初级数据结构 2

水橋パルスィ

Tsinghua University

2021 年 2 月 16 日

树状数组

用于高效维护前缀信息。
需要满足信息可高效合并。
维护区间信息需要满足信息可减。

树状数组

经典使用场景：

给定包含 n 个数的数组 $a_1, a_2, \cdots, a_n$ 。有两种操作：

1. 把 a_x 增加 y 。
2. 查询区间 $[l, r]$ 中数的和。

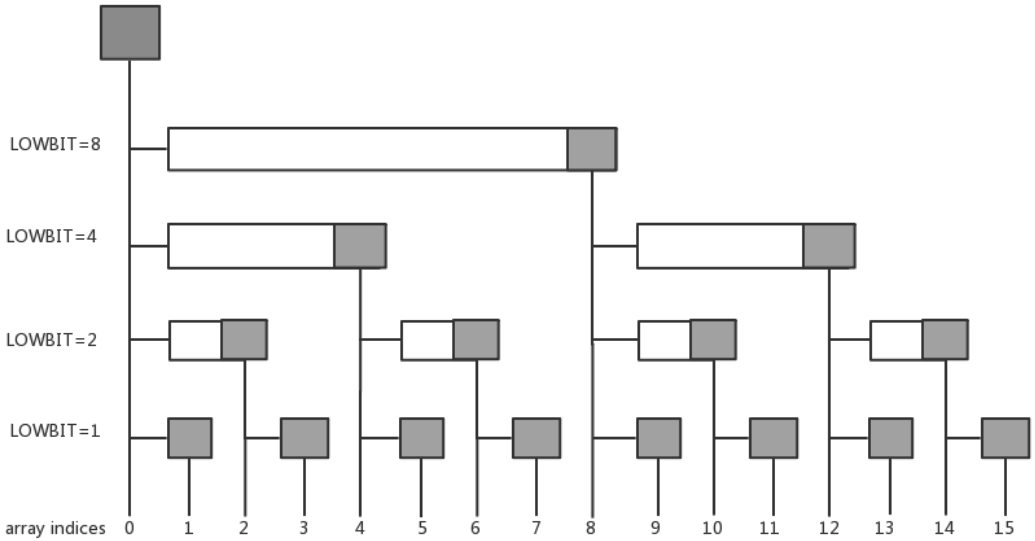
lowbit

对于一个正整数 x，定义 $lowbit(x)$ 为 x 的二进制表示中最右边的 1 所对应的值。

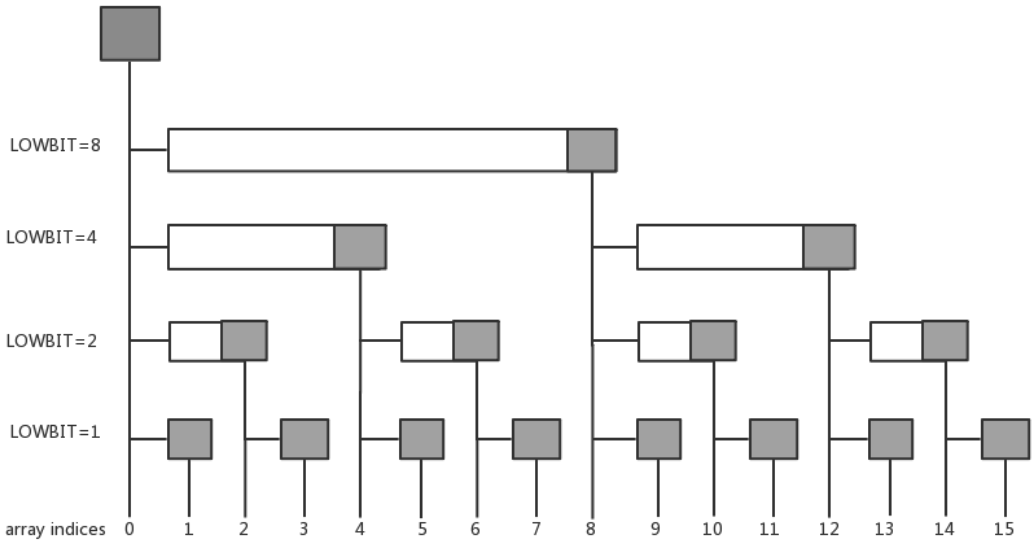
$lowbit(x) = x\& - x$

树状数组

在树状数组中，结点 i 表示的区间为 $[i - lowbit(i) + 1, i]$ 。

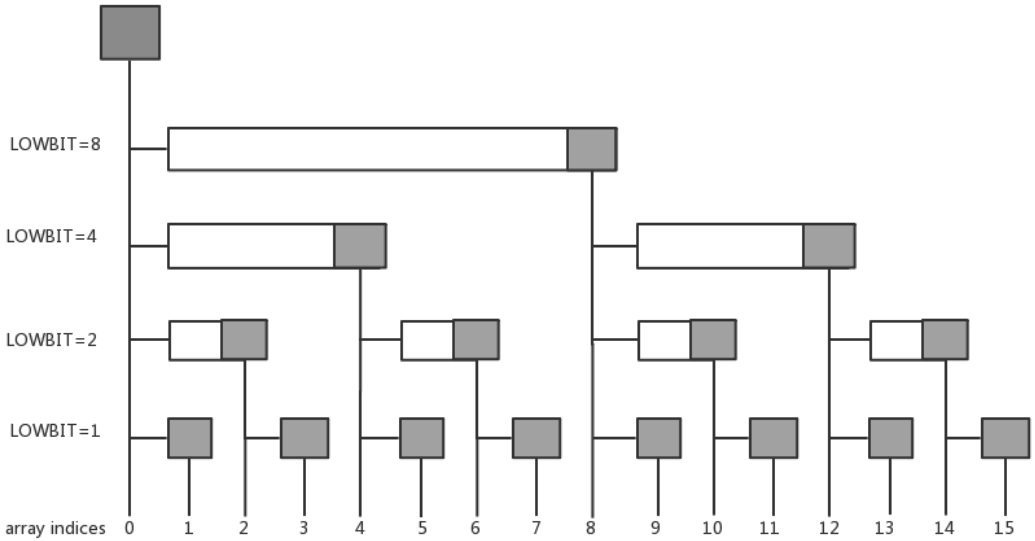


树状数组



同层结点表示区间互不相交；
左儿子表示区间为父亲的子集；
右儿子表示区间和父亲不相交。

树状数组



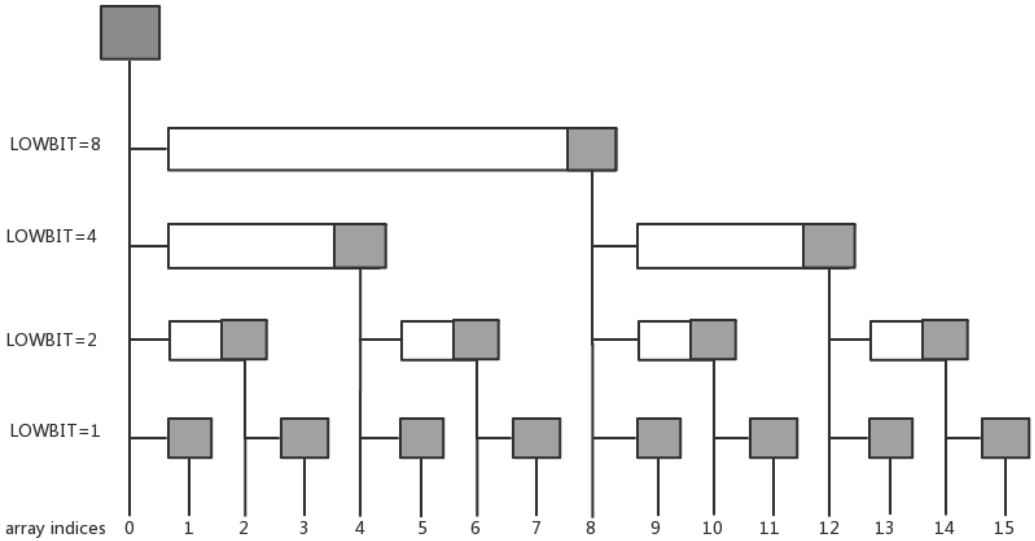
修改时：寻找包含 x 的所有区间。

从 x 点向上走；

若当前点是左儿子，则父亲结点包含 x；

若当前点是右儿子，则父亲结点不包含 x。

树状数组



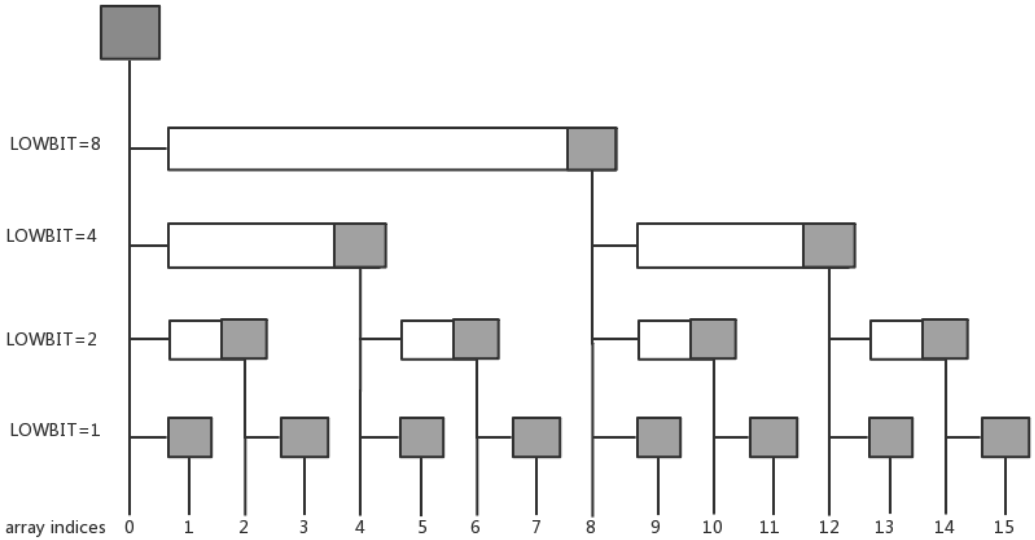
修改时：

寻找第一个大于 x 且是 x 祖先的数。

$=>$ 寻找最小的大于 x 且 $lowbit$ 大于 x 的数。

则至少加 $lowbit(x)$ ，且满足条件的最小数就是 $x + lowbit(x)$ 。

树状数组



查询时：寻找若干不相交区间使其并集为 $[1, x]$ 。
首先取出以 x 结尾的区间 $[x - lowbit(x) + 1, x]$ ；
则只需要继续求 $[1, x - lowbit(x)]$ 。
在树上相当于不断向左走。

树状数组

树高为 $O(\log_2 n)$ 。
修改和查询均只会使用每层最多一个点。
故时间复杂度均为 $O(\log n)$ 。
空间复杂度 $O(n)$ 。

树状数组

单点加法，查询区间和：

$c_i = \sum_{i-lowbit(i)+1}^{i} a_i$ 。

树状数组

```
void update(int x, int y) {
    for (; x <= n; x += x & -x) c[x] += y;
}

int getsum(int x) {
    int res = 0;
    for (; x; x -= x & -x) res += c[x];
    return res;
}
```

树状数组

将 a_x 修改为 y：

$update(x, y - a[x])$;

查询区间 $[l, r]$ 的和：

$return\ getsum(r) - getsum(l - 1)$;

树状数组

区间加法，查询单点值：

$c_i = \sum_{i-lowbit(i)+1}^{i}(a_i - a_{i-1})$ 。

令区间 $[l, r]$ 加上 y ：

$update(r+1, -y); update(l, y);$

查询 x 点的值：

$return\ getsum(x);$

树状数组

区间加法，查询区间和：

$$\begin{aligned}\sum_{i=1}^{x} a_i &= \sum_{i=1}^{x}\sum_{j=1}^{i}(a_j - a_{j-1}) \\ &= \sum_{j=1}^{x}(x-j+1)(a_j - a_{j-1}) \\ &= (x+1)\sum_{i=1}^{x}(a_i - a_{i-1}) - \sum_{i=1}^{x} i(a_i - a_{i-1})\end{aligned}$$

$c_i = \sum_{i-lowbit(i)+1}^{i}(a_i - a_{i-1})$ 。
$d_i = \sum_{i-lowbit(i)+1}^{i} i(a_i - a_{i-1})$ 。

线段树

用于高效维护区间信息。
需要满足信息可高效合并。

线段树

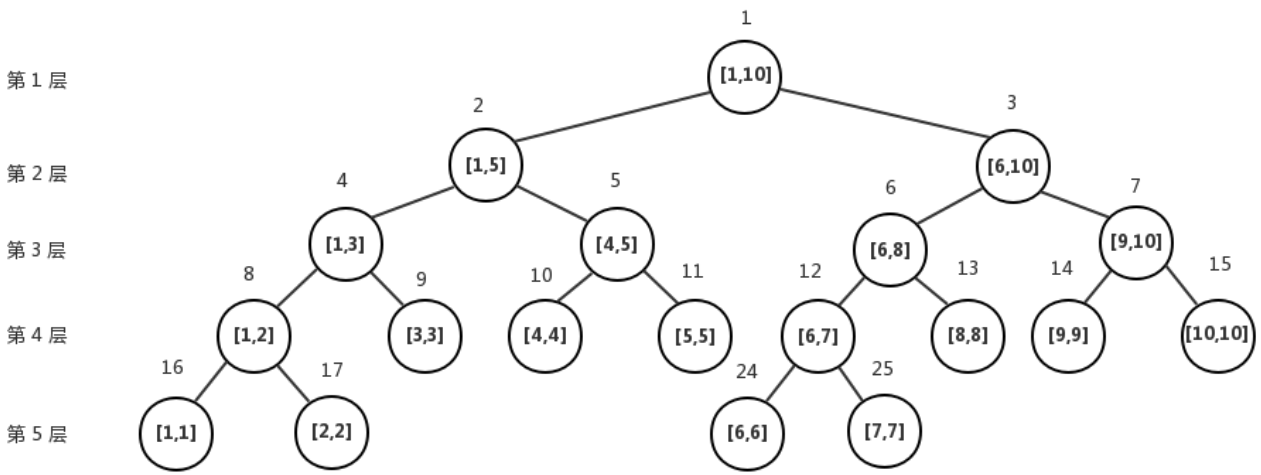
经典使用场景：

给定包含 n 个数的数组 $a_1, a_2, \cdots, a_n$ 。有两种操作：

1. 给区间 $[l, r]$ 中的数增加 x 。
2. 查询区间 $[l, r]$ 中数的最大值。

线段树

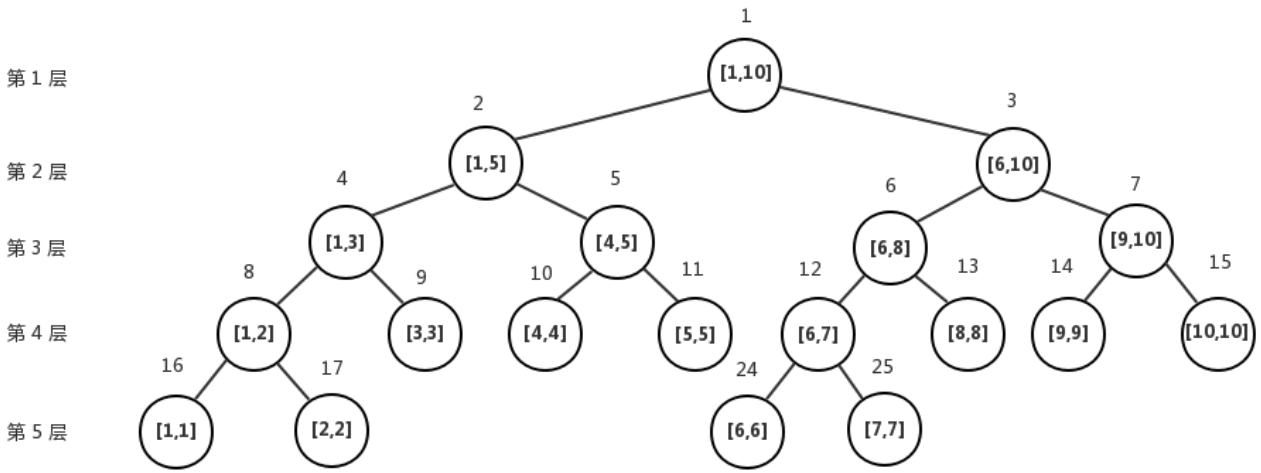
在线段树中，每个结点表示的区间由以下规则确定：
根结点表示的区间为 $[1, n]$；
若一个点表示的区间为 $[l, r]$，且 $l \neq r$，则其有两个儿子结点，取 $m = \lfloor \frac{l+r}{2} \rfloor$，左儿子表示 $[l, m]$，右儿子表示 $[m+1, r]$。



线段树

由此划分后，每一层的最大区间长度为上一层的一半。
故树高 $O(\log n)$ 。
结点数为 $(2n-1)$ 。
常用的编号方法：根结点为 1 ，x 的左儿子为 $2x$ ，右儿子为 $2x+1$ 。
采取这种编号方法需要开 $4n$ 的空间。

线段树

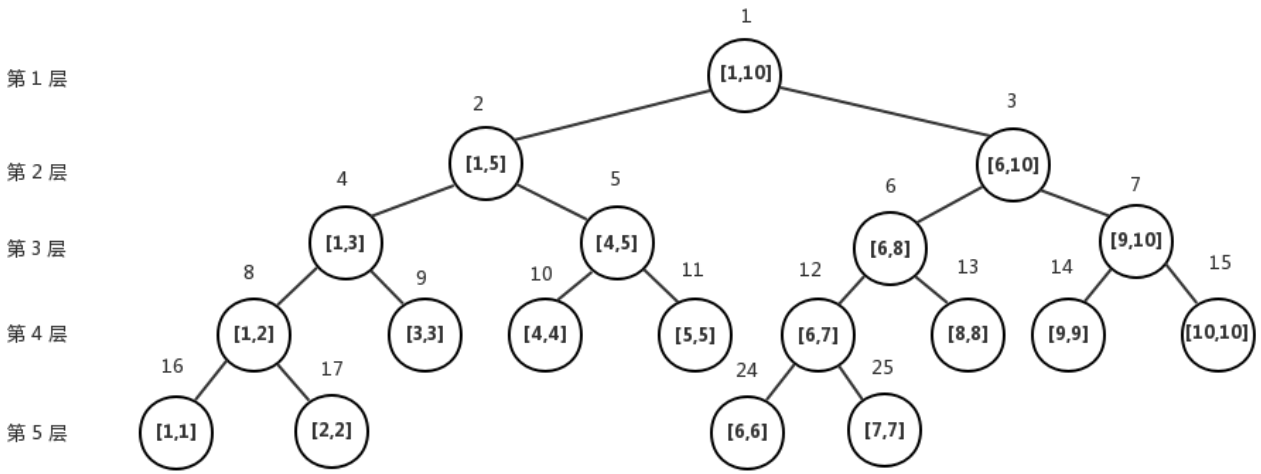


单点修改：将该点位置的值修改后，向上更新。

单点修改

```
void modify(int l, int r, int x, int y, int k) {
    if (l == r) {
        v[k] = y;
        return;
    }
    int m = l + r >> 1;
    if (x <= m)
        modify(l, m, x, y, k << 1);
    else
        modify(m + 1, r, x, y, k << 1 | 1);
    update(k);
}
```

线段树

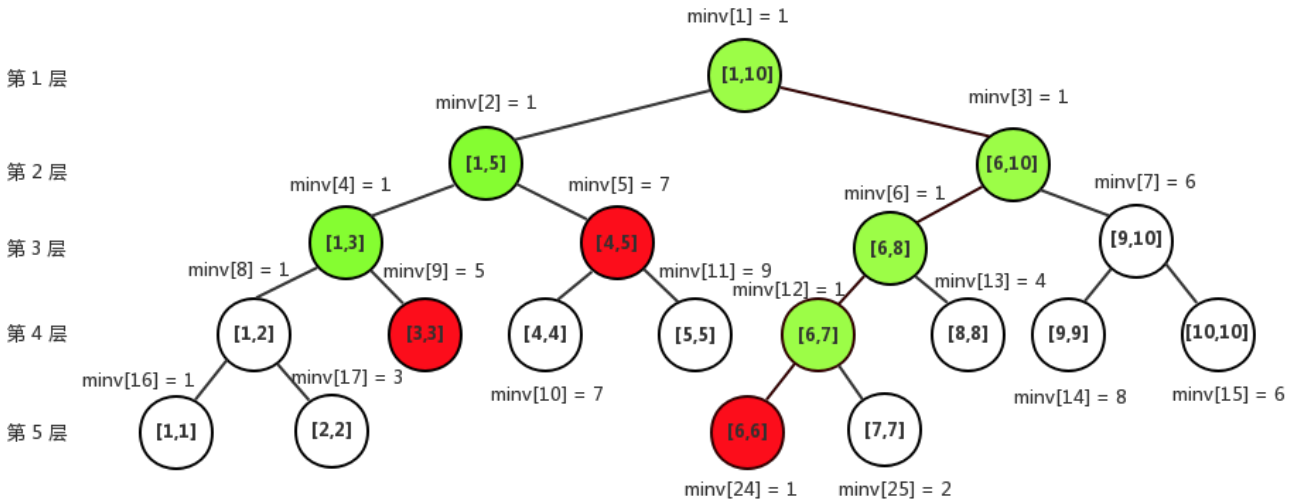


单点查询：直接找到该点的值。

单点查询

```
int query(int l, int r, int x, int k) {
    if (l == r) return v[k];
    int m = l + r >> 1;
    if (x <= m)
        return query(l, m, x, k << 1);
    else
        return query(m + 1, r, x, y, k << 1 | 1);
}
```

线段树



区间查询：寻找尽可能少的不相交区间使其并集为 $[l, r]$ 。

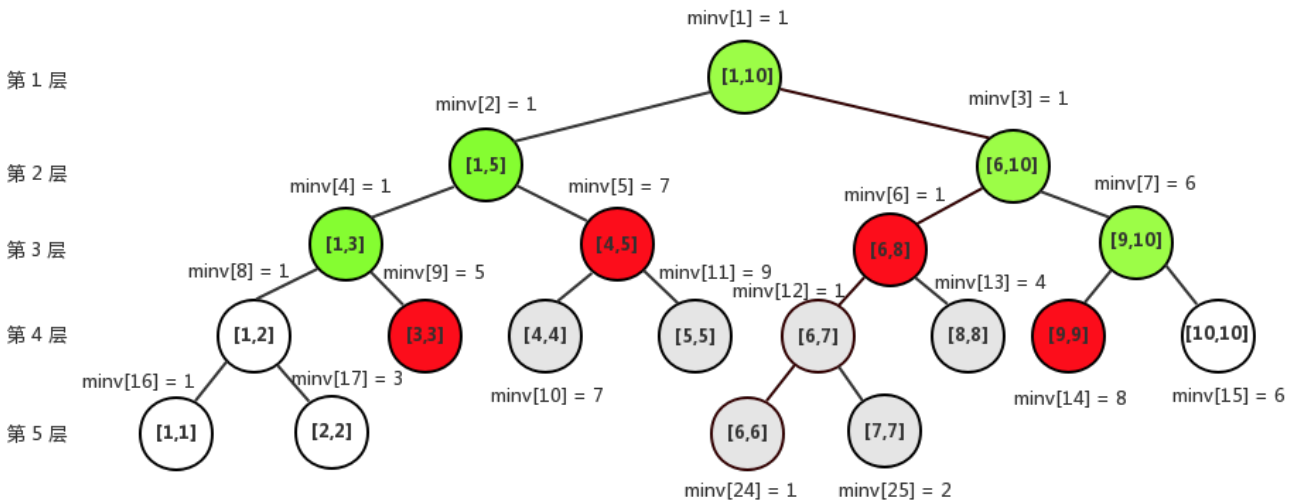
从上往下进行寻找。

每一层绿色/红色的点均不超过两个。

区间查询（以区间和为例）

```
int query(int l, int r, int ql, int qr, int k) {
    if (l == ql && r == qr) return v[k];
    int m = l + r >> 1, res = 0;
    if (ql <= m)
        res += query(l, m, ql, min(m, qr), k << 1);
    if (qr > m)
        res += query(m + 1, r, max(m + 1, ql), qr, k << 1 | 1);
    return res;
}
```

线段树



区间修改：与区间查询操作的结点相同。
仅修改了红色结点，灰色结点的值不正确。
在红色结点处打上标记，经过时下传。

区间修改区间查询（以区间加法区间最大值为例）

```
void update(int x) { v[x] = max(v[x << 1], v[x << 1 | 1]); }
void pushdown(int x) {
    v[x << 1] += lazy[x];
    lazy[x << 1] += lazy[x];
    v[x << 1 | 1] += lazy[x];
    lazy[x << 1 | 1] += lazy[x];
    lazy[x] = 0;
}
int query(int l, int r, int ql, int qr, int k) {
    if (l == ql && r == qr) return v[k];
    pushdown(k);
    int m = l + r >> 1, res = INT_MIN;
    if (ql <= m) res = max(res, query(l, m, ql, min(m, qr), k << 1));
    if (qr > m) res = max(res, query(m + 1, r, max(m + 1, ql), qr, k << 1 | 1));
    return res;
}
void modify(int l, int r, int ql, int qr, int y, int k) {
    if (l == ql && r == qr) {
        v[k] += y;
        lazy[k] += y;
        return;
    }
    pushdown(k);
    int m = l + r >> 1;
    if (ql <= m) modify(l, m, ql, min(m, qr), y, k << 1);
    if (qr > m) modify(m + 1, r, max(m + 1, ql), qr, y, k << 1 | 1);
    update(k);
}
```

zkw 线段树

自底向上访问结点的线段树。
非递归，效率高，代码短。
容易出错，某些功能实现不方便，不能进行标记下放，需要标记永久化。

树套树

二维树状数组/二维线段树；
线段树 + 平衡树；
线段树 + 前缀和（主席树）；
线段树 + 树状数组（可修改主席树）；
kd-tree？堆？ · · ·

DFS 序线段树

按照 DFS 的顺序给一棵树的结点进行编号，那么一个子树内的结点编号是一段区间。
以 DFS 序建立线段树维护信息，则对于子树的操作相当于在线段树上进行区间操作。

P3178 [HAOI2015] 树上操作

Problem

有一棵点数为 N 的树，以点 1 为根，且树点有边权。然后有 M 个操作，分为三种：
操作 1：把某个节点 x 的点权增加 a 。
操作 2：把某个节点 x 为根的子树中所有点的点权都增加 a 。
操作 3：询问某个节点 x 到根的路径中所有点的点权和。

Data Range

对于 100% 的数据，有 $1 \leq N, M \leq 100000$ 。

P3178 [HAOI2015] 树上操作

记点 i 到根结点的路径中所有点的点权和为 d_i 。
对于操作 1，将 x 的子树中所有点的 d_i 加上 a 。
对于操作 2，将 x 的子树中所有点的 d_i 加上 $(dep_i - dep_x + 1)a = dep_i a + (1 - dep_x)a$ 。
后半部分与操作 1 等价，对前半部分只需要预处理出线段树每个结点的区间 dep 和即可。

P4145 上帝造题的七分钟 2

Problem

区间开根号区间求和。

Data Range

对于 100% 的数据，有 $1 \leq N, M \leq 100000$ 。

P4145 上帝造题的七分钟 2

一个数最多开 6 次平方就会变成 1 。
给已经变成 1 的区间打上标记，之后修改经过这些区间时不用继续向下。

矩形面积并

Problem

给定平面上的若干个矩形，求总的覆盖面积。

矩形面积并

扫描线。

画图 time。

区间第 k 小

Problem

给定数列，查询区间第 k 小数字。

Hotel

Problem

给定数列，支持区间置为 0 或 1，查询全局最大连续长度的 0。

mex

Problem

给定数列，询问区间最小的没出现过的数字。

SDOI2009 HH 的项链

Problem

给定数列，支持询问区间内有多少种不同的数字。

BZOJ2819 nim

Problem

给定一棵树，有点权，每次修改或询问一条链上的点权玩 nim 游戏，问是否先手必胜。

主席树

又名可持久化线段树，可理解为值域线段树的前缀和。
可用于查询区间第 k 大等。

值域线段树

将出现的值离散化，线段树上对应区间 $[L, R]$ 的点记录 $[L, R]$ 内的数字总共出现了多少次。
在值域线段树上查询第 k 大，只要从根结点开始每次判断向左还是向右走即可。

前缀和

对于任意一个区间 $[l, r]$，只要得到其值域线段树就可以直接查询其第 k 大。

保存每个前缀的值域线段树，记 T_i 为 $[1, i]$ 的值域线段树，那么 $[l, r]$ 的值域线段树就是 $T_r - T_{l-1}$。

优化

显然不能直接开 n 棵线段树。
考虑第 $i+1$ 棵线段树和第 i 棵的区别：只多了一个数，将某个叶子结点到根结点的链权值 $+1$ 。
也就是只有一条链发生了变化。只要新建这一条链，其他的仍然使用上一棵树的对应结点即可。
画图 time

P3834 【模板】可持久化线段树 1（主席树）

Problem

如题，给定 N 个整数构成的序列，将对于指定的闭区间查询其区间内的第 K 小值。

Data Range

对于 100% 的数据，$n, m \leq 200000$ 。

口胡题目

给一棵树，每次询问链上第 k 大。

口胡题目

给一棵树，每次询问链上第 k 大。
树剖后建主席树。（多一个 log）
在树上建主席树（每个点的树由其父亲改一条链得到），将询问表示为四段同时查询。

P2617 Dynamic Rankings

Problem

给定一个含有 n 个数的序列 $a[1], a[2], a[3]...a[n]$，程序必须回答这样的询问：对于给定的 i, j, k，在 $a[i], a[i+1], a[i+2]...a[j]$ 中第 k 小的数是多少 $(1 \leq k \leq j-i+1)$，并且，你可以改变一些 $a[i]$ 的值，改变后，程序还能针对改变后的 a 继续回答上面的问题。你需要编一个这样的程序，从输入文件中读入序列 a，然后读入一系列的指令，包括询问指令和修改指令。
对于每一个询问指令，你必须输出正确的回答。

Data Range

对于 100% 的数据，$n, m \leq 100000$ 。

P2617 Dynamic Rankings

用树状数组维护前缀和。
查询和操作的复杂度均为 $O(log^2 n)$ 。
(每次对于 $O(logn)$ 棵线段树同时进行操作)

P2617 Dynamic Rankings

用树状数组维护前缀和。
查询和操作的复杂度均为 $O(log^2 n)$ 。
(每次对于 $O(logn)$ 棵线段树同时进行操作)

代码 time
(也可以用整体二分等方法)

Trie

单独一个 Trie 并没有很大用处。
建议报名省选组，学习更多字符串知识（确信）。

堆

一般情况下使用 std::priority_queue 即可。
并不需要学习手写堆。

可并堆

普通的堆进行合并需要将其中一个堆内的元素逐个插入另一个堆，效率极低。
支持快速合并的堆称为可并堆。

可并堆

常见的可并堆：

左偏树

配对堆

斜堆

斐波那契堆

回顾

堆（假设为小根堆）：
基本结构为一棵树；
一个点的值不大于其子树内任意一个点。

左偏树

满足堆性质；
二叉树；
对于一个点 i，其左儿子为 l_i，右儿子为 r_i，定义其 dis 值为 $dis_i = min(dis_{l_i}, dis_{r_i}) + 1$，空结点 dis 为 0。
对于左偏树，规定其 $dis_{l_i} \geq dis_{r_i}$（左偏性质），则显然有 $dis_i = dis_{r_i} + 1$，并有以下推论：
假设左偏树中 dis 最大的值为 k，则其子树内至少有 $2^k - 1$ 个点。
故对于 n 个点的左偏树，dis 值不超过 $\log_2(n+1)$。

合并操作

合并以 x 和 y 为根的两个左偏树 $merge(x, y)$ ：
不妨假设为小根堆且 $v_x \leq v_y$ ，若 $v_x > v_y$ 则交换 x, y ；
递归执行 $merge(r_x, y)$ 即可。
回溯时判断是否需要交换左右儿子以满足左偏性质。

合并操作

合并以 x 和 y 为根的两个左偏树 $merge(x, y)$ ：
不妨假设为小根堆且 $v_x \leq v_y$ ，若 $v_x > v_y$ 则交换 x, y ；
递归执行 $merge(r_x, y)$ 即可。
回溯时判断是否需要交换左右儿子以满足左偏性质。
每次递归调用 $merge(x, y)$ ，$(x + y)$ 的值减 1 。
故层数为 $O(\log n)$ ，即合并的复杂度为 $O(\log n)$ 。

其他操作

插入：将新的点当作一个左偏树进行合并。
弹出最值：删除堆顶后合并其左右儿子。

配对堆

满足堆性质；
以左儿子右兄弟的形式存边；
通过特殊的 pop 操作来保证复杂度。

合并操作

合并以 x 和 y 为根的两个配对堆 $merge(x, y)$ ：
不妨假设为小根堆且 $v_x \leq v_y$ ，若 $v_x > v_y$ 则交换 x, y ；
直接将 y 接在 x 下。
插入：将新的点当作一个配对堆进行合并。

弹出最值操作

弹出以 x 为根的配对堆的堆顶 $pop(x)$ ：
对于 x 的所有儿子 $s_1, s_2, \ldots, s_t$ ，
首先对于所有 k 执行 $merge(2k+1, 2k+2)$ ，得到 $\lceil \frac{t}{2} \rceil$ 个堆；
将这些堆从后往前依次合并得到新的堆。

配对堆

```
void ins(int x,int y){a[y].r=a[x].s;a[x].s=y;}
void kill(int x){a[root].s=a[x].r;a[x].r=0;}
int merge(int x,int y){
    if(!x||!y)return x+y;
    if(a[x].x>a[y].x)swap(x,y);
    ins(x,y);return x;
}
void push(int x){
    a[x].s=a[x].r=0;
    root=merge(root,x);
}
int del(){
    if(!a[root].s)return 0;
    int x=a[root].s,y=a[x].r;
    kill(x);if(y)kill(y);
    return merge(merge(x,y),del());
}
int pop(){
    int r=a[root].x;
    root=del();
    return r;
}
int top(){return a[root].x;}
```

复杂度比较

合并、插入：左偏树 $O(\log n)$ ，配对堆 $O(1)$ 。
弹出堆顶：均为 $O(\log n)$ 。
删除元素、$decrease-key$ ：均为 $O(\log n)$ 。

斐波那契堆

常数大，难写，不常用，不做详细介绍。
配对堆的复杂版，将 $decrease-key$ 优化到了 $O(1)$ 。
堆优化 Dijkstra：
普通堆：$m \log n$ ；
系统堆（不支持 $decrease-key$ ）：$m \log m$ ；
斐波那契堆：$n \log n + m$ 。

斜堆

左偏树的简化版。
不记录 dis 值，每次回溯时都交换左右儿子。
最坏复杂度 $O(n)$ 。

pb_ds

```
#include <ext/pb_ds/priority_queue.hpp>
using namespace pb_ds;
__gnu_pbds::priority_queue <T,cmp(),pairing_heap_tag> q;
```

用法与 std::priority_queue 类似，用 a.join(b) 将堆 b 合并到 a 后清空。
默认配对堆。
支持二叉堆、二项堆、rc 二项堆、改良斐波那契堆。
很多比赛中不能用（如 NOIP），不确定能用最好不用。

罗马游戏

Problem

给定 n 个数，原来每个数都属于不同的集合。
有 m 次操作，操作有两种：
1. 合并第 i 个数和第 j 个数所在的集合（若 i 或 j 已被删除则忽略）；
2. 删除并输出第 i 个数所在集合中最小的数（若 i 已被删除则输出 0 ）。

Data Range

$n \leq 1000000, m \leq 100000$ 。

罗马游戏

用可并堆维护集合。
使用并查集来维护每个数所属的集合对应的堆的根。

派遣

Problem

给定一棵 n 个点的树，每个点有一个薪水和一个领导力；
要求选定一个点 x 和它子树内的若干个点，假设个数为 k，满足这 k 个点的薪水之和不超过预算，求 x 的领导力和 k 的乘积的最大值。

Data Range

$n \leq 100000, m \leq 100000$

派遣

固定 x，则不断选择薪水最小的直到超过预算。
维护大根可并堆，自下而上合并，将超出预算的删除。