Assignment 1

(Last Update: 3 Feb)

**Introduction**

Download the code for this assignment [here](https://drive.google.com/file/d/1oBgO0AJj7wtKLC6_VQhG1P2Rb65J0rvp/view?usp=sharing) and then unzip the archive.

This assignment uses [python 3](https://www.python.org/downloads/). Do not use python 2.

You can work on the assignment using your favorite python editor. We recommend [VSCode](https://en.wikipedia.org/wiki/Visual_Studio_Code).

Post any questions or issues with this assignment to our discussion forum. Alternatively you may also contact your TA directly.

**Problem 1: DFS-GSA**

In this part of the assignment you are going to implement

* a [parser](https://en.wikipedia.org/wiki/Parsing) to read a search problem in the file parse.py, and
* Depth First Search (DFS) - Graph Search Algorithm (GSA) in the file p1.py

Both these python files have already been created for. Do not change anything that has already been implemented. Our autograder relies on the existing code. From the terminal you may run python p1.py 1 to test if your code passes the first test case.

| (base) scdirk@Dirks-Air a1 % python p1.py 1  Grading Problem 1 :  ----------> Test case 1 PASSED <---------- |
| --- |

As you can see we have passed test case 1 here. This is because we have hardcoded the solution for you in this function.

| def dfs\_search(problem):  #Your p1 code here  solution = 'Ar D C\nAr C G'  return solution |
| --- |

Note that this is exactly what the test case expects. To find out what the test case expects, you can open the file test\_cases/p1/1.sol.

| Ar D C  Ar C G |
| --- |

As you can see, the content of the file test\_cases/p1/1.sol is identical to the return value of the function dfs\_search(problem).

Note that test\_cases/p1/1.sol and other solution files consist of two lines of text that represent the following.

* Exploration order (i.e., the order in which states are added to the explored set)
* solution path (i.e., the first solution found)

You should return these two lines of text in the dfs\_search() function above.

The exploration order is Ar, D, C and the solution path is Ar, C, G. The search problem definition can be found in the file test\_cases/p1/1.prob, which we will introduce shortly.

Let’s try another test case by changing the argument to the python program to 2.

| (base) scdirk@Dirks-Air a1 % python p1.py 2  Grading Problem 1 :  ----------> Test case 2 FAILED <----------  Your solution  Ar D C  Ar C G  Correct solution  A D  A D G  Delete "r" from position 1  Add " " to position 2  Add "D" to position 3  Add "  " to position 4  Add "A" to position 5  Delete "C" from position 9  Delete "  " from position 10  Delete "A" from position 11  Delete "r" from position 12  Delete " " from position 13  Delete "C" from position 14  Delete " " from position 15 |
| --- |

We failed this test case. The correct solution can be found in test\_cases/p1/2.sol.

| A D  A D G |
| --- |

We will have to look at the \*.prob files in the test\_case/p1/ folder to load the problem definition and then determine a corresponding solution. The \*.prob files define weighted directed graphs with a single start state, a list of goal states, heuristics and arcs. For example, consider the problem definition of the first test case defined in the file test\_cases/p1/1.prob.

| start\_state: Ar  goal\_states: G  Ar 0  B 0  C 0  D 0  G 0  Ar B 1.0  Ar C 2.0  Ar D 4.0  C G 8.0 |
| --- |

The search problem is specified as follows.

line 1: start state

line 2: list of goal states separated by a space

line 3 … (n+2): (n = number of states) heuristic for each state

<state> <heuristic>

line (n+3) … end: unidirectional state transitions of the form

<start state> <end state> <cost>

Note that we don’t need the transition cost nor the heuristic for solving problem 1. However, you should implement the parsing for everything now so that you don’t have to make modifications later.

You should decide on an appropriate data structure for the problem and return it from the following function in parse.py.

| def read\_graph\_search\_problem(file\_path):  #Your p1 code here  problem = ''  return problem |
| --- |

Once your implementation of both read\_graph\_search\_problem and dfs\_search is complete you can verify that you pass the first test case as follows.

| (base) scdirk@Dirks-Air a1 % python p1.py 1  Grading Problem 1 :  ----------> Test case 1 PASSED <---------- |
| --- |

You may check if you pass all 5 test cases as follows.

| (base) scdirk@Dirks-Air a1 % python p1.py -5  Grading Problem 1 :  ----------> Test case 1 PASSED <----------  ----------> Test case 2 PASSED <----------  ----------> Test case 3 PASSED <----------  ----------> Test case 4 PASSED <----------  ----------> Test case 5 PASSED <---------- |
| --- |

Note that you should add the nodes to the frontier in the order they are listed in the \*.prob file. This is in contrast to adding nodes alphabetically which is what we did in the lecture.

You may import anything from the Python Standard Library.

**Do not import packages that are not included in Python such as numpy.**

Make sure that you pass all provided test cases before moving on to the next question.

Note that we may use novel test cases for marking. You can also design your own new test cases for testing. For our novel test cases you may assume that there is always a single start state and at least one reachable goal state. For multi-goal cases, your program is supposed to return the first one that is reached along the path.

You are allowed to copy any code from the lecture slides as a starting point to solve this assignment.

Marking of this and all other questions will be done manually. We won’t solely rely on the autograder.

**Problem 2: BFS-GSA**

In this part of the assignment you are going to implement Breadth First Search (BFS) - Graph Search Algorithm (GSA) in the file p2.py.

This should just involve copying over your DFS implementation from p1 and changing the pop() to popleft().

Once you have done this, check if you pass all test cases as follows.

| (base) scdirk@Dirks-Air a1 % python p2.py -5  Grading Problem 2 :  ----------> Test case 1 PASSED <----------  ----------> Test case 2 PASSED <----------  ----------> Test case 3 PASSED <----------  ----------> Test case 4 PASSED <----------  ----------> Test case 5 PASSED <---------- |
| --- |

**Problem 3: UCS-GSA**

In this part of the assignment you are going to implement Uniform Cost Search (UCS) - Graph Search Algorithm (GSA) in the file p3.py.

This should involve copying over your BFS implementation from p2 and changing the data structure to the priority queue.

Once you have done this, check if you pass test cases as follows.

| (base) scdirk@Dirks-Air a1 % python p3.py -6  Grading Problem 3 :  ----------> Test case 1 PASSED <----------  ----------> Test case 2 PASSED <----------  ----------> Test case 3 PASSED <----------  ----------> Test case 4 PASSED <----------  ----------> Test case 5 PASSED <----------  ----------> Test case 6 PASSED <---------- |
| --- |

Note that there are 6 test cases this time.

**Problem 4: Greedy**

In this part of the assignment you are going to implement Greedy Search - Graph Search Algorithm (GSA) in the file p4.py.

This should involve copying over your UCS implementation from p3 and making minor modifications.

Once you have done this, check if you pass test cases as follows.

| (base) scdirk@Dirks-Air a1 % python p4.py -6  Grading Problem 4 :  ----------> Test case 1 PASSED <----------  ----------> Test case 2 PASSED <----------  ----------> Test case 3 PASSED <----------  ----------> Test case 4 PASSED <----------  ----------> Test case 5 PASSED <----------  ----------> Test case 6 PASSED <---------- |
| --- |

**Problem 5: A\***

In this part of the assignment you are going to implement A\* - Graph Search Algorithm (GSA) in the file p5.py.

This should involve copying over your Greedy implementation from p4 and adding the backward cost.

Once you have done this, check if you pass test cases as follows.

| (base) scdirk@Dirks-Air a1 % python p5.py -6  Grading Problem 5 :  ----------> Test case 1 PASSED <----------  ----------> Test case 2 PASSED <----------  ----------> Test case 3 PASSED <----------  ----------> Test case 4 PASSED <----------  ----------> Test case 5 PASSED <----------  ----------> Test case 6 PASSED <---------- |
| --- |

**Problem 6: 8 Queens Local Search - Number of Attacks**

The last two problems of this assignment use a different problem. Consider the content of the file test\_cases/p6/1.prob.

| . . . . . . . .  . . . . . . . .  . . . . . . . .  . . . q . . . .  q . . . q . . .  . q . . . q . q  . . q . . . q .  . . . . . . . . |
| --- |

It defines a state of the 8 Queens problem defined in class. In this problem you will load the problem in the following function of the file parse.py.

| def read\_8queens\_search\_problem(file\_path):  #Your p6 code here  problem = ''  return problem |
| --- |

Next, you will determine the attacks for each square in the following function of the file p6.py.

| def number\_of\_attacks(problem):  #Your p6 code here |
| --- |

The number of attacks for a particular square is defined as the number of direct and indirect attacks of other queens to that square, assuming the queen in the same column would move to that square.

For the first test case the correct number of attacks can be found in the file test\_cases/p6/1.sol and is as follows.

| 18 12 14 13 13 12 14 14  14 16 13 15 12 14 12 16  14 12 18 13 15 12 14 14  15 14 14 17 13 16 13 16  17 14 17 15 17 14 16 16  17 17 16 18 15 17 15 17  18 14 17 15 15 14 17 16  14 14 13 17 12 14 12 18 |
| --- |

Once you have implemented both functions correctly you should be able to pass all test cases.

| (base) scdirk@Dirks-Air a1 % python p6.py -4  Grading Problem 6 :  ----------> Test case 1 PASSED <----------  ----------> Test case 2 PASSED <----------  ----------> Test case 3 PASSED <----------  ----------> Test case 4 PASSED <---------- |
| --- |

**Problem 7: 8 Queens Local Search - Get a Better Board**

In this problem you will return a better board in the function better\_board of the file p7.py. The better board moves one queen to the best position. Note that queens can only move to a position in the same column. If there are multiple best (i.e., lowest number of attacks) positions, you should select the first best position found if iterating row by row starting in the upper left.

Consider the first test case available in the file test\_cases/p7/1.prob

| . . . . . . . .  . . . . . . . .  . . . . . . . .  . . . q . . . .  q . . . q . . .  . q . . . q . q  . . q . . . q .  . . . . . . . . |
| --- |

and its corresponding solution available in the file test\_cases/p7/1.sol.

| . q . . . . . .  . . . . . . . .  . . . . . . . .  . . . q . . . .  q . . . q . . .  . . . . . q . q  . . q . . . q .  . . . . . . . . |
| --- |

Note that you may import existing code such as helper functions from your p6.py solution.

Once you are done, check for correctness as follows.

| (base) scdirk@Dirks-Air a1 % python p7.py -6  Grading Problem 7 :  ----------> Test case 1 PASSED <----------  ----------> Test case 2 PASSED <----------  ----------> Test case 3 PASSED <----------  ----------> Test case 4 PASSED <----------  ----------> Test case 5 PASSED <----------  ----------> Test case 6 PASSED <---------- |
| --- |

You may also use the following command to make sure you pass all test cases for this assignment.

| (base) scdirk@Dirks-Air a1 % python grader.py |
| --- |

To submit your assignment to Moodle, \*.zip the following files ONLY:

* p1.py
* p2.py
* p3.py
* p4.py
* p5.py
* p6.py
* p7.py
* parse.py

Do not zip any other files. Use the \*.zip file format.

Make sure that you have submitted the correct files.