$martix$ & 高精度乘法

> 注释:主要应用为高精度乘法

你是否曾因为大数乘法太慢而发愁，高精度不会写而发愁?

没关系，用了我们的*偏方*,大病小病都能治好，不用内存爆，简单易用!

对于数字A和B,找到序列

$$a\{a_1,a_2,a_3,a_4,\cdots,a_n\}和b\{b_1,b_2,b_3,\cdots,b_m\}$$

满足

$$\sum_{k=1}^{n}a_k10^k\ equals\ to\ a$$

$$\sum_{k=1}^{m}b_k10^k\ equals\ to\ b$$

存储为矩阵(向量)

$$A=\begin{pmatrix}a_1\\a_2\\a_3\\\vdots\\a_n\end{pmatrix}和B=\begin{pmatrix}b_1&b_2&b_3&\cdots&b_m\end{pmatrix}$$

只需要计算

$$C=A*B$$

再向左下方"推",便可得到**答案**

原理:

记答案矩阵左上角的元素坐标为$(1,1)$,共n行m列,右下角坐标为(n,m),我们为了理解直观,放到一个矩阵里

$$\begin{pmatrix} 0 & b_1 & b_2 & b_3 & \cdots & b_m \\ a_1 & a_1b_1 & a_1b_2 & a_1b_3 & \cdots & a_1b_m \\ a_2 & a_2b_1 & a_2b_2 & a_2b_3 & \cdots & a_2b_m \\ a_3 & a_3b_1 & a_3b_2 & a_3b_3 & \cdots & a_3b_m \\ a_4 & a_4b_1 & a_4b_2 & a_4b_3 & \cdots & a_4b_m \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_n & a_nb_1 & a_nb_2 & a_nb_3 & \cdots & a_nb_m \end{pmatrix}$$

将a=17,b=23作为例子带入

$$\begin{pmatrix} 0 & 2_1 & 3_0 \\ 1_1 & 2_2 & 3_1 \\ 7_0 & 14_1 & 21_0 \end{pmatrix}$$

其中第一列右下角的数字是$n \to 1$,第一行是$1 \to m$,表示的是10^n中的n

我们拿出其中的1_1和2_1,表示的是17的十位和23的十位,表示的真实的值是

$$10 * 20 = 100$$
$$1_1 * 2_1 = 2_2$$

再推到一个整数里,得到

$$2 * 10^2 + 3 * 10^1 + 14 * 10^1 + 21 * 10^0 = 200 + 30 + 140 + 21 = 391$$

这边是我们需要的答案，对应到$C++$中，只需要处理进位就行了

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#define max(a,b) ((a) > (b) ? (a) : (b))
#define min(a,b) ((a) < (b)? (a) : (b))
using namespace std;
template <unsigned long long rows, unsigned long long cols>
class martix{
    public:
    unsigned long long a[rows + 1][cols + 1];
    void init(){memset(a, 0, sizeof(a));}
    inline unsigned long long *operator[](const unsigned long long &p){return a[p];
    template <unsigned long long rows2, unsigned long long cols2>
    martix<rows, cols2> operator*(martix<rows2, cols2> &O){
        martix<rows, cols2> m;
        m.init();
        for (unsigned long long i = 1; i <= rows; i++){
            for (unsigned long long j = 1; j <= cols2; j++){
                for (unsigned long long k = 1; k <= cols; k++){
                    m[i][j] += (a[i][k] * O[k][j]);
                }
            }
```

```
        }
        return m;
    }
};
int dec_len(int n){
    int len = 0;
    while (n){
        len++;
        n /= 10;
    }
    return max(1, len);
}
int main(){
    long long n, m,lm,ln,cur,ans=0;
    martix<10, 1> m1;martix<1, 10> m2;
    m1.init();m2.init();
    scanf("%lld %lld",&n,&m);
    ln = dec_len(n), lm = dec_len(m),cur = ln;
    while (n){
        m1[cur][1] = n % 10;n /= 10;cur--;
    }
    cur = lm;
    while (m){
        m2[1][cur] = m % 10;m /= 10;cur--;
    }
    martix<10, 10> m3 = m1 * m2;
    for (int i = ln; i >= 1; i--){
        for (int j = lm; j >= 1; j--){
            ans+=((int)pow(10,ln+lm-i-j))*m3[i][j];
        }
    }
    printf("%lld",ans);
}
```