

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ДОНЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ПРИНЯТА

ученым советом ДонГУ
протокол от __ . __ .2026 № __

УТВЕРЖДЕНА

приказом ректора ДонГУ
от __ . __ .2026 № __

ДОПОЛНИТЕЛЬНАЯ ОБЩЕОБРАЗОВАТЕЛЬНАЯ
ОБЩЕРАЗВИВАЮЩАЯ ПРОГРАММА
естественно-научной направленности
«ИНФОРМАТИКА ДЛЯ ПОСТУПАЮЩИХ В ВУЗЫ»

Программа разработана
и реализуется факультетом
математики и информационных
технологий

I. ОБЩИЕ СВЕДЕНИЯ

Программа разработана в соответствии и с учетом требований регламентирующих организацию и осуществление образовательной деятельности нормативных правовых актов, список которых приведен ниже.

1. Федеральный закон от 29.12.2012 № 273-ФЗ «Об образовании в Российской Федерации».
2. Приказ Министерства просвещения Российской Федерации от 27.07.2022 № 629 «Об утверждении Порядка организации и осуществления образовательной деятельности по дополнительным общеобразовательным программам».
3. Санитарные правила СП 2.4.3648-20 «Санитарно-эпидемиологические требования к организациям воспитания и обучения, отдыха и оздоровления детей и молодежи», утвержденные постановлением Главного государственного санитарного врача Российской Федерации от 28.09.2020 № 28 (зарегистрировано Министерством юстиции Российской Федерации 18.12.2020, регистрационный № 61573).
4. Профессиональный стандарт «Педагог дополнительного образования детей и взрослых», утвержденный приказом Министерства труда и социальной защиты Российской Федерации от 22.09.2021 № 652н.
5. Приказ Министерства просвещения Российской Федерации от 12.08.2022 № 732 «О внесении изменений в федеральный государственный образовательный стандарт среднего общего образования, утвержденный приказом Министерства образования и науки Российской Федерации от 17.08.2012 № 413».
6. Приказ Министерства просвещения Российской Федерации от 18.05.2023 № 371 «Об утверждении федеральной образовательной программы среднего общего образования».

Образовательная деятельность по программе направлена на:

- формирование и развитие творческих способностей обучающихся;
- удовлетворение индивидуальных потребностей обучающихся в интеллектуальном развитии;
- профессиональную ориентацию обучающихся;
- выявление, развитие и поддержку обучающихся, проявивших выдающиеся способности.

В целом программа имеет е с т е с т в е н н о - н а у ч н у ю (кибернетическую) направленность. Содержание программы определяется приведенным далее учебно-тематическим планом. Программа может быть реализована в одном из десяти вариантов. Характеристики каждого из них приводятся далее.

Условные обозначения:

N – номер варианта;

T – срок обучения (месяцы);

Σ – трудоемкость освоения программы обучающимся (академические часы);

- ♥ – количество академических часов контактной работы преподавателя с обучающимися (полная учебная нагрузка преподавателя);
- L – учебные занятия (академические часы);
- – количество академических часов, отводимых на консультирование обучающихся преподавателем;
- ⌘ – количество академических часов, отводимых на подготовку к итоговой аттестации и на прохождение (проведение) итоговой аттестации;
- S – самостоятельная работа обучающегося (академические часы).

N	T	Σ	♥	L	□	⌘	S
1	10	328	248	160	80	8	80
2	9	296	224	144	72	8	72
3	8	264	200	128	64	8	64
4	7	232	176	112	56	8	56
5	6	200	152	96	48	8	48
6	5	168	128	80	40	8	40
7	4	136	104	64	32	8	32
8	3	104	80	48	24	8	24
9	2	88	72	48	16	8	16
10	1	56	48	32	8	8	16

Программа адресована в первую очередь лицам, завершающим обучение по основным общеобразовательным программам среднего общего образования (учащимся 11-х классов общеобразовательных организаций). К обучению по программе допускаются лица других категорий без ограничений по возрасту, выразившие интерес к освоению содержания программы. Вступительные испытания для допуска к обучению по программе не предусматриваются.

Педагогическая деятельность по реализации программы осуществляется преподавателем – лицом, имеющим высшее образование по специальности (направлению подготовки), соответствующей (соответствующему) направленности программы, и отвечающим требованиям профессионального стандарта «Педагог дополнительного образования детей и взрослых». Для работы в качестве преподавателя может быть привлечено лицо, обучающееся по образовательной программе высшего образования по специальности (направлению подготовки), соответствующей (соответствующему) направленности программы, и успешно прошедшее промежуточную аттестацию не менее чем за два года обучения.

Программа может реализовываться в течение всего календарного года, включая каникулярное время. Реализация программы осуществляется в соответствии с расписанием занятий группы обучающихся, составляемым для создания наиболее благоприятного режима труда и отдыха обучающихся с учетом пожеланий обучающихся, родителей (законных представителей) несовершеннолетних обучающихся и возрастных особенностей обучающихся.

Текущая и промежуточная аттестация обучающихся программой не предусматриваются. Обучающимся, успешно прошедшим итоговую аттестацию, выдается с е р т и ф и к а т установленного Донецким государственным университетом образца, подтверждающий успешное освоение программы.

Ц е л ь ю реализации программы является развитие, обобщение и систематизация знаний и умений обучающихся, полученных ими при освоении основной образовательной программы среднего общего образования по и н ф о р м а т и к е.

Комплекс з а д а ч реализации программы определяется тем, что в результате ее успешного освоения обучающийся должен (на уровне, предусмотренном основной образовательной программой среднего общего образования):

- обладать образующими систему базовыми знаниями, отражающими вклад информатики в формирование современной научной картины мира;
- владеть умениями и навыками алгоритмического мышления, осознавать необходимость формального описания алгоритмов, владеть элементарными умениями и навыками формализации алгоритмов;
- иметь представление о компьютерно-математическом моделировании, осознавать необходимость установления соответствия модели и моделируемого явления (объекта, процесса), владеть элементарными умениями и навыками компьютерно-математического моделирования;
- иметь понятие о способах представления и обработки данных, владеть элементарными умениями и навыками представления и обработки данных;
- иметь представление о базах данных и системах управления базами данных (СУБД), владеть элементарными умениями и навыками работы с системой управления базой данных;
- иметь общее представление об устройстве и принципах действия вычислительных машин (компьютеров), о тенденциях развития компьютерной техники и компьютерных технологий;
- иметь представление о компьютерных сетях и о той роли, которую они играют в прогрессе человечества, владеть элементарными умениями и навыками работы с компьютерной сетью;
- иметь представление о компьютерных операционных системах, об их основных функциях, об их доступности (или недоступности, в особенности – открыто, свободно, бесплатно) и многообразии, владеть умениями и навыками пользования компьютерной операционной системой;
- иметь общее представление о разработке и функционировании глобально-сетевых компьютерных систем и приложений, владеть умениями и навыками пользования глобальной компьютерной сетью.

Л и ч н о с т н ы е р е з у л ь т а т ы освоения программы обучающимися должны отражать готовность и способность обучающихся руководствоваться сформированной внутренней позицией, системой ценностных ориентаций, позитивных внутренних убеждений, соответствующих традиционным социальным ценностям, расширение жизненного опыта и опыта общественно-значимой и личностно-значимой деятельности. Результаты освоения программы обучающимися должны

быть связаны в том числе с их трудовым воспитанием (развитие интереса к различным сферам профессиональной деятельности), а также с принятием обучающимися ценности научного познания (сформированность мировоззрения, соответствующего современному уровню развития науки и общественной практики).

Нормативно программа реализуется в очной форме. Допускается реализация программы с применением элементов электронного обучения и дистанционных образовательных технологий, предполагающих предоставление обучающимся возможности удаленно-синхронного участия в занятиях. Ответственность за создание соответствующих условий в техническом и организационном плане со стороны Донецкого государственного университета возлагается на преподавателя.

II. УЧЕБНО-ТЕМАТИЧЕСКИЙ ПЛАН

Занятия с группой обучающихся проводятся один или несколько раз в неделю (наиболее приемлемый вариант: все занятия – по субботам). Распределение бюджета времени освоения программы по ее содержательным элементам выполняется преподавателем и может динамически варьироваться с целью наиболее полного удовлетворения образовательных потребностей и интересов обучающихся.

Содержание обучения по программе разработано в соответствии с основными предметными ориентирами курса элементарной информатики, объединенными в четыре тематических раздела: «Системы счисления и форматы данных», «Теоретические основы кибернетики», «Алгоритмы и программирование» и «Информационные технологии».

Т е м а т и ч е с к и й р а з д е л 1 С и с т е м ы с ч и с л е н и я и ф о р м а т ы д а н н ы х

- 1.1. Проблема записи чисел. Необходимость наличия способа записи сколь угодно больших и сколь угодно малых чисел. Понятие о системе счисления. История систем счисления. Позиционные системы счисления. Наиболее часто применяемые позиционные системы счисления: десятичная, двоичная, восьмеричная, шестнадцатеричная.
- 1.2. Информация и данные. Хорошо структурированные и слабо структурированные данные. Понятие о «больших» данных. Понятие о языке как о способе представления информации. Понятие о формате как о языке представления хорошо структурированных данных. Элементарные примеры форматов данных. Обзор популярных языков, традиционно относимых к форматам данных: языки систем счисления, языки разметки текста, язык разметки гипертекста, расширяемый язык разметки, языки описания страниц, другие распространенные форматы. Человеко-ориентированные и машинно-ориентированные форматы. Понятие открытости и закрытости формата данных. Ограниченная применимость закрытых и перспективность широкого применения открытых форматов.

Тематический раздел 2 Теоретические основы кибернетики

- 2.1. Вычислительная машина (компьютер, кибер) как исполнитель, преобразующий по заложенной в нем программе (функциональной схеме) поступающие к нему входные данные и (возможно) выполняющий предписанные его конструкцией и заложенной в нем программой действия, обусловленные содержанием выходных данных, полученных в результате преобразования входных данных. Знакомство с абстрактными вычислительными машинами (машины Поста, машины Тьюринга, нормальные (марковские) алгорифмы, нейросети Калбертсона). Общая идея устройства реальной вычислительной машины (оперативная память; носитель с программой; оперирующий (процессорный) автомат – арифметико-логическое устройство, снабженное (быть может) собственными запоминающими устройствами быстрого действия (регистрами); устройства ввода-вывода, другие периферийные устройства). История вычислительной техники. Понятие об устройстве современных вычислительных машин.
- 2.2. Логика как наука и как искусство построения непротиворечивых формально-истинностных систем, обладающих (быть может, по крайней мере, в простейших случаях) свойствами разрешимости и полноты. Логика высказываний (пропозициональная логика) как простейший раздел логики, являющийся, однако, методологическим базисом всех других логик (разделов логики). Понятие о высказывании (пропозиции, суждении, предложении, утверждении). Синтетические и аналитические высказывания. Аналитические высказывания как предмет логики высказываний. Значение истинности аналитического высказывания как функция значений истинности входящих в его состав более простых высказываний (понятие о логических (булевых) функциях). Понятие о полных системах булевых функций, примеры полных систем. Арифметические и физические модели булевых функций (логические элементы вычислительных машин). Принципиальная возможность трактовки любого исполнения как вычисления. Сводимость любых (строже говоря, наверняка любых дискретных) вычислений к вычислениям, производимым логическими автоматами. Принципиальная возможность дискретного (цифрового) моделирования мира с любой наперед заданной точностью, даже если он континуален. Понятие о квантовой парадигме (имеющееся у современных физиков сомнение относительно континуальности мира). Практическая дискретность мира.
- 2.3. Упрощенная модель действующей вычислительной машины фон-неймановского типа. Другие упрощенные модели действующих вычислительных машин (например, машины MIX и MMIX Дональда Эрвина Кнута). Команда (инструкция) как минимальный осмысленный элемент программы (функциональной схемы) для вычислительной машины. Формат машинных команд. Разработка и испытание простейших представленных в машинном коде программ для реального компьютера и для виртуальных компьютеров.

Т е м а т и ч е с к и й р а з д е л 3

А л г о р и т м ы и п р о г р а м м и р о в а н и е

- 3.1. Алгоритм как фундаментальное неопределимое понятие кибернетики и вообще математики (и, быть может, вообще всей системы человеческих знаний). Способы неформального представления алгоритмов (словесные описания, блок-схемы и другие). Понятие о вообще разрешимой и об алгоритмически разрешимой задаче (проблеме). Понятие о проблеме разрешимости Гильберта (*das Entscheidungsproblem*). Невозможность математически состоятельного исследования проблемы разрешимости с опорой лишь на представление алгоритмов неформальными способами. Описание алгоритмов на специально для этого созданных искусственных языках (алгоритмических языках, языках программирования) как единственный известный сегодня способ практически значимого и математически состоятельного представления (формализации) алгоритмов. История языков программирования. Легендарные языки программирования. Распространенные в наши дни языки программирования. Исчезнет ли в будущем программирование (например, в связи с широким внедрением компьютерных систем (систем искусственного интеллекта), не требующих программирования, но способных обучаться (подобно человеку) на примерах)?
- 3.2. Как кибер принимает решения? Разветвленные алгоритмы. Синтаксис описания ветвления. Разработка и испытание в действии программ, формализующих разветвленные алгоритмы.
- 3.3. Полное заикливание программ (бесконечное заикливание). Конечность времени получения исполнителем решения поставленной задачи как неотъемлемая сущностная характеристика алгоритма. Принципиальная дефектность программ, входящих в вечный цикл, – неописание ими никакого алгоритма. Конечные (невечные) циклы. Конечное заикливание как распространенный прием оптимизации алгоритмов, а иногда и единственный способ их формализации. Синтаксис циклических конструкций. Типичные циклические конструкции. Вложенные циклы. Разработка и испытание в действии программ, формализующих конечно-циклические алгоритмы.
- 3.4. Идея повторного использования программного кода. Номинальная инкапсуляция программного кода для повторного (быть может, многократного) использования. Подпрограммирование. Разновидности подпрограмм (процедуры, функции) и синтаксис подпрограммирования. Способы получения подпрограммами данных для обработки (значений аргументов) и степень автономности работы подпрограмм с данными. Проблема защиты «внешних» данных от модификации подпрограммой. Понятие о контексте выполнения подпрограммы и об области видимости имен (значений ссылок, указателей), номинирующих данные (и подпрограммы). Возможность вызова подпрограммы ею же самой с модифицированным (скорее всего) набором значений аргументов. Понятие о рекурсивных алгоритмах. Принципиальная возможность частично-рекурсивного (по Черчу) представления любого алгоритма. Преимущества

циклических алгоритмов перед рекурсивными (быстродействие, экономия ресурсов памяти и другие). Преимущества рекурсивных алгоритмов перед циклическими (понятность, прозрачность, простота). Примеры элегантных рекурсивных алгоритмов (в сравнении с их громоздкими циклическими аналогами).

- 3.5. Понятие об игре как о своеобразном способе математического моделирования реальности и о наличии в математике возникшего относительно недавно стремительно развивающегося раздела, называемого теорией игр. Понятие об играх (в общечеловеческом, нематематическом смысле слова) как о своеобразном «полигоне» испытания передовых математических (в частности, кибернетических) идей, подходов, методов, техник, технологий и машин. Абсолютная недопустимость увлечения человека (по причине полного неотвратимого и необратимого разрушения человеческой личности) компьютерными играми, принимаемыми как способ отрешения от реальной действительности. Бесспорная актуальность кибернетико-игрового моделирования многих областей человеческого и природного бытия (политика, социум, экономика, прогнозирование погоды и сейсмических явлений, другие области, управляемые пока что мало изученными фундаментальными законами), плохо поддающихся моделированию другими способами. Игровые алгоритмы.

Т е м а т и ч е с к и й р а з д е л 4

И н ф о р м а ц и о н н ы е т е х н о л о г и и

- 4.1. Понятие об операционной системе компьютерного устройства и о пользовании операционной системой. История компьютерных операционных систем. Распространенные в наше время операционные системы. Понятие об открытых (закрытых), свободно (несвободно) распространяемых, бесплатных (платных) операционных системах и вообще об открытом (закрытом), свободно (несвободно) распространяемом, бесплатном (платном) программном обеспечении для компьютерных устройств. Перспективность и способы получения открытого, свободного, бесплатного программного обеспечения. Понятие об алгоритмическом и цифровом суверенитете государства как об одной из высших социальных ценностей. Корпоративно-глобалистические угрозы алгоритмическому и цифровому суверенитету государства в контексте пользования людьми операционными системами и другим программным обеспечением для компьютерных устройств. Типичные подсистемы современной компьютерной операционной системы – пользование ими и применение их в собственном кибернетическом творчестве.
- 4.2. Обзор техник и технологий компьютерного препринтинга – разработки готовых к отправке на печать и (или) к просмотру на экране компьютерного устройства цифровых моделей (макетов) печатных документов. «Офисные» текстовые процессоры. Способы получения и использования открытых, свободных, бесплатных «офисных» текстовых процессоров. Обзор компьютерных

программных комплексов, реализующих идею процессоров электронных таблиц. «Офисные» процессоры электронных таблиц. Способы получения и использования открытых, свободных, бесплатных «офисных» процессоров электронных таблиц. Обзор компьютерных программных комплексов, предоставляющих возможности изготовления и демонстрации мультимедийных анимированных слайд-презентаций, возможности широкого массового компьютерно-полиграфического препринтинга и паблишинга. Способы получения и использования открытых, свободных, бесплатных программных комплексов указанного рода. Перспективность решения типичных «офисных» задач программно-компьютерными средствами «неофисного» характера. Примеры кибернетико-творческих решений типичных «офисных» задач «неофисными» средствами. Значимость свободного кибернетического творчества народных масс в деле борьбы за прогресс человека и человечества.

- 4.3. Проблемы и типичные задачи обработки больших массивов хорошо (четко) структурированных данных. Понятие о базе данных и системе управления базой данных (СУБД). Реляционная модель базы данных. Нереляционные модели баз данных (иерархическая, логическая и другие). История систем управления базами данных. Нетрадиционные способы организации баз данных и систем управления ими (например, путем использования возможностей логического программирования). Преимущественная распространенность в наше время систем управления базами данных, основанных на реляционной модели. Язык структурированных запросов к системе управления базой данных, основанной на реляционной модели. Разработка, заполнение данными и испытательная эксплуатация действующей системы управления базой данных.

III. МЕТОДИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ПРОГРАММЫ

При проведении занятий с обучающимися преподаватель должен учитывать интересы обучающихся, их интеллектуальные и физические возможности. Занятия должны приносить обучающимся радость, пробуждать в них желание изучать информатику, развивать творческие способности. Одной из важнейших воспитательных функций программы является профессиональная ориентация обучающихся, пропаганда выбора ими для дальнейшего обучения направлений подготовки, специальностей высшего образования, предполагающих интенсивное формирование, развитие и использование кибернетических знаний и кибернетической культуры личности (в первую очередь – кибернетических, математических, естественно-научных и инженерных направлений подготовки, специальностей).

Программа предполагает использование преподавателем разнообразных методов обучения: теоретических (изложение учебного материала, беседа, показ способов решения учебных задач), практических (выполнение упражнений, отработка типовых способов осуществления учебной деятельности, тренинги), визуальных (демонстрация и использование статических, динамических и интерак-

тивных визуализаций кибернетических объектов и процессов, демонстрация и рекомендация к самостоятельному просмотру фотографий, слайд-презентаций, фрагментов мультипликационных, документальных, художественных фильмов, развивающих интерес к изучению кибернетики, углубляющих, расширяющих, обобщающих и систематизирующих элементы кибернетических представлений и кибернетической культуры обучающихся), игровых (использование элементов игры при проведении занятий), исследовательских (частично-поисковая и доступная на характеризующем обучающихся уровне развитости интеллектуальных способностей усиленная для них поисковая, исследовательская деятельность) и других.

Преподавателю при проведении занятий и при организации самостоятельной работы обучающихся рекомендуется применять элементы прогрессивных технологий обучения: технологии личностно-ориентированного и развивающего обучения, технологии коллективной творческой деятельности и технологии взаимообучения, эвристические технологии и технологии проблемного обучения, технологии индивидуализации и персонализации обучения, современные информационно-коммуникационные образовательные технологии. При выборе используемых в образовательном процессе компьютерных средств и программного обеспечения должны соблюдаться требования принципа стремления к цифровому и алгоритмическому суверенитету государства.

Занятия с обучающимися проводятся в помещениях учебных корпусов Донецкого государственного университета, соответствующих требованиям законодательства Российской Федерации в области санитарно-эпидемиологического благополучия населения. На первом занятии преподавателем с группой обучающихся проводится инструктаж по безопасности жизнедеятельности, что фиксируется в соответствующем журнале. С родителями (законными представителями) несовершеннолетних обучающихся преподавателем о вопросах безопасности жизнедеятельности обучающихся проводится беседа.

Использование при реализации программы методов и средств обучения и воспитания, образовательных технологий, наносящих вред физическому или психическому здоровью обучающихся, запрещается.

IV. ФОРМА ПОДВЕДЕНИЯ ИТОГОВ ОСВОЕНИЯ ПРОГРАММЫ

Подведение итогов освоения программы производится в форме итоговой аттестации обучающихся, реализуемой путем выполнения обучающимися заданий итоговой контрольной работы и оценивания преподавателем результатов выполнения заданий по пятибалльной шкале («отлично», «хорошо», «удовлетворительно», «неудовлетворительно») в соответствии с заранее установленными критериями. Участие обучающихся в итоговой аттестации является необязательным (добровольным). Обучающимся, успешно прошедшим итоговую аттестацию («отлично», «хорошо», «удовлетворительно»), выдается сертификат об успешном освоении программы установленного Донецким государственным университетом образца.

Итоговая контрольная работа состоит из двух частей, содержащих 24 задания. Часть 1 содержит 12 заданий с кратким ответом базового и повышенного уровней сложности. Часть 2 содержит 12 заданий с развернутым ответом повышенного и высокого уровней сложности. На проведение занятий для непосредственной подготовки к выполнению итоговой контрольной работы отводится 4 академических часа. На выполнение контрольной работы отводится 4 академических часа. Примеры заданий итоговой контрольной работы и критериев оценивания приведены ниже.

Часть 1

1. Анализ информационных моделей. В некотором районе (рис. 1) имеется семь населенных пунктов: A , B , C , D , E , F и G . Приведенный на рисунке граф изображает схему дорожного сообщения этих населенных пунктов. Эту схему следует понимать исключительно в *топологическом* смысле. Например, наличие у графа ребра AC показывает, что существует дорога, соединяющая пункты A и C , а наличие ребра AB говорит о том, что имеется дорога, соединяющая пункты A и B . Ничего кроме топологически-осмысленной информации граф не представляет. Например, ребро AC на рисунке длиннее ребра AB , но это вовсе не означает, что дорога, соединяющая пункты A и C , действительно длиннее дороги, соединяющей пункты A и B . Кроме того, все ребра графа изображены в виде отрезков прямых линий, но это отнюдь не является свидетельством того, что дороги, соединяющие населенные пункты, — прямые.

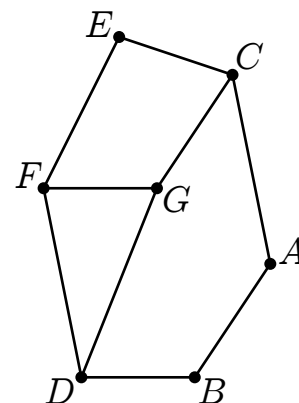


Рис. 1

Справа (рис. 2) приведена таблица, содержащая сведения о дорожном сообщении тех же семи населенных пунктов. Пункты обозначены номерами 1, 2, ..., 7. Так как построением таблицы и вычерчиванием приведенной выше (рис. 1) схемы занимались два разных исполнителя (и они действовали независимо друг от друга), нумерация населенных пунктов в таблице никак не связана с буквенными обозначениями на графе. Например, 1 — вовсе не обязательно номер пункта A , 2 — вовсе не обязательно номер пункта B и так

	1	2	3	4	5	6	7
1	—	—	—	30	3	—	5
2	—	—	—	21	—	13	—
3	—	—	—	—	39	53	2
4	30	21	—	—	—	—	—
5	3	—	39	—	—	8	—
6	—	13	53	—	8	—	—
7	5	—	2	—	—	—	—

Рис. 2

далее. Смысл находящихся в таблице значений ясен из следующих примеров. Во 2-й строке во 2-м столбце находится прочерк. Это означает, что нет дороги, по которой можно, выехав из 2-го пункта, вернуться снова во 2-й пункт, двигаясь все время «только вперед», не разворачиваясь и не посещая никакой другой пункт. В 5-й строке в 4-м столбце находится прочерк. Это означает, что нет дороги, непосредственно соеди-

няющей 5-й пункт с 4-м. Тот же смысл имеет прочерк, находящийся в 5-м столбце в 4-й строке. В 7-й строке в 1-м столбце находится значение 5. Это означает, что есть дорога протяженностью 5 км, непосредственно соединяющая 7-й пункт с 1-м. Тот же смысл имеет значение 5, находящееся в 7-м столбце в 1-й строке. Об этой таблице можно говорить, что она *симметрична* относительно своей *главной диагонали*, соединяющей верхний левый угол таблицы с правым нижним.

Определите, какова сумма протяженности дороги, соединяющей непосредственно пункты D и G , и протяженности дороги, соединяющей непосредственно пункты A и C . Ответ выразите в километрах. Запишите только числовое значение. Ваш ответ: _____.

Задание 1 является тестовым заданием с кратким ответом. От испытуемого не требуется приведение полного текста решения. Однако, это задание не может быть отнесено к типично-репродуктивным. Оно является эвристическим (частично-поисковым). Испытуемый может решить предложенную задачу разными способами. В частности, он может прийти к верному ответу, логически умозаключая от сведений, данных в условии задачи. Пример такого рассуждения приведен ниже.

а) Населенные пункты, непосредственно соединенные какой-либо дорогой, будем называть *соседями*. Пункт, имеющий ровно n соседей, будем называть *n -валентным*. В рассматриваемом районе имеются только двухвалентные (A , B и E) и трехвалентные (C , D , F и G) пункты. Тот факт, что на текущем этапе исследования неизвестно, какие номера присвоены населенным пунктам, можно записать так: $A_?B_?C_?D_?E_?F_?G_?$.

б) 1-й населенный пункт (и только он, в чем можно убедиться, изучив приведенную в условии задачи таблицу) является трехвалентным, имеет двух двухвалентных соседей и одного трехвалентного соседа. Указанным свойством обладает пункт C (и только он, в чем можно убедиться, рассмотрев приведенный в условии задачи граф). Значит, 1-й населенный пункт — это пункт C . Единственным трехвалентным соседом 1-го пункта является 5-й пункт, а единственный трехвалентный сосед пункта C — пункт G . Следовательно, пункт G — это 5-й населенный пункт. Можно записать: $A_?B_?C_1D_?E_?F_?G_5$.

в) 7-й населенный пункт (и только он) двухвалентен, а оба его соседа — трехвалентны. Таким свойством обладает только пункт E . Значит E — 7-й населенный пункт. У 1-го населенного пункта (пункта C) есть два двухвалентных соседа (пункты A и E) — 4-й пункт и 7-й пункт. Но 7-й пункт — это E . Следовательно, 4-й пункт — это A . Поэтому правомерна запись: $A_4B_?C_1D_?E_7F_?G_5$.

г) У 7-го пункта (это E) есть два соседа (это C и F) — 1-й и 3-й пункты. C — 1-й пункт. Следовательно, F — 3-й. Итак: $A_4B_?C_1D_?E_7F_3G_5$.

ε) Соседями 3-го пункта (это F) являются пункты D , E и G — 5-й, 6-й и 7-й пункты. E — 7-й, G — 5-й, значит D — это 6-й населенный пункт. Получаем: $A_4B_2C_1D_6E_7F_3G_5$.

ζ) Единственный пункт, номер которого пока что остался неизвестным, — пункт B . Единственный соответствующий условию задачи номер, который пока что не присвоен никакому пункту, — номер 2. Если предположить, что решение задачи существует, можно утверждать: B — 2-й населенный пункт. Однако, можно обойтись без указанного предположения (и тогда существование решения задачи будет уже не гипотезой, а доказанным фактом). Действительно, 4-й населенный пункт (это A) имеет двух соседей (пункты B и C) — 1-й и 2-й населенные пункты. 1-й — это C . Следовательно, 2-й — это B . Имеющаяся теперь информация о том, какими номерами обозначены населенные пункты рассматриваемого района, является полной: $A_4B_2C_1D_6E_7F_3G_5$. Отметим еще вот что: на каждом шаге нами вполне обоснованно принималось *единственно возможное* утверждение о соответствии того или иного номера тому или иному пункту. Следовательно, условию задачи соответствует единственный (тот, который нами найден) способ нумерации населенных пунктов. Значит, решение задачи не только *существует*, но и *единственно*.

η) Дорога, соединяющая непосредственно пункты D и G (то есть 6-й и 5-й пункты), имеет протяженность 8 км. Дорога, соединяющая непосредственно пункты A и C (это 4-й и 1-й пункты), имеет протяженность 30 км. Сумма указанных протяженностей равна 38 км. В ответ записывается значение 38. Задача полностью решена.

Если испытуемому доступны компьютерные средства, решение задачи может быть выполнено с их использованием. Например, весьма вероятно, что во время испытания испытуемый может работать с компьютером, операционная система которого предоставляет возможность набирать тексты, сохранять их в простых текстовых файлах и открывать эти файлы с помощью корректно функционирующего стандартного Веб-браузера. Большинство современных компьютеризированных рабочих мест (и мест персонального пользования) указанным требованиям удовлетворяют. Ниже приведен пример решения обсуждаемой задачи с использованием указанных возможностей.

Нетривиальной частью решения поставленной задачи является получение ответа на вопрос о том, какой именно номер присвоен составителем таблицы каждому из представленных на графе (схеме) населенных пунктов. Каждому из семи населенных пунктов присвоить некоторый номер, взятый из диапазона 1, 2, ..., 7, можно $7! = 5040$ способами (это — число перестановок семи элементов 1, 2, ..., 7). Однако, не каждая перестановка задает нумерацию, удовлетворяющую условию задачи. Например, перестановка (7 6 5 4 3 2 1) задает нумерацию $A_7B_6C_5D_4E_3F_2G_1$, кото-

рая условию задачи не удовлетворяет. Действительно, из представленных в таблице (рис. 2) данных следует, что 7-й пункт (мы сейчас должны считать, что это A) непосредственно соединен дорогой с 1-м пунктом (мы сейчас должны считать, что это G). Но на графе (рис. 1) нет дороги, непосредственно соединяющей A с G .

Отсутствие или наличие дороги на графе мы «видим своими глазами». Подобное «видение» не является математически осмысленным действием. Нужно построить такую информационную модель данных, представляемых графом, чтобы можно было указать математически осмысленную эффективную процедуру (алгоритм) установления соответствия условию задачи нумераций, задаваемых перестановками указанного выше рода.

Справа (рис. 3) приведена подходящая информационная модель — содержащая 7 строк и 7 столбцов таблица, которую дальше будем называть *матрицей смежности* вершин изображенного на рис. 1 графа. Находящиеся в таблице значения будем называть значениями *элементов* матрицы смежности. Значение элемента, находящегося в i -й строке ($i = 1, 2, \dots, 7$) в j -м столбце ($j = 1, 2, \dots, 7$) матрицы смежности, будем обозначать a_{ij} . Смысл значений элементов матрицы смежности поясним примерами. $a_{22} = 0$. Это означает, что на графе нет «петли» при вершине B — нет дороги, по которой можно, выехав из пункта B , вернуться в пункт B , двигаясь «все время в одну сторону», не разворачиваясь и не посещая никакой другой населенный пункт. $a_{63} = 0$. Это означает, что на графе нет ребра, соединяющего вершины F и C (то есть нет дороги, непосредственно соединяющей населенные пункты F и C). $a_{37} = 1$. Это означает, что имеется (и только одно) ребро графа, соединяющее вершины C и G (есть одна — и только одна — дорога, непосредственно соединяющая населенные пункты C и G).

	A	B	C	D	E	F	G
A	0	1	1	0	0	0	0
B	1	0	0	1	0	0	0
C	1	0	0	0	1	0	1
D	0	1	0	0	0	1	1
E	0	0	1	0	0	1	0
F	0	0	0	1	1	0	1
G	0	0	1	1	0	1	0

Рис. 3

Опираясь на данные приведенной на рис. 2 таблицы, можно составить размещенную справа (рис. 4) еще одну матрицу смежности — матрицу смежности населенных пунктов рассматриваемого района, обозначенных присвоенными им номерами. Эта новая матрица смежности содержит элемент со значением 1 в той (и только в той) позиции, в которой таблица, приведенная на рис. 2, содержит числовое значение, а не прочерк. В тех же позициях, в которых таблица, приведенная на рис. 2, содержит прочерк, новая матрица смежности содержит элемент со значением 0. Смысл значений элементов новой матрицы смежности аналогичен смыслу значений элементов матрицы

	1	2	3	4	5	6	7
1	0	0	0	1	1	0	1
2	0	0	0	1	0	1	0
3	0	0	0	0	1	1	1
4	1	1	0	0	0	0	0
5	1	0	1	0	0	1	0
6	0	1	1	0	1	0	0
7	1	0	1	0	0	0	0

Рис. 4

смежности, приведенной на рис. 3, с тем лишь отличием, что населенные пункты обозначены не буквами, а присвоенными им номерами. Значение элемента, находящегося в i -й строке в j -м столбце матрицы смежности, приведенной на рис. 4, будем обозначать символом $b(i, j)$. Матрицу смежности, приведенную на рис. 3, далее называем первой, а приведенную на рис. 4 — второй.

Нетрудно видеть, что обе матрицы смежности имеют одинаковые размеры (одинаковое число строк и столбцов), а также содержат одинаковое число нулевых и ненулевых элементов (нулей и единиц). Количество единиц в каждой матрице смежности равно 18 — удвоенное число ребер графа, изображенного на рис. 1. Это не удивительно, ведь каждая из матриц смежности «учитывает» всякую соединяющую соседние населенные пункты дорогу дважды — в двух симметричных относительно главной диагонали позициях. Обе матрицы смежности содержат *одну и ту же* информацию, но закодированную с использованием двух разных *ключей*. Ключом к интерпретации данных первой матрицы является «естественный» порядок следования первых семи букв латинского алфавита. Ключ к интерпретации данных второй матрицы — перестановка семи элементов 1, 2, ..., 7, задающая удовлетворяющую условию задачи «перепутанную нумерацию» этих семи букв. Осознав сказанное, можем указать условие, которому должна удовлетворять задающая удовлетворяющую условию задачи нумерацию букв перестановка и только такая перестановка. В дальнейшем такие перестановки будем называть *корректными*.

Поясним сначала нашу мысль примером. Предположим, что корректная перестановка имеет вид (4 2). Она задает следующую нумерацию латинских букв (имен населенных пунктов, вершин графа на рис. 1): $A_4 B_2 \dots$. В первой матрице смежности $a_{12} = 1$ (и $a_{21} = 1$, ибо матрица симметрична). Это сигнализирует о том, что (в соответствии с «естественным» ключом) на графе имеется ребро AB (и оно же — ребро BA). В соответствии с «перепутанным» ключом, заданным указанной выше (согласно предположению, корректной) перестановкой, ту же информацию передает полученный из второй матрицы смежности сигнал $b(4, 2) = 1$ (и сигнал $b(2, 4) = 1$, ибо вторая матрица тоже симметрична). Теперь сформулируем условие корректности перестановки в общем виде: *перестановка $(p_1 p_2 \dots p_7)$ семи элементов 1, 2, ..., 7 корректна тогда и только тогда, когда:*

$$\forall i, j = 1, 2, \dots, 7 \quad b(p_i, p_j) = a_{ij}. \quad (1)$$

Условимся еще вот о чем. Значение, находящееся в представленной на рис. 2 таблице в i -й строке в j -м столбце будем обозначать $r(i, j)$. Если это значение — не прочерк, будем интерпретировать его как то

число, десятичную запись которого это значение представляет. Тогда процедуру построения второй матрицы смежности (рис. 4) по данным таблицы, представленной на рис. 2, можно формализовать так:

$$\forall i, j = 1, 2, \dots, 7 \quad b(i, j) = \begin{cases} 1, & \text{если } r(i, j) - \text{не прочерк.} \\ 0, & \text{если } r(i, j) - \text{прочерк.} \end{cases} \quad (2)$$

Если перестановка $(p_1 p_2 \dots p_7)$ семи элементов 1, 2, ..., 7 корректна, соответствующий задаваемой ею нумерации ответ на вопрос задачи может быть легко вычислен. Обозначим искомое значение буквой S . Тогда $S = DG + AC$, где $DG = r(p_4, p_7)$, а $AC = r(p_1, p_3)$. Следовательно, имеем:

$$S = r(p_4, p_7) + r(p_1, p_3). \quad (3)$$

Входные данные обсуждаемой задачи — это значения элементов первой матрицы смежности (рис. 3) и данные, содержащиеся в таблице, представленной на рис. 2. Словесное описание алгоритма решения задачи дается ниже.

- ① Используя содержащиеся в представленной на рис. 2 таблице данные, выполнив процедуру (2), построить вторую матрицу смежности. Будет получен результат, представленный на рис. 4.
- ② Построить список L — список всех перестановок семи элементов 1, 2, ..., 7. Каждая перестановка должна входить в список только один раз. Список L будет содержать $7! = 5040$ перестановок.
- ③ Последовательно перебрать все перестановки, составляющие список L . Для каждой перестановки $(p_1 p_2 \dots p_7)$, взятой из списка L , выполнить проверку на корректность, используя условие (1). Если перестановка корректна: вычислить ответ S на вопрос задачи по формуле (3), вывести соответствующую перестановке нумерацию населенных пунктов $A_{p_1} B_{p_2} C_{p_3} D_{p_4} E_{p_5} F_{p_6} G_{p_7}$ и вычисленный ответ S .

В результате исполнения описанного выше алгоритма может быть получено одно или несколько решений обсуждаемой задачи. Возможно также, что не будет получено ни одного решения. В любом случае, результат будет целиком удовлетворительным по общему правилу, в соответствии с которым, *решить задачу — это значит найти все ее решения или доказать, что решений не существует*. Доказанность будет совершенной, ведь фактически будет осуществлен полный математически состоятельный перебор.

Исполнение предложенного выше алгоритма решения обсуждаемой задачи может быть осуществлено встроенным в Веб-страницу сценарием (скриптом) на языке JavaScript. Веб-страница может быть открыта с помощью корректно работающего стандартного Веб-браузера. Содер-

жащий код указанной Веб-страницы файл task01.html опубликован по адресу (*URL*) <http://docsagmax.ru/donsu/aea/acaec/programs/informatics/task01.html>. Далее приведен полный листинг этого файла.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Task 01</title>
    <style type="text/css">
      div#output {
        font-family: monospace;
        font-size: 14pt;
      }
      div#output sub {
        font-size: 10pt;
      }
    </style>
    <script type="text/javascript">
      // Входные данные.
      // Первая матрица смежности –
      // информационная модель графа.
      var graphAdjacencyMatrix = [
        /*      A B C D E F G */
        /* A */ [0,1,1,0,0,0,0],
        /* B */ [1,0,0,1,0,0,0],
        /* C */ [1,0,0,0,1,0,1],
        /* D */ [0,1,0,0,0,1,1],
        /* E */ [0,0,1,0,0,1,0],
        /* F */ [0,0,0,1,1,0,1],
        /* G */ [0,0,1,1,0,1,0]
      ];
      // Таблица с рисунка 2.
      var rTable = [
        /*      1  2  3  4  5  6  7 */
        /* 1 */ ['-', '-', '-', 30, 3, '-', 5],
        /* 2 */ ['-', '-', '-', 21, '-', 13, '-'],
        /* 3 */ ['-', '-', '-', '-', 39, 53, 2],
        /* 4 */ [ 30, 21, '-', '-', '-', '-', '-'],
        /* 5 */ [ 3, '-', 39, '-', '-', 8, '-'],
        /* 6 */ ['-', 13, 53, '-', 8, '-', '-'],
        /* 7 */ [ 5, '-', 2, '-', '-', '-', '-']
      ];
      // Конец входных данных.
      // Строится вторая матрица смежности
      // путем выполнения процедуры (2).
      var secondAdjacencyMatrix = [
        [0,0,0,0,0,0,0],
        [0,0,0,0,0,0,0],
        [0,0,0,0,0,0,0],
        [0,0,0,0,0,0,0],
        [0,0,0,0,0,0,0],
        [0,0,0,0,0,0,0],
        [0,0,0,0,0,0,0]
      ];
    </script>
  </head>
  <div id="output">
    <sub>
      <table border="1">
        <caption>Матрица смежности графа</caption>
        <tr><th></th><th>A</th><th>B</th><th>C</th><th>D</th><th>E</th><th>F</th><th>G</th></tr>
        <tr><th>A</th><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
        <tr><th>B</th><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td></tr>
        <tr><th>C</th><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td></tr>
        <tr><th>D</th><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr>
        <tr><th>E</th><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
        <tr><th>F</th><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td></tr>
        <tr><th>G</th><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td></tr>
      </table>
    </sub>
  </div>
</html>
```

```
for (var i = 0; i < secondAdjacencyMatrix.length; i++) {
    for (var j = 0; j < secondAdjacencyMatrix[i].length; j++) {
        if (rTable[i][j] != '-') secondAdjacencyMatrix[i][j] = 1;
    }
}
// Подпрограмма-функция removeElement. Удаляет из массива элемент с
// указанным индексом. Возвращает усеченный массив.
function removeElement(elementIndex, arRay) {
    var newArr = [];
    for (var i = 0; i < arRay.length; i++)
        if (i != elementIndex)
            newArr[newArr.length] = arRay[i];
    return newArr;
}
// Подпрограмма-функция insertFirst. Вставляет переданный ей эле-
// мент в первую позицию переданного ей массива. Возвращает расши-
// ренный массив.
function insertFirst(elElement, arRay) {
    var newArr = [elElement];
    for (var i = 0; i < arRay.length; i++)
        newArr[newArr.length] = arRay[i];
    return newArr;
}
// Подпрограмма-функция constructAllPermutations. Получает массив –
// набор переставляемых элементов (их n штук). Возвращает массив,
// каждый элемент которого – некоторая перестановка элементов ис-
// ходного массива. В возвращаемом массиве все элементы (переста-
// новки) различны. Их n! штук.
function constructAllPermutations(arRay) {
    if (arRay.length == 1) {
        return [arRay];
    }
    var retArr = [];
    for (
        var currentIndex = 0;
        currentIndex < arRay.length;
        currentIndex++
    ) {
        var removedElement = arRay[currentIndex];
        var restedArray = removeElement(currentIndex, arRay);
        var restedPermutations = constructAllPermutations (
            restedArray
        ); // Рекурсивный вызов.
        for (
            var i = 0;
            i < restedPermutations.length;
            i++
        ) retArr[retArr.length] = insertFirst (
            removedElement,
            restedPermutations[i]
        );
    }
    return retArr;
}
```

```
// Подпрограмма-функция printNumeration.  
// Получает перестановку, задающую некоторую  
// нумерацию пунктов A, B, ..., G. Строит  
// запись соответствующей нумерации.  
function printNumeration(arRay) {  
    var arrLetters = ['A','B','C','D','E','F','G'];  
    var strPr = '';  
    for (var i = 0; i < arRay.length; i++) {  
        strPr += arrLetters[i];  
        strPr += '<sub>' + arRay[i] + '</sub>';  
    }  
    return strPr;  
}  
// Подпрограмма-функция isPermutationCorrect.  
// Проверяет корректность нумерации, заданной  
// перестановкой, по соответствию условию (1).  
function isPermutationCorrect(arrPerm) {  
    for (var i = 0; i < 7; i++)  
        for (var j = 0; j < 7; j++)  
            if (  
                secondAdjacencyMatrix[arrPerm[i]-1][arrPerm[j]-1]  
                !=  
                graphAdjacencyMatrix[i][j]  
            ) return false;  
    return true;  
}  
// Главный программный модуль.  
function main() {  
    var el = document.getElementById('output');  
    var arrPerms = constructAllPermutations([1,2,3,4,5,6,7]);  
    var strPr = '';  
    var n = 0;  
    for (var i = 0; i < arrPerms.length; i++) {  
        if (  
            isPermutationCorrect(arrPerms[i])  
        ) {  
            strPr +=  
                ++n + ') ' + printNumeration(arrPerms[i]) +  
                ' S=' +  
                (  
                    rTable[arrPerms[i][3]-1][arrPerms[i][6]-1] +  
                    rTable[arrPerms[i][0]-1][arrPerms[i][2]-1]  
                ) + '<br>';  
        }  
    }  
    el.innerHTML = strPr;  
}  
</script>  
</head>  
<body onLoad="main()">  
    <div id="output">  
    </div>  
</body>  
</html>
```

Интерпретация файла `task01.html` корректно функционирующим стандартным Веб-браузером приводит к получению приведенного ниже сообщения.

1) $A_4B_2C_1D_6E_7F_3G_5 \quad S=38$

Итак, произведен математически состоятельный полный перебор. Решение обсуждаемой задачи существует. Оно является единственным. Искомая сумма протяженностей равна 38 км. В ответ записывается значение 38. Задача полностью решена.

2. Построение таблиц истинности логических выражений. Миша верно построил таблицу истинности логической функции f , заданной выражением

$$f = ((w \rightarrow y) \rightarrow x) \vee \neg z.$$

				f
		1		0
	0			0
	1	0	0	0

Мишина рукопись была повреждена. На рис. 5 показан фрагмент рукописи, который удалось восстановить. В этом восстановленном фрагменте все имеющиеся значения логических переменных и значения логической функции — верные. Однако, сведения о некоторых значениях логических переменных и о том, какой столбец соответствует каждой из логических переменных, восстановить не удалось. Кроме того, восстановленный фрагмент содержит лишь три различных строки верно построенной Мишей таблицы истинности. Определите, какой столбец в восстановленном фрагменте соответствует каждой из логических переменных w , x , y и z . В ответ запишите буквы, обозначающие логические переменные, в том порядке, в каком в восстановленном фрагменте слева направо следуют столбцы таблицы истинности. Буквы в ответе пишите подряд, не вставляя между ними никаких разделителей. Например, если Вы считаете, что порядок следования букв совпадает с «естественным» порядком их следования в конце латинского алфавита, в ответ запишите $wxyz$. Ваш ответ: _____.

Рис. 5

Задание 2 является тестовым заданием с кратким ответом. От испытуемого не требуется приведение полного текста решения. Однако, невозможно утверждать, что для получения верного ответа достаточно стандартно применить имеющиеся у испытуемого знания. Это задание — частично-поисковое (эвристическое). Задачу можно решить разными способами. В частности, можно прийти к верному ответу путем рассуждений и сопоставлений. Пример решения такого рода приведен ниже.

а) Выражение, задающее функцию f , имеет вид $f = \dots \vee \neg z$. По правилу «если хотя бы один дизъюнкт — истина, то дизъюнкция — истина», если $z=0$, то $\neg z=1$, и $f=1$. Во фрагменте таблицы истинности на рис. 5

столбец f содержит только нули. Следовательно, столбец z не должен содержать ни одного значения, наверняка являющегося нулем (ведь иначе в столбце f появились бы единицы). Ни одного значения, наверняка являющегося нулем, не содержит 1-й столбец и только он. Значит, 1-й столбец — это столбец z . Можем восстановленный фрагмент мишиной рукописи считать таким, как показано на рис. 6.

z				f
1		1		0
1	0			0
1	1	0	0	0

Рис. 6

б) Первый дизъюнкт выражения, задающего логическую функцию f , имеет вид $\dots \rightarrow x$. По правилу «истина — из чего угодно», когда $x = 1$, первый дизъюнкт выражения для f принимает значение 1, а тогда и f принимает значение 1. Поскольку во фрагменте таблицы истинности на рис. 6 столбец f содержит только нули, столбец x не должен содержать ни одного значения, наверняка являющегося единицей. Таков 4-й столбец и только он. Значит, можно восстановленный фрагмент мишиной рукописи считать таким, как показано на рис. 7.

z			x	f
1		1	0	0
1	0		0	0
1	1	0	0	0

Рис. 7

γ) Имеются две возможности именования 2-го и 3-го столбцов фрагмента таблицы истинности на рис. 7. Первая возможность: 2-й столбец — столбец w , 3-й столбец — столбец y . Тогда по данным последней строки фрагмента таблицы истинности $z = 1$, $w = 1$, $y = 0$, $x = 0$. Для f вычисляем:

$$f = ((w \rightarrow y) \rightarrow x) \vee \neg z = ((1 \rightarrow 0) \rightarrow 0) \vee \neg 1 = (0 \rightarrow 0) \vee 0 = 1 \vee 0 = 1.$$

Этот результат вступает в противоречие со значением $f = 0$, находящимся в последней строке фрагмента таблицы истинности. Остается вторая (и теперь уже единственная) возможность именования рассматриваемых столбцов: 2-й столбец — столбец y , 3-й столбец — столбец w . Тогда по данным последней строки $z = 1$, $y = 1$, $w = 0$, $x = 0$. Для f можно вычислить:

$$f = ((w \rightarrow y) \rightarrow x) \vee \neg z = ((0 \rightarrow 1) \rightarrow 0) \vee \neg 1 = (1 \rightarrow 0) \vee 0 = 0 \vee 0 = 0.$$

Значение $f = 0$ полностью соответствует данным, представленным столбцом f фрагмента таблицы истинности (рис. 7). Следовательно, восстановленный фрагмент мишиной рукописи приобретает вид, показанный на рис. 8. Задание, сформулированное в условии задачи, выполнено: нам теперь известно, какой столбец в восстановленном фрагменте соответствует каждой из логических переменных w , x , y и z . В ответ записывается слово $zywx$. Задача полностью решена.

z	y	w	x	f
1		1	0	0
1	0		0	0
1	1	0	0	0

Рис. 8

3. Базы данных. По адресу <http://docsagmax.ru/donsu/aea/acaec/programs/informatics/confectionery/> доступен интерфейс пользователя системы управления реляционной базой данных (СУБД) «Кондитерские изделия», содержащей сведения о поставках в магазины нескольких районов некоторого города и продажах в них кондитерских изделий. Пользователь может работать с СУБД, используя идентификатор `confectionery` и пароль `pupil`. СУБД принимает от пользователя запросы на языке **SQL** и выполняет их. Выполнив запрос `PRAGMA table_list`, пользователь может убедиться, что база данных (кроме служебных) содержит три содержательных таблицы: таблицу «Магазин» (формальное наименование `Shop`), таблицу «Движение товаров» (формальное имя `ProductsMovement`) и таблицу «Товар» (формальное наименование `Product`).

Выполнив запрос `PRAGMA table_info(Shop)`, пользователь может увидеть, что таблица `Shop` имеет три поля (три формальных наименования ее граф, столбцов): `shop_id` (уникальный идентификатор магазина, тип данных `TEXT`), `district` (название городского района, в котором находится магазин, тип данных `TEXT`) и `address` (адрес магазина, тип данных `TEXT`). Выполнив запрос `SELECT * FROM Shop`, пользователь может получить все записи (строки) таблицы `Shop` с отображением в каждой строке значений всех полей. Всего таблица содержит 18 записей. Например, магазин с универсальным идентификатором `M10` расположен в городском районе Октябрьский по адресу просп. Революции, 1.

Изучить таблицу `Product` пользователь может, направив к СУБД запрос `PRAGMA table_info(Product)`. У таблицы имеется шесть полей (граф, столбцов): `article` (уникальный идентификатор (артикул) товара, кондитерского изделия, тип данных `INTEGER`), `section` (наименование группы товаров, к которой относится конкретный товар, тип данных этого поля — `TEXT`), `product_name` (наименование товара, тип данных `TEXT`), `measure_unit` (единица измерения количества товара, тип данных этого поля — `TEXT`), `quantity_per_package` (количество единиц товара в одной упаковке, тип данных `INTEGER`), `price_per_package` (цена товара в рублях за одну упаковку, тип данных `INTEGER`). В результате выполнения направленного к СУБД пользователем запроса `SELECT * FROM Product` могут быть получены все записи таблицы `Product` с отображением значений всех полей каждой записи. Таблица содержит 60 записей. Например, товар с артикулом 43, относящийся к группе товаров Печенье, — это товар с наименованием Печенье "Юбилейное". Количество этого товара измеряется в граммах. Одна упаковка содержит 120 граммов печенья. Товар продается по цене 60 рублей за упаковку.

Выполнив запрос `PRAGMA table_info(ProductsMovement)`, можно выявить структуру таблицы `ProductsMovement`. Каждая запись таблицы имеет шесть полей: `operation_id` (уникальный идентификатор операции (номер акта движения товара, под движением товара понимается его поступле-

ние в магазин или продажа), тип данных `INTEGER`), `operation_date` (дата совершения акта движения товара, тип данных `DATE` — дата представляется строковым значением, смысловые группы которого разделяются дефисом, сначала указывается номер года нашей эры, затем — номер месяца в году, затем — номер дня месяца; например, `2023-08-02` — это 2-е августа 2023 года), `shop_id` (уникальный идентификатор магазина, в котором совершен акт движения товара, тип данных `TEXT`; значение этого уникального идентификатора указывает на ту запись в таблице `Shop`, которая содержит сведения о соответствующем магазине), `article` (уникальный идентификатор (артикул) того товара, с которым совершен акт движения, тип данных `INTEGER`; значение этого уникального идентификатора указывает на ту запись в таблице `Product`, которая содержит сведения о соответствующем товаре), `number_of_packages` (количество упаковок товара, с которыми совершен акт движения, тип данных `INTEGER`), и `operation_type` (тип акта движения товара (поступление в магазин или продажа), тип данных `TEXT`).

Получить все записи таблицы `ProductsMovement` с отображением значений всех полей каждой записи можно, выполнив следующий запрос к СУБД: `SELECT * FROM ProductsMovement`. Таблица содержит 6480 записей. Например, запись с `operation_id` равным 5403 сообщает, что в день с меткой даты `2023-08-20` (то есть 20-го августа 2023 года) в магазине с уникальным идентификатором магазина `M1` с товаром артикул 3 был совершен акт движения **Продажа**. Было продано 113 упаковок товара.

Найдите общую массу (в килограммах) всех видов зефира, полученных магазинами, расположенными на проспекте Революции, за период со 2-го по 10-е августа 2023 года включительно. В ответ запишите только числовое значение. Ваш ответ: _____ .

Задание 3 — это тестовое задание с кратким ответом. От испытуемого не требуется приведение полного текста решения. Испытуемый может получить верный ответ, выполнив несколько несложных запросов к системе управления базой данных. Подобный способ решения требует от испытуемого лишь стандартного применения имеющихся знаний. Впрочем, верный ответ может быть получен и путем выполнения одного запроса к СУБД — этот запрос уже не будет предельно простым. Такой способ решения может считаться частично-поисковым (эвристическим). Ниже обсуждаются способы решения задачи.

Записи в таблице `ProductsMovement` не фиксируют наименование товара, с которым совершается акт движения. Зато при каждом акте движения фиксируется артикул товара. Нужно узнать, каковы артикулы товаров, являющихся различными видами зефира. Нужная информация имеется в таблице `Product` — каждая ее запись содержит наименование товара и соответствующий артикул. Можно выполнить следующий запрос.

```
SELECT * FROM Product  
WHERE product_name LIKE '%Зефир%'
```

Будет получена такая информация. Все виды зефира — это товары с артикулами от 4 до 7 включительно (зефир в шоколаде, зефир ванильный, зефир воздушный и зефир лимонный соответственно). Единицей измерения количества каждого из этих товаров является грамм. Массы одной упаковки каждого вида зефира различны (250, 800, 500 и 1000 граммов соответственно).

Записи таблицы **ProductsMovement** не фиксируют адреса магазинов, в которых совершаются акты движения товаров. Зато при каждом акте движения товара фиксируется уникальный идентификатор соответствующего магазина. Нужно узнать, какими идентификаторами обозначены магазины, расположенные на проспекте Революции. Эта информация может быть найдена в таблице **Shop** — каждая ее запись содержит уникальный идентификатор магазина и его адрес. Запрос может быть следующим.

```
SELECT * FROM Shop  
WHERE address LIKE '%Революции%'
```

Полученная в ответ информация такова. В Октябрьском районе действительно имеется проспект Революции, на котором расположены магазины **M10** (дом номер 1) и **M15** (дом номер 29).

Теперь можно составить запрос, в результате выполнения которого из таблицы **ProductsMovement** отбираются записи, отвечающие поступлению товаров с артикулами от 4 до 7 включительно в магазины **M10** и **M15** в период с 2-го по 10-е августа 2023 года включительно. Из каждой записи выделяется количество упаковок поступившего товара, которое умножается на количество соответствующего товара в граммах на одну упаковку (эти сведения извлекаются из таблицы **Product** по соответствующему значению артикула товара). Указанные количества для перевода в килограммы делятся каждое на 1000 и суммируются. Запрос, выражающий сформулированную выше мысль на языке SQL, приведен ниже.

```
SELECT sum(pm.number_of_packages * p.quantity_per_package / 1000)  
FROM Product p,  
      ProductsMovement pm  
WHERE p.article = pm.article AND  
      pm.article >= 4 AND  
      pm.article <= 7 AND  
      (pm.shop_id = 'M10' OR pm.shop_id = 'M15') AND  
      pm.operation_type = 'Поступление' AND  
      pm.operation_date >= '2023-08-02' AND  
      pm.operation_date <= '2023-08-10'
```

Получается: 2550 килограммов. В ответ записывается 2550. Задача полностью решена. Впрочем, отметим еще вот что. В последний (главный) запрос два предыдущих (вспомогательных) запроса можно включить в качестве подзапросов. Выполняя такой комплексный запрос, СУБД «сама» сначала из таблицы **Product** узнает необходимые артикулы товаров, затем в таблице **Shop** отыщет необходимые уникальные идентификаторы магазинов и уж тогда выполнит действия, приводящие к получению (вычислению) искомого результата (основываясь, главным образом, на сведениях, извлекаемых из таблицы **ProductsMovement**). Формулировка комплексного запроса (который один вместо ранее приведенных трех решает задачу) дана ниже.

```
SELECT sum(pm.number_of_packages * p.quantity_per_package / 1000)
FROM Product p,
     ProductsMovement pm
WHERE p.article = pm.article AND
      pm.article IN (
        SELECT article FROM Product
        WHERE product_name LIKE '%Зефир%'
      ) AND
      pm.shop_id IN (
        SELECT shop_id FROM Shop
        WHERE address LIKE '%Революции%'
      ) AND
      pm.operation_type = 'Поступление' AND
      pm.operation_date >= '2023-08-02' AND
      pm.operation_date <= '2023-08-10'
```

Результат получается в точности тот же: 2550 килограммов. В ответ записывается 2550. На этом обсуждение способов решения задачи можно завершить.

4. Кодирование и декодирование информации. По некоторому каналу связи передаются сообщения, содержащие только десять букв: *A, B, C, D, E, F, S, X, Y* и *Z*. Для кодирования букв используется префиксный неравномерный двоичный код. Соответствующие буквам (всем, кроме буквы *B*) кодовые слова (кодона) приведены ниже.

<i>A</i> : 00	<i>F</i> : 1001
<i>B</i> : ?	<i>S</i> : 1100
<i>C</i> : 010	<i>X</i> : 1010
<i>D</i> : 011	<i>Y</i> : 1101
<i>E</i> : 1011	<i>Z</i> : 111

Укажите кратчайшее кодовое слово (кодон) для буквы *B*, при котором код удовлетворяет условию Фано. Если таких кодонов несколько, укажите тот из них, который является двоичной записью наименьшего неот-

рицательного целого числа. *Примечание:* код называется *префиксным*, если никакой из кодонов, составляющих этот код, не является началом какого-либо другого кодона этого кода. Сформулированное только что условие называется *условием Фано*. Условие Фано задает свойство, определяющее код как префиксный. Префиксность кода (то есть удовлетворение им условия Фано) является *достаточным* условием *разделимости* кода (то есть, по сути, его пригодности к кодированию информации). Впрочем, условие Фано не является *необходимым* условием разделимости. Например, код $\alpha: 0, \omega: 01$ условию Фано не удовлетворяет (не является префиксным), однако он разделим (то есть пригоден к кодированию информации).

Результат выполнения Вами поставленного задания таков: _____ .

Задание 4 является тестовым заданием с кратким ответом. От испытуемого не требуется приведение полного текста решения. Получить верный ответ можно разными способами. Например, испытуемый может работать с компьютером, в операционной системе которого имеется простой текстовый редактор и корректно функционирующий стандартный Веб-браузер. Подключение компьютера к локальной или глобальной компьютерной сети не требуется. Ниже обсуждается вариант решения поставленной в задании 4 задачи с использованием указанных средств.

Произвольное конечное множество будем называть *алфавитом*. Элементы алфавита будем называть его *буквами* или *символами*. Дальше в поле нашего внимания будут находиться два алфавита, неявно упомянутые в условии задачи: алфавит исходных сообщений, предназначенных для последующего кодирования и передачи по каналу связи, $A_{10} = \{A, B, C, D, E, F, S, X, Y, Z\}$, и двоичный алфавит закодированных сообщений, передающихся по каналу связи, $A_2 = \{0, 1\}$. Любую последовательность записанных слева направо букв алфавита будем называть *словом* в этом алфавите. Будем также допускать существование *пустого* слова — слова, не содержащего ни одной буквы. Пустое слово обозначаем буквой Λ (большая греческая буква «лямбда»). Количество букв, составляющих слово, называем *длиной* слова. Длину слова w обозначаем $l(w)$. Длина пустого слова равна нулю: $l(\Lambda) = 0$. Длина непустого слова равна некоторому положительному целому (натуральному) числу. *Произведением* слова w_1 и слова w_2 называем слово w_1w_2 , которое получается путем приписывания слова w_2 справа к слову w_1 . Например, если w_1 и w_2 суть слова в алфавите A_2 , и слово w_1 есть 01, а слово w_2 есть 10, то слово w_1w_2 есть 0110. Иногда (особенно, в компьютерном программировании) слова называют *строками*, а произведение слов называют *конкатенацией* строк. В некоторых языках программирования (например, Паскаль, JavaScript) операцию конкатенирования (слияния) строк обозначают знаком + («плюс»). Ясно, что для любых слов v и w справедливы следующие

щие равенства: $l(vw) = l(wv) = l(v) + l(w)$, $w\Lambda = \Lambda w = w$. Дадим теперь важные для дальнейшего обсуждения определения понятий «голова слова» и «хвост слова».

Головой непустого слова w будем называть однобуквенное слово, единственная буква в составе которого — это первая буква слова w . Будем считать, что у пустого слова тоже имеется голова. Ею является пустое слово. Голову слова w обозначаем $\text{car } w$. Например: $\text{car } 0110 = 0$, $\text{car } 1 = 1$, $\text{car } \Lambda = \Lambda$. *Хвостом* непустого и неоднобуквенного слова w называем слово, получающееся из слова w путем вычеркивания (стирания, удаления) его первой буквы. Будем считать, что у пустого слова и у однобуквенного слова тоже имеется хвост. Им является пустое слово. Хвост слова w обозначаем $\text{cdr } w$. Например: $\text{cdr } 0110 = 110$, $\text{cdr } 1 = \Lambda$, $\text{cdr } \Lambda = \Lambda$. Обозначения car и cdr для (соответственно) голов и хвостов слов выбраны нами в лирическом настроении, связанном с теплыми воспоминаниями о прекрасной романтической и героической эпохе становления кибернетики, когда популярен был легендарный язык функционального «программирования в терминах списков» LISP, использовавший взятые нами слова в качестве имен функций, вычисляющих (соответственно) голову и хвост данных основного в этом языке типа — списков.

Формулировка условия Фано содержит понятие «начало слова». Определим его. Слово b называется *началом* слова w , если слово w может быть представлено в следующем виде: $w = bv$, где v — некоторое слово. У всякого слова есть начало. Оно может быть не одно. Очевидны следующие факты: пустое слово является началом любого слова (ибо любое слово w может рассматриваться как $w = \Lambda w$), у пустого слова есть только одно начало — пустое слово (ибо $\Lambda = \Lambda\Lambda$, но не существует отличного от Λ слова w , для которого $\Lambda = w\Lambda$), всякое слово является своим собственным началом (ведь для любого слова w справедливо равенство $w = w\Lambda$), количество существенно различных начал n -буквенного слова ($n = 0, 1, 2, \dots$) равно $n + 1$.

Убедиться в том, что некоторый код удовлетворяет условию Фано (то есть является префиксным), — это значит для каждого слова-кодона, входящего в состав исследуемого кода, убедиться в том, что он не является началом никакого другого кодона, и что ни один другой кодон не является его началом. Необходим алгоритм, позволяющий для данного слова b и еще одного данного слова w установить, является ли слово b началом слова w . Словесное описание такого (рекурсивного) алгоритма приводится далее. Присвоим этому алгоритму имя Ξ_1 .

- ① Если слово b — пустое, то признать, что слово b является началом слова w . На этом исполнение алгоритма завершить. Если слово b не является пустым, то перейти к исполнению предписаний инструкции ②.

- ② Если слово w — пустое, то признать, что слово b не является началом слова w . На этом исполнение алгоритма завершить. Если слово w не является пустым, то перейти к исполнению предписаний инструкции ③.
- ③ Если $\text{cag } b = \text{cag } w$ и $\text{cdr } b$ есть начало $\text{cdr } w$, то признать, что слово b есть начало слова w . В противном случае признать, что слово b не является началом слова w . На этом исполнение алгоритма завершить.

План отыскания двоичного слова (слова в алфавите A_2), которое, соответствуя условию задачи, могло бы быть кодоном для буквы B , таков. Нужно последовательно перебирать все двоичные слова длины 0 (это одно-единственное пустое слово), слова длины 1 (их два — 0 и 1), слова длины 2 (их четыре — 00, 01, 10 и 11) и так далее. Для каждого объекта перебора (двоичного слова) необходимо проверить, является ли оно началом хотя бы одного из указанных в условии задачи кодонов, и является ли хотя бы один из этих кодонов началом рассматриваемого в данный момент объекта перебора. Если проверка дает положительный результат, то перебор нужно продолжать, переходя к рассмотрению следующего члена последовательности перебираемых объектов. Первый из объектов перебора, для которого указанная выше проверка даст отрицательный результат, должен быть объявлен соответствующим условию задачи кодом для буквы B .

Чтобы соответствие условию задачи было полным, в содержании употребленного выше понятия «перебирать последовательно» нужно подчеркнуть следующий смысл: 1) никакое слово некоторой длины не должно быть подвергнуто рассмотрению раньше какого-либо другого слова меньшей длины; 2) слова одинаковой длины должны перебираться в лексикографическом порядке (символ 0 в алфавите A_2 считается предшествующим символу 1), иными словами, из двух слов одинаковой длины раньше должно быть рассмотрено то, которое является двоичной записью меньшего неотрицательного целого числа. Необходим алгоритм, позволяющий для данного двоичного слова w указать следующее за ним в описанном плане перебора двоичное слово x . Словесное описание такого алгоритма приведено ниже. Этому алгоритму присваиваем имя Ξ_2 .

- ① Если слово w — пустое, то в качестве x указывается слово 0 и исполнение алгоритма завершается. Если слово w не является пустым, то производится переход к исполнению предписаний инструкции ②.
- ② Неотрицательное целое число, двоичной записью которого является слово w , обозначается m . Неотрицательное целое число $n = m + 1$. Двоичное слово b — кратчайшая двоичная запись числа n . Если $l(b) = l(w)$, то $x = b$ и исполнение алгоритма завершается. Если же $l(b) > l(w)$, то $x = 0 \text{cdr } b$ и исполнение алгоритма завершается. Нако-

нец, если $l(b) < l(w)$, то $x = 0^k b$, где $k = l(w) - l(b)$, и исполнение алгоритма завершается. *Примечание:* символом 0^k обозначено k -буквенное слово, каждая буква (каждый символ) которого есть 0.

Ясно, что пустое слово, поскольку оно является началом любого слова, не может быть кодом никакого префиксного (то есть удовлетворяющего условию Фано) кода. Поэтому, исключая рассмотрение пустого слова из предложенного выше плана «перебора и проверок» двоичных слов-«претендентов» на то, чтобы быть удовлетворяющим условию обсуждаемой задачи кодом для буквы B , можем теперь дать словесное описание процедуры поиска нужного нам двоичного слова. Эта процедура содержит «квазибесконечный» цикл. Мы надеемся на то, что цикл во время исполнения окажется вовсе не бесконечным — будет найдено искомое двоичное слово, и цикл прервется. Если это действительно случится, мы получим право называть нашу процедуру *алгоритмом* отыскания решения задачи. Однако, на текущем этапе обсуждения мы не имеем доказательства того, что указанное выше «счастливое событие» непременно произойдет. Поэтому и употребляем осторожное выражение «процедура». Присваиваем ей имя Ξ .

- ① Считать, что двоичное слово $b = A$.
- ② Исполнив алгоритм Ξ_2 , получить двоичное слово x , следующее за словом b . Слову x присвоить имя b .
- ③ Исполнив нужное количество раз алгоритм Ξ_1 , проверить является ли слово b началом какого-нибудь из девяти двоичных кодонов, указанных в условии задачи. Проверить тем же способом, является ли какой-нибудь из указанных в условии задачи девяти кодонов началом слова b . Если результаты всех проверок отрицательны, признать двоичное слово b искомым кодом. Исполнение алгоритма завершить. Если в ходе выполнения проверок будет получен положительный результат — проверки прекратить и перейти к исполнению предписаний инструкции ②.

Исполнение процедуры Ξ может быть осуществлено встроенным в Веб-страницу сценарием на языке JavaScript. Если имеющийся в процедуре «квазибесконечный» цикл окажется не бесконечным, после открытия Веб-страницы с помощью корректно функционирующего стандартного Веб-браузера в клиентской области будет выведено слово, являющееся полностью удовлетворяющим условию задачи двоичным кодом. Код такой Веб-страницы помещен нами в файл `task04.html` и опубликован по адресу (*URL*) <http://docsagmax.ru/donsu/aea/acaec/programs/informatics/task04.html>. Далее приведен полный листинг этого файла.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Task 04</title>
```

```
<script type="text/javascript">
/* Входные данные. */
var initialPrefixCode = [
  new String('00'), // A
  new String('010'), // C
  new String('011'), // D
  new String('1011'), // E
  new String('1001'), // F
  new String('1100'), // S
  new String('1010'), // X
  new String('1101'), // Y
  new String('111') // Z
];
/* Конец входных данных. */
/* Метод car для объектов класса String – слов. Если слово пустое,
возвращает false. Если слово непустое, возвращает его голову.
Например:
var lAmbda = new String('');
var w = new String('1011');
var x = lAmbda.car(); // x = false
var y = w.car(); // y = '1'
*/
var s = new String('');
var p = Object.getPrototypeOf(s);
p.car = function() {
  if (this.length == 0) return false;
  return '' + this.charAt(0);
}
/* Метод cdr для объектов класса String – слов. Если слово пустое
или однобуквенное, возвращает false. Во всех остальных случаях
возвращает хвост слова. Например:
var lAmbda = new String('');
var sLetter = new String('0');
var w = new String('1001');
var x = lAmbda.cdr(); // x = false
var y = sLetter.cdr(); // y = false
var z = w.cdr(); // z = '001'
*/
p.cdr = function() {
  if (this.length <= 1) return false;
  return this.substr(1, this.length-1);
}
/* Метод isBeginOf для объектов класса String – слов. Если данное
слово является началом слова-аргумента метода, возвращает true.
В противном случае возвращает false. Например:
var firstWord = new String('00');
var secondWord = new String('01');
var argWord = new String('010');
var x = firstWord.isBeginOf(argWord); // x = false
var y = secondWord.isBeginOf(argWord); // y = true
*/
p.isBeginOf = function(strObject) {
  if (this.length == 0) return true;
  if (strObject.length == 0) return false;
  if (
    (this.car() == strObject.car()) &&
    (this.cdr().isBeginOf(strObject.cdr())) // Рекурсивный вызов
  ) return true;
  return false;
}
```

```
/* Метод nextBinaryWord для объектов класса String – двоичных слов.
   Возвращает двоичное слово, следующее за данным в том смысле,
   какой демонстрируют приведенные ниже примеры. При одинаковой
   длине следующее за данным слово является двоичной записью
   большего неотрицательного целого числа, чем то неотрицательное
   целое число, двоичной записью которого является данное слово.
   Например:
   var bw = new String(''); // bw = ''      длина = 0
   bw = bw.nextBinaryWord(); // bw = '0'     длина = 1
   bw = bw.nextBinaryWord(); // bw = '1'     1 > 0
   bw = bw.nextBinaryWord(); // bw = '00'    длина = 2
   bw = bw.nextBinaryWord(); // bw = '01'    1 > 0
   bw = bw.nextBinaryWord(); // bw = '10'    2 > 1
   bw = bw.nextBinaryWord(); // bw = '11'    3 > 2
   bw = bw.nextBinaryWord(); // bw = '000'   длина = 3
   bw = bw.nextBinaryWord(); // bw = '001'   1 > 0
   bw = bw.nextBinaryWord(); // bw = '010'   2 > 1
   bw = bw.nextBinaryWord(); // bw = '011'   3 > 2
   bw = bw.nextBinaryWord(); // bw = '100'   4 > 3
   // и т. д.
*/
p.nextBinaryWord = function() {
  if (this.length == 0) return new String('0');
  var x = Number('0b' + this);
  x++;
  var strX = new String(x.toString(2));
  if (strX.length > this.length)
    return new String('0' + strX.cdr());
  if (strX.length < this.length) {
    var retStr = new String(strX);
    var n = this.length - strX.length;
    for (var i = 0; i < n; i++)
      retStr = new String('0' + retStr);
    return retStr;
  }
  return strX;
}
/* Метод preservesThePrefixOfCode для объектов класса String –
   слов. В качестве аргумента получает массив слов-кодонов,
   образующих префиксный код. Если добавление к этому коду
   данного слова в качестве еще одного кодона не нарушает свойство
   префиксности для получаемого «расширенного» кода, возвращает
   true. В противном случае возвращает false. Например:
   var initialCode = [
     new String('0'),
     new String('10')
   ];
   var goodCodon = new String('11');
   var badCodon = new String('01');
   var x = goodCodon.preservesThePrefixOfCode(initialCode); // x = true
   var y = badCodon.preservesThePrefixOfCode(initialCode); // y = false
*/
p.preservesThePrefixOfCode = function(arrCodons) {
  for (var i = 0; i < arrCodons.length; i++)
    if (
      (this.isBeginOf(arrCodons[i])) ||
      (arrCodons[i].isBeginOf(this))
    ) return false;
  return true;
}
```

```
/* Главный программный модуль.  
Последовательно перебираются двоичные слова '0', '1', '00',  
'01', '10', '11', '000', '001' и т. д. Как только в этой после-  
довательности обнаружится член, являющийся двоичным словом, год-  
ным для добавления в качестве еще одного кодона к префиксному  
коду, заданному входными данными, не нарушающим свойство пре-  
фиксности для получаемого «расширенного» кода, перебор заверша-  
ется, и выводится то двоичное слово, которое было обнаружено.  
*/  
function main() {  
    var el = document.getElementById("output");  
    var strPr = '';  
    var cOdon = new String('');  
    while (true) {  
        cOdon = cOdon.nextBinaryWord();  
        if (cOdon.preservesThePrefixOfCode(initialPrefixCode)) {  
            el.innerHTML = cOdon;  
            return;  
        }  
    }  
}  
</script>  
</head>  
<body onLoad="main()">  
    <div id="output">  
    </div>  
</body>  
</html>
```

Интерпретация файла `task04.html` корректно функционирующим стандартным Веб-браузером приводит к получению приведенного ниже сообщения.

1000

Итак, процедура Ξ исполнена. Имеющийся в ней «квазibesконечный» цикл оказался не бесконечным. Нужный кодон нашелся. Математически состоятельно можно утверждать, что для обсуждаемой *конкретной* задачи процедура Ξ является *алгоритмом* отыскания решения. В ответ записывается двоичное слово 1000. Задача полностью решена.

Краткое дополнение. Задача действительно решена и решена математически корректно. Но чувство неудовлетворенности все же остается. Дело вот в чем. Процедура Ξ оказалась алгоритмом решения *единичной* задачи. А будет ли процедура Ξ алгоритмом решения *массовой* задачи, частным случаем которой является решенная только что единичная задача? Ведь решить задачу — значит найти и предъявить все искомые объекты или обоснованно утверждать, что ни один искомый объект не существует. Верно ли, что для всякого набора входных данных (то есть для всякого префиксного двоичного кода) существует искомый объект

(то есть отличное от каждого из кодонов данного кода двоичное слово, добавление которого в качестве еще одного кодона к данному коду приводит к получению нового «расширенного» кода, являющегося тоже префиксным)? Конечно же, это неверно. Например, двоичный код $\eta:0, \xi:1$ является префиксным, но любое непустое двоичное слово содержит один из двух кодонов этого кода в себе в качестве начала. А пустое слово, разумеется, является началом каждого из двух кодонов этого кода. Следовательно, для указанного кода, если его рассматривать как набор входных данных массового обобщения обсуждаемой задачи, искомый объект не существует. Если с указанными входными данными «запустить» процедуру Ξ , она никогда не завершит работу, и ни на каком этапе работы исполнитель не сможет обоснованно принять решение о несуществовании искомого объекта, сообщить об этом и прекратить бесперспективную деятельность. Можно ли усовершенствовать процедуру Ξ так, чтобы она в усовершенствованном виде могла бы быть не только алгоритмом решения исходной единичной задачи, но и алгоритмом решения ее массового обобщения? На этот вопрос (к нашей радости и «во спасение» процедуры Ξ) можно дать положительный ответ.

Двоичный префиксный код C будем называть *расширяемым* (обладающим свойством расширяемости), если существует двоичное слово w , не являющееся кодоном кода C , добавление которого к списку кодонов кода C приводит к получению *расширенного* двоичного кода C^* , являющегося, как и код C , префиксным. Такое двоичное слово w будем называть *подходящим* для расширения кода C или *сохраняющим* (не нарушающим) префиксность кода C при его добавлении к списку кодонов кода C . Двоичный префиксный код, не являющийся расширяемым (не обладающий свойством расширяемости), будем называть *нерасширяемым* (обладающим свойством нерасширяемости). Например, двоичный префиксный код $\delta:0, \varphi:10$ — расширяем (подходящим для его расширения двоичным словом является 11), а упомянутый выше двоичный префиксный код $\eta:0, \xi:1$ — нерасширяем (его нерасширяемость продемонстрирована вслед за его упоминанием выше).

Сформулируем теперь массовое обобщение исходной единичной задачи, алгоритмом решения которого мог бы быть усовершенствованный вариант процедуры Ξ . Пусть дан произвольный двоичный префиксный код C . Нужно указать двоичное слово w , подходящее для расширения кода C , если такое слово существует. Если же двоичное слово w , сохраняющее префиксность кода C при его добавлении к списку кодонов кода C , не существует, нужно сообщить об этом.

Докажем справедливость нескольких вспомогательных утверждений (лемм).

Лемма 1. Для любых (не только двоичных) слов отношение «быть началом» транзитивно: если слово a является началом слова b , а слово b является началом слова c , то слово a является началом слова c .

Доказательство. Пусть слово a есть начало слова b . Значит, существует слово x такое, что $b = ax$. Пусть слово b есть начало слова c . Значит, существует слово y такое, что $c = by$. Но $b = ax$, поэтому $c = axy$. Следовательно, существует слово $z = xy$ такое, что $c = az$, а это как раз и означает, что слово a есть начало слова c . Лемма доказана.

Лемма 2. Никакое одно из двух различных слов равной длины не является началом другого. Это утверждение справедливо для любых (не только двоичных) слов.

Доказательство. Пусть a и b — два различных слова одинаковой длины: $l(a) = l(b) = l$, l — неотрицательное целое число. Предположим, что справедливо утверждение, противное доказываемому, — одно из слов a , b является началом другого. Исключительно для определенности, не ограничивая общности, будем считать, что слово a является началом слова b . Значит, существует слово x такое, что $b = ax$. Тогда можно записать: $l(b) = l(a) + l(x)$; $l = l + l(x)$; $l(x) = l - l = 0$. Поскольку $l(x) = 0$, $x = \Lambda$. Тогда: $b = ax = a\Lambda = a$. Поскольку $b = a$, слова a и b — не различны (но просто a и b — два разных имени для одного и того же слова). Получено противоречие. Следовательно, утверждение, противное доказываемому, ложно. Тогда истинно доказываемое утверждение. Лемма доказана.

Лемма 3. Никакое слово не является началом слова меньшей длины. Это утверждение справедливо для любого (не только двоичного) слова.

Доказательство. Пусть a и b — два слова, и слово a длиннее слова b : $l(a) > l(b)$. Если предположить, что слово a есть начало слова b , то должно будет существовать слово x такое, что $b = ax$. Тогда: $l(b) = l(a) + l(x)$; $l(x) = l(b) - l(a)$; $l(x) < 0$. Однако, длина любого слова (в том числе и слова x) не может быть отрицательной. Следовательно, никакое слово a не может быть началом никакого слова b , если длина слова b меньше длины слова a . Что и требовалось доказать.

Лемма 4. Если слово a является началом слова c , и слово b является началом слова c , то или a является началом b , или b является началом a (имеется в виду неэксклюзивное «или»). Это утверждение справедливо для любых (не только двоичных) слов.

Доказательство. Пусть a , b и c — те слова, о которых идет речь в доказываемом утверждении. Слово c может быть либо пустым, либо непустым. Если слово c — пустое, то слова a и b могут быть только пустыми (непустые a и b имели бы положительную длину, большую равной нулю длины слова c , и поэтому, согласно лемме 3, не могли бы быть началами слова c). Пустое слово a , естественно, есть начало пустого слова b , а пустое слово b , конечно же, есть начало пустого слова a . Таким образом, если слово c — пустое, то утверждение леммы доказано.

Если слово c — непустое, то каждое из слов a , b может быть как пустым, так и непустым. Если какое-либо из слов a , b — пустое (или пусты оба слова), то по крайней мере одно из них является началом другого

(ибо пустое слово является началом любого слова). Так что для непустого s и пустого хотя бы одного из a , b утверждение леммы доказано. Остается рассмотреть нетривиальный случай, когда ни одно из слов a , b , c не является пустым.

Пусть положительное целое число m — длина слова c . Слово c можно представить в виде произведения m однобуквенных слов (однобуквенное слово c_i ($i = 1, \dots, m$) состоит из единственной буквы, являющейся i -й буквой записи слова c) следующим образом: $c = c_1 \dots c_m$. Пусть положительные целые числа k и l — длины слов a и b соответственно. Ясно (лемма 3), что каждое из чисел k , l не превосходит по величине числа m . Каждое из слов a , b можно (аналогично тому, как это сделано чуть раньше для слова c) представить в виде произведения однобуквенных слов: $a = a_1 \dots a_k$, $b = b_1 \dots b_l$ (записи $a_1 \dots a_k$, $b_1 \dots b_l$ и $c_1 \dots c_l$ означают просто a_1, b_1 и c_1 соответственно — на те случаи, в которых слово a , слово b или слово c является однобуквенным). Рассмотрим три возможных ситуации: 1) длины слов a и b совпадают; 2) слово a короче слова b ; 3) слово a длиннее слова b .

1) Считаем, что длины слов a и b совпадают ($k = l = p$). Тогда $a = a_1 \dots a_p$, $b = b_1 \dots b_p$, положительное целое число p не превосходит m . Утверждение «слово a длины p есть начало слова c » равносильно верности каждого из равенств: $a_1 = c_1, \dots, a_p = c_p$. Утверждение «слово b длины p есть начало слова c » равносильно верности каждого из равенств: $b_1 = c_1, \dots, b_p = c_p$. Из указанных двух систем равенств (в силу симметричности и транзитивности отношения равенства) немедленно следует: $a_1 = b_1, \dots, a_p = b_p$. Последнее означает, что слова a и b полностью совпадают (не только по длине, но и по записи — друг от друга неотличимы). Тогда любое из слов a , b можно объявить началом другого. В рассматриваемой ситуации справедливость утверждения леммы доказана.

2) Пусть теперь длина k слова a меньше длины l слова b . Утверждение «слово a длины k есть начало слова c » равносильно верности каждого из равенств: $a_1 = c_1, \dots, a_k = c_k$. Утверждение «слово b длины l (большей, чем k) есть начало слова c » равносильно верности каждого из равенств: $b_1 = c_1, \dots, b_k = c_k, \dots, b_l = c_l$. Из k равенств первой системы и первых k равенств второй системы равенств (в силу симметричности и транзитивности отношения равенства) следует: $a_1 = b_1, \dots, a_k = b_k$. Последняя система равенств равносильна утверждению «слово a длины k есть начало слова b ». В рассматриваемой ситуации справедливость утверждения леммы доказана.

3) Пусть наконец длина k слова a больше длины l слова b . Утверждение «слово a длины k (большей, чем l) есть начало слова c » равносильно верности каждого из равенств: $a_1 = c_1, \dots, a_l = c_l, \dots, a_k = c_k$. Утверждение «слово b длины l есть начало слова c » равносильно верности каждого из равенств: $b_1 = c_1, \dots, b_l = c_l$. Из первых l равенств первой системы и l ра-

венств второй системы равенств (в силу симметричности и транзитивности отношения равенства) следует: $b_1 = a_1, \dots, b_l = a_l$. Последняя система равенств равносильна утверждению «слово b длины l есть начало слова a ». В рассматриваемой ситуации справедливость утверждения леммы доказана.

Справедливость утверждения леммы доказана полностью.

Пусть алфавит A_q , из букв которого состоят рассматриваемые в некотором контексте слова (не обязательно двоичные), содержит q ($q > 1$) букв. Полный список слов в алфавите A_q длины n (n — неотрицательное целое число), содержащий по одному каждое слово указанной длины (всего q^n слов), будем обозначать L_n . Пусть C — префиксный (не обязательно двоичный) код, все кодоны которого являются словами в алфавите A_q . Пусть неотрицательное целое число N — такое число, что среди кодонов кода C имеется хотя бы один кодон, являющийся словом длины N , но нет ни одного кодона, являющегося словом длины, большей, чем N . Будем называть число N *максимальной* (предельной, наибольшей) *длиной* кодонов кода C . Полный список слов в алфавите A_q длины, равной максимальной длине кодонов кода C , обозначаем L_N . Например, для двоичного префиксного кода $\delta:0, \varphi:10, \theta:11$ максимальная длина кодонов $N=2$, а список $L_N=(00, 01, 10, 11)$. Сформулируем и докажем как теорему *критерий нерасширяемости* префиксного (не обязательно двоичного) кода C .

Теорема (критерий нерасширяемости префиксного кода). Префиксный код C , все кодоны которого суть слова в алфавите A_q , содержащем $q > 1$ букв, максимальная длина кодонов которого равна N , является нерасширяемым тогда и только тогда, когда полный список L_N слов в алфавите A_q длины, равной N , не содержит ни одного слова, подходящего для расширения кода C (то есть сохраняющего префиксность кода C при его добавлении к списку кодонов кода C).

Доказательство необходимости (часть «только тогда» утверждения теоремы). Нужно доказать, что из нерасширяемости префиксного кода C следует то, что список L_N не содержит ни одного слова, подходящего для расширения кода C . Доказательство является тривиальным. Действительно, если префиксный код C является нерасширяемым, то вообще не существует слова, подходящего для расширения кода C . Разумеется, тогда нет такого слова и в списке L_N . Необходимость доказана.

Доказательство достаточности (часть «тогда» утверждения теоремы). Требуется доказать, что префиксный код C будет нерасширяемым, если соответствующий предельной длине N кодонов кода C полный список слов L_N не содержит ни одного слова, подходящего для расширения кода C . Проведем доказательство от противного. Предположим, что список L_N не содержит ни одного слова, подходящего для расширения кода C , но код C все-таки является расширяемым, то есть существует

слово w , подходящее для расширения кода C . Согласившись с предположением о существовании указанного слова, рассмотрим три ситуации, полностью исчерпывающие в совокупности все возможности: 1) слово w имеет длину, равную N ; 2) слово w имеет длину, большую, чем N ; 3) слово w имеет длину, меньшую, чем N .

1) Если подходящее для расширения кода C слово w имеет длину, равную N , оно является членом списка L_N . Это противоречит принятому нами предположению, что список L_N не содержит ни одного слова, подходящего для расширения кода C . Значит, слово w не может иметь длину, равную N .

2) Пусть подходящее для расширения кода C слово w имеет длину l , бол

