COMP 285 (NC A&T, Fall ‘22) Homework 7 (100 pt)

Fun with Greediness and Dynamic Programming

###### Due 11/15/22 @ 11:59PM ET

Submitting

In order to submit this assignment, you must download your assignment as a ZIP file and upload it to Gradescope. There will be only a coding component.

You will write your solution in `answers.cpp`. We will use an autograder to test your solutions - the results of which can be seen after submitting to Gradescope. Note that you have unlimited tries to submit. In order to receive full credit, you must also provide documentation on your approach, see the commented sections in the `answers.cpp` file.

Question 1: Sharing Passwords (25 pts)

As part of your work for an elite cybersecurity firm, you're currently consulting with two co-CEOs of a major Fortune 500 corporation.

Each CEO has a hard-to-crack, extremely long password. The two CEOs are in the process of setting up a shared network, and would like the password for this new network to be easy to remember for both of them.

As such, they ask you to figure out the following:

Let the string S (of lenth n) and T (of length m) be the two passwords that each CEO has memorized. They want to know if their shared password, which will be a common subsequence of S and T, will be of sufficient length.

Implement an algorithm `lenSharedPassword` in `answers.cpp` that returns the length of the longest possible shared password. If there is no possible password, return 0.

Note that a a subsequence of a string is a new string generated from the original string with some characters (can be none) deleted without changing the relative order of the remaining characters.

For example, "ace" is a subsequence of "abcde". A common subsequence of two strings is a subsequence that is common to both strings.

Here are a few example inputs:

```
 // 3 because the longest shared password will be "ace".
std::cout << "pass1: abcde, pass2: ace, lenShared: " << lenSharedPassword("abcde", "ace") << std::endl

// 3 because the longest shared password will be "abc".
std::cout << "pass1: abc, pass2: abc, lenShared: " << lenSharedPassword("abc", "abc") << std::endl;

// 0 because there is possible shared password.
std::cout << "pass1: abc, pass2: def, lenShared: " << lenSharedPassword("abc", "def") << std::endl;

// 3 because the longest shared password will be "ace".
std::cout << "pass1: abcef, pass2: gahtce, lenShared: " << lenSharedPassword("abcef", "gahtce") << std
```

Write your solution in `answers.cpp`.

Question 2: Transporting Valuable Metals (25 pts)

You have `n` types of metals. For each metal bar `i`, you have an associated weight given by `pounds[i]` (in pounds) and a dollar value, given by `dollars[i]`. You have a secure transport vehicle which can only hold `vehicleCapacity` pounds.

You have an infinite supply of each type of metal.

What is the maximum dollar value you can fit into your secure transport vehicle in a single convoy?

Implement your solution in `transportValue` in `answers.cpp`. For full credit, your algorithm must have time complexity of `O(nW)`.

Here are a few examples:

```
 std::vector<int> pounds = {1, 2, 3};
std::cout << "pounds := {1 , 2, 3}" << std::endl;
std::vector<int> dollars = {1, 4, 6};
std::cout << "dollars := {1, 4, 5}" << std::endl;
// We can take 2 of the second metal, which weighs 2 pounds each. This gives use a total value of 8.
std::cout << "We can fit $" << transportValue(pounds, dollars, 4) << " in a transport that fits 4 poun
```

Write your solution in `answers.cpp`.

Question 3: Moving Back Home (25 pts)

After several years in college, you've collected `n` items that you find very valuable. For each item, `i`, you have an associated weight given by `weight[i]` (in ounzes) and a sentimental value given by `importance[i]`. You only have one shipping box that can hold `W` ounces.

Given these are real-world items, there exists only one copy of each of the `n` items.

What is the maximum sentimental value you can derive by strategically packing your items into the box?

Implement your solution in `mostPricedPossesions` in `answers.cpp`. For full credit, your algorithm must have time complexity of `O(nW)`.

Here are a few examples:

```
 std::vector<int> weight = {1, 2, 3};
std::cout << "weight := {1 , 2, 3}" << std::endl;
std::vector<int> importance = {1, 4, 6};
std::cout << "importance := {1, 4, 5}" << std::endl;
// We can take 1 of the first and last items which will give us the most importance of 6.
std::cout << "We can fit a value of " << mostPricedPossesions(weight, importance, 3) << " in a box tha
```

Write your solution in `answers.cpp`.

Question 4: Minimum Number of Coins (25 pts)

A vending machine stocks pennies (1c), nickels (5c), and quarters (25c). Write a program that determines the fewest number of coins that must be dispensed to return exactly N cents to the customer.

For example:

```
 // N = 30, should return 2 because 25c + 5c = 30c
std::cout << minimumNumberOfCoins(30) << " equals? 2" << std::endl;

// N = 90, should return 6 because 3*25c + 3*5c = 90c
std::cout << minimumNumberOfCoins(90) << " equals? 6" << std::endl;
```

Write your solution in `answers.cpp`.