БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Лабораторная работа № 4

по дисциплине «Структуры и алгоритмы обработки данных»

Вариант № 4

Выполнил студент: Ефименко Павел Викторович,

группа 981063

Минск 2020

**Условие:**

На основе динамических списков необходимо реализовать словарь. Основные операции, выполняемые над данными словаря: поиск, вставка и удаление. В сочетании со списками для построения словарей необходимо использовать открытое хеширование данных.

**Листинг:**

public class HashTable : IHashTable

{

/// <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));

}

// Получаем длину строки.

var hash = value.Length;

return hash;

}

}

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;

}

}

class Program

{

static void Main(string[] args)

{

// Создаем новую хеш таблицу.

var hashTable = new HashTable();

// Добавляем данные в хеш таблицу в виде пар Ключ-Значение.

hashTable.Insert("Little Prince", "I never wished you any sort of harm; but you wanted me to tame you...");

hashTable.Insert("Fox", "And now here is my secret, a very simple secret: It is only with the heart that one can see rightly; what is essential is invisible to the eye.");

hashTable.Insert("Rose", "Well, I must endure the presence of two or three caterpillars if I wish to become acquainted with the butterflies.");

hashTable.Insert("King", "He did not know how the world is simplified for kings. To them, all men are subjects.");

// Выводим хранимые значения на экран.

ShowHashTable(hashTable, "Created hashtable.");

Console.ReadLine();

// Удаляем элемент из хеш таблицы по ключу

// и выводим измененную таблицу на экран.

hashTable.Delete("King");

ShowHashTable(hashTable, "Delete item from hashtable.");

Console.ReadLine();

// Получаем хранимое значение из таблицы по ключу.

Console.WriteLine("Little Prince say:");

var text = hashTable.Search("Little Prince");

Console.WriteLine(text);

Console.ReadLine();

}

/// <summary>

/// Вывод хеш-таблицы на экран.

/// </summary>

/// <param name="hashTable"> Хеш таблица. </param>

/// <param name="title"> Заголовок перед выводом таблицы. </param>

private static void ShowHashTable(HashTable hashTable, string title)

{

// Проверяем входные аргументы.

if (hashTable == null)

{

throw new ArgumentNullException(nameof(hashTable));

}

if (string.IsNullOrEmpty(title))

{

throw new ArgumentNullException(nameof(title));

}

// Выводим все имеющие пары хеш-значение

Console.WriteLine(title);

foreach (var item in hashTable.Items)

{

// Выводим хеш

Console.WriteLine(item.Key);

// Выводим все значения хранимые под этим хешем.

foreach (var value in item.Value)

{

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

}

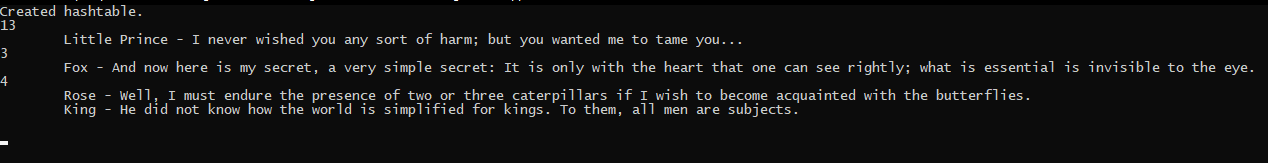
}

Console.WriteLine();

}

}

**Выполнение:**

****