

an $O(n \log \log n)$ algorithm for 3 letter maximum increasing common sequence problem

June 13, 2008

Abstract

Define $b^x(i)$ to be the number of β 's after i 'th α in the $x = \{1st, 2nd\}$ sequence. Define $b^x(i, j)$ to be the number of β 's between i 'th α and the last j 'th γ in the $x = \{1st, 2nd\}$ sequence. Let $\Delta b^x(j)$ be the number of β 's after the last j 'th γ in the $x = \{1st, 2nd\}$ sequence.

Clearly, $b^x(i, j) = b^x(i) - \Delta b^x(j)$ for $x = \{1, 2\}$.

Let r be the number of matching of γ and $a^x(j)$ be the number of α s before the last j 'th γ in the $x = \{1st, 2nd\}$ sequence.

The high level idea of our algorithm is as follows. We suppose there are j γ s in our solution. We will try all possible j and calculate $OPT(j)$, the optimal solution with j γ s. Formally, we will try to calculate

$$OPT(j) = \max_{i \in [0, \min\{a^1(j), a^2(j)\}]} \{i + \min\{b^1(i, j), b^2(i, j)\} + j\}$$

for every $j \in [0, r]$, where $\min\{a^1(j), a^2(j)\}$ means how many pairs of α are available if we require j γ appears in the solution.

Next, we will show how to calculate i which maximizes $OPT(j)$ for every j efficiently, say, in $O(\log \log n)$ time.

First we show for a particular j , we can know efficiently for which i , $b^1(i, j)$ is bigger (or $b^2(i, j)$ is bigger). Actually, we sort $b^1(i) - b^2(i)$ in a nondecreasing order, this will give a permutation σ of i . Now for some j , we know $b^1(i, j) - b^2(i, j) = b^1(i) - b^2(i) + (\Delta b^2(j) - \Delta b^1(j))$. Then we know, for $i \in [\sigma(1), \dots, \sigma(k)]$, $b^2(i, j)$ is bigger, where k is the largest integer with $b^1(i) - b^2(i) + (\Delta b^2(j) - \Delta b^1(j)) \leq 0$. k can be efficiently ($O(\log \log n)$ time) found if we maintain a van Emde Boas structure over data $b^1(i) - b^2(i)$'s.

Next, suppose we have a dynamic range maximum query RMQ1 whose i 'th entry is $\sigma(i) + b^1(\sigma(i))$ if $\sigma(i) \leq \min\{a^1(j), a^2(j)\}$ and $-\infty$ otherwise. We also maintain Another dynamic RMQ2 whose entry is $\sigma(i) + b^2(\sigma(i))$. if $\sigma(i) \leq \min\{a^1(j), a^2(j)\}$ and $-\infty$ otherwise.

Then it is easy to see $OPT(j) = j + \max\{RMQ1[1, k] - \Delta b^1(j), RMQ2[k, n] - \Delta b^2(j)\}$, where $RMQ[i, j]$ means the maximum between element i and j . This is because $\sigma(i) + b^1(\sigma(i)) - \Delta b^1(j) \leq \sigma(i) + b^2(\sigma(i)) - \Delta b^2(j)$ for $i = [1, k]$ and opposite for $i = [k + 1, n]$.

The overall algorithm enumerates j , and in each step some more available $i + b^x(i)$'s are inserted into the RMQ(if we enumerate j in decreasing order).

Using the data structure in [2] will give a running time $O(\log n / \log \log n)$ each step which is not satisfied. However noticing we only need insert and query operation(delete is unnecessary), we can do better.

Fortunately, bounded heap[1] can do this work. The i 'th entry $\sigma(i) + b^x(\sigma(i))$ in RMQ can be seen as a element with key i and priority $\sigma(i) + b^x(\sigma(i))$. The query $RMQ[1, k]$ can be implemented as $Boundedmin(\mathcal{H}, k)$.

1 Concluding Remark

I heard this problem when I visited MPI in Germany. I also heard the same result was also obtained by others.

References

- [1] G.S.Brodal, K.Kaligosi, I.Katriel, M.Kutz Faster Algorithms for Computing Longest Common Increasing Subsequences. CPM06
- [2] D.E.Willard Application of the Fusion Tree Method to Computation Geometry and Searching SODA92