2021

RUET_Bug_Makers

ICPC DHAKA REGIONAL 2020

TEAM NOTEBOOK

Number Theory

Sieve

```
const int MAX = 10000005;
int phi[MAX], dvc[MAX], sig[MAX], mob[MAX];
int least[MAX], lstCnt[MAX], lstSum[MAX];
vector<int> primes;
void RunLinearSieve(int n) {
 n = max(n, 1);
 for(int i = 0; i <= n; i++) least[i] = lstCnt[i] = lstSum[i] = 0;
    primes.clear();
    phi[1] = dvc[1] = sig[1] = mob[1] = 1;
    for(int i = 2; i <= n; i++){
        if(least[i] == 0){
            least[i] = i; lstCnt[i] = 1; lstSum[i] = 1 + i;
            phi[i] = i - 1; dvc[i] = 2; sig[i] = 1 + i; mob[i] = -1;
            primes.push_back(i);
        }
        for(int x : primes){
            if(x > least[i] || i * x > n) break;
            least[i * x] = x;
            if(least[i] == x){
                lstCnt[i * x] = lstCnt[i] + 1;
                lstSum[i * x] = 1 + x * lstSum[i];
                phi[i * x] = phi[i] * x;
                dvc[i * x] = dvc[i] / (lstCnt[i] + 1) * (lstCnt[i * x] + 1);
             sig[i * x] = sig[i] / lstSum[i] * lstSum[i * x];
                mob[i * x] = 0;
            }
            else{
                lstCnt[i * x] = 1;
                lstSum[i * x] = 1 + x;
                phi[i * x] = phi[i] * (x - 1);
                dvc[i * x] = dvc[i] * 2;
                sig[i * x] = sig[i] * (1 + x);
               mob[i * x] = -mob[i]; }}}}
```

Segmented Sieve

```
ll segmented_sieve(ll a, ll b) {
    memset(vis,0,sizeof vis);
    if(b<2) return 0;
    if(a<2) a=2;
    ll xx=sqrt((double)b)+1;
    for(ll i=0; i<prime.size() && prime[i]<=xx; i++) {
        ll j=(a/prime[i]);
        if(a%prime[i]!=0) j++;
        j*=prime[i];
        if(j==prime[i]) j+=prime[i];
        for(; j<=b; j+=prime[i])
            vis[j-a]=1;
    }
    ll cnt=0;
    for(ll i=a; i<=b; i++)
        if(vis[i-a]==0) cnt++;
    return cnt;
}
```

Miller Rabin Primality Test

```
/* Miller Rabin Primality Test for <= 10^18 */
#define ll long long
ll mulmod(ll a, ll b, ll c)
{
 ll x = 0, y = a % c;
 while (b)
    {
        if (b & 1) x = (x + y) % c;
        y = (y << 1) % c;
        b >>= 1;
    }
    return x % c;
}
ll fastPow(ll x, ll n, ll MOD)
{
    ll ret = 1;
    while (n)
    {
        if (n & 1) ret = mulmod(ret, x, MOD);
        x = mulmod(x, x, MOD);
        n >>= 1;
    }
    return ret % MOD;
}
bool isPrime(ll n)
{
    if(n == 2 || n == 3) return true;
    if(n == 1 || !(n & 1)) return false;
    ll d = n - 1;
    int s = 0;
    while (d % 2 == 0)
    {
        s++;
        d /= 2;
    }

    int a[9] = { 2, 3, 5, 7, 11, 13, 17, 19, 23 };
    for(int i = 0; i < 9; i++)
    {
        if(n == a[i]) return true;
        bool comp = fastPow(a[i], d, n) != 1;
        if(comp) for(int j = 0; j < s; j++)
            {
                ll fp = fastPow(a[i], (1LL << (ll)j)*d, n);
                if (fp == n - 1)
                {
                    comp = false;
                    break;
                }
            }
        if(comp) return false;
    }
    return true;
}
```

NOD-SOD

Phi

```
//return euler totient function for 1 int
int phi(int n) {
  int ph = n;
  for (int i = 2; i * i <= n; i++) {
    if (n % i)
      continue;
    while (n % i == 0)
      n /= i;
    ph -= ph / i; // ph * (1 - 1/p)
  }
  if (n > 1) ph -= ph / n;
  return ph;
}
```

Phi Sieve

```
//euler totient function from 1 to N
void phi_sieve(int N) {
  bool mark[N + 1] = {0};
  int phi[N];
  for (int i = 0; i < N; i++)
    phi[i] = i;
  mark[1] = true;
  for (int i = 2; i <= N; i += 2) {
    if (!mark[i]) {
      for (int j = i; j <= N; j += i) {
        mark[j] = true;
        phi[j] = (phi[j] / i) * (i - 1);
      }
    }
  }
}
```

GCD

```
ll gcd(ll a, ll b) { return b ? gcd(b, a % b) : a;}
```

Bigmod

```
ll bigmod(ll a, ll b, ll mod){
    ll res = 1;
    while (b > 0){
        if (b & 1)  res = (res * a) % mod;
           a = (a * a) % mod;
           b >>= 1;
    }
    return res;
}
```

Inverse Modulo

```
ll inverse_mod(ll a, ll b) {
  return 1 < a ? b - inverse_mod(b % a, a) * b / a
: 1;}

ll inv[N]; // inverse modulo pre calculate
void imod() {
  inv[1] = 1;
  for (ll i = 2; i < N; i++)
    inv[i] = (mod - (mod / i) * inv[mod % i]) % mod;
}
```

Extended Euclid

```
typedef pair<ll, ll> pii;
#define x first
#define y second
pii extendedEuclid(ll a, ll b)   // returns x, y for ax
+ by = gcd(a,b)
{
    if(b == 0) return pii(1, 0);
    else {
        pii d = extendedEuclid(b, a % b);
        return pii(d.y, d.x - d.y * (a / b));
    }
}
```

Chinese Remainder Theorem

```
ll CRT_weak(vector<ll>A, vector<ll>B) {
    ll X=0;
    ll N=1;
    ll y,z;
    for(ll i=0; i<B.size(); i++)
        N*=B[i];
    for(ll i=0; i<A.size(); i++) {
        y=N/B[i];
        z=modInv(y,B[i]);
        X+=(A[i]*y*z);
        X%=N;
    }
    return (X+N)%N;
}
```

Combinatorics

nCr

```
ll ncr(ll n, ll r) {
  if (r > n - r)
    r = n - r;
  ll ans = 1;
  for (ll i = 1; i <= r; i++) {
    ans *= n - r + i;
    ans /= i;
  }
  return ans;
}
//new code
const ll maxn=1000005;
ll fact[maxn+5],inv[maxn+5];
ll power(ll a,ll n)
{
    ll res=1;
    while(n)
    {
        if(n%2) res*=a,n--,res%=mod;
        else a*=a,n/=2,a%=mod;
    }
    return res;
}
void pre()
{
    fact[0]=inv[0]=1;
    for(ll i=1;i<maxn;i++) fact[i]=(fact[i-1]*i)%mod;
    inv[maxn-1]=power(fact[maxn-1],(ll)mod-2);
    for(ll i=maxn-2;i>=1;i--)
inv[i]=(inv[i+1]*(i+1))%mod;
}
/*
ll dp[35][35];
ll ncr(ll n, ll r)///n^2
{
    if (r > n)
        return 0LL;
    if (r==0 || r==n)
        return 1LL;
    if(dp[n][r]!=-1)
        return dp[n][r];
    return dp[n][r]= ncr(n-1, r-1) + ncr(n-1, r);
}
*/
signed main()
{
    pre();
    ll t=1,cs=1;
    cin>>t;
    while(t--)
    {
         ll n,k;
         cin>>n>>k;
         ll ans=(fact[n]);/// nlogn
         ll x=(fact[k]*fact[n-k])%mod;
         ans*=power(x,mod-2);
         ans%=mod;
         cout<<"Case "<<cs++<<": ";
         cout<<ans<<el;
        ///nck=fact[n]*inv[k]*inv[n-k];
complexity:O(n+logn)
        // cout<<"Case "<<cs++<<": ";
    }
}cout<<ans<<el;
```

Catalan Numbers

Derangement Number

Sterlin Number

Inclusion Exclusion Principle

```
// find number of multiples of all elements of p in
range [l, r]
ll sum = 0;
for (ll msk=1; msk<(1<<p.size()); ++msk) {
    ll mult = 1, bits = 0;
    for (ll i=0; i<(ll)p.size(); ++i)
        if (msk & (1<<i)) {
            ++bits;
            mult *= p[i];
        }
    ll cur = (r / mult) - ((l-1) / mult);
    if (bits % 2 == 1)
        sum += cur;
    else
        sum -= cur;
}
```

Mobius Function

```
ll mu[mx];
void mobius() {
    for(ll i=1; i<mx; i++) mu[i]=1;
    ll root=sqrt((ll)mx);
    for(ll i=0; i<prime.size() && prime[i]<=root; i++)
{
        ll x=prime[i];
        x=x*x;
        for(ll j=x; j<mx; j+=x)
            mu[j]=0;
    }
    for(ll i=0; i<prime.size(); i++) {
        ll x=prime[i];
        for(ll j=x; j<mx; j+=x)
            mu[j]*=-1;
    }
}
```

Data Structure

Segment_tree with lazy

```
const ll N=200005;
ll arr[N],tree[4*N],lazy[4*N];
void build(ll u,ll b,ll e)
{
    if(b==e)
    {
        tree[u]=arr[b];
        return;
    }
    ll mid=(b+e)/2;
    build(2*u,b,mid);
    build(2*u+1,mid+1,e);
    tree[u]=min(tree[2*u],tree[2*u+1]);
}
ll query(ll u,ll b,ll e ,ll i,ll j)
{
    if(lazy[u]!=0)
    {
        ll dx=lazy[u];
        lazy[u]=0;
        tree[u]+=dx;
        if(b!=e)
        {
            lazy[2*u]+=dx;
            lazy[2*u+1]+=dx;
        }
    }
    if(i>e or j<b) return inf;
    if(b>=i and e<=j) return tree[u];
    ll mid=(b+e)/2;
    ll l=query(2*u,b,mid,i,j);
    ll r=query(2*u+1,mid+1,e,i,j);
    return min(l,r);
}
void update(ll u,ll b,ll e,ll i,ll j,ll x)
{
    if(lazy[u]!=0)
    {
        ll dx=lazy[u];
        lazy[u]=0;
        tree[u]+=dx;
        if(b!=e)
        {
            lazy[2*u]+=dx;
            lazy[2*u+1]+=dx;
        }
    }
    if(i>e or j<b) return;
    if(b>=i and e<=j)
    {
        lazy[u]+=x;
        ll dx=lazy[u];
        lazy[u]=0;
        tree[u]+=dx;
        if(b!=e)
        {
            lazy[2*u]+=dx;
            lazy[2*u+1]+=dx;
        }
        return;
    }
    ll mid=(b+e)/2;
   update(2*u,b,mid,i,j,x);
   update(2*u+1,mid+1,e,i,j,x);
   tree[u]=min(tree[2*u],tree[2*u+1]);
}
```

BIT

```
const int MAXN = 1000005;
ll BIT[2][MAXN];
void update(int cs, int indx, ll val){
    while(indx < MAXN){
        BIT[cs][indx] += val;
        indx += (indx & -indx);
    }
}
ll sum(int cs, int indx){
    ll ans = 0;
    while(indx != 0) {
        ans += BIT[cs][indx];
        indx -= (indx & -indx);
    }
    return ans;
}
void updateRange(int l, int r, ll val){
    update(0,l,val);       update(0,r+1,-val);
    update(1,l,val*(l-1)); update(1,r+1,-val*r);
}
ll sumRange(int indx) {return sum(0,indx)*indx -
sum(1,indx);}
ll QueryRange(int l, int r) {return sumRange(r)-
sumRange(l-1);}
const int LOGN = 20;
int LowerBound(int cs, ll v){
    ll sum = 0;
    int indx = 0;
    for(int i = LOGN; i >= 0; i--){
        int nPos = indx + (1<<i);
     if(nPos < MAXN && sum + BIT[cs][nPos] < v){
            sum += BIT[cs][nPos];
            indx = nPos;
        }
    }
//pos = maximal x such that Sum(x) < v
    return indx + 1; //+1 for LowerBound
}
```

Sparse table

```
ll table[100505][22],arr[100005],n,lg[100005];
void pre()
{
    lop1(i,n) table[i][0]=arr[i];
    for(ll k=1;k<=20;k++)
    {
        for(ll i=1;i+(1<<k)-1<=n;i++)
        {
            table[i][k]=min(table[i][k-1],table[i+(1<<(k-
1))][k-1]);
        }
    }
    ll k=1,cnt=0;
    for(ll i=1;i<=100002;i++)
    {
        k*=2;
        cnt+=1;
        if(k<=i) lg[i]=cnt;
        else k/=2,cnt-=1,lg[i]=cnt;
    }


}
ll query(ll l,ll r)
{
    ll len=r-l+1;
    ll k=lg[len];
    return min(table[l][k],table[r-(1<<k)+1][k]);
}
```

Sqrt+Mo's Algo

```
ll n,block,q;
struct info
{
    ll l,r,i;
};
info qarr[200005];
ll arr[30005],ans[200005],vis[1000005],cnt=0;
bool cmp(info a,info b)
{
    if(a.l/block!=b.l/block) return
(a.l/block)<(b.l/block);

    return a.r<b.r;
}
void add(ll pos)
{
    vis[arr[pos]]++;
    if(vis[arr[pos]]==1) cnt++;
}
void del(ll pos)
{
     vis[arr[pos]]--;
     if(vis[arr[pos]]==0) cnt--;
}
signed main()
{
    fastio;
    ll t=1,cs=1;
   // cin>>t;
    while(t--)
    {
        cin>>n;
        block=sqrt(n);
        for(ll i=0;i<n;i++) cin>>arr[i];
        cin>>q;
        for(ll i=0;i<q;i++)
        {
            cin>>qarr[i].l>>qarr[i].r;
            qarr[i].l--;
            qarr[i].r--;
            qarr[i].i=i;
        }
        sort(qarr,qarr+q,cmp);
        ll ml=0,mr=-1;
        for(ll i=0;i<q;i++)
        {
            ll l=qarr[i].l;
            ll r=qarr[i].r;
            while(ml>l) ml--,add(ml);
            while(mr<r) mr++,add(mr);
            while(ml<l) del(ml),ml++;
            while(mr>r) del(mr),mr--;
            ans[qarr[i].i]=cnt;

        }
        lop0(q) cout<<ans[i]<<el;
```

Graph

Dijkstra

```
//defining node’s other end and cost
struct edge {
 ll v, w;
 bool operator<(const edge &b) const { return w
> b.w; } //boro theke choto
};
const ll N = 1e5 + 6, inf = 1LL << 60;
ll n, m, dis[N], par[N];
```

```
vector<edge> g[N];
bool vis[N];
void dijkstra() {
 lop(n) dis[i + 1] = inf;
 dis[1] = 0;
 priority_queue<edge> q;
 q.push({1, 0});
 while (!q.empty()) {
   auto u = q.top().v;
   q.pop();
   if (vis[u])
     continue;
   vis[u] = 1;
   for (auto z : g[u]) {
     ll v = z.v, w = z.w;
     if (dis[u] + w < dis[v]) {
       dis[v] = dis[u] + w;
       q.push({v, dis[v]});
     }
   }
 }
}
```

Floyed Warshal

```
for(k = 1 ; k <= n ; k ++) //middle man
    for(i = 1 ; i <= n ; i++) //left man
        for(j = 1 ; j <= n ; j++) // r8 man
            if (w[i][j] > w[i][k] + w[k][j])
                w[i][j] = w[i][k]+w[k][j];
```

Bellman Ford

```
//bellman ford with negative cycle print
struct edge {
 ll v, w;
};
const ll N = 3e3 + 6, inf = 1LL << 60;
ll n, m, dis[N], par[N];
vector<edge> g[N];

int bellman_ford() {
 lop(n + 1) dis[i] = inf;
 dis[1] = 0;
 int cy;

 lop(n + 1) {
   cy = -1;
   for (int u = 1; u <= n; u++) {
     for (auto z : g[u]) {
       ll v = z.v, w = z.w;
       if (dis[u] + w < dis[v]) {
         dis[v] = dis[u] + w;
         par[v] = u;
         cy = v; // if(u == n) negative cycle;
       }
     }
   }
 }
 return cy; //cy is a adjacent node or a node of negative cycle
}

int main() {
 cin >> n >> m;
 lop(m) {
   ll u, v, w;
   cin >> u >> v >> w;
   g[u].pb({v, w});
 }

 int x = bellman_ford();
 if (x == -1) {
   //no negative cycle
   return 0;
 }

//x can be not a part of cycle, so if we go through
//path sometimes, x will be a node of cycle
 lop(n) x = par[x];
 vector<int> cycle;
 int i = x;
 while (i != x or cycle.size() <= 1) {
   cycle.pb(i);//retrieving cycle
   i = par[i];
 }
 cycle.pb(i);
 reverse(all(cycle));
 for (int z : cycle)
   cout << z << ' ';
 return 0;
}
```

Kruskal

```
//defining an edge with 2 ends & cost
struct edge {
 int u, v, w;
 bool operator<(const edge &p) const {
   return w < p.w; // sorting non decreasing
 }
};
map<int, int> par; // parrent
vector<edge> e;    // edges with their cost
int findrep(int r) { return (par[r] == r) ? r : par[r] = findrep(par[r]); }
//call this ans = kruskal(n)
int kruskal_mst(int n) {
 sort(e.begin(), e.end());
 for (int i = 1; i <= n; i++)
   par[i] = i; // parenting ownself
 int cnt = 0, s = 0;
 int saiz = e.size();
 for (int i = 0; i < saiz; i++) {
   int u = findrep(e[i].u);
   int v = findrep(e[i].v);
   if (u != v) {
     par[u] = v; // union
     cnt++;       // number of edges of latest graph
     s += e[i].w; // coast of latest graph
     if (cnt == n - 1) // full graph complete
       break;
   }
 }
 return s;
}
```

Topological Sort

```
int n; // number of vertices
vector<vector<int>> adj; // adjacency list of graph
vector<bool> visited;
vector<int> ans;

void dfs(int v) {
    visited[v] = true;
    for (int u : adj[v]) {
        if (!visited[u])
            dfs(u);
    }
    ans.push_back(v);
}

void topological_sort() {
    visited.assign(n, false);
    ans.clear();
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i);
    }
    reverse(ans.begin(), ans.end());
}
```

Articulation Point

```
int n; // number of nodes
vector<vector<int>> adj;
vector<bool> visited;
vector<int> tin, low;
int timer;

void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    int children=0;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] >= tin[v] && p!=-1)
                IS_CUTPOINT(v);
            ++children;
        }
    }
    if(p == -1 && children > 1)
        IS_CUTPOINT(v);
}

void find_cutpoints() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs (i);
    }
}
```

Articulation Bridge

```
int n; // number of nodes
vector<vector<int>> adj; // adjacency list of graph

vector<bool> visited;
vector<int> tin, low;
int timer;

void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] > tin[v])
                IS_BRIDGE(v, to);
        }
    }
}

void find_bridges() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i);
    }
}
```

SCC

```
vector<vector<int>> adj, adj_rev;
vector<bool> used;
vector<int> order, component;
void dfs1(int v) {
    used[v] = true;
    for (auto u : adj[v])
        if (!used[u])dfs1(u);
    order.push_back(v);
}
void dfs2(int v) {
    used[v] = true;
    component.push_back(v);
    for (auto u : adj_rev[v])
        if (!used[u])dfs2(u);}
int main() {
    int n;
    for (;;) {
        int a, b;
        // ... read next directed edge (a,b) ...
        adj[a].push_back(b);
        adj_rev[b].push_back(a);
    }
    used.assign(n, false);
    for (int i = 0; i < n; i++)
        if (!used[i])dfs1(i);
    used.assign(n, false);
    reverse(order.begin(), order.end());
    for (auto v : order)
        if (!used[v]) {
            dfs2 (v);
            // ... processing next component ...
            component.clear();}}
```

LCA

```
#define MAX 200010
#define LOG 20

namespace LCA{
  int sum[MAX] ; int
st[MAX],en[MAX],lg[MAX],par[MAX],a[MAX],
  id[MAX],dp[LOG][MAX] ;
  vector <int> weight[MAX] , g[MAX]; int n , r ,
Time , cur ;
  void init(int nodes, int root){
    n = nodes, r = root, lg[0] = lg[1] = 0;
    for(int i = 2; i <= n; i++) lg[i] = lg[i >> 1] + 1;
    for(int i=0;i<= n;i++) g[i].clear(),
weight[i].clear();
  }
  void addEdge(int u, int v, int w){
    g[u].push_back(v), weight[u].push_back(w);
    g[v].push_back(u), weight[v].push_back(w);
  }
  int lca(int u, int v){
    if( en[u] > en[v] )swap(u,v) ;
    if( st[v] <= st[u] && en[u] <= en[v] ) return v ;
    int l = lg[id[v] - id[u] + 1] ;
    int p1 = id[u] , p2 = id[v] - (1<<l) + 1 ;
    if(sum[dp[l][p1]]<sum[dp[l][p2]]) return
par[dp[l][p1]] ;
    else return par[ dp[l][p2] ] ;
  }
  int dis( int u ,int v ){
    int l = lca(u,v) ;
    return (sum[u] + sum[v] - ( sum[l]<<1LL )) ;
  }
  void dfs(int u, int p , int curSum){
    st[u] = ++Time ; par[u] = p ; sum[u] = curSum ;
    for(int i=0 ; i<g[u].size() ; i++){
      if( g[u][i]==p ) continue ;
      dfs( g[u][i] ,u,curSum+weight[u][i]) ;
    }
    en[u] = ++Time ; a[++cur] = u ; id[u] = cur ;
  }
  void build(){
    cur = Time = 0 ; dfs( r , r , 0 );
    for(int i=1 ; i<=n ; i++) dp[0][i] = a[i] ;
    for(int l=0 ; l<LOG-1 ; l++) {
      for(int i=1 ; i<=n ; i++) {
        dp[l+1][i] = dp[l][i] ;
        if( (1<<l)+i <= n && sum[dp[l][i+(1<<l)]] <
           sum[dp[l][i]]) dp[l+1][i] = dp[l][ i+(1<<l)] ;
      }
    }
  }
}
```

String

Hash (double)

```
const ll MAX_N = 1e6 + 10, mod = 2e9 + 63,
base1 = 1e9 + 21, base2 = 1e9 + 181;
ll pw1[MAX_N], pw2[MAX_N];

void pw_calc() {
    pw1[0] = pw2[0] = 1;
    for (int i = 1; i < MAX_N; i++) {
           pw1[i] = (pw1[i - 1] * base1) % mod;
           pw2[i] = (pw2[i - 1] * base2) % mod;
    }
}
struct Hash {
    char s[MAX_N];
    int slen = strlen(s);
    ll h1[MAX_N], h2[MAX_N];
    void init() {
        h1[0] = h2[0] = 0;
        for (int i = 1; i <= slen; i++) {
            h1[i] = (h1[i - 1] * base1 + s[i - 1]) % mod;
            h2[i] = (h2[i - 1] * base2 + s[i - 1]) % mod;
        }
    }
    inline ll hashVal(int l, int r) {
        ll hsh1 = (h1[r] - h1[l - 1] * pw1[r - l + 1]) %
mod;
        if (hsh1 < 0) hsh1 += mod;
        ll hsh2 = (h2[r] - h2[l - 1] * pw2[r - l + 1]) %
mod;
```

```
        if (hsh2 < 0) hsh2 += mod;
        return (hsh1 << 32) | hsh2;
    }
} fw;
/* call pw_calc() for calculating powers less than MAX_N
 * fw.init() will calculate the double hashes
 * fw.hashVal(l,r) will return [l,,r] merged double hash value
*/
```

Trie

```
const int MAX = 1e5 + 3, T = 10;
int node[MAX][T], nnode,  word[MAX];
char ch = '0'; //  ‘0’ for int, ‘a’, ‘A’ for char
void reset(int n) {
// before first insert make  reset(0), node = 0
        for (int i = 0; i < T; i++)
                node[n][i] = -1;
}
void Insert(string s) {
    int n = s.size(), nw = 0;
    for (int i = 0; i < n; i++)
    {
        if (node[nw][s[i] - ch] == -1)
        {
                node[nw][s[i] - ch] = ++nnode;
                reset(nnode);
        }
        nw = node[nw][s[i] - ch];
    }
    word[nw]++; //end of a word
}
//find maximum subarray xor sum
int doxor(int s) {
  int nw = 0, t = 0;
  for (int i = 31; i >= 0; i--) {
    bool p = (1 << i) & s;
    if (node[nw][p ^ 1] != -1) {
      t |= 1 << i;
      nw = node[nw][p ^ 1];
    } else
      nw = node[nw][p];
  }
  return t;
}
//minimum subarray xor sum
int doxor2(int s) {
  int nw = 0, t = 0;
  for (int i = 31; i >= 0; i--) {
    bool p = (1 << i) & s;
    if (node[nw][p] != -1)
      nw = node[nw][p];
    else {
      t |= 1 << i;
      nw = node[nw][p ^ 1];
    }
  }
  return t;
}
//at first insert(0), then calculate xor before inserting each element of the array
//calculate number of subarray having xor>=k
int doxor(int s) {
  int nw = 0, t = 0;
  for (int i = 31; i >= 0; i--) {
    bool p = (1 << i) & s;
    bool q = (1 << i) & k;
    if (!q) {
      t += (node[nw][p ^ 1] != -1 ? word[node[nw][p ^ 1]] : 0);
      nw = node[nw][p];
    } else
      nw = node[nw][p ^ 1];
    if (nw == -1)
      break;
  }
  if (nw != -1)
    t += word[nw];
  return t;
}
//insert(0), sum returned value, insert prefix xor
```

KMP

```
const ll MAX_N = 1e5 + 10;

char s[MAX_N], pat[MAX_N]; // 1-indexed
ll lps[MAX_N]; // lps[i] = longest proper prefix-suffix in i length's prefix

void gen_lps(ll plen) {
 ll now;
 lps[0] = lps[1] = now = 0;
 for (ll i = 2; i <= plen; i++) {
   while (now != 0 && pat[now + 1] != pat[i])
     now = lps[now];
   if (pat[now + 1] == pat[i])
     lps[i] = ++now;
   else
     lps[i] = now = 0;
 }
}

ll KMP(ll slen, ll plen) {
 ll now = 0;
 for (ll i = 1; i <= slen; i++) {
   while (now != 0 && pat[now + 1] != s[i])
     now = lps[now];
   if (pat[now + 1] == s[i])
     ++now;
   else
     now = 0;
   // now is the length of the longest prefix of pat, which
   // ends as a substring of s in index i.
   if (now == plen)
     return 1;
 }
 return 0;
}

// slen = length of s, plen = length of pat
// call gen_lps(plen); to generate LPS (failure) array
// call KMP(slen, plen) to find pat in s
```

Z-Algo

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

Suffix Array (nlogn)

```
#define MAX_N 1000020
int n, t;
char s[500099];
int SA[MAX_N], LCP[MAX_N];
int RA[MAX_N], tempRA[MAX_N];
int tempSA[MAX_N];
int c[MAX_N];
int Phi[MAX_N], PLCP[MAX_N];
// second approach: O(n log n)
// the input string, up to 100K characters
// the length of input string
// rank array and temporary rank array
// suffix array and temporary suffix array
// for counting/radix sort
void countingSort(int k) {    // O(n)
    int i, sum, maxi = max(300, n);
    // up to 255 ASCII chars or length of n
    memset(c, 0, sizeof c);
    // clear frequency table
    for (i = 0; i < n; i++)
    // count the frequency of each integer rank
    c[i + k < n ? RA[i + k] : 0]++;
    for (i = sum = 0; i < maxi; i++) {
        int t = c[i]; c[i] = sum; sum += t;
    }
    for (i = 0; i < n; i++)
    // shuffle the suffix array if necessary
```

```
   tempSA[c[SA[i] + k < n ? RA[SA[i] + k] : 0]++] = SA[i];
   for (i = 0; i < n; i++)
   // update the suffix array SA
   SA[i] = tempSA[i];
}

void buildSA() {
   int i, k, r;
   for (i = 0; i < n; i++) RA[i] = s[i];
   // initial rankings
   for (i = 0; i < n; i++) SA[i] = i;
   // initial SA: {0, 1, 2, ..., n-1}
   for (k = 1; k < n; k <<= 1) {
      // repeat sorting process log n times
      countingSort(k); // actually radix sort: sort based on the second item
      countingSort(0);
      // then (stable) sort based on the first item
      tempRA[SA[0]] = r = 0;
      // re-ranking; start from rank r = 0
      for (i = 1; i < n; i++)
         // compare adjacent suffixes
         tempRA[SA[i]] = // if same pair => same rank r; otherwise, increase r
            (RA[SA[i]] == RA[SA[i - 1]] && RA[SA[i] + k] == RA[SA[i - 1] + k]) ? r : ++r;
      for (i = 0; i < n; i++)
         // update the rank array RA
         RA[i] = tempRA[i];
      if (RA[SA[n - 1]] == n - 1) break;
      // nice optimization trick
   }
}
```

<u>Special + Geo</u>

FASTIO

```
//non negative numbers
inline int getint(){
   char ch = getchar(); int x = 0;
   while(ch < '0' || ch > '9') ch = getchar();
   while(ch >= '0' && ch <= '9'){
      x = (x<<3)+(x<<1)+ch-'0';
      ch = getchar();
   }
   return x;
}
//all int
inline int readInt () {
        bool minus = 0; int x = 0;
        char ch = getchar();
        while (ch != '-' and (ch < '0' or ch > '9'))
                ch = getchar();
        if (ch == '-') minus = 1; else x = ch-'0';
        while (1) {
                ch = getchar();
                if (ch < '0' || ch > '9') break;
                x = (x<<1)+(x<<3) + ch-48;
        }
        return minus ? -x : x;
}
```

Debug

```
//create test case generator, brute force code, main code and bash script
//bash script
for((i = 1; ; ++i)); do
   echo $i
   ./gen > int
   diff -w <(./main < int) <(./brute < int) || break
done
//save this as s.sh and run by ./s.sh
//random number generator
mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
// return a random number in [l, r] range
inline ll gen_random(ll l, ll r) {
   return uniform_int_distribution<ll>(l, r)(rng);
}
inline double gen_random_real(double l, double r) {
   return uniform_real_distribution<double>(l, r)(rng);
}
```

Custom Hash for Unordered_map

```
// To prevent collision in unordered_map
#include <bits/stdc++.h>
using namespace std;

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

    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM =
chrono::steady_clock::now().time_since_epoch().
count();
        return splitmix64(x + FIXED_RANDOM);
    }
};

/// Declaration: unordered_map <int, int,
custom_hash> numbers;
/// Usage: same as normal unordered_map
/// Ex: numbers[5] = 2;


/// *** To use gp_hash_table (faster than
unordered_map) ****  ///

/// Add these extra two lines:
/// #include <ext/pb_ds/assoc_container.hpp>
/// using namespace __gnu_pbds;

/// Declaration: gp_hash_table<int, int,
custom_hash> numbers;
/// Usage: Same as unordered_map
```

*****BASH SCRIPT**

```
@echo off
gen >in
sol<in >out
brute <in >ok
fc out ok
if ErrorLevel 1 exit /b
run
```

digit dp one time memset

```
///How many zeros in the numbers' digits. Range
of the numbers is (l, r)

#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define pb push_back
ll dp[2][2][12][12];
vector <ll> num;

ll solve(ll isStart, ll isSmall, ll pos, ll val) {
    if(pos == 0)
        return val;
    ll &ret = dp[isStart][isSmall][pos][val];
    if(ret != -1 && isSmall)
        return ret;
    ll lim, pos2 = num.size() - pos;
    if(isSmall)
        lim = 9;
    else
        lim = num[pos2];
    ll rt = 0;
    if(!isStart) {
        for(ll i = 0; i <= lim; i++)
            rt += solve(0, isSmall | i < num[pos2], pos
- 1, (i == 0) + val);
    }
    else {
        for(ll i = 1; i <= lim; i++)
            rt += solve(0, isSmall | i < num[pos2], pos
- 1, val);
        rt += solve(1, 1, pos - 1, 0);
    }
    return ret = rt;
}
ll calc(ll n) {
    if(n < 0)
        return 0;
    if(n < 10)
        return 1;
    ll tmp = n;
    num.clear();
    while(tmp) {
        num.pb(tmp % 10);
        tmp /= 10;
    }
    reverse(num.begin(), num.end());

    return solve(1, 0, num.size(), 0) + 1;    /// + 1 is for the number "0". We are not calculating this number in solve function.
}

int main()

{
    ll t, caseno = 0;
    memset(dp, -1, sizeof(dp));
    cin >> t;
    while(t--) {
        ll l, r;
        scanf("%lld %lld", &l, &r);
        ll ans = calc(r);
        ans -= calc(l - 1);
        printf("Case %lld: %lld\n", ++caseno, ans);
    }
    return 0;
}
```

FFT

```
/***
* Multiply (7x^2 + 8x^1 + 9x^0) with (6x^1 + 5x^0)
* ans = 42x^3 + 83x^2 + 94x^1 + 45x^0
* A = {9, 8, 7}
* B = {5, 6}
* V = multiply(A,B)
* V = {45, 94, 83, 42}
***/
/*** Tricks
* Use vector < bool > if you need to check only the status of the sum
* Use bigmod if the power is over same polynomial && power is big
* Use long double if you need more precision
* Use long long for overflow
***/
typedef vector<int> vi;
const double PI = 2.0 * acos(0.0);
using cd = complex<double>;
void fft(vector<cd> &a, bool invert = 0) {
 int n = a.size();
 for (int i = 1, j = 0; i < n; i++) {
   int bit = n >> 1;
   for (; j & bit; bit >>= 1)
     j ^= bit;
   j ^= bit;

   if (i < j)
     swap(a[i], a[j]);
 }
 for (int len = 2; len <= n; len <<= 1) {
   double ang = 2 * PI / len * (invert ? -1 : 1);
   cd wlen(cos(ang), sin(ang));
   for (int i = 0; i < n; i += len) {
     cd w(1);
     for (int j = 0; j < len / 2; j++) {
       cd u = a[i + j], v = a[i + j + len / 2] * w;
       a[i + j] = u + v;
       a[i + j + len / 2] = u - v;
       w *= wlen;
     }
   }
 }
 if (invert) {
   for (cd &x : a)
     x /= n;
 }
}

void ifft(vector<cd> &p) { fft(p, 1); }

vi multiply(vi const &a, vi const &b) {
 vector<cd> fa(a.begin(), a.end()), fb(b.begin(),
b.end());
 int n = 1;
 while (n < a.size() + b.size())
   n <<= 1;
 fa.resize(n);
 fb.resize(n);

 fft(fa);
 fft(fb);
 for (int i = 0; i < n; i++)
   fa[i] *= fb[i];
 ifft(fa);

 vi result(n);
 for (int i = 0; i < n; i++)
   result[i] = round(fa[i].real());
 return result;
}
```

Line Segment Intersection

```
// A C++ program to check if two given line
segments intersect
struct Point
{
int x;
int y;
};

// Given three colinear points p, q, r, the function
checks if
// point q lies on line segment 'pr'
bool onSegment(Point p, Point q, Point r)
{
if (q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) &&
    q.y <= max(p.y, r.y) && q.y >= min(p.y, r.y))
   return true;

return false;
}

// To find orientation of ordered triplet (p, q, r).
// The function returns following values
// 0 --> p, q and r are colinear
// 1 --> Clockwise
// 2 --> Counterclockwise
int orientation(Point p, Point q, Point r)
{

int val = (q.y - p.y) * (r.x - q.x) -
          (q.x - p.x) * (r.y - q.y);

if (val == 0) return 0;  // colinear
return (val > 0)? 1: 2; // clock or counterclock
wise
}
// The main function that returns true if line
segment 'p1q1'
// and 'p2q2' intersect.
bool doIntersect(Point p1, Point q1, Point p2,
Point q2)
{
// Find the four orientations needed for general
and
// special cases
int o1 = orientation(p1, q1, p2);
int o2 = orientation(p1, q1, q2);
int o3 = orientation(p2, q2, p1);
int o4 = orientation(p2, q2, q1);

// General case
if (o1 != o2 && o3 != o4)
    return true;

// Special Cases
// p1, q1 and p2 are colinear and p2 lies on
segment p1q1
if (o1 == 0 && onSegment(p1, p2, q1)) return true;

// p1, q1 and q2 are colinear and q2 lies on segment p1q1
if (o2 == 0 && onSegment(p1, q2, q1)) return true;

// p2, q2 and p1 are colinear and p1 lies on segment p2q2
if (o3 == 0 && onSegment(p2, p1, q2)) return true;

  // p2, q2 and q1 are colinear and q1 lies on segment p2q2
if (o4 == 0 && onSegment(p2, q1, q2)) return true;

return false; // Doesn't fall in any of the above cases
}
```

Convex Hull

```
#define ll long long
#define siz 100009

struct point {
    ll x, y;
};

point p[siz], hull[2 * siz];
ll sz = 0;
bool cmp(point a, point b) {
          if(a.x != b.x)
                    return a.x < b.x;
          return a.y < b.y;
}
ll cross (point a, point b, point c) {
return (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x - a.x);
}
void ConvexHull(ll n) {
sz = 0;
sort(p, p + n, cmp);

/// Building upper hull
for(ll i = 0; i < n; i++) {
     while (sz > 1 && cross(hull[sz - 2], hull[sz - 1], p[i]) <= 0) --sz;   /// use < 0 for taking co-linear points
         hull[sz++] = p[i];
}

/// Building lower hull
for(int i = n - 2, j = sz + 1; i >= 0; i--) {
     while (sz >= j && cross(hull[sz - 2], hull[sz - 1], p[i]) <= 0) --sz;  /// use < 0 for taking co-linear points
         hull[sz++] = p[i];
}
/// last point is same as first point. so, sz--
sz--;
}
```

Minimum Enclosing Circle

```
/// Minimum radius of circle to enclose all points of a polygon
/// Converting polygon to convex hull before any calculation reduces complexity
#include <bits/stdc++.h>
using namespace std;

#define ll long long
#define iter 30000   /// The more the number of iteration, the more accurate the result is

struct point {
ll x, y;
} p[100005];

ll n;
double X, Y, d, e;

double dist(double a, double b) {
	return a*a + b*b;
}

int main() {
	scanf("%lld", &n);
	for (ll i = 0; i < n; i++) {
		scanf("%lld %lld", &p[i].x, &p[i].y);
		X += p[i].x; Y += p[i].y;
	}
	X /= n; Y /= n;   /// Average center

	double mv = 0.1;
	for (ll i = 0; i < iter; i++) {
		ll f = 0;
		d = dist(X - p[0].x, Y - p[0].y);
		for (ll j = 1; j < n; j++) {
			e = dist(X - p[j].x, Y - p[j].y);
			if (d < e) { d = e; f = j; }
		}
/// Moving center towards the farthest point slightly
		X += (p[f].x - X) * mv;
		Y += (p[f].y - Y) * mv;
		mv *= 0.999;
	}

	printf("X = %.3f Y = %.3f , radius = %.3f\n", X, Y, sqrt(d));
}
```

SQUFOF

```
#define ll long long

// trival divisor O( n^(1/4) )
// ll divisor = SQUFOF(n);   // a divisor of n
const ll multiplier[] = {1, 3, 5, 7, 11, 13,
			3*5, 3*7, 3*11, 3*13, 5*7, 5*11, 5*13, 7*11, 7*13, 11*13,
			3*5*7, 3*5*11, 3*5*13, 3*7*11, 3*7*13, 3*11*13, 5*7*11, 5*7*13, 5*11*13, 7*11*13,
			3*5*7*11, 3*5*7*13, 3*5*11*13, 3*7*11*13, 5*7*11*13, 3*5*7*11*13};

#define nelems(x) (sizeof(x) / sizeof((x)[0]))
ll SQUFOF( ll N ) {
   ll D, Po, P, Pprev, Q, Qprev, q, b, r, s;
   ll L, B, i;
   s = (ll)(sqrtl(N)+0.5);
   if (s*s == N) return s;
   for (int k = 0; k < nelems(multiplier) && N <= UINT64_MAX/multiplier[k]; k++) {
      D = multiplier[k]*N;
      Po = Pprev = P = sqrtl(D);
      Qprev = 1;
      Q = D - Po*Po;
      L = 2 * sqrtl( 2*s );
      B = 3 * L;
      for (i = 2 ; i < B ; i++) {
       b = (ll)((Po + P)/Q);
       P = b*Q - P;
       q = Q;
       Q = Qprev + b*(Pprev - P);
       r = (ll)(sqrtl(Q)+0.5);
       if (!(i & 1) && r*r == Q) break;
       Qprev = q;
       Pprev = P;
      };
      if (i >= B) continue;
      b = (ll)((Po - P)/r);
      Pprev = P = b*r + P;
      Qprev = r;
      Q = (D - Pprev*Pprev)/Qprev;
      i = 0;
      do {
       b = (ll)((Po + P)/Q);
       Pprev = P;
       P = b*Q - P;
       q = Q;
       Q = Qprev + b*(Pprev - P);
       Qprev = q;
       i++;
      }
```

```
        while (P != Pprev);
        r = __gcd(N, Qprev);
        if (r != 1 && r != N) return r;
    }
    return 0;
}
```

SOS (Sum of Subsets)

```
/// Given a fixed array A of 2^N integers, we need to calculate
/// the function F(x) = Sum of all A[i] such that x&i = i, i.e., i is a subset of x.
/// It means i is the subset bitmask of the bitmask of x.
/// Suboptimal Bruteforce Method O(3^n):
// iterate over all the masks
for (int mask = 0; mask < (1<<n); mask++) {
            F[mask] = A[0];
// iterate over all the subsets of the mask
for(int i = mask; i > 0; i = (i-1) & mask){
      F[mask] += A[i];
}
}

/// Two DP methods O(n*2^n):
/// iterative version
for(int mask = 0; mask < (1<<N); mask++){
      dp[mask][0] = A[mask];  //handle base case separately (leaf states)
       for(int i = 0;i < N; i++){
           if(mask & (1<<i))
               dp[mask][i + 1] = dp[mask][i] + dp[mask^(1<<i)][i];
           else
                dp[mask][i + 1] = dp[mask][i];
        }
        F[mask] = dp[mask][N];
}

/// memory optimized, super easy to code.
for(int i = 0; i<(1<<N); i++)
            F[i] = A[i];
for(int i = 0;i < N; ++i) {
for(int mask = 0; mask < (1<<N); ++mask){
     if(mask & (1<<i))
         F[mask] += F[mask^(1<<i)];
}
}
```

Longest Common Subsequence O(n*m/63)

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

#define ll unsigned long long
#define Set(n, pos) n |= ((ll)1 << (pos))
#define check(n, pos) (n >> (pos)) & (ll)1
const int siz = 50002;

char y[siz], x[siz];
ll M[27][siz / 63 + 1], M2[30][siz / 63 + 1],
L[siz][siz / 63 + 1], tmp[siz / 63 + 1], indx;
ll clr, clr2, c;

int main() {
    register int i, j, ans = 0;
    clr = ((ll)1 << 63) - 1;

    cin >> (y + 1) >> (x + 1);
    ll n = strlen(y + 1), m = strlen(x + 1);

    ll lim = m / 63, last = m % 63;
    clr2 = ((ll)1 << (last + 1)) - 1;
    for(i = 1; i <= m; i++) {
        indx = x[i] - 'a';
        Set(M[indx][i / 63], i % 63);
    }

    for(i = 0; i < 26; i++) {
        for(j = 0; j <= lim; j++) {
            M2[i][j] = ~M[i][j];

            if(j == 0) M2[i][j] &= (clr ^ (ll)1);
            if(j == lim) M2[i][j] &= clr2;
            else M2[i][j] &= clr;
        }
    }

    for(i = 0; i <= lim; i++) {
        L[0][i] = clr;

        if(i == 0) L[0][i] &= (clr ^ (ll)1);
        if(i == lim) L[0][i] &= clr2;
    }

    for(j = 1; j <= n; j++) {
        indx = y[j] - 'a';
        for(i = 0; i <= lim; i++)
            tmp[i] = L[j-1][i] & M[indx][i];

        c = 0;
        for(i = 0; i <= lim; i++) {
            L[j][i] = L[j - 1][i] + tmp[i] + c;
            if(i == lim) {
                if(check(L[j][i], last + 1) == 1) c = 1;
                else c = 0;
                L[j][i] &= clr2;
            }
            else {
                if(check(L[j][i], 63) == 1) c = 1;
                else c = 0;
                L[j][i] &= clr;
            }
        }
        for(i = 0; i <= lim; i++)
            L[j][i] |= L[j - 1][i] & M2[indx][i];
        ans += c;
    }
    cout << ans << endl;
    return 0;
}
```

Extra Note

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>;//*(x.find_by_order(i)),x.order_of_key(k)
/* greater<T> for sorting decreasingly
order_of_key (k) : Number of items strictly smaller than k .
find_by_order(k) : K-th element in a set (counting from zero)*/
```

Stars and Bars

The number of ways to put n identical objects into k labeled boxes is
((n+k-1)choose(n)).....(n+k-1)C(n)

1. Suppose, there are n objects to be placed into k bins, ways= (n-1)C(k-1)
2. Statement of 1no. and empty bins are valid, ways= (n+k-1)C(k-1)

GCD

1. gcd(a, b) = gcd(a, a - b)
2. gcd(F(a), F(b)) = F(gcd(a,b)) [F=fibonacci]

coordinate geometry formula

1. Point Slope Form: y − y_1= m(x − x_1)
2. Slope, m = Δy/Δx = (y2 − y1)/(x2 − x1)
3. Slope from line, m = -(A/B)
4. Angle, tan θ = [(m1 − m2)/ (1 + m1m2)]
5. Distance from a Point to a Line, d = [|Ax0 + By0 + C| / √(A2 + B2)]

circle formula

Area of segment in radian angle : A = (½) × r^2 (θ − Sin θ)

sod nod

1. (a+1)(b+1)(c+1) [Number of divisors, a, b, c are powers of prime number]
2. $\frac{(p^{a+1}-1)}{p-1} . \frac{(q^{b+1}-1)}{q-1}$Here p,q is a prime numbers [Sum of Divisors]

n-th term

1.সমান্তর ধারা: nতম পদ=$a + (n - 1)d$, sum=$\frac{n\{2a + (n-1)d\}}{2}$

2.গুণোত্তর ধারা: nতম পদ=ar^{n-1}, sum=$\frac{a(r^{n} - 1)}{r-1}$

3.Catalan Numbers: 1, 1, 2, 5, 14, 42, 132…..
C_n=(2n)!/(n+1)!n! n>=0

Progression

1. Sum of first n positive number = n*(n+1)/2
2. Sum of first n odd number = n^2
3. Sum of first n even number = n*(n+1)

polygon area, diagonal formula

The sum of interior angles of a polygon with “n” sides =180°*(**n-**2). Number of diagonals of a “n-sided” polygon = [n(n-3)]/2. The measure of interior angles of a regular n-sided polygon = [(n-2)180°]/n. The measure of exterior angles of a regular n-sided polygon = 360°/n

modular arithmetic

1. $a^{\varphi(n)} = 1 \% n$where φ(n) is Euler Totient Function.
2. $a^b \% m = a^{b \% \varphi(m)} \% m$ where *a* and *m* are coprime.

Template of Rifat

```
#include<bits/stdc++.h>

#define ll long long
#define pb push_back
#define ff first
#define ss second

#define yes cout << "YES\n"
#define no     cout << "NO\n"
#define Case(i) cout << "Case " << int(i) << ": "

#define lop(n) for (int i = 0; i < n; i++)
#define lopj(n) for (int j = 0; j < n; j++)
#define all(x)     x.begin(), x.end()
#define sortd(x) sort(x.rbegin(), x.rend())
#define bitcount(x) __builtin_popcount(x)

#define vin vector <int>
#define vll vector <ll>
#define pll pair <ll, ll>
#define pii pair <int, int>
#define vpll vector <pll>
#ifndef ONLINE_JUDGE
#define dbg(x) cout << #x << " = " << x << endl;
#define dbg2(x, y) cout << #x << " = " << x << "\t", dbg(y);
#define dbg3(x, y, z) cout << #x << " = " << x << "\t", dbg2(y, z);
#define ddbg(x) cout << #x << " = [ "; for(auto z : x) cout << z << ' '; cout << "]\n";
#else
#define dbg(x)
#define adbg(x)
#define dbg2(x, y)
#define dbg3(x, y, z)
#define ddbg(x)
#endif
#define sob template < typename T
#define sb2 template < typename T, typename TT
sob > void print(T x) {std::cout << x << '\n';}
sb2 > void print(T x, TT y) {std::cout << x << ' ' << y << '\n';}
sb2 > void print(std::pair <T, TT> x) {std::cout << x.ff << ' ' << x.ss << '\n';}
sob > void print(std::vector <T> v) {for (auto z : v) std::cout << z << ' '; std::cout << '\n';}
sob > void print(T x[], int n) {for(int i = 0; i++ < n;) std::cout << *x++ << (i < n ? ' ':'\n');}

using namespace std;

int main()
{
	ios_base::sync_with_stdio(0); cin.tie(0);
```

```
        int tc;
        cin >> tc;
        while (tc--)
        {

        }
        return 0;
}
/* Infos
~   4 Direction
int dr[] = {1,-1,0,0};
int dc[] = {0,0,1,-1};

~   8 Direction
int dr[] = {1,-1,0,0,1,1,-1,-1};
int dc[] = {0,0,1,-1,1,-1,1,-1};

~   Knight Direction
int dr[] = {1,-1,1,-1,2,2,-2,-2};
int dc[] = {2,2,-2,-2,1,-1,1,-1};

~   Hexagonal Direction
int dr[] = {2,-2,1,1,-1,-1};
int dc[] = {0,0,1,-1,1,-1};

~   bitmask operations
int Set(int n, int pos) { return n = n | (1 << pos); }
int reset(int n, int pos) { return n = n & ~(1 << pos); }
bool check(int n, int pos) { return (bool)(n & (1 << pos)); }
bool isPower2(int x) { return (x && !(x & (x - 1))); }
ll LargestPower2<=x(ll x) { for(int i = 1; i <= x / 2; i *= 2) x = x | (x >> i); return (x + 1) / 2;}
*/
```

Template of Arnab

```
/*
#include"bits/stdc++.h"
using namespace std;
typedef long long ll;
#define vi vector<ll>
#define pb push_back
#define ff first
#define ss second
#define inf 2e18
#define ull unsigned long long
#define pi acos(-1.0)
#define mod 1000000007
#define lop0(n) for(ll i=0;i<n;i++)
#define lop(j,n) for(ll j=0;j<n;j++)
#define lop1(i,n) for(ll i=1;i<=n;i++)
#define all(v) v.begin(),v.end()
#define el '\n'
ll Set(ll N,ll pos){ return N=N | (1LL<<pos); }
ll reset(ll N,ll pos){ return N= N & ~(1LL<<pos); }
bool check(ll N,ll pos){ return (bool)(N & (1LL<<pos)); }
ll dx[] = { 1,0,-1,0 };
ll dy[] = { 0,1,0,-1 };
#define fastio ios_base::sync_with_stdio(false);cin.tie(NULL);cout.tie(NULL)
ll H1[MAX+5],H2[MAX+5];
ll power1[MAX+5],power2[MAX+5];
const ll N = 200004, mod1 = 1055482763, base1 = 1055476621, mod2 = 2113605293, base2 = 2049246427;
void powc()
{
    power1[0] = power2[0] = 1;
    for (int i = 1; i < N; i++)
        power1[i] = (power1[i - 1] * base1) % mod1;
    for (int i = 1; i < N; i++)
        power2[i] = (power2[i - 1] * base2) % mod2;
}
void pre(string &str)
{
    ll n=str.size();
    H1[0]=str[0];
    H2[0]=str[0];
    for(ll i=1; i<n; i++)
    {    H1[i]=((base1*H1[i-1])%mod1+(str[i]))%mod1;
    }
    for(ll i=1; i<n; i++)
    {
    H2[i]=((base2*H2[i-1])%mod2+(str[i]))%mod2;
    }
}
ll getHash1(ll L,ll R)
{
    if(L==0)
        return H1[R];
    ll x=H1[R];
    ll y=(H1[L-1]*power1[R-L+1])%mod1;
    return (x-y+mod1+mod1)%mod1;
}
ll getHash2(ll L,ll R)
{
    if(L==0)
        return H2[R];
    ll x=H2[R];
    ll y=(H2[L-1]*power2[R-L+1])%mod2;
    return (x-y+mod2+mod2)%mod2;
}
signed main()
{
    ll t=1,cs=1;
   // cin>>t;
    while(t--)
    {
(Lexicographically minimum string after all cycle
shift)
          powc();
    string s;
    cin>>s;
    s+=s;
    pre(s);
    ll n=s.size();
    ll start=0;
    for(ll i=1;i<n;i++)
    {
        ll lo=0,hi=n-1;
        while(lo<=hi)
        {
            ll mid=(lo+hi)/2;
if(getHash1(i,i+mid)==getHash1(start,start+mid)
and
getHash2(i,i+mid)==getHash2(start,start+mid))
            {
                lo=mid+1;
            }
            else hi=mid-1;
        }
        if(lo<n)
        {
            if(s[i+lo]<s[start+lo]) start=i;
        }
    }
    string ans;
    for(ll i=start;n;i++)
    {
        n-=1;
       cout<<s[i];
    }
    return 0;
        // cout<<"Case "<<cs++<<": ";
    }
}
*/
```

Template of Santo

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

#define ll long long
#define deb(x) cout << #x << "=" << x << endl
#define deb2(x, y) cout << #x << "=" << x << " , "
<< #y << "=" << y << endl
#define _ ios::sync_with_stdio(false); cin.tie(0);
cout.tie(0);
#define ff first
#define ss second
#define pb push_back
#define pp pop_back
void solve()
{
ll n, m;
}
int main()
{_
int t = 1, cs = 1;
cin >> t;
while (t--)
{
solve();
}
return 0;
}
```