German University in Cairo
Department of Computer Science
Dr. Nourhan Ehab

CSEN 703 Analysis and Design of Algorithms, Winter Term 2023
Practice Assignment 7

Exercise 7-1

Using the LCS algorithm discussed in class, get the LCS of "scooby-doo" and "snoopy".

Solution:

	ε	s	c	o	o	b	y	-	d	o	o
ε	0	0	0	0	0	0	0	0	0	0	0
s	0	1	1	1	1	1	1	1	1	1	1
n	0	1	1	1	1	1	1	1	1	1	1
o	0	1	1	2	2	2	2	2	2	2	2
o	0	1	1	2	3	3	3	3	3	3	3
p	0	1	1	2	3	3	3	3	3	3	3
y	0	1	1	2	3	3	4	4	4	4	4

Exercise 7-2

You are given a sequence of numbers $A =< a_1, a_2, a_3, \cdots, a_n >$. You are required to get the Longest increasing subsequence $B =< b_1, b_2, b_3, \cdots, b_n >$ where:

- $b_i < b_j$ given that $i \leq j$
- the numbers do not have to be contiguous in the original sequence

Provide the recursive definition and write down an algorithm that employs the dynamic programming paradigm.

Solution:

Solution 1:
Reduce the problem to the LCS problem. This can be done by finding the LCS between A and the sorted version of A in ascending order.

Solution 2:
The generic definition for getting the length of the Longest increasing subsequence can be stated as follows:

$$Lis[n] = \begin{cases} 1 & \text{if there is no i<n such that s[i]<s[n];} \\ 1 + max_{1<i<n}\{Lis[i] \mid s_i < s_n\} & \text{otherwise.} \end{cases}$$

```
function LIS(A)
    n ← A.length
    l ← empty array of integers of length n
    p ← empty array of integers of length n
    LIS-LENGTH(a, l, p)
    maxIndex ← 1
    for i ← 2 to n do
        if l[i] > l[maxIndex] then
            maxIndex ← i
        end if
    end for
    PRINT-LIS(a, p, maxIndex)
end function
function LIS-LENGTH(A, l, p)
    max ← 0
    source ← −1
    for i ← 1 to n do
        for j ← 1 to i − 1 do
            if A[j] < A[i] and l[j] > max then
                max ← l[j]
                source ← j
            end if
        end for
        l[i] ← max + 1
        p[i] ← source
    end for
end function
function PRINT-LIS(A, p, maxIndex)
    if maxIndex == −1 then
        print A[maxIndex]
    else
        PRINT-LIS(A, p, p[maxIndex])
        print A[maxIndex]
    end if
end function
```

Exercise 7-3

Serling Enterprises buys long steel rods and cuts them into shorter rods, which it then sells. Each cut is free. The management of Serling Enterprises wants to know the best way to cut up the rods. We assume that we know, for $i = 1, 2, \ldots n$, the price p_i in dollars that Serling Enterprises charges for a rod of length i inches. Rod lengths are always an integral number of inches. Given a rod of length n inches and a table of prices p_i for $i = 1, 2, \ldots n$ determine the maximum revenue r_n obtainable by cutting up the rod and selling the pieces.

Solution:

Check the CLRS Introduction to Algorithms book Chapter 15, Section 15.1.

Exercise 7-4

Suppose you have one machine and a set of n jobs $a_1, a_2, \cdots, a_n$ to process on that machine. Each job a_j has a processing time t_j, a profit p_j, and a deadline d_j. The machine can process only one job at a time, and job a_j must run uninterruptedly for t_j consecutive time units. If job a_j is completed by its deadline d_j, you receive a profit p_j, but if it is completed after its deadline, you receive a profit of 0. Give an algorithm to find the schedule that obtains the maximum amount of profit, assuming that all processing times are integers between 1 and n.

Solution:

The recursive definition for the problem:

$$time_{i,j} = \begin{cases} t_i & \text{if } i = j; \\ time_{i,j-1} + t_j & \text{if } (time_{i,j-1} + t_j) \leq d_j); \\ time_{i,j-1} & \text{ if } profit_{i,j-1} \geq profit_{i+1,j}; \\ time_{i+1,j} & \text{if } profit_{i,j-1} < profit_{i+1,j}. \end{cases}$$

$$profit_{i,j} = \begin{cases} p_i & \text{if } i = j; \\ profit_{i,j-1} + p_j & \text{if } (time_{i,j-1} + t_j) \leq d_j); \\ max(profit_{i,j-1}, profit_{i+1,j}) & \text{otherwise.} \end{cases}$$

The algorithm:

```
function DPScheduling(a,t,d,p)
    Sort a, t, d, p with respect to d ascendingly
    int n ← a.length
    profit ← (int) AllocateArray(n, n)
    time ← (int) AllocateArray(n, n)
    for i ← 1 to n do
        profit[i][i] = p_i
        time[i][i] = t_i
    end for
    for  i ← n − 1 downto 1 do
        for j ← i + 1 to n do
            if  ((time[i][j − 1] + t_j) ≤ d_j)  then
                profit[i][j] ← profit[i][j − 1] + p_j
                time[i][j] ← time[i][j − 1] + t_j
            else if  (profit[i][j − 1]  ≥  profit[i + 1][j]) then
                profit[i][j] ← profit[i][j − 1]
                time[i][j] ← time[i][j − 1]
            else
                profit[i][j] ← profit[i + 1][j]
                time[i][j] ← time[i + 1][j]
            end if
        end for
    end for
    return profit
end function
function BacktrackSchedule(profit)
    Set schedule ← {φ}
                                                    ▷ The set contents are unique
    index col ← n
    index row ← 1
    repeat
        if (row = col) then
            schedule ← a_col + schedule
        else if profit[row][col] = profit[row][col − 1] + p_col then
            schedule ← a_col + schedule
            col ← col − 1
        else if profit[row][col] = profit[row][col − 1] then
            col ← col − 1
        else
            row ← row − 1
        end if
```

```
    until (row = col)
    return schedule
end function
```