/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

欧拉函数

O(logN)~~(?)~~

求[1, n - 1]中与n互质的数的个数

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int eular(int n)

{

int res = n, a = n;

for (int i = 2; i \* i <= a; i++)

{

if (a % i == 0)

{

res = res / i \* (i – 1);

while (a % i == 0)

a /= i;

}

}

if (a > 1)

res = res / a \* (a – 1);

return res;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

快速幂

O(logK)

求ak

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int fpow(int a, int k)

{

int base = 1;

while (k)

{

if (k & 1)

base = base \* a;

a = a \* a;

k >>= 1;

}

return base;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

欧几里得

O(log(a + b))

求a, b最大公约数

lcm(a, b) = a \* b / gcd(a, b)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int gcd(int a, int b)

{

return b > 0 ? gcd(b, a % b) : a;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

矩阵定义及乘法运算符重载

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <string.h>

struct Matrix{

static const int MAX = 100;

int n, m; // 行数，列数

int v[MAX][MAX];

Matrix(){ n = 0, m = 0; memset(v, 0, sizeof v); }

};

Matrix getide(const int& n) //单位矩阵

{

Matrix ans = Matrix();

ans.n = ans.m = n;

for (int i = 1; i <= n; i++)

ans.v[i][i] = 1;

return ans;

}

Matrix operator\* (const Matrix a, const Matrix b)

{

Matrix ans = Matrix();

int n = a.n, m = b.m, t = a.m;

ans.n = a.n, ans.m = b.m;

for (int i = 1; i <= n; i++)

for (int j = 1; j <= m; j++)

for (int k = 1; k <= t; k++)

ans.v[i][j] += a.v[i][k] \* b.v[k][j];

return ans;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

fibnacci(矩阵快速幂求法)

O(logN)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 此处矩阵定义及乘法运算符重载直接套用前面的 \*/

const int Mod = 1e9 + 7;

int fibnacci(int n)

{

/\* 此处为f(1) = 1, f(2) = 1的情况 \*/

if (n == 1 || n == 2)

return 1;

n -= 2;

Matrix base, a;

base.n = 2; base.m = 1;

base.v[1][1] = base.v[2][1] = 1;

a.n = a.m = 2;

a.v[1][1] = a.v[1][2] = a.v[2][1] = 1;

while (n > 0)

{

if (n & 1)

base = a \* base;

a = a \* a;

n >>= 1;

}

return base.v[1][1];

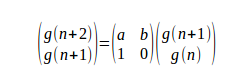
}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

fibnacci矩阵求法公式:

 (来自Wiki：https://en.wikipedia.org/wiki/Fibonacci\_number）

可将fibnacci视为 f(n) = 1 \* f(n – 1) + 1 \* f(n – 2), 那么就得到了 g(n) = a \* g(n – 1) + b \* g(n – 2)的矩阵求法公式：



~~画工极差了解下？~~

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/