CPSC 490 – Problem Solving in Computer Science

Lecture 6: Dynamic Programming (Part 2)

Jason Chiu and Raunak Kumar

2017/01/18

University of British Columbia

Announcements

Assignment 1 and 2

- Assignment 2 was released on Monday
- You should be finishing Assignment 1 ASAP!
- If you leave it to the weekend you may not get enough help!
- A1 in Python: B/D/F/G possible, A/C impossible, E/H not sure

Grading

- Marks over 100 in any assignment will "overflow" to other ones.
  HW part of final grade $= (A_1 + A_2 + A_3 + A_4 + A_5 + A_6)/600 \times 80\%$
- Do NOT count on solving all problems: some may be very hard
- There will be no extensions and no scaling at all.

Example 1 – Longest Common Subsequence

Find the length of the largest common subsequence of two strings a and b of lengths m and n respectively.

$$a = \text{XMJYAUZ}$$

$$b = \text{MZJAWXU}$$

$$\text{LCS} = \text{MJAU}.$$

Example 1 – LCS Solution

DP state: $f(i, j)$ = length of the LCS of $a[1 \ldots i]$ and $b[1 \ldots j]$.

Example 1 – LCS Solution

DP state: $f(i, j)$ = length of the LCS of $a[1 \ldots i]$ and $b[1 \ldots j]$.

Base case: $f(0, j) = f(i, 0) = 0$.

Example 1 – LCS Solution

DP state: $f(i, j)$ = length of the LCS of $a[1 \dots i]$ and $b[1 \dots j]$.

Base case: $f(0, j) = f(i, 0) = 0$.

To compute $f(n)$, either $a[i] = b[j]$ and we can consider the LCS of the 2 shorter strings, or $a[i] \neq b[j]$ and we can choose between making 1 of the strings shorter.

Example 1 – LCS Solution

DP state: $f(i, j)$ = length of the LCS of $a[1 \dots i]$ and $b[1 \dots j]$.

Base case: $f(0, j) = f(i, 0) = 0$.

To compute $f(n)$, either $a[i] = b[j]$ and we can consider the LCS of the 2 shorter strings, or $a[i] \neq b[j]$ and we can choose between making 1 of the strings shorter.

Recurrence relation: $f(i, j) = \begin{cases} f(i-1, j-1) + 1 & \text{if } a[i] = b[j] \\ \max\{f(i-1, j), f(i, j-1)\} & \text{otherwise} \end{cases}$.

Return $f(m, n)$.

Time Complexity: $O(mn)$.

Example 2 – Edit Distance

You are given 2 strings a and b of lengths m and n respectively.

We want to transform a to b by using edit operations: insert/delete/substitute a single character.

What is the minimum number of edit operations required?

a = kitten, b = sitting, edit distance = 3.

kitten $\rightarrow$ sitten

sitten $\rightarrow$ sittin

sittin $\rightarrow$ sitting

Example 2 – Edit Distance Solution

DP state: $f(i, j)$ = minimum number of edit operations required to transform $a[1 \ldots i]$ to $b[1 \ldots j]$.

Example 2 – Edit Distance Solution

DP state: $f(i, j)$ = minimum number of edit operations required to transform $a[1 \ldots i]$ to $b[1 \ldots j]$.

Base case: $f(i, 0) = i$, $f(0, j) = j$.

Example 2 – Edit Distance Solution

DP state: $f(i, j)$ = minimum number of edit operations required to transform $a[1 \dots i]$ to $b[1 \dots j]$.

Base case: $f(i, 0) = i$, $f(0, j) = j$.

To compute $f(i, j)$, either $a[i] = b[j]$ and we recurse on the smaller substrings or $a[i] \neq b[j]$ and we try using all possible edit operations.

Example 2 – Edit Distance Solution

DP state: $f(i, j)$ = minimum number of edit operations required to transform $a[1 \dots i]$ to $b[1 \dots j]$.

Base case: $f(i, 0) = i$, $f(0, j) = j$.

To compute $f(i, j)$, either $a[i] = b[j]$ and we recurse on the smaller substrings or $a[i] \neq b[j]$ and we try using all possible edit operations.

Recurrence relation:

$$f(i,j) = \begin{cases} f(i-1, j-1) & \text{if } a[i] = b[j] \\ 1 + \max\{f(i-1,j), f(i,j-1), f(i-1,j-1)\} & \text{otherwise} \end{cases}.$$

Return $f(m, n)$.

Time Complexity: $O(mn)$.

Problem 1

Find the LCS of two strings that does not contain the string “490”

Problem 1 – Solution

Idea: keep track of which character of “490” we are matching

DP state: $f(i, j, k)$ = length of LCS of $a[1 \ldots i]$ and $b[1 \ldots j]$ that ends with the first k characters of “490”

When we add one character, use k to figure out how many characters of “490” the suffix will match! If we know what was the maximum prefix match before, we know what it is after!

Problem 1 – Solution (continued)

Specifically, say we are at state k, which represent

$$k = \text{length of max prefix of “490” that matches suffix}$$

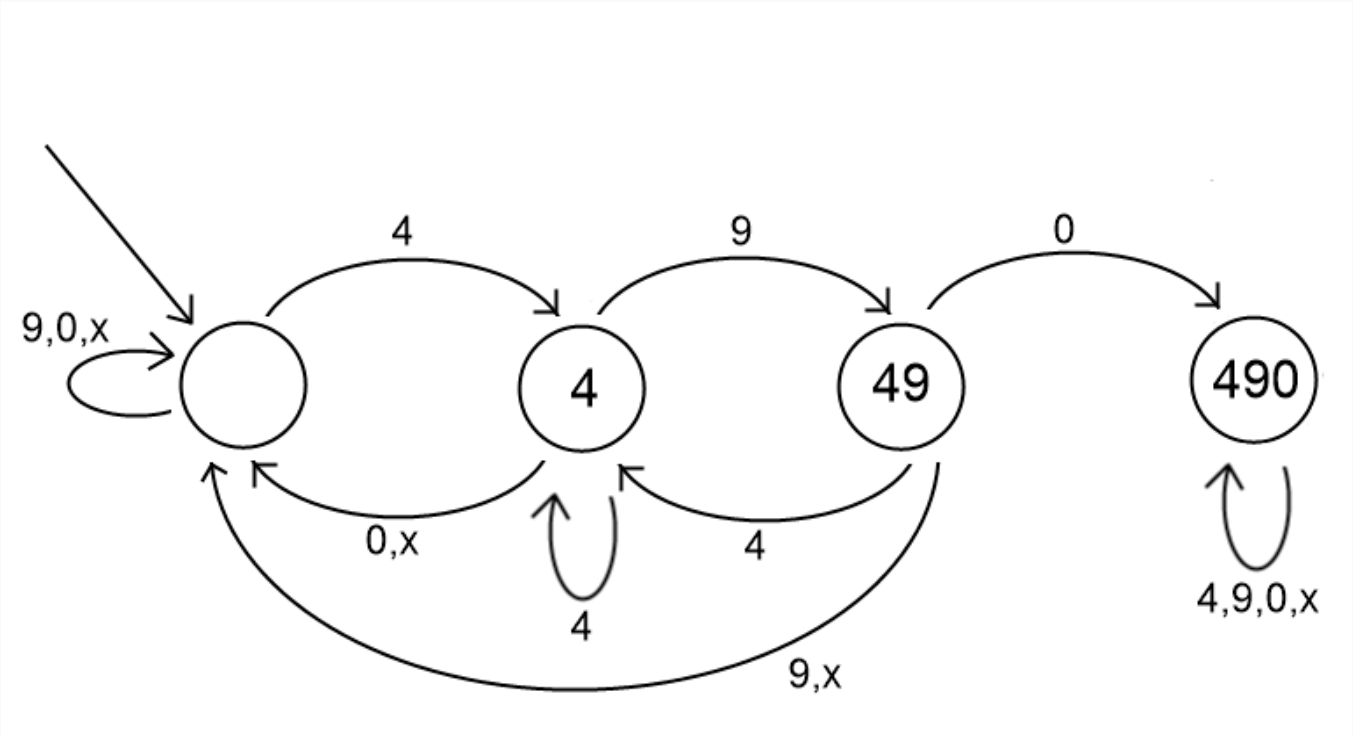
and we see character c, then we can try appending c to the prefix of length k, and find out the new longest prefix that still matches.

In fact we can pre-compute all of this and store it in a table:

$nxt[k][c] =$ length of maxi prefix match of the suffix after we append c, if the length of max prefix match of the suffix was k before.

Problem 1 – Solution (continued)

This “next match” table can be better visualized as a DFA:



Problem 1 – Solution (continued)

We get the recurrence from the DFA and the routine LCS DP:

$$f(i,j,0) = \max \begin{cases} f(i-1,j,0), f(i,j-1,0) & \\ f(i-1,j-1,0)+1 & \text{if } a[i] = b[j] \neq \text{`4'} \\ f(i-1,j-1,1)+1 & \text{if } a[i] = b[j] \neq \text{`4' or `9'} \\ f(i-1,j-1,2)+1 & \text{if } a[i] = b[j] \neq \text{`4' or `0'} \end{cases}$$

$$f(i,j,1) = \max \begin{cases} f(i-1,j,1), f(i,j-1,1) & \\ f(i-1,j-1,k)+1 & \text{if } a[i] = b[j] = \text{`4'} \end{cases}$$

$$f(i,j,2) = \max \begin{cases} f(i-1,j,2), f(i,j-1,2) & \\ f(i-1,j-1,1)+1 & \text{if } a[i] = b[j] = \text{`9'} \end{cases}$$

This idea of "find next match given prev. match" is the heart of KMP.

Answer: $\max(f(m,n,0), f(m,n,1), f(m,n,2))$
Time complexity: $O(mn)$

Problem 2

Given a convex polygon with n vertices in counter-clockwise order.

A triangulation is formed by drawing diagonals between non-adjacent vertices such that they do not intersect. The cost of a triangulation is sum of the perimeters of the component triangles.

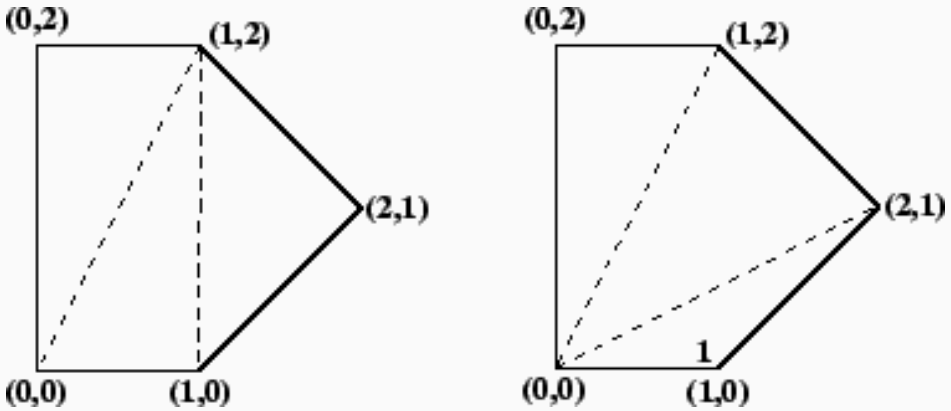
Find the minimum cost triangulation of the given convex polygon.

Problem 2

Given a convex polygon with n vertices in counter-clockwise order.

A triangulation is formed by drawing diagonals between non-adjacent vertices such that they do not intersect. The cost of a triangulation is sum of the perimeters of the component triangles.

Find the minimum cost triangulation of the given convex polygon.



Credit: *Geeks for Geeks*

The polygon on the left has cost $8 + 2\sqrt{2} + 2\sqrt{5} \approx 15.30$ while the one on the right has cost $4 + 2\sqrt{2} + 4\sqrt{5} \approx 15.77$.

Problem 2 – Solution

DP state: $f(i, j)$ = minimum cost of a triangulation of vertices i to j.

Problem 2 – Solution

DP state: $f(i, j)$ = minimum cost of a triangulation of vertices i to j.

Base case: $f(i, j) = dist(i, i+1) + \cdots + dist(j-1, j) + dist(j, i)$ if $j \leq i+2$.

Problem 2 – Solution

DP state: $f(i, j)$ = minimum cost of a triangulation of vertices i to j.

Base case: $f(i, j) = dist(i, i+1) + \cdots + dist(j-1, j) + dist(j, i)$ if $j \leq i+2$.

To compute $f(i, j)$ we consider making a triangle from points i, k and j where $i < k < j$ and this leaves us with 2 smaller polygons.

Problem 2 – Solution

DP state: $f(i, j)$ = minimum cost of a triangulation of vertices i to j.

Base case: $f(i, j) = dist(i, i+1) + \cdots + dist(j-1, j) + dist(j, i)$ if $j \leq i+2$.

To compute $f(i, j)$ we consider making a triangle from points i, k and j where $i < k < j$ and this leaves us with 2 smaller polygons.

Recurrence relation:

$$f(i, j) = \min_{i<k<j} \{f(i, k) + f(k, j) + dist(i, k) + dist(k, j) + dist(j, i)\}$$

Answer: $f(1, n)$
Time complexity: $O(n^3)$

Problem 3

Consider a game of rock-paper-scissors. There are r people who always play rock, s people who always play scissors and p people who always play paper.

In each round, 2 random persons play each other, with all pairs being equally likely and the loser gets knocked out.

What is the probability that only rock players survive in the long run? What about paper players? Scissor players?

Problem 3 – Solution

DP state: $f(i, j, k)$ = probability of i people who play rock, j who play scissors and k who play paper of being in the game.

Problem 3 – Solution

DP state: $f(i, j, k)$ = probability of i people who play rock, j who play scissors and k who play paper of being in the game.

Base case: $f(r, s, p) = 1$.

Problem 3 – Solution

DP state: $f(i, j, k)$ = probability of i people who play rock, j who play scissors and k who play paper of being in the game.

Base case: $f(r, s, p) = 1$.

Let's consider the probability of all the rock players surviving since the other 2 cases are similar. To compute $f(i, j, k)$, we consider the probability of going from e.g. $(i, j + 1, k)$ to (i, j, k) (rock kills scissor).

Problem 3 – Solution

DP state: $f(i, j, k)$ = probability of i people who play rock, j who play scissors and k who play paper of being in the game.

Base case: $f(r, s, p) = 1$.

Let's consider the probability of all the rock players surviving since the other 2 cases are similar. To compute $f(i, j, k)$, we consider the probability of going from e.g. $(i, j+1, k)$ to (i, j, k) (rock kills scissor).

Recurrence relation:
$f(i, j, k) = \frac{ij}{ij+jk+ki} \cdot f(i, j+1, k) + \frac{jk}{ij+jk+ki} \cdot f(i, j, k+1) + \frac{ki}{ij+jk+ki} \cdot f(i+1, j, k)$.

Probability of all rock players surviving = $\sum_{i=1}^{r} f(i, 0, 0)$.

Time Complexity: $O(rsp)$.

Next Class

Advanced Dynamic Programming Techniques!