Российская академия наук
Сибирское отделение
Институт систем информатики
им. А. П. Ершова

Т. А. Андреева

СТРУКТУРНЫЙ АНАЛИЗ И СИСТЕМАТИЗАЦИЯ УСЛОВИЙ ОЛИМПИАДНЫХ ЗАДАЧ ПО ПРОГРАММИРОВАНИЮ

Препринт
149

Новосибирск 2008

Олимпиадные задачи по программированию представляют актуальный и востребованный профиль для изучения и автоматизации процессов разветвленной каталогизации объектов, составленных с использованием естественного языка.

В данной работе проводится исследование возможности массирования и систематизации текстов задач и тестов, линеаризации тем задач, автоматизации подготовки задачных наборов и частичной автоматизации подготовки тестирующих наборов, – с целью дальнейшей автоматизации процесса составления задачных и тестовых наборов.

Siberian Division of the Russian Academy of Sciences
A. P. Ershov Institute of Informatics Systems

Tatiana Andreyeva

STRUCTURE ANALYSIS AND SYSTEMATIZATION OF THE PROGRAMMING CONTEST PROBLEMS

Preprint
149

Novosibirsk 2008

The programming contest problems give an actual and relevant profile for studying and automating the processes of cataloging for the objects composed in the natural language.

This paper represents analysis of the structure and the massiveness of the contest problems and tests, of the linearization of the problem topics, of the automation of preparation of the problem sets, and of the partial automation of preparation of the test sets; with the aim at the automation of the composing processes for the problem and test sets.

ВВЕДЕНИЕ

История олимпиад по информатике и программированию насчитывает уже не один десяток лет. Количество придуманных и решенных задач исчисляется уже не тысячами, а десятками тысяч. Ориентироваться в этом огромном объеме становится все труднее, причем трудность эта растет экспоненциально.

Между тем, до сих пор не существует достаточно хорошей автоматической системы, позволяющей сортировать большое количество задач по программированию.

Автор этой работы поставил себе целью разработать классификационную решетку для олимпиадных задач по программированию, на основании которой можно будет создать систему, облегчающую сортировку и поиск задач.

Такая система нужна

- авторам новых олимпиадных задач во избежание повторов и нежелательных споров о первенстве и авторских правах;
- сотрудникам комитетов по подготовке и проведению олимпиад для составления непересекающихся задачных наборов;
- преподавателям и тренерам, ведущим подготовку студентов для наиболее полного охвата встречающихся на олимпиадах задач.

Автор имеет многолетний опыт участия в подготовке задач для школьных и студенческих олимпиад различных уровней (от районной и городской школьной до Всесибирской[1] студенческой), и это знание проблемы изнутри позволяет ему судить о многих деталях, неочевидных стороннему или менее опытному человеку.

Настоящая работа организована следующим образом.

В разделе 1 («Структура текста задачи») приведен анализ структуры текстов задач.

В разделе 2 («Классификации условий») приведены различные классификации текстов задач.

[1] Открытая Всесибирская олимпиада им. И. В. Поттосина, проводится в Новосибирском государственном университете с 2000 года, имеет статус Всероссийской олимпиады, школьную и студенческую секции. http://olimpic.nsu.ru/index.shtml

В разделе 3 («Серийные задачи») исследуются взаимосвязи между задачами.

В приложении приведены многочисленные примеры задач, иллюстрирующие все обсуждаемые в работе моменты.

1. СТРУКТУРА ТЕКСТА ЗАДАЧИ

Анализ большого количества олимпиадных задач по программированию позволил выявить закономерности, которые отражены в этом разделе.

1.1. Структурные части условия

В тексте задачи по программированию можно с очевидностью выделить следующие части.

Таблица 1

Структурные части условий задач

	Название	Содержание	Обозначение
1.	***Вводная информация***	описание предметной области, к которой относится задача	**ВИ (Лг)**
2.	***Необходимые определения***	соглашения и терминология, используемые в тексте задачи	**НО**
3.	***Описание проблемы***	более или менее формализованная постановка собственно задачи, включая ограничения	**ОП**
4.	***Задание***	требования, при выполнении которых задача будет считаться решенной	**Зд**
5.	***Форматы входных и выходных данных***	пункт, специфичный для задач по программированию	**ФД**
6.	***Пример***	образец правильных входных данных и соответствующего им верного решения	**Пр**

В примерах задач, приведенных в Приложении, тексты условий снабжены пометками, обозначающими название выделяемой структурной части, а также ее вариант – в соответствии с описанными ниже классификациями.

1.2. Степень формализации

Степень формализации любой из структурных частей, перечисленных в предыдущем пункте, может быть различной. Удалось выделить четыре уровня формализации.

0. *Рассказ*
1. *Объяснение*
2. *Описание*
3. *Формальное задание*

1.2.1. Формализуемость структурных частей

Различные части условия подвержены формализации в различной степени. В текстах задач, приведенных в Приложении, для каждой структурной части через дефис указана степень ее формализации (например, **ВИ-1** или **НО-2**).

- ***Вводная информация*** (часть **ВИ**) является наиболее вариативной: очень часто встречаются все четыре степени формализации. Более подробно формализация первой части обсуждается в п. 2.4 «Классификации по Вводной информации».
- ***Необходимые определения***, ***Описание проблемы*** и ***Задание*** (части **НО**, **ОП** и **Зд**) редко встречаются в форме *Рассказа*, однако любой другой уровень формализации может быть использован. Причина этого в том, что назначение частей **НО** и **ОП** – в том, чтобы уточнить часть **ВИ**. Таким образом, уровень формализации частей **НО** и **ОП** всегда выше (или, по крайней мере, не ниже), чем уровень формализации части **ВИ** (если она присутствует в тексте задачи).
- ***Форматы входных и выходных данных*** (часть **ФД**) не встречаются в виде *Рассказа*, что вполне объяснимо, так как от верного понимания этой части в большой степени зависит правильность решения задачи.
- ***Пример*** (часть **Пр**) часто присутствует в условии задачи более одного раза, причем на разных уровнях формализации (см., например, задачу «ACM» в Приложении, П.1.4). Примеры, включенные в текст частей

ВИ, **НО**, **ОП** и **Зд**, чаще всего являются *Описанием*. В то же время, раздел «Пример(ы)», завершающий практически любую задачу по программированию, формализован наиболее сильно, так как он представляет собой образец входного и выходного файлов, отступление от которого может привести к неправильной оценке решения (особенно в случае автоматического тестирования).

1.2.2. Статистика

Для задач, приведенных в приложении, количество структурных частей, соответствующих различным степеням формализации, показано в табл. 2.

Таблица 2

Статистика формализации структурных частей условий задач

Структурные части	**Степени формализации**			
	(0) ***Рассказ***	**(1)** ***Объяснение***	**(2)** ***Описание***	**(3)** ***Формальное задание***
ВИ	2	2	1	1
НО	1	1	1	3
ОП	2	7	2	1
Зд	1	4	4	4
ФД	–	2	6	9

1.2.3. Формализация условий

Для того чтобы при составлении задачных наборов и наполнении базы данных олимпиадных задач отнести задачу к какой-либо теме (см. п. 2.3, «Классификация по теме и сложности»), необходимо предварительно *формализовать* ее условие. Иными словами, необходимо довести степень формализации частей **НО**, **ОП** и **Зд** до максимальной (третьей) и избавиться от части **ВИ.**

После формализации получается задача, *эквивалентная* исходной (см. п. 2.4.3, «Эквивалентные задачи»). При занесении эквивалентных задач в базу данных и отнесении их к той или иной теме (см. п. 2.3, «Классификация по теме и сложности») нужно опираться на наиболее формализованный вариант условия (*базисную* задачу).

Примеры эквивалентных задач, имеющих разные уровни формализации, приведены в Приложении (см. П.5, «Эквивалентные задачи»).

1.3. Организация условия

В процессе анализа условий олимпиадных задач были замечены следующие закономерности, касающиеся последовательности структурных частей, и исключения из них:

- порядок структурных частей может быть произвольным;

  Исключения
  - Если в задаче присутствует ***Вводная информация*** (часть **ВИ**), то она обычно начинает текст условия.
  - Практически всегда условие задачи завершается ***Форматами входных и выходных данных*** и ***Примером*** (части **ФД** и **Пр)**.
- любая структурная часть может оказаться разорванной другой частью или частями;
- любая структурная часть может присутствовать в тексте условия более одного раза, причём с разными степенями формализации;
- любая из шести перечисленных частей может отсутствовать. Все эти случаи подробно рассмотрены в п. 2.1 «Классификация по неполноте условия».

2. КЛАССИФИКАЦИИ УСЛОВИЙ

2.1. Классификация по неполноте условия

Отсутствие той или иной структурной части (см. п. 1.1, «Структурные части условия») в условии задачи совсем не обязательно сделает задачу непригодной к выставлению в качестве олимпиадной.

2.1.1. Отсутствие Вводной информации

Если в условии задачи полностью или почти полностью отсутствует ***Вводная информация*** (часть **ВИ**), то условие становится «сухим», чисто математическим. Работа с такой задачей уже не требует вычленения собственно проблемы: формализация условия уже проведена.

При организации олимпиад по программированию давно уже стало хорошим тоном предлагать задачи с ***легендами*** (см. п. 2.4.2, «Легенды»), однако отсутствие этой части нельзя считать недостатком формулировки за-

дачи. Примером является задача «Теорема Вильсона» (см. Приложение, П.2.1).

2.1.2. Отсутствие Определений

Если автор задачи уверен, что она опирается только на базовые, общепринятые или интуитивные понятия, то в ее условии будут отсутствовать ***Необходимые определения*** (часть **НО**).

Отсутствие этой части само по себе не является изъяном – в том случае, если предположение автора верно и предметная область задачи вполне ясна. В противном же случае могут возникнуть сложно обнаруживаемые ошибки, всплывающие только постфактум, когда из предоставленных решений становится ясно, в каком месте получилось разночтение. Иными словами, условие задачи может стать *недоопределенным* (см. п. 2.2, «Классификация по корректности условия»).

Примером условия, не пострадавшего от отсутствия ***Определений***, может служить задача «Таращенко – чемпион!» или задача «ACM» (см. Приложение, П.1.3 и П.1.4).

2.1.3. Отсутствие Описания проблемы

Описание проблемы (часть **ОП**) может отсутствовать в условии задачи в одном из следующих случаев:

1. ***Вводная информация*** максимально формализована. В этом случае дополнительное ***Описание проблемы*** становится ненужным.
2. ***Описание проблемы*** неявно присутствует в ***Задании***. Это происходит, например, в задачах «Теорема Вильсона, «Счастливый двоечник», «Песнь» и «Подсчет компонент связности» (см. Приложение, П.2.1, П.2.3, П.3.1 и П.5.2).

Замечание. Прямо противоположная ситуация возникает, когда задача состоит из одного только ***Описания проблемы*** (см., например, задачу «Библиотека» в Приложении, П.5.1). Такие задачи являются *задачами на моделирование*.

2.1.4. Отсутствие Задания

Олимпиадные задачи, в которых нет явно выписанного ***Задания*** (часть **Зд**) встречаются довольно редко и, в основном, на олимпиадах нижних уровней. В большинстве таких случаев задание неявно присутствует в частях ***Описание проблемы*** и ***Форматы данных*** (**ОП** и **ФД**). Примером мо-

жет служить задача «Дополнение» (см. Приложение, П.2.2): задание легко восстановить из вопроса.

В то же время, задачу «Библиотека» (см. Приложение, П.5.1), состоящую ровно из одной части – ***Описания проблемы*** – нельзя отнести к полноценным задачам: это, скорее, пока только описание предметной области будущей задачи. Задание вполне возможно идентифицировать и в этом случае, однако отсутствие части ***Форматы данных*** делает невозможной автоматическую проверку этой задачи.

2.1.5. Отсутствие Формата данных

Если нет четко заданных требований к ***Формату входных и выходных данных*** (часть **ФД**), то задача подразумевает разработку наиболее удобного интерфейса и наиболее экономичного внутреннего и внешнего представления данных.

Примером может служить задача «Библиотека» (см. Приложение, П.5.1).

Задачи такого типа невозможно проверять автоматически. Следовательно, их можно предлагать только на олимпиадах самого низкого уровня (например, на школьных районных, где не предполагается умения участников обращаться с файлами).

2.2. Классификация по корректности условия

В тексте любой задачи наиболее важными структурными частями являются ***Описание проблемы*** (**ОП**), ***Задание*** (**Зд**) и ***Форматы данных*** (**ФД**). Ошибки или неясности в этих частях (особенно в ***Задании***) становятся источником в лучшем случае потока вопросов, адресованных составителю; в худшем – неправильной трактовки условия задачи в целом. Если же ошибки или неточности присутствуют в частях ***Вводная информация*** (**ВИ**) и ***Необходимые определения*** (**НО**), то на основании частей **ОП**, **Зд** и **ФД** можно восстановить исходный смысл.

В соответствии с наличием ошибок или неточностей в частях **ОП**, **Зд** и **ФД**, можно выделить три варианта некорректных условий:

1. **Противоречивое условие**

   В тексте задачи допущена серьезная ошибка. Решение такой задачи невозможно, так как область ее решений – пустое множество.

2. **Недоопределенное условие**

   В тексте задачи пропущено важное условие или ограничение, в результате чего становится возможным (или даже необходимым) свободный выбор. Решения таких задач в большинстве случаев не могут пройти тестирование по причине несовпадения допустимых множеств входных данных.

   Пример В задаче «Полиглоты» (см. Приложение, П.4.2) не определены длины строк.

3. **Неоднозначное условие**

   Условие, допускающее неоднозначное толкование, – нечетко сформулированное, допускающее двоякую (еще хуже – многозначную) трактовку. При автоматическом тестировании решение такой задачи дает особенно много псевдо-ошибок.

   Пример В задаче «Отрезки» (см. Приложение, П.4.1) неоднозначность условия проявится в двух случаях: во-первых, если один из отрезков является точкой и эта точка содержится в отрезке, во-вторых, если отрезки расположены на одной прямой, но имеют только одну общую точку. В обоих случаях из текста ***Задания*** и ***Формата данных*** неясно, какой вариант будет правильным: пересечение или наложение отрезков.

2.3. Классификация по теме и сложности

2.3.1. Тезаурусы

Разработать *полную и универсальную* классификацию тем задач представляется почти невозможным, поэтому здесь мы ограничимся лишь указанием на то, что (в первом приближении) тему задачи отражают части условия ***Описание проблемы*** и ***Задание***.

Организаторы каждой олимпиады составляют свой список используемых тем (*Тезаурус*) – в зависимости от уровня олимпиады.

В такой *Тезаурус* могут входить, например,

    - Геометрия
- Графы
- Динамическое программирование
- Целые числа
    - Двоичная арифметика
    - Простые числа
    - Теория чисел
    - т.д.
- Техника программирования
    - Использование машинной арифметики
    - Экономичное представление данных
    - Применение улучшенных сортировок
    - т.д.
- Обработка текстов
    - т.д.

Очевидно, что список этот не обязан быть линейным. Напротив, между его элементами наблюдаются пересечения и взаимосвязи. Одна и та же задача может одновременно относиться к нескольким темам (например, к темам «Простые числа» и «Длинная арифметика», или к темам «Динамическое программирование» и «Экономичное представление массивов»).

База данных олимпиадных задач обязана отражать такую множественность рубрикации условий и решений.

2.3.2. Сложность

Учет сложности решения задачи расширяет классификацию по темам. Например, раздел «Техника программирования» добавляется в *Тезаурус* именно по этой причине.

Зачастую одно только расширение множества допустимых входных данных может кардинально изменить тему (одну из тем), к которой относится задача. Например, необходимость обрабатывать целые числа, превосходящие $2^{32}–1$, может превратить задачу из «задачи на целые числа» в «задачу на длинную арифметику». Разумеется, сложность решения в таком случае возрастает.

Близкие по теме, но различающиеся сложностью решения задачи образуют *серию*. Подробнее серии задач разбираются в разд. 3, «Серийные задачи».

2.3.3. Отнесение задачи к теме

Для того чтобы отнести задачу к какой-либо теме (включая учет сложности ее решения), необходимо

1) максимально *формализовать* ее условие, по возможности избавившись от части **ВИ** и от «воды» в частях **НО**, **ОП** и **Зд**;
2) определить способ решения этой задачи.

Важно помнить, что на основании одного только текста условия задачи не всегда возможно однозначно определить тему, к которой относится задача. Например, задача «Теорема Вильсона» (см. Приложение, П.2.1) является задачей не на длинную арифметику (вычисление факториалов больших чисел), как можно было бы заключить из условия, а на определение простоты натуральных чисел.

2.4. Классификации по Вводной информации

2.4.1. Степень формализации Вводной информации

Напомним, что в п. 1.2 («Степень формализации») были приведены четыре уровня формализации структурных частей условий задач.

0. *Рассказ*
1. *Объяснение*
2. *Описание*
3. *Формальное задание*

Наибольшей вариативностью в смысле формализации обладает часть ***Вводная информация*** (**ВИ**), поэтому мы рассмотрим ее более подробно, чем остальные структурные части.

2.4.2. Легенды

Определение. ***Легендой*** (**Лг**) будем называть ***Вводную информацию*** (**ВИ**) наименьшего уровня формализации (*Рассказ* или *Объяснение*) с некоторой литературной обработкой.

Примеры ***Легенд*** можно найти в задачах «Дарвинизм» и «Таращенко – чемпион!» (см. Приложение, П.1.1 и П.1.3).

Пример сильно формализованной ***Вводной информации*** дает задача «Финский язык» (см. Приложение, П.1.2).

Многие составители считают несолидными задачи с многословным литературным введением. Однако следует отметить, что задача, условие которой написано красочно, с оригинальной вводящей историей, а иногда и с юмором, гораздо приятнее для чтения и понимания. Нет сомнения, что в условиях олимпиадного стресса удовольствие от прочтения остроумной и неожиданной формулировки снимает нервное напряжение и улучшает настроение участникам.

Например, задачи V Всесибирской олимпиады (2004 г.) были снабжены великолепными легендами (автор легенд – Христенко Олег Богданович), связавшими задачи каждого тура воедино. Большое удовольствие участникам доставило чтение условий, безупречных в отношении стиля и грамотности. Для задач очного студенческого тура была выбрана тема «*Понедельник начинается в субботу*», а школьного командного тура – «*Летопись первая из эпопеи «Властелин колец». Хранители*».

2.4.3. Эквивалентные задачи

Для того чтобы отнести задачу к какой-либо теме (включая учет сложности ее решения), необходимо максимально *формализовать* ее условие. Возможно, что в результате такой формализации несколько задач превратятся в одну и ту же – *базисную* задачу.

При занесении задач в базу данных нужно опираться на базисную задачу, на основании которой производится отнесение задачи к той или иной теме Тезауруса (см. п. 2.3, «Классификация по теме и сложности»), – однако необходимо также учитывать и все ***Легенды***.

Обратный процесс «наращивания» различных ***Легенд*** (см. п. 2.4.2, «Легенды») на одну и ту же базисную задачу позволяет получать разные по виду, но одинаковые по сути задачи, что очень полезно для составителей олимпиадных задач, для кружковой работы, и т.п.

В Приложении (см. п. П.5, «Эквивалентные задачи») приведены примеры пар эквивалентных задач, имеющих разные уровни формализации.

2.4.4. Задачи на понимание условия

Отдельного упоминания заслуживает особый вид олимпиадных задач – так называемые *задачи на понимание условия*. Отличительными особенностями таких задач могут быть

- Слишком объемные ***Вводная информация*** и/или ***Описание проблемы***, содержащие больше сведений, чем требуется для решения задачи

Например, в задаче «ACM» (см. Приложение, П.1.4) подробное описание начисления штрафного времени совершенно не нужно для решения, ведь в четвертой части сказано, что все задачи решаются с первой попытки, следовательно, штрафного времени возникнуть просто не может.

- Намеренно усложненные части ***Необходимые определения*** и ***Описание проблемы*** (а иногда и ***Задание***): сквозь обилие цепляющихся друг за друга определений очень трудно «продраться»

  В качестве примеров можно привести задачу «Песнь», предложенную на Российской школьной олимпиаде по программированию в 1999 году, или задачу «Фортификация» (см. Приложение, П.3.1 и П.3.2).

2.5. Классификация по количеству входных данных

2.5.1. Задачи без входных данных

Если у задачи нет входных данных, то она не может претендовать на то, чтобы называться задачей по программированию, поскольку любой алгоритм (а в написании именно алгоритма и состоит решение задач по программированию), по определению, должен обладать свойством *массовости,* то есть решать не одну конкретную задачу, а целый класс аналогичных задач.

Если вариативность входных данных отсутствует, то такую задачу можно отнести к теоретическим и решать ее любым доступным способом: хоть на бумаге, хоть в уме. Можно написать программу, состоящую ровно из одной строки, которая выдает найденный ответ. Такой «алгоритм» будет, конечно, обладать еще одним базовым свойством любого алгоритма – *результативностью*, но не свойством *массовости.*

С точки зрения автоматического тестирования, это не доставит никаких проблем: программа работает и выдает результат. Однако смысла в постановке таких задач нет. Простейшая модификация требований всегда может сделать входные данные вариативными даже без усложнения задачи.

Например, задача «Счастливый двоечник» в том виде, в каком она приведена в Приложении (см. П.2.3), не является задачей по программированию. Однако достаточно изменить жестко заданные границы рассмотрения

[0..1024] на [0..2^N], где N – целое число от 1 до 10, и решить задачу без написания программы станет невозможно.

2.5.2. Размерность пространства входных данных

С точки зрения размерности пространства входных данных, задачи можно разделить на три группы.

(К) Количество считываемых программой данных одинаково для всех тестов (размерность пространства входных данных является константой). Это могут быть любые тесты, длина которых задана изначально в тексте самой задачи. Для удобства понимания такие данные могут располагаться на разных строках, однако составленный из них вектор будет иметь одну и ту же длину для всех тестов данной задачи.

Пример достаточно сложного, но все-таки конечно-постоянного входного вектора можно найти в задаче «Дарвинизм» (см. Приложение, П.1.1). Более простой пример дает задача «Отрезки» (см. Приложение, П.4.1).

(С) Размер пространства входных данных может быть различным для разных тестов, но конкретное количество входных элементов в текущем тесте (длина вектора входных данных) всегда в нем же самом и указывается.

Примером могут служить входные данные для задачи «Дополнение» (см. Приложение, П.2.2).

(Н) Размерность пространства входных данных явно не определена, длина вектора входных данных заранее неизвестна. Более того, ни один тест не содержит явного указания на собственную длину.

Примером могут служить входные данные для задачи «Таращенко – чемпион!» (см. Приложение, П.1.3).

В чистом виде типы **(С)** и **(Н)** встречаются довольно редко. Обычно эти они комбинируются с типом **(К)**. Возможна также их комбинация друг с другом. Вот примеры различных комбинаций:

ФД (КС) «Финский язык» (см. Приложение, П.1.2)
ФД (КН) «АСМ» (см. Приложение, П.1.4)
ФД (СН) «Полиглоты» (см. Приложение, П.4.2)

2.6. Классификация задач по количеству решений

2.6.1. Тестовые наборы

Любую *задачу* можно представить тройкой **<D, C, R>,** где **D** – это набор известных (исходных) данных; **C –** набор условий, выполнение которых необходимо для построения правильного решения; **R –** набор неизвестных величин, определить значения которых необходимо в результате решения задачи.

Способ или *метод решения задачи* – это функционал, который по заданным входным данным **D** и условиям **C** должен построить результирующий набор **R:**

$$\mathbf{S}: \mathbf{D} \times \mathbf{C} \rightarrow \mathbf{R}$$

Алгоритм решения – это более или менее детализированная последовательность взаимосвязанных и взаимозависимых компонент, в совокупности определяющих метод решения.

В случае задач по программированию, **D –** это входные данные, **C** – налагаемые ограничения, **R –** выходные данные, а **S** – собственно программа. Обычно **D** и **C** являются заданными в тексте задачи, и требуется, построив **S**, с его помощью найти **R**.

Такие задачи поддаются автоматическому тестированию на *тестовом наборе*, который представляет собой не что иное, как выборку некоторых точек входной области и соответствующих им точек выходной области. По сути, это сечения функционала: он может быть любым, но обязательно проходящим через эти заданные точки. Аналогичным образом по нескольким дискретным точкам с помощью сплайнов возможно восстановить непрерывную функцию *f*, такую что $f(x) = 0$ для всех заданных $x = (x_1,...x_n)$.

2.6.2. Размерность пространства решений

Мощность множества решений **R** любой задачи может принимать различные значения, которые вполне естественно разделить на пять групп.

$|\mathbf{R}| = 0$ В этом случае задача вообще не имеет решений (например, «*Что лучше: арбуз или балет?»*). Такая задача по определению не корректна, поскольку либо ее исходные данные несовместны, либо невозможно построить метод ее решения, и потому из рассмотрения она исключается. (Заметим, что задача *«Как можно разделить*

три одинаковых яблока на двоих поровну, не разрезая?» имеет единственное решение*: «Такого способа нет».)*

$|\mathbf{R}| = 1$ На задачу можно дать только один правильный ответ: решение существует и единственно.

В этом случае не важно, каким способом было найдено решение, ответ просто проверяется на полное совпадение с эталоном.

$1 < |\mathbf{R}| < \aleph_0$ У задачи существует несколько (конечное число) правильных решений, их множество дискретно.

Такой вариант не обязательно обусловлен недоопределенностью условия. Например, задача нахождения минимального пути вполне может иметь несколько равноправных решений (здесь все зависит от составленных тестов).

Это самый сложный для проверки вариант: необходимо иметь список (или генератор) всех возможных правильных решений, и проверять предоставленный ответ на совпадение с каждым из эталонных решений.

$|\mathbf{R}| = \aleph_0$ (множество решений бесконечно, но не более чем счетно)

Этот случай вплотную смыкается с предыдущим, поскольку понятно, что бесконечные множества решений не доступны для автоматической проверки в силу конечности машинных ресурсов. Поэтому компьютерная «бесконечность» обычно начинается с какого-нибудь достаточно большого числа (например, **MaxLongint**). Верхняя граница может задаваться более круглым числом (например, 1000000), либо вообще быть переменной. Такие задачи формулируются по типу «*Найдите все..., не превосходящие некоторого заданного **N**...*».

$|\mathbf{R}| = \aleph_1$ (примем для краткости, что континуум-гипотеза верна, то есть множество решений имеет континуальную мощность)

Правильные ответы в этом случае укладываются в некоторый интервал (или объединение интервалов), верным считается любое решение из этого интервала (объединения интервалов).

Для интервальной бесконечности существуют ограничения, обусловленные «машинным нулем». Поэтому при автоматической проверке решений этот случай также сводится к проверке конечного числа условий (а именно, таких условий всего два): решение проверяется на принадлежность к заданному интервалу, то есть просто сравнивается с его верхней и нижней границами.

2.6.3. Количество решений и Автоматическая проверка

Из сказанного в предыдущем пункте вытекает, что, с точки зрения автоматической проверки решений, существует лишь два типа задач: *эталонные* и *генераторные*.

- Для проверки решения *эталонной* задачи достаточно сравнить полученный ответ с одним эталоном.

  При проверке интервальных решений (**R**) также достаточно всего одного сравнения: вместо проверки выполнения двух условий

  Lower ≤ R ≤ Upper

  можно ограничиться одной проверкой

  abs(2*R – (Lower + Upper)) ≤ Upper – Lower

- Для проверки решения *генераторной* задачи нужно написать генератор всех правильных решений. Иными словами, если задача сформулирована по типу *«Если решений несколько, выдайте любое из них»*, то для ее проверки нужно решить более широкую задачу *«Если решений несколько, выдайте их все».*

  Следует заметить, что после того, как все правильные решения сгенерированы, проверка сводится к сравнению полученного решения с несколькими эталонами.

Результат – программа

В программировании, как и во всех остальных науках, способ решения любой задачи имеет самостоятельную ценность, поскольку, будучи однажды построенным, может быть применен к целому ряду задач того же типа. Поэтому иногда самоцелью задачи является нахождение именно функционала-программы, отвечающего каким-нибудь жестким условиям. В этом случае заданы **D, C** и **R**, и требуется построить **S**.

Пример

Сигнум

Для произвольного действительного числа x математическая функция *sgn(x)* («знак x») определяется следующим образом:

$$sgn(x) = \begin{cases} 1, \text{ если } x > 0 \\ 0, \text{ если } x = 0 \\ -1, \text{ если } x < 0 \end{cases}$$

Напишите программу, реализующую эту функцию, используя только один оператор ***case*** языка Pascal (или аналогичный оператор ***switch*** языка C).

Здесь нужно построить функционал **S: $\mathfrak{R}$×C → {–1, 0, 1},** где **C** – ограничение, состоящее в обязательном использовании оператора, способного работать не с $\mathfrak{R}$ (множеством всех действительных чисел), а только с **Z** (множеством целых чисел).

В этом случае основной трудностью является нахождение сужения множества всех функционалов, решающих задачу **D → R**, до множества функционалов, решающих задачу **D × C → R**.

Понятно, что такая задача не может быть протестирована автоматически, она подразумевает только текстуальную проверку.

3. СЕРИЙНЫЕ ЗАДАЧИ

Определение *Серией* назовем несколько задач, относящиеся к одной и той же теме (или очень близким темам) Тезауруса, но имеющие разные сложности и, возможно, различные способы решения.

Любая задача, являющаяся частью какой-либо серии, вполне может рассматриваться как самостоятельная и самодостаточная. Однако нам интересны именно те связи, которые возникают между условиями и решениями серийных задач.

3.1. Легенды: серии и не серии

Возможно (хотя и совершенно не обязательно), серийные задачи будут обладать очень похожими ***Легендами*** или вовсе одной и той же ***Легендой***. Назовем серии таких задач *явными*. Примером явной серии может служить серия задач «Поход» (см. Приложение, П.6.1).

Если же ***Легенды*** все-таки разнятся, будем говорить, что это серия *неявная*. Примером неявной серии являются задачи «Комментарии», «Синонимия» и «Составные операторы» (см. Приложение, П.6.4).

В то же время, наличие у нескольких задач похожих ***Легенд*** не может служить достаточным признаком серии. После формализации эти задачи вполне могут превратиться в совершенно разные, никак между собой не связанные задачи.

Поэтому далее в этом разделе мы будем рассматривать только *базисные* (*формализованные) задачи*.

3.2. Серии задач

3.2.1. Линейные серии

Определение *Линейной* назовем серию, все задачи которой с очевидностью можно «выстроить» в одну линию в соответствии с возрастанием сложности их решений (по аналогии с математическим линейным порядком).

Определение Если все задачи в некоторой серии требуют принципиально различных решений, то такую серию мы назовем *серией с независимыми компонентами* или, для краткости, *независимой серией.*

Независимые серии возникают чаще всего в тех случаях, когда в условии задачи граничные условия меняются в сторону расширения области ее определения.

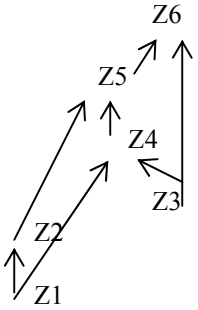
Например, серия из двух задач **N1** «Телевышка» и **N2** «Телебашня», приведенных в Приложении (см. П.6.2), очевидно, является *независимой*: алгоритмы решений этих двух задач различаются очень сильно.

Определение *Серией с зависимыми компонентами (*или, для краткости, *зависимой серией)* назовем серию, состоящую из таких задач, решения которых в той или иной степени базируются на решениях более простых задач из этой же серии.

Пример *зависимой* серии приведен в Приложении (см. П.6.3 и Рисунок 1). При очевидном линейном нарастании сложностей задач **Z1 → Z2 → Z3 → Z4 → Z5 → Z6** в этой серии можно выделить пять *линейных* серий:

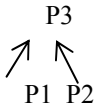
- **Z1 → Z2**: результирующая строка составлена из букв
- **Z1 → Z4**: компоненты результирующей строки наращиваются слева и справа по одному

- **Z2 → Z5 → Z6**: каждая очередная строка строится удвоением предыдущей и вставкой дополнительного компонента между копиями
- **Z3 → Z4 → Z5**: результирующая строка составлена из строк, которые наращиваются слева и справа по одной
- **Z3 → Z6**: результирующая строка составлена из всех ранее сконструированных компонентов



3.2.2. Пирамидальные серии

Определение *Пирамидальной* назовем серию задач, у которых решение более сложных задач может базироваться не на одной, а на нескольких более простых задачах, причем базовые задачи, скорее всего, будут независимыми друг от друга.



Примером *явной пирамидальной серии* может служить серия задач **Z1 → Z2 → Z3 → Z4 → Z5 → Z6** (см. Приложение, П.6.3 и Рисунок 1).

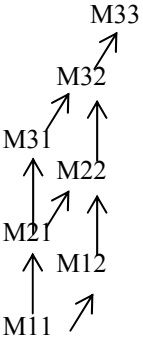
Примером не*явной пирамидальной серии* является серия задач **P1, P2 → P3** (см. Приложение, П.6.4 и Рисунок 2). Обратите внимание, что между задачами **P1** и **P2** нет явно выраженной связи. Если бы она имелась, серия стала бы уже не *пирамидальной*, а *многомерной* (см. следующий пункт).

3.2.3. Многомерные серии

Определение *Многомерной* назовем серию, для задач которой можно выделить не один, а несколько «векторов» возрастания сложности (как аналог частично упорядоченного множества).

В качестве примера приведем двумерную серию (см. Приложение, П.6.5 и Рисунок 3).

Соотношения между задачами таковы, как показано на Рисунок 3 (сложность нарастает снизу вверх и слева направо).



- Задачи левой вертикали (**M11 → M21 → M31**) сформулированы только для букв.
- Задачи правой вертикали (**M12 → M22 → M32**) сформулированы для любых символов.
- Задачи **M11** и **M12** требуют удаления первого символа.
- Задачи **M21** и **M22** требуют удаления последнего символа.
- Задачи **M31**, **M32** и **M33** требуют удаления и первого, и последнего символа.
- Несколько особняком стоит задача **M33**: если результат решения остальных шести задач заведомо укладывается в одну строку, то здесь вполне возможен вариант, когда результирующая строка окажется длиннее 255 символов. Тем самым, решение этой задачи требует совершенно иного алгоритма, чем первые шесть.

Можно сказать, что в этом случае мы имеем не просто многомерную серию задач, а «гибрид» *независимой линейной* серии (**M31 → M32 → M33**) и *зависимой многомерной* серии (**[M11 → M21 → M31]→[M12 → M22 → M32]**).

Приложение

ПРИМЕРЫ ЗАДАЧ

П.1. Полные условия

Тексты задач даны в том виде, в каком они были предложены на олимпиадах.

П.1.1. «Дарвинизм»

Выделяемые части	**Текст задачи «Дарвинизм»**
Лг (ВИ-1)	Под самым потолком экспериментаторской кабины висит банан весом *G* грамм. В помещении также присутствуют маленькая обезьянка и кубический ящик размерами 1x1x1, встав на который, она может дотянуться до банана.
ОП-1	Стены лаборатории ориентированы по сторонам света, юго-западный угол считается имеющим координаты (1, 1), северо-западный – (1, *N*), северо-восточный – (*M, N*), юго-восточный – (*M*, 1), аналогичным образом ориентированы стенки ящика. Ящик тяжелый, передвигать его можно только перекатывая с боку на бок, причем одно перекатывание сдвигает ящик ровно на один шаг в любом направлении. Обезьянка дотягивается до банана только в том случае, когда ящик стоит точно под ним. Все грани ящика сделаны из разных материалов, поэтому имеют различные веса. Про каждую грань известно, сколько калорий тратит обезьянка на то, чтобы перекатить ящик на эту грань (сделать боковую грань нижней). Будем считать, что один грамм банана восстанавливает ровно одну калорию сил.
Зд-1	Если обезьянка выберет самый экономный (с точки зрения затрат калорий) путь перекатывания ящика, а затем съест банан, то останется ли она
НО-1	голодной (вес банана меньше суммы затраченных калорий), наестся (вес банана равен сумме затраченных калорий) или объестся (вес банана больше суммы затраченных калорий)?

ФД-3 (К)	**Формат входных данных:** В первой строке входного файла *input.txt* заданы два натуральных числа *M, N* – размеры комнаты ($1 \le M, N \le 20$); во второй строке – три натуральных числа C, D – место расположения банана ($1 \le C \le M$, $1 \le D \le N$) и вес банана ($1 \le G \le 1000$); в третьей строке два натуральных числа *X, Y* – начальное положение ящика ($1 \le X \le M$, $1 \le Y \le N$); в четвертой строке – шесть натуральных чисел *p1, p2, p3, p4, p5, p6* – затраты на одно перекатывание ящика соответствующей гранью вниз: сначала для грани, которая в исходном положении является нижней, затем для граней, который в начальный момент являются боковыми (в порядке "северная, восточная, южная, западная") и, наконец, для грани, которая в начальный момент является верхней. **Формат выходных данных:** В первой строке выходного файла должно содержаться одно натуральное число – затраты калорий на минимальный путь; во второй строке – слова "hungry" (если обезьянка осталась голодной), "full up" (если она наелась), "full over" (если объелась).
Пр	**Пример** ***input.txt*** 3 3 1 1 10 2 1 10 10 10 4 5 1 ***output.txt*** 10 full up

П.1.2. «Финский язык»

Выделяемые части	**Текст задачи «Финский язык»**
ВИ-3	В финском языке слог может быть образован одним из следующих вариантов: 1) Краткой гласной **a, e, i, o, u, y, ä, ö**; 2) Долгой гласной **aa, ee, ii, oo, uu, yy,ää, öö**; 3) Дифтонгом **ai, au, ei, eu, ie, iu, oi, ou, ui, uo, yi, yö, äi, äy, öi, öy**. Перед дифтонгом может стоять одна гласная (тогда это два слога), два дифтонга подряд не встречаются.
ОП-3	Множественное число слов, стоящих в некоторых косвенных падежах, образуется так: ***<слабая_основа_единственного_числа> + i + <падежное_окончание>*** Косвенные падежи, образуемые по этому правилу, имеют следующие окончания: 1) Аблатив (Abl) **-lta/-ltä** 2) Адессив (Ade) **-lla/-llä** 3) Аллатив (All) **-lle** 4) Инессив (Ine) **-ssa/-ssä** 5) Транслатив (Tra) **-ksi** 6) Элатив (Ela) **-sta/-stä** Если в исходном слове присутствует хотя бы одна из гласных **ä, ö** или **y**, либо все гласные в слове – это только **i** и **e**, то в окончании используется
НО-3	*диэрезис* – надстрочный знак ¨
ОП-3	(это т.н. *правило гармонии гласных*, которому подчиняются все гласные в слове). Гласное окончание основы может изменяться. Изменения подчиняются следующим правилам: 1) Краткие гласные **u, y, o, ö** не изменяются; 2) Краткая гласная **e** исчезает; 3) Краткая гласная **i** превращается в **e**; 4) Долгие гласные сокращаются до кратких; 5) В дифтонгах **uo, yö, ie** исчезает первая гласная, остальные дифтонги не изменяются;

<table>
<tr><td></td><td>6) Гласные a, ä либо исчезают, либо изменяются на o, ö соответственно. Замена происходит в следующих случаях:
a. Если первый слог двусложного слова, оканчивающегося на гласную a, образован краткой или долгой гласной a, e, i, aa, ee, ii;
b. Если многосложное слово (три и более слогов) оканчивается на -kka/kkä, -la/lä, -na/nä, -ija/ijä.</td></tr>
<tr><td>Зд-3</td><td>Напишите программу, которая по заданной основе единственного числа будет строить слово в указанном косвенном падеже, во множественном числе (в соответствии с описанным правилом).</td></tr>
<tr><td>ФД-3
(КС)</td><td>Формат входных данных
В первой строке текстового файла input.txt записано одно число $1 \le N \le 20$.
Далее в каждой из N строк записано по два слова: трехбуквенное обозначение падежа и слабая основа единственного числа (не длиннее чем 20 символов).

Формат выходных данных
В текстовый файл output.txt нужно вывести N строк, каждая из которых является словом в указанном падеже, во множественном числе.
Диэрезисы
Во входном и выходном файлах буквы ä и ö должны быть представлены парой символов: «a:» и «o:» соответственно.</td></tr>
<tr><td>Пр</td><td>Пример
input.txt
12
Ine talo
Ade ja:rve
Ade tuoli
Ela huonee
Abl maa
Ela suo
Ine tyo:
Ade tie
All koira
Ade teema
Ela ikkuna

output.txt
taloissa
ja:rvilla:
tuoleilla
huoneista
mailta
soista
to:issa:
teilla:
koirille
teemoilla
ikkunoista
vena:la:isilta:</td><td>Примечание
talo – дом, taloissa – в домах;
järvi – озеро, järvillä – на озерах;
tuoli – стул, tuoleilla – на стульях;
huone – комната, huoneista – из комнат;
maa – земля, mailta – с земель;
suo – болото, soista – из болот;
työ – работа, töissä –</td></tr>
</table>

	Abl vena:la:ise	на работах; *tie* – дорога, *teillä* – на дорогах; *koira* – собака, *koirille* – собакам; *teema* – тема, *teemoilla* – по темам; *ikkuna* – окно, *ikkunoista* – из окон; *venäläinen* – русский, *venäläisiltä* – от русских

П.1.3. «Таращенко – чемпион!»

Несмотря на то, что в задаче отсутствует часть ***Необходимые определения***, условие отнесено к полным (см. п. 2.1.2, «Отсутствие Определений»), поскольку в данном случае необходимости давать какие-либо определения нет.

Выделяемые части	**Текст задачи «Таращенко – чемпион!»**
ВИ-1 (Лг)	Для того чтобы сделать фигурное катание массовым видом спорта, решили привлечь к судейству всех желающих: соревнования транслируются по Интернету в режиме реального времени, зрители выставляют оценки (в соответствии с правилами), а в зачет фигуристу идет среднее арифметическое всех зрительских оценок. Особое табло отображает итоговую зрительскую оценку.
Зд-2	Необходимо написать программу, которая подсчитывала бы изменение среднего балла по мере поступления зрительских оценок.
ОП-1	Напомним, что по правилам Федерации фигурного катания, выступление фигуриста оценивается так: за каждый технический элемент, выполненный спортсменом, *N* судей выставляют *N* оценок, затем одна самая высокая и одна самая низкая оценки удаляются, для остальных *N*–2 оценок вычисляется среднее арифметическое. Сумма средних баллов по всем техническим элементам является итоговой оценкой выступления.

ФД-2 (Н)	**Формат входных данных** В каждой строке текстового файла *input.txt* содержится по $1 \le S \le 100$ натуральных чисел, каждое из диапазона [1..100] – оценки одного зрителя. Первая оценка соответствует первому техническому элементу, вторая – второму и т.д. Количество зрителей-судей N не может превышать 1000000 человек. Подсчет оценок начинается с момента, когда судей станет трое. Числа S и N заранее неизвестны. **Формат выходных данных** В текстовый файл *output.txt* необходимо вывести $N - 2$ строки по $S + 1$ числу в каждой: последовательность итоговых средних баллов за каждый технический элемент и за все выступление в целом, отображаемых на табло по мере поступления зрительских оценок. Каждый средний балл – это действительное число с двумя знаками после десятичной точки (округление производится по обычным математическим правилам).
Пр	**Пример** *input.txt* *output.txt* 10 20 10 30.00 30.00 30.00 90.00 30 30 30 55.00 45.00 50.00 150.00 100 100 90 53.33 56.66 63.33 173.32 80 60 70 45.00 47.50 52.50 145.00 50 80 100 20 20 20

П.1.4. «ACM»

Несмотря на то, что в задаче отсутствует часть ***Необходимые определения***, условие отнесено к полным (см. п. 2.1.2, «Отсутствие Определений»), поскольку в данном случае необходимости давать какие-либо определения нет.

Первая часть условия отнесена к ***Вводной информации***, а не к ***Описанию проблемы***, как можно было бы подумать, поскольку содержащиеся в ней сведения не являются необходимыми для решения задачи (см. п. 2.4.4, «Задачи на понимание условия»).

Выделяемые части	Текст задачи «ACM»
ВИ-1 (Лг)	Во время олимпиад по программированию, проводимых по правилам Компьютерной Ассоциации ACM (Association for Computer Machinary), любая задача, независимо от трудности, «стоит» ровно один балл, а расчет штрафного времени производится так: для каждой задачи одно неправильное предъявление (если не проходит хотя бы один проверочный тест) оценивается в 20 минут, к итоговой сумме «чисто» штрафных минут после принятия правильно решенной задачи прибавляется еще время, прошедшее от начала олимпиадного тура до момента принятия задачи. Не принятые задачи никакого вноса в штрафное время не делают.
Пр-1	Например, предположим, что к 25-й минуте тура у Вас сдана первая задача со второй попытки; к 43-ей минуте сдана вторая – с первой попытки, в настоящий момент в работе находится еще одна задача, которая уже «провалила» два тестовых предъявления. Следовательно, Ваше штрафное время на текущий момент будет таким: (25 + 20*1) + (43 + 20*0) = 83. Если Вы сдадите третью задачу на 52-ой минуте с третьей попытки, то штрафное время изменится так: (25 + 20*1) + (43 + 20*0) + (52 + 20*2) = 180.
ОП-1	Предположим, что Вы участвуете в олимпиаде по таким правилам. В задачном наборе находится $1 \le N \le 100$ задач, про каждую из которых Вы знаете, сколько времени Вам понадобится на такое ее решение, которое прошло бы все тесты с первой попытки. Среди задач встречаются пары взаимосвязанных, и видно, что для наиболее продуктивной работы сначала нужно решить первую задачу из такой пары, и только затем – вторую (время решения каждой из них известно именно с учетом правильной последовательности их решения). Циклической взаимосвязи между задачами быть не может, то есть невозможна ситуация, когда решение задачи x_1 базируется на решении задачи x_2, решение x_2 – на x_3, а x_3 – на x_1. В каждый момент времени Вы решаете только одну задачу.

Зд-2	Напишите программу, которая определяла бы такой порядок решения задач, который дал бы Вам максимальное количество решенных задач при минимальном штрафном времени. На решение всех задач отведено не более $10 \le M \le 10000$ минут.
ФД-3 (КН)	**Входные данные** В файле *input.txt* содержится следующая информация: • в первой строке: два целых числа N и M, разделенные пробелом; • во второй строке: N целых чисел, каждое из которых является количеством минут, необходимых на решение задачи с соответствующим номером (для всех j: $1 \le P_j \le 100$); • в третьей строке и в остальных строках до конца файла: два целых числа $1 \le F \le N$ и $1 \le S \le N$, разделенные пробелом – информация о парах взаимосвязанных задач (по одной паре на строке). Первой нужно решать задачу с номером F, после этого (может быть, не сразу) – задачу с номером S. **Выходные данные** В файл *output.txt* нужно выдать одно целое число: количество штрафных минут.
Пр-2	**Пример** *input.txt* *output.txt* 3 20 24 10 5 7 1 2 **Комментарий к примеру**. Все три задачи Вы решить не успеете. Следовательно, нужно ограничиться только двумя. Среди трех возможных вариантов (1+2, 1+3, 3+1) самым экономичным является третий. (Штрафное время для указанных вариантов составляет соответственно: 10+15=25, 10+17=27, 7+17=24.)

3.3.

П.2. Неполные условия

П.2.1. «Теорема Вильсона»

Отсутствуют части **ВИ, НО, ОП**.

Выделяемые части	**Текст задачи «Теорема Вильсона»**
ФД-1	В файле заданы $100 \le N \le 100000$ натуральных чисел, каждое не превосходит 10000.
Зд-2	Для каждого числа p нужно вывести остаток от деления $p!$ на p^2.
ФД-3 (Н)	**Входные данные** В файле *input.txt* содержатся натуральные числа (по одному в строке). **Выходные данные** В файл *output.txt* нужно записать остатки от деления – по одному в строке, в порядке поступления чисел.
Пр	**Пример** *input.txt* *output.txt* 100 0 9901 98019900 13 156 4 8

П.2.2. «Дополнение»

Отсутствуют части **ВИ, НО, Зд**.

Выделяемые части	**Текст задачи «Дополнение»**
ОП-2	Заданы $2N$ целых чисел, каждое из интервала [0..10000]. Возможно ли разбить их на N пар так, чтобы суммы чисел каждой пары совпадали?

ФД-3 (С)	**Входные данные** Во входном текстовом файле *input.txt* на первой строке записано число $2N$ ($1 \le N \le 10000$); во второй строке – $2N$ чисел, подлежащих разбиению. Эти числа разделены пробелами. **Входные данные** Вывести нужно либо сумму чисел любой пары, если разбиение возможно, либо слово ‘No’, если разбиение на пары, отвечающие условию, невозможно.
Пр	**Примеры** *input.txt* *input.txt* 6 8 3 2 5 1 6 4 1 1 1 1 1 1 15 15 *output.txt* *output.txt* 7 No

П.2.3. «Счастливый двоечник»

Отсутствуют части **ВИ, ОП, НО, Пр, ФД.**

Выделяемые части	Текст задачи «Счастливый двоечник»
Зд-1	В интервале от 0 до 1024 подсчитайте количество чисел, двоичная запись которых симметрична. Ведущие нули не учитывайте.

П.3. Задачи на понимание условия

П.3.1. «Песнь»

Задача приводится в сокращении.

Выделяемые части	Текст задачи «Песнь»
НО-3	В традиционной музыке используются музыкальные звуки из некоторого набора, именуемого *звукорядом*. Звуки звукоряда принято группировать в *октавы*, в каждой из которых 12 звуков. Порядковый номер звука в пределах одной октавы назовем *нотой*. Таким образом, каждый звук можно задать парой чисел

	– номером октавы и номером ноты. *Номер октавы K* – произвольное целое число, *номер ноты N* принимает значение из интервала [01,12]. Звуки можно обозначать этими двумя числами, записанными рядом без пробелов (второе число всегда двузначное). Эту запись назовем *кодом* Q. Например, Q = –108 для (–1)-й октавы, восьмой ноты. Значение Z, определяемое по формуле Z = K*12 +N назовем *абсолютным номером Z звука в звукоряде* (для приведенного выше примера Z = – 4). Набор всех звуков, ноты которых принадлежат заданному подмножеству P номеров нот, назовем *гармонией G*. Это означает, что любой звук с абсолютным номером Z = K*12 + n, где n – номер ноты из P, принадлежит этой гармонии при любом значении K. Отсюда следует, что гармония однозначно определяется указанием P. Две гармонии назовем *эквивалентными*, если при прибавлении некоторого одного и того же целого числа ко всем абсолютным номерам звуков первой гармонии получаются все элементы второй гармонии. Ограничимся рассмотрением только таких наборов гармоний, в которые наряду с каждой из гармоний входят и все эквивалентные ей. Для описания набора такого вида достаточно указать из каждой совокупности эквивалентных по одной гармонии G_i или соответствующему ей подмножеству P_i. Базой B набора гармоний назовем совокупность всех таких P_i. Всякую совокупность одновременно звучащих звуков (не менее двух) будем называть *аккордом A*. Для некоторого заданного набора гармоний назовем *гармонией аккорда A* такую гармонию G из него, что все звуки аккорда A принадлежат G. Будем говорить, что некоторый звук *в тему* для некоторого аккорда, если этот звук принадлежит хотя бы одной гармонии из множества всех гармоний этого аккорда. Последовательное звучание произвольных звуков назовем *мелодией*. Каждому звуку мелодии может быть сопоставлен аккорд в порядке исполнения мелодии. Будем считать мелодию *благозвучной для этой последовательности аккордов*, если каждый ее звук оказывается в тему для соответствующего ему аккорда. *Кучерявостью* мелодии назовем сумму модулей разностей абсолютных номеров Z последовательно исполняемых звуков данной мелодии.

Зд-3	**Задание** Пусть даны база B набора гармоний, последовательность аккордов A и начальный звук Q. Требуется написать программу, находящую наименее кучерявую из всех благозвучных мелодий, начинающихся с этого звука.

П.3.2. «Фортификация»

Выделяемые части	**Текст задачи «Фортификация»**
ВИ-0 (Лг)	Жил-был очень глупый и очень воинственный король. Для защиты от своих многочисленных врагов он решил построить крепость, но поскольку был очень глупым, сам он сделать этого не мог, и поэтому пригласил военного инженера – специалиста по укреплениям. Признаваться в своей глупости король не хотел, поэтому все предложения инженера, которых он не понимал, он сразу же отвергал. Так что у новой крепости не оказалось ни башен, ни бастионов, ни ронделей.
ОП-1	Замок лишь обнесли крепостной стеной, которая всюду была одинаковой высоты V метров и имела вид квадрата $2*N \times 2*N$ метров.
ОП-0	Однако на строительстве оборонительных фортов инженер каким-то образом сумел все-таки настоять. Король даже обрадовался и велел построить четыре форта – по одному с каждой стороны замка. «Мы поставим их на осях симметрии квадрата!» – воскликнул инженер. король сердито нахмурился (он не любил не только фортификационные, но и математические термины), но спорить не стал.
ОП-1	Высоту фортов решили сделать равной x_1, x_2, x_3 и x_4 метров. Расстояние от всех фортов до крепостной стены было одинаковым – R метров. Между крепостной стеной и фортами король велел выкопать оборонительный ров – в виде четырех прямоугольников, параллельных стенам замка и имеющим длину $2*N$ метров.

ОП-0	Конечно же, эти четыре рва соприкасались только углами, однако король это вовсе не казалось недостатком. Наоборот, так было даже красивее и очень необычно: ни у кого из соседей таких рвов не было, все старались выкопать единый, зачастую кривой ров. Король очень гордился своим изобретением и ни за что не хотел от него отказываться (как мы уже говорили, это был очень глупый король). Пришлось инженеру смириться.
ОП-1	Эскарп и контрэскарп этого рва должны были иметь каменную одежду и быть вертикальными, так что в сечении ров представлял собой простой прямоугольник (о прикрытом пути, гласисах и фоссебергах король и слышать ничего не хотел!). Ширину рва он приказал сделать равной S метрам.
НО-0	Инженер все-таки убедил короля, что если между основанием стены и рвом не будет отступа (бермы), то крепостная стена просто обвалится в ров.
ОП-0	Король испугался и приказал соорудить берму шириной B метров. И еще он приказал сделать ров очень-очень глубоким, чтобы через него никто не мог перебраться. А всю землю, вынутую изо рва, он велел насыпать на его передний край (тем самым, поднимая его переднюю стенку). Инженер указал ему, что земля будет ссыпаться вниз углом, образуя прямоугольный треугольник (о том, что таким образом получится гласис, он мудро промолчал). Король милостиво разрешил треугольник, но велел, чтобы он обязательно был равнобедренным.
ОП-1	Гарнизоны всех фортов должны были видеть сигналы, подаваемые с главной башни замка (она имела в высоту T метров и стояла в самом центре квадрата). Слишком высокая надстройка надо рвом могла закрыть им видимость, поэтому инженеру нужно было выкопать ров такой глубины, чтобы вынутая оттуда земля не образовала бы слишком высокого гласиса.
Зд-1	Напишите программу, которая помогла бы ему максимально возможную глубину рва.
Фд-3 (К)	**Входные данные** В текстовом файле *input.txt* содержится следующая информация: • в первой строке три натуральных числа, описывающие замок и башню: $1 \le N \le 100$, $1 \le V \le 100$, $1 \le T \le 100$;

	• во второй строке три натуральных числа, описывающие ров, берму и расстояние до фортов: $1 \le S \le 10$, $1 \le B \le 10$, $S+B \le R \le 100$; • в третьей строке четыре натуральных числа, описывающие высоты фортов: $1 \le x_1, x_2, x_3, x_4 \le 10$.
Фд-2	**Выходные данные** В текстовый файл *output.txt* нужно вывести одно число – максимально возможную глубину рва. Если король не только глупый, но еще и трусливый и велел построить слишком высокую крепостную стену (или слишком низкие форты), так что ни при каких условиях ни один форт не может увидеть вершину центральной башни, нужно выдать –1.
Пр	**Пример** *input.txt* 13 10 27 2 1 19 3 3 3 3 *output.txt* 56

П.4. Неоднозначные и недоопределенные условия

П.4.1. «Отрезки»

Отсутствуют части **ВИ, НО**.

Выделяемые части	Текст задачи «Отрезки»
ОП-2	Два отрезка на плоскости заданы целочисленными координатами. Отрезки могут стягиваться в точку.
Зд-2 **(неодн.)**	Определите взаимное расположение этих отрезков: • (0) не пересекаются; • (1) пересекаются; • (2) накладываются (при этом не обязательно один из отрезков полностью содержится в другом, допускается и частичное наложение).
ФД-2 (К) **(недоопр.)**	**Входные данные** Входной текстовый файл *input.txt* состоит из двух строк, в каждой из которых находятся по четыре числа, задающих в формате (x_1, y_1, x_2, y_2) расположение отрезка на плоскости.

	Выходные данные В выходной файл *output.txt* следует вывести одно из чисел 0, 1, 2 – номер варианта взаимного расположения отрезков.
Пр	**Примеры** *input.txt* *input.txt* *input.txt* 1 0 2 0 -1 0 1 0 -2 –2 1 1 1 1 2 1 0 1 0 -1 110 110 0 0 *output.txt* *output.txt* *output.txt* 0 1 2

Исправленный вариант ***Задания*** и ***Форматов данных*** может выглядеть так (обратите внимание на изменение степени формализации этих частей):

Зд-3 **(одн.)**	Определите взаимное расположение этих отрезков: • не пересекаются; • пересекаются по одной точке; • пересекаются по нескольким точкам.
ФД-3 (К) **(опр.)**	**Входные данные** Входной файл состоит из двух строк, в каждой из которых находятся по четыре числа, задающих в формате (x_1, y_1, x_2, y_2) расположение отрезка на плоскости. Каждое из чисел находится в диапазоне [–1000..1000]. **Выходные данные** В выходной файл следует вывести одно из чисел 0, 1, 2 в зависимости от того, имеют ли данные отрезки ноль, одну или не менее двух общих точек.

П.4.2. «Полиглоты»

Отсутствуют части **ВИ, НО**.

Выделяемые части	Текст задачи «Полиглоты»
ОП-1	В туристической группе *N* человек, говорящих на разных языках.
Зд-1	Нужно определить минимальное количество переводчиков, необходимых этой группе.

ОП-1	Каждый турист должен понимать либо язык гида, либо язык хотя бы одного переводчика. Каждый переводчик переводит только на один язык. Все переводчики работают одновременно.
ФД-2 (СН) **(недоопр.)**	**Входные данные** В текстовом файле *input.txt* содержится следующая информация: – в первой строке одно целое число $1 \le N \le 100$ – количества туристов в группе; – во второй строке записано название языка, которым пользуется гид; – далее в N строках записаны языки, которыми владеет каждый из туристов. Название любого языка – это строка из $1 \le S \le 20$ русских букв. **Выходные данные** В файл *output.txt* нужно записать одно число – количество переводчиков.
Пр	**Пример** *input.txt* 3 русский английский русский английский немецкий французский немецкий суахили арабский фарси санскрит *output.txt* 1

Исправленный вариант ***Форматов данных*** может выглядеть так:

ФД-3 (СН) **(опр.)**	**Входные данные** В файле *input.txt* содержится следующая информация: – в первой строке одно целое число $1 \le N \le 100$ – количества туристов в группе; – во второй строке записано название языка, которым пользуется гид; – далее в N строках записаны языки, которыми владеет каждый из туристов – не более 10 для каждого человека (одна строка соответствует языковому набору одного туриста). В каждой строке названия языков разделены пробелами и не повторяются. Название любого языка – это строка из $1 \le S \le 20$ русских букв.

	Выходные данные В файл *output.txt* нужно записать одно число – количество переводчиков.

или так (обратите внимание на изменение размерности пространства входных данных):

ФД-3 (КН) **(опр.)**	**Входные данные** В файле *input.txt* содержится следующая информация: – в первой строке одно целое число $1 \le N \le 100$ – количества туристов в группе; – во второй строке записано название языка, которым пользуется гид; – далее в N строках записаны языки, которыми владеет каждый из туристов: сначала одно натуральное число $1 \le L \le 10$ – количество языков, которыми владеет очередной турист, а затем через пробел L названий языков. В строке названия языков не повторяются. Название любого языка – это строка из $1 \le S \le 20$ русских букв. **Выходные данные** В файл *output.txt* нужно записать одно число – количество переводчиков.

П.5. Эквивалентные задачи

П.5.1. «Ключевые слова»

Слабо формализованный вариант условия представлен задачей «Библиотека». Отсутствуют части **ВИ, НО, ФД, Зд, Пр.**

Выделяемые части	Текст задачи «Библиотека»
ОП-1	В библиотечном деле для поиска нужных материалов их содержание обычно характеризуют так называемыми ключевыми словами, заданными или автором материала, или библиотечными работниками с привлечением специалистов по тематике. Нередко такие ключи реально соотносятся с чересчур большим числом источников, среди которых трудно выделить нужное подмножество.

	Более точная идентификация текстов могла бы быть выполнена по информации о частоте вхождения определенных терминов в текст. Эта информация может при необходимости учитывать синонимию.

Формализованный вариант условия представлен задачей «Поиск по ключевым словам». Отсутствуют части **НО, ВИ, Пр.**

Выделяемые части	**Текст задачи «Поиск по ключевым словам»**
ОП-2	Имеется набор ключевых слов. Для некоторых из них заданы также списки их синонимов. Все слова состоят только из латинских букв.
ФД-1 (Н)	Предполагается, что информация о ключевых словах и их синонимах уже записана в файл *keys.txt*: каждый синонимический ряд – на отдельной строке.
Зд-1	Напишите программу, которая для каждого ключевого слова подсчитывала бы, сколько раз оно и все его синонимы встречаются в некотором тексте
ФД-1	(файл *text.txt*), состоящем из латинских слов, пробелов и знаков препинания.
ФД-1	Считайте, что искомые слова встречаются в проверяемом тексте только в той форме, в какой они представлены в списках ключей и синонимов (множественное число падежи, т.п. не учитываются). Результат выдать в порядке убывания частот.

П.5.2. «Компоненты связности»

Слабо формализованный вариант условия представлен задачей «Родственники»**.** Присутствуют все части.

Выделяемые части	**Текст задачи «Родственники»**
ВИ-0 (Лг)	В деревне Большие Ключи живет несколько больших семей.
НО-0	Будем считать, что семья – это абсолютно все люди, являющиеся родственниками друг другу, независимо от степени родства.

ОП-0	Для каждой пары жителей известно, родственники они или нет.
Зд-0	Напишите программу, подсчитывающую количество семей в деревне Большие Ключи.
ФД-2 (КН)	**Формат входных данных** Текстовый файл *input.txt* содержит – в первой строке натуральное число $N \le 1000$ – количество жителей; – в остальных строках: по два имени, разделенных пробелом – пары родственников. (Считаем, что имена всех жителей различны, и состоят не более чем из 10 латинских символов.) **Формат входных данных** В файл *output.txt* нужно вывести одно число – количество семей.
Пр	…

Сильно формализованный вариант условия представлен задачей «Подсчет компонент связности». Отсутствуют части **ВИ, ОП, Пр.**

Выделяемые части	**Текст задачи «Подсчет компонент связности»**
НО-3	*Граф* – это структура, состоящая из одной или нескольких вершин, которые могут быть соединены между собой ребрами, по которым можно двигаться в обе стороны. Считаем, что вершина **V** *достижима* из вершины **S,** если найдется хотя бы один непрерывный путь по ребрам от вершины **S** до вершины **V.** *Компонента связности* – это изолированная часть графа, все вершины которой достижимы друг из друга, а вершины из любой другой компоненты связности – не достижимы.
Зд-3	Напишите программу, подсчитывающую количество компонент связности в графе,
ФД-2 (Н)	заданном перечислением всех его ребер (номеров начальной и конечной вершин для каждого ребра на отдельной строке через пробел). Изолированная вершина задается ее номером на отдельной строке.

П.6. Серийные задачи

Задачи приводятся в сокращении.

П.6.1. Явная серия

(S1) Поход-1

Двое друзей собрались в поход. Взвесив все предметы, которые нужно было взять с собой, они задумались, как разделить между собой общий вес наиболее честным образом.

Напишите программу, которая по заданному количеству предметов ($1 \le N \le 20$) и по их весам (N натуральных чисел, каждое не более 1000) разделила бы этот набор на две части таким образом, чтобы модуль их разности ($|a_1-a_2|$) был наименьшим. Выдать одно число – найденный минимум.

(S2) Поход-2

Двое друзей собрались в поход. Взвесив все предметы, которые нужно было взять с собой, они задумались, как разделить между собой общий вес наиболее честным образом.

Напишите программу, которая по заданному количеству предметов ($1 \le N \le 50$) и по их весам (N натуральных чисел, каждое не более 1000) разделила бы этот набор на две части таким образом, чтобы модуль их разности ($|a_1-a_2|$) был наименьшим. Выдать одно число – найденный минимум.

(S3) Поход-3

Трое друзей собрались в поход. Взвесив все предметы, которые нужно было взять с собой, они задумались, как разделить между собой общий вес наиболее честным образом.

Напишите программу, которая по заданному количеству предметов ($1 \le N \le 50$) и по их весам (N натуральных чисел, каждое не более 1000) разделила бы этот набор на три части таким образом, чтобы сумма модулей их разностей ($|a_1-a_2|+|a_2-a_3|+|a_3-a_1|$) была наименьшей. Выдать одно число – найденный минимум.

П.6.2. Независимая серия

(N1) Телевышка

N точек на плоскости ($2 \le N \le 1000$) заданы своими координатами. Найти радиус минимальной окружности, внутрь или на границу которой попа-

дают все эти точки. Центр этой окружности может и не совпасть ни с одной заданной точкой.

(N2) Телебашня

N точек на плоскости ($2 \le N \le 100000$) заданы своими координатами. Найти радиус минимальной окружности, внутрь или на границу которой попадают все эти точки. Центр этой окружности может и не совпасть ни с одной заданной точкой.

П.6.3. Зависимая серия

(Z1) Буквоед

Построим строку букв следующим образом:

$a_1 = 'a'$
$a_2 = 'bab'$
$a_3 = 'cbabc'$
...
$a_N = lat_N + a_{N-1} + lat_N$

где lat_N – это буква латинского алфавита с номером N.

Напишите программу, которая по номеру позиции в последней строке (a_{26}) будет определять, какая буква на этой позиции стоит.

(Z2) Буквоглот

Построим строку букв следующим образом:

$a_1 = 'a'$
$a_2 = 'aba'$
$a_3 = 'abacaba'$
...
$a_N = a_{N-1} + lat_N + a_{N-1}$

где lat_N – это буква латинского алфавита с номером N.

Напишите программу, которая по номеру позиции в последней строке (a_{26}) будет определять, какая буква на этой позиции стоит.

(Z3) Строкоед

Пусть нам заданы N строк $a_1..a_N$, из них составим другие N строк:

$b_1 = a_1$
$b_2 = a_1+a_2$
$b_3 = a_1+a_2+a_3$
...

$b_N = a_1+...+a_N$

Из этих новых строк составили одну большую строку:

$$S = b_1+b_2+b_3+b_4+...+b_N$$

Напишите программу, которая по номеру позиции K в строке S (начиная с левого края) будет определять, какая буква на этой позиции стоит.

(Z4) Строкоглот

Пусть нам заданы N строк $a_1..a_N$, из них составим другие N строк:

$b_1 = a_1$
$b_2 = a_2+b_1+a_2 = a_2+a_1+a_2$
$b_3 = a3+b_2+a_3 = a_3+a_2+a_1+a_2+a_3$
...
$b_N = a_N+...+a_2+a_1+a_2+...+a_N$

Напишите программу, которая по номеру позиции K в строке b_N (начиная с левого края) будет определять, какая буква на этой позиции стоит.

(Z5) Строкожор

Пусть нам заданы N строк $a_1..a_N$, из них составим другие N строк:

$b_1 = a_1$
$b_2 = b_1+a_2+b_1 = a_1+a_2+a_1$
$b_3 = b_2+a_3+b_2 = a_1+a_2+a_1+a_3+a_1+a_2+a_1$
...
$b_N = b_{N-1}+a_N+b_{N-1}$

Напишите программу, которая по номеру позиции K в строке b_N (начиная с левого края) будет определять, какая буква на этой позиции стоит.

(Z6) Строкожорище

Пусть нам заданы N строк $a_1..a_N$, из них составим другие N строк:

$b_1 = a_1$
$b_2 = b_1+a_2+b_1 = a_1+a_2+a_1$
$b_3 = b_2+a_3+b_2 = a_1+a_2+a_1+a_3+a_1+a_2+a_1$
...
$b_N = b_{N-1}+a_N+b_{N-1}$

Из этих новых строк составили одну большую строку:

$$S = b_1+b_2+b_3+b_4+...+b_N$$

Напишите программу, которая по номеру позиции K в строке S (начиная с левого края) будет определять, какая буква на этой позиции стоит.

Для решения всех шести задач применима одна и та же идея: не нужно никакой рекурсии, не нужно вообще вычислять строку, достаточно рассмотреть ее длину. Во всех задачах номер буквы в латинском алфавите однозначно выражается через номер ее позиции в строке. Это выражение может быть записано математической формулой.

П.6.4. Пирамидальная серия

(P1) Комментарии

Из заданной правильно работающей Pascal-программы удалите все комментарии.

(P2) Синонимия

Имеется набор каких-то ключевых слов. Для некоторых из них заданы также списки их синонимов. Предполагается, что информация о ключевых словах и их синонимах уже записана в файл с именем dic: каждый синонимический ряд – на отдельной строке.

Напишите программу, которая для каждого ключевого слова подсчитывала бы, сколько раз оно и все его синонимы встречаются в некотором тексте. Считайте, что искомые слова встречаются в проверяемом тексте только в той форме, в какой они представлены в списках ключей и синонимов (падежи, склонения, т.п. не учитываются). Результат выдать в порядке убывания частот.

(P3) Составные операторы

Напишите программу, которая подсчитывала бы в тексте некоторой правильно работающей Pascal-программы количество составных операторов.

Приведем здесь решение задачи **P3**. Это решение не является оптимальным, поскольку требует нескольких проходов по файлу. Однако оно демонстрирует использование ранее написанных решений задач **P1** и **P2**.

Составной оператор – это несколько операторов, объединенных в одну группу с помощью операторных скобок `begin/end`, поэтому количество составных операторов с очевидностью можно вычислить, посчитав количество вхождений слова `begin` в текст программы. Слово `end` не подходит для этого, так как описание структуры `record` и оператор-переключатель `case` завершаются словом `end`, однако не являются составными операторами.

Конечно, этот подсчет не даст правильного результата, если в тексте программы хотя бы один раз в операторные скобки было заключено менее двух операторов (один или ноль). Однако для простоты будем считать такие «вырожденные» составные операторы наравне с обычными.

Разобьем решение задачи на три этапа:

I. Ясно, что при подсчете те слова `begin`, которые находятся в комментариях или между апострофами, не должны влиять на конечный результат. Поэтому сначала воспользуемся программой, которая из правильной Pascal-программы вырезает все комментарии. Это решение задачи **P1**. В результате ее применения мы вновь получим правильную Pascal-программу, в которой уже нет комментариев, однако могли еще остаться строки, заключенные между апострофами.

Ia. Этот этап можно считать частью первого: уберем из текста программы все строки, заключенные между апострофами. В результате, в промежуточном файле уже не будет содержаться правильная Pascal-программа, так как от каждой строки останется только один апостроф. Однако нам это и не важно. Главное, что теперь мы готовы считать в нашей исходной программе слова `begin`.

II. В зависимости от взгляда на формальное определение языка функции и процедуры можно также считать составными операторами. Например, скажем, что это не так. Тогда из общего количества слов `begin`, найденных в тексте программы, нужно будет вычесть количество процедур и функций, ведь каждая из них также начинается с этого слова.

Для подсчета воспользуемся программой, решающей задачу **P2**: создадим файл-словарь `dic`, в который запишем две строки

```
begin
procedure function
```

Пусть *Nb* – количество встретившихся слов `begin`, а *Npf* – количество встретившихся слов `procedure` или `function`, которые при нашем подсчете, очевидно, являются абсолютными синонимами.

Если мы считаем процедуры и функции составными операторами, то интересующее нас число *Ns* (количество составных операторов) совпадает с числом *Nb*, в противном же случае вычисляется через их разность: $Ns = Nb - Npf$.

III. Сама программа также заключается в слова `begin/end`, поэтому от итогового количества нужно отнять еще одну единицу. Полученное число и является количеством составных операторов в исходной программе.

П.6.5. Многомерная серия

(M11) Усечение текста

Напишите программу, которая из входного текста (строки или текстового файла) оставляет только первое вхождение каждой буквы.

(M12) Усечение текста

Напишите программу, которая из входного текстового файла, состоящего из не управляющих символов (с кодами из промежутка [32..254]) оставляет только первое вхождение каждого символа.

(M21) Усечение текста

Напишите программу, которая из входного текста (строки или текстового файла) оставляет только последнее вхождение каждой буквы.

(M22) Усечение текста

Напишите программу, которая из входного текстового файла, состоящего из не управляющих символов (с кодами из промежутка [32..254]) оставляет только последнее вхождение каждого символа.

(M31) Усечение текста

Напишите программу, которая из входного текста (строки или текстового файла) удаляет все неповторяющиеся буквы, а для остальных букв оставляет ровно два вхождения: первое и последнее.

(M32) Усечение текста

Напишите программу, которая из входной строки, состоящей из не управляющих символов (с кодами из промежутка [32..254]) оставляет только первое и последнее вхождение каждого символа. Неповторяющиеся символы нужно удалить.

(M33) Усечение текста

Напишите программу, которая из входного файла с текстом, состоящим из не управляющих символов (с кодами из промежутка [32..254], длина текста не более 1 Mb) оставляет только первое и последнее вхождение каждого символа. Неповторяющиеся символы нужно удалить.

СПИСОК ЛИТЕРАТУРЫ

1. Абрамович С.М., Бляхеров О.В., Букатов А.А., Рутман В.Л. Технология программирования: Методы и средства (Современное состояние). – Ростов н/Д: Изд-во Рост. ун-та, 1992.
2. Андреева Т. А. Структура и классификация текстов задач (тезисы доклада) // Мат. IV междунар. конф. «Перспективы систем информатики», 2001. Секция «Школьная информатика». – 2001. – С. 10–11.
3. Андреева Т. А. Структура и классификация текстов олимпиадных задач // Компьютерные инструменты в образовании. – 2002. – № 3–4. – С. 50–59.
4. Андреева Т. А. Серийные олимпиадные задачи // III междунар. телеконф. «Информационные технологии в общеобразовательной школе». – 2002–2003.
5. Андреева Т. А. Серийные задачи в программировании (тезисы доклада) // Мат. V междунар. конф. «Перспективы систем информатики», 2003. Секция «Школьная информатика». – 2003. – С. 2–4.
6. Липаев В.В. Управление разработкой программных средств: Методы, стандарты, технология. – М.: Финансы и статистика, 1993.
7. Майерс Г. Искусство тестирования программ. – М.: Финансы и статистика, 1980.
8. Майерс Г. Надежность программного обеспечения. – М.: Мир, 1992
9. Поттосин И. В. О критериях добротности программ // Системная информатика. – Новосибирск, 1998.

Т. А. Андреева

СТРУКТУРНЫЙ АНАЛИЗ И СИСТЕМАТИЗАЦИЯ УСЛОВИЙ ОЛИМПИАДНЫХ ЗАДАЧ ПО ПРОГРАММИРОВАНИЮ

Препринт
149

Рукопись поступила в редакцию 10.12.08
Рецензент Т.Г. Чурина
Редактор Т. М. Бульонкова

Подписано в печать 28.12.09
Формат бумаги 60 × 84 1/16 Объем 2.9 уч.-изд.л., 3.1 п.л.
Тираж 60 экз.

Центр оперативной печати «Оригинал 2»
г.Бердск, ул. О. Кошевого, 6, оф. 2, тел. (383-41) 2-12-42