

---

**Assignment 1****Due on April 12, 2017 (23:59:59)**

**Instructions.** There are two parts in this assignment. While the first part focuses on blind and informed/heuristic search algorithms, the second part focuses on game playing and adversarial search. The goal of this problem set is to make you understand and familiarize with game playing and search strategy algorithms. You will implement this assignment with Python2.7.

**1 PART I: Search Strategy Algorithms**

For this part of the assignment, you will implement three different pathfinding algorithms for the given game.

**1.1 Game Rules**

You have a simple game for this part of the assignment. You try to transmit a mouse (which is the agent that you control) to the cheese. In the game:

- There is only one agent (mouse) which is represented with 'M' character.
- There is only one goal (cheese) which is represented with '.' character.
- '#' character represents a solid wall which is an obstacle for mouse's movement.
- The aim is to transmit the mouse to the cheese by using the most optimal path.
- The cost of the mouse's each action is 1.
- The mouse can move in four directions: North, South, East, West. The order of the directions is the same for your search algorithms. For example, when your agent is trying to choose a direction, it should consider North first and then South, East and West respectively.

```
#####
####                                M  #
#                                #####
#####                                #
#   ####                          #
#   #####                         #
#   .                               #
#####
```

Figure 1: Example input file

## 1.2 Pathfinding

You will implement a program that takes the game environment as input and finds the optimal path to transmit the mouse to the goal. You will implement three different algorithms: iterative deepening search, breadth first search and A\* algorithm. Your implementation has to generate an output file that contains the actions of the agent for the most optimal path.

1. Iterative deepening search
2. Breadth first search
3. A\* algorithm

### Iterative deepening depth-first search algorithm (IDDFS)

IDDFS is a graph search strategy which can be considered as a combination of depth-first and breadth-first algorithms. The algorithm can be considered as the space-efficient version of the depth-first search (DFS) due to depth limit parameter in the algorithm. The method runs repeatedly with increasing depth limits until the goal is reached. The pseudocode <sup>1</sup> of the algorithm is given below.

---

```
1 function IDDFS (root);
2   for depth from 0 to inf
3     found ← DLS(root, depth)
4     if found ≠ null
5       return found
6 function DLS (node, depth);
7   if depth = 0 and node is a goal
8     return node
9   if depth > 0
10    foreach child of node
11      found ← DLS(child, depth-1)
12      if found ≠ null
13        return found
14 return null
```

---

### A\* search algorithm

As you learned in the class, for A\* algorithm you need a cost and heuristic function. The cost function is given in the game rules section. For heuristic function you are expected to use Manhattan Distance.

$$G(p, q) = \sum_{i=1}^n |p_i - q_i|$$

---

<sup>1</sup>[https://en.wikipedia.org/wiki/Iterative\\_deepening\\_depth-first\\_search](https://en.wikipedia.org/wiki/Iterative_deepening_depth-first_search)

### Breadth first search algorithm (BFS)

As you learned in the class, BFS is a blind search strategy. The pseudocode is given below:

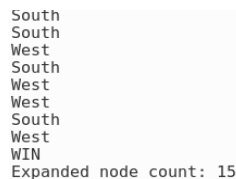
---

```
1 function Breadth-First-Search (Graph, root);  
   Input  : A graph Graph and a starting vertex root of Graph  
   Output: Goal state. The parent links trace the shortest path back to root.  
2 create empty set S  
3 create empty queue Q  
4 root.parent = null  
5 add root to S  
6 Q.enqueue(root)  
7 while Q is not empty:  
8   current = Q.dequeue()  
9   if current is the goal:  
10    return current  
11   for each node n that is adjacent to current:  
12     if n is not in S:  
13       add n to S  
14       n.parent = current  
15       Q.enqueue(n)
```

---

### 1.3 Output

The output file should contain the mouse's actions for the optimal path. You have to print the directions that your agent choose: West, South, North, East. After printing the optimal path you have to print the result of the game: If the mouse reaches the cheese, you will print 'WIN', otherwise you have to print 'LOSE'. The last line should contain the node counts that your algorithm expanded. It can vary for some algorithms due to your implementation. You should observe your algorithms' expanded node counts and comment about them in your report.



```
South  
South  
West  
South  
West  
West  
South  
West  
WIN  
Expanded node count: 15
```

Figure 2: Output file example for Part 1

## 2 PART II: Game Playing - Adversarial Search

In this part of the assignment, you are expected to implement minimax algorithm to get maximum score from the game.

## 2.1 Game Rules

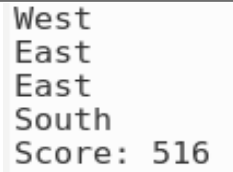
- There can be at least one more agent, which is a cat, besides the mouse. The cat agents are represented with 'C' character.
- If the mouse gets caught by a cat, the game will over.
- The goal is to finish the game by eating all the cheeses to get maximum score.
- '#' character represents a solid wall which is an obstacle for mouse's movement.
- You control the mouse agent only. Each cat agent's movement is fixed.
- The cost of the mouse's each action is 1. The mouse wins 10 points for each cheese that it eats.
- If the mouse finishes the game by eating all cheeses, it will win 500 points.
- The mouse can do one of five actions: moving West, Stop, moving East, moving North and moving South. The order of the directions is the same for your search algorithms. For example, when your agent is trying to choose a direction, it should consider moving West action first and then Stop, moving East, moving North and moving South actions respectively.
- The cat can move in four directions: West, East, North, South. The order of the directions is the same for your search algorithms. For example when your agent is trying to choose a direction, it should consider West first and then East, North and South. The cat agent can turn 0 to (+/-)90 degrees only. For example, at t=0 the cat move to EAST direction, at time t=1 it can not move to WEST direction(180 degrees) even if it is available.

```
#####
#          ###          #
#          ##           #
#                   .    #
#                   M    #
#   C   .                #
#                   #####
#                   #     #
#   .               #     #
#   .....   #####      #
#####
```

Figure 3: Example input for Part 2

## 2.2 Output

The output file should contain the mouse's actions for the path that your algorithm choose. You have to print the actions that your agent choose: Stop, West, South, North, East. The last line should contain total score of the game.



```
West
East
East
South
Score: 516
```

Figure 4: Output file example for Part 1

## NOTES

- Your implementation should take input and output files as arguments.
- Your code should be readable and you should use comment lines.
- Your minimax implementation should take depth of the tree parameter as argument.
- Your main file should be named: 'game.py'
- Your argument list:
  - -i <input file path>: Full path of the input file which contains game environment.
  - -o <output file path>: Full path of the output file that your implementation will generate.
  - -a <algorithm>: It can be 'iddfs' for iterative deepening search, 'bfs' for breadth first search, 'astar' for A\* and 'minimax' for minimax algorithm.
  - -d <depth of the tree>: This parameter is for the minimax algorithm. It determines the depth of the search tree.
- For example, your code will be run from the terminal:

```
python2.7 game.py -i input1.txt -a bfs -o output_bfs.txt
```

```
python2.7 game.py -i input2.txt -a minimax -d 2 -o output_bfs.txt
```

- If your implementation does not generate output file with the command(as above), it won't be evaluated.
- Submit format: A ZIP file contains;
  - report.pdf (PDF file containing your report)
  - code/ (directory containing game.py and the other Python files that you use)

The ZIP file will be submitted via the department's submission system. DO NOT MISS the DEADLINE.

## Grading

- Code (75) : IDDFS (15 point), BFS (10 points), A\* (20 points), Minimax(30 points).
- Report(25): You are recommended to write your report with  $\text{\LaTeX}$ .  $\text{\LaTeX}$  is not an obligation for this assignment.

**Notes for the report:** You should analyse the methods you employed. How does each algorithm work for the given problem(s)? What are the advantages and disadvantages of the search algorithms? Comment about the algorithms that you implement.

## Academic Integrity

All work on assignments must be done individually unless stated otherwise. You are encouraged to discuss with your classmates about the given assignments, but these discussions should be carried out in an abstract way. That is, discussions related to a particular solution to a specific problem (either in actual code or in the pseudocode) will not be tolerated. In short, turning in someone else's work, in whole or in part, as your own will be considered as a violation of academic integrity. Please note that the former condition also holds for the material found on the web as everything on the web has been written by someone else.