

Муниципальный этап
Всероссийской олимпиады школьников
по информатике

в 2016 – 2017 учебном году

Разборы решений и идеи тестов

Муниципальный этап Всероссийской олимпиады
школьников по информатике
в 2016 – 2017 учебном году
11 класс

Время выполнения задач — 4 часа

Ограничение по времени — 2 секунды на тест

Ограничение по памяти — 64 мегабайта

11.1. «Уловка–22». Для заданного n найдите величину $2^n \bmod 22$.

Формат входа: Задано единственное целое число n ($0 \leq n \leq 10^4$).

Формат выхода: Выдайте требуемую величину.

Пример 1		Пример 2	
<u>Вход:</u>	<u>Выход:</u>	<u>Вход:</u>	<u>Выход:</u>
3	8	46	20

11.2. «Готовимся к весне». В школе Пете Торопыжкину дали поручение к весенней ярмарке изготовить H скворечников, которые будут продаваться за s руб. за штуку. В Интернете Петя нашёл чертежи n различных видов скворечников; скворечник i -го вида изготавливается из b_i стандартных деревянных заготовок. Также был найден поставщик таких заготовок, делающий скидки за размер закупки: если покупается не более v_1 заготовок, то их отпускная цена p_1 руб. за штуку; если покупается больше v_1 , но не более v_2 заготовок, то их отпускная цена $p_2 < p_1$ руб. за штуку; если покупается больше v_2 , но не более v_3 заготовок, то их отпускная цена $p_3 < p_2$ руб. за штуку; и т.д., если покупается более v_m заготовок, то их цена p_{m+1} руб. за штуку. Чтобы сделать процесс изготовления наиболее простым, Петя решил делать скворечники одного вида и покупать ровно столько заготовок, сколько требуется для изготовления нужного числа скворечников. Вопрос, который его интересует: какую максимальную прибыль он может организовать для школы от продажи скворечников?

Формат входа: В первой строке задано через пробел два целых числа, характеризующих техническое задание: H — требуемое количество скворечников — и s — цена продажи одного скворечника ($1 \leq H \leq 100$, $1 \leq s \leq 10^4$). Во второй строке задано натуральное число n — количество видов скворечников ($1 \leq n \leq 10^4$). В третьей строке через пробел заданы целые величины b_i — количество заготовок, требуемых для изготовления того или иного вида скворечников ($1 \leq b_i \leq 10^4$). В четвёртой строке задано целое число m — количество ценовых диапазонов у продавца заготовок ($1 \leq m \leq 1000$). В пятой строке через пробел задано m величин v_j , задающих ценовые диапазоны ($1 \leq v_1 < v_2 < v_3 < \dots < v_m \leq 10^4$). Наконец, в шестой строке через пробел задана $m + 1$ величина p_j — цена одной заготовки в соответствующем диапазоне ($2000 \geq p_1 > p_2 > p_3 > \dots > p_{m+1} \geq 1$).

Формат выхода: Выведите единственное целое число — максимально возможную прибыль (разницу между доходом и расходом) при данной схеме изготовления скворечников.

Пример 1

Вход: Выход:
 3 200 420
 2
 10 12
 3
 10 30 100
 20 10 5 2

Пример 2

Вход: Выход:
 3 200 -120
 2
 10 12
 3
 10 30 100
 80 40 20 8

Примечание: В первом примере изготовление трёх скворечников первого типа требует 30 заготовок, то есть закупка ведётся по второму ценовому диапазону (≤ 30) и за покупку заплатить требуется $30 \cdot 10 = 300$ руб. Если же изготавливаются скворечники второго типа, то требуется 36 заготовок, закупка идет по третьему диапазону и заплатить надо всего $36 \cdot 5 = 180$ руб. Выручка же не зависит от выбора типа скворечников и составляет $3 \cdot 200 = 600$ руб. Стало быть, максимальной будет прибыль при изготовлении скворечников второго типа: $600 - 180 = 420$ руб. Второй пример отличается от первого лишь ценами на заготовки, которые в 4 раза выше. Поэтому оптимальный вариант тот же, только здесь уже прибыль отрицательна.

11.3. «Странная конкатенация». Назовём *обращённым лексикографическим порядком* строк порядок, при котором строка s_1 меньше строки s_2 , $s_1 \prec s_2$, если обращение строки s_1 меньше обращения строки s_2 в обычном лексикографическом порядке (то есть сравниваются строки при прочтении от конца к началу). Напомним, что суть лексикографического сравнения заключается в том, что у двух строк отбрасываются совпадающие начальные части и сравнение первой пары несовпадающих символов определяет результат сравнения строк; если одна из строк является собственным началом другой, то она считается меньше. На входе задано n строк $s_1, s_2, \dots, s_n$, требуется списать их в таком порядке $s_{i_1}s_{i_2}\dots s_{i_n}$, что $s_{i_j} \preceq s_{i_{j+1}}$.

Формат входа: В первой строке задано n — количество входных строк ($1 \leq n \leq 30\,000$). В следующих n строках заданы сами строки; каждая из них состоит только из заглавных символов латиницы и имеет длину от 1 до 255 символов.

Формат выхода: Выдайте единственную строку — результат списывания исходных строк в нужном порядке.

Пример

<u>Вход:</u>	<u>Выход:</u>
3	ADCB ABC ADC
ABC	
ADC	
ADCB	

Примечание: В примере в выходной строке поставлены маленькие пробелы, определяющие границы исходных строк.

11.4. «Хитрое число». Родители подарили Пете Торопыжкину числовой набор, в котором имеются n фишек с цифрой 1, m фишек с цифрой 2 и l фишек с цифрой 3. Петя задумался, какое самое длинное число он сможет составить из этих цифр такое, что между каждой парой двоек найдётся хотя бы одна единица, а между каждой парой троек найдётся хотя бы одна единица и хотя бы одна двойка. Напишите программу, которая будет генерировать такое число.

Формат входа: В единственной строке через пробел заданы три неотрицательных целых числа n, m, l — количество единиц, двоек и троек в Петинском наборе ($0 \leq n, m, l \leq 10^5, n + m + l \geq 1$).

Формат выхода: Выдайте какое-нибудь максимально длинное число, удовлетворяющее условиям задачи.

Пример 1		Пример 2	
<u>Вход:</u>	<u>Выход:</u>	<u>Вход:</u>	<u>Выход:</u>
2 2 2	123123	2 2 4	3123123

11.5. «Минимальный палиндром». На уроке информатики Петя Торопыжкин заинтересовался темой палиндромов (то есть строк, которые одинаково читаются от начала к концу и от конца к началу) и задался вопросом, а как в заданной строке найти неодносимвольную подстроку-палиндром, минимальную в лексикографическом порядке? Напишите программу, которая по заданной входной строке вычисляет такой палиндром. Гарантируется, что в предложенной строке есть хотя бы один неодносимвольный подпалиндром.

Формат входа: В единственной входной строке задана исходная непустая строка, состоящая из заглавных символов латиницы и имеющая длину, не превосходящую 10^4 .

Формат выхода: Выдайте единственную строку — минимальную в лексикографическом порядке неодносимвольную подстроку-палиндром исходной строки.

Пример

<u>Вход:</u>	<u>Выход:</u>
ZABCSBABAZZ	ABA

Муниципальный этап Всероссийской олимпиады
школьников по информатике
в 2016–2017 учебном году
11 класс. Разбор решений и идеи тестов

11.1. «Уловка–22». Для заданного n найдите величину $2^n \bmod 22$.

Задача несложна, однако лобовое решение с возведением двойки в указанную степень и дальнейшим вычислением остатка столкнётся с проблемой переполнения. Такое решение получит половину баллов.

Вычислительный подход спасается тем фактом, что

$$2^n \bmod 22 = ((2^{n-1} \bmod 22) \cdot 2) \bmod 22.$$

То есть, остаток от деления на 22 находим на каждом шаге при вычислении степени.

Ещё более тонкий подход включает в себя математическое размышление о том, что остатки должны циклично повторяться. В самом деле, начальная часть ряда остатков выглядят как

1	2	4	8	16	10	20	18	14	6	12	2	4	8	16	10	20	18	14	6	12	2	4	8	...
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	...

(В нижнем ряду приведены номера чисел в ряду, являющиеся показателем соответствующей степени двойки.) Видно, что в этом ряду имеется цикл длины 10, который начинается со 2-го члена ряда. Соответственно, можно утверждать, что результат есть 1, если $n = 0$, и $2^n \bmod 22 = 2^{(n-1) \bmod 10+1} \bmod 22$, если $n > 0$. А для степеней от 0 до 9 проходит идея с прямым вычислением. Такой подход также хорош тем, что количество вычислений не зависит от величины n : потребуется не более 8 умножений.

Впрочем, в языках, поддерживающих длинную арифметику, — Java, Python, Haskell — возможно и лобовое решение.

Идеи тестов:

1. $n = 0$.
- 2–10. Случайные тесты для $0 < n < 32$.
- 11–20. Случайные тесты для $n > 32$.

11.2. «Готовимся к весне». В школе Пете Торопыжскину дали поручение к весенней ярмарке изготовить N скворечников, которые будут продаваться за s руб. за штуку. В Интернете Петя нашёл чертежи n различных видов скворечников; скворечник i -го вида изготавливается из b_i стандартных деревянных заготовок. Также был найден поставщик таких заготовок, делающий скидки за размер закупки: если покупается не более v_1 заготовок, то их отпускная цена p_1 руб. за штуку; если покупается больше v_1 , но не более v_2 заготовок, то их

отпускная цена $p_2 < p_1$ руб. за штуку; если покупается больше v_2 , но не более v_3 заготовок, то их отпускная цена $p_3 < p_2$ руб. за штуку; и т.д., если покупается более v_m заготовок, то их цена p_{m+1} руб. за штуку. Чтобы сделать процесс изготовления наиболее простым, Петя решил делать скворечники одного вида и покупать ровно столько заготовок, сколько требуется для изготовления нужного числа скворечников. Вопрос, который его интересует: какую максимальную прибыль он может организовать для школы от продажи скворечников?

Задача носит технический характер. Её решение подразумевает использование алгоритма поиска по массиву (для поиска цены, соответствующей требуемому объёму закупок) и поиска минимума в наборе чисел (для поиска оптимального типа скворечников). В целом, алгоритмы стандартные, но аккуратная их реализация требует некоторого мастерства. Общая сложность при лобовом решении есть $O(nm)$ — для каждого вида скворечника надо отыскать ценовой диапазон для требуемого набора заготовок. При имеющихся ограничениях такая сложность удовлетворительна. Впрочем, если использовать идеи двоичного поиска в массиве ценовых планов, то сложность можно сократить до $O(n \log m)$.

Идеи тестов:

1. $n = 1, m = 1$.
2. $n > 1, m = 1$.
3. $n = 1, m = 2$.
4. $n = 1, m = 3$.
5. $n = 1, m$ большое.
6. $n = 2, m = 2$.
7. $n = 2, m = 3$.
8. $n = 2, m$ большое.
9. $n = 3, m = 2$.
10. $n = 3, m = 3$.
11. $n = 3, m$ большое.
12. Максимальный тест: $n = 10^4, m = 3$.
13. Максимальный тест: $n = 3, m = 1000$.
14. Максимальный тест: $n = 10^4, m = 1000$.
- 15–20. Случайные тесты для $n > 3, m > 3$.

Примечание: Во всех тестах величина H достаточно велика, чтобы требовать получения ценового плана, по крайней мере, из середины диапазона, что несколько замедляет вариант с линейным поиском.

11.3. «Странная конкатенация». Назовём обращённым лексикографическим порядком строк порядок, при котором строка s_1 меньше строки s_2 , $s_1 \prec s_2$, если обращение строки s_1 меньше обращения строки s_2 в обычном лексикографическом порядке (то есть сравниваются строки при прочтении от конца к началу).

Напомним, что суть лексикографического сравнения заключается в том, что у двух строк отбрасываются совпадающие начальные части и сравнение первой пары несовпадающих символов определяет результат сравнения строк; если одна из строк является собственным началом другой, то она считается меньше. На входе задано n строк $s_1, s_2, \dots, s_n$, требуется списать их в таком порядке $s_{i_1} s_{i_2} \dots s_{i_n}$, что $s_{i_j} \preceq s_{i_{j+1}}$.

Данная задача, по мнению программного комитета, имеет среднюю сложность. В целом, требуется реализовать свою функцию сравнения строк и с использованием этой функции провести сортировку имеющегося набора строк, после чего выдать их в выходной поток. В имеющихся ограничениях данный прямолинейный метод укладывается в ограничения по времени.

Однако если строки в наборе имеют весьма большую длину (миллионы и более символов) и среди них имеются наборы, в рамках каждого из которых строки имеют общее начало, то можно применять следующую идею. Во-первых, использовать сортировку Бентли–Седжвика, в которой в отличие от QuickSort (сортировки Хоара) сортируемый набор делится не на две части, а на три: элементы, меньшие выбранного, равные выбранному и большие выбранного. Во-вторых, проводить не полную лексикографическую проверку, а только на основании некоторого количества первых символов строк. При рекурсивном вызове наборы из меньших и больших элементов также сравниваются с начала, но в наборе, содержащем строки с одинаковым началом сравнение производится не с начала строк, а после этой общей части. Сложность такого алгоритма такая же, $O(n \log n)$ — но лучше константа, поскольку функция сравнения более быстрая.

Идеи тестов:

1. $n = 1$, односимвольная строка.
2. $n = 1$, строка средней длины.
3. $n = 1$, строка длины 255.
4. $n = 2$, две односимвольных строки.
5. $n = 2$, две строки средней длины.
6. $n = 2$, две строки длины 255.
7. Максимальный тест, $n = 30000$, односимвольные строки.
8. Максимальный тест, $n = 30000$, строки средней длины.
9. Максимальный тест, $n = 30000$, строки длины 255.
- 10–20. Случайные тесты.

11.4. «Хитрое число». Родители подарили Пете Торпыжскину числовой набор, в котором имеются n фишек с цифрой 1, m фишек с цифрой 2 и l фишек с цифрой 3. Петя задумался, какое самое длинное число он сможет составить из этих цифр такое, что между каждой парой двоек найдётся хотя бы одна единица, а между каждой парой троек найдётся хотя бы одна единица и хо-

тя бы одна двойка. Напишите программу, которая будет генерировать такое число.

Одно из решений данной задачи является идейным: в его основе лежат размышления о структуре числа, удовлетворяющего приведённым требованиям. Технически же реализация придуманного правила построения числа не слишком сложна. Другой вариант решения привлекает принцип динамического программирования.

Обсудим идейное решение. Полезные наблюдения:

1°. Если у нас в наборе нет ни двоек, ни троек, то все имеющиеся единицы можно выставить друг за другом: их расположение ничто не ограничивает.

2°. Если у нас имеются только двойки и единицы, то наибольшее число двоек, которые можно выставить, превосходит на 1 количество единиц: $21212 \dots 212$. Соответственно, если n меньше, чем $m - 1$, то не все двойки можно выставить.

3°. Аналогично наблюдению 2° число троек не может превосходить числа двоек более, чем на 1.

4°. Если выставлены две тройки $3 \dots 3$, то локально возможные конфигурации могут выглядеть как 3123 , или 3213 , или 31213 , или 32123 . При этом первые три конфигурации могут циклически продолжаться:

$3213213213 \dots$, $312131213121 \dots$ и $312131213121 \dots$

Но четвертая конфигурация так продолжена быть не может: будут идти две двойки, между которыми нет единицы: $32123212 \dots$. То есть, если воспроизводить четвёртую комбинацию, то надо писать что-то вроде

$321213212312123 \dots$

Но такие комбинации не увеличивают возможное количество троек в числе.

Из первых двух комбинаций для повторения лучше выбрать первую: при исчерпании троек она позволяет продолжить написание двоек без добавления единицы.

Итак, алгоритм решения следующий:

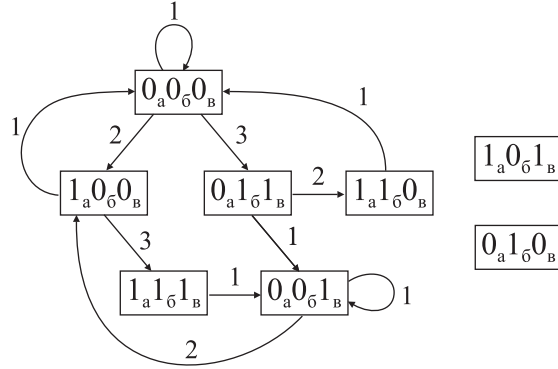
1. Если есть тройки напомним 3. Затем, пока имеются тройки, двойки и единицы, приписываем комбинацию 123.
2. Если остались тройки, конец, максимально возможное число написано.
3. Пусть троек не осталось. Если есть двойки, допишем 2.
4. Пока имеются двойки и единицы, дописываем комбинацию 12.
5. Если единиц не осталось, максимальное число выписано.
6. Если единицы остались, дописываем их все.

Идея динамического программирования заключается в том, что станем рассматривать всевозможные способы приписывания очередной цифры в конец числа, контролируя, какие цифры можно приписать и сколько каких цифр остаётся. Текущее состояние приписывания имеет три показателя:

- а) стоит ли после последней двойки единица;

- б) стоит ли после последней тройки единица;
 в) стоит ли после последней тройки двойка.

Кодировать состояние можно тремя битами: бит_а бит_б бит_в, то есть числами от 0 до 7. В каждом состоянии можно дописать единицу с соответствующим изменением состояния. В состояниях, где бит_а равен нулю, то есть в состояниях 0, 1, 2, 3, можно также дописать двойку. В состояниях, где сброшены бит_б и бит_в, то есть в состояниях 0 и 4, можно также дописывать тройку. Схема возможных изменений состояния приведена на рисунке:



Из схемы видно, что ребра с меткой 1 выходят из всех положений, с меткой 2 — только из положений с нулевым битом_а, с меткой 3 — только из положений с нулевыми битом_б и битом_в. Соответственно, ребра с меткой 1 приходят в положение, где сброшены бит_а и бит_б, с меткой 2 — где сброшен бит_в, но выставлен бит_а, а с меткой 3 — где выставлены бит_б и бит_в. При такой схеме переходов недостижимы два состояния, выходы из них не обозначены — это не требуется. Однако эти состояния разумно оставить для удобства индексации.

Для реализации проведения процесса дописывания цифр создадим двумерный массив `ar[1..n+m+l, 0..7]`. Первый индекс в массиве — текущая длина числа, второй — текущее состояние процесса приписывания. В ячейках хранятся пять величин: `lastDigit` — последняя приписанная цифра, `lastState` — каково было состояние до приписывания последней цифры (нужно для генерации числа после окончания процесса приписывания), `d1` — оставшееся количество единиц, `d2` — оставшееся количество двоек, `d3` — оставшееся количество троек.

Изначально во всех ячейках `lastDigit` равен 0, символизируя, что данная ячейка ещё не обрабатывалась. В качестве начальных условий при наличии в наборе хотя бы одной соответствующей цифры полагаем:

```
ar[1,0] = {lastDigit = 1, d1 = n - 1, d2 = m, d3 = l};
ar[1,4] = {lastDigit = 2, d1 = n, d2 = m - 1, d3 = l};
ar[1,3] = {lastDigit = 3, d1 = n, d2 = m, d3 = l - 1}.
```

Для дальнейшей обработки массива введём флаг `haveAdded`, значение `true` которого говорит, что к какому-то из чисел меньшей длины была добавлена ещё одна цифра и есть возможность увеличивать число дальше, и счётчик `i` — текущую длину числа. Обработка массива для конкретного `i` заключается в проверке возможности каждого из одиннадцати переходов, соответствующих одиннадцати

стрелкам на схеме, и реализации возможных из них. До начала обработки флаг **haveAdded** устанавливается в **false**; если какой-то из переходов осуществлен, он выставляется в **true**.

Обработка очередного перехода заключается в том, что достигнуто ли состояние, из которого выходит этот переход, и имеются ли ещё цифры, соответствующие этому переходу:

```
if (ar[i,s].lastDigit <> 0) and (ar[i,s].{d1,d2,d3} > 0) then ...
```

(выбирается поле, соответствующее цифре перехода).

Однако тут встаёт вопрос корректности этой процедуры, поскольку в одно и то же состояние при заданной длине числа можно прийти, приписывая разные последовательности цифр. Например, последовательности 1 1 1, ..., 1 2 1 ... и 3 2 1 ... после приписывания третьей цифры (при условии достаточного запаса цифр) приведут в состояние 0, однако дадут разное количество оставшихся цифр различных номиналов. Какую информацию записывать?

Эта проблема решается «жадным» приписыванием: из всех состояний мы выбираем то, в котором меньше осталось троек, а при равенстве количества троек то, в котором меньше двоек. Соответственно, для заполнения разумно нового состояния разумно написать небольшую процедуру:

```
writeNewState (iNew, stateNew, stateOld, d1New, d2New, d3New, writtenDigit)
  if (ar[iNew,stateNew].lastDigit = 0) (* если это состояние ещё не достигалось ... *)
    or ((ar[iNew,stateNew].d3 > d3New) or
        ((ar[iNew,stateNew].d3 = d3New) and (ar[iNew,stateNew].d2 > d2New))) then
      (* ... или состояние уже записывалось, но новая идея лучше,
        тогда записываем *)
      ar[iNew,stateNew].lastDigit := writtenDigit;
      ar[iNew,stateNew].lastState := stateOld;
      ar[iNew,stateNew].d1 := d1New;
      ar[iNew,stateNew].d2 := d2New;
      ar[iNew,stateNew].d3 := d3New;
end;
```

«Жадный» подход обуславливается тем, что исписав единицы, мы потеряем возможность писать тройки и двойки, и так же исписав двойки, потеряем возможность писать тройки. Поэтому в первую очередь надо писать цифры так, чтобы уменьшить количество троек и двоек.

Процедура обработки состояний заканчивается, когда либо достигнут конец массива, то есть записаны все имеющиеся цифры, либо когда не совершено приписывания ни по одному из вариантов (флаг **haveAdded** к концу обработки текущей длины числа равен **false**).

После окончания процедуры приписывания формируем число: в последней обработанной колонке ищем состояние, у которого **lastDigit** отлична от 0, и передвигаемся в направлении уменьшения длины числа, используя поле **lastState** и собирая число из отдельных цифр.

Основной вопрос при рассмотрении технического решения, опирающегося на

принцип динамического программирования (ДП) — а зачем так сложно, если решение, основанное на обдумывании ситуации, куда как короче и проще в реализации? Ответ состоит в том, что подход, основанный на ДП стандартен, мы не обдумываем глобальную оптимальность решения, а используем оптимальность локальную. Но, конечно, зачастую проигрываем в накладных расходах — в сложности написания программы, в потребной памяти и, возможно, в скорости исполнения. Выбор конкретного пути решения, конечно, остаётся за участником олимпиады.

Идеи тестов:

1. Одна единица.
2. Одна двойка.
3. Одна тройка.
4. Только единицы.
5. Только двойки.
6. Только тройки.
7. Единицы и двойки, единиц значительно больше, чем двоек.
8. Единицы и двойки, единиц на одну больше, чем двоек.
9. Единицы и двойки, единиц столько же, сколько двоек.
10. Единицы и двойки, единиц на одну меньше, чем двоек.
11. Единицы и двойки, единиц недостаточно, чтобы написать все двойки.
12. Тройки и одна единица.
13. Тройки и много единиц.
14. Тройки и одна двойка.
15. Тройки и много двоек.
16. Тройки и по одной двойке и единице.
17. Тройки, двойки и одна единица.
18. Максимальный тест: $n = 100000$, $m = 0$, $l = 0$.
19. Максимальный тест: $n = 0$, $m = 100000$, $l = 0$.
20. Максимальный тест: $n = 100000$, $m = 100000$, $l = 0$.
21. Максимальный тест: $n = 0$, $m = 0$, $l = 100000$.
22. Максимальный тест: $n = 100000$, $m = 0$, $l = 100000$.
23. Максимальный тест: $n = 0$, $m = 100000$, $l = 100000$.
24. Максимальный тест: $n = 100000$, $m = 100000$, $l = 100000$.
- 25–50. Случайные тесты.

11.5. «Минимальный палиндром». На уроке информатики Петя Торопыжкин заинтересовался темой палиндромов (то есть строк, которые одинаково читаются от начала к концу и от конца к началу) и задался вопросом, а как в заданной строке найти неодносимвольную подстроку-палиндром, минимальную в лексикографическом порядке? Напишите программу, которая по заданной

входной строке вычисляет такой палиндром. Гарантируется, что в предложенной строке есть хотя бы один неодносимвольный подпалиндром.

Данная задача является достаточно сложной, поскольку использует не вполне тривиальную идею о правильной работе с подстроками.

Лобовое решение выглядит следующим образом: перебираем все подстроки, каждую очередную подстроку проверяем на симметричность, среди выявленных палиндромов ищем лексикографически минимальный. Однако сложность такого алгоритма в худшем случае $O(n^4)$. Действительно: перебор подстрок — $O(n^2)$, поскольку перебираем индексы начала подстроки и всевозможные длины; для каждой подстроки проверка на симметричность — $O(n)$; для каждого найденного палиндрома сравнение с текущим минимумом — ещё $O(n)$. Однако реально данный алгоритм работает много быстрее, поскольку чаще всего проверка на симметричность находит несопадающую пару символов не к концу проверки, а после нескольких первых пар. Так же лексикографическое сравнение двух строк чаще всего не требует пройти по всей длине строки, несопадающие символы, определяющие порядок, обнаруживаются очень быстро. Худший случай для такого алгоритма — строки, состоящие из малого количества различных символов, в наихудшем варианте — из одного символа.

Алгоритм, имеющий гарантированно лучшую сложность $O(n^2)$ выглядит следующим образом:

- 1) для каждой позиции в строке выясняем длину кратчайшего палиндрома, начинающегося в этой позиции, или выясняем, что в этой позиции не начинается ни одного палиндрома;
- 2) этих найденных палиндромов находим лексикографически минимальный классическим алгоритмом поиска экстремума, в котором используем лобовое лексикографическое сравнение.

Сложность второго пункта — очевидно $O(n)$: из n строк длиной до n символов каждой нужно выбрать минимальную, что потребует $n - 1$ сравнение по n сравнений символов каждое.

Оказывается, что и первый пункт алгоритма можно реализовать со сложностью $O(n^2)$. Для этого перебираем возможные места центров палиндромов — позиции со второй по $(n - 1)$ -ю — и последовательно увеличивая длину, проверяем на совпадение символы, расположенные симметрично от центра. Пока имеет место совпадение — мы нашли какой-то палиндром с центром в выбранном месте, находим позицию, с которой он начинается и при необходимости корректируем данные о том, какой самый короткий палиндром там начинается. Как только найдена пара несопадающих символов, расположенных симметрично относительно данного центра, работа с данным центром прекращается — более длинные строки гарантированно палиндромами не являются. В целом, перебор имеет сложность $O(n^2)$: $2n - 1$ возможных позиций центра ($n - 2$ символов и $n - 1$ позиций между символами) и для каждой позиции до $n/2$ сравнений пар символов.

Во всех предлагаемых категориях тестов имеются строки, содержащие длинные подстроки из одинаковых символов, в том числе и строки, целиком состоящие из одинаковых символов.

Идеи тестов:

- 1–15. Тесты с длиной строки от 10 до 100.
- 16–30. Тесты с длиной строки от 100 до 1000.
- 31–45. Тесты с длиной строки от 1000 до 10000.
- 46–50. Максимальные тесты с длиной строки 10000.