分块算法及简单扩展

黄哲威

Peking University

2017 年 5 月 22 日

可能涉及的几个词语解释

1. 区间：数列中连续一段的元素
2. 区间操作：将某个区间 $[a, b]$ 的所有元素进行某种改动的操作
3. 块：我们将数列划分成若干个不相交的区间，每个区间称为一个块
4. **整块**：在一个区间操作时，完整包含于区间的块
5. **不完整的块**：在一个区间操作时，只有部分包含于区间的块，即区间左右端点所在的两个块

- **给出一个长为 n 的数列，以及 n 个操作，操作涉及区间加法，单点查值。**

- **给出一个长为 n 的数列，以及 n 个操作，操作涉及区间加法，单点查值。**
- 以下默认 n 是 10^5 级别的数。
- 这是一道能用许多数据结构优化的经典题，可以用于不同数据结构训练。比如线段树：只要操作与询问满足区间信息能够快速合并，直接用线段树就能达到比分块更优的复杂度。
- 让我们来熟悉一下：数列分块就是把数列中每 m 个元素打包起来，达到优化算法的目的。

- 以此题为例，如果我们把每 m 个元素分为一块，共有 $\frac{n}{m}$ 块，每次**区间加的操作会涉及 $O(\frac{n}{m})$ 个整块，以及区间两侧两个不完整的块中至多 $2m$ 个元素。**
- 我们给每个块设置一个加法标记（就是记录这个块中元素一起加了多少），每次操作对每个整块直接 $O(1)$ 标记，而不完整的块由于元素比较少，暴力修改元素的值。
- 每次询问时返回元素的值加上其所在块的加法标记。

- 以此题为例，如果我们把每 m 个元素分为一块，共有 $\frac{n}{m}$ 块，每次**区间加的操作会涉及 $O(\frac{n}{m})$ 个整块，以及区间两侧两个不完整的块中至多 $2m$ 个元素。**
- 我们给每个块设置一个加法标记（就是记录这个块中元素一起加了多少），每次操作对每个整块直接 $O(1)$ 标记，而不完整的块由于元素比较少，暴力修改元素的值。
- 每次询问时返回元素的值加上其所在块的加法标记。
- 这样的总复杂度是 $O(n\frac{n}{m} + nm)$，根据均值不等式，$n\frac{n}{m} + nm \geq 2n\sqrt{n}$，当 m 取 $\sqrt{n}$ 时取等，总复杂度最低。

- 给出一个长为 n 的数列，以及 n 个操作，操作涉及区间加法，询问区间内小于某个值 x 的元素个数。

- **给出一个长为 n 的数列，以及 n 个操作，操作涉及区间加法，询问区间内小于某个值 x 的元素个数。**
- 有了上一题的经验，我们可以发现，数列简单分块问题实际上有三项东西要我们思考，对于每次区间操作：
    1. **不完整的块**的 $O(\sqrt{n})$ 个元素怎么处理？

- **给出一个长为 n 的数列，以及 n 个操作，操作涉及区间加法，询问区间内小于某个值 x 的元素个数。**
- 有了上一题的经验，我们可以发现，数列简单分块问题实际上有三项东西要我们思考，对于每次区间操作：
  1. **不完整的块**的 $O(\sqrt{n})$ 个元素怎么处理？
  2. $O(\sqrt{n})$ 个**整块**怎么处理？

- **给出一个长为 n 的数列，以及 n 个操作，操作涉及区间加法，询问区间内小于某个值 x 的元素个数。**
- 有了上一题的经验，我们可以发现，数列简单分块问题实际上有三项东西要我们思考，对于每次区间操作：
  1. **不完整的块**的 $O(\sqrt{n})$ 个元素怎么处理？
  2. $O(\sqrt{n})$ 个**整块**怎么处理？
  3. 要预处理什么信息（复杂度不能超过后面的操作）？

分块入门 2

- 我们先来思考只有询问操作的情况
  1. 不完整的块枚举统计即可
  2. 要在每个整块内寻找小于 x 的元素数量：**不得不**要求块内元素是有序的，这样就能使用二分法对块内查询，需要预处理时每块做一遍排序，复杂度 $O(n\log\frac{n}{m})$，每次查询在 $\frac{n}{m}$ 个块内二分，以及暴力 $2m$ 个元素。
- 复杂度是 $O(n\frac{n}{m}\log\frac{n}{m}+nm)$
- 如果 $m=\sqrt{n}$ 的话，总复杂度是 $O(n\sqrt{n}\log n)$，实际测试时 $m=2\sqrt{n}$ 的效果要好一点。

分块的调试检测技巧

- 一般来说，m 的取值有这么几种：$C \cdot \sqrt{n}$，$C \cdot \sqrt{n}/logn$，其中 $C = \{0.5, 1, 2, 3\}$。
- 可以生成一些大数据，然后用两份分块大小不同的代码来对拍，还可以根据运行时间尝试调整分块大小，减小常数。

- 那么区间加怎么办呢?

- 那么区间加怎么办呢?
- 套用第一题的方法，维护一个加法标记 tag，略有区别的地方在于，不完整的块修改后可能会使得该块内数字乱序，所以头尾两个不完整块需要重新排序，复杂度分析略。
- 在加法标记下的询问操作，块外还是暴力，查询小于 $x - tag$ 的元素个数，块内用 $x - tag$ 作为二分的值即可。
- 总复杂度依旧是 $O(n\sqrt{n}\log n)$

- 给出一个长为 n 的数列，以及 n 个操作，操作涉及区间加法，询问区间内小于某个值 x 的前驱（比其小的最大元素）。

- **给出一个长为 n 的数列，以及 n 个操作，操作涉及区间加法，询问区间内小于某个值 x 的前驱（比其小的最大元素）。**
- 接着第二题的解法，其实只要把块内查询的二分稍作修改即可。
- 这题其实有一个启发意义：可以在块内维护其它结构使其更具有拓展性，比如放一个 *set*，这样如果还有插入、删除元素的操作，会更加的方便。
- 时间复杂度不变。代码复杂度降低不少。

- **给出一个长为 n 的数列，以及 n 个操作，操作涉及区间加法，区间求和。**

- **给出一个长为 n 的数列，以及 n 个操作，操作涉及区间加法，区间求和。**
- 这题的询问变成了区间上的询问：不完整的块还是暴力；而要想快速统计完整块的答案，需要维护每个块的元素和，要预处理一下。
- 考虑区间修改操作，不完整的块直接改，顺便更新块的元素和；完整的块类似之前标记的做法，直接根据块的元素和所加的值计算元素和的增量。
- 当然线段树可以直接做。

- 给出一个长为 n 的数列，以及 n 个操作，操作涉及区间开方，区间求和。

- **给出一个长为 n 的数列，以及 n 个操作，操作涉及区间开方，区间求和。**
- 稍作思考可以发现，开方操作比较棘手，主要是对于整块开方时，必须要知道每一个元素，才能知道他们开方后的和，也就是说，难以快速对一个块信息进行更新。
- 看来我们要另辟蹊径。不难发现，这题的修改就只有下取整开方，而一个数经过几次开方之后，它的值就会变成 0 或者 1。

- 如果每次区间开方只不涉及完整的块，意味着不超过 $2\sqrt{n}$ 个元素，直接暴力即可。
- 如果涉及了一些完整的块，这些块经过几次操作以后就会都变成 $0/1$，于是我们采取一种分块优化的暴力做法，只要每个整块暴力开方后，记录一下元素是否都变成了 $0/1$，区间修改时跳过那些全为 $0/1$ 的块即可。
- 这样每个元素至多被开方不超过 4 次，显然复杂度没有问题。

- 如果每次区间开方只不涉及完整的块，意味着不超过 $2\sqrt{n}$ 个元素，直接暴力即可。
- 如果涉及了一些完整的块，这些块经过几次操作以后就会都变成 0/1，于是我们采取一种分块优化的暴力做法，只要每个整块暴力开方后，记录一下元素是否都变成了 0/1，区间修改时跳过那些全为 0/1 的块即可。
- 这样每个元素至多被开方不超过 4 次，显然复杂度没有问题。
- 当然线段树可以直接做。

- 给出一个长为 n 的数列，以及 n 个操作，操作涉及单点插入，单点询问，数据随机生成。

- **给出一个长为 n 的数列，以及 n 个操作，操作涉及单点插入，单点询问，数据随机生成。**
- 先说随机数据的情况

- **给出一个长为 n 的数列，以及 n 个操作，操作涉及单点插入，单点询问，数据随机生成。**
- 先说随机数据的情况
- 之前提到过，如果我们块内用数组以外的数据结构，能够支持其它不一样的操作，比如此题每块内可以放一个动态的数组，每次插入时先找到位置所在的块，再暴力插入，把块内的其它元素直接向后移动一位，当然用链表也是可以的。

- **给出一个长为 n 的数列，以及 n 个操作，操作涉及单点插入，单点询问，数据随机生成。**
- 先说随机数据的情况
- 之前提到过，如果我们块内用数组以外的数据结构，能够支持其它不一样的操作，比如此题每块内可以放一个动态的数组，每次插入时先找到位置所在的块，再暴力插入，把块内的其它元素直接向后移动一位，当然用链表也是可以的。
- 查询的时候类似，复杂度分析略。

- 但是这样做有个问题，如果数据不随机怎么办？

- **但是这样做有个问题，如果数据不随机怎么办?**
- 如果先在一个块有大量单点插入，这个块的大小会大大超过 $\sqrt{n}$，那块内的暴力就没有复杂度保证了。

- **但是这样做有个问题，如果数据不随机怎么办?**
- 如果先在一个块有大量单点插入，这个块的大小会大大超过 $\sqrt{n}$，那块内的暴力就没有复杂度保证了。
- 还需要引入一个操作：重新分块（重构
- 每 $\sqrt{n}$ 次插入后，重新把数列平均分一下块，重构需要的复杂度为 $O(n)$，重构的次数为 $\sqrt{n}$，所以重构的复杂度没有问题，而且保证了每个块的大小相对均衡。

- **但是这样做有个问题，如果数据不随机怎么办?**
- 如果先在一个块有大量单点插入，这个块的大小会大大超过 $\sqrt{n}$，那块内的暴力就没有复杂度保证了。
- 还需要引入一个操作：重新分块（重构
- 每 $\sqrt{n}$ 次插入后，重新把数列平均分一下块，重构需要的复杂度为 $O(n)$，重构的次数为 $\sqrt{n}$，所以重构的复杂度没有问题，而且保证了每个块的大小相对均衡。
- 当然，也可以当某个块过大时重构，或者只把这个块分成两半。

- **但是这样做有个问题，如果数据不随机怎么办？**
- 如果先在一个块有大量单点插入，这个块的大小会大大超过 $\sqrt{n}$，那块内的暴力就没有复杂度保证了。
- 还需要引入一个操作：重新分块（重构
- 每 $\sqrt{n}$ 次插入后，重新把数列平均分一下块，重构需要的复杂度为 $O(n)$，重构的次数为 $\sqrt{n}$，所以重构的复杂度没有问题，而且保证了每个块的大小相对均衡。
- 当然，也可以当某个块过大时重构，或者只把这个块分成两半。
- 当然 Splay 可以直接做。

- **给出一个长为 n 的数列，以及 n 个操作，操作涉及区间乘法，区间加法，单点询问。**

- **给出一个长为 n 的数列，以及 n 个操作，操作涉及区间乘法，区间加法，单点询问。**
- 很显然，如果只有区间乘法，和分块入门 1 的做法没有本质区别，但要思考如何同时维护两种标记。
- 我们让乘法标记的优先级高于加法（如果反过来的话，新的加法标记无法处理）
- 若当前的一个块乘以 m_1 后加上 a_1，这时进行一个乘 m_2 的操作，则原来的标记变成 $(m_1 m_2, a_1 m_2)$
- 若当前的一个块乘以 m_1 后加上 a_1，这时进行一个加 a_2 的操作，则原来的标记变成 $(m_1, a_1 + a_2)$

- **给出一个长为 n 的数列，以及 n 个操作，操作涉及区间乘法，区间加法，单点询问。**
- 很显然，如果只有区间乘法，和分块入门 1 的做法没有本质区别，但要思考如何同时维护两种标记。
- 我们让乘法标记的优先级高于加法（如果反过来的话，新的加法标记无法处理）
- 若当前的一个块乘以 m_1 后加上 a_1，这时进行一个乘 m_2 的操作，则原来的标记变成 (m_1m_2, a_1m_2)
- 若当前的一个块乘以 m_1 后加上 a_1，这时进行一个加 a_2 的操作，则原来的标记变成 $(m_1, a_1 + a_2)$
- 当然线段树可以直接做（行星序列）。

- **给出一个长为 n 的数列，以及 n 个操作，操作涉及区间询问等于一个数 c 的元素，并将这个区间的所有元素改为 c。**

- **给出一个长为 n 的数列，以及 n 个操作，操作涉及区间询问等于一个数 c 的元素，并将这个区间的所有元素改为 c。**
- 区间修改没有什么难度，这题难在区间查询比较奇怪，因为权值种类比较多，似乎没有什么好的维护方法。

- **给出一个长为 n 的数列，以及 n 个操作，操作涉及区间询问等于一个数 c 的元素，并将这个区间的所有元素改为 c。**
- 区间修改没有什么难度，这题难在区间查询比较奇怪，因为权值种类比较多，似乎没有什么好的维护方法。
- 模拟一些数据可以发现，询问后一整段都会被修改，几次询问后数列可能只剩下几段不同的区间了。类似刚才开根号那题。

- **给出一个长为 n 的数列，以及 n 个操作，操作涉及区间询问等于一个数 c 的元素，并将这个区间的所有元素改为 c。**
- 区间修改没有什么难度，这题难在区间查询比较奇怪，因为权值种类比较多，似乎没有什么好的维护方法。
- 模拟一些数据可以发现，询问后一整段都会被修改，几次询问后数列可能只剩下几段不同的区间了。类似刚才开根号那题。
- 我们思考这样一个暴力，还是分块，维护每个块是否只有一种权值，区间操作的时候，对于同权值的一个块就 $O(1)$ 统计答案，否则暴力统计答案，并修改标记，不完整的块也暴力。

- 这样看似最差情况每次都会耗费 $O(n)$ 的时间，但其实可以这样分析：
- 假设初始序列都是同一个值，那么查询是 $O(\sqrt{n})$
- 如果这时进行一个区间操作，它最多破坏首尾 2 个块的标记，所以只能使后面的询问至多多 2 个块的暴力时间，所以均摊每次操作复杂度还是 $O(\sqrt{n})$。

- 这样看似最差情况每次都会耗费 $O(n)$ 的时间，但其实可以这样分析：
- 假设初始序列都是同一个值，那么查询是 $O(\sqrt{n})$
- 如果这时进行一个区间操作，它最多破坏首尾 2 个块的标记，所以只能使后面的询问至多多 2 个块的暴力时间，所以均摊每次操作复杂度还是 $O(\sqrt{n})$。
- 换句话说，要想让一个操作耗费 $O(n)$ 的时间，要先花费 $\sqrt{n}$ 个操作对数列进行修改。
- 初始序列不同值，经过类似分析后，就可以放心的暴力啦。

- 给出一个长为 n 的数列，以及 n 个操作，操作涉及询问区间内多少个数出现正偶数次。

- **给出一个长为 n 的数列，以及 n 个操作，操作涉及询问区间内多少个数出现正偶数次。**
- 这是一道类似区间众数的经典题，区间众数可参考 WJMZBMR(膜) 的《区间众数解题报告》。

- **给出一个长为 n 的数列，以及 n 个操作，操作涉及询问区间内多少个数出现正偶数次。**
- 这是一道类似区间众数的经典题，区间众数可参考 WJMZBMR(膜) 的《区间众数解题报告》。

- 所以我们可以预处理 f(i,j) 表示第 i 块到第 j 块的答案（枚举 i 开个桶扫一遍）

- 所以我们可以预处理 f(i,j) 表示第 i 块到第 j 块的答案（枚举 i 开个桶扫一遍）
- 对于一个询问 [l,r]，中间包含在完整块内的数 [x,y] 答案已经得到，考虑不完整的块中每个数对答案的影响

- 所以我们可以预处理 f(i,j) 表示第 i 块到第 j 块的答案（枚举 i 开个桶扫一遍）
- 对于一个询问 [l,r]，中间包含在完整块内的数 [x,y] 答案已经得到，考虑不完整的块中每个数对答案的影响
- 若能快速得出一个数在某个区间内出现次数，每次只要再求 $2\sqrt{n}+1$ 个元素在 [l,r] 和 [x,y] 的出现次数，这题就解决了。

- 所以我们可以预处理 f(i,j) 表示第 i 块到第 j 块的答案（枚举 i 开个桶扫一遍）
- 对于一个询问 [l,r]，中间包含在完整块内的数 [x,y] 答案已经得到，考虑不完整的块中每个数对答案的影响
- 若能快速得出一个数在某个区间内出现次数，每次只要再求 $2\sqrt{n}+1$ 个元素在 [l,r] 和 [x,y] 的出现次数，这题就解决了。
- 由于没有修改，只要离散化以后，给每个数 x 开个 vector，按顺序存下 x 出现的位置，每次询问 x 时把区间的左右端点放进对应 vector 二分一下即可。
- 根据均值不等式，可以算出分块大小大概是 $\sqrt{n/logn}$

- 以上是数列分块的基本方法，将数列中一些关联的信息整合起来，是很多数据结构都会运用到的思路。

- 以上是数列分块的基本方法，将数列中一些关联的信息整合起来，是很多数据结构都会运用到的思路。
- 这种思维方式可以运用到其它类型的问题中。

- 以上是数列分块的基本方法，将数列中一些关联的信息整合起来，是很多数据结构都会运用到的思路。
- 这种思维方式可以运用到其它类型的问题中。
- 例如对询问区间分块，对数字权值分块，对平面中的点分块，对树进行分块等等。

1. 数列分块入门
2. 中场休息
   - BZOJ 圈地
3. 莫队算法
4. 树上分块
5. 参考文献

- 2 维平面上有 n 个点，取 3 个点使得形成的三角形的面积最小。
- 最小面积可以是 0。$n \leq 1000$

- 2 维平面上有 n 个点，取 3 个点使得形成的三角形的面积最小。
- 最小面积可以是 0。$n \leq 1000$
- 暴力 n^3 不多说。但是没什么用。

- 2 维平面上有 n 个点，取 3 个点使得形成的三角形的面积最小。
- 最小面积可以是 0。$n \le 1000$
- 暴力 n^3 不多说。但是没什么用。
- 把这些点分成 $\sqrt{n}$ 块，块内暴力，轻松愉快!（不一定非要 $\sqrt{n}$)
- 这样做不靠谱，所以我们可以随机旋转坐标系，rand 个四五十次就可以把这题水过了 OwO。

1. 数列分块入门
2. 中场休息
3. 莫队算法
   - 分块入门 9 离线版
4. 树上分块
5. 参考文献

- 一些信息维护的问题，允许使用离线算法，维护的信息不支持区间的快速合并，比如询问区间内不同数字的个数。

- 一些信息维护的问题，允许使用离线算法，维护的信息不支持区间的快速合并，比如询问区间内不同数字的个数。
- 这类问题是线段树等数据结构难以胜任的，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 $O(n)$ 的（数组维护出现次数，记录不同数字的个数）。

- 一些信息维护的问题，允许使用离线算法，维护的信息不支持区间的快速合并，比如询问区间内不同数字的个数。
- 这类问题是线段树等数据结构难以胜任的，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 $O(n)$ 的（数组维护出现次数，记录不同数字的个数）。
- 但这种问题数据支持单点增量——如果已经有一个包含 $num[l, r-1]$ 的所有数字的出现次数的数组 cnt，不同数字的个数 ans，则只需要令 $cnt[num[r]]+1$，并判断其是否为 1 即可。

- 一些信息维护的问题，允许使用离线算法，维护的信息不支持区间的快速合并，比如询问区间内不同数字的个数。
- 这类问题是线段树等数据结构难以胜任的，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 $O(n)$ 的（数组维护出现次数，记录不同数字的个数）。
- 但这种问题数据支持单点增量——如果已经有一个包含 $num[l, r-1]$ 的所有数字的出现次数的数组 cnt，不同数字的个数 ans，则只需要令 $cnt[num[r]]+1$，并判断其是否为 1 即可。
- 同理，单点减量也可行。这样的问题适用于莫队算法。

- 莫队算法的核心就是，对于两个询问 [a,b][c,d]

- 莫队算法的核心就是，对于两个询问 [a,b][c,d]
- 设 x 次合并的时间为 $f(x)$，可以以 $f(|a-c|+|b-d|)$ 的时间从 [a,b] 的数据转移到 [c,d]

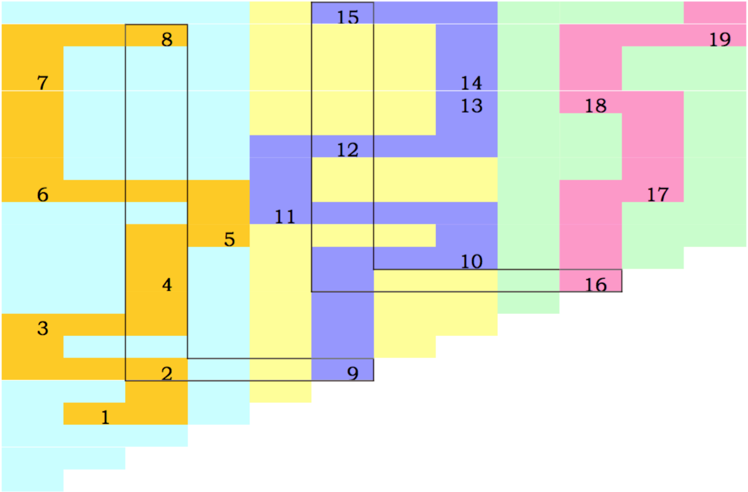
- 莫队算法的核心就是，对于两个询问 [a,b][c,d]
- 设 x 次合并的时间为 $f(x)$，可以以 $f(|a-c|+|b-d|)$ 的时间从 [a,b] 的数据转移到 [c,d]
- 如果把询问看成二维平面上的点，转移的时间与曼哈顿距离有关，按照一定顺序访问这些点，就能回答所有询问

- 莫队算法的核心就是，对于两个询问 [a,b][c,d]
- 设 x 次合并的时间为 $f(x)$，可以以 $f(|a-c|+|b-d|)$ 的时间从 [a,b] 的数据转移到 [c,d]
- 如果把询问看成二维平面上的点，转移的时间与曼哈顿距离有关，按照一定顺序访问这些点，就能回答所有询问
- 先离线所有询问，问题就转为求平面上所有点的最短曼哈顿距离的哈密顿路径

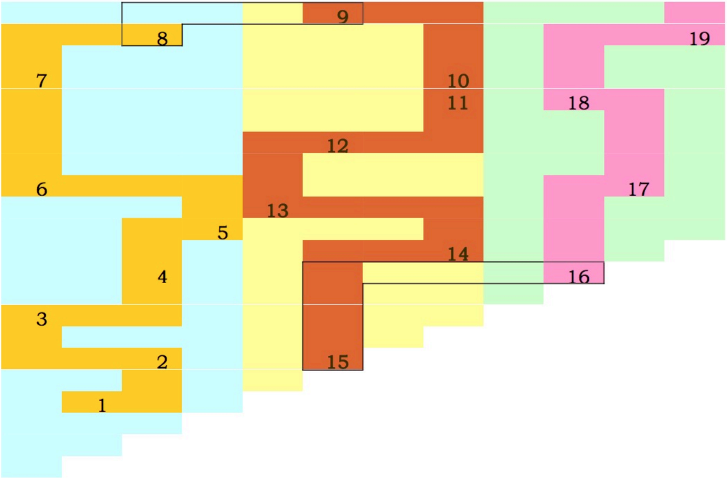
- 莫队算法的核心就是，对于两个询问 [a,b][c,d]
- 设 x 次合并的时间为 $f(x)$，可以以 $f(|a-c|+|b-d|)$ 的时间从 [a,b] 的数据转移到 [c,d]
- 如果把询问看成二维平面上的点，转移的时间与曼哈顿距离有关，按照一定顺序访问这些点，就能回答所有询问
- 先离线所有询问，问题就转为求平面上所有点的最短曼哈顿距离的哈密顿路径
- 这个问题难以解决，但我们可以选择一个替代品：对询问端点分块

- 莫队算法的核心就是，对于两个询问 [a,b][c,d]
- 设 x 次合并的时间为 $f(x)$，可以以 $f(|a-c|+|b-d|)$ 的时间从 [a,b] 的数据转移到 [c,d]
- 如果把询问看成二维平面上的点，转移的时间与曼哈顿距离有关，按照一定顺序访问这些点，就能回答所有询问
- 先离线所有询问，问题就转为求平面上所有点的最短曼哈顿距离的哈密顿路径
- 这个问题难以解决，但我们可以选择一个替代品：对询问端点分块
- 设块的大小为 H，对每个询问 [a,b] 以 block[a] 为第一关键字，b 为第二关键字

- 莫队算法的核心就是，对于两个询问 [a,b][c,d]
- 设 x 次合并的时间为 $f(x)$，可以以 $f(|a-c|+|b-d|)$ 的时间从 [a,b] 的数据转移到 [c,d]
- 如果把询问看成二维平面上的点，转移的时间与曼哈顿距离有关，按照一定顺序访问这些点，就能回答所有询问
- 先离线所有询问，问题就转为求平面上所有点的最短曼哈顿距离的哈密顿路径
- 这个问题难以解决，但我们可以选择一个替代品：对询问端点分块
- 设块的大小为 H，对每个询问 [a,b] 以 block[a] 为第一关键字，b 为第二关键字



- 用另一种颜色表示块内处理路径，框框表示块切换时的路径



- 交替对块的纵坐标升序降序排列，可以优化常数，复杂度分析略

- 离线的话有很多做法，这里顺便介绍一种奇怪的分块，也被称为“大小分治算法”

- 离线的话有很多做法，这里顺便介绍一种奇怪的分块，也被称为“大小分治算法”
- 按询问左端点排序处理，维护右端点在 1-x 的答案

- 离线的话有很多做法，这里顺便介绍一种奇怪的分块，也被称为“大小分治算法”
- 按询问左端点排序处理，维护右端点在 1-x 的答案
- 每次考虑删除左端点的元素，若把数列分段，每出现一个和左端点相同的值就划分一段，右端点的答案是一段 +1，一段-1 的...

- 离线的话有很多做法，这里顺便介绍一种奇怪的分块，也被称为“大小分治算法”
- 按询问左端点排序处理，维护右端点在 1-x 的答案
- 每次考虑删除左端点的元素，若把数列分段，每出现一个和左端点相同的值就划分一段，右端点的答案是一段 +1，一段-1 的...
- 考虑用树状数组暴力维护，每次复杂度是左端点权值出现次数 * logn

- 离线的话有很多做法，这里顺便介绍一种奇怪的分块，也被称为“大小分治算法”
- 按询问左端点排序处理，维护右端点在 1-x 的答案
- 每次考虑删除左端点的元素，若把数列分段，每出现一个和左端点相同的值就划分一段，右端点的答案是一段 +1，一段-1 的...
- 考虑用树状数组暴力维护，每次复杂度是左端点权值出现次数 * logn

- 再考虑优化这个暴力

- 再考虑优化这个暴力
- 以 $M = \sqrt{n}$ 为界，出现次数大于 M 的权值不超过 M 种

- 再考虑优化这个暴力
- 以 $M = \sqrt{n}$ 为界，出现次数大于 M 的权值不超过 M 种
- 出现次数小于 M 的依然用暴力统计答案，由于出现次数少，不会退化
- 出现次数大于 M 的数，每个数维护一个树状数组，每次询问用这些树状数组得出每个数在区间内的出现次数

- 再考虑优化这个暴力
- 以 $M = \sqrt{n}$ 为界，出现次数大于 M 的权值不超过 M 种
- 出现次数小于 M 的依然用暴力统计答案，由于出现次数少，不会退化
- 出现次数大于 M 的数，每个数维护一个树状数组，每次询问用这些树状数组得出每个数在区间内的出现次数
- 复杂度类似分块

- 再考虑优化这个暴力
- 以 $M = \sqrt{n}$ 为界，出现次数大于 M 的权值不超过 M 种
- 出现次数小于 M 的依然用暴力统计答案，由于出现次数少，不会退化
- 出现次数大于 M 的数，每个数维护一个树状数组，每次询问用这些树状数组得出每个数在区间内的出现次数
- 复杂度类似分块
- 显然用莫队算法可以轻松解决且常数很小

1. 数列分块入门
2. 中场休息
3. 莫队算法
4. 树上分块
   - BZOJ3720 Gty 的妹子树
   - BZOJ1086 王室联邦
5. 参考文献

- 近年来大家都喜欢上树搞事情
- 从树上倍增，树上博弈，树链剖分到点分治，树上后缀数组，树上斜率优化
- 序列问题花样上树 OwO

- 近年来大家都喜欢上树搞事情
- 从树上倍增，树上博弈，树链剖分到点分治，树上后缀数组，树上斜率优化
- 序列问题花样上树 OwO
- 这让我们不由陷入沉思。

- 近年来大家都喜欢上树搞事情
- 从树上倍增，树上博弈，树链剖分到点分治，树上后缀数组，树上斜率优化
- 序列问题花样上树 OwO
- 这让我们不由陷入沉思。

1. 数列分块入门
2. 中场休息
3. 莫队算法
4. 树上分块
   - BZOJ3720 Gty 的妹子树
   - BZOJ1086 王室联邦
5. 参考文献

- 维护一棵 n 个点的点权树，支持下列操作：
- 1. 询问某棵子树中有多少个节点的权值大于 x
- 2. 修改某个节点的权值
- 我会 dfs 序 + 树状数组 + 主席树!

- 维护一棵 n 个点的点权树，支持下列操作：
- 1. 询问某棵子树中有多少个节点的权值大于 x
- 2. 修改某个节点的权值
- 我会 dfs 序 + 树状数组 + 主席树!
- 3. 增加一个叶子节点
- 离线 dfs 序!

- 维护一棵 n 个点的点权树，支持下列操作：
- 1. 询问某棵子树中有多少个节点的权值大于 x
- 2. 修改某个节点的权值
- 我会 dfs 序 + 树状数组 + 主席树!
- 3. 增加一个叶子节点
- 离线 dfs 序!
- $n, m \leq 30000$ 强制在线!

- 维护一棵 n 个点的点权树，支持下列操作：
- 1. 询问某棵子树中有多少个节点的权值大于 x
- 2. 修改某个节点的权值
- 我会 dfs 序 + 树状数组 + 主席树!
- 3. 增加一个叶子节点
- 离线 dfs 序!
- $n, m \leq 30000$ 强制在线!

- 不强制在线的话线段树可解

- 不强制在线的话线段树可解
- 块状树!

- 不强制在线的话线段树可解
- 块状树!
- dfs，如果节点的父亲节点所在块未满，就塞进父节点所在块中，否则自成一块

- 不强制在线的话线段树可解
- 块状树!
- dfs，如果节点的父亲节点所在块未满，就塞进父节点所在块中，否则自成一块
- 每块内维护一个有序的数组，块与块之间连上边方便查询，查询分为一个不完整的块和若干整块，类似数列

- 不强制在线的话线段树可解
- 块状树!
- dfs，如果节点的父亲节点所在块未满，就塞进父节点所在块中，否则自成一块
- 每块内维护一个有序的数组，块与块之间连上边方便查询，查询分为一个不完整的块和若干整块，类似数列
- 修改和加点也十分容易

- 不强制在线的话线段树可解
- 块状树!
- dfs，如果节点的父亲节点所在块未满，就塞进父节点所在块中，否则自成一块
- 每块内维护一个有序的数组，块与块之间连上边方便查询，查询分为一个不完整的块和若干整块，类似数列
- 修改和加点也十分容易
- 可惜这种分块会被菊花图卡 QAQ

- 一个国家由 n 个城市组成一棵树，要将其划分为若干个省
- 每个省的大小为 $[B, 3B]$，每个省有一个省会（不一定要在省内）
- 使得每个省的所有城市到省会的简单路径上不能经过其它的省。

- 一个国家由 n 个城市组成一棵树，要将其划分为若干个省
- 每个省的大小为 $[B, 3B]$，每个省有一个省会（不一定要在省内）
- 使得每个省的所有城市到省会的简单路径上不能经过其它的省。
- $1 \leq n, B \leq 1000$

- 我们希望一棵树的分块连通且大小合适
- 但是对于菊花图，根本没有这样的效果

- 我们希望一棵树的分块连通且大小合适
- 但是对于菊花图，根本没有这样的效果
- 这题提供了一种做法，深搜时维护一个栈，每个点退出递归时压栈，自下至上进行合并
- 如果某棵子树深搜完之后栈内元素数大等 B 就把当前的栈内元素合并为一个块

- 我们希望一棵树的分块连通且大小合适
- 但是对于菊花图，根本没有这样的效果
- 这题提供了一种做法，深搜时维护一个栈，每个点退出递归时压栈，自下至上进行合并
- 如果某棵子树深搜完之后栈内元素数大等 B 就把当前的栈内元素合并为一个块
- 但若某棵子树深搜之后不到 B 可能在深搜下一个子树内部的某个位置超过 B，这样会导致分块不连通

- 我们希望一棵树的分块连通且大小合适
- 但是对于菊花图，根本没有这样的效果
- 这题提供了一种做法，深搜时维护一个栈，每个点退出递归时压栈，自下至上进行合并
- 如果某棵子树深搜完之后栈内元素数大等 B 就把当前的栈内元素合并为一个块
- 但若某棵子树深搜之后不到 B 可能在深搜下一个子树内部的某个位置超过 B，这样会导致分块不连通
- 在每次进入递归时维护一个当前栈底，该栈底以下的元素不能弹栈

- 我们希望一棵树的分块连通且大小合适
- 但是对于菊花图，根本没有这样的效果
- 这题提供了一种做法，深搜时维护一个栈，每个点退出递归时压栈，自下至上进行合并
- 如果某棵子树深搜完之后栈内元素数大等 B 就把当前的栈内元素合并为一个块
- 但若某棵子树深搜之后不到 B 可能在深搜下一个子树内部的某个位置超过 B，这样会导致分块不连通
- 在每次进入递归时维护一个当前栈底，该栈底以下的元素不能弹栈
- 这样会使得分块大小介于 $[B, 2B]$，题目中给了 3B 是为了处理分完块后剩下来不足 B 的点

- 于是我们就学会了块状树

- 于是我们就学会了块状树
- 刚才提到，序列上的询问可以容易的转为平面上的点，所以比较容易运用莫队算法

- 于是我们就学会了块状树
- 刚才提到，序列上的询问可以容易的转为平面上的点，所以比较容易运用莫队算法
- 那树上的询问呢？

- 于是我们就学会了块状树
- 刚才提到，序列上的询问可以容易的转为平面上的点，所以比较容易运用莫队算法
- 那树上的询问呢？
- 可以用刚才的树分块方式来对询问进行排序，然后就变成树上莫队辣！

- 于是我们就学会了块状树
- 刚才提到，序列上的询问可以容易的转为平面上的点，所以比较容易运用莫队算法
- 那树上的询问呢?
- 可以用刚才的树分块方式来对询问进行排序，然后就变成树上莫队辣!
- 或者按 dfs 的时间戳来划分，复杂度也是有保证的

- 于是我们就学会了块状树
- 刚才提到，序列上的询问可以容易的转为平面上的点，所以比较容易运用莫队算法
- 那树上的询问呢？
- 可以用刚才的树分块方式来对询问进行排序，然后就变成树上莫队辣！
- 或者按 dfs 的时间戳来划分，复杂度也是有保证的
- 通常可以把树上的路径拆成两条链 (偷懒)

- 于是我们就学会了块状树
- 刚才提到，序列上的询问可以容易的转为平面上的点，所以比较容易运用莫队算法
- 那树上的询问呢?
- 可以用刚才的树分块方式来对询问进行排序，然后就变成树上莫队辣!
- 或者按 dfs 的时间戳来划分，复杂度也是有保证的
- 通常可以把树上的路径拆成两条链 (偷懒)
- 可以再脑补一下把树上的一条路径一步一步转为另一条路径，有兴趣的大佬可以复习一下糖果公园

- 罗剑桥《浅谈分块思想在一类数据处理问题中的应用》
- 陈俊琨《莫队分块算法研究》
- 陈立杰《区间众数解题报告》
- 翁家翌《如何缩代码、如何港记、如何骗分以及如何进队》
- PoPoQQQ kuribohG vfleaking 的博客