In [1]:
```
###1. Implement A* Search algorithm.



def aStarAlgo(start_node, stop_node):


    open_set = set(start_node)
    closed_set = set()
    g = {}
    parents = {}
    g[start_node] = 0
    parents[start_node] = start_node

    while len(open_set) > 0 :
        n = None

        for v in open_set:
            if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
                n = v

        if n == stop_node or Graph_nodes[n] == None:
            pass
        else:
            for (m, weight) in get_neighbors(n):


                if m not in open_set and m not in closed_set:
                    open_set.add(m)
                    parents[m] = n
                    g[m] = g[n] + weight





                else:
                    if g[m] > g[n] + weight:

                        g[m] = g[n] + weight

                        parents[m] = n


                        if m in closed_set:
                            closed_set.remove(m)
                            open_set.add(m)

        if n == None:
            print('Path does not exist!')
            return None



        if n == stop_node:
            path = []

            while parents[n] != n:
                path.append(n)
                n = parents[n]

            path.append(start_node)

            path.reverse()

            print('Path found: {}'.format(path))
            return path




        open_set.remove(n)
        closed_set.add(n)

    print('Path does not exist!')
    return None



def get_neighbors(v):
    if v in Graph_nodes:
        return Graph_nodes[v]
    else:
        return None



def heuristic(n):
    H_dist = {
```

```
        'A': 10,
        'B': 8,
        'C': 5,
        'D': 7,
        'E': 3,
        'F': 6,
        'G': 5,
        'H': 3,
        'I': 1,
        'J': 0
    }

    return H_dist[n]


Graph_nodes = {

    'A': [('B', 6), ('F', 3)],
    'B': [('C', 3), ('D', 2)],
    'C': [('D', 1), ('E', 5)],
    'D': [('C', 1), ('E', 8)],
    'E': [('I', 5), ('J', 5)],
    'F': [('G', 1),('H', 7)] ,
    'G': [('I', 3)],
    'H': [('I', 2)],
    'I': [('E', 5), ('J', 3)],

}
aStarAlgo('A', 'J')
```

```
Path found: ['A', 'F', 'G', 'I', 'J']
```

Out[1]:
```
['A', 'F', 'G', 'I', 'J']
```

In [1]:
```python
###2. Implement AO* Search algorithm.

class Graph:
    def __init__(self, graph, heuristicNodeList, startNode):

        self.graph = graph
        self.H=heuristicNodeList
        self.start=startNode
        self.parent={}
        self.status={}
        self.solutionGraph={}

    def applyAOStar(self):
        self.aoStar(self.start, False)

    def getNeighbors(self, v):
        return self.graph.get(v,'')

    def getStatus(self,v):
        return self.status.get(v,0)

    def setStatus(self,v, val):
        self.status[v]=val

    def getHeuristicNodeValue(self, n):
        return self.H.get(n,0)

    def setHeuristicNodeValue(self, n, value):
        self.H[n]=value


    def printSolution(self):
        print("FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE STARTNODE:",self.start)
        print("------------------------------------------------------------")
        print(self.solutionGraph)
        print("------------------------------------------------------------")

    def computeMinimumCostChildNodes(self, v):
        minimumCost=0
        costToChildNodeListDict={}
        costToChildNodeListDict[minimumCost]=[]
        flag=True
        for nodeInfoTupleList in self.getNeighbors(v):
            cost=0
            nodeList=[]
            for c, weight in nodeInfoTupleList:
                cost=cost+self.getHeuristicNodeValue(c)+weight
                nodeList.append(c)

            if flag==True:
                minimumCost=cost
                costToChildNodeListDict[minimumCost]=nodeList
                flag=False
            else:
                if minimumCost>cost:
                    minimumCost=cost
                    costToChildNodeListDict[minimumCost]=nodeList


        return minimumCost, costToChildNodeListDict[minimumCost]


    def aoStar(self, v, backTracking):

        print("HEURISTIC VALUES :", self.H)
        print("SOLUTION GRAPH :", self.solutionGraph)
        print("PROCESSING NODE :", v)

        print("-----------------------------------------------------------------------------------------")

        if self.getStatus(v) >= 0:
            minimumCost, childNodeList = self.computeMinimumCostChildNodes(v)
            self.setHeuristicNodeValue(v, minimumCost)
            self.setStatus(v,len(childNodeList))

            solved=True

            for childNode in childNodeList:
                self.parent[childNode]=v
                if self.getStatus(childNode)!=-1:
                    solved=solved & False

            if solved==True:
                self.setStatus(v,-1)
                self.solutionGraph[v]=childNodeList


            if v!=self.start:
```

```
                self.aoStar(self.parent[v], True)

            if backTracking==False:
                for childNode in childNodeList:
                    self.setStatus(childNode,0)
                    self.aoStar(childNode, False)



h1 = {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J':1, 'T': 3}
graph1 = {
    'A': [[('B', 1), ('C', 1)], [('D', 1)]],
    'B': [[('G', 1)], [('H', 1)]],
    'C': [[('J', 1)]],
    'D': [[('E', 1), ('F', 1)]],
    'G': [[('I', 1)]]
}
G1= Graph(graph1, h1, 'A')
G1.applyAOStar()
G1.printSolution()

h2 = {'A': 1, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
graph2 = {
    'A': [[('B', 1), ('C', 1)], [('D', 1)]],
    'B': [[('G', 1)], [('H', 1)]],
    'D': [[('E', 1), ('F', 1)]]
}

G2 = Graph(graph2, h2, 'A')
G2.applyAOStar()
G2.printSolution()
```

```
HEURISTIC VALUES : {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, 'T':
3}
SOLUTION GRAPH : {}
PROCESSING NODE : A
-----------------------------------------------------------------------------------------
HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, 'T':
3}
SOLUTION GRAPH : {}
PROCESSING NODE : B
-----------------------------------------------------------------------------------------
HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, 'T':
3}
SOLUTION GRAPH : {}
PROCESSING NODE : A
-----------------------------------------------------------------------------------------
HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, 'T':
3}
SOLUTION GRAPH : {}
PROCESSING NODE : G
-----------------------------------------------------------------------------------------
HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1, 'T':
3}
SOLUTION GRAPH : {}
PROCESSING NODE : B
-----------------------------------------------------------------------------------------
HEURISTIC VALUES : {'A': 10, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1, 'T':
3}
SOLUTION GRAPH : {}
PROCESSING NODE : A
-----------------------------------------------------------------------------------------
HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1, 'T':
3}
SOLUTION GRAPH : {}
PROCESSING NODE : I
-----------------------------------------------------------------------------------------
HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 0, 'J': 1, 'T':
3}
SOLUTION GRAPH : {'I': []}
PROCESSING NODE : G
-----------------------------------------------------------------------------------------
HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1, 'T':
3}
SOLUTION GRAPH : {'I': [], 'G': ['I']}
PROCESSING NODE : B
-----------------------------------------------------------------------------------------
HEURISTIC VALUES : {'A': 12, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1, 'T':
3}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE : A
-----------------------------------------------------------------------------------------
HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1, 'T':
3}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE : C
-----------------------------------------------------------------------------------------
HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1, 'T':
3}
```

```
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE : A
-----------------------------------------------------------------------------------------
HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1, 'T': 
3}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE : J
-----------------------------------------------------------------------------------------
HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 0, 'T': 
3}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G'], 'J': []}
PROCESSING NODE : C
-----------------------------------------------------------------------------------------
HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 1, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 0, 'T': 
3}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J']}
PROCESSING NODE : A
-----------------------------------------------------------------------------------------
FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE STARTNODE: A
------------------------------------------------------------
{'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J'], 'A': ['B', 'C']}
------------------------------------------------------------
HEURISTIC VALUES : {'A': 1, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH : {}
PROCESSING NODE : A
-----------------------------------------------------------------------------------------
HEURISTIC VALUES : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH : {}
PROCESSING NODE : D
-----------------------------------------------------------------------------------------
HEURISTIC VALUES : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH : {}
PROCESSING NODE : A
-----------------------------------------------------------------------------------------
HEURISTIC VALUES : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH : {}
PROCESSING NODE : E
-----------------------------------------------------------------------------------------
HEURISTIC VALUES : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 0, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH : {'E': []}
PROCESSING NODE : D
-----------------------------------------------------------------------------------------
HEURISTIC VALUES : {'A': 11, 'B': 6, 'C': 12, 'D': 6, 'E': 0, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH : {'E': []}
PROCESSING NODE : A
-----------------------------------------------------------------------------------------
HEURISTIC VALUES : {'A': 7, 'B': 6, 'C': 12, 'D': 6, 'E': 0, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH : {'E': []}
PROCESSING NODE : F
-----------------------------------------------------------------------------------------
HEURISTIC VALUES : {'A': 7, 'B': 6, 'C': 12, 'D': 6, 'E': 0, 'F': 0, 'G': 5, 'H': 7}
SOLUTION GRAPH : {'E': [], 'F': []}
PROCESSING NODE : D
-----------------------------------------------------------------------------------------
HEURISTIC VALUES : {'A': 7, 'B': 6, 'C': 12, 'D': 2, 'E': 0, 'F': 0, 'G': 5, 'H': 7}
SOLUTION GRAPH : {'E': [], 'F': [], 'D': ['E', 'F']}
PROCESSING NODE : A
-----------------------------------------------------------------------------------------
FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE STARTNODE: A
------------------------------------------------------------
{'E': [], 'F': [], 'D': ['E', 'F'], 'A': ['D']}
------------------------------------------------------------
```

In [11]:
```python
#3. For a given set of training data examples stored in a .CSV file, implement and demonstrate the
#      Candidate-Elimination algorithmto output a description of the set of all hypotheses consistent
#      with the training examples.



import csv

with open("/home/rock/Desktop/Lab AI and ML/trainingexamples.csv") as f:
    csv_file = csv.reader(f)
    data = list(csv_file)

    specific = data[1][:-1]
    general = [['?' for i in range(len(specific))] for j in range(len(specific))]

    for i in data:
        if i[-1] == "Yes":
            for j in range(len(specific)):
                if i[j] != specific[j]:
                    specific[j] = "?"
                    general[j][j] = "?"

        elif i[-1] == "No":
            for j in range(len(specific)):
                if i[j] != specific[j]:
                    general[j][j] = specific[j]
                else:
                    general[j][j] = "?"

        print("\nStep " + str(data.index(i)+1) + " of Candidate Elimination Algorithm")
        print(specific)
        print(general)

    gh = [] # gh = general Hypothesis
    for i in general:
        for j in i:
            if j != '?':
                gh.append(i)
                break
    print("\nFinal Specific hypothesis:\n", specific)
    print("\nFinal General hypothesis:\n", gh)
```

```
Step 1 of Candidate Elimination Algorithm
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Step 2 of Candidate Elimination Algorithm
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Step 3 of Candidate Elimination Algorithm
['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Step 4 of Candidate Elimination Algorithm
['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]

Step 5 of Candidate Elimination Algorithm
['Sunny', 'Warm', '?', 'Strong', '?', '?']
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final Specific hypothesis:
 ['Sunny', 'Warm', '?', 'Strong', '?', '?']

Final General hypothesis:
 [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]
```

In [7]:
```
#4. Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an
#    appropriate data set for building the decision tree and apply this knowledge toclassify a new sample.


import pandas as pd
import math

def base_entropy(dataset):
    p = 0
    n = 0
    target = dataset.iloc[:, -1]
    targets = list(set(target))
    for i in target:
        if i == targets[0]:
            p = p + 1
        else:
            n = n + 1
    if p == 0 or n == 0:
        return 0
    elif p == n:
        return 1
    else:
        entropy = 0 - (
            ((p / (p + n)) * (math.log2(p / (p + n))) + (n / (p + n)) * (math.log2(n/ (p + n)))))
        return entropy


def entropy(dataset, feature, attribute):
    p = 0
    n = 0
    target = dataset.iloc[:, -1]
    targets = list(set(target))
    for i, j in zip(feature, target):
        if i == attribute and j == targets[0]:
            p = p + 1
        elif i == attribute and j == targets[1]:
            n = n + 1
        if p == 0 or n == 0:
            return 0
        elif p == n:
            return 1
        else:
            entropy = 0 - (
                ((p / (p + n)) * (math.log2(p / (p + n))) + (n / (p + n)) * (math.log2(n/ (p + n)))))
            return entropy


def counter(target, attribute, i):
    p = 0
    n = 0
    targets = list(set(target))
    for j, k in zip(target, attribute):
        if j == targets[0] and k == i:
            p = p + 1
        elif j == targets[1] and k == i:
            n = n + 1
    return p, n



def Information_Gain(dataset, feature):
    Distinct = list(set(feature))
    Info_Gain = 0
    for i in Distinct:
        Info_Gain = Info_Gain + feature.count(i) / len(feature) * entropy(dataset,feature, i)
        Info_Gain = base_entropy(dataset) - Info_Gain
    return Info_Gain



def generate_childs(dataset, attribute_index):
    distinct = list(dataset.iloc[:, attribute_index])
    childs = dict()
    for i in distinct:
        childs[i] = counter(dataset.iloc[:, -1], dataset.iloc[:, attribute_index], i)
    return childs


def modify_data_set(dataset,index, feature, impurity):
    size = len(dataset)
    subdata = dataset[dataset[feature] == impurity]
    del (subdata[subdata.columns[index]])
    return subdata


def greatest_information_gain(dataset):
    max = -1
    attribute_index = 0
```

```
    size = len(dataset.columns) - 1
    for i in range(0, size):
        feature = list(dataset.iloc[:, i])
        i_g = Information_Gain(dataset, feature)
        if max < i_g:
            max = i_g
            attribute_index = i
    return attribute_index


def construct_tree(dataset, tree):
    target = dataset.iloc[:, -1]
    impure_childs = []
    attribute_index = greatest_information_gain(dataset)
    childs = generate_childs(dataset, attribute_index)
    tree[dataset.columns[attribute_index]] = childs
    targets = list(set(dataset.iloc[:, -1]))
    for k, v in childs.items():
        if v[0] == 0:
            tree[k] = targets[1]
        elif v[1] == 0:
            tree[k] = targets[0]
        elif v[0] != 0 or v[1] != 0:
            impure_childs.append(k)
    for i in impure_childs:
        sub = modify_data_set(dataset,attribute_index,
        dataset.columns[attribute_index], i)
        tree = construct_tree(sub, tree)
    return tree


def main():
    df = pd.read_csv("/home/rock/Desktop/Lab AI and ML/tennisdata.csv")
    tree = dict()
    result = construct_tree(df, tree)
    for key, value in result.items():
        print(key, " => ", value)


if __name__ == "__main__":
    main()
```

```
Outlook  =>  {'Sunny': (2, 3), 'Overcast': (4, 0), 'Rain': (3, 2)}
Overcast  =>  Yes
Temperature  =>  {'Mild': (2, 1), 'Cool': (1, 1)}
Hot  =>  No
Cool  =>  Yes
Humidity  =>  {'Normal': (1, 1)}
High  =>  No
Normal  =>  Yes
Windy  =>  {'Weak': (1, 0), 'Strong': (0, 1)}
Weak  =>  Yes
Strong  =>  No
```

In [12]:

```
#5. Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the
#   same using appropriate data sets.



import numpy as np
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)
X = X/np.amax(X,axis=0)
y = y/100


def sigmoid (x):
    return 1/(1 + np.exp(-x))


def derivatives_sigmoid(x):
    return x * (1 - x)


epoch=5000
lr=0.1
inputlayer_neurons = 2
hiddenlayer_neurons = 3
output_neurons = 1


wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))



for i in range(epoch):


    hinp1=np.dot(X,wh)
    hinp=hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1=np.dot(hlayer_act,wout)
    outinp= outinp1+ bout
    output = sigmoid(outinp)


    EO = y-output
    outgrad = derivatives_sigmoid(output)
    d_output = EO* outgrad
    EH = d_output.dot(wout.T)


    hiddengrad = derivatives_sigmoid(hlayer_act)
    d_hiddenlayer = EH * hiddengrad


    wout += hlayer_act.T.dot(d_output) *lr
    wh += X.T.dot(d_hiddenlayer) *lr


print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.89412853]
 [0.88284738]
 [0.89294528]]
```

In [14]:
```
#6. Write a program to implement the naïve Bayesian classifier for a sample training data set stored
#   as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.


import pandas as pd
from sklearn import tree
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import GaussianNB

data = pd.read_csv('/home/rock/Desktop/Lab AI and ML/tennisdata.csv')
print("The first 5 Values of data is :\n", data.head())
```

```
The first 5 Values of data is :
     Outlook Temperature Humidity   Windy PlayTennis
0     Sunny         Hot     High    Weak         No
1     Sunny         Hot     High  Strong         No
2  Overcast         Hot     High    Weak        Yes
3      Rain        Mild     High    Weak        Yes
4      Rain        Cool   Normal    Weak        Yes
```

In [15]:
```
X = data.iloc[:, :-1]
print("\nThe First 5 values of the train data is\n", X.head())
```

```

The First 5 values of the train data is
     Outlook Temperature Humidity   Windy
0     Sunny         Hot     High    Weak
1     Sunny         Hot     High  Strong
2  Overcast         Hot     High    Weak
3      Rain        Mild     High    Weak
4      Rain        Cool   Normal    Weak
```

In [16]:
```
y = data.iloc[:, -1]
print("\nThe First 5 values of train output is\n", y.head())
```

```

The First 5 values of train output is
 0     No
1     No
2    Yes
3    Yes
4    Yes
Name: PlayTennis, dtype: object
```

In [17]:
```
# convert them in numbers
le_outlook = LabelEncoder()
X.Outlook = le_outlook.fit_transform(X.Outlook)

le_Temperature = LabelEncoder()
X.Temperature = le_Temperature.fit_transform(X.Temperature)

le_Humidity = LabelEncoder()
X.Humidity = le_Humidity.fit_transform(X.Humidity)

le_Windy = LabelEncoder()
X.Windy = le_Windy.fit_transform(X.Windy)

print("\nNow the Train output is\n", X.head())
```

```

Now the Train output is
    Outlook  Temperature  Humidity  Windy
0        2            1         0      1
1        2            1         0      0
2        0            1         0      1
3        1            2         0      1
4        1            0         1      1

/usr/lib/python3/dist-packages/pandas/core/generic.py:5170: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html
#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy)
  self[name] = value
```

In [18]:
```
le_PlayTennis = LabelEncoder()
y = le_PlayTennis.fit_transform(y)
print("\nNow the Train output is\n",y)
```

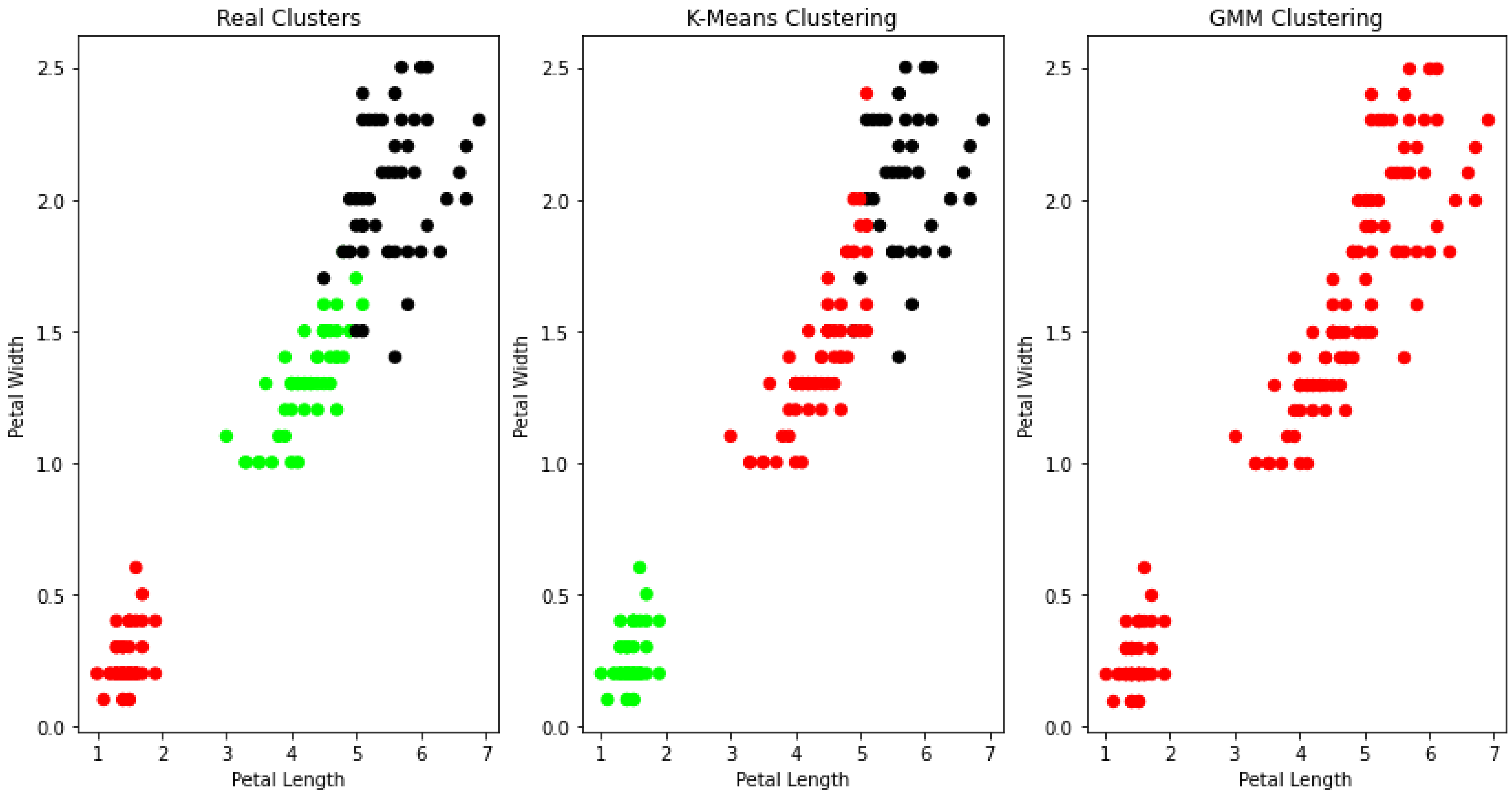
```

Now the Train output is
 [0 0 1 1 1 0 1 0 1 1 1 1 1 0]
```

In [19]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.20)

classifier = GaussianNB()
classifier.fit(X_train, y_train)

from sklearn.metrics import accuracy_score
print("Accuracy is:", accuracy_score(classifier.predict(X_test), y_test))
```

```
Accuracy is: 1.0
```

In [20]:
```
#7. Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for
#   clustering using k-Means algorithm. Compare the results of these two algorithms and comment
#   on the quality of clustering. You can add Java/Python ML library classes/API in the program.



import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import pandas as pd
import numpy as np


iris = datasets.load_iris()
X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y = pd.DataFrame(iris.target)
y.columns = ['Targets']


model = KMeans(n_clusters=3)
model.fit(X)


plt.figure(figsize=(14,7))
colormap = np.array(['red', 'lime', 'black'])


plt.subplot(1, 3, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Clusters')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')


plt.subplot(1, 3, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K-Means Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')


from sklearn import preprocessing


scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=40)
gmm.fit(xs)
plt.subplot(1, 3, 3)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[0], s=40)
plt.title('GMM Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

print('''Observation: The GMM using EM algorithm based clustering
      matched the true labels more closely than the Kmeans.''')
```

```
Observation: The GMM using EM algorithm based clustering matched the true labels more closely than the Kmea
ns.
```



Real Clusters
K-Means Clustering
GMM Clustering
Petal Width
Petal Length

In [2]:
```
#8. Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print
#   both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets


iris=datasets.load_iris()
print("Iris Data set loaded...")
x_train, x_test, y_train, y_test = train_test_split(iris.data,iris.target,test_size=0.1)


#random_state=0
for i in range(len(iris.target_names)):
    print("Label", i , "-",str(iris.target_names[i]))

classifier = KNeighborsClassifier(n_neighbors=2)
classifier.fit(x_train, y_train)
y_pred=classifier.predict(x_test)

print("Results of Classification using K-nn with K=1 ")
for r in range(0,len(x_test)):
    print(" Sample:", str(x_test[r]), " Actual-label:", str(y_test[r])," Predicted-label:", str(y_pred[r]))

    print("Classification Accuracy :" , classifier.score(x_test,y_test));
    print("\n")
```

```
Iris Data set loaded...
Label 0 - setosa
Label 1 - versicolor
Label 2 - virginica
Results of Classification using K-nn with K=1
 Sample: [5.  3.4 1.5 0.2]  Actual-label: 0  Predicted-label: 0
Classification Accuracy : 0.9333333333333333


 Sample: [6.9 3.1 5.1 2.3]  Actual-label: 2  Predicted-label: 2
Classification Accuracy : 0.9333333333333333


 Sample: [6.3 2.9 5.6 1.8]  Actual-label: 2  Predicted-label: 2
Classification Accuracy : 0.9333333333333333


 Sample: [4.6 3.6 1.  0.2]  Actual-label: 0  Predicted-label: 0
Classification Accuracy : 0.9333333333333333


 Sample: [6.  2.2 4.  1. ]  Actual-label: 1  Predicted-label: 1
Classification Accuracy : 0.9333333333333333


 Sample: [4.9 3.  1.4 0.2]  Actual-label: 0  Predicted-label: 0
Classification Accuracy : 0.9333333333333333


 Sample: [5.6 2.5 3.9 1.1]  Actual-label: 1  Predicted-label: 1
Classification Accuracy : 0.9333333333333333


 Sample: [4.3 3.  1.1 0.1]  Actual-label: 0  Predicted-label: 0
Classification Accuracy : 0.9333333333333333


 Sample: [6.4 2.8 5.6 2.2]  Actual-label: 2  Predicted-label: 2
Classification Accuracy : 0.9333333333333333


 Sample: [4.8 3.4 1.6 0.2]  Actual-label: 0  Predicted-label: 0
Classification Accuracy : 0.9333333333333333


 Sample: [7.3 2.9 6.3 1.8]  Actual-label: 2  Predicted-label: 2
Classification Accuracy : 0.9333333333333333


 Sample: [5.9 3.2 4.8 1.8]  Actual-label: 1  Predicted-label: 2
Classification Accuracy : 0.9333333333333333


 Sample: [6.4 2.7 5.3 1.9]  Actual-label: 2  Predicted-label: 2
Classification Accuracy : 0.9333333333333333


 Sample: [6.7 3.3 5.7 2.5]  Actual-label: 2  Predicted-label: 2
Classification Accuracy : 0.9333333333333333
```

In [22]:
```
###9. Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points.
#      Select appropriate data set for your experiment and draw graphs


import numpy as np
import matplotlib.pyplot as plt

def local_regression(x0, X, Y, tau):
    x0 = [1, x0]
    X = [[1, i] for i in X]
    X = np.asarray(X)
    xw = (X.T) * np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 * tau))
    beta = np.linalg.pinv(xw @ X) @ xw @ Y @ x0
    return beta

def draw(tau):
    prediction = [local_regression(x0, X, Y, tau) for x0 in domain]
    plt.plot(X, Y, 'o', color='black')
    plt.plot(domain, prediction, color='red')
    plt.show()

X = np.linspace(-3, 3, num=1000)
domain = X
Y = np.log(np.abs(X ** 2 - 1) + .5)

draw(10)
draw(0.1)
draw(0.01)
draw(0.001)
```