Приложение №9.4.31
к ППССЗ по специальности 09.02.03
Программирование в компьютерных системах

ФОНД ОЦЕНОЧНЫХ СРЕДСТВ
ПО УЧЕБНОЙ ДИСЦИПЛИНЕ
ОП.08 ТЕОРИЯ АЛГОРИТМОВ

для специальности

09.02.03 Программирование в компьютерных системах

Уровень подготовки – базовой

Год начала подготовки - 2020

2023

Оглавление

1. Паспорт фонда оценочныхсредств

Фонд оценочных средств предназначен для контроля и оценки образовательных достижений обучающихся, освоивших программу учебной дисциплины ОП.08 Теория алгоритмов.

ФОС включают контрольные материалы для проведения текущего контроля и промежуточной аттестации в форме дифференцированногозачёта.

ФОС разработаны на основании положений:

программы подготовки специалистов среднего звена по специальности СПО 09.02.03 Программирование в компьютерных системах;

программы учебной дисциплины ОП.08 Теория алгоритмов;

учебного плана по специальности СПО 09.02.03 Программирование в компьютерных системах;

положения «О фонде оценочных средств для проведения текущего контроля успеваемости промежуточной и итоговой аттестации студентов и обучающихся филиала СамГУПС в г. Алатыре».

2. Результаты освоения дисциплины, подлежащиепроверке

2.1 Перечень умений, знаний, общихкомпетенций

В результате освоения учебной дисциплины Теория алгоритмов обучающийся должен обладать предусмотренными ФГОС по специальности СПО 09.02.03 Программирование в компьютерных системах (базовый уровень) следующими умениями, знаниями, которые формируют профессиональную компетенцию, и общими компетенциями:

У1. - разрабатывать алгоритмы для конкретныхзадач;
У2. - определять сложность работыалгоритмов.

З1. - основные модели алгоритмов;
З2. - методы построения алгоритмов;
З3. - методы вычисления сложности работы алгоритмов.

ОК 1. Понимать сущность и социальную значимость своей будущей профессии, проявлять к ней устойчивый интерес.
ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.
ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность.
ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.
ОК 5. Использовать информационно-коммуникационные технологии в профессиональной деятельности.
ОК 6. Работать в коллективе и в команде, эффективно общаться с коллегами, руководством, потребителями.
ОК 7. Брать на себя ответственность за работу членов команды (подчиненных), за результат выполнения заданий.
ОК 8. Самостоятельно определять задачи профессионального и личностного развития, заниматься самообразованием, осознанно планировать повышение квалификации.
ОК 9. Ориентироваться в условиях частой смены технологий в профессиональной деятельности.
деятельности.
ПК 1.5Осуществлять оптимизацию программного кода модуля.
ПК 2.3 Решать вопросы администрирования базы данных.
ПК 3.2 Выполнять интеграцию модулей в программную систему.
ПК 3.3 Выполнять отладку программного продукта с использованием специализированных программных средств.

В рамках программы учебной дисциплины реализуется программа воспитания, направленная на формирование следующих личностных результатов (дескриптеров):

ЛР 5. Демонстрирующий приверженность к родной культуре, исторической памяти на основе любви к Родине, родному народу, малой родине, принятию традиционных ценностей многонационального народа России.

ЛР 7.Осознающий приоритетную ценность личности человека; уважающий собственную и чужую уникальность в различных ситуациях, во всех формах и видах деятельности.

ЛР 13. Демонстрирующий готовность обучающегося соответствовать ожиданиям работодателей: ответственный сотрудник, дисциплинированный, трудолюбивый, нацеленный на достижение поставленных задач, эффективно взаимодействующий с членами команды, сотрудничающий с другими людьми, проектно мыслящий.

ЛР 17. Ценностное отношение обучающихся к своему Отечеству, к своей малой и большой Родине, уважительного отношения к ее истории и ответственного отношения к ее современности.

ЛР 18. Ценностное отношение обучающихся к людям иной национальности, веры, культуры; уважительного отношения к их взглядам.

ЛР 19. Уважительное отношения обучающихся к результатам собственного и чужого труда.

ЛР 22 Приобретение навыков общения и самоуправления.

ЛР 23. Получение обучающимися возможности самораскрытия и самореализация личности.

2.2. Форма аттестации

Формой аттестации по учебной дисциплине является дифференцированный зачет

Таблица 1.1

Результаты обучения: умения, знания и общие компетенции	**Показатели оценки результата**	**Форма контроля и оценивания**
Уметь:		
У1. *- разрабатывать алгоритмы для конкретных задач;* ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность икачество. ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностногоразвития. ОК 5. Использовать информационно-коммуникационные технологии в профессиональной деятельности.	- строить блок-схемы алгоритмов; - составлять алгоритмы различной структуры в зависимости от требований задачи.	Устный опрос Самостоятельные работы; практические работы контрольные работы;
У 2 *- определять сложность работы алгоритмов.****;*** ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность икачество. ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность.	-давать количественную и качественную оценку сложности алгоритма.	Устный опрос Самостоятельные работы; практические работы контрольные работы;
Знать:		
З1 *- основные модели алгоритмов;* ОК 1. Понимать сущность и социальную значимость своей будущей профессии, проявлять к ней устойчивый интерес. ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность икачество. ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность. ОК 6. Работать в коллективе и в команде, эффективно общаться с коллегами, руководством, потребителями. ОК 7. Брать на себя ответственность за	- типы алгоритмов; -основные свойства алгоритмов.	

32 - *методы построения алгоритмов;* ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития. ОК 5. Использовать информационно-коммуникационные технологии в	- нисходящее построение алгоритма; - метод последовательной детализации	
33 - *методы вычисления сложности работы алгоритмов*. ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность икачество. ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них	- методы количественной и качественной оценки сложностиалгоритма.	

3. Оценка освоения учебнойдисциплины

3.1. Формы и методыоценивания

Предметом оценки служат умения и знания, предусмотренные ФГОС по дисциплине ОП.08 Теория алгоритмов, направленные на формирование общих и профессиональных компетенций.

Оценка освоения дисциплины ОП.08 Теория алгоритмовтекущий контроль успеваемости, итоговую аттестацию в виде дифференцированного зачета. Проведение текущего контроля успеваемости осуществляется в форме устных опросов, письменных заданий, практических занятий. Для этих целей формируются фонды оценочных средств, включающие типовые задания, тесты и методы контроля, позволяющие оценить знания, умения и уровень приобретенных компетенций.

Контроль и оценка освоения учебной дисциплины по темам (разделам)

Таблица 2.2

Элемент учебной дисциплины	Формы и методы контроля			
	Контроль в ходе изучения дисциплины		Промежуточная аттестация	
	Форма контроля	Проверяемые ОК, У, З	Форма контроля	Проверяя-емые ОК, У, З
Раздел 1 Основные модели алгоритмов			Дифференцированный зачет	У1, З1, ОК 4, ОК 5,ОК2
Тема 1.1 Введение в теорию алгоритмов	Тестирование Самостоятельные работы	У1, З1, ОК 4, ОК 5,ОК 2		
Тема 1.2 Модели вычислений	Тестирование Самостоятельные работы	У1, З1, ОК 4, ОК 5,ОК 2		
Раздел 2. Методы построения алгоритмов			Дифференцированный зачет	У1 *З3ОК 2, ОК4, ОК5, ОК3*
Тема. 2.1 Программирование в алгоритмах	Практическое занятие №1-21 Тестирование Самостоятельные работы	У2 *З3ОК 2, ОК4, ОК5, ОК3*		
Раздел 3. Методы вычисления сложности работы алгоритмов			Дифференцированный зачет	У2 *З3ОК 2, ОК4, ОК5, ОК3*
Тема 3.1 Введение ванализ алгоритмов	Практическое занятие № 22-23 Самостоятельные работы	У2 *З3ОК 2, ОК4, ОК5, ОК3*		

3.2. Типовые задания для оценки освоения учебнойдисциплины

Практические занятия

Практическое занятие №1 по теме: «Решение задач с использованием линейных алгоритмов.»

Цель: Научиться разрабатывать простейшие линейные алгоритмы для решения конкретных задач, строить блок-схемы линейных алгоритмов, записывать линейный алгоритм на алгоритмическом языке и писать программу линейной структуры на языке Turbo Pascal 7.0

Ход занятия:

1. Повторить основные элементы блок-схемалгоритмов.
2. Повторить основные типы алгоритмов и блок-схемы основных алгоритмических структур.
3. Написать,отладитьипроверитьработупрограммдлярешениязадач,указанныхв вариантах1-14.
4. Написать отчет по результатамработы.

Вариант	Задача 1	Задача 2
1	Найти площадь кольца по заданным внешнему и внутреннему радиусам	Написать программу вычисления стоимости поездки на автомобиле на дачу (туда и обратно). Исходными данными являются: расстояние до дачи (км); количество бензина, которое потребляет автомобиль на 100 км пробега; цена одного литра бензина.
2	Даны катеты прямоугольного треугольника. Найти его периметр.	
3	Известны координаты двух точек на плоскости. Найти расстояние между ними.	
4	Дано трехзначное число. Вывестина экран это число в обратномпорядке.	
5	Найти стоимость покупки, если известны цены товаров за единицу и количество товара.	
6	Дан ромб с заданной стороной и одна из его диагоналей. Вычислить вторую диагональ.	
7	Даны три стороны треугольника. Вычислить его площадь по формуле Герона.	
8	Написать программу вычисления сопротивления электрической цепи, состоящей из двух параллельно соединенных сопротивлений.	
9	Написать программу вычисления сопротивления электрической цепи, состоящей из двух последовательно соединенных сопротивлений.	

10	Написать программу, вычисляющую скорость, с которой бегун пробежал дистанцию.	
11	Написать программу пересчета расстояния из верст в километры (1 верста — это 1066,8 м).	
12	Написать программу пересчета величины временного интервала, заданного в минутах, в величину, выраженную в часах и минутах.	
13	Написать программу пересчета веса из фунтов в килограммы (1 фунт — 405,9 грамма).	
14	Написать программу вычисления площади поверхности цилиндра.	

Практическое занятие №2 по теме: «Решение задач с использованием разветвляющихся алгоритмов.»

Цель работы: Научиться применять оператор ветвления и множественный выбор при решении задач, составлять блок-схему алгоритма, записывать алгоритм на алгоритмическом языке, писать по данному алгоритму программу на языке Turbo Pascal 7.0

Ход занятия:

5. Повторитьосновныеэлементыблок-схемалгоритмов,основныетипыалгоритмови блок-схемы основных алгоритмическихструктур.
6. Написать алгоритмы, построить блок-схемы алгоритмов.
7. Написать,отладитьипроверитьработупрограммдлярешениязадач,указанныхв вариантах1-14.
8. Написать отчет по результатамработы.

Вар	Задачи
1	1. Написать алгоритм, который вычисляет частное двух чисел. Алгоритм должен проверять правильность введенных пользователем данных и, если ониневерные (делитель равен нулю), вы давать сообщение обошибке. 2. Напишите алгоритм, который запрашивает у пользователя номер месяца и затем выводитсоответствующееназваниевременигода.Вслучае,еслипользовательведет недопустимое число, алгоритм должен вывести сообщение "Ошибка вводаданных".
2	1. Написать алгоритм вычисления площади кольца. Алгоритм должен проверять правильность исходных данных. (радиус отверстия не должен быть большерадиуса кольца) 2. Написатьалгоритм,которыйзапрашиваетупользователяномерднянеделиивыводит одно из сообщений: "Рабочий день", "Суббота" или"Воскресенье".
3	1. Написать алгоритм, который переводит время из минут и секунд в секунды. Алгоритм должен проверять правильность введенных пользователем данных и в случае,еслиданныеневерные,выводитьсоответствующеесообщение.(количество секунд не может быть больше60) 2. Написатьалгоритм,которыйпомаксимальномуколичествупассажировопределяет вид транспорта (велосипед, автомобиль, маршрутка,автобус).

4	1. Написать алгоритм, который проверяет, является ли годвисокосным. 2. Написатьалгоритм,которыйповведеннойоценке-цифревыводитоценку-слово(4-хорошо).
5	1. Написатьалгоритмвычислениясопротивленияэлектрическойцепи,состоящейиз двух сопротивлений. Сопротивления могут быть соединены последовательно или параллельно. 2. Написать алгоритм, который реализует эпизод из русской сказки: витязь нараспутье.
6	1. Написать алгоритм решения квадратного уравнения. Алгоритм должен проверять правильность исходных данных и в случае, если коэффициент при второйстепени неизвестного равен нулю, выводить соответствующеесообщение. 2. Написать алгоритм, который по введенному символу-знаку выводит название математическойоперации.
7	1. Написатьалгоритмвычислениястоимостипокупкисучетомскидки.Скидкав10% предоставляется, если сумма покупки больше 1000руб. 2. Написать алгоритм, который запрашивает номер месяца, а выводит егоназвание.
8	1. Написать алгоритм вычисления стоимости покупки с учетом скидки. Скидка в 3% предоставляется, если сумма покупки больше 500 руб, в 5% — если суммабольше 1000 руб. 2. Написать алгоритм, который запрашивает название столицы, а выдает название государства. Предусмотреть вариант ответа, когда город незнакомый.Использовать конструкциювыбор.
9	1. Написать алгоритм проверки знания даты основания г.Алатыря. В случае неправильногоответапользователя,алгоритмдолженвыводитьправильныйответ. 2. Написатьалгоритм,которыйзапрашиваетномерпризовогоместа,авыдаетназвание (золото, серебро, бронза). Предусмотреть вариант ответа для 4 и 5места.
10	1. Написатьалгоритмпроверкизнаниядатыначалавтороймировойвойны.Вслучае неправильного ответа пользователя, алгоритм должен выводить правильный ответ. 2. Написать алгоритм, который по введенной первой букве (М,Ж) определяетпол
11	1. Написать алгоритм, который сравнивает два введенных с клавиатуры числа. Алгоритм должен указать, какое число больше, или, если числа равны,вывести соответствующее сообщение. 2. Написать алгоритм, который запрашивает номер региона России, а выдаетего название.
12	1. Написать алгоритм, который выводит пример на умножение двух однозначных чисел, запрашивает ответ пользователя, проверяет его и выводит сообщение "Правильно!" или "Вы ошиблись" и правильныйрезультат. 2. Написатьалгоритм,которыйзапрашиваетназваниепроизведения(популярного, известного), а выдает имя егоавтора.
13	1. Написать алгоритм, который выводит пример на вычитание (в пределах100), запрашивает ответ пользователя, проверяет его и выводит сообщение "Правильно!" или "Вы ошиблись" и правильный результат. 2. Написать алгоритм, который запрашивает имя режиссера, а выдает названияего фильмов.
14	1. Написатьалгоритм,которыйпроверяет,являетсяливведенноепользователемцелое число четным. 2. Написать алгоритм, который запрашивает название дня недели, а выдаетрасписание занятий на этотдень.

Дополнительные задания

3. Написатьалгоритм,которыйпроверяет,делитсялинатривведенноесклавиатурыцелое число.

4. Написатьалгоритмвычислениястоимостиразговорапотелефонусучетом20%скидки, предоставляемой по субботам ивоскресеньям.
5. Написатьалгоритм,которыйвычисляетоптимальныйвесдляпользователя,сравнивает его с реальным и выдает рекомендацию о необходимости поправиться или похудеть. Оптимальный вес вычисляется по формуле: Рост (см) —100.
6. Написать алгоритм, который после введенного с клавиатуры числа (в диапазоне от 1 до 999), обозначающего денежную единицу, дописывает слово "рубль" в правильнойформе. Например, 12 рублей, 21 рубль и т. д.
7. Написать алгоритм, который после введенного с клавиатуры числа (в диапазоне от 1 до 99), обозначающего денежную единицу, дописывает слово "копейка" в правильнойформе. Например, 5 копеек, 41 копейка и т.д.
8. Написать алгоритм, который вычисляет дату следующегодня.

Практическое занятие №3 по теме: «Решение задач с использованием циклических алгоритмов.»

Цель работы: Научиться применять операторы цикла при решении задач.

Теория Операторы цикла

№	Синтаксис	Пример

1	**While** условие цикла **do begin**
	Операторы; Итерация переменной;
	end;
		a:=0;
		while a<5 do begin
		c := 0.5 *a ; a:=a+1; end;

2	**repeat**
	операторы; итерация переменной;
	until условие выхода из цикла;
		A:=-1;
		repeat
		c: = 3*a+4; a:=a+1;
		until a>3;

3	**for** переменная:=нач.зн. **to** кон.знач. **do begin** операторы; **end;**	for i:=0 to10 do begin s:=i*I; l:=l+2;end;

Задания для самостоятельного решения

Вар	Задача 1	Задача 2 Для ряда, члены которого вычисляются по формуле, подсчитать сумму первых 10 членов ряда и вывести их на экран.
1	Напечатать таблицу значений функции $y=2x^2-3x+4$от -2 до 5 с шагом 0,5	$a_n = (-1^n)\frac{1}{(n+1)(n+2)(n+3)}$
2	Напечатать таблицу значений функции $y=-2x^2-3x+4$от -2 до 2 с шагом 0,1	$a_n = (-1^n)\frac{n+1}{n^3+2}$
3	Напечатать таблицу значений функции $y=3x^2-2x+5$от -5 до 5 с шагом 0,5	$a_n = (-1^n)(1-\frac{2n-1}{2(n+1)})$
4	Напечатать таблицу значений функции $y=4x^2-x+4$от -1,2 до 1,5 с шагом 0,05	$a_n = (-1^n)\frac{n^2+1}{n^3+3}$

5	Напечатать таблицу значений функции $y=-x^2-3x+4$от 2 до 5 с шагом 0,3	$a_n = (-1^n)\ \frac{n+1}{3^n+2^n}$
6	Напечатать таблицу значений функции $y=2x^2+3x+4$от -3 до 3 с шагом 0,5	$a_n = (-1^n)(1-\ \frac{(n+1)^2}{(n+2)^2})$
7	Напечатать таблицу значений функции $y=x^2-3x-4$от -2 до 2 с шагом 0,5	$a_n = (-1^n)\frac{2^n}{n^{n+1}+1}$
8	Напечатать таблицу значений функции $y=3x^2-3x+1$от -1 до 3 с шагом 0,5	$a_n = (-1^n)(1 - \frac{2^n}{2^n+1}\ )$
9	Напечатать таблицу значений функции $y=2x^2-2x-4$от -4 до 3 с шагом 0,5	$a_n = (-1^n)\frac{n+1}{2^{n-1}}$
10	Напечатать таблицу значений функции $y=2x^2-3x-7$от -2 до 5 с шагом 0,5	$a_n = (-1^n)\ \frac{n+1}{n^3-n^2+1}$
11	Напечатать таблицу значений функции $y = x^2 - 3x + 4$ от -2 до 5 с шагом 0,2	$a_n = (-1^n)\ \frac{2^{n+1}}{2^{2n}+1}$
12	Напечатать таблицу значений функции $y=-x^2+3x-4$от 2 до 5 с шагом 0,5	$a_n = (-1^n)\ \frac{1}{n^2+2^n}$
13	Напечатать таблицу значений функции $y=-x^2-x+2$от 2 до 5 с шагом 0,4	$a_n = (-1^n)\frac{1+3n}{3^n}$
14	Напечатать таблицу значений функции $y=-2x^2-3x+4$от 2 до 5 с шагом 0,8	$a_n = (-1^n)\ \frac{n+1}{n^3+1}$

Дополнительное задание: Написать программу, которая выводит на экран квадрат Пифагора — таблицуумножения.

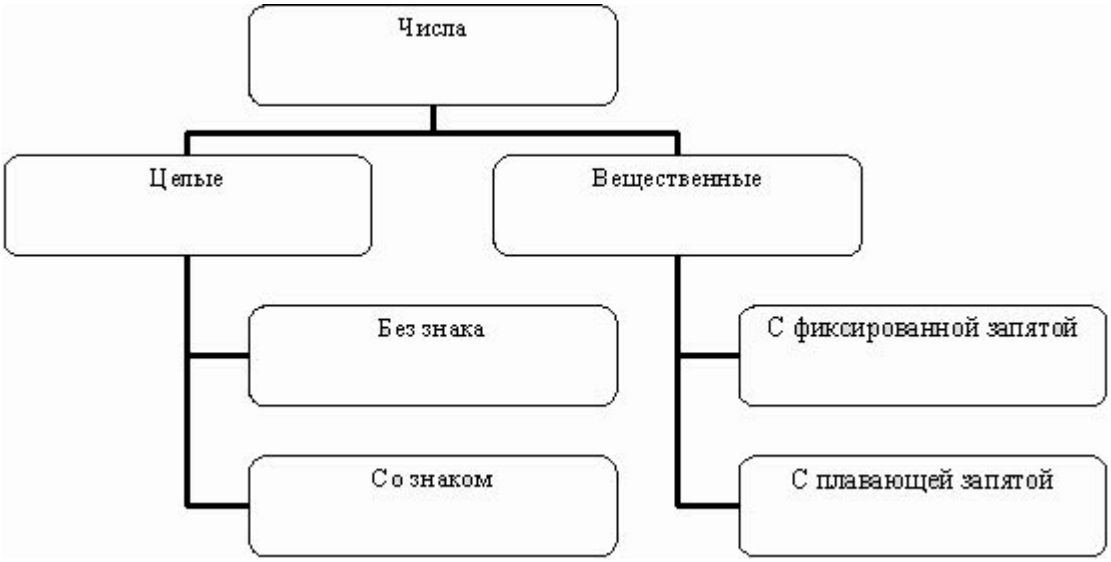
Практическое занятие №4 по теме: «Алгоритмы выполнения арифметических действий с многоразрядными числами.»

Вопросы, выносимые на обсуждение:

- Представление целых чисел без знака и со знаком в памятиЭВМ.
- Формат чисел с фиксированной и плавающейзапятой.
- Арифметика нормализованныхчисел.

Краткие теоретические сведения Числовая информация была первым видом информации, который начали обрабатывать ЭВМ. В данное время все виды информации (текст, звук, графика и т.д.) кодируются с помощью чисел.

Существует большое разнообразие типов и представлений чисел:



Кроме этого, современные процессоры способны обрабатывать несколько форматов каждого вида данных. Память ЭВМ построена из двоичных запоминающих элементов, обладающих устойчивыми состояниями, одно из которых соответствует "0", другое - "1".
Таким физическим элементом (битом) в ЭВМ представляется каждый разряд двоичного числа.
Совокупность определенного количества таких элементов служит для представления многоразрядных двоичных чисел и составляет разрядную сетку ЭВМ.
Каждая группа из 8 бит (байт) пронумерована. Номер байта называется его адресом. ЭВМ производит операции над числами параллельно, сразу в некотором количестве разрядов. Совокупность разрядов, обрабатываемых компьютером параллельно, называется словом. Для разных ЭВМ длина слова различна - два, четыре или восемь байт.

Целые числа без знака

Беззнаковые целые числа представляются в машине наиболее просто. Достаточно перевести требуемое число в двоичную форму и дополнить полученный результат нулями слова до стандартной разрядности.
Рассматриваемые коды называют прямыми кодами.
Минимальное число без знака при любом количестве разрядов N равняется нулю; максимальное, образованное всеми единицами, легко вычислить по формуле
$Max=2^N-1$, где N-число разрядов.
Переполнение - ситуация, когда при последовательном увеличении числа на единицу мы доходим до максимального значения, прибавляем единицу еще раз и получаем необычный результат: все существующие разряды обнуляются, а единица переноса в старший разряд теряется.
В результате после максимального значения мы скачком возвращаемся к минимальному. При вычитании единицы имеет место обратная картина. Это происходит из-за того, что слово памяти может состоять только из конечного числа битов. Таким образом, числовая система ЭВМ является конечной и цикличной. Таким образом, числовая система ЭВМ является конечной и цикличной.

Целые числа со знаком

Добавление отрицательных значений приводит к появлению некоторых новых черт. Ровно половина из всех 2^N чисел теперь будут отрицательными; учитывая необходимость нулевого значения, положительных будет на единицу меньше.
Для того чтобы различать положительные и отрицательные числа, в двоичном представлении чисел выделяется знаковый разряд. По традиции для кодирования знака используется самый

старший бит, причем нулевое значение в нем соответствует знаку "+", а единичное - знаку "-".

Для упрощения конструкций арифметических устройств вычислительных машин все арифметические операции, как правило, сводятся к сложению (операция вычитания) или к сериям сложений и сдвигов (операции умножения и деления). Умножение и деление могут быть сведены к сложению и вычитанию на основании всем известных алгоритмов выполнения этих действий. Замена операции вычитания (алгебраического сложения) на арифметическое сложение в ЭВМ осуществляется с помощью обратного и дополнительного кодов. При этом учитываются знаки чисел, автоматически определяется знак результата и признак возможного переполнения разрядной сетки.

Дополнительный код положительного числа совпадает с этим числом, а для отрицательного числа он равен дополнению его величины до числа, возникающего при переполнении разрядной сетки q^n, где q - основание системы счисления, n - число разрядов в разрядной сетке, q^n называют константой образования дополнительного кода.

Операцию C = A - B, где A и B - целые положительные числа в любой системе счисления, можно представить в виде:

$C=A-B=A+(-B)=A+(-B)+q^n-q^n=A+(q^n-B)-q^n$ где q - основание системы счисления;

n - число разрядов в разрядной сетке;

A - первое слагаемое;

q^n - B - дополнительный код числа - B (второго слагаемого);

q^n - константа, ликвидирующая единицу переполнения.

Пример 1.

$A = 95_{10}$, $B = 43_{10}$, n = 2, q=10.

Найти C=A - B. $[-B_{дк}] = 100 - 43 = 57$; $C= 95 + [-B_{дк}] - 100 = 95 + 57 - 100 = 152 - 100 = 52$.

Единицу в старшем разряде суммы можно просто зачеркнуть.

Для перехода к дополнительным кодам в рассмотренных выше примерах мы использовали операцию вычитания. Необходимо найти способ получения дополнения произвольного числа X до q^n без использования вычитания. Перепишем разность C=A-B в следующем виде: $C=A-B=A+(-B)=A+(-B)+q^n-q^n+1-1=A+(q^n-1-B)-q^n+1$.

Выражение q^n - 1 - B определяет число получаемое заменой каждой цифры числа B на ее дополнение до цифры q - 1. Так, 24510 + = 245 + 1000 - 1 - 245 = 245 - 754 = 999. называется обратным кодом числа B, q^n - 1 - константой образования обратного кода

Основные сведения о машинных кодах:

1. прямойкодравныхпоабсолютнойвеличинечиселотличаетсятолькознаковым разрядом;
2. прямой, обратный и дополнительный коды положительного числасовпадают;
3. прямойкодиспользуетсядляхраненияположительныхиотрицательныхчиселв запоминающих устройствах и для представления положительных чисел при выполнении операций в арифметическихустройствах;
4. обратный и дополнительный коды позволяют заменить операциювычитания сложением;
5. обратныйкодотрицательногодвоичногочислаобразуетсяизпрямогокодаравногопо абсолютной величине положительного числа путем замены всех единиц нулями, а нулей единицами, включая знаковый разряд;
6. дополнительныйкодотрицательногочислаобразуетсяпутемдобавленияединицык младшему разряду обратного кода этого жечисла.

Действия над машинными кодами чисел

Будем считать, что для представления чисел используется 8-разрядная сетка: 1 разряд знаковый и 7 разрядов значащих.

Пример. Даны числа A=34 и B=30. Найдем A + B, A - B, B - A.

Составим машинные коды этих чисел:
$[A_{пк}] = 0\ 0100010_2$, $[A_{ок}] = 0\ 0100010_2$, $[A_{дк}] = 0\ 0100010_2$;
$[-A_{пк}] = 1\ 0100010_2$, $[-A_{ок}] = 1\ 1011101_2$, $[-A_{дк}] = 1\ 1011110_2$;
$[B_{пк}] = 0\ 0011110_2$, $[B_{ок}] = 0\ 0011110_2$, $[B_{дк}] = 0\ 0011110_2$;
$[-B_{пк}] = 1\ 0011110_2$, $[-B_{ок}] = 1\ 1100001_2$, $[-B_{дк}] = 1\ 1100010_2$;
Найдем сумму A + B:
Для этого найдем $[A_{дк}]+[B_{дк}]$
$0\ 0100010_2$
+
00011110_2

$0\ 1000000_2$
Результат получен в дополнительном коде. Но дополнительный код положительного числа совпадает с прямым. Таким образом, окончательный ответ: 1000000_2, или 64_{10}.
Найдем разность A - B, заменив операцию вычитания сложением: A - B = A + (-B).
Для найдем [Адк]+ [-Вдк]:
$0\ 0100010_2$
+
11100010_2

$10\ 0000100_2$
Единица, вышедшая за знаковый разряд, отбрасывается. Нуль в знаковом разряде свидетельствует о том, что результат положительный и равен 100_2 или 4_{10}.
Найдем разность B - A, заменив операцию вычитания сложением: B - A = B + (-A). Для этого найдем[Вдк]+[-Адк]:
00011110_2
+
11011110_2

$1\ 1111100_2$
Результат отрицательный и получен в дополнительном коде. Перейдем к обратному коду, вычитая из младшего разряда 1:
$1\ 1111100_2$
-
1_2

$1\ 1111011_2$

С помощью операции инверсии, не затрагивая знаковый разряд, от обратного кода перейдем к прямому: 1 0000100.
Окончательный результат:- 100_2, или - 4_{10}.

Представление чисел в формате с фиксированной точкой.

При представлении в ЭВМ чисел в естественной форме устанавливается фиксированная длина разрядной сетки. Занятую (точку) можно зафиксировать в начале, середине или конце разрядной сетки. При этом распределение разрядов между целой и дробной частями остается неизменным для любых чисел. В связи с этим существует другое название естественной формы представление в формате с фиксированной точкой (запятой).
Любые правильная дробь и целое число в двоичной системе счисления имеют,

соответственно, вид:

$$Адр = \pm\sum_{i=1}^{n} a_{-i} \cdot 2^{-i} \quad \text{и} \quad Ац = \pm\sum_{i=1}^{n} a_i \cdot 2^{n-i}$$

Анализ этих формул показывает:

1. минимальное положительное число равно 0,00...01 для дробных и единице дляцелых чисел; числа, по абсолютной величине меньшие A_{min} (единицы младшего разряда n-разрядной машинной сетки), называются машиннымнулем;
2. максимальное положительное число равно 0,1 1...1 $(1-2^{-n})$ для дробных чисел (вовсех разрядах должны быть записаны единицы) и 11...1 $(2^n - 1)$ для целыхчисел;
3. общее количество чисел, которое можно записать в n-разрядную сетку, равно $M = 2^n$.

Представление чисел в формате с плавающей запятой

Любое вещественное число x, представленное в системе счисления с основанием N, можно записать в виде:

$$x=\pm mN^{\pm p}$$

где |m| - мантисса, p - характеристика (или порядок) числа. Эту форму еще называют экспоненциальная.

Если |m| < 1, то запись числа называется нормализованной слева.

Следующие примеры показывают, как можно представить любое число в форме с плавающей запятой:

а) в десятичной системе счисления	б) в двоичной системе счисления
$372,95 = 0,37295 \times 10^3$; $25 = 0,025 \times 10^3 = 0,25 \times 10^2$; $0,0000015 = 0,15 \times 10^{-5} = 0,015 \times 10^{-4}$.	$11010,1101 = 0,0110101101 \times 2^6 = 0,110101101 \times 2^5$; $0,011011 = 0,11011 \times 2^{-1}$; $0,1 = 0,1 \times 2^0$. (здесь порядок определяет, на сколько разрядов необходимо осуществить сдвиг относительно запятой).

Число называют нормализованным справа, если после запятой в мантиссе стоит не нуль. Например, числа $0,00076_{10}$ и $0,00011_2$, представленные соответственно в виде $0,076 \times 10^{-2}$ и $0,011 \times 2^{-2}$, не являются нормализованными справа, а в виде $0,76 \times 10^{-3}$ и $0,11 \times 2^{-3}$ являются такими.Принормализованнойформемантиссаотвечаетусловию$q^{-1} \le |m| < 1$.Вдальнейшем под нормализацией числа будем понимать нормализациюсправа.

Упражнения к практическому занятию

1. Записатьдополнительныйкодотрицательногочисла-2002для16-разрядного компьютерногопредставления.
2. Найдите десятичные представления чисел, записанных в дополнительномкоде:
    1. 11111000
    2. 10011011
    3. 11101001
3. Выполните операцию сложения в соответствующей системесчисления
    1. $0,397621*10^3+0,237900*10^1$
    2. $0,101101_2*2^{100}+0,110001_2*2^{101}$
4. Выполните операцию вычитания в соответствующей системесчисления:
    1. $0,982563*10^2-0,745623*10^2$
    2. $0,110011_2*2^{-1}-0,100101_2*2^{01}$
5. Выполните операциюумножения:
    1. $0,235601*10^2*0,859842*10^3 2.0,101010*10^2*0,311325*10^1$
6. Выполните операциюделения

1. $0{,}01178005*10^{2}:0{,}235601*10^{3}$
2. $0{,}4818115*10^{-4}:0{,}963623*10^{-2}$

7. Используя дополнительный код, найдите следующие разности в десятичнойсистеме счисления:
   1. 53-35
   2. 102-51
8. Выполните действия над машинными кодами чисел с плавающей точкой в 32-разрядномформате
   Дано: A=125,75 B=-50 Найти X=A+B

Практическое занятие №5 по теме: «Перестановки, сочетания и размещения без повторений.»

Цель работы:

На конкретных примерах научиться находить число всех k-элементных подмножеств множества из n элементов и число перестановок множества, состоящего из n элементов. Закрепить полученные знания по данной тематике при решении занимательных задач. Научиться использовать в своей работе компьютер как инструмент.

Содержание работы:

Задача 1: A – количество студентов; B – посещают математический кружок; C – физический; D – не посещают ни одного из кружков. Сколько студентов посещают и математический, и физический кружок? Сколько студентов посещают только математический кружок? |A|=37; |B|=20;

Решение:

$|C|=16;|D|=17.$ $|BUC|=|B|+|C|-|B\cap C|$ $|\overline{BUC}|=17$

$|\overline{BUC}|=|A|-|BUC|=37-17=20$ $20=20+16-|B\cap C|$ $|B\cap C|=16$

$|B\backslash C|=|B|-|B\cap C|=20-16=4$

Ответ: 16; 4

Задача 2: Сколькими способами из A студентов можно выбрать делегацию, если делегация состоит из B студентов? |A|=42; |B|=8.

Решение:

$C_n^k=\frac{n!}{k!(n-k)!}$ k=8;n=42 $C_n^k=\frac{42!}{8!34!}$

Ответ: $\frac{42!}{8!34!}$

Задача 3: Сколько имеется четырехзначных чисел, у которых каждая следующая цифра меньше предыдущей?

Решение:

$C_n^k=\frac{n!}{k!(n-k)!}$ n=10;k=4; $C_n^k=\frac{10!}{4!(10-4)!}=210$

Ответ: 210

Задача 4: На собрании должны выступить 4 человека A,B,C,D. Сколькими способами их можно разместить в списке ораторов, если B не может выступить до того момента, пока не выступит A.

Решение:

$\frac{4!}{2!}$ =12

CDAB DCAB CADB CABD DACB DABC ABCD ABDC ACBD ACDB ADBC ADCB

Ответ: 12

Задача 5: сколькими способами могут разместиться А покупателей в очереди в кассу. |A|=15

Решение:

$P_n^k = \frac{n!}{(n-k)!}$ n=15;k=15 $P_n^k = \frac{15!}{0!}$=15!

Ответ: 15!

Практическое занятие №6 по теме: «Перестановки , сочетания и размещения с повторением»

Цель работы:

На конкретных примерах научиться находить число перестановок и сочетанийс повторениями. Закрепить полученные знания по данной тематике при решении занимательныхзадач.

Содержаниеработы:

Задача 1: Сколько различных слов можно составить, переставляя буквы слова КОМБИНАТОРИКА.

Решение:

$P_n\{k_1, k_2, \ldots, k_m\} = \frac{n!}{k_1! k_2! \ldots k_m!}$

n=13 {2, 2, 1, 1, 2, 1, 2, 1, 1}

$k_1=2, k_2=2, k_3=1, k_4=1, k_5=2, k_6=1, k_7=2, k_8=1, k_9=1$

$P_n = \frac{13!}{2!2!1!1!2!1!2!1!1!}$ = 389188800

Ответ: 389188800

Задача 2: Сколькими способами можно выбрать А одинаковых или разных пирожных в кондитерской, где есть В разных сортов пирожных. A=5; B=12

Решение:

$U_n^k = n^k$

n=12; k=5

$U_n^k = 12^5$= 248832

Ответ: 248832

Задача 3: Пусть A={a, b, c, d, e} и B={a, b, c, f, m, k}. Указать все элементы множества AxB – декартова произведения множеств A и B.

Решение:

AxB= {(a, a), (a, b), (a, c), (a, f), (a, m), (a,k),
(b, a), (b, b), (b, c), (b, f), (b, m), (b,k),
(c, a), (c, b), (c, c), (c, f), (c, m), (c, k),
(d, a), (d, b), (d, c), (d, f), (d, m), (d, k),

Практическое занятие №7 по теме: «Алгоритмическое перечисление(генерирование) некоторых комбинаторных объектов»

Цель работы: приобретение навыков практического использования алгоритмов над объектами дискретнойматематики

Оборудование: ПК IBM PC

Содержание работы:

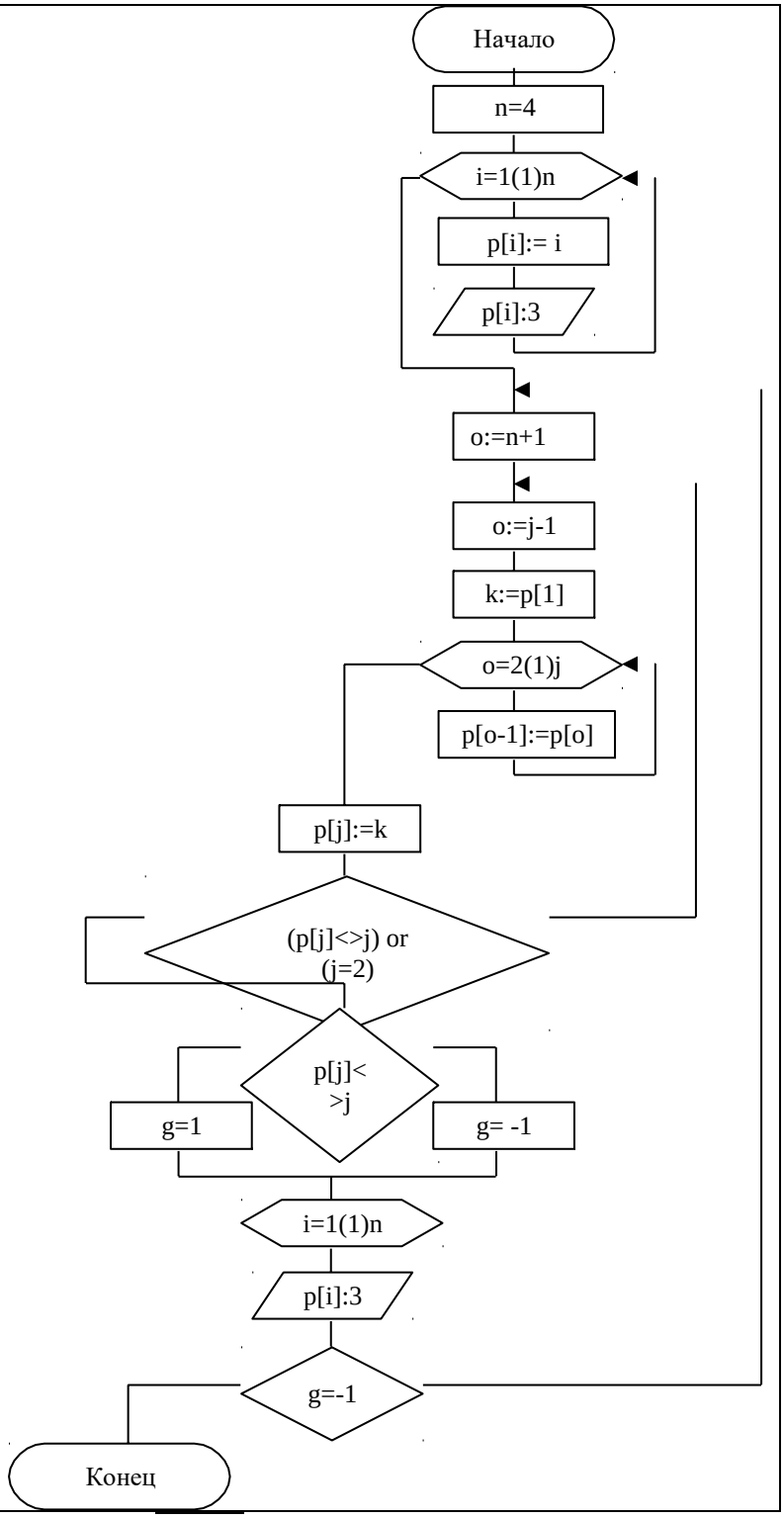
1. Составить блок-схему алгоритма по своемуварианту
2. Составить тестовый пример по своемуварианту
3. Ответить на контрольныевопросы
4. Оформить отчет ираспечатать

Контрольные вопросы:

1. Что такое "алгоритмическоеперечисление"?
2. С какого вектора начинают работать обагенератора?
3. По какому признаку заканчивает работу генераторперестановок?
4. По какому признаку заканчивает работу генератор сочетаний?

```
program perest;
uses crt;
const n=4;
type mas=array [1..n] of integer;
var p:mas; i,g:integer;
procedure genper (n:word; var p:mas; var
g:integer);
var o,j,k:word;
begin
o:=n+1;
repeat
     o:=j-1;
     k:=p[1];
     for o:=2 to j do
        p[o-1]:=p[o];
     p[j]:=k;
until (p[j]<>j) or (j=2);
if (p[j]<>j) then g:=1 else g:=-1 end;

begin
clrscr;
for i:= 1 to n do
   begin
   p[i]:= i;
   write (p[i]:3)
   end;
writeln;
repeat genper(n,p,g);
for i:=1 to n do write (p[i]:3);
writeln;
until g=-1
end.
```



Ответы:

1. Алгоритмическое перечисление (выборка) – способ задания множества при помощи некоторогоалгоритма.
2. Оба генератора начинают работать с вектора123…n.
3. Генераторперестановок заканчивает работу, когда все элементы встают на свои первоначальные места.
4. Генератор сочетаний заканчивает работу, когда на первом месте стоит число, равное (n-k+1), гдеn–максимальноезначениеэлемента,ak–количествоэлементоввектора.

Практическое занятие №8 по теме: «Алгоритмы сортировки массивов. Сортировка вставками.»

Ход занятия:

1. Изучить алгоритм сортировки массивавставками.
2. Научиться применять алгоритм при решении задач.

ТЕОРИЯ

Сортировка вставками — примитивный алгоритм сортировки с высокой вычислительной сложностью: O(n^2).

Плюсы:

- эффективен на небольших наборах данных, на наборах данных до десятков элементов может оказатьсялучшим;
- эффективен на наборах данных, которые уже частичноотсортированы;
- это устойчивый алгоритм сортировки (не меняет порядок элементов, которые уже отсортированы);
- может сортировать список по мере егополучения;
- использует O(*1*) временной памяти, включая стек.

Минусы:

- Очень высокая вычислительная сложность алгоритмаO(n^2).

Описание алгоритма

На каждом шаге алгоритма мы выбираем один из элементов входных данных и вставляем его на нужную позицию в уже отсортированном списке, до тех пор, пока набор входных данных не будет исчерпан. Метод выбора очередного элемента из исходного массива произволен; может использоваться практически любой алгоритм выбора. Обычно (и с целью получения устойчивого алгоритма сортировки), элементы вставляются по порядку их появления во входном массиве. Приведенный ниже алгоритм использует именно эту стратегию выбора.

Анализ алгоритма

Время выполнения алгоритма зависит от входных данных: чем большее множество нужно отсортировать, тем большее время выполняется сортировка. Также на время выполнения влияет исходная упорядоченность массива. Так, лучшим случаем является отсортированный массив, а худшим — массив, отсортированный в порядке, обратном нужному. Временная сложность алгоритма при худшем варианте входных данных — $\theta(n^2)$.

Псевдокод

```
Вход: массив A, состоящий из элементов A[1], A[2], ..., A[n]
for i = 2 to n do begin
   key:=A[i];       j := i – 1;
   while (j >= 1) and (A[j] > key) do begin
      A[j+1] :=A[j];
      j := j – 1;end;
   A[j+1] := key;
end;
```

Задание для самостоятельного выполнения.

Задать случайным образом одномерный массив целых чисел. Число элементов массива ввести с клавиатуры. Выполнить сортировку массива по возрастанию, затем по убыванию, используя алгоритм сортировки вставками.

Практическое занятие №9 по теме: «Алгоритмы сортировки массивов. Сортировка методом пузырька.»
Ход занятия:

1. Изучить алгоритм сортировки массивапузырьком.
2. Научиться применять алгоритм при решении задач.

ТЕОРИЯ

Сортировка простыми обменами, сортиро$_0$ вка пузырько$_0$ м (англ. *bubble sort*) — простой алгоритм сортировки. Для понимания и реализации этот алгоритм — простейший, но эффективен он лишь для небольших массивов. Сложность алгоритма: O(n^2).

Алгоритм считается учебным и практически не применяется вне учебной литературы, вместо него на практике применяются более эффективные алгоритмы сортировки. В то же время метод сортировки обменами лежит в основе некоторых более совершенных алгоритмов, таких как шейкерная сортировка, пирамидальная сортировка и быстрая сортировка.

Описание алгоритма
Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов. Проходы по массиву повторяются N-1 раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован. При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший элемент перемещается на одну позицию к началу массива («всплывает» до нужной позиции как пузырёк в воде, отсюда и название алгоритма). При обмене элементов массива обычно используется "буферная" (третья) переменная, куда временно помещается значение одного из элементов.

Псевдокод

На входе: массив A[N], состоящий из N элементов, с нумерацией от A[1] доA[N]

```
for j=1 to n-1 dobegin
for i=1 to n-j  dobegin
if a[i]>A[i+1] then begin
k:=a[i]; a[i]:=a[i+1]; a[i+1]:=k;
end; end; end;
```

Задание для самостоятельного выполнения.
Задать случайным образом одномерный массив целых чисел. Число элементов массива ввести с клавиатуры. Выполнить сортировку массива по возрастанию, затем по убыванию, используя алгоритм сортировки пузырьком.

Практическое занятие №10 по теме: «Алгоритмы сортировки массивов. Сортировка методом Шелла.»
Ход занятия:

1. Изучить алгоритм сортировки массивапузырьком.
2. Научиться применять алгоритм при решениизадач.

ТЕОРИЯ

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

Аналогичный метод усовершенствования пузырьковой сортировки называется сортировка расчёской.

Описание алгоритма

При сортировке Шелла сначала сравниваются и сортируются между собой значения, отстоящие один от другого на некотором расстоянии d. После этого процедура повторяется для некоторых меньших значений d, а завершается сортировка Шелла упорядочиванием элементов при $d = 1$(то есть обычной сортировка вставками). Эффективность сортировки Шеллавопределённыхслучаяхобеспечиваетсятем,чтоэлементы«быстрее»встаютнасвои места (в простых методах сортировки, например, пузырьковой, каждая перестановка двух элементовуменьшаетколичествоинверсийвспискемаксимумна1,априсортировкеШелла это число может быть больше).

Невзирая на то, что сортировка Шелла во многих случаях медленнее, чем быстрая сортировка, она имеет ряд преимуществ:

- отсутствие потребности в памяти подстек;
- отсутствие деградации при неудачных наборах данных — быстрая сортировка легко деградирует до O(n²), что хуже, чем худшее гарантированное время для сортировки Шелла.

Пример, иллюстрирующийсортировку Шелла.

```
Procedure Sort( var a : seq);
Var d, i, t : integer;
  k : boolean; { признак перестановки }
 begin
 d:=N div 2; { начальное значение интервала }
 while d>0 do begin { цикл с уменьшением интервала до 1 }
  { пузырьковая сортировка с интервалом d }
  k:=true;
  while k do begin { цикл, пока есть перестановки }
   k:=false; i:=1;
   for i:=1 to N-d do begin
    { сравнениеэл-товнаинтервале d } if
    a[i]>a[i+d] then begin
     t:=a[i]; a[i]:=a[i+d]; a[i+d]:=t; { перестановка }
     k:=true; { признакперестановки }
     end; { if ... }
    end; { for ... }
  end; { while k }
 d:=d div 2; { уменьшениеинтервала }
 end; { while d>0 }
```

```
end;
```

Задание для самостоятельного выполнения.
Задать случайным образом одномерный массив целых чисел. Число элементов массива ввести с клавиатуры. Выполнить сортировку массива по возрастанию, затем по убыванию, используя алгоритм сортировки методом Шелла.

Практическое занятие №11 по теме: «Алгоритмы сортировки массивов. Сортировка методом выборки.»
Ход занятия:

1. Изучить алгоритм сортировки массива методомвыборки.
2. Научиться применять алгоритм при решениизадач.

ТЕОРИЯ

Шаги алгоритма:

1. находим номер минимального значения в текущемсписке
2. производимобменэтогозначениясозначениемпервойнеотсортированнойпозиции (обмен не нужен, если минимальный элемент уже находится на даннойпозиции)
3. теперьсортируемхвостсписка,исключивизрассмотренияужеотсортированные элементы

Для реализации устойчивости алгоритма необходимо в пункте 2 минимальный элемент непосредственно вставлять в первую неотсортированную позицию, не меняя порядок остальных элементов.

Пример. Сортировка выбором по возрастанию.

```
program Sort_Vybor1;
 var A:array[1..100] of integer;
 N,i,m,k,x : integer;

begin
 write('количество элементов массива ');
 read(N);
 for i:=1 to n do read(A[i]);
 for k:=n downto 2 do {k- количество элементов для поиска max }
   begin
    m:=1; { m - место max }
    for i:=2 to k do if A[i]>A[m] then m:=i;
    {меняем местами элементы с номером m и номером k}
    x:=A[m]; A[m]:=A[k]; A[k]:=x;
   end;
 for i:=1 to n do write(A[i],' '); {упорядоченныймассив}
end.
```

Задание для самостоятельного выполнения.
Задать случайным образом одномерный массив целых чисел. Число элементов массива ввести с клавиатуры. Выполнить сортировку массива по возрастанию, затем по убыванию, используя алгоритм сортировки методом выборки.

Практическое занятие №12 по теме: «Алгоритмы сортировки массивов. Сортировка методом слияний.»
Ход занятия:

1. Изучить алгоритм сортировки массива методомслияний.
2. Научиться применять алгоритм при решениизадач.

ТЕОРИЯ

Сортировка слиянием (англ. *merge sort*) — алгоритм сортировки, который упорядочивает списки (или другие структуры данных, доступ к элементам которых можно получать только последовательно, например — потоки) в определённом порядке. Эта сортировка — хороший пример использования принципа «разделяй и властвуй». Сначала задача разбивается на несколько подзадач меньшего размера. Затем эти задачи решаются с помощью рекурсивного вызова или непосредственно, если их размер достаточно мал. Наконец, их решения комбинируются, и получается решение исходной задачи.

Пример сортировки слиянием. Сначала делим список на кусочки (по 1 элементу), затем сравниваем каждый элемент с соседним, сортируем и объединяем. В итоге, все элементы отсортированы и объединены вместе.

Для решения задачи сортировки эти три этапа выглядят так:

1. Сортируемый массив разбивается на две части примерно одинаковогоразмера;
2. Каждаяизполучившихсячастейсортируетсяотдельно,например—темжесамым алгоритмом;
3. Два упорядоченных массива половинного размера соединяются водин.

1.1. - 2.1. Рекурсивное разбиение задачи на меньшие происходит до тех пор, пока размер массива не достигнет единицы (любой массив длины 1 можно считать упорядоченным).

3.1. Соединение двух упорядоченных массивов в один.
Основную идею слияния двух отсортированных массивов можно объяснить на следующем примере. Пусть мы имеем два подмассива. Пусть также, элементы подмассивов в каждом из этих подмассивов отсортированы по возрастанию. Тогда:
3.2. Слияние двух подмассивов в третий результирующиймассив.
На каждом шаге мы берём меньший из двух первых элементов подмассивов и записываем его в результирующий массив. Счетчики номеров элементов результирующего массива и подмассива из которого был взят элемент увеличиваем на 1.
3.3. "Прицепление" остатка.
Когда один из подмассивов закончился, мы добавляем все оставшиеся элементы второго подмассива в результирующий массив.

Прежде, чем говорить непосредственно о методе сортировки слиянием, рассмотрим следующую задачу. Требуется объединить («слить») два упорядоченных фрагмента массива A A[k],..., A[m] и A[m+1],..., A[q] в один A[k],...,A[q] тоже упорядоченный ($k \le m \le q$).

Чтобы решить эту задачу, поступим следующим образом: создадим вспомогательный массив D и будем сравнивать очередные элементы каждого фрагмента. Тот элемент, который окажется меньше, будем переносить в массив D. При этом следует не забыть переписать в D оставшиеся элементы того фрагмента, который не успел себя исчерпать.

```
Procedure S1(k, m,q: integer; Var A: Marray);
Var
```

```
      i, j, t: integer;
          D: Marray;
Begin
 i:=k; j:=m+1; t:=1;
 While (i<=m) and (j<=q) Do
    Begin (* пока не закончился хотя бы один массив*)
        If A[i]<= A[j]
          Then
     Begin D[t]:=A[i]; Inc(i);end
           Else
     Begin D[t]:= A[j]; Inc(j); end;
Inc(t);
    end;
  While i<=m Do
     Begin D[t]:= A[i]; Inc(i); Inc(t) End;
  While j<=q Do
     Begin D[t]:= A[j]; Inc(j); Inc(t) End;
   For i:=1 to t-1 Do A[k+i-1]:=D[i];
end;
```

Задание для самостоятельного выполнения.

Задать два одномерных массива целых чисел. Число элементов каждого массива ввести с клавиатуры. Слить массивы, располагая элементы в полученном массиве по возрастанию. Модифицировать алгоритм, чтобы полученный массив был отсортирован по убыванию.

Практическое занятие №13 по теме: «Алгоритмы на графах. Поиск в графе.»

Ход занятия:

1. Разобрать суть метода поиска вграфе.
2. Разобратьпроцедурупоиска.Написатьпрограммуипроверитьеработудля конкретногографа.

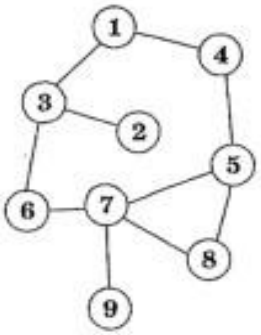
Практическое занятие №14 по теме: «Алгоритмы на графах. Поиск в глубину. »

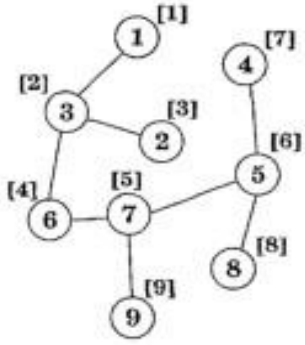
Ход занятия:

3. Разобрать суть метода поиска вглубину.
4. Разобратьпроцедурупоискавглубину.Написатьпрограммуипроверитьее работу для конкретногографа.

Идея метода. Поиск начинается с некоторой фиксированной вершины *v*. Рассматривается вершина *u*, смежная с *v*. Она выбирается. Процесс повторяется с вершиной *u*. Если на очередном шаге мы работаем с вершиной *q* и нет вершин, смежных с *q* и не рассмотренных ранее (новых), то возвращаемся из вершины *q* к вершине, которая была до нее. В том случае, когда это вершина *v*, процесс просмотра закончен. Очевидно, что для фиксации признака, просмотрена вершина графа или нет, требуется структура данных типа: *Nnew : Array[1..N]Of Boolean.*

Пример. Пусть граф описан матрицей смежности *A*. Поиск начинается с первой вершины. На левом рисунке приведен исходный граф, а на правом рисунке у вершин в скобках указана та очередность, в которой вершины графа просматривались в процессе поиска в глубину.





Логика.

```
Procedure    Pg(v:Integer);{*Массивы Nnew
      u A глобальные.*}
 Var     j:Integer;
 Begin
  Nnew[v]:=False;          Write(v:3);
  For j:=1  To  N  Do  If  (A[v,j ]<>0)  And  Nnew[j]
        Then  Pg (j) ;
 End;
```

Фрагмент основной логики.

```
...
```

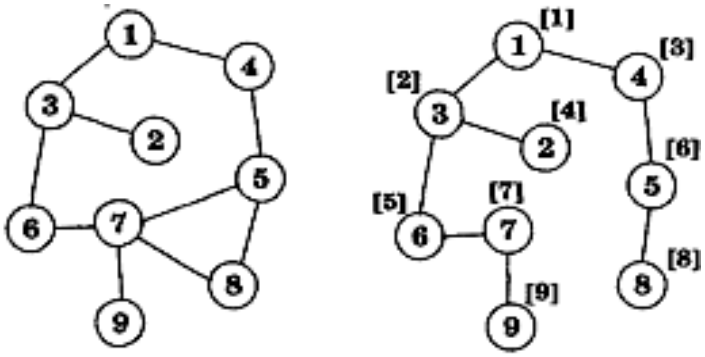
Практическое занятие №15 по теме: «Алгоритмы на графах. Поиск в ширину.»

Ход занятия:

5. Разобрать суть метода поиска вширину.
6. Разобратьпроцедурупоискавширину.Написатьпрограммуипроверитьее работу для конкретногографа.

Идея метода. **Суть (в сжатой формулировке) заключается в том, чтобы рассмотреть все вершины, связанные с текущей. Принцип выбора следующей вершины — выбирается та, которая была раньше рассмотрена. Для реализации данного принципа необходима структура данных очередь.**

Пример. Исходный граф на левом рисунке. На правом рисунке рядом с вершинами в скобках указана очередность просмотра вершин графа.



Приведем процедуру реализации данного метода обхода вершин графа.

Логика просмотра вершин.

```
Procedure      Pw(v:Integer);
  Var Og:Array[1..N] Of 0..N;{*Очередь*}
      yk1,yk2:Integer;{*Указатели  очереди,
      yk1 - запись; yk2 - чтение. *}
      j                                -.Integer;
 Begin
  FillChar(Og,SizeOf(Og),0)  ;
  yk1:=0;yk2:=0;{*Начальнаяинициализация.  *}
  Inc(yk1);Og[yk1]:=v;Nnew[v]:=False;{*В  очередь
          – вершину v. *}
  While yk2<yk1 Do Begin {*Пока очередь не пуста. *}
   Inc(yk2);v:=Og[yk2];Write(v:3);{*"Берем"
          элемент из очереди. *}
   For j:=1 To N Do {*Просмотр всех вершин,
          связанных с вершиной v. *}
    If  (A[v,j]<>0)  And Nnew[j] Then Begin {*Если
          вершина ранее не просмотрена, то заносим
          её номер в очередь. *}
     Inc       (yk1);Og[yk1]:=j;Nnew[j]:=False;
    End;
   End;
  End;
```

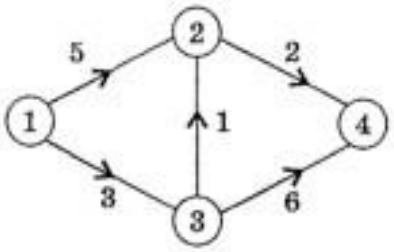
Практическое занятие №16 по теме: «Алгоритмы на графах. Кратчайшие пути.»
Цель занятия:

1. Рассмотреть методику нахождения кратчайшего пути вграфе.
2. Рассмотретьипроверитьработупрограммыпоискакратчайшегопутивграфе.

Постановка задачи. Выводпути.

Дан ориентированный граф $G=<V,E>$, веса дуг — $A[i,j]$ ($i,j=1..N$, где N — количество вершин графа), начальная и конечная вершины — s, $t \in V$. Веса дуг записаны в матрице смежности A, если вершины i и y не связаны дугой, то $A[i,j]=\infty$. Путь между s и t оценивается $\sum_{i,j \in \text{пути}} A[i,j]$. Найти путь с минимальной оценкой.

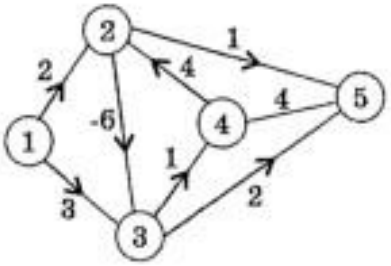
Пример. Кратчайший путь из 1 в 4 проходит через 3-ю и 2-ю вершины и имеет оценку 6.



Особый случай — контуры с отрицательной оценкой.

Пример.

При $s=1$ и $t=5$, обходя контур $3 \to 4 \to 2 \to 3$ достаточное число раз, можно сделать так, что оценка пути между вершинами 1 и 5 будет меньше любого целого числа. Оценку пути назовем его весом или длиной. Будем рассматривать только графы без контуров отрицательного веса.



Нам необходимо найти кратчайший путь, т. е. путь с минимальным весом, между двумя вершинами графа. Эта задача разбивается на две подзадачи: сам путь и значение минимального веса. Обозначим ее через *D[s,t]*. Неизвестны алгоритмы, определяющие только *D[s,t]*, все они определяют оценки от вершины s до всех остальных вершин графа. Определим D как *Array[1..N] Of Integer*. Предположим, что мы определили значения элементов массива *D* — решили вторую подзадачу. Определим сам кратчайший путь. Для *s* и *t* существует такая вершина *v*, что *D[t]=D[v]+A[v,t]*. Запомним *v* (например, в стеке). Повторим процесс поиска вершины *u*, такой, что *D[v]=D[u]+A[u,v]* и так до тех пор, пока не дойдем до вершины с номером *s*. Последовательность *t, v, u, .,., s* дает кратчайший путь.

```
Procedure  Way(s,t:Integer);{*A, - глобальные
           структуры данных. St - локальная структура
           данных для хранения номеров вершин. *}
 Var       v,u:Integer;
  Procedure Print; {*Выводит содержимое St.*}
   Begin

   End;
  Begin
   <почистить St>;
   <занести вершину с номером t в St>;
   v:=t;
   While vOs Do Begin
    u:=<номер вершины, для которой D[v]=D[u]+A[u,v]>;
   <занести вершину с номером v в St>;

 v:=u;
 End;
End;
```

Итак, путь при известном *D* находить мы умеем. Осталось научиться определять значения кратчайших путей, т. е. элементы массива *D*. Идея всех известных алгоритмов заключается в следующем. По данной матрице весов *A* вычисляются первоначальные верхние оценки. А затем пытаются их улучшить до тех пор, пока это возможно. Поиск улучшения, например для *D[vJ*, заключается в нахождении вершин *u*, таких, что *D[u]+A[u,v]<D[v]*. Если такая вершина *и* есть, то значение *D[vJ* можно заменить на *D[u]+A[u,v]*.

Практическое занятие №17 по теме: «Определение связности. Циклы. Нахождение Эйлерова цикла»

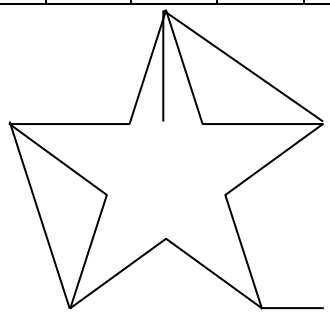
Цель: разобраться со структурой данных типа дерево. Научиться составлять программы на языке Паскаль с использованием структурных данных типа дерево, закрепить навыки по составлению алгоритма с графами.
Задание: составить алгоритм и программу, которая определяет, является ли заданный граф Эйлеровым, вывести граф на печать, проверить результат с заранее составленным тестовым примером.

Ответы на контрольные вопросы:

1. Графы имеющие Эйлеровы циклы (простые циклы, содержащие все ребра графа) называются Эйлеровымиграфами.
2. Гамильтоновым циклом называется простой цикл, проходящий через всевершины графа.
3. Граф называется циклически связным, если через любые две вершины рассматриваемого графа может проходить постойцикл.
4. Граф называется связным, если у него нет изолированныхвершин.
5. Маршрут называется цепью, если каждое ребро встречается в нем не болееераза.
6. Маршрутомназываетсятакаяпоследовательностьребер(e_0,e_1,…,e_{n-1},e_n),чтокаждые 2 соседних ребра смежные, т.е. $\forall i$ e_{i-1} и e_i имеют общую инцидентнуювершину.
7. Простым циклом называют циклический маршрут, если он является простойцепью.

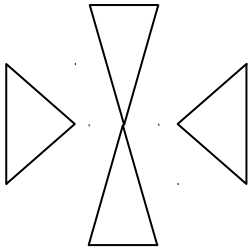
1	2	5	4	1	6	1	4	7	2	8	9	9	10	10
2	3	6	3	5	1	7	5	2	8	3	3	4	4	5

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



9	9	9	9	9	9	9	9	1	3	5	7
1	2	3	4	5	6	7	8	2	4	6	8

9	2	1	9	4	3	9	6	5	9	8	7	9



```
uses crt;
var v_n,v_t:byte;
    mass1,mass2:array [1..100,1..2] of byte;
    stek1,stek2:array [1..100] of byte;
    num1,num2,i,j,k,m:integer;
begin
 clrscr;
 write ('Vvedite kol-vo reber: ');
 read (k);
{---------vvodreber --------- }
 for i:=1 to k do
  begin
   write ('Vvedite ',i,' paru vershin: ');
   readln (mass1[i,1],mass1[i,2]);
   mass2[i,1]:=mass1[i,1];
   mass2[i,2]:=mass1[i,2];
  end;
{----------------main-----------------}
 v_n:=mass1[1,1];
 v_t:=mass1[1,2];
 stek1[1]:=v_n;
 stek1[2]:=v_t;
 num1:=3;
 num2:=1;
 mass1[1,1]:=0;
 mass1[1,2]:=0;

repeat
  m:=0;
  for i:=2 to k do
   begin
    if mass1[i,1]=v_t then
     begin
      if mass1[i,2]=v_n then
       begin
       stek2[num2]:=v_n;
       inc(num2);
        v_n:=v_t;
       end else
       begin
        stek1[num1]:=mass1[i,2];
        inc(num1);
        v_t:=mass1[i,2];
       end;
      mass1[i,1]:=0;
      mass1[i,2]:=0;
      i:=k;
      m:=1;
     end;
    if mass1[i,2]=v_t then
```

```
    begin
     if mass1[i,1]=v_n then
      begin
      stek2[num2]:=v_n;
      inc(num2);
       v_n:=v_t;
      end else
      begin
       stek1[num1]:=mass1[i,1];
       inc(num1);
       v_t:=mass1[i,1];
      end;
     mass1[i,1]:=0;
     mass1[i,2]:=0;
     i:=k;
     m:=1;
    end;
  end;
 if m=0 then
  begin
  dec(num1);
   stek2[num2]:=stek1[num1];
   stek1[num1]:=0;
   v_n:=stek1[num1-1];
   v_t:=stek1[num1-1];
   inc(num2);
  end;
until num2>k+1;
{---------vyvodspiskareber ---------- }
clrscr;
for i:=1 to k do write (mass2[i,1],' ');
writeln;
for i:=1 to k do write (mass2[i,2],' ');
writeln;
writeln;
{---------vyvodvershingrafa----------}
for i:=1 to k+1 do write (stek2[i],' ');
readkey;
end.
```

Практическое занятие №18 по теме: «Метод локальной оптимизации»
Цель занятия:

3. Рассмотреть методику приближенного решения задачикоммивояжера.
4. Рассмотреть и проверить работупрограммы.

Попытаемся отказаться от перебора вариантов при решении задачи о поиске гамильтоновых циклов или, если граф «нагруженный», о поиске путей коммивояжера. При больших размерностях задачи (значение *N)* переборные схемы не работают — остаются только приближенные методы решения. Суть приближенных методов заключается в быстром поиске первого решения (первой оценки), а затем в попытках его улучшения. В методе локальной оптимизации в этих попытках просматриваются только соседние вершины графа (города) найденного пути.

Шаг 1. Получить приближенное решение (первую оценку).

Шаг 2. Пока происходит улучшение решения, выполнять следующий шаг, иначе перейти на шаг 4.

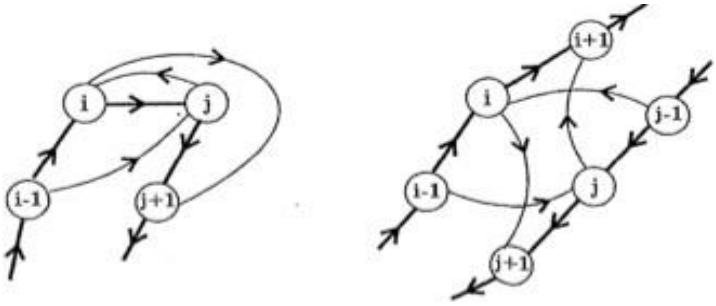
Шаг 3. Для всех пар номеров городов i,j, удовлетворяющих неравенству $(1 \le i < j \le N)$, проверить: $d_{i-1,i}+d_{i,j}+d_{j,j+1} > d_{i-1,j}+d_{j,i}+d_{i,j+1}$ для смежных городов, т. е.

$j=i+1$, $d_{i-1,i}+d_{i,i+1}+d_{j-1,j}+d_{j,j+1} > d_{i-1,j}+d_{j,i+1}+d_{j-1,i}+d_{i,j+1}$

для несмежных городов.

Примечание

На рисунке даны графические иллюстрации первого и второго неравенств. «Жирными» линиями обозначены участки старых маршрутов, «тонкими» — новых.



Если одно из неравенств выполняется, то найдено лучшее решение. Следует откорректировать ранее найденное решение и вернуться на шаг 2.

Шаг 4. Закончить работу алгоритма.

Для реализации логики достаточно матрицы расстояний *(A)* и массива для хранения пути коммивояжера *(Way)*. Вид общей логики:

```
Begin
 Init;{*Ввод из файла матрицы расстояний,
       инициализация глобальных переменных. *}
 One Way;{*Поиск первого варианта пути
          коммивояжера.*}
 Local;{*Локальная оптимизация.  *}
 Out;{*Вывод результата. *}
End.
```

Продолжим уточнение логики. Работа процедур *Init, One-Way, Out* достаточно очевидна. Естественным приемом является вынесение ее в самостоятельную часть работы школьников на занятии. Нам необходимо уточнить процедуру *Local.* Работаем не с частностями, а на содержательном уровне. Предположим, что мы имеем функции *Best1* и *Best2,* их параметры — индексы элементов массива *Way,* определяющих номера городов в пути коммивояжера, а выход естественный — истина или ложь, в зависимости от того, выполняются неравенства или нет. Рассуждаем дальше. Если неравенство выполняется, то нам необходимо изменить путь (соответствующие элементы массива *Way).* Пока не будем заниматься деталями. Пусть эту работу выполняет процедура *Swap,* ее параметры — индексы элементов массива *Way).* Эта процедура, вероятно, должна сообщать о том, что она «что-то изменила», ибо нам необходимо продолжать работу до тех пор, пока что-то меняется, происходят улучшения. Итак, логика процедуры *Local.*

```
ProcedureLocal;
 Var i,j:Integer;
     Change                                    '.Boolean;
 <здесь функции Best1u Best2, а также процедура
 Swap>;
 Begin
  Repeat
   Change:=False;
   For i:=1 To N-1 Do
    For j:=i+1 To N Do
     If i=j+1 Then Begin If Best1(i,j)Then
          Swap (i,j);End
      Else If (i=1) And (j=N) Then Begin If
        Best1 (i,j) Then Swap(i,j) End {*Об этой
             проверке лучше первоначально умолчать,
             чтобы было о чем спросить, "если совсем
             будет плохо" - все понимают и нет
             вопросов.*}
            Else If Best2 (i,j) Then Swap(i,j);
  Until Not(Change);{*Излогики неясно - кто
        изменяет значение переменной Change на True.
        Определите, учитывая, что функции Best1,
        Best2 и процедура Swap локальные по
        отношению к процедуре Local.*}
 End;
```

Практическое занятие №19 по теме: «Алгоритмы вычислительной геометрии. Базовые процедуры»

Цель работы:

1. Рассмотреть базовые процедуры работы с типамиданных.

Точка на плоскости описывается парой вещественных чисел. Введем соответствующую запись. При использовании вещественного типа операции сравнения лучше оформить специальными функциями. Причина известна, на вещественных типах данных в системе программирования Паскаль нет отношения порядка, поэтому записи вида a=b, где a и b вещественные числа, лучше не использовать. Приведем пример, объясняющий это положение. Пусть для хранения вещественного типа используются два десятичных разряда для порядка и шесть разрядов для хранения мантиссы. Есть два числа: a=0,34567*104 и fc=0,98765*10-4. При сложении и вычитании осуществляется: выравнивание порядков, сложение (вычитание) мантисс и нормализация результата. После выравнивания порядков

число Ь=0,0000000098765*104.После сложения мантисс имеем: 0,3456700098765*104. Так как для хранения мантиссы у нас шесть десятичных разрядов (в реальном компьютере тоже есть ограничение, например, для типа Real в Турбо Паскале — 11-12 цифр), то результат округляется и он равен 0,345670*104.
Итак, а+Ь=а при fe>0!!! Компьютерная арифметика не совпадает с обычной арифметикой. На протяжении всей главы используются описанные ниже типы данных, функции и процедуры.

```
Type     Real=Extended;
TPoint=Record  x,  y:  Real;  End;
Const        Eps:  Real=1e-3;{*Точность вычисления.*}
ZeroPnt:   TPoint  =  (X:0;  Y:0);{*Точка
           с  координатами  0,0.*}
```

Реализация операций сравнений: =, <, >, <, >.

```
Function  RealEq(Const a,  b:  Real):  Boolean;
          {*Строго равно. *}
 Begin
  RealEq:=Abs(a-b)<=_Eps;
 End;
```

Остальные функции RealMore (строго больше), RealLess(строго меньше), RealMoreEq (больше или равно), RealLessEq (меньше или равно) пишутся по аналогии. Для определения максимального из двух вещественных чисел приведенные выше функции уже можно использовать.

```
Function  RealMax(Const a,  b:  Real):  Real;
      {*Максимальное  из  двух  вещественных  чисел.  *}
 Begin
  If  RealMore  (a,  b)  Then  RealMax:=a  Else  RealMax:=b;
 End;
```

Аналогично пишется функция RealMin (поиск минимально-го числа) и функция EqPoint совпадения двух точек на плоскости.

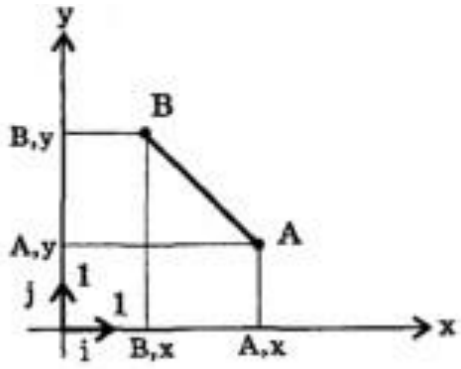
```
Function  EqPoint(ConstA,  B:  TPoint):Boolean;
      {*Совпадают ли точки ?*}
 Begin
  EqPoint:=RealE(A.x,  B.x)  And  RealEq(A.y, B.y);
 End;
```

Функция вычисления расстояния между двумя точками на плоскости имеет вид.

```
Function  Dist(ConstA,  B:  TPoint):  Real;
          {*Расстояние между двумя точками
           на плоскости. *}
 Begin
  Dist:=Sqrt(Sqr(A.x-B.x)  +  Sqrt  (A.y-B.y));
          {*Вспомним теорему Пифагора - «квадрат
```

*гипотенузы прямоугольного треугольника равен сумме квадратов катетов». *)*

End;



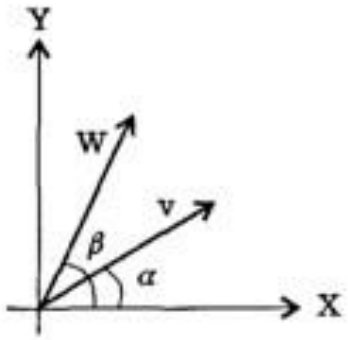
Каждую точку плоскости можно считать вектором, начало которого находится в точке (0,0). Обозначим координаты точки (вектора) *v* через (v_x,v_y), $w=(w_x,w_y)$, $q=(q_x,q_y)$. **Для векторов** определены следующие операции:

- **сумма:** $q=v+w$, $q_x=v_x+w_x$, $q_y=v_y+w_y$;
- **разность:** $q=v-w$, $q_x=v_x-w_x$, $q_y=v_y-w_y$;
- скалярное произведение: $v*w=v_xw_x+vw$;
- векторное произведение: $v\times w=(v_xw_y-v_yw_x)k$.

Скалярным произведением вектора *v* на вектор *w* называется число, определяемое равенством $v*w=|v|*|w|*\cos(\varphi)$, где символом | | обозначается длина вектора, а φ — угол между векторами. Скалярное произведение обращается в нуль в том и только том случае, когда по крайней мере один из векторов является нулевым или когда векторы *v* и *w* ортогональны. Для скалярного произведения справедливы следующие равенства: $(v+z)*w=v*w+v*z$ **и** $(\lambda\ v)*w=v*(\lambda\ w)=\lambda(v*w)$. **Если векторы** заданы своими координатами в ортонормированном базисе *i,j*, т. е. $v=(v_x,v_y)w=(w_x,w_y)$,то формула для скалярного произведения с учетом перечисленных свойств записывается

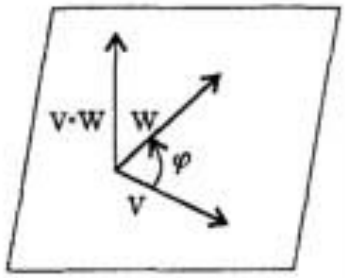
$$v*w=v_xw_x(i*i)+v_yw_x(j*i)+v_xw_y(i*j)+v_yw_y(j*j)=$$
$$v_xw_x(i*i)+v_yw_y(j*j)= \quad v_xw_x+v_yw_y.$$

Для скалярного произведения векторов (v,α) и (w,β) справедливо соотношение $v*w=v_x w_x+v_y w_y=v\cos(\alpha)w\cos(\beta)+v\sin(\alpha)w\sin(\beta)=vw\cos(\alpha-\beta)$. При общей начальной точке у двух векторов скалярное произведение больше нуля, если угол между векторами острый, и меньше нуля, если угол тупой.

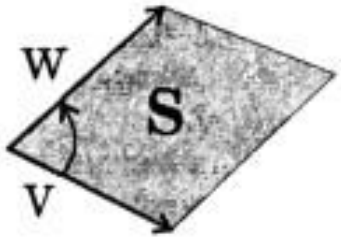


Векторным произведением $v\times w$ называется вектор, такой, что:

- длина вектора $v\times w$ равна $|v|*|w|*\sin(\varphi)$;
- вектор $v\times w$ перпендикулярен векторам v и w, т. е. перпендикулярен плоскости этих векторов;



- вектор *vxw* направлен так, что из конца этого вектора кратчайший поворот от *v* к *w* виден происходящим против часовой стрелки.



Длина векторного произведения численно равна площади параллелограмма, построенного на перемножаемых векторах $\boldsymbol{v}$ и $\boldsymbol{w}$ как на сторонах.

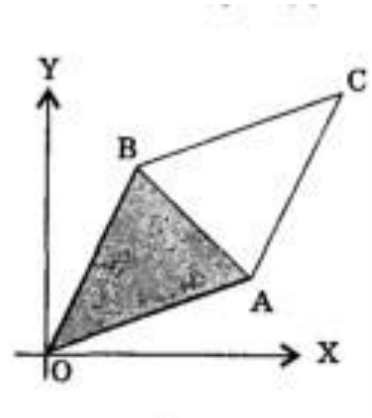
Пусть векторы *v* **и** ***w*** заданы своими координатами в базисе *i*, ***j***, ***k*** (v_x,v_y,v_z) **и** (w_x,w_y,w_z). **Векторное произведение** ***vxw*** = $(v_x\boldsymbol{i}+v_y\boldsymbol{j}+v_z\boldsymbol{k})\times(w_x\boldsymbol{i}+w_y\boldsymbol{j}+w_z\boldsymbol{k})=v_xw_x\,\boldsymbol{i}\times\boldsymbol{i} + v_xw_y\,\boldsymbol{i}\times\boldsymbol{j} + v_xw_z\,\boldsymbol{i}\times\boldsymbol{k} + v_yw_x\,\boldsymbol{j}\times\boldsymbol{i} + v_yw_y\,\boldsymbol{j}\times\boldsymbol{j} + v_yw_z\,\boldsymbol{j}\times\boldsymbol{k} + v_zw_x\,\boldsymbol{k}\times\boldsymbol{i} + v_zw_y\,\boldsymbol{k}\times\boldsymbol{j} + v_zw_z\,\boldsymbol{k}\times\boldsymbol{k}$. **Векторные** произведения координатных ортов равны: $\boldsymbol{i}\times\boldsymbol{i}=0$, $\boldsymbol{j}\times\boldsymbol{j}=0$, $\boldsymbol{k}\times\boldsymbol{k}=\boldsymbol{0}$, $\boldsymbol{i}\times\boldsymbol{j}=\boldsymbol{k}$, $\boldsymbol{j}\times\boldsymbol{i}=-\boldsymbol{k}$, $\boldsymbol{j}\times\boldsymbol{k}=\boldsymbol{i}$, $\boldsymbol{k}\times\boldsymbol{j}=-\boldsymbol{i}$. $\boldsymbol{k}\times\boldsymbol{i}=\boldsymbol{j}$ **и** $\boldsymbol{i}\times\boldsymbol{k}=-\boldsymbol{j}$. **Формула имеет вид:** ***vxw*** $=(v_yw_z-v_zw_y)\boldsymbol{i}+(v_zw_x-v_xw_z)\boldsymbol{j}+(v_xw_y-v_yw_x)\boldsymbol{k}$**Её** можно записать в другом, более компактном виде (через определитель):

$$\boldsymbol{v}\times\boldsymbol{w}=\begin{vmatrix}\boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k}\\ v_x & v_y & v_z\\ w_x & w_y & w_z\end{vmatrix}$$

Раскрывая определитель, мы получаем ту же самую формулу. Учитывая наш случай, а именно то, что координаты *v* и *w* векторов равны нулю, получаем $\boldsymbol{v}\times\boldsymbol{w}=(v_xw-v\,w_x)\boldsymbol{k}$. После несложных преобразований получаем

$\boldsymbol{v}\times\boldsymbol{w}=(v*\cos(\alpha)*w*\sin(\beta)-v*\sin(\alpha)*w*\cos(\beta))\boldsymbol{k}=(v*w*\sin(\beta-\alpha))\boldsymbol{k}$.

Векторное произведение ненулевых векторов равно нулю тогда и только тогда, когда векторы параллельны.



Пример. Найти площадь треугольника *OAB*. Очевидно, что его площадь равна половине площади параллелограмма *OACB*, а площадь последнего равна модулю векторного произведения векторов *OA* и *OB*. **Находим,** $S_{OACB}=A_x*B_y-A_y*B_x$, **отсюда** $S_{OAB}=(1/2)*(A_x*B_y-A_y*B_x)$. **Отметим, что значение** S_{OBA} **имеет** другой знак (ориентированная площадь).

Рассмотрим процедуры работы с векторами в декартовой системе координат. Тип данных для описания векторов имеет **вид:** ***Type TVecCart=TPoint; (Record x, y: Real;End)***

```
Procedure AddVecCart(Consta, b: TVecCart; Var c:
TVecCart) ; {*Сумма двух векторов в декартовой
                системе координат.*}
```

```
Begin
 c.x:=a.x+b.x;c.y:=a.y+b.y;
End;
```

Аналогично пишутся и процедуры вычисления разности двух векторов *SubVecCart*, умножения вектора на число *MulKVecCart*, а также функции нахождения скалярного произведения *SkMulCart* и векторного произведения *VectMulCart* векторов в декартовой системе координат. Например:

```
Function SkMulCart (Const a, b: TVecCart): Real;
                {* Скалярное произведение.*}
Begin
 SkalarMulCart:=a.x*b.x+a.y*b.y;
End;
```

Вектор v можно задать в полярной системе координат через его длину (модуль) v и угол a относительно оси X. Координаты полярной системы координат (v, α) и прямоугольной декартовой (v_x, v) связаны соотношениями: $v_x=v*cos(\alpha)$, $v_y=v*sin(\alpha)$, и $v=Sqrt(v_x^2+v^2)$, $tan(\alpha)=v_y/v_x$. Тип данных для описания вектора в полярной системе координат имеет вид *Type TVecPol=Record rst, angle: Real;End;*.

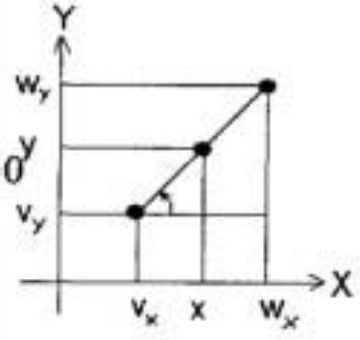
Операции перевода из одной системы координат в другую реализуются с помощью следующих процедур и функций. Функция определения угла используется при переводе в полярную систему координат.

```
Function GetAngle(Const x, y: Real): Real;
          {*Возвращает угол от 0 до 2*Pi
           (в радианах) . *}
 Var rs, c: Real;
 Begin
  rs:=Sqrt(Sqr(x) + Sqr(y));{*Вычисляется
          расстояние от точки (0,0) до (x,y) . Можно
          воспользоваться функцией Dist.*}
  If RealEq(rs, 0) Then GetAngle:=0
   Else Begin
    c:=x/rs;
    If RealEq(c, 0) Then c:=Pi/2
     Else c:=ArcTan (Sqrt (Abs(1-Sqr(c)))/c);
    If RealLess (c, 0) Then c:=Pi+c;
    If RealLess (y, 0) Then c:=2*Pi-c;
    GetAngle:=c;
   End;
 End;

Procedure CartToPol(Const a: TVecCart; Var b:
 TVecPol);{*Перевод из декартовой системы
          координат в полярную. *}
 Begin
  b.rst:=Sqrt(Sqr(a.x) + Sqr(a.y));b.angle
   :=GetAngle(a.x, a.y);
 End;
Procedure PolToCart (Const a: TVecPol; Var b: TVecCart);
          {*Перевод из полярной системы координат
          в декартовую. *}
 Begin
  b.x:=a.rst*cos (a.angle);b.y:=a.rst*sin(a.angle);
 End;
```

Практическое занятие №20 по теме: «Алгоритмы вычислительной геометрии. Прямая линия и отрезок прямой. Треугольник»

Прямая линия на плоскости, проходящая через две точки (v_x,v_y) и (w_x,w_y), определяется следующим линейным уравнением от двух переменных: $(w_x-v_x)*(y-v_y)=(w_y-v_y)*(x-v_x)$. После преобразований получаем: $-(w-v)*x+(w_x-v_x)*y+(w_y-v_y)*v_x-(w_x-v_x)*v_y=0$ или, после соответствующих обозначений: $A*x+B*y+C=0$, где $A=v.y-w.y$, $B=w.x-v.x$, $C=-(v.x*(v.y-w.y)+v.y*(w.x-v.x))$.



Параметры прямой линии описываются с помощью следующего **типа данных:** ***Type TLine=Record A, B, C: Real;End;***

```
Procedure Point2ToLine (Const v, w: TPoint; Var L:
  Thine) ;  {*Определение уравнения прямой
              по координатам двух точек. *}
Begin
 L.A:= v.y - w.y;
 L.B:= w.x -v.x;
 L.C:=-(v.x*L.A    +    v.y*L.B);
End;
```

Если прямые заданы с помощью уравнений $A_1*x+B_1*y+C_1=0$ и $A_2*x+B_2*y+C_2=0$, то точка их пересечения, в случае ее существования, определяется по формулам $(A_1*B_2-A_2*B_1<>0)$:

$$x=(C,*B_2-C_2*B_1)/(A_1*B_2-A_2*B_1),$$

$$y=(A_1*C_2-A_2*C_1)/(A_1*B_2-A_2*B_1)$$

обычным решением системы двух уравнений с двумя неизвестными. Функция *CrossLine* определяет существование точки пе-

ресечения (с учетом тех погрешностей, которые определяются введенной точностью вычислений), а в процедуре *Line2ToPoint* вычисляются координаты точки пересечения.

```
Function CrossLine(ConstL1, L2: TLine) : Boolean;
        {*Определяет существование точки пересечения
        двух прямых. Значение функции равно True,
        если точка есть, и False, если прямые
        параллельны. *}
 Var st: Real;
 Begin
  st:=Ll.A*L2.B-L2.A*Ll.B;
  CrossLine:=Not RealEq(st,0) ;
End;
Procedure Line2ToPoint(ConstL1, L2: TLine; Var P:
    TPoint);{*Определение координат точки
            пересечения двух линий. *}
```

```
Var st: Real;
Begin
 st:=L1.A*L2.B-L2.A*L1.B;
 P.X:=(L1.C*L2.B-L2.C*L1.B)/st;
 P.Y:=(L1.A*L2.C-L2.A*L1.C)/st;
End;
```

Координаты точек отрезка можно задать двумя параметрическими уравнениями от одной независимой переменной *t*:

$$x=v_x+(w_x-v_x)*t,$$

При $0 \le t \le 1$ точка *(x,y)* лежит на отрезке, а при $t<0$ или $t>1$ — вне отрезка на прямой линии, продолжающей отрезок.

Для проверки принадлежности точки *P* отрезку необходимо выполнение равенства в уравнении $(w_x-v_x)*(y-v_y)=(w_y-v_y)*(x-v_x)$ и принадлежность координаты, например, *x* точки *P* интервалу, определяемому координатами *x* концов отрезков.

```
Function AtSegm(Const A, B, P: TPoint): Boolean;
        {*Проверка принадлежности точки P отрезку AB. *}
Begin
 If EqPoint(A,B) Then AtSegm:=EqPoint(A, P)
          {*Точки A и B совпадают, результат
           определяется совпадением точек A и P.*}
  Else
   AtSegm:=RealEq((B.x-A.x*(P.y-A.y), (B.y-A.y) *
     (P.x-A.x))
   And ( (RealMoreEq(P.x, A.x) And RealMoreEq (B.x,
     P.x)) Or
```

```
    (RealMoreEq(P.x,B.x)    And    RealMoreEq(A.x, P.x)));
End;
```

Измените функцию для случая, когда отрезок описывается параметрическими уравнениями.

Процедура, с помощью которой определяются координаты точки пересечения двух отрезков, пишется на основе ранее приведенных процедур построения прямой по двум точкам и нахождения точки пересечения прямых (не утверждается то, что точка пересечения принадлежит отрезкам).

```
Procedure FindPointCross(ConstfL, fR, sL, sR:
   TPoint; Var rs: TPoint);{*Если точки пересечения
       не существует, то значение rs равно (0,0).*}
Var L1, L2: TLine;
Begin
 Point2ToLine (fL, fR, L1);
 Point2ToLine (sL, sR, L2);
 If CrossLine (L1,L2) Then Line2ToPoint (L1, L2, rs)
   Else   rs:=ZeroPnt;
End;
```

Примечание

Процедуру следует доработать в том случае, если точка пересечения прямых совпадает с точкой (0,0).

Пусть два отрезка находятся на одной прямой. Требуется определить их взаимное расположение.

```
Function SegmLineCross(Const fL, fR, sL, sR:
Tpoint): Byte; {*Отрезки находятся на одной
        прямой, проверка их взаимного расположения.
        Результат равен 0, если отрезки пересекаются
        в одной точке, 1 — не имеют пересечения и 2 —
        есть пересечение более чем в одной точке. *}
 Var Minf, Maxf,Mins,Maxs : Real;
 Begin
  Minf:=RealMin(Dist(fL,ZeroPnt),Dist  (fR,ZeroPnt));
  Maxf:=RealMax(Dist(fL,ZeroPnt),Dist  (fR,ZeroPnt));
  Mins:=RealMin(Dist(sL,ZeroPnt),Dist(sR,ZeroPnt));
  Maxs:=RealMax(Dist    (sL,ZeroPnt),Dist(sR,ZeroPnt));
  If RealEq(Minf,Maxs)  Or RealEq(Maxf, Mins)  Then
     SegmLineCross:=0
   ElseIfRealMore(Mins,Maxf) OrRealMore(Minf ,Maxs)
      Then SegmLineCross:=1
         Else SegmLineCross:=2;
End;
```

Даны два отрезка на плоскости, заданные координатами своих концов. Определить их взаимное расположение (до написания функции попробуйте прорисовать все возможные случаи).

```
Function SegmCross(Const fL, fR, sL, sR: Tpoint):
Byte; {* Результат равен 0, если отрезки пересекаются
в одной точке и лежат на одной прямой, 1 - не имеют
пересечения и лежат на одной прямой, 2 - отрезки ле-
жат на одной прямой и есть пересечение более чем в
одной точке, 3 - отрезки лежат на параллельных пря-
мых, 4 - отрезки не лежат на одной прямой и не имеют
точки пересечения, 5 - отрезки не лежат на одной пря-
мой и пересекаются на концах отрезков, 6 - отрезки не
лежат на одной прямой, точка пересечения принадлежит
одному из отрезков и является концом другого отрезка,
7 - отрезки не лежат на одной прямой и пересекаются в
одной точке, не совпадающей ни с одним концом отрез-
ков. *}
   Var rs:TPoint;
       L1,L2:TLine;
   Begin
    Point2ToLine (fL,  fR,  L1);
    Point2ToLine (sL,  sR,  L2);
```

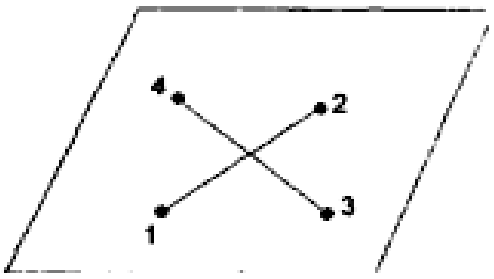
Даны два отрезка на плоскости, заданные координатами своих концов. Определить их взаимное расположение (до написания функции попробуйте прорисовать все возможные случаи).

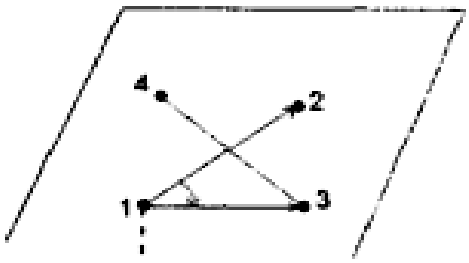
```
Function SegmCross(Const fL, fR, sL, sR: Tpoint): Byte; (* Результат равен 0, если отрезки пересекаются в одной точке и лежат на одной прямой, 1 - не имеют пересечения и лежат на одной прямой, 2 - отрезки лежат на одной прямой и есть пересечение более чем в одной точке, 3 - отрезки лежат на параллельных прямых, 4 - отрезки не лежат на одной прямой и не имеют точки пересечения, 5 - отрезки не лежат на одной прямой и пересекаются на концах отрезков, 6 - отрезки не лежат на одной прямой, точка пересечения принадлежит одному из отрезков и является концом другого отрезка, 7 - отрезки не лежат на одной прямой и пересекаются в одной точке, не совпадающей ни с одним концом отрезков.*)
    Var rs:TPoint;
        L1,L2:TLine;
    Begin
     Point2ToLine (fL, fR, L1);
     Point2ToLine (sL, sR, L2);
```

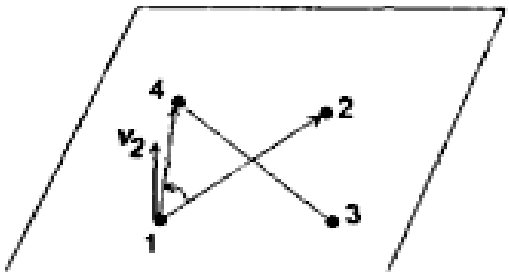
```
If CrossLine (L1,L2) Then Begin
   Line2ToPoint(L1,L2, rs)
   If EqPoint (rs,fL) Or EqPoint(rs,fR) Or EqPoint
      (rs,sL) Or EqPoint (rs,sR)
    Then  SegmCross:=5
    Else  If AtSegm(fL,fR,rs)  And AtSegm(sL,sR,rs)
       Then SegmCross:= 7
          Else If AtSegm(fL,fR,rs) Or AtSegm
             (sL,sR,rs) Then SegmCross:=6
                Else  SegmCross:=4;
   End
 Else   If EqPoint(L1.A*L2.B,L2.A*L1.B)And Not
     (EqPoint (L1.C,L2.C))
          Then  SegmCross:=3
          Else SegmCross:= SegmLineCross ( fL,  fR,
             sL,  sR);
End;
```

Задачу о пересечении отрезков на плоскости можно (естественно) решать по-другому. На рисунке показаны два отрезка. Найдем ряд векторных произведений, они показаны на следующих четырех рисунках.

Рассмотрим произведения длин v_1, v_2, v_3, v_4. Очевидно, что если $v_1*v_2<0$ и $v_3*v_4<0$, то отрезки пересекаются (в произведениях берутся длины векторов v_1, v_2, v_3, v_4). При $v_1*v_2>0$ или $v_3*v_4>0$ отрезки не пересекаются. Рассмотрим варианты, при которых одна из длин равна нулю. Если $v_1=0$, $v_2<>0$ и $v_3*v_4<0$, то третья точка лежит на первом отрезке. При $v_2=0$ и аналогичных условиях четвертая точка лежит на первом отрезке, $v_3=0$ — первая на втором отрезке и при $v_4=0$ — вторая точка на втором отрезке. Особый случай, когда $v_1=0$, $v_2=0$, $v_3=0$, $v_4=0$. Отрезки расположены на одной прямой, необходимы дополнительные проверки для выяснения того, перекрываются они или нет. Все ли особые случаи рассмотрены? Определите значения векторных произведений при совпадении ка-

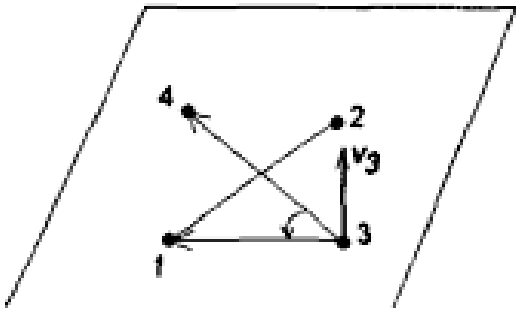


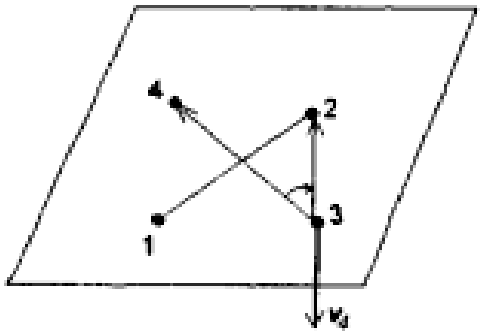




ких-либо двух точек, например 2-й и 3-й.

Разработайте процедуру определения взаимного расположения двух отрезков на плоскости с использованием рассмотренных векторных произведений.





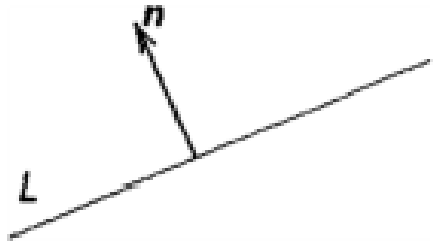
Еще один вариант решения задачи о пересечении отрезков возможен при использовании параметрической формы записи уравнения прямой. Отрезки заданы координатами своих концов: (x_0,y_0)— (x_1,y_1) и (x_2,y_2)- (x_3,y_3). Уравнения имеют вид: $x=x_0+t_1*(x_1-x_0)$ $y=y_0+t_1*(y_1-y_0)$ где $t_1\in[0,1]$, и $x=x_2+t_2*(x_3-x_2)$, $y=y_2+t_2*(y_3-y_2)$, где $t_2\in[0,1]$. В точке пересечения, при её существовании выполняются равенства:

$x_0+t_1*(x_1-x_0) = x_2+t_2*(x_3-x_2)$

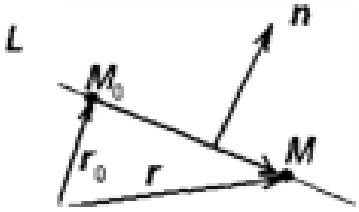
$y_0+t_1*(y_1-y_0) = y_2+t_2*(y_3-y_2)$.

Есть система двух уравнений с двумя неизвестными t_1 и t_2. Если существует единственное решение и $t_1, t_2 \in (0,1)$, то отрезки пересекаются. Если одно из значений t_1 и t_2 равно 0 или 1, а второе принадлежит [0,1], то отрезки пересекаются в одной из вершин, в остальных случаях отрезки не пересекаются. Разработка соответствующей процедуры — ваша задача.

Нормальным вектором (n) к прямой L $(A*x+B*y+C=0)$ называют всякий ненулевой вектор, перпендикулярный этой прямой. Известно, что координаты вектора определяются значениями A и B в уравнении линии L - $n=(A,B)$. Для того, чтобы найти уравнение прямой на плоскости L, проходящей через заданную точку M_0 перпендикулярно заданному направлению $n=(A,B)$, необходимо раскрыть скалярное произведение $(r-r_0, n)=0$. Оно равно $A*(x-x_0) + B*(y-y_0)=0$.




```
Procedure PerLine(Const n: TLine; Const P: Tpoint;
Var L: TLine); {*Линия, перпендикулярная n,
проходящая через точку P.*}
Begin
  L.A:=n.B;
  L.B:=-n.A;
  L.C:=-L.A*P.X-L.B*P.Y
End;
```



Расстояние от точки $P(x_0,y_0)$ до прямой L вычисляется по формуле:
$\rho(P,L)=Abs(L.A*x_0+L.B*y_0+L.C)/(sqrt(L.A^2+L.B^2))$.

```
Function DistPointToLine (Const P: TPoint; Const L:
TLine): Real;{*Расстояние от точки до прямой. *}
Begin
 DistPointToLine:=Abs   ((L.A*P.x+L.B*P.y+L.C))
      /Sqrt(Sqr(L.A)+Sqr(L.B));
End;
```

ТРЕУГОЛЬНИК

Даны три отрезка *a*, *b*, *c*. Для существования треугольника с такими длинами сторон требуется одновременное выполнение условий **(a+b>c)**, **(b+c>a)** и **(a+c>b)**.

```
Function  IsTrian(Const a, b, c: Real): Boolean;
          {*Проверка существования треугольника
           со сторонами a, b, c.*}
Begin
 IsTrian:=RealMore(a+b, c) And RealMore (a+c, b)
     And RealMore(b+c, a);
End;
```

Площадь треугольника. Если известны длины сторон треугольника, то для вычисления площади треугольника обычно используется формула Герона *S=Sqrt(p*(p-a)*(p-b)*(p-c))*, где *p=(a+b+c)/2*.

```
Function    Sq(a,b,c:Real):Real;
 Var  p:Real;
 Begin
  p:=(a+b+c)/2;
  Sq:=Sqrt(p*(p-a)*(p-b)*(p-c));
 End;
```

Возможно использование и других формул:

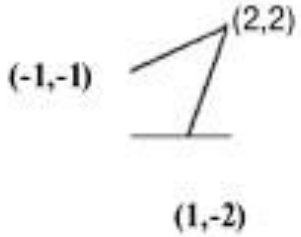
$S = (a*h)/2 = (a*b*\sin(C))/2 = (a*a*\sin(B)*\sin(C))/(2*\sin(A)) = (h*h*\sin(A))/(2*\sin(B)*\sin(C))$,

где большими буквами обозначены углы против соответствующих сторон.

Гораздо интереснее второй способ вычисления площади треугольника — через векторное произведение. Длина вектора дает удвоенную площадь треугольника, построенного на этих векторах. Пусть даны три точки $p_1=(x_1,y_1)$, $p_2=(x_2,y_2)$, $p_3=(x_3,y_3)$ на плоскости, определяющие вершины треугольника. Совместим начало координат с первой точкой. Векторное произведение равно $\begin{vmatrix} i & j & k \\ -x, & y_2-y, & 0 \\ -x, & y_3-y, & 0 \end{vmatrix}$. Вычисление длины вектора равносильно раскрытию определителя $\begin{vmatrix} x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}$ $+(x_2*y_3-x_3*y_2)$(раскрываем по первой строке).

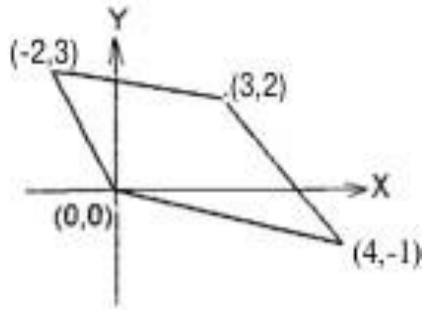
```
Function SquareTrian(Constp1, p2, p3: TPoint) :
  Real; {*Вычисление ориентированной площади
         треугольника.*}
Begin
  SquareTrian := (p1.x*(p2.y-p3.y)-p1.y(p2.x-p3.x)
    +  (p2.x*p3.y-p3.x*p2.y) ) /2;
End;
```

Пример. Вычислим удвоенную площадь треугольника, изображенного на рисунке.



Значение определителя $\begin{pmatrix} 2 & 2 \\ -1 & -1 \\ 1 & -2 \end{pmatrix}$ равно 9. Значение определителя $\begin{pmatrix} 2 & 2 & 1 \\ 1 & -2 & 1 \\ -1 & -1 & 1 \end{pmatrix}$ равно —9. Отличие: в первом случае угол считался против часовой стрелки, во втором — по часовой стрелке.

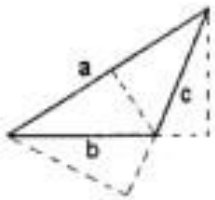
Еще один пример. Определитель $\begin{pmatrix} 4 & -1 & 1 \\ 0 & 0 & 1 \\ -2 & 3 & \end{pmatrix}$ равен 10, а определитель $\begin{pmatrix} 4 & \\ & 2 \\ & 3 \end{pmatrix}$ равен 15. Вывод: при обходе против часовой стрел-



ки получаем положительные величины, а при обходе по часовой стрелке — отрицательные значения.

Замечательные линии и точки в треугольнике

Высоту треугольника, опущенную на сторону a, обозначим через h_a. Через три стороны она выражается формулой



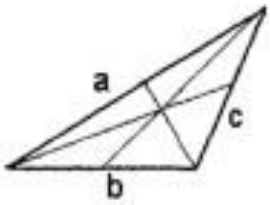
$h_a=2*SQRT(p*(p-a)*(p-b)*(p-c))/a$,

где $p=(a+b+c)/2$.

```
Function GetHeight (Const a, b, c: Real):
Real;{*Вычисление длины высоты, проведенной
из вершины, противоположной стороне треугольника
с длиной a. *}
 Var p: Real;
 Begin
  p:=(a+b+c) /2;
  GetHeight :=2*Sqrt(p* (p-a) * (p-b) * (p-c) ) /a;
 End;
```

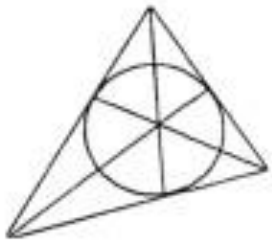
Медианы треугольника пересекаются в одной точке (всегда внутри треугольника), являющейся центром тяжести треугольника. Эта точка делит каждую медиану в отношении 2:1, считая от

вершины. Медиану на сторону *a* обозначим через m_a. Через три стороны треугольника она выражается формулой



$m_a=SQRT(2*b^2+2*c^2-a^2)/2.$

```
Function GetMed(Const a, b, c:
Real): Real; (*Вычисление длины
медианы, проведенной из вершины,
противоположной стороне треугольника с длиной a.
 Begin
  GetMed:=Sqrt(2   *(b*b+c*c)-a*a)/2;
 End;
```



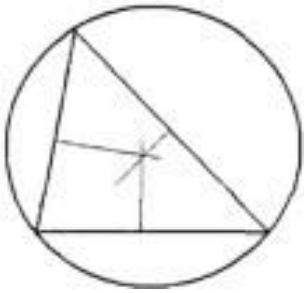
Три биссектрисы треугольника пересекаются в одной точке (всегда внутри треугольника), являющейся центром вписанного круга. Его радиус вычисляется по формуле

$r=SQRT((p-a)*(p-b)*(p-c)/p).$

Биссектрису к стороне *a* обозначим через β_a, ее длина выражается формулой

$fi_a=2*SQRT(b*c*p*(p-a))/(b+c)$, где p — полупериметр.

```
Function GetRadIns(Consta, b, c: Real): Real;
{*Вычисление радиуса окружности, вписанной
в треугольник с длинами сторон a, b, c.*}
 Var p: Real;
 Begin
  p: = (a+b+c) /2;
  GetRadIns :=Sqrt ( (p-a) * (p-b) * (p-c) /p) ;
 End;
Function GetBis(Const a, b, c: Real): Real;
 {*Вычислениедлинысбиссектрисы,проведенной
из вершины, противоположной стороне треугольника
с длиной a . *}
 Var p: Real;
 Begin
  p:=(a+b+c)/2;
  GetBis:=2*Sqrt(b*c*p*(p-a))/(b+c);
 End;
```



Три перпендикуляра к сторонам треугольника, проведенные через их середины, пересекаются в одной точке, служащей центром описанного круга. В тупоугольном треугольнике эта точка лежит вне треугольника, в остроуголь-

ном — внутри, в прямоугольном — на середине гипотенузы. Его радиус вычисляется по формуле

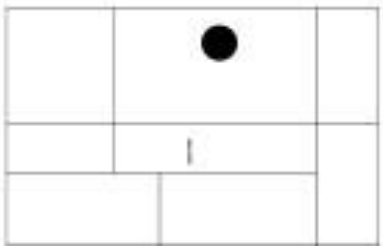
$R=(a*b*c)/(4*SQRT(p*(p-a)*(p-b)*(p-c)))$. **В равнобедренном** треугольнике высота, медиана и биссектриса, опущенные на основание, а также перпендикуляр, проведенный через середину основания, совпадают друг с другом; в равностороннем то же имеет место для всех трех сторон. В остальных случаях ни одна из упомянутых линий не совпадает с другой. Центр тяжести, центр вписанного круга и центр описанного круга совпадают друг с другом только в равностороннем треугольнике.

```
Function GetRadExt(Consta, b, c: Real): Real;
{*Вычисление радиуса окружности, описанной около
треугольника с длинами сторон a, b, c.*}
 Var p: Real;
 Begin
  p:=(a+b+c) /2;
  GetRadExt:=a*b*c/(4*Sqrt (p* (p-a) * (p-b) * (p-c) ) ) ;
 End;
```

Практическое занятие №21 по теме: «Алгоритмы вычислительной геометрии. Задачи о прямоугольниках.»

Цель занятия: Рассмотреть алгоритмы решения задач о прямоугольниках.

1. Дано N $(0 \le N < 5000)$ прямоугольников со сторонами, параллельными координатным осям. Каждый прямоугольник может быть частично или полностью перекрыт другими прямоугольниками. Вершины всех прямоугольников имеют целочисленные координаты в пределах [-10000, 10000] и каждый прямоугольник име-



ет положительную площадь. Периметром называется общая длина внутренних и внешних границ фигур, полученных путем наложения прямоугольников. Найти значение периметра.

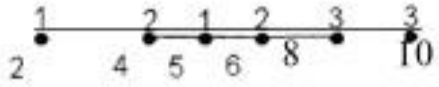
На рисунке показана фигура из 7 прямоугольников. Соответствующие границы полученной фигуры показаны на следующем рисунке.

Пример входного файла:

7 (количество прямоугольников)
-15 0 5 10 (границы прямоугольника — нижняя левая и верхняя правая)
-5 8 20 25
15 -4 24 14
0 -6 16 4
2 15 10 22
30 10 36 20
34 0 40 16

Ответ: 228.

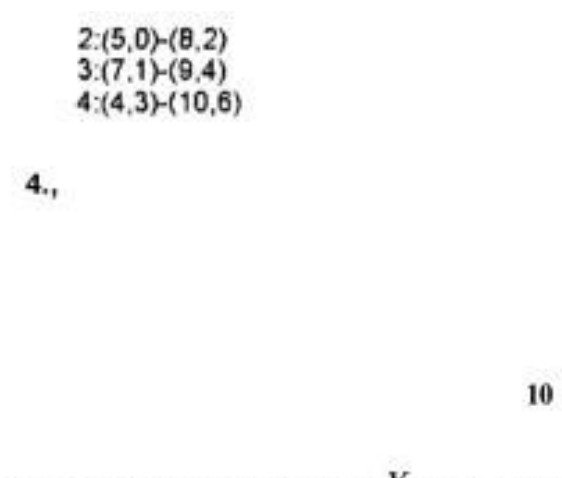
Рассмотрим более простую задачу. На прямую «набросали» какое-то количество отрезков. Последние могут пересекаться, находиться один внутри другого и т. д. Требуется определить количество связных областей, образованных этими отрезками. На рисунке даны три отрезка (2, 5), (4, 6) и (8,10). Количество связных областей две. Как их найти? Используем стандартный прием. Записываем в



один массив координаты отрезков, в другой признаки начала (1) и конца отрезков (-1). Сортируем элементы первого массива по неубыванию, одновременно переставляя соответствующие элементы второго массива. После этого осталось считать сумму значений признаков. Если ее текущее значение равно нулю, то выделена очередная связная область.

Вернемся к исходной задаче. Очевидно, что можно считать периметр отдельно, по частям. Например, первоначально его составляющую по оси *Y*, а затем по оси X. Рассмотрим идею решения на примере (рисунок). Даны четыре прямоугольника. Считаем составляющую периметра по оси *Y*. Ось разбивается координатами прямоугольников на следующие участки: (0,1), (1,2), (2,3), (3,4) и (4,6). Рассматриваем каждый участок, т. е.

определяем прямоугольники (точнее их координаты по X с признаками начала и конца интервала), которые могут дать «вклад» в периметр. Для первого участка — один прямоугольник, для второго — три, для третьего — два, для четвертого — три и для пятого — один. Выписав для каждого участка координаты X с соответствующими признаками, выполняем сортировку и находим количество связных областей. Так, на рисунке для первого сечения (отмечено цифрой 1) количество связных областей равно 1, а для второго сечения — 2. После этого достаточно длину этого участка умножить на количество связных областей и удвоить результат. Получаем составляющую периметра по оси У на этом участке.



После этих замечаний можно приступить к разбору решения. Как обычно, начинаем со структур данных.

```
Const MaxN=5001;
Type   MasPn=Array[1..MaxN*2] Of Integer;
     MasSw=Array[1..MaxN*2, 1..2]  Of  Integer;{*Для
        записи координат и соответствующих
        признаков начала и конца отрезков.*}
Var   N,  i:  Integer;
     Ax,  Ay:  MasPn;{*Для хранения координат
       прямоугольников.*}
```

Основная программа.

```
Begin
 Assign (Input, '...') /Reset (Input) /Read (N) /
 For i:=l To N Do Begin Read (Ax[i*2-l],
   Ay[i*2-1]) /Read(Ax[i*2], Ay[i*2]); End/
 Close (Input)/
 Assign (Output,  '...') /Rewrite (Output) /
 WriteLn(GetPerim(Ax,Ay)  + GetPerim(Ay, Ax));
 Close(Output);
End.
```

Вспомогательную процедуру сортировки по любому методу с временной оценкой $O(N*logN)$ описывать не будем.

```
Procedure Sort(Var nn: MasSw; lf, rg: Integer) ;
 Begin

 End;
```

Функция ***GetPerim.***

```
Function GetPerim (Const Ax, Ay: MasPn): LongInt;
 Var  i: Integer;
sc: LongInt;
D: MasSw;
 Begin
  sc:=0;{*Значение периметра по одному из
      направлений.*}
  FillChar(D, SizeOf(D) , 0) ;
  For i:=1 To 2*N Do D[i, 1] :=Ax[i] ; { *Массив
      координат по этому направлению. *}
  Sort(D, 1, 2*N) ;{*Сортировка.*}
  For i:=1 To N*2-1 Do
   If D[i,1]<>D[i+1, 1] Then{*Если  координаты не
      равны, то вычисляем количество связных
      областей на этом сечении - функция GetPr,
      умножаем на длину участка и на 2.*}
    sc:=sc+GetPr(Ax, Ay, D[i, 1]) * (D[i+1, 1] -
      D[i, 1]) * 2;
  GetPerim:=sc;
```

```
End;
```

Следующий шаг уточнения. Вычисляем количество связных областей на сечении, определяемом значением переменной *x*.

```
Function GetPr(Const Ax, Ay: MasPn; x: Integer):
    Integer;
 Var  i, b, d, sc: Integer;
      nn: ^MasSw;{*Число прямоугольников многовато
           (до 5000), задействуем «кучу». *}
 Begin
  New(nn);
  GetRect(Ax, Ay,  x,  nn^, sc);{*Определяем
          прямоугольники,имеющие отношение к
          рассматриваемому сечению, их количество -
          значение переменной sc, а координаты в
          массиве nn^.*}
  If sc<>0 Then Sort(nn^, 1, sc);{*Выполняем
          сортировку с одновременной перестановкой
          признаков начала и конца отрезков.*}
  b:=0;{*Для хранения суммы значений признаков. *}
  If sc>0 Then d:=1 Else d:=0;
  For i:=1 To sc-1 Do Begin
  b:=b+nn^[i, 2];
  If (b=0) And (nn^[i+1, 1]<>nn^[i,1]) Then Inc(d);
           {*Если текущее значение b равно нулю и
           координаты не совпадают, то увеличиваем
           счетчик числа связных областей. *}
  End;
  GetPr:=d;
  Dispose(nn);
 End;
```

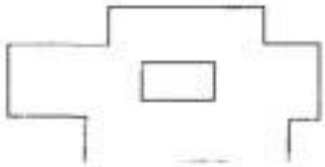
Последний шаг уточнения логики — определение характеристик сечения.

```
Procedure GetRect (Ax, Ay: MasPn; x: Integer;
     Var nn: MasSw;Var sc: Integer);
 Var i:  Integer;
 Begin
  sc:=0;
```

```
For i:=1 To N Do
 If (Ax[i*2-1]<=x) And (x<Ax[i*2]) Then Begin
                {*Если прямоугольник имеет отношение
                 к сечению, то запоминаем координаты
                 по соответствующему направлению (они
                 в массиве Ay) и признаки начала
                 и конца отрезка. *}
  Inc (sc) ;
  nn[sc,    1]:=Ay[i*2-1];nn[sc+1, 1]:=Ay[i*2];
  nn[sc,  2]:=1;nn[sc+1,  2]:=-l;
  Inc (sc);
 End;
End;
```

2. На плоскости задано N $(1 \le N \le 300)$ прямоугольников со сторонами, параллельными координатным осям (пример на рисунке). Координаты вершин прямоугольника *(x,y)* — вещественные числа с точностью до двух знаков после запятой. Написать программу поиска площади фигуры, получающейся в результате объединения прямоугольников.

Продлим все вертикальные линии до пересечения с осью *X*. Эти линии определяют на оси *X* множество интер-



валов, внутри каждого из которых длина сечения по оси У будет постоянной (рисунок). Поэтому для нахождения площади достаточно отсортировать точки на оси *X*, рассмотреть все сечения и добавить к площади <длину интервала> * <длину сечения>. Для поиска длины сечения используем метод из предыдущей задачи. Началу каждого отрезка присвоим значение признака, равное 1, а его концу — −1 и отсортируем точки по значению координаты *Y*. Считаем сумму значения признаков. Если она не равна нулю, то соответствующий интервал «плюсуем» к длине сечения.

Из структур данных приведем только те, которые требует уровень детализации объяснения.

```
ConstMaxN=300;
Type   TPoint=Record X,Y:Real;End;
Var    PrM:  Array[1..2,1..MaxNOf TPoint;
       N:  LongInt;
       Res:  Real;
       Ox, Oy: Array[l. .MaxN*2] Of Real;
```

В процедуре инициализации и ввода данных координаты вершин прямоугольника записываются в массиве *PrM*, причем в *PrM*[1] (это массив) — координаты первой вершины, а в *PrM*[2] — второй. Вершина прямоугольника с меньшими значениями (*X*,*Y*) записывается в *PrM*[1]. Кроме того, значения *X* запоминаются в массиве *Ox*.

Пример.
4 (количество прямоугольников)
0 0 10.1 5.2 (координаты нижнего левого и правого верхнего углов)
-3 3 5.36 7
1 6 9 15
8 3 20 8
Ответ.
195.188

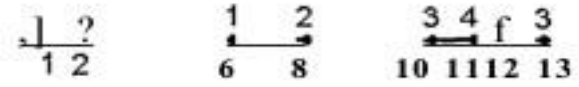
Основная процедура вычисления площади объединения прямоугольников выглядит следующим образом.

```
Procedure   Solve;
 Var   i:  LongInt;
       m:  Real;
 Begin
  Sort(Ox,  1,  N*2);{*Сортируем по неубыванию
                значения координаты X прямоугольников.
                Процедура, в силу её очевидности, не
                приводится.*}
```

```
m:=0;Res:=0;{*m - длина сечения, Res - значение
          площади объединения прямоугольников.*}
 For i:=1 To N*2 Do Begin
  If i<>1 Then Res:=Res + Abs ((Ox[i]-Ox[i-1])*m) ;
        {*Прибавляем площадь очередного сечения.*}
  If (i=l) Or Not(Eq(Ox[i],Ox[i-1])) Then Calc
   (Ox[i], m) ; {*Определяем новое значение длины
         сечения.*}
 End;
End;
```

Уточнению подлежит процедура *Calc*.

```
Procedure Calc(Const x: Real; Var rs: Real);
 Var i, M: LongInt;{*M - количество интервалов
     на данном сечении. *}
 Begin
  <Инициализацияпеременных процедуры, в частности
M:=0>:
  For i:=l To N Do {*Формируем массив ординат для
                данной координаты x. *}
   <Если есть пересечение прямоугольника с прямой,
параллельной оси Y и проходящей через точку x, то
занести координаты Y прямоугольника в рабочий массив,
не забыв при этом сформировать признаки начала и
конца отрезка. Количество пересечений - значение
переменной M.>;
  If M=0 Then rs:=0
   Else Begin
   <Сортируем
```




```
границы интервалов по
оси Y, переставляя
одновременно соответствующие элементы массива
признаков>;
   <Вычисляем новую длину сечения - значение
переменной rs>;{*Так, для примера на рисунке длина-
              сечения равна 10.*}
  End;
 End;
```

Дальнейшее уточнение логики вынесем на самостоятельную работу.

3. Найти площадь пересечения N прямоугольников со сторонами, параллельными осям координат.

Входные данные. Число N (количество прямоугольников $2 \le N \le 1000$) и N четверок действительных чисел X_{i1}, Y_{i1}, X_{i2}, Y_{i2}, задающих координаты левого нижнего и правого верхнего углов прямоугольника.

лов прямоугольника.

Выходные данные. Площадь пересечения (общей части) всех данных прямоугольников, либо выдать, что они не пересекаются.

Пример.
3
0 0 10 10
5 5 15 15
-1.5 0 7 8
Ответ: 6

Задача предельно проста. Приведем ее с целью «закрыть» тему «Прямоугольники на плоскости со сторонами, параллельными осям координат, их периметр и площади». Оказывается, что можно независимо найти область пересечения интервалов на прямой отдельно по осям *X* и *Y*. Эти интервалы определяют прямоугольник, являющийся пересечением заданных прямоугольников.

```
Const MaxN=1000;
      Eps=1e-7;
Type  Segm=Record L, R: Real; End;
      SegmArray=Array[1..MaxN] Of Segm;
Var   N: Integer;
      OnX, OnY: SegmArray;
      ObX, ObY: Segm;
Procedure  Init;
 Begin
  <ввод данных>
 End;
Function  Cross (A,B:Segm;Var New:Segm) -.Boolean;
      (*Устанавливаем факт пересечения двух
        отрезков A и B.*)
```

```
Begin
 If (B.L-A.L)<Eps Then New.L:=A.L Else New.L:=B.L;
 If (A.R-B.R)<Eps Then New.R:=A.R Else New.R:=B.R;
 Cross:=(New.L-New.R)<Eps;
End;
Function SolSegm(Const On: SegmArray;Var Ob:
          Segm): Boolean;{Определяем область
          пересечения интервалов на прямой, если
          она есть.*}
 Var    i ; Integer;
 Begin
  Ob:=On[1];
  i:=2;
  While (i<=N) And Cross(On[i],Ob, Ob) Do Inc(i);
  SolSegm:=i>N;
 End;

Begin
 Init{*Стандартный ввод данных. *}
 Write ('Ответ > ');
  If SolSegm(OnX, ObX) And SolSegm(OnY, ObY)
   Then WriteLn('Площадь',Abs ( (ObX.L-ObX.R) *
                              (ObY.L-ObY.R))    :0:6)
   Else WriteLn(*Пересечения нет. *);
End.
```

Практическое занятие №22 по теме: «Анализ алгоритма решения задачи»

Цель занятия: Научиться делать анализировать сложность алгоритма в зависимости от используемых арифметических действий и их количества.

Основными критериями при выборе представления структур и алгоритмов являются адекватность структуры конкретной задаче, объем памяти, требующийся для её представления, и временная эффективность.

При этом приходится учитывать множество факторов – внутреннюю архитектуру компьютера, объем памяти для размещения данных и методы управления ею, сложность алгоритмов. Как правило, это необходимо делать на ранних стадиях проектирования программы при выборе структуры и алгоритма.

В связи с этим очень важную роль играет анализ вычислительной сложности алгоритма.

Для структур данных применяется методика оценки эффективности алгоритмов в терминах размера структуры, т. е. количества элементов в ней. Преимущество этого метода

заключается в том, что здесь нет необходимости учитывать архитектурные особенности конкретной вычислительной машины. Метод дает абстрактные характеристики эффективности алгоритмов.

Критерии эффективности

В конечном счете алгоритм выполняется на вычислительной системе со специфическим набором команд и периферийными устройствами. Алгоритм может быть разработан для отдельной системы для полного использования её преимуществ. Критерий, называемый *системной эффективностью* , сравнивает скорость выполнения двух или более алгоритмов задачи, выполняя алгоритмы задачи с одними и теми же наборами данных. Можно определить относительное время, используя системные часы. Оценка этого времени становится мерой системной эффективности для каждого из алгоритмов.

При работе с некоторыми алгоритмами могут стать проблемой ограничения памяти. Может потребоваться большой объем памяти для временного хранения, ограничивающий размер набора данных, или дисковая подкачка данных.

Эффективность пространства – это мера относительного количества внутренней памяти, используемой алгоритмом. Эта мера может указать тип компьютера, который может выполнять алгоритм, и полную системную эффективность. Вследствие увеличения памяти в новых системах, анализ пространственной эффективности становится менее важным.

Третий критерий рассматривает внутреннюю структуру алгоритма, анализируя его работу, количество операций в алгоритме. Эти измерения не зависят от вычислительной системы. Критерийизмеряетвычислительнуюсложностьалгоритмаотносительнon-количества элементов в структуре. Этот критерий называется *вычислительной эффективностью* и выражается через нотацию O(n) - Big-O.

Рассмотрим определение *нотации Big-O* для алгоритма сортировки вставками

Времясортировкивставкамизависитотразмерасортируемогомассиваиотисходной упорядоченности массива. Размер входных данных алгоритма может задаваться либо числом элементов в структуре или массиве, числом битов , иногда несколькими числами (числом вершин и числом ребер в графе).

Время работы алгоритма определяется числом шагов, которые он выполняет. Будем считать, что одна строка псевдокода, требует не более, чем фиксированного числа элементарных операций, если строка не является формулировкой каких-либо сложных действий.

При анализе алгоритма сортировки вставками - Insertion-Sort около каждой строки будем отличать ее стоимость – число операций и число раз, которое эта строка выполняется.

	Insertion-Sort (A[1..n])	Стоимость	Число раз
1	for j <- 2 to n	C_1	n
2	do Key <- A[j]	C_2	n-1
3	▷ добавить A[j] в отсортированной части A [1..j-1]		
4	i <- j - 1	C_4	n –1
5	while i > 0 and A[i] > Key	C_5	$\sum_{j=2}^{n} t_j$

6	do A[i+1]<- A[i]	C_6	$\sum_{j=2}^{n}(t_i-1)$
7	i <- i -1	C_7	$\sum_{j=2}^{n}(t_i-1)$
8	A[i+1] <- Key	C_8	n -1

Для каждого j от 2 до n обозначим, сколько раз будет исполнена строка 5 - t_j. .Строка стоимостью C, повторенная в алгоритме m раз, вносит вклад в общее время выполнения алгоритма-с $\times$ mопераций.

Сложив вклады всех строк, получим:

$$T(n)=C_1\times n+C_2\times(n-1)+C_4\times(n-1)+C_5\times\sum_{i=2}^{n}t_j+C_6\times\sum_{i=2}^{n}(t_j-1)+C_7\times\sum_{i=2}^{n}(t_j-1)+C_8\times(n-1)$$

Стоимость зависит не только от объема данных - n, но и от их упорядоченности. Наиболее благоприятный случай, когда массив уже отсортирован по возрастанию. Тогда цикл в строке 5 завершается после первой же проверки, т. к. A[i] < Key при i = j -1 и t_j = 1. Тогда общее время есть:

$$T(n)=C_1\times n+C_2\times(n-1)+C_4\times(n-1)+C_5\times(n-1)+C_8\times(n-1)=$$
$$=(C_1+C_2+C_4+C_5+C_8)\times n-(C_2+C_4+C_5+C_8)$$

Таким образом, T(n) – линейная функция имеет вид: T(n)=a $\times$ n+b

Если же массив расположен в обратном убывающем порядке, время работы функции будет максимальным: каждый элемент A[j] придется сравнивать со всеми элементами A[1],...,A[j-1].

При этом t_j = j. и $\sum_{j=2}^{n}t_j=\sum_{j=2}^{n}j$, а $\sum_{j=2}^{n}t_j-1=\sum_{j=2}^{n}j-1$.

Поскольку $\sum_{j=2}^{n}j=\frac{n*(n+1)}{2}-1$ и $\sum_{j=2}^{n}(j-1)=\frac{n*(n+1)}{2}$, то в худшем случае получим:

$$T(n)=C_1\times n+C_2\times(n-1)+C_4\times(n-1)+C_5\times\left(\frac{n\times(n-1)}{2}-1\right)+C_6\times\left(\frac{n\times(n-1)}{2}\right)+C_7\times\left(\frac{n\times(n-1)}{2}\right)+C_8\times(n-1)=$$
$$=\left(\frac{C_5}{2}+\frac{C_6}{2}+\frac{C_7}{2}\right)\times n^2+\left(C_1+C_2+C_4+\frac{C_5}{2}-\frac{C_6}{2}-\frac{C_7}{2}+C_8\right)\times n^2-(C_2+C_4+C_5+C_8)$$

Теперь функция T(n) – квадратичная и имеет вид: $T(n)=a\times n^2+b\times n+c$

Константы a, b, c определяются значениями C_1,...,C_8.

Время работы в худшем случае

Итак, время работы в худшем и лучшем случаях могут сильно отличаться. Большей частью нас интересует время работы в худшем случае по нескольким причинам:

1. Зная время работы в худшем случае, можно гарантировать, что выполнениеалгоритма закончится за некоторое время, даже не зная, какой данные указанного размера n будут обрабатываться.
2. Напрактике“плохиевходныеданные”могутпопадатьсячасто(например,поиск отсутствующего элемента в баземданных).

3. Время работы в среднем может быть близким к времени в худшем случае. Для функций Insertion_Sort для цикла 5 – 8 в среднем половина элементов массива A[1..j-1] больше A[j] и $t_j = j/2$ в среднем. Тогда T(n) квадратично зависит отn.

В некоторых случаях интересно также среднее время работы алгоритма. Эта величина зависит от распределения вероятностей и распределения входных данных и может отличаться от предполагаемого, равномерного распределения . Иногда можно достигнуть равномерности, используя датчик случайных чисел.

Порядок роста

Вывод T(n) основан на нескольких упрощающих предположениях. Сначала предположим, что время выполнения j - й строки постоянно и равно C_i. Затем огрубим оценку до an^2+bn+c. Если пойти дальше и в худшем случае определить порядок роста n^2, отбрасывая члены меньшего порядка и коэффициент при n^2, то получим $T(n)= O(n^2)$.

Алгоритм с меньшим порядком роста предпочтительнее. Если один алгоритм имеет порядок $O(n^3)$, а другой - $O(n^2)$, то можно сделать вывод, что второй алгоритм эффективнее.

Сравнивая алгоритмы путем анализа T(n), не обязательно искать точное количество выполняемых действий. Достаточно оценить асимптотику роста времени алгоритма при стремлении n к бесконечности. Если у одного алгоритма она меньше, чем у другого, то в большинстве случаев он будет эффективнее. Порядок роста определяет зависимость времени выполнения от размера структуры n.

Традиционно значение аппроксимирующей функции для порядка роста выбирается среди небольшого набора полиномиальных, логарифмических и экспоненциальных функций. Для классических структур эти функции дают наилучшие оценки вычислительной сложности алгоритма.

При выполнении аппроксимации T(n) используется
понятие доминирующего терма. Например, для функции T(n)=n+2 терм n доминирующий и функция T(n) огрубляется за счет отбрасывания не доминирующих термов.

Например, для функции T(n)=n+2 терм n - доминирующий и функция T(n) огрубляется за счет отбрасывания не доминирующих термов:

$$T(n)= \; n+2 \le (n+n=2\times n) \;\; \textbf{для} \;\; n \ge 2,$$

Следовательно,

$$T(n) = O(n) = 2(n).$$

Для функции $T(n) = n^2 + n +1$:

$$T(n)= \; n^2+n+1 \le (n^2+n^2+n^2=3\times n^2) \;\; \textbf{для} \;\; n \ge 1.$$

Классификация алгоритмов по эффективности

1. Если алгоритм имеет порядок O(1), то его вычислительная сложность не зависит от количестваэлементоввструктуре.Примеры:поискминимальногоилимаксимального значения в упорядоченном массиве, присоединение элемента к концу или началу очереди, извлечение элемента а изстека;
2. Алгоритм с порядком роста O(n) называется *линейным*. Например, присоединение элемента к концу очереди , если заголовок очереди не хранит указатель на конец списка, нахождение минимального или максимального элемента вмассиве.
3. Алгоритм порядка $O(\log_2 n)$ называется *логарифмическим*. К таким относятся алгоритмыобработкибинарногодерева,алгоритмбинарногопоискавупорядоченном массиве.
4. Алгоритм порядка $O(n^2)$ называется *квадратическим* (алгоритмы сортировки вставками, обменнойсортировки).
5. Алгоритм порядка $O(n^3)$ называется *кубическим*. Примером может служить алгоритм Уоршалла на графе, алгоритм умноженияматриц.
6. Квадратическиеикубическиеалгоритмынерекомендуетсяиспользоватьпри больших n, т. к. их вычислительная сложность резковозрастает.
7. Алгоритмы порядка $O(2^n)$ называются *экспоненциальными*. Эти алгоритмы настолько медленные, что их можно использовать только при малых n. Обычно такиеалгоритмы называют NP - сложнымиалгоритмами.

Итак, при выборе структуры представления данных и алгоритмов обработки необходимо на ранней стадии проектирования программы оценить вычислительную сложность с помощью нотации Big – O. Это позволяет либо подтвердить обоснованность выбора структуры и алгоритма либо отказаться от принятого решения без тяжёлых последствий. Если оценка показывает неприемлемость сделанного выбора, нужно подобрать или иной алгоритм или иную структуру данных, дающих более эффективное решение.

Практическое занятие №23 по теме: «Анализ рекурсивного алгоритма»
Существует ряд важных практических причин, побуждающих нас заниматься анализом алгоритмов. Одной из них является потребность получения оценок или границ для объема памяти и времени работы, необходимых алгоритму для успешной обработки входных данных. Следует избегать разработки программы, которая за отведенное студенту машинное время не успеет обработать входные данные достаточно большого размера (например, следует признать неудовлетворительной программу обработки графов, если из-за нехватки времени она не дает ответа для графа, состоящего из десяти вершин). Лучше заранее (еще при разработке алгоритма) с помощью карандаша и бумаги оценить объем памяти и время, необходимые разрабатываемому алгоритму, а затем улучшать его (или разработать новый, более эффективный). Хороший анализ может выявить узкие места в ваших программах (например, те части программы, на выполнение которых расходуется большая часть времени), а также выбрать более подходящий алгоритм из широкого класса алгоритмов, решающих одно и то же задание.

Для алгоритма A мы определили функцию *временной сложности* $ВРЕМЯ\ A(n)$, дающуюверхнююграницудлямаксимальноговремениегоработы, т.е.максимальногочисла основных операций (сложения, сравнения и т.д.), которые долженвыполнитьалгоритм A при решении задачи на входных данных размером n. Аналогично можно определить функцию *емкостной сложности* $ПАМЯТЬ\ A(n)$, дающую границу для максимального числа одновременно существующих скалярных значений при выполнении A на входных данных размером n.

Таблица1

Границы размеров входных данных, определяемые скоростью роста временной сложности алгоритма

Алгоритм	Временная сложность	Максимальный размер входных данных, для которых за указанное время алгоритм может быть выполнен		
		1с	1мин	1ч
A_1	n	1000	6×10^4	$3^6 \times 10^6$
A_2	n log n	140	4893	2×10^5
A_3	n^2	31	244	1897
A_4	n^3	10	39	153
A_5	2^n	9	15	21

Таблица2

Эффект десятикратного сокращения времени выполнения на ЭВМ одной основной операции

Алгоритм	Временная сложность	Максимальный размер входных данных, который алгоритм может обработать за заданное время	
		до ускорения	после ускорения
A_1	n	S_1	$10S_1$
A_2	n log n	S_2	Примерно $10S_2$ для больших S_2
A_3	n^2	S_3	$3{,}16S_3$
A_4	N^3	S_4	$2{,}15S_4$
A_5	2^n	S_5	$S_5 + 3{,}3$

Хорошим критерием качества алгоритма является скорость роста его сложности в зависимости от увеличения размера входа. Рассмотрим пример, показывающий влияние порядка роста сложности алгоритма на увеличение максимально допустимого размера входных данных, которого можно достичь увеличением скорости ЭВМ, а также эффект примененияболееэффективногоалгоритма(см.табл.1и2).Изтабл.1видно,что A_1 может обработать за 1 с входные данные размером 1000, в то время как A_5 -- входные данные не длиннее9.Табл.2показывает,чтодляалгоритма A_5 десятикратноеувеличениескорости

ЭВМ позволяет увеличить размер входных данных, к которым его можно применять, только на три, тогда как для алгоритма A_3 размер входных данных можно по крайней мереутроить.

Следует, однако, иметь в виду, что порядок роста сложности алгоритма является определяющим только при обработке данных большого размера. Для задач с малым размером входных данных (а именно на таких данных и проверяется правильность студенческих алгоритмов) следует учитывать мультипликативные константы при сравнении алгоритмов. Например,предположим,чтовременныесложностиалгоритмов A_1, A_2, A_3, A_4 и A_5 равнысоответственно $1000n$, $100n \log n$, $10n^2, n^3$, 2^n. Тогда A_5 будет наилучшим для данныхразмером $2 \le n \le 9, A_3$ -- дляданныхразмером $10 \le n \le 58, A_2$ -- при $59 \le n \le 1024$, а A_1 -- толькопри $n > 1024$.

В качестве примера приведем анализ временной сложности двух алгоритмов генерации перестановок .

Общую эффективность алгоритма генерации перестановок в лексикографическом порядкеопределяютдвеоперации--числотранспозицийичислосравненийпарэлементовв перестановке. Обозначим через I_k и C_k соответственно число транспозиций и число сравнений, выполняемых алгоритмом для порожденияпервых $k!$ из $n!$ перестановок. Поскольку для порождения каждой из n подпоследовательностей перестановок с одним и тем же значением π_1 требуется I_{n-1} транспозиций, а для каждого из $n-1$ переходов от одной подпоследовательности к другой требуется $[(n+1)/2]$ транспозиций, получаем следующее рекуррентноесоотношение:

$$I_n = \begin{cases} nI_{n-1} + (n-1)[\frac{n+1}{2}], & \text{если } n > 0, \\ 0, & \text{если } n = 0. \end{cases}$$

для решения этого рекуррентного соотношения сделаем замену переменных. Взяв $S_n = I_n \frac{n+1}{2}$, получим новое рекуррентноесоотношение:

$$S_n = \begin{cases} 1, & \text{если } n = 1, \\ nS_{n-1}, & \text{если } n > 1 \text{ и } n \text{ нечетно}, \\ n(S_{n-1}+1), & \text{если } n > 1 \text{ и } n \text{ четно}. \end{cases}$$

Решение этого соотношения легко получить:

$$S_n = n! \left(1 + \frac{1}{2!} + \frac{1}{4!} + \frac{1}{6!} + \ldots + \frac{1}{(2[(n-1)/2])!}\right).$$

Поэтому

$$S_n = n! \left(1 + \frac{1}{2!} + \frac{1}{4!} + \frac{1}{6!} + \ldots + \frac{1}{(2[(n-1)/2])!}\right).$$

Аналогично можно показать, что

$$I_n = n! \left(\Sigma_{k=0}^{[(n-1)/2]} \frac{1}{(2k)!}\right) - \left[\frac{n+1}{2}\right] \approx 1,5n!.$$

и получить

$$C_n = \begin{cases} nC_{n-1} + \frac{(n-1)(3n-2)}{2}, & \text{если } n > 1, \\ 0, & \text{если } n = 1 \end{cases}.$$

Что же касается алгоритма порождения перестановок в порядке минимального изменения, то он выполняет $n!$ транспозиций и $n!$ сравнений (более точно, алгоритмделает 1!+2!+3!+...+ n! сравнений). Таким образом, алгоритм порождения перестановок в порядке минимального изменения является болееэффективным.

В общем случае следует иметь в виду, что повышение скорости работы алгоритма может потребовать увеличения необходимой ему памяти. Например, рекурсивный алгоритм порождения перестановок в порядке минимального изменения совсем не производит сравнений, но имеет емкостную сложность , в то время как емкостная сложность нерекурсивного алгоритма равна . Этот же пример показывает справедливость обратного утверждения: за счет увеличения временной сложности алгоритма можно понизить его емкостную сложность.

Как правило, более эффективный алгоритм приводит к программе большего размера и требует больших усилий как на его разработку, так и на его обоснование.

3.2.1 Основные источники

1. Шень А.Х. Практикум по методам построения алгоритмов [Электронный ресурс] / А.Х. Шень. — Электрон. текстовые данные. — М. : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 335 c. — 2227-8397. — Режим доступа:http://www.iprbookshop.ru/52164.html
2. Лубашева Т.В. Основы алгоритмизации и программирования [Электронный ресурс] : учебное пособие / Т.В. Лубашева, Б.А. Железко. — Электрон. текстовые данные. — Минск: Республиканский институт профессионального образования (РИПО), 2016. — 379 c. — 978-985-503-625-9. — Режим доступа:http://www.iprbookshop.ru/67689.html

4. Контрольно-оценочные средства для итоговой аттестации по учебной дисциплине

Зачетная работа по дисциплине «Теория алгоритмов»

Приведены три варианта зачетной работы по дисциплине «Теория алгоритмов». Варианты расположены по уровню сложности (1 — самый легкий).

Первое задание дается с целью проверить, приобрели ли учащиеся навыки проверки правильности работы алгоритма. Для успешного решения первого задания учащиеся должны проверить правильность работы алгоритма на необходимомколичестве тестов. Можно использовать и другие алгоритмы, в описаниях которых умышленно допущены ошибки.

Второе и третье задания заключаются в составлении машины Поста и машины Тьюринга соответственно.

Одобрено на заседании ЦК математических и общих естественнонаучных дисциплин Протокол №_____от «____»_____20__г. Председатель_Р.В. Пасюнина	Билет № 1 Теория алгоритмов учебная дисциплина Группа Семестр 5	УТВЕРЖДАЮ Заместитель директора по УР _____Т.Ю. Базилевич «____»_____20__г.

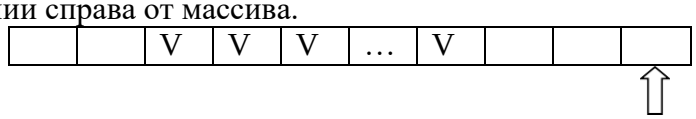
Вариант 1

Задание 1. Ниже приведен алгоритм нахождения суммы конечного ряда $s = x + x^2 + \ldots + x^n$ *для любого положительного n и произвольного* x.

Шаг	Действие
1	Вводим значения *n* и *x.* Выполняем начальные присваивания: s = 0, a = 1, k= 1 (s — искомая сумма, *a* — очередное слагаемое, k — счетчик слагаемых)
2	$a = a \cdot x$ (вычисляем очередное слагаемое)
3	$s = s + a$ (вычисляем сумму первых k элементов)
4	Увеличиваем счетчик k на единицу
5	Если k = *n,* то требуемая сумма подсчитана; конец алгоритма, иначе переходим на шаг.

Укажите, при каких *n* данный алгоритм работает неверно. Напишите правильный алгоритм для решения этой задачи.

Задание 2. Написать программу для машины Поста, которая к массиву, содержащему *n* меток, прибавляет еще одну метку. В исходном состоянии каретка стоит на некотором расстоянии справа от массива.



Задание 3. Опишите, какой алгоритм выполняет данная машина Тьюринга. Известно, что в начальном состоянии автомат обозревает самый левый символ входного слова.

	a_0	0	1
q_1	a_0 Н !	1 П q_1	0 П q_1

К каким словам, составленным из символов данного алфавита, применима машина?

Преподаватель:

Одобрено на заседании ЦК математических и общих естественнонаучных дисциплин Протокол №_____от «____»_____20__г. Председатель_Р.В. Пасюнина	Билет № 2Теория алгоритмов учебная дисциплина Группа Семестр 5	УТВЕРЖДАЮ Заместитель директора по УР _____Т.Ю. Базилевич «____»_____20__г.

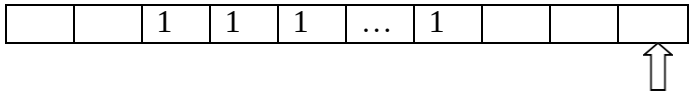
Вариант 2

Задание 1. Ниже приведен алгоритм нахождения суммы конечного ряда $s = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots + \frac{x^n}{n!}$ *для любого четного неотрицательного n (n — показатель степени) и произвольного* x.

Шаг	Действие
1	Вводим значения *n* и *x*
2	Выполняем начальные присваивания: s = 1, *a1* = 1, *a2* = 1, k= 0 *(s* — искомая сумма, *a1*— числитель очередного слагаемого, *a2* — знаменатель очередного слагаемого, k— счетчик слагаемых)
3	Если *n = k*, то требуемая сумма найдена и алгоритм заканчивает свою работу;
4	Увеличиваем счетчик на единицу
5	*al = - a1 • x*^{2} (вычисляем числитель очередного слагаемого)
6	*a2 = a2* • (k - 1) • k (вычисляем знаменатель очередного слагаемого)
7	*s = s-a1/a2* (вычисляем сумму первых k элементов)
8	Переходим на шаг 3.

Укажите, при каких *n* данный алгоритм работает неверно. Напишите правильный алгоритм для решения этой задачи.

Задание 2. Написать программу для машины Поста, которая к массиву, содержащему *n* единиц, прибавляет справа и слева по одной единице. В исходном состоянии каретка стоит на некотором расстоянии справа от массива.



Задание 3. Опишите, какой алгоритм выполняет данная машина Тьюринга.

	a_0	0	1
q_1	a_0 Н !	a_0 П q_1	a_0 П q_1

К каким словам, составленным из символов данного алфавита, применима машина? Какое условие надо наложить на начальное положение автомата для того, чтобы результат применения машины к произвольному слову из алфавита был одним и тем же?

Преподаватель:

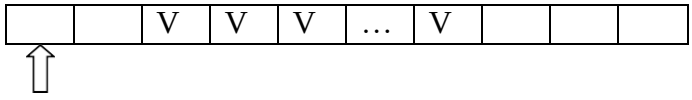
Одобрено на заседании ЦК математических и общих естественнонаучных дисциплин Протокол №_____от «____»_____20__г. Председатель_Р.В. Пасюнина	Билет № 3Теория алгоритмов учебная дисциплина Группа Семестр 5	УТВЕРЖДАЮ Заместитель директора по УР _____Т.Ю. Базилевич «____»_____20__г.

Вариант 3

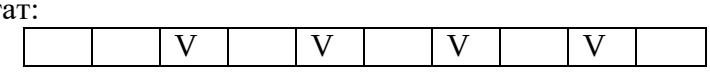
Задание 1. Дана последовательность целых чисел a_1 a_2, ..., a_n. Укажите все ошибки в предлагаемой записи алгоритма нахождения максимального и минимального элемента в заданной последовательности. Идея алгоритма такова: разбиваем последовательность на пары; числа в каждой паре упорядочиваем по возрастанию; максимальный элемент ищем среди четных (по месту расположения) элементов последовательности, а минимальный — среди нечетных.

Шаг	Действие
1	Разобьем исходную последовательность на пары. Последняя пара может быть неполной, если число n — нечетно
2	Положим $m = \left[\frac{n}{2}\right]$, если n — четно, и $m = \left[\frac{n}{2}\right] + 1$ в противном случае *(m* — число пар, которые надообработать);
3	Упорядочим числа в каждой паре по возрастанию. Будем элементы в паре обозначать $P_{k,1}$ и $P_{k,2}$, где k — номер пары
4	Положим $min = P_{1,1}$ и $max = P_{1,2}$
5	Положим $k = 1$
6	Если $P_{k,2} > max$, тогда $max = P_{k,2}$
7	Если $P_{k,1} < min$, тогда $min = P_{k,1}$
8	Увеличиваем k на единицу
9	Если $k = m$, конец алгоритма; иначе переход на п. 6.

Задание 2. Дан массив из n меток (т. е. n идущих подряд отмеченных ячеек). Каретка обозревает крайнюю левую ячейку. Составьте для машины Поста программу, расставляющую эти метки на ленте так, чтобы между отметками было по одной пустой ячейке.



Результат:



Задание 3. Опишите, какой алгоритм выполняет данная машина Тьюринга. Автомат в начальном состоянии обозревает самый правый символ входного слова.

	a_0	+	−
q_1	a_0 Н !	+ Л q_2	− Л q_1
q_2	a_0 Н !	− Л q_1	− Л q_1

К каким словам, составленным из символов данного алфавита, применима машина?

Преподаватель:

Ответы и решения заданий зачетной работы

Вариант 1

1. Данный алгоритм будет работать неверно при любом заданном я, так как начальное значение счетчика положено равным 1. Если требуется посчитать сумму из одного слагаемого, то алгоритм никогда не остановится. Основная ошибка в том, что здесь фактически реализована циклическая конструкция ДЕЛАЙ ... ДО ТЕХ ПОР ПОКА.Эта циклическая конструкция при любом *n* проработает хотя бы одинраз.
Правильный алгоритм:
1) Вводим значение *n* и x. Выполняем начальные присваивания: s = 0, a = 1, k = 0 (s— искомая сумма, *a* — очередное слагаемое, *k* — счетчикслагаемых);
2) если *k* = n, то требуемая сумма подсчитана, конец алгоритма; иначе переход на шаг3;
3) увеличиваем счетчик *k* наединицу;
4) *a* = *a* x *x* (вычисляем очередноеслагаемое);
5) *s* = *s* + *a* (вычисляем сумму первых *k*элементов);
6) переходим на шаг2.
2. Машина Поста для данной задачи может иметь следующийвид:

```
1. ← 2
2. ? 1; 3 {ищем правый край массива}
3. → 4 {вернулись на пустую ячейку справа от массива}
4. V 5 {поставили справа от массива метку}
5. !
```

3. Машина Тьюринга выполняет инверсию двоичных слов (заменяет все «1» на «0» и наоборот). Машина Тьюринга применима к любым двоичным словам. В том числе,она применима и к нулевомуслову.

Вариант 2

1. Данный алгоритм будет работать неверно при любом заданном *n,* так как счетчик увеличивается толькона единицу.Крометого,вформулевычисленияусуммыдопущена ошибка (в пункте 7 надо знак «-» заменить на«+»).
Правильный алгоритм решения данной задачи:
1) Вводим значения *n их;*
2) выполняемначальныеприсваивания:s=1,al=1,a2=1,k=0(s—искомая сумма,al— числитель очередного слагаемого, *a2* — знаменатель очередного слагаемого, k— счетчик слагаемых);
3) если *n* = k, то требуемая сумма найдена и алгоритм заканчивает своюработу;
4) увеличиваем счетчик на два, k = k+2;
5) al = - al • x^2(вычисляем числитель очередногослагаемого);
6) a2 = a2 • (k - 1) • k (вычисляем знаменатель очередногослагаемого);
7) s = *s* + a1/a2 (вычисляем сумму первых kэлементов);
8) переходим на шаг3.

2. Машина Поста *для* данной задачи может иметь следующийвид:

```
1. ← 2
2. ? 1; 3 {ищем правый край массива}
3. → 4
4. V 5 {справа от массива поставили метку}
5. ← 6
6. ? 7; 5 {ищем левый край массива}
7. V 8 {слева от массива поставили метку}
8. !
```

3. Результат применения данной машины Тьюринга к конкретному слову зависит от того, в какомначальномположениинаходитсяавтомат.Еслимы будем считать,чтовначалеработы машины автомат находится против самой левой непустой ячейки, то результат применения машины к произвольному слову будет одинаков: машина сотрет это слово и остановится. Если в начале обозревается произвольная непустая ячейка слова, то стирается правая часть слова.

Данная машина Тьюринга применима к любым словам, которые можно составить из алфавита этой машины Тьюринга, так как каждое слово имеет конечную длину. Действительно, если автомат видит 0 или 1, то он стирает символ в ячейке и сдвигается вправо. То есть автомат стирает шаг за шагом содержимое непустых ячеек, до тех пор, пока не дойдет до пустой ячейки, после чего машина остановится. Если же автомат в начале обозревает пустую ячейку, то он сразу останавливается.

Вариант 3

1. Вприведенномалгоритмеестьдвеошибки.Перваяошибказаключаетсявтом, чтопри нечетном *n* для k = *m* будет неверно выполняться п. 6. Для любого *n*количество пар, которые надо обработать, равно $\left\lceil \frac{n}{2} \right\rceil$, и в качестве начальных значений *min* и *max* надо братьзначениясоответствующихэлементовпоследней*m-й*полнойпарыдлячетных*n*или элемент a_n в противномслучае.

Втораяошибкасостоитввыборетипациклическойконструкции.Впредложенном варианте алгоритма при *n* = 1 получим зацикливание, а при *n* >1 алгоритм не будет обрабатывать последнююпару.

Правильный алгоритм:

1) Разобьем исходнуюпоследовательность на пары. Последняя пара может быть неполной, если число *n*—нечетно;

2) положим*m*= $\left\lceil \frac{n}{2} \right\rceil$ *(m* — число пар, которые надообработать);

3) упорядочим числа в каждой паре повозрастанию.

Будем элементы в паре обозначать $P_{k,1}$ и $P_{k,2}$>гд*е k*— номер пары;

4) положим *min* = a_n и *max* = a_n при нечетном n. В противном случае положим *min* = $P_{m,1}$и *max* = $P_{m,2}$;

5) положим k =1;

6) если k = m, то конецалгоритма;

7) если $P_{k,2}$>*max,* тогда *max* = $P_{k,2}$;

8) если $P_{k,1}$ < *min,* тогда *min* =$P_{k,1}$

9) увеличиваем k на единицу;

10) переходим на п.6.

2. Машина Поста для этой задачи достаточно сложна. Формировать чередующийся массив будем справа от исходного массива. Окончание данных справа и слева будем путемпроверки на две подряд идущие пустые клетки. Алгоритм заканчивает работу в том случае, когда правее крайней левой метки окажется пустаяячейка.

1. X 2 {стираем левую метку}
2. → 3 {ищем первый пробел}
3. ? 4; 2
4. → 5 {ищем второй пробел подряд}
5. ? 6; 2
6. V 7 {ставим метку через пробел}
7. ← 8 {ищем первый пробел}
8. ? 9; 7
9. ← 10 {ищем второй пробел подряд}
10. ? 11; 7
11. → 12
12. → 13 {встали на левую метку}
13. → 14
14. ? 15; 16 {символ правее пустой?}
15. !
16. ← 1

3. Машина применима к входным словам, состоящим из символов « + » и «-».Данная машина Тьюринга каждый второй символ «+» заменяет на символ«-».

Преподаватель:

ПРИЛОЖЕНИЕ

КРИТЕРИИ ОЦЕНОК ПО ДИСЦИПЛИНЕ

Оценка«5»:	- ответ полный и правильный на основании изученныхтеорий; - материализложенвопределеннойлогическойпоследовательности, литературнымязыком; - ответсамостоятельный. - работа выполнена полностью иправильно; - сделаны правильные выводы; - работа выполнена по плану с учетом техникибезопасности
Оценка «4»	- ответ полный и правильный на основании изученныхтеорий; - материализложенвопределеннойлогическойпоследовательности,приэтом допущены две-три несущественные ошибки, исправленные по требованию преподавателя; - работа выполнена правильно с учетом 2-3 несущественныхошибок, исправленных самостоятельно по требованиюпреподавателя.
Оценка «3»	- ответполный,ноприэтомдопущенасущественнаяошибка,илинеполный, несвязный. - работа выполнена правильно не менее чем на половину илидопущена существенная ошибка.
Оценка «2»	- при ответе обнаружено непонимание студентом основного содержания учебного материала или допущены существенные ошибки, которые студент не смог исправить при наводящих вопросахпреподавателя; - отсутствиеответа; - допущены две (и более) существенные ошибки в ходе работы, которые студент не может исправить даже по требованиюпреподавателя; - работа невыполнена

4. Контрольно-оценочные материалы для итоговой аттестации по учебнойдисциплине

Задания к дифференцируемому зачету по дисциплине «Теория алгоритмов»

1. Алгоритмы. Общие сведения. Основные требования калгоритмам.
2. Свойства алгоритмов. Способы представленияалгоритмов.
3. Основные алгоритмическиеструктуры.
4. МашинаПоста.
5. Машина Тьюринга. Устройство машиныТьюринга.
6. Алгоритмически неразрешимыепроблемы
7. Методы построенияалгоритмов
8. Арифметика многоразрядных целыхчисел.
9. Комбинаторные алгоритмы.
10. Перебор и методы егосокращения.
11. Сортировка.
12. Алгоритмы на графах. Поиск в графе. Поиск в глубину. Поиск вширину.
13. Алгоритмы на графах. Кратчайшиепути.
14. Динамическое программирование.
15. Алгоритмы вычислительной геометрии.
16. Сравнительные оценки алгоритмов. Классификация алгоритмов повиду функциитрудоёмкости.
17. Теория сложности вычислений и сложностные классызадач.
18. Рекурсивные алгоритмы и методы иханализа.
19. Сформулируйте определение алгоритма.
20. Перечислите виды алгоритмов.
21. Перечислите способы представления алгоритмов.
22. Сформулируйте определение линейного алгоритма.
23. Сформулируйте определение разветвляющего алгоритма.
24. Сформулируйте определения команд ветвления.
25. Сформулируйте определение блок-схемыалгоритма.
26. Перечислите основные блоки в блок-схеме алгоритма.
27. Перечислите основные свойства алгоритмов.
28. Сформулируйте определения циклическогоалгоритма.
29. Виды сортировок

Выполните тестовое задание (компьютерное тестирование в программе Moodle).

1 вариант

1. Последовательность действий, допустимых для исполнителя, - это...?

А.программа

Б. алгоритм

В. команда

Г. система команд

2. Выявление ошибок и их устранение называется...?
А. отладкойзадачи
Б. отладкой исполнителя
В. отладкой алгоритма
Г. отладкойпрограммы
3. Отдельное указание исполнителю - это...?
А.программа
Б. алгоритм
В. команда
Г. приказ
4. Программы, которые содержат команду повторения, называются ...?
А.линейными
Б. разветвляющими
В. циклическими
Г. вспомогательными
5. Форма организации действий, при которой один и тот же блок выполняется есколько раз, называется...?
А. следованием
Б. циклом
В. телом цикла
Г. командой повторения
6. Составная команда, в которой одни и те же действия (команды) повторяются есколько раз, называются...?
А. командой присваивания
Б. командой повторения
В. вспомогательной программой
Г. командой ветвления
это ...?
7. Совокупность всех команд, которые может выполнить конкретный исполнитель,-
А. система программ
Б. система алгоритмов
В. система команд
Г. система задач
8. Вспомогательная команда - это...?
А.цикл
Б. ветвление
В. процедура
Г.следование
9. Графический способ описания алгоритма - это...?
А.программа
Б. блок-схема
В. алгоритм

Г. словесно-пошаговая запись
10. Сложные условия - это такие условия, которые содержат ...?
А. логическую связкуИ
Б. логическую связку ИЛИ
В. логическую связку НЕ
Г. логические связки И, ИЛИ, НЕ
2 вариант
1. Из чего состоит машинаПоста?
А. из ленты
Б. из ленты и каретки
В. из каретки
Г. из знаков
2. Лента в машине Поста...?
А. конечна
Б. бесконечна
В. ограничена
Г. может быть двух видов
3. Как нумеруются секции на ленте машиныПоста?
А. ненумеруются
Б. относительно каретки
В. жестко пронумерованы
Г. нумеруются пожеланию
4. Что записывается в секции на ленте в машинеПоста?
А. 1 или0
Б. 0
В. 1
Г. Vили ничего
5. Как называется число, стоящее в конце команды машины Поста?
А. отсылкой
Б.пересылкой
В. индексом
Г.постфиксом
6. Сколько команд у машины Поста?
А.4
Б. 3
В. 6
Г. 5
7. Сколько команд у машины Тьюринга?
А.4
Б. 3
В. 6
Г. 5

8. Информация о том, какие секции пусты, а какие отмечены и где стоиткаретка вмашине
Поста, ...?
А. позволяет считать число с ленты
Б. образует состояние ленты
В. образует состояние машины Поста
Г. неизвестно

КРИТЕРИИ ОЦЕНОК ПО ДИСЦИПЛИНЕ

Оценка«5»:	Свыше 75% правильных ответов
Оценка «4»	60%- 75% правильных ответов
Оценка «3»	60% правильных ответов
Оценка «2»	Менее 60 %