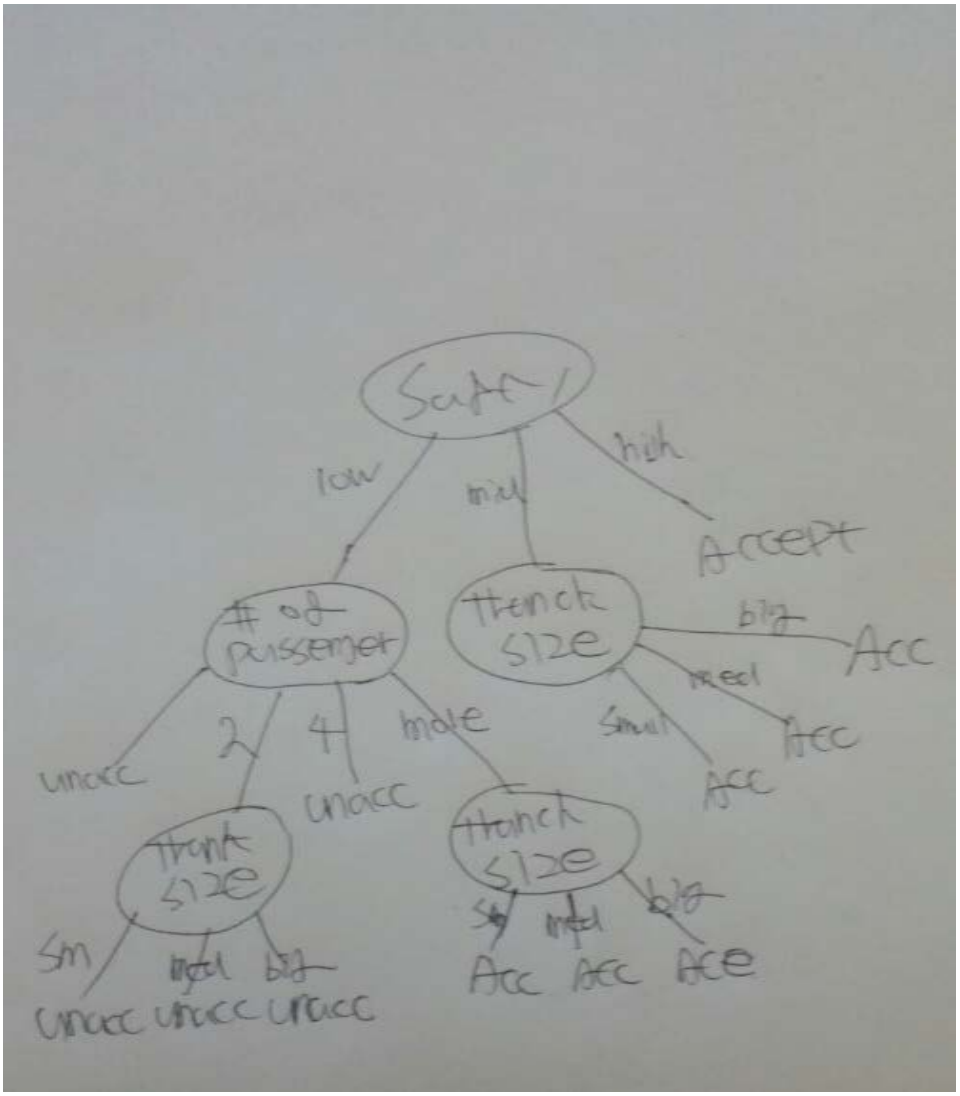
1) Output of my Decision tree algorithm.

```
-/  If Attribute 2 then
        -/  If Attribute 0 then
                -/ 1
                -/  If Attribute 1 then
                        -/  If Attribute 0 then
                        -/ 1
                        -/  If Attribute 0 then
                        -/ 1
                        -/  If Attribute 0 then
                        -/ 1
                -/  If Attribute 1 then
                -/ 1
        -/  If Attribute 0 then
        -/ 1
        -/ 1
```

This is what I got for my decision tree algorithm. Actually, I just used the print function implemented in the template. The reason why I used the template is it was enough to show the hierarchy about the result tree. However, it seems that there was some mistake on my print implementation. So that is why it does not look correct. The algorithm is pretty much in right track.

2) Tree drawn by me



Since the printing did not go well, it was hard for me to draw this tree. I will do better in next homework.

3) Code

```
import sys
import csv
import numpy as np
from Node import Node
from collections import defaultdict

global data_file, n_record, n_attr, input_data, visited


# Function used for input data configuration
def config(args="car_processed.csv"):
    print(args)
```

```
    global data_file, n_record, n_attr, input_data, visited
    data_file = args
    input_data = []
    m_data = 0
    n_data = 0
    with open(data_file, 'r') as f:
        f = csv.reader(f, delimiter=',')
        for line in f:
            m_data += 1
            vec = [int(i) for i in line]
            input_data.append(vec)
            n_data = len(vec)

    n_record = m_data
    n_attr = n_data - 1
    visited = [0] * n_attr


# Main function
def execute():
    root = Node()
    build_tree(input_data, root)
    traverse_tree(root, 0)


# Function to build Tree
# @param data   Data of passed in Node including Attribute data and label data
# @param parent passed in Node

def split(arr, con):
    return [arr[con], arr[~con]]

def output_tree(index_attr):
    golf="golf.csv"
    golf_data=[]
    with open(golf,'r') as f:
        f=csv.reader(f,delimiter=',')
        for line in f:
            vec=[i for i in line]
            golf_data.append(vec)
    if index_attr==0:
        return 'outlook'
    if index_attr==1:
        return 'temperture'
    if index_attr==2:
        return 'humidity'
    if index_attr==3:
        return 'windy'
def output_label(num):
    if num==1:
        return 'Yes'
    if num==2:
        return 'No'



def split_by_label(array, num):
```

```
    e_list = []
    temp_list = []
    for i in num:
        for index, k in enumerate(array[:, -1]):
            if i == k:
                temp_list.append(array[index, :].tolist())

        e_list.append(np.array(temp_list))
        temp_list = []
    return e_list


def allvisited(visit):
    for i in visit:
        if i == 0:
            return False
    return True


def build_tree(data, parent):

    print("Builf Tree begins")

    a = np.array(data)
    label = a[:, -1]
    print('Label(in BT)=Wn', label)
    attr = a[:, :-1]
    print('Attribut(in BT)=Wn', attr)
    gini_list = []
    row, col = np.shape(a)
    print('ROW,COL=', row, col)

    if len(np.unique(a[:, -1])) == 1:
        node = Node()
        node.set_data(parent.get_data()+'Label:'+str(a[0, -1]))
        node.set_parent(parent)
        node.n_count=parent.n_count+1
        parent.add_child(node)
        print("return NODE")
        return node

    else:
        for x in range(col - 1):
            gini_list.append(find_gini(attr[:, x], label))

        next_att = gini_list.index(min(gini_list))

        print('GINI_LIST=', gini_list)
        print('NEXT_attr(min of ginilist)=', next_att)
        node = Node()
        node.set_data(parent.get_data()+' If Attribute ' + str(next_att)+' then ')
        node.set_parent(parent)
        parent.add_child(node)
        uniq_set = np.unique(attr[:, next_att])

        if allvisited(visited):
            node = Node()
            node.set_data(str(a[0, -1]))
```

```
            node.set_parent(parent)
            parent.add_child(node)
            return node
        else:
            visited[next_att] = 1
            # this next_att is chosen by the smallest gini_index
            print(uniq_set)

            temp = []

            for j in uniq_set:
                temp2 = []
                # next_att is the gini_index one

                for y, i in enumerate(a[:, next_att]):
                    if j == i:
                        temp2.append(a[y, :])
                temp.append(temp2)

            print('Temp=₩n', temp)

            for x in temp:
                # x=np.delete(x,next_att,1)
                print('seperaed=₩n', x, '₩n')
                build_tree(x, node)

            return node

        # **-------------------Fill in here-----------------------**/
        # Note: You should break down data to matrix of attribute and array of records
corresponding label

        # Check if all records belong to one label -> add leaf node
        # To add a node by:
        # - Update parent's children list (parent.addChild())
        # - Update children node's parent (child.setParent(parent))

        # Find attribute not visited with minimum gini index using findGini

        # If not found any attribute -> all attribute are visited -> find majority label
and add leaf node
        # Else
        # - Creat new Node with found attribute
        # - Add new Node to parent
        # - Mark visited[min Gini attribute] = 1

        # Continue to build tree by building node for each value of new created node

        # Find gini index of given attribute data and corresponding Labels
        # @param  att Attribute data
        # @param  lab Label data
        # @return gini index of an attribute


def find_gini(att, lab):
    d_attr = 0
    d_label = 0
    total = len(att)
```

```
list_by_label = []
total_len = len(att)
num_of_label = np.unique(lab)
diff_label = len(num_of_label)
# this is number of different label.
con = np.column_stack((att, lab))

aa1 = split_by_label(con, num_of_label)

val_dict = defaultdict(dict)
occur_dict = {}
v_set = set()

for i in range(diff_label):
    print('val i=', i)
    temp_dict = {}
    uniq_v, occur_v = np.unique(aa1[i][:, 0], return_counts=True)
    print('unique values: ', uniq_v, "Number of occurence(attri): ", occur_v)

    for index, x in enumerate(uniq_v):
        temp_dict[x] = occur_v[index]
    val_dict[i] = temp_dict

print("valdict!!= ", val_dict)

array_o = []

for r in val_dict.keys():

    for l, h in val_dict[r].items():
        array_o.append([r, l, h])

numpy1 = np.array(array_o)
print(numpy1)
total_list = []

for value in np.unique(numpy1[:, 1]):
    one = 1
    value_list = []
    for index, attr in enumerate(numpy1[:, 1]):
        if value == attr:
            value_list.append(numpy1[index, -1])
    for v1 in value_list:
        one = one - pow(v1 / sum(value_list), 2)
    one = one * sum(value_list) / total_len
    total_list.append(one)
print('SUM=', sum(total_list))
return sum(total_list)




# number of label
#     **------------------Fill in here-----------------------**/
#     These steps might help you:
#      - Find number of different values in attribute, number of different labels
#      - For each value i, find number of occurrences and number of corresponding
```

```
    labels to calculate ginisplit
    #       - gini = sum of all ginisplit


# Use DFS to traverse tree and print nicely with appropriate indent
# @param node traversing Node
# @param indent appropriate indent for each level
def traverse_tree(node, indent):
    # Print out current node with appropriate indent
    for i in range(indent):
        print('Wt', end="")
    if (node.get_parent() is None):
        print("root")
    else:
        print("-/", node.get_data())

    # Recursive call all the children nodes
    children = []
    children = node.get_children()
    for i in range(node.get_n_child()):
        traverse_tree(children[i], indent + 2)


def main():
    arg = sys.argv[1:]
    if len(arg) > 0:
        config(arg)
    else:
        config()
    execute()


main()
```