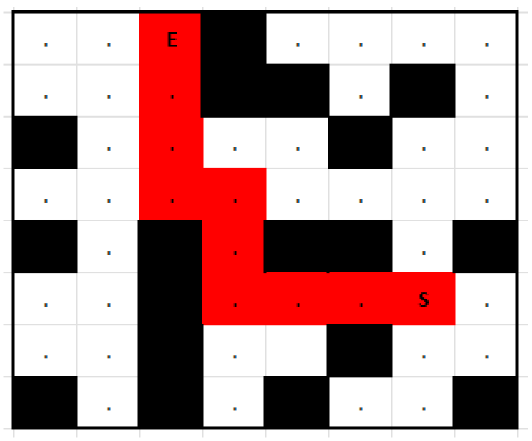


Figure 2: One of the optimal solutions of the problem depicted in Figure 1.

2.2 A* Part

The difference from the UCS part is that the company give the heuristic values of cells. In Figure 3, the cells have heuristics values. It means that alongside cost of agent you need to use heuristics values. In Figure 4, the red path is one of the optimal solution.

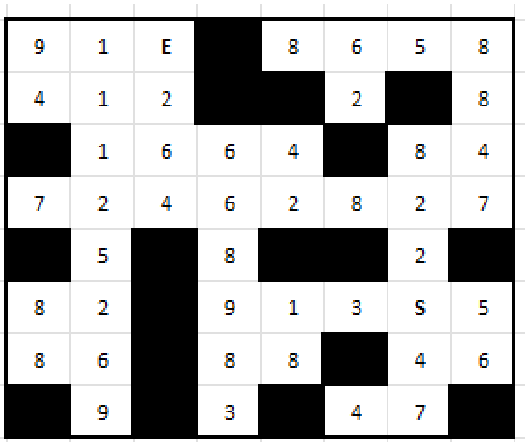


Figure 3: A sample environment of the problem.

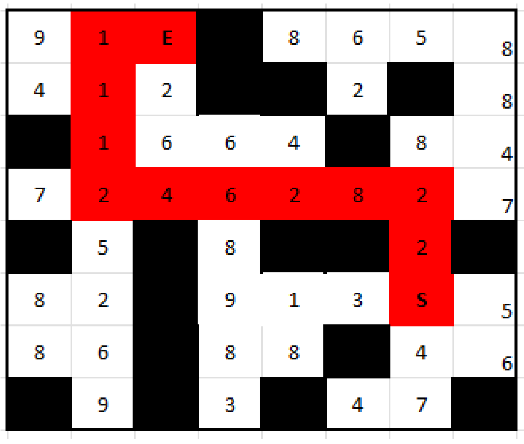


Figure 4: One of the optimal solutions of the problem depicted in Figure 3

3 Specifications

- You are going to implement UCS and A* search in Python 3.
- Each problem is represented with a file and this file will be fed to your implementation along with one of the methods.
- To unify all methods in a single place, you are expected to write the following function:

  ```python
  def InformedSearch(method_name, problem_file_name):
  ```

  The parameter specifications are as follows:

  – `method_name`: specifies which method to be run for the given problem. It can be assigned to one of the values from `["UCS", "AStar"]`.

  – `problem_file_name`: specifies the path of the problem file to be solved. File names are dependent on the problem specified and take values such as "sampleUCS1.txt", "sampleUCS2.txt", "sampleAstar1.txt", "sampleAstar2.txt".

  During the evaluation process, this function will be called and returned information will be inspected. The returned information is dependent on the method fed and as follows:

  – `method_name=="UCS"`: `UnInformedSearch` returns the optimal solution sequence, the list of processed nodes/states during search. If no solution is found then it returns a single `None`.

  – `method_name=="AStar"`: The same output structure as UCS should be generated.

  NOTE: For inspecting the algorithms' behavior, the order of nodes/states/locations in the processed nodes/states/locations list is important so a generic node expansion strategy should be adapted. This strategy is as follows: for both types of problems while expanding a node, (left-action, up-action, right-action, down-action) order should be followed. That is to say, first the child node generated as the result of applying the left-action should be processed, then the up-action's, and so forth.

  Each action (for both problem types) is considered to incur a cost value of 1.

- UCS problems are described in ".txt" files and have the following structure:

  ```
  . . E # . . . .
  . . . # # . # .
  # . . . . # . .
  . . . . . . . .
  # . # . # # . #
  . . # . . . S .
  . . # .   # . .
  # . # . # . . #
  ```

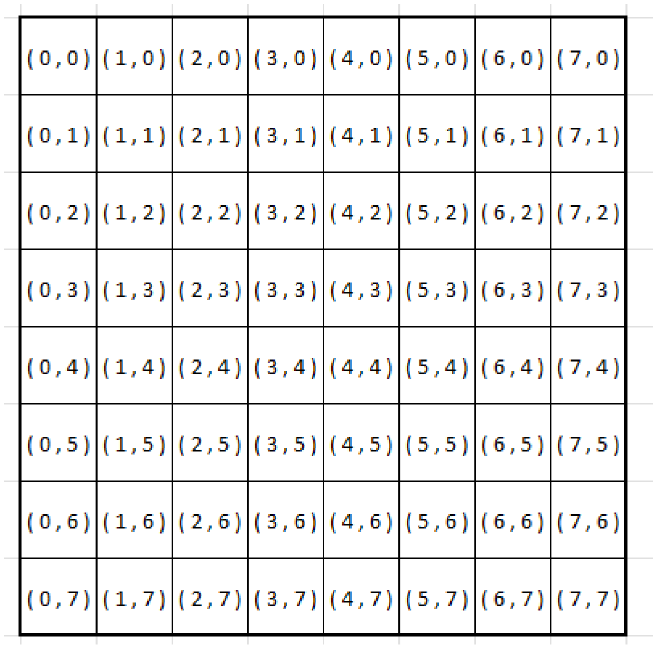
  There are four cell types. The S is the start cell, and the E is the exit cell. The "." (dot) cells are the useable. The `'#'` symbol represents the obstacles. It means agent cannot pass these cells.

- A* problems are described in ".txt" files too and have the following structure:

  ```
  9 1 E # 8 6 5 8
  4 1 2 # # 2 # 8
  # 1 6 6 4 # 8 4
  7 2 4 6 2 8 2 7
  # 5 # 8 # # 2 #
  8 2 # 9 1 3 S 5
  8 6 # 8 8 # 4 6
  # 9 # 3 # 4 7 #
  ```

There are four cell types. The S, E and `'#'` are the same cells type as in the UCS problem. Instead of "." dot cells, in this problem we have values for heuristics.

Both problem have same structure. The coordinates are represented as "(a,b)" where a and b are values of the x-axis and y-axis, respectively. The environment is described and It is divided into cells. Cell types mentioned above. The origin (of the coordinate system, (0, 0)) is located at the top leftmost cell. x and y coordinate values increase rightwards and downwards, respectively as shown in Figure 5.



(0,0)	(1,0)	(2,0)	(3,0)	(4,0)	(5,0)	(6,0)	(7,0)
(0,1)	(1,1)	(2,1)	(3,1)	(4,1)	(5,1)	(6,1)	(7,1)
(0,2)	(1,2)	(2,2)	(3,2)	(4,2)	(5,2)	(6,2)	(7,2)
(0,3)	(1,3)	(2,3)	(3,3)	(4,3)	(5,3)	(6,3)	(7,3)
(0,4)	(1,4)	(2,4)	(3,4)	(4,4)	(5,4)	(6,4)	(7,4)
(0,5)	(1,5)	(2,5)	(3,5)	(4,5)	(5,5)	(6,5)	(7,5)
(0,6)	(1,6)	(2,6)	(3,6)	(4,6)	(5,6)	(6,6)	(7,6)
(0,7)	(1,7)	(2,7)	(3,7)	(4,7)	(5,7)	(6,7)	(7,7)

Figure 5: Coordinate values on a 8x8 environment of problem. Each square represents a single cell.

Environments can be of any square size (10x10, 20x20, etc.), where the first and second numerical values on the size tuple are the width and height of the maze, respectively.

- No `import` statement is allowed and all methods should be implemented in a single solution file. You are expected to implement all the necessary operations by yourself without utilizing any external library.
- Commenting is crucial for understanding your implementation and decisions made during the process. Your implementation can be manually inspected at random or when it does not satisfy the expected outputs.

4 Sample I/O

sampleUCS.txt:

```
. . E # . . . .
. . . # # . # .
# . . . . # . .
. . . . . . . .
# . # . # # . #
. . # . . . S .
. . # .   # . .
# . # . # . . #
```

Expected outputs of the InformedSearch function with different methods on applied on sampleUCS.txt:

```
>>> import hw2solutioneXXXXXXX as hw2
>>> print(hw2.InformedSearch("UCS", "sampleUCS.txt"))
([(2, 0), (2, 1), (2, 2), (2, 3), (3, 3), (3, 4), (3, 5), (4, 5), (5, 5), (6,
    5)])
```

sampleAstar.txt:

```
9 1 E # 8 6 5 8
4 1 2 # # 2 # 8
# 1 6 6 4 # 8 4
7 2 4 6 2 8 2 7
# 5 # 8 # # 2 #
8 2 # 9 1 3 S 5
8 6 # 8 8 # 4 6
# 9 # 3 # 4 7 #
```

Expected outputs of InformedSearch function with different methods applied on sampleAstar.txt:

```
>>> import hw2solutioneXXXXXXX as hw2
>>> print(hw2.InformedSearch("AStar", "sampleAstar.txt"))
([(2, 0), (1, 0), (1, 1), (1, 2), (1, 3), (2, 3), (3, 3), (4, 3), (5, 3), (6,
    3), (6, 4), (6, 5)])
```

The optimal action sequence is represented as an array each element of which defines the next location coordinate to visit by starting from the start location (both start and exit coordinates should be included). The processed search tree nodes are the location coordinates traversed during the search and shown partially (due to their number).

5 Regulations

1. **Programming Language:** You must code your program in Python 3. Since there are multiple versions and each new version adds a new feature to the language, in order to concur on a specific version, please make sure that your implementation runs on İnek machines.

2. **Implementation:** You have to code your program by only using the functions in the standard module of python. Namely, you **cannot** import any module in your program. But you can import priority queue.

3. **Late Submission:** No late submission is allowed. Since we have a strict policy on submissions of homework in order to be able to attend the final exam, please pay close attention to the deadlines.

4. **Cheating: We have zero-tolerance policy for cheating**. People involved in cheating (any kind of code sharing and codes taken from the internet included) will be punished according to the university regulations.

5. **Discussion:** You must follow ODTUClass for discussions and possible updates on a daily basis. If think that your question concerns everyone, please ask them on ODTUClass.

6. **Evaluation:** Your program will be evaluated automatically using the "black-box" technique so make sure to obey the specifications. A reasonable timeout will be applied according to the complexity of test cases. This is not about the code efficiency, its only purpose is avoiding infinite loops due to an erroneous code. In case your implementation does not conform to expected outputs for solutions, it will be inspected manually so commenting will be crucial for grading.

6 Submission

Submission will be done via OdtuClass system. You should upload a **single** python file named in the format **hw2_e<your-student-id>.py** (i.e. hw2_e1234567.py).

7 References

- For the `GRAPH-SEARCH` algorithm, you can refer to the lecture's textbook.
- Announcements Page
- Discussions Page