# KUET_BreakDown

## Contents



# 1 Include

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#define ull unsigned long long
#define ll long long
#define endl '\n'
#define line cout<<"--------------------------"<<endl
#define dd(x) cout<<#x<<" = "<<x<<' '
#define sz(v) (ll)v.size()
#define srt(v) sort(v.begin(),v.end())
#define rsrt(v) sort(v.rbegin(),v.rend())
#define all(v) v.begin(),v.end()
#define pb push_back
#define pi acos(-1)
#define ff first
#define ss second
#define mp make_pair
#define mod 1000000007
#define fast ios_base::sync_with_stdio(false);
    cin.tie(NULL)
#define filein freopen("input.txt","r",stdin)
#define fileout freopen("output.txt","w",stdout)
using namespace std;
using namespace __gnu_pbds;
template <typename T> using os=tree < T, null_type
    ,less<T>,
    rb_tree_tag,tree_order_statistics_node_update >;

///find_by_order(k) : returns an iterator of the k-th
    index
///order_of_key(k) : returns the number of elements
    less than k
mt19937
    rng(chrono::steady_clock::now().time_since_epoch().count
const ll mx=100009;
os<int> s;
int  main()
```

```
{

}
//python:
//a=list(map(int,input().split()))
```

# 2 Math

## 2.1 Sieve

```
const int mx=1000009;
bitset<1000009>prime;
int pr[mx];
vector<int> v;
void sieve()
{
    prime[1]=1;
    for(int i=4;i<mx;i+=2) prime[i]=1;
    for(int i=3;i*i<mx;i+=2)
    {
        if(prime[i]==false)
        {
            for(int j=i*i;j<mx;j+=2*i)
            {

                prime[j]=true;
            }
        }
    }
    v.pb(2);
    for(int i=3;i<mx;i+=2) if(prime[i]==false )
        v.pb(i);
}
void pre()
{
    for(int i=1;i<mx;i++)
    {
        if(i%2==0) pr[i]=2;
        else pr[i]=i;
    }
    for(int i=3;i<mx;i+=2)
    {
        if(pr[i]==i)
        {
            for(int j=i;j<mx;j+=i) pr[j]=i;
        }
    }
}
int main()
{
    sieve();
}
```

## 2.2 Lagrange

```
ll add(ll a, ll b)
{
    if(a+b<mod) return a+b;
    return a+b-mod;
}
ll sub(ll a,ll b)
{
    if(a-b<0) return a-b+mod;
    return a-b;
}
ll mul(ll a, ll b)
{
    return (a*b)%mod;
}
ll bigmod(ll a, ll b, ll m)
{
    ll ans=1;
    a%=m;
    while(b)
    {
        if(b&1) ans=(ans*a)%m;
        b>>=1;
        a=(a*a)%m;
    }
    return ans;
}
ll facto[mx],invfacto[mx];
ll pre[mx],suf[mx];
void pre1()
{
    facto[0]=1;
    for(ll i=1;i<mx;i++) facto[i]=(facto[i-1]*i)%mod;
    invfacto[0]=1;
    invfacto[1]=1;
    for(ll i=2;i<mx;i++)
        invfacto[i]=mod-(mod/i)*invfacto[mod%i]%mod;
        ///pre calculate mod inverse i
    for(ll i=1;i<mx;i++)
        invfacto[i]=(invfacto[i-1]*invfacto[i])%mod;
        ///precalculate inverse factorial
}
ll x,k;
vector<ll> f;
ll get()
{
    if(x<=k+1) return f[x];
    else
    {
        x%=mod;
        ll sum=0;
        ll n=(ll)f.size()-1;
        pre[0]=1;
        suf[k+1]=1;
        for(ll i=0;i<=k+1;i++)
            pre[i+1]=mul(pre[i],(x-i));
        for(ll i=k+1;i>=1;i--)
            suf[i-1]=mul(suf[i],(x-i));
        for(ll i=0;i<=n;i++)
        {
            ll lobb=mul(pre[i],suf[i]);
            lobb=mul(lobb,invfacto[i]);
            lobb=mul(lobb,invfacto[n-i]);
            ll anss=mul(f[i],lobb);
            if((n-i)%2) sum=sub(sum,anss);
            else sum=add(sum,anss);
        }
        return sum;
    }
}
int  main()
{
    fast;
    //find 1^k + 2^k + 3^k + .... + n^k
    //sum is a k+1 degree polynomial, so need (k+2)
        terms to get the actual result
    pre1();
    cin>>x>>k;
    ll sum=0;
    f.pb(0);
```

```
    for(ll i=1;i<=k+1;i++)
    {
        sum=add(sum,bigmod(i,k,mod));
        f.pb(sum);
    }
    cout<<get()<<endl;

}
```

## 2.3 Extended Euclid

```
using namespace std;
int extended(int a, int b,int &x, int &y)
{
    if(b==0)
    {
        x=1; y=0; return a;
    }
    int x1,y1;
    int d=extended(b,a%b,x1,y1);
    x=y1;
    y=x1-(a/b)*y1;
    return d;
}
int  main()
{
    int a,b;
    cin>>a>>b;
    int x,y;
    int ans=extended(a,b,x,y);
    cout<<x<<' '<<y<<endl;
}
```

## 2.4 Catalan

```
ll cat[1120];
ll supcat[1120];
void pre()
{
    cat[0]=1;
    for(ll i=1;i<=50;i++)
    {
        for(ll j=0;j<i;j++)
        {
            cat[i]+=cat[j]*cat[i-1-j];
        }
    }
    supcat[0]=1;
    for(ll i=1;i<=50;i++)
    {
        supcat[i]=supcat[i-1];
        for(ll j=1;j<i;j++)
        {
            supcat[i]+=supcat[j]*supcat[i-j];
        }
    }
    supcat[1]=2;
    for(ll i=1;i<=50;i++) supcat[i]/=2;
}
int main()
{
    pre();
}
```

## 2.5 Chinese Remainder Theorem

```
vector<ll>r,m;
ll extended(ll a, ll b,ll &x, ll &y)
{
    if(b==0)
    {
        x=1; y=0; return a;
    }
    ll x1,y1;
    ll d=extended(b,a%b,x1,y1);
    x=y1; y=x1-(a/b)*y1;
    return d;
}
ll crt() //result is unique when mod by lcm(all m)
{
    ll r1=r[0],m1=m[0];
    for(ll i=1;i<(int)r.size();i++) ///merge solution
        with remaining equation
    {
        ll r2=r[i]; ll m2=m[i];
        ll gcd=__gcd(m1,m2);
        if(r1%gcd != r2%gcd) return -1;
        ll lcm=m1/gcd*m2;
        m1/=gcd; m2/=gcd;
        ll p,q;
        extended(m1,m2,p,q);
        r1=((__int128)(r1*m2)%lcm*q%lcm+(__int128)(r2*m1)%lcm*
        m1=lcm;
        if(r1<0) r1+=lcm;
    }
    return r1; //all solution x=r1+lcm*k
}
int main()
{
    ll n;
    cin>>n;
    for(ll i=0;i<n;i++)
    {
        ll a,b;
        cin>>a>>b;
        m.pb(a); r.pb(b);
    }
    ll ans=crt();
    if(ans==-1) cout<<"Impossible"<<endl;
    else cout<<ans<<endl;
}
```

## 2.6 Derangement Number

```
formula 1: dp[n]=(n-1)*(dp[n-1)+dp[n-2)) where
    dp[1]=0,dp[2]=1

formula 2:
    dp[n]=n!*(1-1/1!+1/2!-1/3!+1/4!-1/5!+1/6!-.....
```

## 2.7 Mobius

```
int mob[10009];
int mobius()
{
    /// 1 when n==1
    /// 0 when there is a prime more than once
```

```
    /// (-1)^p here p is the total distinct prime
        factor

    mob[1]=1;
    for(int i=1;i<=10008;i++)
    {
        for(int j=2*i;j<=10003;j+=i)
        {
            mob[j]-=mob[i];
        }
    }
}
```

## 2.8 Mod Inverse

```
ll ara[mx];
ll facto[mx],invfacto[mx];
void pre()
{
    facto[0]=1;
    for(ll i=1;i<mx;i++) facto[i]=(facto[i-1]*i)%mod;
    invfacto[0]=1;
    invfacto[1]=1;
    for(ll i=2;i<mx;i++)
        invfacto[i]=mod-(mod/i)*invfacto[mod%i]%mod;
        ///pre calculate mod inverse i
    for(int i=1;i<mx;i++)
        invfacto[i]=(invfacto[i-1]*invfacto[i])%mod;
        ///precalculate inverse factorial
}
ll ncr(ll n, ll r)
{
    if(r>n) return 0;
    ll ans=facto[n]*invfacto[n-r];
    ans%=mod;
    ans=(ans*invfacto[r])%mod;
    return ans;
}
```

## 2.9 Phi

```
int toti[mx];
//sum(n)=phi[1]+phi[2]+phi[3]+....phi[n]
//sum(n)=n*(n+1)/2-for(2 to n)sum(floor(n/i))
//sum of coprime number of n = n*phi(n)/2
void phi()
{
    for(int i=1;i<mx;i++) toti[i]=i ;
    for( int i=2;i<mx;i++)
    {
        if(toti[i]==i)
        {
            for(int j=i;j<mx;j+=i)
            {
                toti[j]-=toti[j]/i;
            }
        }
    }
}
int main()
{
   phi();
}
```

## 2.10 Random Generator

```
mt19937
    rng(chrono::steady_clock::now().time_since_epoch().count());
int main()
{
       auto dist=uniform_int_distribution<int>(1,r);
           //random number between 1 and r
       int val=dist(rng);
       cout<<val<<' ';
}
```

## 2.11 Matrix

```
ll dm;
struct matrix
{
    ll mat[20][20];
    matrix()
    {
        memset(mat,0,sizeof mat);
    }
    matrix operator *(const matrix &a) const
    {
        matrix d;
        for(ll i=0;i<dm;i++)
        {
            for(ll j=0;j<dm;j++)
            {
                for(ll k=0;k<dm;k++)
                {
                    d.mat[i][j]+=(mat[i][k]*a.mat[k][j])%mod;
                    d.mat[i][j]%=mod;
                }
            }
        }
        return d;
    }
    void identity()
    {
        for(ll i=0;i<dm;i++)
        {
            mat[i][i]=1;
        }
    }
};
matrix power(matrix c,ll b)
{
    matrix d;
    d.identity();
    while(b)
    {
        if(b&1) d=d*c;
        b>>=1; c=c*c;
    }
    return d;
}
int main()
{
    ///dm->dimension
    ///nth fibonacci
    ///complexity O(d^3 log(n)) where d is dimension
        of matrix
    matrix a;
    int n;
    while(cin>>n){
```

```
        dm=2;
        a.mat[0][0]=1; a.mat[0][1]=1;
        a.mat[1][0]=1; a.mat[1][1]=0;
        if(n==1 or n==2) cout<<1<<endl;
        else {
            a=power(a,n-2);
            ll  ans=a.mat[0][0]+a.mat[0][1];
            cout<<ans<<endl;
        }
    }
}
```

## 2.12 Gauss(mod 2)

```
const ll mx=5e5+9;
ll basis[30];
ll len=0;
void add(ll mask)
{
    for(ll i=0;i<30;i++)
    {
        if((mask & 1LL<<i)==0) continue;
        if(basis[i]==0)
        {
            basis[i]=mask;
            len++;
            return;
        }
        mask^=basis[i];
    }
}
bool check(ll mask) //array element niye particular
    xor banano jay kina
{
    for(ll i=0;i<30;i++)
    {
        if((mask & 1LL<<i)==0) continue;
        if(basis[i]==0) return 0;
        mask^=basis[i];
    }
    return 1;
}
int main()
{
}
```

## 2.13 FFT

```
template <typename float_t>
struct mycomplex {
  float_t x, y;
  mycomplex<float_t>(float_t _x = 0, float_t _y = 0)
      : x(_x), y(_y) {}
  float_t real() const { return x; }
  float_t imag() const { return y; }
  void real(float_t _x) { x = _x; }
  void imag(float_t _y) { y = _y; }
  mycomplex<float_t> &operator+=(const
      mycomplex<float_t> &other) {
    x += other.x;
    y += other.y;
    return *this;
  }
  mycomplex<float_t> &operator-=(const
      mycomplex<float_t> &other) {
    x -= other.x;
    y -= other.y;
    return *this;
  }
  mycomplex<float_t> operator+(const
      mycomplex<float_t> &other) const {
    return mycomplex<float_t>(*this) += other;
  }
  mycomplex<float_t> operator-(const
      mycomplex<float_t> &other) const {
    return mycomplex<float_t>(*this) -= other;
  }
  mycomplex<float_t> operator*(const
      mycomplex<float_t> &other) const {
    return {x * other.x - y * other.y, x * other.y +
        other.x * y};
  }
  mycomplex<float_t> operator*(float_t mult) const {
    return {x * mult, y * mult};
  }
  friend mycomplex<float_t> conj(const
      mycomplex<float_t> &c) {
    return {c.x, -c.y};
  }
  friend ostream &operator<<(ostream &stream, const
      mycomplex<float_t> &c) {
    return stream << '(' << c.x << ", " << c.y << ')';
  }
};
using cd = mycomplex<double>;

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

void multiply(const vector<bool> &a, const
    vector<bool> &b,
              vector<bool> &res) { /* change all the
                  bool to your type needed */
  vector<cd> fa(a.begin(), a.end()), fb(b.begin(),
      b.end());
```

```
  size_t n = 1;
  while (n < max(a.size(), b.size())) n <<= 1;
  n <<= 1;
  fa.resize(n), fb.resize(n);
  fft(fa, false), fft(fb, false);
  for (size_t i = 0; i < n; ++i) fa[i] = fa[i] *
      fb[i];
  fft(fa, true);

  res.resize(n);
  for (size_t i = 0; i < n; ++i) res[i] =
      round(fa[i].real());
}

void pow(const vector<bool> &a, vector<bool> &res,
    long long int k) {
  vector<bool> po = a;
  res.resize(1);
  res[0] = 1;
  while (k) {
    if (k & 1) {
      multiply(po, res, res);
    }
    multiply(po, po, po);
    k /= 2;
  }
}
```

# 3 Geometry

```
const double pi = acos(-1.0);
const double eps = 1e-9;
double ang;
typedef double T;
struct pt
{
    T x, y;
    pt() {}
    pt(T _x, T _y) : x(_x), y(_y) {}
    pt operator+(const pt &p) const
    {
        return pt(x + p.x, y + p.y);
    }
    pt operator-(const pt &p) const
    {
        return pt(x - p.x, y - p.y);
    }
    pt operator*(const T &d) const
    {
        return pt(x * d, y * d);
    }
    pt operator/(const T &d) const
    {
        return pt(x / d, y / d);
    }
    bool operator==(const pt &p) const
    {
        return (x == p.x and y == p.y);
    }
    bool operator!=(const pt &p) const
    {
        return !(x == p.x and y == p.y);
    }
    bool operator<(const pt &p) const
    {
        if (x != p.x)
            return x < p.x;
        return y < p.y;
    }
};

T sq(pt p)
{
    return p.x * p.x + p.y * p.y;
}

double abs(pt p)
{
    return sqrt(sq(p));
}

pt translate(pt v, pt p)
{
    return p + v;
}

pt scale(pt c, double factor, pt p)
{
    return c + (p - c) * factor;
}

pt rot(pt p, double a)
{
    return pt(p.x * cos(a) - p.y * sin(a), p.x *
        sin(a) + p.y * cos(a));
}

pt perp(pt p)
{
    return pt(-p.y, p.x);
}

T dot(pt v, pt w)
{
    return v.x * w.x + v.y * w.y;
}

bool isPerp(pt v, pt w)
{
    return dot(v, w) == 0;
}

double smallAngle(pt v, pt w)
{
    double cosTheta = dot(v, w) / abs(v) / abs(w);
    if (cosTheta < -1)
        cosTheta = -1;
    if (cosTheta > 1)
        cosTheta = 1;
    return acos(cosTheta);
}

T cross(pt v, pt w)
{
    return v.x * w.y - v.y * w.x;
}

T orient(pt a, pt b, pt c)
{
    return cross(b - a, c - a);
}

bool inAngle(pt a, pt b, pt c, pt x)
{
    assert(orient(a, b, c) != 0);
```

```
    if (orient(a, b, c) < 0)
        swap(b, c);
    return orient(a, b, x) >= 0 and orient(a, c, x)
        <= 0;
}

//Line
struct line
{
    pt v;
    T c;
    line() {}
    //From points P and Q
    line(pt p, pt q)
    {
        v = (q - p);
        c = cross(v, p);
    }
    //From equation ax + by = c
    line(T a, T b, T c)
    {
        v = pt(b, -a);
        c = c;
    }
    //From direction vector v and offset c
    line(pt v, T c)
    {
        v = v;
        c = c;
    }

    //These work with T = int / double
    T side(pt p);
    double dist(pt p);
    double sqDist(pt p);
    line perpThrough(pt p);
    bool cmpProj(pt p, pt q);
    line translate(pt t);

    //These require T = double
    line shiftLeft(double dist);
    pt proj(pt p);
    pt refl(pt p);
};

T line ::side(pt p)
{
    return cross(v, p) - c;
}

double line ::dist(pt p)
{
    return abs(side(p)) / abs(v);
}

double line ::sqDist(pt p)
{
    return side(p) * side(p) / (double)sq(v);
}

line line ::perpThrough(pt p)
{
    return line(p, p + perp(v));
}

bool line ::cmpProj(pt p, pt q)
{
    return dot(v, p) < dot(v, q);
}

line line ::translate(pt t)
{
    return line(v, c + cross(v, t));
}

line line ::shiftLeft(double dist)
{
    return line(v, c + dist * abs(v));
}

bool areParallel(line l1, line l2)
{
    return (l1.v.x * l2.v.y == l1.v.y * l2.v.x);
}

bool areSame(line l1, line l2)
{
    return areParallel(l1, l2) and (l1.v.x * l2.c ==
        l2.v.x * l1.c) and (l1.v.y * l2.c == l2.v.y *
        l1.c);
}

bool inter(line l1, line l2, pt &out)
{
    T d = cross(l1.v, l2.v);
    if (d == 0)
        return false;
    out = (l2.v * l1.c - l1.v * l2.c) / d;
    return true;
}

pt line ::proj(pt p)
{
    return p - perp(v) * side(p) / sq(v);
}

pt line ::refl(pt p)
{
    return p - perp(v) * 2 * side(p) / sq(v);
}

line intBisector(line l1, line l2, bool interior)
{
    assert(cross(l1.v, l2.v) != 0);
    double sign = interior ? 1 : -1;
    return line(l2.v / abs(l2.v) + l1.v * sign /
        abs(l1.v),
            l2.c / abs(l2.v) + l1.c * sign /
                abs(l1.v));
}

//Segment
bool inDisk(pt a, pt b, pt p)
{
    return dot(a - p, b - p) <= 0;
}

bool onSegment(pt a, pt b, pt p)
{
    return orient(a, b, p) == 0 and inDisk(a, b, p);
}

bool properInter(pt a, pt b, pt c, pt d, pt &i)
{
    double oa = orient(c, d, a),
          ob = orient(c, d, b),
          oc = orient(a, b, c),
          od = orient(a, b, d);
```

```
    //Proper intersection exists iff opposite signs
    if (oa * ob < 0 and oc * od < 0)
    {
        i = (a * ob - b * oa) / (ob - oa);
        return true;
    }
    return false;
}

bool inters(pt a, pt b, pt c, pt d)
{
    pt out;
    if (properInter(a, b, c, d, out))
        return true;
    if (onSegment(c, d, a))
        return true;
    if (onSegment(c, d, b))
        return true;
    if (onSegment(a, b, c))
        return true;
    if (onSegment(a, b, d))
        return true;
    return false;
}

double segPoint(pt a, pt b, pt p)
{
    if (a != b)
    {
        line l(a, b);
        if (l.cmpProj(a, p) and l.cmpProj(p, b))
            return l.dist(p);
    }
    return min(abs(p - a), abs(p - b));
}

double segSeg(pt a, pt b, pt c, pt d)
{
    pt dummy;
    if (properInter(a, b, c, d, dummy))
        return 0;
    return min(min(min(segPoint(a, b, c), segPoint(a,
        b, d)), segPoint(c, d, a)), segPoint(c, d,
        b));
}
```

```
//int latticePoints (pt a, pt b){
//    //requires int representation
//    return __gcd (abs (a.x - b.x), abs (a.y - b.y))
    + 1;
//}

bool isConvex(vector<pt> &p)
{
    bool hasPos = false, hasNeg = false;
    for (int i = 0, n = p.size(); i < n; i++)
    {
        int o = orient(p[i], p[(i + 1) % n], p[(i + 2)
            % n]);
        if (o > 0)
            hasPos = true;
        if (o < 0)
            hasNeg = true;
    }
    return !(hasPos and hasNeg);
}

double areaTriangle(pt a, pt b, pt c)
{
    return abs(cross(b - a, c - a)) / 2.0;
}

double areaPolygon(const vector<pt> &p)
{
    double area = 0.0;
    for (int i = 0, n = p.size(); i < n; i++)
    {
        area += cross(p[i], p[(i + 1) % n]);
    }
    return fabs(area) / 2.0;
}

ll is_point_in_convex(vector<pt>& p, pt &x) // O(log
    n)
{
    ll n = p.size();
    if (n < 3)
        return 1;
    ll a =orient(p[0], p[1], x), b = orient(p[0], p[n
        - 1], x);
    if (a < 0 || b > 0)
        return 1;
    ll l = 1, r = n - 1;
    while (l + 1 < r)
    {
        int mid = l + r >> 1;
        if (orient(p[0], p[mid], x) >= 0)
            l = mid;
        else
            r = mid;
    }
    ll k = orient(p[l], p[r], x);
    if (k <= 0)
        return -k;
    if (l == 1 && a == 0)
        return 0;
    if (r == n - 1 && b == 0)
        return 0;
    return -1;
}

bool pointInPolygon(const vector<pt> &p, pt q)
    //rezaul vai's
{
    bool c = false;
    for (int i = 0, n = p.size(); i < n; i++)
    {
        int j = (i + 1) % p.size();
        if ((p[i].y <= q.y and q.y < p[j].y or p[j].y
            <= q.y and q.y < p[i].y) and
            q.x < p[i].x + (p[j].x - p[i].x) * (q.y -
                p[i].y) / (p[j].y - p[i].y))
            c = !c;
    }
    return c;
}

pt centroidPolygon(const vector<pt> &p)
{
    pt c(0, 0);
    double scale = 6.0 * areaPolygon(p);
    //    if (scale < eps) return c;
    for (int i = 0, n = p.size(); i < n; i++)
    {
        int j = (i + 1) % n;
        c = c + (p[i] + p[j]) * cross(p[i], p[j]);
    }
```

```
    return c / scale;
}

//Circle
pt circumCenter(pt a, pt b, pt c)
{
    b = b - a;
    c = c - a;
    assert(cross(b, c) != 0);
    return a + perp(b * sq(c) - c * sq(b)) / cross(b,
        c) / 2;
}

bool circle2PtsRad(pt p1, pt p2, double r, pt &c)
{
    double d2 = sq(p1 - p2);
    double det = r * r / d2 - 0.25;
    if (det < 0.0)
        return false;
    double h = sqrt(det);
    c.x = (p1.x + p2.x) * 0.5 + (p1.y - p2.y) * h;
    c.y = (p1.y + p2.y) * 0.5 + (p2.x - p1.x) * h;
    return true;
}

int circleLine(pt c, double r, line l, pair<pt, pt>
    &out)
{
    double h2 = r * r - l.sqDist(c);
    if (h2 < 0)
        return 0; // the line doesn't touch the circle;
    pt p = l.proj(c);
    pt h = l.v * sqrt(h2) / abs(l.v);
    out = make_pair(p - h, p + h);
    return 1 + (h2 > 0);
}

int circleCircle(pt c1, double r1, pt c2, double r2,
    pair<pt, pt> &out)
{
    pt d = c2 - c1;
    double d2 = sq(d);
    if (d2 == 0)
    { //concentric circles
        assert(r1 != r2);
        return 0;
    }
    double pd = (d2 + r1 * r1 - r2 * r2) / 2;
    double h2 = r1 * r1 - pd * pd / d2; // h ^ 2
    if (h2 < 0)
        return 0;
    pt p = c1 + d * pd / d2, h = perp(d) * sqrt(h2 /
        d2);
    out = make_pair(p - h, p + h);
    return 1 + h2 > 0;
}

int tangents(pt c1, double r1, pt c2, double r2, bool
    inner, vector<pair<pt, pt>> &out)
{
    if (inner)
        r2 = -r2;
    pt d = c2 - c1;
    double dr = r1 - r2, d2 = sq(d), h2 = d2 - dr *
        dr;
    if (d2 == 0 or h2 < 0)
    {
        assert(h2 != 0);
        return 0;
    }
    for (int sign : {-1, 1})
    {
        pt v = pt(d * dr + perp(d) * sqrt(h2) * sign)
            / d2;
        out.push_back(make_pair(c1 + v * r1, c2 + v *
            r2));
    }
    return 1 + (h2 > 0);
}

//Convex Hull - Monotone Chain
int sz; //returned polygon's size;
pt H[100000 + 5];
vector<pt> monotoneChain(vector<pt> &v) /// from
    you_know_who
{
    if(v.size()==1)
                return v;

        sort(v.begin(), v.end());
        vector<pt> up(2*v.size()+2),
            down(2*v.size()+2);
    int szup=0, szdw=0;

        for(int i=0;i<v.size();i++)
        {
        // cout<<"p: ("<<v[i].x<<" "<<v[i].y<<"
            )"<<endl;
        while(szup>1 && orient(up[szup-2], up[szup-1],
            v[i])>=0)
            szup--;
        while(szdw>1 && orient(down[szdw-2],
            down[szdw-1], v[i])<=0)
            szdw--;
        up[szup++]=v[i];
        down[szdw++]=v[i];
        }

    if(szdw>1)
        szdw--;

    reverse(up.begin(), up.begin()+szup);
        for(int i=0;i<szup-1;i++)
                down[szdw++]= up[i];


    if(szdw==2 && down[0].x==down[1].x &&
        down[0].y==down[1].y)
    {
        szdw--;
    }
    sz = szdw;

        return down;
}

pt toPoint(pt p1, pt p2)
{
    return pt(p2.x-p1.x,p2.y - p1.y);
}

double angle(pt a, pt o, pt b)
{
    //double result = atan2(P3.y - P1.y, P3.x - P1.x)
        - atan2(P2.y - P1.y, P2.x - P1.x);

    double result;
```

```
    pt oa = toPoint(o, a);
    pt ob = toPoint(o, b);

    if(sq(oa)==0 || sq(ob)==0){
        return 0.0;
    }

    result = acos(dot(oa, ob)/sqrt(sq(oa)*sq(ob)));
    result*=180.0;
    result/=Pi;
    return result;
}
```

# 4 Data Structure

## 4.1 SQRT Decomposition

```
int n;
int ara[10000];
int block[100];
int bs;
void update(int ind, int val)
{
    block[ind/bs]+=val-ara[ind];
    ara[ind]=val;
}
int query(int a,int b)
{
    int sum=0;
    for(int i=a;i<=b;)
    {
        if(i%bs==0 and i+bs-1<=b) sum+=block[i/bs],
            i+=bs;
        else sum+=ara[i], i++;
    }
    return sum;
}
void pre()
{
     bs=sqrtl(n);
     for(int i=0;i<n;i++)
     {
         block[i/bs]+=ara[i];
     }
}
int  main()
{
    cin>>n;
    for(int i=0;i<n;i++) cin>>ara[i];
    pre();
    update(0,0);
    cout<<query(0,3);
}
```

## 4.2 Mo's Algorithm

```
int n,m;
int ara[mx];
int block;
struct query
{
    int l,r,id;
}q[mx];
ll ans[mx];
int cnt[mx];
int anss=0;
bool comp(query a, query b)
{
    if(a.l/block!=b.l/block) return
        a.l/block<b.l/block;
    return a.r<b.r;
}
void add(int ind)
{
   if(cnt[ara[ind]]==0) anss++;
    cnt[ara[ind]]++;
}
void _remove(int ind)
{
   if(cnt[ara[ind]]==1) anss--;
    cnt[ara[ind]]--;
}
void MO()
{
   anss=0;
    int l=0,r=-1;
    for(int i=0;i<m;i++)
    {
        int left=q[i].l;
        int right=q[i].r;
        left--; right--;
        while(l<left) _remove(l++);
        while(l>left) add(--l);
        while(r<right) add(++r);
        while(r>right) _remove(r--);
        ans[q[i].id]=anss;
        //cout<<left<<' '<<right<<' '<<anss<<endl ;
    }
}
int  main()
{
        ///complexity: O(n*sqrt(n)+q*sqrt(n))
   fast;
   int cs=1;
   int t; cin>>t;
   while(t--){
      memset(cnt,0,sizeof cnt);
    cin>>n>>m;
    for(int i=0;i<n;i++) cin>>ara[i];
    for(int i=0;i<m;i++)
    {
        cin>>q[i].l>>q[i].r;
        q[i].id=i;
    }
    block=sqrtl(n);
    sort(q,q+m,comp);
    MO();
    cout<<"Case "<<cs++<<":"<<endl;
    for( int i=0;i<m;i++)
    {
        cout<<ans[i]<<endl;
    }
   }
}
```

## 4.3 Segment Tree(Lazy)

```
int ara[mx];
int tree[mx*4];
int lazy[mx*4];
void init(int node,int b, int e)
{
```

```
    if(b==e)
    {
        tree[node]=ara[b];
        return ;
    }
    int mid=(b+e)>>1;
    init(2*node, b,mid);
    init(2*node+1,mid+1,e);
    tree[node]=tree[2*node]+tree[2*node+1];
}
void pushdown(int node,int b,int e)
{
    if(lazy[node])
    {
        tree[node]+=(e-b+1)*lazy[node];
        if(b!=e)
        {
            lazy[node*2]+=lazy[node];
            lazy[node*2+1]+=lazy[node];
        }
        lazy[node]=0;
    }
}
void update(int node,int b, int e, int i, int j,int
     value)
{
    pushdown(node,b,e);
    if(e<i or b>j) return ;
    if(b>=i and e<=j)
    {
        lazy[node]+=value;
        pushdown(node,b,e);
        return ;
    }
    int mid=(b+e)>>1;
    update(2*node, b,mid,i,j,value);
    update(2*node+1,mid+1,e,i,j,value);
    tree[node]=tree[2*node]+tree[2*node+1];
}
int  query(int node,int b, int e, int i,int j)
{
    pushdown(node,b,e);
    if(e<i or b>j) return 0;
    if(b>=i and e<=j)
    {
        return tree[node];
    }
    int mid=(b+e)>>1;
    int x=query(2*node, b,mid,i,j);
    int y=query(2*node+1,mid+1,e,i,j);
    return x+y;
}
int  main()
{

}
```

## 4.4 Persistent Segment Tree

```
int ara[mx];
int cnt=1;
struct node
{
    int l,r,val;
}tr[4*mx+mx*20];
void init(int node, int b, int e)
{
    if(b==e)
    {
        tr[node].val=0;
        return ;
    }
    int mid=(b+e)>>1;
    tr[node].l=cnt++;
    tr[node].r=cnt++;
    init(tr[node].l,b,mid);
    init(tr[node].r,mid+1,e);
    tr[node].val=tr[tr[node].l].val+tr[tr[node].r].val;
}
int update(int pre, int b, int e, int i, int j,int
     val)
{
    if(b>j or e<i) return pre;
    int curr=cnt++;
    if(b>=i and e<=j)
    {
        tr[curr].val=tr[pre].val+val;
        return curr;
    }
    int mid=(b+e)>>1;
    tr[curr].l=update(tr[pre].l,b,mid,i,j,val);
    tr[curr].r=update(tr[pre].r,mid+1,e,i,j,val);
    tr[curr].val=tr[tr[curr].l].val+tr[tr[curr].r].val;
    return curr;
}
int query(int pre, int cur, int b, int e, int i, int
     j)
{
   if(b>j or e<i) return 0;
   if(b>=i and e<=j) return tr[cur].val-tr[pre].val;
   int mid=(b+e)>>1;
   int x=query(tr[pre].l,tr[cur].l,b,mid,i,j);
   int y=query(tr[pre].r,tr[cur].r,mid+1,e,i,j);
   return x+y;
}
int root[mx];
int main()
{
    //spoj kquery
    int n;
    cin>>n;
    root[0]=0;
    init(root[0],0,n);
    vector<int> v;
    for(int i=0;i<n;i++)
    {
       cin>>ara[i];
       v.pb(ara[i]);
    }
    sort(v.begin(),v.end());
    for(int i=0;i<n;i++)
        ara[i]=lower_bound(v.begin(),v.end(),ara[i])-v.begin(
    for(int i=1;i<=n;i++)
    {
       root[i]=update(root[i-1],0,n,ara[i-1],ara[i-1],1);
    }
    int q; cin>>q;
    while(q--)
    {
       int l,r,k;
       cin>>l>>r>>k;
       k=upper_bound(v.begin(),v.end(),k)-v.begin();
       cout<<query(root[l-1],root[r],0,n,k,n)<<endl;
    }
```

```
}
```

## 4.5 Persistent Segment Tree(Lazy)

```
const ll mx=100009;
struct node
{
    int val,lazy;
    node *l,*r;
    node()
    {
        val=lazy=0; l=r=nullptr;
    }
    node(const node* p)
    {
        if(p==nullptr)
        {
            val=lazy=0; l=r=nullptr;
        }
        else
        {
            val=p->val; lazy=p->lazy;
            l=p->l; r=p->r;
        }
    }
};
void pushdown(node *&cur, int b, int e)
{

    cur->l=new node(cur->l);
    cur->r=new node(cur->r);
    if(cur->lazy)
    {
        cur->val+=(e-b+1)*cur->lazy;
        if(b!=e)
        {
            cur->l->lazy+=cur->lazy;
            cur->r->lazy+=cur->lazy;
        }
        cur->lazy=0;
    }
}
void update(node *&cur, int b, int e, int i, int j,
     int val)
{
    pushdown(cur,b,e);
    if(b>j or e<i) return ;
    if(b>=i and e<=j)
    {
        cur->val+=(e-b+1)*val;
        cur->l->lazy+=val;
        cur->r->lazy+=val;
        return ;
    }
    int mid=(b+e)>>1;
    update(cur->l,b,mid,i,j,val);
    update(cur->r,mid+1,e,i,j,val);
    cur->val=cur->l->val+cur->r->val;
}
int query(node *&cur, int b, int e, int i, int j)
{
    pushdown(cur,b,e);
    if(b>j or e<i) return 0;
    if(b>=i and e<=j) return cur->val;
    int mid=(b+e)>>1;
    int x=query(cur->l,b,mid,i,j);
    int y=query(cur->r,mid+1,e,i,j);
    return x+y;
}
node *root[mx];
int ara[mx];
int main()
{
    root[0]=new node();
    int n; cin>>n;
    for(int i=1;i<=n;i++) cin>>ara[i];
    for(int i=1;i<=n;i++)
    {
        root[i]=new node(root[i-1]);
        update(root[i],1,n,1,ara[i],5); //add 5 to the
            range 1 to ara[i]
    }

}
```

## 4.6 RMQ(sparse table)

```
int ara[mx];
int mini[mx][22];
int n;
void sparse()
{
    for(int i=0;i<n;i++) mini[i][0]=ara[i];
    for(int j=1;(1LL<<j)<=n;j++)
    {
        for(int i=0;i+(1LL<<(j))-1<n;i++)
        {
            int d=1LL<<(j-1);
            mini[i][j]=min(mini[i][j-1],mini[i+d][j-1]);
        }
    }
}
int query(int l, int r)
{
    int d=__lg(r-l+1);
    return min(mini[l][d],mini[r-(1LL<<d)+1][d]);
}
int main()
{
    cin>>n;
    for(int i=0;i<n;i++) cin>>ara[i];
    sparse();
    int q; cin>>q;
    while(q--)
    {
        int l,r;
        cin>>l>>r;
        cout<<query(l,r)<<endl;
    }
}
```

## 4.7 BIT

```
const int mx=3e5;
struct bit
{
    int n;
    int tree[mx];
    void update(int ind, int value)
```

```
    {
        for(int i=ind;i<=n;i+=(i&-i))
        {
            tree[i]+=value;
        }
    }
    int query(int ind)
    {
        int sum=0;
        for(int i=ind;i>0;i-=(i&-i))
        {
            sum+=tree[i];
        }
        return sum;
    }
}a,b;
int  main()
{
    a.n=5;
    a.update(3,4);
    cout<<a.query(5)<<endl;
}
```

## 4.8 Implicit Treap

```
struct node
{
    ll key,pri,sz,lazy,sum;
    node *l, *r;
    bool rev;
    node(){};
    node (ll val)
    {
        lazy=0,sum=val;
        rev=0; sz=0; key=val; pri=rng(); l=r=NULL;
    }
};
node* root=NULL;
int cnt(node *cur)
{
    return cur? cur->sz: 0;
}
void push(node *cur)
{
    if(cur==NULL) return ;
    if(cur and cur->lazy)
    {
        cur->key+=cur->lazy;
        cur->sum+=cur->lazy*cnt(cur);
        if(cur->l) cur->l->lazy+=cur->lazy;
        if(cur->r) cur->r->lazy+=cur->lazy;
        cur->lazy=0;

    }
    if(cur and cur->rev)
    {
        cur->rev^=1;
        swap(cur->l,cur->r);
        if(cur->l) cur->l->rev^=1;
        if(cur->r) cur->r->rev^=1;
    }
}
vector<int> ans;
void path(node *cur)
{
    if(cur==NULL) return ;
    push(cur);
    path(cur->l);
    cout<<cur->key<<' ';
    //ans.pb(cur->key);
    path(cur->r);
}
int getsum(node *cur)
{
   if(cur==NULL) return 0;
   return cur->sum;
}
void update (node *cur)
{
   if(cur==NULL) return ;
   push(cur->l);
   push(cur->r);
   cur->sum=cur->key;
   cur->sz=1+cnt(cur->l)+cnt(cur->r);
   cur->sum+=getsum(cur->l)+getsum(cur->r);

}
void split(node *cur,node *&l, node *&r,int pos, int
    add=0)
{
    if(cur==NULL)
    {
        l=r=NULL;
        return ;
    }
    push(cur);
    int kk=add+cnt(cur->l);
    if(pos<=kk) split(cur->l,l,cur->l,pos,add),
        r=cur; //element at pos goes to node r
    else split(cur->r,cur->r,r,pos,kk+1), l=cur;
    update(cur);
}
void merge(node * &cur, node *l, node *r)
{
    push(l); push(r);
    if(l==NULL or r==NULL) cur=l?l:r;
    else if(l->pri > r->pri) merge(l->r,l->r,r),cur=l;
    else merge(r->l,l,r->l),cur=r;
    update(cur);
}
void insert(node *&cur, node *it,int pos)
{
    node *l,*r;
    split(cur,l,r,pos);
    merge(l,l,it);
    merge(cur,l,r);
}
void erase(node *&cur, int pos)
{
    if(cur==NULL) return ;
    node *l,*mid,*r;
    split(cur,l,mid,pos);
    split(mid,mid,r,1);
    merge(cur,l,r);
}
void reverse(node *root, int a, int b)
{
    node *l,*r,*mid;
    split(root,l,mid,a);
    split(mid,mid,r,b-a+1);
    mid->rev^=1;
    merge(root,l,mid);
    merge(root,root,r);
}
void right_shift(node *root,int a, int b)
```

```
{
    node *a1; //store 0-> b-1;
    node *a2; //store b-> n-1;
    split(root,a1,a2,b);
    node *bb; //store b->b
    split(a2,bb,a2,1);
    //now a2 will store b+1-> n-1
    node *a3; //store 0->a-1
    split(a1,a3,a1,a);
    //now a1 will store a->b-1
    merge(root,a3,bb);
    merge(a1,a1,a2);
    merge(root,root,a1);
}
void left_shift(node *root,int a, int b)
{
    node *a1; //store 0-> b;
    node *a2; //store b+1-> n-1;
    split(root,a1,a2,b+1);
    node *a3; //store a+1 -> b
    split(a1,a1,a3,a+1);
    //now a1 will store 0 -> a
    node *aa; //store a -> a
    split(a1,a1,aa,a);
    //now a1 will store 0 -> a
    merge(a1,a1,a3);
    merge(a1,a1,aa);
    merge(root,a1,a2);
}
// 0 indexing treap
//merge(a,b,c) b and c will be merged and their root
    will be stored in a
//split(a,b,pos) a will store array index from (0 to
    pos-1), b will store array ( pos to n-1)
int main()
{
   insert(root,new node(4),2);
   insert(root, new node(10),0);
   insert(root, new node(15),1);
   path(root);
   cout<<endl;
}
```

# 5 Graph

## 5.1 2-SAT

```
const ll mx=500009;
vector<int> v[mx*2],rev[mx*2];
int scc[mx*2];
bool visit[mx*2];
stack<int> st;
void dfs(int s)
{
    visit[s]=true;
    for(auto x: v[s] )
    {
        if(visit[x]==false)
        {
            dfs(x);
        }
    }
    st.push(s);
}
void dfs1(int s,int no)
{
    visit[s]=true;
    scc[s]=no;
    for(auto x: rev[s])
    {
        if(visit[x]==false)
        {
            dfs1(x,no);
        }
    }
}
int n;
int ans[mx];
bool twosat()
{
    for(int i=0;i<2*n;i++)
     {
         if(visit[i]==false)
         {
             dfs(i);
         }
     }
     int mark=0;
     memset(visit,0,sizeof visit);
     while(st.size())
     {
         int a=st.top();
         st.pop();
         if(visit[a]==false)
         {
             mark++;
             dfs1(a,mark);
         }
     }
    for(int i=0;i<n;i++)
    {
        if (scc[2*i] == scc[2*i+1]) return false;
        ans[2*i] = scc[2*i] > scc[2*i+1];
    }
    return true;
}
void add(int a, int b)
{
    v[a].pb(b);
    rev[b].pb(a);
}
int  main()
{
    cin>>n;
    for(int i=0;i<n;i++)
    {
        int a,b;
        cin>>a>>b;
        add(a,b);
    }
    if(twosat()==0) no();
    else yes();
}
```

## 5.2 Articulation Bridge

```
int n;
vector<int>v[mx];
int dis[mx],low[mx];
bool visit[mx];
int t=0;
set<pair<int,int> > edge;
void dfs( int s, int par)
```

```
{

    dis[s]=low[s]=t++;
    int child=0;
    visit[s]=true;
    for(auto x: v[s])
    {
        if(x==par) continue;
        if(visit[x])
        {
            low[s]=min(low[s],dis[x]);
            continue;
        }
        dfs(x,s);
        child++;
        low[s]=min(low[s],low[x]);
        if(dis[s]<low[x])
        {
            edge.insert(mp(min(s,x),max(s,x)));
        }
    }
}
int  main()
{
        int m ;
        cin>>n>>m;
        for(int i=0;i<m;i++)
        {
            int a,b;
            cin>>a>>b;
            v[a].pb(b);
            v[b].pb(a);
        }
        for(int i=0;i<n;i++)
        {
            if(visit[i]==false)
            {
                dfs(i,-1);
            }
        }
        printf("%d critical links\n",edge.size());
        for(auto x: edge)
        {
            printf("%d - %d\n",x.first,x.second);
        }
}
```

## 5.3 Articulation Point

```
vector<int> v[100];
bool visit[100];
int t ;
int dis[100];
int low[100];
bool ans[100];
void dfs(int s,int par)
{
    dis[s]=t++;
    low[s]=dis[s];
    visit[s]=true;
    int child=0;
    for(auto x: v[s])
    {
        if(x==par) continue;
        if(visit[x])
        {
            low[s]=min(low[s],dis[x]);
            continue;
        }
        dfs(x,s);
        low[s]=min(low[s],low[x]);
        if(dis[s]<=low[x] and par!=-1)
        {
            ans[s]=true; ///one point can be counted
                more than once
        }
        child++;
        if(child>1 and par==-1) ans[s]=true;
    }
}
int  main()
{
        v[1].pb(2);
        v[2].pb(1);
        v[2].pb(3);
        v[3].pb(2);
        dfs(1,-1);
        for(int i=1;i<=7;i++)
        {
            if(ans[i]) cout<<i<<' ' ;
        }
}
```

## 5.4 Dijkstra

```
ll n,m;
vector<pair<ll,ll> > v[100008];
ll dis[100009];
ll par[100009];
void dij()
{
    for(ll i=0;i<=n;i++) dis[i]=LLONG_MAX;
    priority_queue<pair<ll,ll> > pq;
    pq.push(mp(0,1));
    dis[1]=0;
    par[1]=-1;
    par[n]=INT_MAX;
    while(pq.size())
    {
        ll u=pq.top().second;
        int cost=-pq.top().first;
        pq.pop();
        if(cost>dis[u]) continue;
        for(auto x: v[u])
        {
            if(cost+x.second<dis[x.first])
            {
                dis[x.first]=cost+x.second;
                pq.push(mp(-dis[x.first],x.first));
                par[x.first]=u;
            }
        }
    }
}
void path(ll x)
{
    if(x==-1) return ;
    path(par[x]);
    cout<<x<<' ';
}
int  main()
{
    cin>>n>>m;
```

```
    for(ll i=0;i<m;i++)
    {
        ll a,b,c;
        cin>>a>>b>>c;
        v[a].pb(mp(b,c));
        v[b].pb(mp(a,c));
    }
    dij();
    if(par[n]==INT_MAX) cout<<-1<<endl;
    else path(n);

}
```

## 5.5 Heavy Light Dec.

```
vector<ll > v[mx];
int level[mx],heavy[mx],head[mx],pos[mx],parent[mx];
int curr;
int tree[mx];
int ara[mx];
int n;
void init(int node, int b, int e)
{
    if(b==e)
    {
        tree[node]=ara[b];
        return ;
    }
    int mid=(b+e)>>1;
    init(node*2,b,mid);
    init(node*2+1,mid+1,e);
    tree[node]=max(tree[node*2],tree[node*2+1]);
}
void update(int node, int b, int e, int i, int val)
{
    if(b>i or e<i) return ;
    if(b==e)
    {
        tree[node]=val;
        return ;
    }
    int mid=(b+e)>>1;
    update(node*2,b,mid,i,val);
    update(node*2+1,mid+1,e,i,val);
    tree[node]=max(tree[node*2],tree[node*2+1]);
}
int query(int node, int b, int e, int i, int j)
{
    if(b>j or e<i) return 0;
    if(b>=i and e<=j) return tree[node];
    int mid=(b+e)>>1;
    return
        max(query(node*2,b,mid,i,j),query(node*2+1,mid+1,e,i,j));
}
int dfs(int s, int par)
{
    int ss=1; ///subtree size
    int maxi=0;
    for(auto x: v[s])
    {
        if(x!=par)
        {
            parent[x]=s; level[x]=level[s]+1;
            int d=dfs(x,s); ///returns the subtree
                size of x
            ss+=d;
            if(d>maxi)
            {
                maxi=d; heavy[s]=x;
            }

        }
    }
    return ss;
}
void hld(int s, int h)
{
    head[s]=h; ///head of chain in which s lies
    pos[s]=curr; ///position of s in the segment tree
    ara[curr++]=s;
    if(heavy[s]!=-1)
    {
        hld(heavy[s],h); ///increase chain
    }
    for(auto x: v[s])
    {
        if(x!=parent[s] and x!=heavy[s])
        {
            hld(x,x); ///start new chain
        }
    }
}
int solve(int a, int b)
{
    int ans=0;
    for(; head[a]!=head[b];a=parent[head[a]])
        ///maximum log(n) times
    {
        if(level[head[a]]<level[head[b]]) swap(a,b);
        int temp=query(1,0,n-1,pos[head[a]],pos[a]);
        ans=max(temp,ans);
    }
    if(level[a]>level[b]) swap(a,b);
    ans=max(ans,query(1,0,n-1,pos[a],pos[b]));
    return ans;
}
int main()
{
    cin>>n;
    for(int i=0;i<n-1;i++)
    {
        int a,b;
        cin>>a>>b;
        v[a].pb(b);
        v[b].pb(a);
    }
    memset(heavy,-1,sizeof heavy);
    dfs(1,-1);
    hld(1,1);
    init(1,0,n-1);
    cout<<solve(1,6)<<endl; ///maximum value on the
        path from a to b
}
```

## 5.6 LCA

```
int level[mx];
int table[mx][22];
vector<int>v[mx];
void dfs(int curr,int pre)
{
    if(pre==-1) level[curr]=0;
    else level [curr]=level[pre]+1;
    table[curr][0]=pre;
```

```
    for(auto x: v[curr])
    {
        if(x==pre) continue;
        dfs(x,curr);
    }
}
int lca(int p, int q)
{
     if(level[p]<level[q]) swap(p,q);
     int log=log2(level[p]);
     for( int i=log;i>=0;i--)
     {
         if(level[p]-(1<<i)>=level[q])
         {
             p=table[p][i];
         }
     }
     if(p==q) return p ;
      for( int i=log;i>=0;i--)
     {
         if(table[p][i]!=-1 and
             table[p][i]!=table[q][i])
         {
             p=table[p][i],q=table[q][i];
         }
     }
     return table[p][0];
}
void sparse()
{
    for(int j=1;1<<j<=n;j++)
     {
         for(int i=0;i<n;i++) ///node number starts
             from 0
         {
             if(table [i][j-1]!=-1)
             {
                 table [i][j]=table
                     [table[i][j-1]][j-1];
             }
         }
     }
}
int main()
{
    n=6;
        v[0].pb(2);
        v[2].pb(3);
        v[3].pb(4);
        v[4].pb(5);
        v[4].pb(6);
        memset(table,-1,sizeof table);
        dfs(0,-1);
        sparse();
        printf( "%d\n",lca(6,5) );
        return 0;
}
```

## 5.7 SCC

```
vector<int> v[10009],rev[10009],scc[10009];
bool visit[10009];
stack<int> st;
void dfs(int s)
{
    visit[s]=true;
    for(auto x: v[s] )
    {
        if(visit[x]==false)
        {
            dfs(x);
        }
    }
    st.push(s);
}
void dfs1(int s,int no)
{
    visit[s]=true;
    scc[no].pb(s);
    for(auto x: rev[s])
    {
        if(visit[x]==false)
        {
            dfs1(x,no);
        }
    }
}
int main()
{
        ///only for directed graph
    int n;
    scanf("%d",&n);
    set<int> s;
    for(int i=0;i<n;i++)
    {
        int a,b;
        scanf("%d%d",&a,&b);
        v[a].pb(b);
        rev[b].pb(a);
        s.insert(a);
        s.insert(b);
    }
    for(auto x: s)
    {
        if(visit[x]==false)
        {
            dfs(x);
        }
    }
    int mark=0;
    memset(visit,0,sizeof visit);
    while(st.size())
    {
        int a=st.top();
        st.pop();
        if(visit[a]==false)
        {
            mark++;
            dfs1(a,mark);
        }
    }
    for(int i=1;i<=mark;i++)
    {
        cout<<i<<": ";
        for(auto x:scc[i])
        {
            cout<<x<<' ';
        }
        cout<<endl;
    }
}
```

## 5.8 Kruskal

```
struct edge
{
    ll u,v,w;
};
ll n,m;
vector<edge> v;
ll par[1000000];
ll findp(ll x)
{
    if(par[x]==x) return x;
    return par[x]=findp(par[x]);
}
bool cmp(edge a, edge b)
{
    return a.w<b.w;
}
void mst()
{
    sort(v.begin(),v.end(),cmp);
    ll ans=0;
    for(ll i=1;i<=n;i++) par[i]=i;
    for(ll i=0;i<v.size();i++)
    {
        ll a=v[i].u;
        ll b=v[i].v;
        ll cost=v[i].w;
        ll m1=findp(a);
        ll m2=findp(b);
        if(m1!=m2)
        {
            par[m1]=m2;
            ans+=cost;
        }
    }
    int cnt=0;
    for(ll i=1;i<=n;i++)
    {
        ll x=findp(i);
        if(x==i) cnt++;
    }
    if(cnt>1) cout<<"IMPOSSIBLE"<<endl;
    else cout<<ans<<endl;
}
int main()
{
    while(cin>>n>>m and ( n or m)){
    for(ll i=0;i<m;i++)
    {
        edge a;
        cin>>a.u>>a.v>>a.w;
        v.pb(a);
    }
    mst();
    v.clear();
    }
}
```

## 5.9 Bellman Ford

```
struct edge
{
    int a,b,c;
};
vector<edge> v;
ll dis[1009];
int n,m;
void bford()
{
    for(int i=0;i<n;i++) dis[i]=INT_MAX;
    dis[0]=0;
    for(int i=0;i<n-1;i++)
    {
        for(int j=0;j<v.size();j++)
        {
            int a=v[j].a;
            int b=v[j].b;
            int c=v[j].c;
            if(dis[a]==INT_MAX) continue ;
            if(dis[a]+c<dis[b])
            {
                dis[b]=dis[a]+c;
            }
        }
    }
    int k=0;
    for(int j=0;j<v.size();j++)
        {
            int a=v[j].a;
            int b=v[j].b;
            int c=v[j].c;
            if(dis[a]+c<dis[b])
            {
                k=1;
                break;
            }
        }
    if(k) cout<<"find negative cycle"<<endl;
    else cout<<"no negative cycle"<<endl;
}
int  main()
{
    ///check negative cycle
    cin>>n>>m;
    for(int i=0;i<m;i++)
    {
        edge d;
        cin>>d.a>>d.b>>d.c;
        v.pb(d);
    }
    bford();
}
```

## 5.10 Dinic

```
const int mx = 500+5 ;
const int INF = 1000000000 ;
struct edge
{
    int a, b, cap, flow ;
};
int src,snk,level[mx],ptr[mx] ;
vector<edge>e;
vector<int>g[mx];
void add(int a,int b,int cap)
{
    edge e1 = { a, b, cap, 0 } ;
    edge e2 = { b, a, 0 , 0 } ;
    g[a].push_back((int)e.size());
    e.push_back(e1);
    g[b].push_back((int)e.size());
    e.push_back(e2);
}

bool bfs()
```

```
{
    int s = src , t = snk ;
    queue < int > q ;
    q.push(s);
    memset(level,-1,sizeof(level));
    level[s]=0 ;
    while (!q.empty())
    {
        int u=q.front() ;
        q.pop() ;
        for(int i=0; i<(int)g[u].size(); i++)
        {
            int id=g[u][i];
            int v=e[id].b;
            if(level[v]==-1&&e[id].flow<e[id].cap)
            {
                q.push(v) ;
                level[v]=level[u]+1 ;
            }
        }
    }
    return level[t]!=-1 ;
}

int dfs(int u,int flow)
{
    if (!flow) return 0 ;
    if ( u==snk ) return flow ;
    for(; ptr[u]<(int)g[u].size(); ptr[u]++)
    {
        int id = g[ u][ptr[u]];
        int v = e[id].b ;
        if ( level[v] != level[u]+1 ) continue ;
        int tempflow = dfs (v,min
            (flow,e[id].cap-e[id].flow)) ;
        if (tempflow)
        {
            e [id].flow+=tempflow ;
            e [id^1].flow-=tempflow ;
            return tempflow ;
        }
    }
    return 0 ;
}

int dinic()
{
    int flow = 0 ;
    while(bfs())
    {
        memset(ptr,0,sizeof(ptr) ) ;
        while ( int tempflow= dfs(src,INF ) )
            flow += tempflow ;
    }
    return flow ;
}
int main()
{
   // complexity O(v*v*e)
    int n,m;
    cin>>n>>m;
    cin>>src>>snk;
    for(int i=0;i<m;i++)
    {
        int a,b,c;
        cin>>a>>b>>c;
        add(a,b,c);
        add(b,a,c);
    }
        cout<<dinic()<<endl;
}
```

## 5.11 Hopcroft Karp

```
#include <bits/stdc++.h>
#define ull unsigned long long
#define ll long long
#define pb push_back
#define mp make_pair
#define fast ios_base::sync_with_stdio(false);
    cin.tie(NULL)
#define filein freopen("input.txt","r",stdin)
#define fileout freopen("output.txt","w",stdout)
using namespace std;
const ll mx=1009;
const ll inf=(ll)1e15;
vector<ll > v[mx];
ll  n, match[mx], dis[mx];
void add(ll a, ll b)
{
    v[a].pb(b);
}
bool  bfs()
{
    queue<ll > q;
    for(ll  i=1;i<=n;i++)
    {
        if(match[i]==0)
        {
            dis[i]=0; q.push(i);
        }
        else dis[i]=inf;
    }
    dis[0]=inf;
    while(q.size())
    {
        ll  a=q.front(); q.pop();
        for(auto x: v[a])
        {
            if(dis[match[x]]==inf)
            {
                dis[match[x]]=dis[a]+1;
                q.push(match[x]);
            }
        }
    }
    return dis[0]!=inf;
}
bool dfs(ll u)
{
    if(u)
    {
        for(auto x: v[u])
        {
            if(dis[match[x]]==dis[u]+1)
            {
                if(dfs(match[x]))
                {
                    match[x]=u;
                    match[u]=x;
                    return true;
                }
            }
        }
        dis[u]=inf;
        return false;
```

```
    }
    return true;
}
ll  hopcroft()
{
    ll  ans=0;
    while(bfs())
    {
        for(ll i=1;i<=n;i++)
        {
            if(match[i]==false and dfs(i))
            {
                ans++;
            }
        }
    }
    return ans;
}
int  main()
{
    ///complexity O(sqrt(v)*e)
    ///find maximum matching in unweighted graph
    n=4; ///elements of the left side, start from 1
    add(1,6); add(1,7);
    add(2,5); add(3,6);
    add(4,6); add(4,8);
    cout<<hopcroft()<<endl; ///ans 4
}

minimum vertex cover (cover all edges)      : bpm
minimum edge cover (cover all vertices)     : n-bpm
size of maximum independent set             : n-bpm
```

### 5.12 Biconnected Comp.

```
vector<int> v[100];
bool visit[100];
int t ;
int dis[100];
int low[100];
bool ans[100];
stack<pair<int,int> > st;
void dfs(int s,int par)
{
    dis[s]=t++;
    low[s]=dis[s];
    visit[s]=true;
    int child=0;
    for(auto x: v[s])
    {
        if(x==par) continue;
        if(visit[x])
        {
            low[s]=min(low[s],dis[x]);
            if(dis[x]<dis[s]) st.push(mp(s,x));
            continue;
        }
        st.push(mp(s,x));
        dfs(x,s);
        low[s]=min(low[s],low[x]);
        if(dis[s]<=low[x] and par!=-1)
        {
            while(st.top().first!=s or
                st.top().second!=x)
            {
                cout<<st.top().first<<'
                    '<<st.top().second<<endl;
                st.pop();
            }
            cout<<s<<' '<<x<<endl;
            st.pop();
        }
        child++;
        if(child>1 and par==-1)
        {
            while(st.top().first!=s or
                st.top().second!=x)
            {
                cout<<st.top().first<<'
                    '<<st.top().second<<endl;
                st.pop();
            }
            cout<<s<<' '<<x<<endl;
            st.pop();
        }
    }
}
int  main()
{
    ///print biconnected component
        v[1].pb(0);
        v[0].pb(1);
        dfs(1,-1);
        line;
        while(st.size())
        {
            cout<<st.top().first<<'
                '<<st.top().second<<endl;
            st.pop();
        }
}
```

# 6 DP

## 6.1 Digit Dp

```
vector<ll >v;
ll dp[16][2][180];
ll call(ll ind, bool check,ll sum)
{
    if(ind==-1) return sum;
    ll cnt=0;
    ll limit=check? (v[ind]):9;
    ll &ret=dp[ind][check][sum];
    if(ret!=-1 and !check) return ret;
    for(ll i=0;i<=limit;i++)
    {
        bool ck=(i==v[ind])? check:0;
        cnt+=call(ind-1,ck,sum+i);
    }
    if(!check) ret=cnt; ///is used so that we can
        call this function without memset for another
        query
    return cnt;
}
ll  get(ll n)
{
    v.clear();
    while(n)
    {
        v.pb(n%10); n/=10;
    }
    return call((int)v.size()-1,1,0);
}
```

```
int  main()
{
    ll  t;
    cin>>t;
    memset(dp,-1,sizeof dp);
    while(t--){
    ll a,b; ///take string for large number
    cin>>a>>b;
    cout<<get(b)-get(a-1)<<endl;
    }
}
```

## 6.2 Convex Hull(dynamic)

```
const ll mx=200009;
const ll is_query = -(1LL<<62);
struct Line {
    ll m, c;
    mutable function<const Line*()> succ;
    bool operator<(const Line& rhs) const
    {
        if (rhs.c != is_query) return m < rhs.m;
        const Line* s = succ();
        if (!s) return 0;
        ll x = rhs.m;
        return c - s->c < (s->m - m) * x;
    }
};
struct HullDynamic : public multiset<Line> // will
    maintain upper hull for maximum
{
    bool bad(iterator y)
    {
        auto z = next(y);
        if (y == begin())
        {
            if (z == end()) return 0;
            return y->m == z->m && y->c <= z->c;
        }
        auto x = prev(y);
        if (z == end()) return y->m == x->m && y->c <=
            x->c;
        return 1.0*(x->c - y->c)*(z->m - y->m) >=
            1.0*(y->c - z->c)*(y->m - x->m);
    }
    void addline(ll m, ll c)
    {
        auto y = insert({ m, c });
        y->succ = [=] { return next(y) == end() ? 0 :
            &*next(y); };
        if (bad(y)) { erase(y); return; }
        while (next(y) != end() && bad(next(y)))
            erase(next(y));
        while (y != begin() && bad(prev(y)))
            erase(prev(y));
    }
    ll query(ll x)
    {
        auto l = *lower_bound((Line) { x, is_query });
        return l.m * x + l.c;
    }
}a;
int main()
{
    rules:
        Keeps upper hull for maximums.
add lines with -m and -b and return -ans to
make this code working for minimums.
}
```

## 6.3 Convex Hull(semi offline)

```
ll s[mx],p[mx];
ll ara[mx];
struct cht
{
    vector<ll> m,c;
    bool bad(ll f1, ll f2, ll f3) ///change sign
        depends on slope and upper/lower envelop
    {
        return
            1.0*(c[f3]-c[f1])*(m[f1]-m[f2])>=1.0*(c[f2]-c[f1])*(m[f1]-m[f3]);
    }
    void addline(ll m1,ll c1)
    {
        m.pb(m1);
        c.pb(c1);
        ll sz=m.size();
        while(sz>=3 and bad(sz-3,sz-2,sz-1))
        {
            m.erase(m.end()-2);
            c.erase(c.end()-2);
            sz--;
        }
    }
    ll value(ll ind, ll x)
    {
        return m[ind]*x+c[ind];
    }
    ll query(ll x)
    {
        ll low=0,high=(int)m.size()-1;
        ll ans=-LLONG_MAX;
        while(low<=high)
        {
            int mid1 = low + (high-low)/3;
            int mid2 = high - (high-low)/3;
            if(value(mid1,x)>=value(mid2,x)) ///change
                sign depends on minimum/maximum value
            {
                ans=value(mid1,x);
                high=mid2-1;
            }
            else
            {
                ans=value(mid2,x);
                low=mid1+1;
            }
        }
        return ans;
    }
}a;
int main()
{
    ll n;
    cin>>n;
    for(ll i=1;i<=n;i++)
    {
        cin>>ara[i];
        s[i]=s[i-1]+ara[i];
        p[i]=p[i-1]+i*ara[i];
    }
    //a.addline(0,0); if query occurs before adding
        line
```

```
    ll ans=0;
    for(int i=1;i<=n;i++)
    {
        a.addline(-i,i*s[i-1]-p[i-1]-s[i-1]);
        ans=max(ans,a.query(s[i])+p[i]+s[i]);
    }
    cout<<ans<<endl;
}
rules:
bool bad part:
-------------
slope decrease and find minimum: <=
slope decrease and find maximum: >=
-----------------------------------
//slope increase hole sign ulta hobe
-----------------------------------
slope increase and find minimum: >=
slope increase and find maximum: <=

query part:
find minimum:
    >>>>>>>>>>>>>>>>>>ans<<<<<<<<<<<<<<<<<<
find maximum:
    <<<<<<<<<<<<<<<<<<ans>>>>>>>>>>>>>>>>>>
```

### 6.4 Subset DP

```
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)];
}
```

## 7 String

### 7.1 Aho Corasick

```
const int mx=1e5+5;
ll node=1;
ll nxt[mx][68];
ll link[mx];
vector<int> mark[mx];
ll ans[mx];
ll id;
void add(string &pat)
{
       ll now=0;
       for(auto x: pat)
       {
           ll d=x-'A';
           if(nxt[now][d]==-1)
           {
               link[node]=0;
               mark[node].clear();
               nxt[now][d]=node++;
           }
           now=nxt[now][d];
       }
       mark[now].pb(id);
}
void aho()
{
    queue<ll> q;
    ll root=0;
    q.push(root);
    while(q.size())
    {
        ll a=q.front();
        q.pop();
        for(ll i=0;i<60;i++)
        {
            ll d=nxt[a][i];
            if(d!=-1)
            {
                if(a==root) link[d]=root;
                else
                {
                    link[d]=nxt[link[a]][i];
                    ll e=nxt[link[a]][i];
                    for(auto x: mark[e]) mark[d].pb(x);
                    /// add all pattern no to node d
                        which ends to its link node
                }
                q.push(d);
            }
            else
            {
                if(a==root) nxt[a][i]=root;
                else nxt[a][i]=nxt[link[a]][i];
            }
        }
    }
}
void match(string &txt)
{
   ll now=0;
   memset(ans,0,sizeof ans);
   for(ll i=0;i<txt.size();i++)
   {
     ll d=txt[i]-'A';
     now=nxt[now][d];
     for(auto x: mark[now]) ans[x]=true;
    }
}
int main()
{
    ///find whether the pattern is in the text of not
    ///time complexity(total length of (pattern
        +text) + no of occurrences)
    int t;
    cin>>t;
    while(t--)
    {
      node=1;
    memset(nxt,-1,sizeof nxt);
    int n;
    string pat,txt;
    cin>>txt;
    cin>>n;
    for(ll i=0;i<n;i++)
    {
        cin>>pat;
        id=i+1;
        add(pat);
    }
    aho() ;
    match(txt);
    for(int i=1;i<=n;i++)
    {
        if(ans[i]) cout<<"y"<<endl;
        else cout<<"n"<<endl ;
```

```
    }
    }
}
```

## 7.2 Hashing

```
ll  p[mx],p1[mx];
void pre()
{
    p[0]=1; p1[0]=1;
    for(ll  i=1;i<mx;i++)
    {
        p[i]=(p[i-1]*13331LL)%mod;
        p1[i]=(p1[i-1]*23333LL)%mod1;
    }
}
struct Hash
{
    ll h[mx],h1[mx];
    ll  n;
    void init(string &str)
    {
        n=str.size();
         h[0]=0; h1[0]=0;
        for(ll  i=1;i<=n;i++)
        {
            h[i]=h[i-1]+(str[i-1]-'a'+1)*p[i];
                h[i]%=mod;
            h1[i]=h1[i-1]+(str[i-1]-'a'+1)*p1[i];
                h1[i]%=mod1;
        }
    }
    pair<ll ,ll > value(ll l,ll r)
    {
        ll  a=h[r]-h[l-1]; a=(a*p[n-l])%mod;
            a=(a+mod)%mod;
        ll  b=h1[r]-h1[l-1];b=(b*p1[n-l])%mod1;
            b=(b+mod1)%mod1;
        return {a,b};
    }
}ss,pp;
int main()
{
   pre();
   string str;
   cin>>str;

}
```

## 7.3 KMP

```
string str1,pat;
int ara[2*50009]; ///don't need to memset before
    every case
int pre()
{
    int now=0;
    ara[0]=now;
    for(int i=1;i<pat.size();i++)
    {
        while(now and pat[now]!=pat[i]) now=ara[now-1];
        if(pat[now]==pat[i]) now++;
        ara[i]=now;
    }
}
bool match()
{
    int now=0;
    for(int i=0;i<str1.size();i++)
    {
        while(now and str1[i]!=pat[now])
            now=ara[now-1];
        if(str1[i]==pat[now]) now++;
        else now=0;
        if(now==pat.size()) return 1;
    }
    return 0;
}
```

## 7.4 Manacher

```
int d[2][mx];
string str;
//d[0][i]=total number of odd length palindromes
    whose centre is i
//d[1][i]=total number of even length palindromes
    whose centre is i
void pre()
{
    int n=(int)str.size();
    for(int j=0;j<2;j++)
    {
        for(int i=0,l=0,r=-1;i<n;i++)
        {
            int k=(i>r)?(!j):min(d[j][l+r-i+j],r-i+1);
            while(i-k-j>=0 and i+k<n and
                str[i-k-j]==str[i+k]) k++;
            d[j][i]=k--;
            if(i+k>r) l=i-k-j,r=i+k;
        }
    }
}
```

## 7.5 Suffix Array

```
struct SA
{
    vector<int> sa, lcp;
    SA(){};
    void build(string& s, int lim = 256)
    {
        int n = s.size() + 1, k = 0, a, b;
        vector<int> x(s.begin(), s.end() + 1), y(n),
            ws(max(n, lim)), rank(n);
        sa = lcp = y;
        iota(sa.begin(), sa.end(), 0);

        for (int j = 0, p = 0; p < n; j = max(1, j *
            2), lim = p)
        {
            p = j, iota(y.begin(), y.end(), n - j);
            for (int i = 0; i <= n - 1; i++)
            {
                if (sa[i] >= j) y[p++] = sa[i] - j;
            }
            fill(ws.begin(), ws.end(), 0);
            for (int i = 0; i <= n - 1; i++)
                ws[x[i]]++;
```

```
        for (int i = 1; i <= lim - 1; i++) ws[i]
            += ws[i - 1];
        for (int i = n; i--;) sa[--ws[x[y[i]]]] =
            y[i];
        swap(x, y), p = 1, x[sa[0]] = 0;
        for (int i = 1; i <= n - 1; i++)
        {
            a = sa[i - 1], b = sa[i];
            x[b] = (y[a] == y[b] && y[a + j] == y[b
                + j]) ? p - 1 : p++;
        }
    }
    for (int i = 1; i <= n - 1; i++) rank[sa[i]] =
        i;
    for (int i = 0, j; i < n - 1; lcp[rank[i++]] =
        k)
    {
        for (k && k--, j = sa[rank[i] - 1]; s[i +
            k] == s[j + k]; k++);
    }
    for(int i=1;i<n-1;i++) lcp[i]=lcp[i+1];
    lcp[n-1]=0;
  }
}sa;
int main()
{
    fast;
    string str;
    cin>>str;
    sa.build(str);
    ll n=(int)str.size();
    long long ans=n*(n+1)/2;
    for(auto x: sa.lcp) ans-=x;
    cout<<ans<<endl;
}
```

## 7.6 Suffix Automata

```
int nxt[mx<<1][26],link[mx<<1],len[mx<<1];
int last=1,sz=1;
void add(int c)
{
    int cur=++sz;
    int p=last; last=cur;
    len[cur]=len[p]+1;
    while(p and nxt[p][c]==0)
        nxt[p][c]=cur,p=link[p]; //add character c to
        the suffix of p
    if(p==0) link[cur]=1;
    else
    {
        int q=nxt[p][c];
        if(len[q]==len[p]+1) link[cur]=q; //continuous
            transition , so don't change anything
        else
        {
            int cl=++sz; ///non continuous transition
                , so make clone of q
            len[cl]=len[p]+1;
            link[cl]=link[q];
            for(int i=0;i<26;i++) nxt[cl][i]=nxt[q][i];
            while(p and nxt[p][c]==q)
                nxt[p][c]=cl,p=link[p];
            link[cur]=link[q]=cl;
        }
    }
}
int main()
{
    string str;
    cin>>str;
    for(auto x: str) add(x-'a');
    cout<<sz<<endl;
}
```

## 7.7 Trie

```
int nxt[100009*26][26];
bool mark[100009*26];
int node=1;
void add(string str)
{
    int now=0;
    for(int i=0;i<str.size();i++)
    {
        int d=str[i]-'a';
        if(nxt[now][d]==0) nxt[now][d]=node++;
        now=nxt[now][d];
    }
    mark[now]=true;
}
bool check(string str)
{
    int now=0;
    for(int i=0;i<str.size();i++)
    {
        int d=str[i]-'a';
        if(nxt[now][d]==0) return false;
        now=nxt[now][d];
    }
    return mark[now] ;
}
int main()
{
    string str;
    cin>>str;
    add(str);
    string str1;
    cin>>str1;
    if(check(str1)) printf("found");
    else printf("not found");
}
```

## 7.8 Persistent Trie

```
struct edge
{
    ll l,r;
    edge()
    {
        l=0,r=0;
    }
}nxt[mx*50];
int root[mx*50];
ll cnt=1;
void init(ll k)
{
    ll now=1;
    for(ll i=31;i>=0;i--)
    {
        bool d=k&(1<<i);
        if(d==0) nxt[now].l=++cnt;
```

```
        else nxt[now].r=++cnt;
        now=cnt;
    }
}
void add(ll k, ll pre)
{
    ll now=++cnt;
    for(ll i=31;i>=0;i--)
    {
        bool d=k&(1<<i);
        if(d==0)
        {
            nxt[now].l=++cnt;
            nxt[now].r=nxt[pre].r;
            pre=nxt[pre].l;
        }
        else
        {
            nxt[now].r=++cnt;
            nxt[now].l=nxt[pre].l;
            pre=nxt[pre].r;
        }
        now=cnt;
    }
}
int main()
{
    int n;
    cin>>n;
    int r,val;
    cin>>r>>val;
    init(val);
    root[r]=1;
    for(int i=0;i<n;i++)
    {
        int a,b,val;
        cin>>a>>b>>val; //add edge a to b, value of b
            is val
        root[b]=cnt+1;
        add(val,root[a]);
    }
}
```

## 7.9 Palindromic Tree

```
int nxt[mx][26],len[mx],link[mx];
int node;
int last; ///node number which contains longest
    palindromic substring of current prefix
string str;
void pre()
{
    len[1]=-1,len[2]=0;
    link[1]=link[2]=1;
    node=last=2;
}
void add(int p)
{
    while(str[p-len[last]-1]!=str[p]) last=link[last];
    int x=link[last];
    while(str[p-len[x]-1]!=str[p]) x=link[x]; ///find
        the suffix link of current prefix
    int c=str[p]-'a';
    if(nxt[last][c]==false)
    {
        nxt[last][c]=++node;
        len[node]=len[last]+2;
        link[node]=(len[node]==1)? 2: nxt[x][c];
            ///nxt[x][c] is already calculated
    }
    last=nxt[last][c] ;
}
```

## 7.10 Z Algorithm

```
string str,str1 ;
ll ara[200009];///2*sizeof string, need to memset for
    every case
void findz()
{
    ll left=0,right=0;
    for(ll i=1;i<str.size();i++)
    {
        if(right>=i) ara[i]=min(ara[i-left],right-i+1);
        while(i+ara[i]<str.size() and
            str[ara[i]]==str[i+ara[i]]) ara[i]++;
        if(i+ara[i]-1>right) left=i,right=i+ara[i]-1;
    }
}
int  main()
{
    ///minimum length palindrome by adding character
        to the end
    cin>>str1;
    str=str1;
    reverse(str.begin(),str.end());
    str=str+'#'+str1;
    findz();
    ll d=0 ;
    for( ll i=0;i<str.size();i++)
    {
        if(ara[i]+i==str.size())
        {
            d=ara[i];
            break;
        }
    }
    cout<<str1;
    for(ll i=d;str[i]!='#';i++ )
    {
        cout<<str[i];
    }
}
```