ПРЕДИСЛОВИЕ

Целью освоения дисциплины «Программирование для ЭВМ» является формирование необходимых знаний о разработке прикладных программ, методики постановки, решения прикладных задач и выбора оптимального способа для представления данных и их обработки, подготовки и решения инженерных задач на современных персональных компьютерах.

Выполнение курсовой работы является в рамках дисциплины заключительным этапом в развитии таких компетенций как:

- умение выполнять разработку программных систем;
- умение самостоятельно разрабатывать алгоритмы и программы на языке программирования высокого уровня;
- иметь навык выбора и использования различных алгоритмов для решения задач различных классов.

Дисциплина «Программирование для ЭВМ» относится к учебным дисциплинам базовой части образовательной программы направления подготовки Прикладная математика, для успешного освоения которой студент должен владеть знаниями, умениями и навыками, сформированными школьной программной по дисциплине «Основы информатики», а для выполнения курсовой работы также и дисциплинами:

- «Математическая логика»,
- «Программные и аппаратные средства информатики»,
- «Алгоритмы дискретной математики».

Настоящее пособие предназначено для выполнения студентами курсовой работы по дисциплине и содержит организационно-методические рекомендации по выполнению и защите курсовой работы, включающие формируемые компетенции, цель, задание и варианты заданий (в том числе повышенной сложности), краткий теоретический материал, необходимый для выполнения курсовой работы, контрольные вопросы.

Рассматриваемые в пособии теоретические вопросы охватывают три больших раздела: деревья (нелинейные динамические структуры), графы (ориентированные и неориентированные) и поиск подстроки в строке.

В разделе «Деревья» рассмотрены операции, применяемые к данным такого типа, и основные алгоритмы, в которых рекомендовано использование деревьев.

В разделе «Графы» рассмотрены два способа представления графов, основные приемы работы с ориентированным и неориентированным графом, а также алгоритмы Дейкстры и Флойда.

В разделе «Поиск» рассмотрены алгоритмы поиска подстроки в строке: линейный, Боуера и Мура, а также Кнута. Мориса и Пратта.

Пособие имеет прикладной характер и может быть использовано и как справочник при самостоятельной работе, в том числе при подготовке к экзамену.

Навыки, приобретенные в процессе выполнения курсовой работы, могут быть полезны не только в дальнейшей профессиональной деятельности, но в первую очередь, в процессе освоения других дисциплин, таких как:

- «Операционные системы и сети ЭВМ»,
- «Базы данных»,
- «Программные и аппаратные средства информатики»,
- «Компьютерная графика»,
- «Методы и средства визуального программирования»,
- «Проектирование программного обеспечения»,
- «Программирование для Интернет»,

- «Объектно-ориентированное программирование»,
- «Имитационное моделирование»,
- «Архитектура ЭВМ и язык ассемблера»,
- «Математические модели компьютерных сетей».

Пособие имеет прикладной характер, что в целом способствует формированию у студентов компетенций в соответствии с требованиями, содержащимися в рабочей программе по дисциплине «Программирование для ЭВМ» и модели компетенций по направлению подготовки «Прикладная математика».

Оглавление

1. Введение

Курсовая работа – важная часть учебного процесса в вузе. Ее суть – освоение студентом навыков самостоятельной работы по изучаемой дисциплине. Курсовая работа является завершающим шагом в освоении трехсеместровой дисциплины «Программирование для ЭВМ». При выполнении работы у студента есть возможность применить полученные во время занятий знания и заполнить обнаруженные пробелы с помощью самостоятельного изучения темы. Однако, необходимо запастись терпением и быть готовым потратить на выполнение работы достаточно времени. Польза курсовой работы заключается также в том, что она позволяет систематизировать знания и закрепить навыки, полученные в процессе изучения дисциплины.

Задача каждого программиста – уметь сформулировать задачу, выбрать метод для ее решения, разработать программу, отладить, выполнить тестирование, увидеть проблему, найти ее причину, самостоятельно принимать решения. Выполнение курсовой работы – первый шаг к этому. При выполнении курсовой работы студент учится самостоятельной разработке программ на языке высокого уровня, отрабатывая навыки формализации и алгоритмизации, а также учится оформлять работу в соответствии с требованиями действующих стандартов.

В настоящей курсовой работе разрабатывается интерактивная система программного обеспечения на основе нелинейных динамических структур. Разработка системы производится в среде Pascal (любой реализации и версии по выбору автора проекта) с использованием методов структурного программирования.

2. Организационно-методические рекомендации

В соответствии с учебным планом подготовки студентов по направлению 01.03.04 «Прикладная математика» (бакалавриат) и рабочей программой по дисциплине «Программирование для ЭВМ» на выполнение и защиту курсовой работы отводится 22 часа.

Задание на выполнение курсовой работы студент получает от преподавателя по окончании курса лекций по дисциплине ориентировочно на двенадцатой неделе семестра. Материал, необходимый для выполнения курсовой работы, уже прочитан и основные лабораторные работы выполнены.

После получения задания студент внимательно с ним знакомится и при необходимости уточняет детали у преподавателя или обращается за разъяснениями. Для более точного понимания задания студенту рекомендуется представить, каким образом будут выглядеть исходные данные и/или результаты работы.

Далее преподаватель назначает консультации по курсовой работе не реже раза в неделю и знакомит студентов с графиком консультаций. На консультации студенты приходят в случае необходимости с конкретными вопросами, в том числе с ошибками в программах. В случае логических ошибок может потребоваться схема алгоритма.

На предпоследней консультации студент демонстрирует работающую программу, а на последней защищает ее (см. п.8). По окончании защиты студент получает оценку за выполнение работы (см. п.8.5), которая проставляется в ведомость и зачетную книжку. Без сданной курсовой работы студент не допускается на экзамен по дисциплине «Программирование для ЭВМ». Отсутствие оценки за курсовую работу по окончании зачетной недели означает наличие одной задолженности и может привести к недопуску к последнему экзамену сессии.

2.1. Цель и задачи курсовой работы

Общее название курсовой работы – «Разработка алгоритмов и программ реализации нелинейной динамической структуры». В отдельных случаях для индивидуальных вариантов название может уточняться или изменяться.

Целью выполнения курсовой работы является освоение основных приемов работы с нелинейной динамической структурой типа «дерево» и «граф», алгоритмов поиска данных, а также освоения основ оформления программной документации.

Приемы работы с нелинейной динамической структурой:

- создание нелинейной динамической структуры типа «дерево»;
- обход нелинейной динамической структурой типа «дерево»;
- удаление «дерева» и из «дерева»;
- поиск в структуре типа «дерево»;
- представление графа в виде массива.

Задачи, решаемые при выполнении курсовой работы:

- разработка структуры программы;
- разработка алгоритмов процедур;
- формирование модуля, содержащего необходимые процедуры обработки исходных данных;
- оформление программы и формирование пользовательского интерфейса;
- отладка программы;
- оформление пояснительной записки, включающей всю необходимую информацию для программиста;

Для решения поставленных задач необходимо разработать алгоритм и программу на языке программирования высокого уровня в соответствии со своим вариантом, проверить ее работоспособность и оформить сопроводительную документацию.

Разработанная программа должна быть хорошо структурирована, разработана с использованием модульной структуры, оформлена с комментариями, проясняющими работы программы, реализована с использованием языка высокого уровня. Пользовательский интерфейс должен быть дружественным. Пояснительная записка оформляется в соответствии с требованиями, изложенными далее.

2.2. Задание на выполнение курсовой работы

Стандартное задание на выполнение курсовой работы состоит из трех частей, причем первую и вторую части объединяет третья часть. Таким образом, выполнив все три части задания необходимо разработать общий модуль и интерфейс для пользователя, позволяющий осуществлять многократную проверку программы (с различными исходными данными).

1. Разработать алгоритмы и программу в соответствии со своим вариантом, реализующую структуру типа «дерево» на языке Паскаль (любая модификация), включающую создание и обработку.

Программа должна содержать процедуры:

- создания структуры типа «дерево»;
- проверки пустоты «дерева»;
- добавления элемента в «дерево»;
- чтения «дерева»;
- удаления из дерева (вне зависимости от варианта);
- обработки «дерева» в соответствии с вариантом.

В программе необходимо использовать динамическое выделение памяти для формирования древовидной структуры.

Процедура чтения дерева должна осуществлять вывод дерева на экран или в файл в виде, отображающем его структуру.

2. Разработать алгоритм и программу поиска в соответствии со своим вариантом.

Программа должна содержать процедуры:

- поиска данных;
- печати результатов.

3. Разработать алгоритм и программу, использующую результаты двух первых программ (использовать модуль).

Задание для вариантов повышенной сложности может отличаться и состоять из произвольного количества разделов. К вариантам повышенной сложности относятся задачи обработки графа, и соответственно, необходимо разработать алгоритм и программу обработки графа, а также модуль с процедурами и пользовательский интерфейс.

2.3. Требования к исходным данным

Ввод массивов исходных данных осуществлять из файла, скаляры – любым удобным способом по выбору разработчика.

Исходные данные должны быть читабельны и подобраны таким образом, чтобы позволяли полностью проверить работоспособность алгоритма.

При подготовке к защите курсовой работы необходимо определить перечень действий пользователя для полного тестирования программы с указанием выбираемых пунктов меню (при его наличии), ответов системе (при запросах). Предварительно можно подготовить несколько вариантов файлов исходных данных, обеспечивающих все возможные ситуации (для подтверждения множественности разработанного алгоритма), и предложить пользователю выбирать файл в режиме диалога.

2.4. Этапы выполнения курсовой работы

Можно выделить следующие этапы выполнения курсовой работы:

- анализ исходных данных и уточнение задания на выполнение курсовой работы;
- разработка алгоритмов выполнения всех задач;
- разработка и отладка программ по первым двум задачам (в некоторых вариантах обе задачи решаются в одной программе, что понятно из контекста задания);
- формирование модуля, содержащего процедуры из разработанных программ, необходимые для решения третьей задачи (допускается включить в модуль все процедуры);
- разработка и отладка программы для решения третьей задачи с использованием модуля;
- разработка пользовательского интерфейса;
- демонстрация работоспособности программы преподавателю (при демонстрации работоспособности необходимо минимизировать данные, вводимые с экрана; использовать различные варианты исходных данных, позволяющие определить работоспособность программы во всех возможных случаях);
- распечатка результатов, программ (в том числе модуля) и исходных данных;
- оформление пояснительной записки в соответствии с требованиями п.5;

- оформление электронного носителя со всеми файлами курсовой работы и пояснительной записки;
- защита курсовой работы (представление работы и ответы на вопросы преподавателя о выполнении работы и ее теоретическим аспектам) проводится только после демонстрации работоспособности программ и при наличии пояснительной записки и электронного носителя.

3. Оформление курсовой работы

3.1. Состав и общие требования к оформлению пояснительной записки

Курсовая работа оформляется в виде пояснительной записки.

Изложение текста и оформление выполняют в соответствии с требованиями с действующим государственным стандартом по оформлению текстовых документов (ГОСТ 7.32-2001). Пояснительная записка должна быть представлена на бумажном и электронном носителе. Оформление текстовой части выполняется на компьютере. С целью обеспечения совместимости с установленным программным обеспечением следует представлять готовые работы в формате MS Word (версия не ниже 6.0), таблицы на отдельных листах могут быть выполнены в формате MS Excel (версия не ниже 5.0).

Печать на одной стороне листа белой бумаги размером 210х297 мм (формат А4). Отдельные таблицы, иллюстрации, распечатки могут быть выполнены на формате А3.

Поля: левое 30 мм, правое не менее 10 мм, верхнее 20 мм, нижнее 20 мм.

Тип шрифта для текста - Times New Roman, прямой. Высота шрифта – 12.

Интервал – 1,5. Абзац (1,25) должен быть одинаковым по всей работе.

Рисунки должны иметь сквозную нумерацию и подрисуночные подписи.

Таблицы должны иметь сквозную нумерацию и название. Например, «Таблица 1. Переменные процедуры “Obrabotka”» сверху таблицы.

Пояснительная записка должна содержать:

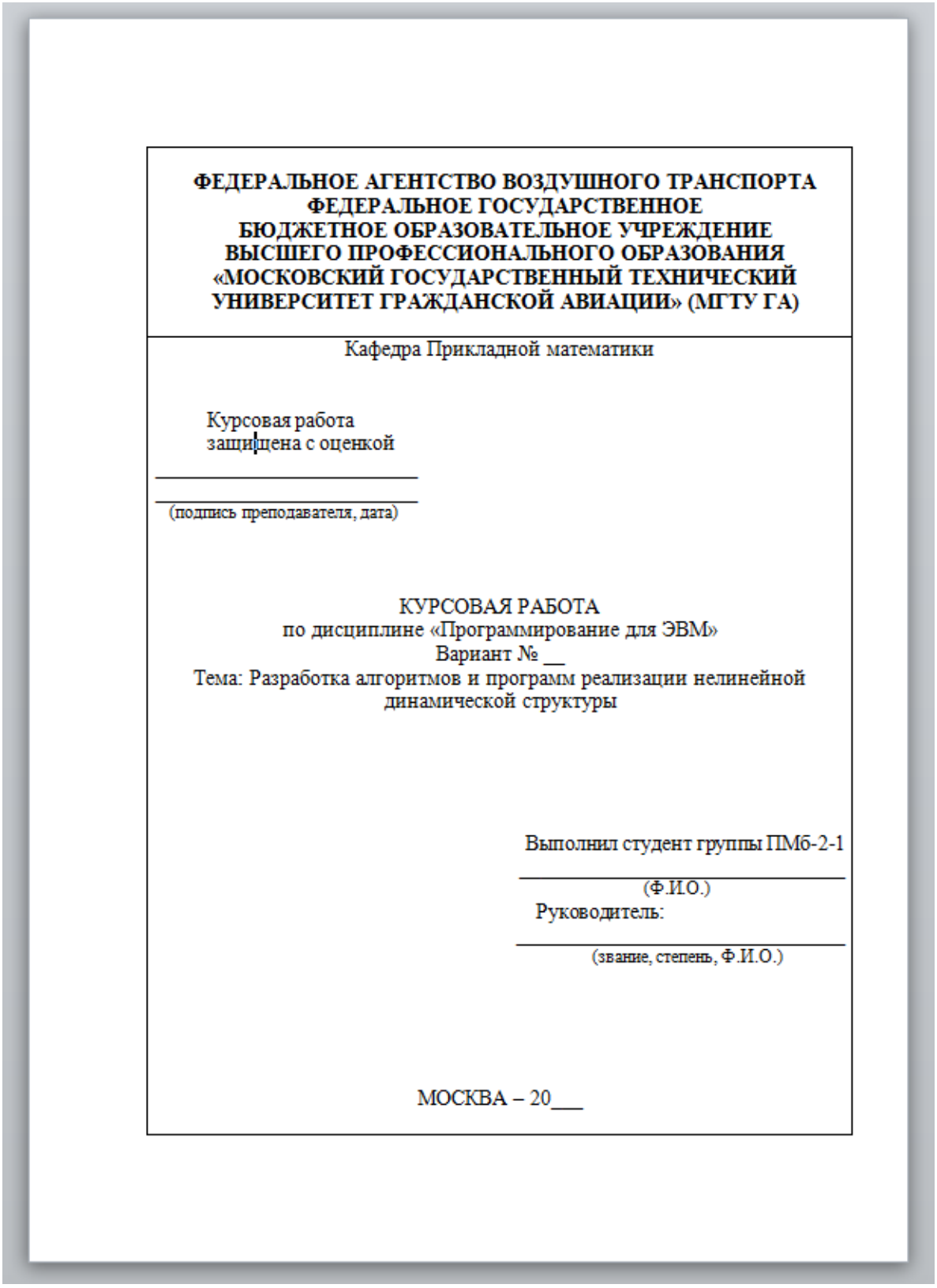
- титульный лист (рекомендуемая форма – в п.3.2);
- содержание (на отдельном листе) с указанием номеров страниц (нумерация листов – автоматическая, содержание - автособираемое);
- цель и задачи курсовой работы;
- задание на выполнение курсовой работы (адаптированное для своего варианта);
- краткое описание курсовой работы (аннотация);
- алгоритмы всех процедур (в виде схем или для линейных алгоритмов возможна словесная форма описания), разработанных самостоятельно;
- таблицу глобальных переменных программы с наименованиями, назначением и типом переменных;
- таблицы переменных процедур (наименование, назначение, тип);
- структуру программы (программ) в виде рисунка направленного графа, отображающего вызовы всех подпрограмм программы;
- таблицу подпрограмм (библиотек программы) с наименованиями, назначениями и страницами приложений на которых представлен их текст;
- руководство пользователя программы;
- распечатки файлов исходных данных и результатов (в случае вывода результатов на экран – скриншоты);
- список использованной литературы (включая ссылки на публикации в Интернете);
- листинги программ.

Пояснительная записка может быть структурирована в соответствии с этапами выполнения курсовой работы или с последовательностью написания программ.

Листы пояснительной записки должны быть жестко скреплены (степлером, скоросшивателем, ниткой). Скрепка, папка-уголок или файл недопустимы.

3.2. Оформление титульного листа

Рекомендуемая форма титульного листа представлена на рисунке ниже.



ФЕДЕРАЛЬНОЕ АГЕНТСТВО ВОЗДУШНОГО ТРАНСПОРТА
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ГРАЖДАНСКОЙ АВИАЦИИ» (МГТУ ГА)

Кафедра Прикладной математики

Курсовая работа
защищена с оценкой

_____
_____
(подпись преподавателя, дата)

КУРСОВАЯ РАБОТА
по дисциплине «Программирование для ЭВМ»
Вариант № __
Тема: Разработка алгоритмов и программ реализации нелинейной динамической структуры

Выполнил студент группы ПМб-2-1
_____
(Ф.И.О.)
Руководитель:
_____
(звание, степень, Ф.И.О.)

МОСКВА – 20___

3.3. Содержание разделов пояснительной записки

Автособираемое содержание оформляется на отдельном листе с указанием номеров страниц. Разделы нумеруются арабскими цифрами, начиная с 1. Глубина вложения – не более трех элементов (например, 1.1.3). Рекомендуемая глубина – 2.

Цель и задачи курсовой работы определены в настоящем пособии. Задачи должны быть подкорректированы с учетом индивидуального варианта.

Задание на выполнение курсовой работы должно быть близко к тому, которое представлено в п.2.2, но слова «в соответствии со своим вариантом» должны быть заменены текстом задания из п.7.

Краткое описание курсовой работы (или «аннотация») обычно размещается до содержания. Тогда аннотация выполняется на отдельном листе. Но в данном случае разработка аннотации является частью курсовой работы и может быть самостоятельным разделом. В аннотации кратко, тремя – четырьмя предложениями раскрывается суть и основные результаты, полученные при выполнении курсовой работы.

Схемы алгоритмов желательно оформлять с использованием программных инструментов (на персональном компьютере), но допустимо и чертить от руки с соблюдением пропорций схем и корректного вхождения соединительных линий. Использование линейки в этом случае – обязательно. Общие правила выполнения схем, а также список графических символов, из которых составляются схемы, и их назначение представлены в [4].

По каждому пункту задания по каждому алгоритму должно быть указано функциональное назначение алгоритма. Описаны входные и выходные данные (в таблице). Представлена таблица локальных переменных алгоритма и его формальных параметров, содержащая имена переменных, их тип и назначение. Глобальные переменные, используемые в алгоритме, представляются списком и ссылкой на таблицу, в которой указано их назначение. Должна быть также дана ссылка на страницу приложения, на которой расположен текст функции данного алгоритма.

Таблица глобальных переменных программы аналогична таблицам локальных переменных. В ней указываются все переменные, объявленные в разделе Var основной программы, а также именованные константы.

Для большей читабельности программ (для удобства отладки и сопровождения) в технической документации отражается структура программы (программ) в виде рисунка направленного графа, отображающего вызовы всех процедур и функций программы (с учетом вложений), а также таблицу подпрограмм (библиотек программы) с наименованиями, назначениями и страницами приложений на которых представлен их текст.

Руководство пользователя программы (или Руководство по использованию программы) должен содержать:

- правила установки, запуска и использования программы;
- перечень файлов, необходимых для запуска программы;
- правила подготовки файлов с исходными данными и файлов для тестирования вариантов обработки данных;
- правила пользования системой диалога (запросами меню);
- формы результатов действий пользователя по каждому виду работы с системой (ссылки на вид экрана с результатами диалога, представленные в приложении).

Раздел «Список использованной литературы» включает все используемые источники: монографии, учебники, пособия, справочники, статьи из периодических изданий, электронные пособия и статьи.

Сведения о книгах должны включать: фамилию и инициалы автора, заглавие книги, место издания, издательство и год издания. Сведения о статье должны включать: фамилию и

инициалы автора, наименование статьи, наименование издания (журнала), наименование серии (если есть), год выпуска, том (если есть), номер издания и номера страниц, на которых помещена статья. Электронные учебники и статьи должны включать ссылки на используемый электронный ресурс.

При ссылке на источник надо приводить в квадратных скобках его порядковый номер по списку литературы, например: [3].

3.4. Оформление приложения

Листинги программ, а также распечатки файлов исходных данных и результатов (в случае вывода результатов на экран – скриншоты) оформляются в виде Приложения. Каждый файл в приложении оформляется как листинг с наименованием файла и его назначением.

Кроме того в Приложение следует поместить скриншоты экрана с различными вариантами диалога пользователя и, при наличии, скриншоты экрана с графическими результатами работы программы.

Каждый раздел “Приложение” должен начинаться с новой страницы с написания в правом углу слова ПРИЛОЖЕНИЕ прописными буквами и иметь заголовок. При наличии более одного приложения все приложения нумеруются арабскими цифрами (ПРИЛОЖЕНИЕ 1, ПРИЛОЖЕНИЕ 2 и т.д.) или обозначаются заглавными буквами русского или латинского алфавита (ПРИЛОЖЕНИЕ А, и т. д.). Например:

ПРИЛОЖЕНИЕ А.

Далее (строчкой ниже) следует заголовок этого приложения.

Листинги программ сопровождаются комментариями. Комментарии должны сопровождать все значимые операторы программ.

Рисунки, помещенные в приложении, нумеруются арабскими цифрами в пределах каждого приложения. Например, третий рисунок первого приложения нумеруется (рисунок П.1.3).

4. Краткие теоретические сведения. Дерево

4.1. Определение дерева

Дерево - это нелинейная связная структура, которая состоит из узлов, соединенных между собой указателями. Дерево имеет корень - узел, из которого связи только выходят, в который они не входят.

Бинарное дерево - это конечное множество элементов, которое либо пусто, либо содержит один элемент, называемый корнем дерева, а остальные элементы множества делятся на два непересекающихся подмножества, каждое из которых само является бинарным деревом. Эти подмножества называются левым и правым поддеревьями (ветвями) исходного дерева.

Если A - корень бинарного дерева и B - корень его левого или правого поддерева, то говорят, что A - отец B, а B - левый или правый сын A.

Узел, не имеющий сыновей, называется листом. Два узла являются братьями, если они сыновья одного и того же отца.

Уровень узла в бинарном дереве определяется следующим образом: корень дерева имеет уровень 0, уровень любого другого узла дерева на 1 больше уровня его отца.

Глубина бинарного дерева - максимальный уровень листа дерева, что равно длине самого длинного пути от корня к листу дерева.

Объявление узла бинарного дерева выглядит следующим образом:

```
Type
        Node =  ^Bder;
        Bder = record
                info : integer;  { <- информация узла дерева }
                left, right : Node;       { <- указатели на сыновей дерева }
                  end;
```

4.2. Операции над деревом

К дереву можно применить следующие операции:

- Создание нового узла бинарного дерева.
- Создание левого (правого) сына.
- Обход дерева.
- Определение указателя на узел дерева.
- Удаление узла дерева.
- Удаление дерева.

Рассмотрим их подробнее.

Создание нового узла бинарного дерева:

```
Function NewD ( X : integer): Node;
        Var P : Node;
begin
        New (P);                P^.info := X;
        P^.left := Nil;  P^.right := Nil;
        NewD :=P;               end;
```

Создание левого сына

```
Procedure SetLeft ( P: Node, X : integer);
Begin  P^.left := NewD (X); end;
```

Правый сын создается аналогично.

Удаление дерева

```
Procedure DeleteDer (P: Node);
Begin If  P <> Nil then begin
        DeleteDer (P^.left); { <- переход к левому сыну при наличии }
        Writeln (P^.info);       { <- печать узла }
        DeleteDer (P^.right);{ <- переход к правому сыну при наличии }
        Dispose (P); P:= Nil;  end;
End;
```

Удаление дерева – рекурсивная процедура, выполняющая постепенный обход дерева и по пути удаляющая элементы. В данном случае процедура печати не имеет отношения к удалению, а просто предназначена для отображения последовательности удаления узлов.

4.3. Обход дерева

Многие алгоритмы и процессы, использующие бинарные деревья распадаются на две фазы: сначала строится бинарное дерево, а затем оно обходится. Существует три метода обхода дерева. Методы определяются рекурсивно так, что прохождение бинарного дерева

требует посещения корня и прохождения его левого и правого поддеревьев. Методы отличаются лишь порядком, в котором эти три действия выполняются.

Прямой порядок:

- Попасть в корень.
- Пройти в прямом порядке левое поддерево.
- Пройти в прямом порядке правое поддерево.

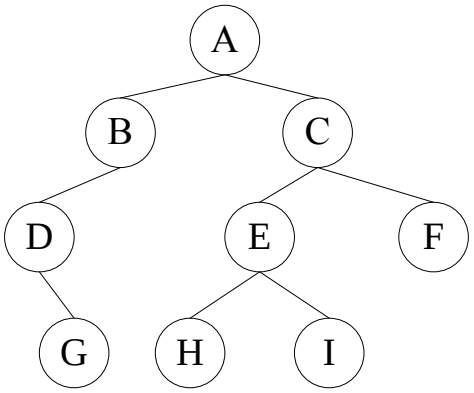
Симметричный порядок:

- Пройти в симметричном порядке левое поддерево.
- Попасть в корень.
- Пройти в симметричном порядке правое поддерево.

Обратный порядок:

- Пройти в обратном порядке левое поддерево.
- Пройти в обратном порядке правое поддерево.
- Попасть в корень.

Рассмотрим пример.



Для дерева на рисунке:

- прямой порядок обхода: A B D G C E H I F;
- симметричный порядок обхода: D G B A H E I C F;
- обратный порядок обхода: G D B H I E F C A.

Заметим, что прямой порядок обхода всегда начинается с корня, а обратный корнем заканчивается.

Обход дерева в симметричном порядке используют как один из методов сортировки данных. Процедура аналогична предыдущей процедуре удаления дерева:

```
Procedure PrintDer (P: Node);
Begin if  P <> Nil then begin
        PrintDer (P^.left); { <- переход к левому сыну при его наличии }
        Writeln (P^.info);      { <- печать узла }
        PrintDer (P^.right); { <- переход к правому сыну при его наличии }
                        end;
end;
```

При прямом и обратном порядках обхода меняется последовательность операторов. Оператор вывода становится соответственно первым и последним оператором в этой процедуре.

4.4. Удаление узла из бинарного дерева

Удаление элемента из упорядоченного дерева реализуется достаточно просто, если этот элемент является листом или у него только один сын. Для этого нужно изменить

соответствующую ссылку у предшествующего элемента. Если же из удаляемой вершины выходят две ветви, то нужно найти подходящий элемент, который можно было бы вставить на место удаляемого. Таким образом, процедура удаления должна различать три случая:

- удаляется лист;
- удаляется узел, имеющий одного сына;
- удаляется узел, имеющий двух сыновей.

Процедура удаления также рекурсивна.

В том случае, если удаляется лист, то память просто освобождается от этого элемента. Если есть только один сын, то он ставится на место удаляемого. При наличии двух сыновей удаления происходит постепенно путем подтягивания узлов «через уровень». В зависимости от алгоритма подтягиваться может либо «правый сын левого сына», либо «левый сын правого сына».

```
Procedure DelTree(x: Integer; Var p: Node);
Var q: Node;
{ процедура поиска подходящего узла и вставка его на место удаляемого }
Procedure D1 (Var r: Node);
Begin
        if r^.Right <> Nil      Then D1(r^.Right)
                                else
                                begin
                                   q^.Data:=r^.Data;
                                   q:=r;
                                   r:=r^.Left;
                                   Dispose (q);
                                end;
end;
begin   { поиск узла для удаления}
if p = Nil then Writeln (‘ Требуемого элемента в списке нет’)
                  else If x < p^.Data Then DelTree (x, p^.Left)
                  else if x > p^.Data Then DelTree (x, p^.Right)
                  else   { узел найден }
                        begin   { удаление узлов? имеющих одного сына}
                                q:=p;
                                if q^.Right = Nil then
                                        begin
                                        p:=q^.Left;
                                        Dispose (q);
                                        end
                        else
                        if q^.Left = Nil then begin
                                        p:=q^.Right;
                                        Dispose (q);
                                                end
                        else D1(q^.Left)        { у узла два сына, удаление постепенно}
                        end;
end;
```

4.5. Применение бинарных деревьев

Бинарное дерево – полезная структура, когда в каждой точке процесса должно быть принято одно решение из двух возможных. Например, при поиске данных требуется большое количество сравнений. Многократное их уменьшение – построение дерева сравнений (п.6.5 в настоящем пособии).

Сортировка данных с использованием дерева – один из самых распространенных алгоритмов: необходимо построить упорядоченное множество и пройти его в симметричном порядке. При этом дерево может быть перестроено из другого дерева, если ключ для упорядочивания другой. На большом объёме данных эффект может быть вполне чувствительным.

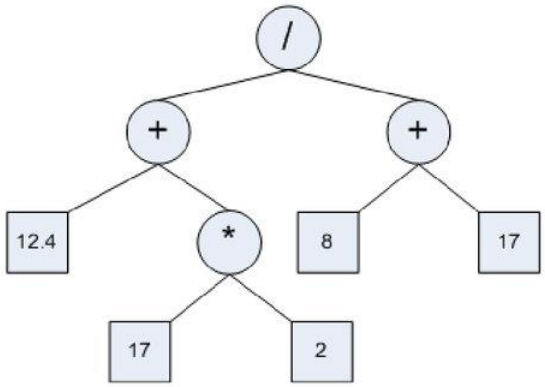
В упорядоченном дереве легко решается задача поиска минимума и максимума (при определенном ранее ключе). Легко догадаться, что минимальным значением ключа будет самый левый потомок корня, а максимальным – самый правый. При поддержании такой структуры задачи поиска минимума и максимума становятся тривиальными.

Еще одна типичная задача – поиск дубликатов. Поскольку каждый новый элемент необходимо сравнить со всеми элементами, то как и в случае поиска, количество сравнений минимизируется (особенно в сбалансированном дереве).

Как развитие задачи на поиск дубликатов можно рассматривать задачу построения частотного словаря: определение количества повторений заданного слова (символа) в заданной последовательности (тексте/файле/строке). Помимо самих символов элементом дерева становится и количество повторений, которое при занесении элемента в дерево равно 1. При обнаружении дубликата элемент в дерево не вносится, увеличивается количество его повторений.

В качестве другого применения бинарных деревьев можно рассмотреть метод представления выражения, содержащего операнду и операцию (оператор) в виде строгого бинарного дерева. Узел, представляющий операцию, содержит два непустых поддерева – операторы, к которым должна быть применена операция. При этом не требуется скобок.

Пример записи выражения $(12.4+17*2)/(8+17)$ представлен на рисунке.



Если пройти такое дерево в прямом порядке, получится выражение в инфиксной форме, если в прямом, то - в префиксной, в обратном – в постфиксной. Заметим, что в выражении в инфиксной форме могут потребоваться скобки. В любом случае, использовать дерево для организации вычисления выражений не только допустимо, но и эффективно.

Рассмотрим еще один пример применения двоичных - кодировка методом Хаффмена (или Хаффмана) или построение деревьев Хаффмена. Рассмотрим следующую проблему. Допустим, у нас есть алфавит из N символов и длинное сообщение, состоящее из этих символов. Мы хотим закодировать сообщение в виде строки битов, присвоив каждому символу определенную последовательность из равного количества бит. Однако такая кодировка не будет эффективной, если ряд символов встречается часто, а некоторые - совсем

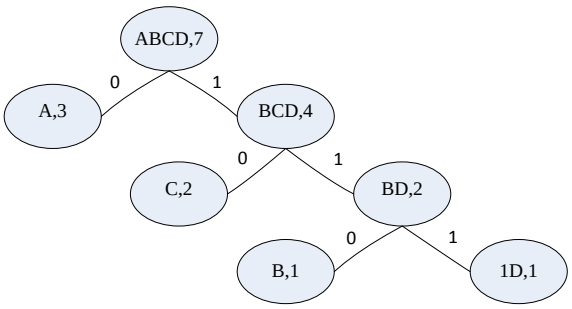
редко. Хаффмен предложил для чаще встречающихся символов использовать более короткий код, а для редко используемых - длинный. Для этого должна быть статистикой по использованию всех символов алфавита. Частота появления символов либо вычисляется с применением «частотного словаря», либо используется стандартная, соответствующая языку, на котором написан кодируемый текст (например, используются данные, лежащие в основе клавиатуры).

Существует два подхода к построению такого дерева: «сверху вниз» и «снизу вверх». «Сверху вниз» строим бинарное дерево, где вершиной будет полный алфавит, а затем последовательно делим на поддеревья так, чтобы частота использования символов в каждом поддереве была примерно одинакова, причем в левом поддереве каждого узла меньше или равно правому. Так поступаем до тех пор, пока листьями дерева не окажутся отдельные символы. Формируем код, каждому левому поддереву присваивая значение 0, а правому - 1.

Рассмотрим пример. Пусть полный алфавит - 4 символа: A,B.C и D. Частота их использования приведена в таблице.

Символ	Частота использования
A	3
B	1
C	2
D	1

Дерево Хаффмена примет вид:



Последовательно обходя дерево, получаем следующую кодировку:

Символ	Частота использования	Код
A	3	0
B	1	110
C	2	10
D	1	111

Необходимо отметить, что при использовании кодов переменной длины, код одного символа не должен совпадать с началом кода другого символа, что хорошо видно из примера (префиксные коды).

При кодировке равным количеством символом кодируемая последовательность требовала 14 бит (при том, что каждый символ кодировался двумя битами). При кодировке методом Хаффмена требуется только 13 бит. При длинных последовательностях выигрыш может быть очень большим.

При методе «снизу вверх». Из элементов алфавита формируется список в порядке возрастания частоты их появления. Каждые два первые элемента списка объединяются в дерево, в котором они являются листами, а их объединение – корнем. При этом частота их появления складывается. Получившееся дерево встраивается в список. (Фактически на этапе формирования кода мы имеем дело с лесом). В результате получим список, состоящий из

одного элемента, который и будет требуемым деревом. Обход его по приведенным ранее правилам и даст необходимый код.

Полная программа разработки кода Хаффмена приведена в [3].

5. Краткие теоретические сведения. Граф

5.1. Определение и представление графа

Граф состоит из множества вершин и множества дуг, вершины также называют узлами, а дуги ребрами. Дуга представляется в виде пары вершин.

Вершины графа можно использовать для представления объектов, а дуги для отношений между объектами (например, вершины – города, а дуги – дороги между городами).

Если пары узлов упорядочены, то граф называют направленным или ориентированным. Если с дугами графа связано какое-либо значение, то граф называют взвешенным или помеченным. Если граф содержит циклы, то он называется циклическим.

Для представления графов можно использовать различные структуры данных. Выбор структуры зависит от операторов, которые будут применяться к вершинам и дугам графа. Одним из наиболее общих представлений является матрица смежности.

Матрица смежностей неориентированного графа - это матрица, в которой при наличии ребра между вершинами ставится единица, а при отсутствии – ноль. Представляется в виде квадратного двумерного массива, размер которого определяется количеством вершин. Легко видеть, что матрица смежностей неориентированного графа всегда симметрична, поэтому для ее представления достаточно хранить только ее верхний треугольник. Т.е. можно использовать алгоритмы работы с треугольными массивами.

Матрица смежностей ориентированного графа - это также квадратная матрица, но единицы ставятся только в том случае, если есть связь между вершинами в определенном направлении. А значит, симметричность отсутствует. Обработке подлежит полный массив.

В качестве единиц и нулей могут использоваться как арифметические константы, так и логические.

Для помеченных графов матрица смежности будет заполняться не единицами, а метками (т.е. соответствующими значениями дуг). Вместо нулей будут использоваться специальные символы, означающие, что клетка пуста. Ноль в данном случае допустим также, но только в том случае, если он ни при каких условиях не может оказаться значением, приписанным дуге (ребру).

Представление графа с помощью матрицы смежностей не всегда удобно, поскольку количество вершин требуется знать заранее. Если граф должен создаваться или изменяться во время исполнения программы, то для каждого добавления или удаления вершины необходимо строить новую матрицу. Кроме того, даже если граф содержит малое число ребер (дуг) и матрица смежностей состоит в основном из нулей, память должна быть отведена для всех возможных дуг вне зависимости от того, существуют ли они. Если граф содержит N вершин, то должна быть отведена память для N*2 элементов. Возможное решение - это использовать динамическую (связанную) структуру данных для представления графа. В этом случае будет более удобным применение классической структуры, называемой списком смежности.

Список смежности содержит для каждой вершины графа список смежных ей вершин. Каждый элемент такого списка является записью R, содержащей в поле R.Stroka вершину графа, а в поле R.Next - указатель на следующую запись в списке (для последней записи в списке R.Next содержит Nil).

5.2. Использование массивов и списков для представления графа

Массивы – наиболее часто используемые структуры данных и во многих случаях самые удобные. Но при этом далеко не всегда это оптимально с точки зрения экономии ресурсов ПК. Так, например, реализация специальных типов массивов нестандартными средствами может привести к значительной экономии оперативной памяти. Некоторые системы программирования имеют специальные подпрограммы для работы с такими массивами.

Треугольные массивы.

Предположим, у нас есть информация об N городах и расстояниях между ними. При помощи массива можно построить матрицу смежности, хранящую информацию о расстоянии между городами, но совершенно очевидно, что на диагонали элементы не будут иметь значений, а остальная информация будет дублирована, т.к. A[I,J] = A [J,I]. Такие массивы и называют треугольными.

При больших значениях N излишние затраты памяти могут быть существенны. Для N городов будет N*(N-1) дублированных элементов и N подобных A[I,I]. Если N=1000, то в массиве будет храниться более 500 тысяч ненужных элементов.

Избежать таких потерь можно, создав одномерный массив и упаковав в него двумерный.

Формула для преобразования A[I,J] в B[X] имеет следующий вид:

X := round (I*(I-1)/2)+J для I > J.

В данном случае нумерация элементов массивов начинается с 0. В этом случае все используемые формулы получаются более простыми.

Диагональные массивы.

В некоторых случаях используются треугольные массивы, использующие диагональные элементы. Такие массивы и называются диагональными. В этом случае требуется сделать два изменения в формуле преобразования индексов:

Преобразование не должно отклонять случаи I = J,

Перед вычислением индекса в массиве B необходимо добавить к I единицу.

Нерегулярные массивы.

В некоторых программах требуются массивы с нестандартной формой. Например, в каждой строке двумерного массива нужно хранить разное количество элементов (например, координаты вершин многоугольников, каждый из которых имеет разное количество вершин).

Можно было бы выровнять такой массив по большему количеству элементов, но в этом случае возможны потери памяти, необходимость хранения количества элементов в каждой строке обуславливает использование второй сложной структуры, а также разработку системы их связывания.

Нерегулярные многомерные массивы организуют с помощью двух массивов меньшей размерности либо массива и связанного списка.

Например, двумерный массив можно представить как два одномерных: один содержит все элементы двумерного, в котором элементы строк размещаются последовательно друг за другом, а второй – номера элементов в первом массиве, где начинается каждая строка. Для удобства обработки во второй массив можно поместить последний элемент, который укажет на элемент, следующий за последним.

Второй случай аналогичен, но вместо номера первого элемента каждой строки нужно хранить во втором массиве адреса в списке первых элементов строк.

Разреженные массивы.

Иногда программы используют большие массивы, которые содержат много нулевых элементов. Такие массивы называют разреженными. При более 50% ненулевых элементов массивы также называют сильно разреженными.

Для организации многомерных разреженных массивов используется одномерный массив, содержащий значения указателей на списки, где элементами списков являются элементы каждой строки массива или, при большей исходной размерности, сами являются указателями на списки. Последнее ввиду сложности обработки используется редко. Разреженные массивы размерности больше двух практически в реальном программировании не встречаются.

В случае сильно разреженных массивов предполагается, что массив вообще может содержать пустые строки, и в этом случае его обработка через массив списков не оптимальна. Массив можно заменить списком. В таком случае будет обработка списка списков. На эту обработку распространяются все правила рассмотренные ранее в разделе о линейных связных структурах. Основное отличие – пересчет индексов в порядковые номера в списках.

5.3. Обход графа

Рассмотрим задачу реализации алгоритмов поиска в ширину и поиска в глубину в неориентированном графе.

Граф представлен в виде матрицы смежности:

smegh : array [1..N, 1..N] of byte.

Массив M : array [1..N] of char используется для хранения вершин графа.

Массив visit : array [1..N] of boolean; { - массив маркеров о посещении вершин }

Процедура обхода графа в глубину:

```
procedure deep( v : integer);
var i : integer;
begin
 visit[v] := true;  { - вершина посещена }
 write(fr, M[v], ' ');  { - вывод символа из вершины }
{ Просмотр непосещенных вершин, смежных с вершиной v }
 for i := 1 to N do
  if not(visit[i]) and (smegh[v,i] = 1) then deep(i) ;
end;
```

Процедура обхода графа в ширину:

```
procedure width( v : integer);
var x, y, i : integer;
begin
 visit[v] := true; { - вершина посещена }
 write(fr, M[v], ' ');
 add(Och, v);
 while Och <> nil do { - очередь не пуста }
  begin
   x := Och^.no;
   del(Och);
   { Перебор непосещенных вершин, смежных с вершиной x }
```

```
  for i := 1 to N do
   if not(visit[i]) and (smegh[x, i] = 1) then
    begin
     visit[i] := true;  { - посетили вершину }
     write(fr, M[i], '  ');
     add(Och, i);
    end;
  end;
end;
```

При матрице смежности:

	a	b	c	d	e	f	g
a	1	1	1	1	1	0	0
b	1	1	0	1	1	0	0
c	1	0	1	0	0	1	1
d	1	1	0	1	1	0	0
e	1	1	0	1	1	0	0
f	0	0	1	0	0	1	1
g	0	0	1	0	0	1	1

Результаты обхода будут:

Поиск в глубину:	Поиск в ширину:
a b d e c f g	a b c d e f g

5.4. Построение остовного дерева графа

Под остовным деревом графа понимается свободное дерево, содержащее все вершины из исходного графа (а ребра не все). Учитывая, что в ориентированном графе направление дуги имеет значение, удаление ребер в таком графе сложнее. Поэтому обычно говоря об остовном дереве, имеют в виду неориентированный граф. Кроме того, чаще всего задача ставится с учетом меток (весов дуг). Т.е. задача решается для неориентированного помеченного (взвешенного) графа. И звучит она следующим образом: «Сформировать остовное дерево минимальной длины». Иногда вместо «длины» используют термин «стоимость» или «вес».

В прикладной постановке задачу можно сформулировать так:

«Дана плоская страна и в ней N городов. Нужно соединить все города телефонной связью так, чтобы общая длина телефонных линий была минимальной».

Минимальное остовное дерево позволяет получить Алгоритм Прима–Краскала (или Крускала), разработанный в начале 60-х годов прошлого века. Рассмотрим алгоритм кратко:

- раскрашиваем вершины графа в разные цвета;
- находим ребро минимальной длины и включаем его в остов. Соединенные вершины остова перекрашиваем в один цвет;
- находим следующее ребро минимальной длины, не входящее в остов, и включаем его в остов. Все вершины, связанные с ребром, перекрашиваем в один цвет;
- повторяем; все вершины графа связаны ребрами – мы получили искомый остов.

Результатом работы алгоритма является набор из N-1 ребер. Они не образуют цикла, так как на каждом из N-1 шагов соединялись вершины разного цвета, т.е. ранее не связанные. Этот граф связный, потому что после проведения 1-го ребра осталось n-1 разных цветов, …, после проведения (N-1)-го ребра остался один цвет, т.е. одна компонента связности. Итак,

полученный набор ребер образует связный граф без циклов, содержащий N-1 ребер и N вершин. Следовательно, граф есть остовное дерево.

Для реализации алгоритма понадобятся:

Matrix	матрица расстояний, значение пересечении i-ой строки и j-го столбца равно расстоянию между i-ой и j-ой вершинами. Если такого ребра нет, то значение равно Fls – просто большому числу
Color	массив цветов вершин
Ribs	массив, в котором запоминаются найденные ребра
a, b	вершины, соединяемые очередным минимальным ребром
len	длина дерева

Матрицу расстояний будем хранить в текстовом файле, где число на первой строке – количество вершин n, а остальные n строк по n чисел в каждой – матрица расстояний. Если расстояние равно 1000 (Fls), то такого ребра нет.

```
Program Prima-Krascala;
Const MaxSize =100;
                Fls =1000;
Var   Matrix: array[1 MaxSize, 1 MaxSize] of integer;
       Color: array[1 MaxSize] of integer;
       Ribs: array[1 MaxSize] of record
                                  a, b: integer;
                                        end;
        n, a, b, k, col, i, len: integer;
Procedure Init;
{ввод исходных данных и подготовка начальных значений}
Var   f: text;
      i, j: integer;
Begin
   Assign(f, 'INPUT.TXT');  Reset(f);
   Readln(f, n);
   For i:=1 to n do Begin
         For j:=1 to n do read(f, matrix[i, j]);
                    Readln(f)
                       End;
    For i:=1 to n do color[i]:=i;
    len:=0
End;

{поиск минимальной длины}
Procedure Findmin(var a, b: integer);
Var min, i, j: integer;
Begin
   min:=Fls;
   For i:=1 to n-1 do
         For j:=i+1 to n do
            If (Matrix[i, j]<min) and (color[i]<>color[j]) then begin
                                                 min:=Matrix[i, j];
```

```
                                        a:=i;
                                        b:=j
                                                    end;
    len:=len+min
end;

{основная программа}
Begin     Init;
     For k:=1 to n-1 do begin
           Findmin(a, b);
           Ribs[k].a:=a;
           Ribs[k].b:=b;
           col:=Color[b];
           For i:=1 to n do
                 If color[i]=col then color[i]:=color[a];
                       end;
     For i:=1 to n-1 do
              Writeln(ribs[i].a, ' –', ribs[i].b);
     Writeln('Length= ', len);
End.
```

5.5. Задачи нахождения кратчайших путей

К типичным задачам для реализации графа относят и задачу нахождения кратчайших путей. Задача решается на основе алгоритмов Флойда и Дейкстры, но алгоритм Дейкстры (иногда в русской литературе встречается вариант Дийкстры, т.к. Dijkstra) обычно используют для решения задачи нахождения кратчайших путей от заданной вершины до всех остальных.

Для программной реализации алгоритма понадобиться два массива: логический visited – для хранения информации о посещенных вершинах и численный distance, в который будут заноситься найденные кратчайшие пути. Итак, имеется граф G=(V, E). Каждая из вершин входящих во множество V, изначально отмечена как не посещенная, т. е. элементам массива visited присвоено значение false.

Поскольку самые выгодные пути только предстоит найти, в каждый элемент вектора distance записывается такое число, которое заведомо больше любого потенциального пути (обычно это максимальное значение конкретного типа данных). В качестве исходного пункта выбирается вершина s и ей приписывается нулевой путь: distance[s]=0.

Далее, находятся все соседние вершины (в которые есть ребро из s) [пусть таковыми будут t и u] и поочередно исследуются, а именно вычисляется стоимость маршрута из s поочередно в каждую из них:

- distance[t]=distance[s]+вес_инцидентного_s_и_t_ребра;
- distance[u]=distance[s]+ вес_инцидентного_s_и_u_ребра.

Вполне вероятно, что в некоторую вершину из s существует несколько путей, поэтому длину пути в такую вершину в массиве distance придется пересматривать, тогда наибольшее (неоптимальное) значение игнорируется, а наименьшее ставиться в соответствие вершине.

После обработки смежных с s вершин она помечается как посещенная: visited[s]=true, и активной становится та вершина, путь из s в которую минимален. Вершина становиться активной и аналогичным образом исследуются соседние вершины (за исключением вершины

s). Алгоритм Дейкстры продолжается до тех пор, пока все доступные из s вершины не будут исследованы.

```
program DijkstraAll;
    const N=6; inf=100000;
    type vektor=array[1..N] of integer;
              tm = array[1..N, 1..N] of integer;
    var start: integer;
// задается матрица начальных значений размера NxN.
    const GR: tm = ({здесь данные}…);
procedure Dijkstra(GR: tm; st: integer);
    var count, index, i, u, m, min: integer;
        distance: vektor;
        visited: array[1..N] of boolean;
begin  m:=st;
        for i:=1 to N do begin distance[i]:=inf; visited[i]:=false; end;
        distance[st]:=0;
        for count:=1 to N-1 do begin min:=inf;
                                        for i:=1 to N do
                                          if (not visited[i]) and (distance[i]<=min) then begin
                                                                  min:=distance[i]; index:=i; end;
        u:=index; visited[u]:=true;
        for i:=1 to N do if (not visited[i]) and (GR[u, i]<>0) and (distance[u]<>inf) and
                                (distance[u]+GR[u, i]<distance[i]) then
                                                        distance[i]:=distance[u]+GR[u, i];
                        end;
        write('Стоимость пути из начальной вершины до остальных:'); writeln;
        for i:=1 to N do if distance[i]<>inf then writeln(m,' > ', i,' = ', distance[i])
                                    else writeln(m,' -> ', i,' = ', 'маршрут недоступен');
end;
{основной блок программы}
begin
  write('Начальная вершина >> '); read(start);
  Dijkstra(GR, start);
end.
```

Алгоритм Флойда имеет более общий характер, а именно, он позволяет определить все расстояния между вершинами. При этом его реализация даже проще. Реализация алгоритма Флойда:

```
Procedure Floyd (var a: massiv; c: massiv);
   Var I, j, k : integer;
   Begin for I := 1 to N do
              for J := 1 to N do
                    A[I,J] := C[I,J];
     for I := 1 to N do A [I,I] := 0;
     for K := 1 to N do
            for I := 1 to N do
                  for j := 1 to N do
```

```
                    if A[I,K] + A[K,J] < A [I,J] then
                            A [I,J] :=  A[I,K] + A[K,J] ;
  End;
```

В этой процедуре masssiv – тип: array [1..N, 1.. N] of real.

Вначале A[I,J] := C[I,J], а диагональные элементы равны нулю. Если дуга отсутствует, то на это место ставят очень большую величину (условно считая ее бесконечностью). Тогда расстояние между такими вершинами будет формироваться с использованием промежуточной вершины, а возможно и нескольких.

Во многих случаях требуется не только определить кратчайшее расстояние между узлами, но и получить кратчайшие пути, что приведенный выше алгоритм сделать не позволял. Для того, чтобы получить пути, нужно ввести еще один дополнительный массив P, в которой каждый элемент P[I,J] содержит вершину K, полученную при прохождении наименьшего значения A[I,J]. Если P[I,J] = 0, то кратчайший путь состоит из одной дуги.

Таким образом, задача нахождения кратчайших путей решается с использованием модифицированного алгоритма Флойда:

```
Procedure Floyd (var a: massiv; c: massiv, p: massiv2);
  Var I, j, k : integer;
  Begin for I := 1 to N do
            for J := 1 to N do begin
                  A[I,J] := C[I,J];
                  P[I,J] := 0;
    for I := 1 to N do A [I,I] := 0;
    for K := 1 to N do
          for I := 1 to N do
                for j := 1 to N do
                    if A[I,K] + A[K,J] < A [I,J] then begin
                            A [I,J] :=  A[I,K] + A[K,J] ;
                             P[I,J] = K                    end;
  End;
```

Для печати получившейся матрицы P можно использовать рекурсивную процедуру:

```
Procedure pechat (I,J : integer);
    Var K : integer;
    Begin
        K:=P [I,J];
        If k = 0 then return;
        Pechat (I,K);
        Writeln (K);
        Pechat (K,J);
End;
```

В этой процедуре I и J – две вершины расстояние между которыми должно быть выведено.

Но во многих задачах интерес представляет не расстояние, а сам факт существования пути. К решению этой задачи можно применить алгоритм Флойда, но еще до него эта задача была решена Уоршеллом (Warshall). Фактически решением такой задачи будет

формирование матрицы транзитивного замыкания, т.е. такой матрицы, в которой единица будет стоять на тех позициях, в которых существует путь из одной вершины в другую (без учета количества вершин на этом пути).

Процедура для реализации аналогична процедуре Флойда, но massiv – другой тип: array [1..N, 1.. N] of boolean, что приводит и к изменению в операциях:

```
procedure Warshall (var a: massiv; c: massiv);
  Var I, j, k : integer;
  Begin for I := 1 to N do
            for J := 1 to N do
                  A[I,J] := C[I,J];
    for I := 1 to N do A [I,I] := 0;
    for K := 1 to N do
          for I := 1 to N do
                for j := 1 to N do
                    if A [I,J] = false then
                          A [I,J] := A[I,K] and A[K,J] ;
   End;
```

6. Краткие теоретические сведения. Поиск

6.1. Методы поиска данных

Одно из наиболее часто встречающихся в программировании действий – поиск. Существует несколько основных алгоритмов поиска и целый ряд алгоритмов позволяющих сделать этот процесс наиболее эффективным.

При дальнейшем рассмотрении исходим из допущения, что группа данных, в которой необходимо отыскать заданный элемент, фиксирована. Процесс поиска заключается в выборе элемента из последовательности элементов в соответствии с ключом. Если ключ входит в состав данных, среди которых производится поиск, то он называется внутренним, в противном случае – внешним. Ключ внешний - относительная позиция записи внутри файла или таблица ключей. Ключ может быть простым или сложным (составным).

К основным методам поиска относятся:

- Линейный поиск (последовательный),
- Бинарный поиск (двоичный, делением пополам),
- Индексно-последовательный поиск (в настоящем пособии не рассматривается),
- Поиск строки (поиск подстроки в строке),
- Поиск по дереву.

Поиск может быть внутренний и внешний. Критерием является размещение данных, среди которых производится поиск: если они размещены в оперативной памяти, то поиск внутренний, а если на внешнем носителе – внешний.

Алгоритм поиска – это алгоритм, который воспринимает некоторый аргумент A и пытается локализовать некоторую запись, ключ которой равен A.

Успешный поиск называется извлечением.

Рассмотрим различные поисковые алгоритмы подробнее.

6.2. Линейный поиск

Если нет никакой дополнительной информации о разыскиваемых данных, то очевидный подход – простой последовательный просмотр массива с увеличением шаг за

шагом той его части, где желаемого элемента не обнаружено. Такой метод называется линейным поиском.

Линейный поиск применяется к таблице, которая организована или как массив, или как связанный список.

Алгоритм линейного поиска – это простой последовательный просмотр массива с увеличением шаг за шагом той его части, в которой не оказалось желаемого элемента.

Условия окончания поиска таковы:

- элемент найден,
- весь массив просмотрен и совпадений не обнаружено.

Это дает нам алгоритм:

```
For i:=1 to N do
        if A[i]=key then begin
                Write (A[i]); Break;
                                end;
```

Если искомый элемент найден, то он будет напечатан (или может быть выполнено другое необходимое действие). Если искомый элемент не уникален, то прерывать работу цикла нельзя (т.е. использовать Break), а в некоторых случаях приходится использовать специальный признак (PR), который говорил бы о том, что поиск был успешен:

```
PR:=false;
For i:=1 to N do
   if A[i]=key then begin
      Write (A[i]); PR:=true;
                end;
```

Таким образом, можно говорить о том, что существуют два типа линейных поисков:

- с уникальным поисковым признаком (поиск прерывается после обнаружения первого же элемента);
- с неуникальным (поиск в любом случае продолжается до конца таблицы); найденные элементы фиксируются, подсчитывается их количество, выводятся на печать, сохраняются в другой структуре и т.п., или успешность поиска отражается в специальном поисковом признаке.

6.3. Бинарный поиск

Способов значительного убыстрения поиска не существует, если нет еще какой-либо информации о данных, среди которых идет поиск. Поиск становится значительно более эффективным, если он ведется среди упорядоченных данных. Т.е поиску предшествует сортировка. Т.к. процесс сортировки также занимает время, его нужно проводить однократно или по мере формирования данных, а не использовать каждый раз непосредственно перед поиском.

Основная идея – выбрать случайно некоторый элемент A[m] и сравнить его с аргументом поиска key. Если он равен key, то поиск закончится. Если он меньше, то из рассмотрения можно исключить все элементы с индексами, меньше m, а если больше, то с индексами, большими m. На этом рассуждении основан простой бинарный поиск:

```
L := 0; R := N-1;   PR := FALSE;
While (L <= R) and not PR do begin
M := (L +R) div 2;
if Key = A (M)          then begin writeln (A(M)); PR:=True; end
                                else if Key < A (M) then R:=M-1 else L:=M+1;
                                    end;
        if not PR then Writeln ('Поиск неуспешен');
```

В этом алгоритме:
L – левая граница массива,
R – правая граница массива.

Условие прекращения поиска L > R (т.е. элемент в просмотренном до конца массиве не обнаружен) или PR=true (т.е. элемент уже обнаружен).

В зависимости от значения ключа поиск ведется справа или слева от проверенного элемента.

Эффективность алгоритма можно несколько улучшить, если не заканчивать поиск немедленно при обнаружении совпадения. Поиск продолжают до тех пор, пока обе секции не "накроют" массив целиком. На этом рассуждении основан быстрый бинарный поиск:

```
L := 0; R := N-1;   PR := FALSE;
While (L <= R) and not PR do begin
M := (L +R) div 2;
if Key = A (M)          then begin writeln (A(M)); PR:=True; end
                                else if Key < A (M) then R:=M-1 else L:=M+1;
                                    end;
        if not PR then Writeln ('Поиск неуспешен');
```

Условие окончания – L>=R. При всех обстоятельствах разность R-L на каждом шаге убывает. По окончании цикла необходима дополнительная проверка на равенство A[R] = key, т.к. L=R не свидетельствует о совпадении.

6.4. Поиск строки

Поиск строки используется в системах обработки текстов. Существует несколько алгоритмов поиска. Такие алгоритмы также называют поиск подстроки в строке, т.к. разбор текста происходит построчно. А подстрока – это та самая строка, которую нужно найти в тексте, ее еще называют образом строки, или просто образом. В данном пособии остановимся на термине «подстрока». Самым простым методом является прямой поиск, который сводится к последовательным сравнениям отдельных символов. Поиск продолжается до тех пор, пока не обнаружится вхождение или пока не будет пройдена вся строка S. При этом просмотр можно закончить при i = N-M, так как при следующих значениях i длина любого фрагмента строки меньше M (где N - длина строки S, M - длина подстроки).

6.4.1. Прямой поиск подстроки в строке

Наиболее простым способом является прямой поиск строки, когда сравнение начинается с первого символа первой строки текста и первого символа подстроки. При этом каждый раз при несовпадении происходит сдвиг строки на один символ, а подстроку просматриваем сначала.

```
I:=-1;
Repeat I:=I+1; J:=0;
        While (J < M ) and (S [I+J] = P[J]) do J:=J+1;
Until (J=M) or (I = N-M);
```

6.4.2. Алгоритм Кнута, Мориса и Пратта

Каждый раз, выполняя сравнение с самого начала, мы не используем полученную на предыдущем шаге информацию. После частичного совпадения начальной строки с подстрокой мы фактически знаем пройденную часть строки и можем вычислить число позиций, на которые можем произвести сдвиг. Причем величина сдвига определяется не только количеством проверенных на совпадение символов, но и зависит от самой подстроки. Такой алгоритм был предложен Кнутом, Морисом и Праттом в 1970 году. Будем его называть алгоритмом КМП. Величина сдвига определяется образом подстроки и от строки не зависит.

Приведем фрагмент программы:

```
J:=1; K:=0; D[1]:=0;
{формирование массива D, определяющего сдвиг}
While J<M do begin
        While  (K>0) and (P[J] <> P[K]) do K:=D[K];
        J:=J+1; K:=K+1;
        If P[J] = P[K]  then D[J] :=D[K]
                                else D[J]:=K;
                end;
I:=0; J:=0;

{непосредственно поиск}
While (I<N) and (J<M) do begin
        While  (J>0) and (P[J] <> S[I] do J:=D[J];
        I:=I+1; J:=J+1;          end;
        If J = M+1 then Writeln ('Поиск удачен');
```

В первой части фрагмента вычисляются значения массива D, определяющие сдвиг. Для каждого j-го символа подстроки величина D[j] определяет номер символа подстроки, с которого следует продолжить сравнение при несовпадении символа i с номером j подстроки. При вычислении D[j] мы должны найти самую длинную последовательность символов подстроки, непосредственно предшествующих позиции j, которая совпадает полностью с началом образа.

6.4.2. Алгоритм Боуера и Мура

В 1975 году Боуэр и Мур предложили алгоритм, который дает выигрыш почти всегда. (Алгоритм БМ). Основная идея - сравнение с конца подстроки, но от начала строки S. Для подстроки P формируется таблица, размер которой определяется по числу всех возможных символов (например, 256, но можно и ограничить, если известно, из чего точно строится алфавит в предлагаемом тексте). Элементами таблицы являются расстояния от последнего символа искомой подстроки до ее каждого символа. Если встречаются одинаковые, то в таблицу заносится расстояние до ближайшего из них. Если символ не входит в подстроку, то в таблицу заносится M - длина подстроки. Когда очередной символ строки не совпадает с

очередным символом подстроки, то по таблице определяется соответствующее расстояние, P сдвигается на соответствующее число позиций.

Рассмотрим функцию поиска по алгоритму БМ:

```
Begin
{--Заполняем матрицу расстояний--}
for i:=1 to 33 do D[i]:=m;
dl:=1;
for i:=m-1 downto 0 do begin
      j:=Number(p[i]);
      if D[j]=m then D[j]:=dl;
      dl:=dl+1;
                          end;

{--Непосредственно поиск--}
i:=m; j:=m;
while (j>0) and (i<=N) do begin
            for h:=1 to m do pr[h]:=false;
            j:=m; k:=i;
            while (j>0) and (S[k]=P[j]) do begin
                                    pr[j]:=true;
                                      k:=k-1;
                                      j:=j-1;
                                              end;
             i:=i+d[number(s[i])];
                                   end;
Poisk:=true;
for i:=1 to m do
          if pr[i]=false then begin poisk:=false; break; end;
End;
```

Один из наиболее хитрых моментов в данном алгоритме – формирование индекса в массиве D. В данном случае предполагалось, что алфавит ограничен 33 элементами (D : array[1..33] of byte;), но необходимо помнить, что здесь всегда количество элементов соответствует полному алфавиту. Если символ алфавита в подстроке не используется, то в массив D заносится длина подстроки (D[i]:=m). Если же символ встречается в подстроке, то нужно вычислить расстояние от него до конца строки, но самое главное, занести это расстояние на место того элемента в массиве D, которое соответствует порядковому номеру символа.

Сделать это можно различными способами. Например, объявляем массив, содержащий полный алфавит:

Simvol:Array[1..33] of char='абвгдежзийклмнопрстуфхцчъыьэюя ';

А затем выясняем номер символа путем простого перебора и сравнения.

```
Function number (c:char) : byte;
Var i1: byte;
Begin
    for i:=1 to 33 do  if c=Simvol [i] then number:=i;
End;
```

Можно сделать проще. Использовать таблицу ASCII-кодов для получения кода символа. В случае 256 символов алфавита можно брать код «в чистом виде», а если алфавит ограничен, то использовать коэффициент для корректировки кода и превращения его в порядковый номер в массиве (чаще всего с использованием операции вычитания).

6.5. Поиск по дереву

Бинарные деревья часто используются для представления множества данных, среди которых идет поиск элементов по уникальному ключу.

Для обеспечения поиска по дереву предварительно строится упорядоченное дерево (двоичное дерево поиска) по определенным правилам:

- у каждого узла не более двух ветвей;
- любое значение, меньше значения узла, становится левым сыном (или сыном левого сына);
- любое значение, больше или равное значению узла, становится правым сыном (или сыном правого сына).

Последовательное сравнение вставляемого значения с потенциальным отцом продолжается до тех пор, пока не будет найдено место для вставки, и повторяется для каждого вставляемого значения до тех пор, пока не будет построено все дерево целиком.

В дереве поиска легко найти элемент с нужным ключом: начав с корня, нужно двигаться в левое или правое поддерево на основании сравнения заданного ключа с ключом текущего узла. Если дерево идеально сбалансировано, поиск среди его N элементов выполняется максимум за $\log_2 N$ сравнений. Подобные деревья широко используются и для сортировки больших массивов данных, так как симметричный обход дерева дает отсортированную в порядке возрастания последовательность ключей.

Над деревом поиска могут быть выполнены все операции, определенные для бинарного дерева:

- поиск элемента с заданным ключом Key в дереве и возвращение адреса элемента, если он найден, в противном случае возвращается значение nil;
- поиск по дереву с включением — поиск элемента с ключом Key в дереве, подключение узла с этим ключом, если поиск был неудачным (элемент с ключом Key не найден), и возвращение адреса элемента с заданным ключом;
- поиск по дереву с исключением — поиск элемента с заданным ключом в дереве и исключение (удаление) этого элемента, если он найден.

7. Варианты заданий на выполнение курсовой работы

7.1. Стандартные задания

Вариант 1.

1. Написать программу, которая создает бинарное дерево, состоящее из целых чисел, печатает все числа бинарного дерева, удаляет из бинарного дерева «отцов, имеющих одного сына». Определить количество таких «отцов», их сумму и произведение. Найти среди оставшихся узлов заданное число и определить его уровень.

2. Даны текстовый файл, содержащий ряд строк с названиями шлягеров, и текстовый файл, содержащий слова для поиска. Определить, встречается ли в названии шлягеров заданные слова и сколько раз. Для поиска использовать метод Кнута, Мориса и Пратта.

3. Получить произведения суммы удаленных «отцов» и количества повторений заданных слов (отдельно по каждому слову).

Вариант 2.

1. Написать программу, которая создает бинарное дерево, состоящее из целых чисел, печатает все числа бинарного дерева, удаляет из бинарного дерева все повторяющиеся числа. Определить количество удаленных чисел и есть ли среди удаленных заданное число.

2. Дана рекламная информация о турфирмах, в составе которой: название фирмы и стран, по которым фирма работает. Дан ряд стран. Определить количество фирм, работающих по заданной стране. Найти и напечатать название страны, по которой работает большее количество компаний. Для поиска использовать метод Кнута, Мориса и Пратта.

3. Определить, есть ли количество компаний, определенных в п.2 задания среди удаленных в п.1.

Вариант 3.

1. Написать программу, которая создает бинарное дерево, каждый элемент которого символ. Напечатать все элементы дерева. Выбрать и напечатать элементы, которые не являются листами дерева.

2. Дан текстовый файл и массив строк длиной не более 8 символов. Найти в текстовом файле строки, содержащие строки массива и удалить их из файла. Определить количество удаленных строк. Использовать метод Кнута, Мориса и Пратта.

3. Добавить в файл, сформированный в результате выполнения п.2, все элементы дерева из п.1, пройденного в симметричном порядке.

Вариант 4.

1. Написать программу, которая создает бинарное дерево, каждый элемент которого - запись, содержащая следующие поля: название журнала, периодичность и цена. Напечатать все записи. Найти и напечатать сведения о самом дорогом и самом дешевом журнале (алгоритм полного перебора недопустим). Найти заданный журнал и определить, является ли он листом, а если – нет, то определить, сколько у него сыновей.

2. Даны 2 массива строк длиной не более 10 символов, содержащие одинаковое количество элементов. Найти в названиях журналов элементы первого массива, заменив их элементами второго массива. Использовать прямой поиск строки. Если элементы первого и второго массивов разной длины, то замену не производить, а просто исключать их из файла.

3. Определить количество произведенных замен по заданию п.2. Вывести новое дерево.

Вариант 5.

1. Написать программу, которая создает бинарное дерево, состоящее из целых чисел, печатает все числа бинарного дерева, находит в бинарном дереве максимальное число среди отрицательных и минимальное - среди положительных. Напечатать номер уровня, на котором находится заданное число.

2. Дан текстовый файл. Подсчитать количество повторений заданных слов в файле. Напечатать эти слова. Для поиска использовать метод Кнута, Мориса и Пратта.

3. Посчитать общее количество повторений заданных слов и добавить его в дерево п.1.

Вариант 6.

1. Написать программу, которая создает бинарное дерево, каждый элемент которого символ. Напечатать все элементы дерева. Выбрать и напечатать элементы, которые являются листами дерева. Определить, есть ли среди листьев заданный символ.

2. Создать упорядоченный однонаправленный список строк длиной до 50 символов. Найти в нем строки, в составе которых есть заданная подстрока. Для поиска подстроки использовать метод Боуера и Мура.

3. Составить из листьев дерева п.1 строку и добавить ее в список п.2.

Вариант 7.

1. Написать программу, которая создает бинарное дерево, состоящее из целых чисел, печатает все числа бинарного дерева, формирует новое бинарное дерево из отрицательных чисел старого дерева. В новом дереве найти и напечатать только те цифры, которые начинаются заданной цифрой.

2. Дан текстовый файл. Даны массивы строковых данных A и B. Найти в файле каждое A[i] слово и заменить его на B[i]. Для поиска использовать метод Кнута, Мориса и Пратта.

3. Посчитать общее количество повторений заданных слов и добавить его в новое дерево (п.1 без отрицательных значений). Определить его уровень.

Вариант 8.

1. Написать программу, которая создает бинарное дерево, состоящее из целых чисел, печатает все числа бинарного дерева, печатает числа бинарного дерева, используемые больше одного раза и количество их повторений. Определить количество чисел, используемых в дереве только один раз.

2. Дан текстовый файл, содержащий информацию в составе: название и основные достопримечательности городах (все достопримечательности – одна строка). Определить названия и количество городов, в которых есть заданные достопримечательности (например, Кремль, краеведческий музей и т.п.). Для поиска использовать алгоритм Боуера и Мура.

3. Определить самый часто используемый символ (п.1) и определить, входит ли он в названия городов в п.2 и сколько раз.

Вариант 9.

1. Написать программу, которая создает бинарное дерево, каждый элемент которого символ. Напечатать все элементы дерева по убыванию. Определить, сколько раз встречаются в дереве заданные символы и сколько раз они являются листом.

2. Создать упорядоченный двунаправленный список вещественных чисел. Найти в нем заданные элементы, используя метод бинарного поиска.

3. Добавить в список п.2 среднее значение повторений заданных символов п.1 и определить, меньше или больше оно среднего значения списка.

Вариант 10.

1. Написать программу, которая создает бинарное дерево, каждый элемент которого – фамилия, длиной до 20 символов. Напечатать все элементы дерева. Выбрать из дерева и напечатать фамилии, которые начинаются и заканчиваются на одну и ту же букву, если эти строки не являются листами.

2. Дан список участников конференции в составе: фамилия, имя, отчество и название доклада (одной строкой). Найти и напечатать сведения обо всех участниках конференции, фамилии которых есть в дереве (п.1). Использовать прямой поиск строки.

3. Определить количество участников конференции, фамилии которых есть в дереве (п.1).

Вариант 11.

1. Написать программу, которая создает бинарное дерево, каждый элемент которого - запись, содержащая следующие поля: фамилия и инициалы, год рождения. Напечатать все записи, упорядочив их по году рождения. Напечатать сведения о всех людях, которые старше человека с заданной фамилией.

2. Найти самого старого человека.

3. Найти в упорядоченном двумерном символьном массиве символ, с которого начинается фамилия самого старого человека, используя метод бинарного поиска.

Вариант 12.

1. Написать программу, которая создает бинарное дерево, каждый элемент которого – название страны (строка, длиной <= 10 символов). Напечатать все элементы дерева. Выбрать из дерева и напечатать строки, которые начинаются и заканчиваются на заданные буквы (первая и последняя буквы могут не совпадать) и определить количество их сыновей.

2. Дан список участников конференции в составе: фамилия, имя, отчество, страна, компания (одной строкой). Найти и напечатать сведения обо всех участниках конференции из стран, которые есть в дереве (п.1). Использовать алгоритм Боуера и Мура.

3. Определить количество стран из п.1, представители которых принимают участие в конференции.

Вариант 13.

1. Написать программу, которая создает бинарное дерево, каждый элемент которого - строка. Напечатать все элементы дерева. Выбрать из дерева и напечатать строки, длина которых больше заданных.

2. Напечатать все листья дерева, в состав строк которых входят заданные подстроки, определить количество вхождений каждой подстроки. Использовать алгоритм прямого поиска строки.

3. Определить общее количество вхождений всех подстрок.

Вариант 14.

1. Написать программу, которая создает бинарное дерево, каждый элемент которого - запись, содержащая следующие поля: название компании, годовой доход. Напечатать все записи в алфавитном порядке по названиям компании. Определить, убыточна ли заданная компания. Удалить из дерева компании, имеющие отрицательный годовой доход (убытки).

2. Дан текстовый файл, содержащий подстроки (слова <=5 символов). Определить, сколько раз в названиях компаний с положительным годовым доходом встречаются заданные подстроки, используя алгоритм Боуера и Мура.

3. Определить самую часто встречающуюся строку.

Вариант 15.

1. Написать программу, которая создает бинарное дерево, каждый элемент которого - запись, содержащая следующие поля: название передачи (ТВ, радио…), время выхода в эфир. Напечатать все записи в алфавитном порядке по названию передач. Напечатать название передачи, выходящей в эфир в наиболее ранее время.

2. Дан массив строк с информацией о передачах. Определить, сколько раз в каждой строке встречаются названия передач, информация о которых содержится в дереве, используя метод прямого поиска строки. Определить самое часто встречающееся название передачи.

3. Определить время выхода в эфир передач, упомянутых в п.2.

Вариант 16.

1. Написать программу, которая создает бинарное дерево, каждый элемент которого - запись, содержащая информацию о самолетах и вертолетах (ВС) в составе: тип ВС (Боинг-737, Ил-96…), год ввода в эксплуатацию. Напечатать все записи в алфавитном порядке по типу ВС. Напечатать типы ВС, введенные в эксплуатацию ранее заданного года

2. Дан массив строк. Определить, сколько раз каждая строка встречается в названиях типов ВС, используя метод Кнута, Мориса и Пратта. Определить наиболее редко встречающуюся строку.

3.Определить общее количество найденных в дереве строк.

Вариант 17.

1. Написать программу, которая создает бинарное дерево, каждый элемент которого - строка, содержащая информацию о вузах в составе: название полное, аббревиатура, год открытия. Напечатать все элементы дерева. Удалить из дерева вузы, открытые позднее заданного года.

2. Дан массив строк. Определить, сколько раз каждая строка встречается в названиях и аббревиатурах вузов в дереве, используя алгоритм Боуера и Мура.

3. Определить общее количество найденных строк.

Вариант 18.

1. Написать программу, которая создает бинарное дерево, каждый элемент которого - запись, содержащая следующие поля: название компании, годовой доход. Напечатать все записи. Напечатать в алфавитном порядке и в порядке уменьшения дохода названия всех компаний, имеющих положительный годовой доход. Определить, выше среднего или нет доход заданной компании.

2. Дан текстовый файл, содержащий информацию о компаниях. Определить, используя метод Кнута, Мориса и Пратта, если в этом файле упоминания о компаниях из дерева п.1. Если есть, то вывести название и количество упоминаний.

3. Определить общее количество упоминаний о компаниях.

Вариант 19.

1. Написать программу, которая создает бинарное дерево, каждый элемент которого - запись, содержащая следующие поля: название компании, годовой доход. Напечатать все записи. Напечатать название компании, имеющей наибольший годовой доход. Сформировать однонаправленный список, содержащий названия компаний, имеющих положительный годовой доход.

2. Дан текстовый файл, содержащий ряд слов. Определить, сколько раз в названиях компаний из списка п.1 встречаются заданные слова, используя метод прямого поиска строки.

3. Добавить в дерево п.1 новую компанию, а если она имеет положительный доход, то и в список.

Вариант 20.

1. Дана последовательность целых чисел. Разработать программу построения упорядоченного бинарного дерева и печати всех имеющихся дубликатов чисел, общее количество дубликатов и количество повторений.

2. Дан массив строк. Каждая строка состоит из слов, разделенных пробелами. Подсчитать количество слов в каждой строке, начинающихся на заданное сочетание символов. Напечатать эти слова. Для поиска использовать метод прямого поиска строки.

3. Определить общее количество найденных слов и проверить, есть ли такое число в дереве (п.1).

Вариант 21.

1. Написать программу, которая создает бинарное дерево, состоящее из вещественных чисел, вводимых с клавиатуры, печатает все числа бинарного дерева, добавляет в бинарное дерево числа из текстового файла, меньшие минимального значения первоначального дерева. Определить количество добавленных узлов и количество чисел, не подлежащих добавлению в дерево.

2. Дан текстовый файл. Даны массивы строковых данных A и B. Найти в файле каждое A[i] слово и заменить его на B[i]. Для поиска использовать метод прямого поиска строки. Вывести номера строк, в которых произошли замены и количество замен по каждому слову.

3. Добавить в дерево общее количество замен и определить уровень, на котором будет находиться добавленный элемент.

Вариант 22.

1. Написать программу, которая создает бинарное дерево, состоящее из вещественных чисел, печатает все числа бинарного дерева, находит сумму отрицательных и произведение положительных элементов дерева. Напечатать все числа, меньшие заданного, найти среди них максимальное.

2. Дан массив строк максимальной длины. Дан файл, содержащий заданные подстроки (слова). Определить количество повторений заданных слов в каждой строке и номера строк, в которых они находятся. Для поиска использовать метод Кнута, Мориса и Пратта.

3. Определить общее количество найденных слов (п.2) и сравнить его с максимальным (п.1)

Вариант 23.

1. Написать программу, которая создает бинарное дерево, состоящее из целых чисел, печатает все числа бинарного дерева. Определить количество уровней дерева, на которых находятся листья. Определить, является ли заданное число листом (если оно вообще есть в дереве). Сформировать новое дерево, состоящее только из листьев старого дерева.

2. Дан текстовый файл и 2 одномерных массива строковых данных, содержащих слова. Найти в файле слова, содержащиеся в первом файле и заменить словами, во втором. (I-е слово второго массива заменить I-м словом второго массива). Для поиска использовать метод Боуера и Мура.

3. Определить количество замен. Сравнить получившееся значение со средним значением нового дерева (п.1).

Вариант 24.

1. Написать программу, которая создает бинарное дерево, состоящее из целых чисел, печатает все числа бинарного дерева, удаляет из бинарного дерева все листья. Определить количество узлов, не являющихся листьями. Определить, есть ли среди этих узлов заданный и если есть, то дать сообщение, меньше, больше или равен он среднему.

2. Дан список компаний с производимым им товаром. Каждая компания – одна строка. Дан массив, содержащий список товаров. Определить по каждому товару количество компаний, его выпускающих. Для поиска использовать метод Кнута, Мориса и Пратта.

3. Определить среднее количество компаний выпускающих заданные товары и определить, больше или меньше корня дерева (п.1) получившееся значение.

7.2. Задания повышенной сложности

Вариант 31.

1. Написать программу, которая создает сильно ветвящееся бинарное дерево заданной династии (Ририковичи, Романовы, Виндзор и т.п.). Состав узла определяется условиями задачи, но может быть и расширен. Определить глубину дерева и количество членов династии, не имевших детей.

2. Организовать поиск представителя династии по дереву и выдачу дополнительной информации по искомому (искомым) представителю(лям) династии. Для хранения

дополнительной информации можно использовать список, файл или массив, упорядоченный по номеру узла. Для поиска в строке использовать прямой поиск.

3. Сформировать список, содержащий только правителей династии с годами правления, упорядоченный по году прихода к власти, и сформировать из него список, упорядоченный по продолжительности нахождения у власти.

Вариант 32.

1. Дан текстовый файл, содержащий программу на алгоритмическом языке Паскаль. Построить код Хаффмена для служебных слов этой программы, используя дерево. Для поиска служебных слов использовать алгоритм Боуэра-Мура.

2. Создать упорядоченный список служебных слов с количеством их повторений и кодом.

3.Определить, на сколько изменится кодировка программы, если использовать построенный код (в байтах).

Вариант 33.

1. Дано ряд арифметических выражений с операциями двух приоритетов. Сформировать лес из деревьев, содержащих эти выражения. Выполнить вычисление выражений и упорядочить лес по возрастанию результата. Сформировать список, содержащий по каждому дереву результат (с мантиссой до 12 знаков), количество операндов, количество операций и глубину дерева.

2. По каждому результату осуществить поиск методом бинарного поиска, входит ли в его состав заданная цифра и сколько раз.

3. Определить общее вхождение заданной цифры и добавить его в список.

Вариант 34.

1. Дано множество задач по проекту T_1, T_2, …, T_n, для выполнения которых необходимо время t_1, t_2, …, t_n, и множество отношений вида «задача T_i должна закончиться до начала задачи T_j». Найдите минимальное время для выполнения всех задач.

2. По каждой задаче есть список исполнителей (одной строкой, исполнители разделяются пробелами). Сформировать дерево из исполнителей и количества задач, в которых они принимают участие.

3. Определить исполнителя, который принимает участие в наибольшем количестве задач, а также максимальную продолжительность, на которую он может уйти в отпуск в период проекта.

Вариант 35.

1. Дано граф, содержащий расстояния между городами. Сформировать программу, которая определяет кратчайшее расстояние между любыми двумя городами. Построить остовное дерево, в котором суммарное расстояние будет наименьшим(минимальное остовное дерево).

2. По каждому городу дана информация (сплошным текстом), содержащая названия и адреса кинотеатров, магазинов и т.п. По запросу пользователя выдавать информацию о заданном объекте или об объектах заданного города. По запросу пользователя определять кратчайший путь до этого города из заданной исходной точки (воспользоваться программой п.1).

3. Сформировать дерево из названий городов, их площадей и количества объектов в них. По запросу пользователя выдавать список городов, упорядоченный по названию, площади или количеству объектов.

Вариант 36.

1. В некоторой файловой системе справочник файлов организован в виде упорядоченного двоичного дерева. Каждой вершине соответствует некоторый файл, информация по которому – имя, расширение, дата создания и дата последнего обращения к нему. Разработать программу, которая удаляет все файлы, последнее обращение к которым было до некоторой определенной даты.

2. По запросу пользователя искать файлы по дате создания, имени, расширению и шаблону. Для поиска по шаблону использовать алгоритм Боуера и Мура.

3. Определить самый старый файл. Вывести файлы в виде упорядоченного списка. Построить из этих файлов сбалансированное дерево.

Вариант 37.

Дан граф. Напишите программу алгоритма Дейкстры, используя связанные списки смежности и очередь с приоритетами, реализованную посредством частично упорядоченного дерева. Напишите программу транзитивной редукции графа.

8. Защита курсовой работы

8.1. Демонстрация работоспособности программы

Отлаженная программа демонстрируется преподавателю на персональном компьютере (в компьютерном классе). При этом необходимо продемонстрировать все возможности программы и доказать ее работоспособность при различных исходных данных. Для этого необходимо эти исходные данные подготовить: сформировать несколько файлов с исходными данными, предварительно подобрав их. При этом преподаватель исполняет роль пользователя программы. То есть интерфейс должен быть таким, чтобы у пользователя не возникало вопросов, что дальше делать, что наживать и какой файл выбирать. Меню (при его наличии) должно быть подробным. Как один из вариантов организации можно обеспечить вывод подсказки по каждому пункту меню. Кроме того, выход из программы должен осуществляться только по желанию пользователя. Все возможности программы должно быть возможно просмотреть, запустив ее на выполнение только один раз.

В процессе демонстрации программы необходимо ответить на вопросы преподавателя и выполнить его небольшие задания. После чего студент получает разрешение на оформление работы и ее защиту.

В случае критических замечаний студент устраняет их позже и демонстрирует работу повторно.

8.2. Подготовка к защите курсовой работы

Для защиты курсовой работы студент:

- оформляет пояснительную записку в соответствии с требованиями, изложенными в настоящем пособии (предварительно перед распечаткой может быть показана преподавателю в электронном виде во время консультации);
- формирует электронный носитель, содержащий файлы курсовой работы (CD-диск или флеш-диск);
- повторяет теоретический материал по теме курсовой работы (по материалам лекций и рекомендуемой литературе);
- отвечает на контрольные вопросы.

8.3. Состав информации на электронном носителе

После проверки работоспособности программы и распечатки пояснительной записки студент формирует электронный носитель, содержащий следующие файлы курсовой работы:

- все программы, в том числе разработанные модули;
- исходные данные для всех программ (все файлы, если их для демонстрации было подготовлено несколько);
- пояснительную записку;
- файл Readme с описанием содержания электронного носителя (назначение файлов, связь между ними).

При необходимости (для удобочитаемости) файлы могут быть структурированы по каталогам.

8.4. Контрольные вопросы

1. Нелинейная динамическая структура: назначение, объявление, типовые операции.
2. Что такое дерево? Бинарное дерево?
3. Что такое узел дерева, лист, уровень дерева, уровень листа?
4. Что такое лес? Поясните способы работы с лесом.
5. Поясните способы обхода дерева.
6. Какой обход дерева используется для сортировки данных?
7. Что такое полное дерево, почти полное дерево?
8. Что такое сбалансированное дерево, идеально сбалансированное дерево?
9. Какие задачи решаются с использованием деревьев?
10. Поясните перевод выражений из инфиксной формы в префиксную/ постфиксную с помощью деревьев.
11. Поясните построение частотного словаря с использованием дерева.
12. Поясните алгоритм поиска по дереву.
13. Поясните алгоритм прямого поиска строки.
14. Поясните алгоритм Боуэра-Мура для поиска строки.
15. Поясните алгоритм Кнута, Мориса и Пратта для поиска строки.
16. Как организовать исходные данные в программе для определения графа.
17. Поясните реализацию графов в виде списков смежности.
18. Что такое остовное дерево?
19. Поясните жадный алгоритм.
20. Поясните алгоритм Дейкстры.
21. Поясните алгоритм Флойда.
22. Что такое обход графа в глубину? Поясните на примере.
23. Что такое обход графа в ширину? Поясните на примере.
24. В чем разница при программировании ориентированных и неориентированных графов?
25. Поясните особенности обхода ориентированного и неориентированного графа в глубину и в ширину.
26. Что такое корень ациклического графа?
27. Что такое транзитивная редукция?

8.5. Критерии оценивания курсовой работы

Курсовая работа оценивается по четырехбалльной системе: «отлично», «хорошо», «удовлетворительно», «неудовлетворительно».

«Отлично» за выполнение курсовой работы ставится при выполнении следующих условий:

- программа выполнена полностью в соответствии с заданием;
- продемонстрирована ее работоспособность;
- пояснительная записка оформлена в соответствии с требованиями,
- на все вопросы даны корректные ответы.

«Хорошо» ставится за небольшие неточности, допущенные при выполнении и оформлении работы, а также при ответе на вопросы.

«Удовлетворительно» ставится в случае, если

- допущены грубые ошибки при выполнении программы или она сделана не полностью;
- пояснительная записка оформлена с грубыми нарушениями;
- ответы на вопросы - некорректные.

«Неудовлетворительно» выставляется в случаях, когда программа не работает, пояснительная записка не оформлена.

Основаниями для снижения оценки могут быть:

- небрежное выполнение, нелогичное использование или неиспользование процедур и функций;
- отсутствие комментариев;
- низкое качество графического материала (схемы небрежно начерчены от руки);
- нелогичный пользовательский интерфейс (недружественный, интуитивно непонятный и т.п.)
- и т.п.

Отчет не может быть принят и подлежит доработке в случае:

- отсутствия необходимых разделов,
- отсутствия необходимого иллюстративного материала,
- некорректной обработки результатов работы программы,
- и т.п.

9. Список рекомендуемой литературы

1.Кремень Ю., Расолько Г.	Теория и практика программирования на языке Pascal / Москва / Литрес/ 2015 г., 449 с.	http://www.iqlib.ru/
2.Зеленяк О.П.	Зеленяк О.П. / Современный задачник по Турбо Паскалю / Москва / ДМК Пресс / 2012 /	http://www.iqlib.ru/
3.Егорова А.А.	Пособие по дисциплине «Структуры данных и методы обработки информации» для студентов IV курса специальности 073000 – М.: МГТУ ГА, 2011г. – 48с.	
4.Егорова А.А.	Пособие по проведению практических занятий по дисциплине «Программирование для ЭВМ» - М.: МГТУ ГА, 2013 - 36	
5.Вирт Н	Вирт Н. / Алгоритмы и структуры данных. Новая версия для Оберона / Москва / ДМК Пресс / 2010	http://www.iqlib.ru/
6.Ахо А., Хопкрофт Д., Ульман Д.	Структуры данных и алгоритмы.: Пер. с англ. : Уч. Пос. – М.: Издательский дом «Вильямс» , 2016, - 384 с.	
7. Климова Л.М.	Pascal 7.0. Практическое программирование. Решение типовых задач. КУДИЦ-ОБРАЗ., 4-е издание	