Dynamic Programming Lecture 3

Wednesday, 28 August 2024 6:06 AM

Edit Distance Problem

The edit distance between two strings is the minimum number of operations required to transform one string into another. The allowed operations are insertion of a character, deletion of a character or substitution of a character with another character.

eg: S1 : "Kitten" S2 : "sitting"

kitten $\xrightarrow{\text{kas}}$ sitten $\xrightarrow{\text{emi}}$ sittin $\xrightarrow{\text{insert g}}$ sitting

$$\text{edit\_dist}(n,m) = \begin{cases} \max(n,m), & \text{if } n=0 \text{ or } m=0 \\ \text{edit\_dist}(n-1,m-1), & \text{if } s1[n-1] = s2[m-1] \\ 1 + \min(\text{edit\_dist}(n-1,m), \\ \quad \text{edit\_dist}(n,m-1), \\ \quad \text{edit\_dist}(n-1,m-1)), & \text{if } s1[n-1] \neq s2[m-1] \end{cases}$$

```
function edit_dist (s1, s2):
    n = length(s1), m = length(s2)
    dp = matrix of size (n+1) x (m+1)
    for (i : 0 → n):
        dp[i][0] = i
    for (i : 0 → m):
        dp[0][i] = i
    for (i : 1 → n):
        for (j : 1 → m):
            if s1[i] == s2[j]:
                dp[i][j] = dp[i-1][j-1]
            else:
                dp[i][j] = 1 + min(dp[i-1][j-1], dp[i-1][j], dp[i][j-1])
    return dp[n][m]
```

$T = O(m \cdot n)$

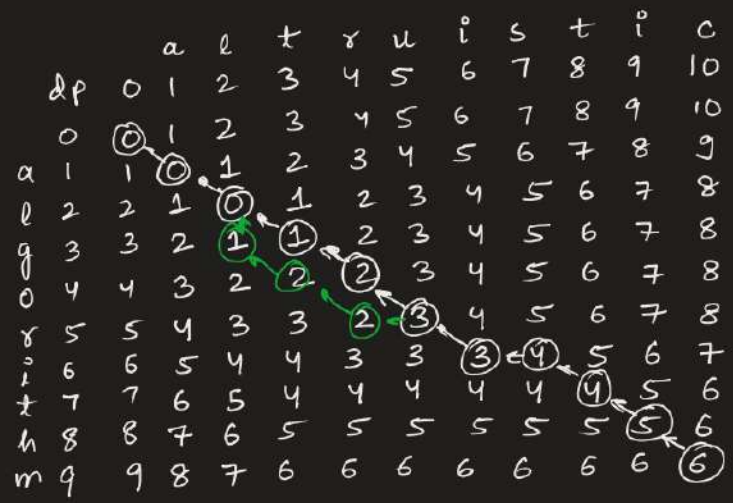
$S = O(m \cdot n)$

eg: s1 = kitten s2 = sitting

n = 6, m = 7

		-	s	i	t	t	i	n	g
	dp	0	1	2	3	4	5	6	7
-	0	0	1	2	3	4	5	6	7
k	1	1	1	2	3	4	5	6	7
i	2	2	2	1	2	3	4	5	6
t	3	3	3	2	1	2	3	4	5
t	4	4	4	3	2	1	2	3	4
e	5	5	5	4	3	2	2	3	4
n	6	6	6	5	4	3	3	2	3

Consider the two strings str1 = "algorithm" and str2 = "altruistic". What is the minimum edit distance between these two strings is ____



dp		a	l	t	r	u	i	s	t	i	c
	0	1	2	3	4	5	6	7	8	9	10
a	1	1	0	1	2	3	4	5	6	7	8
l	2	2	1	0	1	2	3	4	5	6	7	8
g	3	3	2	1	1	2	3	4	5	6	7	8
o	4	4	3	2	2	2	3	4	5	6	7	8
r	5	5	4	3	3	2	3	4	5	6	7	8
i	6	6	5	4	4	3	3	3	4	5	6	7
t	7	7	6	5	4	4	4	4	4	4	5	6
h	8	8	7	6	5	5	5	5	5	5	5	6
m	9	9	8	7	6	6	6	6	6	6	6	6

algorithm
↓ match
algorithC
↓ h→i
algoritic
↓ insert s after i
algoristic → insert u
↓ r↔u
algouistic
↕ o↔r
algruistic
↕ g↔t
altruistic

algoruistic
↕ o↔t
algtruistic
↓ delete g
altruistic

=x=

Longest Increasing Subsequence

↳ longest subsequence in which all the elements are in a strictly increasing order

eg: A = [10, 22, 9, 33, 21, 50, 41, 60, 80]

LIS: { [10, 22, 33, 50, 60, 80]
{ [10, 22, 33, 41, 60, 80]

```
function LIS (arr[], n):
    dp = array of size n, initialized to 1
    for (i: 1 → n-1):
        for (j: 0 → i-1):
            if (arr[j] < arr[i]):
                dp[i] = max(dp[i], dp[j]+1)

    return max(dp)
```

T = O(n²)
S = O(n)

The optimal approach for LIS is $O(n \log n)$ time taking uses binary search and dp.

—x—

Palindrome Partitioning

↳ partition a string into minimum number of substrings such that each substring is a palindrome

eg: s: "abacdfgfddaba"

a b a | c | d f g f d | d | a b a

i k j

4 partitions
5 substrings

$$minCuts(s, i, j) = \begin{cases} 0, & \text{if } s[i:j] \text{ is a palindrome} \\ 1 + \min(minCuts(s, i, k) + minCuts(s, k+1, j)) & \text{for } i < k < j \end{cases}$$

Palindrome checker:- (using dp)

```
function substringPalindromeCheck(s, n):
    P = matrix of size n x n
    for (i: 0 → n-1):                    } length 1 substrings
        P[i][i] = TRUE
    for (i: 0 → n-2):                    } length 2 substrings
        P[i][i+1] = (s[i] == s[i+1])
    for (length: 3 → n):                 } length 3 to n.
        for (i: 0 → n-length):
            j = i + length - 1
            P[i][j] = (s[i] == s[j] && P[i+1][j-1])
    return P
```

a b c d e d g a b a
0 1 2 3 4 5 6 7 8 9

0 → 10-2

0 → 8

i: i + length-1

l a a b c b

return P

eg: s = "aabcb"

P	i / j	a (0)	a (1)	b (2)	c (3)	b (4)
a	0	T	T	F	F	F
a	1	-	T	F	F	F
b	2	-	-	T	F	T
c	3	-	-	-	T	F
b	4	-	-	-	-	T

minCuts(s, 0, i)

$$C[i] = \begin{cases} 0, & \text{if } P[0][i] \text{ is true} \\ 1 + \min(C[j]) & \forall\, 0 \le j < i \text{ and } P[j+1][i] \text{ is true} \end{cases}$$

```
function minPalindromeCuts(s, n):
    P = substringPalindromeCheck(s, n)
    dp = array of size n

    for (i: 0 → n-1)
        if P[0][i]:
            dp[i] = 0
        else:
            dp[i] = ∞
            for (j: 0 → i-1)
                if P[j+1][i]:
                    dp[i] = min(dp[i], dp[j]+1)

    return dp[n-1]
```

dp[i] - minCuts(s, 0, i) i = 12

i	0	1	2	3	4	5	6	7	8	9	10	11	12
	a	b	a	c	d	f	g	f	d	d	a	b	a
dp	0	0	1	2	(1)								

$T = O(n^2)$, $S = O(n^2)$

Consider the following string s = "abcab" and the sequence arr = [3, 10, 2, 1, 20].

~~A.~~ The minimum number of cuts required to partition the string s into palindromic substrings is 2.
~~B.~~ The Longest Common Subsequence (LCS) between s and the string "abcb" is "abc".
C. The length of the Longest Increasing Subsequence (LIS) in the array arr is 3.
~~D.~~ The minimum palindrome partitioning of the string s results in the substrings "a", "bcb", and "a".

a|b|c|a|b cuts → {a, b, c, a, b}

abcab abcb → LCS abcb

3, 10, 2, 1, 20

{3, 10, 20}