

Keshuai

2015-11-16

Copyright ：1.0

数学

一．计数问题7

（一）基本计数方法 7

（二）常见计数问题 7

（三）组合数性质 8

（四）典型问题 8

例题：Uva 11529（极角排序计数问题）8

例题：Codeforces 451E（容斥原理+隔板法）10

二．递推关系12

三．数论12

（一）素数12

1、素数筛选法12

2、米勒-拉宾判素数12

3、大区间筛选素数13

例题：uva 1040（大区间筛选素数）13

4、梅森素数14

5、反素数14

应用：分解质因子14

（二）欧几里得算法18

（三）欧拉函数18

（四）模算术19

1、取模快速幂19

2、逆元19

（五）线性模方程20

1、求线性模方程解20

2、中国剩余定理（孙子定理）20

3、离散对数（大步小步算法）21

（六）其他定理23

1、数论四大定理(威尔逊定理,欧拉定理,孙子定理,费马小定理)23

2、勒让德记号23

3、凯莱定理(cayley)23

4、牛顿插值法23

5、完美洗牌23

四．博弈24

（一）取石子游戏24

例题：uva 1567（k倍动态减法）25

（二）删边游戏27

（三）图上移动石子28

例题：uva 12163（有向图移动石子）28

五．其他30

（一）矩阵快速幂30

（二）高斯消元31

（三）数值方法32

数据结构

一．STL+基本数据结构35

（一）常用STL语法35

1、排序检索35

2、队列+栈35

3、容器36

（二）高精度38

1、C++38

2、java41

3、分数42

（三）辅助43

1、输入输出外挂43

2、手动括栈43

二．树形数据结构44

（一）并查集(union-find)44

（二）树状数组(fenwick)46

（1）单点修改区间查询47

（2）区间修改单点查询47

（3）二维树状数组维护矩形权值47

例题：Hdu 4456（离散化加二维树状数组）47

（三）静态区间最值查询(RMQ)49

（四）线段树(segment-tree)50

（1）区间合并51

例题：Codeforces 484E（可持久化线段处理区间合并）53

（2）扫描线55

例题：Poj 2482(统计矩形内最大点个数问题转化)55

例题：Uva 11983(矩形覆盖 K 次面积问题)57

（3）其他题型60

例题:Uva 12436(维护等差数列和问题)60

例题:Hdu 3333(区间不重复求和)62

（五）树链剖分65

例题:hdu 4897(树链剖分点边权变形)67

（六）最近公共祖先(LCA)71

例题:hdu 2874(离线LCA)72

（七）笛卡尔树(treap-tree) 74

例题:hdu 4605(可持久化笛卡尔树维护树形前缀)75

（八）伸展树(splay-tree)X

（九）动态树X

（十）划分树X

（十一）主席树X

（十二）KD树X

（十三）替罪羊树X

三．字符串78

（一）基本78

1、最长回文串(Manacher)78

2、最小表示法78

（二）KMP79

（三）字典树(Tire)81

（1）匹配去重82

例题:Hdu 4760(字典树去重高级应用)82

（2）模糊匹配85

（3）亦或和最大问题85

例题:Hdu 4757(可持久化字典树解决亦或和)85

（四）AC自动机(Aho-Croasick)88

（1）匹配问题89

（2）结合DP89

例题:Hdu 3341(AC自动机+变进制状压DP)89

（五）后缀数组92

例题:Hdu 5030(切割K次后的最大子串)93

（六）后缀自动机96

例题：spoj 1812（多个串的最长公共子串）97

四．其他数据结构100

（一）莫队算法100

计算几何

一．二维几何103

（一）基础103

1、点/向量103

2、线/线段104

3、圆105

4、多边形107

（二）面积交109

1、圆与圆/多边形109

2、扫描法：求多个图形的面积交110

例题：hdu 4629（多个三角形面积并）110

二．三维几何113

（一）基础113

1、点/向量113

2、线/线段113

3、三角形/四面体114

4、球115

三．几何算法116

（一）凸包116

1、二维凸包116

（1）点集面积和：计算给出点集组成的任意多边形面积和116

例题：Zoj 3871（给定点集）116

例题：Hdu 5448（给定凸包）117

2、三维凸包118

（二）旋转卡壳120

（三）半平面交121

（四）仿射矩阵：旋转问题X

（五）运动规划122

例题：uva 1017（轮廓线）122

图论

一．基础图论131

（一）拓扑排序131

（二）欧拉路132

二．深度优先遍历133

（一）无向图割顶和桥133

（二）无向图双联通分量135

（三）有向图强联通分量137

（四）2-SAT138

例题：uva 12273（2sat变形+限制条件）138

三．最短路X

四．生成树141

（一）最小生成树141

（二）次小生成树143

（三）最小有向生成树144

五．二分图146

（一）二分图判定146

（二）最大匹配147

（三）最佳完美匹配148

（四）稳定婚姻问题149

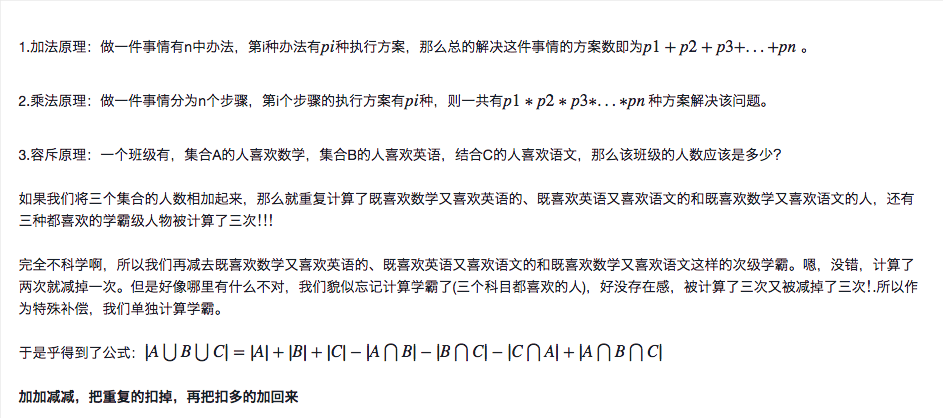
例题：uva 1175（稳定婚姻问题）149

六．网络流X

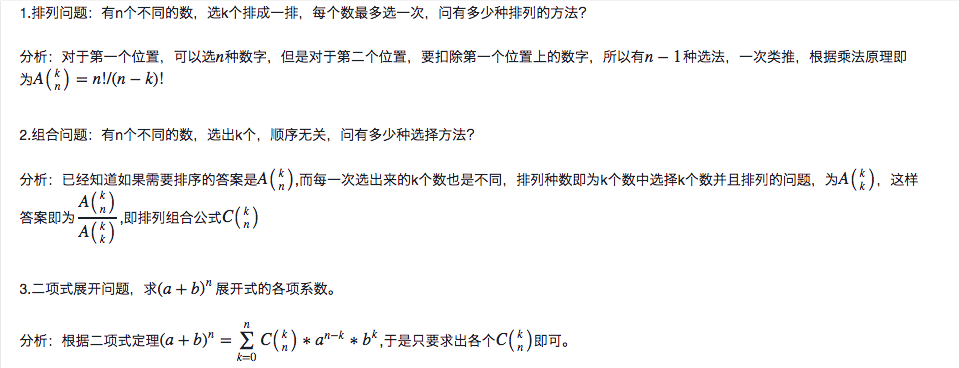
第一章 数学

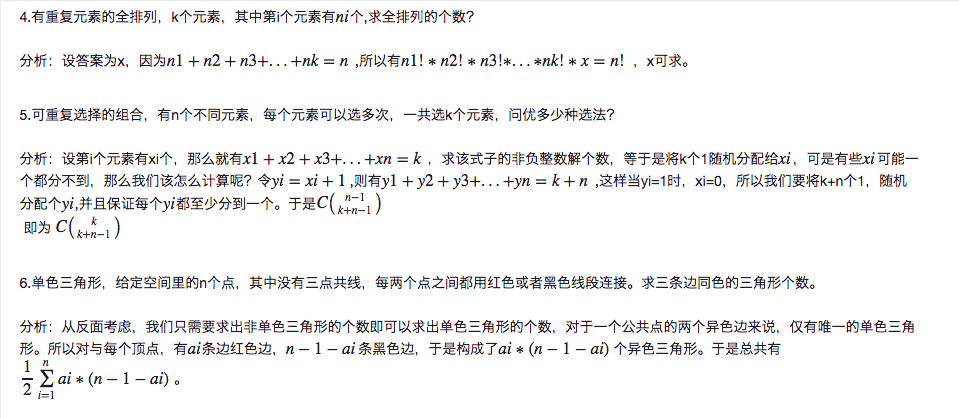
一、计数问题

（一）基本计数方法



（二）常见计数问题





7. 给定n维空间的线段，问说线段经过几个格子

分析：将线段平移至原点，等价于(0,0,0)->(a,b,c)，ans = a + b + c +.. - gcd(a,b) - gcd(a,c) - .. + gcd(a, b, c) ...

（三）组合数性质



lucas定理：用来求解大数的组合数取模问题。



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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Conbin-Number.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

typedef long long type;

type conbin[maxc][maxc];

type pow\_mod (type a, type n, type mod) {

type ans = 1;

while (n) {

if (n&1)

ans = (ans \* a) % mod;

a = (a \* a) % mod;

n >>= 1;

}

return ans;

}

type ConbinNum\_One(type a, type b, type mod) {

if (a < b) return 0;

if (b > a - b)

b = a - b;

type up = 1, down = 1;

for (type i = 0; i < b; i++) {

up = up \* (a-i) % mod;

down = down \* (i+1) % mod;

}

return up \* pow\_mod(down, mod-2, mod) % mod; // 逆元

}

void ConbinNum\_Tab (int n, type mod) {

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

conbin[i][0] = conbin[i][i] = 1;

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

conbin[i][j] = (conbin[i-1][j-1] + conbin[i-1][j]) % mod;

}

}

type lucas (type a, type b, type mod) {

if (b == 0) return 1;

return ConbinNum\_One(a%mod, b%mod, mod) \* lucas(a/mod, b/mod, mod) % mod;

}

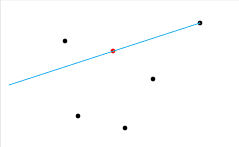
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

（四）典型问题

极角r=atan2(y-o.y, x-o.x),有时需要通过将极角加上2\*pi来保证周期性。

例题：Uva 11529（极角排序计数问题）

题目大意：给出若干个点，保证任意三点不共线，任意选三个点作为三角行，其他点若又在该三角形内，则算是该三角形内部的点，问所有情况的三角形平均每个三角形有多少个内部点。

解题思路：三角形的总数很容易求C(n,3),现在就是要求各个三角形内部点的总数，同样我们可以反过来，求每个点在多少个三角形的内部。然后我们确定一个点，求该点在多少个三角的内部，剩余n-1个点，可以组成C(n−1,3)个三角形，所以只要求出该点在哪些三角形的外部即可。  
  
红色点为选中的点，将周围点按照与选中点的极角进行排序，每次枚举一点，它的极角为a，所有极角小于a+pi的点，这些点组成的三角形，选中点一定在外部。处理一周的方式是将点的数组扩大两倍，将所有点的极角加上pi有保留在延长的数组中。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*uva11529.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <cmath>

#include <algorithm>

using namespace std;

const int N = 1205;

const double pi = 4 \* atan(1.0);

const double eps = 1e-9;

int n;

double s, r[2\*N];

struct point {

double x, y;

}p[N];

double Count (int d) {

int c = 0, mv = 0;

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

if (i == d) continue;

double a = atan2(p[i].y-p[d].y, p[i].x-p[d].x);

r[c] = a;

r[c+n-1] = a + 2\*pi;

c++;

}

c = 2 \* n - 2;

sort(r, r + c);

double ans = 0;

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

double tmp = r[i] + pi;

while (tmp > r[mv])

mv++;

double cnt = mv - i - 1;

ans = ans + cnt \* (cnt-1) / 2;

}

return s - ans;

}

double solve () {

s = (n-1) \* (n-2) \* (n-3) / 6.0;

double c = n \* (n-1) \* (n-2) / 6.0;

double ans = 0;

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

ans += Count(i);

return ans / c;

}

int main () {

int cas = 1;

while (scanf("%d", &n) == 1 && n) {

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

scanf("%lf%lf", &p[i].x, &p[i].y);

printf("City %d: %.2lf\n", cas++, solve());

}

return 0;

}

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

例题：Codeforces 451E（容斥原理+隔板法）

题目大意：有n个花坛，要选s支花，每个花坛有f[i]支花。同一个花坛的花颜色相同，不同花坛的花颜色不同，问说可以有多少种组合。

解题思路：2^n的状态，枚举说哪些花坛的花取超过了，剩下的用C(sum+n−1,n-1)隔板法计算个数，注意奇数的位置要用减的，偶数的位置用加的，容斥原理。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CF451E.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <cmath>

#include <algorithm>

using namespace std;

typedef long long ll;

ll pow\_mod (ll a, ll k, ll p) {

ll ans = 1;

while (k) {

if (k&1)

ans = (ans \* a) % p;

a = (a \* a) % p;

k /= 2;

}

return ans;

}

ll C (ll a, ll b, ll p) {

if (a < b) return 0;

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

ll up = 1, down = 1;

for (ll i = 0; i < b; i++) {

up = up \* (a-i) % p;

down = down \* (i+1) % p;

}

return up \* pow\_mod(down, p-2, p) % p; // 逆元

}

ll lucas (ll a, ll b, ll p) {

if (b == 0)

return 1;

return C(a%p, b%p, p) \* lucas(a/p, b/p, p) % p;

}

const int maxn = 25;

const ll mod = 1e9+7;

int n;

ll s, f[maxn];

ll solve () {

ll ans = 0;

for (int i = 0; i < (1<<n); i++) {

ll sign = 1, sum = s;

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

if (i&(1<<j)) {

sum -= (f[j]+1);

sign \*= -1;

}

}

if (sum < 0) continue;

ans += sign \* lucas(sum + n - 1, n - 1, mod);

ans %= mod;

}

return (ans + mod) % mod;

}

int main () {

scanf("%d%lld", &n, &s);

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

scanf("%lld", &f[i]);

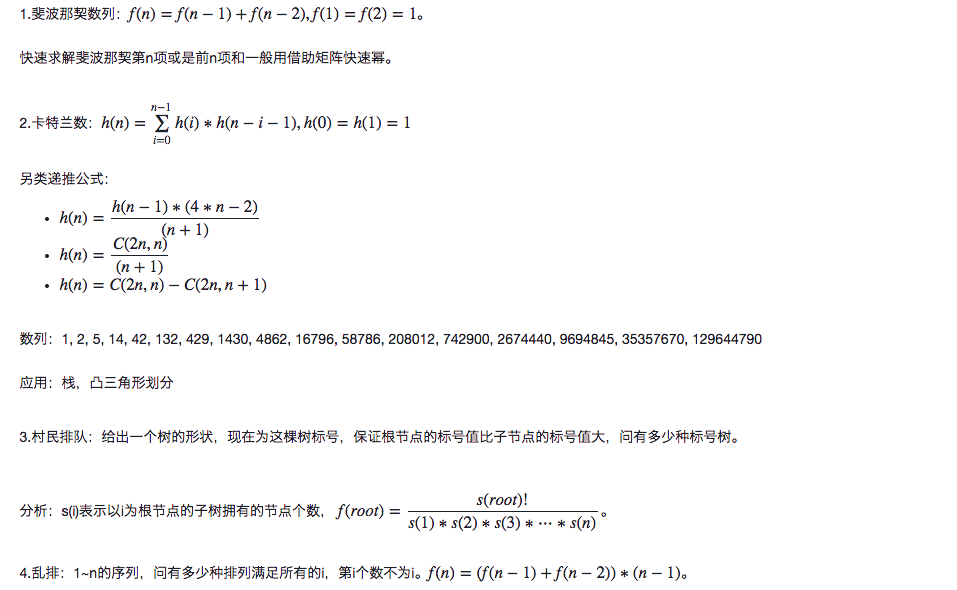
printf("%lld\n", solve());

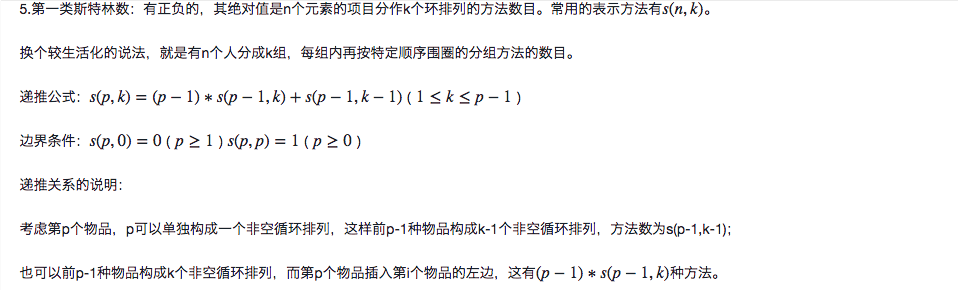
return 0;

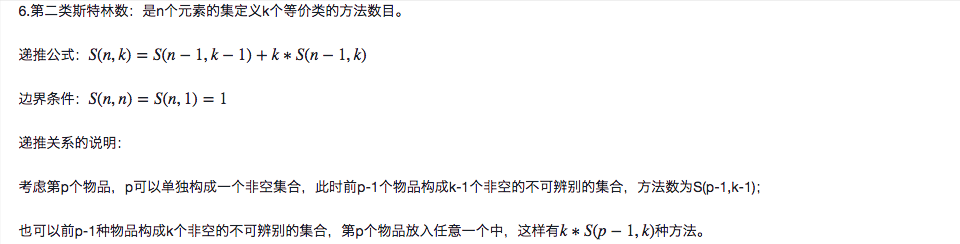
}

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

二、递推关系







三、数论

（一）素数

1、素数筛选法

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*prime-table.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

\* 埃拉托斯特尼素数筛选法

\* 筛选一定范围内的所有素数.

\* 复杂度近似于o(n), 所以数据处理范围为1e6-1e7可以接受.

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

void sieve(int n) {

int m = (int)sqrt(n+0.5);

memset(vis, 0, sizeof(vis));

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

if (!vis[i]) {

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

vis[j] = 1;

}

}

}

int prime\_table (int n, int\* pri) { // 返回n以内素数个数。

sieve(n);

int c = 0;

for (int i = 2; i <= n; i++)

if (!vis[i])

pri[c++] = i;

return c;

}

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

2、米勒-拉宾判素数

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*miller-rabin.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

\* 米勒-拉宾算法(miller-rabin)

\* 判断一个数是否为素数.

\*

\* 1. 费马小定理

\* 对于一个素数来说，a^(n-1) % n 恒等于1 (1 <= a <= n-1)

\* 2. 快速幂取模

\* 3. 大数相乘（long long）取模

\* 两个long long 型的数相乘有可能大于long long型的上限

\*

\* 随机生成一个a, 判断a^(n-1) % n 是否为1, 如果不为1可以确定不是素数.

\* 对于合数来说通过测试的概率不足25%.

\* 判断一定次数，使得该数为素数的概率趋近于1.

\* 3215031751 需要测试较多次，否则会失准.

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

#include <ctime> // possible need;

#include <cstdlib> // possible need;

typedef long long type;

type mul\_mod (type a, type b, type mod) {

type ret = 0;

while (b) {

if (b&1) ret = (ret + a) % mod;

a = (a + a) % mod;

b >>= 1;

}

return ret;

}

type pow\_mod (type a, type n, type mod) { // mod <= 1e18;

type ret = 1;

while (n) {

if (n&1)

ret = ret \* a % mod;

a = a \* a % mod;

n >>= 1;

}

return ret;

}

bool miller\_rabin(type n) {

if (n < 2)

return false;

srand(time(0));

for (int i = 0; i < 20; i++)

if (pow\_mod(rand() % (n-1) + 1, n-1, n) != 1)

return false;

return true;

}

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

3、大区间筛选素数

例题：Uva 1404（大区间筛选素数）

题目大意：如果k个相邻的素数p1,p2,…,pk，满足pk−p1=s，称这些素数组成一个距离为s的素数k元组，给定区间a，b，求有多少个距离s的k元组。

解题思路：筛选素数法，先预处理出[1, sqrt(inf)]的素数表，然后对给定区间[a,b]根据预处理出的素数表筛选出素数即可。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*uva1404.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cmath>

#include <vector>

#include <algorithm>

using namespace std;

const int sqrt\_inf = 46340;

const int maxn = 2 \* 1e9;

int np, pri[sqrt\_inf];

bool vis[maxn+5];

vector<int> vec;

void prime\_table (int n) {

np = 0;

memset(vis, 0, sizeof(vis));

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

if (vis[i])

continue;

pri[np++] = i;

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

vis[j] = 1;

}

}

int solve () {

int ret = 0;

int a, b, s, k;

vec.clear();

memset(vis, 0, sizeof(vis));

scanf("%d%d%d%d", &a, &b, &k, &s);

for (int i = 0; i < np && pri[i] \* pri[i] <= b; i++) {

int u = pri[i], d = (u - a % u) % u;

if (u == a + d)

d += u;

while (d <= b - a) {

vis[d] = 1;

d += u;

}

}

for (int i = 0; i <= b-a; i++) {

if (vis[i] == 0 && a + i > 1)

vec.push\_back(a+i);

}

for (int i = 0; i + k - 1 < vec.size(); i++) {

if (vec[i+k-1] - vec[i] == s)

ret++;

}

return ret;

}

int main () {

prime\_table(sqrt\_inf);

int cas;

scanf("%d", &cas);

while (cas--) {

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

}

return 0;

}

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

4、梅森素数

\* 梅森素数，(2^k) – 1形式的素数

\* 性质：一个数若能差分成若干个不同的梅森素数的积，那么该数的所有因子和可以写成2^n形式。

\* 梅森素数前10项（幂）：2，3，5，7，13，17，19，31，61，89…

5、反素数

反素数是一种素数，当它的数字反过来后，仍然是一个素数。

[13](http://baike.baidu.com/view/40117.htm" \t "_blank), [17](http://baike.baidu.com/view/521455.htm" \t "_blank), [31](http://baike.baidu.com/view/521968.htm" \t "_blank), [37](http://baike.baidu.com/view/249918.htm" \t "_blank), [71](http://baike.baidu.com/view/522169.htm" \t "_blank), [73](http://baike.baidu.com/view/522474.htm" \t "_blank), [79](http://baike.baidu.com/view/522182.htm" \t "_blank), [97](http://baike.baidu.com/view/522168.htm" \t "_blank), [107](http://baike.baidu.com/view/521291.htm" \t "_blank), [113](http://baike.baidu.com/view/604199.htm" \t "_blank), [149](http://baike.baidu.com/view/625925.htm" \t "_blank), [157](http://baike.baidu.com/view/1153039.htm" \t "_blank)...

应用：

（1）分解质因子

方法1：试除法

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*normal-divfactor.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

\* 试除法分解因子

\* 将一个数分解成质因子.

\*

\* 1. 米勒-拉宾(特殊情况下的优化)

\* 判断一个数为素数

\*

\* 枚举2~sqrt(n)的数，判断是否为n的因子.

\* 复杂度为o(sqrt(n)).

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

typedef long long type;

void div\_factor (int& cnt, type\* factor, type n) {

cnt = 0;

type m = (type)sqrt(n+0.5);

for (type i = 2; i <= n && i <= m; i++) {

if (n % i == 0) {

while (n % i == 0) {

factor[cnt++] = i;

n /= i;

}

}

/\* 可以通过用miller-rabin算法判断进行优化 \*/

}

if (n != 1)

factor[cnt++] = n;

}

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

方法2：费马方法

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*feramat-divfactior.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

\* 费马方法（feramat)

\* 将整数拆分成质因子.

\*

\* 1. 米勒-拉宾

\* 判断一个数是否为素数

\*

\* 一个整数N, 可以分解成一个奇数上2^k次方,即N = (2 \* n + 1) \* 2^k;

\* 令M = 2 \* n + 1, 如果M 不为素数, 那么M 肯定可以拆分成两个奇数相乘M = c \* d;

\* 假设c >= d, 令a = (c + d) / 2, b = (c - d) / 2;

\* 那么M = a \* a - b \* b = 4 \* (c + d) / 4;

\* 于是这要枚举a, 保证a \* a - M为完全平数即可.

\*

\* 对于因子为比较大的素数, 并且个数比较少的情况来说,复杂度仍比较高.

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

#include <ctime> // possible need;

#include <cstdlib> // possible need;

typedef long long type;

type pow\_mod (type a, type n, type mod) {

type ans = 1;

while (n) {

if (n&1)

ans = ans \* a % mod;

a = a \* a % mod;

n >>= 1;

}

return ans;

}

bool miller\_rabin(type n) {

if (n < 2)

return false;

srand(time(0));

for (int i = 0; i < 20; i++)

if (pow\_mod(rand() % (n-1) + 1, n-1, n) != 1)

return false;

return true;

}

void feramat\_factor (int& cnt, type\* factor, type n) {

if (miller\_rabin(n)) {

factor[cnt++] = n;

return;

}

type x = (type) sqrt(n + 0.5);

while (x < n) {

type w = x \* x - n;

type y = (type) sqrt(w + 0.5);

if (w == y \* y) {

feramat\_factor(cnt, factor, x+y);

feramat\_factor(cnt, factor, x-y);

break;

}

x++;

}

}

void feramat\_divfactor (int& cnt, type\* factor, type n) { // N = (2\*n+1) \* 2^k;

cnt = 0;

if (n == 0) return;

while ((n&1) == 0) {

factor[cnt++] = 2;

n >>= 1;

}

if (n == 1)

factor[cnt++] = n;

else

feramat\_factor(cnt, factor, n);

}

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

方法3：pollard\_rho算法

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*pollard-rho-divfactor.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

\* pollard\_rho

\* 将整数拆分成质因子.

\*

\* 1. 米勒-拉宾

\* 判断一个束是否为素数

\* 2. 欧几里得

\* 求两个数的最大公约数

\*

\* 根据一定规则生成x, y, 求出y-x和n的最大公约数, 若不为1则为整数的一个因子.

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

#include <ctime> // possible need;

#include <cstdlib> // possible need;

typedef long long type;

type gcd (type a, type b) {

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

}

type pow\_mod (type a, type n, type mod) {

type ans = 1;

while (n) {

if (n&1)

ans = ans \* a % mod;

a = a \* a % mod;

n >>= 1;

}

return ans;

}

bool miller\_rabin(type n) {

if (n < 2)

return false;

srand(time(0));

for (int i = 0; i < 20; i++)

if (pow\_mod(rand() % (n-1) + 1, n-1, n) != 1)

return false;

return true;

}

type pollard\_rho(type n, type tmp) {

int i = 1, k = 2;

type x = rand() % n;

type y = x;

while(true) {

i++;

x = ( mul\_mod(x, x, n) + tmp) % n;

type d = gcd( (y > x ? y - x : x - y), n);

if (d != 1 && d != n)

return d;

if (y == x) return n;

if (i == k) {

y = x;

k <<= 1;

}

}

}

void findfactor (int& cnt, type\* factor, type n) {

if(miller\_rabin(n)) {

factor[cnt++] = n;

return;

}

type p = n;

while(p >= n)

p = pollard\_rho(p, rand() % (n-1) + 1);

findfactor (cnt, factor, p);

findfactor (cnt, factor, n / p);

}

void pollard\_divfactor (int& cnt, type\* factor, type n) {

cnt = 0;

srand(time(0));

findfactor(cnt, factor, n);

}

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

（二）欧几里得算法

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*gcd.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

\* 欧几里得算法：求出a和b的最大公约数d；

\*

\* 拓展欧几里得算法：求解线性方程的解

\* a \* x + b \* y = d；

\* 保证求出的解|x|+|y|最小。

\* d为a和b的最大公约数，即当有线性方程a \* x + b \* y = c，d = gcd(a,b)，c % d != 0时，方程无解。

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

type gcd(type a, type b) {

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

}

void exgcd(type a, type b, type& d, type& x, type& y) {

if (!b)

d = a, x = 1, y = 0;

else {

exgcd(b, a%b, d, y, x);

y-= x \* (a/b);

}

}

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

（三）欧拉函数

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

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

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

\* 欧拉函数：phi(x)等于不超过x且和x互质的整数个数

\*

\* phi(n)= n \* (1-1/p1) \* (1-1/p2) \* ... \* (1-1/pn); (pi为n的质因子)

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

int euler\_phi(int n) {

int m = (int)sqrt(n+0.5);

int ans = n;

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

if (n % i == 0) {

ans = ans / i \* (i-1);

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

}

}

if (n > 1)

ans = ans / n \* (n - 1);

return ans;

}

void phi\_table(int n, int\* phi) {

for (int i = 2; i <= n; i++)

phi[i] = 0;

phi[1] = 1;

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

if (!phi[i]) {

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

if (!phi[j])

phi[j] = j;

phi[j] = phi[j] / i \* (i - 1);

}

}

}

}

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

（四）模算术

1、取模快速幂

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*modular-equation.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

\* 模算术

\* (a \* b) % mod = ((a % mod) \* (b % mod)) % mod

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

type mul\_mod (type a, type b, type mod) { // 当mod^2 > inf 时

type ret = 0;

while (b) {

if (b&1)

ret = (ret + a) % mod;

a = (a + a) % mod;

b >>= 1;

}

return ret;

}

type pow\_mod (type a, type n, type mod) { // mod <= 1e18;

type ret = 1;

while (n) {

if (n&1)

ret = ret \* a % mod;

a = a \* a % mod;

n >>= 1;

}

return ret;

}

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

2、逆元

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*inv.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

\* 逆元：a \* inv(a,mod) % mod = 1;

\* mod 为素数是才有逆元

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

typedef long long type;

type inv (type a, type mod) {

type d, x, y;

exgcd(a, mod, d, x, y);

return d == 1 ? (x+mod)%mod : -1;

}

type inv (type a, type mod) { // 费马小定理，a^(mod-1) % mod = 1;

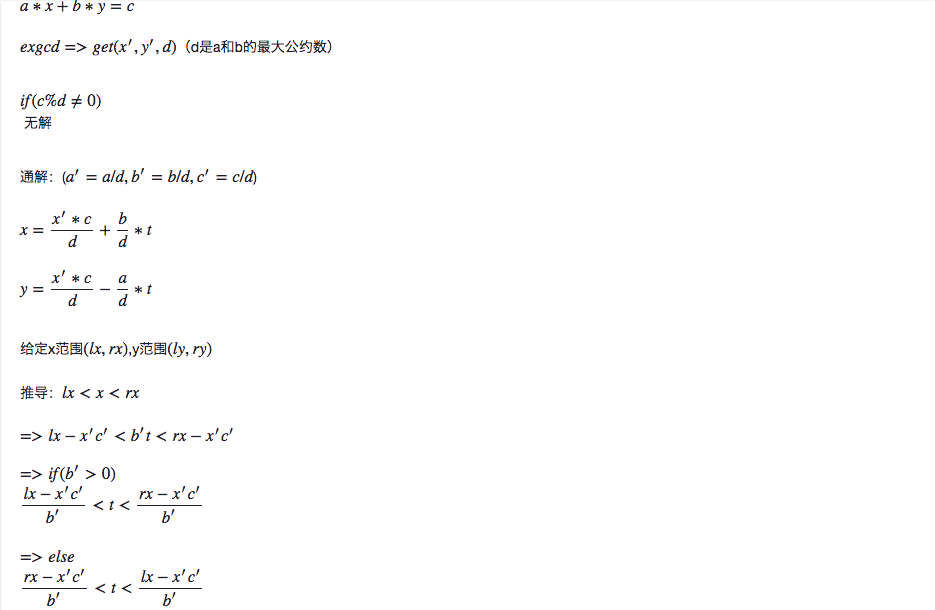
return pow\_mod(a, mod-2, mod);

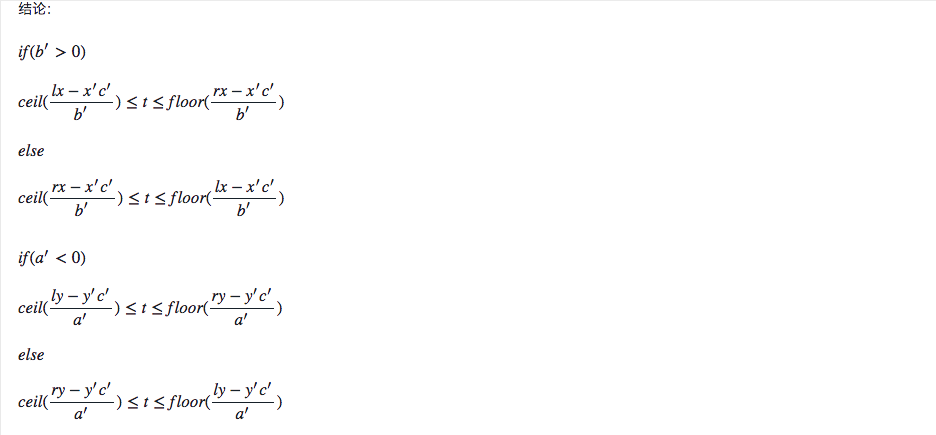
}

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

（五）线性模方程

1、求线性模方程解





2、中国剩余定理（孙子定理）

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*china.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

\* 中国剩余定理（孙子定理）

\*

\* 在多个线性模方程下，x = ai % mi(保证mi和mj在i≠的情况下互质)

\* 令M = 所有mi的积，wi = M / mi，且gcd(wi,mi)=1

\* 用拓展欧几里得求出pi和qi，使得wi\*pi + mi\*qi = 1

\* 令ei = wi \* pi，则方程组的解为x = (e1\*a1 + e2\*a2 + ... + en\*an) % M

\* 即在M的剩余系中x有唯一解

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

typedef long long type;

void exgcd (type a, type b, type& d, type& x, type& y) {

if (b == 0)

d = a, x = 1, y = 0;

else {

exgcd(b, a%b, d, y, x);

y -= (a/b) \* x;

}

}

type china (type\* a, type\* m, int n) {

type Mi = 1;

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

Mi \*= m[i];

type ret = 0;

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

type w = Mi/m[i], d, x, y;

exgcd(m[i], w, d, x, y);

ret = (ret + y\*w\*a[i]) % Mi;

}

return (ret % Mi + Mi) % Mi;

}

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

3、离散对数（大步小步算法）

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

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

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

\* 离散对数（大步小步算法）

\* 求解a^x = b % mod（mod为素数）

\*

\* 根据欧拉定理，只需要检查x=0,1..mod-1是不是解即可（a^(mod-1) % mod = 1）

\* 算法：先检查前m项，m取sqrt(mod)，并且计算出a^m的逆a^-m

\* 然后考虑m+1 ~ 2m项，假设存在ei \* a^m = b % mod，两边同乘a^-m得ei = b' % mod

\* 所以每次将b乘以a^-m之后在1~m中查找即可（map优化）

\* 之所以m取sqrt(mod)是因为在此时复杂度最低

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

#include <map>

using namespace std;

typedef long long type;

type mul\_mod (type a, type b, type mod) {

return (type)a \* b % mod;

}

type pow\_mod (type a, type n, type mod) {

type ans = 1;

while (n) {

if (n&1)

ans = mul\_mod(ans, a, mod);

a = mul\_mod(a, a, mod);

n /= 2;

}

return ans;

}

void exgcd (type a, type b, type& d, type& x, type& y) {

if (!b)

d = a, x = 1, y = 0;

else {

exgcd (b, a%b, d, y, x);

y -= x \* (a/b);

}

}

type inv (type a, type mod) {

type d, x, y;

exgcd(a, mod, d, x, y);

return d == 1 ? (x+mod)% mod : -1;

}

int log\_mod (type a, type b, type mod) {

type m = (type)sqrt(mod+0.5), v, e = 1;

v = inv(pow\_mod(a, m, mod), mod); // 计算a^(-m)

map<type, type> g;

g[1] = 0;

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

e = mul\_mod(e, a, mod);

if (!g.count(e)) // 记录e^i

g[e] = i;

}

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

if (g.count(b))

return i\*m+g[b];

b = mul\_mod(b, v, mod);

}

return -1;

}

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

（六）其他定理

1、数论四大定理（威尔逊定理，欧拉定理，孙子定理，费马小定理）

**威尔逊定理：**

若p是质数，则p可以整除（p-1）!+1

**欧拉定理:**

也称费马-欧拉定理，若有n，a为正整数，且n，a互质

a^phi(n) = 1 (mod n)

**孙子定理:**

也称中国剩余定理，见文档16页

**费马小定理：**

假如p是质数，且(a,p)=1，那么 a^(p-1) ≡1（mod p） 。

2、勒让德记号

对于整数a和**奇素数**p，tag = a^((p-1)/2) (mod p)

* tag = 0时，a = 0 (mod p)
* tag = 1时，存在某个整数x，x^2 = a (mod p)
* tag = -1时，不存在某个整数x，x^2 = a (mod p)

3、凯莱定理（cayley）

有n个标志节点的树的数目等于n^(n−2) (仅是cayley在组合数学中的应用)

4、牛顿插值法

给出一个序列的n项，求满足序列规律的后c项，要求尽量小

每次求两项之间的差，形成一个新的序列，直到序列公差为0时，回带回原来的序列

5、完美洗牌

给定n张牌，从1~n，每次将1~n/2分别放到2x的位置，n/2+1~n的牌放到(x-n/2) \* 2 - 1的位置

等价于每次将第x张牌放到2x %(2p+1)的位置上去

现在要判断洗n-1次牌是否使得牌变成原先序列，只要判断2^(n-1) % (2p+1) = 1

四、博弈

**问题特征：**

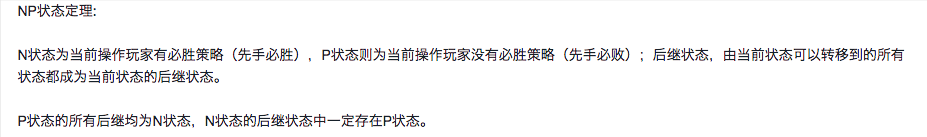
1. 游戏两人轮流操作，并且都以最优方式决策
2. 无法操作时为游戏终止状态
3. 游戏中的同一状态不会重复出现，也不会出现平局
4. 任意状态的决策集合只与当前状态有关，与决策者无关

**问题层次：**

* 对于给定的初始状态判断胜负
* 寻找单回合的必胜选择
* 寻找多回合的必胜选择（交互题）

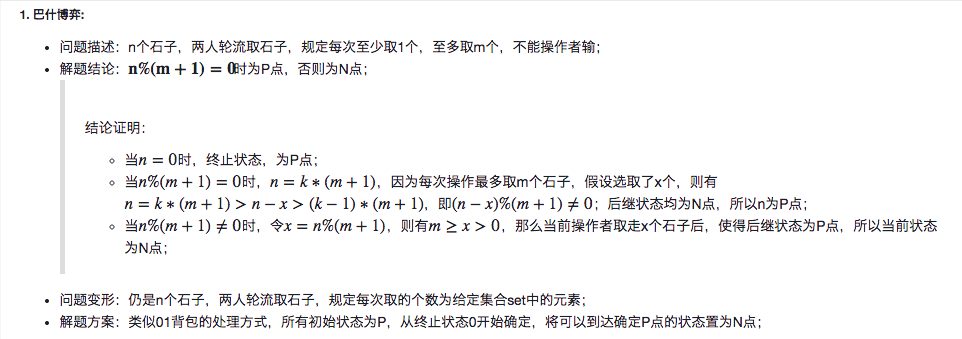
**解决方法：**

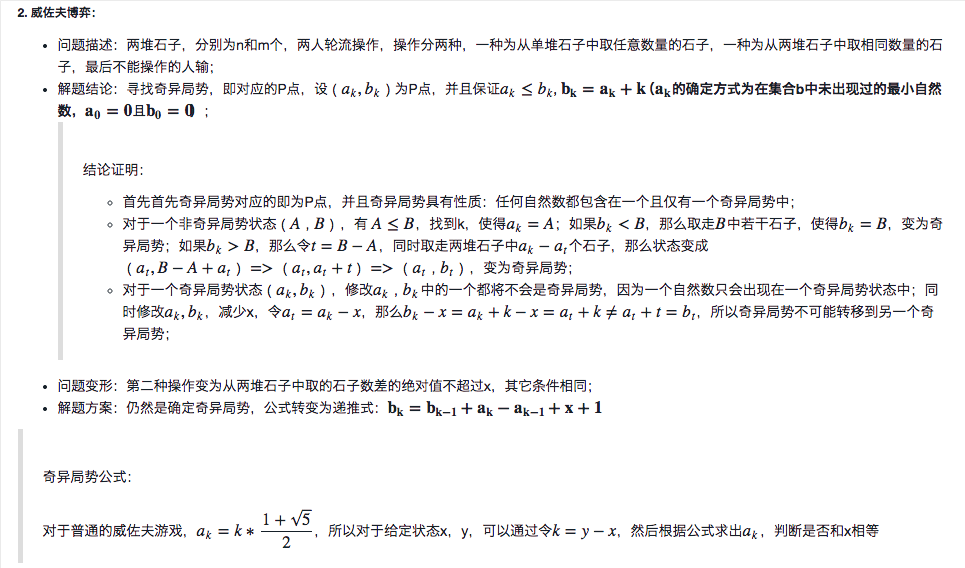
* 拓扑逆序倒推
* 记忆化搜索

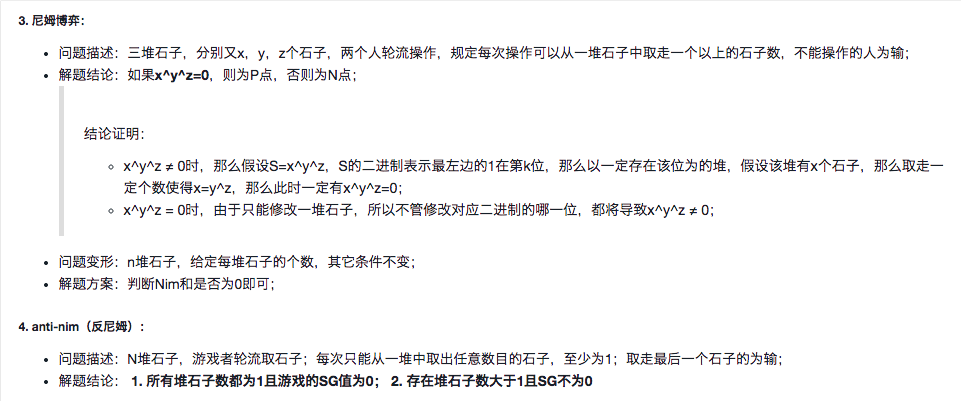


**典型问题：**

1. 取石子游戏







5. K倍动态减法

问题描述：N个石子，，两人轮流操作取石子，取的石子数不能超过对手上一次取的石子数m的K倍，取到最后一个石子的人胜利，第一次取只能取1~N-1个石子。先手必胜时输出最小的首次操作。

解题结论：构造数列，将N写成数列中一些项的和，使得这些被取到的项的相邻两个倍数差距>K

例题：Uva 1567（K倍动态减法）

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*uva1567.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

const int maxn = 1e6+5;

int N, K, a[maxn], b[maxn];

int main () {

int cas;

scanf("%d", &cas);

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

scanf("%d%d", &N, &K);

int p = 0, q = 0;

a[0] = b[0] = 0;

while (a[p] < N) {

a[p+1] = b[p] + 1;

p++;

while (a[q + 1] \* K < a[p])

q++;

b[p] = b[q] + a[p];

}

printf("Case %d: ", i);

if (N == a[p])

printf("lose\n");

else {

int ans;

while (N) {

if (N >= a[p]) {

N -= a[p];

ans = a[p];

}

p--;

}

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

}

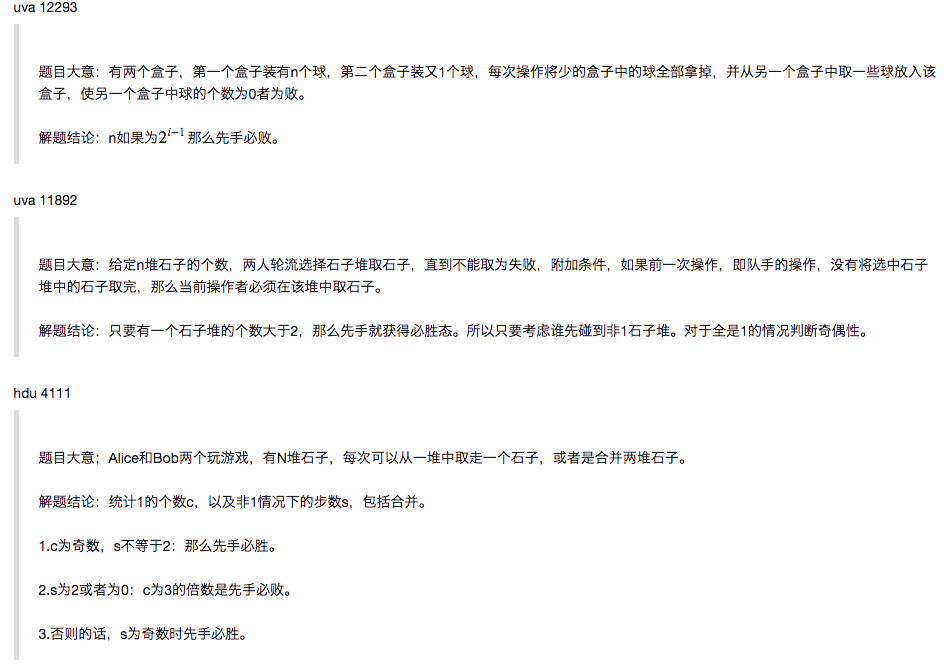
}

return 0;

}

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

6. 一些题目结论



1. 删边游戏

例题： **Uva 12033（树形删边游戏）**

题目大意：给定图，以0为根节点，每条边有一个长度，两个人轮流操作，每次为一条边上色，上一个单位长度，当一条边的颜色被涂满，则算作是减掉整段子树。判断先手是否必胜。

解题思路：SG定理，对于当前节点u，每次考虑字节点v，u-v边的长度为l  
 当l为1时：sg(u) ^= (sg(v) + 1)  
 当l为奇数时： 需要判断sg(v)奇偶性，奇数-1，偶数+1；  
 当l为偶数时：sg(u) ^= sg(v)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*uva12033.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <vector>

#include <algorithm>

using namespace std;

const int maxn = 1005;

int N, W[maxn][maxn];

vector<int> g[maxn];

int dfs (int u, int p) {

int ret = 0;

for (int i = 0; i < g[u].size(); i++) {

int& v = g[u][i];

if (v != p) {

int sg = dfs(v, u);

if (W[u][v] == 1)

ret ^= (sg+1);

else if (W[u][v]&1)

ret ^= (sg + (sg&1 ? -1 : 1));

else

ret ^= sg;

}

}

return ret;

}

int main () {

int cas, u, v, w;

scanf("%d", &cas);

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

scanf("%d", &N);

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

g[i].clear();

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

scanf("%d%d%d", &u, &v, &w);

g[u].push\_back(v);

g[v].push\_back(u);

W[u][v] = W[v][u] = w;

}

printf("Case %d: %s\n", k, dfs(0, -1) ? "Emily" : "Jolly");

}

return 0;

}

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

1. 图上移动石子

例题： **Uva 12163（有向图移动石子）**

题目大意:两个人进行游戏，对于每一局有一个无向图，给出无向图，每个节点有个K值，两人轮流操作，每次可以选中国一个含有石子的节点，将该节点的一个石子拿掉，然后选择K个有边连接的节点加上一个石子（节点可以重复选择），每个节点的子节点不会超过15个。不能操作的人视为失败。每局有n轮，给定每轮中每个节点上石子的初始值，问先手胜利还是失败。

解题思路：有向图上移动石子的组合游戏，对于没有子节点的节点SG值为0，然后对于每个节点，用记忆化的方式处理出SG值，注意因为要选中K个节点，但是子节点的个数最多为15，然后对于选中偶数次的节点可视没选，所以枚举215状态即可，并且要保证选中的个数奇偶性和K值相同。  
对于每一轮给定的初始状态，计算各个子游戏的Nim和即可。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*uva12163.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <vector>

#include <map>

#include <algorithm>

using namespace std;

const int maxn = 105;

vector<int> g[maxn];

int N, M, s[maxn], val[maxn];

inline int bitcount (int x) {

return x == 0 ? x : bitcount(x>>1) + (x&1);

}

int SG (int x) {

if (s[x] != -1)

return s[x];

map<int, int> vis;

int n = g[x].size();

if (n == 0)

return s[x] == 0;

for (int i = 0; i < (1<<n); i++) {

int bit = bitcount(i);

if (bit > val[x] || (bit&1) != (val[x]&1))

continue;

int tmp = 0;

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

if (i&(1<<j))

tmp ^= SG(g[x][j]);

}

vis[tmp] = 1;

}

int ret = -1;

while (vis.count(++ret));

return s[x] = ret;

}

void init () {

scanf("%d%d", &N, &M);

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

g[i].clear();

int u, v;

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

scanf("%d%d", &u, &v);

g[u].push\_back(v);

}

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

scanf("%d", &val[i]);

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

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

s[i] = SG(i);

}

void solve () {

int Q;

scanf("%d", &Q);

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

int ret = 0, x;

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

scanf("%d", &x);

if (x&1)

ret ^= s[j];

}

printf("Round#%d: %s\n", i, ret ? "WINNING" : "LOSING");

}

printf("\n");

}

int main () {

int cas;

scanf("%d", &cas);

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

init();

printf("Game#%d:\n", k);

solve();

}

return 0;

}

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

五、其他

1. 矩阵快速幂

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*pow\_mat.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

const int maxn = 105;

const int mod = 1e9 + 7;

struct Mat {

int r, c, s[maxn][maxn];

void init(int r, int c) {

this->r = r;

this->c = c;

memset(s, 0, sizeof(s));

}

}tmp;

void mul(const Mat& a, const Mat& b, Mat& c) {

tmp.init(a.r, b.c);

for (int i = 0; i < tmp.r; i++) {

for (int j = 0; j < tmp.c; j++)

for (int k = 0; k < a.c; k++)

tmp.s[i][j] = (tmp.s[i][j] + 1LL \* a.s[i][k] \* b.s[k][j] % mod) % mod;

}

c = tmp;

}

Mat pow\_mat(Mat ret, Mat x, int n) {

while (n) {

if (n&1) mul(x, ret, ret);

mul(x, x, x);

n >>= 1;

}

return ret;

}

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

1. 高斯消元

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*guass\_elimination.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <cmath>

#include <algorithm>

using namespace std;

const int maxn = 30;

const double eps = 1e-9;

typedef double Mat[maxn+5][maxn+5];

void gauss\_elimination (Mat a, int n) {

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

int r = i;

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

if (fabs(a[j][i]) > fabs(a[r][i]))

r = j;

if (r != i) {

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

swap(a[r][j], a[i][j]);

}

if (fabs(a[i][i]) < 1e-9)

continue;

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

double f = a[k][i] / a[i][i];

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

a[k][j] -= a[i][j] \* f;

}

}

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

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

a[i][n] -= a[j][j] \* a[i][j];

a[i][i] = a[i][n] / a[i][i];

if (fabs(a[i][i]) < 1e-9)

a[i][i] = 0;

}

}

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

1. 数值方法

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Simpson.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <cmath>

#include <algorithm>

using namespace std;

double F(double x) { return x; }

double simpson(double a, double b) {

double c = (b + a) / 2;

return (F(a) + 4 \* F(c) + F(b)) \* (b - a) / 6;

}

double asr(double a, double b, double ep, double A) {

double c = (a + b) / 2;

double L = simpson(a, c), R = simpson(c, b);

if (fabs(L + R - A) <= 15 \* ep) return L + R + (L + R - A) / 15;

return asr(a, c, ep/2, L) + asr(c, b, ep/2, R);

}

double asr(double a, double b, double ep) {

return asr(a, b, ep, simpson(a, b));

}

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

第二章 数据结构

一、STL+基本数据结构

（一）常用STL语法

1、排序检索：

|  |  |
| --- | --- |
| sort(begin, end, cmp); | 根据cmp函数排序序列 |
| next\_permutation(begin, end, cmp); | 获取全排列中当前序列的下一个序列 |
| reverse(begin, end, cmp); | 翻转整个序列 |
| unique(begin,end); | 将这个数组去重，需要先排序 |
| lower\_bound(begin, end, x); | 返回序列中大于等于x最近的元素的指针 |
| upper\_bound(begin, end, x); | 返回序列中大于x最近的元素的指针 |
| swap(a,b); | 交换a，b的值 |

2、队列+栈：

进行删除或者取首元素时需要考虑是否为空。

（1）队列(queue)

|  |  |
| --- | --- |
| push(); | 将元素入队 |
| pop(); | 队首元素出队 |
| front(); | 取队首元素 |
| size(); | 队列元素个数 |
| empty(); | 判断队列是否为空 |

（2）栈(stack)

|  |  |
| --- | --- |
| push(); | 将元素入栈 |
| pop(); | 栈顶元素出栈 |
| top(); | 取栈顶元素 |
| size(); | 栈元素的个数 |
| empty(); | 判断栈是否为空 |

（3）优先队列(priority\_queue)

需要运算符重载 bool operator < (const Typ& u) const {},重载的是优先级，优先级大的优先出队。

|  |  |
| --- | --- |
| push(); | 将元素入队 |
| pop(); | 队首元素出队 |
| front(); | 取队首元素 |
| size(); | 队列元素个数 |
| empty(); | 判断队列是否为空 |

（4）双端队列（deque）

双端队列的一个应用就是维护单调序列，可以用在一些高效算法的题目降低复杂度。

|  |  |
| --- | --- |
| push\_front(); | 向队列首部添加元素 |
| push\_back(); | 向队列尾部添加元素 |
| pop\_front(); | 队首元素出队 |
| pop\_back(); | 队尾元素出队 |
| size(); | 队列元素个数 |
| empty(); | 判断队列是否为空 |

3、容器：

Type::iterator 迭代器，支持自加自减，通过\*或者->取对应的元素。

（1）vector:可变长度数组。

|  |  |
| --- | --- |
| clear（）; | 清空数组 |
| push\_back(); | 在数组末端添加元素 |
| size(); | 数组元素个数 |
| pop\_back(); | 删除数组末端的元素 |
| erase(iter); | 删除迭代器指向位置元素 |
| begin(); | 返回首元素的地址 |
| end(); | 返回尾元素的地址+1后的地址 |

（2）set/multiset:自动排序集合（去重/不去重）

需要重载bool operator < (const Type& u) const {}

|  |  |
| --- | --- |
| clear(); | 清空set |
| insert(); | 插入元素x |
| count(x); | 统计x元素的个数 |
| size(); | 统计整个set中的元素个数 |
| find(x); | 返回元素x的地址 |
| erase(iter); | 删除iter地址的元素 |
| erase(begin, end); | 删除地址begin到end之间的元素 |
| begin(); | 返回首元素的地址 |
| end(); | 返回尾元素的地址+1后的地址 |

注：如果find(x) = end()，或者count(x) = 0的话，说明set中没有这个x元素。

（3）map/multiset:映射容器（去重/不去重）

需要重载bool operator < (const Type& u) const {}，本质上是一棵红黑树。

|  |  |
| --- | --- |
| iter->fitst | 迭代器iter对应的key值 |
| iter->second | 迭代器iter对应的val值 |
| map[x] = y | 直接通过key值对map元素赋值 |
| clear(); | 清空map |
| count(x); | 统计x元素的个数 |
| size(); | 统计map中总元素的个数 |
| erase(iter); | 删除iter地址的元素 |
| erase(begin, end); | 删除地址begin到end之间的元素 |
| begin(); | 返回首元素的地址 |
| end(); | 返回尾元素的地址+1后的地址 |

（4）string:字符串

|  |  |
| --- | --- |
| length(); | 放回string的长度 |
| + | 拼接两个string，可以拼接char |
| substr(begin = 0, end = n)； | 返回begin到end组成的字符串 |
| >、=、< | 比较两个string的字典序大小 |
| cout | 输出用cout |

（5）bitset:二进制数

|  |  |
| --- | --- |
| bitset<int> | 初始化二进制数位的长度 |
| count()； | 返回位为1的个数 |
| size(); | 返回二进制数位的长度 |
| flip(); | 01置换，0变1，1变0 |
| set(); | 全置为1 |
| set(pos); | 将pos位置置为1 |
| reset(); | 全置为0 |
| operator[] | 重载了[]操作 |
| &、|、^ | 支持二进制数的基本运算 |
| cout | 输出用cout |

（二）高精度

1、C++：

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Bign.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

const int maxn = 10005;

struct bign {

int len, num[maxn];

bign () {

len = 0;

memset(num, 0, sizeof(num));

}

bign (int number) {\*this = number;}

bign (const char\* number) {\*this = number;}

void delzero ();

void Put ();

void operator = (int number);

void operator = (char\* number);

bool operator < (const bign& b) const;

bool operator > (const bign& b) const { return b < \*this; }

bool operator <= (const bign& b) const { return !(b < \*this); }

bool operator >= (const bign& b) const { return !(\*this < b); }

bool operator != (const bign& b) const { return b < \*this || \*this < b;}

bool operator == (const bign& b) const { return !(b != \*this); }

void operator ++ ();

void operator -- ();

bign operator + (const int& b);

bign operator + (const bign& b);

bign operator - (const int& b);

bign operator - (const bign& b);

bign operator \* (const int& b);

bign operator \* (const bign& b);

bign operator / (const int& b);

//bign operator / (const bign& b);

int operator % (const int& b);

};

/\*Code\*/

void bign::delzero () {

while (len && num[len-1] == 0)

len--;

if (len == 0) {

num[len++] = 0;

}

}

void bign::Put () {

for (int i = len-1; i >= 0; i--)

printf("%d", num[i]);

}

void bign::operator = (char\* number) {

len = strlen (number);

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

num[i] = number[len-i-1] - '0';

delzero ();

}

void bign::operator = (int number) {

len = 0;

while (number) {

num[len++] = number%10;

number /= 10;

}

delzero ();

}

bool bign::operator < (const bign& b) const {

if (len != b.len)

return len < b.len;

for (int i = len-1; i >= 0; i--)

if (num[i] != b.num[i])

return num[i] < b.num[i];

return false;

}

void bign::operator ++ () {

int s = 1;

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

s = s + num[i];

num[i] = s % 10;

s /= 10;

if (!s) break;

}

while (s) {

num[len++] = s%10;

s /= 10;

}

}

void bign::operator -- () {

if (num[0] == 0 && len == 1) return;

int s = -1;

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

s = s + num[i];

num[i] = (s + 10) % 10;

if (s >= 0) break;

}

delzero ();

}

bign bign::operator + (const int& b) {

bign a = b;

return \*this + a;

}

bign bign::operator + (const bign& b) {

int bignSum = 0;

bign ans;

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

if (i < len) bignSum += num[i];

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

ans.num[ans.len++] = bignSum % 10;

bignSum /= 10;

}

while (bignSum) {

ans.num[ans.len++] = bignSum % 10;

bignSum /= 10;

}

return ans;

}

bign bign::operator - (const int& b) {

bign a = b;

return \*this - a;

}

bign bign::operator - (const bign& b) {

int bignSub = 0;

bign ans;

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

bignSub += num[i];

bignSub -= b.num[i];

ans.num[ans.len++] = (bignSub + 10) % 10;

if (bignSub < 0) bignSub = -1;

else bignSub = 0;

}

ans.delzero ();

return ans;

}

bign bign::operator \* (const int& b) {

int bignSum = 0;

bign ans;

ans.len = len;

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

bignSum += num[i] \* b;

ans.num[i] = bignSum % 10;

bignSum /= 10;

}

while (bignSum) {

ans.num[ans.len++] = bignSum % 10;

bignSum /= 10;

}

return ans;

}

bign bign::operator \* (const bign& b) {

bign ans;

ans.len = 0;

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

int bignSum = 0;

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

bignSum += num[i] \* b.num[j] + ans.num[i+j];

ans.num[i+j] = bignSum % 10;

bignSum /= 10;

}

ans.len = i + b.len;

while (bignSum){

ans.num[ans.len++] = bignSum % 10;

bignSum /= 10;

}

}

return ans;

}

bign bign::operator / (const int& b) {

bign ans;

int s = 0;

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

s = s \* 10 + num[i];

ans.num[i] = s/b;

s %= b;

}

ans.len = len;

ans.delzero ();

return ans;

}

int bign::operator % (const int& b) {

bign ans;

int s = 0;

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

s = s \* 10 + num[i];

ans.num[i] = s/b;

s %= b;

}

return s;

}

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

注意:除法和取模中，如果遇到值较大，运算过程中可能爆int，那么就要用long long类型去计算。

2、java

（1）头文件

import java.util.\*;

import java.math.\*;

import java.io.\*;

（2）基本框架

public class Main {

//函数格式为public static 类型 函数名() {}

//全局变量为public static 类型 变量名

public static void main(String args[]) {

Scanner cin = new Scanner(System.in);

}

}

（3）读入输出

while (cin.hasNext()) {} // 多组数据的读入格式

cin.nextInt() //读入整型数

cin.next() //读入字符串

int[] a; a = new int[size]; //定义数组

System.out.println(String); //带换行输出

System.out.print(String); //不带换行输出

（4）大数类

高精度数字类型为 BigInteger

高精度浮点数类型为 BigDecimal

|  |  |
| --- | --- |
| add(); | 加法 |
| subtract(); | 减法 |
| multiply(); | 乘法 |
| divide(); | 除法 |
| mod(); | 取模 |
| equals(); | 判相等 |
| compareTo(); | 比较大小，大于返回正，等于返回0，小于返回负 |
| toString(); | 转换成字符串 |
| toPlainString(); | BigDecimal保持高精度输出 |
| stripTrailingZeros().toPlainString() | BigDecimal保持高精度输出 |

（5）初始化

a = new BigType(Type)；

a = BigType.valueOf(Type);

a = cin.next BigType();

3、分数

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Fraction.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

typedef long long type;

struct Fraction {

type mem; // 分子；

type den; // 分母；

bool operator < (const Fraction& u) const;

bool operator > (const Fraction& u) const { return u < \*this; }

bool operator <= (const Fraction& u) const { return !(u < \*this); }

bool operator >= (const Fraction& u) const { return !(\*this < u); }

bool operator != (const Fraction& u) const { return u < \*this || \*this > u; }

bool operator == (const Fraction& u) const { return !(u != \*this); }

Fraction (type mem = 0, type den = 1);

void operator = (type x) { this->set(x, 1); }

Fraction operator \* (const Fraction& u);

Fraction operator / (const Fraction& u);

Fraction operator + (const Fraction& u);

Fraction operator - (const Fraction& u);

void set(type mem, type den);

void put () {

if (mem == 0) {

printf("0");

} else {

printf("%lld", mem);

if (den != 1)

printf("/%lld", den);

}

printf(" ");

}

};

inline type gcd (type a, type b) {

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

}

inline type lcm (type a, type b) {

return a / gcd(a, b) \* b;

}

/\*\*\*\*\* Code \*\*\*\*\*\*/

bool Fraction::operator < (const Fraction& u) const {

return mem \* u.den < u.mem \* den;

}

Fraction::Fraction (type mem, type den) {

this->set(mem, den);

}

Fraction Fraction::operator \* (const Fraction& u) {

type tmp\_p = gcd(mem, u.den);

type tmp\_q = gcd(u.mem, den);

return Fraction( (mem / tmp\_p) \* (u.mem / tmp\_q), (den / tmp\_q) \* (u.den / tmp\_p) );

}

Fraction Fraction::operator / (const Fraction& u) {

type tmp\_p = gcd(mem, u.mem);

type tmp\_q = gcd(den, u.den);

return Fraction( (mem / tmp\_p) \* (u.den / tmp\_q), (den / tmp\_q) \* (u.mem / tmp\_p));

}

Fraction Fraction::operator + (const Fraction& u) {

type tmp\_l = lcm (den, u.den);

return Fraction(tmp\_l / den \* mem + tmp\_l / u.den \* u.mem, tmp\_l);

}

Fraction Fraction::operator - (const Fraction& u) {

type tmp\_l = lcm (den, u.den);

return Fraction(tmp\_l / den \* mem - tmp\_l / u.den \* u.mem, tmp\_l);

}

void Fraction::set (type mem, type den) {

if (den == 0) {

den = 1;

mem = 0;

}

type tmp\_d = gcd(mem, den);

this->mem = mem / tmp\_d;

this->den = den / tmp\_d;

}

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

（三）辅助

1、输入外挂：

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*qscanf.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void scanf\_(int &num) {

char in;

bool neg=false;

while(((in=getchar()) > '9' || in<'0') && in!='-') ;

if(in=='-') {

neg=true;

while((in=getchar()) >'9' || in<'0');

}

num=in-'0';

while(in=getchar(), in>='0'&&in<='9')

num \*= 10,num += in-'0';

if (neg)

num = -num;

}

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

2、手动括栈：

#pragma comment(linker, "/STACK:1024000000,1024000000")

二、树形数据结构

（一）并查集（union-find）

f[i]表示i节点父亲节点，初始化f[i]=i。

|  |  |
| --- | --- |
| find(x); | 查询x节点的根节点 |
| union(u,v); | 合并u和v所在的两棵子树 |

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Union-find.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

const int maxn = 1e5 + 5;

/\*\*\*\*\*\*\*\*\*\*\*\*\*递归版\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int f[maxn];

void UF\_init(int n) {

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

f[i] = i;

}

int UF\_find(int x) {

return x == f[x] ? x : f[x] = UF\_find(f[x]);

}

void UF\_union(int u, int v) {

int fu = UF\_find(u);

int fv = UF\_find(v);

if (fu != fv)

f[fu] = fv;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*加权版\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int f[maxn], r[maxn], relat = 3;

void UF\_init(int n) {

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

f[i] = 1;

r[i] = 0;

}

}

int UF\_find(int x) {

if (f[x] == x)

return x;

int fx = UF\_find(x);

// According Problem;

r[x] = (r[x] + r[fx]) % relat;

return f[x] = fx;

}

bool UF\_union(int u, int v, int rel) {

int fu = UF\_find(u);

int fv = UF\_find(v);

if (fu != fv) {

// According Problem;

r[fu] = ((r[v] + rel - r[u]) % relat + relat) % relat;

f[fu] = fv;

} else

return ((f[u] - f[v]) % relat + relat) % relat == rel;

return true;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*按秩合并\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

\*\*按秩合并\*\*为并查集的一种优化，在合并两个集合时，将集合元素个数少的合并到集合个数多的。root为1时，该节点为根节点，此时parent记录的为集合元素的个数了；root为0时，该节点为非根节点，parent记录的为它的父亲节点。

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

struct UFSet {

int parent, root;

}f[maxn];

void UF\_init(int n) {

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

f[i].parent = 1;

f[i].root = 1;

}

}

int UF\_find(int x) {

int p, q, tmp;

p = q = x;

while (!f[p].root)

p = f[p].parent;

while (q != p) {

tmp = f[q].parent;

f[q].parent = p;

q = tmp;

}

return p;

}

void UF\_union(int u, int v) {

int fu = UF\_find(u);

int fv = UF\_find(v);

if (f[fu].parent < f[fv].parent)

swap(fu, fv);

f[fu].parent += f[fv].parent;

f[fv].parent = fu;

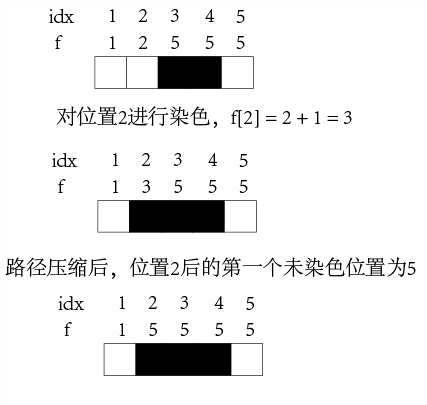
f[fv].root = 0;

}

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

应用：

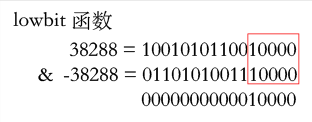
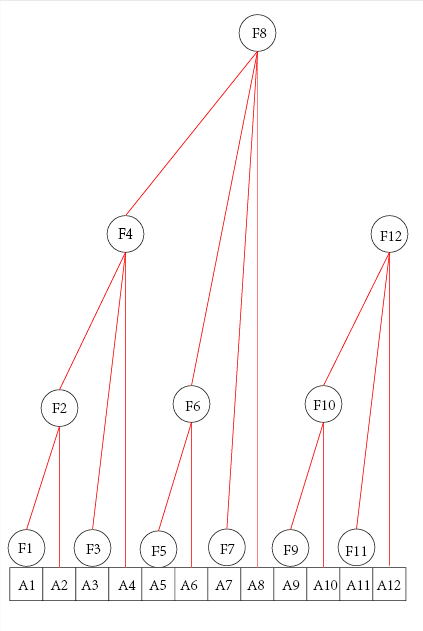
1. 判断图是否联通
2. 确定联通分量个数（每进行一次Union联通分量减少1）
3. 染色问题（可行域查找优化）



（二）树状数组(fenwick):动态连续和查询问题。

fenwick数组下标从1开始，如果需要维护0位置的值，需将整体下标+1。

|  |  |
| --- | --- |
| add(x,d); | 对序列中第x个元素增加d |
| sum(x); | 计算从第一个元素到第x个元素的区间和 |
| find(x); | 解决第K大元素问题时查询第x大的元素值 |
| lowbit(x); | 取x对应二进制下的最后一位 |



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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Fenwick-Tree.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define lowbit(x) ((x)&(-x))

const int maxn = 1e3;

/\*\*\*\*\*\*\*\*\*\*\*\*\*一维\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int fenw[maxn+5];

void fenwick\_add (int x, int v) {

while (x <= maxn) {

fenw[x] += v;

x += lowbit(x);

}

}

int fenwick\_sum (int x) {

int ret = 0;

while (x) {

ret += fenw[x];

x -= lowbit(x);

}

return ret;

}

int fenwick\_find (int x) {

int p = 0, ret = 0;

for (int i = 20; i >= 0; i--) {

p += (1<<i);

if (p > maxn || ret + fenw[p] >= x)

p -= (1<<i);

else

ret += fenw[p];

}

return p + 1;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*二维\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int fenw[maxn+5][maxn+5];

void fenwick\_add (int x, int y, int a) {

for (int i = x; i <= maxn; i += lowbit(i)) {

for (int j = y; j <= maxn; j += lowbit(j))

fenw[i][j] += a;

}

}

int fenwick\_sum(int x, int y) {

int ret = 0;

for (int i = x; i; i -= lowbit(i)) {

for (int j = y; j; j -= lowbit(j))

ret += fenw[i][j];

}

return ret;

}

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

应用：

1. 单点修改区间查询：

add(x，v); sum(r)-sum(l);

1. 区间修改单点查询：

add(l,v),add(r+1,-v); sum(x);

1. 二维树状数组维护矩形权值：

例题：hdu 4456（离散化加二维树状数组）

题目大意：给定N，然后M次操作

* 1 x y z：在x，y的位置加z
* 2 x y z：询问与x，y曼哈顿距离小于z的点值和。

解题思路：将矩阵旋转45度，然后询问就等于是询问一个矩形，使用容斥定理，维护用二维树状数组，但是空间开不下；离散化，将有用到的点处理出来。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*hdu4456.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

const int maxn = 4000005;

const int maxm = 80005;

#define lowbit(x) ((x)&(-x))

int N, M, W, E, H[maxn+5], fenw[maxn + 5];

int O[maxm], X[maxm], Y[maxm], Z[maxm];

inline int find (int x) {

return lower\_bound(H + 1, H + E, x) - H;

}

void hashPoint (int x, int y) {

for (int i = x; i <= W; i += lowbit(i)) {

for (int j = y; j <= W; j += lowbit(j))

H[E++] = i \* W + j;

}

}

void add(int x, int y, int d) {

for (int i = x; i <= W; i += lowbit(i)) {

for (int j = y; j <= W; j += lowbit(j)) {

int pos = find(i \* W + j);

fenw[pos] += d;

}

}

}

int sum (int x, int y) {

int ret = 0;

for (int i = x; i; i -= lowbit(i)) {

for (int j = y; j; j -= lowbit(j)) {

int pos = find(i \* W + j);

if (H[pos] == i \* W + j)

ret += fenw[pos];

}

}

return ret;

}

void init () {

E = 1;

W = 2 \* N;

scanf("%d", &M);

memset(fenw, 0, sizeof(fenw));

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

scanf("%d%d%d%d", &O[i], &X[i], &Y[i], &Z[i]);

int x = X[i] - Y[i] + N;

int y = X[i] + Y[i];

if (O[i] == 1)

hashPoint(x, y);

}

sort(H + 1, H + E);

E = unique(H + 1, H + E) - H;

}

void solve() {

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

int x = X[i] - Y[i] + N;

int y = X[i] + Y[i];

if (O[i] == 1)

add(x, y, Z[i]);

else {

int a = max(1, x - Z[i]);

int b = max(1, y - Z[i]);

int c = min(W, x + Z[i]);

int d = min(W, y + Z[i]);

printf("%d\n", sum(c, d) - sum(c, b-1) - sum(a-1, d) + sum(a-1, b-1));

}

}

}

int main () {

while (scanf("%d", &N) == 1 && N) {

init();

solve();

}

return 0;

}

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

input

8 5

1 8 8 1

1 1 1 -2

2 5 5 6

1 5 5 3

2 2 3 9

3 2

1 3 2 -9

2 3 2 0

0

output

1

1

-9

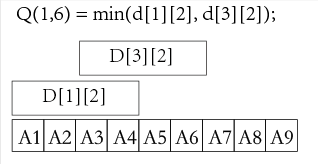
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

（三）静态区间最值查询(RMQ)：范围最值问题。

预处理d[i][j]数组，加速区间最值查询的速度，不支持动态修改。d[i][j]表示从i到i+2^j区间内最值。

|  |  |
| --- | --- |
| init(); | 初始化d数组 |
| query(L,R); | 查询区间L，R的最值 |



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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RMQ.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

const int maxn = 1e5;

const int maxr = 20;

/\*\*\*\*\*\*\*\*\*\*\*\*\*一维\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int d[maxn][maxr];

void rmq\_init (const vector<int>& arr) {

int n = arr.size();

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

d[i][0] = arr[i];

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

for (int i = 0; i + (1<<j) - 1 < n; i++)

d[i][j] = min(d[i][j-1], d[i + (1<<(j-1))][j-1]);

}

}

int rmq\_query (int l, int r) {

int k = 0;

while ((1<<(k+1)) <= r - l + 1)

k++;

return min(d[l][k], d[r-(1<<k)+1][k]);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*二维\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int N, M, Q, g[maxn][maxn], dp[maxn][maxn][9][9];

void rmq\_init(int n, int m) {

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

for (int j = 1; j <= m; j++)

dp[i][j][0][0] = g[i][j];

}

for (int x = 0; (1<<x) <= n; x++)

for (int y = 0; (1<<y) <= m; y++)

if (x + y)

for (int i = 1; i + (1<<x) - 1 <= n; i++)

for (int j = 1; j + (1<<y) - 1 <= m; j++) {

if (x)

dp[i][j][x][y] = max(dp[i][j][x-1][y], dp[i+(1<<(x-1))][j][x-1][y]);

else

dp[i][j][x][y] = max(dp[i][j][x][y-1], dp[i][j+(1<<(y-1))][x][y-1]);

}

}

int rmq\_query(int x1, int y1, int x2, int y2) {

int x = 0, y = 0;

while ((1<<(x+1)) <= x2 - x1 + 1) x++;

while ((1<<(y+1)) <= y2 - y1 + 1) y++;

x2 = x2 - (1<<x) + 1;

y2 = y2 - (1<<y) + 1;

return max( max(dp[x1][y1][x][y], dp[x2][y1][x][y]), max(dp[x1][y2][x][y], dp[x2][y2][x][y]));

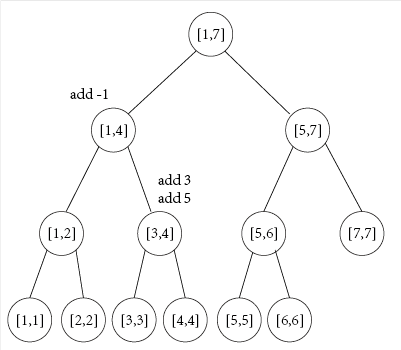
}

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

（四）线段树：动态区间修改查询。

动态维护区间的修改和查询。节点的权值可以直接由左右孩子节点的权值计算得到，延迟操作时可以直接计算出节点的权值。线段树的节点个数需要多开4倍，防止数组越界。

|  |  |
| --- | --- |
| build(u,l,r); | 以u为根节点，对区间[l,r]建树 |
| query(u,l,r); | 查询区间[l，r]的值 |
| insert(u,l,r,v); | 对区间[l,r]上的每个元素增加v |
| pushup(u); | 更新u节点的权值 |
| pushdown(u); | 将u节点的延迟操作向下延迟一层 |



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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Segment-Tree.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

const int maxn = 1e5 + 5;

#define lson(x) ((x)<<1)

#define rson(x) (((x)<<1)|1)

int lc[maxn << 2], rc[maxn << 2], nd[maxn << 2], ad[maxn << 2];

inline void maintain (int u, int w) {

// According Problem;

nd[u] += ad[u] \* (rc[u] - lc[u] + 1);

}

void pushup (int u) {

// According Problem;

nd[u] = nd[lson(u)] + nd[rson(u)];

}

void pushdown (int u) {

// According Problem;

if (ad[u]) {

maintain(lson(u), ad[u]);

maintain(rson(u), ad[u]);

ad[u] = 0;

}

}

void segtree\_build (int u, int l, int r) {

lc[u] = l;

rc[u] = r;

nd[u] = ad[u] = 0;

if (l == r) {

// According Problem;

return;

}

int mid = (l + r) / 2;

segtree\_build(lson(u), l, mid);

segtree\_build(rson(u), mid + 1, r);

pushup(u);

}

void segtree\_modify (int u, int l, int r, int w) {

if (l <= lc[u] && rc[u] <= r) {

// According Problem;

maintain(u, w);

return ;

}

pushdown(u);

int mid = (lc[u] + rc[u]) >> 1;

if (l <= mid)

segtree\_modify(lson(u), l, r, w);

if (r > mid)

segtree\_modify(rson(u), l, r, w);

pushup(u);

}

int segtree\_query (int u, int l, int r) {

if (l <= lc[u] && rc[u] <= r)

return nd[u];

pushdown(u);

int mid = (lc[u] + rc[u]) >> 1, ret = 0;

if (l <= mid)

ret += segtree\_query(lson(u), l, r);

if (r > mid)

ret += segtree\_query(rson(u), l, r);

pushup(u);

return ret;

}

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

应用：

1. 区间合并：一般解决类似一个区间有多少块连续1，连续1的最大长度等问题。

注：L表示区间左端连续1的个数，R表示区间右端连续1的个数，C表示区间上连续有多少段连续的1，S表示区间上最长连续1的长度。

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

void pushup(int u) {

C[u] = C[lson(u)] + C[rson(u)] + (R[lson(u)] && L[rson(u)] ? -1 : 0);

S[u] = max(max(S[lson(u)], S[rson(u)]), R[lson(u)] + L[rson(u)]);

L[u] = L[lson(u)] + (L[lson(u)] == rc[lson(u)] - lc[lson(u)] + 1 ? L[rson(u)] : 0);

R[u] = R[rson(u)] + (R[rson(u)] == rc[rson(u)] - lc[rson(u)] + 1 ? R[lson(u)] : 0);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*区间上多少段连续的1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int query(int u, int l, int r) {

if (l <= lc[u] && rc[u] <= r)

return C[u];

pushdown(u);

int mid = (lc[u] + rc[u]) >> 1, ret;

if (r <= mid)

ret = query(lson(u), l, r);

else if (l > mid)

ret = query(rson(u), l, r);

else

ret = query(lson(u), l, r) + query(rson(u), l, r) - (R[lson(u)] && L[rson(u)] ? -1 : 0);

pushup(u);

return ret;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*区间上最长连续1的长度\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int query (int u, int l, int r) {

if (l <= lc[u] && rc[u] <= r)

return S[u];

pushdown(u);

int mid = (lc[u] + rc[u]) >> 1, ret;

if (r <= mid)

ret = query(lson(u), l, r);

else if (l > mid)

ret = query(rson(u), l, r);

else {

int ll = query(lson(u), l, r);

int rr = query(rson(u), l, r);

int a = min(L[rson(u)], r - mid);

int b = min(R[lson(u)], mid - l + 1);

ret = max(max(ll, rr), a + b);

}

pushup(u);

return ret;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*第K段连续1的区间\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

pii query(int u, int k) {

if (k > C[u])

return make\_pair(0, 0);

pushdown(u);

pii ret;

int mid = (lc[u] + rc[u]) >> 1;

if (C[lson(u)] == k && R[lson(u)])

ret = make\_pair(mid - R[lson(u)] + 1, mid + L[rson(u)]);

else if (C[lson(u)] <= k)

ret = query(lson(u), k);

else

ret = query(rson(u), k - C[lson(u)] + (R[lson(u)] && L[rson(u)] ? 1 : 0));

pushup(u);

return ret;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*满足长度k的最左区间\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int query (int u, int k) {

if (S[u] < k)

return 0;

pushdown(u);

int ret;

if (S[lson(u)] >= k)

ret = query(lson(u), k);

else if (R[lson(u)] + L[rson(u)] >= k) {

int mid = (lc[u] + rc[u]) / 2;

ret = mid - R[lson(u)] + 1;

} else

ret = query(rson(u), k);

pushup(u);

return ret;

}

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

例题：Codeforces 484E（可持久化线段处理区间合并）

题目大意：给定给一个序列，每个位置有一个值，表示高度，现在有若干查询，每次查询l，r，w，表示在区间l，r中，

连续最长长度大于w的最大高度为多少。

解题思路：将高度按照从大到小的顺序排序，然后每次插入一个位置，线段树维护最长连续区间，因为插入是按照从大到小的顺序，所以每次的线段树中的连续最大长度都是满足高度大于等于当前新插入的height值。对于每次查询，二分高度，因为高度肯定是在已有的高度中，所以只接二分下标即可。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CF484E.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <vector>

#include <algorithm>

using namespace std;

const int maxn = 1e6 + 5;

typedef pair<int,int> pii;

struct Node {

int lc, rc, lp, rp, L, R, S;

int length() {

return rp - lp + 1;

}

}nd[maxn << 2];

int N, sz, root[maxn];

pii blo[maxn];

inline int newNode() {

return sz++;

}

inline void pushup(int u) {

int lcid = nd[u].lc, rcid = nd[u].rc;

nd[u].L = nd[lcid].L + (nd[lcid].L == nd[lcid].length() ? nd[rcid].L : 0);

nd[u].R = nd[rcid].R + (nd[rcid].R == nd[rcid].length() ? nd[lcid].R : 0);

nd[u].S = max(nd[lcid].R + nd[rcid].L, max(nd[lcid].S, nd[rcid].S));

}

inline Node merge(Node a, Node b) {

Node u;

u.lp = a.lp; u.rp = b.rp;

u.L = a.L + (a.L == a.length() ? b.L : 0);

u.R = b.R + (b.R == b.length() ? a.R : 0);

u.S = max(a.R + b.L, max(a.S, b.S));

return u;

}

void build(int& u, int l, int r) {

if (u == 0) u = newNode();

nd[u] = (Node){0, 0, l, r, 0, 0, 0};

if (l == r)

return;

int mid = (l + r) >> 1;

build(nd[u].lc, l, mid);

build(nd[u].rc, mid +1, r);

pushup(u);

}

int insert(int u, int x) {

int k = newNode();

nd[k] = nd[u];

if (nd[k].lp == x && x == nd[k].rp) {

nd[k].S = nd[k].L = nd[k].R = 1;

return k;

}

int mid = (nd[k].lp + nd[k].rp) >> 1;

if (x <= mid)

nd[k].lc = insert(nd[k].lc, x);

else

nd[k].rc = insert(nd[k].rc, x);

pushup(k);

return k;

}

Node query(int u, int l, int r) {

if (l <= nd[u].lp && nd[u].rp <= r)

return nd[u];

int mid = (nd[u].lp + nd[u].rp) >> 1;

if (r <= mid)

return query(nd[u].lc, l, r);

else if (l > mid)

return query(nd[u].rc, l, r);

else {

Node ll = query(nd[u].lc, l, r);

Node rr = query(nd[u].rc, l, r);

return merge(ll, rr);

}

}

inline bool cmp (const pii& a, const pii& b) {

return a.first > b.first;

}

void init () {

sz = 1;

scanf("%d", &N);

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

scanf("%d", &blo[i].first);

blo[i].second = i;

}

sort(blo + 1, blo + 1 + N, cmp);

build(root[0], 1, N);

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

root[i] = insert(root[i-1], blo[i].second);

}

int main () {

init();

int q, l, r, w;

scanf("%d", &q);

while (q--) {

scanf("%d%d%d", &l, &r, &w);

int L = 1, R = N;

while (L < R) {

int mid = (L + R) >> 1;

if (query(root[mid], l, r).S >= w)

R = mid;

else

L = mid + 1;

}

printf("%d\n", blo[L].first);

}

return 0;

}

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

input

5

1 2 2 3 3

3

2 5 3

2 5 2

1 5 5

output

2

3

1

input

1

1000000000

1

1 1 1

output

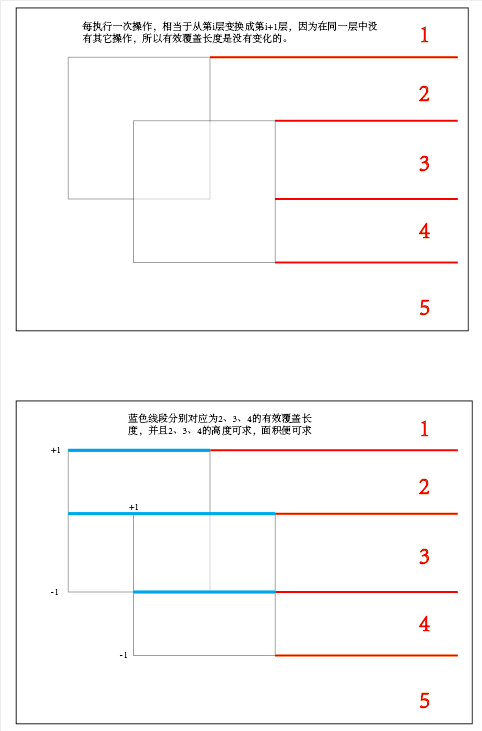
1000000000

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

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

1. 扫描线：进行区间修改，但是没有向下更新。一般问题就是求矩形面积的并，周长的并，或者是覆盖k次的面积。

注：因为横坐标可能会很大，所以很多时候要离散化处理；线段树的维护中不能有pushdown操作，因为每一段区间的加都对应着减，如果将先前的加操作向下更新，那么后面的减操作将导致有负数的存在。



例题：Poj 2482（统计矩形内最大点个数问题转化）

题目大意：平面上有N个星星，问一个W∗H的矩形最多能括进多少个星星。

解题思路：只要以每个点为左上角，建立矩形，这个矩形即为框框左下角放的位置可以括到该点，那么N个星星就有N个矩形，扫描线处理哪些位置覆盖次数最多。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*poj2482.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <vector>

#include <algorithm>

using namespace std;

typedef long long ll;

const int maxn = 40000;

#define lson(x) ((x)<<1)

#define rson(x) (((x)<<1)|1)

int lc[maxn << 2], rc[maxn << 2];

ll v[maxn << 2], s[maxn << 2];

inline void pushup (int u) {

s[u] = max(s[lson(u)], s[rson(u)]) + v[u];

}

inline void maintain (int u, int d) {

v[u] += d;

pushup(u);

}

void build (int u, int l, int r) {

lc[u] = l;

rc[u] = r;

v[u] = s[u] = 0;

if (l == r)

return;

int mid = (l + r) / 2;

build(lson(u), l, mid);

build(rson(u), mid + 1, r);

pushup(u);

}

void modify (int u, int l, int r, int d) {

if (l <= lc[u] && rc[u] <= r) {

maintain(u, d);

return;

}

int mid = (lc[u] + rc[u]) / 2;

if (l <= mid)

modify(lson(u), l, r, d);

if (r > mid)

modify(rson(u), l, r, d);

pushup(u);

}

struct Seg {

ll x, l, r, d;

Seg (ll x = 0, ll l = 0, ll r = 0, ll d = 0) {

this->x = x;

this->l = l;

this->r = r;

this->d = d;

}

friend bool operator < (const Seg& a, const Seg& b) {

return a.x < b.x;

}

};

int N, W, H;

vector<ll> pos;

vector<Seg> vec;

inline int find (ll k) {

return lower\_bound(pos.begin(), pos.end(), k) - pos.begin();

}

void init () {

ll x, y, d;

pos.clear();

vec.clear();

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

scanf("%lld%lld%lld", &x, &y, &d);

pos.push\_back(y - H);

pos.push\_back(y);

vec.push\_back(Seg(x - W, y - H, y, d));

vec.push\_back(Seg(x, y - H, y, -d));

}

sort(pos.begin(), pos.end());

sort(vec.begin(), vec.end());

}

ll solve () {

ll ret = 0;

build (1, 0, pos.size());

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

modify(1, find(vec[i].l), find(vec[i].r) - 1, vec[i].d);

ret = max(ret, s[1]);

}

return ret;

}

int main () {

while (scanf("%d%d%d", &N, &W, &H) == 3) {

init();

printf("%lld\n", solve());

}

return 0;

}

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

input

3 5 4

1 2 3

2 3 2

6 3 1

3 5 4

1 2 3

2 3 2

5 3 1

output

5

6

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

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

例题：Uva 11983（矩形覆盖K次面积问题）

题目大意：给定n个矩形，问说有多少区域的面积被覆盖k次以上。

解题思路：将每个矩形差分成两条线段，一段为添加覆盖值1，一段为减少覆盖值1，同时记录两段的高度（横坐标）。然后对纵坐标离散化建立线段树，然后对线段按照高度排序，维护整段区间中覆盖度大于K的长度，乘上高度上的范围即可。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*uva11983.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <vector>

#include <algorithm>

using namespace std;

const int maxn = 30005;

#define lson(x) ((x)<<1)

#define rson(x) (((x)<<1)+1)

typedef long long ll;

int N, K, R;

vector<int> bins, arr;

struct Segment {

int l, r, h, v;

void set (int l, int r, int h, int v) {

this->l = l;

this->r = r;

this->h = h;

this->v = v;

}

}seg[maxn\*2];

struct Node {

int l, r, add;

int c[12];

void set (int l, int r, int add) {

this->l = l;

this->r = r;

this->add = add;

memset(c, 0, sizeof(c));

}

}node[maxn\*24];

inline bool cmp (const Segment& a, const Segment& b) {

return a.h < b.h;

}

inline int search (int v) {

return lower\_bound(bins.begin(), bins.end(), v) - bins.begin();

}

void pushup (int u) {

memset(node[u].c, 0, sizeof(node[u].c));

if (node[u].l == node[u].r) {

int x = node[u].l;

node[u].c[min(K, node[u].add)] = bins[x+1] - bins[x];

} else {

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

int t = min(K, i + node[u].add);

node[u].c[t] += node[lson(u)].c[i] + node[rson(u)].c[i];

}

}

}

void build\_segTree (int u, int l, int r) {

node[u].set(l, r, 0);

if (l == r) {

pushup(u);

return;

}

int mid = (l + r) / 2;

build\_segTree(lson(u), l, mid);

build\_segTree(rson(u), mid + 1, r);

pushup(u);

}

void insert\_segTree (int u, int l, int r, int v) {

if (l <= node[u].l && node[u].r <= r) {

node[u].add += v;

pushup(u);

return;

}

int mid = (node[u].l + node[u].r) / 2;

if (l <= mid)

insert\_segTree(lson(u), l, r, v);

if (r > mid)

insert\_segTree(rson(u), l, r, v);

pushup(u);

}

int query\_segTree (int u, int l, int r) {

if (l <= node[u].l && node[u].r <= r)

return node[u].c[K];

int mid = (node[u].l + node[u].r) / 2;

int ret = 0;

if (l <= mid)

ret += query\_segTree(lson(u), l, r);

if (r > mid)

ret += query\_segTree(rson(u), l, r);

return ret;

}

void add\_hash (int x) {

if (x)

arr.push\_back(x-1);

arr.push\_back(x);

arr.push\_back(x+1);

}

void init () {

int x1, y1, x2, y2;

arr.clear();

scanf("%d%d", &N, &K);

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

scanf("%d%d%d%d", &x1, &y1, &x2, &y2);

seg[lson(i)].set(x1, x2, y1, 1);

seg[rson(i)].set(x1, x2, y2+1, -1);

add\_hash(x1);

add\_hash(x2);

}

sort(arr.begin(), arr.end());

bins.clear();

bins.push\_back(arr[0]);

for (int i = 1; i < arr.size(); i++) {

if (arr[i] != arr[i-1])

bins.push\_back(arr[i]);

}

R = bins.size() - 2;

build\_segTree (1, 0, R);

}

ll solve () {

sort(seg, seg + 2 \* N, cmp);

ll ret = 0;

for (int i = 0; i < 2 \* N - 1; i++) {

insert\_segTree(1, search(seg[i].l), search(seg[i].r), seg[i].v);

ll tmp = query\_segTree(1, 0, R);

ret += tmp \* (seg[i+1].h - seg[i].h);

}

return ret;

}

int main () {

int cas;

scanf("%d", &cas);

for (int kcas = 1; kcas <= cas; kcas++) {

init();

printf("Case %d: %lld\n", kcas, solve());

}

return 0;

}

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

input

2

2 1

0 0 4 4

1 1 2 5

2 2

0 0 4 4

1 1 2 5

output

Case 1: 27

Case 2: 8

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

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

1. 其它题型：

例题：Uva 12436（维护等差数列和问题）

题目大意：四种操作。

void A( int st, int nd ) {

for( int i = st; i <= nd; i++ ) data[i] = data[i] + (i - st + 1);

}

void B( int st, int nd ) {

for( int i = st; i <= nd; i++ ) data[i] = data[i] + (nd - i + 1);

}

void C( int st, int nd, int x ) {

for( int i = st; i <= nd; i++ ) data[i] = x;

}

long long S( int st, int nd ) {

long long res = 0;

for( int i = st; i <= nd; i++ ) res += data[i];

return res;

}

解题思路：即用线段树维护一个等差数列，因为一个等差加上一个等差还是一个等差数列，所以对于每个节点记录区

间左端的值，也就是首项，以及公差即可。因为还有一个S操作，所以要开一个标记记录区间值是否相同。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*uva12436.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

typedef long long ll;

const int maxn = 250100;

#define lson(x) ((x)<<1)

#define rson(x) (((x)<<1)|1)

int lc[maxn << 2], rc[maxn << 2], v[maxn << 2];

ll nd[maxn << 2], ad[maxn << 2], s[maxn << 2];

void pushup(int u);

void pushdown (int u);

inline int length(int u) {

return rc[u] - lc[u] + 1;

}

inline void change (int u, ll a) {

v[u] = 1;

ad[u] = 0;

nd[u] = a;

s[u] = a \* length(u);

}

inline void maintain (int u, ll a, ll d) {

if (v[u] && lc[u] != rc[u]) {

pushdown(u);

pushup(u);

}

v[u] = 0;

nd[u] += a;

ad[u] += d;

ll n = length(u);

s[u] += a \* n + (((n-1) \* n) / 2) \* d;

}

inline void pushup (int u) {

s[u] = s[lson(u)] + s[rson(u)];

}

inline void pushdown (int u) {

if (v[u]) {

change(lson(u), nd[u]);

change(rson(u), nd[u]);

v[u] = nd[u] = 0;

} else if (nd[u] || ad[u]) {

maintain(lson(u), nd[u], ad[u]);

maintain(rson(u), nd[u] + length(lson(u)) \* ad[u], ad[u]);

nd[u] = ad[u] = 0;

}

}

void build (int u, int l, int r) {

lc[u] = l;

rc[u] = r;

nd[u] = ad[u] = s[u] = 0;

if (l == r) return;

int mid = (l + r) / 2;

build(lson(u), l, mid);

build(rson(u), mid + 1, r);

pushup(u);

}

void modify(int u, int l, int r, ll a, ll d) {

if (l <= lc[u] && rc[u] <= r) {

maintain(u, a + d \* (lc[u] - l), d);

return;

}

pushdown(u);

int mid = (lc[u] + rc[u]) / 2;

if (l <= mid)

modify(lson(u), l, r, a, d);

if (r > mid)

modify(rson(u), l, r, a, d);

pushup(u);

}

void set(int u, int l, int r, ll a) {

if (l <= lc[u] && rc[u] <= r) {

change(u, a);

return;

}

pushdown(u);

int mid = (lc[u] + rc[u]) / 2;

if (l <= mid)

set(lson(u), l, r, a);

if (r > mid)

set(rson(u), l, r, a);

pushup(u);

}

ll query (int u, int l, int r) {

if (l <= lc[u] && rc[u] <= r)

return s[u];

pushdown(u);

ll ret = 0;

int mid = (lc[u] + rc[u]) / 2;

if (l <= mid)

ret += query(lson(u), l, r);

if (r > mid)

ret += query(rson(u), l, r);

pushdown(u);

return ret;

}

int N;

int main () {

while (~scanf("%d", &N)) {

char op[5];

int l, r, x;

build(1, 1, 250000);

while (N--) {

scanf("%s%d%d", op, &l, &r);

if (op[0] == 'A')

modify(1, l, r, 1, 1);

else if (op[0] == 'B')

modify(1, l, r, r - l + 1, -1);

else if (op[0] == 'C') {

scanf("%d", &x);

set(1, l, r, x);

} else

printf("%lld\n", query(1, l, r));

}

}

return 0;

}

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

input

7

A 1 4

B 2 3

S 1 3

C 3 4 -2

S 2 4

B 1 3

S 2 4

output

9

0

3

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

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

例题：Hdu 3333（区间不重复求和）

题目大意：给定一个长度为N的序列，有M次查询，每次查询l，r之间元素的总和，相同元素只算一次。

解题思路：离线操作，将查询按照右区间排序，每次考虑一个询问，将mv ~ r的点全部标记为存在，并且对于每个位置i，如果A[i]在前面已经出现过了，那么将前面的那个位置减掉A[i]，当前位置添加A[i]，这样做维护了每个数尽量做，那么碰到查询，用sum[r] - sum[l-1]即可。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*hdu3333.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <vector>

#include <map>

#include <algorithm>

using namespace std;

typedef long long ll;

const int maxn = 30000;

int N, Q;

ll A[maxn+5], ans[100005];

map<ll, int> G;

#define lson(x) ((x)<<1)

#define rson(x) (((x)<<1)|1)

int lc[maxn << 2], rc[maxn << 2];

ll s[maxn << 2];

inline void pushup (int u) {

s[u] = s[lson(u)] + s[rson(u)];

}

void build (int u, int l, int r) {

lc[u] = l;

rc[u] = r;

s[u] = 0;

if (l == r)

return;

int mid = (lc[u] + rc[u]) / 2;

build(lson(u), l, mid);

build(rson(u), mid + 1, r);

pushup(u);

}

void modify(int u, int x, ll d) {

if (x == lc[u] && rc[u] == x) {

s[u] += d;

return;

}

int mid = (lc[u] + rc[u]) / 2;

if (x <= mid)

modify(lson(u), x, d);

else

modify(rson(u), x, d);

pushup(u);

}

ll query(int u, int l, int r) {

if (l <= lc[u] && rc[u] <= r)

return s[u];

ll ret = 0;

int mid = (lc[u] + rc[u]) / 2;

if (l <= mid)

ret += query(lson(u), l, r);

if (r > mid)

ret += query(rson(u), l, r);

pushup(u);

return ret;

}

struct Seg {

int l, r, id;

Seg (int l = 0, int r = 0, int id = 0) {

this->l = l;

this->r = r;

this->id = id;

}

friend bool operator < (const Seg& a, const Seg& b) {

return a.r < b.r;

}

};

vector<Seg> vec;

void init () {

int l, r;

G.clear();

vec.clear();

scanf("%d", &N);

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

scanf("%I64d", &A[i]);

scanf("%d", &Q);

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

scanf("%d%d", &l, &r);

vec.push\_back(Seg(l, r, i));

}

sort(vec.begin(), vec.end());

}

void solve () {

build (1, 0, N);

int k = 0;

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

for ( ; k <= vec[i].r; k++) {

if (G[A[k]])

modify(1, G[A[k]], -A[k]);

G[A[k]] = k;

modify(1, k, A[k]);

}

ans[vec[i].id] = query(1, vec[i].l, vec[i].r);

}

for (int i = 1; i <= Q; i++)

printf("%I64d\n", ans[i]);

}

int main () {

int cas;

scanf("%d", &cas);

while (cas--) {

init();

solve();

}

return 0;

}

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

input

2

3

1 1 4

2

1 2

2 3

5

1 1 2 1 3

3

1 5

2 4

3 5

output

1

5

6

3

6

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

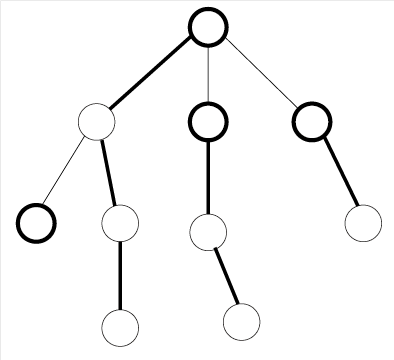
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

（五）树链剖分：维护树的路劲上点权边权问题。

动态维护树上两节点间路径权值的修改和查询。将节点1默认为根节点，通过两次dfs处理出重链和轻链。维护则主要是在重链上操作。

|  |  |
| --- | --- |
| init() | 初始化操作，包括读入边 |
| add\_Edge(u, v); | 填加一条边，以链表的形式储存 |
| dfs(u,pre,d); | 处理出每个节点的far，dep，cnt，son |
| dfs(u,rot); | 处理处每个节点的top，idx |
| modify(u,v,w); | 修改u，v节点之间的路径 |
| query(u,v); | 询问u，v节点之间的路径 |

idx：节点的映射值； dep:节点的深度； top：节点所在重链的根； far：节点的父亲节点；son：与节点在同一条重链的孩子节点； cnt：以节点为根节点的子树的节点数。



注：加粗变为重边，加粗节点为对应重边的根。

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*TreeChain-division.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

const int maxn = 1e5+5;

int N, E, first[maxn], jump[maxn \* 2], link[maxn \* 2], val[maxn];

int id, idx[maxn], dep[maxn], top[maxn], far[maxn], son[maxn], cnt[maxn];

inline void add\_Edge (int u, int v) {

link[E] = v;

jump[E] = first[u];

first[u] = E++;

}

void dfs (int u, int pre, int d) {

far[u] = pre;

dep[u] = d;

cnt[u] = 1;

son[u] = 0;

for (int i = first[u]; i + 1; i = jump[i]) {

int v = link[i];

if (v == pre)

continue;

dfs(v, u, d + 1);

cnt[u] += cnt[v];

if (cnt[son[u]] < cnt[v])

son[u] = v;

}

}

void dfs(int u, int rot) {

top[u] = rot;

idx[u] = ++id;

if (son[u])

dfs(son[u], rot);

for (int i = first[u]; i + 1; i = jump[i]) {

int v = link[i];

if (v == far[u] || v == son[u])

continue;

dfs(v, v);

}

}

void init () {

int u, v;

id = E = 0;

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

/\*\*\* input edge \*\*\*/

dfs(1, 0, 0);

dfs(1, 1);

}

void modify (int u, int v, int w) {

int p = top[u], q = top[v];

while (p != q) {

if (dep[p] < dep[q]) {

swap(p, q);

swap(u, v);

}

/\*\*\* modify idx[p] ~ idx[u] \*\*\*/

u = far[p];

p = top[u];

}

if (dep[u] > dep[v])

swap(u, v);

/\*\*\* 点权 modify idx[u] ~ idx[v] \*\*\*/

/\*\*\* 边权 modify idx[son[u]] ~ idx[v] \*\*\*/

}

int query (int u, int v) {

int p = top[u], q = top[v], ret = 0;

while (p != q) {

if (dep[p] < dep[q]) {

swap(p, q);

swap(u, v);

}

/\*\*\* query idx[p] ~ idx[u] \*\*\*/

u = far[p];

p = top[u];

}

if (dep[u] > dep[v])

swap(u, v);

/\*\*\* 点权 query idx[u] ~ idx[v] \*\*\*/

/\*\*\* 边权 query idx[son[u]] ~ idx[v] \*\*\*/

return ret;

}

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

例题：hdu 4897（树链剖分点边权变形）

题目大意：给定一棵树，每条边有黑白两种颜色，初始都是白色，现在有三种操作：

* 1 u v：u到v路径上的边都取成相反的颜色
* 2 u v：u到v路径上相邻的边都取成相反的颜色（相邻即仅有一个节点在路径上）
* 3 u v：查询u到v路径上有多少个黑色边

解题思路：树链剖分，用两个线段W和L维护，W对应的是每条的黑白情况，L表示的是每个节点的相邻边翻转情况

（对于轻链而言，重链直接在W上修改）

对于1操作，即为普通的树链剖分，直接在W上修改即可。

对于2操作，每次修改在L上，不过链的两端（即重链的部分）还需要在W上修改

对于3操作，每次查询，重链可以直接根据W中的值判断，轻链还要根据对应节点的L值来确定。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*hdu4897.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma comment(linker, "/STACK:1024000000,1024000000")

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

const int maxn = 1e5+5;

#define lson(x) ((x)<<1)

#define rson(x) (((x)<<1)|1)

struct Tree {

int lc[maxn << 2], rc[maxn << 2], fp[maxn << 2], s[maxn << 2];

void splay(int u) {

s[u] = rc[u] - lc[u] + 1 - s[u];

fp[u] ^= 1;

}

void pushup(int u) {

s[u] = s[lson(u)] + s[rson(u)];

}

void pushdown(int u) {

if (fp[u]) {

splay(lson(u));

splay(rson(u));

fp[u] = 0;

}

}

void build (int u, int l, int r) {

lc[u] = l;

rc[u] = r;

fp[u] = s[u] = 0;

if (l == r)

return;

int mid = (l + r) / 2;

build(lson(u), l, mid);

build(rson(u), mid + 1, r);

pushup(u);

}

void modify(int u, int l, int r) {

if (l <= lc[u] && rc[u] <= r) {

splay(u);

return;

}

pushdown(u);

int mid = (lc[u] + rc[u]) / 2;

if (l <= mid)

modify(lson(u), l, r);

if (r > mid)

modify(rson(u), l, r);

pushup(u);

}

int query(int u, int l, int r) {

if (l <= lc[u] && rc[u] <= r)

return s[u];

pushdown(u);

int mid = (lc[u] + rc[u]) / 2, ret = 0;

if (l <= mid)

ret += query(lson(u), l, r);

if (r > mid)

ret += query(rson(u), l, r);

pushup(u);

return ret;

}

}W, L;

int N, Q, E, first[maxn], jump[maxn \* 2], link[maxn \* 2];

int id, idx[maxn], top[maxn], far[maxn], son[maxn], dep[maxn], cnt[maxn];

inline void add\_Edge(int u, int v) {

link[E] = v;

jump[E] = first[u];

first[u] = E++;

}

void dfs (int u, int pre, int d) {

far[u] = pre;

son[u] = 0;

cnt[u] = 1;

dep[u] = d;

for (int i = first[u]; i + 1; i = jump[i]) {

int v = link[i];

if (v == pre)

continue;

dfs(v, u, d + 1);

cnt[u] += cnt[v];

if (cnt[son[u]] < cnt[v])

son[u] = v;

}

}

void dfs (int u, int rot) {

top[u] = rot;

idx[u] = ++id;

if (son[u])

dfs(son[u], rot);

for (int i = first[u]; i + 1; i = jump[i]) {

int v = link[i];

if (v == far[u] || v == son[u])

continue;

dfs(v, v);

}

}

void init () {

E = id = 0;

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

int u, v;

scanf("%d", &N);

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

scanf("%d%d", &u, &v);

add\_Edge(u, v);

add\_Edge(v, u);

}

dfs(1, 0, 0);

dfs(1, 1);

W.build(1, 1, N);

L.build(1, 1, N);

}

void modify(int u, int v, int k) {

int p = top[u], q = top[v];

while (p != q) {

if (dep[p] < dep[q]) {

swap(p, q);

swap(u, v);

}

if (k) {

L.modify(1, idx[p], idx[u]);

W.modify(1, idx[p], idx[p]);

W.modify(1, idx[son[u]], idx[son[u]]);

} else

W.modify(1, idx[p], idx[u]);

u = far[p];

p = top[u];

}

if (dep[u] > dep[v])

swap(u, v);

if (k) {

L.modify(1, idx[u], idx[v]);

W.modify(1, idx[u], idx[u]);

W.modify(1, idx[son[v]], idx[son[v]]);

} else {

if (u == v)

return;

W.modify(1, idx[son[u]], idx[v]);

}

}

int query(int u, int v) {

int p = top[u], q = top[v], ret = 0;

while (p != q) {

if (dep[p] < dep[q]) {

swap(p, q);

swap(u, v);

}

if (u != p)

ret += W.query(1, idx[son[p]], idx[u]);

ret += (W.query(1, idx[p], idx[p]) ^ L.query(1, idx[far[p]], idx[far[p]]));

u = far[p];

p = top[u];

}

if (u == v) return ret;

if (dep[u] > dep[v])

swap(u, v);

ret += W.query(1, idx[son[u]], idx[v]);

return ret;

}

int main () {

int cas;

scanf("%d", &cas);

while (cas--) {

init();

int k, u, v;

scanf("%d", &Q);

while (Q--) {

scanf("%d%d%d", &k, &u, &v);

if (k == 3)

printf("%d\n", query(u, v));

else

modify(u, v, k - 1);

}

}

return 0;

}

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

input

1

10

2 1

3 1

4 1

5 1

6 5

7 4

8 3

9 5

10 6

10

2 1 6

1 3 8

3 8 10

2 3 4

2 10 8

2 4 10

1 7 6

2 7 3

2 1 4

2 10 10

output

3

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

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

（六）最近公共祖先(LCA)

在线算法：far[i][k]表示i节点的第2^k个祖先，每次将询问的节点u和v调整到同一深度，然后从大向小考虑2^i祖先是否相同。

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*LCA.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <vector>

#include <queue>

#include <algorithm>

using namespace std;

typedef pair<int,int> pii;

const int maxn = 10005;

const int inf = 0x3f3f3f3f;

const int BIT = 20;

int dep[maxn], far[maxn][BIT+5];

vector<int> G[maxn];

queue<int> que;

void ST (int n) {

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

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

far[i][k] = far[far[i][k-1]][k-1];

}

memset(dep, inf, sizeof(dep));

for (int i = 1; i <= n; i++) if (far[i][0] == 0) {

dep[i] = 0; que.push(i);

}

while (!que.empty()) {

int u = que.front();

que.pop();

for (int i = 0; i < G[u].size(); i++) {

int v = G[u][i];

if (dep[v] > dep[u] + 1) {

dep[v] = dep[u] + 1;

que.push(v);

}

}

}

}

int LCA(int u, int v) {

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

int mv = dep[v] - dep[u];

for (int i = 0; i <= BIT && mv; mv >>= 1, i++)

if (mv&1) v = far[v][i];

if (u == v) return u;

for (int i = BIT; i >= 0; i--) {

if (far[u][i] == far[v][i]) continue;

u = far[u][i]; v = far[v][i];

}

return far[u][0];

}

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

离线做法：每个节点记录相应的询问，然后在dfs的过程求解答案

例题：hdu 2874（离线LCA）

题目大意：给定一个无向图，边有权值，不存在环，询问两点之间的距离

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*hdu2874.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <vector>

#include <algorithm>

using namespace std;

typedef pair<int,int> pii;

const int maxn = 10005;

const int maxm = 1000005;

int N, M, K, far[maxn], dis[maxn], ans[maxm];

bool mark[maxn], vis[maxn];

vector<pii> G[maxn], Q[maxn];

void init () {

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

G[i].clear();

Q[i].clear();

far[i] = i;

}

int u, v, w;

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

scanf("%d%d%d", &u, &v, &w);

G[u].push\_back(make\_pair(v, w));

G[v].push\_back(make\_pair(u, w));

}

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

scanf("%d%d", &u, &v);

Q[u].push\_back(make\_pair(v, i));

Q[v].push\_back(make\_pair(u, i));

}

memset(vis, 0, sizeof(vis));

memset(mark, 0, sizeof(mark));

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

}

int find(int x) { return x == far[x] ? x : far[x] = find(far[x]); }

void dfs (int u) {

for (int i = 0; i < Q[u].size(); i++) {

int v = Q[u][i].first, idx = Q[u][i].second;

int f = find(v);

if (vis[v] && !mark[f])

ans[idx] = dis[u] - dis[f] + dis[v] - dis[f];

}

for (int i = 0; i < G[u].size(); i++) {

int v = G[u][i].first, w = G[u][i].second;

if (!vis[v]) {

dis[v] = dis[u] + w;

vis[v] = true;

dfs(v);

far[v] = u;

}

}

}

int main () {

while (scanf("%d%d%d", &N, &M, &K) == 3) {

init();

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

if (!vis[i]) {

dis[i] = 0;

vis[i] = true;

dfs(i);

mark[i] = true;

}

}

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

if (ans[i] == -1) printf("Not connected\n");

else printf("%d\n", ans[i]);

}

}

return 0;

}

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

（七）笛卡尔树(treap-tree)

笛卡尔树是一棵二叉树，树的每个节点有两个值，一个为key，一个为val。光看key的话，笛卡尔树是一棵二叉搜索树，每个节点的左子树的key都比它小，右子树都比它大；光看val的话，笛卡尔树有点类似堆，根节点的val是最小（或者最大）的，每个节点的val都比它的子树要大。

在算法竞赛中，val通常用来二叉搜索树的树高，防止树退化成链。

|  |  |
| --- | --- |
| init() | 初始化，设置sizn |
| newNode() | 分配一个新节点 |
| maintain() | 维护一个节点的值 |
| rotate() | 旋转节点，维护树高 |
| insert() | 插入一个值 |
| erase() | 删除一个值 |
| count(int x) | 统计小于x的个数 |

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Treap.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstdlib>

#include <algorithm>

using namespace std;

const int maxn = 1e5 + 5;

struct Node {

int key, val, cnt, siz, ch[2];

}nd[maxn];

int sizn;

int newNode (int key) {

sizn++;

nd[sizn].key = key;

nd[sizn].val = rand();

nd[sizn].cnt = nd[sizn].siz = 1;

nd[sizn].ch[0] = nd[sizn].ch[1] = 0;

return sizn;

}

void maintain(int u) {

nd[u].siz = nd[nd[u].ch[0]].siz + nd[nd[u].ch[1]].siz + nd[u].cnt;

}

void rotate (int& u, int d) {

int k = nd[u].ch[d];

nd[u].ch[d] = nd[k].ch[d^1];

nd[k].ch[d^1] = u;

maintain(u);

maintain(k);

u = k;

}

void insert (int& u, int key) {

if (u == 0) u = newNode(key);

else if (nd[u].key == key) nd[u].cnt++;

else {

int d = nd[u].key < key;

insert(nd[u].ch[d], key);

if (nd[u].val < nd[nd[u].ch[d]].val) rotate(u, d);

}

maintain(u);

}

void erase (int& u, int key) {

if (nd[u].key == key) {

if (nd[u].cnt == 1) {

if (nd[u].ch[0] == 0 && nd[u].ch[1] == 0) {

u = 0; return;

}

rotate(u, nd[nd[u].ch[0]].val < nd[nd[u].ch[1]].val);

erase(u, key);

} else

nd[u].cnt--;

} else

erase(nd[u].ch[nd[u].key < key], key);

maintain(u);

}

int count (int u, int p) {

if (u == 0) return 0;

if (nd[u].key > p) return count(nd[u].ch[0], p);

return nd[u].cnt + nd[nd[u].ch[0]].siz + count(nd[u].ch[1], p);

}

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

例题：hdu 4605（可持久化笛卡尔树维护树形前缀）

题目大意：给定一棵二叉树，保证节点没有孩子节点或者有两个孩子节点，并且每个节点有一个权值W[i]，1为根节点，树给定的方式m个关系u a b，表示u节点的左孩子为a，右孩子为b。现在从根节点放一个权值为X的小球：

* X = W[u]时：小球停留在该节点
* X > W[u]时：小球有1/8的概率移动到左孩子，7/8的概率移动到右孩子
* X < W[u]时：小球有1/2的概率移动到左孩子，1/2的概率移动到右孩子

给定询问，x u，求小球会经过节点u的概率是多少。

解题思路：本题可以用两个树状数组做离线算法。

这里用可持久化笛卡尔树做在线算法。对于每次询问，其实要求的即为节点u到根节点的路径上有多少个点的权值大于小球的质量，并且有多少个点是向右孩子移动的。

笛卡尔树中维护权值以及走向的个数。

以每个节点做一个版本的笛卡尔树，该版本的笛卡尔树维护的是节点u到根节点的情况。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*hdu4605.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <cstdlib>

#include <algorithm>

using namespace std;

const int maxn = 1e5 + 5;

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

int P;

struct Node {

int key, val;

int cnt, siz;

int uri, sri, ch[2];

}nd[maxn \* 20];

int newNode (int key, int rit) {

P++;

nd[P].key = key;

nd[P].val = rand();

nd[P].cnt = nd[P].siz = 1;

nd[P].uri = nd[P].sri = rit;

nd[P].ch[0] = nd[P].ch[1] = 0;

return P;

}

void maintain(int u) {

int ls = nd[u].ch[0], rs = nd[u].ch[1];

nd[u].siz = nd[ls].siz + nd[rs].siz + nd[u].cnt;

nd[u].sri = nd[ls].sri + nd[rs].sri + nd[u].uri;

}

void rotate(int& u, int d) {

int k = nd[u].ch[d];

nd[u].ch[d] = nd[k].ch[d^1];

nd[k].ch[d^1] = u;

maintain(u);

maintain(k);

u = k;

}

int insert(int v, int key, int rit) {

if (v == 0) return newNode(key, rit);

int u = ++P;

nd[u] = nd[v];

if (nd[u].key == key) {

nd[u].cnt++;

if (rit) nd[u].uri++;

} else {

int d = nd[u].key < key;

nd[u].ch[d] = insert(nd[u].ch[d], key, rit);

if (nd[u].val < nd[nd[u].ch[d]].val) rotate(u, d);

}

maintain(u);

return u;

}

bool query(int u, int key, int& x, int& y) {

if (u == 0) return true;

if (nd[u].key == key) return false;

if (nd[u].key > key)

return query(nd[u].ch[0], key, x, y);

int ls = nd[u].ch[0];

x += nd[ls].sri + nd[u].uri;

y += nd[ls].siz + nd[u].cnt;

return query(nd[u].ch[1], key, x, y);

}

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

int N, M, Q, F[maxn], W[maxn], L[maxn], R[maxn];

int T[maxn];

void dfs (int u, int rit) {

if (u == 0) return;

if (u != 1)

T[u] = insert(T[F[u]], W[F[u]], rit);

dfs(L[u], 0);

dfs(R[u], 1);

}

void init () {

scanf("%d", &N);

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

scanf("%d", &W[i]);

memset(L, 0, sizeof(L));

memset(R, 0, sizeof(R));

scanf("%d", &M);

int u, a, b;

while (M--) {

scanf("%d%d%d", &u, &a, &b);

F[a] = F[b] = u;

L[u] = a; R[u] = b;

}

P = T[0] = 0;

nd[P].cnt = nd[P].siz = 0;

nd[P].uri = nd[P].sri = 0;

nd[P].ch[0] = nd[P].ch[1] = 0;

dfs(1, 0);

}

void solve (int u, int w) {

int x = 0, y = 0;

if (!query(T[u], w, x, y))

printf("0\n");

else

printf("%d %d\n", x, y \* 2 + nd[T[u]].siz);

}

int main () {

int cas;

scanf("%d", &cas);

while (cas--) {

init();

scanf("%d", &Q);

int v, x;

while (Q--) {

scanf("%d%d", &v, &x);

solve(v, x);

}

}

return 0;

}

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

三、字符串

（一）基本

1、最长回文判断（manacher）：

传入字符串，返回字符串中最长回文子串的长度;rad和str需要开两倍；起始符和分隔符不能在原字符串中出现。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Manacher.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

const int maxn = 2 \* 1e5 + 5;

int rad[maxn];

char str[maxn];

int manachar(char\* tmpstr) {

int len = strlen(tmpstr), cnt = 0;

str[cnt++] = '$';

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

str[cnt++] = '#';

str[cnt++] = tmpstr[i];

}

int ans = 0, mix = 0, id = 0;

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

if (mix > i)

rad[i] = min(rad[2 \* id - i], mix - i);

else

rad[i] = 1;

for ( ; str[i - rad[i]] == str[i + rad[i]]; rad[i]++) {

if (mix < i + rad[i]) {

mix = i + rad[i];

id = i;

}

}

ans = max(ans, rad[i]);

}

return ans - 1;

}

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

2、最小表示法：

给定一个字符串，字符串首尾相连，求出以x下标起始的字符串字典序最小。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Miniexpress.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int miniexpress(char\* s) {

int n = strlen(s), p = 0, q = 1;

while (p < n && q < n) {

int i;

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

if (s[(p+i)%n] != s[(q+i)%n])

break;

}

if (s[(p+i)%n] > s[(q+i)%n])

p = p + i + 1;

else

q = q + i + 1;

if (p == q)

q++;

}

return min(p, q) + 1;;

}

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

（二）KMP

1、KMP：主要用于字符串匹配，fail[i]表示T[i-fail[i]+1~i]和T[1~fail[i]]是相同的。

|  |  |
| --- | --- |
| getFail(str) | 预处理匹配串的失配数组 |
| match(str1,str2) | 对一个文本或者是字符串进行匹配 |

循环节也就是前缀后缀串，p从fail[n]开始，所有的p=fail[p]，1~p均为循环节。（模板传入字符串从下标1开始）

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*KMP.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

const int maxn = 1e5 + 5;

int fail[maxn];

void getFail (char\* str) {

int n = strlen(str+1);

int p = fail[0] = fail[1] = 0;

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

while (p && str[p+1] != str[i])

p = fail[p];

if (str[p+1] == str[i])

p++;

fail[i] = p;

}

/\*

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

printf("%d%c", fail[i], i == n ? '\n' : ' ');

\*/

}

int match (char\* S, char\* T) {

getFail(T);

int p = 0, ret = 0, n = strlen(S);

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

while (p && T[p+1] != S[i])

p = fail[p];

if (T[p+1] == S[i])

p++;

ret = max(ret, p);

}

return ret;

}

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

str: abaabaaab

fail: 001123412

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

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

2、e-KMP：给出两个字符串A（称为模板串）和B（称为子串），长度分别为lenA和lenB，要求在线性时间内，对于每个A[i]（0<i≤lenA），求出A[i]往前和B的前缀匹配的最大匹配长度，即T[i~i+cpfix[i]-1]和T[1~cpfix[i]]相等。

|  |  |
| --- | --- |
| getFail(str) | 预处理匹配串的失配数组 |
| getExtand(str1,str2) | 对模板串求extand的数组 |

（模板传入字符串从下标1开始）

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*e-KMP.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

const int maxn = 1e5 + 5;

int cpfix[maxn], extand[maxn];

void getCPfix(char\* str) {

int n = strlen(str + 1), tmp = 1;

cpfix[1] = n;

while (tmp < n && str[tmp] == str[tmp+1])

tmp++;

cpfix[2] = tmp-1;

int p = 2;

for (int k = 3; k <= n; k++) {

int e = p + cpfix[p] - 1, l = cpfix[k-p+1]; // e 为目前匹配过的最末位置, l 为对应的偏移;

if (k + l - 1 >= e) {

int j = e - k > 0 ? e - k : 0;

while (k + j <= n && str[k+j] == str[j+1])

j++;

cpfix[k] = j, p = k;

} else

cpfix[k] = l;

}

/\*

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

printf("%d%c", cpfix[i], i == n ? '\n' : ' ');

\*/

}

void getExtand(char\* S, char\* T) {

getCPfix(T);

int sn = strlen(S+1), tn = strlen(T+1);

int n = min(sn, tn), tmp = 1;

while (tmp <= n && S[tmp] == T[tmp])

tmp++;

extand[1] = tmp-1;

int p = 1;

for (int k = 2; k <= sn; k++) {

int e = p + extand[p] - 1, l = cpfix[k-p+1];

if (k + l - 1 >= e) {

int j = e - k > 0 ? e - k : 0;

while (k + j <= sn && j < tn && S[k+j] == T[j+1])

j++;

extand[k] = j, p = k;

} else

extand[k] = l;

}

/\*

for (int i = 1; i <= sn; i++)

printf("%d%c", extand[i], i == sn ? '\n' : ' ');

\*/

}

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

S: abaabaaab

T: aaabaabab

Cpfix: 921021010

Extand:102104210

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

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

（三）字典树(Tire)

对字符串集合建立的快速检索树。

|  |  |
| --- | --- |
| init() | 初始化，设置sz |
| idx(ch) | 映射字符（根据题意映射字符） |
| insert(str, id) | 向字典树中插入一个字符串 |
| match(str) | 完全匹配一个字符串 |

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Tire-Tree.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

const int maxn = 3000005;

const int sigma\_size = 26;

struct Tire {

int sz;

int g[maxn][sigma\_size];

int val[maxn];

void init();

int idx(char ch);

void insert(char\* s);

int match(char\* s);

};

/\*Code\*/

void Tire::init() {

sz = 1;

val[0] = 0;

memset(g[0], 0, sizeof(g[0]));

}

int Tire::idx (char ch) {

return ch - 'a';

}

int Tire::match(char\* s) {

int u = 0, n = strlen(s);

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

int v = idx(s[i]);

if (g[u][v] == 0)

return 0;

u = g[u][v];

}

return val[u];

}

void Tire::insert(char\* s) {

int u = 0, n = strlen(s);

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

int v = idx(s[i]);

if (g[u][v] == 0) {

val[sz] = 0;

memset(g[sz], 0, sizeof(g[sz]));

g[u][v] = sz++;

}

u = g[u][v];

}

val[u]++;

}

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

应用：

1. 匹配去重：利用对所有的字符串建立字典树，两个字符串不相同的话，对应的终止节点肯定不同。

例题：hdu 4760（字典树去重高级应用）

题目大意：有一个防火墙，具有添加一个子网络，删除一个子网络，以及转发包的操作。

* 添加操作包含子网络的id，以及子网络的子网掩码（计算出网络前缀，以及ip的下限），不会超过15个。
* 删除则是给定要删除的子网络id。
* 转发操作，给定两个ip，如果两个ip在同一个子网络中，则可以转发，否则丢弃。

解题思路：对子网掩码前缀建立字典树，每个前缀终止节点用一个set记录属于哪些子网络，ip下限。那么增加和删除操作既可以解决了。对于查询操作，分别查询两个ip，处理除两个ip可能属于的网络，判断有无共同即可。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*hdu4760.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <set>

#include <algorithm>

using namespace std;

typedef long long ll;

typedef pair<int, ll> pii;

typedef set<pii>::iterator iter;

const int maxn = 1024 \* 15 + 10;

const int maxm = 105;

const int sigma\_size = 2;

struct Tire {

int sz, sv;

int g[maxn \* maxm][sigma\_size];

int idx[maxn \* maxm], cnt[1030];

set<pii> ans, vis[maxn];

void init();

int newSet();

void addSet(int id, ll limt);

void insert(char\* str, pii x);

void remove(char\* str, pii x);

void find(char\* str);

bool judge(char\* a, char\* b);

}T;

struct Network {

int n;

ll suf[20];

char ip[20][maxm];

Network() {

n = 0;

memset(suf, 0, sizeof(suf));

}

}net[1030];

ll change(char\* ans, bool flag) {

char str[105];

int n = strlen(str), a[4], b;

if (flag)

scanf("%d.%d.%d.%d/%d", &a[0], &a[1], &a[2], &a[3], &b);

else {

scanf("%d.%d.%d.%d", &a[0], &a[1], &a[2], &a[3]);

b = 32;

}

int t = 0;

ll ret = 0;

for (int i = 0; i < 4; i++) {

for (int j = 7; j >= 0; j--) {

if (t < b)

ans[t] = ((a[i]>>j)&1) + '0';

else if (((a[i]>>j)&1))

ret |= (1LL<<(31-t));

t++;

}

}

ans[t] = '\0';

return ret;

}

int main () {

int id;

char op[5], a[maxm], b[maxm], c[maxm];

T.init();

while (scanf("%s", op) == 1) {

if (op[0] == 'E') {

scanf("%d", &id);

scanf("%d", &net[id].n);

for (int i = 0; i < net[id].n; i++) {

net[id].suf[i] = change(net[id].ip[i], a);

T.insert(net[id].ip[i], make\_pair(id, net[id].suf[i]));

}

} else if (op[0] == 'D') {

scanf("%d", &id);

for (int i = 0; i < net[id].n; i++)

T.remove(net[id].ip[i], make\_pair(id, net[id].suf[i]));

net[id].n = 0;

} else {

change(a, 0);

change(b, 0);

printf("%c\n", T.judge(a, b) ? 'F' : 'D');

}

}

return 0;

}

bool Tire::judge(char\* a, char\* b) {

memset(cnt, 0, sizeof(cnt));

T.find(a);

for (iter i = ans.begin(); i != ans.end(); i++)

cnt[i->first] = 1;

T.find(b);

for (iter i = ans.begin(); i != ans.end(); i++)

if (cnt[i->first])

return true;

return false;

}

void Tire::init() {

sz = sv = 1;

idx[0] = 0;

vis[0].clear();

memset(g[0], 0, sizeof(g[0]));

}

int Tire::newSet() {

vis[sv].clear();

return sv++;

}

void Tire::addSet(int id, ll limt) {

for (iter i = vis[id].begin(); i != vis[id].end(); i++)

if (i->second <= limt)

ans.insert(\*i);

}

void Tire::insert(char\* str, pii x) {

int u = 0, n = strlen(str);

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

int v = str[i] - '0';

if (g[u][v] == 0) {

idx[sz] = 0;

memset(g[sz], 0, sizeof(g[sz]));

g[u][v] = sz++;

}

u = g[u][v];

}

if (idx[u] == 0)

idx[u] = newSet();

vis[idx[u]].insert(x);

}

void Tire::remove(char\* str, pii x) {

int u = 0, n = strlen(str);

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

int v = str[i] - '0';

if (g[u][v] == 0) {

idx[sz] = 0;

memset(g[sz], 0, sizeof(g[sz]));

g[u][v] = sz++;

}

u = g[u][v];

}

vis[idx[u]].erase(x);

}

void Tire::find(char\* str) {

ans.clear();

ll ret = 0;

int u = 0, n = strlen(str);

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

int v = str[i] - '0';

if (g[u][v] == 0)

return;

u = g[u][v];

if (idx[u]) {

ll ret = 0;

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

if (str[j] == '1')

ret |= (1LL<<(31-j));

addSet(idx[u], ret);

}

}

}

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

1. 模糊匹配：即模拟搜索引擎的模拟搜索或者是正则表达式，一般要在字典树上做DFS进行匹配。
2. 亦或和最大问题：对所有的数建立字典树，在有对于既定树，只要在字典树上移动，尽量向亦或和大的方向走。

例题：hdu 4757（可持久化字典树解决亦或和）

题目大意：给定一棵树，每个节点有一个值，现在有Q次询问，每次询问u到v路径上节点值与w亦或值的最大值。

解题思路： 可持久化字典树，在每次插入的同时，不修改原先的节点，而是对所有修改的节点复制一个新的节点，并且在新的节点上做操作，这样做的目的是能够获取某次修改前的状态。同过可持久化的操作，保留了修改前后的公共数据。

对给定树上的所有节点权值建立01字典树，然后每个节点都保存着一棵可持久化字典树，表示的是从根节点到该节点路径节点所形成的字典树。对每个节点建树的过程通过修改其父亲节点而得到。

查询时，对根据u，v，lca(u,v)三棵字典树的情况确定亦或的最大值，注意lca(u,v)这个节点要单独计算。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*hdu4757.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

const int maxn = 1e5 + 5;

int N, Q, E, V[maxn], first[maxn], jump[maxn \* 2], link[maxn \* 2];

int id, idx[maxn], top[maxn], far[maxn], son[maxn], dep[maxn], cnt[maxn];

inline void add\_Edge (int u, int v) {

link[E] = v;

jump[E] = first[u];

first[u] = E++;

}

inline void dfs (int u, int pre, int d) {

far[u] = pre;

son[u] = 0;

dep[u] = d;

cnt[u] = 1;

for (int i = first[u]; i + 1; i = jump[i]) {

int v = link[i];

if (v == pre)

continue;

dfs(v, u, d + 1);

cnt[u] += cnt[v];

if (cnt[son[u]] < cnt[v])

son[u] = v;

}

}

inline void dfs (int u, int rot) {

idx[u] = ++id;

top[u] = rot;

if(son[u])

dfs(son[u], rot);

for (int i = first[u]; i + 1; i = jump[i]) {

int v = link[i];

if (v == far[u] || v == son[u])

continue;

dfs(v, v);

}

}

inline int LCA (int u, int v) {

int p = top[u], q = top[v];

while (p != q) {

if (dep[p] < dep[q]) {

swap(p, q);

swap(u, v);

}

u = far[p];

p = top[u];

}

return dep[u] > dep[v] ? v : u;

}

void init() {

E = id = 0;

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

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

scanf("%d", &V[i]);

int u, v;

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

scanf("%d%d", &u, &v);

add\_Edge(u, v);

add\_Edge(v, u);

}

dfs(1, 0, 0);

dfs(1, 1);

}

struct node {

int g[2], c;

}nd[maxn \* 20];

int sz, root[maxn];

int insert (int r, int w) {

int ret, x;

ret = x = sz++;

nd[x] = nd[r];

for (int i = 15; i >= 0; i--) {

int v = (w>>i)&1;

int t = sz++;

nd[t] = nd[nd[x].g[v]];

nd[t].c++;

nd[x].g[v] = t;

x = t;

}

return ret;

}

void dfs(int u) {

root[u] = insert(root[far[u]], V[u]);

for (int i = first[u]; i + 1; i = jump[i]) {

int v = link[i];

if (v == far[u])

continue;

dfs(v);

}

}

void Tire\_init() {

sz = 1;

root[0] = nd[0].c = 0;

memset(nd[0].g, 0, sizeof(nd[0].g));

dfs(1);

}

int query(int x, int y, int z, int w) {

int ans = V[z] ^ w, ret = 0;

z = root[z];

for (int i = 15; i >= 0; i--) {

int v = ((w>>i)&1) ^ 1;

int cnt = nd[nd[x].g[v]].c + nd[nd[y].g[v]].c - 2 \* nd[nd[z].g[v]].c;

if (cnt)

ret |= (1<<i);

else

v = v^1;

x = nd[x].g[v], y = nd[y].g[v], z = nd[z].g[v];

}

return max(ans, ret);

}

int main () {

while (scanf("%d%d", &N, &Q) == 2) {

init();

Tire\_init();

int u, v, w;

while (Q--) {

scanf("%d%d%d", &u, &v, &w);

printf("%d\n", query(root[u], root[v], LCA(u, v), w));

}

}

return 0;

}

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

（四）AC自动机(Aho-Croasick)

结合KMP和字典树的性质，在字典树上构建失配函数。

|  |  |
| --- | --- |
| init() | 初始化，设置sz |
| idx(ch) | 映射字符（根据题意映射字符） |
| insert(str, id) | 向AC自动机中插入一个字符串 |
| getFail() | 对AC自动机处理失配函数 |
| match(str) | 匹配一个字符串 |
| mark() | 标记匹配到的位置 |

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Aho-Croasick.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

const int maxn = 205;

const int sigma\_size = 26;

struct Aho\_Corasick {

int sz, g[maxn][sigma\_size];

int tag[maxn], fail[maxn], last[maxn];

void init();

int idx(char ch);

void insert(char\* str, int k);

void getFail();

void match(char\* str);

void mark(int x, int y);

};

/\* Code \*/

void Aho\_Corasick::init() {

sz = 1;

tag[0] = 0;

memset(g[0], 0, sizeof(g[0]));

}

int Aho\_Corasick::idx(char ch) {

return ch - 'a';

}

void Aho\_Corasick::mark(int x, int y) {}

void Aho\_Corasick::insert(char\* str, int k) {

int u = 0, n = strlen(str);

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

int v = idx(str[i]);

if (g[u][v] == 0) {

tag[sz] = 0;

memset(g[sz], 0, sizeof(g[sz]));

g[u][v] = sz++;

}

u = g[u][v];

}

tag[u] = k;

}

void Aho\_Corasick::match(char\* str) {

int n = strlen(str), u = 0;

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

int v = idx(str[i]);

while (u && g[u][v] == 0)

u = fail[u];

u = g[u][v];

if (tag[u]) mark(i, u);

else if (last[u]) mark(i, last[u]);

}

}

void Aho\_Corasick::getFail() {

queue<int> que;

for (int i = 0; i < sigma\_size; i++) {

int u = g[0][i];

if (u) {

fail[u] = last[u] = 0;

que.push(u);

}

}

while (!que.empty()) {

int r = que.front();

que.pop();

for (int i = 0; i < sigma\_size; i++) {

int u = g[r][i];

if (u == 0) {

g[r][i] = g[fail[r]][i];

continue;

}

que.push(u);

int v = fail[r];

while (v && g[v][i] == 0)

v = fail[v];

fail[u] = g[v][i];

last[u] = tag[fail[u]] ? fail[u] : last[fail[u]];

}

}

}

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

应用：

1. 匹配问题：不同于字典树的是AC自动机是处理子串匹配。
2. 结合DP：一般有矩阵快速幂（根据自动机构建矩阵），状压DP。

例题：hdu 3341（AC自动机+变进制状压DP）

题目大意：给定一些需要匹配的串，然后在给定一个目标串，现在可以通过交换目标串中任意两个位置的字符，要求最后生成的串匹配尽量多的匹配串，可以重复匹配。

解题思路：这题很明显是AC自动机+DP，但是dp的状态需要开40∗40∗40∗40（记录每种字符的个数），空间承受

不了，但是其实因为目标串的长度有限，为40；所以状态更本不需要那么多，最多只有10∗10∗10∗10，但是通过

40进制的hash转换肯定是不行，可以根据目标串中4种字符的个数，来调整每个位的进制。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*hdu3341.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <queue>

#include <vector>

#include <iostream>

#include <algorithm>

using namespace std;

typedef pair<int,int> pii;

const int maxn = 505;

const int maxs = 11 \* 11 \* 11 \* 11;

const int sigma\_size = 4;

struct Aho\_Corasick {

int sz, g[maxn][sigma\_size];

int tag[maxn], fail[maxn], last[maxn];

int c[4], bit[4], dp[maxs][maxn];

void init();

int idx(char ch);

void insert(char\* str, int k);

void getFail();

void match(char\* str);

void put(int x, int y);

int solve(char\* w);

int hash(int a, int b, int c, int d);

}AC;

int N;

char w[50];

int main () {

int cas = 1;

while (scanf("%d", &N) == 1 && N) {

AC.init();

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

scanf("%s", w);

AC.insert(w, i);

}

scanf("%s", w);

printf("Case %d: %d\n", cas++, AC.solve(w));

}

return 0;

}

int Aho\_Corasick::hash(int a, int b, int c, int d) {

return a \* bit[0] + b \* bit[1] + c \* bit[2] + d;

}

int Aho\_Corasick::solve(char\* w) {

getFail();

int n = strlen(w);

memset(c, 0, sizeof(c));

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

c[idx(w[i])]++;

for (int i = 0; i < 4; i++) {

bit[i] = 1;

for (int j = i + 1; j < 4; j++)

bit[i] \*= (c[j]+1);

}

int ans = 0, t[4];

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

dp[hash(c[0], c[1], c[2], c[3])][0] = 0;

for (t[0] = c[0]; t[0] >= 0; t[0]--)

for (t[1] = c[1]; t[1] >= 0; t[1]--)

for (t[2] = c[2]; t[2] >= 0; t[2]--)

for (t[3] = c[3]; t[3] >= 0; t[3]--) {

int s = hash(t[0], t[1], t[2], t[3]);

for (int i = 0; i < 4; i++) {

if (t[i] == 0)

continue;

int ss = s - bit[i];

for (int k = 0; k < sz; k++) {

if (dp[s][k] < 0)

continue;

int u = k;

while (u && g[u][i] == 0)

u = fail[u];

u = g[u][i];

if (dp[ss][u] < dp[s][k] + tag[u]) {

dp[ss][u] = dp[s][k] + tag[u];

ans = max(ans, dp[ss][u]);

}

}

}

}

return ans;

}

void Aho\_Corasick::init() {

sz = 1;

tag[0] = 0;

memset(g[0], 0, sizeof(g[0]));

}

int Aho\_Corasick::idx(char ch) {

if (ch == 'A')

return 0;

if (ch == 'C')

return 1;

if (ch == 'G')

return 2;

return 3;

}

void Aho\_Corasick::put(int x, int y) {}

void Aho\_Corasick::insert(char\* str, int k) {

int u = 0, n = strlen(str);

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

int v = idx(str[i]);

if (g[u][v] == 0) {

tag[sz] = 0;

memset(g[sz], 0, sizeof(g[sz]));

g[u][v] = sz++;

}

u = g[u][v];

}

tag[u]++;

}

void Aho\_Corasick::match(char\* str) {

int n = strlen(str), u = 0;

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

int v = idx(str[i]);

while (u && g[u][v] == 0)

u = fail[u];

u = g[u][v];

if (tag[u])

put(i, u);

else if (last[u])

put(i, last[u]);

}

}

void Aho\_Corasick::getFail() {

queue<int> que;

for (int i = 0; i < sigma\_size; i++) {

int u = g[0][i];

if (u) {

fail[u] = last[u] = 0;

que.push(u);

}

}

while (!que.empty()) {

int r = que.front();

que.pop();

for (int i = 0; i < sigma\_size; i++) {

int u = g[r][i];

if (u == 0) {

g[r][i] = g[fail[r]][i];

continue;

}

que.push(u);

int v = fail[r];

while (v && g[v][i] == 0)

v = fail[v];

fail[u] = g[v][i];

tag[u] += tag[fail[u]];

//last[u] = tag[fail[u]] ? fail[u] : last[fail[u]];

}

}

}

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

（五）后缀数组

|  |  |
| --- | --- |
| init() | 初始化，读入字符串 |
| build() | 构建字符串的后缀数组 |
| getHeight | 构建字符串的height和rank数组 |

SA：以SA[i]为起点的后缀串字典序大小为i

rank：以i为起点的后缀串字典序大小为rank[i]

height：字典序大小为i的后缀串与i-1的后缀串前height[i]个字符相同

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Suffix-Arr.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

const int maxn = 100005;

struct Suffix\_Arr {

int n, s[maxn];

int SA[maxn], rank[maxn], height[maxn];

int tmp\_one[maxn], tmp\_two[maxn], c[305];

int d[maxn][20];

void init(char\* str);

void build(int m);

void getHeight();

void rmq\_init();

int rmq\_query(int x, int y);

};

void Suffix\_Arr::init(char\* str) {

n = 0;

int len = strlen(str);

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

s[n++] = str[i] - 'a' + 1;

s[n++] = 0;

}

void Suffix\_Arr::rmq\_init() {

for (int i = 0; i < n; i++) d[i][0] = height[i];

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

for (int i = 0; i + (1<<k) - 1 < n; i++)

d[i][k] = min(d[i][k-1], d[i+(1<<(k-1))][k-1]);

}

}

int Suffix\_Arr::rmq\_query(int x, int y) {

if (x == y)

return d[x][0];

if (x > y)

swap(x, y);

x++;

int k = 0;

while ((1<<(k+1) <= y - x + 1)) k++;

return min(d[x][k], d[y - (1<<k) + 1][k]);

}

void Suffix\_Arr::getHeight() {

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

rank[SA[i]] = i;

int mv = height[0] = 0;

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

if (mv) mv--;

int j = SA[rank[i] - 1];

while (s[i+mv] == s[j+mv])

mv++;

height[rank[i]] = mv;

}

}

void Suffix\_Arr::build (int m) {

int \*x = tmp\_one, \*y = tmp\_two;

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

for (int i = 0; i < n; i++) c[x[i] = s[i]]++;

for (int i = 1; i < m; i++) c[i] += c[i-1];

for (int i = n - 1; i >= 0; i--) SA[--c[x[i]]] = i;

for (int k = 1; k <= n; k <<= 1) {

int mv = 0;

for (int i = n - k; i < n; i++) y[mv++] = i;

for (int i = 0; i < n; i++) if (SA[i] >= k)

y[mv++] = SA[i] - k;

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

for (int i = 0; i < n; i++) c[x[y[i]]]++;

for (int i = 1; i < m; i++) c[i] += c[i-1];

for (int i = n - 1; i >= 0; i--) SA[--c[x[y[i]]]] = y[i];

swap(x, y);

mv = 1;

x[SA[0]] = 0;

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

x[SA[i]] = (y[SA[i-1]] == y[SA[i]] && y[SA[i-1] + k] == y[SA[i] + k] ? mv - 1 : mv++);

if (mv >= n)

break;

m = mv;

}

}

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

例题：hdu 5030（切割K次后的最大子串）

题目大意：给定k和一个字符串，要求将字符串拆分成k个子串。将每个子串中字典序最大的子串选出来，组成一个包含k个字符串的集合，要求这个集合中字典序最大的字符串字典序最小。

解题思路： 首先对整个字符串做后缀数组，同时处理出f数组，f[i]表示说以0~sa[i]开头共有多少种不同的子串。然后在0到f[n]之间二分找答案。

判定的方法复杂度为o(n),计算在分割k-1次能否将比当前串字典序大的字符串变小。这里要借助height数组，计算至少需要分割的次数。记录下每个分割的位置，再遍历一遍判定即可（因为有些分割的位置一下切割了不只一个串）

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*hdu5030.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <vector>

#include <algorithm>

using namespace std;

const int maxn = 1e5+5;

typedef long long ll;

struct Suffix\_Arr {

char s[maxn];

int n, SA[maxn], rank[maxn], height[maxn];

int tmp\_one[maxn], tmp\_two[maxn], c[maxn];

ll f[maxn];

void init ();

void build(int m);

void get\_height();

void solve ();

bool judge(ll m);

}AC;

int K;

int main () {

while (scanf("%d", &K) == 1 && K) {

AC.init();

AC.build(256);

AC.get\_height();

AC.solve();

}

return 0;

}

bool Suffix\_Arr::judge(ll m) {

int t = lower\_bound(f + 1, f + n + 1, m) - f;

int R = n - (f[t] - m + 1);

int len = R - SA[t] + 1;

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

c[SA[t]] = len;

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

len = min(len, height[i]);

if (len == 0)

return false;

c[SA[i]] = len;

}

int ret = 0, p = n + 1;

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

if (i == p) {

ret++;

p = n + 1;

}

if (c[i] != -1)

p = min(p, i + c[i]);

if (ret >= K)

return false;

}

return true;

}

void Suffix\_Arr::solve() {

ll l = 1, r = f[n];

for (int i = 0; i < 70; i++) {

ll mid = (l + r) / 2;

if (judge(mid))

r = mid;

else

l = mid;

}

int t = lower\_bound(f + 1, f + n + 1, r) - f;

int len = n - (f[t] - r + 1);

for (int i = SA[t]; i <= len; i++)

printf("%c", s[i]);

printf("\n");

}

void Suffix\_Arr::init() {

scanf("%s", s);

n = strlen(s) + 1;

}

void Suffix\_Arr::get\_height() {

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

rank[SA[i]] = i;

int mv = height[0] = 0;

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

if (mv) mv--;

int j = SA[rank[i] - 1];

while (s[i+mv] == s[j+mv])

mv++;

height[rank[i]] = mv;

}

n--;

f[0] = 0;

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

f[i] = f[i-1] + n - SA[i] - height[i];

}

void Suffix\_Arr::build(int m) {

int \*x = tmp\_one, \*y = tmp\_two;

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

for (int i = 0; i < n; i++) c[x[i] = s[i]]++;

for (int i = 1; i < m; i++) c[i] += c[i-1];

for (int i = n - 1; i >= 0; i--) SA[--c[x[i]]] = i;

for (int k = 1; k <= n; k <<= 1) {

int mv = 0;

for (int i = n - k; i < n; i++) y[mv++] = i;

for (int i = 0; i < n; i++) if (SA[i] >= k)

y[mv++] = SA[i] - k;

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

for (int i = 0; i < n; i++) c[x[y[i]]]++;

for (int i = 1; i < m; i++) c[i] += c[i-1];

for (int i = n - 1; i >= 0; i--) SA[--c[x[y[i]]]] = y[i];

swap(x, y);

mv = 1;

x[SA[0]] = 0;

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

x[SA[i]] = (y[SA[i-1]] == y[SA[i]] && y[SA[i-1] + k] == y[SA[i] + k] ? mv - 1 : mv++);

if (mv >= n)

break;

m = mv;

}

}

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

（六）后缀自动机(Suffix-Automaton)

后缀自动机是一个可以接受子串的自动机，一个构造好的自动机包含了一个由g数组组成DAG和一棵由pre数组组成的树。

|  |  |
| --- | --- |
| init() | 初始化 |
| newNode() | 分配一个新节点 |
| idx() | 对应一个字符的映射 |
| insert | 插入一个字符（构造方式为逐一插入字符串的字符） |
| get\_tuopu() | 获得拓扑序 |
| get\_right | 处理right数组 |

pre：指向了一个能够表示当前状态表示的所有字符串的最长公共后缀的结点。所有的状态的pre指针构成了一个parent树,恰好是字符串的逆序的后缀树。

step：表示该状态能够接受的最长的字符串长度。（step[u]-step[pre[u]]: 以u状态结尾的前缀串个数）

pos：每个状态对应的拓扑序。

right：该状态能够表示的所有字符串均出现过right次。

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SAM.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

const int maxn = 1e5 + 5;

const int SIGMA\_SIZE = 26;

struct SAM {

int sz, last;

int g[maxn<<1][SIGMA\_SIZE], pre[maxn<<1], step[maxn<<1];

int pos[maxn<<1], right[maxn<<1], cnt[maxn<<1];

void newNode(int s) {

step[++sz] = s;

pre[sz] = 0;

memset(g[sz], 0, sizeof(g[sz]));

}

int idx(char ch) { return ch -'a'; }

void init() {

sz = 0, last = 1;

newNode(0);

}

void insert(char ch);

void get\_tuopu();

void get\_right(char\* str);

}SA;

int main () {

return 0;

}

void SAM::insert(char ch) {

newNode(step[last] + 1);

int v = idx(ch), p = last, np = sz;

while (p && !g[p][v]) {

g[p][v] = np;

p = pre[p];

}

if (p) {

int q = g[p][v];

if (step[q] == step[p] + 1)

pre[np] = q;

else {

newNode(step[p] + 1);

int nq = sz;

for (int j = 0; j < SIGMA\_SIZE; j++) g[nq][j] = g[q][j];

pre[nq] = pre[q];

pre[np] = pre[q] = nq;

while (p && g[p][v] == q) {

g[p][v] = nq;

p = pre[p];

}

}

} else

pre[np] = 1;

last = np;

}

void SAM::get\_tuopu() {

for (int i = 0; i <= sz; i++) cnt[i] = 0;

for (int i = 1; i <= sz; i++) cnt[step[i]]++;

for (int i = 1; i <= sz; i++) cnt[i] += cnt[i-1];

for (int i = 1; i <= sz; i++) pos[cnt[step[i]]--] = i;

}

void SAM::get\_right(char\* str) {

int p = 1, n = strlen(str);

for (int i = 0; i <= sz; i++) right[i] = 0;

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

int v = idx(str[i]);

p = g[p][v];

right[p]++;

}

for (int i = sz; i; i--) {

int u = pos[i];

right[pre[u]] += right[u];

}

}

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

例题：spoj 1812（多个串的最长公共子串）

题目大意：给定n个字符串，求这n个字符串的最长公共子串。

解题思路：首先用第一个串构建自动机，然后分别用其它字符串去匹配，每个节点记录两个状态lcs和tlcs，分别表示当多个串的最长公共子串的最后一个字符落在该状态上的长度和当前串的最长公共子串的最后一个字符落在该状态上的长度。每次匹配一个字符串后，在更新lcs数组是，对于u节点的状态，pre[u]同样能接收，

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*spoj1812.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

typedef long long ll;

const int maxn = 100005;

const int SIGMA\_SIZE = 26;

const int inf = 0x3f3f3f3f;

struct SAM {

int sz, last;

int g[maxn<<1][SIGMA\_SIZE], pre[maxn<<1], step[maxn<<1];

int lcs[maxn<<1], tlcs[maxn<<1];

void newNode(int s) {

step[++sz] = s;

pre[sz] = 0;

lcs[sz] = s;

memset(g[sz], 0, sizeof(g[sz]));

}

int idx(char ch) { return ch -'a'; }

void init() {

sz = 0, last = 1;

newNode(0);

memset(tlcs, 0, sizeof(tlcs));

}

void insert(char ch) {

newNode(step[last] + 1);

int v = idx(ch), p = last, np = sz;

while (p && !g[p][v]) {

g[p][v] = np;

p = pre[p];

}

if (p) {

int q = g[p][v];

if (step[q] == step[p] + 1)

pre[np] = q;

else {

newNode(step[p] + 1);

int nq = sz;

for (int j = 0; j < SIGMA\_SIZE; j++) g[nq][j] = g[q][j];

pre[nq] = pre[q];

pre[np] = pre[q] = nq;

while (p && g[p][v] == q) {

g[p][v] = nq;

p = pre[p];

}

}

} else

pre[np] = 1;

last = np;

}

void match(char\* str) {

memset(tlcs, 0 , sizeof(tlcs));

int n = strlen(str), p = 1, ans = 0;

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

int v = idx(str[i]);

if (g[p][v]) p = g[p][v], ans++;

else {

while (p && g[p][v] == 0) p = pre[p];

if (p) ans = step[p] + 1, p = g[p][v];

else ans = 0, p = 1;

}

tlcs[p] = max(tlcs[p], ans);

}

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

int u = id[i];

lcs[u] = min(lcs[u], tlcs[u]);

int v = pre[u];

//printf("%d %d %d!", u, v, tlcs[u]);

tlcs[v] = max(tlcs[v], tlcs[u]);

}

//printf("\n");

}

int solve() {

int ans = 0;

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

//printf("%d ", lcs[i]);

ans = max(ans, lcs[i]);

}

//printf("\n");

return ans;

}

int du[maxn<<1], id[maxn<<1];

void topu() {

memset(du, 0, sizeof(du));

for (int i = 1; i <= sz; i++)

du[pre[i]]++;

int head = 0, rear = 0;

for (int i = 1; i <= sz; i++)

if (du[i] == 0) id[rear++] = i;

while (head < rear) {

int u = id[head++];

du[pre[u]]--;

if (du[pre[u]] == 0 && pre[u])

id[rear++] = pre[u];

}

}

}SA;

char str[20][maxn];

int main () {

int N = 0;

while (gets(str[N]) != NULL) N++;

SA.init();

int len = strlen(str[0]);

for (int i = 0; i < len; i++) SA.insert(str[0][i]);

SA.topu();

for (int i = 1; i < N; i++) SA.match(str[i]);

printf("%d\n", SA.solve());

return 0;

}

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

四、其它数据结构

（一）莫队算法

处理区间询问问题，离线操作，算法复杂度为o(nsqrt(n))

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Modui.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <cmath>

#include <algorithm>

using namespace std;

typedef long long ll;

const int maxn = 50005;

struct State {

int l, r, id;

}Q[maxn];

int S, BSZ; // BSZ = sqrt(n);

bool cmp(const State& a, const State& b) {

if (a.l / BSZ == b.l / BSZ) return a.r < b.r;

return a.l / BSZ < b.l / BSZ;

}

ll query(int u, int v) {

if (u) {

for (int i = Q[u].l; i < Q[v].l; i++) { } // 删除

for (int i = Q[v].l; i < Q[u].l; i++) { } // 增加

for (int i = Q[u].r + 1; i <= Q[v].r; i++) { } //增加

for (int i = Q[v].r + 1; i <= Q[u].r; i++) { } // 删除

} else {

for (int i = Q[v].l; i <= Q[v].r; i++) { } //预处理第一块

}

return S;

}

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

第三章 计算几何

一、二维几何

（一）基础

1、点/向量：

|  |  |
| --- | --- |
| getDot() | 向量点积 |
| getCroess() | 向量叉积 |
| getLength() | 向量长度 |
| getPowLength() | 向量长度的平方 |
| getAngle() | 向量角度，返回弧度制 |
| rotate() | 旋转角度，传入弧度制 |
| getNormal() | 获得向量的单位法向量 |
| torad() | 角度转换成弧度 |

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Point2.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

const double pi = 4 \* atan(1);

const double eps = 1e-8;

inline int dcmp (double x) { if (fabs(x) < eps) return 0; else return x < 0 ? -1 : 1; }

inline double torad(double deg) { return deg / 180 \* pi; }

struct Point {

double x, y;

Point (double x = 0, double y = 0): x(x), y(y) {}

void read () { scanf("%lf%lf", &x, &y); }

void write () { printf("%lf %lf", x, y); }

bool operator == (const Point& u) const { return dcmp(x - u.x) == 0 && dcmp(y - u.y) == 0; }

bool operator != (const Point& u) const { return !(\*this == u); }

bool operator < (const Point& u) const { return dcmp(x - u.x) < 0 || (dcmp(x-u.x)==0 && dcmp(y-u.y) < 0); }

bool operator > (const Point& u) const { return u < \*this; }

bool operator <= (const Point& u) const { return \*this < u || \*this == u; }

bool operator >= (const Point& u) const { return \*this > u || \*this == u; }

Point operator + (const Point& u) { return Point(x + u.x, y + u.y); }

Point operator - (const Point& u) { return Point(x - u.x, y - u.y); }

Point operator \* (const double u) { return Point(x \* u, y \* u); }

Point operator / (const double u) { return Point(x / u, y / u); }

double operator ^ (const Point& u) { return x \* u.y - y \* u.x; } // 叉积

double operator & (const Point& u) { return x \* u.x + y \* u.y; } // 点积

};

typedef Point Vector;

/\* 点积: 两向量长度的乘积再乘上它们夹角的余弦, 夹角大于90度时点积为负 \*/

double getDot(Vector a, Vector b) { return a.x \* b.x + a.y \* b.y; }

/\* 叉积: 叉积等于两向量组成的三角形有向面积的两倍, cross(v, w) = -cross(w, v) \*/

double getCross (Vector a, Vector b) { return a.x \* b.y - a.y \* b.x; }

double getLength (Vector a) { return sqrt(getDot(a,a)); }

double getPowLength (Vector a) { return getDot(a, a); }

double getAngle (Vector u) { return atan2(u.y, u.x); }

double getAngle (Vector a, Vector b) { return acos(getDot(a, b) / getLength(a) / getLength(b)); }

Vector rotate (Vector a, double rad) { return Vector(a.x \* cos(rad) - a.y \* sin(rad), a.x \* sin(rad) + a.y \* cos(rad)); }

/\* 单位法线 \*/

Vector getNormal (Vector a) { double l = getLength(a); return Vector(-a.y/l, a.x/l); }

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

2、线/线段：

|  |  |
| --- | --- |
| getLine() | 获得直线方程式 |
| getIntersection() | 获得两条直线交点 |
| getDistanceToLine() | 点到直线距离 |
| getDistanceToSegment() | 点到线段距离 |
| getPointToLine() | 点在直线上的投影 |
| haveIntersection() | 判断两线段是否有交点 |
| onSegment() | 判断点是否在线段上 |

需要Point2.cpp

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Line2.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct Line {

double a, b, c;

Line (double a = 0, double b = 0, double c = 0): a(a), b(b), c(c) {}

};

Line getLine (double x1, double y1, double x2, double y2) { return Line(y2-y1, x1-x2, y1\*x2-x1\*y2); }

Line getLine (double a, double b, Point u) { return Line(a, -b, u.y \* b - u.x \* a); }

bool getIntersection (Line p, Line q, Point& o) {

if (fabs(p.a \* q.b - q.a \* p.b) < eps)

return false;

o.x = (q.c \* p.b - p.c \* q.b) / (p.a \* q.b - q.a \* p.b);

o.y = (q.c \* p.a - p.c \* q.a) / (p.b \* q.a - q.b \* p.a);

return true;

}

/\* 直线pv和直线qw的交点 \*/

bool getIntersection (Point p, Vector v, Point q, Vector w, Point& o) {

if (dcmp(getCross(v, w)) == 0) return false;

Vector u = p - q;

double k = getCross(w, u) / getCross(v, w);

o = p + v \* k;

return true;

}

/\* 点p到直线ab的距离 \*/

double getDistanceToLine (Point p, Point a, Point b) { return fabs(getCross(b-a, p-a) / getLength(b-a)); }

/\* 点p到线段ab的距离 \*/

double getDistanceToSegment (Point p, Point a, Point b) {

if (a == b) return getLength(p-a);

Vector v1 = b - a, v2 = p - a, v3 = p - b;

if (dcmp(getDot(v1, v2)) < 0) return getLength(v2);

else if (dcmp(getDot(v1, v3)) > 0) return getLength(v3);

else return fabs(getCross(v1, v2) / getLength(v1));

}

/\* 点p在直线ab上的投影 \*/

Point getPointToLine (Point p, Point a, Point b) {

Vector v = b-a;

return a+v\*(getDot(v, p-a) / getDot(v,v));

}

/\* 判断线段是否存在交点 \*/

bool haveIntersection (Point a1, Point a2, Point b1, Point b2) {

double c1=getCross(a2-a1, b1-a1), c2=getCross(a2-a1, b2-a1), c3=getCross(b2-b1, a1-b1), c4=getCross(b2-b1,a2-b1);

return dcmp(c1)\*dcmp(c2) < 0 && dcmp(c3)\*dcmp(c4) < 0;

}

/\* 判断点是否在线段上 \*/

bool onSegment (Point p, Point a, Point b) { return dcmp(getCross(a-p, b-p)) == 0 && dcmp(getDot(a-p, b-p)) < 0; }

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

3、圆：

|  |  |
| --- | --- |
| getLineCircleIntersection() | 求直线与圆的交点 |
| getCircleCircleIntersection() | 求圆与圆的交点 |
| getInscribedCircle() | 三点确定内切圆（需要getDistanceToLine函数） |
| getCircumscribedCircle() | 三点确定外切圆 |
| getTangentsPointToCircle() | 过定点做圆的切线 |
| getTangentsCircleToCircle() | 求圆与圆的切线 |
| getTangPointsPointToCircle() | 过定点做圆切线求切点 |
| getTangPointsCircleToCircle() | 圆与圆切线求切点 |

需要Point2.cpp

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Circle.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct Circle {

Point o;

double r;

Circle () {}

Circle (Point o, double r = 0): o(o), r(r) {}

void read () { o.read(), scanf("%lf", &r); }

Point point(double rad) { return Point(o.x + cos(rad)\*r, o.y + sin(rad)\*r); }

double area (double rad) { return rad \* r \* r / 2; }

};

/\* 直线和圆的交点 \*/

int getLineCircleIntersection (Point p, Point q, Circle O, double& t1, double& t2, vector<Point>& sol) {

Vector v = q - p;

/\* 使用前需清空sol \*/

//sol.clear();

double a = v.x, b = p.x - O.o.x, c = v.y, d = p.y - O.o.y;

double e = a\*a+c\*c, f = 2\*(a\*b+c\*d), g = b\*b+d\*d-O.r\*O.r;

double delta = f\*f - 4\*e\*g;

if (dcmp(delta) < 0) return 0;

if (dcmp(delta) == 0) {

t1 = t2 = -f / (2 \* e);

sol.push\_back(p + v \* t1);

return 1;

}

t1 = (-f - sqrt(delta)) / (2 \* e); sol.push\_back(p + v \* t1);

t2 = (-f + sqrt(delta)) / (2 \* e); sol.push\_back(p + v \* t2);

return 2;

}

/\* 圆和圆的交点 \*/

int getCircleCircleIntersection (Circle o1, Circle o2, vector<Point>& sol) {

double d = getLength(o1.o - o2.o);

if (dcmp(d) == 0) {

if (dcmp(o1.r - o2.r) == 0) return -1;

return 0;

}

if (dcmp(o1.r + o2.r - d) < 0) return 0;

if (dcmp(fabs(o1.r-o2.r) - d) > 0) return 0;

double a = getAngle(o2.o - o1.o);

double da = acos((o1.r\*o1.r + d\*d - o2.r\*o2.r) / (2\*o1.r\*d));

Point p1 = o1.point(a-da), p2 = o1.point(a+da);

sol.push\_back(p1);

if (p1 == p2) return 1;

sol.push\_back(p2);

return 2;

}

/\* 三点确定内切圆 \*/

Circle InscribedCircle(Point p1, Point p2, Point p3) {

double a = getLength(p2 - p3);

double b = getLength(p3 - p1);

double c = getLength(p1 - p2);

Point p = (p1 \* a + p2 \* b + p3 \* c) / (a + b + c);

return Circle(p, getDistanceToLine(p, p1, p2));

}

/\* 三点确定外切圆 \*/

Circle CircumscribedCircle(Point p1, Point p2, Point p3) {

double Bx = p2.x - p1.x, By = p2.y - p1.y;

double Cx = p3.x - p1.x, Cy = p3.y - p1.y;

double D = 2 \* (Bx \* Cy - By \* Cx);

double cx = (Cy \* (Bx \* Bx + By \* By) - By \* (Cx \* Cx + Cy \* Cy)) / D + p1.x;

double cy = (Bx \* (Cx \* Cx + Cy \* Cy) - Cx \* (Bx \* Bx + By \* By)) / D + p1.y;

Point p = Point(cx, cy);

return Circle(p, getLength(p1 - p));

}

/\* 过定点作圆的切线 \*/

int getTangentsPointToCircle (Point p, Circle o, Vector\* v) {

Vector u = o.o - p;

double d = getLength(u);

if (d < o.r) return 0;

else if (dcmp(d - o.r) == 0) {

v[0] = rotate(u, pi / 2);

return 1;

} else {

double ang = asin(o.r / d);

v[0] = rotate(u, -ang);

v[1] = rotate(u, ang);

return 2;

}

}

/\* a[i] 和 b[i] 分别是第i条切线在O1和O2上的切点 \*/

/\* have some problems \*/

int getTangentsCircleToCircle (Circle o1, Circle o2, Point\* a, Point\* b) {

int cnt = 0;

if (o1.r < o2.r) { swap(o1, o2); swap(a, b); }

double d2 = getLength(o1.o - o2.o); d2 = d2 \* d2;

double rdif = o1.r - o2.r, rsum = o1.r + o2.r;

if (d2 < rdif \* rdif) return 0;

if (dcmp(d2) == 0 && dcmp(o1.r - o2.r) == 0) return -1;

double base = getAngle(o2.o - o1.o);

if (dcmp(d2 - rdif \* rdif) == 0) {

a[cnt] = o1.point(base); b[cnt] = o2.point(base); cnt++;

return cnt;

}

double ang = acos( rdif / sqrt(d2) );

a[cnt] = o1.point(base+ang); b[cnt] = o2.point(base+ang); cnt++;

a[cnt] = o1.point(base-ang); b[cnt] = o2.point(base-ang); cnt++;

if (dcmp(d2 - rsum \* rsum) == 0) {

a[cnt] = o1.point(base); b[cnt] = o2.point(base); cnt++;

} else if (d2 > rsum \* rsum) {

double ang = acos( rsum / sqrt(d2) );

a[cnt] = o1.point(base+ang); b[cnt] = o2.point(pi+base+ang); cnt++;

a[cnt] = o1.point(base-ang); b[cnt] = o2.point(pi+base-ang); cnt++;

}

return cnt;

}

/\* 获得点与圆的切点 \*/

void getTangPointsPointToCircle(Point p, Circle c, Point& a, Point& b) {

double k = 2 \* (c.r\*c.r + c.o.x\*p.x + c.o.y\*p.y - c.o.x\*c.o.x - c.o.y\*c.o.y);

double s = 2 \* (p.x-c.o.x), t = 2 \* (p.y - c.o.y);

if (dcmp(t) == 0) {

a.x = b.x = k / s;

double tmp = sqrt(c.r \* c.r - (c.o.x-a.x)\*(c.o.x-a.x));

a.y = c.o.y + tmp;

b.y = c.o.y - tmp;

} else {

double x = c.o.y - k/t;

double A = 1 + (s/t) \* (s/t);

double B = 2 \* (x\*s/t - c.o.x);

double C = c.o.x \* c.o.x + x \* x - c.r \* c.r;

double tmp = sqrt(B \* B - 4 \* A \* C);

a.x = (tmp - B) / 2 / A;

b.x = (-tmp - B) / 2 / A;

a.y = (k - s \* a.x) / t;

b.y = (k - s \* b.x) / t;

}

}

/\* 圆与圆的切点 \*/

void getTangPointsCircleToCircle (Circle c1, Circle c2, Point& a, Point& b) {

double x0 = c2.o.x - c1.o.x, y0 = c2.o.y - c1.o.y, cs, sn, rdis = c1.r - c2.r;

if (dcmp(y0) == 0) {

cs = rdis / x0;

sn = sqrt(1 - cs \* cs);

a = Point(c1.r \* cs + c1.o.x, c1.r \* sn + c1.o.y);

b = Point(c1.r \* cs + c1.o.x, c1.r \* (-sn) + c1.o.y);

} else {

double A = (x0/y0)\*(x0/y0) + 1;

double B = -2 \* x0 \* rdis / y0 / y0;

double C = (rdis / y0) \* (rdis / y0) - 1;

double delta = sqrt(B \* B - 4 \* A \* C);

cs = (-B + delta) / 2 / A;

sn = (rdis - x0 \* cs) / y0;

a = Point(c1.r \* cs + c1.o.x, c1.r \* sn + c1.o.y);

cs = (-B - delta) / 2 / A;

sn = (rdis - x0 \* cs) / y0;

b = Point(c1.r \* cs + c1.o.x, c1.r \* sn + c1.o.y);

}

}

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

4、多边形：

|  |  |
| --- | --- |
| getAngle() | 三边确定角度：余弦定理 |
| getArea() | 获得三角形的面积 |
| getDirArea() | 获得三角形的有向面积 |
| getPolygonArea() | 获得多边形的面积 |
| isPointInPolygon() | 判断点是否在多边形内 |
| simplify() | 删除多边形的共线点 |
| cutPolygon() | 切割多边形 |

需要Point2.cpp

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Polygon.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 余弦定理: 三边确定一角 \*/

double getAngle (double a, double b, double c) { return acos((a\*a+b\*b-c\*c) / (2\*a\*b)); }

double getArea (double a, double b, double c) { double s =(a+b+c)/2; return sqrt(s\*(s-a)\*(s-b)\*(s-c)); }

double getArea (double a, double h) { return a \* h / 2; }

double getArea (Point a, Point b, Point c) { return fabs(getCross(b - a, c - a)) / 2; }

double getDirArea (Point a, Point b, Point c) { return getCross(b - a, c - a) / 2; }

typedef vector<Point> Polygon;

double getPolygonalgetArea (Point\* p, int n) {

double ret = 0;

for (int i = 0; i < n-1; i++)

ret += (p[i]-p[0]) ^ (p[i+1]-p[0]);

return ret/2;

}

int isPointInPolygon (Point o, Point\* p, int n) {

int wn = 0;

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

int j = (i + 1) % n;

if (onSegment(o, p[i], p[j]) || o == p[i]) return 0; // 边界上

int k = dcmp(getCross(p[j] - p[i], o-p[i]));

int d1 = dcmp(p[i].y - o.y);

int d2 = dcmp(p[j].y - o.y);

if (k > 0 && d1 <= 0 && d2 > 0) wn++;

if (k < 0 && d2 <= 0 && d1 > 0) wn--;

}

return wn ? -1 : 1;

}

/\* 去除多边形共线点 \*/

Polygon simplify (const Polygon& poly) {

Polygon ret;

int n = poly.size();

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

Point a = poly[i];

Point b = poly[(i+1)%n];

Point c = poly[(i+2)%n];

if (dcmp((b-a)^(c-b)) != 0 && (ret.size() == 0 || b != ret[ret.size()-1]))

ret.push\_back(b);

}

return ret;

}

/\* 计算半平面相交可以用增量法，o(n^2)，初始设置4条无穷大的半平面 \*/

/\* 用有向直线A->B切割多边形u，返回左侧。可能退化成单点或线段 \*/

Polygon cutPolygon (Polygon u, Point a, Point b) {

Polygon ret;

int n = u.size();

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

Point c = u[i], d = u[(i+1)%n];

if (dcmp((b-a)^(c-a)) >= 0) ret.push\_back(c);

if (dcmp((b-a)^(c-d)) != 0) {

Point t;

getIntersection(a, b-a, c, d-c, t);

if (onSegment(t, c, d))

ret.push\_back(t);

}

}

return ret;

}

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

（二）面积交

1、圆与圆/多边形：

|  |  |
| --- | --- |
| getPublicAreaCircleToCircle() | 圆与圆的面积交 |
| getPublicAreaCircleToTriangle() | 圆与三角形的面积交，其中三角形一顶点为圆心 |
| getPublicAreaCircleToPolugon() | 圆与多边形的面积交 |

需要Point2.cpp、Line2.cpp、Circle.cpp、Polygon.cpp

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*PublicArea.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 圆与圆面积交 \*/

double getPublicAreaCircleToCircle(Circle a, Circle b) {

double dis = getLength(a.o - b.o);

if (dcmp(dis-a.r-b.r) >= 0) return 0;

if (dis <= fabs(a.r - b.r)) { return min(a.getArea(2\*pi), b.getArea(2\*pi)); }

double ang1 = getAngle(a.r, dis, b.r);

double ang2 = getAngle(b.r, dis, a.r);

double ret = ang1 \* a.r \* a.r + ang2 \* b.r \* b.r - dis \* a.r \* sin(ang1);

return ret;

}

/\* 三角形一顶点为圆心 \*/

double getPublicAreaCircleToTriangle (Circle O, Point a, Point b) {

if (dcmp((a-O.o)^(b-O.o)) == 0) return 0;

int sig = 1;

double da = getPowLength(O.o-a), db = getPowLength(O.o-b);

if (dcmp(da-db) > 0) {

swap(da, db);

swap(a, b);

sig = -1;

}

double t1, t2;

vector<Point> sol;

int n = getLineCircleIntersection(a, b, O, t1, t2, sol);

if (dcmp(da-O.r\*O.r) <= 0) {

if (dcmp(db-O.r\*O.r) <= 0) return getDirArea(O.o, a, b) \* sig;

int k = 0;

if (getPowLength(sol[0]-b) > getPowLength(sol[1]-b)) k = 1;

double ret = getArea(O.o, a, sol[k]) + O.getArea(getAngle(sol[k]-O.o, b-O.o));

double tmp = (a-O.o)^(b-O.o);

return ret \* sig \* dcmp(tmp);

}

double d = getDistanceToSegment(O.o, a, b);

if (dcmp(d-O.r) >= 0) {

double ret = O.getArea(getAngle(a-O.o, b-O.o));

double tmp = (a-O.o)^(b-O.o);

return ret \* sig \* dcmp(tmp);

}

double k1 = O.r / getLength(a - O.o), k2 = O.r / getLength(b - O.o);

Point p = O.o + (a - O.o) \* k1, q = O.o + (b - O.o) \* k2;

double ret1 = O.getArea(getAngle(p-O.o, q-O.o));

double ret2 = O.getArea(getAngle(sol[0]-O.o, sol[1]-O.o)) - getArea(O.o, sol[0], sol[1]);

double ret = (ret1 - ret2), tmp = (a-O.o)^(b-O.o);

return ret \* sig \* dcmp(tmp);

}

/\* 多边形和圆的面积交 \*/

double getPublicAreaCircleToPolygon (Circle O, Point\* p, int n) {

if (dcmp(O.r) == 0) return 0;

double area = 0;

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

int u = (i + 1) % n;

area += getPublicAreaCircleToTriangle(O, p[i], p[u]);

}

return fabs(area);

}

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

2、扫描法：

例题：hdu 4629（多个三角形面积并）

题目大意：给定n个三角形，求覆盖k次的面积

解题思路：离散化

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*hdu4629.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <cmath>

#include <algorithm>

using namespace std;

const double eps = 1e-10;

inline int dcmp(double e) { if (fabs(e) < eps) return 0; return e < 0 ? -1 : 1; }

struct Point {

double x, y;

void read() { scanf("%lf%lf", &x, &y); }

Point(double x = 0, double y = 0): x(x), y(y) {}

Point operator < (const Point& u) const { return x < u.x || (dcmp(x - u.x) == 0 && y < u.y); }

Point operator + (const Point& u) const { return Point(x + u.x, y + u.y); }

Point operator - (const Point& u) const { return Point(x - u.x, y - u.y); }

Point operator \* (const double& k) const { return Point(x \* k, y \* k); }

Point operator / (const double& k) const { return Point(x / k, y / k); }

};

typedef Point Vector;

double getDot(Vector a, Vector b) { return a.x \* b.x + a.y \* b.y; }

double getCross(Vector a, Vector b) { return a.x \* b.y - a.y \* b.x; }

double getLength(Vector a) { return sqrt(getDot(a, a)); }

Vector rotate(Vector v, double rad) { return Vector(v.x \* cos(rad) - v.y \* sin(rad), v.x \* sin(rad) + v.y \* cos(rad)); }

double getArea(Point a, Point b, Point c) { return getCross(b - a, c - a); }

/\* 判断线段是否存在交点 \*/

bool haveIntersection (Point a1, Point a2, Point b1, Point b2) {

double c1=getCross(a2-a1, b1-a1), c2=getCross(a2-a1, b2-a1), c3=getCross(b2-b1, a1-b1), c4=getCross(b2-b1,a2-b1);

return dcmp(c1)\*dcmp(c2) < 0 && dcmp(c3)\*dcmp(c4) < 0;

}

/\* 直线pv和直线qw的交点 \*/

bool getIntersection (Point p, Vector v, Point q, Vector w, Point& o) {

if (dcmp(getCross(v, w)) == 0) return false;

Vector u = p - q;

double k = getCross(w, u) / getCross(v, w);

o = p + v \* k;

return true;

}

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

const int maxn = 55;

const int maxm = 1e5 + 5;

typedef pair<double,int> pdi;

int N, M;

double ans[maxn], len[maxn], event[maxm];

Point P[maxn][4];

pdi T[maxn<<1];

bool checkVertical() {

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

for (int j = 0; j < 3; j++)

if (dcmp(P[i][j].x - P[i][j+1].x) == 0) return true;

}

return false;

}

void adjust() {

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

for (int j = 0; j < 4; j++)

P[i][j] = rotate(P[i][j], 0.001);

}

}

void init () {

int n = N = M = 0;

scanf("%d", &n);

while (n--) {

for (int i = 0; i < 3; i++) P[N][i].read();

P[N][3] = P[N][0];

if (dcmp(getArea(P[N][0], P[N][1], P[N][2]))) N++;

}

while (checkVertical()) adjust();

}

inline bool onSegment(Point p, Point a, Point b) { return dcmp(getDot(a - p, b - p)) <= 0;}

bool verticalPos(Point l, Point r, int id, double& in, double& ot) {

int c = 0;

double y[3];

Point tmp;

//printf("%lf %lf %lf %lf\n", l.x, l.y, r.x, r.y);

for (int i = 0; i < 3; i++) {

getIntersection(l, r-l, P[id][i], P[id][i+1]-P[id][i], tmp);

if (onSegment(tmp, P[id][i], P[id][i+1]))

y[c++] = tmp.y;

}

sort(y, y + c);

if (c <= 1) return false;

in = y[0], ot = y[c-1];

return true;

}

void calculate (double x) {

int c = 0;

double in, ot;

Point a = Point(x, 0), b = Point(x, 1);

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

if (verticalPos(a, b, i, in, ot)) {

T[c++] = make\_pair(in, 1);

T[c++] = make\_pair(ot, -1);

}

}

sort(T, T + c);

int mv = 0;

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

if (mv > 0) len[mv] += T[i].first - T[i-1].first;

mv += T[i].second;

}

}

void getPosition() {

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

for (int ki = 0; ki < 3; ki++) {

Point a = P[i][ki], b = P[i][ki+1];

event[M++] = a.x;

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

for (int kj = 0; kj < 3; kj++) {

Point c = P[j][kj], d = P[j][kj+1], e;

if (haveIntersection(a, b, c, d)) {

getIntersection(a, b-a, c, d-c, e);

event[M++] = e.x;

}

}

}

}

}

sort(event, event + M);

M = unique(event, event + M) - event;

}

void solve() {

getPosition();

memset(ans, 0, sizeof(ans));

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

memset(len, 0, sizeof(len));

calculate(event[i-1] - eps);

calculate(event[i] + eps);

double x = event[i] - event[i-1];

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

ans[j] += len[j] \* x / 2;

}

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

printf("%.4lf\n", ans[i]);

}

int main () {

int cas;

scanf("%d", &cas);

while (cas--) {

init();

solve();

}

return 0;

}

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

二、三维几何

（一）基础

1、点/向量：

|  |  |
| --- | --- |
| getDot() | 向量点积 |
| getCroess() | 向量叉积 |
| getLength() | 向量长度 |
| getAngle() | 向量角度，返回弧度制 |
| getNormal() | 获得向量的单位法向量 |
| getDistancePointToPlane() | 点到平面的距离 |
| getPlaneProjection() | 点在平面上的投影 |

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Point3.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

inline int dcmp (double x) { if (fabs(x) < eps) return 0; else return x < 0 ? -1 : 1; }

struct Point3 {

double x, y, z;

Point3 (double x = 0, double y = 0, double z = 0): x(x), y(y), z(z) {}

void read () { scanf("%lf%lf%lf", &x, &y, &z); }

bool operator < (const Point3& u) const { return dcmp(x-u.x)<0 || (dcmp(x-u.x)==0 && dcmp(y-u.y)<0) || (dcmp(x-u.x)==0 && dcmp(y-u.y)==0 && dcmp(z-u.z) < 0); }

bool operator > (const Point3& u) const { return u < (\*this); }

bool operator == (const Point3& u) const { return !(u < (\*this) || (\*this) < u); }

bool operator != (const Point3& u) const { return !((\*this) == u); }

bool operator <= (const Point3& u) const { return \*this < u || \*this == u; }

bool operator >= (const Point3& u) const { return \*this > u || \*this == u; }

Point3 operator + (const Point3& u) const { return Point3(x+u.x, y+u.y, z+u.z); }

Point3 operator - (const Point3& u) const { return Point3(x-u.x, y-u.y, z-u.z); }

Point3 operator \* (const double u) const { return Point3(x\*u, y\*u, z\*u); }

Point3 operator / (const double u) const { return Point3(x/u, y/u, z/u); }

};

typedef Point3 Vector3;

double getDot(Vector3 a, Vector3 b) { return a.x\*b.x + a.y\*b.y + a.z\*b.z; }

Vector3 getCross (Vector3 a, Vector3 b) { return Vector3(a.y\*b.z-a.z\*b.y, a.z\*b.x-a.x\*b.z, a.x\*b.y-a.y\*b.x); }

double getLength(Vector3 a) { return sqrt(getDot(a, a)); }

double getAngle(Vector3 a, Vector3 b) { return acos(getDot(a, b) / getLength(a) / getLength(b)); }

Vector3 getNormal(Point3 a, Point3 b, Point3 c) {

Vector3 u = a-b, v = b-c;

Vector3 k = getCross(u, v);

return k / getLength(k);

}

double getDistancePointToPlane (Point3 p, Point3 p0, Vector3 v) { return fabs(getDot(p-p0, v)); }

Point3 getPlaneProjection (Point3 p, Point3 p0, Vector3 v) { return p - v \* getDot(p-p0, v); }

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

2、线/线段：

|  |  |
| --- | --- |
| getDistancePointToLine() | 空间点到直线距离 |
| getDistancePointToSegment() | 空间点到线段距离 |
| getPointLineToLine() | 异面直线公垂线 |
| getDistanceLineToLine() | 异面直线距离 |
| getDistanceSegmentToSegment() | 异面线段距离 |

需要Point3.cpp

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Line3.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 空间点到直线的距离 \*/

double getDistancePointToLine(Point3 p, Point3 a, Point3 b) {

Vector3 v1 = b-a, v2 = p-a;

return getLength(getCross(v1,v2)) / getLength(v1);

}

/\* 空间点到线段的距离 \*/

double getDistancePointToSegment(Point3 p, Point3 a, Point3 b) {

if (a == b) return getLength(p-a);

Vector3 v1 = b-a, v2 = p-a, v3 = p-b;

if (dcmp(getDot(v1, v2)) < 0) return getLength(v2);

else if (dcmp(getDot(v1, v3)) > 0) return getLength(v3);

else return getLength(getCross(v1, v2)) / getLength(v1);

}

/\* 异面直线公垂线 \*/

bool getPointLineToLine (Point3 a, Vector3 u, Point3 b, Vector3 v, double& s) {

double p = getDot(u, u) \* getDot(v, v) - getDot(u, v) \* getDot(u, v);

if (dcmp(p) == 0) return false;

double q = getDot(u, v) \* getDot(v, a-b) - getDot(v, v) \* getDot(u, a-b);

s = p/q;

return true;

}

/\* 异面直线的距离 \*/

double getDistanceLineToLine (Point3 a, Vector3 u, Point3 b, Vector3 v) {

Vector3 p = getCross(u, v);

return fabs(getDot(p, (a-b)) / getLength(p));

}

/\* 异面线段距离 \*/

double getDistanceSegmentToSegment(Point3 a, Point3 b, Point3 c, Point3 d) {

double s, t;

bool flag1 = getPointLineToLine(a, b-a, c, d-c, s);

bool flag2 = getPointLineToLine(c, d-c, a, b-a, t);

if (flag1 && flag2 && dcmp(s) > 0 && dcmp(s - 1) < 0 && dcmp(t) > 0 && dcmp(t-1) < 0) {

Vector3 u = b-a, v = d-c;

Point3 p = a + u \* s, q = b + v \* t;

return getLength(p-q);

} else {

double ans = 1e20;

ans = min(ans, getDistancePointToSegment(a, c, d));

ans = min(ans, getDistancePointToSegment(b, c, d));

ans = min(ans, getDistancePointToSegment(c, a, b));

ans = min(ans, getDistancePointToSegment(d, a, b));

return ans;

}

}

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

3、三角形/四面体：

|  |  |
| --- | --- |
| getArea() | 获得三角形面积 |
| onTriangle() | 判断点是否在三角形内 |
| haveIntersectionTirSeg() | 线段与三角形交点 |
| getVolume() | 四面体体积（6倍） |

需要Point3.cpp

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Triangle3.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 三角形面积 \*/

double getArea (Point3 a, Point3 b, Point3 c) { return getLength(getCross(b-a, c-a)); }

/\* 判断点是否在三角形内 \*/

bool onTriangle (Point3 p, Point3 a, Point3 b, Point3 c) {

double area1 = getArea(p, a, b);

double area2 = getArea(p, b, c);

double area3 = getArea(p, c, a);

return dcmp(area1 + area2 + area3 - getArea(a, b, c)) == 0;

}

/\* 三角形与线段交点 \*/

bool haveIntersectionTriSeg (Point3 p0, Point3 p1, Point3 p2, Point3 a, Point3 b, Point3& p) {

Vector3 v = getCross(p1-p0, p2-p0);

if (dcmp(getDot(v, b-a)) == 0) return false;

else {

double t = getDot(v, p0 - a) / getDot(v, b-a);

if (dcmp(t) < 0 || dcmp(t-2) > 0) return false;

p = a + (b-a) \* t;

return onTriangle(p, p0, p1, p2);

}

}

/\* 有向体积，是4边形的6倍 \*/

double getVolume (Point3 a, Point3 b, Point3 c, Point3 d) { return fabs(getDot(d-a, getCross(b-a, c-a))/6); } /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

4、球：

|  |  |
| --- | --- |
| getBallVolume() | 球体积 |
| getBallArea() | 球表面积 |
| getBallLength() | 球上两点的弧长 |

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Ball.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double getBallVolume (double r) { return pi \* 4 \* pow(r, 3) / 3; }

double getBallArea (double r) { return pi \* 4 \* pow(r, 2); }

double getBallLength (double lat1, double lot1, double lat2, double lot2, double r) {

double dlot = lot2 - lot1;

double dlat = lat2 - lat1;

double a = pow(sin(dlat/2), 2) + cos(lat1)\*cos(lat2)\*pow(sin(dlot/2), 2);

double c = 2 \* atan2(sqrt(a), sqrt(1-a));

return c \* r;

}

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

三、几何算法

（一）凸包

1、二维凸包：

需要Point2.cpp

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ConvexHull2.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 凸包 \*/

int getConvexHull (Point\* p, int n, Point\* ch) {

sort(p, p + n);

int m = 0;

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

/\* 可共线 \*/

//while (m > 1 && dcmp(getCross(ch[m-1]-ch[m-2], p[i]-ch[m-1])) < 0) m--;

while (m > 1 && dcmp(getCross(ch[m-1]-ch[m-2], p[i]-ch[m-1])) <= 0) m--;

ch[m++] = p[i];

}

int k = m;

for (int i = n-2; i >= 0; i--) {

/\* 可共线 \*/

//while (m > k && dcmp(getCross(ch[m-1]-ch[m-2], p[i]-ch[m-2])) < 0) m--;

while (m > k && dcmp(getCross(ch[m-1]-ch[m-2], p[i]-ch[m-2])) <= 0) m--;

ch[m++] = p[i];

}

if (n > 1) m--;

return m;

}

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

1. 点集面积和：计算给出点集组成的任意多边形面积和

例题：zoj 3871（给定点集）

题目大意：给定一个点集，计算任意点组成的多形的面积和

解题思路： 枚举每条边，计算该条边的贡献值。因为给定集合非凸包，所以对于每个点，要预处理其它点与它的极角，并按照极角排序，然后枚举另一点，则这条边即为当前枚举边，计算有多少个点在其右侧，即可计算。算法复杂度为o(n^2)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*hdu4760.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

const int mod = 998244353;

int N, pow2[maxn];

Point P[maxn], T[maxn \* 2];

inline bool cmp(const Point& a, const Point& b) { return a.ang < b.ang; }

int main () {

pow2[0] = 1;

for (int i = 1; i < maxn; i++) pow2[i] = pow2[i-1] \* 2 % mod;

for (int i = 0; i < maxn; i++) pow2[i] = (pow2[i] - 1 + mod) % mod;

int cas;

scanf("%d", &cas);

while (cas--) {

scanf("%d", &N);

for (int i = 0; i < N; i++) P[i].read();

ll ans = 0;

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

int sz = 0;

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

T[sz] = P[j];

T[sz++].ang = atan2(P[j].y-P[i].y, P[j].x-P[i].x);

}

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

T[j+sz] = T[j];

T[j+sz].ang += pi \* 2;

}

sort(T, T + sz \* 2, cmp);

int r = 0;

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

while (T[r+1].ang - T[j].ang < pi) r++;

ans = (ans + getArea(Point(0, 0), P[i], T[j]) % mod \* pow2[r-j]) % mod;

}

}

printf("%lld\n", ans);

}

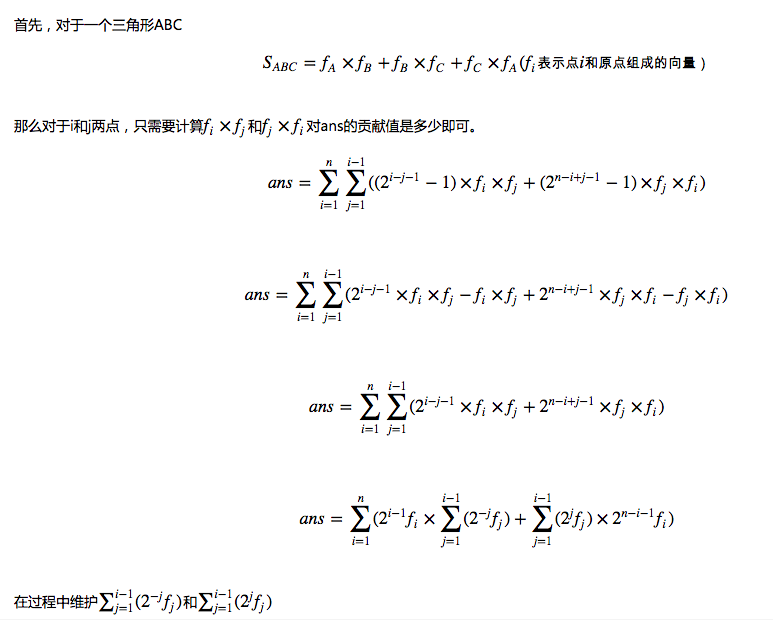
return 0;

}

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

例题：hdu 5448（给定凸包）

题目大意：给定一个点集，点集为凸包，计算任意点组成的多形的面积和

解题思路：算法复杂度o(n) 

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*hdu5448.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

typedef long long ll;

const int maxn = 100005;

const int mod = 1e9 + 7;

struct Point {

int x, y;

Point(int x = 0, int y = 0): x(x), y(y) {}

void read () { scanf("%d%d", &x, &y); }

int operator \* (const Point& u) { return (1LL \* x \* u.y % mod - 1LL \* y \* u.x % mod + mod) % mod; }

Point operator \* (const int u) { return Point(1LL \* u \* x % mod, 1LL \* u \* y % mod); }

Point operator + (const Point& u) { return Point((x+u.x)%mod, (y+u.y)%mod); }

};

int N, mul[maxn], inv[maxn];

int mul\_mod(ll x, int n, int mod) {

int ret = 1;

while (n) {

if (n&1) ret = x \* ret % mod;

x = x \* x % mod;

n >>= 1;

}

return ret;

}

int main () {

mul[0] = inv[0] = 1;

for (int i = 1; i < maxn; i++) mul[i] = 2 \* mul[i-1] % mod;

ll inv2 = mul\_mod(2, mod-2, mod);

for (int i = 1; i < maxn; i++) inv[i] = inv2 \* inv[i-1] % mod;

int cas;

scanf("%d", &cas);

while (cas--) {

int ans = 0;

scanf("%d", &N);

Point fi, fg, fh;

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

fi.read();

ans = (ans + 1LL \* mul[i-1] \* (fi \* fh) % mod) % mod;

ans = (ans + 1LL \* (i==N?inv2:mul[N-i-1]) \* (fg \* fi) % mod) % mod;

fg = fg + fi \* mul[i];

fh = fh + fi \* inv[i];

}

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

}

return 0;

}

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

2、三维凸包：

需要Point3.cpp

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ConvexHull3.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct Face {

int v[3];

Face (int a = 0, int b = 0, int c = 0) { v[0] = a, v[1] = b, v[2] = c;}

Vector3 normal (Point3 \*p) const { return getCross(p[v[1]] - p[v[0]], p[v[2]]-p[v[0]]); }

int cansee (Point3 \*p, int i) const {

return getDot(p[i]-p[v[0]], normal(p)) > 0 ? 1 : 0;

}

};

int vis[1005][1005];

double rand01() { return rand() / (double) RAND\_MAX; }

double randeps() { return (rand01() - 0.5) \* eps; }

Point3 addNoise(Point3 p) { return Point3(p.x+randeps(), p.y+randeps(), p.z+randeps()); }

vector<Face> CH3D (Point3 \*o, int n, Point3\* p) {

for (int i = 0; i < n; i++) p[i] = addNoise(o[i]);

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

vector<Face> cur;

cur.push\_back(Face(0, 1, 2));

cur.push\_back(Face(2, 1, 0));

for (int i = 3; i < n; i++) {

vector<Face> net;

for (int j = 0; j < cur.size(); j++) {

Face& f = cur[j];

int res = f.cansee(p, i);

if (!res) net.push\_back(f);

for (int k = 0; k < 3; k++) vis[f.v[k]][f.v[(k+1)%3]] = res;

}

for (int j = 0; j < cur.size(); j++) {

for (int k = 0; k < 3; k++) {

int a = cur[j].v[k], b = cur[j].v[(k+1)%3];

if (vis[a][b] != vis[b][a] && vis[a][b])

net.push\_back(Face(a, b, i));

}

}

cur = net;

}

return cur;

}

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

（二）旋转卡壳

需要Point2.cpp

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*rotatingCalipers.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 旋转卡壳 \*/

void rotatingCalipers(Point \*p, int n, vector<pii>& sol) {

sol.clear();

int j = 1; p[n] = p[0];

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

while (getCross(p[j+1]-p[i+1], p[i]-p[i+1]) > getCross(p[j]-p[i+1], p[i]-p[i+1]))

j = (j+1) % n;

sol.push\_back(make\_pair(i, j));

sol.push\_back(make\_pair(i + 1, j + 1));

}

}

/\* 求包含凸包点集的最小矩形，分面积和周长 \*/

void rotatingCalipersGetRectangle (Point \*p, int n, double& area, double& perimeter) {

p[n] = p[0];

int l = 1, r = 1, j = 1;

area = perimeter = 1e20;

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

Vector v = (p[i+1]-p[i]) / getLength(p[i+1]-p[i]);

while (dcmp(getDot(v, p[r%n]-p[i]) - getDot(v, p[(r+1)%n]-p[i])) < 0) r++;

while (j < r || dcmp(getCross(v, p[j%n]-p[i]) - getCross(v,p[(j+1)%n]-p[i])) < 0) j++;

while (l < j || dcmp(getDot(v, p[l%n]-p[i]) - getDot(v, p[(l+1)%n]-p[i])) > 0) l++;

double w = getDot(v, p[r%n]-p[i])-getDot(v, p[l%n]-p[i]);

double h = getDistanceToLine (p[j%n], p[i], p[i+1]);

area = min(area, w \* h);

perimeter = min(perimeter, 2 \* w + 2 \* h);

}

}

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

（三）半平面交：线性规划

需要Point2.cpp 和Line3.cpp

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*HalfPlane.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct DirLine {

Point p;

Vector v;

double ang;

DirLine () {}

DirLine (Point p, Vector v): p(p), v(v) { ang = atan2(v.y, v.x); }

bool operator < (const DirLine& u) const { return ang < u.ang; }

};

bool onLeft(DirLine l, Point p) { return dcmp(l.v ^ (p-l.p)) >= 0; }

/\* 半平面相交 \*/

int halfPlaneIntersection(DirLine\* li, int n, Point\* poly) {

sort(li, li + n);

int first, last;

Point\* p = new Point[n];

DirLine\* q = new DirLine[n];

q[first=last=0] = li[0];

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

while (first < last && !onLeft(li[i], p[last-1])) last--;

while (first < last && !onLeft(li[i], p[first])) first++;

q[++last] = li[i];

if (dcmp(q[last].v ^ q[last-1].v) == 0) {

last--;

if (onLeft(q[last], li[i].p)) q[last] = li[i];

}

if (first < last)

getIntersection(q[last-1].p, q[last-1].v, q[last].p, q[last].v, p[last-1]);

}

while (first < last && !onLeft(q[first], p[last-1])) last--;

if (last - first <= 1) { delete [] p; delete [] q; return 0; }

getIntersection(q[last].p, q[last].v, q[first].p, q[first].v, p[last]);

int m = 0;

for (int i = first; i <= last; i++) poly[m++] = p[i];

delete [] p; delete [] q;

return m;

}

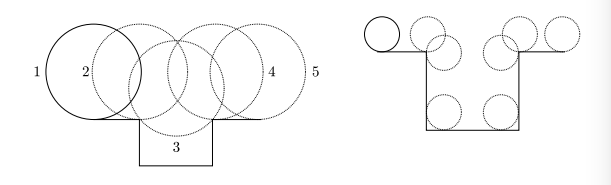
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

（四）放射矩阵

（五）运动规划

例题：uva 1017（轮廓线）

题目大意：求圆心经过的路线，给定折线的点的方式为与前一个点的相对关系



解题思路：将所有点依次连接起来形成一条曲线，圆心移动的轨迹其实就是一条时刻与它距离为r的曲线。

对于线段就是平移，对于点就是一个圆。要求的轨迹其实就是所有线段和圆的轮廓，所以从起始位置开始，每次暴力出下一要移动到的点，距离就是最终的和。

对于从圆上的点A移动到线段或是圆上的点B，角AOB（O为圆心）要尽量小。

对于从线段上的点A移动到线段或是圆上的点B，AB的距离要尽量小。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*uva1017.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <cmath>

#include <vector>

#include <algorithm>

using namespace std;

const double pi = 4 \* atan(1);

const double eps = 1e-8;

inline int dcmp (double x) { if (fabs(x) < eps) return 0; else return x < 0 ? -1 : 1; }

struct Point {

double x, y;

Point (double x = 0, double y = 0): x(x), y(y) {}

void read () { scanf("%lf%lf", &x, &y); }

bool operator == (const Point& u) const { return dcmp(x - u.x) == 0 && dcmp(y - u.y) == 0; }

bool operator != (const Point& u) const { return !(\*this == u); }

bool operator < (const Point& u) const { return x < u.x || (x == u.x && y < u.y); }

bool operator > (const Point& u) const { return u < \*this; }

bool operator <= (const Point& u) const { return \*this < u || \*this == u; }

bool operator >= (const Point& u) const { return \*this > u || \*this == u; }

Point operator + (const Point& u) { return Point(x + u.x, y + u.y); }

Point operator - (const Point& u) { return Point(x - u.x, y - u.y); }

Point operator \* (const double u) { return Point(x \* u, y \* u); }

Point operator / (const double u) { return Point(x / u, y / u); }

double operator \* (const Point& u) { return x\*u.y - y\*u.x; }

};

typedef Point Vector;

double getDot (Vector a, Vector b) { return a.x \* b.x + a.y \* b.y; }

double getCross (Vector a, Vector b) { return a.x \* b.y - a.y \* b.x; }

double getLength (Vector a) { return sqrt(getDot(a, a)); }

double getAngle (Vector u) { return atan2(u.y, u.x); }

double getAngle (Vector a, Vector b) { return acos(getDot(a, b) / getLength(a) / getLength(b)); }

/\* 直线pv和直线qw的交点 \*/

bool getIntersection (Point p, Vector v, Point q, Vector w, Point& o) {

if (dcmp(getCross(v, w)) == 0) return false;

Vector u = p - q;

double k = getCross(w, u) / getCross(v, w);

o = p + v \* k;

return true;

}

/\* 判断线段是否存在交点 \*/

bool haveIntersection (Point a1, Point a2, Point b1, Point b2) {

double c1=getCross(a2-a1, b1-a1), c2=getCross(a2-a1, b2-a1), c3=getCross(b2-b1, a1-b1), c4=getCross(b2-b1,a2-b1);

return dcmp(c1)\*dcmp(c2) <= 0 && dcmp(c3)\*dcmp(c4) <= 0;

/\* 加等号为可为端点 \*/

}

bool onSegment (Point p, Point a, Point b) {

/\* 可否在两端 \*/

if (p == a || p == b) return true;

return dcmp(getCross(a-p, b-p)) == 0 && dcmp(getDot(a-p, b-p)) < 0;

}

struct Circle {

Point o;

double r;

Circle () {}

Circle (Point o, double r = 0): o(o), r(r) {}

void read () { o.read(), scanf("%lf", &r); }

Point point(double rad) { return Point(o.x + cos(rad)\*r, o.y + sin(rad)\*r); }

double getArea (double rad) { return rad \* r \* r / 2; }

};

/\* 直线和圆的交点 \*/

int getLineCircleIntersection (Point p, Point q, Circle O, double& t1, double& t2, vector<Point>& sol) {

Vector v = q - p;

/\* 使用前需清空sol \*/

//sol.clear();

double a = v.x, b = p.x - O.o.x, c = v.y, d = p.y - O.o.y;

double e = a\*a+c\*c, f = 2\*(a\*b+c\*d), g = b\*b+d\*d-O.r\*O.r;

double delta = f\*f - 4\*e\*g;

if (dcmp(delta) < 0) return 0;

if (dcmp(delta) == 0) {

t1 = t2 = -f / (2 \* e);

sol.push\_back(p + v \* t1);

return 1;

}

t1 = (-f - sqrt(delta)) / (2 \* e); sol.push\_back(p + v \* t1);

t2 = (-f + sqrt(delta)) / (2 \* e); sol.push\_back(p + v \* t2);

return 2;

}

/\* 圆和圆的交点 \*/

int getCircleCircleIntersection (Circle o1, Circle o2, vector<Point>& sol) {

double d = getLength(o1.o - o2.o);

if (dcmp(d) == 0) {

if (dcmp(o1.r - o2.r) == 0) return -1;

return 0;

}

if (dcmp(o1.r + o2.r - d) < 0) return 0;

if (dcmp(fabs(o1.r-o2.r) - d) > 0) return 0;

double a = getAngle(o2.o - o1.o);

double da = acos((o1.r\*o1.r + d\*d - o2.r\*o2.r) / (2\*o1.r\*d));

Point p1 = o1.point(a-da), p2 = o1.point(a+da);

sol.push\_back(p1);

if (p1 == p2) return 1;

sol.push\_back(p2);

return 2;

}

const int maxn = 205;

const double inf = 0x3f3f3f3f3f3f3f;

struct Segment {

Point s, e;

Segment () {};

Segment (Point s, Point e): s(s), e(e) {}

};

double R;

int N, M, idx[maxn], nS, nC;

Point Q[maxn], P[maxn];

Segment Gseg[maxn];

Circle Gcir[maxn];

void init () {

M = 0;

nS = nC = 1;

for (int i = 0; i < N; i++) Q[i].read();

int n = 0;

P[n++] = Point(0, 0);

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

if (dcmp(Q[i].x))

P[n] = Point(P[n-1].x + Q[i].x, P[n-1].y), n++;

if (dcmp(Q[i].y))

P[n] = Point(P[n-1].x, P[n-1].y + Q[i].y), n++;

}

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

if (i) {

if (dcmp(P[i-1].x - P[i].x) == 0) {

if (dcmp(P[i-1].y - P[i].y) < 0)

Gseg[nS] = Segment(Point(P[i-1].x - R, P[i-1].y), Point(P[i].x - R, P[i].y));

else

Gseg[nS] = Segment(Point(P[i-1].x + R, P[i-1].y), Point(P[i].x + R, P[i].y));

} else

Gseg[nS] = Segment(Point(P[i-1].x, P[i-1].y + R), Point(P[i].x, P[i].y + R));

idx[M++] = -nS, nS++;

}

Gcir[nC] = Circle(P[i], R);

idx[M++] = nC++;

}

}

void handle (int u, int v, Point& p, int& idx, int cur, Point o) {

Point t = Point(inf , inf);

double k1, k2;

vector<Point> sol;

if (u < 0 && v < 0) {

u = -u, v = -v;

if (haveIntersection(Gseg[u].s, Gseg[u].e, Gseg[v].s, Gseg[v].e)) {

getIntersection(Gseg[u].s, Gseg[u].e-Gseg[u].s, Gseg[v].s, Gseg[v].e-Gseg[v].s, t);

double k1 = getLength(t - Gseg[u].s) / getLength(Gseg[u].e-Gseg[u].s);

double k2 = getLength(p - Gseg[u].s) / getLength(Gseg[u].e-Gseg[u].s);

if (idx == 0 || dcmp(k1 - k2) < 0 || (dcmp(k1-k2) == 0 && idx < cur))

idx = cur, p = t;

}

} else if (u < 0 && v > 0) {

u = -u;

int n = getLineCircleIntersection(Gseg[u].s, Gseg[u].e, Gcir[v], k1, k2, sol);

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

if (onSegment(sol[i], Gseg[u].s, Gseg[u].e)) {

double k1 = getLength(sol[i] - Gseg[u].s) / getLength(Gseg[u].e-Gseg[u].s);

double k2 = getLength(t - Gseg[u].s) / getLength(Gseg[u].e-Gseg[u].s);

if (dcmp(k1 - k2) < 0) t = sol[i];

}

}

double k1 = getLength(t - Gseg[u].s) / getLength(Gseg[u].e-Gseg[u].s);

double k2 = getLength(p - Gseg[u].s) / getLength(Gseg[u].e-Gseg[u].s);

if (idx == 0 || dcmp(k1 - k2) < 0 || (dcmp(k1-k2) == 0 && idx < cur))

idx = cur, p = t;

} else if (u > 0 && v < 0) {

v = -v;

double rad = inf;

int n = getLineCircleIntersection(Gseg[v].s, Gseg[v].e, Gcir[u], k1, k2, sol);

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

if (onSegment(sol[i], Gseg[v].s, Gseg[v].e)) {

double tmp = (o == sol[i] ? 0 : getAngle(o-Gcir[u].o, sol[i]-Gcir[u].o));

if (dcmp((o-Gcir[u].o) \* (sol[i]-Gcir[u].o)) > 0) tmp = 2 \* pi - tmp;

if (dcmp(rad - tmp) >= 0)

rad = tmp, t = sol[i];

}

}

double k = (o == p ? 0 : getAngle(o-Gcir[u].o, p-Gcir[u].o));

if (dcmp((o-Gcir[u].o) \* (p-Gcir[u].o)) > 0) k = 2 \* pi - k;

if (idx == 0 || dcmp(rad - k) < 0 || (dcmp(rad-k) == 0 && idx < cur))

idx = cur, p = t;

} else if (u > 0 && v > 0) {

double rad = inf;

int n = getCircleCircleIntersection (Gcir[u], Gcir[v], sol);

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

double tmp = (o == sol[i] ? 0 : getAngle(o-Gcir[u].o, sol[i]-Gcir[u].o));

if (dcmp((o-Gcir[u].o) \* (sol[i]-Gcir[u].o)) > 0) tmp = 2 \* pi - tmp;

if (dcmp(rad - tmp) >= 0)

rad = tmp, t = sol[i];

}

double k = (o == p ? 0 : getAngle(o-Gcir[u].o, p-Gcir[u].o));

if (dcmp((o-Gcir[u].o) \* (p-Gcir[u].o)) > 0) k = 2 \* pi - k;

if (idx == 0 || dcmp(rad - k) < 0 || (dcmp(rad-k) == 0 && idx < cur))

idx = cur, p = t;

}

}

double solve () {

int mv = 0;

double ans = 0, rad = 0;

Point s = P[0] + Point(0, R);

while (mv + 1 < M) {

int re = 0;

Point e;

for (int i = mv + 1; i < M; i++)

handle(idx[mv], idx[i], e, re, i, s);

if (idx[mv] > 0) {

int u = idx[mv];

double tmp = getAngle(s-Gcir[u].o, e-Gcir[u].o);

if (dcmp((s-Gcir[u].o) \* (e-Gcir[u].o)) > 0) tmp = 2 \* pi - tmp;

rad += tmp;

} else if (dcmp(s.x - e.x) == 0)

ans += fabs(e.y - s.y);

else if (dcmp(s.y - e.y) == 0)

ans += fabs(e.x - s.x);

s = e, mv = re;

}

return ans + rad \* R;

}

int main () {

int cas = 1;

while (scanf("%lf%d", &R, &N) == 2) {

if (R == 0 && N == 0) break;

init ();

printf("Case %d: Distance = %.3lf\n\n", cas++, solve());

}

return 0;

}

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

第四章 图论

一、基础图论

（一）拓扑排序

对一个有向无环图(Directed Acyclic Graph简称DAG)G进行拓扑排序，是将G中所有顶点排成一个线性序列，使得图中任意一对顶点u和v，若边(u,v)∈E(G)，则u在线性序列中出现在v之前。通常，这样的线性序列称为满足拓扑次序(Topological Order)的序列，简称拓扑序列。简单的说，由某个集合上的一个偏序得到该集合上的一个全序，这个操作称之为拓扑排序。

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*TopSort.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <queue>

#include <vector>

#include <algorithm>

using namespace std;

const int maxn = 1e5 + 5;

int in[maxn];

void topSort(int\* a, int n, vector<int>\* g) {

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

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

for (int j = 0; j < g[i].size(); j++) {

int v = g[i][j];

in[v]++;

}

}

queue<int> que;

for (int i = 0; i < n; i++) if (!in[i])

que.push(i);

int mv = n-1;

while (!que.empty()) {

int u = que.front();

que.pop();

a[mv--] = u;

for (int i = 0; i < g[u].size(); i++) {

int v = g[u][i];

in[v]--;

if (in[v] == 0) que.push(v);

}

}

}

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

（二）欧拉路

如果给定无孤立结点图G，若存在一条路，经过图中每边一次且仅一次，这条路称为欧拉路；

如果给定无孤立结点图G，若存在一条回路，经过图中每边一次且仅一次，那么该回路称为欧拉回路。

欧拉路的判定：图联通，并且至多有两个点的度数为奇数，其它点的度数为偶数。

欧拉回路的判定：图联通，点的度数为偶数。

路径输出：Feurly算法。

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Feurly.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <vector>

#include <algorithm>

using namespace std;

typedef pair<int,int> pii;

const int maxn = 1005;

int top = 0, vis[maxn], path[maxn];

vector<pii> G[maxn];

void feurly (int u) {

for (int i = 0; i < G[u].size(); i++) {

int e = G[u][i].first, v = G[u][i].second;

if (vis[e]) continue;

vis[e] = 1;

feurly(v);

path[top++] = e;

}

}

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

二、深度优先遍历

（一）无向图割顶和桥

1、割顶：

对于无向图G，如果删除某个点u后，连通分量数目增加，称u为图的关节点（articulation vertex）或割顶（cut vertex）。

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CutPoint.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <vector>

#include <algorithm>

using namespace std;

const int maxn = 1e5 + 5; // The number of Node

int dfsclock, pre[maxn];

bool iscut[maxn];

vector<int> G[maxn];

int dfs (int u, int f) {

int lowu = pre[u] = ++dfsclock, child = 0;

for (int i = 0; i < G[u].size(); i++) {

int v = G[u][i];

if (!pre[v]) {

child++;

int lowv = dfs(v, u);

lowu = min(lowu, lowv);

if (lowv >= pre[u]) iscut[u] = true;

} else if (pre[v] < pre[u] && v != f)

lowu = min(lowu, pre[v]);

}

if (f < 0 && child == 1) iscut[u] = false;

return lowu;

}

void findCutPoint(int n) {

dfsclock = 0;

memset(pre, 0, sizeof(pre));

memset(iscut, 0, sizeof(iscut));

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

if (!pre[i]) dfs(i, -1);

}

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

2、桥：

对于无向图G，如果删除某条边e后，连通分量数目增加，称e为图的桥（bridge）或割顶（cut edge）。

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CutPoint.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <vector>

#include <algorithm>

using namespace std;

const int maxn = 1e4 + 5; // The number of Node

const int maxm = 1e6 + 5; // The number of Edge

int dfsclock, pre[maxn];

bool iscut[maxm \* 2];

int first[maxn], jump[maxm \* 2], linker[maxm \* 2];

int dfs (int u, int fa) {

int lowu = pre[u] = ++dfsclock;

for (int i = first[u]; i != -1; i = jump[i]) {

int v = linker[i];

if (!pre[v]) {

int lowv = dfs(v, u);

lowu = min(lowu, lowv);

if (lowv > pre[u]) iscut[i] = iscut[i^1] = true; // 正反向边

} else if (pre[v] < pre[u] && v != fa)

lowu = min(lowu, pre[v]);

}

return lowu;

}

void findEdge (int n) {

dfsclock = 0;

memset(pre, 0, sizeof(pre));

memset(iscut, 0, sizeof(iscut));

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

if (!pre[i]) dfs(i, -1);

}

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

（二）无向图双连通分量

1、点双连通：

对于一个连通图，如果任意两点至少存在两条点不重复的路径，则称这个图是点-双连通的。

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Point-biconnected.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <vector>

#include <stack>

#include <algorithm>

using namespace std;

const int maxn = 1e5 + 5;

struct Edge {

int from, to, val;

Edge(int from = 0, int to = 0, int val = 0): from(from), to(to), val(val) {}

};

int dfsclock, cntbcc, pre[maxn], bccno[maxn];

bool iscut[maxn];

vector<int> G[maxn], BCC[maxn];

stack<Edge> S;

int dfs (int u, int fa) { // 割顶的bccno无意义

int lowu = pre[u] = ++dfsclock, child = 0;

for (int i = 0; i < G[u].size(); i++) {

int v = G[u][i];

Edge e = Edge(u, v);

if (!pre[v]) {

child++;

S.push(e);

int lowv = min(lowu, lowv);

if (lowv >= pre[u]) {

iscut[u] = 1;

BCC[++cntbcc].clear();

while (true) {

Edge x = S.top();

S.pop();

if (bccno[x.from] != cntbcc) {

BCC[cntbcc].push\_back(x.from);

bccno[x.from] = cntbcc;

}

if (bccno[x.to] != cntbcc) {

BCC[cntbcc].push\_back(x.to);

bccno[x.to] = cntbcc;

}

if (x.from == u && x.to == v) break;

}

}

} else if (pre[v] < pre[u] && v != fa) {

S.push(e);

lowu = min(lowu, pre[v]);

}

}

if (fa < 0 && child == 1) iscut[u] = 0;

return lowu;

}

void findBCC(int n) {

dfsclock = cntbcc = 0;

memset(pre, 0, sizeof(pre));

memset(iscut, 0, sizeof(iscut));

memset(bccno, 0, sizeof(bccno));

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

if (!pre[i]) dfs(i, -1);

}

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

2、边双连通：

对于一个连通图，如果任意两点至少存在两条边不重复的路径，则称这个图是边-双连通的。

除了桥不属于任何边-双连通分量之外，其他每条边恰好属于一个边-双连通分量。

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Edge-biconnected.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <vector>

#include <algorithm>

using namespace std;

const int maxn = 1e4 + 5; // The number of Node

const int maxm = 1e6 + 5; // The number of Edge

/\* 如果要求边双联通分量，只需在做一遍dfs，保证不经过割边即可。\*/

int dfsclock, pre[maxn], cntbcc, bccno[maxn];

bool iscut[maxm \* 2];

int first[maxn], jump[maxm \* 2], linker[maxm \* 2];

int dfs (int u, int fa) {

int lowu = pre[u] = ++dfsclock;

for (int i = first[u]; i != -1; i = jump[i]) {

int v = linker[i];

if (!pre[v]) {

int lowv = dfs(v, u);

lowu = min(lowu, lowv);

if (lowv > pre[u])

iscut[i] = iscut[i^1] = 1;

} else if (pre[v] < pre[u] && v != fa)

lowu = min(lowu, pre[v]);

}

return lowu;

}

void dfs (int u) {

pre[u] = ++dfsclock;

bccno[u] = cntbcc;

for (int i = first[u]; i + 1; i = jump[i]) {

if (iscut[i]) continue;

int v = linker[i];

if (!pre[v]) dfs(v);

}

}

void findEdge (int n) {

dfsclock = 0;

memset(pre, 0, sizeof(pre));

memset(iscut, 0, sizeof(iscut));

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

if (!pre[i]) dfs(i, -1);

dfsclock = cntbcc = 0;

memset(bccno, 0, sizeof(bccno));

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

if (!pre[i]) { cntbcc++; dfs(i); }

}

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

（三）有向图强连通分量

在有向图中，任意两点之间存在相互可达的路径，则称这个图为强连通的。

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Strong-onnected.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <vector>

#include <stack>

#include <algorithm>

using namespace std;

const int maxn = 1e5 + 5;

stack<int> S;

vector<int> G[maxn];

int dfsclock, cntscc, sccno[maxn], pre[maxn];

int dfs(int u) {

int lowu = pre[u] = ++dfsclock;

S.push(u);

for (int i = 0; i < G[u].size(); i++) {

int v = G[u][i];

if (!pre[v]) {

int lowv = dfs(v);

lowu = min(lowu, lowv);

} else if (!sccno[v])

lowu = min(lowu, pre[v]);

}

if (lowu == pre[u]) {

cntscc++;

while (true) {

int x = S.top();

S.pop();

sccno[x] = cntscc;

if (x == u) break;

}

}

return lowu;

}

void findSCC(int n) {

dfsclock = cntscc = 0;

memset(pre, 0, sizeof(pre));

memset(sccno, 0, sizeof(sccno));

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

if (!pre[i]) dfs(i);

}

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

（四）2-SAT

有n个布尔变量xi，另有m个需要满足的条件，每个条件的形式都是“xi为真/假或者xj为真/假”

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*TwoSAT.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <vector>

#include <algorithm>

using namespace std;

const int maxn = 1e5 + 5;

struct TwoSAT {

int n, s[maxn \* 2], c;

bool mark[maxn \* 2];

vector<int> g[maxn \* 2];

void init (int n) {

this->n = n;

memset(mark, false, sizeof(mark));

for (int i = 0; i < n\*2; i++) g[i].clear();

}

void addClause(int x, int xval, int y, int yval) { // (sx || sy)

x = x \* 2 + xval;

y = y \* 2 + yval;

g[x^1].push\_back(y);

g[y^1].push\_back(x);

}

bool dfs (int x) {

if (mark[x^1]) return false;

if (mark[x]) return true;

mark[x] = true;

s[c++] = x;

for (int i = 0; i < g[x].size(); i++)

if (!dfs(g[x][i])) return false;

return true;

}

bool solve () {

for (int i = 0; i < n\*2; i += 2) {

if (!mark[i] && !mark[i+1]) {

c = 0;

if (!dfs(i)) {

while (c) mark[s[--c]] = false;

if (!dfs(i+1)) return false;

}

}

}

return true;

}

};

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

例题：uva 12273（2sat变形+限制条件）

题目大意：给定一个DNA序列，然后给定一些回文位置的集合，判断是否可以通过-1，0，+1的变换，使得序列满足。

解题思路： 每个位置拆成4个状态考虑，建立关系，并且将每个位置不可转变的形态预先标记。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*uva12273.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <vector>

#include <algorithm>

using namespace std;

const int maxn = 1e4 + 5;

struct TwoSAT {

int n, s[maxn \* 4], c;

bool mark[maxn \* 4], must[maxn \* 4];

vector<int> g[maxn \* 4];

void init (int n) {

this->n = n;

memset(mark, 0, sizeof(mark));

memset(must, 0, sizeof(must));

for (int i = 0; i < 4 \* n; i++) g[i].clear();

}

void addLink(int x, int y) { g[x].push\_back(y); }

bool dfs (int u) {

for (int i = 1; i <= 3; i++)

if (mark[u^i]) return false;

if (must[u]) return false;

if (mark[u]) return true;

mark[u] = true;

s[c++] = u;

for (int i = 0; i < g[u].size(); i++)

if (!dfs(g[u][i])) return false;

return true;

}

void draw(int u) {

if (must[u]) return;

must[u] = true;

for (int i = 0; i < g[u].size(); i++)

draw(g[u][i]);

}

bool solve () {

for (int i = 0; i < 4 \* n; i += 4) {

if (!mark[i] && !mark[i+1] && !mark[i+2] && !mark[i+3]) {

bool flag = true;

c = 0;

for (int k = 0; k < 4 && flag; k++) {

if (must[i+k]) continue;

while (c) mark[s[--c]] = false;

if (dfs(i+k)) flag = false;

}

if (flag) return false;

}

}

return true;

}

}solver;

bool flag;

int N, M, a[maxn];

char S[maxn];

inline int idx(char c) {

if (c == 'A') return 0;

else if (c == 'G') return 1;

else if (c == 'T') return 2;

else return 3;

}

void addClause(int p, int q) {

for (int i = 0; i < 4; i++) {

solver.addLink(p \* 4 + i, q \* 4 + i);

solver.addLink(q \* 4 + i, p \* 4 + i);

}

}

void init () {

solver.init(N);

scanf("%s", S);

int k, x;

while (M--) {

scanf("%d%\*c", &k);

for (int i = 0; i < k; i++) scanf("%d", &a[i]);

for (int i = 0; i < k/2; i++) addClause(a[i], a[k-i-1]);

}

for (int i = 0; i < N; i++) { // 每个位置不能连续变换两次

int v = (idx(S[i]) + 2) % 4;

solver.draw(i\*4 + v);

}

for (int i = 1; i < N; i++) { // 相邻位置只能变化一个

int u = idx(S[i-1]), v = idx(S[i]);

for (int j = 1; j < 4; j++) {

int tu = (u + j) % 4, tv = (v + j) % 4;

solver.addLink((i-1)\*4+tu, i\*4+v);

solver.addLink(i\*4+tv, (i-1)\*4+u);

}

}

}

int main () {

while (scanf("%d%d", &N, &M) == 2 && N + M) {

init();

printf("%s\n", solver.solve() ? "YES" : "NO");

}

return 0;

}

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

三、最短路

四、生成树

（一）最小生成树

最小生成树：给出加权无向图，求一棵生成树，使得树的权值和最小。

最小瓶颈生成树：给出加权无向图，求一棵生成树，使得最大边权值尽量小。

最小瓶颈路：给定加权无向图的两个节点u和v，求从u到v的一条路径，使得路径上的最长边尽量短。

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MST.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

const int maxn = 1e4 + 5; // The number of Node;

const int maxm = 1e6 + 5; // The number of Edge;

struct Edge {

int u, v, w;

Edge(int u = 0, int v = 0, int w = 0): u(u), v(v), w(w) {}

bool operator < (const Edge& a) const { return w < a.w; }

} E[maxm];

int F[maxn];

int find(int x) { return x == F[x] ? x : F[x] = find(F[x]); }

int Kruskal(int n, int m, Edge\* e) {

int ret = 0;

sort(e, e + m);

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

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

int u = e[i].u, v = e[i].v, w = e[i].w;

if (find(u) != find(v)) {

F[find(u)] = find(v);

n--;

ret += w;

}

}

if (n != 1) {} // Can not build a MST

return ret;

}

/\* 最小瓶颈树：从一个空图开始，按照权值从小到大加入，图第一次完全联通时，该图的最小生成树即为原图的最小瓶颈生成树\*/

/\* 最小瓶颈路：求出图的最小生成树，则起点和终点间的唯一路径上权值最大的边即为要求的瓶颈\*/

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

（二）次小生成树

次小生成树：先对给定无向图做最小生成树，然后枚举一条非最小生成树的树边，在加入这条边后，图上会有回路，因此删除回路上权值最大的边。

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SMST.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

typedef pair<int,int> pii;

const int maxn = 1e4 + 5; // The number of Node;

const int maxm = 1e6 + 5; // The number of Edge;

const int inf = 0x3f3f3f3f;

struct Edge {

int u, v, w;

Edge(int u = 0, int v = 0, int w = 0): u(u), v(v), w(w) {}

bool operator < (const Edge& a) const { return w < a.w; }

}E[maxm];

int N, M, F[maxn], D[maxn][maxn], use[maxm];

vector<pii> G[maxn];

int find (int x) { return x == F[x] ? x : F[x] = find(F[x]); }

void dfs (int u, int fa, int w, int\* d) {

d[u] = w;

for (int i = 0; i < G[u].size(); i++) {

int v = G[u][i].first;

if (v == fa) continue;

dfs(v, u, max(w, G[u][i].second), d);

}

}

int Kruskal(int n, int m, Edge\* e) {

int ret = 0;

sort(e, e + m);

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

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

int u = e[i].u, v = e[i].v, w = e[i].w;

if (find(u) != find(v)) {

n--;

use[i] = 1;

ret += w;

F[find(u)] = find(v);

G[u].push\_back(make\_pair(v, w));

G[v].push\_back(make\_pair(u, w));

}

}

if (n != 1) {} // Can not build a MST

return ret;

}

int SMST (int n, int m, Edge\* e) {

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

F[i] = i; G[i].clear();

}

memset(use, 0, sizeof(use));

int ans = Kruskal(n, m, e), ret = inf;

for (int i = 1; i <= N; i++) dfs(i, 0, 0, D[i]);

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

if (use[i]) continue;

int u = e[i].u, v = e[i].v, w = e[i].w;

ret = min(ret, ans + w - D[u][v]);

}

return ret;

}

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

（三）最小有向生成树

有向生成树（directed spanning tree）：恰好有一个入度为0的节点（根节点），其它点入度均为1，可以从根节点到达所有其它节点。

最小有向生成树：给定一个有向带权图G和其中一个节点u，找出一个以u为根节点，权值和最小的有向生成树。

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DMST.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <vector>

#include <algorithm>

using namespace std;

typedef long long ll;

const int inf = 0x3f3f3f3f;

const int maxn = 1005;

struct DMST {

struct Edge {

int u, v, d;

Edge(int u = 0, int v = 0, int d = 0): u(u), v(v), d(d) {}

bool operator < (const Edge& u) const { return d < u.d; }

};

int in[maxn], pre[maxn], id[maxn], vis[maxn];

vector<Edge> edges;

void init () { edges.clear(); }

void addEdge(int u, int v, int d) { edges.push\_back(Edge(u, v, d)); }

bool directedMST(int root, int n, int& ans) {

ans = 0;

while (true) {

// 删除自环+找最小入边

memset(in, inf, sizeof(in));

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

int u = edges[i].u, v = edges[i].v;

if (edges[i].d < in[v] && u != v) {

pre[v] = u;

in[v] = edges[i].d;

}

}

// 判断是否可以到达所有节点

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

if (i == root) continue;

if (in[i] == inf) return false;

}

int cntnode = 0;

//memset(id, -1, sizeof(int) \* n);

//memset(vis, -1, sizeof(int) \* n);

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

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

in[root] = 0;

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

ans += in[i];

int v = i;

while (vis[v] != i && id[v] == -1 && v != root) {

vis[v] = i;

v = pre[v];

}

if (v != root && id[v] == -1) {

for (int u = pre[v]; u != v; u = pre[u])

id[u] = cntnode;

id[v] = cntnode++;

}

}

// 没有找到环, 终止

if (cntnode == 0) break;

// 缩点，重新标记

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

if (id[i] == -1)

id[i] = cntnode++;

}

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

int v = edges[i].v;

edges[i].u = id[edges[i].u];

edges[i].v = id[edges[i].v];

if (edges[i].u != edges[i].v)

edges[i].d -= in[v];

}

n = cntnode;

root = id[root];

}

return true;

}

};

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

五、二分图

（一）二分图判定

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Bipartite.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <vector>

#include <algorithm>

using namespace std;

const int maxn = 1e6 + 5;

struct Bipartite {

int n, color[maxn];

vector<int> g[maxn];

void init (int n) {

this->n = n;

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

for (int i = 0; i <= n; i++) g[i].clear();

}

void addEdge(int u, int v) {

g[u].push\_back(v);

g[v].push\_back(u);

}

bool dfs(int u) {

for (int i = 0; i < g[u].size(); i++) {

int v = g[u][i];

if (color[u] == color[v]) return false;

if (!color[v]) {

color[v] = 3 - color[u];

if (!dfs(v)) return false;

}

}

return true;

}

};

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

（二）最大匹配

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*KM.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <vector>

#include <algorithm>

using namespace std;

const int maxn = 1e5 + 5;

int N, L[maxn];

bool T[maxn];

vector<int> G[maxn];

bool match(int u) {

for (int i = 0; i < G[u].size(); i++) {

int v = G[u][i];

if (!T[v]) {

T[v] = true;

if (!L[v] || match(L[v])) {

L[v] = u;

return true;

}

}

}

return false;

}

int KM () {

int ret = 0;

memset(L, 0, sizeof(L));

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

memset(T, false, sizeof(T));

if (match(i)) ret++;

}

return ret;

}

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

（三）最佳完美匹配

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*PKM.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <cmath>

#include <algorithm>

using namespace std;

const int maxn = 105;

const int inf = 0x3f3f3f3f;

int N, L[maxn], Lx[maxn], Ly[maxn], W[maxn][maxn], slack[maxn];

bool S[maxn], T[maxn];

int match (int u) {

S[u] = true;

for (int i = 1; i <= N; i++) if (!T[i]) {

if (Lx[u] + Ly[i] == W[u][i]) {

T[i] = true;

if (!L[i] || match(L[i])) {

L[i] = u;

return true;

}

} else

slack[i] = min(slack[i], Lx[u]+Ly[i]-W[u][i]);

}

return false;

}

void update () {

int a = inf;

for (int i = 1; i <= N; i++) if (!T[i])

a = min(a, slack[i]);

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

if (S[i]) Lx[i] -= a;

if (T[i]) Ly[i] += a;

}

}

void KM () {

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

L[i] = Lx[i] = Ly[i] = 0;

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

Lx[i] = max(Lx[i], W[i][j]);

}

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

for (int j = 1; j <= N; j++) slack[j] = inf;

while (true) {

for (int j = 1; j <= N; j++) S[j] = T[j] = false;

if (match(i)) break;

else update();

}

}

}

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

（四）稳定婚姻问题

例题：uva 1175（稳定婚姻问题）

题目大意：n个男生，n个女生，每个人都对异性有一个排序，代表对他们的喜欢程度。你的任务是将男生和女生一一配对，使得男生u和女生v不存在以下情况：（1）男生u和女生v不是一对；（2）他们喜欢对方的程度都大于各自当前的舞伴。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*uva1175.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <queue>

#include <algorithm>

using namespace std;

const int maxn = 1005;

int N, pref[maxn][maxn], order[maxn][maxn], jump[maxn];

int future\_husband[maxn], future\_wife[maxn];

queue<int> Q;

void engage (int man, int woman) {

int m = future\_husband[woman];

if (m) {

future\_wife[m] = 0;

Q.push(m);

}

future\_wife[man] = woman;

future\_husband[woman] = man;

}

int main () {

int cas;

scanf("%d", &cas);

while (cas--) {

scanf("%d", &N);

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

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

jump[i] = 1;

future\_wife[i] = 0;

Q.push(i);

}

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

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

int x;

scanf("%d", &x);

order[i][x] = j;

}

future\_husband[i] = 0;

}

while (!Q.empty()) {

int man = Q.front();

Q.pop();

int woman = pref[man][jump[man]++];

if (!future\_husband[woman]) engage(man, woman);

else if (order[woman][man] < order[woman][future\_husband[woman]])

engage(man, woman);

else

Q.push(man);

}

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

printf("%d\n", future\_wife[i]);

if (cas) printf("\n");

}

return 0;

}

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