Эффективная длинная арифметика

Студент Ли Дмитрий Сенунович
Б9121-09.03.03пикд
Руководитель доцент ИМКТ Кленин Александ Сергеевич

Определение

Длинная арифметика – выполняемые с помощью вычислительной машины арифметические операции над числами, разрядность которых превышает длину машинного слова данной вычислительной машины.

Эти операции реализуются программно, с использованием аппаратных средств работы с числами меньших порядков.

Сферы использования

Криптография.

Математическое и финансовое ПО.

Тема в спортивном программировании.

Арифметические операции

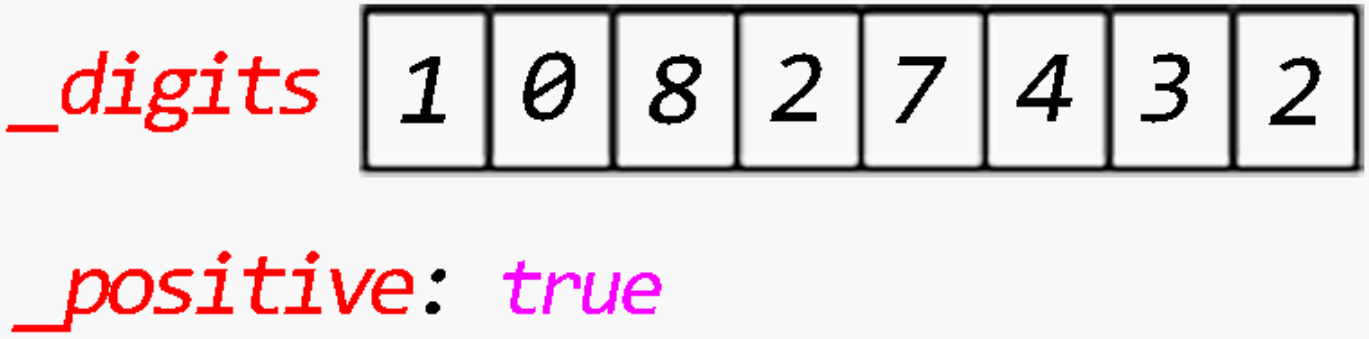
1. Сложение
2. Вычитание
3. Умножение
4. Деление
5. Возведение в степень
6. Сравнение
7. Инкремент и декремент

Хранение чисел

1. Число, представленное в виде динамического массива.
   `vector<int> _digits`

2. Знак числа.
   `bool _positive`

число: 23472108



В случае, если просмотр производится через PDF файл, анимация работать не будет.

Дополнительные функции

1. `void _removeLeftZeros()` – убрать (левые) лидирующие нули.
   Пример: `anyNumber._removeLeftZeros(); //00243 >> 243`

2. `void _afterOperation()` – выполняется после каждой операции.
   Алгоритм: убирает лидирующие нули; проверяет на нуль.

3. `void _doCarryOver(int start = 0)` – перенос переполненных разрядов.
   Используется в `_afterOperation()`

4. `bool _fitsInLongLong()` const – проверяет вмещается число в тип данных `long long` или нет.
   Пример: `anyNumber._fitsInLongLong(); //5345 >> true`

5. `long long _asLongLong() const` – преобразует длинное число в тип данных `long long`.
   Пример: `anyLongLong = anyBigInt._asLongLong();`

Дополнительные функции

6. `size_t _lenght() const` – возвращает длина числа.
   Пример: `anyBigInt._lenght()`

7. `bool _isOdd() const` – проверка на нечетное.
   Пример: `anyBigInt._isOdd() //23 >> true`

8. `bool _isEven() const` – проверка на четное.
   Пример: `anyBigInt._isEven() //23 >> false`

9. `bool _isZero() const` – проверка на нуль.
   Пример: `anyBigInt._isZero() //124 >> false`

10. `bool _isOne() const` – проверка на единицу.
    Пример: `anyBigInt._isOne() //1 >> true`

11. `bool _isPositive() const` – проверка на знак.
    Пример: `anyBigInt._isPositive() //-14253 >> false`

Дополнительные функции

12. `BigInt _times10(int times = 1) const` – добавление нулей (для деления).
    Пример: `anyBigInt._times10(3) //15._times10(3) >> 15000`

13. `BigInt _absoluteValue() const` – возвращает модуль числа.
    Пример: `anyBigInt._absoluteValue() //-124 >> 124`

Ввод числа

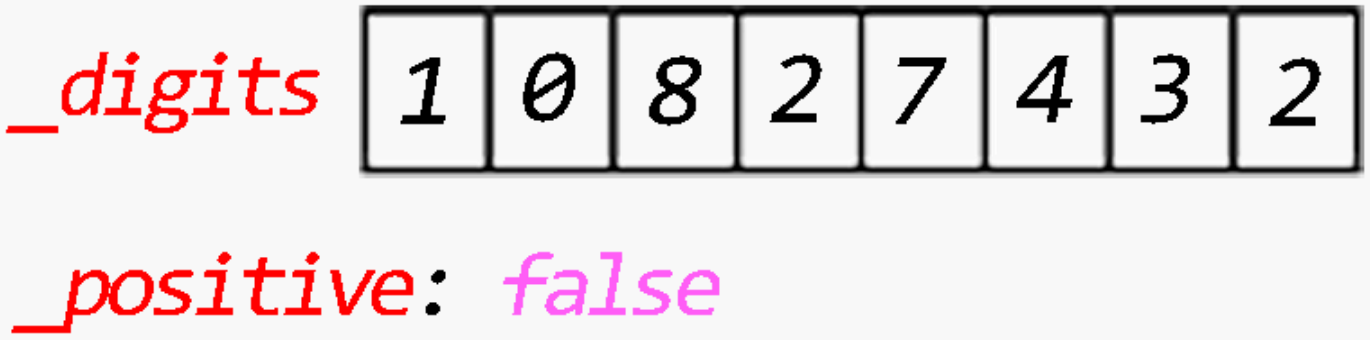
Пример: сохраним число 23 472 801

Число сохраняется в массиве в обратном порядке.

23472801

Вывод числа

Пример: выведем число -23 472 801



Сложение

1	6	4	9	2	5	8	3
+							
9	8	4	0	5	7	9	
=							

```
  38 529 461
+  9 750 489
------------
  48 279 950
```

n3 = max(n2,n1), где
n3 – длина результата
n1 и n2 – длины слагаемых

Слагаемое 1 – a1, слагаемое 2 – a2

Условия:

1. Если a1 или a2 = 0, то возвращается a1 или a2 != 0
2. Иначе если a1 и a2 < 0, то возвращается -(a1+a2)
3. Иначе если a1 или a2 < 0, то возвращается a2-a1 или a1-a2

Вычитание

1	9	4	9	2	4	4	1
9	8	4	0	5	7	9	

```
 14 429 491
-  9 750 489
-------------
   4 679 002
```

Уменьшаемое – a1, вычитаемое – a2

Условия:

1. Если a1 или a2 = 0, то возвращается a1 или a2 != 0
2. Если a2 < 0, то возвращается a1 + a2
3. Если a1 < 0, то возвращается –(a1+a2)

Умножение

1	9	4	9	2	4	4	1	6	2	1
9	8	4	0	5	7	9	4			

12 614 429 491 *
49 750 489
---
627 574 035 633 271 099

Умножаемое – a1, множитель – a2

Так же существуют условия:

1. Если a1 или a2 = 0, то возвращается 0
2. Если a1 или a2 = 1, то возвращается a1 или a2 != 1

Деление

1	9	4	5	2	1	3	2
9	8	4	0	5	7	7	

23 125 491	7 750 489
15 500 978	2
7 624 513	

Делимое – a1, делитель – a2
Целочисленное деление – div, остаток - mod
Так же существуют условия:

1. Если a2 = 0, то выводится ошибка
2. Если a2 = 1, то возвращается div = a1, mod = 0
3. Если a1 = a2, то возвращается div = 1, mod = 0
4. Если a2 > a1, то возвращается div = 0, mod = a1.

Возведение в степень

Операция возведения в степень зависит от умножения.

Число – a1, степень – a2

1. Проверка условий:
    1. Если a1 = a2 = 0, то выводится ошибка
    2. Если a2 < 0, то выводится ошибка
    3. Если a1 = 0, то возвращается 0
    4. Если a2 = 1, то возвращается a1
2. Произвести рекусивное возведение в степень:
    1. Проверить a2 на четное или нечетное.
    2. Если нечетная, то a1*a1*a1 и возвести в степень (a2-1)/2.
    3. Если число четное, то a1*a1 и возвести в степень a2/2.

Сравнение

Для сравнения достаточно двух функций:

- обязательно функцию ==
- на выбор: > или <

Затем встроенными операциями получится создать оставшиеся.

Число 1 – a1, число 2 – a2

Алгоритм сравнения:

1. Если a2 < 0 и a1 > 0, то a1 > a2
2. Иначе если длина a1 < длины a2, то a2 > a1
3. Если длинны равны, то сравнивается каждый разряд до тех пор, пока не найдется больший разряд.
4. В случае если каждый разряд равен, то a1 = a2.

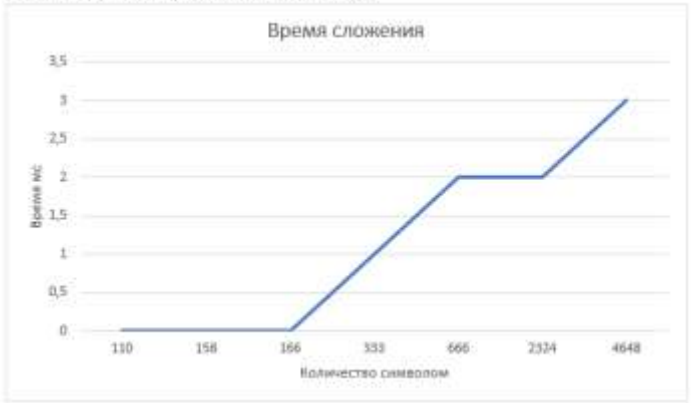
Инкремент и декремент

Функции инкрементирования и декрементирования реализуются:

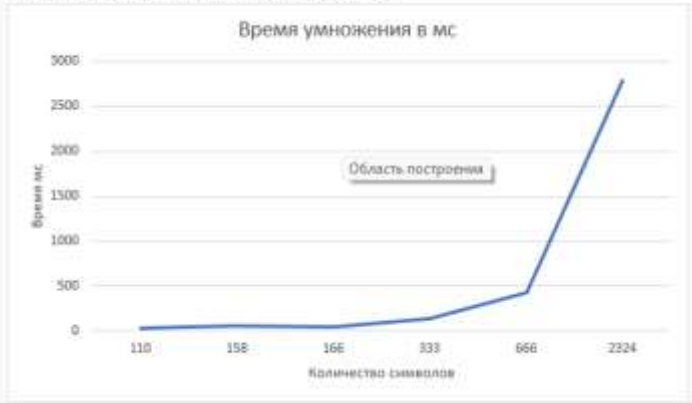
- Добавлением 1 (функция сложения)
- Вычитанием 1 (функция вычитания)

Анализ производительности

Тест скорости работы сложения

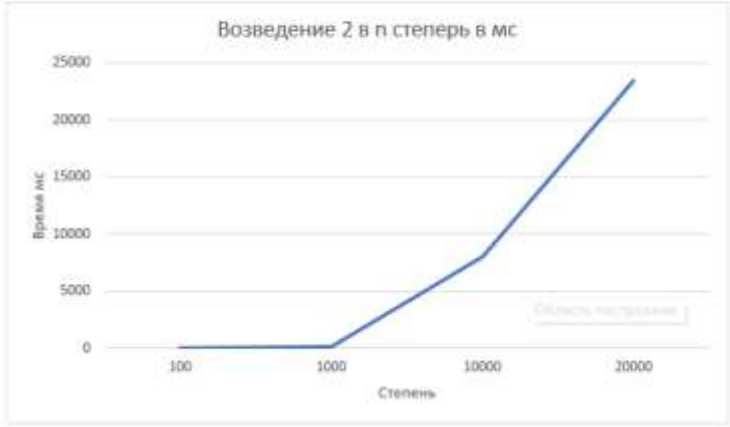


Тест скорости работы умножения

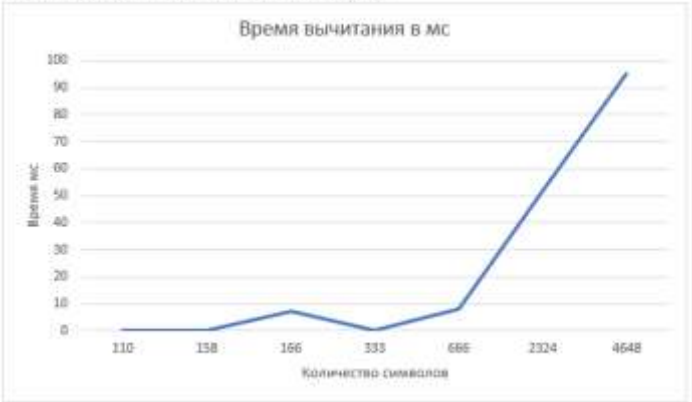


Тест скорости работы возведения в степень

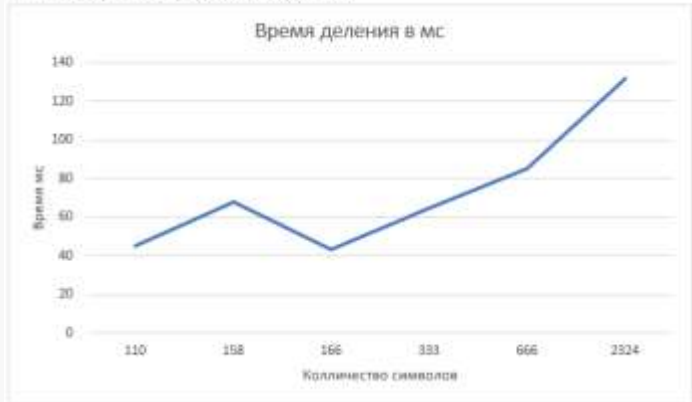
Изначально рассматривается возведение 2 в n степень.



Тест скорости работы вычитания



Тест скорости работы деления



Все тесты скорости работы предоставлены в реферате

Заключение

После изучения большого количества материала на тему «эффективная длинная арифметика»:

Было произведено сжатие и структурирование данных.

Были реализованы операции:

- Сложения
- Вычитания
- Умножения
- Деления (Целочисленное и получение остатка)
- Возведение в степень
- Сравнения

Также было проведено большое количество тестов производительности и проверки работы каждой функции.

Состав репозитория GitHub

В репозиторий было добавлено:

1. Реферат об алгоритме
2. Исходный код алгоритма
3. Тестирующая система
4. Презентация

Ссылка на GitHub: https://github.com/Elsium/ASD