In [14]:

```
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from collections import Counter
import math
```

In [15]:

```
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df["label"] = iris.target
df["label"] = df["label"].map({0: "setosa", 1: "versicolor", 2: "virginica"})
```

In [16]:

```
train_df, test_df = train_test_split(df, test_size=0.2, random_state=42)
```

In [17]:

```
def entropy(data):
    labels = data["label"]
    counts = Counter(labels)
    total = len(labels)
    return -sum((count/total) * math.log2(count/total) for count in counts.values()
```

In [18]:

```
def info_gain(data, attr, threshold):
    left_split = data[data[attr] <= threshold]
    right_split = data[data[attr] > threshold]

    if len(left_split) == 0 or len(right_split) == 0:
        return 0

    total = len(data)
    weighted_entropy = (len(left_split)/total) * entropy(left_split) + \ (len(right
    return entropy(data) - weighted_entropy
```

In [19]:

```
def best_split(data, attributes):
    best_attr, best_threshold, best_gain = None, None, -1

    for attr in attributes:
        values = sorted(data[attr].unique())
        for i in range(len(values) - 1):
            threshold = (values[i] + values[i+1]) / 2
            gain = info_gain(data, attr, threshold)

            if gain > best_gain:
                best_gain = gain
                best_attr = attr
                best_threshold = threshold

    return best_attr, best_threshold, best_gain
```

In [20]:

```
def id3(data, attributes, depth=0, max_depth=3):
    labels = data["label"]

    if len(set(labels)) == 1:
        return labels.iloc[0]
    if len(attributes) == 0 or depth == max_depth:
        return Counter(labels).most_common(1)[0][0]

    best_attr, best_threshold, best_gain = best_split(data, attributes)
    if best_attr is None:
        return Counter(labels).most_common(1)[0][0]

    left_split = data[data[best_attr] <= best_threshold]
    right_split = data[data[best_attr] > best_threshold]

    tree = {
        f"{best_attr} <= {best_threshold:.2f}": id3(left_split, attributes, depth+1
        f"{best_attr} > {best_threshold:.2f}": id3(right_split, attributes, depth+1
    }
    return tree
```

In [21]:

```
def predict(tree, sample):
    if not isinstance(tree, dict):
        return tree

    for condition, subtree in tree.items():
        attr, operator, threshold = condition.rsplit(" ", 2)
        threshold = float(threshold)

        if operator == "<=" and sample[attr] <= threshold:
            return predict(subtree, sample)
        elif operator == ">" and sample[attr] > threshold:
            return predict(subtree, sample)

    return None
```

In [22]:

```
attributes = list(df.columns[:-1])
tree = id3(train_df, attributes, max_depth=3)

print("Decision Tree (ID3):")
print(tree)
```

```
Decision Tree (ID3):
{'petal length (cm) <= 2.45': 'setosa', 'petal length (cm) > 2.45': {'petal length
(cm) <= 4.75': {'petal width (cm) <= 1.65': 'versicolor', 'petal width (cm) > 1.65':
'virginica'}, 'petal length (cm) > 4.75': {'petal width (cm) <= 1.75': 'virginica',
'petal width (cm) > 1.75': 'virginica'}}}
```

In [23]:

```
correct = 0
for _, row in test_df.iterrows():
    pred = predict(tree, row)
    if pred == row["label"]:
        correct += 1

accuracy = correct / len(test_df)
print("\nTest Accuracy:", accuracy)
```

```
Test Accuracy: 0.9666666666666667
```

In [24]:

```
sample = test_df.iloc[0]
print("\nTest Sample:", sample[:-1].to_dict())
```

```python
print("True Label:", sample['label'])
print("Predicted:", predict(tree, sample))
```

```
Test Sample: {'sepal length (cm)': 6.1, 'sepal width (cm)': 2.8, 'petal length (c
m)': 4.7, 'petal width (cm)': 1.2}
True Label: versicolor
Predicted: versicolor
```

In [ ]: