

Matrix Multiplication & Fast Doubling Techniques in Competitive Programming

Nguyễn Quân Bá Hồng*

Ngày 16 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](#).
T_EX: URL: [.tex](#).
- .
PDF: URL: [.pdf](#).
T_EX: 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	6
4 Miscellaneous	6
Tài liệu	6

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 value sof $f(0), f(1), \dots, f(n)$ 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 .

*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>.

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 .

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 $\{f_n\}_{n \in \mathbb{N}}$ 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.

Giải. 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), $a_{k+2} = \sum_{i=1}^k c_i a_{k+2-i} = c_1 a_{k+1} + c_2 a_k + \cdots + c_k a_2$ 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 .

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?

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.