

Algorithm CS 610 Assignment 3

Naren Kumar S (25310044), Dikshit Hegde (25310017)

15 September 2025

Contents

1 Problem Set 6, Question 09	1
1.1 Description	1
1.2 Algorithm	2
1.3 Proof of Correctness	2
1.4 Time Complexity	3
2 Problem Set 6, Question 12	4
2.1 Description	4
2.2 Algorithm	4
2.3 Proof of Correctness	4
2.4 Time Complexity	5
3 Problem Set 6, Question 14	6
3.1 Description	6
3.2 Algorithm	6
3.3 Proof of Correctness	6
3.4 Time Complexity	7

1 Problem Set 6, Question 09

We need to find longest common subsequence between two given sequence A and B .

1.1 Description

Lets consider two strings A and B with length n and m . We first create a 2D dp array which stores the maximum matched elements till i and j . With the following properties, if $A[i] == B[j]$ then $dp[i][j]$ is one greater than $dp[i-1][j-1]$ stating that the previous element was also a match, else we will carry the previous element value where number of matched are greater either in $dp[i-1][j]$ or $dp[i][j-1]$ as we need to find the longest common subsequence.

1.2 Algorithm

Algorithm 1 LCS(A, B)

```
1:  $n \leftarrow \text{length of } A$ 
2:  $m \leftarrow \text{length of } B$ 
3: if  $n == 0 || m == 0$  then
4:   return Null
5: end if
6:  $dp[0 \dots n][0 \dots m] \leftarrow 0$ 
7: for  $i = 1$  to  $n$  do
8:   for  $j = 1$  to  $m$  do
9:     if  $A[i] == B[j]$  then
10:       $dp[i][j] = 1 + dp[i-1][j-1]$ 
11:    else
12:       $dp[i][j] = \text{MAX}(dp[i-1][j], dp[i][j-1])$ 
13:    end if
14:   end for
15: end for
16:  $i = n$ 
17:  $j = m$ 
18: while  $i > 0 \& j > 0$  do
19:   if  $A[i] == B[j]$  then
20:      $\text{STACK.push}(A[i])$ 
21:      $i = i - 1$ 
22:      $j = j - 1$ 
23:   else if  $dp[i][j-1] > dp[i-1][j]$  then
24:      $j = j - 1$ 
25:   else
26:      $i = i - 1$ 
27:   end if
28: end while
29: return  $\text{STACK.pop}()$ 
```

1.3 Proof of Correctness

Lemma: $dp[i][j]$ outputs the length of the maximum consecutive matches we can achieve by deleting the unmatched elements till i^{th} element of sequence A and j^{th} element of sequence B.

Prove by induction

Base Case: If either of the input sequence length is zero then there is no common subsequence between the two sequence, i.e. $dp[0][0] = 0$

By induction hypothesis: Assume that the algorithm works correctly for the i^{th} row and $j-1$ columns of dp sequence of A and B.

Induction Step: For a given sequence of length n and m :

- **Case 1**

if $A[n]$ is equal to $B[m]$,

then the $opt[n][m] = opt[n-1][m-1] + 1$ and

by induction hypothesis $opt[n-1][m-1] = dp[n-1][m-1]$, this implies $opt[n][m] = dp[n-1][m-1] + 1$.

Through our algorithm, we can see that $dp[n][m] = dp[n-1][m-1] + 1$.

Therefore $opt[n][m] = dp[n][m]$ when the last elements of both the sequence match.

- **Case 2:**

if $A[n]$ is not equal to $B[m]$,

then the $opt[n][m] = \max(opt[n][m-1], opt[n-1][m])$, by considering the maximum matched subsequence between A and B .

By induction hypothesis $opt[n][m-1] = dp[n][m-1]$ and $opt[n-1][m] = dp[n-1][m]$,

this implies $opt[n][m] = \max(dp[n][m-1], dp[n-1][m])$.

Through our algorithm, we can see that $dp[n][m] = \max(dp[n][m-1], dp[n-1][m])$.

Therefore $opt[n][m] = dp[n][m]$ when the last elements of both the sequence match.

After getting the maximum length of the common subsequence, we backtrack using the dp matrix where there is match between the elements we push the element in stack, if there is mismatch then we move to the index where there is maximum match for the previous elements.

1.4 Time Complexity

As this algorithm runs over two for loops over n and m , resulting in $O(mn)$, and while loop runs for $n + m$ times. But overall $O(mn)$ is greater than $O(m + n)$. Overall the running time of the complete algorithm is $O(mn)$.

2 Problem Set 6, Question 12

We are given arrays $L[1, \dots, n]$ and $K[1, \dots, n]$ representing the number of lollipops we can get on day i and the subsequent cool down period respectively. We need to determine the maximum number of lollipops we can collect over n days.

2.1 Description

For a given n days we can get $L[1, \dots, n]$ lollipops with a cool down period $K[1, \dots, n]$. We decide on i^{th} day to collect the lollipops or skip and go to the next $(i + 1)^{th}$ day to collect lollipops. Here $dp[i]$ represents the maximum number of lollipops collected from i^{th} day to n^{th} day. i^{th} day is considered for a skip if the lollipops collected from next day $(i + 1)^{th}$ till last is maximum when compared to the lollipops collected from i^{th} day, else i^{th} day is considered.

2.2 Algorithm

Algorithm 2 MaxLollipop(L, K)

```
1:  $n \leftarrow \text{length of } L$ 
2: if  $n == 0$  then
3:   return 0
4: end if
5:  $dp[1 \dots n + 1] \leftarrow 0$ 
6: for  $i = n$  to 1 do
7:    $nextDay = i + K[i] + 1$ 
8:   if  $nextDay < n + 1$  then
9:      $A = L[i] + dp[nextDay]$ 
10:  else
11:     $A = L[i]$ 
12:  end if
13:   $B = dp[i + 1]$ 
14:   $dp[i] = \max(A, B)$ 
15: end for
16: return  $dp[1]$ 
```

2.3 Proof of Correctness

Lemma: $dp[i]$ be the maximum number of lollipops can be collected from i^{th} to n^{th} day. Then $dp[i] = \max(L[i] + dp[i + K[i] + 1], dp[i + 1])$

Prove by Induction

Base Case:

At day $n+1$, $dp[n+1] = 0$ as we exceed the number of days.

Induction Hypothesis:

Assume that the algorithm works correctly for $i \in (n \rightarrow 2)$.

Induction step:

On day 1 we have two choices :

- we can skip
 $B = dp[i+1]$
- If it is not skipped
 $A = L[i] + dp[i+K[i]+1]$

for day 1, we get $dp[1] = \max(A, B)$

The algorithm correctly computes $dp[1]$, the maximum collectable lollipop starting from day 1.

2.4 Time Complexity

As this algorithm runs over two for loops over n , resulting in $O(n)$.

3 Problem Set 6, Question 14

For a given input sequence, splitting the sequence into valid subsequence.

3.1 Description

Let S be a given input sequence, let's assume $IsPattern(a, b, S)$ returns *True* if the subsequence from a to b in S is a valid sequence, else it returns *False*, where $a < b$. Let $dp[i][j]$ stores the validity of the sequence from j to i in S . Here j iterates from 1 to $i - 1$ indicating the starting index of the valid sequence, and i represents the ending index of the valid sequence which is iterated from 1 to n .

3.2 Algorithm

Algorithm 3 GetPattern(S)

```
1:  $n \leftarrow \text{length of } S$ 
2: if  $n == 0$  then
3:   return True
4: end if
5:  $dp[0 \dots n] \leftarrow \text{False}$ 
6:  $dp[0] = \text{True}$ 
7: for  $i = 1$  to  $n$  do
8:   for  $j = 0$  to  $i - 1$  do
9:     if  $dp[j] \wedge IsPattern(j + 1, i, S)$  then
10:       $dp[i] = \text{True}$ 
11:      break
12:    else
13:       $dp[i] = \text{False}$ 
14:    end if
15:  end for
16: end for
17: return
```

3.3 Proof of Correctness

Lemma: Let $dp[i] = \text{True}$ and $dp[i + k] = \text{True}$, then there exist a valid sequence of length k from $i + 1$ to $i + k$.

Proof by induction

Base Case:

If the input sequence is a null sequence, then it's a valid sequence. Therefore $dp[0] = \text{True}$

Induction Hypothesis:

Assume that the algorithm works correctly for $i \in (1 \rightarrow n - 1)$.

Induction step:

At $i = n$, we find a particular j such that the subsequence from j till n results a valid subsequence. If there exists a j then $dp[i] = True$ else it set to *False*.

3.4 Time Complexity

As this algorithm runs over two for loops over n , resulting in $O(n^2)$.