

Санкт-Петербургский Национальный Исследовательский
Университет Информационных Технологий, Механики и Оптики

Факультет инфокоммуникационных технологий

Лабораторная работа №1
Алгоритмы хеширования

Выполнил
Стафеев И.А.

Группа
К3221

Проверил
Иванов С.Е.

Санкт-Петербург,
2025

СОДЕРЖАНИЕ

	Стр.
ВВЕДЕНИЕ	3
1 Упражнение 1.....	4
2 Упражнение 2.....	10
ЗАКЛЮЧЕНИЕ	14

ВВЕДЕНИЕ

Цель работы: изучить алгоритмы хеширования, реализовать алгоритм Рабина-Карпа поиска подстроки в строке с применением хеширования.

Для достижения цели необходимо выполнить следующие задачи:

1. Реализовать поиск одинаковых строк. Дан список строк $S[1 \dots n]$, каждая длиной не более m символов. Требуется найти все повторяющиеся строки и разделить их на группы, чтобы в каждой группе были только одинаковые строки.
2. Реализовать алгоритм Рабина-Карпа поиска подстроки в строке за $O(n)$.

1 Упражнение 1

Задание: Реализовать поиск одинаковых строк. Дан список строк $S[1 \dots n]$, каждая длиной не более m символов. Требуется найти все повторяющиеся строки и разделить их на группы, чтобы в каждой группе были только одинаковые строки.

Решение:

Идея решения заключается в вычислении хэша для каждой строки, а затем в сравнении хэшей, а не самих строк, поскольку сравнение хэшей имеет сложность $O(1)$, в отличие от сравнения строк - $O(m)$. Для каждой строки будет вычисляться полиномиальный хэш по формуле

$$hash(s) = \left(\sum_{i=0}^{len(s)} (s[i] - 'a' + 1) \cdot p^i \right) \bmod mod \quad (1)$$

, где p - число, большее максимального значения для буквы и взаимно простое с mod .

Ниже на рисунках 1-7 описаны входные данные, методы программы и результат ее выполнения.

```
string_list.csv > data
1  ookaaygtty
2  wdhtgim
3  rdpjggif
4  obnynlsrvn
5  elvwogo
6  cml
7  zggbh
8  ysspzdlp
9  dbht
10 lcamyxt
11 hrxzqlq
12 tlsj
13 ykryms
14 nkucsoeaz
15 ovcwbgev
16 sijait
17 gzmhscn
18 ctxnltpyx
19 ctxnltpyx
20 tjzwvbwefb
21 copgpfgec
22 iegebf
23 fmwlrj
24 oufcflejq
25 hagrighh
```

Рисунок 1 — Входные данные для программы

Ссылка: 1

```
static string[] ReadLinesFromFile(string file_name)
{
    string proj_dir = Directory.GetParent(Directory.GetCurrentDirectory()).Parent.Parent.FullName;
    string file_path = Path.Combine(proj_dir, file_name);
    var lines = new List<string>();

    using (var reader = new StreamReader(file_path))
    {
        string line;
        while ((line = reader.ReadLine()) != null)
        {
            lines.Add(line);
        }
    }
    return lines.ToArray();
}
```

Рисунок 2 — Метод для чтения всех строк из файла с входными данными

Ссылка: 1

```
static long[] GetPPows(int p, int m)
{
    // массив степеней p
    long[] p_pows = new long[m];
    p_pows[0] = 1;
    for (int i = 1; i < m; i++)
    {
        p_pows[i] = p_pows[i - 1] * p;
    }
    return p_pows;
}
```

Ссылка: 1

Рисунок 3 — Метод для получения массива степеней p

```

,
Ссылка: 1
static long CalcHash(string s, long[] p_pows, long mod)
{
    // вычисление полиномиального хэша
    long hash = 0;
    for (int i = 0; i < s.Length; i++)
    {
        hash += (s[i] - 'a' + 1) * p_pows[i];
    }
    return hash % mod;
}
Ссылка: 1

```

Рисунок 4 — Метод для вычисления хэша для передаваемой строки *s*

```

Ссылка: 1
static Dictionary<int, long> CreateHashDict(string[] lines, int m, int p, long mod)
{
    var p_pows = GetPPows(p, m);
    var hashes = new Dictionary<int, long>();

    for (int i = 0; i < lines.Length; i++)
    {
        hashes[i] = CalcHash(lines[i], p_pows, mod);
    }

    return hashes;
}
Ссылка: 0

```

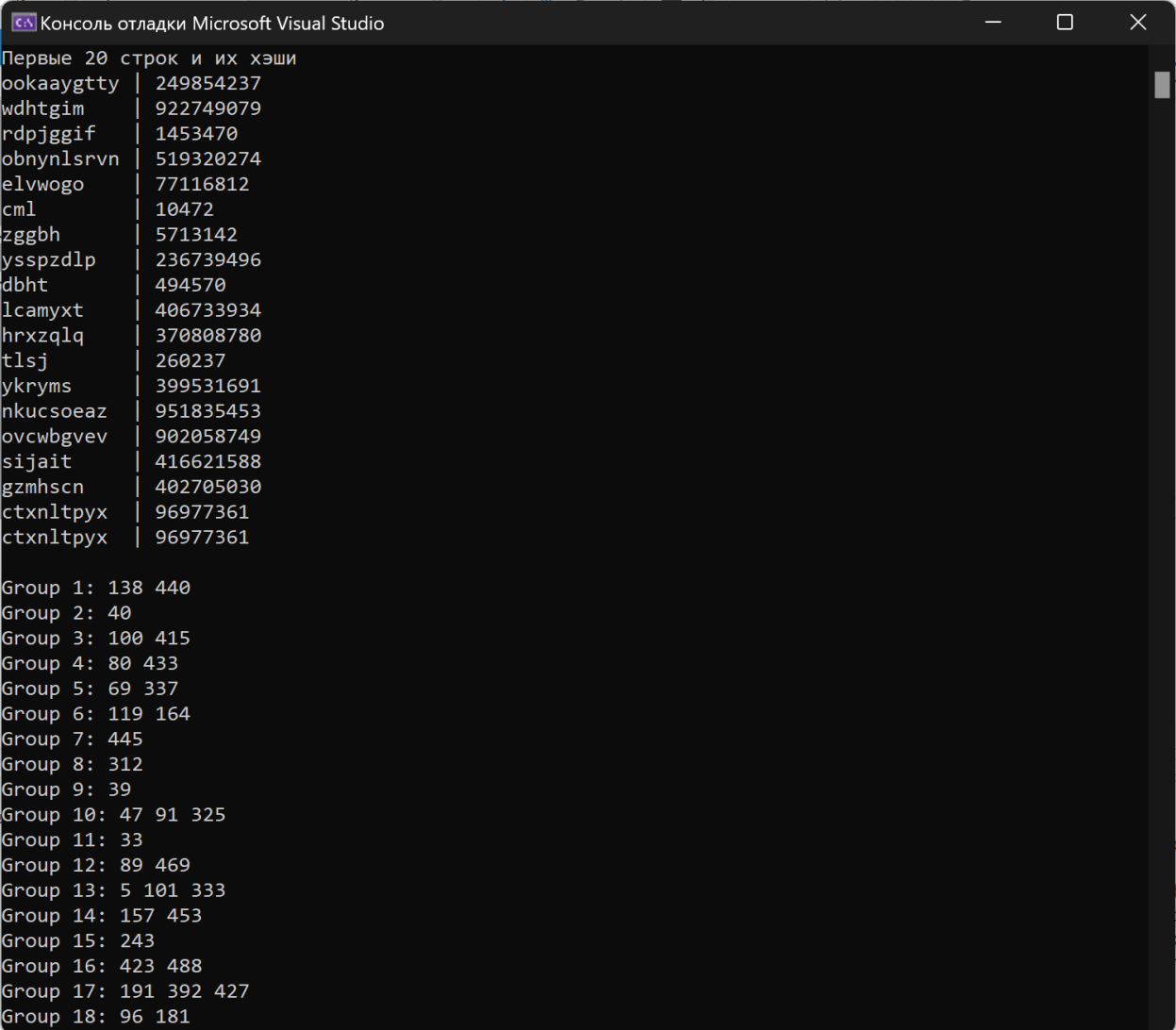
Рисунок 5 — Метод создание словаря, хранящего номер строки и ее хэш

```

57 static void Main()
58 {
59     int m = 10, p = 29;
60     long mod = 1000000007;
61     string file_name = "string_list.csv";
62
63     var lines = ReadLinesFromFile(file_name);
64     var hashes = CreateHashDict(lines, m, p, mod);
65
66     int cnt = 20;
67     Console.WriteLine("Первые {0} строк и их хэши", cnt);
68     int i = 1;
69     foreach (var hash in hashes)
70     {
71         Console.WriteLine("{0, -10} | {1}", lines[hash.Key], hash.Value);
72         i++;
73         if (i == cnt) { break; }
74     }
75
76     hashes = hashes.OrderBy(hash => hash.Value).ToDictionary(hash => hash.Key, hash => hash.Value);
77
78     int group_num = 0;
79     var prev_hash = hashes.First();
80     foreach (var hash in hashes)
81     {
82         if (hash.Equals(hashes.First()) || hash.Value != prev_hash.Value)
83         {
84             group_num++;
85             Console.WriteLine();
86             Console.Write("Group {0}: ", group_num);
87         }
88         Console.Write(hash.Key + " ");
89         prev_hash = hash;
90     }
91 }
92

```

Рисунок 6 — Основной метод программы



```
Консоль отладки Microsoft Visual Studio

Первые 20 строк и их хэши
ookaaygtty | 249854237
wdhtgim | 922749079
rdpjggif | 1453470
obnynlsrvn | 519320274
elvwogo | 77116812
cml | 10472
zggbh | 5713142
ysspzdlp | 236739496
dbht | 494570
lcamyxt | 406733934
hrxzqlq | 370808780
tlsj | 260237
ykryms | 399531691
nkucsoeaz | 951835453
ovcwbgeev | 902058749
sijait | 416621588
gzmhscn | 402705030
ctxnltpyx | 96977361
ctxnltpyx | 96977361

Group 1: 138 440
Group 2: 40
Group 3: 100 415
Group 4: 80 433
Group 5: 69 337
Group 6: 119 164
Group 7: 445
Group 8: 312
Group 9: 39
Group 10: 47 91 325
Group 11: 33
Group 12: 89 469
Group 13: 5 101 333
Group 14: 157 453
Group 15: 243
Group 16: 423 488
Group 17: 191 392 427
Group 18: 96 181
```

Рисунок 7 — Пример работы программы

2 Упражнение 2

Задание: Реализовать алгоритм Рабина-Карпа поиска подстроки в строке за $O(N)$. Дана строка T и текст S , состоящие из маленьких латинских букв. Требуется найти все вхождения строки T в текст S за время $O(|S| + |T|)$.

Решение:

По аналогии с предыдущей задачей, будем использовать полиномиальное хэширование. Для текста S создадим массив хэшей для всех префиксов текста. Благодаря следующему свойству полиномиального хэша

$$hash(S_{0...R}) = hash(S_{0...L-1}) + p^L \cdot hash(S_{L...R}) \quad (2)$$

, задача можно свести к последовательному сравнению хэша искомой строки с хэшем префиксов текста, причем с каждой итерацией текст сокращается слева, чтобы всегда производилось сравнение с префиксом, а не с произвольным срезом.

Ниже на рисунках 8-12 представлены методы и результат выполнения программы.

Ссылка: 1

```
static long[] GetPPows(int p, int m)
{
    // массив степеней p
    long[] p_pows = new long[m];
    p_pows[0] = 1;
    for (int i = 1; i < m; i++)
    {
        p_pows[i] = p_pows[i - 1] * p;
    }
    return p_pows;
}
```

Ссылка: 1

Рисунок 8 — Метод для получения массива степеней p

,

Ссылка: 1

```
static long CalcHash(string s, long[] p_pows, long mod)
{
    // вычисление полиномиального хэша
    long hash = 0;
    for (int i = 0; i < s.Length; i++)
    {
        hash += (s[i] - 'a' + 1) * p_pows[i];
    }
    return hash % mod;
}
```

Ссылка: 1

Рисунок 9 — Метод для вычисления хэша для передаваемой строки s

Ссылка: 1

```
static long[] CalcPrefixHash(string s, long[] p_pows)
{
    // вычисление хэшей для всех префиксов строки
    long[] hashes = new long[s.Length];
    for (int i = 0; i < s.Length; i++)
    {
        hashes[i] = (s[i] - 'a' + 1) * p_pows[i];
        if (i != 0)
        {
            hashes[i] += hashes[i - 1];
        }
    }
    return hashes;
}
```

Рисунок 10 — Метод для вычисления хэша для всех префиксов передаваемой строки *s*

Ссылка: 0

```
static void Main()
{
    int p = 29;

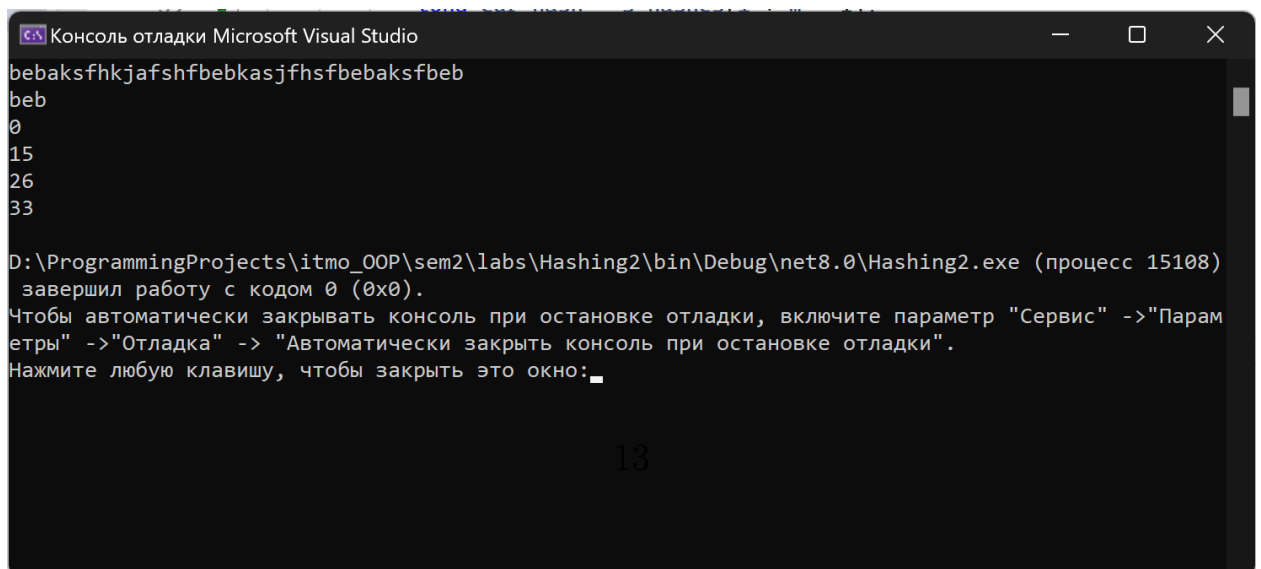
    string s = Console.ReadLine();
    string t = Console.ReadLine();
    int n = s.Length, m = t.Length;

    var p_pows = GetPPows(p, n);
    var s_hashes = CalcPrefixHash(s, p_pows);
    var t_hash = CalcHash(t, p_pows);

    for (int i = 0; i <= n - m; i++)
    {
        long cur_hash = s_hashes[i + m - 1];
        if (i > 0)
        {
            cur_hash = cur_hash - s_hashes[i - 1];
        }

        if (cur_hash == t_hash * p_pows[i])
        {
            Console.WriteLine(i);
        }
    }
}
```

Рисунок 11 — Основной метод программы



ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы были выполнены все задачи, и цель работы достигнута. Были изучены методы хэширования и алгоритм Рабина-Карпа для поиска подстрок. Полученные навыки будут полезны в дальнейшей профессиональной деятельности.