

Contents

1 DataStructure

- 1.1 Treap
- 1.2 Dynamic Segment Tree

2 Math

- 2.1 FFT
- 2.2 NTT
- 2.3 Gaussian-Jordan
- 2.4 Mu
- 2.5 Lucas
- 2.6 Inv
- 2.7 CRT
- 2.8 Generator
- 2.9 Count Primes
- 2.10 Pollard Rho
- 2.11 Formula
 - 2.11.1 Dirichlet Convolution
 - 2.11.2 Burnside's Lemma
 - 2.11.3 Pick Theorem
 - 2.11.4 Fermat's Little Theorem
 - 2.11.5 Wilson's Theorem
 - 2.11.6 Legendre Theorem
 - 2.11.7 Kummer Theorem
 - 2.11.8 ext-Kummer Theorem
 - 2.11.9 Factorial with mod
 - 2.11.10 Properties of nCr with mod
 - 2.11.11 ext-Lucas' Theorem
 - 2.11.12 Catalan Number
 - 2.11.13 modinv table
- 2.12 Matrix

3 String

- 3.1 KMP
- 3.2 Longest Palindrome
- 3.3 Z

4 Graph

- 4.1 one-out-degree (CSES Planets Cycles)
- 4.2 Dijkstra
- 4.3 MaximumFlow
- 4.4 SCC
- 4.5 2-SAT(CSES Giant Pizza)

5 DP

- 5.1 Li-Chao Segment Tree
- 5.2 CHO

6 Geometry

- 6.1 Intersect
- 6.2 Inside
- 6.3 Minimum Euclidean Distance
- 6.4 Convex Hull

7 Tree

- 7.1 Heavy Light Decomposition (modify and query on path)
- 7.2 LCA

8 Misc

- 8.1 Tri Search

1. DataStructure

1.1. Treap

```
1 #define pii pair<int, int>
2 struct node {
3     int tag = 0;
4     int sum = 0;
5     int prio = rand();
6     int lson = 0;
7     int rson = 0;
8     int si = 0;
9     int val = 0;
10 };
11 node treap[400005];
12 int cnt = 0;
13 int root = 0;
14
15 void update(int index) {
16     int lson = treap[index].lson;
```

```
17     int rson = treap[index].rson;
18     treap[index].si = treap[lson].si + treap[rson].si + 1;
19     treap[index].sum = treap[lson].sum;
20     treap[index].sum += treap[rson].sum;
21     treap[index].sum += treap[index].val;
22 }
23 void push(int index) {
24     if (!treap[index].tag)
25         return;
26     swap(treap[index].lson, treap[index].rson);
27     int lson = treap[index].lson;
28     int rson = treap[index].rson;
29     treap[lson].tag ^= 1;
30     treap[rson].tag ^= 1;
31     treap[index].tag = 0;
32 }
33
34 pii split(int rk, int index) {
35     if (!index)
36         return {0, 0};
37     push(index);
38     int lson = treap[index].lson;
39     int rson = treap[index].rson;
40     if (rk <= treap[lson].si) {
41         pii temp = split(rk, lson);
42         treap[index].lson = temp.second;
43         update(index);
44         return {temp.first, index};
45     } else {
46         pii temp = split(rk - treap[lson].si - 1, rson);
47         treap[index].rson = temp.first;
48         update(index);
49         return {index, temp.second};
50     }
51 }
52
53 int merge(int x, int y) {
54     if (!x && !y)
55         return 0;
56     if (!x && y)
57         return y;
58     if (x && !y)
59         return x;
60     push(x);
61     push(y);
62     if (treap[x].prio < treap[y].prio) {
63         treap[x].rson = merge(treap[x].rson, y);
64         update(x);
65         return x;
66     } else {
67         treap[y].lson = merge(x, treap[y].lson);
68         update(y);
69         return y;
70     }
71 }
72
73 void insert(int x, int v) {
74     pii temp = split(x - 1, root);
75     cnt++;
76     treap[cnt].val = v;
77     update(cnt);
78     temp.first = merge(temp.first, cnt);
79     root = merge(temp.first, temp.second);
80 }
81
82 int query(int l, int r) {
83     pii R = split(r, root);
84     pii L = split(l - 1, R.first);
85     int ret = treap[L.second].sum;
86     R.first = merge(L.first, L.second);
87     root = merge(R.first, R.second);
88     return ret;
89 }
90
91 void modify(int l, int r) {
92     pii R = split(r, root);
93     pii L = split(l - 1, R.first);
94     treap[L.second].tag ^= 1;
95     R.first = merge(L.first, L.second);
96     root = merge(R.first, R.second);
97 }
```

1.2. Dynamic Segment Tree

```
1
2 #define int long long
3 using namespace std;
4
5 int n, q;
6 struct node {
7     int data, lson, rson, tag;
8     int rv() { return data + tag; }
9 };
```

```

11 node tree[20000005];
12 int a[200005];
13 int now = 1;
14 int mx = 1000000005;
15
16 void push(int index) {
17     if (!tree[index].lson) {
18         tree[index].lson = ++now;
19     }
20     if (!tree[index].rson) {
21         tree[index].rson = ++now;
22     }
23     int lson = tree[index].lson;
24     int rson = tree[index].rson;
25     tree[lson].tag += tree[index].tag;
26     tree[rson].tag += tree[index].tag;
27     tree[index].data = tree[index].rv();
28     tree[index].tag = 0;
29 }
30
31 void modify(int l, int r, int L, int R, int val, int index) {
32     if (l == L && r == R) {
33         tree[index].tag += val;
34         return;
35     }
36     int mid = (l + r) >> 1;
37     push(index);
38     int lson = tree[index].lson;
39     int rson = tree[index].rson;
40     if (R <= mid) {
41         modify(l, mid, L, R, val, lson);
42     } else if (L > mid) {
43         modify(mid + 1, r, L, R, val, rson);
44     } else {
45         modify(l, mid, L, mid, val, lson);
46         modify(mid + 1, r, mid + 1, R, val, rson);
47     }
48     tree[index].data = tree[lson].rv() + tree[rson].rv();
49 }
50
51 int query(int l, int r, int L, int R, int index) {
52     // cout << L << " " << R << "\n";
53     if (l == L && r == R) {
54         return tree[index].rv();
55     }
56     int mid = (l + r) >> 1;
57     push(index);
58     int lson = tree[index].lson;
59     int rson = tree[index].rson;
60     if (R <= mid) {
61         return query(l, mid, L, R, lson);
62     }
63     if (L > mid) {
64         return query(mid + 1, r, L, R, rson);
65     }
66     return query(l, mid, L, mid, lson) + query(mid + 1, r, mid + 1, R, rson);
67 }
68
69 signed main() {
70     ios::sync_with_stdio(0);
71     cin.tie(0);
72     cout.tie(0);
73     cin >> n >> q;
74     for (int i = 1; i <= n; i++) {
75         cin >> a[i];
76         modify(1, mx, a[i], a[i], 1, 1);
77     }
78     while (q--) {
79         char mode;
80         int x, y;
81         cin >> mode;
82         if (mode == '?') {
83             cin >> x >> y;
84             cout << query(1, mx, x, y, 1) << "\n";
85         } else {
86             cin >> x >> y;
87             modify(1, mx, a[x], a[x], -1, 1);
88             a[x] = y;
89             modify(1, mx, a[x], a[x], 1, 1);
90         }
91     }
92 }

```

2. Math

2.1. FFT

```

1 using namespace std;
2 inline int read() {
3     int ans = 0;

```

```

5     char c = getchar();
6     while (!isdigit(c))
7         c = getchar();
8     while (isdigit(c)) {
9         ans = ans * 10 + c - '0';
10        c = getchar();
11    }
12    return ans;
13 }
14
15 typedef complex<double> comp;
16 const int MAXN = 1000005;
17 const comp I(0, 1);
18 const double PI = acos(-1);
19 comp A[MAXN * 3], B[MAXN * 3], tmp[MAXN * 3], ans[MAXN * 3];
20 void fft(comp F[], int N, int sgn) {
21     if (N == 1)
22         return;
23     memcp(tmp, F, sizeof(comp) * N);
24     for (int i = 0; i < N; i++)
25         *(i % 2 ? F + i / 2 + N / 2 : F + i / 2) = tmp[i];
26     fft(F, N / 2, sgn), fft(F + N / 2, N / 2, sgn);
27     comp *G = F, *H = F + N / 2;
28     comp cur = 1, step = exp(2 * PI / N * sgn * I);
29     for (int k = 0; k < N / 2; k++) {
30         tmp[k] = G[k] + cur * H[k];
31         tmp[k + N / 2] = G[k] - cur * H[k];
32         cur *= step;
33     }
34     memcp(F, tmp, sizeof(comp) * N);
35 }
36
37 int main() {
38     int n = read(), m = read(), N = 1 << __lg(n + m + 1) + 1;
39     for (int i = 0; i <= n; i++)
40         A[i] = read();
41     for (int i = 0; i <= m; i++)
42         B[i] = read();
43     fft(A, N), fft(B, N);
44     for (int i = 0; i < N; i++)
45         ans[i] = A[i] * B[i];
46     fft(ans, N, -1);
47     for (int i = 0; i <= n + m; i++)
48         printf("%d ", int(ans[i].real() / N + 0.1));
49     return 0;
50 }

```

2.2. NTT

```

1 #define ll long long
2 using namespace std;
3
4 const int MAXN = 1000005;
5 const int MOD = 998244353, G = 3;
6 int rev[MAXN * 3];
7
8 int qpow(int x, int y) {
9     int ret = 1;
10    while (y) {
11        if (y & 1) {
12            ret *= x;
13            ret %= MOD;
14        }
15        x *= x;
16        x %= MOD;
17        y >>= 1;
18    }
19    return ret;
20 }
21
22 void ntt(int F[], int N, int sgn) {
23     int bit = __lg(N);
24     for (int i = 0; i < N; i++) {
25         rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (bit - 1));
26         if (i < rev[i])
27             swap(F[i], F[rev[i]]);
28     }
29     for (int l = 1, t = 1; l < N; l <= 1, t++) {
30         int step = qpow(G, ((MOD - 1) >> t) * sgn + MOD - 1);
31         for (int i = 0; i < N; i += l << 1) {
32             for (int k = i, cur = 1; k < i + l; ++k) {
33                 int g = F[k], h = (ll)F[k + l] * cur % MOD;
34                 F[k] = (g + h) % MOD;
35                 F[k + l] = ((g - h) % MOD + MOD) % MOD;
36                 cur = (ll)cur * step % MOD;
37             }
38         }
39     }
40     if (sgn == -1) {
41         int invN = qpow(N, MOD - 2);
42         for (int i = 0; i < N; i++)
43             F[i] = (ll)F[i] * invN % MOD;
44     }
45 }

```

2.3. Gaussian-Jordan

```

1  #define int long long
2  using namespace std;
3
4  int n;
5  double a[105][105];
6
7  // n <= m
8  void gaussian(double a[105][105], int n, int m) {
9      int curi = 0;
10     for (int j = 0; j < m; j++) {
11         int i;
12         for (i = curi; i < n; i++) {
13             if (a[i][j]) {
14                 break;
15             }
16         }
17         if (a[i][j] == 0)
18             continue;
19         for (int k = 0; k < m; k++) {
20             swap(a[i][k], a[curi][k]);
21         }
22         for (int k = m - 1; k >= j; k--) {
23             a[curi][k] /= a[curi][j];
24         }
25         for (int i = 0; i < n; ++i) {
26             if (i != curi) {
27                 for (int k = m - 1; k >= j; k--) {
28                     a[i][k] -= a[curi][k] * a[i][j];
29                 }
30             }
31         }
32         curi++;
33     }
34 }

```

2.4. Mu

```

1  vector<int> prime;
2  bitset<1000005> vis;
3  int n;
4  int mu[1000005];
5
6  void init() {
7      for (int i = 2; i <= n; i++) {
8          if (!vis[i]) {
9              prime.push_back(i);
10             mu[i] = 1;
11         }
12         for (int p : prime) {
13             if (i * p > n)
14                 break;
15             vis[i * p] = 1;
16             if (i % p == 0) {
17                 mu[i * p] = 0;
18                 break;
19             } else {
20                 mu[i * p] = mu[i] * mu[p];
21             }
22         }
23     }
24 }

```

2.5. Lucas

```

1  int fact[100005];
2  int p;
3
4  void init() {
5      fact[0] = 1;
6      for (int i = 1; i <= p; i++) {
7          fact[i] = fact[i - 1] * i % p;
8      }
9  }
10
11 int inv(int x, int p) {
12     if (x == 1)
13         return 1;
14     return (p - p / x) * inv(p % x, p) % p;
15 }
16
17 int c(int x, int y, int p) {
18     if (x < y)
19         return 0;
20     int k = fact[x] * inv(fact[y], p) % p;
21     return k * inv(fact[x - y], p) % p;
22 }
23
24 int lucas(int x, int y, int p) {
25     if (x == 0)
26         return 1;

```

```

27     return lucas(x / p, y / p, p) % p * c(x % p, y % p, p) % p;
28 }

```

2.6. Inv

```

1  int exgcd(int a, int b, int &x, int &y) {
2      if (b == 0) {
3          x = 1;
4          y = 0;
5          return a;
6      }
7      int d = exgcd(b, a % b, y, x);
8      y -= x * (a / b);
9      return d;
10 }
11
12 int inv(int a, int p) {
13     int x, y;
14     exgcd(a, p, x, y);
15     return (x % p + p) % p;
16 }

```

2.7. CRT

```

1  #define int long long
2  using namespace std;
3
4  int n;
5  int a[15];
6  int b[15];
7  int mul = 1;
8
9  void exgcd(int a, int b, int &x, int &y) {
10     if (b == 0) {
11         x = 1;
12         y = 0;
13         return;
14     }
15     exgcd(b, a % b, y, x);
16     y -= (a / b) * x;
17 }
18
19 int inv(int a, int p) {
20     int x, y;
21     exgcd(a, p, x, y);
22     return x;
23 }
24
25 int ans = 0;
26
27 signed main() {
28     cin >> n;
29     for (int i = 1; i <= n; i++) {
30         cin >> a[i] >> b[i];
31         mul *= a[i];
32     }
33     for (int i = 1; i <= n; i++) {
34         ans += inv(mul / a[i], a[i]) * (mul / a[i]) % mul * b[i] % mul;
35         ans %= mul;
36     }
37     ans = (ans + mul) % mul;
38     cout << ans;
39 }

```

2.8. Generator

```

1  #define int long long
2  using namespace std;
3
4  int t;
5  int n, d;
6  bitset<1000005> exist;
7  bitset<1000005> vis;
8  vector<int> prime;
9  int phi[1000005];
10
11 void init() {
12     phi[1] = 1;
13     for (int i = 2; i <= 1000000; i++) {
14         if (!vis[i]) {
15             prime.push_back(i);
16             phi[i] = i - 1;
17         }
18         for (int j : prime) {
19             if (i * j > 1000000)
20                 break;
21             vis[i * j] = 1;
22             if (i % j == 0) {
23                 phi[i * j] = phi[i] * j;
24                 break;
25             } else {

```

```

27     phi[i * j] = phi[i] * phi[j];
28 }
29 }
30 }
31 exist[2] = exist[4] = 1;
32 for (int i : prime) {
33     if (i == 2)
34         continue;
35     for (int j = i; j <= 1000000; j *= i) {
36         exist[j] = 1;
37         if (j * 2 <= 1000000) {
38             exist[j * 2] = 1;
39         }
40     }
41 }
42 }
43
44 vector<int> factors(int x) {
45     vector<int> v;
46     for (int i = 1; i * i <= x; i++) {
47         if (x % i == 0) {
48             v.push_back(i);
49             if (i * i != x) {
50                 v.push_back(x / i);
51             }
52         }
53     }
54     return v;
55 }
56
57 int f(int x, int y, int mod) {
58     int ret = 1;
59     while (y) {
60         if (y & 1) {
61             ret *= x;
62             ret %= mod;
63         }
64         x *= x;
65         x %= mod;
66         y >>= 1;
67     }
68     return (ret % mod + mod) % mod;
69 }
70
71 vector<int> findroot(int x) {
72     vector<int> ret;
73     if (!exist[x])
74         return ret;
75     int phix = phi[x];
76     vector<int> fact = factors(phix);
77     int fst;
78     for (int i = 1; i <= phix; i++) {
79         if (__gcd(i, x) != 1)
80             continue;
81         bool ok = 1;
82         for (int j : fact) {
83             if (j != phix && f(i, j, x) == 1) {
84                 ok = 0;
85                 break;
86             }
87         }
88         if (ok) {
89             fst = i;
90             break;
91         }
92     }
93     int now = fst;
94     // cout << fst << "\n";
95     for (int i = 1; i <= phix; i++) {
96         if (__gcd(i, phix) == 1) {
97             ret.push_back(now);
98         }
99         now *= fst;
100        now %= x;
101    }
102    return ret;
103 }
104
105 signed main() {
106     ios::sync_with_stdio(0);
107     cin.tie(0);
108     cout.tie(0);
109     init();
110     cin >> t;
111     while (t--) {
112         cin >> n >> d;
113         vector<int> v = findroot(n);
114         sort(v.begin(), v.end());
115         cout << v.size() << "\n";
116         for (int i = 0; i < v.size(); i++) {
117             if (i % d == d - 1) {
118                 cout << v[i] << " ";
119             }

```

```

121     }
122     cout << "\n";
123 }

```

2.9. Count Primes

```

1 using namespace std;
2 using i64 = long long;
3 i64 count_pi(i64 N) {
4     if (N <= 1)
5         return 0;
6     int v = sqrt(N + 0.5);
7     int n_4 = sqrt(v + 0.5);
8     int T = min((int)sqrt(n_4 * 2, n_4);
9     int K = pow(N, 0.625) / log(N) * 2;
10    K = max(K, v);
11    K = min<i64>(K, N);
12    int B = N / K;
13    B = N / (N / B);
14    B = min<i64>(N / (N / B), K);
15
16    vector<i64> l(v + 1);
17    vector<int> s(K + 1);
18    vector<bool> e(K + 1);
19    vector<int> w(K + 1);
20    for (int i = 1; i <= v; ++i)
21        l[i] = N / i - 1;
22    for (int i = 1; i <= v; ++i)
23        s[i] = i - 1;
24
25    const auto div = [](i64 n, int d) -> int { return double(n) / d; };
26    int p;
27    for (p = 2; p <= T; ++p)
28        if (s[p] != s[p - 1]) {
29            i64 M = N / p;
30            int t = v / p, t0 = s[p - 1];
31            for (int i = 1; i <= t; ++i)
32                l[i] -= l[i * p] - t0;
33            for (int i = t + 1; i <= v; ++i)
34                l[i] -= s[div(M, i)] - t0;
35            for (int i = v, j = t; j >= p; --j)
36                for (int l = j * p; i >= l; --i)
37                    s[i] -= s[j] - t0;
38            for (int i = p * p; i <= K; i += p)
39                e[i] = 1;
40        }
41    e[1] = 1;
42    int cnt = 1;
43    vector<int> roughs(B + 1);
44    for (int i = 1; i <= B; ++i)
45        if (!e[i])
46            roughs[cnt++] = i;
47    roughs[cnt] = 0x7fffffff;
48    for (int i = 1; i <= K; ++i)
49        w[i] = e[i] + w[i - 1];
50    for (int i = 1; i <= K; ++i)
51        s[i] = w[i] - w[i - (i & -i)];
52
53    const auto query = [&](int x) -> int {
54        int sum = x;
55        while (x)
56            sum -= s[x], x ^= x & -x;
57        return sum;
58    };
59    const auto add = [&](int x) -> void {
60        e[x] = 1;
61        while (x <= K)
62            ++s[x], x += x & -x;
63    };
64    cnt = 1;
65    for (; p <= n_4; ++p)
66        if (!e[p]) {
67            i64 q = i64(p) * p, M = N / p;
68            while (cnt < q)
69                w[cnt] = query(cnt), cnt++;
70            int t1 = B / p, t2 = min<i64>(B, M / q), t0 = query(p - 1);
71            for (; i <= t1; i = roughs[++id])
72                l[i] -= l[i * p] - t0;
73            for (; i <= t2; i = roughs[++id])
74                l[i] -= query(div(M, i)) - t0;
75            for (; i <= B; i = roughs[++id])
76                l[i] -= w[div(M, i)] - t0;
77            for (int i = q; i <= K; i += p)
78                if (!e[i])
79                    add(i);
80        }
81    while (cnt <= v)
82        w[cnt] = query(cnt), cnt++;
83
84    vector<int> primes;

```

```

87 primes.push_back(1);
88 for (int i = 2; i <= v; ++i)
89     if (!e[i])
90         primes.push_back(i);
91 l[1] += i64(w[v] + w[n_4] - 1) * (w[v] - w[n_4]) / 2;
92 for (int i = w[n_4] + 1; i <= w[B]; ++i)
93     l[1] -= l[primes[i]];
94 for (int i = w[B] + 1; i <= w[v]; ++i)
95     l[1] -= query(N / primes[i]);
96 for (int i = w[n_4] + 1; i <= w[v]; ++i) {
97     int q = primes[i];
98     i64 M = N / q;
99     int e = w[M / q];
100     if (e <= i)
101         break;
102     l[1] += e - i;
103     i64 t = 0;
104     int m = w[sqrt(M + 0.5)];
105     for (int k = i + 1; k <= m; ++k)
106         t += w[div(M, primes[k])];
107     l[1] += 2 * t - (i + m) * (m - i);
108 }
109 return l[1];
}

```

2.10. Pollard Rho

```

1 using namespace std;
2 #define LL long long
3 #define uLL __uint128_t
4 #define sub(a, b) ((a) < (b) ? (b) - (a) : (a) - (b))
5 template <class T, class POW> void fastpow(T x, POW n, POW p, T &ans) {
6     for (; n >= 1) {
7         if (n & 1) {
8             ans *= x;
9             ans %= p;
10        }
11        x *= x;
12        x %= p;
13    }
14 }
15 /*input x, n, p, ans, will modify ans to x ^ n % p
16 the first is x, ans and the second is n, p (LL or __int128)
17 */
18 uLL pri[7] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022}; /*2^64*/
19 // int p[3]={2,7,61};/*2^32*/
20 bool check(const uLL x, const uLL p) {
21     uLL d = x - 1, ans = 1;
22     fastpow(p, d, x, ans);
23     if (ans != 1)
24         return 1;
25     for (; !(d & 1);) {
26         d >>= 1;
27         ans = 1;
28         fastpow(p, d, x, ans);
29         if (ans == x - 1)
30             return 0;
31         else if (ans != 1)
32             return 1;
33     }
34     return 0;
35 }
36 bool miller_rabin(const uLL x) {
37     if (x == 1)
38         return 0;
39     for (auto e : pri) {
40         if (e >= x)
41             return 1;
42         if (check(x, e))
43             return 0;
44     }
45     return 1;
46 }
47 template <class T> T gcd(T a, T b) {
48     if (!a)
49         return b;
50     if (!b)
51         return a;
52     if (a & b & 1)
53         return gcd(sub(a, b), min(a, b));
54     if (a & 1)
55         return gcd(a, b >> 1);
56     if (b & 1)
57         return gcd(a >> 1, b);
58     return gcd(a >> 1, b >> 1) << 1;
59 }
60 /*gcd(a,b) denote gcd(a, 0) = a*/
61 mt19937 rnd(time(0));
62 template <class T> T f(T x, T c, T mod) {
63     return (((uLL)x) * x % mod + c) % mod;
64 }
65 template <class T> T rho(T n) {

```

```

67     T mod = n, x = rnd() % mod, c = rnd() % (mod - 1) + 1, p = 1;
68     for (T i = 2, j = 2, d = x;; ++i) {
69         x = f(x, c, mod);
70         p = ((uLL)p * sub(x, d) % mod);
71         if (i % 127 == 0 && gcd(p, n) != 1)
72             return gcd(p, n);
73         if (i == j) {
74             j <<= 1, d = x;
75             if (gcd(p, n) != 1)
76                 return gcd(p, n);
77         }
78     }
79 }
80 template <class T> T pollard_rho(T n) {
81     if (miller_rabin(n))
82         return n;
83     T p = n;
84     while (p == n)
85         p = rho(n);
86     return max(pollard_rho(p), pollard_rho(n / p));
87 }
88 int main() {
89     LL t, n, ans;
90     for (cin >> t; t--;) {
91         cin >> n;
92         ans = pollard_rho(n);
93         if (ans == n)
94             puts("Prime");
95         else
96             printf("%lld\n", ans);
97     }
98 }

```

2.11. Formula

2.11.1. Dirichlet Convolution

$$\varepsilon = \mu * 1$$

$$\varphi = \mu * \text{Id}$$

2.11.2. Burnside's Lemma

Let X be a set and G be a group that acts on X . For $g \in G$, denote by X^g the elements fixed by g :

$$X^g = \{x \in X \mid gx \in X\}$$

Then

$$|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|.$$

2.11.3. Pick Theorem

$$A = i + \frac{b}{2} - 1$$

2.11.4. Fermat's Little Theorem

$$(a + b)^p \equiv a + b \equiv a^p + b^p \pmod{p}$$

2.11.5. Wilson's Theorem

$$(p - 1)! \equiv -1 \pmod{p}$$

2.11.6. Legendre Theorem

$v(n)$:= power of p in n

$$(n)_p := \frac{n}{p^{v(n)}}$$

$s(n)$:= sum of all digits of n in base p

$$v(n!) = \sum_{i=1}^{\infty} \lfloor \frac{n}{p^i} \rfloor = \frac{n - s(n)}{p - 1}$$

2.11.7. Kummer Theorem

$$v\left(\binom{n}{m}\right) = \frac{s(n) + s(m - n) - s(m)}{p - 1}$$

2.11.8. ext-Kummer Theorem

$$v\left(\binom{n}{m_1, m_2, \dots, m_k}\right) = \frac{\sum_{i=1}^k s(m_i) - s(n)}{p - 1}$$

2.11.9. Factorial with mod

$(n!)_p \equiv -1^{\lfloor \frac{n}{p} \rfloor} ((\lfloor \frac{n}{p} \rfloor)!)_p ((n \% p)!) \pmod{p}$ $O(p + \log_p(n))$ with factorial table.

2.11.10. Properties of nCr with mod

If any i in base p satisfies $n_i < m_i$, then $\binom{n_i}{m_i} \% p = 0$. Therefore

$\binom{n}{m} = \prod_{i=0}^{\max(\log_p(a), \log_p(b))} \binom{n_i}{m_i} \% p$ so $\binom{n}{m} \% p = 0$. If $p = 2$, then $\binom{n}{m}$ is odd \Leftrightarrow any bit in $n < m$. Lucas' theorem can be derived from this generating function method without relying on Fermat's Little Theorem. It is also true for polynomials.

2.11.11. ext-Lucas' Theorem

For any $k \in$ positive number, calculate $\binom{n}{m} \% k$ can decompose k by Fundamental Theorem of Arithmetic. And then use crt.

2.11.12. Catalan Number

$C_0 = C_1 = 1$, if $n > 1$ then $C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k} = \frac{\binom{2n}{n+1}}{n+1}$. Also the number of legal placements of n pairs of brackets is C_n . If there are any k kinds of brackets available, then $k^n C_n$.

2.11.13. modinv table

$p = i * (p/i) + p \% i, -p \% i = i * (p/i), inv(i) = -(p/i) * inv(p \% i)$

2.12. Matrix

```

1 #define int long long
2 using namespace std;
3
4 template <class T> T extgcd(T a, T b, T &x, T &y) {
5     if (!b) {
6         x = 1;
7         y = 0;
8         return a;
9     }
10    T ans = extgcd(b, a % b, y, x);
11    y -= a / b * x;
12    return ans;
13 }
14
15 template <class T> T modeq(T a, T b, T p) {
16    T x, y, d = extgcd(a, p, x, y);
17    if (b % d)
18        return 0;
19    return ((b / d * x) % p + p) % p;
20 }
21
22 template <class T> class Matrix {
23     static const T MOD = 1000000007;
24
25 public:
26     vector<vector<T>> v;
27     Matrix(int n, int m, int identity) {
28         v = vector<vector<T>>(n, vector<T>(m, 0));
29         if (identity)
30             for (int i = 0, k = min(n, m); i < k; ++i)
31                 v[i][i] = 1;
32     }
33     Matrix(Matrix &b) { v = b.v; }
34     void in(int l = 0, int m = -1, int u = 0, int n = -1) {
35         if (n < 0)
36             n = v.size();
37         if (m < 0)
38             m = v[0].size();
39         for (int i = u; i < n; ++i)
40             for (int j = l; j < m; ++j)
41                 scanf("%lld", &v[i][j]);
42     }
43     Matrix(int n, int m) {
44         v = vector<vector<T>>(n, vector<T>(m, 0));
45         in();
46     }
47     void out(int l = 0, int m = -1, int u = 0, int n = -1) {
48         if (n < 0)
49             n = v.size();
50         if (m < 0)
51             m = v[0].size();
52         for (int i = u; i < n; ++i)
53             for (int j = l; j < m; ++j)
54                 printf("%lld%c", v[i][j], " \n"[j == m - 1]);
55     }
56     Matrix operator=(Matrix &b) {
57         v = b.v;
58         return *this;
59     }
60     Matrix operator+(Matrix &b) {
61         Matrix ans(*this);
62         int n = v.size(), m = v[0].size();
63         for (int i = 0; i < n; ++i)
64             for (int j = 0; j < m; ++j) {
65                 ans.v[i][j] += b.v[i][j];
66                 if (MOD) {
67                     if (ans.v[i][j] < 0)
68                         ans.v[i][j] = (ans.v[i][j] % MOD + MOD) % MOD;
69                     if (ans.v[i][j] >= MOD)
70                         ans.v[i][j] %= MOD;
71                 }
72             }
73         return ans;
74     }
75 }
```

```

76 Matrix operator+(T x) {
77     Matrix ans(*this);
78     int n = v.size(), m = v[0].size();
79     for (int i = 0; i < n; ++i)
80         for (int j = 0; j < m; ++j) {
81             ans.v[i][j] += x;
82             if (MOD) {
83                 if (ans.v[i][j] < 0)
84                     ans.v[i][j] = (ans.v[i][j] % MOD + MOD) % MOD;
85                 if (ans.v[i][j] >= MOD)
86                     ans.v[i][j] %= MOD;
87             }
88         }
89     return ans;
90 }
91 Matrix operator-(Matrix &b) {
92     Matrix ans(*this);
93     int n = v.size(), m = v[0].size();
94     for (int i = 0; i < n; ++i)
95         for (int j = 0; j < m; ++j) {
96             ans.v[i][j] -= b.v[i][j];
97             if (MOD) {
98                 if (ans.v[i][j] < 0)
99                     ans.v[i][j] = (ans.v[i][j] % MOD + MOD) % MOD;
100                 if (ans.v[i][j] >= MOD)
101                     ans.v[i][j] %= MOD;
102             }
103         }
104     return ans;
105 }
106 Matrix operator-(T x) {
107     Matrix ans(*this);
108     int n = v.size(), m = v[0].size();
109     for (int i = 0; i < n; ++i)
110         for (int j = 0; j < m; ++j) {
111             ans.v[i][j] -= x;
112             if (MOD) {
113                 if (ans.v[i][j] < 0)
114                     ans.v[i][j] = (ans.v[i][j] % MOD + MOD) % MOD;
115                 if (ans.v[i][j] >= MOD)
116                     ans.v[i][j] %= MOD;
117             }
118         }
119     return ans;
120 }
121 Matrix operator+=(Matrix &b) {
122     int n = v.size(), m = v[0].size();
123     for (int i = 0; i < n; ++i)
124         for (int j = 0; j < m; ++j) {
125             v[i][j] += b.v[i][j];
126             if (MOD) {
127                 if (v[i][j] < 0)
128                     v[i][j] = (v[i][j] % MOD + MOD) % MOD;
129                 if (v[i][j] >= MOD)
130                     v[i][j] %= MOD;
131             }
132         }
133     return *this;
134 }
135 Matrix operator+=(T x) {
136     int n = v.size(), m = v[0].size();
137     for (int i = 0; i < n; ++i)
138         for (int j = 0; j < m; ++j) {
139             v[i][j] += x;
140             if (MOD) {
141                 if (v[i][j] < 0)
142                     v[i][j] = (v[i][j] % MOD + MOD) % MOD;
143                 if (v[i][j] >= MOD)
144                     v[i][j] %= MOD;
145             }
146         }
147     return *this;
148 }
149 Matrix operator-=(Matrix &b) {
150     int n = v.size(), m = v[0].size();
151     for (int i = 0; i < n; ++i)
152         for (int j = 0; j < m; ++j) {
153             v[i][j] -= b.v[i][j];
154             if (MOD) {
155                 if (v[i][j] < 0)
156                     v[i][j] = (v[i][j] % MOD + MOD) % MOD;
157                 if (v[i][j] >= MOD)
158                     v[i][j] %= MOD;
159             }
160         }
161     return *this;
162 }
163 Matrix operator-=(T x) {
164     int n = v.size(), m = v[0].size();
165     for (int i = 0; i < n; ++i)
166         for (int j = 0; j < m; ++j) {
167             v[i][j] -= x;
168             if (MOD) {

```



```

169         if (v[i][j] < 0)
171             v[i][j] = (v[i][j] % MOD + MOD) % MOD;
173         if (v[i][j] >= MOD)
175             v[i][j] %= MOD;
177     }
179     return *this;
181 }
183 Matrix operator*(Matrix &b) {
185     int n = v.size();
187     int p = b.v.size();
189     int m = b.v[0].size();
191     Matrix ans(n, m, 0);
193     for (int i = 0; i < n; ++i)
195         for (int k = 0; k < p; ++k)
197             for (int j = 0; j < m; ++j) {
199                 ans.v[i][j] += v[i][k] * b.v[k][j];
201                 if (MOD) {
203                     if (ans.v[i][j] < 0)
205                         ans.v[i][j] = (ans.v[i][j] % MOD + MOD) % MOD;
207                     if (ans.v[i][j] >= MOD)
209                         ans.v[i][j] %= MOD;
211                 }
213             }
215     return ans;
217 }
219 Matrix operator*(T x) {
221     Matrix ans(*this);
223     int n = v.size(), m = v[0].size();
225     for (int i = 0; i < n; ++i)
227         for (int j = 0; j < m; ++j) {
229             ans.v[i][j] *= x;
231             if (MOD) {
233                 if (ans.v[i][j] < 0)
235                     ans.v[i][j] = (ans.v[i][j] % MOD + MOD) % MOD;
237                 if (ans.v[i][j] >= MOD)
239                     ans.v[i][j] %= MOD;
241             }
243         }
245     return ans;
247 }
249 Matrix operator+=(Matrix &b) {
251     int n = v.size();
253     int p = b.v.size();
255     int m = b.v[0].size();
257     Matrix ans(n, m, 0);
259     for (int i = 0; i < n; ++i)
261         for (int k = 0; k < p; ++k)
263             for (int j = 0; j < m; ++j) {
265                 ans.v[i][j] += v[i][k] * b.v[k][j];
267                 if (MOD) {
269                     if (ans.v[i][j] < 0)
271                         ans.v[i][j] = (ans.v[i][j] % MOD + MOD) % MOD;
273                     if (ans.v[i][j] >= MOD)
275                         ans.v[i][j] %= MOD;
277                 }
279             }
281     v = ans.v;
283     return *this;
285 }
287 Matrix operator+=(T x) {
289     int n = v.size(), m = v[0].size();
291     for (int i = 0; i < n; ++i)
293         for (int j = 0; j < m; ++j) {
295             v[i][j] *= x;
297             if (MOD) {
299                 if (v[i][j] < 0)
301                     v[i][j] = (v[i][j] % MOD + MOD) % MOD;
303                 if (v[i][j] >= MOD)
305                     v[i][j] %= MOD;
307             }
309         }
311     return *this;
313 }
315 Matrix operator/=(T x) {
317     int n = v.size(), m = v[0].size();
319     for (int i = 0; i < n; ++i)
321         for (int j = 0; j < m; ++j) {
323             v[i][j] /= x;
325             if (MOD) {
327                 if (v[i][j] < 0)
329                     v[i][j] = (v[i][j] % MOD + MOD) % MOD;
331                 if (v[i][j] >= MOD)
333                     v[i][j] %= MOD;
335             }
337         }
339     return *this;
341 }
343 Matrix operator/=(Matrix &b) {
345     int n = v.size();
347     int p = b.v.size();
349     int m = b.v[0].size();
351     Matrix ans(n, m, 0);
353     for (int i = 0; i < n; ++i)
355         for (int k = 0; k < p; ++k)
357             for (int j = 0; j < m; ++j) {
359                 ans.v[i][j] += v[i][k] * b.v[k][j];
361                 if (MOD) {
363                     if (ans.v[i][j] < 0)
365                         ans.v[i][j] = (ans.v[i][j] % MOD + MOD) % MOD;
367                     if (ans.v[i][j] >= MOD)
369                         ans.v[i][j] %= MOD;
371                 }
373             }
375     v = ans.v;
377     return *this;
379 }
381 Matrix operator/=(T x) {
383     int n = v.size(), m = v[0].size();
385     for (int i = 0; i < n; ++i)
387         for (int j = 0; j < m; ++j) {
389             v[i][j] /= x;
391             if (MOD) {
393                 if (v[i][j] < 0)
395                     v[i][j] = (v[i][j] % MOD + MOD) % MOD;
397                 if (v[i][j] >= MOD)
399                     v[i][j] %= MOD;
401             }
403         }
405     return *this;
407 }
409 Matrix operator/=(Matrix &b) {
411     int n = v.size();
413     int p = b.v.size();
415     int m = b.v[0].size();
417     Matrix ans(n, m, 0);
419     for (int i = 0; i < n; ++i)
421         for (int k = 0; k < p; ++k)
423             for (int j = 0; j < m; ++j) {
425                 ans.v[i][j] += v[i][k] * b.v[k][j];
427                 if (MOD) {
429                     if (ans.v[i][j] < 0)
431                         ans.v[i][j] = (ans.v[i][j] % MOD + MOD) % MOD;
433                     if (ans.v[i][j] >= MOD)
435                         ans.v[i][j] %= MOD;
437                 }
439             }
441     v = ans.v;
443     return *this;
445 }
447 Matrix operator/=(T x) {
449     int n = v.size(), m = v[0].size();
451     for (int i = 0; i < n; ++i)
453         for (int j = 0; j < m; ++j) {
455             v[i][j] /= x;
457             if (MOD) {
459                 if (v[i][j] < 0)
461                     v[i][j] = (v[i][j] % MOD + MOD) % MOD;
463                 if (v[i][j] >= MOD)
465                     v[i][j] %= MOD;
467             }
469         }
471     return *this;
473 }
475 Matrix operator/=(Matrix &b) {
477     int n = v.size();
479     int p = b.v.size();
481     int m = b.v[0].size();
483     Matrix ans(n, m, 0);
485     for (int i = 0; i < n; ++i)
487         for (int k = 0; k < p; ++k)
489             for (int j = 0; j < m; ++j) {
491                 ans.v[i][j] += v[i][k] * b.v[k][j];
493                 if (MOD) {
495                     if (ans.v[i][j] < 0)
497                         ans.v[i][j] = (ans.v[i][j] % MOD + MOD) % MOD;
499                     if (ans.v[i][j] >= MOD)
501                         ans.v[i][j] %= MOD;
503                 }
505             }
507     v = ans.v;
509     return *this;
511 }
513 Matrix operator/=(T x) {
515     int n = v.size(), m = v[0].size();
517     for (int i = 0; i < n; ++i)
519         for (int j = 0; j < m; ++j) {
521             v[i][j] /= x;
523             if (MOD) {
525                 if (v[i][j] < 0)
527                     v[i][j] = (v[i][j] % MOD + MOD) % MOD;
529                 if (v[i][j] >= MOD)
531                     v[i][j] %= MOD;
533             }
535         }
537     return *this;
539 }
541 Matrix operator/=(Matrix &b) {
543     int n = v.size();
545     int p = b.v.size();
547     int m = b.v[0].size();
549     Matrix ans(n, m, 0);
551     for (int i = 0; i < n; ++i)
553         for (int k = 0; k < p; ++k)
555             for (int j = 0; j < m; ++j) {
557                 ans.v[i][j] += v[i][k] * b.v[k][j];
559                 if (MOD) {
561                     if (ans.v[i][j] < 0)
563                         ans.v[i][j] = (ans.v[i][j] % MOD + MOD) % MOD;
565                     if (ans.v[i][j] >= MOD)
567                         ans.v[i][j] %= MOD;
569                 }
571             }
573     v = ans.v;
575     return *this;
577 }
579 Matrix operator/=(T x) {
581     int n = v.size(), m = v[0].size();
583     for (int i = 0; i < n; ++i)
585         for (int j = 0; j < m; ++j) {
587             v[i][j] /= x;
589             if (MOD) {
591                 if (v[i][j] < 0)
593                     v[i][j] = (v[i][j] % MOD + MOD) % MOD;
595                 if (v[i][j] >= MOD)
597                     v[i][j] %= MOD;
599             }
601         }
603     return *this;
605 }
607 Matrix operator/=(Matrix &b) {
609     int n = v.size();
611     int p = b.v.size();
613     int m = b.v[0].size();
615     Matrix ans(n, m, 0);
617     for (int i = 0; i < n; ++i)
619         for (int k = 0; k < p; ++k)
621             for (int j = 0; j < m; ++j) {
623                 ans.v[i][j] += v[i][k] * b.v[k][j];
625                 if (MOD) {
627                     if (ans.v[i][j] < 0)
629                         ans.v[i][j] = (ans.v[i][j] % MOD + MOD) % MOD;
631                     if (ans.v[i][j] >= MOD)
633                         ans.v[i][j] %= MOD;
635                 }
637             }
639     v = ans.v;
641     return *this;
643 }
645 Matrix operator/=(T x) {
647     int n = v.size(), m = v[0].size();
649     for (int i = 0; i < n; ++i)
651         for (int j = 0; j < m; ++j) {
653             v[i][j] /= x;
655             if (MOD) {
657                 if (v[i][j] < 0)
659                     v[i][j] = (v[i][j] % MOD + MOD) % MOD;
661                 if (v[i][j] >= MOD)
663                     v[i][j] %= MOD;
665             }
667         }
669     return *this;
671 }
673 Matrix operator/=(Matrix &b) {
675     int n = v.size();
677     int p = b.v.size();
679     int m = b.v[0].size();
681     Matrix ans(n, m, 0);
683     for (int i = 0; i < n; ++i)
685         for (int k = 0; k < p; ++k)
687             for (int j = 0; j < m; ++j) {
689                 ans.v[i][j] += v[i][k] * b.v[k][j];
691                 if (MOD) {
693                     if (ans.v[i][j] < 0)
695                         ans.v[i][j] = (ans.v[i][j] % MOD + MOD) % MOD;
697                     if (ans.v[i][j] >= MOD)
699                         ans.v[i][j] %= MOD;
701                 }
703             }
705     v = ans.v;
707     return *this;
709 }
711 Matrix operator/=(T x) {
713     int n = v.size(), m = v[0].size();
715     for (int i = 0; i < n; ++i)
717         for (int j = 0; j < m; ++j) {
719             v[i][j] /= x;
721             if (MOD) {
723                 if (v[i][j] < 0)
725                     v[i][j] = (v[i][j] % MOD + MOD) % MOD;
727                 if (v[i][j] >= MOD)
729                     v[i][j] %= MOD;
731             }
733         }
735     return *this;
737 }
739 Matrix operator/=(Matrix &b) {
741     int n = v.size();
743     int p = b.v.size();
745     int m = b.v[0].size();
747     Matrix ans(n, m, 0);
749     for (int i = 0; i < n; ++i)
751         for (int k = 0; k < p; ++k)
753             for (int j = 0; j < m; ++j) {
755                 ans.v[i][j] += v[i][k] * b.v[k][j];
757                 if (MOD) {
759                     if (ans.v[i][j] < 0)
761                         ans.v[i][j] = (ans.v[i][j] % MOD + MOD) % MOD;
763                     if (ans.v[i][j] >= MOD)
765                         ans.v[i][j] %= MOD;
767                 }
769             }
771     v = ans.v;
773     return *this;
775 }
777 Matrix operator/=(T x) {
779     int n = v.size(), m = v[0].size();
781     for (int i = 0; i < n; ++i)
783         for (int j = 0; j < m; ++j) {
785             v[i][j] /= x;
787             if (MOD) {
789                 if (v[i][j] < 0)
791                     v[i][j] = (v[i][j] % MOD + MOD) % MOD;
793                 if (v[i][j] >= MOD)
795                     v[i][j] %= MOD;
797             }
799         }
801     return *this;
803 }
805 Matrix operator/=(Matrix &b) {
807     int n = v.size();
809     int p = b.v.size();
811     int m = b.v[0].size();
813     Matrix ans(n, m, 0);
815     for (int i = 0; i < n; ++i)
817         for (int k = 0; k < p; ++k)
819             for (int j = 0; j < m; ++j) {
821                 ans.v[i][j] += v[i][k] * b.v[k][j];
823                 if (MOD) {
825                     if (ans.v[i][j] < 0)
827                         ans.v[i][j] = (ans.v[i][j] % MOD + MOD) % MOD;
829                     if (ans.v[i][j] >= MOD)
831                         ans.v[i][j] %= MOD;
833                 }
835             }
837     v = ans.v;
839     return *this;
841 }
843 Matrix operator/=(T x) {
845     int n = v.size(), m = v[0].size();
847     for (int i = 0; i < n; ++i)
849         for (int j = 0; j < m; ++j) {
851             v[i][j] /= x;
853             if (MOD) {
855                 if (v[i][j] < 0)
857                     v[i][j] = (v[i][j] % MOD + MOD) % MOD;
859                 if (v[i][j] >= MOD)
861                     v[i][j] %= MOD;
863             }
865         }
867     return *this;
869 }
871 Matrix operator/=(Matrix &b) {
873     int n = v.size();
875     int p = b.v.size();
877     int m = b.v[0].size();
879     Matrix ans(n, m, 0);
881     for (int i = 0; i < n; ++i)
883         for (int k = 0; k < p; ++k)
885             for (int j = 0; j < m; ++j) {
887                 ans.v[i][j] += v[i][k] * b.v[k][j];
889                 if (MOD) {
891                     if (ans.v[i][j] < 0)
893                         ans.v[i][j] = (ans.v[i][j] % MOD + MOD) % MOD;
895                     if (ans.v[i][j] >= MOD)
897                         ans.v[i][j] %= MOD;
899                 }
901             }
903     v = ans.v;
905     return *this;
907 }
909 Matrix operator/=(T x) {
911     int n = v.size(), m = v[0].size();
913     for (int i = 0; i < n; ++i)
915         for (int j = 0; j < m; ++j) {
917             v[i][j] /= x;
919             if (MOD) {
921                 if (v[i][j] < 0)
923                     v[i][j] = (v[i][j] % MOD + MOD) % MOD;
925                 if (v[i][j] >= MOD)
927                     v[i][j] %= MOD;
929             }
931         }
933     return *this;
935 }
937 Matrix operator/=(Matrix &b) {
939     int n = v.size();
941     int p = b.v.size();
943     int m = b.v[0].size();
945     Matrix ans(n, m, 0);
947     for (int i = 0; i < n; ++i)
949         for (int k = 0; k < p; ++k)
951             for (int j = 0; j < m; ++j) {
953                 ans.v[i][j] += v[i][k] * b.v[k][j];
955                 if (MOD) {
957                     if (ans.v[i][j] < 0)
959                         ans.v[i][j] = (ans.v[i][j] % MOD + MOD) % MOD;
961                     if (ans.v[i][j] >= MOD)
963                         ans.v[i][j] %= MOD;
965                 }
967             }
969     v = ans.v;
971     return *this;
973 }
975 Matrix operator/=(T x) {
977     int n = v.size(), m = v[0].size();
979     for (int i = 0; i < n; ++i)
981         for (int j = 0; j < m; ++j) {
983             v[i][j] /= x;
985             if (MOD) {
987                 if (v[i][j] < 0)
989                     v[i][j] = (v[i][j] % MOD + MOD) % MOD;
991                 if (v[i][j] >= MOD)
993                     v[i][j] %= MOD;
995             }
997         }
999     return *this;
1001 }
1003 Matrix operator/=(Matrix &b) {
1005     int n = v.size();
1007     int p = b.v.size();
1009     int m = b.v[0].size();
1011     Matrix ans(n, m, 0);
1013     for (int i = 0; i < n; ++i)
1015         for (int k = 0; k < p; ++k)
1017             for (int j = 0; j < m; ++j) {
1019                 ans.v[i][j] += v[i][k] * b.v[k][j];
1021                 if (MOD) {
1023                     if (ans.v[i][j] < 0)
1025                         ans.v[i][j] = (ans.v[i][j] % MOD + MOD) % MOD;
1027                     if (ans.v[i][j] >= MOD)
1029                         ans.v[i][j] %= MOD;
1031                 }
1033             }
1035     v = ans.v;
1037     return *this;
1039 }
1041 Matrix operator/=(T x) {
1043     int n = v.size(), m = v[0].size();
1045     for (int i = 0; i < n; ++i)
1047         for (int j = 0; j < m; ++j) {
1049             v[i][j] /= x;
1051             if (MOD) {
1053                 if (v[i][j] < 0)
1055                     v[i][j] = (v[i][j] % MOD + MOD) % MOD;
1057                 if (v[i][j] >= MOD)
1059                     v[i][j] %= MOD;
1061             }
1063         }
1065     return *this;
1067 }
1069 Matrix operator/=(Matrix &b) {
1071     int n = v.size();
1073     int p = b.v.size();
1075     int m = b.v[0].size();
1077     Matrix ans(n, m, 0);
1079     for (int i = 0; i < n; ++i)
1081         for (int k = 0; k < p; ++k)
1083             for (int j = 0; j < m; ++j) {
1085                 ans.v[i][j] += v[i][k] * b.v[k][j];
1087                 if (MOD) {
1089                     if (ans.v[i][j] < 0)
1091                         ans.v[i][j] = (ans.v[i][j] % MOD + MOD) % MOD;
1093                     if (ans.v[i][j] >= MOD)
1095                         ans.v[i][j] %= MOD;
1097                 }
1099             }
1101     v = ans.v;
1103     return *this;
1105 }
1107 Matrix operator/=(T x) {
1109     int n = v.size(), m = v[0].size();
1111     for (int i = 0; i < n; ++i)
1113         for (int j = 0; j < m; ++j) {
1115             v[i][j] /= x;
1117             if (MOD) {
1119                 if (v[i][j] < 0)
1121                     v[i][j] = (v[i][j] % MOD + MOD) % MOD;
1123                 if (v[i][j] >= MOD)
1125                     v[i][j] %= MOD;
1127             }
1129         }
1131     return *this;
1133 }
1135 Matrix operator/=(Matrix &b) {
1137     int n = v.size();
1139     int p = b.v.size();
1141     int m = b.v[0].size();
1143     Matrix ans(n, m, 0);
1145     for (int i = 0; i < n; ++i)
1147         for (int k = 0; k < p; ++k)
1149             for (int j = 0; j < m; ++j) {
1151                 ans.v[i][j] += v[i][k] * b.v[k][j];
1153                 if (MOD) {
1155                     if (ans.v[i][j] < 0)
1157                         ans.v[i][j] = (ans.v[i][j] % MOD + MOD) % MOD;
1159                     if (ans.v[i][j] >= MOD)
1161                         ans.v[i][j] %= MOD;
1163                 }
1165             }
1167     v = ans.v;
1169     return *this;
1171 }
1173 Matrix operator/=(T x) {
1175     int n = v.size(), m = v[0].size();
1177     for (int i = 0; i < n; ++i)
1179         for (int j = 0; j < m; ++j) {
1181             v[i][j] /= x;
1183             if (MOD) {
1185                 if (v[i][j] < 0)
1187                     v[i][j] = (v[i][j] % MOD + MOD) % MOD;
1189                 if (v[i][j] >= MOD)
1191                     v[i][j] %= MOD;
1193             }
1195         }
1197     return *this;
1199 }
1201 Matrix operator/=(Matrix &b) {
1203     int n = v.size();
1205     int p = b.v.size();
1207     int m = b.v[0].size();
1209     Matrix ans(n, m, 0);
1211     for (int i = 0; i < n; ++i)
1213         for (int k = 0; k < p; ++k)
1215             for (int j = 0; j < m; ++j) {
1217                 ans.v[i][j] += v[i][k] * b.v[k][j];
1219                 if (MOD) {
1221                     if (ans.v[i][j] < 0)
1223                         ans.v[i][j] = (ans.v[i][j] % MOD + MOD) % MOD;
1225                     if (ans.v[i][j] >= MOD)
1227                         ans.v[i][j] %= MOD;
1229                 }
1231             }
1233     v = ans.v;
1235     return *this;
1237 }
1239 Matrix operator/=(T x) {
1241     int n = v.size(), m = v[0].size();
1243     for (int i = 0; i < n; ++i)
1245         for (int j = 0; j < m; ++j) {
1247             v[i][j] /= x;
1249             if (MOD) {
1251                 if (v[i][j] < 0)
1253                     v[i][j] = (v[i][j] % MOD + MOD) % MOD;
1255                 if (v[i][j] >= MOD)
1257                     v[i][j] %= MOD;
1259             }
1261         }
1263     return *this;
1265 }
1267 Matrix operator/=(Matrix &b) {
1269     int n = v.size();
1271     int p = b.v.size();
1273     int m = b.v[0].size();
1275     Matrix ans(n, m, 0);
1277     for (int i = 0; i < n; ++i)
1279         for (int k = 0; k < p; ++k)
1281             for (int j = 0; j < m; ++j) {
1283                 ans.v[i][j] += v[i][k] * b.v[k][j];
1285                 if (MOD) {
1287                     if (ans.v[i][j] < 0)
1289                         ans.v[i][j] = (ans.v[i][j] % MOD + MOD) % MOD;
1291                     if (ans.v[i][j] >= MOD)
1293                         ans.v[i][j] %= MOD;
1295                 }
1297             }
1299     v = ans.v;
1301     return *this;
1303 }
1305 Matrix operator/=(T x) {
1307     int n = v.size(), m = v[0].size();
1309     for (int i = 0; i < n; ++i)
1311         for (int j = 0; j < m; ++j) {
1313             v[i][j] /= x;
1315             if (MOD) {
1317                 if (v[i][j] < 0)
1319                     v[i][j] = (v[i][j] % MOD + MOD) % MOD;
1321                 if (v[i][j] >= MOD)
1323                     v[i][j] %= MOD;
1325             }
1327         }
1329     return *this;
1331 }
1333 Matrix operator/=(Matrix &b) {
1335     int n = v.size();
1337     int p = b.v.size();
1339     int m = b.v[0].size();
1341     Matrix ans(n, m, 0);
1343     for (int i = 0; i < n; ++i)
1345         for (int k = 0; k < p; ++k)
1347             for (int j = 0; j < m; ++j) {
1349                 ans.v[i][j] += v[i][k] * b.v[k][j];
1351                 if (MOD) {
1353                     if (ans.v[i][j] < 0)
1355                         ans.v[i][j] = (ans.v[i][j] % MOD + MOD) % MOD;
1357                     if (ans.v[i][j] >= MOD)
1359                         ans.v[i][j] %= MOD;
1361                 }
1363             }
1365     v = ans.v;
1367     return *this;
1369 }
1371 Matrix operator/=(T x) {
1373     int n = v.size(), m = v[0].size();
1375     for (int i = 0; i < n; ++i)
1377         for (int j = 0; j < m; ++j) {
1379             v[i][j] /= x;
1381             if (MOD) {
1383                 if (v[i][j] < 0)
1385                     v[i][j] = (v[i][j] % MOD + MOD) % MOD;
1387                 if (v[i][j] >= MOD)
1389                     v[i][j] %= MOD;
1391             }
1393         }
1395     return *this;
1397 }
1399 Matrix operator/=(Matrix &b) {
1401     int n = v.size();
1403     int p = b.v.size();
1405     int m = b.v[0].size();
1407     Matrix ans(n, m, 0);
1409     for (int i = 0; i < n; ++i)
1411         for (int k = 0; k < p; ++k)
1413             for (int j = 0; j < m; ++j) {
1415                 ans.v[i][j] += v[i][k] * b.v[k][j];
1417                 if (MOD) {
1419                     if (ans.v[i][j] < 0)
1421                         ans.v[i][j] = (ans.v[i][j] % MOD + MOD) % MOD;
1423                     if (ans.v[i][j] >= MOD)
1425                         ans.v[i][j] %= MOD;
1427                 }
1429             }
1431     v = ans.v;
1433     return *this;
1435 }
1437 Matrix operator/=(T x) {
1439     int n = v.size(), m = v[0].size();
1441     for (int i = 0; i < n; ++i)
1443         for (int j = 0; j < m; ++j) {
1445             v[i][j] /= x;
1447             if (MOD) {
1449                 if (v[i][j] < 0)
1451                     v[i][j] = (v[i][j] % MOD + MOD) % MOD;
1453                 if (v[i][j] >= MOD)
1455                     v[i][j] %= MOD;
1457             }
1459         }
1461     return *this;
1463 }
1465 Matrix operator/=(Matrix &b) {
1467     int n = v.size();
1469     int p = b.v.size();
1471     int m = b.v[0].size();
1473     Matrix ans(n, m, 0);
1475     for (int i = 0; i < n; ++i)
1477         for (int k = 0; k < p; ++k)
1479             for (int j = 0; j < m; ++j) {
1481                 ans.v[i][j] += v[i][k] * b.v[k][j];
1483                 if (MOD) {
1485                     if (ans.v[i][j] < 0)
1487                         ans.v[i][j] = (ans.v[i][j] % MOD + MOD) % MOD;
1489                     if (ans.v[i][j] >= MOD)
1491                         ans.v[i][j] %= MOD;
1493                 }
1495             }
1497     v = ans.v;
1499     return *this;
1501 }
1503 Matrix operator/=(T x) {
1505     int n = v.size(), m = v[0].size();
1507     for (int i = 0; i < n; ++i)
1509         for (int j = 0; j < m; ++j) {
1511             v[i][j] /= x;
1513             if (MOD) {
1515                 if (v[i][j] < 0)
1517                     v[i][j] = (v[i][j] % MOD + MOD) % MOD;
1519                 if (v[i][j] >= MOD)
1521                     v[i][j] %= MOD;
1523             }
1525         }
1527     return *this;
1529 }
1531 Matrix operator/=(Matrix &b) {
1533     int n = v.size();
1535     int p = b.v.size();
1537     int m = b.v[0].size();
1539     Matrix ans(n, m, 0);
1541     for (int i = 0; i < n; ++i)
1543         for (int k = 0; k < p; ++k)
1545             for (int j = 0; j < m; ++j) {
1547                 ans.v[i][j] += v[i][k] * b.v[k][j];
1549                 if (MOD) {
1551                     if (ans.v[i][j] < 0)
1553                         ans.v[i][j] = (ans.v[i][j] % MOD + MOD) % MOD;
1555                     if (ans.v[i][j] >= MOD)
1557                         ans.v[i][j] %= MOD;
1559                 }
1561             }
1563     v = ans.v;
1565     return *this;
1567 }
1569 Matrix operator/=(T x) {
1571     int n = v.size(), m = v[0].size();
1573     for (int i = 0; i < n; ++i)
1575         for (int j = 0; j < m; ++j) {
1577             v[i][j] /= x;
1579             if (MOD) {
1581                 if (v[i][j] < 0)
1583                     v[i][j] = (v[i][j] % MOD + MOD) % MOD;
1585                 if (v[i][j] >= MOD)
1587                     v[i][j] %= MOD;
1589             }
1591         }
1593     return *this;
1595 }
1597 Matrix operator/=(Matrix &b) {
1599     int n = v.size();
1601     int p = b.v.size();
1603     int m = b.v[0].size();
1605     Matrix ans(n, m, 0);
1607     for (int i = 0; i < n; ++i)
1609         for (int k = 0; k < p; ++k)
1611             for (int j = 0; j < m; ++j) {
1613                 ans.v[i][j] += v[i][k] * b.v[k][j];
1615                 if (MOD) {
1617                     if (ans.v[i][j] < 0)
1619                         ans.v[i][j] = (ans.v[i][j] % MOD + MOD) % MOD;
1621                     if (ans.v[i][j] >= MOD)
1623                         ans.v[i][j] %= MOD;
1625                 }
1627             }
1629     v = ans.v;
1631     return *this;
1633 }
1635 Matrix operator/=(T x) {
1637     int n = v.size(), m = v[0].size();
1639     for (int i = 0; i < n; ++i)
1641         for (int j = 0; j < m; ++j) {
1643             v[i][j] /= x;
1645             if (MOD) {
1647                 if (v[i][j] < 0)
1649                     v[i][j] = (v[i][j] % MOD + MOD) % MOD;
1651                 if (v[i][j] >= MOD)
1653                     v[i][j] %= MOD;
1655             }
1657         }
1659     return *this;
1661 }
1663 Matrix operator/=(Matrix &b) {
1665     int n = v.size();
1667     int p = b.v.size();
1669     int m = b.v[0].size();
1671     Matrix ans(n, m, 0);
1673     for (int i = 0; i < n; ++i)
1675         for (int k = 0; k < p; ++k)
1677             for (int j = 0; j < m; ++j) {
1679                 ans.v[i][j] += v[i][k] * b.v[k][j];
1681                 if (MOD) {
1683                     if (ans.v[i][j] < 0)
1685                         ans.v[i][j] = (ans.v[i][j] % MOD + MOD) % MOD;
1687                     if (ans.v[i][j] >= MOD)
1689                         ans.v[i][j] %= MOD;
1691                 }
1693             }
1695     v = ans.v;
1697     return *this;
1699 }
1701 Matrix operator/=(T x) {
1703     int n = v.size(), m = v[0].size();
1705     for (int i = 0; i < n; ++i)
1707         for (int j = 0; j < m; ++j) {
1709             v[i][j] /= x;
1711             if (MOD) {
1713                 if (v[i][j] < 0)
1715                     v[i][j] = (v[i][j] % MOD + MOD) % MOD;
1717                 if (v[i][j] >= MOD)
1719                     v[i][j] %= MOD;
1721             }
1723         }
1725     return *this;
1727 }
1729 Matrix operator/=(Matrix &b) {
1731     int n = v.size();
1733     int p = b.v.size();
1735     int m = b.v[0].size();
1737     Matrix ans(n, m, 0);
1739     for (int i = 0; i < n; ++i)
1741         for (int k = 0; k < p; ++k)
1743             for (int j = 0; j < m; ++j) {
1745                 ans.v[i][j] += v[i][k] * b.v[k][j];
1747                 if (MOD) {
1749                     if (ans.v[i][j] < 0)
1751                         ans.v[i][j] = (ans.v[i][j] % MOD + MOD) % MOD;
1753                     if (ans.v[i][j] >= MOD)
1755                         ans.v[i][j] %= MOD;
1757                 }
1759             }
1761     v = ans.v;
1763     return *this;
1765 }
1767 Matrix operator/=(T x) {
1769     int n = v.size(), m = v[0].size();
1771     for (int i = 0; i < n; ++i)
1773         for (int j = 0; j < m; ++j) {
1775             v[i][j] /= x;
1777             if (MOD) {
1779                 if (v[i][j] < 0)
1781                     v[i][j] = (v[i][j] % MOD + MOD) % MOD;
1783                 if (v[i][j] >= MOD)
1785                     v[i][j] %= MOD;
1787             }
1789         }
1791     return *this;
1793 }
1795 Matrix operator/=(Matrix &b) {
1797     int n = v.size();
1799     int p = b.v.size();
1801     int m = b.v[0
```

```

25     ret++;
26     j = pmt[j - 1];
27 }
28 }
29 return ret;
30 }

```

3.2. Longest Palindrome

```

1  #define int long long
2  using namespace std;
3
4  string s;
5  string t;
6  int n;
7  int d[2000005];
8  int ans = 0;
9
10 signed main() {
11     cin >> t;
12     n = t.size();
13     for (int i = 0; i < 2 * n + 1; i++) {
14         if (i & 1 ^ 1) {
15             s += '0';
16         } else {
17             s += t[i / 2];
18         }
19     }
20     n = s.size();
21     d[0] = 1;
22     for (int i = 0, l = 0, r = 0; i < n; i++) {
23         if (i > r) {
24             d[i] = 1;
25             bool a = i + d[i] < n;
26             bool b = i - d[i] >= 0;
27             bool c = (s[i + d[i]] == s[i - d[i]]);
28             while (a && b && c) {
29                 d[i]++;
30                 a = i + d[i] < n;
31                 b = i - d[i] >= 0;
32                 c = (s[i + d[i]] == s[i - d[i]]);
33             }
34             l = i - d[i] + 1;
35             r = i + d[i] - 1;
36         } else {
37             int j = l + r - i;
38             if (j - d[j] + 1 > l) {
39                 d[i] = d[j];
40             } else {
41                 d[i] = r - i + 1;
42                 a = i + d[i] < n;
43                 b = i - d[i] >= 0;
44                 c = (s[i + d[i]] == s[i - d[i]]);
45                 while (a && b && c) {
46                     d[i]++;
47                     a = i + d[i] < n;
48                     b = i - d[i] >= 0;
49                     c = (s[i + d[i]] == s[i - d[i]]);
50                 }
51                 l = i - d[i] + 1;
52                 r = i + d[i] - 1;
53             }
54         }
55         // cout << d[i] << " ";
56         if (d[i] > d[ans]) {
57             ans = i;
58         }
59     }
60     for (int i = ans - d[ans] + 1; i < ans + d[ans]; i++) {
61         if (s[i] ^ '0') {
62             cout << s[i];
63         }
64     }
65 }

```

3.3. Z

```

1  #define int long long
2  using namespace std;
3
4  string s, t;
5  int ans = 0;
6
7  int z[2000005];
8
9  signed main() {
10     ios::sync_with_stdio(0);
11     cin.tie(0);
12     cout.tie(0);
13     cin >> s >> t;
14     s = t + '0' + s;

```

```

17     int n, m;
18     n = s.size();
19     m = t.size();
20     for (int i = 0, l = 0, r = 0; i < n; i++) {
21         if (z[i - l] < r - i + 1) {
22             z[i] = z[i - l];
23         } else {
24             z[i] = max(r - i + 1, (int)0);
25             while (i + z[i] < n && s[i + z[i]] == s[z[i]]) {
26                 z[i]++;
27             }
28             l = i;
29             r = i + z[i] - 1;
30             if (z[i] == m) {
31                 ans++;
32             }
33         }
34     }
35     cout << ans;
36 }

```

4. Graph

4.1. one-out-degree (CSES Planets Cycles)

```

1  #define int long long
2  using namespace std;
3
4  int n, q;
5  int a[200005];
6  int r[200005];
7  int d[200005];
8  int cycle[200005];
9  int len[200005];
10 int cnt = 0;
11 vector<int> v[200005];
12 bitset<200005> vis1;
13 bitset<200005> vis2;
14
15 void findcycle(int x) {
16     while (!vis1[x]) {
17         vis1[x] = 1;
18         x = a[x];
19     }
20     cnt++;
21     cycle[x] = cnt;
22     r[x] = 0;
23     len[cnt] = 1;
24     int temp = a[x];
25     while (temp ^ x) {
26         r[temp] = len[cnt];
27         len[cnt]++;
28         cycle[temp] = cnt;
29         temp = a[temp];
30     }
31 }
32
33 void dfs(int x) {
34     if (vis2[x])
35         return;
36     vis2[x] = 1;
37     for (int i : v[x]) {
38         dfs(i);
39     }
40 }
41
42 void dfs2(int x) {
43     if (cycle[x] || d[x])
44         return;
45     dfs2(a[x]);
46     d[x] = d[a[x]] + 1;
47     r[x] = r[a[x]];
48     cycle[x] = cycle[a[x]];
49 }
50
51 signed main() {
52     ios::sync_with_stdio(0);
53     cin.tie(0);
54     cout.tie(0);
55     cin >> n;
56     for (int i = 1; i <= n; i++) {
57         cin >> a[i];
58         v[i].push_back(a[i]);
59         v[a[i]].push_back(i);
60     }
61     for (int i = 1; i <= n; i++) {
62         if (!vis2[i]) {
63             findcycle(i);
64             dfs(i);
65         }
66     }
67 }

```



```

69     for (int i = 1; i <= n; i++) {
70         if (!cycle[i] && !r[i]) {
71             dfs2(i);
72         }
73     }
74     for (int i = 1; i <= n; i++) {
75         cout << d[i] + len[cycle[i]] << " ";
76     }
77 }

```

4.2. Dijkstra

```

1 vector<pair<int, int>> v[100005], v2[100005];
2 vector<edge> es;
3 int dis1[100005];
4 int dis2[100005];
5 bitset<100005> vis1, vis2;
6
7 void dijkstra(int x, int *dis, vector<pair<int, int>> *v, bitset<100005> &vis) {
8     priority_queue<pair<int, int>, vector<pair<int, int>>,
9         greater<pair<int, int>>>
10         pq;
11     memset(dis, 0x3f, sizeof(dis));
12     vis.reset();
13     dis[x] = 0;
14     pq.push({0, x});
15     while (!pq.empty()) {
16         pair<int, int> now = pq.top();
17         pq.pop();
18         if (vis[now.second])
19             continue;
20         vis[now.second] = 1;
21         for (auto [i, w] : v[now.second]) {
22             if (vis[i])
23                 continue;
24             if (dis[now.second] + w < dis[i]) {
25                 dis[i] = dis[now.second] + w;
26                 pq.push({dis[i], i});
27             }
28         }
29     }
30 }

```

4.3. MaximumFlow

```

1 #define int long long
2 using namespace std;
3
4 int n, m;
5 vector<int> v[1005];
6 int head[1005];
7 int c[1005][1005];
8 int lv[1005];
9 int ans = 0;
10
11 bool bfs() {
12     memset(head, 0, sizeof(head));
13     memset(lv, 0, sizeof(lv));
14     queue<int> q;
15     q.push(1);
16     while (!q.empty()) {
17         int now = q.front();
18         q.pop();
19         if (now == n)
20             continue;
21         for (int i : v[now]) {
22             if (i != 1 && c[now][i] && !lv[i]) {
23                 lv[i] = lv[now] + 1;
24                 q.push(i);
25             }
26         }
27     }
28     return lv[n];
29 }
30
31 int dfs(int x, int flow) {
32     int ret = 0;
33     if (x == n)
34         return flow;
35     for (int i = head[x]; i < v[x].size(); i++) {
36         int y = v[x][i];
37         head[x] = y;
38         if (c[x][y] && lv[y] == lv[x] + 1) {
39             int d = dfs(y, min(flow, c[x][y]));
40             flow -= d;
41             c[x][y] -= d;
42             c[y][x] += d;
43             ret += d;
44         }
45     }
46     return ret;
47 }

```

```

49 signed main() {
50     cin >> n >> m;
51     while (m--) {
52         int x, y, z;
53         cin >> x >> y >> z;
54         if (c[x][y] || c[y][x]) {
55             c[x][y] += z;
56             continue;
57         }
58         v[x].push_back(y);
59         v[y].push_back(x);
60         c[x][y] = z;
61     }
62     while (bfs()) {
63         ans += dfs(1, INT_MAX);
64     }
65     cout << ans;
66 }

```

4.4. SCC

```

1 int n, m;
2 vector<int> v[100005];
3 int d[100005];
4 int low[100005];
5 int cnt = 0;
6 stack<int> s;
7 int scc[100005];
8 int now = 0;
9
10 void dfs(int x) {
11     d[x] = low[x] = ++cnt;
12     s.push(x);
13     for (int i : v[x]) {
14         if (scc[i])
15             continue;
16         if (d[i]) {
17             low[x] = min(low[x], d[i]);
18         } else {
19             dfs(i);
20             low[x] = min(low[x], low[i]);
21         }
22     }
23     if (d[x] == low[x]) {
24         now++;
25         while (!s.empty()) {
26             int k = s.top();
27             s.pop();
28             scc[k] = now;
29             if (k == x)
30                 break;
31         }
32     }
33 }

```

4.5. 2-SAT(CSES Giant Pizza)

```

1 #define int long long
2 using namespace std;
3
4 int n, m;
5 vector<int> v[200005];
6 int d[200005];
7 int low[200005];
8 int cnt = 0;
9 int now = 0;
10 int scc[200005];
11 stack<int> s;
12 int op[200005];
13 vector<int> v2[200005];
14 int ind[200005];
15 queue<int> q;
16 int ans[200005];
17
18 int no(int x) {
19     if (x > m)
20         return x - m;
21     return x + m;
22 }
23
24 void dfs(int x) {
25     d[x] = low[x] = ++cnt;
26     s.push(x);
27     for (int i : v[x]) {
28         if (scc[i])
29             continue;
30         if (d[i]) {
31             low[x] = min(low[x], d[i]);
32         } else {
33             dfs(i);
34             low[x] = min(low[x], low[i]);
35         }
36     }
37 }

```

```

    }
    if (d[x] == low[x]) {
        now++;
        while (!s.empty()) {
            int k = s.top();
            s.pop();
            scc[k] = now;
            if (k == x)
                break;
        }
    }
}

signed main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    cin >> n >> m;
    while (n--) {
        char a, b;
        int x, y;
        cin >> a >> x >> b >> y;
        if (a == '-')
            x = no(x);
        if (b == '-')
            y = no(y);
        v[no(x)].push_back(y);
        v[no(y)].push_back(x);
    }
    for (int i = 1; i <= 2 * m; i++) {
        if (!d[i]) {
            dfs(i);
        }
    }
    for (int i = 1; i <= m; i++) {
        if (scc[i] ^ scc[i + m]) {
            op[scc[i]] = scc[i + m];
            op[scc[i + m]] = scc[i];
        } else {
            cout << "IMPOSSIBLE";
            exit(0);
        }
    }
    for (int i = 1; i <= 2 * m; i++) {
        for (int j : v[i]) {
            if (scc[i] ^ scc[j]) {
                v2[scc[j]].push_back(scc[i]);
                ind[scc[i]]++;
            }
        }
    }
    for (int i = 1; i <= now; i++) {
        if (!ind[i]) {
            q.push(i);
        }
    }
    while (!q.empty()) {
        int k = q.front();
        q.pop();
        if (!ans[k]) {
            ans[k] = 1;
            ans[op[k]] = 2;
        }
        for (int i : v2[k]) {
            ind[i]--;
            if (!ind[i]) {
                q.push(i);
            }
        }
    }
    for (int i = 1; i <= m; i++) {
        if (ans[scc[i]] == 1) {
            cout << "+ ";
        } else {
            cout << "- ";
        }
    }
}

```

5. DP

5.1. Li-Chao Segment Tree

```

1 struct line {
2     int a, b = 1000000000000000000;
3     int y(int x) { return a * x + b; }
4 };
5
6 line tree[4000005];
7 int n, x;
8 int s[200005];

```

```

9 int f[200005];
10 int dp[200005];
11
12 void update(line ins, int l = 1, int r = 1e6, int index = 1) {
13     if (l == r) {
14         if (ins.y(l) < tree[index].y(l)) {
15             tree[index] = ins;
16         }
17         return;
18     }
19     int mid = (l + r) >> 1;
20     if (tree[index].a < ins.a)
21         swap(tree[index], ins);
22     if (tree[index].y(mid) > ins.y(mid)) {
23         swap(tree[index], ins);
24         update(ins, l, mid, index << 1);
25     } else {
26         update(ins, mid + 1, r, index << 1 | 1);
27     }
28 }
29
30 int query(int x, int l = 1, int r = 1000000, int index = 1) {
31     int cur = tree[index].y(x);
32     if (l == r) {
33         return cur;
34     }
35     int mid = (l + r) >> 1;
36     if (x <= mid) {
37         return min(cur, query(x, l, mid, index << 1));
38     } else {
39         return min(cur, query(x, mid + 1, r, index << 1 | 1));
40     }
41 }

```

5.2. CHO

```

1 struct line {
2     int a, b;
3     int y(int x) { return a * x + b; }
4 };
5
6 struct CHO {
7     deque<line> dq;
8     int intersect(line x, line y) {
9         int d1 = x.b - y.b;
10        int d2 = y.a - x.a;
11        return d1 / d2;
12    }
13    bool check(line x, line y, line z) {
14        int I12 = intersect(x, y);
15        int I23 = intersect(y, z);
16        return I12 < I23;
17    }
18    void insert(int a, int b) {
19        if (!dq.empty() && a == dq.back().a)
20            return;
21        while (dq.size() >= 2 &&
22            !check(dq[dq.size() - 2], dq[dq.size() - 1], {a, b})) {
23            dq.pop_back();
24        }
25        dq.push_back({a, b});
26    }
27    void update(int x) {
28        while (dq.size() >= 2 && dq[0].y(x) >= dq[1].y(x)) {
29            dq.pop_front();
30        }
31    }
32    int query(int x) {
33        update(x);
34        return dq.front().y(x);
35    }
36 };

```

6. Geometry

6.1. Intersect

```

1 struct point {
2     int x, y;
3     point operator+(point b) { return {x + b.x, y + b.y}; }
4     point operator-(point b) { return {x - b.x, y - b.y}; }
5     int operator*(point b) { return x * b.x + y * b.y; }
6     int operator^(point b) { return x * b.y - y * b.x; }
7 };
8
9 bool onseg(point x, point y, point z) {
10     return ((x - z) ^ (y - z)) == 0 && (x - z) * (y - z) <= 0;
11 }
12
13 int dir(point x, point y) {
14     int k = x ^ y;

```

```

15 if (k == 0)
16     return 0;
17 if (k > 0)
18     return 1;
19 return -1;
20 }
21
22 bool intersect(point x, point y, point z, point w) {
23     if (onseg(x, y, z) || onseg(x, y, w))
24         return 1;
25     if (onseg(z, w, x) || onseg(z, w, y))
26         return 1;
27     if (dir(y - x, z - x) * dir(y - x, w - x) == -1 &&
28         dir(z - w, x - w) * dir(z - w, y - w) == -1) {
29         return 1;
30     }
31     return 0;
32 }

```

6.2. Inside

```

1 int inside(point p) {
2     int ans = 0;
3     for (int i = 1; i <= n; i++) {
4         if (onseg(a[i], a[i + 1], {p.x, p.y})) {
5             return -1;
6         }
7         if (intersect({p.x, p.y}, {INF, p.y}, a[i], a[i + 1])) {
8             ans ^= 1;
9         }
10        point temp = a[i].y > a[i + 1].y ? a[i] : a[i + 1];
11        if (temp.y == p.y && temp.x > p.x) {
12            ans ^= 1;
13        }
14    }
15    return ans;
16 }

```

6.3. Minimum Euclidean Distance

```

1 #define int long long
2 #define pii pair<int, int>
3 using namespace std;
4
5 int n;
6 vector<pair<int, int>> v;
7 set<pair<int, int>> s;
8 int dd = LONG_LONG_MAX;
9
10 int dis(pii x, pii y) {
11     return (x.first - y.first) * (x.first - y.first) +
12            (x.second - y.second) * (x.second - y.second);
13 }
14
15 signed main() {
16     ios::sync_with_stdio(0);
17     cin.tie(0);
18     cout.tie(0);
19     cin >> n;
20     for (int i = 0; i < n; i++) {
21         int x, y;
22         cin >> x >> y;
23         x += 1000000000;
24         v.push_back({x, y});
25     }
26     sort(v.begin(), v.end());
27     int l = 0;
28     for (int i = 0; i < n; i++) {
29         int d = ceil(sqrt(dd));
30         while (l < i && v[l].first - v[i].first > d) {
31             s.erase({v[l].second, v[l].first});
32             l++;
33         }
34         auto x = s.lower_bound({v[i].second - d, 0});
35         auto y = s.upper_bound({v[i].second + d, 0});
36         for (auto it = x; it != y; it++) {
37             dd = min(dd, dis({it->second, it->first}, v[i]));
38         }
39         s.insert({v[i].second, v[i].first});
40     }
41     cout << dd;
42 }

```

6.4. Convex Hull

```

1 #define int long long
2 #define fastio
3 ios_base::sync_with_stdio(0);
4 cin.tie(0);
5 cout.tie(0);

```

```

7 using namespace std;
8
9 template <typename T> pair<T, T> operator-(pair<T, T> a, pair<T, T> b) {
10     return make_pair(a.first - b.first, a.second - b.second);
11 }
12
13 template <typename T> T cross(pair<T, T> a, pair<T, T> b) {
14     return a.first * b.second - a.second * b.first;
15 }
16
17 template <typename T> vector<pair<T, T>> getCH(vector<pair<T, T>> v) {
18     int n = v.size();
19     sort(v.begin(), v.end());
20     vector<pair<T, T>> hull;
21     for (int i = 0; i < 2; i++) {
22         int t = hull.size();
23         for (auto x : v) {
24             while (hull.size() - t >= 2 &&
25                 cross(hull[hull.size() - 1] - hull[hull.size() - 2],
26                     x - hull[hull.size() - 2]) <= 0)
27                 hull.pop_back();
28             hull.push_back(x);
29         }
30         hull.pop_back();
31         reverse(v.begin(), v.end());
32     }
33     return hull;
34 }

```

7. Tree

7.1. Heavy Light Decomposition (modify and query on path)

```

1 #define int long long
2 using namespace std;
3
4 int tree[800005];
5
6 int n, q;
7 int a[200005];
8 int st[200005];
9 int tp[200005];
10 int p[200005];
11 int cnt = 0;
12 int d[200005];
13 int si[200005];
14 vector<int> v[200005];
15 int b[200005];
16
17 void build(int l = 1, int r = n, int index = 1) {
18     if (l == r) {
19         tree[index] = b[l];
20         return;
21     }
22     int mid = (l + r) >> 1;
23     build(l, mid, index << 1);
24     build(mid + 1, r, index << 1 | 1);
25     tree[index] = max(tree[index << 1], tree[index << 1 | 1]);
26 }
27
28 int query(int L, int R, int l = 1, int r = n, int index = 1) {
29     if (L == l && R == r) {
30         return tree[index];
31     }
32     int mid = (l + r) >> 1;
33     if (R <= mid) {
34         return query(L, R, l, mid, index << 1);
35     }
36     if (L > mid) {
37         return query(L, R, mid + 1, r, index << 1 | 1);
38     }
39     return max(query(L, mid, l, mid, index << 1),
40               query(mid + 1, R, mid + 1, r, index << 1 | 1));
41 }
42
43 void modify(int x, int val, int l = 1, int r = n, int index = 1) {
44     if (l == r) {
45         tree[index] = val;
46         return;
47     }
48     int mid = (l + r) >> 1;
49     if (x <= mid) {
50         modify(x, val, l, mid, index << 1);
51     } else {
52         modify(x, val, mid + 1, r, index << 1 | 1);
53     }
54     tree[index] = max(tree[index << 1], tree[index << 1 | 1]);
55 }

```

```

57 void dfs(int x, int pre) {
59     si[x] = 1;
61     for (int i : v[x]) {
63         if (i == pre)
64             continue;
65         p[i] = x;
66         d[i] = d[x] + 1;
67         dfs(i, x);
68         si[x] += si[i];
69     }
70 }
71 void dfs2(int x, int pre, int t) {
72     tp[x] = t;
73     st[x] = ++cnt;
74     int ma = 0;
75     for (int i : v[x]) {
76         if (i == pre)
77             continue;
78         if (si[i] > si[ma]) {
79             ma = i;
80         }
81     }
82     if (!ma)
83         return;
84     dfs2(ma, x, t);
85     for (int i : v[x]) {
86         if (i == pre || i == ma)
87             continue;
88         dfs2(i, x, i);
89     }
90 }
91 int f(int x, int y) {
92     int ret = 0;
93     while (tp[x] ^ tp[y]) {
94         if (d[tp[x]] < d[tp[y]]) {
95             swap(x, y);
96         }
97         ret = max(ret, query(st[tp[x]], st[x]));
98         x = p[tp[x]];
99     }
100     if (d[x] > d[y])
101         swap(x, y);
102     ret = max(ret, query(st[x], st[y]));
103     return ret;
104 }
105 }
106 signed main() {
107     ios::sync_with_stdio(0);
108     cin.tie(0);
109     cout.tie(0);
110     cin >> n >> q;
111     for (int i = 1; i <= n; i++) {
112         cin >> a[i];
113     }
114     for (int i = 1; i < n; i++) {
115         int x, y;
116         cin >> x >> y;
117         v[x].push_back(y);
118         v[y].push_back(x);
119     }
120     dfs(1, 0);
121     dfs2(1, 0, 1);
122     for (int i = 1; i <= n; i++) {
123         b[st[i]] = a[i];
124     }
125     build();
126     while (q--) {
127         int mode, x, y;
128         cin >> mode >> x >> y;
129         if (mode == 1) {
130             modify(st[x], y);
131         } else {
132             cout << f(x, y) << " ";
133         }
134     }
135 }

```

7.2. LCA

```

1 #define int long long
2 using namespace std;
3
4 int n, q;
5 int a[200005][21];
6 int d[200005];
7 vector<int> v[200005];
8
9 void init() {

```

```

11     for (int j = 1; j < 21; j++) {
12         for (int i = 1; i <= n; i++) {
13             a[i][j] = a[a[i][j - 1]][j - 1];
14         }
15     }
16 }
17 void dfs(int x, int pre) {
18     for (int i : v[x]) {
19         if (i == pre)
20             continue;
21         a[i][0] = x;
22         d[i] = d[x] + 1;
23         dfs(i, x);
24     }
25 }
26
27 int lca(int x, int y) {
28     while (d[x] ^ d[y]) {
29         if (d[x] < d[y]) {
30             swap(x, y);
31         }
32         int k = __lg(d[x] - d[y]);
33         x = a[x][k];
34     }
35     if (x == y) {
36         return x;
37     }
38     for (int i = 20; i >= 0; i--) {
39         if (a[x][i] != a[y][i]) {
40             x = a[x][i];
41             y = a[y][i];
42         }
43     }
44     return a[x][0];
45 }
46
47 signed main() {
48     ios::sync_with_stdio(0);
49     cin.tie(0);
50     cout.tie(0);
51     cin >> n >> q;
52     for (int i = 1; i < n; i++) {
53         int x, y;
54         cin >> x >> y;
55         v[x].push_back(y);
56         v[y].push_back(x);
57     }
58     dfs(1, 0);
59     init();
60     while (q--) {
61         int x, y;
62         cin >> x >> y;
63         int k = lca(x, y);
64         cout << (d[x] + d[y] - 2 * d[k]) << "\n";
65     }
66 }

```

8. Misc

8.1. Tri Search

```

1 using namespace std;
2 int n;
3 double a[15], x, y;
4
5 double get(double x) {
6     double ret = 0;
7     double k = 1;
8     for (int i = 0; i <= n; i++) {
9         ret += k * a[i];
10        k *= x;
11    }
12    return -ret;
13 }
14
15 template <class T> T bi_search(T l, T r, T end) {
16     if (!check(r - end))
17         return r - end;
18     for (; r - l > end; ) {
19         T mid = (l + r) / 2;
20         if (check(mid))
21             r = mid;
22         else
23             l = mid;
24     }
25     return l;
26 }
27
28 /*check gives 000000001111 find the last 0*/
29

```

```
31 template <class T> T tri_search(T l, T r, T end) {
    T midl, midr;
    for (;;) {
33         midl = (l + r) / 2;
35         midr = (midl + r) / 2;
37         if (midr - midl < end)
            break;
39         if (get(midr) > get(midl))
            r = midr;
41         else
            l = midl;
    }
    for (; r - l > end;) {
43         midl = (l + r) / 2;
45         if (get(r) > get(l))
            r = midl;
47         else
            l = midl;
    }
49     return l;
    }
51 /*get gives the value, find the minimum*/

53 int main() {
    cin >> n >> x >> y;
55     for (int i = n; i >= 0; i--) {
        cin >> a[i];
57     }
    cout << fixed << setprecision(7) << tri_search<double>(x, y, 1e-7);
59 }
```