CS 344 Assignment 5

December 3, 2018

Name : Yang Bao
NetID : yb184
Section : 02
The classmate I discussed with : Zitian Qin, Hao Jiang

Problem 1 :
In pseudocode of this question :
$T[N]$ is the smallest number of bills you need to make $\$N$
$S[N]$ is the value of bill been used based on the former choice.
The DP-relation: $T[i] = min(T[i-1], T[i-6], T[i-27], T[i-38], T[i-50]) + 1$
Assuming that when the index of array is negative, array always return $-\infty$.

```
MinBills(int N){
    Initialize table T of length N + 1;
    Initialize table S of length N + 1;
    T[0] = 0;
    S[0] = 0;
    for i = 1 to N{                          //O(N)
        T[i] = min(T[i-1],T[i-6],T[i-27],T[i-38],T[i-50]) + 1;
        s[i] = k belongs {1,6,27,38,50} such that T[i] = 1 + T[i - k];
    }
    Output # bills: return T[N]
    Output the sequence of the bills used :
        temp = N;
        while temp != 0
            Print S[N];
            temp -= S[N]};
}
```

The running time is $O(N)$.

Problem 2 :
Description of this question :

1. T[i][j] will store the maximum value if the available objects are $s_1, s_2, ..., s_i$ and the maximum weight of the knapsack is j.
   w is the weight of the current object we are considered.
   v is the value of the current object we are considered.
   Assuming that when any index of table is negative, table always return $-\infty$.

2. Initialize : $T[0][j]$ and $T[i][0]$ are all 0

3. Order : compute $T[1][j]$ first then $T[2][j]$, row by row.

4. Compute : T[i][j] = max(T[i-1][j], T[i-1][j-w]+v)

5. return the highest value of all T[i][j]

6. running time : $O(1)$ for every slot and $n \cdot W$ slots, so $O(nW)$ in total

Problem 3 :
Describing the Algorithm:

1. $T[i]$ will store the Maximum interval sum in $A[1]...A[i]$ that ends in $A[i]$
   Assuming that when the index of array is negative, array always return $-\infty$.

2. initialize $T[0] = A[0]$

3. Order : compute $T[1]$ then $T[2]$ then ... $T[n]$

4. Compute $T[i]$ by doing : $T[i] = max(A[i], A[i] + T[i-1])$

5. Return maximum of all $T[i]$

6. running time : $O(1)$ per $T[i]$, so $O(n)$ in total

Problem 4 :
Describing the Algorithm:

1. initialize 2-D table A of size length(S) by length(T)
2. $A[i][j]$ will store the Longest Common Substring in $S_1, S_2, ..., S_i$ and $T_1, T_2, ..., T_i$
3. initialize $A[i][0] =$ ”” and $A[0][i] =$ ””
4. Order : compute $A[i][j]$ by $A[i-1][kj]$ and $A[i][j-1]$
5. Compute $A[i][j]$ by doing :
   (a) Case 1: $S_i \neq T_i$. $A[i][j] \leftarrow$ the longer of $A[i][j-1]$ and $A[i-1][j]$
   (b) Case 2: $S_i = T_i$. $A[i][j] \leftarrow$ the longer of $A[i][j-1]$ and $A[i-1][j]$ + S_i
6. traverse the table A and return the longest substring
7. running time: $O(1)$ of all slot, $length(S) \cdot length(T)$ slots, so $O(length(S) \cdot length(T))$in total

The pseudocode is :
Assuming that when any index of table is negative, table always return ””.

```
LCS(int N){
    initialize 2-D table A of size length(S) by length(T)
    for i = 0 to length(S){
        A[i][0] = "";
    }
    for i = 0 to length(T){
        A[0][i] = "";
    }
    for i = 1 to length(S){
        for j = 1 to length(T){
            if(S[i] == T[j]) {
                if (length(A[i-1][j]) < length(A[i][j-1]){
                    A[i][j] = A[i][j-1] + S[i]
                }
                else{
                    A[i][j] = A[i-1][j] + S[i]
                }
            }
            else{
                if (length(A[i-1][j]) < length(A[i][j-1]){
```

```
                A[i][j] = A[i][j-1]
            }
            else{
                A[i][j] = A[i-1][j]
            }
        }
    }
}
string max = ""
for i = 1 to length(S){
    for j = 1 to length(T){
        if(length(A[i][j]) > length(max)){
            max = A[i][j]
        }
    }
}
return max;
```

Extra Credit :
This question is almost the same as the third question. We just need to make a new array that contains the value of $P[i] - P[i-1]$ and compute the Maximum Interval Sum and figure out the beginning and the end of the interval and return it.
So, here is the pseudocode :
Assuming that when the index of array is negative, array always return 0.

```
EC(array P){
    Initialize array A of size n
    for i from 0 to n-1{
        A[i] = P[i] - P[i-1];
    }
    Initialize table T of size n
    T[0] = 0;
    for i from 1 to n-1{
        T[i] = max(T[i-1] + A[i], A[i]);
    }
    int max = 0, begin = 0 , end = 0;
    for i from 0 to n-1{
        if (max < T[i]){
            end = i;
        }
    }
    for i from end to 0{
        if (T[i] = A[i]){
            begin = i;
            break;
        }
    }
    return (begin, end);
}
```

There are n slots and each slot is $O(1)$, so $O(n)$ in total.