202312

T1

试题编号：	**202312-1**
试题名称：	仓库规划
时间限制：	1.0s
内存限制：	512.0MB
题目分类：	暴力，枚举

问题描述

西西艾弗岛上共有 n 个仓库，依次编号为 1…n。每个仓库均有一个 m 维向量的位置编码，用来表示仓库间的物流运转关系。

具体来说，每个仓库 i 均可能有一个**上级**仓库 j，满足：仓库 j 位置编码的每一维**均大于**仓库 i 位置编码的对应元素。比如编码为 (1,1,1) 的仓库可以成为 (0,0,0) 的上级，但不能成为 (0,1,0) 的上级。如果有多个仓库均满足该要求，则选取其中编号最小的仓库作为仓库 i 的上级仓库；如果没有仓库满足条件，则说明仓库 i 是一个物流中心，没有上级仓库。

现给定 n 个仓库的位置编码，试计算每个仓库的上级仓库编号。

输入格式

从标准输入读入数据。

输入共 n+1 行。

输入的第一行包含两个正整数 n 和 m，分别表示仓库个数和位置编码的维数。

接下来 n 行依次输入 n 个仓库的位置编码。其中第 i 行（1≤i≤n）包含 m 个整数，表示仓库 i 的位置编码。

输出格式

输出到标准输出。

输出共 n 行。

第 i 行（1≤i≤n）输出一个整数，表示仓库 i 的上级仓库编号；如果仓库 i 没有上级，则第 i 行输出 0。

样例输入

```
4 2
0 0
-1 -1
1 2
0 -1
```

样例输出

```
3
1
0
3
```

样例解释

对于仓库 2:(−1,−1) 来说，仓库 1:(0,0) 和仓库 3:(1,2) 均满足上级仓库的编码要求，因此选择编号较小的仓库 1 作为其上级。

子任务

50% 的测试数据满足 m=2；

全部的测试数据满足 0<m≤10、0<n≤1000，且位置编码中的所有元素均为**绝对值**不大于 106 的整数。

题解

代码如下：

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

using namespace std;

int main() {
  int n, m;
  cin >> n >> m;
  // 获取初始n和m的值 n表示n个仓库，m表示m个编码 -> 快速根据m构建m个元素的仓库
  vector<vector<int> > a(n + 1, vector<int>(m));
  // 这里注意创建vector的语法
  for (int i = 1; i <= n; i++) {  // 仓库编号从1开始，编码编号从0开始
    for (int j = 0; j < m; j++) {
      cin >> a[i][j];
    }
  }
  for (int i = 1; i <= n; i++) {  // i为上层
    // cout << "here i=" << i << endl;
    int flag = 1;
    for (int j = 1; j <= n; j++) {  // j为内层
      if (j == i) continue;
      // cout << "here j=" << j << endl;
      flag = 1;
```

```
      for (int k = 0; k < m; k++) {
        if (a[i][k] >= a[j][k]) {
          flag = 0;
          break;
        }
      }
      // cout << "here flag=" << flag << endl;
      if (flag == 1) {
        cout << j << endl;
        break;
      }
    }
    if (flag == 0) {
      cout << 0 << endl;
    }
  }
}
```

核心语句

1.核心语法

```
vector<vector<int> > a(n + 1, vector<int>(m));
// 左边是数组个数。右边看作是每个元素含有的元素个数
```

2.核心设计

> 暴力枚举：好在不会超时，flag的设计稍微繁琐了：不过按照自己的方式来就好

```
for (int i = 1; i <= n; i++) {  // i为上层
    // cout << "here i=" << i << endl;
    int flag = 1;
    for (int j = 1; j <= n; j++) {  // j为内层
      if (j == i) continue;
      // cout << "here j=" << j << endl;
      flag = 1;
      for (int k = 0; k < m; k++) {
        if (a[i][k] >= a[j][k]) {
          flag = 0;
          break;
        }
      }
      // cout << "here flag=" << flag << endl;
      if (flag == 1) {
        cout << j << endl;
        break;
      }
    }
    if (flag == 0) {
      cout << 0 << endl;
```

```
    }
```

T2

试题编号：	**202312-2**
试题名称：	因子化简
时间限制：	2.0s
内存限制：	512.0MB
题目分类：	质因数分解，map，朴素算法

题目背景

质数（又称“素数”）是指在大于 1 的自然数中，除了 1 和它本身以外不再有其他因数的自然数。

问题描述

小 P 同学在学习了素数的概念后得知，任意的正整数 n 都可以唯一地表示为若干素因子相乘的形式。如果正整数 n 有 m 个不同的素数因子 p1,p2,⋯,pm，则可以表示为：n=p1t1×p2t2×⋯×pmtm。

小 P 认为，每个素因子对应的指数 ti 反映了该素因子对于 n 的重要程度。现设定一个阈值 k，如果某个素因子 pi 对应的指数 ti **小于** k，则认为该素因子不重要，可以将 piti 项从 n 中除去；反之则将 piti 项保留。最终剩余项的乘积就是 n 简化后的值，如果**没有剩余项**则认为简化后的值等于 1。

试编写程序处理 q 个查询：

- 每个查询包含两个正整数 n 和 k，要求计算按上述方法将 n 简化后的值。

输入格式

从标准输入读入数据。

输入共 q+1 行。

输入第一行包含一个正整数 q，表示查询的个数。

接下来 q 行每行包含两个正整数 n 和 k，表示一个查询。

输出格式

输出到标准输出。

输出共 q 行。

每行输出一个正整数，表示对应查询的结果。

样例输入

```
3
2155895064 3
2 2
10000000000 10
```

样例输出

```
2238728
1
10000000000
```

样例解释

查询一：

- n=23×32×234×107
- 其中素因子 3 指数为 2，107 指数为 1。将这两项从 n 中除去后，剩余项的乘积为 23×234=2238728。

查询二：

- 所有项均被除去，输出 1。

查询三：

- 所有项均保留，将 n 原样输出。

子任务

40% 的测试数据满足：n≤1000；

80% 的测试数据满足：n≤10 5；

全部的测试数据满足：1<n≤10 10 且 1<k,q≤10。

题解

我有点惯性思维啦emm，就用最朴素的方法就能拿到大部分分数了

> 下面是朴素的素数筛法，这种朴素的素数筛怎么会想不到呢

```
#include <bits/stdc++.h>
using namespace std;

int main() {
  int k;
  scanf("%d", &k);
  while (k--) {
    long long a, b;
    cin >> a >> b;
    long long int x = a + b;
    int ans = 1;
    for (int i = 2; i * i <= x; i++) {
```

```
      if (x % i == 0) {
        int count = 0;
        while (x % i == 0) {
          count++;
          x /= i;
        }
        ans = ans * (count + 1);
      }
    }
    // 下面是获取到总因子的数量
    if (x != 1) printf("%d\n", ans * 2);
    else
      printf("%d\n", ans);
  }
}
```

代码如下：

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

using namespace std;

// 题目本身不难，难的是素因数分解板子
int main() {
  int q;
  cin >> q;
  while (q--) {
    long long int n;
    int k;
    cin >> n >> k;
    map<int, int> mp;
    for (int i = 2; i * i <= n; i++) {
      if (n % i == 0) {
        int cnt = 0;
        while (n % i == 0) {
          n /= i;
          cnt++;
        }
        mp[i] = cnt;
      }
    }
    if (n != 1) {
      mp[n] = 1;
    }
    // for (auto &it : mp) {
    //   cout << it.first << " " << it.second << endl;
    // }
    // cout << n << endl;
    // 为什么这里除不干净呀,需要为不是1的情况单独处理

    long long int ret = 1;
```

```
    for (auto &it : mp) {
      if (it.second >= k) {
        for (int i = 0; i < it.second; i++) {
          ret *= it.first;
        }
      }
    }
    cout << ret << endl;
  }
  return 0;
}
```

> 关于素数筛再去熟练一下
>
> 总想着曾经好像质数分解的方法，其实都是基于最简单的素数筛呢
>
> 前两题朴素啊朴素啊
>
> 这题本身也要剥离下素数筛，不然难度臆想啦

202309

T1

试题编号：	202309-1
试题名称：	坐标变换（其一）
时间限制：	1.0s
内存限制：	512.0MB
题目类型：	模拟，前缀和

问题描述

对于平面直角坐标系上的坐标 (x,y)，小 P 定义了一个包含 n 个操作的序列 T=(t1,t2,⋯,tn)。其中每个操作 ti（1≤i≤n）包含两个参数 dxi 和 dyi，表示将坐标 (x,y) 平移至 (x+dxi,y+dyi) 处。

现给定 m 个初始坐标，试计算对每个坐标 (xj,yj)（1≤j≤m）依次进行 T 中 n 个操作后的最终坐标。

输入格式

从标准输入读入数据。

输入共 n+m+1 行。

输入的第一行包含空格分隔的两个正整数 n 和 m，分别表示操作和初始坐标个数。

接下来 n 行依次输入 n 个操作，其中第 i（1≤i≤n）行包含空格分隔的两个整数 dxi、dyi。

接下来 m 行依次输入 m 个坐标，其中第 j（1≤j≤m）行包含空格分隔的两个整数 xj、yj。

输出格式

输出到标准输出中。

输出共 m 行，其中第 j（1≤j≤m）行包含空格分隔的两个整数，表示初始坐标 (xj,yj) 经过 n 个操作后的位置。

样例输入

```
3 2
10 10
0 0
10 -20
1 -1
0 0
```

样例输出

```
21 -11
20 -10
```

样例说明

第一个坐标 (1,−1) 经过三次操作后变为 (21,−11)；第二个坐标 (0,0) 经过三次操作后变为 (20,−10)。

评测用例规模与约定

全部的测试数据满足：n,m≤100，所有输入数据（x,y,dx,dy）均为整数且绝对值不超过 100000。

60分代码：

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

using namespace std;

int main() {
  int n, m;
  cin >> n >> m;
  // 由于行数是对应的所以不能使用map
  vector<vector<int> > a(1, vector<int>(2));
  vector<vector<int> > b(m + 1, vector<int>(2));
  for (int i = 1; i <= n; i++) {
    int dx, dy;
    cin >> dx >> dy;
    a[0][0] += dx;
    a[0][1] += dy;
  }
  for (int i = 1; i <= m; i++) {
    int x, y;
```

```
    cin >> x >> y;
    b[i][0] = x;
    b[i][1] = y;
  }
  for (int i = 1; i <= m; i++) {
    if (i > n) break;
    b[i][0] += a[0][0];
    b[i][1] += a[0][1];
  }
  // cout << endl << "-----------";
  for (int i = 1; i <= m; i++) {
    cout << b[i][0] << " " << b[i][1] << endl;
  }
}
```

正解代码：

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

using namespace std;

int main() {
  int n, m;
  cin >> n >> m;
  // 由于行数是对应的所以不能使用map
  long long int a[1][2] = {0};
  vector<vector<long> > b(m + 1, vector<long>(2));
  for (int i = 1; i <= n; i++) {
    int dx, dy;
    cin >> dx >> dy;
    a[0][0] += dx;
    a[0][1] += dy;
  }
  for (int i = 1; i <= m; i++) {
    int x, y;
    cin >> x >> y;
    b[i][0] = x;
    b[i][1] = y;
  }
  for (int i = 1; i <= m; i++) {
    b[i][0] += a[0][0];
    b[i][1] += a[0][1];
  }
  // cout << endl << "-----------";
  for (int i = 1; i <= m; i++) {
    cout << b[i][0] << " " << b[i][1] << endl;
  }
}
```

> 问题应该不是出来long那里，是

```
if (i > n) break;
// 这句代码是在错误的理解下的产物
```

> 题目还是一定要看清，多去理解几遍，尤其是测试样例，都去看几遍，这样自己的模拟就不容易出错

T2

试题编号：	202309-2
试题名称：	坐标变换（其二）
时间限制：	2.0s
内存限制：	512.0MB
题目类型：	变换，模拟，前缀和，差分

问题描述

对于平面直角坐标系上的坐标 (x,y)，小 P 定义了如下两种操作：

1. 拉伸 k 倍：横坐标 x 变为 kx，纵坐标 y 变为 ky；
2. 旋转 θ：将坐标 (x,y) 绕坐标原点 (0,0) **逆时针**旋转 θ 弧度（$0\le\theta<2\pi$）。易知旋转后的横坐标为 $x\cos\theta-y\sin\theta$，纵坐标为 $x\sin\theta+y\cos\theta$。

设定好了包含 n 个操作的序列 (t1,t2,⋯,tn) 后，小 P 又定义了如下查询：

- `i j x y`：坐标 (x,y) 经过操作 ti,⋯,tj（$1\le i\le j\le n$）后的新坐标。

对于给定的操作序列，试计算 m 个查询的结果。

输入格式

从标准输入读入数据。

输入共 n+m+1 行。

输入的第一行包含空格分隔的两个正整数 n 和 m，分别表示操作和查询个数。

接下来 n 行依次输入 n 个操作，每行包含空格分隔的一个整数（操作类型）和一个实数（k 或 θ），形如 1 k（表示拉伸 k 倍）或 2 θ（表示旋转 θ）。

接下来 m 行依次输入 m 个查询，每行包含空格分隔的四个整数 i、j、x 和 y，含义如前文所述。

输出格式

输出到标准输出中。

输出共 m 行，每行包含空格分隔的两个实数，表示对应查询的结果。

样例输入

```
10 5
2 0.59
2 4.956
1 0.997
1 1.364
1 1.242
1 0.82
2 2.824
1 0.716
2 0.178
2 4.094
1 6 -953188 -946637
1 9 969538 848081
4 7 -114758 522223
1 9 -535079 601597
8 8 159430 -511187
```

样例输出

```
-1858706.758 -83259.993
-1261428.46 201113.678
-75099.123 -738950.159
-119179.897 -789457.532
114151.88 -366009.892
```

样例说明

第五个查询仅对输入坐标使用了操作八：拉伸 0.716 倍。

横坐标：159430×0.716=114151.88

纵坐标：−511187×0.716=−366009.892

由于具体计算方式不同，程序输出结果可能与真实值有微小差异，样例输出仅保留了三位小数。

评测用例规模与约定

80% 的测试数据满足：n,m≤1000；

全部的测试数据满足：

- n,m≤100000；
- 输入的坐标均为整数且绝对值不超过 1000000；
- 单个拉伸操作的系数 k∈[0.5,2]；
- 任意操作区间 ti,⋯,tj（1≤i≤j≤n）内拉伸系数 k 的乘积在 [0.001,1000] 范围内。

评分方式

如果你输出的浮点数与参考结果相比，满足绝对误差不大于 0.1，则该测试点满分，否则不得分。

提示

- C/C++：建议使用 `double` 类型存储浮点数，并使用 `scanf("%lf", &x);` 进行输入，`printf("%f", x);` 输出，也可以使用 `cin` 和 `cout` 输入输出浮点数；`#include <math.h>` 后可使用三角函数 `cos()` 和 `sin()`。
- Python：直接使用 `print(x)` 即可输出浮点数 `x`；`from math import cos, sin` 后可使用相应三角函数。
- Java：建议使用 `double` 类型存储浮点数，可以使用 `System.out.print(x);` 进行输出；可使用 `Math.cos()` 和 `Math.sin()` 调用三角函数。

代码

> 80分超时代码如下：

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

using namespace std;

void func1(double k, double &x, double &y) {
  x = x * k;
  y = y * k;
}

void func2(double k, double &x, double &y) {
  double tmp = x;
  x = x * cos(k) - y * sin(k);
  y = tmp * sin(k) + y * cos(k);
  // 这种细节错误不要犯噢
}

int main() {
  ios::sync_with_stdio(false);
  cin.tie(0);
  cout.tie(0);
  // 加速cin/cout
  int n, m;
  cin >> n >> m;
  vector<int> n1(n + 1);
  vector<double> n2(n + 1);
  // 初始化n数组
  for (int i = 1; i <= n; i++) {
    int x;
    double y;
    cin >> x >> y;
    n1[i] = x;
    n2[i] = y;
  }
  // 导入数据：忽略第0空间

  vector<vector<double> > ret(m + 1, vector<double>(2));
```

```
  // 牢记这种初始化方式
  for (int k = 1; k <= m; k++) {
    int i, j;
    double x, y;
    cin >> i >> j;
    cin >> x >> y;
    for (int l = i; l <= j; l++) {
      if (n1[l] == 1) {
        func1(n2[l], x, y);
      } else {
        func2(n2[l], x, y);
      }
    }
    // ret[k][0] = x;
    // ret[k][1] = y;
    printf("%.3f %.3f\n", x, y);
    // 依旧运行超时?
  }
  // for (int i = 1; i <= m; i++) {
  //   printf("%.3f %.3f\n", ret[i][0], ret[i][1]);
  // }
  return 0;
}
```

> 遍历了多层for循环导致超时，如何优化时间呢?
>
> 从i到j到操作应该可以用1来模拟
>
> 输入参数直接乘积对应输出结果就行

优化代码

> 优化不了啊，由于旋转变换导致无法状态继承之前的积
>
> 故参考下题解
>
> 分别继承k和角度，最后进行一次变换即可
>
> 下面是参考题解的满分代码

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

using namespace std;

void func1(double k, double &x, double &y) {
  x = x * k;
  y = y * k;
}

void func2(double k, double &x, double &y) {
  double tmp = x;
  x = x * cos(k) - y * sin(k);
  y = tmp * sin(k) + y * cos(k);
```

```
}

int main() {
  ios::sync_with_stdio(false);
  cin.tie(0);
  cout.tie(0);
  int n, m;
  cin >> n >> m;
  vector<vector<double> > op(n + 1, vector<double>(2));
  op[0][0] = 1;
  op[0][1] = 0;
  for (int kk = 1; kk <= n; kk++) {
    int opp;
    double k;
    cin >> opp >> k;
    if (opp == 1) {
      op[kk][0] = k;
      op[kk][1] = 0;
    } else {
      op[kk][0] = 1;
      op[kk][1] = k;
    }
  }
  for (int i = 1; i <= n; ++i) {
    op[i][0] *= op[i - 1][0];  // 乘积的倍数k累积
    op[i][1] += op[i - 1][1];  // 角度累积
  }

  for (int k = 1; k <= m; k++) {
    int i, j;
    double x, y;
    cin >> i >> j;
    cin >> x >> y;
    func1(op[j][0] / op[i - 1][0], x, y);
    func2(op[j][1] - op[i - 1][1], x, y);
    printf("%.3f %.3f\n", x, y);
  }
}
```

> 我是想着继承结果，原来继承参数就好了啊！

202305

T1

试题编号：	**202305-1**
试题名称：	重复局面
时间限制：	1.0s
内存限制：	512.0MB
问题分类：	枚举，字符串

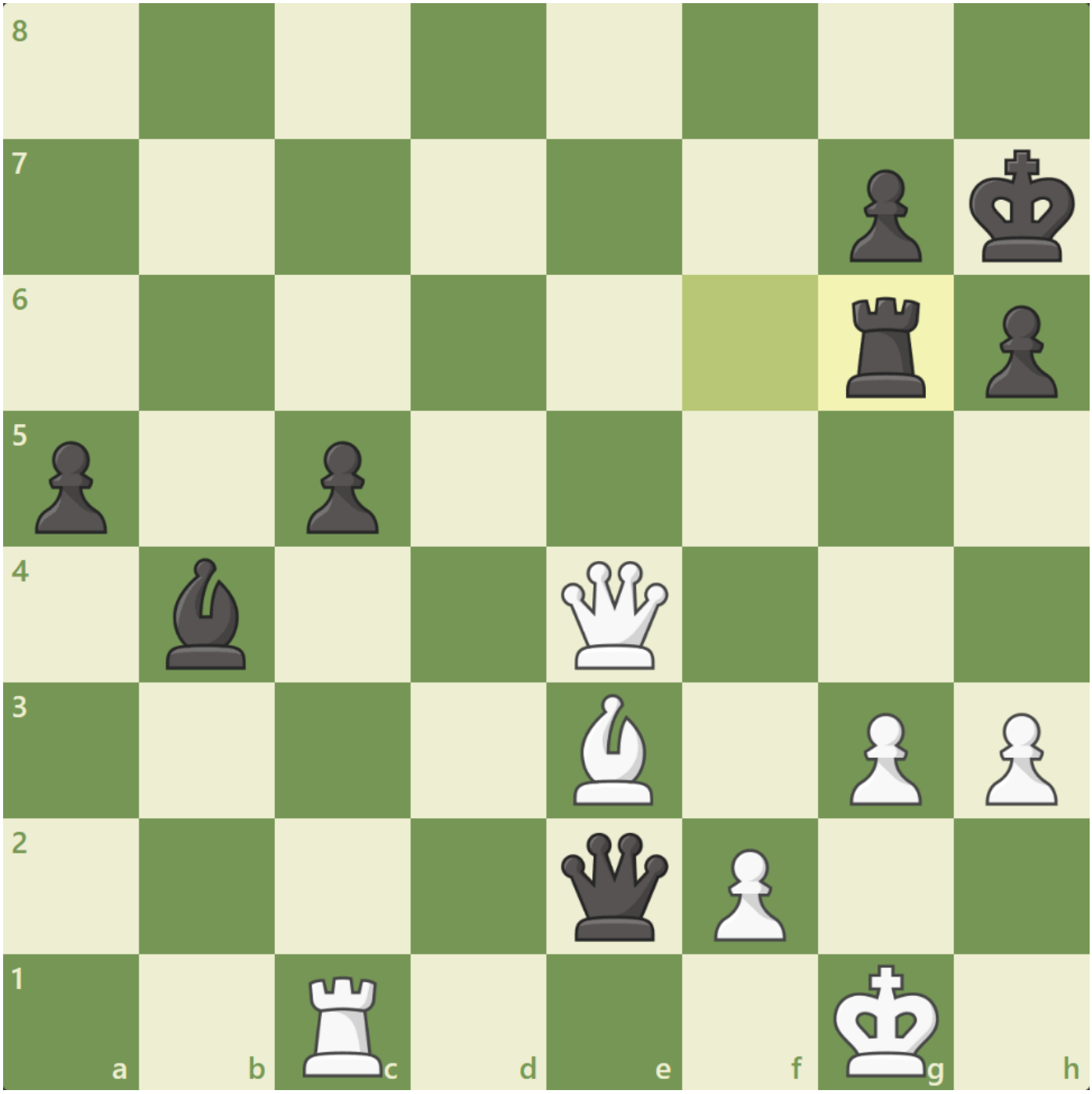
题目背景

国际象棋在对局时，同一局面连续或间断出现3次或3次以上，可由任意一方提出和棋。

问题描述

国际象棋每一个局面可以用大小为 8×8 的字符数组来表示，其中每一位对应棋盘上的一个格子。六种棋子王、后、车、象、马、兵分别用字母 `k`、`q`、`r`、`b`、`n`、`p` 表示，其中大写字母对应白方、小写字母对应黑方。棋盘上无棋子处用字符 `*` 表示。两个字符数组的每一位均相同则说明对应同一局面。

现已按上述方式整理好了每步棋后的局面，试统计每个局面分别是第几次出现。



输入格式

从标准输入读入数据。

输入的第一行包含一个正整数 n，表示这盘棋总共有 n 步。

接下来 8×n 行，依次输入第 1 到第 n 步棋后的局面。具体来说每行包含一个长度为 8 的字符串，每 8 行字符串共 64 个字符对应一个局面。

输出格式

输出到标准输出中。

输出共 n 行，每行一个整数，表示该局面是第几次出现。

样例输入

```
8
********
******pk
*****r*p
p*pQ****
********
**b*B*PP
****qP**
**R***K*
********
******pk
*****r*p
p*pQ****
*b******
****B*PP
****qP**
**R***K*
********
******pk
*****r*p
p*p*****
*b**Q***
****B*PP
****qP**
**R***K*
******k*
******p*
*****r*p
p*p*****
*b**Q***
****B*PP
****qP**
**R***K*
******k*
******p*
*****r*p
p*pQ****
*b******
****B*PP
****qP**
**R***K*
********
******pk
*****r*p
p*pQ****
*b******
****B*PP
****qP**
**R***K*
```

```
********
******pk
*****r*p
p*p*****
*b**Q***
****B*PP
****qP**
**R***K*
********
******pk
******rp
p*p*****
*b**Q***
****B*PP
****qP**
**R***K*
```

样例输出

```
1
1
1
1
1
2
2
1
```

样例说明

第 6、7 步后的局面分别与第 2、3 步后的局面相同。第 8 步后的局面与上图相对应。

子任务

输入数据满足 n≤100。

提示

判断重复局面仅涉及字符串比较，无需考虑国际象棋实际行棋规则。

代码

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

using namespace std;

// 将8行矩阵作为一个字符串处理即可
```

```cpp
int main() {
  // ios::sync_with_stdio(false);
  // cin.tie(0);
  // cout.tie(0);
  int n;
  cin >> n;
  getchar();
  // 向这里数字转字符务必使用getchar()，否则会出现错误
  string a[n];
  map<int, int> mp;
  int cnt = 0;
  int tmp = n;
  while (n--) {
    string s;
    for (int i = 0; i < 8; i++) {
      string t;
      getline(cin, t);
      s += t;
    }
    a[cnt++] = s;
  }
  for (int i = 0; i < tmp; i++) {
    mp[i] = 0;
    for (int j = 0; j <= i; j++) {
      if (a[i] == a[j]) {
        mp[i]++;
      }
    }
  }
  for (auto &i : mp) {
    cout << i.second << endl;
  }
  return 0;
}
// 暴力枚举检索，怎么会出错呢？从原理上来说不应该啊？
```

> 代码没有问题，理所当然是满分，稍微注意下判题系统，似乎是头文件的问题，待进一步慢慢调试发现，注意看上面的注释掉的代码
>
> 测试发现万能头文件没问题，问题出在 解除绑定 cin和cout上，看来没事不用用那个，除非题目分数给TLE了

T2

试题编号：	202305-2
试题名称：	矩阵运算
时间限制：	5.0s
内存限制：	512.0MB
题目类型：	模拟，循环

题目背景

Softmax(Q×KTd)×V 是 Transformer 中注意力模块的核心算式，其中 Q、K 和 V 均是 n 行 d 列的矩阵，KT 表示矩阵 K 的转置，× 表示矩阵乘法。

问题描述

为了方便计算，顿顿同学将 Softmax 简化为了点乘一个大小为 n 的一维向量 W：
`(W·(Q×K^T))×V`
点乘即对应位相乘，记 W(i) 为向量 W 的第 i 个元素，即将 (Q×KT) 第 i 行中的每个元素都与 W(i) 相乘。

现给出矩阵 Q、K 和 V 和向量 W，试计算顿顿按简化的算式计算的结果。

输入格式

从标准输入读入数据。

输入的第一行包含空格分隔的两个正整数 n 和 d，表示矩阵的大小。

接下来依次输入矩阵 Q、K 和 V。每个矩阵输入 n 行，每行包含空格分隔的 d 个整数，其中第 i 行的第 j 个数对应矩阵的第 i 行、第 j 列。

最后一行输入 n 个整数，表示向量 W。

输出格式

输出到标准输出中。

输出共 n 行，每行包含空格分隔的 d 个整数，表示计算的结果。

样例输入

```
3 2
1 2
3 4
5 6
10 10
-20 -20
30 30
6 5
4 3
2 1
4 0 -5
```

样例输出

```
480 240
0 0
-2200 -1100
```

子任务

70 的测试数据满足：n≤100 且 d≤10；输入矩阵、向量中的元素均为整数，且绝对值均不超过 30。

全部的测试数据满足：n≤104 且 d≤20；输入矩阵、向量中的元素均为整数，且绝对值均不超过 1000。

提示

请谨慎评估矩阵乘法运算后的数值范围，并使用适当数据类型存储矩阵中的整数。

> 模拟运算即可，本质上没有难度

代码

> 70分超时代码如下

```cpp
#include <iostream>
#include <vector>

using namespace std;

void getMatrix(vector<vector<long long int>> &matrix, int n, int d) {
  for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= d; j++) {
      cin >> matrix[i][j];
    }
  }
}

void getTMatrix(vector<vector<long long int>> &from,
                vector<vector<long long int>> &to, int n, int d) {
  for (int i = 1; i <= n; i++) {
```

```
    for (int j = 1; j <= d; j++) {
      to[j][i] = from[i][j];
    }
  }
}

int main() {
  long long int n, d;
  cin >> n >> d;
  vector<vector<long long int>> Q(n + 1, vector<long long int>(d + 1));
  vector<vector<long long int>> K(n + 1, vector<long long int>(d + 1));
  vector<vector<long long int>> V(n + 1, vector<long long int>(d + 1));
  getMatrix(Q, n, d);
  getMatrix(K, n, d);
  getMatrix(V, n, d);
  // 获取三个初始矩阵，这里应该不会超时吧

  vector<vector<long long int>> KT(d + 1, vector<long long int>(n + 1));
  getTMatrix(K, KT, n, d);
  // 获得转置矩阵

  vector<int> M(n + 1, 0);
  for (int i = 1; i <= n; i++) {
    cin >> M[i];
  }

  vector<vector<long long int>> QKT(n + 1, vector<long long int>(n + 1, 0));
  for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) {
      for (int k = 1; k <= d; k++) {
        QKT[i][j] += Q[i][k] * KT[k][j];
      }
    }
  }
  // n*d的矩阵和d*n的矩阵乘积，结果是n*n的矩阵
  // 举证乘积要求：：n d d n 中间两位相同即可

  for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) {
      QKT[i][j] = QKT[i][j] * M[i];
    }
  }
  // 点乘运算

  // 计算 QKT 和 V 的乘积
  vector<vector<long long int>> QKTV(n + 1, vector<long long int>(d + 1, 0));
  for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= d; j++) {
      for (int k = 1; k <= n; k++) {
        QKTV[i][j] += QKT[i][k] * V[k][j];
      }
    }
```

```
  }
  // n*n的矩阵和n*d的矩阵，结果是n*d的矩阵

  // 输出 QKTV 矩阵
  for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= d; j++) {
      cout << QKTV[i][j] << " ";
    }
    cout << endl;
  }

  return 0;
}
```

> 矩阵运算：
>
> 1. 打草稿
> 2. 核心明确结果矩阵的行和列
> 3. i和j控制结果矩阵的填值，k分别控制列和行的改变
> 4. 几乎所有矩阵的乘积都满足第三点

优化代码

> 优化失败：依旧是70分超时

```
#include <iostream>
#include <vector>

using namespace std;

void getMatrix(vector<vector<long long int>> &matrix, int n, int d) {
  for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= d; j++) {
      cin >> matrix[i][j];
    }
  }
}

void getTMatrix(vector<vector<long long int>> &from,
                vector<vector<long long int>> &to, int n, int d) {
  for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= d; j++) {
      to[j][i] = from[i][j];
    }
  }
}

int main() {
  ios::sync_with_stdio(false);
  cin.tie(0);
```

```
    cout.tie(0);
    long long int n, d;
    cin >> n >> d;
    vector<vector<long long int>> Q(n + 1, vector<long long int>(d + 1));
    vector<vector<long long int>> K(n + 1, vector<long long int>(d + 1));
    vector<vector<long long int>> V(n + 1, vector<long long int>(d + 1));
    getMatrix(Q, n, d);
    getMatrix(K, n, d);
    getMatrix(V, n, d);

    vector<vector<long long int>> KT(d + 1, vector<long long int>(n + 1));
    getTMatrix(K, KT, n, d);

    vector<int> M(n + 1, 0);
    for (int i = 1; i <= n; i++) {
      cin >> M[i];
    }

    vector<vector<long long int>> QKT(n + 1, vector<long long int>(n + 1, 0));
    for (int i = 1; i <= n; i++) {
      for (int j = 1; j <= n; j++) {
        for (int k = 1; k <= d; k++) {
          QKT[i][j] += Q[i][k] * KT[k][j];
        }
      }
    }

    for (int i = 1; i <= n; i++) {
      for (int j = 1; j <= n; j++) {
        QKT[i][j] = QKT[i][j] * M[i];
      }
    }

    // 计算 QKT 和 V 的乘积
    vector<vector<long long int>> QKTV(n + 1, vector<long long int>(d + 1, 0));
    for (int i = 1; i <= n; i++) {
      for (int j = 1; j <= d; j++) {
        for (int k = 1; k <= n; k++) {
          QKTV[i][j] += QKT[i][k] * V[k][j];
        }
        cout << QKTV[i][j] << " ";
      }
      cout << endl;
    }

    return 0;
}
```

优化代码

> 优化失败：依旧超时，三个代码均为70分，没啥太大差异

```cpp
#include <iostream>
#include <vector>

using namespace std;

void getMatrix(vector<vector<long long int>> &matrix, int n, int d) {
  for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= d; j++) {
      cin >> matrix[i][j];
    }
  }
}

int main() {
  ios::sync_with_stdio(false);
  cin.tie(0);
  cout.tie(0);

  long long int n, d;
  cin >> n >> d;

  vector<vector<long long int>> Q(n + 1, vector<long long int>(d + 1));
  vector<vector<long long int>> K(n + 1, vector<long long int>(d + 1));
  vector<vector<long long int>> V(n + 1, vector<long long int>(d + 1));

  getMatrix(Q, n, d);
  getMatrix(K, n, d);
  getMatrix(V, n, d);

  vector<int> M(n + 1, 0);
  for (int i = 1; i <= n; i++) {
    cin >> M[i];
  }

  // 计算 QKT 和 V 的乘积
  vector<vector<long long int>> QKTV(n + 1, vector<long long int>(d + 1, 0));
  for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= d; j++) {
      long long int sum = 0;
      for (int k = 1; k <= n; k++) {
        long long int qkt = 0;
        for (int l = 1; l <= d; l++) {
          qkt += Q[i][l] * K[k][l];
        }
        qkt *= M[i];
        sum += qkt * V[k][j];
      }
      QKTV[i][j] = sum;
      cout << QKTV[i][j] << " ";
    }
    cout << endl;
  }
```

```
    return 0;
}
```

202303

T1

试题编号：	202303-1
试题名称：	田地丈量
时间限制：	1.0s
内存限制：	512.0MB
题目类型：	暴力枚举，循环，矩形交集计算

问题描述

西西艾弗岛上散落着 n 块田地。每块田地可视为平面直角坐标系下的一块矩形区域，由左下角坐标 (x1,y1) 和右上角坐标 (x2,y2) 唯一确定，且满足 x1<x2、y1<y2。这 n 块田地中，任意两块的交集面积均为 0，仅边界处可能有所重叠。

最近，顿顿想要在南山脚下开垦出一块面积为 a×b 矩形田地，其左下角坐标为 (0,0)、右上角坐标为 (a,b)。试计算顿顿选定区域内已经存在的田地面积。

输入格式

从标准输入读入数据。

输入共 n+1 行。

输入的第一行包含空格分隔的三个正整数 n、a 和 b，分别表示西西艾弗岛上田地块数和顿顿选定区域的右上角坐标。

接下来 n 行，每行包含空格分隔的四个整数 x1、y1、x2 和 y2，表示一块田地的位置。

输出格式

输出到标准输出。

输出一个整数，表示顿顿选定区域内的田地面积。

样例输入

```
4 10 10
0 0 5 5
5 -2 15 3
8 8 15 15
-2 10 3 15
```

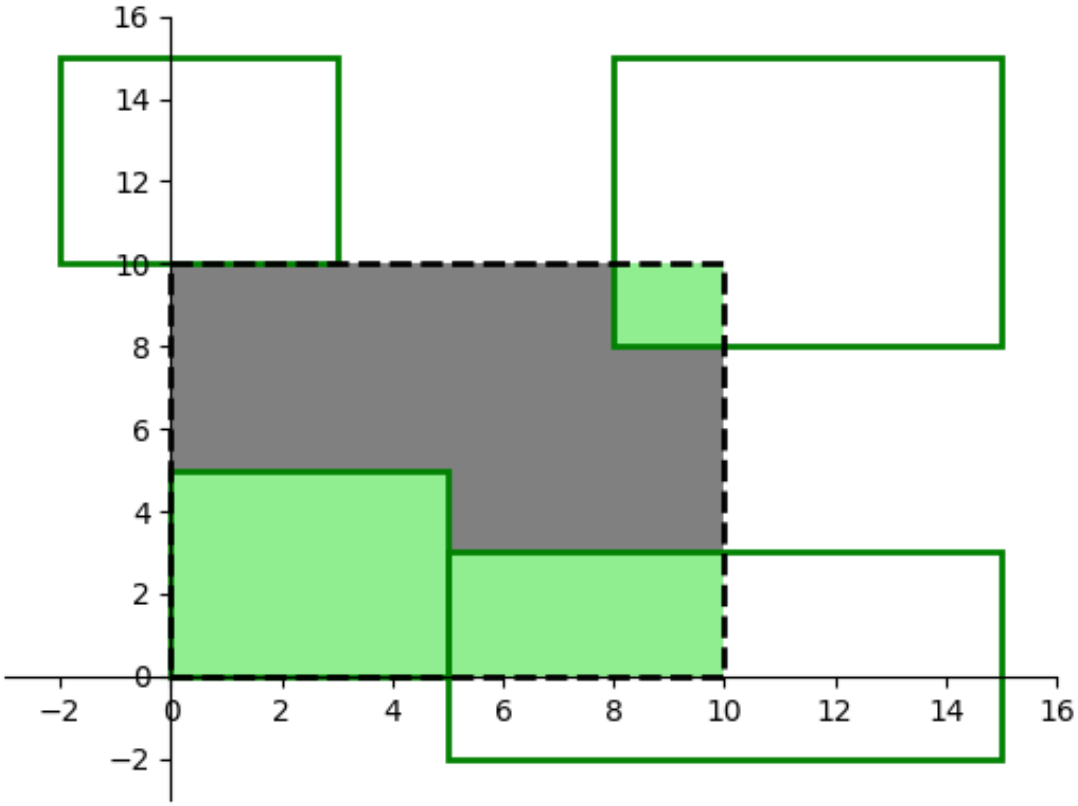
样例输出

```
44
```

样例解释

> 我能不能便利遍历每个矩阵的四个点，然后得到在矩形内的点？但是还有矩形外的点需要考虑，比如说跨过了整个矩形的情况

如图所示，选定区域内田地（绿色区域）面积为 44。



子任务

全部的测试数据满足 n≤100，且所有输入坐标的绝对值均不超过 104。

> 也就是已知基本坐标和干扰坐标，求完全未相交的面积

> 干扰坐标的交集为0
>
> 多读几遍题目，最后的输出结果是交集，绿色区域
>
> 那么既然干扰坐标面积的交集为0，那么枚举计算不就行了，都不需要考虑交集部分，暴力循环每个干扰坐标即可

代码

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

using namespace std;

int getArea(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4) {
  int a1 = min(max(x1, x2), max(x3, x4));
  int a2 = min(max(y1, y2), max(y3, y4));
  int b1 = max(min(x1, x2), min(x3, x4));
  int b2 = max(min(y1, y2), min(y3, y4));
  if (a1 > b1 && a2 > b2) {
    return (a1 - b1) * (a2 - b2);
  }
  return 0;
}

int main() {
  int n, x1, y1;
  cin >> n >> x1 >> y1;
  int ret = 0;
  while (n--) {
    int x3, y3, x4, y4;
    cin >> x3 >> y3 >> x4 >> y4;
    ret += getArea(0, 0, x1, y1, x3, y3, x4, y4);
  }
  cout << ret << endl;
  return 0;
}
```

> 矩形的交集面积可以收录了，写起来发现没那么简单
>
> 我不好评价，google搜索的，蓝桥杯基础练习题，但凡懂这个算法就没丁点难度，但凡不懂的话考场多半写不出来。
>
> 算法我曾经见过，但是没有收录，“我爱笔记”！！

算法步骤

1. **计算相交区域的右上角坐标**：

   int a1 = min(max(x1, x2), max(x3, x4));

   int a2 = min(max(y1, y2), max(y3, y4));

   - `a1` 是两个矩形右边界的最小值。

- `a2` 是两个矩形上边界的最小值。

2. **计算相交区域的左下角坐标**：

   int b1 = max(min(x1, x2), min(x3, x4));

   int b2 = max(min(y1, y2), min(y3, y4));

   - `b1` 是两个矩形左边界的最大值。
   - `b2` 是两个矩形下边界的最大值。

3. **判断是否相交并计算面积**：

   if (a1 > b1 && a2 > b2) {

    return (a1 - b1) * (a2 - b2);

   }

   return 0;

   - 如果 `a1 > b1` 且 `a2 > b2`，则两个矩形相交，返回相交区域的面积 `(a1 - b1) * (a2 - b2)`。
   - 否则，返回 0，表示两个矩形不相交。
   - 口诀：右上角两大取小，左下角两小取大

> 也就是说直接按照相交计算得到相交区域左下角和右上角的坐标
>
> 最小的最大 最大的最小 emm 直接看图看能不能看懂

T2

试题编号：	**202303-2**
试题名称：	垦田计划
时间限制：	1.0s
内存限制：	512.0MB
题目类型：	贪心，规划，决策

问题描述

顿顿总共选中了 n 块区域准备开垦田地，由于各块区域大小不一，开垦所需时间也不尽相同。据估算，其中第 i 块（1≤i≤n）区域的开垦耗时为 ti 天。这 n 块区域可以同时开垦，所以总耗时 tTotal 取决于耗时最长的区域，即：t(Total)=max{t1,t2,⋯,tn}

为了加快开垦进度，顿顿准备在部分区域投入额外资源来缩短开垦时间。具体来说：

- 在第 i 块区域每投入 ci 单位资源，便可将其开垦耗时缩短 1 天；
- 耗时缩短天数以整数记，即第 i 块区域投入资源数量必须是 ci 的整数倍；
- 在第 i 块区域最多可投入 ci×(ti−k) 单位资源，将其开垦耗时缩短为 k 天；
- 这里的 k 表示开垦一块区域的最少天数，满足 0<k≤min{t1,t2,⋯,tn}；换言之，如果无限制地投入资源，所有区域都可以用 k 天完成开垦。

现在顿顿手中共有 m 单位资源可供使用，试计算开垦 n 块区域最少需要多少天?

输入格式

从标准输入读入数据。

输入共 n+1 行。

输入的第一行包含空格分隔的三个正整数 n、m 和 k，分别表示待开垦的区域总数、顿顿手上的资源数量和每块区域的最少开垦天数。

接下来 n 行，每行包含空格分隔的两个正整数 ti 和 ci，分别表示第 i 块区域开垦耗时和将耗时缩短 1 天所需资源数量。

输出格式

输出到标准输出。

输出一个整数，表示开垦 n 块区域的最少耗时。

样例输入1

```
4 9 2
6 1
5 1
6 2
7 1
```

样例输出1

```
5
```

样例解释

如下表所示，投入 5 单位资源即可将总耗时缩短至 5 天。此时顿顿手中还剩余 4 单位资源，但无论如何安排，也无法使总耗时进一步缩短。

i	基础耗时 ti	缩减 1 天所需资源 ci	投入资源数量	实际耗时
1	6	1	1	5
2	5	1	0	5
3	6	2	2	5
4	7	1	2	5

样例输入2

```
4 30 2
6 1
5 1
6 2
7 1
```

样例输出2

```
2
```

样例解释

投入 20 单位资源，恰好可将所有区域开垦耗时均缩短为 k=2 天；受限于 k，剩余的 10 单位资源无法使耗时进一步缩短。

子任务

70% 的测试数据满足：0<n,ti,ci≤100 且 0<m≤10^6；

全部的测试数据满足：0<n,ti,ci≤105 且 0<m≤10^9。

> 无比熟悉的题目，估计又是蓝桥杯基础练习题之一了

参考题解

思路讲解：

> 这道题也不难，使用标志数组记录耗时为i天的区域降低一天的总花费，然后从高向低降，最后就可以得出答案了。

代码

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

using namespace std;

int main() {
  int n, k;  // n块田地，最少时间
  long long int m;
  map<int, int> tim, res, flag;
  cin >> n >> m >> k;
  int max = 0;
  for (int i = 0; i < n; ++i) {
    cin >> tim[i] >> res[i];
    max = max > tim[i] ? max : tim[i];  // 最大天数
    flag[tim[i]] += res[i];
    // 把耗时相同的作为一组
  }
  // 那么现在就是有组，有消耗，有资源
  for (int i = max; i > 0; i--) {
```

```
    if (max == k) break;
    if (m > flag[i]) {
      m -= flag[i];
      flag[i - 1] += flag[i];  // 高天的组减去一天就并归到低天的组
      max--;  // 最大天数减一天
    } else
      break;
  }
  cout << max;
  // cout<<endl<<flag[100]<<endl;  // 即使flag[100]不存在但是这样操作之后就开了一个空间
  // cout<<flag.size()<<endl;  // size为8，前面的max遍历加上上面这个新开的空间
  return 0;
}
```

> 这里学下组别并归的思维，将相同特征或者相似特征的组别并为一组，将对应需要消耗的资源累加即可

202212

T1

T2