矩阵

Matrix

```cpp
struct Matrix{
    LL ma[N][N];
    int r, c;

    Matrix(){ r = c = 0; }
    Matrix(int rr, int cc){
        r = rr, c = cc;
        for(int i = 1; i <= r; i++)
            for(int j = 1; j <= c; j++)
                ma[i][j] = 0;
    }
    void unit(int n){
        r = c = n;
        for(int i = 1; i <= n; i++)
            for(int j = 1; j <= n; j++)
                ma[i][j] = i == j;
    }
    Matrix operator + (const Matrix &A){
        Matrix res(r, c);
        for(int i = 1; i <= r; i++)
            for(int j = 1; j <= c; j++)
                res.ma[i][j] = (ma[i][j] + A.ma[i][j]) % MOD;
        return res;
    }
    Matrix operator * (const Matrix &A){
        Matrix res(r, A.c);
        for(int i = 1; i <= res.r; i++)
            for(int j = 1; j <= res.c; j++)
                for(int k = 1; k <= c; k++)
                    (res.ma[i][j] += ma[i][k] * A.ma[k][j]) %= MOD;
        return res;
    }
    void print(){
        for(int i = 1; i <= r; i++){
            for(int j = 1; j <= c; j++)
                printf("%lld ", ma[i][j]);
            puts("");
        }
    }
};

Matrix fpow(Matrix bs, LL idx){
    Matrix res;
    res.unit(bs.r);
    while(idx){
        if(idx & 1) res = res * bs;
        idx >>= 1;
        bs = bs * bs;
    }
    return res;
}
LL fpow(LL bs, LL idx){
    LL res = 1;
    bs %= MOD;
    while(idx){
        if(idx & 1) (res *= bs) %= MOD;
        (bs *= bs) %= MOD;
        idx >>= 1;
    }
    return res;
}

bool getInverse(Matrix &A){ // return false: no inverse; true: A is the inverse
    int n = A.r;
    Matrix res; res.unit(n);
    for(int j = 1; j <= n; j++){
        int r = j;
        for(int i = j + 1; i <= n; i++)
            if(A.ma[i][j] > A.ma[j][j])
                r = i;
        if(r != j)  swap(A.ma[r], A.ma[j]), swap(res.ma[r], res.ma[j]);
        if(A.ma[j][j] == 0) return false;
        for(int i = 1; i <= n; i++){
            if(i == j)  continue;
            LL div = A.ma[i][j] * fpow(A.ma[j][j], MOD-2) % MOD;
            for(int k = 1; k <= n; k++){
                A.ma[i][k] -= div * A.ma[j][k] % MOD;
                ((A.ma[i][k] %= MOD) += MOD) %= MOD;
                res.ma[i][k] -= div * res.ma[j][k] % MOD;
                ((res.ma[i][k] %= MOD) += MOD) %= MOD;
            }
        }
    }
    for(int i = 1; i <= n; i++){
        LL inv = fpow(A.ma[i][i], MOD-2);
        for(int j = 1; j <= n; j++)
            (res.ma[i][j] *= inv) %= MOD;
    }
    A = res;
    return true;
}
```