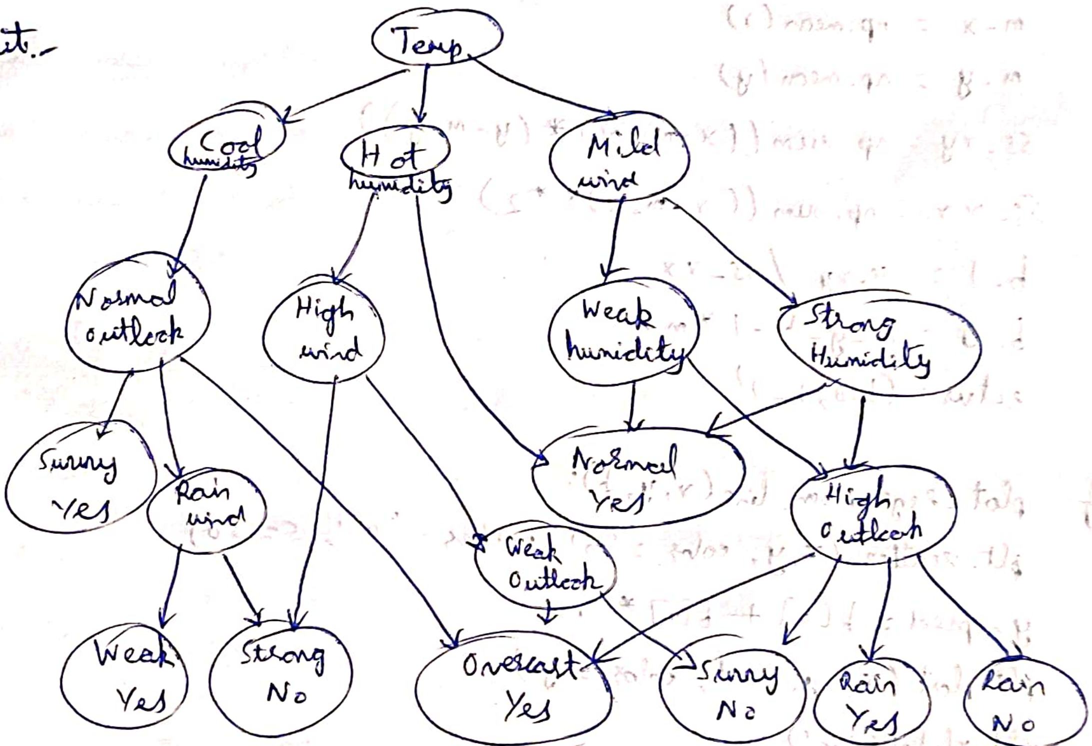
Lab-2

Implement ID3 algorithm

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import maths
import copy

dataset = pd.read_csv('tennis.csv')
x = dataset.iloc[:, :].values
x
attribute = ['Outlook', 'Temp', 'Humidity', 'Wind']


class Node(object):
    def _init_(self):
        self.value = None
        self.decision = None
        self.child = None

def findEntropy(data, rows):
    yes = 0
    no = 0
    ans = -1
    idx = len(data[0]) - 1
    entropy = 0

    for i in rows:
        if data[i][idx] == "Yes":
            yes = yes + 1
        else:
            no = no + 1
    x = yes / (yes + no)
    y = no / (yes + no)
    if x!=0 and y!=0:
        entropy = -1 *(x *math.log2(x) + y *math.log2(y))
    if x==1:
        ans =1
    if y==1;
        ans =0
    return entropy, ans


def findMaxGain(data, rows, columns):
    maxGain =0
    retidx =-1
    entropy, ans = findEntropy (data,rows)
    if entropy ==0:
        return maxGain, retidx, ans
    for j in columns:
        mydict = {}
        idx=j
        for i in rows:
            key = data[i][idx]
            if key not in mydict:
                mydict[key] =1
            else:
                mydict[key] = mydict[key] +1
        gain = entropy
        for key in mydict:
            yes =0
            no = 0
            for k in rows:
                if data[k][j]== key:
                    if data[k][-1] == 'Yes':
                        yes = yes+1
                    else:
                        no = no +1
            x = yes/(yes+no) if yes+no >0 else 0
            y = no/(yes+no) if yes+no >0 else 0
            if x>0 and y>0:
                gain -= (mydict[key]/len(rows))*(x*math.log2(x)+
                                                 y*math.log2(y))
    if gain > maxGain :
        maxGain, retidx = gain, j
return maxGain, retidx, ans


def buildTree (data, rows, columns):
    maxGain, idx, ans = findMaxGain (data, rows, columns)
    root = Node()
    if maxGain ==0:
        root.value= 'Yes' if ans==1 else 'No'
        return root

    root.value = attribute [idx]
    mydict = { data[i][idx] : [] for i in rows}
    for i in rows:
        mydict [data[i][idx]].append(i)

    new_columns = [col for col in columns if col != idx]
    for key in mydict :
        child = buildTree (data, mydict[key], new_columns)
        child.decision = key
        root.childs.append(child)
    return root


def visualize_tree(root):
    dot = graphviz.Digraph(format = 'png')
    def add_nodes_edges (node, parent_name = "Root"):
        if node:
            node_name = f"{node.decision}\n {node.value}'
            dot.node(node_name, label = node.value if node.decision is
                None else f"{node.decision}\n {node.value}")
            if parent_name != "Root":
                dot.edge(parent_name, node_name)
            for child in node.childs:
                add_nodes_edges(child, node_name)
    add_nodes_edges(root)
    dot.render('decision_tree', format='png', view=True)
```

```
def calculate():
    rows = list(range(len(x)))
    columns = list(range(len(attribute)))
    root = buildTree(X, rows, columns)
    visualize_tree(root)

calculate()
```

Output:-



Sneha S B
17/3/25