简单的区间修改和查询问题

WNJXYK

Jilin University

2017 年 2 月

目录

Part Zero

预备知识

时间复杂度

只是非常粗略、简单地介绍一下，为之后的知识做铺垫

时间复杂度

只是非常粗略、简单地介绍一下，为之后的知识做铺垫

一般来说，一个算法执行完毕所花费的时间与其语句执行次数成正比，所以我们通过语句执行次数来衡量一个算法在时间方面的优劣。

时间复杂度

只是非常粗略、简单地介绍一下，为之后的知识做铺垫

一般来说，一个算法执行完毕所花费的时间与其语句执行次数成正比，所以我们通过语句执行次数来衡量一个算法在时间方面的优劣。

一般来说，一个算法的语句执行次数是算法面对的问题规模 n 的一个函数，我们记为 $T(n)$。再引入一个辅助函数 $f(n)$，使得 $T(n) \div f(n) = C, n \rightarrow \infty$（C 是一个常数）。

举两个例子

选择排序的时间复杂度为 $O(n)$

```
for(int i=1;i<n;i++)
  for (int j=i+1;j<=n;j++)
    if (num[i]>num[j]) swap(num[i],num[j]);
```

二分发的时间复杂度为 $O(Log_2 n)$

```
int left=1,right=n,Ans=n+1;
while(left<=right){
  int mid=(left+right)/2;
  if (check(mid)){
    And=min(mid,Ans);
    right=mid-1;
  }else{
    left=mid+1;
  }
}
```

一般来说，我们认为我们使用的计算机可以在 1S 的时间之内处理数量级在 1e8 的语句，我们把问题规模带入时间复杂度，然后就能比较计算出我们使用的算法是否会超时。

Part Zero

问题引入

例题 0-0

有一个长度为 n 的序列 $A_1, A_2, ..., A_n$，给出 m 个询问区间 $[L, R]$，求区间和。

1. 数据范围 $n \in [1, 100000]$,$m \in [1, 100]$

例题 0-0

有一个长度为 n 的序列 $A_1, A_2, ..., A_n$，给出 m 个询问区间 $[L, R]$，求区间和。

1. 数据范围 $n \in [1, 100000]$,$m \in [1, 100]$
2. 数据范围 $n \in [1, 100000]$,$m \in [1, 100000]$

例题 0-0

有一个长度为 n 的序列 $A_1, A_2, ..., A_n$，给出 m 个询问区间 $[L, R]$，求区间和。

1. 数据范围 $n \in [1, 100000]$,$m \in [1, 100]$
2. 数据范围 $n \in [1, 100000]$,$m \in [1, 100000]$
3. 问题修改：现在的 m 个操作里，不仅有询问区间和操作，还有还可以选择修改序列中的某一个数

例题 0-0

有一个长度为 n 的序列 $A_1, A_2, ..., A_n$，给出 m 个询问区间 $[L, R]$，求区间和。

1. 数据范围 $n \in [1, 100000]$,$m \in [1, 100]$
2. 数据范围 $n \in [1, 100000]$,$m \in [1, 100000]$
3. 问题修改：现在的 m 个操作里，不仅有询问区间和操作，还有还可以选择修改序列中的某一个数
4. 问题修改：选择修改序列中的某一个数更改为选择序列中的某个区间 $[L, R]$，给区间上每个数字增加或者减少 x

例题 0-0

有一个长度为 n 的序列 $A_1, A_2, ..., A_n$，给出 m 个询问区间 $[L, R]$，求区间和。

1. 数据范围 $n \in [1, 100000]$,$m \in [1, 100]$
2. 数据范围 $n \in [1, 100000]$,$m \in [1, 100000]$
3. 问题修改：现在的 m 个操作里，不仅有询问区间和操作，还有还可以选择修改序列中的某一个数
4. 问题修改：选择修改序列中的某一个数更改为选择序列中的某个区间 $[L, R]$，给区间上每个数字增加或者减少 x

Ext. 在 1、2、3、4 中的询问改为询问最大（小）值呢?

朴素做法

我们用一个数组把所有的数字读入存储，然后遇到一个询问我们就用一个 For 循环遍历这个询问的区间，进行累加操作 (Ext. 求最值操作)

朴素做法

我们用一个数组把所有的数字读入存储，然后遇到一个询问我们就用一个 For 循环遍历这个询问的区间，进行累加操作 (Ext. 求最值操作)

看起来非常靠谱！通过刚才对时间复杂度的学习，我们可以计算出这个算法可以应对问题 1 以及问题 1.Ext。但是对于之后的问题，很明显就会超出问题时限。所以我们要引入新的算法。

Part One

前缀和

例题 1-1

问题：长度为 n 的序列，m 个区间和询问，$n \in [1, 100000]$,$m \in [1, 100000]$

例题 1-1

问题：长度为 n 的序列，m 个区间和询问，$n \in [1, 100000]$,$m \in [1, 100000]$

假设原序列为 $num[x]$，我们定义个一个新的序列 $sum[x] = \sum_{i=1}^{x} num[i]$，我们很容易发现

$$\sum_{i=L}^{R} num[i] = sum[R] - sum[L-1]$$

例题 1-1

问题：长度为 n 的序列，m 个区间和询问，$n \in [1, 100000]$,$m \in [1, 100000]$

假设原序列为 $num[x]$，我们定义个一个新的序列 $sum[x] = \sum_{i=1}^{x} num[i]$，我们很容易发现

$$\sum_{i=L}^{R} num[i] = sum[R] - sum[L-1]$$

这样一来，我们在回答问题之前只需要预处理出 $sum[x]$ 数组，就可以将原来一个需要 $O(n)$ 的时间复杂度才能完成的询问，优化到 $O(1)$ 了，这样就能很轻松的解决这个问题。

前缀和

假设原序列为 $num[x]$，我们定义个一个新的序列 $sum[x] = \sum_{i=1}^{x} num[i]$，我们就把 $sum[x]$ 称为原序列的前缀和。

前缀和

假设原序列为 $num[x]$，我们定义个一个新的序列 $sum[x] = \sum_{i=1}^{x} num[i]$，我们就把 $sum[x]$ 称为原序列的前缀和。现在我们已经会了前缀和，我们发现他有以下特点

- 线性预处理
- 常数级别询问
- 不支持修改！
- 只支持加减乘除、异或这类可逆的运算（比如区间 GCD 啊，你就不能前缀和）

例题 1-2

定义一个数字是美的，当且仅当这个数字各位不同，有 $T \leq 1000$ 个询问，问区间 $[l, r]$ 内有多少个美的数字。
$1 \leq l \leq r \leq 100000$

来源 HDU 5327

例题 1-2

定义一个数字是美的，当且仅当这个数字各位不同，有 $T \leq 1000$ 个询问，问区间 $[l, r]$ 内有多少个美的数字。
$1 \leq l \leq r \leq 100000$

来源 HDU 5327

我们可以建立一个新的 01 序列，0 表示这个位置上的数字不是美的，1 反之，我们发现之前的询问就变为这里的区间求和。我们用线性的复杂度预处理出这个序列，然后用前缀和搞一搞就能快速 Accepted 了！（其实用一开始最朴素的方法，是有大概率卡过这题的。）

Part One

差分

如果我们把之前的前缀和序列当作原序列，那么之前的原序列在现在这里就是差分序列。用数学表示就是

$$sub[x] = num[x] - num[x-1], num[0] = 0$$

如果想要从差分序列得到原序列，就要从头求和即

$$num[x] = \sum_{i=1}^{x} sub[i]$$

差分

如果我们把之前的前缀和序列当作原序列，那么之前的原序列在现在这里就是差分序列。用数学表示就是

$$sub[x] = num[x] - num[x-1], num[0] = 0$$

如果想要从差分序列得到原序列，就要从头求和即

$$num[x] = \sum_{i=1}^{x} sub[i]$$

看起来没有什么用处诶，

如果我们把之前的前缀和序列当作原序列，那么之前的原序列在现在这里就是差分序列。用数学表示就是

$$sub[x] = num[x] - num[x-1], num[0] = 0$$

如果想要从差分序列得到原序列，就要从头求和即

$$num[x] = \sum_{i=1}^{x} sub[i]$$

看起来没有什么用处诶，现在的确没什么用处。

如果我们把之前的前缀和序列当作原序列，那么之前的原序列在现在这里就是差分序列。用数学表示就是

$$sub[x] = num[x] - num[x-1], num[0] = 0$$

如果想要从差分序列得到原序列，就要从头求和即

$$num[x] = \sum_{i=1}^{x} sub[i]$$

看起来没有什么用处诶，现在的确没什么用处。

但是我们可以发现，如果在差分序列的某一个元素上加减个 delta，就相当给原序列的这个位置上及以后的所有数字都加减个 delta。

ST 表

例题 2-1

问题：长度为 n 的序列，m 个区间最大值询问，$n \in [1, 100000]$,$m \in [1, 100000]$

例题 2-1

问题：长度为 n 的序列，m 个区间最大值询问，$n \in [1, 100000]$,$m \in [1, 100000]$

我们之前的前缀和不能支持最大值操作！虽然不能使用前缀和了，但是预处理的思想还是可以用的。But.

例题 2-1

问题：长度为 n 的序列，m 个区间最大值询问，$n \in [1, 100000]$,$m \in [1, 100000]$

我们之前的前缀和不能支持最大值操作！虽然不能使用前缀和了，但是预处理的思想还是可以用的。But.

发现：对于两个重叠的已知区间答案 $[l1, r1]$, $[l2, r2]$, $r1 \geq 2$，可以很容易合并答案

例题 2-1

问题：长度为 n 的序列，m 个区间最大值询问，$n \in [1, 100000]$,$m \in [1, 100000]$

我们之前的前缀和不能支持最大值操作！虽然不能使用前缀和了，但是预处理的思想还是可以用的。But.

发现：对于两个重叠的已知区间答案 $[l1, r1]$, $[l2, r2]$, $r1 \geq 2$，可以很容易合并答案

解决方案：有选择的预处理！只处理 2^k 长度的区间。

ST 表

ST 表

- 预处理 (运用 DP 思想)

- 预处理 (运用 DP 思想)
  定义 $st[i][j]$ 为从 i 位置开始长度为 2^j 的区间的最大值。
  $st[i][0]$ 已知
  $st[i][j] = max\left(st[i][j-1], st\left[i+2^{(j-1)}\right][j-1]\right)$

ST 表

- 预处理 (运用 DP 思想)
  定义 $st[i][j]$ 为从 i 位置开始长度为 2^j 的区间的最大值。
  $st[i][0]$ 已知
  $st[i][j] = max\left(st[i][j-1], st\left[i+2^{(j-1)}\right][j-1]\right)$
- 询问

ST 表

- 预处理 (运用 DP 思想)
  定义 $st[i][j]$ 为从 i 位置开始长度为 2^j 的区间的最大值。
  $st[i][0]$ 已知
  $st[i][j] = max\left(st[i][j-1], st\left[i+2^{(j-1)}\right][j-1]\right)$
- 询问
  找两个长度合适区间拼起来，假设询问区间为 $[left, right]$
  我们令 k 满足 $2^k \leq right - left + 1, 2^{(k+1)} > right - left + 1$，
  就可以立刻找到从 left 向右长度为 2^k 和从 right 向左长度为 2^k 的区间重叠拼合出答案

ST 表

- 预处理 (运用 DP 思想)
  定义 $st[i][j]$ 为从 i 位置开始长度为 2^j 的区间的最大值。
  $st[i][0]$ 已知
  $st[i][j] = max\left(st[i][j-1], st\left[i+2^{(j-1)}\right][j-1]\right)$
- 询问
  找两个长度合适区间拼起来，假设询问区间为 $[left, right]$
  我们令 k 满足 $2^k \leq right - left + 1, 2^{(k+1)} > right - left + 1$，
  就可以立刻找到从 left 向右长度为 2^k 和从 right 向左长度为 2^k 的区间重叠拼合出答案
- 性能?

ST 表

- 预处理 (运用 DP 思想)
  定义 $st[i][j]$ 为从 i 位置开始长度为 2^j 的区间的最大值。
  $st[i][0]$ 已知
  $st[i][j] = max\left(st[i][j-1], st\left[i+2^{(j-1)}\right][j-1]\right)$
- 询问
  找两个长度合适区间拼起来，假设询问区间为 $[left, right]$
  我们令 k 满足 $2^k \leq right - left + 1, 2^{(k+1)} > right - left + 1$，
  就可以立刻找到从 left 向右长度为 2^k 和从 right 向左长度为 2^k 的区间重叠拼合出答案
- 性能？$O(nLogn)$ 预处理，$O(1)$ 查询。

- 预处理 (运用 DP 思想)
  定义 $st[i][j]$ 为从 i 位置开始长度为 2^j 的区间的最大值。
  $st[i][0]$ 已知
  $st[i][j] = max\left(st[i][j-1], st\left[i+2^{(j-1)}\right][j-1]\right)$
- 询问
  找两个长度合适区间拼起来，假设询问区间为 $[left, right]$
  我们令 k 满足 $2^k \le right - left + 1, 2^{(k+1)} > right - left + 1$，
  就可以立刻找到从 left 向右长度为 2^k 和从 right 向左长度为 2^k 的区间重叠拼合出答案
- 性能？$O(nLogn)$ 预处理，$O(1)$ 查询。
- 支持功能？

- 预处理 (运用 DP 思想)
  定义 $st[i][j]$ 为从 i 位置开始长度为 2^j 的区间的最大值。
  $st[i][0]$ 已知
  $st[i][j] = max\left(st[i][j-1], st\left[i+2^{(j-1)}\right][j-1]\right)$
- 询问
  找两个长度合适区间拼起来，假设询问区间为 $[left, right]$
  我们令 k 满足 $2^k \le right - left + 1, 2^{(k+1)} > right - left + 1$，
  就可以立刻找到从 left 向右长度为 2^k 和从 right 向左长度为 2^k 的区间重叠拼合出答案
- 性能？$O(nLogn)$ 预处理，$O(1)$ 查询。
- 支持功能？最值、GCD、LCM ...

代码实现 (最值)

```
int stMax[Maxn+5][20];

inline void ST(int n){
    for (int i=1;i<=n;i++) stMax[i][0]=num[i];
    for (int j=1;(1<<j)<=n;j++)
        for (int i=1;i+(1<<j)-1<=n;i++)
            stMax[i][j]=max(stMax[i][j-1],stMax[i+(1<<(j
                -1))][j-1]);
}

int queryMax(int left,int right){
    int k=0;
    while((1<<(k+1))<=right-left+1) k++;
    return max(stMax[left][k],stMax[right-(1<<k)+1][k]);
}
```

例题 2-2

一个 $n \leq 1000$ 位数，删除 m 位，使剩下最小。

来源 HDU 3183

例题 2-2

一个 $n \leq 1000$ 位数，删除 m 位，使剩下最小。

来源 HDU 3183

两个数字比较，如果高位比出大小，低位就不需要比较了。所以我们要用贪心的思想，保证后面够删的情况下，当前位尽量选择的小。

例题 2-2

一个 $n \leq 1000$ 位数，删除 m 位，使剩下最小。

来源 HDU 3183

两个数字比较，如果高位比出大小，低位就不需要比较了。所以我们要用贪心的思想，保证后面够删的情况下，当前位尽量选择的小。

于是转化为一个区间最值问题，用 ST 表解决。

例题 2-3

一个长度为 $n \le 100000$ 的序列，询问若干个区间的区间 GCD:gcd_i 和区间 GCD 等于 gcd_i 的区间数量。

来源 HDU 5726

一个长度为 $n \le 100000$ 的序列，询问若干个区间的区间 GCD:gcd_i 和区间 GCD 等于 gcd_i 的区间数量。

来源 HDU 5726

提示：枚举左端点，确定了左端点，从左端点右延伸的 GCD 不同区间只有 Log 数量级个，因为一个数的质因数只有 Log 数量级个。

例题 2-3

一个长度为 $n \leq 100000$ 的序列，询问若干个区间的区间 GCD:gcd_i 和区间 GCD 等于 gcd_i 的区间数量。

来源 HDU 5726

提示：枚举左端点，确定了左端点，从左端点右延伸的 GCD 不同区间只有 Log 数量级个，因为一个数的质因数只有 Log 数量级个。

枚举左端点，然后二分区间 GCD 变化的右端点，这其中计算区间 GCD 的操作使用 ST 表来实现。

Part Three

树状数组

例题 3-1

问题：长度为 n 的序列，m 个单点修改和区间和询问，$n \in [1, 100000]$,$m \in [1, 100000]$

例题 3-1

问题：长度为 n 的序列，m 个单点修改和区间和询问，$n \in [1, 100000]$,$m \in [1, 100000]$

有了修改，那么之前的方法全都不奏效了，因为之前的所有方法只能针对一个静态的询问问题。

例题 3-1

问题：长度为 n 的序列，m 个单点修改和区间和询问，$n \in [1,100000]$,$m \in [1,100000]$

有了修改，那么之前的方法全都不奏效了，因为之前的所有方法只能针对一个静态的询问问题。

有了之前的 ST 表的划分区间再拼合的想法，我们想能不能找到一种划分方法能把任何一个区间划分为 k1 个不相交的区间且任何一个点只被 k2 个我们划分好的区间包含。（分别对应询问和修改的复杂度）

例题 3-1

问题：长度为 n 的序列，m 个单点修改和区间和询问，$n \in [1, 100000]$,$m \in [1, 100000]$

有了修改，那么之前的方法全都不奏效了，因为之前的所有方法只能针对一个静态的询问问题。

有了之前的 ST 表的划分区间再拼合的想法，我们想能不能找到一种划分方法能把任何一个区间划分为 k1 个不相交的区间且任何一个点只被 k2 个我们划分好的区间包含。（分别对应询问和修改的复杂度）答案是肯定的

树状数组

树状数组

- 预备知识

树状数组

- 预备知识
  一个数字 x 对应的二进制串长度为 $\lceil Log_2 x \rceil$
  $lowbit(x)$ 表示取出 x 的二进制最低位的 1，如
  $lowbit(10100_{(2)}) = 100_{(2)}$，实现：$x \& -x$
- 树状数组

- 预备知识
  一个数字 x 对应的二进制串长度为 $\lceil Log_2 x \rceil$
  $lowbit(x)$ 表示取出 x 的二进制最低位的 1，如
  $lowbit(10100_{(2)}) = 100_{(2)}$，实现：$x \& -x$
- 树状数组
  原序列 $num[x]$, 划分序列 $sum[x] = \sum_{x-lowbit(x)+1}^{x} num[x]$
- 单点修改

树状数组

- 预备知识
  一个数字 x 对应的二进制串长度为 $\lceil Log_2 x \rceil$
  $lowbit(x)$ 表示取出 x 的二进制最低位的 1，如
  $lowbit(10100_{(2)}) = 100_{(2)}$，实现：$x\& -x$
- 树状数组
  原序列 $num[x]$, 划分序列 $sum[x] = \sum_{x-lowbit(x)+1}^{x} num[x]$
- 单点修改
  修改位置 x, 只需要修改包含这个位置 x 的划分区间，即
  $sum[i], i \geq x > i - lowbit(i)$

- 预备知识
  一个数字 x 对应的二进制串长度为 $\lceil Log_2 x \rceil$
  $lowbit(x)$ 表示取出 x 的二进制最低位的 1，如
  $lowbit(10100_{(2)}) = 100_{(2)}$，实现：$x\& - x$
- 树状数组
  原序列 $num[x]$, 划分序列 $sum[x] = \sum_{x-lowbit(x)+1}^{x} num[x]$
- 单点修改
  修改位置 x, 只需要修改包含这个位置 x 的划分区间，即
  $sum[i], i \geq x > i - lowbit(i)$
- 区间查询

树状数组

- 预备知识
  一个数字 x 对应的二进制串长度为 $\lceil Log_2 x \rceil$
  $lowbit(x)$ 表示取出 x 的二进制最低位的 1，如
  $lowbit(10100_{(2)}) = 100_{(2)}$，实现：$x\& -x$
- 树状数组
  原序列 $num[x]$, 划分序列 $sum[x] = \sum_{x-lowbit(x)+1}^{x} num[x]$
- 单点修改
  修改位置 x, 只需要修改包含这个位置 x 的划分区间，即
  $sum[i], i \geq x > i - lowbit(i)$
- 区间查询
  查询位置 x，返回 $[1, x]$ 的区间和。
  因为如上划分了区间，所以我们只需要通过 $i - lowbit(i)$ 来寻找下个拼凑区间。

树状数组

- 预备知识
  一个数字 x 对应的二进制串长度为 $\lceil Log_2 x \rceil$
  $lowbit(x)$ 表示取出 x 的二进制最低位的 1，如
  $lowbit(10100_{(2)}) = 100_{(2)}$，实现：$x\& - x$
- 树状数组
  原序列 $num[x]$, 划分序列 $sum[x] = \sum_{x-lowbit(x)+1}^{x} num[x]$
- 单点修改
  修改位置 x, 只需要修改包含这个位置 x 的划分区间，即
  $sum[i], i \geq x > i - lowbit(i)$
- 区间查询
  查询位置 x，返回 $[1, x]$ 的区间和。
  因为如上划分了区间，所以我们只需要通过 $i - lowbit(i)$ 来寻找下个拼凑区间。
- 性能?

树状数组

- 预备知识
  一个数字 x 对应的二进制串长度为 $\lceil Log_2 x \rceil$
  $lowbit(x)$ 表示取出 x 的二进制最低位的 1，如
  $lowbit(10100_{(2)}) = 100_{(2)}$，实现：$x\& - x$
- 树状数组
  原序列 $num[x]$, 划分序列 $sum[x] = \sum_{x-lowbit(x)+1}^{x} num[x]$
- 单点修改
  修改位置 x, 只需要修改包含这个位置 x 的划分区间，即
  $sum[i], i \geq x > i - lowbit(i)$
- 区间查询
  查询位置 x，返回 $[1, x]$ 的区间和。
  因为如上划分了区间，所以我们只需要通过 $i - lowbit(i)$ 来寻找下个拼凑区间。
- 性能？均为 $O(Logn)$
- 功能？

- 预备知识
  一个数字 x 对应的二进制串长度为 $\lceil Log_2 x \rceil$
  $lowbit(x)$ 表示取出 x 的二进制最低位的 1，如
  $lowbit(10100_{(2)}) = 100_{(2)}$，实现：$x\& - x$
- 树状数组
  原序列 $num[x]$, 划分序列 $sum[x] = \sum_{x-lowbit(x)+1}^{x} num[x]$
- 单点修改
  修改位置 x, 只需要修改包含这个位置 x 的划分区间，即
  $sum[i], i \geq x > i - lowbit(i)$
- 区间查询
  查询位置 x，返回 $[1, x]$ 的区间和。
  因为如上划分了区间，所以我们只需要通过 $i - lowbit(i)$ 来寻找下个拼凑区间。
- 性能？均为 $O(Logn)$
- 功能？单点修改区间查询，区间修改单点查询，加减乘除异或（单点修改，1-X 区间最值）（区间修改，区间求和）

代码实现

```
const int Maxn=1e5;
int sum[Maxn+5]

inline int lowbit(int x){
  return x&-x;
}

inline void add(int x,int val){
  for (int i=x;i<=Maxn;i+=lowbit(i)) sum[i]+=val;
}

inline int sum(int x){
  int ret=0;
  for (int i=x;i;i-=lowbit(i)) ret+=sum[i];
  return ret;
}
```

例题 3-2

二位平面上 $n(n \le 15000)$ 个点 $(x_i, y_i), 1 \le x, y \le 32000$，每个点的等级等于 $(0,0)-(x_i, y_i)$ 内不包括自己的点的数量，分别输出等级为 0 到 $n-1$ 的点的数量。

来源 HDU1541

例题 3-2

二位平面上 $n(n \le 15000)$ 个点 $(x_i, y_i), 1 \le x, y \le 32000$，每个点的等级等于 $(0,0)-(x_i, y_i)$ 内不包括自己的点的数量，分别输出等级为 0 到 $n-1$ 的点的数量。

来源 HDU1541

二位平面?

例题 3-2

二位平面上 $n(n \leq 15000)$ 个点 $(x_i, y_i), 1 \leq x, y \leq 32000$，每个点的等级等于 $(0,0) - (x_i, y_i)$ 内不包括自己的点的数量，分别输出等级为 0 到 $n-1$ 的点的数量。

来源 HDU1541

二位平面?
排序能让第一维有序

例题 3-2

二位平面上 $n(n \leq 15000)$ 个点 $(x_i, y_i), 1 \leq x, y \leq 32000$，每个点的等级等于 $(0,0)-(x_i, y_i)$ 内不包括自己的点的数量，分别输出等级为 0 到 $n-1$ 的点的数量。

来源 HDU1541

二位平面?
排序能让第一维有序
第二维使用树状数组解决!

二位平面上 $n(n \le 15000)$ 个点 $(x_i, y_i), 1 \le x, y \le 32000$，每个点的等级等于 $(0,0) - (x_i, y_i)$ 内不包括自己的点的数量，分别输出等级为 0 到 $n-1$ 的点的数量。

来源 HDU1541

二位平面?
排序能让第一维有序
第二维使用树状数组解决!
问题解决!

例题 3-3

一个序列长度为 $n, (n \leq 100000)$，交换两个数字 x_i, x_j 的代价是 $x_i + x_j$，求把一个序列交换成升序的最小代价。

来源 HDU2838

例题 3-3

一个序列长度为 $n, (n \leq 100000)$，交换两个数字 x_i, x_j 的代价是 $x_i + x_j$，求把一个序列交换成升序的最小代价。

来源 HDU2838

交换序列的最小代价，逆序对

例题 3-3

一个序列长度为 $n, (n \leq 100000)$，交换两个数字 x_i, x_j 的代价是 $x_i + x_j$，求把一个序列交换成升序的最小代价。

来源 HDU2838

交换序列的最小代价，逆序对

树状数组维护逆序对数量和逆序对总和

例题 3-3

一个序列长度为 $n,(n \leq 100000)$，交换两个数字 x_i, x_j 的代价是 $x_i + x_j$，求把一个序列交换成升序的最小代价。

来源 HDU2838

交换序列的最小代价，逆序对

树状数组维护逆序对数量和逆序对总和

问题解决!

例题 3-4

一个序列长度为 $n, (n \leq 100000)$，区间加，单点查询　　来源 HDU1556

例题 3-4

一个序列长度为 $n, (n \leq 100000)$，区间加，单点查询 来源 HDU1556

之前讲的差分派上用场了

例题 3-4

一个序列长度为 $n, (n \leq 100000)$，区间加，单点查询 来源 HDU1556

之前讲的差分派上用场了

用了差分之后，区间修改相当于在树状数组上改两个点而查询则是区间求和。

例题 3-4

一个序列长度为 $n, (n \leq 100000)$，区间加，单点查询　来源 HDU1556

之前讲的差分派上用场了

用了差分之后，区间修改相当于在树状数组上改两个点而查询则是区间求和。

不细讲

Part Four

线段树（单点修改）

例题 4-1

问题：长度为 n 的序列，m 个 单点修改 或者 求区间和 或者 求区间最值 ，$n \in [1, 100000]$,$m \in [1, 100000]$

例题 4-1

问题：长度为 n 的序列，m 个 单点修改 或者 求区间和 或者 求区间最值 ，$n \in [1, 100000]$,$m \in [1, 100000]$

树状数组不支持区间最值的查询（其实是部分支持的）

例题 4-1

问题：长度为 n 的序列，m 个 单点修改 或者 求区间和 或者 求区间最值 ，$n \in [1, 100000]$,$m \in [1, 100000]$

树状数组不支持区间最值的查询（其实是部分支持的）

就只能使用线段树了

线段树

- 概念

- 概念
  线段树是一棵二叉树，每个节点对应序列上的一个区间，父节点的区间可以由两个子节点合并得到。如果某一父节点对应序列上的区间为 $[L, R]$，则令 $mid = (L + R)/2$，其左儿子对应区间为 $[L, mid]$，右儿子对应区间为 $[mid + 1, R]$。线段树的每个叶子节点，对应顺次对应序列上的一个元素。空间复杂度 $O(2n)$，插入、修改、查询操作时间复杂度 $O(Logn)$。
- 建树

线段树

- 概念
  线段树是一棵二叉树，每个节点对应序列上的一个区间，父节点的区间可以由两个子节点合并得到。如果某一父节点对应序列上的区间为 $[L, R]$，则令 $mid = (L + R)/2$，其左儿子对应区间为 $[L, mid]$，右儿子对应区间为 $[mid + 1, R]$。线段树的每个叶子节点，对应顺次对应序列上的一个元素。空间复杂度 $O(2n)$，插入、修改、查询操作时间复杂度 $O(Logn)$。
- 建树 一般来说，为了使用方便，我们的线段树要进行建树来初始化。
- 单点修改

线段树

- 概念
  线段树是一棵二叉树，每个节点对应序列上的一个区间，父节点的区间可以由两个子节点合并得到。如果某一父节点对应序列上的区间为 $[L, R]$，则令 $mid = (L + R)/2$，其左儿子对应区间为 $[L, mid]$，右儿子对应区间为 $[mid + 1, R]$。线段树的每个叶子节点，对应顺次对应序列上的一个元素。空间复杂度 $O(2n)$，插入、修改、查询操作时间复杂度 $O(Logn)$。
- 建树 一般来说，为了使用方便，我们的线段树要进行建树来初始化。
- 单点修改 和树状数组一样，我们只需要修改树上节点管理区间包含我们要修改位置的那些树上节点即可。
- 合并区间

线段树

- 概念
  线段树是一棵二叉树，每个节点对应序列上的一个区间，父节点的区间可以由两个子节点合并得到。如果某一父节点对应序列上的区间为 $[L, R]$，则令 $mid = (L + R)/2$，其左儿子对应区间为 $[L, mid]$，右儿子对应区间为 $[mid + 1, R]$。线段树的每个叶子节点，对应顺次对应序列上的一个元素。空间复杂度 $O(2n)$，插入、修改、查询操作时间复杂度 $O(Logn)$。
- 建树 一般来说，为了使用方便，我们的线段树要进行建树来初始化。
- 单点修改 和树状数组一样，我们只需要修改树上节点管理区间包含我们要修改位置的那些树上节点即可。
- 合并区间 因为两个子区间可以合并成一个父区间，所以我们可以让线段树支持很多操作，求和，求最值，还有很多鬼畜的操作。
- 区间查询

- 概念
  线段树是一棵二叉树，每个节点对应序列上的一个区间，父节点的区间可以由两个子节点合并得到。如果某一父节点对应序列上的区间为 $[L, R]$，则令 $mid = (L + R)/2$，其左儿子对应区间为 $[L, mid]$，右儿子对应区间为 $[mid + 1, R]$。线段树的每个叶子节点，对应顺次对应序列上的一个元素。空间复杂度 $O(2n)$，插入、修改、查询操作时间复杂度 $O(Logn)$。
- 建树 一般来说，为了使用方便，我们的线段树要进行建树来初始化。
- 单点修改 和树状数组一样，我们只需要修改树上节点管理区间包含我们要修改位置的那些树上节点即可。
- 合并区间 因为两个子区间可以合并成一个父区间，所以我们可以让线段树支持很多操作，求和，求最值，还有很多鬼畜的操作。
- 区间查询 我们需要找到若干个区间能拼合成我们的目标区间。
- 画图演示

单点修改的线段树

这里，演示一个单点加减修改，区间查询和的线段树。

线段树的实现：结构体

建议使用结构体来实现线段树（好看），一般来说结构体应该包含以下几个元素。

```
const int Maxn=5e4;
struct Btree{
  int left,right;
  long long sum;
};
Btree tree[Maxn*4+5];
```

一般，我们给线段树写一个合并区间函数，用于把两个子区间的信息合并到个父区间。

```
inline void update(int x){
  if (tree[x].left==tree[x].right) return;
  tree[x].sum=tree[x*2].sum+tree[x*2+1].sum;
}
```

线段树的实现：建树

我们按照线段树的概念建树，由完全二叉树建树的技巧，我们认为规定如果父亲节点的编号为 x，那么他的左儿子编号为 $2x$，右儿子编号为 $2x+1$，整个树的根的编号为 1。在建树的时候，我们分配好 left,right，还可以初始化树节点上的信息。

```
void build(int x,int left,int right){
  tree[x].left=left;
  tree[x].right=right;
  if (left==right){
    //scanf("%lld",&tree[x].sum);
  }else{
    int mid=(left+right)/2;
    build(x*2,left,mid);
    build(x*2+1,mid+1,right);
    update(x);
  }
}
```

线段树的实现：单点修改

由区间划分情况，我们只需要从根节点，一路向下找子节点包含要修改位置的节点修改即可，一直修改到树的叶子节点。

```
void modify(int x,int pos,int val){
  if (tree[x].left==tree[x].right){
    tree[x].sum+=val;
  }else{
    int mid=(tree[x].left+tree[x].right)/2;
    if (pos<=mid)
      modify(x*2,pos,val);
    else
      modify(x*2+1,pos,val);
    update(x);
  }
}
```

线段树的实现：区间查询

我们需要找到若干个区间拼合成目标区间

```
LL query(int x,int left,int right){
  if (left<=tree[x].left && tree[x].right<=right){
    return tree[x].sum;
  }else{
    LL ret=0;
    int mid=(tree[x].left+tree[x].right)/2;
    if (left<=mid) ret+=query(x*2,left,right);
    if (right>=mid+1) ret+=query(x*2+1,left,right);
    return ret;
  }
}
```

例题 4-2

问题：长度为 n 的序列，m 个 单点修改 或者 求区间和，$n \in [1, 50000]$,$m \in [1, 40000]$

来源 HDU`1166`

例题 4-2

问题：长度为 n 的序列，m 个 单点修改 或者 求区间和 ，$n \in [1, 50000]$,$m \in [1, 40000]$

来源 HDU1166

线段树的裸题

问题：长度为 n 的序列，m 个 单点修改 或者 求区间最大值，$n \in [1, 200000]$,$m \in [1, 5000]$

来源 HDU1754

例题 4-3

问题：长度为 n 的序列，m 个 单点修改 或者 求区间最大值，$n \in [1, 200000]$,$m \in [1, 5000]$

来源 HDU1754

线段树的裸题

例题 4-4

问题：有 n 个点组成一个线段，标号 x 点与 $x-1, x+1$ 相连，m 个操作：

- 摧毁一个点
- 恢复上一个被摧毁的点
- 问与点 x 所在的线段长度

$n \in [1, 50000]$,$m \in [1, 50000]$

来源 HDU1540

例题 4-4

问题：有 n 个点组成一个线段，标号 x 点与 $x-1, x+1$ 相连，m 个操作：

- 摧毁一个点
- 恢复上一个被摧毁的点
- 问与点 x 所在的线段长度

$n \in [1, 50000]$,$m \in [1, 50000]$

来源 HDU1540

线段树维护一个 01 序列！维护很多信息！

例题 4-4

问题：有 n 个点组成一个线段，标号 x 点与 $x-1, x+1$ 相连，m 个操作：

- 摧毁一个点
- 恢复上一个被摧毁的点
- 问与点 x 所在的线段长度

$n \in [1, 50000]$,$m \in [1, 50000]$

来源 HDU1540

线段树维护一个 01 序列！维护很多信息！
维护每个点能向左延伸多长，能向右延伸多长。

例题 4-4

问题：有 n 个点组成一个线段，标号 x 点与 $x-1, x+1$ 相连，m 个操作：

- 摧毁一个点
- 恢复上一个被摧毁的点
- 问与点 x 所在的线段长度

$n \in [1, 50000]$,$m \in [1, 50000]$

来源 HDU1540

线段树维护一个 01 序列！维护很多信息！
维护每个点能向左延伸多长，能向右延伸多长。
为此，我们还需要额外维护一个区间是否被完全覆盖的标记

例题 4-4

问题：有 n 个点组成一个线段，标号 x 点与 $x-1, x+1$ 相连，m 个操作：

- 摧毁一个点
- 恢复上一个被摧毁的点
- 问与点 x 所在的线段长度

$n \in [1, 50000]$,$m \in [1, 50000]$

来源 HDU1540

线段树维护一个 01 序列！维护很多信息！
维护每个点能向左延伸多长，能向右延伸多长。
为此，我们还需要额外维护一个区间是否被完全覆盖的标记

例题 4-4

```
struct Btree{
    int left,right;
    int lMax,rMax;
    int sum;
    inline int len(){
        return right-left+1;
    }
};
Btree tree[Maxn*4+5];

inline void update(Btree &tree, Btree left, Btree right){
    tree.left=left.left;
    tree.right=right.right;
    tree.lMax=max(left.lMax,(left.len()==left.sum?left.len()+right.lMax:0));
    tree.rMax=max(right.rMax,(right.len()==right.sum?right.len()+left.rMax:0));
    tree.sum=left.sum+right.sum;
}
```

线段树的实现：区间修改?

如果我们要修改一个区间？看了区间查询的代码收到启发，能不能也按照单点修改的样子修改一个区间。

线段树的实现：区间修改?

如果我们要修改一个区间？看了区间查询的代码收到启发，能不能也按照单点修改的样子修改一个区间。

```
void modify(int x,int left,int right,int val){
  if (left<=tree[x].left && tree[x].right<=right){
    tree[x].sum+=val;
  }else{
    int mid=(tree[x].left+tree[x].right)/2;
    if (left<=mid)   modify(x*2,pos,val);
    if (right>=mid+1) modify(x*2+1,pos,val);
    update(x);
  }
}
```

线段树的实现：区间修改?

如果我们要修改一个区间？看了区间查询的代码收到启发，能不能也按照单点修改的样子修改一个区间。

```
void modify(int x,int left,int right,int val){
  if (left<=tree[x].left && tree[x].right<=right){
    tree[x].sum+=val;
  }else{
    int mid=(tree[x].left+tree[x].right)/2;
    if (left<=mid)  modify(x*2,pos,val);
    if (right>=mid+1) modify(x*2+1,pos,val);
    update(x);
  }
}
```

这样子复杂度是不能保证的 OwQ

线段树的实现：区间修改?

如果我们要修改一个区间？看了区间查询的代码收到启发，能不能也按照单点修改的样子修改一个区间。

```
void modify(int x,int left,int right,int val){
  if (left<=tree[x].left && tree[x].right<=right){
    tree[x].sum+=val;
  }else{
    int mid=(tree[x].left+tree[x].right)/2;
    if (left<=mid)   modify(x*2,pos,val);
    if (right>=mid+1) modify(x*2+1,pos,val);
    update(x);
  }
}
```

这样子复杂度是不能保证的 OwQ
解决方案?

如果我们要修改一个区间？看了区间查询的代码收到启发，能不能也按照单点修改的样子修改一个区间。

```
void modify(int x,int left,int right,int val){
  if (left<=tree[x].left && tree[x].right<=right){
    tree[x].sum+=val;
  }else{
    int mid=(tree[x].left+tree[x].right)/2;
    if (left<=mid)  modify(x*2,pos,val);
    if (right>=mid+1) modify(x*2+1,pos,val);
    update(x);
  }
}
```

这样子复杂度是不能保证的 OwQ
解决方案？LazyTag！

LazyTag：懒标记

线段树有修改和询问的操作，如果我们不询问的时候，线段树内部是否更新完毕和我们是没有关系的，于是我们可以引入一个懒标记的概念。即，在区间修改的时候，如果一个节点所管理的区间被这个修改完全覆盖了，那么我们给这个节点打上一个懒标记，表示这个节点所管理的区间全部需要修改，但是不修改这个节点的子节点们，知道下一次查询需要查询或者修改这个节点管理的区间的子区间时，我们连同这个这次的操作和之前的懒标记一并做。（因为我们可以发现，查询和修改对于区间的访问操作时相同的）

LazyTag：懒标记

线段树有修改和询问的操作，如果我们不询问的时候，线段树内部是否更新完毕和我们是没有关系的，于是我们可以引入一个懒标记的概念。即，在区间修改的时候，如果一个节点所管理的区间被这个修改完全覆盖了，那么我们给这个节点打上一个懒标记，表示这个节点所管理的区间全部需要修改，但是不修改这个节点的子节点们，知道下一次查询需要查询或者修改这个节点管理的区间的子区间时，我们连同这个这次的操作和之前的懒标记一并做。（因为我们可以发现，查询和修改对于区间的访问操作时相同的）

画图模拟模拟

线段树有修改和询问的操作，如果我们不询问的时候，线段树内部是否更新完毕和我们是没有关系的，于是我们可以引入一个懒标记的概念。即，在区间修改的时候，如果一个节点所管理的区间被这个修改完全覆盖了，那么我们给这个节点打上一个懒标记，表示这个节点所管理的区间全部需要修改，但是不修改这个节点的子节点们，知道下一次查询需要查询或者修改这个节点管理的区间的子区间时，我们连同这个这次的操作和之前的懒标记一并做。（因为我们可以发现，查询和修改对于区间的访问操作时相同的）

画图模拟模拟

相比较单点修改的线段树，我们多了一个懒标记下推到子节点的操作

这里，演示一个区间赋值修改，区间查询和的线段树。

线段树的实现：结构体

与单点相比，增加了 Tag

```
struct Btree{
  int tag;
  int sum;
  int left,right;
  inline int len(){
    return right-left+1;
  }
};
Btree tree[Maxn*4+5];
```

这就是我们之前所说的懒标记下推，把父节点的懒标记下推到子节点，然后更新子节点所管理的区间和子节点的懒标记，最后清除父节点的懒标记，我年轻的时候喜欢叫 clean，现在想想还是叫 pushDown 好

```
void clean(int x){
  if (tree[x].left==tree[x].right) return;
  int tag=tree[x].tag;
  tree[x].tag=0;
  if (tag!=0){
    tree[x*2].sum=tree[x*2].len()*tag;
    tree[x*2+1].sum=tree[x*2+1].len()*tag;
    tree[x*2].tag=tree[x*2+1].tag=tag;
  }
}
```

线段树的实现：合并区间

与单点线段树相同

```
inline void update(int x){
  if (tree[x].left==tree[x].right) return;
  tree[x].sum=tree[x*2].sum+tree[x*2+1].sum;
}
```

与单点无差

```
void build(int x,int left,int right){
  tree[x].left=left;
  tree[x].right=right;
  tree[x].sum=tree[x].tag=0;
  if (left==right){
    tree[x].sum=1;
  }else{
    int mid=(left+right)/2;
    build(x*2,left,mid);
    build(x*2+1,mid+1,right);
    update(x);
  }
}
```

多了区间打标记和下推标记的操作

```
void modify(int x,int left,int right,int val){
  if (left<=tree[x].left && tree[x].right<=right){
    tree[x].sum=tree[x].len()*val;
    tree[x].tag=val;
  }else{
    clean(x);
    int mid=(tree[x].left+tree[x].right)/2;
    if (left<=mid) modify(x*2,left,right,val);
    if (mid+1<=right) modify(x*2+1,left,right,val);
    update(x);
  }
}
```

线段树的实现：区间查询

多了下推标记的操作

```
int query(int x,int left,int right){
  if (left<=tree[x].left && tree[x].right<=right){
    return tree[x].sum;
  }else{
    clean(x);
    int mid=(tree[x].left+tree[x].right)/2;
    int ret=0;
    if (left<=mid) ret+=query(x*2,left,right);
    if (mid+1<=right) ret+=query(x*2+1,left,right);
    return ret;
  }
}
```

例题 4-5

有一颗 n 个节点的树（$n \leq 50000$），有 m 个操作／询问（$m \leq 50000$）。操作是，选择一个点，把这个点为根的子树上的所有节点的值更新为 x。询问是，询问一个节点的当前值。(初始时，所有点的值为 -1）

来源 HDU3974

例题 4-5

有一颗 n 个节点的树（$n \leq 50000$），有 m 个操作／询问（$m \leq 50000$）。操作是，选择一个点，把这个点为根的子树上的所有节点的值更新为 x。询问是，询问一个节点的当前值。(初始时，所有点的值为 -1)

来源 HDU3974

DFS 序就是 DFS 整棵树依次访问到的结点组成的序列。DFS 序有一个很强的性质: 一颗子树的所有节点在 DFS 序内是连续的一段。

例题 4-5

有一颗 n 个节点的树（$n \le 50000$），有 m 个操作／询问（$m \le 50000$）。操作是，选择一个点，把这个点为根的子树上的所有节点的值更新为 x。询问是，询问一个节点的当前值。(初始时，所有点的值为 -1）

来源 HDU3974

DFS 序就是 DFS 整棵树依次访问到的结点组成的序列。DFS 序有一个很强的性质: 一颗子树的所有节点在 DFS 序内是连续的一段。

把节点按照 DFS 序编号，然后按照节点编号建立线段树。然后对树上子树的操作就对应线段树上一个区间的操作，询问就是单点询问。

例题 4-5

一个长度为 n 的序列，有 m 个操作。要求支持区间加，区间乘，区间赋值，区间求和，区间求每个元素平方的和，区间求每个元素立方的和。（$n, m \leq 100000$）

来源 HDU4578

例题 4-5

一个长度为 n 的序列，有 m 个操作。要求支持区间加，区间乘，区间赋值，区间求和，区间求每个元素平方的和，区间求每个元素立方的和。($n, m \leq 100000$)

来源 HDU4578

线段树！打 Tag！每个操作各要打一个 Tag，要注意标记之间执行的顺序。

例题 4-5

一个长度为 n 的序列，有 m 个操作。要求支持区间加，区间乘，区间赋值，区间求和，区间求每个元素平方的和，区间求每个元素立方的和。（$n, m \le 100000$）

来源 HDU4578

线段树！打 Tag！每个操作各要打一个 Tag，要注意标记之间执行的顺序。

加法标记下清的方法，把 $(num + x)^3, (num + x)^2$ 拆开化简就很明了了。

例题 4-6

一个长度为 n 的序列，有 m 个操作。要求支持区间开方，区间求和。$n, m \leq 100000$）

来源 HDU4027

例题 4-6

一个长度为 n 的序列，有 m 个操作。要求支持区间开方，区间求和。$n, m \leq 100000$）

来源 HDU4027

开方操作不支持合并，只能单点修改。

例题 4-6

一个长度为 n 的序列，有 m 个操作。要求支持区间开方，区间求和。$n, m \le 100000$）

来源 HDU4027

开方操作不支持合并，只能单点修改。

一个数字被开方若干次变为 1，我们可以不用操作元素已经全部变为 1 的区间。

例题 4-6

一个长度为 n 的序列，有 m 个操作。要求支持区间开方，区间求和。$n, m \leq 100000$）

来源 HDU4027

开方操作不支持合并，只能单点修改。

一个数字被开方若干次变为 1，我们可以不用操作元素已经全部变为 1 的区间。

复杂度合理。

Extra Part

拓展

仅介绍一些名词

- 滋糍区间加，区间求和的树状数组
- 滋糍单点修改，区间查询最值的树状数组
- 动态开点线段树
- 主席树（可持久化线段树）（函数式线段树）
- 二维线段树（树套树）

Coda

Thanks