

高精度整数运算库详细说明

概述

本文档详细说明高精度整数运算库 `bigint.cpp` 的实现和使用方法。该库提供了任意精度的整数运算能力，支持正负数运算、基本算术运算、比较运算以及字符串转换功能。

存储结构与表示方式

该库使用 10^9 进制存储大整数，具有以下特点：

- **存储结构**：每个大整数由 `std::vector<int> a` 和一个符号位 `sign` 组成
- **进制基数**： $base = 10^9$ ，每个数组元素存储一个 0 到 $10^9 - 1$ 之间的数值
- **存储顺序**：数组 `a` 从低位到高位存储，即 `a[0]` 存储最低 9 位十进制数
- **符号表示**：`sign = 1` 表示正数，`sign = -1` 表示负数，零的符号为 1
- **规范化存储**：通过 `trim()` 函数确保数组末尾无前导零，零表示为空数组

构造函数与基本操作

默认构造函数 `BigInt()`

- **功能**：创建值为 0 的大整数
- **时间复杂度**： $O(1)$
- **使用示例**：`BigInt x;` 创建值为 0 的大整数

整型构造函数 `BigInt(long long v)`

- **功能**：从长整型创建大整数，可支持负数与 0 的情况
- **时间复杂度**： $O(\log_{10^9} |v|)$
- **注意事项**：支持最大范围为长整型所能表示的整数

字符串构造函数 `BigInt(const string& s)`

- **功能**：从十进制字符串创建大整数
- **时间复杂度**： $O(L)$ ，其中 L 为字符串长度
- **输入格式**：

- 允许前导空格
- 可选正负号
- 十进制数字序列，不允许前导零（除非数值本身就是 0）
- 示例: "-12345678901234567890"
- **异常情况**: 如果字符串包含非数字字符（除前导空格和正负号外），行为未定义

转换为字符串 `string toString() const`

- **功能**: 将大整数转换为十进制字符串表示
- **时间复杂度**: $O(n)$
- **输出格式**: 与输入格式兼容，无前导零（零除外）

流输入输出 `operator>>` 和 `operator<<`

- **功能**: 支持标准 C++ 流式输入输出
- **时间复杂度**: 与字符串转换相同
- **实现细节**:
 - `operator>>`: 读取字符串后调用 `read()` 函数
 - `operator<<`: 调用 `toString()` 函数并输出
- **使用示例**:

```
1 BigInt a, b;
2 cin >> a >> b;
3 cout << a + b << endl;
```

绝对值函数 `BigInt abs() const`

- **功能**: 返回当前大整数的绝对值
- **时间复杂度**: $O(1)$ （仅复制，不复制底层数组数据）

零值判断 `bool isZero() const`

- **功能**: 判断大整数是否为零
- **时间复杂度**: $O(1)$

规范化函数 `void trim()`

- **功能**: 移除数组末尾的零元素，规范化表示
- **时间复杂度**: $O(n)$ ，最坏情况下需要检查所有元素
- **调用时机**: 所有可能产生前导零的操作后都应调用此函数

比较运算符

比较运算的时间复杂度为 $O(\max(n, m))$ ，其中 n 和 m 分别为两个操作数的 10^9 进制位数。

绝对大小比较 `static int absCompare(const BigInt& x, const BigInt& y)`

- 功能：比较两个大整数的绝对值大小
- 返回值：
 - -1 : $|x| < |y|$
 - 0 : $|x| = |y|$
 - 1 : $|x| > |y|$

带符号比较 `int compare(const BigInt& v) const`

- 功能：比较两个大整数（考虑符号）

比较运算符列表

- `bool operator<(const BigInt& a, const BigInt& b)`: 小于
- `bool operator>(const BigInt& a, const BigInt& b)`: 大于
- `bool operator<=(const BigInt& a, const BigInt& b)`: 小于等于
- `bool operator>=(const BigInt& a, const BigInt& b)`: 大于等于
- `bool operator==(const BigInt& a, const BigInt& b)`: 等于
- `bool operator!=(const BigInt& a, const BigInt& b)`: 不等于

所有比较运算符均通过调用 `compare()` 函数实现。

算术运算符

取负运算符 `BigInt operator-() const`

- 功能：返回当前大整数的相反数
- 时间复杂度: $O(1)$
- 使用示例: `BigInt b = -a;`

绝对值加法 `static BigInt absAdd(const BigInt& x, const BigInt& y)`

- 功能：计算两个非负大整数的和
- 时间复杂度: $O(\max(n, m))$

绝对值减法 `static BigInt absSub(const BigInt& x, const BigInt& y)`

- **功能**: 计算两个非负大整数的差, 要求 $|x| \geq |y|$
- **时间复杂度**: $O(n)$, 其中 n 为 x 的位数
- **前提条件**: 调用者需确保 $|x| \geq |y|$

加法赋值运算符 `BigInt& operator+=(const BigInt& v)`

- **功能**: 将当前大整数加上另一个大整数
- **时间复杂度**: $O(\max(n, m))$
- **使用示例**: `a += b;`

减法赋值运算符 `BigInt& operator--=(const BigInt& v)`

- **功能**: 将当前大整数减去另一个大整数
- **时间复杂度**: 与加法相同
- **使用示例**: `a -= b;`

二元加法运算符 `BigInt operator+(BigInt a, const BigInt& b)`

- **功能**: 返回两个大整数的和
- **时间复杂度**: $O(\max(n, m))$
- **使用示例**: `BigInt c = a + b;`

二元减法运算符 `BigInt operator-(BigInt a, const BigInt& b)`

- **功能**: 返回两个大整数的差
- **时间复杂度**: $O(\max(n, m))$
- **使用示例**: `BigInt c = a - b;`

乘以整数的赋值运算符 `BigInt& operator*=(int m)`

- **功能**: 将当前大整数乘以一个整数
- **时间复杂度**: $O(n)$
- **注意事项**: m 应小于 10^9 , 否则可能溢出
- **使用示例**: `a *= 123;`

乘以大整数的赋值运算符 `BigInt& operator*=(const BigInt& v)`

- **功能**: 将当前大整数乘以另一个大整数
- **时间复杂度**: $O(n \cdot m)$
- **使用示例**: `a *= b;`

乘以整数的二元运算符

- `BigInt operator*(BigInt a, int m)`: 大整数乘以整数
- `BigInt operator*(int m, BigInt a)`: 整数乘以大整数
- 实现细节: 通过拷贝和乘法赋值实现
- 时间复杂度: $O(n)$
- 使用示例:

```
1 BigInt b = a * m;  
2 BigInt c = m * a;
```

乘以大整数的二元运算符 `BigInt operator*(BigInt a, const BigInt& b)`

- 功能: 返回两个大整数的乘积
- 时间复杂度: $O(n \cdot m)$
- 使用示例: `BigInt c = a * b;`

除以整数的赋值运算符 `BigInt& operator/=(int m)`

- 功能: 将当前大整数除以一个整数, 结果为整数商
- 时间复杂度: $O(n)$
- 注意事项: 不检查除零错误 (调用者需确保 $m \neq 0$)
- 使用示例: `a /= 123;`

整数取模运算符 `int operator%(int m) const`

- 功能: 返回当前大整数除以整数的余数 (考虑符号)
- 时间复杂度: $O(n)$
- 注意事项: m 不应为 0
- 使用示例: `int r = a % 123;`

除以整数的二元运算符 `BigInt operator/(BigInt a, int m)`

- 功能: 返回大整数除以整数的整数商
- 时间复杂度: $O(n)$
- 使用示例: `BigInt b = a / 123;`

长除法算法 `static pair<BigInt, BigInt> divmod(const BigInt& a1, const BigInt& b1)`

- 功能: 计算大整数除法的商和余数
- 时间复杂度: $O(n \cdot (n - m))$, 其中 n 为被除数位数, m 为除数位数

除以大整数的赋值运算符 `BigInt& operator/=(const BigInt& v)`

- **功能：**将当前大整数除以另一个大整数，结果为整数商
- **时间复杂度：** $O(n \cdot (n - m))$
- **异常情况：**若除数为零，抛出异常
- **使用示例：**`a /= b;`

取模赋值运算符 `BigInt& operator%=(const BigInt& v)`

- **功能：**计算当前大整数除以另一个大整数的余数
- **时间复杂度：** $O(n \cdot (n - m))$
- **异常情况：**若模数为零，抛出异常
- **使用示例：**`a %= b;`

除以大整数的二元运算符 `BigInt operator/(BigInt a, const BigInt& b)`

- **功能：**返回两个大整数的整数商
- **时间复杂度：** $O(n \cdot (n - m))$
- **异常情况：**若除数为零，抛出异常
- **使用示例：**`BigInt c = a / b;`

取模二元运算符 `BigInt operator%(BigInt a, const BigInt& b)`

- **功能：**返回两个大整数相除的余数
- **时间复杂度：** $O(n \cdot (n - m))$
- **异常情况：**若模数为零，抛出异常
- **使用示例：**`BigInt c = a % b;`

使用注意事项

1. **异常处理：**除法和取模操作在除数为零时抛出 `std::runtime_error` 异常
2. **内存管理：**所有操作自动管理内存，无内存泄漏
3. **线程安全：**类本身不是线程安全的，多个线程同时操作同一对象需要外部同步
4. **性能考虑：**
 - 乘法使用朴素算法，对于超大整数可能较慢
 - 频繁创建临时对象可能影响性能
5. **输入验证：**字符串构造函数不验证输入格式，调用者需确保输入合法
6. **溢出处理：**乘以整数时，如果整数过大可能导致中间结果溢出