Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Институт информационных технологий

Факультет компьютерных технологий

Лабораторная работа № 14

Вариант № 14

Выполнил: студент гр. 981064 Шичко П.В.

Минск 2020

Вариант № 14

Задание № 1.

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.

Код программы:

class Program  
{  
 static void Main(string[] args)  
 {  
 Console.WriteLine("Введите название словоря:");  
 var title = Console.ReadLine();  
  
 var library = new CustomHashMap<string, string>();  
 library.Insert("Key1", "Data");  
 library.Insert("Key2", "Data");  
 Console.WriteLine("Поиск: {0}", library.Search("Key1"));  
  
 library.ShowHashTable(library, title);  
 library.Delete("Key1");  
 Console.WriteLine("После удаленя:");  
 library.ShowHashTable(library, title);  
 Console.ReadLine();  
 }  
  
 public class ValidateClass<TK, TV>  
 {  
 */// <summary>  
 /// Максимальный размер ключа.  
 /// </summary>* private readonly byte \_maxKeySize = 255;  
  
 public void ValidateEntryParams(TK key, TV value)  
 {  
 if (key == null || value == null)  
 {  
 throw new ArgumentNullException(key == null ? nameof(key) : nameof(value));  
 }  
 }  
  
 public void ValidateKey(TK key)  
 {  
 if (key == null)  
 {  
 throw new ArgumentNullException(nameof(key));  
 }  
 }  
  
 public void ValidateKeySize(TK key)  
 {  
 if (key.GetHashCode().ToString().Length > \_maxKeySize)  
 {  
 throw new ArgumentException($"Максимальная длина ключа составляет {\_maxKeySize} символов.");  
 }  
 }  
 }  
  
 public class HashMapItem<TK, TV>  
 {  
 *// Ключ.* public TK Key { get; private set; }  
  
 *// Значение.* public TV Value { get; private set; }  
  
  
 *// Создание нового значения.* public HashMapItem(TK key, TV value)  
 {  
 var validateClass = new ValidateClass<TK, TV>();  
 validateClass.ValidateEntryParams(key, value);  
  
 Key = key;  
 Value = value;  
 }  
  
 public override string ToString()  
 {  
 return Value.ToString();  
 }  
 }  
  
 public class CustomHashMap<TK, TV>  
 {  
 private readonly byte \_maxKeySize = 255;  
  
 private Dictionary<int, IList<HashMapItem<TK, TV>>> \_hashMapItems = null;  
  
 public IReadOnlyCollection<KeyValuePair<int, IList<HashMapItem<TK, TV>>>> HashMapItems =>  
 \_hashMapItems?.ToList()?.AsReadOnly();  
  
 public CustomHashMap()  
 {  
 *// Инициализируем словарь с ограничением максимальной длине ключа.* \_hashMapItems = new Dictionary<int, IList<HashMapItem<TK, TV>>>(\_maxKeySize);  
 }  
  
  
 *// Метод добавления новых значений в хеш-таблицу.  
 // Элементы с уникальным хешем(длиной строки) добавляем новым элементом в общий словарь.  
 // Элементы с уникальным ключом, но не уникальным хешем добавляем в существующий элемент словаря.* public void Insert(TK key, TV value)  
 {  
 *// Создаем экземпляр класса валидации.* var validate = new ValidateClass<TK, TV>();  
 validate.ValidateEntryParams(key, value);  
  
 *// Проверяем длину ключа.* validate.ValidateKeySize(key);  
  
 *// Создаём новый элемент.* var newHashMapItem = new HashMapItem<TK, TV>(key, value);  
  
 *// Создаём новых хеш ключа.* var newHashMapItemHash = GetHash(key);  
  
 List<HashMapItem<TK, TV>> newHashMapItemList = new List<HashMapItem<TK, TV>>();  
 if (\_hashMapItems.ContainsKey(newHashMapItemHash))  
 {  
 *// Находим старый элемент с существующим хешем.* newHashMapItemList = \_hashMapItems[newHashMapItemHash]?.ToList();  
  
 *// Пытаемся найти старый элемент с существующим ключом.* var oldItemWithTheSameKey =  
 newHashMapItemList.SingleOrDefault(obj => EqualityComparer<TK>.Default.Equals(obj.Key, key));  
  
 *// Если такой ключ уже существует, то выбрасываем ошибку и не добавляем значение.* if (oldItemWithTheSameKey != null)  
 {  
 throw new ArgumentException(  
 $"Хеш-таблица уже содержит элемент с ключом {key}. Ключ должен быть уникален.",  
 nameof(key));  
 }  
  
 *// Добавляем в существующий элемент словаря новое значение с уникальным ключом.* \_hashMapItems[newHashMapItemHash].Add(newHashMapItem);  
 }  
 else  
 {  
 *// Создаем новый элемент словаря с уникальным ключом.* newHashMapItemList = new List<HashMapItem<TK, TV>> {newHashMapItem};  
 \_hashMapItems.Add(newHashMapItemHash, newHashMapItemList);  
 }  
 }  
  
 *// Метод удаления по ключу.* public void Delete(TK key)  
 {  
 *// Создаем экземпляр класса валидации.* var validate = new ValidateClass<TK, TV>();  
  
 *// Проверяем наличие ключа.* validate.ValidateKey(key);  
  
 *// Проверяем размер ключа.* validate.ValidateKeySize(key);  
  
 var hash = GetHash(key);  
  
 if (!\_hashMapItems.ContainsKey(hash))  
 {  
 throw new KeyNotFoundException($"There is no such key: {nameof(key)}");  
 }  
  
 var oldHashTableItem = \_hashMapItems[hash];  
  
 var item = oldHashTableItem.SingleOrDefault(obj => EqualityComparer<TK>.Default.Equals(obj.Key, key));  
  
 *// Если элемент найден - удаляем.* if (item != null)  
 {  
 oldHashTableItem.Remove(item);  
 }  
 }  
  
 public TV Search(TK key)  
 {  
 *// Создаем экземпляр класса валидации.* var validate = new ValidateClass<TK, TV>();  
  
 validate.ValidateKey(key);  
  
 validate.ValidateKeySize(key);  
  
 var hash = GetHash(key);  
  
 if (!\_hashMapItems.ContainsKey(hash))  
 {  
 throw new KeyNotFoundException($"There is no such key: {nameof(key)}");  
 }  
  
 var oldHashTableItem = \_hashMapItems[hash];  
 if (oldHashTableItem != null)  
 {  
 var item = oldHashTableItem.SingleOrDefault(obj =>  
 EqualityComparer<TK>.Default.Equals(obj.Key, key));  
  
 if (item != null)  
 {  
 return item.Value;  
 }  
 }  
  
 *// Возвращаем пустое значение.* return default(TV);  
 }  
  
 public int GetHash(TK key)  
 {  
 *// Создаем экземпляр класса валидации.* var validate = new ValidateClass<TK, TV>();  
  
 validate.ValidateKey(key);  
  
 validate.ValidateKeySize(key);  
  
 return key.ToString().Length;  
 }  
  
 public void ShowHashTable(CustomHashMap<TK, TV> 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 hashMapItems in hashTable.HashMapItems)  
 {  
 *// Выводим хеш* Console.WriteLine("Хеш: {0}", hashMapItems.Key);  
  
 *// Выводим все значения хранимые под этим хешем.* foreach (var value in hashMapItems.Value)  
 {  
 Console.WriteLine($"\t{value.Key} - {value.Value}");  
 }  
 }  
  
 Console.WriteLine();  
 }  
 }

