CS4311
Design and Analysis of Algorithms

Lecture 12: Dynamic Programming IV

Subsequence of a String

- Let $S = s_1s_2...s_m$ be a string of length m
- Any string of the form

$$s_{i_1}\ s_{i_2}\ \ldots\ s_{i_k}$$

with $i_1 < i_2 < \ldots < i_k$ is a subsequence of S

- E.g., if S = farmers

➔ fame, arm, mrs, farmers, are some of the subsequences of S

Longest Common Subsequence

- Let S and T be two strings
- If a string is both
    - a subsequence of S and
    - a subsequence of T,

  it is a common subsequence of S and T

- In addition, if it is the longest possible one, it is a longest common subsequence

Longest Common Subsequence

- E.g.,

  S = algorithms

  T = logarithms

- Then, aim, lots, ohms, grit, are some of the common subsequences of S and T
- Longest common subsequences:

  lorithms , lgrithms

Longest Common Subsequence

- Let $S = s_1s_2...s_m$ be a string of length m
- Let $T = t_1t_2...t_n$ be a string of length n

Can we quickly find a longest common subsequence (LCS) of S and T ?

Optimal Substructure

Let $X = x_1x_2 \ldots x_k$ be an LCS of

$S_{1,i} = s_1\, s_2 \ldots s_i$ and $T_{1,j} = t_1t_2 \ldots t_j$.

Lemma:

- If $s_i = t_j$, then $x_k = s_i = t_j$, and $x_1x_2 \ldots x_{k-1}$ must be the LCS of $S_{1,i-1}$ and $T_{1,j-1}$
- If $s_i \neq t_j$, then X must either be
  - (i) an LCS of $S_{1,i}$ and $T_{1,j-1}$, or
  - (ii) an LCS of $S_{1,i-1}$ and $T_{1,j}$

Optimal Substructure

Let $len_{i,j}$ = length of the LCS of $S_{1,i}$ and $T_{1,j}$

Lemma: For any $i, j \geq 1$,

- if $s_i = t_j$, $len_{i,j} = len_{i-1,j-1} + 1$
- if $s_i \neq t_j$, $len_{i,j} = \max\{ len_{i,j-1}, len_{i-1,j} \}$

Length of LCS

Define a function Compute_L(i,j) as follows:

Compute_L(i, j) /* Finding $len_{i,j}$ */

1. if (i == 0 or j == 0) return 0; /* base case */
2. if (s_i == t_j)

   return Compute_L(i-1,j-1) + 1;
3. else

   return max {Compute_L(i-1,j), Compute_L(i,j-1)};

Compute_L(m, n) runs in $O(2^{m+n})$ time

Overlapping Subproblems

To speed up, we can see that :

To Compute_L(i,j) and Compute_L(i-1,j+1), has a common subproblem:

Compute_L(i-1,j)

In fact, in our recursive algorithm, there are many redundant computations !

Question: Can we avoid it ?

Bottom-Up Approach

- Let us create a 2D table L to store all $len_{i,j}$ values once they are computed

BottomUp_L( ) `/* Finding min #operations */`

1. For all i and j, set L[i,0] = L[0, j] = 0;
2. for (i = 1,2,..., m)

    Compute L[i,j] for all j;

    `// Based on L[i-1,j-1], L[i-1,j], L[i,j-1]`

4. return L[m,n] ;

Running Time = $\Theta(mn)$

Remarks

- Again, a slight change in the algorithm allows us to obtain a particular LCS
- Also, we can make minor changes to the recursive algorithm and obtain a memoized version (whose running time is $O(mn)$)

Example Run: After Step 1

		D	O	R	M	I	T	O	R	Y
	0	0	0	0	0	0	0	0	0	0
D	0									
I	0									
R	0									
T	0									
Y	0									
R	0									
O	0									
O	0									
M	0									

Example Run: After Step 2, i = 1

		D	O	R	M	I	T	O	R	Y
	0	0	0	0	0	0	0	0	0	0
D	0	1	1	1	1	1	1	1	1	1
I	0									
R	0									
T	0									
Y	0									
R	0									
O	0									
O	0									
M	0									

Example Run: After Step 2, i = 2

		D	O	R	M	I	T	O	R	Y
	0	0	0	0	0	0	0	0	0	0
D	0	1	1	1	1	1	1	1	1	1
I	0	1	1	1	1	2	2	2	2	2
R	0									
T	0									
Y	0									
R	0									
O	0									
O	0									
M	0									

Example Run: After Step 2, i = 3

		D	O	R	M	I	T	O	R	Y
	0	0	0	0	0	0	0	0	0	0
D	0	1	1	1	1	1	1	1	1	1
I	0	1	1	1	1	2	2	2	2	2
R	0	1	1	2	2	2	2	2	3	3
T	0									
Y	0									
R	0									
O	0									
O	0									
M	0									

Example Run: After Step 2, i = 4

		D	O	R	M	I	T	O	R	Y
	0	0	0	0	0	0	0	0	0	0
D	0	1	1	1	1	1	1	1	1	1
I	0	1	1	1	1	2	2	2	2	2
R	0	1	1	2	2	2	2	2	3	3
T	0	1	1	2	2	2	3	3	3	3
Y	0									
R	0									
O	0									
O	0									
M	0									

Example Run: After Step 2

		D	O	R	M	I	T	O	R	Y
	0	0	0	0	0	0	0	0	0	0
D	0	1	1	1	1	1	1	1	1	1
I	0	1	1	1	1	2	2	2	2	2
R	0	1	1	2	2	2	2	2	3	3
T	0	1	1	2	2	2	3	3	3	3
Y	0	1	1	2	2	2	3	3	3	4
R	0	1	1	2	2	2	3	3	4	4
O	0	1	2	2	2	2	3	4	4	4
O	0	1	2	2	2	2	3	4	4	4
M	0	1	2	2	3	3	3	4	4	4

Extra information to obtain an LCS

		D	O	R	M	I	T	O	R	Y
	0	0	0	0	0	0	0	0	0	0
D	0	1↖	1←	1←	1←	1←	1←	1←	1←	1←
I	0									
R	0									
T	0									
Y	0									
R	0									
O	0									
O	0									
M	0									

Extra Info: After Step 2, i = 2

		D	O	R	M	I	T	O	R	Y
	0	0	0	0	0	0	0	0	0	0
D	0	1↖	1←	1←	1←	1←	1←	1←	1←	1←
I	0	1↑	1↑	1↑	1↑	2↖	2←	2←	2←	2←
R	0									
T	0									
Y	0									
R	0									
O	0									
O	0									
M	0									

Extra Info: After Step 2, i = 3

		D	O	R	M	I	T	O	R	Y
	0	0	0	0	0	0	0	0	0	0
D	0	1↖	1←	1←	1←	1←	1←	1←	1←	1←
I	0	1↑	1↑	1↑	1↑	2↖	2←	2←	2←	2←
R	0	1↑	1↑	2↖	2←	2↑	2↑	2↑	3↖	3←
T	0									
Y	0									
R	0									
O	0									
O	0									
M	0									

Extra Info: After Step 2

		D	O	R	M	I	T	O	R	Y
	0	0	0	0	0	0	0	0	0	0
D	0	1↖	1←	1←	1←	1←	1←	1←	1←	1←
I	0	1↑	1↑	1↑	1↑	2↖	2←	2←	2←	2←
R	0	1↑	1↑	2↖	2←	2↑	2↑	2↑	3↖	3←
T	0	1↑	1↑	2↑	2↑	2↑	3↖	3←	3←	3←
Y	0	1↑	1↑	2↑	2↑	2↑	3↑	3←	3←	4↖
R	0	1↑	1↑	2↑	2↑	2↑	3↑	3↑	4↖	4↑
O	0	1↑	2↖	2↑	2↑	2↑	3↑	4↖	4↑	4↑
O	0	1↑	2↖	2↑	2↑	2↑	3↑	4↑	4↑	4←
M	0	1↑	2↑	2↑	3↖	3←	3←	4↑	4↑	4↑

LCS obtained by tracing from L[m,n]

		D	O	R	M	I	T	O	R	Y
	0	0	0	0	0	0	0	0	0	0
D	0	1↖	1←	1←	1←	1←	1←	1←	1←	1←
I	0	1↑	1↑	1↑	1↑	2↖	2←	2←	2←	2←
R	0	1↑	1↑	2↖	2←	2↑	2↑	2↑	3↖	3←
T	0	1↑	1↑	2↑	2↑	2↑	3↖	3←	3←	3←
Y	0	1↑	1↑	2↑	2↑	2↑	3↑	3←	3←	4↖
R	0	1↑	1↑	2↑	2↑	2↑	3↑	3↑	4↖	4↑
O	0	1↑	2↖	2↑	2↑	2↑	3↑	4↖	4↑	4↑
O	0	1↑	2↖	2↑	2↑	2↑	3↑	4↑	4↑	4←
M	0	1↑	2↑	2↑	3↖	3←	3←	4↑	4↑	4↑