МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Информационные системы»

**Лабораторная работа №4**по дисциплине «Основы алгоритмизации и программирования»

Выполнил:  
студент(ка) гр. ПИбд-12 Дозорова Алена

Ульяновск  
2021

Задание на лабораторную работу:

1. Реализовать хэш-таблицу с:

Вариант 6: двойным хэшированием

1. Реализовать хэш-таблицу с:

Вариант 6: методом цепочек, где элементы таблицы – односвязные списки

Краткое описание используемых алгоритмов и структур

В программе реализовано два класса: Hash, где реализована хэш таблица на базе массива с двойным хэшированием и HashList, где реализована хэш-таблица на основе листа с использованием односвязного списка.

Код лабораторной работы:

public class Hash

{

public string[] Mas;

int ras;

public Hash(int size)

{

Mas = new string[size];

ras = size;

for(int i = 0; i<size; i++)

{

Mas[i] = "";

}

}

public void Print()

{

for (int i = 0; i < ras; i++)

{

if (Mas[i] == "")

{

Console.Write(" . ");

}

else Console.Write(Mas[i] + " ");

}

}

public int Key(int v)

{

int k = v+1;

return k<ras ? k : Math.Abs(k-ras);

}

public int Key2(int v)

{

int k = v+3;

return k<ras? k : Math.Abs(ras - k + 1);

}

public void Add(int key, string vulue)

{

if(Mas[key]!="")

{

int k = 1;

while(Mas[key]!="")

{

key = Key(key) + k \* Key2(key);

k++;

}

}

Mas[key] = vulue;

}

public int Search(int key, string val)

{

if (Mas[key]==val)

{

return key;

}

else

{

int k = 1;

while (key < ras && k < ras && Mas[key]!=val)

{

key = Key(key) + k \* Key2(key);

k++;

}

if (key >= ras || Mas[key] != val) return -1;

else return key;

}

}

public void Delete(int key)

{

Mas[key] = "";

}

}

public class HashList

{

List<Linked<int>> list = new List<Linked<int>>(10);

public HashList()

{

Random r = new Random();

for (int i = 0; i < 10; i++)

{

Linked<int> l = new Linked<int>();

list.Add(l);

}

for (int i = 0; i < 10; i++)

{

for (int j = 0; j < 10; j++)

{

list[Key(i + j)].Add(r.Next(100) + 1);

}

}

}

public int Key(int v)

{

int k = v+2;

while (k>=10)

{

k = k % 10;

k = k / 10;

}

return k;

}

public void Add(int key, int val)

{

list[key].Add(val);

}

public void Print()

{

for (int i = 0; i < list.Count; i++)

{

Console.Write((i+1)+". ");

list[i].Print();

Console.WriteLine("\n");

}

}

}

public class Node<T>

{

public Node(T data)

{

Data = data;

}

public T Data { get; set; }

public Node<T> Next { get; set; }

}

public class Linked<T> // односвязный список

{

Node<T> head; // головной/первый элемент

Node<T> tail; // последний/хвостовой элемент

int count; // количество элементов в списке

public void Print()

{

Node<T> current = head;

while (current != null)

{

Console.Write(current.Data + " ");

current = current.Next;

}

}

// добавление элемента

public void Add(T data)

{

Node<T> node = new Node<T>(data);

if (head == null)

head = node;

else

tail.Next = node;

tail = node;

count++;

}

// удаление элемента

public bool Remove(T data)

{

Node<T> current = head;

Node<T> previous = null;

while (current != null)

{

if (current.Data.Equals(data))

{

// Если узел в середине или в конце

if (previous != null)

{

// убираем узел current, теперь previous ссылается не на current, а на current.Next

previous.Next = current.Next;

// Если current.Next не установлен, значит узел последний,

// изменяем переменную tail

if (current.Next == null)

tail = previous;

}

else

{

// если удаляется первый элемент

// переустанавливаем значение head

head = head.Next;

// если после удаления список пуст, сбрасываем tail

if (head == null)

tail = null;

}

count--;

return true;

}

previous = current;

current = current.Next;

}

return false;

}

public int Count { get { return count; } }

public bool IsEmpty { get { return count == 0; } }

// очистка списка

public void Clear()

{

head = null;

tail = null;

count = 0;

}

// содержит ли список элемент

public bool Contains(T data)

{

Node<T> current = head;

while (current != null)

{

if (current.Data.Equals(data))

return true;

current = current.Next;

}

return false;

}

// добвление в начало

public void AppendFirst(T data)

{

Node<T> node = new Node<T>(data);

node.Next = head;

head = node;

if (count == 0)

tail = head;

count++;

}

}

static void Main(string[] args)

{

Console.WriteLine("Введите размер таблицы: ");

int s = Convert.ToInt32(Console.ReadLine());

Hash h = new Hash(s);

Console.WriteLine("Введите добавляемый элемент");

string add = Console.ReadLine();

h.Add(h.Key(add.Length), add);

Console.WriteLine("Введите добавляемый элемент");

add = Console.ReadLine();

h.Add(h.Key(add.Length), add);

h.Print();

Console.WriteLine("\nВведите искомый элемент: ");

string search = Console.ReadLine();

Console.WriteLine(h.Search(h.Key(search.Length), search));

Console.WriteLine("Удалим последний добавленный элемент\n");

h.Delete(h.Key(add.Length));

h.Print();

Console.WriteLine("\nВыведем хэш использующий метод цепочек для разрешения коллизий\n");

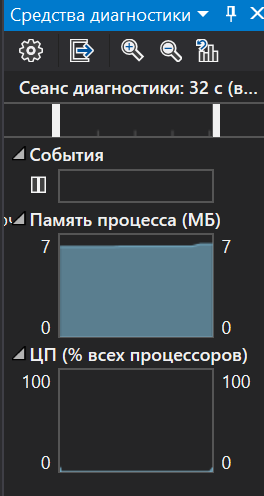
HashList l = new HashList();

l.Print();

Console.ReadKey();

}

Результаты тестирования:



Выводы:

Я изучила алгоритмы хеширования данных и такую структуру данных как хэш-таблица.