算法竞赛中的分块思想

杭州学军中学　谷晟

信息学竞赛中经常会有这类问题

初学者通常使用 for 循环遍历来解决大量数据的问题
直到...

- 操作# L R：对第 L 至第 R 个数每个数进行*&^#^&$*!@
- 操作* L R：输出第 L 至第 R 个数的*#&(*@&$(*&@#$(@
- $N \leq 10^5$
- $Q \leq 10^5$

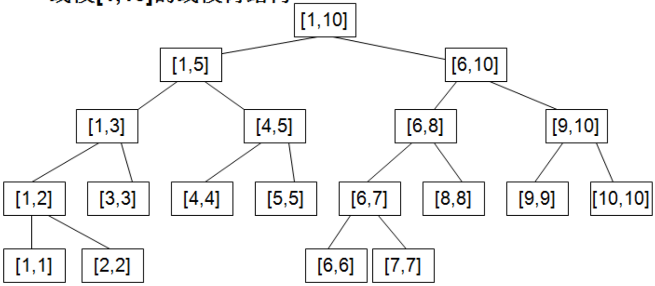
遍历操作，复杂度至少 $O(QN)$，无法承受。
怎么办?

多次区间操作的问题的解决方法

- 数据结构
  树状数组，线段树，平衡树，etc.
- 分块
  这是什么?

常用数据结构-线段树（例）

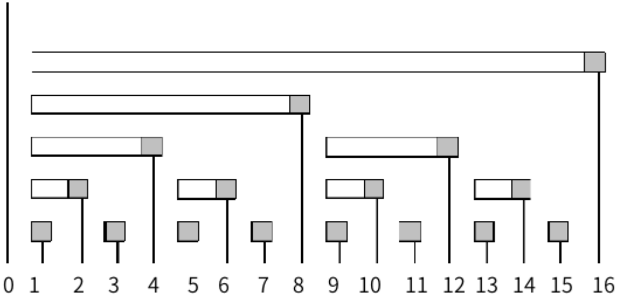
线段[1,10]的线段树结构



常用数据结构-线段树

- 可以维护可相加（合并）的区间信息
  可相加（合并）：知道左右两半的信息就能算出整体的信息
  例子：权值和，最大值，最小值
  反例：中位数，众数
- 单次操作复杂度 $O(\log N)$
- 单点询问，单点修改，区间询问，区间修改（需要标记下传）
- 权值线段树：维护集合，可以查询第 k 大（小）数/有几个数比 x 大（小）
- 可持久化：修改过之后可以访问旧版本。
  “主席树”：像前缀和一样，对原序列的每个前缀存一个线段树版本

常用数据结构-树状数组（例）



$$C[i] = A[i - lowbit(i) + 1] + \cdots + A[i]$$

常用数据结构-树状数组

- 可以维护可相加、相减的区间信息
  例如：区间和。上面提到的最大（小）值就不行了
- 单次操作复杂度 $O(\log N)$
- 单点修改，前缀和查询
- 和线段树相比，代码简单，常数小，占空间小

常用数据结构-平衡树

- 维护集合：插入删除，查询某个数，查询排名，查询第 k 大
- 维护序列：与线段树相比，还支持在中间插入、删除元素，部分实现支持区间分裂、合并、反转
- 单次操作复杂度 $O(\log N)$（与实现有关：严格、均摊、期望）
- 常用实现：Treap,Splay,SBT, 替罪羊树
- 部分实现能可持久化

分块?

通常我们可以用 $O(1)$ 的时间进行单点操作或全体操作，区间操作却比较麻烦。
分块巧妙地将两者结合起来，实现了区间操作：

- 将原序列每连续 B 个元素分为一 “块”
- 每个操作区间 $[L, R]$
  一定是类似 “[L. 块的一部分]-[块]-...-[块]-[块的一部分.R]”
  的形式
- 对于中间完整的块，进行整体操作。对于两侧两个块，对受影响的元素暴力单点操作

如果整体操作和单点操作的复杂度都是 $O(1)$，则

- 整体操作次数不超过块的总数，复杂度 $O(\frac{N}{B})$
- 单点操作最多只影响两个块，复杂度 $O(B)$

取 $B = \sqrt{N}$ 时，一次区间操作的复杂度为 $O(\sqrt{N})$，达到最优。
如果复杂度公式较为复杂，也可以通过实验确定最优块大小。

简单例子

有 N 个数，Q 个操作，操作有以下两种：

- 操作1 L R x：将第 L 到第 R 个数每个都加 x
- 操作2 L R：输出第 L 到第 R 个数的和

$N, Q \leq 100000$

简单例子-解答

数据结构做法：直接使用线段树维护序列。复杂度
$O(N + Q\log N)$
分块做法：

- 每 $\sqrt{N}$ 个数分为一个块，每个块维护块内所有数之和。
- 更新时，对于中间的完整的块，把增加值记录在块上；两端两个块暴力更新
- 询问时，两端暴力遍历，中间完整的块直接使用已经维护好的“块内所有数之和”
- 复杂度 $O(N + Q\sqrt{N})$

复杂度太高?

刚才的题目中，分块做法的复杂度明显比数据结构做法高。是否说明分块比不上高级数据结构呢?

BZOJ3065 带插入区间 K 小值

第一行一个正整数 n，表示原来有 n 只跳蚤排成一行做早操。
第二行有 n 个用空格隔开的非负整数，从左至右代表每只跳蚤的弹跳力。
第三行一个正整数 q，表示下面有多少个操作。
下面一共 q 行，对原序列的操作：（假设此时一共 m 只跳蚤）

1. Q x y k：询问从左至右第 x 只跳蚤到从左至右第 y 只跳蚤中，弹跳力第 k 小的跳蚤的弹跳力是多少。
   $(1 \le x \le y \le m, 1 \le k \le y - x + 1)$
2. M x val：将从左至右第 x 只跳蚤的弹跳力改为 val。
   $(1 \le x \le m)$
3. I x val：在从左至右第 x 只跳蚤的前面插入一只弹跳力为 val 的跳蚤。即插入后从左至右第 x 只跳蚤是我刚插入的跳蚤。
   $(1 \le x \le m + 1)$

操作中输入的整数都要异或上一次询问的结果（没有上一次询问则为 0）进行解码（**强制在线**）。
原序列长度 ≤ 35000，插入个数 ≤ 35000，修改个数 ≤ 70000，查询个数 ≤ 70000，$0 \le$ 每时每刻权值 ≤ 70000

BZOJ3065 带插入区间 K 小值

1. `Q x y k`: 询问区间 $[x, y]$ 内第 k 大。
   $(1 \le x \le y \le m, 1 \le k \le y - x + 1)$
2. `M x val`: 第 x 个数改为 val。$(1 \le x \le m)$
3. `I x val`: 在第 x 个数之前插入一个 val。$(1 \le x \le m + 1)$

操作中输入的整数都要异或上一次询问的结果进行解码（**强制在线**）。
原序列长度 ≤ 35000，插入个数 ≤ 35000，修改个数 ≤ 70000，查询个数 ≤ 70000,$0 \le$ 每时每刻权值 ≤ 70000
不会做？可尝试解决以下较简单的情况：

- 没有修改和插入
- 没有插入
- 不要求强制在线

BZOJ3065 数据结构做法

3065: 带插入区间K小值 系列题解之一——替罪羊套函数式线段树

3065: 带插入区间K小值 系列题解之二——Treap套函数式线段树

3065: 带插入区间K小值 系列题解之三——划分树崛起

【朝鲜树套函数式线段树】(TLE)

Problem 3065 >> B树做法

[redacted] @ 2014-05-27 17:50:21

不小心用B树套Spaly A掉了
要是不知道B树可以去Google一下
我用的是2,3-树，把每个点像线段树一样用，分裂的时候暴力合并就行了。
（本人纯属弱菜，省选向来暴零，写的常数太大了，跑得死慢，求指教）

以上算法中，最优复杂度为 $O(N\log^2 N)$

（由于题目中 $N, M, Q, MaxV$ 同阶，为了叙述方便，均用 N 代替。）

BZOJ3065

用分块怎么做?
先考虑一个简单的问题：有 C 个排好序的数组，求第 K 大数。

BZOJ3065

用分块怎么做?
先考虑一个简单的问题：有 C 个排好序的数组，求第 K 大数。
二分答案，通过在每个数组里二分查找比它小的数的个数来验证。
单次操作复杂度 $O(C\log^2 N)$

BZOJ3065 分块做法

- 将原序列分块（设初始块大小为 B），每个块内维护原块和排序后的块
- 查找第 K 大：先把两端的区间内元素取出并排序，作为一个临时块。然后二分答案，在每个块（区间中间的完整块 $+$ 两端元素排序生成的临时块）里二分查找比它小的数的个数来验证。
- 修改：只需暴力更新一个块
- 插入：暴力更新一个块，如果这个块的大小超过 $2B$，将它分裂成两个块

复杂度：

- 初始化 $O(N \log B)$
- 单次询问 $O(B \log B + \frac{N}{B} \times \log N \log B)$
- 单次修改 $O(B)$
- 单次插入（无分裂）$O(B)$，分裂 $O(B \log B)$

复杂度不优?

BZOJ3065 实际表现

官方题解：

No.	RunID	User	Memory	Time	Language	Code_Length	Submit_Time
1	353661	3065_scap_seg	179260 KB	31496 MS	C++	8608 B	2013-02-22 10:59:45
2	353685	3065_seg_treap	60352 KB	43416 MS	C++	7093 B	2013-02-22 11:22:54
3	353671	3065_treap_seg	199052 KB	44860 MS	C++	8007 B	2013-02-22 11:05:29

普通情况：

User	Problem	Result	Memory	Time	Language	Code_Length
subconscious0	3065	Accepted	144672 kb	31188 ms	C++	3707 B
yyhs	3065	Accepted	144672 kb	30880 ms	C++	3724 B
balabalatest	3065	Accepted	144672 kb	36044 ms	C++	4293 B
Ryoko_Hirosue	3065	Accepted	144672 kb	36388 ms	C++	4293 B
Evan	3065	Accepted	335084 kb	18316 ms	C++	5101 B
tlzmybm	3065	Accepted	335084 kb	18076 ms	C++	5111 B
Evan	3065	Accepted	335084 kb	17864 ms	C++	5111 B

BZOJ3065 实际表现-分块

截至 2016-3-20，分块做法排名在 BZOJ AC 记录第一页：

No.	RunID	User	Memory	Time	Language	Code_Length	Submit_Time
1	925018	faebdc	77444 KB	7100 MS	C++	3565 B	2015-04-09 12:50:52
2	403414(3)	hta1	119532 KB	7212 MS	C++	4696 B	2013-05-02 18:56:43
3	403602	hta	119532 KB	7236 MS	C++	4648 B	2013-05-02 20:52:58
4	358235	ldf921	57432 KB	7332 MS	C++	5608 B	2013-03-01 14:09:38
5	1235791(4)	yang1999422	77476 KB	7408 MS	C++	3349 B	2016-01-17 16:10:47
6	583347	guoyu1098	238668 KB	11276 MS	C++	4029 B	2014-03-22 15:15:03
7	406044(2)	xu_yue	86312 KB	11600 MS	C++	4523 B	2013-05-06 15:20:47
8	549636(10)	wangyisong1996	131288 KB	15044 MS	C++	8226 B	2014-02-18 16:51:35
9	894513(4)	litc	84896 KB	15420 MS	C++	5126 B	2015-03-17 19:55:32
10	355115(3)	Seter	19480 KB	15616 MS	C++	3902 B	2013-02-24 18:55:42
11	894753	Hereafter	31404 KB	16104 MS	C++	5084 B	2015-03-17 21:01:24
12	434474(5)	FalseMan	493744 KB	16104 MS	C++	5937 B	2013-06-14 12:09:47
13	943980(3)	rfy	315496 KB	16544 MS	C++	4013 B	2015-04-22 13:18:44
14	846974(2)	nodgd	276444 KB	16692 MS	C++	6712 B	2015-01-24 19:01:29
15	678859(3)	hoblovski_1	354388 KB	16968 MS	Pascal	6977 B	2014-06-24 16:42:17
16	617877(3)	hehehehe111	285012 KB	16984 MS	C++	6349 B	2014-04-18 20:16:31
17	943948(7)	1234567891	315496 KB	17160 MS	C++	4352 B	2015-04-22 12:57:11
18	690046	Hoblovski	354388 KB	17212 MS	Pascal	6997 B	2014-07-08 13:09:44
19	752619(6)	Lweb	440752 KB	17360 MS	C++	5511 B	2014-10-13 17:30:47
20	1323760(25)	SQRT_decomp_dafahao	2492 KB	17440 MS	C++	5326 B	2016-03-14 08:17:18

优势：常数小，占用内存小

BZOJ3065 其它分块做法

【块状链表套线段树】

看标题知算法……每个块里维护个线段树，代表这个块里出现的数值。然后……啊……是吧……

不多说了……大家都懂……查询还是照例在线段树上走一走~

O(logm * sqrt(n)) （m是权值最大值）

常数巨大……

【将询问分块】

也就是说根号的算法的末日到了?

没有！！！！！

刚才带O(sqrt(n))的，不妨换一种思路，把根号戴在q头上。（q是询问次数）

每操作一段时间就暴力重建出函数式线段树前缀和，新来的插入和查询用一个数组暴力记下来。询问也乱搞下。

这个是Seter的AC方法，详细可以看Seter的题解：http://psr.2333333.tk/

orz Seter……跑得比O(lognlogm)的还快……碾压标程……

数据结构 vs 分块

	（树形）数据结构	分块
时间复杂度	$O(\log N)$ 级别	$O(\sqrt{N})$ 级别
常数	通常较大	较小
空间占用	嵌套和可持久化数据结构空间占用非常大 然后写个基于引用计数的垃圾收集器~	一般
限制	需要信息有明显的可合并性质，维护复杂信息通常需要较复杂的嵌套数据结构，甚至不可做	限制通常较为宽松
编码复杂度	通常较大	通常较小
模板	结构一致，容易模板化	结构较乱，难以模板化

算法竞赛中，应根据题目特点选择合适的解法，不要盲目认为“分块复杂度高，不如高级数据结构”。

在比赛时，如果某题正解细节复杂，比赛剩余时间不够，而分块方法简便，能拿到大部分分数，也可以考虑使用。

小知识-带插入的分块实现

- 数组：定位元素 $O(1)$，插入删除需要搬动元素 $O(N)$
- 链表：定位元素需要遍历 $O(N)$，插入删除 $O(1)$

如何实现分块的 $O(\sqrt{N})$?

1. 块状链表：每个块一个链表元素，里面是对应的块的数组。
2. “数组套数组”：外层数组的每个元素是一个指向对应的块的数组的指针。此方法分裂块时需要搬动至多 $O(\sqrt{N})$ 个元素。

以上方法定位和插入删除均为 $O(\sqrt{N})$

小知识-数组与指针

C/C++/Pascal 语言中，指针可以像数组一样访问，数组名也可以看成是一个指针（Pascal 里只有下标从 0 开始的数组可以这么做）。

动态开辟数组：

C 语言

```
int *a=(int*)malloc(n*sizeof(int));
free(a);
```

C++

```
int *a=new int[n];
delete[] a;
```

Pascal

```
getmem(a,n*sizeof(longint));
freemem(a);
```

Pascal 代码中 a 的定义为var a:^longint。

小知识-动态/静态开内存

但是我们通常不用 new/delete，因为速度较慢，容易被卡常数。
算法竞赛时，一般使用“静态开内存，动态数据结构”。

更多...

分块不仅仅能做这些...

Codechef NOV14.FNCS Chef and Churu

有一个含 N 个数字的数组 A，元素标号 1 到 N，同时他也有 N 个函数，也标号 1 到 N。第 i 个函数会返回数组中标号 L[i] 和 R[i] 之间的元素的和。
有两种操作：

- 1 x y 将数组的第 x 个元素修改为 y。
- 2 m n 询问标号在 m 和 n 之间的函数的值的和。

$N, Q \leq 10^5$
10%:$N <= 1000, Q <= 1000$
10%:$R - L \leq 10$, 所有的 x 各不相同

Codechef NOV14.FNCS Chef and Churu 解答

- 每 B 个函数分为一个块。
- $Cnt_{i,j}::=A_j$ 在第 i 块函数中被累加了多少次。预处理。
- $Val_i::=$ 第 i 块元素的返回值之和。预处理初值。
- 树状数组 $C::=$ 维护数组 A 前缀和。
- 数组单点修改：根据 $Cnt_{i,x}$ 更新整个 Val 数组。更新树状数组。
- 函数区间询问：中间的完整块直接使用 Val 中已经维护好的值，两端剩下的元素用树状数组暴力。

复杂度：

- 预处理 $O(\frac{N^2}{B})$
- 单次修改 $O(\frac{N}{B}+\log N)$
- 单次询问 $O(\frac{N}{B}+B\log N)$
- 空间 $O(\frac{N^2}{B})$

取 $B=\sqrt{\frac{N}{\log N}}$，总的时间复杂度为 $O((N+Q)\sqrt{N\log N})$

Codechef MAY15.Chef and Balanced Strings

定义“平衡字符串”为每种字符都出现偶数次的字符串。
有一个只包含小写字母的字符串 S（长度为 N）和 Q 个询问。
询问 (L, R, T)：输出 S 的所有左右端点在 $[L, R]$ 内的平衡子串的长度的 T 次方之和。$T \in \{0, 1, 2\}$
强制在线。
$N, Q \leq 10^5$

Codechef MAY15.Chef and Balanced Strings 解答 Part1

用一个 26 位二进制数表示一个字符串中每种字符出现次数的奇偶性（以下简称“奇偶性”）。

A_i 表示 $S[1..i]$ 的奇偶性（其中 $A_0 = 0$）。

$S[L+1..R]$是平衡字符串 $\iff A_L = A_R$

数组 A 中有 $O(N)$ 种不同元素，可以离散化。

Codechef MAY15.Chef and Balanced Strings 解答 Part2

$O(N)$ 算法

$T=0$，仅计数:

for $i \leftarrow L..R$ **do**

  $ans \leftarrow ans + cnt[A[i]]$

  $cnt[A[i]] \leftarrow cnt[A[i]] + 1$

end for

$T=1$，长度和，$\sum(i-j) = Cnt \times i - \sum j$:

for $i \leftarrow L..R$ **do**

  $ans \leftarrow ans + cnt[A[i]] \times i - sum[A[i]]$

  $cnt[A[i]] \leftarrow cnt[A[i]] + 1$

  $sum[A[i]] \leftarrow sum[A[i]] + i$

end for

$T=2$，二次方和，$\sum(i-j)^2 = Cnt \times i^2 + \sum j^2 - 2Cnt \times i \times \sum j$

Codechef MAY15.Chef and Balanced Strings 解答 Part3

分块

块大小为 B。预处理出以下信息：

- $Pre[T][i,j] ::= L = i \times B, R = j$ 的答案
- $Suf[T][i,j] ::= L = i, R = j \times B$ 的答案

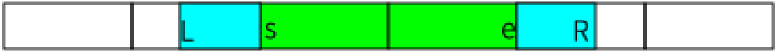
预处理后，对于左右端点至少有一个在块边界上的情况，答案都已知。预处理复杂度 $O(\frac{N^2}{B})$。

Codechef MAY15.Chef and Balanced Strings 解答 Part4

分块

询问 (L,R,T)：
L,R 在一个块内：暴力；

一般情况：设 $[L,R]$ 中由完整的块组成的部分是 $[s,e]$



$$\begin{aligned} Ans[L,R] &= Ans[L,e] + Ans[s,R] - Ans[s,e] \\ &+ Sum\{(j-i)^T \mid A_i = A_j, i \in [L,s-1], j \in [e+1,R]\} \end{aligned} \quad (1)$$

$Ans[s,e], Ans[L,e], Ans[s,R]$ 均已预处理出。剩下的部分涉及的范围不超过两个块，可以暴力。
取 $B=\sqrt{N}$，时间复杂度 $O((N+Q)\sqrt{N})$，空间复杂度 $O(N\sqrt{N})$。

Codechef OCT14.Children Trips

给一棵 N 个点的树，边上有权值，边权为 1 或 2。有 Q 个操作。操作 (X, Y, P)：从 X 走到 Y，每一步可以经过多条边，但总长度不得超过 P，求最少步数。
$N, Q \leq 10^5$

提示："从 X 走到 Y" 和 "从 X，Y 分别走到离 LCA 距离不超过 P 的位置，再走 1-2 步" 答案是相同的。思考算法时，可以只考虑 Y 是 X 的祖先的情况。

Codechef OCT14.Children Trips 解答

$F_{i,j}$::= 从 i 向上走$\underline{2^j}$ 条边到达的点。
$L_{i,j}$::= 从 i 向上走$\underline{2^j}$ 条边经过的路程。
P 较大（步数较小）的情况：一步一步向上走，每一步使用倍增法（倍增求能走几条边)。时间 $O(\frac{N \log N}{T})$

Codechef OCT14.Children Trips 解答

$F_{i,j}$::= 从 i 向上走2^j 条边到达的点。
$L_{i,j}$::= 从 i 向上走2^j 条边经过的路程。
P 较大（步数较小）的情况：一步一步向上走，每一步使用倍增法（倍增求能走几条边）。时间 $O(\frac{N\log N}{T})$

P 较小的情况：
这样的 P 不多，考虑预处理：
$G_{P,i,j}$::= 一步长度限制为 P，从 i 向上走2^j 步到达的点。预处理复杂度 $O(T \times N\log N)$
倍增求需要走几步。$O(\log N)$
按 P 的大小对询问排序，每次只保留一个 P 预处理出的结果，空间复杂度降为 $O(N\log N)$
取 $T = \sqrt{N}$，时间复杂度 $O((N+Q)\sqrt{N}\log N)$

BZOJ2038 [2009 国家集训队] 小 Z 的袜子

有 N 个正整数（每个数都不超过 N），M 个询问。
询问 (L, R)：从第 L 至第 R 个数中随机选出两个数（不同位置），问两个数相等的概率。
$N, M \leq 50000$

BZOJ2038 [2009 国家集训队] 小 Z 的袜子解答

$Ans = \frac{\sum C_i(C_i-1)}{(R-L+1)(R-L)}$

维护当前区间 $[L, R]$ 中每种数出现次数 C_i 和答案 Ans，则有性质：

已知区间 $[L, R]$ 的信息，可以用较小的代价算出区间 $[L \pm 1, R]$ 或 $[L, R \pm 1]$ 的信息（本题为 $O(1)$）。

BZOJ2038 [2009 国家集训队] 小 Z 的袜子解答

$Ans = \frac{\sum C_i(C_i-1)}{(R-L+1)(R-L)}$
维护当前区间 $[L, R]$ 中每种数出现次数 C_i 和答案 Ans，则有性质：
已知区间 $[L, R]$ 的信息，可以用较小的代价算出区间 $[L \pm 1, R]$ 或 $[L, R \pm 1]$ 的信息（本题为 $O(1)$）。
“莫队算法”：

- 将所有询问以 $\frac{L}{\sqrt{N}}$ 为第一关键字，R 为第二关键字排序
- 按排好的顺序，通过移动当前区间左右端点来回答询问

复杂度:

- 右端点移动复杂度：左端点所在块不变时，右端点单调向右移动。总共有 $\sqrt{N}$ 个块，总时间复杂度 $O(N\sqrt{N})$
- 左端点移动复杂度：左端点在某个块内移动时，复杂度 $O(\sqrt{N})$；左端点所在块改变时（最多 $\sqrt{N}$ 次），复杂度 $O(N)$
- 以上两者总的时间复杂度：$O((N+Q)\sqrt{N})$

SPOJ Count on a tree II

有一个 N 个节点的树，每个点上有整数点权。M 个询问。
询问 (u, v)：求树上从 u 到 v 的路径上有几种不同的点权。
$N \leq 40000, M \leq 100000$

SPOJ Count on a tree II 解答 Part1

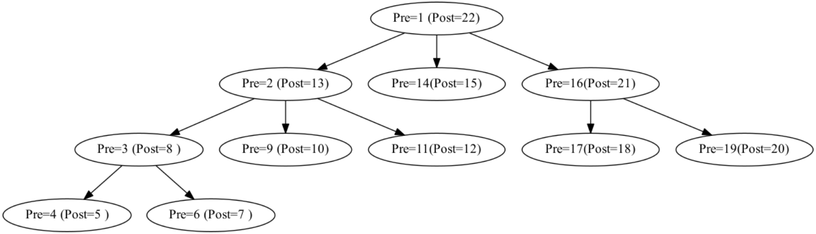
如果题目中操作的是序列而不是树，可以直接套用上题的算法。
如何将“莫队算法”扩展到树上的情况？

SPOJ Count on a tree II 解答 Part2

树上莫队

如何标号和分块?

- DFS 序
- 差值为两点在 DFS 的过程中的距离



以 Pre_i（进入节点 i 的时间）为节点标号。排序询问时，以 $(\frac{Pre_u}{B}, Pre_v)$ 为关键字进行双关键字排序。

SPOJ Count on a tree II 解答 Part2

树上莫队

如何维护当前答案，移动“左右端点”？

- 维护信息：当前询问是 $[u, v]$，维护从 u 到 v 的路径上所有点的信息，但不包括 LCA
- $[u, v] \to [u', v']$：u 点暴力移向 u'，v 点暴力移向 v'，经过的点（除了 $LCA(u, u'), LCA(v, v')$）状态（是否被答案包含）取反
- 回答询问时临时把 LCA 加进去即可

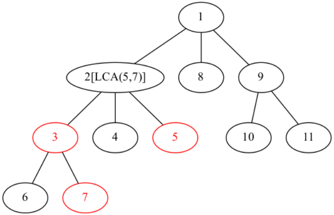


Figure: 询问 $[7, 5]$

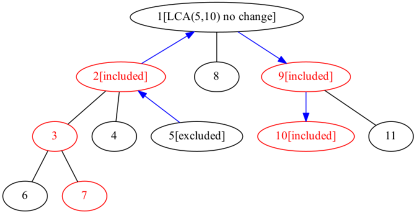


Figure: $[7, 5] \to [7, 10]$

树分块（块状树）

刚才提到，莫队算法可以扩展到树上的情况。
那么，一般的分块能不能扩展到树上的情况呢？

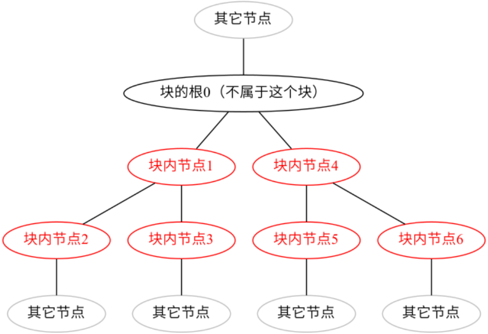
树分块

简单的想法：DFS 时另维护一个栈 S，访问一个节点时将节点入栈（类似 Tarjan 算法求强连通分量），当栈中节点个数超过 B 个时将这些节点分为一个块并清空栈。
相当于按 DFS 序分块，块内节点连通性没有保证。

树分块 - 常用方法

正确的做法：DFS 时另维护一个栈 S，访问一个节点时将节点入栈，当栈中当前节点的子孙个数超过 B 时，将这些节点分为一个块并弹出这些节点。

树分块 - 效果



每个树块都是与一个“根”相连的若干个连通块（注意“根”不属于这个树块）。
为什么要这样做？为什么不规定一个树块是~~一个连通块~~？
好处：即使树上存在度数很大的点（“菊花图”），也能保证一个块的大小在 $[B, 3B]$ 的范围内。

树分块 - 用途

类似序列的分块，可以在每个树块上维护一些块内的信息，达到加速算法的目的。

例题:BZOJ1036 [ZJOI2008] 树的统计

三种操作：树上单点修改，询问树上路径权值和，询问树上路径权值最大值

树分块 - 用途

类似序列的分块，可以在每个树块上维护一些块内的信息，达到加速算法的目的。
例题:BZOJ1036 [ZJOI2008] 树的统计
三种操作：树上单点修改，询问树上路径权值和，询问树上路径权值最大值

数据结构做法：树链剖分或 Link-Cut Tree，一次操作 $O(\log^2 N)$ 或 $O(\log N)$。
暴力：修改 $O(1)$，询问时一步一步暴力向上走到 LCA，$O(N)$。
树分块：每个点维护它到“所在块的根”的权值和以及最小值，修改时更新一个块 $O(\sqrt{N})$，询问时一块一块向上走，$O(\sqrt{N})$。

END