Longest Square Subsequence Problem Revisited

Takafumi Inoue[1], Shunsuke Inenaga[1], Hideo Bannai[2]

[1]Kyushu University, [2]Tokyo Medical and Dental University

Background

Finding subsequences with specific properties:

Numeric sequences

- Longest increasing subsequence
  - $O(n \log n)$ time [Schensted 1961]
- Longest rollercoasters
  - $O(n)$ time [Gawrychowski et al. 2019]

Strings

- Longest palindromic subsequence
  - $O(n^2)$ time [Folklore]
- Longest square subsequence (LSS)
  - $O(n^2)$ time [Kosowski 2004]
  - $O(n^2(\log \log n)^2/\log^2 n)$ time [Tiskin 2013]

Subsequences

Definition (Subsequence)

A string is a subsequence of string *S*, if it can be obtained by deleting (possibly 0) characters from *S*.

Example

S = **S**tring**P**rocessingand**I**nformation**RE**trieval

SPIRE is a subsequence of *S*

Longest Square Subsequence (LSS) Problem

Longest Square Subsequence (LSS) Problem [Kosowski 2004]

Input : string S of length n.

Output : longest square subsequence of S, i.e., a subsequence of form xx, for some x.

Example

S = iwantedtobeinorlandoflorida

LSS(S) = andoorandoor

- $O(n^2)$ time [Kosowski 2004]
- $O(n^2(\log\log n)^2/\log^2 n)$ time [Tiskin 2013]
- No $O(n^{2-\varepsilon})$ time algorithm under SETH [Bringmann&Kunnemann 2015]

Our contributions

An algorithm for Longest Square Subsequence (LSS) problem that depends on different parameters.

$O(r \min \{n, M\} \log n/r + M \log n + n)$ time

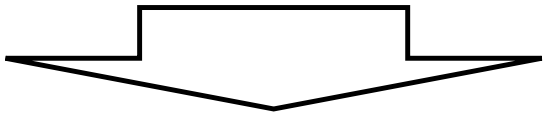
where:

r: length of LSS
M: # of matching points ($|\{ (i, j) \mid S[i] = S[j] \}|$) $\leq n^2$

Better than previous algorithms when:
$r = o(n(\text{loglog } n)^2/\log^3 n)$ and $M = o(n^2(\text{loglog } n)^2/\log^3 n)$

Overview

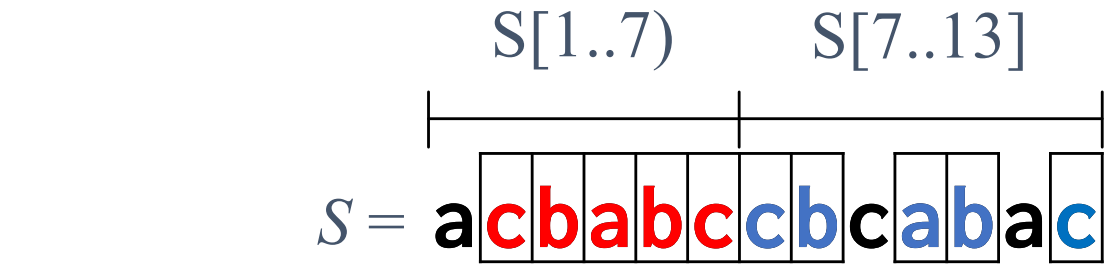
Longest Square Subsequence (LSS) problem



Dynamic Longest Common Subsequence (LCS) problem
with **append$_X$()** and **delete_front$_Y$()**

Dynamic Longest Increasing Subsequence (LIS) problem
with **append()** and **delete_min_all()**

LSS to Dynamic LCS



xx is a square subsequence of *S*
⇔ *x* is a common subsequence of *S*[1..*k*) and *S*[*k*..*n*] for some *k* = 2, ..., *n*

- $\mathrm{LSS}(S) = \max_{k=2,\ldots,n} 2 * \mathrm{LCS}(S[1..k), S[k..n])$

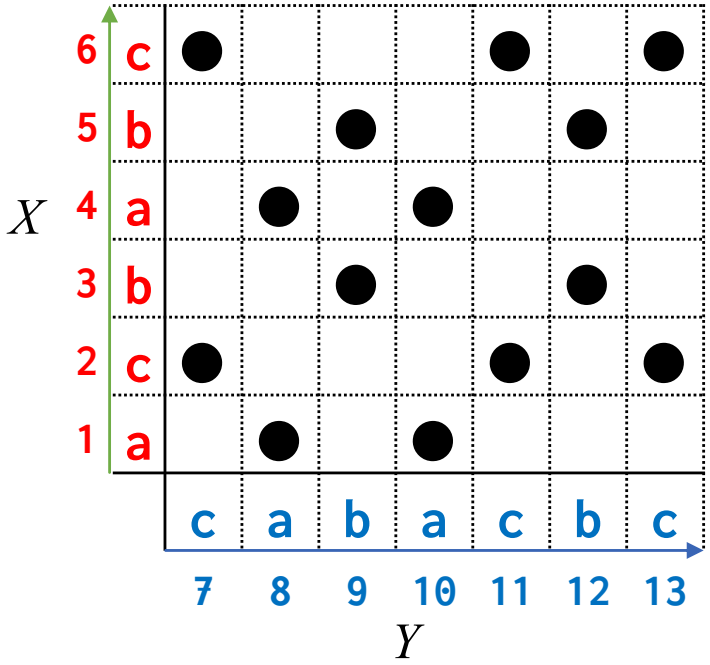
Naive computation of LCS for each *k* ➔ O(n^3) time

- **Dynamic LCS** with **append** and **`delete_front`**
  LCS(*S*[1..*k*), *S*[*k*..*n*]) to LCS(*S*[1..*k*+1), *S*[*k*+1..*n*])
  ➔ solution to LSS

LCS to LIS

Consider integer sequence T corresponding to partition $S=XY$

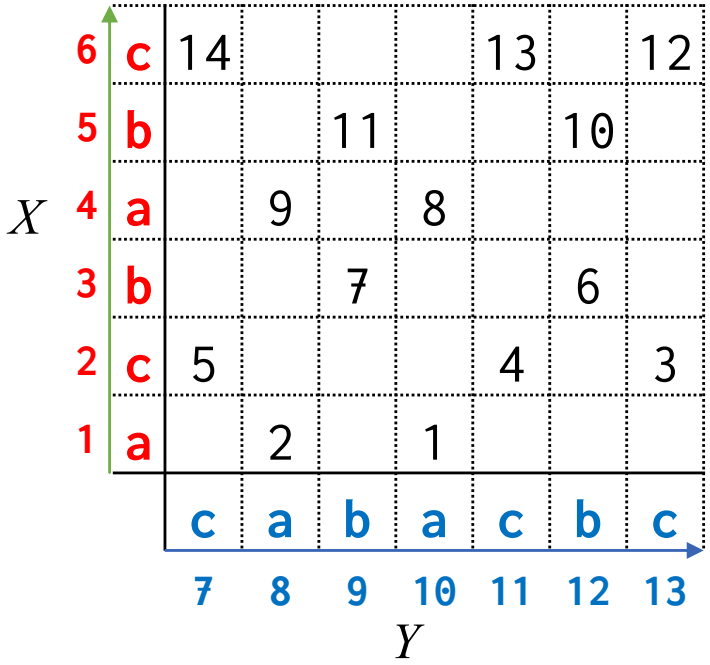
	X						Y						
	1	2	3	4	5	6	7	8	9	0	1	2	3
$S=$	a	c	b	a	b	c	c	a	b	a	c	b	c



LCS to LIS

Consider integer sequence T corresponding to partition $S=XY$

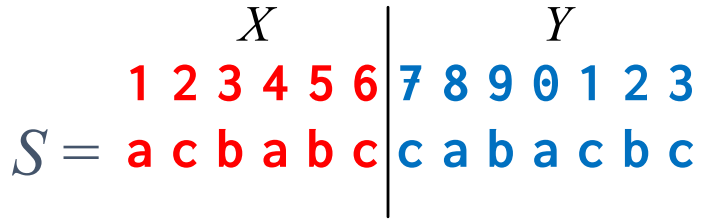
	X						Y						
	1	2	3	4	5	6	7	8	9	10	11	12	13
$S=$	a	c	b	a	b	c	c	a	b	a	c	b	c

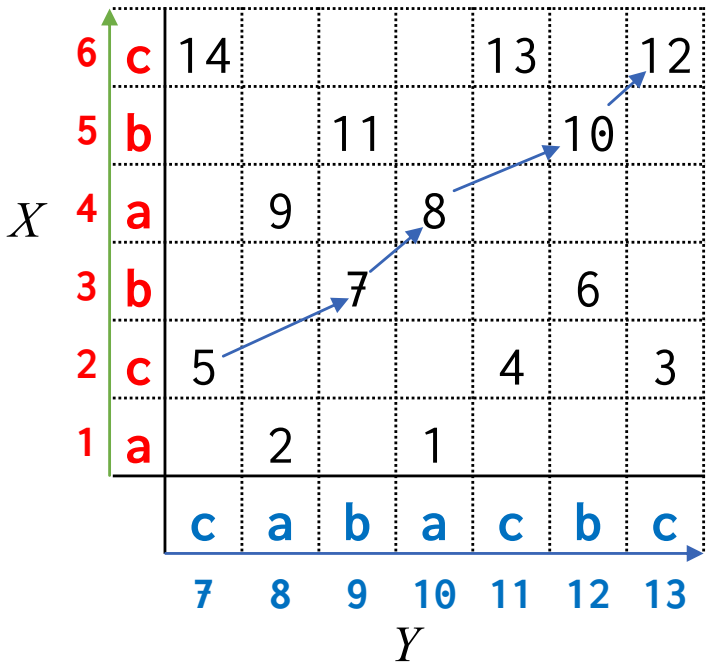


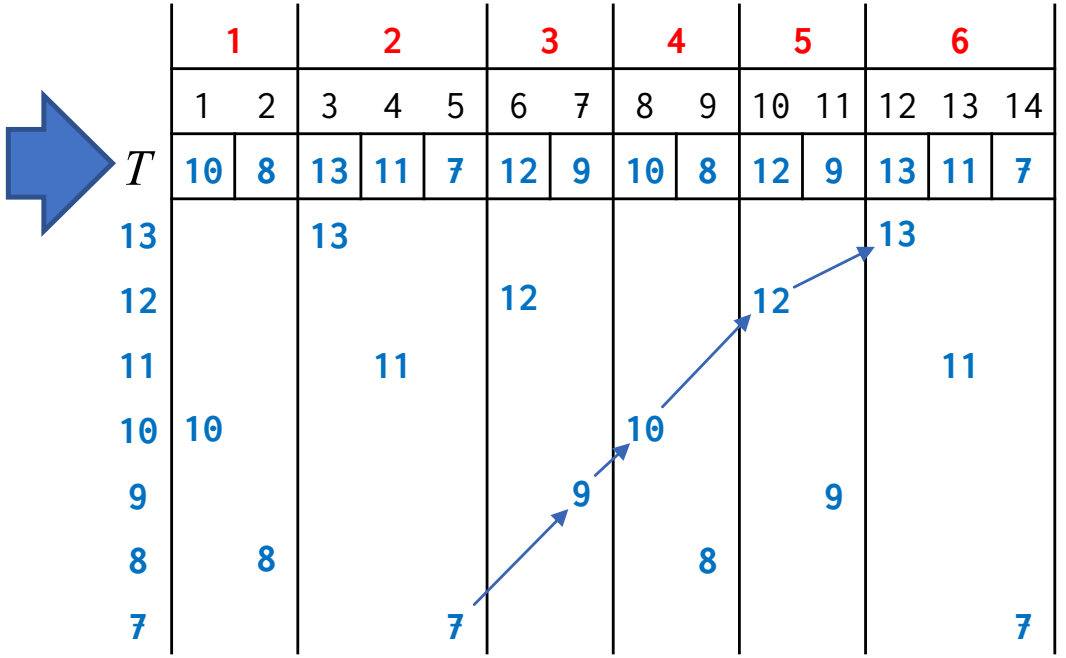
	1		2			3		4		5		6		
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
T	10	8	13	11	7	12	9	10	8	12	9	13	11	7
13			13									13		
12						12				12				
11				11									11	
10	10							10						
9							9				9			
8		8							8					
7					7									7

LCS to LIS

Consider integer sequence T corresponding to partition $S=XY$



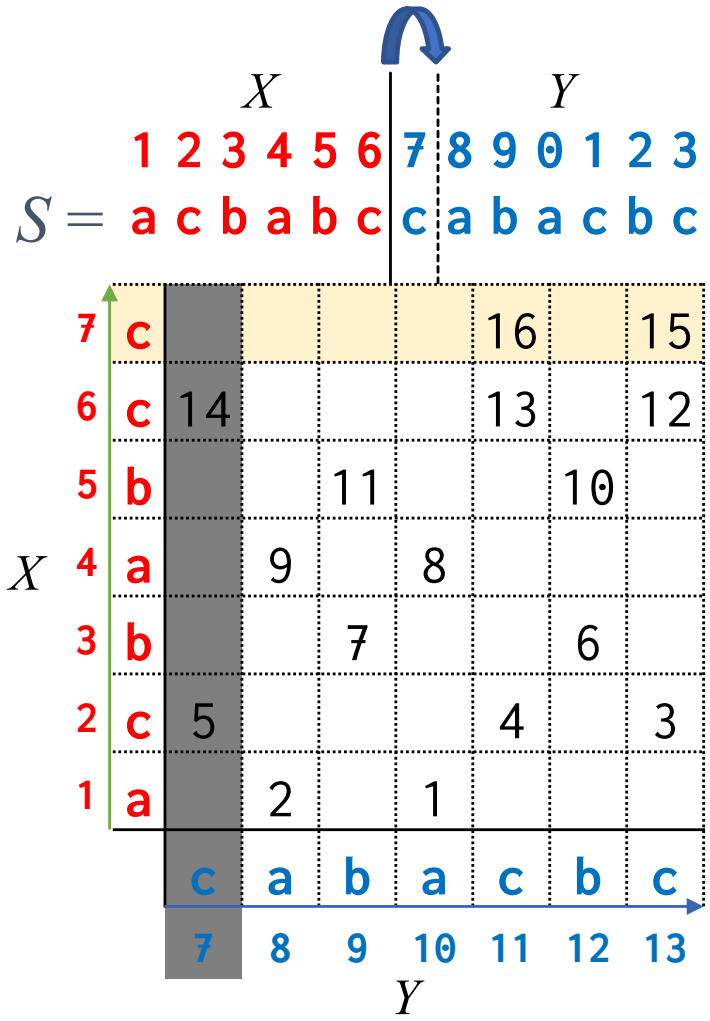


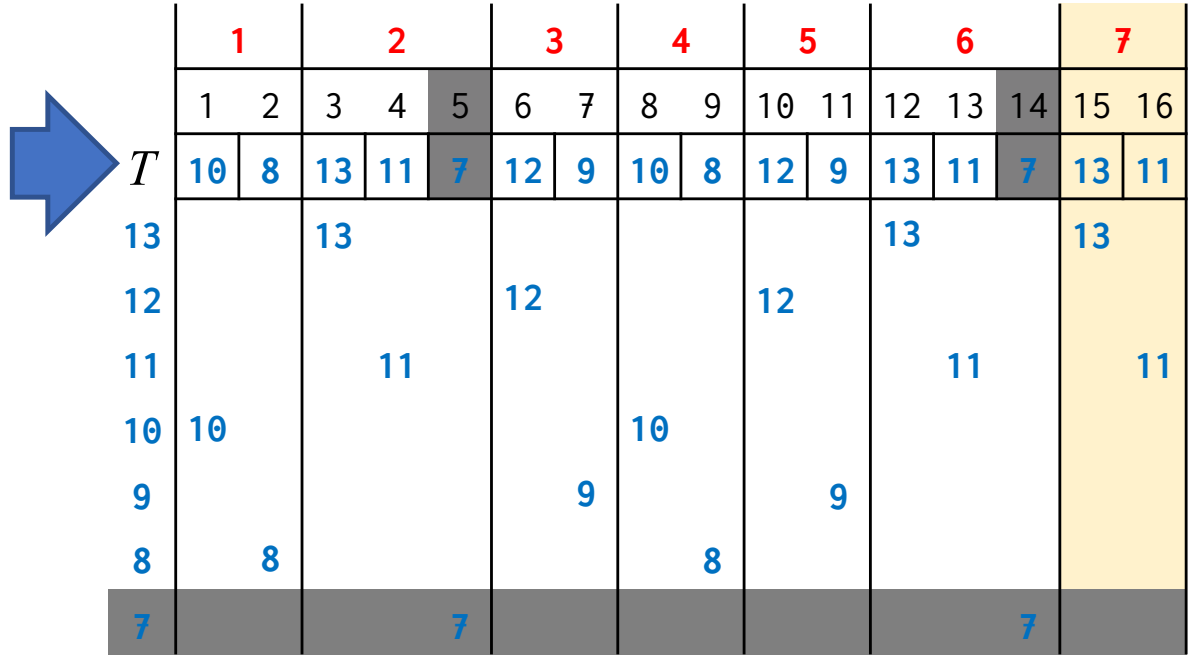


Common subsequence of X, Y of length k

$\Leftrightarrow$ Increasing subsequence of T of length k

Dynamic LCS to Dynamic LIS





append to X ⇔ **append** to T

delete_front from Y ⇔ **delete_min_all** from T

Dynamic LIS with **append** and **delete_min_all**

Theorem [Chen et al. 2013]
There is a data structure for maintaining an LIS of a numeric sequence of length M allowing

- **append** in $O(\log M)$ time
- **insert** / **delete** in $O(r \log (M/r))$ where r is the length of the LIS

We can use **append**, as is, M times

Total: $O(M \log M) = O(M \log n)$ time

If we use **delete** M times for **delete_min_all**:

Total: $O(Mr \log M/r) = O(Mr \log n/r)$ time

➔ Tweak Chen et al.'s algorithm to conduct **delete** in batches:

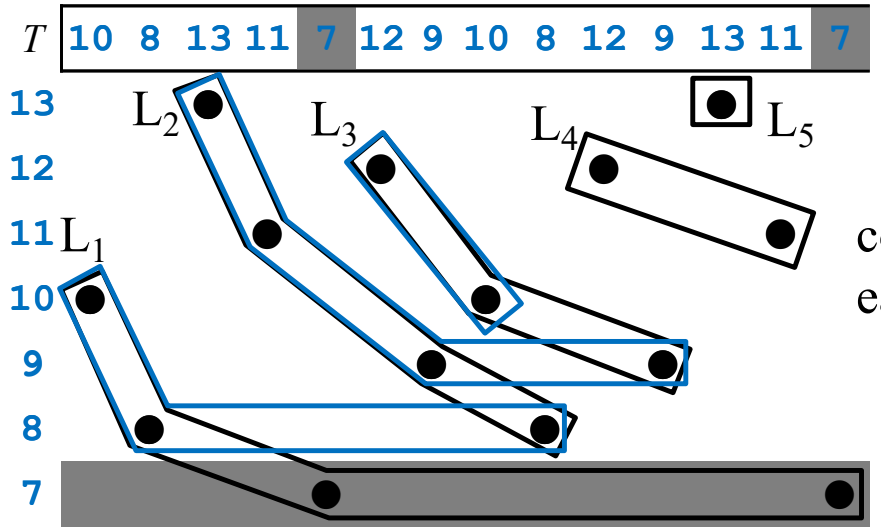
- $O(r \log n/r)$ time for each batch, $\min\{n,M\}$ batches.

Total: $O(\underbrace{r \min\{n,M\} \log n/r}_{\texttt{delete\_min\_all}} + \underbrace{M \log n}_{\texttt{append}} + \underbrace{n}_{\text{computing matching points } (n+M)})$ time

Dynamic LIS: `delete_min_all` in O(r log n/r) time

Like [Chen et al. 2013], maintain L_1, ..., L_r where r = LIS of T,

and L_i : the list of positions p such that
the length of LIS ending at p is i.



corresponding values of T in each L_i is non-increasing

- Due to deletion in L_i, elements in L_{i+1} that are less than or equal to the minimum of the new L_i move to L_i.
- Using balanced search trees for each L_i, they can be found and moved in total $O(\sum_{k=1}^{r} \log |L_k|) = O(r \log n/r)$ time.

Summary

- Showed that Longest Square Subsequence problem can be transformed to a Dynamic Longest Increasing Subsequence with **`append`** and **`delete_min_all`** operations.
- LSS can be solved in
  $O(r \min \{n, M\} \log n/r + M \log n + n)$ time
  where:
  r: length of LCS
  M: # of matching points ($|\{ (i, j) \mid S[i] = S[j] \}| \leq n^2$)

Open Questions:

- Longest periodic subsequence?
- Longest common square subsequence problem (CPM 2018) upper bound $O(n^6)$, conditional lower bound $\Omega(n^{4-\varepsilon})$.