Tutorial on Dynamic Programming

TA: Bishal Santra
Sep-17, 2021

Agenda

1. Longest Common Subsequence
2. Last Year's Assignment

Basic Routine for most DP problems

- Write down the recursive formulation of the solution.
  - Make sure that the optimal substructure property holds.
- Figure out the repeated subproblems.

Memoization vs. Tabulation

Writing the loop for *Memoization*

- Top-Down
- Calls made are similar to the actual recursive implementation.
- Just remember the solutions to previous calls to the recursive function.

Writing the loop for *Tabulation*

- Bottom-up
- You know all the subproblems, so start solving from the base-case.

Longest Common Subsequence (LCS)

Given two input strings A and B, identify the longest common subsequence.

Example

- ACT, $ATTC$, T, $ACTTGC$ are all subsequences.
- TTA is not a subequence

Longest Common Subsequence (LCS)

Find a subsequence of maximal length, appearing in both A and B.

Examples:

LCS for input sequences “ABCDGH” and “AEDFHR” is “ADH” of length 3.

LCS for input sequences “AGGTAB” and “GXTXAYB” is “GTAB” of length 4.

LCS

Brute Force

1. Enumerate all subsequence of A
2. Check if it is also a subsequence of B

What would be the overall complexity?

1. Finding all subsequence of A is a combinatorial problem.
2. Matching can be solved in linear time w.r.t the length of B.

LCS - Recursive Solution

- Note the optimal substructure.
  - Try to prove this at home.
- This leads us to a recursive solution.

Let $X =< x_1, x_2, \ldots, x_m >$ and $Y =< y_1, y_2, \ldots, y_n >$ be sequences, and let $Z =< z_1, z_2, \ldots, z_k >$ be any LCS of X and Y.

1. **If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is an LCS of X_{m-1} and Y_{n-1}.**
2. **If $x_m \neq y_n$, then $z_k \neq x_m$ implies that Z is an LCS of X_{m-1} and Y.**
3. **If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is an LCS of X and Y_{n-1}.**

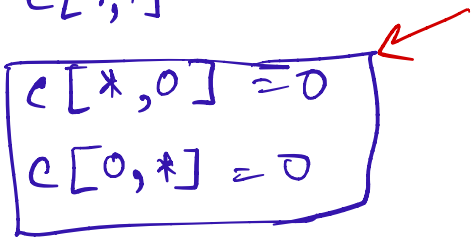
X_{M-1}

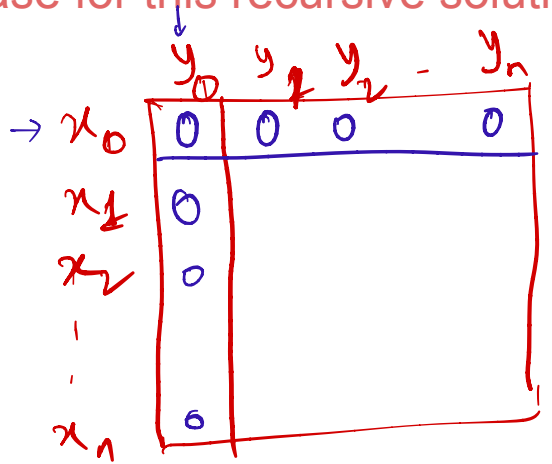
$x_1 \ldots x_{M-1}$

LCS - Recursive Solution

$$c[i,j] = \begin{cases} c[i-1,j-1]+1 & \text{if } i,j>0 \text{ and } x_i = y_j \,, \\ \max(c[i,j-1], c[i-1,j]) & \text{if } i,j>0 \text{ and } x_i \neq y_j \,. \end{cases}$$

- What would be the base case for this recursive solution?





LCS - Recursive Solution

$$c[i,j] = \begin{cases} 0 & \textbf{if } i = 0 \textbf{ or } j = 0 \ , \\ c[i-1,j-1]+1 & \textbf{if } i,j > 0 \textbf{ and } x_i = y_j \ , \\ \max(c[i,j-1], c[i-1,j]) & \textbf{if } i,j > 0 \textbf{ and } x_i \neq y_j \ . \end{cases}$$

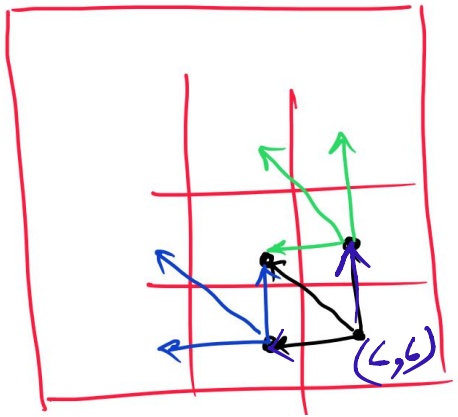
Overlapping Subproblems

- LCS for input sequences “ABCDGH” and “AEDFHR” is “ADH” of length 3.

$$c[6,6] = c[5,5] + 1 \qquad x_6 = y_6$$

$$\text{or } \max(c[6,5], c[5,6])$$

- Recursive calls solve the same subproblems repeatedly.



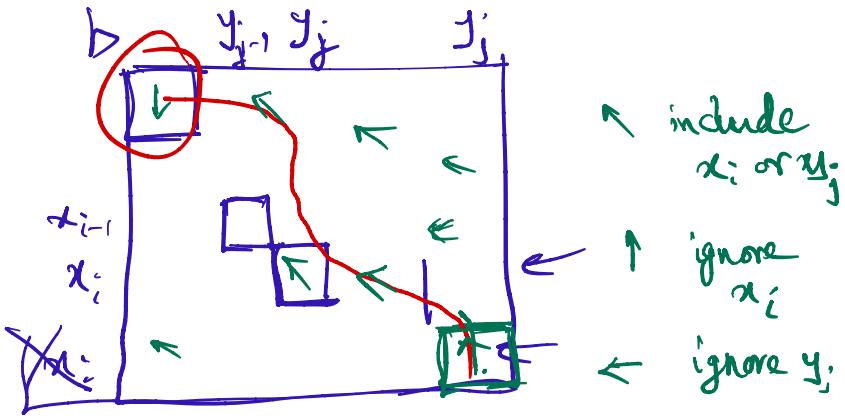
Hence,

1. We cache previously solved subproblems,
2. Or, we solve it bottom up.

Dynamic Programming Solution

```
LCS − Length(X, Y)
1   m ← length[X]
2   n ← length[Y]
3   for i ← 1 to m
4         do c[i, 0] ← 0
5   for j ← 0 to n
6         do c[0, j] ← 0
7   for i ← 1 to m
8         do for j ← 1 to n
9               do if x_i = y_j
10                    then c[i, j] ← c[i − 1, j − 1] + 1
11                         b[i, j] ← "↖"
12                    else if c[i − 1, j] ≥ c[i, j − 1]
13                            then c[i, j] ← c[i − 1, j]
14                                 b[i, j] ← "↑"
15                            else c[i, j] ← c[i, j − 1]
16                                 b[i, j] ← "←"
17  return c and b
```

1. Is this solution based on *tabulation* or *memoization*?
2. What is the use of array b?



DP Lab Assignment - 2020
Solution Code

Problem Statement

Recently you have joined a chemistry lab and they are leading a research on a system which consists of a sequence of independent and essential chemical reactions. Surprisingly the researchers of the lab have discovered a unique catalyst and it can be included in any of the reactions of the system. The use of the catalyst enhances the rate of the reaction and hence its probability of success also. Also they have proven that the probability of success of the reactions depends on the usage of the catalyst, typically expressed in terms of number of units involved in the reaction. The more the quantity (units) of catalyst is used in a reaction, the probability of success increases or remains the same, but never decreases. The probability of success of the entire system is the product of the probability of success of each of the individual reactions. Given the limited quantity (units) of the catalyst, the researchers want to gain the maximum success of the system. As an expert in algorithms, you are given the task of maximizing the success probability of the system with the constrained amount of the catalyst. You have to formulate a **Dynamic Programming solution** for the problem (use of memoization is **NOT** allowed).

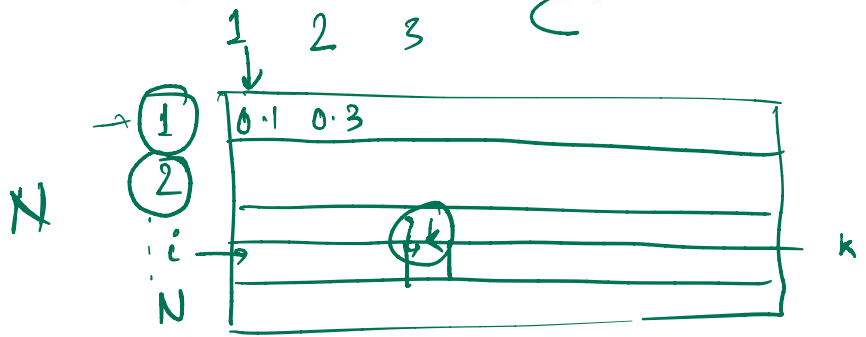
You are given N reactions, numbered from 1 to N, all of which must be completed for the successful completion of reactions of the system. The system has a total of C units of catalyst ($C \geq N$) that can be used in different reactions. Let $e(k, p)$ denote the probability of success of the reaction k if p units of catalyst are used in the reaction. It is given that $e(k, 0) = 0$ for all k, so at least one unit of catalyst must be used by each of the reaction. Also, for a reaction k, $e(k, p)$ values are non-decreasing with increasing values of p (i.e., the chance of success never decreases if more units of catalyst are assigned to the reactions). You need to assign the C units of the catalyst to the N reactions so that the probability of success of the system is maximized.

You are required to design an efficient dynamic programming algorithm for the problem.

Part II

Write a `main()` function to implement the dynamic programming. Read the input file as follows

1. The first line contains the number of reactions N (assume $N < 10$)
2. Read in C (assume $C < 30$. You can also assume C will be entered as $\geq N$, no need to check).
3. Read in the $e(k, p)$ values for each reaction exactly in this order in a 2-d array (enter the values in each row in non-decreasing order):
   $e(1,1), e(1,2), ..., e(1,C)$
   $e(2,1), e(2,2), ..., e(2,C)$
   ....
   $e(N,1), e(N,2), ..., e(N,C)$



Sample input file

FILE: *input.txt*

```
3
5
0.1 0.3 0.5 0.6 0.6
0.1 0.2 0.4 0.6 0.8
0.2 0.4 0.4 0.7 0.9
```

You have to print the followings exactly same as shown in the sample output file

1. The maximum probability of success, of the system.
2. The assignment of the catalyst in units to each of the reaction that gives the maximum success probability of the system.

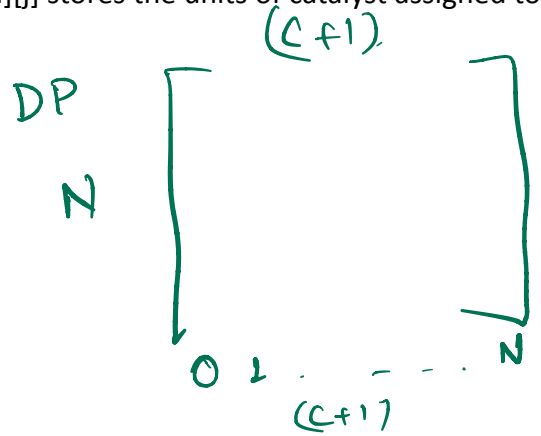
Sample output file

FILE: *output.txt*

```
0.012
reaction 1 : 2
reaction 2 : 2
reaction 3 : 1
```

Recursive Solution and Overlapping Subproblems

DP[i][j] as the max success-probability if reaction i has j units of catalyst.
comp[i][j] stores the units of catalyst assigned to the ith reaction such that we can get the max success-probability.



DP[i][j] → upto reaction i, max catalyst used j

$j \le C$

$DP[i][j] = \max\{$
- $z=0$: $DP[i-1][j-1] \cdot e[i,1]$,
- $z=1$: $DP[i-1][j-2] \cdot e[i,2]$
- ⋮
- $z=j-i$: $DP[i-1][i-1] \cdot e[i,j-i+1]$

For reaction i
z → Extra amt catalyst
z : 0 to (j-i)

i catalyst.
(j-i)

$0 \le z \le j-i$

D[i][j] = max { D[i-1] . . .

already optimal.

→ D[i-1, —] e[i; 3*] ← z*

D*[i-1, —]

}