**MCS 253**

**HW 8**

[Read this first on what to submit for homework.](https://drive.google.com/open?id=1vvTZzNrXNrxFCFVuzryEuAX6c8rfGERsEc7t4TS1vAA)

**Special Notes for HW 8:**

* **FOR ALL PROBLEMS, YOU MUST USE A DYNAMIC PROGRAMMING APPROACH!!!**
* **No edge case testing needed for report, HOWEVER test cases will test certain edge cases!!!**
* **You may store paths as you perform your DP (ie. no need to save space by waiting till dp is finished to reconstruct the solution path via backtracking)**
  + **HOWEVER, you should know how to construct a solution path via backtracking from your dp solution. (If that is not clear to you, you may want to implement one of the problems to extract the solution in this way to practice).**

**HW 8.1 - Paying in Coins (25 pts)**

You have a pocket full of coins (pennies, nickels, dimes, and quarters with values of 1, 5, 10, and 25 cents respectively) that you wish to get rid of. Each time you go to buy an item, you try to ***pay with the most number of coins possible***.

Construct an algorithm that, given a cost and the number of coins you have available, pays the cost with as many coins as possible.

Your main will start with a pocket full of coins (loaded from stdin). Everytime an item is bought, the value of the item will be entered via stdin and your algorithm should be used to determine how you will pay for the item, and determine your remaining coins.

Input:

* starting number of each coin
* value of items to be bought

Output:

* for each item bought, the coins that were used, and the number of coins remaining in your pocket

Example:

input:

7 10 1 3 *⇐ pennies nickels dimes quarters*

36

15

40

15

output:

7 10 1 3

- 6 6 0 0

------------

1 4 1 3

- 0 3 0 0

------------

1 1 1 3

- 0 1 1 1

------------

1 0 0 2

cannot pay 15 cents!

------------

1 0 0 2

**HW 8.2 - Paying in Coins Part II (25 pts)**

You have a pocket full of coins again, but this time you know ahead of time what you will purchase as well as how much each item will cost. Your goal is again to use as many coins as possible, but this time while trying to buy as many items as you can.

Construct an algorithm that, given the number of coins you have available and a list of item costs, uses as many coins as possible while trying to maximize the number of items that can be bought.

Input:

* starting number of each coin
* value of items to be bought

Output:

* for each item bought, the coins that were used, and the number of coins remaining in your pocket

Example:

input:

7 10 1 3 *⇐ pennies nickels dimes quarters*

36

15

40

15

output:

7 10 1 3

- 6 1 0 1

------------

1 9 1 2

- 0 3 0 0

------------

1 6 1 2

- 0 3 0 1

------------

1 3 1 1

- 0 3 0 0

------------

1 0 1 1

*(or solution that uses equivalent number of coins)*

**HW 8.3 - DNA Sequence Alignment Part I (25 pts)**

In DNA sequences, there are four types of repeating units (called bases) denoted as A, T, C, and G. Two opposite strands are aligned if All A’s in one string are matched with T’s in the opposing string, and all C’s in one string are matched with G’s in the opposing string.

An example of a valid alignment is:

A C C T C G

T G G A G C

...and an example of an invalid alignment is:

A **G** **C** **T** T **G**

T **G** **C** **G** A **A**

Assume that given opposing strands that do not align, and that you are allowed to put spacers to align them. For example:

We could transform this:

A **G** **C** **T** T **G**

T **G** **C** **G** A **A**

...into this:

A \_ **G** **C** **T** T **G**

T **G** **C** **G** A **A** \_

Write a program that takes two opposing strands of DNA, and determines the best way to align the sequences by inserting the minimum number of spacers.

Feel free to watch an explanation for a variant of the problem where the goal is to have two sequences perfectly match after addition of spacers.

* [Part 1](https://drive.google.com/open?id=1oXPl2bXEAMjjqjVFxMolfaPKpsgVy-Rx)
* [Part 2](https://drive.google.com/open?id=1HcqjKDHXZ7YqVdA3jqdxlY5KLXyVERPc)

*(Ignore references to Piazza and lecture as the videos were in reference to a different quarter).*

Input:

* two DNA sequences as strings

Output:

* a final sequence alignment with minimal number of spacers inserted

Example:

input:

A G C T T G

T G C G A A

output:

A \_ G C T T G

T G C G A A \_

**HW 8.4 - DNA Sequence Alignment Part II (25 pts)**

Repeat the DNA alignment problem from 8.3, however, this time allowing invalid base-pairings with penalties such that:

* T - T, C - C, and T - C mis-pairings are penalized 3/2 times as much as a spacer
* T - G and A - C mis-pairings are penalized 7/4 times as much as a spacer
* A - A, G - G, and A - G mis-pairings are penalized 5/2 times as much as a spacer

Input:

* two DNA sequences as strings

Output:

* a final sequence alignment to maximize alignment score based on the above criteria

Example:

input:

T T C G C T

A A G G C T

output:

T T C G C \_ T

A A G \_ G C T *(or equivalent solution)*

**LeetCode Substitution Problems (25 points each)**

Up to *50pts* may be substituted.

Possible LeetCode Problems:

* 322. Coin Change
* 518. Coin Change 2
* 879. Profitable Schemes
* 741. Cherry Pickup
* 44. Wildcard Matching
* 1301. Number of Paths with Max Score
* 174. Dungeon Game
* 312. Burst Balloons
* 354. Russian Doll Envelopes
* 403. Frog Jump