**Template**

#pragma GCC optimize("Ofast")

#include <bits/stdc++.h>

using namespace std;

#define x first

#define y second

#define ll long long

#define pi pair<int,int>

#define pl pair<ll,ll>

#define pd pair<double,double>

#define ld long double

#define pld pair<ld,ld>

#define lg length()

#define sz size()

#define vi vector<int>

#define vl vector<ll>

#define vp vector<pi>

#define vpl vector<pl>

#define pb push\_back

#define INF 1000000005

#define LINF 1000000000000000005

int32\_t main(){

ios\_base :: sync\_with\_stdio(0); cin.tie(); cout.tie();

#ifdef LOCAL\_DEFINE

ifstream cin("input.txt");

#endif

}

#ifdef LOCAL\_DEFINE

mt19937 rng(69);

#else

seed\_seq seq{

(uint64\_t) chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now().time\_since\_epoch()).count(),

(uint64\_t) \_\_builtin\_ia32\_rdtsc(),

(uint64\_t) (uintptr\_t) make\_unique<char>().get()

};

mt19937 rng(seq);

#endif

**Euclid Extended**

long long a,b,c,d,x,y,t;

void gcd(long long a, long long b, long long \*d, long long \*x, long long \*y){

if(b==0){

\*d=a;

\*x=1;

\*y=0;

}

else{

long long x0, y0;

gcd(b,a%b,d,&x0,&y0);

\*x=y0;

\*y=x0-(a/b)\*y0;

}

}

int main()

{

ifstream in("euclid3.in");

ofstream out("euclid3.out");

in >> t;

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

in >> a >> b >> c;

gcd(a,b,&d,&x,&y);

if(c%d) out << 0 << ' ' << 0 << '\n';

else out << x\*c/d << ' ' << y\*c/d << '\n';

}

}

**String matching**

string a,b;

int p[2000005],ans;

vector <int> sol;

int main(){

ios\_base :: sync\_with\_stdio(0); cin.tie(); cout.tie();

ifstream cin("strmatch.in");

ofstream cout("strmatch.out");

cin >> a >> b;

int j=0;

for(int i=1;i<a.sz;i++){

while(j>0 && a[j]!=a[i]) j=p[j-1];

if(a[j]==a[i]) j++;

p[i]=j;

}

j=0;

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

while(j>0 && b[i]!=a[j]) j=p[j-1];

if(b[i]==a[j]) j++;

if(j==a.sz){

ans++;

if(ans<=1000) sol.push\_back(i-a.sz+1);

}

}

cout << ans << '\n';

for(int i=0;i<sol.sz;i++){

cout << sol[i] << ' ';

}

}

**Max matching in bipartite graph**

int n,m,e,f[240005],c[240005],x,y,z,p[20005],k[20005];

vector <pi> g[20005];

int inv(int x){

if(x<=e+n+m) return x+e+n+m;

else return x-e-n-m;

}

int32\_t main(){

ios\_base :: sync\_with\_stdio(0); cin.tie(); cout.tie();

ifstream cin("cuplaj.in");

ofstream cout("cuplaj.out");

cin >> n >> m >> e;

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

cin >> x >> y;

g[x].pb({n+y,i});

g[n+y].pb({x,e+n+m+i});

c[i]=1;

}

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

g[0].pb({i,e+i});

g[i].pb({0,2\*e+n+m+i});

c[e+i]=1;

}

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

g[n+i].pb({n+m+1,e+n+i});

g[n+m+1].pb({n+i,2\*e+2\*n+m+i});

c[e+n+i]=1;

}

int gd=0;

do{

gd=0;

for(int i=1;i<=n+m+1;i++) p[i]=-1,k[i]=0;

queue <int> q;

q.push(0);

while(q.sz){

int t=q.front();

q.pop();

for(pi i : g[t]){

if(p[i.x]==-1 && i.x && i.x!=n+m+1 && f[i.y]<c[i.y]){

q.push(i.x);

p[i.x]=t;

k[i.x]=i.y;

}

}

}

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

int s=e+i;

if(p[i]!=-1 && f[s]<c[s]){

gd=1;

int fl=c[s]-f[s],t=i;

while(t){

fl=min(fl,c[k[t]]-f[k[t]]);

t=p[t];

}

t=i;

f[s]+=fl;

f[inv(s)]-=fl;

while(t){

f[k[t]]+=fl;

f[inv(k[t])]-=fl;

t=p[t];

}

}

}

}while(gd);

int ans=0;

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

ans+=f[e+n+i];

}

cout << ans << '\n';

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

for(pi j : g[i]){

if(j.x && f[j.y]){

cout << i << ' ' << j.x-n << '\n';

}

}

}

}

**Alternative**

#define MAX 100001

#define NIL 0

#define INF (1<<28)

vector< int > G[MAX];

int n, m, match[MAX], dist[MAX];

// n: number of nodes on left side, nodes are numbered 1 to n

// m: number of nodes on right side, nodes are numbered n+1 to n+m

// G = NIL[0] ∪ G1[G[1---n]] ∪ G2[G[n+1---n+m]]

bool bfs() {

int i, u, v, len;

queue< int > Q;

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

if(match[i]==NIL) {

dist[i] = 0;

Q.push(i);

}

else dist[i] = INF;

}

dist[NIL] = INF;

while(!Q.empty()) {

u = Q.front(); Q.pop();

if(u!=NIL) {

len = G[u].size();

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

v = G[u][i];

if(dist[match[v]]==INF) {

dist[match[v]] = dist[u] + 1;

Q.push(match[v]);

}

}

}

}

return (dist[NIL]!=INF);

}

bool dfs(int u) {

int i, v, len;

if(u!=NIL) {

len = G[u].size();

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

v = G[u][i];

if(dist[match[v]]==dist[u]+1) {

if(dfs(match[v])) {

match[v] = u;

match[u] = v;

return true;

}

}

}

dist[u] = INF;

return false;

}

return true;

}

int hopcroft\_karp() {

int matching = 0, i;

// match[] is assumed NIL for all vertex in G

while(bfs())

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

if(match[i]==NIL && dfs(i))

matching++;

return matching;

}

**Trie**

int n;

struct trie{

int cnt,f;

int k[26];

} gol;

vector <trie> t;

string s;

void Add(int nod, int p){

if(t[nod].k[s[p]-'a']==-1){

t.push\_back(gol);

t[nod].k[s[p]-'a']=t.size()-1;

}

t[t[nod].k[s[p]-'a']].cnt++;

if(p==s.lg-1){

t[t[nod].k[s[p]-'a']].f++;

return;

}

Add(t[nod].k[s[p]-'a'],p+1);

}

void Rem(int nod, int p){

t[t[nod].k[s[p]-'a']].cnt--;

if(p==s.lg-1){

t[t[nod].k[s[p]-'a']].f--;

return;

}

Rem(t[nod].k[s[p]-'a'],p+1);

}

int Cnt(int nod, int p){

if(t[nod].k[s[p]-'a']==-1) return 0;

if(p==s.lg-1) return t[t[nod].k[s[p]-'a']].f;

return Cnt(t[nod].k[s[p]-'a'],p+1);

}

int Pre(int nod, int p){

if(t[nod].k[s[p]-'a']==-1) return 0;

int ret=0;

if(t[t[nod].k[s[p]-'a']].cnt) ret=p+1;

if(p==s.lg-1) return ret;

return max(ret,Pre(t[nod].k[s[p]-'a'],p+1));

}

int32\_t main(){

ios\_base :: sync\_with\_stdio(0); cin.tie(); cout.tie();

for(int i=0;i<26;i++) gol.k[i]=-1; gol.cnt=0; gol.f=0;

t.push\_back(gol);

ifstream cin("trie.in");

ofstream cout("trie.out");

while(cin >> n >> s){

if(n==0){

Add(0,0);

}

else if(n==1){

Rem(0,0);

}

else if(n==2){

cout << Cnt(0,0) << '\n';

}

else{

cout << Pre(0,0) << '\n';

}

}

}

**Alternative**

using namespace std;

int c; string s;

struct Trie{

int nr, nrfii;

Trie \*fii[26];

Trie(){

nr = nrfii=0;

for(int i=0;i<=26;i++) fii[i]=0;

}

};

Trie \*T = new Trie;

void add(Trie \*nod, char \*s){

if(\*s=='\n'){

nod->nr++;

return;

}

else{

if(nod->fii[\*s-'a']==0){

nod->fii[\*s-'a'] = new Trie;

nod->nrfii++;

}

add(nod->fii[\*s-'a'], s+1);

}

}

bool rem(Trie \*nod, char \*s){

if(\*s=='\n'){

nod->nr--;

}

else if(rem(nod->fii[\*s-'a'],s+1)){

nod->nrfii--;

nod->fii[\*s-'a']=0;

}

if(nod->nr==0 && nod->nrfii==0 && nod!=T){

delete nod; return 1;

}

return 0;

}

int qry(Trie \*nod, char \*s){

if(\*s=='\n'){

return nod->nr;

}

else{

if(nod->fii[\*s-'a']==0) return 0;

else{

return qry(nod->fii[\*s-'a'],s+1);

}

}

}

int lng(Trie \*nod, char \*s, int p){

if(\*s=='\n'){

return p;

}

else{

if(nod->fii[\*s-'a']==0) return p;

else{

return lng(nod->fii[\*s-'a'],s+1,p+1);

}

}

}

int main()

{

char line[ 32 ];

freopen( "trie.in", "r", stdin );

freopen( "trie.out", "w", stdout );

fgets( line, 32, stdin );

while( !feof( stdin ) ) {

switch( line[0] ) {

case '0': add( T, line+2 ); break;

case '1': rem( T, line+2 ); break;

case '2': printf( "%d\n", qry( T, line+2 ) ); break;

case '3': printf( "%d\n", lng( T, line+2, 0 ) ); break;

}

fgets( line, 32, stdin );

}

return 0;

}

**Heaps**

using namespace std;

int h[200005],n,sh,c,x,a[200005],b[200005],t;

void up(int pos){

if(pos==1) return;

if(h[pos/2]<=h[pos]) return;

swap(a[b[pos/2]],a[b[pos]]);

swap(b[pos/2],b[pos]);

swap(h[pos/2],h[pos]);

up(pos/2);

}

void down(int pos){

if(2\*pos>sh) return;

if(2\*pos==sh){

if(h[pos]<=h[2\*pos]) return;

swap(a[b[pos]],a[b[2\*pos]]);

swap(b[pos],b[2\*pos]);

swap(h[pos],h[2\*pos]);

}

else{

if(h[pos]<=h[2\*pos] && h[pos]<=h[2\*pos+1]) return;

int fiu;

if(h[2\*pos]<h[2\*pos+1]) fiu=2\*pos;

else fiu=2\*pos+1;

swap(a[b[pos]],a[b[fiu]]);

swap(b[pos],b[fiu]);

swap(h[pos],h[fiu]);

down(fiu);

}

}

void del(int pos){

if(pos==sh){

sh--;

return;

}

h[pos]=-1;

swap(h[pos],h[sh]);

swap(a[b[pos]],a[b[sh]]);

swap(b[pos],b[sh]);

sh--;

if(pos==1) down(pos);

else if (h[pos]>h[pos/2]) down(pos);

else up(pos);

}

int main()

{

ifstream in("heapuri.in");

ofstream out("heapuri.out");

in >> n;

for(int i=1;i<=n;i++) h[i]=-1;

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

in >> c;

if(c==1){

in >> x;

sh++, t++;

h[sh]=x;

b[sh]=t;

a[t]=sh;

up(sh);

}

else if(c==2){

in >> x;

//cout << a[x] << ' ' << sh << ' ' << h[a[x]] << ' ' << b[a[x]] << ' ' << x << ' ' << h[1] << '\n';

del(a[x]);

}

else out << h[1] << '\n';

//for(int i=1;i<=sh;i++) cout << h[i] << ' ';

//cout << '\n';

}

}

**SCC (Tarjan)**

int n,m,x,y,v[200005],k,w[200005];

vector <int> g[200005];

vector <vector<int>> ans;

stack <int> s;

void DFS(int nod){

if(w[nod]) return;

k++;

w[nod]=v[nod]=k;

s.push(nod);

for(int i : g[nod]){

DFS(i);

if(v[i]>0) v[nod]=min(v[nod],v[i]);

}

if(v[nod]==w[nod]){

int t; ans.pb({});

do{

t=s.top();

s.pop();

v[t]\*=-1;

ans[ans.sz-1].pb(t);

}while(t!=nod);

}

}

int32\_t main(){

ios\_base :: sync\_with\_stdio(0); cin.tie(); cout.tie();

ifstream cin("ctc.in");

ofstream cout("ctc.out");

cin >> n >> m;

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

cin >> x >> y;

g[x].pb(y);

}

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

if(!v[i]) DFS(i);

}

cout << ans.sz << '\n';

for(vector <int> i : ans){

for(int j : i) cout << j << ' ';

cout << '\n';

}

}

**Convex Hull**

int n,id;

pld a[120005];

ld x,y=LINF;

ld cp(pld a, pld b, pld c){

return (b.y-a.y)\*(c.x-a.x)-(b.x-a.x)\*(c.y-a.y);

}

bool comp(pld x, pld y){

return cp(a[1],x,y)<0;

}

int32\_t main(){

ios\_base :: sync\_with\_stdio(0); cin.tie(); cout.tie();

ifstream cin("infasuratoare.in");

ofstream cout("infasuratoare.out");

cin >> n;

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

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

if(a[i].y<y) y=a[i].y,x=a[i].x,id=i;

else if(a[i].y==y && a[i].x<x) x=a[i].x,id=i;

}

swap(a[1],a[id]);

sort(a+2,a+n+1,comp);

vector <pld> p;

p.pb({x,y});

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

while(p.sz>1){

pld x=p[p.sz-2],y=p[p.sz-1],z=a[i];

if(cp(x,y,z)>0) p.pop\_back();

else break;

}

p.pb(a[i]);

}

cout << p.sz << '\n';

cout << fixed << setprecision(13);

for(pld i : p){

cout << i.x << ' ' << i.y << '\n';

}

}

**Biconnected Components**

int n,m,x,y,v[200005],k,w[200005];

vector <int> g[200005];

vector <vector<int>> ans;

stack <pi> s;

void DFS(int nod, int p, int h){

v[nod]=h; w[nod]=h;

for(int i : g[nod]){

if(i==p) continue;

if(!v[i]){

s.push({nod,i}),DFS(i,nod,h+1);

w[nod]=min(w[nod],w[i]);

if(w[i]>=v[nod]){

pi t; ans.pb({});

do{

t=s.top(); s.pop();

ans[ans.sz-1].pb(t.y);

}while(t!=make\_pair(nod,i));

ans[ans.sz-1].pb(nod);

}

}

else w[nod]=min(w[nod],v[i]);

}

}

int32\_t main(){

ios\_base :: sync\_with\_stdio(0); cin.tie(); cout.tie();

ifstream cin("biconex.in");

ofstream cout("biconex.out");

cin >> n >> m;

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

cin >> x >> y;

g[x].pb(y);

g[y].pb(x);

}

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

if(!v[i]) DFS(i,0,1);

}

cout << ans.sz << '\n';

for(vector <int> i : ans){

for(int j : i) cout << j << ' ';

cout << '\n';

}

}

**Max Flow**

int n,m,c[1005][1005],f[1005][1005],v[1005],x,y,z,ans,p[1005];

vector <int> g[1005];

queue <int> l;

bool BFS(){

l.push(1); v[1]=1;

while(l.size()){

int nod=l.front();

l.pop();

if(nod==n) continue;

for(int i : g[nod]){

if(c[nod][i]==f[nod][i] || v[i]) continue;

l.push(i); p[i]=nod; v[i]=1;

}

}

return v[n];

}

int32\_t main(){

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

in >> n >> m;

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

in >> x >> y >> z;

g[x].pb(y);

g[y].pb(x);

c[x][y]=z;

}

while(BFS()){

for(int i : g[n]){

if(c[i][n]==f[i][n] || !v[i]) continue;

int fl=1e9;

p[n]=i;

for(int j=n;j!=1;j=p[j]) fl=min(fl,c[p[j]][j]-f[p[j]][j]);

if(fl==0) continue;

for(int j=n;j!=1;j=p[j]) f[p[j]][j]+=fl,f[j][p[j]]-=fl;

ans+=fl;

}

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

}

out << ans;

}

**Eulerian Cycle**

int n,m,x,y,v[500005],vz[100005];

vector <pi> g[100005];

vector <int> ans[100005];

vector <int> q;

void Fill(int nod){

if(vz[nod]) return;

vz[nod]=1;

for(pi i : g[nod]) Fill(i.x);

}

void DFS(int nod){

q.push\_back(nod);

while(g[nod].sz && v[g[nod].back().y]) g[nod].pop\_back();

if(g[nod].sz){

v[g[nod].back().y]=1;

int nxt=g[nod].back().x;

g[nod].pop\_back();

DFS(nxt);

}

}

void Print(int s){

int t;

while(ans[s].sz){

t=ans[s].back();

ans[s].pop\_back();

s=t;

out << s << ' ';

}

}

int32\_t main(){

ios\_base :: sync\_with\_stdio(0); cin.tie(); cout.tie();

in >> n >> m;

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

in >> x >> y;

g[x].pb({y,i});

g[y].pb({x,i});

}

Fill(1);

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

if(!vz[i]){

out << -1;

return 0;

}

}

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

if(g[i].sz%2){

out << -1;

return 0;

}

}

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

while(g[i].sz){

q.clear();

DFS(i);

for(int i=q.sz-1;i>0;i--){

ans[q[i-1]].pb(q[i]);

}

}

}

Print(1);

}

**RMQ LCA**

vector<int> a[100005];

int b[400005],h[100005],sb=0;

int rmq[25][400005];

int n,m,x,y;

void Make(int nod){

b[++sb]=nod;

if(!h[nod]) h[nod]=sb;

for(int i=0;i<a[nod].size();i++){

Make(a[nod][i]);

b[++sb]=nod;

}

}

void BuildRMQ(){

for(int i=1;i<=sb;i++) rmq[0][i]=b[i];

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

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

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

}

}

}

int RMQ(int l, int r){

int k = (int)log2(r-l+1);

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

}

int main()

{

ifstream in("lca.in");

ofstream out("lca.out");

in >> n >> m;

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

in >> x;

a[x].push\_back(i);

}

Make(1);

//for(int i=1;i<=sb;i++) cout << b[i] << ' ';

BuildRMQ();

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

in >> x >> y;

if(h[x]<h[y]) out << RMQ(h[x],h[y]) << '\n';

else out << RMQ(h[y],h[x]) << '\n';

}

}

**Bellman-Ford**

int n,m,d[50005],x,y,z,v[50005];

vector <pi> g[50005];

int32\_t main(){

ios\_base :: sync\_with\_stdio(0); cin.tie(); cout.tie();

ifstream cin("bellmanford.in");

ofstream cout("bellmanford.out");

cin >> n >> m;

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

cin >> x >> y >> z;

g[x].pb({y,z});

}

vector <int> q,p;

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

q.pb(1); d[1]=0;

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

for(int j : q){

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

v[j]=i;

for(pi t : g[j]){

if(d[t.x]>d[j]+t.y) d[t.x]=d[j]+t.y,p.pb(t.x);

}

}

q=p;

p.clear();

}

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

for(pi j : g[i]){

if(d[j.x]>d[i]+j.y){

cout << "Ciclu negativ!";

return 0;

}

}

}

for(int i=2;i<=n;i++) cout << d[i] << ' ';

}

**Polygon Area**

int n;

double ans,x[100005],y[100005];

int32\_t main(){

ios\_base :: sync\_with\_stdio(0); cin.tie(); cout.tie();

ifstream cin("aria.in");

ofstream cout("aria.out");

cin >> n;

for(int i=0;i<n;i++) cin >> x[i] >> y[i];

for(int i=0;i<n;i++) ans+=(x[i]\*y[(i+1)%n]-x[(i+1)%n]\*y[i]);

ans/=2;

cout << fixed << setprecision(6) << abs(ans);

}

**Treap**

unordered\_map <int,bool> m;

struct nod{

nod \*l, \*r;

int v,p,cnt;

nod(int vt){

l=r=0;

v=vt;

cnt=1;

do{

p=rand()\*1000+rand();

}while(m[p]);

m[p]=1;

}

};

nod \*t = 0;

int get\_cnt(nod\* t){

if(t==0) return 0;

else return t->cnt;

}

void upd\_cnt(nod\* t){

if(t) t->cnt=1+get\_cnt(t->l)+get\_cnt(t->r);

}

void Split(nod\* t, nod\* &l, nod\* &r, int v){

if(t==0){

l=r=0;

return;

}

if(t->v > v) Split(t->l,l,t->l,v),r=t;

else Split(t->r,t->r,r,v),l=t;

upd\_cnt(t);

}

void Merge(nod\* &t, nod\* l, nod\* r){

if(l==0 || r==0) t=(l==0?r:l);

else if(l->p > r->p) Merge(l->r,l->r,r), t=l;

else Merge(r->l,l,r->l), t=r;

upd\_cnt(t);

}

void Insert(nod\* &t, nod\* nw){

if(t==0) t=nw;

else if(t->p < nw->p) Split(t,nw->l,nw->r,nw->v),t=nw;

else if(t->v < nw->v) Insert(t->r,nw);

else Insert(t->l,nw);

upd\_cnt(t);

}

void Erase(nod\* &t, int v){

if(t==0) return;

if(t->v == v) Merge(t,t->l,t->r);

else if(t->v > v) Erase(t->l,v);

else Erase(t->r,v);

upd\_cnt(t);

}

int Qry(nod\* t, int k){

if(t==0) return -1;

if(get\_cnt(t->l)+1==k) return t->v;

else if(get\_cnt(t->l)>=k) return Qry(t->l,k);

else return Qry(t->r,k-get\_cnt(t->l)-1);

}

**Treap2**

struct treap{

int v,k,cnt,h,l,r,p;

};

vector <treap> t;

void Init(){

t.push\_back({1000000007,0,0,0,-1,-1,-1});

srand(time(NULL));

}

void RotL(int nod){

int p=t[nod].p,k=t[nod].l,d=t[k].r;

if(t[p].l==nod)

t[p].l=k;

else

t[p].r=k;

t[k].p=p;

t[nod].p=k;

t[k].r=nod;

t[nod].l=d;

if(d!=-1) t[d].p=nod;

if(t[k].l!=-1) t[nod].h-=t[t[k].l].h;

t[nod].h--;

if(t[nod].r!=-1) t[k].h+=t[t[nod].r].h;

t[k].h++;

}

void RotR(int nod){

int p=t[nod].p,k=t[nod].r,d=t[k].l;

if(t[p].l==nod)

t[p].l=k;

else

t[p].r=k;

t[k].p=p;

t[nod].p=k;

t[k].l=nod;

t[nod].r=d;

if(d!=-1) t[d].p=nod;

if(t[k].r!=-1) t[nod].h-=t[t[k].r].h;

t[nod].h--;

if(t[nod].l!=-1) t[k].h+=t[t[nod].l].h;

t[k].h++;

}

void Balance(int nod){

if(t[nod].l!=-1 && t[t[nod].l].k<t[nod].k) RotL(nod);

else if(t[nod].r!=-1 && t[t[nod].r].k<t[nod].k) RotR(nod);

}

void Insert(int nod, int v, int k){

t[nod].h++;

if(v==t[nod].v){

t[nod].cnt++;

return;

}

if(v<t[nod].v){

if(t[nod].l==-1){

t.push\_back({v,k,1,1,-1,-1,nod});

t[nod].l=t.size()-1;

}

else Insert(t[nod].l,v,k);

}

if(v>t[nod].v){

if(t[nod].r==-1){

t.push\_back({v,k,1,1,-1,-1,nod});

t[nod].r=t.size()-1;

}

else Insert(t[nod].r,v,k);

}

Balance(nod);

}

bool Erase(int nod, int v){

if(v==t[nod].v){

if(t[nod].cnt>1){

t[nod].cnt--;

t[nod].h--;

return 1;

}

if(t[nod].l==-1 && t[nod].r==-1){

if(t[t[nod].p].l==nod) t[t[nod].p].l=-1;

else t[t[nod].p].r=-1;

return 1;

}

if(t[nod].l==-1) RotR(nod);

else if(t[nod].r==-1) RotL(nod);

else if(t[t[nod].l].k<t[t[nod].r].k) RotL(nod);

else RotR(nod);

t[t[nod].p].h--;

return Erase(nod,v);

}

else{

if(v>t[nod].v){

if(t[nod].r!=-1){

bool ret=Erase(t[nod].r,v);

if(ret) t[nod].h--;

return ret;

}

else return 0;

}

else{

if(t[nod].l!=-1){

bool ret=Erase(t[nod].l,v);

if(ret) t[nod].h--;

return ret;

}

else return 0;

}

}

}

bool Find(int nod, int v){

if(t[nod].v==v) return 1;

if(v>t[nod].v){

if(t[nod].r==-1) return 0;

else return Find(t[nod].r,v);

}

else{

if(t[nod].l==-1) return 0;

else return Find(t[nod].l,v);

}

}

int Kth(int nod, int k){

int x=0;

if(t[nod].l!=-1) x=t[t[nod].l].h;

if(x+1<=k && x+t[nod].cnt>=k) return t[nod].v;

else{

if(x+1>k) return Kth(t[nod].l,k);

else if(t[nod].r!=-1) return Kth(t[nod].r,k-x-t[nod].cnt);

else return -1;

}

}

void QryI(int v){

int p=rand()\*1000+rand()+1;

Insert(0,v,p);

}

void QryE(int v){

if(Erase(0,v)) t[0].h--;

}

bool QryF(int v){

return Find(0,v);

}

int QryK(int v){

return Kth(0,v);

}

**Treap3**

struct Treap{

Treap \*l,\*r;

int v,k,cnt;

Treap(){

v=0; k=0; l=0; r=0;

}

Treap(int v1, int k1, int cnt, Treap \*l1, Treap \*r1){

v=v1; k=k1;

l=l1; r=r1;

}

} \*T=new Treap();

void RotL(Treap \*&nod){

Treap \*kid=nod->l;

nod->l=kid->r;

kid->r=nod;

nod=kid;

}

void RotR(Treap \*&nod){

Treap \*kid=nod->r;

nod->r=kid->l;

kid->l=nod;

nod=kid;

}

void Balance(Treap \*&nod){

if(nod->l && nod->l->k<nod->k) RotL(nod);

else if(nod->r && nod->r->k<nod->k) RotR(nod);

}

void Insert(Treap \*&nod, int v, int p){

if(nod->v>v){

if(nod->l) Insert(nod->l,v,p);

else nod->l=new Treap(v,p,1,0,0);

}

else if(nod->v<v){

if(nod->r) Insert(nod->r,v,p);

else nod->r=new Treap(v,p,1,0,0);

}

else nod->k=p, nod->cnt++;

Balance(nod);

}

void QryE(Treap \*&nod, int v){

if(nod->v==v){

if(nod->cnt>1){

nod->cnt--;

return;

}

if(nod->)

if(nod->l){

RotL(nod);

QryE(nod,v);

}

else if(nod->r){

RotR(nod);

QryE(nod,v);

}

else{

nod=0;

}

}

else if(nod->v>v){

if(nod->l) QryE(nod->l,v);

}

else{

if(nod->r) QryE(nod->r,v);

}

}

void QryI(Treap \*&t, int v, int cnt=1){

int p=rand()\*30000+rand()+1;

if(t->k){

Insert(t,v,p);

}

else{

t=new Treap(v,p,cnt,0,0);

}

}

void Split(Treap \*&t, Treap \*&l, Treap \*&r, int v){

Insert(t,v,0);

l=t->l;

r=t->r;

if(t->cnt>1){

QryI(r,v,t->cnt-1);

}

}

void Join(Treap \*&t, Treap \*&l, Treap \*&r, int v){

t=new Treap(v,0,1,l,r);

QryE(t,v);

}

void Print(Treap \*nod,int n){

string s="";

for(int i=1;i<=n;i++) s+=' ';

cout << s;

cout << nod->v << '\n';

if(nod->l) cout << s << "L:", Print(nod->l,n+1);

if(nod->r) cout << s << "R:", Print(nod->r,n+1);

}

srand(time(0));

**Trie**

int n;

struct trie{

int cnt,f;

int k[26];

} gol;

vector <trie> t;

string s;

void Add(int nod, int p){

if(t[nod].k[s[p]-'a']==-1){

t.push\_back(gol);

t[nod].k[s[p]-'a']=t.size()-1;

}

t[t[nod].k[s[p]-'a']].cnt++;

if(p==s.lg-1){

t[t[nod].k[s[p]-'a']].f++;

return;

}

Add(t[nod].k[s[p]-'a'],p+1);

}

void Rem(int nod, int p){

t[t[nod].k[s[p]-'a']].cnt--;

if(p==s.lg-1){

t[t[nod].k[s[p]-'a']].f--;

return;

}

Rem(t[nod].k[s[p]-'a'],p+1);

}

int Cnt(int nod, int p){

if(t[nod].k[s[p]-'a']==-1) return 0;

if(p==s.lg-1) return t[t[nod].k[s[p]-'a']].f;

return Cnt(t[nod].k[s[p]-'a'],p+1);

}

int Pre(int nod, int p){

if(t[nod].k[s[p]-'a']==-1) return 0;

int ret=0;

if(t[t[nod].k[s[p]-'a']].cnt) ret=p+1;

if(p==s.lg-1) return ret;

return max(ret,Pre(t[nod].k[s[p]-'a'],p+1));

}

int32\_t main(){

ios\_base :: sync\_with\_stdio(0); cin.tie(); cout.tie();

for(int i=0;i<26;i++) gol.k[i]=-1; gol.cnt=0; gol.f=0;

**CHT (min value)**

#define eps 0.0000000000001

struct line{

ll a,b;

double x;

};

double Cross(line a, line b){

return (double)(a.b-b.b)/(b.a-a.a);

}

struct CHT{

struct comp1{

bool operator()(line a, line b){

if(a.a!=b.a) return a.a>b.a;

else return a.b<b.b;

}

};

struct comp2{

bool operator()(line a, line b){

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

else if(a.a!=b.a) return a.a>b.a;

else return a.b<b.b;

}

};

set <line,comp1> s1;

set <line,comp2> s2;

void Insert(ll a, ll b){

line p={a,b,0};

pair<set<line,comp1>::iterator,bool> f=s1.insert(p);

if(!f.y) return;

auto it=f.x;

//Remove Parallels

auto it1=it; it1++;

if(it1!=s1.end() && (\*it1).a==(\*it).a){

s2.erase(\*it1); s1.erase(it1++);

}

auto it2=it; it2--;

if(it!=s1.begin() && (\*it2).a==(\*it).a){

s1.erase(it);

return;

}

//Check if good

if(it!=s1.begin() && it1!=s1.end() && Cross(\*it2,\*it)-Cross(\*it,\*it1)>-eps){

s1.erase(it);

return;

}

while(1){

if(it==s1.begin()) break;

it1=it; it1--;

if(it1==s1.begin()) break;

it2=it1; it2--;

if(Cross(\*it2,\*it1)-Cross(\*it1,\*it)>-eps){

s2.erase(\*it1); s1.erase(it1);

}

else break;

}

while(1){

it1=it; it1++;

if(it1==s1.end()) break;

it2=it1; it2++;

if(it2==s1.end()) break;

if(Cross(\*it,\*it1)-Cross(\*it1,\*it2)>-eps){

s2.erase(\*it1); s1.erase(it1);

}

else break;

}

//Insert properly

it1=it; it1++; it2=it; it2--;

line temp=\*it;

if(it==s1.begin()) temp.x=-LINF;

else temp.x=Cross(\*it2,temp);

s1.erase(it); s1.insert(temp); s2.insert(temp);

if(it1!=s1.end()){

line temp2=\*it1;

temp2.x=Cross(temp,temp2);

s2.erase(\*it1); s1.erase(it1); s1.insert(temp2); s2.insert(temp2);

}

}

ll Qry(ll x){

auto it=s2.upper\_bound({0,0,(double)x}); it--;

return (\*it).a\*x+(\*it).b;

}

};

CHT h;

**CHT Non-dynamic (Max value, add in decreasing order) (Swap once for each change, don’t swap for both)**

int n;

struct d{

ll a,b,c;

}a[1000005];

bool comp(d a, d b){

if(a.a!=b.a) return a.a>b.a;

else return a.b<b.b;

}

#define eps 0.0000000000001

struct line{

ll a,b;

ld x;

};

ld Cross(line a, line b){

return (ld)(a.b-b.b)/(b.a-a.a);

}

struct CHT{

vector <line> s;

void Insert(ll a, ll b){

if(s.sz==0){

s.pb({a,b,-1.0});

}

else if(s.sz==1){

s.pb({a,b,-1.0});

s[1].x=Cross(s[0],s[1]);

}

else{

line nw={a,b,-1.0};

while(s.sz>=2){

if(Cross(s[s.sz-2],s[s.sz-1])-Cross(s[s.sz-1],nw)<eps) s.pop\_back();

// >-eps to swap

else break;

}

nw.x=Cross(s[s.sz-1],nw);

s.pb(nw);

}

}

ll Qry(ll x){

ll l=0,r=s.sz-1;

while(l!=r){

int mid=(l+r+1)/2;

if(s[mid].x-x<eps) r=mid-1;

// >-eps to swap

else l=mid;

}

return s[l].a\*x+s[l].b;

}

};

CHT h;

**Wavelet Trees**

struct nodwav{

int l,r,hi,lo;

vector <int> a,b;

} gol={-1,-1,0,INF,{},{}};

vector <nodwav> wav;

void GenWav(int nod){

int l=INF,r=0;

for(int i : wav[nod].a){

l=min(l,i);

r=max(r,i);

}

wav[nod].hi=r; wav[nod].lo=l;

if(l>=r) return;

int mid=(l+r)/2;

wav.push\_back(gol);

wav[nod].l=wav.size()-1;

wav.push\_back(gol);

wav[nod].r=wav.size()-1;

int cnt=0;

for(int i : wav[nod].a){

if(i<=mid) wav[wav[nod].l].a.push\_back(i),wav[nod].b.push\_back(++cnt);

else wav[wav[nod].r].a.push\_back(i),wav[nod].b.push\_back(cnt);

}

GenWav(wav[nod].l);

GenWav(wav[nod].r);

}

int QryWavS(int nod, int lt, int rt, int k){

if(lt>=rt) return 0;

int l=wav[nod].lo,r=wav[nod].hi;

if(l==r && l<=k) return rt-lt;

else if(l>=r) return 0;

int mid=(l+r)/2;

int x,y;

if(lt>=0) x=wav[nod].b[lt];

else x=0;

if(rt>=0) y=wav[nod].b[rt];

else y=0;

if(mid>=k)

return QryWavS(wav[nod].l,x-1,y-1,k);

else

return y-x+QryWavS(wav[nod].r,lt-x,rt-y,k);

}

int QryWavK(int nod, int lt, int rt, int p){

int l=wav[nod].lo,r=wav[nod].hi;

if(l==r) return l;

int x,y;

if(lt>=0) x=wav[nod].b[lt];

else x=0;

if(rt>=0) y=wav[nod].b[rt];

else y=0;

if(y-x>=p){

return QryWavK(wav[nod].l,x-1,y-1,p);

}

else{

return QryWavK(wav[nod].r,lt-x,rt-y,p-y+x);

}

}

int QryWavB(int nod, int lt, int rt, int p, int q){

if(lt>=rt) return 0;

int l=wav[nod].lo,r=wav[nod].hi;

if(l>r) return 0;

else if(l>=p && r<=q) return rt-lt;

else if(l>q || r<p) return 0;

else if(l==r) return 0;

int mid=(l+r)/2;

int x,y;

if(lt>=0) x=wav[nod].b[lt];

else x=0;

if(rt>=0) y=wav[nod].b[rt];

else y=0;

return QryWavB(wav[nod].l,x-1,y-1,p,q)+QryWavB(wav[nod].r,lt-x,rt-y,p,q);

}

int QWS(int l, int r, int v){

return QryWavS(0,l-2,r-1,v);

}

int QWK(int l, int r, int p){

return QryWavK(0,l-2,r-1,p);

}

int QWC(int l, int r, int p){

return QryWavS(0,l-2,r-1,p)-QryWavS(0,l-2,r-1,p-1);

}

int QWB(int l, int r, int p, int q){

return QryWavB(0,l-2,r-1,p,q);

}

int main(){

ios\_base :: sync\_with\_stdio(0); cin.tie();

wav.push\_back(gol);

wav[0].a={1,9,5,7,6,5,4,5,7};

GenWav(0);

cout << QWB(2,8,4,6) << '\n';

}

**FFT**

vector <complex<double>> RecFFT(vector <double> a){

int n=a.size();

if(n==1) return {{a[0],0}};

vector <double> a1,a2;

for(int i=0;i<n;i+=2) a1.push\_back(a[i]);

for(int i=1;i<n;i+=2) a2.push\_back(a[i]);

vector <complex<double>> x,y,ans;

x=RecFFT(a1);

y=RecFFT(a2);

//a(x)=a1(x^2)+x\*a2(x^2)

for(int i=0;i<n;i++) ans.push\_back({0,0});

complex <double> t={0,2\*acos(-1)/n},w={1,0};

t=exp(t);

for(int i=0;i<n/2;i++){

ans[i]=x[i]+w\*y[i];

ans[i+n/2]=x[i]-w\*y[i];

w\*=t;

}

return ans;

}

vector <complex <double>> InvFFT(vector <complex <double>> a){

int n=a.size();

if(n==1) return {{a[0],0}};

vector <complex<double>> x,y,ans;

for(int i=0;i<n;i+=2) x.push\_back(a[i]);

for(int i=1;i<n;i+=2) y.push\_back(a[i]);

x=InvFFT(x);

y=InvFFT(y);

for(int i=0;i<n;i++) ans.push\_back({0,0});

complex <double> t={0,-2\*acos(-1)/n},w={1,0};

t=exp(t);

for(int i=0;i<n/2;i++){

ans[i]=x[i]+w\*y[i];

ans[i+n/2]=x[i]-w\*y[i];

w\*=t;

}

return ans;

}

vector <double> DoubFFT(vector <double> a, vector <double> b){

int sa=a.size(),sb=b.size(),n=sa+sb-1;

int k=1;

while(k<n) k\*=2;

for(int i=sa;i<k;i++) a.push\_back(0);

for(int i=sb;i<k;i++) b.push\_back(0);

vector <complex<double>> x=RecFFT(a);

vector <complex<double>> y=RecFFT(b);

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

x[i]\*=y[i];

}

x=InvFFT(x); a.clear();

for(int i=0;i<sa+sb-1;i++) a.push\_back(real(x[i])/k);

return a;

}

vector <int> IntFFT(vector <int> a, vector <int> b){

int sa=a.size(),sb=b.size(),n=sa+sb-1;

int k=1;

while(k<n) k\*=2;

vector <double> p,q;

for(int i=0;i<sa;i++) p.push\_back(a[i]);

for(int i=0;i<sb;i++) q.push\_back(b[i]);

for(int i=sa;i<k;i++) p.push\_back(0);

for(int i=sb;i<k;i++) q.push\_back(0);

vector <complex<double>> x=RecFFT(p);

vector <complex<double>> y=RecFFT(q);

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

x[i]\*=y[i];

}

x=InvFFT(x); a.clear();

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

if(real(x[i])>0) a.push\_back((int)(real(x[i])/k+0.5));

else a.push\_back((int)(real(x[i])/k-0.5));

}

return a;

}

vector <long long> LongFFT(vector <long long> a, vector <long long> b){

int sa=a.size(),sb=b.size(),n=sa+sb-1;

int k=1;

while(k<n) k\*=2;

vector <double> p,q;

for(int i=0;i<sa;i++) p.push\_back(a[i]);

for(int i=0;i<sb;i++) q.push\_back(b[i]);

for(int i=sa;i<k;i++) p.push\_back(0);

for(int i=sb;i<k;i++) q.push\_back(0);

vector <complex<double>> x=RecFFT(p);

vector <complex<double>> y=RecFFT(q);

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

x[i]\*=y[i];

}

x=InvFFT(x); a.clear();

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

if(real(x[i])>0) a.push\_back((long long)(real(x[i])/k+0.5));

else a.push\_back((long long)(real(x[i])/k-0.5));

}

return a;

}

**Centroid Decomposition**

int n,x,y,s[200005],v[200005],p[200005][25],root,dist[200005];

vi g[200005];

void FindSize(int nod, int par){

s[nod]=1;

for(int i : g[nod]){

if(!v[i] && i!=par) FindSize(i,nod),s[nod]+=s[i];

}

}

int FindRoot(int nod, int par, int t){

for(int i : g[nod]){

if(!v[i] && i!=par && s[i]>t/2) return FindRoot(i,nod,t);

}

return nod;

}

void FillPar(int nod, int par, int depth, int root){

p[nod][depth]=root;

for(int i : g[nod]){

if(!v[i] && i!=par) FillPar(i,nod,depth,root);

}

}

void CD(int nod, int depth){

FindSize(nod,-1);

nod=FindRoot(nod,-1,s[nod]);

FillPar(nod,-1,depth,nod);

v[nod]=1;

if(!depth) root=nod;

for(int i : g[nod]){

if(!v[i]) CD(i,depth+1);

}

}