МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ

«БРАТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Кафедра управления в технических системах

УТВЕРЖДАЮ:
Проректор по учебной работе
_____ Е.И. Луковникова
«__» мая 2019 г.

РАБОЧАЯ ПРОГРАММА ДИСЦИПЛИНЫ

СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ

Б1.В.15

НАПРАВЛЕНИЕ ПОДГОТОВКИ

27.03.04 Управление в технических системах

ПРОФИЛЬ ПОДГОТОВКИ

Управление и информатика в технических системах

Программа академического бакалавриата

Квалификация (степень) выпускника: бакалавр

Программа составлена в соответствии с федеральным государственным образовательным стандартом высшего образования по направлению подготовки 27.03.04 Управление в технических системах от 20.10.2015 г № 1171 и учебным планом ФГБОУ ВО «БрГУ» от 01.04.2019 г № 196 для заочной формы обучения набора 2019 года

СОДЕРЖАНИЕ ПРОГРАММЫ **Стр.**

1. ПЕРЕЧЕНЬ ПЛАНИРУЕМЫХ РЕЗУЛЬТАТОВ ОБУЧЕНИЯ ПО ДИСЦИПЛИНЕ, СООТНЕСЕННЫХ С ПЛАНИРУЕМЫМИ РЕЗУЛЬТАТАМИ ОСВОЕНИЯ ОБРАЗОВАТЕЛЬНОЙ ПРОГРАММЫ

Вид деятельности выпускника

Дисциплина охватывает круг вопросов, относящихся к научно-исследовательскому виду профессиональной деятельности выпускника в соответствии с компетенциями и видами деятельности, указанными в учебном плане.

Цель дисциплины

Формирование у обучающихся знаний базовых классов структур данных и алгоритмов их программной обработки; формирование навыков проектирования эффективных структур и алгоритмов обработки данных при решении практических задач.

Задачи дисциплины

Подготовка обучающихся к самостоятельному решению теоретических и прикладных задач связанных с проектированием эффективных структур и алгоритмов обработки данных.

Код компетенции	Содержание компетенций	Перечень планируемых результатов обучения по дисциплине
1	2	3
ОК-7	способность к самоорганизации и самообразованию	**знать:** основные методы разработки алгоритмов и программ, структуры данных, используемые для представления типовых информационных объектов. **уметь:** использовать стандартные пакеты прикладных программ для решения практических задач. **владеть:** навыками самоорганизации и самообразования.
ОПК-6	способность осуществлять поиск, хранение, обработку и анализ информации из различных источников и баз данных, представлять ее в требуемом формате с использованием информационных, компьютерных и сетевых технологий	**знать:** современные типовые алгоритмы обработки данных. **уметь:** представлять информацию в требуемом формате с использованием информационных, компьютерных и сетевых технологий. **владеть:** методами построения современных проблемно-ориентированных прикладных программных средств.
ПК-5	способность осуществлять сбор и анализ исходных данных для расчета и проектирования систем и средств автоматизации и управления	**знать:** современные средства автоматизации и управления **уметь:** анализировать исходные данные для расчета систем. **владеть:** методами сбора информации.

2. МЕСТО ДИСЦИПЛИНЫ В СТРУКТУРЕ ОБРАЗОВАТЕЛЬНОЙ ПРОГРАММЫ

Дисциплина Б1.В.15 Структуры и алгоритмы обработки данных относится к вариативной части.

Дисциплина Структуры и алгоритмы обработки данных базируется на знаниях, полученных при изучении учебных дисциплин Б1.Б.5 Математика, Б1.В.7 Информатика.

Основываясь на изучении перечисленных дисциплин, структуры и алгоритмы обработки данных представляет основу для изучения дисциплин: Б1.Б.13 Вычислительные машины, системы и сети, Б1.В.16 Системы управления базами данных, Б1.В.18 Технологии программирования. Такое системное междисциплинарное изучение направлено на достижение требуемого ФГОС уровня подготовки по квалификации бакалавр.

3. РАСПРЕДЕЛЕНИЕ ОБЪЕМА ДИСЦИПЛИНЫ

3.1. Распределение объема дисциплины по формам обучения

Форма обучения	*Курс*	*Семестр*	*Трудоемкость дисциплины в часах*						*Курсовая работа (проект), контрольная работа, реферат, РГР*	*Вид промежуточной аттестации*
			Всего часов (с экз.)	*Аудиторных часов*	*Лекции*	*Лабораторные работы*	*Практические занятия*	*Самостоятельная работа*		
1	2	3	4	5	6	7	8	9	10	11
Очная	2	3	180	68	34	34	-	112	к	экзамен
Заочная	2	-	180	16	8	8	-	164	к	экзамен
Заочная (ускоренное обучение)	-	-	-	-	-	-	-	-	-	-
Очно-заочная	-	-	-	-	-	-	-	-	-	-

3.2. Распределение объема дисциплины по видам учебных занятий и трудоемкости

Вид учебных занятий	*Трудо-емкость (час.)*	*в т.ч. в интерактивной, активной, иннова-ционной формах, (час.)*	*Распределение по семестрам, час*
			3
1	2	3	4
I. Контактная работа обучающихся с преподавателем (всего)	68	18	68
Лекции (Лк)	34	-	34
Лабораторные работы (ЛР)	34	18	34
Контрольная работа (к)	+	-	+
Групповые (индивидуальные) консультации	+	-	+
II.Самостоятельная работа обучающихся (СР)	58	-	58
Подготовка к лабораторным работам	14	-	14
Подготовка к зачету	18	-	18
Выполнение курсовой работы	13	-	13
Подготовка к экзамену в течении семестра	13	-	13
III. Промежуточная аттестация экзамен	+ 54	-	+ 54
Общая трудоемкость дисциплины	180	-	180
зач. ед.	5	-	5

4. Содержание дисциплины

4.1. Распределение разделов дисциплины по видам учебных занятий - для очной формы обучения:

№ раздела и темы	*Наименование раздела и тема дисциплины*	*Трудоемкость, (час.)*	*Виды учебных занятий, включая самостоятельную работу обучающихся и трудоемкость; (час.)*		
			учебные занятия		*самостоятельная работа обучающихся*
			лекции	*лабораторные работы*	
1	2	3	4	5	6
1.	**Классификация структур данных.**	**10**	**5**	**5**	**-**
1.1.	Массивы, записи, множества, динамические структуры данных.	5	2,5	2,5	-
1.2.	Списки.	5	2,5	2,5	-
2.	**Типы данных линейной структуры.**	**13**	**5**	**5**	**3**
2.1.	Данные элементарных типов.	5	1	1	3
2.2.	Данные числового, вещественного, символьного, логического типов.	4	2	2	-
2.3.	Данные типа указатель.	4	2	2	-
3.	**Алгоритмы сортировки массивов.**	**21**	**5**	**5**	**11**
3.1.	Сортировка массивов методом выбора.	5,25	1,25	1,25	2
3.2.	Сортировка методом обмена.	5,25	1,25	1,25	3
3.3.	Сортировка челночным методом.	5,25	1,25	1,25	3
3.4.	Сортировка методом вставок.	5,25	1,25	1,25	3
4.	**Хеширование.**	**21**	**5**	**5**	**11**
4.1.	Хеш-функции.	12	3	3	6
4.2.	Методы разрешения коллизий.	9	2	2	5
5.	**Файлы.**	**21**	**5**	**5**	**11**
5.1.	Внешние файлы.	10,5	2,5	2,5	5,5
5.2.	Внутренние файлы.	10,5	2,5	2,5	5,5
6.	**Типы данных нелинейной структуры.**	**21**	**5**	**5**	**11**
6.1.	Деревья.	5	1	1	3
6.2.	Бинарные деревья.	8	2	2	4
6.3.	Пирамиды.	8	2	2	4
7.	**Графы.**	**19**	**4**	**4**	**11**
7.1	Основные понятия теории графов.	4	1	-	3
7.2.	Основные процедуры работы с графами.	8	2	2	4
7.3.	Алгоритмы на графах: поиск в глубину и поиск в ширину.	7	1	2	4
	ИТОГО	**126**	**34**	**34**	**58**

для заочной формы обучения:

№ раз-дела и темы	*Наименование раздела и тема дисциплины*	*Трудоем-кость, (час.)*	*Виды учебных занятий, включая самостоятельную работу обучающихся и трудоемкость; (час.)*		
			учебные занятия		*самостоя тельная работа обучаю-щихся*
			лекции	*лабораторные работы*	
1	2	3	4	5	6
1.	**Классификация структур данных.**	**21**	**0,5**	**0,5**	**20**
1.1.	Массивы, записи, множества, динамические структуры данных.	10,25	0,25	-	10
1.2.	Списки.	10,75	0,25	0,5	10
2.	**Типы данных линейной структуры.**	**30**	**1,5**	**1,5**	**27**
2.1.	Данные элементарных типов.	10	0,5	0,5	9
2.2.	Данные числового, вещественного, символьного, логического типов.	10	0,5	0,5	9
2.3.	Данные типа указатель.	10	0,5	0,5	9
3.	**Алгоритмы сортировки массивов.**	**30**	**1**	**1**	**28**
3.1.	Сортировка массивов методом выбора.	7,5	0,25	0,25	7
3.2.	Сортировка методом обмена.	7,5	0,25	0,25	7
3.3.	Сортировка челночным методом.	7,5	0,25	0,25	7
3.4.	Сортировка методом вставок.	7,5	0,25	0,25	7
4.	**Хеширование.**	**28**	**1**	**1**	**26**
4.1.	Хеш-функции.	14	0,5	0,5	13
4.2.	Методы разрешения коллизий.	14	0,5	0,5	13
5.	**Файлы.**	**28**	**1**	**1**	**26**
5.1.	Внешние файлы.	14	0,5	0,5	13
5.2.	Внутренние файлы.	14	0,5	0,5	13
6.	**Типы данных нелинейной структуры.**	**30**	**1,5**	**1,5**	**27**
6.1.	Деревья.	10	0,5	0,5	9
6.2.	Бинарные деревья.	10	0,5	0,5	9
6.3.	Пирамиды.	10,5	0,5	0,5	9
7.	**Графы.**	**4**	**1,5**	**1,5**	**1**
7.1	Основные понятия теории графов.	1	0,5	0,5	-
7.2.	Основные процедуры работы с графами.	1	0,5	0,5	-
7.3.	Алгоритмы на графах: поиск в глубину и поиск в ширину.	2	0,5	0,5	1
	ИТОГО	**171**	**8**	**8**	**155**

4.2. Содержание дисциплины, структурированное по разделам и темам

Раздел 1. Классификация структур данных.

Программируя решение любой задачи, необходимо выбрать уровень абстрагирования. Иными словами, определить множество данных, представляющих предметную область решаемой задачи. При выборе следует руководствоваться проблематикой решаемой задачи и способами представления информации. Здесь необходимо ориентироваться на те средства, которые предоставляют системы программирования и вычислительная техника, на которой будут выполняться программы. Во многих случаях эти критерии не являются полностью независимыми.

Вопросы представления данных часто разбиваются на различные уровни детализации. Уровню языков программирования соответствуют абстрактные типы и структуры данных. Рассмотрим их реализацию в языке программирования Turbo-Pascal. Простейшим типом данных является переменная. Существуют несколько встроенных типов данных. Например, описание

```
Var
i, j : integer;
x : real;
s: string;
```

объявляет переменные i, j целочисленного типа, x - вещественного и s - строкового.

Переменной можно присвоить значение, соответствующее ее типу

```
I:=46;
X:=3.14;
S:="строка символов";
```

Такие переменные представляют собой лишь отдельные элементы. Для того чтобы можно было говорить о структурах данных, необходимо иметь некоторую агрегацию переменных. Примером такой агрегации является массив.

Тема 1.1 Массивы, записи, множества, динамические структуры данных.

Массивы.

Массив объединяет элементы одного типа данных. Более формально его можно определить как упорядоченную совокупность элементов некоторого типа, адресуемых при помощи одного или нескольких индексов. Частным случаем является одномерный массив

```
Var
l : array [1..100] of integer;
```

В приведенном примере описан массив l, состоящий из элементов целочисленного типа. Элементы могут быть адресованы при помощи индекса в диапазоне значений от 1 до 100. В математически х расчетах такой массив соответствует вектору. Массив не обязательно должен быть одномерным. Можно описать в виде массива матрицу 100*100

```
Var
M : array [1..100,1..100] of real;
```

В этом случае можно говорить уже о двумерном массиве. Аналогичным образом можно описать массивс большим числом измерений, например трехмерный

```
Var
M_3_d : array [0..10,0..10,0..10] of real;
```

Теперь можно говорить уже о многомерном массиве. Доступ к элементам любого массиваосуществляется при помощи индексов как к обычной переменной.

```
M_3_d [0,0,10]:=0.25;
M[10,30]:=m_3_d[0,0,10]+0.5;
L[i]:=300;
```

Записи.

Более сложным типом является запись. Основное отличие записи заключается в том, что она может объединять элементы данных разных типов.

Рассмотрим пример простейшей записи

```
Type
Person = record
Name: string;
Address: string;
Index: longint;
end;
```

Запись описанного типа объединяет четыре поля. Первые три из них символьного типа, а четвертое - целочисленного. Приведенная конструкция описывает тип записи. Для того чтобы использовать данные описанного типа, необходимо описать сами данные. Один из вариантов использования отдельных записей - объединение их в массив, тогда описание массива будет выглядеть следующим образом

```
Var
```

Persons : array[1..30] of person;

Следует заметить, что в Turbo-pascal эти два описания можно объединить в виде описания так называемого массива записей

```
Var
Persons : array[1..30] of record
Name: string;
Address: string;
Index: longint;
end;
```

Доступ к полям отдельной записи осуществляется через имя переменной и имя поля.

```
Persons[1] . Name:="Иванов";
Persons[1] . Adress:='город Санкт-Петербург ┘ ";
Persons[2] . Name:="Петров";
Persons[2] . Adress:='город Москва ┘ ";
```

Разумеется, что запись можно использовать в качестве отдельной переменной, для этого соответствующая переменная должна иметь тип, который присвоен описанию записи

```
Type
Person = record
Name: string;
Address: string;
Index: Longint;
end;
Var
Person1: person;
Begin
Person1.index:=190000;
```

Множества.

Наряду с массивами и записями существует еще один структурированный тип - множество. Этот тип используется не так часто, хотя его применение в некоторых случаях является вполне оправданным.

Тип множество соответствует математическому понятию множества в смысле операций, которые допускаются над структурами такого типа. Множество допускает операции объединения множеств (+),пересечения множеств (*), разности множеств (-) и проверки элемента на принадлежность к множеству (in). Множества также как и массивы объединяют однотипные элементы. Поэтому в описании множества обязательно должен быть указан тип его элементов.

```
Var
RGB, YIQ, CMY : Set of string;
```

Здесь мы привели описание двух множеств, элементами которых являются строки. В отличие от массивов и записей здесь отсутствует возможность индексирования отдельных элементов.

```
CMY:= ["M ","C ","Y "];
RGB:= ["R","G","B"];
YIQ:=[ "Y ","Q ","I "];
Writeln ("Пересечение цветовых систем RGB и CMY ", RGB*CMY);
Writeln ("Пересечение цветовых систем YIQ и CMY ",YIQ*CMY);
```

Операции выполняются по отношению ко всей совокупности элементов множества. Можно лишь добавить, исключить или выбрать элементы, выполняя допустимые операции.

Динамические структуры данных.

Мы ввели базовые структуры данных: массивы, записи, множества. Мы назвали их базовыми, поскольку из них можно образовывать более сложные структуры. Цель описания типа данных и определения некоторых переменных как относящихся к этому типу состоит в том, чтобы зафиксировать диапазон значений, присваиваемых этим переменным, и соответственно размер выделяемой для них памяти. Поэтому такие переменные называются статическими. Однако существует возможность создавать более сложные структуры данных. Для них характерно, что в процессе обработки данных изменяются не только значения переменных, но и сама их структура .Соответственно динамически изменяется и размер памяти, занимаемый такими структурами. Поэтому такие данные получили название данных с динамической структурой.

Тема 1.2 Списки.

Линейные списки.

Наиболее простой способ связать некоторое множество элементов - это организовать линейный список. При такой организации элементы некоторого типа образуют цепочку. Для связывания элементов в списке используют систему указателей. В рассматриваемом случае любой элемент линейного списка имеет один указатель, который указывает на следующий элемент в списке или является пустым указателем, что означает конец списка.

В языке Turbo Pascal предусмотрены два типа указателей - типизированные и не типизированные указатели. В случае линейного списка описание данных может выглядеть следующим образом.

```
type
```

```
element = record
data:string;
next: pointer;
end;
var
head: pointer;
current, last : ^element;
```

В данном примере элемент списка описан как запись, содержащая два поля. Поле data строкового типа служит для размещения данных в элементе списка. Другое поле next представляет собой не типизированный указатель, который служит для организации списковой структуры.

В описании переменных описаны три указателя head, last и current. Head является не типизированным указателем. Указатель current является типизированным указателем, что позволяет через него организовать доступ к полям внутри элемента, имеющего тип element. Для объявления типизированного указателя обычно используется символ ^, размещаемый непосредственно перед соответствующим типом данных. Однако описание типизированного указателя еще не означает создание элемента списка. Рассмотрим, как можно осуществить создание элементов и их объединение в список.

В системе программирования Turbo Pascal для размещения динамических переменных используется специальная область памяти "heap-область". Heap-область размещается в памяти компьютера следом за областью памяти, которую занимает программа и статические данные, и может рассматриваться как сплошной массив, состоящий из байтов.

Попробуем создать первый элемент списка. Это можно осуществить при помощи процедуры New

```
New(Current);
```

После выполнения данной процедуры в динамической области памяти создается динамическая переменная, тип которой определяется типом указателя. Сам указатель можно рассматривать как адрес переменной в динамической памяти. Доступ к переменной может быть осуществлен через указатель. Заполним поля элемента списка.

```
Current^.data:= "данные в первом элементе списка " ;
Current^.next:=nil;
```

Значение указателя nil означает пустой указатель. Обратим внимание на разницу между присвоением значения указателю и данным, на которые он указывает. Для указателей допустимы только операции присваивания и сравнения. Указателю можно присваивать значение указателя того же типа или константу nil. Данным можно присвоить значения, соответствующие декларируемым типам данных. Для того чтобы присвоить значение данным, после указателя используется символ ^. Поле Current^.next само является указателем, доступ к его содержимому осуществляется через указатель Current.

В результате выполнения описанных действий мы получили список из одного элемента. Создадим еще один элемент и свяжем его с первым элементом.

```
Head:=Current;
New(last);
Last^.data:= "данные во втором элементе списка " ;
Last^.next:=nil;
Current^.next:=nil;
```

Непосредственно перед созданием первого элемента мы присвоили указателю Head значение указателя Current. Это связано с тем, что линейный список должен иметь заголовок. Другими словами, первый элемент списка должен быть отмечен указателем. В противном случае, если значение указателя Current в дальнейшем будет переопределено, то мы навсегда потеряем возможность доступа к данным, хранящимся в первом элементе списка.

Динамическая структура данных предусматривает не только добавление элементов в список, но и их удаление по мере надобности. Самым простым способом удаления элемента из списка является переопределение указателей, связанных с данным элементом (указывающих на него). Однако сам элемент данных при этом продолжает занимать память, хотя доступ к нему будет навсегда утерян. Для корректной работы с динамическими структурами следует освобождать память при удалении элемента структуры. В языке TurboPascal для этого можно использовать функцию Dispose. Покажем, как следует корректно удалить первый элемент нашего списка.

```
Head:=last;
Dispose(current);
```

Рассмотрим пример более сложной организации списка. Линейный список неудобен тем, что при попытке вставить некоторый элемент перед текущим элементом требуется обойти почти весь список, начиная с заголовка, чтобы изменить значение указателя в предыдущем элементе списка. Чтобы устранить данный недостаток введем второй указатель в каждом элементе списка. Первый указатель связывает данный элемент со следующим, а второй - с предыдущим. Такая организация динамической структуры данных получила название линейного двунаправленного списка (двусвязного списка).

Интересным свойством такого списка является то, что для доступа к его элементам вовсе необязательно хранить указатель на первый элемент. Достаточно иметь указатель на любой элемент списка. Первый элемент всегда можно найти по цепочке указателей на предыдущие элементы, а последний - по цепочке указателей на следующие. Но наличие указателя на заголовок списка в ряде случаев ускоряет работу со списком.

Наличие указателя на заголовок списка ускоряет процесс поиска, так как не требуется сначала найти первый элемент, а затем - сделать обход списка.

Циклические списки

Линейные списки характерны тем, что в них можно выделить первый и последний элементы, причем для однонаправленного линейного списка обязательно нужно иметь указатель на первый элемент.

Циклические списки также как и линейные бывают однонаправленными и двунаправленными. Основное отличие циклического списка состоит в том, что в списке нет пустых указателей.

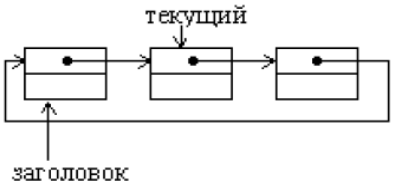


Рис. Однонаправленный циклический список

Последний элемент списка содержит указатель, связывающий его с первым элементом. Для полного обхода такого списка достаточно иметь указатель только на текущий элемент.

В двунаправленном циклическом списке система указателей аналогична системе указателей двунаправленного линейного списка.

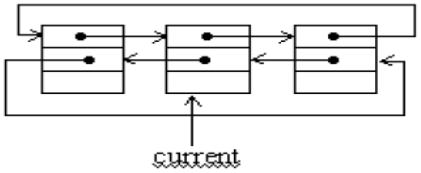


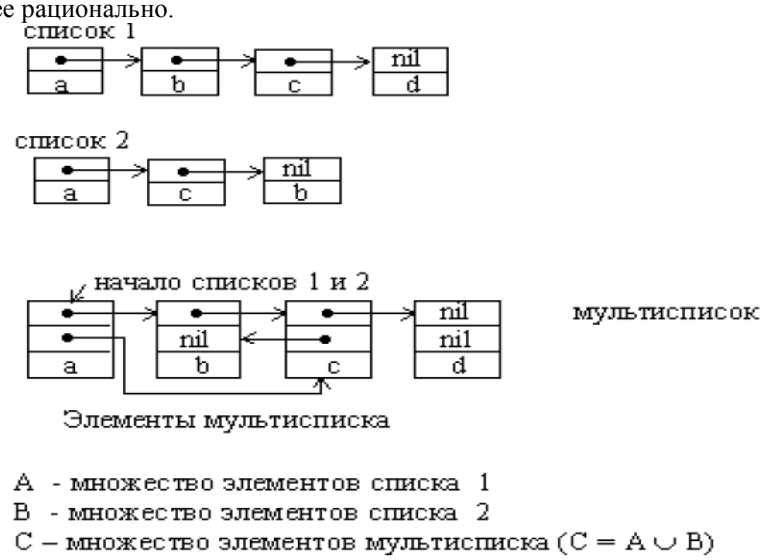
Рис. Двунаправленный циклический список

Двунаправленный циклический список позволяет достаточно просто осуществлять вставки и удаления элементов слева и справа от текущего элемента. В отличие от линейного списка, элементы являются равноправными и для выделения первого элемента необходимо иметь указатель на заголовок. Однако во многих случаях нет необходимости выделять первый элемент списка и достаточно иметь указатель на текущий элемент. В данном примере указатель на первый элемент списка отсутствует. Для предотвращения зацикливания при обходе списка во время поиска указатель на текущий элемент предварительно копируется и служит ограничителем.

Мультисписки.

Иногда возникают ситуации, когда имеется несколько разных списков, которые включают в свой состав одинаковые элементы. В таком случае при использовании традиционных списков происходит многократное дублирование динамических переменных и нерациональное использование памяти. Списки фактически используются не для хранения элементов данных, а для организации их в различные структуры. Использование мультисписков позволяет упростить эту задачу.

Мультисписок состоит из элементов, содержащих такое число указателей, которое позволяет организовать их одновременно в виде нескольких различных списков. За счет отсутствия дублирования данных память используется более рационально.



A - множество элементов списка 1
B - множество элементов списка 2
C – множество элементов мультисписка ($C = A \cup B$)

Рис. Объединение двух линейных списков в один мультисписок.

Экономия памяти - далеко не единственная причина, по которой применяют мультисписки. Многие реальные структуры данных не сводятся к типовым структурам, а представляют собой некоторую комбинацию из них. Причем комбинируются в мультисписках самые различные списки - линейные и циклические, односвязные и двунаправленные.

Раздел 2. Типы данных линейной структуры.

Тема 2.1 Данные элементарных типов.

Данные элементарных типов представляют собой единое и неделимое целое. В каждый момент времени они могут принимать только одно значение. Набор элементарных типов в разных языках программирования несколько различаются, однако есть типы, которые поддерживаются практически везде. Рассмотрим их на примере языка Паскаль:

```
var
  i, j: integer;
  x: real;
  s: char;
  b: boolean;
  p: pointer;
```

Здесь объявлены переменные i, j целочисленного типа, x – вещественного, s – символьного, b – логического типа и p – указатель.

К данным элементарных типов можно обращаться по их именам:

```
i := 46;
x := 3.14;
s := 'A';
b := true;
```

Тема 2.2 Данные числового, вещественного, символьного, логического типов.

Данные числовых типов.

С помощью целых чисел может быть представлено количество объектов, являющихся дискретными по своей природе (т. е. счетное число объектов).

Диапазон возможных значений целых типов зависит от их внутреннего представления, которое может занимать 1, 2 или 4 байта. В таблице приводится перечень целых типов, размер памяти для их внутреннего представления в битах, диапазон возможных значений.

Тип	Диапазон значений	Машинное представление
shortint	–128...127	8 бит со знаком
integer	–32768...32767	16 бит со знаком
longint	–2147483648...2147483647	32 бита со знаком
byte	0...255	8 бит без знака
word	0...65535	16 бит без знака
comp	$-2^{63}+1...2^{63}-1$	64 бита со знаком

Операции над данными числовых типов

Над числовыми типами, как и над всеми другими возможны, прежде всего, четыре основных операции: создание, уничтожение, выбор, обновление.

Специфическими операциями над числовыми типами являются арифметические операции: сложение, вычитание, умножение, деление. Операция возведения в степень в некоторых языках также является базовой и обозначается специальным символом или комбинацией символов, в других – выполняется встроенными функциями.

Следует обратить внимание на то, что операция деления по-разному выполняется для целых и вещественных чисел. При делении целых чисел дробная часть результата отбрасывается, как бы близка к 1 она ни была. В связи с этим в языке Паскаль имеются даже разные обозначения для деления вещественных и целых чисел – операции «/» и «div» соответственно.

В других языках оба вида деления обозначаются одинаково, а тип деления определяется типом операндов. Для целых операндов возможна еще однаоперация – остаток от деления (в языке Паскаль операция «mod»).

Еще одна группа операций над числовыми типами – операции сравнения >, <, ≥, ≤, =, <>. Существенно, что хотя операндами этих операций являются данные числовых типов, результат их имеет логический тип – «истина» или «ложь». Говоря об операциях сравнения, следует обратить внимание на особенность выполнения сравнений на равенство/неравенство вещественных чисел. Поскольку эти числа представляются в памяти с некоторой (не абсолютной) точностью, сравнения их не всегда могут быть абсолютно достоверны.

Поскольку одни и те же операции допустимы для разных числовых типов, возникает проблема арифметических выражений со смешением типов. В реальных задачах выражения со смешанными типами встречаются довольно часто. Поэтому большинство языков допускает выражения, операнды которых имеют разные числовые типы, но обрабатываются такие выражения в разных языках по-разному. В одних языках все операнды выражения приводятся к одному типу, а именно к типу той переменной, в которую будет записан

результат, а затем уже выражение вычисляется. В других (например, язык Си) преобразование типов выполняется в процессе вычисления выражения, при выполнении каждой отдельной операции, без учета других операций; каждая операция вычисляется с точностью самого точного участвующего в ней операнда.

Данные вещественного типа

Данные вещественного типа в отличие от целочисленных типов, значения которых всегда представляются в памяти ЭВМ абсолютно точно, значение вещественных типов определяет число лишь с некоторой конечной точностью, зависящей от внутреннего формата вещественного числа.

Суммарное количество байтов, диапазоны допустимых значений чисел вещественных типов, а также количество значащих цифр после запятой в представлении чисел приведены в таблице.

Тип	Диапазон значений	Значащие цифры	Размер в байтах
real	$2{,}9\cdot10^{-39}\ldots1{,}7\cdot10^{38}$	11–12	6
single	$1{,}4\cdot10^{-45}\ldots3{,}4\cdot10^{38}$	7–8	4
double	$4{,}9\cdot10^{-324}\ldots1{,}8\cdot10^{308}$	15–16	8
extended	$3{,}1\cdot10^{-4944}\ldots1{,}2\cdot10^{4932}$	19–20	10

Данные символьного типа

Значением символьного типа char являются символы из некоторого предопределенного множества. В качестве примеров этих множеств можно назвать ASCII (American Standard Code for Information Interchange). Это множество состоит из 256 разных символов, упорядоченных определенным образом, и содержит символы заглавных и строчных букв, цифр идругих символов, включая специальные управляющие символы.

Значение символьного типа char занимает в памяти 1 байт. Код от 0 до 255 в этом байте задает один из 256 возможных символов ASCII таблицы.

ASCII включает в себя буквенные символы только латинского алфавита. Символы национальных алфавитов занимают «свободные места» в таблице кодов и, таким образом, одна таблица может поддерживать только один национальный алфавит. Этот недостаток преодолен во множестве UNICODE. В этом множестве каждый символ кодируется двумя байтами, что обеспечивает более 216 возможных кодовых комбинаций и дает возможность иметь единую таблицу кодов, включающую в себя все национальные алфавиты. UNICODE, безусловно, является перспективным, однако, повсеместный переход к двухбайтным кодам символов может вызвать необходимость переделки значительной части существующего программного обеспечения.

Специфические операции над символьными типами – только операции сравнения. При сравнении коды символов рассматриваются как целые числа без знака. Кодовые таблицы строятся так, что результаты сравнения подчиняются лексикографическим правилам: символы, занимающие в алфавите места с меньшими номерами, имеют меньшие коды, чем символы, занимающие места с большими номерами.

Данные логического типа

Значениями логического типа может быть одна из предварительно объявленных констант false (ложь) или true (истина). Данные логического типа занимают один байт памяти. При этом значению false соответствует нулевое значение байта, а значению true –любое ненулевое значение байта.

Над логическими типами возможны операции булевой алгебры – НЕ (not), ИЛИ (or), И (and), ИСКЛЮЧАЮЩЕЕ ИЛИ (xor). Последняя операция реализована для логического типа не во всех языках. Кроме того, следует помнить, что результаты логического типа получаются при сравнении данных любых типов.

Интересно, что в языке Си данные логического типа отсутствуют, их функции выполняют данные числовых типов, чаще всего типа int. В логических выражениях операнд любого числового типа, имеющий нулевое значение, рассматривается как «ложь», а ненулевое – как «истина». Результатами выражений логического типа являются целые числа 0 (ложь) или 1 (истина).

Тема 2.3 Данные типа указатель.

Тип указателя представляет собой адрес ячейки памяти. Физическое представление адреса существенно зависит от аппаратной архитектуры вычислительной системы.

В языках программирования высокого уровня определена специальная константа nil, которая означает пустой указатель или указатель, не содержащий какой-либо конкретный адрес.

При решении прикладных задач с использованием языков высокого уровня наиболее частые случаи, когда могут понадобиться указатели, следующие:

1) при необходимости представить одну и ту же область памяти, а следовательно, одни и те же физические данные как данные разной логической структуры. В этом случае вводятся два или более указателей, которые содержат адрес одной и той же области памяти, но имеют разный тип. Обращаясь к этой области памяти по тому или иному указателю, можно обрабатывать ее содержимое как данные того или иного типа;

2) при работе с динамическими структурами данных. Память под такие структуры выделяется в ходе выполнения программы, стандартные процедуры/функции выделения памяти возвращают адрес выделенной области памяти – указатель на нее. К содержимому динамически выделенной области памяти можно обращаться только через такой указатель.

В языках высокого уровня указатели могут быть типизированными и нетипизированными. При объявлении типизированного указателя определяется и тип данного в памяти, адресуемого этим указателем. Приведем пример объявления в языке Паскаль различных типизированных указателей:

```
var
  ipt: ^integer;
  cpt: ^char;
```

Здесь переменная ipt содержит адрес области памяти, в которой хранится целое число, а cpt – адрес области памяти, в которой хранится символ. Хотя физическая структура адреса не зависит от типа и значения данных, хранящихся по этому адресу, считается, что указатели ipt и cpt имеют разный тип.

Нетипизированный указатель служит для представления адреса, по которому содержатся данные неизвестного типа. В Паскале это тип pointer. Работа с нетипизированными указателями существенно ограничена, они могут использоваться только для сохранения адреса, а обращение по адресу, задаваемому нетипизированным указателем, невозможно.

Основными операциями, в которых участвуют указатели, являются присваивание, получение адреса, выборка.

Присваивание является двухместной операцией, оба операнда которой указатели. Как и для других типов, операция присваивания копирует значение одного указателя в другой, в результате оба указателя

будут содержать один и тот же адрес памяти. Если оба указателя, участвующие в операции присваивания типизированные, то оба они должны указывать на данные одного и того же типа.

Операция получения адреса – одноместная, ее операнд может иметь любой тип. Результатом является типизированный (в соответствии стипом операнда) указатель, содержащий адрес операнда.

Операция выборки – одноместная, ее операндом является типизированный указатель. Результатом этой операции являются данные, выбранные из памяти по адресу, заданному операндом. Тип результата определяется типом указателя.

Перечисленных операций достаточно для решения задач прикладного программирования, поэтому набор операций над указателями, допустимых в языке Паскаль, этим и ограничивается. Системное программирование требует более гибкой работы с адресами, поэтому, например, в языке Си доступны также операции адресной арифметики.

Раздел 3. Алгоритмы сортировки массивов.

Задача сортировки является такой же базовой, как задача поиска. В практических условиях эти задачи взаимосвязаны. Решению проблем, связанных с сортировкой, посвящено множество фундаментальных научных исследований, разработано множество алгоритмов.

В общем случае сортировку следует понимать как процесс перегруппировки заданного множества объектов в определенном порядке. Часто при сортировке больших объемов данных нецелесообразно переставлять сами элементы, поэтому для решения задачи выполняется упорядочивание элементов по индексам. То есть индексы элементов выстраивают в такой последовательности, что соответствующие им значения элементов оказываются отсортированными по условию задачи.

Сортировка применяется для облегчения поиска элементов в упорядоченном множестве. Задача сортировки одна из фундаментных в программировании.

Сортировка – это упорядочивание набора однотипных данных по возрастанию или убыванию.

Тема 3.1 Сортировка массивов методом выбора.

Это наиболее естественный алгоритм упорядочивания. При данной сортировке из массива выбирается элемент с наименьшим значением и обменивается с первым элементом. Затем из оставшихся n - 1 элементов снова выбирается элемент с наименьшим ключом и обменивается со вторым элементом, и т.д. (рис.)

Шаги алгоритма:

- находим минимальное значение в текущей части массива;
- производим обмен этого значения со значением на первой неотсортированной позиции;
- далее сортируем хвост массива, исключив из рассмотрения уже отсортированные элементы.

Исходная последовательность $N=6$:

1 7 5 6 4 2

max =6 из *a[0]…a[N-1]*

1 2 5 6 4 7

max =5 из *a[0]…a[N-2]*

1 2 5 4 6 7

max =4 из *a[0]…a[N-3]*

1 2 4 5 6 7

max =3 из *a[0]…a[N-4]*

1 2 4 5 6 7

max =2 из *a[0]…a[N-5]*

1 2 4 5 6 7

max =1 из *a[0]…a[1]*

1 2 4 5 6 7

```
//Описание функции сортировки методом простого выбора
void SelectionSort (int k,int x[max]) {
  int i,j,min,temp;
  for (i=0;i<k-1;i++) {
  //устанавливаем начальное значение минимального индекса
    min=i;
    //находим минимальный индекс элемента
    for (j=i+1;j<k;j++){
      if (x[j]<x[min])
        min=j;
      //меняем значения местами
      }
      temp=x[i];
      x[i]=x[min];
      x[min]=temp;
  }
}
```

Как и в пузырьковой сортировке, внешний цикл выполняется $n-1$ раз, а внутренний – в среднем $n/2$ раз. Следовательно, сортировка методом простого выбора требует

$$\frac{(n^2-n)}{2}$$

сравнений.

Таким образом, это алгоритм порядка n^2, из-за чего он считается слишком медленным для сортировки большого количества элементов. Несмотря на то, что количество сравнений в пузырьковой сортировке и сортировке простым выбором одинаковое, в последней количество обменов в среднем случае намного меньше, чем в пузырьковой сортировке.

Тема 3.2 Сортировка методом обмена (“пузырька”).

Самый известный алгоритм – *пузырьковая сортировка* (bubble sort, сортировка методом пузырька или просто сортировка пузырьком). Его популярность объясняется интересным названием и простотой самого алгоритма.

Алгоритм попарного сравнения элементов массива в литературе часто называют «методом пузырька», проводя аналогию с пузырьком, поднимающимся со дна бокала с газированной водой. По мере всплывания пузырек сталкивается с другими пузырьками и, сливаясь с ними, увеличивается в объеме. Чтобы аналогия стала очевидной, нужно считать, что элементы массива расположены вертикально друг над другом, и их нужно так упорядочить, чтобы они увеличивались сверху вниз.

Алгоритм состоит в повторяющихся проходах по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов. Проходы по массиву повторяются до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает – массив отсортирован. При проходе алгоритма элемент, стоящий не на своём месте, «всплывает» до нужной позиции (рис.).

Исходная последовательность *N=5*:	**6**	**3**	**7**	**1**	**5**
Шаг 1:	**6**	**3**	**7**	**1**	**5**
	3	**6**	**7**	**1**	**5**
	3	**6**	**7**	**1**	**5**
	3	**6**	**1**	**7**	**5**
	3	**6**	**1**	**5**	**7**
Шаг 2:	**3**	**6**	**1**	**5**	7
	3	**6**	**1**	**5**	7
	3	**1**	**6**	**5**	7
	3	**1**	**5**	**6**	7
Шаг 3:	**3**	**1**	**5**	6	7
	1	**3**	**5**	6	7
	1	**3**	**5**	6	7
Шаг 4:	**1**	**3**	5	6	7
	1	**3**	5	6	7
Итоговая последовательность:	1	3	5	6	7

```
//Описание функции сортировки методом "пузырька"
void BubbleSort (int k,int x[max]) {
  int i,j,buf;
```

```
  for (i=k-1;i>0;i--)
    for (j=0;j<i;j++)
    if (x[j]>x[j+1]) {
      buf=x[j];
      x[j]=x[j+1];
      x[j+1]=buf;
     }
}
```

В пузырьковой сортировке количество сравнений всегда одно и то же, поскольку два цикла `for` повторяются указанное количество раз независимо от того, был список изначально упорядочен или нет. Это значит, что алгоритм пузырьковой сортировки всегда выполняет

$$\frac{(n^2 - n)}{2}$$

сравнений, где *n* – количество сортируемых элементов. Данная формула выведена на том основании, что внешний цикл выполняется $n-1$ раз, а внутренний выполняется в среднем $n/2$ раз.

Пузырьковая сортировка имеет такую особенность: неупорядоченные элементы на «большом» конце массива занимают правильные положения за один проход, но неупорядоченные элементы в начале массива поднимаются на свои места очень медленно. Поэтому, вместо того чтобы постоянно просматривать массив в одном направлении, в последовательных проходах можно чередовать направления. Таким образом, элементы, сильно удаленные от своих положений, быстро станут на свои места. Данная версия пузырьковой сортировки носит название ***шейкер-сортировки*** (shaker sort сортировка перемешиванием, сортировка взбалтыванием, сортировка встряхиванием), поскольку действия, производимые ею с массивом, напоминают взбалтывание или встряхивание. Ниже показана реализация шейкер-сортировки.

```
//Описание функции шейкер-сортировки
void Shaker(int k,int x[max]){
  int i,t;
  bool exchange;
  do {
    exchange = false;
    for(i=k-1; i > 0; --i) {
      if(x[i-1] > x[i]) {
        t = x[i-1];
        x[i-1] = x[i];
        x[i] = t;
        exchange = true;
      }
    }
    for(i=1; i < k; ++i) {
      if(x[i-1] > x[i]) {
        t = x[i-1];
        x[i-1] = x[i];
        x[i] = t;
        exchange = true;
      }
    }
  } while(exchange);
    //сортировать до тех пор, пока не будет обменов
}
```

Хотя шейкер-сортировка и является улучшенным вариантом по сравнению с пузырьковой сортировкой, она по-прежнему имеет время выполнения порядка n^2. Это объясняется тем, что количество сравнений не изменилось, а количество обменов уменьшилось лишь на относительно небольшую величину.

Тема 3.3 Сортировка челночным методом.

При сортировке массива *a[0], a[1], ..., a[N-1]* челночным методом, начиная с первого элемента сравнивают два соседних элемента (*a[i]* и *a[i+1]*), если порядок не нарушен (*a[i]<=a[i+1]*), то продвигаются на один элемент вперед (*i=i+1*), если порядок нарушен *(a[i]>a[i+1]),* то производится перестановка и сдвигаются на один элемент назад *(i=i-1)*. Таким образом сравнивая и переставляя элементы доходят до конца массива *(i=N-1)*, при этом массив становится упорядоченным. Работа алгоритма иллюстрируется следующим примером:

Исходная последовательность *N=5*

i=0 (порядок нарушен)

i=0 (порядок не нарушен)

i=1 (порядок не нарушен)

i=2 (порядок нарушен)

i=1 (порядок не нарушен)

i=2 (порядок не нарушен)
i=3 (порядок нарушен)
i=2 (порядок нарушен)
i=1 (порядок нарушен)
i=0 (порядок не нарушен)
i=1 (порядок не нарушен)
i=2 (порядок не нарушен)
i=3 (порядок не нарушен)
i=4: Итоговая последовательность

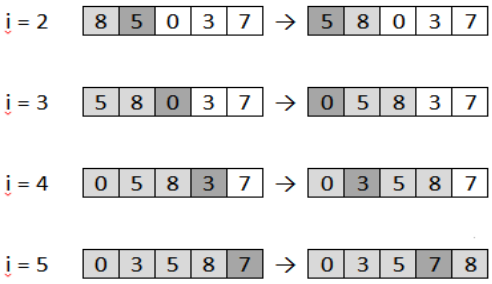
Программная реализация этого метода на языке Си, будет выглядеть следующей:

```
#include<conio.h>
#include<iostream.h>
#include<stdlib.h>
main()
{
const N=5;
int A[N];
int i;
for (i=0; i<N; i++);
cin>>A[i]; //ввод значений массива
int swap;
i=0;
while (i<N-1)
  if (A[i]>A[i+1]) {swap=A[i+1];
                    A[i+1]=A[i];
                    A[i]=swap;
                    i--;
                    if (i<0) i=0;
                    }
     else i++;
cout <<A[i]<<"" "";
getch();
                     }
}
```

Если в условии if (**A[j]>A[j+1]**), поменять знак операции сравнения на меньше (**<**), то последними элементами в подмассивах будут оказываться наименьшие значения и сортировка массива будет осуществляться по убыванию.

Тема 3.4 Сортировка методом вставок.

Данный метод сортировки массива основан на вставке элементов из неупорядоченной части массива в упорядоченную. Считается, что первоначально массив *a[0], a[1], ..., a[N-1]* является не отсортированным, поэтому упорядоченная часть будет состоять из одного первого элемента *a[0]*, а неупорядоченная из всех элементов массива *a[1], ..., a[N-1]*. На каждом шаге метода из неупорядоченной части извлекается первый элемент и помещается в упорядоченную, так чтобы не нарушался порядок. При этом количество элементов в упорядоченной части увеличивается на 1, а в неупорядоченной уменьшается на 1. Процесс вставки элементов происходит до тех пор, пока весь массив не окажется отсортированным.



Программная реализация этого метода на языке Си++, будет выглядеть следующей:

```
#include<conio.h>
#include<iostream.h>
#include<stdlib.h>
main()
{
const N=6;
int A[N];
int i,j,element;
```

```
for (i=0; i<N; i++)
for (i=1; i<N; i++)
cin>>A[i]; //ввод значений массива
    { element=A[i];
      j=i;
      while (A[j-1]>element)
         {A[j]=A[j-1];
          j--;
          if (j<1) break;
          }
      A[j]=element;
    }
for (i=0; i<N; i++) cout << A[i]<< " ";
cout <<A[i]<<"" "";
getch();
  }
}
```

Раздел 4. Хеширование.

Тема 4.1 Хеш-функции.

В качестве хеш-функции h(k) выбирается функция, которая более равномерно рассеивает ключи по пространству таблицы. Это важно для уменьшения числа коллизий. Для рассеянных таблиц характерно заранее назначать объем памяти, которая отводится под таблицу. Предположим, что хеш-функция имеет более M различных значений $0 \le h(k) < M$. Хорошая хеш-функция должна удовлетворять двум требованиям: ее вычисление должно быть быстрым, а число коллизий – минимальным. Экспериментально была выявлена хорошая работа двух типов хеш-функций. Один из них основан на делении, другой на умножении.

Метод деления использует остаток от деления на M: $h(k) = k \bmod M$. Очевидно, что некоторые значения M лучше других. Например, если M четно, то значение h(k) будет четным при четном k и нечетным при нечетном k. Если в некотором файле все ключи четны, то рассеяние будет плохим и память будет использоваться только на 50%.

Можно предложить следующий способ определения функции хеширования: $h(k) = (a \cdot k + b) \bmod M$, где a, b – некоторые константы. Это псевдослучайное преобразование ключа в адрес.

Пример:

a = 17; b = 23; M = 100.

h(42) = 37.

Мультипликативную схему хеширования также легко реализовать. Пусть число A представлено в формате с фиксированной точкой и эта точка установлена слева от машинного слова, в котором записано A. Тогда хеш-функция вычисляется следующим образом: $h(k) = M \cdot ((A \cdot k) \cdot \bmod 1)$.

Пример:

A = 0,385; M = 100.

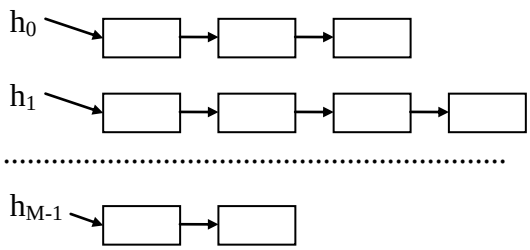
h(42) = 17.

Было придумано много других методов хеширования, но ни один из них не оказался предпочтительней описанных выше простых схем деления и умножения.

Тема 4.2 Методы разрешения коллизий.

Метод цепочек переполнения.

Наиболее простой способ разрешения коллизий состоит в том, чтобы поддерживать M линейных списков – по одному на каждый возможный хеш-адрес. Все записи должны иметь поля связи, кроме того, нужно иметь M голов списков Хеш-функция вычисляет номер списка, в котором следует искать запись. Поиск записи в списке производится последовательным перебором. Если имеется N записей и M списков, то средний размер списка равен N/M.



Тогда основные операции с таблицей выполняются следующим образом. Вставка: вычисляется номер списка; запись помещается в начало этого списка. Поиск: вычисляется номер списка, далее последовательный перебор. Удаление: операция поиска + удаление записи из списка.

Можно комбинировать списки с деревьями. Это поможет улучшить время поиска и вставки.

Метод открытой адресации.

Метод состоит в том, чтобы, отправляясь от вычисленного хеш-адреса, просматривать записи до тех пор, пока не будет найден искомый ключ или свободная позиция. Простейшая схема открытой адресации, известная как линейное опробирование, использует циклическую последовательность проб h(k), h(k)+1, h(k)+2, ..., M-1, 0, 1, ... h(k)-1.

Рассмотрим операции вставки, поиска и удаления.

Вставка. Вычисляется хеш-адрес. Если позиция по этому адресу свободна, то запись туда помещается. Если занята, от отправляясь от вычисленного хеш-адреса перебираем позиции таблицы одну за другой циклически в поисках свободной записи. Если свободная запись позиция найдется, следовательно, произведем вставку. Если не найдется, следовательно, таблица переполнилась. Тогда нужно выполнить рехеширование с большим значением M.

Поиск. Вычисляется хеш-адрес и начиная с него, перебираются все записи до тех пор, пока не обнаружим искомый ключ или не встретим свободную позицию. Если найдена свободная позиция, следовательно, ключа в таблице нет, ибо при вставке использовался тот же метод просмотра таблицы. Может потребоваться просмотр всей таблицы для поиска отсутствующего ключа, если таблица вся заполнена.

Удаление. Просто удалить запись, пометив ее как свободную нельзя, так как при этом нарушится работа алгоритмов поиска и вставки. Поэтому будем помечать записи как свободные, занятые и удаленные. При поиске удаленные записи трактуются как занятые. В случае вставки удаленные записи будут трактоваться как свободные. Следовательно, если встретится удаленная запись, вставка будет произведена в нее. Однако эта идея применима только при очень редких удалениях, ибо однажды занятая запись никогда не станет свободной. Поэтому после множества вставок и удалений все свободное пространство в таблице исчезнет и каждый неудачный поиск будет требовать просмотра всех M записей, т.е. сведется к последовательному. Следовательно, требуется рехеширование.

Такой метод работает хорошо, пока таблица не слишком заполнена. Для линейного апробирования характерно явление скучивания – скопления записей группами. Шанс для нового ключа попасть в кучу тем больше, чем больше размер кучи. Действительно, если адреса h(k), h(k)-1, ..., h(k)-i уже заняты (т.е. образовалась куча), то добавление новой записи с хеш-адресом из диапазона [h(k)-i, h(k)] увеличивает кучу. Следовательно куча склонна к экспоненциальному росту. Чтобы избежать скучивания используют двойное хеширование. Тогда шаг апробирования позиции и адреса будет следующим: h(k), h(k) + h_2(k), h(k) + 2h_2(k), ... Значение вторичной хеш-функции должно быть числом, взаимно простым с M, т.е. не должно иметь общих простых делителей.

Раздел 5. Файлы.

Массивы не пригодны для длительного хранения данных, по окончании работы программы данные теряются. Иногда результаты вычислений требуется сохранить для дальнейшей работы. Для этого используются файлы.

Файл это совокупность однотипных данных, имеющих имя и находящихся в памяти компьютера (внешней или в Фортране оперативной).

Файл состоит из записей. Запись – это единица обмена между программой и внешней памятью.

Входные и выходные данные программы могут размещаться в файлах.

Количество компонент в файле заранее не оговаривается. Объём информации хранимой на внешнем носителе ограничивается ёмкостью ЗУ

Различают два вида файлов: внешние и внутренние.

Тема 5.1 Внешние файлы.

Внешние файлы различают:

- по способу доступа к данным;
- по структуре (в зависимости от представления данных.

По способу доступа (методу доступа) к данным:

- последовательный;
- прямой.

Метод доступа определяет порядок, в котором можно обрабатывать записи файла.

Последовательный метод доступа предполагает последовательное считывание данных, до нахождения искомого. В файлах последовательного доступа новые данные добавляются только в конец файла. Последовательные неформатные файлы не могут непосредственно редактироваться. Записи в последовательном файле могут иметь разную длину. Для изменения записи в последовательном файле возможен только такой путь: прочитать все записи в массив, сделать изменения и записать массив в файл или использовать для редактирования временный вспомогательный файл, сделать в нем необходимые изменения, и затем переписать содержимое временного файла в исходный. По окончании работы временный файл автоматически удаляется.

Прямой метод доступа позволяет читать записи в произвольном порядке (добраться до любой компоненты, не перебирая предыдущие). При такой организации компоненты файла (записи) имеют фиксированную длину (recl, которая указывается в операторе open)и свой уникальный номер (от 1 до n). Для доступа к данным надо указать только их номер. Их можно считывать и записывать в любом порядке.

В зависимости от представления данных внешние файлы могут быть (различаются по структуре):

Форматными;

Неформатными;

Бинарными;

Вид записи задается в операторе Open. Записи форматного файла хранятся в виде символов таблицы ASCII и представляют собой последовательность символов. Числа хранятся в символьном представлении. Каждая запись заканчивается символом возврата каретки(CR) и перевод строки(LF). Форматные файлы могут создаваться, редактироваться и просматриваться в любом текстовом редакторе (поддерживающим кодировку). Если необходим визуальный просмотр данных, то следует использовать форматный файл.

В неформатных файлах данные хранятся во внутреннем представлении. Они обрабатываются быстрее, т. к. не требуется преобразование из внутреннего представления во внешнее (форматное). Служат в основном для запоминания промежуточных результатов вычислений. Неформатные файлы могут быть созданы и просмотрены только программным путем.

Данные в бинарных файлах хранятся в двоичном представлении. Эти файлы – нетипичны для Фортрана (поддерживаются не во всех версиях), и мы их рассматривать не будем.

Таким образом, Фортран поддерживает два метода доступа и три структуры (форматные, неформатные, бинарные). Поэтому в Фортране существуют 6 разновидностей файлов. Мы будем рассматривать только типичные файлы:

Форматные последовательные (текстовые);

Неформатные последовательные;

Неформатные прямые

Обычно в программе при организации ввода/вывода предполагается следующий порядок операторов:

Сначала оператором open файл открывается (присоединяется к устройству, и определяются его свойства). Затем осуществляются действия по передаче данных (read, write). После окончания работы с файлом, его отсоединяют от устройства (закрывают файл - close).

При выборе типа файла надо учитывать далее перечисленные обстоятельства. Если необходим визуальный контроль данных, то следует использовать форматные файлы. Но в текстовых файлах могут возникнуть ошибки округления. Неформатные файлы имеют меньший размер, по сравнению с текстовыми файлами. В неформатных файлах передача данных осуществляется быстрее, т.к. не требуется преобразования данных.

Тема 5.2 Внутренние файлы.

Внутренний файл - это область оперативной памяти, заданная символьной переменной. Он хранится (существует только) в оперативной памяти и открыт по умолчанию. Запись данных во внутренний файл происходит гораздо быстрее, чем во внешний файл. Внутренние файлы используются для преобразования данных из одного типа в символьный (другой). Они открыты по умолчанию. Внутренние файлы имеют только последовательный метод доступа. Устройством внутреннего файла является имя строки. Это символьная переменная (такой файл содержит одну запись, размер которой совпадает с размером переменной), элемент символьного массива или символьный массив (такой файл это последовательность элементов, каждый из которых является записью и число записей равно числу элементов массива)

Пример

```
Real:: c=23.5
Integer n
Character(15) st
Write(st,*)c ! преобразование «число-строка»
Print*, st ! на экране: _____23.50000
Read(st,’(I5)’)n !преобразование строка-число
Print*,n !_____23
End
```

Внешний файл - это файл, который хранится на диске (последовательность записей на носителе) или внешние устройства (клавиатура, экран).

Чтобы работать с внешним файлом, его надо открыть (присоединить к устройству ввода/вывода). При открытии и создании файла указатель устанавливается на начало файла. Чтение и запись данных из файла автоматически вызывает перемещение указателя, т.е. указатель перемещается к началу следующей записи. При записи данных после последней компоненты ставится признак конца файла.

Для определения конца файла используется логическая функция Eof, которая возвращает значение .TRUE., если достигнут конец файла и .FALSE.в противном случае.

После окончания работы с файлом его надо закрыть.

В Фортране обращение к файлам данных происходит через канал (или устройство). Это логическое понятие, т.е. канал не устройство в обычном понимании, а воображаемый. Прежде чем выполнять ввод/вывод необходимо установить связь между физическим файлом и устройством, т.е. присоединить файл к устройству (каналу). Устройство обозначается идентификатором и это - звездочка * или целочисленное скалярное выражение. Для внешнего файла - это числовое значение от 1 до 32767 (2_147_483_640). Максимальное значение номера устройства зависит от конкретной реализации.

Каждому файлу соответствует свое логическое устройство. Одно физическое устройство может соответствовать разным логическим устройствам.

К устройствам ввода/вывода могут быть присоединены стандартные физические устройства (клавиатура и экран). Для экрана и клавиатуры в Фортране предусмотрены каналы с определенными номерами. В Фортран-программах существуют устройства с идентификаторами *, 0, 5, 6. Причем по умолчанию к устройствам *, 0, 5 присоединена клавиатура, а к устройствам *, 0, 6 – экран.

Write(n, m)<список вывода>

Real :: x=5.2

Write(*,'(f6.2)') x

Write(0,'(f6.2)') x

Write(6,'(f6.2)') x

Все операторы это – вывод на экран.

Отсоединяются эти устройства автоматически после окончания работы программы. Эти устройства могут присоединяться к любому файлу.

Каждый внешний файл имеет имя. Оно должно удовлетворять правилам именования операционной системы: 'c\users\A123\f1.txt' – это спецификация внешнего файла или, если это устройства - зарезервированные имена, например con, prn.

Внешний файл присоединяется к устройству ввода/вывода в результате выполнения оператора OPEN . Теперь доступ к внешнему файлу выполняется по номеру устройства, к которому он присоединен.

OPEN (unit=2, file = 'c\users\A133\f1.txt')

Устройство не может быть присоединено более, чем к одному файлу и наоборот.

Файл состоит из записей. Под словом «запись» понимается «логическая запись». Записью называют единицу обмена данными между программой и внешней памятью.

Запишем в файлы по два целых числа различными способами:

```
program main
Integer::x=25, y=10
open(1, file='numbers1.txt')
write(1,*) x
write(1,*) y
close(1)
open(2, file='numbers2.txt')
write(2,*) x, y
close(2)
end
! В файле number1.txt будет:
25
10
!в файле number2.txt будет: 25 10
```

Запишем в файл 5 чисел, используя различные форматы при записи:

```
program main
integer i
open(10, file='number1.txt')
do i=1,5
write (10, '(I3\)') i
enddo
close(1)
end
```

В файле получим: __1__2__3__4__5

```
program main
integer i
open(10, file='number1.txt')
do i=1, 5
write (10, '(I3)') i
enddo
close(1)
end
```

В файле получим:

__1
__2
__3
__4
__5

Все записи в файле имеют одинаковый вид.

В Фортране различают следующие виды записей (в зависимости от представления данных):

форматные – они состоят из символов кодовой таблицы. При вводе данные преобразуются из внешнего (символьного) представления - во внутреннее представление, а при выводе – из внутреннего во внешнее;

неформатные (бесформатные), которые содержат данные во внутреннем машинном представлении, служат в основном для запоминания результатов в промежуточных вычислениях.

конец файла.

Вид записи задается в операторе (теперь говорят в утверждении) open спецификаторами form='formated' или form='unformated'

Запись "конец файла" не содержит данных и ставится автоматически за последней записью файла.

Раздел 6. Типы данных нелинейной структуры.

.

Тема 6.1 Деревья.

Корневое дерево (rooted tree) – это структура данных, которая представляет собой совокупность связанных узлов (nodes), один из которых является корнем дерева (root).

Ребра (edges) связывают узлы дерева и устанавливают между ними отношение .родитель-дочерний. (parent-child). На рис. 1 приведен пример корневого дерева.

Узел, который не имеет дочерних вершин, называется внешним узлом (external node), или листом (leaf node). На рис. это узлы: 4, 8, 9, 6 и 7. Аналогично, узел дерева, имеющий дочерние вершины называется внутренним узлом (internal node).

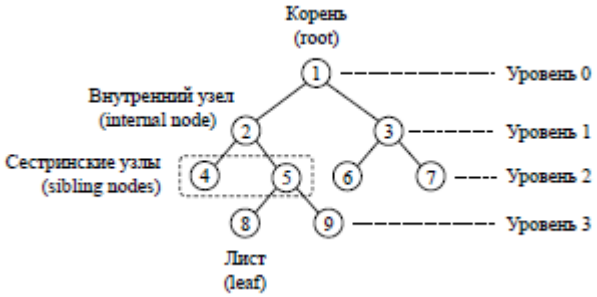


Рис. Корневое дерево из 9 элементов (высота дерева 3).

Любой внутренний узел по отношению к своим дочерним вершинам (child nodes) называется родительским (parent node). Узлы, имеющие общего родителя называются родственными, или сестринскими узлами (sibling nodes). Любой узел в поддереве некоторой вершины называется ее потомком (descendant), а вершина для этих узлов является предком (ancestor).

Степень узла (node degree) – это количество его дочерних узлов.

Высота узла (node height) – количество ребер в длиннейшем пути от узла до листа.

Высота дерева (tree height) – количество ребер в длиннейшем пути от корня до листа.

Глубина узла (node depth) или уровень узла (node level) – количество ребер в пути от узла до корня.

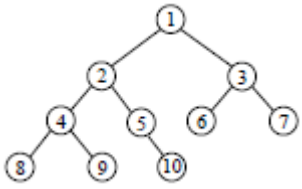
Рассмотрим подробнее введенные определения. На рис. приведено дерево из 9 узлов. Вершина 2 является родителем для вершин 4 и 5, а также предком для узлов 4, 5, 8 и 9. Узлы 4 и 5 является сестринскими. Высота дерева равна 3, а количество уровней – 4. Степень узла 5 равна 2, а его высота 1.

Тема 6.2 Бинарные деревья.

Бинарное дерево (двоичное дерево, binary tree) – это корневое дерево, в котором каждый узел имеет не более двух дочерних вершин (0, 1 или 2 вершины).

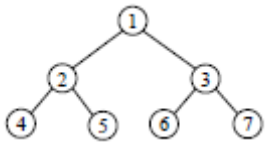
Полное бинарное дерево (full binary tree) – это бинарное дерево, в котором каждый узел имеет 0 или 2 дочерних узла. Пример такого дерева приведен на рис.

Завершенное бинарное дерево (complete binary tree) – это бинарное дерево, в котором каждый уровень, возможно за исключением последнего, полностью заполнен узлами, а заполнение последнего уровня осуществляется слева направо (рис.).



Совершенное бинарное дерево (perfect binary tree).

Это бинарное дерево, в котором все листья имеют одинаковую глубину (рис.). При числе узлов n минимальной высотой $n = \log2(n + 1) - 1$ обладает совершенное бинарное дерево, а максимальной высотой – бинарное дерево, в котором каждый внутренний узел имеет единственную дочернюю вершину (дерево вырождается в цепочку высоты $n - 1$). Следовательно, высота n любого бинарного дерева из n узлов ограничена снизу и сверху $n\log2(n + 1)n - 1 \le n \le n - 1$.



Обход бинарных деревьев.

Во многих алгоритмах, использующих деревья, возникает необходимость перебрать все узлы дерева начиная с корня. Такая процедура называется обходом дерева (tree traversal). Каждая вершина должна быть посещена один раз. Посещение вершины может быть связано с выполнением некоторых вычислений и или обработкой данных, хранящихся в ней.

Обход бинарного дерева можно выполнять в ширину и в глубину. Все способы обходов дерева начинают свою работу с корня.

При обходе в ширину (breadth first search, BFS) узлы дерева посещаются слева направо, уровень за уровнем. Например, при обходе в ширину дерева на рис. вершины будут посещены в следующем порядке:

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

При обходе в глубину (depth first search, DFS) сперва посещаются все дочерние вершины левого поддерева и уже потом происходит переход к правому поддереву текущего узла.

Существует три способа обхода дерева в глубину, различающихся порядком посещения вершин и их дочерних узлов.

1. Обход в прямом порядке (префиксный обход, preorder traversal) – каждый узел посещается до того, как рекурсивно посещены его потомки. Для каждого узла *x* рекурсивно выполняется последовательность действий посетить узел *x*; обойти левое поддерево; обойти правое поддерево.

При таком обходе узлы дерева на рис. будут посещены в следующем порядке:

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

2. Симметричный обход (инфиксный обход, in-order traversal) – сперва посещаются потомки левого поддерева узла, затем сам

Результат симметричного обхода дерева:

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

3. Обход в обратном порядке (постфиксный обход, post-order traversal) – каждый узел посещается после того, как рекурсивно посещены его потомки. Для каждого узла *x* рекурсивно выполняется последовательность действий: обойти левое поддерево, обойти правое поддерево, посетить узел *x*. Результат обхода в обратном порядке дерева на рис.:

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

Тема 6.3 Пирамиды.

Представляемые массивами деревья находят применение в имеющих большое значение приложениях с пирамидами (heaps), являющимися законченными бинарными деревьями, имеющими упорядочение узлов по уровням. В максимальной пирамиде (maximum heap) родительский узел больше или равен каждому из своих сыновей. В минимальной пирамиде (minimum heap) родительский узел меньше или равен каждому из своих сыновей. Эти ситуации изображены на рис. В максимальной пирамиде корень содержит наибольший элемент, а в минимальной — наименьший.

Пирамида является списком, который хранит некоторый набор данных в виде бинарного дерева. Пирамидальное упорядочение предполагает, что каждый узел пирамиды содержит значение, которое меньше или равно значению любого из его сыновей. При таком упорядочении корень содержит наименьшее значение данных. Как абстрактная списковая структура пирамида допускает добавление и удаление элементов. Процесс вставки не подразумевает, что новый элемент занимает конкретное место, а лишь требует, чтобы поддерживалось пирамидальное упорядочение. Однако при удалении из списка выбрасывается наименьший элемент (корень). Пирамида используется в тех приложениях, где клиенту требуется прямой доступ к минимальному элементу. Как список пирамида не имеет операции поиска и осуществляет прямой доступ к минимальному элементу в режиме "только чтение". Все алгоритмы обработки пирамид сами должны обновлять дерево и поддерживать пирамидальное упорядочение.

Пирамидальная сортировка.

Пирамидальная сортировка (heap sort) основывается на организации элементов в массиве по типу двоичного (бинарного) дерева. Двоичным деревом называют иерархическую структуру данных, в которой каждый элемент имеет не более двух потомков:

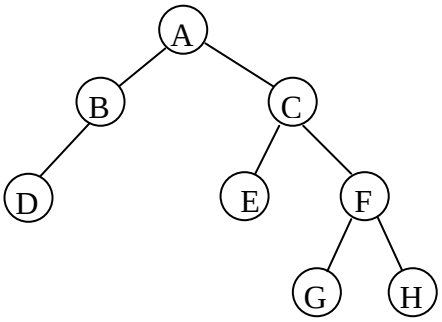


Рис Пример бинарного дерева.

Пирамида представляет собой особый вид бинарного дерева, в котором значение каждого элемента больше, чем значение каждого из его потомков. Непосредственные потомки каждого узла не упорядочены, поэтому иногда левый потомок может оказаться больше правого, а иногда – наоборот. Пирамида представляет собой полное дерево, в котором заполнение нового уровня начинается только после того, как предыдущий уровень полностью заполнен, а все узлы на одном уровне заполняются последовательно:

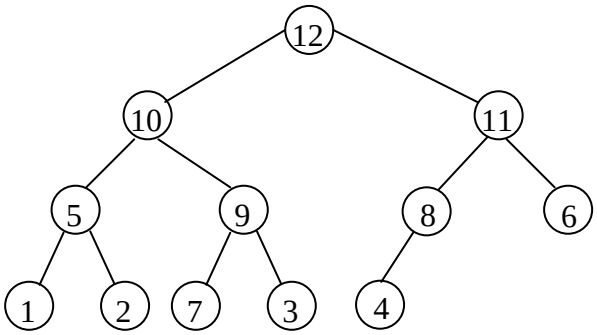


Рис. Пример пирамиды.

Сортировка начинается с построения пирамиды, содержащей все элементы исходной последовательности. Ясно, что максимальный элемент окажется в вершине дерева, поскольку все её потомки обязательно должны быть меньше. Затем, вершина пирамиды записывается в конец последовательности, а пирамида с удаленным максимальным элементом переформировывается. В результате, на её вершине оказывается максимальный из оставшихся элементов, он записывается на соответствующее место, и процедура продолжается до тех пор, пока пирамида не опустеет.

Таким образом, основную нагрузку алгоритма несут две процедуры: начальное построение пирамиды и переформирование пирамиды на каждом шаге. Для построения пирамиды можно воспользоваться её свойством – у каждого внутреннего узла ровно два непосредственных потомка, за исключением, быть может, одного узла на предпоследнем уровне. Следовательно, можно пронумеровать узлы, начиная с вершины по правилу: если узел имеет номер i, то его потомкам назначим номера 2*i и 2*i+1. В результате все узлы пирамиды получат уникальные номера в диапазоне 1 ~ N, что дает возможность хранить пирамиду в виде последовательности:

12	10	11	5	9	8	6	1	2	7	3	4
1	2	3	4	5	6	7	8	9	10	11	12

При удалении наибольшего элемента пирамиды из вершины, корневой узел остается свободным. После перестроения пирамиды в него должен попасть больший из двух непосредственных потомков, в свою очередь аналогично нужно определить, кто встанет на его место и т.д. При этом пирамида должна оставаться настолько близкой к полному дереву, насколько возможно. С этой целью переформирование пирамиды будем начинать с крайнего правого элемента на нижнем уровне, перенося его на вершину. Затем к вершине применим процедуру, называемую «просеиванием вниз»:

- если элемент не имеет потомков, то конец;
- иначе меняем местами значения элемента и максимального из двух его непосредственных потомков;
- переходим к изменившемуся потомку и выполняем для него ту же процедуру с шага 1.

Такой подход к перестроению пирамиды можно использовать также и для построения её начального состояния: любые два элемента можно считать потомками некоторого свободного узла. Поскольку все элементы являются «листьями», со второй половиной последовательности можно ничего не делать. Начиная с набора листовых элементов, можно соединять их попарно до тех пор, пока не будет сформирована общая большая пирамида. Первый добавляемый корень в последовательности из N элементов будет иметь индекс N / 2 - 1. В итоге алгоритм начального построения пирамиды будет заключаться в последовательном применении процедур «просеивания вниз» ко всем элементам с индексами от N / 2 - 1 до 0.

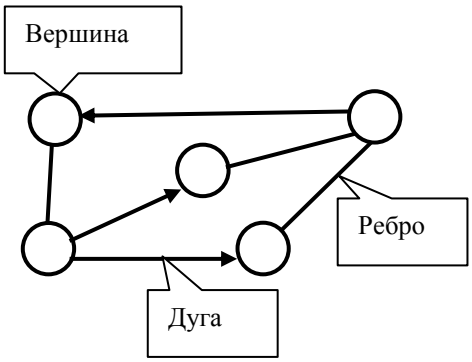
```
// Функция "просеивает" элемент номер i вниз в пирамиде heap размера size
void fixHeap( double * heap, int i, const int size )
{
   // Индекс максимального элемента в текущей тройке элементов:
   int maxIdx = i ;
   // Значение текущего элемента:
   double value = heap[i];

   while ( true )
   {
     int childIdx = i * 2 + 1; //Индекс левого потомка

     // Если есть левый потомок и он больше текущего элемента,
     if ( ( childIdx < size ) && ( heap[ childIdx ] > value ) )
       maxIdx = childIdx; //  то он считается максимальным

      childIdx = i * 2 + 2; //Индекс правого потомка
            // Если есть правый потомок и он больше максимального,
      if ( ( childIdx < size ) && ( heap[ childIdx ] > heap[ maxIdx ] ) )
         maxIdx = childIdx; //  то он считается максимальным

      // Если текущий элемент является максимальным из трёх
      // (т.е. если он больше своих детей), то конец:
      if ( maxIdx == i )
        break;

      // Меняем местами текущий элемент с максимальным:
      heap[i] = heap[ maxIdx ];
      heap[ maxIdx ] = value;

      // Переходим к изменившемуся потомку
      i = maxIdx;
   }
}
// Пирамидальная сортировка массива heap размера size
void heapSort( double * heap, int size )
{
   // Построение пирамиды из массива:
   for( int i = size / 2 - 1; i >= 0; --i )
     fixHeap( heap, i, size );
      // Сортировка с помощью пирамиды
   while( size > 1 ) // пока в пирамиде больше одного элемента
   {
      --size; // Отделяем последний элемент
      // Обмениваем местами корневой элемент и отделённый:
      double firstElem = heap[0];
      heap[0] = heap[ size ];
      heap[ size ] = firstElem;
            // "Просеиваем" новый корневой элемент вниз:
      fixHeap( heap, 0, size );
   }
}
```

Раздел 7. Графы.

.

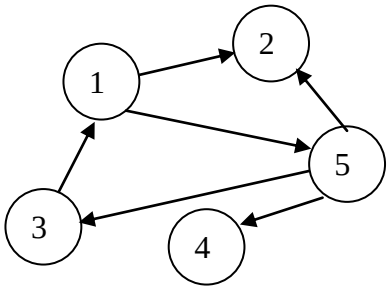
Тема 7.1 Основные понятия теории графов.

Граф (graph) – объект, состоящий из множества кружков (точек) и множество соединяющих их линий. Кружки (точки) называются вершинами графа (nodes), линии со стрелками – дугами (arcs), без стрелок – ребрами (edges). Т.е. граф – это пара G=(V,E), где V - множество вершин, а E - семейство пар ребер $e_i=(v_{i1}, v_{i2})$, v_{ij} принадлежит V. Вершины графа можно использовать для представления объектов, а дуги — для отношений между объектами.



Графы обычно изображаются в виде геометрических фигур. В классическом графе отсутствуют *петли*, то есть ребра, соединяющие вершину саму с собой.

Граф, в котором направление линий принципиально (линии являются дугами) называется *ориентированным (орграф)*. В отличие от ребер, дуги соединяют две неравноправные вершины: одна из них называется началом дуги (дуга из нее исходит), вторая - концом дуги (дуга в нее входит). Пример: G=(V,E): V={1, 2, 3, 4, 5}; E={(1,2), (1,5), (3,1), (5,2), (5,3), (5,4)}.



Граф, в котором направление линий не выделяется (все линии являются ребрами), называется ***неориентированным*** (две вершины равноправны, нет никакой разницы между "началом" и "концом" ребра).

Чаще всего рассматривают графы, в которых все ребра имеют один тип – либо ориентированные, либо неориентированные.

Две вершины **v** и **u** называются ***смежными***, если они соединены ребром (дугой) **e**: **e**=(**v,u**). Смежные вершины называются граничными вершинами соответствующего ребра (дуги), а это ребро (дуга) - ***инцидентным*** соответствующим вершинам. Любому ребру инцидентно ровно две вершины, а вершине может быть инцидентно произвольное количество ребер.

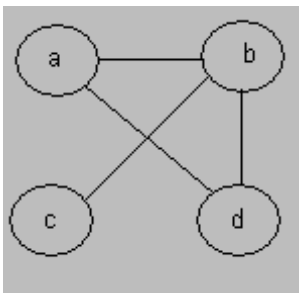
Два ребра называются ***смежными***, если они инцидентны одной вершине.

Степенью вершины в неориентированном графе называется число инцидентных ей ребер. Для ориентированного графа различают исходящую степень, определяемую как число выходящих из него ребер, и входящую степень, определяемую как число входящих в нее ребер. Сумма исходящей и входящей степеней называется степенью вершины. Изолированная вершина – это вершина со степенью 0 (нуль-граф).

Мультигаф – из одной вершины в другую можно перейти разными способами (граф с кратными ребрами).

Путем называется последовательность вершин v1, v2, …, vn, для которой существуют ребра (или дуги в орграфе) v1 → v2, v2 → v3, ..., vn-1 → vn. Этот путь начинается в вершине v1 и, проходя через вершины v2, v3, ..., vn-1, заканчивается в вершине vn. Длина пути— количество дуг (ребер), составляющих путь, в данном случае длина пути равна n – 1. Путь называется простым, если все вершины на нем, за исключением, может быть, первой и последней, различны. Для неориентированного графа на рис. путями будут adbc и abc.

Замкнутый путь без повторяющихся ребер называется циклом (или контуром в орграфе); без повторяющихся вершин (кроме первой и последней) - простым циклом.



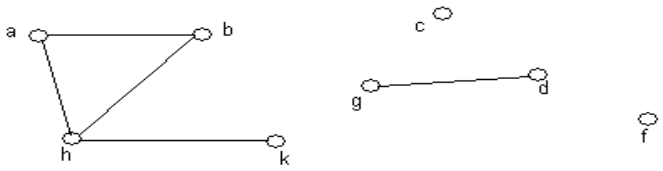
Полным называется граф, в котором проведены все возможные ребра. Для графа, имеющего n вершин, таких ребер будет n(n-1)/2.

Вершина v достижима из вершины u, если существует путь, начинающийся в u и заканчивающийся в v.

Граф называется связным, если все его вершины взаимно достижимы. На рисунке изображен связный граф.

Если граф не является связным, то его можно разбить на связные подграфы, называемые компонентами.

Компонента связности – это максимальный связный подграф. В общем случае граф может состоять из произвольного количества компонент связности. Любая изолированная вершина является отдельной компонентой связности. Например, в приведённом ниже графе содержится 4 компоненты связности: abhk, gd, c, f.



Взвешенный граф – это граф, некоторым элементам которого (вершинам, ребрам или дугам) сопоставлены числа. Числа-пометки носят названия вес, длина, стоимость.

Длина пути во взвешенном графе – это сумма весов ребер (дуг), из которых состоит путь.

Тема 7.2 Основные процедуры работы с графами.

Добавление вершины

Процедура AddNode добавляет в граф вершину с заданным номером n и весом x. Предполагается, что уникальность номера обеспечивается вызывающей программой.

```
Procedure AddNode(Var graph: refnode; n,x:integer);
Var p:refnode;
Begin new(p);
     With p^ do
          Begin id:=n;
               Infnode:=x;
               Arclist:=nil;
               Next:graph
          End;
          graph:=p;
end;
```

Добавление дуги

Процедура добавления дуги AddArc в граф использует в качестве входной информации ссылки на соединяемые вершины и значения вес создаваемой дуги. Обе вершины должны существовать к моменту выполнения процедуры.

```
Procedure AddArc(u,v:refnode; y:integer);
Var a:refarc;
Begin if u=nil or v=nil then writeln('Отсутствует вершина')
        Else begin
             New(a);
             With a^ do begin
                Infarc:=y;
                Adj:=v;
                Next:=u^.arclist
              End;
         u^.arclist:=a
   end
end;
```

Удаление дуги вершин

Для удаления дуги указываются две ссылки на вершины, которые эта дуга соединяет. Приведенная ниже процедура удаляет элемент из списка дуг, выходящих из вершины u, если в нем записана ссылка на вершину v. Если таких элементов несколько, то удаляется первый встретившийся.

```
Procedure DeleteArc(u,v:refnode);
Var a,before:refarc;
    Run:boolean;
Begin
  If u<>nil then Begin
    a:=u^.arclist;
    run:=truel
    while a<>nil and run do
      if a^.adj=v then run:=false
                  else begin
                       before:=a;
                       a:=a^.next
                       end;
   if a<>nil then begin
        if a=u^.arclist then u^.arclist:=a^.next
                  else before^.next:=a^.next;
        dispose(a)
  end
 end
end;
```

Процедура удаления вершины более сложная, так как кроме удаления дескриптора вершины и подцепленного к ней списка выходящих дуг, необходимо просмотреть всю оставшуюся часть графа в поисках висячих ссылок. Эти ссылки удаляются с помощью процедуры DeleteArc.

```
Procedure DeleteNode(Var graph:refnode; v:refnode);
Var before,p,q:refnode;
     a,after:refarc;
begin
 p:=graph;
 while p<>nil do {цикл по всем вершинам графа}
  begin
   q:=p^.next;
   if p<>v then begin
    DeleteArc(p,v) {удаление дуг, направленных удаляемой вершине}
```

```
        before:=p;
       end
      else Begin {удаление вершины v и всех дуг, исходящих из нее}
        if p=graph then graph:=q; {если это первая вершина}
         a:=p^.arclist;
         {удаление списка дуг вершины v}
         while a<>nil do
          begin
           after:=a^.next;
           dispose(a);
           a:=after
          end;
         if p=graph then graph:=q
         else before^.next:=q;
         dispose(p);
       end;
      p:=q;
    end
  end;
```

Просмотр графа

Просмотр графа заключается в посещении всех вершин и дуг графа и выводе в стандартный файл всей информации о графе. Для непустого графа информация группируется по вершинам (списки смежности).

```
Procedure BrowseGraph(graph:refnode);
Var a:refarc;
     Nn,na: integer; {счетчики количества вершин и дуг}
Begin
 Nn:=0; na:=0;
 while graph<>nil do
  begin
   writeln('Вершина ', graph^.id,' с весом ' ,graph^.infnode);
   writeln('Список дуг:');
   a:=graph^.arclist;
   if a=nil then writeln('пуст');
   while a<>nil do begin
    writeln('Дуга к вершине ', (a^.adj)^.id,', вес дуги ',a^.infarc);
    na:=na+1;
    a:=a^.next;
   end;
   nn:=nn+1;
   graph:=graph^.next;
 end;
   writeln('В графе ', nn, 'вершин и ', na,' дуг')
end;
```

Тема 7.3 Алгоритмы на графах: поиск в глубину и поиск в ширину.

Обход (поиск) в глубину

Метод поиска в глубину (depth first search, DFS) — обобщение метода обхода дерева в прямом порядке. Обход в глубину (называемый иногда стандартным обходом), есть обход графа по следующим правилам:

1. Добавляет начальную вершину в стек и помечает её как использованную.
2. Рассматривает верхнюю вершину стека и добавляет в стек первую попавшуюся смежную ей неиспользованную вершину, помечая её как использованную. Если такой вершины нет, извлекает вершину стека.
3. Если стек не пуст, переходит к пункту 2.

Таким образом, этот алгоритм обходит все вершины, достижимые из начальной, и каждую вершину обрабатывает не более одного раза.

Текст процедуры обхода в глубину (рекурсивный вариант):

```
var a:array[1..nmax,1..nmax]of integer;// матрица смежности
   used:array[1..nmax]of boolean; // список меток
   n:integer; // количество вершин графа
 procedure dfs(v:integer);
 var
   i:integer;
 begin
   used[v]:=true; // помещенная в стек вершина помечается использованной
   for i:=1 to n do // рассматриваются все ребра, исходящие из этой вершины
```

```
// проверка, использована ли вершина, в которую ведет выбранное ребро, если да, то поиск продолжается из этой вершины.
      if (a[v,i]=1)and(not used[i]) then
        dfs(i);
  end;
...
  dfs(X); // здесь X - номер начальной вершины
```

При выполнении обхода графа по этим правилам стремимся проникнуть "вглубь" графа так далеко, как это возможно, затем отступаем на шаг назад и снова стремимся пройти вперед и так далее.

Обход (поиск) в ширину

Обход графа в ширину (breadth first search, BFS) основывается на замене стека очередью:

1. Добавляет начальную вершину в очередь и помечает её как использованную.

2. Извлекает следующую вершину из очереди и добавляет в очередь смежные ей неиспользованные вершины, помечая их как использованные.

3. Если очередь не пуста, переходит к пункту 2.

После такой модификации чем раньше посещается вершина (помещается в очередь), тем раньше она используется (удаляется из очереди). Использование вершины происходит с помощью просмотра сразу всех еще не просмотренных вершин, смежных этой вершины. Таким образом, "поиск ведется как бы во всех возможных направлениях одновременно".

Очередность просмотра вершин при поиске в ширину:

```
procedure bfs(v:integer);
var Og:array[1..nn]of integer;
    yk1,yk2:integer;
    i:integer;
begin
// в элементе og[i] хранится номер группы вершины i. Изначально номер группы всех вершин кроме стартовой равен 0, это значит, что они ещё не были использованы.
   for i:=1 to n do
    og[i]:=0;
// Инициализация очереди, т.е. добавление в неё начальной вершины с номером v
   yk2:=0;
   yk1:=1;
   Og[yk1]:=v;
   used[v]:=true; // пометка вершины использованной
   while yk2 < yk1 do // цикл работает, пока очередь не пуста
     begin
       inc(yk2);v:=Og[yk2];
       write(v:2);
// просматриваются все рёбра, исходящие из первой вершины очереди
       for i:=1 to n do
// использована ли вершина, в которую ведёт выбранное ребро, если нет , то вершина добавляется в очередь
          if (a[v,i] <> 0) and not used[i] then
            begin
// сдвигается указатель конца очереди
              inc(yk1);
              Og[yk1]:=i;
              used[i]:=true;
            end;
     end;
end;
```

Чаще всего поиск в ширину используется для нахождения кратчайшего пути от одной вершины X до другой – Y: нужно запустить поиск в ширину из вершины X и при добавлении новой вершины в очередь смотреть, не является ли она вершиной Y.

Если программа не завершит работу в условном цикле, это значит, что пути из X в Y не существует.

4.3. Лабораторные работы

№ п/п	*Номер раздела дисциплины*	*Наименование лабораторной работы*	*Объем (час.)*	*Вид занятия в интерактивной, активной, инновационной формах, (час.)*
1	**2**	**3**	**4**	**5**
1	**1.**	Списки.	2	разбор конкретных ситуаций (2 часа)
2	**2.**	Типы линейной структуры данных.	4	разбор конкретных ситуаций (2 часа)
3	**3.**	Алгоритмы сортировки массивов методами выбора, обмена, вставок и челночным методом.	4	разбор конкретных ситуаций (2 часа)
4	**4.**	Хеширование.	2	разбор конкретных ситуаций (2 часа)
5	**5.**	Файлы.	4	разбор конкретных ситуаций (2 часа)
6	**6.**	Деревья.	4	разбор конкретных ситуаций (2 часа)
7	**6.**	Бинарные деревья.	5	разбор конкретных ситуаций (2 часа)
8	**6.**	Пирамиды.	5	разбор конкретных ситуаций (2 часа)
9	**7.**	Графы.	4	разбор конкретных ситуаций (2 часа)
		ИТОГО	**34**	**18**

4.4. Практические занятия

Учебным планом не предусмотрено

4.5. Контрольные мероприятия: контрольная работа.

Цель: закрепить теоретические знания в области сортировки массивов, познакомиться с видами сортировки и провести их сравнение.

Структура: каждое индивидуальное задание предполагает выполнение обучающимся следующих разделов:

1. Введение.
2. Задание.
3. Сортировка массивов методом выбора.
4. Сортировка массивов методом обмена.
5. Сортировка массивов челночным методом.
6. Сортировка массивов методом вставок.
7. Вывод.

Основная тематика: сортировка массивов.

Рекомендуемый объем: пояснительная записка объемом 15 - 20 страниц должна содержать титульный лист, задание, описание выполняемых действий по каждому разделу и полученные результаты.

Выдача задания, прием и защита проводится в соответствии с календарным учебным графиком

Оценка	Критерии оценки
отлично	Контрольная работа сдана в первую неделю защит. В контрольной работе полностью раскрыта тема работы. Правильно решены все задания работы.
хорошо	Контрольная работа сдана в срок, но содержит незначительные ошибки.
удовлетворительно	Контрольная работа сдана в срок, но содержит значительное количество ошибок, или ошибка подразумевает полную переработку всей контрольной работы.
неудовлетворительно	Контрольная работа не сдана в установленный срок.

5. МАТРИЦА СООТНЕСЕНИЯ РАЗДЕЛОВ УЧЕБНОЙ ДИСЦИПЛИНЫ К ФОРМИРУЕМЫМ В НИХ КОМПЕТЕНЦИЯМ И ОЦЕНКЕ РЕЗУЛЬТАТОВ ОСВОЕНИЯ ДИСЦИПЛИНЫ

Компетенции *№, наименование разделов дисциплины*	*Кол-во часов*	*Компетенции*			*Σ комп.*	$t_{ср}$, *час*	*Вид учебных занятий*	*Оценка результатов*
		ОК	*ОПК*	*ПК*				
		7	*6*	*5*				
1	2	3	4	5	6	7	8	9
1. Классификация структур данных.	10	+	+	+	3	3,3	Лк, ЛР, СРС	экзамен
2. Типы данных линейной структуры.	13	+	+	+	3	4,3	Лк, ЛР, СРС	экзамен
3. Алгоритмы сортировки массивов.	21	+	+	+	3	7	Лк, ЛР, СРС	экзамен, к
4. Хеширование.	21	+	+	+	3	7	Лк, ЛР, СРС	экзамен
5. Файлы.	21	+	+	+	3	7	Лк, ЛР, СРС	экзамен
6. Типы данных нелинейной структуры.	21	+	+	+	3	7	Лк, ЛР, СРС	экзамен
7. Графы.	19	+	+	+	3	6,3	Лк, ЛР, СРС	экзамен
всего часов	**126**	**42**	**42**	**42**	**3**	**42**		

6. ПЕРЕЧЕНЬ УЧЕБНО-МЕТОДИЧЕСКОГО ОБЕСПЕЧЕНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ ОБУЧАЮЩИХСЯ ПО ДИСЦИПЛИНЕ

Мейер, Б. Инструменты, алгоритмы и структуры данных / Б. Мейер. - 2-е изд., испр. - М.: Национальный Открытый Университет «ИНТУИТ», 2016. - 543 с. : схем., ил. - Библиогр. в кн.; то же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=429033 (стр. 257-338).

7. ПЕРЕЧЕНЬ ОСНОВНОЙ И ДОПОЛНИТЕЛЬНОЙ ЛИТЕРАТУРЫ, НЕОБХОДИМОЙ ДЛЯ ОСВОЕНИЯ ДИСЦИПЛИНЫ

№	*Наименование издания*	*Вид занятия*	*Количество экземпляров в библиотеке, шт.*	*Обеспеченность, (экз./ чел.)*
1	2	3	4	5
	Основная литература			
1.	Мейер, Б. Инструменты, алгоритмы и структуры данных / Б. Мейер. - 2-е изд., испр. - М.: Национальный Открытый Университет «ИНТУИТ», 2016. - 543 с. : схем., ил. - Библиогр. в кн.; то же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=429033.	Лк, ЛР, КР	ЭР	1
2.	Курносов М.Г. Введение в структуры и алгоритмы обработки данных: учебник / М.Г. Курносов – Новосибирск: Автограф, 2015. – 179 с., то же [Электронный ресурс] http://ecat.brstu.ru/catalog/Ресурсы%20свободного%20доступа/Курносов%20М.Г.%20Введение%20в%20структуры%20и%20алгоритмы%20обработки%20данных.%20Учебник.%202015.pdf	Лк, ЛР, КР	ЭР	1
	Дополнительная литература			
3.	Хусаинов, Б.С. Структуры и алгоритмы обработки данных. Примеры на языке Си: учебное пособие для вузов / Б. С. Хусаинов. - Москва: Финансы и статистика, 2004. – 464 с.	Лк, ЛР, КР	9	0,5
4.	Серебряная, Л. В. Структуры и алгоритмы обработки данных: учеб.-метод. пособие / Л. В. Серебряная, И. М. Марина. – Минск : БГУИР, 2013. – 51 с., то же [Электронный ресурс] http://ecat.brstu.ru/catalog/Ресурсы%20свободного%20доступа/Серебряная%20Л.В.%20Структуры%20и%20алгоритмы%20обработки%20данных.%20Учеб.-метод.%20пособие.%202013.pdf	Лк, ЛР, КР	ЭР	1

8. ПЕРЕЧЕНЬ РЕСУРСОВ ИНФОРМАЦИОННО - ТЕЛЕКОММУНИКАЦИОННОЙ СЕТИ «ИНТЕРНЕТ» НЕОБХОДИМЫХ ДЛЯ ОСВОЕНИЯ ДИСЦИПЛИНЫ

1. Электронный каталог библиотеки БрГУ
http://irbis.brstu.ru/CGI/irbis64r_15/cgiirbis_64.exe?LNG=&C21COM=F&I21DBN=BOOK&P21DBN=BOOK&S21CNR=&Z21ID=.
2. Электронная библиотека БрГУ
http://ecat.brstu.ru/catalog .

3. Электронно-библиотечная система «Университетская библиотека online» http://biblioclub.ru .
4. Электронно-библиотечная система «Издательство «Лань» http://e.lanbook.com .
5. Информационная система "Единое окно доступа к образовательным ресурсам" http://window.edu.ru .
6. Научная электронная библиотека eLIBRARY.RU http://elibrary.ru .
7. Университетская информационная система РОССИЯ (УИС РОССИЯ) https://uisrussia.msu.ru/ .
8. Национальная электронная библиотека НЭБ http://xn--90ax2c.xn--p1ai/how-to-search /.

9. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ОБУЧАЮЩИХСЯ ПО ОСВОЕНИЮ ДИСЦИПЛИНЫ

9.1. Методические указания для обучающихся по выполнению лабораторных работ

Лабораторная работа №1
Списки.

Цель занятия: научиться строить однонаправленные, двунаправленные и кольцевые списки.
Задание:
1. По заданному варианту с помощью списков реализовать программу «Считалочка». *N* ребят расположены по кругу. Начав отсчет от первого, удаляют каждого *к*-ого, смыкая при этом круг. Определить порядок удаления ребят из круга. Эту задачу необходимо исследовать для различных значений *N* от 1 до 64, составив таблицу оставшихся ребят (*t* – номер оставшегося ребенка).
2. По заданному варианту построить однонаправленный линейный список абонентов телефонной станции, упорядоченный лексикографически, содержащий ФИО и семизначный номер телефона. Составить процедуры определения:
– по номеру телефона фамилии;
– по фамилии списка номеров телефонов.
3. По заданному варианту построить двунаправленный неупорядоченный список номеров телефонов: семизначных – абонентов; трехзначных – спецслужб. Просмотреть список справа налево и построить упорядоченный однонаправленный список, не
включая в него номера телефонов спецслужб.
Порядок выполнения:
соответствует пунктам 1 – 3 задания.
Форма отчетности:
отчёт сдаётся в печатном виде. В отчёте должны присутствовать:

1. Номер варианта индивидуального задания;
2. Цель работы;
3. Задание;
4. Поэтапное выполнение всех заданий варианта индивидуального задания;
5. Заключение.

Задания для самостоятельной работы:
Предусмотрены индивидуальным заданием обучающегося.
Рекомендации по выполнению заданий и подготовке к лабораторному занятию:
Ознакомиться с теоретическим материалом, представленным в первом разделе данной дисциплины.

Основная литература

1. Мейер, Б. Инструменты, алгоритмы и структуры данных / Б. Мейер. - 2-е изд., испр. - М.: Национальный Открытый Университет «ИНТУИТ», 2016. - 543 с. : схем., ил. - Библиогр. в кн.; то же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=429033.

2. Курносов М.Г. Введение в структуры и алгоритмы обработки данных: учебник / М.Г. Курносов – Новосибирск: Автограф, 2015. – 179 с., то же [Электронный ресурс] http://ecat.brstu.ru/catalog/Ресурсы%20свободного%20доступа/Курносов%20М.Г.%20Введение%20в%20структуры%20и%20алгоритмы%20обработки%20данных.%20Учебник.%202015.pdf.

Дополнительная литература

1. Хусаинов, Б.С. Структуры и алгоритмы обработки данных. Примеры на языке Си: учебное пособие для вузов / Б. С. Хусаинов. - Москва: Финансы и статистика, 2004. – 464 с.
2. Серебряная, Л. В. Структуры и алгоритмы обработки данных: учеб.-метод. пособие / Л. В. Серебряная, И. М. Марина. – Минск : БГУИР, 2013. – 51 с., то же [Электронный ресурс] http://ecat.brstu.ru/catalog/Ресурсы%20свободного%20доступа/Серебряная%20Л.В.%20Структуры%20и%20алгоритмы%20обработки%20данных.%20Учеб.-метод.%20пособие.%202013.pdf.

Контрольные вопросы для самопроверки

1. Дать краткое описание однонаправленных, двунаправленных и кольцевых списков.

Лабораторная работа №2
Типы линейной структуры данных.

Цель занятия: овладеть навыками в программировании линейных структур данных и отладке программ.

Задание:

1. По варианту задано двузначное число, найти:
- число десятков в нем;
- число единиц в нем;
- сумму его цифр;
- произведение его цифр.

Порядок выполнения:

Соответствует пункту задания.

Форма отчетности:

Отчёт сдаётся в печатном виде. В отчёте должны присутствовать:

1. Номер варианта индивидуального задания;
2. Цель работы:
3. Задание;
4. Поэтапное выполнение всех заданий варианта индивидуального задания;
5. Заключение.

Задания для самостоятельной работы:

Предусмотрены индивидуальным заданием обучающегося.

Рекомендации по выполнению заданий и подготовке к лабораторному занятию:

Ознакомиться с теоретическим материалом, представленным во втором разделе данной дисциплины.

Основная литература

1. Мейер, Б. Инструменты, алгоритмы и структуры данных / Б. Мейер. - 2-е изд., испр. - М.: Национальный Открытый Университет «ИНТУИТ», 2016. - 543 с. : схем., ил. - Библиогр. в кн.; то же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=429033.
2. Курносов М.Г. Введение в структуры и алгоритмы обработки данных: учебник / М.Г. Курносов – Новосибирск: Автограф, 2015. – 179 с., то же [Электронный ресурс] http://ecat.brstu.ru/catalog/Ресурсы%20свободного%20доступа/Курносов%20М.Г.%20Введение%20в%20структуры%20и%20алгоритмы%20обработки%20данных.%20Учебник.%202015.pdf.

Дополнительная литература

1. Хусаинов, Б.С. Структуры и алгоритмы обработки данных. Примеры на языке Си: учебное пособие для вузов / Б. С. Хусаинов. - Москва: Финансы и статистика, 2004. – 464 с.
2. Серебряная, Л. В. Структуры и алгоритмы обработки данных: учеб.-метод. пособие / Л. В. Серебряная, И. М. Марина. – Минск : БГУИР, 2013. – 51 с., то же [Электронный ресурс]

http://ecat.brstu.ru/catalog/Ресурсы%20свободного%20доступа/Серебряная%20Л.В.%20Структуры%20и%20алгоритмы%20обработки%20данных.%20Учеб.-метод.%20пособие.%202013.pdf.

Контрольные вопросы для самопроверки:

1. Какая программа называется линейной?
2. Дайте определение термину «функция».
3. Из каких частей состоит функция?
4. Почему необходимо указывать тип используемой переменной при ее описании?
5. Как следует выбирать тип переменных?

Лабораторная работа №3
Алгоритмы сортировки массивов методами выбора, обмена, вставок и челночным методом.

Цель занятия: овладеть навыками сортировки массивов методами выбора, обмена, вставок и челночным методом.

Задание:

1. По варианту задан массив, отсортировать его по возрастанию и убыванию методами:
- выбора;
- обмена;
- вставок;
- челночным.

Порядок выполнения:

Соответствует пункту задания.

Форма отчетности:

Отчёт сдаётся в печатном виде. В отчёте должны присутствовать:

1. Номер варианта индивидуального задания;
2. Цель работы:
3. Задание;
4. Поэтапное выполнение всех заданий варианта индивидуального задания;
5. Заключение.

Задания для самостоятельной работы:

Предусмотрены индивидуальным заданием обучающегося.

Рекомендации по выполнению заданий и подготовке к лабораторному занятию:

Ознакомиться с теоретическим материалом, представленным в третьем разделе данной дисциплины.

Основная литература

1. Мейер, Б. Инструменты, алгоритмы и структуры данных / Б. Мейер. - 2-е изд., испр. - М.: Национальный Открытый Университет «ИНТУИТ», 2016. - 543 с. : схем., ил. - Библиогр. в кн.; то же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=429033.
2. Курносов М.Г. Введение в структуры и алгоритмы обработки данных: учебник / М.Г. Курносов – Новосибирск: Автограф, 2015. – 179 с., то же [Электронный ресурс] http://ecat.brstu.ru/catalog/Ресурсы%20свободного%20доступа/Курносов%20М.Г.%20Введение%20в%20структуры%20и%20алгоритмы%20обработки%20данных.%20Учебник.%202015.pdf.

Дополнительная литература

1. Хусаинов, Б.С. Структуры и алгоритмы обработки данных. Примеры на языке Си: учебное пособие для вузов / Б. С. Хусаинов. - Москва: Финансы и статистика, 2004. – 464 с.
2. Серебряная, Л. В. Структуры и алгоритмы обработки данных: учеб.-метод. пособие / Л. В. Серебряная, И. М. Марина. – Минск : БГУИР, 2013. – 51 с., то же [Электронный ресурс] http://ecat.brstu.ru/catalog/Ресурсы%20свободного%20доступа/Серебряная%20Л.В.%20Структуры%20и%20алгоритмы%20обработки%20данных.%20Учеб.-метод.%20пособие.%202013.pdf.

Контрольные вопросы для самопроверки:

1. Дать краткое описание методов сортировки.

Лабораторная работа №4
Хеширование.

Цель занятия: научиться строить однонаправленные, двунаправленные и кольцевые списки.
Задание:
1. В соответствии с заданным вариантом составить хеш-функцию и проанализировать ее.
2. При необходимости доработать хеш-функцию.
3. Используя полученную хеш-функцию разработать на языке программирования высокого уровня программу, которая должна выполнять следующие функции:
- создавать хеш-таблицу;
- добавлять элементы в хеш-таблицу;
- просматривать хеш-таблицу;
- искать элементы в хеш-таблице;
- удалять элементы из хеш-таблицы;
- строить диаграмму, анализ которой показывает, как распределены коллизии.

Порядок выполнения:
соответствует пунктам 1 – 3 задания.
Форма отчетности:
отчёт сдаётся в печатном виде. В отчёте должны присутствовать:

1. Номер варианта индивидуального задания;
2. Цель работы;
3. Задание;
4. Поэтапное выполнение всех заданий варианта индивидуального задания;
5. Заключение.

Задания для самостоятельной работы:
Предусмотрены индивидуальным заданием обучающегося.
Рекомендации по выполнению заданий и подготовке к лабораторному занятию:
Ознакомиться с теоретическим материалом, представленным в четвертом разделе данной дисциплины.

Основная литература

1. Мейер, Б. Инструменты, алгоритмы и структуры данных / Б. Мейер. - 2-е изд., испр. - М.: Национальный Открытый Университет «ИНТУИТ», 2016. - 543 с. : схем., ил. - Библиогр. в кн.; то же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=429033.
2. Курносов М.Г. Введение в структуры и алгоритмы обработки данных: учебник / М.Г. Курносов – Новосибирск: Автограф, 2015. – 179 с., то же [Электронный ресурс] http://ecat.brstu.ru/catalog/Ресурсы%20свободного%20доступа/Курносов%20М.Г.%20Введение%20в%20структуры%20и%20алгоритмы%20обработки%20данных.%20Учебник.%202015.pdf.

Дополнительная литература

1. Хусаинов, Б.С. Структуры и алгоритмы обработки данных. Примеры на языке Си: учебное пособие для вузов / Б. С. Хусаинов. - Москва: Финансы и статистика, 2004. – 464 с.
2. Серебряная, Л. В. Структуры и алгоритмы обработки данных: учеб.-метод. пособие / Л. В. Серебряная, И. М. Марина. – Минск : БГУИР, 2013. – 51 с., то же [Электронный ресурс] http://ecat.brstu.ru/catalog/Ресурсы%20свободного%20доступа/Серебряная%20Л.В.%20Структуры%20и%20алгоритмы%20обработки%20данных.%20Учеб.-метод.%20пособие.%202013.pdf.

Контрольные вопросы для самопроверки
1. В чем заключается метод поиска - хеширование?
2. Перечислите основные хеш - функции?
3. В чем суть каждой хеш - функции?
4. Что такое коллизия?
5. Какие методы для разрешения коллизий вы знаете?
6. В чем заключается суть каждого метода для разрешения коллизии?

Лабораторная работа №5
Файлы.

Цель занятия: овладеть навыками работы с внутренними и внешними файлами.
Задание:
1. В соответствии с вариантом задания написать программу на ЭВМ отображающую работу с файлами.
2. Прочитать и напечатать созданный файл.
Порядок выполнения:
Соответствует пункту 1 – 2 задания.
Форма отчетности:
Отчёт сдаётся в печатном виде. В отчёте должны присутствовать:

1. Номер варианта индивидуального задания;
2. Цель работы:
3. Задание;
4. Поэтапное выполнение всех заданий варианта индивидуального задания;
5. Заключение.

Задания для самостоятельной работы:
Предусмотрены индивидуальным заданием обучающегося.
Рекомендации по выполнению заданий и подготовке к лабораторному занятию:
Ознакомиться с теоретическим материалом, представленным в пятом разделе данной дисциплины.

Основная литература

1. Мейер, Б. Инструменты, алгоритмы и структуры данных / Б. Мейер. - 2-е изд., испр. - М.: Национальный Открытый Университет «ИНТУИТ», 2016. - 543 с. : схем., ил. - Библиогр. в кн.; то же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=429033.
2. Курносов М.Г. Введение в структуры и алгоритмы обработки данных: учебник / М.Г. Курносов – Новосибирск: Автограф, 2015. – 179 с., то же [Электронный ресурс] http://ecat.brstu.ru/catalog/Ресурсы%20свободного%20доступа/Курносов%20М.Г.%20Введение%20в%20структуры%20и%20алгоритмы%20обработки%20данных.%20Учебник.%202015.pdf.

Дополнительная литература

1. Хусаинов, Б.С. Структуры и алгоритмы обработки данных. Примеры на языке Си: учебное пособие для вузов / Б. С. Хусаинов. - Москва: Финансы и статистика, 2004. – 464 с.
2. Серебряная, Л. В. Структуры и алгоритмы обработки данных: учеб.-метод. пособие / Л. В. Серебряная, И. М. Марина. – Минск : БГУИР, 2013. – 51 с., то же [Электронный ресурс] http://ecat.brstu.ru/catalog/Ресурсы%20свободного%20доступа/Серебряная%20Л.В.%20Структуры%20и%20алгоритмы%20обработки%20данных.%20Учеб.-метод.%20пособие.%202013.pdf.

Контрольные вопросы для самопроверки:
1. Объяснить,что означают следующие термины: файл, запись, метод доступа, структура записи?
2. Каково назначение операторов открытия и закрытия файлов в языке программирования Object Pascal?
3. Допустимы ли различные типы данных для элементов одной записи?
4. Указать, с помощью каких операторов осуществляется запись данных в файл последовательного доступа, чтение из файла?
5. Привести примеры использования файлов последовательного доступа.
6. Как распознать конец файла данных? Как распознать файл на диске?

Лабораторная работа №6 Деревья.

Цель занятия: исследовать и изучить процедуры, используемые при работе с деревьями.
Задание:
1. Построить дерево в соответствии с заданным вариантом задания.
2. Вывести его на экран в виде дерева.

3. Реализовать основные операции работы с деревом: обход дерева, включение, исключение и поиск узлов.
4. Сравнить эффективность алгоритмов сортировки и поиска в зависимости от высоты дерева и степени его ветвления.
Порядок выполнения:
Соответствует пунктам 1 – 4 задания.
Форма отчетности:
Отчёт сдаётся в печатном виде. В отчёте должны присутствовать:

1. Номер варианта индивидуального задания;
2. Цель работы:
3. Задание;
4. Поэтапное выполнение всех заданий варианта индивидуального задания;
5. Заключение.

Задания для самостоятельной работы:
Предусмотрены индивидуальным заданием обучающегося.
Рекомендации по выполнению заданий и подготовке к лабораторному занятию:
Ознакомиться с теоретическим материалом, представленным в шестом разделе данной дисциплины.

Основная литература

1. Мейер, Б. Инструменты, алгоритмы и структуры данных / Б. Мейер. - 2-е изд., испр. - М.: Национальный Открытый Университет «ИНТУИТ», 2016. - 543 с. : схем., ил. - Библиогр. в кн.; то же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=429033.
2. Курносов М.Г. Введение в структуры и алгоритмы обработки данных: учебник / М.Г. Курносов – Новосибирск: Автограф, 2015. – 179 с., то же [Электронный ресурс] http://ecat.brstu.ru/catalog/Ресурсы%20свободного%20доступа/Курносов%20М.Г.%20Введение%20в%20структуры%20и%20алгоритмы%20обработки%20данных.%20Учебник.%202015.pdf.

Дополнительная литература

1. Хусаинов, Б.С. Структуры и алгоритмы обработки данных. Примеры на языке Си: учебное пособие для вузов / Б. С. Хусаинов. - Москва: Финансы и статистика, 2004. – 464 с.
2. Серебряная, Л. В. Структуры и алгоритмы обработки данных: учеб.-метод. пособие / Л. В. Серебряная, И. М. Марина. – Минск : БГУИР, 2013. – 51 с., то же [Электронный ресурс] http://ecat.brstu.ru/catalog/Ресурсы%20свободного%20доступа/Серебряная%20Л.В.%20Структуры%20и%20алгоритмы%20обработки%20данных.%20Учеб.-метод.%20пособие.%202013.pdf.

Контрольные вопросы для самопроверки:
1. Что такое дерево?
2. Как выделяется память под представление деревьев?
3.Какие бывают типы деревьев?
4. Какие стандартные операции возможны над деревьями?

Лабораторная работа №7. Бинарные деревья.

Цель занятия: исследовать и изучить процедуры, используемые при работе с бинарными (двоичными) деревьями.
Задание:
1. По варианту задано арифметическое выражение. Необходимо:
- построить для него бинарное дерево;
- получить запись арифметического выражения в префиксной и постфиксной форме.
2. Для заданной последовательности целых чисел построить бинарное дерево поиска. Реализовать алгоритмы для работы с деревом поиска, считая, что дерево имеет связное представление:
- построение дерева;
- включение узла;
- печать дерева в наглядной форме;

- сортировка последовательности по возрастанию и убыванию;
- поиск элемента в дереве;
- удаление дерева.

Порядок выполнения:

Соответствует пунктам 1 – 2 задания.

Форма отчетности:

Отчёт сдаётся в печатном виде. В отчёте должны присутствовать:

1. Номер варианта индивидуального задания;
2. Цель работы:
3. Задание;
4. Поэтапное выполнение всех заданий варианта индивидуального задания;
5. Заключение.

Задания для самостоятельной работы:

Предусмотрены индивидуальным заданием обучающегося.

Рекомендации по выполнению заданий и подготовке к лабораторному занятию:

Ознакомиться с теоретическим материалом, представленным в шестом разделе данной дисциплины.

Основная литература

1. Мейер, Б. Инструменты, алгоритмы и структуры данных / Б. Мейер. - 2-е изд., испр. - М.: Национальный Открытый Университет «ИНТУИТ», 2016. - 543 с. : схем., ил. - Библиогр. в кн.; то же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=429033.
2. Курносов М.Г. Введение в структуры и алгоритмы обработки данных: учебник / М.Г. Курносов – Новосибирск: Автограф, 2015. – 179 с., то же [Электронный ресурс] http://ecat.brstu.ru/catalog/Ресурсы%20свободного%20доступа/Курносов%20М.Г.%20Введение%20в%20структуры%20и%20алгоритмы%20обработки%20данных.%20Учебник.%202015.pdf.

Дополнительная литература

1. Хусаинов, Б.С. Структуры и алгоритмы обработки данных. Примеры на языке Си: учебное пособие для вузов / Б. С. Хусаинов. - Москва: Финансы и статистика, 2004. – 464 с.
2. Серебряная, Л. В. Структуры и алгоритмы обработки данных: учеб.-метод. пособие / Л. В. Серебряная, И. М. Марина. – Минск : БГУИР, 2013. – 51 с., то же [Электронный ресурс] http://ecat.brstu.ru/catalog/Ресурсы%20свободного%20доступа/Серебряная%20Л.В.%20Структуры%20и%20алгоритмы%20обработки%20данных.%20Учеб.-метод.%20пособие.%202013.pdf.

Контрольные вопросы для самопроверки:

1. Что такое двоичное дерево?
2. Как выделяется память под представлении двоичных деревьев?
3.Какие бывают типы двоичных деревьев?
4. Какие стандартные операции возможны над двоичными деревьями?

Лабораторная работа №8
Пирамиды.

Цель занятия: исследовать и изучить процедуры, используемые при работе пирамидами.

Задание:

1. По варианту задан массив. Необходимо:
- преобразовать массив в пирамиду;
- вставить элемент в пирамиду;
- удалить элемент из пирамиды;
- выполнить пирамидальную сортировку.

Порядок выполнения:

Соответствует пункту задания.

Форма отчетности:

Отчёт сдаётся в печатном виде. В отчёте должны присутствовать:

1. Номер варианта индивидуального задания;

2. Цель работы:
3. Задание;
4. Поэтапное выполнение всех заданий варианта индивидуального задания;
5. Заключение.

Задания для самостоятельной работы:

Предусмотрены индивидуальным заданием обучающегося.

Рекомендации по выполнению заданий и подготовке к лабораторному занятию:

Ознакомиться с теоретическим материалом, представленным в седьмом разделе данной дисциплины.

Основная литература

1. Мейер, Б. Инструменты, алгоритмы и структуры данных / Б. Мейер. - 2-е изд., испр. - М.: Национальный Открытый Университет «ИНТУИТ», 2016. - 543 с. : схем., ил. - Библиогр. в кн.; то же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=429033.
2. Курносов М.Г. Введение в структуры и алгоритмы обработки данных: учебник / М.Г. Курносов – Новосибирск: Автограф, 2015. – 179 с., то же [Электронный ресурс] http://ecat.brstu.ru/catalog/Ресурсы%20свободного%20доступа/Курносов%20М.Г.%20Введение%20в%20структуры%20и%20алгоритмы%20обработки%20данных.%20Учебник.%202015.pdf.

Дополнительная литература

1. Хусаинов, Б.С. Структуры и алгоритмы обработки данных. Примеры на языке Си: учебное пособие для вузов / Б. С. Хусаинов. - Москва: Финансы и статистика, 2004. – 464 с.
2. Серебряная, Л. В. Структуры и алгоритмы обработки данных: учеб.-метод. пособие / Л. В. Серебряная, И. М. Марина. – Минск : БГУИР, 2013. – 51 с., то же [Электронный ресурс] http://ecat.brstu.ru/catalog/Ресурсы%20свободного%20доступа/Серебряная%20Л.В.%20Структуры%20и%20алгоритмы%20обработки%20данных.%20Учеб.-метод.%20пособие.%202013.pdf.

Контрольные вопросы для самопроверки:

1. Что такое пирамида?
2. Дайте краткое описание пирамидальной сортировке.

Лабораторная работа №9
Графы.

Цель занятия: научиться представлять и анализировать ориентированные графы, определять в них центр и находить заданные маршруты.

Задание:

1. Создать помеченный орграф с помощью матрицы смежности или списков смежности.
2. Указать вершину-источник и результирующую вершину на графе.
3. Найти кратчайший путь между заданными вершинами.
4. Найти самый длинный путь между заданными вершинами.
5. Найти все пути между заданными вершинами, упорядочив их по возрастанию.
6. Найти центр орграфа.

Порядок выполнения:

Соответствует пунктам 1 – 6 задания.

Форма отчетности:

Отчёт сдаётся в печатном виде. В отчёте должны присутствовать:

1. Номер варианта индивидуального задания;
2. Цель работы:
3. Задание;
4. Поэтапное выполнение всех заданий варианта индивидуального задания;
5. Заключение.

Задания для самостоятельной работы:

Предусмотрены индивидуальным заданием обучающегося.

Рекомендации по выполнению заданий и подготовке к лабораторному занятию:

Ознакомиться с теоретическим материалом, представленным в седьмом разделе данной

дисциплины.

Основная литература

1. Мейер, Б. Инструменты, алгоритмы и структуры данных / Б. Мейер. - 2-е изд., испр. - М.: Национальный Открытый Университет «ИНТУИТ», 2016. - 543 с. : схем., ил. - Библиогр. в кн.; то же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=429033.
2. Курносов М.Г. Введение в структуры и алгоритмы обработки данных: учебник / М.Г. Курносов – Новосибирск: Автограф, 2015. – 179 с., то же [Электронный ресурс] http://ecat.brstu.ru/catalog/Ресурсы%20свободного%20доступа/Курносов%20М.Г.%20Введение%20в%20структуры%20и%20алгоритмы%20обработки%20данных.%20Учебник.%202015.pdf.

Дополнительная литература

1. Хусаинов, Б.С. Структуры и алгоритмы обработки данных. Примеры на языке Си: учебное пособие для вузов / Б. С. Хусаинов. - Москва: Финансы и статистика, 2004. – 464 с.
2. Серебряная, Л. В. Структуры и алгоритмы обработки данных: учеб.-метод. пособие / Л. В. Серебряная, И. М. Марина. – Минск : БГУИР, 2013. – 51 с., то же [Электронный ресурс] http://ecat.brstu.ru/catalog/Ресурсы%20свободного%20доступа/Серебряная%20Л.В.%20Структуры%20и%20алгоритмы%20обработки%20данных.%20Учеб.-метод.%20пособие.%202013.pdf.

Контрольные вопросы для самопроверки:

1. Дать определение графа.
2. Дать определение основного дерева графа.
3. Перечислить способы представления графа для решения задач с помощью компьютера.

9.2. Методические указания по выполнению контрольной работы

Контрольная работа посвящена сортировки массивов различными методами. Задание включает в себя следующие разделы:

1. Сортировка массивов методом выбора.
2. Сортировка массивов методом обмена.
3. Сортировка массивов челночным методом.
4. Сортировка массивов методом вставок.

Расчет производиться каждым студентом индивидуально, по вариантам.

10. ПЕРЕЧЕНЬ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, ИСПОЛЬЗУЕМЫХ ПРИ ОСУЩЕСТВЛЕНИИ ОБРАЗОВАТЕЛЬНОГО ПРОЦЕССА ПО ДИСЦИПЛИНЕ

Информационно-коммуникационные технологии (ИКТ) – преподаватель использует для:

- получения информации при подготовке к занятиям,
- создания презентационного сопровождения лекций;
- интерактивного общения;
- ОС Windows 7 Professional;
- Microsoft Office 2007 Russian Academic OPEN NO Level;
- Антивирусное программное обеспечение Kaspersky Security;
- ПО “Антиплагиат”.

11. ОПИСАНИЕ МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЙ БАЗЫ, НЕОБХОДИМОЙ ДЛЯ ОСУЩЕСТВЛЕНИЯ ОБРАЗОВАТЕЛЬНОГО ПРОЦЕССА ПО ДИСЦИПЛИНЕ

Вид занятия (Лк, ЛР)	*Наименование аудитории*	*Перечень основного оборудования*	*№ Лк, ЛР*
1	3	4	5
Лк	Лекционная аудитория	-	Лк 1-7
ЛР	Дисплейные классы	Персональные компьютеры	ЛР 1-9
КР	Дисплейные классы	Персональные компьютеры	-
СРС	ЧЗЗ	-	-

Приложение 1

ФОНД ОЦЕНОЧНЫХ СРЕДСТВ ДЛЯ ПРОВЕДЕНИЯ ПРОМЕЖУТОЧНОЙ АТТЕСТАЦИИ ОБУЧАЮЩИХСЯ ПО ДИСЦИПЛИНЕ

1. Описание фонда оценочных средств (паспорт)

№ компетенции	Элемент компетенции	Раздел	Тема	ФОС
ОК-7	способность к самоорганизации и самообразованию	**1.** Классификация структур данных.	**1.1** Массивы, записи, множества, динамические структуры данных. **1.2** Списки.	Экзаменац. билет
		2. Типы данных линейной структуры.	**2.1** Данные элементарных типов. **2.2** Данные числового, вещественного, символьного, логического типов. **2.3** Данные типа указатель.	Экзаменац. билет
		3. Алгоритмы сортировки массивов.	**3.1** Сортировка массивов методом выбора. **3.2** Сортировка методом обмена. **3.3** Сортировка челночным методом. **3.4** Сортировка методом вставок.	Экзаменац. билет
		4. Хеширование.	**4.1** Хеш-функции. **4.2** Методы разрешения коллизий.	Экзаменац. билет
		5. Файлы.	**5.1** Внешние файлы. **5.2** Внутренние файлы.	Экзаменац. билет
		6. Типы данных нелинейной структуры.	**6.1** Деревья. **6.2** Бинарные деревья. **6.3** Пирамиды.	Экзаменац. билет
		7. Графы.	**7.1** Основные понятия теории графов. **7.2** Основные процедуры работы с графами. **7.3** Алгоритмы на графах: поиск в глубину и поиск в ширину.	Экзаменац. билет
ОПК-6	способность осуществлять поиск, хранение, обработку и анализ информации из различных источников и баз	**1.** Классификация структур данных.	**1.1** Массивы, записи, множества, динамические структуры данных. **1.2** Списки.	Экзаменац. билет
		2. Типы данных линейной	**2.1** Данные элементарных типов.	Экзаменац. билет

	данных, представлять ее в требуемом формате с использованием информационных, компьютерных и сетевых технологий	структуры.	**2.2** Данные числового, вещественного, символьного, логического типов. **2.3** Данные типа указатель.	
		3. Алгоритмы сортировки массивов.	**3.1** Сортировка массивов методом выбора. **3.2** Сортировка методом обмена. **3.3** Сортировка челночным методом. **3.4** Сортировка методом вставок.	Экзаменац. билет
		4. Хеширование.	**4.1** Хеш-функции. **4.2** Методы разрешения коллизий.	Экзаменац. билет
		5. Файлы.	**5.1** Внешние файлы. **5.2** Внутренние файлы.	Экзаменац. билет
		6. Типы данных нелинейной структуры.	**6.1** Деревья. **6.2** Бинарные деревья. **6.3** Пирамиды.	Экзаменац. билет
		7. Графы.	**7.1** Основные понятия теории графов. **7.2** Основные процедуры работы с графами. **7.3** Алгоритмы на графах: поиск в глубину и поиск в ширину.	Экзаменац. билет
ПК-5	способность осуществлять сбор и анализ исходных данных для расчета и проектирования систем и средств автоматизации и управления	**1.** Классификация структур данных.	**1.1** Массивы, записи, множества, динамические структуры данных. **1.2** Списки.	Экзаменац. билет
		2. Типы данных линейной структуры.	**2.1** Данные элементарных типов. **2.2** Данные числового, вещественного, символьного, логического типов. **2.3** Данные типа указатель.	Экзаменац. билет
		3. Алгоритмы сортировки массивов.	**3.1** Сортировка массивов методом выбора. **3.2** Сортировка методом обмена. **3.3** Сортировка челночным методом. **3.4** Сортировка методом вставок.	Экзаменац. билет
		4. Хеширование.	**4.1** Хеш-функции. **4.2** Методы разрешения коллизий.	Экзаменац. билет
		5. Файлы.	**5.1** Внешние файлы.	Экзаменац.

			5.2 Внутренние файлы.	билет
		6. Типы данных нелинейной структуры.	**6.1** Деревья. **6.2** Бинарные деревья. **6.3** Пирамиды.	Экзаменац. билет
		7. Графы.	**7.1** Основные понятия теории графов. **7.2** Основные процедуры работы с графами. **7.3** Алгоритмы на графах: поиск в глубину и поиск в ширину.	Экзаменац. билет

Экзаменационные вопросы

№ п/п	Компетенции		ЭКЗАМЕНАЦИОННЫЕ ВОПРОСЫ	№ и наименование раздела
	Код	Определение		
1	2	3	4	5
1.	ОК-7	способность к самоорганизации и самообразованию	**1.1** Массивы, записи, множества, динамические структуры данных.	**1.** Классификация структур данных.
			1.2 Списки.	
			2.1 Данные элементарных типов.	**2.** Типы данных линейной структуры.
			2.2 Данные числового, вещественного, символьного, логического типов.	
			2.3 Данные типа указатель.	
			3.1 Определение опорного решения задачи методом северо-западного угла.	**3.** Алгоритмы сортировки массивов.
			3.2 Определение опорного решения задачи методом наименьшего элемента.	
			3.3 Определение оптимального решения задачи методом потенциалов.	
			3.4 Сортировка методом вставок.	
			4.1 Хеш-функции	**4.** Хеширование.
			4.2 Методы разрешения коллизий.	
			5.1 Внешние файлы.	**5.** Файлы.
			5.2 Внутренние файлы.	
			6.1 Деревья.	**6.** Типы данных нелинейной структуры.
			6.2 Бинарные деревья.	
			6.3 Пирамиды.	
			7.1 Основные понятия теории графов.	**7.** Графы.
			7.2 Основные процедуры работы с графами.	
			7.3 Алгоритмы на графах: поиск в глубину и поиск в ширину.	
2.	ОПК-6	способность осуществлять поиск, хранение,	**1.1** Массивы, записи, множества, динамические структуры данных.	**1.** Классификация структур данных.
			1.2 Списки.	

		обработку и анализ информации из различных источников и баз данных, представлять ее в требуемом формате с использованием информационных, компьютерных и сетевых технологий	**2.1** Данные элементарных типов.	**2.** Типы данных линейной структуры.
			2.2 Данные числового, вещественного, символьного, логического типов.	
			2.3 Данные типа указатель.	
			3.1 Определение опорного решения задачи методом северо-западного угла.	**3.** Алгоритмы сортировки массивов.
			3.2 Определение опорного решения задачи методом наименьшего элемента.	
			3.3 Определение оптимального решения задачи методом потенциалов.	
			3.4 Сортировка методом вставок.	
			4.1 Хеш-функции	**4.** Хеширование.
			4.2 Методы разрешения коллизий.	
			5.1 Внешние файлы.	**5.** Файлы.
			5.2 Внутренние файлы.	
			6.1 Деревья.	**6.** Типы данных нелинейной структуры.
			6.2 Бинарные деревья.	
			6.3 Пирамиды.	
			7.1 Основные понятия теории графов.	**7.** Графы.
			7.2 Основные процедуры работы с графами.	
			7.3 Алгоритмы на графах: поиск в глубину и поиск в ширину.	
3.	ПК-5	способность осуществлять сбор и анализ исходных данных для расчета и проектирования систем и средств автоматизации и управления	**1.1** Массивы, записи, множества, динамические структуры данных.	**1.** Классификация структур данных.
			1.2 Списки.	
			2.1 Данные элементарных типов.	**2.** Типы данных линейной структуры.
			2.2 Данные числового, вещественного, символьного, логического типов.	
			2.3 Данные типа указатель.	
			3.1 Определение опорного решения задачи методом северо-западного угла.	**3.** Алгоритмы сортировки массивов.
			3.2 Определение опорного решения задачи методом наименьшего элемента.	
			3.3 Определение оптимального решения задачи методом потенциалов.	
			3.4 Сортировка методом вставок.	
			4.1 Хеш-функции	**4.** Хеширование.
			4.2 Методы разрешения коллизий.	
			5.1 Внешние файлы.	**5.** Файлы.
			5.2 Внутренние файлы.	
			6.1 Деревья.	**6.** Типы данных нелинейной структуры.
			6.2 Бинарные деревья.	
			6.3 Пирамиды.	

			7.1 Основные понятия теории графов.	**7.** Графы.
			7.2 Основные процедуры работы с графами.	
			7.3 Алгоритмы на графах: поиск в глубину и поиск в ширину.	

3. Описание показателей и критериев оценивания компетенций

Показатели	Оценка	Критерии
Знать (ОК-7): - основные методы разработки алгоритмов и программ, структуры данных, используемые для представления типовых информационных объектов. (ОПК-6): - современные типовые алгоритмы обработки данных. (ПК-5): - современные средства автоматизации и управления **Уметь** (ОК-7): - использовать стандартные пакеты прикладных программ для решения практических задач. (ОПК-6): - представлять информацию в требуемом формате с использованием информационных, компьютерных и сетевых технологий. (ПК-5): - анализировать исходные данные для расчета систем. **Владеть** (ОК-7): - навыками самоорганизации и самообразования. (ОПК-6): - методами построения современных проблемно-ориентированных прикладных программных	**отлично**	Во время ответа обучающийся демонстрирует глубокое и прочное усвоение программного материала: знает основные методы разработки алгоритмов и программ, структуры данных, используемые для представления типовых информационных объектов, современные типовые алгоритмы обработки данных; умеет использовать стандартные пакеты прикладных программ для решения практических задач, представлять информацию в требуемом формате с использованием информационных, компьютерных и сетевых технологий; владеет навыками самоорганизации и самообразования, методами построения современных проблемно-ориентированных прикладных программных средств.
	хорошо	Ответ содержит неточности. Дополнительные вопросы требуется, но обучающийся с ними справляется отлично.
	удовлетворительно	Ответил только на один вопрос, либо слабо ответил на оба вопроса. На дополнительные вопросы отвечает неуверенно.
	неудовлетворительно	На оба вопроса обучающийся отвечает неубедительно. На дополнительные вопросы преподавателя также не может ответить.

средств. (ПК-5): - методами сбора информации.		

4. Методические материалы, определяющие процедуры оценивания знаний, умений, навыков и опыта деятельности

Дисциплина «Структуры и алгоритмы обработки данных» направлена на изучение основ математического моделирования, построения математических моделей, анализа теоретических и экспериментальных исследований в сфере профессиональной деятельности.

Изучение дисциплины «Структуры и алгоритмы обработки данных» предусматривает:

- лекции;
- лабораторные работы;
- контрольную работу;
- самостоятельную работу обучающихся;
- экзамен.

В ходе освоения раздела 1 «Классификация структур данных» обучающиеся должны изучить основные положения и определения массивов, записей, множеств, динамических структур данных, знать основные виды списков.

В ходе освоения раздела 2 «Типы данных линейной структуры» обучающиеся должны изучить основные типы данных.

В ходе освоения раздела 3 «Алгоритмы сортировки массивов» обучающиеся должны уметь сортировать массивы методами выбора, обмена, вставок и челночным методом.

В ходе освоения 4 раздела «Хеширование» обучающиеся должны знать основные Хеш-функции, а также методы разрешения коллизий.

В ходе освоения 5 раздела «Файлы» обучающиеся должны овладеть навыками работы с внутренними и внешними файлами.

В ходе освоения 6 раздела «Типы данных нелинейной структуры» обучающиеся должны исследовать и изучить процедуры, используемые при работе с деревьями, бинарными деревьями и пирамидами.

В ходе освоения 7 раздела «Графы» обучающиеся должны знать основные понятия теории графов, основные процедуры работы с графами, изучить основные алгоритмы на графах.

В процессе выполнения лабораторных работ происходит закрепление знаний, формирование умений и навыков работы со структурами и алгоритмами обработки данных.

Работа с литературой является важнейшим элементом в получении знаний по дисциплине. Прежде всего, необходимо воспользоваться списком рекомендуемой по данной дисциплине литературой. Дополнительные сведения по изучаемым темам можно найти в периодической печати и Интернете.

К экзамену допускаются студенты, которые выполнили и оформили все лабораторные работы, сдали и защитили контрольную работу.

Оценка знаний, умений, навыков осуществляется в процессе промежуточной аттестации обучающихся по дисциплине, которая осуществляется в виде зачета и экзамена. Для оценивания знаний, умений, навыков используются ФОС по дисциплине, содержащий, вопросы к зачету и экзамену.

Предусмотрено проведение аудиторных занятий в интерактивной форме в сочетании с внеаудиторной работой.

Приложение 2

АННОТАЦИЯ
рабочей программы дисциплины
Структуры и алгоритмы обработки данных

1. Цель и задачи дисциплины

Целью изучения дисциплины является: формирование у обучающихся знаний базовых классов структур данных и алгоритмов их программной обработки; формирование навыков проектирования эффективных структур и алгоритмов обработки данных при решении практических задач.

Задачей изучения дисциплины является: подготовка обучающихся к самостоятельному решению теоретических и прикладных задач связанных с проектированием эффективных структур и алгоритмов обработки данных.

2. Структура дисциплины

2.1 Распределение трудоемкости по отдельным видам учебных занятий, включая самостоятельную работу: лекций – 34 часов, лабораторные работы – 34 часа, самостоятельная работа студента – 58 часов,

Общая трудоемкость дисциплины составляет 180 часов, 3 зачетные единицы.

2.2 Основные разделы дисциплины:

1. Классификация структур данных.
2. Типы данных линейной структуры.
3. Алгоритмы сортировки массивов.
4. Хеширование.
5. Файлы.
6. Типы данных нелинейной структуры.
7. Графы.

3. Планируемые результаты обучения (перечень компетенций)

Процесс изучения дисциплины направлен на формирование следующих компетенций:

ОК-7 - способность к самоорганизации и самообразованию.

ОПК-6 - способность осуществлять поиск, хранение, обработку и анализ информации из различных источников и баз данных, представлять ее в требуемом формате с использованием информационных, компьютерных и сетевых технологий.

ПК-5 - способность осуществлять сбор и анализ исходных данных для расчета и проектирования систем и средств автоматизации и управления

4. Вид промежуточной аттестации: экзамен.

Приложение 3

Протокол о дополнениях и изменениях в рабочей программе
на 20___-20___ учебный год

1. В рабочую программу по дисциплине вносятся следующие дополнения:

_____
_____
_____

2. В рабочую программу по дисциплине вносятся следующие изменения:

_____
_____
_____

Протокол заседания кафедры №____ от «___» _____ 20 __ __ г.,
(разработчик)

Заведующий кафедрой _____ _____
(подпись) *(Ф.И.О.)*