South China University of Technology

SCUT_gugugu

# TEMPLATE


Last build at October 20, 2020

# Contents

**1 Graph Theory** **3**
1.1 Shortest Path . . . 3
1.2 Network Flow . . . 3
1.3 Tree Related . . . 5
1.4 LCA . . . 5
1.5 Tarjan . . . 5
1.6 Cactus . . . 5

**2 Data Structures** **6**
2.1 Basic Structures . . . 6
2.2 Heap Structures . . . 6
2.3 Sequence Structures . . . 6
2.4 Persistent Data Structures . . . 6
2.5 Tree Structures . . . 6

**3 String** **7**
3.1 Basics . . . 7
3.2 String Matching . . . 7
3.3 Suffix Related . . . 7
3.4 Palindrome Related . . . 7
3.5 Substring Automaton . . . 7

**4 Math** **10**

**5 Geometry** **11**

**6 Game Theory** **12**
6.1 Bash's Game . . . 12
6.2 Wythoff' s Game . . . 12
6.3 Fibonacci' s Game / Zeckendorf's theory . . . 12
6.4 Nim' s Game / Anti-Nim's Game / K-Nim's Game / Anti-K-Nim's Game . . . 12
6.5 阶梯博弈 . . . 13
6.6 Multi-Nim . . . 14
6.7 Every-SG . . . 14
6.8 树的删边游戏 . . . 15
6.9 Chomp's theory? . . . 15
6.10 Other's theory? . . . 15
6.11 SG Theory . . . 15
6.12 SJ Theory . . . 16
6.13 Surreal Number Theory . . . 16

**7 动态规划** **17**
7.1 最大子矩阵 . . . . . 17
7.2 斜率优化 . . . . . 18
7.3 四边形不等式优化 . . . . . 18
7.4 忘情水二分 . . . . . 18

# 1 Graph Theory

## 1.1 Shortest Path

## 1.2 Network Flow

修订稿 Created By ChrisJaunes 最大权闭合子图建模 eg: (+++) - (—)
定义：在一个有向图中，每个点都有一个点权。闭合子图：对于这个子图，它任意一个点的的后继必须在这个子图中。最大权闭合子图：在所有的闭合子图中，该图的点权和最大。
建模：
1. 建立超级源 S 和超级汇 T
2. 把所有点权为正的点与 S 连接一条有向边，方向是从 S 到 u，边权是点权；把所有点权为负的点与 T 连接一条有向边，方向是从 u 到 T，边权是点权的相反数；
3. 原图中所有有向边的边权是 INT_MAX；也可以根据情况按照割的思想处理。
答案：最大权闭合子图的权值和就是所有点的点权和-该生成图的最小割

独立集: 一个点集，点集中的各点没有关系。
最大独立集: 点的个数最多的独立集。
求解一般无向图的最大团、最大独立集是 NPC 问题。
二分图最大独立集 = 点的总数 - 最小点覆盖。
最大点权独立集的点权和 = 所有点权和-最小点权覆盖的点权
路径覆盖：给定有向图 G=(V,E)。设 P 是 G 的一个简单路 (顶点不相交) 的集合。如果 V 中每个定点恰好在 P 的一条路上，则称 P 是 G 的一个路径覆盖。
最小路径覆盖：G 的最小路径覆盖是 G 所含路径条数最少的路径覆盖
建模：拆点，对于原图的 (u,v), 连接 (u->v')
答案：原图的结点数-新图的最大匹配数

最小割的可行边和必须边 (所有割集的交集和并集)
注意：必须边 可行边.
1. 残余网络中有剩余流量的边一定不在最小割中
2. 残余网络中一条满流边的首尾还能相互到达，那么这条边不是可行边
3. 残余网络中一条满流的首尾分别与 S 和 T 在一个强连通分量中，那么这条边为必须边
具体实现, 需要先跑最大流, 然后 Tarjan 缩强连通分量, 条件是:
可行边: 两端不在一个强连通分量内。
必须边: 一端在 S 的分量内, 另一端在 T 的分量内。
必须边对于小规模也可以强行割去然后跑流判断流是否减少，暴力对费用流必须边同样适用

求割边最少的最小割: 把边的权值全部乘以一个较大的数 E 再加 1, E 要严格大于边的数量
最小割: ans/E;
最小割下的最小割边: ans % E
公平分配模型：
题意：n 个球队，已经有一些胜负场，现在还有一些场次，你去分配胜负，问每支球队有没有可能获胜
解：
1. 枚举每一个人，贪心计算比赛最大获胜次数 max_win

2. 将比赛当作节点，从源到比赛连比赛次数边
3. 分配比赛胜利者，从比赛到人连无穷边
4. 从人到汇连 max_win - 已经获胜的比赛次数的边
5. 检查是否能将比赛全部安排完

混合图重定向为欧拉回路
1. 将所有的无向边随便定向，计算得到每个点的出度与入度之差 deg
2. 建立源点汇点 S,T
3. 对于 deg>0 的点 v，连一条边 Su, 流为 deg/2;
4. 对于 deg<0 的点 u，连一条边 uT，流为-deg/2;
5. 对于有向边，忽略；对于无向边 (u,v)，连 (u,v,1)
5. 判断是否能跑满流

费用流：物品通过代价重复使用
1. 拆点，使用物品前和使用物品后
2. 对于节点 [使用物品前]：考虑物品来源: 源点购买能力、购买费用、节点 [使用物品后]INF、购买费用; 去向: 汇点使用量、0
3. 对于节点 [使用物品后]：考虑物品来源: 源点使用量、0，去向: 节点 [使用物品前]

费用流：物品通过代价延后销售
1. 拆点，生产物品和出售物品
2. 对于节点 [生产物品]：考虑物品来源: 源点生产能力、生产费用; 去向: 节点 [出售物品]INF、保存的代价
3. 对于节点 [出售物品]：考虑物品来源: 通过节点生产物品，去向: 汇点销售能力、销售费用

费用流：n 个点 m 条带权有向边的图，要给边染色，染色的边形成若干个回路且每个点都恰好属于其中 k 个回路。问最少要染多少边权和的路。
一个回路里面各个点的入度 = 出度 =1，各个点如果都恰好属于 k 个回路那么各个点的入度 = 出度 =k。
这样就考虑用最小费用最大流了：
1. 所有点 u 拆成两点 u 和 u'，分别代表出度和入度
2. 原点向 u 连容量 k 费用 0 的边，u' 向汇点连容量 k 费用 0 的边
3. 所有有向边 <u,v>，u 向 v' 连容量 1 费用边权的边
4. 这样跑最小费用最大流，如果最大流等于 n*k，也就是说各个点的出度 = 入度 =k，那么就有解，最小费用就是最小的解

费用流：费用不是线性关系
费用为 $a * x^2$ : 拆边，拆成 a * 1, a * 3, a * 5, ...
费用为之前的等待时间 + a : 反向考虑，倒数第 i 个点对整体贡献为 a * i, 拆点，拆成 a, a*2, a*3, ...

费用流：平面上给定一些黑点白点，要求一个黑点连接一个白点，并且所有线段都不相交
合法情况的连线，总长度明显比不合法情况的小，题意可以转化为求让每条线段的长度和最小的方案
1、建立超级源超级汇
2、从超级源向左侧点连流量为 1，费用为 0 的边
3、从左侧点向右侧点连流量为 1，费用为两点欧几里得距离的边

4、从右侧点向超级汇连流量为 1，费用为 0 的边
5、跑最小费用最大流

区间 k 覆盖问题：给定 n 个带权开区间，区间最多可以选一次，要求保证实轴上的所有数被选择的次数要小于 k 次，问最大的权值。
1. 建立超级源超级汇
2. 源点向第一个点连一条容量为 k、费用为 0 的边
3. 相邻节点连上用容量为 k 费用为 0 的边
4. 最后一个点向超级汇连一条容量为 k、费用为 0 的边
5. 区间的两个端点之间连接一条容量为 1 的，费用为 w 的边

区间至少 ai 覆盖问题
1. 建立超级源超级汇
2. 源点向第一个点连一条容量为 INF、费用为 0 的边
3. 相邻节点连上用容量为 INF-ai 费用为 0 的边
4. 最后一个点向超级汇连一条容量为 INF、费用为 0 的边
5. 区间的两个端点之间连接一条容量为 INF 的，费用为 c 的边

分治优化建图
1. 费用为 |ai-aj| 可以利用差分性质通过累积权值之间的差值实现费用的绝对值
这样可以采用分治方式将边的数量从 n*n 将为 n*logn
2. 一个点向一个区间的每一个点连边，可以利用线段树转为每个点最多向 log 个虚点连边
线段树上的点向孩子节点连边 INF，叶子节点向孩子连边 INF

## 1.3 Tree Related

尚未开始

## 1.4 LCA

尚未开始

## 1.5 Tarjan

尚未开始

## 1.6 Cactus

尚未开始

# 2 Data Structures

## 2.1 Basic Structures

尚未开始

## 2.2 Heap Structures

尚未开始

## 2.3 Sequence Structures

尚未开始

## 2.4 Persistent Data Structures

尚未开始

## 2.5 Tree Structures

尚未开始

# 3 String

## 3.1 Basics

尚未开始

## 3.2 String Matching

尚未开始

## 3.3 Suffix Related

尚未开始

## 3.4 Palindrome Related

尚未开始

## 3.5 Substring Automaton

求解子序列个数，不重复, 包括空串

```
int dfs(int x) {
    if(dp[x]) return dp[x];
    dp[x] = 1;
    for(int i = 0; i < 26; i++)
        if(ch[x][i] != -1) dp[x] += dfs(ch[x][i]);
    return dp[x];
}
```

求解回文公共子序列个数，不重复，包括空串
原串与反串都建一遍, 这样有两个字符串,x+y<=n+1 这个序列才是合法的
x+y=n+1 以外的情况统计奇回文序列要 ++f[x][y]
特殊处理 x+y=n+1、x+y<n+1 时不 ++f[x][y], 可以统计偶回文序列
相减可以统计奇回文序列

```
LL dfs(LL x,LL y) {
    if(f[x][y]) return f[x][y];
    for(LL i = 0; i < 26; i++)
    if(ch1[x][i] != -1 && ch2[y][i] != -1) {
        if(ch1[x][i]+ch2[y][i] > n+1) continue;
        if(ch1[x][i]+ch2[y][i] < n+1) f[x][y]++;
        f[x][y] = (f[x][y]+dfs(ch1[x][i],ch2[y][i]))%mod;
    }
    return ++f[x][y];
}
```

求解两个字符串的所有公共子序列个数，可重复

```
for(int i=1;i<=n;i++)
    for(int j=1;j<=m;j++) {
        f[i][j]=f[i-1][j]+f[i][j-1]-f[i-1][j-1];
        if(arr[i]==brr[j]) f[i][j]+=f[i-1][j-1]+1;
        if(f[i][j]<0) f[i][j]+=mod;
        if(f[i][j]>=mod) f[i][j]%=mod;
    }
```

求解两个字符串的所有公共子序列个数，去重复，包括了空串输入：a 串长度、b 串长度、a 串、b 串、（0：输出所有的公共子序列个数 |1：按照字典序输出所有不重复公共子序列）

```
#include <bits/stdc++.h>
using namespace std;
typedef long long LL;
const LL MOD = 1e9;
const int MAXN = 3020;
const int MAXM = 3e6 + 7;
const int MAXS = 58;
struct Big {//压位高精度
    int len, a[20];
    Big() {
        len = 0;
        for (int i = 0; i < 20; i++) a[i] = 0;
    }
    void put() {
        printf("%d", a[len]);
        for (int i = len - 1; i >= 0; i--) printf("%09d", a[i]);
    }
    void operator=(int o) {
        if (o >= MOD) {
            a[0] = o % MOD;
            a[len = 1] = 0 / MOD;
        } else
            a[len = 0] = o;
    }
    void operator+=(const Big& o) {
        len = max(len, o.len);
        for (int i = 0; i <= len; i++) {
            a[i] += o.a[i];
            if (a[i] >= MOD) {
                a[i + 1] += a[i] / MOD;
                a[i] %= MOD;
            }
        }
        while (a[len + 1]) len++;
    }
} dp[MAXM];

char sta[MAXN];
int top = -1;

char a[MAXN], b[MAXN];
int an, bn, k;
LL ans;
int cha[MAXN][MAXS], chb[MAXN][MAXS];

int markn, mark[MAXN][MAXN];
void build(char* s, int sn, int (*ch)[58]) {
    for (int j = 0; j < 58; j++) ch[sn][j] = ch[sn + 1][j] = -1;
    for (int i = sn; i >= 1; i--) {
        for (int j = 0; j < 58; j++) ch[i - 1][j] = ch[i][j];
        ch[i - 1][s[i] - 'A'] = i;
    }
}
int dfs0(int x, int y) {
    if (mark[x][y])
        return mark[x][y];
    int p = mark[x][y] = ++markn;
    dp[p] = 1;
```

```
    for (int i = 0; i < 58; i++) {
        if (cha[x][i] != -1 && chb[y][i] != -1) {
            dp[p] += dp[dfs0(cha[x][i], chb[y][i])];
        }
    }
    return mark[x][y];
}
LL dfs1(int x, int y) {
    puts(sta);
    LL cnt = 1;
    for (int i = 0; i < 58; i++) {
        if (cha[x][i] != -1 && chb[y][i] != -1) {
            sta[++top] = i + 'A';
            cnt += dfs1(cha[x][i], chb[y][i]);
            sta[top--] = '\0';
        }
    }
    return cnt;
}
int main() {
    scanf("%d%d%s%s%d", &an, &bn, a + 1, b + 1, &k);
    build(a, an, cha);
    build(b, bn, chb);
    if (k == 1)
        dfs1(0, 0);
    dfs0(0, 0);
    dp[mark[0][0]].put();
    return 0;
}
```

# 4 Math

尚未开始

# 5 Geometry

尚未开始

# 6 Game Theory

## 6.1 Bash's Game

Bash's Game 巴什博弈
有一堆个数为 n 的石子，游戏双方依次从中拿取，满足:
1. 每次至少取 1 个，最多取 m 个.
最后取光者得胜。
结论: n = t(m+1) + r, 必败态:r = 0;
巴什博弈变种:
取一个指定集合的石头个数
取到最后一个石子输, n = t(m + 1) + r, r = 1;

## 6.2 Wythoff' s Game

Wythoff' s Game（威佐夫博弈）
有两堆分别为 (an, bn) 的石子, 游戏双方依次从中拿取，满足:
1. 从任意一堆中取任意个 $> 1$。2. 从两堆中取同样多个。最后取完者胜.
结论: 对于任意的局势 (a, b)(a $<$ b), 必败点为 (b-a)*(sqrt(5)+1)/2=a.

## 6.3 Fibonacci' s Game / Zeckendorf's theory

Fibonacci' s Game（斐波那契博弈）
有一堆个数为 n 的石子，游戏双方轮流取石子，满足：
1. 先手不能在第一次把所有的石子取完；
2. 之后每次可以取的石子数介于 1 到对手刚取的石子数的 2 倍之间（包含 1 和对手刚取的石子数的 2 倍）。
结论: 必败点是斐波那契数

齐肯多夫定理: 任何正整数可以表示为若干个不连续的 Fibonacci 数之和

## 6.4 Nim' s Game / Anti-Nim's Game / K-Nim's Game / Anti-K-Nim's Game

Nim' s Game（尼姆博弈）
石子的个数可以等价成某个游戏的 SG 函数。

有 n 堆石子，游戏双方依次从中拿取, 满足:
1. 规定每次只能从一堆中取若干根，可将一堆全取走，但不可不取.
最后取完者为胜。
结论:
T 态: 所有火柴数异或和为 0
S 态: 所有火柴数异或和不为 0

必胜态:S

有 n 堆石子, 游戏双方依次从中拿取, 满足:
1. 规定每次只能从一堆中取若干根，可将一堆全取走，但不可不取.
最后取完者为败。
结论:
S0 态: 即仅有奇数个孤单堆
T0 态: 即仅有偶数个孤单堆
S1 态: 异或和大于 0, 且有 1 个充裕堆
T1 态: 不存在
S2 态: 异或和大于 0, 且有多个充裕堆
T2 态: 异或和等于 0, 且有多个充裕堆
必胜态:T0,S1,S2
必败态:S0,T2

有 n 堆石子，游戏双方依次从中拿取, 满足:
1. 规定每次只能至多 k 堆中取若干根，可将 k 堆全取走，但不可不取.
最后取完者为胜。
结论:
对于每一堆，把它石子的个数用二进制表示
必败态: 对所有的石子堆，如果在任何一个二进制位上 1 的个数总是 k+1 的整数倍

有 n 堆石子，游戏双方依次从中拿取, 满足:
1. 规定每次只能至多 k 堆中取若干根，可将 k 堆全取走，但不可不取
最后取完者为败。
结论:
1. 对于每一堆，把它石子的个数用二进制表示
2. 所有的堆（非零堆，下同）全是 1, 此时如果 1 堆个数模 k+1 的结果是 1 则必败，否则必胜 (我们可以通过拿走 0 到 k 个堆来随意调整当前状态模的结果，然后再将所有大于 1 的堆降到 1 就行了)
3. 有多于 k 个堆的个数大于 1。必胜

## 6.5 阶梯博弈

有 n 个阶梯呈升序排列，每个阶梯上有若干个石子，游戏双方轮流取石子, 满足:
1. 将一个阶梯上的石子移任意个（>0）到前一个台阶。
当没有可行操作时（所有石子都被移动到了地面，即第 0 号台阶）输。
结论:
奇数号台阶的 Nim 游戏
变种 1: 树上, 每个石子只能往父亲节点移动.
变种 2:
游戏双方在一个 1*N 的格子内挪动棋子，刚开始在若干个位置上有棋子, 每个位置至多一个棋子
每一个选手可以进行的操作时选择一个棋子并把它向左方移动，当然不能越过其它的棋子，也不能超出边界。
谁不能移动谁就输了。求谁会赢？

结论:
将棋子位置按升序排列, 然后从后往前两两绑成一对, 如果个数是奇数, 那么将第一个和边界外绑定.
一对棋子的前一个和前一对棋子的后一个之间有多少个空位置对最终的结果是没有影响的。
于是我们只需要考虑同一对的两个棋子之间有多少空位, 将同一对棋子间的空位视为石子, 做 nim 游戏
两对棋子间的空格数当奇数位石子, 其他当偶数位石子, 石子相右边移动
变种 3:
山上有 n 个人，每个人给出距离山顶的距离，给出其中一个人为 king，每次能挑选一个人向上移动，不能越过其他人，最后将 king 移动到山顶者获胜。问获胜者。
结论:
只要把 King 当作普通人一样处理即可。除了两种特殊情况：
1. 当 King 是第一个人时，Alice 直接胜
2. 当 King 是第二个人且一共有奇数个人时，第一堆的大小需要减 1。

## 6.6 Multi-Nim

有 n 堆石子，游戏双方依次从中拿取, 满足:
1. 任意一堆石子中拿任意多个石子 (不能不拿)
2. 把一堆数量不少于 2 石子分为两堆不为空的石子
最后取完者为胜。
结论:
操作一与普通的 Nim 游戏等价
操作二实际上是将一个游戏分解为两个游戏，根据 SG 定理，我们可以通过异或运算把两个游戏连接到一起，作为一个后继状态

$$SG(x) \equiv \begin{cases} x-1 & (x mod 4 = 0) \\ x & (x mod 4 = 1 or 2) \\ x+1 & (x mod 4 = 3) \end{cases} \tag{1}$$

Multi-SG 游戏规定，在符合拓扑原则的前提下，一个单一游戏的后继可以为多个单一游戏。
Multi-SG 其他规则与 SG 游戏相同。
注意在这里要分清楚后继与多个单一游戏
对于一个状态来说，不同的划分方法会产生多个不同的后继，而在一个后继中可能含有多个独立的游戏
一个后继状态的 SG 值即为后继状态中独立游戏的异或和
该状态的 SG 值即为后继状态的 SG 值中未出现过的最小值

## 6.7 Every-SG

给定一张无向图，上面有一些棋子，两个顶尖聪明的人在做游戏，每人每次必须将可以移动的棋子进行移动，不能移动的人
因为两个人都顶尖聪明，因此当一个人知道某一个游戏一定会输的话，它一定会尽力缩短游戏的时间，当它知道某一个游戏一定会赢的话，一定会尽力延长游戏的时间
对于还没有结束的单一游戏，游戏者必须对该游戏进行一步决策；
其他规则与普通 SG 游戏相同

Every-SG 游戏与普通 SG 游戏最大的不同就是它多了一维时间
对于 SG 值为 0 的点，我们需要知道最少需要多少步才能走到结束
对于 SG 值不为 0 的点，我们需要知道最多需要多少步结束
这样我们用 step 变量来记录这个步数

$$step(x) \equiv \begin{cases} 0 & u \\ maxstep(v) & sg(u)!=0vusg(v)=0 \\ minstep(v) & sg(u)=0vu \end{cases} \tag{2}$$

## 6.8 树的删边游戏

给出一个有 N 个点的树，有一个点作为树的根节点。游戏者轮流从树中删去边，删去一条边后，不与根节点相连的部分将被移走。谁无法移动谁输。
结论:
Colon Principle: 对于树上的某一个点，ta 的分支可以转化成以这个点为根的一根竹子，这个竹子的长度就是 ta 各个分支的边的数量的异或和
叶子节点的 SG 值为 0；中间节点的 SG 值为它的所有子节点的 SG 值加 1 后的异或和。

## 6.9 Chomp's theory?

取一个无关紧要的位置, 如果对方必胜, 则学习其策略, 我方必胜.

## 6.10 Other's theory?

有 n 堆石子，游戏双方依次从中拿取, 满足:
1. 规定每次能从任意多堆中取 1 根, 不可不取.
最后取完者为胜。
结论: 如果全是偶数, 先手必败, 否者先手必胜

一个无相联通图，有一个点作为图的根。
游戏者轮流从图中删去边，删去一条边后，不与根节点相连的部分将被移走。
谁无路可走谁输。
结论:
Fusion Principle: 环上的点可以融合，且不改变图的 SG 值, 我们可以把一个带有奇数边的环等价成只有一个端点的一条边而偶数边的环等价于一个点

## 6.11 SG Theory

```
memset(mex, 0, sizeof mex);
for (int i = 1; i < maxN;++i) {
    for (int j = 1; j <= n;++j) {
        if (a[j] <= i)
            mex[SG[i - a[j]]] = i;
            for (int k = 1; i - k - a[j] > 0;++k)
                mex[SG[k] ^ SG[i - k - a[j]]] = 1;
```

```
    }
    for (int j = 0;; ++j)
        if (mex[j]!=i){
            SG[i] = j;
            break;
        }
}
```

## 6.12 SJ Theory

反公平游戏 Anti-SG Game

DAG 上没有出度的点为胜利状态，其它定义与一般游戏相同。

现在的问题是解决多个反公平游戏的合并。

SJ 定理说明：先手必胜，当且仅当以下两个条件同时成立或同时不成立：

1. 合并的 SG 值为 0；
2. 所有游戏的 SG 值不超过 1。

## 6.13 Surreal Number Theory

# 7 动态规划

## 7.1 最大子矩阵

单调栈法：

```cpp
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 1007;
int a[MAXN][MAXN], b[MAXN][MAXN];
pair<int, int> mp[MAXN * MAXN];
pair<int, int> off[MAXN][MAXN];
int tail;
pair<int, int> st[MAXN];
int Up[MAXN][MAXN];
int ans = 0;
void maintain(pair<int, int> v) {
    for(;tail && st[tail].first >= v.first; tail--) {
        ans = max(ans, st[tail].first * (v.second - st[tail-1].second-1));
    }
}
void clear(int pos) {
    for(;tail; tail--)
        ans = max(ans, st[tail].first * (pos -st[tail-1].second-1));
}
int main() {
    ios::sync_with_stdio(0); cin.tie(0);
    int n, m;
    cin >> n >> m;
    for(int i = 1; i <= n; i++)
        for(int j = 1; j <= m; j++) {
            cin >> a[i][j];
            mp[a[i][j]] = {i,j};
        }
    for(int i = 1; i <= n; i++)
        for(int j = 1; j <= m; j++) {
            cin >> b[i][j];
            off[i][j] = {i - mp[b[i][j]].first, j - mp[b[i][j]].second};

        }
    for(int i = 1; i <= n; i++) {
        st[0].second = 0;
        for(int j = 1; j <= m; j++) {
            Up[i][j] = i > 1 && off[i][j] == off[i-1][j] ? Up[i-1][j] + 1 : 1;
            if(j > 1 && off[i][j] == off[i][j-1])
                maintain({Up[i][j], j}); else {clear(j); st[0].second = j-1;}
            st[++tail] = {Up[i][j], j};
        }
        clear(m+1);
    }
    cout << ans << endl;
    return 0;
}
```

悬线法：

```cpp
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 1007;
int a[MAXN][MAXN], b[MAXN][MAXN];
pair<int, int> mp[MAXN * MAXN];
```

```cpp
pair<int, int> off[MAXN][MAXN];
int L[MAXN][MAXN], R[MAXN][MAXN], UP[MAXN][MAXN];
int main() {
    ios::sync_with_stdio(0); cin.tie(0);
    int n, m;
    cin >> n >> m;
    for(int i = 1; i <= n; i++)
        for(int j = 1; j <= m; j++) {
            cin >> a[i][j];
            mp[a[i][j]] = {i,j};
        }
    for(int i = 1; i <= n; i++)
        for(int j = 1; j <= m; j++) {
            cin >> b[i][j];
            off[i][j] = {i - mp[b[i][j]].first, j - mp[b[i][j]].second};
            L[i][j] = R[i][j] = j;
            UP[i][j] = 1;
        }
    for(int i = 1; i <= n; i++)
        for(int j = 2; j <= m; j++) {
            if(off[i][j] == off[i][j-1]) L[i][j] = L[i][j-1];
        }
    for(int i = 1; i <= n; i++)
        for(int j = m - 1; j >= 1; j--) {
            if(off[i][j] == off[i][j+1]) R[i][j] = R[i][j+1];
        }
    int ans = 0;
    for(int i = 1; i <= n; i++)
        for(int j = 1; j <= m; j++) {
            if(i > 1 && off[i][j] == off[i-1][j]) {
                UP[i][j] = UP[i-1][j] + 1;
                L[i][j] = max(L[i][j], L[i-1][j]);
                R[i][j] = min(R[i][j], R[i-1][j]);
            }
            ans = max(ans, (R[i][j] - L[i][j] + 1) * UP[i][j]);
        }
    cout << ans << endl;
    return 0;
}
```

## 7.2 斜率优化

## 7.3 四边形不等式优化

## 7.4 忘情水二分