0.1 可修改的最小生成树

给 $n \leq 2 * 10^5$ 个点和 $m \leq 2 * 10^5$ 条双向边，每条边的权值为 w_i 和单位减少代价为 c_i，边权可以减少为负，在不超过代价 S 的条件下求出最小生成树的代价和，以及选择的边的编号和修改后的边权。

先把边按照边权排序，求最小生成树。当最小生成树的所有边确定了以后，那么减少哪一条边的边权也就确定了：因为边权可以为负，那么最优一定是减少 c 最小的边。

然后枚举不在最小生成树中的边，对于一开始求好的最小生成树，最多只会改变一条边。对于边 i 如果要把它作为最终的最小生成树中的边，那么把这条边添加进求好的最小生成树会形成一个环，它要替换的一定是环上边权最大的边，那么问题就变成了：求树上两点路径上最大边权。利用 LCA 和倍增解决。

时间复杂度：$O(m \log m + mlogn + n \log n)$

```
typedef long long ll;
const int MAX_N = 200010;
const int MAX_M = 200010;

int n, m, S, MinC, MinId;
int fa[MAX_N], vis[MAX_M], depth[MAX_N], anc[MAX_N][20];

struct Edge {
    int u, v, w, c, id;

    bool operator < (const Edge& rhs) const {
        return w < rhs.w;
    }
} ;

Edge edge[MAX_M], dp[MAX_N][20];
vector<Edge> g[MAX_N];

bool cmp_id(Edge a, Edge b) {
    return a.id < b.id;
}

inline int find(int x) {
    return fa[x] == x ? x : fa[x] = find(fa[x]);
}

ll Kruskal() {
    MinC = (int)(1e9) + 10;
    memset(vis, 0, sizeof (vis));
    for (int i = 0; i <= n; ++i) fa[i] = i, g[i].clear();

    sort(edge + 1, edge + m + 1);
    ll ret = 0;
    for (int i = 1; i <= m; ++i) {
        int u = edge[i].u, v = edge[i].v;
        int fu = find(u), fv = find(v);
        if (fu != fv) {
            fa[fu] = fv;
            ret += edge[i].w;
            if (MinC > edge[i].c) {
                MinC = edge[i].c;
                MinId = edge[i].id;
            }

            vis[edge[i].id] = 1;
            Edge tmp = edge[i];
            g[u].push_back(tmp);
            swap(tmp.u, tmp.v);
            g[v].push_back(tmp);
        }
```

```
    }
    return ret;
}

void dfs(int u, int p, int d) {
    depth[u] = d;
    if (u != 1) dp[u][0] = edge[p];
    for (int i = 1; i < 20; ++i) {
        anc[u][i] = anc[anc[u][i - 1]][i - 1];
        dp[u][i] = max(dp[u][i - 1], dp[anc[u][i - 1]][i - 1]);
    }
    for (int i = 0; i < g[u].size(); ++i) {
        if (g[u][i].id == p) continue;
        anc[g[u][i].v][0] = u;
        dfs(g[u][i].v, g[u][i].id, d + 1);
    }
}

Edge LCA(int u, int v) {
    if (depth[u] > depth[v]) swap(u, v);
    Edge ret;
    ret.w = -1;
    for (int i = 0; i < 20; ++i) {
        if (((depth[v] - depth[u]) >> i) & 1) {
            ret = max(ret, dp[v][i]);
            v = anc[v][i];
        }
    }
    if (u == v) return ret;
    for (int i = 19; i >= 0; --i) {
        if (anc[u][i] != anc[v][i]) {
            ret = max(ret, dp[u][i]);
            ret = max(ret, dp[v][i]);
            u = anc[u][i], v = anc[v][i];
        }
    }
    ret = max(ret, dp[u][0]);
    ret = max(ret, dp[v][0]);
    return ret;
}

void solve() {
    ll sum = Kruskal();
    ll ans = sum - S / MinC;
    sort(edge + 1, edge + m + 1, cmp_id);
    anc[1][0] = 1;
    dfs(1, -1, 0);
    int a = -1, b = MinId;
    for (int i = 1; i <= m; ++i) {
        if (vis[edge[i].id]) continue;
        Edge ee = LCA(edge[i].u, edge[i].v);
        ll tmp = sum - ee.w + (edge[i].w - S / edge[i].c);
        if (tmp < ans) {
            ans = tmp;
            a = ee.id, b = i;
        }
    }
    printf("%lld\n", ans);
    for (int i = 1; i <= m; ++i) {
        if (!vis[i] || i == a || i == b) continue;
        printf("%d %d\n", i, edge[i].w);
    }
    printf("%d %d\n", b, edge[b].w - S / edge[b].c);
}

```

```
int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= m; ++i) scanf("%d", &edge[i].w);
    for (int i = 1; i <= m; ++i) scanf("%d", &edge[i].c);
    for (int i = 1; i <= m; ++i) {
        scanf("%d%d", &edge[i].u, &edge[i].v);
        edge[i].id = i;
    }
    scanf("%d", &S);
    solve();
    return 0;
}
```

0.2 求曼哈顿距离最小生成树

对每个点进行 45 度角分割成八个区域，每个区域只会选择一个点建边。根据对称性，对每个点只要找右半边的四个点即可。不妨设找 y 轴右边 45 度角区域，对于点 (x_0, y_0) 应该要找满足 $x_1 \geq x_0$ 且 $y_1 - x_1 \geq y_0 - x_0$ 的 $x_1 + y_1$ 最小的点 (x_1, y_1)。

因为此时曼哈顿距离为:$(x_1 - x_0) + (y_1 - y_0)$。先把点坐标按照 x 排序，将 $y - x$ 离散化，借助树状数组查询和更新区间最小值。这样子最多只会建 $4 * n$ 条边，再用 $Kruskal$ 跑最小生成树就可以 $O(n \log n)$ 解决了。

```
const int MAX_N = 100010;
const int inf = 0x3f3f3f3f;

int n, edge_num, cases = 0;
int store[MAX_N], fa[MAX_N];

struct Point {
    int x, y, diff, id;
    bool operator < (const Point& rhs) const {
        return x == rhs.x ? y < rhs.y : x < rhs.x;
    }
} P[MAX_N];

struct Bit {
    int value[MAX_N], id[MAX_N];

    void init() {
        memset(value, 0x3f, sizeof(value));
        memset(id, -1, sizeof(id));
    }
    int lowbit(int x) {
        return x & -x;
    }
    void update(int x, int y, int z) {
        for (int i = x; i > 0; i -= lowbit(i)) {
            if (y < value[i]) value[i] = y, id[i] = z;
        }
    }
    pair<int, int> query(int x) {
        int ret = inf, t = -1;
        for (int i = x; i <= n; i += lowbit(i)) {
            if (value[i] < ret) ret = value[i], t = id[i];
        }
        return make_pair(ret, t);
    }
} bit;

struct Edge {
    int u, v, w;
    Edge() {}
```

```
41      Edge(int _u, int _v, int _w): u(_u), v(_v), w(_w) {}
42
43      bool operator < (const Edge& rhs) const {
44          return w < rhs.w;
45      }
46  } edge[MAX_N * 10];
47
48  inline int find(int x) {
49      return fa[x] == x ? x : fa[x] = find(fa[x]);
50  }
51
52  ll solve() {
53      edge_num = 0;
54      for (int i = 0; i <= n; ++i) fa[i] = i;
55
56      for (int dir = 1; dir <= 4; ++dir) {
57          if (dir == 2 || dir == 4) {
58              for (int i = 1; i <= n; ++i) swap(P[i].x, P[i].y);
59          } else if (dir == 3) {
60              for (int i = 1; i <= n; ++i) P[i].x = -P[i].x;
61          }
62          sort(P + 1, P + n + 1); // sorted by x
63          for (int i = 1; i <= n; ++i) {
64              store[i - 1] = P[i].diff = P[i].y - P[i].x;
65          }
66          sort(store, store + n);
67          int tot = unique(store, store + n) - store;
68          bit.init();
69          for (int i = n; i >= 1; --i) {
70              int pos = lower_bound(store, store + tot, P[i].diff) - store + 1;
71              pair<int, int> ret = bit.query(pos);
72              int a = ret.first, b = ret.second;
73              if (b != -1) {
74                  edge[edge_num++] = Edge(P[i].id, b, a - (P[i].x + P[i].y));
75              }
76              bit.update(pos, P[i].x + P[i].y, P[i].id);
77          }
78      }
79
80      sort(edge, edge + edge_num);
81      ll ans = 0;
82      for (int i = 0; i < edge_num; ++i) {
83          int u = edge[i].u, v = edge[i].v, w = edge[i].w;
84          int fu = find(u), fv = find(v);
85          if (fu != fv) {
86              ans += w;
87              fa[fu] = fv;
88          }
89      }
90      return ans;
91  }
92
93  int main() {
94      while (~scanf("%d", &n) && n) {
95          for (int i = 1; i <= n; ++i) {
96              scanf("%d%d", &P[i].x, &P[i].y);
97              P[i].id = i;
98          }
99          printf("Case %d: Total Weight = %lld\n", ++cases, solve());
100     }
101     return 0;
102 }
```

0.3 其他

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <algorithm>
#include <iostream>
using namespace std;
typedef long long ll;
const int MAX_N = 50010;

int n, m;
int data[MAX_N], cnt[MAX_N], pos[MAX_N];

struct Query {
    int L, R, id;
    ll nume, deno;

    bool operator < (const Query& rhs) const {
        return pos[L] < pos[rhs.L] || (pos[L] == pos[rhs.L] && R < rhs.R);
    }
    void relax() {
        ll g = __gcd(nume, deno);
        nume /= g, deno /= g;
    }
} Q[MAX_N];

bool cmp_id(const Query& a, const Query& b) {
    return a.id < b.id;
}

inline void update(int value, ll& ans, int add) {
    ans += 2 * add * cnt[value] + 1;
    cnt[value] += add;
}

void solve() {
    memset(cnt, 0, sizeof(cnt));
    int d = (int)sqrt(n + 0.5);
    for (int i = 1; i <= n; ++i) pos[i] = (i - 1) / d + 1;
    sort(Q + 1, Q + m + 1);
    int L = 1, R = 0;
    ll ans = 0;
    for (int i = 1; i <= m; ++i) {
        while (L < Q[i].L) {
            update(data[L++], ans, -1);
        }
        if (L > Q[i].L) {
            while (--L >= Q[i].L) update(data[L], ans, 1);
            ++L;
        }
        if (R < Q[i].R) {
            while (++R <= Q[i].R) update(data[R], ans, 1);
            R--;
        }
        while (R > Q[i].R) {
            update(data[R--], ans, -1);
        }
        int len = Q[i].R - Q[i].L + 1;
        Q[i].nume = ans - len;
        Q[i].deno = 1ll * len * (len - 1);
    }
    sort(Q + 1, Q + m + 1, cmp_id);
    for (int i = 1; i <= m; ++i) {
        Q[i].relax();
```

```
        printf("%lld/%lld\n", Q[i].nume, Q[i].deno);
    }
}

int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; ++i) scanf("%d", &data[i]);
    for (int i = 1; i <= m; ++i) {
        scanf("%d%d", &Q[i].L, &Q[i].R);
        Q[i].id = i;
    }
    solve();
    return 0;
}
```

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <algorithm>
#include <iostream>
using namespace std;
typedef long long ll;
const int MAX_N = 100010;

int n, m;
int data[MAX_N], store[MAX_N], real[MAX_N], pos[MAX_N], cnt[MAX_N];

struct Query {
    int L, R, id;
    ll ans;

    bool operator < (const Query& rhs) const {
        return pos[L] < pos[rhs.L] || (pos[L] == pos[rhs.L] && R < rhs.R);
    }
} Q[MAX_N];

inline void update(int value, ll& ans, int add) {
    ans += 3ll * add * cnt[value] * cnt[value] + 3 * cnt[value] + add;
    cnt[value] += add;
}

bool cmp_id(const Query& a, const Query& b) {
    return a.id < b.id;
}

void solve() {
    memset(cnt, 0, sizeof(cnt));
    sort(Q + 1, Q + m + 1);
    int L = 1, R = 0;
    ll ans = 0;
    for (int i = 1; i <= m; ++i) {
        while (L < Q[i].L) {
            update(real[L++], ans, -1);
        }
        if (L > Q[i].L) {
            while (--L >= Q[i].L) update(real[L], ans, 1);
            ++L;
        }
        if (R < Q[i].R) {
            while (++R <= Q[i].R) update(real[R], ans, 1);
            --R;
        }
        while (R > Q[i].R) {
            update(real[R--], ans, -1);
        }
```

```
51          Q[i].ans = ans;
52      }
53      sort(Q + 1, Q + m + 1, cmp_id);
54      for (int i = 1; i <= m; ++i) {
55          printf("%I64d\n", Q[i].ans);
56      }
57  }
58
59  int main() {
60      while (~scanf("%d", &n)) {
61          for (int i = 1; i <= n; ++i) {
62              scanf("%d", &data[i]);
63              store[i - 1] = data[i];
64          }
65          sort(store, store + n);
66          int d = (int)sqrt(n + 0.5);
67          int tot = unique(store, store + n) - store;
68          for (int i = 1; i <= n; ++i) {
69              real[i] = lower_bound(store, store + tot, data[i]) - store + 1;
70              pos[i] = (i - 1) / d + 1;
71          }
72          scanf("%d", &m);
73          for (int i = 1; i <= m; ++i) {
74              scanf("%d%d", &Q[i].L, &Q[i].R);
75              Q[i].id = i;
76          }
77          solve();
78      }
79      return 0;
80  }
```