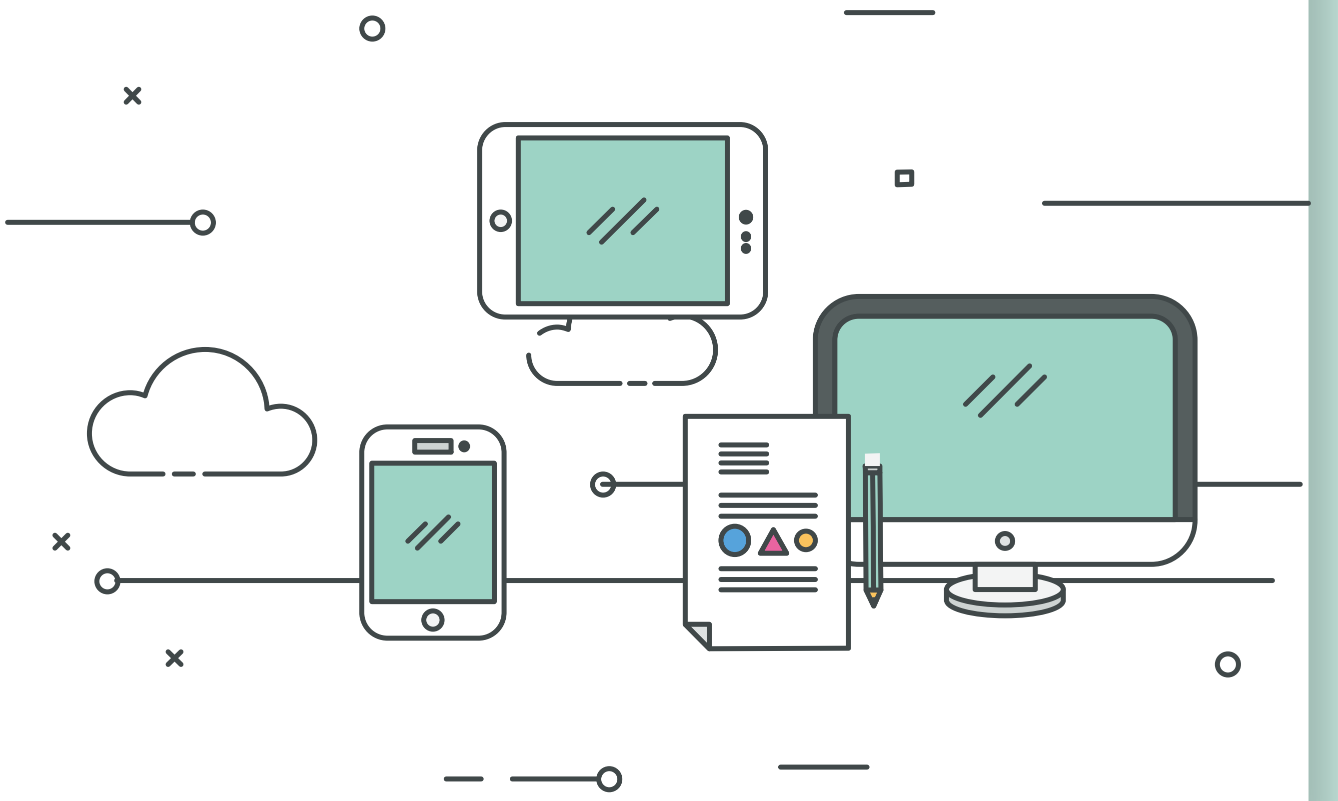
提高组

线段树

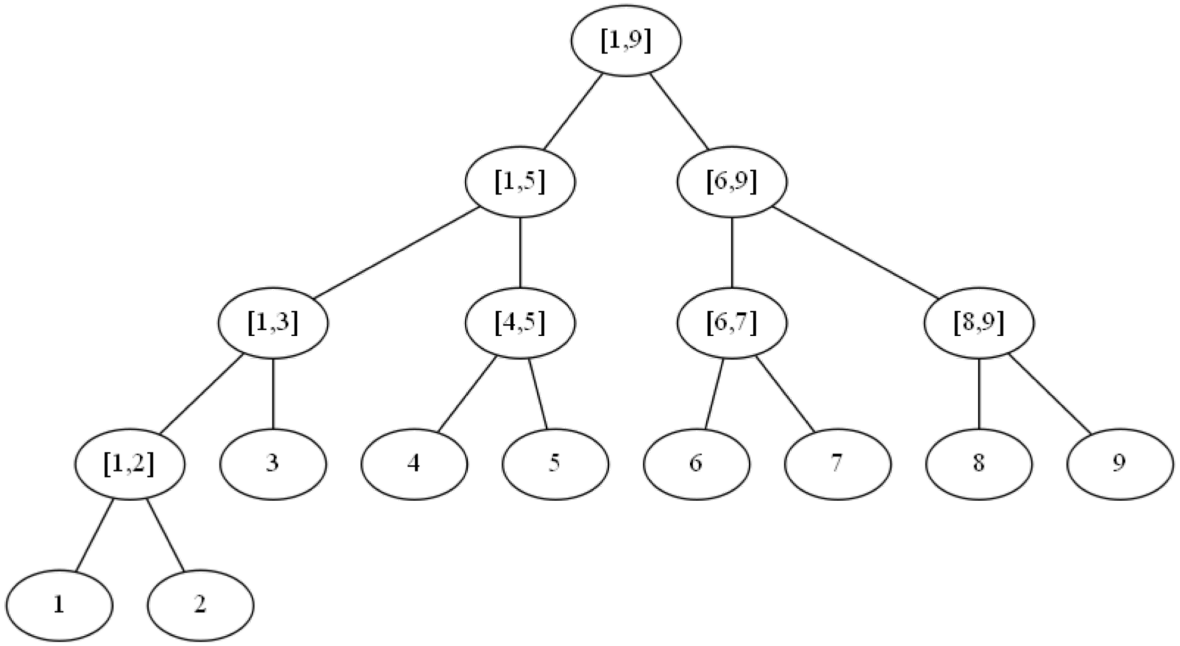


线段树

区间修改与查询

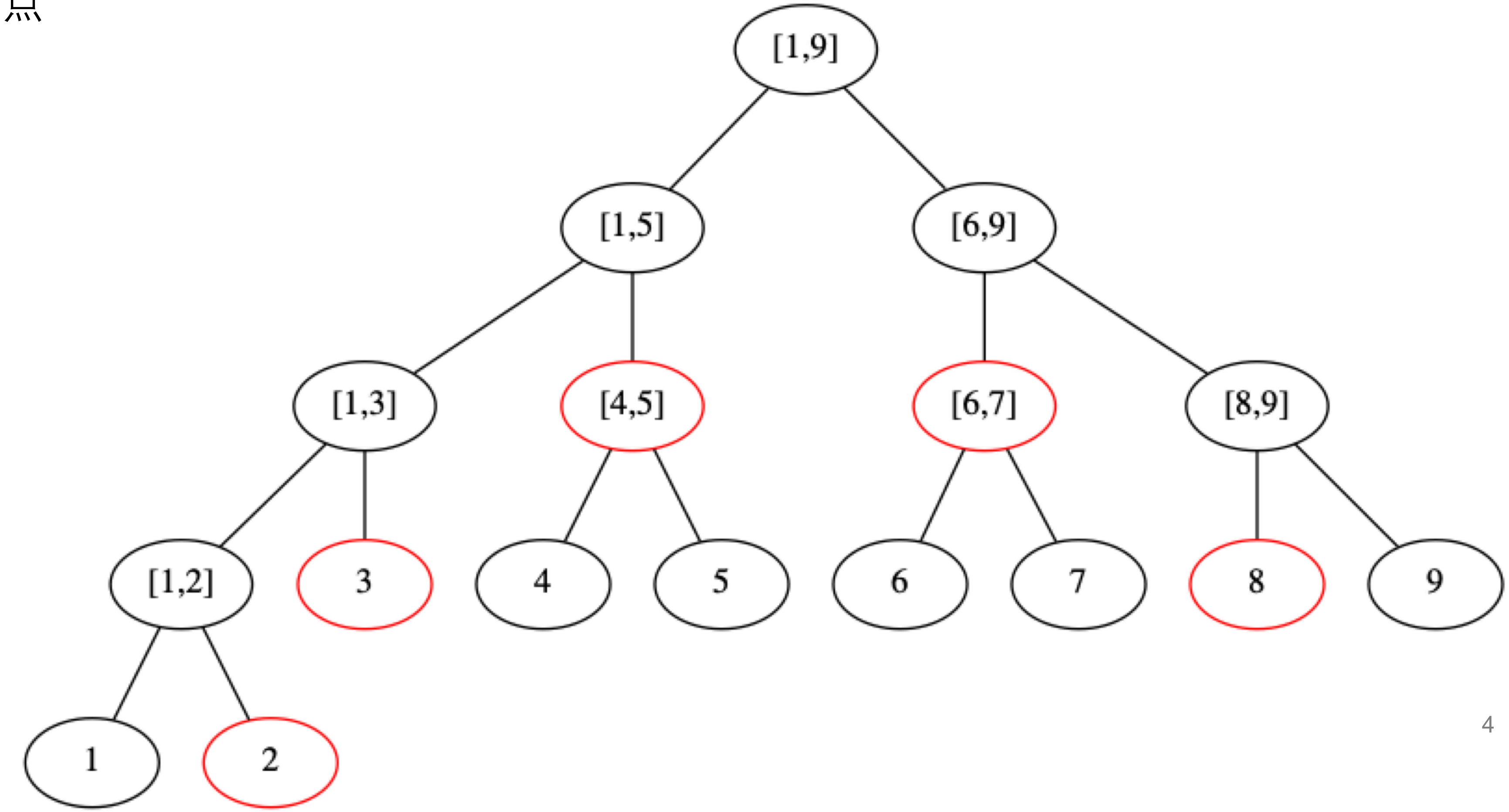
线段树(Segment Tree)的定义

- 一棵二叉树
- 树上的每个结点对应于一个区间[a,b]，子结点分别对应区间[a,(a+b)/2], [(a+b)/2+1,b]
- 同一层的结点所代表的区间，相互不会重叠。同一层结点所代表的区间，加起来是个连续的区间
- 叶子结点的区间是单位长度，不能再分
- 深度数量级log(n)
- 叶子结点个数是N
- 叶子结点的数目和根结点表示区间的长度相同.
- 线段树结点要么0度，要么2度



区间分解

- 如果有某个结点代表的区间， 完全属于待分解区间，则该结点为 “终止” 结点，不再继续往下分解
- 所有 “终止” 结点所代表的区间都不重叠，且加在一起就恰好等于整个待分解区间
- 如图就是区间[2,8]的分解
- 每层最多两个非终止结点



线段树的特征

- 线段树深度是O(log(n))的
- 对线段树上进行更新和查询的复杂度是O(log(n))的
- 线段树适用于和区间统计有关的问题。比如某些数据可以按区间进行划分，按区间动态进行修改，而且还需要按区间多次进行查询，那么使用线段树可以达到较快的速度
- 对于每一个子结点而言，都表示整个序列中的一段子区间；对于每个叶子结点而言，都表示序列中的单个元素信息；子结点不断向自己的父亲结点传递信息，而父结点存储的信息则是它的每一个子结点信息的整合。

存储与表示

用数组存储树，树根在下标为1的位置，i结点左儿子在2*i，右儿子在2*i+1位置

为了方便，定义两个函数用来求左儿子和右儿子在数组中的下标

```
inline int lc(int p) { return p << 1; }

inline int rc(int p) { return p << 1 | 1; }
```

P2880 [USACO07JAN]平衡的阵容Balanced Lineup

- 每天,农夫 John 的N(1 <= N <= 50,000)头牛总是按同一序列排队. 有一天, John 决定让一些牛们玩一场飞盘比赛. 他准备找一群在对列中位置连续的牛来进行比赛. 但是为了避免水平悬殊,牛的身高不应该相差太大. John 准备了Q (1 <= Q <= 180,000) 个可能的牛的选择和所有牛的身高 (1 <= 身高 <= 1,000,000). 他想知道每一组里面最高和最低的牛的身高差别.
- 我们用a数组来存储每个牛的身高，用mi数组来表示树的每个节点对应的区间内身高的最小值，用ma数组来表示最大值
- 递归建树，之后在树上做查询即可

建树

```cpp
const int MAXN=50005;
const int MAXH=1000005;
int mi[MAXN * 4];
int ma[MAXN * 4];
int a[MAXN];
int n, q;

inline int lc(int p) { return p << 1; }

inline int rc(int p) { return p << 1 | 1; }

void pushUp(int p) {
    mi[p] = min(mi[lc(p)], mi[rc(p)]);
    ma[p] = max(ma[lc(p)], ma[rc(p)]);
}
```

```cpp
void buildTree(int p, int l, int r) {
    if (l == r) {
        mi[p] = a[l];
        ma[p] = a[l];
        return;
    }
    int mid = (l + r) >> 1;
    buildTree(lc(p), l, mid);
    buildTree(rc(p), mid + 1, r);
    pushUp(p);
}
```

查询最小值

```
int queryMin(int p, int l, int r, int ql, int qr) {
    if (ql <= l && r <= qr) {
        return mi[p];
    }
    int ans = MAXH;
    int mid = (l + r) >> 1;
    if (ql <= mid) {
        ans = min(ans, queryMin(lc(p), l, mid, ql, qr));
    }
    if (mid < qr) {
        ans = min(ans, queryMin(rc(p), mid + 1, r, ql, qr));
    }
    return ans;
}
```

主体

```
int main() {
    scanf("%d%d", &n, &q);
    for (int i = 1; i <= n; ++i) {
        scanf("%d", &a[i]);
    }
    buildTree(1,1,n);
    while (q--) {
        int ql, qr;
        scanf("%d%d", &ql, &qr);
        int rmax = queryMax(1, 1, n, ql, qr);
        int rmin = queryMin(1, 1, n, ql, qr);
        printf("%d\n", rmax - rmin);
    }
    return 0;
}
```

区间修改

- 单点修改是区间修改的特例，我们只讨论区间修改
- 区间修改例子：把某个区间内所有数字都加n
- 如果每次所有修改都下沉传递到叶子节点，复杂度太高。我们可以在父结点保留一个lazy tag，懒标记，表示对子区间需要做某个操作，但是目前先不拆开做了。
- 如果后续查询完整覆盖当前区间，不需要传递标记。
- 如果后续查询（修改)当前区间，且查询（修改）一部分，需要把标记下沉

P3372 【模板】线段树 1

已知一个数列，你需要进行下面两种操作：

1.将某区间每一个数加上x

2.求出某区间每一个数的和

```cpp
#include <cstdio>

using namespace std;
typedef long long ll;
const int MAXN = 100005;
ll n, m;
ll a[MAXN];
ll sum[MAXN * 4];
ll tag[MAXN * 4];

inline ll lc(ll p) { return p << 1; }

inline ll rc(ll p) { return p << 1 | 1; }

void pushUp(ll p) {
    sum[p] = sum[lc(p)] + sum[rc(p)];
}
```

```cpp
void buildTree(ll p, ll l, ll r) {
    if (l == r) {
        sum[p] = a[l];
        return;
    }
    ll mid = (l + r) >> 1;
    buildTree(lc(p), l, mid);
    buildTree(rc(p), mid + 1, r);
    pushUp(p);
}
```

```
void moveTag(ll p, ll l, ll r, ll t) {
    sum[p] += t * (r - l + 1);
    tag[p] += t;
}

void pushDown(ll p, ll l, ll r) {
    ll mid = (l + r) >> 1;
    moveTag(lc(p), l, mid, tag[p]);
    moveTag(rc(p), mid + 1, r, tag[p]);
    tag[p] = 0;
}
```

```
void update(ll p, ll l, ll r, ll ql, ll qr, ll d) {
    if (ql <= l && r <= qr) {
        sum[p] += d * (r - l + 1);
        tag[p] += d;
        return;
    }
    pushDown(p, l, r);
    ll mid = (l + r) >> 1;
    if (ql <= mid) {
        update(lc(p), l, mid, ql, qr, d);
    }
    if (mid < qr) {
        update(rc(p), mid + 1, r, ql, qr, d);
    }
    pushUp(p);
}
```

```cpp
ll query(ll p, ll l, ll r, ll ql, ll qr) {
    if (ql <= l && r <= qr) {
        return sum[p];
    }
    pushDown(p,l,r);
    ll mid = (l + r) >> 1;
    ll result = 0;
    if (ql <= mid) {
        result += query(lc(p), l, mid, ql, qr);
    }
    if (mid < qr) {
        result += query(rc(p), mid + 1, r, ql, qr);
    }
    return result;
}
```

作业

- P1531 I Hate It
- P1890 gcd区间
- P3373 【模板】线段树 2
- P1198 [JSOI2008]最大数

离散化

- 有时，区间的端点不是整数，或者区间太大导致建树内存开销过大MLE ,那么就需要进行“离散化”后再建树
- 建立一个映射关系，把原始的区间端点，映射到比较小的整数范围内

P3740 [HAOI2014]贴海报

Bytetown城市要进行市长竞选，所有的选民可以畅所欲言地对竞选市长的候选人发表言论。为了统一管理，城市委员会为选民准备了一个张贴海报的electoral墙。

张贴规则如下：

- electoral墙是一个长度为N个单位的长方形，每个单位记为一个格子；
- 所有张贴的海报的高度必须与electoral墙的高度一致的；
- 每张海报以 “A B”表示，即从第A个格子到第B个格子张贴海报；
- 后贴的海报可以覆盖前面已贴的海报或部分海报。

现在请你判断，张贴完所有海报后，在electoral墙上还可以看见多少张海报。

数据范围

0<= N <= 10000000 1<=M<=1000 1<= Ai <= Bi <=10000000

可以见到，N非常大，但是M很小，每张海报可能很长，中间空很大没意义

如何离散化？

- 把每个海报的端点排序去重，映射到1开始的整数上
- 比如[100,200]和[30,150]，映射为[2,4]和[1,3]

- 但是区间[5,7],[1,5],[7,8]映射到区间[2,3],[1,2],[3,4]，按这样覆盖就会把第一张海报覆盖掉，但实际上第一张海报没有完全被覆盖
- 区间长度如果大于2，则在两个端点之间再加一个点

- 样例数据[1,4],[2,6],[8,10],[3,4],[7,10]

原始端点	1	2	3	4	6	7	8	10
离散坐标	1	2	3	4	6	7	8	10

如何统计？

- 按照从后往前的顺序，贴上海报，先把最后一张海报贴上去，此时它一定是能露出来的。
- 之后，倒着贴海报，如果它覆盖的面积没被上面的海报全部覆盖，则它也会露出来

普通离散化

```
for (int i = 0; i < N; ++i) {
    scanf("%d", &a[i]);
    dis[i] = a[i];
}
sort(dis, dis + N);
int dn = unique(dis, dis + N) - dis;
for (int i = 0; i < N; ++i) {
    a[i] = lower_bound(dis, dis + dn, a[i]) - dis + 1;
}
```

```cpp
//返回true表示当前区间已经被上面的海报完全覆盖了
bool query(int p, int l, int r, int ql, int qr) {
    if (ql <= l && r <= qr) {
        return cover[p];
    }
    pushDown(p);
    int mid = (l + r) >> 1;
    bool res = true;
    if (ql <= mid) {
        if (!query(lc(p), l, mid, ql, qr)) {
            res = false;
        }
    }
    if (mid < qr) {
        if (!query(rc(p), mid + 1, r, ql, qr)) {
            res = false;
        }
    }
    return res;
}
```

```cpp
void update(int p, int l, int r, int ql, int qr) {
    if (ql <= l && r <= qr) {
        cover[p] = true;
        tag[p] = true;
        return;
    }
    pushDown(p);
    int mid = (l + r) >> 1;
    if (ql <= mid) {
        update(lc(p), l, mid, ql, qr);
    }
    if (mid < qr) {
        update(rc(p), mid + 1, r, ql, qr);
    }
    pushUp(p);
}
```

```cpp
int main() {
    cin >> n >> m;
    for (int i = 1; i <= m; ++i) {
        cin >> posterL[i] >> posterR[i];
        a[2 * i - 1] = posterL[i];
        a[2 * i] = posterR[i];
    }
    sort(a + 1, a + 1 + 2 * m);
    uniqueN = unique(a + 1, a + 1 + 2 * m) - (a + 1);
    treeN = 1;
    for (int i = 1; i < uniqueN; ++i) {
        bucket[a[i]] = treeN;
        if (a[i + 1] - a[i] == 1) {
            treeN++;
        } else {
            treeN += 2;
        }
    }
    bucket[a[uniqueN]] = treeN;
```

```cpp
    int ans = 0;
    for (int i = m; i >= 1; --i) {
        if (!query(1, 1, treeN, bucket[posterL[i]], bucket[posterR[i]])) {
            ans++;
            update(1, 1, treeN, bucket[posterL[i]], bucket[posterR[i]]);
        }
    }
    cout << ans << endl;
    return 0;
}
```

合并区间线段树

区间合并的小技巧

P2894 [USACO08FEB]酒店Hotel

第一行输入n，m ，n代表有n个房间，编号为1---n，开始都为空房，m表示以下有m行操作，以下 每行先输入一个数 i ，表示一种操作：

若i为1，表示查询房间，再输入一个数x，表示在1--n 房间中找到长度为x的连续空房，输出连续x个房间中左端的房间号，尽量让这个房间号最小，若找不到长度为x的连续空房，输出0。

若i为2，表示退房，再输入两个数 x，y 代表 房间号 x---x+y-1 退房，即让房间为空。

P2894 [USACO08FEB]酒店Hotel

为了整洁，我们用结构体来装线段树。父结点的最大连续空房的个数num，无法由子节点直接得到，因为有可能两个子节点左右相接能得到更大的num，所以我们还需要维护左右子节点接近边界的最大连续空房的个数

```
const int MAXN = 50005;
struct Node {
    int len;//区间长度
    int num;//区间最大连续0的长度
    int lNum;//从左侧开始的最大长度
    int rNum;//以区间右侧结尾的最大长度
    int tag;//0表示无，1表示住满了，2表示全退了
} tree[MAXN * 4];
```

P2894 [USACO08FEB]酒店Hotel

合并的时候，如果左边全是空房，那么右边的lNum是可以直接接到左边的。

同理，右边都是空房，左边的rNum也可以直接连上

```
void pushUp(int p) {
    if (tree[lc(p)].num == tree[lc(p)].len) {
        tree[p].lNum = tree[lc(p)].num + tree[rc(p)].lNum;
    } else {
        tree[p].lNum = tree[lc(p)].lNum;
    }
    if (tree[rc(p)].num == tree[rc(p)].len) {
        tree[p].rNum = tree[rc(p)].num + tree[lc(p)].rNum;
    } else {
        tree[p].rNum = tree[rc(p)].rNum;
    }
    tree[p].num = tree[lc(p)].rNum + tree[rc(p)].lNum;
    tree[p].num = max(tree[p].num, tree[lc(p)].num);
    tree[p].num = max(tree[p].num, tree[rc(p)].num);
}
```

```
void buildTree(int p, int l, int r) {
    tree[p].len = r - l + 1;
    if (l == r) {
        tree[p].num = tree[p].lNum = tree[p].rNum = 1;
        return;
    }
    int mid = (l + r) >> 1;
    buildTree(lc(p), l, mid);
    buildTree(rc(p), mid + 1, r);
    pushUp(p);
}

void moveTag(int p, int l, int r, int tag) {
    if (tag == 1) {
        tree[p].num = tree[p].lNum = tree[p].rNum = 0;
        tree[p].tag = 1;
    } else {
        tree[p].num = tree[p].lNum = tree[p].rNum = r - l + 1;
        tree[p].tag = 2;
    }
}
```

```
void pushDown(int p, int l, int r) {
    if (tree[p].tag == 0) return;
    int mid = (l + r) >> 1;
    moveTag(lc(p), l, mid, tree[p].tag);
    moveTag(rc(p), mid + 1, r, tree[p].tag);
    tree[p].tag = 0;
}
```

```
 int query(int p, int l, int r, int k) {
    if (l == r) return l;
    int mid = (l + r) >> 1;
    pushDown(p, l, r);
    if (tree[lc(p)].num >= k) return query(lc(p), l, mid, k);
    if (tree[lc(p)].rNum + tree[rc(p)].lNum >= k) return mid + 1 - tree[lc(p)].rNum;
    if (tree[rc(p)].num >= k) return query(rc(p), mid + 1, r, k);
    return 0;
 }
void checkIn(int p, int l, int r, int ql, int qr) {
   if (ql <= l && r <= qr) {
       tree[p].num = tree[p].lNum = tree[p].rNum = 0;
       tree[p].tag = 1;
       return;
   }
   pushDown(p, l, r);
   int mid = (l + r) >> 1;
   if (ql <= mid) {
       checkIn(lc(p), l, mid, ql, qr);
   }
   if (mid < qr) {
       checkIn(rc(p), mid + 1, r, ql, qr);
   }
   pushUp(p);
}
```

```c
void checkOut(int p, int l, int r, int ql, int qr) {
    if (ql <= l && r <= qr) {
        tree[p].num = tree[p].lNum = tree[p].rNum = r - l + 1;
        tree[p].tag = 2;
        return;
    }
    pushDown(p, l, r);
    int mid = (l + r) >> 1;
    if (ql <= mid) {
        checkOut(lc(p), l, mid, ql, qr);
    }
    if (mid < qr) {
        checkOut(rc(p), mid + 1, r, ql, qr);
    }
    pushUp(p);
}
```

```c
int main() {
    scanf("%d%d", &n, &m);
    buildTree(1, 1, n);
    for (int i = 0; i < m; ++i) {
        int op, x, y;
        scanf("%d", &op);
        if (op == 1) {
            scanf("%d", &x);
            int pos = query(1, 1, n, x);
            printf("%d\n", pos);
            if (pos != 0) {
                checkIn(1, 1, n, pos, pos + x - 1);
            }
        } else {
            scanf("%d%d", &x, &y);
            checkOut(1, 1, n, x, x + y - 1);
        }
    }
    return 0;
}
```

SP1716 GSS3 - Can you answer these queries III

- n 个数，q 次操作
- 操作0 x y把A_x 修改为y
- 操作1 l r询问区间[l, r] 的最大子段和

SP1716 GSS3 - Can you answer these queries III

- 线段树维护每个区间上的最大子段和，左侧开始的最大子段和，右侧开始的最大子段和，这个区间的和。
- 合并的时候merge两个结点

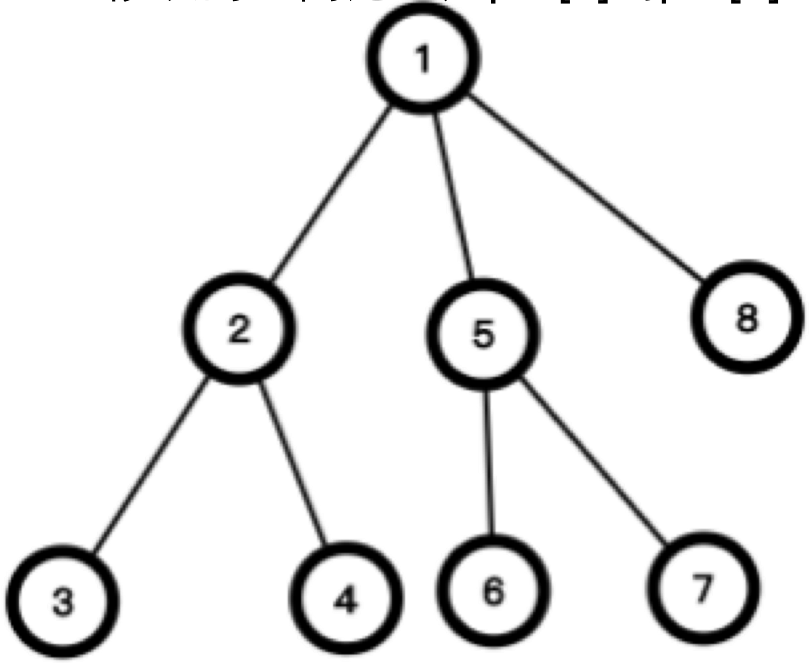
```
Node mergeNode(const Node &x, const Node &y) {
    Node r;
    r.sum = x.sum + y.sum;
    r.ls = max(x.ls, x.sum + y.ls);
    r.rs = max(y.rs, x.rs + y.sum);
    r.msum = max(x.msum, y.msum);
    r.msum = max(r.msum, x.rs + y.ls);
    return r;
}
```

DFS序线段树

把树压成序列

DFS序线段树

- 要处理树上的问题，包括子树的修改，和子树上的查询。如果把子树里面每个点都处理一遍，复杂度O(n)
- 在树上DFS一遍，可以得到DFS序，按照这个DFS序，可以把一棵树，变成一个序列。
- 这个序列上，一个子树里面所有的点，是连续的。所以子树上的区间修改和查询，正好对应序列上的区间修改和查询。所以每次子树修改的复杂度变成了log(n)，可以接受。如果当前要改以u为根的子树，那么要修改的区间范围是pre[u]到pre[u]+sz[u]-1，其中sz数组表示子树的大小。



U142060 【模板】dfs序线段树

一棵有n个结点的树，规定树根是1，树上每个点有个初始权w。在树上进行m次操作，每次操作指定一棵子树，把子树里面的每个点的权增加d，或者查询子树上每个点权之和。

请你对每一次查询，输出答案。

```cpp
int n, m, k, nn;
int sz[MAXN], pre[MAXN], id[MAXN], w[MAXN], dep[MAXN], dt,
head[MAXN];
ll sum[MAXN * 4], tag[MAXN * 4];
struct Edge {
    int v, next;
} pool[MAXN * 2];
void addEdge(int u, int v) {
    pool[++nn].v = v;
    pool[nn].next = head[u];
    head[u] = nn;
}
int init(int u, int f) {
    pre[u] = ++dt;
    dep[u] = dep[f] + 1;
    id[dt] = u;
    sz[u] = 1;
    for (int i = head[u]; i; i = pool[i].next) {
        int v = pool[i].v;
        if (v == f) continue;
        sz[u] += init(v, u);
    }
    return sz[u];
}
```

```
void buildTree(int p, int l, int r) {
    if (l == r) {
        sum[p] = w[id[l]];
        return;
    }
    int mid = (l + r) >> 1;
    buildTree(lc(p), l, mid);
    buildTree(rc(p), mid + 1, r);
    pushUp(p);
}
```

```c
scanf("%d%d", &n, &m);
for (int i = 0; i < n - 1; ++i) {
    int u, v;
    scanf("%d%d", &u, &v);
    addEdge(u, v);
    addEdge(v, u);
}
for (int i = 1; i <= n; ++i) {
    scanf("%d", &w[i]);
}
init(1, 0);
buildTree(1, 1, n);
while (m--) {
    int op, x, d;
    scanf("%d%d", &op, &x);
    if (op == 1) {
        scanf("%d", &d);
        update(1, 1, n, pre[x], pre[x] + sz[x] - 1, d);
    } else {
        printf("%lld\n", querySum(1, 1, n, pre[x], pre[x] + sz[x] - 1));
    }
}
```

问题变形

- 给出n个点的一棵树，树上每个点有点权。有m次操作，每次操作修改一条路径x到y，把经过的每个点的点权加d，或者询问树上某个点x的点权。

问题变形

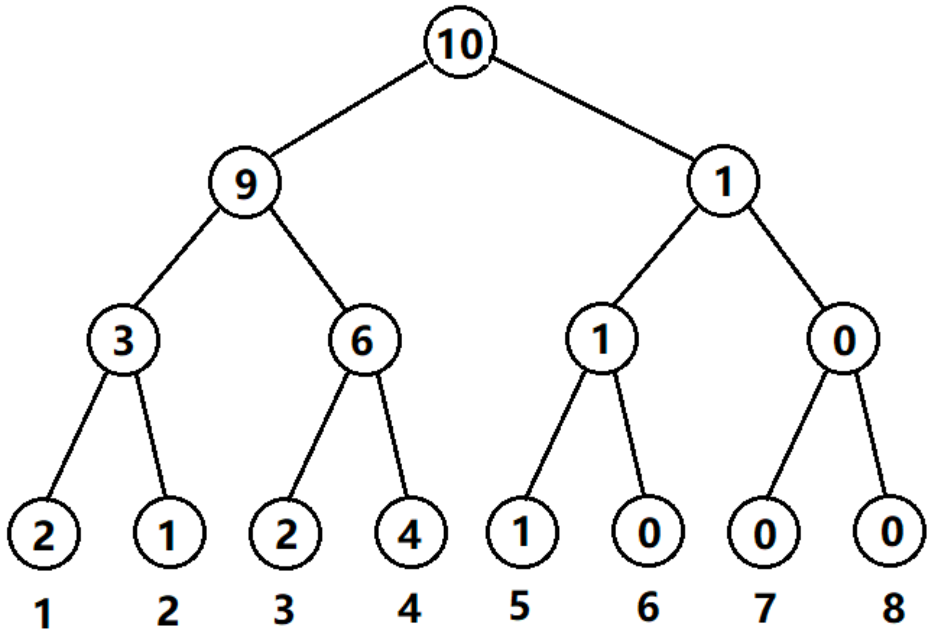
- 借助树上差分的思想，当x到y路径加的时候，在diff[x],diff[y]上加，在diff[lca]和 diff[lca的父亲]上减。
- 当询问某个点的时候，查询它子树里面diff的和，因为子树里面的路径加操作，对当前点有影响。

权值线段树

线段树版本的桶

权值线段树

- 记录权值的线段树。记录权值指的是，每个点上存的是区间内的数字出现的总次数。比如一个长度为10的数组[1,1,2,3,3,4,4,4,4,5]。
- 其中1出现了两次，那么[1,1]这个节点的值为2，2出现了1次，那么[2,2]这个节点的值为1，那么显然[1,2]这个节点的值为3，即1出现的次数和2出现的次数加和。那么如果我想要知道这个数组上的第k小，我就可以在这棵权值线段树上用logn的时间来实现。
- 如果原始输入的值域范围比较大，可能需要先离散化。



P1908 逆序对

- 先离散化，把所有点映射到1到cnt范围内
- 每当加入点x的时候，统计x+1到cnt范围内数字的个数，然后把x插入树中

```cpp
typedef long long ll;
const int MAXN = 500005;

struct Node {
    int val, pos;

    bool operator<(const Node &n) const {
        return val < n.val;
    }
} x[MAXN];

int dis[MAXN],n;

inline int lc(int p) { return p << 1; }

inline int rc(int p) { return p << 1 | 1; }

int sum[MAXN * 4];

void pushUp(int p) {
    sum[p] = sum[lc(p)] + sum[rc(p)];
}
```

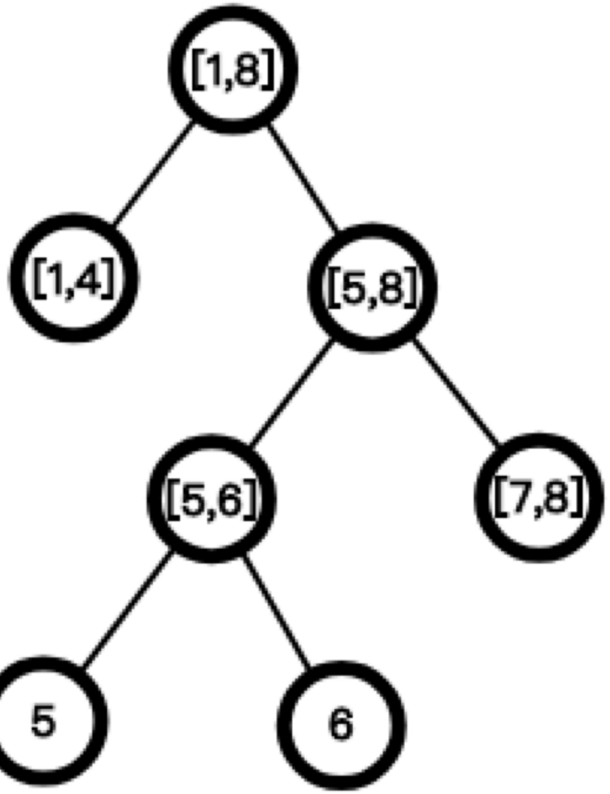
```cpp
void update(int p, int l, int r, int q) {
    if (l == r) {
        sum[p]++;
        return;
    }
    int mid = (l + r) >> 1;
    if (q <= mid) {
        update(lc(p), l, mid, q);
    } else {
        update(rc(p), mid + 1, r, q);
    }
    pushUp(p);
}
```

```cpp
ll query(int p, int l, int r, int ql, int qr) {
    ll ans = 0;
    if (ql <= l && r <= qr) {
        return sum[p];
    }
    int mid = (l + r) >> 1;
    if (ql <= mid) {
        ans += query(lc(p), l, mid, ql, qr);
    }
    if (mid < qr) {
        ans += query(rc(p), mid + 1, r, ql, qr);
    }
    return ans;
}
```

```cpp
int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; ++i) {
        scanf("%d", &x[i].val);
        x[i].pos = i;
    }
    sort(x + 1, x + n + 1);
    x[0].val = 1000000005;
    int cnt = 1;
    for (int i = 1; i <= n; ++i) {
        if (x[i - 1].val == x[i].val) dis[x[i].pos] = cnt;
        else dis[x[i].pos] = ++cnt;
    }
    long long ans = 0;
    for (int i = 1; i <= n; ++i) {
        ans += query(1, 1, cnt, dis[i] + 1, cnt);
        update(1, 1, cnt, dis[i]);
    }
    printf("%lld\n", ans);
    return 0;
}
```

动态开点

- 如果线段树需要维护的区间很大很大，但是实际用到的节点很少很少。
- 在线操作不能离散化
- 不要开这么多的节点，用到的时候再向内存要，总空间消耗nlogn
- 建立了一棵残疾的线段树，缺少很多枝叶



U74894有便便的厕所

- 众所周知，梁老师家的狗特别喜欢拉便便。梁老师为了方便它方便，在家里修建了10^9个马桶，依次排开，成一条直线，为了方便（这里的方便意思是叙述方便，不是“方便”），依次编号1到10^9。
- 豆豆每次会选择一个马桶方便，但是很不幸，它不会冲厕所。梁老师为了冲厕所方便，修建了一个巨型水桶，可以一次冲掉一个区间内每个厕所。（当然，区间上如果有没用过的厕所，也会一起冲，这样也许有些浪费水）
- 豆豆还有一个特殊的爱好，它想观察某个区间上所有有便便的厕所中，编号第k大的是哪个厕所，以方便跟它的朋友小野和花卷吹嘘自己。比如它观察区间2到10，想找到其中第2大的。而区间2到10中，有2，4，5，6四个厕所里面目前还有便便，那么答案就是5.或者说，在区间里面，把每个便便对应的编号拿出来，从大到小排序，要求排在第k位置上的编号。可惜厕所太多了，它算不过来，所以请你写个程序帮忙。
- 注意，因为豆豆不讲卫生，如果一个厕所里面有便便，它可能还继续在这个厕所方便，此时这个厕所里面算有2个便便，计算第k大的时候，这个重复的也参与排序。
- 给你Q个操作：
- 操作1的格式是：{1 x}: 表示豆豆在x号位置拉便便
- 操作2的格式是：{2 l r}: 表示梁老师冲掉l<=x<=r的所有便便。如果一个厕所里面有多个便便，也都一起冲掉。
- 操作3的格式是：{3 l r k}: 表示豆豆想在区间l到r范围内，包括l和r，寻找第k大的厕所编号（若不存在输出-1）

有便便的厕所

数据范围很大，决定动态开点。

有范围删除操作，记录一个del的tag，表示是否当前结点内所有元素都被删除了。

```cpp
const int INF = 1e9 + 5;
const int MAXN = 1e5 + 5;
struct Node {
    int cnt, del, lc, rc;
} tree[MAXN * 50];
int q, nn = 1;

void pushUp(int p) {
    tree[p].cnt = tree[tree[p].lc].cnt + tree[tree[p].rc].cnt;
}

void moveTag(int p) {
    tree[p].del = 1;
    tree[p].cnt = 0;
}

void pushDown(int p) {
    if (tree[p].del == 1) {
        tree[p].del = 0;
        moveTag(tree[p].lc);
        moveTag(tree[p].rc);
    }
}
```

```cpp
void insert(int p, int l, int r, int q) {
    if (l == r) {
        tree[p].del = 0;
        tree[p].cnt++;
        return;
    }
    pushDown(p);
    int mid = (l + r) >> 1;
    if (q <= mid) {
        if (tree[p].lc == 0) tree[p].lc = ++nn;
        insert(tree[p].lc, l, mid, q);
    } else {
        if (tree[p].rc == 0) tree[p].rc = ++nn;
        insert(tree[p].rc, mid + 1, r, q);
    }
    pushUp(p);
}
```

```cpp
void remove(int p, int l, int r, int ql, int qr) {
    if (p == 0 || tree[p].del == 1 || tree[p].cnt == 0) return;
    if (ql <= l && r <= qr) {
        tree[p].del = 1;
        tree[p].cnt = 0;
        return;
    }
    int mid = (l + r) >> 1;
    pushDown(p);
    if (ql <= mid) {
        remove(tree[p].lc, l, mid, ql, qr);
    }
    if (mid < qr) {
        remove(tree[p].rc, mid + 1, r, ql, qr);
    }
    pushUp(p);
}
```

```cpp
int querySum(int p, int l, int r, int ql, int qr) {
    if (p == 0 || tree[p].del == 1 || tree[p].cnt == 0) {
        return 0;
    }
    if (ql <= l && r <= qr) {
        return tree[p].cnt;
    }
    pushDown(p);
    int ans = 0;
    int mid = (l + r) >> 1;
    if (ql <= mid) {
        ans += querySum(tree[p].lc, l, mid, ql, qr);
    }
    if (mid < qr) {
        ans += querySum(tree[p].rc, mid + 1, r, ql, qr);
    }
    return ans;
}
```

```cpp
int queryK(int p, int l, int r, int ql, int qr, int k) {
    if (p == 0 || tree[p].cnt == 0 || tree[p].del == 1) return -1;
    if (l == r) {
        if (tree[p].cnt >= k) return l;
        else return -1;
    }
    pushDown(p);
    int mid = (l + r) >> 1;
    int cnt = 0;
    int lc = tree[p].lc, rc = tree[p].rc;
    if (mid < qr) {
        cnt = querySum(rc, mid + 1, r, ql, qr);
        if (cnt >= k)
            return queryK(rc, mid + 1, r, ql, qr, k);
    }
    if (ql <= mid) {
        return queryK(lc, l, mid, ql, qr, k - cnt);
    }
    return -1;
}
```

```cpp
int main() {
    scanf("%d", &q);
    for (int i = 0; i < q; ++i) {
        int op, x, l, r, k;
        scanf("%d", &op);
        if (op == 1) {
            scanf("%d", &x);
            insert(1, 1, INF, x);
        } else if (op == 2) {
            scanf("%d%d", &l, &r);
            remove(1, 1, INF, l, r);
        } else {
            scanf("%d%d%d", &l, &r, &k);
            printf("%d\n", queryK(1, 1, INF, l, r, k));
        }
    }
    return 0;
}
```

U74895有便便的厕所2

- 众所周知，梁老师家的狗特别喜欢拉便便。梁老师为了方便它方便，在家里修建了10^6个马桶，依次排开，成一条直线，为了方便（这里的方便意思是叙述方便，不是“方便”），依次编号1到10^6。
- 豆豆喜欢每次在这些厕所里面选择一个来方便一下，产生一个便便。
- 但是豆豆很快发现，它如果每次只在一个厕所里面方便，很快就被梁老师冲掉了，为了能尽可能保留便便，它每次会选择一个区间l到r，在这个区间内（包括l和r）每个厕所里面都拉一个便便。
- 豆豆还有一个特殊的爱好，它每在第n号厕所里面拉一个便便，就在它的狗脑里面记一个数字n。如果一个厕所里面多次方便，就记多个数字。它的朋友小野每次会给定一个区间[l,r]，要求豆豆回答这个区间内的数字的和还有平方和。可惜数字太多了，它算不过来，所以请你写个程序帮忙。
- 给你Q个操作：
- 操作1的格式是：{1 x}: 表示豆豆在x号位置拉便便
- 操作2的格式是：{2 l r}: 豆豆在l到r区间都拉一个便便，包括l和r
- 操作3的格式是：{3 l r}: 表示小野询问豆豆的区间是l到r。包括l和r

有便便的厕所2

每个点tag的含义是，这个范围内每个数都需要再加tag[p]个

```
ll rangeSum(ll n) {
    return (n + 1) * n / 2;
}

ll rangeSquareSum(ll n) {
    return n * (n + 1) * (2 * n + 1) / 6;
}

void moveTag(int p, int l, int r, int t) {
    sum[p] += t * (rangeSum(r) - rangeSum(l - 1));
    square[p] += t * (rangeSquareSum(r) - rangeSquareSum(l - 1));
    tag[p] += t;
}
```

邮局选址问题

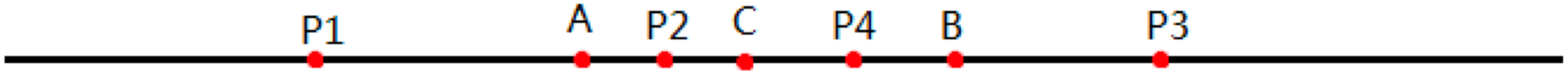
在数轴上找一个点，使得该点到其他所有点的距离之和最小

结论：选中位数这个点，如果有偶数个数字，选中间两个点中间任何位置都可以。

证明：当只有两个点A和B的时候，p点的位置有三种选择

显然在A和B中间某位置最好

当有三个点的时候：在AB中间的时候，点到A和B的距离之和相等，所以我们需要优化到C点的距离，所以刚好选择C点的时候，距离最小



邮局选址问题

当有四个点的时候，在AB中间，到AB的距离和相等。在CD中间，到CD的距离和相等，所以一定要选在CD中间

那么对于多个点时：

首先找到最外面的两个点，点P要在它们之间

然后在找次外面的点，点P也要在它们之间

……

一直找到只剩1或2个点

如果只剩一个点，那么最优方案就是P取这个点

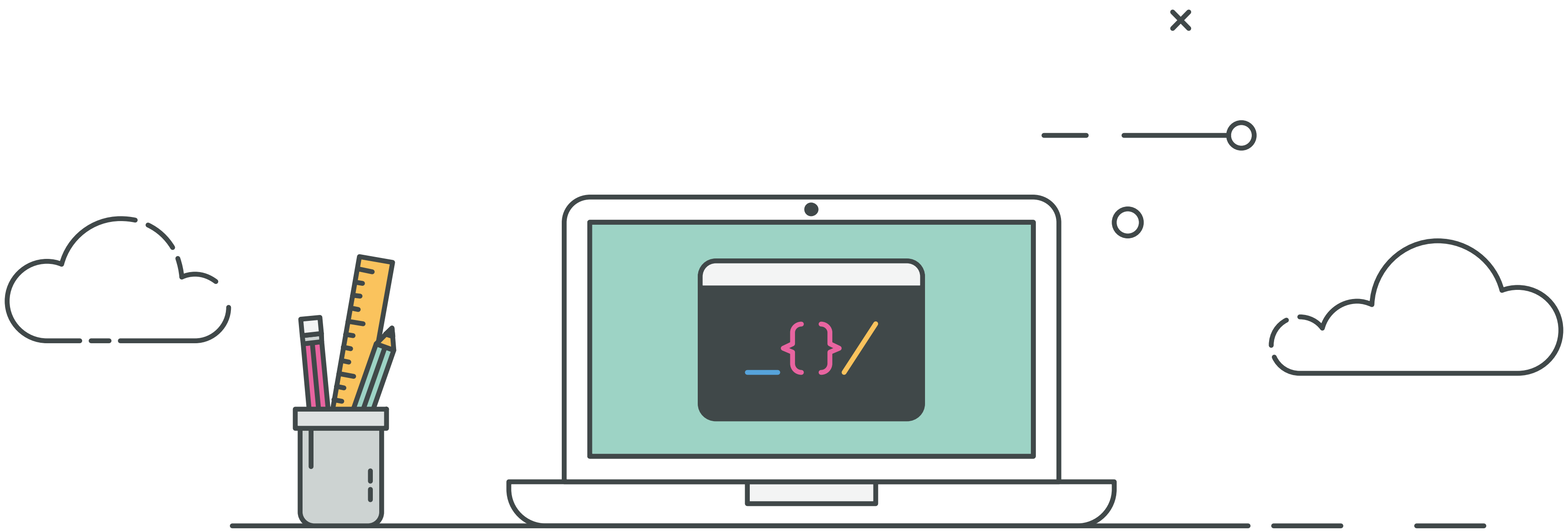
否则P可以取两个点之间的任意位置

这样就可以保证方案最优

即：找到大小为中位数的点（如有两个取之间任意一点都行）

P3871 [TJOI2010]中位数

- 先离散化，再用权值线段树取中位数



Thanks
It's the thoughts that count