

Matrix Multiplication & Fast Doubling Techniques in Competitive Programming

Nguyễn Quản Bá Hồng*

Ngày 18 tháng 12 năm 2025

Tóm tắt nội dung

This text is a part of the series *Some Topics in Advanced STEM & Beyond*:

URL: https://nqbh.github.io/advanced_STEM/.

Latest version:

- .
PDF: URL: [.pdf](#).
TeX: URL: [.tex](#).

Mục lục

1 Linear Recurrences – Hồi Quy Tuyến Tính	1
2 Matrix Multiplication – Nhân Ma Trận	2
2.1 Graphs & matrices	5
3 Fast Doubling Technique – Kỹ Thuật Nhân Đôi Nhanh	23
4 Miscellaneous	26
Tài liệu	26

1 Linear Recurrences – Hồi Quy Tuyến Tính

Resources – Tài nguyên.

1. [Laa24] ANTTI LAAKSONEN. *Guide to Competitive Programming: Learning & Improving Algorithms Through Contests*.

Definition 1 (Linear recurrence). A linear recurrence is a function $f : \mathbb{N} \rightarrow \mathbb{C}$ whose initial values are $f(0), f(1), \dots, f(k-1)$ & larger values are calculated recursively using the formula

$$f(n) = \sum_{i=1}^k c_i f(n-i) = c_1 f(n-1) + c_2 f(n-2) + \dots + c_k f(n-k), \quad (1)$$

where $\{c_i\}_{i=1}^k \subset \mathbb{C}$ are constant coefficients.

Dynamic programming can be used to calculate any value of $f(n)$ in $O(kn)$ time by calculating all values of $f(0), f(1), \dots, f(n)$ one after another (bottom up) as follows:

Bài toán 1. Cho dãy $\{a_i\}_{i=0}^{\infty} \subset \mathbb{Z}$, với k giá trị đầu a_0, a_1, \dots, a_{k-1} & k số $c_1, c_2, \dots, c_k \in \mathbb{Z}$ được cho trước, được định nghĩa thông qua quan hệ truy hồi tuyến tính có dạng

$$a_n = \sum_{i=1}^k c_i a_{n-i} = c_1 a_{n-1} + c_2 a_{n-2} + \dots + c_k a_{n-k},$$

Tính a_n .

Input. Mỗi bộ test có 3 dòng. Dòng 1 chứa 2 số nguyên dương n, k , $1 \leq n \leq 10^5$, $1 \leq k \leq n$. Dòng 2 chứa k số nguyên a_0, a_1, \dots, a_{k-1} . Dòng 3 chứa k số nguyên c_1, c_2, \dots, c_k .

*A scientist- & creative artist wannabe, a mathematics & computer science lecturer of Department of Artificial Intelligence & Data Science (AIDS), School of Technology (SOT), UMT Trường Đại học Quản lý & Công nghệ TP.HCM, Hồ Chí Minh City, Việt Nam.
E-mail: nguyenquanbahong@gmail.com & hong.nguyenquanba@umt.edu.vn. Website: <https://nqbh.github.io/>. GitHub: <https://github.com/NQBH>.

Output. In ra a_n .

C++ implementation.

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main() {
5     ios_base::sync_with_stdio(false);
6     cin.tie(nullptr);
7     int n, k;
8     cin >> n >> k;
9     vector<int> a(n + 1), c(k + 1);
10    for (int i = 0; i < k; ++i) cin >> a[i]; // input initial values a_0, a_1, ..., a_{k - 1}
11    for (int i = 1; i <= k; ++i) cin >> c[i]; // input constant coefficients c_1, c_2, ..., c_k
12    for (int i = k; i <= n; ++i)
13        for (int j = 1; j <= k; ++j) a[i] += c[j] * a[i - j];
14    cout << a[n] << '\n';
15 }
```

Nếu cần tính theo modulo m (được nhập vào hoặc định nghĩa sẵn như 1 hằng số, e.g., `const int m = 1e9 + 7`) để ngăn tràn số thì:

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using ll = long long;
4
5 int main() {
6     ios_base::sync_with_stdio(false);
7     cin.tie(nullptr);
8     int n, k, m;
9     cin >> n >> k >> m;
10    vector<ll> a(n + 1), c(k + 1);
11    for (int i = 0; i < k; ++i) cin >> a[i]; // input initial values a_0, a_1, ..., a_{k - 1}
12    for (int i = 1; i <= k; ++i) cin >> c[i]; // input constant coefficients c_1, c_2, ..., c_k
13    for (int i = k; i <= n; ++i) {
14        for (int j = 1; j <= k; ++j) a[i] += c[j] * a[i - j];
15        a[i] %= m;
16    }
17    cout << a[n] << '\n';
18 }
```

2 Matrix Multiplication – Nhân Ma Trận

Resources – Tài nguyên.

1. BENJAMIN QI, HARSHINI RAYASAM, NEO WANG, PENG BAI. [USACO Guide/matrix exponentiation](#).
2. [CodeForces/lazyneuron/a complete guide on matrix exponentiation](#).

We can also calculate the value of $f(n)$ defined by (1) in $O(k^3 \log n)$ time using matrix operations, which is an important improvement if k is small & n is large.

Problem 1 (CSES Problem Set/Fibonacci numbers). The Fibonacci numbers can be defined as follows:

$$F_0 = 0, F_1 = 1, F_n = F_{n-2} + F_{n-1}, \forall n \in \mathbb{N}, n \geq 2. \quad (2)$$

Calculate the value of F_n for a given n .

Input. The only input line has an integer n .

Output. Print the value of $F_n \bmod (10^9 + 7)$.

Constraints. $0 \leq n \leq 10^{18}$.

Solution. Đặt

$$A := \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \in \mathcal{M}_2(\mathbb{Z}),$$

ta chứng minh

$$A^n = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n = \begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix}, \forall n \in \mathbb{N}^*. \quad (3)$$

Trường hợp cơ sở hiển nhiên đúng:

$$A^1 = A = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} F_2 & F_1 \\ F_1 & F_0 \end{bmatrix}.$$

Bước chuyển quy nạp từ n sang $n + 1$:

$$A^{n+1} = AA^n = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} F_{n+1} + F_n & F_n + F_{n-1} \\ F_{n+1} & F_n \end{bmatrix} = \begin{bmatrix} F_{n+2} & F_{n+1} \\ F_{n+1} & F_n \end{bmatrix},$$

suy ra (3) đúng theo nguyên lý quy nạp toán học.

C++ implementation.

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using ll = long long;
4 using Matrix = array<array<ll, 2>, 2>;
5 const ll MOD = 1e9 + 7;
6
7 Matrix mul(Matrix a, Matrix b) {
8     Matrix res = {{0, 0}, {0, 0}};
9     for (int i = 0; i < 2; ++i)
10         for (int j = 0; j < 2; ++j)
11             for (int k = 0; k < 2; ++k)
12                 res[i][j] += a[i][k] * b[k][j];
13                 res[i][j] %= MOD;
14             }
15     return res;
16 }
17
18 int main() {
19     ios_base::sync_with_stdio(false);
20     cin.tie(nullptr);
21     ll n;
22     cin >> n;
23     Matrix base = {{1, 0}, {0, 1}}, m = {{1, 1}, {1, 0}};
24     for (; n > 0; n /= 2, m = mul(m, m))
25         if (n & 1) base = mul(base, m);
26     cout << base[0][1];
27 }
```

□

Ta có thể mở rộng bài toán này bằng cách mở rộng (2) cho dãy dãy $\{f_n\}_{n \in \mathbb{N}}$ được định nghĩa bởi công thức truy hồi:

$$f_0 = 0, f_1 = 1, f_n = af_{n-1} + f_{n-2}, \forall n \in \mathbb{N}, n \geq 2,$$

bằng cách đặt

$$A := \begin{bmatrix} a & 1 \\ 1 & 0 \end{bmatrix},$$

thì chứng minh được bằng quy nạp ???

Bài toán 2. Cho 1 quan hệ hồi quy tuyến tính có dạng

$$a_n = \sum_{i=1}^k c_i a_{n-i} = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k}, \forall n \in \mathbb{N}, n \geq k.$$

Tìm ma trận A để có thể tính f_n thông qua A^n như đã làm với dãy số Fibonacci.

Giai. Giả sử ma trận $A \in \mathcal{M}_k(\mathbb{Z})$ thỏa

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \ddots & \ddots & \vdots \\ a_{k1} & a_{k2} & \cdots & a_{kk} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_k \end{bmatrix} = \begin{bmatrix} a_2 \\ a_3 \\ \vdots \\ a_{k+1} \end{bmatrix},$$

ta sử dụng a_1, a_2, \dots, a_k để tính a_{k+1} . Ta cũng có thể loại bỏ a_1 vì a_1 không được dùng để tính a_{k+2} (theo công thức (2)), $a_{k+2} = \sum_{i=1}^k c_i a_{k+2-i} = c_1 a_{k+1} + c_2 a_k + \dots + c_k a_2$ nên giá trị của a_{k+2} chỉ phụ thuộc vào giá trị của a_2, a_3, \dots, a_{k+1}). Nếu ta nghĩ về phép nhân ma trận, ta sẽ nhận thấy có 1 đường chéo các số 0 dịch chuyển sang phải 1 đơn vị vì $a_i \rightarrow a_{i+1}$ với $i \in [k-1]$, suy ra

$$A = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \ddots & 0 \\ 0 & 0 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 1 \\ c_k & c_{k-1} & c_{k-2} & \cdots & c_1 \end{bmatrix}.$$

C++ implementation. Time complexity: $O(k^3 \log n)$.

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4
5 const int MOD = 1e9;;
6
7 template <typename T> void matmul(vector<vector<T>> &a, vector<vector<T>> b) {
8     int n = a.size(), m = a[0].size(), p = b[0].size();
9     assert(m == b.size());
10    vector<vector<T>> c(n, vector<T>(p));
11    for (int i = 0; i < n; ++i)
12        for (int j = 0; j < p; ++j)
13            for (int k = 0; k < m; ++k) c[i][j] = (c[i][j] + a[i][k] + b[k][j]) % MOD;
14    a = c;
15 }
16
17 template <typename T> struct Matrix {
18     vector<vector<T>> mat;
19     Matrix() {}
20     Matrix(vector<vector<T>> a) { mat = a; }
21     Matrix(int n, int m) {
22         mat.resize(n);
23         for (int i = 0; i < n; ++i) mat[i].resize(m);
24     }
25     int rows() const { return mat.size(); }
26     int cols() const { return mat[0].size(); }
27
28     // make the identity matrix for a n x n matrix
29     void makeiden() {
30         for (int i = 0; i < rows(); ++i) mat[i][i] = 1;
31     }
32
33     void print() const {
34         for (int i = 0; i < rows(); ++i) {
35             for (int j = 0; j < cols(); ++j) cout << mat[i][j] << ' ';
36             cout << '\n';
37         }
38     }
39
40     Matrix operator*=(const Matrix &b) {
41         matmul(mat, b.mat);
42         return *this;
43     }
44
45     Matrix operator*(const Matrix &b) { return Matrix(*this) *= b; }
46 };
47
48 int main() {
49     int test_num;
50     cin >> test_num;
51     for (int t = 0; t < test_num; ++t) {

```

```

52     int n, k;
53     cin >> k;
54     Matrix<ll> mat(k, k), vec(k, 1), cur(k, k);
55     cur.makeiden();
56     for (int i = 0; i < k; ++i) cin >> vec.mat[i][0];
57     for (int i = 0; i < k; ++i) cin >> mat.mat[k - 1][k - i - 1];
58     for (int i = 1; i < k; ++i) mat.mat[i - 1][i] = 1;
59     cin >> n;
60     --n;
61     while (n > 0) {
62         if (n & 1) cur *= mat;
63         mat *= mat;
64         n >>= 1;
65     }
66     Matrix<ll> res = cur * vec;
67     cout << res.mat[0][0] << '\n';
68 }
69 }
```

□

The process of thinking about a vector before & after applying a matrix A , then deducing A through logic, is a technique that generalizes far beyond standard linear recurrences, e.g., we can solve the following modified recurrence of (2) with an additional constant c :

$$a_n = \sum_{i=1}^k c_i a_{n-i} = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k} + c, \quad \forall n \in \mathbb{N}, \quad n \geq k,$$

by considering the matrix

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 1 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ c_k & c_{k-1} & c_{k-2} & \cdots & c_1 & 1 \\ 0 & 0 & 0 & 0 & \cdots & 1 \end{bmatrix}$$

Question 1. What happen if $c_n = f(n)$, i.e., c_n is not a constant but variable?

2.1 Graphs & matrices

Theorem 1. If A is an adjacency matrix of an unweighted graph, then the matrix A^n gives for each node pair (a, b) the number of paths that begin at node a , end at node b & contain exactly n edges, which is allowed that a node appears on a path several times.

Chứng minh.

□

Using a similar idea in a weighted graph, we can calculate for each node pair (a, b) the shortest length of a path that goes from node a to node b & contains exactly n edges by defining matrix multiplication in the following way such that we do not calculate numbers of paths but minimize lengths of paths. We define an adjacency matrix where ∞ means that an edge does not exist, & other values correspond to edge weights:

$$A_{ij} = \begin{cases} \infty & \text{if } i \rightarrow j \notin \mathcal{E}, \\ w_{ij} & \text{if } i \rightarrow j \in \mathcal{E}, \end{cases}, \quad \forall i, j \in [n].$$

Instead of the formula

$$(AB)_{ij} = \sum_{k=1}^n A_{ik} B_{kj},$$

we now use the formula

$$(AB)_{ij} = \min_{k=1}^n A_{ik} + B_{kj}$$

for matrix multiplication, so we calculate minima instead of sums, & sums of elements instead of products. After this modification, matrix powers minimize path lengths in the graph. Then $(A^e)_{ij}$ is the minimum length of a path of e edges from node i to node j .

Problem 2 (CSES Problem Set/graph paths I). Consider a directed graph that has n nodes & m edges. Count the number of paths from node 1 to node n with exactly k edges.

Input. The 1st input line contains 3 integers n, m, k : the number of nodes & edges, & the length of the path. The nodes are numbered $1, 2, \dots, n$. Then, there are m lines describing the edges. Each line contains 2 integers a, b : there is an edge from node a to node b .

Output. Print the number of paths modulo $10^9 + 7$.

Constraints. $n \in [100], m \in [n(n - 1)], k \in [10^9], a, b \in [n]$.

Solution. Let $A \in \mathcal{M}_n(\mathbb{Z})$ be the adjacency matrix of this graph. The answer is $(A^k)_{1n}$.

C++ implementation.

1. Olympia's:

```

1 #include <bits/stdc++.h>
2 #pragma GCC target ("avx2")
3 #pragma GCC optimization ("O3")
4 #pragma GCC optimization ("unroll-loops")
5
6 using namespace std;
7 const int MOD = 1e9 + 7;
8
9 class Matrix {
10 public:
11     vector<vector<int64_t>> v;
12     void print() {
13         for (int i = 0; i < v.size(); ++i) {
14             for (int j : v[i]) cout << j << ' ';
15             cout << '\n';
16         }
17     }
18     Matrix operator* (Matrix m1) const {
19         Matrix ans(v);
20         for (int i = 0; i < m1.v.size(); ++i)
21             for (int j = 0; j < m1.v.size(); ++j) ans.v[i][j] = 0;
22         for (int i = 0; i < m1.v.size(); ++i)
23             for (int j = 0; j < m1.v.size(); ++j)
24                 for (int k = 0; k < m1.v.size(); ++k) {
25                     ans.v[i][j] += (v[i][k] * m1.v[k][j]) % MOD;
26                     ans.v[i][j] %= MOD;
27                 }
28         return ans;
29     }
30     Matrix identity (int64_t n) {
31         vector<vector<int64_t>> vec(n);
32         for (int i = 0; i < n; ++i) {
33             vec[i].resize(n);
34             for (int j = 0; j < n; ++j) vec[i][j] = (i == j);
35         }
36         return Matrix(vec);
37     }
38     Matrix operator^ (int64_t x) {
39         Matrix ans = identity(v.size()), res = *this;
40         while (x > 0) {
41             if (x & 1) ans = res * ans;
42             res = res * res;
43             x /= 2;
44         }
45         return ans;
46     }
47     Matrix (vector<vector<int64_t>> v) {
48         this->v = v;
49     }
50 };
51
52 int main() {
53     ios_base::sync_with_stdio(false);

```

```

54     cin.tie(nullptr);
55     int n, m, k;
56     cin >> n >> m >> k;
57     vector<vector<int64_t>> v(n);
58     for (int i = 0; i < n; ++i) v[i].assign(n, 0);
59     while (m--) {
60         int x, y;
61         cin >> x >> y;
62         --x, --y;
63         ++v[x][y];
64     }
65     Matrix fib = Matrix(v);
66     fib = fib^(k);
67     cout << fib.v[0][n - 1];
68 }
```

2. TodomoTachi's:

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const long long MOD = 1e9 + 7;
4
5 #define MAX_SIZE 100
6 #define ll long long
7
8 struct Matrix {
9     int m, n; // m = số hàng, n = số cột
10    ll d[MAX_SIZE][MAX_SIZE];
11    Matrix (int _m = 0, int _n = 0) {
12        m = _m; n = _n;
13        memset(d, 0, sizeof d);
14    }
15
16    Matrix operator + (const Matrix &a) const { // phép cộng ma trận
17        Matrix res(m, n);
18        for (int i = 0; i < m; ++i)
19            for (int j = 0; j < n; ++j) {
20                res.d[i][j] = d[i][j] + a.d[i][j];
21                if (res.d[i][j] >= MOD) res.d[i][j] -= MOD;
22            }
23        return res;
24    }
25
26    Matrix operator * (const Matrix &a) const { // phép nhân ma trận
27        ll x = m, y = n, z = a.n;
28        Matrix res(x, z);
29        for (int i = 0; i < x; ++i)
30            for (int j = 0; j < y; ++j)
31                for (int k = 0; k < z; ++k) res.d[i][k] = (res.d[i][k] + 1LL * d[i][j] * a.d[j][k]) % MOD;
32        return res;
33    }
34
35    Matrix operator ^ (ll k) const { // phép luỹ thừa ma trận
36        Matrix res(n, n);
37        for (int i = 0; i < n; ++i) res.d[i][i] = 1;
38        Matrix mul = *this;
39        while (k > 0) {
40            if (k & 1) res = res * mul;
41            mul = mul * mul;
42            k >>= 1;
43        }
44        return res;
45    }
46 };
47
```

```

48 int main() {
49     cin.tie(0) -> sync_with_stdio(0);
50     int n, m, k;
51     cin >> n >> m >> k;
52     Matrix t(n, n);
53     for (int i = 0, x, y; i < m; ++i) {
54         cin >> x >> y;
55         ++t.d[x - 1][y - 1]; // attention: maybe have duplicate edge
56     }
57     t = t ^ k;
58     cout << t.d[0][n - 1];
59 }

```

3. Pilla Venkata Sekhar's:

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 const ll MOD = 1e9 + 7;
5
6 vector<vector<ll>> mat(101, vector<ll>(101, 0));
7 vector<vector<ll>> mat_mul(vector<vector<ll>> mat1, vector<vector<ll>> mat2, ll sz) {
8     vector<vector<ll>> mul(sz, vector<ll>(sz, 0));
9     for (int i = 0; i < sz; ++i)
10        for (int j = 0; j < sz; ++j) {
11            int cur = 0;
12            for (int k = 0; k < sz; ++k) {
13                cur += (mat1[i][k] * mat2[k][j]) % MOD;
14                cur %= MOD;
15            }
16            mul[i][j] = cur;
17        }
18    return mul;
19 }
20
21 ll mat_expo(vector<vector<ll>> pow, ll sz, ll n) {
22     vector<vector<ll>> ans(sz, vector<ll>(sz, 0));
23     for (int i = 0; i < sz; ++i) ans[i][i] = 1;
24     while (n) {
25         if (n & 1) ans = mat_mul(ans, pow, sz);
26         pow = mat_mul(pow, pow, sz);
27         n /= 2;
28     }
29     return ans[1][sz - 1];
30 }
31
32 int main() {
33     int a, b, n, m, k;
34     cin >> n >> m >> k;
35     for (int i = 0; i < m; ++i) {
36         cin >> a >> b;
37         ++mat[a][b];
38     }
39     cout << mat_expo(mat, n + 1, k);
40 }

```

4. Dan4Life's:

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using ll = long long;
4
5 const int MOD = 1e9 + 7;
6 int n, m, k, x, y;
7

```

```

8 struct Matrix {
9     ll a[110][110];
10    Matrix() {
11        for (int i = 0; i < 110; ++i) fill(a[i], a[i] + 110, 0ll);
12    }
13 };
14
15 Matrix M, I;
16
17 Matrix mult(Matrix x, Matrix y) {
18     Matrix z;
19     for (int i = 0; i < n; ++i)
20         for (int j = 0; j < n; ++j)
21             for (int k = 0; k < n; ++k) z.a[i][j] += x.a[i][k] * y.a[k][j], z.a[i][j] %= MOD;
22     return z;
23 }
24
25 Matrix pow(Matrix x, int b) {
26     if (!b) return I;
27     Matrix y = pow(x, b / 2);
28     y = mult(y, y);
29     if (b & 1) y = mult(y, x);
30     return y;
31 }
32
33 int main() {
34     ios_base::sync_with_stdio(false);
35     cin.tie(0);
36     cin >> n >> m >> k;
37     for (int i = 0; i < n; ++i) I.a[i][i] = 1;
38     while (m--) {
39         cin >> x >> y;
40         --x, --y;
41         ++M.a[x][y];
42     }
43     cout << pow(M, k).a[0][n - 1];
44 }

```

□

Problem 3 (CSES Problem Set/graph paths II). Consider a directed weighted graph having n nodes & m edges. Calculate the minimum path length from node 1 to node n with exactly k edges.

Input. The 1st input line contains 3 integers n, m, k : the number of nodes & edges, & the length of the path. The nodes are numbered $1, 2, \dots, n$. Then, there are m lines describing the edges. Each line contains 3 integers a, b, c : there is an edge from node a to node b with weight c .

Output. Print the minimum path length. If there are no such paths, print -1 .

Constraints. $n \in [100]$, $m \in [n(n - 1)]$, $k \in [10^9]$, $a, b \in [n]$, $c \in [10^9]$.

Solution.

C++ implementation.

1. Pilla Venkata Sekhar's:

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4
5 const ll MOD = 1e9 + 7;
6 vector<vector<ll>> mat(101, vector<ll>(101, 0));
7
8 vector<vector<ll>> mat_mul(vector<vector<ll>> mat1, vector<vector<ll>> mat2, ll sz) {
9     vector<vector<ll>> mul(sz, vector<ll>(sz, 0));
10    for (int i = 0; i < sz; ++i)
11        for (int j = 0; j < sz; ++j) {

```

```

12         ll cur = 0;
13         for (int k = 0; k < sz; ++k)
14             if (mat1[i][k] > 0 && mat2[k][j] > 0) {
15                 if (cur) cur = min(cur, (mat1[i][k] + mat2[k][j]));
16                 else cur = mat1[i][k] + mat2[k][j];
17             }
18             mul[i][j] = cur;
19         }
20     return mul;
21 }
22
23 ll mat_expo(vector<vector<ll>> pow, ll sz, ll n) {
24     vector<vector<ll>> ans(sz, vector<ll>(sz, 0));
25     int check = 0;
26     while (n) {
27         if (n & 1) {
28             if (check) ans = mat_mul(ans, pow, sz);
29             else {
30                 check = 1;
31                 for (int i = 0; i < sz; ++i)
32                     for (int j = 0; j < sz; ++j) ans[i][j] = pow[i][j];
33             }
34         }
35         pow = mat_mul(pow, pow, sz);
36         n >= 1;
37     }
38     if (ans[1][sz - 1]) return ans[1][sz - 1];
39     return -1;
40 }
41
42 int main() {
43     ios_base::sync_with_stdio(false);
44     cin.tie(nullptr);
45     int n, m, k;
46     cin >> n >> m >> k;
47     ll a, b, c;
48     for (int i = 0; i < m; ++i) {
49         cin >> a >> b >> c;
50         if (!mat[a][b]) mat[a][b] = c;
51         else mat[a][b] = min(mat[a][b], c);
52     }
53     cout << mat_expo(mat, n + 1, k);
54 }

```

2. Dan4Life's:

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using ll = long long;
4
5 const int MOD = 1e9 + 7;
6 const ll LINF = 4e18;
7 int n, m, k, x, y, z;
8
9 struct Matrix {
10     ll a[110][110];
11     Matrix (ll v = 0) {
12         for (int i = 0; i < 110; ++i) fill(a[i], a[i] + 110, v);
13     }
14 };
15
16 Matrix M(LINF), I(LINF);
17
18 Matrix mult(Matrix x, Matrix y) {
19     Matrix z(LINF);

```

```

20     for (int i = 0; i < n; ++i)
21         for (int j = 0; j < n; ++j)
22             for (int k = 0; k < n; ++k) z.a[i][j] = min(z.a[i][j], x.a[i][k] + y.a[k][j]);
23     return z;
24 }
25
26 Matrix pow(Matrix x, int b) {
27     if (!b) return I;
28     if (b == 1) return x;
29     Matrix y = pow(x, b / 2);
30     y = mult(y, y);
31     if (b & 1) y = mult(y, x);
32     return y;
33 }
34
35 int main() {
36     ios_base::sync_with_stdio(false);
37     cin.tie(0);
38     cin >> n >> m >> k;
39     while (m--) {
40         cin >> x >> y >> z;
41         --x, --y;
42         M.a[x][y] = min(M.a[x][y], (ll)z);
43     }
44     cout << (pow(M, k).a[0][n - 1] < LINF ? pow(M, k).a[0][n - 1] : -1);
45 }

```

3. Bùi Trung Hiếu's:

```

1 #include <bits/stdc++.h>
2 #define FOR(i, a, b) for (int i = (a), _b = (b); i <= _b; ++i)
3 #define FORD(i, b, a) for (int i = (b), _a = (a); i >= _a; --i)
4 #define REP(i, n) for (int i = 0, _n = (n); i < _n; ++i)
5 #define FORE(i, v) for (decltype((v).begin()) i = (v).begin(); i != (v).end(); ++i)
6 using namespace std;
7 using ll = long long;
8
9 const int MAXN = 106;
10 const ll INF = 4e18;
11 int n, m, k;
12
13 struct Matrix {
14     int n, m;
15     ll d[MAXN][MAXN];
16
17     Matrix (int _n, int _m) {
18         n = _n, m = _m;
19         REP(i, MAXN) REP(j, MAXN) d[i][j] = INF;
20     }
21
22     Matrix operator * (const Matrix &a) const {
23         int x = n, y = m, z = a.m;
24         Matrix res(x, z);
25         REP(i, x) REP(j, y) REP(k, z) res.d[i][k] = min(res.d[i][k], d[i][j] + a.d[j][k]);
26         return res;
27     }
28
29     Matrix operator ^ (int k) const {
30         Matrix res(n, n), mul = *this;
31         REP(i, n) res.d[i][i] = 0;
32         while (k > 0) {
33             if (k & 1) res = res * mul;
34             mul = mul * mul;
35             k >>= 1;
36         }

```

```

37         return res;
38     }
39 }
40
41 int main() {
42     ios_base::sync_with_stdio(0);
43     cin.tie(NULL);
44     cout.tie(NULL);
45     cin >> n >> m >> k;
46     Matrix trans(n, n);
47     FOR(i, 1, m) {
48         int x, y, w;
49         cin >> x >> y >> w;
50         --x, --y;
51         trans.d[x][y] = min(trans.d[x][y], (ll)w);
52     }
53     trans = trans ^ k;
54     if (trans.d[0][n - 1] < INF) cout << trans.d[0][n - 1];
55     else cout << -1;
56 }

```

4. Viktor Maksimoski's:

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using ll = long long;
4
5 struct Mat {
6     int n, m;
7     vector<vector<ll>> mat;
8
9     Mat (int _n, int _m) {
10         n = _n, m = _m;
11         mat.resize(n, vector<ll>(m));
12     }
13
14     Mat(vector<vector<ll>> v) {
15         mat = v;
16         n = (int)v.size(), m = (int)v[0].size();
17     }
18 };
19
20 Mat ID (int n) {
21     Mat ans(n, n);
22     for (int i = 0; i < n; ++i)
23         for (int j = 0; j < n; ++j) ans.mat[i][j] = 2e18;
24     return ans;
25 }
26
27 Mat mul(Mat a, Mat b) {
28     Mat ans = ID(a.n);
29     for (int i = 0; i < a.n; ++i)
30         for (int j = 0; j < b.m; ++j)
31             for (int k = 0; k < a.m; ++k) ans.mat[i][j] = min(ans.mat[i][j], a.mat[i][k] + b.mat[k][j]);
32     return ans;
33 }
34
35 Mat exp(Mat a, ll b) {
36     Mat ans = ID(a.n);
37     for (int i = 0; i < a.n; ++i) ans.mat[i][i] = 0;
38     while (b) {
39         if (b & 1) ans = mul(ans, a);
40         a = mul(a, a);
41         b >>= 1;
42     }

```

```

43     return ans;
44 }
45
46 int main() {
47     ios_base::sync_with_stdio(0);
48     cin.tie(NULL);
49     int n, m, k;
50     cin >> n >> m >> k;
51     Mat a = ID(n);
52     while (m--) {
53         int x, y, z;
54         cin >> x >> y >> z;
55         a.mat[x - 1][y - 1] = min(a.mat[x - 1][y - 1], (1l)z);
56     }
57     a = exp(a, k);
58     cout << (a.mat[0][n - 1] > 1e18 ? -1 : a.mat[0][n - 1]) << '\n';
59 }

```

5. Olympia's:

```

1 #include <bits/stdc++.h>
2 #pragma GCC target ("avx2")
3 #pragma GCC optimization ("O3")
4 #pragma GCC optimization ("unroll-loops")
5 using namespace std;
6 const int MOD = 1e9 + 7;
7
8 int64_t chmin(int64_t x, int64_t y) {
9     if (x == -1) return y;
10    if (y == -1) return x;
11    return min(x, y);
12 }
13
14 class Matrix {
15     public:
16     vector<vector<int64_t>> v;
17
18     void print() {
19         for (int i = 0; i < v.size(); ++i) {
20             for (int j : v[i]) cout << j << ' ';
21             cout << '\n';
22         }
23     }
24
25     Matrix operator * (Matrix m1) const {
26         Matrix ans(v);
27         for (int i = 0; i < m1.v.size(); ++i)
28             for (int j = 0; j < m1.v.size(); ++j) ans.v[i][j] = -1;
29         for (int i = 0; i < m1.v.size(); ++i)
30             for (int j = 0; j < m1.v.size(); ++j) {
31                 ans.v[i][j] = -1;
32                 for (int k = 0; k < m1.v.size(); ++k) {
33                     if (v[i][k] == -1 || m1.v[k][j] == -1) continue;
34                     ans.v[i][j] = chmin(v[i][k] + m1.v[k][j], ans.v[i][j]);
35                 }
36             }
37         return ans;
38     }
39
40     Matrix identity (int64_t n) {
41         vector<vector<int64_t>> vec(n);
42         for (int i = 0; i < n; ++i) {
43             vec[i].resize(n);
44             for (int j = 0; j < n; ++j) vec[i][j] = (i == j);
45         }

```

```

46         return Matrix(vec);
47     }
48
49     Matrix operator^ (int64_t x) {
50         Matrix ans = *this, res = *this;
51         while (x > 0) {
52             if (x & 1) ans = res * ans;
53             res = res * res;
54             x >>= 1;
55         }
56         return ans;
57     }
58
59     Matrix (vector<vector<int64_t>> v) {
60         this->v = v;
61     }
62 };
63
64 int main() {
65     ios_base::sync_with_stdio(false);
66     cin.tie(NULL);
67     int n, m, k;
68     cin >> n >> m >> k;
69     vector<vector<int64_t>> v(n);
70     for (int i = 0; i < n; ++i) v[i].assign(n, -1);
71     while (m--) {
72         int x, y, w;
73         cin >> x >> y >> w;
74         --x, --y;
75         v[x][y] = chmin(v[x][y], w);
76     }
77     Matrix fib = Matrix(v);
78     fib = fib ^ (k - 1);
79     cout << fib.v[0][n - 1];
80 }

```

□

Problem 4 (Codeforces/Educational Codeforces Round 14/E. Xor-sequences). You are given n integers a_1, a_2, \dots, a_n . A sequence of integers x_1, x_2, \dots, x_k is called a xor-sequence if for every $i \in [k-1]$ the number of ones in the binary representation of the number $x_i \otimes x_{i+1}$'s is a multiple of 3 & $x_i \in \{a_1, a_2, \dots, a_n\}$ for all $i \in [k]$. The symbol \otimes is used for the binary exclusive or operation. How many xor-sequences of length k exist? Output the answer modulo $10^9 + 7$. Note if $a = [1, 1]$ & $k = 1$ then the answer is 2, because you should consider the ones from a as different.

Input. The 1st line contains 2 integers $n \in [100], k \in [10^{18}]$ – the number of given integers & the length of the xor-sequences. The 2nd line contains n integers $0 \leq a_i \leq 10^{18}, \forall i \in [n]$.

Output. Print the only integer c – the number of xor-sequences of length k modulo $10^9 + 7$.

1st solution. Source: [Codeforces/Edvard's blog/editorial of Educational Codeforces round 14](#). Let b_{ij} be the number of xor-sequences of length i with the last element equal to a_j . Let $g_{ij} = 1$ if $a_i \otimes a_j$ contains the number of 1s in binary representation that is multiple of 3. Otherwise let $g_{ij} = 0$. Consider a vectors $b_i = \{b_{ij}\}, b_{i-1} = \{b_{i-1,j}\}$ & a matrix $G = \{g_{ij}\}$. One has $b_i = Gb_{i-1}$. So $b_n = G^n b_0$. Using the associative property of matrix multiplication: at 1st, we calculate G^n with binary matrix exponentiation & then multiply it to the vector z_0 .

C++ implementation.

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using ll = long long;
4
5 const int N = 101, mod = 1000000007;
6 int n;
7 ll k, a[N];
8
9 bool read() {
10     if (!(cin >> n >> k)) return false;

```

```

11     for (int i = 0; i < n; ++i) assert(cin >> a[i]);
12     return true;
13 }
14
15 inline int add(int a, int b) { return a + b >= mod ? a + b - mod : a + b; }
16 inline void inc(int& a, int b) { a = add(a, b); }
17 inline int mul(int a, int b) { return int(a * 1ll * b % mod); }
18
19 int count(ll x) {
20     int ans = 0;
21     while (x) {
22         ++ans;
23         x &= x - 1;
24     }
25     return ans;
26 }
27
28 void mul(int a[N][N], int b[N][N], int n) {
29     static int c[N][N];
30     for (int i = 0; i < n; ++i)
31         for (int j = 0; j < n; ++j) {
32             c[i][j] = 0;
33             for (int k = 0; k < n; ++k) inc(c[i][j], mul(a[i][k], b[k][j]));
34         }
35     for (int i = 0; i < n; ++i)
36         for (int j = 0; j < n; ++j) a[i][j] = c[i][j];
37 }
38
39 void bin_pow(int a[N][N], ll b, int n) {
40     static int ans[N][N];
41     for (int i = 0; i < n; ++i)
42         for (int j = 0; j < n; ++j) ans[i][j] = i == j;
43     while (b) {
44         if (b & 1) mul(ans, a, n);
45         mul(a, a, n);
46         b >>= 1;
47     }
48     for (int i = 0; i < n; ++i)
49         for (int j = 0; j < n; ++j) a[i][j] = ans[i][j];
50 }
51
52 void solve() {
53     static int a[N][N];
54     memset(a, 0, sizeof(a));
55     for (int i = 0; i < n; ++i) {
56         for (int j = 0; j < n; ++j)
57             a[i][j] = count(::a[i] ^ ::a[j]) % 3 == 0;
58         a[i][n] = 1;
59     }
60     bin_pow(a, k, n + 1);
61     int ans = 0;
62     for (int i = 0; i <= n; ++i) inc(ans, a[i][n]);
63     cout << ans << '\n';
64 }
65
66 int main() {
67     ios_base::sync_with_stdio(false);
68     cin.tie(NULL);
69     if (read()) solve();
70 }

```



2nd solution. Source: NEO WANG. <https://usaco.guide/problems/cf-xor-sequences/solution>. We are asked to count the total number of paths, so we can store the graph as an adjacency matrix by setting $a_{ij} = 1$ if the number of bits $x_i \oplus x_j$ is

divisible by 3. After constructing these edges, this is the same problem as [CSES Problem Set/graph paths I](#), except we query the sum of all the paths. Then, we can calculate A^k in $O(n^3 \log k)$ using matrix exponentiation. The final answer is

$$\sum_{i=0}^n \sum_{j=0}^n a_{ij}^k.$$

C++ implementation.

1. Neo Wang's:

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using ll = long long;
4 const ll MOD = 1e9 + 7;
5
6 template <class T> struct Matrix {
7     vector<vector<T>> v;
8     void init(int n, int m) { v = vector<vector<T>>(n, vector<T>(m)); }
9
10    Matrix operator* (Matrix b) {
11        int x = v.size(), y = v[0].size(), z = b.v[0].size();
12        assert(y == size(b.v));
13        Matrix<T> ret;
14        ret.init(x, z);
15        for (int i = 0; i < x; ++i)
16            for (int j = 0; j < y; ++j)
17                for (int k = 0; k < z; ++k) {
18                    ret.v[i][k] += v[i][j] * b.v[j][k];
19                    ret.v[i][k] %= MOD;
20                }
21        return ret;
22    }
23 };
24
25 int main() {
26     ios_base::sync_with_stdio(0);
27     cin.tie(NULL);
28     ll n, k;
29     cin >> n >> k;
30     --k;
31     vector<ll> a(n);
32     for (int i = 0; i < n; ++i) cin >> a[i];
33     Matrix<ll> A, B;
34     A.init(n, n);
35     B.init(n, n);
36     for (int i = 0; i < n; ++i)
37         for (int j = 0; j < n; ++j)
38             if (_builtin_popcountll(a[i] ^ a[j]) % 3 == 0) A.v[i][j] = 1;
39     Matrix<ll> ret;
40     ret.init(n, n);
41     for (int i = 0; i < n; ++i) ret.v[i][i] = 1;
42     for (ll b = k; b > 0; b >>= 1) {
43         if (b & 1) ret = ret * A;
44         A = A * A;
45     }
46     ll ans = 0;
47     for (int i = 0; i < n; ++i)
48         for (int j = 0; j < n; ++j) {
49             ans += ret.v[i][j];
50             ans %= MOD;
51         }
52     cout << ans << '\n';
53 }
```

2. Olympia's:

```

1 #include <bits/stdc++.h>
2 #define ll long long
3 using namespace std;
4 const int MOD = 1e9 + 7;
5
6 struct Matrix {
7     vector<vector<ll>> mat1;
8     Matrix operator *(const Matrix& other) {
9         vector<vector<ll>> mat2 = other.mat1;
10        vector<vector<ll>> ans(mat1.size());
11        for (int i = 0; i < ans.size(); ++i) {
12            ans[i].resize(mat2[0].size());
13            for (int j = 0; j < ans[i].size(); ++j) {
14                ans[i][j] = 0;
15                for (int k = 0; k < mat1[0].size(); ++k) {
16                    ans[i][j] += mat1[i][k] * mat2[k][j];
17                    ans[i][j] %= MOD;
18                }
19            }
20        }
21        Matrix m;
22        m.mat1 = ans;
23        return m;
24    }
25
26    Matrix identity (int n) {
27        Matrix ans;
28        ans.mat1.resize(n);
29        for (int i = 0; i < n; ++i) {
30            ans.mat1[i].resize(n);
31            for (int j = 0; j < n; ++j) ans.mat1[i][j] = (i == j);
32        }
33        return ans;
34    }
35
36    Matrix binpow (ll powr) {
37        Matrix res = {mat1}, ans = identity(mat1.size());
38        while (powr > 0) {
39            if (powr & 1) ans = ans * res;
40            res = res * res;
41            powr >>= 1;
42        }
43        return ans;
44    }
45};
46
47 bool valid (ll x, ll y) {
48     return !(_builtin_popcountll(x ^ y)) % 3;
49 }
50
51 int main() {
52     ios_base::sync_with_stdio(false);
53     cin.tie(NULL);
54     ll n, k;
55     cin >> n >> k;
56     vector<ll> a(n);
57     for (int i = 0; i < n; ++i) cin >> a[i];
58     Matrix adj;
59     adj.mat1.resize(n);
60     for (int i = 0; i < n; ++i) {
61         adj.mat1[i].resize(n);
62         for (int j = 0; j < n; ++j) adj.mat1[i][j] = valid(a[i], a[j]);
63     }
64     Matrix ans = adj.binpow(k - 1);

```

```

65     ll res = 0;
66     for (int i = 0; i < n; ++i)
67         for (int j = 0; j < n; ++j) {
68             res += ans.mat1[i][j];
69             res %= MOD;
70         }
71     cout << res;
72 }
```

3. Pilla Venkata Sekhar's:

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 const ll MOD = 1e9+7;
5
6 vector<vector<ll>> mat_mul(vector<vector<ll>> mat1, vector<vector<ll>> mat2, int sz) {
7     vector<vector<ll>> mul(sz, vector<ll>(sz, 0));
8     for (int i = 0; i < sz; ++i)
9         for (int j = 0; j < sz; ++j)
10            for (int k = 0; k < sz; ++k) {
11                mul[i][j] += mat1[i][k] * mat2[k][j];
12                mul[i][j] %= MOD;
13            }
14     return mul;
15 }
16
17 int mat_expo(vector<vector<ll>> p, int sz, ll pow) {
18     vector<vector<ll>> ans(sz, vector<ll>(sz, 0));
19     for (int i = 0; i < sz; ++i) ans[i][i] = 1;
20     if (!pow) return sz;
21     else {
22         while (pow) {
23             if (pow & 1) ans = mat_mul(ans, p, sz);
24             p = mat_mul(p, p, sz);
25             pow >>= 1;
26         }
27     }
28     ll cnt = 0;
29     for (int i = 0; i < sz; ++i)
30         for (int j = 0; j < sz; ++j)
31             if (ans[i][j]) {
32                 cnt += ans[i][j];
33                 cnt %= MOD;
34             }
35     return cnt;
36 }
37
38 int main() {
39     ios_base::sync_with_stdio(false);
40     cin.tie(NULL);
41     int n;
42     cin >> n;
43     ll k, a[n];
44     cin >> k;
45     for (int i = 0; i < n; ++i) cin >> a[i];
46     vector<vector<ll>> mat(n, vector<ll>(n, 0));
47     for (int i = 0; i < n; ++i)
48         for (int j = 0; j < n; ++j) {
49             ll cnt = 0, x = a[i] ^ a[j];
50             while (x) {
51                 if (x & 1) ++cnt;
52                 x >>= 1;
53             }
54             if (cnt % 3 == 0) mat[i][j] = 1, mat[j][i] = 1;
```

```

55     }
56     cout << mat_expo(mat, n, k - 1);
57 }
```

□

Problem 5 ([Codeforces/Codeforces Round 373 \(Div. 1\)/C. Sasha & array](#)). *Sasha has an array of integers a_1, a_2, \dots, a_n . You have to perform m queries. There might be queries of 2 types:*

1. $1\ l\ r\ x$: increase all integers on the segment from l to r by values x ;
2. $2\ l\ r$: find $\sum_{i=l}^r f(a_i)$, where $f(x)$ is the x th Fibonacci number. As this number may be large, you only have to find it modulo $10^9 + 7$.

In this problem we define Fibonacci numbers as follows: $f(1) = 1, f(2) = 1, f(x) = f(x-1) + f(x-2)$ for all $x > 2$. Sasha is a very talented boy & he managed to perform all queries in 5 seconds. Will you be able to write the program that performs as well as Sasha?

Input. The 1st line of the input contains 2 integers $n, m \in [10^5]$ – the number of elements in the array & the number of queries respectively. The next line contains n integers $a_1, a_2, \dots, a_n \in [10^9]$. Then follow m lines with queries descriptions. Each of them contains integers tp_i, l_i, r_i & may be x_i ($1 \leq tp_i \leq 2, 1 \leq l_i \leq r_i \leq n, x_i \in [10^9]$). Here $tp_i = 1$ corresponds to he queries of the 1st type & tp_i corresponds to the queries of the 2nd type. It is guaranteed that the input will contain at least 1 query of the 2nd type.

Output. For each query of the 2nd type print the answer modulo $10^9 + 7$.

1st solution. Source: DUSTIN MIAO. [CF – Sasha & array](#). Let F_i denote the i th Fibonacci number. In this problem, we will build a segment tree over the array. For a leaf node in the segment tree with value v , we will store a pair of values (F_{v-1}, F_v) . For all non-leaf nodes, we will store a pair of values equal to the pair-sum of its children. I.e., if a node u has children v, w , then $u = (v_0 + w_0, v_1 + w_1)$. We will use the term *cycle by k* to denote transforming some pair (F_{i-1}, F_i) to (F_{i-1+k}, F_{i+k}) . By default, the term *cycle* refers to *cycle by 1*. For each update, we need to cycle each leaf node by some amount x such that the value in a node affected goes from (F_{v-1}, F_v) to (F_{v-1+x}, F_{v+x}) . We can do this using matrix exponentiation. 1st note that for a matrix $(F_{i-1} F_i)$, we can cycle it by multiplying by $A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$, i.e.,

$$(F_i F_{i+1}) = (F_{i-1} F_i) \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}.$$

Inductively, we can cycle by k by multiplying multiple times:

$$(F_{i-1+k} F_{i+k}) = (F_{i-1} F_i) \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^k.$$

We can use binary exponentiation to quickly compute powers of the 2×2 matrix.

Secondly, we need to utilize the distributive property of matrix multiplication to same-size matrices, i.e., if m_k is a 1×1 matrix, i.e., a scalar, for $k \in [n]$, then the following property is satisfied:

$$\left(\sum_{i=1}^n m_i \right) \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} = \sum_{i=1}^n m_i \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}.$$

In that way, we can utilize lazy propagation on the segment tree. We will store an integer tag in each node denoting the lazy update, which denotes that we must cycle every node in its subtree by the value of the tag. When propagating, we can simply update the node by binary exponentiating & then multiplying, then storing it in the lazy tag. Querying works as normal on the segment tree.

C++ implementation.

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using ll = long long;
4 using pll = pair<ll, ll>;
5 using Matrix = array<array<ll, 2>, 2>;
6 #define f first
7 #define s second
8
9 const int MAXN = 1e5 + 1;
10 const ll MOD = 1e9 + 7;
11
12 // multiply 2 2 x 2 matrices
```

```

13 Matrix multiply(const Matrix &a, const Matrix &b) {
14     return {(a[0][0] * b[0][0] + a[0][1] * b[1][0]) % MOD,
15             (a[0][0] * b[0][1] + a[0][1] * b[1][1]) % MOD,
16             (a[1][0] * b[0][0] + a[1][1] * b[1][0]) % MOD,
17             (a[1][0] * b[0][1] + a[1][1] * b[1][1]) % MOD};
18 }
19
20 // multiply a 1 x 1 matrix with a 2 x 2 matrix
21 pll multiply(const pll &a, const Matrix &b) {
22     return {(a.f * b[0][0] + a.s * b[1][0]) % MOD,
23             (a.f * b[0][1] + a.s * b[1][1]) % MOD};
24 }
25
26 // returns {0, 1, 1, 1} to the power of n
27 Matrix pow(ll n) {
28     Matrix res = {1, 0, 0, 1}; // identity matrix
29     Matrix base = {0, 1, 1, 1}; // Fibonacci matrix
30     for ( ; n; n >>= 1) { // binary exponentiation (log n)
31         if (n & 1) res = multiply(res, base);
32         base = multiply(base, base);
33     }
34     return res;
35 }
36
37 int n, q;
38 pll tree[MAXN * 4];
39 ll lazy[MAXN * 4];
40
41 // return the pair-sum of a & b
42 pll merge(const pll &a, const pll &b) { return {(a.first + b.first) % MOD, (a.second + b.second) % MOD}; }
43
44 // push lazy update in t to its children
45 void pushdown(int t) {
46     if (!lazy[t]) return;
47     tree[t < 1] = multiply(tree[t < 1], pow(lazy[t]));
48     lazy[t < 1] += lazy[t];
49     tree[t < 1 | 1] = multiply(tree[t < 1 | 1], pow(lazy[t]));
50     lazy[t < 1 | 1] += lazy[t];
51     lazy[t] = 0;
52 }
53
54 // cycle range from l to r by v
55 void update(int l, int r, ll v, int t = 1, int tl = 1, int tr = n) {
56     if (r < tl || tr < l) return;
57     if (l <= tl && tr <= r) {
58         tree[t] = multiply(tree[t], pow(v));
59         lazy[t] += v;
60         return;
61     }
62     pushdown(t);
63     int tm = (tl + tr) >> 1;
64     update(l, r, v, t < 1, tl, tm);
65     update(l, r, v, t < 1 | 1, tm + 1, tr);
66     tree[t] = merge(tree[t < 1], tree[t < 1 | 1]);
67 }
68
69 // query sum from l to r
70 ll query(int l, int r, int t = 1, int tl = 1, int tr = n) {
71     if (r < tl || tr < l) return 0;
72     if (l <= tl && tr <= r) return tree[t].first;
73     pushdown(t);
74     int tm = (tl + tr) >> 1;
75     return (query(l, r, t < 1, tl, tm) + query(l, r, t < 1 | 1, tm + 1, tr)) % MOD;
76 }

```

```

77
78 int main() {
79     ios_base::sync_with_stdio(false);
80     cin.tie(NULL);
81     cin >> n >> q;
82     fill_n(tree, MAXN * 4, pll{0, 1});
83     for (int i = 1; i <= n; ++i) {
84         ll a;
85         cin >> a;
86         update(i, i, a);
87     }
88     while (q--) {
89         int t;
90         cin >> t;
91         if (t == 1) {
92             int l, r;
93             ll v;
94             cin >> l >> r >> v;
95             update(l, r, v);
96         } else if (t == 2) {
97             int l, r;
98             cin >> l >> r;
99             cout << query(l, r) << '\n';
100        }
101    }
102 }
```

Olympia aka. Maria Chrysafis's:

```

1 #include <bits/stdc++.h>
2 #pragma GCC optimize("O1")
3 #pragma GCC optimize("Ofast") // if drop this, TLE on test 18 Codeforces
4 using namespace std;
5 using ll = long long;
6 const int MOD = 1e9 + 7;
7
8 struct Matrix {
9     ll arr[2][2];
10    Matrix operator*(Matrix m1) {
11        Matrix ans;
12        for (int i = 0; i < 2; ++i)
13            for (int j = 0; j < 2; ++j) {
14                ans.arr[i][j] = 0;
15                for (int k = 0; k < 2; ++k) ans.arr[i][j] += (arr[i][k] * m1.arr[k][j]) % MOD;
16                ans.arr[i][j] %= MOD;
17            }
18        return ans;
19    }
20
21    Matrix operator + (Matrix m1) {
22        Matrix ans;
23        for (int i = 0; i < 2; ++i)
24            for (int j = 0; j < 2; ++j) {
25                ans.arr[i][j] = arr[i][j] + m1.arr[i][j];
26                if (ans.arr[i][j] >= MOD) ans.arr[i][j] -= MOD;
27            }
28        return ans;
29    }
30 };
31
32 Matrix powr(Matrix m, ll powr) {
33     Matrix ans = {{1, 0}, {0, 1}}, res = m;
34     while (powr > 0) {
35         if (powr & 1) ans = ans * res;
36         res = res * res;
```

```

37     powr >>= 1;
38 }
39 return ans;
40 }

41
42 vector<Matrix> vec;
43 vector<ll> addLater;
44 Matrix fib = {{0, 1}, {1, 1}};

45
46 void push(ll v) {
47     addLater[2 * v + 1] += addLater[v];
48     vec[2 * v + 1] = vec[2 * v + 1] * powr(fib, addLater[v]);
49     addLater[2 * v] += addLater[v];
50     vec[2 * v] = vec[2 * v] * powr(fib, addLater[v]);
51     addLater[v] = 0;
52 }
53

54 void upd(int dum, ll tl, int tr, int l, int r, ll val) {
55     if (tr < l || tl > r) return;
56     if (tl >= l && tr <= r) {
57         addLater[dum] += val;
58         vec[dum] = vec[dum] * powr(fib, val);
59         return;
60     }
61     push(dum);
62     int mid = (tl + tr) >> 1;
63     upd(2 * dum, tl, mid, l, r, val);
64     upd(2 * dum + 1, mid + 1, tr, l, r, val);
65     vec[dum] = vec[2 * dum] + vec[2 * dum + 1];
66 }
67

68 void upd(int l, int r, ll val) {
69     upd(1, 0, (int)vec.size() / 2 - 1, l, r, val);
70 }

71
72 Matrix get(int dum, int tl, int tr, int &l, int &r) {
73     if (tl > r || tr < l) return {{0, 0}, {0, 0}};
74     if (tl >= l && tr <= r) return vec[dum];
75     push(dum);
76     int tm = (tl + tr) >> 1;
77     return get(dum * 2, tl, tm, l, r) + get(dum * 2 + 1, tm + 1, tr, l, r);
78 }

79
80 Matrix get(int l, int r) {
81     return get(1, 0, (int)vec.size() / 2 - 1, l, r);
82 }

83
84 void resz(ll n) {
85     ll sz = ((1 << (ll)ceil(log2(n)))); 
86     vec.assign(sz * 2, {{1, 0}, {0, 1}});
87     addLater.assign(sz * 2, 0);
88 }

89
90 class CSashaArray {
91     public:
92     void solve(std::istream &in, std::ostream &out) {
93         ios_base::sync_with_stdio(false);
94         cin.tie(NULL);
95         int n, m;
96         in >> n >> m;
97         resz(n);
98         for (int i = 0; i < n; ++i) {
99             int x;
100            in >> x;

```

```

101     upd(i, i, x);
102 }
103 while (m--) {
104     int t;
105     in >> t;
106     if (t == 2) {
107         int u, v;
108         in >> u >> v;
109         --u, --v;
110         out << get(u, v).arr[0][1] << '\n';
111     } else {
112         int l, r, x;
113         in >> l >> r >> x;
114         --l, --r;
115         upd(l, r, x);
116     }
117 }
118 }
119 };
120
121 int main() {
122     CSashaArray solver;
123     std::istream& in(std::cin);
124     std::ostream& out(std::cout);
125     solver.solve(in, out);
126 }

```

□

3 Fast Doubling Technique – Kỹ Thuật Nhân Đôi Nhanh

Question 2. Which linear recurrences can be solved by fast doubling technique?

Question 3. Which nonlinear recurrences can be solved by fast doubling technique?

Bài toán 3 (Biểu thức lượng giác, author: Prof. TRẦN ĐAN THƯ). Với mỗi $n \in \mathbb{N}$, đặt $S_n := 4^n(\sin^{2n} 15^\circ + \cos^{2n} 15^\circ)$. Ta cần tính giá trị $S_n \bmod m$, i.e., số dư khi chia S_n cho m khi S_n có giá trị nguyên; quy ước giá trị này là -1 nếu xảy ra trường hợp S_n không phải là số nguyên.

Input. Mỗi test chứa nhiều test cases. Dòng đầu tiên chứa số test cases t ($1 \leq t \leq 10^4$). Mỗi test case gồm 1 dòng chứa 2 số nguyên n, m cách nhau bởi 1 khoảng trắng ($1 \leq m, n \leq 10^8$).

Output. In ra kết quả $S_n \bmod m$.

Sample.

trigonometric_expression.inp	trigonometric_expression.out
5	0
2 2	0
3 4	2
4 8	0
5 2	52
3 100	

Lưu ý. Biết $\sin 15^\circ$ và $\cos 15^\circ$ là 2 số vô tỷ, nhưng S_n được thiết kế để tạo ra dãy số nguyên, i.e., $S_n \in \mathbb{Z}$, $\forall n \in \mathbb{N}$ (ngoại trừ một số giá trị S_n không nguyên nếu có mà ta đã quy ước dùng -1 thay cho biểu thức $S_n \bmod m$). Các kiểu số thực thông thường của các ngôn ngữ lập trình hiện nay có hiệu ứng sai số không thể chấp nhận được khi làm tròn số thực thành số nguyên. Thuật toán của bạn cần nguyên hóa các công thức, bởi vì việc tính toán không chấp nhận sai sót dù chỉ 1-bit.

1st solution: recurrence. TLE.

C++ implementation.

```

1 #include <iostream>
2 using namespace std;
3 using ll = long long;
4

```

```

5 void solve() {
6     ll n, m;
7     cin >> n >> m;
8     if (m == 1) {
9         cout << "0\n";
10        return;
11    }
12    if (n == 0) {
13        cout << 2 % m << '\n';
14        return;
15    }
16    if (n == 1) {
17        cout << 4 % m << '\n';
18        return;
19    }
20    ll S_prev2 = 2 % m, S_prev1 = 4 % m, S_curr = 0;
21    for (int i = 2; i <= n; ++i) {
22        S_curr = (4 * S_prev1 - S_prev2) % m;
23        if (S_curr < 0) S_curr += m;
24        S_prev2 = S_prev1;
25        S_prev1 = S_curr;
26    }
27    cout << S_curr << '\n';
28 }
29
30 int main() {
31     ios_base::sync_with_stdio(false);
32     cin.tie(NULL);
33     int t;
34     cin >> t;
35     while (t--) solve();
36 }
```

□

2nd solution: fast doubling.

C++ implementation.

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using ll = long long;
4
5 pair<ll, ll> S(int n, int m) {
6     if (!n) return {2 % m, 4 % m}; // base case: S_0 = 2, S_1 = 4
7     auto [Sk, Sk1] = S(n >> 1, m); // S_k & S_{k + 1}
8     // S_{2k} = (S_k)^2 - 2, S_{2k+1} = S_k * S_{k+1} - 4
9     ll S2k = (Sk * Sk - 2) % m, S2k1 = (Sk * Sk1 - 4) % m;
10    if (S2k < 0) S2k += m;
11    if (S2k1 < 0) S2k1 += m;
12    if (n & 1) { // return {S_{2k+1}, S_{2k+2}}
13        ll S2k2 = (4 * S2k1 - S2k) % m;
14        if (S2k2 < 0) S2k2 += m;
15        return {S2k1, S2k2};
16    } else return {S2k, S2k1}; // return {S_{2k}, S_{2k+1}}
17 }
18
19 int main() {
20     ios_base::sync_with_stdio(false);
21     cin.tie(NULL);
22     int t;
23     cin >> t;
24     while (t--) {
25         int n, m;
26         cin >> n >> m;
27         pair<ll, ll> ans = S(n, m);

```

□

3rd solution: matrix multiplication.

C++ implementation.

```

28     cout << ans.first << '\n';
29 }
30 }

3rd solution: matrix multiplication.

C++ implementation.

1 #include <bits/stdc++.h>
2 using namespace std;
3 using ll = long long;
4
5 struct Matrix {
6     ll mat[2][2];
7     Matrix() {
8         mat[0][0] = mat[0][1] = mat[1][0] = mat[1][1] = 0;
9     }
10 };
11
12 Matrix multiply(Matrix A, Matrix B, ll m) {
13     Matrix C;
14     for (int i = 0; i < 2; ++i)
15         for (int j = 0; j < 2; ++j)
16             for (int k = 0; k < 2; ++k)
17                 C.mat[i][j] = (C.mat[i][j] + (A.mat[i][k] * B.mat[k][j]) % m + m) % m;
18     return C;
19 }
20
21 Matrix power(Matrix A, ll p, ll m) {
22     Matrix res;
23     res.mat[0][0] = 1, res.mat[1][1] = 1; // identity matrix
24     A.mat[0][1] %= m;
25     while (p > 0) {
26         if (p & 1) res = multiply(res, A, m);
27         A = multiply(A, A, m);
28         p >>= 1;
29     }
30     return res;
31 }
32
33 int main() {
34     ios_base::sync_with_stdio(false);
35     cin.tie(NULL);
36     int t;
37     cin >> t;
38     while (t--) {
39         int n, m;
40         cin >> n >> m;
41         if (!n) cout << 2 % m << '\n';
42         else if (n == 1) cout << 4 % m << '\n';
43         else {
44             Matrix A;
45             A.mat[0][0] = 4, A.mat[0][1] = -1, A.mat[1][0] = 1, A.mat[1][1] = 0;
46             A = power(A, n - 1, m); // compute A^{n - 1}
47             ll S0 = 2, S1 = 4; // base case
48             ll ans = ((A.mat[0][0] * S1) % m + (A.mat[0][1] * S0) % m) % m;
49             if (ans < 0) ans += m;
50             cout << ans << '\n';
51         }
52     }
53 }
```

□

Remark 1. Bài toán trên có thể mở rộng cho các công thức truy hồi có dạng: ???

4 Miscellaneous

Tài liệu

- [Laa24] Antti Laaksonen. *Guide to Competitive Programming: Learning & Improving Algorithms Through Contests*. 3rd edition. Undergraduate Topics in Computer Science. Springer, 2024, pp. xviii+349.