МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное учреждение высшего образования
«Дальневосточный федеральный университет»
(ДВФУ)

ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ

Департамент компьютерного и математического моделирования

ДОКЛАД

Эффективная длинная арифметика

по образовательной программе подготовки бакалавров

по направлению 09.03.03 «Прикладная информатика»

Студент группы № Б9121-09.03.03пикд-5
_____ Ли Д. С.

(подпись)

«_____» _____ 2022г.

г. Владивосток

2022

Оглавление

Введение

Известно, что арифметические действия, выполняемые компьютером в ограниченном числе разрядов, не всегда позволяют получить точный результат. Более того, существуют ограничения – размер чисел, с которыми возможно работать.

Если необходимо выполнить арифметические действия над очень большими числами, например 30! = 265252859812191058636308480000000. То в таких случаях необходимо позаботиться о представлении больших чисел в машине и о точном выполнении арифметических операций над ними.

Числа, для представления которых в стандартных компьютерных типах данных не хватает количества двоичных разрядов, называются «длинными». Реализация арифметических операций над такими «длинными» числами получила название «Длинной арифметики».

«Длинная арифметика» - в вычислительной технике операции (сложение, умножение, вычитание, деление, возведение в степень и т.д.) на числами, разрядность которых превышает длину машинного слова данной вычислительной машины. Эти операции реализуются не аппаратно, а программно, используя базовые аппаратные средства работы с числами меньших порядков.

Проблема: существует класс задач, которые нельзя решить с помощью стандартных типов данных.

Цель: изучение метода «Длинная арифметика».

Данная исследовательская работа посвящена «Длинной арифметике» и её реализации на языке С++».

Теоретическая часть

Стандартные типы данных

Рассмотрим основные целочисленные типы данных языка C++ и диапазон их значений (табл.1)

Таблица 1 [Источник]

short	От -32 768 до 32 767
unsigned short	От 0 до 65 535
int	От -2 147 483 648 до 2 147 483 647
unsigned int	От 0 до 4 294 967 295
long	От -2 147 483 648 до 2 147 483 647
unsigned long	От 0 до 4 294 967 295
long long	От -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807
unsigned long long	От 0 до 18 446 744 073 709 551 615

Из таблицы 1 следует, что наибольшее число, которым мы можем оперировать это 18 446 744 073 709 551 615 или просто 2^{64}-1.

Но число 2^{64} уже не помещается ни в один из представленных типов данных. Для расчёта «длинных» чисел» потребуется другой метод.

Определение

Общее

Длинная арифметика – набор алгоритмов для поразрядной работы с числами произвольной длины. Она применяется как с относительно небольшими числами, превышающими ограничения типа long long в несколько раз, так и с по-настоящему большими числами (чаще всего до 10^{100000}).

Для работы с «длинными числами их разбивают на разряды. Размер разряда может быть произвольным, но чаще всего используются следующие:

- 10 – по аналогии с цифрами числа в десятичной системе, для простоты понимания и отладки.
- 10^4 – наибольшая степень десятки, квадрат которой не превышает ограничения типа int. Используется для максимальной эффективности при хранении разрядов как чисел типа int.
- 10^9 – аналогично предыдущему пункту, но для типа long long. Позволяет достичь максимально возможной эффективности.

(Ограничения на квадрат размера разряда связанны с необходимостью перемножать между собой разряды. Если квадрат разряда превышает ограничение своего типа, при умножении возможны переполнения.)

В большинстве реализаций разряды хранятся в порядке, обратным привычному для упрощения работы с ними. Например, число 578002300 при размере разряда 10^4 представляется следующим массивом:

$$\{2300,7800,5\}$$

Количество разрядов числа может быть как ограничено, так и не ограничено, в зависимости от типа используемого контейнера: массива константной длины или вектора.

Класс

Для начала следует определиться, какие переменные в классе нам необходимы.

Из пункта «Общее» следует, что необходимо создать массив, в котором будут храниться разряды «длинного» числа. Для максимальной эффективности следует использовать _int64 для хранение в одном разряде числа с ограничением 10^9. Но для упрощения написания деления и умножения в проекте будем использовать 1 цифру в 1 разряде. Так же поставим цель упросить код, для понимания

Далее требуется определять знак (положительный\отрицательный) у «длинного» числа. Для этого следует добавить переменную типа данных bool. Что в дальнейшем будет означать: при true – положительное, а при false – отрицательное.

Так же следует добавить константу, которая будет отмечать ограничение разряда числа. В данном случае эта константа будет равна 1 000 000 000, что равно 10^9. Но так как цель упростить код, то использовать константу для ограничения числа не нужно. (Так как в 1 разряде 1 цифра)

Все переменные следует отнести к доступу private, для ограничения доступа вне класса, так же в доступ private попадают функции, которые пользователь не должен вызывать. Так же отдельно они будут помечены: название функций будет начинаться с «_».

Сложение

«Длинную арифметику» часто сравниваю с детским вычислением «в столбик». Это достаточно справедливо, так как оба метода основаны на поразрядных операциях.

Перед сложением необходимо проверить следующие условия:

- Первое число отрицательное, второе положительное: в этом случае достаточно отнять из второго числа первое, поменяв знак первого.
- Первое число положительно, второе отрицательное: в этом случае достаточно отнять из первого числа второе, поменяв знак второго.
- Оба числа отрицательные: нужно сложить модули чисел, а затем поменять знак.

Учитывая вышеописанные условия в алгоритм вычитание могут попасть числа только положительные. Для примера будем использовать массив с ограничением размера разряда 10^4 (int), обратный порядок хранения разрядов. Сложим числа:
994 825 743
  75 183 847

Рисунок 1

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

На рисунке 1 показан способ хранения чисел в массиве. Далее сложим первый разряд.

Рисунок 2

3	4	7	5	2	8	4	9	9
7	4	8	3	8	1	5	7	
10								

На рисунке 2 показан результат сложения первого разряда. Сложим остальные разряды.

Рисунок 3

3	4	7	5	2	8	4	9	9
7	4	8	3	8	1	5	7	0
10	8	15	8	10	9	9	16	9

На рисунке 3 показан результат сложения остальных разрядом. Можем заметить, что есть разряды, в котором количество цифр превышает задуманное. Проведем операцию переноса лишних разрядов.

Рисунок 4

3	4	7	5	2	8	4	9	9	
7	4	8	3	8	1	5	7	0	
10 - 10 = 0	8 + 1 = 9	15 - 10 = 5	8 + 1 = 9	10 - 10 = 0	9 + 1 = 10 10 - 10 = 0	9 + 1 = 10 10 - 10 = 0	16 + 1 = 17 17 - 10 = 7	9 + 1 = 10 10 - 10 = 0	0 + 1 = 1

На рисунке 4 представлен полный результат сложение двух вышеописанных чисел. Он равен 1 070 009 590. Нужно заметить, что лишнее число в разряде переносится в следующий разряд, если в сложении получается еще один лишний разряд, то действие повторяется. В конце создаем дополнительную ячейку для разряда, так как в предыдущем количество цифр превысило допустимое.

Таким образом в дальнейшем будет реализовано сложение.

Вычитание

Вычитание реализуется симметрично сложению. Так, как и сложение, происходит в столбик.

Перед вычитанием необходимо проверить следующие условия:

- Второе число отрицательное: достаточно сложить модули двух чисел.
- Первое число отрицательное: сложить модули двух чисел, затем поменять знак результата.
- Первое число меньше второго: отнять из второго числа первое, затем поменять знак результата.

Учитывая вышеописанные условия в алгоритм вычитание могут попасть числа только положительные и гарантированно первое число будет больше второго. Условия для примера вычитания будут аналогичны сложению. Произведем вычитание чисел:
543 544 825
390 241 523

Рисунок 5

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

На рисунке 5 показано расположение. Произведем вычитание первого разряда.

Рисунок 6

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

На рисунке 6 результат вычитания первых разрядов. Произведем вычитание остальных разрядов.

Рисунок 7

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

На рисунке 7 результат вычитания всех разрядов. Можно заметить, что один из разрядов получился отрицательным. Следует занять единицу у следующего разряда.

Рисунок 8

5	2	8	4	4	5	3	4	5
3	2	5	1	4	2	0	9	3
2	0	3	3	0	3	3	-5 + 10 = 5	2 - 1 = 1

На рисунке 8 показано занятие 1 у разряда выше. Завершим вычитание. Вычитание завершено. Разница равна 153 303 302

Таким образом в дальнейшем будет реализовано вычитание.

Умножение

Реализаций умножения существует множество. Но самая эффективная – алгоритм Карацубы. Его сложность $O(N^{1.58})$ [источник]. Что превосходит обычное умножение в стобик ($O(N^2)$). Алгоритм Карацубы основан на парадигме «разделяй и властвуй».

Перед умножением необходимо проверить следующие условия:

- Если одно из чисел равно 0, то вернуть 0
- Если одно из чисел равно 1, то вернуть другое число

Вначале нужно проверить длину входящих чисел, если они меньше 10 (оптимальный вариант длинны для разделения метода карацубы и обычного умножени), то преобразуем числа к типу данных longlong и производим обычное умножение. В ином случае – используем алгоритм Карацубы.

Дальше разделяем числа на половину длины наибольшего. Рекурсивно перемножаем соответствующие части чисел в отдельные переменные (X, Y). Также перемножаем суммы соответствующих частей в отдельную переменную (S).

Далее создаем 3 переменные, в которые будут добавляться нули в зависимости от их разделения по разрядам. (Максимум нулей, половина нулей, ноль нулей).

Складываем 3 созданные переменные. (Алгоритм действует рекурсивно, до длинны числа меньше 10).

Деление и остаток

Производится в отдельной функции, которая возвращает pair<BigInt,BigInt>, где first – результат целочисленного деления, а second – остаток от деления.

Перед делением необходимо проверить следующие условия:

- Если делитель равен 0, то вывести ошибку.
- Если делитель равен 1, то вернуть:
  - Как результат целочисленного деления – делимое.
  - Как остаток от деления – 0.
- Если делимое и делитель равны, то вернуть:
  - Как результат целочисленного деления – 1.
  - Как остаток от деления – 0.

- Если делитель больше делимого, то вернуть:
    - Как результат целочисленного деления – 0.
    - Как остаток от деления – делимое.

Сразу производит деление чисел, которые возможно преобразовать в longlong с помощью встроенного деления в ЯП.

Если не получилось преобразовать в longlong, то создает 2 переменные типа BigInt – mod и div, остаток от деления и результат деления соответственно, где mod изначально равен делимому. Так же создает абсолютное значение делителя (без знака).

Далее в цикле подставляет делитель под высшие разряды делимого с помощью функции добавления нулей. Производит вычитание из переменной mod, до тех пор, пока mod больше делителя. При каждом вычитании из mod прибавляет к переменной div значение, к которому добавили такое же количество нулей, что и в делитель. И повторяет цикл.

На выходе получается 2 числа упомянутых выше. Которые соответственно возвращаются.

Возведение в степень

Рекурсивно умножает число само на себя, до получения нужной степени.

Так же прежде тем, чтобы выполнить возведение в степень проверяются условия:

- Если число и степень равны 0, то выводит ошибку.
- Если степень отрицательная, то выводит ошибку.
- Если число равно 0, то возвращает 0.
- Если степень равна 0, то возвращает 1.

Сравнение

Для описания функций, всех операторов сравнения, достаточно описать оператор равенства и один из операторов больше или меньше.

Рассмотрим равенство: для начала проверим различие знаков. Далее проверим числа на нули. После этого проверим числа по длине, если они не равны, то числа не равны. Дальше сравним каждый разряд, если какой-то из разрядов не равен другому, то числа не равны. В ином случае числа равны.

Рассмотрим операцию меньше: для начала проверим равенство, благодаря описанному выше алгоритму. Далее проверим различие знаков «длинных» чисел. Далее сравним длину чисел и, наконец, сравним поразрядно.

Все остальные операции сравнения (!=, >, <=, >=) реализуются с помощью уже написанных операций.

Инкрементирование

Достаточно с помощью сложения вернуть значение +1. (При условии ++x).

Реализация x++: увеличивание на 1, затем возвращение значения -1.

Декрементирование

Аналогично инкрементированию, описанного выше.

Дополнительные функции

- Функция изменения знака: копирует текущее «длинное» число, меняет переменную, которая хранит в себе знак, на противоположную себе.
- Функция преобразования в строку: использует библиотеку stringstream для создания строки в потоке.
- Абсолютное значение: избавляется от отрицательного знака.
- Добавление 0: принимает значение общего количество разрядов, которое нужно получить, затем добавляет нули.
- Разделение числа: разделяет число на 2 части.
- Проверка на вместимость в longlong: проверяет поместится ли число в тип данных longlong.
- Преобразования в longlong: преобразует длинное число в longlong
- Длина числа: возвращает длину длинного числа
- Является четным или нечетным: возвращает true или false в зависимости от числа и выбранной функции.
- Проверка числа на 0 и на 1: возвращает true или false.
- Проверка на положительное значение: возвращает true соответственно.
- Удаление нулей слева: удаляет все первые нули числа.
- Перенос разряда: вспомогательная функция для сложения и вычитания.
- Функция, которая используется после любой операции: удаляет лидирующие нули, затем проверяет число на нуль.

Реализация

Язык программирования и IDE

Выбран язык программирования C++, потому что уровень знания этого ЯП достаточно высокий для реализации длинных чисел.

Выбрана IDE – Visual Studio, потому что использование тестирующей среды в данной IDE является наиболее удобным и простым, за счет того, что IDE Visual Studio сама выстраивает необходимые связи и зависимости.

Структура

Структура содержит в себе:

- Массив чисел, представленный в виде `vector<int> _digits`.
- Переменная типа `bool _positive`, которая хранит в себе:
  - Значение `true`, если число положительное.
  - Значение `false`, если число отрицательное.

Сложение

Реализация сложения занимает всего 18 строчек кода:

- Первые 8 строчек кода являются условиями, описанными в Теоретической части.
- Следующие 9 строчек кода являются самим сложением. Туда входят:
  - Цикл, который проходится по разрядам.
  - Условие, которое проверяет длину чисел, в зависимости от которой либо складывает разряды, либо добавляет новый (в случае разной длины чисел).
- Последние 2 строчки кода выполняют функцию `_afterOperation` и возвращают результат.

Вычитание

Реализация вычитания занимает 30 строчек кода:

- Первые 8 строчек кода являются условиями, описанными в Теоретической части.
- Следующие 2 строчки создают векторы `result` и `smaller`:
- Следующие 6 строчек являются условием, которое определяет меньшее число в переменную `smaller`, а большее в `result`.
- Следующие 11 строчек кода являются вычитанием. Туда входят:
  - Цикл, который проходится по разрядам.
  - Переменная, которая записывает в себя результат вычитания разрядов.
  - Условие, которое при результате меньше нуля проходится по разряда дальше и ищет у какого можно занять разряд.
  - В конце цикла переменная помещается в вектор `result`.

- Последние 3 строчки вычисляют знак результата, выполняют функцию `_afterOperation` и возвращают результат.

Умножение

Реализация умножения занимает всего 18 строчек кода:

- Первые 3 строчки кода являются условиями, описанными в Теоретической части.
- Следующие 2 строчки кода проверяют числа на вместимость в `long long`, в случае положительного результата умножают числа встроенными методами.
- Следующие 4 строчки кода вычисляют максимальную длину и делят числа пополам.
- Следующие 3 строчки кода являются рекурсией, где перемножаются соответствующие части чисел, а также перемножаются суммы частей.
- В следующих 2 строчках выравниваются разряды результатов перемножения.
- Последняя строчка возвращает сумму выравненных частей.

Деление и остаток

Реализация деления занимает 25 строчек кода:

- Первые 8 строчек кода являются условиями, описанными в Теоретической части.
- Следующие 4 строчки кода проверяют числа на вместимость в `long long`, в случае положительного результата производят деление и вычисление остатка, затем возвращают их.
- Следующие 3 строчки кода создают переменные:
  - Переменная `mod`, в которой хранится модуль делимого.
  - Переменная `absoluteNumber`, в которой хранится модуль делителя.
  - Переменная `div`, в которой в дальнейшем будет хранится целочисленный результат деления.
- Следующие 6 строчек кода реализуют деление в столбик:
- Последние 4 строчки кода вычисляют знак целочисленного деления, а также выполняют функцию `_afterOperation` для результатов целочисленного деления и остатка от деления, а также возвращают пару чисел.

Возведение в степень

Реализация возведения в степень содержит в себе 12 строчек кода:

- Первые 8 строчек кода являются условиями, описанными в Теоретической части.
- Последние 4 строчки кода являются условием для степени:
  - Если степень четная, то число умножается на себя 3 раза, затем входит в рекурсию, где степень вычисляется по формуле (n-1)/2.
  - Если степень четная, то число умножается на себя, затем входит в рекурсию, где степень вычисляет по формуле n/2.

Сравнение

Реализация ВСЕХ видов сравнения занимает 13 строчек кода.

Дополнительные функции

- Функция вывода числа занимает 2 строчки кода и использует функцию `string`.
- Функция ввода числа через оператор `cin>>` занимает 4 строчки кода.
- Функция `string`, которая преобразует число в тип данных `string`, занимает 6 строчек кода.
- Функция модуля числа занимает 3 строчки кода.
- Функция восстановления разрядов `_times10` занимает 5 строчек кода.
- Функция разделения числа, используемая в умножении, занимает 12 строчек кода.
- Функции занимающие 1 строчку кода:
    - Функция проверки вмещения в `long long` - `_fitsInLongLong`
    - Функция преобразования в `long long` - `_asLongLong`
    - Функция вычисления длины числа - `_lenght`
    - Функции определения четности/нечетности числа - `_isEven`/`_isOdd`
    - Функции проверки на 0 и 1 - `_isZero`/`_isOne`
    - Функция проверки на знак - `_isPositive`
    - Функция удаления первых нулей - `_removeLeftZeros`
    - Функция переноса разряда - `_doCarryOver`
    - Функция выполняемая после каждой операции - `_afterOperation`, которая выполняет функцию `_removeLeftZeros` и в случае проверки на 0 меняет знак на положительный.

Тесты

Предисловие

Тесты проведены с характеристиками пк:

- Процессор: AMD Ryzen 3 3250U with Radeon Graphics 2.60 GHz
- Оперативная память: 12,0 ГБ (доступно: 9,92 ГБ)
- Тип система: 64-разрядная операционная система, процессор x64
- Видеокарта: встроенная

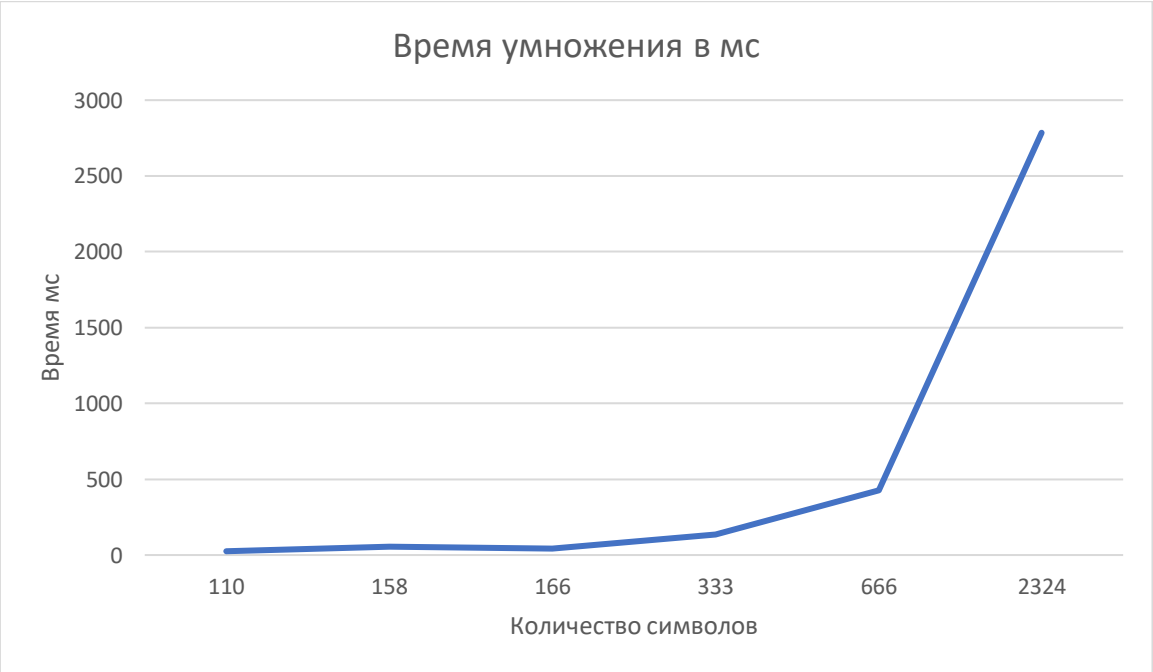
Тест скорости работы сложения

Можно заметить, что сложность повышается резко на 1000 символов до 1мс, дальше сложность растет медленно 2000 символов 2мс, 4000 символов 3мс. Предположительно дальше: 6000 символов 4мс, 8000 символов 5мс.

Тест скорости работы вычитания

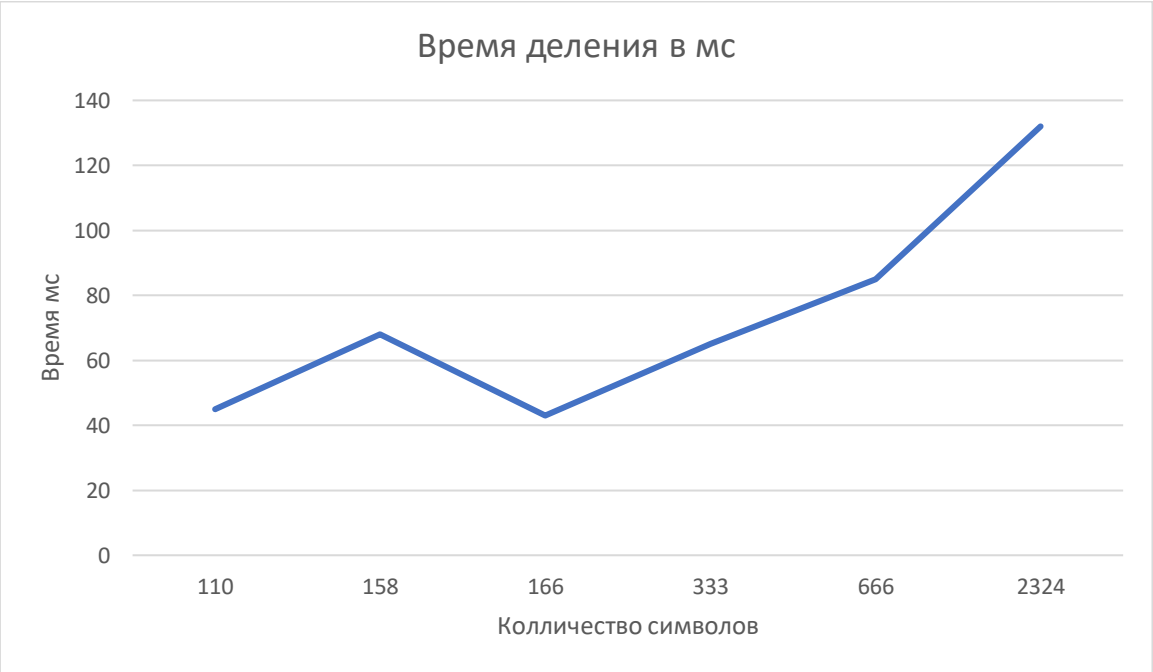
Можно заметить, что сложность растет слишком быстро после 1000 символов. На 2300 символов уже выполняется за 52мс, а на 4500 за 95мс. Предполагая дальнейший рост: 6000 символов 130мс, 8000 символов 160мс.

Тест скорости работы умножения



Можно заметить, что сложность растет на 600 символов уже до 500мс, а на 2300 уже до 2400мс. Предполагая дальнейший рост: 5000 символов 5000мс, 7000 символов 7000мс.

Тест скорости работы деления



Можно заметить, что сложность растет плавно. 666 символов 80мс, 2000 символов 120мс. Предполагая дальнейший рост: 3000 символов 160мс, 4000 символов 200мс.

Тест скорости работы возведения в степень

Изначально рассматривается возведение 2 в n степень.

Можно заметить слишком высокий рост. 10000 степень занимает 8000мс, 20000 степень занимает 23000мс. Предполагая дальнейший рост: 30000 степень 100000мс, 40000 степень 400000мс.

Заключение

В процессе работы над данной темой был создан класс работы с «длинными» числами, написаны функции для основных арифметических операций (сложение, вычитание, умножение, деление, взятие остатка). А так же различные функции для удобства работы с написанным классом.

Так же были проведены тесты корректной работы класса длинных чисел и скорости работы класса.

Источники

1. https://brestprog.by/topics/longarithmetics/
2. https://ru.wikipedia.org/wiki/Длинная_арифметика
3. https://e-maxx.ru/algo/big_integer
4. https://habr.com/ru/post/207754/
5. https://inf2086.ru/crypto_basics/book/algo_long_arithmetic.html
6. https://megaobuchalka.ru/6/33526.html
7. https://topref.ru/referat/50385.html
8. https://intellect.icu/dlinnaya-arifmetika-s-primerami-na-si-8291
9. http://cppstudio.com/post/5036/
10. https://rg-gaming.ru/kompjutery/dlinnaja-arifmetika-c-delenie
11. http://comp-science.narod.ru/DL-AR/okulov.htm
12. https://habr.com/ru/post/124258/
13. https://studfile.net/preview/7014549/page:6/
14. https://forkettle.ru/vidioteka/programmirovanie-i-set/algoritmy-i-struktury-dannykh/73-lektsii-ot-nou-intuit/bazovye-algoritmy-dlya-shkolnikov-lektsii-ot-nou-intuit/572-lektsiya-9-dlinnaya-arifmetika
15. https://moluch.ru/archive/180/46418/
16. https://dic.academic.ru/dic.nsf/ruwiki/1299482
17. https://studassistent.ru/c/dlinnaya-arifmetika-na-si-c-si
18. https://www.pvsm.ru/algoritmy/29587
19. https://itnan.ru/post.php?c=1&p=451860
20. https://habr.com/ru/post/172285/
21. https://pro-prof.com/forums/topic/разность-чисел-длинная-арифметика-си
22. https://lisiynos.github.io/s1/long_ar.html
23. https://habr.com/ru/post/135590/
24. https://inf.1sept.ru/2000/1/art/okul1.htm
25. https://habr.com/ru/post/578718/
26. https://ru.stackoverflow.com/questions/1320123/Оптимизация-длинной-арифметики-с
27. https://www.stud24.ru/programming-computer/dlinnaya-arifmetika/22260-63531-page1.html
28. https://metanit.com/cpp/tutorial/2.3.php