Разбор задачи «N. Великий Флатландский Забор»

Авторы задачи — жюри олимпиады, автор разбора — А. Лахно

Решение на 30 баллов

Отметим, что решение можно искать в классе K-угольников. Если существует Забор, содержащий менее K башен и ограничивающий площадь S, то сущесвует и Забор, содержащий ровно K башен и ограничивающий площадь не менее S.

Забор однозначно задаётся номерами включаемых в него башен. При условии $N \leq 20$ ограничение по времени позволяет явно перебрать все сочетания из N по K, задающие возможные варианты построения Забора. Поскольку в случае выпуклой границы все диагонали целиком лежат внутри Флатландии, то для решения задачи достаточно из всех задаваемых сочетаниями вариантов выбрать тот, которому соответствует многоугольник наибольшей площади. Асимптотика такого решения составляет $O(C_N^K \cdot K)$.

Решение на 60 баллов

В случае невыпуклой границы некоторые диагонали могут выходить за пределы Флатландии. Из множества всех сочетаний из N по K, задающих варианты построения Забора, необходимо рассматривать только те, которые не содержат таких диагоналей.

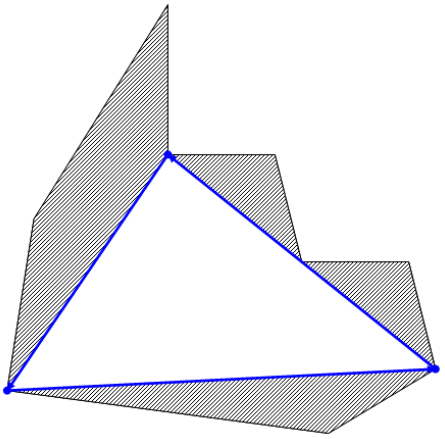
Диагональ $(P_i, P_j), 1 \leq i < j \leq N$, целиком лежит внутри многоугольника $(P_1, P_2, \ldots, P_N)$, если выполнены следующие три условия:

- Сумма площадей частей $(P_i, P_{i+1}, \ldots, P_j)$ и $(P_j, P_{j+1}, \ldots, P_N, P_1, \ldots, P_i)$ равна площади всего многоугольника $(P_1, P_2, \ldots, P_N)$.
- Диагональ не пересекает ни одну из сторон многоугольника, за исключением, быть может, пересечений по вершинам. Ситуация, когда сторона целиком лежит на диагонали, также не считается за пересечение.
- Рассмотрим множество вершин, лежащих на диагонали (P_i, P_j) за исключением самих P_i и P_j. Среди смежных к ним не должно существовать вершин, лежащих по разные стороны от прямой (P_i, P_j).

Проверка этих условий для одной диагонали производится за $O(N)$. Соответственно проверка всех диагоналей может быть реализована за $O(N^3)$. Общая асимптотика решения составляет $O(N^3 + C_N^K \cdot K)$.

Правильное решение

Рассмотрим граф G, в котором вершинам соответствуют башни, а рёбрам — диагонали, их соединяющие. Граф является ориентированным — диагонали (P_i, P_j) соответствуют сразу два ребра: $(i \to j)$ и $(j \to i)$. Вес ребра $(i \to j)$ определим как площадь отсекаемой диагональю (P_i, P_j) части Флатландии. Отсекаемой является та часть, которая остаётся по правую сторону от диагонали при движении от башни i к башне j.



В вышеописанных терминах задача построения оптимального Забора сводится к нахождению в графе G цикла минимального веса, содержащего ровно K рёбер. Обозначим через $A^k_{N\times N}$ матрицу длин кратчайших путей, содержащих ровно k рёбер. Элемент a^k_{ij} содержит длину кратчайшего пути из вершины i в вершину j. В частности, A^1 совпадает с матрицей весов рёбер графа G. Матрица A^K может быть вычислена при помощи соотношения $A^k = A^{k-1} \times A^1$, где операция умножения $C = A \times B$ задаётся следующей формулой:

$$c_{ij} = \min_k(a_{ik} + b_{kj}).$$

Площадь, ограничиваемая оптимальным Забором, равна $\min(a^K_{ii}|i = 1 \dots N)$.

Время перемножения двух матриц размером $N \times N$ составляет порядка $O(N^3)$. Итоговая асимптотика описанного решения — $O(N^3 \cdot K)$. Если же для вычисления A^k вместо соотношения $A^k = A^{k-1} \times A^1$ использовать алгоритм быстрого возведения в степень, то задача может быть решена за $O(N^3 \log K)$.

Ниже приведён текст решения с использованием алгоритма быстрого возведения в степень:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h>

const char inpfile[] = "h.in";
const char outfile[] = "h.out";

const int NMAX = 305;
const int MMAX = 20;
const int INF = INT_MAX / 2;

struct Point
{
    int x, y;
};

Point border[2*NMAX];
int square;
int n, k;

int a[MMAX][NMAX][NMAX];
int b[MMAX][NMAX][NMAX];
int prev[MMAX];
int m = 1;

int Vc(const Point &p1, const Point &p2)
```

```
{
    return p1.x * p2.y - p1.y * p2.x;
}

int Vc(const Point &p1, const Point &p2, const Point &p3, const Point &p4)
{
    return (p2.x - p1.x) * (p4.y - p3.y) - (p2.y - p1.y) * (p4.x - p3.x);
}

int Sc(const Point &p1, const Point &p2, const Point &p3, const Point &p4)
{
    return (p2.x - p1.x) * (p4.x - p3.x) + (p2.y - p1.y) * (p4.y - p3.y);
}

int Sign(int x)
{
    if (x > 0)
        return 1;
    else if (x == 0)
        return 0;
    else
        return -1;
}

void BuildFenceSegment(int i, int j)
{
    int s1 = Vc(border[j], border[i]);
    for (int k = i; k < j; k++)
        s1 += Vc(border[k], border[k + 1]);

    int s2 = Vc(border[i], border[j]);
    for (int k = j; k < i + n; k++)
        s2 += Vc(border[k], border[k + 1]);

    bool ok = abs(s1) + abs(s2) == square;

    bool sn[3];
    memset(sn, 0, sizeof(sn));

    if (ok)
        for (int k = 0; k < n; k++)
            if (Vc(border[i], border[j], border[i], border[k]) == 0
                && Sc(border[k], border[i], border[k], border[j]) < 0)
            {
                int k1 = k > 0 ? k - 1 : n - 1;
                int k2 = k + 1;

                sn[Sign(Vc(border[i], border[j], border[i], border[k1])) + 1] = true;
```

```
                sn[Sign(Vc(border[i], border[j], border[i], border[k2])) + 1] = true;
                if (sn[0] && sn[2])
                {
                    ok = false;
                    break;
                }

            }


    if (ok)
        for (int k = 0; k < n; k++)
            if (Sign(Vc(border[k], border[k + 1], border[k], border[i])) *
            Sign(Vc(border[k], border[k + 1], border[k], border[j])) < 0
            && Sign(Vc(border[i], border[j], border[i], border[k])) *
            Sign(Vc(border[i], border[j], border[i], border[k + 1])) < 0)
        {
            ok = false;
            break;
        }

    if (ok)
    {
        a[0][i][j] = s1;
        a[0][j][i] = s2;
    }
    else
    {
        a[0][i][j] = INF;
        a[0][j][i] = INF;
    }
}

void Mul(int w1[][NMAX], int w2[][NMAX], int r[][NMAX], int p[][NMAX])
{
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
        {
            r[i][j] = INF;
            for (int k = 0; k < n; k++)
                if (r[i][j] >= w1[i][k] + w2[k][j])
                {
                    r[i][j] = w1[i][k] + w2[k][j];
                    p[i][j] = k;
                }
        }
}
```

```
void RestoreFence(int m, int i, int j)
{
    if (m > 0)
    {
        RestoreFence(m - 1, i, b[m][i][j]);
        printf("%d ", b[m][i][j] + 1);
        RestoreFence(prev[m], b[m][i][j], j);
    }
}

int main()
{
    freopen(inpfile, "r", stdin);
    freopen(outfile, "w", stdout);

    scanf("%d%d", &n, &k);
    for (int i = 0; i < n; i++)
    {
        scanf("%d%d", &border[i].x, &border[i].y);
        border[i + n] = border[i];
    }

    square = 0;
    for (int i = 0; i < n; i++)
        square += Vc(border[i], border[i + 1]);

    for (int i = 0; i < n; i++)
        a[0][i][i] = INF;

    for (int i = 0; i < n - 1; i++)
        for (int j = i + 1; j < n; j++)
            BuildFenceSegment(i, j);

    int r = 1;
    while (k >> r)
        r++;

    for (int h = r - 2; h >= 0; h--)
    {
        Mul(a[m - 1], a[m - 1], a[m], b[m]);
        prev[m] = m - 1;
        m++;

        if (k & (1 << h))
        {
            Mul(a[m - 1], a[0], a[m], b[m]);
            prev[m] = 0;
            m++;
```

```
        }
    }

    int res = INF;
    int v = 0;

    for (int i = 0; i < n; i++)
        if (a[m - 1][i][i] < res)
        {
            res = a[m - 1][i][i];
            v = i;
        }

    printf("%.5lf\n", (square - res) / 2.0);
    printf("%d\n", k);

    printf("%d ", v + 1);
    RestoreFence(m - 1, v, v);

    return 0;
}
```