Contents

[1. FOR BIT 1](#_Toc26106242)

[2. TERNARY SEARCH 2](#_Toc26106243)

[3. DINIC 2](#_Toc26106244)

[4. TARJAN 2](#_Toc26106245)

[5. CAY KHUNG NHO NHAT KRUSKAL + DISJOIN SET 3](#_Toc26106246)

[6. PST 3](#_Toc26106247)

[7. IMPLICIT ST 4](#_Toc26106248)

[8. DIJKSTRA 4](#_Toc26106249)

[9. MO 5](#_Toc26106250)

[10. LCA 5](#_Toc26106251)

[11. MOD INVERSE 5](#_Toc26106252)

[12. MATRIX POWER 5](#_Toc26106253)

[13. TRIE 6](#_Toc26106254)

[14. FFT 6](#_Toc26106255)

[15. GEOMETRY 7](#_Toc26106256)

[16. EDGE ARRAY 10](#_Toc26106257)

[17. HASH 11](#_Toc26106258)

[18. KMP 11](#_Toc26106259)

[19. SUFFIX ARRAY AND LCP 11](#_Toc26106260)

[20. MANACHER 12](#_Toc26106261)

[21. Z-FUNCTION 12](#_Toc26106262)

[22. TÌM CHỮ SỐ THỨ K TRONG DÃY LIÊN TỤC + XÓA CHẴN LẺ 12](#_Toc26106263)

[23. LAZY SEGMENT TREE (IT) 13](#_Toc26106264)

[24. POLLARD’S RHO 13](#_Toc26106265)

[25. PHI 14](#_Toc26106266)

[26. FLOYD’S CYCLE FINDING 14](#_Toc26106267)

[27. GAUSSIAN ELIMINATION 14](#_Toc26106268)

[28. ARRAY’S MAXIMUM XOR 15](#_Toc26106269)

[29. EULERIAN GRAPH 15](#_Toc26106270)

[30. BINARY SUBSEQUENCE SUM 15](#_Toc26106271)

[31. CONVEX HULL TRICK 15](#_Toc26106272)

[32. RABIN-MILLER TEST 16](#_Toc26106273)

[33. LINE SWEEP – CLOSEST POINTS 17](#_Toc26106274)

[34. HOPCROFTKARP – MAX MATCHING IN GRAPH 18](#_Toc26106275)

[35. INDEX TREE 19](#_Toc26106276)

[36. NUMBER THEORY 19](#_Toc26106277)

[37. LINEAR EQUATIONS + MATRIX INVERSE + DETERMINANT 20](#_Toc26106278)

[38. THEOREMS 21](#_Toc26106279)

[39. MIN COST MATCHING 22](#_Toc26106280)

[40. MIN COST MAX FLOW 23](#_Toc26106281)

[41. CÔNG THỨC TOÁN HỌC 24](#_Toc26106282)

[42. DEFAULT CODE 24](#_Toc26106283)

[43. BIGNUM 24](#_Toc26106284)

1. FOR BIT

**for** **(**int split **=** mask **&** **(**mask**-**1**);** split**;** split **=** **(**split**-**1**)** **&** mask**)**

1. TERNARY SEARCH

FOR(i,0,silk.vertexNumber-1) {

ll LL = +1;

ll RR = i+silk.vertexNumber-1;

ll ML=(LL+LL+RR)/3;

ll MR=(LL+RR+RR)/3;

while ((LL!=ML) && (ML!=MR) && (MR!=RR)) {

if (calDistance(silk.vertex[i], silk.vertex[ML]) < calDistance(silk.vertex[i], silk.vertex[MR])) LL=ML;

else RR=MR;

ML=(LL+LL+RR)/3;

MR=(LL+RR+RR)/3;

}

FOR(j,LL,RR) res = max(res, calDistance(silk.vertex[i], silk.vertex[j]));

}

1. DINIC

const int MAX = 1000000000;

const int oo = 0x3c3c3c3c;

struct Edge {

int from, to, cap, flow, index;

Edge(int from, int to, int cap, int flow, int index) : from(from), to(to), cap(cap), flow(flow), index(index) {}

};

struct Dinic {

int N, flow, t;

vector<int> d, Dfs, ptr;

vector<vector<Edge> > G;

queue<int> q;

Dinic(int N) {

this->N = N;

G.resize(N);

ptr.resize(N);

d.resize(N);

Dfs.resize(N);

flow = t = 0;

}

void addEdge(int u, int v, int gt) {

G[u].push\_back(Edge(u, v, gt, 0, G[v].size()));

if (u == v) G[u].back().index++;

G[v].push\_back(Edge(v, u, 0, 0, G[u].size() - 1));

}

bool bfs(int S, int T) {

FOR(i,0,N-1) d[i] = 0;

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

q.push(S); d[S]=1;

while (!q.empty()) {

int u = q.front();

q.pop();

if (u == T) return true;

FOR(i,0,sz(G[u])-1) {

int v = G[u][i].to;

if (!d[v] && G[u][i].cap - G[u][i].flow > 0) {

q.push(v);

d[v] = d[u] + 1;

}

}

}

return false;

}

int visit(int u, int Min, int T) {

if (u == T) return Min;

if (Dfs[u]!=t) Dfs[u]=t;

else return 0;

for (; ptr[u] < (int) G[u].size(); ++ptr[u]) {

int v = G[u][ptr[u]].to;

if (G[u][ptr[u]].cap - G[u][ptr[u]].flow > 0)

if (Dfs[v] != t && d[v] == d[u]+1)

if (int x = visit(v, min(Min, G[u][ptr[u]].cap - G[u][ptr[u]].flow), T)) {

G[u][ptr[u]].flow += x;

G[v][G[u][ptr[u]].index].flow -= x;

return x;

}

}

return 0;

}

void getFlow(int S, int T) {

while (bfs(S, T)) {

FOR(i,0,N-1) ptr[i] = 0;

while (1) {

t++;

int x = visit(S, oo, T);

if (!x) break;

flow += x;

}

}

}

};

1. TARJAN

const ll mod = 1e9+7;

const ll maxN = 1e6+5;

const ll N = 1e6;

ll n,m,u,v;

vector<ll> d[maxN];

ll child,cnt,root;

vector<ii> cau;

ll num[maxN],low[maxN],parent[maxN],AP[maxN];

void tarjan(ll u) {

num[u] = low[u] = ++cnt;

for (ll v : d[u]) {

if (num[v] == 0) {

parent[v] = u;

if (root == u) child++;

tarjan(v);

if (low[v] >= num[u]) AP[u] = 1;

if (low[v] > num[u]) cau.pb(ii(u,v));

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

} else {

if (parent[u] != v) low[u] = min(low[u], num[v]);

}

}

}

void solve() {

FOR(i,1,n) {

if (num[i] == 0) {

root = i;

child = 0;

tarjan(i);

if (child > 1) AP[i] = 1;

else AP[i] = 0;

}

}

ll dem = 0;

FOR(i,1,n) if (AP[i]) dem++;

cout<<dem<<" "<<cau.size();

}

int main() {

ios\_base::sync\_with\_stdio(0);

cin>>n>>m;

FOR(i,1,m) {

cin>>u>>v;

d[u].pb(v);

d[v].pb(u);

}

solve();

return 0;

}

1. CAY KHUNG NHO NHAT KRUSKAL + DISJOIN SET

int cha[maxN];

int \_find(int u) {

if(cha[u] == -1) return u;

return cha[u] = \_find(cha[u]);

}

bool \_join(int u, int v){

int pu = \_find(u);

int pv = \_find(v);

if(pu != pv) {

cha[pu] = pv;

return 1;

}

return 0;

}

struct Canh {

ll u,v,w;

Canh(ll u, ll v, ll w) : u(u), v(v), w(w) {

}

bool operator < (const Canh &o)const {

return w < o.w;

}

};

vector<Canh> kruskal(vector<Canh> v) {

vector<Canh> res;

FOR(i,1,n) cha[i] = -1;

sort(all(v));

for (auto x : v) {

if (\_join(x.u, x.v)) {

res.pb(x);

}

}

return res;

}

1. PST

int n**,** m**,** a**[**100010**],** b**[**100010**],** N**,** nNode**,** ver**[**100010**];**

map**<**int**,** int**>** ma**;**

struct node **{**

int lef**,** rig**,** cnt**;**

**}** t**[**2100010**];**

void build**(**int k**,** int l**,** int r**){**

**if** **(**l **<=** r**)** nNode **=** max**(**k**,** nNode**);**

**if** **(**l **>=** r**)** **return;**

t**[**k**].**lef **=** k**\***2**;**

t**[**k**].**rig **=** k**\***2**+**1**;**

int m **=** **(**l**+**r**)** **>>** 1**;**

build**(**t**[**k**].**lef**,**l**,**m**);**

build**(**t**[**k**].**rig**,**m**+**1**,**r**);**

**}**

int update**(**int oldId**,** int l**,** int r**,** int u**)** **{**

**if** **(**l **==** r**)** **{**

**++**nNode**;**

t**[**nNode**].**lef **=** t**[**nNode**].**rig **=** 0**;**

t**[**nNode**].**cnt **=** t**[**oldId**].**cnt**+**1**;**

**return** nNode**;**

**}**

int cur **=** **++**nNode**;**

int m **=** **(**l**+**r**)** **>>** 1**;**

**if** **(**u **<=** m**)** **{**

t**[**cur**].**lef **=** update**(**t**[**oldId**].**lef**,**l**,**m**,**u**);**

t**[**cur**].**rig **=** t**[**oldId**].**rig**;**

**}**

**else** **{**

t**[**cur**].**rig **=** update**(**t**[**oldId**].**rig**,**m**+**1**,**r**,**u**);**

t**[**cur**].**lef **=** t**[**oldId**].**lef**;**

**}**

t**[**cur**].**cnt **=** t**[**t**[**cur**].**rig**].**cnt **+** t**[**t**[**cur**].**lef**].**cnt**;**

**return** cur**;**

**}**

int get**(**int oldId**,** int newId**,** int l**,** int r**,** int k**)** **{**

**if** **(**l **==** r**)** **return** l**;**

int m **=** **(**l**+**r**)** **>>** 1**;**

int h **=** t**[**t**[**newId**].**lef**].**cnt **-** t**[**t**[**oldId**].**lef**].**cnt**;**

**if** **(**k **<=** h**)** **return** get**(**t**[**oldId**].**lef**,**t**[**newId**].**lef**,**l**,**m**,**k**);**

**return** get**(**t**[**oldId**].**rig**,**t**[**newId**].**rig**,**m**+**1**,**r**,**k**-**h**);**

**}**

int main**()** **{**

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

FOR**(**i**,**1**,**n**)** **{**

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

ma**[**a**[**i**]];**

**}**

**for** **(**map**<**int**,** int**>::**iterator it **=** ma**.**begin**();** it**!=**ma**.**end**();** it**++)** **{**

ma**[**it**->**X**]** **=** **++**N**;**

b**[**N**]** **=** it**->**X**;**

**}**

build**(**1**,**1**,**N**);**

ver**[**0**]** **=** 1**;**

FOR**(**i**,**1**,**n**)** ver**[**i**]** **=** update**(**ver**[**i**-**1**],**1**,**N**,**ma**[**a**[**i**]]);**

**while** **(**m**--)** **{**

int u**,** v**,** k**;**

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

printf**(**"%d\n"**,** b**[**get**(**ver**[**u**-**1**],** ver**[**v**],** 1**,** N**,** k**)]);**

**}**

**}**

1. IMPLICIT ST

struct seg **{**

ll b**,** l**,** r**;**

bool lz**;**

seg **\***L**,** **\***R**;**

seg**(){**b **=** l **=** r **=** lz **=** 0**;** L **=** R **=** **NULL;}**

seg**(**ll x**,** ll y**){**l **=** x**,** r **=** y**,** b **=** 0**,** lz **=** 0**;** L **=** R **=** **NULL;}**

inline void le**(){**

**if** **(**l **==** r **||** L **!=** **NULL)** **return;**

ll mid **=** **(**l**+**r**)/**2**;**

L **=** **new** seg**(**l**,** mid**);**

**}**

inline void ri**(){**

**if** **(**l **==** r **||** R **!=** **NULL)** **return;**

ll mid **=** **(**l**+**r**)/**2**;**

R **=** **new** seg**(**mid**+**1**,** r**);**

**}**

inline void color**(**bool h**)** **{**

**if** **(!**h**)** **return;**

lz **=** **!**lz**;**

b **=** r **-** l **+** 1 **-** b**;**

**}**

inline void shift**()** **{**

le**();**ri**();**

L **->** color**(**lz**);**

R **->** color**(**lz**);**

lz **=** **false;**

**}**

inline void upd**(**ll x**,** ll y**){**

**if** **(**y **<** l **||** r **<** x **||** x **>** y**)** **return;**

**if** **(**x **<=** l **&&** r **<=** y**)** **{**

color**(true);**

**return;**

**}**

shift**();**

L **->** upd**(**x**,** y**);**

R **->** upd**(**x**,** y**);**

b **=** L **->** b **+** R **->** b**;**

**}**

inline ll cnt**(**ll x**,** ll y**){**

**if** **(**y **<** l **||** r **<** x **||** x **>** y**)** **return** 0**;**

**if** **(**x **<=** l **&&** r **<=** y**)** **return** b**;**

shift**();**

**return** L **->** cnt**(**x**,** y**)** **+** R **->** cnt**(**x**,** y**);**

**}**

**};**

1. DIJKSTRA

const int oo=2000000000;

typedef pair<int,int> ii;

vector<ii> a[10001];

int d[10001];

int n,m,u,v,w,p,q,k;

bool x[10001],y[10001];

struct cmp{

bool operator()(ii a, ii b) {return(a.second>b.second);}

};

priority\_queue<ii, vector<ii>, cmp > xx;

void dijkstra(int k) {

int u,v,du,uv;

for (int i=1;i<=n;i++) d[i]=oo;

d[k]=0;

for (int i=1;i<=n;i++) a[i].push\_back(ii(0,0));

xx.push(ii(k,0));

while (xx.size()!=0)

{

u=xx.top().first;

du=xx.top().second;

xx.pop();

if (du!=d[u]) continue;

for (int i=0;v=a[u][i].first;i++)

{

if (k==1 && x[v]==true) continue;

if (k==n && y[v]==true) continue;

uv=a[u][i].second;

if (d[v]>du+uv)

{

d[v]=du+uv;

xx.push(ii(v,d[v]));

}

}

}

}

1. MO

bool cmp(data a, data b) {

if (a.f == b.f) return a.r < b.r;

return a.f < b.f;

}

void add**(**ll u**)** **{** res **+=** **(**1 **+** 2**\***cnt**[**a**[**u**]]++)\***a**[**u**];** **}**

void del**(**ll u**)** **{** res **-=(-**1 **+** 2**\***cnt**[**a**[**u**]]--)\***a**[**u**];** **}**

ll k **=** long**(**sqrt**(**n**));**

part **=** n **/** k **+** **(**n **%** k **>** 0**);**

FOR**(**i**,**1**,**q**)** **{**

scanf**(**"%I64d %I64d"**,** **&**b**[**i**].**l**,** **&**b**[**i**].**r**);**

b**[**i**].**cs **=** i**;**

b**[**i**].**f **=** b**[**i**].**l **/** part **-** **(**b**[**i**].**l**%**part**==** 0**);**

**}**

sort(b+1, b+q+1, cmp);

ll currL **=** 1**;**

ll currR **=** 1**;**

res **=** a**[**1**];**

cnt**[**a**[**1**]]++;**

FOR**(**i**,**1**,**q**)** **{**

**while** **(**currR **>** b**[**i**].**r**)** del**(**currR**--);**

**while** **(**currR **<** b**[**i**].**r**)** add**(++**currR**);**

**while** **(**currL **<** b**[**i**].**l**)** del**(**currL**++);**

**while** **(**currL **>** b**[**i**].**l**)** add**(--**currL**);**

p**[**b**[**i**].**cs**]** **=** res**;**

**}**

1. LCA

**if** **(**d**[**v**]** **<** d**[**u**])** FORE**(**i**,**17**,**0**)** **if** **(**d**[**parent**[**u**][**i**]]** **>=** d**[**v**])** u **=** parent**[**u**][**i**];**

**if** **(**d**[**u**]** **<** d**[**v**])** FORE**(**i**,**17**,**0**)** **if** **(**d**[**parent**[**v**][**i**]]** **>=** d**[**u**])** v **=** parent**[**v**][**i**];**

FORE**(**i**,**17**,**0**)** **if** **(**parent**[**u**][**i**]** **!=** parent**[**v**][**i**])** **{** u **=** parent**[**u**][**i**];** v **=** parent**[**v**][**i**];** **}**

**while** **(**u **!=** v**)** **{** u **=** parent**[**u**][**0**];** v **=** parent**[**v**][**0**];** **}**

1. MOD INVERSE

ll x,y,ucln;

void uc(ll a, ll b) {

if (b==0) {

ucln=a; x=1; y=0;

}

else {

uc(b,a%b);

ll tg=x; x=y;

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

}

}

ll mod(ll a,ll m) {

uc(a,m);

return (x%m+m)%m;

}

// Neu m la so nguyen to mod=a^(m-2)%m

1. MATRIX POWER

const ll M = 22;

ll sub(ll x, ll y) {

return (x-y+base) % base;

}

ll add(ll x, ll y) {

return (x+y)%base;

}

ll mul(ll x, ll y) {

return ((x%base) \* (y%base)) % base;

}

struct Matrix {

ll a[M][M];

Matrix() {

FOR(i,0,M-1) fill(a[i], a[i] + M, 0);

}

ll\* operator[](ll x) {

return a[x];

}

Matrix operator \* (Matrix &to) {

Matrix ans;

FOR(i,0,M-1)

FOR(j,0,M-1)

FOR(k,0,M-1) ans[i][j] = add(ans[i][j], mul(a[i][k], to[k][j]));

return ans;

}

vector<ll> operator \* (vector<ll> to) {

vector<ll> ans(22);

FOR(i,0,21)

FOR(j,0,21) ans[i] = add(ans[i], mul(a[i][j], to[j]));

return ans;

}

} ONE;

Matrix mu(Matrix A, ll n) {

Matrix B = ONE;

while (n) {

if (n&1) B = B\*A;

A = A\*A;

n >>= 1;

}

return B;

}

1. TRIE

struct Trie{

int val, child[30],lev;

Trie(){

val=lev=0;

memset(child, 0, sizeof(child));

}

} trie[M];

int nTrie=1;

void \_insert(string s){

int p=1;

foru(i,0,s.size()-1){

if(!trie[p].child[s[i]-'a']) trie[p].child[s[i]-'a'] = ++nTrie;

p=trie[p].child[s[i]-'a'];

}

trie[p].val++;

}

int \_find(string s){

int p=1;

foru(i,0,s.size()-1){

if(!trie[p].child[s[i]-'a']) return 0;

p=trie[p].child[s[i]-'a'];

}

return trie[p].val;

}

1. FFT

ll my\_round**(**double x**)** **{**

**if** **(**x **<** 0**)** **return** **-**my\_round**(-**x**);**

**return** **(**ll**)** **(**x **+** 1e-3**);**

**}**

const double PI **=** acos**((**double**)** **-**1.0**);**

const ll MN **=** 300000**;**

**typedef** complex**<**ld**>** cplex**;**

ll rev**[**MN**],** a**[**MN**],** b**[**MN**];**

ll d**[**MN**];**

cplex wlen\_pw**[**MN**],** fa**[**MN**],** fb**[**MN**];**

void fft**(**cplex a**[],** ll n**,** bool invert**)** **{**

FOR**(**i**,**0**,**n**-**1**)** **if** **(**i **<** rev**[**i**])** swap **(**a**[**i**],** a**[**rev**[**i**]]);**

**for** **(**ll len **=** 2**;** len **<=** n**;** len **<<=** 1LL**)** **{**

long double alpha **=** 2 **\*** PI **/** len **\*** **(**invert **?** **-**1 **:** **+**1**);**

ll len2 **=** len **>>** 1LL**;**

wlen\_pw**[**0**]** **=** cplex**(**1**,** 0**);**

cplex wlen**(**cos**(**alpha**),** sin**(**alpha**));**

**for** **(**ll i **=** 1**;** i **<** len2**;** **++**i**)** wlen\_pw**[**i**]** **=** wlen\_pw**[**i**-**1**]** **\*** wlen**;**

**for** **(**ll i **=** 0**;** i **<** n**;** i **+=** len**)** **{**

cplex t**,** **\***pu **=** a**+**i**,** **\***pv **=** a **+** i **+** len2**,**

**\***pu\_end **=** a **+** i **+** len2**,** **\***pw **=** wlen\_pw**;**

**for** **(;** pu **!=** pu\_end**;** **++**pu**,** **++**pv**,** **++**pw**)** **{**

t **=** **\***pv **\*** **\***pw**;**

**\***pv **=** **\***pu **-** t**;**

**\***pu **+=** t**;**

**}**

**}**

**}**

**if** **(**invert**)** FOR**(**i**,**0**,**n**-**1**)** a**[**i**]** **/=** n**;**

**}**

void calcRev**(**ll n**,** ll logn**)** **{**

FOR**(**i**,**0**,**n**-**1**)** **{**

rev**[**i**]** **=** 0**;**

FOR**(**j**,**0**,**logn**-**1**)** **if** **(**i **&** **(**1LL **<<** j**))** rev**[**i**]** **|=** 1LL **<<** **(**logn **-** 1 **-** j**);**

**}}**

void mulpoly**(**ll a**[],** ll b**[],** ll c**[],** ll na**,** ll nb**,** ll **&**n**)** **{**

ll l **=** max**(**na**,** nb**),** logn **=** 0**;**

**for** **(**n **=** 1**;** n **<** l**;** n **<<=** 1LL**)** **++**logn**;**

n **<<=** 1LL**;** **++**logn**;**

calcRev**(**n**,** logn**);**

FOR**(**i**,**0**,**n**-**1**)** fa**[**i**]** **=** fb**[**i**]** **=** cplex**(**0**);**

FOR**(**i**,**0**,**na**-**1**)** fa**[**i**]** **=** cplex**(**a**[**i**]);**

FOR**(**i**,**0**,**nb**-**1**)** fb**[**i**]** **=** cplex**(**b**[**i**]);**

fft**(**fa**,** n**,** **false);**

fft**(**fb**,** n**,** **false);**

FOR**(**i**,**0**,**n**-**1**)** fa**[**i**]** **\*=** fb**[**i**];**

fft**(**fa**,** n**,** **true);**

FOR**(**i**,**0**,**n**-**1**)** {

c**[**i**]** **=** **(**ll**)(**fa**[**i**].**real**()** **+** 0.5**);**

**if (c[i] < 0) c[i]--;**

**}**

**}**

int main**()** **{**

ll n**;**

ios**::**sync\_with\_stdio**(**0**);**

freopen**(**"inp.txt"**,** "r"**,** stdin**);**

cin **>>** n**;**

FOR**(**i**,**1**,**n**)** **{**

ll k**;**

cin **>>** k**;**

a**[**k**+**50000**]++;**

**}**

FOR**(**i**,**1**,**n**)** **{**

ll k**;**

cin **>>** k**;**

b**[**k**+**50000**]++;**

**}**

ll m **=** n**;**

mulpoly**(**a**,**b**,**d**,**100001**,**100001**,**n**);**

ll res **=** 0**;**

FOR**(**i**,**1**,**m**)** **{**

ll k**;**

cin **>>** k**;**

res **+=** d**[**k**+**100000**];**

**}**

cout **<<** res**;**

**return** 0**;**

**}**

1. GEOMETRY

#include <bits/stdc++.h>

#define ll long long

#define ld long double

#define X first

#define Y second

#define pb push\_back

#define EPS 1e-15

#define all(a) (a).begin(), (a).end()

#define sz(a) int((a).size())

#define ms(s, n) memset(s, n, sizeof(s))

#define FOR(i,a,b) for (int i = (a); i <= (b); i++)

#define FORE(i,a,b) for (int i = (a); i >= (b); i--)

#define sqr(a) (a)\*(a)

#define foru(i,a,b) for (ll i = (a); i <= (b); ++i)

#define ford(i,a,b) for (ll i = (a); i >= (b); --i)

#define cbit(a) \_\_builtin\_popcount(a)

#define uni(a) (a).resize(unique(all(a)) - (a).begin())

using namespace std;

typedef pair<ll, ll> II;

typedef pair<int, int> ii;

const int M=1e6+7;

const int MM=1e7+10;

const ll MOD=1e9+7;

const ll oo=1e9;

const ll MAX=1e18+10;

const ld PI = acos(-1);

inline int cmp(ld a, ld b) {

return (a < b - EPS) ? -1 : ((a > b + EPS) ? 1 : 0);

}

ld DEG\_to\_RAD(ld d) { return d \* PI / 180.0; }

ld RAD\_to\_DEG(ld r) { return r \* 180.0 / PI; }

// Point section ---------------------------------------------------------

struct point {

ld x, y;

point(ld x = 0, ld y = 0) : x(x), y(y) {}

point operator+ (const point &a) const { return point(x+a.x, y+a.y); }

point operator- (const point &a) const { return point(x-a.x, y-a.y); }

point operator \* (ld k) const { return point(x\*k, y\*k); }

point operator / (ld k) const { return point(x/k, y/k); }

bool operator== (point &p) {

return (fabs(x - p.x) < EPS && fabs(y - p.y) < EPS);

}

bool operator< (const point &p) const {

if (fabs(x - p.x) > EPS) return x < p.x;

return y < p.y;

}

point rotate(ld alpha) {

return point(x \* cos(alpha) - y \* sin(alpha), x \* sin(alpha) + y \* cos(alpha));

}

void read() {

cin >> x >> y;

}

void write() {

cout << fixed << setprecision(6) << x << " " << y << endl;

}

};

ld dist(point a, point b) { return sqrt((a.x - b.x) \* (a.x - b.x) + (a.y - b.y) \* (a.y - b.y)); }

// end of Point section --------------------------------------------------

// Line section ----------------------------------------------------------

struct line {

ld a, b, c;

point A, B;

line(ld a = 0, ld b = 0, ld c = 0) : a(a), b(b), c(c) {}

line(point A, point B) : A(A), B(B) {

a = B.y - A.y;

b = A.x - B.x;

c = -(a \* A.x + b \* A.y);

}

line(point P, ld m) {

a = -m; b = 1;

c = -((a \* P.x) + (b \* P.y));

}

ld f(point A) {

return a\*A.x + b\*A.y + c;

}

bool areIn(point p) {

if(abs(f(p)) < EPS) return true;

return false;

}

};

bool areParallel(line l1, line l2) { return cmp(l1.a\*l2.b, l1.b\*l2.a) == 0; }

bool areIntersect(line l1, line l2, point &p) {

if (areParallel(l1, l2)) return false;

ld dx = l1.b\*l2.c - l2.b\*l1.c;

ld dy = l1.c\*l2.a - l2.c\*l1.a;

ld d = l1.a\*l2.b - l2.a\*l1.b;

p = point(dx/d, dy/d);

return true;

}

// end of Line section ---------------------------------------------------

// Vector section --------------------------------------------------------

struct vecto {

ld x, y;

vecto(ld x = 0, ld y = 0) : x(x), y(y) {}

bool operator== (vecto &v) {

return (fabs(x - v.x) < EPS && fabs(y - v.y) < EPS);

}

bool operator< (const vecto &v) const {

if (fabs(x - v.x) > EPS) return x < v.x;

return y < v.y;

}

};

ld dist(vecto a) { return sqrt(a.x \* a.x + a.y \* a.y); }

vecto toVec(point a, point b) { return vecto(b.x - a.x, b.y - a.y); }

vecto scale(vecto v, ld s) { return vecto(v.x \* s, v.y \* s); }

point translate(point p, vecto v) { return point(p.x + v.x, p.y + v.y); }

ld dot(vecto a, vecto b) { return a.x\*b.x + a.y\*b.y; }

ld dot(point a, point b, point c) { return dot(toVec(a, b), toVec(a, c)); }

ld cross(vecto a, vecto b) { return a.x\*b.y - a.y\*b.x; }

ld cross2(point a, point b) { return (a.x - b.x) \* (a.y + b.y); }

ld cross(point a, point b, point c) { return cross(toVec(a, b), toVec(a, c)); }

ld norm\_sq(vecto v) { return v.x \* v.x + v.y \* v.y; }

ld distToLine(point p, point a, point b, point &c) {

vecto ap = toVec(a, p);

vecto ab = toVec(a, b);

ld u = dot(ap, ab) / norm\_sq(ab);

c = translate(a, scale(ab, u));

return dist(p, c);

}

ld distToLineSegment(point p, point a, point b, point &c) {

vecto ap = toVec(a, p);

vecto ab = toVec(a, b);

ld u = dot(ap, ab) / norm\_sq(ab);

if (u < 0) {

c = point(a.x, a.y);

return dist(p, a);

}

if (u > 1) {

c = point(b.x, b.y);

return dist(p, b);

}

return distToLine(p, a, b, c);

}

ld angle(point a, point o, point b) {

// angle aob in rad

vecto oa = toVec(o, a), ob = toVec(o, b);

return acos(dot(oa, ob) / sqrt(norm\_sq(oa) \* norm\_sq(ob)));

}

bool ccw(point p, point q, point r) { return cross(toVec(p, q), toVec(p, r)) >= 0; }

bool collinear(point p, point q, point r) { return fabs(cross(toVec(p, q), toVec(p, r))) < EPS; }

point intersection(point a, point b, point c, point d) {

b = b - a; d = c - d; c = c - a;

return a + b \* cross(c, d) / cross(b, d);

}

// end of Vector section -------------------------------------------------

struct segment {

point \_begin, \_end;

line \_line;

segment () {}

segment (point \_begin, point \_end): \_begin(\_begin), \_end(\_end) { \_line = line(\_begin, \_end);}

bool areIn(point A) {

if((A == \_begin || A == \_end)) return 1;

if(!\_line.areIn(A)) return 0;

if(A.x > max(\_begin.x, \_end.x) || A.x < min(\_begin.x, \_end.x)) return 0;

if(A.y > max(\_begin.y, \_end.y) || A.y < min(\_begin.y, \_end.y)) return 0;

return 1;

}

};

bool areIntersect(segment a, segment b, point &p) {

if(!areIntersect(a.\_line, b.\_line, p)) return 0;

if(a.areIn(p) && b.areIn(p)) return 1;

return 0;

}

// Circle section --------------------------------------------------------

struct circle {

point center;

ld r;

circle() {} // @suppress("Class members should be properly initialized")

circle(ld x, ld y, ld r) {

center.x = x;

center.y = y;

this->r = r;

}

void read() {

cin >> center.x >> center.y >> r;

}

};

bool areIntersect(circle u, circle v) {

if (cmp(sqrt(dist(u.center, v.center)), u.r + v.r) > 0) return false;

if (cmp(sqrt(dist(u.center, v.center)) + v.r, u.r) < 0) return false;

if (cmp(sqrt(dist(u.center, v.center)) + u.r, v.r) < 0) return false;

return true;

}

vector<point> intersection(line l, circle cir) {

ld r = cir.r, a = l.a, b = l.b, c = l.c + l.a\*cir.center.x + l.b\*cir.center.y;

vector<point> res;

ld x0 = -a\*c/(a\*a+b\*b), y0 = -b\*c/(a\*a+b\*b);

if (c\*c > r\*r\*(a\*a+b\*b)+EPS) return res;

else if (fabs(c\*c - r\*r\*(a\*a+b\*b)) < EPS) {

res.push\_back(point(x0, y0) + point(cir.center.x, cir.center.y));

return res;

}

else {

ld d = r\*r - c\*c/(a\*a+b\*b);

ld mult = sqrt (d / (a\*a+b\*b));

ld ax,ay,bx,by;

ax = x0 + b \* mult;

bx = x0 - b \* mult;

ay = y0 - a \* mult;

by = y0 + a \* mult;

res.push\_back(point(ax, ay) + point(cir.center.x, cir.center.y));

res.push\_back(point(bx, by) + point(cir.center.x, cir.center.y));

return res;

}

}

vector<point> circleIntersect(circle u, circle v) {

vector<point> res;

if (!areIntersect(u, v)) return res;

ld d = dist(u.center, v.center);

ld alpha = acos((u.r \* u.r + d\*d - v.r \* v.r) / 2.0 / u.r / d);

point p1 = (v.center - u.center).rotate(alpha);

point p2 = (v.center - u.center).rotate(-alpha);

res.push\_back(p1 / dist(p1, point(0,0)) \* u.r + u.center);

res.push\_back(p2 / dist(p1, point(0,0)) \* u.r + u.center);

return res;

}

// end of Circle section -------------------------------------------------

// Polygon section -------------------------------------------------------

ld area(vector<point> &a)

{

ld res = 0;

FOR(i,0,sz(a)-1) {

point x = a[i], y = a[(i+1) % sz(a)];

res += x.x \* y.y - x.y \* y.x;

}

return res;

}

bool leftLower(point a, point b)

{

if (a.x == b.x) return a.y < b.y;

return a.x < b.x;

}

vector<point> ch, upper, lower;

point origin;

bool smallerAngle(point first, point second)

{

first = first - origin;

second = second - origin;

if (cmp(first.y, 0) == 0 && cmp(second.y, 0) == 0 && cmp(first.x \* second.x, 0) <= 0) return first.x > second.x;

if (cmp(first.y, 0) == 0 && cmp(first.x, 0) >= 0 && cmp(second.y, 0) != 0) return true;

if (cmp(second.y, 0) == 0 && cmp(second.x, 0) >= 0 && cmp(first.y, 0) != 0) return false;

if (cmp(first.y \* second.y, 0) < 0) return first.y > second.y;

ld cp = cross(first, second);

return cmp(cp, 0) > 0 || (cmp(cp, 0) == 0 && cmp(abs(first.x) - abs(second.x), 0) < 0) || (cmp(cp, 0) == 0 && cmp(abs(first.x) - abs(second.x), 0) == 0 && cmp(abs(first.y) - abs(second.y), 0) < 0);

}

void convexHull(vector<point> &a)

{

// return clockwised convex hull

ch.clear();

upper.clear();

lower.clear();

if (sz(a) <= 3) {

FOR(i,0,sz(a)-1) ch.pb(a[i]);

if (sz(ch) == 3) {

if (cross(ch[0], ch[1], ch[2]) > 0) reverse(all(ch));

else if (cross(ch[0], ch[1], ch[2]) == 0) ch.pop\_back();

}

return;

}

sort(all(a), leftLower);

upper.pb(a[0]);

upper.pb(a[1]);

FOR(i,2,sz(a)-1) {

upper.pb(a[i]);

while (sz(upper) >= 3 && cross(upper[sz(upper)-3], upper[sz(upper)-2], upper[sz(upper)-1]) >= 0) upper.erase(upper.end()-2);

}

FOR(i,0,sz(upper)-1) ch.pb(upper[i]);

lower.pb(a[sz(a)-1]);

lower.pb(a[sz(a)-2]);

FORE(i,sz(a)-3,0) {

lower.pb(a[i]);

while (sz(lower) >= 3 && cross(lower[sz(lower)-3], lower[sz(lower)-2], lower[sz(lower)-1]) >= 0) lower.erase(lower.end()-2);

}

FOR(i,1,sz(lower)-2) ch.pb(lower[i]);

}

// the convex hull should be sorted with origin point first sort(all(ch), smallerAngle);

// origin is a point that lies inside the convex hull, for example centroid of 3 points on the convex hull

bool isInside(vector<point> &a, point p)

{

int dau = 0, cuoi = sz(a)-1, mid;

while (dau <= cuoi) {

mid = (dau + cuoi) >> 1;

if (!smallerAngle(p, a[mid])) dau = mid+1;

else cuoi = mid-1;

}

if (cuoi < 0) cuoi = sz(a)-1;

int now = cuoi;

int nex = (now + 1) % sz(a);

if (ccw(a[now], a[nex], p)) return true;

return false;

}

bool inPolygon(point pt, vector<point> &p)

{

if (sz(p) < 3) return false;

ld sum = 0;

FOR(i,0,sz(p)-1) {

if (ccw(pt, p[i], p[(i+1)%sz(p)])) sum += angle(p[i], pt, p[i+1]);

else sum -= angle(p[i], pt, p[i+1]);

}

return fabs(fabs(sum) - 2\*PI) < EPS;

}

vector<point> polygon\_cut(vector<point> P, line l){

vector<point> Q;

FOR(i,0,sz(P)-1) {

point A = P[i], B = P[(i+1)%sz(P)];

if (cross(toVec(l.A, l.B), toVec(l.A, A)) >= 0) Q.push\_back(A);

if (cross(toVec(l.A, l.B), toVec(l.A, A)) \* cross(toVec(l.A, l.B), toVec(l.A, B)) < 0) {

point p;

areIntersect(line(A, B), l, p);

Q.push\_back(p);

}

}

return Q;

}

// end of Polygon section ------------------------------------------------

point center\_from(double bx, double by, double cx, double cy) {

double B=bx\*bx+by\*by, C=cx\*cx+cy\*cy, D=bx\*cy-by\*cx;

return point((cy\*B-by\*C)/(2\*D), (bx\*C-cx\*B)/(2\*D));

}

circle circle\_from(point A, point B, point C) {

point I = center\_from(B.X-A.X, B.Y-A.Y, C.X-A.X, C.Y-A.Y);

return circle(I+A, abs(I));

}

1. EDGE ARRAY

E **=** 0**;** memset**(**last**,** **-**1**,** **sizeof(**last**));**

void addEdge**(**int x**,** int y**)** **{** adj**[**E**]** **=** y**;** next**[**E**]** **=** last**[**x**];** last**[**x**]** **=** E**++;** **}**

1. HASH

ll getHashT**(**int i**,**int j**)** **{** **return** **(**hashT**[**j**]-**hashT**[**i**-**1**]\***POW**[**j**-**i**+**1**]+**base**\***base**)%**base**;** **}**

int main**()** **{**

string T**,**P**;**

cin **>>** T **>>** P**;**

int m**=**T**.**size**(),**n**=**P**.**size**();**

T**=**" "**+**T**;**P**=**" "**+**P**;**

POW**[**0**]=**1**;**

FOR**(**i**,**1**,**m**)** POW**[**i**]=(**POW**[**i**-**1**]\***26**)** **%** base**;**

FOR**(**i**,**1**,**m**)** hashT**[**i**]=(**hashT**[**i**-**1**]\***26**+**T**[**i**]-**'a'**)** **%** base**;**

ll hashP**=**0**;**

FOR**(**i**,**1**,**n**)** hashP**=(**hashP**\***26**+**P**[**i**]-**'a'**)** **%** base**;**

FOR**(**i**,**1**,**m**-**n**+**1**)** **if(**hashP**==**getHashT**(**i**,**i**+**n**-**1**))** printf**(**"%d "**,**i**);**

**}**

1. KMP

static const int MAXC = 26;

static const char FIRST\_CHAR = 'a';

void calcPrefix() {

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

int k = prefix[i - 1];

while (k && s[i] != s[k]) k = prefix[k - 1];

if (s[i] == s[k]) k++;

prefix[i] = k;

}

}

void calcNextState() { // nextState[i][c - 'a']

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

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

char x = j + FIRST\_CHAR;

if (i == n || (i && x != s[i])) {

nextState[i][j] = nextState[prefix[i - 1]][j];

}

else nextState[i][j] = i + (x == s[i]);

}

}

}

1. SUFFIX ARRAY AND LCP

int t, SA[20010], RA[20010], c[20010], tempRA[20010], tempSA[20010], n, behind[20010], LCP[20010], PLCP[20010];

string s;

void countingSort(int k) {

int sum = 0, maxi = max(300, n);

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

FOR(i,0,n-1)

if (i+k < n) c[RA[i+k]]++;

else c[0]++;

FOR(i,0,maxi-1) {

int t = c[i];

c[i] = sum;

sum += t;

}

FOR(i,0,n-1) {

int a = 0;

if (SA[i]+k < n) a = RA[SA[i]+k];

tempSA[c[a]++] = SA[i];

}

FOR(i,0,n-1) SA[i] = tempSA[i];

}

void suffixArray() {

int r;

FOR(i,0,n-1) SA[i] = i;

FOR(i,0,n-1) RA[i] = s[i];

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

countingSort(k);

countingSort(0);

tempRA[SA[0]] = r = 0;

FOR(i,1,n-1) {

if (RA[SA[i]] != RA[SA[i-1]]) {

tempRA[SA[i]] = ++r;

continue;

}

int a, b;

if (SA[i]+k >= n) a = 0;

else a = RA[SA[i]+k];

if (SA[i-1]+k >= n) b = 0;

else b = RA[SA[i-1]+k];

if (a == b) tempRA[SA[i]] = r;

else tempRA[SA[i]] = ++r;

}

FOR(i,0,n-1) RA[i] = tempRA[i];

if (r == n-1) break;

}

}

void buildLCP() {

behind[SA[0]] = -1;

FOR(i,1,n-1) behind[SA[i]] = SA[i-1];

int L = 0;

FOR(i,0,n-1) {

if (behind[i] == -1) {

PLCP[i] = 0;

continue;

}

while (s[i+L] == s[behind[i]+L]) L++;

PLCP[i] = L;

L = max(L-1,0);

}

FOR(i,0,n-1) LCP[i] = PLCP[SA[i]];

}

1. MANACHER

const int maxN **=** int**(**1e5**)+**111**;**

char S**[**2**\***maxN**];**

int F**[**2**\***maxN**];**

int Length\_Of\_Longest\_Palindrome**(**char S**[],** int N**)** **{**

int i**,** L**,** R **=** L **=** **-**1**,** M**,** d **=** 0**,** res **=** 0**;**

M **=** 2**\***N**+**1**;**

**for(**i **=** M**-**1**;** i **>=** 0**;** i**--)** **{**

**if** **(**d**)** S**[**i**]** **=** S**[(**i**-**1**)** **>>** 1**];**

**else** S**[**i**]** **=** '#'**;**

d **=** 1**-**d**;**

**}**

**for(**i **=** 0**;** i **<=** M**-**1**;** i**++)** **{**

**if** **(**i **<=** R**)** d **=** min**(**F**[**L**+**R**-**i**],** R**-**i**);**

**else** d **=** 0**;**

**while** **((**i**-**d**-**1 **>=** 0**)** **&&** **(**i**+**d**+**1 **<** M**)** **&&** **(**S**[**i**-**d**-**1**]** **==** S**[**i**+**d**+**1**]))** d**++;**

F**[**i**]** **=** d**;**

**if** **(**i**+**d **>** R**)** L **=** i**-**d**,** R **=** i**+**d**;**

res **=** max**(**res**,** d**);**

**}**

**return** res**;**

**}**

1. Z-FUNCTION

string s, t;

int z[2000010];

void ZFunction() {

int L = 0, R = 0;

z[0] = s.length();

FOR(i,1,s.length()-1)

if (i > R) {

L = R = i;

while (R < s.length() && s[R] == s[R-L]) R++;

z[i] = R-L; R--;

}

else {

int k = i-L;

if (z[k] < R-i+1) z[i] = z[k];

else {

L = i;

while (R < s.length() && s[R] == s[R-L]) R++;

z[i] = R-L; R--;

}

}

}

1. Aho Corasick

struct Vertex {

static const int ALPHABET\_SIZE = 26;

int child[ALPHABET\_SIZE], go[ALPHABET\_SIZE];

int leaf = false, par = -1, link = -1, nextLeaf = -1;

char parChar; // edge par -> cur

Vertex(int par = -1, char ch = '$') : par(par), parChar(ch) {

memset(child, -1, sizeof(child)); memset(go, -1, sizeof(go));

}

};

struct Trie {

static const char ALPHA = 'a';

vector<Vertex> node;

Trie() { node.push\_back({0}); }

int add(string s) {

int cur = 0; // root = 0

for (auto ch: s) {

int c = ch - ALPHA;

if (node[cur].child[c] == -1) {

node[cur].child[c] = node.size();

node.push\_back({cur, ch});

}

cur = node[cur].child[c];

}

node[cur].leaf = true;

return cur;

}

int go(int cur, char ch) { // nextState[i][c]

int c = ch - ALPHA;

if (node[cur].go[c] == -1) {

if (node[cur].child[c] != -1) {

node[cur].go[c] = node[cur].child[c];

}

else {

node[cur].go[c] = (cur == 0) ? 0 : go(getLink(cur), ch);

}

}

return node[cur].go[c];

}

int getLink(int cur) {

if (node[cur].link == -1) {

if (!cur || !node[cur].par) node[cur].link = 0;

else {

node[cur].link = go(getLink(node[cur].par), node[cur].parChar);

}

}

return node[cur].link;

}

int getNextLeaf(int cur) {

if (cur == 0) return 0;

if (node[cur].nextLeaf != -1) return node[cur].nextLeaf;

int nxt = getLink(cur);

if (node[nxt].leaf.size()) return node[cur].nextLeaf = nxt;

return node[cur].nextLeaf = getNextLeaf(nxt);

}

1. TÌM CHỮ SỐ THỨ K TRONG DÃY LIÊN TỤC + XÓA CHẴN LẺ

ll d[20], T,t,n;

ll lastNumber(ll n){ // Tim chu so ton tai cuoi cung sau khi sao chan, le

ll a = 1;

ll b = 2;

ll t = 1;

ll kt = 1;

while (n > 1) {

if (kt%2==1) {

t = t\*2;

a = b+t;

swap(a,b);

} else {

t = t\*2;

b = a+t;

}

if (kt%2 == 1) {

if (n%2==1) {

n = n/2;

kt++;

} else {

kt++;

n = n/2;

}

} else {

if (n%2==1) {

n=(n+1)/2;

kt++;

} else {

kt++;

n = n/2;

}

}

}

return a;

}

ll bac(ll n) { // Tim so chu so

ll res = 0;

while (n > 0) {

res ++;

n/=10;

}

return res;

}

ll mu(ll n) { // Mu 10

ll res = 1;

FOR(i,1,n) res\*=10;

return res;

}

ll cntlen(ll n) { // Dem do dai 123..101112

ll res = 0;

ll b = bac(n);

FOR(i,1,b-1) res += d[i];

res += (n-mu(b-1)+1)\*b;

return res;

}

ll tim(ll k) { // Tim so thu k trong 123..101112

int i=1;

while (k>d[i]){

k-=d[i];

i++;

}

ll tt = k/i;

if (k%i != 0) tt++;

ll vt = k%i;

if (vt == 0) vt = i;

vt = i-vt+1;

ll so;

if (i == 1) so = tt;

else so = mu(i-1)+tt-1;

ll res=-1;

while (vt > 0) {

res = so%10;

so /= 10;

vt--;

}

return res;

}

int main() {

d[1] = 9;

FOR(i,2,17) {

d[i] = d[i-1]\*10;

}

FOR(i,2,17) d[i] \*= i;

cin>>T;

while (T--) {

cin>>n;

ll t = lastNumber(cntlen(n));

cout<<tim(t);

}

}

1. LAZY SEGMENT TREE (IT)

void lazydown(ll id) {

ll tmp = t[id].Y;

t[id\*2].X += tmp;

t[id\*2].Y += tmp;

t[id\*2+1].X += tmp;

t[id\*2+1].Y += tmp;

t[id].Y = 0;

}

void update(ll id, ll l, ll r, ll u, ll v, ll val) {

if (l > v || r < u) return;

if (l >= u && r <= v) {

t[id].X += val;

t[id].Y += val;

return;

}

lazydown(id);

ll mid = (l+r)/2;

update(id\*2,l,mid,u,v,val);

update(id\*2+1,mid+1,r,u,v,val);

t[id].X = max(t[id\*2].X, t[id\*2+1].X);

ll get(ll id, ll l, ll r, ll u, ll v) {

if (l > v || r < u) return -1e18;

if (l>=u && r <= v) return t[id].X;

lazydown(id);

ll mid = (l+r)/2;

return max(get(id\*2,l,mid,u,v), get(id\*2+1,mid+1,r,u,v));

}

1. POLLARD’S RHO

#define abs\_val(a) (((a)>0)?(a):-(a))

typedef long long ll;

ll mulmod(ll a, ll b, ll c) // returns (a \* b) % c, and minimize overflow

{

llx=0,y=a%c;

while (b > 0)

{

if (b % 2 == 1) x = (x + y) % c;

y=(y\*2)%c;

b/=2;

}

return x % c;

}

ll gcd(ll a,ll b) {

return !b ? a : gcd(b, a % b); // standard gcd

}

ll pollard\_rho(ll n) {

int i = 0, k = 2;

ll x = 3, y = 3; // random seed = 3, other values possible

while (1) {

i++;

x = (mulmod(x, x, n) + n - 1) % n; // generating function

ll d = gcd(abs\_val(y - x), n); // the key insight

if (d != 1 && d != n) return d; // found one non-trivial factor

if (i == k) y = x, k \*= 2;

}

}

int main() {

ll n = 2063512844981574047LL; // we assume that n is not a large prime

ll ans = pollard\_rho(n); // break n into two non trivial factors

if (ans > n / ans) ans = n / ans; // make ans the smaller factor

printf("%lld %lld\n", ans, n / ans); // should be: 1112041493 1855607779

} // return 0;

1. PHI

ll EulerPhi(ll N) {

ll PF\_idx = 0, PF = primes[PF\_idx], ans = N; // start from ans = N

while (PF \* PF <= N) {

if (N % PF == 0) ans -= ans / PF; // only count unique factor

while (N % PF == 0) N /= PF;

PF = primes[++PF\_idx];

}

if (N != 1) ans -= ans / N; // last factor

return ans;

}

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

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

if (phi[i] == i)

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

phi[j] -= phi[j]/i;

////Mobius

mu[1] = 1;

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

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

mu[j] -= mu[i];

1. FLOYD’S CYCLE FINDING

ii floydCycleFinding(int x0) // function int f(int x) is defined earlier

{

// 1st part: finding k\*mu, hare’s speed is 2x tortoise’s

int tortoise = f(x0), hare = f(f(x0)); // f(x0) is the node next to x0

while (tortoise != hare)

{

tortoise = f(tortoise);

hare = f(f(hare));

}

// 2nd part: finding mu, hare and tortoise move at the same speed

int mu = 0;

hare = x0;

while (tortoise != hare)

{

tortoise = f(tortoise);

hare = f(hare);

mu++;

}

// 3rd part: finding lambda, hare moves, tortoise stays

int lambda = 1;

hare = f(tortoise);

while (tortoise != hare)

{

hare = f(hare);

lambda++;

}

return ii(mu, lambda);

}

1. GAUSSIAN ELIMINATION

#define MAX\_N 100 // adjust this value as needed

struct AugmentedMatrix {

double mat[MAX\_N][MAX\_N + 1];

};

struct ColumnVector {

double vec[MAX\_N];

};

ColumnVector GaussianElimination(int N, AugmentedMatrix Aug) {

// input: N, Augmented Matrix Aug, output: Column vector X, the answer

int i, j, k, l;

double t;

ColumnVector X;

for (j = 0; j < N - 1; j++) // the forward elimination phase

{

l=j;

for (i = j + 1; i < N; i++) // which row has largest column value

if (fabs(Aug.mat[i][j]) > fabs(Aug.mat[l][j]))

l = i; // remember this row l

// swap this pivot row, reason: to minimize floating point error

for (k = j; k <= N; k++) // t is a temporary double variable

t = Aug.mat[j][k], Aug.mat[j][k] = Aug.mat[l][k], Aug.mat[l][k] = t;

for (i = j + 1; i < N; i++) // the actual forward elimination phase

for (k = N; k >= j; k--)

Aug.mat[i][k] -= Aug.mat[j][k] \* Aug.mat[i][j] / Aug.mat[j][j];

}

for (j = N - 1; j >= 0; j--) // the back substitution phase

{

for (t = 0.0, k = j + 1; k < N; k++) t += Aug.mat[j][k] \* X.vec[k];

X.vec[j] = (Aug.mat[j][N] - t) / Aug.mat[j][j]; // the answer is here

}

return X;

}

1. ARRAY’S MAXIMUM XOR

ull check\_bit(ull N,int POS)

{

return (N & (1ULL<<POS));

}

vector<ull>v;

ull gaussian\_elimination()

{

int n=v.size();

int ind=0; // Array index

for(int bit=log2(v[0]); bit>=0; bit--)

{

int x=ind;

while(x<n&&check\_bit(v[x],bit)==0) x++;

if(x==n) continue; // skip if there is no number below ind where current bit is 1

swap(v[ind],v[x]);

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

if(j!=ind&&check\_bit(v[j],bit)) v[j]^=v[ind];

ind++;

}

ull ans=v[0];

for(int i=1; i<n; i++) ans=max(ans,ans^v[i]);

return ans;

}

int main() {

int i,j,k,l,m,n,t,kase=1;

scanf("%d",&n);

ull x;

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

{

cin>>x;

v.push\_back(x);

}

sort(v.rbegin(),v.rend());

cout<<gaussian\_elimination()<<"\n";

return 0;

}

1. EULERIAN GRAPH

// An Euler path is defined as a path in a graph which visits each edge of the graph exactly once.

// An Euler tour/cycle is an Euler path which starts and ends on the same vertex.

// A graph which has either an Euler path or an Euler tour is called an Eulerian Graph.

// Eulerian Graph check: all its vertices have even degrees

// Euler path check:all except 2 vertices have even degrees

list<int> cyc;

void EulerTour(list<int>::iterator i, int u) {

for (int j = 0; j < (int)AdjList[u].size(); j++) {

ii v = AdjList[u][j];

if (v.second) // if this edge can still be used/not removed {

v.second = 0; // make the weight of this edge to be 0 (‘removed’)

for (int k = 0; k < (int)AdjList[v.X].size(); k++) {

ii uu = AdjList[v.first][k]; // remove bi-directional edge

if (uu.first == u && uu.second) {

uu.second = 0;

break;

}

}

EulerTour(cyc.insert(i, u), v.first);

}

}

}

// inside int main()

cyc.clear();

EulerTour(cyc.begin(), A); // cyc contains an Euler tour starting at A

for (list<int>::iterator it = cyc.begin(); it != cyc.end(); it++) printf("%d\n", \*it);

1. BINARY SUBSEQUENCE SUM

// can tinh mang dp[0...2^n-1] voi dp[s] = sum(A[T] | T la mot tap con cua S)

// khoi tao dp = A

FOR(i,0,n-1)

FOR(mask,1,(1<<n)-1)

if (mask>>i&1) dp[mask] += dp[mask ^ (1<<i)];

1. CONVEX HULL TRICK

// dp[i] = min(dp[j] + b[j] \* a[i] + C)

// line: y = x \* a + b

// dp[i] = a[i] \* b[j] + dp[j] + C

// addLine(a, b) => addLine(b[i], dp[i]) (after calc dp[i])

// y = getMin(x) => dp[i] = getMin(a[i]) + C

// return max

// add (-a, b), return -query if need min

struct Line {

mutable long long k, m, p;

bool operator < (const Line& o) const {return k < o.k;}

bool operator < (long long x) const {return p < x;}

};

struct LineContainer : multiset<Line, less<>> {

// (for doubles, use inf = 1/.0, div(a,b) = a/b)

const long long inf = LLONG\_MAX;

long long div(long long a, long long b) { // floored division

return a / b - ((a ^ b) < 0 && a % b); }

bool isect(iterator x, iterator y) {

if (y == end()) { x->p = inf; return false; }

if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;

else x->p = div(y->m - x->m, x->k - y->k);

return x->p >= y->p;

}

void add(long long k, long long m) {

auto z = insert({k, m, 0}), y = z++, x = y;

while (isect(y, z)) z = erase(z);

if (x != begin() && isect(--x, y)) isect(x, y = erase(y));

while ((y = x) != begin() && (--x)->p >= y->p)

isect(x, erase(y));

}

long long query(long long x) {

assert(!empty());

auto l = \*lower\_bound(x);

return l.k \* x + l.m;

}

1. DP CHIA ĐỂ TRỊ

// dp[i][k] = min(dp[j][k - 1] + C(j + 1, i))

// C(l, r): a convex function a[l] -> a[r] (exp: a[l] + ... + a[r])

// opt[i][k] <= opt[i + 1][k]

long long cost(int i, int j) {

if (i > j) return 0;

return (sum[j] - sum[i - 1]) \* (j - i + 1);

}

void solve(int k, int l, int r, int optL, int optR) {

if (l > r) return;

int mid = (l + r) / 2;

dp[mid][k] = INF;

int optM = 0;

for (int i = optL; i <= min(optR, mid); i++) {

long long new\_cost = dp[i][k - 1] + cost(i + 1, mid);

if (dp[mid][k] > new\_cost) {

dp[mid][k] = new\_cost;

optM = i;

}

}

solve(k, l, mid - 1, optL, optM);

solve(k, mid + 1, r, optM, optR);

}

for (int i = 1; i <= n; i++) dp[i][1] = cost(1, i);

for (int k = 2; k <= n; k++) solve(k, 1, n, 0, n);

1. RABIN-MILLER TEST

ll nt[1000010];

ll nhan(ll a, ll b, ll mod) {

if (b == 0) return 0%mod;

if (b == 1) return a%mod;

ll g = nhan(a,b/2,mod);

if (b%2) return ((g+g)%mod+a)%mod;

return (g+g)%mod;

}

ll modpow(ll a, ll b, ll mod) {

if (b == 0) return 1%mod;

if (b == 1) return a%mod;

ll g = modpow(a,b/2,mod);

if (b%2) return nhan(nhan(g,g,mod),a,mod);

return nhan(g,g,mod);

}

bool MillerRabin(ll n, ll seed) {

ll k = 0;

if (n < 2) return false;

if (n == 2) return true;

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

ll m = n - 1;

while (!(m & 1)) m >>= 1, k++;

ll a = seed;

a = modpow(a, m, n);

if (a == 1 || a == n - 1) return true;

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

a = modpow(a, 2, n);

if (a == 1) return false;

if (a == n - 1) return true;

}

return false;

}

void Sieve() {

FOR(i,2,1000000)

if (nt[i] == 0) {

nt[i] = 1;

for (ll j = i\*i; j <= 1000000; j += i) nt[j] = -1;

}

}

bool PrimalityTest(ll n) {

if (n <= 1000000) return nt[n] == 1;

else return MillerRabin(n, 2) && MillerRabin(n, 13) && MillerRabin(n, 23) && MillerRabin(n, 1662803);

}

int main() {

ll t, n, k;

Sieve();

cin >> t;

while (t--) {

cin >> n >> k;

if (n < 2\*k) cout << "No\n";

else if (n == 2\*k) cout << "Yes\n";

else if (k == 1) cout << (PrimalityTest(n) ? "Yes" : "No") << endl;

else if (n % 2 == 0) cout << "Yes\n";

else {

if (n == 5) cout << "Yes\n";

else if (k == 2) cout << (PrimalityTest(n-2) ? "Yes" : "No") << endl;

else cout << "Yes\n";

}

}

return 0;

}

1. LINE SWEEP – CLOSEST POINTS

set<point> box;

point a[100010];

int n;

int main() {

//int t = 0;

//int n;

cin >> n;

FOR(i,1,n) {

cin >> a[i].x >> a[i].y;

a[i].cs = i-1;

}

sort(a+1, a+n+1, cmp);

double best = 1e18;

int lef = 1;

box.insert(a[1]);

int pA, pB;

FOR(i,2,n) {

while (lef < i && a[i].x - a[lef].x > best) box.erase(a[lef++]);

for (set<point>::iterator it = box.lower\_bound(point(0, (ll) ceil(a[i].y-best))); it != box.end() && a[i].y + best >= it->y; it++) {

ll dis = dist(\*it, a[i]);

if (sqrt(dis) < best) {

best = sqrt(dis);

pA = it->cs;

pB = a[i].cs;

}

}

box.insert(a[i]);

}

if (pA > pB) swap(pA, pB);

cout << pA << " " << pB << " ";

cout << fixed << setprecision(6) << best;

return 0;

}

1. HOPCROFTKARP – MAX MATCHING IN GRAPH

#include <bits/stdc++.h>

using namespace std;

/\*

\* Complexity: O(E \* sqrt(V))

\*/

#define HK HopcroftKarp

namespace HopcroftKarp {

const int maxv = 1e3 + 5;

const int maxe = 1e6 + 5;

int nx, ny, E;

int adj[maxe], nxt[maxe];

int lst[maxv], ptr[maxv], lev[maxv], que[maxv], matx[maxv], maty[maxv];

void init(int \_nx, int \_ny) {

nx = \_nx, ny = \_ny;

E = 0, fill\_n(lst, nx, -1);

fill\_n(matx, nx, -1), fill\_n(maty, ny, -1);

}

void add(int x, int y) {

adj[E] = y, nxt[E] = lst[x], lst[x] = E++;

}

int bfs() {

int qh = 0, qe = 0;

for (int x = 0; x < nx; x++) {

if (~matx[x]) {

lev[x] = 0;

}

else {

lev[x] = 1;

que[qe++] = x;

}

}

int res = 0;

while (qh < qe) {

int x = que[qh++];

for (int e = lst[x]; ~e; e = nxt[e]) {

int y = adj[e];

if (!~maty[y]) {

res = 1;

}

else if (!lev[maty[y]]) {

lev[maty[y]] = lev[x] + 1;

que[qe++] = maty[y];

}

}

}

return res;

}

int dfs(int x) {

for (int& e = ptr[x]; ~e; e = nxt[e]) {

int y = adj[e];

if (!~maty[y] || (lev[maty[y]] == lev[x] + 1 && dfs(maty[y]))) {

matx[x] = y;

maty[y] = x;

return 1;

}

}

return 0;

}

int maxmat() {

int res = 0;

while (bfs()) {

for (int x = 0; x < nx; x++) {

ptr[x] = lst[x];

}

for (int x = 0; x < nx; x++) {

if (!~matx[x]) {

res += dfs(x);

}

}

}

return res;

}

int vis[2][maxv];

void dfs(int u, int r) {

vis[r][u] = 1;

if (!r) {

for (int e = lst[u]; ~e; e = nxt[e]) {

int v = adj[e];

if (matx[u] != v && !vis[!r][v]) {

dfs(v, !r);

}

}

}

else {

int v = maty[u];

if (~v && !vis[!r][v]) {

dfs(v, !r);

}

}

}

vector<int> mincover() {

vector<int> res;

fill\_n(vis[0], nx, 0), fill\_n(vis[1], ny, 0);

for (int x = 0; x < nx; x++) {

if (!~matx[x] && !vis[0][x]) {

dfs(x, 0);

}

}

for (int x = 0; x < nx; x++) {

if (!vis[0][x]) {

res.push\_back(x);

}

}

for (int y = 0; y < ny; y++) {

if (vis[1][y]) {

res.push\_back(nx + y);

}

}

return res;

}

vector<int> maxind() {

vector<int> res;

fill\_n(vis[0], nx, 0), fill\_n(vis[1], ny, 0);

for (int x = 0; x < nx; x++) {

if (!~matx[x] && !vis[0][x]) {

dfs(x, 0);

}

}

for (int x = 0; x < nx; x++) {

if (vis[0][x]) {

res.push\_back(x);

}

}

for (int y = 0; y < ny; y++) {

if (!vis[1][y]) {

res.push\_back(nx + y);

}

}

return res;

}

}

int main() {

HK::init(3, 3);

HK::add(0, 1);

HK::add(0, 2);

HK::add(1, 2);

cout << HK::maxmat() << "\n";

for (int x : HK::mincover()) {

cout << x << " ";

}

cout << "\n";

for (int x : HK::maxind()) {

cout << x << " ";

}

cout << "\n";

return 0;

}

1. INDEX TREE

#include <ext/pb\_ds/assoc\_container.hpp>

using namespace \_\_gnu\_pbds;

typedef tree<int,null\_type,less<int>,rb\_tree\_tag,tree\_order\_statistics\_node\_update> IndexTree;

s={2,5,6,10}

\*s.find\_by\_order(2)=6;

s.order\_of\_key(x) == lower\_bound(x)-s.begin();

1. NUMBER THEORY

// This is a collection of useful code for solving problems that

// involve modular linear equations. Note that all of the

// algorithms described here work on nonnegative integers.

// C++11

typedef vector<int> VI;

typedef pair<int, int> PII;

// return a % b (positive value)

int mod(int a, int b) {

return ((a%b) + b) % b;

}

// computes gcd(a,b)

int gcd(int a, int b) {

while (b) { int t = a%b; a = b; b = t; }

return a;

}

// computes lcm(a,b)

int lcm(int a, int b) {

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

}

// (aˆb) mod m via successive squaring

int powermod(int a, int b, int m) {

int ret = 1;

while (b) {

if (b & 1) ret = mod(ret\*a, m);

a = mod(a\*a, m);

b >>= 1;

}

return ret;

}

// returns g = gcd(a, b); finds x, y such that d = ax + by

int extended\_euclid(int a, int b, int &x, int &y) {

int xx = y = 0;

int yy = x = 1;

while (b) {

int q = a / b;

int t = b; b = a%b; a = t;

t = xx; xx = x - q\*xx; x = t;

t = yy; yy = y - q\*yy; y = t;

}

return a;

}

// finds all solutions to ax = b (mod n)

VI modular\_linear\_equation\_solver(int a, int b, int n) {

int x, y;

VI ret;

int g = extended\_euclid(a, n, x, y);

if (!(b%g)) {

x = mod(x\*(b / g), n);

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

ret.push\_back(mod(x + i\*(n / g), n));

}

return ret;

}

// computes b such that ab = 1 (mod n), returns -1 on failure

int mod\_inverse(int a, int n) {

int x, y;

int g = extended\_euclid(a, n, x, y);

if (g > 1) return -1;

return mod(x, n);

}

// Chinese remainder theorem (special case): find z such that

// z % m1 = r1, z % m2 = r2. Here, z is unique modulo M = lcm(m1, m2).

// Return (z, M). On failure, M = -1.

PII chinese\_remainder\_theorem(int m1, int r1, int m2, int r2) {

int s, t;

int g = extended\_euclid(m1, m2, s, t);

if (r1%g != r2%g) return make\_pair(0, -1);

return make\_pair(mod(s\*r2\*m1 + t\*r1\*m2, m1\*m2) / g, m1\*m2 / g);

}

// Chinese remainder theorem: find z such that

// z % m[i] = r[i] for all i. Note that the solution is

// unique modulo M = lcm\_i (m[i]). Return (z, M). On

// failure, M = -1. Note that we do not require the a[i]’s

// to be relatively prime.

PII chinese\_remainder\_theorem(const VI &m, const VI &r) {

PII ret = make\_pair(r[0], m[0]);

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

ret = chinese\_remainder\_theorem(ret.second, ret.first, m[i], r[i]);

if (ret.second == -1) break;

}

return ret;

}

// computes x and y such that ax + by = c

// returns whether the solution exists

bool linear\_diophantine(int a, int b, int c, int &x, int &y) {

if (!a && !b) {

if (c) return false;

x = 0; y = 0;

return true;

}

if (!a) {

if (c % b) return false;

x = 0; y = c / b;

return true;

}

if (!b) {

if (c % a) return false;

x = c / a; y = 0;

return true;

}

int g = gcd(a, b);

if (c % g) return false;

x = c / g \* mod\_inverse(a / g, b / g);

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

return true;

}

int main() {

// expected: 2

cout << gcd(14, 30) << endl;

// expected: 2 -2 1

int x, y;

int g = extended\_euclid(14, 30, x, y);

cout << g << " " << x << " " << y << endl;

// expected: 95 451

VI sols = modular\_linear\_equation\_solver(14, 30, 100);

for (int i = 0; i < (int) sols.size(); i++) cout << sols[i] << " ";

cout << endl;

// expected: 8

cout << mod\_inverse(8, 9) << endl;

// expected: 23 105

// 11 12

PII ret = chinese\_remainder\_theorem(VI({ 3, 5, 7 }), VI({ 2, 3, 2 }));

cout << ret.first << " " << ret.second << endl;

ret = chinese\_remainder\_theorem(VI({ 4, 6 }), VI({ 3, 5 }));

cout << ret.first << " " << ret.second << endl;

// expected: 5 -15

if (!linear\_diophantine(7, 2, 5, x, y)) cout << "ERROR" << endl;

cout << x << " " << y << endl;

return 0;

}

1. LINEAR EQUATIONS + MATRIX INVERSE + DETERMINANT

// Gauss-Jordan elimination with full pivoting.

//

// Uses:

// (1) solving systems of linear equations (AX=B)

// (2) inverting matrices (AX=I)

// (3) computing determinants of square matrices

//

// Running time: O(nˆ3)

//

// INPUT: a[][] = an nxn matrix

// b[][] = an nxm matrix

//

// OUTPUT: X = an nxm matrix (stored in b[][])

// Aˆ{-1} = an nxn matrix (stored in a[][])

// returns determinant of a[][]

#include <iostream>

#include <vector>

#include <cmath>

using namespace std;

const double EPS = 1e-10;

typedef vector<int> VI;

typedef double T;

typedef vector<T> VT;

typedef vector<VT> VVT;

T GaussJordan(VVT &a, VVT &b) {

const int n = a.size();

const int m = b[0].size();

VI irow(n), icol(n), ipiv(n);

T det = 1;

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

int pj = -1, pk = -1;

for (int j = 0; j < n; j++) if (!ipiv[j])

for (int k = 0; k < n; k++) if (!ipiv[k])

if (pj == -1 || fabs(a[j][k]) > fabs(a[pj][pk])) {

pj = j;

pk = k;

}

if (fabs(a[pj][pk]) < EPS) {

cerr << "Matrix is singular." << endl;

return 0;

}

ipiv[pk]++;

swap(a[pj], a[pk]);

swap(b[pj], b[pk]);

if (pj != pk) det \*= -1;

irow[i] = pj;

icol[i] = pk;

T c = 1.0 / a[pk][pk];

det \*= a[pk][pk];

a[pk][pk] = 1.0;

for (int p = 0; p < n; p++) a[pk][p] \*= c;

for (int p = 0; p < m; p++) b[pk][p] \*= c;

for (int p = 0; p < n; p++) if (p != pk) {

c = a[p][pk];

a[p][pk] = 0;

for (int q = 0; q < n; q++) a[p][q] -= a[pk][q] \* c;

for (int q = 0; q < m; q++) b[p][q] -= b[pk][q] \* c;

}

}

for (int p = n-1; p >= 0; p--) if (irow[p] != icol[p]) {

for (int k = 0; k < n; k++) swap(a[k][irow[p]], a[k][icol[p]]);

}

return det;

}

int main() {

const int n = 4;

const int m = 2;

double A[n][n] = { {1,2,3,4},{1,0,1,0},{5,3,2,4},{6,1,4,6} };

double B[n][m] = { {1,2},{4,3},{5,6},{8,7} };

VVT a(n), b(n);

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

a[i] = VT(A[i], A[i] + n);

b[i] = VT(B[i], B[i] + m);

}

double det = GaussJordan(a, b);

// expected: 60

cout << "Determinant: " << det << endl;

// expected: -0.233333 0.166667 0.133333 0.0666667

// 0.166667 0.166667 0.333333 -0.333333

// 0.233333 0.833333 -0.133333 -0.0666667

// 0.05 -0.75 -0.1 0.2

cout << "Inverse: " << endl;

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

for (int j = 0; j < n; j++) cout << a[i][j] << ' ';

cout << endl;

}

// expected: 1.63333 1.3

// -0.166667 0.5

// 2.36667 1.7

// -1.85 -1.35

cout << "Solution: " << endl;

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

for (int j = 0; j < m; j++) cout << b[i][j] << ' ' ;

cout << endl;

}

}

1. THEOREMS

**Cayley’s Formula**: There are n^(n−2) spanning trees of a complete graph with n labeled vertices.

**Derangement**: A permutation of the elements of a set such that none of the elements appear in their original position. The number of derangements der(n) can be computed as follow: der(n)=(n−1)×(der(n−1)+der(n−2)) where der(0) = 1 and der(1) = 0.

**Planar Graph**: V−E+F=2,where F is the number of faces of the Planar Graph.

Moser’s: Determine the number of pieces into which a circle is divided if n points on its circumference are joined by chords with no three internally concurrent. Solution : g(n)= nC4 + nC2 + 1

**Pick’s Theorem**: Let i be the number of integer points in the polygon, A be the area of the polygon, and b be the number of integer points on the boundary, then A= i + b/2 −1.

The number of spanning tree of a complete bipartite graph K(n, m) is m^(n−1)×n^(m−1).

**Erdos Gallai’s Theorem** gives a necessary and sufficient condition for a finite sequence of natural numbers to be the degree sequence of a simple graph. A sequence of non negative integers d1 >= d2 >= .. >= dn can be the degree sequence of a simple graph on n vertices if sum(di) is even and sum(I = 1 -> k) di <= k\*(k-1) + sum(I = k+1 -> n) min(di,k) holds for 1<=k<=n

**Kirchhoff’s theorem**: (Calculate the number of spanning trees of a graph)

Construct matrix L: L[i,i]=deg[i]; L[i,j]=-1 (if edge ij) else L[i,j]=0;

Remove any row and any column, the result is Det of the remaining matrix

**Point inside a polygon**: Ve 1 tia tu diem dó. Neu tia cat so le canh thi return 1; else return 0;

**Burnside lemma**: (sum of c(k))/n, there are n ways to change, c(k) combinations remain unchanged when applying the k th way.

**Matrix-Graph**: V[i,j]=1 if there is an edge from i -> j

Vn[i,j] number of path of n edges from i->j

**Shortest path** : AB[i,j]=min (A[i,k]+B[k,j]); V[i,j]=INF if no egde i->j

Vn[i,j] shortest path of length n from i->j

**Phi Euler** : phi(n) = (p-1)\*p^(k-1); với n=p^k;

**Đồng dư :** Nếu gcd(d,n)=1, a=b mod n, a\*d = b\*d mod n;

***Định lý Euler:***Cho, và là hàm Euler. Khi đó ta có:*;* Cho, *p* là số nguyên tố ;{\displaystyle \phi (n)=(p-1)p^{k-1}}

1. MIN COST MATCHING

// Min cost bipartite matching via shortest augmenting paths

//

// This is an O(nˆ3) implementation of a shortest augmenting path

// algorithm for finding min cost perfect matchings in dense

// graphs. In practice, it solves 1000x1000 problems in around 1

// second.

//

// cost[i][j] = cost for pairing left node i with right node j

// Lmate[i] = index of right node that left node i pairs with

// Rmate[j] = index of left node that right node j pairs with

//

// The values in cost[i][j] may be positive or negative. To perform

// maximization, simply negate the cost[][] matrix.

typedef vector<double> VD;

typedef vector<VD> VVD;

typedef vector<int> VI;

double MinCostMatching(const VVD &cost, VI &Lmate, VI &Rmate) {

int n = int(cost.size());

// construct dual feasible solution

VD u(n);

VD v(n);

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

u[i] = cost[i][0];

for (int j = 1; j < n; j++) u[i] = min(u[i], cost[i][j]);

}

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

v[j] = cost[0][j] - u[0];

for (int i = 1; i < n; i++) v[j] = min(v[j], cost[i][j] - u[i]);

}

// construct primal solution satisfying complementary slackness

Lmate = VI(n, -1);

Rmate = VI(n, -1);

int mated = 0;

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

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

if (Rmate[j] != -1) continue;

if (fabs(cost[i][j] - u[i] - v[j]) < 1e-10) {

Lmate[i] = j;

Rmate[j] = i;

mated++;

break;

}

}

}

VD dist(n);

VI dad(n);

VI seen(n);

// repeat until primal solution is feasible

while (mated < n) {

// find an unmatched left node

int s = 0;

while (Lmate[s] != -1) s++;

// initialize Dijkstra

fill(dad.begin(), dad.end(), -1);

fill(seen.begin(), seen.end(), 0);

for (int k = 0; k < n; k++) dist[k] = cost[s][k] - u[s] - v[k];

int j = 0;

while (true) {

// find closest

j = -1;

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

if (seen[k]) continue;

if (j == -1 || dist[k] < dist[j]) j = k;

}

seen[j] = 1;

// termination condition

if (Rmate[j] == -1) break;

// relax neighbors

const int i = Rmate[j];

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

if (seen[k]) continue;

const double new\_dist = dist[j] + cost[i][k] - u[i] - v[k];

if (dist[k] > new\_dist) {

dist[k] = new\_dist;

dad[k] = j;

}

}

}

// update dual variables

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

if (k == j || !seen[k]) continue;

const int i = Rmate[k];

v[k] += dist[k] - dist[j];

u[i] -= dist[k] - dist[j];

}

u[s] += dist[j];

// augment along path

while (dad[j] >= 0) {

const int d = dad[j];

Rmate[j] = Rmate[d];

Lmate[Rmate[j]] = j;

j = d;

}

Rmate[j] = s;

Lmate[s] = j;

mated++;

}

double value = 0;

for (int i = 0; i < n; i++) value += cost[i][Lmate[i]];

return value;

}

1. MIN COST MAX FLOW

// Implementation of min cost max flow algorithm using adjacency

// matrix (Edmonds and Karp 1972). This implementation keeps track of

// forward and reverse edges separately (so you can set cap[i][j] !=

// cap[j][i]). For a regular max flow, set all edge costs to 0.

//

// Running time, O(|V|ˆ2) cost per augmentation

// max flow: O(|V|ˆ3) augmentations

// min cost max flow: O(|V|ˆ4 \* MAX\_EDGE\_COST) augmentations

//

// INPUT:

// - graph, constructed using AddEdge()

// - source

// - sink

//

// OUTPUT:

// - (maximum flow value, minimum cost value)

// - To obtain the actual flow, look at positive values only.

typedef vector<int> VI;

typedef vector<VI> VVI;

typedef long long L;

typedef vector<L> VL;

typedef vector<VL> VVL;

typedef pair<int, int> PII;

typedef vector<PII> VPII;

const L INF = numeric\_limits<L>::max() / 4;

struct MinCostMaxFlow {

int N;

VVL cap, flow, cost;

VI found;

VL dist, pi, width;

VPII dad;

MinCostMaxFlow(int N) : N(N), cap(N, VL(N)), flow(N, VL(N)), cost(N, VL(N)), found(N), dist(N), pi(N), width(N), dad(N) {}

void AddEdge(int from, int to, L cap, L cost) {

this->cap[from][to] = cap;

this->cost[from][to] = cost;

}

void Relax(int s, int k, L cap, L cost, int dir) {

L val = dist[s] + pi[s] - pi[k] + cost;

if (cap && val < dist[k]) {

dist[k] = val;

dad[k] = make\_pair(s, dir);

width[k] = min(cap, width[s]);

}

}

L Dijkstra(int s, int t) {

fill(found.begin(), found.end(), false);

fill(dist.begin(), dist.end(), INF);

fill(width.begin(), width.end(), 0);

dist[s] = 0;

width[s] = INF;

while (s != -1) {

int best = -1;

found[s] = true;

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

if (found[k]) continue;

Relax(s, k, cap[s][k] - flow[s][k], cost[s][k], 1);

Relax(s, k, flow[k][s], -cost[k][s], -1);

if (best == -1 || dist[k] < dist[best]) best = k;

}

s = best;

}

for (int k = 0; k < N; k++) pi[k] = min(pi[k] + dist[k], INF);

return width[t];

}

pair<L, L> GetMaxFlow(int s, int t) {

L totflow = 0, totcost = 0;

while (L amt = Dijkstra(s, t)) {

totflow += amt;

for (int x = t; x != s; x = dad[x].first) {

if (dad[x].second == 1) {

flow[dad[x].first][x] += amt;

totcost += amt \* cost[dad[x].first][x];

}

else {

flow[x][dad[x].first] -= amt;

totcost -= amt \* cost[x][dad[x].first];

}

}

}

return make\_pair(totflow, totcost);

}

};

// BEGIN CUT

// The following code solves UVA problem #10594: Data Flow

int main() {

int N, M;

while (scanf("%d%d", &N, &M) == 2) {

VVL v(M, VL(3));

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

scanf("%Ld%Ld%Ld", &v[i][0], &v[i][1], &v[i][2]);

L D, K;

scanf("%Ld%Ld", &D, &K);

MinCostMaxFlow mcmf(N+1);

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

mcmf.AddEdge(int(v[i][0]), int(v[i][1]), K, v[i][2]);

mcmf.AddEdge(int(v[i][1]), int(v[i][0]), K, v[i][2]);

}

mcmf.AddEdge(0, 1, D, 0);

pair<L, L> res = mcmf.GetMaxFlow(0, N);

if (res.first == D) printf("%Ld\n", res.second);

else printf("Impossible.\n");

}

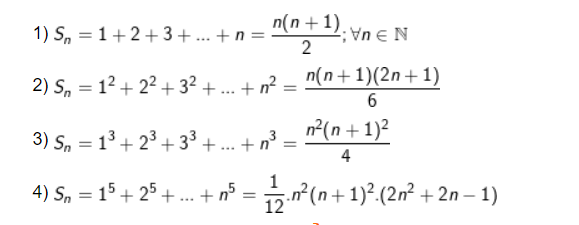
return 0;

}

// END CUT

1. CÔNG THỨC TOÁN HỌC

Công thức tính diện tích tam giác:



1. DEFAULT CODE

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

typedef pair<ll,ll>ii;

#define X first

#define Y second

#define pb push\_back

#define FOR(i,a,b) for(ll i=a;i<=b;i++)

#define FOD(i,a,b) for(ll i=a;i>=b;i--)

#define all(a) (a).begin(),(a).end()

#define uni(a) (a).resize(unique(all(a)) - (a).begin())

const ll base = 1e6+7;

const ll maxN = 1e6+5;

int main() {

//freopen("input.txt", "r", stdin);

//freopen("output.txt", "w", stdout);

ios\_base::sync\_with\_stdio(0);

return 0;

}

1. BIGNUM

string add(string a, string b) {

string res="";

while(a.length() < b.length()) a="0"+a;

while(b.length() < a.length()) b="0"+b;

int carry=0;

for(int i=a.length()-1;i>=0;i--)

{

int tmp=a[i]-48 + b[i]-48 + carry;

carry=tmp/10;

tmp=tmp%10;

res=(char)(tmp+48)+res;

}

if(carry>0) res="1"+res;

return res;

}

string sub(string a, string b) {

string res="";

while(a.length() < b.length()) a="0"+a;

while(b.length() < a.length()) b="0"+b;

bool neg=false;

if(a<b)

{

swap(a,b);

neg=true;

}

int borrow=0;

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

int tmp=a[i]-b[i]-borrow;

if(tmp<0) {

tmp+=10;

borrow=1;

}

else borrow=0;

res=(char)(tmp%10 + 48) + res;

}

while(res.length()>1 && res[0]=='0') res.erase(0,1);

if(neg) res="-"+res;

return res;

}

string mul(string a, string b) {

string res="";

int n=a.length();

int m=b.length();

int len=n+m-1;

int carry=0;

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

int tmp=0;

for(int j=n-1;j>=0;j--)

if(i-j<=m && i-j>=1)

{

int a1=a[j]-48;

int b1=b[i-j-1]-48;

tmp+=a1\*b1;

}

tmp+=carry;

carry=tmp/10;

res=(char)(tmp%10 + 48)+res;

}

while(res.length()>1 && res[0]=='0') res.erase(0,1);

return res;

}

1. COMPRESS

template <class T>

struct Compressor {

vector<T> values, orig;

void add(T x) {

values.push\_back(x);

}

void compress() {

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

values.erase(unique(values.begin(), values.end()), values.end());

orig.resize(values.size() + 1);

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

orig[i + 1] = values[i];

}

}

T find(T x) {

return lower\_bound(values.begin(), values.end(), x) - values.begin() + 1;

}

};

1. DUVAL + MIN\_CYCLE\_STRING

vector<string> duval(string const& s) {

int n = s.size();

int i = 0;

vector<string> factorization;

while (i < n) {

int j = i + 1, k = i;

while (j < n && s[k] <= s[j]) {

if (s[k] < s[j])

k = i;

else

k++;

j++;

}

while (i <= k) {

factorization.push\_back(s.substr(i, j - k));

i += j - k;

}

}

return factorization;

}

string min\_cyclic\_string(string s) {

s += s;

int n = s.size();

int i = 0, ans = 0;

while (i < n / 2) {

ans = i;

int j = i + 1, k = i;

while (j < n && s[k] <= s[j]) {

if (s[k] < s[j])

k = i;

else

k++;

j++;

}

while (i <= k)

i += j - k;

}

return s.substr(ans, n / 2);

}

1. RANDOM

mt19937 gen(chrono::steady\_clock::now().time\_since\_epoch().count());

mt19937\_64 gen(chrono::steady\_clock::now().time\_since\_epoch().count());

ll Rand(ll l, ll r){

uniform\_int\_distribution<ll> rnd(l,r); return rnd(gen);

}

srand(time(NULL));

rand() % (max - min + 1) + min;