Энгельсский технологический институт (филиал) федерального государственного
бюджетного образовательного учреждения
высшего образования
«Саратовский государственный технический университет имени Гагарина Ю.А.»

Кафедра «Естественные и математические науки»

Оценочные материалы по дисциплине

Б.1.3.1.1 «Структуры и алгоритмы обработки данных»

09.03.04 «Программная инженерия»
профиль «Управление разработкой программных проектов»

Энгельс 2023

1. Перечень компетенций и уровни их сформированности по дисциплинам (модулям), практикам в процессе освоения ОПОП ВО

В процессе освоения образовательной программы у обучающегося в ходе изучения дисциплины «Структуры и алгоритмы обработки данных» должны сформироваться компетенции: ПК-2.

Критерии определения сформированности компетенций на различных уровнях их формирования

Индекс компетенции	Содержание компетенции
ПК-2	Способен применять методы проектирования и разрабатывать сопровождающую документацию на ПО

Код и наименование индикатора достижения компетенции	Виды занятий для формирования компетенции	Оценочные средства для оценки уровня сформированности компетенции
ИД-4 $_{\text{ПК-2}}$ Знает структуры и алгоритмы обработки данных и анализирует их для проектирования ПО	Лекции, практические занятия, самостоятельная работа	Задания для практических работ. Вопросы к зачету Контрольная работа для студентов заочной формы обучения

Уровни освоения компетенции

Уровень освоения компетенции	Критерии оценивания
Продвинутый (отлично)	Знает: технологию разработки алгоритмов и программ, методы отладки и решения задач на ЭВМ в различных режимах на продвинутом уровне. Умеет: ставить задачу и разрабатывать алгоритм ее решения, использовать прикладные системы программирования, разрабатывать основные программные документы на продвинутом уровне. Владеет: языками процедурного и объектно-ориентированного программирования, навыками разработки и отладки программ не менее, чем на одном из процедурных языков программирования высокого уровня на продвинутом уровне.
Повышенный (хорошо)	Знает: технологию разработки алгоритмов и программ, методы решения задач на ЭВМ среднего уровня сложности. Умеет: разрабатывать алгоритм решения задачи, использовать прикладные системы программирования, разрабатывать основные программные документы не в полном объеме. Владеет: языками процедурного и объектно-ориентированного программирования, но не в полной мере навыками разработки программ на одном из процедурных языков программирования высокого уровня.

Пороговый (базовый) (удовлетворительно)	Знает: на базовом уровне технологию разработки алгоритмов и программ, методы решения простых задач на ЭВМ. Умеет: разрабатывать алгоритм решения простых задач, допуская ошибки использовать прикладные системы программирования, разрабатывать основные программные документы. Владеет: языками процедурного и объектно-ориентированного программирования, на базовом уровне навыками разработки простых программ не менее, чем на одном из процедурных языков программирования высокого уровня.

2. Методические, оценочные материалы и средства, определяющие процедуры оценивания сформированности компетенций (элементов компетенций) в процессе освоения ОПОП ВО

2.1 Оценочные средства для текущего контроля

Вопросы для устного опроса

Тема 1. Массивы данных

1. Назовите известные вам методы сортировки.
2. Какие методы сортировки могут быть предпочтительнее метода пузырька и почему?
3. Что такое сортировка вставками и как она работает?
4. Как реализовать сортировку выбором?
5. Какие особенности и преимущества имеет сортировка методом Шелла по сравнению с другими методами сортировки?
6. Что такое опорный элемент в быстрой сортировке и как он выбирается?
7. Как можно выбрать опорный элемент в быстрой сортировке для достижения лучшей производительности?
8. Какая роль рекурсии в быстрой сортировке и как можно избежать проблем с переполнением стека вызовов?
9. Предположим, что необходимо отсортировать список элементов, состоящий из уже упорядоченного списка, который следует за несколькими «случайными» элементами. Какой из рассмотренных в теоретических сведениях, или изученных вами самостоятельно методов сортировки наиболее подходит для решения этой задачи?
10. Алгоритм сортировки называется устойчивым, если он сохраняет исходный порядок следования элементов с одинаковыми значениями ключей. Какие из рассмотренных в теоретических сведениях, или изученных вами самостоятельно методов сортировки являются устойчивыми?
11. Что такое последовательный поиск и как он работает?
12. Как реализовать последовательный поиск в C#? Предоставьте пример кода.
13. Какая сложность времени у последовательного поиска в лучшем, среднем и худшем случаях?
14. Какие дополнительные меры можно принять для оптимизации последовательного поиска?
15. Какие альтернативные методы поиска существуют и почему они могут быть предпочтительнее последовательного поиска?
16. Как обрабатывать повторяющиеся элементы в последовательном поиске?
17. Какие особенности следует учитывать при использовании последовательного поиска в больших массивах данных?
18. Какие факторы могут повлиять на производительность последовательного поиска?
19. Какие ситуации могут потребовать использования последовательного поиска вместо других методов?
20. Какие структуры данных могут быть эффективными для последовательного поиска?
21. Что такое двоичный поиск и как он работает?
22. Как реализовать двоичный поиск в C#? Предоставьте пример кода.
23. Какая сложность времени у двоичного поиска в лучшем, среднем и худшем случаях?

24. Какие дополнительные меры можно принять для оптимизации двоичного поиска?
25. Какие альтернативные методы поиска существуют и почему они могут быть предпочтительнее двоичного поиска?
26. Как обрабатывать повторяющиеся элементы в двоичном поиске?
27. Какие особенности следует учитывать при использовании двоичного поиска в больших массивах данных?
28. Какие факторы могут повлиять на производительность двоичного поиска?
29. Какие ситуации могут потребовать использования двоичного поиска вместо других методов?
30. Какие структуры данных могут быть эффективными для двоичного поиска?
31. Что такое оценка реального времени выполнения программы?
32. Как можно измерить время выполнения определенного участка кода в C#?
33. Какие инструменты или классы в C# можно использовать для измерения времени выполнения программы?
34. Какая роль имеет класс Stopwatch из пространства имен System.Diagnostics при оценке времени выполнения программы?
35. Какие факторы могут влиять на время выполнения программы?
36. Как можно оптимизировать время выполнения программы?
37. Какие методы анализа производительности программы существуют в C#?
38. Какие инструменты или подходы можно использовать для сравнения времени выполнения различных алгоритмов или решений?
39. Какие особенности следует учитывать при оценке времени выполнения программы на больших объемах данных?
40. Как можно использовать результаты оценки времени выполнения программы для оптимизации ее работы?

Тема 2. Структуры данных

1. Что такое структура данных "стек"?
2. Какие операции можно выполнять со стеком?
3. Что делает метод Push в стеке?
4. Что делает метод Pop в стеке?
5. Какие примеры практических задач могут быть решены с использованием стеков?
6. Как можно реализовать стек на языке программирования?
7. Какова сложность операций Push и Pop в стеке?
8. Какие альтернативные структуры данных существуют для решения аналогичных задач?
9. Что такое структура данных "очередь"?
10. Какие операции можно выполнять с очередью?
11. Что делает метод Add в очереди?
12. Что делает метод Take в очереди?
13. Какие примеры практических задач могут быть решены с использованием очередей?
14. Как можно реализовать очередь на языке программирования?
15. Какова сложность операций Add и Take в очереди?
16. Какие альтернативные структуры данных существуют для решения аналогичных задач?
17. Что такое однонаправленный список общего вида?
18. Какие операции можно выполнять с однонаправленным списком?
19. Что делает метод Find в однонаправленном списке?
20. Что делает метод Insert в однонаправленном списке?
21. Что делает метод Delete в однонаправленном списке?
22. Как можно реализовать однонаправленный список на языке программирования?
23. Какова сложность операций Find, Insert и Delete в однонаправленном списке?
24. Какие примеры практических задач могут быть решены с использованием однонаправленных списков общего вида?

Тема 3. Алгоритмы обработки данных

1. Что такое дерево в контексте структуры данных?
2. Что такое дерево бинарного поиска и как оно отличается от обычного дерева?

3. Что делает метод Add при добавлении элемента в дерево бинарного поиска?
4. Что делает метод Node.Add при добавлении узла в дерево бинарного поиска?
5. Что означают методы Preorder, Inorder и Postorder при обходе дерева?
6. Что делает метод Node.Preorder при обходе дерева в префиксном порядке?
7. Что делает метод Node.Inorder при обходе дерева в инфиксном порядке?
8. Что делает метод Node.Postorder при обходе дерева в постфиксном порядке?
9. Что делают методы Delete и Node.Delete при удалении элемента из дерева бинарного поиска?
10. Какие примеры практических задач могут быть решены с использованием деревьев бинарного поиска?

Тема 4. Алгоритмы на графах

1. Что такое граф в контексте алгоритмов на графах?
2. Какие способы представления графов существуют?
3. Какие алгоритмы используются для обхода графа?
4. Какие алгоритмы используются для нахождения кратчайших путей в графе?
5. Как можно программно реализовать абстрактный тип данных (АТД) "граф"?
6. Какие практические задачи могут быть решены с использованием графов?

Задания к практическим работам

Практическая работа №1

Массивы данных: Сортировка.

Цель работы: изучить сортировку методом «пузырька», вставками, посредством выбора, методом Шелла и быструю сортировку.

Задание:

Дана матрица размерностью n×n, содержащая целые числа. Отсортировать:

1) каждую строчку матрицы по убыванию элементов методом «пузырька»;
2) каждую строчку матрицы по убыванию элементов методом выбора;
3) каждую строчку матрицы по убыванию элементов методом вставки;
4) каждый столбец матрицы по возрастанию элементов методом Шелла;
5) каждый столбец матрицы по возрастанию элементов алгоритмом быстрой сортировки;
6) каждый столбец матрицы по возрастанию элементов методом «пузырька»;
7) диагонали матрицы, параллельные главной, по убыванию элементов методом выбора;
8) диагонали матрицы, параллельные главной, по убыванию элементов методом вставки;
9) диагонали матрицы, параллельные главной, по убыванию элементов алгоритмомШелла;
10) диагонали матрицы, параллельные главной, по убыванию элементов методом быстрой сортировки;
11) диагонали матрицы, параллельные побочной, по возрастанию элементов методом «пузырька»;
12) диагонали матрицы, параллельные побочной, по возрастанию элементов методомвыбора;
13) диагонали матрицы, параллельные побочной, по возрастанию элементов методомвставки;
14) диагонали матрицы, параллельные побочной, по возрастанию элементов алгоритмомШелла;
15) каждый столбец матрицы с номером 2i по убыванию элементов, а с номером 2i+1 по возрастанию элементов методом быстрой сортировки;
16) каждый столбец матрицы с номером 2i по возрастанию элементов, а с номером 2i+1 поубыванию элементов методом «пузырька»;
17) диагонали матрицы, расположенные выше главной, по убыванию элементов, а диагонали матрицы, расположенные ниже главной, по возрастанию элементов методом выбора;
18) диагонали матрицы, расположенные выше главной, по возрастанию элементов, а диагонали матрицы, расположенные ниже главной, по убыванию методом вставки;

19) диагонали матрицы, расположенные выше побочной, по убыванию элементов, а диагонали матрицы, расположенные ниже побочной, по возрастанию элементов алгоритмом Шелла;
20) диагонали матрицы, расположенные выше побочной, по возрастанию элементов, а диагонали матрицы, расположенные ниже побочной, по убыванию элементов методом быстрой сортировки.

Практическая работа №2

Массивы данных: Поиск. Оценка реального времени выполнения программ.

Цель работы: изучить последовательный и двоичный поиск, научиться оценивать реальное время выполнения программ.

Задание:

1. Напишите программу на C#, которая находит первое вхождение заданного элемента в массиве и возвращает его индекс. Если элемент не найден, программа должна вернуть -1.
2. Напишите программу на C#, которая находит все вхождения заданного элемента в массиве и возвращает список их индексов.
3. Напишите программу на C#, которая находит количество вхождений заданного элемента в массиве.
4. Напишите программу на C#, которая находит максимальный элемент в массиве и возвращает его значение.
5. Напишите программу на C#, которая находит минимальный элемент в массиве и возвращает его значение.
6. Напишите программу на C#, которая находит сумму всех элементов массива.
7. Напишите программу на C#, которая находит среднее арифметическое всех элементов массива.
8. Напишите программу на C#, которая находит наибольшее и наименьшее значение в массиве и возвращает их.
9. Напишите программу на C#, которая находит сумму элементов массива, удовлетворяющих заданному условию.
10. Напишите программу на C#, которая находит количество элементов массива, удовлетворяющих заданному условию.
11. Напишите программу на C#, которая находит первое вхождение заданного элемента в массиве и возвращает его индекс. Если элемент не найден, программа должна вернуть -1.
12. Напишите программу на C#, которая находит все вхождения заданного элемента в массиве и возвращает список их индексов.
13. Напишите программу на C#, которая находит количество вхождений заданного элемента в массиве.
14. Напишите программу на C#, которая находит максимальный элемент в массиве и возвращает его значение.
15. Напишите программу на C#, которая находит минимальный элемент в массиве и возвращает его значение.
16. Напишите программу на C#, которая находит сумму всех элементов массива.
17. Напишите программу на C#, которая находит среднее арифметическое всех элементов массива.
18. Напишите программу на C#, которая находит наибольшее и наименьшее значение в массиве и возвращает их.
19. Напишите программу на C#, которая находит сумму элементов массива, удовлетворяющих заданному условию.
20. Напишите программу на C#, которая находит количество элементов массива, удовлетворяющих заданному условию.
21. Напишите программу на C#, которая находит первое вхождение заданного элемента в массиве и возвращает его индекс. Если элемент не найден, программа должна вернуть -1.
22. Напишите программу на C#, которая находит все вхождения заданного элемента в массиве и возвращает список их индексов.
23. Напишите программу на C#, которая находит количество вхождений заданного элемента в массиве.
24. Напишите программу на C#, которая находит максимальный элемент в массиве и возвращает его значение.
25. Напишите программу на C#, которая находит минимальный элемент в массиве и возвращает его значение.
26. Напишите программу на C#, которая находит сумму всех элементов массива.

27. Напишите программу на C#, которая находит среднее арифметическое всех элементов массива.
28. Напишите программу на C#, которая находит наибольшее и наименьшее значение в массиве и возвращает их.
29. Напишите программу на C#, которая находит сумму элементов массива, удовлетворяющих заданному условию.
30. Напишите программу на C#, которая находит количество элементов массива, удовлетворяющих заданному условию.

Практическая работа №3
Структуры данных: Списки. Стек. Метод Push. Метод Pop. Решение практических задач с использованием стеков.

Цель работы: изучение и практическое применение структур данных списков и стеков, а также методов Push и Pop, чтобы решать практические задачи, которые требуют использования стеков.

Задание:

Задание 1

Замечание

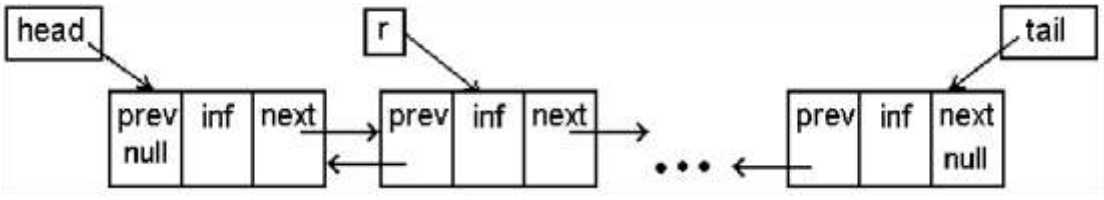
Каждую задачу данного раздела решить, реализовав список в виде линейно связанной структуры: списка общего вида. Исходный и измененный список вывести на экран.

1. На основе файла создать список. Удвоить в нем вхождение каждого четного элемента.
2. На основе файла создать список, включив в него только повторяющиеся числа.
3. На основе файла создать список, включив в него только уникальные числа.
4. На основе файла создать список. Перед каждым элементом равным x вставить элемент равный y.
5. На основе файла создать список. Перед каждым элементом равным x удалить один элемент.
6. На основе файла создать список. Заменить каждую последовательность повторяющихся элементов на один элемент.
7. На основе файла создать список. Найти минимальный элемент, и удалить из списка все элементы равные минимальному.
8. На основе файла создать список. Поменять в списке местами первый максимальный и последний минимальный элементы.
9. На основе файла создать список, вычислив среднее арифметическое значение всех элементов. Удалить из списка элементы, значение которых меньше среднего арифметического всех элементов исходного списка.
10. На основе файла создать список. Найти максимальный элемент, и после каждого максимального элемента в списке вставить элемент, значение которого равно среднему арифметическому значению предыдущих элементов.

*Задание **2***

Мы рассмотрели организацию однонаправленных списков. Однако во многих приложениях возникает необходимость организовывать эффективное перемещение по списку как в прямом, так и в обратном направлении. В этой ситуации можно прибегнуть к организации двунаправленного списка, структура которого изображена ниже:

Предложенная структура содержит ссылки на первый, последний и произвольный элементы



двунаправленного списка. Разработайте класс, реализующий данную модель организации двунаправленного списка, самостоятельно определив необходимые функциональные члены класса.

С помощью разработанного класса решите следующую задачу: дан файл, компонентами которого являются целые числа. На основе файла создайте двунаправленный список так, чтобы элементы заносились в него в порядке возрастания значений. Выведите на экран содержимое списка в прямом и обратном порядке.

*Задание **3***

Пусть дано математическое выражение, в котором используются лексемы (синтаксически неделимые единицы):

1) целые и действительные числа;
2) математические операции: +, –, *, /;
3) круглые скобки;
4) однобуквенные переменные.

Для программного подсчета значения математического выражения необходимо:

1) разбить данное математическое выражение на лексемы;
2) проверить корректность математической записи;
3) записать выражение в виде обратной польской нотации;
4) по обратной польской нотации подсчитать значение выражения (если в выражении встречаются переменная, то ее значение должно запрашиваться с клавиатуры только один раз).

Замечание

Существуют три способа записи арифметических выражений: инфиксная (привычная для нас – знак математической операции помещается между операндами), префиксная (знак математической операции помещается перед операндами) и постфиксная (знак математической операции помещается после операндов). Постфиксную запись арифметического выражения называют обратной польской записью, или нотацией).

Рассмотрим алгоритм формирования обратной польской нотации математического выражения. Для его реализации нам потребуется два списка: очередь (основной список) и стек (вспомогательный список). Напомним, что математические операции умножения и деления имеют высший приоритет по отношению к сложению и вычитанию. При формировании обратной польской нотации будем использовать приведенные ниже правила.

1. Если текущая лексема является числом, или переменной, то она помещается в очередь.
2. Если текущая лексема является открывающейся скобкой, то она помещается в стек.
3. Если текущая лексема является математической операцией и стек пуст, или вершиной стека является открывающаяся скобка, то лексема помещается в стек.
4. Если текущая лексема является математической операцией и стек не пуст, причем вершиной стека не является открывающаяся скобка, то:

a) если вершиной стека является математическая операция одного приоритета с текущей лексемой, то эта операция извлекается из стека и помещается в очередь, а текущая лексема записывается в стек;

b) если вершиной стека является математическая операция с приоритетом выше текущей лексемы, то все операции до открывающейся скобки извлекаются из стека и записываются в очередь, а текущая операция помещается в стек;

c) если вершиной стека является математическая операция с приоритетом ниже текущей лексемы, то текущая лексема помещается в стек.

5. Если текущая лексема является закрывающей скобкой, то из стека извлекаются все операции до открывающейся скобки и помещаются в очередь; открывающаяся скобка также извлекается из стека;
6. Если лексемы закончились и стек оказался не пуст, то все операции извлекаются из стекаи помещаются в очередь.

Проиллюстрируем правила формирования обратной польской нотации на примере математического выражения: $3+(4*a/7-3.5/(2.1*4))*10-a$.

№	Текущая лексема	Стек	Очередь
	3		3
1	+	+	3
2	(	(+	3
3	4	(+	3 4
4	*	* (+	3 4
5	a	* (+	3 4 a
6	/	/ (+	3 4 a *
7	7	/ (+	3 4 a * 7
8	-	- (+	3 4 a * 7 /
9	3.5	- (+	3 4 a * 7 / 3.5
10	/	/ - (+	3 4 a * 7 / 3.5
11	(	(/ - (+	3 4 a * 7 / 3.5
12	2.1	(/ - (+	3 4 a * 7 / 3.5 2.1
13	*	* (/ - (+	3 4 a * 7 / 3.5 2.1
14	4	* (/ - (+	3 4 a * 7 / 3.5 2.1 4
15	)	/ - (+	3 4 a * 7 / 3.5 2.1 4 *
16	)	+	3 4 a * 7 / 3.5 2.1 4 * / -
17	*	* +	3 4 a * 7 / 3.5 2.1 4 * / -
18	10	* +	3 4 a * 7 / 3.5 2.1 4 * / - 10
19	-	-	3 4 a * 7 / 3.5 2.1 4 * / - 10 * +
20	a	-	3 4 a * 7 / 3.5 2.1 4 * / - 10 * + a

Больше лексем нет, поэтому итоговая очередь, содержащая польскую запись математического выражения, будет выглядеть следующим образом:

```
3 4 a * 7 / 3.5 2.1 4 * / - 10 * + a -
```

Чтобы подсчитать значение выражения по обратной польской нотации необходимо последовательно применять операцию к двум аргументам, стоящим слева от операции. Для рассматриваемого выражения порядок выполнения операций будет иметь следующий вид:

```
                  1 2              3  4  5     6  7    8
((3 ((((4 a *) 7 /) (3.5 (2.1 4 *) /) -) 10 *) +) a -)
```

Реализуйте рассмотренный алгоритм самостоятельно.

*Замечание **1***

Разбиение исходного выражения на лексемы можно проводить с помощью стандартных методов для работы со строками, а проверку корректности записи математического выражения осуществлять на этапе формирования польской записи выражения.

*Замечание **2***

*Если выражение начинается с унарного минуса, например, «–7+4/2», или «-a*b», то рекомендуется свести унарный минус к бинарному минусу добавлением незначащего нуля следующим образом: «0-7+4/2» или «0-a*b».*

Практическая работа №4

Структуры данных: Очередь. Метод Add. Метод Take. Решение практических задач с использованием очередей

Цель работы: изучение и практическое применение структуры данных очереди, а также методов Add и Take, чтобы решать практические задачи, которые требуют использования очередей.

Задание:

Задание 1

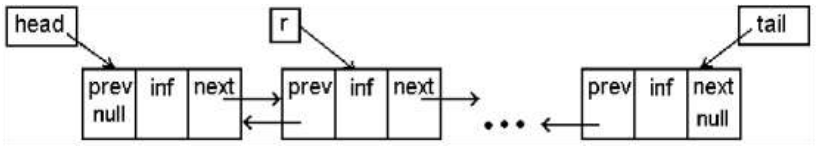
Замечание

Каждую задачу данного раздела решить, реализовав список в виде линейно связанной структуры: очереди. Исходный и измененный список вывести на экран.

1. На основе файла создать список. Удвоить в нем вхождение каждого четного элемента.
2. На основе файла создать список, включив в него только повторяющиеся числа.
3. На основе файла создать список, включив в него только уникальные числа.
4. На основе файла создать список. Перед каждым элементом равным x вставить элемент равный y.
5. На основе файла создать список. Перед каждым элементом равным x удалить один элемент.
6. На основе файла создать список. Заменить каждую последовательность повторяющихся элементов на один элемент.
7. На основе файла создать список. Найти минимальный элемент, и удалить из списка все элементы равные минимальному.
8. На основе файла создать список. Поменять в списке местами первый максимальный и последний минимальный элементы.
9. На основе файла создать список, вычислив среднее арифметическое значение всех элементов. Удалить из списка элементы, значение которых меньше среднего арифметического всех элементов исходного списка.
10. На основе файла создать список. Найти максимальный элемент, и после каждого максимального элемента в списке вставить элемент, значение которого равно среднему арифметическому значению предыдущих элементов.

*Задание **2***

Мы рассмотрели организацию однонаправленных списков. Однако во многих приложениях возникает необходимость организовывать эффективное перемещение по списку как в прямом, так и в обратном направлении. В этой ситуации можно прибегнуть к организации двунаправленного списка, структура которого изображена ниже:



Предложенная структура содержит ссылки на первый, последний и произвольный элементы двунаправленного списка. Разработайте класс, реализующий данную модель организации двунаправленного списка, самостоятельно определив необходимые функциональные члены класса.

С помощью разработанного класса решите следующую задачу: дан файл, компонентами которого являются целые числа. На основе файла создайте двунаправленный список так, чтобы элементы заносились в него в порядке возрастания значений. Выведите на экран содержимое списка в прямом и обратном порядке.

*Задание **3***

Пусть дано математическое выражение, в котором используются лексемы (синтаксически неделимые единицы):

5) целые и действительные числа;

6) математические операции: +, –, *, /;

7) круглые скобки;

8) однобуквенные переменные.

Для программного подсчета значения математического выражения необходимо:

5) разбить данное математическое выражение на лексемы;

6) проверить корректность математической записи;

7) записать выражение в виде обратной польской нотации;

8) по обратной польской нотации подсчитать значение выражения (если в выражении встречаются переменная, то ее значение должно запрашиваться с клавиатуры только один раз).

Замечание

Существуют три способа записи арифметических выражений: инфиксная (привычная для нас – знак математической операции помещается между операндами), префиксная (знак математической операции помещается перед операндами) и постфиксная (знак математической операции помещается после операндов). Постфиксную запись арифметического выражения называют обратной польской записью, или нотацией).

Рассмотрим алгоритм формирования обратной польской нотации математического выражения. Для его реализации нам потребуется два списка: очередь (основной список) и стек (вспомогательный список). Напомним, что математические операции умножения и деления имеют высший приоритет по отношению к сложению и вычитанию. При формировании обратной польской нотации будем использовать приведенные ниже правила.

7. Если текущая лексема является числом, или переменной, то она помещается в очередь.

8. Если текущая лексема является открывающейся скобкой, то она помещается в стек.

9. Если текущая лексема является математической операцией и стек пуст, или вершиной стека является открывающаяся скобка, то лексема помещается в стек.

10. Если текущая лексема является математической операцией и стек не пуст, причем вершиной стека не является открывающаяся скобка, то:

a) если вершиной стека является математическая операция одного приоритета с текущей лексемой, то эта операция извлекается из стека и помещается в очередь, а текущая лексема записывается в стек;

b) если вершиной стека является математическая операция с приоритетом выше текущей лексемы, то все операции до открывающейся скобки извлекаются из стека и записываются в очередь, а текущая операция помещается в стек;

c) если вершиной стека является математическая операция с приоритетом ниже текущей лексемы, то текущая лексема помещается в стек.

11. Если текущая лексема является закрывающей скобкой, то из стека извлекаются все операции до открывающейся скобки и помещаются в очередь; открывающаяся скобка также извлекается из стека;

12. Если лексемы закончились и стек оказался не пуст, то все операции извлекаются из стекаи помещаются в очередь.

Проиллюстрируем правила формирования обратной польской нотации на примере математического выражения: $3+(4*a/7-3.5/(2.1*4))*10-a$.

№	Текущая лексема	Стек	Очередь
	3		3
1	+	+	3
2	(	(+	3
3	4	(+	3 4
4	*	* (+	3 4
5	a	* (+	3 4 a
6	/	/ (+	3 4 a *
7	7	/ (+	3 4 a * 7
8	-	- (+	3 4 a * 7 /
9	3.5	- (+	3 4 a * 7 / 3.5
10	/	/ - (+	3 4 a * 7 / 3.5
11	(	(/ - (+	3 4 a * 7 / 3.5

12	2.1	(/ - (+	3 4 a * 7 / 3.5 2.1
13	*	* (/ - (+	3 4 a * 7 / 3.5 2.1
14	4	* (/ - (+	3 4 a * 7 / 3.5 2.1 4
15	)	/ - (+	3 4 a * 7 / 3.5 2.1 4 *
16	)	+	3 4 a * 7 / 3.5 2.1 4 * / -
17	*	* +	3 4 a * 7 / 3.5 2.1 4 * / -
18	10	* +	3 4 a * 7 / 3.5 2.1 4 * / - 10
19	-	-	3 4 a * 7 / 3.5 2.1 4 * / - 10 * +
20	a	-	3 4 a * 7 / 3.5 2.1 4 * / - 10 * + a

Больше лексем нет, поэтому итоговая очередь, содержащая польскую запись математического выражения, будет выглядеть следующим образом:

```
3 4 a * 7 / 3.5 2.1 4 * / – 10 * + a –
```

Чтобы подсчитать значение выражения по обратной польской нотации необходимо последовательно применять операцию к двум аргументам, стоящим слева от операции. Для рассматриваемого выражения порядок выполнения операций будет иметь следующий вид:

```
                    1 2               3  4  5     6  7    8
((3 ((((4 a *) 7 /) (3.5 (2.1 4 *) /) –) 10 *) +) a –)
```

Реализуйте рассмотренный алгоритм самостоятельно.

*Замечание **1***

Разбиение исходного выражения на лексемы можно проводить с помощью стандартных методов для работы со строками, а проверку корректности записи математического выражения осуществлять на этапе формирования польской записи выражения.

*Замечание **2***

*Если выражение начинается с унарного минуса, например, «–7+4/2», или «-a*b», то рекомендуется свести унарный минус к бинарному минусу добавлением незначащего нуля следующим образом: «0-7+4/2» или «0-a*b».*

Практическая работа №5

Структуры данных: Однонаправленные списки общего вида. Метод Find. Метод Insert. Метод Delete. Решение практических задач с использованием однонаправленных списков общего вида.

Цель работы: изучение и практическое применение структуры данных однонаправленных списков общего вида, а также методов Find, Insert и Delete для работы с этими списками.

Задание:

Задание 1

Замечание

Каждую задачу данного раздела решить, реализовав список в виде линейно связанной структуры: списка общего вида. Исходный и измененный список вывести на экран.

3. На основе файла создать список. Удвоить в нем вхождение каждого четного элемента.
4. На основе файла создать список, включив в него только повторяющиеся числа.

3. На основе файла создать список, включив в него только уникальные числа.
4. На основе файла создать список. Перед каждым элементом равным x вставить элемент равный y.
5. На основе файла создать список. Перед каждым элементом равным x удалить один элемент.
6. На основе файла создать список. Заменить каждую последовательность повторяющихся элементов на один элемент.
7. На основе файла создать список. Найти минимальный элемент, и удалить из списка все элементы равные минимальному.

8. На основе файла создать список. Поменять в списке местами первый максимальный и последний минимальный элементы.

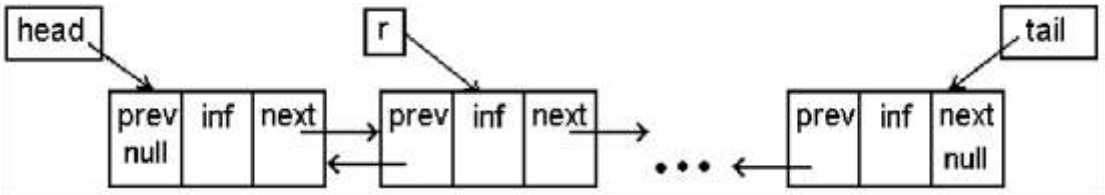
9. На основе файла создать список, вычислив среднее арифметическое значение всех элементов. Удалить из списка элементы, значение которых меньше среднего арифметического всех элементов исходного списка.

10. На основе файла создать список. Найти максимальный элемент, и после каждого максимального элемента в списке вставить элемент, значение которого равно среднему арифметическому значению предыдущих элементов.

*Задание **2***

Мы рассмотрели организацию однонаправленных списков. Однако во многих приложениях возникает необходимость организовывать эффективное перемещение по списку как в прямом, так и в обратном направлении. В этой ситуации можно прибегнуть к организации двунаправленного списка, структура которого изображена ниже:

Предложенная структура содержит ссылки на первый, последний и произвольный элементы



двунаправленного списка. Разработайте класс, реализующий данную модель организации двунаправленного списка, самостоятельно определив необходимые функциональные члены класса.

С помощью разработанного класса решите следующую задачу: дан файл, компонентами которого являются целые числа. На основе файла создайте двунаправленный список так, чтобы элементы заносились в него в порядке возрастания значений. Выведите на экран содержимое списка в прямом и обратном порядке.

*Задание **3***

Пусть дано математическое выражение, в котором используются лексемы (синтаксически неделимые единицы):

9) целые и действительные числа;
10) математические операции: +, –, *, /;
11) круглые скобки;
12) однобуквенные переменные.

Для программного подсчета значения математического выражения необходимо:

9) разбить данное математическое выражение на лексемы;
10) проверить корректность математической записи;
11) записать выражение в виде обратной польской нотации;
12) по обратной польской нотации подсчитать значение выражения (если в выражении встречаются переменная, то ее значение должно запрашиваться с клавиатуры только один раз).

Замечание

Существуют три способа записи арифметических выражений: инфиксная (привычная для нас – знак математической операции помещается между операндами), префиксная (знак математической операции помещается перед операндами) и постфиксная (знак математической операции помещается после операндов). Постфиксную запись арифметического выражения называют обратной польской записью, или нотацией).

Рассмотрим алгоритм формирования обратной польской нотации математического выражения. Для его реализации нам потребуется два списка: очередь (основной список) и стек (вспомогательный список). Напомним, что математические операции умножения и деления имеют высший приоритет по отношению к сложению и вычитанию. При формировании обратной польской нотации будем использовать приведенные ниже правила.

13. Если текущая лексема является числом, или переменной, то она помещается в очередь.

14. Если текущая лексема является открывающейся скобкой, то она помещается в стек.

15. Если текущая лексема является математической операцией и стек пуст, или вершиной стека является открывающаяся скобка, то лексема помещается в стек.

16. Если текущая лексема является математической операцией и стек не пуст, причем вершиной стека не является открывающаяся скобка, то:

a) если вершиной стека является математическая операция одного приоритета с текущей лексемой, то эта операция извлекается из стека и помещается в очередь, а текущая лексема записывается в стек;

b) если вершиной стека является математическая операция с приоритетом выше текущей лексемы, то все операции до открывающейся скобки извлекаются из стека и записываются в очередь, а текущая операция помещается в стек;

c) если вершиной стека является математическая операция с приоритетом ниже текущей лексемы, то текущая лексема помещается в стек.

17. Если текущая лексема является закрывающей скобкой, то из стека извлекаются все операции до открывающейся скобки и помещаются в очередь; открывающаяся скобка также извлекается из стека;

18. Если лексемы закончились и стек оказался не пуст, то все операции извлекаются из стекаи помещаются в очередь.

Проиллюстрируем правила формирования обратной польской нотации на примере математического выражения: $3+(4*a/7-3.5/(2.1*4))*10-a$.

№	Текущая лексема	Стек	Очередь
	3		3
1	+	+	3
2	(	(+	3
3	4	(+	3 4
4	*	* (+	3 4
5	a	* (+	3 4 a
6	/	/ (+	3 4 a *
7	7	/ (+	3 4 a * 7
8	-	- (+	3 4 a * 7 /
9	3.5	- (+	3 4 a * 7 / 3.5
10	/	/ - (+	3 4 a * 7 / 3.5
11	(	(/ - (+	3 4 a * 7 / 3.5
12	2.1	(/ - (+	3 4 a * 7 / 3.5 2.1
13	*	* (/ - (+	3 4 a * 7 / 3.5 2.1
14	4	* (/ - (+	3 4 a * 7 / 3.5 2.1 4
15	)	/ - (+	3 4 a * 7 / 3.5 2.1 4 *
16	)	+	3 4 a * 7 / 3.5 2.1 4 * / -
17	*	* +	3 4 a * 7 / 3.5 2.1 4 * / -
18	10	* +	3 4 a * 7 / 3.5 2.1 4 * / - 10
19	-	-	3 4 a * 7 / 3.5 2.1 4 * / - 10 * +
20	a	-	3 4 a * 7 / 3.5 2.1 4 * / - 10 * + a

Больше лексем нет, поэтому итоговая очередь, содержащая польскую запись математического выражения, будет выглядеть следующим образом:

```
3 4 a * 7 / 3.5 2.1 4 * / - 10 * + a –
```

Чтобы подсчитать значение выражения по обратной польской нотации необходимо последовательно применять операцию к двум аргументам, стоящим слева от операции. Для рассматриваемого выражения порядок выполнения операций будет иметь следующий вид:

```
                  1 2              3  4  5     6  7    8
((3 ((((4 a *) 7 /) (3.5 (2.1 4 *) /) –) 10 *) +) a –)
```

Реализуйте рассмотренный алгоритм самостоятельно.

*Замечание **1***

Разбиение исходного выражения на лексемы можно проводить с помощью стандартных методов для работы со строками, а проверку корректности записи математического выражения осуществлять на этапе формирования польской записи выражения.

*Замечание **2***

*Если выражение начинается с унарного минуса, например, «–7+4/2», или «-a*b», то рекомендуется свести унарный минус к бинарному минусу добавлением незначащего нуля следующим образом: «0-7+4/2» или «0-a*b».*

Практическая работа №6

Алгоритмы обработки данных: Деревья. Деревья бинарного поиска. Методы Add и Node.Add. Методы Preorder (Node.Preorder), Inorder (Node.Inorder), Postorder(Node. Postorder). Методы Delete и Node.Delete. Решение практических задач.

Цель работы: изучить принципы работы алгоритмов обработки данных, их применение для решения различных задач и оценка их эффективности.

Задание:

Задание 1

В файле input.txt хранится последовательность целых чисел. По входной последовательности построить дерево бинарного поиска и найти для него:

1) сумму нечетных значений узлов дерева;
2) количество четных значений узлов дерева;
3) среднее арифметическое положительных значений узлов дерева;
4) наибольшее из значений листьев;
5) сумму значений листьев;
6) количество узлов, имеющих только одного левого потомка;
7) сумму значений узлов, имеющих только одного правого потомка;
8) количество узлов, имеющих двух потомков;
9) среднее арифметическое значение узлов, имеющих два потомка;
10) количество узлов, значение которых больше среднего арифметического.

Задание 2

В файле input.txt хранится последовательность целых чисел. По входной последовательности построить дерево бинарного поиска и:

1) распечатать узлы k-го уровня дерева;
2) найти количество узлов на k-том уровне дерева;
3) распечатать дерево по уровням;
4) вычислить глубину заданного узла;
5) для каждого узла дерева вычислить его глубину;
6) заменить отрицательные значения узлов дерева на противоположные;
7) проверить, является ли дерево идеально сбалансированным;
8) проверить, можно ли удалить какой-то один узел так, чтобы дерево стало идеально сбалансирванным;
9) проверить, останется ли дерево деревом бинарного поиска, если из него удалить все нечетные узлы;

10) проверить, является ли первое дерево поддеревом второго, и найти для него наименьшее из значений листьев;
11) найти для него количество узлов, имеющих только одного правого потомка;
12) найти для него сумму значений узлов, имеющих только одного потомка;
13) определить, количество узлов дерева со значением, отличным от k;
14) проверить, является ли данное дерево деревом бинарного поиска;
15) поменять в нем местами узлы, хранящие минимальное и максимальное значение;
16) найти глубину заданного узла.

Задание 3

Мы рассмотрели реализацию АТД «дерево бинарного поиска», которое по способу построения может оказаться «перегруженным» на левое или правое поддерево. В некоторых задачах возникает необходимость равномерно разместить элементы некоторой последовательности по уровням дерева. В этой ситуации лучше использовать идеально сбалансированные деревья.

Разработайте класс, реализующий АТД «идеально сбалансированное дерево», самостоятельно определив в нем функциональные члены класса.

С помощью разработанного класса решите следующую задачу: дан файл, компонентами которого являются целые числа. На основе файла создайте идеально сбалансированное дерево. Выведите его на экран по уровням. Удалите в нем наибольший и наименьший элементы так, чтобы дерево осталось идеально сбалансированным. Затем еще раз выведите дерево по уровням.

Задание 4

Код Хаффмана.

Приведем пример применения бинарных деревьев в качестве структур данных. Для этого рассмотрим задачу конструирования кода Хаффмана. Предположим, что мы имеем сообщение, состоящее из последовательности символов. В каждом сообщении символы независимы и появляются с известной вероятностью, не зависящей от позиции символа в сообщении. Например, мы имеем сообщение из шести символов a, b, c, +, *, / с вероятностью появления символов в тексте 0.32, 0.1, 0.11, 0.22, 0.13 и 0.12 соответственно.

Мы хотим закодировать каждый символ последовательностью из нулей и единиц так, чтобы код любого символа являлся префиксом кода сообщения, состоящего из последующих символов. Это префиксное свойство позволяет декодировать строку из нулей и единиц последовательным удалением префиксов (кодов символов) из этой строки. В таблице показаны две возможные кодировки наших символов.

символ	вероятность	код 1	код 2
a	0.32	110	11
b	0.1	000	000
c	0.11	001	001
+	0.22	010	01
*	0.13	101	101
/	0.12	100	100

Таблица 1

Первый код обладает префиксным свойством, поскольку любая последовательность из трех битов будет префиксом для другой последовательности из трех битов. Алгоритм декодирования очень прост: надо брать по три бита и преобразовывать каждую группу битов в соответствующие символы. Так, например, последовательности 110010000100001010110 (110 010 000 100 001 010 110 – после разбиения на группы битов) будет соответствовать исходное сообщение a+b/c+a.

Второй код также обладает префиксным свойством, однако процесс декодирования чуть сложнее: последовательность нельзя заранее разбить на группы из трех битов, так как символы могут кодироваться двумя или тремя битами. Для примера рассмотрим последовательность 11010001000010111 (11 01 000 100 001 01 11 – после разбиения на группы битов), которой будет соответствовать то же исходное сообщение a+b/c+a.

Преимуществом второго кода является то, что сообщение, закодированное с его помощью, будет иметь минимизированную среднюю длину кода. Так код 1 имеет среднюю длину кода 3, а

код 2 – 2.7, благодаря чему кодовая последовательность исходного сообщения при кодировании кодом 2 уменьшилась на четыре бита.

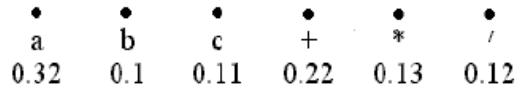
Таким образом, задача конструирования кода Хаффмана заключается в следующем: имея множество символов и значения вероятностей их появления в сообщениях, построить код с префиксным свойством, чтобы средняя длина кода (в вероятностном смысле) последовательности символов была минимальна.

Для построения кода мы будем использовать лес – совокупность деревьев, чьи листья будут помечены символами, для которых разрабатывается кодировка, а корни помечены суммой вероятностей этих символов. Мы будем называть эти суммарные вероятности весом дерева. В начале работы каждому символу соответствует дерево, состоящее из одного узла. В конце мы получим одно дерево, все листья которого помечены кодируемыми символами. В этом дереве путь от корня к какому-либо листу представляет код для символа-метки этого листа, составленный по схеме, согласно которой левый сын узла соответствует метке 0, правый сын узла – метке 1.

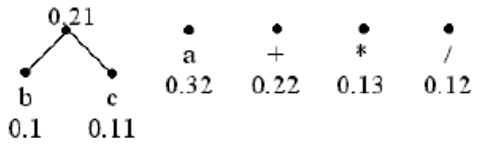
Важным этапом в формировании кода является выбор из леса двух деревьев с наименьшими весами. Эти два дерева комбинируются в одно с весом, равным сумме весов составляющих его деревьев, причем в качестве левого поддерева выбирается дерево с наименьшим весом.

Рассмотрим конструирования кода Хаффмана по шагам.

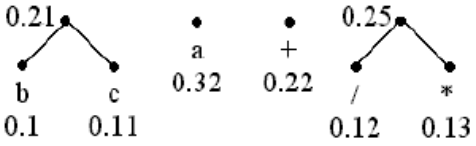
1) Исходная ситуация



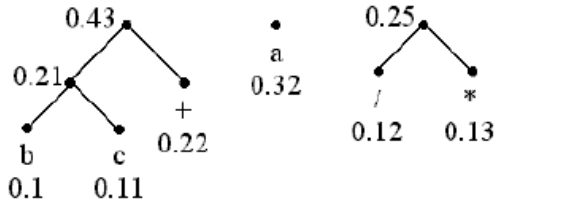
2) Слияние поддеревьев с весами 0.1 и 0.11



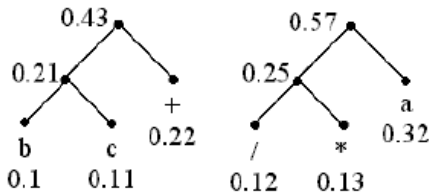
3) Слияние поддеревьев с весами 0.12 и 0.13



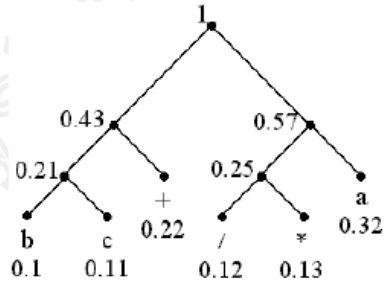
4) Слияние поддеревьев с весами 0.21 и 0.22



5) Слияние поддеревьев с весами 0.25 и 0.32



6) Слияние поддеревьев с весами 0.43 и 0.57



Итоговое бинарное дерево, представляющее код Хаффмана с префиксным свойством изображено на рис.10.

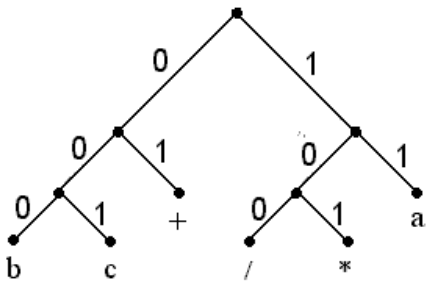


Рисунок 1

Чтобы по дереву получить код символа, необходимо спускаться от корня к соответствующему листу, последовательно записывая метки. Коды построенного дерева совпадают с кодом 2 см. таблица 1.

Реализуйте алгоритм построения кода Хаффмана самостоятельно и с его помощью проведите кодирование и декодирование произвольных сообщений.

Практическая работа №7

Алгоритмы на графах: Графы. Способы представления графов. Алгоритмы обхода графа. Алгоритмы нахождения кратчайших путей. Программная реализация АТД «граф». Решение практических задач с использованием графов.

Цель работы: ознакомление и понимание основных концепций и алгоритмов, связанных с работой с графами.

Задание:

Задание 1

Во входном файле указывается количество вершин графа/орграфа и матрица смежности. Для заданного графа:

1) подсчитать количество вершин, смежных с данной;
2) вывести на экран все вершины, не смежные с данной;
3) удалить из графа ребро, соединяющее вершины a и b;
4) добавить в граф ребро, соединяющее вершину a и b;
5) добавить новую вершину;
6) исключить данную вершину;
7) выяснить, соседствуют ли две заданные вершины с третьей.

Для заданного орграфа:

8) для данной вершины вывести на экран все "выходящие" соседние вершины;
9) для данной вершины вывести на экран все "входящие" соседние вершины;
10) удалить дугу, соединяющую вершины a и b;
11) добавить дугу, соединяющую вершину a и b;
12) исключить данную вершину;
13) добавить новую вершину. Для взвешенного графа:
14) найти все вершины графа, недостижимые из данной;
15) определить, существует ли путь длиной не более L между двумя данными вершинами;
16) найти все вершины, из которых существует путь в данную;
17) найти все истоки графа;
18) найти все стоки графа;
19) найти вершину, наиболее удаленную от центральной вершины графа;
20) найти вершину, наиболее приближенную к центральной вершине графа.

Задание 2

Во входном файле задается:

1) в первой строке N – количество городов;
2) начиная со второй строки через пробел названия N-городов;
3) с новой строки матрица смежности взвешенного графа, описывающая схему дорог. Например, входной файл для рис. 3 может выглядеть следующем образом:

```
5
Березовка Еремеевка Октябрьское Рузаевка Сосновка
0  70 0  0  20
70 0  75 25 15
0  75 0  40 60
0  25 40 0  0
20 15 60 0  0
```

Остальные данные, необходимые для решения задачи, вводятся с клавиатуры.

1. Найти кратчайший путь, соединяющий города A и B и проходящий только через заданное множество городов.
2. Определить, существует ли город, из которого можно добраться до каждого их остальных городов, проезжая не более N км.
3. N-периферией называется множество городов, расстояние от которых до выделенного города (столицы) больше N. Определить N-периферию для заданной столицы и значения N.
4. Определите, между какими городами нужно построить дорогу, что бы расстояние между любыми городами не превышало N км.
5. Определите, какое наименьшее количество дорог нужно закрыть (и между какими городами), чтобы из города A нельзя было попасть в город B.

Задание 3

Модифицировать рассмотренные алгоритмы обходов графа и алгоритмы Дейкстры и Флойда, реализовав граф в виде списка смежности.

Оценочные средства для промежуточного контроля

Вопросы к зачету

1. Базовые типы данных
2. Преобразования типов
3. Подставляемые функции
4. Перегрузка функции
5. Рекурсивные функции
6. Одномерные массивы
7. Двумерные массивы
8. Задачи сортировки элементов массива
9. Структуры
10. Объединения
11. Одномерные динамические массивы
12. Двумерные динамические массивы
13. Динамические структуры данных: однонаправленные списки
14. Динамические структуры данных: двунаправленные списки
15. Динамические структуры данных: очередь
16. Динамические структуры данных: стек
17. Динамические структуры данных: бинарные деревья
18. Алгоритм перебора с возвратом
19. Алгоритмы поиска в линейных структурах
20. Алгоритмы хеширования данных
21. Алгоритмы поиска на основе деревьев
22. Алгоритмов поиска на графе
23. Алгоритмы нахождения кратчайшего пути в графах

Оценивание результатов обучения в форме уровня сформированности элементов компетенций проводится путем контроля во время промежуточной аттестации в форме зачета:

а) оценка «зачтено» – компетенция(и) или ее часть(и) сформированы на базовом уровне;

б) оценка «не зачтено» – компетенция(и) или ее часть(и) не сформированы.

Критерии, на основе которых выставляются оценки при проведении текущего контроля и промежуточной аттестации приведены в табл. 1.

Оценки «Не зачтено» ставятся также в случаях, если обучающийся не приступал к выполнению задания, а также при обнаружении следующих

нарушений:

– списывание;

– плагиат;

– фальсификация данных и результатов работы.

Таблица 1 – Критерии выставления оценок при проведении текущего контроля и промежуточной аттестации

Шкала оценки	Оценка	Критерий выставления оценки
Двухбалльная шкала	Зачтено	Обучающийся ответил на теоретические вопросы. Показал знания в рамках учебного материала. Выполнил практические задания. Показал удовлетворительные умения и владения навыками применения полученных знаний и умений при решении задач в рамках учебного материала
	Не зачтено	Обучающиеся при ответе на теоретические вопросы и при выполнении практических заданий продемонстрировал недостаточный уровень знаний и умений при решении задач в рамках учебного материала. При ответах на дополнительные вопросы было допущено множество неправильных ответов

2.3. Итоговая диагностическая работа по дисциплине

ЗАДАНИЯ ДЛЯ ДИАГНОСТИЧЕСКОЙ РАБОТЫ ПО ДИСЦИПЛИНЕ «Структуры и алгоритмы обработки данных»

Номер задания	Правильный ответ *	Содержание вопроса	Компетенция	Код и наименование индикатора достижения компетенции
1.	Сортировка	Процесс упорядочивания элементов в массиве или коллекции в соответствии с определенным порядком.	ПК-2	**ИД-4** $_{\text{ПК-2}}$ Знает структуры и алгоритмы обработки данных и анализирует их для проектирования ПО
2.	1. Алгоритм начинает сортировку с поиска наименьшего элемента во всей последовательности. 2. Найденный наименьший элемент меняется местами с первым элементом в последовательности. 3. Затем алгоритм ищет наименьший элемент в оставшейся части последовательности (без уже отсортированных элементов) и меняет его местами с вторым элементом.	Как работает алгоритм сортировки посредством выбора?	ПК-2	**ИД-4** $_{\text{ПК-2}}$

	4. Этот процесс повторяется до тех пор, пока все элементы не будут отсортированы			
3.	b, c, f	Какие примеры практических задач могут быть решены с использованием стеков? (несколько вариантов ответа) a. Сложные математические вычисления b. Управление вызовами функций c. Обход деревьев d. Работа с графами e. Сортировка больших объемов данных f. Обратная польская запись	ПК-2	**ИД-4 ПК-2**
4.	Стек	Структура данных, которая представляет собой коллекцию элементов, упорядоченных по принципу "последним пришел - первым вышел".	ПК-2	**ИД-4 ПК-2**
5.	c	Что делает метод Insert в однонаправленном списке? a. Удаляет элемент из списка b. Ищет элемент с определенным значением в списке c. Вставляет элемент на определенную позицию в списке d. Добавляет элемент в конец списка	ПК-2	**ИД-4 ПК-2**
6.	Особый тип двоичного дерева, в котором каждый узел содержит ключ и имеет двух потомков: левого и правого.	Что такое «дерево бинарного поиска»?	ПК-2	**ИД-4 ПК-2**
7.	Граф в контексте алгоритмов на графах	Абстрактная структура данных, состоящая из вершин (узлов) и ребер (связей), которые соединяют эти вершины.	ПК-2	**ИД-4 ПК-2**
8.	b	Что делает метод Pop в стеке? a. Добавляет элемент в стек b. Удаляет последний добавленный элемент, и возвращает его значение c. Возвращает верхний элемент стека без удаления d. Удаляет все элементы из стека	ПК-2	**ИД-4 ПК-2**
9.	Эффективность. Последовательный поиск имеет линейную сложность,	В чем состоит основное отличие последовательного поиска от двоичного?	ПК-2	**ИД-4 ПК-2**

	a двоичный - логарифмическую.			
10.	В разбиении массива на подмассивы, сортировке каждого подмассива отдельно и объединении их в итоговый отсортированный массив.	В чем заключается метод быстрой сортировки?	ПК-2	**ИД-4** $_{\text{ПК-2}}$
11.	Класс Stopwatch, объявленный в пространстве имен System.Diagnostics	Использование какого класса является одним из самых простых способов измерения времени, затраченного на выполнение программы?	ПК-2	**ИД-4** $_{\text{ПК-2}}$
12.	Алгоритм Шелла	Как называется усовершенствованный вариант алгоритма сортировки вставками?	ПК-2	**ИД-4** $_{\text{ПК-2}}$
13.	a, d	Какие операции можно выполнять с очередью? (несколько вариантов ответа) a. Добавление элемента в очередь b. Доступ к элементам по индексу c. Сортировка d. Проверка наличия элементов в очереди e. Произвольное удаление элемента	ПК-2	**ИД-4** $_{\text{ПК-2}}$
14.	e	В чем отличие однонаправленных списков общего вида от других списков? a. Однонаправленность b. Вставка и удаление элементов c. Эффективность по памяти d. Применение e. Все вышеперечисленное	ПК-2	**ИД-4** $_{\text{ПК-2}}$
15.	Они определяют порядок посещения узлов.	Зачем нужны методы Preorder, Inorder и Postorder при обходе дерева?	ПК-2	**ИД-4** $_{\text{ПК-2}}$
16.		Какие способы представления графов существуют? 1. Матрица инцендентности 2. Список смежности 3. Матрица смежности 4. Список ребер a. Верно только 1 и 3 b. Верно только 2 и 4 c. Верно только 2, 3 и 4 d. Верно все	ПК-2	**ИД-4** $_{\text{ПК-2}}$
17.	Очередь	Специальный тип коллекции, который хранит элементы в порядке FIFO (First-In-First-Out), то есть первый добавленный элемент будет первым извлеченным.	ПК-2	**ИД-4** $_{\text{ПК-2}}$

18.	a, b, c	Какие факторы могут влиять на время выполнения программы? (возможны несколько вариантов ответа) a. Ресурсы компьютера b. Размер входных данных c. Язык программирования d. Личные предпочтения e. Все вышеперечисленное	ПК-2	**ИД-4** $_{\text{ПК-2}}$
19.	a, b, c, e	Какие практические задачи могут быть решены с использованием графов? (несколько вариантов ответа) a. Анализ социальных сетей b. Моделирование компьютерных сетей c. Поисковая оптимизация d. Обработка больших объемов данных e. Оптимизация маршрутов доставки	ПК-2	**ИД-4** $_{\text{ПК-2}}$
20.	Метод «пузырька», сортировка вставками, сортировка посредством выбора, алгоритм сортировки Шелла, быстрая сортировка	Какие методы используются для сортировки данных?	ПК-2	**ИД-4** $_{\text{ПК-2}}$

Критерии и шкалы оценивания уровня сформированности компетенции

Таблица – Критерии выставления оценок при итоговой диагностической работе по дисциплине по дисциплине «Структуры и алгоритмы обработки данных»

Шкала оценки	Оценка	Критерий выставления оценки
100-процентная шкала	Отлично	85-100 %% правильных ответов
	Хорошо	71-84 %% правильных ответов
	Удовлетворительно	50-70 %% правильных ответов
	Неудовлетворительно	менее 50 % правильных ответов