**Suffix Array**:

int wa[mx],wb[mx],wv[mx],Ws[mx];

//(1-indexed) sa[i] = starting position (0...n-1) of ith lexicographically smallest suffix in s

//(0-indexed) Rank[i] = lexicographical rank of s[i....n-1] ((i+1)th suffix by position)

//LCP[i] = longest common prefix of sa[i] & sa[i-1]

int sa[mx],Rank[mx],LCP[mx];

int cmp(int \*r,int a,int b,int l) {return r[a]==r[b] && r[a+l]==r[b+l];}

//Suffix Array (O(nlogn))

//m = maximum possible ASCII value of a string character (alphabet size)

//also, m = maximum number of distinct character in string (when compressed)

void buildSA(string s,int\* sa,int n,int m){

int i,j,p,\*x=wa,\*y=wb,\*t;

for(i=0; i<m; i++) Ws[i]=0;

for(i=0; i<n; i++) Ws[x[i]=s[i]]++;

for(i=1; i<m; i++) Ws[i]+=Ws[i-1];

for(i=n-1; i>=0; i--) sa[--Ws[x[i]]]=i;

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

for(p=0,i=n-j; i<n; i++) y[p++]=i;

for(i=0; i<n; i++) if(sa[i]>=j) y[p++]=sa[i]-j;

for(i=0; i<n; i++) wv[i]=x[y[i]];

for(i=0; i<m; i++) Ws[i]=0;

for(i=0; i<n; i++) Ws[wv[i]]++;

for(i=1; i<m; i++) Ws[i]+=Ws[i-1];

for(i=n-1; i>=0; i--) sa[--Ws[wv[i]]]=y[i];

for(t=x,x=y,y=t,p=1,x[sa[0]]=0,i=1; i<n; i++)

x[sa[i]]=cmp(y,sa[i-1],sa[i],j) ? p-1 : p++;

}

}

//Kasai's LCP algorithm (O(n))

void buildLCP(string s,int \*sa,int n){

int i,j,k=0;

for(i=1; i<=n; i++) Rank[sa[i]]=i;

for(i=0; i<n; LCP[Rank[i++]]=k)

for(k?k--:0, j=sa[Rank[i]-1]; s[i+k]==s[j+k]; k++);

return;}

// Pattern Subtring hisbe ace kina

bool Pattern(string &text,string &pat)

{ int lo=1,hi=text.size();

while(lo<=hi)

{ int mid=(lo+hi)/2;

int ok=0;

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

{ if(text[i+sa[mid]]>pat[i]) {ok=1;break;}

if(text[i+sa[mid]]<pat[i]) {ok=-1;break;}

}

if(!ok) return true;

if(ok>0) hi=mid-1;

else lo=mid+1;

}

return false;

}

pair<int,int> Patterntern\_occurence(string Text ,string Pattern)

{

int n=Text.size();

int m=Pattern.size();

int be=1,en=n;

while(be<en)

{ int mid = (en+be)/2;

int ok=0;

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

{ if(Text[i+sa[mid]]>Pattern[i]){ok=1;break;}

if(Text[i+sa[mid]]<Pattern[i]){ok=-1;break;}

}

if(ok+1) en=mid;

else be=mid+1;

}

bool ok = 1;

for(int i=0;i<m;i++) if(Text[i+sa[be]]!=Pattern[i]){ok=0;break;}

if(!ok) return {-1,-1};

pair<int,int> re;

re.first=be;

be=1,en=n;

while(be<en)

{ int mid = (en+be)/2;

int ok=0;

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

{ if(Text[i+sa[mid]]>Pattern[i]){ok=1;break;}

if(Text[i+sa[mid]]<Pattern[i]){ok=-1;break;}

}

if(ok>0) en=mid;

else be=mid+1;

}

ok = 1;

for(int i=0;i<m;i++) if(Text[i+sa[en]]!=Pattern[i]){ok=0;break;}

if(!ok) en--;

re.second=en;

return re;

}

/// this is for LCP from index i to index j.

/// just run a query from min(Rank[i-1],Rank[j-1])+1 to max(Rank[i-1],Rank[j-1])

int ST[mx][22];

int Jump\_LOG[mx];

void Build\_Sparse(int n)

{ for(int i=1;i<=n;i++)ST[i][0]=LCP[i];

for(int i=2;i<=n;i++)Jump\_LOG[i]=Jump\_LOG[i-1]+!(i&(i-1));

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

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

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

}

}

}

int query(int i,int j){

int boro\_lav=Jump\_LOG[j-i+1];

return min(ST[i][boro\_lav],ST[j-(1<<boro\_lav)+1][boro\_lav]);

}

void solve(){

buildSA(s,sa,n+1,130); //Important

buildLCP(s,sa,n);

for(int i=1;i<=n;i++) cout<<sa[i]<<" "; cout<<endl;

for(int i=0;i<n;i++) cout<<Rank[i]<<" "; cout<<endl;

for(int i=1;i<=n;i++) cout<<LCP[i]<<" ";

pair<int,int>re=Patterntern\_occurence(s,t);

if(re.second==-1)printf("0\n");

else printf("%d\n",re.second-re.first+1 );

}

**Aho Corasick:**

struct Aho\_Corasick{

int Trie[mx][27],Suffix\_Link[mx];

vector<int> Mark[mx];

int Node;

void Init() {

fill(Trie[0],Trie[0]+26,-1);

Mark[0].clear();

Node=0;

}

void Insert(char ch[],int idx) {

int len=strlen(ch);

int cur=0;

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

int val=ch[i]-'a';

if(Trie[cur][val]==-1) { Trie[cur][val]=++Node;

fill(Trie[Node],Trie[Node]+26,-1);

Mark[Node].clear();

}

cur=Trie[cur][val];

}

Mark[cur].push\_back(idx);

}

void Cal\_Suffix\_Link() {

queue<int>q;

Suffix\_Link[0]=0;

for(int i=0;i<26;i++){

if(Trie[0][i]!=-1){

q.push(Trie[0][i]);

Suffix\_Link[Trie[0][i]]=0;

}

else Trie[0][i]=0;

}

while(!q.empty()){

int u=q.front();

q.pop();

for(int v: Mark[Suffix\_Link[u]]){

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

}

for(int i=0;i<26;i++) {

if(Trie[u][i] != -1) {

Suffix\_Link[Trie[u][i]] = Trie[Suffix\_Link[u]][i];

q.push(Trie[u][i]);

}

else

Trie[u][i] = Trie[Suffix\_Link[u]][i];

}

}

}

}Automata;

/// Pattern Occurence Count

int cnt[mx];

void Count\_Pattern(char ch[]){

int cur=0;

int len=strlen(ch);

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

int val=ch[i]-'a';

cur= Automata.Trie[cur][val];

for(int id: Automata.Mark[cur])cnt[id]++;

}

}

void solve(){

char ch1[1000005],ch[mx];

scanf("%d%s",&n,ch1);

Automata.Init();

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

scanf("%s",ch);

Automata.Insert(ch,i);

}

Automata.Cal\_Suffix\_Link();

Count\_Pattern(ch1);

/// print Occurence Frequency

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

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

cnt[i]=0;

}

}

**Hashing:**

/\*backup prime 307,367,1040160883,1066517951

,1e9+7,1e9+9,1072857881,1000004249 \*/

struct Hash\_dui{

ll base,mod;

int sz;

vector<int>Rev,Forw,P;

Hash\_dui(){}

Hash\_dui(const char\* s,ll b,ll m)

{

sz=strlen(s),base=b,mod=m;

Rev.resize(sz+2,0),Forw.resize(sz+2,0),P.resize(sz+2,1);

for(int i=1;i<=sz;i++)P[i]=(base\*P[i-1])%mod;

for(int i=1;i<=sz;i++)Forw[i]=(Forw[i-1]\*base+(s[i-1]-'a'+1))%mod; /// digit hole s[i-1]-'0'

for(int i=sz;i>=1;i--)Rev[i]=(Rev[i+1]\*base+(s[i-1]-'a'+1))%mod; ///alphabet hole s[i-1]-'a'

}

void Single\_char\_ad(char cc) {

P.push\_back((P.back()\*base)% mod);

Forw.push\_back((Forw.back()\*base+(cc-'a'+1))% mod);

}

inline int Range\_Hash(int l,int r) {

int re\_hash=Forw[r+1]-((ll)P[r-l+1]\*Forw[l]%mod);

if(re\_hash<0)re\_hash+=mod;

return re\_hash;

}

inline int Reverse\_Hash(int l,int r) {

int re\_hash=Rev[l+1]-((ll)P[r-l+1]\*Rev[r+2]%mod);

if(re\_hash<0)re\_hash+=mod;

return re\_hash;

}

};

struct Hash\_Main

{

Hash\_dui h1,h2;

Hash\_Main(){}

Hash\_Main(const char\* s){

h1=Hash\_dui(s,1949313259, 2091573227);

h2=Hash\_dui(s,1997293877, 2117566807);

}

void Char\_Add(char cc){

h1.Single\_char\_ad(cc);

h2.Single\_char\_ad(cc);

}

inline ll Range\_Hash(int l,int r) /// O base index

{

return ((ll)h1.Range\_Hash(l,r)<<32)^h2.Range\_Hash(l,r);

}

inline ll Reverse\_Hash(int l,int r) /// O base index

{

return ((ll)h1.Reverse\_Hash(l,r)<<32)^h2.Reverse\_Hash(l,r);

}

};

void solve(){

int n;

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

string re=ch;

Hash\_Main h\_ek(ch);

ll h1=h\_ek(l,r)//0 base

}

**Manachers:**

int oddPlen[mx],evenPlen[mx];

void Manachers(){

int l=0,r=-1;

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

int k=(i>r)?1:min(oddPlen[l+r-i],r-i+1);

while(k<=i && i+k<n && ch[i-k]==ch[i+k]) k++;

oddPlen[i]=k--;

if(i+k>r){

l=i-k;

r=i+k;

}

}

l=0,r=-1;

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

int k=(i>r)?0:min(evenPlen[l+r-i+1],r-i+1);

while(k+1<=i && i+k<n && ch[i-k-1]==ch[i+k])k++;

evenPlen[i]=k--;

if(i+k>r){

l=i-k-1;

r=i+k;

}

}

}

void solve()

{ Manachers();

for(int i=0;i<n;i++)printf("%d %d\n",oddPlen[i]\*2-1,evenPlen[i]\*2);

}

**Pi Table / Prefix Functions:**

vector<int> Create\_Pi\_Table(const char\* s){

int sz=strlen(s);

vector<int>pi(sz);

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

int j=pi[i-1];

while(j>0 && s[i]!=s[j])j=pi[j-1];

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

pi[i]=j;

}

return pi;

}

void solve(){

vector<int> pi=Create\_Pi\_Table(ch);

for(int i=0;i<n;i++)printf("%d\n",pi[i] );

}

**HLD(value in edge):**

vector<pair<int,int>>g[mx];

int par[mx],sub\_sz[mx];

int Head[mx],st[mx],sesh[mx];

int Rin[mx]; /// Segment Tree er init ye Tree[bode]=ar[Rin[be]] likte hobe

int T;

using namespace Segment\_Tree;

void sz\_dfs(int u,int p){

sub\_sz[u]=1;

par[u]=p;

for(auto &v: g[u]){

if(v.first==p)continue;

sz\_dfs(v.first,u);

sub\_sz[u]+=sub\_sz[v.first];

if(sub\_sz[v]>sub\_sz[g[u][0].first])swap(v,g[u][0]);

}

}

void hld\_dfs(int u,int p,int cost){

st[u]=++T;

Rin[st[u]]=u;

ar[st[u]]=cost; /// node ye nai , sgement tree build array

for(auto v:g[u]) {

if(v.first==p)continue;

Head[v.first]= (v.first==g[u][0].first ? Head[u]:v.first);

hld\_dfs(v,u,v.second);

}

sesh[u]=T;

}

void hld\_build(int root){

T=0;

Head[root]=root;

sz\_dfs(root,root);

hld\_dfs(root,root,0);

}

bool Is\_it\_parent(int p,int u){

return st[p]<=st[u] && sesh[u]<=sesh[p];

}

int path\_query(int u,int v){

int re=-inf;

while(1){

if(Is\_it\_parent(Head[u],v))break;

re=max(re,query(1,1,n,st[Head[u]],st[u])); /// for sum we will do just add all query sum

u=par[Head[u]];

}

swap(u,v);

while(1){

if(Is\_it\_parent(Head[u],v))break;

re=max(re,query(1,1,n,st[Head[u]],st[u])); /// for sum we will do just add all query sum

u=par[Head[u]];

}

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

re=max(re,query(1,1,n,st[u]+1,st[v])); /// node hole st[u] theke start

return re;

}

void path\_update(int u,int v,int val){

while(1){

if(Is\_it\_parent(Head[u],v))break;

Rupdate(1,1,n,st[Head[u]],st[u],val);

u=par[Head[u]];

}

swap(u,v);

while(1){

if(Is\_it\_parent(Head[u],v))break;

Rupdate(1,1,n,st[Head[u]],st[u],val);

u=par[Head[u]];

}

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

Rupdate(1,1,n,st[u]+1,st[v],val); /// node hole st[u] theke start

}

void update\_subtree(int u,int val){

Rupdate(1,1,n,st[u]+1,sesh[u],val);

}

**1D Sparse Table:**

int ST[mx][MAX\_logN];

int Jump\_LOG[mx];

void Build\_Sparse(){

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

for(int i=2;i<=n;i++)Jump\_LOG[i]=Jump\_LOG[i-1]+!(i&(i-1));

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

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

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

}

}

}

int query(int i,int j){

int boro\_lav=Jump\_LOG[j-i+1];

return min(ST[i][boro\_lav],ST[j-(1<<boro\_lav)+1][boro\_lav]);

}

**2D Sparse (Rectangle):**

int ST[mx][mx][MAX\_logN][MAX\_logN];

void Build\_2D\_Sparse(){

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

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

ST[i][j][0][0]=ar[i][j];

}

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

int pre=1<<(l-1);

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

ST[i][j][0][l]=min(ST[i][j][0][l-1],ST[i][j+pre][0][l-1]);

}

}

}

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

int pre=1<<(l-1);

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

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

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

ST[i][j][l][k]=min(ST[i][j][l-1][k],ST[i+pre][j][l-1][k]);

}

}

}

}

}

int query(int i,int j,int p,int q) {

int boro\_jum1=log2(p-i+1);

int boro\_jum2=log2(q-j+1);

int pre1=1<<boro\_jum1;

int pre2=1<<boro\_jum2;

int re1=min(ST[i][j][boro\_jum1][boro\_jum2],ST[i][q-pre2+1][boro\_jum1][boro\_jum2]);

int re2=min(ST[p-pre1+1][j][boro\_jum1][boro\_jum2],ST[p-pre1+1][q-pre2+1][boro\_jum1][boro\_jum2]);

return min(re1,re2);

}

**2D Sparse (Square):**

int ST[mx][mx][MAX\_logN];

void Build\_2D\_Sparse(){

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

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

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

if(l==0)ST[i][j][l]=ar[i][j];

else{

int pre=1<<(l-1);

int val1=min(ST[i][j][l-1],ST[i+pre][j][l-1]);

int val2=min(ST[i][j+pre][l-1],ST[i+pre][j+pre][l-1]);

ST[i][j][l]=min(val1,val2);

}}}}}

int query(int i,int j,int sz){

int boro\_lav=log2(sz);

int pre=1<<(boro\_lav);

int val1=min(ST[i][j][boro\_lav],ST[i+sz-pre][j][boro\_lav]);

int val2=min(ST[i][j+sz-pre][boro\_lav],ST[i+sz-pre][j+sz-pre][boro\_lav]);

return min(val1,val2);

}

**MO:**

namespace MO{

const int N=100005;

const int Q=100005;

int ar[N],BlockId[N],ans[Q];

bool vis[N];

struct node {

int l,r,id;

node(){}

node(int l,int r,int id){

this->l=l;

this->r=r;

this->id=id;

}

bool operator < (const node& u) {

int a=BlockId[l],b=BlockId[u.l];

if(a==b){

return (a & 1 ? (r > u.r) : (r < u.r));

}

else return a<b;

}

}query[Q];

int boro=0;

int cnt[mx],cnt\_tot[mx];

void check(int pos) {

if(vis[pos]){

cnt\_tot[cnt[ar[pos]]]--;

cnt[ar[pos]]--;

if(cnt[ar[pos]])cnt\_tot[cnt[ar[pos]]]++;

if(cnt\_tot[boro]==0)boro--;

}

else{

if(cnt[ar[pos]])cnt\_tot[cnt[ar[pos]]]--;

cnt[ar[pos]]++;

cnt\_tot[cnt[ar[pos]]]++;

if(cnt\_tot[boro+1])boro++;

}

vis[pos]^=1;

}

}

using namespace MO;

void solve(){

int q;

boro=0;

scanf("%d%d",&n,&q);

int sz=sqrt(n);

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

BlockId[i]=i/sz;

vis[i]=false;

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

}

memset(cnt,0,sizeof(cnt));

memset(cnt\_tot,0,sizeof(cnt\_tot));

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

int x,y;

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

query[i]=node(x,y,i);

}

sort(query+1,query+q+1);

int left=query[1].l;

int right=left-1;

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

node Now=query[i];

while(left<Now.l)check(left++);

while(left>Now.l)check(--left);

while(right<Now.r)check(++right);

while(right>Now.r)check(right--);

ans[Now.id]=boro;

} }

**MO’s On tree:**

int n,m,ii,k,LOG;

int depth[mx];

int par[mx][25];

namespace MO{

const int N=100005;

const int Q=100005;

int ar[N],br[N],BlockId[N],ans[Q];

bool vis[N];

struct node {

int l,r,id,lca;

node(){}

node(int l,int r,int lca,int id){

this->l=l;

this->r=r;

this->lca=lca;

this->id=id;

}

bool operator < (const node& u) {

int a=BlockId[l],b=BlockId[u.l];

return (a==b)?(r<u.r):a<b;

}

}query[Q];

int re=0,sz;

int cnt[100005];

void check(int pos) {

if(vis[pos]){

if(cnt[ar[pos]]==1)re--;

cnt[ar[pos]]--;

}

else{

if(cnt[ar[pos]]==0)re++;

cnt[ar[pos]]++;

}

vis[pos]^=1;

}

vector<int> g[N];

int Euler[2\*N],st[N],en[N],Time;

void dfs(int u,int p,int lvl) {

st[u]=++Time;

Euler[Time]=u;

par[u][0]=p;

depth[u]=lvl;

for(int v:g[u]) {

if(v==p)continue;

dfs(v,u,lvl+1);

}

en[u]=++Time;

Euler[Time]=u;

}

}

using namespace MO;

void init(int root){

dfs(root,-1,1);

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

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

if(par[i][j-1]!=-1) {

par[i][j]=par[par[i][j-1]][j-1];

}

else par[i][j]=-1;

}

}

}

int lca(int u,int v){

if(depth[u]<depth[v])swap(u,v);

int log=1;

while(1) {

int next=log+1;

if(depth[u]<(1<<next))break;

log++;

}

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

if(depth[u]-(1<<i)>=depth[v]) {

u=par[u][i];

}

}

if(u==v)return u;

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

if(par[u][i]!=-1 && par[u][i]!=par[v][i]) {

u=par[u][i];

v=par[v][i];

}

}

return par[v][0];

}

void solve(){

int q;

scanf("%d%d",&n,&q);

LOG=log2(n)+1;

Time=0;

re=0;

sz=sqrt(n);

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

scanf("%d",&ar[i]),br[i]=ar[i],BlockId[i]=i/sz,vis[i]=false,cnt[i]=0;

// Compressing Coordinates . its a alternative of map

sort(br+1,br+n+1);

k = unique(br+1,br+n+1)-br-1;

for(int i=1;i<=n;i++) ar[i]=lower\_bound(br+1,br+k+1,ar[i])-br;

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

int x,y;

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

g[x].push\_back(y);

g[y].push\_back(x);

}

init(1);

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

int x,y;

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

if(st[x]>st[y])swap(x,y);

int p=lca(x,y);

if(x==p)query[i]=node(st[x],st[y],-1,i);

else query[i]=node(en[x],st[y],p,i);

}

sort(query+1,query+1+q);

int left=query[1].l;

int right=left-1;

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

node Now=query[i];

while(left<Now.l)check(Euler[left++]);

while(left>Now.l)check(Euler[--left]);

while(right<Now.r)check(Euler[++right]);

while(right>Now.r)check(Euler[right--]);

if(Now.lca!=-1)check(Now.lca);

ans[Now.id]=re;

if(Now.lca!=-1)check(Now.lca);

}

for(int i=1;i<=q;i++)printf("%d\n",ans[i]);

for(int i=1;i<=n;i++)g[i].clear();

}

**Trie (max min xor subarray):**

int Trie[mx][2];

int End[mx];

int ar[50005];

int Trie[50000\*32][2];

int n,ii,st=1;

void Insert(int val){

int cur=1;

for(int i=31;i>=0;i--){

int bit=0;

if(((1<<i) & val))bit=1;

if(Trie[cur][bit]==0)Trie[cur][bit]=++st;

cur=Trie[cur][bit];

}

End[cur]=val;

}

int query\_min(int val){

int cur=1;

for(int i=31;i>=0;i--) {

int bit=0;

if(((1<<i) & val))bit=1;

if(Trie[cur][bit])cur=Trie[cur][bit];

else if(Trie[cur][bit^1])cur=Trie[cur][bit^1];

}

return End[cur]^val;

}

int query\_max(int val){

int cur=1;

for(int i=31;i>=0;i--){

int bit=0;

if(((1<<i) & val))bit=1;

if(Trie[cur][bit^1])cur=Trie[cur][bit^1];

else if(Trie[cur][bit])cur=Trie[cur][bit];

}

return End[cur]^val;

}

void solve(){

int suffix=0;

int re\_min=INT\_MAX,re\_max=0;

Insert(0);

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

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

suffix^=ar[i];

re\_min=min(re\_min,query\_min(suffix));

re\_max=max(re\_max,query\_max(suffix));

Insert(suffix);

}

}

**SegTree :**

**Bracket Sequence:**

struct info{

int open,close,ans;

};

info Merge(info a,info b){

info re;

int valid=min(a.open,b.close);

re.open=a.open+b.open-valid;

re.close=a.close+b.close-valid;

re.ans=a.ans+b.ans+valid; /// this code works for maximum length of correct bracket sequence in l to r range

return re;

}

**Kth element merge sort tree:**

int query(int node,int be,int en,int l,int r,int k){

if(be==en)return seg[node][0];

int pos =upper\_bound(seg[node\*2+1].begin(),seg[node\*2+1].end(),r)

-lower\_bound(seg[node\*2+1].begin(),seg[node\*2+1].end(),l);

int mid=(be+en)/2;

if(pos>=k) {

return query(node\*2+1,be,mid,l,r,k);

}

else return query(node\*2+2,mid+1,en,l,r,k-pos);

}

**Delete Type Id Found:**

int id\_query(int node,int be,int en,int pos){

if(be==en)return be;

int mid=(be+en)/2;

if(Present[node\*2]>=pos){

return id\_query(node\*2,be,mid,pos);

}

else return id\_query(node\*2+1,mid+1,en,pos-Present[node\*2]);

}

**Rang max subarray / suffix-prefix sum:**

struct info{

ll max\_pref,max\_suf,ans,sum;

void Merge(info p1,info p2){

sum=p1.sum+p2.sum;

max\_pref=max(p1.max\_pref,p1.sum+p2.max\_pref); max\_suf=max(p2.max\_suf,p2.sum+p1.max\_suf); ans=max(max(p1.ans,p2.ans),p1.max\_suf+p2.max\_pref);

}};

void Relax(int node,int be,int en){

if(!cur[node])return;

Tree[node].sum=Lazy[node]\*(en-be+1);

Tree[node].max\_pref=max(0LL,Tree[node].sum);

Tree[node].max\_suf=max(0LL,Tree[node].sum);

Tree[node].ans=max(0LL,Tree[node].sum);

if(be!=en){

Lazy[node\*2]=Lazy[node];

Lazy[node\*2+1]=Lazy[node];

cur[node\*2]=true;

cur[node\*2+1]=true;

}

cur[node]=false;

Lazy[node]=0;

}

**Centroid Decomposition:**

int dis[18][mx],re[mx],vis[mx];

int p[mx],sub[mx],lvl[mx];

vector<int>g[mx],ng[mx];

/\* p[u] = parent of u in centroid tree

dis[x][u] = distance from u to a parent of u at level x of centroid tree

if u is in subtree of centroid c, then dis[lvl[c]][u] = dist(c, l)

If (x, y) edge exist, then x must be in g[y] and y must be in g[x]\*/

/\* we can do more pre work in dfs function\*/

void dfs(int l,int u,int par){

if(par!=-1)dis[l][u]=dis[l][par]+1;

for(int v:g[u])

if(v!=par && !vis[v])dfs(l,v,u);

}

int centroid(int u,int par,int r){

for(int v:g[u])

if(v!=par && !vis[v] && sub[v]>r)return centroid(v,u,r);

return u;

}

void pre\_cal(int u,int par){

sub[u]=1;

for(int v:g[u])

if(v!=par && !vis[v])pre\_cal(v,u),sub[u]+=sub[v];

}

void decompose(int u,int par){

pre\_cal(u,-1);

int tem=centroid(u,-1,sub[u]>>1);

vis[tem]=1,p[tem]=par,lvl[tem]=0;

if(par!=-1)lvl[tem]=lvl[par]+1,ng[par].push\_back(tem);

dfs(lvl[tem],tem,-1);

for(int v:g[tem])

if(v!=par && !vis[v])decompose(v,tem);

}

void update(int u){

for(int v=u;v!=-1;v=p[v])re[v]=min(re[v],dis[lvl[v]][u]);

}

int query(int u){

int ans=1e9;

for(int v=u;v!=-1;v=p[v])

ans=min(ans,re[v]+dis[lvl[v]][u]);

return ans;

}

int lca(int u,int v){

if(lvl[u]<lvl[v])swap(u,v);

while(lvl[u]>lvl[v])u=p[u];

while(u!=v && p[u]!=-1)u=p[u],v=p[v];

return u;

}

int dist(int u,int v){

int lc=lca(u,v);

return dis[lvl[lc]][u]+dis[lvl[lc]][v];

}

int GetRoot(int u){

while(p[u]!=-1)u=p[u];

return u;

}

/// for all pair

void update(int u,int p){

int val=dis[lvl[p]][u];

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

cnt[i][chk(val,i)]++;

}

for(int v:ng[u])update(v,p);

}

void query(int u,int p){

int val=dis[lvl[p]][u]^ar[p];

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

ans+=cnt[i][!chk(val,i)]\*(1LL<<i);

}

for(int v:ng[u])query(v,p);

}

void Go\_Ahead(int u){

memset(cnt,0,sizeof(cnt));

for(int i=0;i<20;i++)cnt[i][chk(ar[u],i)]++;

for(int v:ng[u]){

query(v,u);

update(v,u);

}

ans+=ar[u];

for(int v:ng[u])Go\_Ahead(v);

}

// at first call decompose(1,-1)

**Dinic:**

const ll eps = 0;

struct edge {

int a, b;

ll cap,flow;

int yo, x, y;

};

struct Dinic {

int s,t,d[mx], ptr[mx] ;

//int Id[mx][mx];

vector<edge>e;

vector<int>g[mx];

void init() {

e.clear();

memset(d,0,sizeof(d));

for(int i = 0; i < mx ; i++)g[i].clear();

// for(int i=0;i<mx;i++)

// {

// for(int j=0;j<mx;j++)

// {

// Id[i][j]=0;

// }

// }

}

void addEdge(int a,int b,ll cap, int x = -1, int y= -1) {

edge e1 = { a, b, cap, 0, 1, x, y } ;

edge e2 = { b, a, 0, 0, 0, x, y } ;

// Id[a][b]=e.size();

g[a].push\_back((int)e.size());

e.push\_back(e1);

// Id[b][a]=e.size();

g[b].push\_back((int)e.size());

e.push\_back(e2);

}

bool bfs() {

queue < int > Q ;

Q.push(s);

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

d[s]=0 ;

while (!Q.empty()) {

int u=Q.front() ;

Q.pop() ;

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

int id=g[u][i];

int v=e[id].b;

// printf("%d %d %0.3lf %0.3lf\n",u,v,e[id].cap,e[id].flow);

if(d[v]==-1&&e[id].flow<e[id].cap) {

Q.push(v) ;

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

}} }

return d[t]!=-1 ;

}

ll dfs(int u,ll flow) {

if (flow<=eps) return 0 ;

if ( u==t ) return flow ;

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

int id = g[u][i];

int v = e[id].b ;

if ( d[v] != d[u]+1 ) continue ;

ll pushed = dfs (v,min (flow,e[id].cap-e[id].flow)) ;

//cout << "pushed " << pushed << endl;

if (pushed>eps) {

e [id].flow+=pushed ;

e [id^1].flow-=pushed ;

return pushed ;

}

}

return 0 ;

}

ll dinic() {

ll flow = 0 ;

while(true) {

if(!bfs()) break ;

memset(ptr, 0, sizeof(ptr)) ;

while (true){

ll pushed = dfs(s,INF );

if(pushed<=eps)break;

flow += pushed ;

}

}

return flow ;

}

};

**Upper Lower Limit:**

Dinic dc;

int x,y;

struct tem{

int u,v,a,b;

};

vector<tem>ed;

ll func(ll val){

dc.init();

dc.s=n+1;

dc.t=n+2;

/// for upperbound(0,val)

// dc.addEdge(y,n+3,val); /// sink to super super source

// dc.addEdge(n+1,x,0); /// sink to source

// dc.addEdge(n+3,n+2,0); /// super super source to super sink

// dc.addEdge(n+3,x,val); /// super super source to source

/// for lowerbound(val,inf)

dc.addEdge(y,n+3,INF); /// sink to super super source

dc.addEdge(n+1,x,val); /// sink to source

dc.addEdge(n+3,n+2,val); /// super super source to super sink

dc.addEdge(n+3,x,INF); /// super super source to source

for(auto it:ed){

dc.addEdge(n+1,it.v,it.a);

dc.addEdge(it.u,n+2,it.a);

dc.addEdge(it.u,it.v,it.b-it.a);

}

return dc.dinic();

}

void solve(){

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

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

dc.addEdge(y,x,INF);

dc.s=n+1;

dc.t=n+2;

ll val=0;

ll en=0;

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

int u,v,a,b;

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

ed.push\_back({u,v,a,b});

val+=a;

en+=b;

dc.addEdge(n+1,v,a);

dc.addEdge(u,n+2,a);

dc.addEdge(u,v,b-a);

}

if(dc.dinic()<val){

printf("0\n");

return;

}

ll be=val;

ll re=be;

while(be<=en){

ll mid=(be+en)/2;

ll have=func(mid);

if(have>=mid+val) {

re=mid;

be=mid+1;

}

else en=mid-1;

}

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

}

**Hopcroft\_Karp:**

#define mx 40005

#define INF (1<<28)

struct Hopcroft\_Karp

{

vector< int > g[mx];

int n, m, Matching[mx], Distance[mx];

// 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

void init(int num){

for(inti=0;i<=num;i++)Matching[i]=0,Distance[i]=0,g[i].clear();

}

void addEdge(int u,int v) {

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

}

bool bfs() {

int i, u, v, len;

queue< int > q;

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

if(Matching[i]==0) {

Distance[i] = 0;

q.push(i);

}

else Distance[i] = INF;

}

Distance[0] = INF;

while(!q.empty()) {

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

if(u!=0) {

for(int v:g[u]) {

if(Distance[Matching[v]]==INF) {

Distance[Matching[v]] = Distance[u] + 1;

q.push(Matching[v]);

}

}

}

}

return (Distance[0]!=INF);

}

bool dfs(int u) {

int i, v, len;

if(u!=0) {

for(int v:g[u]) {

if(Distance[Matching[v]]==Distance[u]+1) {

if(dfs(Matching[v])) {

Matching[v] = u;

Matching[u] = v;

return true;

}

}

}

Distance[u] = INF;

return false;

}

return true;

}

int hopcroft\_karp() {

int Matchinging = 0, i;

while(bfs())

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

if(Matching[i]==0 && dfs(i))

Matchinging++;

return Matchinging;

}

};

Hopcroft\_Karp hk;

**Hungarian (visit all node with minimum cost):**

#define INF 1e18

pair<ll,vector<int>> hungarian(vector<vector<ll>>mat,int f,int sz){

vector<int>par(sz+1,0),way(sz+1,0),match(sz+1,0);

vector<bool>vis(sz+1,0);

vector<ll>U(sz+1,0),V(sz+1,0),MinV(sz+1,0);

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

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

mat[i][j]\*=f;

}

}

int a,b,d;

ll r,w;

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

par[0]=i;

b=0;

for(int j=1;j<=sz;j++)MinV[j]=INF,vis[j]=0;

do{

vis[b]=1;

a=par[b],d=0,w=INF;

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

if(!vis[j]) {

r=mat[a][j]-U[a]-V[j];

if(r<MinV[j])MinV[j]=r,way[j]=b;

if(MinV[j]<w)w=MinV[j],d=j;

}

}

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

if(vis[j])U[par[j]]+=w,V[j]-=w;

else MinV[j]-=w;

}

b=d;

}

while(par[b]!=0);

do{

d=way[b];

par[b]=par[d],b=d;

}

while(b!=0);

}

for(int j=1;j<=sz;j++)match[par[j]]=j;

return {-f\*V[0],match};

}

// called hungarain(mat,1,n)

**Min Cost Max Flow:**

typedef long long T1;//for cost

typedef long long T2;//for flow

const int maxn = 20100;

const T1 INF = 1e12;

const T2 inf = 1e12;

const T1 eps = 0;

struct Edge {

int from, to;

T2 cap, flow;

T1 cost;

};

int n,m,k,ii;

struct MCMF {//0-indexed

int n, m, s, t;

vector<Edge> edges;

vector<int> G[maxn];

int p[maxn],inq[maxn];

T1 d[maxn];

T2 a[maxn];

void init() {

for(int i = 0; i < n; i++) G[i].clear();

edges.clear();

}

void AddEdge(int from,int to,T2 cap,T1 cost) {

edges.push\_back((Edge){from, to, cap, 0, cost});

edges.push\_back((Edge){to, from, 0, 0, -cost});

m = edges.size();

G[from].push\_back(m-2);

G[to].push\_back(m-1);

}

pair<T1,T2> Mincost() {//bellmanFord

T1 tot\_cost = 0;

T2 tot\_flow = 0;

while(true) {

for(int i = 0; i < n; i++) d[i] = INF;

d[s] = 0;

p[s] = 0;

a[s] = inf;

bool up=true;

while(up) {

up=false;

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

if(d[u]-INF>=-eps)continue;

for(int j:G[u]) {

Edge &e=edges[j];

if(e.cap > e.flow && d[e.to] > d[u] + e.cost+eps) {

d[e.to] = d[u] + e.cost;

p[e.to] = j;

a[e.to] = min(a[u], e.cap - e.flow);

up=true;

}

}

}

}

if(abs(d[t]-INF)<=eps)break;

tot\_cost += (T1)d[t] \* a[t];

tot\_flow += (T2)a[t];

int u = t;

while(u != s) {

edges[p[u]].flow += a[t];

edges[p[u]^1].flow -= a[t];

u = edges[p[u]].from;

}

}

return {tot\_cost,tot\_flow};

}

pair<T1,T2> Mincost2() {//SPFA

T1 tot\_cost = 0;

T2 tot\_flow = 0;

while(true) {

for(int i = 0; i < n; i++) d[i] = INF;

memset(inq, 0, sizeof(inq));

d[s] = 0;

inq[s] = 1;

p[s] = 0;

a[s] = inf;

queue<int> Q;

srand(time(NULL));

Q.push(s);

while(!Q.empty()) {

int u = Q.front();

Q.pop();

inq[u] = 0;

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

Edge& e = edges[G[u][i]];

if(e.cap > e.flow && d[e.to] > d[u] + e.cost+eps) {

d[e.to] = d[u] + e.cost;

p[e.to] = G[u][i];

a[e.to] = min(a[u], e.cap - e.flow);

if(!inq[e.to]) {

Q.push(e.to);

inq[e.to] = 1;

}

}

}

}

if(abs(d[t]-INF)<=eps)break;

tot\_cost += (T1)d[t] \* a[t];

tot\_flow += a[t];

int u = t;

while(u != s) {

edges[p[u]].flow += a[t];

edges[p[u]^1].flow -= a[t];

u = edges[p[u]].from;

}

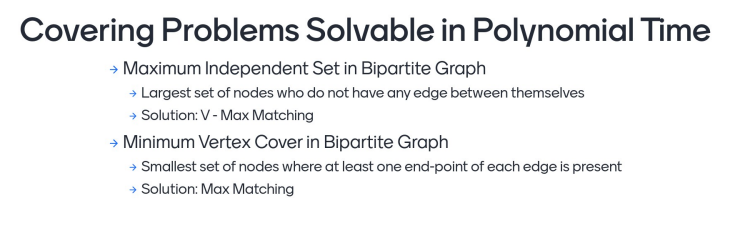
}

return {tot\_cost,tot\_flow};

}

} mcmf;





**Kuhn:**

struct BPM{

bool Done[mx];

vector<int>g[mx];

int macth[mx];

void addEdge(int u,int v) {

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

}

void init() {

for(int i=0;i<mx;i++)g[i].clear();

}

bool Tem\_Matching(int u){

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

int v=g[u][i];

if(Done[v]) continue;

Done[v] = true;

if(macth[v]==-1 || Tem\_Matching(macth[v])) {

macth[v] = u;

return true;

}

}

return false;

}

int Max\_Matching(int num) {

// Be Careful with this section. when passin num.

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

int re = 0;

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

memset(Done,false,sizeof(Done));

if(Tem\_Matching(i)) re++;

}

return re;

}

};

**LCA(value on edge):**

int par[mx][20];

ll ans[mx][20];

int depth[mx],LOG;

vector<pair<int,ll>>g[mx];

void dfs(int u,int p,int lvl){

par[u][0]=p;

depth[u]=lvl;

for(auto it:g[u]) {

int v=it.first;

ll w=it.second;

if(v==p)continue;

ans[v][0]=w;

dfs(v,u,lvl+1);

}

}

void init(int root){

dfs(root,-1,1);

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

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

if(par[i][j-1]!=-1){

par[i][j]=par[par[i][j-1]][j-1];

ans[i][j]=max(ans[i][j-1],ans[par[i][j-1]][j-1]);

}

else par[i][j]=-1;

}

}

}

ll query(int u,int v){

if(u==v)return 0;

if(depth[u]<depth[v])swap(u,v);

int diff=depth[u]-depth[v];

ll re=0;

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

if(diff>=(1<<i)) {

diff-=(1<<i);

re=max(re,ans[u][i]);

u=par[u][i];

}

}

if(u==v)return re;

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

if( par[u][i]!=par[v][i]){

re=max({re,ans[u][i],ans[v][i]});

u=par[u][i];

v=par[v][i];

}

}

re=max({re,ans[u][0],ans[v][0]});

return re;

}

int dist(int u,int v){

return depth[u]+depth[v]-2\*depth[lca(u,v)];

}

int kth\_parent(int u,int k){

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

if(k>=(1<<i)) {

k-=(1<<i);

u=par[u][i];

}

if(u==-1)return u;

}

return u;

}

solve(){

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

g[i].clear();

for(int j=0;j<LOG;j++)ans[i][j]=0,par[i][j]=-1;

}

LOG=log2(n)+1;

}

**LCA(value in node):**

//dfs function ye ans[u][0] line likha jabe nah

// init function same

// query function er sesh ye ei 3 line likhbo

re=max(re,ans[u][0]);

re=max(re,ans[v][0]);

re=max(re,ans[par[v][0]][0]);

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

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

ans[i][0]=ar[i];

}

**DSU:**

int Size[mx];

int Findparent(int x){

return (x==parent[x])?x:(parent[x]=Findparent(parent[x]));

}

void Union(int x,int y){

int px=Findparent(x);

int py=Findparent(y);

if(px==py)return;

if(Size[px]>Size[py]) {

Size[px]+=Size[py];

parent[py]=px;

}

else {

Size[py]+=Size[px];

parent[px]=py; }

}

void initialize(){

for(int i=0;i<=n;i++)parent[i]=i,Size[i]=1;

}

**Bellman Ford:**

vector<Edge>E;

ll dist[100];

bool bellman\_ford(){

/// here i can start from 1 .if given that stating node i can set dist[src]=0

for(int i=1;i<=n;i++)dist[i]=10000000;

dist[1]=0;

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

for(Edge it: E)

if(dist[it.v]>dist[it.u]+it.w)

dist[it.v]=dist[it.u]+it.w;

for(Edge it:E)

if(dist[it.v]>dist[it.u]+it.w)return true;//negative cycle

return false;

}

**Floyed Warshal:**

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

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

if(i==j || dis[i][j]>0)continue;

dis[i][j]=1e18;

}

}

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

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

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

dis[i][j]=min(dis[i][j],dis[i][l]+dis[l][j]);}}}

**Articulation Point:**

vector<int>g[mx];

int articular\_point[mx];

int st[mx],low[mx];

int Time=1;

int dfs(int u,int p){

st[u]=low[u]=Time++;

int child=0;

for(auto it:g[u]) {

if(it==p)continue;

if(st[it]==0) {

child++;

dfs(it,u);

if(st[u]<=low[it])articular\_point[u]=1;

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

}

else low[u]=min(low[u],st[it]);

}

return child;

}

void solve(){

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

if(st[i])continue;

articular\_point[i]=(dfs(i,-1)>1);

}

}

**Articulations Bridge:**

vector<int>g[mx];

vector<pair<int,int>>Bridge;

int st[mx],low[mx];

int Time=1;

void dfs(int u,int p){

st[u]=low[u]=Time++;

int child=0;

for(auto it:g[u]) {

if(it==p)continue;

if(st[it]==0){

dfs(it,u);

if(st[u]<low[it])Bridge.push\_back({u,it});

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

}

else low[u]=min(low[u],st[it]);

}

}

void solve(){

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

if(st[i])continue;

dfs(i,-1);

}

}

**Strongly Connected Component:**

vector<int>g[mx],g\_rev[mx],st(mx),en(mx),component[mx],option,visit;

vector<pair<int,int>>dekhi;

int node,edge,cnt,tem;

int mp[mx];

void dfs1(int u){

visit[u]=true;

st[u]=++cnt;

for(auto it:g[u]) {

if(visit[it])continue;

dfs1(it);

}

en[u]=++cnt;

}

void dfs2(int u){

visit[u]=true;

component[cnt].push\_back(u);

for(auto it:g\_rev[u]) {

if(visit[it])continue;

dfs2(it);

}

}

void clean(){

for(int i=1;i<=node+2;i++) {

g[i].clear();

g\_rev[i].clear();

component[i].clear();

}

option.clear();

cnt=0;

st.clear();

en.clear();

dekhi.clear();

memset(mp,0,sizeof(mp));

}

void solve(){

scanf("%d%d",&node,&edge);

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

int u,v;

scanf("%d%d",&u,&v);///directed graph

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

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

mp[u]++;

mp[v]++;

}

visit.assign(node+2,false);

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

if(visit[i]==true || mp[i]==0)continue;

dfs1(i);

}

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

if(visit[i]==true && mp[i])dekhi.push\_back({en[i],i});

}

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

reverse(dekhi.begin(),dekhi.end());

visit.assign(node+2,false);

cnt=1;

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

int pos=dekhi[i].second;

if(visit[pos] || mp[pos]==0)continue;

dfs2(pos);

cnt++;

}

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

for(auto it:component[i]) {

cout<<it<<" ";

}

cout<<endl;

}}

**Matrix Expo:**

#define MAX 105

#define ll long long int

const ll MOD = 1e9 + 7;

const ll MOD2 = MOD \* MOD \* 3;

inline ll bigMod(ll a,ll b){

ll res=1;

while(b){

if(b&1) res=(res\*a)%MOD;

a=(a\*a)%MOD; b>>=1;

}

return res;

}

inline ll inv(ll n) {return bigMod(n,MOD-2);}

inline ll Mul(ll a,ll b) {return (a\*b)%MOD;}

inline ll Div(ll a,ll b) {return Mul(a,inv(b));}

/\* 1 base row columun index \*/

struct Matrix{

int row, col;

ll m[MAX][MAX];

Matrix() {memset(m,0,sizeof(m));}

void Set(int r,int c) {row = r; col = c;}

Matrix(int r,int c) {memset(m,0,sizeof(m)); Set(r,c);}

void normalize(){

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

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

m[i][j] %= MOD;

if(m[i][j] < 0) m[i][j] += MOD;

}

}

}

};

Matrix Multiply(Matrix A,Matrix B){

Matrix ans(A.row,B.col);

for(int i=1;i<=A.row;i++){

for(int j=1;j<=B.col;j++){

ans.m[i][j]=0;

ll sm = 0;

for(int k=1;k<=A.col;k++){

sm+=(A.m[i][k]\*B.m[k][j]);

if(sm >= MOD2) sm -= MOD2;

}

ans.m[i][j] = sm % MOD;

}

}

return ans;

}

Matrix Power(Matrix mat,ll p){

Matrix res(mat.row , mat.col);

Matrix ans(mat.row , mat.col);

int n = ans.row;

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

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

ans.m[i][j]=0;

res.m[i][j]=mat.m[i][j];

}

ans.m[i][i]=1;

}

while(p){

if(p&1) ans=Multiply(ans,res);

res=Multiply(res,res);

p=p/2;

}

return ans;

}

**// Gaussian Elimination Offline**

ll a[MAX], n; //0 base index

ll maxxor(){

int r = 0; ll ret = 0;

for(int c = 63; c >= 0; c--){

int idx = -1;

for(int i = r; i < n && idx < 0; i++)

if(a[i] >> c & 1) idx = i;

if(idx == -1) continue;

swap(a[r], a[idx]);

for(int i = 0; i < n; i++) if(i != r)

if(a[i] >> c & 1) a[i] ^= a[r];

r++;

}

for(int i = 0; i < n; i++) ret = max(ret, ret ^ a[i]);

return ret;

}

**Inclusion Exclusion:**

/// koto gulo number ace[1,n] jara a1 or a2 or a3...,am dara divide

/// if m=3 and 3 values are a1,a2,a3 then

/// |a1 U a2 U a3|=|a1|+|a2|+|a3|-|a1 union a2|-|a2 union a3|-|a1 union a3|+|a1 union a2 union a3|

/// if number of cadidate is odd do add or do substract

/// time complexity 2^m.

/// for better perform use recusive

void func(int idx,int cnt,ll lcm){

if(lcm>n)break;

if(idx==m) {

if(cnt==0)return;

if(cnt & 1)re1+=n/lcm;

else re1-=n/lcm;

return;

}

func(idx+1,cnt+1,(lcm\*ar[idx])/\_\_gcd(lcm,(ll)ar[idx]));

func(idx+1,cnt,lcm);

}

void solve(){

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

for(int i=0;i<m;i++)scanf("%d",&ar[i]);

///using bitmask

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

ll lcm=1;

int cnt=0;

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

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

cnt++;

lcm=(lcm\*ar[j])/\_\_gcd(lcm,(ll)ar[j]);

if(lcm>n)break;

}

}

if(cnt&1)re+=n/lcm;

else re-=n/lcm;

}

}

**Linear Sieve:**

bitset<mx>is\_composite;

vector<int>prime;

int phi[mx],mobius[mx];

void seive(int n){

phi[1]=mobius[1]=1;

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

mobius[i]=1;

if(!is\_composite[i]){

prime.push\_back(i);

phi[i]=i-1; ///i is prime

}

for(int j=0;j<prime.size() && i\*prime[j]<=n;j++) {

is\_composite[i\*prime[j]]=true;

if(i%prime[j]==0) {

phi[i\*prime[j]]=phi[i]\*prime[j]; ///prime[j] divides i

break;

}

else {

phi[i\*prime[j]]=phi[i]\*phi[prime[j]]; ///prime[j] do not divide i

}

}

}

for(int val:prime) {

int temp=val\*val;

if(temp>n)break;

for(int j=temp;j<=n;j+=temp)mobius[j]=0;

}

for(int val:prime){

for(int j=val;j<=n;j+=val)mobius[j]\*=-1;

}

}

**Eulor totient:**

int phi[mx];

void eulor\_totient(int n){

for(int i=2; i<=n; i++) phi[i]=i;

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

phi[i]>>=1;

}

for(int i=3; i<=n; i+=2) {

if(phi[i]==i) {

phi[i]--;

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

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

}

}

}

**CRT:**

ll ar[mx],br[mx];

struct GCD\_type { ll x, y, d; };

GCD\_type ex\_GCD(ll a, ll b){

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

GCD\_type pom = ex\_GCD(b, a % b);

return {pom.y, pom.x - a / b \* pom.y, pom.d};

}

ll normalize(ll val,ll mod){

val%=mod;

if(val<0)val+=mod;

return val;

}

void solve(){

ll ans=br[1]; /// here br remainder

ll lcm=ar[1];

bool f=true;

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

auto pom=ex\_GCD(lcm,ar[i]);

ll x1=pom.x;

ll d=pom.d;

if((br[i]-ans)%d!=0){

f=false;break;

}

ans=ans+x1\*(br[i]-ans)/d%(ar[i]/d)\*lcm;

ans=normalize(ans,lcm\*ar[i]/d);

lcm=(lcm\*ar[i])/\_\_gcd(lcm,ar[i]);

}

if(f)printf("%lld %lld\n",ans,lcm); /// here is the smallest answer .next xth answer will be ans+x\*lcm where x=[1,2,....]

}

**Extended Euclidean (inverse):**

int Extended\_Euclidean(int a,int b,int &x,int &y)

{

if(b==0){

x=1;y=0;

return a;

}

int d=Extended\_Euclidean(b,a%b,y,x);

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

return d;

}

int Inverse\_Modulo(int a,int m){

int x,y,d;

d=Extended\_Euclidean(a,m,x,y);

if(d==1) return (x+m)%m;

return -1; //No Solution

}

**Big Mod, Fact:**

ll bigmod(ll e,ll x){

if(!x)return 1;

ll p=bigmod(e,x/2);

p=(p\*p)%mod;

if(x%2)p=(p\*e)%mod;

return p;

}

void fact\_cal(){

fact[0]=1,inv[0]=1;

for(int i=1;i<=mx-3;i++){

fact[i]=(fact[i-1]\*i)%mod;

}

inv[mx-3]=bigmod(fact[mx-3],mod-2);

for(int i=mx-4;i>=1;i--)inv[i]=(inv[i+1]\*(i+1))%mod;

}

**Stirling Number of 2nd kind:**

ll dp[mx][mx];

ll func(int nn,int kk){

if(kk==1)return 1;

if(nn==kk)return 1;

if(kk==0)return 0;

ll &val=dp[nn][kk];

if(val!=-1)return val;

val=func(nn-1,kk-1)+1LL\*kk\*func(nn-1,kk);

return val;

}

**Pollard RHO:**

#define pii pair<ll,int>

ll Mul(ll a,ll b,ll Mod){

ll Ans=0;

while(b){

if(b&1) {Ans+=a; if(Ans>=Mod) Ans-=Mod;}

a+=a; if(a>=Mod) a-=Mod;

b>>=1;

}

return Ans;

}

ll bigMod(ll n,ll r,ll Mod){

if(r==0) return 1LL;

ll ret=bigMod(n,r/2,Mod);

ret=Mul(ret,ret,Mod);

if(r%2==1) ret=Mul(ret,n,Mod);

return ret;

}

//Miller-Rabin

bool witness(ll wit,ll n){

if(wit>=n) return false;

int s=0; ll t=n-1;

while(t%2==0) s++,t/=2;

wit=bigMod(wit,t,n);

if(wit==1 || wit==n-1) return false;

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

wit=Mul(wit,wit,n);

if(wit==1) return true;

if(wit==n - 1) return false;

}

return true;

}

//Is n prime?

bool miller(ll n){

if(n==2) return true;

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

if(witness(2,n) || witness(7,n) || witness(61,n)) return false;

return true;

}

// Pollard's Rho

// a must not equal 0 or -2.

// returns a divisor, a proper one when succeeded, equal to n if failed

// in case of failure, change a

ll rho(ll n,ll a) {

auto f=[&](ll x) {return (Mul(x,x,n)+a)%n; };

ll x=2,y=2;

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

x=f(x); y=f(f(y));

ll d=\_\_gcd(n,abs(x-y));

if(d!=1) return d;

}

return n;

}

ll get\_factor(ll n){

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

if(n%3==0) return 3;

if(n%5==0) return 5;

while(true){

ll a=2+rand()%100;

ll d=rho(n,a);

if(d!=n) return d;

}

return n;

}

void factorize(ll n,vector<ll> &x) {

if(n==1) return;

else if(miller(n)) x.push\_back(n);

else{

ll d=get\_factor(n);

factorize(d,x);

factorize(n/d,x);

}

}

vector<ll>factorize(ll n) {vector<ll>x; factorize(n, x); return x;}

vector<pii>Factors; // store factor

vector<ll>Divisors;//strore divisors

void findDiv(int pos,ll val){

if(pos<0) {Divisors.push\_back(val); return;}

ll Now=1;

for(int i=0;i<=Factors[pos].second;i++){

findDiv(pos-1,val\*Now);

Now=Now\*Factors[pos].first;

}

}

void findAllDiv(ll n){

vector<ll>now=factorize(n);

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

Factors.clear();

Divisors.clear();

int Count=1;

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

if(now[i]==now[i-1]) Count++;

else {Factors.push\_back({now[i-1],Count}); Count=1;}

}

Factors.push\_back({now.back(),Count});

findDiv(Factors.size()-1,1);

}

**2D Geometry:**

typedef double Tf;

typedef Tf Ti; /// use long long for exactness

const Tf PI = acos(-1), EPS = 1e-9;

int dcmp(Tf x) { return abs(x) < EPS ? 0 : (x<0 ? -1 : 1);}

struct Point {

Ti x, y;

Point(Ti x = 0, Ti y = 0) : x(x), y(y) {}

Point operator + (const Point& u) const { return Point(x + u.x, y + u.y); }

Point operator - (const Point& u) const { return Point(x - u.x, y - u.y); }

Point operator \* (const long long u) const { return Point(x \* u, y \* u); }

Point operator \* (const Tf u) const { return Point(x \* u, y \* u); }

Point operator / (const Tf u) const { return Point(x / u, y / u); }

bool operator == (const Point& u) const { return dcmp(x - u.x) == 0 && dcmp(y - u.y) == 0; }

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

bool operator < (const Point& u) const { return dcmp(x - u.x) < 0 || (dcmp(x - u.x) == 0 && dcmp(y - u.y) < 0); }

friend istream &operator >> (istream &is, Point &p) { return is >> p.x >> p.y; }

friend ostream &operator << (ostream &os, const Point &p) { return os << p.x << " " << p.y; }

};

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

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

Tf length(Point a) { return sqrt(dot(a, a)); }

Ti sqLength(Point a) { return dot(a, a); }

Tf distance(Point a, Point b) {return length(a-b);}

Tf angle(Point u) { return atan2(u.y, u.x); }

// returns angle between oa, ob in (-PI, PI]

Tf angleBetween(Point a, Point b) {

double ans = angle(b) - angle(a);

return ans <= -PI ? ans + 2\*PI : (ans > PI ? ans - 2\*PI : ans);

}

// Rotate a ccw by rad radians

Point rotate(Point a, Tf rad) {

static\_assert(is\_same<Tf, Ti>::value);

return Point(a.x \* cos(rad) - a.y \* sin(rad), a.x \* sin(rad) + a.y \* cos(rad));

}

// rotate a ccw by angle th with cos(th) = co && sin(th) = si

Point rotatePrecise(Point a, Tf co, Tf si) {

static\_assert(is\_same<Tf, Ti>::value);

return Point(a.x \* co - a.y \* si, a.y \* co + a.x \* si);

}

Point rotate90(Point a) { return Point(-a.y, a.x); }

// scales vector a by s such that length of a becomes s

Point scale(Point a, Tf s) {

static\_assert(is\_same<Tf, Ti>::value);

return a / length(a) \* s;

}

// returns an unit vector perpendicular to vector a

Point normal(Point a) {

static\_assert(is\_same<Tf, Ti>::value);

Tf l = length(a);

return Point(-a.y / l, a.x / l);

}

// returns 1 if c is left of ab, 0 if on ab && -1 if right of ab

int orient(Point a, Point b, Point c) {

return dcmp(cross(b - a, c - a));

}

bool half(Point p){ // returns true for point above x axis or on negative x axis

return p.y > 0 || (p.y == 0 && p.x < 0);

}

bool polarComp(Point p, Point q){ //to be used in sort() function

return make\_tuple(half(p), 0) < make\_tuple(half(q), cross(p, q));}

struct Segment {

Point a, b;

Segment(Point aa, Point bb) : a(aa), b(bb) {}

};

typedef Segment Line;

struct Circle {

Point o;

Tf r;

Circle(Point o = Point(0, 0), Tf r = 0) : o(o), r(r) {}

// returns true if point p is in || on the circle

bool contains(Point p) {

return dcmp(sqLength(p - o) - r \* r) <= 0;

}

// returns a point on the circle rad radians away from +X CCW

Point point(Tf rad) {

static\_assert(is\_same<Tf, Ti>::value);

return Point(o.x + cos(rad) \* r, o.y + sin(rad) \* r);

}

// area of a circular sector with central angle rad

Tf area(Tf rad = PI + PI) { return rad \* r \* r / 2; }

// area of the circular sector cut by a chord with central angle alpha

Tf sector(Tf alpha) { return r \* r \* 0.5 \* (alpha - sin(alpha)); }

};

namespace Linear {

// returns true if point p is on segment s

bool onSegment(Point p, Segment s) {

return dcmp(cross(s.a - p, s.b - p)) == 0 && dcmp(dot(s.a - p, s.b - p)) <= 0;

}

// returns true if segment p && q touch or intersect

bool segmentsIntersect(Segment p, Segment q) {

if(onSegment(p.a, q) || onSegment(p.b, q)) return true;

if(onSegment(q.a, p) || onSegment(q.b, p)) return true;

Ti c1 = cross(p.b - p.a, q.a - p.a);

Ti c2 = cross(p.b - p.a, q.b - p.a);

Ti c3 = cross(q.b - q.a, p.a - q.a);

Ti c4 = cross(q.b - q.a, p.b - q.a);

return dcmp(c1) \* dcmp(c2) < 0 && dcmp(c3) \* dcmp(c4) < 0;

}

bool linesParallel(Line p, Line q) {

return dcmp(cross(p.b - p.a, q.b - q.a)) == 0;

}

// lines are represented as a ray from a point: (point, vector)

// returns false if two lines (p, v) && (q, w) are parallel or collinear

// true otherwise, intersection point is stored at o via reference

bool lineLineIntersection(Point p, Point v, Point q, Point w, Point& o) {

static\_assert(is\_same<Tf, Ti>::value);

if(dcmp(cross(v, w)) == 0) return false;

Point u = p - q;

o = p + v \* (cross(w,u)/cross(v,w));

return true;

}

// returns false if two lines p && q are parallel or collinear

// true otherwise, intersection point is stored at o via reference

bool lineLineIntersection(Line p, Line q, Point& o) {

return lineLineIntersection(p.a, p.b - p.a, q.a, q.b - q.a, o);

}

// returns the distance from point a to line l

Tf distancePointLine(Point p, Line l) {

return abs(cross(l.b - l.a, p - l.a) / length(l.b - l.a));

}

// returns the shortest distance from point a to segment s

Tf distancePointSegment(Point p, Segment s) {

if(s.a == s.b) return length(p - s.a);

Point v1 = s.b - s.a, v2 = p - s.a, v3 = p - s.b;

if(dcmp(dot(v1, v2)) < 0) return length(v2);

else if(dcmp(dot(v1, v3)) > 0) return length(v3);

else return abs(cross(v1, v2) / length(v1));

}

// returns the shortest distance from segment p to segment q

Tf distanceSegmentSegment(Segment p, Segment q) {

if(segmentsIntersect(p, q)) return 0;

Tf ans = distancePointSegment(p.a, q);

ans = min(ans, distancePointSegment(p.b, q));

ans = min(ans, distancePointSegment(q.a, p));

ans = min(ans, distancePointSegment(q.b, p));

return ans;

}

// returns the projection of point p on line l

Point projectPointLine(Point p, Line l) {

static\_assert(is\_same<Tf, Ti>::value);

Point v = l.b - l.a;

return l.a + v \* ((Tf) dot(v, p - l.a) / dot(v, v));

}

} // namespace Linear

typedef vector<Point> Polygon;

namespace Polygonal {

// returns the signed area of polygon p of n vertices

Tf signedPolygonArea(Polygon p) {

Tf ret = 0;

for(int i = 0; i < (int) p.size() - 1; i++)

ret += cross(p[i]-p[0], p[i+1]-p[0]);

return ret / 2;

}

// given a polygon p of n vertices, generates the convex hull in ch

// in CCW && returns the number of vertices in the convex hull

int convexHull(Polygon p, Polygon &ch) {

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

int n = p.size();

ch.resize(n + n);

int m = 0; // preparing lower hull

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

while(m > 1 && dcmp(cross(ch[m - 1] - ch[m - 2], p[i] - ch[m - 1])) <= 0) m--;

ch[m++] = p[i];

}

int k = m; // preparing upper hull

for(int i = n - 2; i >= 0; i--) {

while(m > k && dcmp(cross(ch[m - 1] - ch[m - 2], p[i] - ch[m - 2])) <= 0) m--;

ch[m++] = p[i];

}

if(n > 1) m--;

ch.resize(m);

return m;

}

// for a point o and polygon p returns:

// -1 if o is strictly inside p

// 0 if o is on a segment of p

// 1 if o is strictly outside p

// computes via winding numbers

int pointInPolygon(Point o, Polygon p) {

using Linear::onSegment;

int wn = 0, n = p.size();

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

int j = (i + 1) % n;

if(onSegment(o, Segment(p[i], p[j])) || o == p[i]) return 0;

int k = dcmp(cross(p[j] - p[i], o - p[i]));

int d1 = dcmp(p[i].y - o.y);

int d2 = dcmp(p[j].y - o.y);

if(k > 0 && d1 <= 0 && d2 > 0) wn++;

if(k < 0 && d2 <= 0 && d1 > 0) wn--;

}

return wn ? -1 : 1;

}

// returns the longest line segment of l that is inside or on the

// simply polygon p. O(n lg n). TESTED: TIMUS 1955

Tf longestSegInPoly(Line l, const Polygon &p) {

using Linear::lineLineIntersection;

int n = p.size();

vector<pair<Tf, int>> ev;

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

Point a = p[i], b = p[(i + 1) % n], z = p[(i - 1 + n) % n];

int ora = orient(l.a, l.b, a), orb = orient(l.a, l.b, b), orz = orient(l.a, l.b, z);

if(!ora) {

Tf d = dot(a - l.a, l.b - l.a);

if(orz && orb) {

if(orz != orb) ev.emplace\_back(d, 0);

}

else if(orz) ev.emplace\_back(d, orz);

else if(orb) ev.emplace\_back(d, orb);

}

else if(ora == -orb) {

Point ins;

lineLineIntersection(l, Line(a, b), ins);

ev.emplace\_back(dot(ins - l.a, l.b - l.a), 0);

} }

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

Tf ret = 0, cur = 0, pre = 0;

bool active = false;

int sign = 0;

for(auto &qq : ev) {

int tp = qq.second;

Tf d = qq.first;

if(sign) {

cur += d - pre;

ret = max(ret, cur);

if(tp != sign) active = !active;

sign = 0;

}

else {

if(active) cur += d - pre, ret = max(ret, cur);

if(tp == 0) active = !active;

else sign = tp;

}

pre = d;

if(!active) cur = 0;

}

ret /= length(l.b - l.a);

return ret;

}

} // namespace Polygonal

namespace Convex {

///Tested on Kattis::fenceortho

void rotatingCalipersGetRectangle(Point\* p, int n, Tf& area, Tf& perimeter) {

using Linear::distancePointLine;

static\_assert(is\_same<Tf, Ti>::value);

p[n] = p[0];

int l = 1, r = 1, j = 1;

area = perimeter = 1e100;

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

Point v = (p[i + 1] - p[i]) / length(p[i + 1] - p[i]);

while(dcmp(dot(v, p[r % n] - p[i]) - dot(v, p[(r + 1) % n] - p[i])) < 0) r++;

while(j < r || dcmp(cross(v, p[j % n] - p[i]) - cross(v, p[(j + 1) % n] - p[i])) < 0) j++;

while(l < j || dcmp(dot(v, p[l % n] - p[i]) - dot(v, p[(l + 1) % n] - p[i])) > 0) l++;

Tf w = dot(v, p[r % n] - p[i]) - dot(v, p[l % n] - p[i]);

Tf h = distancePointLine(p[j % n], Line(p[i], p[i + 1]));

area = min(area, w \* h);

perimeter = min(perimeter, 2 \* w + 2 \* h);

} }

// returns the left side of polygon u after cutting it by ray a->b

Polygon cutPolygon(Polygon u, Point a, Point b) {

using Linear::lineLineIntersection, Linear::onSegment;

Polygon ret;

int n = u.size();

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

Point c = u[i], d = u[(i + 1) % n];

if(dcmp(cross(b-a, c-a)) >= 0) ret.push\_back(c);

if(dcmp(cross(b-a, d-c)) != 0) {

Point t;

lineLineIntersection(a, b - a, c, d - c, t);

if(onSegment(t, Segment(c, d))) ret.push\_back(t);

} }

return ret;

}

// returns true if point p is in or on triangle abc

bool pointInTriangle(Point a, Point b, Point c, Point p) {

return dcmp(cross(b - a, p - a)) >= 0

&& dcmp(cross(c - b, p - b)) >= 0

&& dcmp(cross(a - c, p - c)) >= 0;

}

// pt must be in ccw order with no three collinear points

// returns inside = -1, on = 0, outside = 1

int pointInConvexPolygon(const Polygon &pt, Point p) {

int n = pt.size();

assert(n >= 3);

int lo = 1, hi = n - 1;

while(hi - lo > 1) {

int mid = (lo + hi) / 2;

if(dcmp(cross(pt[mid] - pt[0], p - pt[0])) > 0) lo = mid;

else hi = mid;

}

bool in = pointInTriangle(pt[0], pt[lo], pt[hi], p);

if(!in) return 1;

if(dcmp(cross(pt[lo] - pt[lo - 1], p - pt[lo - 1])) == 0) return 0;

if(dcmp(cross(pt[hi] - pt[lo], p - pt[lo])) == 0) return 0;

if(dcmp(cross(pt[hi] - pt[(hi + 1) % n], p - pt[(hi + 1) % n])) == 0) return 0;

return -1;

}

// Extreme Point for a direction is the farthest point in that direction

// poly is a convex polygon, sorted in CCW, doesn't contain redundant points

// u is the direction for extremeness

int extremePoint(const Polygon &poly, Point u = Point(0, 1)) {

int n = (int) poly.size();

int a = 0, b = n;

while(b - a > 1) {

int c = (a + b) / 2;

if(dcmp(dot(poly[c] - poly[(c + 1) % n], u)) >= 0 && dcmp(dot(poly[c] - poly[(c - 1 + n) % n], u)) >= 0) {

return c;

}

bool a\_up = dcmp(dot(poly[(a + 1) % n] - poly[a], u)) >= 0;

bool c\_up = dcmp(dot(poly[(c + 1) % n] - poly[c], u)) >= 0;

bool a\_above\_c = dcmp(dot(poly[a] - poly[c], u)) > 0;

if(a\_up && !c\_up) b = c;

else if(!a\_up && c\_up) a = c;

else if(a\_up && c\_up) {

if(a\_above\_c) b = c;

else a = c;

}

else {

if(!a\_above\_c) b = c;

else a = c;

} }

if(dcmp(dot(poly[a] - poly[(a + 1) % n], u)) > 0 && dcmp(dot(poly[a] - poly[(a - 1 + n) % n], u)) > 0)

return a;

return b % n;

}

// For a convex polygon p and a line l, returns a list of segments

// of p that are touch or intersect line l.

// the i'th segment is considered (p[i], p[(i + 1) modulo |p|])

// #1 If a segment is collinear with the line, only that is returned

// #2 Else if l goes through i'th point, the i'th segment is added

// If there are 2 or more such collinear segments for #1,

// any of them (only one, not all) should be returned (not tested)

// Complexity: O(lg |p|)

vector<int> lineConvexPolyIntersection(const Polygon &p, Line l) {

assert((int) p.size() >= 3);

assert(l.a != l.b);

int n = p.size();

vector<int> ret;

Point v = l.b - l.a;

int lf = extremePoint(p, rotate90(v));

int rt = extremePoint(p, rotate90(v) \* Ti(-1));

int olf = orient(l.a, l.b, p[lf]);

int ort = orient(l.a, l.b, p[rt]);

if(!olf || !ort) {

int idx = (!olf ? lf : rt);

if(orient(l.a, l.b, p[(idx - 1 + n) % n]) == 0)

ret.push\_back((idx - 1 + n) % n);

else ret.push\_back(idx);

return ret;

}

if(olf == ort) return ret;

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

int lo = i ? rt : lf;

int hi = i ? lf : rt;

int olo = i ? ort : olf;

while(true) {

int gap = (hi - lo + n) % n;

if(gap < 2) break;

int mid = (lo + gap / 2) % n;

int omid = orient(l.a, l.b, p[mid]);

if(!omid) {

lo = mid;

break;

}

if(omid == olo) lo = mid;

else hi = mid;

}

ret.push\_back(lo);

}

return ret;

}

// Calculate [ACW, CW] tangent pair from an external point

constexpr int CW = -1, ACW = 1;

bool isGood(Point u, Point v, Point Q, int dir) { return orient(Q, u, v) != -dir; }

Point better(Point u, Point v, Point Q, int dir) { return orient(Q, u, v) == dir ? u : v; }

Point pointPolyTangent(const Polygon &pt, Point Q, int dir, int lo, int hi) {

while(hi - lo > 1) {

int mid = (lo + hi) / 2;

bool pvs = isGood(pt[mid], pt[mid - 1], Q, dir);

bool nxt = isGood(pt[mid], pt[mid + 1], Q, dir);

if(pvs && nxt) return pt[mid];

if(!(pvs || nxt)) {

Point p1 = pointPolyTangent(pt, Q, dir, mid + 1, hi);

Point p2 = pointPolyTangent(pt, Q, dir, lo, mid - 1);

return better(p1, p2, Q, dir);

}

if(!pvs) {

if(orient(Q, pt[mid], pt[lo]) == dir) hi = mid - 1;

else if(better(pt[lo], pt[hi], Q, dir) == pt[lo]) hi = mid - 1;

else lo = mid + 1;

}

if(!nxt) {

if(orient(Q, pt[mid], pt[lo]) == dir) lo = mid + 1;

else if(better(pt[lo], pt[hi], Q, dir) == pt[lo]) hi = mid - 1;

else lo = mid + 1;

}

}

Point ret = pt[lo];

for(int i = lo + 1; i <= hi; i++) ret = better(ret, pt[i], Q, dir);

return ret;

}

// [ACW, CW] Tangent

pair<Point, Point> pointPolyTangents(const Polygon &pt, Point Q) {

int n = pt.size();

Point acw\_tan = pointPolyTangent(pt, Q, ACW, 0, n - 1);

Point cw\_tan = pointPolyTangent(pt, Q, CW, 0, n - 1);

return make\_pair(acw\_tan, cw\_tan);

}

}

namespace Circular {

// Extremely inaccurate for finding near touches

// compute intersection of line l with circle c

// The intersections are given in order of the ray (l.a, l.b)

vector<Point> circleLineIntersection(Circle c, Line l) {

static\_assert(is\_same<Tf, Ti>::value);

vector<Point> ret;

Point b = l.b - l.a, a = l.a - c.o;

Tf A = dot(b, b), B = dot(a, b);

Tf C = dot(a, a) - c.r \* c.r, D = B\*B - A\*C;

if (D < -EPS) return ret;

ret.push\_back(l.a + b \* (-B - sqrt(D + EPS)) / A);

if (D > EPS)

ret.push\_back(l.a + b \* (-B + sqrt(D)) / A);

return ret;

}

// signed area of intersection of circle(c.o, c.r) &&

// triangle(c.o, s.a, s.b) [cross(a-o, b-o)/2]

Tf circleTriangleIntersectionArea(Circle c, Segment s) {

using Linear::distancePointSegment;

Tf OA = length(c.o - s.a);

Tf OB = length(c.o - s.b);

// sector

if(dcmp(distancePointSegment(c.o, s) - c.r) >= 0)

return angleBetween(s.a-c.o, s.b-c.o) \* (c.r \* c.r) / 2.0;

// triangle

if(dcmp(OA - c.r) <= 0 && dcmp(OB - c.r) <= 0)

return cross(c.o - s.b, s.a - s.b) / 2.0;

// three part: (A, a) (a, b) (b, B)

vector<Point> Sect = circleLineIntersection(c, s);

return circleTriangleIntersectionArea(c, Segment(s.a, Sect[0]))

+ circleTriangleIntersectionArea(c, Segment(Sect[0], Sect[1]))

+ circleTriangleIntersectionArea(c, Segment(Sect[1], s.b));

}

// area of intersecion of circle(c.o, c.r) && simple polyson(p[])

// Tested : https://codeforces.com/gym/100204/problem/F - Little Mammoth

Tf circlePolyIntersectionArea(Circle c, Polygon p) {

Tf res = 0;

int n = p.size();

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

res += circleTriangleIntersectionArea(c, Segment(p[i], p[(i + 1) % n]));

return abs(res);

}

// locates circle c2 relative to c1

// interior (d < R - r) ----> -2

// interior tangents (d = R - r) ----> -1

// concentric (d = 0)

// secants (R - r < d < R + r) ----> 0

// exterior tangents (d = R + r) ----> 1

// exterior (d > R + r) ----> 2

int circleCirclePosition(Circle c1, Circle c2) {

Tf d = length(c1.o - c2.o);

int in = dcmp(d - abs(c1.r - c2.r)), ex = dcmp(d - (c1.r + c2.r));

return in < 0 ? -2 : in == 0 ? -1 : ex == 0 ? 1 : ex > 0 ? 2 : 0;

}

// compute the intersection points between two circles c1 && c2

vector<Point> circleCircleIntersection(Circle c1, Circle c2) {

static\_assert(is\_same<Tf, Ti>::value);

vector<Point> ret;

Tf d = length(c1.o - c2.o);

if(dcmp(d) == 0) return ret;

if(dcmp(c1.r + c2.r - d) < 0) return ret;

if(dcmp(abs(c1.r - c2.r) - d) > 0) return ret;

Point v = c2.o - c1.o;

Tf co = (c1.r \* c1.r + sqLength(v) - c2.r \* c2.r) / (2 \* c1.r \* length(v));

Tf si = sqrt(abs(1.0 - co \* co));

Point p1 = scale(rotatePrecise(v, co, -si), c1.r) + c1.o;

Point p2 = scale(rotatePrecise(v, co, si), c1.r) + c1.o;

ret.push\_back(p1);

if(p1 != p2) ret.push\_back(p2);

return ret;

}

// intersection area between two circles c1, c2

Tf circleCircleIntersectionArea(Circle c1, Circle c2) {

Point AB = c2.o - c1.o;

Tf d = length(AB);

if(d >= c1.r + c2.r) return 0;

if(d + c1.r <= c2.r) return PI \* c1.r \* c1.r;

if(d + c2.r <= c1.r) return PI \* c2.r \* c2.r;

Tf alpha1 = acos((c1.r \* c1.r + d \* d - c2.r \* c2.r) / (2.0 \* c1.r \* d));

Tf alpha2 = acos((c2.r \* c2.r + d \* d - c1.r \* c1.r) / (2.0 \* c2.r \* d));

return c1.sector(2 \* alpha1) + c2.sector(2 \* alpha2);

}

// returns tangents from a point p to circle c

vector<Point> pointCircleTangents(Point p, Circle c) {

static\_assert(is\_same<Tf, Ti>::value);

vector<Point> ret;

Point u = c.o - p;

Tf d = length(u);

if(d < c.r) ;

else if(dcmp(d - c.r) == 0) {

ret = { rotate(u, PI / 2) };

}

else {

Tf ang = asin(c.r / d);

ret = { rotate(u, -ang), rotate(u, ang) };

}return ret; }

// returns the points on tangents that touches the circle

vector<Point> pointCircleTangencyPoints(Point p, Circle c) {

static\_assert(is\_same<Tf, Ti>::value);

Point u = p - c.o;

Tf d = length(u);

if(d < c.r) return {};

else if(dcmp(d - c.r) == 0) return {c.o + u};

else {

Tf ang = acos(c.r / d);

u = u / length(u) \* c.r;

return { c.o + rotate(u, -ang), c.o + rotate(u, ang) };

}

}

// for two circles c1 && c2, returns two list of points a && b

// such that a[i] is on c1 && b[i] is c2 && for every i

// Line(a[i], b[i]) is a tangent to both circles

// CAUTION: a[i] = b[i] in case they touch | -1 for c1 = c2

int circleCircleTangencyPoints(Circle c1, Circle c2, vector<Point> &a, vector<Point> &b) {

a.clear(), b.clear();

int cnt = 0;

if(dcmp(c1.r - c2.r) < 0) {

swap(c1, c2); swap(a, b);

}

Tf d2 = sqLength(c1.o - c2.o);

Tf rdif = c1.r - c2.r, rsum = c1.r + c2.r;

if(dcmp(d2 - rdif \* rdif) < 0) return 0;

if(dcmp(d2) == 0 && dcmp(c1.r - c2.r) == 0) return -1;

Tf base = angle(c2.o - c1.o);

if(dcmp(d2 - rdif \* rdif) == 0) {

a.push\_back(c1.point(base));

b.push\_back(c2.point(base));

cnt++;

return cnt; }

Tf ang = acos((c1.r - c2.r) / sqrt(d2));

a.push\_back(c1.point(base + ang));

b.push\_back(c2.point(base + ang));

cnt++;

a.push\_back(c1.point(base - ang));

b.push\_back(c2.point(base - ang));

cnt++;

if(dcmp(d2 - rsum \* rsum) == 0) {

a.push\_back(c1.point(base));

b.push\_back(c2.point(PI + base));

cnt++;

}

else if(dcmp(d2 - rsum \* rsum) > 0) {

Tf ang = acos((c1.r + c2.r) / sqrt(d2));

a.push\_back(c1.point(base + ang));

b.push\_back(c2.point(PI + base + ang));

cnt++;

a.push\_back(c1.point(base - ang));

b.push\_back(c2.point(PI + base - ang));

cnt++;}

return cnt;

}}

// Given a bunch of segments. Check if any two intersect.

// Sweep Line. O(n lg n). TESTED: CF 1359F

namespace IntersectingSegments {

Tf yvalSegment(const Line &s, Tf x) {

if(dcmp(s.a.x - s.b.x) == 0) return s.a.y;

return s.a.y + (s.b.y - s.a.y) \* (x - s.a.x) / (s.b.x - s.a.x);

}

struct SegCompare {

bool operator () (const Segment &p, const Segment &q) const {

Tf x = max(min(p.a.x, p.b.x), min(q.a.x, q.b.x));

return dcmp(yvalSegment(p, x) - yvalSegment(q, x)) < 0;

} };

multiset<Segment, SegCompare> st;

typedef multiset<Segment, SegCompare>::iterator iter;

iter prev(iter it) {

return it == st.begin() ? st.end() : --it;

}

iter next(iter it) {

return it == st.end() ? st.end() : ++it;}

struct Event {

Tf x; int tp, id;

Event(Ti x, int tp, int id) : x(x), tp(tp), id(id) { }

bool operator < (const Event &p) const {

if(dcmp(x - p.x)) return x < p.x;

return tp > p.tp;

}};

bool anyIntersection(const vector<Segment> &v) {

using Linear::segmentsIntersect;

vector<Event> ev;

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

ev.push\_back(Event(min(v[i].a.x, v[i].b.x), +1, i));

ev.push\_back(Event(max(v[i].a.x, v[i].b.x), -1, i));

}

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

st.clear();

vector<iter> where(v.size());

for(auto &cur : ev) {

int id = cur.id;

if(cur.tp == 1) {

iter nxt = st.lower\_bound(v[id]);

iter pre = prev(nxt);

if(pre != st.end() && segmentsIntersect(\*pre, v[id])) return true;

if(nxt != st.end() && segmentsIntersect(\*nxt, v[id])) return true;

where[id] = st.insert(nxt, v[id]); }

else {

iter nxt = next(where[id]);

iter pre = prev(where[id]);

if(pre != st.end() && nxt != st.end() && segmentsIntersect(\*pre, \*nxt))

return true;

st.erase(where[id]); } }

return false; }}}}

// **Ashik’s extra formula**

inner circle radius, r = area \* s

outer circle area, A = (abc)/4R

**BitMask:**

ll Set(ll N,ll pos) return N=N|(1LL<<pos);

ll Reset(ll N,ll pos) return N=N & ~(1LL<<pos);

bool chk(ll N,ll pos) return (bool)(N &(1LL<<pos));

///int id= \_\_builtin\_ctz(mask); its give the position of the first one from the left

/// int tot= \_\_builtin\_popcount(mask); number of one bit .

**Digit Dp All digit sum:**

ll dp[15][2][400][2];

const ll mpos=11;

char ch[40];

void convert(ll n){

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

ch[i]=(n%10)+'0';

n/=10;

}

reverse(ch,ch+mpos);

ch[mpos]=0;

}

ll func(ll pos,ll smallornot,ll digitvalcnt,ll startornot){

if(pos==mpos)

return digitvalcnt;

if(dp[pos][smallornot][digitvalcnt][startornot]!=-1)

return dp[pos][smallornot][digitvalcnt][startornot];

ll be=0, en=9,re=0;

if(!smallornot)

en=ch[pos]-'0';

for(ll i=be; i<=en; i++)

{

ll ismallornot= smallornot | (i<en);

ll idigitvalcnt=digitvalcnt+ i;

ll istartornot= startornot | (i!=0);

re+=func(pos+1,ismallornot,idigitvalcnt,istartornot);

}

return dp[pos][smallornot][digitvalcnt][startornot]=re;

}

func(0,0,0,0);

**SOS DP:**

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

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

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

dp[ar[i]]=ar[i];

}

for(int i=0;i<22;i++){

for(int mask=0;mask<(1<<22);mask++){

if(chk(mask,i))dp[mask]=max(dp[mask],dp[mask^(1<<i)]);

}

}

int boro=(1<<22)-1;

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

printf("%d ",dp[boro^ar[i]]);

}

/\* iterate all the submask of a mask

for(int mask=1;mask<(1<<sz);mask++) {

int tmask=mask&(mask-1);

while(tmask) {

cout<<tmask<<endl;

// dp[mask]=min(dp[mask],dp[tmask]+dp[mask^tmask]);

tmask=(tmask-1)&mask;

}

}\*/

**Combinatorics Notes:**

///0\*nC0+1\*nC1+2\*nC2+3\*nC3+.....+n\*nCn=n\*2^(n-1).

///0Cr+1Cr+2Cr+3Cr+4Cr+5Cr+6Cr+....+nCr= (n+1)C(r+1)

///(nC0)^2+(nC1)^2+(nC2)^2+....+(nCn)^2=(2\*n)Cn

///how many ways you can go to (0,0) to (n,m) coordinate(you can only up and right).

like n=2,m=3,so = 5!/(2!\*3!)

if there are more than two dimensions you will do just total moves time! / (x axis moves times!\* y axis moves time! \*.....)

///you have n balls k bucket # of ways insert the ball into bucket such that every bucket has more than 0 balls

total ways is (n-1)C(k-1).

modification , any numbers of ball then answer is,(n+k-1)C(k-1)

modification , per bucket condition 0<=k\_i<x\_i

for 0<=k\_i, RESULT1 =(n+k-1)C(k-1)

for k\_i>=x\_i, val\_i=kCi\*(n-i\*x\_i+k-1)C(k-1)

RESULT2 = ((-k)^1)\*val\_1+((-k)^2)\*val\_2+((-k)^1)\*val\_3+.....((-k)^k)\*val\_k

[But some time we have not calculated overall val1 to valk ,Because (n-(x\*kth)+k-1) will be <0]

Final result=RESULT1-RESULT2

///catalan number Cn=(1/(n+1))\*((2\*n)Cn)

In other form Cn=((2\*n)C(n))-((2\*n)C(n+1))

**Bitset:**

bitset<mx>bt;

bt.set() /// all bit 1

bt.reset() ///all bit 0

bt.count() // total number of 1 bit

bt.\_Find\_first() // palce of the first 1 bit

bt.\_Find\_next() // next one bit

for(int i=bt.\_Find\_first();i<mx;i=bt.\_Find\_next()) // for traversing all 1 node

**Iterative Stack:**

template<typename T, typename Container = std::deque<T>>

class iterable\_stack

: public std::stack<T, Container>

{

using std::stack<T, Container>::c;

public:

auto begin() { return std::begin(c); }

auto end() { return std::end(c); }

auto begin() const { return std::begin(c); }

auto end() const { return std::end(c); }

};

iterable\_stack<int> st;

st.push(2);

for(auto i: st)

std::cout << i << ' ';

**PBDS:**

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

#include <ext/pb\_ds/tree\_policy.hpp>

using namespace \_\_gnu\_pbds;

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

solve(){

ordered\_set os;

// 10. how many numbers are smaller than a given value(7)

cout << os.order\_of\_key(7);

// 12. how many numbers are greater than a given value(7)

cout << os.size() - os.order\_of\_key(8) << "\n";

// 14. if the given numbers are sorted in ascending order, what is the k'th number

cout << \*os.find\_by\_order(2) << "\n";

// 16. delete the k'th smallest number

os.erase(os.find\_by\_order(k));

// 22. what is the smallest number which is greater than or equal to a given number(7)

cout << \*os.lower\_bound(7) << "\n";

// 23. what is the smallest number which is greater than to a given number(7)

cout << \*os.upper\_bound(7) << "\n";

}

**Ashraful’s Template:**

s.sh:

for((i=1;i<100;i++));do

./gen $i>int

./a<int>out1

./brute<int>out2

diff out1 out2 || break

Done

gen.cpp:

mt19937\_64 rng(chrono::steady\_clock::now().time\_since\_epoch().count());

ll my\_rand(ll l, ll r) {

return uniform\_int\_distribution<ll>(l, r) (rng);

}

**Ashik’s Fast I/O:**

#define faster\_io ios\_base::sync\_with\_stdio(0);cin.tie(0);cout.tie(0);

#define watch2(x,y) cout<< \_LINE\_ << " says: " <<#x<<" = "<<x<<" "<<#y<<" = "<<y <<endl

freopen("input.txt","r",stdin); ("output.txt","w",stdout);

**BIG Integer:**

import static java.lang.System.in;

import java.util.Scanner;

import java.math.BigInteger;

public class Main {

public static void main(String[] args) {

Scanner in = new Scanner(System.in);

int n;

n = in.nextInt();

BigInteger ci;

ci = new BigInteger("1");

//BigInteger b = in.nextBigInteger();

//BigInteger carry;

//carry = (a.multiply(b)).divide(a.gcd(b));

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

int temp = 4 \* (i + 1) - 6;

BigInteger tem = BigInteger.valueOf(temp);

ci = ci.multiply(tem);

ci = ci.divide(BigInteger.valueOf(i+1));

}

System.out.println(ci);

}

}

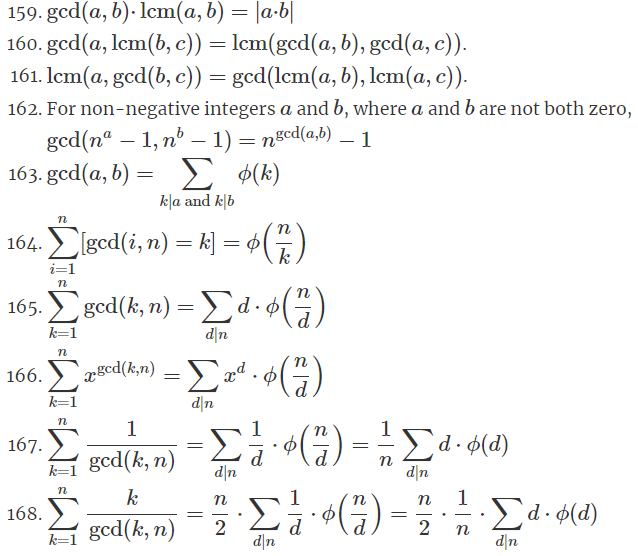
**De-arrangement:**

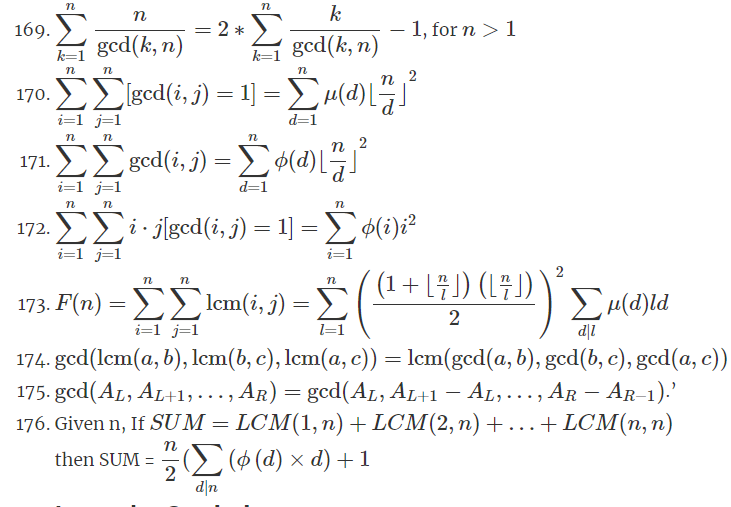
d(n)=(n−1)⋅(d(n−1)+d(n−2))whered(0)=1,d(1)=0

**Fibonacchi:**

Fn=15–√(1+5–√2)n−15–√(1−5–√2)n

**GCD**:





/\* Good Luck Ashik, Selim, Ashraful \*/