**MSCF Programming Prep, 2022**

**Homework 2**

***Due At 11:59 pm U.S. Eastern time, Thursday, Aug. 18***

***You will lose 10 points per hour after that time***

1. **(70 points) Dynamic Programming**

Create a file **hw2\_1.py** to contain the code for parts 1.a, 1.b, and 1.c, below. You can copy-and-paste useful code from the **hw2\_code\_chunks.py** file.

1. Given a **list** of **int** values, such as: **[2, -4, 7, 2, 0, 5, -3, 4, -2]** find the *maximum sum of all possible sequential sublists*. In this case, the solution is **15** (and the sublist would be **[7, 2, 0, 5, -3, 4]**). (You can just look this up on the web, of course, but at least wrestle with it if you don’t know the answer. Approach it along the lines of the rod cutting question: What if there is only one value in the list? What if there is one more value in the list? And so forth. Amusingly and dauntingly, the O(N) algorithm for this was discovered by a CMU professor.)

Define your algorithm as a function **max\_sublist\_sum(***list***)**, and test your function with this code:

**import numpy as np**

**np.random.seed(1) # so results match**

**list1 = [2, -4, 7, 2, 0, 5, -3, 4, -2]**

**print("max\_sublist\_sum(", list1,**

**"): ", max\_sublist\_sum(list1))**

**for n in range(10, 20):**

**listn = list(np.random.randint(-5, 9, size=n))**

**print("max\_sublist\_sum(", listn,**

**"): ", max\_sublist\_sum(listn))**

1. Define a function **knapsack01()** that takes three arguments:

**item\_wgts** a list of integer item weights

**item\_vals** a list of integer item values

**tot\_wgt** the integer total weight that will fit into your knapsack

This function should implement the 0/1 Knapsack algorithm to return the *maximum value* of items that will fit into the knapsack.

Test your **knapsack01()** function with this code:

**item\_weights = [3, 1, 12, 5, 2]**

**item\_values = [9, 7, 18, 3, 11]**

**total\_weight = 10**

**print("max value:",**

**knapsack01(item\_weights, item\_values,**

**total\_weight))**

Confirm that the max value returned is 27 in this case.

Test your **knapsack01()** function again with this code:

**item\_weights = [2, 3, 5, 1, 12]**

**item\_values = [11, 9, 3, 7, 18]**

**total\_weight = 10**

**print("max value:",**

**knapsack01(item\_weights, item\_values,**

**total\_weight))**

Confirm that the max value returned is also 27 in this case (the order of the items does not matter).

Test your **knapsack01()** function a third time with this code:

**item\_weights = [2, 4, 9, 3, 12, 4, 1, 3]**

**item\_values = [8, 4, 2, 11, 9, 5, 2, 8]**

**total\_weight = 20**

**print("max value:",**

**knapsack01(item\_weights, item\_values,**

**total\_weight))**

What max value do you get?

1. Define a function **min\_coins()** that takes two arguments:

**coins** a list of integer denominations of coins (all >0, no duplicates)

**value** an integer total value of change to be made from the coins

This function should return the *minimum number of coins* (from among the denominations in **coins**) needed to make value worth of change.

Test your **min\_coins()** function with this code:

**coins = [2, 3, 6]**

**value = 10**

**print("value", value, "requires",**

**min\_coins(coins, value), "coins minimum")**

Confirm that 3 coins minimum are required.

Test your **min\_coins()** function again with this code:

**coins = [1, 5, 10, 25]**

**value = 117**

**print("value", value, "requires",**

**min\_coins(coins, value), "coins minimum")**

Confirm that 8 coins minimum are required.

Test your **min\_coins()** function a third time with this code:

**coins = [1, 5, 10, 18, 25, 37]**

**value = 273**

**print("value", value, "requires",**

**min\_coins(coins, value), "coins minimum")**

What is the minimum number of coins required?

1. **(30 points) More Binary Search Tree**

Make a copy of your **BinaryTree.py** code file from Homework 1 named **BinaryTree\_hw2.py**. Make a copy of your **BT\_app2.py** code file from Homework 1 named **BT\_app\_hw2.py**. Modify **BT\_app\_hw2.py** to import **BinaryTree\_hw2**, then run **BT\_app\_hw2.py** to confirm that your code works correctly.

1. In your **BinaryTree** class, define a method **\_\_contains\_\_()** that takes a value as an argument, and returns **True** if the value is contained in the current **BinaryTree** object, and **False** otherwise.

At the bottom of **BT\_app\_hw2.py**, add these test statements:

**print('18 in bt3:', 18 in bt3)**

**print('21 in bt3:', 21 in bt3)**

**print('13 in bt1:', 13 in bt1)**

**print('13 in bt3:', 13 in bt2)**

Save and run, and confirm that the output makes sense. (For a reasonably balanced Binary Search Tree, this is an **O(log N)** operation.)

1. In your **BinaryTree** class, define a method **copy()** that returns a copy of the current **BinaryTree** object.

At the bottom of **BT\_app\_hw2.py**, add these test statements:

**bt3r = bt3 # copy or reference?**

**print('\nbt3r.print\_pretty():')**

**bt3r.print\_pretty()**

**print('bt3r == bt3:', bt3r == bt3)**

**print('bt3r is bt3:', bt3r is bt3)**

**bt3c = bt3.copy() # copy or reference?**

**print('\nbt3c.print\_pretty():')**

**bt3c.print\_pretty()**

**print('bt3c == bt3:', bt3c == bt3)**

**print('bt3c is bt3:', bt3c is bt3)**

Save and run, and confirm that the output makes sense.

1. In your **BinaryTree** class, define a method **negate()** that modifies all of the values in the current **BinaryTree** object to be the negative of their original values. That is, if 7 is a value in the current **BinaryTree**, its value should be changed to -7. ***Notice***that this requires restructuring the tree to maintain the rule that values lower than the current node’s value are in the left child, and values greater than the current node’s value are in the right child.

At the bottom of **BT\_app\_hw2.py**, add these test statements:

**bt3c.negate()**

**print('\nbt3c.print\_pretty():')**

**bt3c.print\_pretty()**

Save and run, and confirm that the output makes sense.

***REMEMBER*** to put all team members’ names (Andrew IDs) into your source code files.Put your **breadth\_first\_search.py, hw2\_1.py, BinaryTree\_hw2.py** and **BT\_app\_hw2.py** files into a **Team***N***\_HW2.zip** archive, where *N* is your team number, and upload to Canvas.