Matrix Multiplication & Fast Doubling Techniques in Competitive Programming

Nguyễn Quản Bá Hồng*

Ngày 15 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.
- .
  PDF: URL: .pdf.
  TEX: URL: .tex.

Mục lục

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* $f:\mathbb{N}\to\mathbb{C}$ *whose initial values are* $f(0),f(1),\ldots,f(k-1)$ *& larger values are calculated recursively using the formula*

$$f(n)=\sum_{i=1}^k c_if(n-i)=c_1f(n-1)+c_2f(n-2)+\cdots+c_kf(n-k),$$

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 $f(0),f(1),\ldots,f(n)$ 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* $a_0,a_1,\ldots,a_{k-1}$ *&* k *số* $c_1,c_2,\ldots,c_k\in\mathbb{Z}$ *được cho trước, được định nghĩa thông qua quan hệ truy hồi tuyến tính có dạng*

$$a_n=\sum_{i=1}^k c_ia_{n-i}=c_1a_{n-1}+c_2a_{n-2}+\cdots+c_ka_{n-k},$$

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* $a_0, a_1, \ldots, a_{k-1}$. *Dòng 3 chứa* k *số nguyên* $c_1, c_2, \ldots, c_k$.

Output. *In ra* a_n.

C++ implementation.

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

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    int n, k;
    cin >> n >> k;
    vector<int> a(n + 1), c(k + 1);
    for (int i = 0; i < k; ++i) cin >> a[i]; // input initial values a_0, a_1,..., a_{k - 1}
    for (int i = 1; i <= k; ++i) cin >> c[i]; // input constant coefficients c_1, c_2,..., c_k
    for (int i = k; i <= n; ++i)
        for (int j = 1; j <= k; ++j) a[i] += c[j] * a[i - j];
    cout << a[n] << '\n';
}
```

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:

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    int n, k, m;
    cin >> n >> k >> m;
    vector<ll> a(n + 1), c(k + 1);
    for (int i = 0; i < k; ++i) cin >> a[i]; // input initial values a_0, a_1,..., a_{k - 1}
    for (int i = 1; i <= k; ++i) cin >> c[i]; // input constant coefficients c_1, c_2,..., c_k
    for (int i = k; i <= n; ++i) {
        for (int j = 1; j <= k; ++j) a[i] += c[j] * a[i - j];
        a[i] %= m;
    }
    cout << a[n] << '\n';
}
```

2 Matrix Multiplication – Nhân Ma Trận

Resources – Tài nguyên.

1. Benjamin Qi, Harshini Rayasam, Neo Wang, Peng Bai. USACO Guide/matrix exponentatiation.

2. CodeForces/lazyneuron/a complete guide on matrix exponentiation.

3.

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

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

Calculate the value of F_n *for a given* n.

Input. *The only input line has an integer* n.

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

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

$$A^n = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n = \begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix},\ \forall n \in \mathbb{N}^\star. \tag{2}$$

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 (2) đúng theo nguyên lý quy nạp toán học.

C++ implementation.

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
using Matrix = array<array<ll, 2>, 2>;
const ll MOD = 1e9 + 7;

Matrix mul(Matrix a, Matrix b) {
    Matrix res = {{{0, 0}, {0, 0}}};
    for (int i = 0; i < 2; ++i)
        for (int j = 0; j < 2; ++j)
            for (int k = 0; k < 2; ++k) {
                res[i][j] += a[i][k] * b[k][j];
                res[i][j] %= MOD;
            }
    return res;
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    ll n;
    cin >> n;
    Matrix base = {{{1, 0}, {0, 1}}}, m = {{{1, 1}, {1, 0}}};
    for (; n > 0; n /= 2, m = mul(m, m))
        if (n & 1) base = mul(base, m);
    cout << base[0][1];
}
```

□

Ta có thể mở rộng bài toán này bằng cách mở rộng (1) 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.

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 $a_1, a_2, \ldots, a_k$ 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 $a_2, a_3, \ldots, a_{k+1}$). 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 $a_i \to a_{i+1}$ 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: $O(k^3 \log n)$.

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

const int MOD = 1e9;;

template <typename T> void matmul(vector<vector<T>> &a, vector<vector<T>> b) {
    int n = a.size(), m = a[0].size(), p = b[0].size();
    assert(m == b.size());
    vector<vector<T>> c(n, vector<T>(p));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < p; ++j)
            for (int k = 0; k < m; ++k) c[i][j] = (c[i][j] + a[i][k] + b[k][j]) % MOD;
    a = c;
}

template <typename T> struct Matrix {
    vector<vector<T>> mat;
    Matrix() {}
    Matrix(vector<vector<T>> a) { mat = a; }
    Matrix(int n, int m) {
        mat.resize(n);
        for (int i = 0; i < n; ++i) mat[i].resize(m);
    }
    int rows() const { return mat.size(); }
    int cols() const { return mat[0].size(); }

    // make the identity matrix for a n x n matrix
    void makeiden() {
        for (int i = 0; i < rows(); ++i) mat[i][i] = 1;
    }

    void print() const {
        for (int i = 0; i < rows; ++i) {
            for (int j = 0; j < cols(); ++j) cout << mat[i][j] << ' ';
            cout << '\n';
        }
    }

    Matrix operator*=(const Matrix &b) {
        matmul(mat, b.mat);
        return *this;
    }

```

```
    Matrix operator*(const Matrix &b) { return Matrix(*this) *= b; }
};

int main() {
    int test_num;
    cin >> test_num;
    for (int t = 0; t < test_num; ++t) {
        int n, k;
        cin >> k;
        Matrix<ll> mat(k, k), vec(k, 1), cur(k, k);
        cur.makeiden();
        for (int i = 0; i < k; ++i) cin >> vec.mat[i][0];
        for (int i = 0; i < k; ++i) cin >> mat.mat[k - 1][k - i - 1];
        for (int i = 1; i < k; ++i) mat.mat[i - 1][i] = 1;
        cin >> n;
        --n;
        while (n > 0) {
            if (n & 1) cur *= mat;
            mat *= mat;
            n >>= 1;
        }
        Matrix<ll> res = cur * vec;
        cout << res.mat[0][0] << '\n';
    }
}
```

□

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:

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

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?*

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.