|  |
| --- |
| 浙江工业大学 |
| **zjut\_DD ACM/ICPC例程** |
| Personal Use Only |
|  |
| **王俊俏** |
| **2011/9/22** |

|  |
| --- |
| ACM/ICPC Routine Library |

1. 目录

[数论 7](#_Toc304453933)

[基础数论模板 7](#_Toc304453934)

[gcd lcm 7](#_Toc304453935)

[扩展欧几里得 7](#_Toc304453936)

[逆元 7](#_Toc304453937)

[a^b %mod 7](#_Toc304453938)

[欧拉函数 7](#_Toc304453939)

[解模方程 ax=b (mod n) 7](#_Toc304453940)

[解模线性方程组 8](#_Toc304453941)

[分解整数n 8](#_Toc304453942)

[求约数 8](#_Toc304453943)

[求原根 8](#_Toc304453944)

[大数质因数分解 9](#_Toc304453945)

[求sigma (n /i) 10](#_Toc304453946)

[AC的离散对数 10](#_Toc304453947)

[a^b的前几位 11](#_Toc304453948)

[大组合数取模 模质数 11](#_Toc304453949)

[大组合数取模,模P^t 12](#_Toc304453950)

[高斯解n元模方程组 13](#_Toc304453951)

[计算1~y中有几个gcd(a,i)=1 15](#_Toc304453952)

[中国剩余定理 15](#_Toc304453953)

[N!中0的个数 15](#_Toc304453954)

[数学 16](#_Toc304453955)

[多项式展开 16](#_Toc304453956)

[N进制位操作 17](#_Toc304453957)

[分数类 18](#_Toc304453958)

[矩阵 18](#_Toc304453959)

[计算 sigma(i^k) 19](#_Toc304453960)

[浮点型高斯消元 19](#_Toc304453961)

[模2域高斯消元 20](#_Toc304453962)

[复合Simpson积分公式 20](#_Toc304453963)

[计算行列式|A|模mod 21](#_Toc304453964)

[解Pell方程 x^2-N\*y^2=1 22](#_Toc304453965)

[求catalan数第n项 22](#_Toc304453966)

[线段下方的整点数目 23](#_Toc304453967)

[BigInteger 23](#_Toc304453968)

[图论 26](#_Toc304453969)

[邻接表类 26](#_Toc304453970)

[最小生成树 prim 27](#_Toc304453971)

[次小生成树 27](#_Toc304453972)

[单点k度限制最小生成树 28](#_Toc304453973)

[最短路dijkstra 30](#_Toc304453974)

[LCA离线算法 Tarjan 31](#_Toc304453975)

[LCA离线算法 Tarjan 手写递归 31](#_Toc304453976)

[LCA在线算法 rmq 32](#_Toc304453977)

[有向图强连通分量Kosaraju算法 33](#_Toc304453978)

[有向图强连通分量Tarjan算法 34](#_Toc304453979)

[无向图边双连通 34](#_Toc304453980)

[无向图点双连通 35](#_Toc304453981)

[2SAT 36](#_Toc304453982)

[最小树形图 38](#_Toc304453983)

[数据结构 40](#_Toc304453984)

[RMQ一维 40](#_Toc304453985)

[RMQ二维 40](#_Toc304453986)

[树状数组一维 41](#_Toc304453987)

[树状数组二维 42](#_Toc304453988)

[四向循环网状链表 42](#_Toc304453989)

[平衡树SBT 43](#_Toc304453990)

[平衡树Treap 44](#_Toc304453991)

[笛卡尔树 45](#_Toc304453992)

[伸展树 46](#_Toc304453993)

[动态树 48](#_Toc304453994)

[划分树 50](#_Toc304453995)

[左偏树 51](#_Toc304453996)

[线段树 下标 52](#_Toc304453997)

[线段树 指针 53](#_Toc304453998)

[四分树 55](#_Toc304453999)

[快速vector模板 56](#_Toc304454000)

[vector做平衡树 56](#_Toc304454001)

[哈希表 57](#_Toc304454002)

[单调队列,单调dp 57](#_Toc304454003)

[凸队列 斜率dp 58](#_Toc304454004)

[STL priority\_queue 58](#_Toc304454005)

[快速 queue 59](#_Toc304454006)

[字符串 59](#_Toc304454007)

[最长回文 O(n) 59](#_Toc304454008)

[kmp 60](#_Toc304454009)

[扩展kmp 60](#_Toc304454010)

[串的最小表示 kmp求循环节 61](#_Toc304454011)

[AC自动机 62](#_Toc304454012)

[AC自动机dp 63](#_Toc304454013)

[后缀数组 64](#_Toc304454014)

[不同子串个数nlog(n) 65](#_Toc304454015)

[网络流 65](#_Toc304454016)

[最大流dinic 邻接表 65](#_Toc304454017)

[最大流dinic 邻接阵 66](#_Toc304454018)

[最大流ek 改流量 67](#_Toc304454019)

[无向图最小割 68](#_Toc304454020)

[无源汇上下界可行流 69](#_Toc304454021)

[有源汇上下界可行流 70](#_Toc304454022)

[有源汇上下界最大流 邻接阵 72](#_Toc304454023)

[有源汇上下界最大流 邻接表 73](#_Toc304454024)

[有源汇上下界最小流 75](#_Toc304454025)

[最小费用最大流 77](#_Toc304454026)

[总结 78](#_Toc304454027)

[搜索算法 78](#_Toc304454028)

[dlx精确覆盖 78](#_Toc304454029)

[dlx重复覆盖 80](#_Toc304454030)

[A\*算法伪代码 82](#_Toc304454031)

[计算几何 82](#_Toc304454032)

[基础几何模板 82](#_Toc304454033)

[凸包 86](#_Toc304454034)

[多边形重心 86](#_Toc304454035)

[多边形面积 87](#_Toc304454036)

[半平面交 87](#_Toc304454037)

[三角形和圆的面积交 88](#_Toc304454038)

[最近点对 二维 89](#_Toc304454039)

[最近点对 三维 90](#_Toc304454040)

[匹配 91](#_Toc304454041)

[一些结论 91](#_Toc304454042)

[二分图最大匹配 邻接阵 91](#_Toc304454043)

[二分图最大匹配 邻接表 92](#_Toc304454044)

[二分图最大匹配HK 92](#_Toc304454045)

[二分图最大权值匹配 93](#_Toc304454046)

[一般图匹配 94](#_Toc304454047)

[随机算法 96](#_Toc304454048)

[费马点 模拟退火 96](#_Toc304454049)

[费马点 双重三分 98](#_Toc304454050)

[随机增量算法 99](#_Toc304454051)

[最小包含球 100](#_Toc304454052)

[组合数学 101](#_Toc304454053)

[N\*M数码可达性判断 101](#_Toc304454054)

[Polya计数 102](#_Toc304454055)

[公式 103](#_Toc304454056)

[容斥原理 103](#_Toc304454057)

[广义容斥原理 103](#_Toc304454058)

[应用 104](#_Toc304454059)

[一维最多不交叉线段数,基于递归的dp状态 104](#_Toc304454060)

[不同子串个数 SA nlog(n) 105](#_Toc304454061)

[不同子串个数 哈希n^2logn 105](#_Toc304454062)

[部分点最小生成树 106](#_Toc304454063)

[插头dp 107](#_Toc304454064)

[插头dp 2进制 108](#_Toc304454065)

[插头dp 3进制 109](#_Toc304454066)

[插头dp 4进制 112](#_Toc304454067)

[单串随机生成期望 115](#_Toc304454068)

[多项式去括号 115](#_Toc304454069)

[格点计数 116](#_Toc304454070)

[矩形面积并 n^3 117](#_Toc304454071)

[矩形面积并 nlogn 118](#_Toc304454072)

[最佳整数问题,类似欧几里得的翻转 119](#_Toc304454073)

[全排列哈希 120](#_Toc304454074)

[树链剖分 120](#_Toc304454075)

[树上基于点的分治 123](#_Toc304454076)

[树上最长路径,条数 126](#_Toc304454077)

[树套树 128](#_Toc304454078)

[树同构判定 130](#_Toc304454079)

[图同构判定 131](#_Toc304454080)

[无向图着色问题 132](#_Toc304454081)

[数独, 各种大小 133](#_Toc304454082)

[最优比率生成树 136](#_Toc304454083)

[树的删边游戏 139](#_Toc304454084)

[四边形不等式dp 140](#_Toc304454085)

[第k个square-free number 140](#_Toc304454086)

[高斯解模方程组 141](#_Toc304454087)

[8数码问题 143](#_Toc304454088)

[原根,计数 146](#_Toc304454089)

[Java常用 147](#_Toc304454090)

[IO 148](#_Toc304454091)

[常用算法 148](#_Toc304454092)

[排序 149](#_Toc304454093)

[BigInteger开根号 149](#_Toc304454094)

[java计算时间差 149](#_Toc304454095)

[HashMap TreeMap 149](#_Toc304454096)

[ext\_gcd求逆元 151](#_Toc304454097)

[输出格式 151](#_Toc304454098)

[字符串流 152](#_Toc304454099)

[树状数组 152](#_Toc304454100)

[Solutions 152](#_Toc304454101)

数论

## 基础数论模板

### gcd lcm

typedef long long LL;

int gcd(int a,int b){ return b ? gcd(b,a%b) : a; }

int lcm(int a,int b){ return a/gcd(a,b)\*b; }

### 扩展欧几里得

LL ext\_gcd(LL a,LL b,LL &x,LL &y){

if(b==0){ x=1,y=0; return a; }

LL d=ext\_gcd(b,a%b,x,y) , temp=x;

x=y,y=temp-a/b\*y; return d;

}

### 逆元

int get\_ni(int a,int p){ //返回x,ax=1 (mod p) ap互质

int x,y;

ext\_gcd(a,p,x,y);

return x;

}

### a^b %mod

int power(int a,int b,int mod){ //a^b % mod

int ans=1;

while(b){

if(b&1) ans=ans\*a%mod;

a=a\*a%mod; b>>=1;

}

return ans;

}

### 欧拉函数

int phi(int n){ //复杂度O(sqrt(n)) 已测

int ret=1,i;

for (i=2;i\*i<=n;i++){

if (n%i==0){

n/=i,ret\*=(i-1);

while (n%i==0) n/=i,ret\*=i;

}

}

if (n>1) ret\*=n-1; return ret;

}

### 解模方程 ax=b (mod n)

//解模方程ax=b(mod n), 返回解的个数,存于solve

//ax+ny=b有解(x0,y0),那么(x0+n/d,y0-a/d)也是解

int modular(LL a,LL b,LL n,LL \*solve){

assert(b!=0);

LL x0,y, d=ext\_gcd(a,n,x0,y);

if( b%d!=0 ) return 0; //无解

x0=x0\*(b/d); x0=(x0%(n/d)+(n/d))%(n/d); //x0为最小的非负整数使得ax=b(mod n)

for(int i=0;i<d;i++) solve[i]=(x0+i\*(n/d))%n;

return (int)d; //有d个解

}

### 解模线性方程组

//ret=b[i] (mod w[i]) i=0~n-1

//返回解,如果是-1表示无解

LL modular\_linear\_system(int \*b,int \*w,int n){ // 要求b[i]<w[i]

LL x0,y,d, ret=b[0], g=1;

for(LL i=1;i<n;i++){

g=g/gcd(g,w[i-1]) \* w[i-1];

LL B=(b[i]-ret%w[i] + w[i])%w[i]; if( B==0 ) continue;

d=ext\_gcd(g,w[i],x0,y); if( B%d ) return -1; //无解

x0=x0\*(B/d); x0=(x0%(w[i]/d) + w[i]/d)%(w[i]/d); //最小的x0,使得g\*x0=B(mod w[i])

ret+=x0 \* g;

}

return ret;

}

### 分解整数n

//分解一个数n={(p1,num1),(p2,num2)...},复杂度O(sqrt(n))

typedef pair<int,int> PII;

vector<PII> express(int n){ //复杂度O(sqrt(n))

vector<PII> ret;

for(int i=2;i\*i<=n;i++){

if( n%i==0 ){

int num=0;

while( n%i==0 ) num++,n/=i;

ret.push\_back(PII(i,num) );

}

}

if( n!=1 ){

ret.push\_back(PII(n,1) );

}

return ret;

}

### 求约数

vector<int> divisor(int n){ //包括n自己

vector<int> ret;

for(int i=1;i\*i<=n;i++){

if( n%i==0 ){

ret.push\_back(i);

if( i\*i != n) ret.push\_back(n/i);

}

}

return ret;

}

### 求原根

只有奇素数才有原根,共phi(p-1)个...

int primitive\_root(int p){

vector<int> D=divisor(p-1);

for(int rt=2;;rt++){

bool flag=true;

for(int i=0;i<D.size() && flag;i++){

if( power(rt,D[i],p)==1 ) flag=false;

}

if( flag ) return rt;

}

}

## 大数质因数分解

//hdu 3864 大数质因数分解

using namespace std;

typedef unsigned long long LL;

const int MAXT = 100; //max Test

const int MAXN = 30;

LL len, dig, limit;

LL random() {

return rand()\*(LL)rand()\*rand()\*rand();

}

LL mul(LL a,LL b,LL c) { // a\*b %c 全是非负的

LL ret=0,tmp=a%c;

for (;b;b>>=1){

if(b&1) if( (ret+=tmp) >=c) ret-=c;

if( (tmp<<=1) >=c) tmp-=c;

}

return ret;

}

LL power(LL a,LL b,LL c) { // a^b %c 全是非负的

LL res=1;

for (;b;b >>= 1) {

if (b & 1) res=mul(res,a,c);

a=mul(a,a,c);

}

return res;

}

bool Miller\_Rabin(LL n) { //n是不是素数

if (n<2 ) return false;

if (n==2) return true;

if (n%2==0) return false;

LL i, j, k, m, a;

for (m = n-1,k = 0;!(m&1);m >>= 1,k++);

for (i = 0;i < MAXT;i++) { //保守出错率为 1/(2^MAXT)

a = power(random()%(n-1) + 1,m,n);

if (a == 1) continue;

for(j = 0; j < k; j ++) {

if (a == n - 1) break;

a = mul(a,a,n);

}

if (j == k) return false;

}

return true;

}

LL gcd(LL a, LL b) {

return b ? gcd(b,a%b) : a;

}

LL f(LL x, LL n) { //x^2+1 %n

return ( mul(x,x,n) +1) %n;

}

LL Pollard\_Rho(LL n) { //寻找n的一个因子

if (n <= 2) return 0;

if (!(n & 1)) return 2;

for (LL i = 1;i < MAXT;i++) {

LL x = random() % n;

LL xx = f(x,n);

LL p = gcd((xx + n - x) % n, n);

while (p == 1) {

x = f(x, n);

xx = f(f(xx,n),n);

p = gcd((xx + n - x) % n, n) % n;

}

if (p) return p;

}

return 0;

}

LL Prime(LL a) { //寻找a的一个素因子

if (Miller\_Rabin(a)) return a;

return Prime( Pollard\_Rho(a) );

}

LL fac[MAXN],m;

void fenjie(LL a){//所有素约数保存于fac[]中,m为其个数base0

m = 0;

while (a > 1) {

if (Miller\_Rabin(a)) break;

LL t = Prime(a);

fac[m++] = t;

a /= t;

}

if (a > 0) fac[m++] = a;

}//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*大数分解素因子\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int main() {

LL a,t;

srand(time(NULL));

limit = (1LL) << 63;

while (cin>>a) { //a可以达到10^18大小...

fenjie(a);

sort(fac,fac+m);

if( m==3 && fac[0]==fac[1] && fac[0]==fac[2] ){

cout<<fac[0]<<' '<<fac[0]\*fac[0]<<' '<<fac[0]\*fac[0]\*fac[0]<<endl;

continue;

}

if (m != 2) printf("is not a D\_num\n");

else {

if( fac[0]==fac[1] ) puts("is not a D\_num");

else

cout<<fac[0]<<' '<<fac[1]<<' '<<fac[0]\*fac[1]<<'\n';

}

}

}

## 求sigma (n /i)

//n/1+n/2+...+n/n 收敛很快,复杂度能保证

inline int big(int n,int val){ //最大的ret使得n/ret = val

return n/val;

}

int get(int n){ //计算1,2,3，...,n的因子数目之和,就是算n/1+n/2+...+n/n

int ret=0;

for(int i=1;i<=n;i++){

int val=n/i, R=big(n,val);

ret+=val\*(R-i+1);

i=R;

}

return ret;

}

## AC的离散对数

计算最小的x, a^x=n(%mod)

//Author : AekdyCoin

typedef long long LL;

int gcd(int a,int b){return b?gcd(b,a%b):a;}

int ext\_gcd(int a,int b,int& x,int& y){

int t,ret;

if (!b){x=1,y=0;return a;}

ret=ext\_gcd(b,a%b,x,y);

t=x,x=y,y=t-a/b\*y;

return ret;

}

int Inval(int a,int b,int n){

int x,y,e;

ext\_gcd(a,n,x,y);

e=(LL)x\*b%n;

return e<0?e+n:e;

}

int pow\_mod(LL a,int b,int c){

LL ret=1%c;while(b){if(b&1)ret=ret\*a%c;a=a\*a%c;b>>=1;}return ret;}

int BabyStep(int A,int B,int C)//**A^x = B(mod C)**

{

map<int,int> Hash;

LL buf=1%C,D=buf,K;

int i,d=0,tmp;

for(i=0;i<=100;buf=buf\*A%C,++i)if(buf==B)return i;

while((tmp=gcd(A,C))!=1)

{

if(B%tmp)return -1;

++d;

C/=tmp;

B/=tmp;

D=D\*A/tmp%C;

}

Hash.clear();

int M=sqrt(C+0.0);

for(buf=1%C,i=0;i<=M;buf=buf\*A%C,++i)if(Hash.find(buf)==Hash.end())Hash[buf]=i;

for(i=0,K=pow\_mod(A,M,C);i<=M;D=D\*K%C,++i)

{

tmp=Inval(D,B,C);

if(tmp>0&&Hash.find(tmp)!=Hash.end())return i\*M+Hash[tmp]+d;

}

return -1;

}

int main(){

int A,B,C;

while(scanf("%d%d%d",&A,&C,&B)!=EOF)

{

if(B>=C){ puts("Orz,I can’t find D!"); continue; }

int tmp=BabyStep(A,B,C);

if(tmp<0)puts("Orz,I can’t find D!");

else printf("%d\n",tmp);

}

return 0;

}

## a^b的前几位

//10^3.542 = 10^3 \* 10^0.542 = 1000 \* 10^0.542

//可见真正的数字还是靠幂的小数部分的

int E(int a,int b,int l){

double ret=b\*1.0\*log10((double)a);

ret-=floor(ret);

ret=pow(10.0,ret);

while(--l)

ret\*=10.0;

return (int)floor(ret);

}

## 大组合数取模 模质数

//大组合数取模 质数 O(p)

/\*

Lucas定理:

求C(n,m)%p (p是素数)

n=a0+a1\*p+...+ak\*p^k

m=b0+b1\*p+...+bk\*p^k

则C(n,m)=C(a0,b0)\*C(a1,b1)\*...\*C(ak,bk)%p

\*/

const int mod=10567201;

int mul[100005]={1};//mul[i]=i!

void e\_gcd(int a,int b,int &x,int &y) {//扩展欧几里得定理：解ax+by==1。

if( !b ){

x=1; y=0;

}

else{

e\_gcd(b,a%b,x,y);

int temp=x;

x=y; y=temp-a/b\*y;

}

}

//算逆元,注意\*\*\*对一个负数取逆元是没有意义的

int get\_ni(int a,int p){ //返回x,ax=1 (mod p) p是素数

int x,y; e\_gcd(a,p,x,y);

return x;

}

inline int C(int n,int m,int mod){

if( m>n ) return 0;

return ((long long)mul[n]\*get\_ni(mul[m],mod)%mod \* get\_ni(mul[n-m],mod)%mod + mod)%mod;

}

int main(){

int n,m,p,cas; scanf("%d",&cas);

while(cas--){

scanf("%d%d%d",&n,&m,&p); n+=m;

for(int i=1;i<=p;i++) mul[i]=(long long)mul[i-1]\*i%p;

long long ret=1;

while(n) {

ret=ret\*C(n%p,m%p,p)%p;

n/=p; m/=p;

}

printf("%d\n",ret);

}

}

## 大组合数取模,模P^t

//大组合数取模P,P是一个素数p的t次幂,复杂度O(P) 已测试hdu3439

/\*

Lucas定理:

求C(n,m)%p (p是素数)

n=a0+a1\*p+...+ak\*p^k

m=b0+b1\*p+...+bk\*p^k

则C(n,m)=C(a0,b0)\*C(a1,b1)\*...\*C(ak,bk)%p

注意\*\*\*

t!的逆元 = t!%p的逆元

\*/

int mul[100005]={1};//mul[i]=i!

void e\_gcd(int a,int b,int &x,int &y) { //扩展欧几里得定理：解ax+by=gcd(a,b)。

if( !b ){

x=1; y=0;

}else{

e\_gcd(b,a%b,x,y);

int temp=x;

x=y;

y=temp-a/b\*y;

}

}

//算逆元

int get\_ni(int a,int p){ //返回x,ax=1 (mod p) ap互质

int x,y;

e\_gcd(a,p,x,y);

return x;

}

int Power(int a,int b,int p){

int ret=1;

while( b ){

if( b&1 ) ret=(long long)ret\*a%p;

a=(long long)a\*a%p;

b>>=1;

}

return ret;

}

//计算n!中有几个p,放在num里,其它相乘结果放在res

void Calculate(int n,int p,int mod,int &res,int &num){

res=1; num=0;

while( n!=0 ){

res=(long long)res\*Power(mul[mod],n/mod,mod)%mod;

res=(long long)res\*mul[n%mod]%mod;//注意这里是mod 不是 p

num+=n/p;

n/=p;

}

}

//n!/( m!\*(n-m)! )

int Chose(int n,int m,int p,int t){ //N中选M,结果模P^t,P是素数

int MM=1;

for(int i=1;i<=t;i++) MM\*=p;

if( m>n ) return 0;

for(int i=1;i<=min(MM,n);i++){ //mul[i]是i!中不包含p的乘积

if(i%p==0) mul[i]=mul[i-1]; //i%p==0是另外考虑的

else mul[i]=(long long)mul[i-1]\*i%MM;//mul[0]=1;

}

int fenzi=0,fenmu=0,num,res,ans=1;

Calculate(n,p,MM,res,num);

ans=(long long)ans\*res%MM; fenzi+=num;

Calculate(m,p,MM,res,num);

ans=(long long)ans\*get\_ni(res,MM)%MM; fenmu+=num;

m=n-m;

Calculate(m,p,MM,res,num);

ans=(long long)ans\*get\_ni(res,MM)%MM; fenmu+=num;

if(fenzi-fenmu >= t ) return 0;

ans=(long long)ans\*Power(p,fenzi-fenmu,MM)%MM;

return (ans+MM)%MM;

}

int main(){

int n,m,p,t,cas;

scanf("%d",&cas);

while(cas--)

{

scanf("%d%d%d",&n,&m,&p);

printf("%d\n",Chose(n+m,m,p,1));

}

}

## 高斯解n元模方程组

#define maxn 55

typedef long long LL;

const LL mod=200000000000000003ll;

inline void fix(LL &a){

if( a<=-mod ) a+=mod;

if( a>=mod ) a-=mod;

}

LL mul(LL a,LL b,LL mod){ //b为正数,否则死循环

LL ret=0;

bool flag=false; if( b<0 ){ b=-b; flag=true; }

while( b ){

if( b&1 ) ret=(ret+a),fix(ret);

a<<=1; fix(a);

b>>=1;

}

if( flag ) return -ret;

return ret;

}

void e\_gcd(LL a,LL b,LL &x,LL &y) {

if( !b ){

x=1; y=0;

}else {

e\_gcd(b,a%b,x,y);

LL temp=x;

x=y;

y=temp-mul(a/b,y,mod);

}

}

LL get\_ni(LL a,LL p) {//返回x,ax=1 (mod p) p是素数

LL x,y;

e\_gcd(a,p,x,y);

return x;

}

/\*

高斯解模方程组,当无解或者无限组解时会assertRE,有解会放在x里面

c[1][1]\*x[1]+c[1][2]\*x[2]+...+c[1][n]\*x[n]=b[1]

c[2][1]\*x[1]+c[2][2]\*x[2]+...+c[2][n]\*x[n]=b[2]

....

c[n][1]\*x[1]+c[n][2]\*x[2]+...+c[n][n]\*x[n]=b[n]

\*/

LL c[maxn][maxn],x[maxn],b[maxn];

void gauss(int N){ //C\*X=B

for(int i=1,k;i<=N;i++){

if( c[i][i]==0 ){

LL ma=0;

for(int j=i;j<=N && ma==0;j++){

if( c[j][i]!=ma ){

ma=c[j][i];

k=j;

}

}

for(int j=i;j<=N;j++) swap(c[i][j],c[k][j]);

swap(b[i],b[k]);

}

if( c[i][i]<0 ){

for(int j=i;j<=N;j++) c[i][j]=-c[i][j];

b[i]=-b[i];

}

assert(c[i][i]!=0);//\*\*\*\*\*等于0的话无解或者有无限组解

for(int k=i+1;k<=N;k++){

if( c[k][i]==0 ) continue;

LL m=get\_ni(c[i][i],mod);

m=mul(m,c[k][i],mod);

for(int j=i;j<=N;j++){

c[k][j]-=mul(m,c[i][j],mod);

fix(c[k][j]);

}

assert(c[k][i]==0);

b[k]-=mul(m,b[i],mod);

fix(b[k]);

}

}

for(int i=N;i>=1;i--){

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

b[i]-=mul(c[i][j],x[j],mod);

fix(b[i]);

}

x[i]=mul(b[i],get\_ni(c[i][i],mod),mod);

}

}//--------------------模板结束--------------------

int main(){

int n;

while(cin>>n){

for(int i=1;i<=n;i++) for(int j=1;j<=n;j++) cin>>c[i][j];

for(int i=1;i<=n;i++) cin>>b[i];

gauss(n);

for(int i=1;i<=n;i++) printf("%I64d ",x[i]); puts("");

}

}

/\*

input:

3

1 2 3

4 5 6

7 8 10

33 75 124

output:

2 5 7

\*/

## 计算1~y中有几个gcd(a,i)=1

int \_dfs(vector<int> &p,int d,int x,int y){

if( d==p.size() ) return y/x;

return \_dfs(p,d+1,x,y) + \_dfs(p,d+1,x \* (-p[d]),y);

}

//计算1~y中有几个gcd(a,i)=1,总量减去能整出因数的,容斥原理,复杂度sqrt(a)+2^(p.size())

int cal(int a,int y){

vector<int> p;

for(int i=2;i\*i<=a;i++){

if( a%i==0 ){

p.push\_back(i);

while( a%i==0 ) a/=i;

}

}

if( a!=1 ) p.push\_back(a);

return \_dfs(p,0,1,y);

}

## 中国剩余定理

对于每个数n，它可以分解为n=n1\*n2\*n3\*n4\*...\*nk，且他们之间互质则：

1，小于n的整数p都可以唯一地表示成一个k元组(a1,a2,a3,...,ak)其中ai=p%ni;

2，每个k元组都对应一个小于n的整数

下面给出2的证明：

对于k元组，我们可以给出k个基，第i个基是(0,0,...,1,0,0,...0)其中1在第i个地方

显然地，第i个基对应的整数就是( ci=(n/ni)\*(模ni下n/ni的逆元) ).

所以k元组(a1,a2,...,ak)对应的整数就是(a1\*c1+a2\*c2+a3\*c3+...ak\*ck)mod n

## N!中0的个数

const int Factor=1005;

int zeros[Factor],a[2577]={1};

void process(){

double bitNum=1.0;

for(int n=2,beg=0,e=0;n<=Factor;++n){

e=bitNum+=log10(n\*1.0);

while(a[beg]==0)

beg++;

for(int j=beg,and=0;j<e;j++,and/=10)

a[j]=(and+=n\*a[j])%10;

zeros[n]=count(a+beg,a+e,0)+beg;

}

}

int main(){

process();

int n;

while(scanf("%d",&n)!=EOF)

printf("%d\n",zeros[n]);

}

数学

## 多项式展开

//多项式展开,用于母函数展开\*\*\*\*

typedef long long type;

struct Polynomial{

map<int,type> mp; //x的幂, 系数

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Polynomial(){ mp.clear(); }

void init(vector< pair<int,type> > &v){

mp.clear();

for(int i=0;i<v.size();i++){

mp[ v[i].first ] += v[i].second;

}

}

void init(int \*mi,type \*c,int ind){

mp.clear();

for(int i=0;i<ind;i++){

mp[ mi[i] ] += c[i];

}

}

void init(type val){

mp.clear();

mp[0]=val;

}

Polynomial operator\*(Polynomial &p){

map<int,type>::iterator it1,it2;

Polynomial ret; ret.mp.clear();

for(it1=mp.begin();it1!=mp.end();++it1){

for(it2=p.mp.begin();it2!=p.mp.end();++it2){

int n=it1->first + it2->first;

ret.mp[ n ]+=it1->second \* it2->second;

}

}

return ret;

}

Polynomial operator+(Polynomial &p){

map<int,type>::iterator it;

Polynomial ret=\*this;

for(it=p.mp.begin();it!=p.mp.end();++it){

ret.mp[ it->first ]+=it->second;

}

return ret;

}

void print(){

map<int,type>::iterator it1=mp.end();

bool first=true;

do{

--it1;

if( !first ) putchar('+');

if( it1->second!=1 ) cout<<it1->second;

if( it1->first!=1 )printf("x^%d",it1->first);

else putchar('x');

first=false;

}while( it1!=mp.begin() );

cout<<endl;

}

}st[100];

## N进制位操作

//位运算快一点的

//每个实例都是相应进制的函数集合, 用于连通性状态压缩dp

template<int N> //N只能为2,4,8, 2 4 已测!!

struct Number{

int w[30]; //X=b0\*w[0]+b1\*w[1]+...+b[n]\*w[n]

int L,mask; //二进制位数,

Number(){ //初始化

if( N==2 ) L=1;

else if( N==4 ) L=2;

else L=3;

mask=N-1; w[0]=1;

for(int i=1;i<30;i++) w[i]=w[i-1]<<L;

}

int arr2int(int \*d,int t){ //将一个数组计算出一个N进制数,ind base0

int s=0;

for(int i=t-1;i>=0;i--) s=(s<<L)|d[i];

return s;

}

void int2arr(int s,int \*d,int &t){ //将一个N进制数解析成一个数组

t=0;

while( s ){ d[t++]=s&mask; s>>=L; }

if( t==0 ) d[t++]=0;

}

int test(int s,int wei){ //取出N进制数s的第wei位 wei:base0

return (s>>(wei\*L))&mask;

}

void set(int &s,int wei,int b){ //将N进制数s的第wei位设置为b wei:base0

s+=( (b-test(s,wei))<<(wei\*L) );

}

};//\*\*\*\*\*\*\*\*\*\*\*\*模板结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Number<4> tool4;

//每个实例都是相应进制的函数集合, 用于连通性状态压缩dp

template<int N> //可以任意>=2的进制

struct Number{

int w[30]; //X=b0\*w[0]+b1\*w[1]+...+b[n]\*w[n]

Number(){ //对于每个实例必须先初始化

w[0]=1; for(int i=1;i<30;i++) w[i]=w[i-1]\*N;

}

int arr2int(int \*d,int t){ //将一个数组计算出一个N进制数,ind base0

int s=0;

for(int i=0;i<t;i++) s+=d[i]\*w[i];

return s;

}

void int2arr(int s,int \*d,int &t){ //将一个N进制数解析成一个数组

t=0;

while( s ){ d[t++]=s%N; s/=N; }

if( t==0 ) d[t++]=0;

}

int test(int s,int wei){ //取出N进制数val的第wei位

return s%w[wei+1]/w[wei];

}

void set(int &s,int wei,int b){ //将N进制数val的第wei位设置为b

s+=(b-test(s,wei))\*w[wei];

}

};//\*\*\*\*\*\*\*\*\*\*\*\*模板结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Number<4> tool4;

## 分数类

//\*\*\*\*\*\*\*\*\*\*\*分数类\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

typedef long long int64;

template<typename T>

T gcd(T a,T b) {

return b == 0 ? a : gcd(b,a % b);

}

struct Fraction {

int64 n,d; // n/d

Fraction(int64 \_n = 0,int64 \_d = 1) {

assert(\_n || \_d); // 0/0

n=\_n,d=\_d;

int64 g = gcd(n,d);

n /= g,d /= g;

if (d < 0) n = -n,d = -d;

}

void print(){ cout<<n<<'/'<<d<<'\n'; }

};

Fraction operator +(const Fraction& L,const Fraction& R) {

return Fraction(L.n \* R.d + L.d \* R.n, L.d \* R.d);

}

Fraction operator -(const Fraction& L,const Fraction& R) {

return Fraction(L.n \* R.d - L.d \* R.n, L.d \* R.d);

}

Fraction operator \*(const Fraction& L,const Fraction& R) {

return Fraction(L.n \* R.n, L.d \* R.d);

}

Fraction operator /(const Fraction& L,const Fraction& R) {

return Fraction(L.n \* R.d, L.d \* R.n);

}

bool operator <(const Fraction& L,const Fraction& R) {

return L.n \* R.d < R.n \* L.d;//要比大小用这个

}

## 矩阵

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*矩阵模板\*\*\*\*\*\*\*\*\*\*\*

#define maxn 100

typedef long long type;

type tmp[maxn][maxn];

struct Matrix{

int R,C,mod; //0~R-1, 0~C-1

type m[maxn][maxn];

void init(int r,int c,int \_mod){

R=r,C=c; mod=\_mod;

for(int i=0;i<r;i++) fill(m[i],m[i]+c,0);

}

void set(int i,int j,int val){ m[i][j]=val; }

friend void mul(const Matrix &s1,const Matrix &s2,Matrix &des){ //des=s1 \* s2

assert( s1.C == s2.R ); //确保矩阵相容

for(int i=0;i<s1.R;i++){

for(int j=0;j<s2.C;j++){

type p=0;

for(int k=0;k<s1.C;k++){

p=(p+s1.m[i][k]\*s2.m[k][j])%s1.mod;

}

tmp[i][j]=p;

}

}

des.R=s1.R, des.C=s2.C;

for(int i=0;i<s1.R;i++) memcpy(des.m[i],tmp[i],sizeof(type)\*s2.C);

}

}mat1,mat2; //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*模板结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## 计算 sigma(i^k)

x^k=(x-1 + 1)^k,然后二项式展开,得出递推式

typedef \_\_int64 int64;

\*\*这里是矩阵模板(被省略了)\*\*

int C[100][100];

int calc(int n,int k,int mod){ // 计算 1^k + 2^k + 3^k + 4^k + ...

if( n==0 ) return 0;

int i,j;

for(i=0;i<100;i++) C[i][0]=1;

for(i=1;i<100;i++) for(j=1;j<=i;j++) C[i][j]=(C[i-1][j]+C[i-1][j-1])%mod;

mat1.init(1,k+2,mod);

mat2.init(k+2,k+2,mod);

for(j=0;j<k+2;j++) mat1.set(0,j,1);

for(j=0;j<k+1;j++){

for(int i=0;i<=j;i++) mat2.set(i,j,C[j][i]);

}

for(i=0;i<=k;i++) mat2.m[i][k+1]=mat2.m[i][k];

mat2.m[k+1][k+1]=1;

n--;

while( n ){

if( n&1 ) mul(mat1,mat2,mat1);

mul(mat2,mat2,mat2);

n/=2;

}

return mat1.m[0][k+1];

}//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## 浮点型高斯消元

//浮点型高斯消元,返回是否有确定解

double a[MAXN][MAXN],x[MAXN],b[MAXN];

bool gauss() {

for(int i=0;i<n;i++){ //下半角全部清0

double m=abs(a[i][i]); //注意m的初值

index=i;

for(int j=i+1;j<n;j++){ //找到绝对值最大的一行

if(abs(a[j][i]) >m){

m=abs(a[j][i]);

index=j;

}

}

if(m<eps)return false; //下面的全是0时，返回无解

if(index!=i){

swap(b[i],b[index]); //整个增广矩阵的两行交换下

for(int k=0;k<n;k++)

swap(a[i][k],a[index][k]);

}

for(int k=i+1;k<n;k++) { //下面是第i行去减第k行，第k行的0~i-1已经全是0的了

double d=a[k][i]/a[i][i]; //算出比例

b[k]-=b[i]\*d;

for(int p=i;p<n;p++)

a[k][p]-=a[i][p]\*d;

}

}

for(int i=n-1;i>=0;i--){

for(int j=i+1;j<n;j++)

b[i]-=x[j]\*a[i][j];

x[i]=b[i]/a[i][i];

}

return true;

}

## 模2域高斯消元

//模2情况下, c[][] \* x[] = b[]

//高斯法求秩,判断方程组是否有解,-1表示无解

int gauss(int n,int m,int c[][maxm],int \*b){ //会改变c和b数组

int zhi=0,k,i,j;

for(i=1,j=1;i<=n && j<=m;){

int cur=0;

for(k=i;cur==0 && k<=n;k++) {

if(c[k][j]) { cur=1; break; }

}

if( cur==0 ){ j++; continue; } //

for(int jj=j;jj<=m;jj++) swap(c[i][jj],c[k][jj]);

swap(b[i],b[k]);

for(k=i+1;k<=n;k++){

if(c[k][j]){

for(int jj=j;jj<=m;jj++)

c[k][jj]^=c[i][jj];

b[k]^=b[i];

}

}

zhi++,i++,j++;

}

while(i<=n) if(b[i++]) return -1;

return zhi;

}

## 复合Simpson积分公式

/\*

复合Simpson积分公式

I(f) = ∫(a,b)f(x)dx = ∑k[0,n-1]∫(xk,x(k+1))f(x)dx

= (h/6)\*{f(a) + 4\*∑k[0,n-1]f(x((2\*k+1)/2))dx + 2\*∑k[1,n-1]f(xk)dx + f(b)}

xk = a+k\*h(k = 0,1,...,n)

h = (b-a) / n;

\*/

const long double H = 5e-5;

long double r1, r2;

long double f(long double x){//被积函数

return x\*x;

}

long double simpson(long double a,long double b){

long double sum = 0;

sum += f(a) + f(b);

int n = int((b - a) / H);

long double h = (b - a) / n / 2;

for (int i = 1;i < 2 \* n;i++){

if (i & 1)

sum += 4 \* f(a+i\*h);

else

sum += 2 \* f(a+i\*h);

}

return sum \* H / 6;

}

int main() {

int re;

scanf("%d", &re);

for (int ri = 1; ri <= re; ++ri) {

scanf("%Lf%Lf", &r1, &r2);

if (r1 > r2)

swap(r1, r2);

long double ans = simpson(0,r1);

printf("%.6Lf\n",ans);

}

return 0;

}

## 计算行列式|A|模mod

//http://www.spoj.pl/problems/DETER3/

//计算行列式|A|%mod, 复杂度n^3logC 算法: 一行将另一行消去的时候,采用欧几里得算法,辗转相消

//初始化 A数组, N, mod

typedef long long int64;

#define maxn 202

int A[maxn][maxn],N,mod; //N\*N的方阵, base1

void \_swap(int r1,int r2){

for(int j=1;j<=N;j++) swap(A[r1][j],A[r2][j]);

}

void \_dec(int r1,int r2,int mul){ //r1行减去 mul\*r2行

for(int j=1;j<=N;j++) A[r1][j]=(A[r1][j]-(int64)A[r2][j]\*mul)%mod;

}

int calc\_delta(){ //结果模mod

bool neg=false;

for(int k=1;k<=N;k++){

for(int i=k+1;i<=N;i++){

while( true ){//循环logC次

if( A[k][k]==0 || A[i][k]==0 ) break;

if( abs(A[k][k]) >= abs(A[i][k]) ){

\_dec(k,i,A[k][k]/A[i][k]);

}else{

\_dec(i,k,A[i][k]/A[k][k]);

}

}

if( A[i][k]!=0 ) \_swap(i,k), neg=!neg; //两行互换,需要修改符号的

}

if( A[k][k]==0 ) return 0;

}

int64 ret= neg ? -1 : 1;

for(int k=1;k<=N;k++) ret=ret\*A[k][k]%mod;

return (int)ret;

}//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int main(){

while(scanf("%d%d",&N,&mod)!=EOF){

for(int i=1;i<=N;i++) for(int j=1;j<=N;j++) scanf("%d",A[i]+j);

printf("%d\n",(calc\_delta()+mod)%mod);

}

}

## 解Pell方程 x^2-N\*y^2=1

#define maxn 100000

int p[maxn],q[maxn];

int g[maxn],h[maxn];

int a[maxn];

void Pell(int N,int &x,int &y){ //N为非完全平方数

p[0]=0,p[1]=1;

q[0]=1,q[1]=0;

a[2]=(int)sqrt(N\*1.0);

g[1]=0;

h[1]=1;

for(int i=2;i<100000;i++){

g[i]=-g[i-1]+a[i]\*h[i-1];

h[i]=(N-g[i]\*g[i])/h[i-1];

a[i+1]=(g[i]+a[2])/h[i];

p[i]=a[i]\*p[i-1]+p[i-2];

q[i]=a[i]\*q[i-1]+q[i-2];

if(p[i]\*p[i]-N\*q[i]\*q[i]==1){

x=p[i],y=q[i];

return;

}

}

}

int main(){

int N;

while(cin>>N){

int x=0,y=0;

Pell(N,x,y);

printf("%d\*%d-%d\*%d\*%d=1\n",x,x,N,y,y);

}

}

## 求catalan数第n项

/\*

求catalan数第n项，

1, C(n)=(2n)!/ (n! \* (n+1)! )

2, C(n)=(4\*n-2)/(n+1) \* C(n-1) 且 C(0)=1;

O(n)的算法

令外：

求前n项和模m

先将m分解质因数，然后将(4 \* i - 2) 和 (i + 1) 和 c[i - 1]都分成两份，一份是与m互质的，另外一份是与由m包含的质因数构成的。然后分别维护这两份。前者只要用“数”来维护就好，对于前者，由于和m互质了，除数可以求逆元，然后与(4\*i - 2)和c[i - 1]的第一份边乘边模，对于后者，可以用m的质因数表来维护。最后c[i]就是把质因数表中的数乘到第一份里，边乘边模m。注意可能超int的地方。

\*/

#define MAXN 200005 // 2\*n

typedef long long int64;

int isp[MAXN],num[MAXN],tmp[MAXN];

void init\_isp(){

memset(isp,0,sizeof(isp));

for(int i=2;i\*i<MAXN;i++){

if( isp[i] ) continue;

for(int j=i\*i;j<MAXN;j+=i)

isp[j]=i;

}

//printf("init\n");

}

void resolve(int \*num,int n){ //resolve n! to num array

for(int i=0;i<=n;i++) num[i]=1;

for(int i=n;i>=4;i--){

if( isp[i]==0 ) continue;

num[ isp[i] ]+=num[i];

num[ i/isp[i] ]+=num[i];

num[i]=0;

}

}

int cal\_catalan(int n,int mod){//计算catalan数第n项，结果模mod

static bool init=false;

if( !init ) init\_isp(), init=true;

resolve(num,2\*n);

resolve(tmp,n);

for(int i=2;i<=n;i++) num[i]-=tmp[i];

resolve(tmp,n+1);

for(int i=2;i<=n+1;i++) num[i]-=tmp[i];

int ret=1;

for(int i=2;i<=2\*n;i++){

for(int j=1;j<=num[i];j++)

ret=(int64)ret \* i %mod;

}

return ret;

}

## 线段下方的整点数目

//计算一条线段下方的整点数目,包含上边界,不报含y=0的下边界(x的范围也是<n,而没有等于n)

//y=(a+d\*x)/m x属于[0,n)

int64 count(int64 a,int64 d,int64 m,int64 n){

int64 ret=0;

if( a/m ){

ret+=a/m\*n; a%=m;

}

if( d/m ){

ret+=d/m \*n\*(n-1)/2;

d%=m;

}

if( d==0 ) return ret;

int l=(a+d\*n)/m;

return ret+count( (a+d\*n)%m,m,d,l );

}

## BigInteger

//+ - \* / % 五种运算均通过了50万组随机数据粗暴测试

//-------------------------------------------------------------

#define BASE 10000//进制

#define SIZE 201 //默认长度 最大表示长度为SIZE\*4

struct BigInteger{

int len,data[SIZE],neg; //长度,数据,是否为负

BigInteger() { len=neg=0; }

BigInteger(int v) {len=neg=0; if(v<0){neg=1;v=-v;} for(;v;v/=BASE)data[len++]=v%BASE; }

BigInteger(char \*s);

BigInteger(const BigInteger&v) { len=v.len; neg=v.neg; memcpy(data,v.data,len\*sizeof(\*data)); }

//以上构造函数

BigInteger& operator=(const BigInteger&v);

BigInteger& operator<<(const int cnt);

int&operator[](int index){return data[index];}

int operator[](int index)const{return data[index];}

friend bool operator==(const BigInteger&a,const BigInteger&b);

friend bool operator <(const BigInteger&a,const BigInteger&b);

friend BigInteger operator+(const BigInteger&a,const BigInteger&b);

friend BigInteger operator-(const BigInteger&a,const BigInteger&b);

friend BigInteger operator\*(const BigInteger&a,const BigInteger&b);

friend BigInteger operator/(const BigInteger&a,const BigInteger&b);

friend BigInteger operator%(const BigInteger&a,const BigInteger&b);

void print(char\* result); //将这个BigInteger放到result里面

void print(); //直接输出到屏幕上,并换行

};

BigInteger::BigInteger(char \*p){

len=neg=0;

if( p[0]=='-' ) { neg=1; p++; };

for(int j=strlen(p)-4;j>-4;j-=4){

int tmp=0;

for(int k=j;k<j+4;k++){

if( k>=0 ) tmp=10\*tmp+p[k]-'0';

}

data[len++]=tmp;

}

while( len && data[len-1]==0 )len--; //前导0

}

BigInteger& BigInteger::operator=(const BigInteger&v){

len=v.len;

neg=v.neg;

memcpy(data,v.data,len\*sizeof(\*data));

return \*this;

}

BigInteger& BigInteger::operator<<(const int cnt){

if(len==0||cnt<=0)return \*this;

memcpy(data+cnt,data,len\*sizeof(\*data));

memset(data,0,cnt\*sizeof(\*data));

len+=cnt;

return \*this;

}

int \_cmp(const BigInteger&a,const BigInteger&b){

int i;

if(a.neg==1&&b.neg==0)return -1;

if(a.neg==0&&b.neg==1)return 1;

if(a.neg==0){

if(a.len!=b.len)return a.len>b.len?1:-1;

for(i=a.len-1;i>=0&&a[i]==b[i];i--);

if(i<0)return 0;

return a[i]>b[i]?1:-1;

}else{

if(a.len!=b.len)return a.len>b.len?-1:1;

for(i=a.len-1;i>=0&&a[i]==b[i];i--);

if(i<0)return 0;

return a[i]>b[i]?-1:1;

}

}

bool operator==(const BigInteger&a,const BigInteger&b){return \_cmp(a,b)==0;}

bool operator< (const BigInteger&a,const BigInteger&b){return \_cmp(a,b)<0;}

BigInteger operator+(const BigInteger&a,const BigInteger&b){

BigInteger ret;

int i,c=0;

if(a.neg==b.neg){

for(i=0;i<a.len||i<b.len||c;i++){

if(i<a.len)c+=a[i];

if(i<b.len)c+=b[i];

ret[i]=c%BASE;

c/=BASE;

}ret.len=i;

ret.neg=a.neg;

}else{

BigInteger va=a,vb=b;

va.neg=0;

vb.neg=0;

ret=va-vb;

if(a.neg)ret.neg=!ret.neg;

}

return ret;

}

BigInteger operator-(const BigInteger&a,const BigInteger&b){

BigInteger ret;

int i,c=0;

if(a.neg==b.neg){

if( a.neg==1 ){

BigInteger ta=a, tb=b;

ta.neg=tb.neg=0;

return tb-ta;

}

if(a<b){

ret=b-a;

ret.neg=1;

}else{

for(i=0;i<a.len;i++){

ret[i]=a[i]-c;

if(i<b.len)ret[i]-=b[i];

if(ret[i]<0){c=1;ret[i]+=BASE;}

else c=0;

}while(i>0&&ret[i-1]==0)i--;

ret.len=i;

ret.neg=0;

}

}else{

BigInteger vb=b;

vb.neg=!b.neg;

ret=a+vb;

}

return ret;

}

BigInteger operator\*(const BigInteger&a,const BigInteger&b){

int i,j;

if(!b.len)return 0;

BigInteger ret;

for(i=0;i<a.len;i++){

int c=0;

for(j=0;j<b.len||c;j++){

if(j<b.len)c+=(int)a[i]\*(int)b[j];

if(i+j<ret.len){

c+=ret[i+j];

ret[i+j]=c%BASE;

}else ret[ret.len++]=c%BASE;

c/=BASE;

}

}ret.neg=a.neg^b.neg;

return ret;

}

BigInteger operator/(const BigInteger&a,const BigInteger&b){

BigInteger ret,c;

if( b.neg==1 ){

ret=a; c=b;

ret.neg=!ret.neg; c.neg=!c.neg;

return ret/c;

}

int i,lo,hi,mid;

for(i=a.len-1;i>=0;i--){

c=(c<<1)+a[i];

lo=0;

hi=BASE-1;

while(lo<hi){

mid=(lo+hi+1)/2;

if(\_cmp(b\*mid,c)<=0)lo=mid;

else hi=mid-1;

}ret[i]=lo;

c=c-b\*lo;

}ret.len=a.len;

while(ret.len&&ret[ret.len-1]==0)ret.len--;

if(ret.len)ret.neg=a.neg^b.neg;

else ret.neg=0;

return ret;

}

BigInteger operator%(const BigInteger&a,const BigInteger&b){

BigInteger c;

int i,lo,hi,mid;

for(i=a.len-1;i>=0;i--){

c=(c<<1)+a[i];

lo=0;

hi=BASE-1;

while(lo<hi){

mid=(lo+hi+1)/2;

if(\_cmp(b\*mid,c)<=0)lo=mid;

else hi=mid-1;

}c=c-b\*lo;

}c.neg=a.neg^b.neg;

return c;

}

void BigInteger::print(){

if(len==0){ puts("0"); return; }

if(neg==1) printf("-");

printf("%d",data[len-1]);

for(int i=len-2;i>=0;i--)

printf("%04d",data[i]);

puts("");

}

void BigInteger::print(char\* s){

if(len==0){s[0]='0';s[1]=0;return;}

if(neg==1){\*s='-';s++;}

sprintf(s,"%d",data[len-1]);

int i;

for(i=len-2;i>=0;i--)

sprintf(s+strlen(s),"%04d",data[i]);

}

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

int main(){

int a,b;

while(scanf("%d %d",&a,&b)!=EOF){

BigInteger A(a), B(b);

(A-B).print();

}

}

图论

## 邻接表类

#define maxn 220005

#define maxm 3000005

struct Edge{ int v,val,next; }E[maxm];

int e; //每个case必须初始化

struct AdjList{

int l[maxn]; //

void insert(int a,int b,int val=0){ //单向边

E[e].v=b; E[e].val=val; E[e].next=l[a]; l[a]=e++;

}

void init(){

memset(l,-1,sizeof(l));

}

}adj;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*模板结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## 最小生成树 prim

//最小生成树 prim，需要临界表\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

struct DD{

int v, val;

DD(int \_v=0,int \_val=0):v(\_v),val(\_val){};

bool operator<(const DD &b)const{

return val > b.val;

}

};

priority\_queue<DD> pq;

int dis[maxn];

bool vst[maxn];

int prim(int n){ //0~n-1的最小生成树，返回总权值

const int inf=1 << 29;

while( !pq.empty() ) pq.pop();

for(int i=0;i<n;i++) dis[i]=inf,vst[i]=false;

pq.push(DD(0,0)); dis[0]=0;

while( !pq.empty() ){

DD temp=pq.top(); pq.pop();

if( vst[temp.v] ) continue;

vst[temp.v]=true; //加入生成树中

for(int p=list[temp.v];p!=-1;p=E[p].next) {

int v=E[p].v;

if( vst[v]==false && dis[v] > E[p].val ){

dis[v]=E[p].val;

pq.push(DD(v,dis[v]));

}

}

}

int ret=0;

for(int i=0;i<n;i++) ret+=dis[i];

return ret;

}//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*模板结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## 次小生成树

//次小生成树 O(n^2) pku1679 Ural 1416

//solu: 先算出一棵mst,然后枚举根dfs,然后枚举添加edge[u][root]删除root->u最大边

#define maxn 510

#define inf 0x1fffffff

struct Edge{ int v,val,x; }E[11000];

int l[maxn],e;

void init(){ e=0; memset(l,-1,sizeof(l)); }

void insert(int a,int b,int val=0){ //单向边

E[e].v=b; E[e].val=val; E[e].x=l[a]; l[a]=e++;

}

int N,M,cost,cost2;//点数N base0,边数,最小,次小

int mi[maxn],pre[maxn],vst[maxn];

int dis[maxn][maxn];

void MST(int N,int r){

fill(pre,pre+N,0);

fill(mi,mi+N,inf);

cost=mi[r]=0;

for(int i=0;i<N;i++){

int v=-1;

for(int j=0;j<N;j++){

if( pre[j]<0 )continue;

if( v<0 || mi[j]<mi[v] ) v=j;

}

if( v!=r ){

insert(pre[v],v,dis[ pre[v] ][v]);

insert(v,pre[v],dis[ pre[v] ][v]);

}

for(int j=0;j<N;j++){

if( pre[j]>=0 && mi[j]>dis[v][j] ) {

mi[j]=dis[v][j];

pre[j]=v;

}

}

cost+=mi[v]; pre[v]=-1; //标记已选择

}

}

void dfs(int u,int r,int D,int maxE){

if( D>=3 ) cost2=min(cost2,cost+dis[u][r]-maxE);

vst[u]=1;

for(int p=l[u];p>=0;p=E[p].x){

int v=E[p].v;

if( vst[v] ) continue;

dfs(v,r,D+1,max(maxE,E[p].val));

}

}

int main(){

while( scanf("%d %d",&N,&M)!=EOF ){

for(int i=0;i<N;i++) fill(dis[i],dis[i]+N,inf);

for(int i=0;i<M;i++){

int a,b,val; scanf("%d %d %d",&a,&b,&val);

a--,b--;

dis[a][b]=dis[b][a]=min(dis[a][b],val);

}

init();

MST(N,0);

cost2=inf;

for(int i=0;i<N;i++){

fill(vst,vst+N,0);

dfs(i,i,1,0); //基于无重边的算法

}

if( cost2==inf ) cost2=-1;

printf("Cost: %d\nCost: %d\n",cost,cost2);

}

}

## 单点k度限制最小生成树

//单点k度限制最小生成树 pku1639 O(VlogE+E+V\*K)

//基于无重边的算法

//solu: 先计算一个度数最小的生成树,然后逐步调整,每次增加一个度数

#define maxn 1100

#define maxm 110000

#define inf 0x3fffffff

struct Edge{ int v,val,sel,x; }E[maxm];//6倍边数

int e; //每个cas初始化

struct AdjList{

int l[maxn];//

void insert(int a,int b,int val=0,int sel=0){

E[e].v=b; E[e].val=val; E[e].sel=sel; E[e].x=l[a]; l[a]=e++;

}

void init(){ memset(l,-1,sizeof(l)); }

}adj,adj2;

struct DD{

int v, val, eid;

DD(int \_v=0,int \_val=0,int \_eid=0):v(\_v),val(\_val),eid(\_eid){};

bool operator<(const DD &b)const{

return val > b.val;

}

};

priority\_queue<DD> pq;

int dis[maxn],vst[maxn];

int prim(int n,int r){ //0~n-1的最小生成树，r为根,返回总权值

fill(dis,dis+n,inf);

fill(vst,vst+n,0);

pq.push(DD(r,0,-1)); dis[r]=0;

int ret=0;

while( !pq.empty() ){

DD tmp=pq.top(); pq.pop();

if( vst[tmp.v] ) continue;

ret+=tmp.val; vst[tmp.v]=1; //加入生成树中

if( tmp.eid>=0 ) E[tmp.eid].sel=E[tmp.eid^1].sel=1; //选择

for(int p=adj.l[tmp.v];p>=0;p=E[p].x) {

int v=E[p].v;

if( vst[v]==0 && dis[v]>E[p].val ){

dis[v]=E[p].val;

pq.push(DD(v,dis[v],p));

}

}

}

return ret;

}

int de,ae; //delete eid, add eid

void dfs(int u,int D,int r,int meid){

vst[u]=1;

if( D>=3 ){

for(int p=adj2.l[u];p>=0;p=E[p].x){

if( E[p].v!=r ) continue;

if( de<0 || E[meid].val-E[p].val>E[de].val-E[ae].val ){

de=meid; ae=p;

}

}

}

for(int p=adj2.l[u];p>=0;p=E[p].x){

if( vst[E[p].v] || !E[p].sel ) continue;

int me=meid;

if( D>=2 && E[p].val>E[me].val ) me=p;

dfs(E[p].v,D+1,r,me);

}

}

int ve[maxm];

int MST\_kD(int n,int r,int k){ //0~n-1, r为根,r有不超过k度限制

const int A=1000000; //大于总边权的一个值\*\*\*\*\*\*\*

for(int p=adj.l[r];p>=0;p=E[p].x) E[p].val+=A;

int ret=prim(n,r), deg=ret/A; //deg当前度数

if( k<deg ) return -1; //无解

ret=ret-A\*deg;

//精简化 adj2

adj2.init();

fill(ve,ve+e,0);

for(int i=0;i<n;i++){

for(int p=adj.l[i];p>=0;p=E[p].x){

if( E[p].val>=A ) E[p].val-=A;

if( (i==r || E[p].sel) && !ve[p] ){

adj2.insert(i,E[p].v,E[p].val,E[p].sel); //保持^1性质

adj2.insert(E[p].v,i,E[p].val,E[p].sel);

ve[p]=ve[p^1]=1;

}

}

}

int ie=e; e+=2;

E[ie].val=-inf; //特别设计的

while( deg<k ){

fill(vst,vst+n,0);

de=ae=-1;

dfs(r,1,r,ie);

if( de<0 || E[de].val-E[ae].val<=0 ) break;

E[de].sel=E[de^1].sel=0;

E[ae].sel=E[ae^1].sel=1;

ret-=(E[de].val-E[ae].val);

deg++;

}

return ret;

}// e=0; adj.init(); adj.insert(...); MST\_kD(n,r,k)...

int main(){

int n,k;

while(cin>>n){

map<string,int> mp;

e=0; adj.init();

for(int i=0;i<n;i++){

string s1,s2; int val;

cin>>s1>>s2>>val;

if( mp.find(s1)==mp.end() ){

int id=mp.size(); mp[s1]=id;

}

if( mp.find(s2)==mp.end() ){

int id=mp.size(); mp[s2]=id;

}

adj.insert(mp[s1],mp[s2],val);

adj.insert(mp[s2],mp[s1],val);

}

cin>>k;

printf("Total miles driven: %d\n",MST\_kD(mp.size(),mp["Park"],k));

}

}

## 最短路dijkstra

//dijkstra

struct DD{

int v,dis;

DD(int \_v=0,int \_dis=0):v(\_v),dis(\_dis){};

bool operator<(const DD &b)const{

return dis > b.dis;

}

};

void dijkstra(int N,int source,int \*dis){//点数为N,1~N,源点为source

#define inf 0x3fffffff

priority\_queue<DD> pq;

for(int i=1;i<=N;i++) dis[i]=inf;

dis[source]=0;

pq.push(DD(source,0));

while( !pq.empty()){

DD temp=pq.top();

pq.pop();

int u=temp.v;

if( temp.dis > dis[u] ) continue; //之前扩展过

for(int p=list[u],v; p!=-1 ;p=E[p].next){

v=E[p].v;

if( dis[v] > dis[u]+E[p].val ){

dis[v]=dis[u]+E[p].val;

pq.push(DD(v,dis[v]));

}

}

}

}

## LCA离线算法 Tarjan

/\*

LCA离线tarjan算法, 可以是森林中两个同一颗树上的点的lca !!! rmq版本的是不支持的!!! 但是有爆栈危险!!

qq数组也是Edge类型,其中的val是询问的序号,即id, query要插双向边

eg.

adjq.insert(a,b,i);

adjq.insert(b,a,i);

...

tarjan.LCA(\_N, adj.l, adjq.l, E, E, ans);

\*/

struct Tarjan{

Edge \*E, \*qq; //qq表示query的边表指针

int pa[maxn], \*l, \*lq, \*ans ;

int find(int u){

return (pa[u]==u) ? u : (pa[u]=find(pa[u])); //爆栈

}

void \_lca(int u){//爆栈

pa[u]=u;

for(int p=l[u];p!=-1;p=E[p].next){

int v=E[p].v;

if( pa[v]>=0 ) continue;

\_lca(v);

pa[v]=u; //其它点和v的lca至少是u以上的点了

}

for(int p=lq[u];p!=-1;p=qq[p].next){

int v=qq[p].v;

if( pa[v]>=0 ) ans[ qq[p].val ] = find(v);

}

}

//0~N-1, 结果存在ans里

void LCA(int N,int \*\_l,int \*\_lq,Edge \*\_E ,int \*\_ans){ //l和lq同一性质的,E和qq同一性质的

fill(pa,pa+N,-88); //负值=未访问

l=\_l; lq=\_lq;

qq=E=\_E; ans=\_ans;

for(int i=0;i<N;i++){

if( pa[i]<0 )

\_lca(i);

}

}

}tarjan; //---

## LCA离线算法 Tarjan 手写递归

//手写了递归,不会爆栈

/\*

LCA离线tarjan算法, 可以是森林中两个同一颗树上的点的lca !!! rmq版本的是不支持的!!! 但是有爆栈危险!!

qq数组也是Edge类型,其中的val是询问的序号,即id, query要插双向边

eg.

adjq.insert(a,b,i);

adjq.insert(b,a,i);

...

tarjan.LCA(\_N, adj2.l, adjq.l, E, E, ans);

\*/

struct Tarjan{

Edge \*E, \*qq; //qq表示query的边表指针, 这里

int pa[maxn], \*l, \*lq, \*ans ;

int st[maxn\*2],top;

int find(int u){

return (pa[u]==u) ? u : (pa[u]=find(pa[u]));

}

void \_lca(int r){ //手写递归

top=0; st[top++]=r; st[top++]=l[r]; pa[r]=r;

while( top ){

if( st[top-1]<0 ){

top-=2;

for(int p=lq[ st[top] ];p!=-1;p=qq[p].next){

int nv=qq[p].v;

if( pa[nv]>=0 ) ans[ qq[p].val ] = find(nv);

}

if( top==0 ) break;

int v=st[top],u=st[top-2]; //u->v

pa[find(v)]=find(u); //u,v次序不能变

}else{

int p=st[top-1], v=E[p].v;

st[top-1]=E[p].next;

if( pa[v]>=0 ) continue;

st[top++]=v; st[top++]=l[v]; pa[v]=v;

}

}

}

//处理0~N-1个节点的lca问题,结果存在ans里

void LCA(int N,int \*\_l,int \*\_lq,Edge \*\_E,int \*\_ans){ //l和lq同一性质的,E和qq同一性质的

fill(pa,pa+N,-88); //负值=未访问

l=\_l; lq=\_lq;

qq=E=\_E; ans=\_ans;

for(int i=0;i<N;i++){

if( pa[i]<0 )

\_lca(i);

}

}

}tarjan; //------

## LCA在线算法 rmq

/\*

Algorithm: LCA RMQ 必须保证是一棵树,完全连通的,不是森林!!

Author: zjut\_DD

\*/

//手写模拟递归了,不会爆栈

struct LCA\_RMQ{

bool visit[maxn]; Edge \*E;

int pa[maxn], list[maxn];//这个list是为了不破坏外部那个list

int deep[maxn], sequence[2\*maxn],ind;//长度为2\*N-1

int mi[maxn][20];//存放的是i，比较的是deep[i]

int wei[maxn];//存放节点i在sequence里的位置

stack<int> st;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void LCA(int N,int \*list,Edge \*E){//0~N-1,进行lca

memcpy(this->list,list,sizeof(this->list));

memset(visit,false,sizeof(visit));

this->E=E;

dfs(0); RMQ(N); //默认根是0

}

void dfs(int root){ //生成dfs序列

while( !st.empty()) st.pop();

pa[root]=-1,deep[root]=0;

st.push(root); ind=0; //序列清零

while(!st.empty()){

visit[st.top()]=true; wei[st.top()]=ind;

sequence[ind++]=st.top(); bool flag=false;

for(int p=list[st.top()]; p!=-1;p=E[p].next){

int v=E[p].v;

if(!visit[v]){

flag=true; pa[v]=st.top();

deep[v]=deep[st.top()]+1;

list[st.top()]=E[p].next;//邻接表改变了

st.push(v); break;

}

}

if(flag) continue;

else st.pop();

}

}

void RMQ(int N){

int sum=2\*N-1, M=get(sum);

for(int i=0;i<sum;i++) mi[i][0]=sequence[i];

for(int g=1;g<=M;g++){

int L=sum-(1<<g);

for(int i=0;i<=L;i++){

mi[i][g]= deep[ mi[i][g-1] ]<deep[ mi[i+(1<<(g-1))][g-1] ] ?

mi[i][g-1]:mi[i+(1<<(g-1))][g-1];//888888888

}

}

}

int get(int L){

return int (log(L\*1.0)/log(2.0));

}

int query(int &a,int &b){//询问a和b的lca,返回编号

int wa=wei[a], wb=wei[b];

if(wa>wb) swap(wa,wb);

int g=get(wb-wa+1);

return deep[ mi[wa][g] ]<deep[ mi[wb-(1<<(g))+1][g] ] ?

mi[wa][g]:mi[wb-(1<<(g))+1][g];//\*\*\*\*和上面不一样的

}

}lca\_rmq;

## 有向图强连通分量Kosaraju算法

/\*

有向图强连通分量Kosaraju算法,两次dfs

必须在反向图上第二次dfs的\*\*\*

\*/

struct Kosaraju{

Edge \*E; bool vst[maxn];

int buf[maxn],top , \*list ,\*label , compo;

AdjList \_adj; //反向图邻接表

//处理0~N-1,结果存在label数组里,label[i]表示i这个节点属于谁,返回收缩后还有几个点base0

int DirGraphShrink(int N,int \*list,Edge \*E,int \*label){

this->E=E; this->list=list; this->label=label;

compo=top=0;

fill(vst,vst+N,false);

for(int i=0;i<N;i++) if( !vst[i] ) \_dfs1(i);

\_adj.init();

for(int i=0;i<N;i++){

for(int p=list[i];p!=-1;p=E[p].next) \_adj.insert(E[p].v, i);

}

fill(vst,vst+N,false);

for(int i=top-1;i>=0;i--) if( !vst[ buf[i] ] ) { \_dfs2( buf[i] ); compo++; }

return compo;

}

void \_dfs1(int u){

vst[u]=true;

for(int p=list[u];p!=-1;p=E[p].next)

if( !vst[ E[p].v ] ) \_dfs1( E[p].v );

buf[top++]=u;

}

void \_dfs2(int u){

vst[u]=true; label[u]=compo;

for(int p=\_adj.list[u];p!=-1;p=E[p].next)

if( !vst[ E[p].v ] ) \_dfs2( E[p].v );

}

}kosaraju;//\*\*\*\*\*\*\*\*\*\*\*模板结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## 有向图强连通分量Tarjan算法

/\*

有向图强连通分量Tarjan算法

如果一个节点是根,那么DFN(深度优先序号)==Low(能达到的最低标号)

\*/

struct Tarjan{

Edge \*E; bool in[maxn];

int DFN[maxn],Low[maxn], st[maxn],top , \*list ,\*label , compo, deep; //深度必须是一个全局量!!!!

//处理0~N-1,结果存在label数组里,label[i]表示i这个节点属于谁,返回收缩后还有几个点base0

int DirGraphShrink(int N,int \*list,Edge \*E,int \*label){

this->list=list; this->E=E; this->label=label;

memset(in,false,sizeof(in)); memset(DFN,0,sizeof(DFN));

top=compo=deep=0;

for(int i=0;i<N;i++) if( DFN[i]==0 ) \_tarjan(i);

return compo;

}

void \_tarjan(int u){

DFN[u]=Low[u]=++deep;

st[top++]=u; in[u]=true;

for(int p=list[u];p!=-1;p=E[p].next){

int v=E[p].v;

if( DFN[v]==0 ){

\_tarjan(v);

Low[u]=min(Low[u],Low[v]);

}else if( in[v] )//防止uv是横叉边\*\*\*\*必须在栈中

Low[u]=min(Low[u],DFN[v]);

}

if( Low[u]==DFN[u] ){

int v;

do{

v=st[--top];

label[v]=compo; in[v]=false;

}while( v!=u );

compo++;

}

}

}dir\_tarjan;//\*\*\*\*\*\*\*\*\*\*\*\*\*\*模板结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## 无向图边双连通

/\*

pku 3177 pku 3352 hdu 4005

\*/

//无向图边双连通(点的集合),以桥为分割. 8字形变成一个点

struct bidirGraph{

int dep[maxn],low[maxn],q[maxn];

int \*l; Edge \*E;

void \_tarjan(int u, int D, int \_i){ //当前点,深度,入边序号

low[u]=dep[u]=D;

for(int i=l[u];i>=0;i=E[i].next) {

if( (i^1)==\_i ) continue;

int v=E[i].v;

if( dep[v]<0 ){

\_tarjan(v,D+1,i);

low[u]=min(low[u],low[v]);

}else

low[u]=min(low[u],dep[v]);

}

}

bool qiao(int u,int v){

if( dep[u]>dep[v] ) swap(u,v);

return low[v]>dep[u];

}

//n点数0~n-1, label每个点在新图中的编号,可以根据label来rebuild

int main(int n,int \*\_l,Edge \*\_E,int \*label){

l=\_l,E=\_E;

for(int i=0;i<n;i++) dep[i]=label[i]=-1;

for(int i=0;i<n;i++) if( dep[i]<0 ) \_tarjan(i,0,-1);

int c=0,s,t;

for(int i=0;i<n;i++){

if( label[i]>=0 ) continue;

s=t=0; q[t++]=i; label[i]=c;

while( s<t ){ //bfs

int u=q[s++],v;

for(int j=l[u];j>=0;j=E[j].next){

v=E[j].v;

if( qiao(u,v) || label[v]>=0 ) continue;

q[t++]=v; label[v]=c;

}

}

c++;

}

return c;

}

}ins;//-----end-----

int label[maxn];

void rebuild(int n){

for(int i=0;i<n;i++){

for(int p=adj.l[i];p>=0;p=E[p].next){

int j=E[p].v;

if( ins.qiao(i,j) ) {

adj2.insert(label[i],label[j],E[p].val);

adj2.insert(label[j],label[i],E[p].val);

}

}

}

}//----

## 无向图点双连通

/\*

无向图点双连通(边的集合),以割点为分割. 8字形变成3个点.

1， 求割点测试过pku 1144, 返回的点双连通个数正确

2， hdu 3749点双连通,p2c,c2e集合测试正确

3, hdu 3686测试 e2c数组正确, rebuild之后的图正确

4, bupt 邀请赛B,割点测试正确

\*/

//p->割点 c->点双连通 e->边

typedef pair<int,int> PII;

set<int> p2c[maxn]; //每个点属于哪些点双连通 //点双连通的编号

set<PII> c2e[maxm]; //每个点双连通有哪些边

map<PII,int> e2c; //每条边属于哪个点双连通

int cp2n[maxn]; //重构图后割点对应的新节点号,非割点一律为-1(cp2n[i]>0可用来判断割点)

int dep[maxn],low[maxn];

struct Bidir\_Tarjan{

int \*l; Edge \*E; //init

int N,cnt,root,top;

PII st[maxm], tmp;

//0~n-1的无向图,返回有几个点双连通base0

//点双联通，边双联通的话调用之后找桥

int bidirGraphCut(int n,int \*\_l,Edge \*\_E){

N=n; l=\_l; E=\_E;

for(int i=0;i<N;i++) dep[i]=cp2n[i]=-1;

e2c.clear();

for(int i=0;i<N;i++) p2c[i].clear(), c2e[i].clear(); //最多N-1个点双连通

cnt=0; //

for(int i=0;i<N;i++){

if( dep[i]>=0 ) continue;

top=0,root=i;

\_dfs(i,0,-1);

assert(top==0);

}

return cnt; //返回点双连通的个数

}

//N个点rebuild之后可能变成2N个点

//rebuild之后前cnt个点都是点双连通的点

//新图中只有 割点和点双连通,是一棵树, 割点和点双连通连边叶子都是点双连通,割点都在内部

int rebuild(AdjList &adj){

int c=cnt; //点双连通序号保持不变

for(int i=0;i<N;i++) if( cp2n[i]>0 ) cp2n[i]=c++;

adj.init(); //

set<int>::iterator it;

for(int i=0;i<N;i++){

if( cp2n[i]==-1 ) continue;

for(it=p2c[i].begin();it!=p2c[i].end();++it){

adj.insert(cp2n[i], \*it);

adj.insert(\*it, cp2n[i]);

}

}

return c; //新图中点的数目

}

bool isBridge(int u,int v){ //u,v有边

if( dep[u]>dep[v] ) swap(u,v);//保证\_dfs中u->v的

return low[v]>dep[u];

}

void \_pop(PII p){

do{

tmp=st[ --top ];

p2c[ tmp.first ].insert(cnt);

p2c[ tmp.second].insert(cnt);

if( tmp.first > tmp.second ) swap(tmp.first, tmp.second);

c2e[ cnt ].insert(tmp);

e2c[ tmp ]=cnt;

}while( top>0 && st[top] != p );

cnt++;

}

void \_dfs(int u,int D,int \_ind){ //当前节点, 深度, 边序号

dep[u]=low[u]=D; int c=0;

for(int p=l[u];p!=-1;p=E[p].next){

if( (p^1) == \_ind ) continue;

int v=E[p].v;

st[ top++ ]=PII(u,v);

if( dep[v]>=0 ){

if( dep[v] > dep[u] ) top--; //这样的边必须弹出来!!!\*\*\*\*\*\*\*\*\*\*

low[u]=min(low[u],dep[v]); //it is dep, not low

continue;

}

\_dfs(v,D+1,p); c++;

if( low[v]>= dep[u] ){

if( u!=root || c>=2 ) cp2n[u]=1; //

\_pop(PII(u,v));

}else{

low[u]=min(low[u],low[v]); //这个是这个里面\_dfs的，故要更新的是low[v],而不是dep[v]

}

}

}

}bidir\_tarjan;//\*\*\*\*\*\*\*\*\*模板结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## 2SAT

#define maxn 2005

#define maxm 4000005

struct Edge{ int v,val,next; }E[maxm];

struct AdjList{

static int eid; //每个case初始化为0

int list[maxn]; //邻接表的核心就是这个list数组

void insert(int a,int b,int val=0){ //单向边\*\*\*\*\*\*\*\*\*\*\*\*\*\*

E[eid].v=b; E[eid].val=val; E[eid].next=list[a]; list[a]=eid++;

}

void init(){

memset(list,-1,sizeof(list));

}

}adj;

int AdjList::eid;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*模板结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//有向图强连通分量Tarjan算法,如果一个节点是根,那么DFN(深度优先序号)==Low(能达到的最低标号)

struct Tarjan{

Edge \*E; bool in[maxn];

int DFN[maxn],Low[maxn], st[maxn],top , \*list ,\*label , compo, deep; //深度必须是一个全局量!!!!

//处理0~N-1,结果存在label数组里,label[i]表示i这个节点属于谁,返回收缩后还有几个点base0

int DirGraphShrink(int N,int \*list,Edge \*E,int \*label){

this->list=list; this->E=E; this->label=label;

memset(in,false,sizeof(in)); memset(DFN,0,sizeof(DFN));

top=compo=deep=0;

for(int i=0;i<N;i++) if( DFN[i]==0 ) \_tarjan(i);

return compo;

}

void \_tarjan(int u){

DFN[u]=Low[u]=++deep;

st[top++]=u; in[u]=true;

for(int p=list[u];p!=-1;p=E[p].next){

int v=E[p].v;

if( DFN[v]==0 ){

\_tarjan(v);

Low[u]=min(Low[u],Low[v]);

}else if( in[v] )

Low[u]=min(Low[u],DFN[v]); //DFN[v]必须是\*\*\*\*不是Low[v]

}

if( Low[u]==DFN[u] ){

int v;

do{

v=st[--top];

label[v]=compo; in[v]=false;

}while( v!=u );

compo++;

}

}

}dir\_tarjan;//\*\*\*\*\*\*\*\*\*\*\*\*\*\*模板结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*

2-sat 复杂度O(E)

共N个人,取N/2个,满足一定限制条件

不能选a+n:

adj.insert(a+n,0); adj.insert(a+n,n); 或者 adj.insert(a+n,a); 都行

\*/

struct TwoSAT{

int N, label[maxn], \_N, another[maxn]; //\_N表示收缩后还有几个点base0

AdjList adj,\_adj; //收缩后的图的邻接表 和逆向表

int \*list, st[maxn],ind ; Edge \*E;

bool vst[maxn], temp[maxn];

int \_h(int t){ //成对的另外一个

if( t>=(N>>1) ) return t-(N>>1);

else return t+(N>>1);

}

bool twoSAT(int N,int \*list, Edge \*E){

this->N=N; this->list=list; this->E=E;

\_N=dir\_tarjan.DirGraphShrink(N,list,E,label);

for(int i=0;i<N;i++)

if( label[i] == label[ \_h(i) ] ) return false;

return true;

}

void construct(bool \*sel){ //构造一组解,sel[i]=true表示第i个被选上了

adj.init(); \_adj.init();

for(int i=0;i<N;i++){ //重构图

for(int p=list[i];p!=-1;p=E[p].next){

int v=E[p].v;

if( label[i] == label[v] ) continue;

adj.insert(label[i], label[v]);

\_adj.insert(label[v], label[i]);

}

}

ind=0; fill(vst,vst+\_N+1,false);

for(int i=0;i<\_N;i++){ //拓扑排序

if( vst[i]==false ) dfs\_topo(i);

}

for(int i=0;i<N;i++){ //收缩后的图结对子

another[ label[i] ]=label[ \_h(i) ];

another[ label[\_h(i)] ]=label[ i ];

}

fill(vst,vst+\_N+1,false);

fill(temp,temp+\_N+1,false);

for(int i=0;i<ind;i++){ //自底向上选择

if( vst[ st[i] ] ) continue;

temp[ st[i] ]=true;

if( !vst[another[ st[i] ]] )\_delete( another[ st[i] ] );

}

for(int i=0;i<N;i++) sel[i] = temp[ label[i] ]; //染色

}

void dfs\_topo(int u){ //拓扑排序

vst[u]=true;

for(int p=adj.list[u];p!=-1;p=E[p].next)

if( vst[ E[p].v ]==false ) dfs\_topo( E[p].v );

st[ind++]=u;

}

void \_delete(int u){

vst[u]=true;

for(int p=\_adj.list[u];p!=-1;p=E[p].next)

if( vst[ E[p].v ]==false ) dfs\_topo( E[p].v );

}

}\_2sat; //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*模板结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## 最小树形图

//最小树形图,复杂度O(VE),可以打印树的构造---------zjut\_DD

//hdu 2121 一个有向图,找一个根然后使最小树形图权值最小

typedef int type;

#define inf 0x7fffffff

#define maxn 1005

#define maxm 10005

struct Edge{//入边表

int iu,iv,ru,rv;//虚(imagine),实(real) u->v

type val;

int next;

}E[2][maxm];

int l[2][maxn],e[2];

int ip[maxn],rp[maxn];//rp是实际的点的父节点,用于输出怎么构造

int sign[maxn],ID[maxn];

type In[maxn];

void init(bool f=false){//默认参数为第一次调用服务

memset(l[f],-1,sizeof(l[f]));

e[f]=0;

}

inline void \_ins(bool f,int iu,int iv,int ru,int rv,type val){

E[f][ e[f] ].iu=iu;

E[f][ e[f] ].iv=iv;

E[f][ e[f] ].ru=ru;

E[f][ e[f] ].rv=rv;

E[f][ e[f] ].val=val;

E[f][ e[f] ].next=l[f][iv];

l[f][iv]=e[f]++;

}

inline void insert(int u,int v,type val){//第一次插入

if(u==v) return;

\_ins(false,u,v,u,v,val);

}

bool Directed\_MST(int root,int N,type &ret){ //0~N-1

ret=0;

bool f=false;//从f到!f

while( true ){

//找最小入边

for(int i=0;i<N;i++){

if(i==root) continue;

In[i]=inf;

int ind;

for(int p=l[f][i];p>=0;p=E[f][p].next){

//下面这个判断是基于最小编号考虑的,hdu2121

if((E[f][p].val < In[i])||(

(E[f][p].val == In[i]) && (E[f][p].rv<E[f][ind].rv) ) )

{

In[i]=E[f][p].val;

ind=p;

}

}

if(In[i] < inf){

ret+=In[i];

ip[ E[f][ind].iv ]=E[f][ind].iu;

rp[ E[f][ind].rv ]=E[f][ind].ru;

}else

return false; //无解

}

//找环,重新标号

memset(sign,-1,sizeof(sign));

memset(ID,-1,sizeof(ID));

int cnt=0;

for(int i=0;i<N;i++){

int v=i;

while(sign[v]==-1 && v!=root){ //没搜到过 & 不是根

sign[v]=i;

v=ip[v];

}

if(sign[v]==i){

int u=v;

do{

ID[v]=cnt;

v=ip[v];

}while(v!=u);

cnt++;

}

}

if( cnt==0 ) break;

for(int i=0;i<N;i++) if(ID[i]==-1) ID[i]=cnt++;

//转移为下一幅图

init(!f);

#define EE (E[f][p])

for(int i=0;i<N;i++){

for(int p=l[f][i];p!=-1;p=E[f][p].next){

if(ID[EE.iu] != ID[EE.iv]){

\_ins(!f,ID[EE.iu],ID[EE.iv],EE.ru,EE.rv,EE.val-In[EE.iv]);

}

}

}

N=cnt;

root=ID[root];

f=!f;

}

return true;

}//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*模板结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int main(){

int N,M;

while(scanf("%d%d",&N,&M)!=EOF) {

init();

int a,b,c;

for(int i=0;i<M;i++) {

scanf("%d%d%d",&a,&b,&c);

insert(a,b,c);

}

int KKK=100000000,ret;

for(int i=0;i<N;i++) insert(N,i,KKK);

Directed\_MST(N,N+1,ret);

if( ret < 2\*KKK) {

int t=0;

while(rp[t] != N) t++; //找根

printf("%d %d\n",ret-KKK,t);

}

else

puts("impossible");

puts("");

}

}

数据结构

## RMQ一维

template<typename T>

struct RMQ{ //类型T必须重载operator<

T mi[65536][16];

void rmq(T\* d, int N){ //预处理，传入一个数组，0~N-1

int M=get(N);

for(int i=0;i<N;i++) mi[i][0]=d[i];

for(int g=1;g<=M;g++){

int L=N-(1<<g);

for(int i=0;i<=L;i++)

mi[i][g]= min(mi[i][g-1], mi[i+(1<<(g-1))][g-1]);

}

}

int get(int L){ //如果查询太多可以打个表，这个很费时间的

return int (log(L\*1.0)/log(2.0));

}

T query(int &a,int &b){//询问[a,b]之间的最小值

if( a>b ) swap(a,b);

int g=get(b-a+1);

return min( mi[a][g], mi[b-(1<<(g))+1][g]);

}

};//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*模板结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## RMQ二维

//二维RMQ

#define maxn 152

#define LOG 8

int \_log[maxn+1];

inline int getlog(int m) {

return \_log[m];

}

void RMQ(int d[][maxn],int M,int N,int ret[][maxn][LOG][LOG] ){ //下标从1开始,M是行,N是列

for(int m=1;m<=maxn;m++) //需要计算很多次时耗时很大!!!需要预处理

\_log[m]=(int)(log(m\*1.0)/log(2.0));

for(int i=1;i<=M;i++)

for(int j=1;j<=N;j++)

ret[i][j][0][0]=d[i][j];

int row=getlog(M), col=getlog(N);

for(int R=0;R<=row;R++){

for(int C=0;C<=col;C++){

for(int i=1;i+(1<<R)-1 <=M;i++){

for(int j=1;j+(1<<C)-1 <=N;j++){

if( R==0 && C==0 ) continue;

if( R==0 )

ret[i][j][R][C]=max(ret[i][j][R][C-1],ret[i][j+(1<<(C-1))][R][C-1]);

else if( C==0 )

ret[i][j][R][C]=max(ret[i][j][R-1][C],ret[i+(1<<(R-1))][j][R-1][C]);

else

ret[i][j][R][C]=max( max(ret[i][j][R-1][C-1],ret[i+(1<<(R-1))][j][R-1][C-1]) ,

max(ret[i][j+(1<<(C-1))][R-1][C-1],

ret[i+(1<<(R-1))][j+(1<<(C-1))][R-1][C-1])

);

}

}

}

}

}

inline int query(int r1,int c1,int r2,int c2,int ret[][maxn][LOG][LOG]){

int R=getlog(r2-r1+1);

int C=getlog(c2-c1+1);

return max( max( ret[r1][c1][R][C], ret[r2-(1<<R)+1][c1][R][C]),

max( ret[r1][c2-(1<<C)+1][R][C],

ret[r2-(1<<R)+1][c2-(1<<C)+1][R][C])

);

}

## 树状数组一维

typedef long long type;

#define maxn 1000006

#define M 65536 //找第k大用的

struct IndexArray{

type s[maxn+5]; int N;

void init(int n){

N=n; fill(s,s+n+1,0);

}

int lowbit(int t){ return t&(-t); }

void update(int i,type val){

while( i<=N ){

s[i]+=val; i+=lowbit(i);

}

}

type get\_sum(int i){

type ret=0;

while( i>0 ){

ret+=s[i]; i-=lowbit(i);

}

return ret;

}

int findK(int k){ //找第k大，assert(k>0)

if( k==0 ) return -1;

int bas=0,step=M;

while( step ){

if( s[bas+step]<k ) k-=s[bas+=step];

step>>=1;

}

return bas+1;

}

}ins1,ins2;//\*\*\*\*\*\*\*\*\*\*\*\*\*模板结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## 树状数组二维

#define maxn 1010

#define maxm 1010

struct IndexArray2{

int s[maxn][maxm],N,M;

inline int lowbit(int t){ return t&(-t); }

void init(int n,int m){

N=n; M=m;

for(int i=1;i<=n;i++) for(int j=0;j<=m;j++) s[i][j]=0;

}

int getsum(int r,int c){ //求和

int ret=0;

for(int tmp;r>=1;r-=lowbit(r)){

for(tmp=c;tmp>=1;tmp-=lowbit(tmp))

ret+=s[r][tmp];

}

return ret;

}

void update(int r,int c,int val){ //[r,c]增加val

for(int tmp;r<=N;r+=lowbit(r)){

for(tmp=c;tmp<=M;tmp+=lowbit(tmp))

s[r][tmp]+=val;

}

}

}ins;//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## 四向循环网状链表

//\*\*\*\*\*\*\*\*\*\*\*四向循环网状链表的节点类\*\*\*\*\*\*\*\*\*\*\*\*

#define MAXR 1005

#define MAXC 1005

struct Node{

int row,col;

Node \*u,\*d,\*l,\*r;

void init(int \_r=0,int \_c=0){ u=d=l=r=this; row=\_r; col=\_c; }

void add\_lr(Node &b){ b.r=r; r->l=&b; r=&b; b.l=this; } //在右边增加一个节点b

void add\_ud(Node &b){ b.d=d; d->u=&b; d=&b; b.u=this; } //在下方增加一个节点b

void remove\_lr(){ l->r=r; r->l=l; } //解除当前节点的左右链接

void remove\_ud(){ u->d=d; d->u=u; } //解除当前节点的上下链接

void resume\_lr(){ l->r=this; r->l=this; } //恢复当前节点的左右链接

void resume\_ud(){ u->d=this; d->u=this; } //恢复当前节点的上下链接

}dd[MAXR][MAXC];

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void remove(int r,int c){ //删除第r行,第c列

//remove row

for(Node \*ptr=dd[r][c].r;ptr!=&dd[r][c];ptr=ptr->r){

ptr->remove\_ud();

}

//remove col

for(Node \*ptr=dd[r][c].u;ptr!=&dd[r][c];ptr=ptr->u){

ptr->remove\_lr();

}

}

## 平衡树SBT

/\*

size balanced tree -------------zjut\_DD实现

为什么maintain不会死循环,这个可以从\_rot函数看到

定义A=sigma(depth(i))/N,那么每次\_rot都会使得A减少,故不会死循环

\*/

#include<iostream>

#include<cstdio>

using namespace std;

#define maxn 50005\*20

typedef int type;

struct Node{

type key;

Node\* ch[2]; int size;

friend int SZ(Node\* p){ return p==NULL ? 0 : p->size; }

void init(type \_key=0){ key=\_key; size=1; ch[0]=ch[1]=NULL; }//必须初始化

void update() { size=1+SZ(ch[0])+SZ(ch[1]); }

}dd[maxn];

int e; //初始化为0

Node\* \_alloc(type val){ dd[e].init(val); return &dd[e++]; }

void \_rot(Node\* &p, int f){ //f=0为 往 左旋

Node \*tmp=p->ch[!f];

p->ch[!f]=tmp->ch[f]; tmp->ch[f]=p;

p->update(); tmp->update();

p=tmp;

}

void \_m(Node\* &p, int f){ //维护函数, 检查p->ch[f]是否满足>=p->c[!f]->ch[0,1]

if( !p || !p->ch[!f] ) return;

if( SZ(p->ch[f]) < SZ(p->ch[!f]->ch[!f]) ){ //手画画

\_rot(p,f);

\_m(p->ch[f],0); \_m(p->ch[f],1);

}else if( SZ(p->ch[f]) < SZ(p->ch[!f]->ch[f]) ) {

\_rot(p->ch[!f],!f); \_rot(p,f);

\_m(p->ch[!f],f); \_m(p->ch[f],!f);

}else return; //

\_m(p,0); \_m(p,1); //保驾护航

}

void insert(Node\* &p,type v){ //插入

if(p==NULL) { p=\_alloc(v); return; }

bool f= (v>p->key);

insert(p->ch[f],v);

p->update(); \_m(p,!f);

}

type \_rev(Node\* &p){

if( p->ch[0] ){

type tmp=\_rev(p->ch[0]);

p->update();

\_m(p,0); return tmp;

}else{

type tmp=p->key; p=p->ch[1]; return tmp;

}

}

void remove(Node\* &p,type key){ //删除,如果有多个则删除一个

if( p==NULL ) return; //删除一个SBT里没有的值

if(key < p->key){

remove(p->ch[0],key); p->update(); \_m(p,0);

}else if(key > p->key){

remove(p->ch[1],key); p->update(); \_m(p,1);

}else{

if( p->ch[0]==NULL) p=p->ch[1];

else if( p->ch[1]==NULL) p=p->ch[0];

else {

p->key=\_rev(p->ch[1]);

p->update(); \_m(p,1);

}

}

}///-----------SBT----------------

int lower\_bound(Node\*p,type key){ //查找排名

if( !p ) return 1;

if( key>p->key ) return SZ(p->ch[0])+1+lower\_bound(p->ch[1],key);

else return lower\_bound(p->ch[0],key);

}

int upper\_bound(Node\*p,type key){

return lower\_bound(p,key+1);

}

type find(Node \*p,int r){ //查询排名为r的元素

if( r<=SZ(p->ch[0]) ) return find(p->ch[0],r);

else r-=SZ(p->ch[0]);

if( r==1 ) return p->key; else r--;

return find(p->ch[1],r);

}///-----------部分查询函数-------

int main(){

int Q,val,r;

while( scanf("%d",&Q)!=EOF ){

e=0; //初始化

Node \*root=NULL;

for(int i=1;i<=Q;i++){

char op; scanf(" %c",&op);

if( op=='I' ){ //插入

scanf("%d",&val); insert(root,val);

}else if( op=='D' ){ //删除

scanf("%d",&val); remove(root,val);

}else if( op=='N' ){ //有几个数<=val

scanf("%d",&val); printf("%d\n",upper\_bound(root,val)-1);

}else if( op=='F' ){ //查找排名为r的元素

scanf("%d",&r); printf("%d\n",find(root,r));

}

}

}

}

## 平衡树Treap

//Treap = Tree+Heap

//如果Hkey确定了，那么Treap就是确定的

#define maxn 110000

struct Node{

int Tkey,Hkey;//二叉树关键字和堆关键字

Node \*ls,\*rs;

void init(int \_T,int \_H,Node\* \_ls,Node\* \_rs){

Tkey=\_T,Hkey=\_H;

ls=\_ls,rs=\_rs;

}

}node[maxn],\*root;

int e;

Node\* Insert(Node\* r,Node \*p){ //r表示树根，p表示要插入节点的指针

if(r==NULL)//插入空树

return r=p;

if(p->Tkey < r->Tkey)

r->ls=Insert(r->ls,p);//可能会旋转，所以儿子可能会变

else

r->rs=Insert(r->rs,p);

if(r->ls!=NULL && r->ls->Hkey > r->Hkey){ //右旋

Node\* temp=r->ls;//temp指向r的左儿子

r->ls=temp->rs;

temp->rs=r;

return temp;

}

if(r->rs!=NULL && r->rs->Hkey > r->Hkey){ //左旋

Node\* temp=r->rs;//temp指向r的右儿子

r->rs=temp->ls;

temp->ls=r;

return temp;

}

return r;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

inline int Hkey(Node\* p){

return p ? p->Hkey : -0x7ffffffe;

}

//删除节点操作，传入要删除的节点指针，把它旋转到叶节点，然后删除

Node\* Delete(Node \*r){//删除节点r（指针）

if(r->ls==NULL && r->rs==NULL)//已经是叶节点了

return NULL;

if(Hkey(r->ls) > Hkey(r->rs) ){

Node\* temp=r->ls;//temp指向r的左儿子

r->ls=temp->rs;

temp->rs=Delete(r);

return temp;

}else{

Node\* temp=r->rs;//temp指向r的右儿子

r->rs=temp->ls;

temp->ls=Delete(r);

return temp;

}

}

## 笛卡尔树

/\*

笛卡尔树(就是一棵Treap的O(n)构造过程)

zju 1985 一列宽为1高为height[i]的木棍,求最大延伸面积,

利用了T=i,H=height[i],然后父子之间夹住的区间......类似splay的想法...

\*/

#include<iostream>

#include<stdio.h>

using namespace std;

#define maxn 110000

struct DD{

int T,H; DD \*ch[2]; //Tree,Heap

void init(int \_T,int \_H){ T=\_T; H=\_H; ch[0]=ch[1]=NULL; }

friend bool operator<(const DD &a,const DD &b){

return a.T < b.T;

}

}dd[maxn],\*root;

DD \*st[maxn]; int top;

void makeTree(int n){ //节点0~n-1

top=0;

for(int i=0;i<n;i++){

DD \*lson=NULL;

while( top>0 ){

if( dd[i].H < st[top-1]->H ){

lson=st[--top];

}else break;

}

if( top==0 ){

dd[i].ch[0]=lson;

root=&dd[i];

}else{

st[top-1]->ch[1]=&dd[i];

dd[i].ch[0]=lson;

}

st[top++]=&dd[i];

}

}

long long ans;

int dfs(DD\*p){

if( p==NULL ) return 0;

int num=1;

num+=dfs(p->ch[0]);

num+=dfs(p->ch[1]);

ans=max(ans,(long long)num \* p->H);

return num;

}

int main(){

int n;

while( scanf("%d",&n) && n){

for(int i=0;i<n;i++) {

int H; scanf("%d",&H);

dd[i].init(i,H);

}

makeTree(n);

ans=0;

dfs(root);

printf("%lld\n",ans);

}

}

## 伸展树

//干模板1.5kb 已测 hdu 1890 zjut 1708

//给定n个数,每次对[a,b]区间的数同时+val, 查询[a,b]的和

/\*

几个要点:

1, 修改之后一定要splay到根,因为f的儿子变了但没有对f实行update操作!!!

2, 只有tmp->splay(NULL)才是绝对安全的,因为null不用update

3, 我们在find或者其他操作之后,从根到这个点的路径上已经没有延迟信息了!!!

\*/

typedef long long int64;

#define maxn 110000

struct Node{

int val;

int64 sum; //统计信息

int64 add; //延迟标记

//上面数据

Node\* ch[2], \*f; int size;

void init(){

f=ch[0]=ch[1]=NULL; size=1; //必须

add=0; //延迟标记清0

}

void rot(int c){//c=0左旋,c=1右旋

Node \*y=f,\*z=y->f;

//y->push\_down(); push\_down();//y在上

y->ch[!c]=ch[c]; if( ch[c] ) ch[c]->f=y;

f = z ; if( z ) if( y==z->ch[0]) z->ch[0]=this; else z->ch[1]=this;

ch[c]=y; y->f=this;

y->update();//x节点最后才维护的

}

void splay(Node \*fa){//将自己旋转到father下面

for(push\_down(); f != fa; ){

if( f->f == fa) {

if( f->ch[0] == this) rot(1);

else rot(0);

}else{

Node \*y=f , \*z=y->f;

if( y->ch[0] == this){

if( z->ch[0]==y ) { y->rot(1); rot(1); }

else{ rot(1); rot(0); }

}else{

if( z->ch[0]!=y ) { y->rot(0); rot(0); }

else{ rot(0); rot(1); }

}

}

}

update(); //最后维护这个x节点

}

Node\* find(int r){ //找到数列中第rank大的地址 base1

push\_down();

int L=0; if( ch[0] ) L+=ch[0]->size;

if( r<=L ) return ch[0]->find(r);

if( r==L+1) return this;

return ch[1]->find(r-L-1);

}

void update(){

size=1;

if( ch[0] ) size+=ch[0]->size;

if( ch[1] ) size+=ch[1]->size; //以上必须

sum=val;

if( ch[0] ) sum+=ch[0]->sum;

if( ch[1] ) sum+=ch[1]->sum;

}

void push\_down(){//将延迟信息push下去

if( add ){

if( ch[0] ) {

ch[0]->add+=add; ch[0]->val+=add; ch[0]->sum+=ch[0]->size \* add;

}

if( ch[1] ) {

ch[1]->add+=add; ch[1]->val+=add; ch[1]->sum+=ch[1]->size \* add;

}

add=0;

}

}

}dd[maxn];

int e; //初始化

inline Node\* \_alloc() { dd[e].init(); return &dd[e++]; }

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*伸展树模板结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// e=0; Node \*root=\_make(0,n+1);

int d[maxn];

Node\* \_make(int l,int r){

if( l>r ) return NULL;

int mid=(l+r)/2;

Node\* p=\_alloc(); p->val=d[mid];

p->ch[0]=\_make(l,mid-1); if( p->ch[0] ) p->ch[0]->f=p;

p->ch[1]=\_make(mid+1,r); if( p->ch[1] ) p->ch[1]->f=p;

p->update(); return p;

}

int main(){

int n,m,Te=1;

while( scanf("%d %d",&n,&m)!=EOF ){

for(int i=1;i<=n;i++) scanf("%d",d+i);

d[0]=d[n+1]=0;

e=0;

Node \*root=\_make(0,n+1),\*tmp;

printf("case #%d:\n",Te++);

for(int i=1;i<=m;i++){

char op; int a,b,val;

scanf(" %c",&op);

if( op=='a' ){

scanf("%d %d %d",&a,&b,&val); a++,b++;

tmp=root->find(a-1);

tmp->splay(NULL); root=tmp;

tmp=root->find(b+1);

tmp->splay(root);

tmp=tmp->ch[0];

tmp->val+=val; tmp->add+=val; tmp->sum+=tmp->size \* (int64)val;

tmp->splay(NULL); root=tmp;

}else{

scanf("%d %d",&a,&b); a++,b++;

tmp=root->find(a-1);

tmp->splay(NULL); root=tmp;

tmp=root->find(b+1);

tmp->splay(root);

tmp=tmp->ch[0];

printf("%lld\n",tmp->sum);

}

}

}

}

## 动态树

//动态树问题 hdu2475 动态查询根是谁 hdu3966也AC了,延迟标记没问题

//Link-Cut-Tree

#define maxn 100005

struct Node{

int id;

//上面

Node \*up;

Node\* ch[2], \*f; int size;

void init(){

up=f=ch[0]=ch[1]=NULL; size=1;

}

void rot(int c){ //c=0左旋,c=1右旋

Node \*y=f,\*z=y->f;

//y->push\_down(); push\_down();//如果有延迟信息,则不能删

y->ch[!c]=ch[c]; if( ch[c] ) ch[c]->f=y;

f = z ; if( z ) if( y==z->ch[0]) z->ch[0]=this; else z->ch[1]=this;

ch[c]=y; y->f=this;

y->update();//x节点最后才维护的

}

void splay(Node \*fa){//将自己旋转到f下面

for(push\_down(); f != fa; ){

if( f->f == fa) {

//f->push\_down(); push\_down(); //如果有rev操作,这个不能删了

if( f->ch[0] == this) rot(1);

else rot(0);

}else{

Node \*y=f , \*z=y->f;

//z->push\_down(); y->push\_down(); push\_down();//如果有rev操作,这个不能删了

if( y->ch[0] == this){

if( z->ch[0]==y ) { y->rot(1); rot(1); }

else{ rot(1); rot(0); }

}else{

if( z->ch[0]!=y ) { y->rot(0); rot(0); }

else{ rot(0); rot(1); }

}

}

}

update(); //最后维护这个x节点

}

Node\* find(int r){ //找到数列中第rank大的地址

//push\_down(); //如果有延迟信息,则不能删

int L=0; if( ch[0] ) L+=ch[0]->size;

if( r<=L ) return ch[0]->find(r);

if( r==L+1) return this;

return ch[1]->find(r-(L+1));

}

void update(){

size=1;

if( ch[0] ) size+=ch[0]->size;

if( ch[1] ) size+=ch[1]->size; //以上必须

}

void push\_down(){//将延迟标记push下去

}

}dd[maxn];

//一条preferred path在伸展树中是从小大的,比如A->B->C->D,则A表示最小,D最大

void init(int n){ //初始化,需要重写

for(int i=0;i<=n;i++) dd[i].init(),dd[i].id=i;

}

void access(Node \*p){//s是p下面一条路径的根节点

Node \*s=NULL;

while( p!=NULL ){

p->splay(NULL);

if( p->ch[1] ) { //安顿后继

p->ch[1]->find(1)->splay(p);

p->ch[1]->up=p; p->ch[1]->f=NULL; //f=NULL表示将它设置为树根

} //p是根

p->ch[1]=s;

if( s ) { s->up=NULL; s->f=p; s->splay(NULL); }//连接 s旋转到根,保持各种更新，延迟信息

else{ s=p; s->update(); } //s是根了

p=s->find(1)->up;

}

}

Node\* find\_root(Node \*s){ //动态树的树根

access(s); s->splay(NULL);

return s->find(1);

}

void link(Node \*s, Node \*f){ //增加f->s, 保证s是一个根,

s->up=f;

access(s); //保持各种更新，延迟信息

}

void cut(Node \*s){ //除去s和其父亲的关系

access(s); s->splay(NULL);

if( s->ch[0]==NULL ) return; //没有父亲

s->ch[0]->f=NULL;

s->ch[0]=s->up=NULL;

}

bool isAncester(Node \*f,Node \*s){ //判断f是不是s的祖先

access(s);

f->splay(NULL);

while( s->f )s=s->f;

return s==f;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*模板结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int main(){

int n,Te=0;

while( scanf("%d",&n)!=EOF ){

init(n);

for(int i=1;i<=n;i++){

int f; scanf("%d",&f);

dd[i].up=&dd[f];

}

int Q; scanf("%d",&Q);

if( Te++) puts("");

while( Q-- ){

char sign[10];

scanf(" %s",sign);

if( sign[0]=='M' ){

int a,b; scanf("%d%d",&a,&b);

if( isAncester(&dd[a],&dd[b]) ) continue;

cut(&dd[a]);

link(&dd[a],&dd[b]);

}else{

int a; scanf("%d",&a);

access(&dd[a]); dd[a].splay(NULL);

printf("%d\n",dd[a].find(2)->id);

}

}

}

}

## 划分树

/\*

划分树

从整体来看是一棵二叉树,最顶上是一个节点,第二层是两个节点...

其次还是一棵划分树,第k+1层是对第k层的一个划分(二分)

\*/

//hdu 3473 给你区间[a,b],求sigma| x-val[i] |的最小值

typedef long long int64;

#define maxn 100005

struct Node { int L,R; } node[5\*maxn] ;//二叉树节点

int val[18][maxn],sorted[maxn];

int \_left[18][maxn];//分到左边有几个

int64 sum[18][maxn];//和

void divide(int d,int hid,int L,int R){ //要对第几层d划分,左右边界,二叉树节点下标heap-id

node[hid].L=L; node[hid].R=R;

if( L == R ) return ;

int mid=(L+R)>>1,same=0,midval=sorted[mid];

for(int i=L;i<=R;i++) if( val[d][i] < midval ) same++;

same=mid-L+1 - same;//有这些个和midval一样的要分到左边

int lp=L,rp=mid+1;

for(int i=L;i<=R;i++){

if( i==L ) \_left[d][i]=0;

else \_left[d][i]=\_left[d][i-1];

if( val[d][i]<midval || (val[d][i]==midval && same!=0) ){

val[d+1][lp++]=val[d][i];

\_left[d][i]++;

if( val[d][i]==midval ) same--;

}else{

val[d+1][rp++]=val[d][i];

}

}

divide(d+1,hid\*2, L, mid);

divide(d+1,hid\*2+1,mid+1,R);

//以上是必须的

}

int64 ans=0;

int query(int d,int hid,int L,int R,int rank){

int toL1,toL2,toR1,toR2; // |---\*\*\*\*\*\*-----|,\*为我们的[L,R]区间,

int BL=node[hid].L , BR=node[hid].R , Bmid=(BL+BR)>>1 ;//bound 区间边界

if( L==BL ){

toL1=toR1=0;

toL2=\_left[d][R]; toR2=(R-L+1)-toL2;

}else{

toL1=\_left[d][L-1]; toR1=L-BL - toL1;

toL2=\_left[d][R]-toL1; toR2=(R-L+1)-toL2;

}

//以上这些是必须的

if(L==R ) return val[d][L];

if( toL2 >= rank){ //rank分在左边

int key=query(d+1,hid\*2,BL+toL1,BL+toL1+toL2-1,rank);

ans+=sum[d+1][Bmid+toR1+toR2]-sum[d+1][Bmid+toR1] - (int64)key \* toR2;

return key;

}else{

int key=query(d+1,hid\*2+1,Bmid+toR1+1,Bmid+toR1+toR2,rank-toL2);

ans+=(int64)key\*toL2 - (sum[d+1][BL+toL1+toL2-1]-sum[d+1][BL+toL1-1]);

return key;

}

}

int main(){

int cas,Te=1,N,Q,L,R;

scanf("%d",&cas);

while( cas-- ){

scanf("%d",&N);

for(int i=1;i<=N;i++) scanf("%d",val[0]+i);

memcpy(sorted,val[0],sizeof(sorted));

sort(sorted+1,sorted+1+N);

divide(0,1,1,N);

for(int i=0;i<18;i++){

for(int j=1;j<=N;j++)

sum[i][j]=sum[i][j-1]+val[i][j];

}

scanf("%d",&Q);

printf("Case #%d:\n",Te++);

while( Q-- ){

scanf("%d%d",&L,&R);

L++,R++;

ans=0;

query(0,1,L,R,(R-L+2)/2);

printf("%I64d\n",ans);

}

puts("");

}

}

## 左偏树

//zjut 1567 数列调整

//可并堆（默认最大堆）, 具有左偏性质的堆序二叉树 --zjut\_DD

#define maxn 55002

typedef int type; //type要有operator<

struct Node{

type key;

//以上

int dis; Node \*ls,\*rs,\*pa;

void init(type \_key){ key=\_key; dis=0; ls=rs=pa=NULL; }

}dd[maxn];

int e; //每次初始化为0

struct LeftistTree{

Node \*root; int size;

LeftistTree(){ clear(); }

void clear(){ root=NULL; size=0; }

void push(type key){ //插入一个元素

size++;

dd[e].init(key);

root=\_Merge(root,&dd[e++]);

}

void pop(){ //删除堆首元素

size--; root=\_Merge(root->ls,root->rs);

}

type& top()const{ return root->key; } //取堆首元素

void putAll(LeftistTree &L){ //插入整棵树

size+=L.size; root=\_Merge(root,L.root);

}

int \_d(Node \*p){ return p?p->dis:-1; }

Node\* \_Merge(Node\* A,Node\* B){

if(A==NULL) return B;

if(B==NULL) return A;

if( A->key < B->key ) swap(A,B);//主客调换

A->rs=\_Merge(A->rs,B); A->rs->pa=A;

if( \_d(A->ls) < \_d(A->rs) ) swap(A->ls,A->rs);

A->dis=\_d(A->rs)+1; return A;

}

}st[maxn];//\*\*\*\*\*\*\*\*\*\*\*\*\*模板结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int a[maxn],N;

int l[maxn],r[maxn]; //第i棵树在a数列中的左右边界

int work(){ //处理上升序列的情况

int t=0;

for(int i=1;i<=N;i++){

st[t].clear(); l[t]=r[t]=i;

st[t].push(a[i]);

while( t && st[t].top()<st[t-1].top()){

st[t-1].putAll(st[t]);

r[t-1]=r[t]; t--;

while( st[t].size > (r[t]-l[t]+2)/2 ) st[t].pop();

}

t++;

}

int ret=0;

for(int i=0;i<t;i++){

for(int j=l[i];j<=r[i];j++)

ret+=abs(st[i].top()-a[j]);

}

return ret;

}

int main(){

while(scanf("%d",&N)!=EOF){

for(int i=1;i<=N;i++) scanf("%d",a+i);

int ans=0x3fffffff;

e=0; ans=min(ans,work());

reverse(a+1,a+1+N);

e=0; ans=min(ans,work());

printf("%d\n",ans);

}

}

## 线段树 下标

#define maxn 110000

int v[maxn];

//单棵线段树模板, zjut1708 n个数,每次对[a,b]区间的数同时+val, 查询[a,b]的和

typedef long long int64;

struct \_Node{

int64 add; //延迟信息(作用域为下方,不包括自身)

int64 sum; //统计信息

//

int l,r; //boudd

void init(int \_l,int \_r){

l=\_l, r=\_r; add=0;//初始化时,延迟信息必须清0

}

int len() { return r-l+1; }

int mid() { return (l+r)>>1; }

}dd[maxn\*5];

void push\_down(int i){ //保证有儿子

int64 &add=dd[i].add;

if( add==0 ) return;

dd[i\*2 ].add+=add; dd[i\*2 ].sum+=add\*dd[i\*2 ].len(); //push下去的时候,统计信息也要更新

dd[i\*2+1].add+=add; dd[i\*2+1].sum+=add\*dd[i\*2+1].len();

add=0; //去掉

}

void update(int i){ //保证有儿子

dd[i].sum=dd[i\*2].sum + dd[i\*2+1].sum;

}

void makeTree(int i,int l,int r){

dd[i].init(l,r);

if( l==r ) { dd[i].sum=v[l]; return; }

int mid=dd[i].mid();

makeTree(i\*2,l,mid);

makeTree(i\*2+1,mid+1,r);

update(i);

}

void modify(int i,int l,int r,int val){

if( dd[i].l==l && dd[i].r==r ){

dd[i].add+=val; dd[i].sum+=(int64)val\*dd[i].len();

return;

}

push\_down(i);

int mid=dd[i].mid();

if( r<=mid) modify(i\*2,l,r,val);

else if( l>mid ) modify(i\*2+1,l,r,val);

else {

modify(i\*2,l,mid,val);

modify(i\*2+1,mid+1,r,val);

}

update(i);

}

int64 query(int i,int l,int r){

if( dd[i].l==l && dd[i].r==r ) return dd[i].sum;

push\_down(i);

int mid=dd[i].mid();

if( r<=mid ) return query(i\*2,l,r);

else if( l>mid ) return query(i\*2+1,l,r);

else return query(i\*2,l,mid) + query(i\*2+1,mid+1,r);

}

//\*\*\*\*\*\*模板结束\*\*\*\*\*\*\*

int main(){

int N,Q,Te=1;

while( scanf("%d %d",&N,&Q)!=EOF ){

for(int i=1;i<=N;i++) scanf("%d",v+i);

makeTree(1,1,N);

printf("case #%d:\n",Te++);

for(int i=1;i<=Q;i++){

char op; scanf(" %c",&op);

int l,r,val;

if( op=='a' ){ //对区间加操作

scanf("%d %d %d",&l,&r,&val);

modify(1,l,r,val);

}else{ //区间求和

scanf("%d %d",&l,&r);

printf("%lld\n",query(1,l,r));

}

}

}

}

## 线段树 指针

//线段树模板, 给定n个数,每次对[a,b]区间的数同时+val, 查询[a,b]的和

//此线段树模板在需要多棵树的时候(树的的路径剖分等)使用

typedef long long int64;

#define maxn 110000

int d[maxn];

struct Node{

int64 add; //延迟操作信息

int64 sum; //统计信息

//

int l,r;

Node \*ls,\*rs; //左右儿子

void init(int \_l,int \_r){

l=\_l,r=\_r; ls=rs=NULL; add=0; //延迟信息必须清0!!!

}

int mid(){ return (l+r)>>1; }

int len(){ return (r-l+1); }

void push\_down(){ //保证有儿子

if( add==0 ) return;

ls->add+=add; ls->sum+=ls->len()\*add; //push下去的时候,统计信息也要更新

rs->add+=add; rs->sum+=rs->len()\*add;

add=0; //去掉

}

void update(){ //保证有儿子

sum=ls->sum+rs->sum; //更新统计信息

}

void modify(int a,int b,int val){

if( l==a && r==b ){

add+=val; sum+=(int64)len()\*val;

return;

}

push\_down(); //要进入子树,先要把树根的延迟信息push下去

if( b<=mid() ) ls->modify(a,b,val);

else if( a>mid() ) rs->modify(a,b,val);

else {

ls->modify(a,mid(),val);

rs->modify(mid()+1,b,val);

}

update();

}

int64 query(int a,int b){

if( l==a && r==b ) return sum;

push\_down(); //因为查询的时候没有更新任何信息, 所以不用update上来

if( b<=mid() ) return ls->query(a,b);

else if( a>mid() ) return rs->query(a,b);

else {

return ls->query(a,mid()) + rs->query(mid()+1,b);

}

}

}dd[maxn\*5];

int e; // 初始化

Node\* \_alloc(int l,int r) { dd[e].init(l,r); return &dd[e++]; }

Node\* \_make(int l,int r){ //构造一棵树,并返回树根

Node\* p=\_alloc(l,r);

if( l==r ) {

p->add=0,p->sum=d[l]; return p; //这里用到d数组,需要初始化

}

p->ls=\_make(l,p->mid());

p->rs=\_make(p->mid()+1,r);

p->update();

return p;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*线段树模板结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//初始化 e=0; 建树方式 Node\* root=\_make(1,N);

int main(){

int N,Q,Te=1;

while( scanf("%d %d",&N,&Q)!=EOF ){

e=0; //

for(int i=1;i<=N;i++) scanf("%d",d+i);

Node\* root=\_make(1,N);

printf("case #%d:\n",Te++);

for(int i=1;i<=Q;i++){

char op; scanf(" %c",&op);

int l,r,val;

if( op=='a' ){ //对区间加操作

scanf("%d %d %d",&l,&r,&val);

root->modify(l,r,val);

}else{ //区间求和

scanf("%d %d",&l,&r);

printf("%lld\n",root->query(l,r));

}

}

}

}

## 四分树

//二维线段树 四分树 w=h=2^n

//两种操作,给x,y点增加一个数值,查询(x1,x2,y1,y2)矩形内的数字和

typedef long long int64;

#define maxn 510000

struct Node{

int64 sum;

int x1,x2,y1,y2;

Node \*lu,\*ld,\*ru,\*rd;

void init(int \_x1,int \_x2,int \_y1,int \_y2){

x1=\_x1,x2=\_x2;

y1=\_y1,y2=\_y2;

lu=ld=ru=rd=NULL;

sum=0;

}

void update(){

sum=0;

if( lu ) sum+=lu->sum;

if( ld ) sum+=ld->sum;

if( ru ) sum+=ru->sum;

if( rd ) sum+=rd->sum;

}

}dd[maxn],\*root;

int e; //初始化为0

Node\* \_alloc(int x1,int x2,int y1,int y2){

dd[e].init(x1,x2,y1,y2); return &dd[e++];

}

void insert(Node \*p,int x,int y,int g){

if( p->x1==p->x2 && p->y1==p->y2 ) {

p->sum+=g;

return;

}

int midx=(p->x1+p->x2)/2;

int midy=(p->y1+p->y2)/2;

if( x<=midx && y>midy ) {

if( p->lu==NULL ) p->lu=\_alloc(p->x1,midx,midy+1,p->y2);

insert(p->lu,x,y,g);

}else if( x<=midx && y<=midy ){

if( p->ld==NULL ) p->ld=\_alloc(p->x1,midx,p->y1,midy);

insert(p->ld,x,y,g);

}else if( x>midx && y>midy ){

if( p->ru==NULL ) p->ru=\_alloc(midx+1,p->x2,midy+1,p->y2);

insert(p->ru,x,y,g);

}else{

if( p->rd==NULL ) p->rd=\_alloc(midx+1,p->x2,p->y1,midy);

insert(p->rd,x,y,g);

}

p->update();

}

int64 query(Node \*p,int x1,int x2,int y1,int y2){

if( p==NULL ) return 0;

x1=max(x1,p->x1);

y1=max(y1,p->y1);

x2=min(x2,p->x2);

y2=min(y2,p->y2);

if( x1>x2 || y1>y2 ) return 0;

if( x1==p->x1 && x2==p->x2 && y1==p->y1 && y2==p->y2 ) return p->sum;

return query(p->lu,x1,x2,y1,y2) + query(p->ld,x1,x2,y1,y2)+

query(p->ru,x1,x2,y1,y2) + query(p->rd,x1,x2,y1,y2);

}

int main(){

char buf[222];

bool ins=false;

while( gets(buf) ){

if( buf[0]=='E' ) break;

if( buf[0]=='I' ){

e=0;

dd[e].init(1,32768,1,32768); //32768=2^31

root=&dd[e++];

ins=true; continue;

}

if( buf[0]=='Q' ){

ins=false; continue;

}

if( ins ) {

int x,y,g;

sscanf(buf,"%d %d %d",&x,&y,&g);

insert(root,x,y,g);

}else{

int x1,x2,y1,y2;

sscanf(buf,"%d %d %d %d",&x1,&x2,&y1,&y2);

printf("%lld\n",query(root,x1,x2,y1,y2));

}

}

}

## 快速vector模板

//下面这个vector适用于最大长度有限的场合

template<typename T, int max\_size=40>

struct Vector{

T \_buf[max\_size];

int \_s;

Vector(){ clear(); }

void clear(){ \_s=0; }

void push\_back(T val){ \_buf[\_s++]=val; }

int size(){ return \_s; }

T& operator[](int i){ return \_buf[i]; }

}; //快速vector模板\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## vector做平衡树

//vector做平衡树用,zju2112树套树可以AC,在10^5内,效率是SBT的1/2!!擦

vector<int>::iterator it;

void insert(vector<int> &v,int val){ //插入

it=lower\_bound(v.begin(),v.end(),val);

v.insert(it,val);

}

void remove(vector<int> &v,int val){ //删除

it=lower\_bound(v.begin(),v.end(),val);

if( it!=v.end() && \*it == val ) v.erase(it);

}

int find(vector<int> &v,int val){ // 查询<=val的个数

if( v.size()==0 ) return 0;

int l=0,r=v.size()-1;

if( v[r]<=val ) return r+1;

if( v[l]> val ) return 0;

while( l+1 < r ){

int mid=(l+r)/2;

if( v[mid]<=val ) l=mid;

else r=mid;

}

return l+1;

}//\*\*\*\*\*\*\*\*结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## 哈希表

//哈希表

const int mod=99997; //映射范围

const int maxn=200000; //节点总数

template<class TK,class TV>

struct HashMap{

struct Entry{ TK k; TV v; int x; } dd[maxn];

int l[mod],e;

int (\*hash) (TK);

HashMap() { clear(); }

void clear(){ e=0; memset(l,-1,sizeof(l)); }

void setCallBack(int (\*func)(TK) ){ hash=func; }

int find(TK k){ //查找下标,没有返回-1

int i=hash(k);

for(int p=l[i];p>=0;p=dd[p].x)

if( dd[p].k==k ) return p;

return -1;

}

void insert(TK k,TV v){ //插入

int i=hash(k);

dd[e].k=k; dd[e].v=v; dd[e].x=l[i]; l[i]=e++;

}

TV& operator[] (TK k){

int t=find(k);

if( t==-1 ) { insert(k,TV()); return dd[e-1].v;}

else return dd[t].v;

}

};

inline int h(int V){ //需要修改

return V%mod;

}//HashMap<int,int> mp; mp.setCallBack(h); ...

## 单调队列,单调dp

struct LinearQueue{ //单调队列

int s,t; // [s,t)

pair<int,int64> buf[maxn]; //index value

void init(){ s=t=0; }

int64 getValue(int i,int k){ //距离大于k的不要了

while( s<t && buf[s].first<i-k ) s++;

if( s==t ) return -inf;

return buf[s].second;

}

void push(int i,int64 value){

while( s<t && buf[t-1].second<=value ) t--;

buf[t].first=i, buf[t].second=value;

t++;

}

}LQ; //\*\*\*\*\*\*\*\*\*\*\*\*模板结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## 凸队列 斜率dp

#define maxn 30005

typedef long long LL;

struct Point{

int id; LL x,y;

Point(int \_id=0,LL \_x=0,LL \_y=0):id(\_id),x(\_x),y(\_y){};

friend double slope(const Point &a,const Point &b){ //斜率

if( a.x == b.x ){

if( b.y > a.y ) return 9999999999999.0;

else return -999999999999.0;

}

return (double)(a.y-b.y) / (a.x-b.x);

}

};

struct ConvexQueue{ //维护一个下凸队列

Point que[maxn]; int s,t;

void clear(){ s=t=0; }

void push(Point p){

while( t>=s+2 ){

if( slope(que[t-2],que[t-1]) > slope(que[t-1],p) ) //这里决定下凸

t--;

else break;

}

que[t++]=p;

}

Point getFirst(){

if( s==t ) return Point(0,0,(LL)1 <<60 );//这个要保证不给力,看题目定数字

return que[s];

}

void pop(double \_slope){ //斜率不够的弹出

while(t>=s+2){

if( slope(que[s],que[s+1]) < \_slope ) s++;

else break;

}

}

void pop(int ID){ //去掉下标小于ID的

while( s<t ){

if( que[s].id < ID ) s++;

else break;

}

}

}cq;

## STL priority\_queue

struct DD{

int val,id;

DD(int \_v=0,int \_i=0):val(\_v),id(\_i){};

friend bool operator<(const DD &a,const DD &b){

return a.val < b.val;

}

friend bool operator>(const DD &a,const DD &b){ //给greater<DD>用的

return a.val > b.val;

}

};

priority\_queue<DD> pq1; //默认最大堆

priority\_queue<DD,vector<DD>,greater<DD> > pq2; //最小堆

//priority\_queue<int,vector<int>,less\_equal<int> > pq2;

/\*\* addition

equal\_to 相等

not\_equal\_to 不相等

less 小于

greater 大于

less\_equal 小于等于

greater\_equal 大于等

这些在所有的排序算法中同样适用

\*/

## 快速 queue

template<class T>

class queue{

private:

int s,t,maxLen,curLen;

T \*data;

public:

//queue():s(0),t(0),maxLen(0),curLen(0){};

queue(int size=1000000):s(0),t(0),curLen(0){

maxLen=size;

data=(T\*)malloc(maxLen\*sizeof(T));

}

T front(){ return data[s]; }

void pop(){ s++; s=(s==maxLen?0:s); curLen--; }

void push(T v){ data[t++]=v; t=(t==maxLen?0:t); curLen++; }

bool empty(){ return curLen==0; }

int size(){ return curLen; }

void clear(){ s=t=0; curLen=0; }

~queue(){ free(data); }

};

字符串

## 最长回文 O(n)

//最长回文

//The Manacher Algorithm 求一个字符串中以某个位置为中心的最大半径

//利用的要点是 \*m\*\*\*\*\*I\*\*\*\*\*c\* m是c的对称点,那么算c的半径的时候,(\*m\*) 和 (\*c\*)是对称的

#define maxn 200100

int rado[maxn]; //rado[i],长度奇数的 max k{[i-k,...,i-1, i ,i+1,...,i+k] }

int rade[maxn]; //rade[i],长度为偶数的 max k{[i-k-1,...,i-1,i, i+1,i+2,...,i+k] }

void Manacher(char \*str,int len){ //index base 0

int I=0,R=0,r=0;

for(int i=0;i<len;){

while(i-(r+1)>=0 && i+(r+1)<len && str[i-(r+1)]==str[i+(r+1)] ) r++;

rado[i]=r;

if( i+r > I+R ) I=i,R=r;

i++;

if( I+R < i ){

r=0; continue;

}else{

int mir=I-(i-I); //mir 镜像对称点

r=min(rado[mir],I+R-i);

}

}

I=0,R=0,r=0;

for(int i=0;i<len;){

while( i-r>=0 && i+1+r<len && str[i-r]==str[i+1+r] ) r++;

rade[i]=r;

if( i+r > I+R ) I=i,R=r;

i++;

if( I+R<=i ) {

r=0; continue;

}else{

int mir=I-(i-I);

r=min(rade[mir],I+R-i);

}

}

}//\*\*\*\*\*\*\*\*\*\*\*\*\*\*模板结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## kmp

#define maxn 110000

int next[maxn];

//在str中寻找pattern出现的位置(起始位置)

void kmp(char \*str,char \*pat,vector<int> &pos){ //base1

int L=strlen(str+1),Lp=strlen(pat+1);

int q=0;

next[1]=q; pos.clear();

for(int i=2;i<=Lp;i++){

while( q && pat[q+1]!=pat[i] ) q=next[q];

if( pat[q+1]==pat[i] ) q++;

next[i]=q;

}

q=0;

for(int i=1;i<=L;i++){

while( q && pat[q+1]!=str[i] ) q=next[q];

if( pat[q+1]==str[i] ) q++;

if( q==Lp ){

pos.push\_back(i-Lp+1);

q=next[q];

}

}

}

## 扩展kmp

//扩展KMP by zjut\_dd

//后缀A[i]和B[i]和A的最长公共前缀长度

#define maxn 200002

char A[maxn],B[maxn];

int AA\_lcp[maxn],BA\_lcp[maxn];//分别记录后缀A[i]和B[i]和A的最长公共前缀长度 base1

void extend\_kmp(char \*A,char \*B,int lenA,int lenB){ //base1

//先计算AA\_lcp数组

AA\_lcp[1]=lenA; //\*\*\*必须的\*\*\*\*

int j=0,k;

while(A[1+j]==A[2+j] && 2+j<=lenA) j++;//到了串未0是不可能再匹配下去的

AA\_lcp[2]=j,k=2;

for(int i=3;i<=lenA;i++){ //k+AA\_lcp[k]是不减的

int Len=k+AA\_lcp[k]-1,L=AA\_lcp[i-k+1];//Len是A[k]延伸到的最远处

if(Len-i+1 > L)

AA\_lcp[i]=L;

else{

j=max(0,Len-i+1);

while(A[1+j]==A[i+j] && i+j<=lenA) j++;

AA\_lcp[i]=j;

k=i;

}

}

//下面计算BA\_lcp数组

j=0;

while(B[1+j]==A[1+j] && 1+j<=lenA && 1+j<=lenB) j++;

BA\_lcp[1]=j,k=1;

for(int i=2;i<=lenB;i++){

int Len=k+BA\_lcp[k]-1,L=AA\_lcp[i-k+1];

if(Len-i+1 > L)

BA\_lcp[i]=L;

else{

j=max(0,Len-i+1);

while(A[1+j]==B[i+j] && 1+j<=lenA && i+j<=lenB) j++;

BA\_lcp[i]=j;

k=i;

}

}

}

## 串的最小表示 kmp求循环节

//串的最小表示 kmp求循环节 hdu 3374

#define maxn 1000005

//串的最小表示,下标从1开始,返回下标

int min\_present(char \*ch,int len){

int p1=1,p2=2,k=0;

while(p1<=len && p2<=len && k<len){ //找不到比它还小的了 或者 完全匹配上了

int n1=p1+k; if(n1>len) n1-=len;

int n2=p2+k; if(n2>len) n2-=len;

if(ch[n1] == ch[n2]) k++; // 相等的话,检测长度加1

else if(ch[n1] > ch[n2] ) p1+=k+1,k=0; //大于的话,说明p1肯定不是最小表示(必须的)

else p2+=k+1,k=0; //理由同上

if(p1==p2) p2++;

}

return min(p1,p2);

}

//串的最大表示,base1,返回下标

int max\_present(char \*ch,int len){

int p1=1,p2=2,k=0;

while(p1<=len && p2<=len && k<len){

int n1=p1+k; if(n1>len) n1-=len;

int n2=p2+k; if(n2>len) n2-=len;

if(ch[n1] == ch[n2]) k++;

else if(ch[n1] < ch[n2] ) p1+=k+1,k=0;

else p2+=k+1,k=0;

if(p1==p2) p2++;

}

return min(p1,p2);

}

int next[maxn];

int kmp(char \*ch,int len){ //找循环节长度

int q=0;

next[1]=0;

for(int i=2;i<=len;i++){

while(q!=0 && ch[q+1]!=ch[i]) q=next[q];

if(ch[q+1]==ch[i]) q++;

next[i]=q;

}

int t=len-next[len];

if(len%t) return len;

else return t;

}

char ch[maxn];

int main(){

while(gets(ch+1)>0){

int len=strlen(ch+1);

int minp=min\_present(ch,len);

int maxp=max\_present(ch,len);

int t=len/kmp(ch,len);

printf("%d %d %d %d\n",minp,t,maxp,t);

}

}

## AC自动机

//hdu2222 给你一个字典,和一个串,找这个串里包含几个单词

//广义的fail指针更快

//调用fail\_pointer后，自动机的fail和x都不可能是NULL的\*\*\*\*\*\*\*\*\*\*\*

#define CH 26 //几叉树

struct DD{

int num;

DD \*fail,\*x[CH]; //x代表next

void init(){

num=0;

memset(x,NULL,sizeof(x));

}

}dd[1000000],\*que[1000000];

struct ACAutoM{

int e;

DD \*root;

DD\* \_alloc(){ dd[e].init(); return &dd[e++]; }

void init(){

e=0; root=\_alloc();

}

void insert(char \*ch){ //插入一个单词

DD \*p=root;

for(int i=0,d;ch[i];i++){

d=ch[i]-'a';

if(p->x[d]==NULL) p->x[d]=\_alloc();

p=p->x[d];

}

p->num++; //单词个数

}

void fail\_pointer(){

root->fail=root; //

int s=0,t=0;

DD \*p;

for(int i=0;i<CH;i++){ //由于root->fail = root故需要特殊考虑

if(root->x[i]){

root->x[i]->fail=root;

que[t++]=root->x[i];

}

else

root->x[i]=root;

}

while(s<t){

p=que[s++];

for(int i=0;i<CH;i++){

if(p->x[i]){

p->x[i]->fail=p->fail->x[i];

que[t++]=p->x[i];

}

else

p->x[i]=p->fail->x[i];

}

}

}

}acm;//\*\*\*\*\*\*\*\*\*\*\*\*\*\*模板结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int work(DD\* root,char\* s){ //找起~~~~

int d,ans=0;

DD \*p=root,\*tmp;

for(int i=0;s[i];i++){

d=s[i]-'a';

tmp=p=p->x[d];

while(tmp!=root && tmp->num!=-1){ //-1表示之前已经找过了

ans+=tmp->num;

tmp->num=-1;

tmp=tmp->fail;

}

}

return ans;

}

char s[1000005];

char ch[55];

int main(){

int cas,n; scanf("%d ",&cas);

while(cas--){

acm.init();

scanf("%d ",&n);

for(int i=1;i<=n;i++)

gets(ch),acm.insert(ch);

acm.fail\_pointer();

gets(s);

printf("%d\n",work(acm.root,s));

}

}

## AC自动机dp

#define CH 4 //几叉树

struct Node{

int id,kill; //kill是一个标记

Node \*fail,\*x[CH]; //x就是next

void init(){

kill=0;

memset(x,NULL,sizeof(x));

}

}dd[100000],\*que[100000];

Node\* ptr[10005]; //ptr[i]表示第i个状态的节点地址

int cnt; //状态总数

struct ACAutoM{

int e;

Node \*root;

Node\* \_alloc(){ dd[e].init(); return &dd[e++]; }

void init(){

cnt=e=0; root=\_alloc();

root->id=cnt; ptr[cnt++]=root; //

}

void insert(char \*ch){ //插入一个单词

Node \*p=root;

for(int i=0,d;ch[i];i++){

d=ch[i]-'a';

if(p->x[d]==NULL) {

p->x[d]=\_alloc();

p->x[d]->id=cnt; ptr[cnt++]=p->x[d];

}

p=p->x[d];

}

p->kill=1;

}

void get\_fail\_point(){

int s=0,t=0;

Node \*p;

root->fail=root;

for(int i=0;i<CH;i++){ //由于root->fail = root故需要特殊考虑

if(root->x[i]){

root->x[i]->fail=root;

que[t++]=root->x[i];

}

else

root->x[i]=root;

}

while(s<t){

p=que[s++];

if( p->fail->kill ) p->kill=1; //回退的时候可能。。。

for(int i=0;i<CH;i++){

if(p->x[i]){

p->x[i]->fail=p->fail->x[i];

que[t++]=p->x[i];

}else

p->x[i]=p->fail->x[i];

}

}

}

}acm;//\*\*\*\*\*\*\*\*\*\*\*\*\*\*模板结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// acm.init(); acm.insert(...); acm.get\_fail\_point();

## 后缀数组

//用法: 1,赋值data(base1). 2,赋值n(串长). 3,make\_SA(). 4,make\_H().

char data[maxn];

//SA是后缀数组,SA[i]表示排第i位的后缀是谁

//rnk是排名数组,rnk[i]表示后缀i排第几

//H是高度数组,H[i]为SA[i]和SA[i-1]的LCP,故从2开始

int SA[maxn],rnk[maxn],H[maxn],n; //n字符串长度

#define getR(a)((a)>n?0:rnk[a])

inline bool cmp(int a,int b){ return data[a] < data[b]; }

int c[maxn],a[maxn],\_SA[maxn],\_rnk[maxn],\_h[maxn],L;

void Double(){

memset(c,0,sizeof(int) \* (n + 1));int i;

for (i=1;i<=n;i++) c[getR(i + L)]++;

a[0] = 1;for (i=1;i<=n;i++) a[i] = a[i - 1] + c[i - 1];

for (i=1;i<=n;i++) \_SA[a[getR(i + L)]++] = i;

memset(c,0,sizeof(int) \* (n + 1));

for (i=1;i<=n;i++) c[getR(i)]++;

a[0] = 1;for (i=1;i<=n;i++) a[i] = a[i - 1] + c[i - 1];

for (i=1;i<=n;i++) SA[a[getR(\_SA[i])]++] = \_SA[i];

\_rnk[SA[1]] = 1;

for (i=2;i<=n;i++)

if (getR(SA[i]) == getR(SA[i - 1]) && getR(SA[i] + L) == getR(SA[i - 1] + L))

\_rnk[SA[i]] = \_rnk[SA[i - 1]];

else

\_rnk[SA[i]] = \_rnk[SA[i - 1]] + 1;

memcpy(rnk,\_rnk,sizeof(int) \* (n + 1));

L <<= 1;

}

void make\_SA(){ //跟字符串值大小无关的!!

int i;

for (i=1;i<=n;i++) SA[i] = i;

sort(SA + 1,SA + n + 1,cmp);rnk[SA[1]] = 1;

for (i=2;i<=n;i++)

if (data[SA[i]] == data[SA[i - 1]]) rnk[SA[i]] = rnk[SA[i - 1]];

else rnk[SA[i]] = rnk[SA[i - 1]] + 1;

L = 1; while (L < n) Double();

}

void make\_H(){

int i;

memset(\_h,0,sizeof(int) \* (n + 1));

for (i=1;i<=n;i++) if (rnk[i] == 1) \_h[i] = 0;

else if (i == 1 || \_h[i - 1] <= 1)

while (data[i + \_h[i]] == data[SA[rnk[i] - 1] + \_h[i]]) \_h[i]++;

else{

\_h[i] = \_h[i - 1] - 1;

while (data[i + \_h[i]] == data[SA[rnk[i] - 1] + \_h[i]]) \_h[i]++;

}

for (i=1;i<=n;i++) H[rnk[i]] = \_h[i];

}//\*\*\*\*\*\*\*\*\*\*\*\*\*模板结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## 不同子串个数nlog(n)

++++后缀数组模板++++

int diff\_substr(char \*str){ //计算不同子串个数nlog(n), str base1

n=strlen(str+1);

memcpy(data,str,(n+1)\*sizeof(char));

make\_SA(); make\_H();

int ret=n-SA[1]+1;

for(int i=2;i<=n;i++){

int l=n-SA[i]+1;

if( l>H[i] ) ret+=l-H[i];

}

return ret;

} //diff\_substr(str);...

网络流

## 最大流dinic 邻接表

//最大流 dinic pku 3469

#define MAXN 20005

#define MAXE 600000

#define inf 0x3f3f3f3f

struct Edge{ int v,c,x; }E[MAXE]; //指向的节点, 剩余可增广的流量

int l[MAXN],e; //必须保证e的初始值是偶数

void init(){ e=0; memset(l,-1,sizeof(l)); }

inline void insert(int u,int v,int f,int invf){ //u->v=f v->u=invf

E[e].v=v; E[e].c=f; E[e].x=l[u]; l[u]=e++;

E[e].v=u; E[e].c=invf;E[e].x=l[v]; l[v]=e++;

}

struct Netflow{

int src,sink; //需要初始化

//以上

int L[MAXN],Q[MAXN]; //L=level Q=queue

int \_bfs(){ //广搜,并标记level(只取流量大于0的边)

int s=0,t=0,u;

memset(L,0xff,sizeof(L));

L[src]=0; Q[t++]=src;

while (s<t){

u=Q[s++];

for (int p=l[u];p>=0;p=E[p].x)

if (E[p].c && L[E[p].v]==-1)

L[(Q[t++]=E[p].v)]=L[u]+1;

}

return L[sink]!=-1;

}

int \_find(int u,int in){ //in:能流入u节点的最大流量. 返回u节点能流出的最大流量

if (u==sink) return in;

int t,w=0; //w表示已经从u流出的总流量

for (int p=l[u];p>=0 && w<in;p=E[p].x){

if (E[p].c>0 && L[E[p].v]==L[u]+1){

if (t=\_find(E[p].v,min(E[p].c,in-w))){

E[ p].c-=t;

E[p^1].c+=t; //这里表示必须

w+=t; //多路增广优势巨大

}

}

}

if( w<in )L[u]=-1;//关键的一句话

return w;

}

int dinic(){

int t,res=0;

while (\_bfs())while (t=\_find(src,inf))res+=t;

return res;

}

}flow; //\*\*\*\*\*\*\*\*模板结束\*\*\*\*\*\*\*\*\*\*\*

//init(); insert(...); flow.src=.. ; flow.dinic();

## 最大流dinic 邻接阵

//最大流dinic 邻接矩阵

#define maxn 500

#define inf 0x3f3f3f3f

int n,adj[maxn][maxn],src,sink; //总节点数n(base0), 流网络adj, 源s汇t

int L[maxn],Q[maxn]; //L=level Q=queue

int \_bfs(){ //广搜,并标记level(只取流量大于0的边)

int s=0,t=0,u;

memset(L,0xff,sizeof(L));

L[src]=0; Q[t++]=src;

while (s<t){

u=Q[s++];

for (int v=0;v<n;v++)

if (adj[u][v] && L[v]==-1)

L[ Q[t++]=v ]=L[u]+1;

}

return L[sink]!=-1;

}

int \_find(int u,int in){ //in:能流入u节点的最大流量. 返回u节点能流出的最大流量

if (u==sink) return in;

int t,w=0; //w表示已经从u流出的总流量

for (int v=0;v<n && w<in;v++){

if (adj[u][v] && L[v]==L[u]+1){

if (t=\_find(v,min(adj[u][v],in-w))){

adj[u][v]-=t;

adj[v][u]+=t;

w+=t;

}

}

}

if( w<in )L[u]=-1;//关键的一句话

return w;

}

int dinic(){

int t,res=0;

while (\_bfs())while (t=\_find(src,inf))res+=t;

return res;

}//n,adj,src,sink赋值之后,调用dinic().....

int main(){

int m,cas,Te=1; cin>>cas;

while( cas-- && cin>>n>>m ){

src=0,sink=n-1;

memset(adj,0,sizeof(adj));

for(int i=0;i<m;i++){

int a,b,val; cin>>a>>b>>val;

a--,b--;

adj[a][b]+=val;

}

printf("Case %d: %d\n",Te++,dinic());

}

}

## 最大流ek 改流量

//最大流算法ek,邻接阵, 单路增广,复杂度O(VE^2)

#define maxn 222

#define inf 0x3f3f3f3f

int n,adj[maxn][maxn],s,t; //总节点数n(base0), 流网络adj, 源s汇t

int pre[maxn]; //bfs之后每个点的前驱节点,pre[s]=-1;

int vst[maxn]; //最后S={vst[i]=1}|{vst[i]=0}=T组成割,且这个算法保证|S|最小

int bfs(){

fill(pre,pre+n,-1);

fill(vst,vst+n,0 );

queue<int>q; q.push(s); vst[s]=1;

while ( !q.empty() ) {

int u=q.front(); q.pop();

for (int i=0;i<=n;i++){

if( adj[u][i]>0 && !vst[i] ){

vst[i]=1; pre[i]=u;

q.push(i);

if( i==t ) return 1;

}

}

}

return 0;

}

int edmonds\_karp(){

int f=0;

while (bfs()){

int mi=inf;

for (int i=t;pre[i]!=-1;i=pre[i]) {

mi=min(mi,adj[pre[i]][i]);

}

f+=mi;

for (int i=t;pre[i]!=-1;i=pre[i]){

adj[pre[i]][i]-=mi;

adj[i][pre[i]]+=mi;

}

}

return f;

}//n=...; clear(adj); adj[i][j]=...; ans=edmonds\_karp();

//修改流量的网络流(关键在于寻找增广路(单路的)和对流量的控制...)

//UVA 4957 题意:一个R\*C的01阵满足每行的和为R[i],每列的和为C[j],求一个字典序最小的阵

int main(){

int R,C,Te=0;

while(cin>>R>>C && (R||C) ){

int s1=0,s2=0,a;

n=R+C+2; s=0; t=n-1;

memset(adj,0,sizeof(adj));

for(int i=1;i<=R;i++){

cin>>a;

s1+=a; adj[s][i]=a;

}

for(int j=1;j<=C;j++){

cin>>a;

s2+=a; adj[R+j][t]=a;

}

for(int i=1;i<=R;i++) for(int j=1;j<=C;j++) adj[i][R+j]=1;

if( Te++ ) puts("");

if( s1!=s2 || edmonds\_karp() != s1 ){

puts("Impossible"); continue;

}

for(int i=1;i<=R;i++){//逐格确定

for(int j=1;j<=C;j++){

if( adj[i][R+j]==1 ){

adj[i][R+j]=0; //不允许再被修改

putchar('N'); continue;

}

s=i; t=R+j;

if( bfs() ){

putchar('N');

for(int k=t;k!=s;k=pre[k]){

adj[pre[k]][k]-=1;

adj[k][pre[k]]+=1;

}

adj[R+j][i]=0; //不允许再被修改

}else{

putchar('Y');

adj[R+j][i]=0; //不允许再被修改

}

}

puts("");

}

}

}

## 无向图最小割

//pku 2914

#include <iostream>

using namespace std;

int mat[600][600];

int Mincut(int n) { //无向图最小割 base 0

int res=1<<29, node[600], dist[600];

bool visit[600];

int i, prev, maxj, j, k;

for (i = 0; i < n; i++) //保存顶点 ，固定顶点为0

node[i] = i;

while (n > 1) {

maxj = 1;

for (i = 1; i < n; i++) {

dist[node[i]] = mat[node[0]][node[i]]; //初始化距离数组dist[]

if (dist[node[i]] > dist[node[maxj]]) //寻找最大距离——求最大生成树

maxj = i;

}

prev = 0;

memset(visit, false, sizeof (visit));

visit[node[0]] = true;

//求最大生成树，并进行最小割操作。

for (i = 1; i < n; i++) {

if (i == n - 1) { //只剩最后一个没加入集合的点，更新最小割

res = \_\_min(res, dist[node[maxj]]);

for (k = 0; k < n; k++) //合并最后一个点以及推出它的集合中的点

mat[node[k]][node[prev]] = (mat[node[prev]][node[k]] += mat[node[k]][node[maxj]]);

node[maxj] = node[--n]; //缩点后的图

}

visit[node[maxj]] = true; //标记已经访问

prev = maxj;

maxj = -1;

for (j = 1; j < n; j++){

if (!visit[node[j]]) { //将上次求的maxj加入集合，合并与它相邻的边到割集

dist[node[j]] += mat[node[prev]][node[j]]; //dist[]保存的是一个积累量。

if (maxj == -1 || dist[node[maxj]] < dist[node[j]])

maxj = j; //保存更新值

}

}

}

}

return res;

}

int main()

{

int n, m, a, b, v;

while (scanf("%d%d", &n, &m) != EOF) {

memset(mat, 0, sizeof (mat));

while (m--) {

scanf("%d%d%d", &a, &b, &v);

mat[a][b] += v;

mat[b][a] += v;

}

printf("%d\n", Mincut(n));

}

return 0;

}

## 无源汇上下界可行流

//无源汇可行流 输出方案 sgu194

//增加源汇,对于(u->v,b,c) 增加(src->v,b),(u->sink,b)跑最大流使满流

#define maxn 205

#define inf 0x3f3f3f3f

int n,adj[maxn][maxn],src,sink; //总节点数n(base0), 流网络adj, 源s汇t

int L[maxn],Q[maxn]; //L=level Q=queue

int \_bfs(){ //广搜,并标记level(只取流量大于0的边)

int s=0,t=0,u;

memset(L,0xff,sizeof(L));

L[src]=0; Q[t++]=src;

while (s<t){

u=Q[s++];

for (int v=0;v<n;v++)

if (adj[u][v] && L[v]==-1)

L[ Q[t++]=v ]=L[u]+1;

}

return L[sink]!=-1;

}

int \_find(int u,int in){ //in:能流入u节点的最大流量. 返回u节点能流出的最大流量

if (u==sink) return in;

int t,w=0; //w表示已经从u流出的总流量

for (int v=0;v<n && w<in;v++){

if (adj[u][v] && L[v]==L[u]+1){

if (t=\_find(v,min(adj[u][v],in-w))){

adj[u][v]-=t;

adj[v][u]+=t;

w+=t;

}

}

}

if( w<in )L[u]=-1;//关键的一句话

return w;

}

int dinic(){

int t,res=0;

while (\_bfs())while (t=\_find(src,inf))res+=t;

return res;

}//n,adj,src,sink赋值之后,调用dinic().....

struct Edge{

int u,v,b,c;// b<=f<=c 上下界

}e[40004];

int main(){

int m;

while( scanf("%d%d",&n,&m)!=EOF ){

n+=2; src=0; sink=n-1;

memset(adj,0,sizeof(adj));

int sf=0;

for(int i=0;i<m;i++){

int u,v,b,c;

scanf("%d %d %d %d",&u,&v,&b,&c);

adj[u][sink]+=b;

adj[src ][v]+=b;

adj[u][v]+=c-b; //保证b<=c??

sf+=b;

e[i].u=u; e[i].v=v; e[i].b=b; e[i].c=c;

}

if( dinic() != sf ){

puts("NO");

}else{

puts("YES");

for(int i=0;i<m;i++){

int u=e[i].u, v=e[i].v;

printf("%d\n",e[i].c-adj[u][v]);//无重边

}

}

}

}

## 有源汇上下界可行流

/\*有源汇可行流 输出方案 pku2396

prob: 一个N\*M的矩阵,给出每行每列的和,每个格子的上下限制,求一种可行方案

s->t的有源汇可行流, t->s连无穷大的边,转化为无源汇可行流

\*/

#define maxn 255

#define inf 0x3f3f3f3f

int n,adj[maxn][maxn],src,sink; //总节点数n(base0), 流网络adj, 源s汇t

int L[maxn],Q[maxn];

int \_bfs(){

int s=0,t=0,u;

memset(L,0xff,sizeof(L));

L[src]=0; Q[t++]=src;

while (s<t){

u=Q[s++];

for (int v=0;v<n;v++)

if (adj[u][v] && L[v]==-1)

L[ Q[t++]=v ]=L[u]+1;

}

return L[sink]!=-1;

}

int \_find(int u,int in){

if (u==sink) return in;

int t,w=0;

for (int v=0;v<n && w<in;v++){

if (adj[u][v] && L[v]==L[u]+1){

if (t=\_find(v,min(adj[u][v],in-w))){

adj[u][v]-=t;

adj[v][u]+=t;

w+=t;

}

}

}

if( w<in )L[u]=-1;

return w;

}

int dinic(){

int t,res=0;

while (\_bfs())while (t=\_find(src,inf))res+=t;

return res;

}//n,adj,src,sink赋值之后,调用dinic().....

struct Edge{

int u,v,b,c;// b<=f<=c 上下界

}e[400004];

//计算无源汇上下界可行流, 传入无源汇边集, cnt base0, 选取的src和sink, 总节点数n base0

bool no\_src\_feasible\_flow(Edge \*e,int cnt,int s,int t,int n){

::n=n; src=s; sink=t;

memset(adj,0,sizeof(adj));

int sf=0;

for(int i=0;i<cnt;i++){

int u=e[i].u, v=e[i].v, b=e[i].b, c=e[i].c;

if( b>c || c<0 ) return false; //无解

adj[u][sink]+=b;

adj[src ][v]+=b;

adj[u][v]+=c-b;

sf+=b;

}

return dinic()==sf;

}//----over----

int l[222][22],h[222][22];//确定上下界

void read\_constraint(int N,int M){

int con; scanf("%d",&con);

while( con-- ){

int r,c,val; char op;

scanf(" %d %d %c %d",&r,&c,&op,&val);

if( op=='<' ) val--;

else if(op=='>') val++;

for(int i=(r==0?1:r);i<=(r==0?N:r);i++){

for(int j=(c==0?1:c);j<=(c==0?M:c);j++){

if( op=='='){

l[i][j]=max(l[i][j],val);

h[i][j]=min(h[i][j],val);

}else if( op=='<' ){

h[i][j]=min(h[i][j],val);

}else if( op=='>' ){

l[i][j]=max(l[i][j],val);

}else{

assert(false);

}

}

}

}

}

int main(){

int N,M,cas; scanf("%d",&cas);

while( cas-- ){

scanf("%d%d",&N,&M);

int cnt=0,ts=N+M+1,tt=N+M+2;

for(int i=1;i<=N;i++) {//每行的和

scanf("%d",&e[cnt].b);

e[cnt].u=ts; e[cnt].v=i;

e[cnt].c=e[cnt].b; cnt++;

}

for(int j=1;j<=M;j++){ //每列的和

scanf("%d",&e[cnt].b);

e[cnt].u=N+j; e[cnt].v=tt;

e[cnt].c=e[cnt].b; cnt++;

}

for(int i=1;i<=N;i++) for(int j=1;j<=M;j++) l[i][j]=0,h[i][j]=inf;//每个格子的上下限

read\_constraint(N,M); //读取限制条件

for(int i=1;i<=N;i++){

for(int j=1;j<=M;j++){

e[cnt].u=i; e[cnt].v=N+j; e[cnt].b=l[i][j]; e[cnt].c=h[i][j]; cnt++;

}

}

e[cnt].u=tt; e[cnt].v=ts;

e[cnt].b=0; e[cnt].c=inf; cnt++; //从tt引一条无穷大的边到ts,绝对不能少!!

if( no\_src\_feasible\_flow(e,cnt,0,tt+1,tt+2)==false ){

puts("IMPOSSIBLE");

}else{

for(int i=1;i<=N;i++)

for(int j=1;j<=M;j++)

printf("%d%s",h[i][j]-=adj[i][N+j],j==M ? "\n" : " ");

}

puts("");

}

}

## 有源汇上下界最大流 邻接阵

/\*有源汇上下界最大流 zju3229

solu:把必要流分离,集中到新的原汇点上,对于新的原汇点求一次最大流,判断是否有解.

再对于原来的原汇点求一次最大流就可以了. 第二种方法是二分,跟上下界最小流一样.

\*/

#define maxn 1400

#define inf 0x3f3f3f3f

int n,adj[maxn][maxn],src,sink; //总节点数n(base0), 流网络adj, 源s汇t

int L[maxn],Q[maxn]; //L=level Q=queue

int \_bfs(){ //广搜,并标记level(只取流量大于0的边)

int s=0,t=0,u;

memset(L,0xff,sizeof(L));

L[src]=0; Q[t++]=src;

while (s<t){

u=Q[s++];

for (int v=0;v<n;v++)

if (adj[u][v] && L[v]==-1)

L[ Q[t++]=v ]=L[u]+1;

}

return L[sink]!=-1;

}

int \_find(int u,int in){ //in:能流入u节点的最大流量. 返回u节点能流出的最大流量

if (u==sink) return in;

int t,w=0; //w表示已经从u流出的总流量

for (int v=0;v<n && w<in;v++){

if (adj[u][v] && L[v]==L[u]+1){

if (t=\_find(v,min(adj[u][v],in-w))){

adj[u][v]-=t;

adj[v][u]+=t;

w+=t;

}

}

}

if( w<in )L[u]=-1;//关键的一句话

return w;

}

int dinic(){

int t,res=0;

while (\_bfs())while (t=\_find(src,inf))res+=t;

return res;

}//n,adj,src,sink赋值之后,调用dinic().....

struct Edge{

int u,v,b,c;// b<=f<=c 上下界

}e[400004];

//插入 u->v b<=f<=c 新增源汇为ns,nt

void ins\_l(int u,int v,int b,int c,int ns,int nt){

adj[u][nt]+=b;

adj[ns][v]+=b;

adj[u][v]+=c-b;

}

int main(){

int N,M;

while( scanf("%d %d",&N,&M)!=EOF ){

//os,ot是建模的时候添加的, ns,nt是求解的时候添加的

int os=N+M+1,ot=N+M+2, ns=0, nt=N+M+3; //old new

n=nt+1; src=ns; sink=nt;

int sum=0;

for(int i=0;i<n;i++) fill(adj[i],adj[i]+n,0);

for(int j=N+1;j<=N+M;j++){

int g; scanf("%d",&g);// (j->ot) >=g

ins\_l(j,ot,g,inf,ns,nt);

sum+=g;

}

int id=0;

for(int i=1;i<=N;i++){

int C,D; scanf("%d %d",&C,&D); //(os->i) <=D

adj[os][i]+=D;

for(int k=0;k<C;k++){

int T,l,r; scanf("%d %d %d",&T,&l,&r);

T+=N+1; //(i->T) <l,r>

e[id].u=i; e[id].v=T; e[id].b=l; e[id++].c=r;

ins\_l(i,T,l,r,ns,nt);

sum+=l;

}

}

adj[ot][os]+=inf;

if( dinic()!=sum ){

puts("-1\n"); continue; // 无解

}

//adj[ot][os]=adj[os][ot]=0; //由于流量是最后统计的(不会统计os->ot边),所以这些东西都不用删了

//for(int i=0;i<n;i++) adj[ns][i]=adj[nt][i]=0;

src=os; sink=ot;

dinic();

sum=0;

for(int i=0;i<id;i++){

e[i].c-=adj[ e[i].u ][ e[i].v ];

sum+=e[i].c;

}

printf("%d\n",sum);

for(int i=0;i<id;i++) printf("%d\n",e[i].c);

puts("");

}

}

## 有源汇上下界最大流 邻接表

/\*有源汇上下界最大流 zju3229

solu:把必要流分离,集中到新的原汇点上,对于新的原汇点求一次最大流,判断是否有解.

再对于原来的原汇点求一次最大流就可以了.

\*/

#define MAXN 1400

#define MAXE 1000000

#define inf 0x3f3f3f3f

struct Edge{ int v,c,x; }E[MAXE]; //指向的节点, 剩余可增广的流量

int l[MAXN],e; //必须保证e的初始值是偶数

void init(){ e=0; memset(l,-1,sizeof(l)); }

inline void insert(int u,int v,int f,int invf){ //u->v=f v->u=invf

E[e].v=v; E[e].c=f; E[e].x=l[u]; l[u]=e++;

E[e].v=u; E[e].c=invf;E[e].x=l[v]; l[v]=e++;

}

struct Netflow{

int src,sink; //需要初始化

//以上

int L[MAXN],Q[MAXN];

int \_bfs(){

int s=0,t=0,u;

memset(L,0xff,sizeof(L));

L[src]=0; Q[t++]=src;

while (s<t){

u=Q[s++];

for (int p=l[u];p>=0;p=E[p].x)

if (E[p].c && L[E[p].v]==-1)

L[(Q[t++]=E[p].v)]=L[u]+1;

}

return L[sink]!=-1;

}

int \_find(int u,int in){

if (u==sink) return in;

int t,w=0;

for (int p=l[u];p>=0 && w<in;p=E[p].x){

if (E[p].c>0 && L[E[p].v]==L[u]+1){

if (t=\_find(E[p].v,min(E[p].c,in-w))){

E[ p].c-=t;

E[p^1].c+=t;

w+=t;

}

}

}

if( w<in )L[u]=-1;

return w;

}

int dinic(){

int t,res=0;

while (\_bfs())while (t=\_find(src,inf))res+=t;

return res;

}

}flow; //\*\*\*\*\*\*\*\*模板结束\*\*\*\*\*\*\*\*\*\*\*

//init(); insert(...); flow.src=.. ; flow.dinic();

//插入 u->v b<=f<=c 新增源汇为ns,nt

void ins\_l(int u,int v,int b,int c,int ns,int nt){

insert(u,nt,b,0);

insert(ns,v,b,0);

insert(u,v,c-b,0);

}

int f[1400][1400];

int main(){

int N,M;

while( scanf("%d %d",&N,&M)!=EOF ){

//os,ot是建模的时候添加的, ns,nt是求解的时候添加的

int os=0,ot=N+M+1, ns=ot+1, nt=ns+1; //old new

int sum=0;

init();

for(int i=0;i<=nt;i++) fill(f[i],f[i]+nt+1,-1);

for(int j=N+1;j<=N+M;j++){

int g; scanf("%d",&g);// (j->ot) >=g

ins\_l(j,ot,g,inf,ns,nt);

sum+=g;

}

int id=0;

for(int i=1;i<=N;i++){

int C,D; scanf("%d %d",&C,&D); //(os->i) <=D

ins\_l(os,i,0,D,ns,nt);

for(int k=0;k<C;k++){

int T,l,r; scanf("%d %d %d",&T,&l,&r);

T+=N+1; //(i->T) <l,r>

f[i][T]=r;

ins\_l(i,T,l,r,ns,nt);

sum+=l;

}

}

ins\_l(ot,os,0,inf,ns,nt); //转化

flow.src=ns; flow.sink=nt;

if( flow.dinic()!=sum ){ //判断可行流是否存在

puts("-1\n"); continue; //无解

}

//adj[ot][os]=adj[os][ot]=0; //由于流量是最后统计的(不会统计os->ot边),所以这些东西都不用删了

//for(int i=0;i<n;i++) adj[ns][i]=adj[nt][i]=0;

flow.src=os; flow.sink=ot;

flow.dinic();

sum=0; //统计流量

for(int i=1;i<=N;i++){

for(int p=l[i];p>=0;p=E[p].x){

int v=E[p].v;

if( !(v>N && v<=N+M ) ) continue;

f[i][v]-=E[p].c;

sum+=f[i][v];

}

}

printf("%d\n",sum); //输出每条边的容量

for(int i=1;i<=N;i++)

for(int j=N+1;j<=N+M;j++) if( f[i][j]>=0 ) printf("%d\n",f[i][j]);

puts("");

}

}

## 有源汇上下界最小流

//有源汇上下界最小流 sgu176

//solu: 二分答案mid, t->s的流量为mid,看是否有无源汇可行流

#define maxn 140

#define inf 0x3f3f3f3f

int n,adj[maxn][maxn],src,sink; //总节点数n(base0), 流网络adj, 源s汇t

int L[maxn],Q[maxn]; //L=level Q=queue

int \_bfs(){ //广搜,并标记level(只取流量大于0的边)

int s=0,t=0,u;

memset(L,0xff,sizeof(L));

L[src]=0; Q[t++]=src;

while (s<t){

u=Q[s++];

for (int v=0;v<n;v++)

if (adj[u][v] && L[v]==-1)

L[ Q[t++]=v ]=L[u]+1;

}

return L[sink]!=-1;

}

int \_find(int u,int in){ //in:能流入u节点的最大流量. 返回u节点能流出的最大流量

if (u==sink) return in;

int t,w=0; //w表示已经从u流出的总流量

for (int v=0;v<n && w<in;v++){

if (adj[u][v] && L[v]==L[u]+1){

if (t=\_find(v,min(adj[u][v],in-w))){

adj[u][v]-=t;

adj[v][u]+=t;

w+=t;

}

}

}

if( w<in )L[u]=-1;//关键的一句话

return w;

}

int dinic(){

int t,res=0;

while (\_bfs())while (t=\_find(src,inf))res+=t;

return res;

}//n,adj,src,sink赋值之后,调用dinic().....

struct Edge{

int u,v,b,c;// b<=f<=c 上下界

}e[40004];

//计算无源汇上下界可行流, 传入无源汇边集, cnt base0, 选取的src和sink, 总节点数n base0

bool no\_src\_feasible\_flow(Edge \*e,int cnt,int ns,int nt,int n){

::n=n; src=ns; sink=nt;

for(int i=0;i<n;i++) fill(adj[i],adj[i]+n,0);

int sf=0;

for(int i=0;i<cnt;i++){

int u=e[i].u, v=e[i].v, b=e[i].b, c=e[i].c;

if( b>c || c<0 ) return false; //无解

adj[u][sink]+=b;

adj[src ][v]+=b;

adj[u][v]+=c-b;

sf+=b;

}

return dinic()==sf;

}//----over----

int main(){

int N,E;

while( scanf("%d %d",&N,&E)!=EOF ){

//os,ot是建模的时候的, ns,nt是求解的时候添加的

int os=1,ot=N, ns=0, nt=N+1; //old new

for(int j=0;j<E;j++){

int u,v,b,s;

scanf("%d %d %d %d",&u,&v,&b,&s);

assert(u\*v>0);

e[j].u=u; e[j].v=v; e[j].c=b;

if( s==0 ){ //不必满流

e[j].b=0;

}else{ //必须满流

e[j].b=b;

}

}

e[E].u=ot; e[E].v=os; e[E].b=0;

int l=0,r=100000005;

e[E].c=r;

if( no\_src\_feasible\_flow(e,E+1,ns,nt,nt+1)==false ){

puts("Impossible"); continue; // 无解

}

e[E].c=0;

if( no\_src\_feasible\_flow(e,E+1,ns,nt,nt+1)==true ) {

l=r=0;

goto output;

}

while( l+1<r ){

int mid=(l+r)/2;

e[E].c=mid;

if( no\_src\_feasible\_flow(e,E+1,ns,nt,nt+1) ) r=mid;

else l=mid;

}

output:;

e[E].c=r;

no\_src\_feasible\_flow(e,E+1,ns,nt,nt+1);

printf("%d\n",r);

for(int i=0;i<E;i++) printf("%d%s",e[i].c-adj[e[i].u][e[i].v],(i==E-1?"\n":" ") );

}

}

## 最小费用最大流

//通过了 pku2135 pku3422 hdu3667 hdu 3376

//最小费用最大流 dd实现

#define maxn 10005

#define maxm 300005

#define inf 0x3fffffff

struct Edge{ int u,v,cap,cost,x; }E[maxm];

int e,l[maxn];

inline void init(){ e=0; memset(l,-1,sizeof(l)); }

inline void insert(int u,int v,int cap,int cost){ //单向边

E[e].u=u; E[e].v=v; E[e].cap=cap; E[e].cost= cost; E[e].x=l[u]; l[u]=e++;

E[e].u=v; E[e].v=u; E[e].cap=0 ; E[e].cost=-cost; E[e].x=l[v]; l[v]=e++;

}

int q[20\*maxn],s,t, inq[maxn], dis[maxn],eid[maxn];

void min\_cost\_max\_flow(int src,int sink,int &cap,int &cost){//0~n-1

cost=cap=0;

while( true ){ //不断增广

//begin spfa

for(int i=0;i<maxn;i++) dis[i]=inf; //-inf 最大费

s=t=0; q[t++]=src; inq[src]=1; dis[src]=0;

while(s<t){

int u=q[s++]; inq[u]=0;

for(int p=l[u];p>=0;p=E[p].x){

if( E[p].cap<=0 ) continue;

int v=E[p].v;

if( dis[v]>dis[u]+E[p].cost ){ //<

dis[v]=dis[u]+E[p].cost; eid[v]=p;

if( inq[v]==0 ){

inq[v]=1; q[t++]=v;

}

}

}

} //end spfa

if( dis[sink]>=inf ) return; //<=-inf

int c=inf;

for(int i=sink;i!=src;i=E[eid[i]].u) c=min(c,E[eid[i]].cap);

cost+=dis[sink]\*c; cap+=c;

for(int i=sink;i!=src;i=E[eid[i]].u) {

int p=eid[i]; E[p].cap-=c; E[p^1].cap+=c;

}

}

}//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*模板结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int main(){

int n,m,u,v,cap,cost;

while(scanf("%d%d",&n,&m)!=EOF ){

init(); //初始化

for(int i=1;i<=m;i++){

scanf("%d %d %d %d",&u,&v,&cap,&cost);

insert(u,v,cap,cost);

}

min\_cost\_max\_flow(0,n-1,cap,cost);

printf("capicity: %d cost: %d\n",cap,cost);

}

}

## 总结

最大流：DINIC or SAP

最小费用最大流：SPFA+增广(费用的值较离散) or ZKW(费用的值集中)

有源汇的上下界最大流：新建s', t',用(i, j, l, r)表示i到j有一条下界为l上界为r的边，将每条这样的边拆成(s', j, 0, l), (i, t', 0, l), (i, j, 0, r-l)，加入边(t, s, 0, max)再从s‘到t'求最大流，再去掉(t, s, 0, max)这条边，从s到t求最大流

有源汇的上下界最小可行流：基本同上，将最后一步改成从t到s反求一遍最大流来退流；也可以二分(t, s, 0, max)这条边的容量

有源汇的上下界最小费用可行流：拆边方法同上，从s'向t'求一遍最小费用最大流

有源汇的上下界最小费用最大流：基本同上，还要再s向t求一遍最小费用最大流(\*这个是自己YY的，可能不对，求指教)

无源汇的最大流(无向图全局最小割)：Stoer-Wagner算法

无源汇的所有点对间最大流：分治，在当前点集内随便选两个点求最小割，用这个割更新一遍所有跨在两边的点对(不一定只是当前点集内的点)，再将自己的点集割成两部分，递归做

无源汇的上下界可行流：拆边，直接从s'到t'跑一遍最大流

无源汇的上下界最小费用可行流：拆边，直接从s'到t'跑一遍最小费用最大流

平面图最小割转最短路：将平面区域当成点，两个点之间的边权为原来这两个平面区域之间的边的容量，补上一条汇到源的正无穷边之后，求这条正无穷边的一边到另一边的最短路

搜索算法

## dlx精确覆盖

//pku 3740

#define MAXR 16\*16\*16//row

#define MAXC 16\*16\*4//column

#define TOTAL (MAXR+1)\*(MAXC+1)

using namespace std;

int mat[MAXR][MAXC];//初始矩阵，0或1，从1，1开始用

int cid[TOTAL],rid[TOTAL],s[MAXC];

int l[TOTAL],r[TOTAL],u[TOTAL],d[TOTAL];

int lastc[MAXC];//第i列的最下面一个元素

void remove(const int &col){

l[r[col]]=l[col];

r[l[col]]=r[col];

for(int i=d[col];i!=col;i=d[i]){ //枚举每一行

for(int j=r[i];j!=i;j=r[j]){ //枚举每一列

d[u[j]]=d[j];

u[d[j]]=u[j];

s[ cid[j] ]--;

}

}

}

void resume(const int &col){ //回溯

for(int i=u[col];i!=col;i=u[i]){

for(int j=l[i];j!=i;j=l[j]){

d[u[j]]=j;

u[d[j]]=j;

s[ cid[j] ]++;

}

}

l[r[col]]=col;

r[l[col]]=col;

}

inline void addlr(const int &ln,const int &rn){ //完了后还是成环的

l[ r[ln] ]=rn; r[rn]=r[ln];

r[ln]=rn; l[rn]=ln;

}

inline void addud(const int &un,const int &dn){

u[ d[un] ]=dn; d[dn]=d[un];

d[un]=dn; u[dn]=un;

}

int res[MAXR],ind;//每次取的哪行

bool search(int k){ //将开始第几层搜索

if(r[0]==0){

ind=k;

return true;//找到一组解

}

int c,mi=100000000;

for(int i=r[0];i!=0;i=r[i]){

if(s[i]<=mi){ //诡异，要<=,列的顺序是行->列->宫->格子，估计是后面的宫的那个限制条件卡的紧一些，所有先选后面的会好些

mi=s[i]; c=i;

}

}

if(mi==0) return false; //可以不要，包括在下面这个for里的

remove(c);

for(int i=d[c];i!=c;i=d[i]){

res[k]=rid[i];//选择了哪一行，这个数组可以用来输出选择了那些东东

for(int j=r[i];j!=i;j=r[j]) remove(cid[j]);

if(search(k+1)) return true;

for(int j=l[i];j!=i;j=l[j]) resume(cid[j]);

}

resume(c);

return false;

}

/\* 链表的形状类似这样

#######

. . ..

... .

. ..

\*/

void adapt(int n,int m){ //传入01矩阵mat的规格,适配函数（由mat构造dancing link 表）

memset(s,0,sizeof(s));//每列元素个数清0

l[0]=r[0]=0;

for(int j=1;j<=m;j++){

addlr(j-1,j);

u[j]=d[j]=lastc[j]=j;

}

int c=m+1;//从m+1开始用

for(int i=1;i<=n;i++){

bool f=true;

for(int j=1;j<=m;j++){

if(mat[i][j]){

if(f){

f=false;

l[c]=r[c]=c;

}else{

addlr(c-1,c);

}

addud(lastc[j],c);

rid[c]=i; cid[c]=j;

s[j]++; lastc[j]=c++;

}

}

}

}//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*模板结束\*\*\*\*\*\*\*\*\*\*\*

int main(){

int n,m;

while(scanf("%d%d",&n,&m)!=EOF){

for(int i=1;i<=n;i++)

for(int j=1;j<=m;j++)

scanf("%d",mat[i]+j);

adapt(n,m);

if(search(1))

cout<<"Yes, I found it\n";

else

cout<<"It is impossible\n";

}

}

## dlx重复覆盖

//重复覆盖问题 NUAA 1507

#include<iostream>

#define MAXR 62//row

#define MAXC 62//column

#define TOTAL (MAXR+1)\*(MAXC+1)

using namespace std;

int n,m,mat[MAXR][MAXC];//初始矩阵，0或1，从1，1开始用

int cid[TOTAL],rid[TOTAL],s[MAXC];//显示行号，列号，每列的总个数

int res[MAXR];//每次取的哪行

int l[TOTAL],r[TOTAL],u[TOTAL],d[TOTAL];

int lastc[MAXC];//第i列的最下面一个元素

void remove(const int &col){//这一整列没了，包括列头\*\*\*\*\*\*\*\*\*\*

for(int i=d[col];i!=col;i=d[i]){

r[l[i]]=r[i];

l[r[i]]=l[i];

}

}

void backtrack(const int &col){

for(int i=u[col];i!=col;i=u[i]){

r[l[i]]=i;

l[r[i]]=i;

}

}

inline void addlr(const int &ln,const int &rn){//完了后还是成环的

l[ r[ln] ]=rn; r[rn]=r[ln];

r[ln]=rn; l[rn]=ln;

}

inline void addud(const int &un,const int &dn){

u[ d[un] ]=dn; d[dn]=d[un];

d[un]=dn; u[dn]=un;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bool visit[MAXC];

int H(){//估价函数，表示要重复覆盖全部还至少需要几行

int res=0;

memset(visit,false,sizeof(visit));

for(int c=r[0];c!=0;c=r[c]){

if(!visit[c]){

res++;

visit[c]=true;

for(int i=d[c];i!=c;i=d[i])

for(int j=r[i];j!=i;j=r[j])

visit[cid[j]]=true;

}

}

return res;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int mi;

void search(int k,int sum){//将开始第几层搜索,k是输出答案数组的下表,sum是当前已经多少层了

if(sum+H()>=mi)

return ;

if(r[0]==0){

mi=min(mi,sum);

return;//找到一组解

}

int c,mi=1000000;

for(int i=r[0];i!=0;i=r[i]){

if(s[i]<mi){

mi=s[i];

c=i;

}

}

for(int i=d[c];i!=c;i=d[i]){

res[k]=rid[i];

remove(i);

for(int j=r[i];j!=i;j=r[j])

remove(j);

search(k+1,sum+1);

for(int j=l[i];j!=i;j=l[j])

backtrack(j);

backtrack(i);

}

}

/\*

这个就是DLX的建模模型，0都是为了服务中间的\*的,左上角的是（0,0）

00000000

\*\*\*\*\*\*\*

\*\*\*\*\*\*\*

\*\*\*\*\*\*\*

\*\*\*\*\*\*\*

\*/

void adapt(int n,int m){//传入01矩阵mat的规格,适配函数（由mat构造dancing link 表）

memset(s,0,sizeof(s));//每列元素个数清0

l[0]=r[0]=0; //总头子是(0,0)\*\*\*\*\*\*\*\*\*\*\*

for(int i=1;i<=m;i++){ //总共有m列，总编号跟列号是一样的\*\*\*\*

addlr(i-1,i);

u[i]=d[i]=i;//自成循环

lastc[i]=i; //最下面一个元素号码

}

int ind=m+1;//从m+1开始用

for(int i=1;i<=n;i++){

bool isfirst=true;

for(int j=1;j<=m;j++){

if(mat[i][j]){

if(isfirst){

isfirst=false;

l[ind]=r[ind]=ind;

}else{

addlr(ind-1,ind);

}

addud(lastc[j],ind); lastc[j]=ind;

cid[ind]=j; rid[ind]=i;

s[j]++; ind++;

}

}

}

}

int main(){

int n,m;

while(scanf("%d%d",&n,&m)!=EOF){

int x,k;

for(int i=1;i<=n;i++){

scanf("%d",&x);

for(int j=1;j<=x;j++)

scanf("%d",&k),mat[i][k]=1;

}

adapt(n,m);

mi=m;

search(1,0);

printf("%d\n",mi);

memset(mat,0,sizeof(mat));

}

}

## A\*算法伪代码

Best\_First\_Search()

{

　Open = [起始节点];

　Closed = [];

　while (Open表非空)

　{

　　从Open中取得一个节点X，并从OPEN表中删除。

　　if (X是目标节点)

　　{

　　　求得路径PATH；

　　　返回路径PATH；

　　}

　　for (每一个X的子节点Y)

　　{

　　　if (Y不在OPEN表和CLOSE表中)

　　　{

　　　　求Y的估价值；

　　　　并将Y插入OPEN表中；

　　　}

　　　//还没有排序

　　　else if (Y在OPEN表中)

　　　{

　　　　if (Y的估价值小于OPEN表的估价值)

　　　　　更新OPEN表中的估价值；

　　　}

　　　else //Y在CLOSE表中

　　　{

　　　　if (Y的估价值小于CLOSE表的估价值)

　　　　{

　　　　　更新CLOSE表中的估价值；

　　　　　从CLOSE表中移出节点，并放入OPEN表中；

　　　　}

　　　}

　　　将X节点插入CLOSE表中；

　　　按照估价值将OPEN表中的节点排序；

　　}//end for

　}//end while

}//end func

计算几何

几何模板非本人所写,基本摘自ZJU

## 基础几何模板

//浮点几何函数库 from ZJU

#include <math.h>

#define eps 1e-8

#define zero(x) (((x)>0?(x):-(x))<eps)

struct point{double x,y;};

struct line{point a,b;};

//计算cross product (P1-P0)x(P2-P0)

double xmult(point p1,point p2,point p0){

return (p1.x-p0.x)\*(p2.y-p0.y)-(p2.x-p0.x)\*(p1.y-p0.y);

}

double xmult(double x1,double y1,double x2,double y2,double x0,double y0){

return (x1-x0)\*(y2-y0)-(x2-x0)\*(y1-y0);

}

//计算dot product (P1-P0).(P2-P0)

double dmult(point p1,point p2,point p0){

return (p1.x-p0.x)\*(p2.x-p0.x)+(p1.y-p0.y)\*(p2.y-p0.y);

}

double dmult(double x1,double y1,double x2,double y2,double x0,double y0){

return (x1-x0)\*(x2-x0)+(y1-y0)\*(y2-y0);

}

//两点距离

double distance(point p1,point p2){

return sqrt((p1.x-p2.x)\*(p1.x-p2.x)+(p1.y-p2.y)\*(p1.y-p2.y));

}

double distance(double x1,double y1,double x2,double y2){

return sqrt((x1-x2)\*(x1-x2)+(y1-y2)\*(y1-y2));

}

//判三点共线

int dots\_inline(point p1,point p2,point p3){

return zero(xmult(p1,p2,p3));

}

int dots\_inline(double x1,double y1,double x2,double y2,double x3,double y3){

return zero(xmult(x1,y1,x2,y2,x3,y3));

}

//判点是否在线段上,包括端点

int dot\_online\_in(point p,line l){

return zero(xmult(p,l.a,l.b))&&(l.a.x-p.x)\*(l.b.x-p.x)<eps&&(l.a.y-p.y)\*(l.b.y-p.y)<eps;

}

int dot\_online\_in(point p,point l1,point l2){

return zero(xmult(p,l1,l2))&&(l1.x-p.x)\*(l2.x-p.x)<eps&&(l1.y-p.y)\*(l2.y-p.y)<eps;

}

int dot\_online\_in(double x,double y,double x1,double y1,double x2,double y2){

return zero(xmult(x,y,x1,y1,x2,y2))&&(x1-x)\*(x2-x)<eps&&(y1-y)\*(y2-y)<eps;

}

//判点是否在线段上,不包括端点

int dot\_online\_ex(point p,line l){

return dot\_online\_in(p,l)&&(!zero(p.x-l.a.x)||!zero(p.y-l.a.y))&&(!zero(p.x-l.b.x)||!zero(p.y-l.b.y));

}

int dot\_online\_ex(point p,point l1,point l2){

return dot\_online\_in(p,l1,l2)&&(!zero(p.x-l1.x)||!zero(p.y-l1.y))&&(!zero(p.x-l2.x)||!zero(p.y-l2.y));

}

int dot\_online\_ex(double x,double y,double x1,double y1,double x2,double y2){

return dot\_online\_in(x,y,x1,y1,x2,y2)&&(!zero(x-x1)||!zero(y-y1))&&(!zero(x-x2)||!zero(y-y2));

}

//判两点在线段同侧,点在线段上返回0

int same\_side(point p1,point p2,line l){

return xmult(l.a,p1,l.b)\*xmult(l.a,p2,l.b)>eps;

}

int same\_side(point p1,point p2,point l1,point l2){

return xmult(l1,p1,l2)\*xmult(l1,p2,l2)>eps;

}

//判两点在线段异侧,点在线段上返回0

int opposite\_side(point p1,point p2,line l){

return xmult(l.a,p1,l.b)\*xmult(l.a,p2,l.b)<-eps;

}

int opposite\_side(point p1,point p2,point l1,point l2){

return xmult(l1,p1,l2)\*xmult(l1,p2,l2)<-eps;

}

//判两直线平行

int parallel(line u,line v){

return zero((u.a.x-u.b.x)\*(v.a.y-v.b.y)-(v.a.x-v.b.x)\*(u.a.y-u.b.y));

}

int parallel(point u1,point u2,point v1,point v2){

return zero((u1.x-u2.x)\*(v1.y-v2.y)-(v1.x-v2.x)\*(u1.y-u2.y));

}

//判两直线垂直

int perpendicular(line u,line v){

return zero((u.a.x-u.b.x)\*(v.a.x-v.b.x)+(u.a.y-u.b.y)\*(v.a.y-v.b.y));

}

int perpendicular(point u1,point u2,point v1,point v2){

return zero((u1.x-u2.x)\*(v1.x-v2.x)+(u1.y-u2.y)\*(v1.y-v2.y));

}

//判两线段相交,包括端点和部分重合

int intersect\_in(line u,line v){

if (!dots\_inline(u.a,u.b,v.a)||!dots\_inline(u.a,u.b,v.b))

return !same\_side(u.a,u.b,v)&&!same\_side(v.a,v.b,u);

return dot\_online\_in(u.a,v)||dot\_online\_in(u.b,v)||dot\_online\_in(v.a,u)||dot\_online\_in(v.b,u);

}

int intersect\_in(point u1,point u2,point v1,point v2){

if (!dots\_inline(u1,u2,v1)||!dots\_inline(u1,u2,v2))

return !same\_side(u1,u2,v1,v2)&&!same\_side(v1,v2,u1,u2);

return dot\_online\_in(u1,v1,v2)||dot\_online\_in(u2,v1,v2)||dot\_online\_in(v1,u1,u2)||dot\_online\_in(v2,u1,u2);

}

//判两线段相交,不包括端点和部分重合

int intersect\_ex(line u,line v){

return opposite\_side(u.a,u.b,v)&&opposite\_side(v.a,v.b,u);

}

int intersect\_ex(point u1,point u2,point v1,point v2){

return opposite\_side(u1,u2,v1,v2)&&opposite\_side(v1,v2,u1,u2);

}

//计算两直线交点,注意事先判断直线是否平行!

//线段交点请另外判线段相交(同时还是要判断是否平行!)

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;

}

point intersection(point u1,point u2,point v1,point v2){

point ret=u1;

double t=((u1.x-v1.x)\*(v1.y-v2.y)-(u1.y-v1.y)\*(v1.x-v2.x))

/((u1.x-u2.x)\*(v1.y-v2.y)-(u1.y-u2.y)\*(v1.x-v2.x));

ret.x+=(u2.x-u1.x)\*t;

ret.y+=(u2.y-u1.y)\*t;

return ret;

}

//点到直线上的最近点

point ptoline(point p,line l){

point t=p;

t.x+=l.a.y-l.b.y,t.y+=l.b.x-l.a.x;

return intersection(p,t,l.a,l.b);

}

point ptoline(point p,point l1,point l2){

point t=p;

t.x+=l1.y-l2.y,t.y+=l2.x-l1.x;

return intersection(p,t,l1,l2);

}

//点到直线距离

double disptoline(point p,line l){

return fabs(xmult(p,l.a,l.b))/distance(l.a,l.b);

}

double disptoline(point p,point l1,point l2){

return fabs(xmult(p,l1,l2))/distance(l1,l2);

}

double disptoline(double x,double y,double x1,double y1,double x2,double y2){

return fabs(xmult(x,y,x1,y1,x2,y2))/distance(x1,y1,x2,y2);

}

//点到线段上的最近点

point ptoseg(point p,line l){

point t=p;

t.x+=l.a.y-l.b.y,t.y+=l.b.x-l.a.x;

if (xmult(l.a,t,p)\*xmult(l.b,t,p)>eps)

return distance(p,l.a)<distance(p,l.b)?l.a:l.b;

return intersection(p,t,l.a,l.b);

}

point ptoseg(point p,point l1,point l2){

point t=p;

t.x+=l1.y-l2.y,t.y+=l2.x-l1.x;

if (xmult(l1,t,p)\*xmult(l2,t,p)>eps)

return distance(p,l1)<distance(p,l2)?l1:l2;

return intersection(p,t,l1,l2);

}

//点到线段距离

double disptoseg(point p,line l){

point t=p;

t.x+=l.a.y-l.b.y,t.y+=l.b.x-l.a.x;

if (xmult(l.a,t,p)\*xmult(l.b,t,p)>eps)

return distance(p,l.a)<distance(p,l.b)?distance(p,l.a):distance(p,l.b);

return fabs(xmult(p,l.a,l.b))/distance(l.a,l.b);

}

double disptoseg(point p,point l1,point l2){

point t=p;

t.x+=l1.y-l2.y,t.y+=l2.x-l1.x;

if (xmult(l1,t,p)\*xmult(l2,t,p)>eps)

return distance(p,l1)<distance(p,l2)?distance(p,l1):distance(p,l2);

return fabs(xmult(p,l1,l2))/distance(l1,l2);

}

//矢量V以P为顶点逆时针旋转angle并放大scale倍

point rotate(point v,point p,double angle,double scale){

point ret=p;

v.x-=p.x,v.y-=p.y;

p.x=scale\*cos(angle);

p.y=scale\*sin(angle);

ret.x+=v.x\*p.x-v.y\*p.y;

ret.y+=v.x\*p.y+v.y\*p.x;

return ret;

}

//以下 added by zjut\_DD

//矢量V逆时针旋转rad(弧度)

P rotate(P v,double rad){

P ret; ret.x=ret.y=0;

double px=cos(rad);

double py=sin(rad);

ret.x+=v.x\*px-v.y\*py;

ret.y+=v.x\*py+v.y\*px;

return ret;

}

## 凸包

// CONVEX HULL III by VB

//构造凸包函数,传入原始点集大小n,点集p(p原有顺序被打乱!)

//返回凸包大小,凸包的点在ch中

//参数maxsize为1包含共线点,为0不包含共线点,缺省为0

//返回的凸包为逆时针顺序

//保留共线点时重点会引起算法不稳定，而所有点在一条直线上时中间点会重复出现

//不保留共线点时没有以上问题

#include<stdlib.h>

#define eps 1e-8

#define zero(x) (((x)>0?(x):-(x))<eps)

struct point{double x,y;};

double xmult(point p1,point p2,point p0){

return (p1.x-p0.x)\*(p2.y-p0.y)-(p2.x-p0.x)\*(p1.y-p0.y);

}

int cmp(const void \*a,const void \*b){

point \*p1=(point \*)a,\*p2=(point \*)b;

return zero(p1->y-p2->y)?(p1->x>p2->x+eps?1:-1):(p1->y>p2->y+eps?1:-1);

}

int graham(int n,point p[],point ch[],int maxsize=0)

{

const double e1=maxsize?eps:-eps;

int i,j,k;

if (n<3){

for(i=0;i<n;i++) ch[i]=p[i];

return n;

}

qsort(p,n,sizeof(p[0]),cmp);

for(ch[0]=p[0],ch[1]=p[1],i=j=2;i<n;ch[j++]=p[i++])

while(j>1 && xmult(ch[j-2],p[i],ch[j-1])>e1) j--;

for(ch[k=j++]=p[n-2],i=n-3;i>0;ch[j++]=p[i--])

while(j>k && xmult(ch[j-2],p[i],ch[j-1])>e1) j--;

while(j>k && xmult(ch[j-2],ch[0],ch[j-1])>e1) j--;

return j;

}

## 多边形重心

point barycenter(int n,point\* p){

point ret,t;

double t1=0,t2;

int i;

ret.x=ret.y=0;

for (i=1;i<n-1;i++)

if (fabs(t2=xmult(p[0],p[i],p[i+1]))>eps){

t=barycenter(p[0],p[i],p[i+1]);

ret.x+=t.x\*t2;

ret.y+=t.y\*t2;

t1+=t2;

}

if (fabs(t1)>eps)

ret.x/=t1,ret.y/=t1;

return ret;

}

## 多边形面积

//计算多边形面积,顶点按顺时针或逆时针给出

double area\_polygon(int n,point\* p){

double s1=0,s2=0;

int i;

for (i=0;i<n;i++)

s1+=p[(i+1)%n].y\*p[i].x,s2+=p[(i+1)%n].y\*p[(i+2)%n].x;

return fabs(s1-s2)/2;

}

## 半平面交

//半平面交,O(n^2) from watashi's blog

const int MAXN = 100;

const double EPS = 1e-8;

struct Point {

double x, y;

Point() { }

Point(double x, double y) : x(x), y(y) { }

};

inline bool zero(double x) {

return ((x > 0 ? x : -x) < EPS);

}

inline double xmult(const Point & p1, const Point & p2, const Point & p0) {

return (p1.x - p0.x) \* (p2.y - p0.y) - (p2.x - p0.x) \* (p1.y - p0.y);

}

bool sameSide(const Point & p1, const Point & p2, const Point & l1, const Point & l2) {

return xmult(l1, p1, l2) \* xmult(l1, p2, l2) > EPS;

}

Point intersection(const Point & u1, const Point & u2, const Point & v1, const Point & v2) {

Point ret = u1;

double t = ((u1.x - v1.x) \* (v1.y - v2.y) - (u1.y - v1.y) \* (v1.x - v2.x)) / ((u1.x - u2.x) \* (v1.y - v2.y) - (u1.y - u2.y) \* (v1.x - v2.x));

ret.x += (u2.x - u1.x) \* t;

ret.y += (u2.y - u1.y) \* t;

return ret;

}

//点的个数0~n-1,逆时针的点集p, 插入的直线上的两个点l1,l2,和判断是要哪一边的side点

void polygonCut(int & n, Point \* p, const Point & l1, const Point & l2, const Point & side) {

Point pp[MAXN];

int m = 0, i;

for (i = 0; i < n; i++) {

if (sameSide(p[i], side, l1, l2)) {

pp[m++] = p[i];

}

if (!sameSide(p[i], p[(i + 1) % n], l1, l2) && !(zero(xmult(p[i], l1, l2)) && zero(xmult(p[(i + 1) %n ], l1, l2)))) {

pp[m++] = intersection(p[i], p[(i + 1) % n], l1, l2);

}

}

for (n = i = 0; i < m; i++) {

if (!i || !zero(pp[i].x - pp[i - 1].x) || !zero(pp[i].y - pp[i-1].y)) {

p[n++] = pp[i];

}

}

if (zero(p[n - 1].x - p[0].x) && zero(p[n - 1].y - p[0].y)) {

n--;

}

if (n < 3) {

n = 0;

}

}

//获得半平面交的面积,0~n-1

double areaPolygon(int n, const Point\* p) {

double s1 = 0, s2 = 0;

for (int i = 0; i < n; i++) {

s1 += p[(i + 1) % n].y \* p[i].x;

s2 += p[(i + 1) % n].y \* p[(i + 2) % n].x;

}

return fabs(s1 - s2)/2;

}

//用法: p[0],p[1],p[2],p[3]初始化,设置整体的外边框. 然后调用polygonCut...

## 三角形和圆的面积交

//pku 2986 三角形和圆的面积交

struct point{ double x,y; };

double outer(point a,point b,point c){

return (a.x-c.x)\*(b.y-c.y)-(a.y-c.y)\*(b.x-c.x);

}

double inner(point a,point b,point c){

return (a.x-c.x)\*(b.x-c.x)+(a.y-c.y)\*(b.y-c.y);

}

double dist(point a,point b){

return sqrt((a.x-b.x)\*(a.x-b.x)+(a.y-b.y)\*(a.y-b.y));

}

double calcarea(point p1,point p2,point o,double r){

double A,B,C,x,y,tS;

A=dist(p2,o);

B=dist(p1,o);

C=dist(p2,p1);

if(A<r&&B<r)

return outer(p1,p2,o)/2;

else if(A<r&&B>=r){

x=(inner(p1,o,p2)+sqrt(r\*r\*C\*C-outer(p1,o,p2)\*outer(p1,o,p2)))/C;

tS=outer(p1,p2,o)/2;

return asin(tS\*(1-x/C)\*2/r/B)\*r\*r/2+tS\*x/C;

}else if(A>=r&&B<r){

y=(inner(p2,o,p1)+sqrt(r\*r\*C\*C-outer(p2,o,p1)\*outer(p2,o,p1)))/C;

tS=outer(p1,p2,o)/2;

return asin(tS\*(1-y/C)\*2/r/A)\*r\*r/2+tS\*y/C;

}else if(fabs(outer(p1,p2,o))>=r\*C||inner(p2,o,p1)<=0||inner(p1,o,p2)<=0)

if(inner(p1,p2,o)<0)

if(outer(p1,p2,o)<0)

return (-acos(-1.0)-asin(outer(p1,p2,o)/A/B))\*r\*r/2;

else

return (acos(-1.0)-asin(outer(p1,p2,o)/A/B))\*r\*r/2;

else

return asin(outer(p1,p2,o)/A/B)\*r\*r/2;

else{

x=(inner(p1,o,p2)+sqrt(r\*r\*C\*C-outer(p1,o,p2)\*outer(p1,o,p2)))/C;

y=(inner(p2,o,p1)+sqrt(r\*r\*C\*C-outer(p2,o,p1)\*outer(p2,o,p1)))/C;

tS=outer(p1,p2,o)/2;

return (asin(tS\*(1-x/C)\*2/r/B)+asin(tS\*(1-y/C)\*2/r/A))\*r\*r/2+tS\*((y+x)/C-1);

}

}

double tri\_cir\_area(P p0,P p1,P p2,P o,double r){ //计算三角形和圆的面积交

double sum=0;

sum+=calcarea(p0,p1,o,r);

sum+=calcarea(p1,p2,o,r);

sum+=calcarea(p2,p0,o,r);

return fabs(sum);

}//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int main(){

P p0,p1,p2,o;

double r;

while(cin>>p0.x>>p0.y>>p1.x>>p1.y>>p2.x>>p2.y>>o.x>>o.y>>r){

printf("%.2lf\n",tri\_cir\_area(p0,p1,p2,o,r));

}

}

## 最近点对 二维

//分治法

#include<iostream>

#include<algorithm>

#include<cmath>

using namespace std;

#define maxn 100009

struct point{

double x,y;

}p[maxn];

int index\_y[maxn];//存的是按Y排序后的下标

inline double dis(point &a,point &b){

return sqrt((a.x-b.x)\*(a.x-b.x)+(a.y-b.y)\*(a.y-b.y));

}

inline bool cmpx(const point &a,const point &b){

return a.x<b.x;

}

inline bool cmpy(int a,int b){

return p[a].y<p[b].y;

}

inline double Min(double a,double b){

return a<b?a:b;

}

double find(int l,int r){

if(l+1==r)

return dis(p[l],p[r]);

if(l+2==r)

return Min(dis(p[l],p[l+1]),Min(dis(p[l],p[l+2]),dis(p[l+1],p[l+2])));

int mid=(l+r)>>1;

double Mindis=Min(find(l,mid),find(mid+1,r));

int ind=0;

for(int i=mid;i>=l && p[mid+1].x-p[i].x<=Mindis;i--)

index\_y[ind++]=i;

for(int i=mid+1;i<=r && p[i].x-p[mid].x<=Mindis;i++)

index\_y[ind++]=i;

sort(index\_y,index\_y+ind,cmpy);

for(int i=0;i<ind;i++){

for(int j=i+1;j<ind && p[ index\_y[j] ].y-p[ index\_y[i] ].y<=Mindis;j++)

Mindis=Min(Mindis,dis(p[ index\_y[j] ],p[ index\_y[i] ]));

}

return Mindis;

}

int main(){

int N;

while(scanf("%d",&N)!=EOF&&N){

for(int i=0;i<N;i++)

scanf("%lf%lf",&p[i].x,&p[i].y);

sort(p,p+N,cmpx);//这个必须在前

double ans=find(0,N-1);

printf("%.2lf\n",ans/2.0);

}

}

## 最近点对 三维

//分治法

#define maxn 100009

struct point{

double x,y,z;

}p[maxn];

int index\_y[maxn];//存的是按Y排序后的下标

inline double dis(point a,point b){

return sqrt((a.x-b.x)\*(a.x-b.x)+(a.y-b.y)\*(a.y-b.y)+(a.z-b.z)\*(a.z-b.z));

}

inline bool cmpy(int a,int b){

return p[a].y<p[b].y;

}

inline bool cmpz(point &a,point &b){

if(a.z!=b.z)

return a.z<b.z;

return a.y!=b.y?a.y<b.y:a.x<b.x;

}

inline double Min(double a,double b){

return a<b?a:b;

}

double find(int l,int r){

if(l+1==r)

return dis(p[l],p[r]);

if(l+2==r)

return Min(dis(p[l],p[l+1]),Min(dis(p[l],p[l+2]),dis(p[l+1],p[l+2])));

int mid=(l+r)>>1;

double Mindis=Min(find(l,mid),find(mid+1,r));

int ind=0;

for(int i=mid;i>=l && p[mid+1].z-p[i].z<=Mindis;i--)

index\_y[ind++]=i;

for(int i=mid+1;i<=r && p[i].z-p[mid].z<=Mindis;i++)

index\_y[ind++]=i;

sort(index\_y,index\_y+ind,cmpy);

for(int i=0;i<ind;i++){

for(int j=i+1;j<ind && p[ index\_y[j] ].y-p[ index\_y[i] ].y<=Mindis;j++)

Mindis=Min(Mindis,dis(p[ index\_y[j] ],p[ index\_y[i] ]));

}

return Mindis;

}

int main(){

int N;

while(scanf("%d",&N)!=EOF&&N){

for(int i=0;i<N;i++)

scanf("%lf%lf%lf",&p[i].x,&p[i].y,&p[i].z);

sort(p,p+N,cmpz);//这个必须在前

double ans=find(0,N-1);

printf("%.2lf\n",ans);

}

}

匹配

## 一些结论

//定理：二分图最大独立集=结点数-最优匹配（一般图的最大独立集是NP完全问题）

//证明：易知最大独立集的补集必定覆盖了所有的边，为使最大独立集包含的点的个数最多，

//我们要求的补集就成了最小覆盖，二分图的最小覆盖=最大匹配。

//故二分图的最大独立集=结点数-最大匹配。

//zju 1137 最大独立集

//路径覆盖就是在dag图中找一些路经，使之覆盖了图中的所有顶点，且任何一个顶点有且只有一条路径与之关联(建图要分点的）

//对于公式：最小路径覆盖＝｜Ｐ｜－最大匹配数；可以这么来理解；

//如果匹配数为零，那么Ｐ中不存在有向边，于是显然有：

//最小路径覆盖＝｜Ｐ｜－最大匹配数＝｜Ｐ｜－０＝｜Ｐ｜；即Ｐ的最小路径覆盖数为｜Ｐ｜；

//如果在Ｐ＇中增加一条匹配边pi'－－＞pj''，那么在图P的路径覆盖中就存在一条由pi连接pj的边，

//也就是说pi与pj 在一条路径上，于是路径覆盖数就可以减少一个；

//pku 3216 最小路径覆盖

## 二分图最大匹配 邻接阵

//\*\*\*\*\*\*\*\*\*\*\*\*二分图最大匹配,邻接矩阵\*\*\*\*\*\*\*\*

#define maxn 555

int nL,nR; //nL->nR base1

int match[maxn]; //1~m

bool vst[maxn];

bool mat[maxn][maxn];

void init(){

memset(match,-1,sizeof(match));

memset(mat,false,sizeof(mat));

}

bool dfs(int u){ //匈牙利递归

for(int v=1;v<=nR;v++){

if(mat[u][v] && !vst[v]) {//vst保证只搜一次

vst[v]=true;

int tmp=match[v];

match[v]=u;

if(tmp==-1 || dfs(tmp))

return true;

match[v]=tmp;

}

}

return false;

}

int max\_match(){

int ret=0;

memset(match,-1,sizeof(match));

for(int i=1;i<=nL;i++){

memset(vst,false,sizeof(vst));

if( dfs(i) ) ret++;

}

return ret;

}//\*\*\*\*\*\*\*\*\*\*\*以上模板\*\*\*\*\*\*\*\*\*\*

## 二分图最大匹配 邻接表

#define maxn 1100

struct Edge{int v,x; }E[1000000];

int l[maxn],e;

void init(){ memset(l,-1,sizeof(l)); e=0; }

void insert(int a,int b){ //单向边

E[e].v=b; E[e].x=l[a]; l[a]=e++;

}

int pre[maxn], vst[maxn]; //pre是 右边->zuobian , vst标记的是右边的点

int dfs(int i){ //匈牙利递归,每次递归都要先clear vst数组

for(int p=l[i];p!=-1;p=E[p].x){

int v=E[p].v;

if( !vst[v] ) {//这里可以保证不会回头的\*\*\*\*\*\*\*\*\*\*\*\*

vst[v]=1;//这个不要回溯，因为这次不行，下次一样不行

int tmp=pre[v];

pre[v]=i;

if(tmp==-1 || dfs(tmp)) return 1;

pre[v]=tmp;

}

}

return 0;

}

int max\_match(int n,int m){//base0

memset(pre,-1,sizeof(pre));

int ret=0;

for(int i=0;i<n;i++){

fill(vst,vst+m,0);

ret+=dfs(i);

}

return ret;

}// init(); insert(...); max\_match();

## 二分图最大匹配HK

//Hopcroft Karp算法在增加匹配集合M时,每次寻找多条增广路,复杂度O(V^0.5\*E)

const int MAXN = 50005;

const int MAXM = 150005;

int nx,ny,m,x,y,ans;

int edge,head[MAXN],nxt[MAXM],e[MAXM];

int cx[MAXN], cy[MAXN];// cx[i]表示xi对应的匹配,cy[i]表示yi对应的匹配.

int distx[MAXN], disty[MAXN]; // 层的概念,即在BFS中的第几层.

int que[MAXN];

void init() {

memset(cx,-1,sizeof(cx));

memset(cy,-1,sizeof(cy));

memset(head,-1,sizeof(head));

edge = 0;

ans = 0;

}

inline void addedge(int u,int v) {

e[edge] = v;

nxt[edge] = head[u];

head[u] = edge++;

}

bool BFS() {

bool flag = 0;

memset(distx,0,sizeof(distx));

memset(disty,0,sizeof(disty));

int l = 0,r = 0,h,t;

for (int i = 1;i <= nx;i++)

if (cx[i] == -1)

que[r++] = i;

while (l != r) {

for (h = l,t = r;h != t;h = (h + 1) % MAXN) {

int u = que[h];

for (int i = head[u];i != -1;i = nxt[i]) {

int v = e[i];

if (!disty[v]) {

disty[v] = distx[u] + 1;

if (cy[v] == -1) flag = 1;

else {

distx[cy[v]] = disty[v] + 1;

que[r] = cy[v];

r = (r + 1) % MAXN;

}

}

}

}

l = t;

}

return flag;

}

bool DFS(int u) { //递归层数为sqrt(n)

for (int i = head[u];i != -1;i = nxt[i]) {

int v = e[i];

if (disty[v] == distx[u] + 1) { //说明v是u的后继结点.

disty[v] = 0; // v被用过了,不能再作为其他点的后继结点了.

if (cy[v] == -1 || DFS(cy[v])) {

cx[u] = v, cy[v] = u;

return 1;

}

}

}

return 0;

}

void Hopcroft\_Karp() {

while (BFS())

for (int i = 1;i <= nx;i++)

if (cx[i] == -1 && DFS(i))

ans++;

}

int main() {

while (scanf("%d %d %d",&nx,&ny,&m) != EOF) {

init();

for (int i = 0;i < m;i++) {

scanf("%d%d",&x,&y);

addedge(x,y);

}

Hopcroft\_Karp();

printf("%d\n",ans);

}

}

## 二分图最大权值匹配

//二分图最大权值匹配,KM算法 O(m^3) (在n<m的时候比zju模板慢)

//modified by zjut\_DD

typedef int type;

const int MAX = 330;

const type inf = 1<<30;

int n,m,match[MAX]; // n<=m match:Y->X

type w[MAX][MAX]; // 权值

type lx[MAX],ly[MAX]; // 标号

bool sx[MAX],sy[MAX]; // 是否被搜索过

bool dfs(int u){

sx [u] = true;

for(int v=0;v<m;v++){

if(!sy[v] && lx[u]+ly[v]==w[u][v]){

sy[v]=true;

if(match[v]==-1 || dfs(match[v])){

match[v]=u;

return true;

}

}

}

return false;

}

type bestmatch() {//二分图最大权匹配,权值改成负的最小权匹配

if( n<m ){ //左边点数少于右边,补成m\*m的

for(int i=n;i<m;i++) for(int j=0;j<m;j++) w[i][j]=0;

}

for(int i=0;i<m;i++){

lx[i]=-inf; ly[i]=0;

for(int j=0;j<m;j++){ //任意时刻,两定标和>=边权

if (lx[i]<w[i][j]) lx[i]=w[i][j];

}

}

memset(match,-1,sizeof(match));

for(int u=0;u<m;u++){

while (1) {

memset(sx,0,sizeof(sx));

memset(sy,0,sizeof(sy));

if (dfs (u) ) break; //能增广了

type dx = inf; // 修改标号

for(int i=0;i<m;i++){

if (!sx[i]) continue;

for(int j=0;j<m;j++){

if(sy[j]) continue;

dx=min(lx[i]+ly[j]-w[i][j],dx);

}

}

for(int i=0;i<m;i++) if(sx[i]) lx[i]-=dx;

for(int j=0;j<m;j++) if(sy[j]) ly[j]+=dx;

}

}

type sum = 0;

for(int j=0;j<m;j++) sum+=w[ match[j] ][j];

return sum;

}//\*用法,初始化n,m,w[][], 调用bestmatch

int main(){

while( scanf("%d %d",&n,&m)!=EOF ){

for(int i=0;i<n;i++) for(int j=0;j<m;j++) scanf("%d",w[i]+j);

printf("%d\n",bestmatch());

}

}

## 一般图匹配

//author: momodi@whuacm

//ural 1099

//一般图匹配,带花树算法

//初始化需要 n的大小(base0),adj矩阵

#include<iostream>

using namespace std;

#define maxn 1000

int n;//节点个数,用的是0~n-1,

int match[maxn];//每个点匹配情况

bool adj[maxn][maxn];//邻接阵,连通为true

void clear() {

memset(adj, 0, sizeof(adj));

n = 0;

}

void insert(const int &u, const int &v) {

adj[u][v] = adj[v][u] = 1;

}

int Q[maxn], pre[maxn], base[maxn];

bool \_h[maxn];

bool in\_blossom[maxn];

void argument(int u) {

while (u != -1) {

int v = pre[u];

int k = match[v];

match[u] = v;

match[v] = u;

u = k;

}

}

void change\_blossom(int b, int u) {

while (base[u] != b) {

int v = match[u];

in\_blossom[base[v]] = in\_blossom[base[u]] = true;

u = pre[v];

if (base[u] != b) {

pre[u] = v;

}

}

}

int find\_base(int u, int v) {

bool in\_path[maxn] = {};

while (true) {

in\_path[u] = true;

if (match[u] == -1) {

break;

}

u = base[pre[match[u]]];

}

while (!in\_path[v]) {

v = base[pre[match[v]]];

}

return v;

}

int contract(int u, int v) {

memset(in\_blossom, 0, sizeof(in\_blossom));

int b = find\_base(base[u], base[v]);

change\_blossom(b, u);

change\_blossom(b, v);

if (base[u] != b) {

pre[u] = v;

}

if (base[v] != b) {

pre[v] = u;

}

return b;

}

int bfs(int p) {

memset(pre, -1, sizeof(pre));

memset(\_h, 0, sizeof(\_h));

for (int i = 0; i < n; ++i) {

base[i] = i;

}

Q[0] = p; \_h[p] = 1;

for (int s = 0, t = 1; s < t; ++s) {

int u = Q[s];

for (int v = 0; v < n; ++v) {

if (adj[u][v] && base[u] != base[v] && v != match[u]){

if (v == p || (match[v] != -1 && pre[match[v]] != -1)){

int b = contract(u, v);

for (int i = 0; i < n; ++i) {

if (in\_blossom[base[i]]) {

base[i] = b;

if (\_h[i] == 0){

\_h[i] = 1;

Q[t++] = i;

}

}

}

} else if (pre[v] == -1) {

pre[v] = u;

if (match[v] == -1) {

argument(v);

return 1;

}

else{

Q[t++] = match[v];

\_h[match[v]] = 1;

}

}

}

}

}

return 0;

}

int max\_match(){ //节点编号从0~n-1

memset(match, -1, sizeof(match));//没匹配的就是-1

int ans = 0;

for (int i = 0; i < n; ++i) {

if (match[i] == -1) {

ans += bfs(i);

}

}

return ans;

}//\*\*\*\*\*\*\*\*\*\*\*\*\*\*模板结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int main(){

int N,a,b;

scanf("%d",&N);

n=N;

memset(adj,false,sizeof(adj));

while(scanf("%d%d",&a,&b)!=EOF){

adj[a-1][b-1]=adj[b-1][a-1]=true;

}

int ret=max\_match();

printf("%d\n",ret\*2);

for(int i=0;i<N;i++){

if( match[i]!=-1){

printf("%d %d\n",i+1,match[i]+1);

match[ match[i] ]=-1;

}

}

}

随机算法

## 费马点 模拟退火

实现模拟退火的一些细节

1,每次随机一个方向可能不够平均

2,平均分好几个方向又太死,可能没有我们想要的方向

3,首先随机一个基,然后平均L次 for(int j=0;j<L;j++) base+2\*PI/L \* j

4,wa了之后,测试选那种极端的数据,比如是三角形的话,选那种很扁很扁的那种 a+b几乎等于c的那种,多次运行看结果

//求多边形费马点 模拟退火

const double PI=acos(-1.0);

int N=4; //N边形

const double eps=1e-5; //精度要求

const double alf=0.95; //冷却系数

const int S=3; //并行解的个数

const int L=3; //调整尝试次数

const int F=10; //360度 等分数

const double A=2\*PI/F; //每次累加弧度

const int BS=10000; //随机化池的大小

double one[BS][F][2]; //随机化单位向量池

struct P{

double x,y;

P(double \_x=0,double \_y=0):x(\_x),y(\_y){};

double dis(const P &b)const{

return sqrt( (x-b.x)\*(x-b.x) + (y-b.y)\*(y-b.y) );

}

}p[5555];

double all\_dis(P a){

double ret=0;

for(int i=0;i<N;i++) ret+=a.dis(p[i]);

return ret;

}

inline double get\_double(){

return ( ( (int)rand()<<15 )|rand() )/(double)((1<<30)-1); //vs c++

}

void initOne(){ //初始化等分单位向量

for(int i=0;i<BS;i++){

double bas=get\_double()\*A;

for(int j=0;j<F;j++){

one[i][j][0]=cos(bas+j\*A);

one[i][j][1]=sin(bas+j\*A);

}

}

}

int main(){

freopen("in.txt","r",stdin);

freopen("out7.txt","w",stdout);

srand(time(NULL));

initOne();

while( scanf("%d",&N)!=EOF ){

for(int i=0;i<N;i++) scanf("%lf %lf",&p[i].x,&p[i].y);

if( p[1].x<0 ) break;

P res[S];

double d[S];

double T=2000;

for(int i=0;i<S;i++){ //随机初始解集

int tx,ty,sx=0,sy=0;

res[i].x=res[i].y=0;

for(int k=0;k<N;k++){

tx=rand()%1000; sx+=tx;

ty=rand()%1000; sy+=ty;

res[i].x+=p[k].x\*tx;

res[i].y+=p[k].y\*ty;

}

res[i].x/=sx;

res[i].y/=sy;

//printf("%.3lf %.3lf\n",res[i].x,res[i].y);

d[i]=all\_dis(res[i]);

}

while( T>eps ){

for(int i=0;i<S;i++){

P cur=res[i];

for(int j=0;j<L;j++){ //尝试调整

int ind=rand()%BS;

for(int k=0;k<F;k++){

P tmp;

tmp.x=cur.x+T\*one[ind][k][0];

tmp.y=cur.y+T\*one[ind][k][1];

double D=all\_dis(tmp);

if( D<d[i] ){

res[i]=tmp; d[i]=D; //break;

}

}

}

}

T\*=alf; //冷却

}

int id=0;

for(int i=0;i<S;i++){

if( d[i]<d[id] ) id=i;

}

printf("(%.3lf,%.3lf) %.4lf\n",res[id].x,res[id].y,d[id]);

}

}

## 费马点 双重三分

//求多边形费马点 双重三分 (求解精度1e-4无压力)

const double PI=acos(-1.0);

const double eps=1e-7;

const double alf=0.45; //<0.5 !!! 0.49就接近二分了

int N=4; //N边形

struct P{

double x,y;

P(double \_x=0,double \_y=0):x(\_x),y(\_y){};

double dis(const P &b)const{

return sqrt( (x-b.x)\*(x-b.x) + (y-b.y)\*(y-b.y) );

}

}p[5555];

double all\_dis(P a){

double ret=0;

for(int i=0;i<N;i++) ret+=a.dis(p[i]);

return ret;

}

double miX,maX,miY,maY;//三分的范围,要初始化

double min\_disX(double x,double &y){//x为定值,三分y

double l=miY,r=maY, d1,d2;

while( r-l>eps ){

double y1=l+(r-l)\*alf, y2=r-(r-l)\*alf;

d1=all\_dis(P(x,y1)), d2=all\_dis(P(x,y2));

if( d1>d2 ) l=y1;

else r=y2;

}

y=l;

return all\_dis(P(x,y));

}

double min\_dis(double &x,double &y){//三分x

double l=miX,r=maX, d1,d2;

while( r-l>eps ){

double x1=l+(r-l)\*alf, x2=r-(r-l)\*alf;

d1=min\_disX(x1,y), d2=min\_disX(x2,y);

if( d1>d2 ) l=x1;

else r=x2;

}

x=l;

return min\_disX(x,y);

}

int main(){

while( scanf("%d",&N)!=EOF ){

miX=miY=1000000000.0; maX=maY=0;

for(int i=0;i<N;i++) {

scanf("%lf %lf",&p[i].x,&p[i].y);

miX=min(miX,p[i].x);

miY=min(miY,p[i].y);

maX=max(maX,p[i].x);

maY=max(maY,p[i].y);

}

double x,y,ans;

ans=min\_dis(x,y);

printf("(%.3lf,%.3lf) %.4lf\n",x,y,ans);

}

}

## 随机增量算法

//最小覆盖圆,随即增量算法O(n) zju 1450

/\*

algorithm:

A、令Ci表示为前i个点的最小覆盖圆。当加入新点pi时如果pi不在Ci-1里那么pi必定在Ci的边界上。

B、再从新考虑这样一个问题，Ci为前i个点最小覆盖圆且p在Ci的的边界上！同理加入新点pi时如果pi不在Ci-1里那么pi必定在Ci的边界上。这时我们就包含了两个点在这个最小圆的边界上。

C、再从新考虑这样一个问题，Ci为前i个点最小覆盖圆且有两个确定点再边界上！此时先让

\*/

struct Point{

double x,y;

};

int n;

Point p[200000],O;

double r,eps=1e-8;

double dist(Point a,Point b){

return sqrt( (a.x-b.x)\*(a.x-b.x)+(a.y-b.y)\*(a.y-b.y) );

}

//注意到三角形里两条中垂线不可能平行，所以不会产生除0错误

void calc(double a,double b,double c,double d,double e,double f){ //给出两条直线ax+by+c=0,dx+ey+f=0 求交点

O.y=(c\*d-f\*a)/(b\*d-e\*a);

O.x=(c\*e-f\*b)/(a\*e-b\*d);

}

int main(){

srand(time(NULL));

while(scanf("%d",&n)!=EOF && n ){

for (int i=1;i<=n;++i) scanf("%lf%lf",&p[i].x,&p[i].y);

for(int i=1;i<=n;i++){ //将点随机化，防止线性递增的序列使得复杂度变成n^3

swap(p[i],p[rand()%n+1]);

}

O=p[1];r=0; //初始C1

for (int i=2;i<=n;++i){

if (dist(O,p[i])<r+eps) continue; //有1/i的概率不continue

O=p[i];r=0;

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

if (dist(O,p[j])<r+eps) continue; //有1/j的概率不continue

O.x=(p[i].x+p[j].x)/2;O.y=(p[i].y+p[j].y)/2;r=dist(O,p[j]);

for (int k=1;k<=j-1;++k){

if (dist(O,p[k])<r+eps) continue; //有1/k的概率不continue

calc(p[j].x-p[i].x,p[j].y-p[i].y,

(p[j].x\*p[j].x+p[j].y\*p[j].y-p[i].x\*p[i].x-p[i].y\*p[i].y)/2,

p[k].x-p[i].x,p[k].y-p[i].y,

(p[k].x\*p[k].x+p[k].y\*p[k].y-p[i].x\*p[i].x-p[i].y\*p[i].y)/2);

r=dist(O,p[k]);

}

}

}

printf("%.2lf %.2lf %.2lf\n",O.x,O.y,r);

}

return 0;

}

## 最小包含球

//PKU2069 Super Star 最小包含球

const int MAX=120;

const double INF=1e20;

const double eps=1e-6;

struct point{

double x,y,z;

}ps[MAX],q; //q是球心\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int n; //多少个点,标号0~n-1

double dist(point a,point b){

a.x-=b.x;

a.y-=b.y;

a.z-=b.z;

return sqrt(a.x\*a.x+a.y\*a.y+a.z\*a.z);

}

int maxD(point p){

double res=0;

int k=0;

for(int i=0;i<n;++i){

double tmp=dist(p,ps[i]);

if(tmp>res){

k=i;

res=dist(p,ps[i]);

}

}

return k;

}

double min\_ball(){

double step=100; //精度\*\*\*\*\*\*\*\*\*\*\*\*

double radius=INF;

q.x=q.y=q.z=0.0;

int k=1;

while(step>eps){ //精度没到,继续做

int d=maxD(q);

double tmp=dist(ps[d],q);

if(tmp<radius)

radius=tmp;

double dx=ps[d].x-q.x;

double dy=ps[d].y-q.y;

double dz=ps[d].z-q.z;

dx/=tmp;dy/=tmp;dz/=tmp;

q.x=q.x+dx\*step;

q.y=q.y+dy\*step;

q.z=q.z+dz\*step;

step\*=0.98;

}

return radius+1e-7;

}

int main(){

while(scanf("%d",&n),n){

for(int i=0;i<n;++i)

scanf("%lf%lf%lf",&ps[i].x,&ps[i].y,&ps[i].z);

//printf("%lf %lf %lf\n",q.x,q.y,q.z);

printf("%.5f\n",min\_ball() );

}

return 0;

}

组合数学

## N\*M数码可达性判断

//zjut 1462 N\*M数码可达性判断问题

//分情况讨论

//1：当N=1即0不能上下移动时。。

//2：当M为奇数时除0外的逆序数的奇偶性必须相同

//3：当M为偶数时，0上下移动一次，逆序数改变，但0的行号也改变，抓住这个

//以上所说的逆序数都是去掉0后的。。。

int a[2600];

int b[2600];

int sum[2600];

inline int lowbit(int t){ return t&(-t); }

void update(int i){

while(i<2600){ sum[i]++; i+=lowbit(i); }

}

int getsum(int i){

int s=0;

while(i>=1){ s+=sum[i]; i-=lowbit(i); }

return s;

}

int work(int \*t,int num){ //用树状数组求逆序数

memset(sum,0,sizeof(sum));

int ret=0;

for(int i=num;i>=1;i--){

ret+=getsum(t[i]);

update(t[i]);

}

return ret;

}

int main(){

int N,M;

while(scanf("%d%d",&N,&M)!=EOF){

int v,ind=1, ax,ay,bx,by;

for(int i=1;i<=N;i++){

for(int j=1;j<=M;j++){

scanf("%d",&v);

if(v) a[ind++]=v;

else ax=i,ay=j;

}

}

ind=1;

for(int i=1;i<=N;i++){

for(int j=1;j<=M;j++){

scanf("%d",&v);

if(v) b[ind++]=v;

else bx=i,by=j;

}

}

bool flag=true;

if(N==1){ //特殊考虑

for(int i=1;i<ind;i++)

if(a[i]!=b[i]){

flag=false;

break;

}

if(flag) puts("YES");

else puts("NO");

}else{

int suma=work(a,ind-1);

int sumb=work(b,ind-1);

if(M%2){

if(suma%2 == sumb%2) puts("YES");

else puts("NO");

}else{

suma+=ax-1; sumb+=bx-1;

if(suma%2 == sumb%2) puts("YES");

else puts("NO");

}

}

}

}

## Polya计数

//pku 1286 Polya

//题意: 用三种颜色的珠子组成总共有n个珠子的项链，问总共有多少种不同组法。2种项链旋转，翻转后能变为同一种的，看做是相同的

\_\_int64 \_pow(int a,int b){

\_\_int64 ans=1;

while(b){

if(b&1) ans\*=a;

a\*=a;

b>>=1;

}

return ans;

}

int gcd(int a,int b){

if(b==0) return a;

int temp;

while(a%b){

temp=a%b;

a=b;

b=temp;

}

return b;

}

int main(){

int n;

while(scanf("%d",&n)!=EOF&&n!=-1){

if(n==0){

printf("0\n");

continue;

}

\_\_int64 ans=0;

for(int i=0;i<n;i++)

ans+=\_pow(3,gcd(i,n));

if(n&1)

ans+=\_pow(3,n/2+1)\*n;

else

ans+=\_pow(3,n/2)\*n/2+\_pow(3,n/2+1)\*n/2;

printf("%I64d\n",ans/2/n);

}

}

## 公式

错排公式为M(n)=n!(1/2!-1/3!+…..+(-1)^n/n!)

容斥原理

1, 

2, 

n个不同的点,使之连通的连边方案数(无重边). pku1737



## 容斥原理

使用容斥,关键是先定好n个条件,也就是所谓的n个圈圈,然后求n个圈圈的并集神马的....

1, 

2, 

## 广义容斥原理

给定集合N和性质

令,

则计数了具有m+k个性质的元素次.

令表示恰好有m种性质的元素个数,则有



特别的.错排公式可由这个推导出来.

应用

## 一维最多不交叉线段数,基于递归的dp状态

/\*

problem : 给你一些线段,求最多能找到几条是相互之间都没有交叉关系的

分析下解的结构: 解是由一些线段组成,而这些线段的上面还有线段(被它包含着的),那么很明显这是个递归定义的结构

如果我们在原来基础上再加上一条[-inf,inf]的线段,那么解就有更加清晰地子结构了...

算法1： 可以将一条线段拆分成两个点,然后变成了2\*n个点,对这些点排序后进行dp,复杂度为(2\*n)^3

算法2： 我们设计一个基于递归定义的状态,dp[i][j]表示线段i之后以线段j结束的结构的最大值,这样可以枚举结束点来转移 复杂度n^3

\*/

#define maxn 305

struct DD{

int L,R;

bool operator<(const DD &b)const{

if( L!=b.L ) return L < b.L;

return R>b.R;

}

}dd[maxn];

int dp[maxn][maxn]; //dp[i][j]表示第i段上面以第j段为最后的结构最多有几段

int mdp[maxn]; //mdp[i]表示第i段上面连同自己共有几段

int main(){

int n,cas; cin>>cas;

while(cas-- && cin>>n){

for(int i=0;i<n;i++) cin>>dd[i].L>>dd[i].R;

dd[n].L=-0x6fffffff,dd[n++].R=0x6fffffff;

sort(dd,dd+n);

memset(dp,0,sizeof(dp));

for(int i=0;i<n;i++) mdp[i]=1;

for(int i=n-1;i>=0;i--){

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

if( dd[j].R> dd[i].R) continue;

int ma=1+mdp[j];

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

if( dd[k].R<=dd[i].R && dd[k].R<=dd[j].L ) //k在i内,且和j不相交

ma=max(ma,dp[i][k]+mdp[j]);

}

dp[i][j]=max(dp[i][j],ma);

mdp[i]=max(mdp[i],ma);

//printf("dp[%d][%d]=%d\n",i,j,dp[i][j]);

}

}

cout<<mdp[0]-1<<endl;

}

}

最长公共子串转最长上升子序列

/\*

設有序列A，B。

記序列A中各個元素在B 中的位子(降序排列)

然後按在A中的位置依次列

出按後求A的最長遞增子序列

例如：有A={a,b,a,c,x}，B={b,a,a,b,c,a}

則有a={6,3,2},b={4,1},c={5};x=/;（注意降序排列）

然後按A中次序排出

{a(6,3,2),b(4,1),a(6,3,2),c(5),x()}={6,3,2,4,1,6,3,2,5}；

對此序列求最長遞增子序列即可~~

\*/

int sq[100005],inds;//0~inds-1

int y[10005][12];//

int iy[10005]; //存放下标

int a[100010],b[100010];

int LCS(int n,int m){//最长公共子序列nlogn

int len=n\*m;

memset(iy,0,sizeof(iy));

for(int i=1;i<=len;i++){

int v=a[i];

y[ v ][ iy[v]++ ]=i;

}

for(int i=1;i<n;i++){

sort(y[i],y[i]+m);

}

inds=0;

for(int i=1;i<=len;i++){

for(int j=m-1;j>=0;j--) {

int v=y[ b[i] ][j];//1~100000

if( inds==0 ){

sq[inds++]=v;

continue;

}

int left=0,right=inds-1;

if(sq[left]>=v){

sq[left]=min(sq[left],v);

continue;

}

if( sq[right]<v){

sq[inds++]=v;

continue;

}

while( left+1<right){

int mid=(left+right)>>1;

if( sq[mid] >= v) right=mid;

else left=mid;

}

sq[right]=min(sq[right],v);

}

}

return inds;

}

## 不同子串个数 SA nlog(n)

++++后缀数组模板++++

int diff\_substr(char \*str){ //计算不同子串个数nlog(n), str base1

n=strlen(str+1);

memcpy(data,str,(n+1)\*sizeof(char));

make\_SA(); make\_H();

int ret=n-SA[1]+1;

for(int i=2;i<=n;i++){

int l=n-SA[i]+1;

if( l>H[i] ) ret+=l-H[i];

}

return ret;

} //diff\_substr(str);...

## 不同子串个数 哈希n^2logn

#define maxn 2100

typedef unsigned long long uint64;

typedef pair<uint64,uint64> PUU;

int b1=130, b2=131; //进制

uint64 pow1[maxn], pow2[maxn]; //幂次

uint64 h1[maxn],h2[maxn];

PUU res[maxn\*maxn];

PUU getValue(int l,int r){

return PUU( h1[r]-h1[l-1]\*pow1[r-l+1], h2[r]-h2[l-1]\*pow2[r-l+1] );

}

int calc(char \*str){ //计算一个字符串有多少个不同的子串,base1

int len=strlen(str+1);

pow1[0]=pow2[0]=1;

for(int i=1;i<=len;i++){

pow1[i]=pow1[i-1]\*b1;

pow2[i]=pow2[i-1]\*b2;

}

for(int i=1;i<=len;i++){

h1[i]=h1[i-1]\*b1 + str[i];

h2[i]=h2[i-1]\*b2 + str[i];

}

int ind=0;

for(int i=1;i<=len;i++){

for(int j=i;j<=len;j++){

//if( j-i+1<L || can1[i]==false || can2[j]==false ) continue;

res[ind++]=getValue(i,j);

}

}

sort(res,res+ind);

ind=unique(res,res+ind)-res;

return ind;

}

## 部分点最小生成树

/\*

部分点最小生成树(steiner tree),共n个点,前sel个是需要连通的

算法: dp[i][mask]表示覆盖的点集为mask,向外扩展的接口是i的情况下的最小树...

由于没有什么拓扑顺序,所以只能选择spfa一样的逐步求精的思路...

复杂度 n\*(1<<sel)^2

\*/

#define maxn 1100

#define maxm 10500

struct Edge{ int v,val,next; }E[2\*maxm];

int list[maxn],eid;

void init(){ memset(list,-1,sizeof(list)); eid=0; }

inline void insert(int a,int b,int val){

E[eid].v=b; E[eid].val=val; E[eid].next=list[a]; list[a]=eid++;

E[eid].v=a; E[eid].val=val; E[eid].next=list[b]; list[b]=eid++;

}

int dp[maxn][1<<6]; //总状态为1000\*2^6的最短路

bool in[maxn][1<<6];

queue<pair<int,int> > q;

int steiner(int n,int sel){

for(int i=0;i<n;i++)

for(int j=0;j<(1<<sel);j++)

dp[i][j]=0x1fffffff,in[i][j]=false;

while( !q.empty()) q.pop();

int u,msk;

for(int i=0;i<n;i++){ //初始化队列

if(i<sel){ //主动扩展的肯定是selected的那些点

dp[i][1<<i]=0;

q.push(pair<int,int>(i,1<<i) );

in[i][1<<i]=true;

}else

dp[i][0]=0;

}

int minCost=0x7fffffff;

while( !q.empty() ){

u=q.front().first, msk=q.front().second;

q.pop(); in[u][msk]=false;

if( msk==(1<<sel)-1 ) minCost=min(minCost,dp[u][msk]); //剪枝

if( dp[u][msk]>=minCost ) continue;

for(int p=list[u];p!=-1;p=E[p].next){

int v=E[p].v, sub;

int M=(1<<sel)-1 - msk; //

for(sub=M;;sub=(sub-1)&M){

int nxt=sub|msk;

if( dp[v][nxt] > dp[u][msk]+dp[v][sub]+E[p].val){

dp[v][nxt] = dp[u][msk]+dp[v][sub]+E[p].val;

if( in[v][nxt]==false ){

in[v][nxt]=true;

q.push( pair<int,int>(v,nxt) );

}

}

if( sub==0 ) break; //

}

}

}

return minCost;

}

int main(){

int n,m,p;

while(scanf("%d%d%d",&n,&m,&p)!=EOF){

init();

for(int i=1;i<=n+m;i++) {

int a; scanf("%d",&a);

insert(0,i,a);

}

for(int i=0;i<p;i++){

int a,b,val;

scanf("%d%d%d",&a,&b,&val);

insert(a,b,val);

}

printf("%d\n",steiner(n+m+1,n+1));

}

}

## 插头dp

/\*

fzu 1997 插头dp 模板样式

注意点:

1， 如果要求是只有一个环,那么只有最后一个空闲格子才能"()"直接连

2， 往后找第一个 没匹配的 右括号!!!

3, 如果状态太稀疏,可以预处理下

插头dp解题步骤:

0, 思考状态的表示

1, 搜索有效状态 (可能需要记录括号的匹配情况,getMatch()函数)

2, 存入hash表中(可用STL map替代)

3, L,U 分情况讨论

4, 每次需要将第i行末的转到第i+1行首

用STL的map比用自己的HashTable慢3倍

Number的位运算的比不用位运算的快1.5倍

\*/

### 插头dp 2进制

//hdu 1693 吃树 最简单的插头dp

#include<iostream>

using namespace std;

int mat[11][11];

\_\_int64 dp[12][12][1<<12];

inline bool test(int v,int w){ //测试整数v的第w位(base0)

return v&(1<<w);

}

inline int set(int v,int w){

if( v&(1<<w) ) return v;

else return v|(1<<w);

}

inline int reset(int v,int w){

if( v&(1<<w) ) return v^(1<<w);

else return v;

}

int main(){

int cas,n,m,Te=1; cin>>cas;

while(cas-- ){

cin>>n>>m;

for(int i=0;i<n;i++) for(int j=0;j<m;j++) cin>>mat[i][j];

memset(dp,0,sizeof(dp));

dp[0][0][0]=1;

for(int i=0;i<n;i++){

for(int j=0;j<m;j++){

for(int s=0;s<(1<<(m+1));s++){

if( dp[i][j][s]==0 ) continue;

if( mat[i][j]==0 ){

if( (s&(3<<j)) == 0 ) dp[i][j+1][s]+=dp[i][j][s];

}else{

if( test(s,j)==false && test(s,j+1)==false ){

dp[i][j+1][s+(3<<j)]+=dp[i][j][s];

}else if( test(s,j)==true && test(s,j+1)==false) {

dp[i][j+1][s]+=dp[i][j][s];

dp[i][j+1][set( reset(s,j),j+1) ]+=dp[i][j][s];

}else if( test(s,j)==false && test(s,j+1)==true) {

dp[i][j+1][s]+=dp[i][j][s];

dp[i][j+1][set(reset(s,j+1),j) ]+=dp[i][j][s];

}else{

dp[i][j+1][reset(reset(s,j+1),j) ]+=dp[i][j][s];

}

}

}

}

for(int s=0;s<(1<<(m));s++){

dp[i+1][0][s<<1]=dp[i][m][s];

}

}

printf("Case %d: There are %I64d ways to eat the trees.\n",Te++,dp[n][0][0]);

//cout<<dp[n][0][0]<<endl;

}

}

### 插头dp 3进制

//用于连通性状态压缩dp

template<int N> //N只能为2,4,8,下面全是是用位运算实现的

struct Number{

static int w[30]; //X=b0\*w[0]+b1\*w[1]+...+b[n]\*w[n]

static int L;

static void init(){ //对于每个实例必须先初始化

L=N/2; w[0]=1;

for(int i=1;i<30;i++) w[i]=w[i-1]<<L;

}

int toInt(int \*d,int ind){ //将一个数组计算出一个N进制数,ind base0

int ret=0;

for(int i=ind-1;i>=0;i--) ret=(ret<<L)|d[i];

return ret;

}

void parse(int val,int \*d,int &ind){ //将一个N进制数解析成一个数组

ind=0;

while( val ){ d[ind++]=val&( (1<<L)-1 ); val>>=L; }

if( ind==0 ) d[ind++]=0;

}

int got(int val,int wei){ //取出N进制数val的第wei位

return (val & (w[wei+1]-1) )>>(wei<<(L>>1));

}

void set(int &val,int wei,int b){ //将N进制数val的第wei位设置为b

val+=( (b-got(val,wei))<<(wei<<(L>>1)) );

}

};

template<int N>

int Number<N>::w[30];

template<int N>

int Number<N>::L;

//\*\*\*\*\*\*\*\*\*\*\*\*模板结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//映射关系TK -> TV, 映射范围\_mod, 总节点数目\_n

//默认没有重复的key

template<class TK,class TV,int \_mod=9997,int \_n=50000>

class HashTable{

struct H{

TK key;

TV val;

int next;

}h[\_n];

int list[\_mod],eid;

int (\*hash) (TK);

public:

HashTable(){ //构造函数

clear();

}

void clear(){ //清0

memset(list,-1,sizeof(list));

eid=0;

}

void setCallBack(int (\*hash)(TK) ){ //设置回调函数(哈希函数)

this->hash=hash;

}

int find(TK key){ //查找关键字是否在哈希表中,没有返回-1

int ind=hash(key);

for(int p=list[ind];p!=-1;p=h[p].next){

if( h[p].key == key ) return p;

}

return -1;

}

void insert(TK key,TV val){ //插入

int ind=hash(key);

h[eid].key=key; h[eid].val=val;

h[eid].next=list[ind]; list[ind]=eid++;

}

TV& operator[] (TK key){ //等同于map的[],必须保证里面有key,否则下标为-1

return h[ find(key) ].val;

}

void print(TK \*out,int &N){ //0~N-1

for(int i=0;i<eid;i++)

out[i]=h[i].key;

N=eid;

}

};

const int mod=9997;

inline int h(int V){

return V%mod;

}//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*模板结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Number<4> ins;

int n,m; long long dp[2][41835]; //真正可用的状态时41835个

HashTable<int,int> ht;

int b[20], sta[50000],index;

void dfs(int pos,int val,int bracket){

if( pos==m+1 ){

if( bracket==0 ){

sta[index++]=val;

}

return;

}

dfs(pos+1, val, bracket);

dfs(pos+1, val+1\*b[pos], bracket+1);

if( bracket>0 ) dfs(pos+1, val+2\*b[pos], bracket-1);

}

void init(){

b[0]=1; for(int i=1;i<=16;i++) b[i]=b[i-1]\*4;

index=0;

dfs(0,0,0); //搜索所有的有用状态,存在sta数组中\*\*\*\*\*\*\*\*

sort(sta,sta+index);

ht.setCallBack(h);

ht.clear();

for(int i=0;i<index;i++) ht.insert(sta[i],i); //虽然状态数目不多,但是状态值很大,不能用数组直接映射...故用hash表\*\*\*\*\*\*\*

//cout<<index<<endl;

}

int lx,ly;

inline bool can(int i,int j){

if( i!=lx ) return i>lx;

return j>=ly;

}

int main(){

ins.init();

int d[30],ind;

int cas,Te=1; char mat[15][15];

scanf("%d",&cas);

while( cas-- && scanf("%d %d",&n,&m) ){

if( n==0 && m==0 ) break;

memset(dp,0,sizeof(dp));

init();

lx=-1,ly=-1;

for(int i=n-1;i>=0;i--){

scanf(" %s",mat[i]);

}

for(int i=0;i<n;i++) for(int j=0;j<m;j++){ //寻找最后那个可以左右括号连结的坐标

if( mat[i][j]=='O' ){

lx=i,ly=j;

}

}

dp[0][ ht[ 0 ] ]=1;

bool flag=false; //flag -> !flag

long long ans=0;

for(int i=0;i<n;i++){

for(int j=0;j<m;j++){

for(int p=0;p<index;p++){

int s=sta[p]; //状态值为s,可能是10^8级别\*\*\*\*\*

if( dp[flag][p]==0 ) continue;

int L=ins.got(s,j), U=ins.got(s,j+1); //取得相应位置的插头类型\*\*\*\*

if( mat[i][j]=='X' ){

if( L==0 && U==0 ) dp[!flag][p]+=dp[flag][p];

}else{

if( L==0 && U==0 ){

if( mat[i][j]=='\*' ){

dp[!flag][p]+=dp[flag][p];

}

int ns=s;

ins.set(ns,j,1);

ins.set(ns,j+1,2);

dp[!flag][ht[ns] ]+=dp[flag][p];

}else if( L==0 || U==0 ){

dp[!flag][p]+=dp[flag][p];

int ns=s;

ins.set(ns,j,U);

ins.set(ns,j+1,L);

dp[!flag][ ht[ns] ]+=dp[flag][p];

}else if( L==1 && U==1 ){

ins.parse(s,d,ind);

d[j]=d[j+1]=0;

for(int k=j+2,num1=0;;k++){ //往后找第一个 没匹配的 右括号!!!

if( d[k]==0 ) continue;

if( d[k]==1 ) { num1++; continue; };

if( num1==0 ){ d[k]=1; break; }

if( num1 ) num1--;

}

int ns=ins.toInt(d,ind);

dp[!flag][ ht[ns] ]+=dp[flag][p];

}else if( L==2 && U==2 ){

ins.parse(s,d,ind);

d[j]=d[j+1]=0;

for(int k=j-1,num2=0;;k--){

if( d[k]==0 ) continue;

if( d[k]==2 ) { num2++; continue; };

if( num2==0 ){ d[k]=2; break; }

if( num2 ) num2--;

}

int ns=ins.toInt(d,ind);

dp[!flag][ht[ns] ]+=dp[flag][p];

}else if( L==2 && U==1 ){

int ns=s;

ins.set(ns,j,0);

ins.set(ns,j+1,0);

dp[!flag][ ht[ns] ]+=dp[flag][p];

}else if( L==1 && U==2 ){

if( can(i,j) ){

int ns=s;

ins.set(ns,j,0);

ins.set(ns,j+1,0);

if( ns==0 ) ans+=dp[flag][p];

//dp[!flag][ rnk[ns] ]+=dp[flag][p];

}

}

}

}

flag=!flag;

memset(dp[!flag],0,sizeof(long long)\*index);

}

for(int p=0;p < index;p++){

int s=sta[p];

if((s<<2)> sta[index-1] ) break;

dp[!flag][ ht[(s<<2)] ]=dp[flag][ p ];

}

flag=!flag;

memset(dp[!flag],0,sizeof(long long)\*index);

}

printf("Case %d: ",Te++);

ans+=dp[flag][0];

cout<<ans<<endl;

}

}

### 插头dp 4进制

// SCU 3994 4进制插头dp, 从(0,0)到(n-1,m-1)的路径有多少条(有障碍)

#include<...>

using namespace std;

//每个实例都是相应进制的函数集合, 用于连通性状态压缩dp

template<int N> //N只能为2,4,8,

struct Number{

int w[30]; //X=b0\*w[0]+b1\*w[1]+...+b[n]\*w[n]

int L,mask; //二进制位数,

Number(){ //初始化

if( N==2 ) L=1;

else if( N==4 ) L=2;

else L=3;

mask=N-1; w[0]=1;

for(int i=1;i<30;i++) w[i]=w[i-1]<<L;

}

int arr2int(int \*d,int t){ //将一个数组计算出一个N进制数,ind base0

int s=0;

for(int i=t-1;i>=0;i--) s=(s<<L)|d[i];

return s;

}

void int2arr(int s,int \*d,int &t){ //将一个N进制数解析成一个数组

t=0;

while( s ){ d[t++]=s&mask; s>>=L; }

if( t==0 ) d[t++]=0;

}

int test(int s,int wei){ //取出N进制数s的第wei位 wei:base0

return (s>>(wei\*L))&mask;

}

void set(int &s,int wei,int b){ //将N进制数s的第wei位设置为b wei:base0

s+=( (b-test(s,wei))<<(wei\*L) );

}

};

Number<4> tool4;

//\*\*\*\*\*\*\*\*\*\*\*\*模板结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*

插头dp解题步骤:

0, 思考状态的表示

1, 搜索有效状态 (可能需要记录括号的匹配情况,getMatch()函数)

2, 存入hash表中(可用STL map替代)

3, L,U 分情况讨论

4, 每次需要将第i行末的转到第i+1行首

\*/

map<int,int> ht;

int sta[9713],ind;//valid status

int n,m; //n\*m的格子 <=10

char mat[12][12]; //地图

int dp[12][12][9713]; // 经过搜索,只有5798个有效状态

int match[9713][12]; // 每个状态的括号序列中的匹配关系

int \_buf[20];

stack<int> \_st;

bool getMatch(){ //对括号进行匹配,顺便测试 独立插头是否被包在一对括号里

while( !\_st.empty() ) \_st.pop();

for(int i=0;i<m+1;i++){

if( \_buf[i]==1 ) \_st.push(i);

else if( \_buf[i]==2 ){

match[ind][i]=\_st.top();

match[ind][\_st.top()]=i;

\_st.pop();

}else if( \_buf[i]==3 && \_st.size() ) return false; //返回失败

}

return true; //返回成功

}

void dfs(int p,int sum,bool f){

if( p==m+1 ){

if( sum || !f) return;

if( getMatch() ) sta[ind++]=tool4.arr2int(\_buf,m+1);

return;

}

\_buf[p]=0; dfs(p+1,sum,f);

\_buf[p]=1; dfs(p+1,sum+1,f);

if( sum ){

\_buf[p]=2; dfs(p+1,sum-1,f);

}

if( f==false ){

\_buf[p]=3; dfs(p+1,sum,true);

}

}

void findState(){

ind=0;

ht.clear(); //ht.setCallBack(h);

dfs(0,0,false); //不能排序了,因为match数组已经确定了

//printf("total state: %d\n",ind);

for(int i=0;i<ind;i++)

ht[ sta[i] ]=i;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

inline void add(int &a,int b){

a=(a+b)%10007;

}

int main(){

while( scanf("%d %d",&n,&m)!=EOF ){

for(int i=0;i<n;i++) scanf(" %s",mat[i]);

findState();

for(int i=0;i<=n;i++) for(int j=0;j<=m;j++) for(int k=0;k<ind;k++) dp[i][j][k]=0;

dp[0][0][ ht[3] ]=1; //只有一个独立插头

for(int i=0;i<n;i++){

for(int j=0;j<m;j++){

for(int k=0;k<ind;k++){

int s=sta[k]; // s: real state

int L=tool4.test(s,j), U=tool4.test(s,j+1);

if( mat[i][j]=='#' ){

if( L==0 && U==0 ){

add(dp[i][j+1][k],dp[i][j][k]);

}

continue;

}

if( L==0 && U==0 ){

add(dp[i][j+1][k],dp[i][j][k]);

int ns=s;

tool4.set(ns,j,1);

tool4.set(ns,j+1,2);

add(dp[i][j+1][ht[ns]],dp[i][j][k]);

}else if( L==0 || U==0 ){

add(dp[i][j+1][k],dp[i][j][k]);

int ns=s;

tool4.set(ns,j,U);

tool4.set(ns,j+1,L);

add(dp[i][j+1][ht[ns]],dp[i][j][k]);

}else if( L==1 && U==1 ){

int ns=s;

tool4.set(ns,j,0);

tool4.set(ns,j+1,0);

tool4.set(ns,match[k][j+1],1);

add(dp[i][j+1][ht[ns]],dp[i][j][k]);

}else if( L==1 && U==2 ){

//do nothing

}else if( L==2 && U==1 ){

int ns=s;

tool4.set(ns,j,0);

tool4.set(ns,j+1,0);

add(dp[i][j+1][ht[ns]],dp[i][j][k]);

}else if( L==2 && U==2 ){

int ns=s;

tool4.set(ns,j,0);

tool4.set(ns,j+1,0);

tool4.set(ns,match[k][j],2);

add(dp[i][j+1][ht[ns]],dp[i][j][k]);

}else if( L==1 && U==3 ){

//invalid state

}else if( L==2 && U==3 ){

int ns=s;

tool4.set(ns,j,0);

tool4.set(ns,j+1,0);

tool4.set(ns,match[k][j],3);

add(dp[i][j+1][ht[ns]],dp[i][j][k]);

}else if( L==3 && U==1 ){

int ns=s;

tool4.set(ns,j,0);

tool4.set(ns,j+1,0);

tool4.set(ns,match[k][j+1],3);

add(dp[i][j+1][ht[ns]],dp[i][j][k]);

}else if( L==3 && U==2 ){

//invalid state

}

}

}

int limit=0;

tool4.set(limit,m,1);

for(int k=0;k<ind;k++){

int ns=sta[k]\*4;

if( ht.find(ns)!=ht.end() ){

dp[i+1][0][ ht[ns] ]=dp[i][m][k];

}

}

}

int final=0;

tool4.set(final,m,3);

cout<<dp[n-1][m][ht[final]]<<endl;

}

}

## 单串随机生成期望

//单串随即生成期望问题 zju 2916 Generator

//套用抛硬币的结论

char s[14];

int pi[14];

long long int a[14];

void kmp(){//计算fail指针，即next函数

pi[1]=0;

int len=strlen(s+1),q=0;

for(int i=2;i<=len;i++){

while(q>0&&s[q+1]!=s[i])

q=pi[q];

if(s[q+1]==s[i])

q++;

pi[i]=q;

}

return ;

}

int main(){

int cas,N,t=1;

scanf("%d",&cas);

while(cas--){

scanf("%d",&N);

scanf("%s",s+1);

kmp();

int len=strlen(s+1);

a[0]=1;//N的几次方，每次产生的东西有N中可能性

for(int i=1;i<=len;i++)

a[i]=N\*a[i-1];

long long int ans=0;

ans+=a[len];//套抛硬币的结论

int q=pi[len];

while(q){

ans+=a[q];//套抛硬币的结论

q=pi[q];

}

if(t!=1)

puts("");

printf("Case %d:\n%lld\n",t++,ans);

}

}

## 多项式去括号

char str[10005];

int bro[1005];

bool out[1005];

//多项式去括号!!!

bool dfs(int L,int R){ //返回是否是一个多项式

if( str[L]=='(' && bro[L]==R ){

out[L]=out[R]=false;

return dfs(L+1,R-1);

}

bool ret=false;

stack<pair<int,int> > st;

for(int i=L;i<=R;i++){

if( isalpha(str[i]) ){

st.push(make\_pair(i,i));

}else if( str[i]=='+' ){

ret=true;

if( st.size() == 1){

if( st.top().first != st.top().second ){

out[ st.top().first ]=false;

out[ st.top().second]=false;

}

st.pop();

}else{

while( !st.empty()) st.pop();

}

}else if( str[i]=='(') {

bool flag=dfs(i+1,bro[i]-1);

if( flag==false ) out[i]=out[ bro[i] ]=false;

st.push(make\_pair(i,bro[i]));

i=bro[i];

}

}

if( st.size() == 1){

if( st.top().first != st.top().second ){

out[ st.top().first ]=false;

out[ st.top().second]=false;

}

}

return ret;

}

int main(){

while( scanf(" %s",str)!=EOF ){

stack<int> st;

memset(out,true,sizeof(out));

for(int i=0;str[i];i++){ //匹配好

if( str[i]=='(' ){

st.push(i);

}else if( str[i]==')' ){

int s=st.top();

st.pop();

bro[s]=i, bro[i]=s;

}else if( str[i]=='+') {

bro[i]=-1;

}else{

bro[i]=i;

}

}

int len=strlen(str);

dfs(0,len-1);

for(int i=0;i<len;i++) if( out[i] ) putchar(str[i]);

puts("");

}

}

## 格点计数

char str[1000005];

char str2[2000005];

//格点计数,比较恶心的-\_-

int main(){

while( scanf(" %s",str)!=EOF ){

int len=strlen(str);

int len2=0;

for(int i=0;i<len;i++){

if( (str[i]-'0')%2==0 ){

str2[len2++]=str[i];

}else{

if( str[i]=='1' ){

str2[len2++]='2';

str2[len2++]='0';

}else if( str[i]=='3' ){

str2[len2++]='4';

str2[len2++]='2';

}else if( str[i]=='5' ){

str2[len2++]='6';

str2[len2++]='4';

}else{

str2[len2++]='0';

str2[len2++]='6';

}

}

}

\_\_int64 sum=0; int x=0,y=0,last=str2[len2-1]-'0';

for(int i=0;i<len2;i++){ //下面这段的处理值得注意下!!

if( str2[i]=='0' ){

if( last==4 ) sum++;

else if( last==6 || last==0 )sum-=(y-1);

x++;

}else if( str2[i]=='2' ){

if( last==0 )sum-=(y-1);

else if( last==6 ) sum-=(y-1);

y++;

}else if( str2[i]=='4' ){

if( last==0 ) sum++;

else if(last==2 || last==4 ) sum+=y;

x--;

}else {

if( last==2 || last==4 ) sum+=y;

y--;

}

last=str2[i]-'0';

}

if( sum<0 ) sum=-sum;

cout<<sum<<endl;

}

}

## 矩形面积并 n^3

//矩形面积并，复杂度n^3

typedef long long int64;

#define maxn 1005

struct Rec{

int x1,y1,x2,y2;

bool operator<(const Rec &b)const{

return (x2-x1)\*(y2-y1) > (b.x2-b.x1)\*(b.y2-b.y1);

}

}r[maxn];

int n; int64 ans; //n base0

void dfs(int x1,int y1,int x2,int y2,int i){

if( i==n || x1==x2 || y1==y2 ) return;

int \_x1=max(x1,r[i].x1);

int \_y1=max(y1,r[i].y1);

int \_x2=min(x2,r[i].x2);

int \_y2=min(y2,r[i].y2);

if( \_x1<\_x2 && \_y1<\_y2 ){

ans+=(\_x2-\_x1)\*(int64)(\_y2-\_y1);

dfs( x1, y1,\_x1, y2,i+1);

dfs(\_x2, y1, x2, y2,i+1);

dfs(\_x1, y1,\_x2,\_y1,i+1);

dfs(\_x1,\_y2,\_x2, y2,i+1);

}else{

dfs(x1,y1,x2,y2,i+1);

}

}

int64 calc\_area(){ //矩形面积并，复杂度n^3

sort(r,r+n);

int x1=1<<30,y1=x1,x2=-x1,y2=x2;

for(int i=0;i<n;i++){

x1=min(x1,r[i].x1);

y1=min(y1,r[i].y1);

x2=max(x2,r[i].x2);

y2=max(y2,r[i].y2);

}

ans=0;

dfs(x1,y1,x2,y2,0);

return ans;

}//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*模板结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int main(){

while(cin>>n){

for(int i=0;i<n;i++) cin>>r[i].x1>>r[i].y1>>r[i].x2>>r[i].y2;

cout<<calc\_area()<<endl;

}

}

## 矩形面积并 nlogn

//矩形面积并 nlogn

typedef long long int64;

#define maxn 110000

int y[maxn]; //离散化了的y轴

struct \_Node{

int val;

int64 len; // 统计信息

//

int l,r; //boudd

void init(int \_l,int \_r){

l=\_l, r=\_r; val=0;

}

int mid() { return (l+r)>>1; }

}dd[maxn\*5];

void update(int i){

if( dd[i].val ) dd[i].len=y[ dd[i].r ]-y[ dd[i].l ];

else if( dd[i].l+1 == dd[i].r ) dd[i].len=0;

else dd[i].len=dd[i\*2].len+dd[i\*2+1].len;

}

void makeTree(int i,int l,int r){

dd[i].init(l,r);

if( l+1==r ) { dd[i].len=0; return; }

int mid=dd[i].mid();

makeTree(i\*2,l,mid);

makeTree(i\*2+1,mid,r);

update(i);

}

void modify(int i,int l,int r,int val){

if( dd[i].l==l && dd[i].r==r ){

dd[i].val+=val;

update(i);

return;

}

int mid=dd[i].mid();

if( r<=mid) modify(i\*2,l,r,val);

else if( l>=mid ) modify(i\*2+1,l,r,val);

else {

modify(i\*2,l,mid,val);

modify(i\*2+1,mid,r,val);

}

update(i);

}//\*\*\*\*\*\*\*\*\*\*\*\*\*\*线段树模板结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

struct Rec{

int x1,y1,x2,y2;

}r[maxn];

struct Line{

int x,y1,y2,val; //al=1左边，val=-1右边

void init(int \_x,int \_y1,int \_y2,int \_val){

x=\_x; y1=\_y1; y2=\_y2; val=\_val;

}

bool operator<(const Line &b)const{

if( x!=b.x ) return x<b.x;

return val>b.val;

}

}l[2\*maxn];

int64 calc\_area(Rec \*r,int n){ // base0

if(n==0) return 0;

int yid=0,lid=0;

for(int i=0;i<n;i++){

l[lid++].init(r[i].x1,r[i].y1,r[i].y2,1);

l[lid++].init(r[i].x2,r[i].y1,r[i].y2,-1);

y[yid++]=r[i].y1;

y[yid++]=r[i].y2;

}

sort(l,l+lid);

sort(y,y+yid);

yid=unique(y,y+yid)-y;

makeTree(1,0,yid-1);

int64 area=0;

for(int i=0;i<lid-1;i++){

int d=lower\_bound(y,y+yid,l[i].y1)-y;

int u=lower\_bound(y,y+yid,l[i].y2)-y;

modify(1,d,u,l[i].val);

area+=dd[1].len \* (l[i+1].x-l[i].x);

}

return area;

}//\*\*\*\*\*\*\*\*\*\*\*\*\*\*矩形面积并模板结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int main(){

int n;

while(cin>>n){

for(int i=0;i<n;i++) cin>>r[i].x1>>r[i].y1>>r[i].x2>>r[i].y2;

cout<<calc\_area(r,n)<<endl;

}

}

## 最佳整数问题,类似欧几里得的翻转

//求最小的ans使得存在k满足: z1/m1 <= k/ans <z2/m2 ............

//亦即对于给定的两个double,求最小的一个整数ans,使得ans\*low和ans\*high之间夹着一个整数

//类似欧几里得的算法.......(不停地反转,取出整数)

//flag表示=在哪边,false是>=,true是<=

void dfs(long long z1,long long m1,long long z2,long long m2,long long &k,long long &ans,bool flag){

if( z1<m1 && m2<z2 ){ //不能再反转了(取不出整数了)

if( z1==0 && flag==false) k=0;

else k=1;

ans=1;

return;

}

if( z1<m1 ){

dfs(m2,z2,m1,z1,ans,k,!flag);

return;

}

long long p=z1/m1;

long long \_k,\_ans;

dfs(z1-m1\*p,m1,z2-m2\*p,m2,\_k,\_ans,flag);

ans=\_ans;

k=p\*ans+\_k;

return;

}

long long mul[111];

int main(){

int Te=1;

mul[0]=1;

for(int i=1;i<=12;i++) mul[i]=mul[i-1]\*10;

long long A;

char \*B;

string s;

while( cin>>s ){

if( s[0]=='-' ) break;

A=s[0]-'0';

B=(char\*) s.substr(1).c\_str();

int lenB=strlen(B+1);

long long t=mul[lenB]; //cout<<" t "<<t<<endl;

long long BB=0; for(int i=1;i<=lenB;i++) BB=10\*BB+B[i]-'0';

t\*=10;

long long k,ans=0;

long long z1=A\*t+10\*BB-5;

long long z2=A\*t+10\*BB+5;

dfs(z1,t,z2,t,k,ans,false);

printf("Case %d: %lld\n",Te++,ans);

}

}

## 全排列哈希

int getHash(int \*a,int n){ //全排列的完美哈希,复杂度O(n^2),判重用的

int ret=0,mul=1;

for(int i=1;i<n;i++){

int rev=0;

for(int j=0;j<i;j++) if( a[j] > a[i] ) rev++;

ret+=mul\*rev;

mul\*=(i+1);

}

return ret;

}

## 树链剖分

//树链剖分 hdu3966 给定一棵树，动态改变一些树链上点的权值，动态查询

#define maxn 55000

struct Node{

int val;

int add; //延迟标记

//

int l,r;

Node \*ls,\*rs; //左右儿子

void init(int \_l,int \_r){

l=\_l,r=\_r; ls=rs=NULL; add=0;//延迟信息必须清0!!!

}

int mid(){ return (l+r)>>1; }

void push\_down(){ //保证有儿子

ls->add+=add;

rs->add+=add;

add=0;

if( ls->l==ls->r ) { ls->val+=ls->add; ls->add=0; }

if( rs->l==rs->r ) { rs->val+=rs->add; rs->add=0; }

}

void update(){ //保证有儿子

}

void modify(int a,int b,int val){

if( l==a && r==b ){

add+=val;

if( a==b ) { this->val+=add; add=0; }

return;

}

push\_down(); //要进入子树,先要把树根的延迟信息push下去

if( b<=mid() ) ls->modify(a,b,val);

else if( a>mid() ) rs->modify(a,b,val);

else {

ls->modify(a,mid(),val);

rs->modify(mid()+1,b,val);

}

}

int query(int id){

if( l==r ) return val;

push\_down();

if( id<=mid() ) return ls->query(id);

else return rs->query(id);

}

}dd[maxn\*5];

int e; // 初始化

Node\* \_alloc(int l,int r) { dd[e].init(l,r); return &dd[e++]; }

int d[maxn],ind;

Node\* \_make(int l,int r){ //构造一棵树,并返回树根

Node\* p=\_alloc(l,r);

if( l==r ) {

p->val=d[l]; return p; //这里用到d数组,需要初始化

}

p->ls=\_make(l,p->mid());

p->rs=\_make(p->mid()+1,r);

return p;

}//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*线段树模板结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

vector<int> adj[maxn]; //邻接表

struct Tree{

Node \*root;

int dep,pa; //这段树链的深度,父亲点

}tree[maxn];

int tid,sz[maxn],vst[maxn];

struct Info{

int tid,tind; //每个节点所在的 树编号,树上的id

}info[maxn];

int count(int u){ //计算子树大小

sz[u]=vst[u]=1;

for(int i=0;i<adj[u].size();i++){

int v=adj[u][i];

if( vst[v]==0 ){

sz[u]+=count(v);

}

}

return sz[u];

}

int v[maxn]; //初始值

queue<int> q; //son dep pa

void divide(int u,int dep,int pa){ //树链剖分 非递归写法

q.push(u); q.push(dep); q.push(pa);

while( !q.empty() ){

u=q.front(); q.pop();

dep=q.front(); q.pop();

pa=q.front(); q.pop();

ind=0;

while( true ){

vst[u]=1;

info[u].tid=tid, info[u].tind=ind; d[ind++]=v[u];

int ma=-1, ver=-1; // size,vertex

for(int i=0;i<adj[u].size();i++){

int v=adj[u][i];

if( vst[v] ) continue;

if( ma==-1 ){

ma=sz[v]; ver=v;

}else if( sz[v]>ma ){

q.push(ver); q.push(dep+1); q.push(u);

ma=sz[v], ver=v;

}else{

q.push(v); q.push(dep+1); q.push(u);

}

}

if( ma<0 ) break;

u=ver;

}

tree[tid].dep=dep; tree[tid].pa=pa;

tree[tid++].root=\_make(0,ind);

}

}

void paint(int a,int b,int val){ //a到b的树链上的前权值全部加val

int ta=info[a].tid, tb=info[b].tid;

int ida=info[a].tind, idb=info[b].tind;

if( ta == tb ){

if( ida>idb ) swap(ida,idb);

tree[ta].root->modify(ida,idb,val);

return;

}

if( tree[ta].dep > tree[tb].dep){

tree[ta].root->modify(0,ida,val);

paint(tree[ta].pa,b,val);

}else {

tree[tb].root->modify(0,idb,val);

paint(a,tree[tb].pa,val);

}

}//\*\*\*\*\*\*\*\*\*\*\*\*\*以上必须\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int find(int u){

return tree[ info[u].tid ].root->query(info[u].tind);

}

int main(){

int n,m,Q;

while( scanf("%d %d %d",&n,&m,&Q)!=EOF ){

e=0; //

for(int i=1;i<=n;i++) adj[i].clear();

for(int i=1;i<=n;i++) scanf("%d",v+i);

for(int i=1;i<n;i++){

int u,v; scanf("%d %d",&u,&v);

adj[u].push\_back(v);

adj[v].push\_back(u);

}

memset(vst,0,sizeof(vst));

count(1);

memset(vst,0,sizeof(vst));

tid=0; divide(1,0,-1);

while( Q-- ){

char op; scanf(" %c",&op);

if( op=='I' ){

int a,b,val; scanf("%d %d %d",&a,&b,&val);

paint(a,b,val);

}else if( op=='D' ){

int a,b,val; scanf("%d %d %d",&a,&b,&val);

paint(a,b,-val);

}else{

int u; scanf("%d",&u);

printf("%d\n",find(u));

}

}

}

}

## 树上基于点的分治

//pku 1741 树上点的分治 (递归版本)

#define maxn 10010

struct Edge{ int v,val,x; }E[2\*maxn];

int list[maxn],e;

inline void init(){

memset(list,-1,sizeof(list));

e=0;

}

inline void insert(int a,int b,int val){

E[e].v=b; E[e].val=val; E[e].x=list[a]; list[a]=e++;

}//邻接表 以上

int N,L,ans;

int vst[maxn],Te=1; //调用每个函数前都要Te++

bool isRoot[maxn];

int num[maxn];//子树有几个点,总共几个点

int calc(int \*d,int ind,int L){ //计算有多少对d[i]+d[j]<=L(i可能等于j),下标从0开始

int ret=0;

sort(d,d+ind);

for(int i=1,j=-1;i<ind;i++){

if( j>=0 && d[i]+d[j]>L )

while( j>=0 && d[i]+d[j]>L )j--;

else

while( j+1<i && d[i]+d[j+1]<=L )j++;

ret+=j+1;

}

return ret;

}

int d[maxn],ind;//距离数组

void \_dis(int u,int D){ //算距离,需先初始化ind

vst[u]=Te; d[ind++]=D;

for(int i=list[u];i>=0;i=E[i].x){

int v=E[i].v;

if( isRoot[v] || vst[v]==Te ) continue;

\_dis(v,D+E[i].val);

}

}

//以下可以重用

void \_count(int u){ //统计节点个数

vst[u]=Te; num[u]=1;

for(int i=list[u];i>=0;i=E[i].x){

int v=E[i].v;

if( isRoot[v] || vst[v]==Te ) continue;

\_count(v);

num[u]+=num[v];

}

}

int findRoot(int n,int u){ //n当前子树总点数

vst[u]=Te; bool can=true;

if( n-num[u] > (n+1)/2 ) return -1; //

for(int i=list[u];i>=0;i=E[i].x){

int v=E[i].v, r;

if( isRoot[v] || vst[v]==Te ) continue;

if( num[v] > (n+1)/2 ) can=false;

if( (r=findRoot(n,v))>=0 ) return r;

}

if( can ) return u;

return -1;

}

void dfs(int u){ //主递归过程

Te++; \_count(u);

Te++; int r=findRoot(num[u],u);

isRoot[r]=true;

Te++; ind=0; \_dis(r,0);

ans+=calc(d,ind,L);

for(int i=list[r];i>=0;i=E[i].x){

int v=E[i].v;

if( isRoot[v] ) continue;

Te++; ind=0; \_dis(v,0);

ans-=calc(d,ind,L-2\*E[i].val);

}

for(int i=list[r];i>=0;i=E[i].x){

int v=E[i].v;

if( isRoot[v] ) continue;

dfs(v);

}

}

int main(){

while( scanf("%d%d",&N,&L)){

if( N==0 && L==0 ) break;

memset(isRoot,false,sizeof(isRoot));

init();

for(int i=1;i<N;i++){

int a,b,val;

scanf("%d%d%d",&a,&b,&val);

insert(a,b,val);

insert(b,a,val);

}

ans=0;

dfs(1);

printf("%d\n",ans);

}

}

//树上基于点的分治 Spoj 1825. Free tour II

//题意: 题目大意：给定一棵带权树，节点分黑白两色，求一条最长的简单路径的长度，且满足路径上黑点不超过K

#define maxn 420005

#define maxm 3000005

struct Edge{ int v,val,next; }E[maxm];

struct AdjList{

static int e; //每个case初始化为0

int l[maxn]; //

void insert(int a,int b,int val=0){ //单向边\*\*\*\*\*\*\*\*\*\*\*\*\*\*

E[e].v=b; E[e].val=val; E[e].next=l[a]; l[a]=e++;

}

void init(){

memset(l,-1,sizeof(l));

}

}adj;

int AdjList::e;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*模板结束\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

const int inf=-(1<<29);

bool isr[maxn]; //是否被选为了根

int \_st[maxn],\_i[maxn],top; //栈

int vst[maxn],cnt; //标记

//对一颗子树进行拓扑排序,返回的arr中树根在前,子树在后,c base0

void topo\_sort(int u,int \*arr,int &c){

c=top=0;

cnt++;

\_st[++top]=u,\_i[top]=adj.l[u],arr[c++]=u,vst[u]=cnt;

while( top ){

u=\_st[top];

if( \_i[top]==-1 ) --top;

else{

int v=E[\_i[top]].v;

\_i[top]=E[\_i[top]].next;

if( vst[v]==cnt || isr[v] ) continue;

\_st[++top]=v,\_i[top]=adj.l[v],arr[c++]=v,vst[v]=cnt;

}

}

}

int n,K,m;

bool black[maxn];

int num[maxn];

int \_b[maxn],\_b2[maxn],\*ptr; //缓冲区

int findRoot(int u){ //在一颗子树中寻找一个合适的根

int n;

topo\_sort(u,\_b,n); //n=子树大小

cnt++;

for(int i=n-1;i>=0;i--){ //calc num[] array

int a=\_b[i], f=1; num[a]=1;

for(int j=adj.l[a];j!=-1;j=E[j].next){

int b=E[j].v;

if( vst[b]==cnt && !isr[b] ) {

num[a]+=num[b];

if( num[b]>n/2) f=0;

}

}

vst[a]=cnt;

if( n-num[a]>n/2 ) f=0;

if( f ) return a;

}

}

//以上函数可以通用 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

struct DD{

int \*d,s;

bool operator<(const DD &b)const{

return s<b.s;

}

}dd[maxn];

int ind;

typedef pair<int,int> PII;

PII p[maxn];

int ans;

inline void check(int &a,int b){

if( a<b ) a=b;

}

void dfs(int u){

int r=findRoot(u),c; //printf("root: %d\n",r);

isr[r]=true;

ptr=\_b2; ind=0;

for(int j=adj.l[r];j!=-1;j=E[j].next){

int v=E[j].v;

if( isr[v] ) continue; //

topo\_sort(v,\_b,c);

for(int i=0;i<=c;i++) ptr[i]=inf;

p[v]=PII(black[v],E[j].val); //

cnt++;

for(int i=0;i<c;i++){

int sa=\_b[i];

check(ptr[p[sa].first],p[sa].second);

for(int k=adj.l[sa];k!=-1;k=E[k].next){

int sb=E[k].v;

if( vst[sb]==cnt || isr[sb] ) continue;

p[sb].first=p[sa].first+black[sb];

p[sb].second=p[sa].second+E[k].val;

}

vst[sa]=cnt;

}

for(int i=0;i<=c;i++){

if( i ) check(ptr[i],ptr[i-1]);

if( i+black[r] <=K ) check(ans,ptr[i]);

}

dd[ind].d=ptr, dd[ind++].s=c; ptr+=(c+1);

}

sort(dd,dd+ind);

//merge

for(int i=1;i<ind;i++){

for(int j=0;j<=dd[i].s;j++){

int t=K-j-black[r];

if( t<0 ) continue;

if( t<=dd[i-1].s ) check(ans,dd[i].d[j]+dd[i-1].d[t]);

else check(ans,dd[i].d[j]+dd[i-1].d[ dd[i-1].s ]);

}

for(int j=0;j<=dd[i].s;j++){

if(j<=dd[i-1].s ) check(dd[i].d[j],dd[i-1].d[j]);

if( j ) check(dd[i].d[j],dd[i].d[j-1]);

}

}

for(int j=adj.l[r];j!=-1;j=E[j].next){

int v=E[j].v;

if( !isr[v] ) dfs(v);

}

}

int main(){

//freopen("in.txt","r",stdin);

while( scanf("%d %d %d",&n,&K,&m)!=EOF ){

memset(isr,false,sizeof(isr));

memset(black,false,sizeof(black));

adj.init();

int id;

AdjList::e=0;

for(int i=1;i<=m;i++) scanf("%d",&id), black[id]=true;

for(int i=1;i<n;i++){

int a,b,val; scanf("%d %d %d",&a,&b,&val);

adj.insert(a,b,val);

adj.insert(b,a,val);

}

ans=0;

dfs(1);

cout<<ans<<endl;

}

}

## 树上最长路径,条数

//hdu 3534 树上最长路径,条数

#define maxn 100005

struct Edge{int v,val,x; }E[2\*maxn];

int l[maxn],e;

void init(){ e=0; memset(l,-1,sizeof(l)); }

void insert(int a,int b,int val){

E[e].v=b; E[e].val=val; E[e].x=l[a]; l[a]=e++;

}

bool vst[maxn];

struct PII{ int len,num; }ans;

void insert(PII &p1,PII &p2,PII p3){

if( p3.len==p1.len ){ //需要化归

if( p3.len+p1.len>ans.len ){

ans.len=p3.len+p1.len;

ans.num=p3.num\*p1.num; //溢出?

}else if( p3.len+p1.len==ans.len ){

ans.num+=p3.num\*p1.num;

}

p1.num+=p3.num;

}else if( p3.len>p1.len ){

p2=p1; p1=p3;

}else if( p3.len>=p2.len ){

p2=p3;

}

}

PII dfs(int u){ //<len,num>

vst[u]=true;

PII p1,p2,tmp;

p1.len=p2.len=0;

p1.num=p2.num=1;

for(int i=l[u];i>=0;i=E[i].x){

int v=E[i].v;

if( vst[v] ) continue;

tmp=dfs(v); tmp.len+=E[i].val;

insert(p1,p2,tmp);

}

if( p1.len+p2.len>ans.len ){

ans.len=p1.len+p2.len;

ans.num=p1.num\*p2.num; //溢出?

}else if( p1.len+p2.len==ans.len ){

ans.num+=p1.num\*p2.num;

}

return p1;

}

int main(){

int n,a,b,val;

while( scanf("%d",&n)!=EOF ){

init();

for(int i=1;i<n;i++){

scanf("%d %d %d",&a,&b,&val);

insert(a,b,val);

insert(b,a,val);

}

memset(vst,false,sizeof(vst));

ans.len=-1;

dfs(1);

printf("%d %d\n",ans.len,ans.num);

}

}

## 树套树

/\*

AC: zju2112 支持修改的第k小数 线段树套平衡树

\*/

#define maxn 50005\*20

typedef int type;

struct Node{

type key;

Node\* ch[2]; int size;

friend int SZ(Node\* p){ return p==NULL ? 0 : p->size; }

void init(type \_key=0){ key=\_key; size=1; ch[0]=ch[1]=NULL; }//必须初始化

void update() { size=1+SZ(ch[0])+SZ(ch[1]); }

}dd[maxn];

int e; //初始化为0

Node\* \_alloc(type val){ dd[e].init(val); return &dd[e++]; }

void \_rot(Node\* &p, int f){ //f=0为 往 左旋

Node \*tmp=p->ch[!f];

p->ch[!f]=tmp->ch[f]; tmp->ch[f]=p;

p->update(); tmp->update();

p=tmp;

}

void \_m(Node\* &p, int f){ //维护函数, 检查p->ch[f]是否满足>=p->c[!f]->ch[0,1]

if( !p || !p->ch[!f] ) return;

if( SZ(p->ch[f]) < SZ(p->ch[!f]->ch[!f]) ){ //按照论文

\_rot(p,f);

\_m(p->ch[f],0); \_m(p->ch[f],1);

}else if( SZ(p->ch[f]) < SZ(p->ch[!f]->ch[f]) ) {

\_rot(p->ch[!f],!f); \_rot(p,f);

\_m(p->ch[!f],f); \_m(p->ch[f],!f);

}else return; //

\_m(p,0); \_m(p,1); //保驾护航

}

void insert(Node\* &p,type v){ //插入

if(p==NULL) { p=\_alloc(v); return; }

bool f= (v>p->key);

insert(p->ch[f],v);

p->update(); \_m(p,!f);

}

type \_rev(Node\* &p){

if( p->ch[0] ){

type tmp=\_rev(p->ch[0]);

p->update();

\_m(p,0); return tmp;

}else{

type tmp=p->key; p=p->ch[1]; return tmp;

}

}

void remove(Node\* &p,type key){ //删除,如果有多个则删除一个

if( p==NULL ) return; //删除一个SBT里没有的值

if(key < p->key){

remove(p->ch[0],key); p->update(); \_m(p,0);

}else if(key > p->key){

remove(p->ch[1],key); p->update(); \_m(p,1);

}else{

if( p->ch[0]==NULL) p=p->ch[1];

else if( p->ch[1]==NULL) p=p->ch[0];

else {

p->key=\_rev(p->ch[1]);

p->update(); \_m(p,1);

}

}

}///-----------SBT----------------

int lower\_bound(Node\*p,type key){ //查找排名

if( !p ) return 1;

if( key>p->key ) return SZ(p->ch[0])+1+lower\_bound(p->ch[1],key);

else return lower\_bound(p->ch[0],key);

}

int upper\_bound(Node\*p,type key){

return lower\_bound(p,key+1);

}

type find(Node \*p,int r){ //查询排名为r的元素

if( r<=SZ(p->ch[0]) ) return find(p->ch[0],r);

else r-=SZ(p->ch[0]);

if( r==1 ) return p->key; else r--;

return find(p->ch[1],r);

}

///-----------部分查询函数--------------------------------

int N,M,val[50005];

struct ST{

int L,R;

Node \*root;

}st[50005\*5];

void makeTree(int i,int L,int R){

st[i].L=L; st[i].R=R;

st[i].root=NULL;

for(int j=L;j<=R;j++)

insert(st[i].root,val[j] );

if( L==R ) return;

int mid=(L+R)>>1;

makeTree(i\*2,L,mid);

makeTree(i\*2+1,mid+1,R);

}

void change(int i,int t,int V){

remove(st[i].root,val[t]);

insert(st[i].root,V);

if( st[i].L==st[i].R ){

val[t]=V;

return;

}

int mid=(st[i].L+st[i].R)>>1;

if( t<=mid ) change(i\*2,t,V);

else change(i\*2+1,t,V);

}

int get\_num(int i,int L,int R,int V){ //小于等于V的有几个

if(st[i].L==L && st[i].R==R ){

return upper\_bound(st[i].root,V)-1;

}

int mid=(st[i].L+st[i].R)>>1;

if(R<=mid ) return get\_num(i\*2,L,R,V);

else if( L>mid ) return get\_num(i\*2+1,L,R,V);

else{

return get\_num(i\*2,L,mid,V) +

get\_num(i\*2+1,mid+1,R,V);

}

}

int find(int a,int b,int rank){

int L=0,R=0x7fffffff,mid;

while( L+1 < R ){

mid=(L+R)>>1;

if( get\_num(1,a,b,mid)>=rank ) R=mid;

else L=mid;

}

return R;

}

int main(){

int cas; scanf("%d",&cas);

while( cas-- ){

scanf("%d%d",&N,&M);

for(int i=1;i<=N;i++) scanf("%d",val+i);

e=0;

makeTree(1,1,N);

for(int i=1;i<=M;i++){

char sign; scanf(" %c",&sign);

if( sign=='C' ){

int t,V;

scanf("%d%d",&t,&V);

change(1,t,V);

}else{

int a,b,rank;

scanf("%d%d%d",&a,&b,&rank);

printf("%d\n",find(a,b,rank));

}

}

}

}

## 树同构判定

//树同构判定 pku1635

typedef unsigned int uint;

typedef vector<int> VI;

#define maxn 11000

VI adj[maxn]; //init---

int vst[maxn];

uint shift(uint v,int w){ //循环左移w位

return (v<<w) | (v>>( (sizeof(uint)<<3) -w));

}

uint \_dfs(int u,int shl){

VI h; vst[u]=1;

for(int i=0;i<adj[u].size();i++){

int v=adj[u][i];

if( vst[v] ) continue;

h.push\_back(\_dfs(v,shl));

}

if( h.size()==0 ) return 1;

sort(h.begin(),h.end());

uint ret=0;

for(int i=0;i<h.size();i++){

ret=shift(ret,shl) ^ h[i];

}

return ret;

}

//有根树的hash值, 传入根,左移(用于多次判定),gcd(32,shl)=1

uint hash(int root,int shl){

fill(vst,vst+maxn,0);

return \_dfs(root,shl);

}//\*\*\*\*\*\*\*树的哈希\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//http://acm.fafu.edu.cn/showproblem?problem\_id=1024

//无根树同构,枚举根计算全部的hash值,排序后比较 O(n^2)

int main(){

int n1,n2;

while( scanf("%d%d",&n1,&n2)!=EOF ){

for(int i=1;i<=n1;i++) adj[i].clear();

for(int i=1;i<n1;i++){

int u,v; scanf("%d %d",&u,&v);

adj[u].push\_back(v);

adj[v].push\_back(u);

}

uint h[110],t[110];

for(int i=1;i<=n1;i++) h[i]=hash(i,5);

for(int i=1;i<=n2;i++) adj[i].clear();

for(int i=1;i<n2;i++){

int u,v; scanf("%d %d",&u,&v);

adj[u].push\_back(v);

adj[v].push\_back(u);

}

for(int i=1;i<=n2;i++) t[i]=hash(i,5);

sort(h+1,h+1+n1);

sort(t+1,t+1+n2);

bool same=(n1==n2);

for(int i=1;i<=n1;i++) if( h[i]!=t[i] ) same=false;

if( same ) puts("same");

else puts("different");

}

}

## 图同构判定

/\*

无向图可以看成有向图!!!

有向图同构,输出对应点 pku2040

因为是图,不是树,因此没有拓扑结构,我们只有综合考虑所有的点.

我们采用迭代的思想,每次用第i层所有点的哈希值,计算出第i+1层的哈希值,

为了防止最后趋于稳定,我们选取一个播种源x.

h[i+1][u]=a\*h[i][u] + b\*h[i][v] + c\*h[i][w] + d;

其中u->v,w->u属于边集E,当u==x时d才不为0!!

为了让x的种子遍布整个图(连通吗?),至少迭代n次

\*/

#define maxn 100

#define maxm 30005

typedef unsigned int uint;

struct Edge{ int v,val,next; }E[maxm];

int e;

struct AdjList{

int l[maxn]; //

void insert(int a,int b,int val=0){ //单向边

E[e].v=b; E[e].val=val; E[e].next=l[a]; l[a]=e++;

}

void init(){

memset(l,-1,sizeof(l));

}

}adj[2];

//计算哈希值,0~n-1,播种点,哈希值数组,邻接表,4个参数

void calc(int n,int x,uint \*h,AdjList &adj, uint a,uint b,uint c,uint d){

uint tmp[maxn]={0};

for(int i=0;i<n;i++){

tmp[i]+=a\*h[i];

if( i==x ) tmp[i]+=d;

for(int j=adj.l[i];j>=0;j=E[j].next){

int v=E[j].v;

tmp[i]+=b\*h[v];

tmp[v]+=c\*h[i];

}

}

memcpy(h,tmp,sizeof(uint)\*n);

}

bool same(uint \*d,int n){

for(int i=0;i<n;i++) for(int j=0;j<i;j++) if( d[i]==d[j] ) return true;

return false;

}

int main(){

freopen("in.txt","r",stdin);

freopen("out8.txt","w",stdout);

int n,Te=0;

while(cin>>n && n){

e=0;

map<string,int> mp[2];

string s[2][maxn];

for(int i=0;i<2;i++){

adj[i].init(); mp[i].clear();

for(int j=0;j<n;j++){

string s1,s2;

cin>>s1>>s2;

if( mp[i].find(s1)==mp[i].end() ){

int t=mp[i].size();

mp[i][s1]=t;

}

if( mp[i].find(s2)==mp[i].end() ){

int t=mp[i].size();

mp[i][s2]=t;

}

adj[i].insert(mp[i][s1],mp[i][s2]);

}

for(map<string,int>::iterator it=mp[i].begin();it!=mp[i].end();++it)

s[i][it->second]=it->first;

}//以上建图

int W=mp[0].size(), cnt;

uint h[2][maxn];

pair<string,string> ans[maxn];

for(int x=0;x<W;x++){ //枚举第一副图中0和谁对应

for(int j=0;j<W;j++) h[0][j]=h[1][j]=1; //初始化

int ti=0;

while(true){ //迭代,至少W次, tle??

ti++;

if( ti>=W && !same(h[0],W) && !same(h[1],W) ) break;

calc(W,0,h[0],adj[0],5423,899535389,1467557,5433525);

calc(W,x,h[1],adj[1],5423,899535389,1467557,5433525);

}

cnt=0;

for(int i=0;i<W;i++){ //一一匹配

for(int j=0;j<W;j++){

if( h[0][i]==h[1][j] ){

ans[cnt].first=s[0][i];

ans[cnt].second=s[1][j];

cnt++;

break;

}

}

}

if( cnt==W ) break;

}

if( Te++ ) puts("");

sort(ans,ans+W);

for(int i=0;i<W;i++) {

cout<<ans[i].first<<"/"<<ans[i].second<<'\n';

}

}

}

## 无向图着色问题

/\*

hdu 3762 无向图着色 贪心

思路: 假设现在图可以被k种颜色着色,

那么先考虑度数最小的点u,他肯定可以被着色. 那么u的着色放在最后.

去掉u后,再去找度数最小的,递归下去....

\*/

#define maxn 11000

struct Edge{ int v,x; }E[210000];

int l[maxn],e;

void init(){ e=0; memset(l,-1,sizeof(l)); }

inline void insert(int u,int v){

E[e].v=v; E[e].x=l[u]; l[u]=e++;

}

struct DD{

int d,v; //degree, vertex

DD(int \_d=0,int \_v=0):d(\_d),v(\_v){};

bool operator<(const DD &b)const{

return d > b.d;

}

};

priority\_queue<DD> pq;

int n,m;

int deg[maxn],col[maxn],tmp[maxn];

int o[maxn]; //贪心的顺序

bool vst[maxn];

void paint(){

int k=0,oid=0; //k种颜色

for(int i=1;i<=n;i++) {

col[i]=-1; vst[i]=false; tmp[i]=deg[i];

pq.push(DD(tmp[i],i));

k=max(k,deg[i]);

}

if( k%2==0 ) k++; printf("%d\n",k);//题目规定是奇数

while( !pq.empty() ){

DD d=pq.top(); pq.pop();

if( tmp[ d.v ]!=d.d ) continue;

o[oid++]=d.v; vst[d.v]=true;

for(int p=l[d.v];p>=0;p=E[p].x){

int v=E[p].v;

if( vst[v] ) continue; //已经取出来了

tmp[v]--;

pq.push(DD(tmp[v],v));

}

}

for(int i=oid-1;i>=0;i--){ //从后往前

int u=o[i],s;

fill(vst,vst+min(k+1,deg[u]+2),false); //暴力染色

for(int p=l[u];p>=0;p=E[p].x){

int v=E[p].v;

if( col[v]>=0 ) vst[ col[v] ]=true;

}

for(s=1;vst[s];s++);

col[u]=s; //if( s>k ) while(1) puts("f");

}

}

int main(){

int cas=1; scanf("%d",&cas);

while( cas-- ){

scanf("%d %d",&n,&m);

init();

memset(deg,0,sizeof(deg));

for(int i=1;i<=m;i++){

int u,v; scanf("%d %d",&u,&v);

deg[u]++; deg[v]++;

insert(u,v); insert(v,u);

}

paint();

for(int i=1;i<=n;i++) printf("%d\n",col[i]);

}

}

## 数独, 各种大小

//数独 pku3076 pku3074

#define MAXR 16\*16\*16//row

#define MAXC 16\*16\*4//column

#define TOTAL (MAXR+1)\*(MAXC+1)

using namespace std;

int mat[MAXR][MAXC];//初始矩阵，0或1，从1，1开始用

int cid[TOTAL],rid[TOTAL],s[MAXC];

int l[TOTAL],r[TOTAL],u[TOTAL],d[TOTAL];

int lastc[MAXC];//第i列的最下面一个元素

void remove(const int &col){

l[r[col]]=l[col];

r[l[col]]=r[col];

for(int i=d[col];i!=col;i=d[i]){ //枚举每一行

for(int j=r[i];j!=i;j=r[j]){ //枚举每一列

d[u[j]]=d[j];

u[d[j]]=u[j];

s[ cid[j] ]--;

}

}

}

void resume(const int &col){ //回溯

for(int i=u[col];i!=col;i=u[i]){

for(int j=l[i];j!=i;j=l[j]){

d[u[j]]=j;

u[d[j]]=j;

s[ cid[j] ]++;

}

}

l[r[col]]=col;

r[l[col]]=col;

}

inline void addlr(const int &ln,const int &rn){ //完了后还是成环的

l[ r[ln] ]=rn; r[rn]=r[ln];

r[ln]=rn; l[rn]=ln;

}

inline void addud(const int &un,const int &dn){

u[ d[un] ]=dn; d[dn]=d[un];

d[un]=dn; u[dn]=un;

}

int res[MAXR],ind;//每次取的哪行

bool search(int k){ //将开始第几层搜索

if(r[0]==0){

ind=k;

return true;//找到一组解

}

int c,mi=100000000;

for(int i=r[0];i!=0;i=r[i]){

if(s[i]<=mi){ //诡异，要<=,列的顺序是行->列->宫->格子，估计是后面的宫的那个限制条件卡的紧一些，所有先选后面的会好些

mi=s[i]; c=i;

}

}

if(mi==0) return false; //可以不要，包括在下面这个for里的

remove(c);

for(int i=d[c];i!=c;i=d[i]){

res[k]=rid[i];//选择了哪一行，这个数组可以用来输出选择了那些东东

for(int j=r[i];j!=i;j=r[j]) remove(cid[j]);

if(search(k+1)) return true;

for(int j=l[i];j!=i;j=l[j]) resume(cid[j]);

}

resume(c);

return false;

}

/\* 链表的形状类似这样

#######

. . ..

... .

. ..

\*/

void adapt(int n,int m){ //传入01矩阵mat的规格,适配函数（由mat构造dancing link 表）

memset(s,0,sizeof(s));//每列元素个数清0

l[0]=r[0]=0;

for(int j=1;j<=m;j++){

addlr(j-1,j);

u[j]=d[j]=lastc[j]=j;

}

int c=m+1;//从m+1开始用

for(int i=1;i<=n;i++){

bool f=true;

for(int j=1;j<=m;j++){

if(mat[i][j]){

if(f){

f=false;

l[c]=r[c]=c;

}else{

addlr(c-1,c);

}

addud(lastc[j],c);

rid[c]=i; cid[c]=j;

s[j]++; lastc[j]=c++;

}

}

}

}//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*模板结束\*\*\*\*\*\*\*\*\*\*\*

const int N=3; //数独大小是(N^2)\*(N^2) N可以是2,3,4

const int L=9; //N^2

const int b0=0,b1=L\*L,b2=2\*L\*L,b3=3\*L\*L; //base

inline int block(int i,int j){

return (i-1)/N\*N+(j-1)/N+1;

}

struct QQ{

int r,c,val;

void paint(int i){

for(int j=1;j<=L\*L\*4;j++) mat[i][j]=0; //初始化

int off;

off=(r-1)\*L+val; mat[i][b0+off]=1;

off=(c-1)\*L+val; mat[i][b1+off]=1;

off=(block(r,c)-1)\*L+val; mat[i][b2+off]=1;

off=(r-1)\*L+c; mat[i][b3+off]=1;

}

}qq[16\*16\*16+5];

int main(){

int Te=0;

char str[22][22];

while( scanf(" %9s",str[1]+1)!=EOF ){

if( str[1][1]=='e' ) break;

for(int i=2;i<=L;i++) scanf("%9s",str[i]+1);

int R=0;

for(int i=1;i<=L;i++){

for(int j=1;j<=L;j++){

if( str[i][j]=='.'){

for(int k=1;k<=L;k++) {

qq[++R].r=i, qq[R].c=j, qq[R].val=k;

qq[R].paint(R);

}

}else{

qq[++R].r=i, qq[R].c=j, qq[R].val=str[i][j]-'0';

qq[R].paint(R);

}

}

}

adapt(R,L\*L\*4);

search(0);

for(int i=0;i<ind;i++){

int d=res[i];

str[ qq[d].r ][ qq[d].c ]='0'+qq[d].val;

}

for(int i=1;i<=L;i++) for(int j=1;j<=L;j++) putchar(str[i][j]);

puts("");

}

}

## 最优比率生成树

//pku 2728 最小比率生成树 Dinkelbach 参数迭代逼近

//稠密图不要用priority\_queue，直接用一个for循环找到最小（大）值

//对边的选择\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//r = ∑(benifit[i] \* x[i]) / ∑(cost[i]\*x[i]), 0≤i<m .为了使 r 最大,

//r就是要求的参数\*\*\*\*\*\*\*\*\*\*\*\*\*构造另外一个函数来求它\*\*\*\*\*\*\*\*\*\*\*\*

//设计一个子问题---> 让 z = ∑(benifit[i]\*x[i]) - k\*∑(cost[i] \* x[i])

//= ∑(d[i] \* x[i]) 最大 (d[i] = benifit[i] - k\* cost[i]) , 并记为z(k).

//我们可以兴高采烈地把z(l)看做以d为边权的最大生成树的总权值.

#define MAXN 1005

using namespace std;

struct point{

double x,y,z;

}p[MAXN];

struct T{

double height,dis;

}t[MAXN];

double h[MAXN],d[MAXN],key[MAXN];

inline double distance(double &x1,double &y1,double &x2,double &y2){

return sqrt( (x1-x2)\*(x1-x2)+(y1-y2)\*(y1-y2) );

}

struct Node{

int id;

double key;

Node(int \_id=0,double \_key=0.0):id(\_id),key(\_key){};

};

inline bool operator<(const Node &a,const Node &b){

return a.key>b.key;

}

int N;

bool visit[MAXN];

void prim(double r){

priority\_queue<Node> pq;

memset(visit,false,sizeof(visit));

for(int i=1;i<=N;i++)

key[i]=1000000000.0;

pq.push(Node(1,0.0));

int ind=1;

bool flag=true;

double hh,dd;

while(!pq.empty()){

int id=pq.top().id;

pq.pop();

if(visit[id])

continue;

visit[id]=true;

if(flag)

flag=false;

else{

t[ind].dis=d[id];

t[ind++].height=h[id];

}

for(int i=1;i<=N;i++)

if(!visit[i]){

hh=fabs(p[id].z-p[i].z);

dd=distance(p[id].x,p[id].y,p[i].x,p[i].y);

if(hh-r\*dd<key[i]){//构造出来的函数

h[i]=hh;

d[i]=dd;

key[i]=h[i]-r\*d[i];

pq.push(Node(i,key[i]));

}

}

}

}

int main(){

double r,temp;

while(scanf("%d",&N)!=EOF&&N){

for(int i=1;i<=N;i++)

scanf("%lf%lf%lf",&p[i].x,&p[i].y,&p[i].z);

r=0.0;

while(1){

prim(r); //用一个参数去解出一个解向量

double m=0.0;

for(int i=1;i<N;i++)

m+=t[i].dis;

temp=0.0; //计算这次的参数值

for(int i=1;i<N;i++)

temp+=t[i].height/m;

if(fabs(temp-r)<1e-4) //前后两个参数的差距很小的时候

break;

r=temp;

}

printf("%.3lf\n",r);

}

}

/\* 稠密图用下面这个更快

void prim(double r){

for(int i=1;i<=N;i++)

key[i]=1000000000.0;

memset(visit,false,sizeof(visit));

key[1]=0.0;

int ind=1,index;

double mi;

bool flag=true;

for(int i=1;i<=N;i++){

mi=1000000006.0;

for(int i=1;i<=N;i++)

if(!visit[i]&&key[i]<mi)

mi=key[i],index=i;

visit[index]=true;

if(flag)

flag=false;

else{

t[ind].dis=d[index];

t[ind++].height=h[index];

}

for(int i=1;i<=N;i++)

if(!visit[i]&&(fabs(p[index].z-p[i].z)-r\*distance(p[index].x,p[index].y,p[i].x,p[i].y)<key[i])){

h[i]=fabs(p[index].z-p[i].z);

d[i]=distance(p[index].x,p[index].y,p[i].x,p[i].y);

key[i]=h[i]-r\*d[i];

}

}

}

//pku 2728 最小比率生成树 二分参数搜索 2000ms

//稠密图不要用priority\_queue，直接用一个for循环找到最小（大）值

//构造出来的那个函数是单调的，先取左边的l（代入是正值），再取一个右边的值r（代入是负值）

//每次把mid代入，解出一个解向量X，再缩小l和r的间距了，知道fabs(l-r)<1e-n........

#define MAXN 1005

using namespace std;

struct point{

double x,y,z;

}p[MAXN];

struct T{

double height,dis;

}t[MAXN];

double h[MAXN],d[MAXN],key[MAXN];

inline double distance(double &x1,double &y1,double &x2,double &y2){

return sqrt( (x1-x2)\*(x1-x2)+(y1-y2)\*(y1-y2) );

}

int N;

bool visit[MAXN];

void prim(double r){

for(int i=1;i<=N;i++)

key[i]=1000000000.0;

memset(visit,false,sizeof(visit));

key[1]=0.0;

int ind=1,index;

double mi;

bool flag=true;

for(int i=1;i<=N;i++){

mi=1000000006.0;

for(int i=1;i<=N;i++)

if(!visit[i]&&key[i]<mi)

mi=key[i],index=i;

visit[index]=true;

if(flag)

flag=false;

else{

t[ind].dis=d[index];

t[ind++].height=h[index];

}

for(int i=1;i<=N;i++)

if(!visit[i]&&(fabs(p[index].z-p[i].z)-r\*distance(p[index].x,p[index].y,p[i].x,p[i].y)<key[i]))

{

h[i]=fabs(p[index].z-p[i].z);

d[i]=distance(p[index].x,p[index].y,p[i].x,p[i].y);

key[i]=h[i]-r\*d[i];

}

}

}

double cal(double k){

double m=0;

for(int i=1;i<=N;i++)

m+=t[i].height-k\*t[i].dis;

return m;

}

int main(){

while(scanf("%d",&N)!=EOF&&N){

for(int i=1;i<=N;i++)

scanf("%lf%lf%lf",&p[i].x,&p[i].y,&p[i].z);

double l=0.0,r=100.0,mid;

while(1){

mid=(l+r)/2.0;

prim(mid);//代入求下一个解向量X

double ttt=cal(mid);//代入mid解出构造出的函数的值

if(fabs(l-r)<1e-6)

break;

if(ttt>=0)

l=mid;

else

r=mid;

}

printf("%.3lf\n",r);

}

}

## 树的删边游戏

//pku 3710 多棵树,树上有环 树的删边游戏

#define maxn 105

struct Edge{

int v,next;

}E[4\*maxn];

int list[maxn],eid;

bool visN[maxn],visM[4\*maxn];//点访问,边访问

bool inner[maxn]; //是否是环的内部点,不包括环和树的公共点

int st[maxn],top; //节点的访问栈

void init(){

memset(list,-1,sizeof(list));

memset(visN,false,sizeof(visN));

memset(visM,false,sizeof(visM));

memset(inner,false,sizeof(inner));

eid=0; top=0;

}

void insert(int a,int b){

E[eid].v=b; E[eid].next=list[a]; list[a]=eid++;

E[eid].v=a; E[eid].next=list[b]; list[b]=eid++;

}

int dfs(int u){

visN[u]=true;

int nim=0;

st[++top]=u;

for(int p=list[u];p!=-1;p=E[p].next){

if( visM[p] ) continue;

visM[p]=visM[p^1]=true; //

int ret,v=E[p].v;

if( !visN[v] ){

ret=dfs(v);

}else{

while( st[top]!= v){

inner[st[top--]]=true;

}

return 1;

}

if( inner[v] ) nim^=(!ret);

else nim^=(ret+1);

}

return nim;

}

int main(){

int Tree,nim;

while( scanf("%d",&Tree)!=EOF ){ //几棵树

nim=0;

for(int T=1;T<=Tree;T++){

init();

int N,M;

scanf("%d%d",&N,&M);

for(int i=1;i<=M;i++){

int a,b; scanf("%d%d",&a,&b);

insert(a,b);

}

nim^=dfs(1);

}

//cout<<nim<<endl;

if( nim ) puts("Sally");

else puts("Harry");

}

}

## 四边形不等式dp

状态转移方程形如:

dp[i][j]=min{ dp[i][k]+dp[k+1][j]+w[i][j] }

然后对于 k1 < k2 < k3 < k4 有:

w[k2][k3] <= w[k1][k4] //区间包含单调性

w[k1][k3] + w[k2][k4] <= w([k2][k3] + w[k1][k4] //四边形不等式

//hdu 3516 Tree construction 四边形不等式

/\*

dp[i][j]=min{ dp[i][k]+dp[k+1][j] + x[k+1]-x[i] + y[k]-y[j] } (i<=k<j ) \*\*\*注意变量范围

令dp[i][j]+x[i]+y[j] = m[i][j]

则m[i][j]=min{ m[i][k]+m[k+1][j] - (x[i]+y[j]) } (i<=k<j) 发现权值和k无关了\*\*\*\*

\*/

#include<iostream>

using namespace std;

int m[1005][1005],sel[1005][1005];

int x[1005],y[1005];

int main(){

int n;

while( scanf("%d",&n)!=EOF ){

for(int i=1;i<=n;i++) scanf("%d%d",x+i,y+i);

for(int i=1;i<=n;i++) {

m[i][i]=x[i]+y[i];

sel[i][i]=i;

}

for(int len=2;len<=n;len++){

for(int i=1;i+len-1 <=n ;i++){

int j=i+len-1;

int mi=1000000000,s;

//这里k<j一定要加上去,推导必须是严谨的

for(int k=sel[i][j-1];k<=sel[i+1][j] && k<j;k++){

if( m[i][k] + m[k+1][j] < mi){

mi=m[i][k]+m[k+1][j];

s=k;

}

}

m[i][j]=mi-x[i]-y[j];

sel[i][j]=s;

}

}

int ans=m[1][n]-x[1]-y[n];

cout<<ans<<endl;

}

}

## 第k个square-free number

//http://cs.scu.edu.cn/soj/problem.action?id=3191

//求第k个square-free number(所有质因数幂次为1)

//算法,二分,然后用容斥判断

//使用容斥,关键是先定好n个条件,也就是所谓的n个圈圈,然后求n个圈圈的并集....

typedef pair<int,int> PII;

vector<PII> express(int n){ //复杂度O(sqrt(n))

vector<PII> ret;

for(int i=2;i\*i<=n;i++){

if( n%i==0 ){

int num=0;

while( n%i==0 ) num++,n/=i;

ret.push\_back(PII(i,num) );

}

}

if( n!=1 ){

ret.push\_back(PII(n,1) );

}

return ret;

}

vector<PII> buf[50005];

int squre\_free(int n){ //<=n的square-free有几个

int minus=0;

for(int i=2;i\*i<=n;i++){

bool can=true;

for(int j=0;j<buf[i].size();j++){

if( buf[i][j].second >1 ) can=false;

}

if( can==false ) continue;

if( buf[i].size()%2 ) minus+=n/(i\*i);

else minus-=n/(i\*i);

}

return n-minus;

}

int main(){

for(int i=2;i<=50000;i++) buf[i]=express(i);

int cas,r; cin>>cas;

while( cas-- ){

cin>>r;

int L=0,R=1644934085;

while( L+1 < R ){

int mid=(L+(long long)R)/2;

if( squre\_free(mid)>= r ) R=mid;

else L=mid;

}

cout<<R<<endl;

}

}

## 高斯解模方程组

/\*

高斯解模方程组

hdu 3571 N-dimensional Sphere

求N维空间的N+1个点的中心,类似于一维的中点,二维的圆心,三维的球心~~

设结果为(x1,x2,...,xN)

N维空间的两个点为(a1,a2,...,aN),(b1,b2,...,bN)

由于距离相等,那么(x1-a1)^2 + ... + (xN-aN)^2 == (x1-b1)^2 + ... + (xN-bN)^2

减掉右边的,就变成了2\*(b1-a1)x1+2\*(b2-a2)x2+...+2\*(bN-aN)xN=(b1^2-a1^2) + ... + (bN^2-aN^2)

这样子N+1个点就联立了N个线性无关的方程,问题转化为解这么一个N元方程组

问题在于所有数的范围是10^17,因此double精度不够,那么就转化成一个模方程组好了,mod取比10^17大的整数

\*/

#include<iostream>

#include<cassert>

#include<algorithm>

using namespace std;

#define maxn 55

typedef long long LL;

const LL mod=200000000000000003ll;

inline void fix(LL &a){

if( a<=-mod ) a+=mod;

if( a>=mod ) a-=mod;

}

LL mul(LL a,LL b,LL mod){ //b为正数,否则死循环

LL ret=0;

bool flag=false; if( b<0 ){ b=-b; flag=true; }

while( b ){

if( b&1 ) ret=(ret+a),fix(ret);

a<<=1; fix(a);

b>>=1;

}

if( flag ) return -ret;

return ret;

}

void e\_gcd(LL a,LL b,LL &x,LL &y) {

if( !b ){

x=1; y=0;

}else {

e\_gcd(b,a%b,x,y);

LL temp=x;

x=y;

y=temp-mul(a/b,y,mod);

}

}

LL get\_ni(LL a,LL p) {//返回x,ax=1 (mod p) p是素数

LL x,y;

e\_gcd(a,p,x,y);

return x;

}

/\*

c[1][1]\*x[1]+c[1][2]\*x[2]+...+c[1][n]\*x[n]=b[1]

c[2][1]\*x[1]+c[2][2]\*x[2]+...+c[2][n]\*x[n]=b[2]

....

c[n][1]\*x[1]+c[n][2]\*x[2]+...+c[n][n]\*x[n]=b[n]

\*/

LL c[maxn][maxn],x[maxn],b[maxn];//c系数矩阵,x结果向量,b是c\*x的结果

void gauss(int N){ //高斯解模方程组C\*X=B,N\*N的矩阵c

for(int i=1,k;i<=N;i++){

if( c[i][i]==0 ){

LL ma=0;

for(int j=i;j<=N && ma==0;j++){

if( c[j][i]!=ma ){

ma=c[j][i];

k=j;

}

}

for(int j=i;j<=N;j++) swap(c[i][j],c[k][j]);

swap(b[i],b[k]);

}

if( c[i][i]<0 ){

for(int j=i;j<=N;j++) c[i][j]=-c[i][j];

b[i]=-b[i];

}

assert(c[i][i]!=0);//\*\*\*\*\*等于0的话无解了

for(int k=i+1;k<=N;k++){

if( c[k][i]==0 ) continue;

LL m=get\_ni(c[i][i],mod);

m=mul(m,c[k][i],mod);

for(int j=i;j<=N;j++){

c[k][j]-=mul(m,c[i][j],mod);

fix(c[k][j]);

}

assert(c[k][i]==0);

b[k]-=mul(m,b[i],mod);

fix(b[k]);

}

}

for(int i=N;i>=1;i--){

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

b[i]-=mul(c[i][j],x[j],mod);

fix(b[i]);

}

x[i]=mul(b[i],get\_ni(c[i][i],mod),mod);

}

}//--------------------模板结束--------------------

LL a[maxn][maxn];

void process(int N){

for(int i=1;i<=N;i++){

LL temp=0;

for(int j=1;j<=N;j++){

temp=(temp+mul(a[i][j],a[i][j],mod)-mul(a[i-1][j],a[i-1][j],mod));

fix(temp);

c[i][j]=(a[i][j]+a[i][j]-a[i-1][j]-a[i-1][j]);

fix(c[i][j]);

}

b[i]=temp;

}

}

void change(LL &val,int id,int N){

LL temp[55];

int ind=0;

temp[ind++]=val;

for(int i=0;i<=N;i++) temp[ind++]=a[i][id];

sort(temp,temp+ind);

if( temp[0]==val && temp[1]>val ){

val+=mod;

}

else if( temp[ind-1]==val && temp[ind-2]<val ){

val-=mod;

}

}

int main(){

int cas,N; scanf("%d",&cas);

for(int Te=1;Te<=cas;Te++){

scanf("%d",&N);

for(int i=0;i<=N;i++)

for(int j=1;j<=N;j++){

scanf("%I64d",&a[i][j]);

}

process(N);

gauss(N);

for(int i=1;i<=N;i++) change(x[i],i,N);

printf("Case %d:\n",Te);

for(int i=1;i<=N;i++) printf("%I64d%s",x[i],i==N?"\n":" ");

}

}

## 8数码问题

A\*算法

struct PII{

int r,c;

PII(int \_r=0,int \_c=0):r(\_r),c(\_c){};

}; //用于存放坐标的

struct EIGHT{ //数码的状态类

int \_d[3][3];

PII \_z; //0的坐标

int g,h;

vector<int> path; //搜到这个状态的路径

int getRev()const{ //逆序数

int a[8],ind=0,i,j,Rev=0;

for(i=0;i<3;i++){

for(j=0;j<3;j++){

if( \_d[i][j]!=0 )

a[ind++]=\_d[i][j];

}

}

for(i=1;i<8;i++){

for(j=0;j<i;j++)

if( a[j] > a[i] )

Rev++;

}

return Rev;

}

bool getNext(int dir,EIGHT &ns){//移动，返回移动是否成功(下一个状态在引用中传递)

memcpy(ns.\_d,\_d,sizeof(\_d));

ns.path=path; ns.\_z=\_z;

int r=\_z.r, c=\_z.c, nr,nc;

if( dir==0 ){ // down

if( \_z.r==2 ) return false;

nr=r+1, nc=c;

}else if( dir==1 ){ // left

if( \_z.c==0 ) return false;

nr=r, nc=c-1;

}else if( dir==2 ){ // right

if( \_z.c==2 ) return false;

nr=r, nc=c+1;

}else{ // up

if( \_z.r==0 ) return false;

nr=r-1, nc=c;

}

swap(ns.\_d[r][c],ns.\_d[nr][nc]);

ns.\_z=PII(nr,nc);

ns.path.push\_back(dir);

return true;

}

int getH(EIGHT &des){ //估价函数2

PII b[8];

for(int i=0;i<3;i++){

for(int j=0;j<3;j++){

if( \_d[i][j] )

b[ \_d[i][j]-1 ]=PII(i,j);

}

}

int ret=0;

for(int i=0;i<3;i++){

for(int j=0;j<3;j++){

if( des.\_d[i][j] ){

int v=des.\_d[i][j];

ret+=abs(i-b[v-1].r)+abs(j-b[v-1].c);

}

}

}

return ret;

}

/\*int getH(EIGHT &des){ //估价函数1

int ret=0;

for(int i=0;i<3;i++){

for(int j=0;j<3;j++){

if( \_d[i][j]==0 ) continue;

if( \_d[i][j] != des.\_d[i][j] ){ //不在位数

ret++;

}

}

}

return ret;

}\*/

void init(int \*d){ //用数组初始化

for(int i=0;i<9;i++){

\_d[i/3][i%3]=d[i];

if( d[i]==0 ){

\_z.r=i/3, \_z.c=i%3;

}

}

this->path.clear();

}

int getHash(){ //全排列的完美哈希,复杂度O(n^2),判重用的

int a[9];

for(int i=0;i<3;i++){

for(int j=0;j<3;j++) a[3\*i+j]=\_d[i][j];

}

int ret=0,mul=1;

for(int i=1;i<9;i++){

int rev=0;

for(int j=0;j<i;j++) if( a[j] > a[i] ) rev++;

ret+=mul\*rev;

mul\*=(i+1);

}

return ret;

}

void print(int step){ //打印8数码

printf("step %d:\n%d %d %d\n%d %d %d\n%d %d %d\n",step,

\_d[0][0],\_d[0][1],\_d[0][2],\_d[1][0],\_d[1][1],

\_d[1][2],\_d[2][0],\_d[2][1],\_d[2][2]);

}

bool operator<(const EIGHT &b)const{ //优先队列中的比较

return g+h > b.g+b.h;

}

bool operator==(const EIGHT &b)const{ //判相等,这里只要3\*3的board相等就ok了

return memcmp(\_d,b.\_d,9\*sizeof(int)) == 0;

}

}src,des;

priority\_queue< EIGHT > open;

//queue< EIGHT > close;

void trace(vector<int> path){

EIGHT s=src,t;

int step=0;

s.print(step++);

for(int i=0;i<(int)path.size();i++){

cout<<"--->"<<endl;

s.getNext(path[i],t);

t.print(step++); s=t;

}

}

bool hash[500000];

void Astar(){

if( src.getRev()%2 != des.getRev()%2 ){//判解

puts("unSolvable");

return;

}

vector<int> path;

fill(hash,hash+500000,false);

while( !open.empty()) open.pop();

//while( !close.empty()) close.pop();

src.g=0,src.h=src.getH(des);

open.push(src);

int searched=0, visited=0;

while( true ){

EIGHT cur=open.top(),nxt;

open.pop();

if( cur==des ){

puts("I find it!");

path=cur.path;

break;

}

for(int i=0;i<4;i++){ //0往4个方向移动

if( cur.getNext(i,nxt) ){

nxt.g=0;

nxt.g=cur.g+1;

nxt.h=nxt.getH(des);//改成nxt.g=0就是普通的广搜了

//nxt.h=0;

int hh=nxt.getHash();

if( hash[ hh ] ) continue;

hash[ hh ]=true;

open.push(nxt);

visited++; //访问过的节点个数+1

}

}

searched++; //搜索过的节点个数+1

//cout<<searched<<endl;

}

trace(path); //打印搜索的路径

printf("最短路径%d,访问了%d个状态,搜索了%d个状态\n",path.size(),visited,searched);

}

int main(){

int d[9]={1,2,3,8,0,4,7,6,5};

des.init(d);

while( cin>>d[0] ){

for(int i=1;i<9;i++) cin>>d[i];

src.init(d);

Astar();

}

}

/\*

2 1 0

3 4 5

6 7 8

I find it!

最短路径24,访问了2368个状态,搜索了1472个状态

\*/

## 原根,计数

//fzu2031

//求1~P内有多少对整数x<=y,有x^x+y^y=0 (mod P), P是素数

/\*

原根Primitive Root。

　　设m是正整数，a是整数，若a模m的阶等于φ(m)，则称a为模m的一个原根。（其中φ(m)表示m的欧拉函数）

　　假设一个数g对于P来说是原根，那么g^i mod P的结果两两不同,且有 1<g<P, 0<i<P,那么g可以称为是P的一个原根,归根到底就是g^(P-1) = 1 (mod P)当且当指数为P-1的时候成立.(这里P是素数).

　　简单来说，g^i mod p ≠ g^j mod p （p为素数）

　　其中i≠j且i, j介於1至(p-1)之间

　　则g为p的原根。

　　求原根目前的做法只能是从2开始枚举，然后暴力判断g^(P-1) = 1 (mod P)是否当且当指数为P-1的时候成立

　　而由于原根一般都不大，所以可以暴力得到.

\*/

#define maxn 1000006

int val[maxn];

int rm[maxn];

int ge[maxn];

int P;

int getPrimitiveRoot(int P){//必须是奇素数才有原根

for(int i=2;;i++){

int rem=1;

bool suc=true;

for(int j=1;j<P-1;j++){

rem=(long long)rem\*i%P;

if( rem==1 ){

suc=false;

break;

}

}

if( suc ) return i;

}

}

/\*

由于P是奇素数，所以Inv(i) -> j, 其中j是 [1.. P-1]内的一个数字

并且Inv()这个函数是一一映射的：

对于一个j, 有且只存在一个i,使得Inv(i) = j ( mod P)

对于一个i, Inv(i) = j (mod P), 只对应一个j

(证明用原根和欧拉定理)

\*/

int main(){

int cas;

scanf("%d",&cas);

while( cas-- ){

scanf("%d",&P);

if( P==2 ){

cout<<1<<endl;

continue;

}

int r=getPrimitiveRoot(P),rem=1;

for(int i=1;i<=P;i++) ge[i]=0;

//continue;

for(int i=1;i<P;i++){

rem=(long long)rem\*r%P;

val[i]=rem;

rm[rem]=i;

//cout<<rem<<' '<<i<<endl;

//mi[rem]=i;

int t=(long long)i\*rem%(P-1);

//cout<<"t: "<<t<<endl;

if( t==0 )t=P-1;

ge[t]++;

}

int ans=0;

for(int i=1;i<P;i++){

///if( i>P/2 ) break;

//if( ge[i] && ge[P-i])

ans+=ge[i]\*ge[ rm[ P-val[i] ] ];

}

printf("%d\n",ans/2);

}

}

Java常用

## IO

1, 控制台IO

//输入

Scanner cin = new Scanner(new BufferedInputStream(System.in));

Scanner cin = new Scanner(System.in);

int n = cin.nextInt(); 相当于 scanf("%d", &n);

String s = cin.next(); 相当于 scanf("%s", s);

double t = cin.nextDouble(); 相当于 scanf("%lf", &t);

String s = cin.nextLine(); 相当于 gets(s);

判断是否有下一个输入可以用 cin.hasNext()

//输出

PrintWriter cout = new PrintWriter(new BufferedOutputStream(System.out));

System.out.println(s);

2, 文件 IO

try

{

Scanner reader = new Scanner( new File("in.txt") );//读入

PrintWriter out = new PrintWriter( "out.txt");//写入

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*下面是处理部分

int a;

while( reader.hasNext())

{

a=reader.nextInt();

System.out.println(a);

out.println(a);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

out.close();//保存文件

}

catch(IOException e)

{}

3, 快一点的IO

PS: 有更快的 IO 方法，在大量输入时，可快10倍以上

BufferedReader cin = new BufferedReader(new InputStreamReader(System.in));

如果需要 读到文件尾，可：

tree=cin.readLine();

while(tree!=null) {

}

BufferedReader cin = new BufferedReader( new FileReader("test.in") );

默认的文件地址在，工程下

## 常用算法

Arrays.fill() 相当于memset

Arrays.sort() 相当于sort

## 排序

排序:

class Point implements Comparable//必须加 implements Comparable

{

public int x;

public int compareTo(Object obj)

{

Point b=(Point)obj;

if( this.x < b.x ) return -1;

if( this.x > b.x ) return 1;

return 0;

}

}

Arrays.sort(p, 0 , n); //快速排序 0~n-1

## BigInteger开根号

static public BigInteger sqrt(BigInteger n)

{

BigInteger L=BigInteger.valueOf(0) , R=n , M;

if( L.multiply(L).compareTo(n)==0) return L;

if( L.multiply(L).compareTo(n)==0) return R;

while( L.add(BigInteger.valueOf(1)).compareTo(R)<0 )

{

M=L.add(R).divide(BigInteger.valueOf(2));

if( M.multiply(M).compareTo(n)<=0 ) L=M;

else R=M;

}

return L;

}

## java计算时间差

//java计算时间差,不要导入util包

long startT=new Date().getTime();

int count=1000000000;

while( count!=0 ) count--;

long endT=new Date().getTime();

## HashMap TreeMap

import java.io.\*;

import java.util.\*;

import java.math.\*;

import java.lang.\*;

// Map类下有 HashMap TreeMap 等等...

class ACM

{

public static HashMap<String,Integer> HM=

new HashMap<String,Integer>();

public static HashMap<Integer,BigInteger> KK;

public static TreeMap<String,Integer> TM=

new TreeMap<String,Integer>();

//遍历Map

public static void PrintMap(Map mp)

{

Iterator it=mp.entrySet().iterator();

while(it.hasNext())

{

Map.Entry entry=

(Map.Entry) it.next();

System.out.println(entry.getKey()+" "+entry.getValue());

}

/\*

\*

Iterator it=TM.entrySet().iterator();

while(it.hasNext())

{

Map.Entry<String, Integer> entry=

(Map.Entry<String, Integer>)it.next();

System.out.println(entry.getKey()+" "+entry.getValue());

}

\*/

}

public static void TestTreeMap()

{

System.out.println("Begin to Test TreeMap");

TM.put("dd", 4);

TM.put("aa", 99);

TM.put("bb", 44);

PrintMap(TM);

}

public static void TestHashMap()

{

System.out.println("Begin to Test HashMap");

HM.put("dd", 4);

HM.put("aa", 99);

HM.put("bb", 44);

PrintMap(HM);

}

public static void main(String[] args)

{

TestTreeMap();

TestHashMap();

//移除

HM.remove("aa");

if( HM.isEmpty() ) System.out.println("empty");

if( HM.containsKey("dd") )

System.out.println("the value of dd is "+HM.get("dd"));

PrintMap(HM);

}

}

//TreeMap用法 pku2418

import java.io.\*;

import java.util.\*;

class Main{

public static void main(String[] args)throws IOException{

//Scanner reader=

// new Scanner(new BufferedInputStream(System.in));

BufferedReader reader=

new BufferedReader(new InputStreamReader(System.in));

TreeMap<String,Integer> mp=

new TreeMap<String,Integer>();

int Num=0;

//System.out.println("ddd");

while( true ){

String s=reader.readLine(); if( s==null ) break; //

if( mp.containsKey(s) ){

Map.Entry<String,Integer> entry=mp.ceilingEntry(s);

mp.put(entry.getKey(), entry.getValue()+1);

}else{

mp.put(s, 1);

}

Num++;

}

Iterator<Map.Entry<String, Integer> > it=mp.entrySet().iterator();

while(it.hasNext()){

Map.Entry<String, Integer> entry=it.next();

System.out.printf("%s %.4f\n",entry.getKey(),100.0\*entry.getValue()/Num );

}

reader.close();

}

}

## ext\_gcd求逆元

//java ext\_gcd求逆元

int[] ext\_gcd(int a,int b){ //int[] -> (x,y,gcd(a,b) )

if( b==0 ){

int[] r=new int[3];

r[0]=1; r[1]=0; r[2]=a;

return r;

}else{

int[] r=ext\_gcd(b,a%b);

int tmp=r[0]; r[0]=r[1]; r[1]=tmp-a/b\*r[1];

return r;

}

}

int get\_ni(int a,int p){ //p is prime

int[] r=ext\_gcd(a,p); return r[0];

}

import java.text.\*;

## 输出格式

//格式.....这里0指一位数字，#指除0以外的数字。

import java.text.DecimalFormat;

DecimalFormat f = new DecimalFormat("#.00#");

DecimalFormat g = new DecimalFormat("0.000");

double a = 123.45678, b = 0.12;

System.out.println(f.format(a));

System.out.println(f.format(b));

System.out.printf("%.3lf\n",3.555);

## 字符串流

//java字符串流

//input

String s="124 4523 5423";

Scanner reader=new Scanner(s);

while( reader.hasNext()){

int a=reader.nextInt();

System.out.println(a);

}

//print

StringWriter w=new StringWriter();

w.write("hello!");

System.out.println(w.toString());

## 树状数组

//树状数组

final int maxn=110000;

class IndexArray{

long[] sum;

int n;

public IndexArray(){

sum=new long[maxn];

}

void init(int n){ //1~n

this.n=n;

Arrays.fill(sum, 1, n, 0);

}

void update(int i,long val){

while(i<=n){

sum[i]+=val;

i+=Integer.lowestOneBit(i);

}

}

long get\_sum(int i){

long ret=0;

while(i>0){

ret+=sum[i];

i-=Integer.lowestOneBit(i);

}

return ret;

}

}//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## Solutions

1,构造一个解,然后调整优化. eg,pku上某题用匈牙利递归调优,zju月赛某题用栈...

2,dp的时候发现跟答案的奇偶性有关系,那么dp[i][0]表示答案为偶数的最优值,dp[i][1]...uva上某题101秒过的那个

3,2006/2007shanghai n\*sqrt(n)\*log(n)的复杂度....取折中的一个分界点....

4,统计类问题,当直接算很难算时,可以考虑列几个方程联立解未知数的方法...sample: 平面上三角形内部有几个点

5,dp可以设计下基于递归定义的状态,比如最多几条线段不相交......

6,对于树的题目,解决方案有以下:

1： dfs序列 (括号序列,欧拉Tour,作用于某棵子树的全部节点)

2： 基于点的分治

3： 基于链的分治(路径剖分)

7, 直接dp有后效性,需要一定排序,然后dp

/\*

zju 3077 http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=3077

直接dp有后效行,因为选择的顺序也是有影响的!!

对于任意的两个东西(a1，b1，val1), (a2，b2，val2)

如果最佳答案中这两个只选择了一个,那么顺序是没有关系的,都能dp出来

如果最佳答案中这两个都选了的,那么他们的顺序应该如何呢,

要是两个都选上那么, C-a1+b1 > a2或者 C-a2+b2 > a1

=> C + b1 >a1+a2 或者 C+b2>a1+a2

显然对于给定的C,那么b越大这个式子越成立,故应该按照b降序dp!!

\*/

8,最大流建模,为了满足一个点的流量最多只能流到相邻的点,那么可以把每个点分成两个,第一个专门送出,第二个专门接受

9,可行点的问题. srm510的一道题. 数据范围是10^10,但是可行值是2048个,怎么样才能节省枚举的点呢? 找必要条件,如果有充分条件的话,用逆否命题的形式反过来.

10,几种贪心方法.

1,最经典的.之前的值都是贪心的,放在一个栈或者队列里,然后当前值贪心不能时,设法对之前的值进行调整.

11,几种dp方法.

1,有些问题不能直接dp,需要先排个序. 关键是导出他们的比较函数.(设两个东西在最佳方案中一前一后,然后导出关系) 见7.

12,括号序列的一个性质: 如果从前往后扫描的话,那么把当和小于0时,可以将<=curPos的 \*任何一个\* ')'变成'('来解决问题!. eg,codeforces 3D

13,二维的最长上升子序列问题. 因为要严格上升,那么(x,y)要同时大于.那么可以这样先给他们排个序,首先x从小到大,如果x相等,那么y从大到小. 这样如果<i的dd[j].y < dd[i].y,那么必须有dd[j].x<dd[i].x. eg,codeforces 4D

14,括号序列问题. 这个明显的栈结构,可以看成是那种"消消球",有成对的就消掉.那么栈顶的就是第一个没有消掉的!

15,简单平面图,N(>=3)个点的话,那么边最多有3\*(N-2)条,因为形成的face必须是三角形,如果是4边形或更多,那么可以再连几条...

The Euler formula states that Vertex + Face = Edge + 2.

这样理解:

首先平面上一条线段 vertex=2 Face=1 Edge=1 满足

然后,每加一条边,要么是多了一个点,要么是两点连起来了,多了一个面

16,容斥原理解题的时候,可能类别不是10几个这样,可能有1000+个,但是他们都是两三个就超limit的那种,这样暴力枚举的时候就需要排个序,从大到小的.

这样可以少很多枚举. 比如 bound=N, a<b<c lcm(a,b)<N lcm(b,c)>N 那么从小到大枚举就会枚举到lcm(1,2,3),从大到小就不会了.

17,

void test(){ //在计算机里,0乘无穷大不是0!!!!!!

double c; cin>>c; //如果输入0.000 那么不是报错,而是输出-1.#IND00

double a=2/c;

double b=a\*c;

printf("%lf\n",b);

}

18, 树状数组可以完成这样的维护:

每次给区间[1,x]一个val,取最小操作. 查询一个数x的值.

具体实现,想下树状数组的那棵树... 取最小用get\_sum,查询用update

19,给定n,计算1~n里面gcd(i,n)=1的i的和.... 结果为 phi(n)\*n/2 , 1特判,以为算phi(1)本来就是\*\*\*\*

20,数学推导题目，来源于我的组合数学作业……可以看成是坐标系中，从(0,0)点走到(m, n)点，并且跟y=x-1这条直线不相交的方案数。(0,0)点关于直线y=x-1的对称点是(1,-1)，从(1,-1)点走到(m, n)点的所有方案均一定会与直线y=x-1相交，对于这些方案，将从(1,-1)点到与y=x-1的第一个交点之间的路径关于y=x-1对称翻转过去，就可以得到所有不满足题意的从(0,0)点走到(m, n)点的方案，于是最终答案就是C(n+m, n)-C(n+m,m-1)。

原题的取模的数是一个素数，但是在省选的时候，有些大牛知道结论，于是又被秒了，5555555555555555555，所以这次，把取模的数改成了今天的日期，同时也祝大家五一快乐，呵呵

21,一种递推方法

prob: n个数,共有2^n个permutation,求有多少种不同的per...

solution: 动态规划,dp[i]表示以a[i]结束的per...数,考虑两种情况

1: 不存在j<i使得a[j]=a[i],那么dp[i]=1+dp[1]+...+dp[i-1]

2: 存在最大的j<i有a[j]=a[i],那么dp[i]=dp[j]+...+dp[i-1]

22, 一些估算常识

1， O(1/1+1/2+1/3+...+1/n)是O(ln n)级别的,由积分算出来的

2， n以内的素数个数是n/ln(n)

3, n的真因数很少,10万以内的整数，真因数最多只有80+，

23, 没有奇环的图是二分图! (tricky)

24,prob:N个任务,每个任务有一个持续时间t和一个截止时间d,cost[i]=max(0,c[i]-d[i]),其中c[i]是它的完成时间.

合理安排使得最大的两个cost的和最小.

tips: 如果一个问题很难转化的时候,可以先思考下这个问题的简化版.

首先考虑这个问题: N个任务,每个任务有一个持续时间t和一个截止时间d,问能不能完成.

这个问题比较明显的,我们只要按照截止时间排序,然后O(n)判断即可.

现在假设最大的两个值是A和B(A>=B),那么所有的任务都必须在d[i]+B时间内完成除了一个可以在d[i]+A内完成,所以只要

按照d排序后,然后枚举一个可以在A内完成的任务.

25,

平面上有n个点,每个点都由一个正方形范围(边长为2\*L),每个点跟在它所在范围内的点连通

求和某个点连通的点数目.

solution: 思路就是构建一个能表示所有点连通性且边数不太多的无向图,然后每次bfs(或并查集)

首先,左右对称下,只要每个点和它左范围内的点连边,那么是可以的. 这个还是边太多.

再考虑上下方向,如果我们将某个点和它左范围内y轴距离最小的两个点(一上一下)连边,

那么这个传递性就可以一直延续下去了!(可以自己手动画画)

26,

prob:n个数,求二进制第k位是1的个数

solution: 把多余k位的数折叠回来(取个模),因为高位不影响低位,然后总共可能的区间是[2^k-1,2^k)这些数...搞搞

27,网络流Tle了,看看是不是边太多了!!有些边是可以合并的!!

28,原根:

/\*

原根Primitive Root。

　　设m是正整数，a是整数，若a模m的阶等于φ(m)，则称a为模m的一个原根。（其中φ(m)表示m的欧拉函数）

　　假设一个数g对于P来说是原根，那么g^i mod P的结果两两不同,且有 1<g<P, 0<i<P,那么g可以称为是P的一个原根,归根到底就是g^(P-1) = 1 (mod P)当且当指数为P-1的时候成立.(这里P是素数).

　　简单来说，g^i mod p ≠ g^j mod p （p为素数）

　　其中i≠j且i, j介於1至(p-1)之间

　　则g为p的原根。

　　求原根目前的做法只能是从2开始枚举，然后暴力判断g^(P-1) = 1 (mod P)是否当且当指数为P-1的时候成立

　　而由于原根一般都不大，所以可以暴力得到.

\*/

/\*

如果P是奇素数，那么Inv(i) -> j, 其中j是 [1.. P-1]内的一个数字

并且Inv()这个函数是一一映射的：

对于一个j, 有且只存在一个i,使得Inv(i) = j ( mod P)

对于一个i, Inv(i) = j (mod P), 只对应一个j

(证明用原根和欧拉定理)

这样子可以把底数换成一样的了,比如a^s和b^t做比较,那么可以把底数a用原根定理的一一映射搞成b^k…

\*/

29,（威尔逊（Wilson）定理）设http://www.pep.com.cn/gzsx/jszx/sxjs/jsjz/200709/W020070925583505023856.gif为质数，则http://www.pep.com.cn/gzsx/jszx/sxjs/jsjz/200709/W020070925583505335651.gif。