БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Лабораторная работа № 14

по дисциплине «Структуры и алгоритмы обработки данных»

Вариант № 4

Выполнил студент: Ефименко Павел Викторович,

группа 981063

Минск 2020

**Условие:**

*Задание.* На основе динамических списков необходимо реализовать словарь и хранить его во внешней памяти. Основные операции, выполняемые над данными словаря: поиск, вставка и удаление. В сочетании со списками для построения и обработки словарей использовать *хешированные файлы*, позволяющее ускорить выполнение операций с данными словаря.

**Листинг:**

using System;

using System.Collections.Generic;

using System.Linq;

namespace Lr14

{

class Program

{

public class HashMap<K, V>

{

// Ключ

public K Key { get; private set; }

// Значение

public V Value { get; private set; }

// Создание нового значения

public HashMap(K key, V value)

{

Key = key;

Value = value;

}

// Переопределение метода ToString

public override string ToString()

{

return Value.ToString();

}

}

public class MyHashMap<K, V>

{

private Dictionary<int, IList<HashMap<K, V>>> \_hashMaps = null;

public IReadOnlyCollection<KeyValuePair<int, IList<HashMap<K, V>>>> HashMaps => \_hashMaps?.ToList()?.AsReadOnly();

public MyHashMap()

{

// Инициализируем словарь

\_hashMaps = new Dictionary<int, IList<HashMap<K, V>>>();

}

// Метод добавления новых значений в хеш-таблицу

public void Add(K key, V value)

{

// Создаём новый элемент

var newHashMapItem = new HashMap<K, V>(key, value);

// Создаём новый хеш ключа

var newHashMapHash = GetHash(key);

List<HashMap<K, V>> newHashMapList = new List<HashMap<K, V>>();

if (\_hashMaps.ContainsKey(newHashMapHash))

{

// Находим старый элемент с существующим хешем

newHashMapList = \_hashMaps[newHashMapHash]?.ToList();

// Пытаемся найти старый элемент среди списка с существующим ключом

var oldItemWithTheSameKey = newHashMapList.SingleOrDefault(value => EqualityComparer<K>.Default.Equals(value.Key, key));

// Если такой ключ уже существует, то вызываем ошибку и не добавляем значение

if (oldItemWithTheSameKey != null)

{

throw new ArgumentException($"Ключ должен быть уникален!: ", nameof(key));

}

// Добавляем в существующий элемент словаря новое значение с уникальным ключом

\_hashMaps[newHashMapHash].Add(newHashMapItem);

}

else

{

// Создаем новый элемент словаря с уникальным ключом

newHashMapList = new List<HashMap<K, V>> { newHashMapItem };

\_hashMaps.Add(newHashMapHash, newHashMapList);

}

}

// Метод для удаления по ключу

public void Delete(K key)

{

var hash = GetHash(key);

if (!\_hashMaps.ContainsKey(hash))

{

throw new KeyNotFoundException($"Такого ключа нет: {nameof(key)}");

}

var oldHashTable = \_hashMaps[hash];

var item = oldHashTable.SingleOrDefault(item => EqualityComparer<K>.Default.Equals(item.Key, key));

// Если элемент найден, то удаляем

if (item != null)

{

oldHashTable.Remove(item);

}

}

public V Search(K key)

{

var hash = GetHash(key);

if (!\_hashMaps.ContainsKey(hash))

{

throw new KeyNotFoundException($"There is no such key: {nameof(key)}");

}

var oldHashTable = \_hashMaps[hash];

if (oldHashTable != null)

{

var item = oldHashTable.SingleOrDefault(item => EqualityComparer<K>.Default.Equals(item.Key, key));

if (item != null)

{

return item.Value;

}

}

// Возвращаем пустое значение

return default(V);

}

public int GetHash(K key)

{

return key.ToString().Length;

}

public void ShowHashTable(MyHashMap<K, V> hashTable, string title)

{

// Проверяем аргументы

if (hashTable == null)

{

throw new ArgumentNullException(nameof(hashTable));

}

if (string.IsNullOrEmpty(title))

{

throw new ArgumentNullException(nameof(title));

}

foreach (var hashMaps in hashTable.HashMaps)

{

// Выводим все значения хранимые под этим хешем.

foreach (var value in hashMaps.Value)

{

Console.WriteLine($"\t{value.Key} - {value.Value}");

}

}

Console.WriteLine();

}

}

static void Main()

{

// Инициилизируем хеш таблицу

MyHashMap<int, string> TestDictionary = new MyHashMap<int, string>();

// Добавим слова

TestDictionary.Add(1, "Petrushka");

TestDictionary.Add(2, "Morkovka");

TestDictionary.Add(3, "Svekla");

TestDictionary.Add(4, "Lyk");

TestDictionary.Add(5, "Basilik");

TestDictionary.Add(6, "Kartoshka");

// Выводим хешированную таблицу

Console.WriteLine("Изначальная таблица: ");

TestDictionary.ShowHashTable(TestDictionary, "nameTable");

// Ищем по ключу (Basilik)

Console.WriteLine($"Результат поиска по ключу (5): {TestDictionary.Search(5)}\n");

// Удаляем значение с ключом (1)

TestDictionary.Delete(1);

// Результат после удаления

Console.WriteLine("Результат после удаления: ");

TestDictionary.ShowHashTable(TestDictionary, "nameTable");

}

}

}

**Выполнение:**

