**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«Сибирский государственный университет науки и технологий  
имени академика М.Ф. Решетнева»**

Институт информатики и телекоммуникаций

Кафедра информатики и вычислительной техники

**КУРСОВАЯ РАБОТА**

Методы поиска во внешней памяти

тема работы

Руководитель \_\_\_\_\_\_\_\_\_\_\_ Тынченко В. В.

подпись, дата инициалы, фамилия

Обучающийся БПИ22-02, 221219037 \_\_\_\_\_\_\_\_\_\_ Соколов М. А.

номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2023

Институт информатики и телекоммуникаций

Кафедра информатики и вычислительной техники

|  |
| --- |
| **ЗАДАНИЕ**  на курсовую работу по дисциплине «Алгоритмы и структуры данных»  студенту |
| Группа БПИ22-02 Форма обучения очная |  |
| 1. Тема работы: Методы поиска во внешней памяти |
| 2. Срок сдачи студентом работы: 30.12.23 |
| 3. Перечень вопросов, подлежащих разработке при написании теоретической части:  Изучить основные методы поиска во внешней памяти.  Выяснить, какие методы являются наиболее подходящими. |
| 4. Перечень вопросов, подлежащих разработке при написании практической части (либо указать номер варианта задания)  Реализовать алгоритмы поиска во внешней памяти.  Провести исследование эффективности методов поиска во внешней памяти. |
| 5. Дата выдачи задания: 08.09.23 |
| Руководитель: Тынченко В.В. |
| Подпись |
|  |
| Задание принял к исполнению (дата): 08.09.23 |
| (подпись студента) |

Содержание

Введение 4

1 Методы поиска во внешней памяти 5

1.1 Последовательный поиск 5

1.2 Двоичный поиск 5

1.3 Хэширование 7

1.4 Одноколоночные индексы 7

1.5 Выводы 8

2 Описание программы 9

2.1 Общая характеристика программы 9

2.2 Структура программы и данных 9

2.3 Интерфейс программы 10

2.4 Тестирование программы 12

2.5 Выводы 14

3 Использование программы 15

3.1 Порядок эксплуатации программы 15

3.2 Результаты проведения вычислительных экспериментов 15

3.3 Выводы 16

Заключение 18

Список использованных источников 19

Приложение А Листинг программы 20

Приложение Б Проверка на оригинальность 25

Введение

В современном технологически развитом обществе огромные объемы информации требуют эффективной обработки. Иногда такой поток данных настолько обширен, что легко запутаться и утратить важные сведения. Для предотвращения этого и улучшения процесса обработки информации используются разнообразные методы поиска данных.

Алгоритмы поиска являются важным инструментом в программировании. Правильный выбор алгоритма способен значительно ускорить процесс поиска. Одной из основных задач программирования является разработка именно эффективных алгоритмов.

В данной курсовой работе рассматриваются методы поиска во внешней памяти, анализируются различные алгоритмы, используемые для эффективного поиска данных в условиях хранения этих на внешних устройствах, с акцентом на повышение производительности и оптимизацию времени выполнения операций поиска.

Целью курсовой является изучение особенностей и принципов алгоритмов поиска во внешней памяти и исследование их эффективности.

Задачи курсовой работы:

* изучить теоретический материал по данной теме курсовой работы;
* исследовать алгоритмы поиска во внешней памяти;
* разработать программу для работы с алгоритмами поиска;
* провести тестирование программы.

1. Методы поиска во внешней памяти

Поиск во внешней памяти, также известный как внешний поиск или внешняя сортировка, представляет собой процесс поиска данных, хранящихся на внешних устройствах, таких как жесткие диски, съемные диски или сетевые устройства.

Внешняя память, в отличие от оперативной памяти, имеет гораздо большую емкость, но и более медленную скорость доступа. Поиск во внешней памяти обычно требует выполнения большого количества операций чтения и записи данных на диск, что делает его более медленным по сравнению с поиском в оперативной памяти.

* 1. Последовательный поиск

Метод последовательного поиска во внешней памяти также известен как метод простого перебора или метод линейного поиска. Он заключается в последовательном просмотре элементов внешней памяти сравнением каждого элемента с искомым значением.

Основные шаги метода последовательного поиска во внешней памяти:

* чтение первого элемента из внешней памяти;
* сравнение прочитанного элемента с искомым значением;
* если элемент совпадает с искомым значением, то поиск успешно завершается, и возвращается индекс этого элемента;
* если элемент не совпадает с искомым значением, то переход к следующему элементу, повторение шагов 2 и 3;
* если все элементы внешней памяти были просмотрены, и не было найдено совпадений, то поиск неуспешен, и возвращается специальное значение (например, -1 или NULL).

Преимущества метода последовательного поиска во внешней памяти:

* простота реализации;
* не требует предварительной сортировки элементов;
* работает для любых типов данных.

Недостатки метода последовательного поиска во внешней памяти:

* эффективность зависит от объема данных и скорости доступа к внешней памяти;
* при большом количестве данных и медленном доступе к внешней памяти время поиска может быть значительным;
* максимальная асимптотическая сложность метода составляет O(n), где n - количество элементов во внешней памяти.

Итак, метод последовательного поиска во внешней памяти прост и универсален, но может быть неэффективным для больших объемов данных.

* 1. Двоичный поиск

Метод двоичного поиска во внешней памяти, также известный как внешне сортировочный метод двоичного поиска, используется для поиска элемента в больших объемах данных, которые не могут поместиться в оперативную память.

Основная идея метода двоичного поиска во внешней памяти заключается в том, чтобы разделить данные на блоки, которые могут быть загружены в оперативную память одновременно. Затем каждый блок сортируется внутри себя.

При поиске элемента во внешней памяти, сначала ищется блок, в котором может находиться искомый элемент. Для этого используется двоичный поиск по блокам. Затем найденный блок загружается в память, и в нем выполняется обычный двоичный поиск для поиска конкретного элемента.

Однако, чтобы использовать метод двоичного поиска во внешней памяти, требуется эффективная организация данных и доступа к ним. Например, данные могут быть разделены на файлы фиксированного размера, которые хранятся на диске. Также может использоваться индексная структура для быстрого поиска блока, в котором находится искомый элемент.

Важным аспектом метода двоичного поиска во внешней памяти является оптимизация операций чтения и записи данных. Часто используются буферы чтения и записи для уменьшения количества операций ввода-вывода. Также может быть использован алгоритм внешней сортировки для эффективной сортировки блоков данных.

Этапы двоичного поиска во внешней памяти:

* загрузка данных. Изначально, необходимо загрузить часть данных из внешней памяти в оперативную память для выполнения поиска. Обычно, размер этой загружаемой части данных определяется доступной оперативной памятью;
* определение границ. Затем, алгоритм определяет начальную и конечную границы интервала, в котором будет выполняться поиск. Обычно, начальной границей является первый элемент загруженных данных, а конечной - последний элемент;
* поиск середины. Затем, алгоритм определяет средний элемент в текущем интервале поиска, путем деления суммы начальной и конечной границ на два;
* сравнение. Алгоритм сравнивает искомый элемент со средним элементом. Если они равны, то поиск завершается, так как искомый элемент найден;
* обновление интервала. Если искомый элемент больше среднего элемента, то начальная граница интервала обновляется на следующий элемент после среднего, иначе - конечная граница интервала обновляется на предыдущий элемент перед средним;
* повторение шагов. Алгоритм повторяет шаги 3-5, пока не будет найден искомый элемент или интервал поиска не будет сокращен до одного элемента;
* загрузка новых данных. Если искомый элемент не найден в текущем интервале, то алгоритм загружает новую часть данных из внешней памяти и повторяет шаги 2-6;
* завершение поиска. Поиск завершается, когда искомый элемент найден или интервал поиска сокращен до одного элемента и элемент не равен искомому значению.

В целом, метод двоичного поиска во внешней памяти является сложным и требует детальной организации данных и доступа к ним. Он позволяет эффективно выполнять поиск элемента в больших объемах данных, которые не могут быть полностью загружены в оперативную память.

* 1. Хэширование

Основным принципом метода хэширования для поиска во внешней памяти является использование функции хэширования, которая преобразует ключи поиска в адреса на внешнем устройстве, где данные хранятся.

Процесс работы метода хэширования для поиска во внешней памяти может быть описан следующим образом:

* создается хэш-таблица, которая служит для хранения адресов элементов на внешнем устройстве. Хэш-таблица может быть представлена, например, массивом со списками в каждой ячейке;
* функция хэширования применяется к ключу поиска. Эта функция преобразует ключ в число, которое указывает на определенную ячейку в хэш-таблице;
* если в ячейке хэш-таблицы уже есть элементы, то используется метод разрешения коллизий для поиска свободной ячейки. Возможные методы разрешения коллизий включают метод цепочек или метод открытой адресации;
* после нахождения свободной ячейки, адрес текущего элемента записывается в эту ячейку;
* при поиске элемента с определенным ключом, функция хэширования применяется к ключу, чтобы получить адрес на внешнем устройстве;
* затем, используя полученный адрес, программа обращается к внешнему устройству, чтобы получить доступ к данным.

Преимущества метода хэширования для поиска во внешней памяти включают быстрый доступ к данным и эффективное использование памяти. Однако недостатком этого метода является возможность коллизий, которые могут привести к дополнительным затратам на поиск свободной ячейки.

* 1. Одноколоночные индексы

Принцип работы одноколоночных индексов заключается в создании структуры данных, которая обеспечивает быстрый доступ к информации в определенном столбце таблицы базы данных. Индекс содержит уникальные значения из этого столбца. Это позволяет системе эффективно выполнять операции поиска.

Основные шаги создания и использования одноколоночных индексов:

* разделение данных. Исходные данные разбиваются на стобцы, чтобы упростить процесс поиска и обработки;
* создание индекса. Для каждого блока данных создается запись в индексной таблице, содержащая ключевой атрибут и ссылку на местоположение блока. Ключевой атрибут может быть любым полем, которое можно использовать для идентификации и поиска данных, например, идентификатором или именем. Индексная таблица содержит сортированный список записей с ключами;
* упорядочение и построение индекса. Записи в индексной таблице обычно упорядочены по ключевому атрибуту. Это позволяет проводить эффективный поиск с использованием алгоритмов, таких как двоичный поиск или интерполяционный поиск;
* оптимизация. Для повышения производительности индексных таблиц могут быть использованы различные оптимизации, такие как кластеризация записей с близкими значениями ключей или использование дополнительной структуры индексирования, такой как хэш-таблица.

Для выполнения поиска во внешней памяти с использованием индексных таблиц необходимо выполнить следующие шаги:

* чтение индекса. Загрузите файл в оперативную память. Это позволяет быстро получить доступ к записям индекса и определить местоположение нужных данных;
* поиск: Используйте алгоритм поиска, чтобы найти нужную запись в индексной таблице. Ключевой атрибут, соответствующий искомым данным, поможет найти соответствующую запись;
* переход к данным. После нахождения нужной записи в индексной таблице имеется ссылка на местоположение блока данных, на которые ссылается запись. Перейдите к этому местоположению на внешнем носителе и загрузите данные в оперативную память для дальнейшей обработки или отображения.

Преимущество метода одноколоночных индексов для поиска во внешней памяти заключается в том, что он позволяет быстро находить нужные данные на основе их ключевого атрибута, даже если объем данных очень большой. Однако создание и поддержка одноколоночных индексов может потребовать дополнительных ресурсов и времени.

* 1. Выводы

Были рассмотрены методы поиска во внешней памяти и их принципы. Также были выявлены основные шаги создания различных алгоритмов поиска. Стало понятно, что алгоритмы поиска во внешней памяти различны и имеют подходящие методы реализации и степень эффективности для самых разных задач.

1. Описание программы
   1. Общая характеристика программы

Для изучения темы курсовой работы была разработана и создана программа, способная выполнять поиск элемента во внешней памяти. Программа реализована на языке программирования C++. Она может выполнять поиск элемента во внешней памяти одним из четырех доступных способов: последовательный поиск, двоичный поиск, поиск с помощью индексирования, поиск с помощью хеширования. В конце обработки данных программа выводит размер обработанной последовательности и время обработки в отдельный файл.

Код программы написан в *Microsoft Visual Studio* 2022 версии 17.4.3, там же была выполнена сама программа. При тестировании практической части работы особых задержек замечено не было, поскольку сами тесты выполнялись на неслабой системе. Но при эксплуатации на некоторых системах в нашем университете могут возникнуть неприятные ситуации в виде долгой обработки больших данных. Чтобы избежать этого, используемая система должна соответствовать минимальным требованиям представленным в таблице 1.

Таблица 1 – Системные требования

|  |  |
| --- | --- |
| Операционная система | *Windows* 8 и выше |
| Процессор | 2 ядра, частота 2ГГц и более |
| Оперативная память | 2 ГБ и более |
| Свободное дисковое пространство | Не менее 200 Мб |

* 1. Структура программы и данных

Для решения поставленной задачи в рамках курсовой работы был выбран процедурный подход.

В ходе разработки программы были реализованы 4 алгоритма поиска во внешней памяти:

* *sequentialSearch()* –реализация алгоритма последовательного поиска во внешней памяти;
* *binarySearch()* – реализация алгоритма двоичного поиска во внешней памяти;
* *indexedSearch()* – реализация алгоритма поиска во внешней памяти с помощью индексирования;
* *search()* –реализация алгоритма поиска во внешней памяти с помощью хеширования.

Функции *sequentialSearch()* и *binaruSearch()* принимают данные типа *int* и ничего не возвращают. Функция *indexedSearch()* принимает данные типа *string* и возвращает либо найденный номер строки, если значение найдено, либо значение -1, если значение не найдено. Метод *search()* принадлежит классу *hTab*, реализующему алгоритмы работы с хеш-таблицей. В данном классе имеются метод *HashFunction()*, метод *insert()*, позволяющий вставлять новые элементы в хеш-таблицу, метод *search()*, позволяющий найти элемент по значению в хеш-таблице и возвращающий значение типа *bool*. Также в программе есть функция *consistent()*, принимающая значение типа *int*, которая заполняет файл последовательными значениями. Был выбран именно такой набор исходных данных, так как все методы поиска либо нуждаются в предварительной сортировке последовательности данных, либо же сами выполняют эту сортировку. Такой способ эффективен в плане затраты ресурсов на реализацию и выполнение программы.

Программа создает несколько последовательностей, характеристики которых заданы пользователем, и выводит результат их обработки в отдельный файл. В роли значения для поиска используется число равное 3\4 от размера последовательности.

Программа состоит из файлов *main.cpp* и *search.h*. Файл *search.h* хранит в себе реализацию класса *hTab*, а файл *main.cpp* – реализацию остальных функций и меню взаимодействия. Само меню реализовано с помощью оператора *switch*.

* 1. Интерфейс программы

Интерфейс помогает пользователю вести диалог с программой, не имея информации об алгоритмах, выполнение которых следует за выбором того или иного пункта меню. Главная задача пользовательского меню состоит в том, чтобы пользователь знал, куда нажимать и что выбирать.

Программа работает в автоматическом режиме, то есть пользователю достаточно ввести размер первой последовательности, последней последовательности и шаг размера последовательности, затем выбрать способ поиска. И программа сама выведет результат обработки нескольких последовательностей в отдельный файл.

Программа выполнена в виде консольного приложения. В результате запуска в окне высвечивается меню и несколько его пунктов, названия которых описывают их назначение. Первый пункт можно выбрать, введя цифру «1». В данном пункте пользователю предлагается задать размер первой последовательности, последней последовательности и шаг размера последовательности.

Во втором, третьем, четвертом, пятом пунктах пользователь может выбрать способ поиска во внешней памяти. В зависимости от выбора будет использована та или иная функция для поиска. С помощью функции *clock()* и переменных *s\_time* и *e\_time* определяется время поиска: перед вызовом функции поиска значение *clock()* передается в переменную *s\_time*, а после вызова – в переменную *e\_time*. В файл выводится значение их разницы.

В последнем пункте, который выбирается вводом цифры «0», значение переменной *flag* равняется *false*, из-за чего цикл *while*, внутри которого находится оператор *switch*, прерывается и функция *main()*, как и программа, завершает свою работу. Схема интерфейса представлена на рисунке 1.

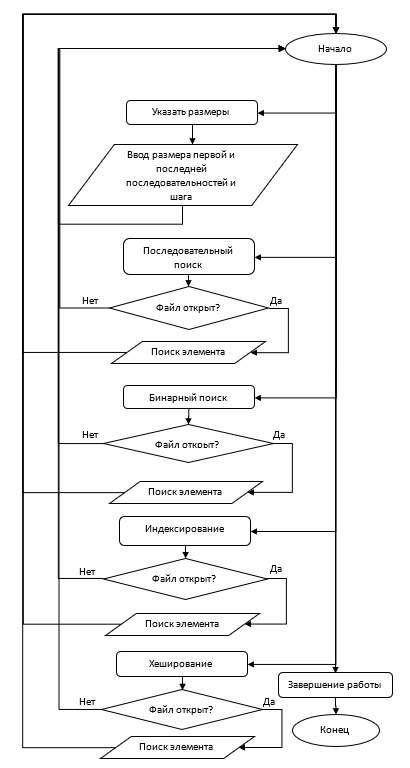


Рисунок 1 – Схема работы меню

* 1. Тестирование программы

Тестирование выполнялось методом черного ящика, то есть субъект тестирования не знает информации о работе программы и ее устройстве и задумывается только о том, что делает программа, а не как она это делает. Таким образом, можно изучить поведение реакции в зависимости от входных данных и других внешних воздействий.

При запуске программы в консоли отображается главное меню, соответствующее рисунку 2. Пользователю предоставляется возможность задать размер первой последовательности, размер последней последовательности и шаг размера последовательности, выбрать один из методов поиска во внешней памяти или завершить работу программы. Если пользователь выберет один из методов поиска, проигнорировав первый пункт, то сам поиск будет выполнен в рамках одной последовательности размером 100 элементов.

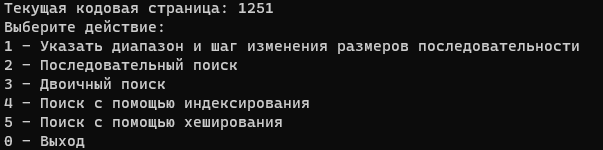


Рисунок 2 – Главное меню

При выборе пункта 1 на экран выводится текст, информирующий о необходимости ввести длину первой последовательности, длину последней последовательности и шаг. Введем значения, пусть они будут равны 10, 11 и 1 соответственно. Действие продемонстрированно на рисунке 3.



Рисунок 3 – Ввод размера первой и последней последовательностей и шага

Затем снова высвечивается меню, в котором мы уже выбираем способ поиска. Выберем пункт 2, последовательный поиск. Это показано на рисунке 4.

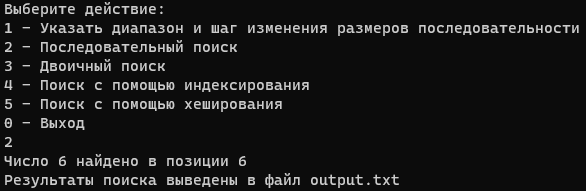


Рисунок 4 – Выбор метода поиска значения в последовательности

В результате нашего выбора создается файл *input.txt*, который заполняется упорядоченной последовательностью чисел, выполняется алгоритм поиска, и на экран выводится число, которое мы ищем, и его позиция в последовательности. В следующей строке программа предупреждает нас, что результаты поиска, то есть размер последовательности и время, были выведены в файл *output.txt*. Содержимое файлов *input.txt* и *output.txt* можно увидеть на рисунках 5 – 6 соответственно.

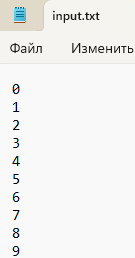


Рисунок 5 – Содержимое файла input.txt

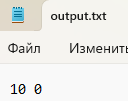


Рисунок 6 – Содержимое файла output.txt

После выполнения поиска выбираем последний пункт в главном меню в консоли, из чего следует завершение работы программы, показанное на рисунке 7.

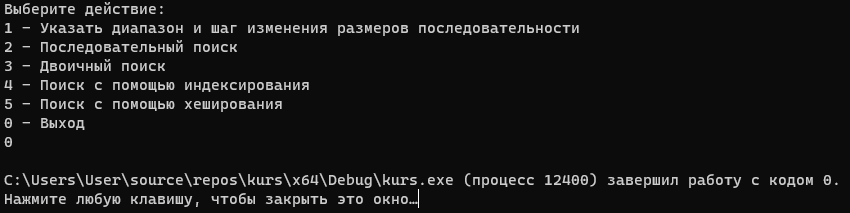


Рисунок 7 – Завершение работы программы

* 1. Выводы

Во втором разделе курсовой работы была рассмотрена программа, а именно ее характеристики, структура и интерфейс. Было проведено тестирование, показавшее работоспособность программы. Таким образом, мы имеем полное представление о программе.

1. Использование программы
   1. Порядок эксплуатации программы

Порядок эксплуатации программы пользователем выглядит следующим образом:

1. задание размера первой последовательности;
2. задание размера последней последовательности;
3. задание шага размера последовательности;
4. выбор метода поиска элемента;
5. завершение работы программы.

Пользователь имеет возможность изучить способы поиска во внешней памяти, проведя расчеты с любыми размерами данных. Удобство эксплуатации заключается в том, что результат поиска значения выводится на экран, а размер последовательности, в которой проводился поиск, и время поиска выводятся в отдельный файл *output.txt*.

* 1. Результаты проведения вычислительных экспериментов

Выбраны размеры последовательностей: 110000, 120000, 130000, 140000, 150000, 160000, 170000, 180000, 190000, 200000. Диапазон значений для каждой последовательности: от 0 до значения размера последовательности. В качестве способа формирования использована упорядоченная последовательность элементов.

Проанализировав рисунок 8, на котором изображен график с полученными значениями времени для каждого метода поиска, можно утверждать, что самым медленным способом поиска элемента во внешней памяти является поиск с помощью хеширования, максимальное время которого достигает примерно 65 секунд при 200000 элементов. Остальные методы поиска сильно вырвались вперед по времени выполнения задачи.

Чтобы изучить другие методы поиска, был построен еще один график, изображенный на рисунке 9. Здесь можно наблюдать, что в рамках этих трех способов самым долгим является поиск с помощью индексирования. Максимальное время, затраченное на выполнение задачи данным методом, составляет чуть меньше 1 секунды при 200000 элементов. Это в разы дольше, чем последовательный поиск, время выполнения которого при максимальном размере последовательности составляет меньше трети секунды. Самым быстрым алгоритмом поиска элемента во внешней памяти является метод бинарного поиска. Среднее время выполнения задачи данным способом составило 1 миллисекунда.

Также замечено, что при увеличении значения размера последовательности, увеличивается и разница во времени поиска между алгоритмами.

Рисунок 8 – Время поиска при упорядоченной последовательности, все методы

Рисунок 9 – Время поиска при упорядоченной последовательности, без хеширования

* 1. Выводы

Из полученных результатов экспериментов можно сделать вывод, что самым эффективным способом поиска при нормальных условиях является бинарный поиск. Если бы последовательность надо было предварительно сортировать, то поиск с помощью данного метода занял бы больше времени. Вторым по эффективности является последовательный поиск. Однако его применение может быть настолько же эффективным, если размер последовательности небольшой и нужный элемент находится не в самом конце файла. Следующим по эффективности методом поиска является поиск с помощью индексирования. Его преимущество заключается в том, что он не нуждается в предварительной сортировке и что при этом он выполняет поиск с небольшим отставанием от предыдущих методов. Самым неэффективным способом поиска при данных условиях оказался поиск с помощью хеширования. Однако при других условиях он имеет хорошее преимущество. Данный метод будет особо эффективен при поиске элемента в неупорядоченной последовательности больших размеров.

Заключение

В результате выполнения курсовой работы была разработана рабочая программа, предназначенная для получения и накопления данных об алгоритмах поиска элемента во внешней памяти и изучения эффективности данных методов. Было выяснено, какие методы поиска самые эффективные и для каких задач они подходят.

Чтобы достичь поставленную цель, были выполнены следующие задачи:

* изучен теоретический материал по данной теме курсовой работы;
* исследованы алгоритмы поиска во внешней памяти;
* разработана программа для работы с алгоритмами поиска;
* проведено тестирование программы.

Написанная программа может использоваться для узконаправленных задач по поиску элемента во внешней памяти. Однако разработанную программу можно усовершенствовать, добавив возможность сортировки данных для некоторых способов поиска и другие алгоритмы поиска.

Мы справились со всеми поставленными задачами, разработали программу для подробного изучения темы и провели исследование. Таким образом, работа может считаться завершенной.

Список использованных источников

1. CITFORUM.RU : сайт. – Методы поиска и сортировки – 2007. – URL: http://citforum.ru/programming/theory/sorting/sorting2.shtml (дата обращения: 07.10.2023). – Текст: электронный.
2. NEERC.IFMO.RU : сайт. – 2022. – URL: https://neerc.ifmo.ru/wiki/index.php?title=Алгоритмы\_во\_внешней\_памяти.\_Базовые\_конструкции (дата обращения: 07.10.2023). – Текст: электронный.
3. UCHET-JKH.RU : сайт. – 2023. – URL: https://uchet-jkh.ru/i/process-xesirovaniya-urovnei-v-pamyati-osnovy-i-principy/ (дата обращения: 08.10.2023). – Текст: электронный.
4. HARB.COM : сайт. – 2015. – URL: https://habr.com/ru/articles/247373/ (дата обращения: 09.10.2023). – Текст: электронный.
5. https://en.wikipedia.org/wiki/Depth-first\_search

Приложение А Листинг программы

Содержимое файла *search.h*:

#pragma once

class hTab

{

private:

const size\_t table\_size; // размер таблицы

vector<string> table; // таблица

public:

hTab(int size) : table\_size(size), table(table\_size, "") {} // конструктор

int HashFunction(const string& key); // хеш-функция

void insert(const string& key); // вставка ключа

bool search(const string& key); // поиск ключа

};

// хеш-функция

int hTab::HashFunction(const std::string& key)

{

int hash = 0;

for (char ch : key)

{

hash += ch;

}

return hash % table\_size;

}

// вставка ключа

void hTab::insert(const string& key)

{

int index = HashFunction(key);

int count = 1;

while (!table[index].empty())

{

index = (index + count \* count) % table\_size;

count++;

}

table[index] = key;

}

// поиск ключа

bool hTab::search(const string& key)

{

int index = HashFunction(key);

int count = 1;

while (!table[index].empty())

{

if (table[index] == key) return true;

index = (index + count \* count) % table\_size;

count++;

}

return false;

}

Содержимое файла *main.cpp*:

#include <iostream>

#include <fstream>

#include <string>

#include <Windows.h>

#include <unordered\_map>

#include <sstream>

using namespace std;

#include "search.h"

// последовательный поиск

void sequentialSearch(const int& value, int& indx)

{

ifstream file("input.txt");

int index = -1;

if (file.is\_open())

{

int tmp;

while (!file.eof())

{

file >> tmp;

index++;

if (tmp == value)

{

file.close();

indx = index;

break;

}

}

file.close();

}

else

cout << "Файл закрыт!\n";

}

// функция для выполнения двоичного поиска во внешней памяти

void binarySearch(const int& value, int& indx, const int size)

{

ifstream file("input.txt");

if (!file.is\_open())

{

cout << "Ошибка открытия файла.\n";

return;

}

// переменные для хранения левой и правой границы поиска

int left = file.tellg();

file.seekg(size);

int right = file.tellg();

// выполняем двоичный поиск

while (left <= right)

{

int mid = left + (right - left) / 2;

file.seekg(mid);

int num = file.tellg();

if (num == value)

{

// элемент найден, возвращаем его индекс

file.close();

indx = mid;

return;

}

else if (num < value)

{

// искомый элемент находится в правой половине

left = mid + 1;

}

else

{

// искомый элемент находится в левой половине

right = mid - 1;

}

}

cout << "Элемент не найден\n";

}

// заполнение файла последовательными значениями

void consistent(const int& size)

{

ofstream file("input.txt");

if (file.is\_open())

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

file << i << endl;

else

cout << "Файл закрыт!\n";

file.close();

}

// поиск индексированием

unordered\_map<string, int> buildIndexFromFile()

{

unordered\_map<string, int> index;

ifstream file("input.txt");

if (!file.is\_open())

{

cerr << "Ошибка открытия файла." << endl;

return index;

}

string line;

int lineNumber = 0;

while (getline(file, line))

{

++lineNumber;

istringstream iss(line);

string word;

while (iss >> word)

{

index[word] = lineNumber;

}

}

file.close();

return index;

}

int indexedSearch(const unordered\_map<string, int>& indx, const string& value)

{

auto it = indx.find(value);

if (it != indx.end())

return it->second; // найденный номер строки

return -1;

}

int main(void)

{

system("chcp 1251");

int left = 100, right = 101, step = 1;

int s\_time, e\_time;

int a;

bool flag = true;

while (flag)

{

cout << "Выберите действие:\n"

<< "1 - Указать диапазон и шаг изменения размеров последовательности\n"

<< "2 - Последовательный поиск\n"

<< "3 - Двоичный поиск\n"

<< "4 - Поиск с помощью индексирования\n"

<< "5 - Поиск с помощью хеширования\n"

<< "0 - Выход\n";

cin >> a; switch (a) {

case 1:

{

cout << "Укажите длину первой последовательности: ";

cin >> left;

cout << "Укажите длину последней последовательности: ";

cin >> right;

cout << "Укажите шаг изменения размера последовательности: ";

cin >> step;

break;

}

case 2:

{

int indx = 0;

ofstream output("output.txt");

for (int i = left; i < right; i += step)

{

consistent(i);

s\_time = clock();

sequentialSearch(i / 4 \* 3, indx);

e\_time = clock();

cout << "Число " << i / 4 \* 3 << " найдено в позиции " << indx << endl;

output << i << "\t" << e\_time - s\_time << endl;

}

output.close();

cout << "Результаты поиска выведены в файл output.txt\n";

break;

}

case 3:

{

int indx = 0;

ofstream output("output.txt");

for (int i = left; i < right; i += step)

{

consistent(i);

s\_time = clock();

binarySearch(i / 4 \* 3, indx, i);

e\_time = clock();

cout << "Число " << i / 4 \* 3 << " найдено в позиции " << indx << endl;

output << i << "\t" << e\_time - s\_time << endl;

}

output.close();

cout << "Результаты поиска выведены в файл output.txt\n";

break;

}

case 4:

{

ofstream output("output.txt");

for (int i = left; i < right; i += step)

{

consistent(i);

s\_time = clock();

// cоздание индекса

unordered\_map<string, int> index = buildIndexFromFile();

int result = indexedSearch(index, to\_string(i / 4 \* 3));

e\_time = clock();

cout << "Число " << i / 4 \* 3 << " найдено в позиции " << result << endl;

output << i << "\t" << e\_time - s\_time << endl;

}

output.close();

cout << "Результаты поиска выведены в файл output.txt\n";

break;

}

case 5:

{

string line;

int indx = 0;

ofstream output("output.txt");

for (int i = left; i < right; i += step)

{

hTab table(i);

consistent(i);

ifstream input("input.txt");

s\_time = clock();

while (getline(input, line))

{

if (table.search(line))

continue;

table.insert(line);

}

bool f = table.search(to\_string(i / 4 \* 3));

e\_time = clock();

input.close();

output << i << "\t" << e\_time - s\_time << endl;

table.~hTab();

}

output.close();

cout << "Результаты поиска выведены в файл output.txt\n";

break;

}

case 0:

{

flag = false;

break;

}

default:

{

cout << "Введенны некорректные данные.\n";

break;

}

}

}

}

Приложение Б Проверка на оригинальность

Проверка проводилась на сайте https://users.antiplagiat.ru.

Результат проверки можно увидеть на рисунке Б.1.

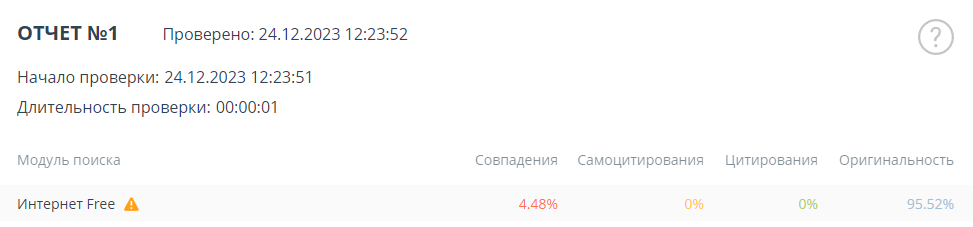


Рисунок Б.1 – Результат проверки на оригинальность