My Courses

CST2020 1-1 A+B problem

描述

抱歉，这题实际是 A*B problem。

邓俊辉老师的作业常常过于简单，数据类型只需使用 int。助教们一致认为，向同学们介绍 Python 中自带的长整型是十分有必要的。例如，它可以计算几百位的整数乘法。 但是，在介绍长整型之前，助教决定让你自己实现一遍长整型乘法，以加深对它的理解。

输入

输入共包含 n + 1 行，第 1 行包含一个整数 n，表示你需要计算 n 组乘法。

接下来 n 行，每行包含两个非负整数 a 和 b。

输出

输出共包含 n 行，请对于每一组输入的 a、b，输出他们的乘积。

输入样例

```
3
1 1
2 2
123123 789789
```

*此样例是第 1 个测试点。

*本课堂的编程作业中，对于非全 int 输入的题目，有的会把一个样例作为第 1 个测试点方便调试。

输出样例

```
1
4
97241191047
```

*注：参考答案的每一行都有换行符，最后一行也有换行符。但 OJ 比对文本时一般会忽略行末空格和文末空行。

数据范围

1 ≤ n ≤ 500

0 ≤ a < 10^5000

0 ≤ b < 10^5000

资源限制

时间限制：1 sec

内存限制：256 MB

工具

我们这里提供一个用于自我检测程序正确性的方式：对拍

对拍主要用在以下场景：我拥有一个数据生成器，一个程序 A，一个程序 B，程序 A 和程序 B 是执行相同任务的程序。我们使用一个对拍脚本，每次先调用数据生成器，然后将数据分别给 A 和 B 得到两组输出，通过对比输出是否一致来判断程序是否运行正确。这样做的原因在于往往我们可以想到一个简单的方法 A，但是效率低下，而高效率的程序 B 的正确性无法保证，通过这样的过程我们可以简单评测，以及找到使得 B 错误的数据用来调试。

对拍是一种调试技巧。本次作业提供了一套对拍的工具，大家可以学习使用，并运用到之后的代码调试中去。

工具包主要由 3 个部分组成：

1. check.py 实现了对拍的功能，其使用 Python3 编写。

2. a.cpp 和 b.py 为程序 A 和 B。对于 a.cpp，使用前需要将其编译。

3. makedata.py 为一个数据生成器，其使用 Python3 编写，可以生成一组数据并输出到标准输出。

如何使用该工具：

1. 将 a.cpp 和 b.py 替换为你需要对拍的程序，或者不进行替换，然后编译需要编译的程序。

2. 设置 makedata.py 中的有关参数。

3. 运行 check.py。

运行 check.py 时需指定程序 A 和 B，以及数据生成器的运行命令。

例如在 Linux 中：

```
python3 check.py ./a.out "python3 b.py" "python3 makedata.py"
```

或在 Windows 中：

```
python3 check.py a.exe "python3 b.py" "python3 makedata.py"
```

工具包下载 (attachment/9fe0/9fe05677323a7d7083ed2e34f04f2a7b6a88371f.zip)

提示

对于所有问题，请尽力优化你的算法并确保其运行正确。用于黑盒测评的 19, 20 两个测例常常是整个问题中规模最大（或者对边界条件最高）的两个测例，而且可能会特意构造特殊情况和边界情况。

即便是 64 位整型，由于存储字节有限，所以不能完整表示一个很大整数的精确值（它只能区分 2^64 个不同的数值），无法表示两个乘数，遑论计算。而对于更大整数的运算，这时候就得用到其他的方法，我们称之为高精度算法。例如我们考查高精度加法，一种最简单的思路如下

例：

```
  12345678910111213 + 1111111111111111111
```

使用两个数组存储：

```
a[] = {3, 1, 2, 1, 1, 1, 0, 1, 9, 8, 7, 6, 5, 4, 3, 2, 1};
b[] = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1};
```

两个数组分别把数值倒存,逐位相加,每位加后判断是否大于 10，再进行进位即可。

对于高精度的乘法，问题相应来说会复杂一些，类似于刚才的加法思路，这一过程中有这样一些问题值得考虑：

1. 对于这两个整数应当如何存储？2 进制？10 进制？（或者其它）低位在前？高位在前？（或者其它）

2. 如何存储可以保证正确的情况下最大程度上优化算法的常数。
