Lab04-Dynamic Programming

Exercises for Algorithms by Xiaofeng Gao, 2018 Spring Semester.

∗ If there is any problem, please contact TA Jiaxi Liu (jsxzljx@163.com).

∗

1. Given an integer, we now translate it to a string by following these rules: 0 is translated to ‘a’, 1 is translated to ‘b’, ..., 11 is translated to ‘l’, ..., 25 is translated to ‘z’. Note that an given integer can be translated to different kinds of strings, for example, there are 5 different translations for 12248 and they are “bccfi”, “bwfi”, “bczi”, “mcfi”, and “mzi” respectively. Please design an algorithm to calculate how many kinds of translations there are for an arbitrarily given integer. Specifically, you should

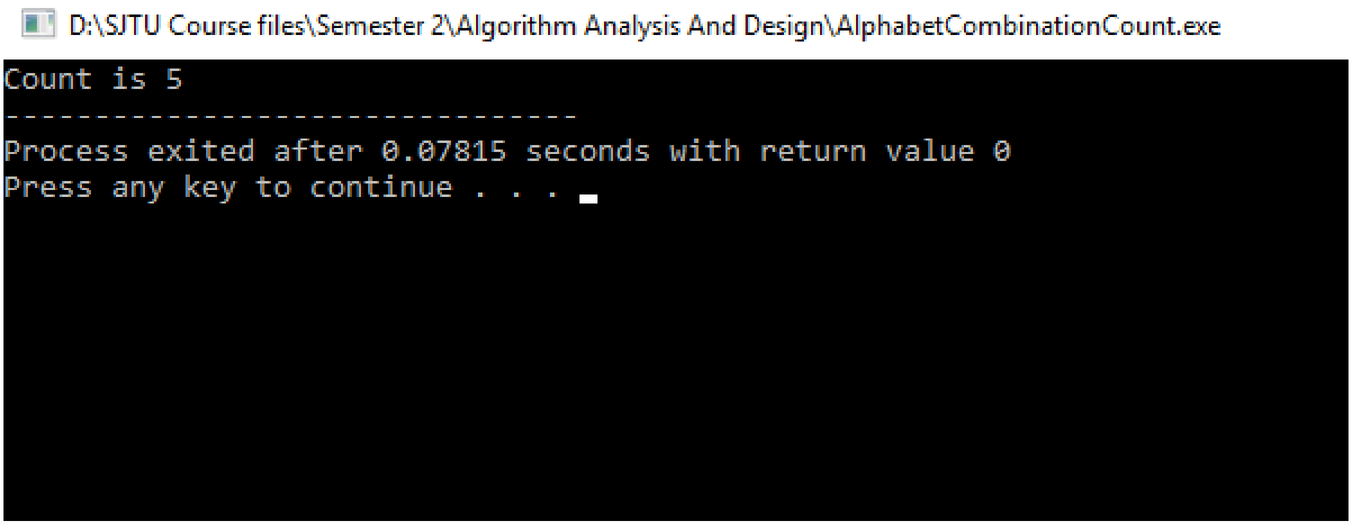
   (a) Give the recursive formula and the explanation of your algorithm.

   (b) Analyze both the time and the space complexity.

   Solution: (a)

```
#include <iostream>
#include <cstring>
using namespace std;

// Given a digit sequence of length n, returns count of possible
// decodings by replacing 1 with A, 2 woth B, ... 26 with Z
int countDecoding(char *digits, int n)
{
    // base cases
    if (n == 0 || n == 1)
        return 1;

    int count = 0;  // Initialize count

    // If the last digit is not 0, then last digit must add to
    // the number of words
    if (digits[n-1] > '0')
        count =  countDecoding(digits, n-1);

    // If the last two digits form a number smaller than or equal to 26,
    // then consider last two digits and recur
    if (digits[n-2] == '1' || (digits[n-2] == '2' && digits[n-1] < '7') )
        count +=  countDecoding(digits, n-2);

    return count;
}

// Driver program to test above function
int main()
{
    char digits[] = "12248";
    int n = strlen(digits);
    cout << "Count is " << countDecoding(digits, n);
    return 0;
}
```

Listing 1: Recursive Algorithm Code for finding combinations of alphabets



D:\SJTU Course files\Semester 2\Algorithm Analysis And Design\AlphabetCombinationCount.exe

```
Count is 5
--------------------------------
Process exited after 0.07815 seconds with return value 0
Press any key to continue . . .
```

Figure 1: shows Output of the Recursive Algo

Explanation: In this recursive problem for sequences of length 0 or 1, there is only one translation (Base case). For larger sequences of digits n, possible number of translations could be either equal to those for sequence of digits n-1, or we might need to add number of possible translation for sequence of digits length n-2 if digits at position n-2 and n-1 can together translate to a valid character.

(b): Time and Space Complexity:

The time Complexity for the algorithm is $O(n^2)$ as at each recurrence, size of problem reduces by 1, so our recurrence tree will have a depth of n. Now at each step we either have 1 or 2 branches, so our time complexity is exponential in nature. and space complexity is $O(n)$ as well.

2. You are working at the cash counter at a fun-fair. Suppose you have different denominations of coins (i.e., different types of coins) and assume the quantity for each denomination is infinite. One day a customer comes to you and asks for coins change for a specific amount. Please design a dynamic programming algorithm to compute *the fewest number* of coins that you need to make up that amount. If that amount of money cannot be made up by any combination of the coins, your algorithm should return -1. Here are 2 examples:

   - Example 1: suppose the coins set is $\{1, 2, 5\}$, the amount is 11, then your answer is 3 since $11 = 5 + 5 + 1$
   - Example 2: the coins set is $\{2\}$ and the amount 3, then your answer is -1 because you can not find a right change.

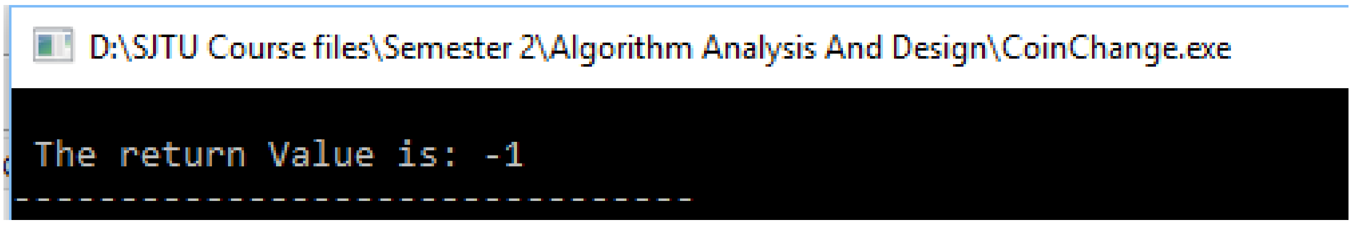
   Describe your algorithm and give the time complexity analysis.

   Solution:

```
// A Dynamic Programming based C++ program to find minimum of coins
// to make a given change V
#include<bits/stdc++.h>
using namespace std;

// m is size of coins array (number of different coins)
int minCoins(int coins[], int m, int V)
{
    // table[i] will be storing the minimum number of coins
    // required for i value.  So table[V] will have result
    int table[V+1];

    // Base case (If given value V is 0)
    table[0] = 0;

    // Initialize all table values as Infinite
    for (int i=1; i<=V; i++)
        table[i] = INT_MAX;

    // Compute minimum coins required for all
    // values from 1 to V
    for (int i=1; i<=V; i++)
    {
        // Go through all coins smaller than i
        for (int j=0; j<m; j++)
          if (coins[j] <= i)
          {
              int sub_res = table[i-coins[j]];
              if (sub_res != INT_MAX && sub_res + 1 < table[i])
                  table[i] = sub_res + 1;
          }
    }
    return table[V];
}

// Driver program to test above function
int main()
{  int rtn=0;
    int coins[] =  {1,2,5};
    int m = sizeof(coins)/sizeof(coins[0]);
    int V = 11;
    cout << "Minimum coins required is " << minCoins(coins, m, V);
         cout<<"\n";
   if(m<2){
    rtn = -1;
    }
    else {
      rtn = m;
    }
    cout<<" The return Value is: "<<rtn;
    return rtn;
}
```

Listing 2: Coin Change Problem Via Dynammic Programming

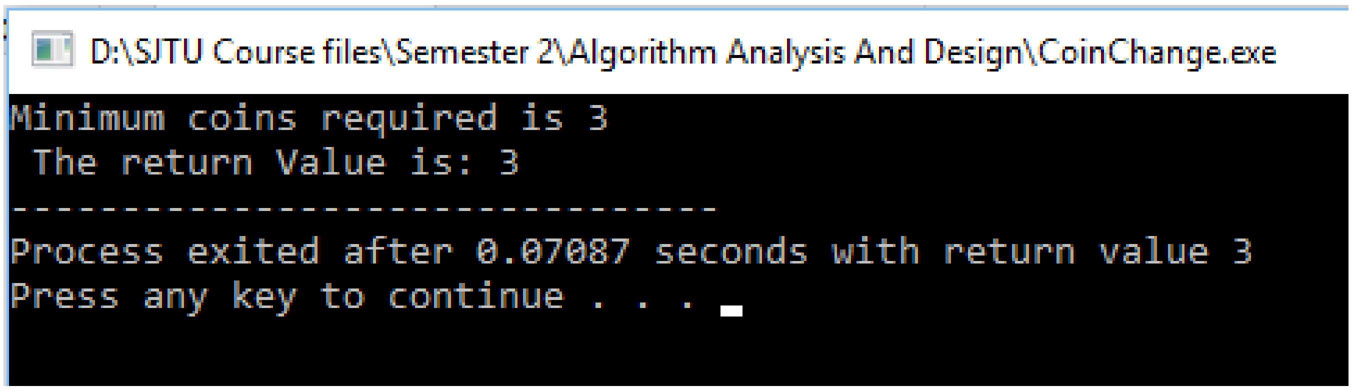
Explanation: If amount is 0, we will return 0 number of coins. We calculate minimum number of coins required for all values from 1 to amount (outer for loop). Next we loop over all the coins available for every value of money. Notice that we only need to consider those coin for particular amount whose value is less or equal to the amount. To see if the coin will be used for optimal solution of amount V, we compare coins[j] array from i and check its optimal amount of coins, call it sub_res. Next we compare sub_res with current optimal value for amount i, and update it if we can reduce it using sub_res + 1.Then we will call the function and m will have the size of the coins array. Notice that if number of coins required is 1 this means, currently there is no optimal solution for this value.
Time Complexity: $O(mV)$ as the algorithm requires to iterate over all possible values of amount starting from 1, for all available values of coins.



D:\SJTU Course files\Semester 2\Algorithm Analysis And Design\CoinChange.exe

```
 The return Value is: -1
--------------------------------
```

Figure 2: When the int Coins[]=2 and Amount V is 3



D:\SJTU Course files\Semester 2\Algorithm Analysis And Design\CoinChange.exe

```
Minimum coins required is 3
 The return Value is: 3
--------------------------------
Process exited after 0.07087 seconds with return value 3
Press any key to continue . . .
```

Figure 3: When the int Coins[]=1,2,5 and Amount V is 11

3. Given a sequence of n integers $\{a_1 \cdots a_n\}$, determine a subsequence (not necessarily contiguous) of maximum length in which the values in the subsequence form a strictly increasing sequence. For example, given sequence $\{1, 7, 3, 5, 9, 4, 8\}$, one subsequence of maximum length can be {1,3,5,9} and thus the corresponding length is 4.

   (a) Give an algorithm to find the maximum *length* of all subsequences and analyze both time and space complexity.

   (b) Wavio is a sequence of integers. It has some interesting properties.

   - Wavio is of odd length, i.e., $L = 2m + 1$.
   - The first $m + 1$ integers of Wavio sequence makes a strictly increasing sequence.
   - The last $m + 1$ integers of Wavio sequence makes a strictly decreasing sequence.
   - No two adjacent integers are the same in a Wavio sequence.

   For example {1, 2, 3, 4, 5, 4, 3, 2, 0} is an Wavio sequence of length 9. However, {1, 2, 3, 4, 5, 4, 3, 2, 2} is not a valid wavio sequence. Now, given a sequence of integers. You have to design an algorithm to find out the *length* of the longest Wavio sequence which is a subsequence of the given sequence. For example, the given sequence as {1, 2, 3, 2, 1, 2, 3, 4, 3, 2, 1, 5, 4, 1, 2, 3, 2, 2, 1}. Here the longest Wavio sequence is {1, 2, 3, 4, 5, 4, 3, 2, 1}. So, the output will be 9.
   Hint. Use the algorithm you have designed in part (a).

Solution: (a)
Algorithm: LIS Lengths

```
Algorithm 1: Algorithm For LIS Length
1 $INPUT$ : $Array[1..n]$ n $geq$ size of an array
2 $OUTPUT$ : Length of LIS
3 LIS[n] $\geq$ array of size n initialized to 1;
4 for $i \leftarrow 0$ to $n$ do do
5     for $j \leftarrow 1$ to $i$ do do
6         if Array[i]>Array[j] then
7             LIS[i] = max(LIS[i],LIS[j+1]);
8 return LIS[...];
```

Time Complexity: Algorithm clearly requires a run-time of $O(n^2)$ as obvious from two nested for loops.
Space Complexity: We only require to store LIS array of length n. Therefore space complexity is O(n).
Notice that array LIS contains size of maximal increasing sequences for all indices. To find maximum increasing sub-sequence, we just need to find maximum value within array.

(b):

Algorithm: LIS Lengths

```
Algorithm 2: Algorithm For WAVIO Length
1 $INPUT$ : $Array[1..n]$ n $geq$ size of an array
2 $OUTPUT$ : Length of Longest wavio
3 LIS_length $\geq$ LISLengths(Array[...],n)
4 LDS_length $\geq$ LISLengths(Array[...],n)
5 maxLength $\geq$ 0;
6 for $i \leftarrow 0$ to $n$ do do
7     maxLength = max(maxLength,min(LIS_length,LDS_length));
8 return (maxLength *2) -1;
```

Explanation : We use algorithm in part(a) to find LIS lengths of input arrays, then we do longest decreasing order by using the input array in reverse order.
Notice : Time complexity is O(n^2) (two O(n^2) of increasing/decreasing sequence and one O(n) to find the length) and space complexity is O(n) (two O(n) Array[1..n]).