莫队及分块扩展的研究

福州一中 陈俊锟

0 Abstract

分块是一种通过均摊思想解决问题的数据结构。这种数据结构一般是以 $n^{1/2}$ 为维护数据的单位，其时间复杂度一般也和 $n^{1/2}$ 有关。莫队算法是一种离线算法，可以解决很多序列及树形结构上的问题。有一些分块的扩展，支持把离线算法转化为在线算法，从而解决一系列强制在线的问题。

莫队算法的要求：**允许离线**、信息支持**单点增量**和**单点减量**，修改**可以撤销**。

分块预处理的要求：**内存充足**、信息支持**单点增量**、**无修改**。

分块重建的要求：可以通过查询判断修改对它的影响。

一些情况下，以上方法需要有比较充裕的时间限制。

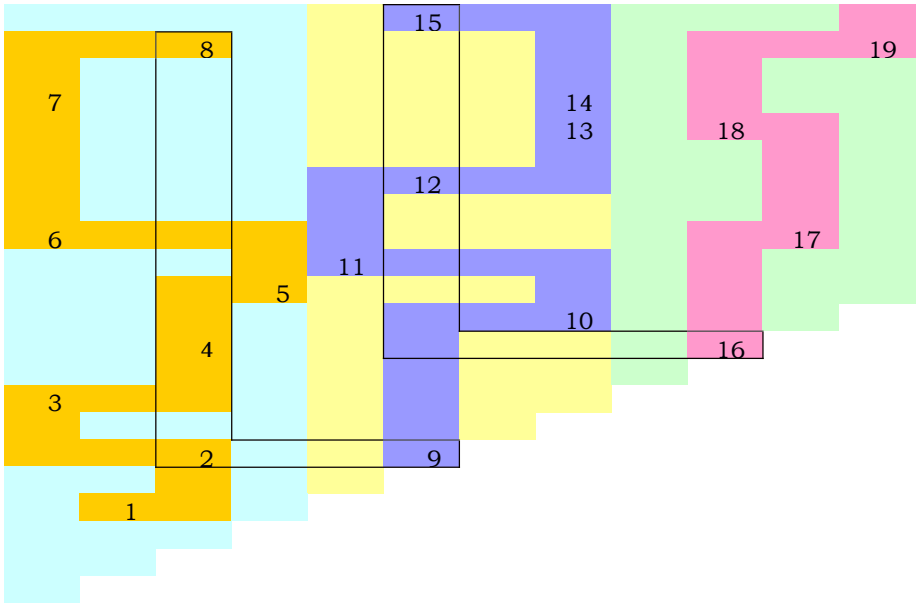
1 莫队算法

有一些信息维护的问题，允许使用**离线算法**，其维护的信息不支持区间快速合并，比如询问区间内不同数字的个数。这类问题是线段树这类数据结构难以胜任的[1]，因为区间内不同数字的个数和这些数字具体是什么有关，所以区间合并的复杂度是Ω(n)的（数组维护出现次数，记录不同数字个数）。但这种问题的数据支持**单点增量**——如果已经有了一个包含 num[l,r-1]的所有数字的出现次数的数组 cnt[]、一个 num[l,r-1]内不同数字的答案 ans，则我们只需要令 cnt[num[r]]+1，并判断 cnt[num[r]]是否为 1 即可。同理，**单点减量**也是可行的（判断-1 以后的 cnt[num[r]]是否为 0）。这样的问题就可以用莫队算法解决。

莫队算法的核心就是，对于两个询问[a,b]和[c,d]，设 x 次合并的时间是 f(x)，我们就可以以 f(|a-c|+|b-d|)的时间复杂度从[a,b]的数据转移到[c,d]的数据。如果把它们看成二维点，转移的时间就和它们的曼哈顿距离|a-c|+|b-d|有关。因此，按照一定的顺序访问这些点，就能回答所有询问。

我们先离线处理所有询问，接着问题就转化为：求平面上所有点的最短曼哈顿距离的哈密顿路径。这个问题不可解。但我们有一个很好的替代品：对询问的某个端点分块。设块的大小为 H，对于每个询问(a,b)以 B[a]（包含 a 的块）为第一关键字，b 为第二关键字排序，再按这个顺序处理。由于 l ≤r，因此坐标系是缺角的。

[1] 在 2015 年徐寅展的集训队论文中有提到一种离线用树状数组维护的方式

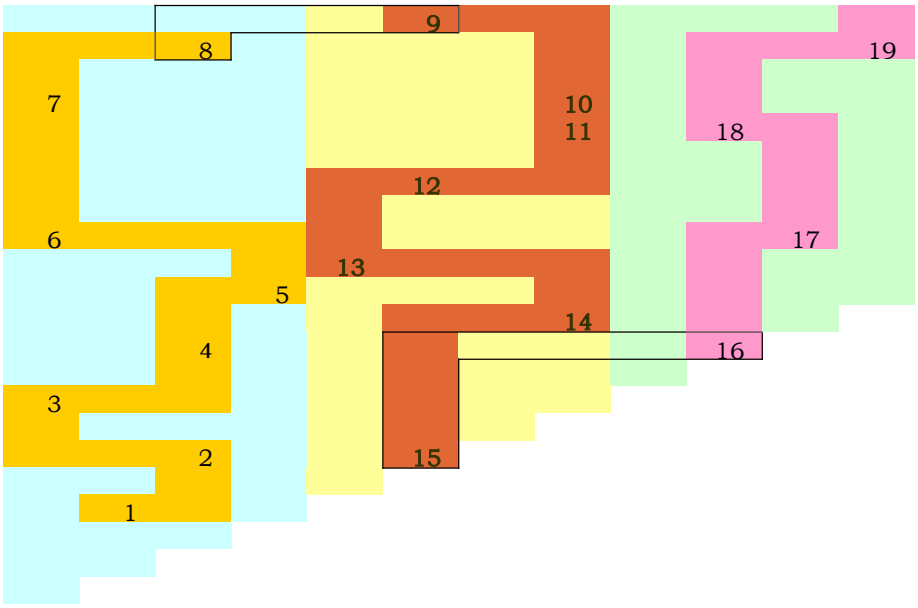


其中，用另一种颜色表示块内点的处理路径，框框表示块切换时的路径。

注意到同一个块中，纵坐标的是单调的，每块内改变量≤n，切换块的时候改变量≤n，因此纵坐标的改变量是 $O(n^2/H)$的。而在相等块内每次切换为下一个点的时候，横坐标的改变量不会超过 H（如 5 到 6），切换块的时候改变量不会超过 2H（如 15 到 16），这样横坐标的改变量是 O(nH)的。取 $H=n^{1/2}$，总时间复杂度 $O(f(n^{3/2}))$。

1.1 一个效果很好的优化[2]

对于分块的处理，我提出了一个问题：在上图中 8 到 9 的块的切换似乎浪费了很多纵坐标的移动。其实，我们可以换一种顺序：



将中间块的纵坐标倒序，发现左边块的切换过来的纵坐标改变量大大缩短，

[2] 原创

中间块切换到右边块的纵坐标改变量也缩短了。虽然总复杂度没变，但实际上常数大大缩小了——在询问较多，分布均匀的时候，切换块的纵[3]坐标改变量是期望 O(1)的。

因此，我们应交替对块的纵坐标升序、降序排列。具体方法就是如果块编号为奇数，则升序，否则降序。

例 1 一道果题[4]

给出一个长度为 n 的序列 num[]，1 ≤ num[i] ≤ n。有 m 个询问，每次询问区间[l,r]内数字大小范围在[a,b]内的数字种数（不同数字个数）。

范围及限制：n ≤ 100,000，m ≤ 500,000，时间限制 5s，内存 256M。

这题是前面那个问题的加强版，给不同数字个数加上了范围。在徐寅展的论文中提出了一种利用线段树分治套树状数组的离线算法，复杂度为 $O(n\log^2 n)$，但实际代码不是很好写。

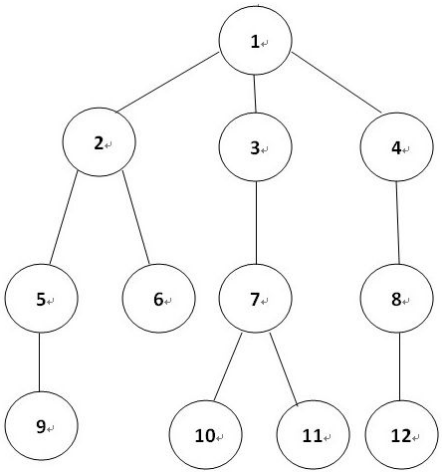
同样考虑莫队算法。如果数字不加范围，只需要 1 个辅助变量，但如果加了范围，就需要别的数据结构维护了。有同学可能想到用权值树状数组维护权值范围内不同数字个数，但这样操作的效率是修改 O(logn)查询 O(logn)的，总复杂度将达到 $O((n+q)n^{1/2}\log n)$，无法在规定时间内通过。

其实我们的瓶颈只在修改上，询问完全可以做到更高的复杂度。因此我们用权值分块维护权值范围内不同数字个数，修改只需修改所在块，O(1)，查询是分块的 $O(n^{1/2})$。时间复杂度降为 $O((n+q)n^{1/2})$，就可以 2.5s 左右出解了。

1.2 树上的莫队[5]

序列上的询问可以用区间表示，因而也可以转化为坐标，方便解题。但树上路径的询问就没那么简单了——如果直接用询问的端点记录询问，则将会将坐标扩展到多个维度，几乎不可写。

考虑如何将树上询问转化为序列问题。对于一棵树，将其括号序列表示出来：



[3] 已更正

[4] Source：YZOJ2063

[5] 有参考学习汪文潇的代码

1	2	5	9	9	5	6	6	2	3	7	10	10	11	11	7	3	4	8	12	12	8	4	1
【	【	【	【	】	】	【	】	】	【	【	【	】	【	】	】	】	【	【	【	】	】	】	】

考虑括号序列的性质：

1、每个点有且只有两个括号，因此括号序列长度为 2n。

2、每个点的括号之间表示它的子树，括号不会相交

3、若 L[a]<L[b]，R[b]<R[a]，则 b 在 a 的子树中

但这些性质似乎对于路径并没有什么帮助。对于询问(a,b)，我们考虑 a 的左括号 L[a]到 b 的左括号 L[b]之间的序列。（假设 L[a]≤L[b]）。

【1】(9,11):

9	9	5	6	6	2	3	7	10	10	11
【	】	】	【	】	】	【	【	【	】	【

【2】(3,12):

3	7	10	10	11	11	7	3	4	8	12
【	【	【	】	【	】	】	】	【	【	【

【3】(1,11):

1	2	5	9	9	5	6	6	2	3	7	10	10	11
【	【	【	【	】	】	【	】	】	【	【	【	】	【

【4】(8,8):[6]

8
【

在上面的括号序列中删去左右括号都在序列中的点（称为取对称差），我们发现了一些神奇的性质：

1、【1】【2】：若 a 和 b **无祖先关系**（L[a]<R[a]<L[b]<R[b]），则取对称差后的括号序列包含 **a~b 之间路径去掉 a 和 LCA(a,b)**的点的集合。

2、【3】：若 a 是 b 的**祖先**（L[a]<L[b]<R[b]<R[a]），则取对称差之后的括号序列就是 **a~b 之间路径**上的点集合。

3、【4】[7]：若 a=b，则括号序列只有 a

为了取对称差，我们对每个点维护一个 mark 标记，表示这个点是否在区间中。并使用 reverse()函数对点进行反转：若在区间中，则删除，否则加入。伪代码如下：

```
reverse(i):
    if mark[i]:
        mark[i] = false
        info.remove(i)
```

[6] 已更正

[7] 已更正

```
else:
    mark[i] = true
    info.add(i)
```

对于第 3 种情况，可以直接回答，无需莫队，现在考虑怎么回答前两种问题。我们用(L[a],L[b])的方式将树上问题转化为区间问题。假设我们获得了区间[L[a], L[b]]的信息 info，我们还需要判断是 1 或 2 中哪一种情况。——只需根据括号序列 O(1)判断即可。伪代码如下：

```
solve(a, b):
    info = adjust(L[a], L[b])          #使用莫队获得 L[a]到 L[b]的数据
    if R[a] < L[b]:                    #情况 1
        info.add({a, LCA(a, b)})       #将 a 及 LCA 加入 info
    ans = info.ans
    if R[a] < L[b]:                     #现在要把 a 和 LCA 删掉
        info.remove({a, LCA(a, b)})
    return ans
```

这样，我们就以同阶的复杂度解决了树上路径询问。注意到可能要求出 LCA，而倍增算法太慢，我们考虑在 dfs 维护括号序列的时候顺便做**树链剖分**。如果 dfs 会爆栈，则用链表+bfs 维护括号序列。

例 2 苹果树[8]

给出一棵树，每个点有一种颜色。每次询问从 x 到 y 的路径上，如果把颜色 a 看成颜色 b，会有多少种不同的颜色。

树上的点和询问数目不超过 100,000，时间限制 2s，内存 256M。

这是一题非常简单的树上莫队问题。只需要维护 cnt[]和不同颜色个数即可。统计答案时，如果 a≠b，且 cnt[a]和 cnt[b]都不为 0，则答案-1。这个算法可以在 1s 之内通过全部测试点。

例 3 Jc 的宿舍[9]

给出一棵 n 个点的树，每个点有一个权值。有 m 个询问，每次询问是将一条链上所有点的权值存到数组 a[]中，可以打乱顺序，然后求 a[]的前缀和数组 b[]，要求最小化 b 中所有元素的和（即接水问题，要求最小化等候时间和）。

n≤50,000，m≤100,000，强制在线。时间限制 8s，内存 512M。

考虑没有强制在线的情况。接水问题的贪心解法是将 a 从小到大排序后求前缀和再求和，则我们可以用平衡树或线段树维护这个值。对于每个节点维护大小 size、总和 sum 和前缀和求和 ans，则两段区间信息的合并为：

```
operator (L) + (R):                    #保证 L 中所有元素小于 R，即符合性质
    result.size = L.size + R.size
    result.sum = L.sum + R.sum
    result.ans = L.ans + R.ans + L.sum * R.size
    #这个式子的解释是：R 中所有 R.size 人每人都要等候 L.sum 的时间
```

[8] Source：BZOJ3757

[9] Source：BZOJ3460

return result

因此用线段树或平衡树维护信息即可。

可这一题不是强制在线吗？但我们仔细看看强制在线的加密式子（input_k 是输入数据，last 是上一次答案）：

$$k = (input\_k + (last \bmod 2) * key) \bmod n + 1$$

实质上 k 只有可能有两个值：input_k mod n + 1 和(input_k + key) mod n + 1。因此把一个询问变成两个询问都解决，然后从头到尾输出、决定取哪一个询问的值即可。时间复杂度为 $O(n^{3/2}\log n)$。

这个方法在本机上测试，17 组数据只要 10s，交上去总共 60s。后面还会介绍一些“正统”的在线算法。

1.3 带修改的莫队

前面的莫队都是不带修改的，但实际上莫队也能解决一系列带修改的问题。

如果每一次修改，我们都对前面这些询问进行一遍莫队算法，在修改比较多的情况是会超时的。因此我们考虑：如果能把所有询问按一个顺序处理，又能回到任何一次修改后的状态，那么就可以解决了。

因此，我们把询问描述为(l,r,time)，表示一次询问的左右端点和时间，对应到三维空间中的点(x,y,z)。如果我们按照一定顺序处理这些询问，就能解决问题。但是，z 不一定是递增的，因此我们的修改就必须**能够撤销**。这样，当 z 减小的时候，我们才能处理出对应的数据。

结合 KD-Tree 的一些常识，我们知道，在三维空间中莫队算法的块大小最好是 $n^{2/3}$。然后按照 l 的块为第一关键字，r 的块为第二关键字，time 为第三关键字，结合 1.1 节中的优化，就能做到 $O(f(n^{5/3}))$的复杂度。看起来和暴力 $O(n^2)$ 区别不大，实际有很快的运行速度。

例 4 糖果公园[10]

给出一棵 n 个节点的树，树上有范围为[1,m]的权值。给定数组 v、w。有 q 个操作，每次可以修改一个点的权值，或询问点 x、y 之间，score[i]（1≤i≤m）的和。其中 score[i]为所有 w[i]*v[j]（1≤j≤cnt[i]）的和。

n≤100,000，m≤100,000，q≤100,000。时间限制 10s，内存 512M。

若存在将 x 修改为 y 的修改操作，只需要记录 x 修改前的颜色 a，则其反操作即为“把 x 修改为 a”，而修改操作已经给出，所以每个询问的反操作是给定的，因此可以用树上莫队解决。直接用即可。

应用了 1.1 节中的优化和树链剖分找 LCA，本题在 UOJ 上以 7s 内通过了所有 10 组测试点，是一个非常优秀的算法。本题存在一个时间复杂度同阶的在线算法[11]，但常数较大。

---

[10] Source：WC2013 park

[11] 见 2013 年王子昱的集训队论文

2 分块预处理

分块预处理既可以看做分块的扩展，也可以看做莫队的扩展。它具有将在离线转化为在线的方式，但付出的是内存的代价。

分块把序列从 $n^{1/2}$ 个关键点分割开，如果我们预处理出所有关键点两两之间的答案，那么对于询问(l,r)，我们可以直接找到 l 后面第一个关键点 a 和 r 前面第一个关键点 b，获得预处理的(a,b)之间的答案，然后把这个答案分别合并上[l,a-1]和[b+1,r]所有点的数据。由于没有撤销，因此无需支持单点减量。

预处理的方式有很多。最简单的方式就是从每个关键点开始维护到后面所有关键点的信息。这样的时间复杂度是 $O(f(n^{3/2}))$的。处理完后，每次询问的时间也是 $O(f(n^{1/2}))$的，和莫队同阶。

如果暴力储存所有状态，容易爆内存，因此可以考虑可持久化数据结构。由于这些数据结构一般带 log，因此通过设置 $H=n/\log^{1/2}n$ 来获得 $O(n^{3/2}\log^{1/2}n)$的复杂度。但 log 一般都不大于 20，优化其实只是理论上的。

例 3 Jc 的宿舍

题目前面已经有了。

如果强制在线是异或上一次答案，那么上一个方法就不能用了。可以用分块预处理的思想，对括号序列分块预处理。使用可持久化平衡树维护。要注意做好内存回收。注意到从一个关键点到下一个关键点中，如果节点是在上一个关键点之后出现，那么就可以直接修改，不必持久化。经过一定的内存优化后，可以做到 $O(n^{3/2}\log^{1/2}n)$的时间复杂度。但几乎不具备可写性。

本题在徐寅展 2015 年论文中还有一个对权值分块并结合虚树的在线做法。

3 分块重建

分块重建本质上是对时间线进行分块，结合的是暴力思想。

考虑单点修改、询问区间和的问题。假设当前我们维护了前缀和，并且前面有 C 个修改，设考虑 x 次修改的时间为 g(x)，我们可以以 O(g(C))的时间回答一个询问——只需要暴力枚举询问判断有没在区间内即可。如果我们每 H 次暴力重建一次前缀和，则询问的复杂度不会超过 O(g(H))，添加新的修改 O(1)，总共要重建 m/H 次前缀和，每次 O(n)，因此总复杂度为 O(m*g(H)+n*m/H)。若 n、m 同阶，g(x)=O(x)，取 $H=n^{1/2}$，则总复杂度看做 $O(n^{3/2})$。

但这样有一个问题：如果修改是互相影响的，修改就不是 O(1)了。比如带插入的序列，如果第一次在 5 的位置插入，第二次在 3 的位置，那么实际上第一次的最终位置是 6。这时候我们就要处理这种影响。如果插入在 a 位置，则遍历插入数组 mod[]。如果 a≤mod[i].at，则令 mod[i].at+1。这样修改操作变为 $O(n^{1/2})$，总时间复杂度不变。

这类题目一般维护一个序列或树的某些信息，需要用到一类数据结构。我们在每隔 H 次重构数据结构的期间，常常会用到当前序列或树的值。所以我们一般需要用平衡树或动态树维护序列或树的形态，在修改的时候直接操作，重构的时候直接用。

分块重建因为结合的是暴力做法，因此思维复杂度和代码复杂度都不高，常数也很小，有时甚至比正解还快。

例 5 带插入区间 k 小值[12]

维护一个序列，每次有 3 种操作：修改一个数的值、在一个位置前面插入一个数，询问区间第 k 小的值。

序列最后长度 n≤70,000，询问个数 m≤175,000，所有权值≤70,000。时间限制 6s，内存 512M。

由于 n 和权值范围同阶，以下权值范围视为 n。

这题的标准做法[13]是不旋转的平衡树套线段树或平衡树，利用不旋转的性质维护权值线段树或平衡树，在不平衡时暴力重构。时间复杂度 $O((n+m)\log^2 n)$。但这么写常数就是一个问题（写 Splay 会 T），内存又是一个问题。出题人采用了引用计数才解决了内存问题。还有一种算法是划分树套权值平衡树，内存小了，难度也增加了。我写的替罪羊树套平衡树写的非常长，实际情况也不是很理想。

考虑分块重建。不带插入不带修改的区间 k 小使用可持久化权值线段树，O(nlogn)初始化，O(log n)单次询问的。而一个简单的平衡树或直接用 STL 里的 rope 就可以维护一个序列，我们可以结合这两项进行操作。

设 $H=m^{1/2}\log n$，将权值存到平衡树里。维护插入和修改操作的集合（数组即可）。插入的时候，在平衡树中插入，在集合中插入一个插入操作和一个修改操作（如果在 x 位置插入 y，则插入操作记录 x，修改操作记录 x 位置 y 的个数+1），并更新之前插入和修改操作的位置编号（参照第 3 节开头的做法），时间复杂度 O(H)。修改的时候，在平衡树中修改，并在集合中插入两个修改操作（如果把 x 位置改成 y，原来这个数是 a，则记录 x 位置 a 的个数-1，x 位置 y 的个数+1），时间复杂度 O(log n)。

询问 l,r 的时候，O(H)扫描插入集合，算出 l 和 r 在上次重建的时候的位置 L 和 R，然后将所有修改位置在 l 到 r 之间的修改提取出来存到一个集合 S 中。利用 R 和 L-1 的可持久化权值线段树二分，用 S 中修改后的值在左区间的修改更新左区间的个数，决定往左或往右走，然后从 S 中剔除修改后的值不在当前区间中修改即可。时间复杂度 O(Hlogn)，但实际上在后期从 S 中剔除了无用的修改后，时间只是略大于 O(H)。

每 H 个修改或插入操作后，清空修改和插入集合，利用平衡树重建可持久化线段树，时间复杂度 O(nlogn)。考虑总时间复杂度。总共 m/H 次重建，每次 nlogn，总共是 $O(nm^{1/2})$的。三种操作中询问最耗时间，复杂度是 $O(m^{1/2}\log^2 n)$

[12] Source：BZOJ3065

[13] http://vfleaking.blog.163.com/blog/static/174807634201312365981 8/

的，所以总复杂度是 $O((n+m\log^2 n)m^{1/2})$。实际操作中询问复杂度很小。

这个做法在 BZOJ 上跑了 23s，而替罪羊树的版本跑了 40s 左右。如果为了平衡复杂度把 H 设为 $m^{1/2}$，超时。

例 6 带 link 和 cut 的链上 k 大[14]

给出一棵 n 个点的 LCT，m 次操作，每次可以 link、cut 或询问两点之间链上 k 大。

如果是静态树，则可以一边 DFS 一边维护可持久化权值线段树。考虑每 H 次重建一次可持久化权值线段树。而如果我们在查询 x、y 之前进行了 H 次 link、cut 操作，则我们可以把链 x、y 拆成不超过 H 条原来树上的链，也就是 2H 棵可持久化线段树的和、差。利用陈立杰 2012 年集训队论文的方式，可以在这 2H 棵线段树上同时二分，以 O(Hlogn)的时间复杂度找到 k 大。

现在问题就变成怎么找这 H 条链了。注意到只有 link 操作会导致原来两条旧的链拼接起来，我们用一个标记 mark 表示这个点是否有被 link 过，在 link 的时候顺便标记，并把这个操作存到一个 set 中。在 LCT 上，Splay 维护的信息是“当前点的子树中是否存在 mark 了的节点”。对于每个询问，先在 LCT 上取出这条链的 Splay，暴力对 Splay 进行 dfs（只需要访问子树中有 mark 过节点的节点），把标记过的节点按顺序存到数组中。由于 mark 过的节点不超过 2H 个，其祖先（也就是我们 dfs 访问到的点）不会超过 Hlogn 个。对这些节点进行判断，如果数组中相邻的两个点是 link 起来的（通过 set 判断），则说明有两条拼接的链。这样 O(HlogH)次扫描过后，我们就能获得这些链，用上一段的方式解决问题了。

复杂度分析：修改复杂度 O(logn)，询问复杂度 O(Hlogn)。共重建 m/H 次，每次 O(nlogn)，总共 O(nmlogn/H)。总时间复杂度是(mHlogn+nmlogn/H)。当 H 取 $n^{1/2}$ 时，时间复杂度 $O(mn^{1/2}\log n)$。

4 总结

莫队和分块都是基于暴力的做法，具有容易想、容易写、常数小的特点。虽然应用范围不是很广，但我们也应该掌握，在分析题目解决方案的时候，可以多一种选择，

[14] 经典问题