Name: Raghav Sharma

Roll no: 25201313

In [1]:
```
import numpy as np
from math import log2
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

In [2]:
```
# ---- Entropy ----
def entropy(y):
    values, counts = np.unique(y, return_counts=True)
    probs = counts / counts.sum()
    return -np.sum(probs * np.log2(probs))
```

In [3]:
```
# ---- Information Gain ----
def information_gain(X, y, feature_index):
    total_entropy = entropy(y)
    values, counts = np.unique(X[:, feature_index], return_counts=True)

    weighted_entropy = 0
    for v, c in zip(values, counts):
        subset_y = y[X[:, feature_index] == v]
        weighted_entropy += (c / len(y)) * entropy(subset_y)

    return total_entropy - weighted_entropy
```

In [4]:
```
# ---- ID3 Tree Builder ----
class Node:
    def __init__(self, feature=None, value=None, label=None):
        self.feature = feature  # splitting feature index
        self.value = value      # for rule display
        self.label = label      # leaf label
        self.children = {}      # feature_value -> child Node
```

In [5]:
```
def id3(X, y, features):
    # If only one class → leaf
    if len(np.unique(y)) == 1:
        return Node(label=y[0])

    # If no more features → leaf with majority class
    if len(features) == 0:
        values, counts = np.unique(y, return_counts=True)
        return Node(label=values[np.argmax(counts)])

    # Choose best feature for split
    gains = [information_gain(X, y, f) for f in features]
    best_feature = features[np.argmax(gains)]

    node = Node(feature=best_feature)

    for value in np.unique(X[:, best_feature]):
        subset_X = X[X[:, best_feature] == value]
        subset_y = y[X[:, best_feature] == value]

        if len(subset_y) == 0:
```

```
            values, counts = np.unique(y, return_counts=True)
            node.children[value] = Node(label=values[np.argmax(counts)])
        else:
            remaining_features = features.copy()
            remaining_features.remove(best_feature)
            child = id3(subset_X, subset_y, remaining_features)
            child.value = value
            node.children[value] = child

    return node
```

In [6]:
```
# ---- Prediction ----
def predict_one(node, x):
    while node.label is None:
        v = x[node.feature]
        if v in node.children:
            node = node.children[v]
        else:
            return None
    return node.label

def predict(node, X):
    return np.array([predict_one(node, x) for x in X])
```

In [7]:
```
# ---- Load Iris Dataset ----
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Discretization to make ID3 work with categorical values
X_discrete = np.array(np.round(X))  # quick discretization

X_train, X_test, y_train, y_test = train_test_split(X_discrete, y, test_size=0.3

features = list(range(X_train.shape[1]))

tree = id3(X_train, y_train, features.copy())
y_pred = predict(tree, X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9333333333333333
```

In [ ]:
```

```