题目

E A*B Problem (Integer Ver.)

时间限制：1000ms 内存限制：65536kb

通过率：108/140 (77.14%) 正确率：108/415 (26.02%)

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 $A \times B$。

思路

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。

代码实现

```
#include <iostream>
#include <vector>

using namespace std;

vector<int> mul(vector<int> &A, vector<int> &B) {
    vector<int> C(A.size() + B.size() + 7, 0); // 初始化为 0，C的size可以大一点
```

```
    for (int i = 0; i < A.size(); i++)
        for (int j = 0; j < B.size(); j++)
            C[i + j] += A[i] * B[j];

    int t = 0;
    for (int i = 0; i < C.size(); i++) { // i = C.size() - 1时 t 一定小于 10
        t += C[i];
        C[i] = t % 10;
        t /= 10;
    }

    while (C.size() > 1 && C.back() == 0) C.pop_back(); // 必须要去前导 0，因为最高
位很可能是 0
    return C;
}

int main() {
    int T;
    cin >> T;
    while(T--)
    {
    string a, b;
    cin >> a >> b;

    vector<int> A, B;
    for (int i = a.size() - 1; i >= 0; i--)
        A.push_back(a[i] - '0');
    for (int i = b.size() - 1; i >= 0; i--)
        B.push_back(b[i] - '0');

    auto C = mul(A, B);

    for (int i = C.size() - 1; i >= 0; i--)
        cout << C[i];
    cout << endl;
    }
    return 0;
}
```

要注意的细节

注意要去前导 0，因为最高位很可能是 0