Algorithm 4: Decision Tree

```
def h(x):
    return h_recurse(root, x)

def h_recurse(node, x):
    if node.type == "Leaf":
        return node.vote
    else:
        next = branches[x, node]
        return h_recurse(next, x)

def train(D):
    root = train_recurse(D)
    store root

def train_recurse(D):
    Let p = new Node()

    **Base Case**
    If (a) all labels in D' are the same
        (b) D' is empty
        (c) For each attribute in D', all values are identical
        p.type = "Leaf"
        p.vote = majority vote(D')
        return p

    **Recursive Case**
    Otherwise
    p.type = "Internal"
    p.m = best attribute according to splitting criterion
    = argmax_m {e_m, ..., M_m}

    store attribute on which to split
    for each value v of attribute X_m:
        D_{x_m=v} = \{(x, y) \in D' : x_m = v\}
        child_v = train_recurse(D_{x_m=v})
        p.branches[v] = child_v
    return p
```

### Class Node:
- str type // "Leaf" or "Internal"
- int vote // label for a leaf node
- int branches // map from attribute values to Node objects
- int m // attribute for internal node
Example: Decision Tree Learning w/Accuracy as Splitting Criterion

Create root: D has \([S^+, 3^-, 3^-]\)

\[p \text{branches } [v] = \text{child}_v\]
Add a branch w/label \(v\)

\[\text{return } p\]

Mutual Information

Given a random variable \(Y\) over \(K\) classes \(\Sigma_1, \ldots, \Sigma_K\)

Def: Entropy \(H(Y; D) = - \sum_{k=1}^{K} P(Y=k) \log_2 P(Y=k)\)

(informal): "how important are the labels from \(Y\)"

Def: a set of values is pure if all are the same

"how much randomness there is in \(Y\)"

(for DT): want to reduce entropy of r.v. we are trying to predict
Definition of Mutual Information (for binary attribute $X_m$)

$$I(Y, X_m; D) = H(Y; D) - \left( P(X_m = 0) H(Y; D|X_m = 0) + P(X_m = 1) H(Y; D|X_m = 1) \right)$$

For DT:

$$P(X_m = v) = \frac{\# \text{ } X_m = v}{|D|}$$

$$P(A, C) = P(A) \frac{P(C)}{P(C|A)}$$

$$P(A = a_1, C = c) = P(A = a_1) P(C = c)$$

$$P(A = a_1) = 0.1 + 0.05 + 0.15$$

$$P(A = a_2) =$$

$$P(A = a_3) =$$

$$P(C = c_1) =$$

$$P(C = c_2) =$$

$$P(C = c_3) =$$

```python
def h_recurse(node, myx):
    if ...
        if ...
            ...
        else:
            if 1. * m1 == 0:
                ...
```
else:
    if myx[node.m] == 0:
        next = node.left_child
    else:
        next = node.right_child

branches = {red: n1
            green: n2
            blue: n3}

branch[red] =

P(b1 | A=a2 , C=c4) = \frac{p(b1, a2, c4)}{\sum_{b \in \{b1, b2, b3\}} p(b, a2, c4)}