Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Институт информационных технологий

Факультет компьютерных технологий

Кафедра информационных систем и технологий

Дисциплина: Основы алгоритмизации и программирования (ОАиП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

«СРАВНИТЕЛЬНЫЙ АНАЛИЗ МЕТОДОВ ПОИСКА ДАННЫХ В ОТСОРТИРОВАННЫХ ПОСЛЕДОВАТЕЛЬНОСТЯХ»

Студент:

гр. 281078 Войкель В.И.

Руководитель:

ассистент Болтак С.В.

Минск 2023

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Институт информационных технологий

Факультет компьютерных технологий

УТВЕРЖДАЮ

Заведующий кафедрой ИСиТ

––––––––––––––––––––––––

(подпись)

––––––––––––––––– 2023 г.

ЗАДАНИЕ

на курсовую работу

студенту      Войкелю Виктору Иосифовичу

1. Тема работы    «Сравнительный анализ методов поиска данных в отсортированных последовательностях»

2. Срок сдачи студентом законченной работы––15.05.2023 г.–––

3. Исходные данные к работе

Файл с отсортированными массивами разных размеров

4. Содержание пояснительной записки

Введение

1. Моделирование программного средства

2. Проектирование программного средства

3. Оценка работы (тестирование) программного средства и анализ результатов

Выводы

Список используемых источников

Приложение А. Фрагменты программного кода

5. Перечень графического материала

1. "Сравнительный анализ методов поиска данных в отсортированных последовательностях", схема программы, чертеж – формат А3, лист 1.

2. "Последовательный поиск", схема алгоритма, чертеж – формат А3, лист 1.

3. "Бинарный поиск", схема алгоритма, чертеж – формат А3, лист 1.

6. Консультант по курсовой работе Болтак С.В.

7. Дата выдачи задания – 18.02.2023 г.

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

|  |  |  |  |
| --- | --- | --- | --- |
| № | Содержание работ | Срок  выполнения | % от общего объёма работы |
| 1 | Раздел 1 | 04.03.2023 | 15 % |
| 2 | Раздел 2 | 01.04.2023 | 30 % |
| 3 | Раздел 3 | 15.04.2023 | 60 % |
| 4 | Раздел 4 | 06.05.2023 | 90 % |
| 5 | Оформление пояснительной записки и графического материала | 15.05.2023 | 100 % |
| 6 | Защита курсовой работы |  |  |

РУКОВОДИТЕЛЬ–––––– С.В.Болтак

(подпись)

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

(дата и подпись студента)

**АННОТАЦИЯ**

Цель данной курсовой работы – провести сравнительный анализ методов поиска данных в отсортированных последовательностях. Предметом исследования являются алгоритмы линейного поиска, бинарного поиска.

Для экспериментального исследования была разработана программа, которая реализует эти алгоритмы, а также проводит сравнение эффективности этих алгоритмов на заранее подготовленных отсортированных последовательностях данных.

Были проведены эксперименты, в ходе которых было произведено сравнение времени работы каждого алгоритма. При этом были использованы последовательности разной длины. Результаты экспериментов показали, что алгоритм двоичного поиска является наиболее эффективным при работе с отсортированными последовательностями. Алгоритм линейного поиска работает соизмеримо по эффективности с алгоритмом бинарного поиска только на малых последовательностях, в случае увеличении размера последовательности начинает сильно отставать по эффективности.

Результаты данного исследования могут быть использованы в различных областях, где требуется поиск данных в отсортированных последовательностях. Например, это может быть поиск в базах данных, поиск элементов в массивах, поиск в текстовых файлах.

Работа была сделана в полном объеме, на основе результатов анализа, можно сделать вывод, что если необходимо проводить поиск в отсортированных последовательностях, то однозначно необходимо использовать алгоритм бинарного поиска.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 6](#_Toc132998549)

[1 МОДЕЛИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 7](#_Toc132998550)

[2 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 21](#_Toc132998551)

[3 ОЦЕНКА РАБОТЫ (ТЕСТИРОВАНИЕ) ПРОГРАММНОГО СРЕДСТВА И АНАЛИЗ РЕЗУЛЬТАТОВ 24](#_Toc132998552)

[ВЫВОДЫ 29](#_Toc132998553)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 30](#_Toc132998554)

[ПРИЛОЖЕНИЕ А 31](#_Toc132998555)

# ВВЕДЕНИЕ

Объектом исследования являются методы поиска в отсортированных последовательностях.

Конечной целью любой системы обработки данных является нахождение и предоставление нужной информации. Для достижения этой цели разрабатываются различные методы и алгоритмы поиска данных в отсортированных последовательностях.

Сравнительный анализ этих методов и алгоритмов является актуальной задачей в современной информатике и компьютерных науках. Это позволяет определить наиболее эффективные методы поиска данных в отсортированных последовательностях и выбрать наиболее подходящий алгоритм для решения конкретной задачи.

Цель данной курсовой работы заключается в проведении сравнительного анализа методов поиска данных в отсортированных последовательностях. В ходе работы будут рассмотрены различные алгоритмы и методы поиска.

Результаты данной работы могут быть использованы для выбора наиболее эффективного метода поиска данных в отсортированных последовательностях и повышения эффективности программного обеспечения, использующего эти методы.

Алгоритм – совокупность точно заданных правил решения некоторого класса задач или набор инструкций, описывающих порядок действий исполнителя для решения определённой задачи [1].

Для решения данной задачи необходимо создать консольное приложение, которое будет выполнять сравнение эффективности работы двух алгоритмов поиска в подготовленных отсортированных последовательностях и выводить результат работы программы в виде таблицы.

Так как различные программы в процессе своей работы могут искать какие-либо данные в отсортированных последовательностях, то сравнение скорости работы различных алгоритмов поиска – это важный момент, ведь если данных много, то при неэффективном алгоритме поиск данных может занять очень продолжительный момент времени.

Приложение для сравнения работы алгоритмов будет написано на языке программирования высокого уровня C#, с использованием IDE Microsoft Visual Studio 2019.

# 1 МОДЕЛИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

Моделирование используется во многих сферах жизни, начиная с ранних цивилизаций, таких как Древний Египет, Рим и Греция, где моделирование использовалось для создания мелкомасштабных планов в искусстве и архитектуре. Моделирование широко используется в науке и проектировании для предоставления абстракций системы с некоторым уровнем точности и детализации. Затем модель анализируется, чтобы лучше понять разрабатываемую систему.

При проектировании и разработке программного обеспечения на основе моделей моделирование программного обеспечения используется как неотъемлемая часть процесса разработки программного обеспечения. Модели строятся и анализируются до реализации системы и используются для руководства последующей реализацией. Лучшее понимание системы можно получить, рассматривая ее с различных точек зрения, например, модели требований, статические модели и динамические модели программной системы. Графический язык моделирования, такой как UML, помогает в разработке, понимания и передачи различных представлений.

Рассмотрим некоторые методы и нотации моделирования программного обеспечения.

Объектно-ориентированные концепции имеют решающее значение для анализа и проектирования программного обеспечения, поскольку они решают фундаментальные вопросы модифицируемости, адаптации и эволюции программного обеспечения. Объектно-ориентированные методы основаны на концепциях сокрытия информации, классов и наследования. Сокрытие информации может привести к созданию систем, которые являются более самодостаточными и, следовательно, более модифицируемыми и обслуживаемыми. Наследование обеспечивает подход для систематической адаптации класса. С распространением нотаций и методов для объектно-ориентированного анализа, и проектирования программных приложений, был разработан унифицированный язык моделирования (UML) для обеспечения стандартизированного графического языка и нотации для описания объектно-ориентированных моделей.

Однако, поскольку UML не зависит от методологии, его необходимо использовать вместе с объектно-ориентированным методом анализа и проектирования.

Современные методы объектно-ориентированного анализа и проектирования основаны на моделях и используют комбинацию моделирования сценариев использования, статического моделирования, моделирования машин состояний и моделирования взаимодействия объектов. Почти все современные объектно-ориентированные методы используют нотацию нотацию UML для описания требований к программному обеспечению, моделей анализа и проектирования.

Рассмотрим такое понятие, как жизненный цикл программного средства.

Жизненный цикл программного обеспечения (SDLC) – это поэтапный подход к разработке программного обеспечения с конкретными результатами и вехами на каждом этапе. Модель жизненного цикла программного обеспечения – это абстракция процесса разработки программного обеспечения, которую удобно использовать для целей планирования.

Жизненный цикл разработки систем состоит из отдельных этапов работы, которые используются системными инженерами и разработчиками систем для создания информационных систем. Как и все, что производится на конвейере, SDLC нацелен на создание высококачественных систем, которые соответствуют или превосходят ожидания, основанные на требованиях, путем поставки систем в запланированные сроки и сметы. Компьютерные системы сложны и часто объединяют компоненты различного происхождения.

Методологии SDLC вписываются в спектр гибкости: от гибких, итеративных и последовательных. Agile-методологии, такие как XP и Scrum, фокусируются на легких процессах, которые позволяют быстро вносить изменения. Итеративные методологии, такие как Rational Unified Process и метод разработки динамических систем, сосредоточены на стабилизации масштабов проекта и итеративном расширении или улучшении продуктов. Последовательные модели или модели «большого проектирования наперед» (BDUF), такие как водопад, фокусируются на полном и правильном планировании, чтобы направлять более крупные проекты и ограничивать риски для успешных и предсказуемых результатов.

SDLC – это не методология как таковая, а скорее описание фаз, которые должна рассматривать методология. Список фаз не является окончательным, но обычно включает планирование, анализ, проектирование, кодирование, тестирование, внедрение и сопровождение/поддержку.

Разберем эти этапы подробнее:

1. Анализ требований. Это первая фаза жизненного цикла разработки программного обеспечения, которая включает в себя сбор требований к программному обеспечению. Сбор и анализ требований является наиболее важной фазой SDLC. На этапе сбора требований бизнес-аналитик собирает требования от клиента и других заинтересованных сторон в соответствии с потребностями бизнеса и документирует их в документе.
2. Планирование. Следующим шагом в процессе SDLC является определение и документирование требований к продукту и их утверждение клиентом и другими заинтересованными сторонами. Этот документ известен как документ спецификации требований к программному обеспечению. Спецификация требований к программному обеспечению состоит из всех требований к продукту, которые должны быть спроектированы и разработаны в течение жизненного цикла разработки программного обеспечения.
3. Проектирование. Фаза проектирования в жизненном цикле разработки программного обеспечения включает в себя два функциональных этапа. Высокоуровневое проектирование обеспечивает архитектуру программного продукта. Проектирование низкого уровня объясняет функционирование каждой функции и компонента в продукте и их работу.
4. Разработка. На этапе разработки разработчики программного обеспечения начинают писать код продукта, происходит установка физического оборудования для серверов, дизайнерами создается пользовательский интерфейс. Фаза разработки – это процесс преобразования дизайна в рабочий программный продукт.
5. Тестирование. Команда тестирования проверяет, все ли требования были выполнены, убеждается, что все функциональные возможности работают так, как ожидалось, выявляет все возможные ошибки и сообщает о них в систему отслеживания ошибок. Затем эти ошибки передаются разработчикам и исправляются.
6. Развертывание/сопровождение. После разработки программного средства, успешного его тестирования, исправления всех ошибок, приложение попадает к конечному пользователю. В процессе эксплуатации могут быть выявлены ранее незамеченные ошибки, а также может потребоваться добавление нового функционала. Все это происходит в рамках этого этапа и до вывода приложения из эксплуатации.

Рассмотрим некоторые методологии разработки программного обеспечения.

Методология Agile – это один из лучших подходов к разработке программного обеспечения, позволяющий часто вносить изменения в проект разработки. Agile используется для минимизации рисков путем разработки программного обеспечения за короткие промежутки времени, которые называются итерациями и обычно длятся от одной недели до одного месяца.

Плюсы:

1. Удовлетворение потребностей клиентов за счет быстрого и непрерывного выпуска полезного программного обеспечения.
2. Акцент делается на человеческом взаимодействии, а не на процессе и инструментах разработки. Заказчики, разработчики и тестировщики постоянно взаимодействуют друг с другом.
3. Agile имеет адаптивный подход, способный реагировать на изменяющиеся требования клиентов.
4. Прямое общение и постоянная обратная связь с представителями заказчика.

Минусы:

1. Отсутствуют четкая структура и план проекта.
2. Требуется постоянное взаимодействие с клиентом.
3. Все процессы завязаны на команду.
4. Компания слишком сфокусирована на деталях.

Экстремальное программирование или же XP – это гибкая методология разработки программного обеспечения. Эта методология в основном используется для создания программного обеспечения в очень нестабильной среде. Она позволяет добиться большей гибкости в процессе моделирования. Основная цель этой модели – снизить стоимость требований к программному обеспечению.

Плюсы:

1. Основное преимущество экстремального программирования заключается в том, что эта методология позволяет компаниям, занимающимся разработкой программного обеспечения, экономить затраты и время, необходимое для реализации проекта. Экономия времени возможна благодаря тому, что XP фокусируется на своевременной доставке конечных продуктов. Команды экстремального программирования экономят много денег, потому что они не используют слишком много документации.

2. Подчеркивает вовлеченность клиента.

3. Помогает установить рациональные планы и графики и заставить разработчиков лично придерживаться своих графиков, что, безусловно, является большим преимуществом модели XP.

4. Эта модель согласуется с большинством современных методов разработки.

Минусы:

1. Некоторые специалисты говорят, что экстремальное программирование сосредоточено на коде, а не на дизайне. Это может быть проблемой, потому что хороший дизайн чрезвычайно важен для программных приложений. Он помогает продавать их на рынке программного обеспечения. Кроме того, в проектах XP не всегда хорошо ведется документация по дефектам.

2. Эта методология эффективна лишь настолько, насколько эффективны вовлеченные в нее люди.

3. Требует проведения совещаний через короткие промежутки времени.

4. Она требует слишком много изменений в разработке, которые очень трудно каждый раз принимать разработчику программного обеспечения.

5. В этой методологии, как правило, невозможно узнать точную оценку трудозатрат, необходимых для составления сметы, поскольку в начале проекта никто не знает всего объема и требований проекта.

BDD (разработка через поведение) – это методология разработки программного обеспечения, являющаяся ответвлением от методологии разработки через тестирование.

BDD – это несколько вещей:

1. Это язык, используемый для определения поведения приложения.

2. Это инструмент для совместной работы менеджеров по продукту, инженеров и тестировщиков.

3. Это система для автоматического тестирования такого поведения.

BDD был создан более десяти лет назад Дэном Нортом, консультантом по технологиям и организационным вопросам из Лондона, как способ устранить путаницу между разработчиками, тестировщиками и бизнесменами. Хотя он начинался как простая модификация разработки на основе тестирования, BDD сегодня является полноценной методологией разработки программного обеспечения.

Разработка через поведение устраняет разрыв и обеспечивает более простой способ сотрудничества и коммуникации между различными отделами, чтобы они могли более эффективно управлять проектами по разработке программного обеспечения и обеспечивать их выполнение. BDD гарантирует, что проекты разработки остаются сфокусированными на реальных потребностях бизнеса и в то же время отвечают требованиям пользователей.

Вне зависимости от того, внедряете ли вы еще BDD или ваша команда уже использует TDD и хочет рассмотреть BDD, очевидно, что преимуществ у разработки много. Но даже самые лучшие подходы к разработке имеют свои недостатки, и BDD не является исключением.

Хотя, по словам ее создателя, методология была разработана для решения повторяющихся проблем в преподавании разработки на основе тестирования, очевидно, что разработка на основе поведения требует знакомства с более широким кругом концепций по сравнению с TDD.

Плюсы:

1. Кооперация между всеми участниками разработки. Благодаря общему языку владельцы продукта, разработчики и тестировщики имеют полное представление о ходе проекта.

2. Более короткая кривая обучения. Поскольку BDD объясняется очень простым языком, кривая обучения для всех участников процесса намного короче.

3. Высокая наглядность. Будучи по своей природе нетехническим процессом, разработка, ориентированная на поведение, может охватить гораздо более широкую аудиторию.

4. Быстрые итерации. BDD позволяет быстро реагировать на все отзывы пользователей и вносить улучшения, отвечающие их потребностям.

5. Устранение потерь. BDD позволяет четко передавать требования, что уменьшает количество переделок.

6. Фокус на потребностях пользователей. Удовлетворенные пользователи полезны для бизнеса, а методология BDD позволяет удовлетворять потребности пользователей в процессе разработки программного обеспечения.

7. Выполнение бизнес-целей. При использовании BDD каждая разработка может быть отслежена до реальных бизнес-целей.

Минусы:

1. Для работы в BDD требуется предварительное знакомство с концепциями TDD.

2. Разработка через поведение несовместима с водопадным подходом.

3. Тестировщики, использующие BDD, должны обладать достаточным техническим опытом.

4. BDD может оказаться неэффективным, если требования не были тщательно прописаны.

5. Помните, что использование BDD не требует использования определенных программных языков или инструментов. Это в первую очередь концептуальный подход, и поэтому превращение его в чисто техническую практику или даже в практику, зависящую от конкретных инструментов, означает, что вы полностью упустили суть.

Каскадная модель

Это простой, линейный метод, в котором этапы разработки организованы в последовательные, каскадные процессы.

Водопадная методология разработки легко воспринимается, что делает ее популярной для команд с меньшим опытом разработки. Каждый этап должен быть завершен, прежде чем переходить к следующему. Например, все требования должны быть установлены до начала проектирования.

Плюсы:

1. Линейность водопадной модели делает ее простой для понимания, особенно для начинающих разработчиков.

2. Все спецификации и результаты прописаны до начала разработки.

3. В водопадной модели нет возможности для неправильного восприятия информации, поскольку она четко определена на каждом этапе.

Минусы:

1. Не учитывается обратная связь с заказчиком на ранних стадиях, что повышает риск отклонения проекта от цели.

2. Тестирование проводится только в конце разработки. Некоторые проблемы сложнее устранить на более поздней стадии.

3. Жесткость водопадной модели не оставляет места для изменений, что делает ее непригодной для сложных проектов.

4. Команда может тратить слишком много времени на документацию вместо того, чтобы предоставлять решения, которые решают проблемы пользователей.

Методология бережливой разработки

В рамках бережливой методологии перед разработчиками ставится задача выявить узкие места, которые могут помешать процессу. Цель – создать эффективную систему, которая будет работать безупречно. Методология также подчеркивает уважение к человеку, что означает, что общение является ключом к улучшению сотрудничества в команде.

Плюсы:

1. Сокращает потери в проекте, такие как избыточный код, ненужная документация и повторяющиеся задачи.

2. Общая стоимость разработки снижается благодаря принципам бережливого производства.

3. Сокращается время вывода программного обеспечения на рынок, поскольку бережливая разработка способствует повышению эффективности.

4. Повышается мотивация членов команды, поскольку они получают больше полномочий для принятия решений.

Минусы:

1. Для успешного внедрения бережливой разработки вам потребуется команда высококвалифицированных разработчиков, которую не так-то просто собрать.

2. Менее квалифицированные разработчики могут быть перегружены обязанностями и потерять концентрацию на проекте.

3. Необходима подробная документация, что создает огромную нагрузку на бизнес-аналитика.

Методология прототипирования программного обеспечения

Вместо разработки полноценного программного обеспечения модель прототипа позволяет разработчикам работать над прототипом конечной версии продукта.

На основе собранных отзывов прототип проходит несколько итераций доработки, пока заказчик не сочтет его удовлетворительным. Привлекательность подхода с использованием прототипа заключается в его тщательной оценке, которая позволяет выявить возможные проблемы до начала фактической разработки.

Успех этого подхода заключается не только в команде разработчиков, но и в том, насколько хорошо они общаются с клиентами при проведении тестирования. Стоит также отметить, что разработчики часто берут на себя расходы по созданию прототипа.

Плюсы:

1. Хорошо помогает устранить потенциальные проблемы на ранней стадии разработки, что значительно снижает риск отказа продукта.

2. Гарантирует, что заказчик доволен "продуктом", прежде чем начнется реальная разработка.

3. Установление взаимопонимания с заказчиком на ранней стадии обсуждения, что помогает на протяжении всего проекта.

4. Сбор подробной информации с помощью прототипа, которая в дальнейшем используется при создании окончательной версии.

Минусы:

1. Чрезмерное тестирование прототипа с заказчиком может затянуть сроки разработки.

2. Ожидания заказчика от реального продукта могут не совпадать с прототипом.

3. Существует риск перерасхода средств, поскольку работы по созданию прототипа часто оплачиваются разработчиком.

Методология Rational Unified Process, которую называют RUP, обеспечивает разработку программного обеспечения с использованием рациональных инструментов. Была придумана в 2003 Rational Software Corportaion.

Эта методология разделяет процесс расширения на четыре различных этапа, каждый из которых включает в себя бизнес-моделирование, анализ и проектирование, внедрение, тестирование и утилизацию.

Это объектно-ориентированная методология разработки программ с использованием веб-технологий.

Модель призвана помочь разработчикам программного обеспечения в определении руководящих принципов, шаблонов и образцов для всех функций и этапов разработки программного обеспечения.

Плюсы:

1. Уделяется большое внимание документации.

2. Устраняет риски проекта, связанные с меняющимися потребностями клиента

3. Очень низкие требования к интеграции.

Минусы:

1. Требуется наличие высококвалифицированных разработчиков программного обеспечения.

2. Процедура разработки сложна.

3. Интеграция может привести к путанице.

4. Сложная для понимания.

Методология RAD

Модель быстрой разработки приложений (RAD) была представлена в 1991 году и послужила основой для современных итеративных фреймворков. Она фокусируется на создании продуктов в гораздо более короткие сроки без ущерба для качества.

RAD – это 4-этапная структура, которая представляет собой защиту требований проекта, прототипирование, тестирование и внедрение. В отличие от линейных моделей, RAD делает акцент на создании прототипов с заданными требованиями и их тестировании у заказчика. Это делается в течение нескольких итераций, пока заказчик не будет доволен результатами.

Тщательное тестирование прототипа приводит к получению ценной обратной связи, которая помогает устранить большую часть рисков, связанных с продуктом. Использование RAD приводит к высоким шансам успешного выпуска продукта в установленные сроки. В RAD часто используются инструменты разработки, которые могут автоматизировать и упростить процесс разработки.

Плюсы:

1. Снижение рисков благодаря регулярной обратной связи с клиентами.

2. Повышение удовлетворенности клиентов.

3. Хорошо подходит для небольших и средних приложений.

4. Сокращает время выхода на рынок.

Минусы:

1. Высокая зависимость от отзывчивого заказчика.

2. Требуется команда высококвалифицированных и опытных разработчиков.

3. Не подходит для проектов с ограниченным бюджетом.

4. Отсутствие документации для отслеживания хода работы.

Методология Scrum

Scrum – это, пожалуй, одна из самых гибких методологий разработки программного обеспечения. Она основана на философии Agile и предпочитается за инкрементный и итеративный подходы. В методологии Scrum участвуют владелец продукта, мастер Scrum и команда разработчиков.

Владелец продукта получает информацию от клиента и следит за тем, чтобы команда выполняла требования клиента. Scrum-мастер выступает в роли координатора и следит за тем, чтобы члены команды были знакомы с процессом Scrum. Команда берет на себя ответственность за выполнение разработки.

Что делает Scrum идеальной методологией в быстро меняющейся среде, так это то, как задачи выполняются в спринтах. Каждый спринт занимает до 4 недель. Быстрота выполнения позволяет командам за короткий период времени выявить проблемы, представить решения, протестировать и собрать обратную связь. Это значительно облегчает работу над быстро меняющимися проектами.

Плюсы:

1. Короткие итерации позволяют быстро решать проблемы.

2. Scrum очень быстро реагирует на изменения, поскольку процесс включает в себя регулярную обратную связь.

3. Scrum экономичен и эффективен.

4. Регулярные встречи позволяют членам команды всегда быть на одной волне.

5. Вклад отдельных членов команды замечается и ценится на собраниях Scrum.

Минусы:

1. Для того чтобы Scrum работал, все члены команды должны быть одинаково квалифицированы и преданы делу.

2. Ежедневные собрания Scrum могут быть утомительными для членов команды.

3. Может увеличиться время выхода на рынок, если нет строгого контроля над сроками.

4. Не подходит для больших проектов.

Задача данной курсовой работы состоит в проведении сравнительного анализа методов поиска данных в отсортированных последовательностях. Для выполнения этой задачи необходимо:

1. Изучить существующие методы поиска данных в отсортированных последовательностях.
2. Разработать программное средство для реализации выбранных методов поиска данных в отсортированных последовательностях.
3. Протестировать программное средство на различных объемах данных и произвести анализ результатов.
4. Сравнить эффективность и производительность различных методов поиска данных в отсортированных последовательностях на основе полученных результатов.
5. Сделать выводы о наиболее эффективных методах поиска данных в отсортированных последовательностях и их применимости для решения конкретных задач.

Требуется разработать консольное приложение на языке программирования высока уровня C#. Программа должна читать готовые отсортированные массивы из файла.

При запуске программы открывается консоль, в которую необходимо ввести число, которое мы будем искать в последовательности. Программа проводит поиск используя алгоритмы последовательного и бинарного поиска. Поиск проводится на отсортированных последовательностях, указанных во входном файле, то есть при надобности можно его дополнить, и программа отработает точно так же, но на другом объеме данных. В конце работы программа выводит таблицу, в которой содержится информация о количестве итераций, необходимых для поиска элемента в последовательности для каждого алгоритма.

Вычислительная сложность – понятие в информатике и теории алгоритмов, обозначающее функцию зависимости объёма работы, которая выполняется некоторым алгоритмом, от размера входных данных [2].

Оценка сложности алгоритма является важной задачей в информатике, так как позволяет оценить производительность алгоритмов и сравнить их между собой. В данной курсовой работе рассматривается оценка сложности алгоритмов поиска данных в отсортированных последовательностях.

Сложность алгоритма оценивается с помощью асимптотической нотации, которая позволяет оценить, как быстро растет время выполнения алгоритма при увеличении размера входных данных. Оценка сложности алгоритма включает в себя оценку временной и пространственной сложности.

Наиболее часто используемые обозначения асимптотической нотации это O-большое, Ω-большое и Θ-большое. O-большое («большое O») показывает верхнюю границу сложности алгоритма, то есть скорость роста времени выполнения алгоритма не превышает функцию, описываемую O-большим. Ω-большое («большое Омега») показывает нижнюю границу сложности алгоритма, то есть скорость роста времени выполнения алгоритма не медленнее функции, описываемой Ω-большим. Θ-большое («большое тета») показывает точную оценку сложности алгоритма, то есть скорость роста времени выполнения алгоритма точно соответствует функции, описываемой Θ-большим.

Для оценки сложности алгоритмов используем нотацию Big O.

Алгоритм бинарного поиска и алгоритм последовательного (линейного) поиска являются двумя основными алгоритмами поиска элемента в массиве. Оба алгоритма используются в различных сферах, но различаются в эффективности и скорости выполнения.

Алгоритм бинарного поиска работает на отсортированном массиве данных и заключается в том, чтобы постоянно делить массив на половины, и проверять в какой из них находится искомый элемент. Каждый раз, когда делается новое сравнение, массив сужается вдвое, и таким образом алгоритм рекурсивно повторяет эту операцию до тех пор, пока искомый элемент не будет найден.

Алгоритм бинарного поиска можно описать следующим образом:

1. Определить центральный элемент отсортированного массива.
2. Сравнить этот центральный элемент с искомым элементом.
3. Если элементы равны, поиск завершается.
4. Если искомый элемент меньше центрального, продолжить поиск в левой половине массива.
5. Если искомый элемент больше центрального, продолжить поиск в правой половине массива.
6. Повторять шаги 1–5 до тех пор, пока элемент не будет найден или массив не будет полностью просмотрен.

Временная сложность алгоритма бинарного поиска составляет O(log n), где n – размер отсортированного массива. Это означает, что время выполнения алгоритма растет логарифмически с увеличением размера массива.

Алгоритм последовательного поиска:

1. Пройти по всему отсортированному списку, начиная с первого элемента.
2. Сравнить каждый элемент с искомым значением.
3. Если искомое значение найдено, вернуть индекс искомого элемента.
4. Если искомое значение не найдено, вернуть -1.

Сравнение алгоритмов:

1. Сложность алгоритма бинарного поиска составляет O(log n), что означает, что время выполнения алгоритма увеличивается логарифмически при увеличении размера списка. В то время как сложность алгоритма последовательного поиска составляет O(n), что означает, что время выполнения алгоритма растет линейно с увеличением размера списка.
2. Бинарный поиск работает эффективно только в случае отсортированных списков, тогда как последовательный поиск может применяться для неупорядоченных списков.
3. Бинарный поиск обычно требует меньше операций сравнения, чем последовательный поиск, что делает его более эффективным для больших списков.
4. Бинарный поиск не всегда находит первый элемент, равный искомому значению, тогда как последовательный поиск всегда находит первый элемент, равный искомому значению.
5. Таким образом, бинарный поиск является более эффективным алгоритмом поиска в отсортированных списках, чем последовательный поиск. Однако, если список не отсортирован или имеет малый размер, то последовательный поиск может оказаться более эффективным.

Скорость роста вычислительной сложности алгоритма последовательного поиска растет гораздо быстрее, чем алгоритма бинарного поиска. Для лучшего понимания сравним графики двух функций на рисунке 1.1. Зеленый цвет – последовательный поиск, синий – бинарный.



Рисунок 1.1 – Сравнение графиков функций

Разберем алгоритмы последовательного и бинарного поиска. Схема алгоритма последовательного поиска представлена на рисунке 1.2. Схема алгоритма бинарного поиска представлена на рисунке 1.3.

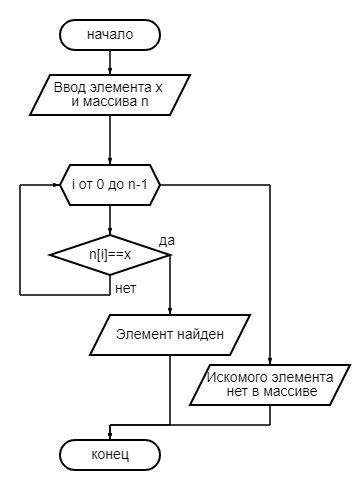


Рисунок 1.2 – Блок-схема алгоритма последовательного поиска

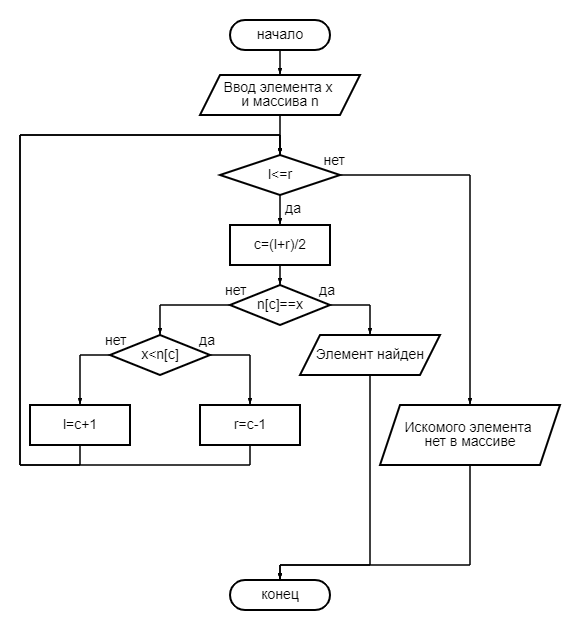


Рисунок 1.3 – Блок-схема алгоритма бинарного поиска

# 2 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

Проектирование программного средства – это важный этап разработки, на котором определяется архитектура, структура и функциональность программы.

Проектирование программного средства является ключевым этапом в разработке программного продукта, поскольку от его правильного выполнения зависит дальнейшая работа над проектом. Этот процесс может быть разбит на несколько этапов, каждый из которых имеет свои собственные задачи и цели.

Одной из первых задач проектирования программного средства является сбор и анализ требований. На этом этапе осуществляется определение функциональных и нефункциональных требований к программному продукту, а также анализ его целевой аудитории и конкурентной среды.

После этого осуществляется разработка архитектуры программного продукта. Архитектура программного продукта определяет его структуру, компоненты, интерфейсы и протоколы взаимодействия между компонентами. Она также определяет, как компоненты программного продукта будут взаимодействовать друг с другом, как будут обмениваться данными и как будут решаться возможные проблемы.

Далее проводится этап проектирования деталей программного продукта. На этом этапе специфицируются алгоритмы, используемые в программном продукте, проектируются пользовательские интерфейсы и проводится другая детализация, необходимая для создания полного программного продукта.

В конце процесса проектирования программного средства проводится тестирование и отладка программного продукта. На этом этапе проверяется, соответствует ли продукт требованиям, и исправляются любые ошибки, которые могут быть обнаружены.

Одной из основных целей проектирования программного средства является создание программного продукта, который будет легко использоваться и понятен для конечного пользователя, а также который будет работать без сбоев и проблем. Кроме того, проектирование программного средства должно учитывать возможность его дальнейшего развития и совершенствования, чтобы удовлетворять растущим потребностям пользователей.

Для проектирования программного средства для сравнительного анализа методов поиска данных в отсортированных последовательностях, необходимо определить следующие компоненты:

Интерфейс пользователя: необходимо определить, каким образом пользователь будет взаимодействовать с программным средством. Например, пользователь может вводить отсортированные последовательности и запрашивать поиск определенных значений.

Алгоритмы поиска: необходимо реализовать алгоритмы бинарного и последовательного поиска.

Модуль сравнения: необходимо разработать модуль, который позволит сравнивать результаты работы алгоритмов.

Модуль генерации тестовых данных: для проверки эффективности алгоритмов поиска необходимо сгенерировать тестовые данные, на основе которых будут производиться замеры времени выполнения.

Модуль измерения времени выполнения: для сравнения эффективности алгоритмов необходимо замерять время их выполнения на тестовых данных.

Модуль анализа результатов: необходимо разработать модуль, который позволит анализировать результаты замеров времени выполнения и сравнивать эффективность алгоритмов.

Тестирование: после завершения проектирования необходимо провести тестирование программного средства на тестовых данных для проверки его работоспособности и эффективности [3].

Программа разрабатывается на языке программирования высокого уровня C# с использованием IDE Microsoft Visual Studio 2019.

Для проектирования программного средства, которое позволит сравнить алгоритмы поиска данных в отсортированных последовательностях, необходимо определить основные требования к программному продукту и его функциональность.

Основные требования к программному средству:

1. Возможность загрузки отсортированной последовательности данных из внешнего файла.
2. Реализация алгоритмов бинарного и последовательного поиска для поиска элементов в последовательности.
3. Возможность задания искомого элемента для поиска.
4. Возможность измерения времени выполнения каждого из алгоритмов поиска и сравнения их результатов.
5. Графический интерфейс для удобства работы с программой.

Функциональность программного средства:

1. Загрузка отсортированной последовательности данных из внешнего файла.

2. Реализация алгоритмов бинарного и последовательного поиска. Программа должна иметь возможность выполнять оба алгоритма поиска и возвращать результат поиска пользователю.

3. Возможность задания искомого элемента. Пользователь должен иметь возможность ввести искомый элемент для поиска в последовательности.

4. Измерение времени выполнения каждого из алгоритмов поиска. Программа должна измерять время выполнения каждого из алгоритмов поиска и выводить результаты на экран.

5. Графический интерфейс. В качестве интерфейса используется консоль, в которой пользователь задает элемент для поиска и получает результат работы программы в виде таблицы.

6. Также выводить полученную таблицу с данным в файл.

Для изучения особенностей реализации алгоритмов использовалась книга Дональда Кнута, которая представляет собой исчерпывающее исследование алгоритмов и структур данных [4].

В процессе написания программы для выполнения поставленной задачи, использовались следующие элементы языка программирования:

1. Класс StreamReader, для чтения данных из файла.
2. Класс StreamWriter, для записи результата работы программы в файл.
3. Класс List, в качестве массива, в котором хранились прочтенные данные из файла.
4. Также использовались такие функции языка как цикл с предусловием, цикл с параметром, условный оператор.

# 3 ОЦЕНКА РАБОТЫ (ТЕСТИРОВАНИЕ) ПРОГРАММНОГО СРЕДСТВА И АНАЛИЗ РЕЗУЛЬТАТОВ

Оценка работы программного средства в рамках данной курсовой работы будет проводиться на основе анализа результатов, полученных в процессе сравнительного анализа алгоритмов поиска данных в отсортированных последовательностях. Для этого необходимо провести следующие шаги:

Провести тестирование программного средства. На этом этапе необходимо убедиться в корректной работе программы при различных сценариях использования и в различных условиях. Для этого можно использовать тестовые данные с разными размерами отсортированных последовательностей и разными искомыми элементами.

Получить результаты измерения времени выполнения алгоритмов. Для этого необходимо измерить время, затраченное каждым алгоритмом на поиск искомого элемента в отсортированной последовательности данных. Это можно сделать путем запуска каждого алгоритма несколько раз и усреднения результатов.

Проанализировать результаты измерения времени выполнения алгоритмов. Необходимо сравнить время, затраченное на поиск искомого элемента с помощью каждого из алгоритмов. Также следует оценить, насколько эффективным был каждый из алгоритмов в зависимости от размера отсортированной последовательности данных.

Произвести сравнительный анализ алгоритмов поиска. На основе результатов измерения времени выполнения алгоритмов следует провести сравнительный анализ и оценить, какой из алгоритмов является более эффективным для поиска данных в отсортированных последовательностях.

Сделать выводы. На основе результатов тестирования и анализа эффективности каждого из алгоритмов следует сделать выводы о том, какой алгоритм является более эффективным для поиска данных в отсортированных последовательностях.

Тестирование программы проводится на 5 массивах длиной 10, 100, 1000, 10000, 100000 элементов, которые при этом отсортированы.

Для измерения времени выполнения каждого из алгоритмов можно использовать возможности, предоставляемый классом StopWatch, который позволит нам засечь время перед началом выполнения алгоритма поиска и засечь время в момент окончания работы алгоритма, что позволит нам найти количество времени, за которое выполняется алгоритм. [5]

Важно также учитывать, что время выполнения алгоритмов может зависеть от аппаратной конфигурации компьютера, на котором проводятся эксперименты. Поэтому необходимо обеспечить одинаковые условия для каждого из экспериментов, например, использовать один и тот же компьютер с одинаковыми настройками.

Проведем несколько замеров скорости работы алгоритмов, но проводя поиск разных чисел, результаты которых представлены на рисунках 3.1, 3.2, 3.3, 3.4,3.5.

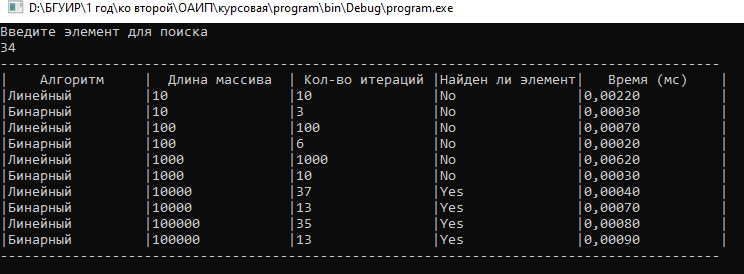


Рисунок 3.1 – Результат работы программы при поиске числа 34

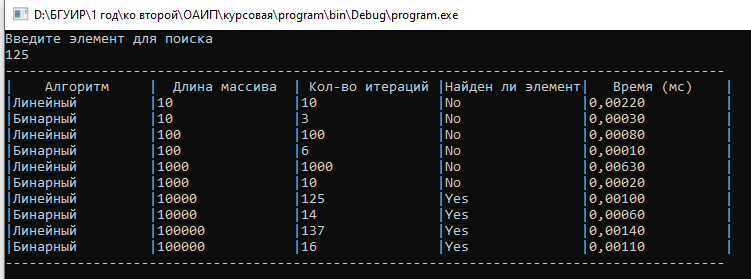


Рисунок 3.2 – Результат работы программы при поиске числа 125

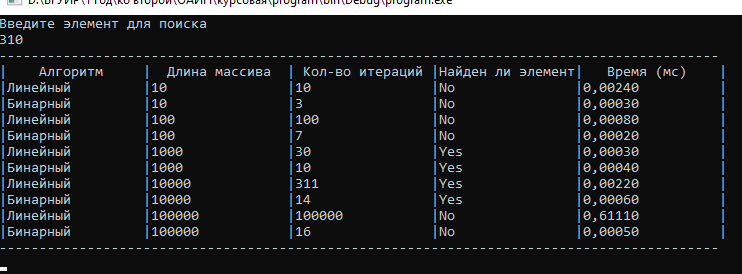


Рисунок 3.3 – Результат работы программы при поиске числа 310

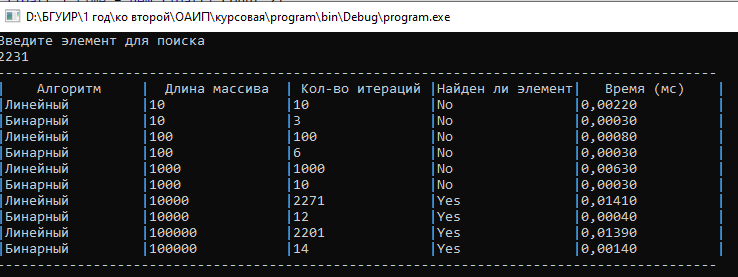


Рисунок 3.4 – Результат работы программы при поиске числа 2231

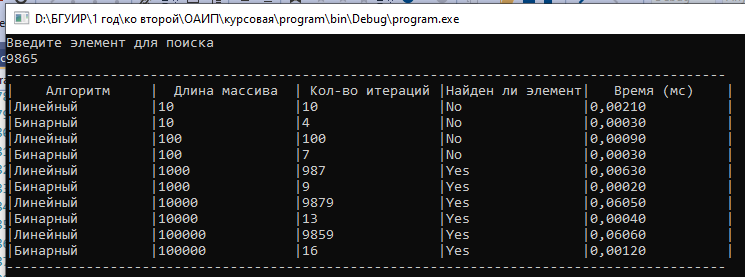


Рисунок 3.5 – Результат работы программы при поиске числа 9865

Как видно из полученных результатов алгоритм бинарного поиска быстрее алгоритма линейного поиска, но, если искомый элемент находится вначале массива алгоритм последовательного поиска может добраться до него быстрее чем алгоритм бинарного поиска.

В графе кол-во сравнений отображается количество итераций алгоритма, которые алгоритму потребовалось пройти, чтобы однозначно найти элемент или же установить, что искомого элемента нет в массиве.

В графе время указывается количество миллисекунд, за которые отработал алгоритм.

Как мы видим алгоритм бинарного поиска работает за меньшее количество времени, чем алгоритм линейного поиска. Это особенно заметно с увеличением размера массива.

Можно было так же провести тесты на еще больших объемах данных, но становится очевидным, что отрыв был бы еще больше.

В качестве элементов для поиска выступали числа, но эти алгоритмы будут работать для любых других последовательностей, независимо от типа объекта, главное, чтобы был описан способ сравнения данных объектов.

Блок-схема программы, которая проводила сравнение алгоритмов поиска в отсортированных последовательностях изображена на рисунке 3.6.

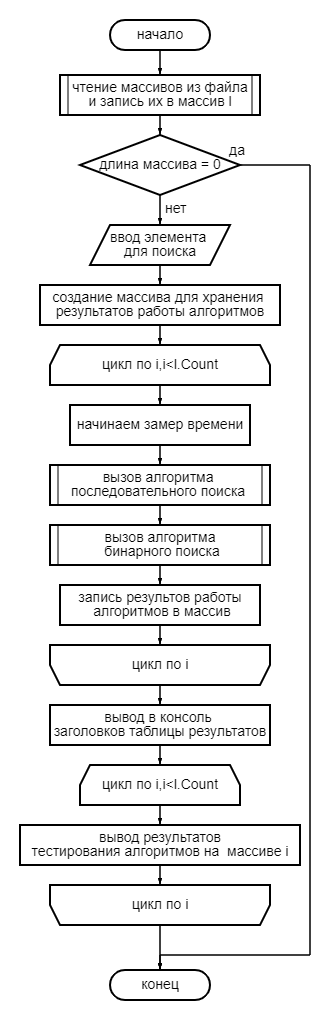


Рисунок 3.6 – Блок-схема основного алгоритма программы

# ВЫВОДЫ

В данной курсовой работе был проведен сравнительный анализ двух методов поиска данных в отсортированных последовательностях: бинарного и последовательного.

Было разработана программа на языке программирования C# для поиска данных в отсортированных последовательностях. В рамках разработки были реализованы оба метода поиска, а также вычисление времени выполнения каждого метода.

В результате тестирования было установлено, что бинарный поиск работает значительно быстрее последовательного поиска при поиске данных в отсортированных последовательностях на объемах данных, превышающих 10 000 элементов. При этом, на малых объемах данных, преимущество бинарного поиска было незначительным.

Таким образом, можно заключить, что бинарный поиск является более эффективным методом поиска данных в отсортированных последовательностях на больших объемах данных, тогда как последовательный поиск может быть полезен на малых объемах данных или в случае, когда последовательность неотсортирована.

В целом, данная работа помогла лучше понять принципы работы методов поиска данных в отсортированных последовательностях, а также дала возможность провести сравнительный анализ и выбрать наиболее оптимальный метод для конкретной задачи.

Уникальность работы при проверке на antiplagiat.ru равняется 98,92 %.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Вирт, Н. Алгоритмы и структуры данных / Н. Вирт. – СПб. : Невский диалект, 2005. − 352 с.

2. Джонсон, Д. Вычислительные машины и труднорешаемые задачи / Д. Джонсон, М. Гери. – М. : Мир, 1982. – 419 с.

3. Куликов, С. Тестирование программного обеспечения. Базовый курс. 3-e издание / С. Куликов, М. Гери. – М. : ЧЕТЫРЕ ЧЕТВЕРТИ, 2020. – 362 с.

4. Кнут, Д. Э. Искусство программирования. В 3 т. Т 3: Сортировка и поиск / Д. Э. Кнут. – М . : Вильямс, 2011. − 824 с.

5. Джепикс, Ф. Язык программирования C# 7 и платформы .NET и .NET Core / Ф. Джепикс, Э. Троелсен, Дж. Сети, Р., Ульман, Д. – М. : Издательский дом "Вильямс", 2018. – 1328 с.

# ПРИЛОЖЕНИЕ А

(обязательное)

**Фрагменты программного кода**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.IO;

using System.Threading;

using System.Threading.Tasks;

using static program.Utilities;

using System.Diagnostics;

namespace program

{

class Program

{

static List<List<int>> ReadFromFile(StreamReader stream)

{

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

while (!stream.EndOfStream)

list.Add(stream.ReadLine().Split(' ').Select(int.Parse).ToList()); // в цикле парсим каждую строку входного файла, которую затем преобразуем в один элемент массива, повторяем до конца файла

return list; // возвращаем полученный массив

}

static bool SequentialSearch(List<int> array, int element, out int index, out int comparisons)

{

comparisons = 0;

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

{

comparisons++; // считаем количество сравнений

if (array[i] == element) // проходим по всем элементам массива и сравниваем их с искомым, если находим, то запоминаем индекс и возвращаем положительный ответ

{

index = i;

return true;

}

}

index = -1;

return false; // если мы не смогли найти искомый элемент, возвращаем false и индекс -1

}

static bool BinarySearch(List<int> array, int element, out int index, out int comparisons)

{

int l = 0; // левый указатель

int r = array.Count - 1; // правый указатель

comparisons = 0;

while (l <= r) // пока между указателями расстояние не отрицательно

{

int c = l + (r - l) / 2; // находим центральный элемент

comparisons++;

if (array[c] == element) // сравниваем центральный элемент с искомымы, если нашли, то возвращаем ответ true и индекс элемента

{

index = c;

return true;

}

if (element < array[c]) // если мы не нашли элемент, то сдвигаем границу поиска, ориентируясь на отношение между центральным и искомым элементом (в данном случае элемент меньше центрального, то сдвигаем правую границу, в ином случае левую границу)

{

r = c - 1;

}

else

{

l = c + 1; //сдвигаем леву границу

}

}

index = -1;

return false; // если так и не нашли элемент - возвращаем false и индекс -1

}

static void Main(string[] args)

{

int index;

int comparisons;

//Utilities.Generate();

List<List<int>> l = ReadFromFile(new StreamReader("in.txt")); // читаем данные с входного файла

if (l.Count == 0) // проверка на пустоту файла

{

Console.WriteLine("нет массива");

Console.ReadKey();

return;

}

Console.WriteLine("Введите элемент для поиска");

int element = Convert.ToInt32(Console.ReadLine()); // ввдоим элемент для поиска

SequentialSearch(new List<int>(1), 1, out index, out comparisons); // вызываем функции для предворительной компиляции, для более точных значений позднее

BinarySearch(new List<int>(1), 1, out index, out comparisons);

int[,] m = new int[l.Count,4]; // массив, в котором будем хранить результаты тестов

Stopwatch sw = new Stopwatch(); // Счетчик прошедшего времени

float[,] time = new float[l.Count,2];

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

{

List<int> arr = l[i]; // построчно считываем последовательности из массива

sw.Start(); // начинаем замер времени

SequentialSearch(arr, element, out index, out comparisons); // вызов алгоритма поиска

sw.Stop(); // останавливаем счетчик

m[i, 0] = comparisons; // записываем кол-во сравнений

m[i, 2] = index; // записываем индекс

m[i, 3] = arr.Count; // записываем длину массива

time[i, 0] = (float)sw.Elapsed.Ticks/TimeSpan.TicksPerMillisecond; // записываем время

sw.Reset(); // сбрасываем счетчик

sw.Start(); // начинаем замер времени

BinarySearch(arr, element, out index, out comparisons); // вызов алгоритма поиска

sw.Stop(); // останавливаем счетчик

time[i, 1] = (float)sw.Elapsed.Ticks / TimeSpan.TicksPerMillisecond; // записываем время

sw.Reset(); // сбрасываем счетчик

m[i, 1] = comparisons; // записываем кол-во сравнений

}

PrintLine(); // отрисовываем верхнюю границу таблицы

PrintRow(true,"Алгоритм","Длина массива","Кол-во итераций","Найден ли элемент", "Время (мс)"); // рисуем названия столбцов

for (int i = 0; i < m.GetLength(0); i++) // перебираем полученные данные

{

PrintRow(false,"Линейный", m[i, 3].ToString(), m[i, 0].ToString(), m[i, 2] != -1 ? "Yes" : "No", time[i, 0].ToString("0.00000")); // выводим данные для алгоритма линейного поиска при обработке i-го массива

PrintRow(false,"Бинарный", m[i, 3].ToString(), m[i, 1].ToString(), m[i, 2] != -1 ? "Yes" : "No", time[i, 1].ToString("0.00000")); // выводим данные для алгоритма бинарного поиска при обработке i-го массива

}

PrintLine(); // отрисовываем нижнюю границу таблицы

Console.ReadKey();

}

}

}

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace program

{

class Utilities

{

static int tableWidth = 90;

static StreamWriter writer = new StreamWriter(new FileStream("out.txt", FileMode.Create));

public static void Generate() // заполняет файл массивами для проведения последующего анализа алгоритмов

{

File.WriteAllText("in.txt", string.Empty);

StreamWriter strw = new StreamWriter("in.txt");

Random r = new Random();

int[] l = new int[10];

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

{

l[i] = r.Next(10000);

}

Array.Sort(l);

strw.WriteLine(string.Join(" ", l.Select(x => x.ToString())));

l = new int[100];

for (int i = 0; i < 100; i++)

{

l[i] = r.Next(10000);

}

Array.Sort(l);

strw.WriteLine(string.Join(" ", l.Select(x => x.ToString())));

l = new int[1000];

for (int i = 0; i < 1000; i++)

{

l[i] = r.Next(10000);

}

Array.Sort(l);

strw.WriteLine(string.Join(" ", l.Select(x => x.ToString())));

l = new int[10000];

for (int i = 0; i < 10000; i++)

{

l[i] = r.Next(10000);

}

Array.Sort(l);

strw.WriteLine(string.Join(" ", l.Select(x => x.ToString())));

l = new int[100000];

for (int i = 0; i < 100000; i++)

{

l[i] = r.Next(100000);

}

Array.Sort(l);

strw.WriteLine(string.Join(" ", l.Select(x => x.ToString())));

strw.Flush();

strw.Close();

}

public static void PrintLine() // печатает строку из дефисов

{

Console.WriteLine(new string('-', tableWidth));

writer.WriteLine(new string('-', tableWidth));

writer.Flush();

}

public static void PrintRow(bool align, params string[] columns) // печатает в строку переданные строки при этом выравнивая их

{

int width = (tableWidth - columns.Length) / columns.Length;

string row = "|";

if (align)

{

foreach (string column in columns)

{

row += AlignCentre(column, width) + "|";

}

}

else

{

foreach (string column in columns)

{

row += AlignLeft(column, width) + "|";

}

}

Console.WriteLine(row);

writer.WriteLine(row);

writer.Flush();

}

static string AlignCentre(string text, int width) // выравнивает по центру переданную строку

{

text = text.Length > width ? text.Substring(0, width - 3) + "..." : text;

if (string.IsNullOrEmpty(text))

{

return new string(' ', width);

}

else

{

return text.PadRight(width - (width - text.Length) / 2).PadLeft(width);

}

}

static string AlignLeft(string text, int width) // выравнивает переданную строку по левому краю

{

text = text.Length > width ? text.Substring(0, width - 3) + "..." : text;

if (string.IsNullOrEmpty(text))

{

return new string(' ', width);

}

else

{

return text+new string(' ',width-text.Length);

}

}

}

}