Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Институт информационных технологий

Факультет компьютерных технологий

Кафедра информационных систем и технологий

**Лабораторная работа 14**

по дисциплине

«Алгоритмы и структуры данных»

|  |  |
| --- | --- |
|  | Выполнил студент группы 181072  Матвеев Артур Андреевич |
|  | Проверил Потоцкий Дмитрий Сергеевич |

Минск 2022

**Индивидуальные задания**

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.

using AISD\_LR14;

var rnd = new Random();

var current = new HashFile();

int value = default;

//Создание сегмента

for (int i = 0; i < 20; i++)

{

value = rnd.Next(1, 100);

//Добавляем / связываем блоки

current.Add(value);

}

current.ShowAll();

Console.WriteLine("\n\rSearch");

current.Search(500);

current.Search(value);

Console.WriteLine($"Remove {value}");

current.Remove(value);

current.Remove(value);

current.ShowAll();

namespace AISD\_LR14

{

public record HashFile

{

//ключ блока

public byte? HashKey { get; set; }

//Значения блока

public List<int> Value { get; set; }

//ссылка на следующий блок

public HashFile Next { get; set; }

public HashFile()

{

Value = new List<int>();

}

public void ShowAll()

{

var current = this;

//вывод значений первого блока

PrintValue(current);

while (current.Next is not null)

{

current = current.Next;

//вывод значений текущего блока

PrintValue(current);

}

Console.WriteLine();

}

public void Add(int value)

{

var hash = GetHash(value);

var current = this;

//проверка первого элемента

if (current.HashKey is null || current.HashKey == hash)

{

current.HashKey = hash;

current.Value.Add(value);

return;

}

while (current.Next is not null)

{

current = current.Next;

//ключ текущего блока не равен хешу нового элемента

if (current.HashKey != hash)

continue;

//проверка на наличие этого элемента в блоке

if (!current.Value.Any(x => x == value))

{

current.Value.Add(value);

}

else

{

Console.WriteLine($"Значение {value} уже существует!");

}

return;

}

//создание нового элемента/блока

var newHashFile = new HashFile()

{

HashKey = hash,

Value = new List<int>() { value }

};

//присваение ссылки на новый эл. у последнего блока

current.Next = newHashFile;

}

public void Search(int value)

{

var hash = GetHash(value);

var current = this;

//поиск значения у первого блока, если ключ равен хешу

if (current.HashKey == hash)

{

SearchInCurrentHashFile(value, current);

return;

}

while (current.Next is not null)

{

current = current.Next;

if (current.HashKey != hash)

continue;

SearchInCurrentHashFile(value, current);

return;

}

//блока с данным значением нет

Console.WriteLine("Not Found");

}

public void Remove(int value)

{

var hash = GetHash(value);

var current = this;

if (current.HashKey == hash)

{

current.Value.Remove(value);

return;

}

while (current.Next is not null)

{

current = current.Next;

if (current.HashKey == hash && current.Value.Any(x => x == value))

{

current.Value.Remove(value);

return;

}

}

Console.WriteLine("Элемент не существует");

}

//поиск значения в текущем блоке

private void SearchInCurrentHashFile(int value, HashFile current)

{

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

{

if (current.Value[i] == value)

{

Console.WriteLine($"HashKey: {current.HashKey} Value {value}");

return;

}

}

Console.WriteLine("Not Found");

}

//созданите хеша

private byte GetHash(int number) =>

byte.Parse(number.ToString().ToCharArray().FirstOrDefault().ToString());

private void PrintValue(HashFile current)

{

if (current.HashKey is null)

return;

Console.Write("\n\rHashKey: " + current.HashKey + " Value: ");

foreach (var item in current.Value)

{

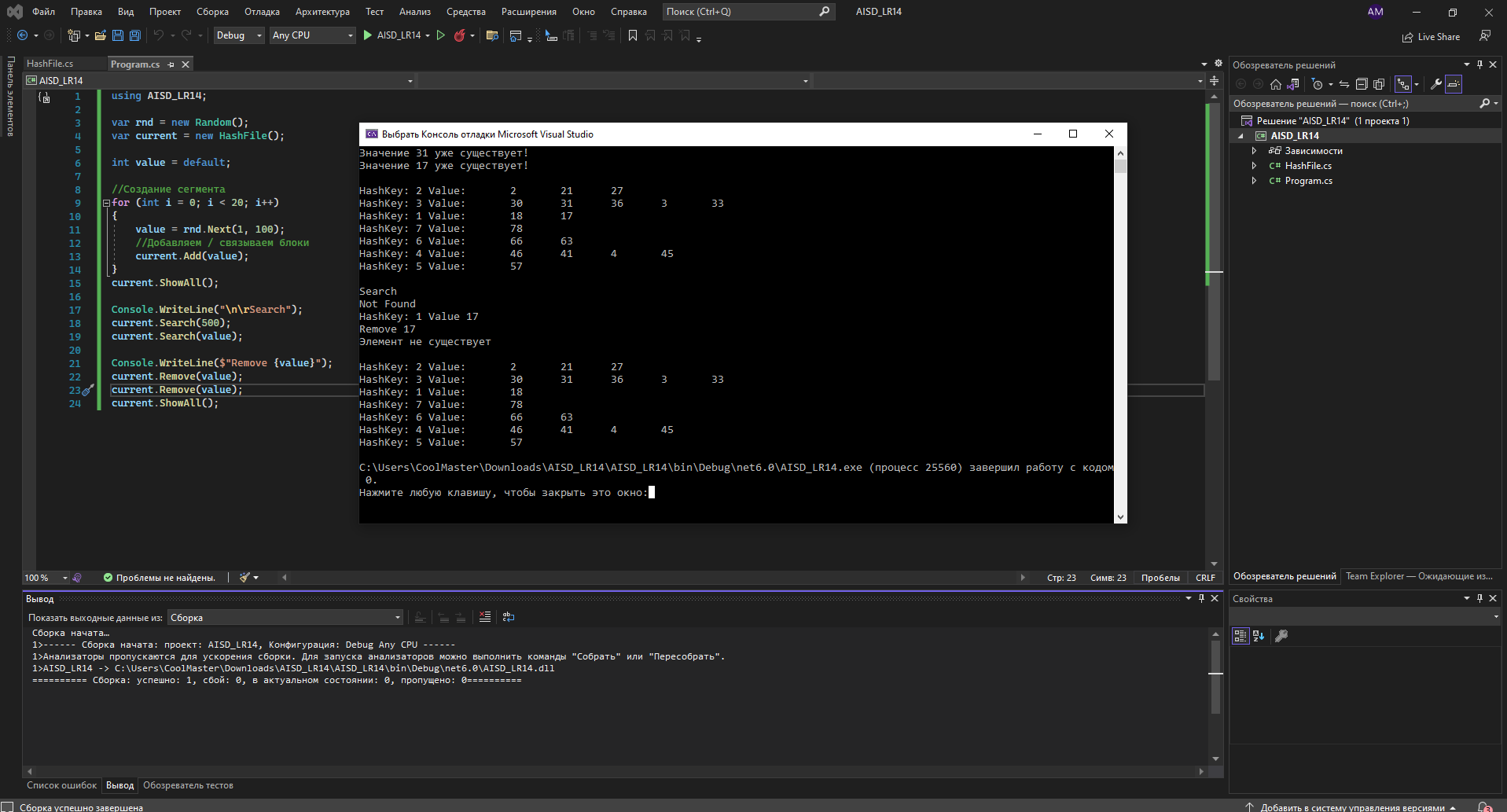
Console.Write("\t" + item);

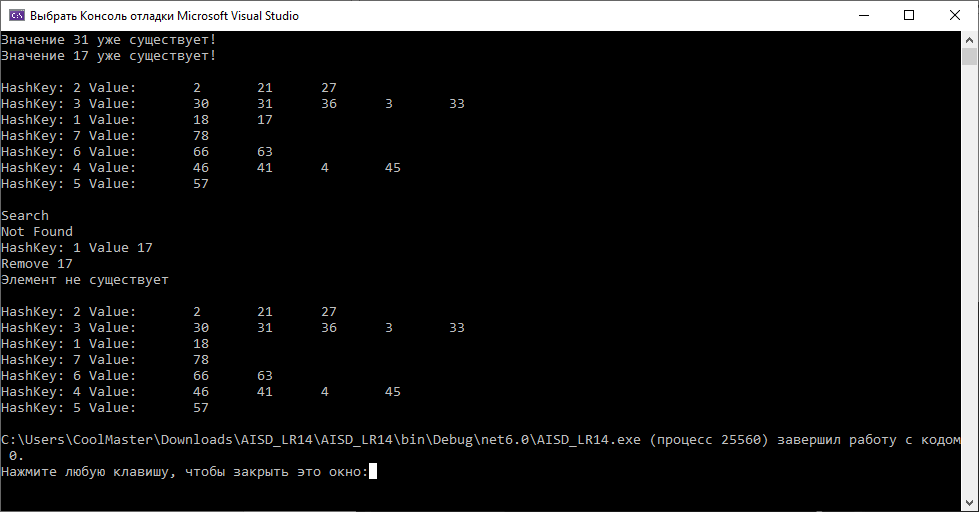
}

}

}

}





1. В чем заключаются особенности операций с данными, хранящимися во внешней памяти?

Природа устройств внешней памяти такова, что время, необходимое для поиска блока и чтения его в основную память, достаточно велико по сравнению со временем, которое требуется для относительно простой обработки данных, содержащихся в этом блоке.

Допустим, имеется блок из 1000 целых чисел на диске, вращающемся со скоростью 1000 об./мин. Время, которое требуется для позиционирования считывающей головки над дорожкой, содержащей этот блок (время установки головок), плюс время, затрачиваемое на ожидание, пока требуемый блок сделает оборот и окажется под головкой (время ожидания), может в среднем составлять 100 миллисекунд. Процесс записи блока в определенное место во вторичной памяти занимает примерно столько же времени. При этом за те же 100 миллисекунд машина, как правило, успевает выполнить 100000 команд. Этого времени вполне достаточно, чтобы выполнить простую обработку 1000 целых числе, когда они находятся в основной памяти.

Оценивая время работы алгоритмов, в которых используются данные, хранящиеся в виде файлов, приходится в первую очередь учитывать количество обращений к блокам, т.е. сколько раз блок считывается в основную память или записывается во вторичную память. Такая операция называется доступом к блоку. Поскольку размер блока фиксирован в операционной системе, нет возможности ускорить работу алгоритма, увеличив размер блока и сократив тем самым количество обращений к блокам. Поэтому мерой качества алгоритма, работающего с внешней памятью, является количество обращений к блокам.

1. Что общего и различного у хешированных файлов и алгоритма открытого хеширования, предназначенного для работы с данными в оперативной памяти.

Хешированные файлы во много напоминают метод открытого хеширования, для работы с данными в оперативной памяти. Отличие заключается в том, что в случае хешированных файлов элементы, хранящиеся в одном блоке сегмента, не требуется связывать друг с другом с помощью указателей, связывать между собой нужно только блоки.

3.  Почему удается ускорить обработку данных при использовании индексированных файлов.

Вывод: в данной лабораторной работе был писан алгоритм хеширования файлов.