C++ 高精度减法讲义

1. 思想

既然高精度加法可以通过“反转 -> 补零 -> 逐位相加”的流程简化计算，减法是否也可以呢?

答案是肯定的！通过**完全相同的预处理步骤**，我们同样可以将减法问题转化为一个从左到右的、下标对齐的简单循环。

产生了新的问题:

我们已经知道如何用字符串来模拟大数的加法。减法同样可以模拟，但比加法多了三个问题:

- **预处理**：减法必须先判断大小，确定结果的符号。
- **计算**：将进位(`carry`)逻辑替换为借位(`borrow`)逻辑。
- **后处理**：减法的结果可能产生前导零(例如`123-120=003`)，需要特殊处理。

2. 算法步骤详解

判断符号(这是减法独有的)

- 在计算前，必须比较`a`和`b`的大小。
- 创建一个`bool negative = false;`标志位。
- 如果`b`比`a`大，说明结果是负数。我们记下`negative = true;`，然后交换`a`和`b`。
- **目的**：通过这一步，我们保证了接下来的计算永远是`大数 - 小数`，极大地简化了借位逻辑。

反转与补零(与加法完全一致)

- 将`a`和`b`(已经是大数和小数了)同时反转。
- 在较短的字符串**末尾**补`0`，使其与较长的等长。
- **目的**：让原本的“个位、十位、百位...”对齐到字符串的索引`0, 1, 2...`，从而可以使用一个简单的`for`循环从左到右处理。

核心循环——逐位相减
这是算法的核心，我们将在这里实现借位逻辑。

- 创建一个`int borrow = 0;`用于记录借位。
- 循环从`i = 0`开始:
    1. **取数计算**：`diff = (a[i]-'0') - (b[i]-'0') - borrow;`
        - `a[i]`是被减数，`b[i]`是减数。
        - `borrow`是来自**更低位**的借位请求。如果上一轮（`i-1`）发生了借位，`borrow`就是`1`，意味着当前位的`a[i]`已经被借走了一个`1`。
    2. **判断并处理借位**:
        - 如果`diff < 0`，说明当前位不够减。
            - `diff += 10;`(向更高位借来的`1`在本位体现为`10`)。
            - `borrow = 1;`(设置借位标记，告诉**下一轮循环**，它的`a[i+1]`被借走了)。
        - 如果`diff >= 0`，说明够减。

    - `borrow = 0;` (重置借位标记)。
3. **拼接结果**：`ans += to_string(diff);`

去除前导零 (在反转前操作)

- 我们得到的 `ans` 是一个倒序的结果。例如，`123 - 120` 的计算结果 `3`，在 `ans` 中会是 `"300"`。
- 如果直接反转，会得到 `"003"`。
- 所以，我们必须在**最终反转之前**，去除 `ans` **末尾**多余的 `0`。
- **注意**：要保留最后一位 `0` (例如 `10 - 10` 的结果是 `0`)，所以循环条件是 `ans.size() > 1`。

最终反转与添加符号

- 将清理好的 `ans` 字符串反转回来，得到正确的顺序。
- 检查设置的 `negative` 标志位。如果为 `true`，在结果的最前面加上 `"-"`。

3. 完整代码实现

```
// 比较函数，判断 a 是否大于 b
bool isGreater(string a, string b) {
    if (a.length() != b.length()) {
        return a.length() > b.length();
    }
    return a > b; // 长度相同时，直接按字典序比较
}

string sub(string a, string b) {
    // 判断符号，并确保 a >= b
    bool negative = false;
    if (isGreater(b, a)) {
        negative = true;
        swap(a, b);
    }

    // 将两个输入字符串都反转
    reverse(a.begin(), a.end());
    reverse(b.begin(), a.end());

    // 在较短的字符串末尾补 '0'，使其与较长的等长
    while (b.size() < a.size())  b += '0';

    string ans = "";
    int borrow = 0;

    // 核心循环，逐位相减
    for (int i = 0; i < a.size(); i++) {
        // 计算当前位的差值（考虑上一位的借位）
        int diff = (a[i] - '0') - (b[i] - '0') - borrow;

        // 判断是否需要向更高位借位
        if (diff < 0) {
            // 当前位不够减，需要借位
            diff += 10;   // 借来的 1 在本位变成了 10
```

```
            borrow = 1;   // 产生了借位，标记下一轮计算要被借走
        } else {
            // 当前位够减，无需借位
            borrow = 0;   // 重置借位标记
        }
        // 将当前位的结果拼接到 ans 末尾
        ans += to_string(diff);
    }

    // 在反转前，把倒序结果末尾的 0 去掉（处理前导零）
    while (ans.size() > 1 && ans.back() == '0') {
        ans.pop_back();
    }

    // 将最终结果反转回来
    reverse(ans.begin(), ans.end());

    // 如果结果为负，添加负号
    if (negative) ans = "-" + ans;

    return ans;
}
```