A题
小水獭和高精度乘法

20374090 梁跞方

目录

CONTENES

第
壹
章

PART. 1

题 目 介 绍

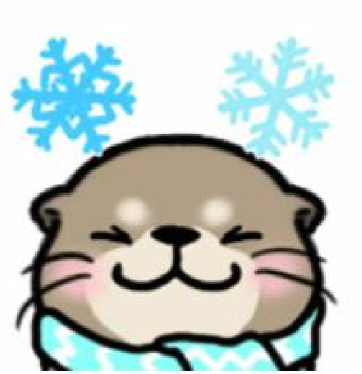
第壹章回

A 小水獭和高精度乘法

时间限制：3000ms 内存限制：65536kb

通过率：164/189 (86.77%) 正确率：164/1146 (14.31%)

题目描述



小水獭想请你编写一个高精度乘法计算器，给定正整数 A 和 B，请你输出 $C = AB$。

输入格式

第一行一个正整数 t （$1 \le t \le 10$），表示数据组数。

对于每组数据，第一行两个正整数 A, B （$1 \le A, B < 10^{10^5}$），含义如题目所示。

输出格式

对于每组数据，输出一行一个正整数 $C = AB$。

第贰章

PART. 2

FFT 算 法

第 贰 章 回

直接利用字符串进行计算？
O（n²），太复杂

利用FFT算法，将系数表示转点值表示

所以伟大数学家傅里叶取了一些特殊的点代入，从而进行优化。
他规定了点值表示中的n个x为n个模长为1的复数。这n个复数不是随机的，而是**单位根。**
把上述的n个复数（单位根）$\omega_n^0, \omega_n^1, \ldots\ldots, \omega_n^{n-1}$代入多项式，能得到一种特殊的点值表示，这种点值表示就叫**DFT（离散傅里叶变换）**。

虽然DFT能把多项式转换成点值，但它仍然是暴力代入n 个数，复杂度仍然是O(n2)，所以它只是快速傅里叶变换的朴素版。所以我们要考虑利用单位根的性质，加速我们的运算，得到FFT（快速傅里叶变换）

FFT，利用单位根的性质

对于多项式 $A(x) = a_0 + a_1x + a_2x^2 + \ldots + a_{n-1}x^{n-1}$

将A(x)的每一项按照下标的奇偶分成两部分:

$A(x) = a_0 + a_2x^2 + \ldots + a_{n-2}x^{n-2} + x * (a_1 + a_3x^2 + \ldots + a_{n-1}x^{n-2})$

设两个多项式 $A_0(x)$和$A_1(x)$，令:

$A_0(x) = a_0x^0 + a_2x^1 + \ldots + a_{n-2}x^{n/2-1}$

$A_1(x) = a_1x^0 + a_3x^1 + \ldots + a_{n-1}x^{n/2-1}$

显然，$A(x) = A_0(x^2) + x * A_1(x^2)$

假设$k < n$，代入$x = \omega_n^k$（n次单位根）

$A(\omega_n^k) = A_0(\omega_n^{2k}) + \omega_n^k * A_1(\omega_n^{2k})$

$= A_0(\omega_{\frac{n}{2}}^k) + \omega_n^k * A_1(\omega_{\frac{n}{2}}^k)$

考虑A1(x)和A2(x)分别在$(\omega_{\frac{n}{2}}^1, \omega_{\frac{n}{2}}^2, \omega_{\frac{n}{2}}^3, \ldots, \omega_{\frac{n}{2}}^{\frac{n}{2}-1})$的点值表示已经求出，就可以O(n)求出A(x)在$(\omega_n^1, \omega_n^2, \omega_n^3, \ldots, \omega_n^{n-1})$处的点值表示。这个操作叫**蝴蝶变换**

$A(\omega_n^{k+\frac{n}{2}}) = A_0(\omega_n^{2k+n}) + \omega_n^{k+\frac{n}{2}} * A_1(\omega_n^{2k+n})$

$= A_0(\omega_{\frac{n}{2}}^k) - \omega_n^k * A_1(\omega_{\frac{n}{2}}^k)$

→ **蝴蝶变换**

第叁章

PART. 3

迭 代 FFT

第 叁 章 回

迭代FFT时，要把各个系数不断分组并放到两侧，一个系数原来的位置和最终的位置的规律如下：

初始位置：$\omega_n^0\ \omega_n^1\ \omega_n^2\ \omega_n^3\ \omega_n^4\ \omega_n^5\ \omega_n^6\ \omega_n^7$

第一轮后：$\omega_n^0\ \omega_n^2\ \omega_n^4\ \omega_n^6 | \omega_n^1\ \omega_n^3\ \omega_n^5\ \omega_n^7$

第二轮后：$\omega_n^0\ \omega_n^4 | \omega_n^2\ \omega_n^6 | \omega_n^1\ \omega_n^5 | \omega_n^3\ \omega_n^7$

第三轮后：$\omega_n^0 | \omega_n^4 | \omega_n^2 | \omega_n^6 | \omega_n^1 | \omega_n^5 | \omega_n^3 | \omega_n^7$

“|”代表分组界限

第 叁 章 回

将每个位置用二进制表现出来，位置 `x` 上的数，最后所在的位置为： `x` 二进制翻转后得到的数字；

例如：

4（100）最后所在位置为：1（001）；

5（101）最后所在位置为：5（101），不变；

3（011）最后所在位置为：6（110）。

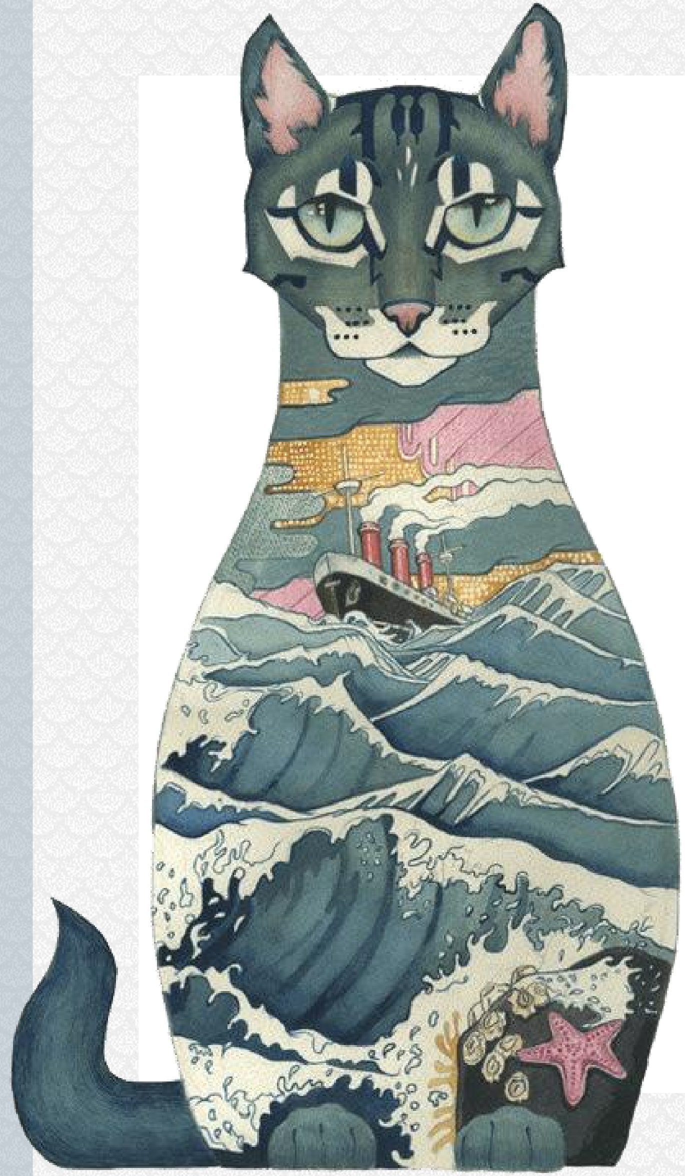
每一次回溯时只扫当前前面一半的序列，即可得出后面一半序列的答案

$n == 1$时只有一个常数项，直接$return$

时间复杂度$O(n\log_2 n)$

```
void my_reverse(int k){
    int len=1<<k;
    for(int i=0;i<len;i++)
        rev[i]=(rev[i>>1]>>1)|((i&1)<<(k-1));
}
```

第 叁 章 回



FFT ： 系数 → 点值表示法

FFT： 多项式相乘 → 点值相乘

IFFT ： 点值表示法→系数表示

一个重要结论

把多项式A(x)的离散傅里叶变换结果作为另一个多项式B(x)的系数，取单位根的倒数即$\omega_n^0, \omega_n^{-1} \ldots\ldots, \omega_n^{1-n}$作为x代入B(x)，得到的每个数再除以n，得到的是A(x)的各项系数，这就实现了傅里叶变换的逆变换了。相当于在FFT基础上再搞一次FFT。

因此只需要对点值表示的复数值再进行一次FFT，就可以将其变为系数表示法

第肆章

PART. 4

代 码 展 示

第肆章回

```cpp
#include<bits/stdc++.h>
#include <complex>
using namespace std;
//complex是stl自带的定义复数的容器
typedef complex<double> cp;
//N要比位数还多大一些，因为算法要求位数是2的次幂
#define N 300005
//pi表示圆周率π
const double pi=acos(-1);
//int n;

class BIGMUL{
public:
    cp a[N],b[N];
    int rev[N],ans[N];
    char s1[N],s2[N];
    //读入优化
    int read(){
        int sum=0,f=1;
        char ch=getchar();
        while(ch>'9'||ch<'0'){if(ch=='-')f=-1;ch=getchar();}
        while(ch>='0'&&ch<='9'){sum=(sum<<3)+(sum<<1)+ch-'0';ch=getchar();}
        return sum*f;
    }
    //初始化每个位置最终到达的位置(恰好为二进制取反)
    //k见main函数中的说明
    void my_reverse(int k){
        int len=1<<k;
        for(int i=0;i<len;i++)
            rev[i]=(rev[i>>1]>>1)|((i&1)<<(k-1));
    }
    //a表示要操作的系数，n表示序列长度
    //若flag为1，则表示FFT，为-1则为IFFT(需要求倒数)
```

```cpp
    //a表示要操作的系数，n表示序列长度
    //若flag为1，则表示FFT，为-1则为IFFT(需要求倒数)
    void fft(cp *a,int n,int flag){
        for(int i=0;i<n;i++)
        {
            //i小于rev[i]时才交换，防止同一个元素交换两次，回到它原来的位置。
            if(i<rev[i])swap(a[i],a[rev[i]]);
        }
        for(int h=1;h<n;h*=2)//h是准备合并序列的长度的二分之一
        {
            cp wn=exp(cp(0,flag*pi/h));//求单位根w_n^1
            for(int j=0;j<n;j+=h*2)//j表示合并到了哪一位
            {
                cp w(1,0);
                for(int k=j;k<j+h;k++)//只扫左半部分，得到右半部分的答案
                {
                    cp x=a[k];
                    cp y=w*a[k+h];
                    a[k]=x+y;  //这两步是蝴蝶变换
                    a[k+h]=x-y;
                    w*=wn; //求w_n^k
                }
            }
        }
        //判断是否是FFT还是IFFT
        if(flag==-1)
        for(int i=0;i<n;i++)
        a[i]/=n;
    }
    void MAIN(){
        scanf("%s%s",s1,s2);
        int l1 = strlen(s1), l2 = strlen(s2);
        // k表示数组的二进制位数，s表示分割的长度
        // 2的k次幂大于l1+l2, s = 2^k
        // 长度必须补成2的k次幂是分治的要求
        int k = 1, s = 2;
        for (k = 1; (1 << k) < l1 + l2 - 1; k++) s <<= 1;
        //读入的数的每一位看成多项式的一项，保存在复数的实部
        for(int i=0;i<l1;i++) a[i]=(double)(s1[l1-i-1]-'0');
        for(int i=0;i<l2;i++) b[i]=(double)(s2[l2-i-1]-'0');
        //k表示转化成二进制的位数
        my_reverse(k);
        //FFT 把a的系数表示转化为点值表示
        fft(a,s,1);
        //FFT 把b的系数表示转化为点值表示
        fft(b,s,1);
```

```cpp
        fft(b,s,1);
        //FFT 两个多项式的点值表示相乘
        for(int i=0;i<s;i++)
            a[i]*=b[i];
        //IFFT 把这个点值表示转化为系数表示
        fft(a,s,-1);
        //保存答案的每一位(注意进位)
        for(int i=0;i<s;i++)
        {
            //取实数四舍五入，此时虚数部分应当为0或由于浮点误差接近0
            ans[i]+=(int)(a[i].real()+0.5);
            ans[i+1]+=ans[i]/10;
            ans[i]%=10;
        }
        //删除前导零
        while(!ans[s]&&s>-1) s--;
        //结果为0
        if(s==-1)printf("0");
        else
        for(int i=s;i>=0;i--)
            printf("%d",ans[i]);
        printf("\n");
        return;
    }
};


int main(){
    //n=read();
    int t;
    scanf("%d",&t);
    while(t--){
        BIGMUL *bigmul = new BIGMUL();
        bigmul->MAIN();
        delete bigmul;
    }
    return 0;
}
```

谢谢大家