|  |  |
| --- | --- |
|  | МИНПРОСВЕЩЕНИЯ РОССИИ  Федеральное государственное бюджетное образовательное учреждение  высшего образования  «Тульский государственный педагогический университет им. Л.Н. Толстого»  (ТГПУ им. Л.Н. Толстого) |

Кафедра информатики и информационных технологий

**КУРСОВАЯ РАБОТА**

по дисциплине  
 Структуры и алгоритмы компьютерной обработки данных

на тему:

**Программная реализация алгоритмов разбиения натурального числа на слагаемые**

Выполнил:

студент 2 курса группы 121601

факультета математики, физики и информатики

направления «Математическое обеспечение и администрирование информационных систем»

профиля «Информационные системы и базы данных»

Мурашев Лев Владиславович

Руководитель:

доцент, к.п.н., доцент

Мартынюк Юлия Михайловна

Тула 2022

**Содержание**

[Введение 3](#_Toc106067231)

[Глава 1. Теоретические основы разработки алгоритмов разбиения натурального числа на слагаемые 5](#_Toc106067232)

[1.1 Общие понятия классических комбинаторных задач 5](#_Toc106067233)

[1.2 Разбиение натурального числа на слагаемые. 6](#_Toc106067234)

[Глава 2. Разработка программ, реализующих задачи разбиения 9](#_Toc106067235)

[2.1 Реализация класса SplitCounter 9](#_Toc106067236)

[2.2 Вторая реализация класса SplitCounter 10](#_Toc106067237)

[2.3 Генерация разбиений. 13](#_Toc106067238)

[2.4 Получение разбиения по его номеру 16](#_Toc106067239)

[2.5 Получение номера по разбиению. 20](#_Toc106067240)

[2.6 Пользовательский интерфейс и тесты программы. 23](#_Toc106067241)

[Заключение 27](#_Toc106067242)

[Список литературы 28](#_Toc106067243)

# **Введение**

Разбиения чисел на слагаемые естественным образом применяются в ряде математических задач и теорий. Наиболее значимой из них является теория представлений симметрической группы, где разбиения параметризуют все неприводимые представления. Суммы по всем разбиениям натурального числа часто встречаются в задачах математического анализа.

*Разбиение натурального числа* – это такое представление числа в виде суммы положительных целых чисел, которое, в отличие от композиции, не учитывает порядок слагаемых.

Слагаемые в разбиении называются *частями*. В канонической записи разбиения слагаемые перечисляются в невозрастающем порядке.

Целью курсовой работы является изучение алгоритмов для решения основных задач разбиения натурального числа на слагаемые, и их реализация на языке программирования C#.

Цель определяет следующие задачи:

* Рассмотреть общие понятия и теоретические основы разработки алгоритмов разбиения натурального числа на слагаемые.
* Разработать класс для решения задачи о количестве разбиений;
* Разработать класс для решения задачи о генерации всех разбиений;
* Разработать класс для решения задач о получении разбиения по его номеру и номера разбиения по самому разбиению;
* Разработать и протестировать программное приложение, демонстрирующее работу классов.

Работа состоит из двух глав. В первой главе описаны общие понятия и теоретические основы разработки алгоритмов разбиения натурального числа на слагаемые. Во второй главе представлена программная реализация классов и пользовательского интерфейса программного приложения на языке программирования С#.

# **Глава 1. Теоретические основы разработки алгоритмов разбиения натурального числа на слагаемые**

## **1.1 Общие понятия классических комбинаторных задач**

При решении практических задач часто приходится выбирать из некоторого конечного множества объектов подмножество элементов, обладающих теми или иными свойствами, размещать элементы множеств в определенном порядке и т. д. Эти задачи принято называть комбинаторными задачами. Многие из комбинаторных задач решаются с помощью двух основных правил — суммы и произведения. Если удается разбить множество объектов на классы, и каждый объект входит в один и только один класс, то общее число объектов равно сумме числа объектов во всех классах (правило сложения). Правило умножения чуть сложнее. Даны два произвольных конечных множества объектов A и B. Количество объектов в A равно N, в B — M. Составляются упорядоченные пары (a, b), где a∈A и b∈B. Количество таких пар (объектов) равно N×M. Правило обобщается и на большее количество множеств объектов.

***Перестановки***

— число перестановок. Классической задачей комбинаторики является задача о числе перестановок — сколькими способами можно переставить N различных предметов, расположенных на N различных местах.

N различных предметов, расположенных на N различных местах, можно переставить N! (N-факториал) = 1\*2\*3\*…\*N способами ( =N!).

***Размещения***

*.* — число размещений из N по M

Задача формулируется следующим образом: сколькими способами можно выбрать и разместить по M различным местам M из N различных предметов?

Число размещений вычисляется по формуле

Или

***Сочетания (выборки)***

*.* — число сочетаний из N по M

Другой классической задачей комбинаторики является задача о числе выборок: сколькими способами можно выбрать M из N различных предметов?

Сочетание можно определить и как подмножество из M различных натуральных чисел, принадлежащих множеству чисел из интервала от 1 до N. Число сочетаний вычисляется по формуле

***Разбиения****.*

Требуется подсчитать число разбиений конечного множества предметов S, где |S|=N (количество предметов), на k различных подмножеств , попарно не пересекающихся,

| | = , i=1, 2, …, k и + +…+ = N

Схема рассуждений. можно сформировать способами, , таким образом число разбиений равно произведению подобных множителей, при упрощении формула примет вид,

## **1.2 Разбиение натурального числа на слагаемые.**

Дано натуральное число N. Его можно записать в виде суммы натуральных слагаемых: N= + +…+, где k, , …, больше нуля. Будем считать суммы эквивалентными, если они отличаются только порядком слагаемых. Класс эквивалентных сумм приведенного вида однозначно представляется последовательностями , …, , упорядоченными по невозрастанию.

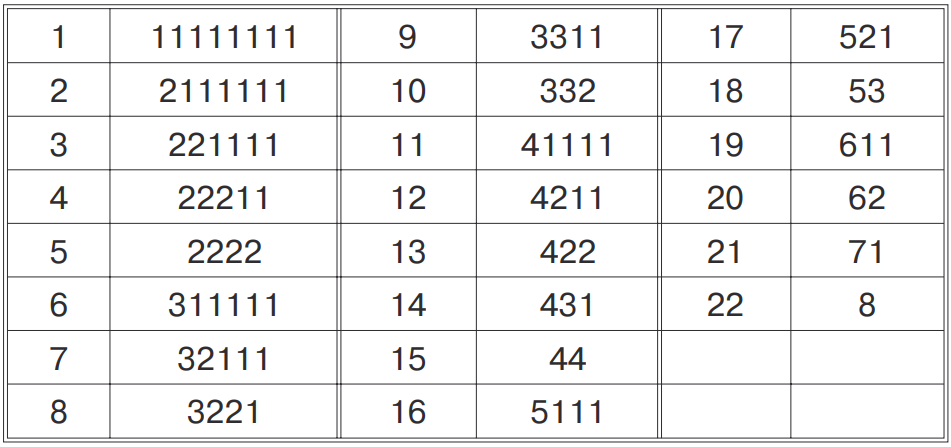
Каждую такую последовательность , …, , назовем разбиением числа N на k слагаемых. Обычно решаются четыре задачи:

1. подсчет количества различных разбиений числа;
2. генерация разбиений относительно определенного отношения порядка, введенного на множестве разбиений;
3. получение разбиения по его номеру;
4. задача, обратная третьей.

*Первая задача.*

Приведен пример для N=8 (см. Табл. 1), а затем рассуждения, позволяющие написать программный код.

Таблица 1. Разбиения числа 8



Число разбиений числа N на k слагаемых обозначим через P(N,k), общее число разбиений — P(N). Очевидно, что значение P(N) равно сумме P(N,k) по значению k. Считаем, что P(0)=P(1)=1. Известен следующий факт.

*Теорема.* Число разбиений числа N на k слагаемых равно числу разбиений числа N с наибольшим слагаемым, равным k.

Теорема доказывается простыми рассуждениями на основе диаграмм Феррерса. Пример таких диаграмм приведен на Рисунке 1

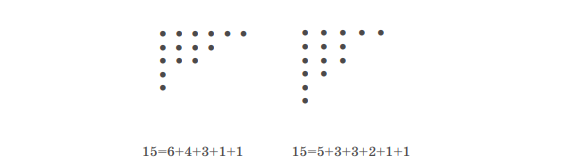


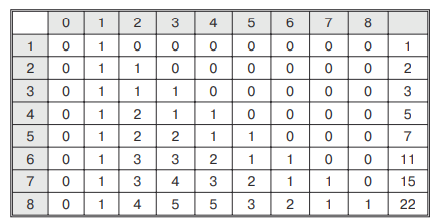
Рисунок 1. Пример диаграмм Феррерса

Главное, что следует из этого факта, — реальный путь подсчета числа разбиений. Число разбиений P(i,j) равно сумме P(i–j,k) по значению k, где k изменяется от 0 до j. Другими словами, j «выбрасывается» из i, и подсчитывается число способов разбиения числа i–j на k слагаемых. Проверим этот алгоритм с помощью таблицы (см Таблица 2).

Например, P(8,3)=P(5,0)+P(5,1)+P(5,2)+P(5,3).

В качестве программной реализации будет статический класс SplitCounter имеющий два метода, GetTable – возвращающий таблицу количества разбиений, на подобии таблицы 2, только заданного размера, и GetCount –возвращающий число разбиений для заданного числа, внутри себя генерируя таблицу вызывая метод GetTable, и считая сумму значений из нужной строки.

Таблица 2. Количество разбиений чисел (первый столбец) на число слагаемых (первая строка)



# **Глава 2. Разработка программного приложения, реализующего задачи разбиения**

Программное приложение демонстрирует работу классов, решающих задачи разбиения. Программа реализована на языке программирования С#:

## **2.1** **Реализация класса SplitCounter**

namespace splitting

{

static class SplitCounter//Статический класс SplitCounter

{

static public int[,] GetTable(int n)//Объявление метода GetTable, метод принимает аргументом целочисленный параметр n, отвечающий за размер таблицы

{

int[,] data = new int[n+1, n+1]; //Инициализация двумерного целочисленного массива data, в нем будет храниться таблица

for (int i = 0; i <= n; i++)//Массив заполняется нулями

{

for (int j = 0; j <= n; j++)

{

data[i, j] = 0;

}

}

data[0, 0] = 1;//Для корректного заполнения массива элементу с индексами 0,0 на время присваиваем значение 1

for (int i = 1; i <= n; i++)//Реализуем вышеописанный алгоритм

{

for (int j = 1; j <= i; j++)

{

for (int k = 0; k <= j; k++)

{

data[i, j] += data[i - j, k];//Массив заполняется по схеме из примера P(8,3)=P(5,0)+P(5,1)+P(5,2)+P(5,3).

}

}

}

data[0, 0] = 0; //Возвращаем верное значение элементу с индексами 0,0, ведь не существует разбиения числа 0 на 0 элементов.

return data; // Метод возвращает сгенерированный массив

}

static public int GetCount(int n) //Объявление метода GetCount, метод принимает аргументом целочисленный параметр n, отвечающий за число для которого будет рассчитано количество разбиений

{

int[,] data = GetTable(n);//Инициализация массива с помощью GetTable

int count = data[n,1];//Переменную, в которой будет храниться количество разбиений, инициализируем первым значащим элементом заданной строки

for (int i = 2; i <= n; i++)//Начиная со второго значащего элемента и до конца строки, к переменной count прибавляется значение следующего элемента строки

{

count += data[n, i];

}

return count;//Метод возвращает посчитанное количество разбиений

}

}

}

Проблема данной реализации состоит в том, что такой подход совсем не раскрывает потенциал предвычислений, ведь для каждого нового числа таблица будет генерироваться заново. Для решения этой проблемы можно изменить архитектуру класса таким образом, чтобы при инициализации экземпляра по заданному значению генерировалась таблица, которая будет хранится в экземпляре, как поле, на протяжении всего существования. В таком случае при инициализации экземпляра проведется своего рода предвычисление, а после вызова у объекта метода GetCount, он будет обращаться к уже вычисленной таблице и работать за О(n) операций. Единственным минусом такого подхода выступает тот факт, что нужно заранее знать, до какого максимального значения генерировать таблицу.

## **2.2 Вторая реализация класса SplitCounter**

namespace splitting

{

class SplitCounter2

{

int[,] data;//Поле data инкапсулирует двумерный целочисленный массив, представляющий таблицу количества разбиений

public int MaxValue { get; private set; } //Поле MaxValue инкапсулирует размер таблицы, закрытый setter позволяет избежать непредвиденного изменения этого значения

public SplitCounter2(int maxValue)//Конструктор класса повторяет логику метода GetTable

{

MaxValue = maxValue;

data = new int[MaxValue + 1, MaxValue + 1]; //Инициализация двумерного целочисленного массива data, в нем будет храниться таблица

for (int i = 0; i <= MaxValue; i++)//Массив заполняется нулями

{

for (int j = 0; j <= MaxValue; j++)

{

data[i, j] = 0;

}

}

data[0, 0] = 1;//Для корректного заполнения массива элементу с индексами 0,0 на время присваиваем значение 1

for (int i = 1; i <= MaxValue; i++)//Реализуем вышеописанный алгоритм

{

for (int j = 1; j <= i; j++)

{

for (int k = 0; k <= j; k++)

{

data[i, j] += data[i - j, k];// Массив заполняется по схеме из примера P(8,3)=P(5,0)+P(5,1)+P(5,2)+P(5,3).

}

}

}

data[0, 0] = 0; //Возвращаем верное значение элементу с индексами 0,0, ведь не существует разбиения числа 0 на 0 элементов.

}

public int GetCount(int N)//Метод отличается от первой версии только тем, что он обращается к уже заполненной таблице и проверят аргумент, чтобы он не выходил за границы массива

{

if (N <= MaxValue)

{

int count = data[N, 1];//Переменную, в которой будет храниться количество разбиений, инициализируем первым значащим элементом заданной строки

for (int i = 2; i <= N; i++)//Начиная со второго значащего элемента и до конца строки, к переменной count прибавляется значение следующего элемента строки

{

count += data[N, i];

}

return count;//Метод возвращает посчитанное количество разбиений

}

else

return -1;

}

}

}

Теперь можно проверить, насколько быстрее будет работать вторая реализация, если по условию задачи требуется посчитать больше одного-двух значений.

Для этого была написана простая программа, которая будет фиксировать время выполнения фрагмента кода, в котором будет подсчитываться количество разбиений всех чисел от одного до 101.

class Program

{

static void Main(string[] args)

{

var sw = new Stopwatch();

Console.WriteLine("Начинается подсчет значений с помощью статической версии класса");

sw.Start();

for (int i = 1; i < 100; i++)

{

SplitCounter.GetCount(i);

}

sw.Stop();

Console.WriteLine($"Времени затрачено: {sw.ElapsedMilliseconds} ms. ");

sw = new Stopwatch();

Console.WriteLine("Начинается подсчет значений с помощью улучшенной версии класса");

sw.Start();

SplitCounter2 sc = new SplitCounter2(101);

for (int i = 1; i < 100; i++)

{

sc.GetCount(i);

}

sw.Stop();

Console.WriteLine($"Времени затрачено: {sw.ElapsedMilliseconds} ms. ");

}

}

Результат работы программы представлен на Рисунке 2.

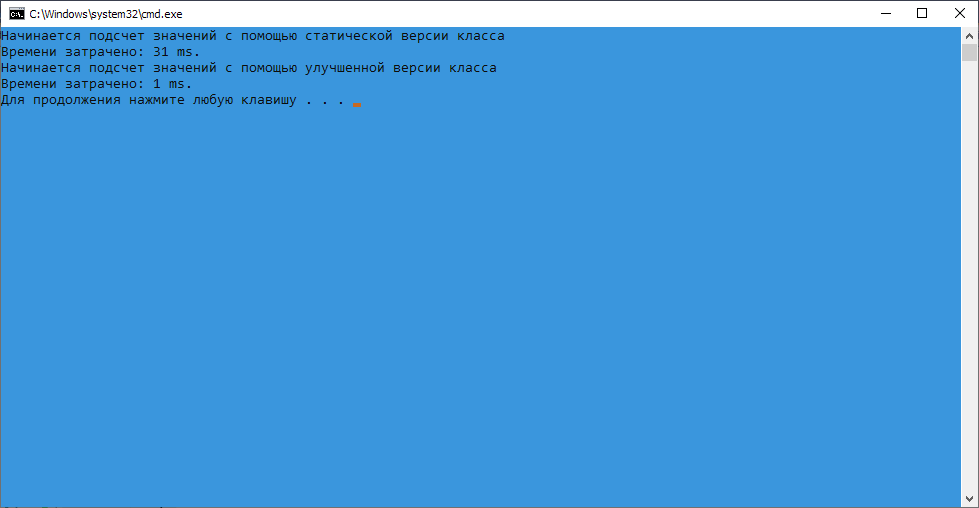


Рисунок 2. Результат работы программы сравнения времени выполнения

На рисунке 2, видно, что даже на таких маленьких значениях решение с предвычислением выигрывает по времени очень сильно. Очевидно, что предвычисления играют важную роль в этом алгоритме. Однако статическая версия класса удобна, когда нужно один раз посчитать количество разбиений и не хочется для этого создавать экземпляр, а потом вызывать у него метод.

Это пригодится, например, при генерации всех разбиений для одного числа.

## **2.3 Генерация разбиений.**

Для генерации разбиений необходима структура данных для хранения разбиения. Это будет целочисленный массив Now, в нулевом элементе будет храниться длина разбиения, т. е. число задействованных элементов массива Now. Возьмем из Таблицы 1 одно из разбиений числа 8, например «41111», следующее «4211». В каком случае можно увеличить Now[2]? Видно, что Now[1] должно быть больше Now[2], т. е. в текущем разбиении находится «скачок» — Now[i–1]>Now[i], Now[i] увеличивается на единицу, а все следующие элементы разбиения берутся минимально возможными. Всегда ли возможна данная схема изменения разбиения? Например, разбиение «44», следующее — «5111». Таким образом, или «скачок», или i равно 1. Кроме того, изменяемый элемент не может быть последним, ибо увеличение без уменьшения невозможно.

В качестве программной реализации выступает класс Splitting (разбиение).

Класс имеет поле Now – целочисленный одномерный массив, переменную N, отвечающую за число, для которого будут генерироваться разбиения, и метод GetNext, который будет изменять массив Now, записывая в него следующее разбиение по вышеописанной схеме.

Реализация класса Splitting

namespace splitting

{

class Splitting

{

public int[] Now { get; private set; }//Оба поля имеют публичные геттеры и закрытые сеттеры, т.к. воздействия со стороны могут легко нарушить работу класса

public int N { get; private set; }

public Splitting(int n)// Конструктор класса принимает аргументом параметр n - число, для которого будут генерироваться разбиения

{

N = n;

Now = new int[N+1];//Инициализируется массив Now размером N+1, т.к. нулевой элемент служебный

Now[0] = N;

for (int i = 1; i < N; i++)

{

Now[i] = 1;

}

}

public void GetNext()

{

int i, j, sc;

i = Now[0] - 1;//Предпоследний элемент разбиения.

sc = Now[i + 1] - 1;

Now[i + 1] = 0; //\*В sc накапливаем сумму, это число представляется затем минимально возможными элементами.

while(i > 1 && Now[i] == Now[i - 1])//Поиск скачка

{

sc += Now[i];

Now[i] = 0;

i--;

}

i++;//Увеличиваем найденный элемент на единицу

Now[0] = i + sc;

for (j = 1; j <= sc; j++) //Изменяем длину разбиения.

{

Now[j + i] = 1;//Записываем минимально возможные элементы, т.е. единицы.

}

}

public override string ToString()//Поскольку в С# все классы неявно являются потомками базового класса Object, для удобства можно определить метод ToString, который будет возвращать строку, содержащую текущее разбиение

{

string rez = "";

for (int i = 0; i < Now[0]; i++)// от нуля до now[0], т.е. выполнить столько раз, сколько элементов массива задействовано в текущем разбиении

{

rez += $"{Now[i + 1] }";//с помощью конкатенации записываем в строку следующий элемент текущего разбиения

}

return rez;

}

}

}

Код программы для проверки работы класса Splitting

namespace splitting

{

class Program

{

static void Main(string[] args)

{

int N = 8, Nsplit = SplitCounter.GetCount(N);//Переменная Nsplit отвечает за количество разбиений. В этом случае удобно использовать статический класс

Splitting split = new Splitting(N);// Создание экземпляра класса

Console.WriteLine($"Все возможные разбиения для числа {N}:");

for (int i = 0; i < Nsplit; i++)//Выводим текущее разбиение, а после генерируем следующее ровно Nsplit раз

{

Console.WriteLine(split.ToString());

split.GetNext();

}

Console.WriteLine($"Итого количество разбиений равно {Nsplit}.");

}

}

}

Результат работы программы представлен на Рисунке 3.

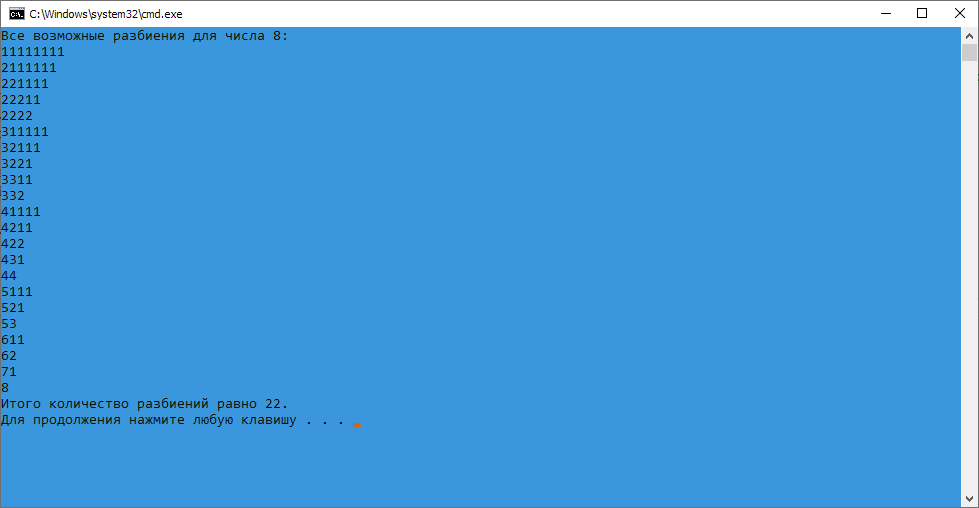


Рисунок 3. Результат работы программы, генерирующей разбиения

На рисунке 3 можно видеть корректность работы программы: все разбиения полностью совпадают со значениями из Таблицы 1.

## **2.4 Получение разбиения по его номеру**

Нумерация разбиений осуществляется относительно введенного отношения порядка. Пусть L равно 0. Приведем набросок алгоритма на языке псевдокода.

...

sc = N;

i = 1;// sc - число, которое представлено разбиением, i – номер позиции в разбиении.

While sc ≠ do // Пока число не исчерпано

Begin

j =1;//Число, которое должно записываться на место i в разбиении.

?????? Значение j определяется по значению L, пока неизвестно как.

Now[i] =j;

sc = sc - j;

i = i+1;//Формируем текущую позицию разбиения и готовимся к переходу для формирование новой.

End;

...

Если вместо знаков «??????» оставить пустое место («заглушку»), то при L, равном 0, данный набросок будет работать — на выходе получится начальное разбиение, состоящее из единиц.

Пусть L равно 1. Обратимся к Таблице 2, проверяем data[8,1]. Если L больше или равно этому значению, тогда нужно уменьшить значение L на data[8,1], увеличивая значение j на единицу и, в конечном итоге, получается разбиение 2111111. Итак, что необходимо сделать? Да просто просмотреть строку массива data с номером sc и определить последний элемент строки, его номер в j, который меньше текущего значения L. Мы последовательно отсекаем разбиения числа sc, начинающиеся с чисел j.

В качестве полноценной программной реализации этого алгоритма будет выступать класс SplitNumber, который в будущем будет дополнен не только методом для получения разбиения по его номеру, но и методом для получения номера разбиения по самому разбиению.

Текущая реализация класса SplitNumber

class SplitNumber

{

int maxvalue;

int N;// Число, для которого будут производиться вычисления

SplitCounter2 sp;//Поле типа SplitCounter2 для получения таблицы data

public SplitNumber(int n)// в конструкторе задается число и проводится предвычисление массива data через инициализацию объекта sp класса SplitCounter2

{

N = n;

sp = new SplitCounter2(n);

maxvalue = SplitCounter.GetCount(N); // maxvalue инициализируется количеством разбиений числа N, для проверки аргумента метода getSplitByNum

}

public int[] getSplitByNum(int Num)//Метод для получения разбиения по номеру Num

{

Num--;//уменьшение Num, т.к. Now уже будет инициализирован первым разбиением

int[] Now = new int[N + 1]; //Инициализируется массив, хранящий разбиение, по той же схеме, как и в классе Splitting, т.е. нулевой элемент хранит количество задействованных элементов массива, остальное – единицы, как первое разбиение любого числа

Now[0] = N;

for (int k = 1; k < Now.Length; k++)

{

Now[k] = 1;

}

if (Num == 0)//В случае, если параметр Num был единицей, т.е. требуется получить первое разбиение, сразу после инициализации возвращается массив Now, а метод прекращает выполнение

return Now;

int i = 1, j, sc = N;//Инициализируются переменные, и выполняется вышеописанный алгоритм

while (sc > 0 || sc < 0)

{

j = 1;

while (Num >= sp.data[sc,j])

{

Num -= sp.data[sc, j];

j++;

}

Now[i] = j;

sc -= j;

i++;

}

Now[0] = i - 1;

return Now; //Метод возвращает полученное разбиение

}

public int getNumOfSplit(int[] split)//Метод еще не реализован

{

return 0;

}

}

Программа для проверки корректности работы алгоритма:

class Program

{

static void Main(string[] args)

{

int n = 8;//Число для которого будут расчитываться разбиения

int m = 6;//Номер разбиения

var sn = new SplitNumber(n);//Инициализация объекта класса SplitNumber

int[] arr = sn.getSplitByNum(m); //Получение шестого разбиения восьмерки

Console.Write($"Разбиение номер {m}, числа {n} :" );

for (int i = 1; i <= arr[0]; i++)//Вывод данных в консоль

{

Console.Write(arr[i]);

}

Console.WriteLine();

}

}

Результат работы программы представлен на Рисунке 4.

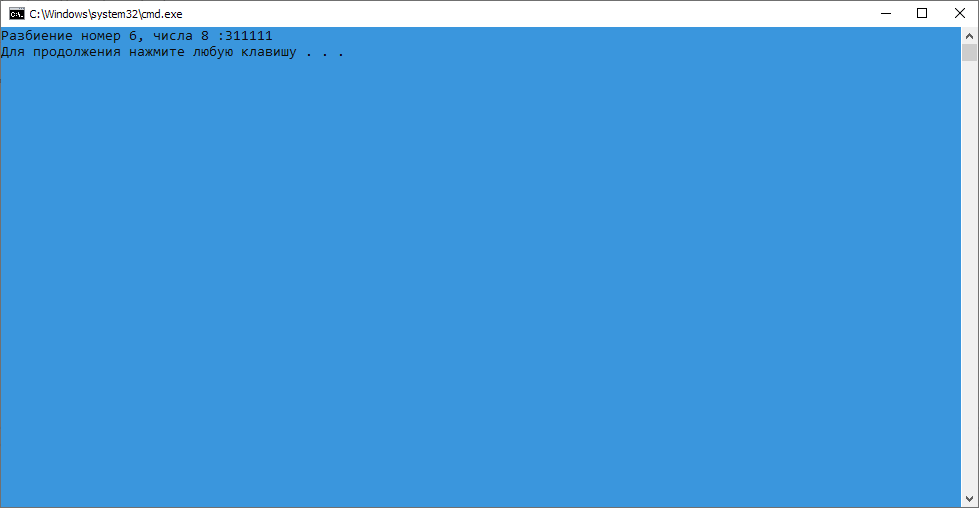


Рисунок 4. Результат работы программы получения разбиения по номеру

Как видно на Рисунке 4, программа отработала без ошибок, разбиение совпадает с разбиением из Таблицы 1.

Возникает вопрос, а почему бы просто не воспользоваться генератором разбиений, получая следующее разбиение ровно Num раз?

Напишем программу, которая позволит узнать, какой способ будет работать быстрее и насколько. Будем фиксировать время, за которое каждым вариантом можно получить тысячное разбиение числа 1000.

Код программы:

class Program

{

static void Main(string[] args)

{

var sw = new Stopwatch();

int n = 1000;//Число, для которого будут рассчитываться разбиения

int m = 1000;//Номер разбиения

var sn = new SplitNumber(n);//Инициализация объекта класса SplitNumber

var sp = new Splitting(n);//Инициализация объекта класса Spliting

Console.WriteLine("Начинается получение разбиения с помощью класса SplitNumber");

sw.Start();

int[] arr = sn.getSplitByNum(m); //Получение нужного разбиения первым способом.

sw.Stop();

Console.WriteLine($"\n Времени затрачено {sw.ElapsedMilliseconds} ms.");

sw = new Stopwatch();

Console.WriteLine("Начинается получение разбиения с помощью класса Spliting");

sw.Start();

for (int i = 0; i <= m; i++)

{

sp.GetNext();

}

sw.Stop();

Console.WriteLine($"\n Времени затрачено {sw.ElapsedMilliseconds} ms.");

}

}

Результат работы программы представлен на Рисунке 5.

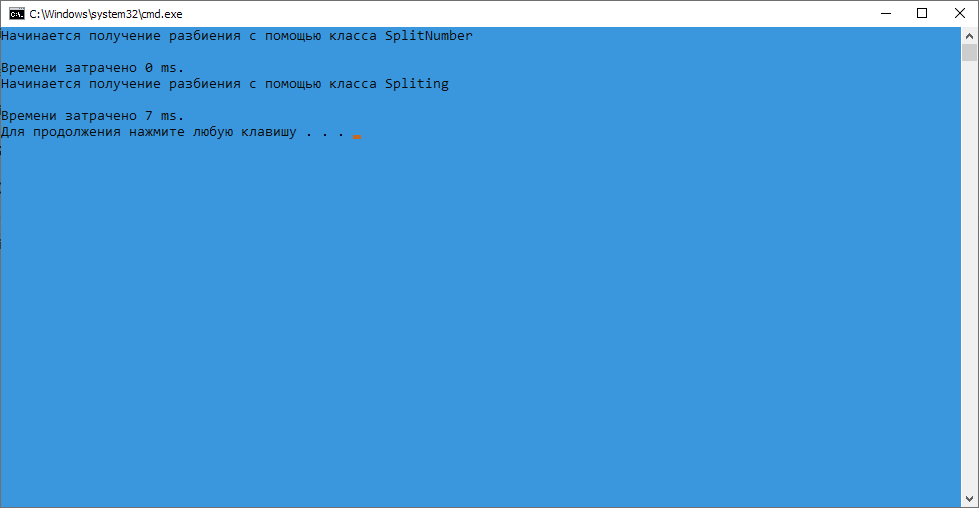


Рисунок 5. Результат работы программы сравнения времени работы SplitNumber и Splitting

Как можно видеть на Рисунке 5, даже на таких маленьких значениях “костыльный” метод через класс Splitting работает в разы медленнее, в то время, как предназначенный для этого класс SplitNumber дает мгновенный результат.

## **2.5 Получение номера по разбиению.**

Идея решения задачи достаточно проста. Следует брать текущее число из разбиения, начиная с первого, и смотреть на его «вклад» в количество номеров.

Например, дано разбиение «22211». Определяем, сколько номеров приходится на первую двойку, затем на вторую и т. д.

Этот метод будет частью класса SplitNumber.

Код доработанного класса:

namespace splitting

{

class SplitNumber

{

int maxvalue;

int N;// Число для которого будут производиться вычисления

SplitCounter2 sp;//Поле типа SplitCounter2 для получения таблицы data

public SplitNumber(int n)// в конструкторе задается число и проводится предвычисление массива data через инициализацию объекта sp класса SplitCounter2

{

N = n;

sp = new SplitCounter2(n);

maxvalue = SplitCounter.GetCount(N);// maxvalue инициализируется количеством разбиений числа N, для проверки аргумента метода getSplitByNum

}

public int[] getSplitByNum(int Num)//Метод для получения разбиения по номеру Num

{

if (Num < 0 || Num > maxvalue)

return new int[1] {-1};//Если аргумент меньше нуля или больше возможного количества разбиений, прекращается работа метода.

Num--;//уменьшение Num, так как Now уже будет инициализирован первым разбиением

int[] Now = new int[N + 1]; //Инициализируется массив, хранящий разбиение, по той же схеме, как и в классе Splitting, т.е. нулевой элемент хранит количество задействованных элементов массива, остальное – единицы, как первое разбиение любого числа

Now[0] = N;

for (int k = 1; k < Now.Length; k++)

{

Now[k] = 1;

}

if (Num == 0)//В случае, если параметр Num был единицей, т.е. требуется получить первое разбиение, сразу после инициализации возвращается массив Now, а метод прекращает выполнение

return Now;

int i = 1, j, sc = N;//Инициализируются переменные, и выполняется вышеописанный алгоритм

while (sc > 0 || sc < 0)

{

j = 1;

while (Num >= sp.data[sc,j])

{

Num -= sp.data[sc, j];

j++;

}

Now[i] = j;

sc -= j;

i++;

}

Now[0] = i - 1;

return Now; //Метод возвращает полученное разбиение

}

public int getNumOfSplit(int[] split)

{

int i, jk=N /\*Номер строки в массиве data\*/, p, sc = 1;//Формируем номер разбиения.

for (i = 1; i <= split[0]; i++)//Просматривается разбиение

{

for (p = 0; p <= split[i] - 1; p++)

{

sc += sp.data[jk, p];//Значение числа из разбиения определяет число столбцов в массиве data

}

jk -= split[i];//Изменяется номер строки

}

return sc;

}

}

}

Для проверки корректности работы класса, была написана программа, которая сначала получает разбиение по его номеру, а затем определяет номер полученного разбиения.

Код программы:

namespace splitting

{

class Program

{

static void Main(string[] args)

{

int n = 8;

int m = 6;

SplitNumber split = new SplitNumber(n);

int[] arr = split.getSplitByNum(m);

Console.WriteLine(split.getNumOfSplit(arr));

}

}

}

В ходе работы программы, сначала рассчитается шестое разбиение числа восемь, а затем выведется номер полученного разбиения, т.е. число шесть.

Результат работы программы представлен на Рисунке 6.

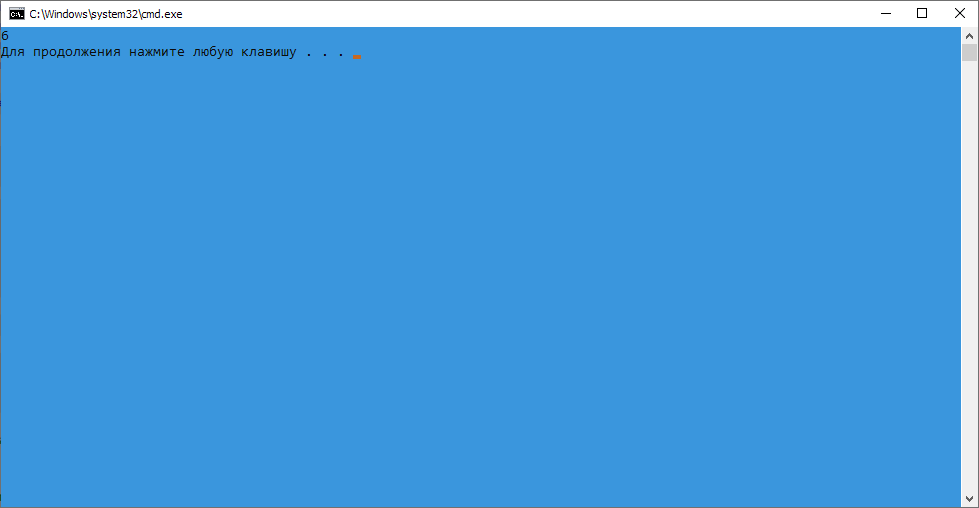


Рисунок 6. Результат работы программы, реализующей класс SplitNumber

Как можно видеть, программа показала верный результат, и метод GetNumOfSplit работает корректно.

**2.6 Пользовательский интерфейс и тесты программы.**

Код пользовательского интерфейса:

namespace splitting

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Введите число");

int a = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Выберите действие для целого числа: ");

Console.WriteLine("1. Посчитать количество разбиений");

Console.WriteLine("2. Сгенерировать все разбиения");

Console.WriteLine("3. Получить разбиение по номеру");

Console.WriteLine("4. Получить номер разбиения");

Console.WriteLine("0. Выход");

int b = Convert.ToInt32(Console.ReadLine());

switch (b)

{

case 1:

Console.WriteLine($"Количество разбиений для числа {a} = {SplitCounter.GetCount(a)}");

break;

case 2:

{

Splitting sp = new Splitting(a);

for (int i = 0; i < SplitCounter.GetCount(a); i++)

{

Console.WriteLine(sp.ToString());

sp.GetNext();

}

break;

}

case 3:

{

Console.WriteLine($"Введите номер разбиения до {SplitCounter.GetCount(a)}");

int c = Convert.ToInt32(Console.ReadLine());

var sn = new SplitNumber(a);

int[] arr = sn.getSplitByNum(c);

for (int i = 1; i <= arr[0]; i++)

{

Console.Write(arr[i]);

}

Console.WriteLine();

break;

}

case 4:

{

Console.WriteLine($"Введите разбиение числа {a}");

var c = Console.ReadLine();

int[] split = new int[a + 1];

for (int i = 0; i < a+1; i++)

{

split[i] = 1;

}

for (int i = 1; i <= c.Length; i++)

{

split[i] = c[i-1] - 48;

}

split[0] = c.Length;

var sn = new SplitNumber(a);

Console.WriteLine(sn.getNumOfSplit(split));

break;

}

case 0:

return;

break;

default:

break;

}

}

}

}

Результаты тестирования программы приведено на рисунках 7-10.

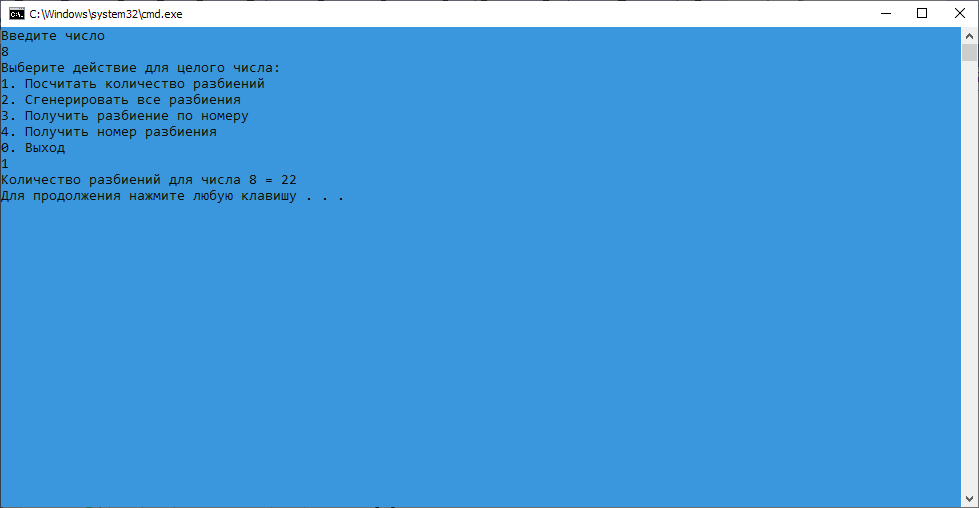


Рисунок 7. Расчет количества разбиений числа 8

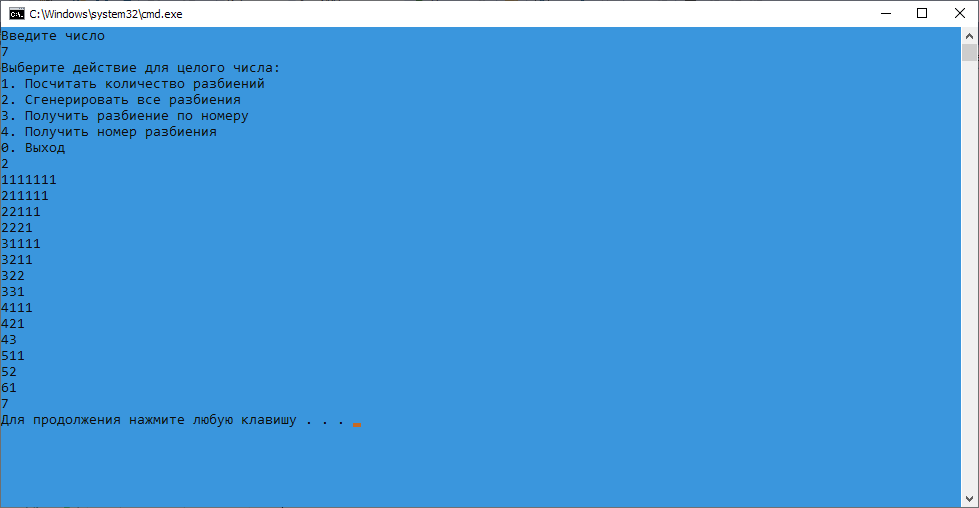


Рисунок 8. Генерация всех разбиений числа 7

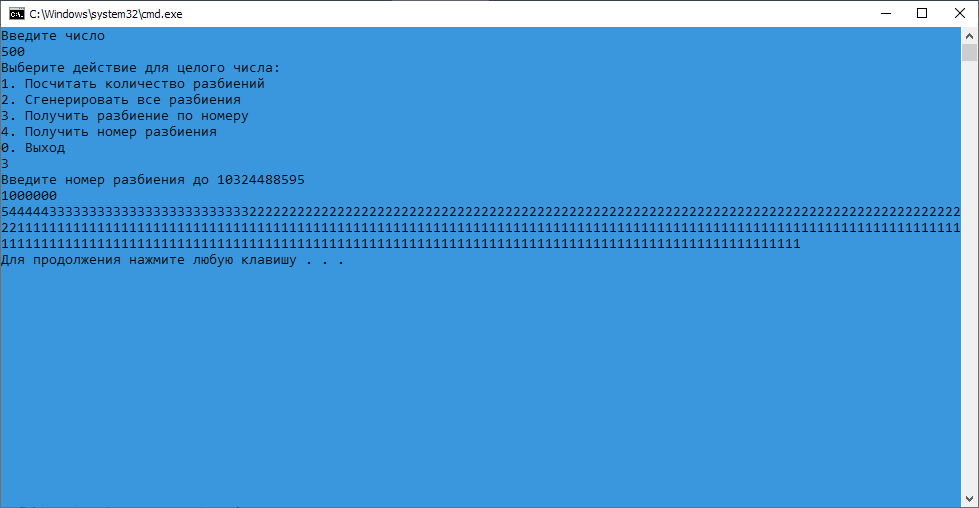


Рисунок 9. Миллионное разбиение числа 500

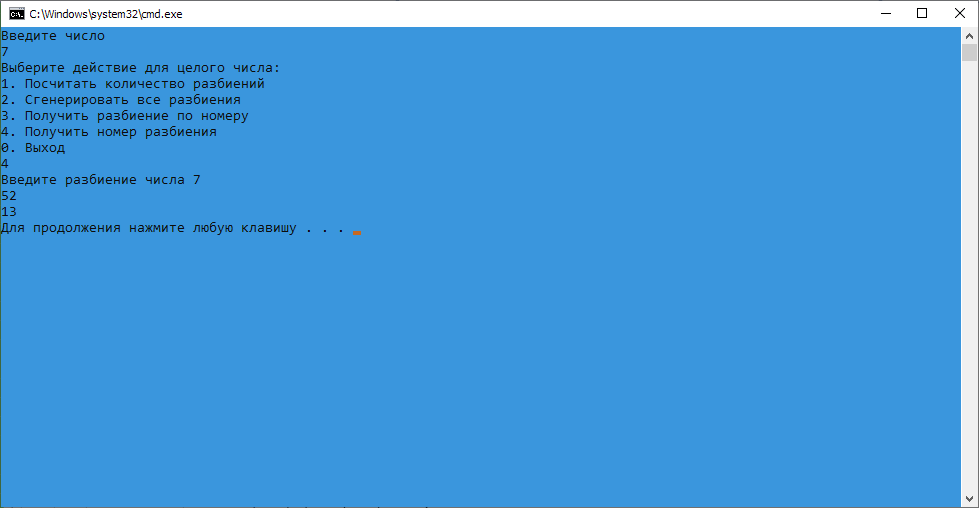


Рисунок 10. "5+2" - это тринадцатое разбиение числа 7

# **Заключение**

В результате выполнения курсовой работы на тему: «Программная реализация алгоритмов разбиения натурального числа на слагаемые», были выполнены все поставленные задачи:

* рассмотрены общие понятия и теоретические основы разработки алгоритмов разбиения натурального числа на слагаемые;
* разработан класс для решения задачи о количестве разбиений;
* разработан класс для решения задачи о генерации всех разбиений;
* разработан класс для решения задач о получении разбиения по его номеру и номера разбиения по самому разбиению;
* разработано и протестировано программное приложение, демонстрирующее работу классов;

Программа работает корректно, но ограничена размерами целочисленных переменных в языке программирования C#.

# **Список литературы**

1. Андреева Е. В., Фалина И. Н. Информатика: Системы счисления и компьютерная арифметика. — М.: Лаборатория Базовых Знаний, 1999.
2. Ахо А.В. Структуры данных и алгоритмы; Пер. с англ.: Уч. пос./ А. В. Ахо, Д. Э. Хопкрофт, Д. Д. Ульман. – М: Изд. дом «Вильямс», 2000.
3. Кирюхин, В.М. Информатика: всероссийские олимпиады. – М: Просвещение, 2008.
4. Клейнберг Дж. Алгоритмы: разработка и применение/ Дж. Клейнберг, Е. Тадос. – СПб: Питер, 2016.
5. Кнут Д. Искусство программирования для ЭВМ. Т 3. Сортировка и поиск/ Д. Кнут./ 2-е издание. Серия: Искусство программирования. – Киев: «Вильямс», 2000.
6. Кормен Т. Алгоритмы. Построение и анализ. 2-е издание / Т. Кормен, Ч. Леверсон, Р. Ривест. – М: Изд. дом «Вильямс», 2005.
7. Макконнелл Дж., Основы современных алгоритмов. 2-е дополнительное издание/ Дж. Макконнелл. — М: Изд. дом «Вильямс», 2004.
8. Мальцев А. И. Алгоритмы и рекурсивные функции. 2-е издание/ А. И. Мальцев. – М: Наука, 1986.
9. Окулов С. М. Алгоритмы компьютерной арифметики/ С. М. Окулов, А. В. Лялин, О. А. Пестов, Е. В. Разова. – 2-е изд.(эл.). – М: БИНОМ. Лаборатория знаний, 2015.
10. Окулов С. М. Программирование в алгоритмах/ С. М. Окулов. – М: БИНОМ. Лаборатория знаний, 2017.
11. Хайнеман Дж. Алгоритмы. Справочник с примерами на С, C++, Java и Python, 2-е изд.: Пер. с англ./ Дж. Хайнеман, Г. Поллис, С. Селков. — СПБ: ООО "Альфа-книга", 2017.