C5-G

岳宗翰

题目分析：

- 平面上存在n个点$\{\mathrm{P}_1, \mathrm{P}_2, \ldots \mathrm{P}_n\}$，求两点间最小曼哈顿距离$\min_{1\le i,j\le n}\{|x_{P_i}-x_{P_j}|+|y_{P_i}-y_{P_j}|\}$
- 数据范围: $2 \le \mathrm{n} \le 1e5,\ 0 \le |x|,|y| \le 1e9$

根据数据范围，可以知道我们不能通过暴力n^2求解，而需要一个O(nlogn)复杂度的算法来解决这个问题。在考虑如何降低时间复杂度之前，先思考一下如何简化这个求曼哈顿距离的问题:

- 首先，分类讨论以去掉距离中的绝对值号
- $x_{P_i} > x_{P_j}, y_{P_i} > y_{P_j}$时，$d(\mathrm{P}_i,\mathrm{P}_j) = x_{P_i} - x_{P_j} + y_{P_i} - y_{P_j} = (x_{P_i}+y_{P_i}) - (x_{P_j}+y_{P_j})$
- $x_{P_i} > x_{P_j}, y_{P_i} < y_{P_j}$时，$d(\mathrm{P}_i,\mathrm{P}_j) = x_{P_i} - x_{P_j} + y_{P_j} - y_{P_i} = (x_{P_i}-y_{P_i}) - (x_{P_j}-y_{P_j})$
- $x_{P_i} < x_{P_j}, y_{P_i} > y_{P_j}$时，$d(\mathrm{P}_i,\mathrm{P}_j) = x_{P_j} - x_{P_i} + y_{P_i} - y_{P_j} = (-x_{P_i}+y_{P_i}) - (-x_{P_j}+y_{P_j})$
- $x_{P_i} < x_{P_j}, y_{P_i} < y_{P_j}$时，$d(\mathrm{P}_i,\mathrm{P}_j) = x_{P_j} - x_{P_i} + y_{P_j} - y_{P_i} = (-x_{P_i}-y_{P_i}) - (-x_{P_j}-y_{P_j})$

通过对所有点的x坐标从小到大排序，保证$i > j$时$x_{P_i} > x_{P_j}$，可以简化为两种情况；

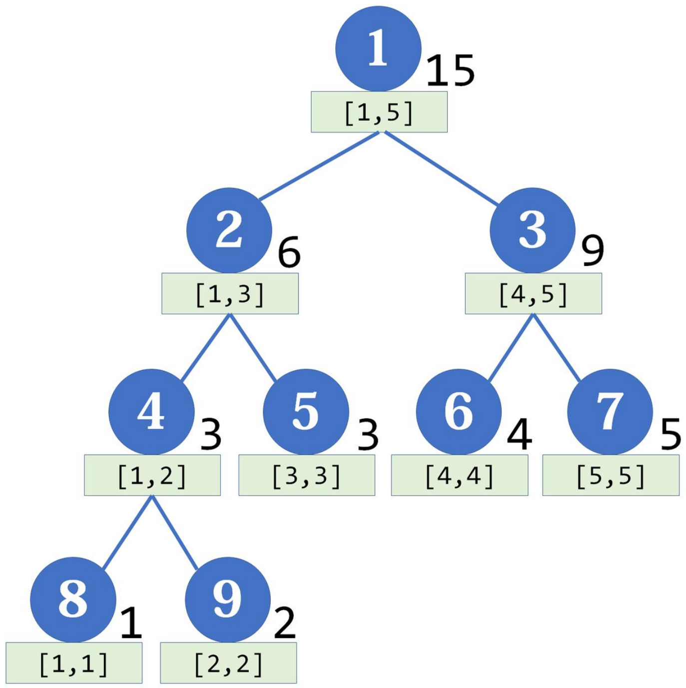
- 遍历排序后的点集，将$x+y$和$x-y$作为点的某种衡量标准，将其分别存储在某种数据结构中；
- 对于当前的点P_i，分别求出前$i-1$个点中所有满足$y_{P_i} > y_{P_j}$的点$\{P_j\}$中具有的最大值$\max(x_{P_j}+y_{P_j})$和所有满足$y_{P_\mathrm{k}} > y_{P_i}$的点$\{P_k\}$中具有的最大值$\max(x_{P_\mathrm{k}}-y_{P_k})$；
- 则该点与前$i-1$个点的最小曼哈顿距离为:

$$\min((x_{P_i}+y_{P_i}) - (x_{P_j}+y_{P_j}), (x_{P_i}-y_{P_i}) - (x_{P_\mathrm{k}}-y_{P_k}))$$

如何实现在O(logn)的时间内查询上述区间中的最大值$\max(x_{P_j}+y_{P_j})$和$\max(x_{P_\mathrm{k}}-y_{P_k})$?

线段树Segment Tree：

　　如图所示，将一个数组分为左右两个区段，再对两个子节点进行二分直到区间长度为1，每个节点可以记录其区间范围和某些特征；对于每个节点的编号和堆类似，构成一颗完全二叉树；
　　例如，可以在每个节点上记录其所对应区间的和，则父节点的值等于两个子节点值的和；同理也可以用于维护区间最大值；
　　和树状数组类似，线段树支持在O(logn)的时间内进行修改和查询操作，但和树状数组相比，线段树还支持区间修改，功能更加全面(树状数组能做的线段树都能做，但线段树能做的更多，虽然写起来也麻烦的多)；



实现线段树的函数：
- update() 通过子节点对父节点进行更新；
- change() 对指定区间进行修改；
- query() 查询指定区间的某一特征；
- pushdown() 在查询区间时，将父节点的更新传递到子节点上

接着分析题目，我们只需要维护两颗线段树，在将点P_i的两个特征值x_{P_i}+y_{P_i} 和x_{P_i}-y_{P_i}分别插入其中时保证其所在叶子节点位置与y_{P_i}在所有点y值的递增排序中的位置相同，即可直接通过其编号对线段树进行查询，从而得到我们想要的$\max(x_{P_j}+y_{P_j})$和$\max(x_{P_\mathrm{k}}-y_{P_k})$；

例如，当前遍历到第四个点P_4，其$\mathrm{x}+y$和$x-y$的值分别为10和4，在所有点的y值中为第9大的数，则用10和4分别去更新两颗线段树的第9个叶子节点，此时路径上的父节点也会得到更新；

要获得$\max(x_{P_j}+y_{P_j})$，只需要调用第一颗线段树的query(1,ID(P_i))，查询所有y值比y_{P_i}小的点中特征值$x+y$的最大值；

同理，调用第二颗线段树的query(ID(P_i)+1,n)，查询$\max(x_{P_\mathrm{k}}-y_{P_k})$；

最后可以获得该点P_i与前$i-1$个点的最小曼哈顿距离，并使用该点的$x+y$和$x-y$两个特征值分别对两颗线段树进行更新；

在遍历完所有点后即可获得答案。

多提一句，要实现对所有点的y值进行编号，可以在对y值排序后用map这一数据结构进行存储，和java里的map类似但使用更方便。例如使用名称为“ID”的map存储，要获得y值为20的点对应编号，直接使用ID[20]即可

```cpp
#include<vector>
#include<iostream>
#include<stdio.h>
#include<algorithm>
#include<map>
#include<numeric>

using namespace std;

const long long inf = 1e18;
const int N = 4e5 + 5;

vector<long long> seg1(N), tag1(N);
vector<long long> seg2(N), tag2(N);

void pushdown(int l, int r, int id, vector<long long> &seg,
              vector<long long> &tag) {
    int m = l + (r - l) / 2;
    if (l != r && tag[id]) {
        tag[2 * id] = max(tag[2 * id], tag[id]);
        tag[2 * id + 1] = max(tag[2 * id + 1], tag[id]);
        seg[2 * id] = max(seg[2 * id], tag[id]);
        seg[2 * id + 1] = max(seg[2 * id + 1], tag[id]);
        tag[id] = -inf;
    }
}

void modify(int ll, int rr, int l, int r, int id, long long val,
            vector<long long> &seg, vector<long long> &tag) {
    if (ll <= l && r <= rr) {
        tag[id] = max(tag[id], val);
        seg[id] = max(tag[id], val);
        return;
    }
    int m = l + (r - l) / 2;
    pushdown(l, r, id, seg, tag);
    if (ll <= m) modify(ll, rr, l, m, 2 * id, val, seg, tag);
    if (m < rr) modify(ll, rr, m + 1, r, 2 * id + 1, val, seg, tag);
    seg[id] = max(seg[2 * id], seg[2 * id + 1]);
}

long long get(int ll, int rr, int l, int r, int id,
vector<long long> &seg,
            vector<long long> &tag) {
    if (ll <= l && r <= rr) return seg[id];
    int m = l + (r - l) / 2;
    long long ret = -inf;
    pushdown(l, r, id, seg, tag);
    if (ll <= m) ret = max(ret, get(ll, rr, l, m, 2 *
id, seg, tag));
    if (m < rr) ret = max(ret, get(ll, rr, m + 1, r,
2 * id + 1, seg, tag));
    return ret;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    int tt;
    cin >> tt;
    while (tt--) {
        int n;
        cin >> n;
        for (int i = 0; i <= 4 * n + 1; i++) {
            seg1[i] = tag1[i] = -inf;
            seg2[i] = tag2[i] = -inf;
        }
        vector<pair<int, int>> p(n);
        vector<int> py(n);
        for (int i = 0; i < n; i++) {
            cin >> p[i].first >> p[i].second;
            py[i] = p[i].second;
        }
sort(py.begin(), py.end());
        map<int, int> id;
        for (int i = 0; i < n; i++) {
            id[py[i]] = i + 1;
        }
        vector<int> order(n);
        iota(order.begin(), order.end(), 0);
        sort(order.begin(), order.end(), [&](int u, int v) {
            return p[u] < p[v];
        });
        long long ans = inf;
        for (int i : order) {
            long long x = p[i].first, y = p[i].second;
            int od = id[y];
            long long d = get(1, od, 1, n, 1, seg1, tag1);
            ans = min(ans, x + y - d);
            d = od + 1 <= n ? get(od + 1, n, 1, n, 1, seg2, tag2) : -inf;
            ans = min(ans, x - y - d);
            modify(od, od, 1, n, 1, x + y, seg1, tag1);
            modify(od, od, 1, n, 1, x - y, seg2, tag2);
        }
        cout << ans << '\n';
    }
    return 0;
}
```

最后附上标程代码