**Suffix Array**:

```
int wa[mx],wb[mx],wv[mx],Ws[mx];
/*(1-indexed) sa[i] = starting position (0...n-1) of ith
lexicographically smallest suffix in s*/
/*(0-indexed) Rank[i] = lexicographical rank of s[i....n-1] ((i+1)th
suffix by position)*/
/*LCP[i] = longest common prefix of sa[i] & sa[i-1]*/
int sa[mx],Rank[mx],LCP[mx];
int cmp(int *r,int a,int b,int l) {return r[a]==r[b] &&
r[a+l]==r[b+l];}
/*m = maximum possible ASCII value of a string character
(alphabet size)
also, m = maximum number of distinct character in string (when
compressed)*/
void buildSA(string s,int* sa,int n,int m){
    int i,j,p,*x=wa,*y=wb,*t;
    for(i=0; i<m; i++) Ws[i]=0;
    for(i=0; i<n; i++) Ws[x[i]=s[i]]++;
    for(i=1; i<m; i++) Ws[i]+=Ws[i-1];
    for(i=n-1; i>=0; i--) sa[--Ws[x[i]]]=i;
    for(j=1,p=1; p<n; j<<=1,m=p){
        for(p=0,i=n-j; i<n; i++) y[p++]=i;
        for(i=0; i<n; i++) if(sa[i]>=j) y[p++]=sa[i]-j;
        for(i=0; i<n; i++) wv[i]=x[y[i]];
        for(i=0; i<m; i++) Ws[i]=0;
        for(i=0; i<n; i++) Ws[wv[i]]++;
        for(i=1; i<m; i++) Ws[i]+=Ws[i-1];
        for(i=n-1; i>=0; i--) sa[--Ws[wv[i]]]=y[i];
        for(t=x,x=y,y=t,p=1,x[sa[0]]=0,i=1; i<n; i++)
           x[sa[i]]=cmp(y,sa[i-1],sa[i],j) ? p-1 : p++;
    }
}
//Kasai's LCP algorithm (O(n))
void buildLCP(string s,int *sa,int n){
    int i,j,k=0;
    for(i=1; i<=n; i++) Rank[sa[i]]=i;
    for(i=0; i<n; LCP[Rank[i++]]=k)
        for(k?k--:0, j=sa[Rank[i]-1]; s[i+k]==s[j+k]; k++);
    return;}
// Pattern Subtring hisbe ace kina
bool Pattern(string &text,string &pat)
{
    int lo=1,hi=text.size();
    while(lo<=hi)
    {
        int mid=(lo+hi)/2;
        int ok=0;
        for(int i=0;i<pat.size();i++)
        {
            if(text[i+sa[mid]]>pat[i]) {ok=1;break;}
            if(text[i+sa[mid]]<pat[i]) {ok=-1;break;}
        }
        if(!ok) return true;
        if(ok>0) hi=mid-1;
        else lo=mid+1;
    }
    return false;
}
pair<int,int> Patterntern_occurence(string Text ,string Pattern)
{
    int n=Text.size();
    int m=Pattern.size();
    int be=1,en=n;
    while(be<en)
    {
        int mid = (en+be)/2;
        int ok=0;
        for(int i=0;i<m;i++)
        {
            if(Text[i+sa[mid]]>Pattern[i]){ok=1;break;}
            if(Text[i+sa[mid]]<Pattern[i]){ok=-1;break;}
        }
        if(ok+1) en=mid;
        else be=mid+1;
    }
    bool ok = 1;
    for(int i=0;i<m;i++) if(Text[i+sa[be]]!=Pattern[i]){ok=0;break;}
    if(!ok) return {-1,-1};
    pair<int,int> re;
    re.first=be;
    be=1,en=n;
    while(be<en)
    {
        int mid = (en+be)/2;
        int ok=0;
        for(int i=0;i<m;i++)
        {
            if(Text[i+sa[mid]]>Pattern[i]){ok=1;break;}
            if(Text[i+sa[mid]]<Pattern[i]){ok=-1;break;}
        }
        if(ok>0) en=mid;
        else be=mid+1;
    }
    ok = 1;
    for(int i=0;i<m;i++) if(Text[i+sa[en]]!=Pattern[i]){ok=0;break;}
    if(!ok) en--;
    re.second=en;
    return re;
}
/// this is for LCP from index i to index j.
/* just run a query from min(Rank[i-1],Rank[j-1])+1 to
max(Rank[i-1],Rank[j-1])*/
int ST[mx][22];
int Jump_LOG[mx];
void Build_Sparse(int n)
{
            for(int i=1;i<=n;i++)ST[i][0]=LCP[i];
            for(int i=2;i<=n;i++)Jump_LOG[i]=Jump_LOG[i-1]+!(i&(i-
1));
            for(int j=1;(1<<j)<=n;j++)
            {
                        for(int i=1;(i+(1<<j)-1)<=n;i++)
                        {
        ST[i][j]=min(ST[i][j-1],ST[i+(1<<(j-1))][j-1]);
```

```
                }
        }
}
int query(int i,int j)
{
        int boro_lav=Jump_LOG[j-i+1];
        return min(ST[i][boro_lav],ST[j-
(1<<boro_lav)+1][boro_lav]);
}
void solve()
{
    buildSA(s,sa,n+1,130); //Important
    buildLCP(s,sa,n);
    for(int i=1;i<=n;i++) cout<<sa[i]<<" "; cout<<endl;
    for(int i=0;i<n;i++) cout<<Rank[i]<<" "; cout<<endl;
    for(int i=1;i<=n;i++) cout<<LCP[i]<<" ";
   pair<int,int>re=Patterntern_occurence(s,t);
   if(re.second==-1)printf("0\n");
  else printf("%d\n",re.second-re.first+1 );
}
```

**Aho Corasick:**

```
struct Aho_Corasick
{
        int Trie[mx][27],Suffix_Link[mx];
        vector<int> Mark[mx];
        int Node;
        void Init()
        {
                fill(Trie[0],Trie[0]+26,-1);
                Mark[0].clear();
                Node=0;
        }
        void Insert(char ch[],int idx)
        {
                int len=strlen(ch);
                int cur=0;
                for(int i=0;i<len;i++)
                {
                        int val=ch[i]-'a';
                        if(Trie[cur][val]==-1)
                           {
Trie[cur][val]=++Node;
        fill(Trie[Node],Trie[Node]+26,-1);
                      Mark[Node].clear();
                              }
                        cur=Trie[cur][val];
                }
                Mark[cur].push_back(idx);
        }
        void Cal_Suffix_Link()
        {
                queue<int>q;
                Suffix_Link[0]=0;
                for(int i=0;i<26;i++)
                {
                        if(Trie[0][i]!=-1)
                        {
                                q.push(Trie[0][i]);
                                Suffix_Link[Trie[0][i]]=0;
                        }
                        else Trie[0][i]=0;
                }
                while(!q.empty())
                {
                        int u=q.front();
                        q.pop();
                        for(int v: Mark[Suffix_Link[u]])
                        {
                                Mark[u].push_back(v);
                        }
                        for(int i=0;i<26;i++)
          {
            if(Trie[u][i] != -1)
            {
              Suffix_Link[Trie[u][i]] =   Trie[Suffix_Link[u]][i];
              q.push(Trie[u][i]);
            }
            else
              Trie[u][i] = Trie[Suffix_Link[u]][i];
          }
                }
         }
}Automata;
/// Pattern Occurence Count
int cnt[mx];
void Count_Pattern(char ch[])
{
   int cur=0;
   int len=strlen(ch);
   for(int i=0;i<len;i++)
   {
        int val=ch[i]-'a';
        cur= Automata.Trie[cur][val];
        for(int id: Automata.Mark[cur])cnt[id]++;
   }
}
void solve()
{
        char ch1[1000005],ch[mx];
        scanf("%d%s",&n,ch1);
        Automata.Init();
        for(int i=0;i<n;i++)
        {
                scanf("%s",ch);
                Automata.Insert(ch,i);
        }
        Automata.Cal_Suffix_Link();
        Count_Pattern(ch1);
        /// print Occurence Frequency
        for(int i=0;i<n;i++)
        {
                printf("%d\n",cnt[i]);
                cnt[i]=0;
        }
}
```

**Hashing:**

```
/*backup prime 307,367,1040160883,1066517951
,1e9+7,1e9+9,1072857881,1000004249 */
struct Hash_dui
{
        ll base,mod;
        int sz;
        vector<int>Rev,Forw,P;
        Hash_dui(){}
        Hash_dui(const char* s,ll b,ll m)
        {
                sz=strlen(s),base=b,mod=m;

        Rev.resize(sz+2,0),Forw.resize(sz+2,0),P.resize(sz+2,1);
    for(int i=1;i<=sz;i++)P[i]=(base*P[i-1])%mod;
    for(int i=1;i<=sz;i++)Forw[i]=(Forw[i-1]*base+(s[i-1]-'a'+1))%mod; /// digit hole s[i-1]-'0'
    for(int i=sz;i>=1;i--)Rev[i]=(Rev[i+1]*base+(s[i-1]-'a'+1))%mod;  ///alphabet hole s[i-1]-'a'
        }
  void Single_char_ad(char cc)
  {
        P.push_back((P.back()*base)% mod);
    Forw.push_back((Forw.back()*base+(cc-'a'+1))% mod);
  }
  inline int Range_Hash(int l,int r)
  {
        int re_hash=Forw[r+1]-((ll)P[r-l+1]*Forw[l]%mod);
        if(re_hash<0)re_hash+=mod;
        return re_hash;
  }
  inline int Reverse_Hash(int l,int r)
  {
        int re_hash=Rev[l+1]-((ll)P[r-l+1]*Rev[r+2]%mod);
        if(re_hash<0)re_hash+=mod;
        return re_hash;
  }
};
struct Hash_Main
{
        Hash_dui h1,h2;
        Hash_Main(){}
        Hash_Main(const char* s)
        {
                h1=Hash_dui(s,1949313259, 2091573227);
                h2=Hash_dui(s,1997293877, 2117566807);
        }
        void Char_Add(char cc)
        {
                h1.Single_char_ad(cc);
                h2.Single_char_ad(cc);
        }
        inline ll Range_Hash(int l,int r) /// O base index
        {
                return ((ll)h1.Range_Hash(l,r)<<32)^h2.Range_Hash(l,r);
        }
        inline ll Reverse_Hash(int l,int r) /// O base index
        {
                return ((ll)h1.Reverse_Hash(l,r)<<32)^h2.Reverse_Hash(l,r);
        }
};
void solve()
{
        int n;
        scanf("%d%s",&n,ch);
        string re=ch;
        Hash_Main h_ek(ch);
        ll h1=h_ek(l,r)//0 base
}
```

**Manachers:**

```
int oddPlen[mx],evenPlen[mx];
void Manachers()
{
  int l=0,r=-1;
  for(int i=0;i<n;i++)
  {
    int k=(i>r)?1:min(oddPlen[l+r-i],r-i+1);
    while(k<=i && i+k<n && ch[i-k]==ch[i+k]) k++;
    oddPlen[i]=k--;
    if(i+k>r){
     l=i-k;
     r=i+k;
    }
  }
  l=0,r=-1;
  for(int i=0;i<n;i++)
  {
    int k=(i>r)?0:min(evenPlen[l+r-i+1],r-i+1);
    while(k+1<=i && i+k<n && ch[i-k-1]==ch[i+k])k++;
    evenPlen[i]=k--;
    if(i+k>r){
     l=i-k-1;
     r=i+k;
    }
  }
}
void solve()
{   Manachers();
  for(int i=0;i<n;i++)printf("%d %d\n",oddPlen[i]*2-1,evenPlen[i]*2);
}
```

**Pi Table / Prefix Functions:**

```
vector<int> Create_Pi_Table(const char* s)
{
        int sz=strlen(s);
        vector<int>pi(sz);
        for(int i=1;i<sz;i++)
        {
            int j=pi[i-1];
            while(j>0 && s[i]!=s[j])j=pi[j-1];
            if(s[j]==s[i])j++;
```

```
                pi[i]=j;
            }
            return pi;
}
void solve()
{           vector<int> pi=Create_Pi_Table(ch);
            for(int i=0;i<n;i++)printf("%d\n",pi[i] );
}
```

**Tree Hash:**

```
vector<int>g[mx][2];
int sub[mx][2];
ll H[mx][2];
ll Base[]={1040160883,1066517951};
ll mod[]={1072857881,1000004249};
ll mul(ll a,ll b,int ty)
{
  a*=b;
  if(a>=mod[ty])a%=mod[ty];
  return a;
}
ll add(ll a,ll b,int ty)
{
  a+=b;
  if(a>=mod[ty])a-=mod[ty];
  return a;
}
void pre()
{
            H[0][0]=H[0][1]=1;
   for(int i=1;i<mx;i++)
   for(int j=0;j<1;j++)H[i][j]=mul(H[i-1][j],Base[j],j);
}
pair<ll,ll> get_hash(int u,int l,int ty)
{
  sub[u][ty]=1;
  pair<ll,ll> re={0,0};
  for(int v:g[u][ty])
  {
    pair<ll,ll>tem=get_hash(v,l+1,ty);
    re.first=add(re.first,tem.first,0);
    re.second=add(re.second,tem.second,1);
    sub[u][ty]+=sub[v][ty];
  }

re.first=add(re.first,mul(add(H[l][0],sub[u][ty],0),sub[u][ty],0),0)
;

re.second=add(re.second,mul(add(H[l][1],sub[u][ty],1),sub[u][ty
],1),1);
  return re;
}
void solve(){
  pair<ll,ll> val1=get_hash(1,1,0);
  pair<ll,ll> val2=get_hash(1,1,1);
}
```

**HLD(value in edge):**

```
vector<pair<int,int>>g[mx];
int par[mx],sub_sz[mx];
int Head[mx],st[mx],sesh[mx];
int Rin[mx];  /// Segment Tree er init ye Tree[bode]=ar[Rin[be]]
likte hobe
int T;
using namespace Segment_Tree;
void sz_dfs(int u,int p)
{
            sub_sz[u]=1;
            par[u]=p;
            for(auto &v: g[u])
            {
                        if(v.first==p)continue;
                        sz_dfs(v.first,u);
                        sub_sz[u]+=sub_sz[v.first];
            if(sub_sz[v.first]>sub_sz[g[u][0].first])swap(v,g[u][0]);
            }
}
void hld_dfs(int u,int p,int cost)
{
    st[u]=++T;
    Rin[st[u]]=u;
    ar[st[u]]=cost; /// node ye nai , sgement tree build array
    for(auto v:g[u])
    {
            if(v.first==p)continue;
            Head[v.first]= (v.first==g[u][0].first ? Head[u]:v.first);
            hld_dfs(v.first,u,v.second);
    }
    sesh[u]=T;
}
void hld_build(int root)
{
            T=0;
            Head[root]=root;
            sz_dfs(root,root);
            hld_dfs(root,root,0);
}
bool Is_it_parent(int p,int u)
{
            return st[p]<=st[u] && sesh[u]<=sesh[p];
}
int path_query(int u,int v)
{
            int re=-inf;
            while(1)
            {
                        if(Is_it_parent(Head[u],v))break;
                        re=max(re,query(1,1,n,st[Head[u]],st[u]));
/*for sum we  will do just add all query sum*/
                        u=par[Head[u]];
            }
            swap(u,v);
            while(1)
            {
                        if(Is_it_parent(Head[u],v))break;
```

```
                re=max(re,query(1,1,n,st[Head[u]],st[u]));
/*for sum we  will do just add all query sum*/
                u=par[Head[u]];
        }
        if(st[u]>st[v])swap(u,v);
        re=max(re,query(1,1,n,st[u]+1,st[v])); /// node hole st[u] theke start
        return re;
}
void path_update(int u,int v,int val)
{
        while(1)
        {
                if(Is_it_parent(Head[u],v))break;
                Rupdate(1,1,n,st[Head[u]],st[u],val);
                u=par[Head[u]];
        }
        swap(u,v);
        while(1)
        {
                if(Is_it_parent(Head[u],v))break;
                Rupdate(1,1,n,st[Head[u]],st[u],val);
                u=par[Head[u]];
        }
        if(st[u]>st[v])swap(u,v);
        Rupdate(1,1,n,st[u]+1,st[v],val); /* node hole st[u] theke start*/
}
void update_subtree(int u,int val)
{
        Rupdate(1,1,n,st[u]+1,sesh[u],val);
}
```

**1D Sparse Table:**

```
///it will be work for range min,max,or,and with no update
int ST[mx][MAX_logN];
int Jump_LOG[mx];
void Build_Sparse()
{
        for(int i=1;i<=n;i++)ST[i][0]=ar[i];
        for(int i=2;i<=n;i++)Jump_LOG[i]=Jump_LOG[i-1]+!(i&(i-1));
        for(int j=1;(1<<j)<=n;j++)
        {
                for(int i=1;(i+(1<<j)-1)<=n;i++)
                {
      ST[i][j]=min(ST[i][j-1],ST[i+(1<<(j-1))][j-1]);
                }
        }
}
int query(int i,int j)
{
        int boro_lav=Jump_LOG[j-i+1];
        return min(ST[i][boro_lav],ST[j-(1<<boro_lav)+1][boro_lav]);
}
```

**2D Sparse (Rectangle):**

```
int ST[mx][mx][MAX_logN][MAX_logN];
void Build_2D_Sparse()
{
        for(int i=1;i<=n;i++)
        {
                for(int j=1;j<=n;j++)
                {
                        ST[i][j][0][0]=ar[i][j];
                }
                for(int l=1;(1<<l)<=n;l++)
                {
                        int pre=1<<(l-1);

                        for(int j=1;j+pre<=n;j++)
                        {
                                ST[i][j][0][l]=min(ST[i][j][0][l-1],ST[i][j+pre][0][l-1]);
                        }
                }
        }
        for(int l=1;(1<<l)<=n;l++)
        {
                int pre=1<<(l-1);
                for(int i=1;i+pre<=n;i++)
                {
                        for(int k=0;(1<<k)<=n;k++)
                        {
                                for(int j=1;j<=n;j++)
                                {

        ST[i][j][l][k]=min(ST[i][j][l-1][k],ST[i+pre][j][l-1][k]);
                                }
                        }
                }
        }
}
int query(int i,int j,int p,int q)  /// two point
{
        int boro_jum1=log2(p-i+1);
        int boro_jum2=log2(q-j+1);
        int pre1=1<<boro_jum1;
        int pre2=1<<boro_jum2;
        int re1=min(ST[i][j][boro_jum1][boro_jum2],ST[i][q-pre2+1][boro_jum1][boro_jum2]);
        int re2=min(ST[p-pre1+1][j][boro_jum1][boro_jum2],ST[p-pre1+1][q-pre2+1][boro_jum1][boro_jum2]);
        return min(re1,re2);
}
```

**2D Sparse (Square):**

```
int ST[mx][mx][MAX_logN];
void Build_2D_Sparse()
{
        for(int l=0;(1<<l)<=n;l++)
        {
                for(int i=1;i+(1<<l)-1<=n;i++)
                {
                        for(int j=1;j+(1<<l)-1<=n;j++)
                        {
                                if(l==0)ST[i][j][l]=ar[i][j];
                                else
                                {
                                        int pre=1<<(l-1);
                                        int
val1=min(ST[i][j][l-1],ST[i+pre][j][l-1]);
                                        int
val2=min(ST[i][j+pre][l-1],ST[i+pre][j+pre][l-1]);

        ST[i][j][l]=min(val1,val2);
                                }
                        }
                }
        }
}
int query(int i,int j,int sz)
{
        int boro_lav=log2(sz);
        int pre=1<<(boro_lav);
        int val1=min(ST[i][j][boro_lav],ST[i+sz-
pre][j][boro_lav]);
        int val2=min(ST[i][j+sz-pre][boro_lav],ST[i+sz-pre][j+sz-
pre][boro_lav]);
        return min(val1,val2);
}
```

**MO:**

```
namespace MO
{
  const int N=100005;
  const int Q=100005;
  int ar[N],BlockId[N],ans[Q];
  bool vis[N];
  struct node
  {
        int l,r,id;
        node(){}
        node(int l,int r,int id)
        {
                this->l=l;
                this->r=r;
                this->id=id;
        }
    bool operator < (const node& u)
    {
        int a=BlockId[l],b=BlockId[u.l];
        if(a==b)
        {
                return (a & 1 ? (r > u.r) : (r < u.r));
        }
        else return a<b;

    }
  }query[Q];
  int boro=0;
  int cnt[mx],cnt_tot[mx];
  void check(int pos)
  {
        if(vis[pos])
        {
                cnt_tot[cnt[ar[pos]]]--;
                cnt[ar[pos]]--;
                if(cnt[ar[pos]])cnt_tot[cnt[ar[pos]]]++;
                if(cnt_tot[boro]==0)boro--;

        }
        else
        {
                if(cnt[ar[pos]])cnt_tot[cnt[ar[pos]]]--;
                cnt[ar[pos]]++;
                cnt_tot[cnt[ar[pos]]]++;
                if(cnt_tot[boro+1])boro++;

        }
        vis[pos]^=1;
  }
}
using namespace MO;
void solve()
{
        int q;
        boro=0;
        scanf("%d%d",&n,&q);
        int sz=sqrt(n);
        for(int i=1;i<=n;i++)
        {
                BlockId[i]=i/sz;
                vis[i]=false;
                scanf("%d",&ar[i]);
        }
        memset(cnt,0,sizeof(cnt));
        memset(cnt_tot,0,sizeof(cnt_tot));
        for(int i=1;i<=q;i++)
        {
                int x,y;
                scanf("%d%d",&x,&y);
                query[i]=node(x,y,i);
        }
        sort(query+1,query+q+1);
        int left=query[1].l;
        int right=left-1;
        for(int i=1;i<=q;i++)
        {
                node Now=query[i];
                while(left<Now.l)check(left++);
                while(left>Now.l)check(--left);
                while(right<Now.r)check(++right);
                while(right>Now.r)check(right--);
    ans[Now.id]=boro;
        } }
```

**MO's On tree:**

```
int n,m,ii,k,LOG;
int depth[mx];
int par[mx][25];
```

```
namespace MO
{
  const int N=100005;
  const int Q=100005;
  int ar[N],br[N],BlockId[N],ans[Q];
  bool vis[N];
  struct node
  {
       int l,r,id,lca;
       node(){}
       node(int l,int r,int lca,int id)
       {
              this->l=l;
              this->r=r;
              this->lca=lca;
              this->id=id;
       }
    bool operator < (const node& u)
    {
       int a=BlockId[l],b=BlockId[u.l];
       return (a==b)?(r<u.r):a<b;
    }
  }query[Q];
  int re=0,sz;
  int cnt[100005];
  void check(int pos)
  {
       if(vis[pos])
       {
              if(cnt[ar[pos]]==1)re--;
              cnt[ar[pos]]--;
       }
       else
       {
              if(cnt[ar[pos]]==0)re++;
              cnt[ar[pos]]++;
       }
       vis[pos]^=1;
  }
  vector<int> g[N];
  int Euler[2*N],st[N],en[N],Time;
  void dfs(int u,int p,int lvl)
  {
   st[u]=++Time;
   Euler[Time]=u;
   par[u][0]=p;
   depth[u]=lvl;
   for(int v:g[u])
   {
       if(v==p)continue;
       dfs(v,u,lvl+1);
   }
   en[u]=++Time;
   Euler[Time]=u;
  }
}
using namespace MO;
```

```
void init(int root)
{
  dfs(root,-1,1);
  for(int j=1;j<LOG;j++)
  {
    for(int i=1;i<=n;i++)
    {
      if(par[i][j-1]!=-1)
      {
        par[i][j]=par[par[i][j-1]][j-1];
      }
      else par[i][j]=-1;
    }
  }
}
int lca(int u,int v)
{
  if(depth[u]<depth[v])swap(u,v);
  int log=1;
  while(1)
  {
    int next=log+1;
    if(depth[u]<(1<<next))break;
    log++;
  }
  for(int i=log;i>=0;i--)
  {
    if(depth[u]-(1<<i)>=depth[v])
    {
      u=par[u][i];
    }
  }
  if(u==v)return u;
  for(int i=log;i>=0;i--)
  {
    if(par[u][i]!=-1 && par[u][i]!=par[v][i])
    {
      u=par[u][i];
      v=par[v][i];
    }
  }
  return par[v][0];
}
void solve()
{
       int q;
       scanf("%d%d",&n,&q);
       LOG=log2(n)+1;
       Time=0;
       re=0;
       sz=sqrt(n);
       for(int i=1;i<=n;i++)

       scanf("%d",&ar[i]),br[i]=ar[i],BlockId[i]=i/sz,vis[i]=false,
cnt[i]=0;
       // Compressing Coordinates . its a alternative of map
       sort(br+1,br+n+1);
```

```
            k = unique(br+1,br+n+1)-br-1;
            for(int i=1;i<=n;i++)
ar[i]=lower_bound(br+1,br+k+1,ar[i])-br;
            for(int i=1;i<n;i++)
            {
                        int x,y;
                        scanf("%d%d",&x,&y);
                        g[x].push_back(y);
                        g[y].push_back(x);
            }
            init(1);
            for(int i=1;i<=q;i++)
            {
     int x,y;
     scanf("%d%d",&x,&y);
     if(st[x]>st[y])swap(x,y);
     int p=lca(x,y);
     if(x==p)query[i]=node(st[x],st[y],-1,i);
     else query[i]=node(en[x],st[y],p,i);
            }
            sort(query+1,query+1+q);
   int left=query[1].l;
            int right=left-1;
            for(int i=1;i<=q;i++)
            {
                        node Now=query[i];
                        while(left<Now.l)check(Euler[left++]);
                        while(left>Now.l)check(Euler[--left]);
                        while(right<Now.r)check(Euler[++right]);
                        while(right>Now.r)check(Euler[right--]);
      if(Now.lca!=-1)check(Now.lca);
      ans[Now.id]=re;
      if(Now.lca!=-1)check(Now.lca);
            }
            for(int i=1;i<=q;i++)printf("%d\n",ans[i]);
   for(int i=1;i<=n;i++)g[i].clear();
}
```

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

```
int Trie[mx][2];
int End[mx];
int ar[50005];
int Trie[50000*32][2];
int n,ii,st=1;
void Insert(int val)
{
   int cur=1;
   for(int i=31;i>=0;i--)
   {
      int bit=0;
      if(((1<<i) & val))bit=1;
      if(Trie[cur][bit]==0)Trie[cur][bit]=++st;
      cur=Trie[cur][bit];
   }
   End[cur]=val;
}
int query_min(int val)
{
   int cur=1;
   for(int i=31;i>=0;i--)
   {
      int bit=0;
      if(((1<<i) & val))bit=1;
      if(Trie[cur][bit])cur=Trie[cur][bit];
      else if(Trie[cur][bit^1])cur=Trie[cur][bit^1];
   }
   return End[cur]^val;
}
int query_max(int val)
{
   int cur=1;
   for(int i=31;i>=0;i--)
   {
      int bit=0;
      if(((1<<i) & val))bit=1;
      if(Trie[cur][bit^1])cur=Trie[cur][bit^1];
      else if(Trie[cur][bit])cur=Trie[cur][bit];
   }
   return End[cur]^val;
}
void solve()
{
   int suffix=0;
   int re_min=INT_MAX,re_max=0;
   Insert(0);
   for(int i=1;i<=n;i++)
   {
      scanf("%d",&ar[i]);
      suffix^=ar[i];
      re_min=min(re_min,query_min(suffix));
      re_max=max(re_max,query_max(suffix));
      Insert(suffix);
   }
}
```

**BIT:**

```
struct BIT
{
   int Tree[N+5][N+5];
   void init()
   {
            memset(Tree,0,sizeof(Tree));
   }
   ll query(int idx,int idy)
   {
      ll re=0;
      int tem=idy;
      while(idx)
      {
            idy=tem;
            while(idy)
            {
            re+=Tree[idx][idy];
            idy-=idy&-idy;
         }
         idx-=idx&-idx;
```

```
    }
    return re;
  }
  void update(int idx,int idy,ll val)
  {
        int tem=idy;
    while(idx<=N)
    {
        idy=tem;
        while(idy<=N)
        {
        Tree[idx][idy]+=val;
        idy+=idy&-idy;
      }
      idx+=idx&-idx;
    }
  }
  int Rquery(int l,int r)
  {
    return query(r)-query(l-1);
  }
  void Rupdate(int l,int r,int val)
  {
    update(l,val);
    update(r+1,val*-1);
  }
};
```

**SegTree 1D:**

```
namespace Segment_Tree
{
        const int N=200005;
        int Tree[N*4];
        int Lazy[N*4];
        void Relax(int node,int be,int en)
        {
                if(!Lazy[node])return;
                Tree[node]+=Lazy[node];
                if(be!=en)
                {
                        Lazy[node*2]+=Lazy[node];
                        Lazy[node*2+1]+=Lazy[node];
                }
                Lazy[node]=0;
        }
        void init(int node,int be,int en)
        {
                Lazy[node]=0;
    if(be==en)
    {
        Tree[node]=ar[be];
        return;
    }
    int mid=(be+en)/2;
    init(node*2,be,mid);
    init(node*2+1,mid+1,en);
    Tree[node]=Tree[node*2]+Tree[node*2+1];
        }
        void update(int node,int be,int en,int pos,int val)
        {
                Relax(node,be,en);
                if(be> pos || en<pos)return;
                if(be==en)
                {
                        Tree[node]+=val;
                        return;
                }
                int mid=(be+en)/2;
                update(node*2,be,mid,pos,val);
                update(node*2+1,mid+1,en,pos,val);

        Tree[node]=max(Tree[node*2],Tree[node*2+1]);
        }
        void Rupdate(int node,int be,int en,int i,int j,int val)
        {
                Relax(node,be,en);
                if(be>j || en<i)return ;
                if(be>=i && en<=j)
                {
                        Lazy[node]+=val;
                        Relax(node,be,en);
                        return;
                }
                int mid=(be+en)/2;
                Rupdate(node*2,be,mid,i,j,val);
                Rupdate(node*2+1,mid+1,en,i,j,val);

        Tree[node]=max(Tree[node*2],Tree[node*2+1]);
        }
        int query(int node,int be,int en,int i,int j)
        {
                Relax(node,be,en);
                if(be>j || en<i)return 0;
                if(be>=i && en<=j)return Tree[node];
                int mid=(be+en)/2;
                return
max(query(node*2,be,mid,i,j),query(node*2+1,mid+1,en,i,j));
        }
}
```

**Is array index L to R holds a permutations :**

```
struct info
{
        ll pre_sum;
        ll xor_sum;
        int id1,id2;
};
info pre[mx];
void func()
{
        for(int i=1;i<=mx-5;i++)
        {
                pre[i].pre_sum=pre[i-1].pre_sum+i;
                pre[i].xor_sum=pre[i-1].xor_sum^i;
                pre[i].id1=1;
                pre[i].id2=i;
        }
}
info Merge(info a,info b)
{
        info c;
        c.pre_sum=a.pre_sum+b.pre_sum;
        c.xor_sum=a.xor_sum^b.xor_sum;
        c.id1=min(a.id1,b.id1);
        c.id2=max(a.id2,b.id2);
        return c;
}
/*if pre[r-l+1] equals to info of L TO R of array then they holds permutations.  */
```

**Bracket Sequence:**

```
struct info
{
   int open,close,ans;
};
info Merge(info a,info b)
{
   info re;
   int valid=min(a.open,b.close);
   re.open=a.open+b.open-valid;
   re.close=a.close+b.close-valid;
   re.ans=a.ans+b.ans+valid;    /* this code works for maximum length of correct bracket sequence in l to r range*/
   /* if you want to see is it valid bracet squence length just change
   re.ans=re.open+re.close;
   In query if re.ans gives 0 thats main the range is correct bracket sequence
   */
   return re;
}
```

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

```
struct info
{
        ll  max_pref,max_suf,ans,sum;
   void Merge(info p1,info p2)
        {
          sum=p1.sum+p2.sum;
max_pref=max(p1.max_pref,p1.sum+p2.max_pref);
max_suf=max(p2.max_suf,p2.sum+p1.max_suf);
ans=max(max(p1.ans,p2.ans),p1.max_suf+p2.max_pref);
        }
};

void Relax(int node,int be,int en)
{
        if(!cur[node])return;
        Tree[node].sum=Lazy[node]*(en-be+1);
        Tree[node].max_pref=max(0LL,Tree[node].sum);
        Tree[node].max_suf=max(0LL,Tree[node].sum);
        Tree[node].ans=max(0LL,Tree[node].sum);
        if(be!=en)
        {
                Lazy[node*2]=Lazy[node];
                Lazy[node*2+1]=Lazy[node];
                cur[node*2]=true;
                cur[node*2+1]=true;
        }
        cur[node]=false;
        Lazy[node]=0;
}
```

**Centroid Decomposition:**

```
 int dis[18][mx],re[mx],vis[mx];
int p[mx],sub[mx],lvl[mx];
vector<int>g[mx],ng[mx];
/* p[u] = parent of u in centroid tree
dis[x][u] = distance from u to a parent of u at level x of centroid tree
if u is in subtree of centroid c, then dis[lvl[c]][u] = dist(c, l)
If (x, y) edge exist, then x must be in g[y] and y must be in g[x]*/
/* we can do more pre work in dfs function*/
void dfs(int l,int u,int par)
{
        if(par!=-1)dis[l][u]=dis[l][par]+1;
        for(int v:g[u])
                if(v!=par && !vis[v])dfs(l,v,u);
}
int centroid(int u,int par,int r)
{
        for(int v:g[u])
                if(v!=par && !vis[v] && sub[v]>r)return centroid(v,u,r);
        return u;
}
void pre_cal(int u,int par)
{
        sub[u]=1;
        for(int v:g[u])
                if(v!=par && !vis[v])pre_cal(v,u),sub[u]+=sub[v];
}
void decompose(int u,int par)
{
        pre_cal(u,-1);
        int tem=centroid(u,-1,sub[u]>>1);
        vis[tem]=1,p[tem]=par,lvl[tem]=0;
        if(par!=-1)lvl[tem]=lvl[par]+1,ng[par].push_back(tem);
        dfs(lvl[tem],tem,-1);
        for(int v:g[tem])
                if(v!=par && !vis[v])decompose(v,tem);
}
void update(int u)
{
        for(int v=u;v!=-1;v=p[v])
                re[v]=min(re[v],dis[lvl[v]][u]);
}
int query(int u)
{
        int ans=1e9;
        for(int v=u;v!=-1;v=p[v])
                ans=min(ans,re[v]+dis[lvl[v]][u]);
        return ans;
}
int lca(int u,int v)
{
        if(lvl[u]<lvl[v])swap(u,v);
        while(lvl[u]>lvl[v])u=p[u];
        while(u!=v && p[u]!=-1)u=p[u],v=p[v];
        return u;
}
int dist(int u,int v)
{
        int lc=lca(u,v);
        return dis[lvl[lc]][u]+dis[lvl[lc]][v];
}
int GetRoot(int u)
{
        while(p[u]!=-1)u=p[u];
        return u;
}
// at first call decompose(1,-1)
```

**Dinic:** O(EV*V)

```
const ll eps = 0;
struct edge {
  int a, b;
  ll cap,flow;
  int yo, x, y;
};
struct Dinic {
  int s,t,d[mx], ptr[mx] ;
  //int Id[mx][mx];
  vector<edge>e;
  vector<int>g[mx];
  void init() {
    e.clear();
    memset(d,0,sizeof(d));
    for(int i = 0; i < mx ; i++)g[i].clear();
    // for(int i=0;i<mx;i++)
    // {
    //    for(int j=0;j<mx;j++)
    //    {
    //              Id[i][j]=0;
    //    }
    // }
  }
  void addEdge(int a,int b,ll cap, int x = -1, int y= -1) {
    edge e1 = { a, b, cap, 0, 1, x, y } ;
    edge e2 = { b, a, 0, 0, 0, x, y } ;
   //  Id[a][b]=e.size();
    g[a].push_back((int)e.size());
    e.push_back(e1);
   //  Id[b][a]=e.size();
    g[b].push_back((int)e.size());
    e.push_back(e2);
  }
  bool bfs() {
    queue < int > Q ;
    Q.push(s);
    memset(d,-1,sizeof(d));
    d[s]=0 ;
    while (!Q.empty()) {
      int u=Q.front() ;
      Q.pop() ;
      for(int i=0; i<g[u].size(); i++) {
        int id=g[u][i];
        int v=e[id].b;
       //  printf("%d %d %0.3lf
%0.3lf\n",u,v,e[id].cap,e[id].flow);
        if(d[v]==-1&&e[id].flow<e[id].cap) {
          Q.push(v) ;
          d[v]=d[u]+1 ;
        }
      }
    }
    return d[t]!=-1 ;
  }
  ll dfs(int u,ll flow) {
    if (flow<=eps)  return 0 ;
    if ( u==t )  return flow ;
    for(int& i = ptr[u] ; i<g[u].size(); i++) {
      int id = g[u][i];
      int v =  e[id].b ;
      if ( d[v] != d[u]+1 )  continue ;
      ll pushed = dfs (v,min (flow,e[id].cap-e[id].flow)) ;
      //cout << "pushed " << pushed << endl;
      if (pushed>eps) {
        e [id].flow+=pushed ;
        e [id^1].flow-=pushed ;
        return pushed ;
      }
    }
    return 0 ;
  }
  ll dinic() {
    ll flow = 0 ;
    while(true) {
      if(!bfs())  break ;
      memset(ptr, 0, sizeof(ptr)) ;
      while (true){
        ll pushed = dfs(s,INF );
        if(pushed<=eps)break;
        flow += pushed ;
      }
    }
    return flow ;
  }
};
```

**Hopcroft_Karp:** O((E+V)*S),S=max matching

```
#define mx 40005
#define INF (1<<28)
struct Hopcroft_Karp
{
        vector< int > g[mx];
        int n, m, Matching[mx], Distance[mx];
        /* n: number of nodes on left side, nodes are
numbered 1 to n*/
        /* m: number of nodes on right side, nodes are
numbered n+1 to n+m*/
        // G = 0[0] ∪ G1[G[1---n]] ∪ G2[G[n+1---n+m]]
        void init(int num)
        {
    for(int
i=0;i<=num;i++)Matching[i]=0,Distance[i]=0,g[i].clear();
        }
  void addEdge(int u,int v)
  {
        g[u].push_back(v);
  }
        bool bfs() {
          int i, u, v, len;
          queue< int > q;
          for(i=1; i<=n; i++) {
            if(Matching[i]==0) {
              Distance[i] = 0;
              q.push(i);
            }
            else Distance[i] = INF;
          }
          Distance[0] = INF;
          while(!q.empty()) {
            u = q.front(); q.pop();
            if(u!=0) {
              for(int v:g[u]) {
                  if(Distance[Matching[v]]==INF) {
                  Distance[Matching[v]] = Distance[u] + 1;
                  q.push(Matching[v]);
                }
              }
            }
          }
          return (Distance[0]!=INF);
        }
        bool dfs(int u) {
          int i, v, len;
          if(u!=0) {
            for(int v:g[u]) {
              if(Distance[Matching[v]]==Distance[u]+1) {
                if(dfs(Matching[v])) {
                  Matching[v] = u;
                  Matching[u] = v;
                  return true;
                }
              }
            }
            Distance[u] = INF;
            return false;
          }
          return true;
        }
        int hopcroft_karp() {
          int Matchinging = 0, i;
          while(bfs())
            for(i=1; i<=n; i++)
              if(Matching[i]==0 && dfs(i))
                Matchinging++;
          return Matchinging;
        }
};
Hopcroft_Karp hk;
```

**Min Cost Max Flow:**

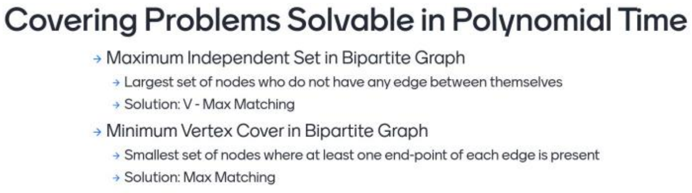
```
typedef long long T1;//for cost
typedef long long T2;//for flow
const int maxn = 20100;
const T1 INF = 1e12;
const T2 inf = 1e12;
const T1 eps = 0;
struct Edge {
  int from, to;
  T2 cap, flow;
  T1 cost;
};

int n,m,k,ii;
struct MCMF {//0-indexed
  int n, m, s, t;
  vector<Edge> edges;
  vector<int> G[maxn];
  int p[maxn],inq[maxn];
  T1 d[maxn];
  T2 a[maxn];
  void init() {
    for(int i = 0; i < n; i++) G[i].clear();
    edges.clear();
  }
  void AddEdge(int from,int to,T2 cap,T1 cost) {
    edges.push_back((Edge){from, to, cap, 0, cost});
    edges.push_back((Edge){to, from, 0, 0, -cost});
    m = edges.size();
    G[from].push_back(m-2);
    G[to].push_back(m-1);
  }
  pair<T1,T2> Mincost() {//bellmanFord
    T1 tot_cost = 0;
    T2 tot_flow = 0;
    while(true) {
      for(int i = 0; i < n; i++) d[i] = INF;
      d[s] = 0;
      p[s] = 0;
      a[s] = inf;
      bool up=true;
      while(up) {
        up=false;
        for(int u = 0; u < n; u++) {
          if(d[u]-INF>=-eps)continue;
          for(int j:G[u]) {
            Edge &e=edges[j];
            if(e.cap > e.flow && d[e.to] > d[u] + e.cost+eps) {
              d[e.to] = d[u] + e.cost;
```

```
                  p[e.to] = j;
                  a[e.to] = min(a[u], e.cap - e.flow);
                  up=true;
               }
            }
         }
      }
      if(abs(d[t]-INF)<=eps)break;
      tot_cost += (T1)d[t] * a[t];
      tot_flow += (T2)a[t];
      int u = t;
      while(u != s) {
         edges[p[u]].flow += a[t];
         edges[p[u]^1].flow -= a[t];
         u = edges[p[u]].from;
      }
   }
   return {tot_cost,tot_flow};
}
pair<T1,T2> Mincost2() {//SPFA
   T1 tot_cost = 0;
   T2 tot_flow = 0;
   while(true) {
      for(int i = 0; i < n; i++) d[i] = INF;
      memset(inq, 0, sizeof(inq));
      d[s] = 0;
      inq[s] = 1;
      p[s] = 0;
      a[s] = inf;
      queue<int> Q;
      srand(time(NULL));
      Q.push(s);
      while(!Q.empty()) {
         int u = Q.front();
         Q.pop();
         inq[u] = 0;
         for(int i = 0; i < G[u].size(); i++) {
            Edge& e = edges[G[u][i]];
            if(e.cap > e.flow && d[e.to] > d[u] + e.cost+eps) {
               d[e.to] = d[u] + e.cost;
               p[e.to] = G[u][i];
               a[e.to] = min(a[u], e.cap - e.flow);
               if(!inq[e.to]) {
                  Q.push(e.to);
                  inq[e.to] = 1;
               }
            }
         }
      }
      if(abs(d[t]-INF)<=eps)break;
      tot_cost += (T1)d[t] * a[t];
      tot_flow += a[t];
      int u = t;
      while(u != s) {
         edges[p[u]].flow += a[t];
         edges[p[u]^1].flow -= a[t];
         u = edges[p[u]].from;
         }
      }
      return {tot_cost,tot_flow};
   }
} mcmf;
```

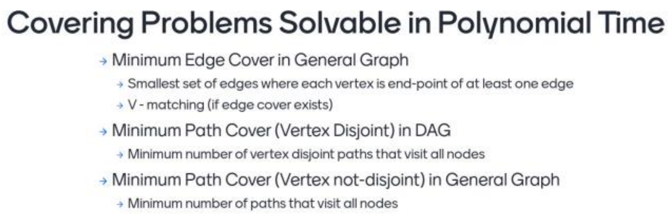
**Kuhn:**

```
struct BPM
{
            bool Done[mx];
   vector<int>g[mx];
   int macth[mx];
   void addEdge(int u,int v)
   {
      g[u].push_back(v);
   }
   void init()
   {
            for(int i=0;i<mx;i++)g[i].clear();
   }
   bool Tem_Matching(int u)
   {
      for(int i=0;i<(int)g[u].size();i++)
      {
            int v=g[u][i];
         if(Done[v]) continue;
         Done[v] = true;
         if(macth[v]==-1 || Tem_Matching(macth[v]))
         {
            macth[v] = u;
            return true;
         }
      }
      return false;
   }
   int Max_Matching(int num)
   {
            // Be Careful with this section. when passin num.
      memset(macth,-1,sizeof(macth));
      int re = 0;
      for(int i=1;i<=num;i++)
      {
         memset(Done,false,sizeof(Done));
         if(Tem_Matching(i)) re++;
      }
      return re;
   }
};
```



Covering Problems Solvable in Polynomial Time

- Maximum Independent Set in Bipartite Graph
  - Largest set of nodes who do not have any edge between themselves
  - Solution: V - Max Matching
- Minimum Vertex Cover in Bipartite Graph
  - Smallest set of nodes where at least one end-point of each edge is present
  - Solution: Max Matching



Covering Problems Solvable in Polynomial Time

- Minimum Edge Cover in General Graph
  - Smallest set of edges where each vertex is end-point of at least one edge
  - V - matching (if edge cover exists)
- Minimum Path Cover (Vertex Disjoint) in DAG
  - Minimum number of vertex disjoint paths that visit all nodes
- Minimum Path Cover (Vertex not-disjoint) in General Graph
  - Minimum number of paths that visit all nodes

**MDST:**

```
const int inf = 1e9 ;
struct edge {
  int u, v, w;
  edge() {}
  edge(int a,int b,int c) : u(a), v(b), w(c) {}
  bool operator < (const edge& o) const {
    if (u == o.u)
      if (v == o.v)return w < o.w;
      else  return v < o.v;
    return u < o.u;
  }
};
int dmst(vector<edge> &edges, int root) { // 0 base node 0 to n-1
  int ans = 0;
  int cur_nodes = n;
  while (true) {
    vector<int> lo(cur_nodes, inf), pi(cur_nodes, inf);
    for (int i = 0; i < edges.size(); ++i) {
      int u = edges[i].u, v = edges[i].v, w = edges[i].w;
      if (w < lo[v] and u != v) {
        lo[v] = w;
        pi[v] = u;
      }
    }
    lo[root] = 0;
    for (int i = 0; i < lo.size(); ++i) {
      if (i == root) continue;
      if (lo[i] == inf) return -1;
    }
    int cur_id = 0;
    vector<int> id(cur_nodes, -1), mark(cur_nodes, -1);
    for (int i = 0; i < cur_nodes; ++i) {
      ans += lo[i];
      int u = i;
      while (u != root and id[u] < 0 and mark[u] != i) {
        mark[u] = i;
        u = pi[u];
      }
      if (u != root and id[u] < 0) { // Cycle
        for (int v = pi[u]; v != u; v = pi[v]) id[v] = cur_id;
        id[u] = cur_id++;
      }
    }
    if (cur_id == 0) break;
    for (int i = 0; i < cur_nodes; ++i)
      if (id[i] < 0) id[i] = cur_id++;
    for (int i = 0; i < edges.size(); ++i) {
      int u = edges[i].u, v = edges[i].v, w = edges[i].w;
      edges[i].u = id[u];
      edges[i].v = id[v];
      if (id[u] != id[v]) edges[i].w -= lo[v];
    }
    cur_nodes = cur_id;
    root = id[root];
  }
  return ans;
}
```

**LCA(value on edge):**

```
int par[mx][20];
ll ans[mx][20];
int depth[mx],LOG;
vector<pair<int,ll>>g[mx];
void dfs(int u,int p,int lvl)
{
  par[u][0]=p;
  depth[u]=lvl;
  for(auto it:g[u])
  {
    int v=it.first;
    ll w=it.second;
    if(v==p)continue;
    ans[v][0]=w;
    dfs(v,u,lvl+1);
  }
}
void init(int root)
{
  dfs(root,-1,1);
  for(int j=1;j<LOG;j++)
  {
    for(int i=1;i<=n;i++)
    {
      if(par[i][j-1]!=-1)
      {
        par[i][j]=par[par[i][j-1]][j-1];
        ans[i][j]=max(ans[i][j-1],ans[par[i][j-1]][j-1]);
      }
      else par[i][j]=-1;
    }
  }
}
ll query(int u,int v)
{
        if(u==v)return 0;
  if(depth[u]<depth[v])swap(u,v);
  int diff=depth[u]-depth[v];
  ll re=0;
  for(int i=LOG-1;i>=0;i--)
  {
    if(diff>=(1<<i))
    {
      diff-=(1<<i);
      re=max(re,ans[u][i]);
      u=par[u][i];
    }
```

```
   }
   if(u==v)return re;
   for(int i=LOG-1;i>=0;i--)
   {
      if( par[u][i]!=par[v][i])
      {
         re=max({re,ans[u][i],ans[v][i]});
         u=par[u][i];
         v=par[v][i];
      }
   }
   re=max({re,ans[u][0],ans[v][0]});
   return re;
}
int dist(int u,int v)
{
   return depth[u]+depth[v]-2*depth[lca(u,v)];
}
int kth_parent(int u,int k)
{
   for(int i=LOG-1;i>=0;i--)
   {
      if(k>=(1<<i))
      {
         k-=(1<<i);
         u=par[u][i];
      }
      if(u==-1)return u;
   }
   return u;
}
solve()
{  for(int i=1;i<=n;i++)
            {
                        g[i].clear();
                        for(int j=0;j<LOG;j++)ans[i][j]=0,par[i][j]=-1;
            }
 LOG=log2(n)+1;
}
```

**LCA(value in node):**

```
//dfs function ye ans[u][0] line likha jabe nah
// init function same
// query function er sesh ye ei 3 line likhbo
   re=max(re,ans[u][0]);
   re=max(re,ans[v][0]);
   re=max(re,ans[par[v][0]][0]);

for(int i=1;i<=n;i++)
{
            scanf("%d",&ar[i]);
            ans[i][0]=ar[i];
}
```

**DSU:**

```
int Size[mx];
int Findparent(int x)
{
   return (x==parent[x])?x:(parent[x]=Findparent(parent[x]));
}
void Union(int x,int y)
{
   int px=Findparent(x);
   int py=Findparent(y);
   if(px==py)return;
   if(Size[px]>Size[py])
   {
      Size[px]+=Size[py];
      parent[py]=px;
   }
   else
   {
      Size[py]+=Size[px];
      parent[px]=py;    }
}
void initialize()
{
   for(int i=0;i<=n;i++)parent[i]=i,Size[i]=1;
}
```

**Bellman Ford:**

```
vector<Edge>E;
ll dist[100];
bool bellman_ford()
{
   /* here i can start from 1 .if given that stating node i can set dist[src]=0*/
   for(int i=1;i<=n;i++)dist[i]=10000000;
   dist[1]=0;
   for(int i=1;i<n;i++)
      for(Edge it: E)
         if(dist[it.v]>dist[it.u]+it.w)
            dist[it.v]=dist[it.u]+it.w;
   for(Edge it:E)
      if(dist[it.v]>dist[it.u]+it.w)return true;//negative cycle
   return false;
}
```

**Floyed Warshal:**

```
for(int i=1;i<=n;i++)
{
            for(int j=1;j<=n;j++)
            {
                        if(i==j || dis[i][j]>0)continue;
                        dis[i][j]=1e18;
            }
}
for(int l=1;l<=n;l++)
{
            for(int i=1;i<=n;i++)
            {
                        for(int j=1;j<=n;j++)
                        {
                                    dis[i][j]=min(dis[i][j],dis[i][l]+dis[l][j]);
                        }
            }
}
```

**Articulation Point:**

```
vector<int>g[mx];
int articular_point[mx];
int st[mx],low[mx];
int Time=1;
int dfs(int u,int p)
{
   st[u]=low[u]=Time++;
   int child=0;
   for(auto it:g[u])
   {
      if(it==p)continue;
      if(st[it]==0)
      {
         child++;
         dfs(it,u);
         if(st[u]<=low[it])articular_point[u]=1;
         low[u]=min(low[u],low[it]);
      }
      else low[u]=min(low[u],st[it]);

   }
   return child;
}

void solve()
{
  for(int i=1;i<=n;i++)
  {
     if(st[i])continue;
     articular_point[i]=(dfs(i,-1)>1);
  }
}
```

**Articulations Bridge:**

```
vector<int>g[mx];
vector<pair<int,int>>Bridge;
int st[mx],low[mx];
int Time=1;
void dfs(int u,int p)
{
   st[u]=low[u]=Time++;
   int child=0;
   for(auto it:g[u])
   {
      if(it==p)continue;
      if(st[it]==0)
      {
         dfs(it,u);
         if(st[u]<low[it])Bridge.push_back({u,it});
         low[u]=min(low[u],low[it]);
      }
      else low[u]=min(low[u],st[it]);
   }
}
void solve()
{
  for(int i=1;i<=n;i++)
  {
     if(st[i])continue;
     dfs(i,-1);
  }
}
```

**Strongly Connected Component:**

```
vector<int>g[mx],g_rev[mx],st(mx),en(mx),component[mx],option,visit;
vector<pair<int,int>>dekhi;
int node,edge,cnt,tem;
int mp[mx];
void dfs1(int u)
{
   visit[u]=true;
   st[u]=++cnt;
   for(auto it:g[u])
   {
      if(visit[it])continue;
      dfs1(it);
   }
   en[u]=++cnt;
}
void dfs2(int u)
{
   visit[u]=true;
   component[cnt].push_back(u);
   for(auto it:g_rev[u])
   {
      if(visit[it])continue;
      dfs2(it);
   }
}
void clean()
{
   for(int i=1;i<=node+2;i++)
   {
      g[i].clear();
      g_rev[i].clear();
      component[i].clear();
   }
   option.clear();
   cnt=0;
   st.clear();
   en.clear();
   dekhi.clear();
   memset(mp,0,sizeof(mp));
}
void solve()
{
   scanf("%d%d",&node,&edge);
   for(int i=1;i<=edge;i++)
   {
      int u,v;
      scanf("%d%d",&u,&v);///directed graph
      g[u].push_back(v);
      g_rev[v].push_back(u);
      mp[u]++;
      mp[v]++;
```

```
  }
  visit.assign(node+2,false);
  for(int i=1;i<=node;i++)
  {
    if(visit[i]==true || mp[i]==0)continue;
    dfs1(i);
  }
  for(int i=1;i<=node;i++)
  {
    if(visit[i]==true && mp[i])dekhi.push_back({en[i],i});
  }
  sort(dekhi.begin(),dekhi.end());
  reverse(dekhi.begin(),dekhi.end());
  visit.assign(node+2,false);
  cnt=1;
  for(int i=0;i<dekhi.size();i++)
  {
    int pos=dekhi[i].second;
    if(visit[pos] || mp[pos]==0)continue;
    dfs2(pos);
    cnt++;
  }
  for(int i=1;i<cnt;i++)
  {
    for(auto it:component[i])
    {
      cout<<it<<" ";
    }
    cout<<endl;
  }}
```

**Matrix Expo:**

```
#define MAX 105
#define ll long long int
const ll MOD = 1e9 + 7;
const ll MOD2 = MOD * MOD * 3;
inline ll bigMod(ll a,ll b){
  ll res=1;
  while(b){
    if(b&1) res=(res*a)%MOD;
    a=(a*a)%MOD; b>>=1;
  }
  return res;
}
inline ll inv(ll n) {return bigMod(n,MOD-2);}
inline ll Mul(ll a,ll b) {return (a*b)%MOD;}
inline ll Div(ll a,ll b) {return Mul(a,inv(b));}
/* 1 base row columun index */
struct Matrix{
  int row, col;
  ll m[MAX][MAX];
  Matrix() {memset(m,0,sizeof(m));}
  void Set(int r,int c) {row = r; col = c;}
  Matrix(int r,int c) {memset(m,0,sizeof(m)); Set(r,c);}
  void normalize(){
    for(int i=1; i<=row; i++){
      for(int j=1; j<=col; j++){
        m[i][j] %= MOD;
        if(m[i][j] < 0) m[i][j] += MOD;
      }
    }
  }
};
Matrix Multiply(Matrix A,Matrix B){
  Matrix ans(A.row,B.col);
  for(int i=1;i<=A.row;i++){
    for(int j=1;j<=B.col;j++){
      ans.m[i][j]=0;
      ll sm = 0;
      for(int k=1;k<=A.col;k++){
        sm+=(A.m[i][k]*B.m[k][j]);
        if(sm >= MOD2) sm -= MOD2;
      }
      ans.m[i][j] = sm % MOD;
    }
  }
  return ans;
}
Matrix Power(Matrix mat,ll p){
  Matrix res(mat.row , mat.col);
  Matrix ans(mat.row , mat.col);
  int n = ans.row;
  for(int i=1;i<=n;i++){
    for(int j=1;j<=n;j++){
      ans.m[i][j]=0;
      res.m[i][j]=mat.m[i][j];
    }
    ans.m[i][i]=1;
  }
  while(p){
    if(p&1) ans=Multiply(ans,res);
    res=Multiply(res,res);
    p=p/2;
  }
  return ans;
}
```

**Gaussian Elimination:**

```
/*format : (a[0]*x[0]+a[1]*x[1] ... a[m-1]*x[m-1]) % k = a[m],
where 0 <= ai < k*/
//number of solution : k^(number of free variable) = k^(n-rank)
ll A[105][105];
ll X[105];
int Rank;
ll gcdExtended(ll a, ll b, ll& x, ll& y){
  if(a==0) {x=0;y=1; return b;}
  ll x1,y1;
  ll gcd = gcdExtended(b%a,a,x1,y1);
  x=y1-(b/a)*x1;
  y=x1;
  return gcd;
}
ll modinverse(ll x,ll y) {ll a,b; gcdExtended(x,y,a,b); return a;}
//n equations (n rows), m variables (m+1 columns)
void Gauss(int n,int m,int k){
  int r,c;
  for(r=0,c=0;r<n && c<m;c++){
    for(int i=r+1;i<n;i++) if(abs(A[i][c])>abs(A[r][c])) swap(A[i],A[r]);
    if(!A[r][c]) continue;
    ll s = modinverse(A[r][c],k);
    for(int i=r+1;i<n;i++) if(A[i][c]){
      ll w = (s*A[i][c])%k;
      /* s bhag hobe r A[i][c] gun hobe*/
      for(int j=c;j<=m;j++){A[i][j]-=(A[r][j]*w)%k; A[i][j]%=k; if(A[i][j]<0) A[i][j]+=k;}
    }
    r++;
  }
  //Rank = r
  for(int i=r;i<n;i++) if(A[i][m]) return;  //No solution
  //Unique Solution for r variables
  for(int i=r-1;i>=0;i--){
    X[i]=A[i][m];
    for(int j=i+1;j<m;j++) {X[i]-=(A[i][j]*X[j])%k; X[i]%=k; if(X[i]<0) X[i]+=k;}

    ll inv=modinverse(A[i][i],k);
    X[i]=(X[i]*inv)%k; if(X[i]<0) X[i]+=k;
  }
}
```

**3D Prefix Sum:**

```
for(int i=1; i<=n; i++)
    for(int j=1; j<=m; j++)
      for(int k=1; k<=l; k++)
        prefix[i][j][k]=prefix[i-1][j][k]+prefix[i][j-1][k]+prefix[i][j][k-1]
                -prefix[i-1][j-1][k]-prefix[i-1][j][k-1]-prefix[i][j-1][k-1]
                +prefix[i-1][j-1][k-1]+ar[i][j][k];
for(int x1=1; x1<=n; x1++)
  {
    for(int x2=x1; x2<=n; x2++)
    {
      for(int y1=1; y1<=m; y1++)
      {
        for(int y2=y1; y2<=m; y2++)
        {
          for(int z1=1; z1<=l; z1++)
          {
            for(int z2=z1; z2<=l; z2++)
            {
              ll re=prefix[x2][y2][z2]-prefix[x1-1][y2][z2]-prefix[x2][y1-1][z2]-prefix[x2][y2][z1-1]
                  +prefix[x1-1][y1-1][z2]+prefix[x1-1][y2][z1-1]+prefix[x2][y1-1][z1-1]
                  -prefix[x1-1][y1-1][z1-1];
              ans=max(re,ans);
            }
          }
        }
      }
    }
```

}**Inclusion Exclusion:**

```
/* koto gulo number ace[1,n] jara a1 or a2 or a3...,am dara divide*/
/* if m=3 and 3 values are a1,a2,a3 then*/
/* |a1 U a2 U a3|=|a1|+|a2|+|a3|-|a1 union a2|-|a2 union a3|-|a1 union a3|+|a1 union a2 union a3|*/
/// if number of cadidate is odd do add or do substract
/// time complexity 2^m.
/// for better perform use recusive
void func(int idx,int cnt,ll lcm)
{
  if(lcm>n)return;
  if(idx==m)
  {
    if(cnt==0)return;

    if(cnt & 1)re1+=n/lcm;
    else re1-=n/lcm;
    return;
  }
  func(idx+1,cnt+1,(lcm*ar[idx])/__gcd(lcm,(ll)ar[idx]));
  func(idx+1,cnt,lcm);
}
void solve()
{
  scanf("%lld%d",&n,&m);
  for(int i=0;i<m;i++)scanf("%d",&ar[i]);
  ///using bitmask
  for(int i=1; i<(1<<m);i++)
  {
    ll lcm=1;
    int cnt=0;
    for(int j=0;j<m;j++)
    {
      if(i & (1<<j))
      {
        cnt++;
        lcm=(lcm*ar[j])/__gcd(lcm,(ll)ar[j]);
        if(lcm>n)break;
      }
    }
    if(cnt&1)re+=n/lcm;
    else re-=n/lcm;
  }
}
```

**Linear sieve:**

```
bitset<mx>is_composite;
vector<int>prime;
int phi[mx],mobius[mx];
void seive(int n)
{
  phi[1]=mobius[1]=1;
  for(int i=2;i<=n;i++)
  {
              mobius[i]=1;
    if(!is_composite[i])
    {
      prime.push_back(i);
      phi[i]=i-1;            ///i is prime
    }
    for(int j=0;j<prime.size() && i*prime[j]<=n;j++)
    {
      is_composite[i*prime[j]]=true;
      if(i%prime[j]==0)
      {
        phi[i*prime[j]]=phi[i]*prime[j];   ///prime[j] divides i
        break;
      }
      else
      {
        phi[i*prime[j]]=phi[i]*phi[prime[j]];  ///prime[j] do not divide i
      }
    }
  }
  for(int val:prime)
  {
    int temp=val*val;
                if(temp>n)break;
    for(int j=temp;j<=n;j+=temp)mobius[j]=0;
  }
  for(int val:prime)
  {
    for(int j=val;j<=n;j+=val)mobius[j]*=-1;
  }
}
```

**CRT:**

```
ll ar[mx],br[mx];
struct GCD_type { ll x, y, d; };
GCD_type ex_GCD(ll a, ll b)
{
  if (b == 0) return {1, 0, a};
  GCD_type pom = ex_GCD(b, a % b);
  return {pom.y, pom.x - a / b * pom.y, pom.d};
}
ll normalize(ll val,ll mod)
{ val%=mod;
 if(val<0)val+=mod;
 return val;
}
void solve(){
 ll ans=br[1]; /// here br remainder
 ll lcm=ar[1];
 bool f=true;
 for(int i=2;i<=n;i++)
 {
   auto pom=ex_GCD(lcm,ar[i]);
   ll x1=pom.x;
   ll d=pom.d;
   if((br[i]-ans)%d!=0)
   {
     f=false;break;
   }
   ans=ans+x1*(br[i]-ans)/d%(ar[i]/d)*lcm;
   ans=normalize(ans,lcm*ar[i]/d);
   lcm=(lcm*ar[i])/__gcd(lcm,ar[i]);
 }
 if(f)printf("%lld %lld\n",ans,lcm);  /* here is the smallest answer .next xth answer will be ans+x*lcm where x=[1,2,....]*/
}
```

**Extended Euclidean (inverse):**

```
int Extended_Euclidean(int a,int b,int &x,int &y)
{
        if(b==0)
        {
                x=1;y=0;
                return a;
        }
        int d=Extended_Euclidean(b,a%b,y,x);
        y=y-(a/b)*x;
        return d;
}
int Inverse_Modulo(int a,int m)
{
        int x,y,d;
        d=Extended_Euclidean(a,m,x,y);
        if(d==1) return (x+m)%m;
        return -1; //No Solution
}
```

**Big Mod, Fact:**

```
ll bigmod(ll e,ll x)
{
  if(!x)return 1;
  ll p=bigmod(e,x/2);
  p=(p*p)%mod;
  if(x%2)p=(p*e)%mod;
  return p;
}
void fact_cal(){
  fact[0]=1,inv[0]=1;
  for(int i=1;i<=mx-3;i++)
  {
    fact[i]=(fact[i-1]*i)%mod;
  }
  inv[mx-3]=bigmod(fact[mx-3],mod-2);
  for(int i=mx-4;i>=1;i--)inv[i]=(inv[i+1]*(i+1))%mod;
}
```

**Stirling Number of 2nd kind:**

```
ll dp[mx][mx];
ll func(int nn,int kk)
{
  if(kk==1)return 1;
  if(nn==kk)return 1;
  if(kk==0)return 0;
  ll &val=dp[nn][kk];
  if(val!=-1)return val;
  val=func(nn-1,kk-1)+1LL*kk*func(nn-1,kk);
  return val;
}
```

**Pollard RHO:**

```
#define pii pair<ll,int>
```

```
ll Mul(ll a,ll b,ll Mod){
    ll Ans=0;
    while(b){
        if(b&1) {Ans+=a; if(Ans>=Mod) Ans-=Mod;}
        a+=a; if(a>=Mod) a-=Mod;
        b>>=1;
    }
    return Ans;
}
ll bigMod(ll n,ll r,ll Mod){
    if(r==0) return 1LL;
    ll ret=bigMod(n,r/2,Mod);
    ret=Mul(ret,ret,Mod);
    if(r%2==1) ret=Mul(ret,n,Mod);
    return ret;
}
//Miller-Rabin
bool witness(ll wit,ll n){
  if(wit>=n) return false;

  int s=0; ll t=n-1;
  while(t%2==0) s++,t/=2;

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

  for(int i=1;i<s;i++){
    wit=Mul(wit,wit,n);
    if(wit==1) return true;
    if(wit==n - 1) return false;
  }
  return true;
}
//Is n prime?
bool miller(ll n){
  if(n==2) return true;
  if(n%2==0 || n<2) return false;
  if(witness(2,n) || witness(7,n) || witness(61,n)) return false;
  return true;
}
// Pollard's Rho
// a must not equal 0 or -2.
/* returns a divisor, a proper one when succeeded, equal to n if failed*/
// in case of failure, change a
ll rho(ll n,ll a) {
  auto f=[&](ll x) {return (Mul(x,x,n)+a)%n; };
  ll x=2,y=2;
  for(int i=1;;i++){
    x=f(x); y=f(f(y));
    ll d=__gcd(n,abs(x-y));
    if(d!=1) return d;
  }
  return n;
}
ll get_factor(ll n){
  if(n%2==0) return 2;
  if(n%3==0) return 3;
  if(n%5==0) return 5;
  while(true){
   ll a=2+rand()%100;
   ll d=rho(n,a);
   if(d!=n) return d;
  }
  return n;
}
void factorize(ll n,vector<ll> &x) {
  if(n==1) return;
  else if(miller(n)) x.push_back(n);
  else{
   ll d=get_factor(n);
   factorize(d,x);
   factorize(n/d,x);
  }
}
vector<ll>factorize(ll n) {vector<ll>x; factorize(n, x); return x;}
vector<pii>Factors; // store factor
vector<ll>Divisors;//strore divisors
void findDiv(int pos,ll val){
    if(pos<0) {Divisors.push_back(val); return;}
    ll Now=1;
    for(int i=0;i<=Factors[pos].second;i++){
        findDiv(pos-1,val*Now);
        Now=Now*Factors[pos].first;
    }
}
void findAllDiv(ll n){
    vector<ll>now=factorize(n);
    sort(now.begin(),now.end());
    Factors.clear();
    Divisors.clear();
    int Count=1;
    for(int i=1;i<now.size();i++){
        if(now[i]==now[i-1]) Count++;
        else {Factors.push_back({now[i-1],Count}); Count=1;}
    }
    Factors.push_back({now.back(),Count});
    findDiv(Factors.size()-1,1);
}
```

**2D Geometry:**

```
const double pi = 4 * atan(1);
const double eps = 1e-6;
inline int dcmp (double x) { if (fabs(x) < eps) return 0; else return x < 0 ? -1 : 1; }
double fix_acute(double th) {return th<-pi ? (th+2*pi): th>pi ? (th-2*pi) : th;}
inline double getDistance (double x, double y) { return sqrt(x * x + y * y); }
inline double torad(double deg) { return deg / 180 * pi; }
struct Point {
    double x, y;
    Point (double x = 0, double y = 0): x(x), y(y) {}
    void read () { scanf("%lf%lf", &x, &y); }
    void write () { printf("%lf %lf", x, y); }
    bool operator == (const Point& u) const { return dcmp(x - u.x) == 0 && dcmp(y - u.y) == 0; }
    bool operator != (const Point& u) const { return !(*this == u); }
    bool operator < (const Point& u) const { return dcmp(x - u.x) < 0 || (dcmp(x-u.x)==0 && dcmp(y-u.y) < 0); }
    bool operator > (const Point& u) const { return u < *this; }
    bool operator <= (const Point& u) const { return *this < u || *this == u; }
    bool operator >= (const Point& u) const { return *this > u || *this == u; }
    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); }
    double operator * (const Point& u) { return x*u.y - y*u.x; }
};
typedef Point Vector;
typedef vector<Point> Polygon;
struct Line {
    double a, b, c;
    Line (double a = 0, double b = 0, double c = 0): a(a), b(b), c(c) {}
};
struct Segment{
    Point a;
    Point b;
    Segment(){}
    Segment(Point aa,Point bb) {a=aa,b=bb;}
};
struct DirLine {
    Point p;
    Vector v;
    double ang;
    DirLine () {}
    DirLine (Point p, Vector v): p(p), v(v) { ang = atan2(v.y, v.x); }
    bool operator < (const DirLine& u) const { return ang < u.ang; }
};
namespace Punctual {
    double getDistance (Point a, Point b) { double x=a.x-b.x, y=a.y-b.y; return sqrt(x*x + y*y); }
};
namespace Vectorial {
    double getDot (Vector a, Vector b) { return a.x * b.x + a.y * b.y; }
    double getCross (Vector a, Vector b) { return a.x * b.y - a.y * b.x; }
    double getLength (Vector a) { return sqrt(getDot(a, a)); }
    double getPLength (Vector a) { return getDot(a, a); }
    double getAngle (Vector u) { return atan2(u.y, u.x); }
    double getSignedAngle (Vector a, Vector b) {return getAngle(b)-getAngle(a);}
    Vector rotate (Vector a, double rad) { return Vector(a.x*cos(rad)-a.y*sin(rad), a.x*sin(rad)+a.y*cos(rad)); }
    Vector ccw(Vector a, double co, double si) {return Vector(a.x*co-a.y*si, a.y*co+a.x*si);}
    Vector cw (Vector a, double co, double si) {return Vector(a.x*co+a.y*si, a.y*co-a.x*si);}
    Vector scale(Vector a, double s = 1.0) {return a / getLength(a) * s;}
    Vector getNormal (Vector a) { double l = getLength(a); return Vector(-a.y/l, a.x/l); }
};
namespace ComplexVector {
    typedef complex<double> Point;
    typedef Point Vector;
    double getDot(Vector a, Vector b) { return real(conj(a)*b); }
    double getCross(Vector a, Vector b) { return imag(conj(a)*b); }
    Vector rotate(Vector a, double rad) { return a*exp(Point(0, rad)); }
};
namespace Linear {
    using namespace Vectorial;
    Line getLine (double x1, double y1, double x2, double y2) { return Line(y2-y1, x1-x2, y1*x2-x1*y2); }
    Line getLine (double a, double b, Point u) { return Line(a, -b, u.y * b - u.x * a); }
    bool getIntersection (Line p, Line q, Point& o) {
        if (fabs(p.a * q.b - q.a * p.b) < eps)
            return false;
        o.x = (q.c * p.b - p.c * q.b) / (p.a * q.b - q.a * p.b);
        o.y = (q.c * p.a - p.c * q.a) / (p.b * q.a - q.b * p.a);
        return true;
    }
    bool getIntersection (Point p, Vector v, Point q, Vector w, Point& o) {
        if (dcmp(getCross(v, w)) == 0) return false;
        Vector u = p - q;
        double k = getCross(w, u) / getCross(v, w);
        o = p + v * k;
        return true;
    }
    double getDistanceToLine (Point p, Point a, Point b) { return fabs(getCross(b-a, p-a) / getLength(b-a)); }
    double getDistanceToSegment (Point p, Point a, Point b) {
        if (a == b) return getLength(p-a);
        Vector v1 = b - a, v2 = p - a, v3 = p - b;
        if (dcmp(getDot(v1, v2)) < 0) return getLength(v2);
        else if (dcmp(getDot(v1, v3)) > 0) return getLength(v3);
        else return fabs(getCross(v1, v2) / getLength(v1));
    }
    double getDistanceSegToSeg (Point a,Point b,Point c,Point d){
        double Ans=INT_MAX;
        Ans=min(Ans,getDistanceToSegment(a,c,d));
        Ans=min(Ans,getDistanceToSegment(b,c,d));
        Ans=min(Ans,getDistanceToSegment(c,a,b));
        Ans=min(Ans,getDistanceToSegment(d,a,b));
        return Ans;
    }
    Point getPointToLine (Point p, Point a, Point b) { Vector v = b-a; return a+v*(getDot(v, p-a) / getDot(v,v)); }
    bool onSegment (Point p, Point a, Point b) { return dcmp(getCross(a-p, b-p)) == 0 && dcmp(getDot(a-p, b-p)) <= 0; }
```

```
  bool haveIntersection (Point a1, Point a2, Point b1, Point b2) {
    if(onSegment(a1,b1,b2)) return true;
    if(onSegment(a2,b1,b2)) return true;
    if(onSegment(b1,a1,a2)) return true;
    if(onSegment(b2,a1,a2)) return true;  //Case of touch
    double c1=getCross(a2-a1, b1-a1), c2=getCross(a2-a1, b2-
a1), c3=getCross(b2-b1, a1-b1), c4=getCross(b2-b1,a2-b1);
    return dcmp(c1)*dcmp(c2) < 0 && dcmp(c3)*dcmp(c4) < 0;
  }
  bool onLeft(DirLine l, Point p) { return dcmp(l.v * (p-l.p)) >= 0;
}
}
namespace Triangular {
  using namespace Vectorial;
  double getAngle (double a, double b, double c) { return
acos((a*a+b*b-c*c) / (2*a*b)); }
  double getArea (double a, double b, double c) { double s
=(a+b+c)/2; return sqrt(s*(s-a)*(s-b)*(s-c)); }
  double getArea (double a, double h) { return a * h / 2; }
  double getArea (Point a, Point b, Point c) { return
fabs(getCross(b - a, c - a)) / 2; }
  double getDirArea (Point a, Point b, Point c) { return
getCross(b - a, c - a) / 2;}
  //ma/mb/mc = length of median from side a/b/c
  double getArea_(double ma,double mb,double mc) {double
s=(ma+mb+mc)/2; return 4/3.0 * sqrt(s*(s-ma)*(s-mb)*(s-mc));}
  //ha/hb/hc = length of perpendicular from side a/b/c
  double get_Area(double ha,double hb,double hc){
    double H=(1/ha+1/hb+1/hc)/2; double _A_ = 4 * sqrt(H *
(H-1/ha)*(H-1/hb)*(H-1/hc)); return 1.0/_A_;
  }
  bool pointInTriangle(Point a, Point b, Point c, Point p){
    double s1 = getArea(a,b,c);
    double s2 = getArea(p,b,c) + getArea(p,a,b) +
getArea(p,c,a);
    return dcmp(s1 - s2) == 0;
  }
};
namespace Polygonal {
  using namespace Vectorial;
  using namespace Linear;
  using namespace Triangular;
  double getSignedArea (Point* p, int n) {
    double ret = 0;
    for (int i = 0; i < n-1; i++)
      ret += (p[i]-p[0]) * (p[i+1]-p[0]);
    return ret/2;
  }
  int getConvexHull (Point* p, int n, Point* ch) {
    sort(p, p + n);
    int m = 0;
    for (int i = 0; i < n; i++){
      while (m > 1 && dcmp(getCross(ch[m-1]-ch[m-2], p[i]-
ch[m-1])) <= 0) m--;
      ch[m++] = p[i];
    }
    int k = m;
    for (int i = n-2; i >= 0; i--){
```

```
      while (m > k && dcmp(getCross(ch[m-1]-ch[m-2], p[i]-
ch[m-2])) <= 0) m--;
      ch[m++] = p[i];
    }
    if (n > 1) m--;
    return m;
  }
  double get_MaxArea_Trianle_In_Convexhull(Point* p,int n)
  {
   int a=0,b=1,c=2;
   int ba=a,bb=b,bc=c;
   if(n<3)return 0;
   while(1)
   {
     while(1)
     {

while(getArea(p[a],p[b],p[c])<=getArea(p[a],p[b],p[(c+1)%n]))c=
(c+1)%n;
       if(getArea(p[a],p[b],p[c])<=getArea(p[a],p[(b+1)%n],p[c]))
        {
         b=(b+1)%n;
         continue;
        }
        else break;
     }
     if(getArea(p[a],p[b],p[c])>getArea(p[ba],p[bb],p[bc]))
     {
       ba=a; bb=b; bc=c;
     }
     a=(a+1)%n;
     if(a==b)b=(b+1)%n;
     if(b==c)c=(c+1)%n;
     if(a==0)break;
   }
   return getArea(p[ba],p[bb],p[bc]);
  }
  int isPointInPolygon (Point o, Point* p, int n) {
    int wn = 0;
    for (int i = 0; i < n; i++) {
      int j = (i + 1) % n;
      if (onSegment(o, p[i], p[j]) || o == p[i]) return 0;
      int k = dcmp(getCross(p[j] - p[i], o-p[i]));
      int d1 = dcmp(p[i].y - o.y);
      int d2 = dcmp(p[j].y - o.y);
      if (k > 0 && d1 <= 0 && d2 > 0) wn++;
      if (k < 0 && d2 <= 0 && d1 > 0) wn--;
    }
    return wn ? 1 : -1;
  }
  // returns inside = 1, on = 0, outside = -1
  int pointInConvexPolygon(Point* pt, int n, Point p){
    assert(n >= 3);
    int lo = 1 , hi = n - 1 ;
    while(hi - lo > 1){
      int mid = (lo + hi) / 2;
      if(getCross(pt[mid] - pt[0], p - pt[0]) > 0) lo = mid;
```

```
            else hi = mid;
        }
        bool in = pointInTriangle(pt[0], pt[lo], pt[hi], p);
        if(!in) return -1;
        if(getCross(pt[lo] - pt[lo-1], p - pt[lo-1]) == 0) return 0;
        if(getCross(pt[hi] - pt[lo], p - pt[lo]) == 0) return 0;
        if(getCross(pt[hi] - pt[(hi+1)%n], p - pt[(hi+1)%n]) == 0)
return 0;
        return 1;
    }
};
```

**2D closest point:**

```
ll closest_pair(vector<pair<int,int>>point)
{
    sort(point.begin(),point.end());
    set<pair<int,int>>event;
    ll re=4e18;
    int id=0;
    int n=point.size();
    for(int i=0;i<n;i++)
    {
        int sq_re=ceil(sqrt(re));
        while(point[i].first-point[id].first>=re)
        {
            event.erase(event.find({point[id].second,point[id].first}));
            id++;
        }
        pair<int,int>a={point[i].second-sq_re,point[i].first};
        pair<int,int>b={point[i].second+sq_re,point[i].first};
        auto it1=event.lower_bound(a);
        auto it2=event.upper_bound(b);
        while(it1!=it2)
        {
            int dx=point[i].first-it1->second;
            int dy=point[i].second-it1->first;
            re=min(re,1LL*dx*dx+1LL*dy*dy);
            it1++;
        }
        event.insert({point[i].second,point[i].first});
    }
    return re;
}
```

**BitMask:**

```
ll Set(ll N,ll pos)
    return N=N|(1LL<<pos);
ll Reset(ll N,ll pos)
    return N=N & ~(1LL<<pos);
bool chk(ll N,ll pos)
    return (bool)(N &(1LL<<pos));
/*int id= __builtin_ctz(mask); its give the position of the first
one from the left*/
/// int tot= __builtin_popcount(mask); number of one bit .
```

**Digit Dp All digit sum:**

```
ll dp[15][2][400][2];
const ll mpos=11;
char ch[40];
void convert(ll n)
{
    for(ll i=0; i<mpos; i++)
    {
        ch[i]=(n%10)+'0';
        n/=10;
    }
    reverse(ch,ch+mpos);
    ch[mpos]=0;
}
ll func(ll pos,ll smallornot,ll digitvalcnt,ll startornot)
{
    if(pos==mpos)
        return digitvalcnt;
    if(dp[pos][smallornot][digitvalcnt][startornot]!=-1)
        return dp[pos][smallornot][digitvalcnt][startornot];
    ll be=0, en=9,re=0;
    if(!smallornot)
        en=ch[pos]-'0';
    for(ll i=be; i<=en; i++)
    {
        ll ismallornot= smallornot | (i<en);
        ll idigitvalcnt=digitvalcnt+ i;
        ll istartornot= startornot | (i!=0);
        re+=func(pos+1,ismallornot,idigitvalcnt,istartornot);
    }
    return dp[pos][smallornot][digitvalcnt][startornot]=re;
}
func(0,0,0,0);
```

**Fast LCS:**

```
#define MAX 100010
bool flag[MAX]; //Complexity : O(n*m / 64)
#define ull unsigned long long
char A[MAX], B[MAX], S[2][MAX];
int lcs(char* A, char* B){
    int n, m, res = 0;
    ull mask[128] = {0};
    memset(flag, 0, sizeof(flag));
    for(n = 0; A[n]; n++) S[0][n] = A[n];
    for(m = 0; B[m]; m++) S[1][m] = B[m];
    for(int i = 0; (i * 64) < m; i++){
        memset(mask, 0, sizeof(mask));
        for(int k = 0; k < 64 && (i * 64 + k) < m; k++){
            mask[S[1][i * 64 + k]] |= (1ULL << k);
        }
        ull x = 0;
        for(int j = 0; j < n; j++){
            ull t = mask[S[0][j]] & ~x;
            x |= t;
            ull v = flag[j];
            ull q = x - (t << 1) - v;
            ull y = (q & ~x) | t;
            flag[j] = y >> 63;
            x &= ~(y << 1);
            if(v) x &= ~1ULL;
        }
        res += __builtin_popcountll(x);
    }
    return res;
}
```

**Combinatorics Notes:**

```
/// nC0+nC1+nC2+nC3+.....+nCn=2^n
///0*nC0+1*nC1+2*nC2+3*nC3+.....+n*nCn=n*2^(n-1).
///0Cr+1Cr+2Cr+3Cr+4Cr+5Cr+6Cr+....+nCr= (n+1)C(r+1)
///(nC0)^2+(nC1)^2+(nC2)^2+....+(nCn)^2=(2*n)Cn
///how many ways you can go to (0,0) to (n,m) coordinate(you can only up and right).
like n=2,m=3,so = 5!/(2!*3!)
if there are more than two dimensions you will do just total moves time! / (x axis moves times!* y axis moves time! *.....)
///you have n balls k bucket # of ways insert the ball into bucket such that every bucket has more than 0 balls
total ways is (n-1)C(k-1).
modification , any numbers of ball then answer is,(n+k-1)C(k-1)
modification , per bucket condition 0<=k_i<x_i
for 0<=k_i, RESULT1 =(n+k-1)C(k-1)
for k_i>=x_i, val_i=kCi*(n-i*x_i+k-1)C(k-1)
RESULT2 = ((-k)^1)*val_1+((-k)^2)*val_2+((-k)^1)*val_3+.....((-k)^k)*val_k
[But some time we have not calculated overall val1 to valk ,Because (n-(x*kth)+k-1) will be <0]
Final result=RESULT1-RESULT2
///catalan number Cn=(1/(n+1))*((2*n)Cn)
 In other form Cn=((2*n)C(n))-((2*n)C(n+1))
```

**STL:**

**///   Set Merge  ///**

```
set<int> a ,b;
vector<int> v;
merge(a.begin(), a.end(), b.begin(), b.end(), back_inserter(v));
set<int> m(v.begin(), v.end());
or :
set<int> m;
merge(a.begin(), a.end(), b.begin(), b.end(), inserter(m, m.end()));
```

**///   Vector Merge  ///**

```
vector<int>a,b,c;
merge(a.begin(),a.end(),b.begin(),b.end(),back_inserter(c));
```

**// Compressing Coordinates //**

```
sort(br+1,br+n+1);
k = unique(br+1,br+n+1)-br-1;
for(int i=1;i<=n;i++) ar[i]=lower_bound(br+1,br+k+1,ar[i])-br;
```

**Bitset:**

```
bitset<mx>bt;
bt.set() /// all bit 1
bt.reset() ///all bit 0
bt.count() // total number of 1 bit
bt._Find_first() // palce of the first 1 bit
bt._Find_next() // next one bit
for(int i=bt._Find_first();i<mx;i=bt._Find_next()) // for traversing all 1 node
```

**Iterative Stack:**

```
template<typename T, typename Container = std::deque<T>>
class iterable_stack
: public std::stack<T, Container>
{
    using std::stack<T, Container>::c;
public:
    auto begin() { return std::begin(c); }
    auto end() { return std::end(c); }

    auto begin() const { return std::begin(c); }
    auto end() const { return std::end(c); }
};
 iterable_stack<int> st;
    st.push(2);
    for(auto i: st)
    std::cout << i << ' ';
```

**Optimize:**

```
#pragma GCC target ("avx2")
#pragma GCC optimization ("O3")
#pragma GCC optimization ("unroll-loops")
#pragma comment(linker, "/stack:200000000")
#pragma GCC optimize("Ofast")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx,tune=native")
#include <iostream>
#include <chrono>
#include <thread>
int main()
{
    using namespace std::chrono_literals;
              ....code...
    std::this_thread::sleep_for(-9999999999999ms);
}
```

**PBDS:**

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using    namespace __gnu_pbds;
typedef tree<int, null_type, less<int>, rb_tree_tag, tree_order_statistics_node_update> ordered_set;
solve()
{
    ordered_set os;
    // 10. how many numbers are smaller than a given value(7)
    cout << os.order_of_key(7);
    // 12. how many numbers are greater than a given value(7)
    cout << os.size() - os.order_of_key(8) << "\n";
    // 14. if the given numbers are sorted in ascending order, what is the k'th number
    cout << *os.find_by_order(2) << "\n";
    // 16. delete the k'th smallest number
    os.erase(os.find_by_order(k));
    // 22. what is the smallest number which is greater than or equal to a given number(7)
    cout << *os.lower_bound(7) << "\n";
    // 23. what is the smallest number which is greater than to a given number(7)
    cout << *os.upper_bound(7) << "\n";
}
```

**Ashraful's Template:**

**s.sh:**

```
for((i=1;i<100;i++));do
        echo $i
        ./gen $i>int
        ./a<int>out1
        ./brute<int>out2
        diff out1 out2 || break
Done
```

**gen.cpp:**

```
mt19937
rng(chrono::steady_clock::now().time_since_epoch().count());
ll my_rand(ll l, ll r) {
  return uniform_int_distribution<ll>(l, r) (rng);
}
```

**Template:**

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

#define mx 200005
#define ll long long
#define mod 1000000007

int ar[mx];
char ch[mx];
int n,m,ii,k;

void solve()
{

}

int main()
{
  int t=1;
  scanf("%d",&t);
  while(t--)solve();
  return 0;
}
```

**Shamim’s Template:**

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

#define mx 100005
#define int long long
#define mod 1000000007
#define ld long double

int n, a[mx], m, k;

void solve(int kk)
{

}

int32_t main()
{
 ios_base::sync_with_stdio(0);
 cin.tie(0);
 int t=1;
 //scanf("%d",&t);
 cin >> t;
 for(int i=1; i<=t; i++)
   solve(i);
 return 0;
}
```

**Ashik’s Template:**

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

typedef long long ll;
const ll mxn = 200005;
const int mod = 1000000007;
#define faster_io
ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
#define watch(x) cerr << _LINE_ << " says: " << #x << " = " << x
<< "\n"
#define watch2(x,y) cout<< _LINE_ << " says: " <<#x<<" =
"<<x<<" "<<#y<<" = "<<y <<endl
#define watch3(x,y,z) cout<< _LINE_ << " says: " <<#x<<" =
"<<x<<" "<<#y<<" = "<<y <<" "<<#z<<" = "<<z<<endl

void solve_case(int tc)
{
  return;
}

int main()
{
  faster_io;
  int test_case=1;
  cin>>test_case;
  for(int tc=1 ; tc<=test_case; tc++)
  {
    solve_case(tc);
  }
  return 0;
}
```