

Team notebook

HCMUS-PenguinSpammers

February 3, 2022

Contents					
1 Algorithms	1	5 Geometry	8	9.4 Discrete Logarithm	16
1.1 Mo's Algorithm	1	5.1 Closest Pair Problem	8	9.5 Ext Euclidean	16
1.2 Mo's Algorithms on Trees	1	5.2 Convex Diameter	8	9.6 Highest Exponent Factorial	16
1.3 Parallel Binary Search	1	5.3 Pick Theorem	9	9.7 Miller - Rabin	16
2 Combinatorics	2	5.4 Square	9	9.8 Mod Integer	16
2.1 Factorial Approximate	2	5.5 Triangle	10	9.9 Mod Inv	17
2.2 Factorial	2	6 Graphs	10	9.10 Mod Mul	17
2.3 Fast Fourier Transform	2	6.1 Bridges	10	9.11 Mod Pow	17
2.4 General purpose numbers	3	6.2 Dijkstra	10	9.12 Number Theoretic Transform	17
2.5 Lucas Theorem	3	6.3 Directed MST	11	9.13 Pollard Rho Factorize	17
2.6 Multinomial	4	6.4 Edge Coloring	11	9.14 Primes	17
2.7 Others	4	6.5 Eulerian Path	11	9.15 Totient Sieve	18
2.8 Permutation To Int	4	6.6 Floyd - Warshall	12	9.16 Totient	18
2.9 Sigma Function	4	6.7 Ford - Bellman	12	10 Probability and Statistics	18
3 Data Structures	4	6.8 Gomory Hu	12	10.1 Continuous Distributions	18
3.1 Binary Index Tree	4	6.9 Karp Min Mean Cycle	12	10.1.1 Uniform distribution	18
3.2 Disjoint Set Union (DSU)	5	6.10 Konig's Theorem	13	10.1.2 Exponential distribution	18
3.3 Fake Update	5	6.11 LCA	13	10.1.3 Normal distribution	18
3.4 Fenwick Tree	5	6.12 Math	13	10.2 Discrete Distributions	18
3.5 Hash Table	5	6.13 Push Relabel	13	10.2.1 Binomial distribution	18
3.6 Range Minimum Query	6	6.14 SCC Kosaraju	14	10.2.2 First success distribution	19
3.7 STL Treap	6	6.15 Topological Sort	14	10.2.3 Poisson distribution	19
3.8 Segment Tree	6	7 Linear Algebra	14	10.3 Probability Theory	19
3.9 Sparse Table	6	7.1 Polynomial	14	11 Strings	19
3.10 Trie	7	8 Misc	14	11.1 Hashing	19
4 Dynamic Programming Optimization	7	8.1 Dates	14	11.2 Incremental Aho Corasick	19
4.1 Convex Hull Trick	7	9 Number Theory	15	11.3 KMP	20
4.2 Divide and Conquer	7	9.1 Chinese Remainder Theorem	15	11.4 Minimal String Rotation	20
		9.2 Convolution	15	11.5 Suffix Array	20
		9.3 Diophantine Equations	15	11.6 Suffix Automation	21
				11.7 Suffix Tree	21
				11.8 Z Algorithm	22

1 Algorithms

1.1 Mo's Algorithm

```

/*
  https://www.spoj.com/problems/FREQ2/
*/
vector<int> MoQueries(int n, vector<query> Q){
    block_size = sqrt(n);
    sort(Q.begin(), Q.end(), [](const query &A, const
        query &B){
        return (A.l/block_size != B.l/block_size)?
            (A.l/block_size < B.l/block_size) : (A.r <
                B.r);
    });
    vector<int> res;
    res.resize((int)Q.size());

    int L = 1, R = 0;
    for(query q: Q){
        while (L > q.l) add(--L);
        while (R < q.r) add(++R);

        while (L < q.l) del(L++);
        while (R > q.r) del(R--);

        res[q.pos] = calc(1, R-L+1);
    }
    return res;
}

```

1.2 Mo's Algorithms on Trees

```

/*
Given a tree with N nodes and Q queries. Each node has
an integer weight.
Each query provides two numbers u and v, ask for how
many different integers weight of nodes
there are on path from u to v.

-----
Modify DFS:
-----
For each node u, maintain the start and the end DFS
time. Let's call them ST(u) and EN(u).
=> For each query, a node is considered if its
occurrence count is one.

```

```

-----
Query solving:
-----
Let's query be (u, v). Assume that ST(u) <= ST(v).
Denotes P as LCA(u, v).

Case 1: P = u
Our query would be in range [ST(u), ST(v)].

Case 2: P != u
Our query would be in range [EN(u), ST(v)] + [ST(p),
    ST(p)]
*/

void update(int &L, int &R, int qL, int qR){
    while (L > qL) add(--L);
    while (R < qR) add(++R);

    while (L < qL) del(L++);
    while (R > qR) del(R--);
}

vector<int> MoQueries(int n, vector<query> Q){
    block_size = sqrt((int)nodes.size());
    sort(Q.begin(), Q.end(), [](const query &A, const
        query &B){
        return (ST[A.l]/block_size !=
            ST[B.l]/block_size)? (ST[A.l]/block_size <
                ST[B.l]/block_size) : (ST[A.r] < ST[B.r]);
    });
    vector<int> res;
    res.resize((int)Q.size());

    LCA lca;
    lca.initialize(n);

    int L = 1, R = 0;
    for(query q: Q){
        int u = q.l, v = q.r;
        if(ST[u] > ST[v]) swap(u, v); // assume that
            S[u] <= S[v]
        int parent = lca.get(u, v);

        if(parent == u){
            int qL = ST[u], qR = ST[v];
            update(L, R, qL, qR);
        }else{
            int qL = EN[u], qR = ST[v];
            update(L, R, qL, qR);
            if(cnt_val[a[parent]] == 0)
                res[q.pos] += 1;
        }
    }
}

```

```

        res[q.pos] += cur_ans;
    }
    return res;
}

```

1.3 Parallel Binary Search

```

int lo[N], mid[N], hi[N];
vector<int> vec[N];

void clear() //Reset
{
    memset(bit, 0, sizeof(bit));
}

void apply(int idx) //Apply ith update/query
{
    if(ql[idx] <= qr[idx])
        update(ql[idx], qa[idx]),
            update(qr[idx]+1, -qa[idx]);
    else
    {
        update(1, qa[idx]);
        update(qr[idx]+1, -qa[idx]);
        update(ql[idx], qa[idx]);
    }
}

bool check(int idx) //Check if the condition is
    satisfied
{
    int req=reqd[idx];
    for(auto &it:owns[idx])
    {
        req-=pref(it);
        if(req<0)
            break;
    }
    if(req<=0)
        return 1;
    return 0;
}

void work()
{
    for(int i=1;i<=q;i++)
        vec[i].clear();
    for(int i=1;i<=n;i++)
        if(mid[i]>0)
            vec[mid[i]].push_back(i);

    clear();
}

```

```

for(int i=1;i<=q;i++)
{
    apply(i);
    for(auto &it:vec[i]) //Add appropriate
        check conditions
    {
        if(check(it))
            hi[it]=i;
        else
            lo[it]=i+1;
    }
}

void parallel_binary()
{
    for(int i=1;i<=n;i++)
        lo[i]=1, hi[i]=q+1;
    bool changed = 1;
    while(changed)
    {
        changed=0;
        for(int i=1;i<=n;i++)
        {
            if(lo[i]<hi[i])
            {
                changed=1;
                mid[i]=(lo[i] + hi[i])/2;
            }
            else
                mid[i]=-1;
        }
        work();
    }
}

```

2 Combinatorics

2.1 Factorial Approximate

Approximate Factorial:

$$n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \quad (1)$$

2.2 Factorial

n	1	2	3	4	5	6	7	8	9	10
$n!$	1	2	6	24	120	720	5040	40320	362880	3628800
n	11	12	13	14	15	16	17			
$n!$	4.0e7	4.8e8	6.2e9	8.7e10	1.3e12	2.1e13	3.6e14			
n	20	25	30	40	50	100	150	171		
$n!$	2e18	2e25	3e32	8e47	3e64	9e157	6e262	>DBL_MAX		

2.3 Fast Fourier Transform

```

/**
 * Fast Fourier Transform.
 * Useful to compute convolutions.
 * computes:
 *   C(f star g)[n] = sum_m(f[m] * g[n - m])
 * for all n.
 * test: icpc live archive, 6886 - Golf Bot
 */

```

```

using namespace std;
#include <bits/stdc++.h>
#define D(x) cout << #x " = " << (x) << endl
#define endl '\n'

```

```

const int MN = 262144 << 1;
int d[MN + 10], d2[MN + 10];

```

```

const double PI = acos(-1.0);

```

```

struct cpx {
    double real, image;
    cpx(double _real, double _image) {
        real = _real;
        image = _image;
    }
    cpx(){}
};

```

```

cpx operator + (const cpx &c1, const cpx &c2) {
    return cpx(c1.real + c2.real, c1.image + c2.image);
}

```

```

cpx operator - (const cpx &c1, const cpx &c2) {
    return cpx(c1.real - c2.real, c1.image - c2.image);
}

```

```

cpx operator * (const cpx &c1, const cpx &c2) {

```

```

    return cpx(c1.real*c2.real - c1.image*c2.image,
        c1.real*c2.image + c1.image*c2.real);
}

```

```

int rev(int id, int len) {
    int ret = 0;
    for (int i = 0; (1 << i) < len; i++) {
        ret <<= 1;
        if (id & (1 << i)) ret |= 1;
    }
    return ret;
}

```

```

cpx A[1 << 20];

```

```

void FFT(cpx *a, int len, int DFT) {
    for (int i = 0; i < len; i++)
        A[rev(i, len)] = a[i];
    for (int s = 1; (1 << s) <= len; s++) {
        int m = (1 << s);
        cpx wm = cpx(cos( DFT * 2 * PI / m), sin(DFT * 2 *
            PI / m));
        for(int k = 0; k < len; k += m) {
            cpx w = cpx(1, 0);
            for(int j = 0; j < (m >> 1); j++) {
                cpx t = w * A[k + j + (m >> 1)];
                cpx u = A[k + j];
                A[k + j] = u + t;
                A[k + j + (m >> 1)] = u - t;
                w = w * wm;
            }
        }
    }
    if (DFT == -1) for (int i = 0; i < len; i++)
        A[i].real /= len, A[i].image /= len;
    for (int i = 0; i < len; i++) a[i] = A[i];
    return;
}

```

```

cpx in[1 << 20];

```

```

void solve(int n) {
    memset(d, 0, sizeof d);
    int t;
    for (int i = 0; i < n; ++i) {
        cin >> t;
        d[t] = true;
    }
    int m;
    cin >> m;
    vector<int> q(m);
    for (int i = 0; i < m; ++i)
        cin >> q[i];

```

```

for (int i = 0; i < MN; ++i) {
    if (d[i])
        in[i] = cpx(1, 0);
    else
        in[i] = cpx(0, 0);
}

FFT(in, MN, 1);
for (int i = 0; i < MN; ++i) {
    in[i] = in[i] * in[i];
}
FFT(in, MN, -1);

int ans = 0;
for (int i = 0; i < q.size(); ++i) {
    if (in[q[i]].real > 0.5 || d[q[i]]) {
        ans++;
    }
}
cout << ans << endl;
}

int main() {
    ios_base::sync_with_stdio(false); cin.tie(NULL);
    int n;
    while (cin >> n)
        solve(n);
    return 0;
}

```

2.4 General purpose numbers

Bernoulli numbers

EGF of Bernoulli numbers is $B(t) = \frac{t}{e^t - 1}$ (FFT-able).

$$B[0, \dots] = [1, -\frac{1}{2}, \frac{1}{6}, 0, -\frac{1}{30}, 0, \frac{1}{42}, \dots]$$

Sums of powers:

$$\sum_{i=1}^n i^m = \frac{1}{m+1} \sum_{k=0}^m \binom{m+1}{k} B_k \cdot (n+1)^{m+1-k}$$

Euler-Maclaurin formula for infinite sums:

$$\sum_{i=m}^{\infty} f(i) = \int_m^{\infty} f(x) dx - \sum_{k=1}^{\infty} \frac{B_k}{k!} f^{(k-1)}(m)$$

$$\approx \int_m^{\infty} f(x) dx + \frac{f(m)}{2} - \frac{f'(m)}{12} + \frac{f'''(m)}{720} + O(f^{(5)}(m))$$

Stirling numbers of the first kind

Number of permutations on n items with k cycles.

$$c(n, k) = c(n-1, k-1) + (n-1)c(n-1, k), \quad c(0, 0) = 1$$

$$\sum_{k=0}^n c(n, k) x^k = x(x+1) \dots (x+n-1)$$

$$c(8, k) = 8, 0, 5040, 13068, 13132, 6769, 1960, 322, 28, 1$$

Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

$$S(n, k) = S(n-1, k-1) + kS(n-1, k)$$

$$S(n, 1) = S(n, n) = 1$$

$$S(n, k) = \frac{1}{k!} \sum_{j=0}^k (-1)^{k-j} \binom{k}{j} j^n$$

Eulerian numbers

Number of permutations $\pi \in S_n$ in which exactly k elements are greater than the previous element. k j :s s.t. $\pi(j) > \pi(j+1)$, $k+1$ j :s s.t. $\pi(j) \geq j$, k j :s s.t. $\pi(j) > j$.

$$E(n, k) = (n-k)E(n-1, k-1) + (k+1)E(n-1, k)$$

$$E(n, 0) = E(n, n-1) = 1$$

$$E(n, k) = \sum_{j=0}^k (-1)^j \binom{n+1}{j} (k+1-j)^n$$

Bell numbers

Total number of partitions of n distinct elements. $B(n) = 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, \dots$ For p prime,

$$B(p^m + n) \equiv mB(n) + B(n+1) \pmod{p}$$

Labeled unrooted trees

on n vertices: n^{n-2}

on k existing trees of size n_i : $n_1 n_2 \dots n_k n^{k-2}$

with degrees d_i : $(n-2)! / ((d_1-1)! \dots (d_n-1)!)$

Catalan numbers

$$C_n = \frac{1}{n+1} \binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \frac{(2n)!}{(n+1)!n!}$$

$$C_0 = 1, \quad C_{n+1} = \frac{2(2n+1)}{n+2} C_n, \quad C_{n+1} = \sum C_i C_{n-i}$$

$$C_n = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, \dots$$

[noitemsep]sub-diagonal monotone paths in an $n \times n$ grid. strings with n pairs of parenthesis, correctly nested. binary trees with $n+1$ leaves (0 or 2 children). ordered trees with $n+1$ vertices. ways a convex polygon with $n+2$ sides can be cut into triangles by connecting vertices with straight lines. permutations of $[n]$ with no 3-term increasing subseq.

2.5 Lucas Theorem

For non-negative integers m and n and a prime p , the following congruence relation holds:

$$\binom{m}{n} \equiv \prod_{i=0}^k \binom{m_i}{n_i} \pmod{p},$$

where:

$$m = m_k p^k + m_{k-1} p^{k-1} + \dots + m_1 p + m_0,$$

and:

$$n = n_k p^k + n_{k-1} p^{k-1} + \dots + n_1 p + n_0$$

are the base p expansions of m and n respectively. This uses the convention that $\binom{m}{n} = 0$ if $m \leq n$.

2.6 Multinomial

```

/**
 * Description: Computes  $\displaystyle \binom{k_1 + \dots + k_n}{k_1, k_2, \dots, k_n} = \frac{(\sum k_i)!}{k_1! k_2! \dots k_n!}$ .
 * Status: Tested on kattis:lexicography
 */
#pragma once

long long multinomial(vector<int>& v) {
    long long c = 1, m = v.empty() ? 1 : v[0];
    for (long long i = 1; i < v.size(); i++) {
        for (long long j = 0; j < v[i]; j++) {
            c = c * ++m / (j + 1);
        }
    }
    return c;
}

```

2.7 Others

Cycles Let $g_S(n)$ be the number of n -permutations whose cycle lengths all belong to the set S . Then

$$\sum_{n=0}^{\infty} g_S(n) \frac{x^n}{n!} = \exp \left(\sum_{n \in S} \frac{x^n}{n} \right)$$

Derangements Permutations of a set such that none of the elements appear in their original position.

$$D(n) = (n-1)(D(n-1) + D(n-2)) = nD(n-1) + (-1)^n = \left\lfloor \frac{n!}{e} \right\rfloor$$

Burnside's lemma Given a group G of symmetries and a set X , the number of elements of X up to symmetry equals

$$\frac{1}{|G|} \sum_{g \in G} |X^g|,$$

where X^g are the elements fixed by g ($g.x = x$).

If $f(n)$ counts “configurations” (of some sort) of length n , we can ignore rotational symmetry using $G = Z_n$ to get

$$g(n) = \frac{1}{n} \sum_{k=0}^{n-1} f(\gcd(n, k)) = \frac{1}{n} \sum_{k|n} f(k) \phi(n/k).$$

2.8 Permutation To Int

```
/**
 * Description: Permutation -> integer conversion. (Not
 *             order preserving.)
 * Integer -> permutation can use a lookup table.
 * Time: O(n)
 */
int permToInt(vector<int>& v) {
    int use = 0, i = 0, r = 0;
    for(int x : v) r = r * ++i +
        __builtin_popcount(use & ~(1<<x)),
        use |= 1 << x; // (note:
        minus, not ~!)
    return r;
}
```

2.9 Sigma Function

The Sigma Function is defined as:

$$\sigma_x(n) = \sum_{d|n} d^x$$

when $x = 0$ is called the divisor function, that counts the number of positive divisors of n .

Now, we are interested in find

$$\sum_{d|n} \sigma_0(d)$$

If n is written as prime factorization:

$$n = \prod_{i=1}^k P_i^{e_k}$$

We can demonstrate that:

$$\sum_{d|n} \sigma_0(d) = \prod_{i=1}^k g(e_k + 1)$$

where $g(x)$ is the sum of the first x positive numbers:

$$g(x) = (x * (x + 1)) / 2$$

3 Data Structures

3.1 Binary Index Tree

```
struct BIT {
    int n;
    int t[2 * N];

    void add(int where, long long what) {
        for (where++; where <= n; where += where &
            -where) {
            t[where] += what;
        }
    }

    void add(int from, int to, long long what) {
        add(from, what);
        add(to + 1, -what);
    }
}
```

```
long long query(int where) {
    long long sum = t[0];

    for (where++; where > 0; where -= where &
        -where) {
        sum += t[where];
    }

    return sum;
}
};
```

3.2 Disjoint Set Union (DSU)

```
class DSU{
public:
    vector<int> parent;
    void initialize(int n){
        parent.resize(n+1, -1);
    }

    int findSet(int u){
        while(parent[u] > 0)
            u = parent[u];
        return u;
    }

    void Union(int u, int v){
        int x = parent[u] + parent[v];
        if(parent[u] > parent[v]){
            parent[v] = x;
            parent[u] = v;
        }else{
            parent[u] = x;
            parent[v] = u;
        }
    }
};
```

3.3 Fake Update

```
vector<int> fake_bit[MAXN];

void fake_update(int x, int y, int limit_x){
    for(int i = x; i < limit_x; i += i&(-i))
        fake_bit[i].pb(y);
}
```

```

void fake_get(int x, int y){
    for(int i = x; i >= 1; i -= i&(-i))
        fake_bit[i].pb(y);
}

vector<int> bit[MAXN];

void update(int x, int y, int limit_x, int val){
    for(int i = x; i < limit_x; i += i&(-i)){
        for(int j = lower_bound(fake_bit[i].begin(),
            fake_bit[i].end(), y) -
            fake_bit[i].begin(); j <
            fake_bit[i].size(); j += j&(-j))
            bit[i][j] = max(bit[i][j], val);
        }
    }

int get(int x, int y){
    int ans = 0;
    for(int i = x; i >= 1; i -= i&(-i)){
        for(int j = lower_bound(fake_bit[i].begin(),
            fake_bit[i].end(), y) -
            fake_bit[i].begin(); j >= 1; j -= j&(-j))
            ans = max(ans, bit[i][j]);
        }
    }
    return ans;
}

int main(){
    _io
    int n; cin >> n;
    vector<int> Sx, Sy;
    for(int i = 1; i <= n; i++){
        cin >> a[i].fi >> a[i].se;
        Sx.pb(a[i].fi);
        Sy.pb(a[i].se);
    }
    unique_arr(Sx);
    unique_arr(Sy);
    // unique all value
    for(int i = 1; i <= n; i++){
        a[i].fi = lower_bound(Sx.begin(), Sx.end(),
            a[i].fi) - Sx.begin();
        a[i].se = lower_bound(Sy.begin(), Sy.end(),
            a[i].se) - Sy.begin();
    }

    // do fake BIT update and get operator
    for(int i = 1; i <= n; i++){
        fake_get(a[i].fi-1, a[i].se-1);
        fake_update(a[i].fi, a[i].se, (int)Sx.size());
    }
}

```

```

for(int i = 0; i < Sx.size(); i++){
    fake_bit[i].pb(INT_MIN); // avoid zero
    sort(fake_bit[i].begin(), fake_bit[i].end());
    fake_bit[i].resize(unique(fake_bit[i].begin(),
        fake_bit[i].end()) - fake_bit[i].begin());
    bit[i].resize((int)fake_bit[i].size(), 0);
}

// real update, get operator
int res = 0;
for(int i = 1; i <= n; i++){
    int maxCurLen = get(a[i].fi-1, a[i].se-1) + 1;
    res = max(res, maxCurLen);
    update(a[i].fi, a[i].se, (int)Sx.size(),
        maxCurLen);
}
}

```

3.4 Fenwick Tree

```

template<typename T>
class FenwickTree{
    vector<T> fenw;
    int n;
public:
    void initialize(int _n){
        this->n = _n;
        fenw.resize(n+1);
    }

    void update(int id, T val) {
        while (id <= n) {
            fenw[id] += val;
            id += id&(-id);
        }
    }

    T get(int id){
        T ans{};
        while(id >= 1){
            ans += fenw[id];
            id -= id&(-id);
        }
        return ans;
    }
};

```

3.5 Hash Table

```

/*
 * Micro hash table, can be used as a set.
 * Very efficient vs std::set
 */

const int MN = 1001;
struct ht {
    int _s[(MN + 10) >> 5];
    int len;
    void set(int id) {
        len++;
        _s[id >> 5] |= (1LL << (id & 31));
    }
    bool is_set(int id) {
        return _s[id >> 5] & (1LL << (id & 31));
    }
};

```

3.6 Range Minimum Query

```

/*
    return min(v[a], v[a + 1], ..., v[b - 1]) in
    constant time
*/

template<class T>
struct RMQ {
    vector<vector<T>> jmp;
    RMQ(const vector<T>& V) : jmp(1, V) {
        for (int pw = 1, k = 1; pw * 2 <= sz(V);
            pw *= 2, ++k) {
            jmp.emplace_back(sz(V) - pw * 2 +
                1);
            rep(j, 0, sz(jmp[k]))
                jmp[k][j] = min(jmp[k -
                    1][j], jmp[k - 1][j +
                        pw]);
        }
    }
    T query(int a, int b) {
        assert(a < b); // or return inf if a == b
        int dep = 31 - __builtin_clz(b - a);
        return min(jmp[dep][a], jmp[dep][b - (1
            << dep)]);
    }
};

```

3.7 STL Treap

```

struct Node {
    Node *l = 0, *r = 0;
    int val, y, c = 1;
    Node(int val) : val(val), y(rand()) {}
    void recalc();
};

int cnt(Node* n) { return n ? n->c : 0; }
void Node::recalc() { c = cnt(l) + cnt(r) + 1; }

template<class F> void each(Node* n, F f) {
    if (n) { each(n->l, f); f(n->val); each(n->r, f); }
}

pair<Node*, Node*> split(Node* n, int k) {
    if (!n) return {};
    if (cnt(n->l) >= k) { // "n->val >= k" for
        lower_bound(k)
        auto pa = split(n->l, k);
        n->l = pa.second;
        n->recalc();
        return {pa.first, n};
    } else {
        auto pa = split(n->r, k - cnt(n->l) - 1); // and just "k"
        n->r = pa.first;
        n->recalc();
        return {n, pa.second};
    }
}

Node* merge(Node* l, Node* r) {
    if (!l) return r;
    if (!r) return l;
    if (l->y > r->y) {
        l->r = merge(l->r, r);
        l->recalc();
        return l;
    } else {
        r->l = merge(l, r->l);
        r->recalc();
        return r;
    }
}

Node* ins(Node* t, Node* n, int pos) {
    auto pa = split(t, pos);
    return merge(merge(pa.first, n), pa.second);
}

```

```

// Example application: move the range [l, r) to index k
void move(Node*& t, int l, int r, int k) {
    Node *a, *b, *c;
    tie(a, b) = split(t, l); tie(b, c) = split(b, r - l);
    if (k <= l) t = merge(ins(a, b, k), c);
    else t = merge(a, ins(c, b, k - r));
}

```

3.8 Segment Tree

```

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

const int N = 1e5 + 10;

int node[4*N];

void modify(int seg, int l, int r, int p, int val) {
    if (l == r) {
        node[seg] += val;
        return;
    }
    int mid = (l + r) / 2;
    if (p <= mid) {
        modify(2*seg + 1, l, mid, p, val);
    } else {
        modify(2*seg + 2, mid + 1, r, p, val);
    }
    node[seg] = node[2*seg + 1] + node[2*seg + 2];
}

int sum(int seg, int l, int r, int a, int b) {
    if (l > b || r < a) return 0;
    if (l >= a && r <= b) return node[seg];
    int mid = (l + r) / 2;
    return sum(2*seg + 1, l, mid, a, b) + sum(2*seg + 2, mid + 1, r, a, b);
}

```

3.9 Sparse Table

```

template <typename T, typename func = function<T(const
    T, const T)>>
struct SparseTable {
    func calc;
}

```

```

int n;
vector<vector<T>> ans;

SparseTable() {}

SparseTable(const vector<T>& a, const func& f) :
    n(a.size()), calc(f) {
    int last = trunc(log2(n)) + 1;
    ans.resize(n);
    for (int i = 0; i < n; i++) {
        ans[i].resize(last);
    }
    for (int i = 0; i < n; i++) {
        ans[i][0] = a[i];
    }
    for (int j = 1; j < last; j++) {
        for (int i = 0; i <= n - (1 << j); i++) {
            ans[i][j] = calc(ans[i][j - 1], ans[i +
                (1 << (j - 1))][j - 1]);
        }
    }
}

T query(int l, int r) {
    assert(0 <= l && l <= r && r < n);
    int k = trunc(log2(r - l + 1));
    return calc(ans[l][k], ans[r - (1 << k) + 1][k]);
}
}

```

3.10 Trie

```

const int MN = 26; // size of alphabet
const int MS = 100010; // Number of states.

struct trie {
    struct node {
        int c;
        int a[MN];
    };

    node tree[MS];
    int nodes;

    void clear() {
        tree[nodes].c = 0;
        memset(tree[nodes].a, -1, sizeof tree[nodes].a);
        nodes++;
    }
}

```

```

void init(){
    nodes = 0;
    clear();
}

int add(const string &s, bool query = 0){
    int cur_node = 0;
    for(int i = 0; i < s.size(); ++i){
        int id = gid(s[i]);
        if(tree[cur_node].a[id] == -1){
            if(query) return 0;
            tree[cur_node].a[id] = nodes;
            clear();
        }
        cur_node = tree[cur_node].a[id];
    }
    if(!query) tree[cur_node].c++;
    return tree[cur_node].c;
}
};

```

4 Dynamic Programming Optimization

4.1 Convex Hull Trick

```

#define long long long
#define pll pair<long, long>
#define all(c) c.begin(), c.end()
#define fastio ios_base::sync_with_stdio(false);
    cin.tie(0)

struct line{
    long a, b;
    line() {}
    line(long a, long b) : a(a), b(b) {}
    bool operator < (const line &A) const {
        return pll(a,b) < pll(A.a,A.b);
    }
};

bool bad(line A, line B, line C){
    return (C.b - B.b) * (A.a - B.a) <= (B.b - A.b) *
        (B.a - C.a);
}

void addLine(vector<line> &memo, line cur){
    int k = memo.size();
    while (k >= 2 && bad(memo[k - 2], memo[k - 1],
        cur)){

```

```

        memo.pop_back();
        k--;
    }
    memo.push_back(cur);
}

long Fn(line A, long x){
    return A.a * x + A.b;
}

long query(vector<line> &memo, long x){
    int lo = 0, hi = memo.size() - 1;
    while (lo != hi){
        int mi = (lo + hi) / 2;
        if (Fn(memo[mi], x) > Fn(memo[mi + 1], x)){
            lo = mi + 1;
        }
        else hi = mi;
    }
    return Fn(memo[lo], x);
}

const int N = 1e6 + 1;
long dp[N];

int main()
{
    fastio;
    int n, c; cin >> n >> c;
    vector<line> memo;
    for (int i = 1; i <= n; i++){
        long val; cin >> val;
        addLine(memo, {-2 * val, val * val + dp[i - 1]});
        dp[i] = query(memo, val) + val * val + c;
    }
    cout << dp[n] << '\n';
    return 0;
}

```

4.2 Divide and Conquer

```

/**
 * recurrence:
 *   dp[k][i] = min dp[k-1][j] + c[i][j - 1], for all
 *       j > i;
 *
 * "comp" computes dp[k][i] for all i in O(n log n) (k
 *   is fixed)
 *
 * Problems:

```

```

* https://icpc.kattis.com/problems/branch
* http://codeforces.com/contest/321/problem/E
* */

```

```

void comp(int l, int r, int le, int re) {
    if (l > r) return;

    int mid = (l + r) >> 1;

    int best = max(mid + 1, le);
    dp[cur][mid] = dp[cur ^ 1][best] + cost(mid, best - 1);
    for (int i = best; i <= re; i++) {
        if (dp[cur][mid] > dp[cur ^ 1][i] + cost(mid, i - 1)) {
            best = i;
            dp[cur][mid] = dp[cur ^ 1][i] + cost(mid, i - 1);
        }
    }

    comp(l, mid - 1, le, best);
    comp(mid + 1, r, best, re);
}

```

5 Geometry

5.1 Closest Pair Problem

```

struct point {
    double x, y;
    int id;
    point() {}
    point (double a, double b) : x(a), y(b) {}
};

double dist(const point &o, const point &p) {
    double a = p.x - o.x, b = p.y - o.y;
    return sqrt(a * a + b * b);
}

double cp(vector<point> &p, vector<point> &x,
    vector<point> &y) {
    if (p.size() < 4) {
        double best = 1e100;
        for (int i = 0; i < p.size(); ++i)
            for (int j = i + 1; j < p.size(); ++j)
                best = min(best, dist(p[i], p[j]));
        return best;
    }
}

```



```

int ls = (p.size() + 1) >> 1;
double l = (p[ls - 1].x + p[ls].x) * 0.5;
vector<point> xl(ls), xr(p.size() - ls);
unordered_set<int> left;
for (int i = 0; i < ls; ++i) {
    xl[i] = x[i];
    left.insert(x[i].id);
}
for (int i = ls; i < p.size(); ++i) {
    xr[i - ls] = x[i];
}

vector<point> yl, yr;
vector<point> pl, pr;
yl.reserve(ls); yr.reserve(p.size() - ls);
pl.reserve(ls); pr.reserve(p.size() - ls);
for (int i = 0; i < p.size(); ++i) {
    if (left.count(y[i].id))
        yl.push_back(y[i]);
    else
        yr.push_back(y[i]);

    if (left.count(p[i].id))
        pl.push_back(p[i]);
    else
        pr.push_back(p[i]);
}

double dl = cp(pl, xl, yl);
double dr = cp(pr, xr, yr);
double d = min(dl, dr);
vector<point> yp; yp.reserve(p.size());
for (int i = 0; i < p.size(); ++i) {
    if (fabs(y[i].x - l) < d)
        yp.push_back(y[i]);
}
for (int i = 0; i < yp.size(); ++i) {
    for (int j = i + 1; j < yp.size() && j < i + 7; ++j) {
        d = min(d, dist(yp[i], yp[j]));
    }
}
return d;
}

double closest_pair(vector<point> &p) {
    vector<point> x(p.begin(), p.end());
    sort(x.begin(), x.end(), [](const point &a, const
        point &b) {
        return a.x < b.x;
    });
    vector<point> y(p.begin(), p.end());

```

```

sort(y.begin(), y.end(), [](const point &a, const
    point &b) {
    return a.y < b.y;
});
return cp(p, x, y);
}

```

5.2 Convex Diameter

```

struct point{
    int x, y;
};

struct vec{
    int x, y;
};

vec operator - (const point &A, const point &B){
    return vec{A.x - B.x, A.y - B.y};
}

int cross(vec A, vec B){
    return A.x*B.y - A.y*B.x;
}

int cross(point A, point B, point C){
    int val = A.x*(B.y - C.y) + B.x*(C.y - A.y) +
        C.x*(A.y - B.y);
    if(val == 0)
        return 0; // coline
    if(val < 0)
        return 1; // clockwise
    return -1; //counter clockwise
}

vector<point> findConvexHull(vector<point> points){
    vector<point> convex;
    sort(points.begin(), points.end(), [](const point
        &A, const point &B){
        return (A.x == B.x)? (A.y < B.y): (A.x < B.x);
    });
    vector<point> Up, Down;
    point A = points[0], B = points.back();
    Up.push_back(A);
    Down.push_back(A);

    for(int i = 0; i < points.size(); i++){
        if(i == points.size()-1 || cross(A, points[i],
            B) > 0){
            while(Up.size() > 2 &&
                cross(Up[Up.size()-2], Up[Up.size()-1],

```

```

                points[i]) <= 0)
                Up.pop_back();
            Up.push_back(points[i]);
        }
        if(i == points.size()-1 || cross(A, points[i],
            B) < 0){
            while(Down.size() > 2 &&
                cross(Down[Down.size()-2],
                    Down[Down.size()-1], points[i]) >= 0)
                Down.pop_back();
            Down.push_back(points[i]);
        }
    }
    for(int i = 0; i < Up.size(); i++)
        convex.push_back(Up[i]);
    for(int i = Down.size()-2; i > 0; i--)
        convex.push_back(Down[i]);
    return convex;
}

int dist(point A, point B){
    return (A.x - B.x)*(A.x - B.x) + (A.y - B.y)*(A.y -
        B.y);
}

double findConvexDiameter(vector<point> convexHull){
    int n = convexHull.size();

    int is = 0, js = 0;
    for(int i = 1; i < n; i++){
        if(convexHull[i].y > convexHull[is].y)
            is = i;
        if(convexHull[js].y > convexHull[i].y)
            js = i;
    }

    int maxd = dist(convexHull[is], convexHull[js]);
    int i, maxi, j, maxj;
    i = maxi = is;
    j = maxj = js;
    do{
        int ni = (i+1)%n, nj = (j+1)%n;
        if(cross(convexHull[ni] - convexHull[i],
            convexHull[nj] - convexHull[j]) <= 0){
            j = nj;
        }else{
            i = ni;
        }
    }
    int d = dist(convexHull[i], convexHull[j]);
    if(d > maxd){
        maxd = d;
        maxi = i;
        maxj = j;
    }
}

```

```

    }
    }while(i != is || j != js);
    return sqrt(maxd);
}

```

5.3 Pick Theorem

```

struct point{
    ll x, y;
};

//Pick: S = I + B/2 - 1

ld polygonArea(vector <point> &points){
    int n = (int)points.size();
    ld area = 0.0;
    int j = n-1;
    for(int i = 0; i < n; i++){
        area += (points[j].x + points[i].x) *
                (points[j].y - points[i].y);
        j = i;
    }

    return abs(area/2.0);
}

ll boundary(vector <point> points){
    int n = (int)points.size();
    ll num_bound = 0;
    for(int i = 0; i < n; i++){
        ll dx = (points[i].x - points[(i+1)%n].x);
        ll dy = (points[i].y - points[(i+1)%n].y);
        num_bound += abs(__gcd(dx, dy)) - 1;
    }
    return num_bound;
}

```

5.4 Square

```

typedef long double ld;

const ld eps = 1e-12;
int cmp(ld x, ld y = 0, ld tol = eps) {
    return ( x <= y + tol) ? (x + tol < y) ? -1 : 0 : 1;
}

struct point{

```

```

    ld x, y;
    point(ld a, ld b) : x(a), y(b) {}
    point() {}
};

```

```

struct square{
    ld x1, x2, y1, y2,
    a, b, c;
    point edges[4];
    square(ld _a, ld _b, ld _c) {
        a = _a, b = _b, c = _c;
        x1 = a - c * 0.5;
        x2 = a + c * 0.5;
        y1 = b - c * 0.5;
        y2 = b + c * 0.5;
        edges[0] = point(x1, y1);
        edges[1] = point(x2, y1);
        edges[2] = point(x2, y2);
        edges[3] = point(x1, y2);
    }
};

```

```

ld min_dist(point &a, point &b) {
    ld x = a.x - b.x,
        y = a.y - b.y;
    return sqrt(x * x + y * y);
}

```

```

bool point_in_box(square s1, point p) {
    if (cmp(s1.x1, p.x) != 1 && cmp(s1.x2, p.x) != -1 &&
        cmp(s1.y1, p.y) != 1 && cmp(s1.y2, p.y) != -1)
        return true;
    return false;
}

```

```

bool inside(square &s1, square &s2) {
    for (int i = 0; i < 4; ++i)
        if (point_in_box(s2, s1.edges[i]))
            return true;

```

```

    return false;
}

```

```

bool inside_vert(square &s1, square &s2) {
    if ((cmp(s1.y1, s2.y1) != -1 && cmp(s1.y1, s2.y2) !=
        1) ||
        (cmp(s1.y2, s2.y1) != -1 && cmp(s1.y2, s2.y2) !=
        1))
        return true;
    return false;
}

```

```

bool inside_hori(square &s1, square &s2) {

```

```

    if ((cmp(s1.x1, s2.x1) != -1 && cmp(s1.x1, s2.x2) !=
        1) ||
        (cmp(s1.x2, s2.x1) != -1 && cmp(s1.x2, s2.x2) !=
        1))
        return true;
    return false;
}

```

```

ld min_dist(square &s1, square &s2) {
    if (inside(s1, s2) || inside(s2, s1))
        return 0;

```

```

    ld ans = 1e100;
    for (int i = 0; i < 4; ++i)
        for (int j = 0; j < 4; ++j)
            ans = min(ans, min_dist(s1.edges[i],
                                    s2.edges[j]));

```

```

    if (inside_hori(s1, s2) || inside_hori(s2, s1)) {
        if (cmp(s1.y1, s2.y2) != -1)
            ans = min(ans, s1.y1 - s2.y2);
        else
            if (cmp(s2.y1, s1.y2) != -1)
                ans = min(ans, s2.y1 - s1.y2);
    }

```

```

    if (inside_vert(s1, s2) || inside_vert(s2, s1)) {
        if (cmp(s1.x1, s2.x2) != -1)
            ans = min(ans, s1.x1 - s2.x2);
        else
            if (cmp(s2.x1, s1.x2) != -1)
                ans = min(ans, s2.x1 - s1.x2);
    }

```

```

    return ans;
}

```

5.5 Triangle

Let a, b, c be length of the three sides of a triangle.

$$p = (a + b + c) * 0.5$$

The inradius is defined by:

$$iR = \sqrt{\frac{(p-a)(p-b)(p-c)}{p}}$$

The radius of its circumcircle is given by the formula:

$$cR = \frac{abc}{\sqrt{(a+b+c)(a+b-c)(a+c-b)(b+c-a)}}$$

6 Graphs

6.1 Bridges

```
struct Graph {
    vector<vector<Edge>> g;
    vector<int> vi, low, d, pi, is_b; // vi = visited
    int bridges_computed;
    int ticks, edges;

    Graph(int n, int m) {
        g.assign(n, vector<Edge>());
        id_b.assign(m, 0);
        vi.resize(n);
        low.resize(n);
        d.resize(n);
        pi.resize(n);
        edges = 0;
        bridges_computed = 0;
    }

    void addEge(int u, int v) {
        g[u].push_back(Edge(v, edges));
        g[v].push_back(Edge(u, edges));
        edges++;
    }

    void dfs(int u) {
        vi[u] = true;
        d[u] = low[u] = ticks++;
        for (int i = 0; i < g[u].size(); i++) {
            int v = g[u][i].to;
            if (v == pi[u]) continue;
            if (!vi[v]) {
                pi[v] = u;
                dfs(v);
                if (d[u] < low[v]) is_b[g[u][i].id] = true;
                low[u] = min(low[u], low[v]);
            } else {
                low[u] = min(low[u], low[v]);
            }
        }
    }

    // multiple edges from a to b are not allowed.

```

```
// (they could be detected as a bridge).
// if we need to handle this, just count how many
// edges there are from a to b.
void compBridges() {
    fill(pi.begin(), pi.end(), -1);
    fill(vi.begin(), vi.end(), false);
    fill(d.begin(), d.end(), 0);
    fill(low.begin(), low.end(), 0);
    ticks = 0;
    for (int i = 0; i < g.size(); i++)
        if (!vi[i]) dfs(i);
    bridges_computed = 1;
}

map<int, vector<Edge>> bridgesTree() {
    if (!bridges_computed) compBridges();
    int n = g.size();
    Dsu dsu(n);
    for (int i = 0; i < n; i++)
        for (auto e : g[i])
            if (!is_b[e.id]) dsu.Join(i, e.to);
    map<int, vector<Edge>> tree;
    for (int i = 0; i < n; i++)
        for (auto e : g[i])
            if (is_b[e.id])
                tree[dsu.Find(i)].emplace_back(dsu.Find(e.to),
                e.id);
    return tree;
}
};

```

6.2 Dijkstra

```
struct edge {
    int to;
    long long w;
    edge() {}
    edge(int a, long long b) : to(a), w(b) {}
    bool operator<(const edge &e) const {
        return w > e.w;
    }
};

typedef <vector<vector<edge>> graph;
const long long inf = 1000000LL * 100000000LL;
pair<vector<int>, vector<long long>> dijkstra(graph& g,
    int start) {
    int n = g.size();
    vector<long long> d(n, inf);
    vector<int> p(n, -1);
    d[start] = 0;

```

```
priority_queue<edge> q;
q.push(edge(start, 0));
while (!q.empty()) {
    int node = q.top().to;
    long long dist = q.top().w;
    q.pop();
    if (dist > d[node]) continue;
    for (int i = 0; i < g[node].size(); i++) {
        int to = g[node][i].to;
        long long w_extra = g[node][i].w;
        if (dist + w_extra < d[to]) {
            p[to] = node;
            d[to] = dist + w_extra;
            q.push(edge(to, d[to]));
        }
    }
}
return {p, d};
}

```

6.3 Directed MST

```
struct Edge { int a, b; ll w; };
struct Node { /// lazy skew heap node
    Edge key;
    Node *l, *r;
    ll delta;
    void prop() {
        key.w += delta;
        if (l) l->delta += delta;
        if (r) r->delta += delta;
        delta = 0;
    }
    Edge top() { prop(); return key; }
};

Node *merge(Node *a, Node *b) {
    if (!a || !b) return a ? b;
    a->prop(), b->prop();
    if (a->key.w > b->key.w) swap(a, b);
    swap(a->l, (a->r = merge(b, a->r)));
    return a;
}

void pop(Node*& a) { a->prop(); a = merge(a->l, a->r); }

pair<ll, vi> dmst(int n, int r, vector<Edge>& g) {
    RollbackUF uf(n);
    vector<Node*> heap(n);
    for (Edge e : g) heap[e.b] = merge(heap[e.b],
        new Node{e});
    ll res = 0;
    vi seen(n, -1), path(n), par(n);

```

```

seen[r] = r;
vector<Edge> Q(n), in(n, {-1,-1}), comp;
deque<tuple<int, int, vector<Edge>>> cys;
rep(s,0,n) {
    int u = s, qi = 0, w;
    while (seen[u] < 0) {
        if (!heap[u]) return {-1,{};};
        Edge e = heap[u]->top();
        heap[u]->delta -= e.w,
        pop(heap[u]);
        Q[qi] = e, path[qi++] = u,
        seen[u] = s;
        res += e.w, u = uf.find(e.a);
        if (seen[u] == s) { // found
            cycle, contract
            Node* cyc = 0;
            int end = qi, time =
                uf.time();
            do cyc = merge(cyc, heap[w]
                = path[--qi]);
            while (uf.join(u, w));
            u = uf.find(u), heap[u] =
                cyc, seen[u] = -1;
            cys.push_front({u, time,
                {&Q[qi], &Q[end]}});
        }
    }
    rep(i,0,qi) in[uf.find(Q[i].b)] = Q[i];
}

for (auto& [u,t,comp] : cys) { // restore sol
    (optional)
    uf.rollback(t);
    Edge inEdge = in[u];
    for (auto& e : comp) in[uf.find(e.b)] =
        e;
    in[uf.find(inEdge.b)] = inEdge;
}
rep(i,0,n) par[i] = in[i].a;
return {res, par};
}

```

6.4 Edge Coloring

```

vi edgeColoring(int N, vector<pii> eds) {
    vi cc(N + 1), ret(sz(eds)), fan(N), free(N),
    loc;
    for (pii e : eds) ++cc[e.first], ++cc[e.second];
    int u, v, ncols = *max_element(all(cc)) + 1;
    vector<vi> adj(N, vi(ncols, -1));
    for (pii e : eds) {

```

```

        tie(u, v) = e;
        fan[0] = v;
        loc.assign(ncols, 0);
        int at = u, end = u, d, c = free[u], ind
            = 0, i = 0;
        while (d = free[v], !loc[d] && (v =
            adj[u][d]) != -1) {
            loc[d] = ++ind, cc[ind] = d,
            fan[ind] = v;
            cc[loc[d]] = c;
            for (int cd = d; at != -1; cd ^= c ^ d,
                at = adj[at][cd])
                swap(adj[at][cd], adj[end =
                    at][cd ^ c ^ d]);
            while (adj[fan[i]][d] != -1) {
                int left = fan[i], right =
                    fan[i+1], e = cc[i];
                adj[u][e] = left;
                adj[left][e] = u;
                adj[right][e] = -1;
                free[right] = e;
            }
            adj[u][d] = fan[i];
            adj[fan[i]][d] = u;
            for (int y : {fan[0], u, end})
                for (int& z = free[y] = 0;
                    adj[y][z] != -1; z++);
        }
        rep(i,0,sz(eds))
            for (tie(u, v) = eds[i]; adj[u][ret[i]]
                != v;) ++ret[i];
        return ret;
    }
}

```

6.5 Eulerian Path

```

struct DirectedEulerPath
{
    int n;
    vector<vector<int>> > g;
    vector<int> path;

    void init(int _n){
        n = _n;
        g = vector<vector<int>> > (n + 1,
            vector<int> ());
        path.clear();
    }

    void add_edge(int u, int v){
        g[u].push_back(v);
    }
}

```

```

}

void dfs(int u)
{
    while(g[u].size())
    {
        int v = g[u].back();
        g[u].pop_back();
        dfs(v);
    }
    path.push_back(u);
}

bool getPath(){
    int ctEdges = 0;
    vector<int> outDeg, inDeg;
    outDeg = inDeg = vector<int> (n + 1, 0);
    for(int i = 1; i <= n; i++)
    {
        ctEdges += g[i].size();
        outDeg[i] += g[i].size();
        for(auto &u:g[i])
            inDeg[u]++;
    }
    int ctMiddle = 0, src = 1;
    for(int i = 1; i <= n; i++)
    {
        if(abs(inDeg[i] - outDeg[i]) > 1)
            return 0;
        if(inDeg[i] == outDeg[i])
            ctMiddle++;
        if(outDeg[i] > inDeg[i])
            src = i;
    }
    if(ctMiddle != n && ctMiddle + 2 != n)
        return 0;
    dfs(src);
    reverse(path.begin(), path.end());
    return (path.size() == ctEdges + 1);
}
}

```

6.6 Floyd - Warshall

```

const ll inf = 1LL << 62;
void floydWarshall(vector<vector<ll>>& m) {
    int n = sz(m);
    rep(i,0,n) m[i][i] = min(m[i][i], 0LL);
    rep(k,0,n) rep(i,0,n) rep(j,0,n)
        if (m[i][k] != inf && m[k][j] != inf) {

```

```

        auto newDist = max(m[i][k] +
            m[k][j], -inf);
        m[i][j] = min(m[i][j], newDist);
    }
    rep(k,0,n) if (m[k][k] < 0) rep(i,0,n)
        rep(j,0,n)
            if (m[i][k] != inf && m[k][j] != inf)
                m[i][j] = -inf;
}

```

6.7 Ford - Bellman

```

const ll inf = LLONG_MAX;
struct Ed { int a, b, w, s() { return a < b ? a : -a; } };
struct Node { ll dist = inf; int prev = -1; };

void bellmanFord(vector<Node>& nodes, vector<Ed>& eds,
    int s) {
    nodes[s].dist = 0;
    sort(all(eds), [](Ed a, Ed b) { return a.s() < b.s(); });

    int lim = sz(nodes) / 2 + 2; // /3+100 with
    shuffled vertices
    rep(i,0,lim) for (Ed ed : eds) {
        Node cur = nodes[ed.a], &dest =
            nodes[ed.b];
        if (abs(cur.dist) == inf) continue;
        ll d = cur.dist + ed.w;
        if (d < dest.dist) {
            dest.prev = ed.a;
            dest.dist = (i < lim-1 ? d :
                -inf);
        }
    }
    rep(i,0,lim) for (Ed e : eds) {
        if (nodes[e.a].dist == -inf)
            nodes[e.b].dist = -inf;
    }
}

```

6.8 Gomory Hu

```

#include "PushRelabel.cpp"

typedef array<ll, 3> Edge;
vector<Edge> gomoryHu(int N, vector<Edge> ed) {

```

```

    vector<Edge> tree;
    vi par(N);
    rep(i,1,N) {
        PushRelabel D(N); // Dinic also works
        for (Edge t : ed) D.addEdge(t[0], t[1],
            t[2], t[2]);
        tree.push_back({i, par[i], D.calc(i,
            par[i])});
        rep(j,i+1,N)
            if (par[j] == par[i] &&
                D.leftOfMinCut(j)) par[j] =
                i;
    }
    return tree;
}

```

6.9 Karp Min Mean Cycle

```

/**
 * Finds the min mean cycle, if you need the max mean
 * cycle
 * just add all the edges with negative cost and print
 * ans * -1
 *
 * test: uva, 11090 - Going in Cycle!!
 */

const int MN = 1000;
struct edge{
    int v;
    long long w;
    edge(){} edge(int v, int w) : v(v), w(w) {}
};

long long d[MN][MN];
// This is a copy of g because increments the size
// pass as reference if this does not matter.
int karp(vector<vector<edge> > g) {
    int n = g.size();

    g.resize(n + 1); // this is important

    for (int i = 0; i < n; ++i)
        if (!g[i].empty())
            g[n].push_back(edge(i,0));
    ++n;

    for(int i = 0; i < n; ++i)
        fill(d[i], d[i] + (n+1), INT_MAX);

    d[n - 1][0] = 0;

```

```

    for (int k = 1; k <= n; ++k) for (int u = 0; u < n;
        ++u) {
        if (d[u][k - 1] == INT_MAX) continue;
        for (int i = g[u].size() - 1; i >= 0; --i)
            d[g[u][i].v][k] = min(d[g[u][i].v][k], d[u][k -
                1] + g[u][i].w);
    }

    bool flag = true;

    for (int i = 0; i < n && flag; ++i)
        if (d[i][n] != INT_MAX)
            flag = false;

    if (flag) {
        return true; // return true if there is no a cycle.
    }

    double ans = 1e15;

    for (int u = 0; u + 1 < n; ++u) {
        if (d[u][n] == INT_MAX) continue;
        double W = -1e15;

        for (int k = 0; k < n; ++k)
            if (d[u][k] != INT_MAX)
                W = max(W, (double)(d[u][n] - d[u][k]) / (n -
                    k));

        ans = min(ans, W);
    }

    // printf("%.2lf\n", ans);
    cout << fixed << setprecision(2) << ans << endl;

    return false;
}

```

6.10 Konig's Theorem

In any bipartite graph, the number of edges in a maximum matching equals the number of vertices in a minimum vertex cover

6.11 LCA

```

#include "../Data Structures/RMQ.h"

struct LCA {

```

```

int T = 0;
vi time, path, ret;
RMQ<int> rmq;

LCA(vector<vi>& C) : time(sz(C)),
    rmq((dfs(C,0,-1), ret)) {}
void dfs(vector<vi>& C, int v, int par) {
    time[v] = T++;
    for (int y : C[v]) if (y != par) {
        path.push_back(v),
        ret.push_back(time[v]);
        dfs(C, y, v);
    }
}

int lca(int a, int b) {
    if (a == b) return a;
    tie(a, b) = minmax(time[a], time[b]);
    return path[rmq.query(a, b)];
}
//dist(a,b){return depth[a] + depth[b] -
//2*depth[lca(a,b)];}
};

```

6.12 Math

Number of Spanning Trees

Create an $N \times N$ matrix mat , and for each edge $a \rightarrow b \in G$, do $\text{mat}[a][b]--$, $\text{mat}[b][b]++$ (and $\text{mat}[b][a]--$, $\text{mat}[a][a]++$ if G is undirected). Remove the i th row and column and take the determinant; this yields the number of directed spanning trees rooted at i (if G is undirected, remove any row/column).

Erdős–Gallai theorem

A simple graph with node degrees $d_1 \geq \dots \geq d_n$ exists iff $d_1 + \dots + d_n$ is even and for every $k = 1 \dots n$,

$$\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min(d_i, k).$$

6.13 Push Relabel

```

struct PushRelabel {
    struct Edge {
        int dest, back;
        ll f, c;
    };
};

```

```

vector<vector<Edge>> g;
vector<ll> ec;
vector<Edge*> cur;
vector<vi> hs; vi H;
PushRelabel(int n) : g(n), ec(n), cur(n),
    hs(2*n), H(n) {}

void addEdge(int s, int t, ll cap, ll rcap=0) {
    if (s == t) return;
    g[s].push_back({t, sz(g[t]), 0, cap});
    g[t].push_back({s, sz(g[s])-1, 0, rcap});
}

void addFlow(Edge& e, ll f) {
    Edge &back = g[e.dest][e.back];
    if (!ec[e.dest] && f)
        hs[H[e.dest]].push_back(e.dest);
    e.f += f; e.c -= f; ec[e.dest] += f;
    back.f -= f; back.c += f; ec[back.dest]
        -= f;
}

ll calc(int s, int t) {
    int v = sz(g); H[s] = v; ec[t] = 1;
    vi co(2*v); co[0] = v-1;
    rep(i,0,v) cur[i] = g[i].data();
    for (Edge& e : g[s]) addFlow(e, e.c);
}

```

```

for (int hi = 0;;) {
    while (hs[hi].empty()) if (!hi--)
        return -ec[s];
    int u = hs[hi].back();
    hs[hi].pop_back();
    while (ec[u] > 0) // discharge u
        if (cur[u] == g[u].data()
            + sz(g[u])) {
            H[u] = 1e9;
            for (Edge& e :
                g[u]) if (e.c
                    && H[u] >
                    H[e.dest]+1)
                H[u] =
                    H[e.dest]+1,
                    cur[u]
                    = &e;
            if (++co[H[u]],
                !--co[hi] &&
                hi < v)
                rep(i,0,v)
                    if (hi
                        < H[i]
                        && H[i]
                        < v)

```

```

--co[H[i]],
    H[i]
    =
    v
    +
    1;

    hi = H[u];
} else if (cur[u]->c &&
    H[u] ==
    H[cur[u]->dest]+1)
    addFlow(*cur[u],
        min(ec[u],
            cur[u]->c));
    else ++cur[u];
}
}
bool leftOfMinCut(int a) { return H[a] >=
    sz(g); }
};

```

6.14 SCC Kosaraju

// SCC = Strongly Connected Components

```

struct SCC {
    vector<vector<int>> g, gr;
    vector<bool> used;
    vector<int> order, component;
    int total_components;

    SCC(vector<vector<int>>& adj) {
        g = adj;
        int n = g.size();
        gr.resize(n);
        for (int i = 0; i < n; i++)
            for (auto to : g[i])
                gr[to].push_back(i);

        used.assign(n, false);
        for (int i = 0; i < n; i++)
            if (!used[i])
                GenTime(i);

        used.assign(n, false);
        component.assign(n, -1);
        total_components = 0;
        for (int i = n - 1; i >= 0; i--) {
            int v = order[i];
            if (!used[v]) {
                vector<int> cur_component;
                Dfs(cur_component, v);
            }

```

```

        for (auto node : cur_component)
            component[node] = total_components;
    }
}

void GenTime(int node) {
    used[node] = true;
    for (auto to : g[node])
        if (!used[to])
            GenTime(to);
    order.push_back(node);
}

void Dfs(vector<int>& cur, int node) {
    used[node] = true;
    cur.push_back(node);
    if (!used[to])
        Dfs(cur, to);
}

vector<vector<int>> CondensedGraph() {
    vector<vector<int>> ans(total_components);
    for (int i = 0; i < int(g.size()); i++) {
        for (int to : g[i]) {
            int u = component[i], v = component[to];
            if (u != v)
                ans[u].push_back(v);
        }
    }
    return ans;
}
};

```

6.15 Topological Sort

```

vi topoSort(const vector<vi>& gr) {
    vi indeg(sz(gr)), ret;
    for (auto& li : gr) for (int x : li) indeg[x]++;
    queue<int> q; // use priority_queue for lexic.
    largest ans.
    rep(i,0,sz(gr)) if (indeg[i] == 0) q.push(i);
    while (!q.empty()) {
        int i = q.front(); // top() for priority
        queue
        ret.push_back(i);
        q.pop();
        for (int x : gr[i])
            if (--indeg[x] == 0) q.push(x);
    }
    return ret;
}

```

```

}

```

7 Linear Algebra

7.1 Polynomial

```

struct Poly {
    vector<double> a;
    double operator()(double x) const {
        double val = 0;
        for (int i = sz(a); i--;) (val *= x) +=
            a[i];
        return val;
    }
    void diff() {
        rep(i,1,sz(a)) a[i-1] = i*a[i];
        a.pop_back();
    }
    void divroot(double x0) {
        double b = a.back(), c; a.back() = 0;
        for(int i=sz(a)-1; i--;) c = a[i], a[i]
            = a[i+1]*x0+b, b=c;
        a.pop_back();
    }
};

```

8 Misc

8.1 Dates

```

//
// Time - Leap years
//

// A[i] has the accumulated number of days from months
// previous to i
const int A[13] = { 0, 0, 31, 59, 90, 120, 151, 181,
    212, 243, 273, 304, 334 };
// same as A, but for a leap year
const int B[13] = { 0, 0, 31, 60, 91, 121, 152, 182,
    213, 244, 274, 305, 335 };
// returns number of leap years up to, and including, y
int leap_years(int y) { return y / 4 - y / 100 + y /
    400; }
bool is_leap(int y) { return y % 400 == 0 || (y % 4 ==
    0 && y % 100 != 0); }
// number of days in blocks of years

```

```

const int p400 = 400*365 + leap_years(400);
const int p100 = 100*365 + leap_years(100);
const int p4 = 4*365 + 1;
const int p1 = 365;
int date_to_days(int d, int m, int y)
{
    return (y - 1) * 365 + leap_years(y - 1) +
        (is_leap(y) ? B[m] : A[m]) + d;
}
void days_to_date(int days, int &d, int &m, int &y)
{
    bool top100; // are we in the top 100 years of a 400
    block?
    bool top4; // are we in the top 4 years of a 100
    block?
    bool top1; // are we in the top year of a 4 block?

    y = 1;
    top100 = top4 = top1 = false;

    y += ((days-1) / p400) * 400;
    d = (days-1) % p400 + 1;

    if (d > p100*3) top100 = true, d -= 3*p100, y += 300;
    else y += ((d-1) / p100) * 100, d = (d-1) % p100 + 1;

    if (d > p4*24) top4 = true, d -= 24*p4, y += 24*4;
    else y += ((d-1) / p4) * 4, d = (d-1) % p4 + 1;

    if (d > p1*3) top1 = true, d -= p1*3, y += 3;
    else y += (d-1) / p1, d = (d-1) % p1 + 1;

    const int *ac = top1 && (!top4 || top100) ? B : A;
    for (m = 1; m < 12; ++m) if (d <= ac[m + 1]) break;
    d -= ac[m];
}

```

9 Number Theory

9.1 Chinese Remainder Theorem

```

/**
 * Chinese remainder theorem.
 * Find z such that z % x[i] = a[i] for all i.
 */
long long crt(vector<long long> &a, vector<long long>
    &x) {
    long long z = 0;
    long long n = 1;
    for (int i = 0; i < x.size(); ++i)

```

```

    n *= x[i];

for (int i = 0; i < a.size(); ++i) {
    long long tmp = (a[i] * (n / x[i])) % n;
    tmp = (tmp * mod_inv(n / x[i], x[i])) % n;
    z = (z + tmp) % n;
}

return (z + n) % n;
}

```

9.2 Convolution

```

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

```

```

inline bool is_pow2(LL x) {
    return (x & (x-1)) == 0;
}

```

```

inline int ceil_log2(LL x) {
    int ans = 0;
    --x;
    while (x != 0) {
        x >>= 1;
        ans++;
    }
    return ans;
}

```

```

/* Returns the convolution of the two given vectors in
   time proportional to n*log(n).
   * The number of roots of unity to use nroots_unity
   must be set so that the product of the first
   * nroots_unity primes of the vector nth_roots_unity is
   greater than the maximum value of the
   * convolution. Never use sizes of vectors bigger than
   2^24, if you need to change the values of
   * the nth roots of unity to appropriate primes for
   those sizes.
*/

```

```

vector<LL> convolve(const vector<LL> &a, const
    vector<LL> &b, int nroots_unity = 2) {
    int N = 1 << ceil_log2(a.size() + b.size());
    vector<LL> ans(N, 0), fA(N), fB(N), fC(N);
    LL modulo = 1;
    for (int times = 0; times < nroots_unity; times++) {
        fill(fA.begin(), fA.end(), 0);
        fill(fB.begin(), fB.end(), 0);
        for (int i = 0; i < a.size(); i++) fA[i] = a[i];
        for (int i = 0; i < b.size(); i++) fB[i] = b[i];
    }
}

```

```

LL prime = nth_roots_unity[times].first;
LL inv_modulo = mod_inv(modulo % prime, prime);
LL normalize = mod_inv(N, prime);
ntfft(fA, 1, nth_roots_unity[times]);
ntfft(fB, 1, nth_roots_unity[times]);
for (int i = 0; i < N; i++) fC[i] = (fA[i] * fB[i])
    % prime;
ntfft(fC, -1, nth_roots_unity[times]);
for (int i = 0; i < N; i++) {
    LL curr = (fC[i] * normalize) % prime;
    LL k = (curr - (ans[i] % prime) + prime) % prime;
    k = (k * inv_modulo) % prime;
    ans[i] += modulo * k;
}
modulo *= prime;
}
return ans;
}

```

9.3 Diophantine Equations

```

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

```

```

bool find_any_solution(long long a, long long b, long
    long c, long long &x0,
    long long &y0, long long &g) {
    g = gcd(abs(a), abs(b), x0, y0);
    if (c % g) {
        return false;
    }

    x0 *= c / g;
    y0 *= c / g;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}

```

```

void shift_solution(long long &x, long long &y, long
    long a, long long b,
    long long cnt) {
    x += cnt * b;
    y -= cnt * a;
}

```

```

long long find_all_solutions(long long a, long long b,
    long long c,
    long long minx, long long maxx, long long miny,
    long long maxy) {
    long long x, y, g;
    if (!find_any_solution(a, b, c, x, y, g)) return 0;
    a /= g;
    b /= g;

```

```

    long long sign_a = a > 0 ? +1 : -1;
    long long sign_b = b > 0 ? +1 : -1;

```

```

    shift_solution(x, y, a, b, (minx - x) / b);
    if (x < minx) shift_solution(x, y, a, b, sign_b);
    if (x > maxx) return 0;
    long long lx1 = x;

```

```

    shift_solution(x, y, a, b, (maxx - x) / b);
    if (x > maxx) shift_solution(x, y, a, b, -sign_b);
    long long rx1 = x;

```

```

    shift_solution(x, y, a, b, -(miny - y) / a);
    if (y < miny) shift_solution(x, y, a, b, -sign_a);
    if (y > maxy) return 0;
    long long lx2 = x;

```

```

    shift_solution(x, y, a, b, -(maxy - y) / a);
    if (y > maxy) shift_solution(x, y, a, b, sign_a);
    long long rx2 = x;

```

```

    if (lx2 > rx2) swap(lx2, rx2);
    long long lx = max(lx1, lx2);
    long long rx = min(rx1, rx2);

```

```

    if (lx > rx) return 0;
    return (rx - lx) / abs(b) + 1;
}

```

9.4 Discrete Logarithm

```

// Computes x which a ^ x = b mod n.

```

```

long long d_log(long long a, long long b, long long n) {

```



```

long long mod_pow(long long a, long long exp, long long
mod) {
    long long ans = 1;
    while (exp > 0) {
        if (exp & 1)
            ans = mod_mul(ans, a, mod);
        a = mod_mul(a, a, mod);
        exp >>= 1;
    }
    return ans;
}

```

9.12 Number Theoretic Transform

```

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

```

```

/* The following vector of pairs contains pairs (prime,
generator)
* where the prime has an Nth root of unity for N being
a power of two.
* The generator is a number g s.t g^(p-1)=1 (mod p)
* but is different from 1 for all smaller powers */

```

```

vector<PLL> nth_roots_unity {
    {1224736769,330732430},{1711276033,927759239},{167772161,167489322},
    {469762049,343261969},{754974721,643797295},{1107296257,883865065}
}

```

```

PLL ext_euclid(LL a, LL b) {
    if (b == 0)
        return make_pair(1,0);
    pair<LL,LL> rc = ext_euclid(b, a % b);
    return make_pair(rc.second, rc.first - (a / b) *
rc.second);
}

```

```

//returns -1 if there is no unique modular inverse
LL mod_inv(LL x, LL modulo) {
    PLL p = ext_euclid(x, modulo);
    if ( (p.first * x + p.second * modulo) != 1 )
        return -1;
    return (p.first+modulo) % modulo;
}

```

```

//Number theory fft. The size of a must be a power of 2
void ntfft(vector<LL> &a, int dir, const PLL
&root_unity) {
    int n = a.size();
    LL prime = root_unity.first;
    LL basew = mod_pow(root_unity.second, (prime-1) / n,
prime);

```

```

if (dir < 0) basew = mod_inv(basew, prime);
for (int m = n; m >= 2; m >>= 1) {
    int mh = m >> 1;
    LL w = 1;
    for (int i = 0; i < mh; i++) {
        for (int j = i; j < n; j += m) {
            int k = j + mh;
            LL x = (a[j] - a[k] + prime) % prime;
            a[j] = (a[j] + a[k]) % prime;
            a[k] = (w * x) % prime;
        }
        w = (w * basew) % prime;
    }
    basew = (basew * basew) % prime;
}
int i = 0;
for (int j = 1; j < n - 1; j++) {
    for (int k = n >> 1; k > (i ^= k); k >>= 1);
    if (j < i) swap(a[i], a[j]);
}
}

```

9.13 Pollard Rho Factorize

```

long long pollard_rho(long long n) {
    long long x, y, i = 1, k = 2, d;
    x = y = rand() % n;
    while (1) {
        ++i;
        x = mod_mul(x, x, n);
        x += 2;
        if (x >= n) x -= n;
        if (x == y) return 1;
        d = __gcd(abs(x - y), n);
        if (d != 1) return d;
        if (i == k) {
            y = x;
            k *= 2;
        }
    }
    return 1;
}

```

```

// Returns a list with the prime divisors of n
vector<long long> factorize(long long n) {
    vector<long long> ans;
    if (n == 1)
        return ans;
    if (miller_rabin(n)) {
        ans.push_back(n);

```

```

} else {
    long long d = 1;
    while (d == 1)
        d = pollard_rho(n);
    vector<long long> dd = factorize(d);
    ans = factorize(n / d);
    for (int i = 0; i < dd.size(); ++i)
        ans.push_back(dd[i]);
}
return ans;
}

```

9.14 Primes

```

namespace primes {
    const int MP = 100001;
    bool sieve[MP];
    long long primes[MP];
    int num_p;
    void fill_sieve() {
        num_p = 0;
        sieve[0] = sieve[1] = true;
        for (long long i = 2; i < MP; ++i) {
            if (!sieve[i]) {
                primes[num_p++] = i;
                for (long long j = i * i; j < MP; j += i)
                    sieve[j] = true;
            }
        }
    }

    // Finds prime numbers between a and b, using basic
    primes up to sqrt(b)
    // a must be greater than 1.
    vector<long long> seg_sieve(long long a, long long b)
    {
        long long ant = a;
        a = max(a, 3LL);
        vector<bool> pmap(b - a + 1);
        long long sqrt_b = sqrt(b);
        for (int i = 0; i < num_p; ++i) {
            long long p = primes[i];
            if (p > sqrt_b) break;
            long long j = (a + p - 1) / p;
            for (long long v = (j == 1) ? p + p : j * p; v <=
b; v += p) {
                pmap[v - a] = true;
            }
        }
        vector<long long> ans;
        if (ant == 2) ans.push_back(2);

```

```

int start = a % 2 ? 0 : 1;
for (int i = start, I = b - a + 1; i < I; i += 2)
    if (pmap[i] == false)
        ans.push_back(a + i);
return ans;
}

vector<pair<int, int>> factor(int n) {
    vector<pair<int, int>> ans;
    if (n == 0) return ans;
    for (int i = 0; primes[i] * primes[i] <= n; ++i) {
        if ((n % primes[i]) == 0) {
            int expo = 0;
            while ((n % primes[i]) == 0) {
                expo++;
                n /= primes[i];
            }
            ans.emplace_back(primes[i], expo);
        }
    }

    if (n > 1) {
        ans.emplace_back(n, 1);
    }
    return ans;
}
}

```

9.15 Totient Sieve

```

for (int i = 1; i < MN; i++)
    phi[i] = i;

for (int i = 1; i < MN; i++)
    if (!sieve[i]) // is prime
        for (int j = i; j < MN; j += i)
            phi[j] -= phi[j] / i;

```

9.16 Totient

```

long long totient(long long n) {
    if (n == 1) return 0;
    long long ans = n;
    for (int i = 0; primes[i] * primes[i] <= n; ++i) {
        if ((n % primes[i]) == 0) {
            while ((n % primes[i]) == 0) n /= primes[i];
            ans -= ans / primes[i];
        }
    }
}

```

```

}
if (n > 1) {
    ans -= ans / n;
}
return ans;
}

```

10 Probability and Statistics

10.1 Continuous Distributions

10.1.1 Uniform distribution

If the probability density function is constant between a and b and 0 elsewhere it is $U(a, b)$, $a < b$.

$$f(x) = \begin{cases} \frac{1}{b-a} & a < x < b \\ 0 & \text{otherwise} \end{cases}$$

$$\mu = \frac{a+b}{2}, \sigma^2 = \frac{(b-a)^2}{12}$$

10.1.2 Exponential distribution

The time between events in a Poisson process is $\text{Exp}(\lambda)$, $\lambda > 0$.

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

$$\mu = \frac{1}{\lambda}, \sigma^2 = \frac{1}{\lambda^2}$$

10.1.3 Normal distribution

Most real random values with mean μ and variance σ^2 are well described by $\mathcal{N}(\mu, \sigma^2)$, $\sigma > 0$.

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

If $X_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and $X_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$ then

$$aX_1 + bX_2 + c \sim \mathcal{N}(\mu_1 + \mu_2 + c, a^2\sigma_1^2 + b^2\sigma_2^2)$$

10.2 Discrete Distributions

10.2.1 Binomial distribution

The number of successes in n independent yes/no experiments, each which yields success with probability p is $\text{Bin}(n, p)$, $n = 1, 2, \dots$, $0 \leq p \leq 1$.

$$p(k) = \binom{n}{k} p^k (1-p)^{n-k}$$

$$\mu = np, \sigma^2 = np(1-p)$$

$\text{Bin}(n, p)$ is approximately $\text{Po}(np)$ for small p .

10.2.2 First success distribution

The number of trials needed to get the first success in independent yes/no experiments, each which yields success with probability p is $\text{Fs}(p)$, $0 \leq p \leq 1$.

$$p(k) = p(1-p)^{k-1}, k = 1, 2, \dots$$

$$\mu = \frac{1}{p}, \sigma^2 = \frac{1-p}{p^2}$$

10.2.3 Poisson distribution

The number of events occurring in a fixed period of time t if these events occur with a known average rate κ and independently of the time since the last event is $\text{Po}(\lambda)$, $\lambda = t\kappa$.

$$p(k) = e^{-\lambda} \frac{\lambda^k}{k!}, k = 0, 1, 2, \dots$$

$$\mu = \lambda, \sigma^2 = \lambda$$

10.3 Probability Theory

Let X be a discrete random variable with probability $p_X(x)$ of assuming the value x . It will then have an expected value (mean) $\mu = E(X) = \sum_x x p_X(x)$ and variance $\sigma^2 = V(X) = E(X^2) - (E(X))^2 = \sum_x (x - E(X))^2 p_X(x)$ where σ is the standard deviation. If X is instead continuous it will have a probability density function $f_X(x)$ and the sums above will instead be integrals with $p_X(x)$ replaced by $f_X(x)$.

Expectation is linear:

$$E(aX + bY) = aE(X) + bE(Y)$$

For independent X and Y ,

$$V(aX + bY) = a^2V(X) + b^2V(Y).$$

11 Strings

11.1 Hashing

```
struct H {
    typedef uint64_t ull;
    ull x; H(ull x=0) : x(x) {}
#define OP(O,A,B) H operator O(H o) { ull r = x; asm \
    (A "addq %%rdx, %0\n adcq $0,%0" : "+a"(r) : \
    B); return r; }
    OP(+, "d"(o.x)) OP(*, "mul %1\n", "r"(o.x) : \
    "rdx")
    H operator-(H o) { return *this + ~o.x; }
    ull get() const { return x + !~x; }
    bool operator==(H o) const { return get() == \
    o.get(); }
    bool operator<(H o) const { return get() < \
    o.get(); }
};

static const H C = (11)1e11+3; // (order ~ 3e9; random
    also ok)

struct HashInterval {
    vector<H> ha, pw;
    HashInterval(string& str) : ha(sz(str)+1), \
    pw(ha) {
        pw[0] = 1;
        rep(i,0,sz(str))
            ha[i+1] = ha[i] * C + str[i],
            pw[i+1] = pw[i] * C;
    }
    H hashInterval(int a, int b) { // hash [a, b)
        return ha[b] - ha[a] * pw[b - a];
    }
};

vector<H> getHashes(string& str, int length) {
    if (sz(str) < length) return {};
    H h = 0, pw = 1;
    rep(i,0,length)
        h = h * C + str[i], pw = pw * C;
    vector<H> ret = {h};
    rep(i,length,sz(str)) {
        ret.push_back(h = h * C + str[i] - pw * \
        str[i-length]);
    }
    return ret;
}
```

```
}
H hashString(string& s){H h{}; for(char c:s)
    h=h*C+c;return h;}
```

11.2 Incremental Aho Corasick

```
class IncrementalAhoCorasick {
    static const int Alphabets = 26;
    static const int AlphabetBase = 'a';
    struct Node {
        Node *fail;
        Node *next[Alphabets];
        int sum;
        Node() : fail(NULL), next{}, sum(0) {}
    };

    struct String {
        string str;
        int sign;
    };

public:
    //totalLen = sum of (len + 1)
    void init(int totalLen) {
        nodes.resize(totalLen);
        nNodes = 0;
        strings.clear();
        roots.clear();
        sizes.clear();
        que.resize(totalLen);
    }

    void insert(const string &str, int sign) {
        strings.push_back(String{ str, sign });
        roots.push_back(nodes.data() + nNodes);
        sizes.push_back(1);
        nNodes += (int)str.size() + 1;
        auto check = [&]() { return sizes.size() > 1 && \
        sizes.end()[-1] == sizes.end()[-2]; };
        if(!check())
            makePMA(strings.end() - 1, strings.end(), \
            roots.back(), que);
        while(check()) {
            int m = sizes.back();
            roots.pop_back();
            sizes.pop_back();
            sizes.back() += m;
            if(!check())
                makePMA(strings.end() - m * 2, strings.end(), \
                roots.back(), que);
        }
    }
}
```

```
}
}

int match(const string &str) const {
    int res = 0;
    for(const Node *t : roots)
        res += matchPMA(t, str);
    return res;
}

private:
    static void makePMA(vector<String>::const_iterator \
    begin, vector<String>::const_iterator end, Node \
    *nodes, vector<Node*> &que) {
        int nNodes = 0;
        Node *root = new(&nodes[nNodes ++]) Node();
        for(auto it = begin; it != end; ++ it) {
            Node *t = root;
            for(char c : it->str) {
                Node *&n = t->next[c - AlphabetBase];
                if(n == nullptr)
                    n = new(&nodes[nNodes ++]) Node();
                t = n;
            }
            t->sum += it->sign;
        }
        int qt = 0;
        for(Node *&n : root->next) {
            if(n != nullptr) {
                n->fail = root;
                que[qt ++] = n;
            } else {
                n = root;
            }
        }
        for(int qh = 0; qh != qt; ++ qh) {
            Node *t = que[qh];
            int a = 0;
            for(Node *n : t->next) {
                if(n != nullptr) {
                    que[qt ++] = n;
                    Node *r = t->fail;
                    while(r->next[a] == nullptr)
                        r = r->fail;
                    n->fail = r->next[a];
                    n->sum += r->next[a]->sum;
                }
                ++ a;
            }
        }
    }
}
```

```
static int matchPMA(const Node *t, const string &str)
{
    int res = 0;
    for(char c : str) {
        int a = c - AlphabetBase;
        while(t->next[a] == nullptr)
            t = t->fail;
        t = t->next[a];
        res += t->sum;
    }
    return res;
}
```

```
vector<Node> nodes;
int nNodes;
vector<String> strings;
vector<Node*> roots;
vector<int> sizes;
vector<Node*> que;
};
```

```
int main() {
    int m;
    while(~scanf("%d", &m)) {
        IncrementalAhoCorasic iac;
        iac.init(600000);
        rep(i, m) {
            int ty;
            char s[300001];
            scanf("%d%s", &ty, s);
            if(ty == 1) {
                iac.insert(s, +1);
            } else if(ty == 2) {
                iac.insert(s, -1);
            } else if(ty == 3) {
                int ans = iac.match(s);
                printf("%d\n", ans);
                fflush(stdout);
            } else {
                abort();
            }
        }
        return 0;
    }
}
```

11.3 KMP

```
vi pi(const string& s) {
    vi p(sz(s));
```

```
        rep(i, 1, sz(s)) {
            int g = p[i-1];
            while (g && s[i] != s[g]) g = p[g-1];
            p[i] = g + (s[i] == s[g]);
        }
        return p;
    }

vi match(const string& s, const string& pat) {
    vi p = pi(pat + '\0' + s), res;
    rep(i, sz(p)-sz(s), sz(p))
        if (p[i] == sz(pat)) res.push_back(i - 2
            * sz(pat));
    return res;
}
```

11.4 Minimal String Rotation

```
// Lexicographically minimal string rotation
int lmsr() {
    string s;
    cin >> s;
    int n = s.size();
    s += s;
    vector<int> f(s.size(), -1);
    int k = 0;
    for (int j = 1; j < 2 * n; ++j) {
        int i = f[j - k - 1];
        while (i != -1 && s[j] != s[k + i + 1]) {
            if (s[j] < s[k + i + 1])
                k = j - i - 1;
            i = f[i];
        }
        if (i == -1 && s[j] != s[k + i + 1]) {
            if (s[j] < s[k + i + 1]) {
                k = j;
            }
            f[j - k] = -1;
        } else {
            f[j - k] = i + 1;
        }
    }
    return k;
}
```

11.5 Suffix Array

```
const int MAXN = 200005;
```

```
const int MAX_DIGIT = 256;
void countingSort(vector<int>& SA, vector<int>& RA, int
    k = 0) {
    int n = SA.size();
    vector<int> cnt(max(MAX_DIGIT, n), 0);
    for (int i = 0; i < n; i++)
        if (i + k < n)
            cnt[RA[i + k]]++;
    else
        cnt[0]++;
    for (int i = 1; i < cnt.size(); i++)
        cnt[i] += cnt[i - 1];
    vector<int> tempSA(n);
    for (int i = n - 1; i >= 0; i--)
        if (SA[i] + k < n)
            tempSA[--cnt[RA[SA[i] + k]]] = SA[i];
        else
            tempSA[--cnt[0]] = SA[i];
    SA = tempSA;
}
```

```
vector<int> constructSA(string s) {
    int n = s.length();
    vector<int> SA(n);
    vector<int> RA(n);
    vector<int> tempRA(n);
    for (int i = 0; i < n; i++) {
        RA[i] = s[i];
        SA[i] = i;
    }
    for (int step = 1; step < n; step <= 1) {
        countingSort(SA, RA, step);
        countingSort(SA, RA, 0);
        int c = 0;
        tempRA[SA[0]] = c;
        for (int i = 1; i < n; i++) {
            if (RA[SA[i]] == RA[SA[i - 1]] && RA[SA[i] +
                step] == RA[SA[i - 1] + step])
                tempRA[SA[i]] = tempRA[SA[i - 1]];
            else
                tempRA[SA[i]] = tempRA[SA[i - 1]] + 1;
        }
        RA = tempRA;
        if (RA[SA[n - 1]] == n - 1) break;
    }
    return SA;
}
```

```
vector<int> computeLCP(const string& s, const
    vector<int>& SA) {
    int n = SA.size();
    vector<int> LCP(n), PLCP(n), c(n, 0);
```

```

for (int i = 0; i < n; i++)
    c[SA[i]] = i;
int k = 0;
for (int j, i = 0; i < n-1; i++) {
    if(c[i] - 1 < 0)
        continue;
    j = SA[c[i] - 1];
    k = max(k - 1, 0);
    while (i+k < n && j+k < n && s[i+k] == s[j+k])
        k++;
    PLCP[i] = k;
}
for (int i = 0; i < n; i++)
    LCP[i] = PLCP[SA[i]];
return LCP;
}

```

11.6 Suffix Automation

```

/*
 * Suffix automaton:
 * This implementation was extended to maintain
 * (online) the
 * number of different substrings. This is equivalent
 * to compute
 * the number of paths from the initial state to all
 * the other
 * states.
 *
 * The overall complexity is O(n)
 * can be tested here:
 * https://www.urionlinejudge.com.br/judge/en/problems/view/1530
 */

```

```

struct state {
    int len, link;
    long long num_paths;
    map<int, int> next;
};

```

```

const int MN = 200011;
state sa[MN << 1];
int sz, last;
long long tot_paths;

```

```

void sa_init() {
    sz = 1;
    last = 0;
    sa[0].len = 0;
    sa[0].link = -1;
}

```

```

sa[0].next.clear();
sa[0].num_paths = 1;
tot_paths = 0;
}

void sa_extend(int c) {
    int cur = sz++;
    sa[cur].len = sa[last].len + 1;
    sa[cur].next.clear();
    sa[cur].num_paths = 0;
    int p;
    for (p = last; p != -1 && !sa[p].next.count(c); p =
        sa[p].link) {
        sa[p].next[c] = cur;
        sa[cur].num_paths += sa[p].num_paths;
        tot_paths += sa[p].num_paths;
    }

    if (p == -1) {
        sa[cur].link = 0;
    } else {
        int q = sa[p].next[c];
        if (sa[p].len + 1 == sa[q].len) {
            sa[cur].link = q;
        } else {
            int clone = sz++;
            sa[clone].len = sa[p].len + 1;
            sa[clone].next = sa[q].next;
            sa[clone].num_paths = 0;
            sa[clone].link = sa[q].link;
            for (; p != -1 && sa[p].next[c] == q; p =
                sa[p].link) {
                sa[p].next[c] = clone;
                sa[q].num_paths -= sa[p].num_paths;
                sa[clone].num_paths += sa[p].num_paths;
            }
            sa[q].link = sa[cur].link = clone;
        }
    }
    last = cur;
}

```

11.7 Suffix Tree

```

struct SuffixTree {
    enum { N = 200010, ALPHA = 26 }; // N ~
        2*maxlen+10
    int toi(char c) { return c - 'a'; }
    string a; // v = cur node, q = cur position
    int t[N][ALPHA], l[N], r[N], p[N], s[N], v=0, q=0, m=2;
}

```

```

void ukkadd(int i, int c) { suff:
    if (r[v]<=q) {
        if (t[v][c]==-1) { t[v][c]=m;
            l[m]=i;
            p[m++]=v; v=s[v]; q=r[v];
            goto suff; }
        v=t[v][c]; q=l[v];
    }
    if (q==-1 || c==toi(a[q])) q++; else {
        l[m+1]=i; p[m+1]=m; l[m]=l[v];
        r[m]=q;
        p[m]=p[v]; t[m][c]=m+1;
        t[m][toi(a[q])]=v;
        l[v]=q; p[v]=m;
        t[p[m]][toi(a[l[m]])]=m;
        v=s[p[m]]; q=l[m];
        while (q<r[m]) {
            v=t[v][toi(a[q])];
            q+=r[v]-l[v]; }
        if (q==r[m]) s[m]=v; else
            s[m]=m+2;
        q=r[v]-(q-r[m]); m+=2; goto suff;
    }
}

```

```

SuffixTree(string a) : a(a) {
    fill(r,r+N,sz(a));
    memset(s, 0, sizeof s);
    memset(t, -1, sizeof t);
    fill(t[1],t[1]+ALPHA,0);
    s[0] = 1; l[0] = l[1] = -1; r[0] = r[1]
        = p[0] = p[1] = 0;
    rep(i,0,sz(a)) ukkadd(i, toi(a[i]));
}

```

```

// example: find longest common substring (uses
// ALPHA = 28)
pii best;
int lcs(int node, int i1, int i2, int olen) {
    if (l[node] <= i1 && i1 < r[node])
        return 1;
    if (l[node] <= i2 && i2 < r[node])
        return 2;
    int mask = 0, len = node ? olen +
        (r[node] - l[node]) : 0;
    rep(c,0,ALPHA) if (t[node][c] != -1)
        mask |= lcs(t[node][c], i1, i2,
            len);
    if (mask == 3)
        best = max(best, {len, r[node] -
            len});
    return mask;
}

```

```

static pii LCS(string s, string t) {
    SuffixTree st(s + (char)('z' + 1) + t +
        (char)('z' + 2));
    st.lcs(0, sz(s), sz(s) + 1 + sz(t), 0);
    return st.best;
}
};

```

11.8 Z Algorithm

```

vector<int> compute_z(const string &s){
    int n = s.size();
    vector<int> z(n,0);
    int l,r;
    r = l = 0;

```

```

for(int i = 1; i < n; ++i){
    if(i > r) {
        l = r = i;
        while(r < n and s[r - 1] == s[r])r++;
        z[i] = r - l;r--;
    }else{
        int k = i-1;
        if(z[k] < r - i +1) z[i] = z[k];
        else {
            l = i;
            while(r < n and s[r - 1] == s[r])r++;
            z[i] = r - l;r--;
        }
    }
}
return z;
}

```

```

int main(){

    //string line;cin>>line;
    string line = "alfalfa";
    vector<int> z = compute_z(line);

    for(int i = 0; i < z.size(); ++i ){
        if(i)cout<<" ";
        cout<<z[i];
    }
    cout<<endl;

    // must print "0 0 0 4 0 0 1"

    return 0;
}

```
