МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИМЕНИ И.С. ТУРГЕНЕВА»

Кафедра программной инженерии

Работа допущена к защите

\_\_\_\_\_\_\_\_\_\_\_\_\_\_Руководитель

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_г.

**КУРСОВОЙ ПРОЕКТ**

по дисциплине «Качество и тестирование программного обеспечения»

на тему: «Организация и проведения комплексного тестирования программного обеспечения для решения задачи сортировки во внешней памяти алгоритмами прямого и естественного слияния»

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кожухова О.А.

Шифр 170582

Институт приборостроения, автоматизации и информационных технологий

Направление подготовки 09.03.04 «Программная инженерия»

Группа 71-ПГ

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ужаринский А.Ю.

Оценка: «\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_» Дата \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Орел 2020

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИМЕНИ И.С. ТУРГЕНЕВА»

Кафедра программной инженерии

УТВЕРЖДАЮ:

\_\_\_\_\_\_\_\_\_\_\_\_Зав. кафедрой

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_г.

**ЗАДАНИЕ**

**на курсовой проект**

по дисциплине «Качество и тестирование программного обеспечения»

Студент Кожухова О.А. Шифр 170582

Институт приборостроения, автоматизации и информационных технологий

Направление подготовки 09.03.04 «Программная инженерия»

Группа 71-ПГ

1 Тема курсового проекта

«Организация и проведения комплексного тестирования программного обеспечения для решения задачи сортировки во внешней памяти алгоритмами прямого и естественного слияния»

2 Срок сдачи студентом законченной работы «16» мая 2020

3 Исходные данные

Описание задачи, требования к разрабатываемому программному обеспечению.

4 Содержание курсового проекта

Проектирование и реализация программного обеспечения, тестирование программного обеспечения, оценка качества разработанного программного обеспечения.

5 Отчетный материал курсового проекта

Пояснительная записка курсового проекта; приложение, записанное на CD-диске

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ужаринский А.Ю.

Задание принял к исполнению: «16» марта 2020

Подпись студента\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc37051486)

[1 ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 6](#_Toc37051487)

[1.1 Анализ требований к разрабатываемому программному обеспечению 6](#_Toc37051488)

[1.2 Проектирование разрабатываемого программного обеспечения 10](#_Toc37051489)

[1.2.1 Общий вид архитектуры разрабатываемого программного обеспечения 10](#_Toc37051490)

[1.2.2 Модуль пользовательского интерфейса 11](#_Toc37051491)

[1.2.3 Модуль сортировки прямым слиянием 12](#_Toc37051492)

[1.2.4 Модуль сортировки естественным слиянием 14](#_Toc37051493)

[1.3 Реализация разрабатываемого программного обеспечения 15](#_Toc37051494)

[1.3.1 Реализация главного модуля программного обеспечения и модуля пользовательского интерфейса 15](#_Toc37051495)

[1.3.2 Реализация модуля сортировки прямым слиянием 16](#_Toc37051496)

[1.3.3 Реализация модуля сортировки естественным слиянием 17](#_Toc37051497)

[2 ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 18](#_Toc37051498)

[2.1 Организация процесса тестирования программного обеспечения 18](#_Toc37051499)

[2.2 Тестирование элементов разработанного программного обеспечения 20](#_Toc37051500)

[2.3 Тестирование интеграции модулей разработанного программного обеспечения 28](#_Toc37051501)

[2.4 Тестирование правильности разработанного программного обеспечения 29](#_Toc37051502)

[2.5 Системное тестирование разработанного программного обеспечения 32](#_Toc37051503)

[3 ОЦЕНКА КАЧЕСТВА РАЗРАБОТАННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 33](#_Toc37051504)

[3.1 Статическая оценка качества разработанного программного обеспечения 33](#_Toc37051505)

[3.2 Динамическая оценка качества разработанного программного обеспечения 33](#_Toc37051506)

[ЗАКЛЮЧЕНИЕ 34](#_Toc37051507)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 35](#_Toc37051508)

[Приложение 1](#_Toc37051509)  [Листинг кода главного модуля программного обеспечения 36](#_Toc37051510)

[Приложение 2](#_Toc37051511) [Листинг кода модуля сортировки прямым слиянием 38](#_Toc37051512)

[Приложение 3](#_Toc37051513) [Листинг кода модуля сортировки естественным слиянием 41](#_Toc37051514)

## ВВЕДЕНИЕ

Тестирование – это процесс исполнения программы с целью обнаружения ошибок. Исходя из данного определения, можно сделать вывод, что программа тестируется не для того, чтобы показать, что она работает, а наоборот тестирование начинается с того, что делается предположение, что в программе есть ошибки.

Тестирование *–* это один из наиболее трудоемких этапов процесса разработки программного обеспечения. Трудоемкость данного этапа составляет от 30% до 60% общей трудоемкости. Грамотно организованное тестирование позволяет добиться повышения качества разрабатываемого программного обеспечения.

В рамках данного курсового проекта необходимо реализовать и протестировать программу для решения задачи сортировки во внешней памяти алгоритмами прямого и естественного слияния.

**Целью** курсового проекта является организация и проведение комплексного тестирования для подтверждения качества разработанного программного обеспечения.

**Задачи** курсового проекта, которые необходимо выполнить для достижения поставленной цели следующие:

1) изучить задачу внешней сортировки и алгоритмы ее решения;

2) спроектировать программное обеспечение (ПО) для решения задачи внешней сортировки слиянием;

3) реализовать ПО для решения задачи сортировки во внешней памяти алгоритмами прямого и естественного слияния;

4) провести комплексное тестирование разработанного ПО;

5) оценить качество разработанного ПО.

## ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

## Анализ требований к разрабатываемому программному обеспечению

Сортировка данных является одной из основных задач программирования.

Задача сортировки данных во внешней памяти возникает тогда, когда записей больше, чем может поместиться в быстродействующей оперативной памяти. Внешняя сортировка в корне отличается от внутренней, и объясняется это тем, что время доступа к массиву на внешних носителях жестко ограничено.

В общем случае сортировка – это процесс целенаправленного перемещения элементов заданной конечной последовательности, результатом которого является последовательность, в которой элементы расположены в порядке возрастания (или убывания) их значений.

Наиболее известными алгоритмами внешних сортировок являются: сортировки слиянием (прямое, естественное и сбалансированное многопутевое слияние) и улучшенные сортировки (многофазное слияние).

В ходе курсового проекта будут рассматриваться сортировки слиянием.

Алгоритм сортировки слияниями был изобретён Джоном фон Нейманом в 1945 году. Слияние – это объединение двух или более упорядоченных отрезков в один упорядоченный более длинный отрезок.

Отрезок (или серия) – это как можно более длинный участок, представляющий собой отсортированные последовательности в конечной числовой последовательности a1, a2, …, an. Отрезки должны удовлетворять условию: (ai-1 > ai) & (∀k: i ≤ k < j: ak > ak+1) & (aj > aj+1).

Рассмотрим пример слияния двух отрезков.

Производится сравнение начальных элементов каждого из отрезков: меньший элемент выводят, а на его место становится следующий элемент данной последовательности. Данная операция повторяется до тех пор, пока один из отрезков не закончится. Оставшиеся элементы другого отрезка выводятся без изменения их порядка.

Схематично пример слияния двух отрезков изображен на рисунке 1.1.

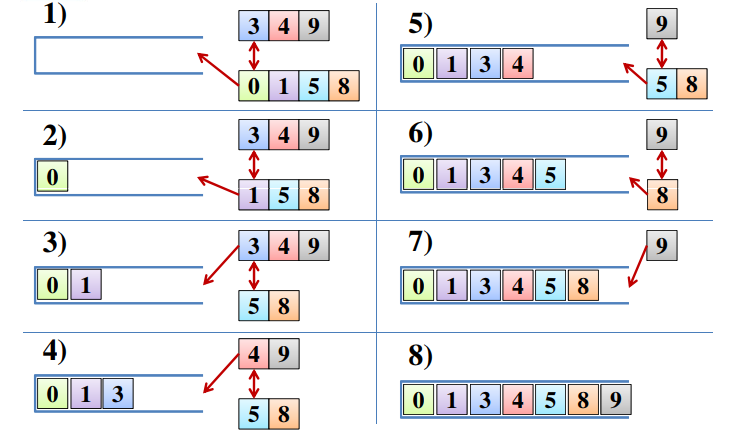


Рисунок 1.1 – Пример слияния двух отрезков

В случае наличия нескольких отрезков в последовательности необходимо обнаружение конца каждого из отрезков. Существует три способа выделения отрезков в последовательности:

1. выполнение операций сравнения. Заключается в том, что ключ выведенной записи сравнивают с ключом следующей записи. Если значение ключа новой записи меньше, то отрезок закончен;
2. использование специальной метки конца отрезка. В качестве метки, указывающей на конец отрезка, вставляют специальный ключ, значение которого не входит в диапазон значений реальных ключей;
3. использование отрезков фиксированной длины.

При использовании метода прямого слияния применяет третий способ выделения отрезков, а именно установление фиксированной длинны отрезка. Каждая начальная последовательность рассматривается как набор отрезков единичной длины.

В данном методе длина каждого отрезка фиксирована на каждом шаге, и в исходном файле отрезки имеют длину равную единице, после первого шага – длина отрезков составляет 2, после второго – 4, после k-го шага – 2k.

Схематично пример сортировки прямым слиянием приведен на рисунке 1.2.

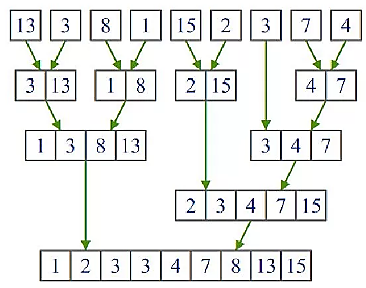


Рисунок 1.2 – Пример сортировки прямым слиянием

Метод естественного слияния – это сортировка, при которой всегда сливаются упорядоченные подпоследовательности максимальной длины.

В данном методе используются два понятия: фаза и проход (этап).

Фаза – действия по однократной обработке всего множества данных.

Проход (этап) – это наименьший подпроцесс, повторение которого составляет процесс сортировки.

Метод естественного слияния – это несбалансированная, двухфазная двухпутевая сортировка слиянием.

Схема двухфазной двухпутевой сортировки приведена на рисунке 1.3.

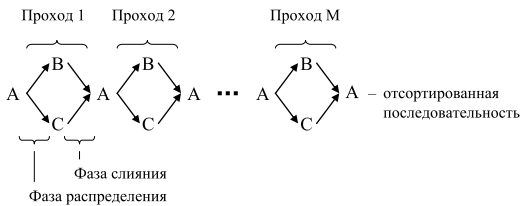


Рисунок 1.3 – Схема двухфазной двухпутевой сортировки

В приведенной схеме А – исходная последовательность элементов, В и С – вспомогательные последовательности.

Схематично пример сортировки естественным слиянием приведен на рисунке 1.4.

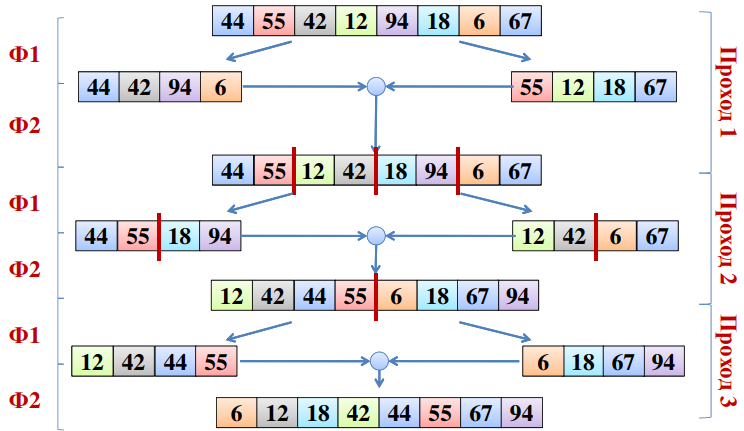


Рисунок 1.4 – Пример сортировки естественным слиянием

Разрабатываемое в ходе курсового проекта ПО должно осуществлять внешнюю сортировку слиянием выбранным пользователем методом (метод прямого слияния или метод естественного слияния). ПО должно быть модульным. Необходимо реализовать элементарный пользовательский интерфейс. При выполнении каждой сортировки необходимо засекать время ее выполнения для определенного количества записей в файле и сохранять результаты в отдельный файл.

## Проектирование разрабатываемого программного обеспечения

Проектирование программного обеспечения – это процесс определения архитектуры, интерфейсов и других характеристик программного обеспечения, а также его конечного результата.

## Общий вид архитектуры разрабатываемого программного обеспечения

Разрабатываемое в ходе курсового проекта ПО будет модульным.

Модульность – это свойство ПО, определяющее возможность его декомпозиции на ряд слабо зависящих друг от друга модулей.

Модуль – это автономно компилируемая программная единица, основное свойство которой – независимость от других модулей.

Разрабатываемое ПО будет состоять из 4 модулей:

* модуль пользовательского интерфейса, отвечающий за представление работы ПО пользователю;
* модуль сортировки прямым слиянием, реализующий алгоритм сортировки данных методом прямого слияния;
* модуль сортировки естественным слиянием, реализующий алгоритм сортировки данных методом естественного слияния;
* главный модуль, выполняющий взаимосвязь всех модулей в единое ПО.

Модуль сортировки прямым слиянием и модуль сортировки естественным слиянием вместе образуют блок логики программы.

Также для реализации ПО необходим вспомогательный пакет подключаемых библиотек.

Общий вид архитектуры разрабатываемого программного обеспечения представлен на рисунке 1.5.

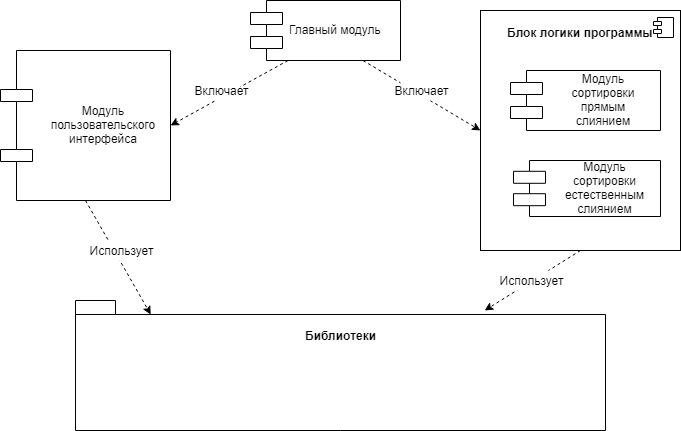


Рисунок 1.5 – Общий вид архитектуры разрабатываемого ПО

## Модуль пользовательского интерфейса

Спроектируем работу модуля пользовательского интерфейса. Результат проектирования изобразим в виде диаграммы состояний и представим на рисунке 1.6.

При запуске ПО пользователю представляется возможность выбора метода сортировки данных: метод прямого слияния или метод естественного слияния. Далее выполняется сортировка, за которую отвечает блок логики программы, о завершении которой пользователя информирует сообщение о завершении сортировки. После сообщения пользователь может выполнить новую сортировку, для чего необходимо выбрать метод сортировки, или же завершить работу ПО.

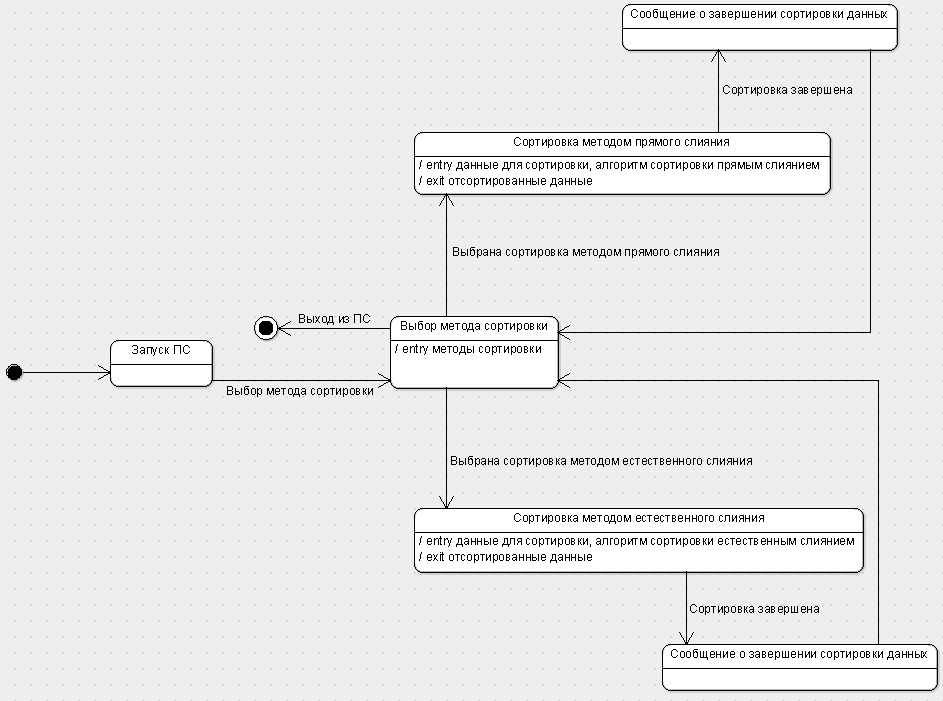


Рисунок 1.6 – Диаграмма состояний модуля пользовательского интерфейса

## Модуль сортировки прямым слиянием

Опишем алгоритм, реализуемый данным модулем.

В общем виде алгоритм прямого слияния описывает следующая последовательность действий:

1. Открыть исходный файл.
2. Равномерно распределить отрезки на 2 файла и посчитать количество отрезков.
3. Сравнить отрезки длины n, слить в отрезки длины 2\*n в исходный файл.
4. Повторять пункты 3 и 4 пока не конец одного из файлов или пока не получен один отрезок длины количества начальных отрезков

4.1 Если закончился один из файлов, дописать оставшиеся отрезки в исходный и выйти из алгоритма

Общая схема алгоритма изображена на рисунке 1.7.

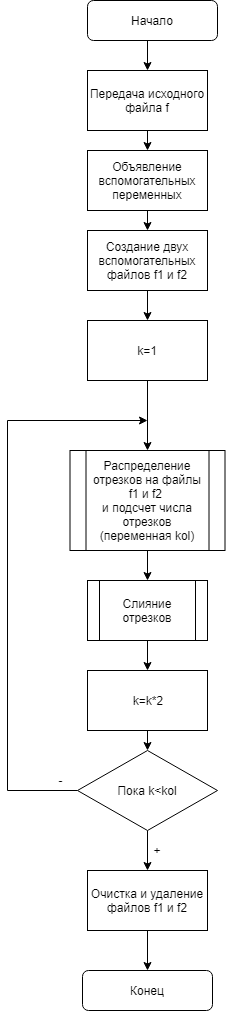


Рисунок 1.7 – Схема алгоритма прямого слияния

## Модуль сортировки естественным слиянием

Опишем алгоритм сортировки естественным слиянием, реализуемый данным модулем.

В общем виде данный алгоритм описывает следующая последовательность действий:

1. Открыть исходный файл.
2. Выполнить распределение отрезков максимальной длины на 2 файла.
3. Слить отрезки в отрезки большей длины.
4. Повторять пункты 3 и 4 пока возможно разделение на отрезки.

Общая схема алгоритма изображена на рисунке 1.8.

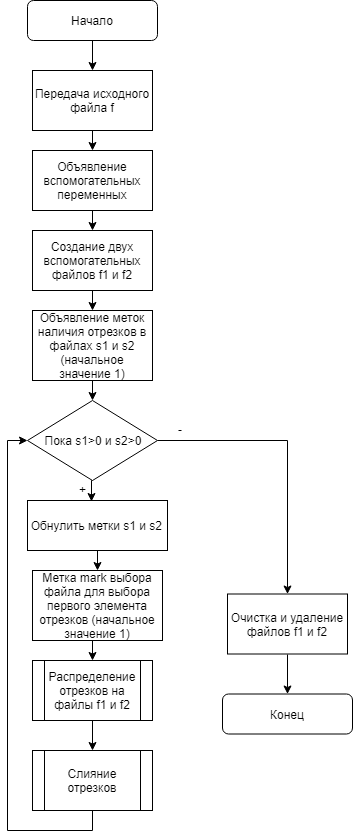


Рисунок 1.8 – Схема алгоритма естественного слияния

Процесс слияния отрезков внутри себя вызывает логическую функцию определения конца блока в файле. Данная функция получает на вход обрабатываемый файл, определяет положение указателя в файле в текущий момент, сравнивает его с символом конца блока (‘) и смещает указатель в файле на разное число байт в зависимости от совпадения с символом ‘. Функция возвращает истину, если указатель указывал на конец блока, в противном случае функция вернет значение ложь.

## Реализация разрабатываемого программного обеспечения

Для реализации программного обеспечения был выбран язык С.

## Реализация главного модуля программного обеспечения и модуля пользовательского интерфейса

Для реализации алгоритмов необходимо изначально сгенерировать файл на заданное количество записей. Максимальное количество записей в файле для выполнения выбранного в последующем алгоритма сортировки хранится в переменной int NSIZE. Генерируемый файл состоит из записей.

В исходном тексте структура данных (файл datatype.h) записи реализована следующим образом:

typedef struct \_Record {

char data[201]; // данные в записи (200 байт)

int key; // ключ записи

} Record;

Ключ записи и поля данных генерируются случайным образом.

Исходный текст функции генерации файла void generation(int a) приведен далее:

void generation(int a){

printf("N = %d\n", a);

printf("Generate file ...\n");

FILE\* f = fopen("generated.txt", "wb");

for (int n = 0; n < a; n++) {

// генерация ключа

int key = rand() % 1000;

// генерация строки в 200 байт

char data[201];

data[0] = '#';

char c = (char)(rand() % 10 + '0');

for (int j = 1; j < 199; j++) {

data[j] = c;

}

data[199] = '#';

data[200] = '\0';

fprintf(f, "%d%s", key, data);

}

fclose(f);

}

Модуль пользовательского интерфейса реализован внутри главного модуля ПО. Выбор метода сортировки реализуется при помощи конструкции switch – case. Бесконечный цикл while(exit) позволяет завершить работу программы лишь при выборе команды закрытия пользователем. Пока данное действие не выбрано пользователем возможен лишь выбор метода сортировки. При неправильном выборе пользователя на экран выводится сообщение об ошибке и вновь меню программы для выбора метода сортировки или же завершения программы.

Листинг кода главного модуля программного обеспечения приведен в приложении 1.

## Реализация модуля сортировки прямым слиянием

Модуль реализует алгоритм внешней сортировки методом прямого слияния.

Функция void Simple\_Merging\_Sort(char \*name), отвечающая за алгоритм прямого слияния, получает на вход имя файла, который необходимо отсортировать.

Цикл do – while отвечает за выполнение данной функции пока длина отрезков, на которые разбивается файл, (переменная k) не превышает количества отрезков в исходном файле (переменная kol). Подсчет количества отрезков реализуется внутри данного цикла в ходе первого распределения отрезков единичной длинны на вспомогательные файлы.

Исходный текст процесса распределения, подсчета, слияния, отрезков, представлен в листинге кода модуля сортировки прямым слиянием (файл simple\_sort.h) в приложении 2.

Удаление вспомогательных файлов, в которые шло распределение, осуществляется функцией remove(file\_name), получающей на вход имя вспомогательного файла.

## Реализация модуля сортировки естественным слиянием

Модуль реализует алгоритм внешней сортировки методом естественного слияния.

Функция void Natural\_Merging\_Sort(char \*name), отвечающая за алгоритм естественного слияния, получает на вход имя файла, который необходимо отсортировать.

Алгоритм будет выполняться до тех пор, пока в вспомогательных файлах есть отрезки для сортировки. Для этого объявляются метки наличия отрезков в вспомогательных файлах s1 и s2, которым присваиваются начальные значения для первого запуска цикла равные единице. Первым действием в цикле идет обнуление данных меток, их дальнейшее изменение осуществляется в ходе процесса распределения отрезков на вспомогательные файлы.

Программная реализация процесса распределения и слияния отрезков представлена в листинге кода модуля сортировки естественным слиянием (файл natural\_sort.h) в приложении 3.

Функция определения конца блока bool End\_Range(FILE \* f), в виде программного кода приведена ниже:

bool End\_Range(FILE \* f) {

int tmp;

tmp = fgetc(f);

if (tmp != '\'')

fseek(f, -1, SEEK\_CUR);

else

fseek(f, 1, SEEK\_CUR);

return tmp == '\'' ? true : false;

}

Удаление вспомогательных файлов, в которые шло распределение, осуществляется функцией remove(file\_name), получающей на вход имя вспомогательного файла.

## ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

## Организация процесса тестирования программного обеспечения

Процесс тестирования объединяет различные способы тестирования в спланированную последовательность шагов, которые приводят к успешному построению ПО. Классический процесс тестирования должен обеспечить проверку результатов, полученных на каждом этапе разработки. Основными этапами разработки ПО являются системный анализ, анализ требований, проектирование и кодирование. Методика тестирования ПО может быть представлена в виде разворачивающейся спирали, представляющей спиральную модель тестирования ПО, изображенной на рисунке 2.1.

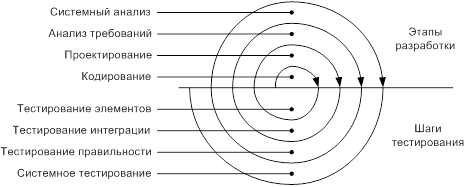


Рисунок 2.1 – Спиральная модель процесса тестирования ПО

В начале осуществляется тестирование элементов (модулей), проверяющее результаты этапа кодирования ПС. На втором шаге выполняется тестирование интеграции, ориентированное на выявление ошибок этапа проектирования ПС. На третьем обороте спирали производится тестирование правильности, проверяющее корректность этапа анализа требований к ПС. На заключительном витке спирали проводится системное тестирование, выявляющее дефекты этапа системного анализа ПС.

Охарактеризуем каждый из шагов процесса тестирования.

1. Тестирование элементов. Цель – индивидуальная проверка каждого модуля. Используются способы тестирования «белого ящика».

2. Тестирование интеграции. Цель – тестирование сборки модулей в программную систему. В основном применяют способы тестирования «черного ящика».

3. Тестирование правильности. Цель – проверить реализацию в программной системе всех функциональных и поведенческих требований, а также требования эффективности. Используются исключительно способы тестирования «черного ящика».

4. Системное тестирование. Цель – проверка правильности объединения и взаимодействия всех элементов компьютерной системы, реализации всех системных функций.

Используемый в классической модели жизненного цикла принцип увеличения детализации проекта находит свое применение и в разработке тестов. На рисунке 2.2 представлена V-модель разработки и тестирования ПО.

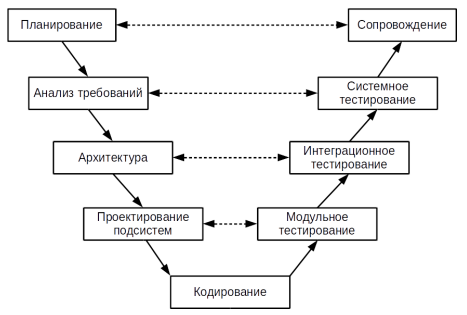


Рисунок 2.2 – V-модель разработки и тестирования ПО

Каждый этап разработки ПО сопровождается соответствующими этапами разработки тестов и выполнением тестирования. В ходе сбора и анализа требований формируются системные тесты, которые используются в ходе системного тестирования. Проект архитектуры позволяет определить механизмы взаимодействия между модулями, что приводит нас к созданию интеграционных тестов и затем к интеграционному тестированию. Проектирование модулей сопровождается модульными тестами и модульным тестированием.

Написание кода ПО приводит к последовательному усложнению тестируемого объекта: модуль → комбинация модулей → функциональное требование. Таким образом, получаются наборы тестов для всестороннего тестирования.

## Тестирование элементов разработанного программного обеспечения

Целью данного этапа тестирования является индивидуальная проверка каждого модуля разрабатываемого ПО способом тестирования «белого ящика».

При способе тестирования «белого ящика» известна внутренняя структура программы, исследуются внутренние элементы программы и связи между ними. Схематично тестирование «белого ящика» изображено на рисунке 2.3.

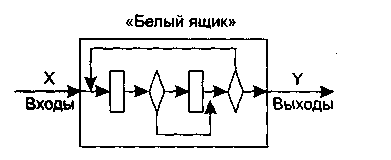


Рисунок 2.3 – Тестирование «белого ящика»

Объектом тестирования здесь является не внешнее, а внутреннее поведение программы. Проверяется корректность построения всех элементов программы и правильность их взаимодействия друг с другом.

Объектом тестирования элементов является наименьшая единица проектирования ПО – модуль. Для обнаружения ошибок в рамках модуля тестируются его важнейшие управляющие пути. Проводится структурное тестирование ПО.

В ходе выполнения курсового проекта тестирование элементов разработанного ПО будет проводится методом тестирования базового пути.

Способ тестирования базового пути дает возможность получить оценку комплексной сложности программы и использовать эту оценку для определения необходимого количества тестовых вариантов.

Тестовые варианты разрабатываются для проверки базового множества путей (маршрутов) в программе. Они гарантируют однократное выполнение каждого оператора программы при тестировании.

Метод заключается в следующей последовательности шагов:

Шаг 1. На основе текста программы формируется потоковый граф.

Шаг 2. Определяется цикломатическая сложность потокового графа.

Шаг 3. Определяется базовое множество независимых линейных путей.

Шаг 4. Подготавливаются тестовые варианты, инициирующие выполнение каждого пути.

Проведем тестирование модулей разработанного ПО.

Потоковый граф главного модуля ПО представлен на рисунке 2.4.

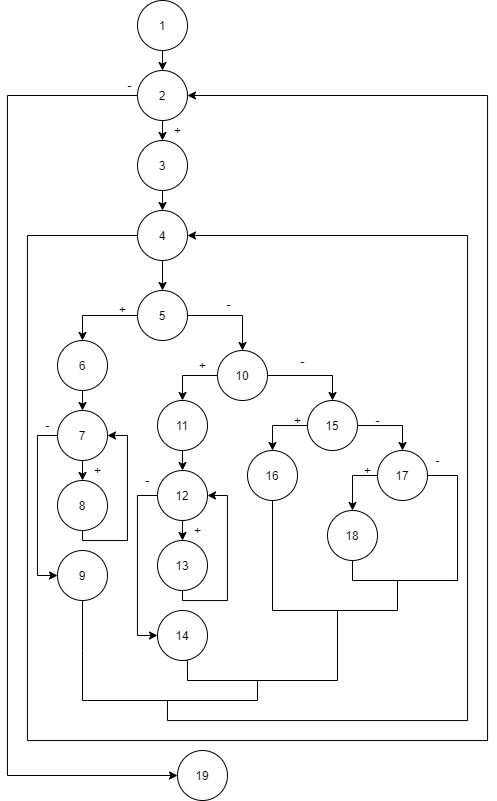


Рисунок 2.4 – Потоковый граф главного модуля

На основе построенного потокового графа и с учетом особенностей программной реализации сформировано следующее базовое множество для определения тестовых вариантов:

* + 1. 1-2-3-4-5-6-7-8-9-19
    2. 1-2-3-4-5-6-7-9-19
    3. 1-2-3-4-5-10-11-12-13-14-19
    4. 1-2-3-4-5-10-11-12-14-19
    5. 1-2-3-4-5-10-15-16-19
    6. 1-2-3-4-5-10-15-17-18-19

Тестовые варианты для сформированного базового множества следующие:

1. ИД: выбрана сортировка прямым слиянием, количество записей в файле не превышает размера файла (input = 1, i < NSIZE (i = 2000, NSIZE = 20001))

ОЖ.РЕЗ.: сообщение о завершении сортировки прямым слиянием, отсортированный файл с результатами и файл с результатами замера времени («Simple Sort finished», отсортированный файл generated, файл simple\_sort\_result)

1. ИД: выбрана сортировка прямым слиянием, количество записей в файле превышает размера файла (input = 1, i > NSIZE (i = 21000, NSIZE = 20001))

ОЖ.РЕЗ.: сообщение о завершении сортировки прямым слиянием, файл с результатами замера времени пуст («Simple Sort finished», пустой файл simple\_sort\_result)

1. ИД: выбрана сортировка естественным слиянием, количество записей в файле не превышает размера файла (input = 2, i < NSIZE) (i = 2000, NSIZE = 20001)

ОЖ.РЕЗ.: сообщение о завершении сортировки естественным слиянием, отсортированный файл с результатами и файл с результатами замера времени («Natural Sort finished», отсортированный файл generated, файл natural\_sort\_result)

1. ИД: выбрана сортировка естественным слиянием, количество записей в файле превышает размера файла (input = 2, i > NSIZE (i = 21000, NSIZE = 20001))

ОЖ.РЕЗ.: сообщение о завершении сортировки естественным слиянием, файл с результатами замера времени пуст («Natural Sort finished», пустой файл natural\_sort\_result)

1. ИД: выбрано корректное завершение работы программы (input = 0)

ОЖ.РЕЗ.: корректное завершение работы программы

1. ИД: введен некорректный символ выбора пункта меню (input != 0, input != 1, input != 2 (input = 5))

ОЖ.РЕЗ.: сообщение об ошибке ввода, вывод меню и ожидание повторного выбора пункта меню (Unable choice., Choose action: 1. Simple sort 2. Natural sort 0. Exit Choice:)

Потоковый граф функции генерации файла представлен на рисунке 2.5.

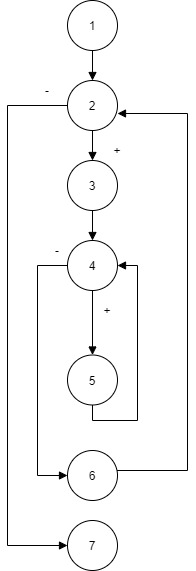


Рисунок 2.5 – Потоковый граф функции генерации файла

На основе построенного потокового графа и с учетом особенностей программной реализации сформировано следующее базовое множество для определения тестовых вариантов:

1. 1-2-7
2. 1-2-3-4-5-6-7

Тестовые варианты для сформированного базового множества следующие:

1. ИД: необходимое количество записей в файле для генерации превышает текущую генерируемую запись (n > a (n = 1001, a = 1000))

ОЖ.РЕЗ.: пустой сгенерированный файл (пустой файл generated)

1. ИД: необходимое количество записей в файле для генерации не превышает текущую генерируемую запись (n < a (n = 101, a = 1000))

ОЖ.РЕЗ.: корректно сгенерированный файл на a записей (файл generated, содержащий 1000 записей)

Потоковый граф модуля сортировки прямым слиянием представлен на рисунке 2.6.

На основе построенного потокового графа и с учетом особенностей программной реализации сформировано следующее базовое множество для определения тестовых вариантов:

1. 1-2-3-…-33-34
2. 1-2-3-…-33-2-3-…-33-…-34

Тестовые варианты для сформированного базового множества следующие:

1. ИД: длина отрезков, на которые разбивается исходный файл, превышает количество отрезков в исходном файле (k > kol (k = 101, kol = 100))

ОЖ.РЕЗ.: ошибка выполнения алгоритма сортировки файла, исходный неотсортированный файл

1. ИД: длина отрезков, на которые разбивается исходный файл, не превышает количество отрезков в исходном файле (k < kol (k = 90, kol = 100))

ОЖ.РЕЗ.: корректно выполненный алгоритм сортировки, отсортированный файл

Потоковый граф модуля сортировки естественным слиянием представлен на рисунке 2.7.

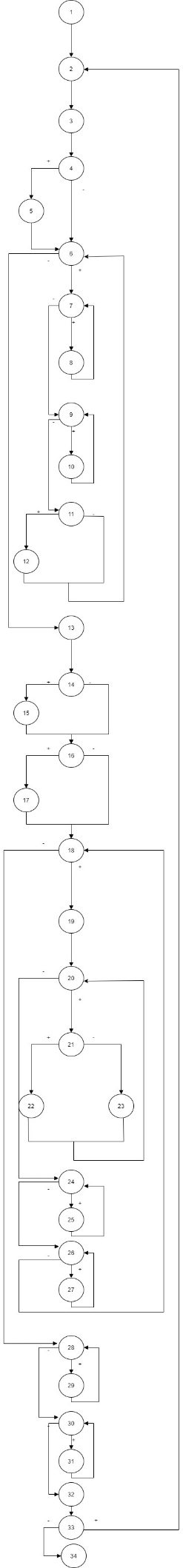


Рисунок 2.6 – Потоковый граф модуля сортировки прямым слиянием

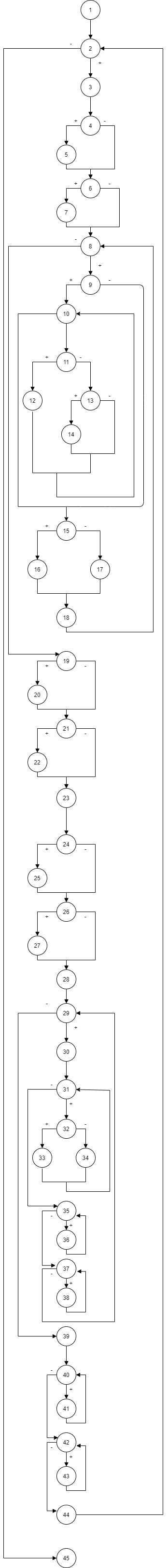


Рисунок 2.7 – Потоковый граф модуля сортировки естественным слиянием

На основе построенного потокового графа и с учетом особенностей программной реализации сформировано следующее базовое множество для определения тестовых вариантов:

1. 1-2-3-…-45
2. 1-2-45

Тестовые варианты для сформированного базового множества следующие:

1. ИД: в любом из двух вспомогательных файлах отсутствуют отрезки для осуществления сортировки (s1 = 0 / s2 = 0)

ОЖ.РЕЗ.: ошибка выполнения алгоритма сортировки файла, исходный неотсортированный файл

1. ИД: в вспомогательных файлах имеются отрезки для осуществления сортировки (s1 > 0 (s1 = 3), s2 > 0 (s2 = 4))

ОЖ.РЕЗ.: корректно выполненный алгоритм сортировки, отсортированный файл

## Тестирование интеграции модулей разработанного программного обеспечения

Целью данного этапа тестирования является проверка корректности сборки модулей в единое программное обеспечение.

Объектом тестирования на данном этапе является сборка программного обеспечения.

На данном этапе тестирования проверяются такие виды ошибок как:

1. потеря данных при прохождении интерфейса;
2. недопустимое сцепление модулей;
3. некорректная работа с глобальными данными.

Основными стратегиями интеграции модулей являются:

* нисходящий способ, при котором модули объединяются движением сверху вниз по управляющей иерархии, начиная от главного управляющего модуля. Главным достоинством данного способа является факт выявления ошибок в главной управляющей части в первую очередь. Недостатком способа является факт возникновения трудностей в ситуациях, когда для тестирования верхнего модуля необходимы результаты обработки нижних модулей.
* восходящий способ, при котором модули подключаются движением снизу вверх. При этом подчиненные модули всегда доступны и отпадает необходимость разработки заглушек. Главным достоинством является факт упрощения разработки тестовых вариантов. Недостатком данного способа является факт того, что система не существует как единое целое до тех пор, пока не добавлен последний модуль.
* комбинированный способ, при котором для верхних модулей применяется нисходящая стратегия, для нижних – восходящая, а также выделяются критические модули, которые должны тестироваться как можно раньше и не один раз.

Для проведения тестирования определим порядок интеграции модулей. Так как для тестирования главного модуля необходимы модули, реализующие основную логику программы, на первом этапе тестируются модуль сортировки прямым слиянием и модуль сортировки естественным слиянием, объединенные в блок логики программы. Далее проведено тестирование части главного модуля, а именно тестирование модуля пользовательского интерфейса, последним этапом является подключение главного модуля и тестирование ПО в целом. Таким образом, в ходе курсового проекта тестирование интеграции модулей проводится восходящим способом.

## Тестирование правильности разработанного программного обеспечения

Целью данного этапа тестирования является проверка и подтверждение реализации в ПО функциональных требований способом тестирования «черного ящика».

При способе тестирования «черного ящика» известны функции программы, исследуется работа каждой функции на всей области определения. Схематично тестирование «черного ящика» изображено на рисунке 2.8.

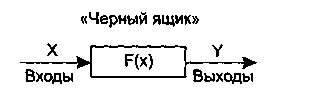


Рисунок 2.8 – Тестирование «черного ящика»

При тестировании «черного ящика» рассматриваются системные характеристики программ, игнорируется их внутренняя логическая структура.

Объектом тестирования правильности на данном этапе является единое программное обеспечение.

В ходе выполнения курсового проекта тестирование правильности разработанного ПО будет проводится методом диаграмм причин-следствий.

Способ тестирования с использование диаграмм причин-следствий обеспечивает формальное выведение высокорезультативных тестовых вариантов, основанное на анализе причинно-следственных связей. Причина – это отдельное входное условие или класс эквивалентности. Следствие – выходное условие или действие системы.

Метод заключается в следующей последовательности шагов:

Шаг 1. Выделение групп причин и следствий.

Шаг 2. Присвоение идентификаторов каждой причине и каждому следствию.

Шаг 3. Построение графа причинно-следственных связей.

Шаг 4. Преобразование графа в таблицу решений.

Шаг 5. Преобразование столбцов решений в тестовые варианты.

Проведем тестирование правильности разработанного ПО.

Выделим следующие причины и присвоим им идентификаторы:

1 – сортировка прямым слиянием

2 – сортировка естественным слиянием

3 – выход из программы

4 – некорректный ввод

Аналогичным образом выделим и присвоим идентификаторы следствиям:

101 – отсортированный файл и файл результатов замера времени

102 – корректное завершение работы программы

103 – сообщение об ошибке

На рисунке 2.9 изобразим граф причинно-следственных связей.

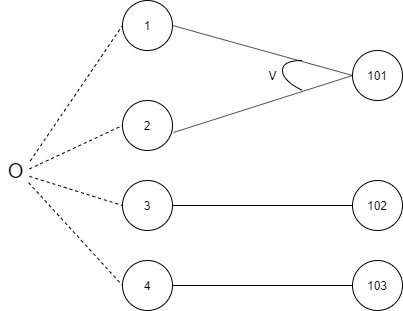


Рисунок 2.9 – Граф причин-следствий

Преобразуем граф в таблицу решений и приведем ее в таблице 1.

Таблица 1 – Таблица решений

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Номер столбца | | 1 | 2 | 3 | 4 |
| Причины | 1 | 1 | 0 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 |
| 3 | 0 | 0 | 1 | 0 |
| 4 | 0 | 0 | 0 | 1 |
| Следствия | 101 | 1 | 1 | 0 | 0 |
| 102 | 0 | 0 | 1 | 0 |
| 103 | 0 | 0 | 0 | 1 |

На основе таблицы решений сформированы следующие тестовые варианты:

1. ИД: выбран способ сортировки прямым слиянием

ОЖ.РЕЗ.: отсортированный файл и файл результатов замера времени

1. ИД: выбран способ сортировки естественным слиянием

ОЖ.РЕЗ.: отсортированный файл и файл результатов замера времени

1. ИД: выбран выход из программы

ОЖ.РЕЗ.: корректное завершение работы программы

1. ИД: осуществлен некорректный ввод

ОЖ.РЕЗ.: сообщение об ошибке

## Системное тестирование разработанного программного обеспечения

*В зависимости от специфики разработанной программы провести один или несколько из видов системного тестирования:*

*1. Тестирование восстановления (время восстановления, правильность повторной инициализации, восстановление данных и др.)*

*2. Тестирование безопасности (проверка функционирования защитных механизмов системы в случае проникновения).*

*3. Стрессовое тестирование (тестовые варианты направлены на оценку стабильности работы ПС при «ненормальных» нагрузках: по частоте запросу, объему данных). Частный случай – тестирование чувствительности.*

*4. Тестирование производительности (проверяется скорость работы ПО в компьютерной системе)*.

## ОЦЕНКА КАЧЕСТВА РАЗРАБОТАННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

## Статическая оценка качества разработанного программного обеспечения

*Рассчитать основные метрики разработанного кода. Можно использовать метрики Холстеда, как на лабораторной. Можно любые другие метрики кода, которые найдёте в интернете. К каждой рассчитываемой характеристике должно быть пояснение – что она значит и как рассчитывается. В конце вывод по результатам рассчёта. Вывод должен быть по существу полученных значений, а не в общем констатация факта, что характеристики были рассчитаны.*

## Динамическая оценка качества разработанного программного обеспечения

*Провести любой из видов нагрузочного тестирования: тестирование производительности, стабильности, стрессовое тестирование или тестирование восстановления. Для этого разработать профиль нагрузки, смоделировать эту нагрузку и посчитать полученные показатели и сделать выводы. Профиль нагрузки включает определение пользователей системы, их количества, наиболее часто выполняемых ими операций*

## ЗАКЛЮЧЕНИЕ

*В заключении делается вывод об итогах тестирования и получееных в результате оценках качества и надёжности разработанного программного обеспечения, приводится модель надёжности ПО. Вероятность возникновения ошибки или сбоя, время наработки на отказ и т.п. В конце делается общий вывод о том, что цель и задачи курсового проекта достигнуты*

## СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1*. Ахо А. Структуры данных и алгоритмы / А. Ахо, Д. Хопкрофт, Д. Ульман. – М.: Издательский дом «Вильямс», 2000. – 384 с.*

*2. Кнут Д. Искусство программирования для ЭВМ, т. 3. – С.-П.: Издательский дом «Вильямс», 2000. – 800 с.*

*3. Вирт Н. Алгоритмы и структуры данных. – М.: Мир, 1989. – 360 с.*

*4. Макконенелл Дж. Анализ алгоритмов. Вводный курс. – М.: Техно-сфера, 2002. – 304 с.*

## Приложение 1

## Листинг кода главного модуля программного обеспечения

Файл datatype.h.

#ifndef \_Record\_H

#define \_Record\_H

// тип записи в файле

typedef struct \_Record {

char data[201]; // данные в записи (200 байт)

int key; // ключ записи

} Record;

#endif

Файл main.c.

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <stddef.h>

#include <ctype.h>

#include <string.h>

#include <stdbool.h>

#include <time.h>

#include "datatype.h"

#include "simple\_sort.h"

#include "natural\_sort.h"

int NSIZE = 20001;

//функция генерации файла

void generation(int a){

//1

printf("N = %d\n", a);

printf("Generate file ...\n");

FILE\* f = fopen("generated.txt", "wb");

//2

for (int n = 0; n < a; n++) {

//3

// генерация ключа

int key = rand() % 1000;

// генерация строки в 200 байт

char data[201];

data[0] = '#';

char c = (char)(rand() % 10 + '0');

//4

for (int j = 1; j < 199; j++) {

//5

data[j] = c;

}

//6

data[199] = '#';

data[200] = '\0';

fprintf(f, "%d%s", key, data);

}

//7

fclose(f);

}

int main(){

//1

int exit = 1;

srand((unsigned)time(NULL));

//2

while (exit){

//3

printf( "Choose action:\n1. Simple sort\n2. Natural sort\n0. Exit\n" );

printf( "Choice: " );

int input;

scanf( "%d", &input);

//4

switch (input) {

//5

case 1:

//6

printf("\nSimple Sort\n\n");

FILE\* fresult1 = fopen("simple\_sort\_result.txt", "wt");

//7

for (int i = 1000; i < NSIZE; i+=1000) {

//8

generation(i);

printf("Testing ...\n");

long t0 = (long)time(NULL);

Simple\_Merging\_Sort("generated.txt");

long t1 = (long)time(NULL);

double tresult = (t1 - t0);

fprintf(fresult1, "N: %d TIME: %.3lf\n\n", i, tresult);

}

//9

fclose(fresult1);

printf("\nSimple Sort finished\n\n");

break;

//10

case 2:

//11

printf("\nNatural Sort\n\n");

FILE\* fresult2 = fopen("natural\_sort\_result.txt", "wt");

//12

for (int i = 1000; i < NSIZE; i+=1000) {

//13

generation(i);

printf("Testing ...\n");

long t0 = (long)time(NULL);

char \*name1 = "generated.txt";

Natural\_Merging\_Sort(name1);

long t1 = (long)time(NULL);

double tresult = (t1 - t0);

fprintf(fresult2, "N: %d TIME: %.3lf\n\n", i, tresult);

}

//14

fclose(fresult2);

printf("\nNatural Sort finished\n\n");

break;

//15

case 0:

//16

exit = 0;

break;

//17

default:

//18

printf( "\nUnable choice.\n\n" );

break;

}

}

//19

return 0;

}

## Приложение 2

## Листинг кода модуля сортировки прямым слиянием

Файл simple\_sort.h.

#ifndef Simple\_Merging\_Sort\_H

#define Simple\_Merging\_Sort\_H

//функция сортировки прямым слиянием

void Simple\_Merging\_Sort(char \*name) {

//1

int a1, a2, k, i, j, kol;

char d1[201], d2[201];

FILE \*f, \*f1, \*f2;

kol = 0;

k = 1;

//2

do{

//3

f = fopen(name, "rb");

f1 = fopen("smsort\_1", "wb");

f2 = fopen("smsort\_2", "wb");

//4

if (!feof(f)) {

//5

fscanf(f, "%d", &a1);

fscanf(f, "%200s", d1);

}

//6

while (!feof(f)) {

//7

for (i = 0; i < k && !feof(f); i++) {

//8

fprintf(f1, "%d ", a1);

fprintf(f1, "%s ", d1);

fscanf(f, "%d", &a1);

fscanf(f, "%200s", d1);

}

//9

for (j = 0; j < k && !feof(f); j++) {

//10

fprintf(f2, "%d ", a1);

fprintf(f2, "%s ", d1);

fscanf(f, "%d", &a1);

fscanf(f, "%200s", d1);

}

//11

if (k == 1){

//12

kol += 2;

}

}

//13

fclose(f2);

fclose(f1);

fclose(f);

f = fopen(name, "wb");

f1 = fopen("smsort\_1", "rb");

f2 = fopen("smsort\_2", "rb");

//14

if (!feof(f1)) {

//15

fscanf(f1, "%d", &a1);

fscanf(f1, "%200s", d1);

}

//16

if (!feof(f2)) {

//17

fscanf(f2, "%d", &a2);

fscanf(f2, "%200s", d2);

}

//18

while (!feof(f1) && !feof(f2)) {

//19

i = 0;

j = 0;

//20

while (i < k && j < k && !feof(f1) && !feof(f2)) {

//21

if (a1 < a2) {

//22

fprintf(f, "%d", a1);

fprintf(f, "%s", d1);

fscanf(f1, "%d", &a1);

fscanf(f1, "%200s", d1);

i++;

}

//23

else {

fprintf(f, "%d", a2);

fprintf(f, "%s", d2);

fscanf(f2, "%d", &a2);

fscanf(f2, "%200s", d2);

j++;

}

}

//24

while (i < k && !feof(f1)) {

//25

fprintf(f, "%d", a1);

fprintf(f, "%s", d1);

fscanf(f1, "%d", &a1);

fscanf(f1, "%200s", d1);

i++;

}

//26

while (j < k && !feof(f2)) {

//27

fprintf(f, "%d", a2);

fprintf(f, "%s", d2);

fscanf(f2, "%d", &a2);

fscanf(f2, "%200s", d2);

j++;

}

}

//28

while (!feof(f1)) {

//29

fprintf(f, "%d", a1);

fprintf(f, "%s", d1);

fscanf(f1, "%d", &a1);

fscanf(f1, "%200s", d1);

}

//30

while (!feof(f2)) {

//31

fprintf(f, "%d", a2);

fprintf(f, "%s", d2);

fscanf(f2, "%d", &a2);

fscanf(f2, "%200s", d2);

}

//32

fclose(f2);

fclose(f1);

fclose(f);

k \*= 2;

}

//33

while (k < kol);

//34

remove("smsort\_1");

remove("smsort\_2");

}

#endif

## Приложение 3

## Листинг кода модуля сортировки естественным слиянием

Файл natural\_sort.h.

#ifndef Natural\_Merging\_Sort\_H

#define Natural\_Merging\_Sort\_H

//функция определения конца блока

bool End\_Range(FILE \* f) {

int tmp;

tmp = fgetc(f);

if (tmp != '\'')

fseek(f, -1, SEEK\_CUR);

else

fseek(f, 1, SEEK\_CUR);

return tmp == '\'' ? true : false;

}

//функция сортировки естественным слиянием

void Natural\_Merging\_Sort(char \*name) {

//1

int s1, s2, a1, a2, mark;

char d1[201], d2[201];

FILE \*f, \*f1, \*f2;

s1 = 1;

s2 = 1;

//2

while (s1 > 0 && s2 > 0) {

//3

s1 = 0;

s2 = 0;

mark = 1;

f = fopen(name, "rb");

f1 = fopen("nmsort\_1", "wb");

f2 = fopen("nmsort\_2", "wb");

fscanf(f, "%d", &a1);

fscanf(f, "%200s", d1);

//4

if (!feof(f)) {

//5

fprintf(f1, "%d", a1);

fprintf(f1, "%s", d1);

}

//6

if (!feof(f)) {

//7

fscanf(f, "%d", &a2);

fscanf(f, "%200s", d2);

}

//8

while (!feof(f)) {

//9

if (a2 < a1) {

//10

switch (mark) {

//11

case 1: {/\*12\*/fprintf(f1, "\'"); mark = 2; s1++; break;}

//13

case 2: {/\*14\*/fprintf(f2, "\'"); mark = 1; s2++; break;}

}

}

//15

if (mark == 1) {

//16

fprintf(f1, "%d", a2);

fprintf(f1, "%s", d2);

s1++;

}

//17

else {

fprintf(f2, "%d", a2);

fprintf(f2, "%s", d2);

s2++;

}

//18

a1 = a2;

fscanf(f, "%d", &a2);

fscanf(f, "%200s", d2);

}

//19

if (s2 > 0 && mark == 2) { /\*20\*/fprintf(f2, "\'"); }

//21

if (s1 > 0 && mark == 1) { /\*22\*/fprintf(f1, "\'"); }

//23

fclose(f2);

fclose(f1);

fclose(f);

f = fopen(name, "wb");

f1 = fopen("nmsort\_1", "rb");

f2 = fopen("nmsort\_2", "rb");

//24

if (!feof(f1)) {

//25

fscanf(f1, "%d", &a1);

fscanf(f1, "%200s", d1);

}

//26

if (!feof(f2)) {

//27

fscanf(f2, "%d", &a2);

fscanf(f2, "%200s", d2);

}

//28

bool file1, file2;

//29

while (!feof(f1) && !feof(f2)) {

//30

file1 = file2 = false;

//31

while (!file1 && !file2) {

//32

if (a1 <= a2) {

//33

fprintf(f, "%d", a1);

fprintf(f, "%s", d1);

file1 = End\_Range(f1);

fscanf(f1, "%d", &a1);

fscanf(f1, "%200s", d1);

}

//34

else {

fprintf(f, "%d", a2);

fprintf(f, "%s", d2);

file2 = End\_Range(f2);

fscanf(f2, "%d", &a2);

fscanf(f2, "%200s", d2);

}

}

//35

while (!file1) {

//36

printf(f, "%d", a1);

fprintf(f, "%s", d1);

file1 = End\_Range(f1);

fscanf(f1, "%d", &a1);

fscanf(f1, "%200s", d1);

}

//37

while (!file2) {

//38

fprintf(f, "%d", a2);

fprintf(f, "%s", d2);

file2 = End\_Range(f2);

fscanf(f2, "%d", &a2);

fscanf(f2, "%200s", d2);

}

}

//39

file1 = file2 = false;

//40

while (!file1 && !feof(f1)) {

//41

fprintf(f, "%d", a1);

fprintf(f, "%s", d1);

file1 = End\_Range(f1);

fscanf(f1, "%d", &a1);

fscanf(f1, "%200s", d1);

}

//42

while (!file2 && !feof(f2)) {

//43

fprintf(f, "%d", a2);

fprintf(f, "%s", d2);

file2 = End\_Range(f2);

fscanf(f2, "%d", &a2);

fscanf(f2, "%200s", d2);

}

//44

fclose(f2);

fclose(f1);

fclose(f);

}

//45

remove("nmsort\_1");

remove("nmsort\_2");

}

#endif