C2-E题题解

题目描述

给定两个非负整数A,B，试求$A \times B$.

输入

本题测试点包含多组数据。

第一行，一个正整数$T(1 \leq T \leq 10)$，表示数据组数。

对于每组数据：

第一行，一个非负整数$A(0 \leq A \leq 10^{2000})$。

第二行，一个非负整数$B(0 \leq B \leq 10^{2000})$。

输出

对于每组数据：

输出一行，一个整数，表示$A \times B$.

输入样例

```
2
0
0
123456789
987654321
```

输出样例

```
0
121932631112635269
```

解题思路

本题需要用到**高精度计算。**

高精度计算

定义

高精度计算（Arbitrary-Precision Arithmetic），也被称作大整数（big num）计算，运用了一些算法结构来支持更大整数间的运算（数字大小超过语言内建整型）

存储

在平常的实现中，高精度数字利用**字符串**表示，每一个字符表示数字的一个十进制位。因此可以说，高精度数值计算实际上是一种特别的字符串处理。

读入字符串时，数字最高位在字符串首（下标小的位置）。但是习惯上，下标最小的位置存放的是数字的**最低位**，即存储反转的字符串。这么做的原因在于，数字的长度可能发生变化，但我们希望同样权值位始终保持对齐（例如，希望所有的个位都在下标 `[0]`，所有的十位都在下标 `[1]`……）；同时，加、减、乘的运算一般都从个位开始进行（回想小学的竖式运算），这都给了「反转存储」以充分的理由。

此后我们将一直沿用这一约定。

高精度乘法

以本题代码为例，介绍一下高精度加法的步骤以及需要注意的地方。

```cpp
#include <bits/stdc++.h>

using namespace std;
// 注意a和b最多是2001位，所以它们相乘的结果最多是4002位，我们数组大小开4010
const int N = 4e3 + 10;
int c[N];
char a[N], b[N];

// 反转字符串
void reverse(char s[]) {
    int temp, i, j;
    for (i = 0, j = strlen(s) - 1; i < j; i++, j--) {
        temp = s[i];
        s[i] = s[j];
        s[j] = temp;
    }
}

int main() {
    int t;
    cin >> t;
    while (t--) {
        memset(c, 0, sizeof c);
        scanf("%s", a);
        scanf("%s", b);
        // 例如一个三位数x三位数，结果的位数最大是多少呢?
        // 当结果最大时，位数最大，所以我们计算999 * 999 它的结果小于1000 * 1000
        // 也就是小于1000000 它是7位，所以三位数*三位数的结果，位数最大为3+3 = 6
        // 所以len_c 设置为len_a + len_b
        int len_a = strlen(a), len_b = strlen(b), len_c = len_a + len_b;
        // 然后我们反转a和b
        reverse(a), reverse(b);
        // 下面是一个模仿竖式计算的过程
        // 在竖式计算中，a在下，b在上，进行计算
```

```
        for (int i = 0; i < len_a; i++) {
            for (int j = 0; j < len_b; j++) {
                // 数组c是int型，a,b都是char型
                c[i + j] += (a[i] - '0') * (b[j] - '0');
                // 考虑需要进位的情况
                if (c[i + j] > 9) {
                    c[i + j + 1] += c[i + j] / 10;
                    c[i + j] %= 10;
                }
            }
        }
        // 去除前导0
        // 前导0只会出现在下标大于等于1的位置，如果出现在下标为0的位置，那么结果就是0，且此时它不可以被称作前导0，且不可被去除。
        // 所以我们限制前导0的查找范围，避免结果为0的时候被丢掉。
        while (len_c > 1 && c[len_c - 1] == 0)len_c--;
        // 以先输出高位，再输出低位的形式输出结果
        for (int i = len_c - 1; i >= 0; i--)printf("%d", c[i]);
        puts("");

    }

    return 0;
}
```