```
#pragma GCC optimize("Ofast")
#pragma GCC target("avx,avx2,fma")
#pragma GCC optimization ("unroll-loops")
#include<bits/stdc++.h>
using namespace std;
typedef long long int ll;
typedef pair<int,int> pii;
typedef vector<int> vii;
typedef vector<ll> vll;
typedef vector<pii> vpii;
typedef unordered_map<int,int> umap;
typedef long double ld;
#define fi first
#define se second
#define pb push_back
#define mp make_pair
#define popcount __builtin_popcount
#define case cout<<"Case "<<__test-test<<": ";
int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);
    int test=1;
    cin>>test;
    int __test=test;
    while(test--){
        int n,m,i,j,k;
   }
}
```

Combinatorics:

- $\left\langle {n \atop k} \right\rangle = \left\langle {n-1 \atop k-1} \right\rangle + \left\langle {n-1 \atop k} \right\rangle$
- $\sum \left\langle {n \atop k} \right\rangle = 2^n$
- $\sum_0^n \left\langle {n \atop k} \right\rangle k = n2^{n-1}$
- $\sum_{j=0}^{n} \left\langle {j \atop k} \right\rangle = \left\langle {n+1 \atop k+1} \right\rangle$
- $\sum_{i=0}^{r} \left\langle {n+i \atop i} \right\rangle = \left\langle {n+r+1 \atop r} \right\rangle$
- $\sum_{j=0}^{r} \left\langle {m \atop j} \right\rangle \left\langle {n \atop r-j} \right\rangle = \left\langle {m+n \atop r} \right\rangle$
- $\sum_{j=0}^{n} \left\langle {n \atop j} \right\rangle^2 = \left\langle {2n \atop n} \right\rangle$
- $\sum_{j=0}^{p} \left\langle {p+q-j \atop q} \right\rangle \left\langle {r+j \atop r} \right\rangle = \left\langle {p+q+r+1 \atop q+r+1} \right\rangle$
- $x_1 + x_2 + x_3 + \cdots x_k = n;$ $way = \left\langle {n+k-1 \atop k-1} \right\rangle$ for x≥ 0
- $\sum_{k=0}^{n} \frac{k!}{(k-r)!} \left\langle {n \atop k} \right\rangle = \frac{n!}{(n-r)!} . 2^{n-r}$
- $\sum_{k=0}^{n} \left\langle {n \atop k} \right\rangle k^m = \sum_{j=0}^{m} \left\{ {m \atop j} \right\} \left\langle {n \atop k} \right\rangle j!\, 2^{n-j}$
- $\sum_{k=0}^{n} \left\langle {n \atop k} \right\rangle \frac{1}{k+1} = \frac{2^{n+1}-1}{n+1}$
- Bell numbers: B_i is the number of way to partition a set

  $B_{n+1} = \sum_{i=0}^{n} \left\langle {n \atop i} \right\rangle B_i$
- $C_n = \left\langle {2n \atop n} \right\rangle - \left\langle {2n \atop n-1} \right\rangle = \frac{1}{n+1} \left\langle {2n \atop n} \right\rangle$

  $C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}$

  way from (0,0) to (n,n) not crossing diagonal

  ways of triangulation of (n+2) ways of correct paranthesis.
- $Eulerian\ number$:

$number\ of\ permutation\ of\ \{1,..n\}$ $with\ m\ a_i > a_{i+1}$

$A(n,m) = (n-m)A(n-1,m-1) + (m+1)A(n-1,m)$

$= \sum_{k=0}^{m+1} (-1)^k \left\langle {n+1 \atop k} \right\rangle (m+1-k)^n$

$$\sum_{m=0}^{n} A(n,m) = n!$$

Template:

```
#define mod
const int N;
ll fact[N],inv[N];
ll power(int b,int n=mod-2){
   if(b==0) return 0; ll ans=1ll;
  while(n){if(n&1)ans=ans*b%mod;b=b*1ll*b%mod;
n>>=1;}
   return ans;
}
void prep(){
   fact[0]=inv[0]=1;
   for(int i=1;i<N;i++)fact[i]=i*fact[i-1] %mod;
   inv[N-1]=power(fact[N-1]);
   for(int i=N-2;i>=0;i--)inv[i]=(i+1)*inv[i+1] %mod;
}
inline ll comb(int n,int k){
   if(n<k || k<0) return 0;
   return fact[n]*(inv[k]*inv[n-k] %mod) %mod;}
```

Stirling Number 1st Kind:

```
// include NTT here
int S(int n, int r) {
        int nn = 1;
        while(nn < n) nn <<= 1;
        for(int i = 0; i < n; ++i) {
                v[i].push_back(i);
                v[i].push_back(1);
        }
        for(int i = n; i < nn; ++i) {
                v[i].push_back(1);
        }

        for(int j = nn; j > 1; j >>= 1){
                int hn = j >> 1;
                for(int i = 0; i < hn; ++i){
        v[i]=multiply(v[i], v[i + hn]);
                }
        }
return v[0][r];}
```

Number Theory:

Miller Rabin test:

```
using u64 = uint64_t;
using u128 = __uint128_t;
u64 binpower(u64 base, u64 e, u64 mod)
{
   u64 result = 1;
   base %= mod;
   while (e)
   {
      if (e & 1)
         result = (u128)result * base % mod;
      base = (u128)base * base % mod;
      e >>= 1;
   }
   return result;
}
bool check_composite(u64 n, u64 a, u64 d, int s)
{
   u64 x = binpower(a, d, n);
   if (x == 1 || x == n - 1)
      return false;
   for (int r = 1; r < s; r++)
   {
      x = (u128)x * x % n;
      if (x == n - 1)
         return false;
   }
   return true;
};
bool MillerRabin(u64 n)   // returns true if n is
{
   ,! prime, else returns false.
      if (n < 2)
         return false;
   int r = 0;
               u64 d = n - 1;
               while ((d & 1) == 0)
{
   d >>= 1;
   r++;
}
for (int a : {2, 3, 5, 7, 11, 13, 17, 19, 23,
              ,! 29, 31, 37
             })
{
   if (n == a)
         return true;
      if (check_composite(n, a, d, r))
         return false;
   }
   return true;
            }
```

Discrete logarithm:

```
// Returns minimum x for which a^x=b(%m)
int solve(int a, int b, int m) {
   a%=m, b%=m;
   int k = 1, add = 0, g;
   while ((g = __gcd(a, m)) > 1) {
      if (b == k)
         return add;
      if (b % g)
         return -1;
      b /= g, m /= g, ++add;
      k = (k * 1ll * a / g) % m;
   }
   int n = sqrt(m) + 1;
   int an = 1;
   for (int i = 0; i < n; ++i)
      an = (an * 1ll * a) % m;

   unordered_map<int, int> vals;
   for (int q=0,cur=b; q<=n; ++q){
      vals[cur] = q;
      cur = (cur * 1ll * a) % m;
   }
   for (int p=1, cur=k; p<=n; ++p) {
      cur = (cur * 1ll * an) % m;
      if (vals.count(cur)) {
         int ans =n*p-vals[cur]+add;
         return ans;
      }
   }
   return -1;
}
```

Primitive root:

```
/*
exist if n=1,2,4,prime^k,2*prime^k;
returns minimum primitive root
(primitive root of n) is min g such that
g^x = a( %n) for any gcd(a,n)=1
solution for x exists....
*/
int powmod (int a, int b, int p) {
   int res=1;
   while(b)
      if(b&1) res=(int)(res*1ll*a%p),--b;
      else a=(int)(a*1ll*a%p),b>>=1;
   return res;
}
int generator (int p) {
   vector<int> fact;
   int phi=p-1,n=phi;
   for(int i=2;i*i<=n;++i)
   if(n%i==0){
      fact.push_back(i);
      while(n%i==0) n/=i;
   }
   if(n>1) fact.push_back(n);
   for(int res=2;res<=p;res++){
      bool ok=true;
      for(size_t i=0;i<fact.size() &&ok;i++)
         ok&=powmod(res,phi/fact[i],p)!=1;
      if(ok) return res;
   }
   return -1;
}
```

Discrete Root:

```
int powmod(int a, int b, int p) {
   int res = 1;
   while(b){
      if(b&1) res=res*1ll*a%p;
      a=a*1ll*a%p; b>>=1;
   }
   return res;
}
// Finds the primitive root modulo p
int generator(int p) {
```

```
    vector<int> fact;
    int phi = p-1, n = phi;
    for (int i = 2; i * i <= n; ++i) {
        if (n % i == 0) {
            fact.push_back(i);
            while (n % i == 0)
                n /= i;
        }
    }
    if (n > 1)
        fact.push_back(n);

    for (int res = 2; res <= p; ++res) {
        bool ok = true;
        for (int factor : fact) {
            if(powmod(res,phi/factor,p)==1){
                ok = false;
                break;
            }
        }
        if (ok) return res;
    }
    return -1;
}

// finds x such that x^k = a (mod n)
int main() {
    int n, k, a;
    scanf("%d %d %d", &n, &k, &a);
    if (a == 0) {
        puts("1\n0");
        return 0;
    }
    int g = generator(n);
    // Baby-step giant-step discrete logarithm algorithm
    int sq = (int) sqrt (n + .0) + 1;
    vector<pair<int, int>> dec(sq);
    for (int i = 1; i <= sq; ++i)
        dec[i-1] = {powmod(g, i * sq * k % (n - 1), n), i};
    sort(dec.begin(), dec.end());
    int any_ans = -1;
    for (int i = 0; i < sq; ++i) {
        int my = powmod(g, i * k % (n - 1), n) * a % n;
        auto it = lower_bound(dec.begin(), dec.end(),
make_pair(my, 0));
        if (it != dec.end() && it->first == my) {
            any_ans = it->second * sq -
i;Type equation here.
            break;
        }
    }
    if (any_ans == -1) {
        puts("0");
        return 0;
    }

    // Print all possible answers
    int delta = (n-1) / __gcd(k, n-1);
    vector<int> ans;
    for (int cur = any_ans % delta; cur < n-1; cur +=
delta)
        ans.push_back(powmod(g, cur, n));
    sort(ans.begin(), ans.end());
    printf("%d\n", ans.size());
    for (int answer : ans)
        printf("%d ", answer);
}
```

Euler totient:

- $\varphi(p^k) = p^k - p^{k-1}$
- $\phi(ab) = \phi(a) \cdot \phi(b) \cdot \frac{d}{\phi(d)}$
  where d=gcd(a,b).
- $\sum_{d|n} \phi(d) = n$
- $$\varphi(n) = \sum_{k|n} k\mu(\frac{n}{k})$$

Code to find phi from 1 to n

```
void phi_1_to_n(int n) {
    vector<int> phi(n + 1);
    phi[0] = 0;
    phi[1] = 1;
    for (int i = 2; i <= n; i++)
        phi[i] = i - 1;

    for (int i = 2; i <= n; i++)
        for (int j = 2 * i; j <= n; j += i)
            phi[j] -= phi[i];
}
```

Gray code:
G(n)=n^(n>>1)
rev_G(g)
{int n=0; while(g){n^=g;g>>=1;} return n; }

Enumerating submask:

```
for (int s=m; s; s=(s-1)&m);
```

FFT:

```
//this is the inplace version of fft
typedef complex<double> cd;
const double PI = acos(-1);
// this function doesn't return dft(a);
// it just manipulate in place
void fft(vector<cd> &a,bool invert){
    int n=a.size();

    for(int i=1,j=0;i<n;i++){
        int bit=n>>1;
        for(;j&bit;bit>>=1) j^=bit;
        j^=bit;
        if(i<j) swap(a[i],a[j]);
    }
    for(int len=2;len<=n;len<<=1){
        double ang=2*PI/len*(invert?-1:1);
        cd wlen(cos(ang),sin(ang));
        for(int i=0;i<n;i+=len){
            cd w(1);
            for(int j=0;j<len/2;j++){
                cd u= a[i+j], v=a[i+j+len/2]*w;
                a[i+j] = u+v;
                a[i+j+len/2] =u-v;
                w*=wlen;
            }
        }
    }
    if(invert) for(cd &x : a) x/=n;
}
//it takes two normal polynomial and returns a new
polynomial;
```

```
//possibly with 0s at the end
vector<int> multiply_pol(vector<int> const &a, vector<int> const& b){
   vector<cd>
fa(a.begin(),a.end()),fb(b.begin(),b.end());
   int n=1;
   while(n<a.size()+b.size()) n<<=1;
   fa.resize(n);
   fb.resize(n);
   fft(fa,false);
   fft(fb,false);
   for(int i=0;i<n;i++) fa[i]*=fb[i];
   fft(fa,true);
   vector<int> result(n);
   for(int i=0;i<n;i++)
      result[i]=round(fa[i].real);
   return result;
}
```

NTT:

```
int primitive_root(int p) {
   vector<int> factor;
   int phi = p-1,  n = phi;

   for (int i=2; i*i<=n; ++i)
      if (n%i == 0) {
         factor.push_back (i);
         while (n%i==0)  n/=i;
      }

   if (n>1)  factor.push_back(n);

   for (int res=2; res<=p; ++res) {
      bool ok = true;
      for (int i=0; i<factor.size() && ok; ++i)
         ok &= power(res, phi/factor[i], p) != 1;
      if (ok)  return res;
   }
   return -1;
}
int nttdata(int mod, int &root, int &inv, int &pw) {
   int c = 0, n = mod-1;
   while (n%2 == 0) c++, n/=2;
   pw = (mod-1)/n;
   int g = primitive_root(mod);
   root = power(g, n, mod);
   inv = power(root, mod-2, mod);
   return c;
}
*/
const int mod ,root, inv ,pw;
/*
mod=P=c*2^k+1 & pw=2^k;
7340033, 5, 4404020, 1<<20
13631489, 11799463,6244495, 1<<20
23068673, 177147,17187657, 1<<21
463470593, 428228038, 182429, 1<<21
1214251009, 1168772018, 1178420892, 1<<21
415236097, 73362476, 247718523, 1<<22
918552577, 86995699, 324602258, 1<<22
998244353, 15311432, 469870224, 1<<23
else use primitive function for root= g^c;
inv=root^-1;
*/
void ntt(vii &a,bool invert){
   int n=a.size();
   for(int i=1,j=0;i<n;i++){
      int bit=n>>1;
      for(;j&bit;bit>>=1) j^=bit;
      j^=bit;
      if(i<j) swap(a[i],a[j]);
   }
   for(int len=2;len<=n;len<<=1){
      int wlen=invert?inv:root;
      for(int i=len;i<pw;i<<=1)
         wlen=(int)(1ll*wlen*wlen%mod);
      for(int i=0;i<n;i+=len){
         int w=1;
         for(int j=0;j<len/2;j++){
            int
u=a[i+j],v=(int)(1ll*a[i+j+len/2]*w%mod);
            a[i+j]=u+v<mod?u+v:u+v-mod;
            a[i+j+len/2]=u-v>=0?u-v:u-v+mod;
            w=(int)(1ll*w*wlen%mod);
         }
      }
   }
   if(invert){int n_1=powmod(n);for(auto &x: a)
x=(int)(1ll*x*n_1%mod);}
}
vii multiply(vii &a,vii &b){
   vii fa(a.begin(),a.end()),fb(b.begin(),b.end());
   int n=1;
   while(n<a.size()+b.size()) n<<=1;
   fa.resize(n),fb.resize(n);
   ntt(fa,false),ntt(fb,false);
   for(int i=0;i<n;i++) fa[i]=(fa[i]*1ll*fb[i]%mod);
   ntt(fa,true);
   return fa;
}
```


Convolution:

N must be power of 2

$$C_i = \sum_{j \emptyset k = i} a_j * b_k$$

a is input and A is result

```
void fwht_xor(vii &a,vii &A,bool inv=false){
   int n=a.size();
   A.assign(a.begin(),a.end());
   for (int s = 2, h = 1; s <= n; s <<= 1, h <<= 1) {
      for (int l = 0; l < n; l += s) {
         for (int i = 0; i < h; i++) {
            int t = A[l + h + i];
            A[l + h + i] = A[l + i] - t;
            A[l + i] += t;
            if (inv) A[l + h + i] /= 2, A[l + i] /= 2;
         }
      }
   }
}
void fwth_and(vii &a,vii &A,int dir=1){
   A.assign(a.begin(),a.end());
   for(int s=2,h=1;s<=n;s<<=1,h<<=1)
      for(int l=0;l<n;l+=s)
         for(int i=0;i<h;i++)
            A[l+i]+=dir*A[l+h+i];
}
void fwht_or(vii &a,vii &A,int dir=1){
   int n=a.size();
   A.assign(a.begin(),a.end());
   for(int s=2,h=1;s<=n;s<<=1,h<<=1)
      for(int l=0;l<n;l+=s)
         for(int i=0;i<h;i++)
            A[l+h+i]+=dir*A[l+i];
}
```

String:

Aho corasick:

```
typedef unordered_map<char,int> umpc;
typedef struct node{umpc to;bool leaf=false;int link;}
node;
vector<node> aho(1);
void add_string(string &s){
   int i=0;
   for(auto c: s){
      if(!aho[i].to[c])
         aho[i].to[c]=aho.size(),aho.emplace_back();
      i=aho[i].to[c];
   }
   aho[i].leaf=true;
}
void push_link(){
   vector<int> que(aho.size());
   aho[0].link=0;
   int s=0,e=1;
   que[0]=0;
   while(s<e){
      int v=que[s++];
      for(auto it: aho[v].to){
         int u=it.se,j=aho[v].link;
         char c=it.fi;
         while(j!=0 && !aho[j].to[c]) j=aho[j].link;
         if(j!=0) aho[u].link=aho[j].to[c];
         que[e++]=u;
      }
   }
}
```

Manacher:

```
void manacher(string s,vii &d1,vii &d2){
   int n=(int)s.length();
   d1.resize(n),d2.resize(n);
   for(int i=0,l=0,r=-1;i<n;i++){
      int k=(i>r)?1:min(d1[l+r-i],r-i+1);
      while(k<=i && i+k<n && s[i-k]==s[i+k]) k++;
      d1[i]=k--;
      if(i+k>r) l=i-k,r=i+k;
   }
   for(int i=0,l=0,r=-1;i<n;i++){
      int k=(i>r)?0:min(d2[l+r-i+1],r-i+1);
      while(k+1<=i && i+k<n && s[i-k-1]==s[i+k])
k++;
      d2[i]=k--;
      if(i+k>r) l=i-k-1,r=i+k;
   }
}
```

Minimum Rotation:

```
int minimumExpression(string s) {
        s = s + s;
        int i = 0, j = 1, k = 0, len = s.size();
        while(i+k<len && j+k<len){
      if(s[i+k]==s[j+k]) k++;
      else if(s[i+k]<s[j+k]){j=max(j+k+1,i+1);k=0;}
      else{i=max(i+k+1,j+1);k=0;}
        }
        return min(i, j);
}
```

Palindrome Tree:

```
const int N = 1e5+10;
typedef unordered_map<char,int> umpc;
umpc tree[N];
int idx;
int len[N], link[N], last;
char s[N];
inline void pre_comp()
{len[1]=-1,link[1]=link[2]=1,len[2]=0,idx=last=2;}
void extend(int p){
  while(s[p-len[last]-1] != s[p]) last=link[last];
  int x=link[last];char c=s[p];
  while(s[p-len[x]-1] != s[p]) x=link[x];
  if(!tree[last][c]) {
    tree[last][c]= ++idx;
    len[idx]=len[last]+2;
    link[idx]=len[idx]==1?2:tree[x][c];
  } last = tree[last][c];
}
```

Trie:

```
typedef struct node{
   unordered_map<char,int> next;
   bool leaf=false;
}node;
vector<node> trie;
void add_string(string &s){
   int i=0;
   for(char c: s){
      if(trie[i].next[c]==0){
         trie[i].next[c]=trie.size();
         trie.emplace_back();
      }
      i=trie[i].next[c];
   }
   trie[i].leaf=true;
}
bool search_string(string &s){
   int i=0;
   for(char c: s){
      if(trie[i].next[c]==0) return false;
      i=trie[i].next[c];
   }
   return true;
}
```

Kmp:

```
vector<int> prefix_function(string s) {
   int n = (int)s.length();
   vector<int> pi(n);
   for (int i = 1; i < n; i++) {
      int j = pi[i-1];
      while(j>0 && s[i]!=s[j]) j=pi[j-1];
      if (s[i]==s[j]) j++;
      pi[i] = j;
   }
   return pi;
}
```

Z:

```
vector<int> z_function(string s) {
   int n = (int) s.length();
   vector<int> z(n);
   for (int i = 1, l = 0, r = 0; i < n; ++i) {
      if(i<=r) z[i]=min(r-i+1,z[i-l]);
      while(i+z[i]<n && s[z[i]]==s[i+z[i]]) ++z[i];
      if(i+z[i]-1>r) l=i, r=i+z[i]-1;
   }
   return z;
```

```
}
```

Suffix Array:

```
#include<bits/stdc++.h>
using namespace std;
#define MAX 1000005

int wa[MAX],wb[MAX],wv[MAX],Ws[MAX];
int cmp(int *r,int a,int b,int l) {return r[a]==r[b] && r[a+l]==r[b+l];}

//(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[MAX],Rank[MAX],LCP[MAX];

//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;
}

int main(){
    string s="abababab";
    int n=s.size();
    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]<<" ";
}
```

Finding Repetition:

```
vector<int> z_function(string const& s) {
    int n = s.size();
    vector<int> z(n);
    for (int i = 1, l = 0, r = 0; i < n; i++) {
        if (i <= r)
            z[i] = min(r-i+1, z[i-l]);
        while (i + z[i] < n && s[z[i]] == s[i+z[i]])
            z[i]++;
        if (i + z[i] - 1 > r) {
            l = i;
            r = i + z[i] - 1;
        }
    }
    return z;
}
int get_z(vector<int> const& z, int i) {
    if (0 <= i && i < (int)z.size()) return z[i];
    else return 0;
}
vector<pair<int, int>> repetitions;
void convert_to_repetitions(int shift, bool left, int cntr, int l, int k1, int k2) {
    for (int l1 = max(1, l - k2); l1 <= min(l, k1); l1++) {
        if (left && l1 == l) break;
        int l2 = l - l1;
        int pos = shift + (left ? cntr - l1 : cntr - l - l1 + 1);
        repetitions.emplace_back(pos, pos + 2*l - 1);
    }
}

void find_repetitions(string s, int shift = 0) {
    int n = s.size();
    if (n == 1) return;
    int nu = n / 2;
    int nv = n - nu;
    string u = s.substr(0, nu);
    string v = s.substr(nu);
    string ru(u.rbegin(), u.rend());
    string rv(v.rbegin(), v.rend());
    find_repetitions(u, shift);
    find_repetitions(v, shift + nu);
    vector<int> z1 = z_function(ru);
    vector<int> z2 = z_function(v + '#' + u);
    vector<int> z3 = z_function(ru + '#' + rv);
    vector<int> z4 = z_function(v);
    for (int cntr = 0; cntr < n; cntr++) {
        int l, k1, k2;
        if (cntr < nu) {
            l = nu - cntr;
            k1 = get_z(z1, nu - cntr);
            k2 = get_z(z2, nv + 1 + cntr);
        } else {
            l = cntr - nu + 1;
            k1 = get_z(z3, nu + 1 + nv - 1 - (cntr - nu));
            k2 = get_z(z4, (cntr - nu) + 1);
        }
        if (k1 + k2 >= l)
            convert_to_repetitions(shift, cntr < nu, cntr, l, k1, k2);
    }
}
```

GEOMETRY:

BASIC: (for int all double should be integers)

```
inline double dot(point a, point b)
{
   return a.x*b.x+a.y*b.y;
}
inline double cross(point a, point b)
{
   return a.x*b.y-b.x*a.y;
}
inline double dist( point a, point b)
{
   return sqrt(a.x*b.x+a.y*b.y);
}
inline double dist(point &p,line &l){
   return
abs(l.a*p.x+l.b*p.y+l.c)/sqrt(l.a*l.a+l.b*l.b);
}
inline long double area(point a, point b, point c)
{
   Return a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-
b.y);
}
inline int sgn(double &x){return (x>0)-(x<0);}
```

INTERSECTIONS:

```
vector<point> line_line(line p,line q){
   if((p.a*q.b-p.b*q.a)<eps) return {};
   return {(q.a*p.c-p.a*q.c)/(p.a*q.b-
q.a*p.b),(q.b*p.c-p.b*q.c)/(p.b*q.a-q.b*p.a)};
}
vector<point> circle_line(circle &c,line &l){
   double d=dist(l,c.c);
   if(d>r+eps) return {};
   double t=sqrt(c.r*c.r-d*d),m=-l.a/l.b;
   double r=t/sqrt(m*m+1.00000000000);
   point cn=line_line(l,l.norm(c.c)).back();
   if(abs(d-r)<eps) return {cn};
   point del={r,m*r};
   return {cn-del,cn+del};
}
vector<point> circle_circle(circle &p,circle &q){
   double d=dist(p.c,p.c);
   if(d>p.r+q.r+eps || d<abs(p.r-q.r)-eps) return
{};
   point
pt=point(p.c.x*q.r+q.c.x*p.r,p.c.y*q.r+q.c.y*p.r)
/(p.r+q.r);
   return circle_line(p,line(p.c,q.c).norm(pt);
}
```

DOUBLE:

```
struct point
{
   double x, y;
   point (double x = 0, double y = 0): x(x), y(y)
{}
   bool operator == (const point& u) const
   {
      return abs(x - u.x) < eps && abs(y - u.y) <
eps;
   }
   bool operator < (const point& u) const
   {
      return (u.x-x>eps) || (abs(x-u.x)<eps &&
(u.y-y)>eps);
   }
   point operator + (const point& u)
   {
      return point(x + u.x, y + u.y);
   }
   point operator - (const point& u)
   {
      return point(x - u.x, y - u.y);
   }
   point operator * (const double u)
   {
      return point(x * u, y * u);
   }
   point operator / (const double u)
   {
      return point(x / u, y / u);
   }
   bool operator <= (const point& u) const
   {
      return *this < u || *this == u;
   }
};
typedef struct line
{
   double a,b,c;
   line (double a,double b,double c): a(a),b(b),c(c){}
   line (point p,double m) : a(m),b(-1),c(p.y-m*p.x){}
   line (point p,point q)
   {
      a=p.y-q.y;
      b=q.x-p.x;
      double z=sqrt(a*a+b*b);
      a/=z,b/=z;
      c=-a*p.x-b*p.y;
   }
   inline bool online(const point &p)
   {
      return abs(a*p.x+b*p.y+c)<eps;
   }
   inline line norm(point t={0,0}){
      return line(b,-a,a*t.y-b*t.x);
   }
} line;
typedef struct circle{
   double r;
   point c;
   circle(double r,point c): r(r),c(c){}
} circle:
```

CIRCUMCIRCLE: (INTEGERS/DOUBLE)

```
circle mec(vector<point> &P,vector<point>
R={}){
   int n=(int)P.size();
   if(n==0 || R.size()==3){
      if(R.size()==1) return {0,R[0]};
      if(R.size()==2) return
{dist(R[0],R[1])/2,(R[0]+R[1])/2};
      for(int i=1;i<3;i++) R[i]-=R[0];
      point b=R[1],c=R[2];
      double B=dot(b,b),C=dot(c,c),D=cross(b,c);
      point c={(c.y*B-b.y*C)/(2*D),(b.x*C-
c.x*B)/(2*D)};
      return {dist(c,R[1]),c+R[0]};
   }
   point d= P.back();
```

```
    P.pop_back();
    circle T=mec(P,R);
    if(dist(d,T.cen)<T.r-eps) return T;
    R.push_back(d);
    return mec(P,R);
}
```

NEAREST POINT PAIR:

```
vector<point> a;
pair<point,point> ans;
double D=1e18;
inline void upd_ans(point &p,point &q){
    if(dist(p,q)<D) D=dist(p,q),ans={p,q};
}
inline bool cmp_y(point &a,point &b){
    return a.y<b.y;
}
vector<point> t;
void rec(int l,int r){
    if(r-l<=3){
        for(int i=l;i<=r;i++)
            for(int j=i+1;j<=r;j++)
                upd_ans(a[i],a[j]);
        sort(a.begin()+l,a.end().r,cmp_y);
    }
    int m=(l+r)>>1;
    int mdx=a[m].x;
    rec(l,m),rec(m,r);

merge(a.begin()+l,a.begin()+m,a.begin()+m,a.be
gin()+r,t.begin(),cmp_y);
    copy(t.begin(),t.begin()+r-l,a.begin()+l);
    int tst=0;
    for(int i=l;i<r;i++){
        if(abs(a[i].x-midx)>=D) continue;
        for(int j=tst-1;j>=0 && a[i].y-t[j].y<D;j--)
            upd_ans(a[i],a[j]);
        t[tst++]=a[i];
    }
}
```

MINKOWSKI SUM:

```
//for summing two convex polygon P,Q
vector<point> minkowski(vector<point>
P,vector<point> Q){
    P.pb(P[0]),P.pb(P[1]);
    Q.pb(Q[0]),Q.pb(Q[1]);
    vector<point> result;
    int i=0,j=0;
    while(i<P.size()-2 || j<Q.size()-2){
        result.pb(P[i]+Q[i]);
        auto crs = cross(P[i+1]-P[i],Q[j+1]-Q[j]);
        if(crs>=0) i++;
        if(crs<=0) j++;
    }
    return result;
}
```

MAX/MIN DIST AT A DIRECTION:

```
int max_dist(vector<int> &P,point &d){
    int n=P.size();
    int l=1,r=n-1;
    while(l<r){
        int m2=l+(2*(r-l))/3,m1=l+(r-l)/3;
        if(l==r-1) m1=l,m2=r;
        if(dot(P[m1],d)>=dot(P[m2],d)) l=m1;
        else r=m2;
    }
    return l;
}
```

CONVEX HULL:

```
void convex_hull(vector<point>& a) {
    if (a.size() == 1)
        return;
    sort(a.begin(), a.end());
    point p1 = a[0], p2 = a.back();
    vector<point> up, down;
    up.push_back(p1);
    down.push_back(p1);
    for (int i = 1; i < (int)a.size(); i++) {
        if (i == a.size() - 1 || area(p1, a[i], p2)<0) {
            while (up.size() >= 2 &&
area(up[up.size()-2], up[up.size()-1], a[i])>=0)
                up.pop_back();
            up.push_back(a[i]);
        }
        if (i == a.size() - 1 || area(p1, a[i], p2)>0) {
            while(down.size() >= 2 &&
area(down[down.size()-2], down[down.size()-1],
a[i])<=0)
                down.pop_back();
            down.push_back(a[i]);
        }
    }
    a.clear();
    for (int i = 0; i < (int)up.size(); i++)
        a.push_back(up[i]);
    for (int i = down.size() - 2; i > 0; i--)
        a.push_back(down[i]);
}
```

All point pair:

```
vector<pair<int,int>> a;
bool cmp_point(const pii i,const pii j){
    pii u=mp(a[i.se].fi-a[i.fi].fi,a[i.se].se-a[i.fi].se);
    pii v=mp(a[j.se].fi-a[j.fi].fi,a[j.se].se-a[j.fi].se);
    return u.fi*1ll*v.se<u.se*1ll*v.fi;
}
int main(){
    //take necessary input: points in vector a

    sort(a.begin(),a.end());
    vii p(n); //current position of a point
    for(i=0;i<n;i++) p[i]=i;
    vector<pair<int,int>> rot;
    for(i=0;i<n;i++)
        for(j=i+1;j<n;j++) rot.pb({i,j});
    sort(rot.begin(),rot.end(),cmp_point);

    for(auto d: rot){
        //do operation for points d.fi & d.se
        //p[i] means cur pos of ith point in a

        swap(p[d.fi],p[d.se]);
    }
}
```

Pick's Theorem:

S=I + B/2 -1,S=area of polygon
I=Internal Lattice points
B=Boundary Lattice points

Data Structure:

Sparse Table:

```
struct sparseTable
{
   vector<vi> table;
   int z;
   sparseTable(vi &arr)
   {
      int n =(int) arr.size();
      z=log2(n)+1;
      table = vector<vi> (n, vi(z));
      for (int i = 0; i < n; i++)
         table[i][0] = arr[i];
      for (int j = 1; j <= z; j++)
         for (int i = 0; i + (1 << j) <= n;
               ,! i++)
            table[i][j] = min(table[i][j-1],
                        table[i + (1 << (j - 1))][j -
                              1]);
      ,!
      ,!
   }
   int query(int x, int y)
   {
      int po=log2(y-x+1);
      int
      ,! m=min(table[x][po],table[y-(1<<po)+1][po])
      return m;
   }
};
```

Fenwick2D(point update range sum):

```
struct fenwick2D
{
   int n, m;
   vector<vector<ll>> bit;
   fenwick2D(int n, int m)
   {
      bit.assign(n+1, vector<ll>(m+1));
      this->n = n;
      this->m = m;
   }
   void update(int x, int y, ll val)
   {
      for (; x<=n; x+=x&-x)
         for (int j=y; j<=m; j += j&-j)
            bit[x][j] += val;
   }
   ll get(int x, int y)
   {
      ll ans = 0;
      for (; x; x-=x&-x)
         for (int j=y; j; j -= j&-j)
            ans += bit[x][j];
      return ans;
   }
   ll get(int x1, int y1, int x2, int y2)
   {
      return get(x2, y2) - get(x1-1, y2) -
            ,! get(x2, y1 - 1) + get(x1-1, y1-1);
   }
};
```

Fenwick1D (range sum point update):

```
struct FT
{
   vi BIT;
   int N;
   FT(int n)
   {
      N = n+5;
      BIT = vi(N, 0);
   }
   void update(int x,int val)
   {
      ++x;
      while(x<=N)
         ,! { BIT[x]+=val ; x+=(x&-x); }
      }
   int query(int x)
   {
      ++x;
      int res=0;
      while(x>0)
      {
         res+=BIT[x];
         x-=(x&-x);
      }
      return res;
   }
   ,!
   ,!
   int query(int l, int r)
   {
      return query(r) -
            ,! query(l-1);
   }
};
```

Range Update Range Max:

```
struct SegmentTree
{
   vector<ll> arr, tree, lazy;
   int n;
   SegmentTree(const vector<ll>& in_arr)
   {
      n = in_arr.size();
      arr = vector<ll>(in_arr);
      tree = vector<ll>(4*n+5);
      lazy = vector<ll>(4*n+5);
      build_tree(1, 0, n-1);
   }
   void build_tree(int node, int a, int b)
   {
      if(a > b) return; // Out of range
      if(a == b)   // Leaf node
      {
         tree[node] = arr[a]; // Init value
         return;
      }
      build_tree(node*2,a,(a+b)/2);  // Init left
child
      build_tree(node*2+1,  1+(a+b)/2,  b);  //Init
right child
      tree[node]  =  max(tree[node*2],
tree[node*2+1]); // Init root value
   }
   void update(int i, int j, int val)
   {
```

```
    update_tree(1, 0, n-1, i, j, val);
  }
  /**Increment elements within range [i, j] with value value*/
  void update_tree(int node, int a, int b, int i, int j, ll value)
  {
    if(lazy[node] != 0)  // This node needs to be updated
    {
      tree[node] += lazy[node]; // Update it
      if(a != b)
      {
        lazy[node*2] += lazy[node]; //Mark child as lazy
        lazy[node*2+1] += lazy[node]; // Mark child as lazy
      }
      lazy[node] = 0; // Reset it
    }
    if(a > b || a > j || b < i) // Current segment is not within range [i, j]
      return;
      if(a >= i && b <= j)  // Segment is fully within range
    {
      tree[node] += value;
      if(a != b)  // Not leaf node
      {
        lazy[node*2] += value;
        lazy[node*2+1] += value;
      }
      return;
    }
    update_tree(node*2, a, (a+b)/2, i, j,value); // Updating left child
    update_tree(1+node*2, 1+(a+b)/2, b, i, j,value); // Updating right child
    tree[node] = max(tree[node*2],tree[node*2+1]); // Updating root with max value
  }
    ll query(int i, int j)
    {
      return query_tree(1, 0, n-1, i,j);
    }
    /**Query tree to get max element value within range [i, j]*/
    ll query_tree(int node, int a, int b, int i,int j)
    {
      if(a > b || a > j || b < i) return LLONG_MIN; // Out of range
      if(lazy[node] != 0)  // This node needs to be updated
      {
        tree[node] += lazy[node]; // Update it
        if(a != b)
        {
          lazy[node*2] += lazy[node]; //Mark child as lazy
          lazy[node*2+1] += lazy[node]; //Mark child as lazy
        }
        lazy[node] = 0; // Reset it
      }
      if(a >= i && b <= j) // Current segment is totally within range [i, j]
        return tree[node];
      ll q1 = query_tree(node*2, a, (a+b)/2, i, j); // Query left child
      ll q2 = query_tree(1+node*2, 1+(a+b)/2, b,i, j); // Query right child
      ll res = max(q1, q2); // Return final result
      return res;
    }
  };
```

Slope Trick:

*****don't use pragma with slope trick**

```
#include<bits/stdc++.h>
using namespace std;
template<typename T>
struct slope_trick{
  const T inf = numeric_limits<T>::max()/3;
  priority_queue<T,vector<T>,less<T>> L;
  priority_queue<T,vector<T>,greater<T>> R;
  T xl=0,xr=0,min_f=0;
  slope_trick(){L.push(-inf),R.push(inf);}
  inline void left_func(const T c){
    min_f+=max((T)0,c-R.top()-xr);
    if(c+xr<=R.top()) L.push(c-xl);
    else{R.push(c-xr);L.push(R.top()+xr-xl);R.pop();}
  }
  inline void right_func(const T c){
    min_f+=max((T)0,L.top()+xl-c);
    if(c+xl>=L.top()) R.push(c-xr);
    else{L.push(c-xl);R.push(L.top()+xl-xr);L.pop();}
  }
};
#define slope_trick struct slope_trick
```


CHT:

```
// convex hull trick is used when for different lines we have to find answer at specific point
// this can be used Specifically when slope of inserting lines ARE INCREASING!!!
// for minimum lower convex hull is build & vice-versa

typedef int ftype;
typedef struct point{
  ftype x,y;
  point(ftype x,ftype y): x(x),y(y){};
  inline point conj(){return point(x,-y);};
  inline ftype dot(const point &b){return x*b.x+y*b.y;};
  inline ftype cross(const point &b){return x*b.y-y*b.x;}
  point operator +(const point b){return point(x+b.x,y+b.y);};
  point operator -(const point b){return point(x-b.x,y-b.y);};
  point operator *(const point b){return point(x*b.x-y*b.y,x*b.y+y*b.x);}
}point;
vector<point> hull,vecs;
```

```
void add_line(ftype k,ftype b){
   point nw(k,b);
   while(!vecs.empty() && vecs.back().dot(nw-hull.back())<0){
      hull.pop_back();
      vecs.pop_back();
   }
   if(!hull.empty()) vecs.push_back(point(0,1)*(nw-hull.back()));
   hull.push_back(nw);
}
ftype get(ftype x){
   point query = {x,1};
   auto it=lower_bound(vecs.begin(),vecs.end(),query,[](point a,point b){
            return cross(a,b)>0;
         });
   return dot(query.hull[it-vecs.begin()]);
}
```

LiChao tree(1D):

```
// this is an example for minimum extraction
// keys:   fi,se
typedef struct line{
   int first;
   int second;
} line;

//  cht.assign(4*n+4,mp(0,INT_MAX));
//  *for maximum use mp(0,-INT_MAX);
//  **beware of long long or int
vector<line> cht;

inline int f(line u,int p){ return u.fi*p+u.se;}
void line_in(line ln,int s,int e,int ind=0){
   int m=s+(e-s)/2;
   //for maximum use > in both lines....
   bool left=f(ln,s)<f(cht[ind],s);
   bool mid=f(ln,m)<f(cht[ind],m);
   if(mid) swap(ln,cht[ind]);
   if(s==e) return;
   if(mid==left) line_in(ln,m+1,e,2*ind+2);
   else line_in(ln,s,m,2*ind+1);
}
int get(int p,int s,int e,int ind=0){
   if(s==e) return f(cht[ind],p);
   int m=s+(e-s)/2;
   //for maximum use max...
   if(p<=m) return min(f(cht[ind],p),get(p,s,m,2*ind+1));
   else return min(f(cht[ind],p),get(p,m+1,e,2*ind+2));
}
```

Policy based:

```
#include <ext/pb_ds/assoc_container.hpp> // Common file
#include <ext/pb_ds/tree_policy.hpp>
#include <functional> // for less
using namespace __gnu_pbds;
using namespace std;

typedef tree<int,null_type,less<int>,rb_tree_tag,tree_order_statistics_node_update> new_data_set;


p.insert(x);
*p.find_by_order(nth);
p.find_by_key(x);
```

LiChao tree(2D):

```
typedef struct plane{
   int fi;
   int se;
   int th;
} pln;

vector<pln> cht;
//      cht.assign(6*n+6,{0,0,INT_MAX}) :::  n=|x|*|y|
//      *for maximum use {0,0,-INT_MAX}
//      ** beware of long long or int
//           z= u.fi*x  +u.se*y  +u.th
inline int f(pln u,int x,int y){ return u.fi*x+u.se*y+u.th;}
void pln_in(pln pl,int sx,int ex,int sy,int ey,int ind=0){
   if(sx>ex || sy>ey) return;
   int mx=sx+(ex-sx)/2,  my=sy+(ey-sy)/2;
   //for maximum use > in 3 lines.....
   bool mid=f(pl,mx,my)<f(cht[ind],mx,my);
   int id=0;
   if(f(pl,mx,sy)<f(cht[ind],mx,sy)) id&=(1<<1);
   if(f(pl,sx,my)<f(cht[ind],sx,my)) id&=1;
   if(mid) swap(pl,cht[ind]) , id&=(1<<2);
   if(sx==ex && sy==ey) return;
   if(id!=7 && id!=0) pln_in(pl,sx,mx,sy,my,4*ind+1);
   if(id!=6 && id!=1) pln_in(pl,mx+1,ex,sy,my,4*ind+2);
   if(id!=5 && id!=2) pln_in(pl,sx,mx,my+1,ey,4*ind+3);
   if(id!=4 && id!=3) pln_in(pl,mx+1,ex,my+1,ey,4*ind+4);
}

int get(int x,int y,int sx,int ex,int sy,int ey,int ind=0){
   if(sx>ex || sy>ey) return INT_MAX;
   int mx=sx+(ex-sx)/2,    my=sy+(ey-sy)/2;
   if(x<=mx && y<=my) return min(f(cht[ind],x,y),get(x,y,sx,mx,sy,my,4*ind+1));
   if(x>mx && y<=my) return min(f(cht[ind],x,y),get(x,y,mx+1,ex,sy,my,4*ind+2));
   if(x<=mx && y>my) return min(f(cht[ind],x,y),get(x,y,sx,mx,my+1,ey,4*ind+3));
   if(x>mx && y>my) return min(f(cht[ind],x,y),get(x,y,mx+1,ex,my+1,ey,4*ind+4));
}
```

Augmented DSU:

```
#include<bits/stdc++.h>
using namespace std;
const int N=10000;
int flaw;
//counting numbers of inconsistent assertions
int pot[N],prec[N];
void initialize(int n)
{
        flaw=0;
        for(int i=1;i<=n;++i){ prec[i]=i;pot[i]=0;}
}
int find(int x)
```

```
{
        if(prec[x]==x) return x;
        int rx=find(prec[x]);   // rx is the root of x
        pot[x]=pot[prec[x]]+pot[x];
        prec[x]=rx;
        return rx;
}

void merge(int a,int b,int d)
{
        int ra=find(a); int rb=find(b);
        if(ra==rb && pot[a]-pot[b]!=d) flaw++;
        else if(ra!=rb){
    pot[ra]=d+pot[b]-pot[a];
                prec[ra]=rb;
        }
}
int main()
{
        int n; cin>>n;  //no. of variables
        int m; cin>>m;  // no. of equations
         //consider 1-based indexing of variables
        initialize(n);
        for(int i=1;i<=m;++i)
        {
                int a,b,d;cin>>a>>b>>d;
                merge(a,b,d);//asserting a-b=d;
        }
        cout<<"No. of inconsistencies= "<<flaw;
        return 0;
}
```

MO Algorithm:

```
int block;
vii a;
struct queries{
   int l,r,id;
   bool operator <(const queries &d) const&{
      return (l/block==d.l/block?r<d.r:l<d.l);
   }
} *query;
ll sum=0;
inline void add(int n,vii &count){
   sum+=(count[n]*2ll+1)*n;
   count[n]++;
   return;
}
inline void del(int n,vii &count){
   count[n]--;
   sum-=(count[n]*2ll+1)*n;
   return;
}
int main(){
   ios_base::sync_with_stdio(false);
   cin.tie(NULL);
   cout.tie(NULL);
   int n,m,i,j,k,q;
   cin>>n>>q;
   a.resize(n);
   for(i=0;i<n;i++) cin>>a[i];
   block=(int)(sqrt(n));
   query=new struct queries [q];
   for(i=0;i<q;i++){
      cin>>query[i].l>>query[i].r;
      query[i].l--;
      query[i].r--;
      query[i].id=i;
   }
   sort(query,query+q);
   struct queries qr;
   vii count((int)(1e6+1),0);
   for(i=query[0].l;i<=query[0].r;i++) add(a[i],count);
   int L=query[0].l;
   int R=query[0].r;
   ll ans[q];
   i=0;
   while(i<q){
      qr=query[i++];
      while(L<qr.l) del(a[L++],count);
      while(R>qr.r) del(a[R--],count);
      while(L>qr.l) add(a[--L],count);
      while(R<qr.r) add(a[++R],count);
      ans[qr.id]=sum;
   }
   for(i=0;i<q;i++) cout<<ans[i]<<endl;
}
```

Enumerating all submask:

```
for (int m=0; m<(1<<n); ++m)
   for (int s=m; s; s=(s-1)&m)
 ... s and m ...
```

Tree:

Heavy-light decomposition:

```
#include <bits/stdc++.h>
#define pb              push_back
#define endl "\n"
#define fill(x, y)     memset(x, y, sizeof(x))
#define all(x)         (x).begin(), (x).end()
#define debug(x)        { cerr << #x << " = " << x << endl; }
#define IO                {
ios_base::sync_with_stdio(false); cin.tie(0); }
#define read(x)           freopen(x, "r", stdin)
#define write(x)       freopen(x, "w", stdout)
using namespace std;
typedef long long      ll;
typedef pair<int, int> ii;
typedef vector<int>    vi;
vector<int> parent, depth, heavy, head, pos;
int cur_pos;
int dfs(int v, vi adj[]) {
int size = 1;
int max_c_size = 0;
for (int c : adj[v]) {
if (c != parent[v]) {
parent[c] = v, depth[c] = depth[v] + 1;
int c_size = dfs(c, adj);
size += c_size;
if (c_size > max_c_size)
max_c_size = c_size, heavy[v] = c;
}
}
return size;
}
void decompose(int v, int h, vi adj[]) {
head[v] = h, pos[v] = cur_pos++;
if (heavy[v] != -1)
decompose(heavy[v], h, adj);
for (int c : adj[v]) {
if (c != parent[v] && c != heavy[v])
decompose(c, c, adj);
}
}
struct SegmentTree {
vector<ll> t;
int n;
inline ll ops(ll a, ll b) {return (a+b);}
SegmentTree(vector<ll> &a) {
n = a.size();
t.resize(2*n);
for (int i = 0; i < n; i++)
t[n + i] = a[i];
for (int i = n - 1; i >= 1; i--)
t[i] = ops(t[2 * i],t[2 * i + 1]);
}
void update(int p, ll value)
{
for (t[p += n] = value; p > 1; p >>= 1) t[p>>1] =
ops(t[p] , t[p^1]);
}
ll get( int l, int r)
{
ll res = 0;
for (l += n, r += n + 1; l < r; l >>= 1, r >>= 1) {
if (l&1) res = ops(res, t[l++]);
if (r&1) res = ops(t[--r], res);
}
return res;
}
};
ll moveUp(int u, int v, SegmentTree& st) { // move
from u to v
if (depth[v] >= depth[head[u]]) {
int r = pos[u], l = pos[u] - (depth[u] - depth[v]);
return st.get(l, r);
} else {
int l = pos[head[u]], r = pos[u];
ll ans = st.get(l, r);
ans = (ans + moveUp(parent[head[u]], v, st));
return ans;
}
}
int main() {
IO;
int n, q; cin >> n;
vector<vi> edges;
map<ii, int> cst;
vi adj[n+1];
for (int i=1; i<n; i++) {
int u, v, w; cin >> u >> v >> w;
adj[u].pb(v);
adj[v].pb(u);
cst[{u, v}] = cst[{v, u}] = w;
edges.push_back({u, v, w});
}
parent.assign(n+1, 0);
depth.assign(n+1, 0);
heavy.assign(n+1, -1);
head.assign(n+1, 0);
pos.assign(n+1, 0);
cur_pos = 0;
dfs(1, adj);
decompose(1, 1, adj);
int go[n+1][20];
fill(go, 0);
for (int i=1; i<=n; i++) go[i][0] = parent[i];
for (int j=1; j<20; j++) {
for (int i=1; i<=n; i++) {
go[i][j] = go[go[i][j-1]][j-1];
}
}
vector<ll> c(n+10);
for (int i=1; i<=n; i++) {
c[pos[i]] = cst[{i, parent[i]}];
}
SegmentTree st(c);
}
```

Centroid decomposition:

```
    /*calling decomp would start the work
    where dfs is to travel through all the subtree
    */


const pii cendef=mp(10000000,10000000);
pii cen=cendef;
//initially all si=0 & block=false
vector<int> si;
vector<bool> block;
int find_centre(int u,int p,int s)
{
   int count=1,m=0;
   for(auto i: g[u])
   {
      if(i==p || block[i])
```

```
            continue;
        int k=find_centre(i,u,s);
        count+=k;
        m=max(m,k);
    }
    m=max(m,s-count);
    cen=min(cen,mp(m,u));
    si[u]=count;
    return count;
}
void dfs(int u,int p /*other parameters*/)
{
    //include things for current node u
    for(auto i: g[u])
    {
        if(i==p || block[i])
            continue;
        dfs(i,u,in,out,dis+1);
    }
    return;
}
void decomp(int root,int par,int s)
{
    cen=cendef;
    find_centre(root,par,s);
    int c=cen.se;
    // do works across centre c including dfs for all the
subtree
    block[c]=true;
    for(auto i: g[c])
    {
        if(block[i])
            continue;
        if(si[i]>si[c]) si[i]=s-si[c];
        decomp(i,c,si[i]);
    }
    return;
}
```

Binary Lifting:

```
vector<int> *g;

int l;

int T;
vii tin,tout;
vector<vii> up;

void dfs(int u,int p){
    tin[u]=++T;
    up[u][0]=p;
    for(int i=1;i<=l;++i) up[u][i]=up[up[u][i-1]][i-1];
    for(auto i: g[u]) if(i!=p) dfs(i,u);
    tout[u]=++T;
}
#define is_ancestor(u,v) (tin[u]<=tin[v] &&
tout[u]>=tout[v])
int lca(int u,int v){
    if(is_ancestor(u,v)) return u;
    if(is_ancestor(v,u)) return v;
    for(int i=l;i>=0;i--) if(!is_ancestor(up[u][i],v))
u=up[u][i];
    return up[u][0];
}
void preprocess(int root,int n){
    tin.resize(n+1);
    tout.resize(n+1);
    T=0;
    l=ceil(log2(n));
    up.assign(n,vii(l+1));
    dfs(root,root);
}
```

LCA:

```
int n, l;
vector<vector<int>> adj;

int timer;
vector<int> tin, tout;
vector<vector<int>> up;

void dfs(int v, int p)
{

    tin[v] = ++timer;
    up[v][0] = p;
    for (int i = 1; i <= l; ++i)
        up[v][i] = up[up[v][i-1]][i-1];

    for (int u : adj[v]) {
        if (u != p)
            dfs(u, v);
    }

    tout[v] = ++timer;
}

inline bool is_ancestor(int u, int v)
{
    return tin[u] <= tin[v] && tout[u] >= tout[v];
}

int lca(int u, int v)
{
    if (is_ancestor(u, v))
        return u;
    if (is_ancestor(v, u))
        return v;
    for (int i = l; i >= 0; --i) {
        if (!is_ancestor(up[u][i], v))
            u = up[u][i];
    }
    return up[u][0];
}

void preprocess(int root) {
    tin.resize(n+1);
    tout.resize(n+1);
    timer = 0;
    l = ceil(log2(n));
    up.assign(n+1, vector<int>(l + 1));
    dfs(root, root);
}
```

DSU:

struct DSU

```
{
   vi Arr, sz;
   int root (int i){
   while(Arr[ i ] != i){Arr[i] = Arr[ Arr[i] ] ;i=Arr[i];}
   return i;
   }
   DSU(int n){
      Arr = vi(n+5);
      sz = vi(n+5);
      for(int i = 0; i<n; i++){Arr[i] = i ;sz[i] = 1;}
   }
   void Union(int A,int B)
   {
      int root_A = root(A), root_B = root(B);
      if(sz[root_A] < sz[root_B ]){
         Arr[ root_A ] = Arr[root_B];
         sz[root_B] += sz[root_A];
      }
      else{Arr[root_B] = Arr[root_A];sz[root_A] += sz[root_B];}
   }
};
```

Rollback:

```
int par[N], sz[N];
int parent(int u){
   while(par[u]!=u) u=par[u];
   return u;
}
vii history;
bool connect(int u,int v){
   u=parent(u);
   v=parent(v);
   if(u==v) return false;
   if(sz[u]>sz[v]) swap(u,v);
   par[u]=v;
   sz[v]+=sz[u];
   history.pb(u);
   return true;
}
void rollback(){
   int u=history.back();
   history.pop_back();
   int v=par[u];
   par[u]=u;
   sz[v]-=sz[u];
}
void init(int n){
   for(int i=1;i<=n;i++) par[i]=i,sz[i]=1;
}
```

Sack:

```
// dsu on tree:: finding the sum of all dominating color in a subtree
vii sz,*g,cnt,col,in;
vpii tim;
int T=0;
int size_dfs(int u,int p){
   tim[u].fi=++T;
   in[T]=u;
   int &s=sz[u]=1;
   for(auto i: g[u]) if(i!=p) s+=size_dfs(i,u);
   tim[u].se=T;
   return s;
}
vector<ll> ans;
pair<int,ll> dfs(int u,int p,bool keep){
   pair<int,ll> answer;
   int post=-1,mx=0;
   for(auto i: g[u]) if(i!=p) if(sz[i]>mx) mx=sz[i],post=i;
   for(auto i : g[u]) if(i!=p && i!=post) dfs(i,u,false);
   if(post!=-1) answer=dfs(post,u,true);
   else answer={0,0};
   cnt[col[u]]++;
   if(cnt[col[u]]>answer.fi){
      answer.fi=cnt[col[u]];
      answer.se=col[u];
   }
   else if(cnt[col[u]]==answer.fi){
      answer.se+=col[u];
   }

   for(auto i: g[u]){
      if(i!=p && i!=post){
         for(int t=tim[i].fi;t<=tim[i].se;t++){
            cnt[col[in[t]]]++;
            if(cnt[col[in[t]]]>answer.fi){
               answer.fi=cnt[col[in[t]]];
               answer.se=col[in[t]];

            }
            else if(cnt[col[in[t]]]==answer.fi){
               answer.se+=col[in[t]];
            }
         }
      }
   }
   ans[u]=answer.se;
   if(!keep){
      for(int t=tim[u].fi;t<=tim[u].se;t++) cnt[col[in[t]]]--;
   }
   return answer;
}
```

Dynamic Connectivity Offline:

Query: +(u v) -(u v) ?(no of components)

```
#pragma GCC optimize("Ofast")
#pragma GCC target("avx,avx2,fma")
#pragma GCC optimization ("unroll-loops")
#include<bits/stdc++.h>
using namespace std;
typedef long long int ll;
typedef pair<int,int> pii;
typedef vector<int> vii;
typedef vector<pii> vpii;
typedef unordered_map<int,int> umap;
typedef long double ld;
#define fi first
#define se second
#define pb push_back
#define mp make_pairNa
#define popcount __builtin_popcount
#define case cout<<"Case "<<z-t<<": ";
#define delay(n) clock_t start=clock();while(clock()<start+1000*n);
#define N 300005
int par[N], sz[N];
int parent(int u){
```

```
    while(par[u]!=u) u=par[u];
    return u;
}
vii history;
bool connect(int u,int v){
    u=parent(u);
    v=parent(v);
    if(u==v) return false;
    if(sz[u]>sz[v]) swap(u,v);
    par[u]=v;
    sz[v]+=sz[u];
    history.pb(u);
    return true;
}
void rollback(){
    int u=history.back();
    history.pop_back();
    int v=par[u];
    par[u]=u;
    sz[v]-=sz[u];
}
void init(int n){
    for(int i=1;i<=n;i++) par[i]=i,sz[i]=1;
}
typedef struct edge{
    int u,v,t;
    bool operator <(const edge &e) const{
        return u==e.u? (v==e.v? abs(t)<abs(e.t) : v<e.v) :
u<e.u;
    }
} edge;
vii ans;
typedef struct node{
    vii ed;
    bool qr=false;
} node;
vector<node> seg;
vector<edge> e;
void update(int st,int et,int s,int e,int ind,int val){
    if(!seg[ind].qr) return;
    if(st==s && et==e){
        seg[ind].ed.pb(val);
        return;
    }
    int m=s+(e-s)/2;
    if(et<=m) update(st,et,s,m,2*ind+1,val);
    else if(st>m) update(st,et,m+1,e,2*ind+2,val);
    else{
        update(st,m,s,m,2*ind+1,val);
        update(m+1,et,m+1,e,2*ind+2,val);
    }
    return;
}
int total;
void run(int l,int r,int ind){
    if(!seg[ind].qr) return;
    int x=0;
    for(auto i: seg[ind].ed){
        if(connect(e[i].u,e[i].v)) total--,x++;
    }
    if(l==r){
        ans[l]=total;
    }
    if(l<r){
        int m=l+(r-l)/2;
        run(l,m,2*ind+1);
        run(m+1,r,2*ind+2);
    }
    while(x--) rollback(),total++;
    return;
}
int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);
    int n,m,i,j,k;
    cin>>n>>m;
    total=n;
    ans.assign(m+1,-1);
    seg.resize(4*m+4);
    init(n+1);
    for(i=0;i<m;i++){
        char c;
        int u=0,v=0;
        cin>>c;
        if(c!='?') cin>>u>>v;
        if(u>v) swap(u,v);
        if(c=='+') e.pb({u,v,i});
        else if(c=='-') e.pb({u,v,-i});
        else{
            int ind=0;
            int l=0,r=m-1;
            while(1){
                seg[ind].qr=true;
                if(l==r){
                    break;
                }
                int md=l+(r-l)/2;
                if(i<=md) r=md,ind=2*ind+1;
                else l=md+1,ind=2*ind+2;
            }
        }
    }
    sort(e.begin(),e.end());
    int sz=e.size();
    for(i=0;i<sz;i++){
        int s=e[i].t,r;
        if(i==sz-1 || e[i+1].u!=e[i].u || e[i+1].v!=e[i].v)
r=m-1;
        else r=-e[++i].t;
        update(s,r,0,m-1,0,i);
    }
    run(0,m-1,0);
    for(i=0;i<m;i++)                    if(ans[i]!=-1)
cout<<ans[i]<<endl;
}
```


Shortest Path:

Dijkstra:

```
/************ set *************/
const int inf = ;
void dijkstra(int s, vector<int> & d, vector<int> & p)
{
    int n = adj.size();
    d.assign(n, inf);
    p.assign(n, -1);

    d[s] = 0;
    set<pair<int, int>> q;
```

```
    q.insert({0, s});
    while (!q.empty()) {
        int v = q.begin().se;
        q.erase(q.begin());

        for (auto e : g[v]) {
            int to = e.fi;
            int len = e.se;

            if (d[v] + len < d[to]) {
                q.erase({d[to], to});
                d[to] = d[v] + len;
                p[to] = v;
                q.insert({d[to], to});
            }
        }
    }
}

/********** priority queue **********/
const int inf = ;
void dijkstra(int s, vector<int> & d, vector<int> & p)
{
    int n = adj.size();
    d.assign(n, inf);
    p.assign(n, -1);
    d[s] = 0;
    using pii = pair<int, int>;
    priority_queue<pii, vector<pii>, greater<pii>> q;
    q.push({0, s});
    while (!q.empty()) {
        int v = q.top().second;
        int d_v = q.top().first;
        q.pop();
        if (d_v != d[v])
            continue;
        for (auto e : g[v]) {
            int to = e.fi;
            int len = e.se;
            if (d[v] + len < d[to]) {
                d[to] = d[v] + len;
                p[to] = v;
                q.push({d[to], to});
            }
        }
    }
}
```

SPFA:

```
const int INF = 1000000000;

vector<pair<int,ll>> *g;
bool spfa(int s, vector<ll>& d) {
    int n = adj.size();
    d.assign(n, INF);
    vector<int> cnt(n, 0);
    vector<bool> inqueue(n, false);
    queue<int> q;

    d[s] = 0;
    q.push(s);
    inqueue[s] = true;
    while (!q.empty()) {
        int v = q.front();
        q.pop();
        inqueue[v] = false;

        for (auto i : g[v]) {
            int to = i.fi;
            ll len = i.se;

            if (d[v] + len < d[to]) {
                d[to] = d[v] + len;
                if (!inqueue[to]) {
                    q.push(to);
                    inqueue[to] = true;
                    cnt[to]++;
                    if (cnt[to] > n)
                        return false;  // negative cycle
                }
            }
        }
    }
    return true;
}
```

Floyd-Warshell:

```
for (int k = 0; k < n; ++k) {
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (d[i][k] < INF && d[k][j] < INF)
                d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
        }
    }
}
```

Topological Sort:

```
int n; // number of vertices

vector<vector<int>> adj; // adjacency list of graph
vector<bool> visited;
vector<int> ans;
void dfs(int v) {
    visited[v] = true;
    for (int u : adj[v]) {
        if (!visited[u])
            dfs(u);
    }
    ans.push_back(v);
}
void topological_sort() {
    visited.assign(n, false);
    ans.clear();
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i);
    }
    reverse(ans.begin(), ans.end());
}
```

MST:

Prim:

```
const int INF = 1000000000;
struct Edge {
    int w = INF, to = -1;
    bool operator<(Edge const& other) const {
        return make_pair(w, to) < make_pair(other.w, other.to);
    }
```

```
};
int n;
vector<vector<Edge>> adj;
void prim() {
    int total_weight = 0;
    vector<Edge> min_e(n);
    min_e[0].w = 0;
    set<Edge> q;
    q.insert({0, 0});
    vector<bool> selected(n, false);
    for (int i = 0; i < n; ++i) {
        if (q.empty()) {
            cout << "No MST!" << endl;
            exit(0);
        }

        int v = q.begin()->to;
        selected[v] = true;
        total_weight += q.begin()->w;
        q.erase(q.begin());

        if (min_e[v].to != -1)
            cout << v << " " << min_e[v].to << endl;

        for (Edge e : adj[v]) {
            if (!selected[e.to] && e.w < min_e[e.to].w) {
                q.erase({min_e[e.to].w, e.to});
                min_e[e.to] = {e.w, v};
                q.insert({e.w, e.to});
            }
        }
    }
    cout << total_weight << endl;
}
```

Kruskal:

```
vector<int> parent, rank;

void make_set(int v) {
    parent[v] = v;
    rank[v] = 0;
}

int find_set(int v) {
    if (v == parent[v])
        return v;
    return parent[v] = find_set(parent[v]);
}

void union_sets(int a, int b) {
    a = find_set(a);
    b = find_set(b);
    if (a != b) {
        if (rank[a] < rank[b])
            swap(a, b);
        parent[b] = a;
        if (rank[a] == rank[b])
            rank[a]++;
    }
}

struct Edge {
    int u, v, weight;
    bool operator<(Edge const& other) {
        return weight < other.weight;
    }
};

int n;
vector<Edge> edges;

int cost = 0;
vector<Edge> result;
parent.resize(n);
rank.resize(n);
for (int i = 0; i < n; i++)
    make_set(i);

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

for (Edge e : edges) {
    if (find_set(e.u) != find_set(e.v)) {
        cost += e.weight;
        result.push_back(e);
        union_sets(e.u, e.v);
    }
}
```

2 SAT:

```
//(a1 V a2)&(a3 V a4)....
/*
to establish the relations : the directed edges to be added
a | b: a' --> b &&&  b' --> a;
a & b: a' --> a &&&  b' --> b;
a ^ b: a' --> b &&&  a  --> b' &&& b' --> a &&& b --> a';
a x-nor b: a --> b &&& b-->a &&& a' --> b' &&& b' --> a';
Check for SCC: if comp[x]==comp[x'] 2-sat is unsatisfied
if comp[x]<comp[x'] x=false     else x=true;
ASSIGNMENT OF X DOESN'T DEPEND ON OTHER VARIABLE ASSIGNMENT!!!
*/
int n;
vector<int> *g, *gt;
vector<bool> used;
vector<int> order, comp;
vector<bool> assignment;

void dfs1(int v) {
    used[v]=true;
    for(auto i: g[v]) if(!used[i]) dfs1(i);
    order.push_back(v);
}
void dfs2(int v, int cl) {
    comp[v] = cl;
    for(int i: gt[v]) if(comp[i]==-1) dfs2(i);
}
bool solve_2SAT() {
    order.clear();
    used.assign(n+1, false);
    for (int i=1;i<=n;i++) if (!used[i]) dfs1(i);
    comp.assign(n+1, -1);
    for (int i = 0, j = 0; i < n; ++i) {
        int v = order[n - i - 1];
        if (comp[v] == -1) dfs2(v, j++);
    }
    assignment.assign(n / 2, false);
    for (int i = 0; i < n; i += 2) {
        if (comp[i] == comp[i + 1]) return false;
        assignment[i / 2] = comp[i] > comp[i + 1];
    }
    return true;
}
```

Maximum_Bipartite_Matching:

```
struct BipartiteMatcher {
 vector<vector<int>> G;
 vector<int> L, R, Viz;

 BipartiteMatcher(int n, int m) :
 G(n), L(n, -1), R(m, -1), Viz(n) {}
 void AddEdge(int a, int b) {
  G[a].push_back(b);
 }
 bool Match(int node) {
  if (Viz[node])
   return false;
  Viz[node] = true;
  for (auto vec : G[node]) {
   if (R[vec] == -1) {
    L[node] = vec;
    R[vec] = node;
    return true;
   }
  }
  for (auto vec : G[node]) {
   if (Match(R[vec])) {
    L[node] = vec;
    R[vec] = node;
    return true;
   }
  }
  return false;
 }
 int Solve() {
  bool ok = true;
  while (ok--) {
   fill(Viz.begin(), Viz.end(), 0);
   for (int i = 0; i < (int)L.size(); ++i)
    if (L[i] == -1)
     ok |= Match(i);
  }

  int ret = 0;
  for (int i = 0; i < L.size(); ++i)
   ret += (L[i] != -1);
  return ret;
 }
};
```

Custom Hash:

```
struct custom_hash {
   static uint64_t splitmix64(uint64_t x) {
      // http://xorshift.di.unimi.it/splitmix64.c
      x += 0x9e3779b97f4a7c15;
      x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
      x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
      return x ^ (x >> 31);
   }

   size_t operator()(uint64_t x) const {
      static const uint64_t
      FIXED_RANDOM =
chrono::steady_clock::now().time_since_epoch().cou
nt();
      return splitmix64(x + FIXED_RANDOM);
   }
};
//use unordered_map<int,int,custom_hash>
better_umap;
//use unordered_set<int,int,custom_hash> better_uset;
```

Permutation generator:

```
#include <bits/stdc++.h>
using namespace std;
void printArr(int a[], int n)
{
        for (int i = 0; i < n; i++)
                cout << a[i] << " ";
        printf("\n");
}

void heapPermutation(int a[], int size, int n)
{
        if (size == 1) {
                printArr(a, n);
                return;
        }
        for (int i = 0; i < size; i++) {
                heapPermutation(a, size - 1, n);
                if (size % 2 == 1)
                        swap(a[0], a[size - 1]);
                else
                        swap(a[i], a[size - 1]);
        }
}
int main()
{
        int a[] = { 1, 2, 3 ,4};
        int n = sizeof a / sizeof a[0];
        heapPermutation(a, n, n);
   cout<<co<<endl;
}
```


RANDOM SHUFFLE OF ARRAY:

```
void random_shuffle(vector<int> &rds){
   srand(time(NULL));
 for(int i=1;i<rds.size();i++)
swap(rds[i],rds[rand()%(i+1)]);
}
```

Matrix Exponentiation:

```
const int M = 1e9 + 7;
using ll = long long;
vector<vector<int>>matrixProduct
(vector<vector<int>> &a,vector<vector<int>> &b)
{
int c1 = a[0].size();
int r1 = a.size();
int r2 = b.size();
int c2 = b[0].size();
vector<vector<int>> mult(r1, vector<int>(c2,0));
int i,j,k;
for(i = 0; i < r1; ++i)
for(j = 0; j < c2; ++j)
for(k = 0; k < c1; ++k)
   mult[i][j] =
   ((ll)mult[i][j]+(ll)a[i][k]*(ll)b[k][j])%M;
return mult; }
vector<vector<int>> matrixPow(vvii &a, int n)
{
if (n == 1) return a;
auto res = matrixPow(a, n/2);
auto res2 = matrixProduct(res, res);
if (n & 1) return matrixProduct(a, res2);
return res2;
}
```

Mobius:

- $\mu(n) = 1$ if n is a square-free positive integer with an even number of prime factors.
- $\mu(n) = -1$ if n is a square-free positive integer with an odd number of prime factors.
- $\mu(n) = 0$ if n has a squared prime factor.
-

$$\sum_{d|n} \mu(d) = \begin{cases} 1 & n == 1 \\ 0 & n > 0 \end{cases}$$

Mobius Inversion:

If

$$g(n) = \sum_{d|n} f(d) \quad for\ n > 1$$

Then

$$f(n) = \sum_{d|n} \mu(d) g(\frac{n}{d})$$

Or

$$f(n) = \prod_{d|n} g\left(\frac{n}{d}\right)^{\mu(n)}$$

$$\gcd(a,b) = \sum_{d|\gcd(a,b)} \mu(d) = \sum_{d=1}^{n} [d|\gcd(a,b)]\mu(d)$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} [\gcd(i,j) = 1] = \sum_{d=1}^{n} \mu(d) \left\lfloor \frac{n}{d} \right\rfloor^2$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} \gcd(i,j) = \sum_{l=1}^{n} \varphi(l) \left\lfloor \frac{n}{l} \right\rfloor^2$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} lcm(i,j) = \sum_{l=1}^{n} \frac{\left\lfloor \frac{n}{l} \right\rfloor^2 \left(1 + \left\lfloor \frac{n}{l} \right\rfloor\right)^2}{4} \sum_{d|l} \mu(d) l d$$

Where $g(l) = \sum_{d|l} \mu(d) d\ ; in\ g(p^k) = 1 - p$

$$\sum_{a\in A}\sum_{b\in B} lcm(a,b) =$$

$$\sum_{t} \sum_{l|t} \frac{l}{t} \mu(l) \times \sum_{a\in A \cup t|a} a \times \sum_{b\in B \cup t|b} b$$

Finding Solution of 2 var equation:

```
int gcd(int a, int b, int& x, int& y) {
    if (b == 0) { x = 1, y = 0; return a; }
    int x1, y1;
    int d = gcd(b, a % b, x1, y1);
    x = y1; y = x1 - y1 * (a / b);
    return d;
}
bool find_any_solution(int a, int b, int c, int &x0,
int &y0, int &g) {
    g = gcd(abs(a), abs(b), x0, y0);
    if (c % g) {
        return false;
    }
    x0 *= c / g;
    y0 *= c / g;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}
```



```
vector<short int> mobius;
void generate_mobius(int N){
mobius.assign(N+1,-2);
mobius[1]=1;
for(int i=2;i<=N;i++) if(mobius[i]==-2){
for(int j=i;j<=N;j+=i)
mobius[j]=(mobius[j]==-2?-1:-mobius[j]);
for(int j=1;j*1ll*i*i<=N;j++)
mobius[j*i*i]=0;
}
}
```

Vector basis:

```
int basis[d];
int sz;
void insertVector(int mask) {
for (int i = 0; i < d; i++) {
if ((mask & 1<<i) == 0) continue;
if (!basis[i]){
basis[i] = mask;++sz;
return;
}
mask ^= basis[i];
}
}
```

Half-Plane Intersection:

```
const long double eps = 1e-9, inf = 1e9;
// Basic point/vector struct.
//include struct point with +,-,*,cross,dot
// Basic half-plane struct.
struct Halfplane {
Point p, pq; //line passes through p at pq direction
//the shade of the plane is at the left side of pq
long double angle;
Halfplane() {}
Halfplane(const Point& a,const Point& b): p(a),pq(b-a){
angle = atan2l(pq.y, pq.x);
}
bool out(const Point& r) {
return cross(pq, r - p) < -eps;
}
bool operator <(const Halfplane& e) const {
if (fabsl(angle - e.angle) < eps)
return cross(pq, e.p - p) < 0;
return angle < e.angle;
}
bool operator == (const Halfplane& e) const {
return fabsl(angle - e.angle) < eps;
}
friend Point inter(const Halfplane& s, const Halfplane& t) {
long double alpha=cross((t.p-s.p),t.pq)/cross(s.pq, t.pq);
return s.p + (s.pq * alpha);
}
};
// Actual algorithm returns common area as convex polygon
vector<Point>
hp_intersect(vector<Halfplane>& H) {
Point box[4] = {  // Bounding box in CCW order
Point(inf, inf),
Point(-inf, inf),
Point(-inf, -inf),
Point(inf, -inf)
};
for(int i = 0; i<4; i++) {
Halfplane aux(box[i], box[(i+1) % 4]);
H.push_back(aux);
}
// Sort and remove duplicates
sort(H.begin(), H.end());
H.erase(unique(H.begin(), H.end()), H.end());
deque<Halfplane> dq;
int len = 0;
for(int i = 0; i < int(H.size()); i++) {
while(len>1 && H[i].out(inter(dq[len-1],dq[len-2]))){
dq.pop_back(); --len;
}
while (len > 1 && H[i].out(inter(dq[0], dq[1]))) {
dq.pop_front(); --len;
}
dq.push_back(H[i]); ++len;
}
while(len>2 && dq[0].out(inter(dq[len-1], dq[len-2]))) {
dq.pop_back();--len;
}
while (len > 2 && dq[len-1].out(inter(dq[0], dq[1]))) {
dq.pop_front();
--len;
}// Report empty intersection if necessary
if (len < 3) return vector<Point>();
vector<Point> ret(len);
for(int i = 0; i+1 < len; i++) {
ret[i] = inter(dq[i], dq[i+1]);
}
ret.back() = inter(dq[len-1], dq[0]);
return ret;
}
```

ONLINE FFT:

```
//a[i] = Sum of a[j]*b[i-j-1] for j in range [0,i-1]
const ll MAX = 1e5 + 9;
ll a[2*MAX+10], b[2*MAX+10];
vector<ll> vec1,vec2,res;
int main(){
b[0]=b[1]=1;
for(int i=2;i< MAX;i++) b[i]=bigMod(i,i-1);
a[0]=1;
for(int i = 0; i < MAX; i++){
//+1 because of the format of the recurrence
a[i + 0 + 1] += (a[i] * b[0]) % mod;
a[i + 1 + 1] += (a[i] * b[1]) % mod;
if(a[i + 0 + 1] >= mod) a[i + 0 +1]-=mod;
if(a[i + 1 + 1] >= mod) a[i + 1 +1]-=mod;
ll cc = 1;
ll tmp = i;
while(tmp && (tmp & 1) == 0){
tmp = tmp / 2; cc = cc * 2;
vec1.clear(); vec2.clear();
for(int j=i-cc;j<i;j++) vec1.push_back(a[j]);
for(int j=cc;j<cc+cc;j++)
vec2.push_back(b[j]);
ntt.multiply(vec1, vec2, res);
ll Beg = i+1;
//+1 because of the format of the recurrence
for(ll j = 0; j < res.size(); j++){
a[Beg+j] += res[j];
if(a[Beg+j]>=mod) a[Beg+j]-=mod;
}
}
}
}
```

Wavelet tree:

```
const int N = 3e5, M = N;
const int MAX = 1e6;
vi g[N];
int a[N];
struct wavelet_tree{
#define vi vector<int>
#define pb push_back
int lo, hi;
wavelet_tree *l, *r;
vi b;
//nos are in range [x,y]
//array indices are [from, to)
wavelet_tree(int *from, int *to, int x, int y){
lo = x, hi = y;
if(lo == hi or from >= to) return;
int mid = (lo+hi)/2;
auto f = [mid](int x){
return x <= mid;
};
b.reserve(to-from+1);
b.pb(0);
for(auto it = from; it != to; it++)
b.pb(b.back() + f(*it));
//see how lambda function is used here
auto pivot = stable_partition(from, to, f);
l = new wavelet_tree(from, pivot, lo, mid);
r = new wavelet_tree(pivot, to, mid+1, hi);
}
//kth smallest element in [l, r]
int kth(int l, int r, int k){
if(l > r) return 0;
if(lo == hi) return lo;
int inLeft = b[r] - b[l-1];
int lb = b[l-1];
//amt of nos in first (l-1) nos that go in left
int rb = b[r];
//amt of nos in first (r) nos that go in left
if(k <= inLeft)
return this->l->kth(lb+1, rb , k);
return this->r->kth(l-lb, r-rb, k-inLeft);
}
//count of nos in [l, r] Less than or equal to k
int LTE(int l, int r, int k) {
if(l > r or k < lo) return 0;
if(hi <= k) return r - l + 1;
int lb = b[l-1], rb = b[r];
return this->l->LTE(lb+1, rb, k)
+ this->r->LTE(l-lb, r-rb, k);
}
//count of nos in [l, r] equal to k
int count(int l, int r, int k) {
if(l > r or k < lo or k > hi) return 0;
if(lo == hi) return r - l + 1;
int lb = b[l-1], rb = b[r], mid = (lo+hi)/2;
if(k <= mid)
return this->l->count(lb+1, rb, k);
return this->r->count(l-lb, r-rb, k);
}
~wavelet_tree(){
delete l;
delete r;
}
};
//wavelet_tree T(a, a+n, 1, MAX);
```

Implicit Treap:

```
#include<bits/stdc++.h>
using namespace std;
#define pb push_back
typedef long long int ll;
struct implicit_treap {
  mt19937 rnd;
  struct item{
    ll priority, value, cnt;
    bool rev;
    item *l = NULL, *r = NULL;
    item(ll v, ll p) {
      value = v;
      cnt = 1;
      rev = false;
      priority = p;
    }
};
int cnt (item *it) {
  return it ? it->cnt : 0;
}
void upd_cnt (item *it) {
  if (it)
    it->cnt = cnt(it->l) + cnt(it->r) + 1;
}
void push (item *it) {
  if (it && it->rev) {
    //cout << it->value << endl;
    it->rev = false;
    swap (it->l, it->r);
    if (it->l)  it->l->rev ^= true;
    if (it->r)  it->r->rev ^= true;
  }
}
item* merge (item *l, item *r) {
  push (l);
  push (r);
  item* t = NULL;
  if (!l|| !r)
    t = (l) ? l : r;
  else if (l->priority > r->priority)
  {
    t = l;
    t->r = merge(l->r, r);
  }
  else{
    t = r;
    t->l = merge(l, r->l);
  }
  upd_cnt (t);
  return t;
}
vector<item*> split (item *t, int key) {
  if (!t)
    return {NULL, NULL};
  push (t);
```

```
    int cur_key = cnt(t->l);
    if (key <= cur_key) {
        auto x = split(t->l, key);
        t->l = NULL;
        x[1] = merge(x[1], t);
        upd_cnt (x[0]); upd_cnt(x[1]);
        return x;
    }
    else {
        auto x = split(t->r, key - cnt(t->l) - 1);
        t->r = NULL;
        x[0] = merge(t, x[0]);
        upd_cnt (x[0]); upd_cnt(x[1]);
        return x;
    }
}
void reverse (item *t, int l, int r) {
    auto fs = split(t, l);
    auto ss = split(fs[1], r-l+1);
    ss[0]->rev ^= true;
    t = merge(fs[0], ss[0]);
    t = merge(t, ss[1]);
}
void output (item *t) {
    if (!t)  return;
    push (t);
    output (t->l);
    printf ("%lld ", t->value);
    output (t->r);
}
item *root;
implicit_treap(vector<ll> arr) {
    rnd = mt19937(time(0));
    root = nullptr;
    for (ll i : arr) {
        insert(i);
    }
}
implicit_treap(item *r) {
    root = r;
    }

    void insert(ll val) {
        ll p = rnd();
        item *newNode = new item(val, p);
        root = merge(root, newNode);
    }

    void print() {
        output(root);
    }
};
int main() {
    //IO;
    int n; cin >> n;
    vector<ll> arr(n);
    for (int i=1; i<=n; i++) {
        arr[i-1] = i;
    }
    implicit_treap it(arr);
    int sz = n;
    while(n--) {
        int a, b; cin >> a >> b;
        a--; b--;
        if (b <= a) continue;
        int len = min(b - a, sz - b);
        auto x = it.split(it.root, b);
        auto y = it.split(x[0], a);
        auto z = it.split(x[1], len);
        auto w = it.split(y[1], len);
        auto t = it.merge(y[0], z[0]);
        auto t2 = it.merge(w[1], w[0]);
        t2 = it.merge(t2, z[1]);
        t = it.merge(t, t2);
        it = implicit_treap(t);
        //it.print();
    }
    it.print();
}
```

Matrix Exponentiation:

```
const int M = 1e9 + 7;
using ll = long long;
vector<vector<int>>matrixProduct
(vector<vector<int>>
&a,vector<vector<int>> &b)
{
int c1 = a[0].size();
int r1 = a.size();
int r2 = b.size();
int c2 = b[0].size();
vector<vector<int>>                    mult(r1,
vector<int>(c2,0));
int i,j,k;
for(i = 0; i < r1; ++i)
for(j = 0; j < c2; ++j)
for(k = 0; k < c1; ++k)
   mult[i][j] =
   ((ll)mult[i][j]+(ll)a[i][k]*(ll)b[k][j])%M;
return mult;
}
vector<vector<int>> matrixPow(vvii &a, int
n)
{
if (n == 1) return a;
auto res = matrixPow(a, n/2);
auto res2 = matrixProduct(res, res);
if (n & 1) return matrixProduct(a, res2);
return res2;
}
```