question1.py

```python
import numpy as np
import pandas as pd
import math

# Define a Pandas dataframe with the contents of the table from the problem statement.
data = pd.DataFrame([
    [1, 1, 0, 0, 1, 1],
    [1, 1, 1, 0, 1, 1],
    [0, 0, 1, 0, 0, 0],
    [0, 1, 1, 0, 1, 0],
    [0, 1, 1, 0, 0, 1],
    [0, 0, 1, 1, 1, 1],
    [1, 0, 0, 0, 1, 0],
    [0, 1, 0, 1, 1, 1],
    [0, 0, 1, 0, 1, 1],
    [1, 0, 0, 0, 0, 0],
    [1, 1, 1, 0, 0, 1],
    [0, 1, 1, 1, 1, 0],
    [0, 0, 0, 0, 1, 0],
    [1, 0, 0, 1, 0, 1],
], columns=['Early', 'Finished HMK', 'Senior', 'Likes Coffee', 'Liked The Last Jedi', 'A'])

# Define an entropy function.  The input is an attribute (one column from the table).
def entropy(attr):
    total = len(attr)
    # If there are no values, the entropy is 0
    if total == 0:
        return 0
    # Attributes are binary (0 and 1), so proportions are calculated by summing and dividing.
    # Proportion of attr = 1
    p1 = sum(attr) / total
    # Proportion of attr = 0
    p0 = 1 - p1
    # Calcuate entropy
    if p1 == 0 or p0 == 0:
        # Entropy is 0 if either proportion is 0.
         e = 0
    else:
        # Otherwise, use the formula.
        e = - (p1 * math.log2(p1) + p0 * math.log2(p0))
    return e

# Compute the entropy of the dataset.
# This value is global so it can be reference in the information_gain
# function defined below.
H_S = entropy(data['A'])

```

```
# Define a function to calculate information gain for an attribute.
def information_gain(data, attribute, target):
    # Get the unique values for the attribute (in this case, 0 and 1)
    values = data[attribute].unique()
    # Initialize the weighted entropy to zero.
    weighted_entropy = 0
    # Loop through each unique value.
    for v in values:
        # Extract the subset that matches this value.
        subset = data[data[attribute] == v][target]
        # Calculate the weighted entropy of the subset, and add it to
        # the overall weighted entropy.
        weighted_entropy += (len(subset) / len(data)) * entropy(subset)
    return H_S - weighted_entropy

# Compute the information gain for each attribute.
attributes = ['Early', 'Finished HMK', 'Senior', 'Likes Coffee', 'Liked The Last Jedi']
i_g_values = {attr: information_gain(data, attr, 'A') for attr in attributes}

print(f"Information gain for all attributes at depth 1:\n{i_g_values}")
# Select the best attribute (max information gain) to split at depth 1.
best_attr = max(i_g_values, key=i_g_values.get)

# Split data based on that best attribute
# The splits are in a hash where the key is the 0/1 value of the split attribute,
# and the value is the subset of the original table for that attribute value.
depth_1_split = {v: data[data[best_attr] == v] for v in data[best_attr].unique()}
print("----------------------------------------")
print(f"depth_1_split: {depth_1_split}")

# Select the best attributes to split at depth 2.
# These splits will also go in a hash as before.
depth_2_splits = {}
# Loop through the keys/values of the depth 1 split.
for v, subset in depth_1_split.items():
    # If there is only one target value in the split, then no further
    # splits are needed or possible.
    if len(subset['A'].unique()) == 1:
        depth_2_splits[v] = None
    else:
        # Generate a list of the remaining attributes.
        remaining_attrs = [attr for attr in attributes if attr != best_attr]
        # For each of the remaining attributes, calculate its information gain.
        # Store those in a hash where the key is the attribute.
        ig_sub = {attr: information_gain(subset, attr, 'A') for attr in remaining_attrs}
        print(f"Information gain at depth 2 for {best_attr} = {v}:\n{ig_sub}")
        # Pick the highest information gain.
        best_sub_attr = max(ig_sub, key=ig_sub.get)
        # Store that attribute.
```

```
        depth_2_splits[v] = best_sub_attr

print("----------------------------------------")
print(f"depth_2_splits: {depth_2_splits}")
print("----------------------------------------")

# Construct decision trees for depth 1 and depth 2
decision_tree_depth_1 = {best_attr: {v: {'Entropy': entropy(subset['A']), 'Positives': sum(subset['A']), 'Negatives': len(subset) - sum(subset['A'])} for v, subset in depth_1_split.items()}}
decision_tree_depth_2 = {best_attr: {v: {'Entropy': entropy(subset['A']), 'Positives': sum(subset['A']), 'Negatives': len(subset) - sum(subset['A']), 'Next Split': depth_2_splits[v]} for v, subset in depth_1_split.items()}}

# Extend to depth 3 for part C of this section.

# The depth_2_split hash that I created above does not have the table values.  It only has the attributes.
# So to compute the depth 3 split, I have to start again with the depth 1 split and work down to it.
# This is inefficient and could be improved.

depth_3_splits = {}
for v, subset in depth_1_split.items():
    if len(subset['A'].unique()) == 1:
        depth_3_splits[v] = None
    else:
        remaining_attrs = [attr for attr in attributes if attr != best_attr]
        ig_sub = {attr: information_gain(subset, attr, 'A') for attr in remaining_attrs}
        best_sub_attr = max(ig_sub, key=ig_sub.get)

        split_2 = {vv: subset[subset[best_sub_attr] == vv] for vv in subset[best_sub_attr].unique()}

        depth_3_splits[v] = {best_sub_attr: {}}

        for vv, subset_2 in split_2.items():
            if len(subset_2['A'].unique()) == 1:
                depth_3_splits[v][best_sub_attr][vv] = None
            else:
                remaining_attrs_2 = [attr for attr in remaining_attrs if attr != best_sub_attr]
                ig_sub_2 = {attr: information_gain(subset_2, attr, 'A') for attr in remaining_attrs_2}
                print(f"Information gain at depth 3 for {best_attr} = {v}, {best_sub_attr} = {vv}:\n{ig_sub_2}")
                best_sub_attr_2 = max(ig_sub_2, key=ig_sub_2.get)
                depth_3_splits[v][best_sub_attr][vv] = best_sub_attr_2

# Construct decision tree for depth 3
```

```
decision_tree_depth_3 = {best_attr: {}}
for v, subset in depth_1_split.items():
    decision_tree_depth_3[best_attr][v] = {
        'Entropy': entropy(subset['A']),
        'Positives': sum(subset['A']),
        'Negatives': len(subset) - sum(subset['A']),
        'Next Split': depth_2_splits[v],
        'Depth 3 Splits': depth_3_splits[v] if depth_3_splits[v] is not None else "Leaf Node"
    }

print("----------------------------------------")
print(f"depth_3_splits: {depth_3_splits}")
print("----------------------------------------")

# Display the decision tree results.
print("Decision tree, depth = 1:")
print(decision_tree_depth_1)
print("Decision tree, depth = 2")
print(decision_tree_depth_2)
print("Decision tree, depth = 3:")
print(decision_tree_depth_3)

```