Faculty of Computer and Information Sciences, Ain Shams University: Too Wrong to Pass Too Correct to Fail

Pillow, Isaac, Mostafa, Islam

2021

Contents

1 Combinatorics

1.1 Burnside Lemma

```

 // |Classes|=sum (k ^C(pi))  / |G|

 // C(pi)  the number of cycles in the permutation pi

 // |G| the number of permutations
```

1.2 Catlan Numbers

```
const int MOD = ....
const int MAX = ....
int catalan[MAX];
void init() {
    catalan[0] = catalan[1] = 1;
    for (int i=2; i<=n; i++) {
        catalan[i] = 0;
        for (int j=0; j < i; j++) {
            catalan[i] += (catalan[j] * catalan[i-j-1]) % MOD;
            if (catalan[i] >= MOD) {
                catalan[i] -= MOD;
            }
        }
    }
}

// 1- Number of correct bracket sequence consisting of n opening and n closing brackets.
// 2- The number of rooted full binary trees with n+1 leaves (vertices are not numbered).
//    A rooted binary tree is full if every vertex has either two children or no children.
// 3- The number of ways to completely parenthesize n+1 factors.
// 4- The number of triangulations of a convex polygon with n+2 sides
//    (i.e. the number of partitions of polygon into disjoint triangles by using the diagonals).
// 5- The number of ways to connect the 2n points on a circle to form n disjoint chords.
// 6- The number of non-isomorphic full binary trees with n internal nodes (i.e. nodes having at least one son).
// 7- The number of monotonic lattice paths from point (0,0) to point (n,n) in a square lattice of size nxn,
//    which do not pass above the main diagonal (i.e. connecting (0,0) to (n,n)).
// 8- Number of permutations of length n that can be stack sorted
//    (i.e. it can be shown that the rearrangement is stack sorted if and only if
//    there is no such index i<j<k, such that ak<ai<aj ).
// 9- The number of non-crossing partitions of a set of n elements.
// 10- The number of ways to cover the ladder 1..n using n rectangles
// (The ladder consists of n columns, where ith column has a height i).
```

2 Algebra

2.1 Primitive Roots

```
int powmod (int a, int b, int p) {
    int res = 1;
    while (b)
        if (b & 1)
            res = int (res * 1ll * a % p),  --b;
        else
            a = int (a * 1ll * a % p),  b >>= 1;
    return res;
}

int generator (int p) {
    vector<int> fact;
    int phi = p - 1,  n = phi;
    for (int i = 2; i * i <= n; ++i)
        if (n % i == 0) {
            fact.push_back (i);
            while (n % i == 0)
                n /= i;
        }
    if (n > 1)
        fact.push_back (n);

    for (int res = 2; res <= p; ++res) {
        bool ok = true;
        for (size_t i = 0; i < fact.size() && ok; ++i)
            ok &= powmod (res, phi / fact[i], p) != 1;
        if (ok)  return res;
    }
    return -1;
}
```

2.2 Discrete Logarithm

```
// Returns minimum x for which a ^ x % m = b % m, a and m are coprime.
int solve(int a, int b, int m) {
    a %= m, b %= m;
    int n = sqrt(m) + 1;

    int an = 1;
    for (int i = 0; i < n; ++i)
        an = (an * 1ll * a) % m;

    unordered_map<int, int> vals;
    for (int q = 0, cur = b; q <= n; ++q) {
        vals[cur] = q;
        cur = (cur * 1ll * a) % m;
    }

    for (int p = 1, cur = 1; p <= n; ++p) {
        cur = (cur * 1ll * an) % m;
        if (vals.count(cur)) {
            int ans = n * p - vals[cur];
            return ans;
        }
    }
    return -1;
}

//When a and m are not coprime
// Returns minimum x for which a ^ x % m = b % m.
int solve(int a, int b, int m) {
    a %= m, b %= m;
    int k = 1, add = 0, g;
    while ((g = gcd(a, m)) > 1) {
```

```
        if (b == k)
            return add;
        if (b % g)
            return -1;
        b /= g, m /= g, ++add;
        k = (k * 1ll * a / g) % m;
    }

    int n = sqrt(m) + 1;
    int an = 1;
    for (int i = 0; i < n; ++i)
        an = (an * 1ll * a) % m;

    unordered_map<int, int> vals;
    for (int q = 0, cur = b; q <= n; ++q) {
        vals[cur] = q;
        cur = (cur * 1ll * a) % m;
    }

    for (int p = 1, cur = k; p <= n; ++p) {
        cur = (cur * 1ll * an) % m;
        if (vals.count(cur)) {
            int ans = n * p - vals[cur] + add;
            return ans;
        }
    }
    return -1;
}
```

2.3 Iteration over submasks

```
int s = m;
while (s > 0) {
 ... you can use s ...
 s = (s-1) & m;
}
```

2.4 Totient function

```
void phi_1_to_n(int n) {
    vector<int> phi(n + 1);
    phi[0] = 0;
    phi[1] = 1;
    for (int i = 2; i <= n; i++)
        phi[i] = i;

    for (int i = 2; i <= n; i++) {
        if (phi[i] == i) {
            for (int j = i; j <= n; j += i)
                phi[j] -= phi[j] / i;
        }
    }
}
```

2.5 CRT and EEGCD

```
ll extended(ll a, ll b, ll &x, ll &y) {

    if(b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    ll x0, y0;
    ll g = extended(b, a % b, x0, y0);
    x = y0;
    y = x0 - a / b * y0;

    return g ;
}
ll de(ll a, ll b, ll c, ll &x, ll &y) {

    ll g = extended(abs(a), abs(b), x, y);
    if(c % g) return -1;

    x *= c / g;
    y *= c / g;

    if(a < 0)x = -x;
    if(b < 0)y = -y;
    return g;
}
pair<ll, ll> CRT(vector<ll> r, vector<ll> m) {

    ll r1 = r[0], m1 = m[0];

    for(int i = 1; i < r.size(); i++) {

        ll r2 = r[i], m2 = m[i];
        ll x0, y0;
        ll g = de(m1, -m2, r2 - r1, x0, y0);

        if(g == -1) return {-1, -1} ;

        ll nr = x0 * m1 + r1;
        ll nm = m1 / g * m2;

        r1 = (nr % nm + nm) % nm;
        m1 = nm;
    }
    return {r1, m1};
}
```

2.6 FFT

```
#include<iostream>
#include <bits/stdc++.h>
#define ll long long
#define ld long double
#define rep(i, a, b) for(int i = a; i < (b); ++i)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
#define IO ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
using namespace std;
typedef complex<double> C;
typedef vector<double> vd;
typedef vector<int> vi;
typedef pair<int, int> pii;
void fft(vector<C>& a) {
    int n = sz(a), L = 31 - __builtin_clz(n);
    static vector<complex<long double>> R(2, 1);
    static vector<C> rt(2, 1); // (^ 10% fas te r i f double)
    for (static int k = 2; k < n; k *= 2) {
        R.resize(n);
        rt.resize(n);
        auto x = polar(1.0L, acos(-1.0L) / k);
        rep(i, k, 2 * k) rt[i] = R[i] = i & 1 ? R[i / 2] * x : R[i / 2];
    }
    vi rev(n);
    rep(i, 0, n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    rep(i, 0, n) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2)
        for (int i = 0; i < n; i += 2 * k) rep(j, 0, k) {
            C z = rt[j + k] * a[i + j + k]; //
            a[i + j + k] = a[i + j] - z;
            a[i + j] += z;
        }
}
vd conv(const vd& a, const vd& b) {
```

```
    if (a.empty() || b.empty()) return {};
    vd res(sz(a) + sz(b) - 1);
    int L = 32 - __builtin_clz(sz(res)), n = 1 << L;
    vector<C> in(n), out(n);
    copy(all(a), begin(in));
    rep(i, 0, sz(b)) in[i].imag(b[i]);
    fft(in);
    for (C& x : in) x *= x;
    rep(i, 0, n) out[i] = in[-i & (n - 1)] - conj(in[i]);
    fft(out);
    rep(i, 0, sz(res)) res[i] = imag(out[i]) / (4 * n);
    return res;
}

int main() {
    IO
    //Applications
    //1-All possible sums

    //2-All possible scalar products
    // We are given two arrays a[] and b[] of length n.
    //We have to compute the products of a with every cyclic shift of b.
    //We generate two new arrays of size 2n: We reverse a and append n zeros to it.
    //And we just append b to itself. When we multiply these two arrays as polynomials,
    //and look at the coefficients c[n-1], c[n], ..., c[2n-2] of the product c, we get:
    //c[k]=sum i+j=k  a[i]b[j]

    //3-Two stripes
    //We are given two Boolean stripes (cyclic arrays of values 0 and 1) a and b.
    //We want to find all ways to attach the first stripe to the second one,
    //such that at no position we have a 1 of the first stripe next to a 1 of the second stripe.
}
```

2.7 Fibonacci

```


// F(n-1) * F(n+1) - F(n)^2 = (-1)^n

// F(n+k) = F(k) * F(n+1) + F(k-1) * F(n)

// F(2*n) = F(n) * (F(n+1) + F(n-1))

//GCD ( F(m) , F(n) ) = F(GCD(n,m))
```

2.8 Gauss Determinant

```
const double EPS = 1E-9;
int n;
vector < vector<double> > a (n, vector<double> (n));

double det = 1;
for (int i=0; i<n; ++i) {
    int k = i;
    for (int j=i+1; j<n; ++j)
        if (abs (a[j][i]) > abs (a[k][i]))
            k = j;
    if (abs (a[k][i]) < EPS) {
        det = 0;
        break;
    }
    swap (a[i], a[k]);
    if (i != k)
        det = -det;
    det *= a[i][i];
    for (int j=i+1; j<n; ++j)
        a[i][j] /= a[i][i];
    for (int j=0; j<n; ++j)
        if (j != i && abs (a[j][i]) > EPS)
            for (int k=i+1; k<n; ++k)
                a[j][k] -= a[i][k] * a[j][i];
}

cout << det;
```

2.9 GAUSS SLAE

```
const double EPS = 1e-9;
const int INF = 2; // it doesn't actually have to be infinity or a big number

int gauss (vector < vector<double> > a, vector<double> & ans) {
    int n = (int) a.size();
    int m = (int) a[0].size() - 1;

    vector<int> where (m, -1);
    for (int col = 0, row = 0; col < m && row < n; ++col) {
        int sel = row;
        for (int i = row; i < n; ++i)
            if (abs (a[i][col]) > abs (a[sel][col]))
                sel = i;
        if (abs (a[sel][col]) < EPS)
            continue;
        for (int i = col; i <= m; ++i)
            swap (a[sel][i], a[row][i]);
        where[col] = row;

        for (int i = 0; i < n; ++i)
            if (i != row) {
                double c = a[i][col] / a[row][col];
                for (int j = col; j <= m; ++j)
                    a[i][j] -= a[row][j] * c;
            }
        ++row;
    }

    ans.assign (m, 0);
    for (int i = 0; i < m; ++i)
        if (where[i] != -1)
            ans[i] = a[where[i]][m] / a[where[i]][i];
    for (int i = 0; i < n; ++i) {
        double sum = 0;
        for (int j = 0; j < m; ++j)
            sum += ans[j] * a[i][j];
        if (abs (sum - a[i][m]) > EPS)
            return 0;
    }

    for (int i = 0; i < m; ++i)
        if (where[i] == -1)
            return INF;
    return 1;
}
```

2.10 Matrix Inverse

```
// Sometimes, the questions are complicated - and the answers are simple. //
#pragma GCC optimize ("O3")
#pragma GCC optimize ("unroll-loops")
#include <bits/stdc++.h>
#define ll long long
#define ld  long double
#define IO ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
using namespace std;
vector < vector<double> > gauss (vector < vector<double> > a) {

```

```
    int n = (int) a.size();
    vector<vector<double> > ans(n, vector<double>(n, 0));

    for(int i = 0; i < n; i++)
        ans[i][i] = 1;
    for(int i = 0; i < n; i++) {
        for(int j = i + 1; j < n; j++)
            if(a[j][i] > a[i][i]) {
                swap(a[j], a[i]);
                swap(ans[j], ans[i]);
            }
        double val = a[i][i];
        for(int j = 0; j < n; j++) {
            a[i][j] /= val;
            ans[i][j] /= val;
        }
        for(int j = 0; j < n; j++) {
            if(j == i)continue;
            val = a[j][i];
            for(int k = 0; k < n; k++) {
                a[j][k] -= val * a[i][k];
                ans[j][k] -= val * ans[i][k];
            }
        }
    }
    return ans;
}
int main() {

    IO
    vector<vector<double> > v(3, vector<double> (3) );
    for(int i = 0; i < 3; i++)
        for(int j = 0; j < 3; j++)
            cin >> v[i][j];

    for(auto i : gauss(v)) {
        for(auto j : i)
            cout << j << " ";
        cout << "\n";
    }
}
```

2.11 NTT

```
struct NTT {
    int mod ;
    int root ;
    int root_1 ;
    int root_pw ;

    NTT(int _mod, int primtive_root, int NTT_Len) {

        mod = _mod;
        root_pw = NTT_Len;
        root = fastpower(primtive_root, (mod - 1) / root_pw);
        root_1 = fastpower(root, mod - 2);
    }
    void fft(vector<int> & a, bool invert) {
        int n = a.size();

        for (int i = 1, j = 0; i < n; i++) {
            int bit = n >> 1;
            for (; j & bit; bit >>= 1)
                j ^= bit;
            j ^= bit;

            if (i < j)
                swap(a[i], a[j]);
        }

        for (int len = 2; len <= n; len <<= 1) {
            int wlen = invert ? root_1 : root;
            for (int i = len; i < root_pw; i <<= 1)
                wlen = (int)(1LL * wlen * wlen % mod);


            for (int i = 0; i < n; i += len) {
                int w = 1;
                for (int j = 0; j < len / 2; j++) {
                    int u = a[i + j], v = (int)(1LL * a[i + j + len / 2] * w % mod);
                    a[i + j] = u + v < mod ? u + v : u + v - mod;
                    a[i + j + len / 2] = u - v >= 0 ? u - v : u - v + mod;
                    w = (int)(1LL * w * wlen % mod);
                }
            }
        }

        if (invert) {
            int n_1 = fastpower(n, mod - 2);
            for (int & x : a)
                x = (int)(1LL * x * n_1 % mod);
        }
    }
    vector<int> multiply(vector<int> &a, vector<int> &b) {
        vector<int> fa(a.begin(), a.end()), fb(b.begin(), b.end());
        int n = 1;
        while(n < a.size() + b.size())
            n <<= 1;

        fa.resize(n);
        fb.resize(n);

        fft(fa, 0);
        fft(fb, 0);

        for(int i = 0; i < n; i++)
            fa[i] = 1LL * fa[i] * fb[i] % mod;
        fft(fa, 1);
        return fa;
    }
};
```

2.12 NTT of KACTL

```
///(Note faster than the other NTT)
///If the mod changes don't forget to calculate the primitive root
using ll = long long;
const ll mod = (119 << 23) + 1, root = 3; // = 998244353
// For p < 2^30 there is also e.g. 5 << 25, 7 << 26, 479 << 21
// and 483 << 21 (same root). The last two are > 10^9.
typedef vector<ll> vl;

ll modpow(ll b, ll e) {
    ll ans = 1;
    for (; e; b = b * b % mod, e /= 2)
        if (e & 1) ans = ans * b % mod;
    return ans;
}
void ntt(vl &a) {
    int n = sz(a), L = 31 - __builtin_clz(n);
    static vl rt(2, 1);
    for (static int k = 2, s = 2; k < n; k *= 2, s++) {
        rt.resize(n);
        ll z[] = {1, modpow(root, mod >> s)};
        f(i,k,2*k) rt[i] = rt[i / 2] * z[i & 1] % mod;
    }
    vector<int> rev(n);
    f(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    f(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2)
        for (int i = 0; i < n; i += 2 * k) f(j,0,k) {
            ll z = rt[j + k] * a[i + j + k] % mod, &ai = a[i + j];
            a[i + j + k] = ai - z + (z > ai ? mod : 0);
```

```
            ai += (ai + z >= mod ? z - mod : z);
        }
}
vl conv(const vl &a, const vl &b) {
    if (a.empty() || b.empty()) return {};
    int s = sz(a) + sz(b) - 1, B = 32 - __builtin_clz(s), n = 1 << B;
    int inv = modpow(n, mod - 2);
    vl L(a), R(b), out(n);
    L.resize(n), R.resize(n);
    ntt(L), ntt(R);
    f(i,0,n) out[-i & (n - 1)] = (ll)L[i] * R[i] % mod * inv % mod;
    ntt(out);
    return {out.begin(), out.begin() + s};
}
vector<int> v;
vector<ll> solve(int s, int e) {
    if(s==e) {
        vector<ll> res(2);
        res[0] = 1;
        res[1] = v[s];
        return res;
    }
    int md = (s + e) >> 1;
    return conv(solve(s,md),solve(md+1,e));
}
```

3 Max Flow

4 Matching

5 Trees

6 Strings

7 Geometry

8 Number Theory

9 DP

10 Misc.