[Topic 1 Algorithms]

General instructions:

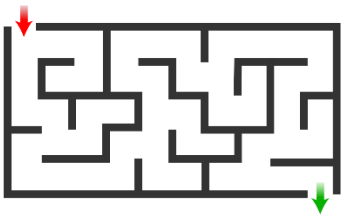
Regarding your task:

1. Download the template file (Topic1.py).
2. **Don't rename** the python file.
3. You should submit **the same python** file in addition to your documentation.
4. Submit **only running** code that you have tested before.
5. **Compressed** files (.zip/.rar) are **not allowed.**
6. Please, read the documentation carefully.
7. Clear the output after being displayed for multiple runs.
8. This project will be **auto graded**.
9. Copying or Getting code from online resources (including YouTube) is considered as cheating case.
10. **Do not change** any class functions signature (parameters, or order).
11. You can add any extra attributes, functions or classes you need as long as the **main structure is left as it is**.
12. **Implement** the given functions.
13. **Install any missing library in your package which is imported in the file and use Python 3.6.**

Plagiarism checking will be applied. Research is subject to rejection in such case. All submissions will be checked for plagiarism automatically.

Algorithm (1)

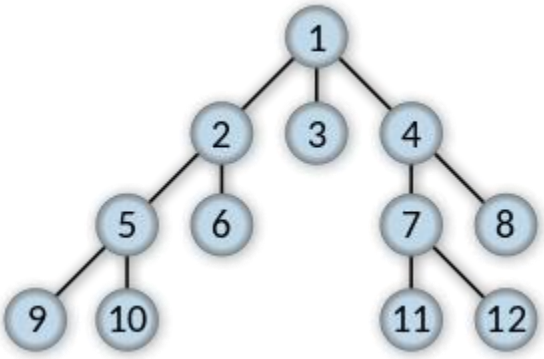
1. In this task, you are expected to solve a 2-D maze using BFS. A maze is path typically from start node ‘S’ to Goal node ‘E’.



Input: 2D maze represented as a string.

Output: the full path from Start node to End node (Goal Node) and direct path to go from Start to End directly.

Example: let’s say the end node is 6.



Full Path: 1, 2, 3, 4, 5, 6.

Path: 1, 2, 6.

The input and output are explained below. Your code should be generic for any dimension of a given maze.

```
Maze: 'S,.,.,#,.,.,. .,#,.,.,.,#,. .,#,.,.,.,.,. .,.,#,#,.,.,.
#,.,#,E,.,#,.'
```

- Maze is a string, rows are separated by **space** and columns are separated by **comma** ‘,’.
- The board is read **row wise**, the nodes are numbered **0-based**startingthe leftmost node.
- You have to create your own board **as a 2D array (NO 1D ARRAY ALLOWED)** of**Nodes**.

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

Topic1.pyfile has search algorithms region

The search algorithms region contains two classes:

a. Class Node represents a cell in the board of game. You can add extra attributes, but you do not delete current attributes or neglect them.

```
class Node:
    id = None  # Unique value for each node.
    up = None  # Represents value of neighbors (up, down, left, right).
    down = None
    left = None
    right = None
    previousNode = None  # Represents value of neighbors.
```

b. Class Search Algorithms:

```
class SearchAlgorithms:
    ''' * DON'T change Class, Function or Parameters Names and Order
        * You can add ANY extra functions,
          classes you need as long as the main
          structure is left as is '''
    path = []  # Represents the correct path from start node to the goal node.
    fullPath = []  # Represents all visited nodes from the start node to the goal node.

    def __init__(self, mazeStr, edgeCost=None):
        ''' mazeStr contains the full board
         The board is read row wise,
        the nodes are numbered 0-based starting
        the leftmost node'''
        pass

    def BFS(self):
        # Fill the correct path in self.path
        # self.fullPath should contain the order of visited nodes
        # self.path should contain the direct path from start node to goal node
        return self.fullPath, self.path
```

3. The Main Function for search algorithm:

```
def SearchAlgorithm_Main():
    searchAlgo = SearchAlgorithms('S,.,.,#,.,.,. .,#,.,.,.,#,. .,#,.,.,.,.,. .,.,#,#,.,.,. #,.,#,E,.,#,.')
    fullPath, path = searchAlgo.BFS()
    print('**BFS**\n Full Path is: ' + str(fullPath) + "\n Path: "+ str(path))
```

Algorithm (2)

2. Implement Perceptron Neural Network for **AND** Logical Function.
   The Step function is the activation function using:
   - threshold = -0.2
   - learning rate = 0.1
   - number of iterations =100

   For the following data:

   Training set input= [[0, 0],
   [1, 1],
   [1, 0],
   [0, 1]]
   Output = [0, 1, 0, 0]

   Only Test with [1, 1] expected output = 1

Topic1.pyfile has NN region

1. The neural network region contains one class:

```
# region NeuralNetwork
class NeuralNetwork():

    def __init__(self, learning_rate, threshold):
        self.learning_rate = learning_rate
        self.threshold = threshold
        self.synaptic_weights = 2 * np.random.random((2, 1)) - 1

    def step(self, x):
        pass

    def train(self, training_inputs, training_outputs, training_iterations):
        pass

    def think(self, inputs):
        pass


# endregion
```

2. The Main Function for neural network algorithm:

```
def NN_Main():
    learning_rate = 0.1
    threshold = -0.2
    neural_network = NeuralNetwork(learning_rate, threshold)

    print("Beginning Randomly Generated Weights: ")
    print(neural_network.synaptic_weights)

    training_inputs = np.array([[0, 0],
                                [0, 1],
                                [1, 0],
                                [1, 1]])

    training_outputs = np.array([[0, 0, 0, 1]]).T

    neural_network.train(training_inputs, training_outputs, 100)

    print("Ending Weights After Training: ")
    print(neural_network.synaptic_weights)

    inputTestCase = [1, 1]

    print("Considering New Situation: ", inputTestCase[0], inputTestCase[1], end=" ")
    print("New Output data: ", end=" ")
    print(neural_network.think(np.array(inputTestCase)))
    print("Wow, we did it!")
```

Algorithm (3)

3. Implement ID3 algorithm on a dataset that holds a diagnosis for the eyes of patients.
    - The diagnosis is based on the following features:
        1. Age: (0) young, (1) adult.
        2. Prescription: (0) myope, (1) hypermetrope.
        3. Astigmatic: (0) no, (1) yes.
        4. Tear production rate: (0) normal, (1) reduced.
        5. Diabetic: (0) not diabetic patient, (1) is a diabetic patient.
    - The output classes are:
        1. Need contact lenses (1): the patient should be fitted with a special type of contact lenses.
        2. No contact lenses (0): the patient should not be fitted with a
        3. Special type of contact lenses.

    - Task:
        1. Classification using ID3 algorithm.
        2. Each feature has only two attributes 0 or 1.
        3. Output classes are only two values => 0 (no special contactlenses) and 1 (need special contact lenses)

Topic1.pyfile has ID3 region

1. The ID3 region contains three classes:

```
class item:
    def __init__(self, age, prescription, astigmatic, tearRate, diabetic, needLense):
        self.age = age
        self.prescription = prescription
        self.astigmatic = astigmatic
        self.tearRate = tearRate
        self.diabetic = diabetic
        self.needLense = needLense

    def getDataset():
        data = []
        labels = [0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0]
        data.append(item(0, 0, 0, 0, 1, labels[0]))
        data.append(item(0, 0, 0, 1, 1, labels[1]))
        data.append(item(0, 0, 1, 0, 1, labels[2]))
        data.append(item(0, 0, 1, 1, 1, labels[3]))
        data.append(item(0, 1, 0, 0, 1, labels[4]))
        data.append(item(0, 1, 0, 1, 1, labels[5]))
        data.append(item(0, 1, 1, 0, 1, labels[6]))
        data.append(item(0, 1, 1, 1, 1, labels[7]))
        data.append(item(1, 0, 0, 0, 0, labels[8]))
        data.append(item(1, 0, 0, 1, 0, labels[9]))
        data.append(item(1, 0, 1, 0, 0, labels[10]))
        data.append(item(1, 0, 1, 1, 0, labels[11]))
        data.append(item(1, 1, 0, 0, 0, labels[12]))
        data.append(item(1, 1, 0, 1, 0, labels[13]))
        data.append(item(1, 1, 1, 0, 0, labels[14]))
        data.append(item(1, 1, 1, 1, 0, labels[15]))
        data.append(item(1, 0, 0, 0, 0, labels[16]))
        data.append(item(1, 0, 0, 1, 0, labels[17]))
        data.append(item(1, 0, 1, 0, 0, labels[18]))
        data.append(item(1, 0, 1, 1, 0, labels[19]))
        data.append(item(1, 1, 0, 0, 0, labels[20]))
        return data
```

```
class Feature:
    def __init__(self, name):
        self.name = name
        self.visited = -1
        self.infoGain = -1


class ID3:
    def __init__(self, features):
        self.features = features

    def classify(self, input):
        # takes an array for the features ex. [0, 0, 1, 1, 1]
        # should return 0 or 1 based on the classification
        pass
```

2. The Main Function for ID3 algorithm:

```
def ID3_Main():
    dataset = item.getDataset()
    features = [Feature('age'), Feature('prescription'), Feature('astigmatic'), Feature('tearRate'), Feature('diabetic')]
    id3 = ID3(features)
    cls = id3.classify([0, 0, 1, 1, 1])
    print('testcase 1: ', cls)
    cls = id3.classify([1, 1, 0, 0, 0])
    print('testcase 2: ', cls)
    cls = id3.classify([1, 1, 1, 0, 0])
    print('testcase 3: ', cls)
    cls = id3.classify([1, 1, 0, 1, 0])
    print('testcase 4: ', cls)
```