МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСТЕТ

Факультет информационных систем и технологий

Кафедра «Информационные системы»

Лабораторная работа №5

по дисциплине «Системное программное обеспечение»

Вариант 26

Выполнил:

студент гр. ПИбд-31

Шупинский В.А.

Проверил:

доцент

Романов А. А.

Ульяновск

2020

**Цель работы:** изучить основные методы организации таблиц идентификаторов, получить представление о преимуществах и недостатках, присущих различным методам организации таблиц идентификаторов.

**Задание:**

3. Рехэширование с помощью произведения.

6. Комбинация хэш-адресации с упорядоченным списком.

**Хэш-функция:**

res += word[i] \* (i + 1)

***3. Рехэширование с помощью произведения.***

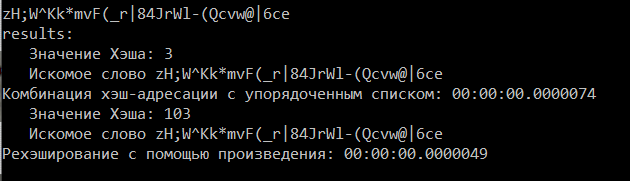
С использованием данного метода при возникновении коллизии новый хэш будет вычисляться по формуле hi(A) = (h(A) \* i) mod N, где h(A) значение предыдущей хэш-функции, а N – размер таблицы.

***6. Комбинация хэш-адресации с упорядоченным списком.***

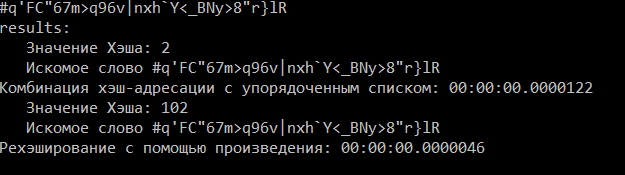
При комбинации хэш-адресации с упорядоченным списком каждому значению хэш-функции в таблице соответствует собственный упорядоченный список, который заполняется по мере возникновения коллизий.

**Результаты работы:**

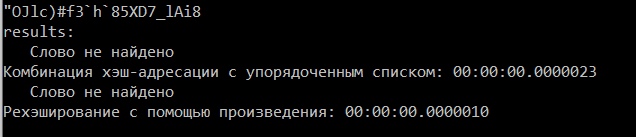
Первая добавленная в таблицу строка:



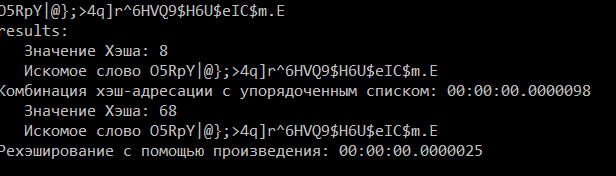
Последняя добавленная в таблицу строка:

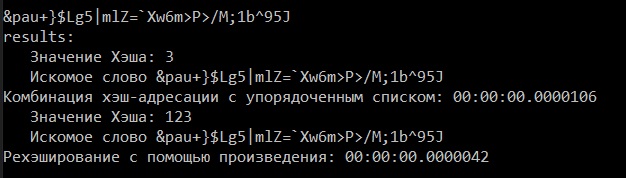


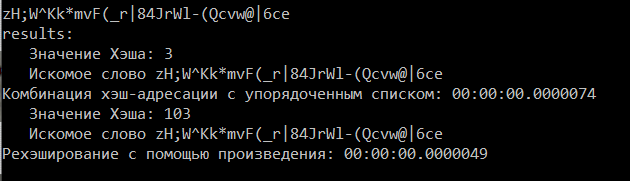
Поиск слова отсутствующего в таблице

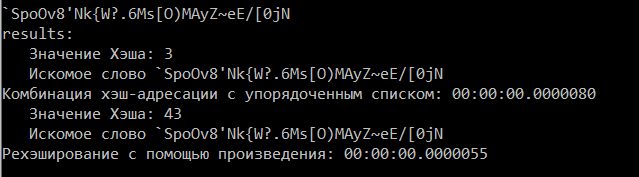


Строка из середины таблицы (примерно)

Случаи с возникновением коллизии







Количество идентификаторов – 75

Размер таблицы с упорядоченными списками – 15

Размер таблицы с рехешированием – 150

Т.к. размер таблицы рехеширования > кол-ва идентификаторов в ней могут быть незаполненные строки

**Листинг:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Diagnostics;

using System.IO;

namespace SPO5

{

class Program

{

const int N = 100;

static void Main(string[] args)

{

Stopwatch sw = new Stopwatch();

string text = File.ReadAllText("test.txt");

string[] words = text.Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

var hashOrdered = CreateHashOrdered(words);

var rehashable = CreateRehashable(words);

string actionWord = "";

while (!(actionWord = Console.ReadLine()).Equals("--exit"))

{

Console.WriteLine("results:");

string res = "";

sw = Stopwatch.StartNew();

res = Search(hashOrdered, actionWord);

sw.Stop();

Console.WriteLine(res);

Console.WriteLine("Комбинация хэш-адресации с упорядоченным списком: {0} ", sw.Elapsed);

sw = Stopwatch.StartNew();

res = Search(rehashable, actionWord);

sw.Stop();

Console.WriteLine(res);

Console.WriteLine("Рехэширование с помощью произведения: {0} ", sw.Elapsed);

Console.WriteLine();

}

}

static List<List<string>> CreateHashOrdered(string[] words)

{

List<List<string>> hash = new List<List<string>>(N / 10);

for (int i = 0; i < N / 10; i++) { hash.Add(null); }

foreach (string word in words)

{

int hashKey = GetHash(word, N / 10);

List<string> list = new List<string>();

if (hash[hashKey] == null)

{

list.Add(word);

hash[hashKey] = list;

}

else

{

list = hash[hashKey];

list.Add(word);

list.Sort();

hash[hashKey] = list;

}

}

return hash;

}

static List<string> CreateRehashable(string[] words)

{

List<string> hash = new List<string>(N);

for (int i = 0; i < N; i++) { hash.Add(null); }

foreach (string word in words)

{

int hashKey = GetHash(word, N);

if (hash[hashKey] == null) { hash[hashKey] = word; }

else

{

int i = 1;

int newHashKey;

while (i <= N)

{

newHashKey = ReHash(hashKey, i, N);

if (hash[newHashKey] != null)

{

hash[hashKey] = word;

break;

}

else { i += 1; }

}

}

}

return hash;

}

static int GetHash(string word, int length)

{

int res = 0;

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

{

res += word[i] \* (i + 1); //Хэш-функция

}

return res % length;

}

static int ReHash(int curentHash, int index, int length)

{

return (curentHash \* index) % length; //в отчёте будет - hi(A) = (h(A) \* i) mod N

}

static string Search(List<List<string>> hash, string word)

{

string res = " Слово не найдено";

int hashKey = GetHash(word, N / 10);

if (hash[hashKey] != null)

{

foreach (string str in hash[hashKey])

{

if (str.Equals(word))

{

res = " Значение Хэша: " + hashKey + "\n Искомое слово " + word;

break;

}

}

}

return res;

}

static string Search(List<string> hash, string word)

{

string res = " Слово не найдено";

int hashKey = GetHash(word, N);

int i = 1;

while (hash[hashKey] != null && i <= N)

{

if (hash[hashKey].Equals(word))

{

res = " Значение Хэша: " + hashKey + "\n Искомое слово " + word;

break;

}

else

{

hashKey = ReHash(hashKey, i, N);

i += 1;

}

}

return res;

}

}

}