# Mystery Matrix

## Matrix for Recursive Equations and Logarithmic Dynamic programming Solution

Mukuldeep Maiti

# Mystery Matrix

**Matrix for Recursive Equations and**

**Logarithmic Dynamic programming Solution**

**Mukuldeep Maiti**

March 11, 2022

# Message from Author

**I stumbled over something while exploring a part of Algorithms and data structures. I digged it deep, understood how that has been amazingly working, then found more possibilities and now you know I am sharing the same with you.**

Mukuldeep Maiti

# Contents

# 1 Introduction

**Matrix, people in black sunglasses wearing full black attire, hahah! just kidding. I'm talking about the rectangular grid filled with numbers. Almost all of us familiar with matrix and related mathematics, but very few are actually aware of real life interpretations. Most of the usages are hidden behind the hood. In this long article I'll talk about one of them. How matrics can solve recursive equation through matrix multiplications. which in turn can be used in various field. In computer science, this methodology can be used to solve few dynamic programming problems in logarithmic time complexity. In competitive programming, you will find one in thousand problems that can be solved using this technique effectively.**

**This method is not new, has already been used to solve problems, one of the popular problem among them is finding $n^{th}$ fibonacci number using matrix operation. If you don't know this, don't panic! I added this example too, how can I forgot to mention such a popular example?**

**I'll try to touch the surface of the concepts required, like matrix, binary exponentiation, recursive equation. If you wanna understand them from ground level, let's meet over a coffee, If you already know these, you can directly jump to chapter 6.**

# 2 Pre-requisite

You should have an insight of these concepts before reading this long article to understand everything mentioned in the next chapters.

- Basic concept of matrix, dimension, elements, row, column etc.
- Basic operation on matrix, like matrix multiplication, addition , substraction etc.
- Programming Language, C++ to understand written snippets in the examples (optional)

- Complexity of a program: time complexity, space complexity
- Basic concept of dynamic programming
- Concept of Exponentiation is a big plus, although mentioned in the chapter 4

# 3 Matrix Multiplication

**If you don't know about Matrix, Matrix is a rectangular grid filled with data, especially numbers, they are written in different form For example**

| | | | |
|---|---|---|---|
| 8 | 3 | 6 | 1 |
| 6 | 4 | 8 | 3 |
| 5 | 7 | 2 | 9 |

or

$$\begin{bmatrix} 8 & 3 & 6 & 1 \\ 6 & 4 & 8 & 3 \\ 5 & 7 & 2 & 9 \end{bmatrix}$$

**Dimensions of a matrix is written as (number of row x number of column). Here, number of rows is 3 and number of column is 4, So, the dimention of this matrix is (3x4).**

**Matrix multiplication is multiplication of two matrices. The necessary condition to be able to multiply two matrices is number of column of the first matrix must be equal to the number of rows of the second matrix. Let A be a matrix of dimensions (2x3) and B be a matrix of dimensions (3x4), then AxB can be calculated, as number of columns in A = number of rows in B = 3. The dimensions of the resultant matrix, AxB will be (2x4) i.e. number of row of the first matrix x number of columns of the second matrix.**

**However, BxA cannot be calculated, as the number of columns in B is not equals to the number of rows in A.**

**Each element of the resulting matrix is calculated as follows.**

1. pick the whole row from the first matrix (the row number is the row number of the element you are calculating for the resulting matrix )
2. pick the whole column from the second matrix (the column number is the column number of the element you are calculating for the resulting matrix )
3. multiply corresponding elements from these two array
4. add them togather to get the current element of the resulting matrix

**For example, let's take two arbitary matrix**

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

$$B = \begin{bmatrix} 7 & 8 \\ 9 & 10 \end{bmatrix}$$

**Dimentions are A (3x2), B (2x2). AxB can be calculated. Resulting matrix will be of dimension, 3x2.**

$$AxB = C = \begin{bmatrix} c11 & c12 \\ c21 & c22 \\ c31 & c32 \end{bmatrix}$$

**For c11, pick row 1 from A . i.e.**

$$\begin{bmatrix} 1 & 2 \end{bmatrix}$$

**pick column 1 from B. i.e.**

$$\begin{bmatrix} 7 \\ 9 \end{bmatrix}$$

**multiply corresponding elements and aff them togather. i.e. 1x7 + 2x9=25. c11=25. Similarly calculate for all the elements in resulting matrix. c12=1x8 + 2x10 = 28, c21=3x7 + 4x9=57, c22=3x8**

**+ 4x10 =64, c31=5x7 + 6x9=89, c32=5x8 + 6x10=100. The resulting matrix is**

$$C = \begin{bmatrix} 25 & 28 \\ 57 & 64 \\ 89 & 100 \end{bmatrix}$$

## 3.1 Implementation in C++

```
/*
 * Created by mukul on 16-02-2022.
 */
#include <bits/stdc++.h>
#define ll long long
#define t9p7 1000000007
using namespace std;

/*
 * Multiply two matrices
 * parameters: 2 matrices as 2D vectors
 * returns: Resultant matrix
 */
template <typename T>
vector<vector<T>>
matrix_multiplication(vector<vector<T>> mat1,vector<vector<T>> mat2){
    //dimensions: mat1  (m1_row x m1_col)
    //            mat2  (m2_row x m2_col)
    ll m1_row=mat1.size();      // no of rows in first matrix i.e. mat1
    ll m1_col=mat1[0].size();   // no of columns in first matrix i.e. mat1
    ll m2_row=mat2.size();      // no of rows in second matrix i.e. mat2
    ll m2_col=mat2[0].size();   // no of columns in second matrix i.e. mat2

    //initializing resulting matrix of dimension: (m1_row x m2_col)
    vector<vector<T>> ans(m1_row,vector<T>(m2_col,0));
    if(m1_col!=m2_row){//checking if multiplication is possible
        cout<<"Can not perform multiplication: mismatches in dimensions"<<endl;
    }else{
        //iterating over rows of resulting matrx
        for (ll i = 0; i < m1_row; i++)
        {
            //iterating over rows of resulting matrix
```

```
            for (ll j = 0; j < m2_col; j++)
            {
                //generating single elemets of the resulting matrix ans[i][j]
                for (ll x = 0; x < mat1[0].size() ; x++)
                {
                    ans[i][j]+= (mat1[i][x] * mat2[x][j])%t9p7;
                    ans[i][j]%=t9p7;
                }
            }
        }
    }
    return ans;//returning resulting matrix
}

//driver function
int main(){
  vector<vector<ll>> mat1={
            {1,2},
            {3,4},
            {5,6}
    };
    vector<vector<ll>> mat2={
            {7,8},
            {9,10},
    };

    vector<vector<ll>> result=matrix_multiplication(mat1,mat2);

    cout<<"resultant matrix is"<<endl;
    for(auto row:result){
        for(auto element:row){
            cout<<element<<" ";
        }cout<<endl;
    }
    return 0;
}
```

## Output:

```
resultant matrix is
25 28
57 64
89 100
```

Figure 1: output of the above c++ snippet

# 4 Binary Exponentiation

**Binary exponentiation is a trick that allows us to calculate $a^n$ using $O(logn)$ multiplications instead of $O(n)$ as by brute force approach. This is accomplished by decomposing the exponent into binary representation and calculating power from previously calculated results.**

**For example, you wanna calculate $5^{12}$. In binary representation, 12 can be written as 1100, i.e. $(12)_{10} = (1100)_2$**

**Now you can decompose the exponent, 12 to sum of powers of 2, i.e. 8+4**

**Now, $5^{12} = 5^8.5^4$ $5^8 and 5^4$ can be calculated as follows**

$p1 = 5^1 = 5$ **(for exponent 1)**

$p2 = p1 * p1 = 25$ **(for exponent 2)**

$p4 = p2 * p2 = 625$ **(for exponent 4)**

$p8 = p4 * p4 = 390625$ **(for exponent 8)**

**To calculate $5^{12}$, you only have to multiply $5^8$ and $5^4$ togather. That can be evaluated along with the above calculations. Exponentiation**

**is valid and works perfectly fine under modulus opetrations.**

## 4.1 why exponentiation?

**Well, Let you wanna calculate $x^{2^{500}}$, obviously under modulus. The exponent part is huge, if you try to multiply x $2^{500}$ times, probably that will take few billions of years to complete by classical computers, but with binary exponentiation, exponent part $2^{500}$ can be represented in atmost 501 bits ($=log_2(2^{500})+1$), will perform atmost 2*501 unit operations. unit operations are consist of multiplication and/or modulus. So, the calculation can be solved in few milliseconds in classical computer.**

**We use exponentiation to reduce the complexities logarithmically. Binary exponentiation is not limited to numbers, can be used in case of matrix as well. Let that be the part of next chapters. Now see the implementation(s)**

## 4.2 Implementation in C++

```
/*
 * Created by mukul on 17-02-2022.
 */
#include <bits/stdc++.h>
#define ll long long
#define t9p7 1000000007
using namespace std;

/*
 * power by binary exponentiaion
```

```
 * parameters: base, exponent, mod(default value 10^9+7)
 * returns: power a^b mod p
 */
ll power_by_binary_exponentiation(ll a,ll b, ll p=t9p7){
    ll result=1;
    a%=p;//modulo the base
    if(b==0)return 1; //in case exponent is 0
    if(a==0)return 0; //in case base is 0
    while(b){
        //if current lowest significant bit is set
        if(b&1)
            //considering current power of 2 worth multiply
            result=(result*a)%p;
        b>>=1;//right shifting b, same as inter division by 2
        //squaring a to get a^(next power of 2)
        a=(a*a)%p;
    }
    return result;//returning result
}

int main(){
    cout<<power_by_binary_exponentiation(1233473443423,78687921793871283);
    return 0;
}
```

**Output: 53423659**

# 5 Brief on Recursive Equations

**Recursive equations are the equation that represents an infinite series, where one element depends on one or more previous element(s) and begins with particular starting element(s). For example, Fibonacci numbers, $n^{th}$ fibonacci number depends on $(n-1)^{th}$**

**and $(n-2)^{th}$ fibonacci numbers. and starting elements are 0,1...**

**Recursive Equation of fibonacci series is written as $f_n = f_{n-1} + f_{n-2}$, where $f_x$ is $x^{th}$ element.**
**the series looks like 0,1,1,2,3,5,8,13,21,34,55 and so on. Similarly few other simple recursive equations are**

**For positive integers, eqn. is $Z_n = Z_{n-1} + 1$ and starts with 1...**
**the series looks like 1,2,3,4,5,6,7,8,9,10.. and so on**

**For sum of last three elements, eqn. is $S_n = S_{n-1} + S_{n-2} + S_{n-3}$ and you can choose starting vector (atleast three elements required as the equation depends on previous three elements).**
**If you choose 2 0 9 as starting elements then the series looks like 2,0,9,11,20,40,71.. and so on**

# 6 $(Matrix)^p$ by Exponentiation

**Similar to the calculation of power of a number, power of a matrix can be calculated by binary exponentiation, iff the matrix is a square matrix. why this condition? Hint. Some what linked to necessary condition for multiplication of two matrices, ohh! you remembered! great dude. In case of matrix, the multiplication will be matrix multiplication. The concept is same, decompose the exponent in binary representation, muliply only those power**

**of 2 of the matrix to the result, whose bit is set in the binary representation.**
**Worried about modulo operation in matrix? it's natural. even I had that question in my mind. do modulo to the arithmatic operations inside the matrix multiplication.**

**Let A be a matrix and R gonna be Resultant matrix, you wanna calculate R,** $R = A^{11}$**. In binary representation, 11 can be written as 1011, i.e.** $(11)_{10} = (1011)_2$

**Now you can decompose the exponent, 11 to sum of powers of 2, i.e. 8+2+1**
**Now,** $A^{11} = A^8.A^2.A^1$ $A^8, A^2, A^1$ **can be calculated as follows**
$a1 = A^1 = A$ **(for exponent 1)**
$a2 = a1 * a1 = A^2$ **(for exponent 2)**
$a4 = a2 * a2 = A^4$ **(for exponent 4)**
$a8 = a4 * a4 = A^8$ **(for exponent 8)**

**To calculate** $A^{11}$**, you only have to multiply** $A^8, A^2, A^1$ **togather. That can be evaluated along with the above calculations, as follows:**
**Set R as unit matrix of the same dimensions as A**
$a1 = A^1 = A$ **multiply a1 with R, i.e. R=a1*R, as** $0^{th}$ **LS bit is set**
$a2 = a1 * a1 = A^2$ **multiply a2 with R, i.e. R=a2*R, as** $1^{st}$ **LS bit is set**
$a4 = a2 * a2 = A^4$ **do nothing, as** $2^{nd}$ **LS bit is not set**

$a8 = a4 * a4 = A^8$ **multiply a8 with R, i.e. R=a8*R, as** $3^{rd}$ **LS bit is set**

**Congratulations! Now you have** $A^{11}$ **in R.**

### 6.1 Implementation in C++

```
V(V(T)) matrix_binary_exponentiation(ll n){
        V(V(T)) res = mat,a=mat;
        n--;

        while (n > 0) {
            if (n & 1)
                res =matrix_multiplication(res,a);
            a=matrix_multiplication(a,a);
            n >>= 1;
        }
        return res;
    }
```

**Where matrix multiplication() multiplies two matrices. ll is long long and V(V(T)) is vector of vector or you can assume a 2D array**

## 7 Fitting Recursive Equation in matrix

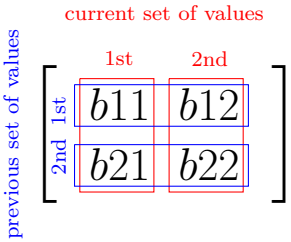
**Fitting recursive equation in matrix is the primary focus of this long article, and probably this is the most important chapter of all. Here I'll discuss how a Recursive Equation can be expressed as matrices.**

**You need two matrices, One is start matrix or start vector and another is base matrix.**

**Start matrix or start vector is a 1D matrix / array, will contain the starting elements of the infinite series. Length of this vector depends on how many previous elements are required in the recursive equation. For example, in fibonacci series, $n^{th}$ element depends on previous two elements, in this case length of the start vector will be atleast two. we can use [0,1] as start vector. Another example, in case of sum of last three as current element, we need last three elements, in that case, the length of the start vector must be atleast 3.**

**Base matrx is a square matrix, will carry recursive information of the recursive equation. Dimension of this matrix will be (length of start vector X length of start vector), contents of this matrix be as follows:**

**Each column represents current part of the series and each row represents previous part of the series.**



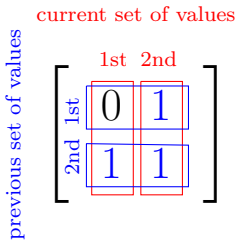
**If $b11 = x$ then x times of 1st element of previous values will be added to 1st element of the current values**
**If $b12 = y$ then y times of 1st element of previous values will be added to 2nd element of the current values**
**If $b21 = z$ then z times of 2nd element of previous values will be added to 1st element of the current values**

**Lets understand this with the help of examples. In fibonacci series $0, 1, 1, 2, 3, 5, 8, 13, 21....$ starting vector will be $[0, 1]$, is actually first set of values from the series. The next set of values will be $[1, 1]$ - simply by sliding the window 1 step towards right. Similarly, next sets be $[1, 2]$ , $[2, 3]$ , $[3, 5]$ , $[5, 8]$ and so on.**

$$\left[\begin{array}{cccccccccc} \boxed{\text{1 - st}} & & \boxed{\text{3 - rd}} & & \boxed{\text{So on .....}} & & & & & \\ 0 & 1 & 1 & 2 & 3 & 5 & 8 & 13 & 21 & \ldots \\ & \boxed{\text{2 - nd}} & & \boxed{\text{4 - th}} & & & & & & \end{array}\right]$$

**From the above sliding windows, you can conclude that,**
**current 1st element = previous second element , so, $b21 = 1$**
**current 2nd element = previous 1st element + previous 2nd element, so, $b12 = 1$ and $b22 = 1$ , therefore, base matrix is**

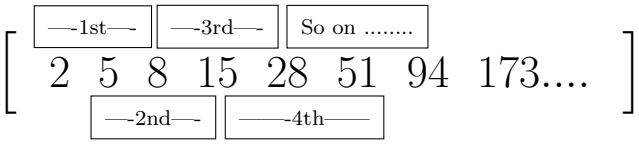
$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$




## 7.1 More examples

### 7.1.1 Sum of last three elements

**Lets choose starting three elements randomly,** $[2, 5, 8]$**. The infinite series is** $[2, 5, 8, 15, 28, 51, 94, 173...]$




**From the above sliding windows, you can conclude that,**

**current 1st element = previous 2nd element , so,** $b21 = 1$

**current 2nd element = previous 3nd element , so,** $b32 = 1$

**current 3nd element = previous 1st element + previous 2nd element + previous 3rd element, so,** $b13 = 1$ **,** $b23 = 1$ **and** $b33 = 1$ **, therefore, base matrix is**

$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

### 7.1.2 Specific recursive equation $f_n = 2 * f_{n-1} + 5 * f_{n-2}$

**$f_n$ depends on only two previous elements, so length of the starting vector is atleast 2. Lets choose starting two elements randomly,** $[1, 2]$**. The infinite series is** $[1, 2, 12, 64, 334, 1798, ...]$
**From the above recursive equation, you can conclude that,**
**current 1st element = previous 2nd element , so,** $b21 = 1$
**current 2nd element =2 x previous 1st element +5 x previous 2nd element , so,** $b12 = 2$ **,** $b22 = 5$ **, therefore, base matrix is**

$$\begin{bmatrix} 0 & 2 \\ 1 & 5 \end{bmatrix}$$

### 7.1.3 Denomination change problem

**Problem: find no of ways , Rs. n change can be made with infinite number of 1,3,5 denominations, order matters.**
**Solution: This is a typical dynamic programming problem. where we calculate no of ways for Change ranging from 0 to n, keeping track of last 5 only. Where** $f_n$ **is calculated by adding** $f_{n-1}$**,** $f_{n-3}$ **and** $f_{n-5}$**.** $f_n$ **depends on only previous five elements, so length of the starting vector is atleast 5. Starting five elements be,** $[1, 1, 2, 3, 5]$**.**
**The infinite series is** $f_n = f_{n-1}+f_{n-3}+f_{n-5}$**, generates** $[1, 1, 2, 3, 5, 8, 12, 19, 30....]$
**From the above recursive equation, you can conclude that,**
**current 1st element = previous 2nd element , so,** $b21 = 1$

**current 2nd element = previous 3rd element , so,** $b32 = 1$

**current 3rd element = previous 4th element , so,** $b43 = 1$

**current 4th element = previous 5th element , so,** $b54 = 1$

**current 5th element =previous 1st element(n-5) +previous 3rd element(n-3)+previous 5th element (n-1), so,** $b15 = 1$ , $b35 = 1$ **and** $b55 = 1$ **, therefore, the base matrix is**

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

## 7.2 Exercise

**Q1. Find base matrix for the recursive equation** $f_n = f_{n-2} + 7 * f_{n-4}$

**Q2. Find base matrix for the recursive equation** $f_n = 42 * f_{n-1} - f_{n-3} + 7 * f_{n-4}$

**Q3. Find starting vector and base matrix for the number of ways an integer n can be formed by throwing dice one or more time.**

**Q4. Find recursive equation for the following base matrix**

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 4 \\ 1 & 0 & 0 & 0 & -5 \\ 0 & 1 & 0 & 0 & 7 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0.5 \end{bmatrix}$$

**Q5. Find first 9 elements of the above matrix, if starting vector is** $[1, 1, 1, 1, 1]$**. Consider one digit in mantissa(fractional part)**

# 8 Solving problems

**Now, you know how to give a shape to a recursive equation using base matrix and starting vector. Next step is to calculate** $n^{th}$ **element of the infinite series. Which is accomplished by muliplying** $StartingVector$ **with** $(BaseMatrix)^{exponent}$**. This** $(BaseMatrix)^{exponent}$ **part is calculated by using matrix binary exponentiation in logarithmic complexity.**

**we usually use last element of the result of the above multiplication as the answer. If the starting vector contains first t elements then for exponent=0, the result be same as starting vector. Last element is actually** $x^{th}$ **element of the series. similarly,**

**exponent=1, provide us** $(x+1)^{th}$ **element**

**exponent=2, provide us** $(x+2)^{th}$ **element**

**exponent=3, provide us** $(x+3)^{th}$ **element**

**exponent=4, provide us $(x+4)^{th}$ element**

**...**

**exponent=n, provides us $(n+x)^{th}$ element**

**In other words, for $n^{th}$ element, exponent =n-x**

**If exponent is less than or equals to x, you can simple took the data from starting vector.**

# 9 Examples

## 9.1 fibonacci number

### 9.1.1 problem

**Find $n^{th}$ fibonacci number under modulo, where n is a large number in the range of $2^{1000000}$ . i.e. $f(n) mod P$ where, $1 <= n <= 2^{1000000}$ and $P = 10^9 + 7$**

### 9.1.2 solution

**The problem can be solved by exponentiating the base matrix, otherwise the time complexity for calculating the same using loop be huge, I mean a long long while. $2^{999980}$ seconds of classical computer. To solve the problem with matrix exponentation**

**First choose start vector, which is fixed for fibonacci series, i,e, [0,1]. Size of start vector = 2 is choosen as current element only depends on last two elements.**

**Then, base matrix, In chapter 7, I already explained how base**

**matrix for fibonacci series is calculated. which is**

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$

**if exponent is less than equal to size of the starting vector, i.e. 2, then choose the answer from starting vertex itself. Otherwise, calculate effective exponent, exponent = n-x as descibed in the chapter 8.**

**Now, Calculate** $BaseMatrix^{exponent}$ **using binary ecponenetiation, as described in chapter 6.**

**Multiply base matrix with start vector** $[StartVector]$ **x** $[BaseMatrix]$

**Last element of the resulting vector is required** $n^{th}$ **element of the series.**

### 9.1.3 complexity comparison

$O(n^3)$ **for multiplying matrix, here n=2, O(8)=O(1)**

$O(logn)$ **for exponentiation, where n is the exponent. Here,** $O(log_2 2^{1000000}) = O(1000000)$

**So, the overall complexity be** $O(log(exponent))$

### 9.1.4 implementation (in c++)

```cpp
/*
 * Created by mukul on 02-03-2022.
 */
 #include <bits/stdc++.h>
//io
#define O cout<<
#define I cin>>
#define El endl;

//ds
#define ll long long
#define V(x) vector<x>

//constants
#define t9p7 1000000007

using namespace std;

template <typename T>
class matrix{
public:
    ll row,col;
    V(V(T)) mat;

    matrix(ll row,ll col,V(V(T)) mat){
        this->row=row;
        this->col=col;
        this->mat=mat;
    }

     //matrix multiplication
    V(V(T)) matrix_multiplication(V(V(T)) mat1,V(V(T)) mat2){
        V(V(T)) ans(mat1.size(),V(T)(mat2[0].size()));
        if(mat1[0].size()!=mat2.size()){
            O "error: mismatches in dimensions of the matrices"<<El
        }else{
            for (ll i = 0; i < mat1.size(); i++)
            {
                for (ll j = 0; j < mat1[0].size(); j++)
```

```
                {
                    ans[i][j] = 0;
                    for (ll x = 0; x < mat1[0].size() ; x++)
                    {
                        ans[i][j]+= (mat1[i][x] * mat2[x][j])%t9p7;
                        ans[i][j]%=t9p7;
                    }
                }
            }
        }
        return ans;
    }

    // matrix binary exponentiation
    V(V(T)) matrix_binary_exponentiation(ll n){
        V(V(T)) res = mat,a=mat;
        n--;

        while (n > 0) {
            if (n & 1)
                res =matrix_multiplication(res,a);
            a=matrix_multiplication(a,a);
            n >>= 1;
        }
        return res;
    }
};


void main(){
    ll n;
    I n;//input n ( to find n-th fibonacci number)
    V(V(ll)) arr={ //base matrix
      //curr 1 2
            {0,1},//prev 1st
            {1,1},//prev 2nd
    };
    V(V(ll)) ini_mat={ //starting vector
            {0,1}
```

```
    };
    if(n<=2){//if n less than size of
        O ini_mat[0][n-1]<<El
        return;
    }

    //effective exponent
    ll n_pow=n-2;

    //creating instance of matrix class with base matrix
    matrix<ll> mat(2,2,arr);

    //binary exponentiation of base matrix
    V(V(ll)) res = mat.matrix_binary_exponentiation(n_pow);

    //multiplying the above result with start vector
    V(V(ll)) res2 = mat.matrix_multiplication(ini_mat, res);

    O res2[0][1]<<El

    return 0;
}
```

**Input: 31626731231212312 Output: 641522053**

**i.e.** $fib_{31626731231212312} \; mod \; 1000000007 \; = \; 641522053$

## 9.2 Sum of Last three

### 9.2.1 problem

**Find $n^{th}$ number of the infinite series whose each element is sum of previous three elements, under modulo, where n is a large number . i.e. $f(n) \; mod \; P$. The infinite series start with 2,5,8,15,28...**

### 9.2.2 solution

**The problem can be solved by exponentiating the base matrix, otherwise the time complexity for calculating the same using loop be huge. To solve the problem with matrix exponentation**
**First choose start vector, which is given here, i,e, [2,5,8]. Size of start vector = 3 is choosen as current element only depends on previous three elements.**
**Then, base matrix, In chapter 7.1.1, I already explained how base matrix for sum of last three series is calculated. which is**

$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

**if exponent is less than equal to size of the starting vector, i.e. 3, then choose the answer from starting vertex itself. Otherwise, calculate effective exponent, exponent = n-x as descibed in the chapter 8.**
**Now, Calculate $BaseMatrix^{exponent}$ using binary ecponenetiation, as described in chapter 6.**
**Multiply base matrix with start vector $[StartVector]$ x $[BaseMatrix]$**
**Last element of the resulting vector is required $n^{th}$ element of the series.**

### 9.2.3 implementation (in c++)

```
/*
 * Created by mukul on 02-03-2022.
 */

 #include <bits/stdc++.h>

#define O cout<<
#define I cin>>
#define El endl;
//ds
#define ll long long
#define V(x) vector<x>

//constants
#define t9p7 1000000007
using namespace std;

template <typename T>
class matrix{
public:
    ll row,col;
    V(V(T)) mat;

    matrix(ll row,ll col,V(V(T)) mat){
        this->row=row;
        this->col=col;
        this->mat=mat;
    }

    //matrix multiplication
    V(V(T)) matrix_multiplication(V(V(T)) mat1,V(V(T)) mat2){
        V(V(T)) ans(mat1.size(),V(T)(mat2[0].size()));
        if(mat1[0].size()!=mat2.size()){
            O "error: mismatches in dimensions of the matrices"<<El
        }else{
            for (ll i = 0; i < mat1.size(); i++)
            {
```

```
                for (ll j = 0; j < mat1[0].size(); j++)
                {
                    ans[i][j] = 0;
                    for (ll x = 0; x < mat1[0].size() ; x++)
                    {
                        ans[i][j]+= (mat1[i][x] * mat2[x][j])%t9p7;
                        ans[i][j]%=t9p7;
                    }
                }
            }
        }
        return ans;
    }

    // matrix binary exponentiation
    V(V(T)) matrix_binary_exponentiation(ll n){
        V(V(T)) res = mat,a=mat;
        n--;
        while (n > 0) {
            if (n & 1)
                res =matrix_multiplication(res,a);
            a=matrix_multiplication(a,a);
            n >>= 1;
        }
        return res;
    }
};


void main(){
    ll n;
    I n;
    V(V(ll)) arr={ //base matrix
      //curr 1 2 3
            {0,0,1},//prev 1st
            {1,0,1},//prev 2nd
            {0,1,1},//prev 3rd
    };
    V(V(ll)) ini_mat={ //starting vector
```

```
            {2,5,8}
    };
    if(n<=3){//if n less than size of
        O ini_mat[0][n-1]<<El
        return;
    }

    //effective exponent
    ll n_pow=n-3;

    //creating instance of matrix class with base matrix
    matrix<ll> mat(3,3,arr);

    //binary exponentiation of base matrix
    V(V(ll)) res = mat.matrix_binary_exponentiation(n_pow);

    //multiplying the above result with start vector
    V(V(ll)) res2 = mat.matrix_multiplication(ini_mat, res);

    //answer is the last element of the previous result
    O res2[0][2]<<El
}
```

**Input: 12638127468124587122 Output: 487113806**

**i.e.** $fib_{12638127468124587122} \, mod \, 1000000007 \; = \; 487113806$

## 9.3 Problem from CSES website

### 9.3.1 problem

**Copied from CSES 'Your task is to count the number of ways to construct sum n by throwing a dice one or more times. Each throw produces an outcome between 1 and 6.**

**For example, if n=3, there are 4 ways:**

**1+1+1**

**1+2**

**2+1**

**3**

**Input**

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

**Output**

**Print the number of ways modulo $10^9 + 7$.**

**Constraints**

$1 <= n <= 10^6$

**Example**

**Input:**

**3**

**Output:**

**4**

**You can find the problem Dice Combinations from CSES. Here https://cses.fi/problemset/task/1633.**

### 9.3.2 solution

**If you have a vibe of dynamic programming, You should think about relation between different n. For example, for n=1, throwing dice once and get a '1', one possiblity, for n=2, dice thrown once and get 2, dice thrown twice and get 1 in each, so two possibilities. similarly n=3, 4 possibilities. n=4, 8 possibilities. n=5,**

**16 possibilities. n=6, 32 possibilities. and so on. you can calculate first few using normal dynamic programming approach also. Now, Think about recursive equation. Each n depends on previous 6 elements. how? if you get 1 in last throw to get to n then number of possibility for n-1 to be added to the current, i.e. n, similarly got 2 in last throw, you should include number of possibilities for n-2, and so on upto 6. Hence, the recursive equation becomes** $f_n = f_{n-1} + f_{n-2} + f_{n-3} + f_{n-4} + f_{n-5} + f_{n-6}$**. So, starting vector will be of size 6, which is [1,2,4,8,16,32]. Now, The base matrix be of size 6 x 6. According to the above recursive solution, start vector and base matrix are**

$$\begin{bmatrix} 1 & 2 & 4 & 8 & 16 & 32 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

**Next procedures are same as mentioned in chapter 8, if n¡=6 return from start vetor, otherwise calculate effective n. find** $base\ matrix^{effective\ n}$ **and multiply with start vector.** $6^{th}$ **element of the resulting vector is the** $n^{th}$ **element of the series. i.e. answer of the problem.**

### 9.3.3 implementation (in c++)

```
/*
 * Created by mukul on 11-03-2022.
 */
#include <bits/stdc++.h>
//io
#define O cout<<
#define I cin>>
#define El endl;
//ds
#define ll long long
#define V(x) vector<x>
//constants
#define t9p7 1000000007
using namespace std;
/*
 * matrix binary exponentiation
 */
template <typename T>
class matrix{
public:
    ll row,col;
    V(V(T)) mat;

    matrix(ll row,ll col,V(V(T)) mat){
        this->row=row;
        this->col=col;
        this->mat=mat;
    }

    V(V(T)) matrix_multiplication(V(V(T)) mat1,V(V(T)) mat2){
        V(V(T)) ans(mat1.size(),V(T)(mat2[0].size()));
        if(mat1[0].size()!=mat2.size()){
            O "error: mismatches in dimensions of the matrices"<<El
        }else{
            for (ll i = 0; i < mat1.size(); i++)
            {
                for (ll j = 0; j < mat1[0].size(); j++)
                {
```

```
                    ans[i][j] = 0;
                    for (ll x = 0; x < mat1[0].size() ; x++)
                    {
                        ans[i][j]+= (mat1[i][x] * mat2[x][j])%t9p7;
                        ans[i][j]%=t9p7;
                    }
                }
            }
        }
        return ans;
    }

    V(V(T)) matrix_binary_exponentiation(ll n){
        V(V(T)) res = mat,a=mat;
        n--;

        while (n > 0) {
            if (n & 1)
                res =matrix_multiplication(res,a);
            a=matrix_multiplication(a,a);
            n >>= 1;
        }
        return res;
    }


};



int main(){
    ll n;
    I n;
    //base matrix
    V(V(ll)) arr={
            {0,0,0,0,0,1},
            {1,0,0,0,0,1},
            {0,1,0,0,0,1},
            {0,0,1,0,0,1},
            {0,0,0,1,0,1},
```

```
            {0,0,0,0,1,1}

    };
    //start vector
    V(V(ll)) ini_mat={
            {1,2,4,8,16,32}
    };
    if(n<=6){
        O ini_mat[0][n-1]<<El
        return 0;
    }
    ll n_pow=n-6;
    matrix<ll> mat(6,6,arr);
    //matrix exponentiation
    V(V(ll)) res = mat.matrix_binary_exponentiation(n_pow);
    //multiplication with start vector
    V(V(ll)) res2 = mat.matrix_multiplication(ini_mat, res);
    //result
    O res2[0][5]<<El
    return 0;
}
```

**Input:123421 Output:399867581**

**Input: 21362831273 Output: 72967763**

**code at https://cses.fi/paste/0d8b31e036158d4837dc8b/**

**problem link https://cses.fi/problemset/task/1633**

## 9.4 Problem from coding contests

### 9.4.1 problem

**Copied from Codeforces ‘**

**While playing with geometric figures Alex has accidentally invented a concept of a n-th order rhombus in a cell grid.**

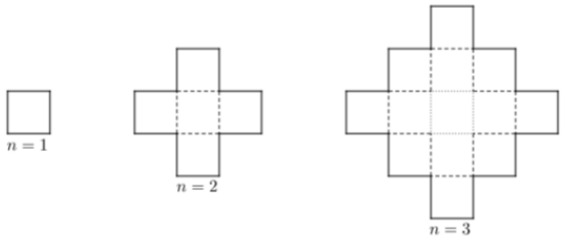



Figure 2: image representation of the problem

**A $1^{st}$ order rhombus is just a square 1x1 (i.e just a cell).**
**A n-th order rhombus for all n¿=2 one obtains from a $n-1^{th}$ order rhombus adding all cells which have a common side with it to it (look at the picture to understand it better).**

**Alex asks you to compute the number of cells in a $n^{th}$ order rhombus.**
**You can find the problem A. Alex and a Rhombus from Codeforces. Here https://codeforces.com/problemset/problem/1180/A.**

### 9.4.2 solution

**From the above problem, it is clearly visible that every time 4*(n-1) square is added to the $n-1^{th}$ solution. Hence, the recursive equation becomes $f_n = f_{n-1} + 4 * (n-1)$. current element dependent previous element and previous n. we will maintain (n-1) as well in this case. for n=1 no of square is 1, for n=2 number of square is 5. So, starting vector [1,2,1,5]. first two element is to maintain**

**(n-1), and second two for last two elements. Now, The base matrix be of size 4 x4 as start vector is of size 4. According to the above recursive solution, start vector and base matrix are**

$$\begin{bmatrix} 1 & 2 & 1 & 5 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 4 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

**Next procedures are same as mentioned in chapter 8, if** $n <= 2$ **return from last two element of start vetor , otherwise calculate effective n. find** $base\ matrix^{effective\ n}$ **and multiply with start vector.** $4^{th}$ **element of the resulting vector is the** $n^{th}$ **element of the series. i.e. answer of the problem.**

### 9.4.3 implementation (in c++)

```
/*
 * Created by mukul on 11-03-2022.
 */
#include <bits/stdc++.h>
//io
#define O cout<<
#define I cin>>
#define El endl;
//ds
#define ll long long
#define V(x) vector<x>
//constants
#define t9p7 1000000007
using namespace std;
```

```cpp
/*
 * matrix binary exponentiation
 */
template <typename T>
class matrix{
public:
    ll row,col;
    V(V(T)) mat;

    matrix(ll row,ll col,V(V(T)) mat){
        this->row=row;
        this->col=col;
        this->mat=mat;
    }

    V(V(T)) matrix_multiplication(V(V(T)) mat1,V(V(T)) mat2){
        V(V(T)) ans(mat1.size(),V(T)(mat2[0].size()));
        if(mat1[0].size()!=mat2.size()){
            O "error: mismatches in dimensions of the matrices"<<El
        }else{
            for (ll i = 0; i < mat1.size(); i++)
            {
                for (ll j = 0; j < mat1[0].size(); j++)
                {
                    ans[i][j] = 0;
                    for (ll x = 0; x < mat1[0].size() ; x++)
                    {
                        ans[i][j]+= (mat1[i][x] * mat2[x][j])%t9p7;
                        ans[i][j]%=t9p7;
                    }
                }
            }
        }
        return ans;
    }

    V(V(T)) matrix_binary_exponentiation(ll n){
        V(V(T)) res = mat,a=mat;
        n--;
```

```
        while (n > 0) {
            if (n & 1)
                res =matrix_multiplication(res,a);
            a=matrix_multiplication(a,a);
            n >>= 1;
        }
        return res;
    }
};


int main(){
    ll n;
    I n;

    V(V(ll)) arr={
      //curr 1 2 3 4
            {1,1,0,0},//prev 1st
            {0,1,0,4},//prev 2nd
            {0,0,0,0},//prev 3rd
            {0,0,1,1} //prev 4th
    };
    V(V(ll)) ini_mat={
            {1,2,1,5}
    };
    if(n<=2){
        O ini_mat[0][n+1]<<El
        return 0;
    }

    ll n_pow=n-2;
    matrix<ll> mat(4,4,arr);
    //matrix exponentiation
    V(V(ll)) res = mat.matrix_binary_exponentiation(n_pow);
    //multiplication with start vector
    V(V(ll)) res2 = mat.matrix_multiplication(ini_mat, res);
    //result
    O res2[0][3]<<El
    return 0;
```

```
}
```

**Input:123421 Output:465239431**

**Input: 21362831273 Output: 420143558**

**problem link https://codeforces.com/problemset/problem/1180/A**

# 10 Future Scope of work

**Well, Several other categories of recursive equation other than sum of terms are not mentioned in this long article, but they can also be done by manipulating matrix multiplication step, rather say it matrix operation step. Moreover, Matrix binary exponentiation has a huge scope in data science, where matrix exponentiation is requird, binary exponentiation of matrix be a lot helpful than just simply multiplying matrix multiple time. Other than all these, you can think of larger dimentional operations (more than just 2D) and their usage.**