NOIP 十连测 08

A. 签到

签到题。

差分算出每个间隔被经过的次数，按从大到小顺序选即可。

时间复杂度 $\mathcal{O}(n \log n)$。

```cpp
#include <bits/stdc++.h>

using i64 = long long;

constexpr int N = 1e6 + 5;

int n, k;
int p[N], pos[N], d[N];

int main() {
    freopen("ball.in", "r", stdin);
    freopen("ball.out", "w", stdout);

    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

    std::cin >> n >> k;
    for (int i = 1; i <= n; i++) std::cin >> p[i], pos[p[i]] = i;
    for (int i = 2; i <= n; i++) {
        int l = pos[i - 1], r = pos[i];
        if (l > r) std::swap(l, r);
        d[l]++; d[r]--;
    }
    for (int i = 1; i <= n; i++) d[i] += d[i - 1];
    std::sort(d + 1, d + n, std::greater<int>());
    for (int i = 1, sum = 0; i < n; i++) {
        sum += d[i];
        if (sum >= k) return std::cout << i << "\n", 0;
    }
    std::cout << "-1" << "\n";
    return 0;
}
```

B. 计数

考虑钦定 a_1 画圆弧的方式，枚举这三种情况，问题就可以转变成链上的问题。

设新得到的序列为 b，长度为 m，我们要做的就是计算 b 的合法方案数。

考虑设 f_i 表示当前到了第 i 个元素，最多能匹配上 f_i 对，g_i 表示匹配的最多时的方案数。

转移枚举和它相同的上一个位置 p，然后 $f_i \leftarrow \max\limits_{j=1}^{p-1} f_j + 1$ 即可，如果最大值相同则方案数要相加。

最后判断 f_m 是否等于 $n-1$ 即可（因为 a_1 已经被我们匹配过了），若成立则方案数加上 g_m。

时间复杂度 $\mathcal{O}(n)$。

```cpp
#include <bits/stdc++.h>

using i64 = long long;

constexpr int N = 1e6 + 5, P = 1e9 + 7;

int n;
int a[N * 3], b[N * 3], prv[N * 3], buc[N];
std::pair<int, int> f[N * 3];
std::vector<int> pos[N];

std::pair<int, int> merge(std::pair<int, int> x, std::pair<int, int> y) {
    if (x.first == y.first) return {x.first, (x.second + y.second) % P};
    return std::max(x, y);
}

int solve(int m) {
    memset(buc, 0, sizeof(buc));
    for (int i = 1; i <= m; i++) {
        f[i] = {0, 0};
        prv[i] = buc[b[i]];
        buc[b[i]] = i;
    }

    f[0] = {0, 1};
    for (int i = 1; i <= m; i++) {
        int p1 = prv[i];
        if (p1) f[i] = {f[p1 - 1].first + 1, f[p1 - 1].second};
        f[i] = merge(f[i], f[i - 1]);
    }
    return f[m].first == n - 1 ? f[m].second : 0;
}

int main() {
    freopen("circle.in", "r", stdin);
    freopen("circle.out", "w", stdout);

    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

    std::cin >> n;
    for (int i = 1; i <= 3 * n; i++) std::cin >> a[i], pos[a[i]].push_back(i);

    int ans = 0;
    for (int i1 = 0; i1 < 3; i1++) for (int i2 = i1 + 1; i2 < 3; i2++) {
        int tot = 0;
        if (i1 == 0 && i2 == 1) {
            int l = pos[a[1]][1] + 1;
            for (int i = l; i <= 3 * n; i++) b[++tot] = a[i];
        }
        if (i1 == 0 && i2 == 2) {
            int l = pos[a[1]][0] + 1, r = pos[a[1]][2] - 1;
            for (int i = l; i <= r; i++) b[++tot] = a[i];
```

```
        }
        if (i1 == 1 && i2 == 2) {
            int l = pos[a[1]][2] + 1, r = pos[a[1]][1] - 1;
            for (int i = l; i <= 3 * n; i++) b[++tot] = a[i];
            for (int i = 1; i <= r; i++) b[++tot] = a[i];
        }
        ans = (ans + solve(tot)) % P;
    }
    std::cout << ans << "\n";

    return 0;
}
```

C. 优化

考虑将一个 pair 抽象成平面上的点，将这些点按照 x 坐标排序。

我们要做的就是，将这些点分成两部分：被 x 删去和被 y 删去。

我们考虑两个点 $A(x_1,y_1),B(x_2,y_2)$，其中 $x_1<x_2,y_1>y_2$（即 A 在 B 的左上角），若点 A 是被 x 删去的，则点 B 一定也是被 x 删去的，证明显然，因为如果想让 B 被 y 删去，则先要删去 A，而 A 是被 x 删去的，所以在删 A 之前 B 一定被删掉，矛盾。

考虑 $f_{i,j}$ 表示当前考虑到 x 坐标前 i 小的点，这 i 个点中被 x 删去的点最大的 y 坐标为 j，我们考虑刷表法，若下一个点的 $y_{i+1}<j$，根据上面的结论，该点只能选择被 x 删去，我们有

$$f_{i+1,j}\leftarrow f_{i,j}+x_{i+1}$$

。否则 $y_{i+1}>j$，该点可以选择被 x 删去，也可以选择被 y 删去，所以有转移：

$$\begin{aligned}f_{i+1,j}&\leftarrow f_{i,j}+y_{i+1}\\f_{i+1,y_{i+1}}&\leftarrow \min(f_{i+1,y_{i+1}},f_{i,j}+x_{i+1})\end{aligned}$$

将所有点的 y 坐标离散化后，我们得到了一个 $\mathcal{O}(n^2)$ 的 dp 做法。

考察转移的形式，我们发现对于 $j>y_{i+1}$，所有的 $f_{i+1,j}$ 在 $f_{i,j}$ 的基础上加上了 x_{i+1}，对于 $j<y_{i+1}$，加上了 y_{i+1}，而对于 $j=y_{i+1}$，我们将他设为了 $x_{i+1}+\min\limits_{j<y_{i+1}} f_{i,j}$，这些操作容易使用区间加区间求最小值的线段树解决，时间复杂度 $\mathcal{O}(n\log n)$。

```
#include <bits/stdc++.h>

using i64 = long long;

constexpr int N = 2e5 + 5;
constexpr i64 INF = (1ll << 60);

struct Point {
    int x, y;
    Point(int x = 0, int y = 0) : x(x), y(y) {}
} P[N];

int n;
Point a[N];
i64 f[1005][1005];

struct SegTree {
```

```
    static constexpr int N = 2e5 + 5;
    i64 tag[N << 2], min[N << 2];

    #define ls(u) (u << 1)
    #define rs(u) (u << 1 | 1)

    void build(int u, int l, int r) {
        if (l == r) return min[u] = INF, void();
        int mid = (l + r) >> 1;
        build(ls(u), l, mid); build(rs(u), mid + 1, r);
        min[u] = std::min(min[ls(u)], min[rs(u)]);
    }

    void pushTag(int u, i64 t) {
        min[u] += t;
        tag[u] += t;
    }

    void pushDown(int u) {
        if (tag[u]) {
            pushTag(ls(u), tag[u]);
            pushTag(rs(u), tag[u]);
            tag[u] = 0;
        }
    }

    void modifySet(int u, int l, int r, int pos, i64 val) {
        if (l == r) return min[u] = val, void();
        int mid = (l + r) >> 1; pushDown(u);
        if (pos <= mid) modifySet(ls(u), l, mid, pos, val);
        if (pos > mid)  modifySet(rs(u), mid + 1, r, pos, val);
        min[u] = std::min(min[ls(u)], min[rs(u)]);
    }

    void modifyAdd(int u, int l, int r, int ql, int qr, i64 val) {
        if (ql > qr) return;
        if (ql <= l && r <= qr) return pushTag(u, val);
        int mid = (l + r) >> 1; pushDown(u);
        if (ql <= mid) modifyAdd(ls(u), l, mid, ql, qr, val);
        if (qr > mid)  modifyAdd(rs(u), mid + 1, r, ql, qr, val);
        min[u] = std::min(min[ls(u)], min[rs(u)]);
    }

    i64 query(int u, int l, int r, int ql, int qr) {
        if (ql > qr) return INF;
        if (ql <= l && r <= qr) return min[u];
        int mid = (l + r) >> 1; i64 ret = INF; pushDown(u);
        if (ql <= mid) ret = std::min(ret, query(ls(u), l, mid, ql, qr));
        if (qr > mid)  ret = std::min(ret, query(rs(u), mid + 1, r, ql, qr));
        min[u] = std::min(min[ls(u)], min[rs(u)]);
        return ret;
    }

} tree;

void chmax(i64 &x, i64 y) { if (y > x) x = y; }
```

```cpp
void chmin(i64 &x, i64 y) { if (y < x) x = y; }

int main() {

    freopen("pairs.in", "r", stdin);
    freopen("pairs.out", "w", stdout);

    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

    std::cin >> n; std::vector<int> dx, dy;
    for (int i = 1; i <= n; i++) {
        std::cin >> a[i].x >> a[i].y;
        dx.push_back(a[i].x);
        dy.push_back(a[i].y);
    }

    std::sort(dx.begin(), dx.end()); dx.resize(std::unique(dx.begin(), dx.end())
- dx.begin());
    std::sort(dy.begin(), dy.end()); dy.resize(std::unique(dy.begin(), dy.end())
- dy.begin());

    for (int i = 1; i <= n; i++) {
        a[i].x = std::lower_bound(dx.begin(), dx.end(), a[i].x) - dx.begin() + 1;
        a[i].y = std::lower_bound(dy.begin(), dy.end(), a[i].y) - dy.begin() + 1;
    }
    std::sort(a + 1, a + 1 + n, [&](Point &lhs, Point &rhs) { return lhs.x <
rhs.x; });

    tree.build(1, 0, n);
    tree.modifySet(1, 0, n, 0, 0);

    for (int i = 1; i <= n; i++) {
        i64 _ = tree.query(1, 0, n, 0, a[i].y - 1);
        tree.modifySet(1, 0, n, a[i].y, _ + dx[a[i].x - 1]); // ---- (1)
        tree.modifyAdd(1, 0, n, 0, a[i].y - 1, dy[a[i].y - 1]); // ---- (2)
        tree.modifyAdd(1, 0, n, a[i].y + 1, n, dx[a[i].x - 1]); // ---- (3)
    }

    std::cout << tree.query(1, 0, n, 0, n) << "\n";

    // for (int i = 0; i < n; i++) {
    //  for (int j = 0; j <= n; j++) {
    //      if (f[i][j] == INF) continue;
    //      if (a[i + 1].y > j) {
    //          chmin(f[i + 1][a[i + 1].y], f[i][j] + dx[a[i + 1].x - 1]); (1)
    //          chmin(f[i + 1][j], f[i][j] + dy[a[i + 1].y - 1]); (2)
    //      }
    //      if (a[i + 1].y < j) chmin(f[i + 1][j], f[i][j] + dx[a[i + 1].x - 1]);
(3)
    //  }
    // }

    //  std::cout << *std::min_element(f[n], f[n] + 1 + n) << "\n";

    return 0;
```

```
}
```

D. 网络流

算法一：我会暴力网络流！

按照题目中暴力跑 n^2 次网络流，期望得分 20 分。

算法二：我会优化网络流！

使用当前弧优化的 Dinic 算法或者 ISAP, HLPP 等时间复杂度较优的网络流算法，期望得分 50 分（理论上 Dinic 是能卡的，这里数据可能造水了没卡掉）。

算法三：我会平面图转对偶图！

首先是一个典型的最大流转最小割，然后由于给定的图是一个平面图，我们建出它的对偶图，每次在上面跑最短路求出最小割，总时间复杂度 $\mathcal{O}(n^3 \log n)$，期望得分 50 分。

算法四：我会做给定图是一棵树的情况！

考虑如果给定的图是一棵树，我们相当于要求的就是所有点对直接的边权最小值。

这是容易的，按边权从大到小加边，使用并查集维护两端 siz 大小即可。

期望得分 0 分，因为并没有这个部分分。

算法五：我会正解！

考虑这张图的优秀性质，我们发现，当我们进入一个面时，我们一定会经过围成这个面中，与外界接壤的权值最小的边，于是我们考虑每次删掉这样的边，令其权值为 w，并且给围成这个面其他的边的权值加上 w，这样我们就转化成了一张新图，且答案与原图的答案一样。

重复上述操作，一直到图不存在面为止（也就是一棵树的情况），此时我们用算法四求出答案即可。

用堆维护上述删边过程，时间复杂度为 $\mathcal{O}(n \log n)$，期望得分 100 分。

```cpp
#include <bits/stdc++.h>

using i64 = long long;

constexpr int N = 5e5 + 5, P = 1e9 + 7;
constexpr i64 INF = (1ll << 60);

struct Edge {
    int u, v; i64 w;
    Edge(int u = 0, int v = 0, i64 w = 0) : u(u), v(v), w(w) {}
};

int n, m;
int fa[N], siz[N], p[N], cnt[N << 1], nxt[N << 2];
bool is[N << 1], vis[N << 1], del[N << 1];
Edge E[N << 1];
std::vector<std::pair<int, int>> adj[N];
std::map<int, int> M[N];

int father(int u) { return fa[u] = (fa[u] == u ? u : father(fa[u])); }
void merge(int u, int v) {
    u = father(u); v = father(v);
    if (u != v) {
```

```
        if (siz[u] < siz[v]) std::swap(u, v);
        fa[v] = u; siz[u] += siz[v];
    }
}

struct Node {
    i64 w; int u, v, id;
    Node(i64 w = 0, int u = 0, int v = 0, int id = 0) : w(w), u(u), v(v), id(id)
{}
    bool operator < (const Node &rhs) const {
        return (w == rhs.w ? id < rhs.id : w > rhs.w);
    }
};

int main() {
    freopen("flow.in", "r", stdin);
    freopen("flow.out", "w", stdout);

    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

    std::cin >> n >> m;
    for (int i = 1; i <= m; i++) {
        int u, v, w; std::cin >> u >> v >> w;
        if (u > v) std::swap(u, v);
        if (u == 1 && v == n) std::swap(u, v);
        E[i] = Edge(u, v, w);
        if (u + 1 == v || (u == n && v == 1)) is[i] = true;
        adj[u].push_back({v, i * 2});
        adj[v].push_back({u, i * 2 + 1});
    }
    for (int i = 1; i <= n; i++) {
        std::sort(adj[i].begin(), adj[i].end(), [&](std::pair<int, int> &lhs,
std::pair<int, int> &rhs) {
            int dl = lhs.first - i, dr = rhs.first - i;
            return (dl < 0 ? dl + n : dl) < (dr < 0 ? dr + n : dr);
        });
        for (int j = 0; j < (int)adj[i].size(); j++) {
            M[i][adj[i][j].first] = j;
            nxt[adj[i][j].second] = j;
        }
    }

    std::priority_queue<Node> q;
    for (int i = 1; i <= m; i++) if (is[i]) q.push({E[i].w, E[i].u, E[i].v, i *
2});

    auto isOut = [&](int idx) -> bool {
        if (is[idx]) return cnt[idx] >= 1;
        return cnt[idx] >= 2;
    };

    int rest = m;
    while (rest > n - 1) {
        int i = q.top().id, u = q.top().u, v = q.top().v; i64 w = q.top().w;
q.pop();
```

```
        if (isOut(i / 2) || w != E[i / 2].w) continue;
        cnt[i / 2]++; del[i / 2] = true; rest--;

        int cu = u, cv = v, cid = i;
        while (true) {
            int t, id;
            std::tie(t, id) = adj[cu][(nxt[cid] + 1) % adj[cu].size()];
            std::tie(cu, cv) = std::make_pair(t, cu); cid = id ^ 1;
            if (cu == u && cv == v) break;
            E[cid / 2].w += w; cnt[cid / 2]++;
            if (!isOut(cid / 2)) q.push({E[cid / 2].w, cv, cu, cid ^ 1});
        }
    }

    for (int i = 1, t = 0; i <= m; i++) if (!del[i]) p[++t] = i;
    std::sort(p + 1, p + n, [&](int &lhs, int &rhs) { return E[lhs].w > E[rhs].w;
});

    for (int i = 1; i <= n; i++) fa[i] = i, siz[i] = 1;

    i64 ans = 0;
    for (int i = 1; i < n; i++) {
        int u = E[p[i]].u, v = E[p[i]].v; i64 w = E[p[i]].w;
        u = father(u); v = father(v);
        ans = (ans + 1ll * siz[u] * siz[v] % P * w % P) % P;
        merge(u, v);
    }
    std::cout << ans << "\n";

    return 0;
}
```