Assignment 4: Dynamic Programming, Greedy Algorithms

Abstract:

I have implemented the bottom-up version of the Smith-Waterman algorithm given by the recursive definition of the function ‘M’ (as seen on the slides).
I have implemented the top-down with memoization version of the Smith-Waterman algorithm given by the recursive definition of the function ‘M’.
Also, I, then recursively print the matching sequence that is derived from both ‘X’ and ‘Y’.
I have found the maximum alignment for a specific value of ‘X’ and ‘Y’ by using the Smith-Waterman algorithm.

Result:

A. The maximum alignment for X = dcdcbacbbb and Y= acdccabdbb by using the Smith-Waterman algorithm.

```
(base) dakshbhuva@Dakshs-Air cs590_hw4_code % ./hw4 10 10 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 2 1 0 0 0 2 1 0
0 0 2 1 4 3 2 1 1 1 0
0 0 1 4 3 3 2 1 3 2 1
0 0 2 3 6 5 4 3 2 2 1
0 0 1 2 5 5 4 6 5 4 4
0 2 1 1 4 4 7 6 5 4 3
0 1 4 3 3 6 6 6 5 4 3
0 0 3 3 2 5 5 8 7 7 6
0 0 2 2 2 4 4 7 7 9 9
0 0 1 1 1 3 3 6 6 9 11

 D D D L L D D D L L
 D D U D D L L U D D
 D U D U D D D D L L
 D D U D D L L U D D
 D U U U D D D L D D
 D L U U D D L D D D
 U D L D D U D D D D
 U U D D U D D L D D
 D U D D U D D D D D
 D U D D U D D D D D


X = dcdcbacbbb
X'= dcdcbacb-bb
Y'= acdcca-bdbb
Y = acdccabdbb
-----------------
M(n,m) = 11
```

Bottom-up SW

```
(base) dakshbhuva@Dakshs-Air cs590_hw4_code % ./hw4 10 10 1


X = dcdcbacbbb
X'= dcdcbacb-bb
Y'= acdcca-bdbb
Y = acdccabdbb
-----------------
M(n,m) = 11
```

Top-down SW

Discussion:

a. **Bottom-up version of the Smith-Waterman algorithm**
    - I have implemented the pseudo-code as given on the lecture slides for bottom-up version with some minor changes.
    - The time-complexity for Smith-Waterman algorithm is O (n x m) when $n \neq m$, and O (n^2) or O (m^2) when n = m.

b. **Top-down with memoization version of the Smith-Waterman algorithm**
    - I have implemented the top-down version using the Auxiliary recursive function for calling them.
    - The time-complexity for Smith-Waterman algorithm is O (n x m) when $n \neq m$, and O (n^2) or O (m^2) when n = m.

Conclusion:

Thus, the Smith-Waterman algorithm is Dynamic Programming algorithm that performs a local sequence alignment. Its practical application is to determine similar regions between two nucleotide or protein sequences.

HOMEWORK 4: DP, GREEDY ALGORITHMS

+ DAKSH BHUVA
+ 10475468

* Ex 15.1 - 2 :

→ Let the lenght of the rod be '4'. In DP, total price would be $\boxed{36}$ as shown below

Length i	1	2	3	4
Price pi	1	20	33	36
pi / li	1	10	11	9

→ Now by greedy strategy first ~~the~~ we will sort the piece lenghts and pieces according to density (pi/li) in decreasing order.

Length i	3	2	4	1
Price pi	33	20	36	1
pi / li	11	10	9	1

→ From this first cut a rod of lenght 3, for a ~~piece~~ price (pi) of 33.

→ Now, total rod length is '4' and we cut out the rod of lenght '3' and so remaining rod lenght is '1'.

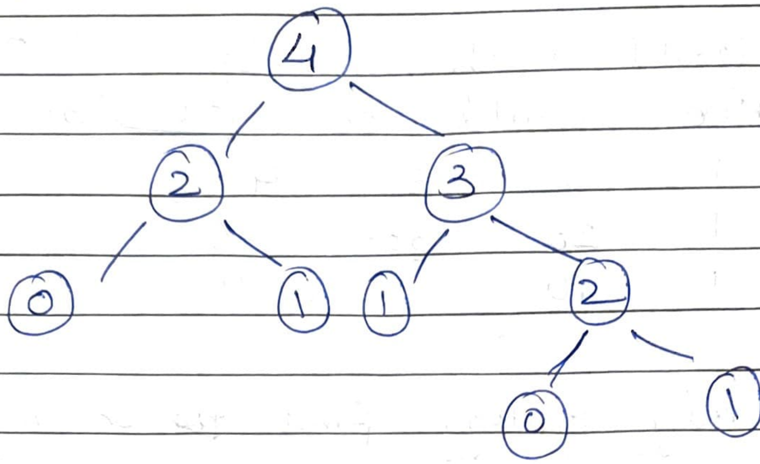
∴ Price to cut remaining rod is 1. (rod lenght = 1)

⇒ Total price is 33 + 1 = $\boxed{34}$ which is less than DP.

→ Thus, 'greedy' strategy does not always determine an optimal way to cut rods.

* Ex 15.1-5 :

→ If we consider $n=4$, then subproblem graph would look like



→ Vertices for n^{th} fibonacci series that follows recurrence:

$$V(n) = 1 + V(n-2) + V(n-1) \quad \ldots\ldots \quad (1)$$

→ we know that series is 0, 1, 1, 2, 3, 5 ....

$\therefore$ $V(0) = V(1) = 1$

→ Number of solutions ⇒

$$V(n) = 2 * \text{fibonacci}(n) - 1$$

→ Applying this in eq. (1), we get,

$$\therefore V(n) = 1 + [2 * \text{fibonacci}(n-2)] - 1 + [2 * \text{fibonacci}(n-1)] - 1$$

$\therefore$ Vertices, $V(n) = 2 * \text{fibonacci}(n) - 1$

→ The number of edges that signifies recurrence

$\therefore E(n) = 2 + E(n-1) + E(n-2)$ .... (2)

→ Similarly for ~~vertices~~ edges. $E(0) = E(1) = 0$.

→ Applying it in eq. (2).

$\therefore E(n) = 2 * \text{Fibonacci}(n) - 2$

→ no. of edges

⇒ Algorithm :-

· Algo (FIBONACCI (x))

(1) let F[0.... x+1] be an array.

(2) F[0] = 0 and F[1] = 1

(3) for (2 ≤ i ≤ x) do

(4) F[i] = F[i-1] + F[i-2]

(5) return F[x].

* Ex 15.4-1 :

→ LCS of ⟨1, 0, 0, 1, 0, 1, 0, 1⟩ and ⟨0, 1, 0, 1, 1, 0, 1, 1, 0⟩

		1	0	0	1	0	1	0	1
	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1	1	1
1	0	1 ←	1	1	2	2	2	2	2
0	0	1	2	2	2	3	3	3	3
1	0	1	2	2	3	4	4	4	4
1	0	1	2	2	3	3	4	4	5
0	0	1	2	3	3	4	4	5	5
1	0	1	2	3	4	4	5	5	6
1	0	1	2	3	4	4	5	5	6
0	0	1	2	3	4	5	5	6	← 6

→ LCS is $\boxed{6}$

→ Therefore, the LCS is ⟨1, 0, 1, 0, 1, 0⟩.