Algorithm CS 610 Assignment 2

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

15 September 2025

Contents

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 subsquence.

1.2 Algorithm

Algorithm 1 LCS(A, B)

```
n ← length of A
m ← length of B
if n == 0||m == 0 then
    return Null
end if
dp[0...n][0...m] ← 0
for i = 1 to n do
    for j = 1 to m do
        if A[i] == B[j] then
            dp[i][j] = 1 + dp[i − 1][j − 1]
        else
            dp[i][j] = MAX(dp[i − 1][j], dp[i][j − 1])
        end if
    end for
end for
i = n
j = m
while i > 0&j > 0 do
    if A[i] == B[j] then
        STACK.push(A[i])
        i = i − 1
        j = j − 1
    else if dp[i][j − 1] > dp[i − 1][j] then
        j = j − 1
    else
        i = i − 1
    end if
end while
return 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 $B[n]$,
  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 $L[1, \ldots n]$ and $K[1, \ldots n]$ 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 $L[1, \ldots n]$ lollipops with a cool down period $K[1, \ldots n]$. 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)

```
n ← length of L
if n == 0 then
    return 0
end if
dp[1 ... n+1] ← 0
for i = n to 1 do
    nextDay = i + K[i] + 1
    if nextDay < n + 1 then
        A = L[i] + dp[nextDay]
    else
        A = L[i]
    end if
    B = dp[i + 1]
    dp[i] = max(A, B)
end for
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 $i \in (n \to 2)$.

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, lets assume $IsPattern(a, b, S)$ returns $True$ if the subsqeunce from a to b in S is a valid squence, 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)

```
n ← length of S
if n == 0 then
    return True
end if
dp[0...n] ← False
dp[0] = True
for i = 1 to n do
    for j = 0 to i − 1 do
        if dp[j] ∧ IsPattern(j + 1, i, S) then
            dp[i] = True
            break
        else
            dp[i] = False
        end if
    end for
end for
return
```

3.3 Proof of Correctness

Lemma: Let $dp[i] = True$ and $dp[i + k]) = True$, then there exisit a valid sequence of lenght k from $i + 1$ to $i + k$.

Proof by induction

Base Case:
If the input sequence is a null sequence, then its a valid sequence. Therefore $dp[0] = True$

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 $O(n^2)$.