3. 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 to classify a new sample.

**ID3 Algorithm**

ID3(Examples, Target_attribute, Attributes)

Examples are the training examples. Target_attribute is the attribute whose value is to be predicted by the tree. Attributes is a list of other attributes that may be tested by the learned decision tree. Returns a decision tree that correctly classifies the given Examples.

- Create a Root node for the tree
- If all Examples are positive, Return the single-node tree Root, with label = +
- If all Examples are negative, Return the single-node tree Root, with label = -
- If Attributes is empty, Return the single-node tree Root, with label = most common value of Target_attribute in Examples

- Otherwise Begin
  - A ← the attribute from Attributes that best* classifies Examples
  - The decision attribute for Root ← A
  - For each possible value, v_i, of A,
    - Add a new tree branch below Root, corresponding to the test A = v_i
    - Let Examples_{v_i} be the subset of Examples that have value v_i for A
    - If Examples_{v_i} is empty
      - Then below this new branch add a leaf node with label = most common value of Target_attribute in Examples
      - Else below this new branch add the subtree
        ID3(Examples_{v_i}, Target_attribute, Attributes – {A})
  - End
- Return Root

* The best attribute is the one with highest information gain
ENTROPY:
Entropy measures the impurity of a collection of examples.

\[ \text{Entropy}(S) = -p_+ \log_2 p_+ - p_- \log_2 p_- \]

Where, \( p_+ \) is the proportion of positive examples in \( S \)
\( p_- \) is the proportion of negative examples in \( S \).

INFORMATION GAIN:

- **Information gain**, is the expected reduction in entropy caused by partitioning the examples according to this attribute.
- The information gain, Gain(\( S, A \)) of an attribute \( A \), relative to a collection of examples \( S \), is defined as

\[ \text{Gain}(S, A) = \text{Entropy}(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \text{Entropy}(S_v) \]

**Training Dataset:**

<table>
<thead>
<tr>
<th>Day</th>
<th>Outlook</th>
<th>Temperature</th>
<th>Humidity</th>
<th>Wind</th>
<th>PlayTennis</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Weak</td>
<td>No</td>
</tr>
<tr>
<td>D2</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>D3</td>
<td>Overcast</td>
<td>Hot</td>
<td>High</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D4</td>
<td>Rain</td>
<td>Mild</td>
<td>High</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D5</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D6</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>D7</td>
<td>Overcast</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>D8</td>
<td>Sunny</td>
<td>Mild</td>
<td>High</td>
<td>Weak</td>
<td>No</td>
</tr>
<tr>
<td>D9</td>
<td>Sunny</td>
<td>Cool</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D10</td>
<td>Rain</td>
<td>Mild</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D11</td>
<td>Sunny</td>
<td>Mild</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>D12</td>
<td>Overcast</td>
<td>Mild</td>
<td>High</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>D13</td>
<td>Overcast</td>
<td>Hot</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D14</td>
<td>Rain</td>
<td>Mild</td>
<td>High</td>
<td>Strong</td>
<td>No</td>
</tr>
</tbody>
</table>

**Test Dataset:**

<table>
<thead>
<tr>
<th>Day</th>
<th>Outlook</th>
<th>Temperature</th>
<th>Humidity</th>
<th>Wind</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
</tr>
<tr>
<td>T2</td>
<td>Sunny</td>
<td>Mild</td>
<td>Normal</td>
<td>Strong</td>
</tr>
</tbody>
</table>
Program:

```python
import math
import csv

def load_csv(filename):
    lines=csv.reader(open(filename,"r"));
dataset = list(lines)
headers = dataset.pop(0)
return dataset,headers

class Node:
    def __init__(self,attribute):
        self.attribute=attribute
        self.children=[]
        self.answer=""

def subtables(data,col,delete):
    dic={}
coldata=[row[col] for row in data]
attr=list(set(coldata))

    counts=[0]*len(attr)
r=len(data)
c=len(data[0])
for x in range(len(attr)):
    for y in range(r):
        if data[y][col]==attr[x]:
            counts[x]+=1

for x in range(len(attr)):
    dic[attr[x]]=[[0 for i in range(c)] for j in range(counts[x])]
    pos=0
    for y in range(r):
        if data[y][col]==attr[x]:
            if delete:
                del data[y][col]
            dic[attr[x]][pos]=data[y]
            pos+=1
    return attr,dic
```
def entropy(S):
    attr = list(set(S))
    if len(attr) == 1:
        return 0

    counts = [0, 0]
    for i in range(2):
        counts[i] = sum([1 for x in S if attr[i] == x]) / (len(S) * 1.0)

    sums = 0
    for cnt in counts:
        sums += -1 * cnt * math.log(cnt, 2)
    return sums

def compute_gain(data, col):
    attr, dic = subtables(data, col, delete=False)

    total_size = len(data)
    entropies = [0] * len(attr)
    ratio = [0] * len(attr)

    total_entropy = entropy([row[-1] for row in data])
    for x in range(len(attr)):
        ratio[x] = len(dic[attr[x]]) / (total_size * 1.0)
        entropies[x] = entropy([row[-1] for row in dic[attr[x]]])
        total_entropy -= ratio[x] * entropies[x]
    return total_entropy

def build_tree(data, features):
    lastcol = [row[-1] for row in data]
    if len(set(lastcol)) == 1:
        node = Node('')
        node.answer = lastcol[0]
        return node

    n = len(data[0]) - 1
    gains = [0] * n
    for col in range(n):
        gains[col] = compute_gain(data, col)
    split = gains.index(max(gains))
    node = Node(features[split])
    fea = features[:split] + features[split + 1:]

    attr, dic = subtables(data, split, delete=True)
for x in range(len(attr)):
    child = build_tree(dic[attr[x]], fea)
    node.children.append(({attr[x], child})
return node

def print_tree(node, level):
    if node.answer == "":
        print(" "*level, node.answer)
        return
    print(" "*level, node.attribute)
    for value, n in node.children:
        print(" "*(level+1), value)
        print_tree(n, level+2)

def classify(node, x_test, features):
    if node.answer == "":
        print(node.answer)
        return
    pos = features.index(node.attribute)
    for value, n in node.children:
        if x_test[pos] == value:
            classify(n, x_test, features)

'''Main program'''

dataset, features = load_csv("data3.csv")
nodel = build_tree(dataset, features)

print("The decision tree for the dataset using ID3 algorithm is")
print_tree(nodel, 0)
testdata, features = load_csv("data3_test.csv")
for xtest in testdata:
    print("The test instance:", xtest)
    print("The label for test instance:", end="   ")
    classify(nodel, xtest, features)
The decision tree for the dataset using ID3 algorithm is

```
Outlook
  rain
    Wind
      strong
        no
      weak
        yes
    overcast
      yes
  sunny
    Humidity
      normal
        yes
      high
        no
```

The test instance: ['rain', 'cool', 'normal', 'strong']
The label for test instance: no

The test instance: ['sunny', 'mild', 'normal', 'strong']
The label for test instance: yes