Министерство науки и высшего образования Российской Федерации

Томский государственный университет
систем управления и радиоэлектроники

Н. В. Пермякова

ПРОГРАММИРОВАНИЕ

Методические указания к лабораторным работам,
практическим занятиям и организации самостоятельной работы
для студентов направления «Программная инженерия»
(уровень бакалавриата)

Томск
2018

Пермякова, Наталья Викторовна
Программирование: методические указания к лабораторным работам, практическим занятиям и организации самостоятельной работы для студентов направления «Программная инженерия» (уровень бакалавриата) Н.В. Пермякова. – Томск : Томск. гос. ун-т. систем упр. и радиоэлектроники, 2018. – 37 с.

Настоящее учебно-методическое пособие по выполнению лабораторных работ, подготовке к практическим занятиям и организации самостоятельной работы составлено с учетом требований федерального государственного образовательного стандарта высшего образования (ФГОС ВО).

Учебно-методическое пособие содержит краткое описание теоретического материала, необходимого для выполнения указанных видов занятий. Для каждой лабораторной работы определена цель работы, порядок ее выполнения, приведены примеры выполнения типовых заданий. Для практических занятий описан регламент проведения занятия. Для организации самостоятельной работы перечислены темы домашних заданий и контрольных работ, перечислены дополнительные литературные источники.

Оглавление

ВВЕДЕНИЕ

В методических указаниях к проведению лабораторных работ, практических занятий и организации самостоятельной работы по дисциплине «Программирование» собраны материалы для методической поддержки аудиторных занятий и самостоятельной работы студентов.

Целью проведения лабораторных работ, практических занятий и самостоятельной работы является формирование и развитие практических навыков реализации классических алгоритмов сортировки и поиска на языке Си. Для достижения заданной цели во время проведения занятий последовательно решаются следующие задачи:

– изучение существующих алгоритмов сортировки, среди которых: простые сортировки на месте, сортировка Шелла, сортировка Хоара, поразрядные сортировки, сортировка слиянием;

– изучение алгоритмов поиска (прямой, бинарный и интерполяционный поиски);

– изучение алгоритмов поиска подстроки в строке;

– изучение структур данных, алгоритмов их обработки и их программная реализация;

– изучение вопросов построения эффективных алгоритмов;

– решение прикладных задач программирования на примере решения задач дискретной математики и вычислительной математики.

По окончанию обучения дисциплины «Программирование» студент должен:

– **знать** классические алгоритмы поиска и сортировки;

– **уметь** самостоятельно разрабатывать алгоритмы поставленных перед ним задач,в том числе используя базовые алгоритмы;

– **владеть** навыками использования различных структур данных при решении задач.

Цикл лабораторных работ и практических занятий по дисциплине, выполняемых студентами во время первого семестра, направлен на закрепление навыков использования конструкций структурного программирования в части реализации базовых алгоритмов сортировки и поиска, навыков разработки эффективных программ, навыков использования различных структур данных в программной реализации.

Согласно действующему Федеральному Государственному стандарту высшего образования (ФГОС ВО) значительная доля курса отведена на самостоятельную работу студентов. При изучении дисциплины «Программирование» самостоятельная работа студентов включает несколько видов деятельности – проработка лекционного материала, подготовка к контрольным работам, выполнение домашних заданий, самостоятельное изучение тем, подготовка к экзамену. В методическом пособии предложены рекомендации по организации перечисленных видов деятельности, включающие порядок выполнения отдельных видов самостоятельной работы, краткие теоретические сведения, перечень литературных источников по темам дисциплины, вынесенным на самостоятельное изучение.

1 МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ПРОВЕДЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

1.1 Общие положения

Целью проведения лабораторных работ является развитие навыков структурного программирования в части разработки базовых алгоритмов сортировки и поиска.

Основной формой проведения лабораторных работ является разработка алгоритма решения индивидуальной задачи и его программная реализация на языке Си. Процесс программной реализации включает в себя написание программы, отладку программы и тестирование программы.

К основным способам контроля формирования компетенций при выполнении лабораторных работ относятся индивидуальная защита выполненной работы, организация опроса студентов по теоретическому материалу дисциплины, практическое применение которого осуществляется в ходе выполнения лабораторной работы.

Для получения максимальной оценки за лабораторную работу необходимо выполнить и защитить работу во время, отведенное для ее выполнения, согласно расписанию занятий. Допускается досрочное выполнение лабораторной работы по предварительной договоренности с преподавателем.

Выполнение всех лабораторных работ, предусмотренных рабочей программой дисциплины, является условием допуска к итоговому контролю изучения дисциплины – экзамену.

1.2 Лабораторная работа «Динамические списки»

Цель работы: формирование навыков работы с динамическими структурами данных. При выполнении задания может быть полезен материал пособия [1], стр. 175 – 190 и пособия [2], стр. 224 – 228.

Динамические структуры данных

Часто в серьезных программах необходимо использовать данные, размер и структура которых должны меняться в процессе работы. Динамические массивы здесь не эффективны, поскольку заранее нельзя сказать, сколько памяти надо выделить – это выясняется только в процессе работы. Например, нужно проанализировать текст и определить, какие слова и в каком количество в нем встречаются, причем эти слова нужно расставить по алфавиту.

В таких случаях применяют данные особой структуры, которые представляют собой отдельные элементы, связанные с помощью ссылок.

Каждый элемент (узел) состоит из двух областей памяти: поля данных и ссылок (рис. 1.9). Ссылки – это адреса других узлов этого же типа, с которыми данный элемент логически связан. В языке Си для организации ссылок используются переменные-указатели. При добавлении нового узла в такую структуру выделяется новый блок памяти и (с помощью ссылок) устанавливаются связи этого элемента с уже существующими. Для обозначения конечного элемента в цепи используются нулевые ссылки (NULL).

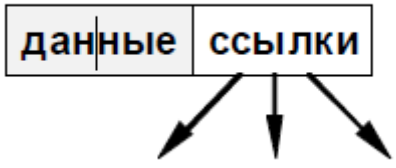

Рисунок 1.9 – Структура узла списка

Линейный список

В простейшем случае каждый узел содержит всего одну ссылку. Для определенности полагается, что решается задача частотного анализа текста – определения всех слов, встречающихся в тексте и их количества. В этом случае область данных элемента включает строку (длиной не более 40 символов) и целое число.

Каждый элемент содержит также ссылку на следующий за ним элемент. У последнего в списке элемента поле ссылки содержит NULL. Чтобы не потерять список, нужно где-то (в переменной) хранить адрес его первого узла – он называется «головой» списка (рис. 1.10).

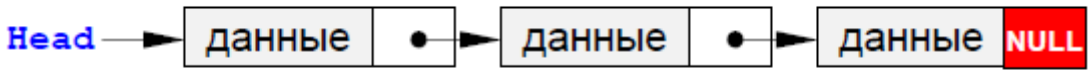

Рисунок 1.10 – Структура линейного списка

В программе следует объявить два новых типа данных – узел списка Node и указатель на него PNode. Узел представляет собой структуру, которая содержит три поля – строку, целое число и указатель на такой же узел. Правилами языка Си допускается объявление:

```
struct Node {
char word[40]; // область данных
int count;
Node *next; // ссылка на следующий узел
};
typedef Node *PNode; // тип данных: указатель на узел
```

В дальнейшем считается, что указатель ***Head*** указывает на начало списка, то есть, объявлен в виде

```
PNode Head = NULL;
```

В начале работы в списке нет ни одного элемента, поэтому в указатель Head записывается нулевой адрес NULL.

Создание элемента списка

Для того, чтобы добавить узел к списку, необходимо создать его, то есть выделить память под узел и запомнить адрес выделенного блока. Будем считать, что надо добавить к списку узел, соответствующий новому слову, которое записано в переменной NewWord. Составим функцию, которая создает новый узел в памяти и возвращает его адрес.

Обратите внимание, что при записи данных в узел используется обращение к полям структуры через указатель.

```
    PNode CreateNode ( char NewWord[] )
    {
    PNode NewNode = (PNode)malloc(sizeof( Node)); // указатель на новый
узел
    strcpy(NewNode->word, NewWord); // записать слово
    NewNode->count = 1; // счетчик слов = 1
    NewNode->next = NULL; // следующего узла нет
    return NewNode; // результат функции – адрес узла
    }
```

После этого узел надо добавить к списку (в начало, в конец или в середину).

Добавление узла

Добавление узла в начало списка

При добавлении нового узла `NewNode` в начало списка надо установить ссылку узла `NewNode` на голову существующего списка (рис. 1.11) и установить голову списка на новый узел (рис. 1.11).

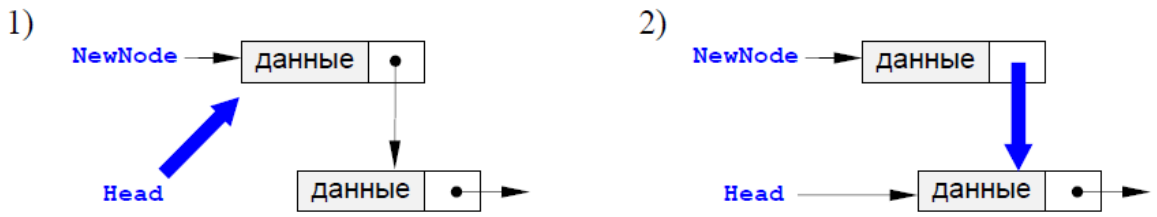

Рисунок 1.11 – Добавление узла в начало списка

По такой схеме работает процедура `AddFirst`. Предполагается, что адрес начала списка хранится в `Head`. Важно, что здесь и далее адрес начала списка передается по ссылке, так как при добавлении нового узла он изменяется внутри процедуры.

```
void AddFirst (PNode *Head, PNode NewNode)
{
NewNode->next = *Head;
*Head = NewNode;
}
```

Добавление узла после заданного

Дан адрес *NewNode* нового узла и адрес p одного из существующих узлов в списке. Требуется вставить в список новый узел после узла с адресом *p*. Эта операция выполняется в два этапа:

1) установить ссылку нового узла на узел, следующий за данным;
2) установить ссылку данного узла p на *NewNode* (рис. 1.12).

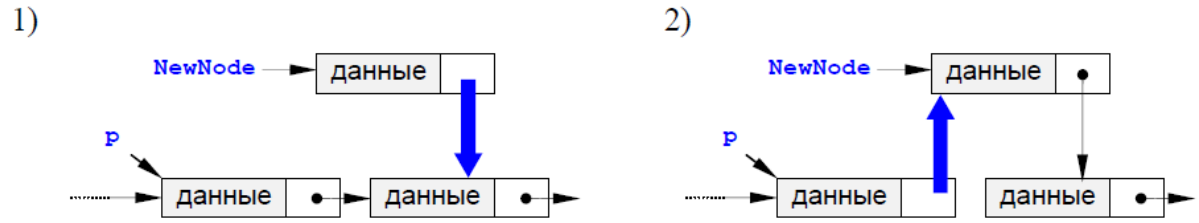

Рисунок 1.12 – Добавление узла после заданного

Последовательность операций менять нельзя, потому что, если сначала поменять ссылку у узла *p*, будет потерян адрес следующего узла.

```
void AddAfter (PNode p, PNode NewNode)
{
NewNode->next = p->next;
p->next = NewNode;
}
```

Добавление узла перед заданным

Эта схема добавления самая сложная. Проблема заключается в том, что в простейшем линейном списке (он называется односвязным, потому что связи направлены только в одну сторону) для того, чтобы получить адрес предыдущего узла, нужно пройти весь список сначала.

Задача сведется либо к вставке узла в начало списка (если заданный узел – первый), либо к вставке после заданного узла.

```
void AddBefore(PNode *Head, PNode p, PNode NewNode)
{
```

```
PNode q = *Head;
if (*Head == p) {
AddFirst(Head, NewNode); // вставка перед первым узлом
return;
}
while (q && q->next!=p) // ищем узел, за которым следует p
q = q->next;
if ( q ) // если нашли такой узел,
AddAfter(q, NewNode); // добавить новый после него
}
```

Существует еще один интересный прием: если надо вставить новый узел *NewNode* до заданного узла *p*, вставляют узел после этого узла, а потом выполняется обмен данными между узлами *NewNode* и *p*. Таким образом, по адресу *p* в самом деле будет расположен узел с новыми данными, а по адресу *NewNode* – с теми данными, которые были в узле *p* Этот прием не сработает, если адрес нового узла *NewNode* запоминается где-то в программе и потом используется, поскольку по этому адресу будут находиться другие данные.

Добавление узла в конец списка

Для решения задачи надо сначала найти последний узел, у которого ссылка равна NULL, а затем воспользоваться процедурой вставки после заданного узла. Отдельно надо обработать случай, когда список пуст.

```
void AddLast(PNode *Head, PNode NewNode)
{
PNode q = *Head;
if (*Head == NULL) { // если список пуст,
AddFirst(Head, NewNode); // вставляем первый элемент
return;
}
while (q->next) q = q->next; // ищем последний элемент
AddAfter(q, NewNode);
}
```

Проход по списку

Для того, чтобы пройти весь список и сделать что-либо с каждым его элементом, надо начать с головы и, используя указатель *next*, продвигаться к следующему узлу.

```
PNode p = Head; // начали с головы списка
while ( p != NULL ) { // пока не дошли до конца
// делаем что-нибудь с узлом p
p = p->next; // переходим к следующему узлу
}
```

Поиск узла в списке

Часто требуется найти в списке нужный элемент (его адрес или данные). Надо учесть, что требуемого элемента может и не быть, тогда просмотр заканчивается при достижении конца списка. Такой подход приводит к следующему алгоритму:

1) начать с головы списка;

2) пока текущий элемент существует (указатель – не NULL), проверить нужное условие и перейти к следующему элементу;

3) закончить, когда найден требуемый элемент или все элементы списка просмотрены.

Например, следующая функция ищет в списке элемент, соответствующий заданному слову (для которого поле *word* совпадает с заданной строкой *NewWord*), и возвращает его адрес или NULL, если такого узла нет.

```
PNode Find (PNode Head, char NewWord[])
```

```
{
PNode q = Head;
while (q && strcmp(q->word, NewWord))
q = q->next;
return q;
}
```

Удаление узла

Эта процедура также связана с поиском заданного узла по всему списку, так как нам надо поменять ссылку у предыдущего узла, а перейти к нему непосредственно невозможно. Если мы нашли узел, за которым идет удаляемый узел, надо просто переставить ссылку (рис. 1.13).

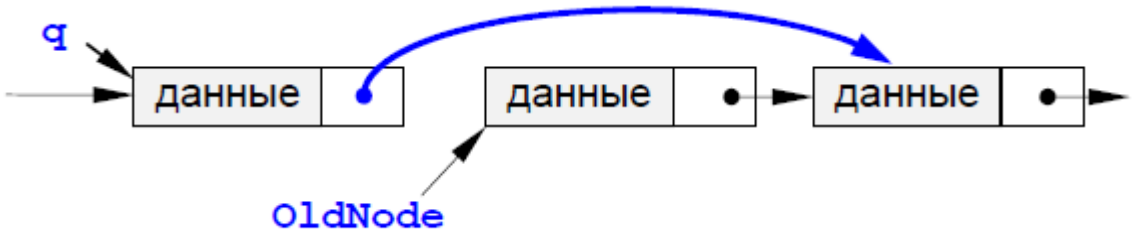

Рисунок 1.13 – Удаление узла

Отдельно обрабатывается случай, когда удаляется первый элемент списка. При удалении узла освобождается память, которую он занимал.

Отдельно рассматривается случай, когда удаляется первый элемент списка. В этом случае адрес удаляемого узла совпадает с адресом головы списка *Head* и надо просто записать в *Head* адрес следующего элемента.

```
void DeleteNode(PNode *Head, PNode OldNode)
{
PNode q = *Head;
if (*Head == OldNode)
*Head = OldNode->next; // удаляем первый элемент
else {
while (q && q->next != OldNode) // ищем элемент
q = q->next;
if ( q == NULL ) return; // если не нашли, выход
q->next = OldNode->next;
}
free( OldNode); // освобождаем память
}
```

Порядок выполнения работы

1. Получить индивидуальный вариант.
2. Написать программу, решающую поставленную задачу с использованием функций.
3. Отладить и протестировать программу.
5. Защитить работу.

Контрольные вопросы

1. В каких задачах рекомендуется использование динамических списков?
2. Какой маркер указывает на последний элемент списка?
3. Какой метод создания списка используется в вашей реализации?
4. Как называется структура данных, из которой первым удаляется тот элемент, который был записан в структуру последним?

5. Почему при удалении элемента из однонаправленного списка необходимо использовать дополнительную переменную?

1.3 Лабораторная работа «Простые сортировки на месте»

Цель работы: ознакомиться и реализовать простые методы сортировки – сортировки обменом, выбором, вставками, бинарными вставками. Исследовать сложность указанных алгоритмов сортировки.

Форма проведения: выполнение индивидуального задания.

Рекомендации по подготовке к лабораторной работе: для выполнения лабораторной работы необходимо ознакомиться с алгоритмами простых сортировок.

Сортировка обменом. Одним из простых методов сортировки на месте (т.е. при работе не требуется дополнительный объем памяти) является сортировка обменом или «пузырьковая» сортировка.

Суть алгоритма состоит в следующем: сравниваются пары рядом стоящих элементов массива, если первый элемент пары меньше второго, то элементы меняются местами. После первого просмотра массива самый большой элемент встает на свое место, а маленькие по значению элементы на один шаг продвигаются к началу массива. Отсюда метод и получил свое название: «легкие» элементы плавно «всплывают» к началу массива. «Тяжелые» элементы быстро «тонут», встают в конец массива.

После того, как все пары элементов массива просмотрены, массив еще не будет отсортирован, поэтому необходимо просмотреть пары элементов еще раз, и тогда еще один самый большой элемент встанет на свое место.

Если массив состоит из n элементов, то пар в таком массиве ровно *n*-1. На каждом шаге алгоритма самый большой элемент массива становится на свое место. Поэтому количество просматриваемых пар уменьшается с каждым шагом на единицу.

Таким образом, в алгоритме явно просматриваются два вложенных цикла. Внутренний цикл отвечает за просмотр пары элементов, количество шагов этого цикла зависит от внешнего цикла. Чем больше шагов выполнил внешний цикл, тем меньше пар просматривает внутренний цикл. Т.к. при выполнении одного шага внешнего цикла самый большой элемент становится на место, то количество шагов цикла равно количеству элементов массива -1. Однако, массив может быть уже отсортированным, до окончания работы внешнего цикла. Можно досрочно остановить выполнение цикла, если на каком-то i- том шаге во внутреннем цикле не произошло ни одного обмена.

Покажем применение сортировки на массиве *1 5 2 4 3.*

1: 1 2 4 3 ***5***

2: 1 2 3 ***4 5***

3: ***1 2 3 4 5***

После третьего шага алгоритм может закончить свою работу, так как не было проведено ни одного обмена.

Сортировка выбором. Сортировка выбором использует другой принцип упорядочивания элементов. На начальном шаге алгоритма выбирается минимальный элемент и ставится на место нулевого элемента. На последующих, *i*-тых шагах выбирается минимальный элемент среди элементов от *i*-того до (*n-1)*-го.

Рассмотрим на примере массива *1 5 2 4 3.*

1: ***1*** *5 2 4 3* Первый минимальный элемент уже стоит на своем месте.

2: ***1 2*** *5 4 3* Второй минимальный элемент – 2, поменяем его местами с 5.

3: ***1 2 3*** *4 5* Третий минимальный элемент – 3, поменяем его местами с 5.

4: ***1 2 3 4*** *5* Четвертый минимальный элемент стоит на своем месте.

После этого сортировка заканчивается, т.к. последний элемент в любом случае будет стоять на своем месте.

Сортировка вставками. Сортировка вставками упорядочивает массив, выполняя следующие действия – на начальном шаге алгоритма считаем, что последовательность из одного, первого элемента упорядоченная последовательность. Найдем место в этой последовательности для второго элемента. После этого упорядочены уже первые два элемента. Далее в текущей упорядоченной последовательности ищутся места для третьего, четвертого и т.д. элементов.

При программировании поиск места для вставляемого элемента лучше всего осуществить с конца упорядоченной последовательности.

Рассмотрим на примере массива *1 5 2 4 3.*

1: ***1 5*** *2 4 3* В упорядоченную последовательность добавляется 5.
2: ***1 2 5*** *4 3* В упорядоченную последовательность добавляется 2.
3: ***1 2 4 5*** *3* В упорядоченную последовательность добавляется 4.
4: ***1 2 3 4 5*** В упорядоченную последовательность добавляется 3.

Порядок выполнения работы

1. Получить индивидуальный вариант.
2. Составить алгоритм простой сортировки.
3. Реализовать алгоритм на языке Си, добавив в программу подсчет количества сравнений и перестановок, проведенных алгоритмом.
4. Найти среднее количество сравнений и перестановок, выполняемых программой для сортировки массивов из 100, 200, 300, 500, …, 10000 элементов, результаты сохраните в текстовом файле.
5. Построить графики зависимости сложности алгоритма (количество сравнений + количество перестановок) от количества обрабатываемых данных.
6. Сделать выводы по работе.

Контрольные вопросы

1. Какая оценка вычислительной сложности у алгоритма сортировки обменом?
2. Сколько перестановок выполняет алгоритм сортировки выбором?
3. Является ли устойчивым метод сортировки вставками?
4. Является ли сортировка обменом естественной?
5. Сколько сравнений и перестановок выполнит сортировка вставками в лучшем случае?

1.4 Лабораторная работа «Оптимизация простых сортировок»

Цель работы: ознакомиться и реализовать оптимизированные методы сортировок – шейкерную сортировку (оптимизация обмена), сортировку бинарными вставками и сортировку вставками со сторожевым элементом (оптимизация простых вставок).

Форма проведения: выполнение индивидуального задания.

Рекомендации по подготовке к лабораторной работе: для выполнения лабораторной работы необходимо ознакомиться с возможными способами оптимизации простых сортировок.

Шейкерная сортировка. Сортировка перемешиванием, или шейкерная сортировка – разновидность сортировки обменом. Анализируя метод пузырьковой сортировки, можно отметить два обстоятельства.

Во-первых, если при движении по части массива перестановки не происходят, то эта часть массива уже отсортирована и, следовательно, её можно исключить из рассмотрения.

Во-вторых, при движении от конца массива к началу минимальный элемент "всплывает" на первую позицию, а максимальный элемент сдвигается только на одну позицию вправо.

Эти две идеи приводят к следующим модификациям в методе пузырьковой сортировки. Границы рабочей части массива (т.е. части массива, где происходит движение) устанавливаются в месте последнего обмена на каждой итерации. Массив просматривается поочередно справа налево и слева направо.

Сортировка бинарными вставками. Сортировка бинарными вставками некоторым образом улучшает алгоритм сортировки вставками, увеличивая скорость нахождения места для вставляемого элемента. Метод использует информацию о том, что часть выборки уже отсортирована. Тогда, можно организовать поиск места для вставки нового элемента следующим образом: пусть переменная F обозначает индекс начала упорядоченной последовательности, а L – индекс конца. Найдем индекс элемента, находящегося в центре отсортированной последовательности – $M = (F + L)2$. Сравним вставляемый элемент *X[j]* с элементом *X[M]*. Если *X[j]> X[M]*, то изменим F на M (будем искать в правой части последовательности). Если *X[j]< =X[M]*, то изменим L на M (будем искать в левой части последовательности). Описанные действия будем проводить до тех пор, пока $L>F$. После выхода из цикла место для вставки элемента найдено.

Рассмотрим работу алгоритма на примере уже упорядоченной последовательности:

1 2 3 5 6 7 8 9 10 ***4.***

F=0, L = 8, M = 4 X[4]=6 > 4, следовательно:

F = 0 L = 4 M = 2 X[2]=3 < 4, следовательно:

F = 2 L = 4 M = 3 X[3]=5 >4, следовательно:

F = 2, L = 3 M = 2 (по правилам деления целых чисел в Си 5/2 = 2) X[2]= 3 < 4, следовательно F = 3 L = 3.

Так как F и L равны (условие выполнения цикла), то вставляемый элемент должен занимать в упорядоченной последовательности место с номером 3.

Сортировка вставками со сторожевым элементом. Еще одна оптимизация метода простых вставок. Очевидно, что на каждом шаге внутреннего цикла выполняется два сравнения – вставляемый элемент сравнивается с просматриваемым элементом и отслеживается возможный выход за границы массива. Второго сравнения можно избежать, проведя перед сортировкой некоторые мероприятия.

1 вариант. Перед сортировкой можно найти минимальный элемент и поменять его местами с первым элементом массива. Таким образом, ни один из оставшихся элементов массива не будет больше первого элемента и второе сравнение во внутреннем цикле можно опустить.

2 вариант. При инициализации массива выделяется память для хранения n+1 элемента (n – размерность массива). Все элементы массива записываются, начиная со второго элемента. Далее, на каждом шаге внешнего цикла в первую позицию записывается вставляемый элемент. После выполнения этих действий так же можно опустить второе сравнение во внутреннем цикле.

Порядок выполнения работы

1. Получить индивидуальный вариант.

2. Составить алгоритм оптимизированной сортировки.

3. Реализовать алгоритм на языке Си, добавив в программу подсчет количества сравнений и перестановок, проведенных алгоритмом.

4. Найти среднее количество сравнений и перестановок, выполняемых программой для сортировки массивов из 100, 200, 300, 500, …, 10000 элементов, результаты сохраните в текстовом файле.

5. Построить графики зависимости сложности алгоритма (количество сравнений + количество перестановок) от количества обрабатываемых данных.

6. Сравнить полученные результаты с результатами лабораторной работы «Простые сортировки на месте».

7. Сделать выводы по работе.

Контрольные вопросы

1. Какая оценка вычислительной сложности у алгоритма сортировки обменом?
2. Сколько перестановок выполняет алгоритм сортировки выбором?
3. Является ли устойчивым метод сортировки вставками?
4. Является ли сортировка обменом естественной?
5. Сколько сравнений и перестановок выполнит сортировка вставками в лучшем случае?

1.5 Лабораторная работа «Улучшенные методы сортировки»

Цель работы: ознакомиться и реализовать методы сортировок, оценка скорости которых не является квадратичной.

Форма проведения: выполнение индивидуального задания.

Рекомендации по подготовке к лабораторной работе: для выполнения лабораторной работы необходимо ознакомиться с сортировками – Хоара, Шелла, пирамидальной, подсчета.

Сортировка Шелла. Сортировка Шелла – алгоритм сортировки, являющийся усовершенствованным вариантом сортировки **вставками**. Идея метода Шелла состоит в сравнении элементов, стоящих не только рядом, но и на определённом расстоянии друг от друга. Иными словами – это сортировка вставками с предварительными «грубыми» проходами.

Сортировка Шелла была названа в честь её изобретателя – Да Шелла, который опубликовал этот алгоритм в 1959 году.

Выбор длин промежутков

– Первоначально используемая Шеллом последовательность длин промежутков: $d_1 = N/2, d_i = d_{i-1}/2, d_k = 1$ в худшем случае, сложность алгоритма составит $O(n^2)$.

– Предложенная Хиббардом последовательность: $2^i - 1 \leq N, i \in N$. Такая последовательность шагов приводит к алгоритму сложностью $O(n^{3/2})$, массив шагов заполняется перед сортировкой.

– Предложенная Седжвиком последовательность: $d_i = 9 \cdot 2^i - 9 \cdot 2^{i/2} + 1$, если i четное и $d_i = 8 \cdot 2^i - 6 \cdot 2^{(i+1)/2} + 1$, если i нечетное. При использовании таких приращений средняя сложность алгоритма составляет: $O(n^{7/6})$, а в худшем случае порядка $O(n^{4/3})$. Массив приращений заполняется перед сортировкой. Последнее значение массива шаг[s-1], если 3*шаг[s] > N (если размер массива меньше 3-х шагов).

– Наиболее часто используемая последовательность шагов - d_i изменяется по правилу $d_{i+1} = (d_i - 1)/2$ (для массивов, содержащих более 500 элементов) и $d_{i+1} = (d_i - 1)/3$ (для массивов, содержащих менее 500 элементов). За d_0 принимается число элементов массива. Метод заканчивает работу, когда d_i становится меньше 1.

Комбинированная сортировка (сортировка «расческой»). Комбинация сортировки обменом и сортировки Шелла. На каждом шаге сравниваются значения, отстоящие друг от друга на заданное значение шага $H_{i+1} = 8 \cdot H_i/11$, но такое сравнение происходит всего один раз. Как только значение смещения становится равным 1, выполняется сортировка до конца методом пузырька. За H_0 принимается число элементов массива.

Пирамидальная сортировка. Пирамида – это частично упорядоченное двоичное дерево, элементы которого расположены в узлах дерева по следующему правилу – каждый элемент родительского узла обязательно больше элементов, расположенных в дочерних узлах. На рис. 1.14 представлена пирамида из 15 элементов:

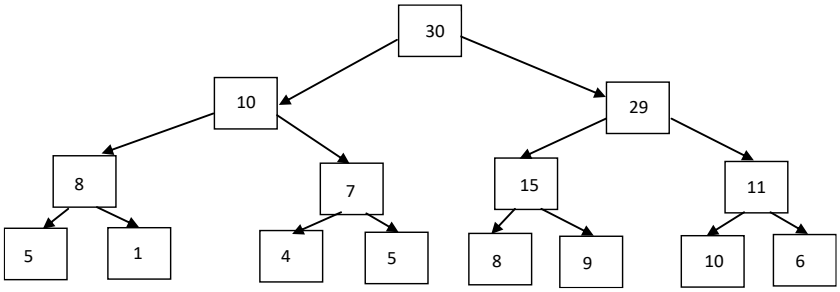

Рисунок 1.14 – Пирамидально упорядоченный массив

Элементы дерева легко представляются в виде массива – пусть родительский узел имеет индекс i, тогда дочерние узлы имеют индексы $2i$ и $2i + 1$. Рассмотренная пирамида может быть представлена массивом: (30, 10, 29, 8, 7, 15, 11, 5, 1, 4, 5, 8, 9, 10, 6).

Т.к. корневой элемент элемент пирамиды всегда является максимальным элементом, то процесс пирамидальной сортировки можно описать следующим образом: поменять верхний элемент пирамиды с нижним элементом и рассматривать в дальнейшем не n элементов исходного массива, а $n - 1$ элемент. При выполнении этих действий нарушается правило расположения элементов в пирамиде, поэтому после обмена необходимо перестроить пирамиду с $n - 1$ элементами и далее, повторять два этих шага, пока пирамида не останется пустой.

Таким образом, необходимо написать процедуру, строящую пирамиду для произвольного массива размерности *n*, далее алгоритм пирамидальной сортировки очень прост. Для формального описания алгоритма назовем процедуру построения пирамиды из массива размерностью *N KeyDown(N)* – т.к. элемент, находящийся в корне пирамиды может быть и не самым большим, то необходимо опустить этот элемент на нижние уровни пирамиды, чтобы выполнялась частичная упорядоченность. Алгоритм может выглядеть следующим образом:

```
1. KeyDown(N,X); // Построение пирамиды на исходном массиве x.
2. X[1] ←→ X[N]; // Обмен первого элемента пирамиды с последним
3. L = N-1; // Изменение размерности пирамиды
4. Пока (L>1) // пока пирамида не пуста
      KeyDown(N-1,X);
   X[1] ←→ X[L];
      L = L-1;
5. Конец.
```

Рассмотрим процесс построения пирамиды на произвольном массиве (в скобках после элементов указаны индексы):

Элементы массива: (25(1), 11(2), 5(3), 11(4), 4(5), 8(6), 3(7), 28(8), 18(9), 10(10), 1(11), 5(12), 4(13), 2(14), 17(15)). Начальное расположение элементов массива в узлах пирамиды показано на рис. 1.15.

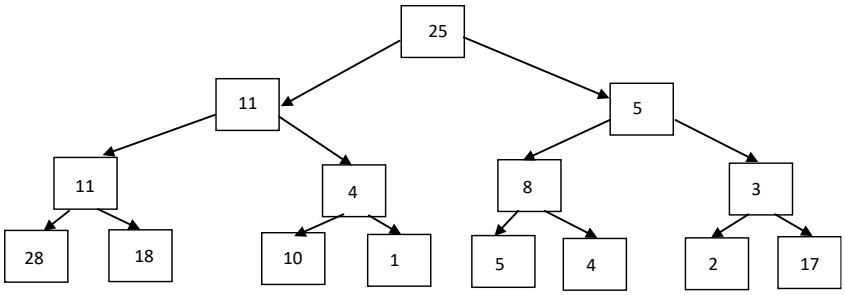

Рисунок 1.15 – Первый этап построения пирамиды

Размерность массива $N = 15$. Для элементов, находящихся на нижнем уровне, не существует дочерних элементов, т.е. эти элементы могут не проверяться на выполнение правила пирамиды, индексы этих элементов от $N/2+1$ до N. Поэтому построение начинается с элемента с номером $N/2$, в рассматриваемом примере это x[7] = 3, сравним этот элемент с наибольшим из элементов x[14] и x[15], вторая нижняя пирамида остается без изменения, третья и четвертая пирамиды изменяются (см. рис. 1.16).

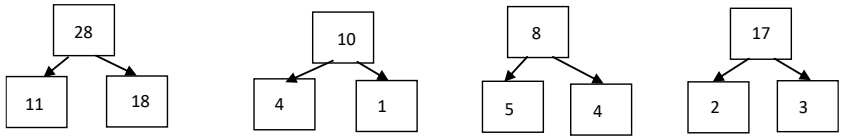

Рисунок 1.16 – Изменение порядка элементов нижнего уровня

Далее необходимо рассмотреть пирамиды с корневыми элементами во втором и третьем элементах (рис. 1.17):

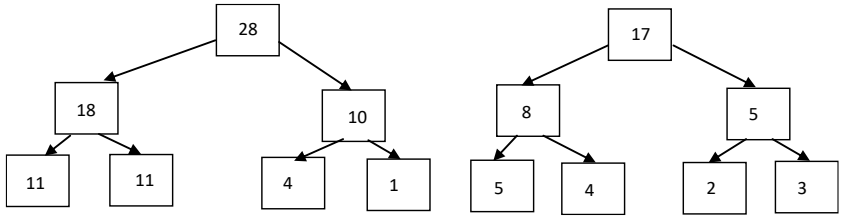

Рисунок 1.17 – Изменение пирамид среднего нижнего уровня

На рис. 1.18 изображен результат построения начальной пирамиды на массиве из 15 элементов.

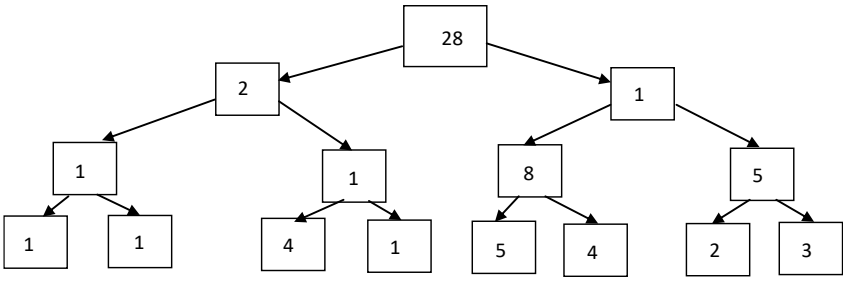

Рисунок 1.18 – Пирамидально упорядоченный массив

Очевидно, что процедура KeyDown(N,X) должна зависеть еще от одного параметра – номера элемента, для которого строится пирамида.

В общем случае для построения пирамиды с корнем в L-том элементе необходимо итеративно выполнить продвижение элемента по дереву вниз (при этом, элемент с номером L меняется местами с большим из своих потомков), продвижение элемента заканчивается, когда значение элемента в позиции L становится больше значений элементов-потомков, или когда достигнут нижний уровень.

Полный алгоритм пирамидальной сортировки выглядит следующим образом:

```
       1.  L = (N/2) +1;
       2.  Пока L>1
L = L-1;
KeyDown(L,N,X);
       3.  N1 = N;
       4.  Пока N1>1
V = x[1]; x[1] = x[N1]; x[N1] = v;
N1 = N1 – 1;
KeyDown(L,N1,X);
       5.  Конец.
```

Сортировка Хоара. Значение произвольного элемента, обычно центрального, принимается за значение опорного элемента. Организуется просмотр элементов массива. При движении по массиву слева направо ищется элемент больше или равный опорному. При движении справа налево ищется элемент меньше или равный опорному элементу. Найденные элементы меняются местами, далее продолжается встречный поиск. После этих действий массив окажется разделенным на две части. В первой части будут расположены элементы меньше либо равные опорному элементу, а справа – больше либо равные. Далее алгоритм рекурсивно выполняется для правой и левой частей.

Сортировка Хоара с выбором медианного элемента. Можно улучшить быструю сортировку, выбирая средний элемент таким образом, чтобы его значение было бы действительно близким к серединному значению массива. Для этого можно воспользоваться двумя стратегиями:

Выбор среднего значения осуществляется случайным образом (с использованием датчиков случайных чисел и информации о размерности массива). Т.к. разделяющий элемент выбирается при каждом вызове процедуры, случайный выбор может быть наиболее правильным и оградит от появления наихудшего случая – когда медианный элемент оказывается наименьшим или наибольшим.

Вторая стратегия состоит *в случайном выборе трех элементов*, по одному из начального, конечного и среднего интервалов сортируемого подмассива. Как разделяющий элемент используется среднее из этих трех чисел.

Порядок выполнения работы

1. Получить индивидуальный вариант.
2. Составить алгоритм сортировки.
3. Реализовать алгоритм на языке Си, добавив в программу подсчет количества сравнений и перестановок, проведенных алгоритмом.
4. Найти среднее количество сравнений и перестановок, выполняемых программой для сортировки массивов из 100, 200, 300, 500, …, 10000 элементов, результаты сохраните в текстовом файле.
5. Построить графики зависимости сложности алгоритма (количество сравнений + количество перестановок) от количества обрабатываемых данных.
6. Сравнить полученные результаты с результатами лабораторных работ «Простые сортировки на месте» и «Оптимизация простых сортировок».
7. Сделать выводы по работе.

Контрольные вопросы

1. Количество каких элементарных операций уменьшается в сортировке Шелла?
2. Является ли устойчивой сортировка Хоара?
3. В каком случае сортировка Хоара показывает наихудшие результаты?

4. Почему при сравнении элементов на расстоянии уменьшается общее количество сравнений и перестановок, выполняемых алгоритмом сортировки?

5. Какова вычислительная сложность пирамидальной сортировки?

1.6 Лабораторная работа «Сортировка слиянием»

Цель работы: ознакомление с принципами сортировки слиянием, исследование сложности алгоритма.

Форма проведения: выполнение индивидуального задания.

Рекомендации по подготовке к лабораторной работе: для выполнения лабораторной работы необходимо ознакомиться с двумя способами слияния и разбиения.

Сортировка слиянием. Слияние – объединение двух отсортированных подмассивов в один. Таким образом, слияние сортирует два подмассива и сливает их для получения отсортированного массива.

Слияние не зависит от характера входных данных. Основной недостаток метода – необходимость дополнительного объема памяти.

Сортировка слиянием используется в следующих случаях:

- Быстродействие выходит на первое место
- Легко применяется для структур с последовательным доступом к данным
- При решении задач следующего типа – основной набор данных уже отсортирован, в систему постоянно поступают новые данные, которые необходимо разместить, не теряя упорядоченности всего набора данных.

Способы программирования сортировки

Нисходящая сортировка слиянием. Первый способ программирования сортировки очень напоминает способ программирования сортировки Хоара. Функция сортировки рекурсивная, зависит от сортируемого массива и границ сортируемого массива.

На рис. 1.19 изображено дерево разбиений, которое строит вышеописанный алгоритм на массиве из 25 элементов (числа в узлах дерева – количество подмассивов).

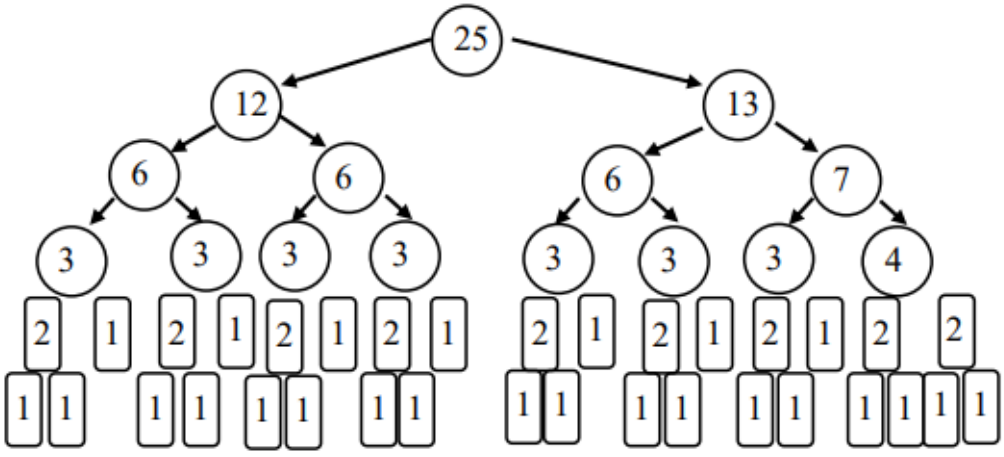

Рисунок 1.19 – Дерево разбиений, построенное нисходящим слиянием

Как только алгоритм выполнил разбиение на подмассивы по одному элементу (нижний уровень дерева) начинает свою работу алгоритм слияния.

Восходящая сортировка слиянием. Дерево разбиений, построенное алгоритмом восходящей сортировки, представлено на рис. 1.20.

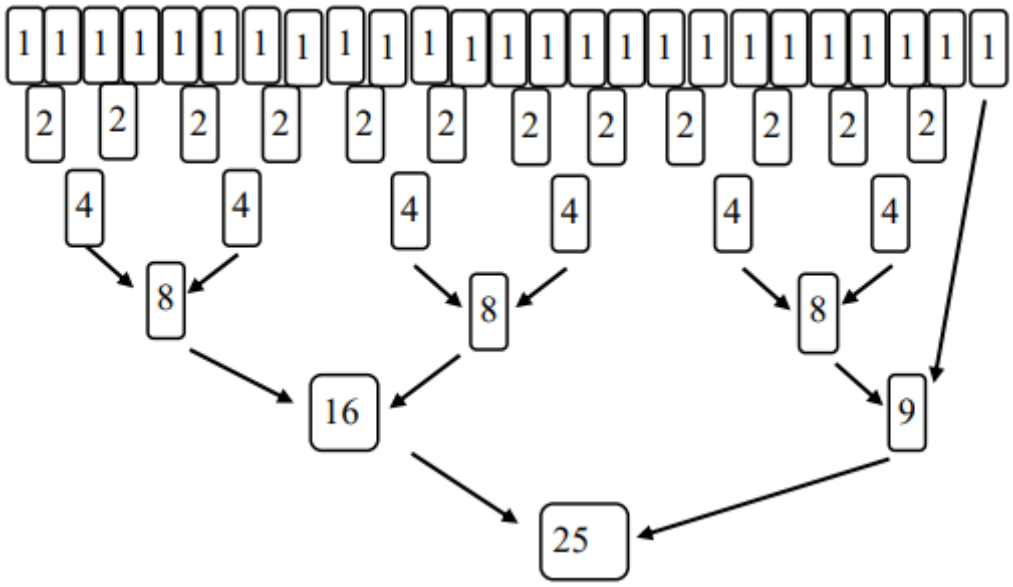

Рисунок 1.20 – Дерево разбиений, построенное восходящим слиянием

Восходящая сортировка слиянием просматривает массив слева направо, и на первом шаге сливает рядом стоящие одиночные элементы в упорядоченные пары элементов. На втором шаге – рядом стоящие упорядоченные пары в упорядоченные четверки элементов. И так далее. Алгоритм восходящей сортировки не рекурсивен. Функция, выполняющая слияние вызывается сразу же после выделения границ объединяемых подмассивов.

Порядок выполнения работы

1. Получить индивидуальный вариант.
2. Составить алгоритм сортировки.
3. Реализовать алгоритм на языке Си, добавив в программу подсчет количества сравнений и перестановок, проведенных алгоритмом.
4. Найти среднее количество сравнений и перестановок, выполняемых программой для сортировки массивов из 100, 200, 300, 500, …, 10000 элементов, результаты сохраните в текстовом файле.
5. Построить графики зависимости сложности алгоритма (количество сравнений + количество перестановок) от количества обрабатываемых данных.
6. Сравнить полученные результаты с результатами предыдущих лабораторных работ.
7. Сделать выводы по работе.

Контрольные вопросы

1. Какие способы разделения массива на подмассивы Вы знаете?
2. Какие способы слияния Вы знаете?
3. В чем преимущество абстрактно-обменного слияния?
4. Перечислите достоинства сортировки слиянием.
5. Перечислите недостатки сортировки слиянием.

1.7 Лабораторная работа «Поразрядная сортировка»

Цель работы: ознакомление с принципами поразрядной сортировки, исследование сложности алгоритма.

Форма проведения: выполнение индивидуального задания.

Рекомендации по подготовке к лабораторной работе: перед проведением занятия необходимо изучить теоретический материал, изложенный в курсе лекций. В качестве дополнительных источников рекомендуется ознакомиться с материалом, представленным в [2]. В главе 5 (стр. 147 – 155).указанного пособия рассматриваются основные принципы поразрядных сортировок.

Порядок выполнения работы

1. Получить индивидуальный вариант.
2. Составить алгоритм сортировки.
3. Реализовать алгоритм на языке Си.
4. Построить графики зависимости сложности алгоритма (количество сравнений + количество перестановок) от количества обрабатываемых данных.
5. Сравнить полученные результаты с результатами предыдущих лабораторных работ.
6. Сделать выводы по работе.

Контрольные вопросы

6. Какие способы разделения массива на подмассивы Вы знаете?
7. Какие способы слияния Вы знаете?
8. В чем преимущество абстрактно-обменного слияния?
9. Перечислите достоинства сортировки слиянием.
10. Перечислите недостатки сортировки слиянием.

1.8 Лабораторная работа «Двоичные деревья »

Цель работы: закрепление навыков программирования динамических структур на примере программирования деревьев двоичного поиска (BST – binary search tree).

Форма проведения: решение индивидуального задания.

Рекомендации по подготовке к работе: перед проведением занятия необходимо изучить теоретический материал, изученный в лекционном курсе дисциплины, а так же изложенный в [2]. Раздел пособия 4.4 (стр. 191 – 205).

Порядок проведения занятия

1. Получить индивидуальный вариант.
2. Составить алгоритм решения задания.
3. Реализовать алгоритм на языке Си.
4. Защитить реализованную программу.

Контрольные вопросы

1. Поясните состав структуры, использующейся для описания узла дерева.
2. Расскажите алгоритм добавления нового узла в BST-дерево.
3. Какой из обходов дерева выведет элементы, хранящиеся в узлах дерева в отсортированном виде?
4. Какова вычислительная сложность алгоритма поиска элемента BST-дерева?.
5. Какой обход дерева (в глубину/в ширину) использует алгоритм обратного обхода?

1.9 Лабораторная работа «Двоичные деревья. Операции над деревьями»

Цель работы: закрепление навыков программирования динамических структур на примере программирования деревьев двоичного поиска (BST – binary search tree).

Форма проведения: выполнение индивидуального задания.

Рекомендации по подготовке к работе: перед выполнением работы необходимо изучить теоретический материал, рассмотренный на лекционных занятиях; изложенный в [3]. Раздел пособия 4.4 (стр. 206 – 227).

Порядок проведения занятия

1. Получить индивидуальный вариант.
2. Составить алгоритм решения задания.
3. Реализовать алгоритм на языке Си.
4. Защитить реализованную программу.

Контрольные вопросы

1. Объясните алгоритм добавления элемента в корень дерева.
2. В каких алгоритмах на деревьях используется алгоритм разделения дерева относительно k-того наименьшего?
3. Для чего выполняется балансировка построенного дерева?
4. Покажите на примере дерева из 10 элементов операцию ротации вправо.
5. Покажите на примере дерева из 10 элементов операцию ротации влево.

2 МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ПРОВЕДЕНИЮ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

2.1 Практическое занятие «Двоичные файлы»

Цель занятия: освоить навыки работы с двоичными файлами.

Форма проведения: решение практических задач, связанных с обработкой информации, хранящейся в двоичных файлах.

Рекомендации по подготовке к занятию: перед проведением занятия необходимо изучить теоретический материал, изложенный в [4]. Глава 9 пособия, стр. 156 – 66.

Порядок проведения занятия

1. Обсуждение темы занятия в аудитории – краткий обзор темы занятия.
2. Выборочный опрос студентов.
3. Решение задач (реализация, отладка программы).
4. Тестирование реализованных задач.
5. Обсуждение полученных результатов.

Примеры задач

Вариант 1

Создать файл, содержащий матрицу действительных чисел A размерности $n \times n$. Не считывая матрицу в память, просмотреть элементы главной диагонали. Если диагональный элемент отрицательный, то дописать его в конец этого файла, а на его место записать 999.99. Вывести на экран информацию в виде: сначала напечатать замененные диагональные элементы, а затем – матрицу размерности $n \times n$ с замененными диагональными элементами. Определить и напечатать число компонентов в реорганизованном файле.

Вариант 2

Создать файл, содержащий матрицу целых чисел A размерности $n \times n$. Не считывая матрицу в память заменить элементы строк с первым положительным элементом на элементы столбца с таким же номером. Вывести на печать построчно исходную и полученную матрицы.

Вариант 3

Создать файл, содержащий вещественные числа. Выбрать из него элементы с номерами, кратными *k*, и записать их в новый файл, затем – с кратными *k1* записать в конец нового, затем – с кратными *k2* и записать в конец файла. Вывести на печать: имена старого и нового файлов; количество записей в них; новый файл в виде матрицы с числом столбцов 10. Старый файл уничтожить. Значения *k, k1, k2* вводить с клавиатуры.

Вариант 4

Создать файл целых чисел, содержащий матрицу A размерности $n \times m$. Не считывая матрицу в память, реорганизовать этот файл, оставив в нем строки, первый элемент которых меньше заданного *k*. Напечатать количество записей в реорганизованном файле и максимальный элемент полученной матрицы с указанием индексов. Значение *k* вводить с клавиатуры.

Вариант 5

Создать файл целых чисел, содержащий матрицу целых чисел A размерности $n \times m, n > m$. Не считывая матрицу в память, удалить из файла все записи, соответствующие строкам с номерами *k1, k2,..., km*, записав их в новый файл. Значения удаляемых строк и их количество вводить с клавиатуры. Распечатать полученные два файла в виде матриц соответствующих размерностей с указанием имен файлов.

2.2 Практическое занятие «Структурные переменные»

Цель занятия: закрепление навыков работы со структурированными данными.

Форма проведения: выполнение индивидуального задания.

Рекомендации по подготовке к занятию: перед занятием необходимо изучить теоретический материал соответствующей лекции, а так же, ознакомиться с материалом, изложенным в [4]. В главе 4 пособия содержится справочная информация об объявлении такого типа данных, как структура (стр. 67 – 69).

Порядок выполнения работы

1. Получить индивидуальный вариант.
2. Создать в папке проекта текстовый файл, содержащий описанные в задании данные.
3. Написать программу, решающую поставленную задачу с использованием функций.
4. Отладить и протестировать программу.
5. Защитить работу.

Пример выполнения индивидуального варианта

Дан массив записей, содержащих информацию о сдаче студентами одной группы экзаменов по математике, физике и программированию. Расположить записи в массиве по убыванию оценки по математике. Вывести отсортированный массив на экран.

```
int main(int argc, char *argv[]) {
  system("chcp 1251");
  typedef struct {
  struct Name{
     char surname[30];
     char name[20];
     char patronymic[30];
    } nstudent;
  int m,f,p;
} Student;
int n = 5,i,vs,j;
Student *array, S_vs;
array = (Student*) malloc(sizeof(Student)*n);
for ( i=0;i<n;i++){
    printf("Вводите информацию о студенте: \n");
    printf("Фамилия: ");
    scanf("%s", array[i].nstudent.surname);
    printf("Имя: ");
    scanf("%s", array[i].nstudent.name);
    printf("Отчество: ");
    scanf("%s", array[i].nstudent.patronymic);
    printf("Математика: ");
    scanf("%d",&array[i].m);
    printf("Физика: ");
    scanf("%d",&array[i].f);
    printf("Программирование: ");
    scanf("%d",&array[i].p);
```

```
 }
printf("Исходные данные: \n");
for(i=0;i<n;i++)
    printf("%15s%15s%3d%3d%3d\n",array[i].nstudent.surname,
                                 array[i].nstudent.name,
                                 array[i].m, array[i].f,
                                 array[i].p);
for (i=1;i<n;i++)
  {  S_vs = array[i];
     vs = array[i].m;
     j = i-1;
     while(vs>array[j].m&&j>=0)
       {array[j+1]=array[j];
        j--;}
     array[j+1] = S_vs;
  }
printf("\n После сортировки:\n");
for(i=0;i<n;i++)
  printf("%15s%15s%3d%3d%3d\n",array[i].nstudent.surname,
                               array[i].nstudent.name,
                               array[i].m, array[i].f,
                               array[i].p);

return 0;}
```

2.3 Практическое занятие «Рекурсивные функции»

Цель занятия: закрепление навыков реализации рекурсивных функций.

Форма проведения: выполнение индивидуального задания.

Рекомендации по подготовке к занятию: при подготовке к заданию необходимо изучить материал, изложенный в лекционном курсе, так же, рекомендован к изучению материал пособия [4], стр. 110 – 113 и пособия [2], стр. 131 – 142.

Примеры заданий

Вариант 1

Напишите рекурсивную и не рекурсивную функции, реализующие алгоритм вычисления факториала натурального числа *n*.

Вариант 2

Напишите рекурсивную и не рекурсивную функции, реализующие алгоритм возведения в степень *n* вещественного числа *a* (*n* – натуральное число).

Вариант 3

Напишите рекурсивную и не рекурсивную функции, реализующие алгоритм вычисления суммы цифр натурального числа.

2.4 Практическое занятие «Численные методы»

Цель занятия: реализация численных методов.

Форма проведения: работа в группах.

Порядок проведения занятия

1. Преподаватель назначает руководителей групп (8 человек).
2. Руководитель группы подбирает команду из 2 – 3 человек.
3. Преподаватель назначает вариант группового задания.
4. Команда изучает полученный численный метод, составляет алгоритм метода. Далее выполняется изоляция корней (экстремумов) для полученных целевых функций. При этом можно применить графический способ, используя онлайн-построение графиков функций – http://yotx.ru/.
5. Команда выполняет программную реализацию метода и проверяет правильность работы программы.

Примеры заданий

1. Метод итераций для решения нелинейного уравнения.
2. Метод секущих для решения нелинейного уравнения.
3. Метод касательных для решения нелинейного уравнения.

2.5 Практическое занятие «Численные методы. Защита»

Цель занятия: получение навыков публичной защиты программных продуктов.

Форма проведения: защита реализованных численных методов.

Рекомендации по подготовке к занятию: для защиты работы, подготовьте слайды и доклад на 3 – 4 минуты. Рекомендуемый план доклада: представление группы и распределение обязанностей между участниками, описание численного метода, результаты графического отделения корней (экстремумов), результаты тестирования программы. Рекомендуемое содержимое слайдов: титульный слайд с названием метода и перечислением участников группы, слайд с алгоритмом метода, слайд с кодом программы, слайд с графиками функций, слайд с результатами тестирования.

2.6 Практическое занятие «Поиск»

Цель занятия: ознакомление с принципами организации поиска подстроки в строке.

Форма проведения: решение практических задач.

Рекомендации по подготовке к занятию: перед проведением занятия необходимо изучить теоретический материал, изложенный в [3]. Раздел пособия 1.9 (стр. 55 – 66).

Порядок проведения занятия

1. Обсуждение основных принципов алгоритмов поиска строк.
2. Выборочный опрос студентов.
3. Решение задач (реализация, отладка программы).
4. Тестирование реализованных задач.
5. Анализ полученных результатов.

Примеры задач

Вариант 1

Напишите программу, реализующую алгоритм поиска Боуера-Мура.

Вариант 2

Напишите программу, реализующую алгоритм поиска Кнута, Морриса и Пратта.

Вариант 3

Напишите программу, реализующую алгоритм прямого поиска подстроки в строке.

2.7 Практическое занятие «Обработка строк»

Цель работы: реализация алгоритмов работы со строками, изучение стандартных функций для работы со строками в языке Си.

Форма проведения: выполнение индивидуального задания.

Рекомендации по подготовке к занятию: для выполнения индивидуального задания необходимо изучить теоретический материал пособия [4]. В главе 8 пособия содержится справочная информация по организации работы со строками (стр. 133 – 143). При подготовке к занятию обратите особое внимание на представление строки в памяти компьютера (стр. 134) и на примеры работы со строками (стр. 139 – 143).

Порядок выполнения работы

1. Обсуждение основных принципов работы со строками в Си.
2. Выборочный опрос студентов.
3. Решение задач (реализация, отладка программы).
4. Тестирование реализованных задач.
5. Анализ полученных результатов.

2.8 Практическое занятие «Алгоритмы на графах»

Цель работы: реализация алгоритмов на графах.

Форма проведения: выполнение индивидуального задания.

Рекомендации по подготовке к занятию: для выполнения индивидуального задания необходимо изучить теоретический материал пособия [2]. В главе 10 пособия разобраны алгоритмы на графах (стр. 250 – 280).

Порядок выполнения работы

1. Краткая теоретическая справка по теме занятия.
2. Выборочный опрос студентов.
3. Решение задач (реализация, отладка программы).
4. Тестирование реализованных задач.
5. Анализ полученных результатов.

3 МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ОРГАНИЗАЦИИ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

3.1 Общие положения

Самостоятельная работа является важной составляющей в изучении дисциплины и состоит из следующих видов деятельности: проработка лекционного материала для подготовки к тестированию и контрольным работам, подготовка к лабораторным работам и практическим заданиям, выполнение домашних заданий, выполнение контрольных работ, самостоятельное изучение тем курса.

Самостоятельная работа над теоретическим материалом направлена систематизацию и закрепление знаний, полученных на лекционных занятиях и на получение новых знаний по дисциплине, путем самостоятельного изучения тем.

Самостоятельная работа по подготовке к лабораторным работам и практическим занятиям направлена на изучение методического и теоретического материала по теме лабораторной работы или практического занятия.

Выполнение домашних заданий (для студентов очной формы обучения) и контрольных работ (для студентов заочной формы обучения) – полностью самостоятельная работа, направленная на получение навыков самостоятельного составления алгоритмов, реализацию программ, их дальнейшей отладки и тестирования.

3.2 Проработка лекционного материала, подготовка к контрольным работам, лабораторным работам и практическим занятиям

Проработка лекционного курса является одной из важных активных форм самостоятельной работы. Этот вид самостоятельной работы может быть организован следующим образом:

– прочитайте конспект лекции, согласовав Ваши записи с информацией на слайдах лекции;

– попробуйте выполнить самостоятельно примеры программ, разобранных на лекции;

– если в лекции рассматривался какой-либо алгоритм, попытайтесь выполнить этот алгоритм на тестовых данных без использования компьютерной программы; такой способ проработки материалов лекции покажет, правильно ли Вы поняли идею алгоритма;

– изучите дополнительные учебные материалы, рекомендованные преподавателем;

– попытайтесь ответить на контрольные вопросы, которыми, как правило, заканчиваются разделы учебных пособий или учебников;

– если после выполненной работы Вы считаете, что материал освоен не полностью, сформулируйте вопросы и задайте их преподавателю.

Методические указания к ведению конспектов лекций. Лекции по дисциплине проводятся с использованием слайдов. Но это не означает, что лекцию можно просто слушать. Ведение конспектов значительно повышает качество последующей проработки лекционного материала. В силу специфики дисциплины на слайдах лекций очень много алгоритмов, кодов программ, примеров демонстрации работы изучаемых алгоритмов. Но этот материал может быть бесполезен, если Вы не делаете записи в течение лекции, потому что большинстве случаев, комментарии по представленным на слайдах примерам, лектор выполняет в устной форме.

Можно рекомендовать распечатывать слайды перед лекцией и вести конспект непосредственно на бумажном варианте слайд-презентации.

Одной из форм текущего мониторинга уровня знаний по дисциплине являются контрольные работы. В первом и втором семестре изучения дисциплины проводятся контрольные работы двух типов: тестовые опросы на лекции и контрольные работы, в которых студентам необходимо применить полученные знания на практике. Выполнение выше перечисленных действий поможет подготовиться и к выполнению контрольных работ. В приложении А указаны темы контрольных работ для студентов очной формы обучения и приведены примерные варианты.

Самостоятельная работа по подготовке к лабораторным работам и практическим занятиям по дисциплине состоит в изучении методических материалов по темам соответствующих видов аудиторных занятий.

Рекомендуется перед выполнением лабораторной работы изучить лекционный и методический материал по теме занятия, ознакомиться с алгоритмами, реализацию которых необходимо выполнить во время проведения занятия. Обратите особое внимание на порядок выполнения работы. Поскольку конечным результатом всех лабораторных работ является компьютерная программа, самостоятельно разработайте структурную схему будущей программы, выполните заготовку проекта, подготовьте самостоятельно тестовые данные. Если при подготовке к занятию остались нерешенные вопросы, обратитесь за консультацией к преподавателю.

3.3 Выполнение домашних заданий и контрольных работ

3.3.1 Общие положения

Выполнение домашних заданий для студентов очной формы обучения и выполнение контрольных работ для студентов заочной формы обучения – это полностью самостоятельный вид деятельности студента. Темами домашних заданий и контрольных работ являются темы дисциплины, не охваченные циклом лабораторных работ и практических занятий.

Выполнение домашнего задания состоит в написании программы по индивидуальному варианту. Обучающиеся самостоятельно разрабатывают алгоритм решения задания, реализуют разработанный алгоритм на языке программирования Си, отлаживают и тестируют написанную программу.

Защита домашнего задания проходит в сроки, установленные преподавателем. Процедура защиты представляет собой индивидуальное собеседование с преподавателем, примерный сценарий которого представлен ниже:

- комментирование студентом кода и логики написанной программы;
- ответы на вопросы преподавателя по коду и логике программы;
- комментирование студентом тестовых данных;
- демонстрация работы программы и пояснение полученных результатов.

Выполнение студентами заочной формы обучения контрольных работ аналогично выполнению домашних заданий студентами очной формы обучения. Контрольные работы выполняются самостоятельно, во время семестра, а защищаются перед преподавателем во время экзаменационных сессий, как это предусмотрено учебным планом дисциплины. Допускается общение с преподавателем во время семестра посредством электронной почты – выполненные контрольные работы могут быть высланы на предварительную проверку.

3.3.1 Домашнее задание «Текстовые файлы»

Цель домашнего задания – закрепление навыков работы с текстовыми файлами. Для выполнения задания необходимо изучить теоретический материал, изложенный в [4]. В главе 9 пособия описаны стандартные функции языка Си для работы с текстовыми файлами (стр. 144 – 155).

Примерный вариант

Напишите программу, читающую из текстового файла массив целых чисел произвольной размерности. Найдите среднее арифметическое элементов массива и допишите найденное значение в исходный текстовый файл с комментарием «Среднее арифметическое элементов массива – <*найденное значение*>».

3.3.2 Домашнее задание «Алгоритмы линейной алгебры»

Цель домашнего задания – показать взаимосвязь двух дисциплин, информатики и линейной алгебры. Для выполнения домашнего задания будет полезен справочный материал из разделов «Массивы» и «Матрицы» пособия [4], стр. 114 – 132, материал, изученный в курсах линейной алгебры.

Примерный вариант

Напишите программу, вычисляющую определитель матрицы 3x3.

3.3.3 Домашнее задание «Генерация комбинаторных объектов»

Цель домашнего задания – показать взаимосвязь двух дисциплин, информатики и дискретной математики. Для выполнения домашнего задания будет полезен справочный материал из разделов «Массивы» и «Матрицы» пособия [4], стр. 114 – 132, материал по комбинаторике пособия [2], стр. 204 – 223.

Примерный вариант

Запишите программу генерации сочетаний в лексикографическом порядке.

3.3.4 Домашнее задание «Многофайловая компиляция»

Цель домашнего задания – формирование навыков работы с проектами. При выполнении задания может быть полезен материал пособия [4], стр. 42 – 43, пособия [2], стр. 224 – 228.

Примерный вариант

Написать функции для работы с очередями с приоритетом. Запрещается использование глобальных переменных. Описания функций хранятся в отдельном заголовочном файле. Для каждого варианта необходимо написать функции:

- создание очереди из N элементов (N и значения самих элементов читать из текстового файла, имя файла запрашивать с клавиатуры);
- добавление нового элемента в очередь;
- удаление элемента с максимальным значением;
- удаление любого элемента с заданным значением;
- поиск элемента с заданным значением;
- изменение значения выбранного элемента;
- тестовый пример, демонстрирующий работу всех написанных функций.

3.3.5 Контрольная работа «Сортировка и поиск»

Контрольная работа «Сортировка и поиск» для студентов заочной формы обучения состоит из четырех заданий, которые необходимо выполнить в форме программы (или отдельных программ) на языке Си. Задания контрольной работы соответствуют темам «Сортировка слиянием», «Поразрядные сортировки», «Алгоритмы генерации комбинаторных объектов» и «Алгоритмы поиска подстроки в строке». При решении контрольной работы приветствуется создание минимального пользовательского интерфейса – при написании программы используйте строки приглашения, оформляйте вывод полученных значений с соответствующими пояснениями. Перед каждым выполненным заданием из контрольной работы в тексте программы обязательно должны быть написаны комментарии, содержащие следующую информацию – фамилия, имя, отчество студента, формулировка задания, словесное описание алгоритма решения задания. Методические материалы, которые могут быть полезны для решения заданий изложены в данном методическом пособии на стр. 18 – 20, 25 – 26, 29.

Примерный вариант

Задание 1. Напишите программу, реализующую нисходящую сортировку слиянием, метод слияния – абстрактно-обменный.

Задание 2. Напишите программу, реализующую MSD-сортировку массива строк.

Задание 3. Напишите программу, реализующую алгоритм генерации перестановок в лексикографическом порядке.

Задание 4. Напишите программу, реализующую КМП-алгоритм.

3.4 Самостоятельное изучение тем теоретической части курса

3.4.1 Двоичные файлы

Перечень вопросов, подлежащих изучению

1. Способы чтения информации из двоичного файла.
2. Способы записи информации в двоичный файл.
3. Организация прямого доступа в двоичных файлах.

Методические рекомендации по изучению

Изучите функции языка Си, позволяющие считывать информацию из файла и записывать ее в файл. Обратите внимание на многообразие функций языка для организации прямого доступа к файлу. Выделите для себя различия между текстовыми и двоичными файлами. Параллельно с изучением справочного материала попробуйте написать функции записи заданной информации в двоичный файл. Попытайтесь открыть двоичный файл в текстовом редакторе. Найдите объяснение увиденной информации.

Рекомендуемые источники

Пермякова, Н. В. Информатика и программирование: Учебное пособие [Электронный ресурс] / Н.В. Пермякова – Томск: ТУСУР, 2016. – 188 с. – Режим доступа: https://edu.tusur.ru/publications/7678, стр. 156 – 166.

Подбельский, В.В. Курс программирования на языке Си [Электронный ресурс] : учебник / В.В. Подбельский, С.С. Фомин. – Электрон. дан. – Москва : ДМК Пресс, 2012. – 384 с. – Режим доступа: https://e.lanbook.com/book/4148 . – Загл. с экрана. Стр. 322 – 334.

3.4.2 Поразрядные сортировки

Перечень вопросов, подлежащих изучению

1. Поразрядная MSD-сортировка.
2. Поразрядная LSD-сортировка.
3. Двоичная быстрая сортировка.

Методические рекомендации по изучению

При изучении этой темы обратите внимание на область применения таких сортировок. Изучите способ сортировки подсчетом – применение этого метода сортировки внутри разряда улучшает оценку сложности алгоритма. Попробуйте оценить сложность поразрядных сортировок. Попытайтесь реализовать изученные способы сортировки.

Рекомендуемые источники

Мещеряков, Р.В. Методы программирования [Электронный ресурс] : учебно-методическое пособие / Р.В. Мещеряков. – Электрон. дан. – Москва : ТУСУР, 2007. – 237 с. – Режим доступа: https://e.lanbook.com/book/11631. – Загл. с экрана. Стр. 147 – 151, 153 – 155.

Потопахин, В. Искусство алгоритмизации [Электронный ресурс] / В. Потопахин. – Электрон. дан. – Москва : ДМК Пресс, 2011. – 320 с. – Режим доступа: https://e.lanbook.com/book/1269 . – Загл. с экрана. Стр. 186 – 191.

3.4.3 Динамические структуры

Перечень вопросов, подлежащих изучению

1. Линейные однонаправленные списки.
2. Список – стек, список – очередь.
3. Добавление элементов в список.
4. Удаление элемента из списка.
5. Создание сортированных списков.

Методические рекомендации по изучению

Изучение теоретического материала по этой теме желательно сопровождать зарисовками представления списков на бумаге. Понимание физического смысла создаваемой динамической структуры – основная задача самостоятельной работы над этой темой и успешное освоение темы напрямую зависит от решения этой задачи. При изучении рекомендуемых источников помните, что программирование – творческий процесс, поэтому одна и та же задача в разных пособиях может иметь различные варианты реализации в виде программного кода.

Рекомендуемые источники

Потопахин, В. Искусство алгоритмизации [Электронный ресурс] / В. Потопахин. – Электрон. дан. – Москва : ДМК Пресс, 2011. – 320 с. – Режим доступа: https://e.lanbook.com/book/1269 . – Загл. с экрана. Стр. 224 – 236.

Вирт, Н. Алгоритмы и структуры данных. Новая версия для Оберона [Электронный ресурс] : учебное пособие / Н. Вирт. – Электрон. дан. – Москва : ДМК Пресс, 2010. – 272 с. – Режим доступа: https://e.lanbook.com/book/1261. – Загл. с экрана. Стр. 175 – 190.

3.4.4 Двоичные деревья поиска (BST-деревья)

Перечень вопросов, подлежащих изучению

1. Способы добавления элементов в дерево.
2. Фундаментальные операции над деревьями.
3. Методы печати элементов дерева.
4. Разделение дерева относительно *k*-того наименьшего.
5. Удаление элементов из дерева.

Методические рекомендации по изучению

Изучение этой темы желательно выполнять после знакомства с динамическими структурами данных. Рекомендуется составить план самостоятельной работы, так как это перечислено выше. Для закрепления материала попробуйте выполнить изученные алгоритмы на бумаге, на примере произвольной последовательности данных. После этого можно приступить к программной реализации алгоритмов. Обратите внимание – большинство функций работы с деревьями реализованы рекурсивно. В качестве дополнительного задания попробуйте выполнить не рекурсивную реализацию.

Рекомендуемые источники

Мещеряков, Р.В. Методы программирования [Электронный ресурс] : учебно-методическое пособие / Р.В. Мещеряков. – Электрон. дан. – Москва : ТУСУР, 2007. – 237 с. – Режим доступа: https://e.lanbook.com/book/11631. – Загл. с экрана. Стр. 81 – 108.

Вирт, Н. Алгоритмы и структуры данных. Новая версия для Оберона [Электронный ресурс] : учебное пособие / Н. Вирт. – Электрон. дан. – Москва : ДМК Пресс, 2010. – 272 с. – Режим доступа: https://e.lanbook.com/book/1261. – Загл. с экрана. Стр. 191 – 255.

3.4.5 Поиск

Перечень вопросов, подлежащих изучению

1. Поиск элементов в массиве.
2. Бинарный поиск.
3. Интерполяционный поиск.
4. Поиск подстроки в строке.

Методические рекомендации по изучению

Алгоритмы поиска встречаются в программировании повсеместно. При изучении этой темы студенты должны выделить три способа организации поиска в линейной структуре данных. После изучения всех трех способов сравните оценки сложности рассматриваемых алгоритмов. Вторая часть темы посвящена различным алгоритмам поиска подстроки в строке (в тексте). К таким алгоритмам относятся алгоритм поиска Боуера-Мура, алгоритм Кнута, Морриса и Пратта, прямой алгоритм. Ознакомьтесь с этими алгоритмами, попробуйте выполнить реализацию. Сравните алгоритмы по сложности.

Рекомендуемые источники

Вирт, Н. Алгоритмы и структуры данных. Новая версия для Оберона [Электронный ресурс] : учебное пособие / Н. Вирт. – Электрон. дан. – Москва : ДМК Пресс, 2010. – 272 с. – Режим доступа: https://e.lanbook.com/book/1261. – Загл. с экрана. Стр. 54 – 64.

3.5 Подготовка к экзамену

Студенты дневной формы обучения заканчивают изучение дисциплины сдачей экзамена. Выполнение всех видов самостоятельных работ, лабораторных работ, посещение практических и лекционных занятий – гарантия успешной сдачи экзамена.

Для подготовки к экзамену рекомендуется повторить темы, вынесенные на экзамен. При подготовке обращайтесь не только к конспектам лекций, но и к рекомендованным преподавателем источникам. Организуйте план повторения материала таким образом, чтобы каждый день прорабатывать примерно одинаковый по объему материал. Изучая учебники и учебные пособия, отвечайте на контрольные вопросы. Прорешайте задачи примерного билета. Если после изучения материала Вы не смогли найти ответы на какие-либо вопросы – посетите консультацию перед экзаменом, кроме ответов на вопросы по теме экзамена на консультации освещаются организационные вопросы проведения экзамена время начала экзамена, время проведения экзамена, план проведения экзамена и т.д..

Пример экзаменационного билета

Билет 1

Теоретическая часть

1. Запишите алгоритм сортировки обменом.

2. Продемонстрируйте работу сортировки Шелла на массиве

9 8 0 6 13 7 18 11 1 13 16 13 12 1 4 16 3 10 1 3 с шагом 3

3. Нарисуйте дерево разбиений, которое строит нисходящая сортировка слиянием при обработке массива из 25 элементов.

4. Продемонстрируйте алгоритм MSD сортировки на массиве:

40464
18401
61242
74330
75675
99500
12042
13384
25745
33713

5. Постройте таблицу сдвигов (КМП-алгоритм) для образа «тригонометрия»

6. Постройте дерево методом вставки в лист, если элементы в дерево добавлялись в следующем порядке 13 12 8 18 22 0 16 25 29 13 6 17 13 17 15 16. Продемонстрируйте ротацию влево в узле 18.

Практическая часть

1. В текстовом файле записан массив целых чисел. Считать массив из файла. Найти сумму чисел. В конец исходного файла дописать строку: «Сумма чисел = [найденная сумма]».

2. Написать:

функцию создания двоичного файла, содержащего матрицу целых чисел размерности nxm. Значения n, m задаются с клавиатуры. Элементы матрицы – целые случайные числа;

функцию печати содержимого двоичного файла;

функцию, выводящую на экран четные столбцы матрицы, не считывая при этом матрицу в память.

3. В текстовом файле хранится произвольное количество чисел. Считать данные из файла в однонаправленный динамический список, организованный по правилу очереди (первое число из файла должно оказаться в «голове», последнее в «хвосте» списка). Поменять местами первый и последний элементы списка.

4 РЕКОМЕНДУЕМЫЕ ИСТОЧНИКИ

1. Мещеряков, Р.В. Методы программирования [Электронный ресурс] : учебно-методическое пособие / Р.В. Мещеряков. – Электрон. дан. – Москва : ТУСУР, 2007. – 237 с. – Режим доступа: https://e.lanbook.com/book/11631. – Загл. с экрана.

2. Потопахин, В. Искусство алгоритмизации [Электронный ресурс] / В. Потопахин. – Электрон. дан. – Москва : ДМК Пресс, 2011. – 320 с. – Режим доступа: https://e.lanbook.com/book/1269 . – Загл. с экрана.

3. Вирт, Н. Алгоритмы и структуры данных. Новая версия для Оберона [Электронный ресурс] : учебное пособие / Н. Вирт. – Электрон. дан. – Москва : ДМК Пресс, 2010. – 272 с. – Режим доступа: https://e.lanbook.com/book/1261. – Загл. с экрана.

4. Пермякова, Н.В. Информатика и программирование: Учебное пособие [Электронный ресурс] / Н.В. Пермякова – Томск: ТУСУР, 2016. – 188 с. – Режим доступа: https://edu.tusur.ru/publications/7678

ПРИЛОЖЕНИЕ А

Темы и примерные варианты контрольных работ

1. Характеристики сортировок. Оценка сложности алгоритма

Вариант 1 Фамилия_____	

Является ли сортировка выбором устойчивой? Поясните, почему. (Приведите пример)	Найдите оценку временной сложности фрагмента программы:
	`int i = 2;`
	`int n = …`
	`while(i<=n){`
	`printf("%d ",i);`
	`i+=3;`
	`}`

2. Улучшенные сортировки

Вариант 1
Фамилия_____
Постройте начальную пирамиду на массиве 1 6 2 0 4 5 7 9 3.

3. Сортировка слиянием

Фамилия _____	
1. Продемонстрируйте последовательность шагов при нисходящей сортировке слиянием на массиве 5 2 6 9 7 3 10 8 11 1 4	2. Постройте дерево разбиений , которое строит восходящая сортировка слиянием при упорядочивании массива из 12 элементов.

4. Поразрядные сортировки

Фамилия _____	
1. Продемонстрируйте алгоритм MSD сортировки с R=10 на следующей последовательности: 0.1234 0.1123 0.2345 0.1135 0.2245 0.2346 0.1126	2. Продемонстрируйте алгоритм LSD сортировки для последовательности 0110 1000 0011 1001 0001

5. Поиск подстроки в строке

Фамилия _____	
Метод Кнута, Морриса и Пратта. Рассчитайте таблицу сдвигов для подстроки **ABSDABSDF**	Метод Боуера-Мура. Рассчитайте таблицу сдвигов для подстроки **VDFSDERH**

6. Двоичные деревья

Фамилия _____	
Постройте BST-дерево из исходного массива 6 4 3 8 1 9 2 7 0 5	Перечислите узлы построенного дерева в порядке прямого обхода

7. Двоичные деревья. Ротации

Вариант 1. Фамилия _____
Постройте BST-дерево. Выполните ротацию влево в корне 13 28 24 26 2 25 30 18 18 14 30 31 10 6 30 17

8. Двоичные деревья. Разделение относительно *k*-того наименьшего

Вариант 1. Фамилия _____
Постройте BST-дерево. Разделите дерево относительно 5-го наименьшего элемента. 13 28 24 26 2 25 30 18 18 14 30 31 10 6 30 17