3字符最长公共弱递增子串的O(*n*loglog*n*)算法

归泳昆

(复旦大学上海智能信息处理重点实验室　上海 200433)

摘　要　最长公共子串(LCS)和最长递增子串(LIS)是两个非常经典的基础算法问题,并且在生物信息学中已有重要应用。2006年,Brodal等人提出了最长公共弱递增字串问题(LCWIS),并且给出了2字符字母表上线性时间算法和3字符字母表上O(*n*log*n*)时间的算法。本文中,我们提出了一种新的在3字符字母表上寻找最长公共弱递增子串(LCWIS)的算法。该算法利用了两个成熟的数据结构:约束堆(Bounded heap)和van Emde Boas树。我们算法的时间复杂度是O(*n*loglog*n*),空间复杂度为O(*n*),两者都是目前为止最优的。

关键词　约束堆,van Emde Boas树,最长弱递增公共子串,生物信息学

O(*n*loglog*n*) Algorithm for LCWIS over 3 Letters

GUI Yong-Kun

(Shanghai Key Library of Intelligent Information Processing, Fudan University, Shanghai 200433)

Abstract　Longest common subsequence (LCS) and longest increasing subsequence (LIS) are two classic algorithm problems. Both of them have important applications in bioinformatics. In 2006, Brodal et al. proposed another string problem called longest common weakly increasing substring (LCWIS) problem, and they gave a linear time algorithm on 2-letter LCWIS problem, and a O(*n*log*n*) time algorithm on 3-letter LCWIS problem. In our paper, we present a new algorithm on 3-letter LCWIS problem. This algorithm relies on existing sophisticated data structures, bounded heap and van Emde Boes tree. The time and space complexity of our algorithm is O(*n*loglog*n*) and O(*n*), both of which is the best so forth.

Keywords　Bounded heap, van Emde Boer Tree, LCWIS, Bioinformatics

1　问题背景

最长公共子串(LCS)和最长递增子串(LIS)是两个经典的算法问题,它们在生物信息学等领域都有重要应用,近年来人们对这两个算法进行了广泛深入的研究。

LCS问题是组合优化和计算生物学中的重要问题,最初是Wagner和Fisher[1]在1974年提出的。1980年,Masek和Paterson[2]证明了一般LCS问题的时间复杂度的上界为O(mn/log$n$)(假设两个字符串分别长m和n,$m\geqslant n$)。但若在特殊情况下该问题仍有更好的启发式算法。

LIS问题也是一个重要的算法问题,它在生物信息学上也有不少重要应用。Schensted[3]和Knuth[4]均给出了LIS上时间复杂度为O(*n*log*n*)的算法,可以证明Θ(*n*log*n*)已经是LIS时间复杂度的下界[5]。

2005年,Yang等人[6]将两个概念结合起来提出了一个新的问题:最长公共递增子串(LCIS),并提出了基于动态规划的时间空间复杂度均为$\Theta(mn)$的算法。随后Chan等人[7]得到了一个LCIS的改进算法。Brodal等人[8]进一步提出了较小字符集上的最长弱递增公共子串(LCWIS)问题。在实际问题中很多问题的字符集是很小的,而且对于较小字符集的情况的研究,可能会发现问题的其他结构,并有助于发现在一般情况下的更好的算法。他们同时给出了在2字符字母表上寻找LCWIS的线性算法,以及在3字符字母表上O(m+$n$log$n$)的算法。

本文研究了在3字符字母表上的LCWIS问题,并且找到了时间复杂度为O(*n*loglog*n*)、空间复杂度为O(*n*)的算法,是当前该问题上最优的算法。

2　问题描述

在3字符字母表上的LCWIS问题可以描述为:令字母表$\Sigma=\{\alpha,\beta,\gamma\}$,在其中定义$\alpha<\beta<\gamma$,有两个序列分别为$A=(a_1,a_2,\cdots,a_m)$,$B=(b_1,b_2,\cdots,b_n)$,$A$和$B$的LCWIS就是字符串$(a_{j1}=b_{k1},a_{j2}=b_{k2},\cdots,a_{jg}=b_{kg})$,其中$j_1<j_2<\cdots<j_g$,$k_1<k_2<\cdots<k_g$,且$a_{j1}\leqslant a_{j2}\leqslant\cdots\leqslant a_{jg}$,$b_{k1}\leqslant b_{k2}\leqslant\cdots\leqslant b_{kg}$。如图1,$A$和$B$的LCWIS就是$\alpha\alpha\beta\beta\gamma\gamma$。

A　αγαββββαγγ

B　βααβγβαγγ

图1　LCWIS的一个简单例子

3　相关数据结构

首先介绍一下我们算法利用的两个重要的数据结构。

3.1　约束堆(Bounded heap)

约束堆[8]是一个存储{关键字,优先级,数据}三元组的数据结构,在关键字取值在{1,2,…,n}的情况下,它支持在均摊O(loglog$n$)时间内完成下面三个操作,但却无法高效地

归泳昆　硕士研究生,主要研究方向为生物信息学。

删除元素：

1) Insert(H,k,p,d)：在约束堆 H 中插入三元组(k,p,d)，其中 k 代表关键字，p 代表优先级，d 代表存储数据；

2) DecreasePriority(H,k,p,d)：如约束堆 H 中不存在关键字为 k 的三元组，则执行 Insert(H,k,p,d)，否则将这个三元组的优先级更改为 p'(原优先级)和 p 中的较小值，且用 d 代替原数据 d'；

3) BoundedMin(H,k)：返回约束堆 H 中关键字小于 k 的三元组中优先级最小的项的数据值。

3.2 van Emde Boas 树

Van Emde Boas 树[9]是一个优先队列，当关键字取值在 $\{1,2,\cdots,n\}$ 中的情况下，它支持在 O(loglog$n$)时间内完成下列操作：

1) Insert(T,k,d)：在 T 中插入一个二元组(k,d)，其中 k 是关键字，d 是所存数据；

2) Delete(T,k)：在 T 中删除关键字为 k 的二元组；

3) Lookup(T,k)：在 T 中查询关键字为 k 的二元组；

4) FindNext(T,k)：在 T 中查找关键字大于且最接近于 k 的二元组；

5) FindPrevious(T,k)：在 T 中查找关键字小于且最接近于 k 的二元组。

4 算法思路

在描述算法之前，有必要定义下列符号。

假设字母表 $\Sigma=\{\alpha,\beta,\gamma\}$，其中 $\alpha<\beta<\gamma$，我们定义 $b_x(i)$ 为在 x(1 或 2)序列中从前数第 i 个 α 之后 β 的个数；$b_x(i,j)$ 为 x 序列中从前数第 i 个 α 和从后数第 j 个 γ 之间的 β 的个数；$\Delta b_x(j)$ 为 x 序列中从后数第 j 个 γ 之后的 β 的个数；$a_x(j)$ 为 x 序列中从后数第 j 个 γ 之前的 α 的个数。

假设我们规定最佳匹配 LCWIS 中 γ 的个数为 j 个的情况下，这时 LCWIS 中字符的个数为 $OPT(j)$，则易得

$$OPT(j)=\max_{i\in[0,\min\{a_1(j),a_2(j)\}]}\{i+\min\{b_1(i,j),b_2(i,j)\}+j\}$$

我们总体上的思路是，只要对于任意 j 如果我们都能高效求得 $OPT(j)$，则遍历所有的 j，其中最大的 $OPT(j)$ 就是我们所要求得的结果。下面给出具体的算法。求 $OPT(j)$ 的主要难点在于对于任意 i 和 j 如何知道 $b_1(i,j)$ 和 $b_2(i,j)$ 的大小，以及如何求 $i\in[0,\min\{a_1(j),a_2(j)\}]$ 时 $i+\min\{b_1(i,j),b_2(i,j)\}+j$ 的最大值。下面我们分别讨论这两个问题。

4.1 $b_1(i,j)$ 和 $b_2(i,j)$ 的大小

由 $b_x(i,j)$ 的定义不难得到

$$b_x(i,j)=b_x(i)-\Delta b_x(j)$$

为了得到 $b_1(i,j)$ 和 $b_2(i,j)$ 的大小，我们考虑 $b_1(i,j)-b_2(i,j)$，$b_1(i,j)-b_2(i,j)=b_1(i)-b_2(i)+\Delta b_2(j)-\Delta b_1(j)$，我们需要知道对于哪些 i，$b_1(i,j)-b_2(i,j)>0$，哪些 i，$b_1(i,j)-b_2(i,j)<0$，而对于固定的 j，$\Delta b_2(j)-\Delta b_1(j)$ 也是固定不变的，因此我们首先想到对 $b_1(i)-b_2(i)$ 进行排序。假设得到 i 的置换 σ，使得 $b_1(\sigma(i))-b_2(\sigma(i))$ 是在 i 上递增的。由于 i 取自 $\{1,2,\cdots,n\}$，因此排序可以通过桶排序在线性时间内完成。

排序后，事实上我们希望找到的是一个分界点 k，使得当 $i\in\{\sigma(1),\sigma(2),\cdots,\sigma(k)\}$ 时 $b_1(i,j)\leqslant b_2(i,j)$；当 $i\in\{\sigma(k+1),\sigma(k+2),\cdots,\sigma(n)\}$ 时，$b_1(i,j)>b_2(i,j)$。

上述操作正好可以利用 van Emde Boas 树的 FindPrevious 操作来实现。首先，预处理时我们把($b_1(\sigma(i))-b_2(\sigma(i)),i$)作为二元组(对所有 $i\in\{1,2,\cdots,n\}$)插入到 van Emde Boas 树 T 中，对于给定 j 我们在 van Emde Boas 树 T 上执行操作 FindPrevious($T,\Delta b_1(j)-\Delta b_2(j)$)，即可在 O(loglog$n$)时间内找到 k。

4.2 求 $\max_i\{i+\min\{b_1(i,j),b_2(i,j)\}+j\}$

由于我们前面已经得到 b_1 和 b_2 的分界线 k，不妨令

$$OPT_1(j)=\max_{\sigma(i)\in[0,\min\{a_1(j),a_2(j)\}],i\leqslant k}\{\sigma(i)+b_1(\sigma(i))-\Delta b_1(j)+j\}$$

$$OPT_2(j)=\max_{\sigma(i)\in[0,\min\{a_1(j),a_2(j)\}],i>k}\{\sigma(i)+b_2(\sigma(i))-\Delta b_2(j)+j\}$$

于是，

$$OPT(j)=\max\{OPT_1(j),OPT_2(j)\}$$

所以我们可以分别处理 $OPT_1(j)$ 和 $OPT_2(j)$。

对于 $OPT_1(j)$ 而言，对于给定的 j，我们需要知道在符合条件 $\sigma(i)\in[0,\min\{a_1(j),a_2(j)\}]$ 的 i 中，且 $i\in[1,k]$，最大的 $\sigma(i)+b_1(\sigma(i))$ 是什么。我们发现，限定堆正好可以被用在这里。我们为 $OPT_1(j)$ 建立限定堆 BH1。对于 BH1，如果符合条件 $\sigma(i)\in[0,\min\{a_1(j),a_2(j)\}]$，则存放三元组($key,p,d$)，其中 $key=i$，$p=-\sigma(i)-b_1(\sigma(i))$(因为我们要求最大值，限定堆只能返回最小的优先级，所以我们需要取一个相反数)，$d=\sigma(i)+b_1(\sigma(i))$，否则，存放($i,1,-1$)。如果我们由大往小遍历 j，那满足条件 $\sigma(i)\in[0,\min\{a_1(j),a_2(j)\}]$ 的 i 会递增，因此我们不需要删除元素，也不需要提高三元组的优先级(限定堆无法高效执行这些操作)，只需将所有三元组预先 insert 一遍，再依次将满足条件的三元组 DecreasePriority 一遍，总时间为 O($n$loglog$n$)。对于给定 k，我们只需 BoundedMin(BH1,k)，就可以在 O(loglog$n$)找到 $OPT_1(j)$。

```
1:  Preprocess();
2:  σ = Bucket_Sort(b1(i)-b2(i));
3:  T = makeVEBTree();
4:  for i = 1 to n do
5:    Insert(T, b1(σ(i))-b2(σ(i)) , i );
6:  end
7:  BH1 = makeBH(); // make an empty bounded heap
8:  BH2 = makeBH(); // make an empty bounded heap
9:  for i = 1 to n do
10:   Insert(BH1, i, 1, -1);
11:   Insert(BH2, i, 1, -1);
12: end
13: lastm = 0;
14: for j = n to 1 do
15:   if  j > num1(γ) or j > num2(γ) )  then
16:    continue;
17:   end
18:   m = min( a1(i), a2(i) );
19:   for i = lastm to m do
20:     DecreasePriority(BH1, σ^-1(i), -i-b1(i), i+b1(i));
21:     DecreasePriority(BH1, n+1-σ^-1(i), -i-b2(i), i+b2(i));
22:   end
23:   lastm = m+1;
24:   (temp,k) = FindPrevious(T, Δb1(j)-Δb1(j));
25:   n1 = BoundedMin(BH1,k);
26:   n2 = BoundedMin(BH2,n+1-k);
27:   OPT(j) = j + max( n1 - Δb1(j) , n2 - Δb2(j) );
28: end
29: SOL = max_j(OPT(j));
```

图 2 算法实现伪代码

计算 $OPT_2(j)$ 与 $OPT_1(j)$ 稍有不同，因为对于 $OPT_2(j)$ 需要找到 $i\in[k+1,n]$ 上 $\sigma(i)+b_2(\sigma(i))$ 的最大值，因此我们

需要对 key 做一下处理。我们令 $key=n+1-i$，对于所有的 $i\in[k+1,n]$，$key\in[1,n-k]$，然后同样令 $p=-\sigma(i)-b_2(\sigma(i))$，$d=\sigma(i)+b_2(\sigma(i))$，这样插入$(key,p,d)$三元组之后，对于给定 k，我们做 BoundedMin(BH2，$n-k+1$)，返回的就是我们需要的 $OPT_2(j)$。

4.3 算法的伪代码实现

伪代码的第 1 行为预处理，用于计算 $b_x(i)$，$b_x(i,j)$，$\Delta b_x(i)$，以及 $\mathrm{num}_x(\gamma)$。$\mathrm{num}_x(\gamma)$表示在序列 x 中 γ 的个数。我们将预先计算的表达式存于数组中，便于将来常数时间内存取。

第 2 行，对 $b_1(i)-b_2(i)$做一个桶排序；

第 3 行建一棵 van Emde Boas 树 T，T 的作用在于快速查询 4.1 节提到的分界点 k。

第 7～8 行，建立两个限定堆 BH1 和 BH2。

第 14～28 行是算法的主循环，这里我们对 j 遍历，对每个 j 求得$OPT(j)$。

第 29 行求对所有 j，$OPT(j)$的最大值。

5 时间和空间复杂度分析

5.1 空间复杂度

我们用到的数据结构包括线性表、限定堆和 van Emde Boas 树，空间复杂度都是 $O(n)$的，因此我们算法的总空间复杂度为线性。

5.2 时间复杂度

我们针对图 2 所示的伪代码逐行分析算法的时间复杂度。

第 1 行预处理，$b_x(i)$，$b_x(i,j)$，$\Delta b_x(i)$，$\mathrm{num}_x(\gamma)$都可以较容易地在线性时间内计算；

第 2 行，桶排序，同样是线性时间的算法；

第 4～6 行，对 van Emde Boas 树 T 作了 $O(n)$次 insert 操作，时间复杂度为 $O(n\mathrm{loglog}n)$；

第 9～12 行，在限定堆 BH1 和 BH2 上累计做了 $2n$ 个 insert 操作，时间复杂度为 $O(n\mathrm{loglog}n)$；

第 19～22 行，在限定堆 BH1 和 BH2 上累计做了 $2n$ 个 DecreasePriority 操作，时间复杂度为 $O(n\mathrm{loglog}n)$；

第 24 行，在 van Emde Boas 树 T 上做了 n 次 FindPrevious 操作，时间复杂度为 $O(n\mathrm{loglog}n)$；

第 25～26 行，在限定堆 BH1 和 BH2 上累计做了 $2n$ 次 BoundedMin 操作，时间复杂度为 $O(n\mathrm{loglog}n)$；

其余操作均为线性或常数的时间复杂度。

综上所述，我们算法的总的时间复杂度为 $O(n\mathrm{loglog}n)$。

结束语 本文针对 Brodal 等人提出的 3 字符字母表上的最长公共弱递增字串(LCWIS)问题，提出了一个新的时间复杂度为 $O(n\mathrm{loglog}n)$，空间复杂度为 $O(n)$的算法。我们算法利用了 van Emde Boas 树和限定堆作为主要的数据结构，时间和空间复杂度都是目前为止最好的。研究 LCWIS 问题，对于进一步研究 LCS 和 LIS 问题都有理论价值，并且在生物信息学上具有潜在应用价值。

尽管我们的算法时间复杂度是目前最好的，但该问题时间复杂度的上界依然未知，是否存在该问题的线性算法也未知。而对于该问题的扩展，4 字母或更多字母字符表上的 LCWIS 是否存在同样高效的算法等问题，依然需要进一步的研究。

参 考 文 献

1 Wagner R A, Fischer M J. The string-to-string correction problem. J ACM, 1974, 21(1): 168～173

2 Masek W J, Paterson M S. A faster algorithm computing string edit distances. J Comput System Sci, 1980, 20(1): 18～31

3 Schensted C. Longest increasing and decreasing subsequences. Canad J Math, 1961, 13: 179～191

4 Knuth D E. Sorting and searching. In: The Art of Computing Programming, Vol 3. Reading, MA: Addison-Wesley, 1973

5 Fredman M L. On computing the length of longest increasing subsequence. Discrete Mathematics, 1975, 11(1): 29～35

6 Yang I-H, Huang C-P, Chao K-M. A fast algorithm for computing a longest common increasing subsequence. Information Processing Letters, 2005, 93(5): 249～253

7 Chan W-T, Zhang Y, Fung S P Y, et al. Efficient algorithms for finding a longest common increasing subsequence. Journal of Combinatorial Optimization, 2007, 13(3): 277～288

8 Brodal G S, Kaligosi K, Katriel I. Faster Algorithms for computing longest common increasing subsequences. In: 16th Annual International Symposium on Algorithms and Computation (ISAAC), Henan, China, 2005

9 Van Emde Boas P, Kaas R, Zijlstra E. Design and implementation of an efficient priority queue. Theory of Computing Systems, 1977, 10(1): 99～127

(上接第 230 页)

结论 本文提出了一个满足信息表最大确定性度量标准的选择 beta 的增量算法 ICObeta，定理 3.1 保证了 ICObeta 方法满足了信息表确定性度量的标准，实验结果证实了 ICObeta 的正确性和高效性。这种 β 选取方法，避免了先验知识对于 β 选取的干扰，能够让我们摆脱先验知识不足的束缚，高效的获取合适的 β 值。进一步的工作，将主要考虑信息表约简，以及规则生成的增量方法展开。

参 考 文 献

1 Pawlak Z. Rough Sets. International Journal of Information and Computer Sciences, 1982, 11(5): 341～356

2 张文修，吴伟志，梁吉业，李德玉. 粗糙集理论与方法. 北京：科学出版社，2001

3 Ziarko W. Variable precision rough set model. Journal of Computer and System Science, 1993, 46: 39～59

4 Beynon M. Reducts within the variable precision rough sets model: a further investigation. European Journal of Operational Research, 2001, 134(3): 592～605

5 Su Chao-Ton, Hsu Jyh-Hwa. Precision parameter in the variable precision rough sets model: an application. The International Journal of Management Science, 2006, 34(2): 149～157

6 Cheng Yusheng, Zhang Yousheng, Hu Xuegang. The Relationships Between Variable Precision Value and Knowledge Reduction Based on Variable Precision Rough Set Model. RSKT2006, Berlin: Springer-Verlag, Lecture Notes in Computer Science, 2006

7 An A, Shan N, Chan C, Cercone N, Ziarko W. Discovering rules for water demand prediction: an enhanced rough-set approach. Engineering Applications in Artificial Intelligence, 1996, 9(6): 645～653

8 Slezak D, Ziarko W. Variable Precision Bayesian Rough Set Model. Lecture notes in computer science, 2003, 2639: 312～315

9 Inuiguchi M, Miyajima T. Variable Precision Rough Set Approach to Multiple Decision Tables. Lecture notes in computer science, 2005, 3641: 304～313