БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра Информационных технологий

Факультет Компьютерных технологий

Специальность Программное обеспечение информационных технологий

Лабораторная работа № 4

по дисциплине «Структуры и алгоритмы обработки данных»

Вариант № 14

Выполнил студент: Шичко Павел Викторович

группа 981064

Минск 2020

Задание №1:

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.

Решение:

static void Main(string[] args)  
{  
 var hashTable = new HashTable();  
 hashTable.Insert("1", "Первый элемент");  
 hashTable.Insert("2", "Второй элемент");  
 hashTable.Insert("3", "Третий элемент");  
 hashTable.Insert("4", "Четвертый элемент");  
 Console.WriteLine("Введите 5 элемент ключ/значение");  
 hashTable.Insert(Console.ReadLine(), Console.ReadLine());  
 ShowHashTable(hashTable, "Создание hashtable");  
  
 Console.WriteLine("Поиск элемента hashtable");  
 Console.WriteLine("Введите ключ элемента которого хотите найти");  
 var text = hashTable.Search(Console.ReadLine());  
 Console.WriteLine(text);  
  
 Console.WriteLine("Введите ключ элемента которого хотите удалить");  
 hashTable.Delete(Console.ReadLine());  
 ShowHashTable(hashTable, "Удаление элемента из hashtable");  
 Console.WriteLine(text);  
}  
  
private static void ShowHashTable(HashTable hashTable, string title)  
{  
 if (hashTable == null)  
 {  
 throw new ArgumentNullException(nameof(hashTable));  
 }  
  
 if (string.IsNullOrEmpty(title))  
 {  
 throw new ArgumentNullException(nameof(title));  
 }  
  
  
 Console.WriteLine(title);  
 foreach (var (key, items) in hashTable.Items.OrderBy(x => x.Key))  
 {  
 Console.WriteLine($"Хеш-значение: {key}");  
 Console.WriteLine("--------------------------");  
  
 foreach (var value in items)  
 {  
 Console.WriteLine($"\t Ключ: {value.Key}; значение: {value.Value}");  
 }  
  
 Console.WriteLine("--------------------------");  
 }  
  
 Console.WriteLine();  
}  
  
  
public class Item  
{  
 public string Key { get; }  
 public string Value { get; }  
 public Item(string key, string value)  
 {  
 if (string.IsNullOrEmpty(key))  
 {  
 throw new ArgumentNullException(nameof(key));  
 }  
  
 if (string.IsNullOrEmpty(value))  
 {  
 throw new ArgumentNullException(nameof(value));  
 }  
   
 Key = key;  
 Value = value;  
 }  
  
  
 public override string ToString()  
 {  
 return Key;  
 }  
}  
  
public class HashTable  
{  
 private readonly byte \_maxSize = 255;  
  
 private Dictionary<int, List<Item>> \_items = null;  
  
 public IEnumerable<KeyValuePair<int, List<Item>>> Items => \_items?.ToList()?.AsReadOnly();  
  
 public HashTable()  
 {  
 \_items = new Dictionary<int, List<Item>>(\_maxSize);  
 }  
  
 public void Insert(string key, string value)  
 {  
 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));  
 }  
  
 var item = new Item(key, value);  
  
 var hash = GetHash(item.Key);  
  
 List<Item> hashTableItem;  
 if (\_items.ContainsKey(hash))  
 {  
 hashTableItem = \_items[hash];  
  
 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));  
 }  
  
 \_items[hash].Add(item);  
 }  
 else  
 {  
 hashTableItem = new List<Item> {item};  
  
 \_items.Add(hash, hashTableItem);  
 }  
 }  
  
  
 public void Delete(string key)  
 {  
 if (string.IsNullOrEmpty(key))  
 {  
 throw new ArgumentNullException(nameof(key));  
 }  
  
 if (key.Length > \_maxSize)  
 {  
 throw new ArgumentException($"Максимальная длинна ключа составляет {\_maxSize} символов.",  
 nameof(key));  
 }  
  
 var hash = GetHash(key);  
  
 if (!\_items.ContainsKey(hash))  
 {  
 return;  
 }  
  
 var hashTableItem = \_items[hash];  
  
 var item = hashTableItem.SingleOrDefault(i => i.Key == key);  
  
 if (item != null)  
 {  
 hashTableItem.Remove(item);  
 }  
 }  
  
  
 public string Search(string key)  
 {  
 if (string.IsNullOrEmpty(key))  
 {  
 throw new ArgumentNullException(nameof(key));  
 }  
  
 if (key.Length > \_maxSize)  
 {  
 throw new ArgumentException($"Максимальная длинна ключа составляет {\_maxSize} символов.",  
 nameof(key));  
 }  
  
  
 var hash = GetHash(key);  
  
 if (!\_items.ContainsKey(hash))  
 {  
 return null;  
 }  
  
 var hashTableItem = \_items[hash];  
  
 var item = hashTableItem?.SingleOrDefault(i => i.Key == key);  
  
 return item?.Value;  
 }  
  
  
 private int GetHash(string value)  
 {  
 if (string.IsNullOrEmpty(value))  
 {  
 throw new ArgumentNullException(nameof(value));  
 }  
  
 if (value.Length > \_maxSize)  
 {  
 throw new ArgumentException($"Максимальная длинна ключа составляет {\_maxSize} символов.",  
 nameof(value));  
 }  
  
 var hash = value.Length;  
 return hash;  
 }  
}

