HW 4
CS 381

Tian Qiu

1.

(a). (b).

	j	0	1	2	3	4	5	6	7	8	9
I		Y	M	I	N	N	E	S	O	T	A
0	X	0	0	0	0	0	0	0	0	0	0
1	W	0	↑0	↑0	↑0	↑0	↑0	↑0	↑0	↑0	↑0
2	A	0	↑0	↑0	↑0	↑0	↑0	↑0	↑0	↑0	↖ 1
3	S	0	↑0	↑0	↑0	↑0	↑0	↖ 1	↑ 1	↑ 1	↑ 1
4	H	0	↑0	↑0	↑0	↑0	↑0	↑ 1	↑ 1	↑ 1	↑ 1
5	I	0	↑0	↖ 1	← 1	← 1	← 1	↑ 1	↑ 1	↑ 1	↑ 1
6	N	0	↑0	↑ 1	↖ 2	↖ 2	← 2	← 2	← 2	← 2	← 2
7	G	0	↑0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 2	↑ 2	↑ 2	↑ 2
8	T	0	↑0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 2	↑ 2	↖ 3	← 3
9	O	0	↑0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 2	↖ 2	↑ 3	↑ 3
10	N	0	↑0	↑ 1	↖ 2	↖ 3	← 3	← 3	↑ 3	↑ 3	↑ 3

(c).

Another subsequence “INO” “INT” are not discovered.

Because in the algorithm,

```
if c[i-1, j] >= c[I, j-1]
then    go up
```

We miss the “O”.

Because the 4th N in MINNESOTA first matching the 6th N in WASHINGTON.

The second N missed.

Our goal is to find one LCS but not all LCS. So the algorithm makes sense.

2.

First come up to the brain. But time is O(2^n) which is because do not use memorizing method.

```
Count = 0
Function wordbreak(string, N) // N is the size of string
    For I to N:
        Prefix = string.substring(0,I)
        If prefix is a word:
            If I == N:
                Count ++
                Return
        WordBreak( string.substring(I, N-I), N – 1)
        End
    End
End
```

So redo the problem by another way of thinking.

Step1: Characterize the structure of optimal solution

Set function F[i] which means how many ways that substring 0 to index “i” can be separated perfectly

Step2: Recursively define the value of optimal solution

```
    If string.substring(j, I) is a word && F[j] != 0:
                F[I] += F[j]
```

Step3: Compute the value by bottom-up way

```
    For I from 1 to N:
        For j from 0 to I-1:
            If string.substring(j, I) is a word && F[j] != 0:
                F[I] += F[j]
        End
    End
```

Step4: Construct the solution from step 3

F[n] is exactly how many ways one could break the string into a sequence of words.

Pseudo code:

```
Function WordBreak (string)
    N = length of string
    // initial array F to record how many ways that string can be separated to word
    For I = 1 to N + 1:
        F [I] = 0
    End

    // for prefix start from 0
    F[0] = 1

    For I from 1 to N:
        For j from 0 to I-1:
            If string.substring(j, I) is a word && F[j] != 0:
                F[I] += F[j]
        End
    End

    Return F[n]
```

Time analyze:

Because it has two loops and inside the loop, the time is constant. Also the initialization time is constant.

T(n) = O(N^2)