OOP Lab5 Report

孔恩燊 23307130021

2025.4.26

补充 bignum_test.cc

在原有的 normal test 基础上，增加了不同符号数的加减乘除：正数和负数、负数和正数、负数和负数。所有测试均能通过。

效率分析与优化建议

1. 当前实现分析

当前 `Bignum` 使用 `std::string` 存储十进制大整数，主要运算通过模拟逐位计算实现：

操作	实现方式	时间复杂度
加法	从低位向高位模拟手算加法	$O(n)$
减法	模拟逐位借位减法	$O(n)$
乘法	“竖式乘法”逐位叠加	$O(n \times m)$
除法	逐位不断减去被除数	$O(n \times d)$

逻辑清晰，实现简单，但复杂度较高，大规模运算将非常缓慢（如上千位整数）。

2. 可能的优化方法

(1) 使用 `std::vector<int>` 代替 `std::string`

将每一位数字转为整数存储，避免频繁的字符-数字转换。更利于做进位、模运算、高基数优化。

实现难点：

所有现有函数需重构，需要额外处理输出格式转换。

(2) 高基数压缩存储

使用如 10^8 为基数，每个 `int` 存储多位十进制数；显著减少乘法、除法中的循环次数。

实现难点：

输入输出格式转换更加复杂，运算时对每位最大值及进位的处理也更为复杂。

(3) Karatsuba 快速乘法

查阅资料看到的递归分治乘法算法，将乘法分解为三次子乘法 + 加法组合，时间复杂度降低为 $O(n^{\log_2 3})$。

实现难点：

分治边界处理麻烦，实现与调试较复杂，对于没那么大的数字的加减乘除，效率没有优化多少。

(4). “二分逼近商”优化除法优化

当前除法是暴力减法，可使用“二分逼近商”来提高效率。

实现难点：

估值精度控制较为繁琐，运算过程需要与基数存储方式兼容。

3. 性能对比测试设计

(1) 测试方法

- 方法1：使用 C++ 自带的 `std::` 计时工具计时，对比加减乘除的耗时。
- 方法2：用第三方工具测试，如 Google 的测试框架 GTest。

(2) 结果对比

- 方法1：根据运行时间的多少来判断性能，可使用绘图工具如 Python 的 Matplotlib 库比较不同输入规模下的耗时曲线。
- 方法2：观察第三方库给出的性能比较结果。