CS 3510: Homework 2A

Due on **Friday** Feb 18

Professor Faulkner

CS 3510 Staff

Problem 1. *(33 points)* You need to make change for n cents, but you only have coins of value 4 and 7 (assume you have an infinite number of both of these coins). Design a dynamic programming algorithm which returns whether it is possible to make n cents with the coins you have.
Example:
Input: Target amount = 18
Output: True because 18 = 4 + 7 + 7

a.) (6 points) Define the entries of your table in words. E.g. $T(i)$ or $T(i, j)$ is ...
We are going to create an output array $T[i]$ which will store Boolean values and the Boolean value will represent if it is or isn't possible to make the target amount with the coins.

b.) (12 Points) State the recurrence for entries of your table in terms of smaller subproblems. Briefly explain in words why it is correct. Don't forget the base cases here.
We can first deal with the base cases of target 0 which we can preset to True and for other values less than 4 we can set to False. For further values, we can say $T[i] = i(\text{mod}4) \equiv 0$ or $T(i-7)$. This works because if it is divisible by 4 then we automatically know that we can make it with only coins of values 4, otherwise we can subtract 7 and check the smaller amount for if it was possible. Then with Boolean logic if either of those are True then we know we can create it using coins of value 4 or 7.

c.) (6 points) Use your recurrence relationship to write pseudocode for your algorithm.

d.) (9 Points) Determine the running time of your algorithm. Briefly provide justification.
In a worst case scenario we are going to be performing mod, $n-4$ times which we can say we are doing $\mathcal{O}(1)$ work roughly $\mathcal{O}(n-4) = \mathcal{O}(n)$ times, which results in the runtime of $\mathcal{O}(n)$.

Problem 2. *(34 points)* You are given a set of k coins each with a different value and a target amount n. Return the fewest number of coins that you need to make up the target amount. If that amount of money cannot be made up by any combination of the coins, return -1. You may assume that you have an infinite number of each kind of coin.

Example:
Input: coins = 1, 2, 5, target amount = 11
Output: 3 because 11 = 5 + 5 + 1

Example 2:
Input: coins = 2, 4, 6, target amount = 7
Output: -1 because you can't make up the target amount with the coins.

a.) (6 points) Define the entries of your table in words. E.g. $T(i)$ or $T(i, j)$ is ...
The entries for the table $T(i)$ will be the number of coins needed to create the value of the index.

b.) (12 Points) State the recurrence for entries of your table in terms of smaller subproblems. Briefly explain in words why it is correct. Don't forget the base cases here.
The base case for this problem is that when the target amount is 0, then the amount required is 0 coins. However, when the target value is greater than 0, then we are going to check the value in the index of the current spot minus the value of the coin we are checking with as $T[i - coins[j]]$ where $coins[]$ is the array of coins that we can use, index i is the current one we are checking, and index j is for which coin in $coins[]$ we are checking with. This works because if the index-coin location was made possible, then by adding the coin value we are able to add the current coin value to it and make the target amount. Then we populate $T[i]$ with $T[i - coins[j]] + 1$ to account for the addition of one coin, being the one that we are currently checking. In this we would iterate over the values of $coins[]$ using the same process and check to see if they are less than the value already in use.

c.) (6 points) Use your recurrence relationship to write pseudocode for your algorithm.

```
Function COIN2(int coins[], int n):
    /* let $T$ be a $n+1$ long list that is filled with $-1$s */
    $T \leftarrow [-1]$
    /* base case */
    $T[0] \leftarrow 0$
    for $i \in 1 \rightarrow n$ do
        for $j \in 0 \rightarrow k-1$ do
            if $coins[j] \leq i$ then
                int $amount = T[i - coins[j]]$
                if $amount > -1 \&\& amount + 1 < T[i]$ then
                    $T[i] \leftarrow amount + 1$
    return $T[n+1]$
```

d.) (10 Points) Determine the running time of your algorithm. Briefly provide justification.
In this algorithm we are going to iterate over each value up to the target for each coin that is input. This implies that we would be doing $\mathcal{O}(k)$ iterations $\mathcal{O}(n)$ times. The comparisons that we make in the if statements all have a running time of $\mathcal{O}(1)$. Therefore, the running time of this algorithm is around $\mathcal{O}(kn)$.

Problem 3. *(33 points)* Christmas will be ruined if we can't unload all of our ships in time. To unload a ship, we have to unload its cargo in a specified order, and we want to minimize the amount of time this will take. Each ship is carrying 3 different shapes of boxes, labelled as $\{A, B, C\}$, and the order of the box shapes that a ship must be unloaded is given by a string U (for example, $U =$ '$ABCCBCCBBBABAB$'). For each unit of time, we can either unload a single box, or we can unload a chunk of boxes if this chunk happens to be "convenient" – the convenient chunks are specified by a list L, which contains acceptable box shape sequences (for example, $L = [AB, AC, BBCC]$). If a chunk of boxes matches a shape sequence on the list L, we can unload this group in one unit of time.

Design a dynamic programming algorithm to output the minimum amount of time required to unload a sequence of box shapes U. Describe the running time of this algorithm. Assume that you can check if a given string is in list L in $O(1)$ time.

Example Input: $U = CAAACBB$, $L = [AAA, ACBB, CBB]$
Example Output: The ship requires 3 units of time to unload, by first unloading the single box C then chunk AAA and finally chunk CBB.

a.) (6 points) Define the entries of your table in words. E.g. $T(i)$ or $T(i, j)$ is ...
The entries of the table $T[i]$ are going to be the number of units of time to unload the subsets of crates from the first create to the nth crate.

b.) (12 Points) State the recurrence for entries of your table in terms of smaller subproblems. Briefly explain in words why it is correct. Don't forget the base cases here.
The base case will be if there are no crates to unload then it will take 0 time to unload all the crates from the ship. For crate counts greater than 0 we are going to first take $T[i] = T[i-1]$, then we are going to compare each convenient chunk from L with the appropriate length substring at the end of the current total substring we are looking at for the $T[i]$ substring. If one of these substrings is works then we are going to look at the value $T[i - chunklength] + 1$ and if it less then the current value in $T[i]$ we are going to $T[i] = T[i - chunklength] + 1$ which will change it to the newest smallest amount of time.

c.) (6 points) Use your recurrence relationship to write pseudocode for your algorithm.

```
Function CRATE(String U, String L[]):
    /* let $T$ be a $n+1$ long list that is filled with $-1$s */
    $T \leftarrow [-1]$
    /* base case */
    $T[0] \leftarrow 0$
    for $i \in 1 \rightarrow U.length$ do
        $T[i] \leftarrow T[i-1] + 1$ for $j \in 0 \rightarrow L[].size$ do
            if $U.substring(L[j].length, i).equals(L[j])$ then
                int $amount = T[i - L[j].length]$
                if $amount > -1 \&\& amount + 1 < T[i]$ then
                    $T[i] \leftarrow amount + 1$
    return $T[n+1]$
```

d.) (9 Points) Determine the running time of your algorithm. Briefly provide justification.
We are performing $\mathcal{O}(n)$ where n is the length of the string of crates on the ship but for each iteration we are going to heck the substrings with the convenient crate options from the list $L[]$, this will then cause the runtime to be dependent on the size of the list L which we will denote as k. Additionally we are going to perform $\mathcal{O}(1)$ work for each of these iteration. Finally that means that we are going to perform $\mathcal{O}(1) \times \mathcal{O}(k) \times \mathcal{O}(n) = \mathcal{O}(kn)$ work. Therefore, the runtime of this algorithm is roughly $\mathcal{O}(kn)$