NOIP 普及组复赛

```
include<bits/stdc++.h>//1-2986    解密 [CSP-J 2022]    wintereta
using namespace std;
long long n,d,e,t;//int 会爆掉
int main()
{
	scanf("%d",&t);
	while(t--)
	{// 多组数据
		scanf("%lld%lld%lld",&n,&d,&e);
		long long delta=(n-e*d+2)*(n-e*d+2)-4*n;
		if(delta<0)
		{		// 特判解为虚数的情况
			printf("NO\n");
			continue;
		}
		long long p=1.0*((n-e*d+2)-sqrt(delta))/2,q=1.0*((n-e*d+2)+sqrt(delta))/2;// 注意要乘上 1.0

		if(p>0&&q>0&&p*q==n&&e*d==(p-1)*(q-1)+1)
			printf("%lld %lld\n",p,q);// 由于精度原因，需要再次判断题目条件
		else printf("NO\n");
	}

	return 0;
}
```

```
include<bits/stdc++.h>//2-2223   csp-j2021   插入排序（sort） wintereta
using namespace std;
int n,q,a[8005],b[8005],num,x,v,ans;
int main()
{

	scanf("%d%d",&n,&q);
	for(int i=1;i<=n;i++)
	{
		scanf("%d",&a[i]);
	}
	for(int i=1;i<=n;i++)
	{
		for(int j=1;j<=i;j++)
		{
			if(i==j||a[j]<a[i]||a[j]==a[i])	b[i]++;
			else		b[j]++;
		}// 这里的 b[i] 就是第 i 个数在数组 a[i] 中的排位
	}
```

```
	for(int i=0;i<q;i++)
	{
		scanf("%d",&num);
		if(num==1)
		{
			scanf("%d%d",&x,&v);
			for(int i=1;i<=n;i++)
			{
				if(i==x)	continue;
				if((a[i]<a[x]||a[i]==a[x]&&i<x)&&(a[i]>v||a[i]==v&&i>x))
				{
					b[i]++;b[x]--;
				}
			else if((a[i]>a[x]||a[i]==a[x]&&i>x)&&(a[i]<v||a[i]==v&&i<x))
				{
					b[i]--;b[x]++;
				}
			}
			a[x]=v;
		}
		if(num==2)
		{
			scanf("%d",&x);
			printf("%d\n",b[x]);
		}
	}
	return 0;
}
```

本质上来说，本题是进制转换，但不允许拆分出来的 2 进制的最低位为 1。由于 2 进制的最低位为 1，得到的 10 进制是奇数，因此奇数不可能有符合条件的拆分。

举例能拆分的正偶数，比如：10，拆分 2 进制后是 1010，那么这个拆分其实是值为 1 的这些 2 进制位对应的十进制，也就是：8 2。

```
#include <bits/stdc++.h>//3-2157-1    csp-j2020    优秀的拆分（power）    javacn
using namespace std;
int a[100];
int k = 0;//a 数组的下标
int n;
int main()
{
	cin>>n;
	if(n%2==1)  // 奇数不可能拆分
	{
		cout<<-1;
		return 0;
	}
	// 将 n 转 2 进制，同时存储每一位 2 进制转 10 进制的结果（2 的次方）
	int t = 1;
	while(n != 0)
	{
		if(n % 2 == 1)     // 存储转 2 进制后，值为 1 的位数对应的 10 进制
		{
			k++;
			a[k] = t;
		}
		n = n / 2;
		t = t * 2;//2 的次方
	}
	for(int i = k; i >= 1; i--)    // 输出分解的结果
	{
		cout<<a[i]<<" ";
	}
	return 0;
}
```

```cpp
#include<iostream>//3-2157-2  csp-j2020  优秀的拆分（power）  anselxu
#include<algorithm>
#include<iomanip>
using namespace std;
int x=1,w=0,a[30],ans[30],n;// 小于 1e8 的 2 次幂 大约 25 个，可以用程序测试得出
/*
思想：2 的幂由小到大排序，搜索该表进行拆分
*/
void dfs(int x,int t)
{//x 是拆分的数，t 为了减少重复查找
	if(x==0)
	{// 满足条件输出并结束
		for(int i=n;i>=1;i--)
		{
			cout<<ans[i]<<' ';
		}
		exit(1);
	}
	for(int i=t;a[i]<=x;i++)
	{
		if(x-a[i]<0)
			break;// 剪枝，因为 a 是升序，减数就是升序，后面没必要判断了
		// 如果拆分后不小于 0 就进行拆分
		ans[++n]=a[i];// 记录当前数
		dfs(x-a[i],i+1);// 进入下一步拆分
		n--;// 回到前一状态

	}
}
```

```
int main()
{
	ios::sync&#95;with&#95;stdio(false);
	cin.tie(0);
	int p;
	cin>>p;
	for(int i=1;x<=1e8;i++)
	{
		x=x*2;
		a[++w]=x;
	}// 存储不超过数据范围的所有 2 的幂
	dfs(p,1);
	cout<<-1;
	return 0;
}
```

本题是求，在有 i 人的情况下，前 w% 人的最低分数线是多少。

本题要注意，数据比较多，因此要注意防止超时：

1. 用 scanf 和 printf 提升读写效率（也可以快读快写）
2. 用数组计数法，统计有 i 个人的情况下，每个分数有多少人（本题分数最多是 600），从大到小数出前 w% 人，就知道分数线是多少了。

```
#include <bits/stdc++.h>//4-2158   csp-j2020   直播获奖（live）   javacn
using namespace std;
int n,a[800];
int w;// 获奖率
int x;// 获奖人数
int t,c = 0;
int main()
{
	scanf("%d%d",&n,&w);
	for(int i = 1;i <= n;i++)
	{
		scanf("%d",&t);
		a[t]++;
		x = max(1,(int)(i*w/100.0));// 获奖人数

		c = 0;// 清零
		// 寻找 x 人获奖的分数线
		for(int j = 600;j >= 0;j--)
		{
			if(c + a[j] >= x)
			{
				printf("%d ",j);
				break;
			}
			else		c += a[j];
		}
	}
	return 0;
}
```

本题要点：

1. 后缀表达式对应于一棵表达式树

2. 由于数据量的限制，不能每次都将整个表达式的树都计算一遍

3. 优化点：记录哪些点的值变化会导致结果的变化

思路：

1. 求表达式的值

2. 深搜求哪些点的值可能会影响表达式的值

3.q 次询问，每次判断当前结点的修改是否影响表达式的值

本题可以在建树的同时，计算出表达式的值，然后只需要 dfs 计算出哪些结点的值修改后会影响结果，就可以。

```cpp
#include <bits/stdc++.h>//5-2159    csp-j2020    表达式（expr）   javacn
using namespace std;

/*
取反某一个操作数的值时，原表达式的值为多少
思路：
1. 读入表达式，将表达式建成二叉树
2. 计算出表达式的值
3. 深搜，求出哪些结点的值修改后对结果有影响
*/
const int N = 1e6 + 10;
struct node{
    int to,next;
}a[N];// 邻接表
int pre[N],k;// 存储以每个点为起点的最后一条边的编号
int num[N];// 存储二叉树每个结点的值
char c[N];// 存储操作符
int m;// 表示 c 数组的下标
bool f[N];// 标记哪些点的值修改后对结果有影响
int n;
string s,w;//w 用来存储每个操作数的下标
stack<int> st;// 用于计算后缀表达式

// 建边
void add(int u,int v)
{
      k++;
      a[k].to = v;
      a[k].next = pre[u];
      pre[u] = k;
}

// 深搜求哪些结点的值修改后对结果有影响
```

```
//深搜求哪些结点的值修改后对结果有影响
void dfs(int x)
{
	f[x] = true;
	if(x <= n)		return;//如果是叶子结点
	if(c[x] == '!')			dfs(a[pre[x]].to);
	else
	{
		int n1 = a[pre[x]].to,n2 = a[a[pre[x]].next].to;
		if(c[x] == '&')
		{
			if(num[n1] == 1)		dfs(n2);
			if(num[n2] == 1)		dfs(n1);
		}
		else if(c[x] == '|')
		{
			if(num[n1] == 0)		dfs(n2);
			if(num[n2] == 0)		dfs(n1);
		}
	}
}

int main()
{
	getline(cin,s);
	cin>>n;
	for(int i = 1;i <= n;i++)
	{
		scanf("%d",&num[i]);
	}
	int x,y;	//分析表达式，建树
	m = n;
```

```
for(int i = 0;i < s.size();i++)
{
	if(isdigit(s[i]))
	{
		w = w + s[i];
		if(i==s.size()-1||!isdigit(s[i+1]))		// 如果连续数字结束
		{	st.push(atoi(w.c_str()));// 将操作数的下标入栈
			w = "";
		}
	}
	else if(s[i] == '!')				// 如果是取反操作符
	{
		m++;
		c[m] = s[i];		// 操作符的编号是 m
		x = st.top();// 取栈顶元素来运算
		st.pop(); // 出栈
		add(m,x);				// 建边
		num[m] = !num[x];
		st.push(m);
	}
	else if(s[i] == '&' || s[i] == '|')
	{
		m++;// 如果是 & | 操作符
		c[m] = s[i];
		x = st.top();				// 取 2 个操作数
		st.pop();
		y = st.top();
		st.pop();
		add(m,x);			// 建边
		add(m,y);
		if(s[i] == '&')	num[m] = num[x] & num[y];// 计算结点的值
		else if(s[i] == '|')	num[m] = num[x] | num[y];
		st.push(m);
	}
}
```

```
        int ans = num[st.top()];
        //cout<<ans;
        // 深搜求哪些结点的值修改后对结果有影响
        dfs(st.top());// 从根开始深搜

        //q 次询问
        int q;
        cin>>q;
        while(q--)
        {
                scanf("%d",&x);
                if(f[x]==true)          printf("%d\n",!ans);
                else  printf("%d\n",ans);
        }

        return 0;
}
```

相关算法：DP

由于本题只能向上、下、右走，且走过的点不允许重复走，因此对于某一列而言，如果向下走了，就不能再向上走，向上走了，就不能再向下走。

那么我们只需要把向上、向下两种情况都尝试一遍，对于每个点而言，如果处于向下的状态，那么走到该点的最大值 = max(走到左侧最大值，走到上方最大值) + 该点的值。向上同理。

```
#include <bits/stdc++.h>//6-2160    csp-j2020    方格取数（number）    javacn
using namespace std;
/*
左上角走到右下角
每一步只能向上、向下或向右走一格
不能重复经过已经走过的方格

计算的结果，要用 long long
*/
const int N = 1010;
int a[N][N];// 读入的每个点的值
typedef long long LL;
LL f[N][N];
int n,m;
```

```
int main()
{
	scanf("%d%d",&n,&m);
	for(int i = 1;i <= n;i++)
	{
		for(int j = 1;j <= m;j++)
		{
			scanf("%d",&a[i][j]);
		}
	}
    /*
    DP 求出走到每个点经过的数字和的最大值
    一列一列求，对于每一列而言，要求从上向下走到每个点经过的数字和的最大值
    也要求从下向上走到每个点经过的数字和的最大值     */
	memset(f,-0x7f,sizeof(f));
	LL ma;// 走到每个格子经过的数字和最大
	f[1][0] = 0;
	for(int j = 1;j <= m;j++)
	{
		// 从上向下求
		ma = -1e18;
		for(int i = 1;i <= n;i++)
		{
			ma = max(ma,f[i][j-1]) + a[i][j];
			f[i][j] = max(f[i][j],ma);
		}
		ma = -1e18;				// 从下向上求
		for(int i = n;i >= 1;i--)
		{
			ma = max(ma,f[i][j-1]) + a[i][j];
			f[i][j] = max(f[i][j],ma);
		}
	}
	cout<<f[n][m];
}
```

遍历每个字符，统计字符 '1' 出现的数量。

```
#include<bits/stdc++.h>//7-1934 -1   csp-j2019   数字游戏（number） javacn
using namespace std;

/*
 遍历每个字符，判断如果是字符’1’则计数
*/
int main()
{
	string s;
	cin>>s;
	int c = 0;
	for(int i = 0;i < s.size();i++)
	{
		if(s[i] == '1') c++;
	}
	cout<<c;
	return 0;
}
```

本题也可以做得简洁一些，读入 8 个字符，逐个判断是否是字符’1’。

```
#include <bits/stdc++.h>//7-1934 -2   csp-j2019   数字游戏（number） javacn
using namespace std;

/*
读入 8 个字符，依次判断是否是字符’1’
*/
char c;
int r = 0;

int main()
{
	for(int i = 1;i <= 8;i++)
	{
		cin>>c;
		if(c == '1') r++;
	}

	cout<<r;
	return 0;
}
```

相关知识：队列、模拟。

维护 45 分钟的优惠券队列，来存储当前可用的优惠券：

1、如果是当前是地铁票，先买票，然后将优惠券存入队列；

2、如果当前是公交车： 从队列中，找出第 1 个没有被用过的，且金额 >= 当前票价的优惠券；

找到：标记该优惠券使用；

没找到：只能买票！ 队列可以用数组模拟（本题不适合用 STL 的 queue，因为 queue 无法遍历，我们需要遍历 queue 找到第 1 张符合条件的优惠券）！

```
/*
有多张优惠票满足条件，
则优先消耗获得最早的优惠票
1. 坐地铁
买票
获得等额的优惠券：金额、获得时间、是否使用过
2. 坐公交
在有效期内的优惠券中，查找第 1 张金额 >= 公交票价的优惠券
找到：使用该优惠券
找不到：买票
使用队列，维护优惠券的信息
*/


#include <bits/stdc++.h>//8-1935   csp-j2019   公交换乘（transfer）   javacn
using namespace std;

const int N = 1e5 + 10;
struct node{
    int price,time;
    bool used;
}q[N];// 优惠券的队列
int h = 1,t = 0;
int n,ans = 0;
```

```
int main()
{
	scanf("%d",&n);
	int type,money,start;
	for(int i = 1;i <= n;i++)
	{
		scanf("%d%d%d",&type,&money,&start);
		if(type == 0)			// 如果是地铁
		{
			ans += money;				// 买票，获得优惠券
			t++;
			q[t].price = money;
			q[t].time = start;
			q[t].used = false;// 默认没有使用
		}
		else
		{
			while(h<=t&&start-q[h].time>45) // 公交
				h++;		// 丢弃过期的优惠期
			bool f = false;// 假设没找到
			for(int j = h;j <= t;j++)
			{// 在有效期内的券，找第一张 >= 公交票价的且没有用过的优惠券
				if(q[j].price >= money && q[j].used == false)
				{
					q[j].used = true;
					f = true;
					break;
				}
			}
			if(f == false) ans += money;  // 如没有找到能用的优惠券，买票
		}
	}
	printf("%d",ans);
	return 0;
}
```

如果从 1 号点走到 x 号点，有最少奇数次的步数 mins，x 号点可以要求生产 >=mins 的奇数阶段的零件。

如果从 1 号点到 x 号点，有最少偶数次的步数 mins，x 号点可以要求生产 >=mins 的偶数阶段的零件。

```
#include <bits/stdc++.h>//9-1988  csp-j2019  加工零件（work）  javacn
using namespace std;
/*
思路：
1. 求出 1 号点到每个点的最短奇数路径，和最短偶数路径；
2. q 次询问，每次询问 x 号工人，如果生产 len 阶段的零件，1 号点是否需要原材料；
如果 1 号点存在到 x 号点的最短奇数路径 mins，且 len 是奇数，且 len >= mins，1 号点就需要提供原材料
偶数性质同上；
SPFA 求奇偶最短路。
*/
const int N = 1e5 + 10;
struct node{
    int to,next;
}a[N*2];
int pre[N],k = 0;
queue<int> q;// 队列
//d[i][0] 代表走到每个点的偶数最短路
int d[N][2];
bool vis[N][2];
int n,m,t;

// 建边
void add(int u,int v)
{
      k++;
      a[k].to = v;
      a[k].next = pre[u];
      pre[u] = k;
}
```

```
int main()
{
	scanf("%d%d%d",&n,&m,&t);
	int x,y;
	for(int i = 1;i <= m;i++)
	{
		scanf("%d%d",&x,&y);
		add(x,y);
		add(y,x);// 双向建边
	}

	// 求 1 号点到每个点的奇偶最短路
	memset(d,0x3f,sizeof(d));
	//1 号点走到自己的偶数最短路是 0
	d[1][0] = 0;
	// 如果 1 号点到其他点没有边
	if(pre[1] == 0)		d[1][0] = 0x3f3f3f3f;
```

```
q.push(1);
while(!q.empty())
{
	int u = q.front();
	for(int i = pre[u];i != 0;i = a[i].next)
	{
		int to = a[i].to;
		if(d[u][0]+1<d[to][1])
		{
			d[to][1] = d[u][0] + 1;
			// 如果该点不在队列，存入队列
			if(!vis[to][1])
			{
				q.push(to);
				vis[to][1] = true;
			}
		}

		// 走到 u 点奇数最短路 +1< 走到 to 点的偶数最短路
		if(d[u][1]+1<d[to][0])
		{
			// 更新走到 to 点的偶数最短路
			d[to][0] = d[u][1] + 1;
			if(!vis[to][0])
			{
				q.push(to);
				vis[to][0] = true;
			}
		}
	}

	q.pop();
}

//t 次询问
```

```
        //t 次询问
        int len;
        while(t--)
        {
                scanf("%d%d",&x,&len);
                if(d[x][len%2]<=len) printf("%s\n","Yes");
                else printf("%s\n","No");
        }

        return 0;
}
```

```cpp
#include<bits/stdc++.h>//10-1565    niop2017    成绩（score）   wintereta
using namespace std;
int main()
{
	double a,b,c;
	cin>>a>>b>>c;
	cout<<a*0.2+b*0.3+c*0.5;


	return 0;
}
```

如果需求码的长度为 len，用图书编码 %10 的 len 次方取余数，得到最后的 len 位，判断是否是需求码 ，如果是，打擂台求最小。

```
#include <bits/stdc++.h>//11-1566  noip2017      图书管理员（librarian）     javacn
using namespace std;
需要的书中图书编码最小的那本书，如果没有他需要的书，请输出 -1
如果需求码的长度为 len，用图书编码 %10 的 len 次方取余数，得到最后的 len 位
判断是否是需求码
const int N = 1010;
int a[N],n,q,len,x;
int main()
{
	cin>>n>>q;
	for(int i = 1;i <= n;i++)
	{
		cin>>a[i];
	}
	int mi;		// 读入需求码
	for(int i = 1;i <= q;i++)
	{
		cin>>len>>x;
		// 遍历所有的图书编码，找满足条件的最小的编码
		mi = INT_MAX;
		for(int j = 1;j <= n;j++)
		{
			int t = pow(10,len);//10 的 len 次方
			if(a[j] % t == x && a[j] < mi)
			{
				mi = a[j];
			}
		}
		if(mi == INT_MAX) cout<<-1<<endl;
		else cout<<mi<<endl;
	}
	return 0;
}
```

本题是最小步数问题，要注意每个点递归判断下一个点是否有颜色：

1. 下一个点有颜色，同色则总金币不变，不同色，总金币 +1

2. 下一个点无色，则要求走到当前点没有用过魔法，且往下一个点走总金币 +2

```
#include<bits/stdc++.h>//12-1483    noip2017    棋盘（chess）    javacn
using namespace std;
/*
1. 红色、黄色或没有任何颜色
2. 你所站在的位置必须是有颜色
3. 同色不花费金币，不同色花费 1 个金币
4. 可以花费 2 个金币将无色改有色
5. 魔法不能连用
 c=1 代表红色，c=2 代表黄色
*/
int n,m,a[1010][1010],d[1010][1010];
int fx[5]={0,0,1,0,-1};
int fy[5]={0,1,0,-1,0};
```

```
void dfs(int x,int y,int sum,bool magic)
{
	d[x][y] = sum;// 更新最少步数（金币数）

	// 尝试 4 个方向
	for(int i = 1;i <= 4;i++)
	{
		int tx = x + fx[i];
		int ty = y + fy[i];
		// 如果在迷宫内
		if(tx>=1&&tx<=m&&ty>=1&&ty<=m)
		{
			// 如果下一格有色
			if(a[tx][ty] != 0)
			{
				// 同色计算，sum 不变
			if(a[tx][ty] == a[x][y] && sum < d[tx][ty]) dfs(tx,ty,sum,false);
			else if(sum+1 < d[tx][ty]) dfs(tx,ty,sum+1,false);// 非同色，花 1
个金币
			}
			else
			{
				// 如果下一格无色
				// 如果到本格没有使用魔法
				if(magic == false && sum+2 < d[tx][ty])
				{
					a[tx][ty] = a[x][y];// 使用魔法变为同色
					dfs(tx,ty,sum+2,true);
					a[tx][ty] = 0;// 递归回来，魔法失效
				}
			}
		}

	}
}
```

```
int main()
{
	cin>>m>>n;
	int x,y,t;
	// 将 d 数组初始化为 INT_MAX
	for(int i=1;i<=m;i++)
	{
		for(int j=1;j<=m;j++)
		{
			d[i][j]=INT_MAX;
		}
	}

	// 读入 n 个有颜色的点
	for(int i=1;i<=n;i++)
	{
		cin>>x>>y>>t;
		a[x][y]=t+1;// 读入颜色 +1
	}
	// 从 1,1 点出发，使用金币数为 0，没有用过魔法
	dfs(1,1,0,false);

	// 如果走不到
	if(d[m][m] == INT_MAX) cout<<-1;
	else cout<<d[m][m];

	return 0;
}
```

打擂台求最小花费：

```
#include <bits/stdc++.h>//13-1567    noip2016    买铅笔（pencil）    javacn
using namespace std;

int n,x,p;// 要买 n 支，每种包装 x 支，p 元
int mi = INT_MAX;//mi 存储最少花费

int main()
{
	cin>>n;
	int t;
	for(int i = 1;i <= 3;i++)
	{
		cin>>x>>p;
		if(n % x == 0)		t = n / x * p;
		else		t = (n / x + 1) * p;

		mi = min(t,mi);
	}
	cout<<mi;
	return 0;
}
```

使用数组模拟队列，求解本题：

```
#include <bits/stdc++.h>//14-1571  noip2016  海港（port）  javacn
using namespace std;
struct node{
	int num,time;// 国籍和到岗时间
};
node q[300010];
int h = 1,t = 0;// 默认队列为空
int n,ti,ki,x,ans = 0;
int na[100010];// 统计每个数出现的次数
int main()
{
	scanf("%d",&n);
	while(n--)
	{
		scanf("%d%d",&ti,&ki);
		while(h<=t&&ti-q[h].time>=86400)	// 讨论出有没有过时出队列的元素
		{
			na[q[h].num]--;
			if(na[q[h].num] == 0) ans--;
			h++;
		}
		for(int i = 1;i <= ki;i++)// 将 ki 个人入队列，统计国籍种类数
		{
			scanf("%d",&x);
			na[x]++;
			if(na[x] == 1) ans++;// 自增结束，值为 1，说明该数是第一次出现
			t++;
			q[t].num = x;
			q[t].time = ti;
		}
		printf("%d\n",ans);
	}
	return 0;
}
```

1、理解四个点之间的关系

因为： b-a=2(d-c) ， b-a<(c-b)/3，假设 d-c=k

所以：b-a=2*k，c-b>6*k

2、理解本题所述的方案求解数学原理

（1）假设 k = 2，有如下数字。

1 2 3 4 5 6 ... 19 20 21

c d

19 作为第三个数字 c 出现的可能有：1 5、2 6，2 种不同的可能。

21 作为第四个数字 d 出现的可能也是 2 种，原理同上。

备注：如果 19 有 3 个，那么就有 3 * 2 = 6 种解法得到数字 d

（2）如果有重复的数字，可以用数组计数法（桶排序）统计出每个数字出现的次数。

比如：假设 k = 1，统计结果如下

出现次数：2 1 0 3 2 4 2 1

数据元素：1 2 3 4 5 6 7 8 9 10 11 12 13 14

13 作为数字 c 出现的可能有：(2 * 0 + 1 * 3 + 0 * 2 + 3 * 4) * 1 = 15 种

15 作为数字 d 出现的可能有：(2 * 0 + 1 * 3 + 0 * 2 + 3 * 4) * 2 = 30 种

也就是：

f3[c] = sum(cnt[a] * cnt[b]) * cnt[d]

f4[d] = sum(cnt[a] * cnt[b]) * cnt[c]

由于 b、c 之间只要满足：c - b > 6 * k，这样一来，越往左的 a、b 数对越容易满足题意，也就是可以从左向右求前缀和。

反过来枚举 ab 其中一个点时，cd 越往右越容易满足，因此从右往左求前缀和。

（3）枚举的每个点的范围

枚举步长 k 可能的范围：由图可知 1+2k+6k+k<n，因此：k 的范围 [1,n/9)
 枚举 d 出现的位置：[1+9*k+1,n]，有了 d 的位置，c 的位置必定是 d-k，a 的位置必定是：d-9*k-1，b 的位置必定是：a+2*k。

枚举 a 出现的位置：[1,n-9*k-1]，有了 a 的位置，b 的位置必定是 a+k*2，d 的位置必定是 a+9*k+1，c 的位置必定是 d-k。

按照该枚举方式，本题时间复杂度 = n/9×n = O(n^2/9)

```
#include <bits/stdc++.h>//15-2215    noip2016    魔法阵（magic） javacn
using namespace std;

/*
m 个魔法物品，Xi 是不超过 n 的正整数
可能有多个物品的魔法值相同
Xa < Xb < Xc < Xd,Xb-Xa=2(Xd-Xc)
Xb-Xa<(Xc-Xb)/3
*/
const int N = 15010,M = 40010;
int cnt[N];// 统计每个数出现的次数
// 计算每个数分别作为第 1 个数 ~ 第 4 个数的不同可能
int f1[N],f2[N],f3[N],f4[N];
int x[M];
int n,m;

int main()
{
      scanf("%d%d",&n,&m);
      // 读入 m 个数，统计每个数出现的次数
      for(int i = 1;i <= m;i++)
      {
            scanf( “%d” ,&x[i]);
            cnt[x[i]]++;
      }

    // 枚举可能的步长
```

```
        // 枚举可能的步长
        int a,b,c,d;
        int sum;// 前缀和
        for(int k = 1;1 + 9 * k < n;k++)
        {
                sum = 0;
                // 枚举 d 可能的范围
                for(d = 1 + 9 * k + 1;d <= n;d++)
                {
                        a = d - 9 * k - 1;
                        b = a + 2 * k;
                        c = d - k;
                        sum += cnt[a] * cnt[b];
                        f3[c] += sum * cnt[d];
                        f4[d] += sum * cnt[c];
                }
                sum = 0;
                // 枚举 a 可能的范围
                for(a = n - 9 * k - 1;a >= 1;a--)
                {
                        b = a + 2 * k;
                        d = a + 9 * k + 1;
                        c = d - k;
                        sum += cnt[c] * cnt[d];
                        f1[a] += sum * cnt[b];
                        f2[b] += sum * cnt[a];
                }
        }
        // 输出结果
        for(int i = 1;i <= m;i++)
        {
                printf("%d %d %d %d\n",f1[x[i]],f2[x[i]],f3[x[i]],f4[x[i]]);
        }
        return 0;
}
```

思路：循环 n 天，当发放 k 金币的天数等于 k 时，金币数要增加

```
#include<bits/stdc++.h>//16-1520    noip2015    骑士的金币（coin）    javacn
using namespace std;

int main()
{
    // 定义整数 n 代表天数，sum 代表总金币，k 代表每天发放的金币数，c 代表发放 k 金币的天数
    int n,sum=0,c=0,k=1;
    cin>>n;
    // 循环 n 天
    for(int i=1;i<=n;i++)
    {
        sum+=k;// 总金币
        c++;// 发放 k 金币数的天数
        // 当发放 k 金币的天数等于 k 时
        if(c==k)
        {
            c=0;//c 要清 0，重新开始计数
            k++;// 发放金币数 +1
        }
    }
    cout<<sum<<endl;
    return 0;
}
```

思路：用字符型二维数组读入地图

遍历地图的每一格，如果是*（地雷）直接输出

如果是?(非地雷格)，计算其八方向中有多少个地雷并输出

```
#include<bits/stdc++.h>//17-1580-1    noip2015    扫雷（mine）    javacn
using namespace std;

char a[110][110];// 存储地图
int n,m;

int main()
{
	cin>>n>>m;
	int i,j,c;
	// 读入地图
	for(i = 1;i <= n;i++)
	{
		for(j = 1;j <= m;j++)
		{
			cin>>a[i][j];
		}
	}

	// 输出地图
```

```
        //输出地图
        for(i = 1;i <= n;i++)
        {
                for(j = 1;j <= m;j++)
                {
                        if(a[i][j] == '*') cout<<a[i][j];
                        else
                        {
                                //计算 i,j 这个单元格的 8 方向中有几个地雷
                                c = 0;
                                if(a[i-1][j-1]=='*') c++;
                                if(a[i-1][j]=='*') c++;
                                if(a[i-1][j+1]=='*') c++;
                                if(a[i][j+1]=='*') c++;
                                if(a[i+1][j+1]=='*') c++;
                                if(a[i+1][j]=='*') c++;
                                if(a[i+1][j-1]=='*') c++;
                                if(a[i][j-1]=='*') c++;

                                cout<<c;
                        }
                }

                cout<<endl;//第 i 行打印结束，输出换行符
        }
}
```

根据题目可以知道每个格子相邻的地雷数最多 8 个，所以可以利用此特征，将有地雷的方格赋值一个大于 8 的整数，输出时判断，小于等于 8 的存储单元才是有效的地雷数，否则输出地雷符号。 读入地图时某位置如果是地雷则该位置赋值为 9，对该点相邻的 8 个点累加 1，有‘*’号的点本身赋值大于 8，累加后也大于 8，对输出无影响。 边界问题，数组从 f（1，1）开始，最小行列坐标点的上面和左面有行下标 [0] 和列下标 [0]，不会越界。

```
#include <iostream>//17-1580-2    noip2015    扫雷（mine）    anselxu
using namespace std;
int a[105][105];// 存储输出
int main()
{
      int n,m;
      char c;
      cin>>n>>m;
      for(int i=1;i<=n;i++)
      {

            for(int j=1;j<=m;j++)
            {
                  cin>>c;// 读入符号
                  if(c=='*')
                  {     // 如果读入的是地雷，进行赋值操作
                        // 地雷的位置赋值一个大于 8 的数，相邻位置 --> 累加
                        a[i][j]=9;
                        a[i-1][j]++;
                        a[i-1][j+1]++;
                        a[i-1][j-1]++;
                        a[i][j-1]++;
                        a[i][j+1]++;
                        a[i+1][j]++;
                        a[i+1][j+1]++;
                        a[i+1][j-1]++;
                  }
            }
      }
```

```
	for(int i=1;i<=n;i++)
	{
		for(int j=1;j<=m;j++)
			if(a[i][j]<=8)//输出时判断是否有效地雷数
				cout<<a[i][j];
			else//无效地雷数输出地雷符号
				cout<<'*';
		cout<<endl;
	}
	return 0;
}
```

如果有三个数 x、y、z 是三元组，则一定满足 y - x = z - y，也就是必定是等差数列。

上述算式简化可得 2 y = x + z，也就是 x + z 是偶数，那么 x 和 z 必定同为奇数，或者同为偶数。

通过上述分析得知，本题和 y 没关系，只需要计算 x 和 z 就行了，但如果穷举 x 和 z 的值，时间任然会超限。

利用三元组的第 2 个条件 colorx = colorz，也就是 x 和 z 颜色也必定相同。那么就有了 2 个条件：x 和 z 必定奇偶相同，x 和 z 必定颜色值相同。

利用分组思想：把相同颜色分为一组，在每个颜色中按下标的奇偶分组，这样在同一组内的数值就满足三元组的规律。

假设有满足条件的分组对应额总和如下：

sum =

(x1 + x2) (nx1 + nx2) + (x2 + x3) (nx2 + nx3) + (x1 + x3) (nx1 + nx3)

= x1 nx1 + x1 nx2 + x2 nx1 + x2 nx2

x2 nx2 + x2 nx3 + x3 nx2 + x3 nx3

x1 nx1 + x1 nx3 + x3 nx1 + x3 nx3

= x1 (nx1 + nx2 + nx1 + nx3) + x2 (nx1 + nx2 + nx2 + nx3)

x3 (nx2 + nx3 + nx1 + nx3)

= x1 (nx1 + nx2 + nx3 + (3-2)*nx1)

x2 (nx1 + nx2 + nx3 + (3-2)nx2)

x3 (nx1 + nx2 + nx3 + (3-2)nx3)

```cpp
#include <bits/stdc++.h>//18-1592    noip2015    求和（sum）    javacn
using namespace std;

const int N = 1e5 + 10,MOD = 10007;
int num[N],color[N];
// 将所有的数据按照颜色分组、再按奇偶分组
// 求对应分组的数有几个、对应分组的数值和是多少
int cnt[N][2],sum[N][2];
int n,m,res = 0;

int main()
{
	scanf("%d%d",&n,&m);
	// 读入 n 个数的数值
	for(int i = 1;i <= n;i++)	scanf("%d",&num[i]);

	// 读入颜色
	for(int i = 1;i <= n;i++)
	{
		scanf("%d",&color[i]);
		// 分组统计
		cnt[color[i]][i%2]++;
		sum[color[i]][i%2]=(sum[color[i]][i%2]+num[i])%MOD;
	}

	// 计算
	for(int i = 1;i <= n;i++)
	{
	    res = (res + i * (sum[color[i]][i%2]+(cnt[color[i]][i%2]-2)*num[i] %
MOD)) % MOD;
	}

	printf("%d",res);
	return 0;
}
```

相关知识：贪心、前缀最大值、前缀和。

Si：1 2 2 4 5

Ai：5 4 3 2 1

（1）本质分析

假设选 x 家，本题计算结果 ans = max(max(si)2+sum(ai))

假设 x = 2，选 2 家推销，如果选择 ai 值最大的 2 家，ans = 5 + 4 + 2 2 = 13

但是这并不是最优解，因为如果放弃掉其中较小的 ai=4，选一个更远的 si，会得到更优解，也就是 si 2 + ai 的最大值 = 5 2 + 1 = 11，因此最优解 = 11 + 5 = 16。

分析：是否可能放弃 ai 最大的 2 个解，通过更远的距离来获得更优解呢？答案是不能。

因为：si 2 + ai 已经选了最大，再放弃最大的 ai，不可能得到更大的和。

因此，本题的最优解：

f[x] = max(最大的 x 个 ai 的和 + 选出点的最大 si 2, 前 x-1 个最大 ai 和 + 更远的点中 si2+ai 的最大值)

（2）计算优化

采用暴力计算的时间复杂度为 O(n2)，不能拿满分。

优化点：

A、对所有数据按照 ai 降序排序；

B、求出 ai 的前缀和。

C、求出 si 的前缀最大值。

D、求出每个点向后看的 2 si + ai 的最大值。

E、套公式求解；

```
#include <bits/stdc++.h>//19-1593   noip2015   推销员（salesman）   javacn
using namespace std;
/*
每走 1 米就会积累 1 点疲劳值
向第 i 家住户推销产品会积累 Ai 点疲劳值
*/
const int N = 1e5 + 10;
struct node{
    int s,a;
}num[N];
int fa[N];//ai 的前缀和
int fs[N];//si 的前缀最大值
int fr[N];// 从 i~n 之间 2*si+ai 的最大值
int n;
```

```
// 排序规则：按推销疲劳值降序排序
bool cmp(node n1,node n2){
    return n1.a > n2.a;
}

int main()
{
      scanf("%d",&n);
      for(int i = 1;i <= n;i++)      scanf("%d",&num[i].s);
      for(int i = 1;i <= n;i++)      scanf("%d",&num[i].a);
      // 将所有的点按照推销疲劳值降序排序
      sort(num+1,num+n+1,cmp);

      //ai 的前缀和
      for(int i = 1;i <= n;i++)       fa[i] = fa[i-1] + num[i].a;
      //si 的前缀最大值
      for(int i = 1;i <= n;i++)       fs[i] = max(fs[i-1],num[i].s);
      // 从 i~n 之间 2*si+ai 的最大值
      for(int i = n;i >= 1;i--)       fr[i] = max(fr[i+1],2*num[i].s+num[i].a);
      // 计算
      for(int i = 1;i <= n;i++)
      {
            printf("%d\n",max(fs[i]*2+fa[i],fa[i-1]+fr[i]));
      }
      return 0;
}
```

题意是问：n 个数的数组中，有多少个数，是另外 2 个数的和。

比如：有 5 个数，1 2 3 4 5，那么有 3 个数是另外两个数的和，分别是 3、4、5。

大家不要误解为：有多少组 2 个数的和，在数组中，如果你这样理解，上述案例中就有 4 组数的和在数组中了，分别是：1+2、1+3、1+4、2+3。

由于本题，数组中的数都在 10000 的范围内，因此可以用标记数组来标记哪些数出现过，穷举数组中的两两组合，只要组合对应的数在数组中有，计数器自增，并标记这个数没有出现。

```
#include <iostream>//20-1594   noip2014   珠心算测验（count）   javacn
using namespace std;
int a[100];// 数组中的数
bool f[20010];// 标记哪些数出现了
int c = 0;// 计数器
int n;
int main()
{
	cin>>n;
	for(int i = 1; i <= n; i++)
	{
		cin>>a[i];
		f[a[i]] = true;// 标记出现了
	}
	for(int i = 1; i <= n; i++)
	{
		for(int j = i + 1; j <= n; j++)
		{
			// 如果该数存在
			if(f[a[i] + a[j]] == true)
			{
				c++;
				// 这个数统计过了，标记为不存在
				f[a[i] + a[j]] = false;
			}
		}
	}
	cout<<c;
}
```

```cpp
#include <bits/stdc++.h>//21-1595   noip2014   比例简化（ratio） David0724
using namespace std;
/*
试试这个行不行
c d 都在 l 之内 一起枚举
要求条件
1 c d 互质
2 a/b ?= c/d
3 a/b - c/d 尽可能小
*/
bool hz(int a,int b)// 互质
{

	while(a % b != 0)
	{
		int c = a % b;
		a = b;
		b = c;
	}
	if(b == 1)return 1;
	else return 0;
}
double mi =INT_MAX * 1.0;// 差值的最小值
```

```
int main()
{
	int a, b, c,d;
	int l;// 上限
	cin>>a>>b>>l;

//	cout<<a * 1.0 / b<<endl;

	for(int i = 1 ; i <= l; i++)
	{
		for(int j = 1; j <= l;j++)
		{

			if(hz(i,j) and i * 1.0 /j >= a * 1.0 / b)
			{// 在满足上述条件下求最小值
    // printf("%d %d %f %f\n",i,j,i * 1.0 /j, i * 1.0 /j -a * 1.0 / b );

				if( i * 1.0 /j -a * 1.0 / b < mi)
				{
					mi = i * 1.0 /j -a * 1.0 / b;
					c = i;
					d = j;
				//printf("%d %d %f\n",i,j,mi);
				}
			}
		}
	}
	printf("%d %d\n",c,d);
	return 0;
}
```

第一步：分解并存储每个运算数（只存最后 4 位）、分解并存储每个运算符

第二步：遍历每个数字，如果对应的运算符是 *，先将乘法计算，将结果存储到下一位，本位清零

注意本题中的数学原理：

将两个整数求和取最后 4 位 = 将两个这个数分别取最后 4 位后求和，再取最后 4 位

将两个整数求积取最后 4 位 = 将两个这个数分别取最后 4 位后求积，再取最后 4 位

之所以要不断的取最后 4 位，是为了防止溢出。

```
#include<bits/stdc++.h>//22-1666  noip2013  表达式求值    javacn
using namespace std;

/*
第一步：分解并存储每个运算数（只存最后 4 位）、分解并存储每个运算符
第二步：遍历每个数字，如果对应的运算符是 *，先将乘法计算，将结果存储到下一位
        本位清零
*/
string s,w = "";
const int N = 100010;
int a[N],k1 = 0,k2 = 0,t;// 运算数
char f[N];// 运算符
int r = 0;// 结果
```

```
int main()
{
	cin>>s;
	// 分解数字和运算符
	for(int i = 0;i < s.size();i++)
	{
		// 如果是数字
		if(isdigit(s[i]))
		{
			w = w + s[i];
			// 判断连续数字结束
			if(i==s.size()-1||!isdigit(s[i+1]))
			{
				t = atoi(w.c_str());
				k1++;
				a[k1] = t % 10000;
				w = "";// 清空 w 存放下一个连续的数字
			}
		}
		else
		{
			k2++;
			f[k2] = s[i];
		}
	}

	// 计算乘法
	// 循环运算符
```

```
        //计算乘法
        //循环运算符
        for(int i = 1;i <= k2;i++)
        {
                if(f[i] =='*')
                {
                        a[i+1] = (a[i+1] * a[i]) % 10000;
                        a[i] = 0;
                }
        }

        //求和
        for(int i = 1;i <= k1;i++)
        {
                r = (r + a[i]) % 10000;
        }

        cout<<r;
}
```

先求出以每个数结尾的最大和，再取和的前缀最大值作为每个人的特征值（因为不一定以第 i 个数结尾的最大和，是前 i 个数取连续数的最大和）。

接下来根据题意算分数，再计算分数的最大值。

```
#include <bits/stdc++.h>//23-1800    noip2014    小朋友的数字    javacn
using namespace std;
typedef long long LL;
const int N = 1e6 + 10;
LL n,p,a[N];
//f: 存放特征值
LL s[N],f[N],score[N],r,ma = LONG_LONG_MIN;//r 存放每个人的分数
int main()
{
	cin>>n>>p;
	// 读入 n 个值
	for(int i = 1; i <= n; i++)
	{
		cin>>a[i];
		// 以每个点结束的最大和
		s[i] = max(s[i-1]+a[i],a[i]);
		ma = max(ma,s[i]);
		f[i] = ma % p;// 特征值
	}
	// 第一个人的分数是特征值
	score[1] = f[1];
	LL ans = score[1];
	ma = LONG_LONG_MIN;
	for(int i = 2; i <= n; i++)    // 计算每个人的得分
	{
		ma = max(ma,f[i-1]+score[i-1]);
		score[i] = ma;
		ans = max(ans,ma) % p;
	}
	cout<<ans;
	return 0;
}
```

按题意计算正确的识别码，判断 ISBN 是否正确，如果有误，修正识别码。

```
#include <bits/stdc++.h>//24-1798  noip 更早   ISBN 号码   javacn
using namespace std;
x-xxx-xxxxx-x
前 9 个数字，是用于计算的数字，最后一位是识别码
string s;
int r,t = 1;//r 前 9 位按条件计算的结果，t 表示权重
int main()
{
	cin>>s;
	for(int i = 0;i < s.size() - 1;i++)
	{
		if(isdigit(s[i]))
		{
			r = r + (s[i] - '0') * t;
			t++;
		}
	}

	r = r % 11; // 计算出来的识别码
	// 判断计算出来的识别码是否 == ISBN 的最后一位
	if(r < 10)
	{
		// 如果识别码正确
		if(s[12]-'0'==r) cout<<"Right";
		else cout<<s.substr(0,12)<<r;// 正确的 ISBN
	}
	else
	{
		if(s[12]=='X') cout<<"Right";
		else cout<<s.substr(0,12)<<'X';// 正确的 ISBN
	}
	return 0;
}
```

```cpp
#include <bits/stdc++.h>//25-1482-1    noip 更早    花生采摘   javacn
using namespace std;
//p：存放数组的坐标及坐标对应的花生数量
int m,n,k,a[30][30],p[1000][3];
int main()
{
	int i,j,s = 0;
	// 读入数据
	cin>>m>>n>>k;
	for(i = 1;i <= m;i++)
	{
		for(j = 1;j <= n;j++)
		{
			cin>>a[i][j];
			if(a[i][j] > 0)
			{
				p[s][0] = i;
				p[s][1] = j;
				p[s][2] = a[i][j];
				s++;
			}
		}
	}
	// 对 p 数组按照花生数量由大到小排序
	for(i = 1;i < s;i++)
	{
		for(j = 0;j <= s - i - 1;j++)
		{
			if(p[j][2] < p[j+1][2])
			{
				swap(p[j],p[j+1]);
			}
		}
	}
	// 统计时间和摘到的花生数量
```

```
	//统计时间和摘到的花生数量
	int time = 0;
	int sum = 0;
	//遍历每个花生点
	for(i = 0;i < s;i++)
	{
		//如果是第一个花生点
		if(i == 0)
		{
			if(p[i][0] + 1 + p[i][0] <= k)
			{
				time = p[i][0] + 1;
				sum = sum + p[i][2];
			}
			else
			{
				break;
			}
		}
		else
		{
			if(time + abs(p[i][0]-p[i-1][0]) + abs(p[i][1]-p[i-1][1]) + 1
+ p[i][0] <= k){
				time = time + abs(p[i][0]-p[i-1][0]) + abs(p[i][1]-p[i-
1][1]) + 1;
				sum = sum + p[i][2];
			}
			else
			{
				break;
			}
		}
	}
	cout<<sum<<endl;
}
```

鲁宾逊先生说：“你先找出花生最多的植株，去采摘它的花生；然后再找出剩下的植株里花生最多的，去采摘它的花生；依此类推，不过你一定要在我限定的时间内回到路边。”
题目说要从最大的开始采摘，所以是个贪心算法，否则就成了动态规划了.

```
#include<bits/stdc++.h>//25-1482-2    noip更早    花生采摘    jiangyf70

using namespace std;
struct huashengdi{
    int x, y, s;
} a[410];
bool cmp(huashengdi a, huashengdi b)
{
    return a.s > b.s;
}

int main()
{
      int m, n, k, sum = 0;
      cin >> m >> n >> k;
      int b = 0;
      for(int i = 1; i <= m; i++)
      {
            for(int j = 1; j <= n; j++)
            {
                  a[b].x = i;
                  a[b].y = j;
                  cin >> a[b].s;
                  b++;
            }
      }
```

```
	sort(a, a + b, cmp);
	sum = 0;
	int t = 0;
	int jl = 0, c= 0;
	for(int i = 0; i < b; i++)
	{
		if(i == 0)
		{
			if(a[0].x + 1 + a[0].x <= k)
			{
				t = a[0].x + 1;
				sum = a[0].s;
			}
			else
			{
				break;
			}
		}
		else
		{
			jl = abs(a[i].x - a[i-1].x) + abs(a[i].y - a[i-1].y);
			if(t + jl + 1 + a[i].x <= k)
			{
				sum += a[i].s;
				t = t + jl + 1;
			}
			else
			{
				break;
			}
		}
	}
	cout << sum;
	return 0;
}
```

我们可以发现，任何一个位置都只能从左边和右边传过来，这样我们就可以列出我们的方程：dp[i][j]=dp[i-1][j-1]+dp[i-1][j+1]。

dp[i][j]：代表第 i 次传球，第 j 个人获得球的机会数

j!=1&&j!=n：dp[i][j] = dp[i-1][j-1] + dp[i-1][j+1]

j=1 dp[i][j] = dp[i-1][j+1] + dp[i-1][n]，这是计算第 i 次传球，第 1 个人获得球的机会数

j=n dp[i][j] = dp[i-1][j-1] + dp[i-1][1]，这是计算第 i 次传球，第 n 个人获得球的机会数

```
/*
dp[i][j]：代表第 i 次传球，第 j 个人获得球的机会数
j!=1&&j!=n：dp[i][j] = dp[i-1][j-1] + dp[i-1][j+1]
j=1 dp[i][j] = dp[i-1][j+1] + dp[i-1][n]
j=n dp[i][j] = dp[i-1][j-1] + dp[i-1][1]

边界：dp[0][1] = 1
*/
```

```cpp
#include <bits/stdc++.h>//26-1801    noip 更早     传球游戏    javacn
using namespace std;
int n,m;
// 代表第 i 次传球，第 j 个人获得球的机会数
int dp[40][40];
int main()
{
	cin>>n>>m;
	// 边界：第 0 次传球，第 1 个人获得球的机会数
	dp[0][1] = 1;

	// 传 m 次球
	for(int i = 1;i <= m;i++)
	{
		for(int j = 1;j <= n;j++)
		{
			if(j==1) dp[i][j] = dp[i-1][j+1] + dp[i-1][n];
			else if(j==n) dp[i][j] = dp[i-1][j-1] + dp[i-1][1];
			else dp[i][j] = dp[i-1][j-1] + dp[i-1][j+1];
		}
	}

	// 经过 m 次传球，球回到第 1 个人手中的机会数
	cout<<dp[m][1];
	return 0;
}
```

贪心求解：先将每一个横向通道，每一个纵向通道隔开了多少个说话的人统计出来。然后贪心的选择 k 个隔开人数最多的横向通道，l 个隔开人数最多的纵向通道。

```
#include <bits/stdc++.h>//27-1797    noip 更早      排座椅    ojcpp
using namespace std;
/* m 行 n 列 k 条横向通道 l 条纵向通道 d 对说话的同学 */
int m,n,k,l,d;
// 横纵坐标数组，存储在这个位置画线隔开几对同学
int x[1010],y[1010];
// 横纵坐标桶排序的数组
vector<int> rx,ry;

int main()
{
	int x1,y1,x2,y2;// 用来存放说话的人的坐标
	int i,j;
	cin>>m>>n>>k>>l>>d;
	// 读入 d 对说话的人
	for(i = 1;i <= d;i++)
	{
		cin>>x1>>y1>>x2>>y2;
		// 说明在同一行，用列隔开
		if(x1 == x2)
		{
			// 该列的线能隔开几个人（线在人的右侧）
			y[min(y1,y2)]++;
		}
		else
		{
			// 该行的线能隔开几个人（线在人的上方）
			x[min(x1,x2)]++;
		}
	}

	// 桶排序，标记 k 条横向的通道
```

```
// 桶排序，标记 k 条横向的通道
for(i = 1;i <= k;i++)
{
	int maxn = -1;
	int p;
	for(j = 1;j < m;j++)
	{
		if(x[j] > maxn)
		{
			maxn = x[j];
			p = j;
		}
	}
	// 标记该行的线分割了 0 个人
	x[p] = 0;
	rx.push_back(p);
}

// 桶排序，标记 l 条纵向的通道
for(i = 1;i <= l;i++)
{
	int maxn = -1;
	int p;
	for(j = 1;j < n;j++)
	{
		if(y[j] > maxn)
		{
			maxn = y[j];
			p = j;
		}
	}
	// 标记该列的线分割了 0 个人
	y[p] = 0;
	ry.push_back(p);
}
```

```
	sort(rx.begin(),rx.end());
	sort(ry.begin(),ry.end());
	for(i = 0;i < rx.size();i++)
	{
		cout<<rx[i]<<" ";
	}
	cout<<endl;
	for(i = 0;i < ry.size();i++)
	{
		cout<<ry[i]<<" ";
	}

	return 0;
}
```