Муниципальный этап Всероссийской олимпиады школьников по информатике

в 2015 – 2016 учебном году

Разборы решений и идеи тестов

Муниципальный этап Всероссийской олимпиады школьников по информатике в 2015 – 2016 учебном году

11 класс

Время выполнения задач — 4 часа

Ограничение по времени — 2 секунды на тест

Ограничение по памяти — 64 мегабайта

11.1. «Вычитаем единицу». Задано трёхзначное натуральное число, не содержащее в своей десятичной записи нулей. Какое число получится, если из каждой десятичной цифры исходного числа вычесть единицу?

Формат входа: Задано единственное трёхзначное натуральное число, большее 200 и не содержащее нулей в своей десятичной записи.

Формат выхода: Выдайте новое число.

Пример

Вход:	*Выход:*
536	425

11.2. «Выгодное финансовое вложение». Родители Пети Торопыжкин решили открыть банковский вклад на P дней. Выбранный ими банк предлагает несколько видов вкладов, каждый из которых описывается периодом капитализации (в днях) и процентной ставкой, добавляемой к сумме вклада в конце каждого полного срока капитализации. Если вклад снимается до окончания очередного срока капитализации, то за неполный период проценты не начисляются. Помогите Пете написать программу, которая выяснит, который из видов вкладов наиболее выгоден.

Формат входа: В первой строке задано через пробел два целых числа: P — длина в днях периода, на который планируется разместить деньги в банке, и n — количество видов вкладов ($1 \leqslant P \leqslant 10^5$, $1 \leqslant n \leqslant 100$). В следующих n строках идут описания видов вкладов: два целых числа d_i и s_i — период капитализации в днях очередного вклада и процент по этому вкладу за период капитализации ($1 \leqslant d_i \leqslant 10^5$, $0 \leqslant s_i \leqslant 20$).

Формат выхода: Выведите единственное целое число — номер наиболее выгодного вида вклада (в том порядке, в каком они задавались на входе; первый вид имеет номер 1).

Пример

Вход:	*Выход:*
200 2 100 20 9 2	2

11.3. «В шеренгу становись!». Класс Пети Торопыжкина строится по росту (по убыванию роста) в шеренгу на урок физкультуры. Некоторые из учеников имеют одинаковый рост, так что их взаимные перестановки не изменят правильности шеренги, но сделают шеренгу новой. Нужно посчитать количество различных шеренг, в которые может выстроиться Петин класс. Так как это число может быть слишком большим, то посчитайте его по модулю 50 000 (то есть посчитайте остаток от деления этого числа на 50 000).

Формат входа: В первой строке задано n — количество групп школьников, в каждой из которых ребята имеют один рост ($1 \leqslant n \leqslant 10^5$). Во второй строке через пробел перечислены величины p_i — численность каждой группы ($1 \leqslant p_i \leqslant 10^6$).

Формат выхода: Выдайте единственное цело число — количество различных шеренг, взятое по модулю 10^5.

Пример

Вход:	*Выход:*
`4` `1 7 5 4`	`15200`

11.4. «Получить палиндром». Петя Торопыжкин изучает на уроках информатики работу со строками. Он уже изучил первую из операций — обращение подстроки, то есть изменение порядка символов в подстроке, заданной индексом своего первого символа и длиной, на обратный. Его заинтересовало, можно ли при помощи только этой операции превратить заданную непустую строку в *палиндром*, то есть в строку, одинаково читающуюся с начала и с конца. Помогите ему, написав программу, которая для заданной строки будет отвечать на этот вопрос. В случае возможности получения палиндрома программа должна выдавать последовательность операций, приводящую к получению палиндрома, максимально возможного в лексикографическом порядке. Количество команд не должно превосходить длины строки. Если превращение невозможно, программа должна выдать наименьшее количество символов, которые надо удалить из исходной строки, чтобы такое превращение стало возможным.

Формат входа: В первой строке задано единственное натуральное число n, не превосходящее 10^4 — длина строки. В второй строке задана строка, которую хочет обработать Петя. Она состоит из n заглавных латинских символов.

Формат выхода: Если заданную строку можно превратить в палиндром, то в первой строке выдайте `YES`. Во второй строке выдайте количество операций, за которые может быть получен палиндром, максимальный в лексикографическом порядке (величина, не превосходящая длины строки). В следующих строках описываются операции обращений подстрок (по одной в строке): пара целых чисел через пробел — индекс (отсчитываемый от 1) первого символа обращаемой подстроки и её длина. В случае невозможности превращения строки в палиндром в

первой строке выдайте `NO`, а во второй выдайте единственное целое число — минимальное количество символов, которые надо удалить из строки, чтобы её всё-таки можно было превратить в палиндром.

Пример 1

Вход:	*Выход:*
5	YES
XYXZY	2
	3 2
	1 2

Пример 2

Вход:	*Выход:*
5	NO
YXZXW	2

Примечание: Во втором примере из строки достаточно удалить два символа, например, `Y` и `W`, после чего строку можно превратить в палиндром. Удалением одного символа обойтись нельзя.

11.5. «Лягушка-поскакушка». В саду у Пети Торопыжкина много лягушек. Вот и сейчас лягушка скачет по прямой садовой дорожке, вымощенной n квадратными плитками, уложенными в ряд. Вначале она сидит на первой плитке, и, конечно, ей хочется проскакать всю дорожку. Каждый раз лягушка прыгает в направлении нужного ей конца дорожки. Первый прыжок она может сделать максимум на m плиток (с 1-й на $(m+1)$-ю), а длина каждого следующего прыжка l_i не может превосходить величины $\lfloor k/l_{i-1}^2 \rfloor$, где $\lfloor \cdot \rfloor$ — округление вниз, а l_{i-1} — величина предыдущего прыжка. Петя задумался: за какое минимальное количество прыжков лягушка может выскочить за пределы последней n-й плитки? Напишите программу, которая находит это число.

Формат входа: В первой строке заданы три целых числа n — количество плиток на дорожке, m — максимальная дальность первого прыжка — и k — параметр прыжка ($1 \leqslant n, m \leqslant 10^5$, $1 \leqslant k \leqslant 1000$).

Формат выхода: Выдайте единственное целое число — минимальное количество прыжков, за которое лягушка может соскочить с дорожки.

Пример

Вход:	*Выход:*
10 5 10	2

Примечание: В примере лягушка может сначала прыгнуть на одну плитку, а вторым прыжком — на 10 плиток, тем самым оказавшись за пределами дорожки.

Муниципальный этап Всероссийской олимпиады школьников по информатике в 2015 – 2016 учебном году
11 класс. Разбор решений и идеи тестов

11.1. «Вычитаем единицу». *Задано трёхзначное натуральное число, не содержащее в своей десятичной записи нулей. Какое число получится, если из каждой десятичной цифры исходного числа вычесть единицу?*

Собственно, для решения задачи нужно понять, что ответ есть разность исходного числа и 111, и нет нужды производить разложение числа на цифры и другие сложные операции с данным числом.

Идеи тестов:

1–10. Случайные тесты.

11.2. «Выгодное финансовое вложение». *Родители Пети Торопыжкин решили открыть банковский вклад на P дней. Выбранный ими банк предлагает несколько видов вкладов, каждый из которых описывается периодом капитализации (в днях) и процентной ставкой, добавляемой к сумме вклада в конце каждого полного срока капитализации. Если вклад снимается до окончания очередного срока капитализации, то за неполный период проценты не начисляются. Помогите Пете написать программу, которая выяснит, который из видов вкладов наиболее выгоден.*

В целом, решение данной задачи включает две процедуры: подсчет суммарного изменения начальных накоплений при заданном плане и поиск плана, дающего максимальное приращение накоплений. Вторая часть есть стандартный алгоритм поиска максимума.

Первая процедура тоже не является сложности. Пусть r_i — количество полных периодов капитализации i-го вклада: $r_i P \operatorname{div} d_i$. За один период вклад увеличится в $(1 + s_i/100)$ раз: на каждую единицу денег будет начислен процент $s_i/100$. Стало быть, суммарно за r_i периодов капитализации вклад увеличится в $(1 + s_i/100)^{r_i}$ раз. Максимум этих величин пл всем i надо найти и выдать номер i, доставляющий этот максимум.

При отсутствии в библиотеке языка специальной функции возведения вещественного числа в степень можно использовать школьную формулу $a^b = e^{b \cdot \ln a}$ (ln — натуральный логарифм, логарифм по основанию числа e), а функции экспоненты `exp` (возведения числа e в заданную вещественную степень) и `log` логарифма натурального есть во всех языках. Ну или принимая во внимание, что r_i — величина целая, можно возведение в степень заменить повторным произведением.

Идеи тестов:

1. Минимальный тест: $n = 1$.
2. Максимальный тест: все величины имеют максимальные значения.

3–10. Случай

11.3. «В шеренгу становись!». *Класс Пети Торопыжкина строится по росту (по убыванию роста) в шеренгу на урок физкультуры. Некоторые из учеников имеют одинаковый рост, так что их взаимные перестановки не изменят правильности шеренги, но сделают шеренгу новой. Нужно посчитать количество различных шеренг, в которые может выстроиться Петин класс. Так как это число может быть слишком большим, то посчитайте его по модулю* 50 000 *(то есть посчитайте остаток от деления этого числа на* 50 000*).*

Средняя сложность данной задачи обусловлена необходимостью применения знаний из комбинаторики и соображений относительно работы с числами по заданному модулю.

Известно, что общее количество перестановок из p различных предметов равно $p! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot p$ — факториалу числа p. Эта величина очень быстро растёт с увеличением p; факториал числа 12 входит в тип int, а 13 — уже нет. Именно этим обусловлен то, что в задаче требуется вычислить не общее количество различных шеренг, а остаток от деления этого числа на 50 000.

Если у нас есть две группы из p_1 и p_2 объектов, то с каждой из $p_1!$ перестановок первой группы можно соединить $p_2!$ перестановок второй, что в общем даёт $p_1! \cdot p_2!$ различных шеренг. Вообще, общее количество перестановок в n группах равно произведению $p_1! \cdot p_2! \cdot p_3! \cdot \ldots \cdot p_n!$. То есть в задаче требуется вычислить остаток от деления такой величины на 50 000.

Несложно обосновываемый факт из теории чисел, облегчающий эту задачу, заключается в том, что $(a \cdot b) \bmod c = \big((a \bmod c) \cdot (b \bmod c)\big) \bmod c$. То есть, чтобы найти остаток от произведения, можно найти остаток от произведения остатков. Таким образом,

$$p! \bmod c = \Big(\ldots \Big(\Big(\big((1 \bmod c) \cdot (2 \bmod c) \big) \bmod c \Big) \cdot \Big(3 \bmod c \Big) \Big) \bmod c \ldots \Big) \bmod c.$$

То есть при подсчёте факториала вместо умножения на очередной сомножитель уже накопленного произведения (по модулю 50 000) умножаем на остаток от деления этого сомножителя на 50 000 и ищем остаток от деления полученного произведения:

```
res := 1;
for i := 1 to n do
  res := ( res * (i mod 50000) ) mod 50000;
```

Эта идея используется и при вычислении факториалов и при вычислении произведения факториалов.

Некоторая тонкость заключается в том, что факториалы (вернее, их остатки) надо просчитать заранее, так как независимое вычисление факториала для каждого из чисел, в целом, может занять слишком большое время. Поэтому решение данной задачи выглядит следующим образом:

```
const
  MaxP = 1000000;
  K = 50000; var
  f: array[1..MaxP] of integer;
  res, p, n, i: integer;
begin
(* считаем факториалы *)
  f[1] := 1;
  for i := 2 to MaxP do f[i] := (f[i-1] * (i mod K)) mod K;

(* считываем числа и считаем результат *)
  res := 1;
  read(n);
  for i := 1 to n do begin
    read(p);
    res := (res * f[p]) mod K;
  end;
  writeln(res);
end.
```

Идеи тестов:

1. Минимальный тест: $n = 1$, $p_1 = 1$.
2. Минимальный тест: $n = 1$, $p_1 = 5$.
3. Минимальный тест: $n = 1$, $p_1 = 50$.
4. $n = 10$, все $p_i = 1$.
5. $n = 10$, все p_i различные, меньшие 12.
6. $n = 10$, все p_i различные, в том числе и бóльшие 12.
7. $n = 100$, все $p_i = 1$.
8. $n = 100$, все p_i различные, меньшие 12.
9. $n = 100$, все p_i различные, в том числе и бóльшие 12.
10. Максимальный тест: $n = 10^5$, все $p_i = 1$.
11. Максимальный тест: $n = 10^5$, все p_i различные, меньшие 12.
12. Максимальный тест: $n = 10^5$, все p_i различные, в том числе и бóльшие 12.

13–20. Случайные тесты.

11.4. «Получить палиндром». *Петя Торопыжкин изучает на уроках информатики работу со строками. Он уже изучил первую из операций — обращение*

подстроки, то есть изменение порядка символов в подстроке, заданной индексом своего первого символа и длиной, на обратный. Его заинтересовало, можно ли при помощи только этой операции превратить заданную непустую строку в палиндром, то есть в строку, одинаково читающуюся с начала и с конца. Помогите ему, написав программу, которая для заданной строки будет отвечать на этот вопрос. В случае возможности получения палиндрома программа должна выдавать последовательность операций, приводящую к получению палиндрома, максимально возможного в лексикографическом порядке. Количество команд не должно превосходить длины строки. Если превращение невозможно, программа должна выдать наименьшее количество символов, которые надо удалить из исходной строки, чтобы такое превращение стало возможным.

По мнению программного комитета, данная задача не очень сложна идейно, но требует аккуратного технического исполнения, так что, в целом, сложность этой задачи выше среднего.

Для работающих на Паскале некоторую сложность может вызвать ввод столь длинной строки: стандартный тип `string` хранит лишь строки длиной не более 255 символов, а чтение такой строки в переменную типа `AnsiString` целиком не поддерживается операторами `read`/`readln`. Однако при наличии длины рабочей строки среди входных данных позволяет считывать её посимвольно и хранить в массиве `array[1..MaxLen] of char`, где `MaxLen` — максимальная длина строки, константа 10^4.

Дальнейшая обработка строки обуславливается следующим классическим наблюдением: строку можно превратить в палиндром, если количество символов, встречающихся в этой строке нечётное число раз, равно 0 или 1. Соответственно, после того, как строка считана в память (или в процессе считывания, если считывание идёт посимвольно) подсчитываем количество символов каждого типа (от `A` до `Z`). Если «нечётных» видов символов больше 1, то строку непосредственно превратить в палиндром нельзя, а чтобы это можно было сделать, нужно удалить из строки по одному символу каждого «нечётного» вида кроме одного.

Если же «нечётных» видов символов 0 или 1, то начинаем превращать строку в палиндром, максимальный в лексикографическом порядке. Для этого ищем символ максимальный из тех, что ещё не стоят на своих местах (для этого нужно помнить, сколько символов уже поставлено) и входящих в необработанную часть строки хотя бы два раза, и обращаем соответствующую подстроку так, чтобы он встал на место, следующее за уже сформированной начальной частью строки. Затем ищем подстроку такую, чтобы такой же символ встал на место перед хвостовой сформированной частью строки. Каждое обращение подстроки выполняется буквально: переставляются соответствующие элементы массивов. По ходу дела, запоминаем проделанные операции. После конца обработки, выдаем в результат их количество и описания. Заметим, что если у нас есть символ, входящий нечётное число раз, то его непарное вхождение автоматически окажется в середине

строки в результате такой обработки.

Время работы алгоритма — $O(n^2)$, где n — длина входной строки. Действительно, нам нужно расставить n символов на свои места, для чего надо переставлять местами символы подстроки, длина которой в худшем случае есть длина всей строки. Заданные ограничения дают возможность сделать описанную обработку в рамках ограничений по времени.

Поскольку в случае возможности превращения строки в палиндром требуемая последовательность операций, вообще говоря, неединственна (например, в первом из примеров к задаче можно поменять местами первую и вторую операции), то проверка задачи ведётся при помощи анализатора, который проделывает следующие действия. Считывается строка из входных данных, ответ жюри и ответ участника. Проверяется совпадение типа ответов (`YES`/`NO`). Если они не совпадают, ответ участника признаётся неверным. В случае совпадения ответов типа `NO` сравниваются количества символов, требуемых к удалению из строки. В случае совпадения ответов типа `YES` прверяется, что количество операций, выданных участником, не слишком большое. Затем проводятся операции, выданные участником, и на копии строки — операции, предложенные жюри. Результаты сравниваются. В случае несовпадения результатов считается, что участник не получил палиндром, максимальный в лексикографическом порядке, и его ответ отвергается.

Идеи тестов:

1. Строка длины 1.
2. Строка длины 2, разные символы.
3. Строка длины 2, одинаковые символы.
4. Строка длины 3, три вида символов.
5. Строка длины 3, два вида символов, строка ещё не палиндром.
6. Строка длины 3, два вида символов, строка уже палиндром.
7. Строка длины 4, два вида символов, строка ещё не палиндром.
8. Строка длины 4, два вида символов, строка уже палиндром, но не максимальный.
9. Строка длины 4, два вида символов, строка уже максимальный палиндром.
10. Строка длины 4, три вида символов.
11. Строка длины 4, четыре вида символов.
12. Строка длины 100, много видов символов, каждого — чётное число, строка еще не палиндром.
13. Строка длины 100, много видов символов, каждого — чётное число, строка уже палиндром, но не максимальный.
14. Строка длины 100, много видов символов, каждого — чётное число, строка уже максимальный палиндром.
15. Строка длины 101, много видов символов, каждого, кроме одного — чёт-

ное число, строка еще не палиндром.

16. Строка длины 101, много видов символов, каждого, кроме одного — чётное число, строка уже палиндром, но не максимальный.
17. Строка длины 101, много видов символов, каждого, кроме одного — чётное число, строка уже максимальный палиндром.
18. Строка длины 100, много видов символов, много «нечётных» видов символов.

19–25. То же, что в тестах 12–18, только длина строки 1000/1001.
26–32. То же, что в тестах 12–18, только длина строки максимальная $10^4/99999$.
33–36. Случайные тесты, длина строки 30–50.
37–40. Случайные тесты, длина строки 100–200.
41–45. Случайные тесты, длина строки 1000–5000.
46–50. Случайные тесты, длина строки 90000–100000.

11.5. «Лягушка-поскакушка». *В саду у Пети Торопыжкина много лягушек. Вот и сейчас лягушка скачет по прямой садовой дорожке, вымощенной n квадратными плитками, уложенными в ряд. Вначале она сидит на первой плитке, и, конечно, ей хочется проскакать всю дорожку. Каждый раз лягушка прыгает в направлении нужного ей конца дорожки. Первый прыжок она может сделать максимум на m плиток (с 1-й на $(m+1)$-ю), а длина каждого следующего прыжка l_i не может превосходить величины $\lfloor k/l_{i-1}^2 \rfloor$, где $\lfloor \cdot \rfloor$ — округление вниз, а l_{i-1} — величина предыдущего прыжка. Петя задумался: за какое минимальное количество прыжков лягушка может выскочить за пределы последней n-й плитки? Напишите программу, которая находит это число.*

Данная задача, по мнению программного комитета, является сложной, так как привлекает не вполне тривиальный вариант принципа динамического программирования. Кроме того, для получения полного решения требуется весьма аккуратная техническая реализация данного принципа.

Для начала заметим, что лягушке нельзя прыгать больше, чем на $L = \lfloor \sqrt{k} \rfloor$ плиток, кроме как в последнем прыжке. В противном случае следующий прыжок можно будет сделать не более, чем на ноль плиток, то есть лягушка после слишком длинного прыжка дальше прыгать не сможет. Максимально допустимая длина разрешённого прыжка (при наибольшем значении k) есть $\lfloor \sqrt{1000} \rfloor = 31$ плитки.

Основной структурой данных будет двумерный массив `plates` целых чисел. В ячейке с индексом `[i,j]` будет храниться количество прыжков, за которые можно добраться до `i`-й плитки с величиной последнего прыжка `j`; индекс `i` меняется в диапазоне от 1 до n, индекс `j` — в диапазоне от 1 до L. В языках без возможности динамического создания массивов следует отвести массив с запасом с диапазоном `1..10000` по первому индексу и диапазоном `1..31` по второму. Если ячейки массива сделать типа int, то потребный объём памяти будет несколько больше мегабайта. Соответственно, размер этого массива будет определять и наихудшее

время работы программы — $O(n \cdot \sqrt{k} \cdot \sqrt{k}) = O(nk)$: n — диапазон цикла обработки плиток, $\sqrt{k}$ — диапазон цикла при обработке очередной плитки, $\sqrt{k}$ — наибольший диапазон прыжка. От параметра m сложность алгоритма не зависит.

Кроме того, храним переменную `res`, которая хранит текущий минимум прыжков, выводящих лягушку с дорожки.

В начале программы `res` равна $+\infty$, то есть величине, заведомо большей, чем потребное количество прыжков. Например, в качестве этой величины можно выбрать 10^6. В массиве `plates` все ячейки с индексом `i` $\geqslant 2$ также заполняются бесконечностью (эти данные ещё не просчитаны), а ячейки с индексом `i` $= 1$, заполняются нулями — лягушка уже стоит на первой плитке, она туда «доберётся» за 0 прыжков.

Вначале надо проверить условие $m + 1 > n$. Если это так, то лягушка за один прыжок может соскочить с дорожки. В этом случае программу следует завершить с результатом 1.

В противном случае запускаем обработку массива `plates`. Сначала отдельно обрабатывается первый прыжок; особенность его обработки в том, что у него нет предыдущего прыжка и его дальность ограничена минимумом величин m и L (то есть тем, что можно сделать, и тем, что разумно сделать). Соответственно, для всех `s` от 1 до $\min\{m, L\}$ присваиваем `plates[s+1,s] := 1`.

Затем обрабатываем последующие прыжки. Пересчёт идёт по индексам `i` от 2 до n, а для каждого значения индекса `i` по индексу `j` от 1 до L. Если значение `plates[i,j]` меньше бесконечности, то есть если на `i`-ю плитку лягушка может допрыгать с последним прыжком величины `j`, то смотрим, куда она может прыгнуть дальше. Для всех значений счётчика `s` от 1 до $\lfloor k/j^2 \rfloor$ (то есть для всех возможных величин нового прыжка) проверяем, если `i + s` $> n$ (то есть если лягушка спрыгивает с дорожки), то пересчитываем `res`:

$$\texttt{res} = \min\{\texttt{res}, \texttt{plates[i,j]} + 1\}.$$

Если же `i + s` $\leqslant n$, то пересчитываем информацию по плитке, куда доскочит лягушка:

$$\texttt{plates[i+s,s]} = \min\{\texttt{plates[i+s,s]}, \texttt{plates[i,j]} + 1\}.$$

После окончания работы этого двойного цикла переменная `res` будет содержать ответ.

Примечание: Возможно, задача решается каким-либо «жадным» алгоритмом, однако он не вполне очевиден, и подход, основанный на принципе динамического программирования, достаточно разумен.

Идеи тестов:

1. Минимальный тест: $n = 1$, $m = 1$, $k = 1$.
2. Минимальный тест: $n = 10$, $m = 1$, $k = 1$.

3. Минимальный тест: $n = 10$, $m = 10$, $k = 1$.
4. Минимальный тест: $n = 10$, $m = n/2 = 5$, $k = 1$.
5. Минимальный тест: $n = 10$, $m = 10 - 1 = 9$, $k = 1$.
6. Минимальный тест: $n = 10$, $m = 1$, $k = 10$.

7–11. То же, что в тестах 2–6, только вместо 10 — 100.

12–16. То же, что в тестах 2–6, только вместо 10 — 10^5 (1000 для k).

17. Максимальный тест: $n = 10^5$, $m = 10^5$, $k = 1000$.
18. Максимальный тест: $n = 10^5$, $m = 10^5 - 1 = 99999$, $k = 1000$.

19–22. Случайные тесты, $m + 1 > n$, $10 \leqslant n \leqslant 100$.

23–26. Случайные тесты, $m + 1 < n$, $m > L$, $10 \leqslant n \leqslant 100$.

27–30. Случайные тесты, $m + 1 < n$, $m < L$, $10 \leqslant n \leqslant 100$.

31–42. Такие же случайные тесты, $1000 \leqslant n \leqslant 10000$.

43–50. Такие же случайные тесты, $90000 \leqslant n \leqslant 100000$.