ACM 模板库

cqsh3vj2

2014-10-16

BITSS

目录

小技巧

C++手动扩栈

```
#pragma comment(linker, "/STACK:102400000,102400000")
```

输入挂

```
int input()
{
	char ch=' ';
	while(ch<'0'||ch>'9')ch=getchar();
	int x=0;
	while(ch<='9'&&ch>='0')x=x*10+ch-'0',ch=getchar();
	return x;
}
```

输出挂

```
void Out(int a)		//输出外挂
{
	 if(a>9)
		Out(a/10);
	 putchar(a%10+'0');
}
```

用位运算快速求解全部组合数

这个函数用来求 C(n, k)，其中 comb 就是二进制形式表示的子集，例如 000111 表示由后三个元素构成的组合或子集，并且该迭代过程可生成字典序递增的组合，这一特性非常有用。
当 k 为 0 时需要直接输出

```
int main(){
	int k=2,n=5;
	int comb = (1<<k) - 1;
	while(comb < 1<<(n))
	{
		//printf("%d\n",comb);//先使用
		int x = comb & -comb, y = comb + x;
		comb = ((comb & ~y) / x >> 1) | y;
	}
	return 0;
}
```

Java 快速输入输出

```
import java.io.*;
import java.math.BigInteger;
import java.util.StringTokenizer;

class Scan {

	BufferedReader buffer;
	StringTokenizer tok;

	Scan() {
		buffer = new BufferedReader(new
InputStreamReader(System.in));
	}

	boolean hasNext() {
		while (tok == null || !tok.hasMoreElements()) {
			try {
				tok = new
StringTokenizer(buffer.readLine());
			} catch (Exception e) {
				return false;
			}
		}
		return true;
	}

	String next() {
		if (hasNext())
			return tok.nextToken();
		return null;
	}

	int nextInt() {
		return Integer.parseInt(next());
	}

	long nextLong() {
		return Long.parseLong(next());
	}
}

public class Main {
	public static void main(String[] args) {
		PrintWriter out = new PrintWriter(new
BufferedWriter(
				new
```

```
OutputStreamWriter(System.out)));
        Scan scan = new Scan();
        int T,R;
        T=scan.nextInt();
        for(R=1;R<=T;R++)
        {
            long n;
            n=scan.nextLong();
            BigInteger eight = BigInteger.valueOf(8);
            BigInteger seven = BigInteger.valueOf(7);
            BigInteger one = BigInteger.valueOf(1);
            BigInteger N = BigInteger.valueOf(n);
            BigInteger ans ;
            ans =
N.multiply(N).multiply(eight).subtract(N.multiply(seven)).a
dd(one);
            out.println("Case #"+R+": "+ans);
        }
        out.flush();
    }
}
```

Java 大数进制转换

```
import java.io.*;
import java.math.BigInteger;
import java.util.StringTokenizer;

class Scan {

    BufferedReader buffer;
    StringTokenizer tok;

    Scan() {
        buffer = new BufferedReader(new
InputStreamReader(System.in));
    }

    boolean hasNext() {
        while (tok == null || !tok.hasMoreElements()) {
            try {
                tok = new
StringTokenizer(buffer.readLine());
            } catch (Exception e) {
                return false;
            }
        }
        return true;
    }

    String next() {
        if (hasNext())
            return tok.nextToken();
        return null;
    }

    int nextInt() {
        return Integer.parseInt(next());
    }

    long nextLong() {
        return Long.parseLong(next());
    }
}

public class Main {
    public static void main(String[] args) {
        PrintWriter out = new PrintWriter(new
BufferedWriter(
                new
OutputStreamWriter(System.out)));
        Scan scan = new Scan();
        int T,R;
        T=scan.nextInt();
        for(R=1;R<=T;R++)
        {
            String s = scan.next();
            BigInteger a = new BigInteger(s,2);
            s = scan.next();
            BigInteger b = new BigInteger(s,2);
            BigInteger c = a.gcd(b);
            out.println("Case #"+R+": "+c.toString(2));
        }
        out.flush();
    }
}
```

对整数的二分查找

```
//二分查找--在 l-r 范围内查找值 v
//返回下标，失败返回-1
//假设 a 数组已经按照从小到大排序

int bs(int a[],int left,int right,int v)
{
    int mid;
```

```
    while(left<right)
    {
        mid=(left+right)>>1;
        if(a[mid]==v)return mid;
        if(a[mid]<v)left=mid+1;
        else right=mid;
    }
    return -1;
}

//二分查找--大于等于 v 的第一个值
//假设 a 已经排好
//传入参数必须 left<=right，返回值 l 总是合理的

int bs(int a[],int left,int right,int v)
{
    int mid;
    while(left<right)
    {
        mid=(left+right)>>1;
        if(a[mid]<v) left=mid+1;//if(不符合条件)
        else right=mid;
    }
    return left;
}

//二分查找--小于等于 v 的第一个值
//假设 a 已经排好
//传入参数必须 left<=right，返回值 l 总是合理的

int bs(int a[],int left,int right,int v)
{
    int mid;
    while(left<right)
    {
        mid=(left+right+1)>>1;
        if(a[mid]>v) right=mid-1;//if(不符合条件)
        else left=mid;
    }
    return right;
}
```

树状数组

1.改点(+/-)求区间

```
void add(int x,int v)
{
    for(;x<=n;x+=x&(-x))
        C[x]+=v;
}

void sub(int x,int v)
{
    for(;x<=n;x+=x&(-x))
        C[x]-=v;
}

int getsum(int x)//求 1-x 的和
{
    int sum=0;
    for(;x>0;x-=x&(-x))
        sum+=C[x];
    return sum;
}
```

2.求比某点 x 小的点的个数

对于 x，求 getsum（x），再加入 x：add（x，1）；

同理可求逆序（倒着插），求平面上在点 A 左下方的点的个数

3.改区间求点

令 del[i]表示 i..MN 段被加了多少。则区间修改[a,b]时，我们让 del[a]++,del[b+1]--即可。（此时树状数组是建立在 del 数组基础上的）这个时候，要求某一个位置 x 的数的值时，用 Getsum(x)来表示。

Sort 与 qsort 使用细则

```
/*qsort 与 sort 调用细则*/
#include<stdlib.h>
#include<stdio.h>
#include<string.h>

//qsort

//普通数组
    a[1000];
    int comp(const void *a,const void *b)
    {
        return *(int *)a - *(int *)b;//(int *)表示把（const coid * a）强制准换成 int 型指针  外边的*是取内容
    }
    qsort(a,k,sizeof(int),comp);
//对结构体排序
    struct edgetype{
        int a;
        int b;
```

```
        int len;
    }edge[5000];
    int comp(const void *a,const void *b)
    {
        return (((struct edgetype *)a)->len)-(((struct edgetype *)b)->len);
    }
    qsort(edge,k,sizeof(struct edgetype),comp);
//对 double
    int comp(const void *a,const void *b)
    {
        if((((struct edgetype *)a)->len)-(((struct edgetype *)b)->len)>0)
            return 1;
        else if((((struct edgetype *)a)->len)-(((struct edgetype *)b)->len)==0)
            return 0;
        else if((((struct edgetype *)a)->len)-(((struct edgetype *)b)->len)<0)
            return -1;
    }
//对特定长度进行排序
    qsort(a+x,k,sizeof(int),comp);//从 a【x】开始的 k 个数进行排序 例如： a[0]=3 a[1]=2 a[2]=1 a[3]=0 qsort(a+1,2,sizeof(int),comp) 之后 3 1 2 0

//sort 调用细则
#include<algorithm>

//对普通数组
    int a[1000];
    bool cmp(int a,int b)
    {
        return a<b;//当满足当前顺序，返回 1，不满足当前顺序，返回 0
    }
    sort(a+x,a+y,cmp);//对 a【x】-a【y-1】进行排序（y==x 与 y==x+1 效果相同）
//对结构排序
    struct edgetype{
        int a;
        int b;
        int len;
    }edge[5000];
    bool cmp(struct edgetype a,struct edgetype b)
    {
        return a.len<b.len;//当满足当前顺序，返回 1，不满足当前顺序，返回 0
    }
    sort(edge+x,edge+y,cmp);
```

判断日期的小模板

```
int leap_year( int year ) // 判断闰年或平年
{   return ( (year%4==0 && year%100!=0) || year%400==0 ) ? 1 : 0;
}

int year_days(int year)        // 计算一整年的天数
{   return leap_year( year ) ? 366 : 365;
}

int days( int year, int month, int day ) // 计算该天 month,day 是本年 year 的第几天
{   int months[13] = {0,31,28,31,30,31,30,31,31,30,31,30,31}, i;

    if  ( leap_year( year ) && month >2 )
        day++;

    for ( i=1; i<month; i++ )
        day += months[i];

    return day;
}
```

ACM 三角形模板

```
#include <math.h>
struct point{double x,y;};
struct line{point a,b;};

double distance(point p1,point p2){
return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
}

point intersection(line u,line v){
point ret=u.a;
double t=((u.a.x-v.a.x)*(v.a.y-v.b.y)-(u.a.y-v.a.y)*(v.a.x-v.b.x))
/((u.a.x-u.b.x)*(v.a.y-v.b.y)-(u.a.y-u.b.y)*(v.a.x-v.b.x));
ret.x+=(u.b.x-u.a.x)*t;
ret.y+=(u.b.y-u.a.y)*t;
return ret;
}

//外心
```

```
point circumcenter(point a,point b,point c){
line u,v;
u.a.x=(a.x+b.x)/2;
u.a.y=(a.y+b.y)/2;
u.b.x=u.a.x-a.y+b.y;
u.b.y=u.a.y+a.x-b.x;
v.a.x=(a.x+c.x)/2;
v.a.y=(a.y+c.y)/2;
v.b.x=v.a.x-a.y+c.y;
v.b.y=v.a.y+a.x-c.x;
return intersection(u,v);
}

//内心
point incenter(point a,point b,point c){
line u,v;
double m,n;
u.a=a;
m=atan2(b.y-a.y,b.x-a.x);
n=atan2(c.y-a.y,c.x-a.x);
u.b.x=u.a.x+cos((m+n)/2);
u.b.y=u.a.y+sin((m+n)/2);
v.a=b;
m=atan2(a.y-b.y,a.x-b.x);
n=atan2(c.y-b.y,c.x-b.x);
v.b.x=v.a.x+cos((m+n)/2);
v.b.y=v.a.y+sin((m+n)/2);
return intersection(u,v);
}

//垂心
point perpencenter(point a,point b,point c){
line u,v;
u.a=c;
u.b.x=u.a.x-a.y+b.y;
u.b.y=u.a.y+a.x-b.x;
v.a=b;
v.b.x=v.a.x-a.y+c.y;
v.b.y=v.a.y+a.x-c.x;
return intersection(u,v);
}

//重心
//到三角形三顶点距离的平方和最小的点
//三角形内到三边距离之积最大的点
point barycenter(point a,point b,point c){
line u,v;
u.a.x=(a.x+b.x)/2;
u.a.y=(a.y+b.y)/2;
u.b=c;
v.a.x=(a.x+c.x)/2;
v.a.y=(a.y+c.y)/2;
v.b=b;
return intersection(u,v);
}

//费马点
//到三角形三顶点距离之和最小的点
point fermentpoint(point a,point b,point c){
point u,v;
double
step=fabs(a.x)+fabs(a.y)+fabs(b.x)+fabs(b.y)+fabs(c.x)+fabs
(c.y);
int i,j,k;
u.x=(a.x+b.x+c.x)/3;
u.y=(a.y+b.y+c.y)/3;
while (step>1e-10)
for (k=0;k<10;step/=2,k++)
for (i=-1;i<=1;i++)
for (j=-1;j<=1;j++){
v.x=u.x+step*i;
v.y=u.y+step*j;
if
(distance(u,a)+distance(u,b)+distance(u,c)>distance(v,a)+di
stance(v,b)+distance(v,c))
u=v;
}
return u;
}
```

位图法

```
#define SHIFT 5
#define MASK 0x1F
#define DIGITS 32

int map[31500]={0};//6 位数 31250+ 7 位数 312500+ 8 位
数 3125000+（12.5MB）

void map_set(int n)                    //将逻辑位置为 n 的二
进制位置为 1
{
    map[n>>SHIFT] |=(1<<(n&MASK));   //n>>SHIFT 右移
5 位相当于除以 32 求算字节位置，n&MASK 相当于对 32
取余即求位位置，
}
```

```
int map_check(int n)
{
    return map[n>>SHIFT] & (1<<(n&MASK));  //测试逻辑位置为 n 的二进制位是否为 1,为 1 返回 1 否则返回 0
}
```

网络流模板

最大流 dinic 模板

```
const int maxnode = 1000 + 5;
const int maxedge = 2*1000 + 5;
const int oo = 1000000000;
int node, src, dest, nedge;
int head[maxnode], point[maxedge], next1[maxedge], flow[maxedge], capa[maxedge];//point[x]==y 表示第 x 条边连接 y，head，next 为邻接表，flow[x]表示 x 边的动态值，capa[x]表示 x 边的初始值
int dist[maxnode], Q[maxnode], work[maxnode];//dist[i]表示 i 点的等级
void init(int _node, int _src, int _dest){//初始化，node 表示点的个数，src 表示起点，dest 表示终点
    node = _node;
    src = _src;
    dest = _dest;
    for (int i = 0; i < node; i++) head[i] = -1;
    nedge = 0;
}
void addedge(int u, int v, int c1, int c2){//增加一条 u 到 v 流量为 c1，v 到 u 流量为 c2 的两条边
    point[nedge] = v, capa[nedge] = c1, flow[nedge] = 0, next1[nedge] = head[u], head[u] = (nedge++);
    point[nedge] = u, capa[nedge] = c2, flow[nedge] = 0, next1[nedge] = head[v], head[v] = (nedge++);
}
bool dinic_bfs(){
    memset(dist, 255, sizeof (dist));
    dist[src] = 0;
    int sizeQ = 0;
    Q[sizeQ++] = src;
    for (int cl = 0; cl < sizeQ; cl++)
        for (int k = Q[cl], i = head[k]; i >= 0; i = next1[i])
            if (flow[i] < capa[i] && dist[point[i]] < 0){
                dist[point[i]] = dist[k] + 1;
                Q[sizeQ++] = point[i];
            }
    return dist[dest] >= 0;
}
int dinic_dfs(int x, int exp){
    if (x == dest) return exp;
    for (int &i = work[x]; i >= 0; i = next1[i]){
        int v = point[i], tmp;
        if (flow[i] < capa[i] && dist[v] == dist[x] + 1 && (tmp = dinic_dfs(v, min(exp, capa[i] - flow[i]))) > 0){
            flow[i] += tmp;
            flow[i^1] -= tmp;
            return tmp;
        }
    }
    return 0;
}
int dinic_flow(){
    int result = 0;
    while (dinic_bfs()){
        for (int i = 0; i < node; i++) work[i] = head[i];
        while (1){
            int delta = dinic_dfs(src, oo);
            if (delta == 0) break;
            result += delta;
        }
    }
    return result;
}
//建图前,运行一遍 init()；
//加边时，运行 addedge(a,b,c,0),表示点 a 到 b 流量为 c 的边建成（注意点序号要从 0 开始）
//求解最大流运行 dinic_flow(),返回值即为答案
```

最小费用最大流模板

```
const int N = 1010;//点
const int M = 2 * 10010;//边
const int inf = 1000000000;
struct Node{//边，点 f 到点 t，流量为 c，费用为 w
    int f, t, c, w;
}e[M];
int next1[M], point[N], dis[N], q[N], pre[N], ne;//ne 为已添加的边数，next，point 为邻接表,dis 为花费，pre 为父亲节点
bool u[N];
void init(){
    memset(point, -1, sizeof(point));
    ne = 0;
}
```

```
void add_edge(int f, int t, int d1, int d2, int w){//f 到 t 的一条边，流量为 d1,反向流量 d2,花费 w,反向边花费-w（可以反悔）
    e[ne].f = f, e[ne].t = t, e[ne].c = d1, e[ne].w = w;
    next1[ne] = point[f], point[f] = ne++;
    e[ne].f = t, e[ne].t = f, e[ne].c = d2, e[ne].w = -w;
    next1[ne] = point[t], point[t] = ne++;
}
bool spfa(int s, int t, int n){
    int i, tmp, l, r;
    memset(pre, -1, sizeof(pre));
    for(i = 0; i < n; ++i)
        dis[i] = inf;
    dis[s] = 0;
    q[0] = s;
    l = 0, r = 1;
    u[s] = true;
    while(l != r) {
        tmp = q[l];
        l = (l + 1) % (n + 1);
        u[tmp] = false;
        for(i = point[tmp]; i != -1; i = next1[i]) {
            if(e[i].c && dis[e[i].t] > dis[tmp] + e[i].w) {
                dis[e[i].t] = dis[tmp] + e[i].w;
                pre[e[i].t] = i;
                if(!u[e[i].t]) {
                    u[e[i].t] = true;
                    q[r] = e[i].t;
                    r = (r + 1) % (n + 1);
                }
            }
        }
    }
    if(pre[t] == -1)
        return false;
    return true;
}
void MCMF(int s, int t, int n, int &flow, int &cost){//起点 s，终点 t，点数 n，最大流 flow，最小花费 cost
    int tmp, arg;
    flow = cost = 0;
    while(spfa(s, t, n)) {
        arg = inf, tmp = t;
        while(tmp != s) {
            arg = min(arg, e[pre[tmp]].c);
            tmp = e[pre[tmp]].f;
        }
        tmp = t;
        while(tmp != s) {
            e[pre[tmp]].c -= arg;
            e[pre[tmp] ^ 1].c += arg;
            tmp = e[pre[tmp]].f;
        }
        flow += arg;
        cost += arg * dis[t];
    }
}
//建图前运行 init()
//节点下标从 0 开始
//加边时运行 add_edge(a,b,c,0,d)表示加一条 a 到 b 的流量为 c 花费为 d 的边（注意花费为单位流量花费）
// 特别注意双向边，运行 add_edge(a,b,c,0,d),add_edge(b,a,c,0,d)较好，不要只运行一次 add_edge(a,b,c,c,d),费用会不对。
//求解时代入 MCMF(s,t,n,v1,v2)，表示起点为 s，终点为 t，点数为 n 的图中，最大流为 v1，最大花费为 v2
```

二分图相关

二分图匹配有自己的匈牙利算法，不过我更习惯用网络流的算法搞。

首先一些概念：

最大匹配：

在所有的匹配中，边数最多的那个匹配就是二分图的最大匹配

顶点覆盖：

在顶点集合中，选取一部分顶点，这些顶点能够把所有的边都覆盖了。这些点就是顶点覆盖集

最小顶点覆盖：

在所有的顶点覆盖集中，顶点数最小的那个叫最小顶点集合。

独立集：

在所有的顶点中选取一些顶点，这些顶点两两之间没有连线，这些点就叫独立集

最大独立集：

在所有的独立集中，顶点数最多的那个集合

路径覆盖：

在图中找一些路径，这些路径覆盖图中所有的顶点，每个顶点都只与一条路径相关联。

最小路径覆盖：

在所有的路径覆盖中，路径个数最小的就是最小路径覆盖了。

最大匹配=最小顶点覆盖（就像网络流中最大流=最小割）
最大独立集=顶点个数-最小顶点覆盖（最大匹配）

编程之美-格格取数

题目是给出一个 n*m 的矩阵，每个点是一个值，求取出权值和最小的点来把各行各列都覆盖。
于是转换成一个二分图，左边 n 个点，右边 m 个点，中间的边就是原来矩阵的每个点，求权值和最少的边把左右两排点覆盖。
使用最小费用最大流，构图如下：
添加原点和超级原点，汇点和超级汇点。
超级原点到原点流量为 n*m，花费为 0；
汇点到超级汇点流量为 n*m，花费为 0；
左边 2*n 个点，右边 2*m 个点。
上方的 n，m 个点是用来约束覆盖的：流量为 1，费用为负无穷
下方的 n,m 个点是用来提供边的：流量为 m，n（其实就是无穷），费用为 0
关键：从原点到汇点添加流量为 n*m,费用为 0 的边，用来分流
最终结果加上一个（n+m）*负无穷

ACdream1171 下界转上界-最大费用可行流

每一行/列至少取 A/B 的最小费用=SUM-每一行/列至多取 m-A/n-B 的最大费用
由于是求可行流而不是最大流，所以添加超级源汇，在源汇上连一条分流的路

多校-网络流

给一个 n*m 的矩阵，每个位置有一个 0~K 的数，给出每一行及每一列的数的和，判断是否有解，并输出方案。
首先是否有解可以通过 impossible 判定，即从原点到各行流量为各行的和，从各列到汇点流量为各列的和，从每个原点到每个汇点的流量表示某个点的值，所以流量为 K。跑一边网络流，如果最大流=总和，有解。
有解之后的判定，标程是用的残余网络的进一步处理。我比赛时的方法比赛比较暴力，即行/列为 0 的，还有必须填满的都是确定的，把确定的行求出来，并填进去，就可以确定更多的行和列。其实反复两三次就够了。
（有极限数据 矩阵的右下方为相同的数时 需要反复 n 次）
代码写得比较残。

```
#include<stdio.h>
#include<iostream>
#include<string.h>
#include<algorithm>
#include<math.h>
#include<queue>
using namespace std;
#define ll __int64
#define eps 1e-8
const ll Mod=(1e9+7);
const int maxn = 410;
const int maxm = 10100;

int n,m,k;
int r[maxn],c[maxn];
int ma[maxn][maxn];



const int maxnode = 1000 + 5;
const int maxedge = 2*161000 + 5;
const int oo = 1000000000;
int node, src, dest, nedge;
int head[maxnode], point[maxedge], next1[maxedge], flow[maxedge], capa[maxedge];//point[x]==y 表示第 x 条边连接 y，head，next 为邻接表，flow[x]表示 x 边的动态值，capa[x]表示 x 边的初始值
int dist[maxnode], Q[maxnode], work[maxnode];//dist[i]表示 i 点的等级
void init(int _node, int _src, int _dest){//初始化，node 表示点的个数，src 表示起点，dest 表示终点
    node = _node;
```

```
    src = _src;
    dest = _dest;
    for (int i = 0; i < node; i++) head[i] = -1;
    nedge = 0;
}
void addedge(int u, int v, int c1, int c2){//增加一条 u 到 v 流量为 c1，v 到 u 流量为 c2 的两条边
    point[nedge] = v, capa[nedge] = c1, flow[nedge] = 0, next1[nedge] = head[u], head[u] = (nedge++);
    point[nedge] = u, capa[nedge] = c2, flow[nedge] = 0, next1[nedge] = head[v], head[v] = (nedge++);
}
bool dinic_bfs(){
    memset(dist, 255, sizeof (dist));
    dist[src] = 0;
    int sizeQ = 0;
    Q[sizeQ++] = src;
    for (int cl = 0; cl < sizeQ; cl++)
        for (int k = Q[cl], i = head[k]; i >= 0; i = next1[i])
            if (flow[i] < capa[i] && dist[point[i]] < 0){
                dist[point[i]] = dist[k] + 1;
                Q[sizeQ++] = point[i];
            }
    return dist[dest] >= 0;
}
int dinic_dfs(int x, int exp){
    if (x == dest) return exp;
    for (int &i = work[x]; i >= 0; i = next1[i]){
        int v = point[i], tmp;
        if (flow[i] < capa[i] && dist[v] == dist[x] + 1 && (tmp = dinic_dfs(v, min(exp, capa[i] - flow[i]))) > 0){
            flow[i] += tmp;
            flow[i^1] -= tmp;
            return tmp;
        }
    }
    return 0;
}
int dinic_flow(){
    int result = 0;
    while (dinic_bfs()){
        for (int i = 0; i < node; i++) work[i] = head[i];
        while (1){
            int delta = dinic_dfs(src, oo);
            if (delta == 0) break;
            result += delta;
        }
    }
    return result;
}
//建图前,运行一遍 init()；
//加边时，运行 addedge(a,b,c,0),表示点 a 到 b 流量为 c 的边建成（注意点序号要从 0 开始）
//求解最大流运行 dinic_flow(),返回值即为答案


bool judge(int sumrow){
    int flow = 1,cost = 0;
    for(int i = 1;i <= n;i++)
        for(int j = n+1;j <= n+m;j ++)
            addedge(i,j,k,0);
    flow=dinic_flow();
    if(flow != sumrow)
        return false;
    return true;
}
int main(){
    while(scanf("%d%d%d",&n,&m,&k) != EOF){
        init(n+m+2,0,n+m+1);
        int flag = 0;
        int sumrow = 0,colrow = 0;
        for(int i = 1;i <= n;i++){
            scanf("%d",&r[i]);
            addedge(0,i,r[i],0);
            sumrow += r[i];
            if(r[i]<0 || r[i]>m*k)
                flag = 1;
        }
        for(int j = 1;j <= m;j ++){
            scanf("%d",&c[j]);
            addedge(j+n,n+m+1,c[j],0);
            colrow += c[j];
            if(c[j]<0 || c[j]>n*k)
                flag = 1;
        }
        if(sumrow != colrow){
            printf("Impossible\n");
            continue;
        }
        if(!judge(sumrow))
            flag = 1;
        if(flag == 1){
            printf("Impossible\n");
            continue;
        }
        memset(ma,-1,sizeof(ma));
        int i,j;
        for(i=1;i<=n;i++)
```

```
        if(r[i]==0)
            for(j=1;j<=m;j++)
                ma[i][j]=0;
    for(j=1;j<=m;j++)
        if(c[j]==0)
            for(i=1;i<=n;i++)
                ma[i][j]=0;
    int tt=2;
    int sum,num,temp;
    while(tt--)
    {
        for(i=1;i<=n;i++)
        {
            if(r[i]==0)
            {
                for(j=1;j<=m;j++)
                    if(ma[i][j]==-1)
                        ma[i][j]=0;
                continue;
            }
            sum=0;
            num=0;
            for(j=1;j<=m;j++)
            {
                if(ma[i][j]==-1)
                {
                    num++;
                    temp=j;
                    sum+=min(k,c[j]);
                }
            }
            if(num==1)
            {
                ma[i][temp]=r[i];
                r[i]-=ma[i][temp];
                c[temp]-=ma[i][temp];
                continue;
            }
            else if(sum==r[i])
            {
                for(j=1;j<=m;j++)
                {
                    if(ma[i][j]==-1)
                    {
                        ma[i][j]=min(k,c[j]);
                        r[i]-=ma[i][j];
                        c[j]-=ma[i][j];
                    }
                }
            }
        }
        for(j=1;j<=m;j++)
        {
            if(c[j]==0)
            {
                for(i=1;i<=n;i++)
                    if(ma[i][j]==-1)
                        ma[i][j]=0;
                continue;
            }
            sum=0;
            num=0;
            for(i=1;i<=n;i++)
            {
                if(ma[i][j]==-1)
                {
                    num++;
                    temp=i;
                    sum+=min(k,r[i]);
                }
            }
            if(num==1)
            {
                ma[temp][j]=c[j];
                r[temp]-=ma[temp][j];
                c[j]-=ma[temp][j];
                continue;
            }
            else if(sum==c[j])
            {
                for(i=1;i<=n;i++)
                {
                    if(ma[i][j]==-1)
                    {
                        ma[i][j]=min(k,r[i]);
                        r[i]-=ma[i][j];
                        c[j]-=ma[i][j];
                    }
                }
            }
        }
    }
    flag=0;
    for(i=1;i<=n;i++)
        if(r[i]!=0)
        {
            flag=1;
            break;
```

```
            }
        for(j=1;j<=m;j++)
            if(c[j]!=0)
            {
                flag=1;
                break;
            }
        if(flag==1)
            printf("Not Unique\n");
        else
        {
            printf("Unique\n");
            for(i=1;i<=n;i++)
            {
                for(j=1;j<m;j++)
                    printf("%d ",ma[i][j]);
                printf("%d\n",ma[i][m]);
            }
        }
    }
    return 0;
}
```

无源汇上下界网络流

总的来说就是新建一个原点汇点 s,t，对原来的每个点 i，设 m(i)=sum{b<j,i>}-sum{b<i,j>}其中 b<>为流量下界，若 m(i)>0，连一条<s,i>容量为 m(i)的边；若 m(i)<0，连一条<i,t>容量为|m(i)|的边。然后将原来边的容量变为 c<i,j>-b<i,j>。求一次最大流。如果与 s 和 t 关联的边全部满流，则可行流存在，且每条边的流量为现在的流量+流量的下界。否则不存在可行流。

```
#include<iostream>
#include<algorithm>
#include<iostream>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<vector>
#include<queue>
#include<cmath>
using namespace std;
#define ll long long
#define inf 0x3f3f3f3f
int dir[4][2]={0,1,0,-1,1,0,-1,0};
int n,m;
int c[305];
int lo[50000];



const int maxnode = 300 + 5;
const int maxedge = 2*50000 + 5;
const int oo = 1000000000;
int node, src, dest, nedge;
int head[maxnode], point[maxedge], next1[maxedge], flow[maxedge], capa[maxedge];//point[x]==y 表示第 x 条边连接 y，head，next 为邻接表，flow[x]表示 x 边的动态值，capa[x]表示 x 边的初始值
int dist[maxnode], Q[maxnode], work[maxnode];//dist[i]表示 i 点的等级
void init(int _node, int _src, int _dest){//初始化，node 表示点的个数，src 表示起点，dest 表示终点
    node = _node;
    src = _src;
    dest = _dest;
    for (int i = 0; i < node; i++) head[i] = -1;
    nedge = 0;
}
void addedge(int u, int v, int c1, int c2){//增加一条 u 到 v 流量为 c1，v 到 u 流量为 c2 的两条边
    point[nedge] = v, capa[nedge] = c1, flow[nedge] = 0, next1[nedge] = head[u], head[u] = (nedge++);
    point[nedge] = u, capa[nedge] = c2, flow[nedge] = 0, next1[nedge] = head[v], head[v] = (nedge++);
}
bool dinic_bfs(){
    memset(dist, 255, sizeof (dist));
    dist[src] = 0;
    int sizeQ = 0;
    Q[sizeQ++] = src;
    for (int cl = 0; cl < sizeQ; cl++)
        for (int k = Q[cl], i = head[k]; i >= 0; i = next1[i])
            if (flow[i] < capa[i] && dist[point[i]] < 0){
                dist[point[i]] = dist[k] + 1;
                Q[sizeQ++] = point[i];
            }
            return dist[dest] >= 0;
}
int dinic_dfs(int x, int exp){
    if (x == dest) return exp;
    for (int &i = work[x]; i >= 0; i = next1[i]){
        int v = point[i], tmp;
        if (flow[i] < capa[i] && dist[v] == dist[x] + 1 && (tmp = dinic_dfs(v, min(exp, capa[i] - flow[i]))) > 0){
```

```
                flow[i] += tmp;
                flow[i^1] -= tmp;
                return tmp;
            }
        }
        return 0;
}
int dinic_flow(){
        int result = 0;
        while (dinic_bfs()){
            for (int i = 0; i < node; i++) work[i] = head[i];
            while (1){
                int delta = dinic_dfs(src, oo);
                if (delta == 0) break;
                result += delta;
            }
        }
        return result;
}
//建图前,运行一遍 init()；
//加边时，运行 addedge(a,b,c,0),表示点 a 到 b 流量为 c 的边建成（注意点序号要从 0 开始）
//求解最大流运行 dinic_flow(),返回值即为答案


int main()
{
        while(scanf("%d%d",&n,&m)!=EOF)
        {
            int i,j;
            int ta,tb,tc,td,sum=0;;
            init(n+2,0,n+1);
            memset(c,0,sizeof(c));
            for(i=1;i<=m;i++)
            {
                scanf("%d%d%d%d",&ta,&tb,&tc,&td);
                lo[i]=tc;
                addedge(ta,tb,td-tc,0);
                c[ta]-=tc;
                c[tb]+=tc;
            }
            for(i=1;i<=n;i++)
            {
                if(c[i]>0)
                {
                    addedge(0,i,c[i],0);
                    sum+=c[i];
                }
                else if(c[i]<0)
                    addedge(i,n+1,-c[i],0);
            }
            int ans;
            ans=dinic_flow();
            if(ans==sum)
            {
                printf("YES\n");
                for(i=0;i<m;i++)
                    printf("%d\n",flow[i*2]+lo[i+1]);
            }
            else
                printf("NO\n");
        }
        return 0;
}
```

有源汇上下界最大流

处理有源汇有上下界最大流问题是：
1.构造附加网络
2.对 ss、tt 求最大流(ss、tt 满流则有解)
3.若有解，对 s、t 求最大流

而有源汇有上下界最小流问题则是：
1.构造附加网络(不添加[t,s]边)
2.对 ss、tt 求最大流
3.添加[t,s]边
4.对 ss、tt 求最大流
5.若 ss、tt 满流，则[t,s]的流量就是最小流

题意：

一个屌丝给 m 个女神拍照，计划拍照 n 天，每一天屌丝给给定的 C 个女神拍照，每天拍照数不能超过 D 张，而且给每个女神 i 拍照有数量限制[Li，Ri]，对于每个女神 n 天的拍照总和不能少于 Gi，如果有解求屌丝最多能拍多少张照，并求每天给对应女神拍多少张照；否则输出 -1。

分析：

增设一源点 st，汇点 sd，st 到第 i 天连一条上界为 Di 下界为 0 的边，每个女神到汇点连一条下界为 Gi 上界为 oo 的边，对于每一天，当天到第 i 个女孩连一条[Li，Ri]的边。

建图模型：

源点 s，终点 d。超级源点 ss，超级终点 dd。首先判断是否存在满足所有边上下界的可行流，方法可以转化成无源汇有上下界的可行流问题（T->S无穷）。然后再以 s，t 跑一下最大流。

字符串

后缀数组模板

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include <algorithm>
using namespace std;

const int N = 200010;
char s[N];
int r[N], tx[N], ty[N], rs[N], ranks[N], sa[N], height[N], rmq[N][20]; //rs 基数排序

bool cmp(int *r, int a, int b, int len)
{
    return (r[a] == r[b]) && (r[a + len] == r[b + len]);
}
void suffix(int n, int m) //n 为长度，最大值小于 m
{
    int i, j, p, *x = tx, *y = ty, *t;
    for(i = 0; i < m; ++i) rs[i] = 0;
    for(i = 0; i < n; ++i) { x[i] = r[i]; ++rs[x[i]];}
    for(i = 1; i < m; ++i) rs[i] += rs[i - 1];
    for(i = n - 1; i >= 0; --i) sa[--rs[x[i]]] = i;
    for(j = p = 1; p < n; j <<= 1, m = p) {
        for(p = 0, i = n - j; i < n; ++i) y[p++] = i;
        for(i = 0; i < n; ++i) { if(sa[i] >= j) y[p++] = sa[i] - j; }
        for(i = 0; i < m; ++i) rs[i] = 0;
        for(i = 0; i < n; ++i) ++rs[x[y[i]]];
        for(i = 1; i < m; ++i) rs[i] += rs[i - 1];
        for(i = n - 1; i >= 0; --i) sa[--rs[x[y[i]]]] = y[i];
        t = x, x = y, y = t;
        for(i = 1, p = 1, x[sa[0]] = 0; i < n; ++i) {
            if(cmp(y, sa[i - 1], sa[i], j)) x[sa[i]] = p - 1;
            else x[sa[i]] = p++;
        }
    }
}

void calheight(int n)
{
    int i, j, k = 0;
    for(i = 1; i <= n; ++i)
        ranks[sa[i]] = i;
    for(i = 0; i < n; ++i) {
        if(k)
            --k;
        j = sa[ranks[i] - 1];
        while(r[i + k] == r[j + k])
            ++k;
        height[ranks[i]] = k;
    }
}

//////////////////////////////////////////////////////////
//////////rmq 求 lcp
void initrmq(int n)
{
    int i, k;
    for(i = 2; i <= n; ++i)
        rmq[i][0] = height[i];
    for(k = 1; (1 << k) <= n; ++k) {
        for(i = 2; i + (1 << k) - 1 <= n; ++i) {
            rmq[i][k] = min(rmq[i][k - 1],
                rmq[i + (1 << (k - 1))][k - 1]);
        }
    }
}

int Log[N];
void initlog()
{
    Log[0] = -1;
    for(int i=1;i<N;i++)
        Log[i]=(i&(i-1))?Log[i-1]:Log[i-1] + 1;
}

int lcp(int a, int b ,int n)//求 a,b 的后缀的公用前缀长度，从 0 计
{
    if(a==b) return n-a;
    a = ranks[a], b = ranks[b];
    if(a > b)
        swap(a, b);
    ++a;
    int k = (int) Log[b - a + 1] / Log[2];
```

```
    return min(rmq[a][k], rmq[b - (1 << k) + 1][k]);
}

//////////////////////////////////////////////////////////////
//////////求第 k 小串（忽略重复）

__int64 effectNum[N],sumEffectNum[N],allNum;//分别表示 sa 中的 i 后边有效串数，前 i 总有效串数;allNum 为总共不同串的个数
int aimSl,aimSr,aimSlength;//目标串左右端点以及长度
void stStringInit(int n)
{
    sumEffectNum[0]=0;
    for(int i=1;i<=n;i++)
    {
        effectNum[i]=n-sa[i]-height[i];
        sumEffectNum[i]=sumEffectNum[i-1]+effectNum[i];
    }
    allNum=sumEffectNum[n];
}

int aimSbs(int left,int right,__int64 v)
{
    int mid;
    while(left<right)
    {
        mid=(left+right)>>1;
        if(sumEffectNum[mid]<v) left=mid+1;//if(不符合条件)
        else right=mid;
    }
    return left;
}

void getAimString(__int64 aimst,int n)
{
    int temp;
    temp=aimSbs(1,n,aimst);
    aimSl=sa[temp];
    aimSr=aimSl+height[temp]+aimst-sumEffectNum[temp-1]-1;
    aimSlength=aimSr-aimSl+1;
}

//////////////////////////////////////////////////////////////
//////////////////////////////////

int main()
{
    //initlog();
    while( scanf("%s", s)!=EOF)
    {
        int i,j,n;
        n = strlen(s);
        for(i = 0; i < n; ++i)
            r[i] = s[i];
        r[n] = 0;//便于比较
        suffix(n + 1, 128);
        calheight(n);
        //initrmq(n);
    }
    return 0;
}

/*
simple:
    ababcaaabc
    n=10

st  string      ranks   /   sa    saString      height
0   ababcaaabc  3       /   10    0             0
1   babcaaabc   6       /   5     aaabc         0
2   abcaaabc    5       /   6     aabc          2
3   bcaaabc     8       /   0     ababcaaabc    1
4   caaabc      10      /   7     abc           2
5   aaabc       1       /   2     abcaaabc      3
6   aabc        2       /   1     babcaaabc     0
7   abc         4       /   8     bc            1
8   bc          7       /   3     bcaaabc       2
9   c           9       /   9     c             0
10 0            0       /   4     caaabc        1
*/
```

前缀数组 Trie 刘汝佳模板

```
//sigma_size=26 maxnode 为节点最大数：字符串数*每个字符串最大长度+x
struct trie
{
    int ch[maxnode][sigma_size];//ch[a][3]=c 表示第 a 个节点有一个子节点为'd'并且节点序列号为 c
    int val[maxnode];//存各个字符串权值，题目不涉及权值时，可以存 1
    int sz;
    int idx(char c) { return c - 'a';}//找到 char c 对应的 id 值
```

```
    void init()
    {
        sz = 1;
        memset(ch[0], 0, sizeof(ch[0]));
    }
    void insert(char *s, int v)//s 为字符串，v 为权值
    {
        int u = 0, n = strlen(s);
        for(int i = 0; i < n; i++)
        {
            int c = idx(s[i]);
            if(!ch[u][c])
            {
                memset(ch[sz], 0, sizeof(ch[sz]));
                val[sz] = 0;
                ch[u][c] = sz++;
            }
            u = ch[u][c];
        }
        val[u] = v;
    }
    int find(char *s)//查找 s，返回权值
    {
        int u = 0;
        for(int i = 0; s[i]; i++)
        {
            int c = idx(s[i]);
            if(!ch[u][c]) return 0;
            u = ch[u][c];
        }
        return val[u];
    }
}Trie;
```

字符串 hash

最接近 XXX 的素数：

100:97

1000:997

10000:9973

100000:99991

1000000：999983

```
// BKDR Hash Function
unsigned int BKDRHash(char *str)
{
    unsigned int seed = 131; // 31 131 1313 13131 131313 etc..
    unsigned int hash = 0;

    while (*str)
    {
        hash = hash * seed + (*str++);
    }

    return (hash & 0x7FFFFFFF);
}

// DJB Hash Function
unsigned int DJBHash(char *str)
{
    unsigned int hash = 5381;

    while (*str)
    {
        hash += (hash << 5) + (*str++);
    }

    return (hash & 0x7FFFFFFF);
}

// JS Hash Function
unsigned int JSHash(char *str)
{
    unsigned int hash = 1315423911;

    while (*str)
    {
        hash ^= ((hash << 5) + (*str++) + (hash >> 2));
    }

    return (hash & 0x7FFFFFFF);
}

// AP Hash Function
unsigned int APHash(char *str)
{
    unsigned int hash = 0;
    int i;

    for (i=0; *str; i++)
    {
        if ((i & 1) == 0)
        {
            hash ^= ((hash << 7) ^ (*str++) ^ (hash >> 3));
        }
        else
        {
```

```
                hash ^= (~((hash << 11) ^ (*str++) ^ (hash >>
5)));
            }
        }
        return (hash & 0x7FFFFFFF);
}
```

图论

朱刘算法模板-最小树形图

```
/*
最小树形图图模版-朱刘算法
模版说明：点标号必须 0-(N-1)
            必须去除到自身的点（到自身的边的边权赋
无限大）
NV 表示总点数，NE 表示总边数
*/
#define M 109
#define type int
const type inf=(1)<<30;
struct Node{
    int u , v;
    type cost;
}E[M*M+5];
int pre[M],ID[M],vis[M];
type In[M];
int n,m;
type Directed_MST(int root,int NV,int NE) {
    type ret = 0;
    while(true) {
        //1.找最小入边
        for(int i=0;i<NV;i++) In[i] = inf;
        for(int i=0;i<NE;i++){
            int u = E[i].u;
            int v = E[i].v;
            if(E[i].cost < In[v] && u != v) {
                pre[v] = u;
                In[v] = E[i].cost;
            }
        }
        for(int i=0;i<NV;i++) {
            if(i == root) continue;
            if(In[i] == inf)   return -1;//除了跟以外有点
没有入边,则根无法到达它
        }
        //2.找环
        int cntnode = 0;
    memset(ID,-1,sizeof(ID));
    memset(vis,-1,sizeof(vis));
        In[root] = 0;
        for(int i=0;i<NV;i++) {//标记每个环
            ret += In[i];
            int v = i;
            while(vis[v] != i && ID[v] == -1 && v != root)
{
                vis[v] = i;
                v = pre[v];
            }
            if(v != root && ID[v] == -1) {
                for(int u = pre[v] ; u != v ; u = pre[u]) {
                    ID[u] = cntnode;
                }
                ID[v] = cntnode ++;
            }
        }
        if(cntnode == 0)     break;//无环
        for(int i=0;i<NV;i++) if(ID[i] == -1) {
            ID[i] = cntnode ++;
        }
        //3.缩点,重新标记
        for(int i=0;i<NE;i++) {
            int v = E[i].v;
            E[i].u = ID[E[i].u];
            E[i].v = ID[E[i].v];
            if(E[i].u != E[i].v) {
                E[i].cost -= In[v];
            }
        }
        NV = cntnode;
        root = ID[root];
    }
    return ret;
}
```

拓扑排序

一种拓扑排序算法。该算法是简单而直观的，实质上属于广度优先遍历，因此称为广度优先拓扑排序算法。该算法包含下列几个步骤：

[1] 从有向图中找一个没有前趋的结点 v，若 v 不存在，则表明不可进行拓扑排序（图中有环路），结束（不完全成功）；

[2] 将 v 输出；

[3] 将 v 从图中删除，同时删除关联于 v 的所有的边

[4] 若图中全部结点均已输出，则结束（成功），否则转[1]继续进行

```
#include <iostream>
using namespace std;
int map[502][502], indegree[502], m, n, pur[502];

void topsort()
{
    int i, j, k=1;
    for(i=1; i<=n; i++)
    {
        for(j=1; j<=n; j++)
        {
            if(indegree[j]==0)
            {
                indegree[j]--;
                pur[k++] = j;
                for(int x=1; x<=n; x++)
                    if(map[j][x])
                        indegree[x]--;
                break;
            }
            if(j>n)
            {
                cout<<"存在环"<<endl;
                return ;
            }
        }
    }
}

void main()
{
    int i, j;
    while(cin>>n>>m)
    {
        memset(map,0,sizeof(map));
        memset(indegree,0,sizeof(indegree));
        int a, b;
        for(i=1; i<=m; i++)
        {
            cin>>a>>b;
            if(!map[a][b])
            {
                map[a][b] = 1;
                indegree[b]++;
            }
        }
        topsort();
        for(i=1; i<=n; i++)
            if(i!=n)
                cout<<pur[i]<<" ";
            else
                cout<<pur[i]<<endl;
    }
}
```

欧拉路与曼哈顿路

1、无向图存在欧拉回路条件：每个点入度都为偶数

2、无向图存在欧拉路条件：只有两个点入度为奇数或每个点入度都为偶数

3、有向图存在欧拉回路条件：每个点入度都等于出度

4、有向图存在欧拉路条件：只存在这样两个点：一个点入度等于出度+1，另一个点入度=出度-1，其他每个点入度都等于出度，或者每个点入度都等于出度。

给 100 个点，任意两点有且仅有一条有向边，请输出任意一条哈密顿通路，如果没有，输出 impossible
题目中的图叫做竞赛图
定理 1：n(n>=2)阶竞赛图一定存在哈密顿通路
也叫哈密顿回路:在图中找出一条包含所有结点的闭路，并且，出来起点和重点重合外，这条闭路所含结点是互不相同的

基本并查集

//要 int parent【】，并初始化为-1
//注意同一个等价类之间的元素再次 merge 会傻逼

```
int root(int p){
    if(parent[p]==-1) return p;
    else return parent[p]=root(parent[p]);
}

void merge(int a,int b){
    a=root(a);
```

```
    b=root(b);
    parent[a]=b;//在这里其实也可以设置一些值的传递
}
```

并查集的删除

并查集的删除需要一个数组来记录某点的真实对应节点，如果 a 点删除，则直接把 a 点的真实对应点指向一个新的 b，这样原有的结构信息不会错乱。
这题需要统计有几个等价类。。。一直以为可以像以前一样通过 parent 等不等于-1 判断 TAT。其实只要开一个新的数组进行标记，每找到一个新的祖节点就标记一下就好，这样废弃的集合就不会被错算

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

int parent[2000005];
int vis[2000005];
int mark[100005];

int num,n,m;

int root(int p)
{
    if(parent[p]==-1) return p;
    else return parent[p]=root(parent[p]);
}

void merge(int a,int b)
{
    a=root(a);
    b=root(b);
    parent[a]=b;//在这里其实也可以设置一些值的传递
}


int main()
{
    int R=0;
    while(scanf("%d%d",&n,&m)!=EOF &&n!=0)
    {
        R++;
        num=n;
        int i,j,ta,tb,ans=0;
        char ts[5];
        memset(parent,-1,sizeof(parent));
        memset(mark,-1,sizeof(mark));
        memset(vis,0,sizeof(vis));
        for(i=0;i<n;i++)
        {
            mark[i]=i;
        }
        for(i=1;i<=m;i++)
        {
            scanf("%s",ts);
            if(ts[0]=='M')
            {
                scanf("%d%d",&ta,&tb);
                if(root(mark[ta])!=root(mark[tb]))
                    merge(mark[ta],mark[tb]);
            }
            else
            {
                scanf("%d",&ta);
                mark[ta]=num++;
            }
        }
        for(i=0;i<n;i++)
        {
            ta=root(mark[i]);
            if(vis[ta]==0)
            {
                vis[ta]=1;
                ans++;
            }
        }
        printf("Case #%d: %d\n",R,ans);
    }
    return 0;
}
```

Spfa

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <iostream>
#include <algorithm>
#include <queue>
using namespace std;
const int inf =0x7fffffff;
const int maxn=300001;//点数
struct edge
{
```

```
    int from,to,w,next;
}e[1000001];//边数
int head[maxn];
int vis[maxn];
int dist[maxn];
int n,m,t;//n 个点，m 条边，t 用来生成边的标号
void init()//建图前运行 init
{
    t=0;
    memset(head,-1,sizeof(head));
}
void add(int i,int j,int w)
{
    e[t].from=i;
    e[t].to=j;
    e[t].w=w;
    e[t].next=head[i];
    head[i]=t++;
}
void spfa(int s)
{
    queue <int> q;
    for(int i=1;i<=n;i++)
    dist[i]=inf;
    memset(vis,false,sizeof(vis));
    q.push(s);
    dist[s]=0;
    while(!q.empty())
    {
        int u=q.front();
        q.pop();
        vis[u]=false;
        for(int i=head[u];i!=-1;i=e[i].next)
        {
            int v=e[i].to;
            if(dist[v]>dist[u]+e[i].w)
            {
                dist[v]=dist[u]+e[i].w;
                if(!vis[v])
                {
                    vis[v]=true;
                    q.push(v);
                }
            }
        }
    }
}
int main()
{
    int a,b,c,s,e;
    scanf("%d%d",&n,&m);
    init();
    while(m--)
    {
        scanf("%d%d%d",&a,&b,&c);
        add(a,b,c);
    }
    scanf("%d%d",&s,&e);
    spfa(s);
    if(dist[e]==inf) printf("-1\n");
    else printf("%d\n",dist[e]);
    return 0;
}
```

Tarjan 求强连通分量

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <iostream>
#include <algorithm>
#include <queue>
#include <stack>
using namespace std;
const int inf =0x3fffffff;
const int maxn=10005;//点数
struct edge
{
    int from,to,w,next;
}e[50005];//边数
stack<int>pq;
int head[maxn];
int instack[maxn];//是否在栈中
int dfn[maxn];//同时可用于判断是否遍历过
int low[maxn];
int n,m,t,index;//n 个点，m 条边，t 用来生成边的标号,index 用来生成点的标号
void init()//建图前运行 init
{
    t=0;
    index=0;
    memset(head,-1,sizeof(head));
    memset(instack,0,sizeof(instack));
    memset(dfn,-1,sizeof(dfn));
    memset(low,-1,sizeof(low));
    while(!pq.empty())
        pq.pop();
```

```
}
void add(int i,int j,int w)
{
	e[t].from=i;
	e[t].to=j;
	e[t].w=w;
	e[t].next=head[i];
	head[i]=t++;
}
void tarjan(int u)
{
	dfn[u]=low[u]=++index;
	pq.push(u);
	instack[u]=1;
	for(int i=head[u];i!=-1;i=e[i].next)
	{
		if(dfn[e[i].to]==-1)
		{
			tarjan(e[i].to);
			low[u]=min(low[u],low[e[i].to]);
		}
		else if(instack[e[i].to]==1)
		{
			low[u]=min(low[u],dfn[e[i].to]);
		}
	}
	if(dfn[u]==low[u])
	{
		int v;
		v=pq.top();//从这个 v 开始为强连通分量的一个
		//solve v
		pq.pop();
		instack[v]=0;
		while(v!=u)
		{
			v=pq.top();
			//solve v
			pq.pop();
			instack[v]=0;
		}
	}
}
int main()
{
	int a,b;
	int i,j;
	scanf("%d%d",&n,&m);
	init();
	while(m--)
	{
		scanf("%d%d",&a,&b);
		add(a,b,0);
	}
	for(i=1;i<=n;i++)
	{
		if(dfn[i]==-1)
			tarjan(i);
	}
	return 0;
}
```

求无向图的割点（重边不重边神马的无所谓）

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<vector>
using namespace std;

#define N 103
vector<int> g[N];
int n, low[N], dfn[N], f[N];
bool vis[N];

void dfs(int u, int depth, const int &root) { //root 为连通图的树根
	dfn[u] = low[u] = depth;
	vis[u] = true;
	int cnt = 0;
	for (int i=0; i<g[u].size(); i++) {
		int v = g[u][i];
		if (!vis[v]) {
			cnt++;
			dfs(v, depth+1, root);
			low[u] = min(low[u], low[v]);
			if (u!=root && low[v]>=dfn[u]) f[u]++;	//当 u 不为树根的时候
			if (u==root && cnt>=2) f[u]++; //当 u 为搜索树的树根的时候
		} else low[u] = min(low[u], dfn[v]);
	}
}

int cut_point() {
```

```
    memset(f, 0, sizeof(f));
    memset(vis, false, sizeof(vis));
    dfs(1, 1, 1);//认为原来是连通图
    int ans = 0;
    for (int i=1; i<=n; i++) if (f[i] >= 1) ans++;
    return ans;
}

int m;

int main()
{
    while(scanf("%d%d",&n,&m)!=EOF)
    {
        int i,j;
        int ta,tb;
        for(i=1;i<=n;i++)
            g[i].clear();
        for(i=1;i<=m;i++)
        {
            scanf("%d%d",&ta,&tb);
            g[ta].push_back(tb);
            g[tb].push_back(ta);
        }
        int ans;
        ans=cut_point();
        printf("%d\n",ans);
    }
    return 0;
}
```

无向图的桥 双连通分量

```
/*
 *无向图的桥及边的双连通分量，Tarjan 算法 O(E)
 */
#include <cstdio>
#include <cstring>
using namespace std;
#define MAXN 10000
#define MAXM 1000000

struct node {
    int v, w, pre;   //id：边的编号（处理重边情况）
} edge[MAXM];
int pos[MAXN], nEdge; //图的存储：链式前向星（池子法）

struct Bridge {
    int u, v;
} bridge[MAXM];   //用来记录桥
int tot; //桥的个数

int fa[MAXN], cc; //fa：各个点所属的缩点（连通块），cc 连通块的个数
int dfn[MAXN], low[MAXN], time; //时间戳
int stack[MAXN], top;    //用于维护连通块的
int n, m;    //点的个数和边的条数

void connect(int u, int v, int w) { //重边+id
    nEdge++;
    edge[nEdge].pre = pos[u];
    edge[nEdge].v = v;
    edge[nEdge].w = w;   //edge[nEdge].id = id;
    pos[u] = nEdge;
}

void tarjan(int cur, int from) {
    low[cur] = dfn[cur] = time++;
    stack[++top] = cur;
    for (int p=pos[cur]; p; p=edge[p].pre) {
        int v = edge[p].v;
        if (v == from) continue;   //注意一下这里.    如果是重边的话，改成：if (edge[p].id == from) continue; 给 tarjan 传参数 from 的时候，传递的是当前边的 id。
        if (!dfn[v]) {
            tarjan(v, cur);            //多重边：tarjan(v, edge[p].id);
            if (low[v] < low[cur]) low[cur] = low[v];
            if (low[v] > dfn[cur]) {
                bridge[tot].u = cur;
                bridge[tot++].v = v;
                cc++;
                do {
                    fa[stack[top]] = cc;
                } while (stack[top--] != v);
            }
        } else if (low[cur] > dfn[v]) low[cur] = dfn[v];
    }
}
int main() {
    scanf("%d%d", &n, &m);

    memset(pos, 0, sizeof(pos));
    nEdge = 0;
    int u, v, w;
    for (int i=0; i<m; i++) {
        scanf("%d%d%d", &u, &v, &w);
```

```
            connect(u, v, w);//重边+i
            connect(v, u, w);//重边+i
        }

        memset(dfn, 0, sizeof(dfn));
        memset(fa, -1, sizeof(fa));

        cc = tot = 0;
        for (int i=1; i<=n; i++)     //可以处理不连通的无向图，如果连通只需要一次即可
            if (!dfn[i]) {
                top = time = 1;
                tarjan(i, -1);
                ++cc;
                for (int j=1; j<=n; j++)     //特殊处理顶点的连通块
                    if (dfn[j] && fa[j]==-1) fa[j] = cc;
            }

        for (int i=1; i<=n; i++)
            printf("%d ", fa[i]);   //输出每个节点所属于的连通块（缩点标号）
        printf("\n");

        for (int i=0; i<tot; i++)
            printf("%d %d\n", bridge[i].u, bridge[i].v); //输出所有的桥

        return 0;
}
```

处理二维种类并查集

切点

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int n,m;
char ts[10];
int parent[200020];
int root(int p){
    if(parent[p]==-1) return p;
    else return parent[p]=root(parent[p]);
}
void merge(int a,int b){
    a=root(a);
    b=root(b);
    parent[a]=b;
}
int main()
{
    int T;
    scanf("%d",&T);
    while(T--)
    {
        int i,j;
        int ta,tb;
        memset(parent,-1,sizeof(parent));
        scanf("%d%d",&n,&m);
        while(m--)
        {
            scanf("%s%d%d",ts,&ta,&tb);
            if(ts[0]=='D')
            {
                if(root(ta*2)!=root(tb*2+1))
                    merge(ta*2,tb*2+1);
                if(root(ta*2+1)!=root(tb*2))
                    merge(ta*2+1,tb*2);
            }
            else
            {

                if(root(ta*2)==root(tb*2))
                    printf("In the same gang.\n");
                else if(root(ta*2)==root(tb*2+1))
                    printf("In different gangs.\n");
                else
                    printf("Not sure yet.\n");
            }
        }
    }
}
```

食物链

三类动物 A、B、C 构成食物链循环，告诉两个动物的关系（同类或天敌），判断那个关系是和前面已有的冲突。

并查集的高级应用。

假设 father[A]=B ，rank[A]=0 表示 A 与 B 同类；rank[A]=1 表示 B 可以吃 A；rank[A]=2 表示 A 可以吃 B。

对于一组数据 d x y，若 x 与 y 的根节点相同，利用 (rank[y]-rank[x]+3)%3!=d-1 若不相等说明为假话；

若 x 与 y 根节点不同，说明两者不在同一集合里，进行组合。让 x 的根 fx 成为 y 的根 fy 的父节点（father[fy]=fx），rank[fy]的值需要仔细归纳得出。

```
#include <stdio.h>
#include <memory.h>

#define MAXN 50001
int father[MAXN],rank[MAXN];

void Init(int n)
{
    int i;
    for(i=1;i<=n;i++)
        father[i]=i;
    memset(rank,0,sizeof(rank));
}

int Find_Set(int x)
{
    if(x!=father[x])
    {
        int fx=Find_Set(father[x]);
        rank[x]=(rank[x]+rank[father[x]])%3;    // 注意是 rank[father[x]]而不是 rank[fx]
        father[x]=fx;
    }
    return father[x];
}

bool Union(int x,int y,int type)
{
    int fx,fy;
    fx=Find_Set(x);
    fy=Find_Set(y);
    if(fx==fy)
    {
        if((rank[y]-rank[x]+3)%3!=type)return true;
    //这个关系可以通过举例得出
        else return false;
    }
    father[fy]=fx;
    rank[fy]=(rank[x]-rank[y]+type+3)%3;        // 与上式不同 需仔细归纳
    return false;
}

int main()
{
    freopen("in.txt","r",stdin);
    int n,k;
    int sum,i;
    int d,x,y;
    scanf("%d %d",&n,&k);
    //cin>>n>>k;
    Init(n);
    sum=0;
    for(i=0;i<k;i++)
    {
        scanf("%d %d %d",&d,&x,&y);
        //cin>>d>>x>>y;                          // 用 cin 会超时
        if(x>n || y>n ||(x==y && d==2))
            sum++;
        else if(Union(x,y,d-1))                 //传 d-1 方便关系式的表达
            sum++;
    }
    printf("%d\n",sum);
    return 0;
}
```

数论

最大公约与最小公倍

```
int LCM(int a,int b)//最小公倍数
{
    int x=a,y=b;
    int r=x%y;
    while(r>0)
    {
        x=y;
        y=r;
        r=x%y;
    }
    return a*b/y;
}

int GCD(int x,int y) //求最大公约数
{
    if(!x||!y) return x > y ? x : y;
    for (int t; t=x%y;x=y,y=t);
    return y;
}
```

```
#include<algorithm>
using namespace std;
int kgcd(int a,int b)//快速 GCD
{
	if(a==0) return b;
	if(b==0) return a;
	if(!(a&1) && !(b&1)) return kgcd (a>>1,b>>1) << 1;
	else if(!(b&1)) return kgcd(a,b>>1);
	else if(!(a&1)) return kgcd(a>>1,b);
	else return kgcd(abs(a-b),min(a,b));
}
```

高精度幂取模

```
/*
 * FZU1759.cpp
 *
 *   Created on: 2011-10-11
 *       Author: bjfuwangzhu
 * 格式为：
 * cha^chb%c
 * cha、chb 均为字符串
 */
#include<stdio.h>
#include<string.h>
#include<math.h>
#include<stdlib.h>
#define LL long long
#define nnum 1000005
#define nmax 31625
int flag[nmax], prime[nmax];
char cha[nnum], chb[nnum];
int plen;
void mkprime() {
	int i, j;
	memset(flag, -1, sizeof(flag));
	for (i = 2, plen = 0; i < nmax; i++) {
		if (flag[i]) {
			prime[plen++] = i;
		}
		for (j = 0; (j < plen) && (i * prime[j] < nmax); j++)
{
			flag[i * prime[j]] = 0;
			if (i % prime[j] == 0) {
				break;
			}
		}
	}
}
int getPhi(int n) {
	int i, te, phi;
	te = (int) sqrt(n * 1.0);
	for (i = 0, phi = n; (i < plen) && (prime[i] <= te); i++) {
		if (n % prime[i] == 0) {
			phi = phi / prime[i] * (prime[i] - 1);
			while (n % prime[i] == 0) {
				n /= prime[i];
			}
		}
	}
	if (n > 1) {
		phi = phi / n * (n - 1);
	}
	return phi;
}
int cmpBigNum(int p, char *ch) {
	int i, len;
	LL res;
	len = strlen(ch);
	for (i = 0, res = 0; i < len; i++) {
		res = (res * 10 + (ch[i] - '0'));
		if (res > p) {
			return 1;
		}
	}
	return 0;
}
int getModBigNum(int p, char *ch) {
	int i, len;
	LL res;
	len = strlen(ch);
	for (i = 0, res = 0; i < len; i++) {
		res = (res * 10 + (ch[i] - '0')) % p;
	}
	return (int) res;
}
int modular_exp(int a, int b, int c) {
	LL res, temp;
	res = 1 % c, temp = a % c;
	while (b) {
		if (b & 1) {
			res = res * temp % c;
		}
		temp = temp * temp % c;
		b >>= 1;
	}
```

```
    return (int) res;
}
void solve(int a, int c, char *ch) {
    int phi, res, b;
    phi = getPhi(c);
    if (cmpBigNum(phi, ch)) {
        b = getModBigNum(phi, ch) + phi;
    } else {
        b = atoi(ch);
    }
    res = modular_exp(a, b, c);
    printf("%d\n", res);
}
int main() {
    int a, c;
    mkprime();
    while (~scanf("%s %s %d", cha, chb, &c)) {
        a = getModBigNum(c, cha);
        solve(a, c, chb);
    }
    return 0;
}
```

快速幂取模

```
int quickpow(int m,int n,int k)
{
    int b =1;
    while (n >0)
    {
        if (n &1)
            b = (b*m)%k;
        n = n >>1 ;
        m = (m*m)%k;
    }
    return b;
}
```

矩阵快速幂求 Fibonacci

const long long MOD=1e9+7;//c++对于 const 有优化，要比非 const 快

```
struct Matrix
{
    long long mat[2][2];
    Matrix operator *(const Matrix &b)const
    {
        Matrix ret;
        for(int i = 0;i < 2;i++)
            for(int j = 0;j < 2;j++)
            {
                ret.mat[i][j] = 0;
                for(int k = 0;k < 2;k++)
                {
                    ret.mat[i][j] += (long long)mat[i][k]*b.mat[k][j]%MOD;
                    if(ret.mat[i][j] >= MOD)
                        ret.mat[i][j] -= MOD;
                }
            }
        return ret;
    }
};
Matrix pow_M(Matrix a,long long n)
{
    Matrix ret;
    memset(ret.mat,0,sizeof(ret.mat));
    ret.mat[0][0] = ret.mat[1][1] = 1;
    Matrix tmp = a;
    while(n)
    {
        if(n&1)ret = ret*tmp;
        tmp = tmp*tmp;
        n >>= 1;
    }
    return ret;
}

long long calc(long long n)
{
    Matrix A;
    A.mat[0][0] = 0;
    A.mat[1][0] = 1;
    A.mat[0][1] = 1;
    A.mat[1][1] = 1;
    A = pow_M(A,n);
    return A.mat[1][0];
}
```

大型质因数分解

注意 1.....

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

```
#include<stdlib.h>
#include<time.h>
#include<iostream>
#include<algorithm>
using namespace std;


//*****************************************************************
// Miller_Rabin 算法进行素数测试
//速度快，而且可以判断 <2^63 的数
//*****************************************************************
const int S=20;//随机算法判定次数，S 越大，判错概率越小


//计算 (a*b)%c.    a,b 都是 long long 的数，直接相乘可能溢出的
//   a,b,c <2^63
long long mult_mod(long long a,long long b,long long c)
{
    a%=c;
    b%=c;
    long long ret=0;
    while(b)
    {
        if(b&1){ret+=a;ret%=c;}
        a<<=1;
        if(a>=c)a%=c;
        b>>=1;
    }
    return ret;
}



//计算   x^n %c
long long pow_mod(long long x,long long n,long long mod)//x^n%c
{
    if(n==1)return x%mod;
    x%=mod;
    long long tmp=x;
    long long ret=1;
    while(n)
    {
        if(n&1) ret=mult_mod(ret,tmp,mod);
        tmp=mult_mod(tmp,tmp,mod);
        n>>=1;
    }
    return ret;
}



//以 a 为基,n-1=x*2^t      a^(n-1)=1(mod n)   验证 n 是不是合数
//一定是合数返回 true,不一定返回 false
bool check(long long a,long long n,long long x,long long t)
{
    long long ret=pow_mod(a,x,n);
    long long last=ret;
    for(int i=1;i<=t;i++)
    {
        ret=mult_mod(ret,ret,n);
        if(ret==1&&last!=1&&last!=n-1) return true;//合数
        last=ret;
    }
    if(ret!=1) return true;
    return false;
}

// Miller_Rabin()算法素数判定
//是素数返回 true.(可能是伪素数，但概率极小)
//合数返回 false;

bool Miller_Rabin(long long n)
{
    if(n<2)return false;
    if(n==2)return true;
    if((n&1)==0) return false;//偶数
    long long x=n-1;
    long long t=0;
    while((x&1)==0){x>>=1;t++;}
    for(int i=0;i<S;i++)
    {
        long long a=rand()%(n-1)+1;//rand()需要 stdlib.h 头文件
        if(check(a,n,x,t))
            return false;//合数
    }
    return true;
}
```

```
//**********************************************
**
//pollard_rho 算法进行质因数分解
//**********************************************
**
long long factor[100];//质因数分解结果（刚返回时是无序的）
int tol;//质因数的个数。数组小标从 0 开始

long long gcd(long long a,long long b)
{
    if(a==0)return 1;//???????
    if(a<0) return gcd(-a,b);
    while(b)
    {
        long long t=a%b;
        a=b;
        b=t;
    }
    return a;
}

long long Pollard_rho(long long x,long long c)
{
    long long i=1,k=2;
    long long x0=rand()%x;
    long long y=x0;
    while(1)
    {
        i++;
        x0=(mult_mod(x0,x0,x)+c)%x;
        long long d=gcd(y-x0,x);
        if(d!=1&&d!=x) return d;
        if(y==x0) return x;
        if(i==k){y=x0;k+=k;}
    }
}
//对 n 进行素因子分解
void findfac(long long n)
{
    if(Miller_Rabin(n))//素数
    {
        factor[tol++]=n;
        return;
    }
    long long p=n;
    while(p>=n)p=Pollard_rho(p,rand()%(n-1)+1);
    findfac(p);
    findfac(n/p);
}

int main()
{
    srand(time(NULL));//需要 time.h 头文件//POJ 上 G++不能加这句话
    long long n;
    while(scanf("%I64d",&n)!=EOF)
    {
        tol=0;
        if(n==1)continue;//1 既不是指数，也不是合数，需要特殊处理
        findfac(n);
        for(int i=0;i<tol;i++)printf("%I64d ",factor[i]);
        printf("\n");
        if(Miller_Rabin(n))printf("Yes\n");
        else printf("No\n");
    }
    return 0;
}
```

素数打表与素数单个判断

```
bool primeVis[10100000];  //i 是不是素数，从 primeVis[2]开始有效
int prime[1000000];  //prime[1]=2,prime[2]=3...
int primeNum = 0; //N 以内素数个数（1.2...primeNum）
void init_prim(int N)
{
    memset(primeVis, true, sizeof(primeVis));
    for (int i = 2; i <= N; ++i)
    {
        if (primeVis[i] == true)
        {
            primeNum++;
            prime[primeNum] = i;
        }
        for (int j = 1; ((j <= primeNum) && (i * prime[j] <= N));  ++j)
        {
            primeVis[i * prime[j]] = false;
            if (i % prime[j] == 0) break; //点睛之笔
        }
    }
}

bool judge_prime(int temp)//判断一个数是否为质数,返
```

```
回 1 为素数，0 为合数
{
    for(int i=2;i*i<=temp;i++)
        if(temp%i==0)
            return false;
    return true;
}
```

斜率优化 dp

最大区间平均值

```
#include<stdio.h>
#include<iostream>
#include<string.h>
#include<queue>
#include<algorithm>
using namespace std;
const int MAXN=100010;
double sum[MAXN];
int a[MAXN];
int q[MAXN];
int head,tail;

double max(double a,double b)
{
    if(a>b)return a;
    else return b;
}

double getUP(int i,int j)//i>j
{
    return sum[i]-sum[j];
}
int getDOWN(int i,int j)
{
    return i-j;
}



inline int input()
{
    char ch=' ';
    while(ch<'0'||ch>'9')ch=getchar();
    int x=0;
    while(ch<='9'&&ch>='0')x=x*10+ch-'0',ch=getchar();
    return x;
}

int main()
{
    //freopen("in.txt","r",stdin);
    //freopen("out.txt","w",stdout);
    int n,k;
    while(scanf("%d%d",&n,&k)!=EOF)
    {
        sum[0]=0;
        for(int i=1;i<=n;i++)
        {
            scanf("%d",&a[i]);
            sum[i]=sum[i-1]+a[i];
        }
        head=tail=0;
        q[tail++]=0;
        double ans=0;
        for(int i=k;i<=n;i++)
        {

while(head+1<tail&&getUP(i,q[head])*getDOWN(i,q[head
+1])<=getUP(i,q[head+1])*getDOWN(i,q[head]))
                head++;

ans=max(ans,getUP(i,q[head])/getDOWN(i,q[head]));

            int j=i-k+1;
            while(head+1<tail&&getUP(j,q[tail-
1])*getDOWN(q[tail-1],q[tail-2])<=getUP(q[tail-1],q[tail-
2])*getDOWN(j,q[tail-1]))
                tail--;
            q[tail++]=j;

        }
        printf("%.2lf\n",ans);
    }
    return 0;
}
```

Poj1260

题意：要买若干种价值的珍珠，但买某种珍珠必须多付 10 颗此种珍珠的价钱，及如果买价值为 1 的珍珠 100 颗，必须付的钱数为 110。一颗珍珠可以用比它贵的珍珠 充数，因此买多种珍珠的时候用贵的代替便宜的可能更省

钱。例如买 100 颗价值为 2 的、1 颗价值为 1 的，此时买 101 颗价值为 2 的为较优方案。输入要买的若 干种珍珠，可用高价珍珠充数的条件下，问最少需要花费多少钱。

经典 dp：

高档次的珍珠若要和低档次的珍珠合并，那么这些低档次的珍珠必须是和它紧邻的，也就是说，它只有和第 i—1，i—2，…，i—j 类珍珠合并，同时这种合并必须是连续的。用 sum[i] 记录前 i 类珍珠的总数。推出状态转移方程：

dp[i] = min (dp[i], (sum[i] - sum[j] + 10) * p[i] + dp[j]);

```
//////////////////////////////////////////////////////////////
#include<stdio.h>
#include<iostream>
#include<string.h>
#include<queue>
using namespace std;

int dp[105];
int q[105];
int sum[105];
int cost[105];

int head,tail,n;


int getDP(int i,int k)
{
    return dp[k]+(sum[i]-sum[k]+10)*cost[i];
}

int getUP(int k1,int k2)
{
    return dp[k1]-dp[k2];
}
int getDOWN(int k1,int  k2)
{
    return sum[k1]-sum[k2];
}

int main()
{
    int T;
    scanf("%d",&T);
    while(T--)
    {
        scanf("%d",&n);
        int i,j;
        int ta,tb;
        sum[0]=cost[0]=0;
        for(i=1;i<=n;i++)
        {
            scanf("%d%d",&ta,&tb);
            sum[i]=sum[i-1]+ta;
            cost[i]=tb;
        }
        dp[0]=0;
        head=tail=0;
        q[tail++]=0;
        for(i=1;i<=n;i++)
        {
            while(head+1<tail &&
getUP(q[head+1],q[head])<=cost[i]*getDOWN(q[head+1],
q[head]))
                head++;
            dp[i]=getDP(i,q[head]);
            while(head+1<tail && getUP(i,q[tail-
1])*getDOWN(q[tail-1],q[tail-2])<=getUP(q[tail-1],q[tail-
2])*getDOWN(i,q[tail-1]))
                tail--;
            q[tail++]=i;
        }
        printf("%d\n",dp[n]);
    }
    return 0;
}
```

poj 3709 K-Anonymous Sequence 斜率优化 dp 分组有大小限制

题意是给你一个 n 长度递增数列，将其分组，每组不少于 m 个，每组的 cost 是每组中所有元素减去里面最小元素的值的总和，要求你算最小的 cost。我想说这类的分组问题一般都是 dp 解决的，这题也不例外，dp 状态也很好设定，dp[i]表示前 i 个元素的 cost 最小值，当然是已经分好组了，因为没限定你要分几组就是一维的状态，但是把状态方程列出来后就会发现一个问题，状态转移方程 dp[i]=dp[tem]+sum[i]-sum[tem]-(i-tem)*a[tem+1];m<tem<i; 这个复杂度是 n*mn<500000,m<n,显然超时，这时就要用到斜率+单调队列优化

分组，并且每组的大小有限制

注意不一定要分满组才最小，要取 min

```
#include<stdio.h>
#include<iostream>
#include<string.h>
#include<queue>
using namespace std;
const int MAXN=500010;
#define ll __int64

ll dp[MAXN];
ll a[MAXN];
int q[MAXN];//队列
ll sum[MAXN];

int head,tail,n,m;

ll getDP(int i,int k)
{
    return dp[k]+sum[i]-sum[k]-a[k+1]*(i-k);
}

ll getUP(int k1,int k2) //yj-yk 部分
{
    return (dp[k1]-sum[k1]+a[k1+1]*k1)-(dp[k2]-sum[k2]+a[k2+1]*k2);
}
ll getDOWN(int k1,int  k2)
{
    return a[k1+1]-a[k2+1];
}

int main()
{
    int T;
    scanf("%d",&T);
    while(T--)
    {
        scanf("%d%d",&n,&m);
        int i,j;
        sum[0]=0;
        for(i=1;i<=n;i++)
            scanf("%I64d",&a[i]);
        sort(a+1,a+n+1);
        for(i=1;i<=n;i++)
            sum[i]=sum[i-1]+a[i];
        head=tail=0;
        dp[0]=0;
        for(i=1;i<m*2&&i<=n;i++)
            dp[i]=getDP(i,0);
        q[tail++]=m;
        for(i=m*2;i<=n;i++)
        {
            //把斜率转成相乘，注意顺序，否则不等号方向会改变的
            while(head+1<tail && getUP(q[head+1],q[head])<=i*getDOWN(q[head+1],q[head]))
                head++;
            dp[i]=getDP(i,q[head]);

            while(head+1<tail && getUP(i-m+1,q[tail-1])*getDOWN(q[tail-1],q[tail-2])<=getUP(q[tail-1],q[tail-2])*getDOWN(i-m+1,q[tail-1]))
                tail--;
            q[tail++]=i-m+1;
        }
        printf("%I64d\n",dp[n]);
    }
    return 0;
}
```

概率 dp

Hdu4035 树上概率 dp

```
/*
转自 kuangbin
HDU 4035

    dp 求期望的题。
    题意：
    有 n 个房间，由 n-1 条隧道连通起来，实际上就形成了一棵树，
    从结点 1 出发，开始走，在每个结点 i 都有 3 种可能：
        1.被杀死，回到结点 1 处（概率为 ki）
        2.找到出口，走出迷宫 （概率为 ei）
        3.和该点相连有 m 条边，随机走一条
    求：走出迷宫所要走的边数的期望值。

    设 E[i]表示在结点 i 处，要走出迷宫所要走的边数的期望。E[1]即为所求。
```

叶子结点：

E[i] = ki*E[1] + ei*0 + (1-ki-ei)*(E[father[i]] + 1);

= ki*E[1] + (1-ki-ei)*E[father[i]] + (1-ki-ei);

非叶子结点：（m 为与结点相连的边数）

E[i] = ki*E[1] + ei*0 + (1-ki-ei)/m*(E[father[i]]+1 + ∑(E[child[i]]+1));

= ki*E[1] + (1-ki-ei)/m*E[father[i]] + (1-ki-ei)/m*∑(E[child[i]]) + (1-ki-ei);

设对每个结点：E[i] = Ai*E[1] + Bi*E[father[i]] + Ci;

对于非叶子结点 i，设 j 为 i 的孩子结点，则

∑(E[child[i]]) = ∑E[j]

= ∑(Aj*E[1] + Bj*E[father[j]] + Cj)

= ∑(Aj*E[1] + Bj*E[i] + Cj)

带入上面的式子得

(1 - (1-ki-ei)/m*∑Bj)*E[i] = (ki+(1-ki-ei)/m*∑Aj)*E[1] + (1-ki-ei)/m*E[father[i]] + (1-ki-ei) + (1-ki-ei)/m*∑Cj;

由此可得

Ai = (ki+(1-ki-ei)/m*∑Aj) / (1 - (1-ki-ei)/m*∑Bj);

Bi = (1-ki-ei)/m / (1 - (1-ki-ei)/m*∑Bj);

Ci = ((1-ki-ei)+(1-ki-ei)/m*∑Cj) / (1 - (1-ki-ei)/m*∑Bj);

对于叶子结点

Ai = ki;

Bi = 1 - ki - ei;

Ci = 1 - ki - ei;

从叶子结点开始，直到算出 A1,B1,C1;

E[1] = A1*E[1] + B1*0 + C1;

所以

E[1] = C1 / (1 - A1);

若 A1 趋近于 1 则无解...

*/

```
#include<iostream>
#include<algorithm>
#include<iostream>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<vector>
#include<queue>
#include<cmath>
using namespace std;
#define ll long long
#define inf 0x3f3f3f3f
#define maxnode 1000010
#define sigma_size 26
const int dir[4][2]={0,1,0,-1,1,0,-1,0};

int n;
vector<int>pq[10005];

struct Dian
{
    double a,b,c;
    double k,e;
}dians[10005];

void dfs(int temp,int father)
{
    if(pq[temp].size()==1&&father!=-1)
    {
        dians[temp].a=dians[temp].k;
        dians[temp].b=dians[temp].c=1-dians[temp].e-dians[temp].k;
        return;
    }
    double sa=0,sb=0,sc=0,fuck;
    int m=0;
    for(int i=0;i<pq[temp].size();i++)
    {
        m++;
        if(pq[temp][i]!=father)
        {
            dfs(pq[temp][i],temp);
            sa+=dians[pq[temp][i]].a;
            sb+=dians[pq[temp][i]].b;
            sc+=dians[pq[temp][i]].c;
        }
    }
    fuck=(1.0-(1.0-dians[temp].k-dians[temp].e)/m*sb);
    dians[temp].a=(dians[temp].k+(1.0-dians[temp].k-dians[temp].e)/m*sa)/fuck;
    dians[temp].b=((1.0-dians[temp].k-dians[temp].e)/m)/fuck;
    dians[temp].c=((1.0-dians[temp].k-dians[temp].e)+(1.0-dians[temp].k-dians[temp].e)/m*sc)/fuck;
}

int main()
```

```
{
    int T,R=0;
    scanf("%d",&T);
    while(T--)
    {
        R++;
        int i,j;
        int ta,tb;
        scanf("%d",&n);
        for(i=1;i<=n;i++)
            pq[i].clear();
        for(i=1;i<n;i++)
        {
            scanf("%d%d",&ta,&tb);
            pq[ta].push_back(tb);
            pq[tb].push_back(ta);
        }
        for(i=1;i<=n;i++)
        {
            scanf("%d%d",&ta,&tb);
            dians[i].k=ta/100.0;
            dians[i].e=tb/100.0;
        }
        dfs(1,-1);
        if(fabs(1-dians[1].a)<0.0000000001)
            printf("Case %d: impossible\n",R);
        else
            printf("Case  %d:  %.6lf\n",R,dians[1].c/(1.0-dians[1].a));
    }
    return 0;
}
```

倒着的概率

```
/*
HDU 4098
```

题意：有 n 个人排队等着在官网上激活游戏。Tomato 排在第 m 个。

对于队列中的第一个人。有一下情况：

1、激活失败，留在队列中等待下一次激活（概率为 p1)

2、失去连接，出队列，然后排在队列的最后（概率为 p2）

3、激活成功，离开队列（概率为 p3）

4、服务器瘫痪，服务器停止激活，所有人都无法激活了。

求服务器瘫痪时 Tomato 在队列中的位置<=k 的概率

解析：

概率 DP：

设 dp[i][j]表示 i 个人排队,Tomato 排在第 j 个位置，达到目标状态的概率(j<=i)

dp[n][m]就是所求

j==1: dp[i][1]=p1*dp[i][1]+p2*dp[i][i]+p4;

2<=j<=k: dp[i][j]=p1*dp[i][j]+p2*dp[i][j-1]+p3*dp[i-1][j-1]+p4;

k<j<=i: dp[i][j]=p1*dp[i][j]+p2*dp[i][j-1]+p3*dp[i-1][j-1];

化简：

j==1: dp[i][1]=p*dp[i][i]+p41;

2<=j<=k: dp[i][j]=p*dp[i][j-1]+p31*dp[i-1][j-1]+p41;

k<j<=i: dp[i][j]=p*dp[i][j-1]+p31*dp[i-1][j-1];

其中:

p=p2/(1-p1);

p31=p3/(1-p1)

p41=p4/(1-p1)

可以循环 i=1->n 递推求解 dp[i].在求解 dp[i]的时候 dp[i-1]就相当于常数了。

在求解 dp[i][1~i]时等到下列 i 个方程

j==1: dp[i][1]=p*dp[i][i]+c[1];

2<=j<=k:dp[i][j]=p*dp[i][j-1]+c[j];

k<j=i: dp[i][j]=p*dp[i][j]+c[j];

其中 c[j]都是常数了。上述方程可以解出 dp[i]了。

首先是迭代得到 dp[i][i].然后再代入就可以得到所有的 dp[i]了。

注意特判一种情况。就是 p4<eps 时候，就不会崩溃了，应该直接输出 0

```
#include<iostream>
#include<algorithm>
#include<iostream>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<vector>
#include<queue>
#include<cmath>
using namespace std;
#define ll long long
#define inf 0x3f3f3f3f
int dir[4][2]={0,1,0,-1,1,0,-1,0};
const double eps=1e-8;

double c[2005];
double pp[2005];
double dp[2005][2005];
int n,m,k;
double p1,p2,p3,p4,p,p31,p41;
```

```
int main()
{
    while(scanf("%d%d%d%lf%lf%lf%lf",&n,&m,&k,&p1,&p2,&p3,&p4)!=EOF)
    {
        if(p4<eps)
        {
            printf("0.00000\n");
            continue;
        }
        memset(dp,0,sizeof(dp));
        memset(c,0,sizeof(c));
        int i,j;
        p=p2/(1.0-p1);
        p31=p3/(1.0-p1);
        p41=p4/(1.0-p1);
        pp[0]=1.0;
        for(int i=1;i<=n;i++) pp[i]=p*pp[i-1];
        dp[1][1]=p41/(1-p);
        c[1]=p41;
        for(int i=2;i<=n;i++)
        {
            for(int j=2;j<=k&&j<=i;j++)
                c[j]=p31*dp[i-1][j-1]+p41;
            for(int j=k+1;j<=i;j++)
                c[j]=p31*dp[i-1][j-1];
            double tmp=c[1]*pp[i-1];
            for(int j=2;j<=i;j++)
                tmp+=c[j]*pp[i-j];
            dp[i][i]=tmp/(1-pp[i]);
            dp[i][1]=p*dp[i][i]+c[1];
            for(int j=2;j<i;j++)
                dp[i][j]=p*dp[i][j-1]+c[j];
        }
        printf("%.5lf\n",dp[n][m]);
    }
    return 0;
}
```

Hdu4870

概率方程很好写，但是有环

可以看出来 dp[0][k]-dp[k][k]是一组

分组求，可以设参数也可以高斯消元

```
#include<stdio.h>
#include<iostream>
#include<string.h>
#include<algorithm>
#include<math.h>
#include<queue>
using namespace std;

double p,q;
struct Dian
{
    double a;
    double b;
}dp[25];

int main()
{
    while(scanf("%lf",&p)!=EOF)
    {
        q=1-p;
        int i,j;
        double temp=0,yu;
        for(i=19;i>0;i--)
        {
            dp[0].a=1;dp[0].b=0;
            dp[1].a=(1-q)/p;dp[1].b=-(1/p);
            dp[2].a=(dp[1].a-q)/p;dp[2].b=(dp[1].b-1)/p;
            for(j=3;j<=i+1;j++)
            {
                dp[j].a=(dp[j-1].a-q*dp[j-3].a)/p;
                dp[j].b=(dp[j-1].b-q*dp[j-3].b-1)/p;
            }
            yu=(temp-dp[i+1].b)/dp[i+1].a;
            temp=dp[i-1].a*yu+dp[i-1].b;
        }
        printf("%.8lf\n",(p*temp+1)/(1-q));
    }
    return 0;
}
```

线段树扫描线

poj 1151 Atlantis 纯矩形面积并

```
#include<algorithm>
#include<iostream>
#include<stdio.h>
```

```
#include<string.h>
#include<stdlib.h>
#include<vector>
#include<queue>
#include<cmath>
#include<math.h>
using namespace std;
#define inf 0x3f3f3f3f
const int dir[4][2]={0,1,0,-1,1,0,-1,0};
#define maxn 205

struct Tree
{
    int l,r;//左右端点
    int c;//覆盖情况
    double cnt,lf,rf;//cnt 为长度，lf 为实际左坐标，rf 为实际右坐标
}tree[maxn*4];

struct Line
{
    double x,y1,y2;
    int f;//直线的标记
}line[maxn];

bool cmp(Line a,Line b)//sort 排序的函数
{
    return a.x < b.x;
}

double y[maxn];//记录 y 坐标

void build(int l,int r,int root)
{
    tree[root].l=l;tree[root].r=r;
    tree[root].c=tree[root].cnt=0;
    tree[root].lf=y[l];
    tree[root].rf=y[r];
    if(l+1==r)return;
    int mid=(l+r)>>1;
    build(l,mid,root<<1);
    build(mid,r,root<<1|1);
}

void calen(int root)
{
    if(tree[root].c>0)
    {
        tree[root].cnt=tree[root].rf-tree[root].lf;
        return ;
    }
    if(tree[root].l+1==tree[root].r) tree[root].cnt=0;
    else
tree[root].cnt=tree[root<<1].cnt+tree[root<<1|1].cnt;
}

void update(int root,struct Line e)
{
    if(e.y1==tree[root].lf&&e.y2==tree[root].rf)
    {
        tree[root].c+=e.f;
        calen(root);
        return;
    }
    if(e.y2<=tree[root<<1].rf)update(root<<1,e);
    else if(e.y1>=tree[root<<1|1].lf)update(root<<1|1,e);
    else
    {
        Line tmp=e;
        tmp.y2=tree[root<<1].rf;
        update(root<<1,tmp);
        tmp=e;
        tmp.y1=tree[root<<1|1].lf;
        update(root<<1|1,tmp);
    }
    calen(root);
}

int main()
{
    int i,j;
    int n,k,R=0;
    double x1,y1,x2,y2;
    while(scanf("%d",&n)!=EOF&&n!=0)
    {
        R++;
        k=1;
        for(i=1;i<=n;i++)
        {
            scanf("%lf%lf%lf%lf",&x1,&y1,&x2,&y2);
            line[k].x=x1;
            line[k].y1=y1;
            line[k].y2=y2;
            line[k].f=1;
            y[k]=y1;
            k++;
            line[k].x=x2;
            line[k].y1=y1;
```

```
            line[k].y2=y2;
            line[k].f=-1;
            y[k]=y2;
            k++;
        }
        k--;
        sort(line+1,line+k+1,cmp);
        sort(y+1,y+k+1);
        int mm;
        mm=unique(y+1,y+1+k)-y-1;
        build(1,mm,1);
        update(1,line[1]);
        double ans=0;
        for(i=2;i<=k;i++)
        {
            ans+=tree[1].cnt*(line[i].x-line[i-1].x);
            update(1,line[i]);
        }
        printf("Test case #%d\nTotal explored area: %.2lf\n\n",R,ans);
    }
    return 0;
}
```

hdu 1255 覆盖的面积

与 poj1151 略有不同 由于是大于等于 2 的有效 所以应该更新到点

```
#include<iostream>
#include<algorithm>
#include<iostream>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<vector>
#include<queue>
#include<cmath>
#include<math.h>
using namespace std;
#define inf 0x3f3f3f3f
const int dir[4][2]={0,1,0,-1,1,0,-1,0};
#define maxn 2005

struct Tree
{
    int l,r;//左右端点
    int c;//覆盖情况
    double cnt,lf,rf;//cnt 为长度，lf 为实际左坐标，rf 为实际右坐标
}tree[maxn*4];

struct Line
{
    double x,y1,y2;
    int f;//直线的标记
}line[maxn];

bool cmp(Line a,Line b)//sort 排序的函数
{
    return a.x < b.x;
}

double y[maxn];//记录 y 坐标

void build(int l,int r,int root)
{
    tree[root].l=l;tree[root].r=r;
    tree[root].c=tree[root].cnt=0;
    tree[root].lf=y[l];
    tree[root].rf=y[r];
    if(l+1==r)return;
    int mid=(l+r)>>1;
    build(l,mid,root<<1);
    build(mid,r,root<<1|1);
}

void calen(int root)
{
    if(tree[root].c>=2)
    {
        tree[root].cnt=tree[root].rf-tree[root].lf;
        return ;
    }
    if(tree[root].l+1==tree[root].r) tree[root].cnt=0;
    else
tree[root].cnt=tree[root<<1].cnt+tree[root<<1|1].cnt;
}

void update(int root,struct Line e)
{
    if(tree[root].l+1==tree[root].r)
    {
        tree[root].c+=e.f;
        calen(root);
        return;
    }
    if(e.y2<=tree[root<<1].rf)update(root<<1,e);
```

```
    else if(e.y1>=tree[root<<1|1].lf)update(root<<1|1,e);
    else
    {
        Line tmp=e;
        tmp.y2=tree[root<<1].rf;
        update(root<<1,tmp);
        tmp=e;
        tmp.y1=tree[root<<1|1].lf;
        update(root<<1|1,tmp);
    }
    calen(root);
}

int main()
{
    int i,j;
    int n,k,R=0,T;
    scanf("%d",&T);
    double x1,y1,x2,y2;
    while(T--)
    {
        R++;
        scanf("%d",&n);
        k=1;
        for(i=1;i<=n;i++)
        {
            scanf("%lf%lf%lf%lf",&x1,&y1,&x2,&y2);
            line[k].x=x1;
            line[k].y1=y1;
            line[k].y2=y2;
            line[k].f=1;
            y[k]=y1;
            k++;
            line[k].x=x2;
            line[k].y1=y1;
            line[k].y2=y2;
            line[k].f=-1;
            y[k]=y2;
            k++;
        }
        k--;
        sort(line+1,line+k+1,cmp);
        sort(y+1,y+k+1);
        build(1,k,1);
        update(1,line[1]);
        double ans=0;
        for(i=2;i<=k;i++)
        {
            ans+=tree[1].cnt*(line[i].x-line[i-1].x);
            update(1,line[i]);
        }
        printf("%.2lf\n",ans);
    }
    return 0;
}
```

poj 1177 矩形并的周长

可以扫一遍，同时求长和宽的增加段；也可以横纵各扫一遍求边的变化量；

```
#include<algorithm>
#include<iostream>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<vector>
#include<queue>
#include<cmath>
#include<math.h>
using namespace std;
#define inf 0x3f3f3f3f
const int dir[4][2]={0,1,0,-1,1,0,-1,0};
#define maxn 10005

struct Tree
{
    int l,r;//左右端点
    int c;//覆盖情况
    int cnt,lf,rf;//cnt 为长度，lf 为实际左坐标，rf 为实际右坐标
    int se;
    int lc,rc;
}tree[maxn*4];

struct Line
{
    int x,y1,y2;
    int f;//直线的标记
}line[maxn];

bool cmp(Line a,Line b)//sort 排序的函数
{
    return a.x < b.x;
}

double y[maxn];//记录 y 坐标
```

```
void build(int l,int r,int root)
{
	tree[root].l=l;tree[root].r=r;
	tree[root].c=tree[root].cnt=tree[root].se=0;
	tree[root].lc=tree[root].rc=0;
	tree[root].lf=y[l];
	tree[root].rf=y[r];
	if(l+1==r)return;
	int mid=(l+r)>>1;
	build(l,mid,root<<1);
	build(mid,r,root<<1|1);
}

void calen(int root)
{
	if(tree[root].c>0)
	{
		tree[root].cnt=tree[root].rf-tree[root].lf;
		tree[root].se=1;
		tree[root].lc=tree[root].rc=1;
	}
	else if(tree[root].l+1==tree[root].r)
	{
		tree[root].cnt=0;
		tree[root].se=0;
		tree[root].lc=tree[root].rc=0;
	}
	else
	{

	tree[root].cnt=tree[root<<1].cnt+tree[root<<1|1].cnt;
		tree[root].lc=tree[root<<1].lc;
		tree[root].rc=tree[root<<1|1].rc;

	tree[root].se=tree[root<<1].se+tree[root<<1|1].se-
tree[root<<1].rc*tree[root<<1|1].lc;
	}
}

void update(int root,struct Line e)
{
	if(e.y1==tree[root].lf&&e.y2==tree[root].rf)
	{
		tree[root].c+=e.f;
		calen(root);
		return;
	}
	if(e.y2<=tree[root<<1].rf)update(root<<1,e);
	else if(e.y1>=tree[root<<1|1].lf)update(root<<1|1,e);
	else
	{
		Line tmp=e;
		tmp.y2=tree[root<<1].rf;
		update(root<<1,tmp);
		tmp=e;
		tmp.y1=tree[root<<1|1].lf;
		update(root<<1|1,tmp);
	}
	calen(root);
}

int main()
{
	int i,j;
	int n,k,R=0;
	int x1,y1,x2,y2;
	while(scanf("%d",&n)!=EOF&&n!=0)
	{
		R++;
		k=1;
		for(i=1;i<=n;i++)
		{
			scanf("%d%d%d%d",&x1,&y1,&x2,&y2);
			line[k].x=x1;
			line[k].y1=y1;
			line[k].y2=y2;
			line[k].f=1;
			y[k]=y1;
			k++;
			line[k].x=x2;
			line[k].y1=y1;
			line[k].y2=y2;
			line[k].f=-1;
			y[k]=y2;
			k++;
		}
		k--;
		sort(line+1,line+k+1,cmp);
		sort(y+1,y+k+1);
		int mm;
		mm=unique(y+1,y+1+k)-y-1;
		build(1,mm,1);
		int ans=0,temp=0;
		for(i=1;i<=k;i++)
		{
			if(tree[1].cnt!=0)
				ans+=tree[1].se*(line[i].x-line[i-1].x)*2;
			update(1,line[i]);
```

```
                ans+=abs(tree[1].cnt-temp);
                temp=tree[1].cnt;
            }
            printf("%d\n",ans);
        }
        return 0;
}
```

hdu 4007 Dave 求矩形圈点最大值

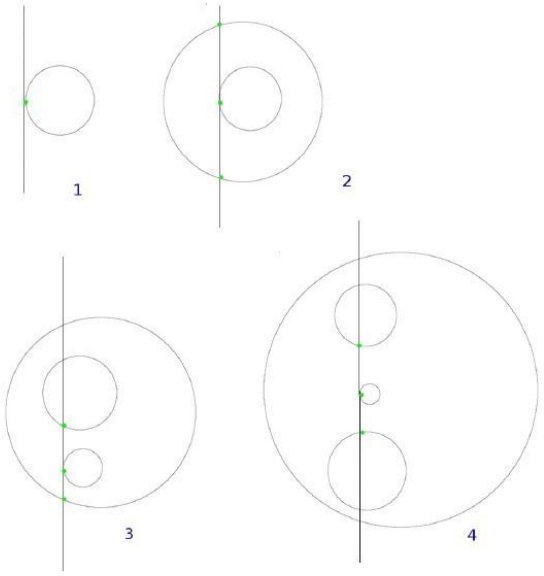
对于每个点，扩成一个矩形，变成求最大重叠层数；
注意边界的处理，这里是横纵均加一；
可能在边界上会产生一些问题，在排序的时候修正；

```
#include<algorithm>
#include<iostream>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<vector>
#include<queue>
#include<cmath>
#include<math.h>
using namespace std;
#define inf 0x3f3f3f3f
const int dir[4][2]={0,1,0,-1,1,0,-1,0};
#define maxn 2005

struct Tree
{
    int l,r;//左右端点
    int c;//覆盖情况
    int cnt,lf,rf;//cnt 为层数，lf 为实际左坐标，rf 为实际右坐标
}tree[maxn*4];

struct Line
{
    int x,y1,y2;
    int f;//直线的标记
}line[maxn];

bool cmp(Line a,Line b)//sort 排序的函数
{
    if(a.x!=b.x)
        return a.x < b.x;
    else
    return a.f < b.f;
}

int y[maxn];//记录 y 坐标

void build(int l,int r,int root)
{
    tree[root].l=l;tree[root].r=r;
    tree[root].c=tree[root].cnt=0;
    tree[root].lf=y[l];
    tree[root].rf=y[r];
    if(l+1==r)return;
    int mid=(l+r)>>1;
    build(l,mid,root<<1);
    build(mid,r,root<<1|1);
}

void calen(int root)
{
    if(tree[root].l+1==tree[root].r)
tree[root].cnt=tree[root].c;
    else
tree[root].cnt=max(tree[root<<1].cnt,tree[root<<1|1].cnt)
+tree[root].c;
}

void update(int root,struct Line e)
{
    if(e.y1==tree[root].lf&&e.y2==tree[root].rf)
    {
        tree[root].c+=e.f;
        calen(root);
        return;
    }
    if(e.y2<=tree[root<<1].rf)update(root<<1,e);
    else if(e.y1>=tree[root<<1|1].lf)update(root<<1|1,e);
    else
    {
        Line tmp=e;
        tmp.y2=tree[root<<1].rf;
        update(root<<1,tmp);
        tmp=e;
        tmp.y1=tree[root<<1|1].lf;
        update(root<<1|1,tmp);
    }
    calen(root);
}

int main()
{
    int i,j;
```

```
    int n,r,k;
    int x1,y1,x2,y2;
    while(scanf("%d%d",&n,&r)!=EOF)
    {
        k=1;
        for(i=1;i<=n;i++)
        {
            scanf("%d%d",&x1,&y1);
            x2=x1+r+1;
            y2=y1+r+1;
            line[k].x=x1;
            line[k].y1=y1;
            line[k].y2=y2;
            line[k].f=1;
            y[k]=y1;
            k++;
            line[k].x=x2;
            line[k].y1=y1;
            line[k].y2=y2;
            line[k].f=-1;
            y[k]=y2;
            k++;
        }
        k--;
        sort(line+1,line+k+1,cmp);
        sort(y+1,y+k+1);
        int mm;
        mm=unique(y+1,y+1+k)-y-1;
        build(1,mm,1);
        int ans=0;
        for(i=1;i<=k;i++)
        {
            update(1,line[i]);
            if(tree[1].cnt>ans)
                ans=tree[1].cnt;
        }
        printf("%d\n",ans);
    }
    return 0;
}
```

圆的扫描线

算法：扫描线
用一条竖直线从左到右扫描所有的圆，处理每个圆"刚接触扫描线"和"刚离开
扫描线"两个事件点。
为了下面描述方便，令某圆 A 的嵌套层数为 f(A)，如果某圆 A 被某圆 B 嵌套且 A 和 B
紧邻，那么说 A 是 B 的儿子，B 是 A 的父亲。如果圆 A，圆 B 同时是圆 C 的儿子，那么 A，
B 互为兄弟,当前考虑的圆为圆 C。



根据"刚接触扫描线"事件点的上下相邻事件点分类有如下情况：
1）没有上方事件点，或者没有下方事件点。这时该圆 C 的嵌套层数 f(C) = 1
2）上方事件点和下方事件点属于同一个圆 A，这时圆 A 必定是圆 C 的父亲，f(C) =
f(A) + 1
3）上方事件点和下方事件点分别属于两个圆 A，B，且 f(A) != f(B)，这里不妨
设 f(A) < f(B)，那么 A 是 C 的父亲，B 是 C 的兄弟。f(C) = f(A) + 1, f(C) = f(B)
4）上方事件点和下方事件点分别属于两个圆 A，B，且 f(A) == f(B),那么 A 是 C
的兄弟，B 是 C 的兄弟,f(C) = f(A) = f(B).
在处理"刚接触扫描线"事件点时插入一对点表示该圆与扫描线的相交情况，
并利用上述分类计算其嵌套层数，在处理" 刚离开扫描线"事件点是删除对应
的那一对点。可以采用 STL 中的 set 来维护

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <set>
#include <algorithm>

using namespace std;

const int UP = 0;
const int DOWN = 1;
const int IN = 0;
```

```
const int OUT = 1;
const int N = 50005;

int Time;

struct circle {
    int x, y, r;
    int w;
    void read() {
        scanf("%d %d %d", &x, &y, &r);
        w = 0;
    }
    int getX(int flag) {
        if( flag == IN )     return x - r;
        else                 return x + r;
    }
    double getY(int flag) {
        double ret = sqrt((double)r*r-
(double)(Time-x)*(Time-x));
        if( flag == UP )     return (double)y + ret;
        else                 return (double)y - ret;
    }

} cir[N];

struct event {
    int x, y, id;
    int flag;
    void get(int _x, int _y, int _id, int _flag) {
        x = _x;
        y = _y;
        id = _id;
        flag = _flag;
    }
    bool operator<(const event ev) const {
        return x < ev.x || x == ev.x && y > ev.y;
    }
} eve[N*2];

struct node {
    int id;
    int flag;
    node(){}
    node(int _id, int _flag) {
        id = _id;
        flag = _flag;
    }
    bool operator<(const node a) const {
        double y1 = cir[id].getY(flag);
        double y2 = cir[a.id].getY(a.flag);
        return y1 > y2 || y1 == y2 && flag < a.flag;
    }
};

int n, eveN;

set<node> line;
set<node>::iterator it, f, e, p;

inline int max(int a, int b) { return a > b ? a : b;}

void moveline() {
    line.clear();
    for(int i = 0; i < eveN; i ++) {
        Time = eve[i].x;
        if( eve[i].flag == OUT ) {
            line.erase(node(eve[i].id, UP));
            line.erase(node(eve[i].id, DOWN));
        } else {
            it = line.insert(node(eve[i].id, UP)).first;
            e = f = it;
            e ++;
            int id = it->id;
            if( it == line.begin() || e == line.end() )
{
                cir[id].w = 1;
            } else {
                f --;
                if( f->id == e->id ) {
                    cir[id].w = cir[f->id].w + 1;
                } else {
                    cir[id].w = max( cir[f->id].w, cir[e-
>id].w);
                }
            }
            line.insert(node(eve[i].id, DOWN));
        }
    }
}

int main() {
    while( scanf("%d", &n) != EOF ) {
        eveN = 0;
        for(int i = 0; i < n; i ++) {
            cir[i].read();
            eve[eveN++].get(cir[i].getX(IN), cir[i].y,
 i, IN);
            eve[eveN++].get(cir[i].getX(OUT), cir[i].
```

```
y, i, OUT);

        }
        sort(eve, eve + eveN);
        moveline();
        int ans = 0;
        for(int i = 0; i < n; i ++) {
            ans = max(ans, cir[i].w);
        }
        printf("%d\n", ans);
    }
}
```

基本初等函数求导公式

(1) $(C)'=0$　　(2) $(x^{\mu})'=\mu x^{\mu-1}$

(3) $(\sin x)'=\cos x$　　(4) $(\cos x)'=-\sin x$

(5) $(\tan x)'=\sec^2 x$　　(6) $(\cot x)'=-\csc^2 x$

(7) $(\sec x)'=\sec x\tan x$　　(8) $(\csc x)'=-\csc x\cot x$

(9) $(a^x)'=a^x\ln a$　　(10) $(e^x)'=e^x$

(11) $(\log_a x)'=\dfrac{1}{x\ln a}$　　(12) $(\ln x)'=\dfrac{1}{x}$,

(13) $(\arcsin x)'=\dfrac{1}{\sqrt{1-x^2}}$　　(14) $(\arccos x)'=-\dfrac{1}{\sqrt{1-x^2}}$

(15) $(\arctan x)'=\dfrac{1}{1+x^2}$　　(16) $(\operatorname{arc}\cot x)'=-\dfrac{1}{1+x^2}$

函数的和、差、积、商的求导法则

设 $u=u(x)$，$v=v(x)$ 都可导，则

(1) $(u\pm v)'=u'\pm v'$　　(2) $(Cu)'=Cu'$（C 是常

(3) $(uv)'=u'v+uv'$　　(4) $\left(\dfrac{u}{v}\right)'=\dfrac{u'v-uv'}{v^2}$

反函数求导法则

若函数 $x=\varphi(y)$ 在某区间 I_y 内可导、单调且 $\varphi'(y)\neq 0$，则它的反函数 $y=f(x)$ 在对应区间 I_x 内也可导，且

$$f'(x)=\frac{1}{\varphi'(y)}\quad 或\quad \frac{dy}{dx}=\frac{1}{\dfrac{dx}{dy}}$$