分块方法的应用

山东省胜利第一中学 王子昱

摘 要

分块方法是一种通用的、高性价比的问题解决方法，能够在不对问题性质做过多分析的情况下给出效率较高的解法。本文通过若干例题，介绍了分块方法的各种应用，并尝试对其共性进行归纳。

引言

在信息学中有一类问题，要求支持序列（或树等结构）上区间中信息的快速统计，并可能带有各类修改操作。这一类问题的传统解法可以分为利用线段树和对序列分块两类，而线段树的使用常常有局限性，此时可行的做法就只有分块，例如第二节中的问题。另一方面，有的问题虽然可以用线段树或其它方法解决，但分块方法更为简单，有时甚至能带来更好的时间复杂度，例如第一节中的两个问题。

对序列分块的方法体现了均摊制约效率的因素的思想。这一的思想不仅可以用在对数据的维护上。定期重建的方法[41]，以及合并两类算法解决问题的方法都体现了这样的思想。这几类问题在后两节中进行了讨论。

[41]可以看做对时间线进行分块

1 Section 1[42]

1.1 Problem 1 : UNTITLE1[43]

题意

给定一个序列$A[1..n](n \leq 50000)$，设计一个数据结构，支持两种操作：

1. 0 x y k : 将$A[x..y]$增加k。
2. 1 x y : 查询$S[x..y]$的最大值，其中$S[i] = \sum_{j=1}^{i} A[j]$。

题解

本题是一类数据结构问题的代表。它要求维护一个序列，支持对区间的修改和询问。对于这类问题，一般的思路是将序列分为若干区间进行维护。如果区间的信息支持合并，我们可以使用线段树；否则，我们只能使用分块的方法。

本题中，我们尝试维护序列$S[x]$。考虑一次操作$(0, x, y, k)$，假设它涉及的区间为$[l_i, r_i]$，则问题相当于将$[l_1, r_1]$中的$S[i]$增加$(i - l_1) \times k$，将$[l_2, r_2]$中的$S[i]$增加$(l_2 - l_1) \times k + (i - l_2) \times k$，……，将$[l_j, r_j]$中的$S[i]$增加$(l_j - l_1) \times k + (i - l_j) \times k$。注意对于同一个区间$[l, r]$中的$S[i]$，其增量具有$delt + (i - l) \times K$的形式，其中C和delt是常数。所以对每个区间我们维护这两个值作为懒标记。

该区间进行的查询相当于查询$C + max(S[i] + i \times K)$的值。为了快速进行查询，我们考虑可能成为最优解的点的性质。设$y_i = S[i], x_i = i$，忽略对所有点相同的常数C，所求值$z = y_i + k \cdot x_i$可以看做过点(x_i, y_i)，斜率为$-k$的直线在y轴上的截距。查询最大值的操作，相当于寻找一个点集中，过某个点的斜率为$-k$的直线的y截距的最大值。考虑一条斜率为$-k$的直线从y轴无穷远位置向下平移，与点集中的第一个点接触时即停止，如图1所示，此时的y截距显然是最大的。该直线只与点集的上凸壳中的点接触，因此，对于任意的z，只有(x_i, y_i)的上凸壳中的点可能构成答案[44]，区间中需要维护的信息就是其凸壳。求出凸壳

[42]本文中每节介绍的问题在做法上存在共性。显式地用节标题对其进行概括是不必要的，因此节标题采用了"Section n"的简单形式

[43]Source : SPOJ

[44]严格的证明略为繁琐，大家可以自行完成，这里略去

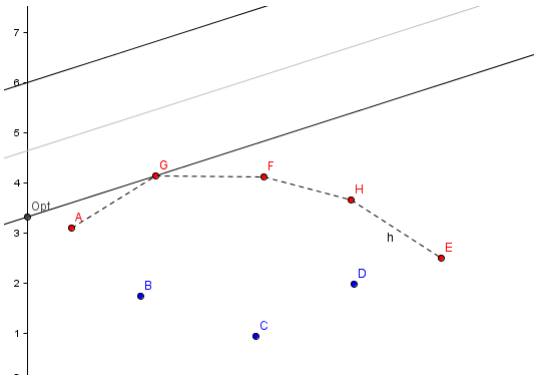

Figure 1: 某一区间对应的点集如图，考虑k=-0.32的情况，最优解对应的点为点G，可能成为最优解的点用红色标出，它们构成了点集的上凸壳

之后，注意到对于凸壳上的点(x,y)，它构成的答案关于x 呈双调。因此，可以使用三分法[45] 进行查询，复杂度为$O(logN)$。

如果使用线段树维护，区间信息的合并无法快速完成[46]。所以我们分块维护。将原序列分为M块，每块中维护$S[]$对应点集的凸壳。修改操作对应于修改至多两个块的一部分，以及修改至多$\frac{N}{M}$块的懒标记。对于修改块的一部分的操作，我们暴力重建该块的凸壳，复杂度为$O(M)$；修改懒标记的操作复杂度为$O(1)\times\frac{N}{M}=O(\frac{N}{M})$，因此，修改的复杂度为$O(M+\frac{N}{M})$。同样地，查询操作对应于查询至多两个块的一部分，以及至多$\frac{N}{M}$块的整体查询。对查询块的一部分的操作，我们暴力完成；整体查询如前所述。总复杂度为$O(M+\frac{NlogN}{M})$。取$M=\sqrt{N}$，得到算法的修改操作复杂度为$O(\sqrt{N})$，查询操作复杂度为$O(\sqrt{N}logN)$。

通过将树划分为$\sqrt{N}$块，这类问题很容易推广到树上，形成的结构称为块状树。实现的细节这里略去。

1.2 Problem 2：静态RMQ问题

题意

给定一个序列$A[1..N]$，每次询问给定l,r，查询$min_{l\le i\le r}(A[i])$。

[45]参见参考文献[4]

[46]凸壳的合并有$logN$的算法，但是过于复杂，可以参考参考文献[1]

题解

$O(N)-O(\sqrt{N})$**做法** 将序列分为$\sqrt{N}$块，将每块中第一个位置称为关键点。预处理出每块内的最小值，并利用这一信息，在$O(N)$时间内预处理出每对关键点之间的最小值；同时预处理出每个位置到离它最近的两个关键点之间的最小值。于是如果询问(l,r)中，l和r不在同一块内，我们可以利用上述信息在$O(1)$时间内解决；否则，我们暴力扫描$A[l..r]$，在$O(\sqrt{N})$时间内得到答案。

$O(NloglogN)-O(1)$**做法** 对于l,r在同一块内的情况，我们没有必要暴力扫描。我们可以对每一块递归地建立上文所述的数据结构，直到待维护的序列长度为1。可以注意到，这一结构相当于一棵每层度数不同的树，其深度为$O(loglogN)$。由于每层上预处理的代价是$O(N)$的，算法的预处理时间共为$O(NloglogN)$。

询问的时候，如果两个端点不在同一块中，我们可以$O(1)$得到答案；否则，我们递归到这一块对应的下一级数据结构中。这个做法是$O(loglogN)$的。但是如果我们每次二分一个最浅的、两端点不在同一块中的层次，询问的复杂度可以降到$O(logloglogN)$ 。

考虑如何得到询问$O(1)$的做法。如果存在整数z使得$N=2^{2^z}$，那么每一层上的块的长度是相同的。考虑长度为L的一个询问，设i 是第一个满足第i 层块长不超过L的整数，于是最浅的、两端点不在同一块中的层次必然是i或者$i-1$，只要分别验证即可。通过预处理每个L对应的i，询问的时间复杂度是$O(1)$ 的。

如果$z>loglogN$，我们可以把每一块的长度L增加到使$\sqrt{L}$是整数的最小值。此时每层块长相同的性质得到保持，上述做法仍然适用。显然，预处理的时间仍为$O(NloglogN)$。

$O(Nlog^*N)-O(1)$**做法** [47] 将序列分为$\frac{N}{logN}$块，预处理每个位置到离他最近的两个关键点之间的最小值。预处理每块内的最小值，并用这$\frac{N}{logN}$个值在$O(N)$时间内建立sparse table。于是对于跨块的询问我们就可以在$O(1)$时间内回答。为了处理块内的询问，我们对每块递归建立这一数据结构，直到当前处理的序列的长度为1。由于每次递归问题的规模降到了对数级，递归的深度是log^*N的；又每层预处理的总代价为$O(N)$，预处理的时间是$O(Nlog^*N)$的。

[47]感谢黄嘉泰告诉我这一算法

询问时用与上一做法类似的手段可以做到$O(1)$。

竞赛中，上述第二个做法的$O(NloglogN) - O(loglogN)$实现最有性价比。在竞赛中，可以利用计算机支持$O(1)$位运算的性质将这个做法优化到$O(1)$，这里不再赘述。对RMQ 问题还存在其他快速做法，这里不再讨论。

另外，上述列举的做法中，第一个和第二个可以扩展到一类一般化的问题：在无修改的情况下，询问区间信息的并，其中并的操作满足区间加法，但不满足区间减法。例如，区间最大连续子段和问题[48]。

2 Section 2

2.1 Problem 3 Fairy[49]

题意

给定一个图，询问存在多少边，使得删掉这条边之后原图为二分图。

顶点数$N \leq 100000$，边数$M \leq 100000$。

题解

本题存在线性做法，但思维和实现的复杂度都比较高。这里我们介绍一种简单暴力的做法。

如果要验证一个图是否为二分图，我们可以用并查集完成：维护每个点到其联通分量中代表元距离的奇偶性，加入边(u, v)时，在并查集中合并点u, v，如果u, v本来在同一联通块中，并且到代表元距离的奇偶性相同，则加入该边会引入奇环，原图不是二分图。

利用这一性质，我们可以将边表分块，对于每一块，首先计算出将此块之外的所有边加入并查集得到的结构；之后对于块中的每一条边，我们将块中除它以外的所有边加入并查集，并判断是否产生矛盾，之后撤销加入本块边的操作。撤销操作可以用栈维护。

注意并查集的$O(\alpha(N))$的复杂度是基于平摊分析得到的，因此此时算法的复杂度不能看做$O(N\sqrt{N}\alpha(N))$，而是$O(N\sqrt{N}logN)$。但是如果我们在将除去

[48]SPOJ GSS1

[49]Source : Codeforces 19E

该块之外的所有边插入之后进行缩点，前后两步的操作就是不相关的了，利用平摊分析可得，此时算法的复杂度为$O(N\sqrt{N}\alpha(N))$。

3 Section 3

3.1 Problem 4 : 糖果公园[50]

题意

给定一棵N个点的树，每个点带权。共有M种不同的权值。Q组操作，每次可以修改一个点的权值，或是询问连接点s_i, t_i 的路径的权值。

其中路径的权定义如下：给定数组V，W，设路径上第i种权值W_i的出现次数为N_w，则该路径的权值为

$$\sum_i \sum_{j=1}^{N_w} W_i V_j$$

数据分为三类：[51]

1. 树是一条链，无修改。
2. 树不是链，无修改。
3. 树不是链，带修改。

本题的标准做法是离线处理询问，并在询问中进行转移，详情参见集训队作业中的冬令营题解。在这里，我们讨论与其不同的一系列在线算法。

3.1.1 链状无修改的情况

对于树为链状，且无修改的情况，我们有以下做法：

将链分为$\sqrt{N}$块，预处理从每块的起点（以下称关键点）出发，到每一个点的答案。这一过程是$O(N\sqrt{N})$的。

考虑一次询问，设其左端点为l，右端点为r，设离l最近的块起点为L，于是$[L, r]$的答案已被求出。我们枚举$[l, L)$中的每一个点，把它纳入统计，并更新

[50]Source : WC2013

[51]去除了不具代表性的一些情况

答案。为了更新答案，我们需要知道某个权值$W[i]$在$(i,r]$中出现的次数，利用前缀和的方法，转化为查询某个权值在$[1,i]$和$[1,r]$中的出现次数的差。因此，我们维护$Okr[i][w]$，表示前i个数中离散化后的权值w出现的次数。

直接用二维数组进行维护的复杂度是$O(N^2)$的，但是注意到$Okr[i]$和$Okr[i-1]$仅在一个位置有差别，因此我们用持久化块状链表维护$Okr[i]$。将$Okr[i]$分为$\sqrt{N}$块，维护指向每块的指针。考虑$Okr[i]$的计算，其中有$\sqrt{N}-1$块和$Okr[i-1]$中相同，我们将指向$Okr[i-1]$中对应块的指针直接复制过来；对于剩下的一块暴力新建。因此构建$Okr[i]$只使用了$O(\sqrt{N})$的额外内存。定位任意一个$Okr[i][j]$的复杂度是$O(1)$的。

于是，预处理的复杂度为$O(N\sqrt{N})$。查询时需要额外考虑至多$\sqrt{N}$个点，其复杂度为$O(\sqrt{N})$。

算法的伪代码如下：

Algorithm 2 Init

```
1: bsize = √n
2: bcnt = [n / bsize]
3: for i = 1 to n step bsize do
4:     for j = i to n do
5:         calculate Ans[i][j]
6:     end for
7: end for
8: for j = 1 to bcnt do
9:     okr[0][j] = new int[bsize + 1]
10: end for
11: for i = 1 to n do
12:     p = [(W[i] − 1) / bsize] + 1
13:     copy okr[i] from okr[i − 1]
14:     okr[i][p] = new int[bsize + 1]
15:     copy okr[i][p] from okr[i][p − 1]
16:     okr[i][p][W[i] mod bsize + 1] + +
17: end for
```

Algorithm 3 Query

Require: the query interval $[l, r]$

Ensure: the answer ans

$L = \frac{l}{bsize} + 1$
if l mod bsize $\neq 0$ **then**
  $L = L + 1$
end if
$ans = Ans[L][r]$
for $i = l$ to $L - 1$ **do**
  $p = [\frac{W[i]-1}{bsize}] + 1$
  q = w mod bsize
  $o = okr[r][p][q] - okr[i][p][q]$
  $ans = ans + V[o]$
end for
return ans

3.1.2 树状无修改的情况

对于树不是链的情况，做法是类似的。将树分为至多$\sqrt{N}$块，使得每块的高度不超过$\sqrt{N}$[52]。用与上一节同样的方法，预处理出每个关键点出发到任意一个点的答案。与链上类似，我们维护$Okr[i][w]$ 表示从根到点i 的路径上离散化后的权值w的出现次数，于是加入点更新答案的操作可以用类似链上的方法解决。算法的复杂度与链上的情况相同，为$O((N+Q)\sqrt{N})$。

3.1.3 树状带修改的情况

在有修改的情况下，我们考虑对操作分块解决问题。

将所有操作分为$Q^{1/3}$块，统一处理同一块中的操作。对于块中的询问，首先将此块之前的所有修改操作应用到树中，然后利用上一节中的做法，处理出仅考虑此块之前所有修改操作情况下，块中的每一个查询的答案（第一步）；之后对于块中的每一个查询，将同一块中在它之前的所有修改纳入统计中，计算它对答案带来的影响（第二步）。

[52]这样的分块显然总能完成

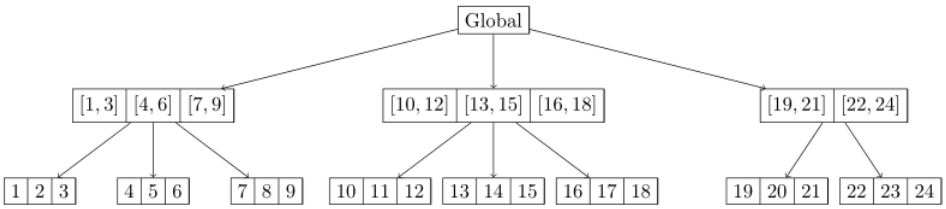

Figure 2: 有26个元素的三层块状表

考虑上面的第一步。我们将树分为$N^{1/3}$块，处理出从每个关键点出发，到达任意顶点的答案。在维护Okr数组的时候，如果我们继续利用持久化块状表，这一步的复杂度就会达到$O(N^{3/2})$，这是难以接受的。所以我们将块状表中的每个块再次分块。[53]将Okr 数组分为$N^{1/3}$组（第一层），每组中的$N^{2/3}$个元素用持久化块状表维护（第二层）。这一结构也可以被看做一个$N^{1/3}$叉树。每次我们只要修改一个（二层的）块状表，并新建一个大小为$N^{1/3}$的（底层）数组，因此，计算Okr数组的复杂度降到了$N^{4/3}$；而在Okr中定位一个元素的复杂度仍然可以看做$O(1)$，因此，处理一个询问的复杂度（在这一步）是$O(N^{2/3})$的。

在第二步中，我们考虑每个在当前询问之前的修改，判断它是否在当前询问的路径上。如果在，就修改答案。使用询问复杂度为$O(1)$的LCA算法，这一步的复杂度为每个询问$O(Q^{2/3})$。因此，算法的总复杂度为$O(Q^{1/3}(N^{4/3}+Q^{4/3}))$。

与标准做法相比，这一做法的优势是它不必离线。利用类似的方法，我们可以在线解决很多“无修改问题可以解决，带修改问题无法直接解决”的问题，或者只有一类操作无法直接解决的问题，例如带插入和修改的区间K值问题。

3.2 类似问题：带插入的区间K值问题[54]

问题描述

给定一个序列$A[1..n]$，支持三种操作：

1. M x y：将$A[x]$修改为y；

[53]对于Okr数组的维护，另外一个解决方案参见参考文献[3]

[54]Source：BZOJ

2. I x y：将y插入到$A[x]$后面；

3. Q l r k：查询$A[l..r]$中的第k小值

操作数q不超过1.75×10^6，原序列长度$n \leq 3.5 \times 10^4$。要求在线算法。

题解

如果只有修改和查询，我们可以用树状数组套线段树在$O((n+q)log^2n)$的时间内解决：树状数组中每个结点维护其对应的区间中所有元素组成的动态线段树；修改时在对应的$logn$棵线段树中修改；询问时二分答案，计算比当前答案小的元素个数，由于所有线段树的结构相同，询问的复杂度为$O(log^2n)$。

在有插入操作的情况下，将所有操作分为m块，将本块之前的所有插入操作应用到序列上，构造当前序列的树状数组套线段树结构。询问时计算考虑本块中插入的情况下，实际的询问区间；在每次二分时扫描所有本块中的插入，修正答案。算法的复杂度为$O(m(nlogn + \frac{q}{m}logn))$，取$m = \sqrt{q}$，得到复杂度为$O((q+n)\sqrt{m}logn)$。

4 Section 4

4.1 Problem 5[55]

题意

给定N个字符串集合，初始时每个集合中仅有一个字符串。要求支持操作：

1. MERGE A B 删除集合A、B，插入集合$A \cup B$。

2. QUERY S s 查询集合S中的串在字符串s中出现的次数。

字符串的总长$Slen$不超过10^5。

[55]Author：**黄嘉泰**

题解

利用二项队列思想维护AC自动机，可以得到$O(NlogN)$的做法。但是这种做法的思维和编程复杂度都比较高。这里介绍一种简单的$O((N+Qlen)\sqrt{Slen})$做法，其中Qlen是询问串的总长度。

设$L=[\sqrt{Slen}]$。将模式串分为长度大于L和小于L的两类，于是长度大于L的字符串至多有$[\frac{Slen}{L}]$个。对每个集合中长度小于L的串，预处理它们的hash 值，并按长度存储在Hash表中；对长度大于L的串，我们计算它们的在KMP算法中的$next$ 数组。

合并时只需要对Hash表进行启发式合并。询问时，对于长度大于L的串，暴力进行KMP，由于它们的$next$ 数组已被计算，这一步的复杂度是$O(qlen\times\frac{Slen}{L})$的，其中$qlen$是当前询问串的长度；对长度小于$L$ 的串，枚举匹配的起始位置，之后枚举模式的长度，在对应的Hash 表中进行查询，复杂度也是$O(qlen\times L)$的。

至此，问题得到解决。

4.2 Problem 6 KAN[56]

题意

如果两个区间有交集，我们称它们互相匹配。

给定N个区间$A[1..N]$。M组询问，每次询问给定一个区间$[l,r]$，确定A中的一个最长连续子序列，使得其中每一个区间都与区间$[l,r]$匹配。

$N\leq 50000, M\leq 200000$。

题解

区间$[l,r]$和$[l',r']$匹配$\Leftrightarrow l\leq r', r\geq l'$。因此，如果要判断一个询问和一个区间集合是否全部匹配，只需要计算该区间集合左端点的最大值（设为$first$）和右端点的最小值（设为$second$），然后与询问的端点进行比较即可。

如果一个询问的答案长度不超过L，我们可以对每个$l\leq L$，预处理出$S[l]$表示长度为l的所有连续区间的$(first,second)$有序对的集合。如果要验证该询问

[56]Source : Algorithmic Engagements 2011

（设为$[l,r]$）的答案是否不小于l，我们只要在$S[l]$中查找$first$不超过r的元素中，$second$的最大值，验证其是否不小于l即可。注意到对于元素a，如果存在b使得$b.first \leq a.first, b.second \geq a.second$，那么元素$a$就是无用的。在$S[l]$中去除所有这样的元素$a$后，按照$(first, second)$排序，每次验证就可以找到$first$不超过$r$的最大元素，验证其$second$是否不小于$l$。于是二分答案，再用上述方法验证即可。这一部分的复杂度是$O(NL + Qlog^2N)$的。

如果询问的答案长度超过L，那么将序列A分为长度不超过L的若干块之后，答案必然横跨至少两块。将序列分块，每块维护块内前i个区间和后i个区间的$(first, second)$对，于是对于每个询问，我们就可以二分得到其在每块中从左端点出发的最长匹配长度，以及从右端点出发的最长（逆序）匹配长度。利用这一信息，我们可以对所有块扫描一遍得到答案。复杂度为$O(Q\frac{N}{L}logN)$。

算法的总复杂度为$O(NL + Q(log^2N + \frac{N}{L}logN))$。取$L = \sqrt{NlogN}$，得到复杂度为$O((N+Q)\sqrt{NlogN} + Qlog^2N)$。

总结

分块方法的优势在于不需要对问题的性质进行过多的分析，具有通用性。其缺陷在于实现比较繁琐，且时间复杂度有时难以承受。在实际应用中，这一方法具有较高的性价比。

致谢

感谢在本文写作过程中提供帮助的罗雨屏、黄嘉泰、邢皓明三位同学。

感谢提供例题的许多同学。

参考文献

[1] 陈立杰，《可持久化数据结构研究》, 2012年集训队作业

[2] 王子昱，《糖果公园 解题报告》, 2013年集训队作业

[3] Anders Kaseorg, “Optimal worst-case fully persistent arrays”, TFP 2009

[4] “Ternary Search”, Wikipedia