Team notebook

IUT_ReverseHash

October 7, 2022

Contents

1 01 Templates

1.1 101 Nafis Template

```
/*
Check and remove this section while coding
1. Get rid of toolbars except compiler and
    main. Enable only logs and status.
2. Use C++17 in global compiler settings.
3. Turn on Wall, Wextra, Wshadow in warnings.
4. Make tab spout 4 spaces
5. Settings -> Compiler -> Linker Settings ->
    Other Linker Options:
    -Wl,--stack,268435456
6. Settings -> Environment -> General
    Settings -> Terminal to launch console
    programs -> select gnome
7. Do you have graph paper with you?
*/

/*
ID: nafis.f1
TASK:
LANG: C++
*/
#include<bits/stdc++.h>
using namespace std;
typedef pair<int, int> pii;
typedef long long LL;
typedef pair<LL, LL> pLL;
#define ff first
#define ss second
#define show(x) cout << #x << ": " << x << ";
    "
#define INF 1000000000000000015
#define MOD 1000000007
#define MAXN 2000006

void solve(int caseno)
{

}

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

        int T = 1, caseno = 0;

        cin >> T;

        while(T--)
        {
                solve(++caseno);
        }
}
```

1.2 201 Herok default

```
/*---------------Go Code GO---------------*/
#include "bits/stdc++.h"
#include "ext/pb_ds/assoc_container.hpp"
#include "ext/pb_ds/tree_policy.hpp"
#ifdef BUG
#include "bits/error.h"
#else
#define debug(x...)
#define endl '\n'
#endif
#pragma GCC target("sse4.2")
#pragma GCC target("avx2")
#pragma GCC optimization("O3")
#pragma GCC optimization("unroll-loops")
using namespace std;
using namespace __gnu_pbds;
template <typename T>
using Order_Set = tree<T, null_type, less<T>,
    rb_tree_tag,
    tree_order_statistics_node_update>;
//(order) Set.order_of_key(); (pointer)
    Set.find_by_order();
int main()
{
#ifdef BUG
   freopen("in.txt", "r", stdin);
#endif
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    debug("HI");
}
```

2 02 Math

2.1 102 Derangements

```
// !n = (n-1) (!(n-1) + !(n-2))
LL D[MAXN];
LL getDerange(LL n)
{
    if(n == 1)
        return D[1] = 0;
    if(n == 2)
        return D[2] = 1;
    if(D[n] != 0)
        return D[n];

    return (n-1) * (getDerange(n-1) +
        getDerange(n-2));
}
```

2.2 301 FFT

```
using CD = complex<double>;
typedef long long LL;
const double PI = acos(-1.0L);

int N;
vector<int> perm;
vector<CD> wp[2];
void precalculate(int n) {
    assert((n & (n - 1)) == 0), N = n;
    perm = vector<int>(N, 0);
    for (int k = 1; k < N; k <<= 1) {
        for (int i = 0; i < k; i++) {
            perm[i] <<= 1;
            perm[i + k] = 1 + perm[i];
        }
    }
    wp[0] = wp[1] = vector<CD>(N);
    for (int i = 0; i < N; i++) {
        wp[0][i] = CD(cos(2 * PI * i / N),
            sin(2 * PI * i / N));
        wp[1][i] = CD(cos(2 * PI * i / N),
            -sin(2 * PI * i / N));
    }
}
void fft(vector<CD> &v, bool invert = false) {
    if (v.size() != perm.size())
        precalculate(v.size());
    for (int i = 0; i < N; i++)
        if (i < perm[i]) swap(v[i],
            v[perm[i]]);
    for (int len = 2; len <= N; len *= 2) {
        for (int i = 0, d = N / len; i < N; i
            += len) {
            for (int j = 0, idx = 0; j < len /
                2; j++, idx += d) {
                CD x = v[i + j];
                CD y = wp[invert][idx] * v[i +
                    j + len / 2];
                v[i + j] = x + y;
                v[i + j + len / 2] = x - y;
            }
        }
    }
    if (invert) {
        for (int i = 0; i < N; i++) v[i] /= N;
    }
}
void pairfft(vector<CD> &a, vector<CD> &b,
    bool invert = false) {
    int N = a.size();
    vector<CD> p(N);
    for (int i = 0; i < N; i++) p[i] = a[i] +
        b[i] * CD(0, 1);
    fft(p, invert);
    p.push_back(p[0]);
    for (int i = 0; i < N; i++) {
        if (invert) {
            a[i] = CD(p[i].real(), 0);
            b[i] = CD(p[i].imag(), 0);
        } else {
            a[i] = (p[i] + conj(p[N - i])) *
                CD(0.5, 0);
            b[i] = (p[i] - conj(p[N - i])) *
                CD(0, -0.5);
        }
    }
}
vector<LL> multiply(const vector<LL> &a,
    const vector<LL> &b) {
    int n = 1;
    while (n < a.size() + b.size()) n <<= 1;
    vector<CD> fa(a.begin(), a.end()),
        fb(b.begin(), b.end());
    fa.resize(n);
    fb.resize(n);
    //       fft(fa); fft(fb);
    pairfft(fa, fb);
    for (int i = 0; i < n; i++) fa[i] = fa[i]
        * fb[i];
    fft(fa, true);
    vector<LL> ans(n);
    for (int i = 0; i < n; i++) ans[i] =
        round(fa[i].real());
    return ans;
}
const int M = 1e9 + 7, B = sqrt(M) + 1;
vector<LL> anyMod(const vector<LL> &a, const
    vector<LL> &b) {
    int n = 1;
    while (n < a.size() + b.size()) n <<= 1;
    vector<CD> al(n), ar(n), bl(n), br(n);
    for (int i = 0; i < a.size(); i++)
        al[i] = a[i] % M / B, ar[i] = a[i] % M
            % B;
    for (int i = 0; i < b.size(); i++)
        bl[i] = b[i] % M / B, br[i] = b[i] % M
            % B;
    pairfft(al, ar);
    pairfft(bl, br);
    //       fft(al); fft(ar); fft(bl);
        fft(br);
    for (int i = 0; i < n; i++) {
        CD ll = (al[i] * bl[i]), lr = (al[i] *
            br[i]);
        CD rl = (ar[i] * bl[i]), rr = (ar[i] *
            br[i]);
        al[i] = ll;
        ar[i] = lr;
        bl[i] = rl;
        br[i] = rr;
    }
    pairfft(al, ar, true);
    pairfft(bl, br, true);
    //       fft(al, true); fft(ar, true);
        fft(bl, true); fft(br, true);
    vector<LL> ans(n);
    for (int i = 0; i < n; i++) {
        LL right = round(br[i].real()), left =
            round(al[i].real());
        ;
        LL mid = round(round(bl[i].real()) +
            round(ar[i].real()));
        ans[i] = ((left % M) * B * B + (mid %
            M) * B + right) % M;
    }
    return ans;
}
```

3 03 Number Theory

3.1 101 Primality Test

```
// Easy BruteForce
bool isPrime(LL n)
{
    LL i;
    for(i = 2; i*i <= n; i++)
    {
        if(n%i == 0)
            return 0;
    }

    return 1;
}

// Fermat's Primality Test, vulnerable to
    Charmichael numbers
bool isPrime2(LL n)
{
    vector<LL> checkerPrimes = {2, 3, 5, 7};
    if(binary_search(checkerPrimes.begin(),
        checkerPrimes.end(), n) == 1)
        return 1;

    vector<LL> carmichael =
        {561,1105,1729,2465,2821,6601,8911,10585,15841,
                                29341,41041,46657,52633,62745,63973,75361,101101,
                                115921,126217,162401,172081,188461,252601,278545,
                                294409,314821,334153,340561,399001,410041,449065,
                                488881,512461};

    if(binary_search(carmichael.begin(),
        carmichael.end(), n) == 1)
        return 0;

    for(auto cp : checkerPrimes)
    {
        if(bigmod(cp, n, n) != cp%n)
            return 0;
    }

    return 1;
}
```

3.2 102 Mulmod

```
LL mulmod(LL a, LL b, LL mod)
{
    if(b == 0)
        return 0;
    LL res = mulmod(a, b>>1, mod);
    res = (res<<1)%mod;
    if(b&1)
        return (res+a)%mod;
    else
        return res;
}

LL mulmod2(LL a, LL b, LL mod)
{
    LL ret = 0;

    while(b)
    {
        if(b&1)
            ret += a;

        a = a*2;
        b /= 2;
    }
    return ret;
}
```

3.3 103 Bigmod

```
LL bigmod(LL a, LL p, LL MOD)
{
    if(p == 0)
        return 1%MOD;
    if(p == 1)
        return a%MOD;
    LL res = bigmod(a, p/2, MOD);
    res = (res*res)%MOD;
    if(p&1)
        return (a*res)%MOD;
    return res;
}

LL bigmod2(LL a, LL p, LL MOD)
{
    LL res = 1%MOD;
    while(p)
    {
        if(p&1)
            res = (res*a)%MOD;
        a = (a*a)%MOD;
        p /= 2;
    }
    return res;
}
```

3.4 104 Invmod

```
LL inv[MAXN];

//only if mod is prime and gcd(a, mod) = 1
LL invmod(LL a, LL mod)
{
    return bigmod(a, mod-2, mod);
}

LL egcd(LL a, LL m, LL& x, LL& y)
{
    if(m == 0)
    {
        x = 1;
        y = 0;
        return a;
    }

    LL x1, y1;
    LL d = egcd(m, a%m, x1, y1);

    x = y1;
```

```
    y = x1 - y1*(a/m);

    return d;
}

//when gcd(a, mod) = 1
LL invod2(LL a, LL mod)
{
    LL x, y;
    egcd(a, mod, x, y);

    return (x%mod + mod) % mod;
}

//when N is prime
void allinvmod()
{
    LL i;
    inv[1] = 1;
    for(i = 2; i < N; i++)
        inv[i] = ((-N/i*inv[N%i]) % N + N) % N;
}
```

3.5 105 nCr

```
/*
We need actual value assuming answer fits in
    long long
    1. O(r): Multiply by (n-i) and divide by
        i, in each step. C(n, r) = C(n-1,
        r-1)*n/r

Prime mod M
    2. O(n): Precalculate factorial and
        inverse factorial array.
       O(1): Answer each query from these
           arrays
    3. O(M): Use Lucas Theorem

Non-prime mod M
    4. O(n*n): Use Pascal's Triangle
    5. O(M): Use Chinese Remainder Theorem
*/

LL fact[2000006];
LL inv[2000006];
LL dp[500][500];

LL getFact(LL n, LL mod);
LL bigmod(LL a, LL p, LL mod);
LL invmod(LL a, LL mod);

LL nCr1(LL n, LL r)
{
    if(n < r)
        return 0;

    r = min(r, n-r);

    if(r == 0)
        return 1;

    return n * nCr1(n-1, r-1) / r;
}

LL nCr2(LL n, LL r, LL mod)
{
    if(n < r)
        return 0;

    LL ret = getFact(n, mod);
    ret = (ret * invmod(getFact(r, mod), mod))
        % mod;
    ret = (ret * invmod(getFact(n-r, mod),
        mod)) % mod;
    return ret;
}

LL nCr3(LL n, LL r, LL mod)
{
    if(n < r)
        return 0;

    LL ret = 1;
    while(r)
    {
        ret = (ret * nCr2(n%mod, r%mod))%mod;
        n /= mod;
        r /= mod;
    }

    return ret;
}

LL nCr4(LL n, LL r, LL mod)
{
    if(n < r)
        return 0;

    if(dp[n][r] != 0)
        return dp[n][r];

    if(!r)
        return dp[n][r] = 1;

    return dp[n][r] = (nCr4(n-1, r-1, mod) +
        nCr4(n-1, r, mod)) % mod;
}

LL nCr5(LL n, LL r, LL mod)
{
    return -1;
}
```

3.6 106 Sieve of Eratosthenes

```
bool flag[MAXN];
vector<LL> primes;

void sieve()
{
    LL i, j;

    flag[2] = 1;
    for(i = 3; i < MAXN; i += 2)
        flag[i] = 1;

    for(i = 3; i*i < MAXN; i += 2)
    {
        if(flag[i])
        {
```

```
            for(j = i*i; j < MAXN; j += 2*i)
                flag[j] = 0;
        }
    }

    for(i = 2; i < MAXN; i++)
    {
        if(flag[i])
            primes.push_back(i);
    }
}
```

3.7 107 Linear Sieve

```
LL leastFactor[MAX];
bool isComposite[MAX];
vector<LL> primes;

void linSieve()
{
    LL i, j;
    for (i = 2; i < MAX; ++i)
    {
        if (leastFactor[i] == 0)
        {
            leastFactor[i] = i;
            primes.push_back(i);
        }
        for (j = 0; j < (LL)primes.size() &&
             primes[j] <= leastFactor[i] &&
             i*primes[j] < MAXN; ++j)
        {
            leastFactor[i * primes[j]] =
                primes[j];
        }
    }
}

void linSieve2()
{
    LL i, j;
    for(i = 2; i < N; i++)
    {
        if (!isComposite[i])
            primes.push_back(i);

        for(j = 0; j < (LL)primes.size() &&
             i*primes[j] < MAXN; j++)
        {
            isComposite[i * primes[j]] = 1;
            if(i % primes[j] == 0)
                break;
        }
    }
}
```

3.8 108 Number of Divisiors (sqrt)

```
bool flag[MAXN];
vector<LL> primes;

void sieve() {}

LL NOD(LL n)
{
    LL i, c, ret = 1;

    for(i = 0; primes[i]*primes[i] <= n; i++)
    {
        for(c = 0; n % primes[i] == 0; c++)
            n /= primes[i];
        ret *= (c+1);
    }

    if(n > 1)
        ret = ret << 1;

    return ret;
}
```

3.9 109 Sum of Divisors

```
bool flag[MAXN];
LL leastFactor[MAXN];
vector<LL> primes;
LL sod[MAXN];

void sieve() {}

LL linSOD(LL n)
{
    LL lf, c, p, ret = 1;

    while(n > 1)
    {
        lf = leastFactor[n];
        p = 1;

        for(c = 0; n%lf == 0; c++)
        {
            n /= lf;
            p *= lf;
        }

        ret *= (p*lf - 1)/(lf - 1);
    }

    return ret;
}

LL SOD(LL n)
{
    LL i, c, ret = 1;

    for(i = 0; primes[i]*primes[i] <= n; i++)
    {
        LL p = 1;
        for(c = 0; n % primes[i] == 0; c++)
        {
            n /= primes[i];
            p = p * primes[i];
        }
        ret *= (p * primes[i] - 1) /
             (primes[i] - 1);
    }

    if(n > 1)
        ret *= (n*n - 1) / (n - 1);
    return ret;
```

```
}

void allSOD()
{
    LL i, j;

    for(i = 1; i < MAXN; i++)
    {
        for(j = i; j < MAXN; j += i)
            sod[j] += i;
    }
}
```

3.10 110 Euler's Totient

```
bool flag[MAXN];
vector<int> primes;
ll phi[MAXN];

void sieve() {}

ll findPhi(ll n)
{
    if(phi[n] != 0)
        return phi[n];

    ll i, cnt, ret = n, temp = n;
    for(i = 0; primes[i] * primes[i] <= n; i++)
    {
        for(cnt = 0; n % primes[i] == 0; cnt++)
            n /= primes[i];

        if(cnt > 0)
            ret = ret / primes[i] * (primes[i]
                - 1);
    }

    if(n > 1)
        ret = ret / n * (n - 1);

    return phi[temp] = ret;
}
```

```
void sievephi()
{
    ll i, j;
    for(i = 1; i < MAXN; i++)
        phi[i] = i;

    for(i = 2; i < MAXN; i++)
    {
        if(phi[i] == i)
        {
            for(j = i; j < MAXN; j += i)
                phi[j] = phi[j] / i * (i - 1);
        }
    }
}

void segsievephi(ll a, ll b)
{
    ll i, j, cnt;

    for(i = a; i <= b; i++)
    {
        phi[i-a] = i;
        val[i-a] = i;
    }

    for(auto p : primes)
    {
        if(p * p > b)
            break;

        for(i = (a + p - 1) / p * p; i <= b; i
            += p)
        {
            for(cnt = 0; val[i - a] % p == 0;
                cnt++)
                val[i - a] /= p;

            if(cnt)
                phi[i - a] = phi[i - a] / p *
                    (p - 1);
        }
    }

    for(i = a; i <= b; i++)
    {
        if(val[i - a] > 1)
            phi[i - a] = phi[i - a] / val[i -
                a] * (val[i - a] - 1);
    }
}
```

3.11 111 Extended GCD

```
LL egcd(LL a, LL b, LL& x, LL& y)
{
    if (b == 0)
    {
        x = 1;
        y = 0;
        return a;
    }
    LL x1, y1;
    LL d = egcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}

LL egcd2(LL a, LL b, LL& x, LL& y)
{
    x = 1, y = 0;
    LL x1 = 0, y1 = 1, a1 = a, b1 = b;
    while (b1)
    {
        LL q = a1 / b1;
        tie(x, x1) = make_tuple(x1, x - q *
            x1);
        tie(y, y1) = make_tuple(y1, y - q *
            y1);
        tie(a1, b1) = make_tuple(b1, a1 - q *
            b1);
    }
    return a1;
}

// Linear Diophantine Equation
// a*x + b*y = c
```

```
bool LDE(LL a, LL b, LL c, LL &x0, LL &y0, LL
    &d)
{
    d = egcd(abs(a), abs(b), x0, y0);
    if (c % d)
        return 0;

    x0 *= c / d;
    y0 *= c / d;
    x0 = (a < 0? -1 : 1) * x0;
    y0 = (b < 0? -1 : 1) * y0;
    return 1;
}

bool LDEall(LL a, LL b, LL c, LL t, LL &x, LL
    &y)
{
    LL d;
    if(LDE(a, b, c, x, y, d))
    {
        x = x + b*t;
        y = y - a*t;
        return 1;
    }

    return 0;
}
```

3.12 112 Matrix Exponentiation

```
typedef vector<vector<LL>> Mat;

Mat Mul(Mat A, Mat B)
{
    Mat ret(A.size(), vector<LL>(B[0].size()));
    LL i, j, k;

    for(i = 0; i < ret.size(); i++)
    {
        for(j = 0; j < ret[0].size(); j++)
        {
            for(k = 0; k < A[0].size(); k++)
                ret[i][j] = (ret[i][j] +
                    (A[i][k]*B[k][j])%MOD)%MOD;
        }
    }

    return ret;
}

Mat Pow(Mat A, LL p)
{
    Mat ret(A.size(), vector<LL>(A[0].size()));

    for(LL i = 0; i < ret.size(); i++)
        ret[i][i] = 1;

    while(p)
    {
        if(p&1)
            ret = Mul(ret, A);
        A = Mul(A, A);
        p >>= 1;
    }
    return ret;
}
```

3.13 113 Shanks' Baby Step, Giant Step

```
// Finds a^x = b (mod p)

LL bigmod(LL b, LL p, LL m) {}

LL babyStepGiantStep(LL a, LL b, LL p)
{
    LL i, j, c, sq = sqrt(p);
    map<LL, LL> babyTable;

    for(j = 0, c = 1; j <= sq; j++, c =
        (c*a)%p)
        babyTable[c] = j;

    LL giant = bigmod(a, sq*(p-2), p);

    for(i = 0, c = 1; i <= sq; i++, c =
        (c*giant)%p)
    {
        if(babyTable.find((c*b)%p) !=
            babyTable.end())
            return i*sq+babyTable[(c*b)%p];
    }

    return -1;
}
```

3.14 301 Combi

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

const int M = 1e9+7;

int power(int a, int p) {
    if (p == 0) return 1;
    int ans = power(a, p/2);
    ans = (1LL*ans * ans)%M;
    if (p%2)   ans = (1LL*ans*a)%M;
    return ans;
}

const int N = 2e5+7;
int fac[N], invfac[N];
void pre() {
    fac[0] = 1;
    for(int i=1; i<N; i++) fac[i] =
        (1LL*i*fac[i-1])%M;
    invfac[N-1] = power(fac[N-1], M-2);
    for (int i=N-2; i>=0; i--) invfac[i] =
        (1LL*invfac[i+1]*(i+1))%M;
}

int C(int n, int r) {
    if (r<0 || r>n) return 0;
    int denom = (1LL*invfac[r]*invfac[n-r])%M;
    return (1LL*fac[n]*denom)%M;
}
```

3.15 302 Linear Sieve(Phi+SOD+NOD)

```
const int maxn = 1e7;
vector <int> primes;
int spf[maxn+5], phi[maxn+5], NOD[maxn+5],
    cnt[maxn+5], POW[maxn+5];
bool prime[maxn+5];
int SOD[maxn+5];
void init(){
    fill(prime+2, prime+maxn+1, 1);
    SOD[1] = NOD[1] = phi[1] = spf[1] = 1;
    for(LL i=2;i<=maxn;i++){
        if(prime[i]) {
            primes.push_back(i), spf[i] = i;
            phi[i] = i-1;
            NOD[i] = 2, cnt[i] = 1;
            SOD[i] = i+1, POW[i] = i;
        }
        for(auto p:primes){
            if(p*i>maxn or p > spf[i]) break;
            prime[p*i] = false, spf[p*i] = p;
            if(i%p == 0){
                phi[p*i]=p*phi[i];
                NOD[p*i]=NOD[i]/(cnt[i]+1)*(cnt[i]+2),
                    cnt[p*i]=cnt[i]+1;
                SOD[p*i]=SOD[i]/SOD[POW[i]]*(SOD[POW[i]]+p*POW[i]),POW[p*i]=p*POW[i];
                break;
            } else {
                phi[p*i]=phi[p]*phi[i];
                NOD[p*i]=NOD[p]*NOD[i],
                    cnt[p*i]=1;
                SOD[p*i]=SOD[p]*SOD[i],
                    POW[p*i]=p;
            }

        }
    }
}
```

3.16 303 Pollard Rho

```
ULL mul(ULL a,ULL b,ULL mod){
    LL ans = a * b - mod * (ULL) (1.L / mod *
        a * b);
    return ans + mod * (ans < 0) - mod * (ans
        >= (LL) mod);
}
ULL bigmod(ULL num,ULL pow,ULL mod){
    ULL ans = 1;
    for( ; pow > 0; pow >>= 1, num = mul(num,
        num, mod))
        if(pow&1) ans = mul(ans,num,mod);
    return ans;
}
bool is_prime(ULL n){
    if(n < 2 or n % 6 % 4 != 1)
        return (n|1) == 3;
    ULL a[] = {2, 325, 9375, 28178, 450775,
        9780504, 1795265022};
    ULL s = __builtin_ctzll(n-1), d = n >> s;
    for(ULL x: a){
        ULL p = bigmod(x % n, d, n), i = s;
        for( ; p != 1 and p != n-1 and x % n
            and i--; p = mul(p, p, n));
        if(p != n-1 and i != s)
            return false;
    }
    return true;
}
ULL get_factor(ULL n) {
    auto f = [&](LL x) { return mul(x, x, n) +
        1; };
    ULL x = 0, y = 0, t = 0, prod = 2, i = 2,
        q;
    for( ; t++ %40 or gcd(prod, n) == 1; x =
        f(x), y = f(f(y)) ){
        (x == y) ? x = i++, y = f(x) : 0;
        prod = (q = mul(prod, max(x,y) -
            min(x,y), n)) ? q : prod;
    }
    return gcd(prod, n);
}
map <ULL, int> factorize(ULL n){
    map <ULL, int> res;
    if(n < 2)  return res;
    ULL small_primes[] = {2, 3, 5, 7, 11, 13,
        17, 19, 23, 29, 31, 37, 41, 43, 47,
        53, 59, 61, 67, 71, 73, 79, 83, 89, 97
        };
    for (ULL p: small_primes)
        for( ; n % p == 0; n /= p, res[p]++);

    auto _factor = [&](ULL n, auto &_factor) {
        if(n == 1)  return;
        if(is_prime(n))
            res[n]++;
        else {
            ULL x = get_factor(n);
            _factor(x, _factor);
            _factor(n / x, _factor);
        }
    };
    _factor(n, _factor);
    return res;
}
```

4 04 DP

5 05 Graph

5.1 101 Cycle Finding

```
LL n;
vector<LL> adj[MAXN];
LL color[MAXN], parent[MAXN];
LL cycle_start, cycle_end;

bool dfs(LL cur, LL p)
{
    parent[cur] = p;
    color[cur] = 1;

    for (LL nxt : adj[cur])
    {
        if (color[nxt] == 0)
        {
            if (dfs(nxt, cur))
                return true;
        }
```

```
        else if (color[nxt] == 1)
        {
            cycle_end = cur;
            cycle_start = nxt;
            return true;
        }
    }
    color[cur] = 2;
    return false;
}

void find_cycle()
{
    fill(color, color+n, 0);
    fill(parent, parent+n, -1);
    cycle_start = -1;

    LL i;

    for (i = 0; i < n; i++)
    {
        if (color[i] == 0 && dfs(i, -1))
            break;
    }

    if (cycle_start == -1)
        cout << "Acyclic" << "\n";
    else
    {
        vector<LL> cycle;
        cycle.push_back(cycle_start);
        for (LL v = cycle_end; v !=
             cycle_start; v = parent[v])
            cycle.push_back(v);
        cycle.push_back(cycle_start);
        reverse(cycle.begin(), cycle.end());

        cout << "Cycle found: ";
        for (LL v : cycle)
            cout << v << " ";
        cout << "\n";
    }
}
```

5.2 102 0-1 BFS

```
vector<pLL> adj[MAXN];
LL dist[MAXN];

void bfs01(LL cur)
{
    LL nxt, d;
    dequeu<pLL> dq;
    dq.push_back({0, cur});
    dist[cur] = 0;

    while(!dq.empty())
    {
        d = dq.front().ff;
        nxt = dq.front().ss;
        dq.pop_front();

        if(dist[nxt] < d)
            continue;

        for(auto [nxt, d2] : adj[cur])
        {
            if(dist[nxt] == -1 || d + d2 <
                dist[nxt])
            {
                dist[nxt] = d + d2;

                if(d2)
                    dq.push_back({dist[nxt],
                        nxt});
                else
                    dq.push_front({dist[nxt],
                        nxt});
            }
        }
    }
}
```

5.3 103 Dijkstra

```
LL n;
vector <pLL> adj[MAXN];
LL dis[MAXN], par[MAXN];

void dijkstra(LL s)
{
    for(LL i = 1; i <= n; i++)
    {
        dis[i] = INF;
        par[i] = -1;
    }

    set<pLL> q;

    dis[s] = 0;
    q.insert({0, s});

    while(!q.empty())
    {
        pLL p = *q.begin();
        q.erase(q.begin());

        LL node = p.ss;
        if(p.ff > dis[node])
            continue;

        for (auto u : adj[node])
        {
            LL len = u.ff;
            LL to = u.ss;
            if (dis[node] + len < dis[to])
            {
                dis[to] = dis[node] + len;
                q.insert({dis[to], to});
                par[to] = node;
            }
        }
    }
}
```

5.4 104 Bellman Ford

```
struct edge
{
    int a, b, cost;
```

```
};

int n, m, v;
vector<edge> e;

void solve()
{
    vector<int> d (n, INF);
    vector<int> p (n, -1);
    d[v] = 0;
    bool any = 1;
    while(any)
    {
        any = 0;

        for (int j=0; j<m; j++)
        {
            if (d[e[j].a] < INF)
            {
                if (d[e[j].b] > d[e[j].a] +
                    e[j].cost)
                {
                    d[e[j].b] = d[e[j].a] +
                        e[j].cost;
                    p[e[j].b] = e[j].a;
                    any = 1;
                }
            }
        }
    }

    if (d[t] == INF)
        cout << "No path from " << v << " to "
            << t << ".";
    else
    {
        vector<int> path;
        for (int cur = t; cur != -1; cur =
            p[cur])
            path.push_back (cur);
        reverse (path.begin(), path.end());

        cout << "Path from " << v << " to " <<
            t << ": ";
        for (size_t i = 0; i < path.size();
            i++)
            cout << path[i] << ' ';
    }
}
```

5.5 105 Floyd warshall

```
LL n;
vector<pLL> adj[MAXN];
LL dis[MAXN][MAXN];

void floyd_warshall()
{
    LL i, j, k;

    memset(dist, -1, sizeof dist);
    for(i = 1; i <= n; i++)
    {
        dist[i][i] = 0;

        for(auto u : adj[i])
        {
            if(dist[i][u.ss] == -1)
                dist[i][u.ss] = dist[u.ss][i] =
                    c;
            else
                dist[i][u.ss] = dist[u.ss][i] =
                    min(dist[i][u.ss], u.ff);
        }
    }

    for(k = 1; k <= n; k++)
    {
        for(i = 1; i <= n; i++)
        {
            for(j = 1; j <= n; j++)
            {
                if(dist[i][k] != -1 &&
                    dist[k][j] != -1)
                {
                    if(dist[i][j] == -1)
                        dist[i][j] = dist[i][k]
                            + dist[k][j];
                    else
                        dist[i][j] =
                            min(dist[i][j],
                            dist[i][k] +
                            dist[k][j]);
                }
            }
        }
    }
}
```

5.6 106 Kruskal

```
LL leader[MAXN], n, m;
vector<pair<LL, pLL>>>edges;

LL Find(LL a)
{
    if(leader[a] == a)
        return a;
    leader[a] = Find(leader[a]);
    return leader[a];
}

void Union(LL a, LL b)
{
    a = Find(a);
    b = Find(b);

    leader[max(a, b)] = min(a, b);
}

LL kruskal()
{
    LL i, ret = 0;
    for(i = 1; i <= n; i++)
        leader[i] = i;

    sort(edges.begin(), edges.end());
    LL cnt = 0;
    for(auto [c, e] : edge)
```

```
    {
        [a, b] = e;

        if(Find(a) != Find(b))
        {
            Union(a, b);
            ret += c;
            cnt++;
            if(cnt == n-1)
                return ret;
        }
    }

    return -1;
}
```

5.7 107 Topsort with DFS

```
vector<int> adj[MAX];
stack<int> st;
int col[MAX];

bool dfs(int s)
{
    int ret = 1;
    col[s] = 1;
    for(auto u : adj[s])
    {
        if(col[u] == 0)
            ret = ret & dfs(u);
        else if(col[u] == 1)
            return 0;
    }
    col[s] = 2;
    st.push(s);
    return ret;
}

//Run it for all nodes
for(i = 1; i <= node; i++)
{
    if(col[i] == 0 && dfs(i) == 0)
    {
        cout << "impossible";
        return 0;
    }
}
```

5.8 108 TopSort with Indegree

```
int n;
vector<int> adj[MAXN];
vector<int> path;
int in[MAXN];

void topsort()
{
    queue<int> Q;
    int i;
    for(i = 1; i <= n; i++)
    {
        if(in[i] == 0)
            Q.push(i);
    }
    while(!Q.empty())
    {
        int node = Q.front();
        Q.pop();
        path.push_back(node);
        for(auto u : adj[node])
        {
            in[u]--;
            if(in[u] == 0)
                Q.push(u);
        }
    }
}

int main()
{
    int i, m;
    cin >> n >> m;
    for(i = 0; i < m; i++)
    {
        int u,v;
        cin >> u >> v;
        adj[u].push_back(v);
        in[v]++;
    }

    topsort();

    if(path.size() != n)
        cout << "impossible";
    else
    {
        for(i = 0; i < path.size(); i++)
            cout << path[i] << " ";
    }

}
```

5.9 109 Articulation Point

```
int n, m;
bool vis[MAXN];
int tin[MAXN], low[MAXN], timer;
vector<int> adj[MAXN];
set<int> AP;

void dfs(int v, int p = -1)
{
    vis[v] = 1;
    timer++;
    tin[v] = low[v] = timer;

    int child = 0;
    for(auto to : adj[v])
    {
        if(to == p)
            continue;
        if(!vis[to])
        {
            child++;
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if(low[to] >= tin[v] && p != -1)
                AP.insert(v);
        }
```

```
        else
            low[v] = min(low[v], tin[to]);
    }

    if(p == -1 && child > 1)
        AP.insert(v);
}

void findAP()
{
    AP.clear();
    timer = 0;
    int i;
    for(i = 1; i <= n; i++)
    {
        vis[i] = 0;
        tin[i] = -1;
        low[i] = -1;
    }

    for(i = 1; i <= n; i++)
    {
        if(!vis[i])
            dfs(i);
    }
}
```

5.10 110 Bridge

```
int n, m;
bool vis[MAXN];
int tin[MAXN], low[MAXN], timer;
vector<int> adj[MAXN];
vector<int> bridge[MAXN];

void dfs(int v, int p = -1)
{
    vis[v] = 1;
    timer++;
    tin[v] = low[v] = timer;

    int child = 0;
    for(auto to : adj[v])
    {
        if(to == p)
            continue;
        if(!vis[to])
        {
            child++;
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if(low[to] > tin[v])
            {
                bridge[v].push_back(to);
                bridge[to].push_back(v);
            }
        }
        else
            low[v] = min(low[v], tin[to]);
    }
}

void findBR()
{
    bridge.clear();
    timer = 0;
    int i;
    for(i = 1; i <= n; i++)
    {
        vis[i] = 0;
        tin[i] = -1;
        low[i] = -1;
    }

    for(i = 1; i <= n; i++)
    {
        if(!vis[i])
            dfs(i);
    }
}
```

5.11 111 Centroid Decomposition

```
int n;
vector <int> adj[MAXN];
int subtree_size[MAXN];

int get_subtree_size(int node, int par = -1)
{
        int ret = 1;

        for (auto next : adj[node])
    {
                if (next != par)
            ret += get_subtree_size(next, node);
        }
        return subtree_size[node] = ret;
}

int get_centroid(int node, int par = -1)
{
        for (auto next : adj[node])
    {
                if (next != par &&
                    subtree_size[next] * 2 > n)
                    return
                        get_centroid(next,
                        node);
        }
        return node;
}
```

5.12 112 Strongly Connected Components

```
vector <int> g[MAXN], gr[MAXN];
bool vis[MAXN];
vector<int> order, component;

void dfs1(int v)
{
    vis[v] = 1;
    for (auto u : g[v])
    {
        if (!vis[u])
            dfs1(u);
    }
    order.push_back (v);
}
```

```
void dfs2(int v)
{
    vis[v] = 1;
    component.push_back(v);
    for (auto u : gr[v])
    {
        if(!vis[u])
            dfs2(u);
    }
}

int main()
{
    int n, m, i, cnt = 0;
    cin >> n >> m;
    for (i = 0; i < m; i++)
    {
        int a, b;
        cin >> a >> b;
        g[a].push_back (b);
        gr[b].push_back (a);
    }

    memset(vis, 0, sizeof vis);
    for (i = 1; i <= n; i++)
    {
        if (!vis[i])
            dfs1(i);
    }

    memset(vis, 0, sizeof vis);
    for (i = 1; i <= n; i++)
    {
        int v = order[n-i];
        if (!vis[v])
        {
            dfs2 (v);
            cout << "Component No. " << ++cnt
                 << ": ";
            for(auto u : component)
                cout << u << " ";
            cout << endl;
            component.clear();
        }
    }
}
```

5.13 301 LCA In O(1)

```
/*
  * LCA in O(1)
  * depth calculates weighted distance
  * level calculates distance by number of
      edges
  * Preprocessing in NlongN
*/

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

typedef long long LL;
typedef pair<int, int> PII;

const int N = 1e6 + 7;
const int L = 21;


namespace LCA {
LL depth[N];
int level[N];

int st[N], en[N], LOG[N], par[N];
int a[N], id[N], table[L][N];

vector<PII> adj[N];
int n, root, Time, cur;

void init(int nodes, int root_) {
    n = nodes, root = root_, LOG[0] = LOG[1] =
        0;
    for (int i = 2; i <= n; i++) LOG[i] =
        LOG[i >> 1] + 1;
    for (int i = 0; i <= n; i++)
        adj[i].clear();
}

void addEdge(int u, int v, int w) {
    adj[u].push_back(PII(v, w));
    adj[v].push_back(PII(u, 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 = LOG[id[v] - id[u] + 1];
    int p1 = id[u], p2 = id[v] - (1 << l) + 1;
    int d1 = level[table[l][p1]], d2 =
        level[table[l][p2]];

    if (d1 < d2) return par[table[l][p1]];
    else return par[table[l][p2]];
}

LL dist(int u, int v) {
    int l = lca(u, v);
    return (depth[u] + depth[v] - (depth[l] *
        2));
}

/* Euler tour */
void dfs(int u, int p) {
    st[u] = ++Time, par[u] = p;

    for (auto [v, w] : adj[u]) {
        if (v == p) continue;
        depth[v] = depth[u] + w;
        level[v] = level[u] + 1;
        dfs(v, u);
    }

    en[u] = ++Time;
    a[++cur] = u, id[u] = cur;
}

/* RMQ */
void pre() {
    cur = Time = 0, dfs(root, root);
    for (int i = 1; i <= n; i++) table[0][i] =
        a[i];
```

```
    for (int l = 0; l < L - 1; l++) {
        for (int i = 1; i <= n; i++) {
            table[l + 1][i] = table[l][i];

            bool C1 = (1 << l) + i <= n;
            bool C2 = level[table[l][i + (1 <<
                l)]] < level[table[l][i]];

            if (C1 && C2) table[l + 1][i] =
                table[l][i + (1 << l)];
        }
    }
}

}
/* namespace LCA */
//tested on kattis-greatestpair

using namespace LCA;

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

}
```

5.14 302 LCA(Binary Lifting)

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

const int N = 3e5 + 7, K = 20;
vector<int> adj[N];

int anc[N][K];
int level[N];

void dfs(int u, int par) {
    level[u] = level[par] + 1;
    anc[u][0] = par;
    for (int k = 1; k < K; k++) anc[u][k] =
        anc[anc[u][k - 1]][k - 1];
    for (int v : adj[u]) {
        if (v == par) continue;
        dfs(v, u);
    }
}

int lca(int u, int v) {
    if (level[u] > level[v]) swap(u, v);
    for (int k = K - 1; k >= 0; k--)
        if (level[u] + (1 << k) <= level[v]) v
            = anc[v][k];

    if (u == v) return u;
    for (int k = K - 1; k >= 0; k--)
        if (anc[u][k] != anc[v][k]) u =
            anc[u][k], v = anc[v][k];
    return anc[u][0];
}

int getanc(int u, int d) {
    for (int k = 0; k < K; k++)
        if (d & (1 << k)) u = anc[u][k];
    return u;
}

int dist(int u, int v) {
    int g = lca(u, v);
    return level[u] + level[v] - 2 * level[g];
}


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

    int n;
    cin >> n;

    for (int i = 1; i < n; i++) {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    dfs(1, 0);

}
```

5.15 303 MCMF

```
/**
Min Cost Max Flow Using Successive Shortest
    Path
Complexity: SPFA: O(ans*VE) ,
    Dijkstra: O(ans*ElogV) + Cost of
        Normalization
    Normalization: Sets potentials (pi) for
        Dijkstra()
        If all edges >= 0: you may comment out
            normalize()
        If graph is DAG, use DP in O(m)
        Otherwise, SPFA() is used in O(mn)

Author: anachor
**/
#include<bits/stdc++.h>
using namespace std;


namespace MCMF {
    typedef long long F; typedef long long C;
    const F infF = 1e18; const C infC = 1e18;

    const int N = 5005;
    typedef pair<C, F> PCF;

    struct Edge {int frm, to; C cost; F cap,
        flow;};

    int n, s, t;
    vector<Edge> edges;
    vector<int> adj[N];
    C pi[N], dis[N];
    F fl[N];
    int prv[N], vis[N];
```

```
void init(int nodes, int source, int sink)
    {
    n = nodes, s = source, t = sink;
    for (int i=0; i<n; i++) pi[i] = 0,
        adj[i].clear();
    edges.clear();
}

void addEdge(int u, int v, F cap,C cost) {
    edges.push_back({u, v, cost, cap, 0});
    edges.push_back({v, u, -cost, 0, 0});
    adj[u].push_back(edges.size()-2);
    adj[v].push_back(edges.size()-1);
}

bool SPFA() {
    for (int i=0; i<n; i++) {
        dis[i] = infC; fl[i] = 0;
        vis[i] = 0; prv[i] = -1;
    }
    queue<int> q;
    q.push(s);
    dis[s] = 0; fl[s] = infF; vis[s] = 1;

    while (!q.empty()) {
        int u = q.front(); q.pop();
        vis[u] = 0;
        for (int eid : adj[u]) {
            Edge &e = edges[eid];
            if (e.cap == e.flow) continue;

            if (dis[u] + e.cost <
                dis[e.to]) {
                dis[e.to] = dis[u] + e.cost;
                fl[e.to] = min(fl[u], e.cap
                    - e.flow);
                prv[e.to] = eid^1;
                if (!vis[e.to])
                    q.push(e.to);
            }
        }
    }
    return fl[t] > 0;
}
```

```
PCF solveSPFA() {
    C cost = 0; F flow = 0;
    while (SPFA()) {
        C pathcost = dis[t];
        cost += pathcost*fl[t]; flow +=
            fl[t];
        for (int u=t, e=prv[u]; e!=-1;
            u=edges[e].to, e=prv[u]) {
            edges[e].flow -= fl[t];
            edges[e^1].flow += fl[t];
        }
    }
    return {cost, flow};
}

void normalize() {
    SPFA();
    for (int i=0; i<n; i++) pi[i] = dis[i];
}

bool Dijkstra() {
    for (int i=0; i<n; i++) {
        dis[i] = infC; fl[i] = 0;
        vis[i] = 0; prv[i] = -1;
    }
    priority_queue<pair<C, int>> pq;
    pq.emplace(0, s);
    dis[s] = 0; fl[s] = infF;

    while (!pq.empty()) {
        int u = pq.top().second; pq.pop();
        if (vis[u]) continue;
        vis[u] = 1;

        for (int eid : adj[u]) {
            Edge &e = edges[eid];
            if (vis[e.to] || e.cap ==
                e.flow) continue;

            C nw = dis[u] + e.cost -
                pi[e.to] + pi[u];
            if (nw < dis[e.to]) {
                dis[e.to] = nw;
                fl[e.to] = min(fl[u], e.cap
                    - e.flow);
                prv[e.to] = eid^1;
                pq.emplace(-dis[e.to],
                    e.to);
            }
        }
    }
    return fl[t] > 0;
}

PCF solveDijkstra() {
    normalize();
    C cost = 0; F flow = 0;
    while (Dijkstra()) {
        for (int i=0; i<n; i++)
            if (fl[i])   pi[i] += dis[i];
        C pathcost = pi[t]-pi[s];
        cost += pathcost*fl[t]; flow +=
            fl[t];

        for (int u=t, e=prv[u]; e!=-1;
            u=edges[e].to, e=prv[u]) {
            edges[e].flow -= fl[t];
            edges[e^1].flow += fl[t];
        }
    }
    return {cost, flow};
}
}

/// More tests:
    https://docs.google.com/spreadsheets/d/1NMolWZsOU
///
    https://open.kattis.com/problems/mincostmaxflow
int main(){
    int n, m, s, t;
    cin>>n>>m>>s>>t;

    MCMF::init(n+1, s, t);

    for (int i=0; i<m; i++) {
        int a, b, c, w;
        cin>>a>>b>>c>>w;
        MCMF::addEdge(a, b, c, w);
    }
```

```
    auto [c, f] = MCMF::solveDijkstra();
    cout<<f<<" "<<c<<endl;
}
```

5.16 304 SCC

```
typedef long long LL;
const LL N = 1e6 + 7;

bool vis[N];
vector<int> adj[N], adjr[N];
vector<int> order, component;
// tp = 0 ,finding topo order, tp = 1 ,
    reverse edge traversal

void dfs(int u, int tp = 0) {
    vis[u] = true;
    if (tp) component.push_back(u);
    auto& ad = (tp ? adjr : adj);
    for (int v : ad[u])
        if (!vis[v]) dfs(v, tp);
    if (!tp) order.push_back(u);
}
int main() {
    for (int i = 1; i <= n; i++) {
        if (!vis[i]) dfs(i);
    }
    memset(vis, 0, sizeof vis);
    reverse(order.begin(), order.end());
    for (int i : order) {
        if (!vis[i]) {
            // one component is found
            dfs(i, 1), component.clear();
        }
    }
}
```

5.17 305 Dinic

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

struct FlowEdge {
    int v, u;
    long long cap, flow = 0;
    FlowEdge(int v, int u, long long cap) :
        v(v), u(u), cap(cap) {}
};

struct Dinic {
    const long long flow_inf = 1e18;
    vector<FlowEdge> edg;
    vector<vector<int>> adj;

    int n, m = 0, s, t;
    vector<int> lvl, ptr;
    queue<int> q;

    // number of nodes, source, sink
    Dinic(int n, int s, int t) : n(n), s(s),
        t(t) {
        adj.resize(n + 100), lvl.resize(n +
            100), ptr.resize(n + 100);
    }

    //directed edge from v to u (not u to v)
    void add_edge(int v, int u, long long cap)
        {
        edg.emplace_back(v, u, cap);
        edg.emplace_back(u, v, 0);
        adj[v].push_back(m++);
        adj[u].push_back(m++);
    }

    bool bfs() {
        while (!q.empty()) {
            int v = q.front(); q.pop();
            for (int id : adj[v]) {
                if (edg[id].cap - edg[id].flow
                    < 1) continue;
                if (lvl[edg[id].u] != -1)
                    continue;
                lvl[edg[id].u] = lvl[v] + 1,
                    q.push(edg[id].u);
            }
        }
        return lvl[t] != -1;
    }

    long long dfs(int v, long long pushed) {
        if (pushed == 0) return 0;
        if (v == t) return pushed;
        for (int& cid = ptr[v]; cid <
            (int)adj[v].size(); cid++) {
            int id = adj[v][cid], u = edg[id].u;
            if (lvl[v] + 1 != lvl[u] ||
                edg[id].cap - edg[id].flow <
                1) continue;
            long long tr = dfs(u, min(pushed,
                edg[id].cap - edg[id].flow));
            if (tr == 0) continue;
            edg[id].flow += tr, edg[id ^
                1].flow -= tr;
            return tr;
        }
        return 0;
    }

    long long flow() {
        long long f = 0;
        while (true) {
            fill(lvl.begin(), lvl.end(), -1);
            lvl[s] = 0, q.push(s);
            if (!bfs()) break;
            fill(ptr.begin(), ptr.end(), 0);
            while (long long pushed = dfs(s,
                flow_inf)) f += pushed;
        }
        return f;
    }
};
/*
 **directed edge from v to u (not u to v)
 ** to find matching edges go over all edges
     where s!=u && s !=v && t!=u && t!=v and
     find flow of 1
*/
```

5.18 306 HLD

```
const int N = 1e6+7;
template <typename DT>
struct Segtree {
    vector<DT> tree, prob, a;
    Segtree(int n) {
        tree.resize(n * 4);
        prob.resize(n), a.resize(n);
    }
    void build(int u, int l, int r) {
        if (l == r) {
            tree[u] = a[l];
            return;
        }
        int mid = (l + r) / 2;
        build(u * 2, l, mid);
        build(u * 2 + 1, mid + 1, r);
        tree[u] = (tree[u * 2] + tree[u * 2 +
            1]);
    }
    void propagate(int u) {
        prob[u * 2] += prob[u], tree[u * 2] +=
            prob[u];
        prob[u * 2 + 1] += prob[u], tree[u * 2
            + 1] += prob[u];
        prob[u] = 0;
    }
    void update(int u, int l, int r, int i,
         int j, int val) {
        if (r < i || l > j) return;
        if (l >= i && r <= j) {
            tree[u] = val;
            return;
        }
        int mid = (l + r) / 2;
        update(u * 2, l, mid, i, j, val);
        update(u * 2 + 1, mid + 1, r, i, j,
            val);
        tree[u] = (tree[u * 2] + tree[u * 2 +
            1]);
    }
    DT query(int u, int l, int r, int i, int
         j) {
        if (l > j || r < i) return 0;
        if (l >= i && r <= j) return tree[u];
        int mid = (l + r) / 2;
        return (query(u * 2, l, mid, i, j) +
            query(u * 2 + 1, mid + 1, r, i,
            j));
    }
};
Segtree<int>tree(N);
vector<int> adj[N];
int depth[N], par[N], pos[N];
int head[N], heavy[N], cnt;

int dfs(int u, int p) {
    int SZ = 1, mxsz = 0, heavyc;
    depth[u] = depth[p] + 1;

    for (auto v : adj[u]) {
        if (v == p) continue;
        par[v] = u;
        int subsz = dfs(v, u);
        if (subsz > mxsz) heavy[u] = v, mxsz =
            subsz;
        SZ += subsz;
    }
    return SZ;
}
void decompose(int u, int h) {
    head[u] = h, pos[u] = ++cnt;
    if(heavy[u]!=-1) decompose(heavy[u], h);

    for(int v : adj[u]) {
        if(v==par[u]) continue;
        if(v!=heavy[u]) decompose(v, v);
    }
}
int query(int a, int b) {
    int ret = 0;
    for(;head[a]!=head[b]; b=par[head[b]]){
       if(depth[head[a]]>depth[head[b]])
            swap(a,b);
       ret += tree.query(1, 0, cnt,
            pos[head[b]], pos[b]);
    }

    if(depth[a]>depth[b]) swap(a,b);
    ret += tree.query(1,0,cnt,pos[a],pos[b]);
    return ret;
}
```

5.19 307 Bridge Tree

```
class Bridge_Tree {
    int n;
    vector <vector <int>> components;
    vector <int> depth, low;
    stack <int> st;
    void find_bridges(int node, Graph &G, int
         par = -1, int d = 0) {
        low[node] = depth[node] = d;
        st.push(node);
        for (int e : G[node]) if(par != e) {
            if(depth[e] == -1) {
                find_bridges(e, G, node, d + 1);
                if (low[e] > depth[node]){
                    bridges.emplace_back(node,
                        e);
                    components.push_back({});
                    for(int x = -1; x!= e; x =
                        st.top(), st.pop())
                       components.back().push_back(st.t
                }
            }
            low[node] = min(low[node], low[e]);
        }
        if(par == -1){
            components.push_back({});
            while(!st.empty())
                components.back().push_back(st.top()),
                    st.pop();
        }
    }
public:
    vector <int> id;
    vector <edge> bridges;
    Graph tree;
    void create_tree() {
        for(auto &comp : components){
            int idx = tree.add_node();
            for(auto &e: comp)
```

```
                id[e] = idx;
        }
        for(auto &[l,r]: bridges)
            tree.add_edge(id[l], id[r]);
    }
    Bridge_Tree(Graph &G): n(G.n) {
        depth.assign(n,-1), id.assign(n, -1),
            low.resize(n);
        for(int i = 0; i < n; i++)
            if(depth[i] == -1)
                find_bridges(i, G);
    }
};
```

6 06 Tree

6.1 101 Least Common Ancestor

```
// Important Note: parent of 1 is 1
LL n;
vector<LL> adj[MAXN];
LL parent[MAXN], level[MAXN], anc[MAXN][21];

void dfs(LL cur, LL p, LL l)
{
    parent[cur] = p;
    level[cur] = l;

    for(auto nxt : adj[cur])
    {
        if(nxt != parent[cur])
            dfs(nxt, cur, l+1);
    }
}

void LCA_init()
{
    dfs(1, 1, 0);

    LL i, j;
    for(i = 1; i <= n; i++)
        anc[i][0] = parent[i];

    for(j = 1; j < 21; j++)
    {
        for(i = 1; i <= n; i++)
            anc[i][j] = anc[anc[i][j-1]][j-1];
    }
}

LL getLCA(LL u, LL v)
{
    if(level[u] < level[v])
        swap(u, v);

    LL i;
    for(i = 20; i >= 0; i--)
    {
        if(level[anc[u][i]] >= level[v])
            u = anc[u][i];
    }

    if(u == v)
        return u;

    for(i = 20; i >= 0; i--)
    {
        if(anc[u][i] != anc[v][i])
        {
            u = anc[u][i];
            v = anc[v][i];
        }
    }

    return parent[u];
}

// LCA_init()
// cout << getLCA(5, 8) << "\n";
```

6.2 102 Heavy Light Decomposition

```
int n, a[MAXN];
vector<int> adj[MAXN];

int parent[MAXN], level[MAXN],
    anc[MAXN][21];//for lca
int heavy[MAXN], subsize[MAXN];
int chainHead[MAXN], chainNo, basePos[MAXN],
    chainIdx[MAXN];//for hld
int base[MAXN], cnt, Tree[MAXN*4];//for
    segment Tree

int dfs(int node, int pr, int l)
{
    subsize[node] = 1;
    parent[node] = pr;
    level[node] = l;

    int mx = 0, x;

    for(auto u : adj[node])
    {
        if(u != parent[node])
        {
            x = dfs(u, node, l+1);
            subsize[node] += x;

            if(mx < x)
            {
                heavy[node] = u;
                mx = x;
            }
        }
    }

    return subsize[node];
}

void lca_init()
{
    int i, j;
    for(i = 1; i <= n; i++)
    {
        for(j = 0; j <= 20; j++)
            anc[i][j] = 1;
    }

    for(i = 1; i <= n; i++)
        anc[i][0] = parent[i];
```

```
    for(j = 1; (1<<j) <= n; j++)
    {
        for(i = 1; i <= n; i++)
            anc[i][j] = anc[anc[i][j-1]][j-1];
    }
}

int lca(int u, int v)
{
    if(level[u] < level[v])
        swap(u, v);

    int i;
    for(i = log2(n) + 1; i >= 0; i--)
    {
        if(level[anc[u][i]] >= level[v])
            u = anc[u][i];
    }

    if(u == v)
        return u;
    for(i = log2(n) + 1; i >= 0; i--)
    {
        if(anc[u][i] != anc[v][i])
        {
            u = anc[u][i];
            v = anc[v][i];
        }
    }
    return parent[u];
}

void hld_init(int u, int pr)
{
        if(chainHead[chainNo] == -1)
        chainHead[chainNo] = u;

        chainIdx[u] = chainNo;
        base[cnt++] = a[u];
        basePos[u] = cnt-1;

        if(heavy[u] > -1)
        hld_init(heavy[u], u);
        for(auto v : adj[u])
        {
                if(v != pr and v != heavy[u])
                {
                        chainNo++;
                        hld_init(v, u);
                }
        }
}

void build_tree(int node, int s, int e)
{
    if(s == e)
    {
        Tree[node] = base[s];
        return;
    }

    int mid = (s+e)/2, left = 2*node, right =
        2*node+1;

    build_tree(left, s, mid);
    build_tree(right, mid+1, e);

    Tree[node]= Tree[left] + Tree[right];
}

void update_tree(int node, int s, int e, int
    pos, int val)
{
    if(s > pos || e < pos)
        return;
    if(s == e)
    {
        base[s] = val;
        Tree[node] = val;
        return;
    }

    int mid = (s+e)/2, left = 2*node, right =
        2*node+1;

    update_tree(left, s, mid, pos, val);
    update_tree(right, mid+1, e, pos, val);

    Tree[node] = Tree[left] + Tree[right];
}

int query_tree(int node, int s, int e, int
    lo, int hi)
{
    if(hi < s || lo > e)
        return 0;
    if(lo <= s && hi >= e)
        return Tree[node];

    int mid = (s+e)/2, left = 2*node, right =
        2*node+1;
    int p1 = query_tree(left, s, mid, lo, hi);
    int p2 = query_tree(right, mid+1, e, lo,
        hi);

    return p1+p2;
}

int query_up(int u, int p)
{
        int uchain, pchain = chainIdx[p], ret
            = 0;

        while(1)
    {
                uchain = chainIdx[u];

                if(uchain == pchain)
                {
                        ret += query_tree(1, 1,
                            n, basePos[p],
                            basePos[u]);
                        break;
                }
                ret += query_tree(1, 1, n,
                    basePos[chainHead[uchain]],
                    basePos[u]);
                u = chainHead[uchain];
                u = parent[u];
        }
        return ret;
}

void update_hld(int p, int val)
```

```
{
    update_tree(1, 1, n, basePos[p], val);
}

int query_hld(int u, int v)
{
    int l = lca(u, v);
    return query_up(u, l) + query_up(v, l) -
        query_up(l, l);
}

void init()
{
    int i;
    for(i = 0; i <= n; i++)
    {
        heavy[i] = -1;
        chainHead[i] = -1;
    }
    dfs(1, 1, 0);
    lca_init();
    cnt = 1, chainNo = 1;
    hld_init(1, 1);
    build_tree(1, 1, n);
}

int main()
{
    int t, i, q, caseno = 0, u, v;
    scanf("%d", &t);

    while(t--)
    {
        scanf("%d", &n);
        for(i = 1; i <= n; i++)
        {
            scanf("%d", &a[i]);
            adj[i].clear();
        }

        for(i = 1; i < n; i++)
        {
            scanf("%d %d", &u, &v);
            adj[u+1].push_back(v+1);
            adj[v+1].push_back(u+1);
        }

        init();

        printf("Case %d:\n", ++caseno);
        scanf("%d", &q);

        while(q--)
        {
            int type, x, y;
            scanf("%d %d %d", &type, &x, &y);
            if(type)
                update_hld(x+1, y);
            else
                printf("%d\n", query_hld(x+1,
                    y+1));
        }
    }
}
```

6.3 103 Euler Tour on Tree

```
ll idx;
ll tour[MAXN];
vector<ll> adj[MAXN];

void createTourTree(ll node, ll p = -1)
{
    for(auto u : adj[node])
    {
        if(u != p)
            dfs(u, node);
    }

    tour[idx] = node;
    idx++;
}

idx = 0; // must before calling this function
```

7 07 Data Structure

7.1 101 Bitset

```
#pragma GCC optimize("O3,unroll-loops")
#pragma GCC
    target("avx2,bmi,bmi2,lzcnt,popcnt")

bitset<MAXN> Add(bitset<MAXN> a, bitset<MAXN>
    b)
{
    bitset<MAXN> carry;
    while(b != bitset<MAXN>(0))
    {
        carry = a&b;
        a = a^b;
        b = carry << 1;
    }

    return a;
}

bitset<MAXN> Sub(bitset<MAXN> a, bitset<MAXN>
    b)
{
    b = ~b;
    b = Add(b, bitset<MAXN>(1));

    return Add(a, b);
}
```

7.2 102 Disjoint Set Union

```
LL leader[MAXN];

LL Find(LL x)
{
    if (x == leader[x])
        return x;
    return leader[x] = find_set(leader[x]);
}
```

```
void Union(LL x, LL y) {
    x = Find(x);
    y = Find(y);

    leader[max(x, y)] = min(x, y);
}
```

7.3 103 Ordered Set

```
#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>;


ordered_set<LL> OS, OS2;

OS.insert(10);
OS.insert(20);
OS.insert(30);

//Find 2nd smallest element O(log(n))
cout << *(OS.find_by_order(1)) << "\n";

//Counting elements strictly less than 15
    O(log(n))
cout << OS.order_of_key(15) << "\n";

//Check existence of 30
cout << (OS.find(30) == OS.end()) << "\n";

//Delete 30
OS.erase(30);

//Swap two policy based data structure in O(1)
OS.swap(OS2);
```

7.4 104 Sparse Table RMQ

```
LL arr[MAXN], sparse[MAXN][20], Log[MAXN];

LL query(LL bg, LL ed)
{
    LL k = Log[ed-bg+1];
    return min(sparse[bg][k], sparse[ed - (1
        << k) + 1][k]);
}

void rmq_init()
{
    LL n, i, j;

    Log[1] = 0;
    for(i = 2; i < MAXN; i++)
        Log[i] = Log[i/2] + 1;

    cin >> n;
    for(i = 0; i <= n; i++)
        sparse[i][0] = arr[i];

    for(j = 1; j < 20; j++)
    {
        for(i = 0; i + (1LL << j) - 1 < n; i++)
            sparse[i][j] = min(sparse[i][j-1],
                sparse[i + (1LL <<
                (j-1))][j-1]);
    }
}
```

7.5 105 Segment tree

```
LL arr[MAXN];
LL Tree[4*MAXN];

void Build(LL node, LL bg, LL ed)
{
    if(bg == ed)
    {
        Tree[node] = arr[bg];
        return;
    }

    LL leftNode = 2*node, rightNode = 2*node +
        1;
    LL mid = (bg + ed)/2;

    Build(leftNode, bg, mid);
    Build(rightNode, mid+1, ed);

    Tree[node] = Tree[leftNode] +
        Tree[rightNode];
}

void Update(LL node, LL bg, LL ed, LL idx, LL
    val)
{
    if(bg == ed)
    {
        Tree[node] = val;
        arr[idx] = val;
        return;
    }

    LL leftNode = 2*node, rightNode = 2*node +
        1;
    LL mid = (bg + ed)/2;

    if(idx <= mid)
        Update(leftNode, bg, mid, idx, val);
    else
        Update(rightNode, mid+1, ed, idx, val);

    Tree[node] = Tree[leftNode] +
        Tree[rightNode];
}

LL Query(LL node, LL bg, LL ed, LL l, LL r)
{
    if(bg > r || ed < l)
        return 0;

    if(l <= bg && ed <= r)
        return Tree[node];

    LL leftNode = 2*node, rightNode = 2*node +
        1;
```

```
    LL mid = (bg + ed)/2;

    LL p1 = Query(leftNode, bg, mid, l, r);
    LL p2 = Query(rightNode, mid+1, ed, l, r);

    return p1 + p2;
}
```

7.6 106 Segment tree (Lazy)

```
// Node e dhukar sathe sathe lazy clear kora
    lagbe.
// Shei node amar desired range er vitore
    thakuk, othoba na thakuk.
// Naile WA khabi sure.
LL arr[MAXN];
LL Tree[4*MAXN], Lazy[4*MAXN];

void Build(LL node, LL bg, LL ed)
{
    if(bg == ed)
    {
        Lazy[node] = 0;
        Tree[node] = arr[bg];
        return;
    }

    LL leftNode = 2*node, rightNode = 2*node +
        1;
    LL mid = (bg + ed)/2;

    Build(leftNode, bg, mid);
    Build(rightNode, mid+1, ed);

    Tree[node] = Tree[leftNode] +
        Tree[rightNode];
    Lazy[node] = 0;
}

void updateRange(LL node, LL bg, LL ed, LL l,
    LL r, LL val)
{
    LL leftNode = 2*node, rightNode = 2*node +
        1;
    LL mid = (bg + ed)/2;

    if(Lazy[node] != 0)
    {
        Tree[node] += (ed - bg + 1) *
            Lazy[node];
        if(bg != ed)
        {
            Lazy[leftNode] += Lazy[node];
            Lazy[rightNode] += Lazy[node];
        }
        Lazy[node] = 0;
    }

    if(bg > r || ed < l)
        return;

    if(l <= bg && ed <= r)
    {
        Tree[node] += (ed - bg + 1) * val;
        if(bg != ed)
        {
            Lazy[leftNode] += val;
            Lazy[rightNode] += val;
        }
        return;
    }

    updateRange(leftNode, bg, mid, l, r, val);
    updateRange(rightNode, mid+1, ed, l, r,
        val);

    Tree[node] = Tree[leftNode] +
        Tree[rightNode];
}

LL queryRange(LL node, LL bg, LL ed, LL l, LL
    r)
{
    LL leftNode = 2*node, rightNode = 2*node +
        1;
    LL mid = (bg + ed)/2;

    if(Lazy[node] != 0)
    {
        Tree[node] += (ed - bg + 1) *
            Lazy[node];
        if(bg != ed)
        {
            Lazy[leftNode] += Lazy[node];
            Lazy[rightNode] += Lazy[node];
        }
        Lazy[node] = 0;
    }

    if(bg > r || ed < l)
        return 0;

    if(l <= bg && ed <= r)
        return Tree[node];

    LL p1 = queryRange(leftNode, bg, mid, l,
        r);
    LL p2 = queryRange(rightNode, mid + 1, ed,
        l, r);

    return (p1 + p2);
}
```

7.7 107 Trie

```
// Sometimes you need to use Add(0) at first

LL trie[MAXN][2], len[MAXN];
LL id;

void Add(LL x)
{
    LL r = 0;

    for(LL i = 34; i >= 0; i--)
    {
        LL bit = ((x & (1LL << i)) >> i);

        if(trie[r][bit] == 0)
            trie[r][bit] = ++id;
```

```
        r = trie[r][bit];
        len[r]++;
    }
}

void Erase(LL x)
{
    LL r = 0;

    for(LL i = 34; i >= 0; i--)
    {
        LL bit = ((x & (1LL << i)) >> i);

        r = trie[r][bit];
        len[r]--;
    }
}
```

7.8 108 Merge Sort Tree

```
vector<LL>Tree[4*MAXN];
LL arr[MAXN];

vector<LL> merge(vector<LL> v1, vector<LL> v2)
{
    LL i = 0, j = 0;
    vector<LL> ret;

    while(i < v1.size() || j < v2.size())
    {
        if(i == v1.size())
        {
            ret.push_back(v2[j]);
            j++;
        }
        else if(j == v2.size())
        {
            ret.push_back(v1[i]);
            i++;
        }
        else
        {
            if(v1[i] < v2[j])
            {
                ret.push_back(v1[i]);
                i++;
            }
            else
            {
                ret.push_back(v2[j]);
                j++;
            }
        }
    }

    return ret;
}

void Build(LL node, LL bg, LL ed)
{
    if(bg == ed)
    {
        Tree[node].push_back(arr[bg]);
        return;
    }

    LL leftNode = 2*node, rightNode = 2*node +
        1;
    LL mid = (bg+ed)/2;

    Build(leftNode, bg, mid);
    Build(rightNode, mid+1, ed);

    Tree[node] = merge(Tree[leftNode],
        Tree[rightNode]);
}

LL query(LL node, LL bg, LL ed, LL l, LL r,
    LL k)
{
    if(ed < l || bg > r)
        return 0;

    if(l <= bg && ed <= r)
        return upper_bound(Tree[node].begin(),
            Tree[node].end(), k) -
            Tree[node].begin();

    LL leftNode = 2*node, rightNode = 2*node +
        1;
    LL mid = (bg + ed)/2;

    return query(leftNode, bg, mid, l, r, k) +
        query(rightNode, mid+1, ed, l, r, k);
}
```

7.9 109 Square Root Decomposition

```
LL arr[MAXN], blocks[MAXN];
void Init()
{
    LL i, len = sqrt(n);
    for(i = 0; i < n; i++)
        blocks[i/len] += a[i];
}

LL Query(LL l, LL r)
{
    LL ret = 0;
    for(i = l; i <= r;)
    {
        if (i % len == 0 && i + len - 1 <= r)
        {
            sum += b[i / len];
            i += len;
        }
        else
        {
            sum += a[i];
            i++;
        }
    }

    return ret;
}

// ALTERNATE
int sq;
int arr[30004];
pair<pii, int> query[200005];
```

```
int freq[1000006], ans[200005];

bool cmp(pair<pii, int> a, pair<pii, int> b)
{
    if(a.ff.ff/sq != b.ff.ff/sq)
        return a.ff.ff < b.ff.ff;

    return a.ff.ss < b.ff.ss;
}

void solve(int caseno)
{
    int n, i, q, l, r, distinct;

    cin >> n;
    sq = sqrt(n);

    for(i = 1; i <= n; i++)
        cin >> arr[i];

    cin >> q;
    for(i = 0; i < q; i++)
    {
        cin >> query[i].ff.ff >>
            query[i].ff.ss;
        query[i].ss = i;
    }

    sort(query, query + q, cmp);

    distinct = 0;
    for(i = 0, l = 1, r = 0; i < q; i++)
    {
        while(r < query[i].ff.ss)
        {
            r++;
            if(freq[arr[r]] == 0)
                distinct ++;
            freq[arr[r]]++;
        }
        while(l > query[i].ff.ff)
        {
            l--;
            if(freq[arr[l]] == 0)
                distinct ++;
            freq[arr[l]]++;
        }

        while(l < query[i].ff.ff)
        {
            if(freq[arr[l]] == 1)
                distinct --;
            freq[arr[l]] --;

            l++;
        }
        while(r > query[i].ff.ss)
        {
            if(freq[arr[r]] == 1)
                distinct --;
            freq[arr[r]] --;

            r--;
        }

        ans[query[i].ss] = distinct;
    }

    for(i = 0; i < q; i++)
        cout << ans[i] << "\n";
}
```

7.10 110 Binary Indexed Tree

```
// Always 1 indexed
LL n;
LL a[MAXN], BIT[MAXN];

void Update(LL i, LL d)
{
    while(i <= n)
    {
        BIT[i] += d;
        i += i & (-i);
    }
}

LL Query(LL i)
{
    LL sum = 0;
    while(i > 0)
    {
        sum += BIT[i];
        i -= i & (-i);
    }

    return sum;
}

// Add 7 in position 0
Update(0, 7);
// Add 20 in position 8
Update(8, 20);

cout << "Sum of First 10 elements: " <<
    Query(10) << "\n";
cout << "Sum of elements in [2, 7]: " <<
    Query(7) - query(1) << "\n";
```

7.11 111 Binary Indexed Tree 2D

```
LL mx = 100, my = 100;
BIT[mx][my];

void update(LL x, LL y, LL val)
{
    LL y1;
    while (x <= mx)
    {
        y1 = y;
        while (y1 <= my)
        {
            BIT[x][y1] += val;
            y1 += (y1 & -y1);
        }
        x += (x & -x);
    }
}
LL query(LL x, LL y)
{
    LL y1, sum = 0;
```

```
    while(x)
    {
        y1 = y;
        while(y1)
        {
            sum += BIT[x][y1];
            y1 -= y1&(-y1);
        }
        x -= x&(-x);
    }
    return sum;
}
```

7.12 301 Segtree

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

const int N = 1e5 + 7;
int a[N];
LL tr[4 * N];
LL lz[4 * N];

/// 1. Merge left and right
LL combine(LL left, LL right) { return left +
    right; }

/// 2. Push lazy down and merge lazy
void propagate(int u, int st, int en) {
    if (!lz[u]) return;
    tr[u] += (en - st + 1) * lz[u];
    if (st != en) {
        lz[2 * u] += lz[u];
        lz[2 * u + 1] += lz[u];
    }
    lz[u] = 0;
}

void build(int u, int st, int en) {
    if (st == en) {
        tr[u] = a[st]; /// 3. Initialize
        lz[u] = 0;
    } else {
        int mid = (st + en) / 2;
        build(2 * u, st, mid);
        build(2 * u + 1, mid + 1, en);
        tr[u] = combine(tr[2 * u], tr[2 * u +
            1]);
        lz[u] = 0; /// 3. Initialize
    }
}

void update(int u, int st, int en, int l, int
    r, int x) {
    propagate(u, st, en);
    if (r < st || en < l)
        return;
    else if (l <= st && en <= r) {
        lz[u] += x; /// 4. Merge lazy
        propagate(u, st, en);
    } else {
        int mid = (st + en) / 2;
        update(2 * u, st, mid, l, r, x);
        update(2 * u + 1, mid + 1, en, l, r,
            x);
        tr[u] = combine(tr[2 * u], tr[2 * u +
            1]);
    }
}

LL query(int u, int st, int en, int l, int r)
    {
    propagate(u, st, en);
    if (r < st || en < l)
        return 0; /// 5. Proper null value
    else if (l <= st && en <= r)
        return tr[u];
    else {
        int mid = (st + en) / 2;
        return combine(query(2 * u, st, mid,
            l, r),
                       query(2 * u + 1, mid + 1,
                           en, l, r));
    }
}

void debug(int u, int st, int en) {
    cout << "--->" << u << " " << st << " " <<
        en << " " << tr[u] << " "
         << lz[u] << endl;
    if (st == en) return;
    int mid = (st + en) / 2;
    debug(2 * u, st, mid);
    debug(2 * u + 1, mid + 1, en);
}
```

7.13 302 BIT-2D

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

const int N = 1008;
int bit[N][N], n, m;
int a[N][N], q;

void update(int x, int y, int val) {
    for (; x < N; x += -x & x)
        for (int j = y; j < N; j += -j & j)
            bit[x][j] += val;
}

int get(int x, int y) {
    int ans = 0;
    for (; x; x -= x & -x)
        for (int j = y; j; j -= j & -j) ans +=
            bit[x][j];
    return ans;
}

int get(int x1, int y1, int x2, int y2) {
    return get(x2, y2) - get(x1 - 1, y2) -
        get(x2, y1 - 1) + get(x1 - 1, y1 - 1);
}
```

8 08 Geometry

8.1 101 0D Geo

```
#include<bits/stdc++.h>
using namespace std;
#define EPS 1e-9
#define PI 2*acos(0.0)

struct point
{
    double x, y;
    point() { x = y = 0.0; }
    point(double _x, double _y) : x(_x), y(_y)
        {}

    bool operator == (point other) const
    {
        return abs(x - other.x) < EPS && abs(y
            - other.y) < EPS;
    }

    bool operator < (point other) const
    {
        if(abs(x - other.x) > EPS)
            return x < other.x;
        return y < other.y;
    }

    double dist(point p1, point p2)
    {
        return sqrt((p1.x - p2.x)*(p1.x -
            p2.x) + (p1.y - p2.y)*(p1.y -
            p2.y));
    }

    //rotate point p by theta degrees CCW
        w.r.t origin (0, 0)
    point Rotate(point p, double theta)
    {
        double rad = theta * PI / 180;
        return point(p.x*cos(rad) -
            p.y*sin(rad),
                   p.x*sin(rad) +
                       p.y*cos(rad));
    }

};
```

8.2 102 2D Geo

```
#include<bits/stdc++.h>
using namespace std;
#define EPS 1e-9
#define PI 2*acos(0.0)

struct point
{
    double x, y;
    point() { x = y = 0.0; }
    point(double _x, double _y) : x(_x), y(_y)
        {}

    bool operator == (point other) const
    {
        return abs(x - other.x) < EPS && abs(y
            - other.y) < EPS;
    }

    bool operator < (point other) const
    {
        if(abs(x - other.x) > EPS)
            return x < other.x;
        return y < other.y;
    }

    double dist(point p1, point p2)
    {
        return sqrt((p1.x - p2.x)*(p1.x -
            p2.x) + (p1.y - p2.y)*(p1.y -
            p2.y));
    }

    //rotate p by theta degrees CCW w.r.t
        origin (0, 0)
    point Rotate(point p, double theta)
    {
        double rad = theta * PI / 180;
        return point(p.x*cos(rad) -
            p.y*sin(rad),
                   p.x*sin(rad) +
                       p.y*cos(rad));
    }
};
```

```
struct line
{
    //ax + by = c
    double a, b, c;

    //the answer is stored in third parameter
        (pass by reference)
    void pointsToLine(point p1, point p2, line
        &l)
    {
        if(abs(p1.x - p2.x) < EPS)
        {
            l.a = 1;
            l.b = 0;
            l.c = -p1.x;
        }
        else
        {
            double delx, dely;

            delx = p2.x - p1.x;
            dely = p2.y - p1.x;

            l.a = -dely / delx;
            l.b = 1; //we fix the value of b to
                1.0
            l.c = -(p1.x*dely - p1.y*delx) /
                delx;
        }
    }

    bool areParallel(line l1, line l2)
    {
        return (abs(l1.a-l2.a) < EPS) &&
            (abs(l1.b-l2.b) < EPS);
    }

    bool areSame(line l1, line l2)
    {
        return areParallel(l1, l2) &&
            (abs(l1.c - l2.c) < EPS);
    }
};
```

8.3 103 Closest pair

```
#include<bits/stdc++.h>
using namespace std;
long long ClosestPair(vector<pair<int, int>>
    pts)
{
    int n = pts.size();
    sort(pts.begin(), pts.end());
    set<pair<int, int>> s;

    long long best_dist = 1e18;
    int j = 0;
    for (int i = 0; i < n; ++i)
    {
        int d = ceil(sqrt(best_dist));
        while (pts[i].first - pts[j].first >=
            best_dist)
        {
            s.erase({pts[j].second,
                pts[j].first});
            j += 1;
        }

        auto it1 =
            s.lower_bound({pts[i].second - d,
            pts[i].first});
        auto it2 =
            s.upper_bound({pts[i].second + d,
            pts[i].first});

        for (auto it = it1; it != it2; ++it)
        {
            int dx = pts[i].first - it->second;
            int dy = pts[i].second - it->first;
            best_dist = min(best_dist, 1LL * dx
                * dx + 1LL * dy * dy);
        }
        s.insert({pts[i].second,
            pts[i].first});
    }
    return best_dist;
}

int main()
{
    vector<pair<int, int> > v;
    v.push_back({0, 2});
    v.push_back({0, 0});

    cout << ClosestPair(v);
}
```

8.4 104 Convex Hull

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long
#define pii pair<ll, ll>
#define ff first
#define ss second

vector<pii> v;

bool cmp(pii a, pii b)
{
    return a.ff < b.ff || (a.ff == b.ff &&
        a.ss < b.ss);
}

bool clockWise(pii a, pii b, pii c)
{
    return
        a.ff*(b.ss-c.ss)+b.ff*(c.ss-a.ss)+c.ff*(a.ss-b.ss)
        <= 0;
    //being !clockWise and being anticlockWise
        aren't same. look at "<="
}

bool anticlockWise(pii a, pii b, pii c)
{
    return
        a.ff*(b.ss-c.ss)+b.ff*(c.ss-a.ss)+c.ff*(a.ss-b.ss)
        >= 0;
    //being !clockWise and being anticlockWise
        aren't same. look at ">="
}

void convex_hull()
{
    if(v.size() == 1)
        return;

    sort(v.begin(), v.end(), cmp);

    pii p1 = v[0], p2 = v.back();

    vector<pii> up, down;
    up.push_back(p1);
    down.push_back(p1);

    for (ll i = 1; i < (ll)v.size(); i++)
    {
        if (i == v.size() - 1 || clockWise(p1,
            v[i], p2))
        {
            while (up.size() >= 2 &&
                !clockWise(up[up.size()-2],
                up[up.size()-1], v[i]))
                up.pop_back();
            up.push_back(v[i]);
        }
        if (i == v.size() - 1 ||
            anticlockWise(p1, v[i], p2))
        {
            while(down.size() >= 2 &&
                !antiClockWise(down[down.size()-2],
                down[down.size()-1], v[i]))
                down.pop_back();
            down.push_back(v[i]);
        }
    }

    v.clear();
    for (ll i = 0; i < (ll)down.size(); i++)
        v.push_back(down[i]);
    for (ll i = up.size() - 2; i > 0; i--)
        v.push_back(up[i]);
}
```

8.5 105 Line Intersection

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

struct point
{
    ll x, y;
};

bool intersect(point p1, point p2, point p3,
    point p4)
{
    ll a1, b1, c1;
    a1 = p1.y - p2.y;
    b1 = p2.x - p1.x;
    c1 = p2.x*p1.y - p1.x*p2.y;

    ll a2, b2, c2;
    a2 = p3.y - p4.y;
    b2 = p4.x - p3.x;
    c2 = p4.x*p3.y - p3.x*p4.y;

    ll det = a1*b2 - b1*a2;

    if(!det)
        return 0;

    ll px = (b2*c1 - b1*c2);
    ll py = (a1*c2 - a2*c1);

    if(px < min(p1.x*det, p2.x*det) || px >
        max(p1.x*det, p2.x*det) || py <
        min(p1.y*det, p2.y*det) || py >
        max(p1.y*det, p2.y*det))
        return 0;
    if(px < min(p3.x*det, p4.x*det) || px >
        max(p3.x*det, p4.x*det) || py <
        min(p3.y*det, p4.y*det) || py >
        max(p3.y*det, p4.y*det))
        return 0;

    return 1;
}

int main()
{
    point p1{10, 0}, p2{0, 20}, p3{5, 5},
        p4{10009, 10009};
    cout << intersect(p1, p2, p3, p4);
}
```

8.6 301 Circular

```
// Extremely inaccurate for finding near
    touches
// compute intersection of line l with circle
    c
// The intersections are given in order of
    the ray (l.a, l.b)
vector<Point> circleLineIntersection(Circle
    c, Line l) {
    static_assert(is_same<Tf, Ti>::value);
    vector<Point> ret;
    Point b = l.b - l.a, a = l.a - c.o;

    Tf A = dot(b, b), B = dot(a, b);
    Tf C = dot(a, a) - c.r * c.r, D = B*B -
        A*C;
    if (D < -EPS) return ret;

    ret.push_back(l.a + b * (-B - sqrt(D +
        EPS)) / A);
    if (D > EPS)
        ret.push_back(l.a + b * (-B + sqrt(D))
            / A);
    return ret;
}

// signed area of intersection of circle(c.o,
    c.r) &&
// triangle(c.o, s.a, s.b) [cross(a-o, b-o)/2]
Tf circleTriangleIntersectionArea(Circle c,
    Segment s) {
    using Linear::distancePointSegment;
    Tf OA = length(c.o - s.a);
    Tf OB = length(c.o - s.b);

    // sector
    if(dcmp(distancePointSegment(c.o, s) -
        c.r) >= 0)
        return angleBetween(s.a-c.o, s.b-c.o)
            * (c.r * c.r) / 2.0;

    // triangle
    if(dcmp(OA - c.r) <= 0 && dcmp(OB - c.r)
        <= 0)
        return cross(c.o - s.b, s.a - s.b) /
            2.0;

    // three part: (A, a) (a, b) (b, B)
    vector<Point> Sect =
        circleLineIntersection(c, s);
    return circleTriangleIntersectionArea(c,
        Segment(s.a, Sect[0]))
        + circleTriangleIntersectionArea(c,
            Segment(Sect[0], Sect[1]))
        + circleTriangleIntersectionArea(c,
            Segment(Sect[1], s.b));
}

// area of intersecion of circle(c.o, c.r) &&
    simple polyson(p[])
// Tested :
    https://codeforces.com/gym/100204/problem/F
    - Little Mammoth
Tf circlePolyIntersectionArea(Circle c,
    Polygon p) {
    Tf res = 0;
    int n = p.size();
    for(int i = 0; i < n; ++i)
        res +=
            circleTriangleIntersectionArea(c,
            Segment(p[i], p[(i + 1) % n]));
    return abs(res);
}

// locates circle c2 relative to c1
// interior          (d < R - r)       ---->
    -2
// interior tangents (d = R - r)     ----> -1
// concentric      (d = 0)
// secants           (R - r < d < R + r) ---->
    0
```

```
// exterior tangents (d = R + r)      ----> 1
// exterior           (d > R + r)       ----> 2
int circleCirclePosition(Circle c1, Circle
    c2) {
    Tf d = length(c1.o - c2.o);
    int in = dcmp(d - abs(c1.r - c2.r)), ex =
        dcmp(d - (c1.r + c2.r));
    return in < 0 ? -2 : in == 0 ? -1 : ex ==
        0 ? 1 : ex > 0 ? 2 : 0;
}

// compute the intersection points between
    two circles c1 && c2
vector<Point> circleCircleIntersection(Circle
    c1, Circle c2) {
    static_assert(is_same<Tf, Ti>::value);

    vector<Point> ret;
    Tf d = length(c1.o - c2.o);
    if(dcmp(d) == 0) return ret;
    if(dcmp(c1.r + c2.r - d) < 0) return ret;
    if(dcmp(abs(c1.r - c2.r) - d) > 0) return
        ret;

    Point v = c2.o - c1.o;
    Tf co = (c1.r * c1.r + sqLength(v) - c2.r
        * c2.r) / (2 * c1.r * length(v));
    Tf si = sqrt(abs(1.0 - co * co));
    Point p1 = scale(rotatePrecise(v, co,
        -si), c1.r) + c1.o;
    Point p2 = scale(rotatePrecise(v, co, si),
        c1.r) + c1.o;

    ret.push_back(p1);
    if(p1 != p2) ret.push_back(p2);
    return ret;
}

// intersection area between two circles c1,
    c2
Tf circleCircleIntersectionArea(Circle c1,
    Circle c2) {
    Point AB = c2.o - c1.o;
    Tf d = length(AB);
    if(d >= c1.r + c2.r) return 0;
    if(d + c1.r <= c2.r) return PI * c1.r *
        c1.r;
    if(d + c2.r <= c1.r) return PI * c2.r *
        c2.r;

    Tf alpha1 = acos((c1.r * c1.r + d * d -
        c2.r * c2.r) / (2.0 * c1.r * d));
    Tf alpha2 = acos((c2.r * c2.r + d * d -
        c1.r * c1.r) / (2.0 * c2.r * d));
    return c1.sector(2 * alpha1) + c2.sector(2
        * alpha2);
}

// returns tangents from a point p to circle c
vector<Point> pointCircleTangents(Point p,
    Circle c) {
    static_assert(is_same<Tf, Ti>::value);

    vector<Point> ret;
    Point u = c.o - p;
    Tf d = length(u);
    if(d < c.r) ;
    else if(dcmp(d - c.r) == 0) {
        ret = { rotate(u, PI / 2) };
    }
    else {
        Tf ang = asin(c.r / d);
        ret = { rotate(u, -ang), rotate(u,
            ang) };
    }
    return ret;
}

// returns the points on tangents that
    touches the circle
vector<Point> pointCircleTangencyPoints(Point
    p, Circle c) {
    static_assert(is_same<Tf, Ti>::value);

    Point u = p - c.o;
    Tf d = length(u);
    if(d < c.r) return {};
    else if(dcmp(d - c.r) == 0)  return {c.o +
        u};
    else {
        Tf ang = acos(c.r / d);
        u = u / length(u) * c.r;
        return { c.o + rotate(u, -ang), c.o +
            rotate(u, ang) };
    }
}

// for two circles c1 && c2, returns two list
    of points a && b
// such that a[i] is on c1 && b[i] is c2 &&
    for every i
// Line(a[i], b[i]) is a tangent to both
    circles
// CAUTION: a[i] = b[i] in case they touch |
    -1 for c1 = c2
int circleCircleTangencyPoints(Circle c1,
    Circle c2, vector<Point> &a,
    vector<Point> &b) {
    a.clear(), b.clear();
    int cnt = 0;
    if(dcmp(c1.r - c2.r) < 0) {
        swap(c1, c2); swap(a, b);
    }
    Tf d2 = sqLength(c1.o - c2.o);
    Tf rdif = c1.r - c2.r, rsum = c1.r + c2.r;
    if(dcmp(d2 - rdif * rdif) < 0) return 0;
    if(dcmp(d2) == 0 && dcmp(c1.r - c2.r) ==
        0) return -1;

    Tf base = angle(c2.o - c1.o);
    if(dcmp(d2 - rdif * rdif) == 0) {
        a.push_back(c1.point(base));
        b.push_back(c2.point(base));
        cnt++;
        return cnt;
    }

    Tf ang = acos((c1.r - c2.r) / sqrt(d2));
    a.push_back(c1.point(base + ang));
    b.push_back(c2.point(base + ang));
    cnt++;
    a.push_back(c1.point(base - ang));
    b.push_back(c2.point(base - ang));
    cnt++;
```

```
    if(dcmp(d2 - rsum * rsum) == 0) {
        a.push_back(c1.point(base));
        b.push_back(c2.point(PI + base));
        cnt++;
    }
    else if(dcmp(d2 - rsum * rsum) > 0) {
        Tf ang = acos((c1.r + c2.r) /
            sqrt(d2));
        a.push_back(c1.point(base + ang));
        b.push_back(c2.point(PI + base + ang));
        cnt++;
        a.push_back(c1.point(base - ang));
        b.push_back(c2.point(PI + base - ang));
        cnt++;
    }
    return cnt;
}
```

8.7 302 Convex

```
///minkowski sum of two polygons in O(n)
Polygon minkowskiSum(Polygon A, Polygon B){
    int n = A.size(), m = B.size();
    rotate(A.begin(), min_element(A.begin(),
        A.end()), A.end());
    rotate(B.begin(), min_element(B.begin(),
        B.end()), B.end());

    A.push_back(A[0]); B.push_back(B[0]);
    for(int i = 0; i < n; i++) A[i] = A[i+1] -
        A[i];
    for(int i = 0; i < m; i++) B[i] = B[i+1] -
        B[i];

    Polygon C(n+m+1);
    C[0] = A.back() + B.back();
    merge(A.begin(), A.end()-1, B.begin(),
        B.end()-1, C.begin()+1,
        polarComp(Point(0, 0), Point(0, -1)));
    for(int i = 1; i < C.size(); i++) C[i] =
        C[i] + C[i-1];
    C.pop_back();
    return C;
}

/// finds the rectangle with minimum area
    enclosing a convex polygon and
/// the rectangle with minimum perimeter
    enclosing a convex polygon
/// Tested on
    https://open.kattis.com/problems/fenceortho
pair< Tf, Tf
    >rotatingCalipersBoundingBox(const
    Polygon &p) {
    using Linear::distancePointLine;
    static_assert(is_same<Tf, Ti>::value);
    int n = p.size();
    int l = 1, r = 1, j = 1;
    Tf area = 1e100;
    Tf perimeter = 1e100;
    for(int i = 0; i < n; i++) {
        Point v = (p[(i+1)%n] - p[i]) /
            length(p[(i+1)%n] - p[i]);
        while(dcmp(dot(v, p[r%n] - p[i]) -
            dot(v, p[(r+1)%n] - p[i])) < 0)
            r++;
        while(j < r || dcmp(cross(v, p[j%n] -
            p[i]) - cross(v, p[(j+1)%n] -
            p[i])) < 0) j++;
        while(l < j || dcmp(dot(v, p[l%n] -
            p[i]) - dot(v, p[(l+1)%n] - p[i]))
            > 0) l++;
        Tf w = dot(v, p[r%n] - p[i]) - dot(v,
            p[l%n] - p[i]);
        Tf h = distancePointLine(p[j%n],
            Line(p[i], p[(i+1)%n]));
        area = min(area, w * h);
        perimeter = min(perimeter, 2 * w + 2 *
            h);
    }
    return make_pair(area, perimeter);
}

// returns the left side of polygon u after
    cutting it by ray a->b
Polygon cutPolygon(Polygon u, Point a, Point
    b) {
    using Linear::lineLineIntersection;
    using Linear::onSegment;

    Polygon ret;
    int n = u.size();
    for(int i = 0; i < n; i++) {
        Point c = u[i], d = u[(i + 1) % n];
        if(dcmp(cross(b-a, c-a)) >= 0)
            ret.push_back(c);
        if(dcmp(cross(b-a, d-c)) != 0) {
            Point t;
            lineLineIntersection(a, b - a, c, d
                - c, t);
            if(onSegment(t, Segment(c, d)))
                ret.push_back(t);
        }
    }
    return ret;
}

// returns true if point p is in or on
    triangle abc
bool pointInTriangle(Point a, Point b, Point
    c, Point p) {
    return dcmp(cross(b - a, p - a)) >= 0
        && dcmp(cross(c - b, p - b)) >= 0
        && dcmp(cross(a - c, p - c)) >= 0;
}

// Tested :
    https://www.spoj.com/problems/INOROUT
// pt must be in ccw order with no three
    collinear points
// returns inside = -1, on = 0, outside = 1
int pointInConvexPolygon(const Polygon &pt,
    Point p) {
    int n = pt.size();
    assert(n >= 3);

    int lo = 1, hi = n - 1;
    while(hi - lo > 1) {
        int mid = (lo + hi) / 2;
        if(dcmp(cross(pt[mid] - pt[0], p -
            pt[0])) > 0) lo = mid;
        else   hi = mid;
    }
```

```
    bool in = pointInTriangle(pt[0], pt[lo],
        pt[hi], p);
    if(!in) return 1;

    if(dcmp(cross(pt[lo] - pt[lo - 1], p -
        pt[lo - 1])) == 0) return 0;
    if(dcmp(cross(pt[hi] - pt[lo], p -
        pt[lo])) == 0) return 0;
    if(dcmp(cross(pt[hi] - pt[(hi + 1) % n], p
        - pt[(hi + 1) % n])) == 0) return 0;
    return -1;
}

// Extreme Point for a direction is the
    farthest point in that direction
// also
    https://codeforces.com/blog/entry/48868
// u is the direction for extremeness
// weakly tested on
    https://open.kattis.com/problems/fenceortho
int extremePoint(const Polygon &poly, Point
    u) {
    int n = (int) poly.size();
    int a = 0, b = n;
    while(b - a > 1) {
        int c = (a + b) / 2;
        if(dcmp(dot(poly[c] - poly[(c + 1) %
            n], u)) >= 0 && dcmp(dot(poly[c] -
            poly[(c - 1 + n) % n], u)) >= 0) {
            return c;
        }

        bool a_up = dcmp(dot(poly[(a + 1) % n]
            - poly[a], u)) >= 0;
        bool c_up = dcmp(dot(poly[(c + 1) % n]
            - poly[c], u)) >= 0;
        bool a_above_c = dcmp(dot(poly[a] -
            poly[c], u)) > 0;

        if(a_up && !c_up) b = c;
        else if(!a_up && c_up) a = c;
        else if(a_up && c_up) {
            if(a_above_c) b = c;
            else a = c;
        }
        else {
            if(!a_above_c) b = c;
            else a = c;
        }
    }

    if(dcmp(dot(poly[a] - poly[(a + 1) % n],
        u)) > 0 && dcmp(dot(poly[a] - poly[(a
        - 1 + n) % n], u)) > 0)
        return a;
    return b % n;
}

// For a convex polygon p and a line l,
    returns a list of segments
// of p that touch or intersect line l.
// the i'th segment is considered (p[i], p[(i
    + 1) modulo |p|])
// #1 If a segment is collinear with the
    line, only that is returned
// #2 Else if l goes through i'th point, the
    i'th segment is added
// Complexity: O(lg |p|)
vector<int> lineConvexPolyIntersection(const
    Polygon &p, Line l) {
    assert((int) p.size() >= 3);
    assert(l.a != l.b);

    int n = p.size();
    vector<int> ret;

    Point v = l.b - l.a;
    int lf = extremePoint(p, rotate90(v));
    int rt = extremePoint(p, rotate90(v) *
        Ti(-1));
    int olf = orient(l.a, l.b, p[lf]);
    int ort = orient(l.a, l.b, p[rt]);

    if(!olf || !ort) {
        int idx = (!olf ? lf : rt);
        if(orient(l.a, l.b, p[(idx - 1 + n) %
            n]) == 0)
            ret.push_back((idx - 1 + n) % n);
        else    ret.push_back(idx);
        return ret;
    }
    if(olf == ort) return ret;

    for(int i=0; i<2; ++i) {
        int lo = i ? rt : lf;
        int hi = i ? lf : rt;
        int olo = i ? ort : olf;

        while(true) {
            int gap = (hi - lo + n) % n;
            if(gap < 2) break;

            int mid = (lo + gap / 2) % n;
            int omid = orient(l.a, l.b, p[mid]);
            if(!omid) {
                lo = mid;
                break;
            }
            if(omid == olo) lo = mid;
            else hi = mid;
        }
        ret.push_back(lo);
    }
    return ret;
}

// Tested : https://toph.co/p/cover-the-points
// Calculate [ACW, CW] tangent pair from an
    external point
constexpr int CW = -1, ACW = 1;
bool isGood(Point u, Point v, Point Q, int
    dir) { return orient(Q, u, v) != -dir; }
Point better(Point u, Point v, Point Q, int
    dir) { return orient(Q, u, v) == dir ? u
    : v; }
Point pointPolyTangent(const Polygon &pt,
    Point Q, int dir, int lo, int hi) {
    while(hi - lo > 1) {
        int mid = (lo + hi) / 2;
        bool pvs = isGood(pt[mid], pt[mid -
            1], Q, dir);
        bool nxt = isGood(pt[mid], pt[mid +
            1], Q, dir);
```

```
        if(pvs && nxt) return pt[mid];
        if(!(pvs || nxt)) {
            Point p1 = pointPolyTangent(pt, Q,
                dir, mid + 1, hi);
            Point p2 = pointPolyTangent(pt, Q,
                dir, lo, mid - 1);
            return better(p1, p2, Q, dir);
        }

        if(!pvs) {
            if(orient(Q, pt[mid], pt[lo]) ==
                dir)          hi = mid - 1;
            else if(better(pt[lo], pt[hi], Q,
                dir) == pt[lo]) hi = mid - 1;
            else    lo = mid + 1;
        }
        if(!nxt) {
            if(orient(Q, pt[mid], pt[lo]) ==
                dir)          lo = mid + 1;
            else if(better(pt[lo], pt[hi], Q,
                dir) == pt[lo]) hi = mid - 1;
            else    lo = mid + 1;
        }
    }

    Point ret = pt[lo];
    for(int i = lo + 1; i <= hi; i++) ret =
        better(ret, pt[i], Q, dir);
    return ret;
}
// [ACW, CW] Tangent
pair<Point, Point> pointPolyTangents(const
    Polygon &pt, Point Q) {
    int n = pt.size();
    Point acw_tan = pointPolyTangent(pt, Q,
        ACW, 0, n - 1);
    Point cw_tan = pointPolyTangent(pt, Q, CW,
        0, n - 1);
    return make_pair(acw_tan, cw_tan);
}
```

8.8 303 Enclosing Circle

```
// returns false if points are collinear,
    true otherwise
// circle p touch each arm of triangle abc
bool inscribedCircle(Point a, Point b, Point
    c, Circle &p) {
    using Linear::distancePointLine;
    static_assert(is_same<Tf, Ti>::value);
    if(orient(a, b, c) == 0) return false;
    Tf u = length(b - c);
    Tf v = length(c - a);
    Tf w = length(a - b);
    p.o = (a * u + b * v + c * w) / (u + v +
        w);
    p.r = distancePointLine(p.o, Line(a, b));
    return true;
}

// set of points A(x, y) such that PA : QA =
    rp : rq
Circle apolloniusCircle(Point P, Point Q, Tf
    rp, Tf rq) {
    static_assert(is_same<Tf, Ti>::value);
    rq *= rq; rp *= rp;
    Tf a = rq - rp;
    assert(dcmp(a));
    Tf g = (rq * P.x - rp * Q.x)/a;
    Tf h = (rq * P.y - rp * Q.y)/a;
    Tf c = (rq * P.x * P.x - rp * Q.x * Q.x +
        rq * P.y * P.y - rp * Q.y * Q.y)/a;
    Point o(g, h);
    Tf R = sqrt(g * g + h * h - c);
    return Circle(o, R);
}

// returns false if points are collinear,
    true otherwise
// circle p goes through points a, b && c
bool circumscribedCircle(Point a, Point b,
    Point c, Circle &p) {
    using Linear::lineLineIntersection;
    if(orient(a, b, c) == 0) return false;
    Point d = (a + b) / 2, e = (a + c) / 2;
    Point vd = rotate90(b - a), ve =
        rotate90(a - c);
    bool f = lineLineIntersection(d, vd, e,
        ve, p.o);
    if(f) p.r = length(a - p.o);
    return f;
}

// Following three methods implement Welzl's
    algorithm for
// the smallest enclosing circle problem:
    Given a set of
// points, find out the minimal circle that
    covers them all.
// boundary(p) determines (if possible) a
    circle that goes
// through the points in p. Ideally |p| <= 3.
// welzl() is a recursive helper function
    doing the most part
// of Welzl's algorithm. Call minidisk with
    the set of points
// Randomized Complexity: O(CN) with C~10
    (practically lesser)

Circle boundary(const vector<Point> &p) {
    Circle ret;
    int sz = p.size();
    if(sz == 0)       ret.r = 0;
    else if(sz == 1)  ret.o = p[0], ret.r = 0;
    else if(sz == 2)  ret.o = (p[0] + p[1]) /
        2, ret.r = length(p[0] - p[1]) / 2;
    else if(!circumscribedCircle(p[0], p[1],
        p[2], ret)) ret.r = 0;
    return ret;
}
Circle welzl(const vector<Point> &p, int fr,
    vector<Point> &b) {
    if(fr >= (int) p.size() || b.size() == 3)
        return boundary(b);

    Circle c = welzl(p, fr + 1, b);
    if(!c.contains(p[fr])) {
        b.push_back(p[fr]);
        c = welzl(p, fr + 1, b);
        b.pop_back();
    }
    return c;
```

```
}
Circle minidisk(vector<Point> p) {
    random_shuffle(p.begin(), p.end());
    vector<Point> q;
    return welzl(p, 0, q);
}
```

8.9 304 Half Planar

```
using Linear::lineLineIntersection;
struct DirLine {
    Point p, v;
    Tf ang;
    DirLine() {}
    /// Directed line containing point P in
        the direction v
    DirLine(Point p, Point v) : p(p), v(v) {
        ang = atan2(v.y, v.x); }
    /// Directed Line for ax+by+c >=0
    DirLine(Tf a, Tf b, Tf c) {
        assert(dcmp(a) || dcmp(b));
        p = dcmp(a) ? Point(-c/a, 0) :
            Point(0,-c/b);
        v = Point(b, -a);
        ang = atan2(v.y, v.x);
    }
    bool operator<(const DirLine& u) const {
        return ang < u.ang; }
    bool onLeft(Point x) const { return
        dcmp(cross(v, x-p)) >= 0; }
};

// Returns the region bounded by the left
    side of some directed lines
// MAY CONTAIN DUPLICATE POINTS
// OUTPUT IS UNDEFINED if intersection is
    unbounded
// Complexity: O(n log n) for sorting, O(n)
    afterwards
Polygon halfPlaneIntersection(vector<DirLine>
    li) {
    int n = li.size(), first = 0, last = 0;
    sort(li.begin(), li.end());
    vector<Point> p(n);
    vector<DirLine> q(n);
    q[0] = li[0];

    for(int i = 1; i < n; i++) {
        while(first < last &&
            !li[i].onLeft(p[last - 1])) last--;
        while(first < last &&
            !li[i].onLeft(p[first])) first++;
        q[++last] = li[i];
        if(dcmp(cross(q[last].v, q[last-1].v))
            == 0) {
            last--;
            if(q[last].onLeft(li[i].p)) q[last]
                = li[i];
        }
        if(first < last)
            lineLineIntersection(q[last - 1].p,
                q[last - 1].v, q[last].p,
                q[last].v, p[last - 1]);
    }

    while(first < last &&
        !q[first].onLeft(p[last - 1])) last--;
    if(last - first <= 1) return {};
    lineLineIntersection(q[last].p, q[last].v,
        q[first].p, q[first].v, p[last]);
    return Polygon(p.begin()+first,
        p.begin()+last+1);
}

// O(n^2 lg n) implementation of Voronoi
    Diagram bounded by INF square
// returns region, where regions[i] = set of
    points for which closest
// point is site[i]. This region is a polygon.
const Tf INF = 1e10;
vector<Polygon> voronoi(const vector<Point>
    &site, Tf bsq) {
    int n = site.size();
    vector<Polygon> region(n);
    Point A(-bsq, -bsq), B(bsq, -bsq), C(bsq,
        bsq), D(-bsq, bsq);

    for(int i = 0; i < n; ++i) {
        vector<DirLine> li(n - 1);
        for(int j = 0, k = 0; j < n; ++j) {
            if(i == j) continue;
            li[k++] = DirLine((site[i] +
                site[j]) / 2, rotate90(site[j]
                - site[i]));
        }
        li.emplace_back(A, B - A);
        li.emplace_back(B, C - B);
        li.emplace_back(C, D - C);
        li.emplace_back(D, A - D);
        region[i] = halfPlaneIntersection(li);
    }
    return region;
}
```

8.10 305 Intersecting Segments

```
// Given a list of segments v, finds a pair
    (i, j)
// st v[i], v[j] intersects. If none, returns
    {-1, -1}
// Tested Timus 1469, CF 1359F
struct Event {
    Tf x;
    int tp, id;
    bool operator < (const Event &p) const {
        if(dcmp(x - p.x)) return x < p.x;
        return tp > p.tp;
    }
};

pair<int, int> anyIntersection(const
    vector<Segment> &v) {
    using Linear::segmentsIntersect;
    static_assert(is_same<Tf, Ti>::value);

    vector<Event> ev;
    for(int i=0; i<v.size(); i++) {
        ev.push_back({min(v[i].a.x, v[i].b.x),
            +1, i});
        ev.push_back({max(v[i].a.x, v[i].b.x),
            -1, i});
```

```
}
sort(ev.begin(), ev.end());

auto comp = [&v] (int i, int j) {
    Segment p = v[i], q = v[j];
    Tf x = max(min(p.a.x, p.b.x),
        min(q.a.x, q.b.x));
    auto yvalSegment = [&x](const Line &s)
        {
        if(dcmp(s.a.x - s.b.x) == 0) return
            s.a.y;
        return s.a.y + (s.b.y - s.a.y) * (x
            - s.a.x) / (s.b.x - s.a.x);
    };
    return dcmp(yvalSegment(p) -
        yvalSegment(q)) < 0;
};

multiset<int, decltype(comp)> st(comp);
typedef decltype(st)::iterator iter;
auto prev = [&st](iter it) {
    return it == st.begin() ? st.end() :
        --it;
};
auto next = [&st](iter it) {
    return it == st.end() ? st.end() :
        ++it;
};

vector<iter> pos(v.size());
for(auto &cur : ev) {
    int id = cur.id;
    if(cur.tp == 1) {
        iter nxt = st.lower_bound(id);
        iter pre = prev(nxt);
        if(pre != st.end() &&
            segmentsIntersect(v[*pre],
            v[id])) return {*pre, id};
        if(nxt != st.end() &&
            segmentsIntersect(v[*nxt],
            v[id])) return {*nxt, id};
        pos[id] = st.insert(nxt, id);
    }
    else {
        iter nxt = next(pos[id]);
        iter pre = prev(pos[id]);
        if(pre != st.end() && nxt !=
            st.end() &&
            segmentsIntersect(v[*pre],
            v[*nxt]))
            return {*pre, *nxt};
        st.erase(pos[id]);
      }
    }
    return {-1, -1};
}
```

8.11 306 Linear

```
// returns true if point p is on segment s
bool onSegment(Point p, Segment s) {
    return dcmp(cross(s.a - p, s.b - p)) == 0
        && dcmp(dot(s.a - p, s.b - p)) <= 0;
}

// returns true if segment p && q touch or
    intersect
bool segmentsIntersect(Segment p, Segment q) {
    if(onSegment(p.a, q) || onSegment(p.b, q))
        return true;
    if(onSegment(q.a, p) || onSegment(q.b, p))
        return true;

    Ti c1 = cross(p.b - p.a, q.a - p.a);
    Ti c2 = cross(p.b - p.a, q.b - p.a);
    Ti c3 = cross(q.b - q.a, p.a - q.a);
    Ti c4 = cross(q.b - q.a, p.b - q.a);
    return dcmp(c1) * dcmp(c2) < 0 && dcmp(c3)
        * dcmp(c4) < 0;
}

bool linesParallel(Line p, Line q) {
    return dcmp(cross(p.b - p.a, q.b - q.a))
        == 0;
}

// lines are represented as a ray from a
    point: (point, vector)
// returns false if two lines (p, v) && (q,
    w) are parallel or collinear
// true otherwise, intersection point is
    stored at o via reference
bool lineLineIntersection(Point p, Point v,
    Point q, Point w, Point& o) {
    static_assert(is_same<Tf, Ti>::value);
    if(dcmp(cross(v, w)) == 0) return false;
    Point u = p - q;
    o = p + v * (cross(w,u)/cross(v,w));
    return true;
}

// returns false if two lines p && q are
    parallel or collinear
// true otherwise, intersection point is
    stored at o via reference
bool lineLineIntersection(Line p, Line q,
    Point& o) {
    return lineLineIntersection(p.a, p.b -
        p.a, q.a, q.b - q.a, o);
}

// returns the distance from point a to line l
Tf distancePointLine(Point p, Line l) {
    return abs(cross(l.b - l.a, p - l.a) /
        length(l.b - l.a));
}

// returns the shortest distance from point a
    to segment s
Tf distancePointSegment(Point p, Segment s) {
    if(s.a == s.b) return length(p - s.a);
    Point v1 = s.b - s.a, v2 = p - s.a, v3 = p
        - s.b;
    if(dcmp(dot(v1, v2)) < 0)    return
        length(v2);
    else if(dcmp(dot(v1, v3)) > 0) return
        length(v3);
    else return abs(cross(v1, v2) /
        length(v1));
}

// returns the shortest distance from segment
    p to segment q
```

```
Tf distanceSegmentSegment(Segment p, Segment
    q) {
    if(segmentsIntersect(p, q)) return 0;
    Tf ans = distancePointSegment(p.a, q);
    ans = min(ans, distancePointSegment(p.b,
        q));
    ans = min(ans, distancePointSegment(q.a,
        p));
    ans = min(ans, distancePointSegment(q.b,
        p));
    return ans;
}

// returns the projection of point p on line l
Point projectPointLine(Point p, Line l) {
    static_assert(is_same<Tf, Ti>::value);
    Point v = l.b - l.a;
    return l.a + v * ((Tf) dot(v, p - l.a) /
        dot(v, v));
}
```

8.12 307 Point

```
typedef double Tf;
typedef double Ti;          /// use long long
    for exactness
const Tf PI = acos(-1), EPS = 1e-9;
int dcmp(Tf x) { return abs(x) < EPS ? 0 :
    (x<0 ? -1 : 1);}

struct Point {
    Ti x, y;
    Point(Ti x = 0, Ti y = 0) : x(x), y(y) {}

    Point operator + (const Point& u) const {
        return Point(x + u.x, y + u.y); }
    Point operator - (const Point& u) const {
        return Point(x - u.x, y - u.y); }
    Point operator * (const long long u) const
        { return Point(x * u, y * u); }
    Point operator * (const Tf u) const {
        return Point(x * u, y * u); }
    Point operator / (const Tf u) const {
        return Point(x / u, y / u); }

    bool operator == (const Point& u) const {
        return dcmp(x - u.x) == 0 && dcmp(y -
        u.y) == 0; }
    bool operator != (const Point& u) const {
        return !(*this == u); }
    bool operator < (const Point& u) const {
        return dcmp(x - u.x) < 0 || (dcmp(x -
        u.x) == 0 && dcmp(y - u.y) < 0); }
    friend istream &operator >> (istream &is,
        Point &p) { return is >> p.x >> p.y; }
    friend ostream &operator << (ostream &os,
        const Point &p) { return os << p.x <<
        " " << p.y; }
};

Ti dot(Point a, Point b) { return a.x * b.x +
    a.y * b.y; }
Ti cross(Point a, Point b) { return a.x * b.y
    - a.y * b.x; }
Tf length(Point a) { return sqrt(dot(a, a)); }
Ti sqLength(Point a) { return dot(a, a); }
Tf distance(Point a, Point b) {return
    length(a-b);}
Tf angle(Point u) { return atan2(u.y, u.x); }

// returns angle between oa, ob in (-PI, PI]
Tf angleBetween(Point a, Point b) {
    Tf ans = angle(b) - angle(a);
    return ans <= -PI ? ans + 2*PI : (ans > PI
        ? ans - 2*PI : ans);
}

// Rotate a ccw by rad radians
Point rotate(Point a, Tf rad) {
    static_assert(is_same<Tf, Ti>::value);
    return Point(a.x * cos(rad) - a.y *
        sin(rad), a.x * sin(rad) + a.y *
        cos(rad));
}

// rotate a ccw by angle th with cos(th) = co
    && sin(th) = si
Point rotatePrecise(Point a, Tf co, Tf si) {
    static_assert(is_same<Tf, Ti>::value);
    return Point(a.x * co - a.y * si, a.y * co
        + a.x * si);
}

Point rotate90(Point a) { return Point(-a.y,
    a.x); }

// scales vector a by s such that length of a
    becomes s
Point scale(Point a, Tf s) {
    static_assert(is_same<Tf, Ti>::value);
    return a / length(a) * s;
}

// returns an unit vector perpendicular to
    vector a
Point normal(Point a) {
    static_assert(is_same<Tf, Ti>::value);
    Tf l = length(a);
    return Point(-a.y / l, a.x / l);
}

// returns 1 if c is left of ab, 0 if on ab
    && -1 if right of ab
int orient(Point a, Point b, Point c) {
    return dcmp(cross(b - a, c - a));
}

///Use as sort(v.begin(), v.end(),
    polarComp(O, dir))
///Polar comparator around O starting at
    direction dir
struct polarComp {
    Point O, dir;
    polarComp(Point O = Point(0, 0), Point dir
        = Point(1, 0))
        : O(O), dir(dir) {}
    bool half(Point p) {
        return dcmp(cross(dir, p)) < 0 ||
            (dcmp(cross(dir, p)) == 0 &&
            dcmp(dot(dir, p)) > 0);
    }
    bool operator()(Point p, Point q) {
```

```
        return make_tuple(half(p), 0) <
            make_tuple(half(q), cross(p, q));
    }
};
struct Segment {
    Point a, b;
    Segment(Point aa, Point bb) : a(aa), b(bb)
        {}
};
typedef Segment Line;

struct Circle {
    Point o;
    Tf r;
    Circle(Point o = Point(0, 0), Tf r = 0) :
        o(o), r(r) {}

    // returns true if point p is in || on the
        circle
    bool contains(Point p) {
        return dcmp(sqLength(p - o) - r * r)
            <= 0;
    }

    // returns a point on the circle rad
        radians away from +X CCW
    Point point(Tf rad) {
        static_assert(is_same<Tf, Ti>::value);
        return Point(o.x + cos(rad) * r, o.y +
            sin(rad) * r);
    }

    // area of a circular sector with central
        angle rad
    Tf area(Tf rad = PI + PI) { return rad * r
        * r / 2; }

    // area of the circular sector cut by a
        chord with central angle alpha
    Tf sector(Tf alpha) { return r * r * 0.5 *
        (alpha - sin(alpha)); }
};
```

8.13 308 Point Rotation Trick

```
/// you may define the processor function in
    this namespace
/// instead of passing as an argument;
    testing shows function
/// defined using lambda and passed as
    argument performs better
/// tested on:
/// constant width strip -
    https://codeforces.com/gym/100016/problem/I
/// constant area triangle -
    https://codeforces.com/contest/1019/problem/D
/// smallest area quadrilateral -
    https://codingcompetitions.withgoogle.com/codejamio/round/0000000000051635/0000000000051e89
/// disjoint triangles count -
    https://codeforces.com/contest/1025/problem/F
/// smallest and largest triangle -
    http://serjudging.vanb.org/?p=561
typedef pair< int , int >PII;
void performTrick(vector< Point >pts, const
    function<void(const vector< Point >&,
    int)> &processor) {
    int n = pts.size();
    sort(pts.begin(), pts.end());
    vector< int >position(n);
    vector< PII >segments;
    segments.reserve((n*(n-1))/2);
    for (int i = 0; i < n; i++) {
        position[i] = i;
        for (int j = i+1; j < n; j++) {
            segments.emplace_back(i, j);
        }
    }
    assert(segments.capacity() ==
        segments.size());
    sort(segments.begin(), segments.end(),
        [&](PII p, PII q) {
        Ti prod =
            cross(pts[p.second]-pts[p.first],
            pts[q.second]-pts[q.first]);
        if (prod != 0) return prod > 0;
        return p < q;
    });
    for (PII seg : segments) {
        int i = position[seg.first];
        assert(position[seg.second] == i+1);
        processor(pts, i);
        swap(pts[i], pts[i+1]);
        swap(position[seg.first],
            position[seg.second]);
    }
}
```

8.14 309 Polygon

```
typedef vector<Point> Polygon;
// removes redundant points
// polygon can't be all colinear points
Polygon RemoveCollinear(const Polygon& poly) {
    Polygon ret;
    int n = poly.size();
    for(int i = 0; i < n; i++) {
        Point a = poly[i];
        Point b = poly[(i + 1) % n];
        Point c = poly[(i + 2) % n];
        if(dcmp(cross(b-a, c-b)) != 0 &&
            (ret.empty() || b != ret.back()))
            ret.push_back(b);
    }
    return ret;
}

// returns the signed area of polygon p of n
    vertices
Tf signedPolygonArea(const Polygon &p) {
    Tf ret = 0;
    for(int i = 0; i < (int) p.size() - 1; i++)
        ret += cross(p[i]-p[0], p[i+1]-p[0]);
    return ret / 2;
}

// given a polygon p of n vertices, generates
    the convex hull in in CCW
// Tested on
    https://acm.timus.ru/problem.aspx?space=1&num=118
// Caution: when all points are colinear AND
    removeRedundant == false
```

```
// output will be contain duplicate points
    (from upper hull) at back
Polygon convexHull(Polygon p, bool
    removeRedundant) {
    int check = removeRedundant ? 0 : -1;
    sort(p.begin(), p.end());
    p.erase(unique(p.begin(), p.end()),
        p.end());

    int n = p.size();
    Polygon ch(n+n);
    int m = 0;     // preparing lower hull
    for(int i = 0; i < n; i++) {
        while(m > 1 && dcmp(cross(ch[m - 1] -
            ch[m - 2], p[i] - ch[m - 1])) <=
            check) m--;
        ch[m++] = p[i];
    }
    int k = m;     // preparing upper hull
    for(int i = n - 2; i >= 0; i--) {
        while(m > k && dcmp(cross(ch[m - 1] -
            ch[m - 2], p[i] - ch[m - 2])) <=
            check) m--;
        ch[m++] = p[i];
    }
    if(n > 1) m--;
    ch.resize(m);
    return ch;
}

// Tested :
    https://www.spoj.com/problems/INOROUT
// returns inside = -1, on = 0, outside = 1
int pointInPolygon(const Polygon &p, Point o)
    {
    using Linear::onSegment;
    int wn = 0, n = p.size();
    for(int i = 0; i < n; i++) {
        int j = (i + 1) % n;
        if(onSegment(o, Segment(p[i], p[j]))
            || o == p[i]) return 0;
        int k = dcmp(cross(p[j] - p[i], o -
            p[i]));
        int d1 = dcmp(p[i].y - o.y);
        int d2 = dcmp(p[j].y - o.y);
        if(k > 0 && d1 <= 0 && d2 > 0) wn++;
        if(k < 0 && d2 <= 0 && d1 > 0) wn--;
    }
    return wn ? -1 : 1;
}

// Tested: Timus 1955, CF 598F
// Given a simple polygon p, and a line l,
    returns (x, y)
// x = longest segment of l in p, y = total
    length of l in p.
pair<Tf, Tf> linePolygonIntersection(Line l,
    const Polygon &p) {
    using Linear::lineLineIntersection;
    int n = p.size();
    vector<pair<Tf, int>> ev;
    for(int i=0; i<n; ++i) {
        Point a = p[i], b = p[(i+1)%n], z =
            p[(i-1+n)%n];
        int ora = orient(l.a, l.b, a), orb =
            orient(l.a, l.b, b), orz =
            orient(l.a, l.b, z);
        if(!ora) {
            Tf d = dot(a - l.a, l.b - l.a);
            if(orz && orb) {
                if(orz != orb)
                    ev.emplace_back(d, 0);
                //else // Point Touch
            }
            else if(orz) ev.emplace_back(d,
                orz);
            else if(orb) ev.emplace_back(d,
                orb);
        }
        else if(ora == -orb) {
            Point ins;
            lineLineIntersection(l, Line(a, b),
                ins);
            ev.emplace_back(dot(ins - l.a, l.b
                - l.a), 0);
        }
    }
    sort(ev.begin(), ev.end());

    Tf ans = 0, len = 0, last = 0, tot = 0;
    bool active = false;
    int sign = 0;
    for(auto &qq : ev) {
        int tp = qq.second;
        Tf d = qq.first;  /// current Segment
            is (last, d)
        if(sign) {        /// On Border
            len += d-last; tot += d-last;
            ans = max(ans, len);
            if(tp != sign) active = !active;
            sign = 0;
        }
        else {
            if(active) { ///Strictly Inside
                len += d-last; tot += d-last;
                ans = max(ans, len);
            }
            if(tp == 0) active = !active;
            else sign = tp;
        }
        last = d;
        if(!active) len = 0;
    }
    ans /= length(l.b-l.a);
    tot /= length(l.b-l.a);
    return {ans, tot};
}
```

9 09 Flow or Matching

9.1 101 Edmonds Karp

```
// Complexity O(VE^2)
LL n;
LL cap[109][109], parent[109];
bool vis[109];
vector<LL> adj[109];

LL bfs(LL s, LL t)
{
    memset(vis, 0, sizeof vis);
```

```
    LL cur, flow, new_flow;

    queue<pLL> q;
    q.push({INF, s});
    vis[s] = 1;

    while(!q.empty())
    {
        flow = q.front().ff;
        cur = q.front().ss;
        q.pop();

        for(auto nxt : adj[cur])
        {
            if(vis[nxt] == 0 && cap[cur][nxt])
            {
                parent[nxt] = cur;
                new_flow = min(flow,
                    cap[cur][nxt]);

                if(nxt == t)
                    return new_flow;

                q.push({new_flow, nxt});
                vis[nxt] = 1;
            }
        }
    }

    return 0;
}

LL maxflow(LL s, LL t)
{
    LL prev, cur;
    LL flow = 0, new_flow;

    while(new_flow = bfs(s, t))
    {
        flow += new_flow;

        for(cur = t; cur != s; cur = prev)
        {
            prev = parent[cur];
            cap[prev][cur] -= new_flow;
            cap[cur][prev] += new_flow;
        }
    }

    return flow;
}

/*
memset(cap, 0, sizeof cap);

cap[a][b] += w;
cap[b][a] += w;
adj[a].pb(b);
adj[b].pb(a);

cout << maxflow(s, t);
*/
```

10 10 String

10.1 101 Hashing

```
LL base = 31;

LL n, p[MAXN];
LL preHash[MAXN];
LL sufHash[MAXN];

void initHash()
{
    LL i;

    p[0] = 1;
    for(i = 1; i < MAXN; i++)
        p[i] = (p[i-1] * base) % MOD;
}

void createHash (string s)
{
    LL i;

    n = s.size();

    for(i = 1; i <= n; i++)
    {
        preHash[i] = (preHash[i-1] * base) %
            MOD;
        preHash[i] = (preHash[i] + s[i-1] -
            'a' + 1) % MOD;
    }

    for(i = n; i > 0; i--)
    {
        sufHash[i] = (sufHash[i+1] * base) %
            MOD;
        sufHash[i] = (sufHash[i] + s[i-1] -
            'a' + 1) % MOD;
    }
}
```

10.2 102 KMP

```
// Longest Prefix which is also a Suffix
LL LPS[MAXN], n;

void KMP(string s)
{
    LL i, j, n = s.size();

    LPS[0]=0;

    for (i = 1; i < n; i++)
    {
        j = LPS[i-1];

        while (j > 0 && s[i] != s[j])
            j = LPS[j-1];

        if (s[i] == s[j])
            j++;

        LPS[i] = j;
    }
}
```

10.3 103 Z algorithm

```
vector<LL> z_function(string s)
{
    LL i, l, r, n = s.size();
    vector<LL> Z(n);
    Z[0] = 0;
    for(i = 1, l = 0, r = 0; i < n; i++)
    {
        if(i <= r) //This condition is false
            when i=1
            Z[i] = min(r-i+1, Z[i-l]);
        while(i+Z[i] < n && s[Z[i]] ==
            s[i+Z[i]])
            Z[i]++; //if Z[1] has previous
                value, it will cause problem
                here

        if(i+Z[i] - 1 > r)
        {
            l = i;
            r = i+Z[i]-1;
        }
    }

    return Z;
}
```

10.4 104 Manacher's Algo

```
LL pal[300009], n;
string s = "#";

void manacher()
{
    LL i, l, r, k;

    for(i = 0, l = 0, r = -1; i < n; i++)
    {
        if(i > r)
            k = 1;
        else
            k = min(pal[l + r - i], r - i + 1);
        while(i-k >= 0 && i+k < n && s[i-k] ==
            s[i+k])
            k++;

        pal[i] = k;
        k--;

        if(i+k > r)
        {
            l = i-k;
            r = i+k;
        }
    }
}

int main()
{
    LL i, ans = 0;
    string stemp;

    cin >> n >> stemp;

    for(i = 0; i < stemp.size(); i++)
    {
        s.push_back(stemp[i]);
        s.push_back('#');
    }

    n = s.size();

    manacher();

    for(i = 0; i < n; i++)
        cout << pal[i] << " ";
}
```

10.5 105 Suffix Array

```
vector<pair<pii, int>> bucket[MAXN];
void radix_sort(vector<pair<pii, int>> &v)
{
    int i, j, n = v.size();

    for(i = 0; i < MAXN; i++)
        bucket[i].clear();

    for(i = 0; i < n; i++)
        bucket[v[i].ff.ss + 1].push_back(v[i]);

    v.clear();
    for(i = 0; i < MAXN; i++)
    {
        for(auto u : bucket[i])
            v.push_back(u);
        bucket[i].clear();
    }

    for(i = 0; i < n; i++)
        bucket[v[i].ff.ff].push_back(v[i]);

    v.clear();
    for(i = 0; i < MAXN; i++)
    {
        for(auto u : bucket[i])
            v.push_back(u);
    }
}

vector<int> get_SA(string s)
{
    int i, len, cnt, n = s.size();
    vector<int> prev(n), sa(n);
    vector<pair<pii, int>> curr;

    for(i = 0; i < n; i++)
        prev[i] = s[i];

    for(len = 2; len <= 2*n; len *= 2)
    {
        curr.clear();
        for(i = 0; i < n; i++)
        {
            if(i+len/2 >= n)
                curr.push_back({{prev[i], -1},
                    i});
            else
```

```
            curr.push_back({{prev[i],
                prev[i+len/2]}, i});
        }

        radix_sort(curr);
        // sort(curr.begin(), curr.end());

        for(i = cnt = 0; i < n; i++)
        {
            if(i == 0 || curr[i].ff ==
                curr[i-1].ff)
                prev[curr[i].ss] = cnt;
            else
                prev[curr[i].ss] = ++cnt;
        }
    }

    for(i = 0; i < n; i++)
        sa[prev[i]] = i;

    return sa;
}

vector<int> get_LCP(string s, vector<int>&sa)
{
    int i, j, k, n = s.size();

    vector<int> lcp(n-1, 0), rank(n, 0);

    for(i = 0; i < n; i++)
        rank[sa[i]] = i;

    for(i = 0, k = 0; i < n; i++)
    {
        if(rank[i] == n-1)
        {
            k = 0;
            continue;
        }

        j = sa[rank[i] + 1];
        while(i+k < n && j+k < n && s[i+k] ==
            s[j+k])
            k++;

        lcp[rank[i]] = k;

        k = max(k-1, 0);
    }

    return lcp;
}
```

10.6 301 Hashing

```
const PLL M = {1e9 + 7, 1e9 + 9}; /// Should
    be large primes
const LL base = 1259;           /// Should be
    larger than alphabet size
const int N = 1e6 + 7;          /// Highest
    length of string

PLL operator+(const PLL& a, LL x) { return
    {a.ff + x, a.ss + x}; }
PLL operator-(const PLL& a, LL x) { return
    {a.ff - x, a.ss - x}; }
PLL operator*(const PLL& a, LL x) { return
    {a.ff * x, a.ss * x}; }
PLL operator+(const PLL& a, PLL x) { return
    {a.ff + x.ff, a.ss + x.ss}; }
PLL operator-(const PLL& a, PLL x) { return
    {a.ff - x.ff, a.ss - x.ss}; }
PLL operator*(const PLL& a, PLL x) { return
    {a.ff * x.ff, a.ss * x.ss}; }
PLL operator%(const PLL& a, PLL m) { return
    {a.ff % m.ff, a.ss % m.ss}; }
ostream& operator<<(ostream& os, PLL hash) {
    return os << "(" << hash.ff << ", " <<
        hash.ss << ")";
}
PLL pb[N]; /// powers of base mod M

/// Call pre before everything
void hashPre() {
    pb[0] = {1, 1};
    for (int i = 1; i < N; i++) pb[i] = (pb[i
        - 1] * base) % M;
}

/// Calculates hashes of all prefixes of s
    including empty prefix
vector<PLL> hashList(string s) {
    int n = s.size();
    vector<PLL> ans(n + 1);
    ans[0] = {0, 0};
    for (int i = 1; i <= n; i++) ans[i] =
        (ans[i - 1] * base + s[i - 1]) % M;
    return ans;
}

/// Calculates hash of substring s[l..r] (1
    indexed)
PLL substringHash(const vector<PLL>&
    hashlist, int l, int r) {
    return (hashlist[r] + (M - hashlist[l -
        1]) * pb[r - l + 1]) % M;
}

/// Calculates Hash of a string
PLL Hash(string s) {
    PLL ans = {0, 0};
    for (int i = 0; i < s.size(); i++) ans =
        (ans * base + s[i]) % M;
    return ans;
}

/// Tested on https://toph.co/p/palindromist
/// appends c to string
PLL append(PLL cur, char c) { return (cur *
    base + c) % M; }

/// Tested on https://toph.co/p/palindromist
/// prepends c to string with size k
PLL prepend(PLL cur, int k, char c) { return
    (pb[k] * c + cur) % M; }

/// Tested on https://toph.co/p/chikongunia
/// replaces the i-th (0-indexed) character
    from right from a to b;
PLL replace(PLL cur, int i, char a, char b) {
    return cur + pb[i] * (M + b - a) % M;
}
```

```
/// Erases c from front of the string with
    size len
PLL pop_front(PLL hash, int len, char c) {
    return (hash + pb[len - 1] * (M - c)) % M;
}

/// Tested on https://toph.co/p/palindromist
/// concatenates two strings where length of
    the right is k
PLL concat(PLL left, PLL right, int k) {
    return (left * pb[k] + right) % M; }

PLL power(const PLL& a, LL p) {
    if (p == 0) return {1, 1};
    PLL ans = power(a, p / 2);
    ans = (ans * ans) % M;
    if (p % 2) ans = (ans * a) % M;
    return ans;
}

PLL inverse(PLL a) {
    if (M.ss == 1) return power(a, M.ff - 2);
    return power(a, (M.ff - 1) * (M.ss - 1) -
        1);
}

/// Erases c from the back of the string
PLL invb = inverse({base, base});
PLL pop_back(PLL hash, char c) { return
    ((hash - c + M) * invb) % M; }

/// Tested on https://toph.co/p/palindromist
/// Calculates hash of string with size len
    repeated cnt times
/// This is O(log n). For O(1), pre-calculate
    inverses
PLL repeat(PLL hash, int len, LL cnt) {
    PLL mul = ((pb[len * cnt] - 1 + M) *
        inverse(pb[len] - 1 + M)) % M;
    PLL ans = (hash * mul);
    if (pb[len].ff == 1) ans.ff = hash.ff *
        cnt;
    if (pb[len].ss == 1) ans.ss = hash.ss *
        cnt;
    return ans % M;
}
```

10.7 302 Manacher's

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

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

    string s;
    cin >> s;

    int n = s.size();
    vector<int> d1(n);
    // d[i] = number of palindromes taking
        s[i] as center
    for (int i = 0, l = 0, r = -1; i < n; i++)
        {
        int k = (i > r) ? 1 : min(d1[l + r -
            i], r - i + 1);
        while (0 <= i - k && i + k < n && s[i
            - k] == s[i + k]) k++;
        d1[i] = k--; if (i + k > r) l = i - k,
            r = i + k;
    }

    vector<int> d2(n);
    // d[i] = number of palindromes taking
        s[i-1] and s[i] as center
    for (int i = 0, l = 0, r = -1; i < n; i++)
        {
        int k = (i > r) ? 0 : min(d2[l + r - i
            + 1], r - i + 1);
        while (0 <= i - k - 1 && i + k < n &&
            s[i - k - 1] == s[i + k]) k++;
        d2[i] = k--;  if (i + k > r) l = i - k
            - 1, r = i + k;
    }
}
```

10.8 303 SuffixArray

```
/**
 Suffix Array implementation with count sort.
 Source: E-MAXX
 Running time:
    Suffix Array Construction: O(NlogN)
    LCP Array Construction: O(NlogN)
    Suffix LCP: O(logN)
**/

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

typedef pair<int, int> PII;
typedef vector<int> VI;

/// Equivalence Class INFO
vector<VI> c;
VI sort_cyclic_shifts(const string &s) {
    int n = s.size();
    const int alphabet = 256;
    VI p(n), cnt(alphabet, 0);

    c.clear();
    c.emplace_back();
    c[0].resize(n);

    for (int i = 0; i < n; i++) cnt[s[i]]++;
    for (int i = 1; i < alphabet; i++) cnt[i]
        += cnt[i - 1];
    for (int i = 0; i < n; i++) p[--cnt[s[i]]]
        = i;

    c[0][p[0]] = 0;
    int classes = 1;

    for (int i = 1; i < n; i++) {
        if (s[p[i]] != s[p[i - 1]]) classes++;
        c[0][p[i]] = classes - 1;
    }

    VI pn(n), cn(n);
    cnt.resize(n);
```

```
    for (int h = 0; (1 << h) < n; h++) {
        for (int i = 0; i < n; i++) {
            pn[i] = p[i] - (1 << h);
            if (pn[i] < 0) pn[i] += n;
        }
        fill(cnt.begin(), cnt.end(), 0);

        /// radix sort
        for (int i = 0; i < n; i++)
            cnt[c[h][pn[i]]]++;
        for (int i = 1; i < classes; i++)
            cnt[i] += cnt[i - 1];
        for (int i = n - 1; i >= 0; i--)
            p[--cnt[c[h][pn[i]]]] = pn[i];

        cn[p[0]] = 0;
        classes = 1;

        for (int i = 1; i < n; i++) {
            PII cur = {c[h][p[i]], c[h][(p[i] +
                (1 << h)) % n]};
            PII prev = {c[h][p[i - 1]],
                c[h][(p[i - 1] + (1 << h)) %
                n]};
            if (cur != prev) ++classes;
            cn[p[i]] = classes - 1;
        }
        c.push_back(cn);
    }
    return p;
}

VI suffix_array_construction(string s) {
    s += "!";
    VI sorted_shifts = sort_cyclic_shifts(s);
    sorted_shifts.erase(sorted_shifts.begin());
    return sorted_shifts;
}

/// LCP between the ith and jth (i != j)
    suffix of the STRING
int suffixLCP(int i, int j) {
    assert(i != j);
    int log_n = c.size() - 1;
    int ans = 0;
    for (int k = log_n; k >= 0; k--) {
        if (c[k][i] == c[k][j]) {
            ans += 1 << k;
            i += 1 << k;
            j += 1 << k;
        }
    }
    return ans;
}

VI lcp_construction(const string &s, const VI
    &sa) {
    int n = s.size();
    VI rank(n, 0);
    VI lcp(n - 1, 0);

    for (int i = 0; i < n; i++) rank[sa[i]] =
        i;

    for (int i = 0, k = 0; i < n; i++) {
        if (rank[i] == n - 1) {
            k = 0;
            continue;
        }

        int j = sa[rank[i] + 1];
        while (i + k < n && j + k < n && s[i +
            k] == s[j + k]) k++;
        lcp[rank[i]] = k;
        if (k) k--;
    }
    return lcp;
}

const int MX = 1e6 + 7, K = 20;
int lg[MX];

void pre() {
    lg[1] = 0;
    for (int i = 2; i < MX; i++) lg[i] = lg[i
        / 2] + 1;
}

struct RMQ {
    int N;
    VI v[K];
    RMQ(const VI &a) {
        N = a.size();
        v[0] = a;

        for (int k = 0; (1 << (k + 1)) <= N;
            k++) {
            v[k + 1].resize(N);
            for (int i = 0; i - 1 + (1 << (k +
                1)) < N; i++) {
                v[k + 1][i] = min(v[k][i],
                    v[k][i + (1 << k)]);
            }
        }
    }

    int findMin(int i, int j) {
        int k = lg[j - i + 1];
        return min(v[k][i], v[k][j + 1 - (1 <<
            k)]);
    }
};
```

10.9 304 SuffixAutomata

```
/**
    Linear Time Suffix Automata contruction.
    Build Complexity: O(n * alphabet)
    To achieve better build complexity and
        linear space,
    use map for transitions.
**/

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

const int MAXN = 1e5+7, ALPHA = 26;
int len[2*MAXN], link[2*MAXN],
    nxt[2*MAXN][ALPHA];
int sz;
```

```
int last;

void sa_init() {
    memset(nxt, -1, sizeof nxt);

    len[0] = 0;
    link[0] = -1;
    sz = 1;
    last = 0;
}

void add(char ch) {
    int c = ch-'a';

    int cur = sz++;
        //create new node
    len[cur] = len[last]+1;

    int u = last;
    while (u != -1 && nxt[u][c] == -1) {
        nxt[u][c] = cur;
        u = link[u];
    }

    if (u == -1) {
        link[cur] = 0;
    }
    else {
        int v = nxt[u][c];
        if (len[v] == len[u]+1) {
            link[cur] = v;
        }
        else {
            int clone = sz++;
                //create node by cloning
            len[clone] = 1 + len[u];
            link[clone] = link[v];

            for (int i=0; i<ALPHA; i++)
                nxt[clone][i] = nxt[v][i];

            while (u != -1 && nxt[u][c] == v) {
                nxt[u][c] = clone;
                u = link[u];
            }

            link[v] = link[cur] = clone;
        }
    }
    last = cur;
}

vector<int> edge[2*MAXN];
///Optional, Call after adding all characters
void makeEdge() {
    for (int i=0; i<sz; i++) {
        edge[i].clear();
        for (int j=0; j<ALPHA; j++)
            if (nxt[i][j]!=-1)
                edge[i].push_back(j);
    }
}

// The following code solves SPOJ SUBLEX
// Given a string S, you have to answer some
    queries:
// If all distinct substrings of string S
    were sorted
// lexicographically, which one will be the
    K-th smallest?

long long dp[2*MAXN];
bool vis[2*MAXN];

void dfs(int u) {
    if (vis[u]) return;
    vis[u] = 1;
    dp[u] = 1;
    for (int i: edge[u]) {
        if (nxt[u][i] == -1)  continue;
        dfs(nxt[u][i]);
        dp[u] += dp[nxt[u][i]];
    }
}

void go(int u, long long rem, string &s) {
    if (rem == 1)  return;
    long long sum = 1;
    for (int i: edge[u]) {
        if (nxt[u][i] == -1)  continue;
        if (sum + dp[nxt[u][i]] < rem) {
            sum += dp[nxt[u][i]];
        }
        else {
            s += ('a' + i);
            go(nxt[u][i], rem-sum, s);
            return;
        }
    }
}

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

    string s;
    cin>>s;

    sa_init();
    for (char c: s) add(c);
    makeEdge();


    dfs(0);
    int q;
    cin>>q;

    while (q--) {
        long long x;
        cin>>x;
        x++;
        string s;
        go(0, x, s);
        cout<<s<<"\n";
    }
}
```

10.10 305 PrefixAutomata

```
/*
 * prefix automaton allows insertion to prefix
     function in O(1) time*
 * Author:lel?
```

```
 * /

#include <bits/stdc++.h>
using namespace std;
const int N = 1e6 + 20, M = 26, del = 'a';

struct prefix_automaton {
    int n;
    vector<int> fail;
    vector<array<int, M>> f;

    prefix_automaton(const string &s) :
        n(s.size()), fail(n + 11), f(n + 11,
        {0}) {
        fail[0] = 0, f[0][s[0] - del] = 1;
        for (int i = 1; i < n; i++) {
            for (int j = 0; j < M; j++) {
                if (j == s[i] - del)
                    f[i][j] = i + 1, fail[i] =
                        f[fail[i - 1]][j];
                else
                    f[i][j] = f[fail[i - 1]][j];
            }
        }
    }

    void push(char ch) {
        for (int j = 0; j < 26; j++) {
            if (j == ch - del)
                f[n][j] = n + 1, fail[n] =
                    f[fail[n - 1]][j];
            else
                f[n][j] = f[fail[n - 1]][j];
        }
        n++;
    }
    void push(const string &s) {
        for (auto e : s) push(e);
    }
    // pop until size of string is smaller or
        equal to sz
    void pop(int sz) { n = sz; }
};
```

11 11 Miscellaneous

11.1 101 Boyer-Moore Majority Voting

```
//Finds the element that is present for more
    than N/2 times (if there is any) in O(n)
    time, O(1) space

ll arr[maxN];

int findMajority()
{
    ll major, cnt = 0, n = arr.size();

    for(i = 0; i < n; i++)
    {
        if(cnt == 0)
        {
            major = arr[i];
            cnt = 0;
        }

        if(arr[i] == major)
            cnt++;
        else
            cnt--;
    }

    cnt = 0;
    for(i = 0; i < n; i++)
    {
        if(arr[i] == major)
            cnt++;
    }

    if(cnt <= n/2)
        return -1;
    else
        return major;
}
```