Standard Code Library

Xavier_Cai

2020 年 10 月 31 日

目录

1 数据结构 **1**

1.1 01 字典树 1

1.1.1 路径为点权异或值求最小生成树（CF888G） 1

1.1.2 可持久化 01 字典树 1

1.2 吉司机线段树 2

1.2.1 区间取 min+ 区间查询 $O(m\log n)$ 2

1.2.2 支持区间加（BZOJ4695 最假女选手）$O(m\log^2 n)$ 4

1.2.3 维护区间最值操作与区间历史最值（洛谷线段树 3）$O(m\log^2 n)$ 6

1.3 二维树状数组 9

1.3.1 单点修改 + 区间查询 9

1.3.2 区间修改 + 单点查询 9

1.3.3 区间修改 + 区间查询 10

1.4 二维线段树 10

1.5 左偏树（可并堆） 12

1.5.1 左偏树 $O(\log n)$ 12

1.5.2 带 push_down 操作的左偏树子树节点合并（[JLOI2015] 城池攻占） . . . 13

1.6 扫描线 15

1.6.1 矩形并面积 15

1.6.2 矩形并周长 16

1.6.3 矩阵求和最值（POJ-2482） 17

1.6.4 旋转扫描线 19

1.7 李超线段树 19

1.7.1 李超上树（[SDOI2016] 游戏）$O(m\log^3 n)$ 19

2 杂项 **22**

2.1 数列归纳 22

2.2 全 1 矩阵个数（51nod1291） 22

3 习题整理 **23**

3.1 可重边集的点能否和当前询问边构成三角形（20 牛客 2H）（动态点开线段树） . . 23

3.2 左偏树离线处理查询成立最多数（HDU5575） 24

1 数据结构

1.1 01 字典树

1.1.1 路径为点权异或值求最小生成树（CF888G）

input	output
4 1 2 3 4	8

```
class Trie { public:
   int T[SIZE<<4][2], top;
   Trie() {
      top = 1;
      memset(T[0], 0, sizeof(T[0]));
   }
   void insert(int x) { // call: tree.insert(x);
      int u = 0;
      for (int i = 30; ~i; i--) {
         int ch = (x >> i) & 1;
         if(!T[u][ch]) {
            memset(T[top], 0, sizeof(T[top]));
            T[u][ch] = top++;
         }
         u = T[u][ch];
      }
   }
   ll query(int rt1, int rt2, int dp) {
      if(dp < 0) return (ll)0;
      ll res1 = -1, res2 = -1;
      if(T[rt1][0] && T[rt2][0]) res1 = query(T[rt1][0], T[rt2][0],
          dp-1);
      if(T[rt1][1] && T[rt2][1]) res2 = query(T[rt1][1], T[rt2][1],
          dp-1);
      if(~res1 && ~res2) return std::min(res1, res2);
      if(~res1) return res1; if(~res2) return res2;
      if(T[rt1][0] && T[rt2][1]) res1 = query(T[rt1][0], T[rt2][1],
          dp-1) + (1 << dp);
      if(T[rt1][1] && T[rt2][0]) res2 = query(T[rt1][1], T[rt2][0],
          dp-1) + (1 << dp);
      if(~res1 && ~res2) return std::min(res1, res2);
      if(~res1) return res1; if(~res2) return res2;
   }
} tree;
ll res;
void dfs(int a, int b) { // call: dfs(0, 30);
   if(b<0) return ;
   if(tree.T[a][0] && tree.T[a][1]) {
      res += 1ll * tree.query(tree.T[a][0], tree.T[a][1], b-1) + 1ll
          * (1 << b);
   }
   if(tree.T[a][0]) dfs(tree.T[a][0], b-1);
   if(tree.T[a][1]) dfs(tree.T[a][1], b-1);
}
```

1.1.2 可持久化 01 字典树

初始有 n 个数，有 m 个操作:

1 A x 添加操作，表示在序列末尾添加一个数 x ，序列的长度 $n+1$

Q l r x 询问操作，你需要找到一个位置 p ，满足 $l \le p \le r$，使得: $a[p] \oplus a[p+1] \oplus ... \oplus a[N] \oplus x$ 最大，输出最大是多少。

```
class HJT_01 { public:
   int ch[MAXN * 70][2], sum[MAXN * 70];
   int tot;
   int update(int rt, int v, int dep) {
      int nrt = ++tot, tmp = nrt;
      for (int i = 30; i >= 0; i--) {
         sum[nrt] = sum[rt] + 1; // 在原版本的基础上更新
```

```
        if ((v & (1 << i)) == 0) {
            if (!ch[nrt][0]) ch[nrt][0] = ++tot;
            ch[nrt][1] = ch[rt][1];
            nrt = ch[nrt][0];
            rt = ch[rt][0];
        } else {
            if (!ch[nrt][1]) ch[nrt][1] = ++tot;
            ch[nrt][0] = ch[rt][0];
            nrt = ch[nrt][1];
            rt = ch[rt][1];
        }
      }
      sum[nrt] = sum[rt] + 1;
      return tmp;
   }
   int query(int lrt, int rrt, int v) {
      int ans = 0;
      for (int i = 30; i >= 0; i--) {
         int t = ((v & (1 << i)) ? 1 : 0);
         if (sum[ch[rrt][!t]] - sum[ch[lrt][!t]]) {
            ans += (1 << i);
            lrt = ch[lrt][!t], rrt = ch[rrt][!t];
         } else lrt = ch[lrt][t], rrt = ch[rrt][t];
      }
      return ans;
   }
} tree;
int a[MAXN], pre[MAXN], root[MAXN];
char opt[5];
int main() {
   int n, m; scanf("%d%d", &n, &m);
   for (int i = 1; i <= n; i++) scanf("%d", &a[i]), pre[i] = pre[i -
       1] ^ a[i];
   int root_cnt = n;
   root[0] = 0;
   for (int i = 1; i <= n; i++) root[i] = tree.update(root[i - 1],
       pre[i], 30);
   while (m--) {
      scanf("%s", opt + 1);
      if (opt[1] == 'A') {
         root_cnt++;
         scanf("%d", &a[root_cnt]);
         pre[root_cnt] = pre[root_cnt - 1] ^ a[root_cnt];
         root[root_cnt] = tree.update(root[root_cnt - 1], pre[
             root_cnt], 30);
      } else {
         int l, r, x;
         scanf("%d%d%d", &l, &r, &x);
         l--, r--;
         if (l == r && l == 0) printf("%d\n", pre[root_cnt] ^ x);
         else printf("%d\n", tree.query(root[max(0, l - 1)], root[r],
             x ^ pre[root_cnt]));
      }
   }
}
```

1.2 吉司机线段树

1.2.1 区间取 min+ 区间查询 $O(m \log n)$

```
class JLS { public:
   struct node {
      int l, r;
      int fi_max, se_max, max_cnt; // 最大值，次大值，最大值个数
      ll sum;
   } T[MAXN << 2];
   int lazy[MAXN << 2];

```

```
inline void push_up(int rt) {
   T[rt].sum = T[rt << 1].sum + T[rt << 1 | 1].sum;
   if (T[rt << 1].fi_max == T[rt << 1 | 1].fi_max) { // 左右儿子的最大值相同
      T[rt].fi_max = T[rt << 1].fi_max;
      T[rt].se_max = max(T[rt << 1].se_max, T[rt << 1 | 1].se_max);
      T[rt].max_cnt = T[rt << 1].max_cnt + T[rt << 1 | 1].max_cnt;
   } else if (T[rt << 1].fi_max > T[rt << 1 | 1].fi_max) {
      T[rt].fi_max = T[rt << 1].fi_max;
      T[rt].se_max = max(T[rt << 1].se_max, T[rt << 1 | 1].fi_max);
      T[rt].max_cnt = T[rt << 1].max_cnt;
   } else {
      T[rt].fi_max = T[rt << 1 | 1].fi_max;
      T[rt].se_max = max(T[rt << 1].fi_max, T[rt << 1 | 1].se_max);
      T[rt].max_cnt = T[rt << 1 | 1].max_cnt;
   }
}

inline void push_down(int rt) {
   if (lazy[rt] != -1) {
      if (T[rt << 1].fi_max > lazy[rt]) { // left son
         T[rt << 1].sum += (1ll * lazy[rt] - T[rt << 1].fi_max) * T[rt << 1].max_cnt;
         T[rt << 1].fi_max = lazy[rt], lazy[rt << 1] = lazy[rt];
      }
      if (T[rt << 1 | 1].fi_max > lazy[rt]) { // right son
         T[rt << 1 | 1].sum += (1ll * lazy[rt] - T[rt << 1 | 1].fi_max) * T[rt << 1 | 1].max_cnt;
         T[rt << 1 | 1].fi_max = lazy[rt], lazy[rt << 1 | 1] = lazy[rt];
      }
      lazy[rt] = -1;
   }
}

void build(int rt, int l, int r) {
   T[rt].l = l, T[rt].r = r;
   lazy[rt] = -1;
   if (l == r) {
      T[rt].sum = T[rt].fi_max = a[l], T[rt].se_max = -1, T[rt].max_cnt = 1;
      return;
   }
   int mid = (l + r) >> 1;
   build(rt << 1, l, mid), build(rt << 1 | 1, mid + 1, r);
   push_up(rt);
}

void update_min(int rt, int L, int R, int val) { // a[L],...,a[R] <- min(val, a[i])
   if (T[rt].fi_max <= val) return;
   if (L <= T[rt].l && T[rt].r <= R && T[rt].se_max < val) {
      if (T[rt].fi_max > val) {
         T[rt].sum += (1ll * val - T[rt].fi_max) * T[rt].max_cnt;
         T[rt].fi_max = val, lazy[rt] = val;
      }
      return;
   }
   push_down(rt);
   int mid = (T[rt].l + T[rt].r) >> 1;
   if (L <= mid) update_min(rt << 1, L, R, val);
   if (R > mid) update_min(rt << 1 | 1, L, R, val);
   push_up(rt);
}
```

```
    int query_max(int rt, int L, int R) { // find max value
        if (L <= T[rt].l && T[rt].r <= R) return T[rt].fi_max;
        push_down(rt);
        int mid = (T[rt].l + T[rt].r) >> 1;
        int ans = -1;
        if (L <= mid) ans = max(ans, query_max(rt << 1, L, R));
        if (R > mid) ans = max(ans, query_max(rt << 1 | 1, L, R));
        return ans;
    }

    ll query_sum(int rt, int L, int R) {
        if (L <= T[rt].l && T[rt].r <= R) return T[rt].sum;
        push_down(rt);
        int mid = (T[rt].l + T[rt].r) >> 1;
        ll ans = 0;
        if (L <= mid) ans += query_sum(rt << 1, L, R);
        if (R > mid) ans += query_sum(rt << 1 | 1, L, R);
        return ans;
    }
} tree;
```

1.2.2 支持区间加（BZOJ4695 最假女选手）$O(m\log^2 n)$

```
class JLS { public:
    struct node {
        int l, r;
        int fi_max, se_max, fi_min, se_min;
        int cnt_max, cnt_min;
        ll sum;
    } T[MAXN << 2];
    ll add[MAXN << 2];
#define lson rt<<1
#define rson rt<<1|1
    inline void push_up(int rt) {
        T[rt].sum = T[lson].sum + T[rson].sum;
        // max
        if (T[lson].fi_max == T[rson].fi_max) {
            T[rt].fi_max = T[lson].fi_max;
            T[rt].se_max = max(T[lson].se_max, T[rson].se_max);
            T[rt].cnt_max = T[lson].cnt_max + T[rson].cnt_max;
        } else if (T[lson].fi_max > T[rson].fi_max) {
            T[rt].fi_max = T[lson].fi_max;
            T[rt].se_max = max(T[lson].se_max, T[rson].fi_max);
            T[rt].cnt_max = T[lson].cnt_max;
        } else {
            T[rt].fi_max = T[rson].fi_max;
            T[rt].se_max = max(T[lson].fi_max, T[rson].se_max);
            T[rt].cnt_max = T[rson].cnt_max;
        }
        // min
        if (T[lson].fi_min == T[rson].fi_min) {
            T[rt].fi_min = T[lson].fi_min;
            T[rt].se_min = min(T[lson].se_min, T[rson].se_min);
            T[rt].cnt_min = T[lson].cnt_min + T[rson].cnt_min;
        } else if (T[lson].fi_min < T[rson].fi_min) {
            T[rt].fi_min = T[lson].fi_min;
            T[rt].se_min = min(T[lson].se_min, T[rson].fi_min);
            T[rt].cnt_min = T[lson].cnt_min;
        } else {
            T[rt].fi_min = T[rson].fi_min;
            T[rt].se_min = min(T[lson].fi_min, T[rson].se_min);
            T[rt].cnt_min = T[rson].cnt_min;
        }
    }

    inline void push_add(int rt, int tg) {
        T[rt].sum += (ll) (T[rt].r - T[rt].l + 1) * tg;
        T[rt].fi_max += tg, T[rt].fi_min += tg;
```

```
    if (T[rt].se_max != -inf) T[rt].se_max += tg;
    if (T[rt].se_min != inf) T[rt].se_min += tg;
    add[rt] += tg;
}

inline void push_min(int rt, int tg) {
    T[rt].sum = T[rt].sum - (ll) (T[rt].fi_max - tg) * T[rt].cnt_max;
    if (T[rt].fi_max == T[rt].fi_min) T[rt].fi_max = T[rt].fi_min = tg;
    else if (T[rt].fi_max == T[rt].se_min) T[rt].fi_max = T[rt].se_min = tg;
    else T[rt].fi_max = tg;
}

inline void push_max(int rt, int tg) {
    T[rt].sum = T[rt].sum + (ll) (tg - T[rt].fi_min) * T[rt].cnt_min;
    if (T[rt].fi_min == T[rt].fi_max) T[rt].fi_min = T[rt].fi_max = tg;
    else if (T[rt].fi_min == T[rt].se_max) T[rt].fi_min = T[rt].se_max = tg;
    else T[rt].fi_min = tg;
}

inline void push_down(int rt) {
    if (add[rt]) {
        push_add(lson, add[rt]), push_add(rson, add[rt]);
        add[rt] = 0;
    }
    if (T[rt].fi_max < T[lson].fi_max) push_min(lson, T[rt].fi_max);
    if (T[rt].fi_max < T[rson].fi_max) push_min(rson, T[rt].fi_max);
    if (T[rt].fi_min > T[lson].fi_min) push_max(lson, T[rt].fi_min);
    if (T[rt].fi_min > T[rson].fi_min) push_max(rson, T[rt].fi_min);
}

void build(int rt, int l, int r) {
    T[rt].l = l, T[rt].r = r;
    add[rt] = 0;
    if (l == r) {
        T[rt].sum = T[rt].fi_max = T[rt].fi_min = a[l];
        T[rt].se_max = -inf, T[rt].se_min = inf;
        T[rt].cnt_min = T[rt].cnt_max = 1;
        return;
    }
    int mid = (l + r) >> 1;
    build(lson, l, mid), build(rson, mid + 1, r);
    push_up(rt);
}

void update_add(int rt, int L, int R, int v) { // add v to [L, R]
    if (L <= T[rt].l && T[rt].r <= R) {
        push_add(rt, v);
        return;
    }
    push_down(rt);
    int mid = (T[rt].l + T[rt].r) >> 1;
    if (L <= mid) update_add(lson, L, R, v);
    if (R > mid) update_add(rson, L, R, v);
    push_up(rt);
}

void update_min(int rt, int L, int R, int v) { // a[L],...,a[R] <- min(val, a[i])
```

```
    if (v >= T[rt].fi_max) return;
    if (L <= T[rt].l && T[rt].r <= R && T[rt].se_max < v) {
      push_min(rt, v);
      return;
    }
    push_down(rt);
    int mid = (T[rt].l + T[rt].r) >> 1;
    if (L <= mid) update_min(lson, L, R, v);
    if (R > mid) update_min(rson, L, R, v);
    push_up(rt);
  }

  void update_max(int rt, int L, int R, int v) { // a[L],...,a[R] <-
      max(val, a[i])
    if (v <= T[rt].fi_min) return;
    if (L <= T[rt].l && T[rt].r <= R && T[rt].se_min > v) {
      push_max(rt, v);
      return;
    }
    push_down(rt);
    int mid = (T[rt].l + T[rt].r) >> 1;
    if (L <= mid) update_max(lson, L, R, v);
    if (R > mid) update_max(rson, L, R, v);
    push_up(rt);
  }

  ll query_sum(int rt, int L, int R) {
    if (L <= T[rt].l && T[rt].r <= R) return T[rt].sum;
    push_down(rt);
    int mid = (T[rt].l + T[rt].r) >> 1;
    ll ans = 0;
    if (L <= mid) ans += query_sum(lson, L, R);
    if (R > mid) ans += query_sum(rson, L, R);
    return ans;
  }

  int query_max(int rt, int L, int R) {
    if (L <= T[rt].l && T[rt].r <= R) return T[rt].fi_max;
    push_down(rt);
    int mid = (T[rt].l + T[rt].r) >> 1;
    int ans = -inf;
    if (L <= mid) ans = max(ans, query_max(lson, L, R));
    if (R > mid) ans = max(ans, query_max(rson, L, R));
    return ans;
  }

  int query_min(int rt, int L, int R) {
    if (L <= T[rt].l && T[rt].r <= R) return T[rt].fi_min;
    push_down(rt);
    int mid = (T[rt].l + T[rt].r) >> 1;
    int ans = inf;
    if (L <= mid) ans = min(ans, query_min(lson, L, R));
    if (R > mid) ans = min(ans, query_min(rson, L, R));
    return ans;
  }
} tree;
```

1.2.3 维护区间最值操作与区间历史最值（洛谷线段树 3）$O(m\log^2 n)$

给出一个长度为 n 的数列 A，同时定义一个辅助数组 B，B 开始与 A 完全相同。接下来进行了 m 次操作，操作有五种类型，按以下格式给出：

1 l r k 对于所有的 $i\in[l,r]$，将 A_i 加上 k（k 可以为负数）。

2 l r v 对于所有的 $i\in[l,r]$，将 A_i 变成 $\min(A_i,v)$。

3 l r 求 $\sum_{i=l}^{r}A_i$。

4 l r 对于所有的 $i\in[l,r]$，求 A_i 的最大值。

5 l r 对于所有的 $i\in[l,r]$，求 B_i 的最大值。

在每一次操作后，我们都进行一次更新，让 $B_i\leftarrow\max(B_i,A_i)$。

```
class JLS { public:
```

```
    struct node1 { // 实时信息
        int l, r; // 左端点，右端点
        int fi_max, se_max, cnt_max; // 当前区间最大值，区间次大值，区间最大值个数
        int fi_add, se_add; // 区间最大值lazy标记，区间次大值lazy标记
        ll sum; // 当前区间和
    } now[MAXN << 2];
    struct node2 { // 历史信息
        int fi_max; // 历史区间最大值
        int fi_add, se_add; // 历史区间最大值lazy标记，区间次大值lazy标记
    } old[MAXN << 2];
#define lson rt<<1
#define rson rt<<1|1
    inline void push_up(int rt) {
        now[rt].sum = now[lson].sum + now[rson].sum;
        old[rt].fi_max = max(old[lson].fi_max, old[rson].fi_max);
        // max
        if (now[lson].fi_max == now[rson].fi_max) {
            now[rt].fi_max = now[lson].fi_max;
            now[rt].se_max = max(now[lson].se_max, now[rson].se_max);
            now[rt].cnt_max = now[lson].cnt_max + now[rson].cnt_max;
        } else if (now[lson].fi_max > now[rson].fi_max) {
            now[rt].fi_max = now[lson].fi_max;
            now[rt].se_max = max(now[lson].se_max, now[rson].fi_max);
            now[rt].cnt_max = now[lson].cnt_max;
        } else {
            now[rt].fi_max = now[rson].fi_max;
            now[rt].se_max = max(now[lson].fi_max, now[rson].se_max);
            now[rt].cnt_max = now[rson].cnt_max;
        }
    }

    // v1, v3 change now; v2, v4 change old;
    inline void push_node(int rt, int v1, int v2, int v3, int v4) {
        old[rt].fi_max = max(old[rt].fi_max, now[rt].fi_max + v2);
        old[rt].fi_add = max(old[rt].fi_add, now[rt].fi_add + v2);
        old[rt].se_add = max(old[rt].se_add, now[rt].se_add + v4);

        now[rt].sum += 1ll * v1 * now[rt].cnt_max +
                       1ll * v3 * (now[rt].r - now[rt].l + 1 - now[rt].cnt_max);
        now[rt].fi_max += v1;
        if (now[rt].se_max != inf) now[rt].se_max += v3;
        now[rt].fi_add += v1, now[rt].se_add += v3;
    }

    inline void push_down(int rt) {
        int tmp = max(now[lson].fi_max, now[rson].fi_max);

        if (now[lson].fi_max == tmp) push_node(lson, now[rt].fi_add, old[rt].fi_add, now[rt].se_add, old[rt].se_add);
        else push_node(lson, now[rt].se_add, old[rt].se_add, now[rt].se_add, old[rt].se_add);

        if (now[rson].fi_max == tmp) push_node(rson, now[rt].fi_add, old[rt].fi_add, now[rt].se_add, old[rt].se_add);
        else push_node(rson, now[rt].se_add, old[rt].se_add, now[rt].se_add, old[rt].se_add);

        now[rt].fi_add = now[rt].se_add = old[rt].fi_add = old[rt].se_add = 0;
    }

    void build(int rt, int l, int r) {
        now[rt].l = l, now[rt].r = r;
        now[rt].fi_add = now[rt].se_add = 0;
        old[rt].fi_add = old[rt].se_add = 0;
        if (l == r) {
```

```
        now[rt].sum = now[rt].fi_max = a[l];
        now[rt].cnt_max = 1;
        now[rt].se_max = -inf;

        old[rt].fi_max = a[l];
        return;
    }
    int mid = (l + r) >> 1;
    build(lson, l, mid), build(rson, mid + 1, r);
    push_up(rt);
}

void update_add(int rt, int L, int R, int v) { // op1
    if (L <= now[rt].l && now[rt].r <= R) {
        push_node(rt, v, v, v, v);
        return;
    }
    push_down(rt);
    int mid = (now[rt].l + now[rt].r) >> 1;
    if (L <= mid) update_add(lson, L, R, v);
    if (R > mid) update_add(rson, L, R, v);
    push_up(rt);
}

void update_min(int rt, int L, int R, int v) { // op2
    if (v >= now[rt].fi_max) return;
    if (L <= now[rt].l && now[rt].r <= R && now[rt].se_max < v) {
        push_node(rt, v - now[rt].fi_max, v - now[rt].fi_max, 0, 0);
        return;
    }
    push_down(rt);
    int mid = (now[rt].l + now[rt].r) >> 1;
    if (L <= mid) update_min(lson, L, R, v);
    if (R > mid) update_min(rson, L, R, v);
    push_up(rt);
}

ll query_sum(int rt, int L, int R) { // op3
    if (L <= now[rt].l && now[rt].r <= R) return now[rt].sum;
    push_down(rt);
    int mid = (now[rt].l + now[rt].r) >> 1;
    ll ans = 0;
    if (L <= mid) ans += query_sum(lson, L, R);
    if (R > mid) ans += query_sum(rson, L, R);
    return ans;
}

int query_max1(int rt, int L, int R) { // op4
    if (L <= now[rt].l && now[rt].r <= R) return now[rt].fi_max;
    push_down(rt);
    int mid = (now[rt].l + now[rt].r) >> 1;
    int ans = -inf;
    if (L <= mid) ans = max(ans, query_max1(lson, L, R));
    if (R > mid) ans = max(ans, query_max1(rson, L, R));
    return ans;
}

int query_max2(int rt, int L, int R) { // op5
    if (L <= now[rt].l && now[rt].r <= R) return old[rt].fi_max;
    push_down(rt);
    int mid = (now[rt].l + now[rt].r) >> 1;
    int ans = -inf;
    if (L <= mid) ans = max(ans, query_max2(lson, L, R));
    if (R > mid) ans = max(ans, query_max2(rson, L, R));
    return ans;
}
#undef lson
#undef rson
```

```
} tree;
```

1.3 二维树状数组

1.3.1 单点修改 + 区间查询

```
class BIT { public:
   ll val[MAXN][MAXN];
   int n, m;
   void init(int _n, int _m) {
      n = _n, m = _m;
      for (int i = 1; i <= n; i++) {
         for (int j = 1; j <= m; j++) {
            val[i][j] = 0;
         }
      }
   }
   inline int lowbit(int x) { return x & (-x); }
   void add(int x, int y, ll v) {
      for (int i = x; i <= n; i += lowbit(i)) {
         for (int j = y; j <= m; j += lowbit(j)) {
            val[i][j] += v;
         }
      }
   }
   inline ll query(int x, int y) {
      ll ans = 0;
      for (int i = x; i >= 1; i -= lowbit(i)) {
         for (int j = y; j >= 1; j -= lowbit(j)) {
            ans += val[i][j];
         }
      }
      return ans;
   }
   ll query(int x1, int y1, int x2, int y2) { // x1 <= y1, x2 <= y2
      return query(x2, y2) - query(x1 - 1, y2) - query(x2, y1 - 1) +
           query(x1 - 1, y1 - 1);
   }
} tree;
```

1.3.2 区间修改 + 单点查询

```
class BIT { public:
   ll val[MAXN][MAXN];
   int n, m;
   void init(int _n, int _m);
   inline int lowbit(int x);
   inline void add(int x, int y, ll v) {
      for (int i = x; i <= n; i += lowbit(i)) {
         for (int j = y; j <= m; j += lowbit(j)) {
            val[i][j] += v;
         }
      }
   }
   ll query(int x, int y) {
      ll ans = 0;
      for (int i = x; i >= 1; i -= lowbit(i)) {
         for (int j = y; j >= 1; j -= lowbit(j)) {
            ans += val[i][j];
         }
      }
      return ans;
   }
   void change(int x1, int y1, int x2, int y2, ll v) { // x1 <= y1,
       x2 <= y2
      add(x1, y1, v);
      add(x2 + 1, y2 + 1, v);
      add(x2 + 1, y1, -v);
```

```
        add(x1, y2 + 1, -v);
    }
} tree;
```

1.3.3 区间修改 + 区间查询

```
class BIT { public:
    ll val[MAXN][MAXN][4];
    int n, m;
    void init(int _n, int _m) {
        n = _n, m = _m;
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= m; j++) {
                for (int k = 0; k < 4; k++) {
                    val[i][j][k] = 0;
                }
            }
        }
    }
    inline int lowbit(int x) { return x & (-x); }
    inline void add(int x, int y, ll v) {
        for (int i = x; i <= n; i += lowbit(i)) {
            for (int j = y; j <= m; j += lowbit(j)) {
                val[i][j][0] += v, val[i][j][1] += v * x, val[i][j][2] +=
                    v * y, val[i][j][3] += v * x * y;
            }
        }
    }
    void change(int x1, int y1, int x2, int y2, ll v) { // x1 <= y1,
        x2 <= y2
        add(x1, y1, v);
        add(x2 + 1, y2 + 1, v);
        add(x2 + 1, y1, -v);
        add(x1, y2 + 1, -v);
    }
    inline ll query(int x, int y) {
        ll ans = 0;
        for (int i = x; i >= 1; i -= lowbit(i)) {
            for (int j = y; j >= 1; j -= lowbit(j)) {
                ans += (ll) (x + 1) * (y + 1) * val[i][j][0] - (ll) (y +
                    1) * val[i][j][1] -
                    (ll) (x + 1) * val[i][j][2] + val[i][j][3];
            }
        }
        return ans;
    }
    ll sum(int x1, int y1, int x2, int y2) { // x1 <= y1, x2 <= y2
        return query(x2, y2) - query(x1 - 1, y2) - query(x2, y1 - 1) +
            query(x1 - 1, y1 - 1);
    }
} tree;
```

1.4 二维线段树

```
class SEG2D { public:
    int n;
    int maxx[MAXN << 2][MAXN << 2], minn[MAXN << 2][MAXN << 2];
    void init(int _n) { n = _n; }
    inline void subPush_up(int frt, int rt) {
        maxx[frt][rt] = max(maxx[frt][rt << 1], maxx[frt][rt << 1 | 1])
            ;
        minn[frt][rt] = min(minn[frt][rt << 1], minn[frt][rt << 1 | 1])
            ;
    }
    inline void push_up(int frt, int rt) {
        maxx[frt][rt] = max(maxx[frt << 1][rt], maxx[frt << 1 | 1][rt])
            ;
```

```
    minn[frt][rt] = min(minn[frt << 1][rt], minn[frt << 1 | 1][rt]);
}

inline void subBuild(int frt, int fl, int fr, int rt, int l, int r) {
    if (l == r) {
        if (fl == fr) maxx[frt][rt] = minn[frt][rt] = a[fl][l];
        else push_up(frt, rt);
        return;
    }
    int mid = (l + r) >> 1;
    subBuild(frt, fl, fr, rt << 1, l, mid), subBuild(frt, fl, fr, rt << 1 | 1, mid + 1, r);
    subPush_up(frt, rt);
}
void build(int rt, int l, int r) {
    if (l == r) {
        subBuild(rt, l, r, 1, 1, n);
        return;
    }
    int mid = (l + r) >> 1;
    build(rt << 1, l, mid), build(rt << 1 | 1, mid + 1, r);
    subBuild(rt, l, r, 1, 1, n);
}

inline void subUpdate(int frt, int fl, int fr, int rt, int x, int y, int v, int be, int en) {
    if (be == en) {
        if (fl == fr) maxx[frt][rt] = minn[frt][rt] = v;
        else push_up(frt, rt);
        return;
    }
    int mid = (be + en) >> 1;
    if (y <= mid) subUpdate(frt, fl, fr, rt << 1, x, y, v, be, mid);
    else subUpdate(frt, fl, fr, rt << 1 | 1, x, y, v, mid + 1, en);
    subPush_up(frt, rt);
}
void update(int rt, int x, int y, int v, int be, int en) {
    if (be == en) {
        subUpdate(rt, be, en, 1, x, y, v, 1, n);
        return;
    }
    int mid = (be + en) >> 1;
    if (x <= mid) update(rt << 1, x, y, v, be, mid);
    else update(rt << 1 | 1, x, y, v, mid + 1, en);
    subUpdate(rt, be, en, 1, x, y, v, 1, n);
}

inline pii push_pii(const pii &ta, const pii &tb) {
    return mp(max(ta.first, tb.first), min(ta.second, tb.second));
}
inline pii subQuery(int frt, int rt, int x1, int x2, int y1, int y2, int be, int en) {
    if (y1 <= be && en <= y2) return mp(maxx[frt][rt], minn[frt][rt]);
    int mid = (be + en) >> 1;
    pii ans = mp(-inf, inf);
    if (y1 <= mid) ans = push_pii(ans, subQuery(frt, rt << 1, x1, x2, y1, y2, be, mid));
    if (y2 > mid) ans = push_pii(ans, subQuery(frt, rt << 1 | 1, x1, x2, y1, y2, mid + 1, en));
    return ans;
}
pii query(int rt, int x1, int x2, int y1, int y2, int be, int en) { // x1 <= x2, y1 <= y2
```

```
68      if (x1 <= be && en <= x2) return subQuery(rt, 1, x1, x2, y1, y2
            , 1, n);
69      int mid = (be + en) >> 1;
70      pii ans = mp(-inf, inf);
71      if (x1 <= mid) ans = push_pii(ans, query(rt << 1, x1, x2, y1,
            y2, be, mid));
72      if (x2 > mid) ans = push_pii(ans, query(rt << 1 | 1, x1, x2, y1
            , y2, mid + 1, en));
73      return ans;
74    }
75  } tree;
```

1.5 左偏树（可并堆）

1.5.1 左偏树 $O(\log n)$

```
1   class LT { public:
2     int pool[MAXN], pool_cnt;
3
4     int fa[MAXN];
5     int find(int x) {
6       if (x == fa[x]) return x;
7       else return fa[x] = find(fa[x]);
8     }
9
10    int val[MAXN], ch[MAXN][2], dist[MAXN];
11
12    int tot;
13    inline int New() {
14      return pool_cnt ? pool[pool_cnt--] : ++tot;
15    }
16    inline void Del(int &rt) {
17      pool[++pool_cnt] = rt;
18      fa[rt] = val[rt] = ch[rt][0] = ch[rt][1] = dist[rt] = 0;
19      rt = 0;
20    }
21    void init() { tot = 0, pool_cnt = 0; }
22    inline int Newnode(int v) { // 初始化左偏树节点
23      int nrt = New();
24      val[nrt] = v, ch[nrt][0] = ch[nrt][1] = dist[nrt] = 0;
25      fa[nrt] = nrt;
26      return nrt;
27    }
28
29    int merge(int x, int y) { // 合并左偏树，
30                              // call：root[fx] = tree.merge(root[fx], root[
                                  fy]);
31      if (!x || !y) return x + y;
32      if (val[x] == val[y] ? x > y : val[x] > val[y]) swap(x, y); //
            小根堆
33      ch[x][1] = merge(ch[x][1], y);
34      if (dist[ch[x][0]] < dist[ch[x][1]]) swap(ch[x][0], ch[x][1]);
35      fa[ch[x][0]] = fa[ch[x][1]] = fa[x] = x;
36      dist[x] = dist[ch[x][1]] + 1;
37      return x;
38    }
39    int insert(int rt, int v) { // call：root[x] = tree.insert(root[x],
            y);
40      return merge(rt, Newnode(v));
41    }
42    int pop(int rt) { // call：root[fx] = tree.pop(root[fx]);
43      int tl = ch[rt][0], tr = ch[rt][1];
44      Del(rt);
45      return merge(tl, tr);
46    }
47    bool isempty(int x) { // call：tree.isempty(root[fx])
48                          // 为空返回1，不为空返回0
49      return x == 0;
```

```
    }
    int top(int x) { // call: tree.top(root[fx])
        return val[x];
    }

} tree;
```

1.5.2 带 push_down 操作的左偏树子树节点合并（[JLOI2015] 城池攻占）

```
#define pii pair<int , ll>
class LT { public:
    int pool[MAXN], pool_cnt;
    int fa[MAXN];
    int find(int x) {
        if (x == fa[x]) return x;
        else return fa[x] = find(fa[x]);
    }
    struct node {
        int id; ll v;
        node() {}
        node(int _id, ll _v) { id = _id, v = _v; }
        bool operator<(const node &tb) { return v < tb.v; }
        bool operator==(const node &tb) { return v == tb.v; }
        bool operator>(const node &tb) { return v > tb.v; }
    } val[MAXN];
    ll add[MAXN], mul[MAXN];

    int ch[MAXN][2], dist[MAXN];
    int tot;

    void init() { tot = 0, pool_cnt = 0; }
    inline int Newnode(ll v, int id) { // 初始化左偏树节点
        int nrt = New();
        val[nrt].v = v, val[nrt].id = id, ch[nrt][0] = ch[nrt][1] =
            dist[nrt] = 0;
        mul[nrt] = 1, add[nrt] = 0;
        fa[nrt] = nrt;
        return nrt;
    }
#define lson ch[rt][0]
#define rson ch[rt][1]
    inline void push_down(int rt) {
        if (mul[rt] != 1) {
            if (lson) val[lson].v *= mul[rt], add[lson] *= mul[rt], mul[
                lson] *= mul[rt];
            if (rson) val[rson].v *= mul[rt], add[rson] *= mul[rt], mul[
                rson] *= mul[rt];
            mul[rt] = 1;
        }
        if (add[rt]) {
            if (lson) val[lson].v += add[rt], add[lson] += add[rt];
            if (rson) val[rson].v += add[rt], add[rson] += add[rt];
            add[rt] = 0;
        }
    }
    int merge(int x, int y) { // 合并左偏树
        if (!x || !y) return x + y;
        if (val[x] == val[y] ? x > y : val[x] > val[y]) swap(x, y); //
            小根堆
        push_down(x);
        ch[x][1] = merge(ch[x][1], y);
        if (dist[ch[x][0]] < dist[ch[x][1]]) swap(ch[x][0], ch[x][1]);
        fa[ch[x][0]] = fa[ch[x][1]] = fa[x] = x;
        dist[x] = dist[ch[x][1]] + 1;
        return x;
    }

```

```
    int insert(int rt, ll v, int id) { return merge(rt, Newnode(v, id));}
    int pop(int rt) { // call: root[fx] = tree.pop(root[fx]);
        push_down(rt);
        int tl = ch[rt][0], tr = ch[rt][1];
        Del(rt);
        return merge(tl, tr);
    }
    bool isempty(int x) { return x == 0;}
    pii top(int x) { return mp(val[x].id, val[x].v); }
} tree;

ll h[MAXN];
int f[MAXN], a[MAXN], c[MAXN];
ll v[MAXN], s[MAXN];

int root[MAXN]; int dep[MAXN]; int res1[MAXN], res2[MAXN];
void dfs(int u, int father) {
    dep[u] = dep[father] + 1;
    for (int i = head[u]; ~i; i = e[i].nex) {
        int tv = e[i].to;
        dfs(tv, u);
    }
    while (!tree.isempty(root[u]) && tree.top(root[u]).second < h[u]) {
        res1[u]++;
        int id = tree.top(root[u]).first;
        res2[id] = dep[c[id]] - dep[u];
        root[u] = tree.pop(root[u]);
    }
    if (tree.isempty(root[u])) return;
    if (a[u]) { // mul
        int ru = root[u];
        tree.val[ru].v *= v[u], tree.add[ru] *= v[u], tree.mul[ru] *= v[u];
    } else {
        int ru = root[u];
        tree.val[ru].v += v[u], tree.add[ru] += v[u];
    }
    if (u == 1) {
        while (!tree.isempty(root[1])) {
            int id = tree.top(root[1]).first;
            res2[id] = dep[c[id]] - dep[1] + 1;
            root[1] = tree.pop(root[1]);
        }
    } else {
        root[f[u]] = tree.merge(root[f[u]], root[u]);
    }
}

int main() {
    int n, m; scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) head[i] = -1;
    for (int i = 1; i <= n; i++) scanf("%lld", &h[i]);
    for (int i = 2; i <= n; i++) {
        scanf("%d%d%lld", &f[i], &a[i], &v[i]);
        addEdge(f[i], i);
    }
    for (int i = 1; i <= m; i++) {
        scanf("%lld%d", &s[i], &c[i]);
        root[c[i]] = tree.insert(root[c[i]], s[i], i); // build a lot of heap
    }
    dfs(1, 1);
    for (int i = 1; i <= n; i++) printf("%d\n", res1[i]);
    for (int i = 1; i <= m; i++) printf("%d\n", res2[i]);
}
```

1.6 扫描线

1.6.1 矩形并面积

```
namespace Discrete {
   ll b[MAXN << 1];
   int tol = 1, blen = 0;
   inline void push(ll x) { b[tol++] = x; }
   void init() { blen = 0; tol = 1;}
   void build() {
      sort(b + 1, b + tol);
      blen = unique(b + 1, b + tol) - (b + 1);
   }
};
using namespace Discrete;
struct Line {
   ll x, y1, y2;
   int mark;
   Line() {}
   Line(ll _x, ll _y1, ll _y2, int _mark) {
      x = _x, y1 = _y1, y2 = _y2, mark = _mark;
   }
   bool operator<(const Line &tb) { return x < tb.x;}
} line[MAXN << 1];

class Seg_Tree { public:
   struct node {
      int l, r, val;
      ll len;
   } T[MAXN << 2];
   inline void push_up(int rt) {
      int l = T[rt].l, r = T[rt].r;
      if (T[rt].val) T[rt].len = b[r + 1] - b[l];
      else T[rt].len = T[rt << 1].len + T[rt << 1 | 1].len;
   }

   void build(int l, int r, int rt) {
      T[rt].l = l, T[rt].r = r;
      T[rt].val = 0;
      T[rt].len = 0;
      if (l == r) {
         T[rt<<1].val = T[rt<<1|1].val = T[rt<<1].len = T[rt<<1|1].
             len = 0;
         return ;
      }
      int mid = (l + r) >> 1;
      build(l, mid, rt << 1), build(mid + 1, r, rt << 1 | 1);
   }
   void update(ll L, ll R, int c, int rt) {
      int l = T[rt].l, r = T[rt].r;
      if (b[r + 1] <= L || R <= b[l]) return;
      if (L <= b[l] && b[r + 1] <= R) {
         T[rt].val += c;
         push_up(rt);
         return;
      }
      update(L, R, c, rt << 1); update(L, R, c, rt << 1 | 1);
      push_up(rt);
   }
} tree;

int main() {
   int n; int kase = 0;
   while (~scanf("%d", &n) && n) {
      init();
      for (int i = 1; i <= n; i++) {
         ll x1, y1, x2, y2; scanf("%lf %lf %lf %lf", &x1, &y1, &x2, &y2
             );
         push(y1); push(y2);
```

```
            line[2 * i - 1] = Line(x1, y1, y2, 1);
            line[2 * i] = Line(x2, y1, y2, -1);
        }
        n <<= 1;
        sort(line + 1, line + 1 + n);
        build();
        tree.build(1, blen - 1, 1);
        ll res = 0;
        for (int i = 1; i < n; i++) {
            tree.update(line[i].y1, line[i].y2, line[i].mark, 1);
            res += tree.T[1].len * (line[i + 1].x - line[i].x);
        }
        printf("Test case #%d\n", ++kase);
        printf("Total explored area: %.2f\n", res);
        printf("\n");
    }
}
```

1.6.2 矩形并周长

```
using namespace Discrete; // 矩形并面积中的namespace Discrete
struct Line {
    int x, y1, y2;
    int mark;
    Line() {}
    Line(int _x, int _y1, int _y2, int _mark) {
        x = _x, y1 = _y1, y2 = _y2, mark = _mark;
    }
    bool operator<(const Line &tb) const {
        if (x == tb.x) return mark > tb.mark; // 如果出现了两条高度相同的扫描线，也就是两矩形相邻,那么需要先扫底边再扫顶边，否则就会多算这条边
        return x < tb.x;
    }
} line[MAXN << 1];
```

```
class Seg_Tree { public:
    struct SegTree {
        int l, r, sum, len, c;
// c表示区间线段条数
        bool lc, rc;
// lc, rc分别表示左、右端点是否被覆盖
// 统计线段条数(tree[x].c)会用到
    } tree[MAXN << 3];

    void build_tree(int x, int l, int r) {
        tree[x].l = l, tree[x].r = r;
        tree[x].lc = tree[x].rc = false;
        tree[x].sum = tree[x].len = 0;
        tree[x].c = 0;
        if (l == r) return;
        int mid = (l + r) >> 1;
        build_tree(x<<1, l, mid);
        build_tree(x<<1|1, mid + 1, r);
    }

    void pushup(int x) {
        int l = tree[x].l, r = tree[x].r;
        if (tree[x].sum) {
            tree[x].len = b[r + 1] - b[l];
            tree[x].lc = tree[x].rc = true;
            tree[x].c = 1;
// 做好相应的标记
        } else {
            tree[x].len = tree[x<<1].len + tree[x<<1|1].len;
            tree[x].lc = tree[x<<1].lc, tree[x].rc = tree[x<<1|1].rc;
            tree[x].c = tree[x<<1].c + tree[x<<1|1].c;
// 如果左儿子左端点被覆盖，那么自己的左端点也肯定被覆盖；右儿子同理
            if (tree[x<<1].rc && tree[x<<1|1].lc) tree[x].c -= 1;
// 如果做儿子右端点和右儿子左端点都被覆盖，
```

```
// 那么中间就是连续的一段，所以要 -= 1
      }
   }

   void edit_tree(int x, int L, int R, int c) {
      int l = tree[x].l, r = tree[x].r;
      if (b[l] >= R || b[r + 1] <= L) return;
      if (L <= b[l] && b[r + 1] <= R) {
         tree[x].sum += c;
         pushup(x);
         return;
      }
      edit_tree(x<<1, L, R, c), edit_tree(x<<1|1, L, R, c);
      pushup(x);
   }
}tree;
int main() {
   int n; scanf("%d", &n);
   init();
   for (int i = 1; i <= n; i++) {
      int x1, y1, x2, y2;
      scanf("%d %d %d %d", &x1, &y1, &x2, &y2);
      push(y1), push(y2);
      line[2 * i - 1] = Line(x1, y1, y2, 1);
      line[2 * i] = Line(x2, y1, y2, -1);
   }
   n <<= 1;
   sort(line + 1, line + 1 + n);
   build(); tree.build_tree(1, 1, blen-1);
   ll res = 0; int pre = 0;
   for (int i = 1; i < n; i++) {
      tree.edit_tree(1, line[i].y1, line[i].y2, line[i].mark);
      res += 1ll * abs(pre - tree.tree[1].len);
      pre = tree.tree[1].len;
      res += 2ll * tree.tree[1].c * (line[i + 1].x - line[i].x);
   }
   res += 1ll * (line[n].y2 - line[n].y1);
   printf("%lld\n", res);
}
```

1.6.3 矩阵求和最值（POJ-2482）

不含边框注意！

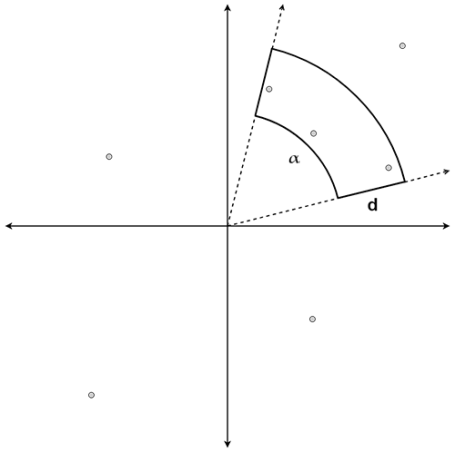
input	output
2 3 5 4 1 2 3 2 3 2 6 3 1 3 5 4 1 2 3 2 3 2 5 3 1	5 6

```
using namespace Discrete; // 矩形并面积中的namespace Discrete
struct LINE {
   int x, y1, y2, l;
   LINE() {}
   LINE(int _x, int _y1, int _y2, int _l) {
      x = _x, y1 = _y1, y2 = _y2, l = _l;
   }
} line[MAXN << 1];
class SEG { public:
   struct node {
      int l, r, maxx;
   } T[MAXN << 3]; // 8倍注意！
   int lazy[MAXN << 3];

```

```
    inline void push_up(int rt) {
        T[rt].maxx = max(T[rt << 1].maxx, T[rt << 1 | 1].maxx);
    }
    inline void push_down(int rt) {
        if (lazy[rt]) {
            T[rt << 1].maxx += lazy[rt], lazy[rt << 1] += lazy[rt];
            T[rt << 1 | 1].maxx += lazy[rt], lazy[rt << 1 | 1] += lazy[
                rt];
            lazy[rt] = 0;
        }
    }

    void build(int rt, int l, int r) {
        T[rt].l = l, T[rt].r = r;
        lazy[rt] = 0;
        if (l == r) {
            T[rt].maxx = 0;
            return;
        }
        int mid = (l + r) >> 1;
        build(rt << 1, l, mid), build(rt << 1 | 1, mid + 1, r);
        push_up(rt);
    }

    void update(int rt, int L, int R, int v) {
        if (L <= T[rt].l && T[rt].r <= R) {
            T[rt].maxx += v;
            lazy[rt] += v;
            return;
        }
        push_down(rt);
        int mid = (T[rt].l + T[rt].r) >> 1;
        if (L <= mid) update(rt << 1, L, R, v);
        if (R > mid) update(rt << 1 | 1, L, R, v);
        push_up(rt);
    }
} tree;

int main() {
    int T; scanf("%d", &T);
    while (T--) {
        int n, w, h; scanf("%d%d%d", &n, &w, &h);
        init();
        for (int i = 1; i <= n; i++) {
            int x, y, l; scanf("%d%d%d", &x, &y, &l);
            line[i] = LINE(x, y, y + h - 1, l);
            line[i+n] = LINE(x + w - 1, y, y + h - 1, -l);
            push(y + h - 1);
            push(y);
        }
        n <<= 1;
        build();
        sort(line + 1, line + 1 + n, [&](const LINE &ta, const LINE &tb
            ) {
            if (ta.x != tb.x) return ta.x < tb.x;
            return ta.l > tb.l;
        });
        tree.build(1, 1, blen);
        int res = 0;
        for (int i = 1; i <= n; i++) {
            int l = lower_bound(b+1, b+1+blen, line[i].y1) - b;
            int r = lower_bound(b+1, b+1+blen, line[i].y2) - b;
            tree.update(1, l, r, line[i].l);
            res = max(res, tree.T[1].maxx);
        }
        printf("%d\n", res);
    }
}
```

1.6.4 旋转扫描线

含边框注意！




```
const double eps = 1e-8;
SEG tree; // 矩阵求和最值（POJ-2482）中的SEG
struct point {
   int x, y;
   bool operator < (const point& tb) const {return y < tb.y;}
}p[SIZE<<1];
int main() {
   int n, d; scanf("%d%d", &n, &d);
   double _alpha; scanf("%lf", &_alpha);
   int alpha = (int)(_alpha * 100.0 + eps);
   for (int i = 0; i < n; i++) {
      int r; double w;
      scanf("%d%lf", &r, &w);
      p[i].x = r; p[i].y = (int)(w * 100.0 + eps);
   }
   std::sort(p, p+n);
   for (int i = 0; i < n; i++) {
      p[i+n].x = p[i].x; p[i+n].y = p[i].y + 36000;
   }
   n *= 2;
   int l = 0, r = 0;
   int res = 0;
   tree.build(0, 100000, 1);
   while (r < n) {
      int cnt = 0;
      for (int i = r; i < n; i++) {
         if (p[i].y == p[r].y) {
            tree.update(std::max(1, p[i].x - d), p[i].x, 1, 1);
            cnt++;
         }
         else break;
      }
      while (l < r) {
         if (p[r].y - p[l].y > alpha) {
            tree.update(std::max(1, p[l].x - d), p[l].x, -1, 1);
            l++;
         }
         else break;
      }
      r += cnt;
      res = std::max(res, tree.T[1].val);
   }
   printf("%d\n", res);
}
```

1.7 李超线段树

1.7.1 李超上树（[SDOI2016] 游戏）$O(m \log^3 n)$

有时，*Alice* 会选择一条从 s 到 t 的路径，在这条路径上的每一个点上都添加一个数字。对于路径上的一个点 r，若 r 与 s 的距离是 dis，那么 *Alice* 在点 r 上添加的数字是 $a \times dis + b$。

有时，*Bob* 会选择一条从 s 到 t 的路径。他需要先从这条路径上选择一个点，再从那个点上

选择一个数字。

Bob 选择的数字越小越好，但大量的数字让 Bob 眼花缭乱。Bob 需要你帮他找出他能够选择的最小的数字。

```
struct Edge {
   int to, nex, w;
} e[MAXN << 1];
int head[MAXN], tol;
void addEdge(int u, int v, int w) {
   e[tol].to = v, e[tol].w = w, e[tol].nex = head[u], head[u] = tol,
       tol++;
}

int son[MAXN], dfn[MAXN], _dfn[MAXN], dfn_cnt, fa[MAXN], dep[MAXN],
    siz[MAXN], top[MAXN];
ll dis[MAXN];
int LCA(int u, int v) {
   while (top[u] != top[v]) {
      dep[top[u]] > dep[top[v]] ? u = fa[top[u]] : v = fa[top[v]];
   }
   return dep[u] > dep[v] ? v : u;
}
void dfs1(int u, int f, int deep) {
   dep[u] = deep, fa[u] = f, siz[u] = 1;
   int maxson = -1;
   for (int i = head[u]; ~i; i = e[i].nex) {
      int v = e[i].to;
      if (v == f) continue;
      dis[v] = dis[u] + e[i].w;
      dfs1(v, u, deep + 1);
      siz[u] += siz[v];
      if (siz[v] > maxson) son[u] = v, maxson = siz[v];
   }
}
void dfs2(int u, int topf) {
   dfn[u] = ++dfn_cnt, _dfn[dfn_cnt] = u;
   top[u] = topf;
   if (!son[u]) return;
   dfs2(son[u], topf);
   for (int i = head[u]; ~i; i = e[i].nex) {
      int v = e[i].to;
      if (v == fa[u] || v == son[u]) continue;
      dfs2(v, v);
   }
}

class LC { public:
   struct Line {
      int k; ll b;
   } p[MAXN];
   int cnt;

   void init() {
      cnt = 0;
      p[0].k = 0, p[0].b = inf;
   }

   void addLine(int k, ll b) {
      cnt++;
      p[cnt].k = k, p[cnt].b = b;
   }

   inline ll cal(int x, int id) {
      return (ll) p[id].k * dis[_dfn[x]] + p[id].b;
   }

   int s[MAXN<<2]; ll minn[MAXN<<2];

   inline void push_up(int rt) {
```

```
    minn[rt] = min(minn[rt], min(minn[rt << 1], minn[rt << 1 | 1]))
        ;
}
void build(int rt, int l, int r) {
    s[rt] = 0, minn[rt] = inf;
    if (l == r) return;
    int mid = (l + r) >> 1;
    build(rt << 1, l, mid), build(rt << 1 | 1, mid + 1, r);
}

void update(int rt, int L, int R, int be, int en, int u) {
    int mid = (be + en) >> 1;
    if (L <= be && en <= R) {
        int v = s[rt];
        if (cal(be, u) <= cal(be, v) && cal(en, u) <= cal(en, v)) {
            s[rt] = u, minn[rt] = min(minn[rt], min(cal(be, u), cal(
                en, u)));
            return;
        }
        if (cal(be, u) >= cal(be, v) && cal(en, u) >= cal(en, v))
            return;
        if (p[u].k < p[v].k) {
            if (cal(mid, u) <= cal(mid, v)) {
                s[rt] = u;
                update(rt << 1, L, R, be, mid, v);
            } else update(rt << 1 | 1, L, R, mid + 1, en, u);
        } else {
            if (cal(mid, u) <= cal(mid, v)) {
                s[rt] = u;
                update(rt << 1 | 1, L, R, mid + 1, en, v);
            } else update(rt << 1, L, R, be, mid, u);
        }
        minn[rt] = min(minn[rt], min(cal(be, u), cal(en, u)));
        push_up(rt);
        return;
    }
    if (L <= mid) update(rt << 1, L, R, be, mid, u);
    if (R > mid) update(rt << 1 | 1, L, R, mid + 1, en, u);
    push_up(rt);
}

ll query(int rt, int L, int R, int be, int en) {
    if (L <= be && en <= R) return minn[rt];
    int mid = (be + en) >> 1;
    ll ans = inf;
    if (p[s[rt]].b != inf) ans = min(cal(max(L, be), s[rt]), cal(
        min(R, en), s[rt]));
    if (L <= mid) ans = min(ans, query(rt << 1, L, R, be, mid));
    if (R > mid) ans = min(ans, query(rt << 1 | 1, L, R, mid + 1,
        en));
    return ans;
}
} tree;

int main() {
    int n, m; scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) head[i] = -1; // init graph
    for (int i = 2; i <= n; i++) {
        int u, v, w; scanf("%d%d%d", &u, &v, &w);
        addEdge(u, v, w), addEdge(v, u, w);
    }
    dfs1(1, 1, 1); dfs2(1, 1);
    tree.init();
    tree.build(1, 1, n);

    while (m--) {
        int opt;
        scanf("%d", &opt);
```

```
    if (opt == 1) {
        int s, t, a, b; scanf("%d%d%d%d", &s, &t, &a, &b);
        int lca = LCA(s, t);
        auto update = [&](int u, int v, int a, int b) {
            while (top[u] != top[v]) {
                tree.update(1, dfn[top[u]], dfn[u], 1, n, tree.cnt);
                u = fa[top[u]];
            }
            tree.update(1, dfn[v], dfn[u], 1, n, tree.cnt);
        };
        tree.addLine(-a, dis[s] * a + b);
        update(s, lca, a, b);
        tree.addLine(a, (dis[s] - (dis[lca] << 1)) * a + b);
        update(t, lca, a, b);
    } else {
        int s, t; scanf("%d%d", &s, &t);
        auto query = [&](int u, int v) {
            ll ans = inf;
            while (top[u] != top[v]) {
                if (dep[top[u]] < dep[top[v]]) swap(u, v);
                ans = min(ans, tree.query(1, dfn[top[u]], dfn[u], 1, n
                    ));
                u = fa[top[u]];
            }
            if (dep[u] > dep[v]) swap(u, v);
            ans = min(ans, tree.query(1, dfn[u], dfn[v], 1, n));
            return ans;
        };
        printf("%lld\n", query(s, t));
    }
  }
}
```

2 杂项

2.1 数列归纳

$f(n) = \frac{(2*n+1)!}{(n+1)}$

1, 3, 40, 1260, 72576,

6652800, 889574400, 163459296000, 39520825344000, 12164510040883200,

4644631106519040000, 2154334728240414720000, 1193170003333152768000000,

777776389315596582912000000

$f(n) = \sum_{i=1}^{n} p, p \in prime$

转 $min_2$5筛

0, 2, 5, 10, 17, 28, 41, 58, 77, 100,

129, 160, 197, 238, 281, 328, 381, 440, 501, 568,

639, 712, 791, 874, 963, 1060, 1161, 1264, 1371, 1480,

1593, 1720, 1851, 1988, 2127, 2276, 2427, 2584, 2747, 2914,

3087, 3266, 3447, 3638, 3831, 4028, 4227, 4438, 4661, 4888

2.2 全 1 矩阵个数（51nod1291）

input	output
3 3	
011	6 3 0
110	3 1 0
100	1 0 0

```
char ss[N];
int h[N], sta[N], top, ans[N][N];
int main() {
    int n, m; scanf("%d%d", &n, &m);
    for (int u = 1; u <= n; u++) {
        scanf("%s", ss + 1);
        top = 0;
```

```
    for (int i = 1; i <= m + 1; i++) {
        h[i] = ss[i] == '1' ? h[i] + 1 : 0;
        while (top > 0 && h[sta[top]] > h[i]) {
            ans[max(h[sta[top - 1]], h[i]) + 1][i - sta[top - 1] -
                1]++;
            ans[h[sta[top]] + 1][i - sta[top - 1] - 1]--;
            top--;
        }
        while (top > 0 && h[sta[top]] == h[i]) top--;
        sta[++top] = i;
    }
}
for (int u = 1; u <= n; u++)
    for (int i = 1; i <= m; i++)
        ans[u][i] = ans[u][i] + ans[u - 1][i];
for (int u = 1; u <= n; u++) {
    int sum = 0, lalal = 0;
    for (int i = 2; i <= m; i++) {
        lalal = lalal + ans[u][i];
        sum = sum + ans[u][i] * i;
    }
    for (int i = 1; i <= m; i++) {
        ans[u][i] = ans[u][i] + sum;
        sum = sum - lalal - ans[u][i + 1];
        lalal = lalal - ans[u][i + 1];
    }
    // 原题解疑惑代码
    // for (int i=1;i<=m;i++) {
    // for (int j=i+1;j<=m;j++)
    // ans[u][i]=ans[u][i]+ans[u][j]*(j-i+1);
    // }
}
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= m; j++) {
        printf("%d ", ans[i][j]); // i * j的全1矩阵个数
    }
    printf("\n");
}
printf("%lld\n", res);
return 0;
}
```

3 习题整理

3.1 可重边集的点能否和当前询问边构成三角形（20 牛客 2H）（动态点开线段树）

input	output
8	
1 1	
3 1	No
1 1	
3 2	No
3 1	Yes
1 2	
2 1	
3 1	No

```
map<int, int> ma;
SEG tree; // 动态点开线段树维护最小值，记得初始化val[0] = inf;
int root;
void add(int x) {
    ma[x]++;
    if (ma[x] == 1) {
        auto it = ma.lower_bound(x); auto R = it;
        if (++R != ma.end() && R-> second == 1) root = tree.update(root
            , R->first, R->first- x, 1, MAXR);
```

```
        if (it != ma.begin()) root = tree.update(root, x, x - (--it)->
            first, 1, MAXR);
        else root = tree.update(root, x, inf, 1, MAXR);
    } else if (ma[x] == 2) root = tree.update(root, x, 0, 1, MAXR);
}
void del(int x) {
    auto it = ma.lower_bound(x);
    ma[x]--;
    int L = -inf;
    if (it != ma.begin()) {
        L = (--it) -> first;
        ++it;
    }
    if (ma[x] == 0) {
        if ((++it) != ma.begin() && it->second == 1)
            root = tree.update(root, it->first, it->first-L, 1, MAXR);
        root = tree.update(root, x, inf, 1, MAXR);
        ma.erase(x);
    } else if (ma[x] == 1) root = tree.update(root, x, x-L, 1, MAXR);
}
int ask(int x) {
    auto it = ma.lower_bound(x/2+1);
    if (it == ma.end()) return inf;
    if (it->second > 1) return it->first;
    if (it != ma.begin()) {
        auto L = it; --L;
        if (L ->first + it ->first > x) return it->first;
    }
    if ((++it) != ma.begin()) return it->first;
    return inf;
}

int main() {
    tree.init();
    int q; scanf("%d", &q);
    while (q--) {
        int opt, x; scanf("%d%d", &opt, &x);
        if (opt == 1) add(x);
        else if (opt == 2) del(x);
        else {
            if (tree.query_min(root, ask(x), MAXR, 1, MAXR) < x) printf("
                Yes\n");
            else printf("No\n");
        }
    }
}
```

3.2 左偏树离线处理查询成立最多数（HDU5575）

0 代表没有水，1 代表有水

input	output
3 4 3 4 1 3 1 2 1 0 2 2 0 3 3 1	3

```
struct Query {
    int x, y;
    Query() {}
    Query(int _x, int _y) { x = _x, y = _y; }
};
int fa[MAXN];
int find(int x) {
    if (x == fa[x]) return x;
    else return fa[x] = find(fa[x]);
```

```
}
vector<Query> q;
int LH[MAXN], RH[MAXN], L[MAXN], R[MAXN];
int root[MAXN], siz[MAXN], edge[MAXN];
void join(int x, int y) {
   int fx = find(x), fy = find(y);
   if (fx == fy) return;
   fa[fy] = fx;
   if (fx < fy) RH[fx] = RH[fy], L[R[fx]] = fx, R[fx] = R[fy];
   else LH[fx] = LH[fy], R[L[fx]] = fx, L[fx] = L[fy];
   root[fx] = tree.merge(root[fx], root[fy]);
   edge[fx] += edge[fy];
   siz[fx] += siz[fy];
}
int main() {
   int T; scanf("%d", &T);
   while (T--) {
      int n, m; scanf("%d%d", &n, &m);
      LH[1] = RH[n] = inf;
      L[n] = n-1;
      for (int i = 1; i < n; i++) {
         scanf("%d", &RH[i]);
         LH[i + 1] = RH[i];
         L[i] = i - 1, R[i] = i + 1;
      }
      tree.init(); q.clear();
      for (int i = 1; i <= n; i++) root[i] = 0; // init LT's root
      int res = 0;
      for (int i = 1; i <= m; i++) {
         int x, y, z; scanf("%d%d%d", &x, &y, &z);
         if (z == 1) {
            q.pb(Query(x, y + 1));
         } else {
            if (root[x]) {
               root[x] = tree.insert(root[x], y);
            } else root[x] = tree.Newnode(y);
            res++;
         }
      }
      sort(q.begin(), q.end(), [&](const Query &ta, const Query &tb)
          {
         if (ta.y != tb.y) return ta.y < tb.y;
         else return ta.x < tb.x;
      });
      for (int i = 1; i <= n; i++) fa[i] = i;
      for (int i = 1; i <= n; i++) siz[i] = edge[i] = 0;
      for (auto &e : q) {
         int x = find(e.x), y = e.y;
         // 向左溢出
         while (y > LH[x]) join(x, L[x]), x = find(x);
         // 向右溢出
         while (y > RH[x]) join(x, R[x]), x = find(x);
         // 删除水位以下的X
         while (!tree.isempty(root[x]) && tree.top(root[x]) < y) {
            root[x] = tree.pop(root[x]);
            siz[x]++;
         }
         // update result
         if (++edge[x] >= siz[x]) {
            res += (edge[x] - siz[x]);
            siz[x] = edge[x] = 0;
         }
      }
      printf("Case #%d: %d\n", kass++, res);
   }
}
```