Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики

Кафедра информатики и прикладной математики

Дисциплина «Алгоритмы и структуры данных»
Лабораторная работа №1

Вариант 18

Выполнил:
Съестов Дмитрий Вячеславович
Группа P3217

Преподаватель:
Зинчик Александр Адольфович

Санкт-Петербург
2018

Задание

Написать программу, которая получает на входе набор идентификаторов, организует таблицу по заданному методу и позволяет осуществить **многократный** поиск идентификатора в этой таблице. Список идентификаторов считать заданным в виде текстового файла. Длина идентификаторов ограничена 32 символами.

Хеш-функция: сумма кодов последних трех букв

Разрешение коллизий: метод цепочек

Хеш-таблица представляет собой массив длиной 196606 элементов (все возможные значения хеш-функции), в каждой ячейке которого хранится связный список идентификаторов.

Листинг (C#)

```
using System;
using System.Diagnostics;
using System.IO;

namespace alg1
{
    internal class MyHashtable
    {
        private const int HASHTABLE_SIZE = char.MaxValue * 3;

        private class Element
        {
            public readonly string Value;
            public Element NextElement;

            public Element(string value)
            {
                Value = value;
                NextElement = null;
            }
        }
        private readonly Element[] table;

        public int ElementCount    { get; private set; }
        public int HashCount       { get; private set; }
        public int CollisionCount  { get; private set; }
        public int ComparisonCount { get; private set; }
        public double AverageCollisions  => HashCount == 0 ? 0 :
              (double)CollisionCount / HashCount;
        public double AverageComparisons => ElementCount == 0 ? 0 :
              (double)ComparisonCount / ElementCount;

        public MyHashtable()
        {
            table = new Element[HASHTABLE_SIZE];
            CollisionCount = 0;
            HashCount = 0;
            ElementCount = 0;
            ComparisonCount = 0;
        }
```

```
        public static MyHashtable FromFile(string filename, bool countComparisons = false)
        {
            if (!File.Exists(filename)) throw new FileNotFoundException();
            var hashtable = new MyHashtable();

            using (var reader = new StreamReader(filename))
            {
                var separators = new[] {' ', '\n', '\t'};
                var words = reader.ReadToEnd().Split(separators,
StringSplitOptions.RemoveEmptyEntries);
                foreach (var word in words) hashtable.Add(word);

                if (countComparisons)
                {
                    foreach (var word in words) hashtable.Search(word);
                }
            }
            return hashtable;
        }

        private static int Hash(string value)
        {
            int startIndex = value.Length - Math.Min(3, value.Length);

            int sum = 0;
            for (int index = startIndex; index < value.Length; index++)
                sum += value[index];

            Debug.Assert(sum <= HASHTABLE_SIZE);
            return sum;
        }

        public void Add(string value)
        {
            int hashcode = Hash(value);
            var element = new Element(value);

            if (table[hashcode] == null)
            {
                table[hashcode] = element;
                HashCount++;
            }
            else
            {
                Element target = table[hashcode];
                if (target.NextElement == null) CollisionCount++;
                while (target.NextElement != null) target = target.NextElement;
                target.NextElement = element;
            }
            ElementCount++;
        }

        public bool Search(string value)
        {
            int hashcode = Hash(value);
            Element element = table[hashcode];
```

```
            while (element != null)
            {
                ComparisonCount++;
                if (element.Value == value) return true;
                element = element.NextElement;
            }

            return false;
        }
    }
}
```