CSCI 115 Lab

Week 12- Dynamic Programming

- Professor: Dr. Matin Pirouz
  Email: mpirouz@csufresno.edu

- TA: Shreeja Miyyar
  Email:shreejarao12@mail.fresnostate.edu

Table of Contents

Dynamic Programming

- Break up a problem into a series of overlapping sub-problems and build up solutions to larger and larger sub-problems.

- Unlike divide and conquer, sub-problems are not independent; Sub-problems may share sub-sub-problems.

- We solve the problem by solving sub-problems of increasing size and saving each optimal solution in a table (usually).

- The table is then used for finding the optimal solution to larger problems. Time is saved since each sub-problem is solved only once.

- Common Examples of Dynamic Programming:
  1. Matrix Chain Multiplication
  2. Longest Common Subsequence
  3. Assembly Line Scheduling

Elements of Dynamic Programming

DP is used to solve problems with the following characteristics:

1. Simple subproblems

   -We should be able to break the original problem into smaller subproblems that have same structure.

2. Optimal sub structure of the problems

   -The optimal solution to the problem contains within it an optimal solution to its subproblems.

3. Overlapping subproblems

   -There exist some places where we can solve the same sub problem more than once.

Longest Common Subsequence

Sequence:

An ordered list of things. These can be numbers or characters.

Subsequence:

A subsequence of sequence is a sequence that appears in the same relative order, but not necessarily contiguous.

Ex: In “abcdefg”, few subsequences are “abc” ,”abg”, “bdf”, “aeg”. But “aa” and ”fa” are not.

Longest Common Subsequence:

This is the problem of finding the longest common subsequence of two sequence of items.

Example

H="BADRULS"

Z="ADUSLR"

(ADUS) and (ADUL) are the longest common subsequence of H and Z whose length is 4.

H="BADRULS"

Z="ADUSIR"

(ADR) is not LCS of H and Z since its length is equal to 3.

Brute Force Solution

- For every subsequence of one string S1, we check if it's a subsequence of another string S2.
- We will have 2^m subsequence in S1 where m is the length of S1.
- Each subsequence takes O(n) time.
- Total running time is $O(n* 2^m)$

Recursive Solution

Let us consider two sequence X[0..m-1] and Y[0…n-1] of length m and n. C[X[0..m-1],Y[0..n-1]] is the length of LCS of X and Y.

- If last characters of both sequences match i.e., X[m-1] == Y[n-1] then

To find their LCS, shorten each sequence by removing the last element, find the LCS of shortened sequence and to that append the removed one element length.

C(X[0..m-1], Y[0..n-1]) = 1 + C(X[0..m-2], Y[0..n-2])

- If last characters of both sequences do not match i.e., X[m-1] != Y[n-1]) then

The LCS of X and Y is the maximum/longest of the two sequence C(X[0..m-2], Y[0..n-1]) and C(X[0..m-1], Y[0..n-2])

C(X[0..m-1], Y[0..n-1]) = MAX (C(X[0..m-2], Y[0..n-1]), C(X[0..m-1], Y[0..n-2]))

Ex: X=”ABDG” and Y=“ZBD”, we must solve LCS of ABD and ZBD first followed by LCS of ABDG and ZB and choose the maximum.

- Time Complexity of Recursive Solution is $O(2^n)$
- This happens when all characters in both sequence are different.

```
                        lcs("AXYT", "AYZX")
                       /
          lcs("AXY", "AYZX")              lcs("AXYT", "AYZ")
          /                                /
lcs("AX", "AYZX") lcs("AXY", "AYZ")   lcs("AXY", "AYZ") lcs("AXYT", "AY")
```

- Many subproblems solved again and again.
- We can use DP so that subproblem solutions are memorized rather than computing every time.

Dynamic Programming Solution

X="XMJYAUZ"
Y="MZJAWXU"

Create a matrix C[i,j]
LCS is MJAU
C[i][j] gives the length of the LCS.

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

		0	1	2	3	4	5	6	7
		Ø	M	Z	J	A	W	X	U
0	Ø	0	0	0	0	0	0	0	0
1	X	0	0	0	0	0	0	1	1
2	M	0	1	1	1	1	1	1	1
3	J	0	1	1	2	2	2	2	2
4	Y	0	1	1	2	2	2	2	2
5	A	0	1	1	2	3	3	3	3
6	U	0	1	1	2	3	3	3	4
7	Z	0	1	2	2	3	3	3	4

Lab Assignment

Hints and Coding Guidelines:

- Create a function to find the LCS which takes two strings and its size as the parameter.
- Create a matrix of size+1
- Write nested for loop to fill up the matrix in bottom-up fashion.
- Write if-else conditions to satisfy the criteria's in previous slide.
- You can also write a method to calculate the maximum of two values to be used in if-else condition above.
- Return the last entry in the matrix which is the length of LCS.
- In the main function define two strings and call the LCS method.

Questions?