Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Основы алгоритмизации и программирования (ОАиП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

АНАЛИЗ ХРАНЕНИЯ И ОБРАБОТКИ БОЛЬШИХ ОБЪЕМОВ ИНФОРМАЦИИ МЕТОДОМ ХЕШИРОВАНИЯ

БГУИР КР 1-40 01 01 204 ПЗ

Студент: гр. 951002 Будович И.В.

Руководитель:

асс. Фадеева Е.Е.

Минск 2020

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой ПОИТ

––––––––––––––––––––––––

(подпись)

––––––––––––––––– 2020 г.

ЗАДАНИЕ

по курсовому проектированию

Студенту Будовичу Ивану Витальевичу

1. Тема работы Анализ хранения и обработки больших объемов информации методом хеширования

2. Срок сдачи студентом законченной работы––01.06.2020 г.–––

3. Исходные данные к работе язык программирования Delphi, файл записей для ассоциативного массива, файл ключей для поиска

4. Содержание расчётно-пояснительной записки (перечень вопросов, которые подлежат разработке)

Введение.

1. Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству;

2. Анализ требований к программному средству и разработка функциональных требований;

3. Проектирование программного средства;

4. Создание (конструирование) программного средства;

5. Тестирование, проверка работоспособности и анализ полученных результатов;

6. Руководство по установке и использованию;

Список используемой литературы

Заключение

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

1. "Анализ хранения и обработки больших объемов информации методом хеширования ", А1, схема программы, чертеж.

6. Консультант по курсовой работе

Фадеева Е.Е.

7. Дата выдачи задания 10.02.2020

8. Календарный график работы над курсовой работой на весь период проектирования (с обозначением сроков выполнения и процентом от общего объёма работы):

раздел 1 к 01.03.2020 – 15 % готовности работы;

разделы 2, 3 к 15.03.2020 – 30 % готовности работы;

разделы 4, 5 к 15.04.2020 – 60 % готовности работы;

раздел 6 к 15.05.2020 – 90 % готовности работы;

оформление пояснительной записки и графического материала к 20.05.2020 – 100 % готовности работы.

Защита курсовой работы с 01.06.2020 по 09.06.2020 г.–––––––––––––––––––––

РУКОВОДИТЕЛЬ–––––– Е.Е.Фадеева

(подпись)

Задание принял к исполнению 10.02.2020–––\_\_\_\_––

(дата и подпись студента)

Содержание

[Содержание 4](#_Toc42457237)

[Введение 6](#_Toc42457238)

[1 Анализ прототипов, литературных источников и формирование требований к проектируемому программному 7](#_Toc42457239)

[1.1 Сфера применения хеширования 7](#_Toc42457240)

[1.2 Размер пакета 8](#_Toc42457241)

[1.3 Плотность заполнения 8](#_Toc42457242)

[1.4 Хеш-функции 9](#_Toc42457243)

[1.5 Организация области переполнения 10](#_Toc42457244)

[1.6 Анализ прототипов 11](#_Toc42457245)

[1.7 Постановка задачи 15](#_Toc42457246)

[2 Анализ предметной области и разработка функциональных требований 16](#_Toc42457247)

[2.1 Описание функциональных требований 16](#_Toc42457248)

[2.2 Описание прочих требований 17](#_Toc42457249)

[3 Проектирование программного средства 18](#_Toc42457250)

[3.1 Разработка алгоритмов хеширования 18](#_Toc42457251)

[3.2 Разработка алгоритмов заполнения и поиска 21](#_Toc42457252)

[3.3 Разработка алгоритма сбора данных 22](#_Toc42457253)

[3.4 Разработка алгоритма подсчета для круговых диаграмм 22](#_Toc42457254)

[4 Конструирование программного средства 23](#_Toc42457255)

[4.1 Разработка динамических структур данных 23](#_Toc42457256)

[4.2 Работа с файлами 23](#_Toc42457257)

[4.3 Взаимосвязь между компонентами формы 24](#_Toc42457258)

[4.4 Структура модулей программы 25](#_Toc42457259)

[4.5 Описание модуля MainFormUnit 25](#_Toc42457260)

[4.6 Описание модуля HashAlgorithm 30](#_Toc42457261)

[4.7 Описание модуля Types 36](#_Toc42457262)

[5 Тестирование, проверка работоспособности и анализ полученных результатов 39](#_Toc42457263)

[5.1 Тестирование алгоритмов хеширования 39](#_Toc42457264)

[5.2 Тестирование компонентов формы 41](#_Toc42457265)

[5.3 Анализ алгоритмов хеширования 44](#_Toc42457266)

[6 Руководство по установке и использованию 47](#_Toc42457267)

[Заключение 50](#_Toc42457268)

[Список литературы 51](#_Toc42457269)

[Приложение A 52](#_Toc42457270)

[Приложение Б 53](#_Toc42457271)

[Приложение В 60](#_Toc42457272)

[Код модуля Types 60](#_Toc42457273)

[Код модуля HashAlgorithm 61](#_Toc42457274)

[Код модуля MainFormUnit 70](#_Toc42457275)

Введение

В современном мире с развитием информатизации значительно увеличивается объем данных, обрабатываемых и хранимых информационными системами. Развиваются базы данных и языки программирования, использующие новые структуры данных, такие как хеш-таблицы.

Идея хеширования берет своё начало еще в эпоху Возрождения, когда ученые, сомневаясь в правильности своего открытия, зашифровывали его с возможностью последующего подтверждения. Подобный метод шифрования получил название хеширования. В настоящее время хеширование получило свое распространение в организации баз данных, криптографии и высокоуровневом программировании (при проектировании словарей).

Цель данной курсовой работы – оценить эффективность разных методов организации хеширования при замещении данных и найти оптимальное решение, которое позволит наиболее эффективно использовать ресурсы системы.

В ходе выполнения курсовой работы я постараюсь найти решение таким техническим вопросам как:

1. Реализация алгоритмов хеширования и методов организации хеш-таблиц.
2. Проектирование удобного пользовательского графического интерфейса в среде Embarcadero Delphi 10;
3. Разбиение проекта на отдельные модули;

В этой пояснительной записке отображены следующие этапы написания курсовой работы:

1. Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству;
2. Анализ требований к программному средству и разработка функциональных требований;
3. Проектирование программного средства;
4. Создание (конструирование) программного средства;
5. Тестирование, проверка работоспособности и анализ полученных результатов;
6. Руководство по установке и использованию.
7. Анализ прототипов, литературных источников и формирование требований к проектируемому программному
   1. Сфера применения хеширования

Часто бывают нужны множества, поддерживающие только «словарные операции» добавления, поиска и удаления элемента. В это случае применяется хеширование; соответствующая структура данных называется хеш-таблица. В худшем случае поиск в хеш-таблице может занимать столько же времени, сколько в списке , но на практике хеширование весьма эффективно. При выполнении некоторых естественных условий математическое ожидание времени поиска элемента в хеш-таблице есть [1]

Хеширование – один из методов, позволяющих по первичному ключу записи получить её физический адрес. [2]

Ключ записи преобразуется в квазислучайное число, которое используется для местоположения записи. Число может указывать на адрес, по которому расположена запись или на область, в которой расположена группа записей. Данная область называется участком записей или пакетом записей. [2]

Адрес определяется следующим образом:

* Ключ записи преобразуется в квазислучайное число от 0 до N, где N – количество пакетов записей;
* Полученное число преобразуется в физический адрес пакета;
* Если в пакете есть свободное место, то запись располагается в нем;
* Если нет, размещается в области переполнения. [2]

Для того, чтобы получить представление о важности хеширования, хочу рассмотреть основные сферы применения данного метода при работе с данными.

* + 1. Базы данных

Хеширование нашло свое применение в базах данных связи с вышеупомянутым свойством быстрого доступа к данным. Оно применяются для адресации данных, хранящихся на устройствах с произвольным доступом, выпуск которых был начат в середине 50-х годов. [3] При этом данные могут храниться на внешних носителях, поэтому важно минимизировать число обращений. методы перемешивания и их применение являются довольно простыми и, кроме того, они имеют два существенных преимущества по сравнению с методами индексирования. Во-первых, большинство записей можно найти в результате одного обращения к внешнему запоминающему устройству. Во-вторых, включение и удаление записей осуществляются сравнительно легко. [3]

* + 1. Словари

Словарь – ассоциативный массив, структура данных, используемая в высокоуровневом программировании, позволяющая добавлять, редактировать, просматривать и удалять записи по ключу. Поддержка словарей присутствует во многих языках высокого уровня. Некоторые из них, например, Delphi (TDictionary) [4], Swift (Dictionary) [5], Python (dict) [6] используют хеширование.

В данном разделе я обращу основное внимание на следующие факторы:

1. Размер пакета;
2. Плотность заполнения, т. е. число участков записей определенного  
   размера, используемых для размещения файла заданного объема;
3. Алгоритм преобразования ключа в адрес;
4. Организацию работы с областями переполнения.
   1. Размер пакета

Если выбрать размер участка небольшим, то записи будут часто попадать в область переполнения, что приведет к увеличению времени доступа к ним. Выбор большого размера участка приведет к уменьшению числа записей, попадающих в область переполнения. [3] С другой стороны, поиск в большом пакете идет дольше. Необходимо найти оптимальный размер.

* 1. Плотность заполнения

Число записей, направляемых в область переполнения, зависит от плотности заполнения первичных участков памяти: чем выше плотность заполнения первичных участков, тем выше вероятность того, что вновь поступающие записи будут направлены в область переполнения. [3]

Плотность заполнения определяется как отношение числа записей, находящихся в участках, к максимальному числу записей, которое можно поместить в эти участки. Изменение плотности заполнения влияет на время доступа и объем памяти. Необходимо найти плотность, при достижении которой необходимо производить реорганизацию данных.

* 1. Хеш-функции

Хеш-функция, или алгоритм перемешивания, представляет собой алгоритм преобразования ключа в адрес. Вначале, если ключ не цифровой, он преобразуется в соответствующее цифровое представление таким образом, чтобы исключить потерю информации. Затем цифровое представление ключа преобразуется в совокупность произвольно распределенных чисел. Желательно, чтобы набор ключей был распределен как можно равномернее в диапазоне, переводимом в допустимые адреса. После чего полученные числа умножаются на константу, что позволяет разместить их строго в диапазоне значений адресов основной области. [3]

Лучшим является не тот метод, который обеспечивает именно случайное  
распределение записей, а тот, который обеспечивает равномерное заполнение  
всего адресного пространства памяти.[3]

Перечислю наиболее известные алгоритмы:

* + 1. Метод деления

Ключ делится на простое число или число, не имеющее малых делителей, близкое к значению к числу пакетов. Остаток от деления – относительный адрес пакета. [2] На практике метод деления используется в большинстве приложений, работающих с хешированием. [7]

* + 1. Метод средних квадратов

Ключ возводится в квадрат, выбираются центральные цифры. Умножаются на константу. [2]

* + 1. Метод сдвига разрядов

Числовое значение ключа делится на две части, младшая складывается со старшей. Данный процесс продолжается до тех пор, пока количество цифр результата не окажется равным количеству цифр числа пакетов. [2]

* + 1. Метод складывания

Число разбивается на три части; центральная часть содержит столько же цифр, сколько цифр в числе пакетов N, первая и третья заворачиваются и складываются. [2]

* + 1. Метод преобразования систем счисления

Преобразуется основание системы счисления. Число представляется в новой системе счисления. Последние цифры числа – относительный адрес пакета. [2]

* 1. Организация области переполнения

Существуют два подхода к организации хранения записей, направляемых в область переполнения.

При первом подходе записи располагаются в области переполнения, отделенной от основной области. К таким методам относятся метод цепочек и метод распределенной области переполнения. [3]

Второй подход заключается в хранении записей переполнения в пределах основной области. Таким является метод открытой адресации.

* + 1. Цепочки участков переполнения

Для связи основой области памяти с областью переполнения используются цепочки адресов. Т.е. пакет в основной области хранит адрес пакета (записи) в области переполнения. [3]

* + 1. Метод распределенной области переполнения

Пакеты переполнения размещены через определенные интервалы среди первичных пакетов. Если первичный пакет переполнен, направленная в него запись направляется в ближайший пакет переполнения, следующий за данным первичным пакетом. [2]

Алгоритмы перемешивания (хеширования) осуществляют преобразование ключа к номеру первичного пакета. Однако алгоритм, преобразующий номер первичного пакета в его машинный адрес, должен учитывать наличие пакетов переполнения между первичными пакетами. [2]

* + 1. Метод открытой адресации (Петерсона)

Происходит рассеивание записей переполнения в основной области. Если пакет переполнен, то запись помещается в следующий за ним пакет. Однако если размер пакетов невелик, то это приводит к необходимости последовательного перебора пакетов в поисках свободного места. При размерах пакета более 10 записей этот метод эффективен при периодической реорганизации БД. Позволяет увеличивать плотность заполнения. [3]

* 1. Анализ прототипов

Данная курсовая работа посвящена анализу методов алгоритмов хеширования. Для того, чтобы провести анализ, необходимо выполнить алгоритмы на некотором тестовом наборе, размер которого позволяет судить об объективности работы, а также представить информацию в удобном виде. Ниже перечислены способы отображения информации при анализе.

* + 1. Табличный вид

Часто результаты исследований отображаются с помощью электронных таблиц, таких как Microsoft Excel, изображение интерфейса которого приведено на рисунке 1.1

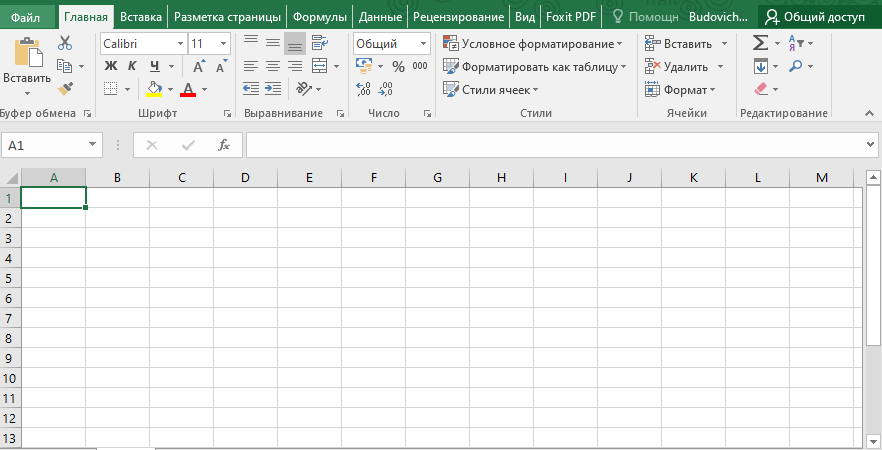


Рисунок 1.1 – интерфейс электронных таблиц Microsoft Excel

Преимущества:

* Позволяют отобразить все данные.
* Обладают фильтрами, функциями для работы с данными.
* На основе таблиц можно строить электронные диаграммы.

Недостатки:

* При больших объемах данных усложняется их восприятие пользователем.
* Не наглядны, следовательно, сложно подвести итоги работы.
  + 1. Графический вид

Данные, полученные в результате анализа, можно представить в графическом виде с помощью диаграмм. Наиболее часто используемые диаграммы: гистограммы, графики, круговые диаграммы.

Рассмотрим графики. Пример приведен на рисунке 1.2.

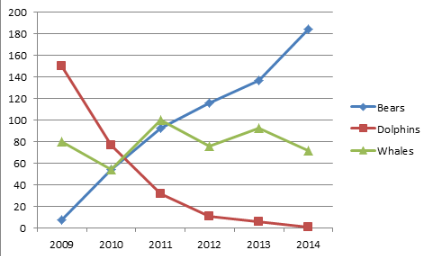


Рисунок 1.2 – пример графика

**Преимущества:**

* Наглядны.
* Компактны.
* Позволяют отобразить все данные и проследить динамику.

**Недостатки:**

* Подписи по оси абсцисс должны представлять собой числовые данные, промежутки могут быть неравномерными.
* В ряде случаев, наличие промежуточных значений нежелательно, а также может зависеть от механизма рисования графиков.

Рассмотрим гистограммы. Пример приведен на рисунке 1.3.



Рисунок 1.3 – пример круговой диаграммы

**Преимущества:**

* Наглядны, позволяющая сравнивать как значения внутри одной колонки, так и в соседних столбцах.
* Компактны.
* Позволяют отобразить все данные.
* Подписи по оси абсцисс могут быть не числовыми, а колонки располагаются на одинаковых расстояниях.

**Недостатки:**

* Сложность отображения большого количества данных при сохранении читабельности.

Рассмотрим круговые диаграммы. Пример приведен на рисунке 1.4.

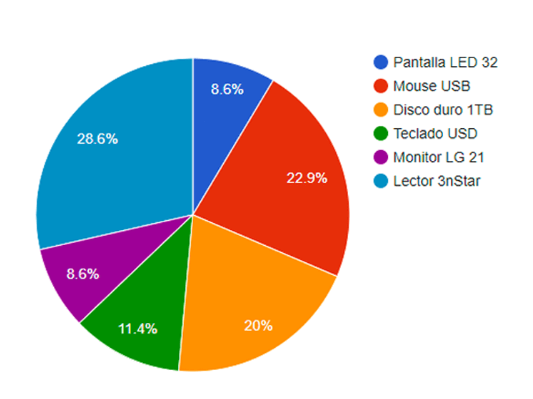


Рисунок 1.4 – пример круговой диаграммы

**Преимущества:**

* Наглядны, особенно при отображении доли как части целого.
* Удобны для подведения итога.
* Компактны.

**Недостатки:**

* Позволяют отобразить ограниченное число данных, которое можно представить в процентном отношении.
* При близких значениях возможны в сложности в определении большего и меньшего.

**Вывод**: в ходе работы для отображения результатов анализа я буду использовать столбиковые диаграммы. Для подведения результатов я воспользуюсь круговыми.

* 1. Постановка задачи

Программное средство должно демонстрировать эффективность разных алгоритмов хеширования, методов организации области переполнения при разных плотностях заполнения, на основе чего должен быть сделан вывод. Программа должна обеспечивать выполнение перечисленных ниже функций:

* + возможность отображения и взаимодействия с пользователем графиков гистограмм и круговых диаграмм;
  + возможность поиска ключа в хеш-таблице с заданными пользователем параметрами.

В папке с программой находятся:

* файл записей для размещения их в хеш-таблице;
* текстовый файл ключей для проведения поиска;
* вспомогательные текстовые данные для отображения на экране.

Пользователь может также вводить с клавиатуры ключ (строку из 6 символов).

На экране должны отобразиться столбиковые диаграммы, показывающие эффективность методов хеширования, а также круговые диаграммы, позволяющие подвести итог. При поиске ключа, введенного с клавиатуры, выводится сообщение о том, найдена запись или нет, и число просмотренных записей. При удачном поиске показывается соответствующее ключу значение.

Отображается также справочная информация о ходе работы, краткая суть каждого алгоритма и метода, чтобы пользователь имел представление. Также на экран выводится вывод, сделанный в ходе создания ПС.

Для разработки я выбрал среду Embarcadero RAD Studio 10.3 и язык программирования Delphi, так как в данной среде можно удобно разрабатывать графический пользовательский интерфейс с помощью концепции быстрой разработки приложения (RAD).

1. Анализ предметной области и разработка функциональных требований
   1. Описание функциональных требований

Программное средство при работе должно получить данные, по которым можно сделать вывод о работе алгоритмов хеширования. Для этого необходимо определить функциональные требования. Они идентифицируют задачи или действия, которые должны быть выполнены, связь входа/выхода в поведении системы.

Для моего программного средства я определил следующие функциональные требования:

* 1. Создается хеш-таблица с определенным соотношением числа пакетов и размера пакетов.
  2. Из файла записей поочередно считываются определенное число записей.
  3. Записи размещаются в хеш-таблице с помощью хеш-функции.
  4. Данные о плотности заполнения основной области и проценте записей, попавших в область переполнения, записываются в массив, отведенный под данные для анализа.
  5. Производится поиск по тестовой последовательности, которая хранит в себе существующие и несуществующие ключи;
  6. Данные о среднем числе просмотренных записей при поиске записываются в массив;
  7. Хеш-таблица удаляется. Выбирается другая хеш-функция и повторяются шаги 1-6.
  8. Выбирается другой метод организации области переполнения и повторяются шаги 1-7.
  9. Изменяется число пакетов и их размер в хеш-таблице и повторяются шаги 1-8.
  10. Изменяется количество записей для заполнения и повторяются шаги 1-9.
  11. Для каждого метода организации области переполнения подсчитывается количество случаев, когда каждая из хеш-функций была наиболее эффективной.
  12. На основании полученных данных строятся гистограммы и круговые диаграммы.
  13. Пользователь может переключаться между отображаемыми диаграммами с помощью графического интерфейса.
  14. Предусматривается возможность введения ключа пользователя с поиском по хеш-таблице с заданными параметрами (алгоритм хеширования, алгоритм организации области переполнения, размер пакета, плотность заполнения.
  15. **Описание прочих требований**

При создании работы определяются дополнительные требования:

* Максимально возможно загрузить в хеш-таблицу 960 записей. Размещение начинается с 480 записей, с увеличением на 120.
* Соотношение существующих ключей к несуществующим в тестовой последовательности при поиске находится в отношении 10 к 2.
* Ключ представляет собой строку из шести символов. Большинство ключей – английские слова.
* Интерфейс программного средства должен быть простым и удобным.
* Программное средство должно поддерживать масштабирование формы.
* Программное средство должно работать на устройстве под управлением операционной системы Windows.

1. Проектирование программного средства
   1. Разработка алгоритмов хеширования
      1. Алгоритм преобразования строкового значения в цифровое

Преобразование строкового значения ключа в цифровое является общим для всех алгоритмов хеширования, поэтому будет логичным вынести это в отдельную функцию. В моей работе ключ имеет фиксированную длину 6 символов. Значение цифрового представления должно быть, как можно более уникальным, но при этом небольшим, чтобы иметь возможность работать с ним в дальнейшем.

Поэтому в алгоритме берутся три соседних символа, их коды складываются между собой и переводятся в строку. Далее происходит конкатенация этой строки со строкой, полученной в результате таких же манипуляций со следующими тремя символами. Итоговая строка переводится в число.

Из-за подобного преобразования хеш-функции, основанные на дальнейшем складывании, такие как метод сдвига разрядов и метод складывания показали абсолютную неэффективность.

Схема алгоритма представлена на рисунке Б.1.

* + 1. Метод средних квадратов

Описание данного метода я привел в пункте 1.4.2. Здесь я остановлюсь на особенности моего алгоритма: после возведения числа в квадрат, я беру 6 центральных цифр. Поэтому они умножаются на очень малую константу. Тем не менее, это более эффективно, чем четыре цифры, как я делал изначально. Схема алгоритма представлена на рисунке 3.1.



Рисунок 3.1 – Схема алгоритма метода средних квадратов

* + 1. Метод преобразования систем счисления

Описание данного метода я привел в пункте 1.4.5. В данной работе преобразование ключа осуществляется из семнадцатиричной системы в десятичную. Именно это основание было выбрано как самое эффективное на промежутке от 11 до 18. Схема алгоритма представлена на рисунке 3.2.

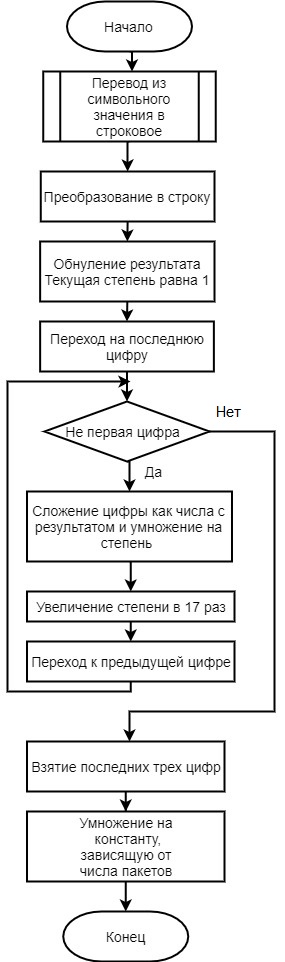


Рисунок 3.2 – Схема алгоритма метода преобразования систем счисления

* + 1. Метод деления

Описание данного метода дано в разделе 1.4.1. В данном методе я выделяю под каждый размер позицию в массиве делителей. Делитель – ближайшее к числу пакетов меньшее простое число. В качестве адреса берется остаток от деления. За счет отдельного подхода для каждого размера, домножение на константу нужно только в случае распределенной области переполнения. Схема алгоритма представлена на рисунке 3.3.



Рисунок 3.3 – Схема алгоритма метода деления

* 1. Разработка алгоритмов заполнения и поиска

Для сравнения я выбрал два метода организации области переполнения: распределенную область переполнения и открытую адресацию, так как их можно реализовать с помощью одинаковой структуры данных, описанной в пункте 3.1. Перечислю особенности реализации:

1. Если хеш-функция определит пакет, больший, чем число пакетов, то запись отправится в первый.
2. Поиск ведется до тех пор, пока:

* Не обнаружена искомая запись
* Не закончена проверка неполного пакета
* Пока не проверены основной пакет и вся область переполнения

1. Число существующих записей относится к несуществующим как 10 к 2.
2. В алгоритме размещения находятся переменные, вычисляющие плотность заполнения основной области и процент записей, попавших в область переполнения, для анализа.
3. В алгоритме поиска находятся переменные, вычисляющие количество проверенных записей при поиске (в среднем), при удачном поиске, при неудачном.
   * 1. Распределенная область переполнения

Особенностью распределенной области переполнения является факт того, что необходимо учитывать наличие пакетов между первичными пакетами. Схема алгоритма заполнения представлена на рисунке Б.2. Схема алгоритма поиска представлена на рисунке Б.3.

* + 1. Открытая адресация

Метод открытой адресации устроен немного проще, так как записи просто направляются в следующий пакет. Схема алгоритма заполнения представлена на рисунке Б.4. Схема алгоритма поиска представлена на рисунке Б.5.

* 1. Разработка алгоритма сбора данных

После реализации алгоритмов хеширования необходимо создать процедуру, в которой будет происходить вызов функция заполнения и поиска с сохранением параметров для анализа. Для хранения я использую статический массив записей.

По внешнему циклу будет изменятся плотность заполнения, начиная с 480 записей на 1200 мест (40%), с шагом 120 (10%).По внутреннему циклу будет изменятся соотношение размера хеш-таблицы к размеру пакета: от 20/60 до 200/6 с увеличением числа пакетов на 20. Схема данного алгоритма представлена на рисунке Б.6.

* 1. Разработка алгоритма подсчета для круговых диаграмм

Данный алгоритм опирается на данные, полученные из алгоритма, описанном в пункте 3.4. В нем сравнивается наиболее эффективный алгоритм хеширования для каждого сочетания (процент заполнения – размер пакета – метод организации области переполнения). Сложение результатов идет в пределах одного метода организации области переполнения. Схема данного алгоритма представлена на рисунке Б.7.

1. Конструирование программного средства
   1. Разработка динамических структур данных

Так как в работе я моделирую работу хеш-таблицы, а именно ее функции добавления и поиска, при этом изменяя размер пакета, в качестве динамической структуры данных я выбрал динамический массив пакетов.

Пакет представляет собой запись, в который будут помещаться записи из файла. Поэтому пакет состоит из переменной, указывающей на фактический размер пакета, и динамического массива записей.

Type

TBucket = record

Bucket: array of InputRecord;

Volume: SmallInt;

end;

THashTable = array of TBucket;

Размер массива типа THashTable изменяется от 20 до 200 с шагом 20. Размер поля Bucket определяется исходя из размера хеш-таблицы (числа пакетов). Принимая во внимание, что хеш-таблица потенциально способна вместить 1200 записей, то размер пакета вычисляется как отношение потенциального размера к числу пакетов с округлением к ближайшему целому.

* 1. Работа с файлами

Для работы ПС используются следующие файлы:

* Файл записей для заполнения. Содержит 1000 записей, состоящих из ключа и значения, пронумерованного от 1 до 1000. В работе используются 960 записей. Остальные – резервные. Каждый раз при заполнении с файла считывается последовательно определенное количество записей.
* Текстовый файл ключей. Содержит одну длинную строку, состоящую из 1200 подстрок, которые используются при поиске в качестве ключей. 48 подстрок резервные.
* Файлы компонентов форм. Содержат текст в кодировке UTF-8 для отображения на экране. Представляют собой справочную информацию о работе, пояснение графиков, сокращенный вывод к работе.
  1. Взаимосвязь между компонентами формы

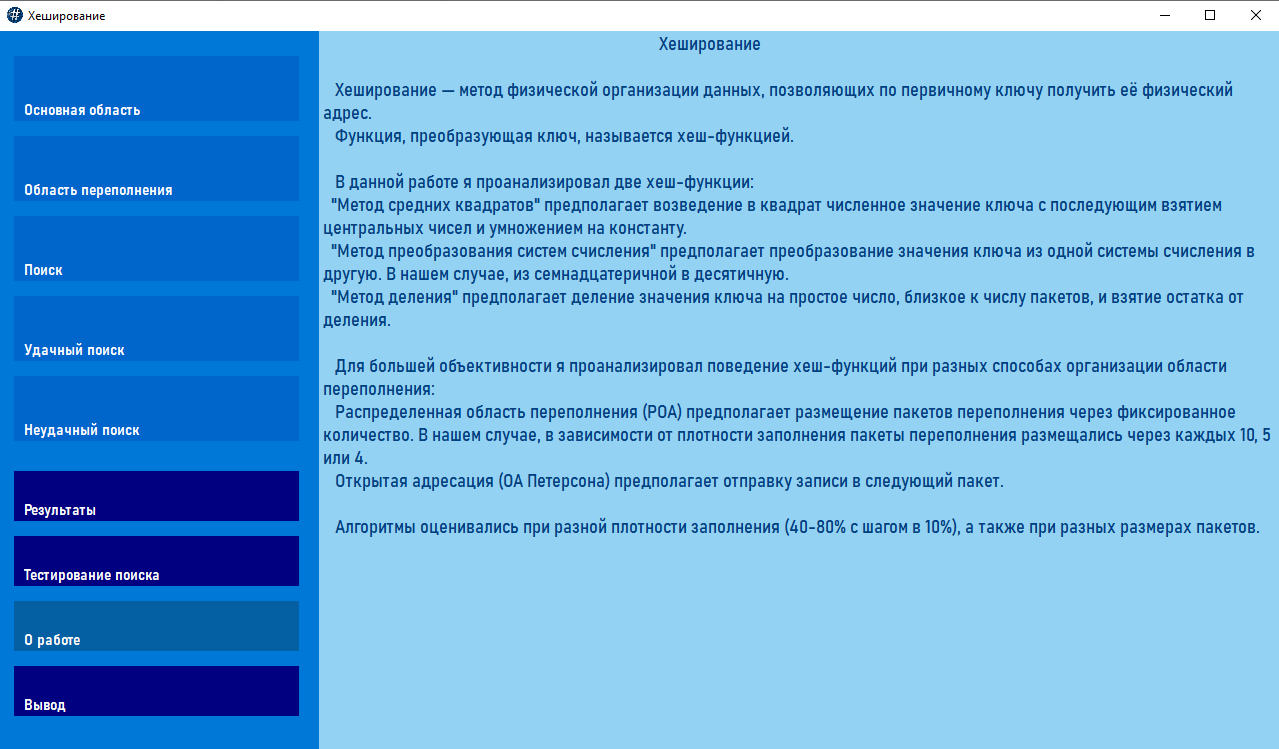
Для работы я создал одну форму MainForm. Форма обеспечивает отображение графиков, справочной информации, вывода, а также переключение между активными компонентами с помощью меню. Её вид приведен на рисунке 4.1.

Рисунок 4.1 –Форма при запуске приложения

Взаимосвязь между компонентами устроена следующим образом:

* 1. Основная форма состоит из четырех основных элементов – панель бокового меню (MainMenuPanel), панель с графиками столбиковых диаграмм (ChartPanel), панель с круговыми диаграммами (ResultPanel), панели поиска SearchTestPanel.
  2. Компонент MainMenuPanel имеет дочерние компоненты DensityPanel, OverflowPanel, SearchPanel, GoodSearchPanel, BadSearchPanel, которые делают видимым компонент ChartPanel и меняют график; а также GrafPanel, которая делает видимым ResultPanel; TestSearchPanel, которая делает видимой панель поиска; InfoPanel, SummaryPanel, показывающие Memo2.
  3. Компонент ChartPanel имеет нижнее меню в виде компоненты DensMenuPanel, изменяющий графики, а также компонент Memo1, отображающий пояснение к графику, и сам график Chart1.
  4. Компонент ResultPanel имеет нижнее меню GraphMenuPanel и график круговых диаграмм Chart2.
  5. Компонент SearchTestPanel имеет несколько компонент класса TRadioGroup для выбора параметров, однострочный редактор текста класса TEdit для ввода ключа и панель для поиска.
  6. Структура модулей программы

Программа состоит из трех модулей:

1. MainFormUnit – модуль основной формы;
2. HashAlgorithm – модуль с алгоритмами хеширования, размещения, поиска. В модуле хранятся процедуры, реализованные в разделе interface, собирающие данные для отображения на графике;
3. Types – модуль, хранящий типы и глобальные константы.
   1. Описание модуля MainFormUnit

Модуль MainFormUnit является модулем формы MainForm и отвечает за отображение элементов формы, их взаимодействие с пользователем (изменение цвета панелей при нажатии, границы при наведении), работу с графиками, организацию выбора параметров для поиска, отображение текстовой информации.

Таблица 4.1 Подпрограммы модуля MainFormUnit

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограм-мы** | **Описание** | **Заголовок подпрограм-мы** | **Имя параметра** | **Назначение параметра** |
| TMainForm. FormCreate | Создание формы с изначальными настройками | procedure TMainForm. FormCreate (Sender: TObject) | Sender | Объект, который сгенерировал событие |

Продолжение таблицы 4.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограм-мы** | **Описание** | **Заголовок подпрограм-мы** | **Имя параметра** | **Назначение параметра** |
| TMainForm. SetDefault | Восстановление цветов бокового меню и очистка серий столбиков | procedure TMainForm. SetDefault (Sender: TObject) | Sender | Объект, который сгенерировал событие |
| TMainForm. SetDefault DensityPanel | Восстановление цветов нижнего меню | procedure TMainForm. SetDefault DensityPanel (Sender: TObject) | Sender | Объект, который сгенерировал событие |
| TMainForm. DensityPanel Click | Нажатие на панель с отображением плотности основной области на графике | procedure TMainForm.DensityPanelClick(Sender: TObject) | Sender | Объект, который сгенерировал событие |
| TMainForm. OverflowPanelClick | Нажатие на панель с отображением процента записей, попавших в область переполнения на графике | procedure TMainForm.Overflow PanelClick (Sender: TObject) | Sender | Объект, который сгенерировал событие |
| TMainForm. SearchPanel Click | Нажатие на панель с отображением результатов поиска | procedure TMainForm. SearchPanel Click(Sender: TObject) | Sender | Объект, который сгенерировал событие |
| TMainForm. GoodSearch PanelClick | Нажатие на панель с отображением результатов удачного поиска | procedure TMainForm.GoodSearch PanelClick (Sender: TObject) | Sender | Объект, который сгенерировал событие |

Продолжение таблицы 4.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограм-мы** | **Описание** | **Заголовок подпрограм-мы** | **Имя параметра** | **Назначение параметра** |
| TMainForm. GrafPanelClick | Нажатие на панель с отображением круговой диаграммы | procedure TMainForm. GrafPanel Click(Sender: TObject) | Sender | Объект, который сгенерировал событие |
| TMainForm. BadSearch Click | Нажатие на панель с отображением результатов неудачного поиска | procedure TMainForm.BadSearch Click(Sender: TObject) | Sender | Объект, который сгенерировал событие |
| TMainForm. InfoPanelClick | Нажатие на панель с информации о работе | procedure TMainForm. InfoPanelClick(Sender: TObject) | Sender | Объект, который сгенерировал событие |
| TMainForm. SearchTestPanelClick | Нажатие на панель для тестирования поиска | procedure TMainForm.SearchTestPanelClick(Sender: TObject); | Sender | Объект, который сгенерировал событие |
| TMainForm. SummaryPanelClick | Нажатие на панель для отображения вывода | procedure TMainForm. SummaryPanelClick(Sender: TObject) | Sender | Объект, который сгенерировал событие |
| TMainForm. DensPanel1 Click | Нажатие на панель с 40% заполнением и обновление графика | procedure TMainForm. DensPanel1 Click(Sender: TObject) | Sender | Объект, который сгенерировал событие |
| TMainForm. DensPanel2 Click | Нажатие на панель с 50% заполнением и обновление графика | procedure TMainForm. DensPanel2 Click(Sender: TObject) | Sender | Объект, который сгенерировал событие |

Продолжение таблицы 4.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограм-мы** | **Описание** | **Заголовок подпрограм-мы** | **Имя параметра** | **Назначение параметра** |
| TMainForm. DensPanel3 Click | Нажатие на панель с 60% заполнением и обновление графика | procedure TMainForm. DensPanel3 Click(Sender: TObject) | Sender | Объект, который сгенерировал событие |
| TMainForm. DensPanel4 Click | Нажатие на панель с 70% заполнением и обновление графика | procedure TMainForm. DensPanel4 Click(Sender: TObject) | Sender | Объект, который сгенерировал событие |
| TMainForm. DensPanel5 Click | Нажатие на панель с 80% заполнением и обновление графика | procedure TMainForm. DensPanel5 Click(Sender: TObject) | Sender | Объект, который сгенерировал событие |
| TMainForm. MouseEnter | Наведение мыши на панель | procedure TMainForm. MouseEnter (Sender: TObject) | Sender | Объект, который сгенерировал событие |
| TMainForm. MouseLeave | Отмена наведения мыши на панель | procedure TMainForm. MouseLeave (Sender: TObject); | Sender | Объект, который сгенерировал событие |
| TMainForm. OAPanelClick | Отображение круговой диаграммы для открытой адресации | procedure TMainForm. OAPanel Click(Sender: TObject) | Sender | Объект, который сгенерировал событие |
| TMainForm. ROPPanel Click | Отображение круговой диаграммы для распределенной области переполнения | procedure TMainForm. ROPPanel Click(Sender: TObject) | Sender | Объект, который сгенерировал событие |

Продолжение таблицы 4.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограм-мы** | **Описание** | **Заголовок подпрограм-мы** | **Имя параметра** | **Назначение параметра** |
| TMainForm. ToResize | Изменение формы и компонент при растяжении | procedure TMainForm. ToResize (Sender: TObject) | Sender | Объект, который сгенерировал событие |
| TMainForm. LabelMouse Enter | Имитация нажатия на панель | procedure TMainForm.LabelMouse Enter(Sender: TObject) | Sender | Объект, который сгенерировал событие |
| TMainForm. Chart1Click | Нажатие на график для отображения следующих/ предыдущих 5 столбиков | procedure TMainForm.Chart1Click (Sender: TObject) | Sender | Объект, который сгенерировал событие |
| TMainForm. GetSearch ResultPanelClick | Нажатие на панель «Поиск» на панели тестирования поиска для вывода результатов | procedure TMainForm.GetSearchResultPanelClick(Sender: TObject) | Sender | Объект, который сгенерировал событие |
| TMainForm. Chart1MouseWheelDown | Прокрутка мыши колесиком для отображения пяти столбиков на графике | procedure TMainForm.Chart1MouseWheelDown (Sender: TObject; Shift: TShiftState;MousePos: TPoint; var Handled: Boolean) | Sender | Объект, который сгенерировал событие |
| Shift | Состояние мыши |
| MousePos | Позиция мыши |
| Handled | Флаг передачи управления |

Продолжение таблицы 4.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограм-мы** | **Описание** | **Заголовок подпрограм-мы** | **Имя параметра** | **Назначение параметра** |
| TMainForm. Chart1MouseWheelUp | Прокрутка мыши колесиком для отображения десяти столбиков на графике | procedure TMainForm.Chart1MouseWheelUp (Sender: TObject; Shift: TShiftState; MousePos: TPoint; var Handled: Boolean) | Sender | Объект, который сгенерировал событие |
| Shift | Состояние мыши |
| MousePos | Позиция мыши |
| Handled | Флаг передачи управления |
| TMainForm. SeriesMouse Enter | Наведение мыши на столбики для отображения подписей графика | procedure TMainForm. SeriesMouseEnter(Sender: TObject) | Sender | Объект, который сгенерировал событие |
| TMainForm. SeriesMouse Left | Отмена наведения на столбики | procedure TMainForm.SeriesMouseLeft(Sender: TObject) | Sender | Объект, который сгенерировал событие |
| TMainForm.FormDestroy | Очистка хеш-таблицы при удалении формы | procedure TMainForm.FormDestroy(Sender: TObject); | Sender | Объект, который сгенерировал событие |

* 1. Описание модуля HashAlgorithm

В данном модуле в разделе interface объявлены подпрограммы, обеспечивающие связь между модулями. Они передают информацию о результатах работы алгоритмов в статический массивы или записи. Здесь объявлены подпрограммы, которые необходимы для вызова поиска из формы.

В модуле есть эмпирически подобранные константы, дающие максимальную эффективность при умножении на результат хеш-функции.

Таблица 4.2 Подпрограммы модуля HashAlgorithm в разделе interface

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограммы** | **Описание** | **Заголовок подпрограммы** | **Имя параметра** | **Назначение параметра** |
| GetHashProcessResults | Запуск заполнения и поиска для разных плотностей и методов с сохранением результатов | procedure GetHashProcessResults (out chartArray: TChartArray) | chartArray | Массив записей, содержащий всю информацию для столбиковых графиков |
| GetMost Sufficient | Определение количества лучших позиций для каждого из алгоритмов хеширования при поиске | procedure GetMost Sufficient (const chartArray: TChartArray; out PetersonPie Record, Distributed PieRecord: TPieRecord) | chartArray | Массив записей с данными |
| PetersonPie Record | Запись с показателями числа лучших позиций по каждой функции для открытой адресации |
| Distributed PieRecord | Запись с показателями числа лучших позиций по каждой функции для распределенной области переполнения |
| HashAverage Square | Хеширование методом средних квадратов | function HashAverage Square(const key: string; const range, HashTableSize: integer): int64; | key | string |
| range | Константа для умножения |
| HashTable Size | Размер хеш-таблицы |

Продолжение таблицы 4.2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограммы** | **Описание** | **Заголовок подпрограммы** | **Имя параметра** | **Назначение параметра** |
| HashSystem Conversion | Хеширование методом преобразования систем счисления | function HashSystem Conversion(const key: string; const range, HashTableSize: integer): int64; | key | string |
| range | Константа для умножения |
| HashTable Size | Размер хеш-таблицы |
| HashDivision | Хеширование методом деления | function HashDivision (const key: string; const range, HashTableSize: integer): int64 | key | string |
| range | Константа для умножения |
| HashTable Size | Размер хеш-таблицы |
| Hash Distributed | Размещение записей в хеш-таблице с использованием распределенной области переполнения | procedure HashDistributed (const HashFunction: THashFunction; const range, HashTableSize, FileVolume, overbucket: integer; out HashTable: THashTable; out percentageOverflow, mainDensity: real) | HashFunction | Хеш-функция |
| range | Константа для умножения |
| HashTable Size | Размер хеш-таблицы |
| fileVolume | Количество записей, с которым необходимо работать |
| HashTable | Хеш-таблица |
| Percentage Overflow | Процент записей в области переполнения |
| mainDensity | Плотность основной области |
| overbucket | Расстояние между пакетами переполнения |

Продолжение таблицы 4.2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограммы** | **Описание** | **Заголовок подпрограммы** | **Имя параметра** | **Назначение параметра** |
| HashPeterson | Размещение записей в хеш-таблице с использованием открытой адресации | procedure HashPeterson (const HashFunction: THashFunction; const range, HashTableSize, FileVolume: integer; out HashTable: THashTable; out percentage Overflow, mainDensity: real) | HashFunction | Хеш-функция |
| range | Константа для умножения |
| HashTable Size | Размер хеш-таблицы |
| fileVolume | Количество записей, с которым необходимо работать |
| HashTable | Хеш-таблица |
| Percentage Overflow | Процент записей в области переполнения |
| mainDensity | Плотность основной области |
| SearchKey Distributed | Поиск ключа по хеш-таблице, использующей распределенную область переполнения | function SearchKeyDistributed(key: string; HashTable: THashTable; const HashFunction: THashFunction; const range, FileVolume, overbucket: integer; out checked: integer; out value: integer): boolean | HashTable | Хеш-таблица |
| HashFunction | Хеш-функция |
| range | Константа для умножения |

Продолжение таблицы 4.2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограммы** | **Описание** | **Заголовок подпрограммы** | **Имя параметра** | **Назначение параметра** |
|  |  |  | fileVolume | Количество записей, с которым необходимо работать (без учета контрольных 20%) |
| overbucket | Расстояние между пакетами переполнения |
| key | Ключ для поиска |
| value | Соответствующее ключу значение |
| SearchKey Peterson | Поиск ключа по хеш-таблице, использующей открытую адресацию | function SearchKeyPeterson(key: string; HashTable: THashTable; const HashFunction: THashFunction; const range, FileVolume: integer; out checked: integer; out value: integer): boolean; | HashTable | Хеш-таблица |
| HashFunction | Хеш-функция |
| range | Константа для умножения |
| fileVolume | Количество записей, с которым необходимо работать (без учета контрольных 20%) |
| key | Ключ для поиска |
| value | Соответствующее ключу значение |

В разделе implementation описываются также алгоритмы перевода строкового ключа в число, а также алгоритмы поиска (для анализа).

Таблица 4.3 Подпрограммы модуля HashAlgorithm в разделе implementation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограммы** | **Описание** | **Заголовок подпрограммы** | **Имя параметра** | **Назначение параметра** |
| StringTo Decimal | Перевод значения ключа из строкового в цифровое представление | function StringTo Decimal(const key: string): int64; | key | Ключ из шести символов |
| Search Distributed | Поиск по хеш-таблице, использующей распределенную область переполнения | procedure SearchDistributed(HashTable: THashTable; const HashFunction: THashFunction; const range, FileVolume, overbucket: integer; out checkedAverage, checkedAverageNotFound, checkedAverage Found: real) | HashTable | Хеш-таблица |
| HashFunction | Хеш-функция |
| range | Константа для умножения |
| fileVolume | Количество записей, с которым необходимо работать (без учета контрольных 20%) |
| overbucket | Расстояние между пакетами переполнения |
| Checked Average | Среднее число просмотренных записей |
| Checked Average Found | Среднее число просмотренных записей при удачном поиске |
| Checked Average notFound | Среднее число просмотренных записей при неудачном поиске |

Продолжение таблицы 4.3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограммы** | **Описание** | **Заголовок подпрограммы** | **Имя параметра** | **Назначение параметра** |
| Search Peterson | Поиск по хеш-таблице, использующей открытую адресацию | procedure SearchPeterson (HashTable: THashTable; const HashFunction: THashFunction; const range, FileVolume: integer, out checkedAverage, checkedAverageNotFound, checkedAverage Found: real) | HashTable | Хеш-таблица |
| HashFunction | Хеш-функция |
| range | Константа для умножения |
| fileVolume | Количество записей, с которым необходимо работать (без учета контрольных 20%) |
| Checked Average | Среднее число просмотренных записей |
| Checked Average Found | Среднее число просмотренных записей при удачном поиске |
| Checked Average notFound | Среднее число просмотренных записей при неудачном поиске |

* 1. Описание модуля Types

Модуль Types хранит объявления структур данных (статических и динамических), а также константы. Все объявлено в разделе interface. Ниже приведены основные структуры данных.

Таблица 4.4 Основные структуры данных

|  |  |  |
| --- | --- | --- |
| **Имя идентификатора структуры** | **Назначение структуры** | **Тип структуры** |
| InputRecord | Запись для размещения в хеш-таблице | Запись  key: string[6]  value: integer |
| ConstArray | Массив констант для хещ-функций | Статический массив  Array [1..5] of integer |
| data | Запись с результатами анализа (все характеристики для одного метода, одной плотности и размера пакета: процент записей, в области переполнения, процент заполнения основной области, среднее число просмотренных записей при поиске, удачном и неудачном) | Запись  percentageOverflow: real;  mainDensity: real;  checkedAverage: real;  checkedAverageNotFound:real;  checkedAverageFound: Real; |
| OrganizationRecord | Промежуточная запись результатов анализа (три метода для одной плотности и размера пакета) | Запись  averageSquare: data;  system: data;  division: data; |
| UnterChartRecord | Промежуточная запись результатов анализа (оба алгоритма организации области переполнения) | Запись  HashTableSize: integer;  Distributed: OrganizationRecord;  Peterson : OrganizationRecord; |
| TUnterChartArray | Промежуточный массив (для всех размеров и одной плотности) | Статический массив  TUnterChartArray = Array [1..N] of UnterChartRecord, где N=10 |

Продолжение таблицы 4.4

|  |  |  |
| --- | --- | --- |
| **Имя идентификатора структуры** | **Назначение структуры** | **Тип структуры** |
| UberChartRecord | Промежуточная запись (для всех размеров и одной плотности) | Запись  fileVolume: integer;  UnterChart: TUnterChartArray |
| TChartArray | Массив данных для графика | Статический массив  array [1..5] of UberChartRecord |
| TBucket | Пакет записей | Запись, состоящая из динамического массива и целого числа  Bucket: array of InputRecord;  Volume: SmallInt; |
| THashTable | Хеш-таблица | Динамический массив пакетов:  array of TBucket |
| TPieRecord | Запись для круговой диаграммы, содержащая счетчики эффективности для каждой хеш-функции | Запись:  AverageSquare: integer;  SystemConversion: integer;  Division: integer; |
| SearchParam | Запись для параметров для поиска | Запись: OverflowOrganizationMethod: byte;  HashFunction: THashFunction;  HashTableSize: Integer;  FileVolume: Integer; |

В результате этапа конструирования создано программное средство. Схема программы представлена в приложении А. Код программы представлен в приложении В.

1. Тестирование, проверка работоспособности и анализ полученных результатов

В данном разделе я остановлюсь на тестировании программного средства, а также проанализирую результаты работы алгоритмов с помощью графического представления.

Здесь также я произведу анализ, в котором я дам оценку алгоритмам.

* 1. Тестирование алгоритмов хеширования

Выполняемые программным средством алгоритмы хеширования скрыты, поэтому проверить правильность их выполнения можно с помощью отладчика среды Embarcadero Delphi.

Таблица 5.1 – тестирование алгоритмов хеширования

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Тестируемая функциональность** | **Исходные данные** | **Ожидаемый результат** | **Полученный результат** |
| 1 | Алгоритм преобразования строкового значения ключа в числовое | Ключ ‘purple’ | 112+117+114=343  112+108+101=321  ‘343’+’321’=  ‘343321’ | Тест пройден |
| 2 | Алгоритм преобразования строкового значения ключа в числовое | Ключ ‘Alaska’ | 65+108+97=270  115+107+97=319  ‘270’+’319’= ‘270319’ | Тест пройден |
| 3 | Метод средних квадратов (относительный) | Ключ ‘purple’ | 343321\*343321=  117 8**69 309**041  **69309‬** | Тест пройден |
| 4 | Метод средних квадратов (относительный) | Ключ ‘Alaska’ | 270319\*270319=  73 0**72 361**761‬  **72361** | Тест пройден |
| 5 | Метод преобразования систем счисления (относительный) | Ключ ‘purple’ | 343321 (17) =4609**296**(10)  **296** | Тест пройден |
| 6 | Метод преобразования систем счисления (относительный) | Ключ ‘Alaska’ | 270319 (17) =3425**254** (10)  **254** | Тест пройден |
| 7 | Метод деления (для 200 пакетов) | Ключ ‘purple’ | 343321 mod 199 = **46** | Тест пройден |
| 8 | Метод деления  (для 40 пакетов) | Ключ ‘Alaska’ | 270319 mod 37=  **34** | Тест пройден |

Продолжение таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Тестируемая функциональность** | **Исходные данные** | **Ожидаемый результат** | **Полученный результат** |
| 9 | Размещение при распределенной области переполнения (200 пакетов, 80%, метод средних квадратов) | ‘washed’, 959 | Размещение | 120 пакет  Тест пройден |
| 10 | Размещение при открытой адресации (200 пакетов, 80%, метод средних квадратов) | ‘washed’, 959 | Размещение | 125 пакет  Тест пройден |
| 11 | Размещение при распределенной области переполнения (20 пакетов, 80%, метод преобразования систем счисления) | ‘washed’, 959 | Размещение | 184-187-(+4)…199  3-4-(+4)…31  31 пакет  Тест пройден |
| 12 | Размещение при открытой адресации (200 пакетов, 80%, метод преобразования систем счисления) | ‘washed’, 959 | Размещение | 184…199  0..16  16 пакет  Тест пройден |
| 13 | Размещение при распределенной области переполнения (200 пакетов, 80%, метод деления) | ‘washed’, 959 | Размещение | 169 пакет Тест пройден |
| 14 | Размещение при открытой адресации (200 пакетов, 80%, метод деления) | ‘washed’, 959 | Размещение | 169 пакет Тест пройден |
| 15 | Удачный поиск при распределенной области переполнения | ‘weasel’ (670) | Найдено: weasel 670. Просмотрено N записей. | Проверено 6 записей  Тест пройден |
| 16 | Удачный поиск при открытой адресации (200 пакетов, 80%, метод средних квадратов) | ‘weasel’ (670) | Найдено: weasel 670. Просмотрено N записей. | Проверено 6 записей  Тест пройден |

Продолжение таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Тестируемая функциональность** | **Исходные данные** | **Ожидаемый результат** | **Полученный результат** |
| 17 | Неудачный поиск при распределенной области переполнения (200 пакетов, 80%, метод средних квадратов) | ‘errata’ | Не найдено. Просмотрено N записей. | Проверено 53 записей  Тест пройден |
| 18 | Неудачный поиск при открытой адресации (200 пакетов, 80%, метод средних квадратов) | ‘errata’ | Не найдено. Просмотрено N записей. | Проверено 70 записей  Тест пройден |
| 19 | Поиск по неверному ключу | ‘abc’ | Проверьте ввод | Тест пройден |

* 1. Тестирование компонентов формы

Так как основная задача программного средства – отображать результаты анализа алгоритма, то взаимодействие с пользователем ограниченно. Тестирование ограничится проверкой реагирования компонент на нажатие, использование колесика мыши, изменение размеров формы.

Таблица 5.2 - тестирование функционала компонент

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Тестируемая функциональность** | **Последовательность действий** | **Ожидаемый результат** | **Полученный результат** |
| 1 | Реакция панели бокового меню на взаимодействие с пользователем. | 1. 1. Наведение мыши на верхнюю панель 2. 2. Нажатие на нее 3. 3.Нажатие на другую панель | 1.Изменение контура панели, изменение курсора мыши  2. Изменение цвета панели. При отведении мыши курсор восстанавливает стандартное состояние, контур исчезает. Появляется/ Изменяется график  3. Восстановление изначального цвета у исходной панели. Изменение цвета у другой | Тест пройден |

Продолжение таблицы 5.2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Тестируемая функциональность** | **Последовательность действий** | **Ожидаемый результат** | **Полученный результат** |
| 2 | Реакция панели нижнее меню на взаимодействие с пользователем. | 1. 1. Наведение мыши на крайнюю правую панель 2. 2. Нажатие на нее 3. 3.Нажатие на другую панель | 1.Изменение контура панели, изменение курсора мыши  2. Изменение цвета панели. При отведении мыши курсор восстанавливает стандартное состояние, контур исчезает. Изменяется график  3. Восстановление изначального цвета у исходной панели. Изменение цвета у другой | Тест пройден |
| 3. | Взаимодействие с графиком столбиковых диаграмм | 1. 1. Наведение мыши на столбик. 2. 2. Щелчок ЛКМ по графику 3. 3. Прокрутка колесиком вверх. 4. 4. Повторный щелчок. 5. 5. Прокрутка колесиком вниз. | 1.Подсветка серии красным. Отображение значений серии.  2.Показ следующих пяти колонок.  3. Показ десяти колонок одновременно  4. Ничего не происходит  5. Показ первых пяти колонок | Тест пройден |

Продолжение таблицы 5.2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Тестируемая функциональность** | **Последовательность действий** | **Ожидаемый результат** | **Полученный результат** |
| 4. | Изменение размеров формы | 1. 1. Растяжение формы на полный экран 2. 2. Сжатие формы до минимума 3. 3. Вытягивание формы по горизонтали 4. 4. Вытягивание формы по вертикали | 1.Увеличение компонент и шрифтов с сохранением большей части пропорций. Максимальный размер 1920x1280.  2. Сжатие компонент и шрифтов с сохранением большинства пропорций. Минимальный размер 1024x576  3. Увеличение компонент с сохранением большей части пропорций по горизонтали. Частичное увеличение шрифтов. Сохранение значений по высоте.  4. Увеличение компонент с сохранением большей части пропорций по вертикали. Частичное увеличение шрифтов. Сохранение значений по ширине. | Тест пройден |

Подводя итог, отмечу, что программа отвечает заданным функциональным требованиям, наблюдается стабильность в работе. Вопросов к эстетической части не имеется.

* 1. Анализ алгоритмов хеширования

В данном разделе я перечислю особенности алгоритмов хеширования, с которыми я столкнулся при реализации, отладке, а также при анализе выведенных мною графиков. Здесь я раскрываю основные наблюдения, которые я провел в ходе работы и считаю этот раздел ключевым.

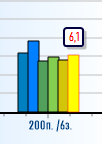
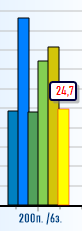
1. Несмотря на то, что считается, что время выполнения поиска и размещения данных хеширования является постоянным, в действительности я наблюдаю ситуацию, когда оно зависит от плотности заполнения хеш-таблицы. Для хеш-таблиц с большим размером пакета рост количества операций, в целом, равномерный. Так, для пакета размером 60 при увеличении процента заполнения таблицы от 40% до 80% с шагом 10% наблюдается рост на 18-20% относительно предыдущего значения. В то время как пакеты малого размера не отличаются подобной стабильностью. Например, если для размера пакета 6 при увеличении плотности с 40% до 50% рост числа операций составляет для метода средних квадратов и открытой адресации составляет 23%, то при переходе от 70% к 80% – 122%. Так как участок 40-60% является относительно стабильным, рекомендую производить реорганизацию данных при достижении **60%**. При заполнении хеш-таблицы более чем на 80% (83,3% - 1000 записей на 1200 мест) алгоритмы размещения для распределенной области переполнения начинают вести себя опасно – возможно зацикливание, даже если пакет переполнения – каждый второй, что еще и неэффективно. В этом случае имеет смысл пожертвовать памятью ради производительности и безопасности.
2. Несмотря на то, что я ранее упомянул преимущество больших пакетов – стабильное увеличение времени поиска, чем меньше размер пакета, тем быстрее будет обнаружена нужная запись. Чтобы пакеты не заполнялись полностью и время поиска оставалось относительно стабильным, можно остановиться на **размере 10-12 записей**. Стоит отметить, что при **хорошем алгоритме** хеширования размер пакета **6 записей** также рекомендуется к использованию.
3. Распределенная область переполнения проигрывает открытой адресации по среднему времени поиска. К тому же стоит учитывать, что при достижении определенного порога необходимо производить реорганизацию данных либо с увеличением размера таблицы (рекомендуется), либо с увеличением числа и частоты пакетов, отданных под переполнение. Преимущество данного метода – в относительной стабильности при увеличении плотности заполнения.
4. **Метод открытой адресации** **прост и быстр**, особенно, при низкой плотности заполнения. По итогам работы я признаю его более эффективным.
5. **Метод деления** я считаю **наиболее эффективным**, особенно в сочетании с открытой адресацией (Рисунок 5.3). Кроме того, данный алгоритм прост в реализации, подходит для любых чисел, а поэтому на практике он используется чаще всего. Однако самый лучший алгоритм дает самые худшие значения при распределенной области переполнения (Рисунок 5.2). Это единственный алгоритм, в котором я могу рекомендовать размер пакета 6 и менее.
6. Метод средних квадратов занимает второе место. Он достаточно эффективный и при открытой адресации иногда незначительно уступает методу деления. Относительно других он эффективен при низкой плотности заполнения и большом размере пакета.
7. Метод преобразования систем счисления не отличается хорошими показателями. Большое количество записей попадает в область переполнения, что свидетельствует о неравномерном заполнении.
8. Неудачный поиск идет для низкой плотности заполнения в среднем в два раза дольше, что логично. При высокой плотности значения могут отличаться в четыре и более раз, что особенно заметно на небольшом размере пакета (Рисунок 5.1).

Рисунок 5.1 – Сравнение удачного и неудачного поиска при размере пакета 6 и плотности заполнения 80%

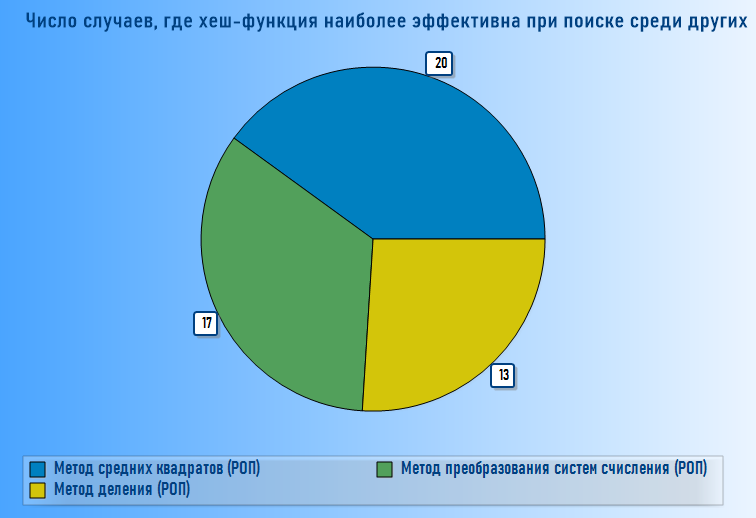
Рисунок 5.2 – Наиболее эффективная хеш-функция при поиске при распределённой области переполнения



Рисунок 5.3 – Наиболее эффективная хеш-функция при открытой адресации

1. Руководство по установке и использованию

Данное программное средство разработано для использования на операционной системе Windows (рис. 6.1).

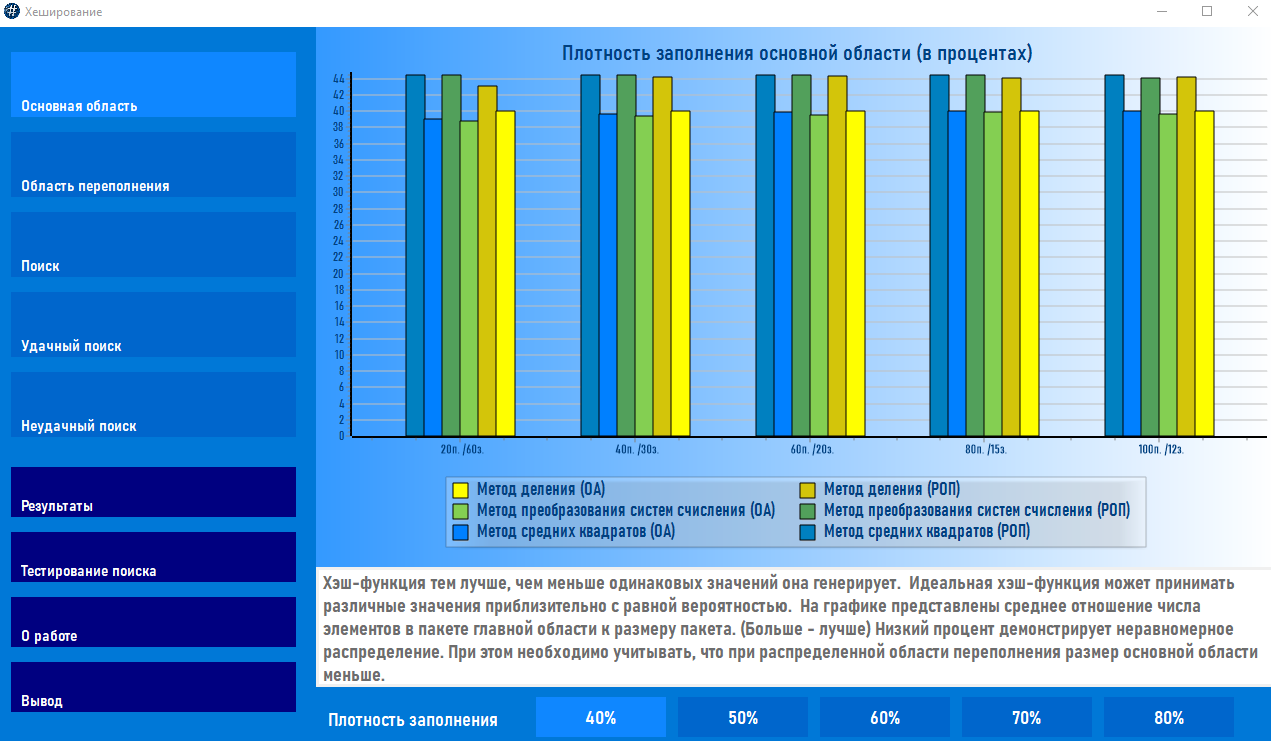


Рисунок 6.1 – Программное средство на устройстве под управлением Windows 10

Для нормальной работы программы папка программного средства должна содержать следующие файлы:

1. Исполняемый файл Hashing.exe
2. Файл хранения записей Hashdata.txt
3. Файл с данными для поиска Search.txt
4. Файлы загрузки справочной текстовой информации DensInfo.txt, OverflowInfo.txt, SearchInfo.txt, Info.txt, Resume.txt.

Взаимодействие с пользователем осуществляется с помощью бокового меню, состоящего из девяти панелей.

При нажатии на любую из пяти верхних панелей меню отображается график столбиковых диаграмм. Заголовок графика и легенда поясняют отображаемое. Изначально демонстрируется пять колонок, при нажатии пользователем левой кнопки мыши на график отображаются следующие пять. Повторный клик возвращает исходное состояние. Прокрутка колесика мыши вверх-вниз осуществляет изменение числа колонок на графике (5 или 10).

В данном состоянии имеется нижнее меню (рис 6.2), состоящее из пяти панелей, позволяющее изменять плотность заполнения для графика. Изначально она равна 40%.

Рисунок 6.2 – Нижнее меню

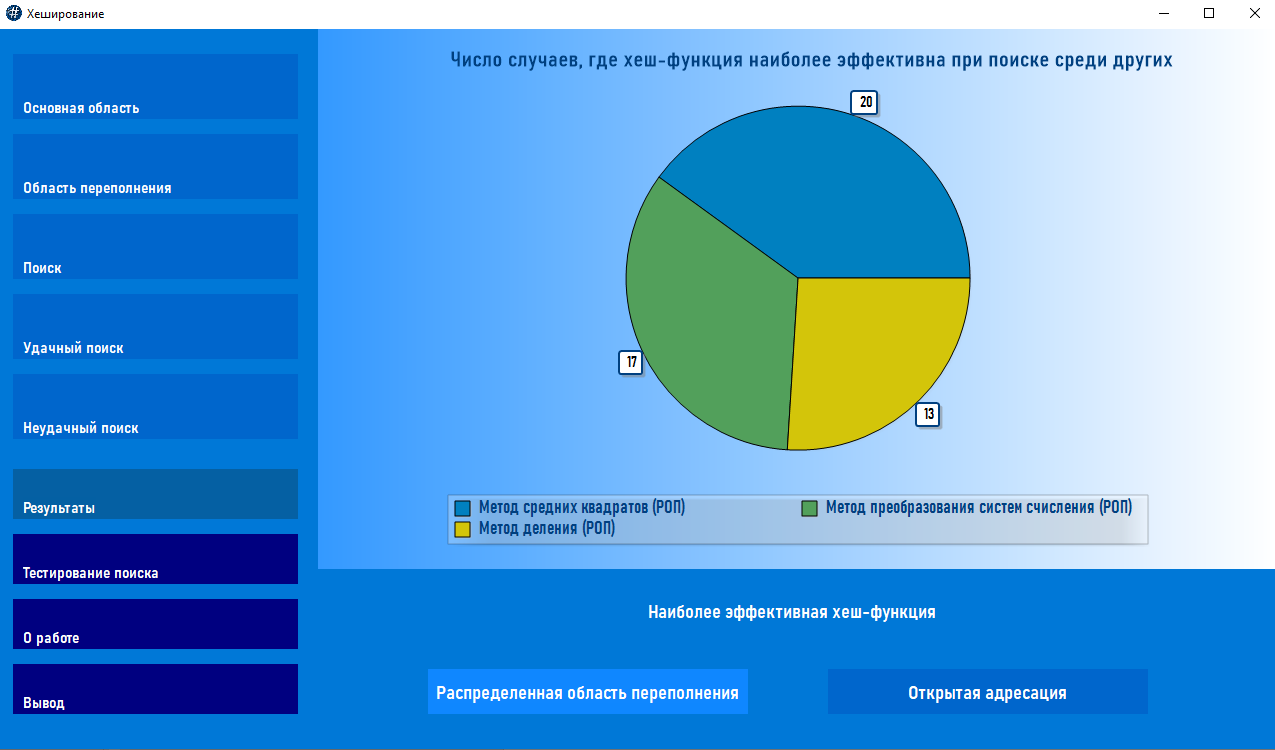
При нажатии на шестую панель с подписью «Результаты» отображается круговая диаграмма (рис. 6.3), демонстрирующая число случаев, в котором каждая хеш-функция показала себя эффективнее других.

Рисунок 6.3 – Приложение с активной панелью «Результаты»

Нижнее меню позволяет переключаться между методами организации области переполнения.

При нажатии на панель «Тестирование поиска» включается окно настраиваемого поиска (рис. 6.4). Пользователь может выбрать параметры из предложенных и произвести поиск введенного с клавиатуры ключа в созданной заранее хеш-таблице.

Необходимо учитывать, что ключ шестисимвольный. Программное средство не позволит ввести больше шести символов, а также выдаст сообщение об ошибке в случае ввода менее, чем шести символов.

Доступные параметры поиска: метод организации области переполнения, метод хеширования, плотность заполнения, число пакетов.

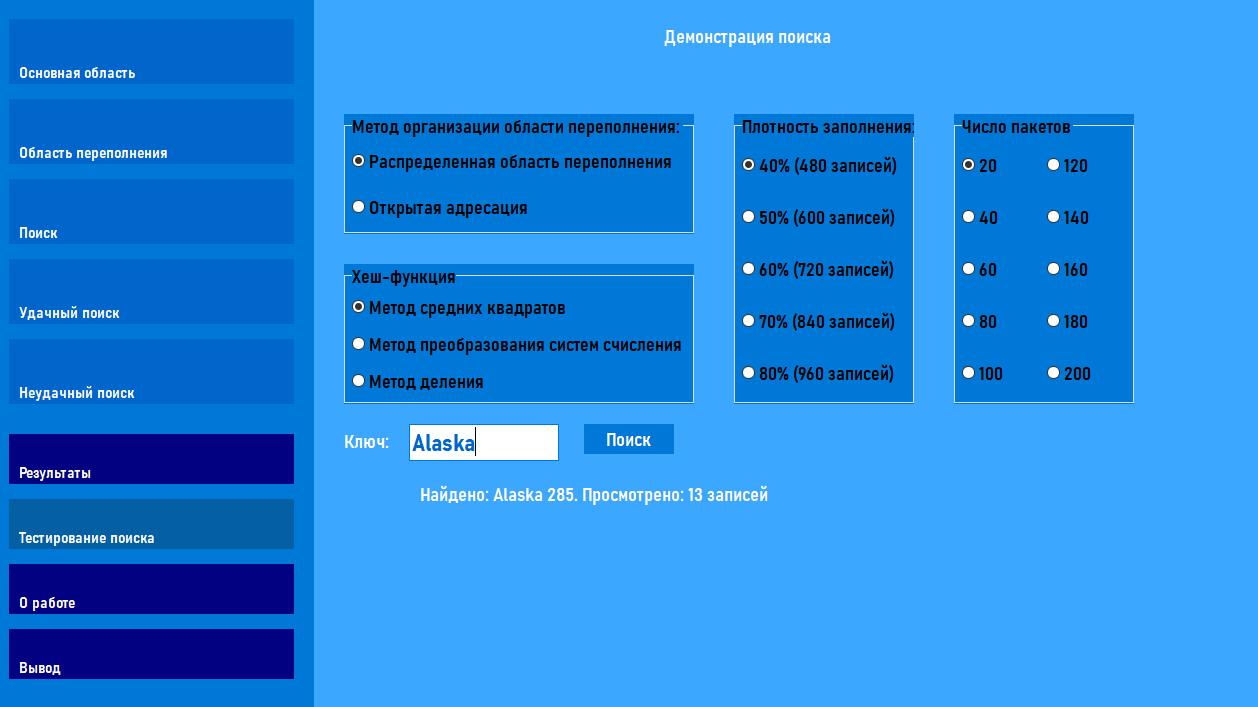


Рисунок 6.4 – окно настраиваемого поиска

Результат поиска отображается в виде информации о том, найдена запись или нет, а также количество просмотренных записей. При удачном поиске выводится ключ и найденное значение.

При нажатии на панель «О работе» отображается справочная информация о использованных алгоритмах.

При нажатии на панель «Вывод» отображается вывод о проведенной работе.

Программное средство поддерживает масштабирование. Максимальный размер формы – 1920x1280, минимальный – 1024x576. Разрешение по умолчанию – 720x1280. Оптимальное соотношение сторон – 16:9.

Заключение

В ходе данной работы создано программное средство, обеспечивающее отображение результатов использования алгоритмов хеширования в сочетании с разными методами организации области переполнения и процентом заполнения основной области.

Путем анализа графического представления полученных данных установлена эффективность алгоритма деления в сочетании с открытой адресацией. Данная информация может быть использована системными администраторами при организации баз данных, проектировщиками языков программирования при создании словарей (хеш-таблиц).

При этом в ходе работы мною получен опыт работы с парадигмой ООП, а также с проектированием графических пользовательских интерфейсов, их адаптацией для удобства взаимодействия.

Список литературы

[1] Кормен, Т. Алгоритмы: построение и анализ, 3-е издание / Т.Х. Кормен [и др.]— М. «Вильямс», 2013. — 1328 с.

[2] Куликов, С.С. Методические указания к выполнению лабораторных работ по курсу "Базы данных" (часть 1) /С.С. Куликов. – БГУИР, Минск, 2009 – 19 с.

[3] Копачев, А.Г. Базы данных, знаний и экспертные системы Методическое пособие для студентов 3-4 курсов специальности 40 01 01 «Программное обеспечение информационных технологий» ч.1/ А.Г. Копачев – БГУИР, Минск – 151 с.

[4] Habr [Электронный ресурс] – Delphi. Что таит в себе TDictionary. – Режим доступа: <https://habr.com/ru/post/282902/>

[5] Apple Documentation [Электронный ресурс] – Dictionary. – Режим доступа: <https://developer.apple.com/documentation/swift/dictionary>

[6] Habr [Электронный ресурс] – Реализация словаря в Python. – Режим доступа: <https://habr.com/ru/company/otus/blog/448350/>

[7] RSDN [Электронный ресурс] – Хеширование. – Режим доступа: <https://rsdn.org/article/alg/bintree/hash.xml>

Приложение A

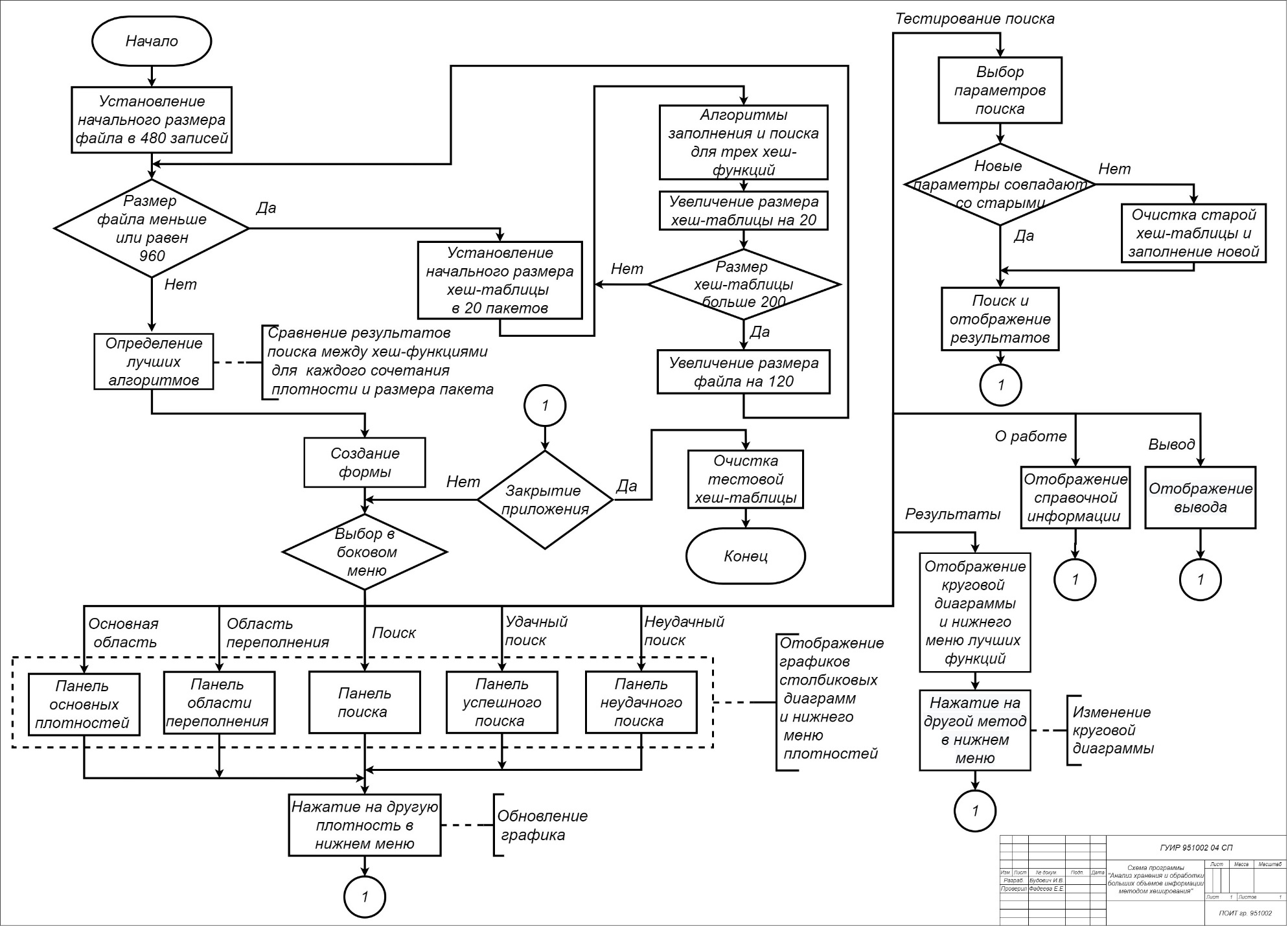


Рисунок А.1 – Схема алгоритма программы

Приложение Б

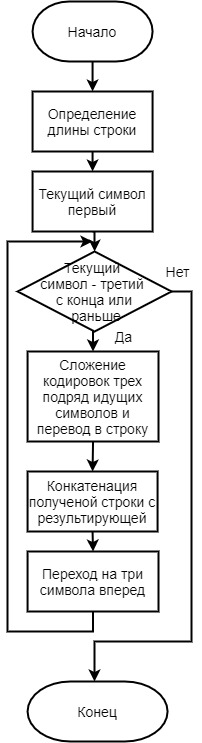


Рисунок Б.1 – Схема алгоритма преобразования строкового значения в числовое

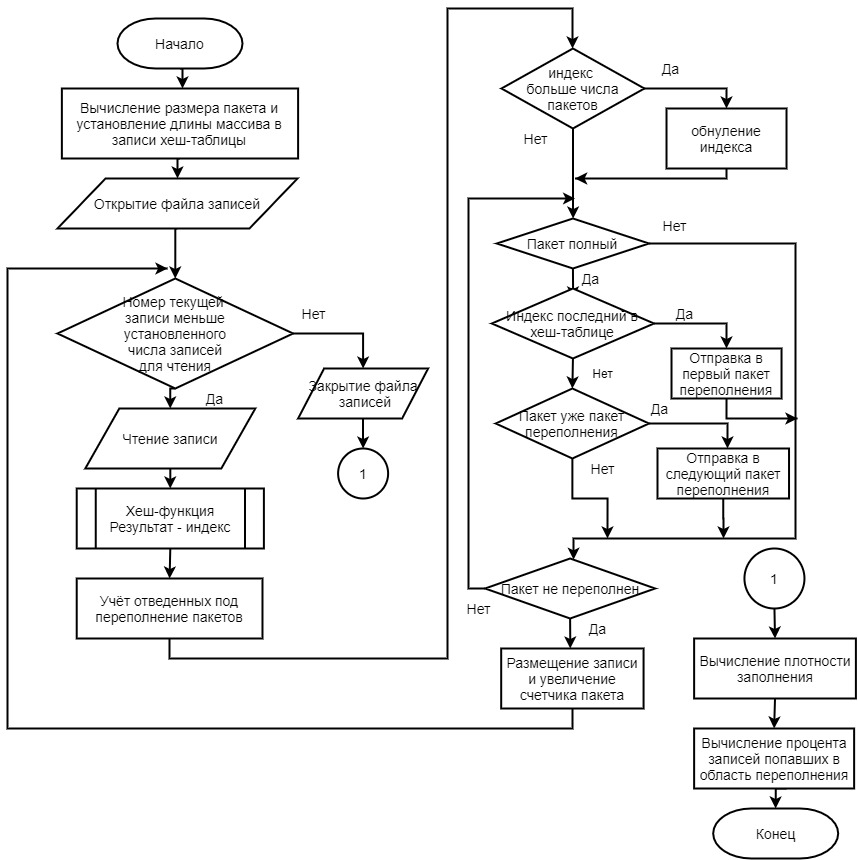


Рисунок Б.2 – Схема алгоритма размещения для распределенной области переполнения

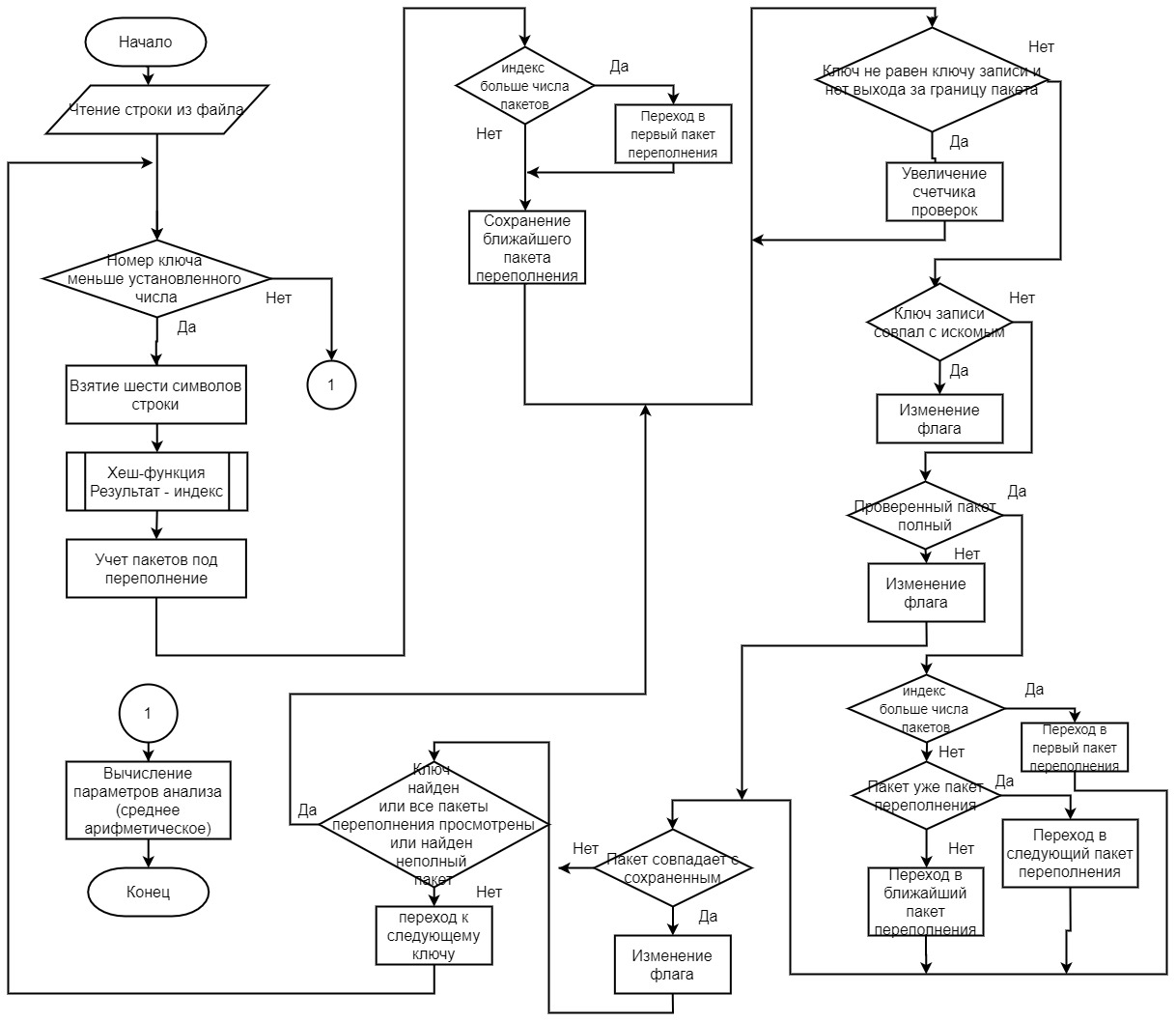


Рисунок Б.3 – Схема алгоритма поиска для распределенной области переполнения

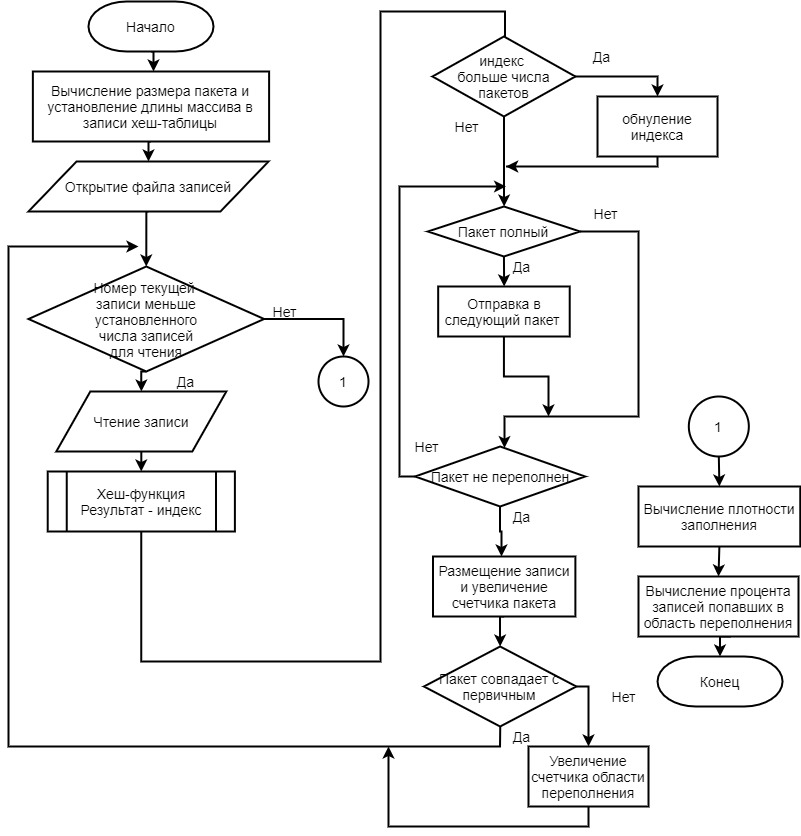


Рисунок Б.4 – Схема алгоритма заполнения для открытой адресации

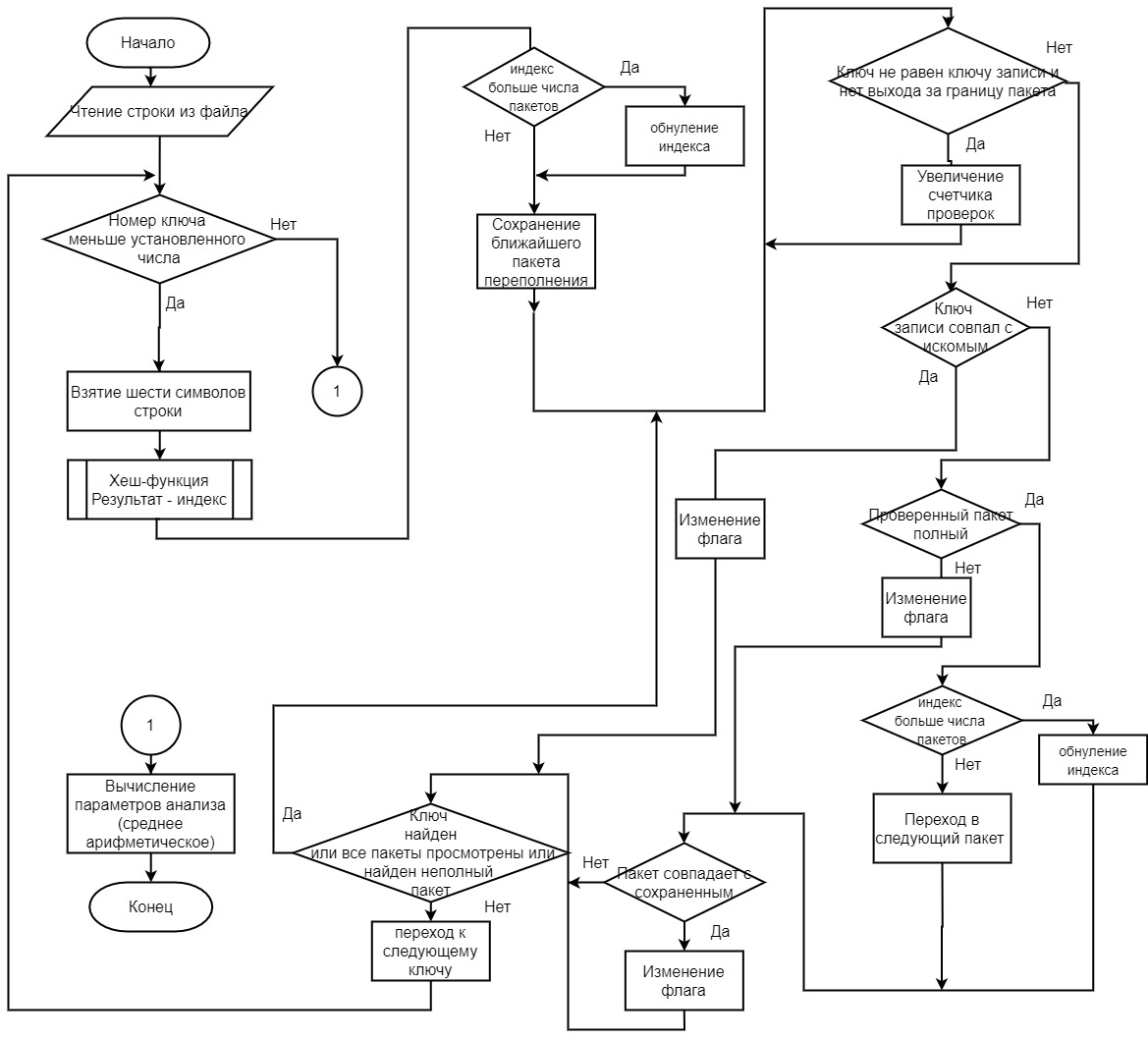


Рисунок Б.5 – схема алгоритма поиска для открытой адресации

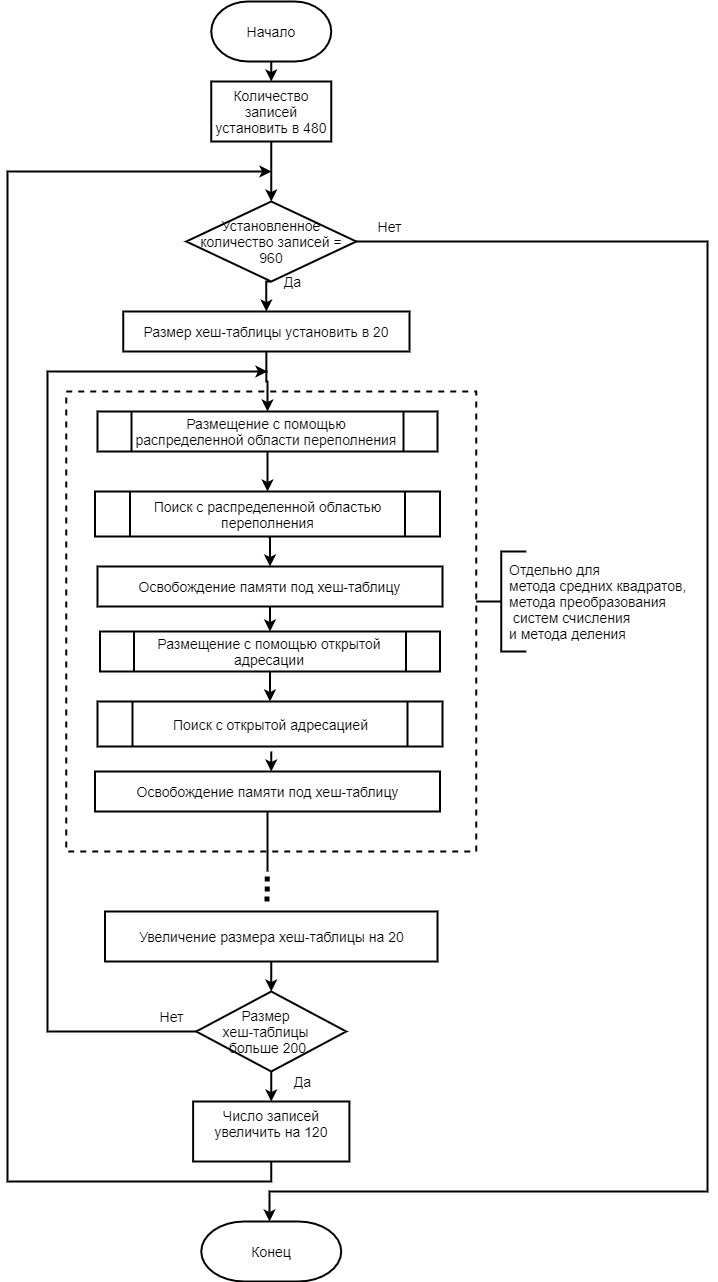


Рисунок Б.6 – схема алгоритма сбора данных

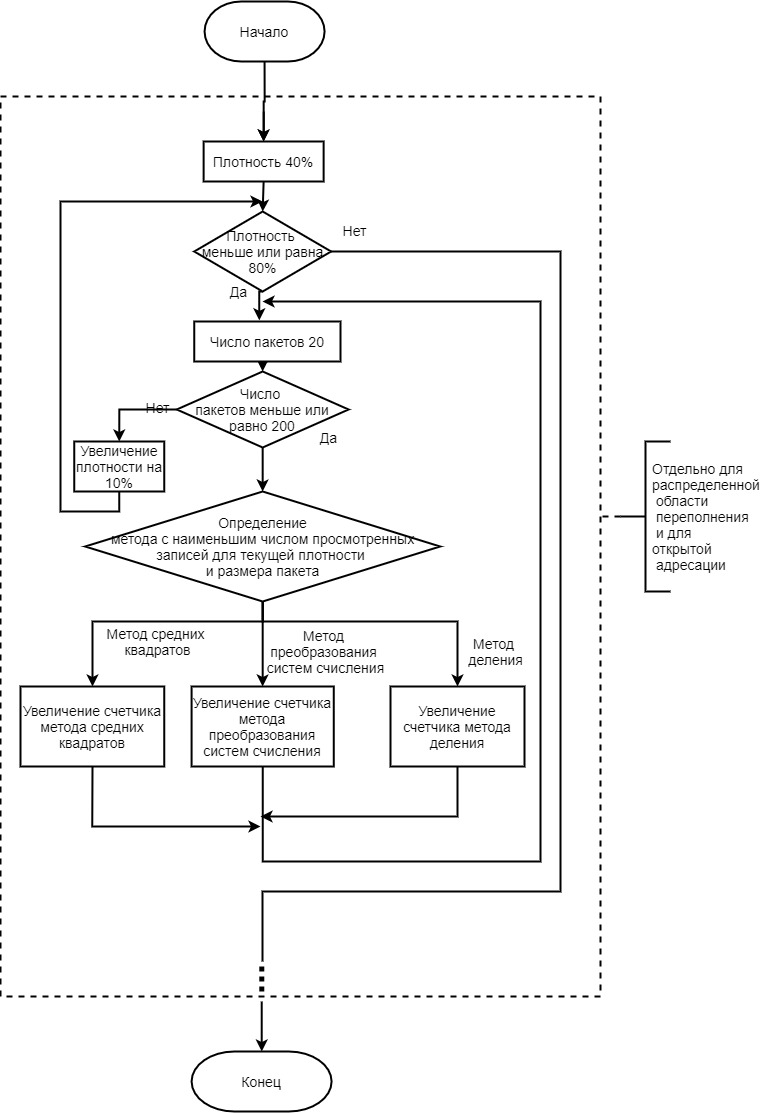


Рисунок Б.7 – Схема алгоритма подсчета для круговых диаграмм

Приложение В

Код модуля Types

unit Types;

interface

const N = 10;

PotentialVolume = 1200;

type

InputRecord = record

key: string[6];

value: integer;

end;

constArray = array [1..5] of integer;

data = record

percentageOverflow: real;

mainDensity: real;

checkedAverage: real;

checkedAverageNotFound:real;

checkedAverageFound: Real;

end;

OrganizationRecord = record

averageSquare: data;

system: data;

division: data;

end;

UnterChartRecord = record

HashTableSize: integer;

Distributed : OrganizationRecord;

Peterson : OrganizationRecord;

end;

TUnterChartArray = Array [1..N] of UnterChartRecord;

UberChartRecord = record

fileVolume: integer;

UnterChart: TUnterChartArray;

end;

TChartArray = array [1..5] of UberChartRecord;

InputFile = file of InputRecord;

TBucket = record

Bucket : array of InputRecord;

Volume: SmallInt;

end;

TPieRecord = record

AverageSquare: integer;

SystemConversion: integer;

Division: integer;

end;

THashTable = array of TBucket;

THashFunction = function(const key: string; const range, HashTableSize: integer): int64;

SearchParam = record

OverflowOrganizationMethod: byte;

HashFunction: THashFunction;

HashTableSize: Integer;

FileVolume: Integer;

end;

implementation

end.

Код модуля HashAlgorithm

unit HashAlgorithm;

interface

uses SysUtils, Types;

const

RangeAverageSquareDistributed: constArray = (115000, 115000, 115000,

135000, 140000);

RangeSystemDistributed: constArray = (1100, 1100, 1100, 1300, 1333);

RangeDivisionDistributed: constArray = (90, 90, 90, 80, 75);

overbucket: constArray = (10, 10, 10, 5, 4);

RangeAverageSquarePeterson = 100500;

RangeSystemPeterson = 1000;

RangeDivisionPeterson = 100;

procedure GetHashProcessResults(out chartArray: TChartArray);

procedure GetMostSufficient(const chartArray: TChartArray;

out PetersonPieRecord, DistributedPieRecord: TPieRecord);

procedure HashDistributed(const HashFunction: THashFunction;

const range, HashTableSize, FileVolume, overbucket: integer;

out HashTable: THashTable; out percentageOverflow, mainDensity: real);

function SearchKeyDistributed(key: string; HashTable: THashTable;

const HashFunction: THashFunction; const range, FileVolume,

overbucket: integer; out checked: integer; out value: integer): boolean;

procedure HashPeterson(const HashFunction: THashFunction;

const range, HashTableSize, FileVolume: integer; out HashTable: THashTable;

out percentageOverflow, mainDensity: real);

function SearchKeyPeterson(key: string; HashTable: THashTable;

const HashFunction: THashFunction; const range, FileVolume: integer;

out checked: integer; out value: integer): boolean;

function HashAverageSquare(const key: string;

const range, HashTableSize: integer): int64;

function HashSystemConversion(const key: string;

const range, HashTableSize: integer): int64;

function HashDivision(const key: string;

const range, HashTableSize: integer): int64;

implementation

function StringToDecimal(const key: string): int64;

var

i, len: integer;

tempString: string;

begin

len := Length(key);

tempString := '';

i := 1;

while i < len - 1 do

begin

tempString := tempString + IntToStr(ord(key[i]) + ord(key[i + 1]) +

ord(key[i + 2]));

inc(i, 3);

end;

result := StrToInt(tempString);

end;

function HashAverageSquare(const key: string;

const range, HashTableSize: integer): int64;

var

Seed: int64;

begin

Seed := StringToDecimal(key);

Seed := sqr(Seed);

result := (Seed div 1000) mod 100000;

result := round(result \* HashTableSize / range);

end;

function HashSystemConversion(const key: string;

const range, HashTableSize: integer): int64;

var

str: string;

i, len, power: integer;

begin

str := IntToStr(StringToDecimal(key));

len := Length(str);

power := 1;

result := 0;

for i := len downto 1 do

begin

result := result + StrToInt(str[i]) \* power;

power := power \* 17;

end;

result := result mod 1000;

result := round(result \* HashTableSize / range);

end;

function HashDivision(const key: string;

const range, HashTableSize: integer): int64;

type

TDivArray = array [1 .. 10] of integer;

const

DivisorArray: TDivArray = (19, 37, 59, 79, 97, 113, 139, 157, 179, 199);

begin

result := StringToDecimal(key);

result := result mod DivisorArray[HashTableSize div 20];

result := round(result \* (range / 100));

end;

procedure HashDistributed(const HashFunction: THashFunction;

const range, HashTableSize, FileVolume, overbucket: integer;

out HashTable: THashTable; out percentageOverflow, mainDensity: real);

var

myfile: InputFile;

BucketSize: integer;

i: integer;

tempRec: InputRecord;

address: integer;

sum: integer;

begin

BucketSize := round(PotentialVolume / HashTableSize);

SetLength(HashTable, HashTableSize);

for i := 0 to HashTableSize - 1 do

begin

HashTable[i].Volume := 0;

SetLength(HashTable[i].Bucket, BucketSize);

end;

Assign(myfile, 'HashData.txt');

Reset(myfile);

tempRec.value := 0;

while tempRec.value < FileVolume do

begin

Read(myfile, tempRec);

address := HashFunction(tempRec.key, range, HashTableSize);

address := address + trunc(address / (overbucket - 1));

if address >= HashTableSize then

address := 0;

repeat

if HashTable[address].Volume = BucketSize then

begin

if address >= HashTableSize - 1 then

address := overbucket - 1;

if ((address + 1) mod overbucket = 0) and (address <> 0) then

inc(address, overbucket)

else

address := overbucket \* (address div overbucket + 1) - 1;

end;

until HashTable[address].Volume < BucketSize;

HashTable[address].Bucket[HashTable[address].Volume] := tempRec;

inc(HashTable[address].Volume);

end;

sum := 0;

i := overbucket - 1;

while i < HashTableSize do

begin

sum := sum + HashTable[i].Volume;

inc(i, overbucket);

end;

percentageOverflow := sum / FileVolume \* 100;

mainDensity := 0;

i := 0;

while i < HashTableSize do

begin

mainDensity := mainDensity + HashTable[i].Volume / BucketSize;

inc(i);

if (i + 1) mod overbucket = 0 then

inc(i);

end;

mainDensity := mainDensity / (HashTableSize \* (1 - 1 / overbucket)) \* 100;

CloseFile(myfile);

end;

procedure SearchDistributed(HashTable: THashTable;

const HashFunction: THashFunction; const range, FileVolume,

overbucket: integer; out checkedAverage, checkedAverageNotFound,

checkedAverageFound: real);

var

k: integer;

key: string[6];

f: textfile;

checked: integer;

temp: string;

kmax: integer;

value:integer;

begin

Assign(f, 'Search.txt');

Reset(f);

read(f, temp);

CloseFile(f);

checkedAverage := 0;

checkedAverageNotFound := 0;

checkedAverageFound := 0;

kmax := round(FileVolume \* 1.2);

for k := 1 to kmax do

begin

key := copy(temp, k \* 6 - 5, 6);

if SearchKeyDistributed(key, HashTable, HashFunction, range, FileVolume,

overbucket, checked, value) then

begin

checkedAverageFound := checkedAverageFound + checked;

end

else

begin

checkedAverageNotFound := checkedAverageNotFound + checked;

end;

checkedAverage := checkedAverage + checked;

end;

checkedAverage := checkedAverage / round(FileVolume \* 1.2);

checkedAverageNotFound := checkedAverageNotFound / round(FileVolume \* 0.2);

checkedAverageFound := checkedAverageFound / FileVolume;

end;

procedure HashPeterson(const HashFunction: THashFunction;

const range, HashTableSize, FileVolume: integer; out HashTable: THashTable;

out percentageOverflow, mainDensity: real);

var

myfile: InputFile;

BucketSize: integer;

i: integer;

tempRec: InputRecord;

address: integer;

sum: integer;

begin

BucketSize := round(PotentialVolume / HashTableSize);

SetLength(HashTable, HashTableSize);

for i := 0 to HashTableSize - 1 do

begin

HashTable[i].Volume := 0;

SetLength(HashTable[i].Bucket, BucketSize);

end;

sum := 0;

Assign(myfile, 'HashData.txt');

Reset(myfile);

tempRec.value := 0;

while tempRec.value < FileVolume do

begin

Read(myfile, tempRec);

address := HashFunction(tempRec.key, range, HashTableSize);

if address >= HashTableSize then

address := 0;

repeat

if HashTable[address].Volume = BucketSize then

begin

if address >= HashTableSize - 1 then

address := 0

else

inc(address);

end;

until HashTable[address].Volume < BucketSize;

HashTable[address].Bucket[HashTable[address].Volume] := tempRec;

inc(HashTable[address].Volume);

if address <> HashFunction(tempRec.key, range, HashTableSize) then

inc(sum);

end;

CloseFile(myfile);

percentageOverflow := sum / FileVolume \* 100;

mainDensity := (FileVolume - sum) / PotentialVolume \* 100;

end;

procedure SearchPeterson(HashTable: THashTable;

const HashFunction: THashFunction; const range, FileVolume: integer;

out checkedAverage, checkedAverageNotFound, checkedAverageFound: real);

var

k, kmax: integer;

key: string[6];

f: textfile;

checked: integer;

temp: string;

value: integer;

begin

Assign(f, 'Search.txt');

Reset(f);

read(f, temp);

CloseFile(f);

checkedAverage := 0;

checkedAverageNotFound := 0;

checkedAverageFound := 0;

kmax := round(FileVolume \* 1.2);

for k := 1 to kmax do

begin

key := copy(temp, k \* 6 - 5, 6);

if SearchKeyPeterson(key, HashTable, HashFunction, range, FileVolume,

checked, value) then

begin

checkedAverageFound := checkedAverageFound + checked;

end

else

begin

checkedAverageNotFound := checkedAverageNotFound + checked;

end;

checkedAverage := checkedAverage + checked;

end;

checkedAverage := checkedAverage / round(FileVolume \* 1.2);

checkedAverageNotFound := checkedAverageNotFound / round(FileVolume \* 0.2);

checkedAverageFound := checkedAverageFound / FileVolume;

end;

function SearchKeyDistributed(key: string; HashTable: THashTable;

const HashFunction: THashFunction; const range, FileVolume,

overbucket: integer; out checked, value: integer): boolean;

var

i, address, last, HashTableSize, BucketSize: integer;

flag, flag2, flag3: boolean;

begin

HashTableSize := Length(HashTable);

BucketSize := round(PotentialVolume / HashTableSize);

address := HashFunction(key, range, HashTableSize);

address := address + trunc(address / (overbucket - 1));

value:=-1;

if address >= HashTableSize then

address := 0;

if address = 0 then

last := overbucket - 1

else

last := overbucket \* (address div overbucket + 1) - 1;

flag := false;

flag2 := true;

flag3 := false;

checked := 1;

repeat

i := 0;

while (key <> HashTable[address].Bucket[i].key) and

(i < HashTable[address].Volume) do

begin

inc(i);

inc(checked);

end;

if key = HashTable[address].Bucket[i].key then

begin

flag := true;

value:=HashTable[address].Bucket[i].value;

end

else if HashTable[address].Volume < BucketSize then

flag3 := true

else

begin

if (address >= (Length(HashTable) - overbucket + 1)) then

address := overbucket - 1

else

begin

if ((address + 1) mod overbucket = 0) and (address <> 0) then

inc(address, overbucket)

else

address := overbucket \* (address div overbucket + 1) - 1;

end;

end;

if address = last then

flag2 := not flag2;

// flag - если найден

// flag 2 - выход если не найдет спустя весь круг

// flag 3 - выход если не найден после неполного пакета

until flag or ((address = last) and flag2) or flag3;

result := flag;

end;

function SearchKeyPeterson(key: string; HashTable: THashTable;

const HashFunction: THashFunction; const range, FileVolume: integer;

out checked, value: integer): boolean;

var

i, address, last, HashTableSize, BucketSize: integer;

flag, flag2, flag3: boolean;

begin

HashTableSize := Length(HashTable);

BucketSize := round(PotentialVolume / Length(HashTable));

address := HashFunction(key, range, HashTableSize);

if address >= HashTableSize then

address := 0;

last := address;

value:=-1;

flag := false;

flag2 := true;

flag3 := false;

checked := 1;

repeat

i := 0;

while (key <> HashTable[address].Bucket[i].key) and

(i < HashTable[address].Volume) do

begin

inc(i);

inc(checked);

end;

if key = HashTable[address].Bucket[i].key then

begin

flag := true;

value:=HashTable[address].Bucket[i].value;

end

else if HashTable[address].Volume < BucketSize then

flag3 := true

else

begin

if (address >= HashTableSize - 1) then

address := 0

else

inc(address);

end;

if address = last then

flag2 := not flag2;

// flag - если найден

// flag 2 - выход если не найдет спустя весь круг

// flag 3 - выход если не найден после неполного пакета

until flag or ((address = last) and flag2) or flag3;

result := flag;

end;

procedure GetHashProcessResults(out chartArray: TChartArray);

var

HashTable: THashTable;

i: integer;

j: integer;

FileVolume: integer;

begin

FileVolume := 480;

j := 1;

while FileVolume <= 960 do

begin

chartArray[j].FileVolume := FileVolume;

i := 1;

chartArray[j].UnterChart[1].HashTableSize := 20;

repeat

HashDistributed(HashAverageSquare, RangeAverageSquareDistributed[j],

chartArray[j].UnterChart[i].HashTableSize, FileVolume, overbucket[j],

HashTable, chartArray[j].UnterChart[i].Distributed.averageSquare.

percentageOverflow, chartArray[j].UnterChart[i]

.Distributed.averageSquare.mainDensity);

SearchDistributed(HashTable, HashAverageSquare,

RangeAverageSquareDistributed[j], FileVolume, overbucket[j],

chartArray[j].UnterChart[i].Distributed.averageSquare.checkedAverage,

chartArray[j].UnterChart[i].Distributed.averageSquare.

checkedAverageNotFound, chartArray[j].UnterChart[i]

.Distributed.averageSquare.checkedAverageFound);

SetLength(HashTable, 0);

HashPeterson(HashAverageSquare, RangeAverageSquarePeterson,

chartArray[j].UnterChart[i].HashTableSize, FileVolume, HashTable,

chartArray[j].UnterChart[i].Peterson.averageSquare.percentageOverflow,

chartArray[j].UnterChart[i].Peterson.averageSquare.mainDensity);

SearchPeterson(HashTable, HashAverageSquare, RangeAverageSquarePeterson,

FileVolume, chartArray[j].UnterChart[i].Peterson.averageSquare.

checkedAverage, chartArray[j].UnterChart[i]

.Peterson.averageSquare.checkedAverageNotFound,

chartArray[j].UnterChart[i].Peterson.averageSquare.checkedAverageFound);

SetLength(HashTable, 0);

HashDistributed(HashSystemConversion, RangeSystemDistributed[j],

chartArray[j].UnterChart[i].HashTableSize, FileVolume, overbucket[j],

HashTable, chartArray[j].UnterChart[i]

.Distributed.System.percentageOverflow,

chartArray[j].UnterChart[i].Distributed.System.mainDensity);

SearchDistributed(HashTable, HashSystemConversion,

RangeSystemDistributed[j], FileVolume, overbucket[j],

chartArray[j].UnterChart[i].Distributed.System.checkedAverage,

chartArray[j].UnterChart[i].Distributed.System.checkedAverageNotFound,

chartArray[j].UnterChart[i].Distributed.System.checkedAverageFound);

SetLength(HashTable, 0);

HashPeterson(HashSystemConversion, RangeSystemPeterson,

chartArray[j].UnterChart[i].HashTableSize, FileVolume, HashTable,

chartArray[j].UnterChart[i].Peterson.System.percentageOverflow,

chartArray[j].UnterChart[i].Peterson.System.mainDensity);

SearchPeterson(HashTable, HashSystemConversion, RangeSystemPeterson,

FileVolume, chartArray[j].UnterChart[i].Peterson.System.checkedAverage,

chartArray[j].UnterChart[i].Peterson.System.checkedAverageNotFound,

chartArray[j].UnterChart[i].Peterson.System.checkedAverageFound);

SetLength(HashTable, 0);

HashDistributed(HashDivision, RangeDivisionDistributed[j],

chartArray[j].UnterChart[i].HashTableSize, FileVolume, overbucket[j],

HashTable, chartArray[j].UnterChart[i]

.Distributed.division.percentageOverflow,

chartArray[j].UnterChart[i].Distributed.division.mainDensity);

SearchDistributed(HashTable, HashDivision, RangeDivisionDistributed[j],

FileVolume, overbucket[j], chartArray[j].UnterChart[i]

.Distributed.division.checkedAverage,

chartArray[j].UnterChart[i].Distributed.division.checkedAverageNotFound,

chartArray[j].UnterChart[i].Distributed.division.checkedAverageFound);

SetLength(HashTable, 0);

HashPeterson(HashDivision, RangeDivisionPeterson,

chartArray[j].UnterChart[i].HashTableSize, FileVolume, HashTable,

chartArray[j].UnterChart[i].Peterson.division.percentageOverflow,

chartArray[j].UnterChart[i].Peterson.division.mainDensity);

SearchPeterson(HashTable, HashDivision, RangeDivisionPeterson, FileVolume,

chartArray[j].UnterChart[i].Peterson.division.checkedAverage,

chartArray[j].UnterChart[i].Peterson.division.checkedAverageNotFound,

chartArray[j].UnterChart[i].Peterson.division.checkedAverageFound);

SetLength(HashTable, 0);

inc(i);

if i <= 10 then

chartArray[j].UnterChart[i].HashTableSize := chartArray[j].UnterChart

[i - 1].HashTableSize + 20;

until i > 10;

inc(FileVolume, 120);

inc(j);

end;

end;

procedure GetMostSufficient(const chartArray: TChartArray;

out PetersonPieRecord, DistributedPieRecord: TPieRecord);

var

i, j: integer;

begin

PetersonPieRecord.averageSquare := 0;

PetersonPieRecord.SystemConversion := 0;

PetersonPieRecord.division := 0;

for i := 1 to 5 do

for j := 1 to 10 do

begin

if chartArray[i].UnterChart[j].Peterson.averageSquare.checkedAverage <

chartArray[i].UnterChart[j].Peterson.System.checkedAverage then

begin

if chartArray[i].UnterChart[j].Peterson.averageSquare.checkedAverage <

chartArray[i].UnterChart[j].Peterson.division.checkedAverage then

inc(PetersonPieRecord.averageSquare)

else

inc(PetersonPieRecord.division);

end

else

begin

if chartArray[i].UnterChart[j].Peterson.System.checkedAverage <

chartArray[i].UnterChart[j].Peterson.division.checkedAverage then

inc(PetersonPieRecord.SystemConversion)

else

inc(PetersonPieRecord.division);

end;

end;

DistributedPieRecord.averageSquare := 0;

DistributedPieRecord.SystemConversion := 0;

DistributedPieRecord.division := 0;

for i := 1 to 5 do

for j := 1 to 10 do

begin

if chartArray[i].UnterChart[j].Distributed.averageSquare.checkedAverage <

chartArray[i].UnterChart[j].Distributed.System.checkedAverage then

begin

if chartArray[i].UnterChart[j].Distributed.averageSquare.checkedAverage

< chartArray[i].UnterChart[j].Distributed.division.checkedAverage then

inc(DistributedPieRecord.averageSquare)

else

inc(DistributedPieRecord.division);

end

else

begin

if chartArray[i].UnterChart[j].Distributed.System.checkedAverage <

chartArray[i].UnterChart[j].Distributed.division.checkedAverage then

inc(DistributedPieRecord.SystemConversion)

else

inc(DistributedPieRecord.division);

end;

end;

end;

end.

Код модуля MainFormUnit

unit MainFormUnit;

interface

uses

Winapi.Windows, Winapi.Messages, System.SysUtils,

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.ExtCtrls,

VCLTee.Series, Vcl.StdCtrls,

VCLTee.Chart, Types, HashAlgorithm, VCLTee.TeeProcs,

VCLTee.TeeGDIPlus, VCLTee.TeEngine;

type

TMainForm = class(TForm)

MenuPanel: TPanel;

DensityPanel: TPanel;

DensityLabel: TLabel;

OverflowPanel: TPanel;

OverflowLabel: TLabel;

SearchPanel: TPanel;

SearchLabel: TLabel;

GoodSearchPanel: TPanel;

GoodSearchLabel: TLabel;

BadSearchPanel: TPanel;

BadSearchLabel: TLabel;

InfoPanel: TPanel;

InfoLabel: TLabel;

SummaryPanel: TPanel;

SummaryLabel: TLabel;

MemoPanel: TPanel;

Memo1: TMemo;

Chart1: TChart;

Series1: TBarSeries;

Series2: TBarSeries;

Series3: TBarSeries;

Series4: TBarSeries;

Memo2: TMemo;

DensControlPanel: TPanel;

DensPanel1: TPanel;

DensPanel2: TPanel;

DensPanel3: TPanel;

DensPanel4: TPanel;

DensPanel5: TPanel;

Label1: TLabel;

Series5: TBarSeries;

Series6: TBarSeries;

GrafPanel: TPanel;

GrafLabel: TLabel;

ChartPanel: TPanel;

ResultPanel: TPanel;

GraphMenuPanel: TPanel;

Chart2: TChart;

Series7: TPieSeries;

PieLabel: TLabel;

ROPPanel: TPanel;

OAPanel: TPanel;

TestSearchPanel: TPanel;

TitleLabel: TLabel;

KeyEdit: TEdit;

GetSearchResultPanel: TPanel;

OverflowOrganizationMethodChoice: TRadioGroup;

DensityChoice: TRadioGroup;

BucketSizeChoice: TRadioGroup;

HashFunctionChoice: TRadioGroup;

Label3: TLabel;

Panel1: TPanel;

SearchTestPanel: TPanel;

SearchTestPanelLabel: TLabel;

procedure FormCreate(Sender: TObject);

procedure DensityPanelClick(Sender: TObject);

procedure OverflowPanelClick(Sender: TObject);

procedure GoodSearchPanelClick(Sender: TObject);

procedure SearchPanelClick(Sender: TObject);

procedure BadSearchClick(Sender: TObject);

procedure InfoPanelClick(Sender: TObject);

procedure SummaryPanelClick(Sender: TObject);

procedure MouseEnter(Sender: TObject);

procedure MouseLeave(Sender: TObject);

procedure ToResize(Sender: TObject);

procedure Chart1MouseWheelDown(Sender: TObject; Shift: TShiftState;

MousePos: TPoint; var Handled: Boolean);

procedure Chart1MouseWheelUp(Sender: TObject; Shift: TShiftState;

MousePos: TPoint; var Handled: Boolean);

procedure SeriesMouseEnter(Sender: TObject);

procedure SeriesMouseLeft(Sender: TObject);

procedure Chart1Click(Sender: TObject);

procedure DensPanel1Click(Sender: TObject);

procedure DensPanel2Click(Sender: TObject);

procedure DensPanel3Click(Sender: TObject);

procedure DensPanel4Click(Sender: TObject);

procedure DensPanel5Click(Sender: TObject);

procedure SetDefault(Sender: TObject);

procedure SetDefaultDensityPanel(Sender: TObject);

procedure LabelMouseEnter(Sender: TObject);

procedure GrafPanelClick(Sender: TObject);

procedure ROPPanelClick(Sender: TObject);

procedure OAPanelClick(Sender: TObject);

procedure SearchTestPanelClick(Sender: TObject);

procedure GetSearchResultPanelClick(Sender: TObject);

procedure FormDestroy(Sender: TObject);

private

public

end;

var

MainForm: TMainForm;

implementation

type

TMethod = procedure(Sender: TObject) of object;

var

ChartArray: TChartArray;

PetersonPie, DistributedPie: TPieRecord;

ChartTitleString: String;

CurrentChoice: integer;

ActiveMethod: TMethod;

ActiveMenuPanel: TPanel;

ActiveDensityControl: TPanel;

OldParam: SearchParam;

HashTable: THashTable;

{$R \*.dfm}

procedure TMainForm.FormCreate(Sender: TObject);

begin

GetHashProcessResults(ChartArray);

GetMostSufficient(ChartArray, PetersonPie, DistributedPie);

CurrentChoice := 1;

ActiveMenuPanel := nil;

InfoPanelClick(Sender);

ActiveDensityControl := DensPanel1;

DensPanel1Click(DensPanel1);

end;

procedure TMainForm.FormDestroy(Sender: TObject);

begin

SetLength(HashTable, 0);

end;

procedure TMainForm.SetDefault(Sender: TObject);

begin

ResultPanel.Visible := false;

if ActiveMenuPanel <> nil then

ActiveMenuPanel.Color := clHotLight

else

begin

InfoPanel.Color := clNavy;

SummaryPanel.Color := clNavy;

GrafPanel.Color := clNavy;

SearchTestPanel.Color := clNavy;

TestSearchPanel.Visible := false;

ResultPanel.Visible := false;

ChartPanel.Visible := true;

Memo2.Visible := false;

ActiveDensityControl.Color := clHotLight;

ActiveDensityControl := DensPanel1;

CurrentChoice := 1;

DensPanel1.Color := $00FF870F;

end;

ActiveMenuPanel := (Sender as TPanel);

(Sender as TPanel).Color := $00FF870F;

Self.Chart1.SeriesList[0].Clear;

Self.Chart1.SeriesList[1].Clear;

Self.Chart1.SeriesList[2].Clear;

Self.Chart1.SeriesList[3].Clear;

Self.Chart1.SeriesList[4].Clear;

Self.Chart1.SeriesList[5].Clear;

Chart1.Title.Text[0] := ChartTitleString;

end;

procedure TMainForm.SetDefaultDensityPanel(Sender: TObject);

begin

ActiveDensityControl.Color := clHotLight;

ActiveDensityControl := (Sender as TPanel);

if (@ActiveMethod <> nil) then

ActiveMethod(Sender);

(Sender as TPanel).Color := $00FF870F;

end;

procedure TMainForm.DensityPanelClick(Sender: TObject);

var

i: byte;

begin

ActiveMethod := DensityPanelClick;

ChartTitleString := 'Плотность заполнения основной области (в процентах)';

SetDefault(DensityPanel);

Self.Memo1.Lines.LoadFromFile('DensInfo.txt', TEncoding.UTF8);

for i := 1 to 10 do

begin

Self.Chart1.SeriesList[0].Add(ChartArray[CurrentChoice].UnterChart[i]

.Distributed.averageSquare.mainDensity,

(IntToStr(ChartArray[CurrentChoice].UnterChart[i].HashTableSize) + 'п. /'

+ IntToStr(round(PotentialVolume / ChartArray[CurrentChoice].UnterChart[i]

.HashTableSize)) + 'з.'));

Self.Chart1.SeriesList[1].Add(ChartArray[CurrentChoice].UnterChart[i]

.Peterson.averageSquare.mainDensity,

(IntToStr(ChartArray[CurrentChoice].UnterChart[i].HashTableSize) + 'п. /'

+ IntToStr(round(PotentialVolume / ChartArray[CurrentChoice].UnterChart[i]

.HashTableSize)) + 'з.'));

Self.Chart1.SeriesList[2].Add(ChartArray[CurrentChoice].UnterChart[i]

.Distributed.System.mainDensity,

(IntToStr(ChartArray[CurrentChoice].UnterChart[i].HashTableSize) + 'п. /'

+ IntToStr(round(PotentialVolume / ChartArray[CurrentChoice].UnterChart[i]

.HashTableSize)) + 'з.'));

Self.Chart1.SeriesList[3].Add(ChartArray[CurrentChoice].UnterChart[i]

.Peterson.System.mainDensity,

(IntToStr(ChartArray[CurrentChoice].UnterChart[i].HashTableSize) + 'п. /'

+ IntToStr(round(PotentialVolume / ChartArray[CurrentChoice].UnterChart[i]

.HashTableSize)) + 'з.'));

Self.Chart1.SeriesList[4].Add(ChartArray[CurrentChoice].UnterChart[i]

.Distributed.Division.mainDensity,

(IntToStr(ChartArray[CurrentChoice].UnterChart[i].HashTableSize) + 'п. /'

+ IntToStr(round(PotentialVolume / ChartArray[CurrentChoice].UnterChart[i]

.HashTableSize)) + 'з.'));

Self.Chart1.SeriesList[5].Add(ChartArray[CurrentChoice].UnterChart[i]

.Peterson.Division.mainDensity,

(IntToStr(ChartArray[CurrentChoice].UnterChart[i].HashTableSize) + 'п. /'

+ IntToStr(round(PotentialVolume / ChartArray[CurrentChoice].UnterChart[i]

.HashTableSize)) + 'з.'));

end;

end;

procedure TMainForm.OverflowPanelClick(Sender: TObject);

var

i: byte;

begin

ActiveMethod := OverflowPanelClick;

ChartTitleString := 'Процент записей в области переполнения';

SetDefault(OverflowPanel);

Self.Memo1.Lines.LoadFromFile('OverflowInfo.txt', TEncoding.UTF8);

for i := 1 to 10 do

begin

Self.Chart1.SeriesList[0].Add(ChartArray[CurrentChoice].UnterChart[i]

.Distributed.averageSquare.percentageOverflow,

(IntToStr(ChartArray[CurrentChoice].UnterChart[i].HashTableSize) + 'п. /'

+ IntToStr(round(PotentialVolume / ChartArray[CurrentChoice].UnterChart[i]

.HashTableSize)) + 'з.'));

Self.Chart1.SeriesList[1].Add(ChartArray[CurrentChoice].UnterChart[i]

.Peterson.averageSquare.percentageOverflow,

(IntToStr(ChartArray[CurrentChoice].UnterChart[i].HashTableSize) + 'п. /'

+ IntToStr(round(PotentialVolume / ChartArray[CurrentChoice].UnterChart[i]

.HashTableSize)) + 'з.'));

Self.Chart1.SeriesList[2].Add(ChartArray[CurrentChoice].UnterChart[i]

.Distributed.System.percentageOverflow,

(IntToStr(ChartArray[i].UnterChart[CurrentChoice].HashTableSize) + 'п. /'

+ IntToStr(round(PotentialVolume / ChartArray[CurrentChoice].UnterChart[i]

.HashTableSize)) + 'з.'));

Self.Chart1.SeriesList[3].Add(ChartArray[CurrentChoice].UnterChart[i]

.Peterson.System.percentageOverflow,

(IntToStr(ChartArray[CurrentChoice].UnterChart[i].HashTableSize) + 'п. /'

+ IntToStr(round(PotentialVolume / ChartArray[CurrentChoice].UnterChart[i]

.HashTableSize)) + 'з.'));

Self.Chart1.SeriesList[4].Add(ChartArray[CurrentChoice].UnterChart[i]

.Distributed.Division.percentageOverflow,

(IntToStr(ChartArray[i].UnterChart[CurrentChoice].HashTableSize) + 'п. /'

+ IntToStr(round(PotentialVolume / ChartArray[CurrentChoice].UnterChart[i]

.HashTableSize)) + 'з.'));

Self.Chart1.SeriesList[5].Add(ChartArray[CurrentChoice].UnterChart[i]

.Peterson.Division.percentageOverflow,

(IntToStr(ChartArray[CurrentChoice].UnterChart[i].HashTableSize) + 'п. /'

+ IntToStr(round(PotentialVolume / ChartArray[CurrentChoice].UnterChart[i]

.HashTableSize)) + 'з.'));

end;

end;

procedure TMainForm.SearchPanelClick(Sender: TObject);

var

i: byte;

begin

ChartTitleString := 'Среднее число просмотренных записей при поиске';

SetDefault(SearchPanel);

ActiveMethod := SearchPanelClick;

Self.Memo1.Lines.LoadFromFile('SearchInfo.txt', TEncoding.UTF8);

for i := 1 to 10 do

begin

Self.Chart1.SeriesList[0].Add(ChartArray[CurrentChoice].UnterChart[i]

.Distributed.averageSquare.checkedAverage,

(IntToStr(ChartArray[CurrentChoice].UnterChart[i].HashTableSize) + 'п. /'

+ IntToStr(round(PotentialVolume / ChartArray[CurrentChoice].UnterChart[i]

.HashTableSize)) + 'з.'));

Self.Chart1.SeriesList[1].Add(ChartArray[CurrentChoice].UnterChart[i]

.Peterson.averageSquare.checkedAverage,

(IntToStr(ChartArray[CurrentChoice].UnterChart[i].HashTableSize) + 'п. /'

+ IntToStr(round(PotentialVolume / ChartArray[CurrentChoice].UnterChart[i]

.HashTableSize)) + 'з.'));

Self.Chart1.SeriesList[2].Add(ChartArray[CurrentChoice].UnterChart[i]

.Distributed.System.checkedAverage,

(IntToStr(ChartArray[CurrentChoice].UnterChart[i].HashTableSize) + 'п. /'

+ IntToStr(round(PotentialVolume / ChartArray[CurrentChoice].UnterChart[i]

.HashTableSize)) + 'з.'));

Self.Chart1.SeriesList[3].Add(ChartArray[CurrentChoice].UnterChart[i]

.Peterson.System.checkedAverage,

(IntToStr(ChartArray[CurrentChoice].UnterChart[i].HashTableSize) + 'п. /'

+ IntToStr(round(PotentialVolume / ChartArray[CurrentChoice].UnterChart[i]

.HashTableSize)) + 'з.'));

Self.Chart1.SeriesList[4].Add(ChartArray[CurrentChoice].UnterChart[i]

.Distributed.Division.checkedAverage,

(IntToStr(ChartArray[CurrentChoice].UnterChart[i].HashTableSize) + 'п. /'

+ IntToStr(round(PotentialVolume / ChartArray[CurrentChoice].UnterChart[i]

.HashTableSize)) + 'з.'));

Self.Chart1.SeriesList[5].Add(ChartArray[CurrentChoice].UnterChart[i]

.Peterson.Division.checkedAverage,

(IntToStr(ChartArray[CurrentChoice].UnterChart[i].HashTableSize) + 'п. /'

+ IntToStr(round(PotentialVolume / ChartArray[CurrentChoice].UnterChart[i]

.HashTableSize)) + 'з.'));

end;

end;

procedure TMainForm.GetSearchResultPanelClick(Sender: TObject);

var

CurrentParam: SearchParam;

range, value: integer;

key: string;

checked: integer;

spare: data;

Curoverbucket: integer;

isFound: Boolean;

begin

key := KeyEdit.Text;

if key.Length <> 6 then Panel1.Caption:='Проверьте ввод'

else

begin

CurrentParam.OverflowOrganizationMethod :=

OverflowOrganizationMethodChoice.ItemIndex;

case HashFunctionChoice.ItemIndex of

0:

CurrentParam.HashFunction := HashAverageSquare;

1:

CurrentParam.HashFunction := HashSystemConversion;

2:

CurrentParam.HashFunction := HashDivision;

end;

case DensityChoice.ItemIndex of

0:

CurrentParam.FileVolume := 480;

1:

CurrentParam.FileVolume := 600;

2:

CurrentParam.FileVolume := 720;

3:

CurrentParam.FileVolume := 840;

4:

CurrentParam.FileVolume := 960;

end;

case BucketSizeChoice.ItemIndex of

0:

CurrentParam.HashTableSize := 20;

1:

CurrentParam.HashTableSize := 40;

2:

CurrentParam.HashTableSize := 60;

3:

CurrentParam.HashTableSize := 80;

4:

CurrentParam.HashTableSize := 100;

5:

CurrentParam.HashTableSize := 120;

6:

CurrentParam.HashTableSize := 140;

7:

CurrentParam.HashTableSize := 160;

8:

CurrentParam.HashTableSize := 180;

9:

CurrentParam.HashTableSize := 200;

end;

if CurrentParam.OverflowOrganizationMethod = 0 then

begin

case HashFunctionChoice.ItemIndex of

0:

range := RangeAverageSquareDistributed[DensityChoice.ItemIndex + 1];

1:

range := RangeSystemDistributed[DensityChoice.ItemIndex + 1];

2:

range := RangeDivisionDistributed[DensityChoice.ItemIndex + 1];

end;

Curoverbucket := overbucket[DensityChoice.ItemIndex + 1];

if ((OldParam.FileVolume <> CurrentParam.FileVolume) or

(@OldParam.HashFunction <> @CurrentParam.HashFunction) or

(OldParam.HashTableSize <> CurrentParam.HashTableSize) or

(OldParam.OverflowOrganizationMethod <>

CurrentParam.OverflowOrganizationMethod)) then

begin

SetLength(HashTable, 0);

HashDistributed(CurrentParam.HashFunction, range,

CurrentParam.HashTableSize, CurrentParam.FileVolume, Curoverbucket,

HashTable, spare.percentageOverflow, spare.mainDensity);

end;

isFound := SearchKeyDistributed(key, HashTable,

CurrentParam.HashFunction, range, CurrentParam.FileVolume,

Curoverbucket, checked, value);

if isFound then Panel1.Caption := 'Найдено: ' + key + ' ' +

IntToStr(value) + '. Просмотрено: ' + IntToStr(checked) +

' записей' else Panel1.Caption := 'Не найдено. Просмотрено: ' +

IntToStr(checked) + ' записей';

end;

if CurrentParam.OverflowOrganizationMethod = 1 then

begin

case HashFunctionChoice.ItemIndex of 0:

range := RangeAverageSquarePeterson;

1:

range := RangeSystemPeterson;

2:

range := RangeDivisionPeterson;

end;

if ((OldParam.FileVolume <> CurrentParam.FileVolume) or

(@OldParam.HashFunction <> @CurrentParam.HashFunction) or

(OldParam.HashTableSize <> CurrentParam.HashTableSize) or

(OldParam.OverflowOrganizationMethod <>

CurrentParam.OverflowOrganizationMethod)) then

begin SetLength(HashTable, 0);

HashPeterson(CurrentParam.HashFunction, range, CurrentParam.HashTableSize,

CurrentParam.FileVolume, HashTable, spare.percentageOverflow,

spare.mainDensity);

end;

isFound := SearchKeyPeterson(key, HashTable, CurrentParam.HashFunction,

range, CurrentParam.FileVolume, checked, value);

if isFound then

Panel1.Caption := 'Найдено: ' + key + ' ' + IntToStr(value) +

'. Просмотрено: ' + IntToStr(checked) + ' записей'

else

Panel1.Caption := 'Не найдено. Просмотрено: ' + IntToStr(checked) +

' записей';

end;

OldParam:=CurrentParam;

end;

end;

procedure TMainForm.GoodSearchPanelClick(Sender: TObject);

var

i: byte;

begin

ActiveMethod := GoodSearchPanelClick;

ChartTitleString := 'Среднее число просмотренных записей при удачном поиске';

SetDefault(GoodSearchPanel);

Self.Memo1.Lines.LoadFromFile('SearchInfo.txt', TEncoding.UTF8);

for i := 1 to 10 do

begin

Self.Chart1.SeriesList[0].Add(ChartArray[CurrentChoice].UnterChart[i]

.Distributed.averageSquare.checkedAverageFound,

(IntToStr(ChartArray[CurrentChoice].UnterChart[i].HashTableSize) + 'п. /'

+ IntToStr(round(PotentialVolume / ChartArray[CurrentChoice].UnterChart[i]

.HashTableSize)) + 'з.'));

Self.Chart1.SeriesList[1].Add(ChartArray[CurrentChoice].UnterChart[i]

.Peterson.averageSquare.checkedAverageFound,

(IntToStr(ChartArray[CurrentChoice].UnterChart[i].HashTableSize) + 'п. /'

+ IntToStr(round(PotentialVolume / ChartArray[CurrentChoice].UnterChart[i]

.HashTableSize)) + 'з.'));

Self.Chart1.SeriesList[2].Add(ChartArray[CurrentChoice].UnterChart[i]

.Distributed.System.checkedAverageFound,

(IntToStr(ChartArray[CurrentChoice].UnterChart[i].HashTableSize) + 'п. /'

+ IntToStr(round(PotentialVolume / ChartArray[CurrentChoice].UnterChart[i]

.HashTableSize)) + 'з.'));

Self.Chart1.SeriesList[3].Add(ChartArray[CurrentChoice].UnterChart[i]

.Peterson.System.checkedAverageFound,

(IntToStr(ChartArray[CurrentChoice].UnterChart[i].HashTableSize) + 'п. /'

+ IntToStr(round(PotentialVolume / ChartArray[CurrentChoice].UnterChart[i]

.HashTableSize)) + 'з.'));

Self.Chart1.SeriesList[4].Add(ChartArray[CurrentChoice].UnterChart[i]

.Distributed.Division.checkedAverageFound,

(IntToStr(ChartArray[CurrentChoice].UnterChart[i].HashTableSize) + 'п. /'

+ IntToStr(round(PotentialVolume / ChartArray[CurrentChoice].UnterChart[i]

.HashTableSize)) + 'з.'));

Self.Chart1.SeriesList[5].Add(ChartArray[CurrentChoice].UnterChart[i]

.Peterson.Division.checkedAverageFound,

(IntToStr(ChartArray[CurrentChoice].UnterChart[i].HashTableSize) + 'п. /'

+ IntToStr(round(PotentialVolume / ChartArray[CurrentChoice].UnterChart[i]

.HashTableSize)) + 'з.'));

end;

end;

procedure TMainForm.BadSearchClick(Sender: TObject);

var

i: byte;

begin

ActiveMethod := BadSearchClick;

ChartTitleString :=

'Среднее число просмотренных записей при неудачном поиске';

SetDefault(BadSearchPanel);

Self.Memo1.Lines.LoadFromFile('SearchInfo.txt', TEncoding.UTF8);

for i := 1 to 10 do

begin

Self.Chart1.SeriesList[0].Add(ChartArray[CurrentChoice].UnterChart[i]

.Distributed.averageSquare.checkedAverageNotFound,

(IntToStr(ChartArray[CurrentChoice].UnterChart[i].HashTableSize) + 'п. /'

+ IntToStr(round(PotentialVolume / ChartArray[CurrentChoice].UnterChart[i]

.HashTableSize)) + 'з.'));

Self.Chart1.SeriesList[1].Add(ChartArray[CurrentChoice].UnterChart[i]

.Peterson.averageSquare.checkedAverageNotFound,

(IntToStr(ChartArray[CurrentChoice].UnterChart[i].HashTableSize) + 'п. /'

+ IntToStr(round(PotentialVolume / ChartArray[CurrentChoice].UnterChart[i]

.HashTableSize)) + 'з.'));

Self.Chart1.SeriesList[2].Add(ChartArray[CurrentChoice].UnterChart[i]

.Distributed.System.checkedAverageNotFound,

(IntToStr(ChartArray[CurrentChoice].UnterChart[i].HashTableSize) + 'п. /'

+ IntToStr(round(PotentialVolume / ChartArray[CurrentChoice].UnterChart[i]

.HashTableSize)) + 'з.'));

Self.Chart1.SeriesList[3].Add(ChartArray[CurrentChoice].UnterChart[i]

.Peterson.System.checkedAverageNotFound,

(IntToStr(ChartArray[CurrentChoice].UnterChart[i].HashTableSize) + 'п. /'

+ IntToStr(round(PotentialVolume / ChartArray[CurrentChoice].UnterChart[i]

.HashTableSize)) + 'з.'));

Self.Chart1.SeriesList[4].Add(ChartArray[CurrentChoice].UnterChart[i]

.Distributed.Division.checkedAverageNotFound,

(IntToStr(ChartArray[CurrentChoice].UnterChart[i].HashTableSize) + 'п. /'

+ IntToStr(round(PotentialVolume / ChartArray[CurrentChoice].UnterChart[i]

.HashTableSize)) + 'з.'));

Self.Chart1.SeriesList[5].Add(ChartArray[CurrentChoice].UnterChart[i]

.Peterson.Division.checkedAverageNotFound,

(IntToStr(ChartArray[CurrentChoice].UnterChart[i].HashTableSize) + 'п. /'

+ IntToStr(round(PotentialVolume / ChartArray[CurrentChoice].UnterChart[i]

.HashTableSize)) + 'з.'));

end;

end;

procedure TMainForm.GrafPanelClick(Sender: TObject);

begin

if ActiveMenuPanel <> nil then

begin

ActiveMenuPanel.Color := clHotLight;

ChartPanel.Visible := false;

end

else

begin

InfoPanel.Color := clNavy;

SummaryPanel.Color := clNavy;

SearchTestPanel.Color := clNavy;

TestSearchPanel.Visible := false;

Memo2.Visible := false;

end;

ResultPanel.Visible := true;

GrafPanel.Color := $00A36005;

ActiveMenuPanel := nil;

ROPPanelClick(Sender);

end;

procedure TMainForm.InfoPanelClick(Sender: TObject);

begin

ActiveMethod := nil;

Self.Memo2.Lines.LoadFromFile('info.txt', TEncoding.UTF8);

Memo2.Visible := true;

if ActiveMenuPanel <> nil then

begin

ActiveMenuPanel.Color := clHotLight;

ActiveDensityControl.Color := clHotLight;

ChartPanel.Visible := false;

end

else

begin

SummaryPanel.Color := clNavy;

GrafPanel.Color := clNavy;

ResultPanel.Visible := false;

SearchTestPanel.Color := clNavy;

TestSearchPanel.Visible := false;

end;

InfoPanel.Color := $00A36005;

ActiveMenuPanel := nil;

end;

procedure TMainForm.SummaryPanelClick(Sender: TObject);

begin

ActiveMethod := nil;

Memo2.Visible := true;

Self.Memo2.Lines.LoadFromFile('resume.txt', TEncoding.UTF8);

if ActiveMenuPanel <> nil then

begin

ActiveMenuPanel.Color := clHotLight;

ActiveDensityControl.Color := clHotLight;

ChartPanel.Visible := false;

end

else

begin

InfoPanel.Color := clNavy;

GrafPanel.Color := clNavy;

SearchTestPanel.Color := clNavy;

TestSearchPanel.Visible := false;

ResultPanel.Visible := false;

end;

SummaryPanel.Color := $00A36005;

ActiveDensityControl.Color := clHotLight;

ActiveMenuPanel := nil;

end;

procedure TMainForm.SearchTestPanelClick(Sender: TObject);

begin

TestSearchPanel.Visible := true;

if ActiveMenuPanel <> nil then

begin

ActiveMenuPanel.Color := clHotLight;

ActiveDensityControl.Color := clHotLight;

Memo2.Visible := false;

ChartPanel.Visible := false;

end

else

begin

InfoPanel.Color := clNavy;

SummaryPanel.Color := clNavy;

GrafPanel.Color := clNavy;

ResultPanel.Visible := false;

end;

SearchTestPanel.Color := $00A36005;

ActiveMenuPanel := nil;

end;

procedure TMainForm.DensPanel1Click(Sender: TObject);

begin

CurrentChoice := 1;

SetDefaultDensityPanel(DensPanel1);

end;

procedure TMainForm.DensPanel2Click(Sender: TObject);

begin

CurrentChoice := 2;

SetDefaultDensityPanel(DensPanel2);

end;

procedure TMainForm.DensPanel3Click(Sender: TObject);

begin

CurrentChoice := 3;

SetDefaultDensityPanel(DensPanel3);

end;

procedure TMainForm.DensPanel4Click(Sender: TObject);

begin

CurrentChoice := 4;

SetDefaultDensityPanel(DensPanel4);

end;

procedure TMainForm.DensPanel5Click(Sender: TObject);

begin

CurrentChoice := 5;

SetDefaultDensityPanel(DensPanel5);

end;

procedure TMainForm.MouseEnter(Sender: TObject);

begin

(Sender as TPanel).BevelKind := bkSoft;

end;

procedure TMainForm.MouseLeave(Sender: TObject);

begin

(Sender as TPanel).BevelKind := bkNone;

end;

procedure TMainForm.OAPanelClick(Sender: TObject);

begin

Chart2.Series[0].Clear;

Series7.AddPie(PetersonPie.averageSquare, 'Метод средних квадратов (ОА)',

$00FF8000);

Series7.AddPie(PetersonPie.SystemConversion,

'Метод преобразования систем счисления (ОА)', $0052CF84);

Series7.AddPie(PetersonPie.Division, 'Метод деления (ОА)', clYellow);

OAPanel.Color := $00FF870F;

ActiveDensityControl.Color := clHotLight;

ActiveDensityControl := OAPanel;

end;

procedure TMainForm.ROPPanelClick(Sender: TObject);

begin

Chart2.Series[0].Clear;

Series7.AddPie(DistributedPie.averageSquare, 'Метод средних квадратов (РОП)',

$00C08000);

Series7.AddPie(DistributedPie.SystemConversion,

'Метод преобразования систем счисления (РОП)', $005BA052);

Series7.AddPie(DistributedPie.Division, 'Метод деления (РОП)', $000AC5D3);

if @ActiveDensityControl <> nil then

ActiveDensityControl.Color := clHotLight;

ROPPanel.Color := $00FF870F;

ActiveDensityControl := ROPPanel;

end;

procedure TMainForm.ToResize(Sender: TObject);

var

x, y, fontres: real;

DensityPanelOffset, MenuPanelOffset: integer;

begin

x := ClientWidth / 1280;

y := ClientHeight / 720;

MenuPanel.Width := round(320 \* x);

MenuPanel.Height := round(720 \* y);

Chart1.Height := round(540 \* y);

Chart2.Height := Chart1.Height;

GraphMenuPanel.Height := round(180 \* y);

PieLabel.Width := round(300 \* x);

PieLabel.Height := round(25 \* y);

PieLabel.Left := round(330 \* x);

PieLabel.Top := round(30 \* y);

ROPPanel.Left := round(110 \* x);

ROPPanel.Height := round(45 \* y);

ROPPanel.Top := round(100 \* y);

ROPPanel.Width := round(320 \* x);

OAPanel.Height := ROPPanel.Height;

OAPanel.Top := ROPPanel.Top;

OAPanel.Width := ROPPanel.Width;

OAPanel.Left := round(510 \* x);

MemoPanel.Top := Chart1.Height;

MemoPanel.Width := ClientWidth - MenuPanel.Width;

MemoPanel.Height := round(120 \* y);

ResultPanel.Left := MenuPanel.Width;

ResultPanel.Width := ClientWidth - MenuPanel.Width;

ResultPanel.Height := ClientHeight;

ChartPanel.Left := MenuPanel.Width;

ChartPanel.Width := ResultPanel.Width;

ChartPanel.Height := ClientHeight;

Memo2.Width := ResultPanel.Width;

Memo2.Height := ClientHeight;

Memo2.Left := MenuPanel.Width;

TestSearchPanel.Left := MenuPanel.Width;

TestSearchPanel.Width := ClientWidth - MenuPanel.Width;

TestSearchPanel.Height := ClientHeight;

fontres := sqrt(x \* y);

MenuPanel.Font.Size := round(fontres \* 12);

Memo1.Font.Size := round(14 \* fontres);

Memo2.Font.Size := round(14 \* fontres);

MenuPanelOffset := round(15 \* y);

DensControlPanel.Font.Size := round(fontres \* 14);

GraphMenuPanel.Font.Size := round(14 \* fontres);

DensityPanel.Left := round(15 \* x);

DensityPanel.Top := round(25 \* y);

DensityPanel.Height := round(65 \* y);

DensityPanel.Width := round(285 \* x);

OverflowPanel.Left := DensityPanel.Left;

OverflowPanel.Height := DensityPanel.Height;

OverflowPanel.Width := DensityPanel.Width;

OverflowPanel.Top := DensityPanel.Top + DensityPanel.Height + MenuPanelOffset;

SearchPanel.Left := DensityPanel.Left;

SearchPanel.Height := DensityPanel.Height;

SearchPanel.Width := DensityPanel.Width;

SearchPanel.Top := OverflowPanel.Top + OverflowPanel.Height + MenuPanelOffset;

GoodSearchPanel.Left := DensityPanel.Left;

GoodSearchPanel.Height := DensityPanel.Height;

GoodSearchPanel.Width := DensityPanel.Width;

GoodSearchPanel.Top := SearchPanel.Top + SearchPanel.Height + MenuPanelOffset;

BadSearchPanel.Left := DensityPanel.Left;

BadSearchPanel.Height := DensityPanel.Height;

BadSearchPanel.Width := DensityPanel.Width;

BadSearchPanel.Top := GoodSearchPanel.Top + GoodSearchPanel.Height + MenuPanelOffset;

GrafPanel.Left := DensityPanel.Left;

GrafPanel.Height := round(50 \* y);

GrafPanel.Width := DensityPanel.Width;

GrafPanel.Top := BadSearchPanel.Top + BadSearchPanel.Height + MenuPanelOffset \* 2;

SearchTestPanel.Left := DensityPanel.Left;

SearchTestPanel.Height := GrafPanel.Height;

SearchTestPanel.Width := DensityPanel.Width;

SearchTestPanel.Top := GrafPanel.Top + GrafPanel.Height + MenuPanelOffset;

InfoPanel.Left := DensityPanel.Left;

InfoPanel.Height := GrafPanel.Height;

InfoPanel.Width := DensityPanel.Width;

InfoPanel.Top := SearchTestPanel.Top + SearchTestPanel.Height + MenuPanelOffset;

SummaryPanel.Left := DensityPanel.Left;

SummaryPanel.Height := InfoPanel.Height;

SummaryPanel.Width := DensityPanel.Width;

SummaryPanel.Top := InfoPanel.Top + InfoPanel.Height + MenuPanelOffset;

DensControlPanel.Top := MemoPanel.Top + MemoPanel.Height;

DensControlPanel.Width := ClientWidth - DensControlPanel.Left;

DensControlPanel.Height := ClientHeight - DensControlPanel.Top;

DensityPanelOffset := round(12 \* x);

Label1.Left := DensityPanelOffset;

Label1.Top := round(20 \* y);

Label1.Width := round(200 \* x);

Label1.Height := round(25 \* y);

DensPanel1.Left := round(220 \* x);

DensPanel1.Width := round(130 \* x);

DensPanel1.Height := round(40 \* y);

DensPanel1.Top := round(10 \* y);

DensPanel2.Left := DensPanel1.Left + DensPanel1.Width + DensityPanelOffset;

DensPanel2.Width := DensPanel1.Width;

DensPanel2.Height := DensPanel1.Height;

DensPanel2.Top := DensPanel1.Top;

DensPanel3.Left := DensPanel2.Left + DensPanel2.Width + DensityPanelOffset;

DensPanel3.Width := DensPanel1.Width;

DensPanel3.Height := DensPanel1.Height;

DensPanel3.Top := DensPanel1.Top;

DensPanel4.Left := DensPanel3.Left + DensPanel3.Width + DensityPanelOffset;

DensPanel4.Width := DensPanel1.Width;

DensPanel4.Height := DensPanel1.Height;

DensPanel4.Top := DensPanel1.Top;

DensPanel5.Left := DensPanel4.Left + DensPanel4.Width + DensityPanelOffset;

DensPanel5.Width := DensPanel1.Width;

DensPanel5.Height := DensPanel1.Height;

DensPanel5.Top := DensPanel1.Top;

TitleLabel.Top:=round(30\*y);

TitleLabel.Left:=Round(350\*x);

OverflowOrganizationMethodChoice.Left:=round(30\*x);

OverflowOrganizationMethodChoice.Top:=round(120\*y);

OverflowOrganizationMethodChoice.Width:=round(350\*x);

OverflowOrganizationMethodChoice.Top:=round(120\*y);

HashFunctionChoice.Left:= OverflowOrganizationMethodChoice.Left;

HashFunctionChoice.Top:= round(270\*y);

HashFunctionChoice.Width:= OverflowOrganizationMethodChoice.Width;

HashFunctionChoice.Height:=round(140\*y);

DensityChoice.Left:=round(420\*x);

DensityChoice.Top:=round(120\*y);

DensityChoice.Width:=round(180\*x);

DensityChoice.Height:=round(290\*y);

BucketSizeChoice.Left:=Round(640\*x);

BucketSizeChoice.Top:=DensityChoice.Top;

BucketSizeChoice.Width:=DensityChoice.Width;

BucketSizeChoice.Height:=DensityChoice.Height;

Label3.Left:= HashFunctionChoice.Left;

Label3.Top:=round(435\*y);

KeyEdit.Left:=round(95\*x);

KeyEdit.Top:= round(430\*y);

KeyEdit.Width:=Round(150\*x);

KeyEdit.Height:=round(30\*x);

GetSearchResultPanel.Left:= round(270\*x);

GetSearchResultPanel.Top:= KeyEdit.Top;

GetSearchResultPanel.Width:= round(90\*x);

GetSearchResultPanel.Height:=KeyEdit.Height;

Panel1.Left:= Label3.Left;

Panel1.Top:=round(480\*y);

Panel1.Width:=Round(500\*x);

Panel1.Height:=round(40\*y);

TestSearchPanel.Font.Size:=Round(fontres\*14);

KeyEdit.Font.Size:=Round(fontres\*18);

end;

procedure TMainForm.LabelMouseEnter(Sender: TObject);

begin

((Sender as TLabel).Parent as TPanel).BevelKind := bkSoft;

end;

procedure TMainForm.Chart1Click(Sender: TObject);

begin

if Self.Chart1.Pages.MaxPointsPerPage = 5 then

begin

if Self.Chart1.Page = 1 then

Self.Chart1.Pages.NextPage

else

Self.Chart1.Pages.PreviousPage;

end;

end;

procedure TMainForm.Chart1MouseWheelDown(Sender: TObject; Shift: TShiftState;

MousePos: TPoint; var Handled: Boolean);

begin

Self.Chart1.Pages.MaxPointsPerPage := 5;

end;

procedure TMainForm.Chart1MouseWheelUp(Sender: TObject; Shift: TShiftState;

MousePos: TPoint; var Handled: Boolean);

begin

Self.Chart1.Pages.MaxPointsPerPage := 10;

end;

procedure TMainForm.SeriesMouseEnter(Sender: TObject);

begin

(Sender as TBarSeries).Marks.Visible := true;

end;

procedure TMainForm.SeriesMouseLeft(Sender: TObject);

begin

(Sender as TBarSeries).Marks.Visible := false;

end;

end.

ВЕДОМОСТЬ

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Обозначение | | | | Наименование | | | | Дополнительные сведения | | | |
|  | | | | Текстовые документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| БГУИР КР 1–40 01 01 204 ПЗ | | | | Пояснительная записка | | | | 85 с. | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | | Графические документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| ГУИР 951002 204 СП | | | | Схема программы «Анализ хранения и обработки больших объемов информации методом хеширования» | | | | Формат А1 | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  |  |  |  |  | БГУИР КР 1-40 01 01 204 Д1 | | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| Изм. | Л. | № докум. | Подп. | Дата | Анализ хранения и обработки больших объемов информации методом хеширования  Ведомость курсовой  работы |  | | | | Лист | Листов |
| Разраб. | | Будович И.В.. |  | 09.06.20 | Т |  | |  | 85 | 85 |
| Пров. | | Фадеева Е.Е. |  | 09.06.20 | Кафедра ПОИТ  гр. 951002 | | | | | |
|  | |  |  |  |
|  | |  |  |  |
|  | |  |  |  |