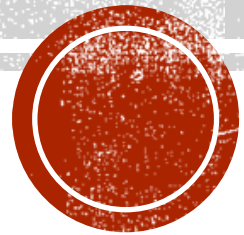
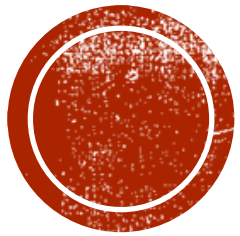
线段树 & 树状数组

赵嘉熠 2021.9.21





线段树

为什么需要线段树？

- **题目：**
  - 给定包含n个数的数组A
  - 进行两种操作
    1. 查询下标在$[l,r]$中的元素的和
    2. 将$[l,r]$中所有元素加C
  - $n \leq 2 \times 10^5$

- **方法一：**
  - 对于修改，直接在数组上改；
  - 对于查询，循环$l \sim r$

- **方法二：**
  - 维护前缀和
  - 对于修改，循环$l \sim r$
  - 对于询问，直接输出前缀和的差

为什么需要线段树？

	方法一	方法二
$\forall i \in [l,r], A_i \mathrel{+}= C$	修改1个元素	修改$r-l+1$个元素
求$\sum_{i=l}^{r} A_i$	计算$r-l+1$个元素	计算2个元素

- 方法一修改快，求和慢；方法二求和快，修改慢
- 是否存在一种数据结构，修改和求和都比较快呢？

线段树

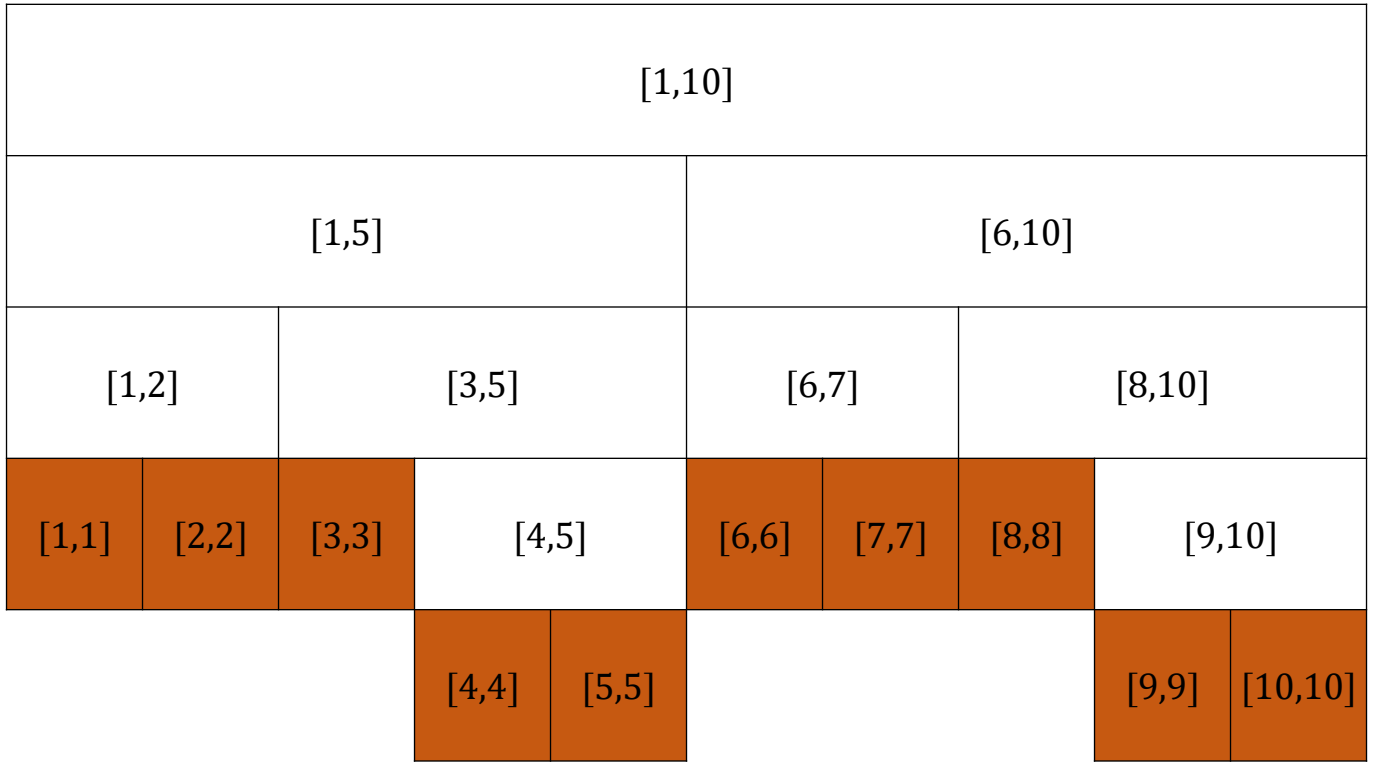
线段树

——概述

- 线段树是算法竞赛中常用的用来维护**区间信息**的数据结构。
- 在$O(\log N)$的时间复杂度内实现单点修改、区间修改、区间查询等操作。

线段树
——结构

- 用**二叉树**来表示线段树，树上每个结点都表示一段**区间的答案。**
- 每个非叶子节点都有左右两个儿子。
- 如果一个结点表示$[l,r]$的答案， 设$mid = \frac{l+r}{2}$。
- 类似于**二分**的思想，左儿子表示$[l,mid]$的答案，右儿子表示$[mid+1,r]$的答案。
- 只要满足$l < r$就继续分下去。

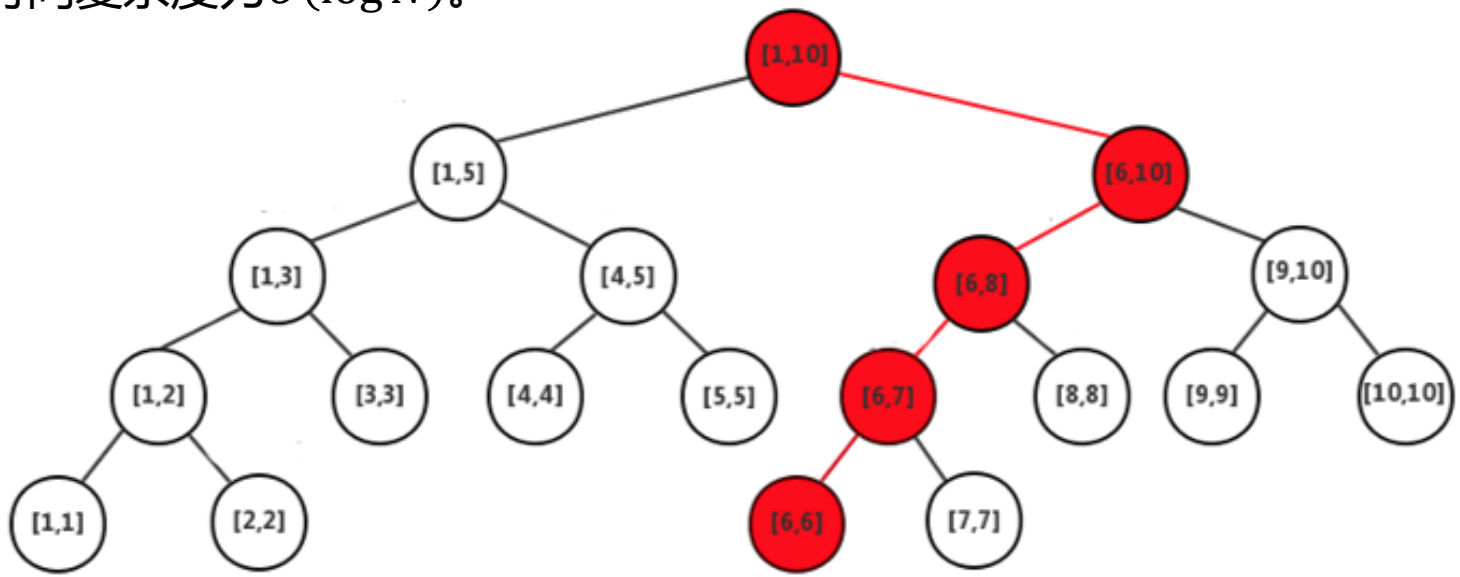


建立线段树

```
1  struct node { //树的结点
2      int sum, maxn, minn;
3  } tree[N << 2];
4  //注意要开四倍空间
5
6  inline void pushup(int id) { //向上更新
7      tree[id].sum = tree[id << 1].sum + tree[id << 1 | 1].sum;
8  }
9
10 void build(int id, int l, int r) {
11     if (l == r) { //如果是子结点
12         tree[id].sum = a[l];
13     }
14     int mid = (l + r) >> 1;
15     build(id << 1, l, mid);
16     build(id << 1 | 1, mid + 1, r);
17     pushup(id);
18 }
```

线段树的单点更新

- **单点更新：**修改指定点的值。
- 一个点的修改，只会影响到包含这个点的区间，而包含这个点的区间实际上是**一条链。**
- 线段树最大深度为$\log N$，故更新一次时间复杂度为$O(\log N)$。

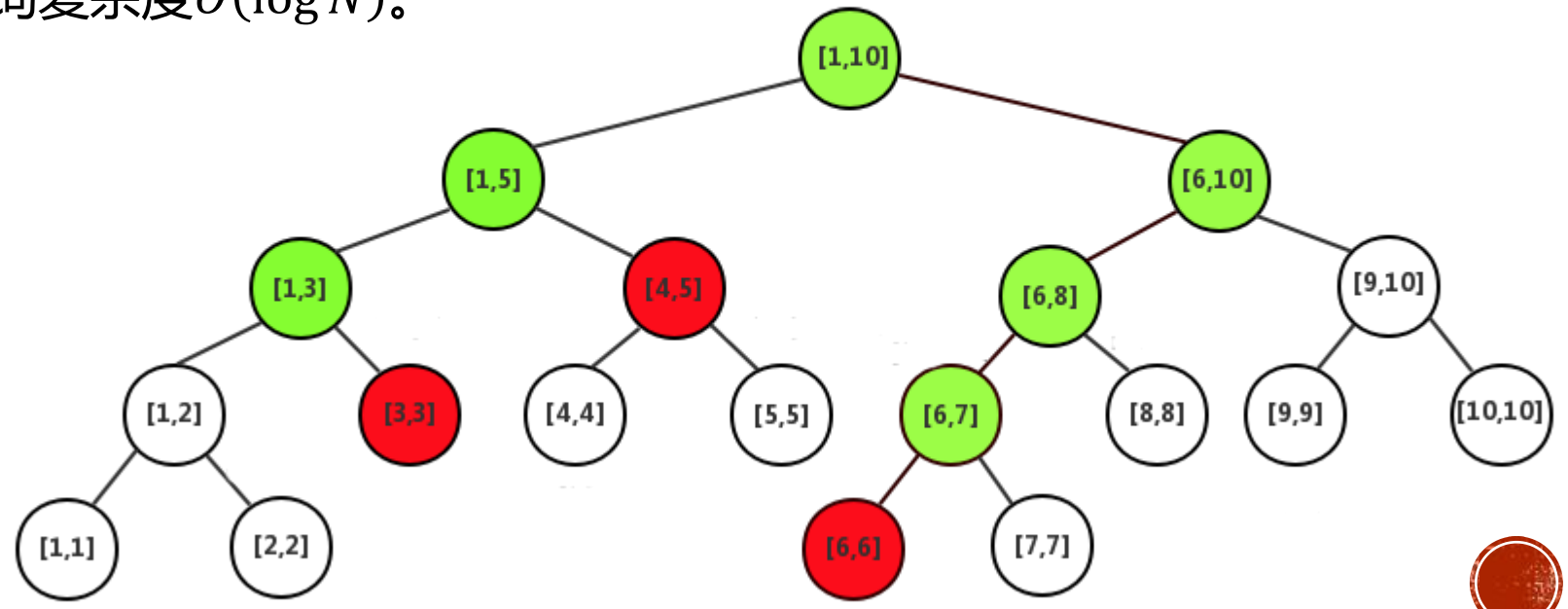


线段树的单点更新

```
1  void update(int id, int l, int r, int x, int v) {
2      if (l == r) {
3          tree[id].sum = v;
4          return;
5      }
6      int mid = (l + r) >> 1;
7      if (x <= mid) {
8          update(id << 1, l, mid, x, v);
9      }
10     else {
11         update(id << 1 | 1, mid + 1, r, x, v);
12     }
13     pushup(id);
14 }
```

线段树的查询

- **单点查询：**与单点更新过程一致，一路走到叶子结点
- **区间查询：**对于查询$[x, y]$，答案就是所有被**完全包含**区间合并起来。
- 例：查询区间[3,6]。绿点表示与被查询的区间$[x, y]$有交集的结点。每层最多有两个绿点（最左端和最右端），所以查询复杂度$O(\log N)$。

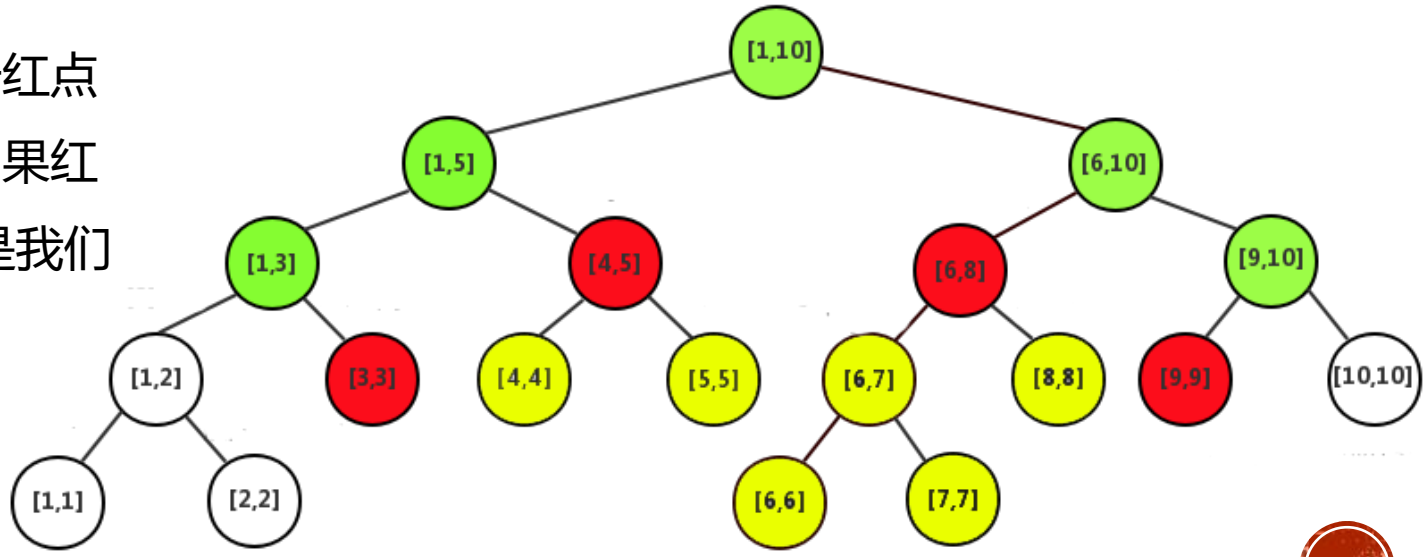


线段树的查询

```
1  int query(int id, int l, int r, int x, int y) {
2      if (x <= l && r <= y) { //完全包含直接返回
3          return tree[id].sum;
4      }
5
6      int mid = (l + r) / 2;
7      int ans = 0;
8      if (x <= mid) { // 如果左区间包含
9          ans += query(id << 1, l, mid, x, y);
10     }
11     if (y > mid) { // 如果右区间包含
12         ans += query(id << 1 | 1, mid + 1, r, x, y);
13     }
14     return ans;
15 }
```

线段树的区间更新

- **区间更新：**更新一个区间，使得区间每个数都变成或加/减/乘/除v。
- **实现思路：**考虑区间查询的思想，把更新的区间划分为线段树上的结点，结点的区间合并起来就是区间$[x, y]$。比如区间[3,9]的分割如右图。
- 对于绿点、红点，可以递归更新；对于红点的子结点即**黄点的信息没有被更新**。如果红点也继续递归，复杂度变成$O(N)$。于是我们引入**懒标记。**



线段树的区间更新
——懒标记

- **懒标记：**通过延迟对节点信息的更改，从而减少可能不必要的操作次数。
- **核心思想：**某个结点需要用到时再去维护其正确性。递归更新时，用父节点的信息更新子结点（pushdown），将懒标记传递下去，然后清空该结点的标记。子结点的懒惰标记是可以**累加**的。
- 实现的时候，递归更新到红点时，将红点的懒标记标记成v即可。时间复杂度$\log N$。

线段树的区间更新

——懒标记

```
1 void pushdown(int id, int l, int r) {
2     if(!tree[id].laz) return;
3     tree[id << 1].laz += tree[id].laz;
4     tree[id << 1 | 1].laz += tree[id].laz;
5     int mid = (l + r) / 2;
6     tree[id << 1].sum += (mid - l + 1) * tree[id].laz;
7     tree[id << 1 | 1].sum += (r - mid) * tree[id].laz;
8     tree[id].laz = 0;
9 }
```

线段树的区间更新
——区间加法

```
1  void update(int id, int l, int r, int x, int y, int v) {
2      if (l >= x && r <= y) {
3          tree[id].sum += (r - l + 1) * v;
4          tree[id].laz += v;
5          return;
6      }
7      pushdown(id, l, r);
8      int mid = (l + r) / 2;
9      if (x <= mid) {
10         update(id << 1, l, mid, x, y, v);
11     }
12     if (y > mid) {
13         update(id << 1 | 1, mid + 1, r, x, y, v);
14     }
15     pushup(id);
16 }
```

线段树的区间更新
——区间查询的变化

```
int query(int id, int l, int r, int x, int y) {
    if (x <= l && r <= y) { //完全包含直接返回
        return tree[id].sum;
    }
    pushdown(id, l, r);
    int mid = (l + r) / 2;
    int ans = 0;
    if (x <= mid) { // 如果左区间包含
        ans += query(id << 1, l, mid, x, y);
    }
    if (y > mid) { // 如果右区间包含
        ans += query(id << 1 | 1, mid + 1, r, x, y);
    }
    return ans;
}
```

二维线段树

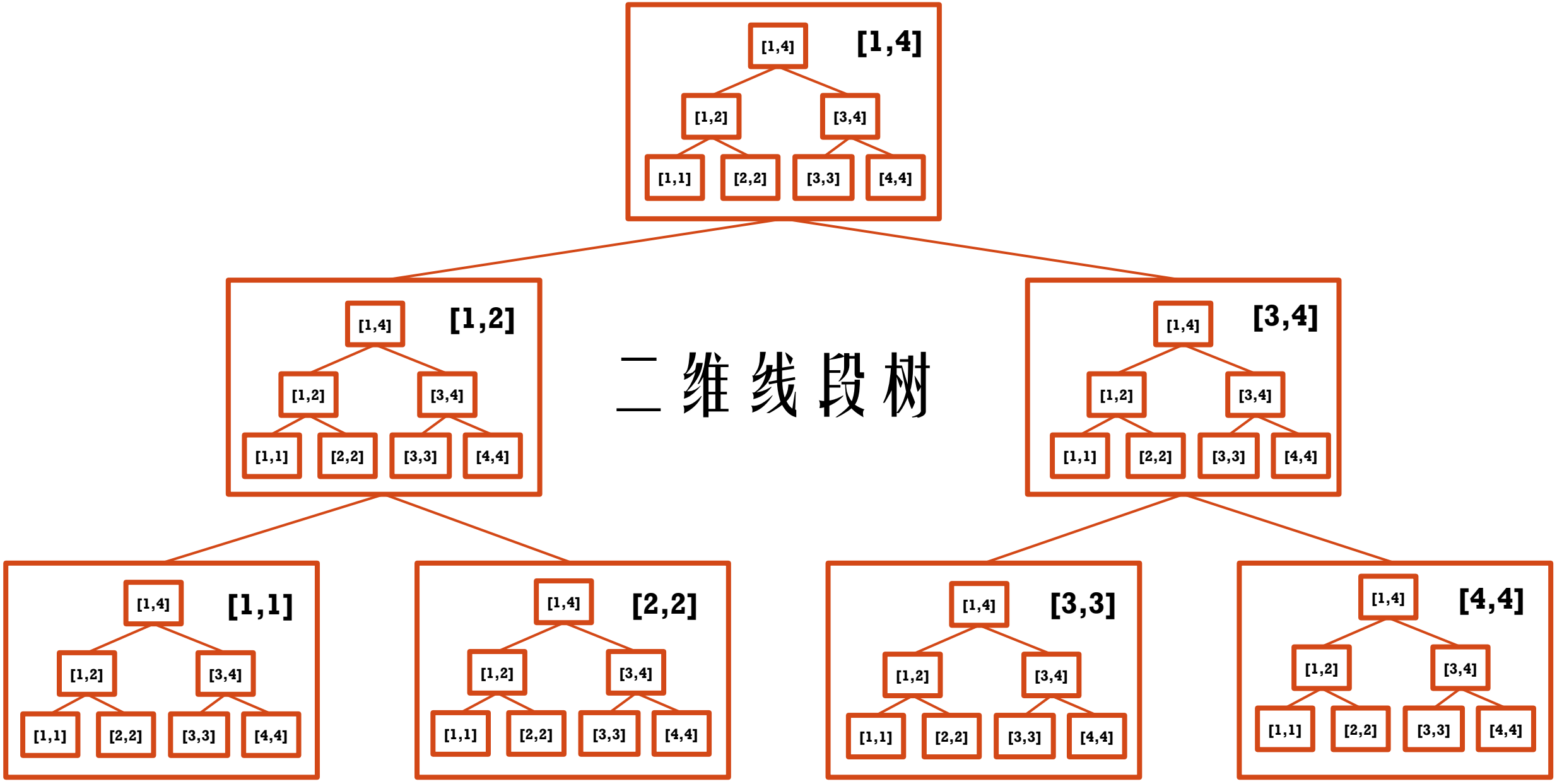
- 当我们的问题变成二维，要统计一块平面区域的时候，就需要用到二维线段树。

- **问题：**
  - 维护一个$n(1 \le n \le 500)$行$m(1 \le m \le 500)$列的数字矩阵，支持下面两种操作
    1. 输入x_1, x_2, y_1, y_2：查询所有满足$x_1 \le x \le x_2, y_1 \le y \le y_2$的格子$(x, y)$的和值。
    2. 输入x, y, v：修改方格(x, y)的值v。
  - 一共有$q(1 \le q \le 40000)$次操作。

二维线段树

- 如果对于每一行建一棵线段树，每次查询要遍历行，时间复杂度$O(N^2 \log N)$。
- 对于行也用线段树维护，每个结点上，除了区间和等信息，**再维护一棵线段树**，这棵线段树维护**行区间固定的情况下，列的区间和。**

二维线段树


[1,4]
[1,4]
[1,2]
[3,4]
[1,1]
[2,2]
[3,3]
[4,4]
[1,2]
[1,4]
[1,2]
[3,4]
[1,1]
[2,2]
[3,3]
[4,4]
[3,4]
[1,4]
[1,2]
[3,4]
[1,1]
[2,2]
[3,3]
[4,4]
[1,1]
[1,4]
[1,2]
[3,4]
[1,1]
[2,2]
[3,3]
[4,4]
[2,2]
[1,4]
[1,2]
[3,4]
[1,1]
[2,2]
[3,3]
[4,4]
[3,3]
[1,4]
[1,2]
[3,4]
[1,1]
[2,2]
[3,3]
[4,4]
[4,4]
[1,4]
[1,2]
[3,4]
[1,1]
[2,2]
[3,3]
[4,4]

二维线段树

——区间查询

```
int query1D(int idx, int id, int l, int r, int ly, int ry) { //在y轴上操作
    if (ly <= l && r <= ry) {
        return s[idx][id];
    }
    int sum = 0;
    int mid = (l + r) >> 1;
    if (ly <= mid) {
        sum += query1D(idx, id << 1, l, mid, ly, ry);
    }
    if (mid < ry) {
        sum += query1D(idx, id << 1 | 1, mid + 1, r, ly, ry);
    }
    return sum;
}

int query2D(int id, int l, int r, int lx, int ly, int rx, int ry) { //在x轴上操作
    if (lx <= l && r <= rx) {
        return query1D(id, 1, 1, m, ly, ry);
    }
    int sum = 0;
    int mid = (l + r) >> 1;
    if (lx <= mid) {
        sum += query2D(id << 1, l, mid, lx, ly, rx, ry);
    }
    if (mid < rx) {
        sum += query2D(id << 1 | 1, mid + 1, r, lx, ly, rx, ry);
    }
    return sum;
}
```

二维线段树

—单点更新

```
void update1D(int idx, int id, int l, int r, int y, int v, bool leaf) {
    if (l == r) {
        if (leaf) {
            s[idx][id] += v;
        } else {
            s[idx][id] = s[idx << 1][id] + s[idx << 1 | 1][id];
        }
        return;
    }
    int mid = (l + r) >> 1;
    if (y <= mid) {
        update1D(idx, id << 1, l, mid, y, v, leaf);
    } else {
        update1D(idx, id << 1 | 1, mid + 1, r, y, v, leaf);
    }
    s[idx][id] = s[idx][id << 1] + s[idx][id << 1 | 1];
}

void update2D(int id, int l, int r, int x, int y, int v) {
    if (l == r) {
        update1D(id, 1, 1, m, y, v, 1);
        return;
    }
    int mid = (l + r) >> 1;
    if (x <= mid) {
        update2D(id << 1, l, mid, x, y, v);
    } else {
        update2D(id << 1 | 1, mid + 1, r, x, y, v);
    }
    update1D(id, 1, 1, m, y, v, 0);
}
```

主席树

- **问题描述：**
- 给定一个长度为n的序列a，给出q个询问(l,r,k)，表示求a_l~a_r中第k小的值。
- $1 \leq n,q \leq 2 \times 10^5$

主席树

- **权值线段树：**每个叶子节点存储某个**元素的出现次数**，一条线段表示区间内**所有数出现次数的总和**。利用权值线段树可以方便地求出整体第k大：从根节点往下走如果k小于等于左子树大小，在左子树查找；否则在右子树中继续查找。
- **线段树的前缀和：**我们对一个长度为n的序列每个位置都建一个权值线段树。第i个线段树的区间$[x,y]$表示1~i这些数，在数组的$[x,y]$区间中出现的次数。如果要查询$[l,r]$的第k大，可以用第r棵减去第$l-1$棵得到区间$[l,r]$的权值线段树。
- **问题：**
  - 空间、时间复杂度至少$O(n^2 \log n)$

线段树的动态开点

- 先建一棵空树，更新的时候需要新开的点就开，不需要就连到原来的点。

```
1  inline int update(int cur, int l, int r, int x) {
2      int rt = ++cnt;
3      tree[rt] = tree[cur];
4      tree[rt].sum++;
5      int mid = (l + r) / 2;
6      if (l < r) {
7          if (x <= mid) tree[rt].l = update(tree[cur].l, l, mid, x);
8          else tree[rt].r = update(tree[cur].r, mid + 1, r, x);
9      }
10     return rt;
11 }
```

线段树的动态开点

- 先建一棵空树，更新的时候需要新开的点就开，不需要就连到原来的点。

```
1  inline int update(int cur, int l, int r, int x) {
2      int rt = ++cnt;
3      tree[rt] = tree[cur];
4      tree[rt].sum++;
5      int mid = (l + r) / 2;
6      if (l < r) {
7          if (x <= mid) tree[rt].l = update(tree[cur].l, l, mid, x);
8          else tree[rt].r = update(tree[cur].r, mid + 1, r, x);
9      }
10     return rt;
11 }
```

主席树
——建树

```
1  inline int build(int l, int r) {
2      int rt = ++cnt;
3      tree[rt].sum = 0;
4      int mid = (l + r) / 2;
5      if (l < r) {
6          tree[rt].l = build(l, mid);
7          tree[rt].r = build(mid + 1, r);
8      }
9      return rt;
10 }
```

主席树
——查询区间第K大

```
1 inline int query(int u, int v, int l, int r, int k) {
2     if (l >= r) return l;
3     int x = tree[tree[v].l].sum - tree[tree[u].l].sum;
4     int mid = (l + r) / 2;
5     if (x >= k) return query(tree[u].l, tree[v].l, l, mid, k);
6     else return query(tree[u].r, tree[v].r, mid + 1, r, k - x);
7 }
```

主席树

离散化&查询

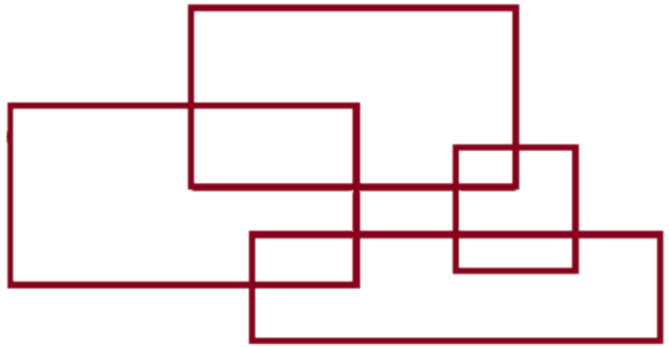
```
int main() {
    n = read(), q = read();
    for (int i = 1; i <= n; i++) {
        a[i] = read();
        b[i] = a[i];
    }

    sort(b + 1, b + n + 1);
    m = unique(b + 1, b + n + 1) - b - 1;
    tid[0] = build(1, m);
    for (int i = 1; i <= n; i++) {
        int t = lower_bound(b + 1, b + m + 1, a[i]) - b;
        tid[i] = update(tid[i - 1], 1, m, t);
    }

    while (q--) {
        int x = read(), y = read(), z = read();
        int t = query(tid[x - 1], tid[y], 1, m, z);
        printf("%d\n", b[t]);
    }

    return 0;
}
```

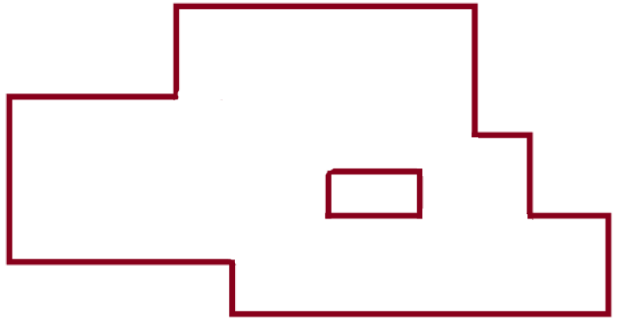
- 求n个矩形的面积并。
- 给出正整数n和四个非负整数x_1, y_1, x_2, y_2 表示一个矩形的左下角坐标为(x_1, y_1),右上角坐标为(x_2, y_2)。
- $1 \le n \le 10^5, 0 \le x_1 < x_2 \le 10^9, 0 \le y_1 < y_2 \le 10^9$

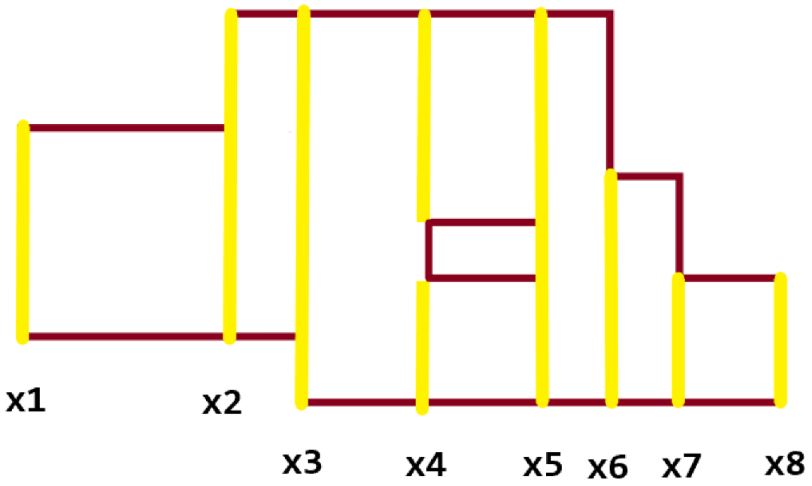
线段树例题

——扫描线

P5490 【模板】扫描线







线段树问题

——扫描线

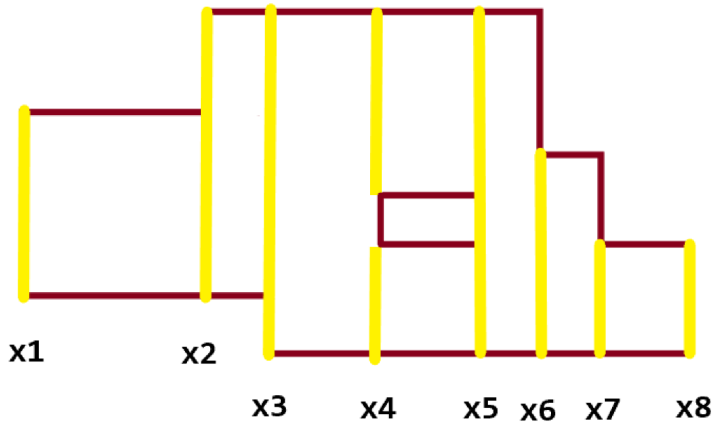
P5490 【模板】扫描线

线段树问题

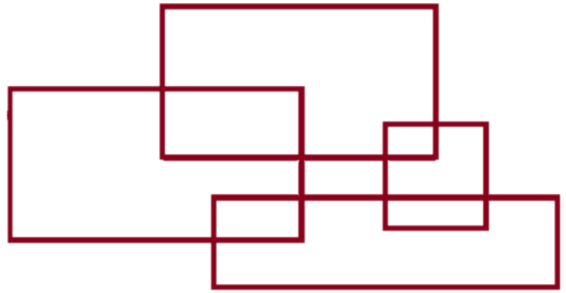
——扫描线

P5490 【模板】扫描线

- **扫描线的思路：**使用一条垂直于x轴的直线，从左到右扫描图形，显然将所有**左边界、右边界**按x值排序即可处理。
- 遇到矩形的左边界，就把这条边加入线段树；遇到右边界就弹出这条边。



- 如上图，黄色部分就是有线段覆盖的部分。要求面积只需要知道$X_1 \sim X_8$的长度即可。线段树就是维护这个长度的。

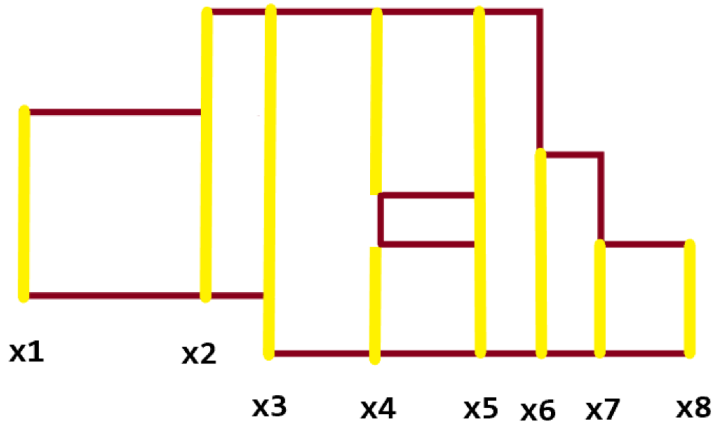


- **过程模拟：**
  - 设$X_1 \sim X_8$的长度分别为$L_1 \sim L_8$
  - $\boldsymbol{X_1}$：将X_1加入线段树，计算出L_1。
  - $\boldsymbol{X_2}$：将X_2加入线段树，计算面积$S \mathrel{+}= L_1 \times (X_2 - X_1)$，计算出$L_2$
  - $\boldsymbol{X_3}$：将X_3加入线段树，计算面积$S \mathrel{+}= L_2 \times (X_3 - X_2)$，计算出$L_3$
  - $\boldsymbol{X_4}$：将X_4加入线段树，首先计算$S \mathrel{+}= L_3 \times (X_4 - X_3)$，然后遇到了第一个矩形的右边界，这时要从线段树中删除一条线段。这时出现了两条线段，计算出它们的和为L_4。

线段树问题
——扫描线

P5490 【模板】扫描线

- **算法流程：**
  - 将所有竖边用结构体存起来，并按x坐标排序。
  - 扫描时，记录前一个操作结束后的L值，和前一个横坐标。
  - 更新S



- **实现过程**
- 边的范围1e9，数组开不下怎么办？
  - 使用**离散化**。用val_i记录编号为i的点对应的坐标
- 线段树怎么建？结点处存什么？怎么更新？
  - 开一个结构体其中sum表示区间被完全覆盖的次数，len表示区间内被覆盖线段的长度。
  - 更新时，如果$\text{sum} > 0$，len即为区间的右端点减去左端点；否则，还是合并子区间信息更新。
- 关于查询
  - 我们每次要查询的是整个y轴上的覆盖区间长度，所以直接访问len_1即可。
  - 由于我们每次只查询根节点的值，所以让子结点自生自灭，不用pushdown。

- **离散化：**当有些数据因为本身很大或者类型不支持，自身无法作为数组的下标来方便地处理，而影响最终结果的只有元素之间的相对大小关系时，我们可以将原来的数据按照从大到小编号来处理问题，即离散化。
- **离散化步骤：**
  - 将所有大数存进一个数组中；
  - 用unique对数组去重；
  - 用lower_bound查找大数的下标，下标即为离散化后的值。

线段树问题

——扫描线

P5490 【模板】扫描线

```
ll n, ans, cnt, sum[N << 2], cov[N << 2],  val[N << 2], tmp;

struct node {
    ll up, low, x;
    bool lr; // 记录是左边界还是右边界
} ed[N << 2];

void pushup(int id, int l, int r) {
    if (sum[id]) cov[id] = val[r] - val[l];
    else cov[id] = cov[ls] + cov[rs];
}

void update(int id, int l, int r, int x, int y, int v) {
    if (val[r] <= x || y <= val[l]) return; // 加等号，因为线段树区间左闭右开
    if (x <= val[l] && val[r] <= y) {
        sum[id] += v;
        pushup(id, l, r);
        return;
    }
    update(ls, l, mid, x, y, v);
    update(rs, mid, r, x, y, v); // 注意是mid不是mid+1
    pushup(id, l, r);
}
```

线段树问题

——扫描线

P5490【模板】扫描线

```cpp
int main() {
    n = read();
    for (int i = 1; i <= n; i++) {
        ll mx1 = read(), my1 = read(), mx2 = read(), my2 = read();
        ed[++cnt].x = mx1; ed[cnt].up = my2;
        ed[cnt].low = my1; ed[cnt].lr = false;
        val[cnt] = my1;
        ed[++cnt].x = mx2; ed[cnt].up = my2;
        ed[cnt].low = my1; ed[cnt].lr = true;
        val[cnt] = my2;
    }
    sort(ed + 1, ed + cnt + 1, cmp);
    sort(val + 1, val + cnt + 1);
    tmp = unique(val + 1, val + cnt + 1) - val - 1;
    ll l = 0;
    for (int i = 1; i <= cnt; i++) {
        if(ed[i].lr == false) update(1,1, tmp, ed[i].low, ed[i].up, 1);
        else update(1, 1, tmp, ed[i].low, ed[i].up, -1);
        ans += (ed[i].x - ed[i - 1].x) * l;
        l = cov[1];
    }

    cout << ans << endl;
    return 0;
}
```

线段树问题

——扫描线

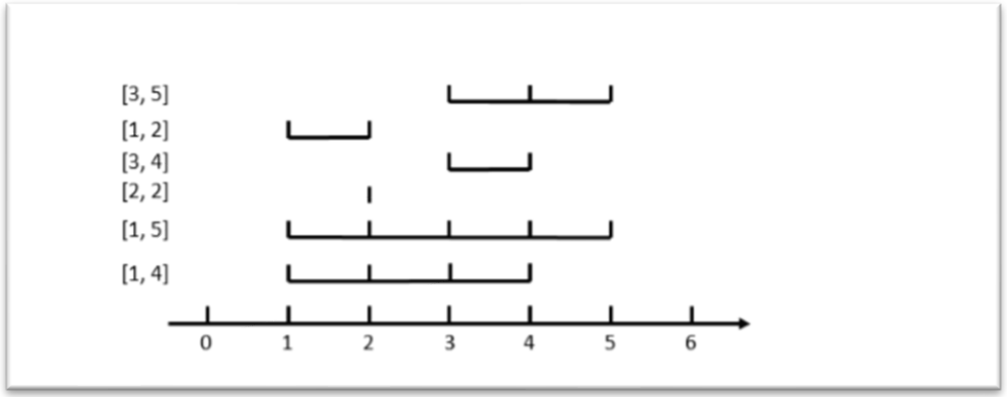
P5490 【模板】扫描线

线段树问题

——区间覆盖

NOI 2016 区间

- 在数轴上有n个闭区间从1至n编号，第i个闭区间为$[l_i, r_i]$。现在要从中选出m个区间，使得这m个区间共同包含至少一个位置。换句话说，就是使得存在一个x，对于每个被选中的区间$[l_i, r_i]$，都有$l_i \le x \le r_i$。
- 对于一个合法的选取方案，它的花费为被选中的**最长区间**长度减去被选中的**最短区间**长度。
- 区间 $[l_i, r_i]$的长度定义为 $(r_i - l_i)$，即等于它的右端点的值减去左端点的值。求所有合法方案中最小的花费。如果不存在合法的方案，输出 -1。
- $1 \le m \le n,\ 1 \le n \le 5 \times 10^5,\ 1 \le m \le 2 \times 10^5,\ 0 \le l_i \le r_i \le 10^9$



- 根据花费的计算公式：被选中的**最长区间**长度减去被选中的**最短区间**长度。
- 将所有区间按长度排序，让最长区间尽可能小，最短区间尽可能大。
- 两个指针L, R表示区间$[L, R]$间的区间加入。逐一将区间加入，当有一个点的覆盖次数达到 m ，统计答案并将前面的区间顺序删掉，直到所有点 $< m$ 。重复上面的过程。
- 如何快速维护区间内点被覆盖的次数？线段树维护区间最值即可。
- 注意到$0 \leq l_i \leq r_i \leq 10^9$，我们需要将数据**离散化。**

线段树问题
——区间覆盖

NOI 2016 区间

```
ll L = 0, R = 0, ans = 1e18;
while (true) {
    while (maxn[1] < m && R <= n) {
        R++;
        update(1, 1, tmp, q[R].l, q[R].r, 1);
    }
    if (maxn[1] < m) break;
    while (maxn[1] >= m && L <= n) {
        L++;
        update(1, 1, tmp, q[L].l, q[L].r, -1);
    }
    ans = min(ans, (bas[R].r - bas[R].l) - (bas[L].r - bas[L].l));
}
```

线段树问题

——区间覆盖

NOI 2016 区间

- 给定一个长度不超过10^5的字符串（小写英文字母），和不超过50000个操作。
- 每个操作L, R, K表示给区间$[L, R]$的字符串排序，$K = 1$为升序，$K = 0$为降序。
- 最后输出最终的字符串。

线段树问题
——计数排序

CodeForces 558E
A Simple Task

线段树问题
——计数排序

CodeForces 558E
A Simple Task

- 因为字符串中只有小写字母，考虑维护26棵线段树对应每一个字母在区间上出现的次数。
- 对区间$[l,r]$进行升序操作时，按 $a\sim z$ 枚举每个字母，先统计这个字母在$[l,r]$中的数量，计算出其排序后覆盖的区间，删除原来的数据并进行区间修改。

abbccacabbcaaac

abaabbcccbcaaac

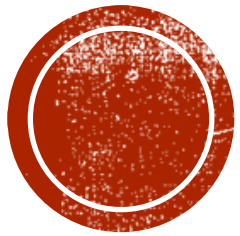
- 同理，进行降序操作时 $z\sim a$ 枚举每个字母即可。

```cpp
int main() {

    for (int i = 1; i <= q; i++) {
        int x, y, op;
        cin >> x >> y >> op;
        int tmp = x;
        if (op == 1) {
            for (int j = 1; j <= 26; j++) {
                int cnt = query(1, 1, n, x, y, j);
                if (!cnt) continue;
                update(1, 1, n, x, y, j, 0);
                update(1, 1, n, tmp, tmp + cnt - 1, j, 1);
                tmp += cnt;
            }
        } else {
            for (int j = 26; j >= 1; j--) {
                int cnt = query(1, 1, n, x, y, j);
                if (!cnt) continue;
                update(1, 1, n, x, y, j, 0);
                update(1, 1, n, tmp, tmp + cnt - 1, j, 1);
                tmp += cnt;
            }
        }
    }

    return 0;
}
```

线段树问题

——计数排序

CodeForces 558E
A Simple Task



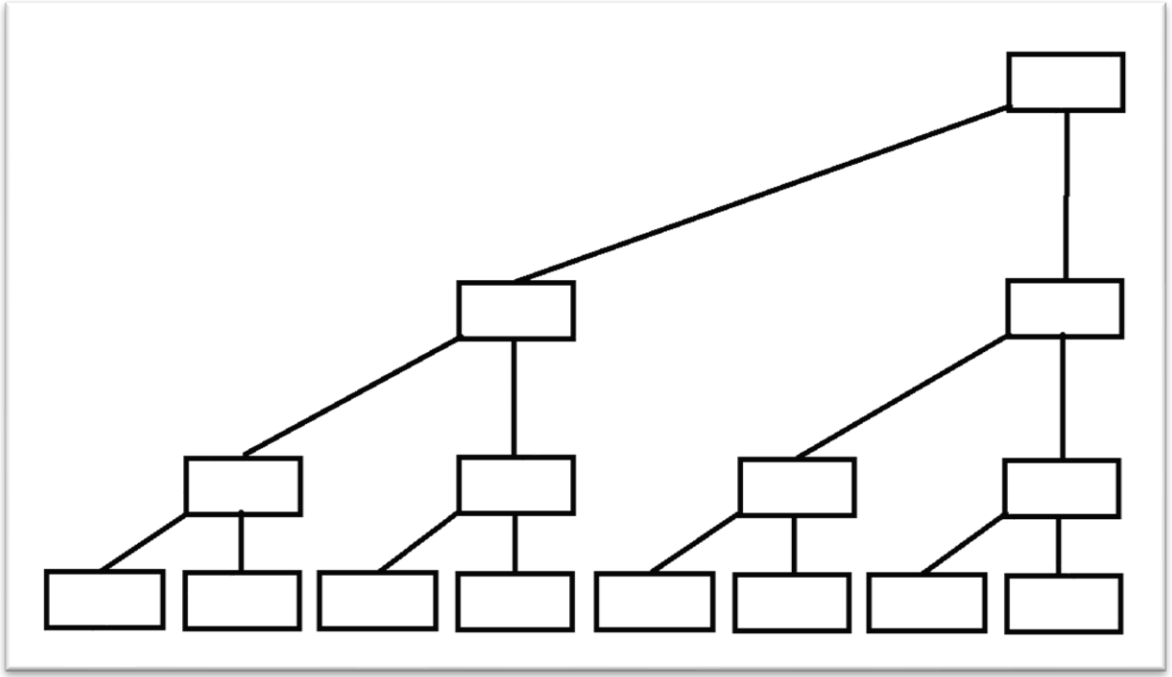
树状数组

树状数组

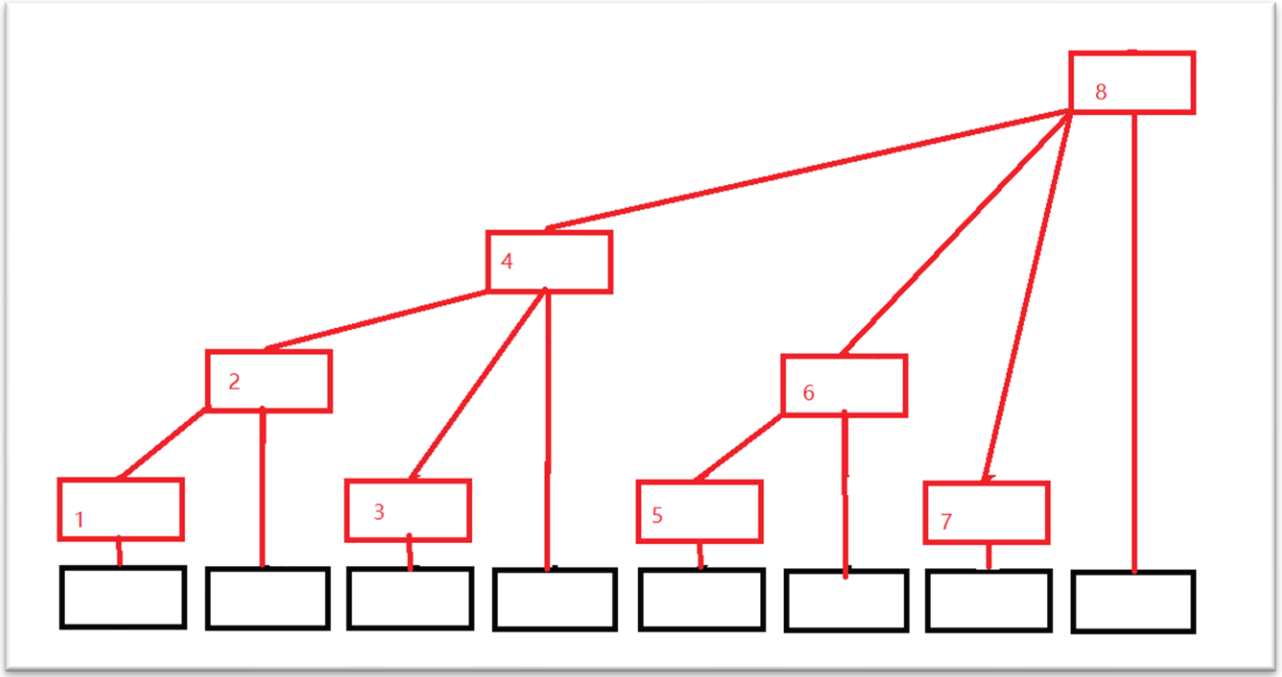
——概述

- 树状数组，用数组模拟树形结构。可以解决大部分基于区间上的更新以及**求和**问题。树状数组可以解决的都可以用线段树来解决。
- 时间复杂度也是$O(\log N)$但常数比线段树小。

树状数组的结构



线段树：一棵二叉树

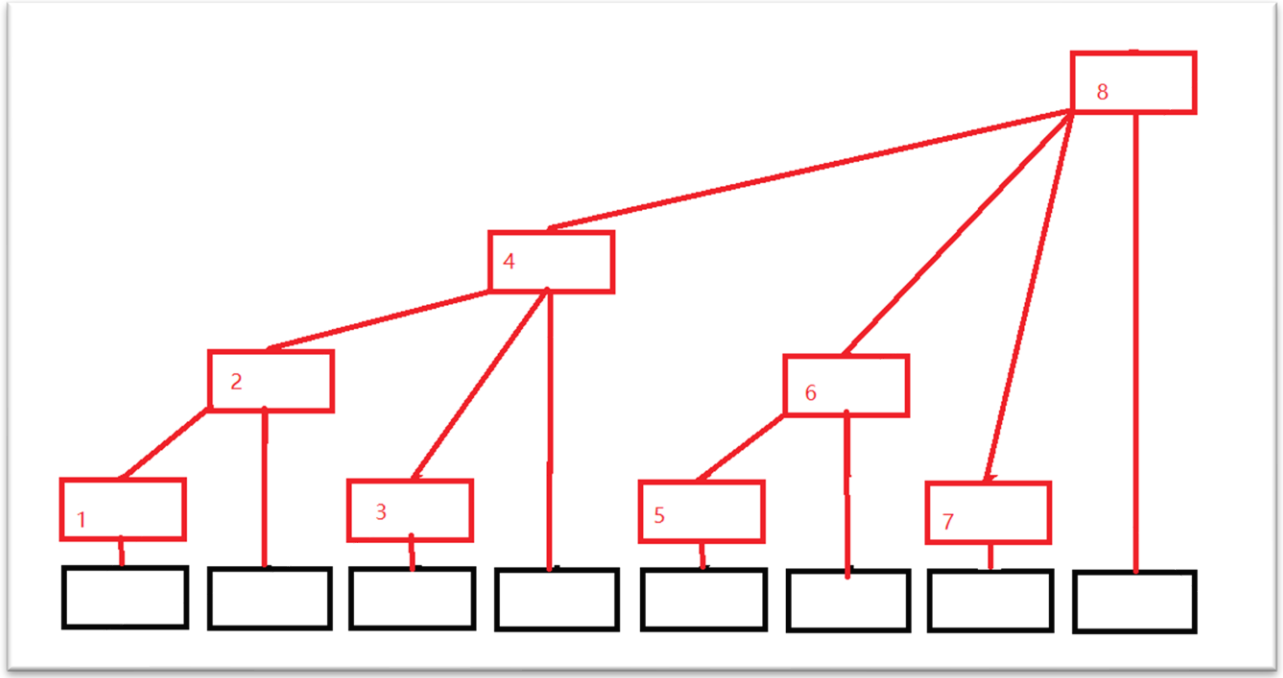


树状数组：省去一些节点以使用数组

树状数组的结构

- **黑色代表原数组A，红色代表树状数组C。**

- $C_1 = A_1$
- $C_2 = A_1 + A_2$
- $C_3 = A_3$
- $C_4 = A_1 + A_2 + A_3 + A_4$
- $C_5 = A_5$
- $C_6 = A_5 + A_6$
- $C_7 = A_7$
- $C_8 = A_1 + A_2 + A_3 + A_4 + A_5 + A_6 + A_7 + A_8$



树状数组的结构

- $C_1 = A_1$
- $C_2 = A_1 + A_2$
- $C_3 = A_3$
- $C_4 = A_1 + A_2 + A_3 + A_4$
- $C_5 = A_5$
- $C_6 = A_5 + A_6$
- $C_7 = A_7$
- $C_8 = A_1 + A_2 + A_3 + A_4 + A_5 + A_6 + A_7 + A_8$

- **可以发现，这棵树是有规律的：**

$$C_i = A_{i-2^k+1} + A_{i-2^k+2} + \cdots + A_i$$

- **k为i的二进制中从最低位到高位连续零的长度**
- 例如，$i = 8(1000)_2$时，$k = 3$
- **我们将2^k称为lowbit**

树状数组的结构

——LOWBIT

- 根据前人的智慧，$\text{lowbit}_i = i\&(-i)$
- 这里利用了**负数的存储特性**，负数是以**补码**存储的，**负数的补码是将其对应正数按位取反后加一。**
- 对于整数运算$x\&(-x)$有：
  1. 当$x = 0$时，即$0\&0 = 0$；
  2. 当x为奇数时，最后一个比特位为1，取反加1没有进位，故x和$-x$**除最后一位**外，前面的位正好相反，按位与结果为0。所以最后结果为1。
  3. 当x为偶数，且为2的m次方时，x的二进制表示中只有一位是1（从右往左的第$m+1$位），其右边有m位0，故x取反加1后，从右到左第有m个0，第$m+1$位及其左边全是1。这样，$x\&(-x)$得到的就是x。
  4. 当x为偶数，却不为2的m次方的形式时，可以写作$x = y \times 2^k$。其中，y的最低位为1。实际上就是把x用一个奇数左移k位来表示。这时，x的二进制表示最右边有k个0，从右往左第$k+1$位为1。当对x取反时，最右边的k位0变成1，第$k+1$位变为0；再加1，最右边的k位就又变成了0，第$k+1$位因为进位的关系变成了1。左边的位因为没有进位，正好和x原来对应的位上的值相反。二者按位与，得到：第$k+1$位上为1，左边右边都为0。结果为2^k。
- 总结：$x\&(-x)$，当x为0时结果为0；x为奇数时，结果为1；x为偶数时，结果为x中2的最大次方的因子。

树状数组的区间求和

- 如果要计算数组A的区间和，比如说要算A_{51}~A_{91}的区间和，可以采用类似倍增的思想：
- 从91开始往前跳，发现C_n管A_{91}这个点，那么继续找A_{90}，发现C_{n-1}管的是A_{90}&A_{89}；那么你就会直接跳到A_{88}，以此类推。C_i管的数的个数是lowbit_i
- 不难得出:

$$\boldsymbol{sum_i = C_i + C_{i-lowbit_i} + C_{i-lowbit_i-lowbit_{i-lowbit_i}} + \cdots}$$

- 其中的k_i是第i项下标的k值。

树状数组的区间求和

——实现

- 假设现在要查询区间$[l,r]$的值，可以先求$[1,r]$的和减去$[1,l-1]$的和。
- 由于$x=x-\text{lowbit}_x$等价于将x的二进制的最后一个1减去。而x的二进制中最多有$\log x$个1，所以查询效率是$\log x$的。

```
1  inline int lowbit(int x) {
2      return x & (-x);
3  }
4
5  int getsum(int x) {
6      int res = 0;
7      for (int i = x; i > 0; i -= lowbit(i)) {
8          res += C[i];
9      }
10     return res;
11 }
```

树状数组的单点更新

- 让A_x增加v，那么只有对于包含位置x的区间会受影响。
- 一直令$x = x + \text{lowbit}_x$，直到$x > n$，这一过程中，所有的x都是受影响的点
- 时间复杂度也是$O(\log n)$

```
1 void update(int x, int v) {
2     for (int i = x; i <= n; i += lowbit(i)) {
3         C[i] += v;
4     }
5 }
```

树状数组的区间更新与区间查询
——差分序列

- 树状数组可以通过差分序列的转换，用来求解一些特殊的区间更新问题。
- 对数列A进行差分，得到数列d。对于每次区间更新$[l,r]$，d_l增加v；$\mathrm{d}_{\mathrm{r}+1}$减少$v$。
- 用树状数组维护差分序列，每次更新只需两次单点更新即可。但此时只能满足单点查询。
- **区间查询：**
  - 根据$\mathrm{A}_i = \sum_{j=1}^{i} d_j$ 有

$$\sum_{i=1}^{x} a_i = \sum_{i=1}^{x}\sum_{j=1}^{i} d_j = \sum_{j=1}^{x}(x-j+1)d_j = (x+1)\times\sum_{j=1}^{x} d_j - \sum_{j=1}^{x} jd_j$$

  - 所以，我们只需要用树状数组维护d_j的前缀和jd_j的前缀和。

树状数组的区间更新与区间查询

——实现

```
int c[2][maxn];

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

void update(int x, int v, int t) {
    for (int i = x; i <= n; i += lowbit(i)) {
        C[t][i] += v;
    }
}

int getans(int x) {
    return (x + 1) * getsum(x, 0) - getsum(x, 1);
}

int getsum(int x, int t) {
    int res = 0;
    for (int i = x; i >= 0; i -= lowbit(i)) {
        res += c[t][i];
    }
    retunr res;
}
```

树状数组的区间更新

——实现

```
int main() {
    cin >> n;
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
        update(i, a[i] - a[i - 1], 0);
        update(i, i * (a[i] - a[i - 1]), 1);
    }
    int q;
    while (q--) {
        int op, l, r, v;
        cin >> op;
        if (op == 0) { // 查询[l,r]
            cin >> l >> r;
            cout << getans(r) - getans(l - 1) << endl;
        } else { // 更新[l,r]
            cin >> l >> r >> w;
            update(l, v, 0);
            update(r + 1, -v, 0);
            update(l, v * l, 1);
            update(r + 1, -v * (r + 1), 1);
        }
    }
    return 0;
}
```

- 给定一个长度为n的正整数序列a，求序列中满足$a_i > a_j, i < j$，的有序对个数。
- $n \leq 5 \times 10^5, a_i \leq 10^9$

树状数组问题
——求逆序对

P1908 逆序对

树状数组问题

——求逆序对

P1908 逆序对

- 思考如何统计第i个数会与第1~i个数构成多少逆序对
  - 建立树状数组。按从左到右的顺序将数据的值对应位置+1。
  - 对于第i项，前$i-1$项都被加入了树状数组，逆序对数量为$i-\text{getsum}(a_i)$。
  - 由于$a_i \le 10^9$，需要离散化。

```
1  int main() {
2      n = read();
3      for (int i = 1; i <= n; i++) {
4          a[i].v = read();
5          a[i].id = i;
6      }
7      sort (a + 1, a + n + 1, cmp);
8      for (int i = 1; i <= n; i++) rk[a[i].id] = i;
9      for (int i = 1; i <= n; i++) {
10         add(rk[i]);
11         ans += i - getsum(rk[i]);
12     }
13     cout << ans;
14     return 0;
15 }
```

树状数组问题

P2154 [SDOI2009]
虔诚的墓主人

- 给定$n \times m$的矩形，每个格点是一块空地或一棵树。定义一块空地的价值为：以这块地为中心，正上、正下、正左、正右都有恰好k棵树的情况的总和。
- 求所有空地价值的总和。
- $1 \leq N, M \leq 10^9, 1 \leq k \leq 10$

THANKS !

参考资料

- 线段树 - OI Wiki
- 可持久化线段树 - OI Wiki
- 树状数组 - OI Wiki
- 信息学 Level 5 课程 - 计蒜客
- 信息学 Level 6 课程 - 计蒜客
- 线段树详解 By 岩之痕 - CSDN博客
- 树状数组详解 By Xenny - 博客园