Министерство образования и науки Российской Федерации

Севастопольский государственный университет

Кафедра ИС

Отчет

По дисциплине: “Управление информационными ресурсами”

Лабораторная работа №2

“Структуры данных, основанные на хеш-таблицах”

Выполнил:

ст.гр. ИС/б-17-2

Долженко И.А.

Проверил:

Дрозин А.Ю.

Севастополь

2020

1 ЦЕЛЬ РАБОТЫ

Исследовать возможности применения нелинейных структур данных – хеш-таблиц для хранения и обработки информации. Приобрести практические навыки использования хеш-таблиц для реализации быстрого доступа к данным. Произвести оценку эффективности использования хеш-таблиц для организации хранения данных.

2 ПОСТАНОВКА ЗАДАЧИ

1. На языке программирования С# в среде программирования Visual Studio, с использованием класса, реализующего хеш-таблицу реализовать Windows-приложение, обеспечивающее выполнения следующих функций:

* Открытие файла исходных данных и построение на его основе файла, содержащего хеш-таблицу с внешним хешированием;
* Отображение на визуальной форме содержимого исходного файла и хеш таблицы;
* Предоставление интерфейса пользователю для выполнения операций добавления, удаления, изменения и поиска элементов в файле и хеш таблице, и отображения результатов выполнения операций на визуальной форме;
* Отображение времени выполнения операций добавления, удаления, изменения и поиска данных по заданному пользователем значению ключевого поля.

2. С использованием разработанной программы выполнить исследования структур данных обычного файла с последовательным расположением записей и хеш-таблицы:

* Используя разработанную программу на основании исходного файла данных создать хеш-таблицу, содержащую N1 записей;
* Выполнить по 5 раз операции добавления, удаления и поиска информации, фиксируя в отчете время выполнения операций;
* Вычислить среднее время выполнения операций добавления, удаления и поиска информации (по ключевому полю).

Вариант 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ключевое поле | N1 | N2 | N3 | N4 |
| Абонент | 20 | 200 | 1400 | 6000 |

Хеш-функция: Сумма по модулю 100 кодов двух последних символов ключа.

3 ТЕКСТ ПРОГРАММЫ

Основной файл Program.cs:

using System;

using System.Diagnostics;

using System.IO;

using System.Windows.Forms;

namespace WindowsFormsApp3

{

public partial class Form1 : Form

{

private readonly string path = Path.Combine(AppDomain.CurrentDomain.BaseDirectory, @"..\..\file.txt");

private HashTable hashTable = new HashTable();

private Stopwatch sWatch = new Stopwatch();

public Form1()

{

InitializeComponent();

readFile();

fillDataHashFromMemory();

}

private void readFile()

{

dataGridViewFile.Rows.Clear();

string[] lines = File.ReadAllLines(path);

string[] values;

for (int i = 0; i < lines.Length; i++)

{

values = lines[i].ToString().Split('\t');

string[] row = new string[values.Length];

for (int j = 0; j < values.Length; j++)

{

row[j] = values[j].Trim();

}

dataGridViewFile.Rows.Add(row);

hashTable.Insert(row[0], row[1]);

}

}

private void fillDataHashFromMemory()

{

dataGridViewHash.Rows.Clear();

foreach (var item in hashTable.Items)

{

foreach (var value in item.Value)

{

dataGridViewHash.Rows.Add(item.Key, value.Key);

}

}

}

private void appendToFile(string[] row)

{

using (StreamWriter writer = File.AppendText(path))

{

for (int i = 0; i < row.Length; i++)

{

writer.Write(row[i] + "\t");

}

writer.WriteLine("");

}

}

private void removeFromFile()

{

string[] lines = File.ReadAllLines(path);

using (StreamWriter writer = new StreamWriter(path))

{

foreach (string line in lines)

{

string[] words = line.Split('\t');

if (words[0] != textBoxSubscriber.Text)

{

writer.WriteLine(line);

}

}

}

}

private bool findInFile()

{

string[] lines = File.ReadAllLines(path);

string[] values;

for (int i = 0; i < lines.Length; i++)

{

values = lines[i].ToString().Split('\t');

if (values[0] == textBoxSearchSubscriber.Text)

{

return true;

}

}

return false;

}

private void resetAllFields()

{

textBoxSubscriber.Text = "";

textBoxPhone.Text = "";

textBoxHome.Text = "";

textBoxSum.Text = "";

}

private void setTimeLog(string title, string time)

{

textBoxLog.Text += title + time + "\r\n";

textBoxLog.SelectionStart = textBoxLog.Text.Length;

textBoxLog.ScrollToCaret();

}

private void buttonAdd\_Click(object sender, EventArgs e)

{

if (textBoxSubscriber.Text == "") return;

string[] row = new string[] { textBoxSubscriber.Text, textBoxPhone.Text, textBoxHome.Text, textBoxSum.Text };

// ДОБАВЛЕНИЕ В ХЕШ-ТАБЛИЦУ

sWatch.Start();

hashTable.Insert(row[0], row[1]);

sWatch.Stop();

setTimeLog("Добавление в хеш-таблицу: ", sWatch.ElapsedTicks.ToString());

sWatch.Reset();

// ДОБАВЛЕНИЕ В ОСНОВНУЮ ТАБЛИЦУ

sWatch.Start();

appendToFile(row);

sWatch.Stop();

setTimeLog("Добавление в основную таблицу: ", sWatch.ElapsedTicks.ToString());

sWatch.Reset();

dataGridViewFile.Rows.Add(row);

fillDataHashFromMemory();

resetAllFields();

}

private void buttonRemove\_Click(object sender, EventArgs e)

{

if (textBoxSubscriber.Text == "") return;

// УДАЛЕНИЕ ИЗ ХЕШ-ТАБЛИЦЫ

sWatch.Start();

hashTable.Delete(textBoxSubscriber.Text);

sWatch.Stop();

setTimeLog("Удаление из хеш-таблицы: ", sWatch.ElapsedTicks.ToString());

sWatch.Reset();

foreach (DataGridViewRow item in dataGridViewFile.Rows)

{

if (item.Cells["Column1"].Value != null && item.Cells["Column1"].Value.ToString() == textBoxSubscriber.Text.ToString())

{

dataGridViewFile.Rows.RemoveAt(item.Index);

}

}

// УДАЛЕНИЕ ИЗ ОСНОВНОЙ ТАБЛИЦЫ

sWatch.Start();

removeFromFile();

sWatch.Stop();

setTimeLog("Удаление из основной таблицы: ", sWatch.ElapsedTicks.ToString());

sWatch.Reset();

fillDataHashFromMemory();

resetAllFields();

}

private void buttonSearch\_Click(object sender, EventArgs e)

{

if (textBoxSearchSubscriber.Text == "") return;

// ПОИСК В ХЕШ-ТАБЛИЦЕ

sWatch.Start();

hashTable.Search(textBoxSearchSubscriber.Text);

sWatch.Stop();

setTimeLog("Поиск в хеш-таблице: ", sWatch.ElapsedTicks.ToString());

sWatch.Reset();

// ПОИСК В ОСНОВНОЙ ТАБЛИЦЕ

sWatch.Start();

findInFile();

sWatch.Stop();

setTimeLog("Поиск в основной таблице: ", sWatch.ElapsedTicks.ToString());

sWatch.Reset();

foreach (DataGridViewRow row in dataGridViewFile.Rows)

{

if (row.Cells["Column1"].Value != null && row.Cells["Column1"].Value.ToString() == textBoxSearchSubscriber.Text)

{

row.Selected = true;

}

}

foreach (DataGridViewRow row in dataGridViewHash.Rows)

{

if (row.Cells["Key"].Value != null && row.Cells["Key"].Value.ToString() == textBoxSearchSubscriber.Text)

{

row.Selected = true;

}

}

}

private void buttonReset\_Click(object sender, EventArgs e)

{

textBoxSearchSubscriber.Text = "";

}

private void dataGridViewFile\_CellClick(object sender, DataGridViewCellEventArgs e)

{

if (e.RowIndex >= 0)

{

DataGridViewRow row = dataGridViewFile.Rows[e.RowIndex];

if (row.Cells["Column1"].Value != null)

{

textBoxSubscriber.Text = row.Cells["Column1"].Value.ToString();

}

}

}

}

}

Файл HashTable.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace WindowsFormsApp3

{

/// <summary>

/// Хеш-таблица.

/// </summary>

/// <remarks>

/// Используется метод цепочек (открытое хеширование).

/// </remarks>

public class HashTable

{

/// <summary>

/// Максимальная длина ключевого поля.

/// </summary>

private readonly byte \_maxSize = 255;

/// <summary>

/// Коллекция хранимых данных.

/// </summary>

/// <remarks>

/// Представляет собой словарь, ключ которого представляет собой хеш ключа хранимых данных,

/// а значение это список элементов с одинаковым хешем ключа.

/// </remarks>

private Dictionary<int, List<Item>> \_items = null;

/// <summary>

/// Коллекция хранимых данных в хеш-таблице в виде пар Хеш-Значения.

/// </summary>

public IReadOnlyCollection<KeyValuePair<int, List<Item>>> Items => \_items?.ToList()?.AsReadOnly();

/// <summary>

/// Создать новый экземпляр класса HashTable.

/// </summary>

public HashTable()

{

// Инициализируем коллекцию максимальным количество элементов.

\_items = new Dictionary<int, List<Item>>(\_maxSize);

}

/// <summary>

/// Добавить данные в хеш таблицу.

/// </summary>

/// <param name="key"> Ключ хранимых данных. </param>

/// <param name="value"> Хранимые данные. </param>

public void Insert(string key, string value)

{

// Проверяем входные данные на корректность.

if (string.IsNullOrEmpty(key))

{

throw new ArgumentNullException(nameof(key));

}

if (key.Length > \_maxSize)

{

throw new ArgumentException($"Максимальная длинна ключа составляет {\_maxSize} символов.", nameof(key));

}

if (string.IsNullOrEmpty(value))

{

throw new ArgumentNullException(nameof(value));

}

// Создаем новый экземпляр данных.

var item = new Item(key, value);

// Получаем хеш ключа

var hash = GetHash(item.Key);

// Получаем коллекцию элементов с таким же хешем ключа.

// Если коллекция не пустая, значит заначения с таким хешем уже существуют,

// следовательно добавляем элемент в существующую коллекцию.

// Иначе коллекция пустая, значит значений с таким хешем ключа ранее не было,

// следовательно создаем новую пустую коллекцию и добавляем данные.

List<Item> hashTableItem = null;

if (\_items.ContainsKey(hash))

{

// Получаем элемент хеш таблицы.

hashTableItem = \_items[hash];

// Проверяем наличие внутри коллекции значения с полученным ключом.

// Если такой элемент найден, то сообщаем об ошибке.

var oldElementWithKey = hashTableItem.SingleOrDefault(i => i.Key == item.Key);

if (oldElementWithKey != null)

{

throw new ArgumentException($"Хеш-таблица уже содержит элемент с ключом {key}. Ключ должен быть уникален.", nameof(key));

}

// Добавляем элемент данных в коллекцию элементов хеш таблицы.

\_items[hash].Add(item);

}

else

{

// Создаем новую коллекцию.

hashTableItem = new List<Item> { item };

// Добавляем данные в таблицу.

\_items.Add(hash, hashTableItem);

}

}

/// <summary>

/// Удалить данные из хеш таблицы по ключу.

/// </summary>

/// <param name="key"> Ключ. </param>

public void Delete(string key)

{

// Проверяем входные данные на корректность.

if (string.IsNullOrEmpty(key))

{

throw new ArgumentNullException(nameof(key));

}

if (key.Length > \_maxSize)

{

throw new ArgumentException($"Максимальная длинна ключа составляет {\_maxSize} символов.", nameof(key));

}

// Получаем хеш ключа.

var hash = GetHash(key);

// Если значения с таким хешем нет в таблице,

// то завершаем выполнение метода.

if (!\_items.ContainsKey(hash))

{

return;

}

// Получаем коллекцию элементов по хешу ключа.

var hashTableItem = \_items[hash];

// Получаем элемент коллекции по ключу.

var item = hashTableItem.SingleOrDefault(i => i.Key == key);

// Если элемент коллекции найден,

// то удаляем его из коллекции.

if (item != null)

{

hashTableItem.Remove(item);

}

}

/// <summary>

/// Поиск значения по ключу.

/// </summary>

/// <param name="key"> Ключ. </param>

/// <returns> Найденные по ключу хранимые данные. </returns>

public string Search(string key)

{

// Проверяем входные данные на корректность.

if (string.IsNullOrEmpty(key))

{

throw new ArgumentNullException(nameof(key));

}

if (key.Length > \_maxSize)

{

throw new ArgumentException($"Максимальная длинна ключа составляет {\_maxSize} символов.", nameof(key));

}

// Получаем хеш ключа.

var hash = GetHash(key);

// Если таблица не содержит такого хеша,

// то завершаем выполнения метода вернув null.

if (!\_items.ContainsKey(hash))

{

return null;

}

// Если хеш найден, то ищем значение в коллекции по ключу.

var hashTableItem = \_items[hash];

// Если хеш найден, то ищем значение в коллекции по ключу.

if (hashTableItem != null)

{

// Получаем элемент коллекции по ключу.

var item = hashTableItem.SingleOrDefault(i => i.Key == key);

// Если элемент коллекции найден,

// то возвращаем хранимые данные.

if (item != null)

{

return item.Value;

}

}

// Возвращаем null если ничего найдено.

return null;

}

/// <summary>

/// Хеш функция.

/// </summary>

/// <remarks>

/// Возвращает длину строки.

/// </remarks>

/// <param name="value"> Хешируемая строка. </param>

/// <returns> Хеш строки. </returns>

private int GetHash(string value)

{

// Проверяем входные данные на корректность.

if (string.IsNullOrEmpty(value))

{

throw new ArgumentNullException(nameof(value));

}

if (value.Length > \_maxSize)

{

throw new ArgumentException($"Максимальная длинна ключа составляет {\_maxSize} символов.", nameof(value));

}

byte[] bytesArray = Encoding.GetEncoding(1251).GetBytes(value);

var hash = (bytesArray[value.Length - 2] + bytesArray[value.Length - 1]) % 100;

return hash;

}

}

}

Файл Item.cs:

using System;

namespace WindowsFormsApp3

{

/// <summary>

/// Элемент данных хеш таблицы.

/// </summary>

public class Item

{

/// <summary>

/// Ключ.

/// </summary>

public string Key { get; private set; }

/// <summary>

/// Хранимые данные.

/// </summary>

public string Value { get; private set; }

/// <summary>

/// Создать новый экземпляр хранимых данных Item.

/// </summary>

/// <param name="key"> Ключ. </param>

/// <param name="value"> Значение. </param>

public Item(string key, string value)

{

// Проверяем входные данные на корректность.

if (string.IsNullOrEmpty(key))

{

throw new ArgumentNullException(nameof(key));

}

if (string.IsNullOrEmpty(value))

{

throw new ArgumentNullException(nameof(value));

}

// Устанавливаем значения.

Key = key;

Value = value;

}

/// <summary>

/// Приведение объекта к строке.

/// </summary>

/// <returns> Ключ объекта. </returns>

public override string ToString()

{

return Key;

}

}

}

4 РЕЗУЛЬТАТЫ

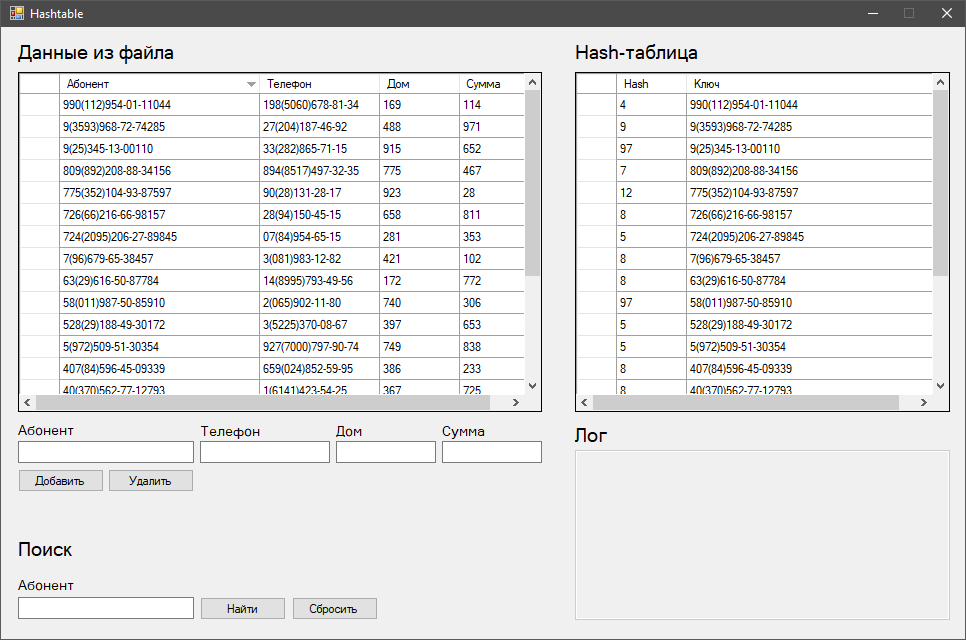


Рисунок 1 – Интерфейс программы

Измерим показатели времени добавления, удаления, поиска и занесем их в таблицу.

Таблица 1. Результаты проделанной работы

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Кол-во записей | № | Файл | | | Хеш-таблица | | |
| Добавление | Удаление | Поиск | Добавление | Удаление | Поиск |
| N1=20 | 1 | 1923 | 2529 | 533 | 15 | 37 | 19 |
| 2 | 1060 | 2193 | 524 | 17 | 28 | 27 |
| 3 | 1129 | 2192 | 498 | 13 | 26 | 20 |
| 4 | 1180 | 2251 | 500 | 19 | 21 | 20 |
| 5 | 1059 | 2213 | 587 | 16 | 21 | 21 |
| **Ср.** | **1270** | **2275** | **528** | **16** | **27** | **21** |
| N2=200 | 1 | 1867 | 5233 | 1737 | 32 | 27 | 20 |
| 2 | 1012 | 4073 | 1702 | 17 | 27 | 22 |
| 3 | 1207 | 4840 | 1723 | 22 | 20 | 24 |
| 4 | 1159 | 4341 | 1142 | 20 | 36 | 21 |
| 5 | 1089 | 4990 | 1029 | 18 | 40 | 23 |
| **Ср.** | **1266** | **4695** | **1466** | **22** | **30** | **22** |
| N3=1400 | 1 | 1266 | 39994 | 10536 | 16 | 84 | 25 |
| 2 | 1010 | 24813 | 31480 | 51 | 39 | 67 |
| 3 | 1076 | 16787 | 11092 | 61 | 43 | 52 |
| 4 | 1050 | 26471 | 1352 | 56 | 101 | 47 |
| 5 | 1075 | 18784 | 4410 | 31 | 46 | 45 |
| **Ср.** | **1095** | **25369** | **11774** | **43** | **63** | **47** |
| N4=6000 | 1 | 1058 | 68617 | 46013 | 19 | 99 | 147 |
| 2 | 1081 | 72669 | 32360 | 21 | 117 | 152 |
| 3 | 1085 | 71570 | 43221 | 62 | 122 | 122 |
| 4 | 1405 | 58705 | 46599 | 133 | 132 | 23 |
| 5 | 1025 | 79744 | 72141 | 81 | 81 | 135 |
| **Ср.** | **1130** | **70261** | **48066** | **63** | **110** | **116** |

Рисунок 2 – Диаграмма среднего времени выполнения операций в обычной таблице

Рисунок 3 – Диаграмма среднего времени выполнения операций в хеш-таблице

ВЫВОД

В ходе выполнения лабораторной работы было разработано приложение для манипулирования данными в обычной таблице и хеш-таблице. Было проведено исследование времени выполнения операций на различном количестве наборов данных, по результатам которого можно сделать следующие выводы: наиболее затратной операцией по времени является операция удаления для обеих таблиц, а быстрее всего выполняется операция поиска. По результатам сравнения среднего значения времени, очевидно, что хеш-таблица значительно быстрее обрабатывает данные.