Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический университет имени П.О. Сухого»

Факультет автоматизированных и информационных систем

Кафедра «Информатика»

Г. П. Косинов

АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ

Практикум
по выполнению лабораторных работ
по одноименной дисциплине для студентов специальности
1-40 04 01 «Информатика и технологии программирования»
для дневной формы обучения

Гомель 2021

УДК
ББК

Рекомендовано к изданию кафедрой «Информатика»
УО «ГГТУ им. П.О. Сухого»
(протокол № 11 от 29.03.2021 г.)

Рецензент: профессор кафедры «Информационные технологии» УО «Гомельский государственный технический университет им. П.О. Сухого», к.ф.-м.н., доцент Е.Г. Стародубцев.

Косинов Г.П.

Алгоритмы и структуры данных: Практикум для выполнения лабораторных работ по одноим. курсу для студентов специальности 1-40 04 01 «Информатика и технологии программирования» / Г. П. Косинов; Гомел. гос. техн. ун-т им. П. О. Сухого. – Гомель: ГГТУ им. П. О. Сухого, 2021. – с.

Данный практикум написан в соответствии с требованиями, предъявленными к оформлению методических пособий доступным языком, и содержит множество примеров Рассмотрены вопросы создания и обработки динамических структур данных, сортировки, архивации, хэширования, алгоритмам на графах.

Издание адресовано студентам специальности 1-40 04 01 «Информатика и технологии программирования», изучающим дисциплину «Алгоритмы и структуры данных».

УДК
ББК

СОДЕРЖАНИЕ

1 ЛАБОРАТОРНАЯ РАБОТА № 1. ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ: ОДНОНАПРАВЛЕННЫЕ И ДВУНАПРАВЛЕННЫЕ СПИСКИ

1.1 Цель работы

Цель работы: изучить понятия, классификацию и объявления списков, особенности доступа к данным и работу с памятью при использовании однонаправленных и *двунаправленных* списков, научиться решать задачи с использованием списков на языке C++.

При выполнении лабораторной работы для каждого задания требуется написать программу на языке C++, в которой выполнено формирование однонаправленного или *двунаправленного* списка в соответствии с постановкой задачи, ввод данных элементов списка с учетом типа информационного поля, их обработка и *вывод* на экран в указанном формате. Для хранения данных списков следует использовать ресурсы динамической памяти. Ввод данных осуществляется с клавиатуры с учетом требований к входным данным, содержащихся в постановке задачи. Ограничениями на *входные данные* являются максимальный размер строковых данных, диапазоны числовых типов полей структуры и допустимый размер области динамической памяти в языке C++.

1.2 Краткие теоретические сведения

Как правило, динамические переменные организуются в списковые структуры данных, элементы которых имеют тип *struct*. Для адресации элементов в структуру включается указатель (адресное поле) на область размещения следующего элемента.

Такой список называют *однонаправленным (односвязным).* Если добавить в каждый элемент ссылку на предыдущий, получится *двунаправленный список (двусвязный),* если последний элемент связать указателем с первым, получится *кольцевой список.*

Например, пусть необходимо создать линейный список, содержащий целые числа, тогда каждый элемент списка должен иметь информационную (*info*) часть, в которой будут находиться данные, и адресную часть (*p*), в которой будут размещаться указатели связей, т.е. элемент такого списка имеет вид:

```
struct Spis {
int info;
Spis *p;
} ;
```

Каждый элемент списка содержит *ключ,* идентифицирующий этот элемент. Ключ обычно бывает либо целым числом, либо строкой и является частью поля данных. В качестве ключа в процессе работы со списком могут выступать разные части поля данных.

Над списками можно выполнять следующие *операции:*

– начальное формирование списка (создание первого элемента);
– добавление элемента в список;
– обработка (чтение, удаление и т.п.) элемента с заданным ключом;
– вставка элемента в заданное место списка (до или после элемента с заданным ключом);
– упорядочивание списка по ключу.

Если программа состоит из функций, решающих вышеперечисленные задачи, то необходимо соблюдать следующие требования:

– все параметры, не изменяемые внутри функций, должны передаваться с модификатором *const*;
– указатели, которые могут изменяться, передаются по адресу. Например, при удалении из списка последнего элемента, измененный указатель на конец списка требует корректировки, т.е. передачи в точку вызова.

Стек – упорядоченный набор данных, в котором размещение новых элементов и удаление существующих производится только с одного его конца, который называют вершиной стека, т.е. стек – это список с одной точкой доступа к его элементам. Стек – структура типа ***LIFO*** (*Last In, First Out*) – последним вошел, первым выйдет. Максимальное число элементов стека ограничивается, т.е. по мере вталкивания в стек новых элементов память под него должна динамически запрашиваться и освобождаться также динамически при удалении элемента из стека. Таким образом, стек – динамическая структура данных, состоящая из переменного числа элементов одинакового типа. Операции, выполняемые над стеком, имеют специальные названия:

push – добавление элемента в стек (вталкивание);

pop – выталкивание (извлечение) элемента из стека, верхний элемент стека удаляется (не может применяться к пустому стеку).

Очередь – упорядоченный набор данных (структура данных), в котором в отличие от стека извлечение данных происходит из начала цепочки, а добавление данных – в конец этой цепочки.

Очередь также называют структурой данных, организованной по принципу *FIFO* (*First In, First Out*) – первый вошел (первый созданный элемент очереди), первый вышел.

При работе с очередью обычно помимо текущего указателя используют еще два указателя, первый указатель устанавливается на начало очереди, а второй – на ее конец.

Основные операции с очередью следующие:

– формирование очереди;
– добавление нового элемента в конец очереди;
– удаление элемента из начала очереди.

1.3 Задания для выполнения лабораторной работы

Написать две программы по созданию, просмотру, добавлению, удалению, сортировке и решению поставленной задачи для линейного списка (список, стек и/или очередь). В первой в качестве структуры данных использовать линейный список с указателем на следующий элемент, во второй – динамический массив с дополнительным полем – номер следующей записи (или двумя). Обязательно разобраться с сортировкой списков, а во второй программе - с удалением элементов. В программе нельзя использовать дополнительные массивы. В программе не использовать STL и шаблоны.

В таблице 1.1 приведены варианты для выполнения индивидуального задания.

Таблица 1.1 – Варианты индивидуальных заданий

Вариант	Задание
1	Создать однонаправленный список из случайных целых чисел и преобразовать его в два списка. Первый должен содержать только положительные числа, а второй – только отрицательные. Затем удалить элементы по модулю меньшие 5

Продолжение таблицы 1.1

2	Для ряда натуральных чисел длиной N>2, представленного в виде списка, построить последовательность: A_1*A_n, A_2*A_{n-1}, ..., которую записать в структуру типа очередь
3	Описать функцию, которая вставляет в однонаправленный список L за первым вхождением элемента E все элементы списка L1, если E входит в L.
4	Из двух списков сформировать новый следующим образом: первый элемент первого списка, второй – второго, третий - первого и т.д.
5	Из двух однонаправленных списков сформировать новый следующим образом: сперва записать четные по значению элементы первого списка, затем – четные по значению элементы второго списка.
6	Из однонаправленного списка сформировать новый, состоящий из элементов, перед которыми и после которых, стоят одинаковые значения.
7	Если два списка идентичны, то сформировать однонаправленный список из элементов первого списка следующим образом – первый элемент, затем последний; второй, затем предпоследний …
8	Если однонаправленный список L1 входит в однонаправленный список L2, то удалить элементы, входящие в список L2, равные элементам из L1
9	Составить список из N чисел. Проверить его на наличие одинаковых элементов. Одинаковые элементы перенести в конец списка.
10	Для списка натуральных чисел длиной N>2, построить новый по следующему принципу: A_1+A_3, A_2+A_4, . . . $A_{n-2}+A_n$
11	В однонаправленном списке переместить минимальный элемент в начало списка, а максимальный – в конец.
12	Из двух однонаправленных списков сформировать новый следующим образом: сперва записать четные по индексу элементы первого списка, затем – четные по индексу элементы второго списка.

Продолжение таблицы 1.1

13	Представить автотрассу в виде списка, элементы которого содержат информацию о названии населенных пунктов и расстоянии между ними. По заданному расстоянию выбрать названия двух населенных пунктов, расстояние между которыми минимально отличается от заданного.
14	Создать список и разместить в нем в случайном порядке данные о колоде игральных карт. Разработать программу, проводящую перемешивание колоды карт путем сдвига ее частей. Результаты перемешивания вывести на печать. (колода – маленькая))))))
15	Произвести проверку соблюдения баланса скобок вида '{', '}' в тексте программы. Использовать системный стек.
16	Составить однонаправленный список из N чисел. Проверить его на наличие одинаковых элементов. Одинаковые элементы удалить и перенести в новый список.
17	Даны однонаправленные списки A и B. Удалить из A числа, имеющиеся в B, добавить в конец списка A числа списка B, которых нет в A.
18	Создать однонаправленный список из случайных целых чисел. Удалить из этого списка все элементы, находящиеся между максимальным и минимальным элементами. Сформировать новый список из удаляемых элементов
19	Создать два однонаправленных списка из случайных целых чисел. Вместо элементов первого списка, заключенных между максимальным и минимальным элементами, вставить второй список.
20	Создать список из случайных положительных и отрицательных целых чисел. Образовать из него два однонаправленных списка, первый должен содержать значения до первого отрицательного числа, а второй – после первого положительного.
21	Создать два однонаправленных списка из случайных целых чисел. В первом найти максимальный элемент и за ним вставить элементы второго.

Продолжение таблицы 1.1

22	Создать двунаправленный список из случайных чисел. Определить среднее арифметическое значение всех элементов. Сформировать два новых списка, поместив в них значения меньшие среднего арифметического и больше соответственно.
23	Даны два списка, значения в которых отсортированы по возрастанию. Сформировать третий список из первых двух, значения в котором также будут отсортированы
24	Создать два однонаправленных списка из случайных целых чисел. В первом найти минимальный элемент и удалить элементы, идущие за ним. Во втором найти максимальный элемент и удалить элементы, идущие перед ним.
25	Создать два однонаправленных списка из целых чисел. Сформировать третий список из элементов, которые есть в обоих.
26	Создать три однонаправленных списка из целых чисел. Сформировать четвертый список из элементов, которые есть только в одном из трех списков, но нет в двух других.
27	Составить однонаправленный список из N чисел. Сформировать два новых списка из элементов, стоящих до первого нуля исходного списка и после соответственно.
28	Составить однонаправленный список из N чисел. Удалить из него минимальный элемент. Сформировать новый список из элементов, которые неравны последнему элементу исходного списка
29	Составить однонаправленный список из N чисел. Сформировать новый список из элементов исходного списка, следующих в обратном порядке
30	Составить однонаправленный список из N чисел. Удалить из списка элементы, значения которых равны первому элементу списка. Сформировать новый список из элементов, которые неравны минимальному элементу исходного списка

1.4 Указания для выполнения лабораторной работы

Каждое задание необходимо решить в соответствии с изученными методами формирования, вывода и обработки данных однонаправленных или *двунаправленных* списков в языке C++. Обработку списков следует выполнить на основе базовых алгоритмов: *поиск* по списку, вставка элемента в *список*, *удаление элемента* из списка, удаление всего списка. При объявлении списков выполните комментирование используемых полей. Следует реализовать каждое задание в соответствии с приведенными этапами:

- изучить постановку задачи, выделив при этом все виды данных;
- сформулировать математическую постановку задачи;
- выбрать метод решения задачи, если это необходимо;
- разработать графическую схему алгоритма;
- записать разработанный алгоритм на языке C++;
- разработать контрольный тест к программе;
- отладить программу;
- представить отчет по работе.

1.5 Требования к отчету по лабораторной работе

Отчет по лабораторной работе должен соответствовать следующей структуре:

- титульный лист;
- цель работы;
- постановка задачи;
- алгоритм решения задачи;
- листинг программы (подраздел должен содержать текст программы на языке программирования C++, реализованный в среде MS Visual Studio);
- контрольный тест (Подраздел содержит наборы исходных данных и полученные в ходе выполнения программы результаты);
- выводы по лабораторной работе.

1.6 Контрольные вопросы к лабораторной работе

1. Любой ли список является связным?
2. В чем отличие первого элемента однонаправленного (*двунаправленного*) списка от остальных элементов этого же списка?
3. В чем отличие последнего элемента однонаправленного (*двунаправленного*) списка от остальных элементов этого же списка?
4. Почему при работе с однонаправленным списком необходимо позиционирование на первый элемент списка?
5. Почему при работе с двунаправленным списком не обязательно позиционирование на первый элемент списка?
6. В чем принципиальные отличия выполнения добавления (удаления) элемента на первую и любую другую позиции в однонаправленном списке?
7. В чем принципиальные отличия выполнения основных операций в однонаправленных и *двунаправленных* списках?
8. С какой целью в программах выполняется проверка на пустоту однонаправленного (*двунаправленного*) списка?
9. С какой целью в программах выполняется удаление однонаправленного (*двунаправленного*) списка по окончании работы с ним? Как изменится работа программы, если операцию удаления списка не выполнять?

2 ЛАБОРАТОРНАЯ РАБОТА № 2. НЕЛИНЕЙНЫЕ ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ

2.1 Цель работы

Цель работы: изучить понятие, формирование, особенности доступа к данным и работы с памятью в бинарных деревьях, научиться решать задачи с использованием *рекурсивных функций* и алгоритмов обхода бинарных деревьев в языке С++.

При выполнении лабораторной работы для каждого задания требуется написать программу на языке С++, в которой выполнено формирование бинарных деревьев в соответствии с постановкой задачи, ввод данных элементов деревьев с учетом типа информационного поля, их обработка и *вывод* на экран в указанном формате. Для хранения данных бинарных деревьев следует использовать ресурсы динамической памяти. Ввод данных осуществляется с клавиатуры с учетом требований к входным данным, содержащихся в постановке задачи.

2.2 Краткие теоретические сведения

Введение в динамическую переменную двух и более полей-указателей позволяет получить нелинейные структуры данных. Наиболее распространенными являются структуры с иерархическим представлением

Дерево состоит из элементов, называемых ***узлами*** (вершинами). Узлы соединены между собой направленными дугами. В случае $X \rightarrow Y$ вершина *X* называется *родителем*, а *Y – сыном* (*дочерью*).

Дерево имеет единственный узел, не имеющий родителей (ссылок на этот узел), который называется ***корнем***. Любой другой узел имеет ровно одного родителя, т.е. на каждый узел дерева имеется ровно одна ссылка.

Узел, не имеющий сыновей, называется ***листом***.

Внутренний узел – это узел, не являющийся ни листом, ни корнем. ***Порядок узла*** равен количеству его узлов-сыновей. ***Степень дерева*** – максимальный порядок его узлов. ***Высота (глубина) узла*** равна числу его родителей плюс один. ***Высота дерева*** – это наибольшая высота его узлов.

Бинарное дерево – это динамическая структура данных, в которой каждый узел-родитель содержит, кроме данных, не более двух сыновей (левый и правый)

Если дерево организовано таким образом, что для каждого узла все ключи его левого поддерева меньше ключа этого узла, а все ключи его правого поддерева – больше, оно называется деревом поиска. Одинаковые ключи здесь не допускаются.

Представление динамических данных в виде древовидных структур оказывается довольно удобным и эффективным для решения задач быстрого поиска информации.

Сбалансированными, или *AVL*-деревьями, называются деревья, для каждого узла которых высóты его поддеревьев различаются не более чем на 1. Причем этот критерий обычно называют *AVL*-сбалансированностью в отличие от идеальной сбалансированности, когда для каждого узла дерева количество узлов в его левом и правом поддеревьях различаются не более чем на единицу [44].

В общем случае при работе с такими структурами необходимо уметь:

– построить (создать) дерево;
– вставить новый элемент;
– обойти все элементы дерева, например, для просмотра или выполнения некоторой операции;
– выполнить поиск элемента с указанным значением в узле;
– удалить заданный элемент.

Обычно бинарное дерево строится сразу упорядоченным, т.е. узел левого сына имеет значение меньшее, чем значение родителя, а узел правого сына – большее.

2.3 Задания для выполнения лабораторной работы

Разработать проект для обработки дерева поиска, каждый элемент которого содержит структуру данных из варианта курсовой работы. Ключ поиска указан в варианте. В качестве структуры использовать 1) структуру с указателями на правого и левого потомка 2) массив с двумя дополнительными полями. Разобраться с балансировками (поворотами) деревьев. В программе должны быть реализованы следующие возможности:

– создание дерева;
– добавление новой записи;

– поиск информации по заданному ключу;
– удаление информации с заданным ключом;
– вывод информации;
– решение индивидуального задания;
– освобождение памяти при выходе из программы.

Индивидуальные задания:

1. Поменять местами информацию, содержащую максимальный и минимальный ключи.
2. Подсчитать число листьев в дереве.
3. Удалить из дерева ветвь с вершиной, имеющей заданный ключ.
4. Определить глубину дерева.
5. Определить число узлов на каждом уровне дерева.
6. Удалить из левой ветви дерева узел с максимальным значением ключа и все связанные с ним узлы.
7. Определить количество узлов с четными ключами.
8. Определить число листьев на каждом уровне дерева.
9. Определить число узлов в дереве, имеющих только одного потомка.
10. Определить количество узлов правой ветви дерева.
11. Определить количество записей в дереве, начинающихся с введенной с клавиатуры буквы.
12. Найти среднее значение всех ключей дерева и найти строку, имеющую ближайший к этому значению ключ.
13. Определить количество узлов левой ветви дерева.
14. Определить число узлов в дереве, имеющих двух потомков.
15. Найти запись с ключом, ближайшим к среднему значению между максимальным и минимальным значениями ключей.
16. Подсчитать число узлов в дереве после узла с заданным ключом.
17. Определить количество узлов в дереве, значения которых отличаются от корня не более чем на заданное значение.
18. В левом поддереве вывести значения с четырьмя наименьшими ключами.
19. В правом поддереве вывести значения с тремя наибольшими ключами.

20. В левом поддереве вывести значения с тремя наибольшими ключами.
21. В правом поддереве вывести значения с четырьмя наименьшими ключами.
22. Найти запись с ключом, равным разности между max и min значениями левого поддерева.
23. Найти число узлов, значения которых больше (min + заданное значение).
24. Найти число узлов, находящихся после узла с заданным значением (листья не считать).
25. Найти среднее значение для всех узлов в дереве, имеющего только одного потомка.
26. Определить три узла, значения которых наиболее близки к корневому.
27. Определить наиболее длинную ветку их узлов, чередующих следующим образом: правый потомок, левый потомок, правый, левый и тд.
28. Вывести наиболее длинную ветку, содержащую правых потомков.
29. Вывести наиболее длинную ветку, содержащую левых потомков.
30. Организовать обработку данных типа дерево, в структуре которой имеется указатель на родительский узел.

2.4 Указания для выполнения лабораторной работы

Каждое из заданий необходимо решить в соответствии с изученными методами и реализованными алгоритмами формирования, вывода и обработки данных бинарных деревьев в языке C++. Обработку бинарных деревьев следует выполнить на основе базовых алгоритмов: *поиск по* дереву, вставка элемента в *дерево*, балансировка дерева, *удаление элемента* из дерева, удаление всего дерева. При объявлении бинарных деревьев выполните комментирование используемых полей.

Следует реализовать каждое задание в соответствии с приведенными этапами:

– изучить постановку задачи, выделив при этом все виды данных;

– выбрать метод решения задачи, если это необходимо;
– разработать графическую схему алгоритма;
– записать разработанный алгоритм на языке C++;
– разработать контрольный тест к программе;
– отладить программу;
– представить отчет по работе.

2.5 Требования к отчету по лабораторной работе

Отчет по лабораторной работе должен соответствовать следующей структуре:

– титульный лист;
– постановка задачи;
– алгоритм решения задачи (в подразделе описывается разработка структуры алгоритма, обосновывается абстракция данных, задача разбивается на подзадачи);
– листинг программы (подраздел должен содержать текст программы на языке программирования C++, реализованный в среде MS Visual Studio);
– контрольный тест (подраздел содержит наборы исходных данных и полученные в ходе выполнения программы результаты);
– выводы по лабораторной работе.

2.6 Контрольные вопросы к лабораторной работе

1. С чем связана популярность использования деревьев в программировании?
2. Можно ли список отнести к деревьям? Ответ обоснуйте.
3. Какие данные содержат адресные поля элемента *бинарного дерева*?
4. Может ли *бинарное дерево* быть строгим и неполным? Ответ обоснуйте.
5. Может ли *бинарное дерево* быть нестрогим и полным? Ответ обоснуйте.
6. Каким может быть почти сбалансированное *бинарное дерево*: полным, неполным, строгим, нестрогим? Ответ обоснуйте.

7. Куда может быть добавлен элемент в *бинарное дерево* в зависимости от его вида (полное, неполное, строгое, нестрогое)? Вид дерева при этом должен сохраниться.
8. Куда может быть добавлен элемент в сбалансированное *бинарное дерево*? Вид дерева при этом должен сохраниться.
9. Чем отличаются, с точки зрения реализации алгоритма, прямой, симметричный и *обратный обходы бинарного дерева*?

3 ЛАБОРАТОРНАЯ РАБОТА № 3. АЛГОРИТМЫ СОРТИРОВКИ МАССИВОВ

3.1 Цель работы

Цель работы: изучить основные алгоритмы внутренних сортировок и научиться решать задачи сортировок массивов различными методами.

При выполнении лабораторной работы для каждого задания требуется написать программу на языке C++, которая получает на входе числовые данные, выполняет генерацию и *вывод* массива *указанного типа* в зависимости от постановки задачи. В каждой задаче необходимо выполнить сортировку данных и реализовать один из алгоритмов в виде отдельных функций. Ввод данных осуществляется с клавиатуры или из файла с учетом требований к *входным данным*, содержащихся в постановке задачи.

3.2 Краткие теоретические сведения

Алгоритмом сортировки называется *алгоритм* для упорядочения некоторого *множества* элементов. Обычно под алгоритмом сортировки подразумевают *алгоритм* упорядочивания *множества* элементов по возрастанию или убыванию.

В алгоритмах сортировки лишь часть данных используется в качестве *ключа сортировки*. *Ключом сортировки* называется *атрибут* (или несколько атрибутов), по значению которого определяется порядок элементов. Таким образом, при написании алгоритмов сортировок массивов следует учесть, что *ключ* полностью или частично совпадает с данными.

Практически каждый *алгоритм* сортировки можно разбить на 3 части:

- сравнение, определяющее упорядоченность пары элементов;
- *перестановку*, меняющую местами пару элементов;
- собственно, сортирующий алгоритм, который осуществляет сравнение и *перестановку* элементов до тех пор, пока все *элементы множества* не будут упорядочены.

Алгоритмы сортировки имеют большое практическое применение. Их можно встретить там, где речь идет об обработке и хранении

больших объемов информации. Некоторые задачи обработки данных решаются проще, если данные заранее упорядочить.

Параметры, по которым будет производиться оценка алгоритмов:

– *время сортировки* – основной параметр, характеризующий быстродействие алгоритма;
– *память* – один из параметров, который характеризуется тем, что ряд алгоритмов сортировки требуют выделения дополнительной памяти под временное хранение данных. При оценке используемой памяти не будет учитываться место, которое занимает исходный массив данных и независящие от входной последовательности затраты, например, на хранение *кода программы*;
– *устойчивость* – это параметр, который отвечает за то, что сортировка не меняет взаимного расположения равных элементов;
– *естественность поведения* – параметр, которой указывает на эффективность метода при обработке уже отсортированных, или частично отсортированных данных. Алгоритм ведет себя естественно, если учитывает эту характеристику входной последовательности и работает лучше.

Все разнообразие и многообразие алгоритмов сортировок можно классифицировать по различным признакам, например, по устойчивости, по поведению, по использованию операций сравнения, по потребности в дополнительной памяти, по потребности в знаниях о структуре данных, выходящих за рамки *операции* сравнения, и другие.

Наиболее подробно рассмотрим классификацию алгоритмов сортировки по сфере применения. В данном случае основные типы упорядочивания делятся следующим образом:

– *Внутренняя сортировка* – это алгоритм сортировки, который в процессе *упорядочивания данных* использует только *оперативную память* (*ОЗУ*) компьютера. То есть оперативной памяти достаточно для помещения в нее сортируемого массива данных с произвольным доступом к любой ячейке и собственно для выполнения алгоритма. Внутренняя сортировка применяется во всех случаях, за исключением однопроходного считывания данных и однопроходной записи отсортированных данных. В зависимости от конкретного алгоритма и его

реализации данные могут сортироваться в той же области памяти, либо использовать дополнительную *оперативную память*.

– *Внешняя сортировка* – это алгоритм сортировки, который при проведении *упорядочивания данных* использует *внешнюю память*, как правило, жесткие диски. *Внешняя сортировка* разработана для обработки больших списков данных, которые не помещаются в *оперативную память*. Обращение к различным носителям накладывает некоторые дополнительные ограничения на данный алгоритм: доступ к носителю осуществляется последовательным образом, то есть в каждый момент времени можно считать или записать только элемент, следующий за текущим; объем данных не позволяет им разместиться в *ОЗУ*.

Внутренняя *сортировка* является базовой для любого алгоритма внешней сортировки – отдельные части массива данных сортируются в оперативной памяти и с помощью специального алгоритма сцепляются в один *массив*, упорядоченный по ключу.

3.3 Задания для выполнения лабораторной работы

Сформировать случайным образом массив записей из 10000 штук. Скопировать его трижды. Провести сортировку по одному ключу, двум ключам и трем ключам. Оценить эффективность каждой сортировки по времени и количеству перестановок и сравнений. Провести четвертую сортировку, включив проверку на каждом шаге, что массив уже отсортирован. Оценить эффективность включения такой оценки. Организовать поиск по трем указанным ключам. Оценить эффективность поиска.

В таблице 3.1 приведены варианты для выполнения индивидуального задания.

Таблица 3.1 – Варианты индивидуальных заданий

Вариант	Типы ключей	Направление сортировки	Тип сортировки	Вид поиска
1	str, float, int	В, ВВ, ВВВ	шейкер	С барьером
2	Float, str, date	В, ВУ, ВВУ	выбор макс	линейный
3	Str, time, int	В, УВ, ВУВ	выбор мин	бинарный

Продолжение таблицы 3.1

4	Time, int, float	В, ВУ, ВУУ	Пузырек слева направо	интерполяци-онный
5	Date, str, int	У, УУ, УУУ	Пузырек справа налево	С барьером
6	Float, int, time	У, ВУ, УВУ	быстрая	линейный
7	Int, date, str	У, УВ, УВВ	Вставки слева направо	бинарный
8	Date, float, str	У, ВВ, УУВ	Вставки справа налево	интерполяци-онный
9	Date, date, str	У, УУ, ВВВ	шейкер	С барьером
10	Time, time, int	У, ВУ, ВВУ	выбор макс	линейный
11	str, float, int	У, УВ, ВУВ	выбор мин	бинарный
12	Float, str, date	У, ВУ, ВУУ	Пузырек слева направо	интерполяци-онный
13	Str, time, int	В, УУ, УУУ	Пузырек справа налево	С барьером
14	Time, int, float	В, ВУ, УВУ	быстрая	линейный
15	Date, str, int	В, УВ, УВВ	Вставки слева направо	бинарный
16	Float, int, time	В, ВВ, УУВ	Вставки справа налево	интерполяци-онный
17	Int, date, str	В, ВВ, ВВВ	шейкер	С барьером
18	Date, float, str	В, ВУ, ВВУ	выбор макс	линейный
19	Date, date, str	В, УВ, ВУВ	выбор мин	бинарный
20	Time, time, int	В, ВУ, ВУУ	Пузырек слева направо	интерполяци-онный

Продолжение таблицы 3.1

21	str, float, int	У, УУ, УУУ	Пузырек справа налево	С барьером
22	Float, str, date	У, ВУ, УВУ	быстрая	линейный
23	Str, time, int	У, УВ, УВВ	Вставки слева направо	бинарный
24	Time, int, float	У, ВВ, УУВ	Вставки справа налево	интерполяционный
25	Date, str, int	У, УУ, ВВВ	шейкер	С барьером
26	Float, int, time	У, ВУ, ВВУ	выбор макс	линейный
27	Int, date, str	У, УВ, ВУВ	выбор мин	бинарный
28	Date, float, str	У, ВУ, ВУУ	Пузырек слева направо	интерполяционный
29	Date, date, str	В, УУ, УУУ	Пузырек справа налево	С барьером
30	Time, time, int	В, ВУ, УВУ	быстрая	линейный
31	Date, date, str	В, УВ, УВВ	Вставки слева направо	бинарный
32	Time, time, int	В, ВВ, УУВ	Вставки справа налево	интерполяционный

При сортировке по двум и более ключам, первый ключ – основной, если значения первого ключа у записей совпадают, то они уже становятся в порядке, предусмотренном вторым ключом и т.д.

Направление сортировки: В –возрастание, У- убывание

3.4 Указания для выполнения лабораторной работы

Каждое задание необходимо решить в соответствии с изученными алгоритмами внутренних сортировок. Программные коды сле-

дует реализовать на языке C++. Этапы решения сопроводить комментариями в коде. В отчете следует отразить разработку, обоснование математической модели решения задачи и примеры входных и выходных файлов.

Следует реализовать каждое задание в соответствии с приведенными этапами:

– изучить постановку задачи, выделив при этом все виды данных;
– выбрать метод решения задачи, если это необходимо;
– разработать графическую схему алгоритма;
– записать разработанный алгоритм на языке C++;
– разработать контрольный тест к программе;
– отладить программу;
– представить отчет по работе.

3.5 Требования к отчету по лабораторной работе

Отчет по лабораторной работе должен соответствовать следующей структуре:

– титульный лист;
– цель работы;
– постановка задачи;
– алгоритм решения задачи;
– листинг программы (подраздел должен содержать текст программы на языке программирования C++, реализованный в среде MS Visual Studio);
– контрольный тест (подраздел содержит наборы исходных данных и полученные в ходе выполнения программы результаты);
– выводы по лабораторной работе.

3.6 Контрольные вопросы к лабораторной работе

1. Чем можно объяснить многообразие алгоритмов сортировок?
2. Почему на данный момент не существует универсального алгоритма сортировки?
3. Как соблюдение свойств устойчивости и естественности влияет на трудоемкость алгоритма сортировки?
4. За счет чего в алгоритмах быстрых сортировок происходит выигрыш при выполнении операций сравнения и перестановок?

5. Какие из перечисленных алгоритмов наиболее эффективны на почти отсортированных массивах? За счет чего происходит выигрыш?
6. Почему алгоритмы быстрых сортировок не дают большого выигрыша при малых размерах массивов?
7. В чем преимущества и недостатки по отношению друг к другу алгоритмов сортировок?
8. Как определить, какому алгоритму сортировки отдать предпочтение при решении задачи?

4 ЛАБОРАТОРНАЯ РАБОТА № 4. АЛГОРИТМЫ ХЕШИРОВАНИЯ ДАННЫХ

4.1 Цель работы

Цель работы: изучить построение функции *хеширования* и алгоритмов *хеширования* данных и научиться разрабатывать алгоритмы открытого и закрытого *хеширования* при решении задач на языке C++.

При выполнении лабораторной работы для каждого задания требуется написать программу на языке C++, которая получает данные с клавиатуры или из входного файла, выполняет их обработку в соответствии с требованиями задания и выводит результат в выходной *файл*. Для обработки данных необходимо реализовать функции алгоритмов *хеширования* данных.

4.2 Краткие теоретические сведения

Для решения задачи поиска необходимого элемента среди данных большого объема был предложен алгоритм ***хеширования*** (*hashing* – перемешивание), при котором создаются ключи, определяющие данные массива и на их основании данные записываются в таблицу, названную ***хеш-таблицей***. Ключи для записи определяются при помощи функции ***i = h(key)***, называемой ***хеш-функцией***. Алгоритм хеширования определяет положение искомого элемента в хеш-таблице по значению его ключа, полученного хеш-функцией.

Понятие ***хеширования –*** это разбиение общего (базового) набора уникальных ключей элементов данных на непересекающиеся наборы с определенным свойством.

Функция, отображающая ключи элементов данных во множество целых чисел (индексы в таблице – ***хеш-таблица***), называется ***функцией хеширования***, или ***хеш-функцией***:

$$i = h(key),$$

где *key* – преобразуемый ключ, *i* – получаемый индекс таблицы, т.е. ключ отображается во множество целых чисел (***хеш-адреса***), которые впоследствии используются для доступа к данным.

Однако хеш-функция для нескольких значений ключа может давать одинаковое значение позиции *i* в таблице. Ситуация, при которой

два или более ключа получают один и тот же индекс (хеш-адрес), называется ***коллизией*** при хешировании.

Хорошей хеш-функцией считается такая функция, которая минимизирует коллизии и распределяет данные равномерно по всей таблице, а совершенной хеш-функцией – функция, которая не порождает коллизий.

Разрешить коллизии при хешировании можно двумя методами:

– методом открытой адресации с линейным опробованием;
– методом цепочек.

Хеш-таблица представляет собой обычный массив с необычной адресацией, задаваемой хеш-функцией.

Хеш-структуру считают обобщением массива, который обеспечивает быстрый прямой доступ к данным по индексу.

Имеется множество схем хеширования, различающихся как выбором удачной функции $h(key)$, так и алгоритма разрешения конфликтов. Эффективность решения реальной практической задачи будет существенно зависеть от выбираемой стратегии

Метод открытой адресации с линейным опробованием. Изначально все ячейки хеш-таблицы, которая является обычным одномерным массивом, помечены как не занятые. Поэтому при добавлении нового ключа проверяется, занята ли данная ячейка. Если ячейка занята, то алгоритм осуществляет осмотр по кругу до тех пор, пока не найдется свободное место («открытый адрес»), т.е. либо элементы с однородными ключами размещают вблизи полученного индекса, либо осуществляют двойное хеширование, используя для этого разные, но взаимосвязанные хеш-функции.

В дальнейшем, осуществляя поиск, сначала находят по ключу позицию i в таблице, и, если ключ не совпадает, то последующий поиск осуществляется в соответствии с алгоритмом разрешения конфликтов, начиная с позиции i по списку.

Метод цепочек используется чаще предыдущего. В этом случае полученный хеш-функцией индекс i трактуется как индекс в хеш-таблице списков, т.е. ключ key очередной записи отображается на позицию $i = h(key)$ таблицы. Если позиция свободна, то в нее помещается элемент с ключом key, если же она занята, то отрабатывается алгоритм разрешения конфликтов, в результате которого такие ключи добавляются в список, начинающийся в i-й ячейке хеш-таблицы.

4.3 Задания для выполнения лабораторной работы

В задании необходимо обеспечить построение хэш-таблицы, а также добавление, удаление данных, поиск. определить вероятность появления коллизии при добавлении данных.

В таблице 4.1 приведены варианты для выполнения индивидуального задания.

Таблица 4.1 – Варианты индивидуальных заданий

Вариант	Вид хэширования	Вид хэширования	Метод разрешения коллизии	Доп. структуры
1	Метод деления	Метод середины квадрата	Метод открытой адресации	–
2	Аддитивный метод	Метод исключающего ИЛИ	Метод цепочек	Дерево
3	Метод середины квадрата	Мультипликативный метод	Метод цепочек	Однонаправленный список
4	Метод исключающего ИЛИ	Метод деления	Метод цепочек	Двунаправленный список
5	Мультипликативный метод	Аддитивный метод	Метод цепочек	Дерево
6	Двойное хеширование	Хеширование кукушки	Метод открытой адресации	–
7	Хеширование кукушки	Двойное хеширование	Метод открытой адресации	–
8	Adler-32	Циклический избыточный код CRC	Метод цепочек	Однонаправленный список
9	FNV	MurmurHash	Метод цепочек	Двунаправленный список

Продолжение таблицы 4.1

10	PJW	Хэш-функция Дженкинса	Метод цепочек	Дерево
11	Метод деления	Аддитивный метод	Метод цепочек	Однонаправленный список
12	Аддитивный метод	Метод середины квадрата	Метод цепочек	Двунаправленный список
13	Метод середины квадрата	Метод исключающего ИЛИ	Метод цепочек	Дерево
14	Метод исключающего ИЛИ	Мультипликативный метод	Метод открытой адресации	–
15	Мультипликативный метод	Метод деления	Метод цепочек	Дерево
16	Adler-32	Циклический избыточный код CRC	Метод цепочек	Двунаправленный список
17	FNV	MurmurHash	Метод цепочек	Дерево
18	PJW	Хэш-функция Дженкинса	Метод цепочек	Однонаправленный список
19	Двойное хеширование	Хеширование кукушки	Метод открытой адресации	–
20	Хеширование кукушки	Двойное хеширование	Метод открытой адресации	–
21	Метод деления	Аддитивный метод	Метод цепочек	Дерево
22	Аддитивный метод	Метод середины квадрата	Метод цепочек	Двунаправленный список
23	Метод середины квадрата	Метод исключающего ИЛИ	Метод цепочек	Дерево

Продолжение таблицы 3.1

24	Метод исключающего ИЛИ	Мультипликативный метод	Метод цепочек	Однонаправленный список
25	Мультипликативный метод	Метод деления	Метод открытой адресации	–
26	Adler-32	Циклический избыточный код CRC	Метод цепочек	Однонаправленный список
27	FNV	MurmurHash	Метод цепочек	Двунаправленный список
28	PJW	Хеш-функция Дженкинса	Метод цепочек	Дерево
29	Двойное хеширование	Хеширование кукушки	Метод открытой адресации	–
30	Хеширование кукушки	Двойное хеширование	Метод открытой адресации	–

4.4 Указания для выполнения лабораторной работы

Каждое задание необходимо решить в соответствии с изученным алгоритмами *хеширования* данных, реализовав программный код на языке C++. Рекомендуется воспользоваться материалами лекции 38, где подробно рассматриваются описание используемых в работе алгоритмов, примеры их реализации на языке C++. Программу для решения каждого задания необходимо разработать методом процедурной абстракции, используя функции, коды которых требуется сопроводить комментариями. Результаты обработки данных следует выводить в выходной *файл* и дублировать *вывод* на экране. В отчете следует отразить разработку, обоснование математической модели решения задачи и результаты тестирования программ.

Следует реализовать каждое задание в соответствии с приведенными этапами:

– изучить постановку задачи, выделив при этом все виды данных;
– выбрать метод решения задачи, если это необходимо;
– разработать графическую схему алгоритма;
– записать разработанный алгоритм на языке C++;
– разработать контрольный тест к программе;
– отладить программу;
– представить отчет по работе.

4.5 Требования к отчету по лабораторной работе

Отчет по лабораторной работе должен соответствовать следующей структуре:

– титульный лист;
– цель работы;
– постановка задачи;
– алгоритм решения задачи (подразделе описывается разработка структуры алгоритма, обосновывается абстракция данных, задача разбивается на подзадачи);
– листинг программы (подраздел должен содержать текст программы на языке программирования C++, реализованный в среде MS Visual Studio);
– контрольный тест (подраздел содержит наборы исходных данных и полученные в ходе выполнения программы результаты);
– выводы по лабораторной работе.

4.6 Контрольные вопросы к лабораторной работе

1. Каков принцип построения хеш-таблиц?
2. Существуют ли универсальные методы построения хеш-таблиц? Ответ обоснуйте.
3. Почему возможно возникновение *коллизий*?
4. Каковы методы устранения *коллизий*? Охарактеризуйте их эффективность в различных ситуациях.
5. Назовите преимущества открытого и закрытого *хеширования*.
6. В каком случае поиск в хеш-таблицах становится неэффективен?
7. Как выбирается метод изменения адреса при повторном *хешировании*?

5 ЛАБОРАТОРНАЯ РАБОТА № 5. АЛГОРИТМЫ НА ГРАФАХ.

5.1 Цель работы

Цель работы: изучить основные алгоритмы на графах.

При выполнении лабораторной работы для каждого задания требуется написать программу на языке C++, которая получает на входе числовые данные, выполняет их обработку в соответствии с требованиями задания и выводит результат на экран. Для обработки данных необходимо реализовать алгоритмы *обхода графа* в соответствии с постановкой задачи. Ввод данных осуществляется из файла с учетом требований к входным данным, содержащихся в постановке задачи.

5.2 Краткие теоретические сведения

Граф – это совокупность двух конечных множеств: *множества* точек и *множества* линий, попарно соединяющих некоторые из этих точек. Множество точек называется *вершинами (узлами) графа*. Множество линий, соединяющих *вершины графа*, называются *ребрами (дугами) графа*.

Ориентированный граф (орграф) – граф, у которого все *ребра* ориентированы, т.е. *ребрам* которого присвоено направление.

Неориентированный граф (неорграф) – граф, у которого все *ребра* неориентированы, т.е. *ребрам* которого не задано направление.

Смешанный граф – граф, содержащий как ориентированные, так и неориентированные *ребра*.

Петлей называется *ребро*, соединяющее вершину саму с собой. Две вершины называются *смежными*, если существует соединяющее их *ребро*. *Ребра*, соединяющие одну и ту же пару вершин, называются *кратными*.

Простой граф – это *граф*, в котором нет ни петель, ни кратных ребер.

Мультиграф – это *граф*, у которого любые две вершины соединены более чем одним *ребром*.

Маршрутом в графе называется конечная чередующаяся последовательность смежных вершин и ребер, соединяющих эти вершины.

Маршрут называется *открытым*, если его начальная и конечная вершины различны, в противном случае он называется замкнутым.

Маршрут называется **цепью**, если все его *ребра* различны. Открытая цепь называется **путем**, если все ее вершины различны.

Замкнутая цепь называется **циклом**, если различны все ее вершины, за исключением концевых.

Граф называется *связным*, если для любой пары вершин существует соединяющий их *путь*.

Вес вершины – число (действительное, целое или рациональное), поставленное в соответствие данной вершине (интерпретируется как *стоимость*, *пропускная способность* и т. д.). *Вес (длина) ребра* – число или несколько чисел, которые интерпретируются по отношению к ребру как *длина*, *пропускная способность* и т. д.

Взвешенный граф – граф, каждому ребру которого поставлено в соответствие некое *значение* (*вес ребра*).

Выбор структуры данных для хранения графа в памяти компьютера имеет принципиальное *значение* при разработке *эффективных алгоритмов*

Под **обходом графов (поиском на графах)** понимается процесс *систематического* просмотра всех ребер или *вершин графа* с целью отыскания ребер или вершин, удовлетворяющих некоторому условию.

При решении многих задач, использующих графы, необходимы эффективные методы регулярного обхода вершин и ребер графов. К стандартным и наиболее распространенным методам относятся:

– *поиск в глубину* (Depth First Search, *DFS*);
– *поиск в ширину* (Breadth First Search, *BFS*).

Эти методы чаще всего рассматриваются на *ориентированных графах*, но они применимы и для неориентированных, *ребра* которых считаются *двунаправленными*. Алгоритмы *обхода в глубину* и в ширину лежат в основе решения различных задач обработки графов, например, построения *остовного леса*, проверки *связности*, ациклично-сти, вычисления расстояний между вершинами и других.

5.3 Задания для выполнения лабораторной работы

1. Найти в графе вершину с минимальной степенью.
2. Найти в графе вершину с максимальной степенью.
3. Дан граф. Можно ли закрыв три дороги, добиться того, чтобы из вершины *A* нельзя было бы попасть в вершину *B*.

4. Найти медиану графа, то есть такую вершину, чтобы сумма расстояний от нее до всех остальных вершин была минимальной.
5. Из графа удалить вершины, из которых недостижима заданная вершина.
6. Источником орграфа назовем вершину, от которой достижимы все другие вершины; стоком – вершину, достижимую от всех других вершин. По орграфу найти все его источники и стоки.
7. Проверить наличие цикла в заданном графе.
8. Проверить наличие цепи в заданном графе.
9. Реализовать алгоритм Дейкстры для нахождения кратчайшего пути между двумя заданными вершинами графа.
10. Задана система двусторонних дорог. Найти два города и соединяющий их путь, который проходит через каждую из дорог графа ровно один раз.
11. Найти диаметр графа, то есть максимальное расстояние между всевозможными парами его вершин.
12. Вычислить диаметр заданного графа.
13. В графе найти вершину, наиболее удаленную от заданной.
14. Проверить связность заданного графа.
15. Определить, является ли заданный граф деревом.
16. В графе найти все вершины, к которым существует путь длины n из некоторой заданной.
17. В графе найти все вершины, от которых существует путь длины n к заданной вершине.
18. Определить, является ли граф двудольным.
19. Определить, является ли граф эйлеровым.
20. Определить, является ли граф гамильтоновым.
21. Построить минимальное покрывающее дерево в графе, используя алгоритм Краскала.
22. Построить минимальное покрывающее дерево в графе, используя алгоритм Прима.
23. В данном графе найти вершину с максимальной степенью.
24. Проверить правильность утверждения для заданного графа: число степеней вершин графа, имеющих нечетную степень, четно.

25. Проверить правильность утверждения для заданного графа: сумма степеней всех вершин графа равна удвоенному числу ребер.
26. По системе односторонних дорог определить, есть ли в ней город, из которого можно попасть во все остальные города.
27. По системе двусторонних дорог определить, можно ли, построив какие-нибудь дороги, добиться, чтобы из данного города можно было попасть во все остальные.
28. По системе двусторонних дорог определить, можно ли, перегородив какие-нибудь дороги, добиться, чтобы из города *A* нельзя было попасть в город *B*.
29. Задана система двусторонних дорог и город *A*. Определить все города, расстояния от которых до города *A* больше *n*.
30. Определить, можно ли в заданной системе односторонних дорого проехать из города *A* в город *B* таким образом, чтобы посетить город *C* и не проезжать никакой дороги более одного раза.

5.4 Указания для выполнения лабораторной работы

Каждое задание необходимо решить в соответствии с изученными алгоритмами обработки данных, реализовав программный код на языке С. Этапы решения сопроводить комментариями в коде. В отчете следует отразить разработку и обоснование математической модели решения задачи, представить результаты тестирования программ.

Следует реализовать каждое задание в соответствии с приведенными этапами:

– изучить постановку задачи, выделив при этом все виды данных;
– выбрать метод решения задачи, если это необходимо;
– разработать графическую схему алгоритма;
– записать разработанный алгоритм на языке С++;
– разработать контрольный тест к программе;
– отладить программу;
– представить отчет по работе.

5.5 Требования к отчету по лабораторной работе

Отчет по лабораторной работе должен соответствовать следующей структуре:

- титульный лист;
- цель работы;
- постановка задачи;
- алгоритм решения задачи;
- листинг программы (подраздел должен содержать текст программы на языке программирования C++, реализованный в среде MS Visual Studio);
- контрольный тест (подраздел содержит наборы исходных данных и полученные в ходе выполнения программы результаты);
- выводы по лабораторной работе.

5.6 Контрольные вопросы к лабораторной работе

1. Как связаны между собой различные способы представления графов?
2. Как от вида или представления графа зависит временная сложность алгоритмов поиска в глубину и в ширину?
3. Как при реализации в коде выполняется возвращение из тупиковых вершин при обходе графа?
4. Как выполняется обход в несвязном графе?
5. Распространяются ли понятия "*поиск в глубину*" и "*поиск в ширину*" на несвязный граф? Ответ обоснуйте.
6. Охарактеризуйте трудоемкость рекурсивного и не рекурсивного алгоритмов *обхода графа*
7. С какими видами графов работают алгоритмы Дейкстры, Флойда и переборные алгоритмы?
8. Как от представления графа зависит эффективность алгоритма его обхода?
9. За счет чего поиск в ширину является достаточно ресурсоемким алгоритмом?
10. В чем преимущества алгоритмов *обхода графа* в ширину?
11. Каким образом в алгоритме перебора с возвратом при обходе графа обрабатывается посещение тупиковых вершин?
12. Поясните на примере *обхода графа* этап обратного хода в волновом алгоритме. Почему его удобно выполнять с конца?

6 ЛАБОРАТОРНАЯ РАБОТА № 6. АЛГОРИТМЫ СЖАТИЯ ДАННЫХ

6.1 Цель работы

Цель работы: изучить основные виды и алгоритмы сжатия данных.

При выполнении лабораторной работы для каждого задания требуется написать программу на языке C++, которая получает на данные с клавиатуры или из входного файла, выполняет их обработку в соответствии с требованиями задания и выводит результат в выходной *файл*.

6.2 Краткие теоретические сведения

Сжатие данных – это процесс, обеспечивающий уменьшение объема данных путем сокращения их избыточности. Сжатие данных связано с компактным расположением порций данных стандартного размера. Сжатие данных можно разделить на два основных типа:

- *сжатие без потерь (полностью обратимое)* – это метод сжатия данных, при котором ранее закодированная порция данных восстанавливается после их распаковки полностью без внесения изменений; для каждого типа данных, как правило, существуют свои оптимальные алгоритмы сжатия без потерь;
- *сжатие с потерями* – это метод сжатия данных, при котором для обеспечения *максимальной степени* сжатия исходного массива данных часть содержащихся в нем данных отбрасывается; для текстовых, числовых и табличных данных использование программ, реализующих подобные методы сжатия, является неприемлемыми; в основном такие алгоритмы применяются для сжатия аудио- и видеоданных, статических изображений.

Алгоритм сжатия данных (алгоритм архивации**)** – это *алгоритм*, который устраняет *избыточность* записи данных.

Алфавит кода – множество всех символов входного потока. При сжатии англоязычных текстов обычно используют множество из 128 *ASCII* кодов. При сжатии изображений множество значений пиксела может содержать 2, 16, 256 или другое количество элементов.

Кодовый символ – наименьшая *единица* данных, подлежащая сжатию. Обычно символ – это 1 *байт*, но он может быть битом, тритом {0,1,2}, или чем-либо еще.

Кодовое слово – это последовательность кодовых символов из алфавита кода. Если все слова имеют одинаковую длину (число символов), то такой код называется *равномерным (фиксированной длины)*, а если же допускаются слова разной длины, то – *неравномерным (переменной длины)*.

Код – полное множество слов.

Токен – *единица* данных, записываемая в сжатый *поток* некоторым алгоритмом сжатия. *Токен* состоит из нескольких полей фиксированной или переменной длины.

Фраза – фрагмент данных, помещаемый в словарь для дальнейшего использования в сжатии.

Кодирование – процесс сжатия данных.

Декодирование – *обратный* кодированию процесс, при котором осуществляется восстановление данных.

Отношение сжатия – одна из наиболее часто используемых величин для обозначения эффективности метода сжатия.

Значение 0,6 означает, что данные занимают 60% от первоначального объема. Значения больше 1 означают, что выходной *поток* больше входного (отрицательное сжатие, или расширение).

Коэффициент сжатия – величина, обратная отношению сжатия. Значения больше 1 обозначают сжатие, а значения меньше 1 – расширение.

Средняя длина кодового слова – это величина, которая вычисляется как взвешенная вероятностями сумма длин всех кодовых слов.

Существуют два основных способа проведения сжатия.

Статистические методы – методы сжатия, присваивающие коды переменной длины символам входного потока, причем более короткие коды присваиваются символам или группам символам, имеющим большую *вероятность* появления во входном потоке. Лучшие *статистические методы* применяют *кодирование* Хаффмана.

Словарное сжатие – это методы сжатия, хранящие фрагменты данных в "словаре" (некоторая *структура данных*). Если строка новых данных, поступающих на вход, идентична какому-либо фрагменту, уже находящемуся в словаре, в выходной *поток* помещается *указатель* на этот фрагмент. Лучшие словарные методы применяют метод Зива-Лемпела.

6.3 Задания для выполнения лабораторной работы

Реализовать алгоритмы сжатия согласно варианту. Определить коэффициенты сжатия и время архивации.

В таблице 6.1 приведены варианты для выполнения индивидуального задания.

Таблица 6.1 – Варианты индивидуальных заданий

Вариант и дополнительные требования		Алгоритм 1	Алгоритм 2
1	С паролем	Шенона-Фано	Хаффмена
2	С разбиением на тома	Хаффмена	Арифметическое сжатие
3	С паролем	Арифметичское сжатие	LZW
4	С разбиением на тома	LZW	RLE+BWT
5	С паролем	RLE+BWT	KWE BWT
6	С разбиением на тома	KWE BWT	Шенона-Фано
7	С паролем	Шенона-Фано	Арифметическое сжатие
8	С разбиением на тома	Хаффмена	LZW
9	С паролем	Арифметическое сжатие	RLE+BWT
10	С разбиением на тома	LZW	KWE BWT
11	С паролем	RLE+BWT	Шенона-Фано
12	С разбиением на тома	KWE BWT	Хаффмена
13	С паролем	Шенона-Фано	LZW
14	С разбиением на тома	Хаффмена	RLE+BWT
15	С паролем	Арифметическое сжатие	KWE BWT
16	С разбиением на тома	LZW	Шенона-Фано

Продолжение таблицы 6.1

17	С паролем	RLE+BWT	Хаффмена
18	С разбиением на тома	KWE BWT	Арифметическое сжатие
19	С паролем	Шенона-Фано	RLE+BWT
20	С разбиением на тома	Хаффмена	KWE BWT
21	С паролем	Арифметическое сжатие	Шенона-Фано
22	С разбиением на тома	LZW	Хаффмена
23	С паролем	RLE+BWT	Арифметическое сжатие
24	С разбиением на тома	KWE BWT	LZW
25	С паролем	Шенона-Фано	KWE BWT
26	С разбиением на тома	Хаффмена	Шенона-Фано
27	С паролем	Арифметическое сжатие	Хаффмена
28	С разбиением на тома	LZW	Арифметическое сжатие
29	С паролем	RLE+BWT	LZW
30	С разбиением на тома	KWE BWT	RLE+BWT

6.4 Указания для выполнения лабораторной работы

Каждое задание необходимо решить в соответствии с изученным алгоритмами сжатия данных и с использованием кодовых деревьев, реализовав программный код на языке C++. Этапы решения сопроводить комментариями в коде. В отчете следует отразить разработку и обоснование математической модели решения задачи и привести примеры входных и выходных файлов, полученных на этапе тестирования программ. В отчете отразить *анализ* степени сжатия данных на примере тестовых заданий.

Следует реализовать каждое задание в соответствии с приведенными этапами:

– изучить постановку задачи, выделив при этом все виды данных;
– сформулировать математическую постановку задачи;
– выбрать метод решения задачи, если это необходимо;
– разработать графическую схему алгоритма;
– записать разработанный алгоритм на языке C++;
– разработать контрольный тест к программе;
– отладить программу;
– представить отчет по работе.

6.5 Требования к отчету по лабораторной работе

Отчет по лабораторной работе должен соответствовать следующей структуре:

– титульный лист;
– цель работы;
– постановка задачи;
– алгоритм решения задачи;
– листинг программы (подраздел должен содержать текст программы на языке программирования C++, реализованный в среде MS Visual Studio);
– контрольный тест (подраздел содержит наборы исходных данных и полученные в ходе выполнения программы результаты);
– выводы по лабораторной работе.

6.6 Контрольные вопросы к лабораторной работе

1. При кодировании каких данных можно использовать сжатие данных с потерями? Ответ обоснуйте.
2. В чем преимущества и недостатки статических методов и словарного сжатия?
3. Каким образом кодирование по алгоритму Хаффмана через *префиксный код* гарантирует минимальную длину кода?
4. За счет чего в методе Хаффмана поддерживается однозначность соответствия кода кодируемому символу?
5. Почему алгоритм Хаффмана малоэффективен для файлов маленьких размеров?

6. Выполните кодирование по методу Хаффмана через *префиксный код* символов, которые встречаются с вероятностями 0,3; 0,2; 0,1; 0,1; 0,1; 0,05; 0,05; 0,04; 0,03; 0,03. Сравните полученный результат с данными *программной реализации*.
7. Докажите, что метод Хаффмана кодирует информацию без потерь.