Matrix Multiplication & Fast Doubling Techniques in Competitive Programming

Nguyễn Quản 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.
  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*

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

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 $f(0),f(1),\ldots,f(n)$ 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, $1 \le n \le 10^5$, $1 \le k \le n$. *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.

```cpp
#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:

```cpp
#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.

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:*

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

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

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

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

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

Solution. Đặt

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

ta chứng minh

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

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.

```cpp
#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 (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.

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 $a_i \to a_{i+1}$ 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;
    }

```

```cpp
    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?*

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:

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

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 $1, 2, \ldots, n$. 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:

```
#include <bits/stdc++.h>
#pragma GCC target ("avx2")
#pragma GCC optimization ("O3")
#pragma GCC optimization ("unroll-loops")

using namespace std;
const int MOD = 1e9 + 7;

class Matrix {
public:
    vector<vector<int64_t>> v;
    void print() {
        for (int i = 0; i < v.size(); ++i) {
            for (int j : v[i]) cout << j << ' ';
            cout << '\n';
        }
    }
    Matrix operator* (Matrix m1) const {
        Matrix ans(v);
        for (int i = 0; i < m1.v.size(); ++i)
            for (int j = 0; j < m1.v.size(); ++j) ans.v[i][j] = 0;
        for (int i = 0; i < m1.v.size(); ++i)
            for (int j = 0; j < m1.v.size(); ++j)
                for (int k = 0; k < m1.v.size(); ++k) {
                    ans.v[i][j] += (v[i][k] * m1.v[k][j]) % MOD;
                    ans.v[i][j] %= MOD;
                }
        return ans;
    }
    Matrix identity (int64_t n) {
        vector<vector<int64_t>> vec(n);
        for (int i = 0; i < n; ++i) {
            vec[i].resize(n);
            for (int j = 0; j < n; ++j) vec[i][j] = (i == j);
        }
        return Matrix(vec);
    }
    Matrix operator^ (int64_t x) {
        Matrix ans = identity(v.size()), res = *this;
        while (x > 0) {
            if (x & 1) ans = res * ans;
            res = res * res;
            x /= 2;
        }
```

```
        return ans;
    }
    Matrix (vector<vector<int64_t>> v) {
        this->v = v;
    }
};

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    int n, m, k;
    cin >> n >> m >> k;
    vector<vector<int64_t>> v(n);
    for (int i = 0; i < n; ++i) v[i].assign(n, 0);
    while (m--) {
        int x, y;
        cin >> x >> y;
        --x, --y;
        ++v[x][y];
    }
    Matrix fib = Matrix(v);
    fib = fib^(k);
    cout << fib.v[0][n - 1];
}
```

2. TodomoTachi's:

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

#define MAX_SIZE 100
#define ll long long

struct Matrix {
    int m, n; // m = số hàng, n = số cột
    ll d[MAX_SIZE][MAX_SIZE];
    Matrix (int _m = 0, int _n = 0) {
        m = _m; n = _n;
        memset(d, 0, sizeof d);
    }

    Matrix operator + (const Matrix &a) const { // phép cộng ma trận
        Matrix res(m, n);
        for (int i = 0; i < m; ++i)
        for (int j = 0; j < n; ++j) {
            res.d[i][j] = d[i][j] + a.d[i][j];
            if (res.d[i][j] >= MOD) res.d[i][j] -= MOD;
        }
        return res;
    }

    Matrix operator * (const Matrix &a) const { // phép nhân ma trận
        ll x = m, y = n, z = a.n;
        Matrix res(x, z);
        for (int i = 0; i < x; ++i)
            for (int j = 0; j < y; ++j)
                for (int k = 0; k < z; ++k) res.d[i][k] = (res.d[i][k] + 1LL * d[i][j] * a.d[j][k]) % MOD;
        return res;
    }

    Matrix operator ^ (ll k) const { // phép luỹ thừa ma trận
        Matrix res(n, n);
        for (int i = 0; i < n; ++i) res.d[i][i] = 1;
        Matrix mul = *this;
```

```
        while (k > 0) {
            if (k & 1) res = res * mul;
            mul = mul * mul;
            k >>= 1;
        }
        return res;
    }
};

int main() {
    cin.tie(0) -> sync_with_stdio(0);
    int n, m, k;
    cin >> n >> m >> k;
    Matrix t(n, n);
    for (int i = 0, x, y; i < m; ++i) {
        cin >> x >> y;
        ++t.d[x - 1][y - 1]; // attention: maybe have duplicate edge
    }
    t = t ^ k;
    cout << t.d[0][n - 1];
}
```

3. Pilla Venkata Sekhar's:

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

vector<vector<ll>> mat(101, vector<ll>(101, 0));
vector<vector<ll>> mat_mul(vector<vector<ll>> mat1, vector<vector<ll>> mat2, ll sz) {
    vector<vector<ll>> mul(sz, vector<ll>(sz, 0));
    for (int i = 0; i < sz; ++i)
        for (int j = 0; j < sz; ++j) {
            int cur = 0;
            for (int k = 0; k < sz; ++k) {
                cur += (mat1[i][k] * mat2[k][j]) % MOD;
                cur %= MOD;
            }
            mul[i][j] = cur;
        }
    return mul;
}

ll mat_expo(vector<vector<ll>> pow, ll sz, ll n) {
    vector<vector<ll>> ans(sz, vector<ll>(sz, 0));
    for (int i = 0; i < sz; ++i) ans[i][i] = 1;
    while (n) {
        if (n & 1) ans = mat_mul(ans, pow, sz);
        pow = mat_mul(pow, pow, sz);
        n /= 2;
    }
    return ans[1][sz - 1];
}

int main() {
    int a, b, n, m, k;
    cin >> n >> m >> k;
    for (int i = 0; i < m; ++i) {
        cin >> a >> b;
        ++mat[a][b];
    }
    cout << mat_expo(mat, n + 1, k);
}
```

4. Dan4Life's:

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

const int MOD = 1e9 + 7;
int n, m, k, x, y;

struct Matrix {
    ll a[110][110];
    Matrix() {
        for (int i = 0; i < 110; ++i) fill(a[i], a[i] + 110, 0ll);
    }
};

Matrix M, I;

Matrix mult(Matrix x, Matrix y) {
    Matrix z;
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            for (int k = 0; k < n; ++k) z.a[i][j] += x.a[i][k] * y.a[k][j], z.a[i][j] %= MOD;
    return z;
}

Matrix pow(Matrix x, int b) {
    if (!b) return I;
    Matrix y = pow(x, b / 2);
    y = mult(y, y);
    if (b & 1) y = mult(y, x);
    return y;
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    cin >> n >> m >> k;
    for (int i = 0; i < n; ++i) I.a[i][i] = 1;
    while (m--) {
        cin >> x >> y;
        --x, --y;
        ++M.a[x][y];
    }
    cout << pow(M, k).a[0][n - 1];
}
```

□

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 $1, 2, \ldots, n$. 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:

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

```
const ll MOD = 1e9 + 7;
vector<vector<ll>> mat(101, vector<ll>(101, 0));

vector<vector<ll>> mat_mul(vector<vector<ll>> mat1, vector<vector<ll>> mat2, ll sz) {
    vector<vector<ll>> mul(sz, vector<ll>(sz, 0));
    for (int i = 0; i < sz; ++i)
        for (int j = 0; j < sz; ++j) {
            ll cur = 0;
            for (int k = 0; k < sz; ++k)
                if (mat1[i][k] > 0 && mat2[k][j] > 0) {
                    if (cur) cur = min(cur, (mat1[i][k] + mat2[k][j]));
                    else cur = mat1[i][k] + mat2[k][j];
                }
            mul[i][j] = cur;
        }
    return mul;
}

ll mat_expo(vector<vector<ll>> pow, ll sz, ll n) {
    vector<vector<ll>> ans(sz, vector<ll>(sz, 0));
    int check = 0;
    while (n) {
        if (n & 1) {
            if (check) ans = mat_mul(ans, pow, sz);
            else {
                check = 1;
                for (int i = 0; i < sz; ++i)
                    for (int j = 0; j < sz; ++j) ans[i][j] = pow[i][j];
            }
        }
        pow = mat_mul(pow, pow, sz);
        n >>= 1;
    }
    if (ans[1][sz - 1]) return ans[1][sz - 1];
    return -1;
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    int n, m, k;
    cin >> n >> m >> k;
    ll a, b, c;
    for (int i = 0; i < m; ++i) {
        cin >> a >> b >> c;
        if (!mat[a][b]) mat[a][b] = c;
        else mat[a][b] = min(mat[a][b], c);
    }
    cout << mat_expo(mat, n + 1, k);
}
```

2. Dan4Life's:

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

const int MOD = 1e9 + 7;
const ll LINF = 4e18;
int n, m, k, x, y, z;

struct Matrix {
    ll a[110][110];
    Matrix (ll v = 0) {
```

```
        for (int i = 0; i < 110; ++i) fill(a[i], a[i] + 110, v);
    }
};

Matrix M(LINF), I(LINF);

Matrix mult(Matrix x, Matrix y) {
    Matrix z(LINF);
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            for (int k = 0; k < n; ++k) z.a[i][j] = min(z.a[i][j], x.a[i][k] + y.a[k][j]);
    return z;
}

Matrix pow(Matrix x, int b) {
    if (!b) return I;
    if (b == 1) return x;
    Matrix y = pow(x, b / 2);
    y = mult(y, y);
    if (b & 1) y = mult(y, x);
    return y;
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    cin >> n >> m >> k;
    while (m--) {
        cin >> x >> y >> z;
        --x, --y;
        M.a[x][y] = min(M.a[x][y], (ll)z);
    }
    cout << (pow(M, k).a[0][n - 1] < LINF ? pow(M, k).a[0][n - 1] : -1);
}
```

3. Bùi Trung Hiếu's:

```
#include <bits/stdc++.h>
#define FOR(i, a, b) for (int i = (a), _b = (b); i <= _b; ++i)
#define FORD(i, b, a) for (int i = (b), _a = (a); i >= _a; --i)
#define REP(i, n) for (int i = 0, _n = (n); i < _n; ++i)
#define FORE(i, v) for (__typeof((v).begin()) i = (v).begin(); i != (v).end(); ++i)
using namespace std;
using ll = long long;

const int MAXN = 106;
const ll INF = 4e18;
int n, m, k;

struct Matrix {
    int n, m;
    ll d[MAXN][MAXN];

    Matrix (int _n, int _m) {
        n = _n, m = _m;
        REP(i, MAXN) REP(j, MAXN) d[i][j] = INF;
    }

    Matrix operator * (const Matrix &a) const {
        int x = n, y = m, z = a.m;
        Matrix res(x, z);
        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]);
        return res;
    }

```

```
    Matrix operator ^ (int k) const {
        Matrix res(n, n), mul = *this;
        REP(i, n) res.d[i][i] = 0;
        while (k > 0) {
            if (k & 1) res = res * mul;
            mul = mul * mul;
            k >>= 1;
        }
        return res;
    }
};

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(NULL);
    cout.tie(NULL);
    cin >> n >> m >> k;
    Matrix trans(n, n);
    FOR(i, 1, m) {
        int x, y, w;
        cin >> x >> y >> w;
        --x, --y;
        trans.d[x][y] = min(trans.d[x][y], (ll)w);
    }
    trans = trans ^ k;
    if (trans.d[0][n - 1] < INF) cout << trans.d[0][n - 1];
    else cout << -1;
}
```

4. Viktor Maksimoski's:

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

struct Mat {
    int n, m;
    vector<vector<ll>> mat;

    Mat (int _n, int _m) {
        n = _n, m = _m;
        mat.resize(n, vector<ll>(m));
    }

    Mat(vector<vector<ll>> v) {
        mat = v;
        n = (int)v.size(), m = (int)v[0].size();
    }
};

Mat ID (int n) {
    Mat ans(n, n);
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j) ans.mat[i][j] = 2e18;
    return ans;
}

Mat mul(Mat a, Mat b) {
    Mat ans = ID(a.n);
    for (int i = 0; i < a.n; ++i)
        for (int j = 0; j < b.m; ++j)
            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]);
    return ans;
}

```

```
Mat exp(Mat a, ll b) {
    Mat ans = ID(a.n);
    for (int i = 0; i < a.n; ++i) ans.mat[i][i] = 0;
    while (b) {
        if (b & 1) ans = mul(ans, a);
        a = mul(a, a);
        b >>= 1;
    }
    return ans;
}

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(NULL);
    int n, m, k;
    cin >> n >> m >> k;
    Mat a = ID(n);
    while (m--) {
        int x, y, z;
        cin >> x >> y >> z;
        a.mat[x - 1][y - 1] = min(a.mat[x - 1][y - 1], (ll)z);
    }
    a = exp(a, k);
    cout << (a.mat[0][n - 1] > 1e18 ? -1 : a.mat[0][n - 1]) << '\n';
}
```

5. Olympia's:

```
#include <bits/stdc++.h>
#pragma GCC target ("avx2")
#pragma GCC optimization ("O3")
#pragma GCC optimization ("unroll-loops")
using namespace std;
const int MOD = 1e9 + 7;

int64_t chmin(int64_t x, int64_t y) {
    if (x == -1) return y;
    if (y == -1) return x;
    return min(x, y);
}

class Matrix {
    public:
    vector<vector<int64_t>> v;

    void print() {
        for (int i = 0; i < v.size(); ++i) {
            for (int j : v[i]) cout << j << ' ';
            cout << '\n';
        }
    }

    Matrix operator * (Matrix m1) const {
        Matrix ans(v);
        for (int i = 0; i < m1.v.size(); ++i)
            for (int j = 0; j < m1.v.size(); ++j) ans.v[i][j] = -1;
        for (int i = 0; i < m1.v.size(); ++i)
            for (int j = 0; j < m1.v.size(); ++j) {
                ans.v[i][j] = -1;
                for (int k = 0; k < m1.v.size(); ++k) {
                    if (v[i][k] == -1 || m1.v[k][j] == -1) continue;
                    ans.v[i][j] = chmin(v[i][k] + m1.v[k][j], ans.v[i][j]);
                }
            }
        return ans;
```

```
    }

    Matrix identity (int64_t n) {
        vector<vector<int64_t>> vec(n);
        for (int i = 0; i < n; ++i) {
            vec[i].resize(n);
            for (int j = 0; j < n; ++j) vec[i][j] = (i == j);
        }
        return Matrix(vec);
    }

    Matrix operator^ (int64_t x) {
        Matrix ans = *this, res = *this;
        while (x > 0) {
            if (x & 1) ans = res * ans;
            res = res * res;
            x >>= 1;
        }
        return ans;
    }

    Matrix (vector<vector<int64_t>> v) {
        this->v = v;
    }
};

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    int n, m, k;
    cin >> n >> m >> k;
    vector<vector<int64_t>> v(n);
    for (int i = 0; i < n; ++i) v[i].assign(n, -1);
    while (m--) {
        int x, y, w;
        cin >> x >> y >> w;
        --x, --y;
        v[x][y] = chmin(v[x][y], w);
    }
    Matrix fib = Matrix(v);
    fib = fib ^ (k - 1);
    cout << fib.v[0][n - 1];
}
```

□

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.