同济大学

实验报告

装

订

线

BigInt Calculation

大数计算

1550431 王甯琪 计三

1. 题目及基本要求描述

1.1.问题引入

在 C++ 中， int 型整数能够表示的数据范围有限，其他类型的整数也只能表示有限的数据。如果计算结果超过数据类型能够表示的范围，就会造成溢出，最终导致结果不正确。

那么在进行大数的运算时， C++ 的自带的数据类型是不够用的。所以需要定义新的类 bigint 来解决这个问题。

1.2.要求

需要自定义一个 bigint 类，来表示一个超过 int 类型可表示范围的大数，并且需要进行 bigint 型数据间的计算，并得到正确结果。

需要完成的计算包括：加、减、乘、除、模、正号、负号、赋值、复合赋值、自增、自减、比较、数组以及函数调用等。

bigint 的运算通过 C++ 的运算符重载来实现，并且操作数和结果只限于整数。数制为十进制有富豪数，且需要能够计算不少于1000000位的大数。

可以借用编译器中超过4字节的大整数定义，如 long long int。

在储存 bigint 类型数据时，不可以一次性申请全部空间，但可以适度浪费（10KB 以内）。

同时需要写一个测试和展示用的 main 函数。由于测试数据大，受限于缓冲区大小，可以采用输入输出重定向的方式读入和输出。

可以借用他人的 main 函数，在 cmd 下用 comp 命令比对输出重定向生成的两个文件，从而观察输出是否正确。

2. 整体设计思路

2.1.考虑需要重载的运算符

根据题目要求，需要重载 +、-、*、/、%、+=、-=、*=、/=、%=、=、>、>=、<、<=、== 以及 != 等运算符。

其中需要考虑到不同运算本身的语意，从而对数据进行不同的操作，并返回不同的数据类型。

还需要考虑到怎样的运算过程能够尽可能减少不必要的系统资源耗费。

2.2.bigint 类中整数的储存方式

由于大数运算的特点所决定，数据不能很方便储存在一个已存在的数据类型中。

如果将大数看作 10^9 进制的一种数，每一位可能值为 0－99999999 ，那么每一个位都可以用 long long 型的数来表示。例如，98765432109876543210 = 9876 * (10^9)^2+ 54321098 * (10^9)^1 + 76543210 * (10^9)^0。此时，当某一位与另外一位进行运算时，其所有可能值都在 long long 型可以表达的范围内。例如 99999999 * 99999999 = 9999999800000001， 结果仍然在 long long 型可表区间内。如果考虑进位等的极限情况，仍然可以发现每一位可能得到的运算结果都不会溢出。

2.3.数据读入的问题

读入数据的时候首先不能确定数的位数，所以可以先将数当作字符串，将每一个位当作 char 型数据临时储存起来。

在读取完毕后，可以将储存的字符 0 - 9 按 2.2. 中提及的规则转换成 bigint 类型的整数保存起来。

2.4.运算的实现

基本的运算只有加减乘除和比较运算。

其他运算可以通过相似的思路或者直接调用基本的运算来表示。

2.5.需要考虑的其他问题

需要注意的有运算的正负号问题，进位和借位的问题，最高位不能为零等等问题。

3. 主要功能的实现

3.1.bigint 类中整数的输入以及储存

定义一个结构体 temp_saver，在重载 >> 时，会先把输入的数字当作字符储存在以 temp_saver 为结点类型的链表中，读取的同时计数。在读取完毕后根据 bigint 类型中的数据特点（即每一位的数字最多8个），将链表中的字符再转换为数存入 bigint。

为了节省申请新空间的时间，一个 temp_saver 结点最多可以保存 x_length （实际操作时取值100）个字符。

而 bigint 的数据储存部分是一个链表。每一个结点保存 value 和 n 两个数据，代表 10^9 进制下某一位的在整个数中的位置和大小，具体来说，一个结点表示的数为 value＊(10^9)^n，如果一个结点的 value 和 n 分别为 61035947 和 59，则代表这个结点代表的是整个数中第 60 位（最小位的 n 为 0），其大小为 61035947 ＊ (10^9)^59。

bigint 中有成员 sign 记录数的正负号。

3.2.+ 的重载

做加法的两个数需要保证同号，否则执行减法操作。

假设做加法的两个数 b1、b2 的位数为 e_1、e_2，则选两个数中的大的那个，并记为 e_{max}。同时生成一个空的 bigint 对象 b_{new}，其数据储存链表中的结点有 e_{max} 个，每个结点的 value 为 0。

那么可以从低位开始，将 b1 和 b2 对应结点的 value 值加在 b_{new} 对应结点的 value 上。

最后从 b_{new} 的数据储存链表的尾结点开始，判断是否要进位，并依次进位。如果首元结点需要进位，则新增一个结点在首元结点前，并储存进位的值。

3.3.- 的重载

做减的两个数需要保证异号，否则执行加法操作。

假设做加法的两个数 b1、b2 的位数为 e_1、e_2，则选两个数中的大的那个，并记为 e_{max}。同

时生成一个空的 bigint 对象 b_{new}，其数据储存链表中的结点有 e_{max} 个，每个结点的 value 为 0。

那么可以从低位开始，将 b1 和 b2 对应结点的 value 值相减，并把结果赋值给 b_{new} 对应结点的 value。

最后从 b_{new} 的数据储存链表的尾结点开始，判断是否要借位。需要的话，本结点的 value 加上 10^9，前一结点的 value 减去1。依次借位。完成后，首元结点（代表最高位）开始，删除 value 为 0 的结点，直至第一个 value 不为 0 的结点（保证最高位不为0）。

3.4.* 的重载

假设做加法的两个数 b1、b2 的位数为 e_1、e_2，记 $e = e_1 + e_2$。同时生成一个空的 bigint 对象 b，其数据储存链表中的结点有 e 个。

那么可以从 b 的最高位开始，依次对结点 jd 操作，将 b1 和 b2 对应结点 jd1、jd2 的 value 值相乘（ jd1、jd2 满足两节点的 n 值相加等于 jd 的 n 值），并把结果赋值给 jd 结点的 value。（本质上是模拟多项式的相乘。）

最后从 b 的数据储存链表的尾结点开始，判断是否要进位，并依次进位。如果首元结点需要进位，则新增一个结点在首元结点前，并储存进位的值。

3.5./ 和 % 的重载

需要将两个参与运算的数转换为字符串的形式，方便操作。

模拟手工计算，从高到低得到商。

如果是模，则返回余数，如果是除则返回商。

3.6.比较运算符的重载

比较运算可以通过 **比较正负 -> 比较位数（10^9进制） -> 一次比较每一位的大小** 的方式，层层比较，最后得到两个数的关系。

3.7.其他细节部分

需要注意，复制构造函数和赋值构造函数中的数据储存链表不能直接赋值，需要重新申请空间。

在析构函数中也需要释放动态申请的空间。

4. 调试过程碰到的问题

4.1.写代码时忽略基本知识

其实课上讲过， const 修饰的对象、 const 修饰的成员函数以及 mutable 修饰的成员的一些关系。但是实际写代码的时候，总是会出现错误，编译器会有各种提示。

所以在动手之前先熟悉一些基础的知识会比较好，能够避免浪费很多时间。

4.2.链表相串

应为运算符重载中涉及的链表很多，而且代码直接复制比较多，所以几次都发现 bigint 不同对象间的数据相互交换，甚至是丢失数据。

这反映了几个问题：

第一是赋值代码后检查不仔细，没有及时更改代码。

第二是这也说明我的变量命名不够有独特性。从一个加法重载函数中赋值的代码粘贴在减法重载函数中，但是所有加法重载函数中的变量名，减法重载函数也使用了。这样命名变量的习惯对之后写更大的程序可能还是有一定影响。

5. 心得体会

5.1.不能纠结

在写除法重载函数中，我纠结了三四种思路，但是往往是一种没写完或者遇到了困难就删掉换下一种方法。

这样的后果是，耽误了很多时间。我写除法重载函数用的时间超过了整个程序用时的一半。但最后的字符串方式的实现，实际只用了二十分钟不到。

在有很多种方法可以选择的时候，先思考每种方法的实现难度和适合度，有一定的预计后再动手，不能直接上手，某种方法完成80%甚至90%后发现不对然后放弃是不对的。

装 订 线

6. 附件：源程序

```
bigint::bigint(const bigint& bi)
{
	ll i;
	element *e_head, *e_now, *e_next, *p_to_bi = bi.head->next;

	sign = bi.sign;
	elements_amounts = bi.elements_amounts;
	is_infinite = bi.is_infinite;

	if (bi.head == NULL)
	{
		return;
	}

	e_head = new(nothrow) element;
	if (e_head == NULL)
	{
		cout << "动态内存申请错误\n";
		exit(ERROR);
	}
	e_now = e_head;
	head = e_head;

	for (i = 1; i <= elements_amounts; i++)
	{
		e_next = new(nothrow) element;
		if (e_next == NULL)
		{
			cout << "动态内存申请错误\n";
			free_element(e_head);
			exit(ERROR);
		}

		e_next->value = p_to_bi->value;
		e_now->next = e_next;
		e_next->prior = e_now;
		e_now = e_next;
		p_to_bi = p_to_bi->next;
	}

	last = e_now;
}

bigint bigint::operator+ (const bigint& bi)//将正负和数值加减分开考虑
{
	if (sign*bi.sign == 1)//同号
	{
		const bigint& bi_longer = (elements_amounts >
bi.elements_amounts) ? (*this) : bi,
			&bi_shorter = (&bi_longer == &bi) ? (*this) : bi;
		const ll n_max = bi_longer.elements_amounts, n_min =
bi_shorter.elements_amounts;
		ll i, result_of_divide;
		bigint temp(n_max);
		element* r1 = bi_longer.last, *r2 = bi_shorter.last, *rt =
temp.last;//分别指向3个bigint的尾结点

		for (i = 1; i <= n_min; i++)
		{
			rt->value = r1->value + r2->value;

			r1 = r1->prior;
			r2 = r2->prior;
			rt = rt->prior;
		}

		for (i = n_min + 1; i <= n_max; i++)
		{
			rt->value = r1->value;

			r1 = r1->prior;
			rt = rt->prior;
		}

		//考虑进位
		rt = temp.last;
		for (i = 2; i <= n_max; i++)
		{
			result_of_divide = rt->value / base;
			if (result_of_divide != 0)//需要进位
			{
				rt->prior->value += result_of_divide;
				rt->value = rt->value%base;
			}

			rt = rt->prior;
		}//出循环时，rt指向最高位

		//最后考虑最高位进位问题
		result_of_divide = rt->value / base;
		if (result_of_divide != 0)//需要进位
		{
			element *add = new(nothrow) element;
			if (add == NULL)
			{
				cout << "动态内存申请错误\n";
				free_element(temp.head);
				exit(ERROR);
			}

			add->next = rt;
			rt->prior = add;
			head->next = add;
			add->prior = head;
			add->value = result_of_divide;
			rt->value = rt->value%base;
		}//end of进位

		if (sign == 1)
		{
			return temp;
		}
		else
		{
			return -temp;
		}
	}
	else//异号
	{
		//a(正)+b(负) <=> a(正) - (-b)(正)
		//a(负) + b(正) <=> a(负) - (-b)(负)
		//return *this - (-bigint(bi));
		int mark = abs_compare(*this, bi);//mark==1则结果不需要+-
号翻转，否则需要
		if (mark == 0)//*this == bi
		{
			return bigint(ll(0));//0
		}

		const bigint& bi_larger = (mark == 1) ? *this : bi, &bi_smaller
= (mark == -1) ? *this : bi;
		ll n_max = bi_larger.elements_amounts, n_min =
```

```
bi_smaller.elements_amounts, i, counts = 0;
		bigint temp(n_max);
		element* r1 = bi_larger.last, *r2 = bi_smaller.last, *rt = temp.last, *to_delete, *checker;

		//接下来只需要考虑 bi_larger-bi_smaller, 结果一定为正, 结果的符号是+还是-不在这部分计算的考虑范围内
		for (i = 1; i <= n_min; i++)
		{
			rt->value = r1->value - r2->value;

			r1 = r1->prior;
			r2 = r2->prior;
			rt = rt->prior;
		}

		for (i = n_min + 1; i <= n_max; i++)
		{
			rt->value = r1->value;

			r1 = r1->prior;
			rt = rt->prior;
		}

		//考虑借位问题
		rt = temp.last;
		for (i = 1; i <= n_max; i++)
		{
			if (rt->value < 0)
			{
				rt->value += base;
				rt->prior->value -= 1;
			}

			rt = rt->prior;
		}

		//考虑最高位不能为0
		checker = temp.head->next;
		for (i = 1; i <= n_max; i++)
		{
			if (checker->value != 0)
			{
				break;
			}

			counts++;
			to_delete = checker;
			checker = checker->next;
			delete to_delete;
		}
		temp.elements_amounts -= counts;
		temp.head->next = checker;
		checker->prior = head;

		//最后考虑正负
		if (sign*mark == 1)//mark==-1 -> 需要翻转, sign==-1 -> 需要翻转
		{
			return temp;
		}
		else
		{
			return -temp;
		}
	}
}

bigint bigint::operator* (const bigint& bi)
{
	bigint temp(elements_amounts + bi.elements_amounts - 1);
	ll  n_most_t = temp.elements_amounts - 1, n_most_1 = elements_amounts - 1, n_most_2 = bi.elements_amounts - 1, i, j, k, result_of_divide;
	element *p1, *p2, *pt;

	//base^k = base^i * base^j
	//接下来模拟多项式乘法
	p1 = last;
	for (i = 0; i <= n_most_1; i++)
	{
		pt = temp.last;
		for (k = 0; k < i; k++)
		{
			pt = pt->prior;
		}

		p2 = bi.last;
		for (j = 0; j <= n_most_2; j++)
		{
			pt->value = p1->value*p2->value;
			p2 = p2->prior;
			pt = pt->prior;
		}

		p1 = p1->prior;
	}

	//考虑进位
	pt = temp.last;
	for (i = 1; i <= n_most_t; i++)
	{
		result_of_divide = pt->value / base;
		if (result_of_divide != 0)//需要进位
		{
			pt->prior->value += result_of_divide;
			pt->value = pt->value%base;
		}

		pt = pt->prior;
	}//出循环时, pt指向最高位

	 //最后考虑最高位进位问题
	result_of_divide = pt->value / base;
	if (result_of_divide != 0)//需要进位
	{
		element *add = new(nothrow) element;
		if (add == NULL)
		{
			cout << "动态内存申请错误\n";
			free_element(temp.head);
			exit(ERROR);
		}

		add->next = pt;
		pt->prior = add;
		temp.head->next = add;
		add->prior = temp.head;
		add->value = result_of_divide;
		pt->value = pt->value%base;
	}//end of进位

	element*checker, *to_delete;
	ll n_max=temp.elements_amounts-1,counts_=0;
	 //考虑最高位不能为0
	checker = temp.head->next;
	for (i = 1; i <= n_max; i++)
	{
		if (checker->value != 0)
		{
			break;
		}

		counts_++;
		to_delete = checker;
		checker = checker->next;
		delete to_delete;
	}
	temp.elements_amounts -= counts_;
```

```
        temp.head->next = checker;
        checker->prior = head;

        temp.sign = sign*bi.sign;

        return temp;
}

bigint bigint::operator/ (const bigint& bi)
{
        if (bi.head == NULL)//bi==0
        {
                bigint temp(ll(0));
                temp.is_infinite = true;

                return temp;
        }

        int cmp_result = abs_compare(*this, bi);

        if (cmp_result == -1)//|*this|<|bi|
        {
                return bigint(ll(0));//return 0
        }
        else if (cmp_result == 0)//|*this|==|bi|
        {
                bigint temp(ll(1));
                temp.sign = sign*bi.sign;
                temp.head->next->value = 1;

                return temp;//return 1 or -1
        }
        else//|*this|>|bi|
        {
                //用字符串来做做看hhh
                char* l;//l,s -> larger,smaller or longer,shorter
                ll l_length, s_length, i, temp_for_divide, highest_value,
this_value, pos = 0,counts=0,res;
                element* p;
                bigint temp(ll(1));

                //*this部分
                temp_for_divide = base / 10;//10000000
                highest_value = head->next->value;
                for (i = 8; i >= 1; i--)
                {
                        if (highest_value / temp_for_divide != 0)
                        {
                                break;
                        }

                        temp_for_divide /= 10;
                }
                l_length = i;
                l_length += unit_length*(elements_amounts - 1);

                l = new(nothrow) char[unsigned(l_length + 1)];//'\0'
                if (l == NULL)
                {
                        cout << "动态内存申请错误\n";
                        exit(ERROR);
                }

                temp_for_divide = base / 10;//10000000
                highest_value = head->next->value;
                for (i = unit_length; i >= 1; i--)
                {
                        if (highest_value / temp_for_divide != 0)
                        {
                                break;
                        }

                        temp_for_divide /= 10;
                }
                while (temp_for_divide != 0)
                {
                        l[pos++] = char(highest_value / temp_for_divide +
'0');
                        highest_value %= temp_for_divide;
                        temp_for_divide /= 10;
                }

                p = head->next->next;
                while (p != NULL)
                {
                        this_value = p->value;
                        temp_for_divide = base / 10;//10000000
                        for (i = unit_length; i >= 1; i--)
                        {
                                l[pos++] = char(this_value / temp_for_divide
+ '0');
                                this_value %= temp_for_divide;
                                temp_for_divide /= 10;
                        }

                        p = p->next;
                }
                l[pos] = '\0';

                //bi部分
                temp_for_divide = base / 10;//10000000
                highest_value = bi.head->next->value;
                for (i = 8; i >= 1; i--)
                {
                        if (highest_value / temp_for_divide != 0)
                        {
                                break;
                        }

                        temp_for_divide /= 10;
                }
                s_length = i;
                s_length += unit_length*(bi.elements_amounts - 1);

                //cout << l << endl << s << endl;
                //l s correct

                pos = 0;
                bigint temp_bi(ll(1));
                temp_bi.head->next->value = 0;
                for(i=0;i<s_length;i++)
                {
                        temp_bi *= bigint(10);
                        temp_bi += bigint(l[pos++] - '0');
                }
                if (abs_compare(temp_bi, bi) == -1)//temp_bi<bi
                {
                        temp_bi *= bigint(10);
                        temp_bi += bigint(l[pos++] - '0');
                }

                i = 0;
                temp_saver* ts_head, *ts_now, *ts_next;
                ts_head = new(nothrow) temp_saver;
                if(ts_head==NULL)
                {
                        cout << "动态内存申请错误\n";
                        delete[]l;
                        exit(ERROR);
                }
                ts_next = new(nothrow) temp_saver;
                if (ts_next == NULL)
                {
                        cout << "动态内存申请错误\n";
                        delete ts_head;
                        delete[]l;
                        exit(ERROR);
                }
```

装 订 线

```
        ts_now = ts_head;
        ts_now->next = ts_next;

        counts = 0;
        while(1)
        {
                res = 0;
                //cout << temp_bi << ' ';
                while(abs_compare(temp_bi, bi)==1)
                {
                        res++;
                        temp_bi -= bi;
                }
                if(abs_compare(temp_bi, bi) == 0)
                {
                        res++;
                        temp_bi = bigint(ll(1));
                }
                //cout << res << endl;


                if (i == x_length)//一个temp_saver结点可以储存
x_length个字符
                {
                        i = 0;
                        ts_next = new(nothrow) temp_saver;
                        if (ts_next == NULL)
                        {
                                cout << "动态内存申请错误\n";
                                free_temp_saver(ts_head);
                                exit(ERROR);
                        }

                        //申请成功
                        ts_now->next = ts_next;
                        ts_now = ts_next;
                }

                *((ts_next->x) + i) = char(res + '0');
                i++;
                //cout << res << endl;

                if(pos>=l_length)
                {
                        break;
                }

                counts++;
                //正常后移一次
                temp_bi *= bigint(10);
                temp_bi += bigint(l[pos++] - '0');
        }//出循环时 temp_bi 为余数

        //将temp_saver -> bigint
        ll numbers_of_elements, n_counter,
first_element_length,temp_value,k,j;
        bigint result;
        numbers_of_elements = counts / unit_length;
        first_element_length = counts%unit_length;//保证除最高位的
element外，其余每一个element都储存了8个数
        if (first_element_length == 0)
        {
                first_element_length = unit_length;
        }
        else
        {
                numbers_of_elements++;//最高位的element需要一个结
点
        }
        n_counter = numbers_of_elements - 1;

        //result.number_amounts = counts;
        result.elements_amounts = numbers_of_elements;//bigint类数
据的填入

element *e_head, *e_now, *e_next;

e_head = new(nothrow) element;
e_now = new(nothrow) element;
if (e_head == NULL || e_now == NULL)
{
        cout << "动态内存申请错误\n";
        free_temp_saver(ts_head);
        exit(ERROR);
}
e_head->next = e_now;
e_now->prior = e_head;
e_next = e_now;
result.head = e_head;

//先处理最高位的element
ts_now = ts_head->next;
i = 0;
temp_value = 0;
for (j = 1; j <= first_element_length; j++)
{
        temp_value *= 10;
        temp_value += *(ts_now->x + i) - '0';
        i++;
}
e_next->value = temp_value;
e_next->n = n_counter;
n_counter--;

//剩余elements,一共(numbers_of_elements-1)个
for (j = 1; j < numbers_of_elements; j++)
{
        e_next = new(nothrow) element;
        if (e_next == NULL)
        {
                cout << "动态内存申请错误\n";
                free_temp_saver(ts_head);
                free_element(e_head);
                exit(ERROR);
        }

        //申请成功
        e_now->next = e_next;
        e_next->prior = e_now;
        e_now = e_next;

        temp_value = 0;
        for (k = 1; k <= unit_length; k++)
        {
                if (i == x_length)
                {
                        ts_now = ts_now->next;
                        i = 0;
                }

                temp_value *= 10;
                temp_value += *(ts_now->x + i) - '0';
                i++;
        }
        e_next->value = temp_value;
        e_next->n = n_counter;
        n_counter--;
}//出循环时，e_next指向尾节点

result.last = e_next;
result.sign = sign*bi.sign;

element*checker, *to_delete;
ll n_max = result.elements_amounts - 1, counts_ = 0;
//考虑最高位不能为0
checker = result.head->next;
for (i = 1; i <= n_max; i++)
{
```

```
                if (checker->value != 0)
                {
                        break;
                }

                counts_++;
                to_delete = checker;
                checker = checker->next;
                delete to_delete;
            }
            result.elements_amounts -= counts_;
            result.head->next = checker;
            checker->prior = head;

            free_temp_saver(ts_head);
            delete[]l;
            return result;
        }
}
```