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

10 класс

Время выполнения задач — 4 часа

Ограничение по времени — 2 секунды на тест

Ограничение по памяти — 64 мегабайта

10.1. «Сумма последовательности». Формула общего члена последовательности —

$$a_n = \texttt{round}\left(\left|10 \cdot \sin(n^2 + 2n + 5)\right|\right).$$

Здесь аргумент синуса подразумевается в радианах, $|\cdot|$ — функция модуля числа, $\texttt{round}(\cdot)$ — функция округления. Для заданного натурального числа S найдите минимальный номер k такой, что сумма последовательности $a_1 + a_2 + \ldots + a_k$ превзойдёт S.

Формат входа: На входе задаётся единственное натуральное число — величина S ($1 \leqslant S \leqslant 1000$).

Формат выхода: Выдайте единственное натуральное число k — искомый индекс.

Пример

Вход:	*Выход:*
22	3

Примечание: Указанная последовательность имеет вид 10, 4, 9, 7, ..., поэтому сумма первых трёх её членов превзойдёт 22.

10.2. «Вариация массива». Петя Торопыжкин назвал *вариацией массива* следующую операцию: массив сортируется по возрастанию элементов, затем находится разность второго и первого элементов отсортированного массива, затем третьего и второго, затем четвёртого и третьего, и т.д. до разности последнего и предпоследнего элементов. После все найденные разности суммируются. Помогите Пете, написав программу, которая проделывает указанную операцию над заданным массивом.

Формат входа: В первой строке задано целое число n — количество элементов в массиве ($2 \leqslant n \leqslant 1000$). В следующей строке через пробел задано n целых чисел, каждое по модулю не превосходит 10^6.

Формат выхода: Выведите единственное целое число — вариацию заданного набора чисел.

Пример

Вход:	*Выход:*
4 5 2 6 1	5

Примечание: Результат получается следующим образом: отсортированный массив: `1 2 5 6`; вычисляем разности: $2-1=1$, $5-2=3$, $6-5=1$; вычисляем сумму разностей: $1+3+1=5$.

10.3. «Космическая стратегия». В одной космической стратегической игре, которая нравится Пете Торопыжкину, производство на планете зависит от величины её населения. Начав новую партию, Петя собрал информацию о численности населения каждой из планет, присутствующих в игре, и хочет выяснить, какая численность населения является наиболее распространённой в игровой Вселенной (чтобы знать, на какой уровень производства можно рассчитывать в среднем). Помогите ему, напишите соответствующую программу.

Формат входа: В первой строке задано целое число n — количество планет в игровой Вселенной ($1 \leqslant n \leqslant 10^5$). В следующей строке через пробел в каком-то порядке задано n натуральных чисел, не превосходящих 10^9 — населения планет.

Формат выхода: Выведите единственное натуральное число — численность населения, наиболее часто встречающуюся в текущей Вселенной. Если таких численностей несколько, выведите меньшую из них (чтобы Пете готовиться к худшему из возможных вариантов).

Пример 1

Вход:	*Выход:*
`4` `100 50 100 5`	`100`

Пример 2

Вход:	*Выход:*
`4` `100 50 100 50`	`50`

10.4. «Корень сравнения». На математическом кружке Петя Торопыжкин изучил тему «Сравнения». В теории чисел *алгебраическим сравнением по модулю* p (p — натуральное число) называют выражение вида

$$a_n x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_2 x^2 + a_1 x + a_0 \equiv 0 \pmod{p},$$

то есть выражение, в левой части которого стоит многочлен с целыми коэффициентами. Корнем сравнения называют такое целое число, которое, будучи подставленным в левую часть, даёт значение, *сравнимое с нулём по модулю* p, то есть дающее остаток 0 при делении на p. Понятно, что если сравнение имеет один корень x_0, то корнями являются также числа $x_0 + k \cdot p$, k — любое целое число.

Пете задали несколько сравнений на дом, и он решил все задачи, кроме одной, в которой он несколько коэффициентов многочлена в левой части записал неразборчиво из-за кончающейся ручки. Тогда он задался вопросом: каковы могут быть значения этих коэффициентов, чтобы число $p-1$ было корнем этого сравнения? Опять понятно, что если какое-то число a_i является ответом к задаче, которую поставил себе Петя, то числа $a_i + k \cdot p$ тоже будут ответами. Поэтому Петя хочет найти ответ, в котором все неопределённые коэффициенты лежат в диапазоне от 0 до $p-1$ включительно. Помогите ему, напишите соответствующую программу. Заметим, что такая задача всегда имеет решение, если хотя бы

один из коэффициентов неопределён. Также считаем, что старший коэффициент заведомо является определённым.

Формат входа: В первой строке перечислены три натуральных числа: n — степень сравнения (степень многочлена в левой части), k — количество неопределённых коэффициентов — и p — модуль сравнения ($1 \leqslant n \leqslant 1000$, $1 \leqslant k \leqslant n$, $2 \leqslant p \leqslant 10^6$). В следующей строке через пробел в порядке убывания перечислены степени переменной x, при которых коэффициенты не определены — целые числа из диапазона от 0 до $n-1$. В третьей строке через пробел перечислены $(n+1-k)$ чисел — заданные коэффициенты в порядке убывания степеней, целые числа, по модулю не превышающие 10^6.

Формат выхода: Выдайте набор из k целых чисел, каждое из диапазона от 0 до $p-1$, таких, что при их подстановке в сравнение вместо неопределённых коэффициентов (в порядке убывания степеней) полученное сравнение будет иметь корень $p-1$. Если ответов несколько, выдайте любой из них.

Пример

Вход:	*Выход:*
3 2 5 2 0 4 -10	0 4

Примечание: В примере задано сравнение $4x^3 + ?x^2 - 10x + ? \equiv 0 \pmod 5$ (? — неопределённые коэффициенты). Подстановка вместо неопределённых коэффициентов чисел из предлагаемого ответа (0 в коэффициент у x^2, 4 вместо свободного члена) даёт сравнение $4x^3 - 10x + 4 \equiv 0 \pmod 5$, которое действительно имеет корень $4 = 5 - 1$: $4 \cdot 4^3 - 10 \cdot 4 + 4 = 256 - 40 + 4 = 220 \equiv 0 \pmod 5$.

10.5. «Странное сравнение строк». Петя Торопыжкин решил придумать новый способ сравнения строк: фиксируется строка-ключ, и рассматривается количество вхождений сравниваемых строк в строку-ключ. Если эти количества не равны, то они определяют порядок строк, если же они равны, то строки сравниваются в лексикографическом порядке. Петя решил написать программу, которая для заданной строки-ключа находит наибольшую строку из заданного набора в смысле придуманного им порядка. Помогите ему в написании такой программы.

Формат входа: В первой строке задана строка-ключ. Во второй строке задано целое число n — количество строк в наборе ($1 \leqslant n \leqslant 10000$). В следующих n строках заданы строки из набора. Каждая строка (из набора и строка-ключ) задаётся в виде `длина␣символы`. Все строки непусты, состоят из заглавных букв латиницы. Строка-ключ имеет длину не более 10000 символов, строки из набора — не более 100 символов. Все строки из набора попарно различны.

Формат выхода: Выдайте единственную строку — наибольшую в смысле предложенного Петей порядка.

Пример 1

Вход:	*Выход:*
8 ABCABCAB 3 2 AB 3 BCD 2 BC	AB

Пример 2

Вход:	*Выход:*
9 ABCABCABC 3 2 AB 3 BCD 2 BC	BC

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

10.1. «Сумма последовательности». *Формула общего члена последовательности —*

$$a_n = \texttt{round}\left(\left|10 \cdot \sin(n^2 + 2n + 5)\right|\right).$$

Здесь аргумент синуса подразумевается в радианах, $|\cdot|$ — функция модуля числа, $\texttt{round}(\cdot)$ *— функция округления. Для заданного натурального числа S найдите минимальный номер k такой, что сумма последовательности $a_1 + a_2 + \ldots + a_k$ превзойдёт S.*

Задача имеет статус утешительной и подразумевает прямолинейное суммирование слагаемых, вычисляемых по заданной формуле. Некоторая сложность состоит в том, что язык, используемый участником, не поддерживает операцию округления. Её замена: $\texttt{round}(x) = [x + 0.5]$, где $[\,\cdot\,]$ — операция отбрасывания целой части.

Идеи тестов:

1. Минимальный тест: $S = 1$.
2. Минимальный тест: $S = 9$.

3–10. Случайные тесты для S в диапазоне от 50 до 1000.

10.2. «Вариация массива». *Петя Торопыжкин назвал вариацией массива следующую операцию: массив сортируется по возрастанию элементов, затем находится разность второго и первого элементов отсортированного массива, затем третьего и второго, затем четвёртого и третьего, и т.д. до разности последнего и предпоследнего элементов. После все найденные разности суммируются. Помогите Пете, написав программу, которая проделывает указанную операцию над заданным массивом.*

Данная задача представляет два различных пути решения, один из которых весьма прост в реализации. Прямолинейное решение подразумевает считывание элементов массива, его сортировку, а затем нахождение и суммирование разностей.

Однако, если задуматься о сути проделываемой операции, то можно понять, что искомый результат можно найти без применения операции сортировки. Действительно, пусть a_1, a_2, …, a_n — последовательность элементов в массиве, отсортированном по возрастанию. При этом a_1 — минимальный элемент, а a_n — максимальный. Тогда искомая величина есть

$$\begin{aligned} S &= (a_2 - a_1) + (a_3 - a_2) + (a_4 - a_3) + \ldots + (a_{n-1} - a_{n-2}) + (a_n - a_{n-1}) = \\ &= a_2 - a_1 + a_3 - a_2 + a_4 - a_3 + \ldots + a_{n-1} - a_{n-2} + a_n - a_{n-1} = a_n - a_1. \end{aligned}$$

Последнее равенство получаем потому, что в выражении в последней строке все слагаемые, кроме a_1 и a_n, встречаются дважды с разными знаками и взаимно уничтожаются.

Стало быть, для получения искомой величины надо найти разность максимального и минимального элементов из заданного набора, которые ищутся стандартными алгоритмами без сортировки массива.

Идеи тестов:

1. Все элементы набора одинаковы.
2–6. Случайные тесты, где величины a_i по модулю не превосходят 1000 и все промежуточные результаты при прямолинейном вычислении гарантированно входят в тип short int.
7–10. Случайные тесты, где величины a_i большие и промежуточные вычисления могут не войти в тип short int.

10.3. «Космическая стратегия». *В одной космической стратегической игре, которая нравится Пете Торопыжкину, производство на планете зависит от величины её населения. Начав новую партию, Петя собрал информацию о численности населения каждой из планет, присутствующих в игре, и хочет выяснить, какая численность населения является наиболее распространённой в игровой Вселенной (чтобы знать, на какой уровень производства можно рассчитывать в среднем). Помогите ему, напишите соответствующую программу.*

Данная задача предполагается имеющей среднюю сложность. Фактически, требуется найти элемент, входящий в данный набор максимальное количество раз.

Прямолинейное решение, основанное на подсчёте количества вхождений каждого числа в заданный набор, может потребовать либо слишком большой памяти (массив на 10^9 элементов), либо слишком большого времени — если последовательно подсчитывать количество вхождений 1, затем 2, затем 3 и т.д. Впрочем, этот путь можно реализовать при помощи специальной структуры данных — словаря (ассоциативного массива). Он позволит хранить записи количеств не всех чисел от 1 до 10^9, а лишь тех, которые реально присутствуют в наборе. Количество их не превосходит размера набора, то есть 10^5, что и уложится в допустимую память, и будет достаточно быстрым по времени. Однако такая структура данных не присутствует в библиотеке языка Паскаль в средах Delphi и FreePascal/Lazarus, а её самостоятельное написание составляет определённую сложность. Однако словарь присутствует в библиотеке языков Pascal.ABC, C++, Java и Python.

Альтернативный путь решения подразумевает предварительную сортировку массива по возрастанию, что приведёт к группировке одинаковых элементов — они будут идти в массиве подряд. Пусть a_i — i-й элемент в отсортированном массиве, $1 \leqslant i \leqslant n$. Затем нужно найти элемент, группа которого имеет наибольшую длину. Для реализации этого алгоритма будем хранить величины `elMax` —

величину элемента, максимально входящего при обработке массива до текущей позиции, и `qntMax` — количество его вхождений. Вначале положим `elMax` $= 0$, `qntMax` $= -1$. Кроме того, понадобится длина текущей обрабатываемой группы `qnt`; вначале положим эту величину равной 1 (первый элемент массива даёт группу длины, по крайней мере, 1). Затем организуем цикл по счётчику i от 2 до n. (Если в наборе всего один элемент. то есть если $n = 1$, ответ тривиален.) Если $a_i = a_{i-1}$, то мы находимся в рамках группы одного элемента, увеличиваем `qnt` на 1. В противном случае `qnt` есть длина группы элемента a_{i-1}; сравниваем `qnt` и `qntMax`; если `qnt` $>$ `qntMax`, то присваиваем `qntMax` $=$ `qnt`, `elMax` $= a_{i-1}$; затем присваиваем `qnt` $= 1$ — длина текущей отсмотренной части группы элемента a_i. Заметим, что строгое сравнение `qnt` и `qntMax` гарантирует сохранение меньшего элемента в качестве ответа при равенстве количеств вхождений, поскольку меньший элемент будет обработан раньше. После окончания цикла надо аналогично сравнить накопленное значение `qnt`, которое описывает количество вхождений элемента a_n.

Идеи тестов:

1. Минимальны тест: $n = 1$.
2–11. Случайные тесты, элемент с максимальным количеством вхождений единственен. Имеется максимальный тест $n = 10^5$.
12–20. Случайные тесты, несколько различных элементов с одинаковым максимальным количеством вхождений. Имеется максимальный тест $n = 10^5$.

10.4. «Корень сравнения». *На математическом кружке Петя Торопыжкин изучил тему «Сравнения». В теории чисел алгебраическим сравнением по модулю p (p — натуральное число) называют выражение вида*

$$a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \ldots + a_2 x^2 + a_1 x + a_0 \equiv 0 \ (mod p),$$

то есть выражение, в левой части которого стоит многочлен с целыми коэффициентами. Корнем сравнения называют такое целое число, которое, будучи подставленным в левую часть, даёт значение, сравнимое с нулём по модулю p, то есть дающее остаток 0 при делении на p. Понятно, что если сравнение имеет один корень x_0, то корнями являются также числа $x_0 + k \cdot p$, k — любое целое число.

Пете задали несколько сравнений на дом, и он решил все задачи, кроме одной, в которой он несколько коэффициентов многочлена в левой части записал неразборчиво из-за кончающейся ручки. Тогда он задался вопросом: каковы могут быть значения этих коэффициентов, чтобы число $p-1$ было корнем этого сравнения? Опять понятно, что если какое-то число a_i является ответом к задаче, которую поставил себе Петя, то числа $a_i + k \cdot p$ тоже будут ответами. Поэтому Петя хочет найти ответ, в котором все неопределённые коэффициенты лежат в диапазоне от 0 до $p-1$ включительно. Помогите ему, напишите

соответствующую программу. Заметим, что такая задача всегда имеет решение, если хотя бы один из коэффициентов неопределён. Также считаем, что старший коэффициент заведомо является определённым.

Решение данной задачи технически не очень сложно, но для формулировки корректного алгоритма требуются определённые размышления из области теории чисел.

В описании решения подразумевается, что все вычисления проделываются по модулю p. При этом следует учесть, что чаще всего в языках программирования функция взятия остатка реализована таким образом, что выдаёт результат, имеющий знак делимого, в то время как традиционное математическое определение этой операции подразумевает неотрицательный результат в диапазоне от 0 до величины, на 1 меньшей делителя. Собственно, коррекция результата весьма проста: если результат получился отрицательным, к нему надо прибавить делитель.

Решение базируется на следующих наблюдениях.

1. Если величина d взаимно проста с p (то есть НОД$(d, p) = 1$), то значение выражения $(d \cdot x) \bmod p$ при переборе значений x от до $p - 1$ даёт все значения из диапазона от 0 до $p - 1$. Это обеспечит существование решения.

2. Величина $p-1$, будучи возведённой в любую неотрицательную степень, взаимно проста с p.

Отсюда путь решения заключается в следующем. Полагаем равными нулю все коэффициенты кроме одного. Пусть это будет коэффициент при некотором слагаемом с x^k, $k \geqslant 0$. Вычисляем сумму s всех слагаемых с определёнными уже коэффициентами. Вычисляем $d = (p-1)^k \bmod p$. В результате имеем сравнение $s + a_k \cdot d \equiv 0 \pmod p$, которое нужно решить относительно коэффициента a_k.

Некоторое рассуждение относительно вычисления s и d. Потенциально нам нужно умножать два числа, модули которых заключены в диапазоне от 0 до $p-1$, то есть в худшем случае до 10^6: при вычислении $(p-1)^j$ надо перемножать величины $p-1$, а потом ещё умножать на коэффициент. Если делать это прямолинейно, то возможно переполнение типа int: произведение может иметь модуль до 10^{12}, которое только потом с использованием операции остатка переводится в диапазон от 0 до $p-1$. Для избежания этого можно использовать более значащий тип int64 или (для тех, кто пишет на Java или Python) применять длинную арифметику. Однако следующее математическое соображение упростит этот момент. Величина $(p-1)$ сравнима с (-1) по модулю p, значит, $(p-1)^2$ сравнима с 1, $(p-1)^3$ снова сравнима с (-1) и т.д. Вообще, $(p-1)^{2k+1} \equiv (-1) \pmod p$, $(p-1)^{2k} \equiv 1 \pmod p$. То есть вычисление остатков от деления $(p-1)^i$ на p вообще не требует умножений, а умножение $(p-1)^i$ на коэффициент не увеличивает разрядность результат и заключается лишь, возможно, в смене знака коэффициента.

После вычисления s и d имеем сравнение $s + a_k \cdot (-1)^{k \bmod 2} \equiv 0 \pmod p$. Откуда $a_k = (-1)^{k \bmod 2+1} \cdot s$. Последнюю величину нужно привести к диапазону от 0 до $p-1$.

Идеи тестов:

1. Линейное сравнение, не определён свободный член, $p = 10$.
2. Линейное сравнение, не определён свободный член, $p = 10^6$.
3. Квадратное сравнение, не определён свободный член, $p = 10$.
4. Квадратное сравнение, не определён второй коэффициент, $p = 10$.
5. Квадратное сравнение, не определены второй коэффициент и свободный член, $p = 10$.

6–8. То же, что в тестах 3–5, только $p = 10^6$.

9. Максимальный тест: $n = 1000$, коэффициенты ± 999999, не определён свободный член, $p = 10^6$.
10. Максимальный тест: $n = 1000$, коэффициенты $\pm 10^6$, не определён свободный член, $p = 10^6$.
11. Максимальный тест: $n = 1000$, коэффициенты ± 999999, не определено около половины коэффициентов, $p = 10^6$.
12. Максимальный тест: $n = 1000$, коэффициенты $\pm 10^6$, не определено около половины коэффициентов, $p = 10^6$.
13. Максимальный тест: $n = 1000$, коэффициенты ± 999999, не определены все коэффициенты кроме старшего, $p = 10^6$.
14. Максимальный тест: $n = 1000$, коэффициенты $\pm 10^6$, не определены все коэффициенты кроме старшего, $p = 10^6$.

15–20. Случайные тесты: $n \in [10, 20]$, модули всех определённых коэффициентов из диапазона $[-30, 30]$, $p \in [10, 100]$.

21–25. Случайные тесты: $n \in [900, 1000]$, модули всех определённых коэффициентов из диапазона $[-10^6, 10^6]$, $p \in [900000, 10^6]$.

10.5. «Странное сравнение строк». *Петя Торопыжкин решил придумать новый способ сравнения строк: фиксируется строка-ключ, и рассматривается количество вхождений сравниваемых строк в строку-ключ. Если эти количества не равны, то они определяют порядок строк, если же они равны, то строки сравниваются в лексикографическом порядке. Петя решил написать программу, которая для заданной строки-ключа находит наибольшую строку из заданного набора в смысле придуманного им порядка. Помогите ему в написании такой программы.*

По мнению программного комитета, данная задача имеет высокую сложность, поскольку требует как достаточно хорошей техники программирования даже для реализации частичного решения, так и знания алгоритмов работы со строками.

В основе решения задачи лежит алгоритм поиска максимума среди заданного набора элементов. Идея его проста: в качестве начального значения максимума положим первый элемент набора. Если очередной считанный элемент больше текущего максимума, запоминаем его в качестве нового найденного максимума. Проблема состоит в том, что функция сравнения требует вычислить количество

вхождений сравниваемых строк в строку-ключ. Для текущего максимума эта информация запомнена, а вот для каждой новой строки эту величину требуется посчитать.

Обозначим через S строку-ключ, а через T — очередную обрабатываемую строку из набора. При реализации простейшего подхода к подсчёту количества вхождений строки T в строку S алгоритм перебирает все позиции строки S и смотрит, не входит ли T в S, начиная с этой позиции. То есть для каждой позиции в S нам нужно сделать сравнений символов в худшем случае столько, какова длина T. Стало быть, сложность обработки одной строки T есть $O(|S| \cdot |T|)$ (модуль означает длину строки). В целом же для обработки набора из n строк потребуется время $O(n \cdot |S| \cdot |T|)$, что для имеющихся ограничений слишком долго.

Для быстрого подсчёта количества (и поиска) вхождений строки T в строку S за время $O(|S| + |T|)$ может быть применён алгоритм Кнута – Морриса – Пратта, основывающийся на понятии *префикс-функции* строки. Значение префикс-функции $\pi(D, i)$ для строки D и индекса $i = 2, \ldots, |D|$, есть наибольшая длина начальной части подстроки $D(1\,..\,i)$ (то есть *префикса* строки $D(1\,..\,i)$), которая является концом (*суффиксом*) этой подстроки (так называемый *префикс-суффикс*). Иногда для удобства считают $\pi(D, 1) = 0$. В качестве классического примера рассматривают префикс-функцию для строки $D = ABCDABSCABCDABIA$ (длина которой равна 16):

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$\pi(D, i)$	0	0	0	0	1	2	0	0	1	2	3	4	5	6	0	1

Действительно, для значения индекса $i = 3$ рассматривается префикс ABC, не имеющий непустого префикс-суффикса. А для значения индекса $i = 12$ рассматривается префикс $ABCDABSCABCD$, который имеет префикс-суффикс $ABCD$ длины 4.

При значении i, равном длине $|D|$ строки D, рассматривают значение $\pi(D, |D|)$, меньшее $|D|$.

Алгоритм вычисления набора значений префикс-функции для всех возможных значений индекса i линеен по размеру строки и основан на принципе днамического программирования. Дальше символ d_i будет обозначать i-й символ строки D. База метода имеется: $\pi(D, 1) = 0$. Пусть $\pi(D, i) = k$. Если $d_{i+1} = d_{k+1}$ (то есть $(i+1)$-й символ D совпадает с $(k+1)$-м символом), то имеется префикс-суффикс длины $k + 1$, положим $\pi(D, i + 1) = k + 1$. Пусть теперь $d_{i+1} \neq d_{k+1}$. Имеем следующую схему:

$$D = \underline{d_1\, d_2\, \ldots\, d_{k-1}\, d_k}\, d_{k+1}\, \ldots\, d_{i-k}\, \underline{d_{i-k+1}\, d_{i-k+2}\, \ldots\, d_{i-1}\, d_i}\, d_{i+1}\, \ldots$$

(подчёркнуты совпадающие подстроки длины k). Стало быть, имеем $d_1 = d_{i-k+1}$, $d_2 = d_{i-k+2}$, $\ldots$, $d_{k-1} = d_{i-1}$, $d_k = d_i$, а $d_{k+1} \neq d_{i+1}$. Нужно найти такой более короткий префикс $D(1\,..\,j)$ строки D некоторой длины j, чтобы, во-первых, он был суффиксом подстроки $D(1\,..\,i)$, а в-вторых, $d_{j+1} = d_{k+1}$:

$$D = \overbrace{\underline{d_1\, d_2\, \ldots\, d_{j-1}\, d_j}}\; d_{j+1}\, \ldots\, d_{k-1}\, d_k\, d_{k+1}\, \ldots\, d_{i-k}\, \underline{d_{i-k+1}\, d_{i-k+2}\, \ldots\, \overbrace{d_{i-j+1}\, d_{i-j+2}\, \ldots\, d_{i-1}\, d_i}}\, d_{i+1}\, \ldots$$

Верхними фигурными скобками выделены подстроки, которые по первому свойству должны совпадать.

Из новой схемы имеем, что

$$d_j = d_i = d_k,\ d_{j-1} = d_{i-1} = d_{k-1},\ d_{j-2} = d_{i-2} = d_{k-2},\ \ldots,$$
$$d_2 = d_{i-j+2} = d_{k-j+2},\ d_1 = d_{i-j+1} = d_{k-j+1}.$$

Здесь вторые равенства обеспечиваются совпадением подчёркнутых подстрок. Отсюда имеем, что подстрока $D(1\,..\,j)$ является префикс-суффиксом подстроки $D(1\,..\,k)$:

$$D = \overbrace{d_1\, d_2\, \ldots\, d_{j-1}\, d_j}\; d_{j+1}\, \ldots\, d_{k-j}\, \overbrace{d_{k-j+1}\, d_{k-j+2}\, \ldots\, d_{k-1}\, d_k}\ldots$$

А максимальная длина префикс-суффикса подстроки $D(1\,..\,k)$ есть $\pi(D,k)$. Стало быть, для проверки сначала надо взять $j = \pi(D,k)$. Если при этом $d_{j+1} = d_{i+1}$, то положить $\pi(D,i+1) = j+1 = \pi(D,k)+1$. Если же $d_{j+1} \neq d_{i+1}$, то надо снова укоротить строку, и в силу аналогичных рассуждений надо взять $j' = \pi(D,j)$, и т.д., пока не будет найден индекс $j^* > 0$ такой, что $d_{j^*+1} = d_{i+1}$, либо не получим, что очередной индекс $j^* = 0$. В первом случае полагаем $\pi(D,i) = j^* + 1$, в последнем — $\pi(D,i+1) = 0$.

В целом, алгоритм построения набора `p[]` значений префикс-функции $\pi(D,\cdot)$ строки D выглядит следующим образом:

```
p[1] := 0;
k := 0;
for i := 1 to |D|-1 do
begin
  while (k > 0) and (D[k+1] <> D[i+1]) do k := p[k];
  if D[k+1] = D[i+1] then inc(k);
  p[i+1] := k;
end;
```

Первая строка в цикле осуществляет (при необходимости) укорочение рассматриваемого префикса. Вторая строка увеличивает на 1 длину найденного префикс-суффикса (если он найден). Третья строка запоминает найденное значение. Сложность этого алгоритма есть $O(n)$, где n — длина обрабатываемой строки. Это, кстати, весьма тонкий вопрос — почему внутренний цикл while не добавляет сложности. За доказательством этого факта отправим читателя к книге Кормен Т.Х., Лейзерсон Ч.И., Ривест Р.Л., Штайн К., Алгоритмы. Построение и анализ. М.: Вильямс, 2013.

Пусть мы умеем строить префикс-функцию заданной строки. Вернёмся к задаче подсчёта вхождений строки T в строку S. Алгоритм Кнута – Морриса – Пратта

базируется на следующей идее. Рассмотрим строку $D = T\#S$ (символ # такой, что он гарантированно не входит ни в T, ни в S; в рамках данной задачи может быть взят любой небуквенный символ) и станем строить префикс-функцию такой строки. Если для какого-либо индекса i^* из диапазона $|T|+2\,..\,|T|+|D|+1$ (то есть из диапазона, соответствующего символам строки S) окажется, что $\pi(D, i^*) = |T|$, то мы нашли место окончания префикс-суффикса длины $|T|$, то есть префикс-суффикса, совпадающего с T, то есть нашли последний символ вхождения подстроки T. Сложность данного алгоритма есть $O(|T| + |S|)$ — время построения префикс-функции для объединенной строки.

Таким образом, общая схема достаточно эффективного решения данной задачи выглядит следующим образом. Считываем строку-ключ S. В цикле считываем строки из набора. Для каждой очередной строки T алгоритмом Кнута – Морриса – Пратта считаем количество её вхождений в строку-ключ (здесь разумно реализовать функцию, которая будет возвращать символ из объединённой строки $T\#S$ по его индексу). Имея эту информацию, отрабатываем алгоритм поиска наибольшего элемента из заданного набора по функции сравнения

$$\mathrm{str}_1 \preceq \mathrm{str}_2 \Leftrightarrow E(\mathrm{str}_1) < E(\mathrm{str}_2) \vee \big(E(\mathrm{str}_1) = E(\mathrm{str}_2) \wedge \mathrm{str}_1 \leqslant \mathrm{str}_2\big).$$

Символ $\preceq$ означает «меньше или равно в смысле Петиного порядка», $E(\cdot)$ — количество вхождения данной строки в строку-ключ, $\leqslant$ — сравнение строк в обычном лексикографическом порядке.

Сложность такого алгоритма — $O\big(n(|S|+|T|)\big)$, где $|S|$ — длина строки-ключа, $|T|$ — (максимальная) длина строки из набора, n — количество строк в наборе. Заметим, что существует ещё более эффективный алгоритм со сложностью $O(|S| + n|T|)$ (алгоритм Ахо – Корасик), однако его реализация значительно более трудоёмка нежели реализация алгоритма Кнута – Морриса – Пратта.

Отметим, что участники, использующие язык Python, при решении данной задачи имеют некоторое преимущество: в стандартной библиотеке этого языка для работы со строками уже реализована быстрая функция подсчёта количества вхождений данной строки в другую.

Далее в идеях тестов L — длина строки-ключа, l — длины строк из набора.

Идеи тестов:

1. Минимальный тест: $L = l = 1$, $n = 5$, у всех строк вхождение 0.
2. Минимальный тест: $L = l = 1$, $n = 5$, есть строка с вхождением 1.
3. $L = 2$, $l = 1$, $n = 5$, у всех строк вхождение 0.
4. $L = 2$, $l = 1$, $n = 5$, у двух строк вхождение 1.
5. $L = 2$, оба символа одинаковы, $l \leq 2$, $n = 5$, есть строка с вхождением 2 и вхождением 1.
6. $L = 20$, $l \leqslant 7$, $n = 5$, есть строка с вхождением 5, вхождения не пересекаются.

7. $L = 20$, $l \leqslant 7$, $n = 5$, есть строка с вхождением 5, вхождения пересекаются.
8. $L = 20$, $l \leqslant 7$, $n = 5$, есть две строки с вхождением 5, вхождения пересекаются.

9–12. Случайные тесты: $L = 100$, $l \in [10, 20]$, $n = 50$; строки могут иметь префикс, являющийся подстрокой ключа.

13–16. Случайные тесты: $L = 1000$, $l \in [40, 60]$, $n = 500$; строки могут иметь префикс, являющийся подстрокой ключа.

17–19. Случайные тесты: $L \in [5000, 8000]$, $l \in [70, 100]$, $n = 1000$; строки могут иметь префикс, являющийся подстрокой ключа.

20. $L = 100$, строка состоит из одинаковых символов, $l \in [1, 15]$, $n = 20$, есть строки разной длины, состоящие из одного символа того же, что и строка-ключ; прочие строки могут иметь префикс, являющийся подстрокой ключа.
21. $L = 1000$, строка состоит из одинаковых символов, $l \in [1, 50]$, $n = 200$, есть строки разной длины, состоящие из одного символа того же, что и строка-ключ; прочие строки могут иметь префикс, являющийся подстрокой ключа.
22. $L = 10000$, строка состоит из одинаковых символов, $l \in [1, 100]$, $n = 2000$, есть строки разной длины, состоящие из одного символа того же, что и строка-ключ; прочие строки могут иметь префикс, являющийся подстрокой ключа.
23. Максимальный тест: $L = 10000$, $l = 100$, $n = 10000$, у всех строк вхождение 0.
24. Максимальный тест: $L = 10000$, $l = 100$, $n = 10000$, есть строки с вхождением 1.
25. Максимальный тест: $L = 10000$, $l = 100$, $n = 10000$, есть строки с большим числом вхождений.