Report of an space efficient algorithmn for solving the $LCSk$ problem

陈雨瑶,李欣然,陈伊葶

Abstract

Abstract : The LCS problem is the problem of finding the largest sequence common to all sequences in a set (usually only two sequences). It has many applications such as pattern matching, file comparison and so on. The article we reseached proposed two space efficient algorithms for $LCSk$ and $LCS_{\geq}k$ problems based on algorithms of Benson et al[1], of $O(kn)$ space complexity, if the size of given sequences are both n. We only present the first algorithm (for $LCSk$) in this report, since the two algorithms are very similar.

Keywords: *LCSk; dynamic programming; divide and conquer*

1. Introduction

The longest common subsequence(LCS) problem is a classical problem in computer science. The longest common subsequence problem is a classical computer science problem, which is the basis of data comparison programs such as the diff utility, and has applications in computational linguistics and bioinformatics. It is also widely used by version-control systems such as Git to coordinate multiple changes to a collection of version-controlled files.Given two sequence A and B, our goal is to find the longest common subsequence in A and B among all the common subsequences. In our researched paper, the authors proposed two space efficient algorithms to solve the $LCSk$ and $LCS_{\geq}k$, which are two varients of the original LCS problem. The two algorithms are based on algorithms of Benson et al.[1].

The $LCSk$ problem is defined as follows.

Definition 1 *Given two sequences $A = a_1a_2\cdots a_n$ and $B = b_1b_2\cdots b_m$, and an integer k, the LCSk problem is to find the maximal length l such that there are l substrings, $a_{i_1}\cdots a_{i_1+k-1},\cdots,a_{i_l}\cdots a_{i_l+k-1}$, identical to $b_{j_1}\cdots b_{j_1+k-1},\cdots,b_{j_l}\cdots b_{j_l+k-1}$ where $\{a_{i_t}\}$ and $\{b_{j_t}\}$ are in increasing order for $1\leq t\leq l$ and any two k-length substrings in the same sequence, do not overlap.*

The $LCS \geq k$ problem is defined as follows.

Definition 2 *Given two sequences $A = a_1a_2\cdots a_n$ and $B = b_1b_2\cdots b_m$, and an integer k, the $LCS \geq k$ problem is to find substrings with maximal total length such that $a_{i_p}\cdots a_{i_p+k+t}$ is identical to $b_{j_p}\cdots b_{j_p+k+t}$ for $-1\leq t\leq k-2$ where $\{a_{i_t}\}$ and $\{b_{j_t}\}$ are in increasing order for $1\leq t\leq l$ and any two substrings in the same sequence, do not overlap.*

In the paper, the authors focus on the space efficient algothms to solve the $LCSk$ and $LCS \geq k$ problem. The first algorithm is for $LCSk$ problem, which is a dynamic programming algorithm using $O(mn)$ time and $O(kn)$ space. The second algorithm is for $LCS \geq k$ problem, using $O(mn)$ time and $O(kn)$ space. Since the latter is similar to the former, we only discussed the first algorithm in our report.

2. The algorithm for $LCSk$ problem

2.1. Description

Define $d(i,j)$ as the length of the longest match between the prefixes of $A[1:i] = a_1a_2\cdots a_i$ and $B[i:j] = b_1b_2\cdots b_j$. The update equation of $d(i,j)$ is Eq. 1.

$$d(i,j) = \begin{cases} 1 + d(i-1, j-1), & \text{if } a_i = b_j \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

Define $f(i,j)$ as the number of k matchings in the longest common subsequence, consisting of k matching in the prefixes $A[i:i]$ and $B[i:j]$. The update equation of $f(i,j)$ is Eq. 2.

$$f(i,j) = \max \begin{cases} f(i-1,j) \\ f(i,j-1) \\ f(i-k,j-k) + \delta(d(i,j)) \end{cases} \tag{2}$$

Where, $\delta(d(i,j))$ is defined by:

$$\delta(i) = \begin{cases} 1, & \text{if } i \geq k \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

So we have a standarded dynamic programming algorithm using $O(mn)$ time and $O(mn)$ space:

Algorithm 1: $LCSk$

```
    Input: input parameters A, B
    Output: $f(i,j)$
1 for $i = 1$ to $n$ do
2     for $j = 1$ to $m$ do
3         if $a_i = b_j$ then $d(i,j) \leftarrow 1 + d(i-1,j-1)$;
4         $f(i,j) \leftarrow \max\{f(i-1,j), f(i,j-1), f(i-1,j-1) + \delta(d(i,j))\}$ ;
5     end
6 end
7 return $f(n,m)$
```

The authors noticed that when computing a particular row of the denamic programming table, we only need the nearest previous k rows. Thus we only need to maintain $k+1$ rows in the memory at a time. We assume $m = O(n)$ so only $(k+1)m = O(kn)$ entries are needed to compute the table.

But if we need to construct the longest common subsequence itself, not only the length, the above algorithm is not enough. By extending the definition of $LCSk$ problem, the authors give a method using divide-and-conquer.The generalized definition of $LCSk$ as follows:

Definition 3 *For the two substrings* $A[i_0 : i_1] = a_{i_0}a_{i_0+1}\cdots a_{i_1}$ *and* $B[j_0 : j_1] = b_{j_0}b_{j_0+1}\cdots b_{j_1}$ *,* $1 \leq i0 \leq i_1 \leq n$*,* $1 \leq j_0 \leq j_1 \leq m$*, the set of all LCSk of* $A[i_0 : i_1]$ *and* $B[j_0 : j_1]$ *is denoted by* $LCSk(i_0, i_1, j_0, j_1)$*. The length of an LCSk in* $LCSk(i_0, i_1, j_0, j_1)$ *is denoted by* $p(i_0, i_1, j_0, j_1)$*.*

Similarly, the set of all LCSks of the two reversed substrings $A[i_0 : i_1] = a_{i_1}a_{i_1-1}\cdots a_{i_0}$ *and* $B[j_0 : j_1] = b_{j_1}b_{j_1-1}\cdots b_{j_0}$ *is denoted by* $LCSkR(i_0, i_1, j_0, j_1)$*. The length of an LCSk in* $LCSkR(i_0, i_1, j_0, j_1)$ *is denoted by* $g(i_0, i_1, j_0, j_1)$*.*

When $i_0 = 1$ and $j_0 = 1$, $f(1, i, 1, j) = f(i, j)$ for $1 \leq i \leq n, 1 \leq j \leq m$. Similarly, when $i_1 = n$ and $j_1 = m$, $g(i, n, j, m) = g(1, 1)$ for $1 \leq i \leq n, 1 \leq j \leq m$. And obviously, when $i_0 = 1, j_0 = 1, i_1 = n$ and $j_1 = m$, $f(n, m) = g(1, 1)$.

The following algorithm $\xi(i_0, i_1, j_0, j_1)$ returns an array $L_1[0 : k][1 : m]$ storing the $(k + 1)m$ entries required to compute the table. Row i is mapped to row$\lambda(i)(1 \leq \lambda(i) \leq k)$ of L_1, and $\lambda(i)$ is defined as:

$$\lambda(i) = (i - 1) \bmod k + 1 \tag{4}$$

Algorithm 2: $\xi(i_0, i_1, j_0, j_1)$

```
   Input: $A[i_0 : i_1], B[j_0 : j_1]$
   Output: $L_1$
 1 for $i = i_0$ to $i_1$ do
 2     for $j = j_0$ to $j_1$ do
 3         $L_1(\lambda(i-1), j) \leftarrow L_1(0, j)))$ ;
 4         if $a_i = b_j$ then $d(\lambda(i), j) \leftarrow 1 + d(\lambda(i-1), j-1)$ ;
 5         $L_1(0, j) \leftarrow \max\{L_1(\lambda(i-1), j), L_1(0, j-1), L_1(\lambda(i-k), j-k) + \delta(d(\lambda(i), j))\}$
 6     end
 7 end
 8 for $j = j_0$ to $j_1$ do $L_1(\lambda(i_1), j) \leftarrow L_1(0, j)$ ;
 9 return $L_1(\lambda(i_1), j_1)$
```

Similarly, $\eta(i_0, i_1, j_0, j_1)$ maintains a L_2 array for $LCSkR$ problem:

Algorithm 3: $\eta(i_0, i_1, j_0, j_1)$

```
    Input: $A[i_0 : i_1]$, $B[j_0 : j_1]$
    Output: $L_2$
 1  for $i = i_1$ to $i_0$ do
 2      for $j = j_1$ to $j_0$ do
 3          $L_2(\lambda(i+1), j) \leftarrow L_2(0, j)))$ ;
 4          if $a_i = b_j$ then
 5              $d(\lambda(i), j) \leftarrow 1 + d(\lambda(i+1), j+1)$ ;
 6          end
 7          $L_2(0, j) \leftarrow$
              $\max\{L_2(\lambda(i+1), j), L_2(0, j+1), L_2(\lambda(i+k), j+k) + \delta(d(\lambda(i), j))\}$ ;
 8      end
 9  end
10  for $j = j_0$ to $j_1$ do
11      $L_2(\lambda(i_0), j) \leftarrow L_2(0, j)$ ;
12  end
13  return $L_2(\lambda(i_0), j_0)$
```

And a *split*$(k_1, k_2, l_1, l_2, s_1, s_2, i_1, j_0, j_1)$ function to find the breaking point k_1, k_2 and common subsequence beginning point s_1, s_2:

Algorithm 4: *split*$(k_1, k_2, l_1, l_2, s_1, s_2, i_1, j_0, j_1)$

```
Input: input parameters $i_1, j_0, j_1$
Output: $k_1, k_2, l_1, l_2, s_1, s_2$
1  $s_1, s_2, tmp \leftarrow 0$ ;
2  for $i = i_1 - k + 1$ to $i_1$ do
3      for $j = j_0 - 1$ to $j_1$ do
4          $t \leftarrow L_1(\lambda(i), j) + L_2(\lambda(i+1), j+1)$ ;
5          if $t > tmp$ then $tmp \leftarrow t, k_1 \leftarrow i, k_2 \leftarrow j$ ;
6      end
7  end
8  $l_1 \leftarrow L_1(\lambda(k_1, k_2))$ ;
9  $l_2 \leftarrow L_2(\lambda(k_1 + 1, k_2 + 1))$ ;
10 if $l_1 = 1$ then $s_1 \leftarrow \min_{j_0 \le j \le j_1} \{j \mid L_1(\lambda(k_1), j) = 1\} - k + 1$ ;
11 if $l_2 = 1$ then $s_2 \leftarrow \min_{j_0 \le j \le j_1} \{j \mid L_2(\lambda(k_1 + 1), j) = 1\}$ ;
12 return $k_1, k_2, l_1, l_2, s_1, s_2$
```

So the divide-and-conquer algorithm to solve the *LCSk* problem with constructing the subsequences is:

Algorithm 5: $D\&C(i_0, i_1, j_0, j_1)$

```
if $i_1 - i_0 + 1 < 2k$ then return;
$l \leftarrow \lfloor (i_1 - i_0 + 1 + k)/2$ ;
$L_1 = \xi(i_0, i_0 + l - 1, j_0, j_1)$ ;
$L_2 = \eta(i_0 + l - k + 1, i_1, j_0, j_1)$ ;
$split(k_1, k_2, l_1, l_2, s_1, s_2, s_0 + l - 1, j_0, j_1)$ ;
if $l_1 > 1$ then $D\&C(i_0, k_1, j_0, k_2)$ ;
else if $l_1 = 1$ then record $s_1$ ;
if $l_2 > 1$ then $D\&C(k_1 + 1, i_1, k_2 + 1, j_1)$ ;
else if $l_2 = 1$ then record $s_2$ ;
```

2.2. Correctness of the algorithm

Proof for applying $D\&C(1, n, 1, m)$ to sequences A and B, with size of n and m respectively, will produce an $LCSk$ of them:

When $n < 2k$ and any $m > 0$, l is always smaller or equal to 1.

If $l_1 = 1$ and $l_2 = 1$, suppose we have two $k-$strings $B[s_1 : s_1+k-1]$ and $B[s_2 : s_2+k-1]$, where

$$\begin{cases} s_1 = \min\limits_{1 \le j \le k_2} \{j \mid f(1, k_1, 1, j) = 1\} - k + 1 \\ s_2 = \min\limits_{k_2+1 \le j \le m} \{j \mid g(k_1 + 1, n, j, m) = 1\} \end{cases} \tag{5}$$

Because $l_1 = 1$ and $l_2 = 1$,it can be verified directly that $B[s_1 : s_1 + k - 1]$ and $B[s_2 : s_2 + k - 1]$ are in $LCSk(1, k_1, 1, k_2)$ and $LCSk(k_1 + 1, n, k_2 + 1, m)$.

If $l_1 = 0$ and $l_2 = 0$, $LCSk(1, k_1, 1, k_2) = LCSk(k_1+1, n, k_2+1, m) = \emptyset$, so the algorithm do nothing.

So the claim is true when $n < 2k$.

Then prove by induction. Suppose the claim is true when the size of A is smaller than n. Then show the claim is also true when the size of A is n: L_1 stores $LCSk(1, \lfloor(n+k)/2\rfloor, 1, m)$, computed by $\xi(1, \lfloor(n+k)/2\rfloor, 1, m)$ and L_2 stores $LCSkR(\lfloor(n+k)/2\rfloor+1, n, 1, m)$, computed by $\eta(\lfloor(n+k)/2\rfloor + 1, n, 1, m)$.

The split points k_1 and k_2 are:

$$\begin{cases} l_1 = f(1, k_1, 1, k_2) \\ l_2 = g(k_1 + 1, n, k_2 + 1, m) \\ \max\limits_{\substack{\lfloor(n+k)/2\rfloor - k+1 \le i \le \lfloor(n+k)/2\rfloor \\ 0 \le j \le m}} \{f(i, j) + g(i + 1, j + 1)\} = l_1 + l_2 \end{cases} \tag{6}$$

When $l_1 > 1$ and $l_2 > 1$, $Z = Z_1 \oplus Z_2$ is the common subsequence of A and B, where $Z_1 \in LCSk(1, k_1, 1, k_2)$ and $Z_2 \in LCSk(k_1 + 1, n, k_2 + 1, m)$.

So we can conclude $D\&C(1, n, 1, m)$ can produce an $LCSk$ of A and B.

2.3. Time and space analysis of the algorithm

Time complexity is $O(mn)$. Denote $T(n, m)$ as the time cost of the algorithm with respect to input size n and m. The recursive equation as follows:

$$T(n, m) = \begin{cases} T(k_1, k_2) + T(n - k_1, m - k_2) + O(mn), & \text{if } n \ge k \\ O(1), & \text{if } n < k \end{cases} \tag{7}$$

The claim is obviously true for $n < 2k$.

When $n \ge 2k$, $n/4 \le k_1 \le 3n/4$. Assume $T(n, m)$ is bounded by $c_1 \cdot mn$, and the $O(mn)$ term in 7 is bounded by $c_2 \cdot mn$. Then:

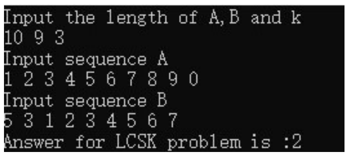
$$\begin{aligned} & T(k_1, k_2) + T(n - k_1, m - k_2) + O(mn) \\ \le & c_1 \cdot (k_1 k_2 + (n - k_1)(m - k_2)) + c_2 \cdot mn \\ = & c_1 \cdot 3mn/4 + c_2 \cdot mn \\ = & (3c_1/4 + c_2) \cdot mn \end{aligned}$$

Obviously $3c_1/4 + c_2 \le c_1$ is satifiable, so the assumption is true. So $T(n, m) = O(mn)$.

Assume A and B are in common storage using $O(m + n)$ space. Then the temporary space $D\&C$ use is only L_1 and L_2. Obviously $\mid L_1 \mid \le (k + 1)m, \mid L_2 \mid \le (k + 1)m$. Exclude the recursive calls to $D\&C$, there are at most $2n - 1$ calls to $D\&C$. So the space cost if propotional to $(k + 1)m$, i.e. $O(kn)$.

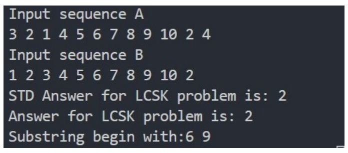
3. Program running environment and results

Our running environment is Windows11, but we believe it can also work well on OSX or Linux with a little modificaton. The STD Answer is the result from the standard DP, which we used to justify the divide and conquer algorithm. We tested some simple sequences, and it seems correct.



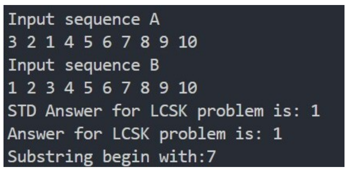
```
Input the length of A,B and k
10 9 3
Input sequence A
1 2 3 4 5 6 7 8 9 0
Input sequence B
5 3 1 2 3 4 5 6 7
Answer for LCSK problem is :2
```

(a) k=2



```
Input sequence A
3 2 1 4 5 6 7 8 9 10 2 4
Input sequence B
1 2 3 4 5 6 7 8 9 10 2
STD Answer for LCSK problem is: 2
Answer for LCSK problem is: 2
Substring begin with:6 9
```

(b) k=3



```
Input sequence A
3 2 1 4 5 6 7 8 9 10
Input sequence B
1 2 3 4 5 6 7 8 9 10
STD Answer for LCSK problem is: 1
Answer for LCSK problem is: 1
Substring begin with:7
```

(c) k=4

Figure 1. Running results for n>2k

4. Conclusion

We find it is similar to the sequence alignment algorithm learned in class, or it can be seen as a specific application of the algorithm. But their state transition equations of dynamic programming are different and the essence of this algorithm is the $split(\cdot)$ function. We modified the $\xi(\cdot)$ and $\eta(\cdot)$ funtion making L_1 and L_2 being global variable so that they do not need to be created and destroyed repeatedly. Our tests were carried out only for small k and small A, B. It is not clear that the large k, for example k = O(n), also has such performance.

5. Labor division

李欣然：Code

陈雨瑶，陈伊葶：Report

References

[1] G. Benson, A. Levy, D.Noifeld, B.R. Shalom, "LCSk: a refined similarity measure," *Theoret. Comput. Sci*, vol. 638, pp. 11–26, 2016. [Online].