Matrix Exponentiation and Multiplication

Matrix Multiplication

```
vector<vector<int>> transpose(vector<vector<int>>& A){
    int n = A.size();
    vector<vector<int>> ans(n, vector<int>(n, 0));
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            ans[i][j] = A[j][i];
        }
    }
    return ans;
}

vector<vector<int>> matrixMultiply(vector<vector<int>>& A, vector<vector<int>>& B1){
    int n = A.size();
    vector<vector<int>> B = transpose(B1);
    vector<vector<int>> M(n, vector<int>(n, 0));
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            int ans = 0;
            for(int k = 0; k < n; k++){
                ans = (ans + (A[i][k] * B[j][k]) % mod) % mod;
            }
            M[i][j] = ans;
        }
    }
    return M;
}

vector<vector<int>> squareMultiply(vector<vector<int>>& A){
    int n = A.size();
    vector<vector<int>> B(n, vector<int>(n, 0));
    vector<vector<int>> ans = B;
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            B[i][j] = A[j][i];
        }
    }
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            int temp = 0;
            for(int k = 0; k < n; k++){
                temp = (temp + (A[i][k] * B[j][k]) % mod) % mod;
            }
            ans[i][j] = temp;
        }
    }
    return ans;
}

vector<vector<int>> matrixExponentiation(vector<vector<int>>& M, int pow){
    int n = M.size();
    vector<vector<int>> next(n, vector<int>(n, 0));
    for(int i = 0; i < n; i++) next[i][i] = 1;

    while(pow > 0){
        if(pow & 1)
            next = matrixMultiply(next, M);
        M = squareMultiply(M);
        pow >>= 1;
    }
    return next;
}
```

Number of Paths with K Edges in a Graph

```cpp
#include<bits/stdc++.h>
using namespace std;
#define int unsigned long long
int INF = 1e19;
int mod = 1e18+7;

vector<vector<int>> matrixMultiply(vector<vector<int>>& A, vector<vector<int>>& B){
    int n = A.size();
    int c = A[0].size();
    int m = B[0].size();
    vector<vector<int>> M(n, vector<int>(m, INF));
    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            for(int k = 0; k < c; k++){
                if(A[i][k] == INF || B[k][j] == INF) continue;
                int ans = (A[i][k] + B[k][j]);
                M[i][j] = min(M[i][j], ans);
            }
        }
    }
    return M;
}

vector<vector<int>> matrixExponentiation(vector<vector<int>>& M, int pow){
    int n = M.size();
    vector<vector<int>> next(n, vector<int>(n, INF));
    for(int i = 0; i < n; i++) next[i][i] = 0;

    while(pow > 0){
        if(pow & 1)
            next = matrixMultiply(next, M);
        M = matrixMultiply(M, M);
        pow >>= 1;
    }
    return next;
}

void solve(){
    int n, m, k;
    cin >> n >> m >> k;
    vector<vector<int>> graph(n, vector<int>(n, INF));
    for(int i = 0; i < m; i++){
        int start, end, weight;
        cin >> start >> end >> weight;
        graph[start-1][end-1] = min(weight, graph[start-1][end-1]);
    }
    graph = matrixExponentiation(graph, k);
    if(graph[0][n-1] == INF) cout << "-1" << endl;
    else cout << graph[0][n-1] << endl;
}

int32_t main(){
    int tt = 1;
    while(tt--){
        solve();
    }
}
```