Sayeef

Articulation Points & Bridges:

```
vector<int>g[100005], tym(100005), articulationpoints;
vector<pair<int,int>>bridges;
int findarticulationpoint(int node, int par, int lev){//node 0 based, lev 1
based
    if(node!=0 && g[node].size()==1) return lev;
    if(node==0 && g[node].size()>=2) articulationpoints.pb(node);
    if(tym[node]) return tym[node];
    tym[node]=lev;
    int res=lev;
    for(int a : g[node]){
        if(a==par) continue;
        res=min(res,findarticulationpoint(a,node,lev+1));}
    if(node==0) return 0;
    if(res>=lev) articulationpoints.pb(node);
    return tym[node]=res;}
int findbridges(int node, int par, int lev){//lev 1 based
    if(tym[node]) return tym[node];
    tym[node]=lev;
    int res, val=lev;
    for(int a : g[node]){
        if(a==par) continue;
        res=findbridges(a,node,lev+1);
        val=min(val,res);
        if(res>lev) bridges.pb({min(a,node),max(a,node)});}
    return tym[node]=val;}
```

Bellman Ford:

```
dist[1]=0;
for(int i=2; i<=n; i++) dist[i]=30000;
for(int i=0; i<n-1; i++){
    int f=0;
    for(int node=1; node<=n; node++){
        if(dist[node]==30000) continue;
        for(pair<int,int> a : g[node]){
          if(dist[a.ff]>dist[node]+a.ss){
               dist[a.ff]=dist[node]+a.ss, f=1;}}}
        if(!f) break;}
```

Centroid Decomp(dis without binary lifting):

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
const int RANDOM =
chrono::high_resolution_clock::now().time_since_epoch().count();
struct chash {
    int operator()(int x) const { return x ^ RANDOM; }};
#define mxn 500005
vector<int>g[mxn];
int n, m, sub[mxn]={}, iscentroid[mxn]={};
int parcentroid[mxn]={}, color[mxn];
vector<int>distoparent[mxn];
gp_hash_table<int, int, chash> d[mxn];
int dfs(int node, int par, int dep, int start){
    sub[node]=1;
    for(int a : g[node]){
        if(a!=par && !iscentroid[a])
sub[node]+=dfs(a,node,dep+1,start);}
    return sub[node];}
int getCentroid(int node, int par, int n){
    for(int a : g[node]){
        if(a!=par && !iscentroid[a] && sub[a]>n/2) return getCentroid(a,
node, n);}
    return node;}
void dfs2(int node, int par, int dis){
    distoparent[node].pb(dis);
    for(int a : g[node]){
        if(a!=par && !iscentroid[a]){
            dfs2(a,node,dis+1);}}}
void decompose(int node, int par, int parentcentroid){
    int n=dfs(node,par,0,node), centroid=getCentroid(node, par, n);
    dfs2(centroid,-1,0);
    iscentroid[centroid]=1;
    if(parentcentroid!=-1){
        parcentroid[centroid]=parentcentroid;}
    for(int a : g[centroid]){
        if(a!=par && !iscentroid[a]) decompose(a,centroid,centroid);}}
void mark(int initial, int node, int height){
    int dist=distoparent[initial][height];
    if(d[node][color[initial]]==0) d[node][color[initial]]=dist+1;
    else d[node][color[initial]]=min(d[node][color[initial]],dist+1);
    if(parcentroid[node]) mark(initial,parcentroid[node],height+1);}
ll query(int initial, int node, ll colnow, int height){
    ll dnow=d[node][colnow];
    if(dnow==0) dnow=inff;
    ll distance=distoparent[initial][height];
    ll res=distance+dnow;
    if(parcentroid[node])
res=min(res,query(initial,parcentroid[node],colnow,height+1));
    return res;}
int main(){
    decompose(1,-1,-1);
    fro(n+1) reverse(all(distoparent[i]));}
```

CHT:

```
/*1. mi > mi+1, minimum. 2. mi > mi+1, maximum.
3. mi < mi+1, minimum. 4. mi < mi+1, maximum.*/
struct line{
    ll m, c;
    line(ll a, ll b){
        m=a, c=b;}};
struct CHT{
    vector<line>v;
    int t, ptr;
    void init(int tp){
        t=tp, ptr=0, v.clear();}
    CHT(int tp){
        t=tp, ptr=0;}
    bool bad(line l1, line l2, line l3){
        __int128 a=(__int128)(l3.c-l1.c)*(l1.m-l2.m);
        __int128 b=(__int128)(l2.c-l1.c)*(l1.m-l3.m);
        if(t==1 orr t==4) return a<=b;
        return a>=b;}
    void add(line a){
        v.pb(a);
        int sz=v.size();
        while(sz>=3 && bad(v[sz-3],v[sz-2],v[sz-1])){
            v.erase(v.end()-2), sz--;}}
    inline ll val(int ind, ll x){
        return v[ind].m*x+v[ind].c;}
    ll query1(ll x){//ternary search
        int l=0, r=v.size()-1;
        ll ans=0;
        while(l<=r){
            int mid1=l+(r-l)/3, mid2=r-(r-l)/3;
            if(t&1){
                if(val(mid1,x)<=val(mid2,x)) r=mid2-1, ans=val(mid1,x);
                else l=mid1+1, ans=val(mid2,x);}
            else{
                if(val(mid1,x)>=val(mid2,x)) r=mid2-1, ans=val(mid1,x);
                else l=mid1+1, ans=val(mid2,x);}}
        return ans;}
    ll query2(ll x){//1,4 if xi<=xi+1; 2,3 if xi>=xi+1
        if(v.empty()) return 0;
        if(ptr>=v.size()) ptr=v.size()-1;
        while(ptr<v.size()-1){
            if(t&1){
                if(val(ptr,x)>val(ptr+1,x)) ptr++;
                else break;}
            else{
                if(val(ptr,x)<val(ptr+1,x)) ptr++;
                else break;}}
        return val(ptr,x);}};
```

Closest pair of points using divide & conquer:

```
#define eps 1e-7
struct point{
    double x, y;}arr[10005];
double get_dis(int i, int j){
   double a=abs(arr[i].x-arr[j].x);
   double b=abs(arr[i].y-arr[j].y);
   return sqrt(a*a+b*b);}
double finddis(int l, int r){
   if(r-l<=3){
      double res=1e20;
      for(int i=l; i<=r; i++){
         for(int j=i+1; j<=r; j++){
            res=min(res,get_dis(i,j));}}
      return res;}
   int mid=(l+r)>>1;
   double left=finddis(l,mid);
   double right=finddis(mid+1,r);
   double k=min(left,right);
   vector<int>v;
   for(int i=l; i<=r; i++){
      if(abs(arr[i].x-arr[mid].x)-k<=eps){
         v.pb(i);}}
   sort(all(v),[](int a, int b){
      if(arr[a].y!=arr[b].y) return arr[a].y<arr[b].y;
      return arr[a].x<arr[b].x;});
   int sz=v.size();
   for(int i=0; i<sz; i++){
      for(int j=i+1; j<min(sz,i+8); j++){
         k=min(k,get_dis(v[i],v[j]));}}
   return k;}
int main(){
   fastio;
   int n;
   while(cin>>n){
      if(!n) break;
      fr(n){
         cin>>arr[i].x>>arr[i].y;}
      sort(arr,arr+n,[](point a, point b){
         if(a.x!=b.x) return a.x<b.x;
         return a.y<b.y;});
      double ans=finddis(0,n-1);
      if(10000-ans<=eps) cout<<"INFINITY"<<nl;
      else cout<<setprecision(4)<<fixed<<ans<<nl;}}
```

Custom Set Comparator:

```
struct cmp {
   bool operator() (pair<int,pair<int,int> > a, pair<int,pair<int,int> > b)
const {
      if(a.ff!=b.ff) return a.ff<b.ff;
      return a.ss.ff>b.ss.ff;}};
```

DAG Connected?:

```
ll dfs4(ll node){
   if(vis[node]) return 0;
   vis[node]=1;
   ll res=1;
   for(ll a : g[node]) res+=dfs4(a);
   for(ll a : revg[node]) res+=dfs4(a);
   return res;}
```

Digit DP:

```
ll solve(ll in, ll n, ll sum, ll flag){
         if(in==n) return sum;
         if(dp[in][sum][flag]!=-1) return dp[in][sum][flag];
         ll limit=9, res=0;
         if(!flag) limit=use[in]-'0';
         for(int i=0; i<=limit; i++){
                  if(flag orr i<limit) res+=solve(in+1, n , sum+i, 1LL);
                  else res+=solve(in+1, n, sum+i, 0LL);}
         return dp[in][sum][flag]=res;}
```

Digit DP one time memset:

```
string s;
int n;
ll dp[22][2][10][1030][2];
ll solve(int in, int flag, int colornow, int mask, int done){
         if(in==0){
                  if(__builtin_popcount(mask)==colornow && done)
return 1;
         return 0;}
         if(dp[in][flag][colornow][mask][done]!=-1 && flag) return
dp[in][flag][colornow][mask][done];
         ll in2=n-in, res=0, upto=(flag!=0)?9:s[in2]-'0', high=upto;
         upto=min(upto,(ll)colornow);
         for(int i=0; i<=upto; i++){
                  int nextmask=(mask==1)?0:mask;
                  nextmask|=(1<<i);
if(__builtin_popcount(nextmask)<=colornow){
                  if(!flag && i==high)

res+=solve(in-1,0,colornow,nextmask,done|(i==colornow));
                  else
res+=solve(in-1,1,colornow,nextmask,done|(i==colornow));}}
         return dp[in][flag][colornow][mask][done]=res;}
```

Dijkstra:

```
ll n, m;
cin>>n>>m;
vector<pair<ll,ll> >g[n+1];
fr(m){
   ll a, b, c;
   cin>>a>>b>>c;
   g[a].pb({b,c});
   g[b].pb({a,c});}
priority_queue<pair<ll,ll> >q;
q.push({-0,-1});
ll dis[n+1], path[n+1];
fr(n+1) dis[i]=LLONG_MAX;
dis[1]=0;
while(!q.empty()){
   ll cost=-q.top().ff, node=-q.top().ss;
   q.pop();
   if(dis[node]!=cost) continue;
   for(pair<ll,ll> a : g[node]){
      if(dis[node]+a.second<dis[a.first]){
         path[a.first]=node;
         dis[a.first]=dis[node]+a.second;
         q.push({-dis[a.first],-a.first});}}}
```

Euler Tour for sum on a path using segtree:

```
void eulertour(int node, int par, ll cost, int dep){//for sum on a
subtree, don't increase the counter while exiting
   depth[node]=dep, ancestor[node][0]=par, stcnt[node]=++c;
   update(1,0,200002,c,cost);
   for(pair<int,ll> a : g[node]){
      if(a.ff!=par){
         eulertour(a.ff,node,a.ss,dep+1);}}
   encnt[node]=++c;
   update(1,0,200002,c,-cost);}
```

Expected Value:

```
double solve(ll ones, ll twos, ll threes){
         if(ones==0 && twos==0 && threes==0) return 0;
         if(dp[ones][twos][threes]!=-1) return dp[ones][twos][threes];
         double res=0;
         if(ones>0){

res+=((ones*1.0)/n)*(1+solve(ones-1,twos,threes));}
         if(twos>0){
         res+=((twos*1.0)/n)*(1+solve(ones+1,twos-1,threes));}
         if(threes>0){
         res+=((threes*1.0)/n)*(1+solve(ones,twos+1,threes-1));}
         res+=((n-(ones+twos+threes))*1.0)/n;
         res/=1.0-((n-(ones+twos+threes))*1.0)/n;
         return dp[ones][twos][threes]=res;}
```

Fastio:

```
#define fastio ios_base::sync_with_stdio(false); cin.tie(NULL)
```

FileIO:

```
ifstream cin; ofstream cout;
cin.open("1_input.txt", ios::in);
cout.open("1_output.txt", ios::out);
```

Floyd Warshal:

```
   for(int l=1; l<=n; l++){
      for(int i=1; i<=n; i++){
        for(int j=1; j<=n; j++){
          dist[i][j]=min(dist[i][j],dist[i][l]+dist[l][j]);}}}
```

Hashing:

```
string s;
ll n, m1=1e9+7, m2=1e9+9, p1=29, p2=31;
ll m1inv=758620695, m2inv=838709685, prime1[1000005],
prime2[1000005];
pair<ll,ll>prefhash[1000005], suffhash[1000005];
void getprimes(){
        prime1[0]=1, prime2[0]=1;
        for(int i=1; i<n+3; i++){
                prime1[i]=(prime1[i-1]*p1)%m1;
        prime2[i]=(prime2[i-1]*p2)%m2;}}
void forhash(){
        ll hash1=0, hash2=0;
        for(int i=0; i<n; i++){
          hash1=(hash1+((s[i]-'a'+1)*prime1[i])%m1)%m1;
          hash2=(hash2+((s[i]-'a'+1)*prime2[i])%m2)%m2;
          prefhash[i]={hash1,hash2};}}
```

Hash Unordered Map:

```
struct custom_hash {
   static uint64_t splitmix64(uint64_t x) {
      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().count();
      return splitmix64(x + FIXED_RANDOM);}};
```

Inclusion-Exclusion:

```
   ll l, r;
   cin>>l>>r;
   ll ans=0, vis[1000006]={};
   fr(1000002) use[i]=1;
   for(int i=2; i<=1000000; i++){
      if(vis[i]) continue;
      for(int j=i; j<=1000000; j+=i){
         if(!use[j]) continue;
         vis[j]=1;
         facts[j]++;
         ll tmp=j, c=0;
         while(tmp%i==0){
            if(c>1) break;
            c++;
            tmp/=i;}
         if(c>1) use[j]=0;//checking if j has more than 1 of the same
factor, because we need 1 occurance of each prime}}
   for(ll i=2; i<=r; i++){
      if(!use[i]) continue;
      ll cnt=0;
      ll tmp1=r/i, tmp2=(l-1)/i;
      cnt=tmp1-tmp2;
      cnt=max((cnt*(cnt-1))/2,0ll);
      if(facts[i]&1){
         ans+=cnt;}
  else{
         ans-=cnt;}}
   if(l==1) l++;
   for(int i=l; i<=r; i++){
      ans-=(r/i - 1);}
   cout<<2*ans<<nl;
```

Indx of 1st element with value less than x in seg tree:

```
int query(int node, int st, int en){
   if(st==en){
      if(seg[node]>=valnow) return st;
      if(st!=l) return st-1;
      return -1;}
   int mid=(st+en)>>1, ans;
   if(seg[2*node]>=valnow) return query(2*node+1,mid+1,en);
   return query(2*node,st,mid);}
```

Intersection of two segments:

```
int z = min(x2, y2) - max(x1, y1);//segment x1->x2 y1->y2
if(z<0) // no intersection
else // intersected by z
```

Kedane's alg for maximum subarray sum:

```
   ll maximum=0, current=0;
   for(int i=0; i<input; i++){
      current+=arr[i];
      if(current>maximum) maximum=current;
      if(current<0) current=0;}
```

Kosaraju's alg for strongly connected comps:

```
stack<int>s;
int vis[100005]={}, cmp[100005]={}, cnt=-1;
vector<int>g[100005],grev[100005];
void dfs(int node){
   if(vis[node]) return;
   vis[node]=1;
   for(int next : g[node]) dfs(next);
   s.push(node);}
void ndfs(int node){
   if(vis[node]) return;
   cmp[node]=cnt;
   vis[node]=1;
   for(int next : grev[node]) ndfs(next);}
int main(){
   fastio;
   ll n, m;
   cin>>n>>m;
   fr(m){
      int a, b;
      cin>>a>>b;
      g[a].pb(b);
      grev[b].pb(a);}
   fr(n) dfs(i);
   memset(vis,0,sizeof(vis));
   while(!s.empty()){
      int top=s.top();
      s.pop();
      if(!vis[top]){
         cnt++;
         ndfs(top);}}}
```

Kth ancestor using binary lifting:

```
class TreeAncestor {
   vector<ll>g[50005];
   ll ancestors[50005][20]={}, dep[50005];
public:
   void dfs(ll node, ll par){
      for(ll a : g[node]){
         if(a!=par){
            dep[a]=dep[node]+1;
            dfs(a,node);}}}
   TreeAncestor(int n, vector<int>& parent) {
      dep[0]=0;
      for (int i=1;i<n; i++){
         g[i].push_back(parent[i]);
         g[parent[i]].push_back(i);}
      dfs(0ll,-1ll);
      for(int i=0; i<n; i++) ancestors[i][0]=parent[i];
      for(int i=1; i<=17; i++){
         for(int j=1; j<n; j++){
            if(dep[j]<(1<<i)) continue;
            ancestors[j][i]=ancestors[ancestors[j][i-1]][i-1];}}}
   int getKthAncestor(int node, int k) {
      if(dep[node]<k) return -1;
      if(!k) return node;
      for(int i=0; i<20; i++){
         if(k&(1<<i)){
            node=ancestors[node][i];}}
      return node;}};
```

Longest Palindromic Substring:

```
   for(int mid=0; mid<n; mid++){
      for(int x=0; mid+x<n && mid-x>=0; x++){
         if(a[mid+x]!=a[mid-x]){
            break;}
        ll len=2*x+1;
        if(len>bestlen){
            bestlen=len;
            outstring=a.substr(mid-x,2*x+1);}}}
    for(int mid=0; mid<n; mid++){
        for(int x=1; mid-x+1>=0 && mid+x<n; x++){
            if(a[mid-x+1]!=a[mid+x]){
                break;}
            ll len=2*x;
            if(len>bestlen){
                bestlen=len;
                outstring=a.substr(mid-x+1,2*x);}}}
```

NCR-NPR in range:

```
//npr = n-1 P r + (n-1 P r-1) * r
ncr[0][0]=ncr[1][0]=ncr[1][1]=1;
    for(int i=2; i<5002; i++){
        ncr[i][0]=ncr[i][i]=1;
        for(int j=1; j<i; j++){
            ncr[i][j]=(ncr[i-1][j]+ncr[i-1][j-1])%mod;}}
```

NOD Sieve:

```
for(int i=1; i<1000001; i++){
        for(int j=i; j<1000001; j+=i){
                divisors[j]++;
        }
}
```

Number of distinct elements in a subarray using persistent seg tree O(logn):

```
#define mxn 1000005
#define mxt 24000005
int arr[30005]={}, arr2[30005], version[200005], n, nextnode=1, v=0;
int seg[mxt], lchild[mxt], rchild[mxt];
int prv[1000005]={};
void build(int node, int st, int en){
        if(st==en){
                seg[node]=arr[st];
                return;}
        lchild[node]=++nextnode;
        rchild[node]=++nextnode;
        ll mid=(st+en)/2;
        build(lchild[node],st,mid);
        build(rchild[node],mid+1,en);
        seg[node]=seg[lchild[node]]+seg[rchild[node]];}
ll update(int node, int st, int en, int ind, int val){
        if(ind>en orr ind<st) return node;
        int newnode=++nextnode;
        if(st==en){
                seg[newnode]=seg[node]+val;
                return newnode;}
ll mid=(st+en)/2;
lchild[newnode]=update(lchild[node],st,mid,ind,val);
rchild[newnode]=update(rchild[node],mid+1,en,ind,val);
seg[newnode]=seg[lchild[newnode]]+seg[rchild[newnode]];
        return newnode;}
ll query(int node, int st, int en, int l, int r){
        if(st>=l && en<=r) return seg[node];
        if(st>r orr en<l) return 0;
        ll mid=(st+en)/2;
        return
query(lchild[node],st,mid,l,r)+query(rchild[node],mid+1,en,l,r);}
int main(){
        fastio;
        cin>>n;
        build(1,0,30000);
        version[0]=1;
        fro(n+1){
                cin>>arr2[i];
                ll val=prv[arr2[i]];
                prv[arr2[i]]=i;
        version[v+1]=update(version[v],0,30000,val,1);
                v++;}
        ll q;
        cin>>q;
        while(q--){
                int l, r;
                cin>>l>>r;
                ll
ans=query(version[r],0,30000,0,l-1)-query(version[l-1],0,30000,0,l-1);
                cout<<ans<<nl;}}
```

Number of inversions using merge sort:

```
ll merge(int l, int mid, int r){
    int i=l, j=mid+1, k=0;
    ll inv=0;
    while(i<=mid && j<=r){
        if(arr[i]<=arr[j]) temp[k++]=arr[i++];
        else temp[k++]=arr[j++], inv+=mid-i+1;}
    while(i<=mid) temp[k++]=arr[i++];
    while(j<=r) temp[k++]=arr[j++];
    k=0;
    for(i=l; i<=r; i++) arr[i]=temp[k++];
    return inv;}
ll msort(int l, int r){
    if(l==r){
        return 0;}
    int mid=(l+r)/2;
    ll a=msort(l,mid), b=msort(mid+1,r), c=merge(l,mid,r);
    return a+b+c;}
```

Number of K length paths in a directed graph matexpo:

```
ll n, m, mat[205][205];
vector<vector<ll> >id(205,vector<ll>(205,0));
vector<int>g[105];
void mul(vector<vector<ll> >&res, vector<vector<ll> >mat1,
vector<vector<ll> >mat2){
    for(int i=0; i<m; i++){
        for(int j=0; j<m; j++){
            for(int k=0; k<m; k++){
  res[i][j]=(res[i][j]+(mat1[i][k]*mat2[k][j])%mod)%mod;}}}}
vector<vector<ll> > matexpo(vector<vector<ll> >mat, ll p){
    if(!p) return id;
    if(p&1){
        vector<vector<ll> >res(m,vector<ll>(m));
        mul(res,mat,matexpo(mat,p-1));
        return res;}
    else{
        vector<vector<ll> >res(m,vector<ll>(m));
        vector<vector<ll> >mat2=matexpo(mat,p/2);
        mul(res,mat2,mat2);
        return res;}}
int main(){
    fastio;
    int e, path;
    cin>>n>>e>>path;
    fr(e){
        int a, b;
        cin>>a>>b;
        a--, b--;
        g[b].pb(a);}
    m=n;
    fr(m){
        frj(m){
            id[i][j]=(i==j);}}
    vector<vector<ll>>matrix(m,vector<ll>(m,0));
    fr(n){
        for(int a : g[i]){
            matrix[i][a]=1;}}
    vector<vector<ll>>ans=matexpo(matrix,path);
    ll sum=0;
    fr(n){
        frj(n){
            sum=(sum+ans[i][j]%mod)%mod;}}}
```

Number of subarrays with xor >= to K(Trie):

```
int trie[31000005][2], cnt[31000005], num;
ll ans;
void init(){
    memset(trie,0,sizeof(trie));
    memset(cnt,0,sizeof(cnt));
    num=1;}
void addElement(int a){
    int node=1;
    for(int i=30; i>=0; i--){
        int ch=(a&(1<<i))!=0?1:0;
        if(!trie[node][ch]) trie[node][ch]=++num;
        node=trie[node][ch], cnt[node]++;}}
void query(int a, int k){
    int node=1, x=a^k, f=1;
    for(int i=30; i>=0; i--){
        int ch=(x&(1<<i))!=0?1:0;
        if((k&(1<<i))){
            if(trie[node][ch]) node=trie[node][ch];
            else break;}
        else{
            if(trie[node][ch^1]) ans+=cnt[trie[node][ch^1]];
            if(trie[node][ch]) node=trie[node][ch];
            else break;}
        if(!i) ans+=cnt[node];}}
int main()
{
    fastio;
    ll n, k, x=0;
    cin>>n>>k;
    init();
    ans=0;
    addElement(x);
    vector<int>v;
    fr(n){
        int a;
        cin>>a;
        x^=a;
        query(x,k);
        addElement(x);}
    cout<<ans<<nl;}
```

PBDS:

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <typename T> using ordered_set = tree<T, null_type, less<T>, rb_tree_tag, tree_order_statistics_node_update>;
//*s.find_by_order(2) -> element at index 2
//s.order_of_key(3) -> lower bound of 3
```

Prime factor in a range sieve:

```
            for(int i=2; i<=200000; i++){
                        if(!nprime[i]){
                                    pf[i].pb(i);
                                    for(int j=i+i; j<=200000; j+=i){
                                                nprime[j]=1;
                                                int jc=j;
                                                while(jc%i==0){
                                                            pf[j].pb(i);
                                                            jc/=i;}}}}
```

Prims MST:

```
#define pii pair<ll,pair<ll,ll> >
vector<pair<ll,ll> >g[100005], g2[100005];
vector<ll>par(100005);
bool vis[100005]={};
ll prim(ll x){
    priority_queue<pii, vector<pii>, greater<pii> >q;
    q.push({0,{x,-1}});
    ll mincost=0;
    while(!q.empty()){
        pair<ll,pair<ll,ll> >element=q.top();
        q.pop();
        if(vis[element.ss.ff]) continue;
        vis[element.ss.ff]=1;
        if(element.ss.ss!=-1){
            g2[element.ss.ff].pb({element.ff,element.ss.ss});
            g2[element.ss.ss].pb({element.ff,element.ss.ff});
            par[element.ss.ff]=element.ss.ss;}
        mincost+=element.ff;
        for(pair<ll,ll> a : g[element.ss.ff]){
            if(!vis[a.ss]){
                q.push({a.ff,{a.ss,element.ss.ff}});}}}
    return mincost;}
```

Rerooting dp:

```
vector<ll>g[200005];
ll n, sz[200005]={}, dp[200005]={}, ans=0;
int dfs1(int node, int par){
    sz[node]=1;
    for(int a : g[node]){
        if(a!=par){
            sz[node]+=dfs1(a,node);}}
    dp[node]=sz[node];
    for(int a : g[node]){
        if(a!=par){
            dp[node]+=dp[a];}}
    return sz[node];}
void reroot(int node, int par){
    ans=max(ans,dp[node]);
    for(int a : g[node]){
        if(a!=par){
            dp[node]-=sz[a]+dp[a];
            sz[node]-=sz[a];
            sz[a]+=sz[node];
            dp[a]+=sz[node]+dp[node];
            reroot(a,node);
            dp[a]-=sz[node]+dp[node];
            sz[a]-=sz[node];
            sz[node]+=sz[a];
            dp[node]+=sz[a]+dp[a];}}}
```

Sack:

```
int cnt[maxn];
bool big[maxn];
void add(int v, int p, int x){
    cnt[ col[v] ] += x;
    for(auto u: g[v])
        if(u != p && !big[u])
            add(u, v, x)}
void dfs(int v, int p, bool keep){
    int mx = -1, bigChild = -1;
    for(auto u : g[v])
       if(u != p && sz[u] > mx)
          mx = sz[u], bigChild = u;
    for(auto u : g[v])
        if(u != p && u != bigChild)
            dfs(u, v, 0);  // run a dfs on small childs and clear them from cnt
    if(bigChild != -1)
        dfs(bigChild, v, 1), big[bigChild] = 1;  // bigChild marked as big and not cleared from cnt
    add(v, p, 1);
    //now cnt[c] is the number of vertices in subtree of vertex v that has color c. You can answer the queries easily.
    if(bigChild != -1)
        big[bigChild] = 0;
    if(keep == 0)
        add(v, p, -1);}
```

Sparse Table:

```
ll s[100005][20];
int main(){
            fastio;
            ll n, q;
            cin>>n;
            ll arr[n];
            fr(n){
                        cin>>arr[i];
                        s[i][0]=arr[i];}
            cin>>q;
            ll upto=31-__builtin_clz(n);
            for(int i=1; i<=upto; i++){
                    for(int j=0; j+(1<<i)-1<n; j++){
                        s[j][i]=min(s[j][i-1],s[j+(1<<(i-1))][i-1]);}}
            while(q--){
                    ll l, r;
                    cin>>l>>r;
                    ll len=r-l+1;
                    ll downto=31-__builtin_clz(len);
    ll ans=min(s[l][downto],s[r-(1<<downto)+1][downto]);
                    cout<<ans<<nl;}}
```

Seg tree lazy:

```
void update(ll node, ll st, ll en, ll l, ll r, ll val){
    if(st>r orr en<l) return;
    if(st>=l && en<=r){
        seg[node]+=(en-st+1)*val;
        lazy[node]+=val;
        return;}
    ll mid=(st+en)/2;
    update(2*node,st,mid,l,r,val);
    update(2*node+1,mid+1,en,l,r,val);
    seg[node]=seg[2*node]+seg[2*node+1]+(en-st+1)*lazy[node];}
ll query(ll node, ll st, ll en, ll l, ll r, ll carry){
    if(st>r orr en<l) return 0;
    if(st>=l && en<=r){
        return (en-st+1)*carry+seg[node];}
    ll mid=(st+en)/2;
    ll q1=query(2*node,st,mid,l,r,carry+lazy[node]);
    ll q2=query(2*node+1,mid+1,en,l,r,carry+lazy[node]);
    return q1+q2;}
```

Sieve:

```
ll nop=10000000;
bitset<10000005>keep;
vector<ll>prime;
void sieve(){
    for(ll i=3; i*i<=nop; i+=2){
        if(!keep[i]){
            for(ll j=i*i; j<=nop; j+=2*i){
                keep[j]=1;}}}
    prime.pb(2);
    for(ll i=3; i<=nop; i+=2){
        if(!keep[i]) prime.pb(i);}}
```

TRIE:

```
struct TRIE{
    vector<vector<int>>trie;
    vector<int>endshere;
    int num;
    void init(){
        trie.clear(), endshere.clear(), num=0;
        trie.pb(vector<int>(2)), endshere.pb(0);}
    void addElement(string s){
        int node=0;
        for(char c : s){
            int ch=c-45;
            if(!trie[node][ch]){
                trie.pb(vector<int>(2)), endshere.pb(0);
                trie[node][ch]=++num;}
            node=trie[node][ch];}
        endshere[node]++;}
    void traverse(int node, string pref){
        if(endshere[node]) cout<<"pref "<<pref<<nl;
        fr(2){
            if(trie[node][i]){
                traverse(trie[node][i],pref+char(45+i));}}}};
```

Unordered Hashing: val1=(val1+(bigmod(base1, a[i],mod1)* (a[i]))%mod1)%mod1;

XOR-sum of all subsequences:

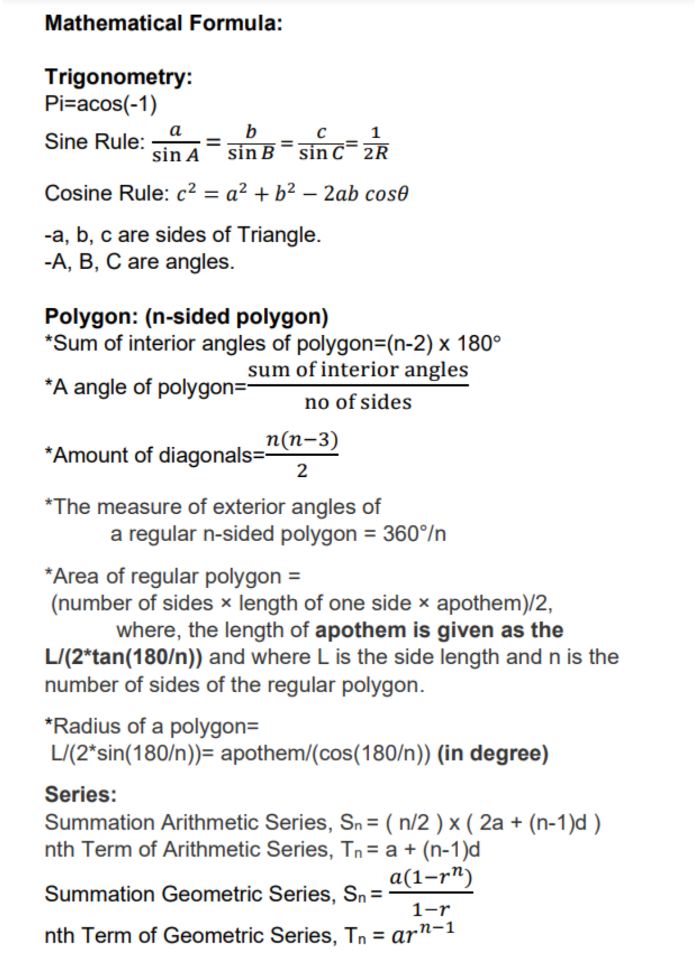
```
ll n, m;
cin>>n>>m;
ll setbits=0;
fr(m){
    ll l, r, x;
    cin>>l>>r>>x;
    setbits|=x;}
ll tmp=bigmod(2,n-1,mod);
cout<<((setbits%mod)*tmp)%mod<<nl;
```

Note:

1. a+b=(a xor b)+2*(a&b)
2. N=a^p * b^q * c^r -> nod=(p+1)*(q+1)*(r+1)

$$\text{S.O.D} = \frac{a^{p+1}-1}{a-1} * \frac{b^{q+1}-1}{b-1} * \frac{c^{r+1}-1}{c-1}$$

3. Pick's Theorem: A = I + (B/2) -1, A = Area of Polygon, B = Number of integral points on edges of polygon, I = Number of integral points strictly inside the polygon
4. floor(n/j)=x for j in [floor(n/(x+1))+1, floor(n/x)].
5. Rotation of a n*n matrix s[i][j] -> s[j][n-i-1] -> s[n-i-1][n-j-1] -> s[n-j-1][i]



Mathematical Formula:

Trigonometry:

Pi=acos(-1)

Sine Rule: $\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C} = \frac{1}{2R}$

Cosine Rule: $c^2 = a^2 + b^2 - 2ab\ cos\theta$

-a, b, c are sides of Triangle.
-A, B, C are angles.

Polygon: (n-sided polygon)

*Sum of interior angles of polygon=(n-2) x 180°

*A angle of polygon=$\frac{\text{sum of interior angles}}{\text{no of sides}}$

*Amount of diagonals=$\frac{n(n-3)}{2}$

*The measure of exterior angles of
a regular n-sided polygon = 360°/n

*Area of regular polygon =
(number of sides × length of one side × apothem)/2,
where, the length of **apothem is given as the L/(2*tan(180/n))** and where L is the side length and n is the number of sides of the regular polygon.

*Radius of a polygon=
L/(2*sin(180/n))= apothem/(cos(180/n)) **(in degree)**

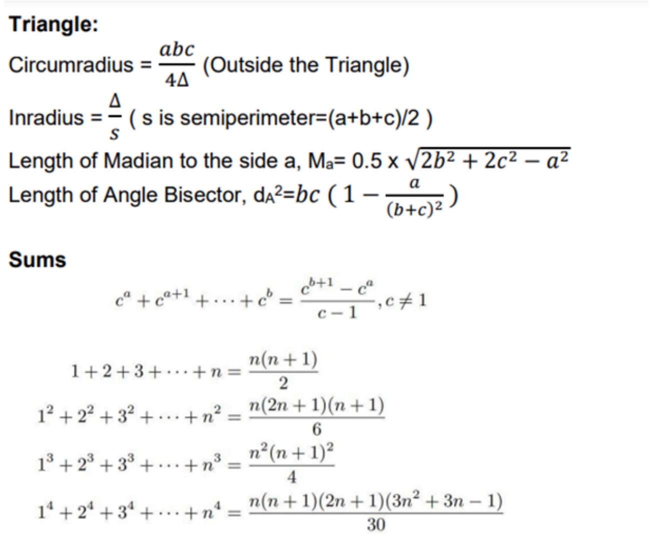
Series:

Summation Arithmetic Series, S_n = (n/2) x (2a + (n-1)d)

nth Term of Arithmetic Series, T_n = a + (n-1)d

Summation Geometric Series, S_n = $\frac{a(1-r^n)}{1-r}$

nth Term of Geometric Series, T_n = ar^{n-1}



Triangle:

Circumradius = $\frac{abc}{4\Delta}$ (Outside the Triangle)

Inradius = $\frac{\Delta}{s}$ (s is semiperimeter=(a+b+c)/2)

Length of Madian to the side a, M_a= 0.5 x $\sqrt{2b^2 + 2c^2 - a^2}$

Length of Angle Bisector, d_A^2=$bc\ (1 - \frac{a}{(b+c)^2})$

Sums

$$c^a + c^{a+1} + \cdots + c^b = \frac{c^{b+1} - c^a}{c - 1}, c \neq 1$$

$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$

$$1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(2n+1)(n+1)}{6}$$

$$1^3 + 2^3 + 3^3 + \cdots + n^3 = \frac{n^2(n+1)^2}{4}$$

$$1^4 + 2^4 + 3^4 + \cdots + n^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$$