Муниципальный этап Всероссийской олимпиады школьников по информатике в 2017 – 2018 учебном году

Разборы решений и идеи тестов

Муниципальный этап Всероссийской олимпиады школьников по информатике в 2017 – 2018 учебном году

8 класс

Время выполнения задач — 4 часа

Ограничение по времени — 2 секунды на тест

Ограничение по памяти — 256 мегабайт

8.1. «Накачать музыки». Родители Пети Торопыжкина оплачивают ему тариф на сотовом телефоне, по которому он может скачать a гигабайт данных. Каждые 100 мегабайт трафика сверх этого количества стоят b рублей; при этом каждая 100-мегабайтная порция данных может быть оплачена только целиком. Каждый месяц родители дают Пете d рублей карманных денег. Сколько мегабайт данных Петя сможет скачать из Сети, если потратит максимальное количество карманных денег на оплату мобильного интернета? Напомним, что 1 Гб = 1024 Мб.

Формат входа: В единственной строке через пробел указаны три целых числа a, b, d ($1 \leqslant a \leqslant 1000$, $1 \leqslant b \leqslant 100$, $0 \leqslant d \leqslant 1000$) — объем данных (в гигабайтах), которые можно скачать по основному тарифу, стоимость дополнительной 100-мегабайтной порции данных сверх тарифа и месячное количество карманных денег у Пети (суммы — в рублях).

Формат выхода: Выведите единственное целое число — объем трафика (в мегабайтах), который сможет скачать Петя.

Пример 1		**Пример 2**	
Вход:	*Выход:*	*Вход:*	*Выход:*
`4 50 0`	`4096`	`4 50 220`	`4496`

8.2. «Обязанности на даче». Летом в саду Петя Торопыжкин был ответственным за полив цветов, поэтому каждое утро он наполнял большую флягу водой. Для этого нужно было добраться с флягой в тележке до одного из двух поселковых колодцев, наполнить флягу и вернуться домой. Путь до первого колодца и обратно занимал p_1 секунд. Чтобы наполнить флягу, нужно было n_1 раз опустить колодезное ведро вниз, вытащить и перелить воду из него во флягу. Каждая такая операция требовала t_1 секунд. Аналогичные показатели для второго колодца — p_2 секунд, n_2 раз, t_2 секунд. К какому колодцу ходил Петя и сколько он тратил на это времени с учётом того, что он хотел побыстрее завершить доставку воды?

Формат входа: В первой строке через пробел перечислены три целых числа p_1, n_1, t_1 — параметры первого колодца. Во второй строке через пробел указаны три целых числа p_2, n_2, t_2 — параметры второго колодца. Ограничения на параметры: $1 \leqslant p_1, p_2, n_1, n_2, t_1, t_2 \leqslant 10^4$.

Формат выхода: Выдайте через пробел два целых числа: номер колодца, который позволял Пете быстрее завершить доставку воды, и время (в секундах), которое требовалось на это. Если с точки суммарных времязатрат колодцы одинаковы, укажите первый.

Пример 1		**Пример 2**	
Вход:	*Выход:*	*Вход:*	*Выход:*
`10 5 10` `20 2 20`	`1 60`	`10 5 10` `15 2 20`	`2 55`

8.3. «Числовая последовательность». На уроке математики Петя Торопыжкин придумал интересное правило пересчёта целого числа. От имеющегося числа отделяется последняя цифра его десятичной записи, возводится в пятую степень, умножается на 20 и прибавляется к числу, получившемуся после отделения этой цифры. Петя считает, что после отделения последней цифры от однозначного числа получается ноль. Математически эту операцию можно описать следующим образом:

$$\overline{a_n a_{n-1} \ldots a_2 a_1 a_0} \to \overline{a_n a_{n-1} \ldots a_2 a_1} + 20 \cdot (a_0)^5.$$

Пусть задано начальное число d, и Петя применяет к нему придуманную операцию k раз, получая еще k чисел. Какое число будет наибольшим среди имеющихся $(k+1)$-го числа?

Формат входа: В первой строке через пробел вводятся два целых числа: d — начальное число — и k — количество применений Петиной операции ($0 \leqslant d \leqslant 10^9$, $0 \leqslant k \leqslant 10^4$).

Формат выхода: Выведите единственное целое число, максимальное в полученном наборе.

Пример

Вход:	*Выход:*
`10 10`	`671116`

Примечание: Получится такой набор чисел: $10 \to 1 \to 20 \to 2 \to 640 \to 64 \to 20486 \to 157568 \to 671116 \to 222631 \to 22283$, в котором максимум равен 671116.

8.4. «Самое частое буквосочетание». Имеется строка, состоящая из заглавных слов латиницы и пробелов, с длиной не более 255 символов. Словом Петя Торопыжкин называет последовательность букв, ограниченную пробелами, началом или концом строки. Пара соседних слов разделена хотя бы одним пробелом. В строке имеется хотя бы одно двухбуквенное слово. Петя Торопыжкин решил выяснить, какое двухбуквенное сочетание подряд идущих букв одного слова является наиболее частым в этом тексте. Помогите ему, напишите программу, которая будет находить требуемую информацию.

Формат входа: В единственной строке задан текст, удовлетворяющий указанным условиям. Длина текста не превосходит 255 символов.

Формат выхода: Выведите единственное двухбуквенное слово, представляющее сочетание букв, наиболее частое в данном тексте. Если таких сочетаний несколько выдайте то, которое больше в лексикографическом порядке. (Сравнение строк в лексикографическом порядке подразумевает поиск первой пары несовпадающих символов, стоящих в строках на соответствующих позициях, которые и определяют порядок строк; если одна строка является началом другой, то она считается меньшей.)

Пример

Вход:	*Выход:*
`ABCABC.A`	`BC`

Примечание: Точкой в примере обозначен пробел.

8.5. «Регистрируем НЛО». Летом в деревне, где отдыхал Петя Торопыжкин, обнаружили НЛО. От местного населения поступило n измерений положения этого объекта в виде `HH:MM:SS X Y`, где `HH:MM:SS` — момент фиксации объекта, `X` и `Y` — координаты места, где был зафиксирован объект. Все моменты времени различны. Очевидцы утверждают, что между моментами фиксации НЛО двигался прямолинейно. Координаты заданы в местной системе координат в метрах. Необходимо вычислить максимальную и минимальную среднюю скорость объекта на промежутках времени между последующими измерениями его положения. Напомним, что длина отрезка с концами в точках $A(x_1, y_1)$ и $B(x_2, y_2)$ находится по формуле $|AB| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

Формат входа: В первой строке задано натуральное число n — количество измерений положения НЛО ($2 \leqslant n \leqslant 20\,000$). В следующих n строках в каком-то порядке приведены имеющиеся измерения в указанном выше формате, ограничения: $00 \leqslant \mathtt{HH} \leqslant 23$, $00 \leqslant \mathtt{MM}, \mathtt{SS} \leqslant 59$, $|\mathtt{X}|, |\mathtt{Y}| \leqslant 10^4$. Считается, что все измерения сделаны в течение одних суток, и нет двух измерений в один момент времени; координаты являются целыми числами.

Формат выхода: Выведите в единственной строке разделённые пробелом два вещественных числа, которые с абсолютной точность $0.5 \cdot 10^{-3}$ приближают точные значения минимальной (первое) и максимальной (второе) средней скорости НЛО между двумя соседними измерениями, выраженные в м/с (то есть отличаются не более, чем на $0.5 \cdot 10^{-3}$ от истинных значений скоростей).

Пример

Вход:	*Выход:*

`4`
`13:01:11 0 0`
`13:01:03 10 10`
`13:01:14 0 10`
`13:01:07 10 0`
	`2.5 3.3333`

Муниципальный этап Всероссийской олимпиады школьников по информатике в 2017 – 2018 учебном году
8 класс. Разбор решений и идеи тестов

8.1. «Накачать музыки». *Родители Пети Торопыжкина оплачивают ему тариф на сотовом телефоне, по которому он может скачать a гигабайт данных. Каждые 100 мегабайт трафика сверх этого количества стоят b рублей; при этом каждая 100-мегабайтная порция данных может быть оплачена только целиком. Каждый месяц родители дают Пете d рублей карманных денег. Сколько мегабайт данных Петя сможет скачать из Сети, если потратит максимальное количество карманных денег на оплату мобильного интернета? Напомним, что 1 Гб = 1024 Мб.*

Данная задача имеет статус «утешительной». Количество дополнительных 100-мегабайтных порций, которые можно купить за d рублей, есть d `div` b. Стало быть, ответ равен $1024 \cdot a + d$ `div` b.

Идеи тестов:

1–4. Случайные тесты, $d = 0$.
5–8. Случайные тесты, $0 < d < b$.
9–12. Случайные тесты, $d = b$.
13–16. Случайные тесты, $d = kb$.
17–20. Случайные тесты, $d = kb + l$, $l < b$.

8.2. «Обязанности на даче». *Летом в саду Петя Торопыжкин был ответственным за полив цветов, поэтому каждое утро он наполнял большую флягу водой. Для этого нужно было добраться с флягой в тележке до одного из двух поселковых колодцев, наполнить флягу и вернуться домой. Путь до первого колодца и обратно занимал p_1 секунд. Чтобы наполнить флягу, нужно было n_1 раз опустить колодезное ведро вниз, вытащить и перелить воду из него во флягу. Каждая такая операция требовала t_1 секунд. Аналогичные показатели для второго колодца — p_2 секунд, n_2 раз, t_2 секунд. К какому колодцу ходил Петя и сколько он тратил на это времени с учётом того, что он хотел побыстрее завершить доставку воды?*

Программный комитет считает данную задачу простой. Решение состоит в вычислении для обоих колодцев полного времени, нужного для наполнения фляги водой и доставки её до дома: $T_i = p_i + n_i \cdot t_i$, и выборе меньшего из них (с контролем совпадения).

Идеи тестов:

1–8. Случайные тесты, первый колодец выгоднее.
9–16. Случайные тесты, второй колодец выгоднее.
17–20. Случайные тесты, колодцы одинаково выгодны.

8.3. «Числовая последовательность». *На уроке математики Петя Торопыжкин придумал интересное правило пересчёта целого числа. От имеющегося числа отделяется последняя цифра его десятичной записи, возводится в пятую степень, умножается на* 20 *и прибавляется к числу, получившемуся после отделения этой цифры. Петя считает, что после отделения последней цифры от однозначного числа получается ноль. Математически эту операцию можно описать следующим образом:*

$$\overline{a_n a_{n-1} \ldots a_2 a_1 a_0} \to \overline{a_n a_{n-1} \ldots a_2 a_1} + 20 \cdot (a_0)^5.$$

Пусть задано начальное число d*, и Петя применяет к нему придуманную операцию* k *раз, получая еще* k *чисел. Какое число будет наибольшим среди имеющихся* $(k+1)$*-го числа?*

Сложность данной задачи чуть ниже средней. Задача носит технический характер, однако требует базовых технических навыков: организация алгоритма поиска максимума и организация вычисления указанной операции, для чего требуется умение работать с числами на уровне цифр — выделять последнюю цифру при помощи операции взятия остатка от деления на 10 и отбрасывания последней цифры числа при помощи целочисленного деления на 10.

Важным моментом является то, что в процессе применения операции мы всегда останемся в рамках 4-байтного целого типа. Действительно, при применении операции число уменьшается в 10 раз, а потом увеличивается на величину, не превосходящую значения $20 \cdot 9^5 = 1180980$. Стало быть, начав даже с числа 999 999 999, на первых нескольких применениях петиной операции будем получать уменьшение числа.

Идеи тестов:

1. $d = 0$, $k = 0$.
2. $0 < d < 10$, $k = 0$.
3. d — трёхзначное, $k = 0$.
4. $d \approx 10^6$, $k = 0$.
5. $d \approx 10^9$, $k = 0$.
6. $d = 0$, $k = 1$.
7. $0 < d < 10$, $k = 1$.
8. d — трёхзначное, $k = 1$.
9. $d \approx 10^6$, $k = 1$.
10. $d \approx 10^9$, $k = 1$.
11. $d = 0$, $k = 100$.
12. $0 < d < 10$, $k = 100$.
13. d — трёхзначное, $k = 100$.
14. $d \approx 10^6$, $k = 100$.
15. $d \approx 10^9$, $k = 100$.
16. $d = 0$, $k = 1000$.
17. $0 < d < 10$, $k = 1000$.

18. d — трёхзначное, $k = 1000$.
19. $d \approx 10^6$, $k = 1000$.
20. $d \approx 10^9$, $k = 1000$.

21–25. Случайные тесты.

8.4. «Самое частое буквосочетание». *Имеется строка, состоящая из заглавных слов латиницы и пробелов, с длиной не более 255 символов. Словом Петя Торопыжкин называет последовательность букв, ограниченную пробелами, началом или концом строки. Пара соседних слов разделена хотя бы одним пробелом. В строке имеется хотя бы одно двухбуквенное слово. Петя Торопыжкин решил выяснить, какое двухбуквенное сочетание подряд идущих букв одного слова является наиболее частым в этом тексте. Помогите ему, напишите программу, которая будет находить требуемую информацию.*

Данная задача, по мнению программного комитета, имеет сложность выше среднего. Основная задача, которую надо решить участнику — как хранить информацию о частотах различных буквосочетаний.

Наиболее разумный вариант такой структуры — двумерный массив размера 26×26 (с индексами, отсчитываемыми от нуля). Информация о количестве вхождений в текст конкретного буквосочетания $\alpha\beta$, α, $\beta \in$ `A..Z` хранится в элементе этого массива с индексами $i = \texttt{ord}(\alpha) - \texttt{ord}('\texttt{A}')$, $j = \texttt{ord}(\beta) - \texttt{ord}('\texttt{A}')$ (`ord` — функция получения кода символа). В начале массив заполнен нулями. Затем в цикле по i от 2 до длины обрабатываемой строки `str` проверяем буквосочетания $\texttt{str}[i-1]\texttt{str}[i]$. Если ни один из двух символов не пробел, вычисляем индексы и увеличиваем на 1 соответствующую ячейку массива.

После того, как строка обработана, двумя вложенными циклами идём по двумерному массиву (внешний по первому индексу, то есть по первому символу буквосочетания, внутренний — по второму индексу) и ищем максимум, запоминая индексы ячейки, в которой этот максимум достигается. Такой порядок перебора ячеек соответствует перебору соответствующих буквосочетаний в лексикографическом порядке. Если какая-то новая ячейка содержит значение, совпадающее с текущим максимумом, то нужно запомнить её индексы. После окончания поиска максимума индексы переводим в символы и выводим соответствующее буквосочетание.

Идеи тестов:

1. Вся строка — единственное двухбуквенное слово.
2. Строка — единственное двухбуквенное слово с пробелом после него.
3. Строка — единственное двухбуквенное слово с несколькими пробелами после него.
4. Строка — единственное двухбуквенное слово с пробелом перед ним.
5. Строка — единственное двухбуквенное слово с несколькими пробелами перед ним.

6. Строка — единственное двухбуквенное слово с пробелом перед ним и после него.
7. Строка — единственное двухбуквенное слово с несколькими пробелами перед ним и после него.
8. Строка представляет собой некоторое количество однобуквенных слов и единственное двухбуквенное, которое стоит в начале строки.
9. Строка представляет собой некоторое количество однобуквенных слов и единственное двухбуквенное, которое стоит в начале строки после одного пробела.
10. Строка представляет собой некоторое количество однобуквенных слов и единственное двухбуквенное, которое стоит в начале строки после нескольких пробелов.
11. Строка представляет собой некоторое количество однобуквенных слов и единственное двухбуквенное, которое стоит в конце строки.
12. Строка представляет собой некоторое количество однобуквенных слов и единственное двухбуквенное, которое стоит в конце строки, имея после себя пробел.
13. Строка представляет собой некоторое количество однобуквенных слов и единственное двухбуквенное, которое стоит в конце строки, имея после себя несколько пробелов.
14. Кроме однобуквенных слов строка содержит несколько двухбуквенных, каждого по 1 штуке.
15. Кроме однобуквенных слов строка содержит несколько двухбуквенных, каждого по 1 штуке, кроме одного, которого 2 штуки; это слово не наибольшее в лексикографическом порядке.
16. Кроме однобуквенных слов строка содержит несколько двухбуквенных, каждого по 2 штуке.
17. Кроме однобуквенных слов строка содержит несколько двухбуквенных в различных количествах; самое частое сочетание не наибольшее в лексикографическом порядке.
18. Кроме однобуквенных слов строка содержит несколько двухбуквенных в различных количествах; несколько буквосочетаний имеют одинаковую максимальную частоту.
19. В строке имеются слова различной длины; каждое буквосочетание уникально.
20. В строке имеются слова различной длины; каждое буквосочетание уникально кроме одного, которого 2 штуки; оно не максимально в лексикографическом порядке.
21. В строке имеются слова различной длины; каждое буквосочетание встречается по 2 раза.
22. В строке имеются слова различной длины; буквосочетания имеют различные частоты; самое частое единственно и немаксимально в лексикографическом порядке.
23. В строке имеются слова различной длины; буквосочетания имеют различные

частоты; несколько сочетаний с максимальной частотой.

24. Вся строка — одно 255-символьное слово, состоящее из одинаковых букв.
25. Вся строка — одно 255-символьное слово, состоящее из разных букв; несколько сочетаний с максимальной частотой.

8.5. «Регистрируем НЛО». *Летом в деревне, где отдыхал Петя Торопыжкин, обнаружили НЛО. От местного населения поступило n измерений положения этого объекта в виде* `HH:MM:SS X Y`*, где* `HH:MM:SS` *— момент фиксации объекта,* `X` *и* `Y` *— координаты места, где был зафиксирован объект. Все моменты времени различны. Очевидцы утверждают, что между моментами фиксации НЛО двигался прямолинейно. Координаты заданы в местной системе координат в метрах. Необходимо вычислить максимальную и минимальную среднюю скорость объекта на промежутках времени между последующими измерениями его положения. Напомним, что длина отрезка с концами в точках* $A(x_1, y_1)$ *и* $B(x_2, y_2)$ *находится по формуле* $|AB| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

Задача считается программным комитетом сложной, хотя её сложность, скорее техническая.

Идеологически решение несложно:
1) сортируем данные по моментам измерений; здесь разумно перевести моменты времени из формата `HH:MM:SS` в целое число — количество секунд, прошедших с полуночи до момента измерения; соответственно, сортируются сложные объекты, содержащие время измерения и координаты;
2) по каждой паре измерений, соседних в смысле нового порядка, считаем расстояние между точками и делим на промежуток времени между этими измерениями, получая среднюю скорость НЛО на этом промежутке;
3) используя классические алгоритмы поиска минимума и максимума, находим минимум и максимум средних скоростей.

Ограничения на объём входных данных таковы, что при сортировке надо использовать быстрые алгоритмы, на больших тестах квадратичные алгоритмы (типа пузырьковой сортировки) не уложатся во время.

В каждом поднаборе тестов есть тесты, где минимальная средняя скорость равна нулю, а также тесты, в которых НЛО вообще стоит на месте.

<u>Идеи тестов:</u>

1. Всего есть два измерения.
2–9. Случайные тесты, $n \leqslant 100$;
10–17. Случайные тесты, $100 \leqslant n \leqslant 7000$;
18–25. Случайные тесты, $15000 \leqslant n \leqslant 20000$; в том числе имеются максимальные тесты с $n = 20000$.

Муниципальный этап Всероссийской олимпиады школьников по информатике в 2017 – 2018 учебном году

9 класс

Время выполнения задач — 4 часа

Ограничение по времени — 2 секунды на тест

Ограничение по памяти — 256 мегабайт

9.1. «Средняя команда». На школьные соревнования по перетягиванию каната класс Пети Торопыжкина может выставить одну из двух команд, каждая из трёх человек. В первой участники могут развивать усилия (выраженные в Ньютонах) F_1, F_2, F_3, а во второй — G_1, G_2, G_3. По условиям соревнований нужно, чтобы сумма усилий участников команды была как можно ближе к некоторому указанному значению A. Которую из команд следует выставить петиному классу в соответствием с этим требованием?

Формат входа: В первой строке через пробел перечислены три целых числа F_1, F_2, F_3 — усилия, развиваемые участниками первой команды. Во второй строке через пробел перечислены три целых числа G_1, G_2, G_3 — усилия, развиваемые участниками второй команды. В третьей строке задано одно целое число A. Все числа лежат диапазоне от 1 до 10^4 включительно.

Формат выхода: Выведите через пробел два целых числа: номер команды (1 или 2), которую следует выставить согласно правилам соревнования, и суммарное усилие, развиваемое членами этой команды. Если суммарные усилия обеих команд равноотстоят от заданного значения, выведите более сильную команду. Если обе команды имеют равные суммарные усилия, выведите первую команду.

Пример 1		**Пример 2**	
Вход:	*Выход:*	*Вход:*	*Выход:*
1 2 3 2 3 5 7	1 6	1 2 3 2 3 5 8	2 10

9.2. «Максимум на сломанном калькуляторе». Петя Торопыжкин познакомился с гипотезой Коллатца: какое бы натуральное число a_0 ни взять, последовательность (часто называемая *сиракузской*), вычисляемая от выбранного числа по формуле

$$a_{n+1} = \begin{cases} a_n/2, & \text{если } a_n \text{ — чётное число,} \\ 3a_n + 1, & \text{если } a_n \text{ — нечётное число,} \end{cases}$$

обязательно достигнет единицы. Он решил поэкспериментировать с последовательностью: задать начальную величину, вычислить сколько-то первых членов и посмотреть, какого наибольшего значения они достигнут. Только на калькуляторе, на котором он

считал, сломался экран, и было видно только три последних разряда получаемых чисел. Поэтому Петя искал максимум из тех чисел, которые он видел. Напишите программу, которая выведет найденный им максимум.

Формат входа: В единственной строке через пробел заданы два целых числа: a_0 — начальный член последовательности — и k — количество членов, которые вычисляет Петя, то есть индекс последнего вычисленного члена ($1 \leqslant a_0 \leqslant 10^7$, $0 \leqslant k \leqslant 10^4$). Гарантируется, что все получаемые члены последовательности не превосходят 10^9.

Формат выхода: Выведите через пробел два целых числа — член последовательности a_i, $0 \leqslant i \leqslant k$, для которого три последние цифры дают максимальное число, и индекс i этого члена. Если таких членов несколько, выведите тот, который имеет бóльший индекс.

Пример

Вход:	*Выход:*
`2518 5`	`1889 3`

Примечание: Получаемая последовательность в данном случае выглядит как $a_0 = 2518 \to a_1 = 1259 \to a_2 = 3778 \to a_3 = 1889 \to a_4 = 5668 \to a_5 = 2834$. Видно, что максимум последних трёх цифр достигается на числе 1889.

9.3. «Циклический сдвиг». Петя Торопыжкин придумал красивую строку, состоящую не менее чем из 2 и не более чем из 255 заглавных символов латиницы. Старший брат решил подшутить над Петей, разрезал строку на две части, возможно, поменял их местами и склеил обратно, то есть осуществил некоторый циклический сдвиг символов в строке (возможно, на нулевое число символов). Помогите Пете восстановить его строку, если он помнит, что она была лексикографически максимальной среди всех строк, которые могли бы получиться при её циклических сдвигах.

Формат входа: В единственной строке задана непустая последовательность заглавных латинских букв; длина последовательности не меньше 2 и не больше 255.

Формат выхода: Выведите единственную строку, лексикографически максимальную из тех, которые могут получиться циклическим сдвигом данной строки.

Пример

Вход:	*Выход:*
`ZAZBY`	`ZBYZA`

9.4. «Большое треугольное число». Известно, что числа получаемые суммированием начального отрезка натурального ряда называют *треугольными*. Название происходит оттого, что n-е треугольное число ($n \geqslant 1$) описывает количество точек, из которых состоит треугольник, на стороне которого лежат n точек (см. рисунок). Требуется по заданному целому числу M найти наименьшее треугольное число, не меньшее, чем M.

Формат входа: В единственной строке задано целое число M ($0 \leqslant M \leqslant 10^{18}$).

Формат выхода: Выведите единственное целое число, являющееся наименьшим треугольным числом, не меньшим M.

Пример 1		**Пример 2**	
Вход:	*Выход:*	*Вход:*	*Выход:*
9	10	10	10

9.5. «База данных Деда Мороза». В своей великоустюгской базе данных Дед Мороз хранит информацию о детях, написавших ему письмо с просьбой о подарке. Для каждого письма хранится фамилия, имя, отчество, дата рождения ребёнка и название желаемого подарка. Данные хранятся в виде пяти строк; при этом имя, фамилия, отчество, название подарка — непустые строки, составленные из заглавных символов латиницы; дата рождения — тоже строка вида `YYYY-MM-DD`. Здесь `YYYY` — год рождения, `MM` — месяц, `DD` — день; месяц и день обязательно представляются двумя символами с ведущим нулём при необходимости. Каждой записи из базы данных Дед Мороз сопоставил беззнаковое 4-байтное целое число — сумму кодов в таблице ASCII всех символов всех строк, составляющих запись. В компьютерных науках такое число называется *хешем*, метод вычисления хеша — *хеш-функцией*.

Соответственно, нужна процедура, которая под заданному хешу будет выдавать все записи, хеш которых совпадает с заданным. В случае отсутствия таких записей, следует сообщить об этом, а также выдать количество записей в базе, значения хешей которых отстоят не более, чем на 100, от заданного значения.

Формат входа: В первой строке задано целое число h — желаемое значение хеша ($0 \leqslant h \leqslant 2^{32} - 1$). Во второй строке задано целое число n — количество записей в базе данных ($1 \leqslant n \leqslant 20000$). В следующих n строках идут записи из базы данных (до 100 символов в каждой): по пять последовательностей символов в строке, разделённые пробелами: фамилия, имя, отчество, дата рождения, подарок.

Формат выхода: Если записи, имеющие нужный хеш, найдены, выведите сообщение «`FOUND RECORDS`», в следующей строке — их количество, затем по одной в строке выведите все такие записи в любом порядке. Если подходящих записей не найдено, выведите «`NO MATCHES`». В следующей строке выведите количество записей, имеющих хеш, отстоящий от заданного значения не более чем на 100, затем по одному в строке выведите эти хеши в любом порядке.

Пример 1

Вход:

```
2312
3
PETROV IVAN SIDOROVICH 2010-12-03 MECH
IVANOV SIDOR PETROVICH 2012-10-30 MECH
SIDOROV PETR IVANOVICH 2011-03-21 MECH
```

Выход:

```
FOUND RECORDS
2
IVANOV SIDOR PETROVICH 2012-10-30 MECH
PETROV IVAN SIDOROVICH 2010-12-03 MECH
```

Пример 2

Вход:	*Выход:*
2413 3 PETROV IVAN SIDOROVICH 2010-12-03 MECH IVANOV SIDOR PETROVICH 2012-10-30 MECH SIDOROV PETR IVANOVICH 2011-03-21 MECH	NO MATCHES 1 2313

Муниципальный этап Всероссийской олимпиады школьников по информатике в 2017 – 2018 учебном году
9 класс. Разбор решений и идеи тестов

9.1. «Средняя команда». *На школьные соревнования по перетягиванию каната класс Пети Торопыжкина может выставить одну из двух команд, каждая из трёх человек. В первой участники могут развивать усилия (выраженные в Ньютонах) F_1, F_2, F_3, а во второй — G_1, G_2, G_3. По условиям соревнований нужно, чтобы сумма усилий участников команды была как можно ближе к некоторому указанному значению A. Которую из команд следует выставить петиному классу в соответствием с этим требованием?*

Данная задача носит утешительный характер. Нужно просуммировать каждую из троек чисел, с помощью ветвления выяснить, которая находится ближе к указанному значению, и вывести информацию об этой тройке. Аккуратно надо обработать случай троек, равноотстоящих от указанного значения.

Идеи тестов:

1–7. Случайные тесты, тройки по разному отстоят от указанного значения; оптимальная более слабая тройка (которая может быть как первой, так и второй).

8–14. Случайные тесты, тройки по разному отстоят от указанного значения; оптимальная более сильная тройка (которая может быть как первой, так и второй).

15–20. Случайные тесты, тройки равноотстоят от указанного значения (есть случаи двух слабых троек, двух сильных; одной слабой и одной сильной).

9.2. «Максимум на сломанном калькуляторе». *Петя Торопыжкин познакомился с гипотезой Коллатца: какое бы натуральное число a_0 ни взять, последовательность (часто называемая* сиракузской*), вычисляемая от выбранного числа по формуле*

$$a_{n+1} = \begin{cases} a_n/2, & \text{если } a_n \text{ — чётное число,} \\ 3a_n + 1, & \text{если } a_n \text{ — нечётное число,} \end{cases}$$

обязательно достигнет единицы. Он решил поэкспериментировать с последовательностью: задать начальную величину, вычислить сколько-то первых членов и посмотреть, какого наибольшего значения они достигнут. Только на калькуляторе, на котором он считал, сломался экран, и было видно только три последних разряда получаемых чисел. Поэтому Петя искал максимум из тех чисел, которые он видел. Напишите программу, которая выведет найденный им максимум.

Задача, по мнению программного комитета, является простой. Решение подразумевает прямолинейное вычисление членов последовательности с поиском максимума по указанному критерию. Для выделения числа, составленного из трёх последних цифр,

нужно применять операцию взятия остатка от деления на 1000. В алгоритме поиска максимума нужно не забыть про учёт начального члена последовательности; впрочем, эта ситуация хорошо обрабатывается версией алгоритма, в которой начальное значение максимума берётся равным первой из обрабатываемых величин. Также при равенстве максимума на новом числе старому значению максимума нужно не забыть переставить индекс максимального элемента на новое число.

Идеи тестов:

1. $k = 0$.
2. $k = 100$, максимум на начальном числе.
3. $k = 900$, максимум на начальном числе.
4. $k = 100$, максимум единственный на промежуточном числе.
5. $k = 900$, максимум единственный на промежуточном числе.
6. $k = 100$, максимум единственный на последнем числе.
7. $k = 900$, максимум единственный на последнем числе.
8. $k = 100$, максимум неединственный.
9. $k = 500$, максимум неединственный.
10. $k = 1000$, максимум неединственный.
11. $k = 5000$, максимум неединственный.
12. $k = 10000$, максимум неединственный.

13–20. Случайные тесты.

9.3. «Циклический сдвиг». *Петя Торопыжкин придумал красивую строку, состоящую не менее чем из 2 и не более чем из 255 заглавных символов латиницы. Старший брат решил подшутить над Петей, разрезал строку на две части, возможно, поменял их местами и склеил обратно, то есть осуществил некоторый циклический сдвиг символов в строке (возможно, на нулевое число символов). Помогите Пете восстановить его строку, если он помнит, что она была лексикографически максимальной среди всех строк, которые могли бы получиться при её циклических сдвигах.*

Задача по мнению программного комитета имеет сложность ниже средней.

Во-первых, при указанных ограничениях можно просто лобовым способом сформировать всевозможные циклические сдвиги строки и найти из них лексикографический максимум. Сложность такого решения — $O(n^2)$: всего есть n циклических сдвигов, на обработку очередного сдвига нужно $O(n)$ операций — $O(n)$ на копирование и $O(n)$ на сравнение с текущим максимумом.

Решение можно чуть оптимизировать, если не формировать реально каждый сдвиг (и, соответственно, не тратиться на копирование данных), а написать умную операцию сравнения, которая принимает индексы первого символа каждой из сравниваемых строк и сама, соответствующим образом оперируя с индексами, правильно выбирает символы сравниваемых циклических сдвигов.

Идеи тестов:

1. Двухбуквенная строка, символы разные.
2. Двухбуквенная строка, символы одинаковые.
3. Трёхбуквенная строка, символы разные.
4. Трёхбуквенная строка, два символа совпадают, они меньше третьего символа; максимум — исходная строка.
5. Трёхбуквенная строка, два символа совпадают, они меньше третьего символа; максимум — какой-то сдвиг.
6. Трёхбуквенная строка, два символа совпадают, они больше третьего символа; максимум — исходная строка.
7. Трёхбуквенная строка, два символа совпадают, они больше третьего символа; максимум — какой-то сдвиг.
8. Трёхбуквенная строка, все символы одинаковые.
9. Много букв, все одинаковые.
10. Много букв, максимум единственный — исходная строка.
11. Много букв, максимум неединственный — исходная строка.
12. Много букв, максимум единственный — какой-то сдвиг.
13. Много букв, максимум неединственный — какой-то сдвиг.

14–18. То же, что и в тестах 9–13, длина строки максимальная.

19–20. Случайные тесты.

9.4. «Большое треугольное число». *Известно, что числа получаемые суммированием начального отрезка натурального ряда называют* треугольными. *Название происходит оттого, что n-е треугольное число ($n \geqslant 1$) описывает количество точек, из которых состоит треугольник, на стороне которого лежат n точек (см. рисунок). Требуется по заданному целому числу M найти наименьшее треугольное число, не меньшее, чем M.*

Данная задача имеет сложность чуть выше средней.

Моменты необходимые для решения задачи:

1) вспомнить формулу арифметической прогрессии $1 + 2 + \ldots + n = n(n+1)/2$;

2) найти минимальное положительное решение n^* квадратичного неравенства, вытекающего из смысла задачи: $n(n+1)/2 > M$, то есть $n^2 + n - 2M > 0$.

Искомое решение равно $\left\lceil \dfrac{-1+\sqrt{1+8M}}{2} \right\rceil$. Здесь $\lceil\cdot\rceil$ — операция округления вверх. Проблем с точностью тут не будет, поскольку при указанных ограничениях числа M и $8M$ без потери точности преобразуется к вещественному типу `double`.

3) По найденному номеру n^* нужно вычислить требуемое треугольное число.

Альтернативой вещественным вычислениям может стать метод двоичного поиска. Проверяем, что 1-е треугольное число меньше M. Треугольное число с номером

$1\,500\,000\,000$ гарантированно больше, чем M, удовлетворяющее ограничениям задачи. Затем начинаем вдвое сужать размер интервала, поддерживая то свойство, что на левом конце имеем номер треугольного числа, меньшего, чем M, а на правом — большего, чем M. Когда интервал сожмётся до длины 1, правый конец будет указывать нужный номер.

Идеи тестов:

1. $M = 0$.
2. $M = 1$.
3. $M = 10^{18}$.

4–8. Случайные тесты такие, что $M < 1000$.

9–13. Случайные тесты такие, что $M < 10^9$.

14–20. Случайные тесты такие, что $10^9 < M < 10^{18}$.

9.5. «База данных Деда Мороза». *В своей великоустюгской базе данных Дед Мороз хранит информацию о детях, написавших ему письмо с просьбой о подарке. Для каждого письма хранится фамилия, имя, отчество, дата рождения ребёнка и название желаемого подарка. Данные хранятся в виде пяти строк; при этом имя, фамилия, отчество, название подарка — непустые строки, составленные из заглавных символов латиницы; дата рождения — тоже строка вида* `YYYY-MM-DD`*. Здесь* `YYYY` *— год рождения,* `MM` *— месяц,* `DD` *— день; месяц и день обязательно представляются двумя символами с ведущим нулём при необходимости. Каждой записи из базы данных Дед Мороз сопоставил беззнаковое 4-байтное целое число — сумму кодов в таблице ASCII всех символов всех строк, составляющих запись. В компьютерных науках такое число называется* хешем*, метод вычисления хеша —* хеш-функцией*.*

Соответственно, нужна процедура, которая под заданному хешу будет выдавать все записи, хеш которых совпадает с заданным. В случае отсутствия таких записей, следует сообщить об этом, а также выдать количество записей в базе, значения хешей которых отстоят не более, чем на 100*, от заданного значения.*

Программный комитет оценивает данную задачу как достаточно сложную. Впрочем, сложность данной задачи скорее техническая: нужно совместить два алгоритма пересчёта объектов.

Действительно, пока не найдена строка, имеющая заданный хеш, нужно считать количество строк, чей хеш отстоит не слишком далеко от заданного. Когда (если) найдена строка, имеющая заданный хеш, то нужно переключиться на поиск и выдачу записей, имеющих заданный хеш.

Следует также заметить, что оптимальная процедура подсчёта хеша не требует разделения входной строки по символам пробела, достаточно суммировать коды всех символов, отличных от пробела.

Идеи тестов:

1. 1 запись, хеш не совпадает с требуемым и отстоит далеко.

2. 1 запись, хеш не совпадает с требуемым, отстоит не далеко от требуемого.
3. 1 запись, хеш совпадает с требуемым.
4. 100 записей; хеши не совпадают с требуемым и отстоят далеко.
5. 100 записей; хеши не совпадают с требуемым; у некоторых записей хеши отстоят недалеко от требуемого.
6. 100 записей; хеши не совпадают с требуемым; у всех записей хеши отстоят недалеко от требуемого.
7. 100 записей; есть записи, у которых хеши совпадают с требуемым, у остальных записей хеши отстоят далеко от требуемого; первая запись с требуемым хешем не первая в базе.
8. 100 записей; есть записи, у которых хеши совпадают с требуемым, у остальных записей хеши отстоят далеко от требуемого; первая запись с требуемым хешем первая в базе.
9. 100 записей; есть записи, у которых хеши совпадают с требуемым; есть записи, у которых хеши отстоят недалеко от требуемого; первая запись с требуемым хешем не первая в базе, до неё нет записей с хешем, отстоящим недалеко от требуемого.
10. 100 записей; есть записи, у которых хеши совпадают с требуемым; есть записи, у которых хеши отстоят недалеко от требуемого; первая запись с требуемым хешем не первая в базе, до неё есть записей с хешем, отстоящим недалеко от требуемого.
11. 100 записей; есть записи, у которых хеши совпадают с требуемым; есть записи, у которых хеши отстоят недалеко от требуемого; первая запись с требуемым хешем первая в базе.

12–19. То же, что в тестах 4–11, только в базе 20000 записей.

20–22. Случайные тесты с ответом «`NO MATCHES`».

23–25. Случайные тесты с ответом «`FOUND RECORDS`».

Муниципальный этап Всероссийской олимпиады школьников по информатике в 2017 – 2018 учебном году

10 класс

Время выполнения задач — 4 часа

Ограничение по времени — 2 секунды на тест

Ограничение по памяти — 256 мегабайт

10.1. «Верным путём идём...». Петя Торопыжкин может пойти в школу двумя разными путями. Первый он проходит за t_1 минут, а второй — за t_2 минут. На обдумывание одной интересной идеи он тратит d минут. Во время движения Петя думает постоянно, начиная обдумывать первую идею сразу по выходу из дома и по окончанию обдумывания предыдущей идеи сразу начинает обдумывать следующую. Когда интересная идея обдумана целиком, он получает f единиц удовольствия, а если к моменту окончания пути обдумывание какой-то идеи началось и не завершилось, он теряет l единиц удовольствия. Каким маршрутом нужно ему пойти, чтобы получить наибольшее удовольствие? Укажите номер маршрута и получаемое количество единиц удовольствия за одно прохождение по нему.

Формат входа: В единственной строке через пробел перечислены пять целых чисел: t_1, t_2, d, f, l — время прохождения по первому и второму маршруту, время обдумывания одной идеи, единицы удовольствия, получаемые за полностью обдуманную идею и теряемые за идею, не обдуманную до конца ($1 \leqslant t_1, t_2 \leqslant 10^4$; $1 \leqslant d \leqslant 100$; $0 \leqslant f, l \leqslant 1000$).

Формат выхода: Выведите через пробел два целых числа: номер маршрута (1 или 2), при движении по которому Петя получит наибольшее удовольствие, и количество единиц этого удовольствия. Если оба маршрута дают одинаковое удовольствие, укажите первый.

Пример 1		**Пример 2**	
Вход:	*Выход:*	*Вход:*	*Выход:*
`15 10 2 3 8`	`2 15`	`15 10 2 3 6`	`1 15`

10.2. «Поиск ключа». Петя Торопыжкин постоянно забывал свои пароли от разных сайтов. Для более лёгкого запоминания он всегда выбирал пароли в виде неубывающих последовательностей цифр, больших и малых символов латиницы. При этом порядок на символах соответствовал порядку их кодов в таблице ASCII: цифры меньше больших букв, которые меньше маленьких букв. Буквы внутри наборов возрастают в алфавитном порядке. Чтобы спрятать эти пароли, он написал программу, которая формировала строку длиной до 10^5 символов, в которой введённый Петей пароль был единственной максимальной по длине неубывающей подстрокой. Теперь надо помочь ему написать процедуру, которая из такой строки выделяла бы пароль.

Формат входа: В единственной строке имеется непустая последовательность цифр, больших и малых символов латиницы, такая, что среди всех неубывающих (в смысле указанного порядка символов) подстрок существует единственная максимальной длины. Количество символов в последовательности не превосходит 10^5.

Формат выхода: Выведите максимальную по длине неубывающую подстроку.

Пример

Вход:	*Выход:*
`A01ABab0123`	`01ABab`

10.3. «Словарь братьев по разуму». В фантастическом романе, который пишет Петя Торопыжкин, инопланетные существа используют алфавит, состоящий из двух символов ⌯ и ⦶, которые в рабочем варианте текста Петя представляет заглавными буквами `F` и `G` (для простоты). Петя даже составил словарь языка этих существ. Для быстроты поиска по словарю он выбрал целое число p и сопоставил каждому слову $w = \alpha_0\alpha_1\ldots\alpha_k$ целое число $h(w) = \sum_{i=0}^{k} a_i \cdot p^i$, где коэффициент a_i равен 0, если $\alpha_i =$ `F`, и 1, если $\alpha_i =$ `G`. Однако такое число может быть большим, поэтому Петя запоминает остаток от деления $h(w)$ на некоторое другое целое число D.

Такое число называется *хешем* слова w, а правило вычисления хеша — *хеш-функцией*. Вычислив один раз хеши слов из словаря, дальше очень просто проверять их на несовпадение: если хеши двух слов различаются, то и сами слова совпадать не могут. А вот если хеши двух слов совпадают (такую ситуацию называют *коллизией*), тогда для точной проверки эти слова надо сравнивать посимвольно.

Для быстрой работы со словарём надо написать программу, которая ищет в нём слова, чей хеш совпадает с заданным значением.

Формат входа: В первой строке через пробел заданы два целых числа p и D, определяющие хеш-функцию ($1 \leqslant p \leqslant 10^9$, $1 \leqslant D \leqslant 2 \cdot 10^9$). Во второй строке задано целое число H — требуемое значение хеша ($0 \leqslant H < D$). В третьей строке задано целое число n — количество слов в словаре ($1 \leqslant n \leqslant 10^3$). В следующих n строках заданы слова — непустые последовательности заглавных символов латиницы `F` и `G` длиной не более 10^3 символов.

Формат выхода: Если слова с указанным хешем найдены, выведите в первой строке «`FOUND`», а затем найденные слова по одному в строке (в каком-либо порядке). Если таких слов не найдено, выведите в первой строке «`NOT FOUND`», а во второй через пробел наименьшее и наибольшее значения хешей слов словаря.

Пример 1		**Пример 2**	
Вход:	*Выход:*	*Вход:*	*Выход:*
`7 100`	`FOUND`	`7 100`	`NOT FOUND`
`1`	`G`	`10`	`1 49`
`3`	`FFFFG`	`3`	
`FFFFG`		`FFFFG`	
`FFG`		`FFG`	
`G`		`G`	

10.4. «Перебираем числа». Простая прямолинейная формулировка: есть набор из n целых положительных чисел. Нужно найти наименьшее общее кратное наибольших общих делителей всевозможных троек чисел из этого набора (в тройке числа берутся из разных позиций в наборе). Так как эта величина может быть очень большой, выдайте остаток от её деления на $10^9 + 7$.

Формат входа: В первой строке задано целое число n — количество чисел в наборе ($3 \leqslant n \leqslant 13000$). Во второй строке через пробел перечислены числа a_i из набора ($1 \leqslant a_i \leqslant 3 \cdot 10^6$).

Формат выхода: Выведите единственное целое число — остаток от деления искомого НОК на $10^9 + 7$.

Пример

Вход:	*Выход:*
`4`	`44`
`12 1100 44 242`	

Примечание: Для этих чисел: НОД$(12, 1100, 44) = 4$, НОД$(12, 1100, 242) = 2$, НОД$(12, 44, 242) = 2$, НОД$(1100, 44, 242) = 22$ и НОК$(4, 2, 2, 22) = 44$.

10.5. «Ещё о паролях». Петя Торопыжкин нашёл свой старый архив, но не может вспомнить пароль, чтобы распаковать его. Всё, что он помнит: пароль непуст, состоит только из заглавных символов латиницы, имеет длину не более l символов, не содержит двух одинаковых символов подряд и в смысле лексикографического порядка расположен между двумя известными строками s_1 и s_2, строго больше s_1 и строго меньше s_2. Ему нужно посчитать, сколько таких строк существует (чтобы понять, а стоит ли вообще пытаться перебирать их). Посчитайте и вы. Но поскольку это число может быть очень большим, выдайте остаток от деления его на $10^9 + 7$.

Формат входа: В первой строке задано натуральное число l — максимальная длина пароля ($1 \leqslant l \leqslant 10^5$). Во второй и третьей строках заданы строки s_1 и s_2 — непустые строки из заглавных символов латиницы длины не более l символов, $s_1 < s_2$.

Формат выхода: Выдайте остаток от деления на 10^9+7 количества строк указанного вида.

Пример

Вход:	*Выход:*

`3`
`A`
`BB`
	`677`

Примечание: Эти 677 строк состоят из строк вида: `A?` — 25 штук (на месте `?` может стоять любой из 25 символов, отличных от `A`), `A??` — $25^2 = 625$ (вместо первого `?` стоит любой из 25 символов, отличных от `A`, вместо второго — любой из 25 символов, отличных от символа, стоящего вместо первого `?`), `B` — 1, `BA` — 1, `BA?` — 25 строк: $25 + 625 + 1 + 1 + 25 = 677$.

Муниципальный этап Всероссийской олимпиады школьников по информатике в 2017 – 2018 учебном году
10 класс. Разбор решений и идеи тестов

10.1. «Верным путём идём...». *Петя Торопыжкин может пойти в школу двумя разными путями. Первый он проходит за t_1 минут, а второй — за t_2 минут. На обдумывание одной интересной идеи он тратит d минут. Во время движения Петя думает постоянно, начиная обдумывать первую идею сразу по выходу из дома и по окончанию обдумывания предыдущей идеи сразу начинает обдумывать следующую. Когда интересная идея обдумана целиком, он получает f единиц удовольствия, а если к моменту окончания пути обдумывание какой-то идеи началось и не завершилось, он теряет l единиц удовольствия. Каким маршрутом нужно ему пойти, чтобы получить наибольшее удовольствие? Укажите номер маршрута и получаемое количество единиц удовольствия за одно прохождение по нему.*

Задача имеет уровень утешительной. Решение требует небольших математических размышлений, которые приводят к следующему алгоритму. Сначала для каждого пути вычисляем
1) количество идей, которые будут обдуманы целиком: t_i `div` d;
2) количество удовольствия, которое будет получено за завершение обдумывания этих идей: $(t_i$ `div` $d) \cdot f$;
3) если t_i `mod` $d \neq 0$, то есть на i-м пути имеется идея, не обдуманная до конца, то из полученной величины надо вычесть l.

Затем сравниваем итоговые числа и выбираем наибольшее, аккуратно обрабатывая ситуацию их совпадения.

Идеи тестов:

1–7. Случайные тесты, первый путь лучше.
8–14. Случайные тесты, второй путь лучше.
15–20. Случайные тесты, пути одинаково хороши.

10.2. «Поиск ключа». *Петя Торопыжкин постоянно забывал свои пароли от разных сайтов. Для более лёгкого запоминания он всегда выбирал пароли в виде неубывающих последовательностей цифр, больших и малых символов латиницы. При этом порядок на символах соответствовал порядку их кодов в таблице ASCII: цифры меньше больших букв, которые меньше маленьких букв. Буквы внутри наборов возрастают в алфавитном порядке. Чтобы спрятать эти пароли, он написал программу, которая формировала строку длиной до 10^5 символов, в которой введённый Петей пароль был единственной максимальной по длине неубывающей подстрокой. Теперь надо помочь ему написать процедуру, которая из такой строки выделяла бы пароль.*

Задача представляется, в целом, достаточно простой, хотя имеет несколько тонких моментов в своём решении.

Идея весьма прямолинейна: выделяем из данной строки неубывающие подстроки и ищем из них максимальную в лексикографическом порядке. Алгоритм выделения неубывающей подстроки, в целом, тоже несложен: в цикле перебираем индекс i от 2 до длины строки `str`; если $\texttt{str}[i-1] > \texttt{str}[i]$, то на символе с индексом $i-1$ закончилась очередная неубывающая часть. Тонкость заключается в том, что нужно не забыть обработать неубывающую подстроку, кончающуюся вместе с обрабатываемой строкой.

Соответственно, алгоритм сравнения строк должен на вход принимать индексы начал сравниваемых подстрок и их длины. То есть, если есть желание сэкономить на копировании подстрок, то процедуру сравнения надо реализовать самому.

Заметим, что сложность разумного алгоритма — $O(n)$: каждый символ один раз проверяется на неубывание при формирование очередной неубывающей строки и один раз сравнивается при сравнении текущей накопленной неубывающей подстроки с текущим максимумом.

Идеи тестов:

1. Односимвольная строка.
2. Двухсимвольная строка, символы совпадают.
3. Двухсимвольная строка, символы убывают.
4. Двухсимвольная строка, символы возрастают.
5. Трёхсимвольная строка $a_1a_2a_3$: $a_1 > a_2 > a_3$.
6. Трёхсимвольная строка $a_1a_2a_3$: $a_1 < a_2 > a_3$, максимум a_1a_2.
7. Трёхсимвольная строка $a_1a_2a_3$: $a_1 < a_2 > a_3$, максимум a_3.
8. Трёхсимвольная строка $a_1a_2a_3$: $a_1 > a_2 < a_3$, максимум $a2a3$.
9. Трёхсимвольная строка $a_1a_2a_3$: $a_1 > a_2 > a_3$, максимум a_1.
10. Трёхсимвольная строка $a_1a_2a_3$: $a_1 = a_2 = a_3$.
11. Алфавит в обратном порядке
12. Много букв, максимум в начале строки, это самая длинная из неубывающих подстрок.
13. Много букв, максимум в начале строки, это не самая длинная из неубывающих подстрок.
14. Много букв, максимум в середине строки, это самая длинная из неубывающих подстрок.
15. Много букв, максимум в середине строки, это не самая длинная из неубывающих подстрок.
16. Много букв, максимум в конце строки, это самая длинная из неубывающих подстрок.
17. Много букв, максимум в конце строки, это не самая длинная из неубывающих подстрок.
18. Много букв, вся строка неубывает.

19–25. То же, что в тестах 12–18, длина строки 10^5.

10.3. «Словарь братьев по разуму». *В фантастическом романе, который пишет Петя Торопыжкин, инопланетные существа используют алфавит, состоящий из двух символов ⌔ и ⦇, которые в рабочем варианте текста Петя представляет заглавными буквами* `F` *и* `G` *(для простоты). Петя даже составил словарь языка этих существ. Для быстроты поиска по словарю он выбрал целое число p и сопоставил каждому слову $w = \alpha_0\alpha_1\ldots\alpha_k$ целое число $h(w) = \sum_{i=0}^{k} a_i \cdot p^i$, где коэффициент a_i равен 0, если $\alpha_i =$* `F`*, и 1, если $\alpha_i =$* `G`*. Однако такое число может быть большим, поэтому Петя запоминает остаток от деления $h(w)$ на некоторое другое целое число D.*

Такое число называется хешем слова w, а правило вычисления хеша — хеш-функцией. Вычислив один раз хеши слов из словаря, дальше очень просто проверять их на несовпадение: если хеши двух слов различаются, то и сами слова совпадать не могут. А вот если хеши двух слов совпадают (такую ситуацию называют коллизией), тогда для точной проверки эти слова надо сравнивать посимвольно.

Для быстрой работы со словарём надо написать программу, которая ищет в нём слова, чей хеш совпадает с заданным значением.

Задача представляется программному комитету весьма технической и, как следствие, имеющей сложность чуть ниже среднего.

Собственно, первая техническая тонкость заключается в том, что нужно постоянно считать степени числа p. Если каждый раз честно производить умножение, то общая сложность такого алгоритма будет $O(nl^2)$, где l — длина строк: для каждого слова, которых n штук, l раз надо возвести число p в степень до l. Однако это вычисление можно упростить, даже несколькими способами.

1) Можно предпросчитать один раз все степени вплоть до максимально возможной, а затем просто извлекать эти данные из массива.

2) Можно при обработке каждого слова хранить предыдущую степень p и на её основе вычислять следующую.

3) Можно применить алгоритм *быстрого возведения в степень*:

$$a^n = \begin{cases} n = 2k, & (a^2)^k, \\ n = 2k+1, & (a^2)^k \cdot a. \end{cases}$$

4) Можно применить схему Горнера вычисления значения многочлена в точке:

$$\begin{aligned} P(x) &= \alpha_m x^m + \alpha_{m-1}x^{m-1} + \alpha_{m-2}x^{m-2} + \ldots + \alpha_2 x^2 + \alpha_1 x + \alpha_0 = \\ &= (\alpha_m x^{m-1} + \alpha_{m-1}x^{m-2} + \alpha_{m-2}x^{m-3} + \ldots + \alpha_2 x + \alpha_1)x + \alpha_0 = \\ &= \big((\alpha_m x^{m-2} + \alpha_{m-1}x^{m-3} + \alpha_{m-2}x^{m-4} + \ldots + \alpha_2)x + \alpha_1\big)x + \alpha_0 = \\ &= \ldots = \Bigg(\ldots\Big(\big((0\cdot x + \alpha_m)x + \alpha_{m-1}\big)x + \alpha_{m-2}\Big)x + \ldots\Bigg)x + \alpha_0. \end{aligned}$$

То есть, чтобы вычислить значение многочлена в точке, можно, начав с нулевого значения аккумулятора, перебирать коэффициенты от старших к младшим (в нашем случае — с конца очередной строки к началу) и проводить итеративную процедуру: предыдущее значение аккумулятора умножается на значение x (в нашем случае p) и к результату прибавляется очередной коэффициент.

Вторая техническая тонкость заключается в том, что все вычисления надо проводить по модулю D: $u + v \to (u + v) \bmod D$, $u \cdot v \to (u \cdot v) \bmod D$. А поскольку значение D достаточно большое, то вычисления следует проводить в 8-байтном целом типе `__int64` (или его аналогах в других языках).

Наконец, третья особенность заключается в том, что разумно не делать два прохода по словам, проверяя, есть ли слова с требуемым хешем, а сразу начать считать минимальное и максимальное значение хеша, а в случае нахождения строки с требуемым значением переключаться на отбор слов с заданным хешем.

Идеи тестов:

1. Одно короткое слово (при вычислении хеша нет выхода за 4-байтный целый тип), хеш у него требуемый.
2. Одно короткое слово, хеш не совпадает с требуемым.
3. Одно длинное слово (при вычислении хеша есть выход за 4-байтный целый тип), хеш у него требуемый.
4. Одно длинное слово, хеш не совпадает с требуемым.
5. 100 коротких слов, есть одно слово с требуемым хешем, это слово первое в списке.
6. 100 коротких слов, есть одно слово с требуемым хешем, это слово не первое в списке.
7. 100 коротких слов, есть много слов с требуемым хешем, первое из таких слов первое в списке.
8. 100 коротких слов, есть много слов с требуемым хешем, первое из таких слов не первое в списке.
9. 100 коротких слов, нет слов с требуемым хешем.

10–14. То же, что в тестах 5–9, только слова длинные.

15–24. То же, что в тестах 5–14, только слов 1000.

25–29. То же, что в тестах 10–14, только слова полной длины 1000.

30–34. То же, что в тестах 24–29, только слов 1000.

35–42. Случайные тесты с короткими словами.

43–50. Случайные тесты с длинными словами.

10.4. «Перебираем числа». *Простая прямолинейная формулировка: есть набор из n целых положительных чисел. Нужно найти наименьшее общее кратное наибольших общих делителей всевозможных троек чисел из этого набора (в тройке числа*

берутся из разных позиций в наборе). Так как эта величина может быть очень большой, выдайте остаток от её деления на $10^9 + 7$.

Задача является достаточно сложной, поскольку лобовое решение, связанное с нахождением всех НОДов, а после — их НОКа, работает только для небольших количеств чисел в наборе — до 500-600. Однако, реализуется оно достаточно просто: $\text{НОД}(a, b, c) = \text{НОД}\big(\text{НОД}(a, b), c\big)$, $\text{НОК}(a_1, a_2, \ldots, a_n) = d \cdot (a_1/d) \cdot (a_2/d) \cdot \ldots \cdot (a_n/d)$, где $d = \text{НОД}(a_1, a_2, \ldots, a_n)$; НОДы находятся алгоритмом Евклида. При этом умножения при вычислении НОКа требуется делать по модулю $10^9 + 7$, что подразумевает использование 64-битного целого типа.

Для получения оптимального по скорости решения следует воспользоваться идеей разложения целого числа на простые множители. Если

$$a = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot \ldots \cdot p_m^{\alpha_m} \quad \text{и} \quad b = p_1^{\beta_1} \cdot p_2^{\beta_2} \cdot \ldots \cdot p_m^{\beta_m},$$

то

$$\text{НОД}(a, b) = p_1^{\min(\alpha_1, \beta_1)} \cdot p_2^{\min(\alpha_2, \beta_2)} \cdot \ldots \cdot p_m^{\min(\alpha_m, \beta_m)}$$

и

$$\text{НОК}(a, b) = p_1^{\max(\alpha_1, \beta_1)} \cdot p_2^{\max(\alpha_2, \beta_2)} \cdot \ldots \cdot p_m^{\max(\alpha_m, \beta_m)}.$$

Формулы не изменяются для вычисления НОД и НОК большего количества чисел.

Из этих формул видно, что действия ведутся независимо с каждым из простых делителей. Поэтому дальше будем рассматривать работу с одним делителем. Пусть задан набор чисел $\{a_i\}$, и простой делитель p у числа a_i имеет кратность α_i. Тогда у наибольших общих делителей $\text{НОД}(a_i, a_j, a_k)$ троек чисел делитель p будет иметь степень $\min(\alpha_i, \alpha_j, \alpha_k)$. А у НОДа всех НОКов делитель p будет иметь степень $\max\limits_{i,j,k} \min(\alpha_i, \alpha_j, \alpha_k)$. Так как рассматриваются все тройки чисел, то в частности будут рассмотрена такая тройка, у чисел которой показатели степени p — три наибольших величины среди всех чисел a_i. Отсюда понятно, что $\max\limits_{i,j,k}$ будет равен третьей величине в упорядоченном по убыванию ряду степеней делителя p среди всех чисел a_i. Впрочем, для нахождения этой величины не требуется сортировка, достаточно модифицировать алгоритм поиска максимума.

Теперь встаёт вопрос разложения чисел на простые множители и хранения информации о степенях тех или иных множителей в этих разложениях. Первым полезным классическим наблюдением здесь является то, что если $a = b \cdot c$ и $b \leqslant c$, то $b \leqslant \sqrt{a} \leqslant c$. Соответственно, для поиска младших делителей нужно перебирать простые числа от 2 до $\sqrt{3 \cdot 10^6} \approx 1732$. Таких чисел 272 штуки, их можно предпросчитать и просто вписать в текст программы. Тогда для каждого числа a_i эти простые делители перебираются, для каждого находится его степень в числе a_i; если после выделения всех делителей, не превосходящих 1742, остаётся число отличное от 1, то это гарантированно простое число, превосходящее 1742, и его тоже нужно учесть.

Для хранения троек наибольших степеней можно прямолинейно использовать массив байтов размером $3\,000\,000 \times 3$ (байтов, поскольку степень простых делителей не

может быть больше 22: $2^{22} = 4194304 > 3000000$ — и для её хранения достаточно байта). При этом, конечно, большинство ячеек, соответствующих составным числам использованы не будут. Также для хранения этой информации можно использовать словарь, в котором ключом является целое число, простой делитель, а значением — тройка его наибольших степеней.

После того, как все числа обработаны и вычислены тройки наибольших степеней каждого из возможных простых делителей, вычисляем НОК: идём по всем ячейкам массива (или элементам словаря) и для каждого числа, у которого третья из наибольших степеней отлична от нуля, умножаем (по модулю $10^9 + 7$) результат на соответствующий простой множитель, возведённый в эту степень. Здесь для некоторого ускорения работы можно применить алгоритм быстрого возведения в степень.

В итоге имеем алгоритм со сложностью $O(n)$, правда, операция с каждым числом (разложение на простые множители) весьма времяёмкая.

Идеи тестов:

1. Три совпадающих небольших простых числа.
2. Три совпадающих больших простых числа.
3. 100 совпадающих небольших простых числа.
4. 100 совпадающих больших простых числа.
5. 13000 совпадающих небольших простых числа.
6. 13000 совпадающих больших простых числа.

7–12. Те же идеи, что и в тестах 1–6, только числа *полупростые*, являющиеся произведением двух простых.

13–18. Те же идеи, что и в тестах 1–6, только имеются три полупростых числа, полученных разной комбинацией из трёх простых делителей.

19–26. Случайные тесты: 100 чисел до 2000.

27–34. Случайные тесты: 100 чисел до до $3 \cdot 10^6$.

35–42. Случайные тесты: до 13000 чисел до 2000.

43–50. Случайные тесты: до 13000 чисел до до $3 \cdot 10^6$.

10.5. «Ещё о паролях». *Петя Торопыжкин нашёл свой старый архив, но не может вспомнить пароль, чтобы распаковать его. Всё, что он помнит: пароль непуст, состоит только из заглавных символов латиницы, имеет длину не более l символов, не содержит двух одинаковых символов подряд и в смысле лексикографического порядка расположен между двумя известными строками s_1 и s_2, строго больше s_1 и строго меньше s_2. Ему нужно посчитать, сколько таких строк существует (чтобы понять, а стоит ли вообще пытаться перебирать их). Посчитайте и вы. Но поскольку это число может быть очень большим, выдайте остаток от деления его на $10^9 + 7$.*

Данная задача задумывалась программным комитетом как сложная. Идея решения — лобовой подсчёт количества требуемых строк — возникает здесь достаточно несложно, однако детальная разработка алгоритма является нетривиальной.

Обозначим символы строки s_1 через α_i, а s_2 — β_j: $s_1 = \alpha_1\alpha_2\ldots\alpha_n$, $s_2 = \beta_1\beta_2\ldots\beta_m$; n — длина s_1, m — длина s_2.

Можно разделить две различных ситуаций:

1) s_1 и s_2 имеют общее начало;

2) s_1 и s_2 не имеют общего начала.

Разделение этих случаев и нахождение длины общего начала s_1 и s_2 (если оно есть) производится в процессе их лексикографического сравнения.

Первый случай сводится ко второму отбрасыванием общего начала s_1 и s_2 и соответствующим уменьшением максимальной длины l рассматриваемых строк.

Ответ во втором случае складывается из трёх компонент:

а) строки, имеющие общее начало с s_1 и бóльшие, чем s_1;

б) строки, начинающиеся с символов, бóльших, чем первый символ s_1, но меньших, чем первый символ s_2 (таких строк может вообще не быть, если $\beta_1 = \alpha_1 + 1$);

в) строки, имеющие общее начало с s_2 и меньшие, чем s_2.

Ответ в пункте б) есть

$$A \cdot (1 + 25 + 25^2 + \ldots + 25^{l-1}) = A \cdot S(l-1),$$

где A — количество возможных первых символов, то есть символов, больших α_1, но меньших β_1, а

$$S(k) = \sum_{i=0}^{k} 25^i, \qquad 0 \leqslant k \leqslant l.$$

Величины $S(k)$ будут нужны и для дальнейшей части решения, поэтому разумно их предпросчитать перед началом алгоритма.

Появление числа 25 поясняется в примечании к примеру в условии задачи. Величина $S(k)$ есть сумма геометрической прогрессии, однако нельзя воспользоваться формулой, получив выражение $(25^k - 1)/(25 - 1)$, поскольку все вычисления производятся по модулю числа $10^9 + 7$ и осуществление операции деления в этом случае затруднено.

Ответ в пункте а) складывается из строк, которые имеют всю строку s_1 своим началом (они автоматически больше, чем s_1), и строк, имеющих с s_1 общее начало длины, меньшей длины s_1. Строк первого вида с длиной $n + 1$ существует 25 штук, с длиной $n + 2$ — 25^2 штук и т.д., то есть всего $25 + 25^2 + \ldots + 25^{l-n} = S(l-n) - 1$ (при $n = l$ как раз получается ноль).

Теперь посчитаем количество строк, имеющих с s_1 общее начало с длиной, меньшей длины s_1. Каждое значение индекса i от 2 до n определяет первый символ, который будет различаться у s_1 и конструируемой строки. Проводя рассуждение, аналогичное пункту б), получаем, что число таких строк равно

$$B_i \cdot (1 + 25 + 25^2 + \ldots + 25^{l-i}) = B_i \cdot S(l-i),$$

где B_i — количество символов, больших α_i.

Итак, ответ к пункту а) есть

$$\big(S(l-n)-1\big)+\sum_{i=2}^{n} B_i \cdot S(l-i).$$

Ответ к пункту в) ищется, исходя из похожих рассуждений. Строка, начинающаяся с символа β_1 и меньшая s_2, должна для какого-то i, $2 \leqslant i \leqslant m$, иметь i-й символ, меньший соответствующего символа β_i строки s_2 и произвольный хвост без повторяющихся символов. Ответ получается

$$\sum_{i=2}^{m} C_i \cdot S(l-i),$$

где C_i — количество символов, меньших β_i.

Итак, в третьем случае общий ответ вычисляется как

$$\big(S(l-n)-1\big)+\sum_{i=2}^{n} B_i \cdot S(l-i) \quad + \quad A \cdot S(l-1) \quad + \quad \sum_{i=2}^{m} C_i \cdot S(l-i),$$

где $A = \texttt{ord}(\beta_1) - \texttt{ord}(\alpha_1) - 1$, $B_i = \texttt{ord('Z')} - \texttt{ord}(\alpha_i)$, $C_i = \texttt{ord}(\beta_i) - \texttt{ord('A')}$; $\texttt{ord}(\cdot)$ — функция взятия кода символа (для Паскаля) или сам символ (для C/C++/Java/C#). Из этой формулы следует, что сложность алгоритма есть $O(n+m)$.

Все вычисления в рамках этой формулы и при нахождении $S(k)$, конечно, ведутся по модулю 10^9+7.

<u>Идеи тестов:</u>

1. Короткие строки, общего начала нет, $s_2 > s_1$, $\beta_1 = \alpha_1 + 1$.
2. Короткие строки, общего начала нет, $s_2 > s_1$, $\beta_1 > \alpha_1 + 1$.
3. Короткие строки, есть общее начало, $s_2 > s_1$, $\beta_i = \alpha_i + 1$.
4. Короткие строки, есть общее начало, $s_2 > s_1$, $\beta_i > \alpha_i + 1$.
5. Короткие строки, s_1 — начало s_2, $s_2 > s_1$.

6–10. То же, что в тестах 1–5, строки длиной 1000.

11–15. То же, что в тестах 1–5, строки длинные.

16–25. Случайные тесты.

Муниципальный этап Всероссийской олимпиады школьников по информатике в 2017 – 2018 учебном году

11 класс

Время выполнения задач — 4 часа

Ограничение по времени — 2 секунды на тест

Ограничение по памяти — 256 мегабайт

11.1. «Фантастический роман». В фантастическом романе, который пишет Петя Торопыжкин, инопланетные существа используют алфавит, состоящий из двух символов ⌔ и ⦶, которые в рабочем варианте текста Петя представляет заглавными буквами `F` и `G` (для простоты). Петя даже составил словарь языка этих существ. Для быстроты поиска по словарю он выбрал целое число p и сопоставил каждому слову $w = \alpha_0\alpha_1\ldots\alpha_k$ целое число $h(w) = \sum_{i=0}^{k} a_i \cdot p^i$, где коэффициент a_i равен 0, если $\alpha_i = \texttt{F}$, и 1, если $\alpha_i = \texttt{G}$. Однако такое число может быть большим, поэтому Петя запоминает остаток от деления $h(w)$ на число $D = 10^9 + 7$.

Такое число называется *хешем* слова w, а правило вычисления хеша — *хеш-функцией*. Вычислив один раз хеши слов из словаря, дальше очень просто проверять их на несовпадение: если хеши двух слов различаются, то и сами слова совпадать не могут. А вот если хеши двух слов совпадают (такую ситуацию называют *коллизией*), тогда для точной проверки эти слова надо сравнивать посимвольно.

Для работы со словарём нужна программа, по заданному слову находящую его хеш.

Формат входа: В первой строке задано целое число p, участвующее в вычислении хеша ($1 \leqslant p \leqslant 10^9$). Во второй строке задано слово w — непустая последовательность заглавных символов латиницы `F` и `G` длиной не более 1000 символов.

Формат выхода: Выведите единственное целое число — хеш слова w (остаток от деления $h(w)$ на D).

Пример 1		**Пример 2**	
Вход:	*Выход:*	*Вход:*	*Выход:*
`10` `FGFG`	`1010`	`10` `FGFGFGFGFGFGFG`	`10030303`

11.2. «Хитрая операция». Петя Торопыжкин придумал новую операцию над целыми числами: от числа отделяется последняя цифра его десятичной записи и к получившемуся после отделения числу прибавляется эта отделённая цифра, умноженная на сто. Если предыдущее число было однозначным, то после отделения его последней цифры получается ноль. Для заданного начального числа a выдайте результат, который получится после k применений придуманной Петей операции.

Формат входа: В единственной строке через пробел заданы два целых числа: a — начальный член вычисляемой последовательности — и k — количество применений операции ($0 \leqslant a, k \leqslant 2 \cdot 10^9$).

Формат выхода: Выдайте число, получаемое после k применений операций к заданному исходному числу.

Пример

Вход:	*Выход:*
123456789 4	12924

11.3. «Логистический центр». В большом городе с квадратной застройкой введена координатная система так, что все прямые $x = a$ и $y = b$ для целых a и b — это улицы, по которым возможно передвижение транспорта. На некоторых перекрёстках расположены магазины. Владелец сети магазинов решил разместить на каком-то перекрёстке (возможно, на том, где уже есть магазин) логистический центр так, чтобы сумма расстояний (при движении по улицам) от него до всех магазинов была наименьшей. Напишите программу, которая будет находить подходящее место.

Формат входа: В первой строке задано единственное целое число n — количество магазинов ($1 \leqslant n \leqslant 10^5$). В следующих n строках через пробел перечислены пары координат x_i, y_i магазинов ($|x_i|, |y_i| \leqslant 10^9$).

Формат выхода: Через пробел выведите координаты логистического центра и сумму расстояний от него до всех магазинов. Если наилучший результат может быть обеспечен размещением центра в более, чем одной точке, выведите любую из них.

Пример

Вход:	*Выход:*
5 0 0 4 2 1 4 0 0 2 2	1 2 12

11.4. «Дискретное показательное уравнение». Часто операции с числами ведутся по модулю какого-либо числа p, например, $a \uparrow_p b = a^b \bmod p$. Напишите программу, которая для заданных величин p и a будет находить все решения b_i и c_j уравнения $a \uparrow_p b_i = c_j \bmod p$, выбираемые из заданных множеств. Нужно разделить исходные множества чисел b и c на пары соответствующих друг другу наборов: любое число b и любое число c из пары наборов удовлетворяют указанному уравнению.

Формат входа: В первой строке через пробел заданы два целых числа p и a ($1 \leqslant a, p \leqslant 10^9$). Во второй строке задано целое число n — размер набора целых чисел $\{b_i\}$, которые через пробел перечислены в третьей строке. В четвёртой строке задано целое число m — размер набора целых чисел $\{c_j\}$, которые через пробел перечислены в пятой строке. Выполнены ограничения $1 \leqslant n, m \leqslant 10^5$; $0 \leqslant b_i, c_j \leqslant 10^9$.

Формат выхода: Результирующие пары наборов нужно вывести парами строк: в первой строке из пары содержится набор чисел b, во второй — набор чисел c. В начале каждой строки выводится количество чисел в соответствующем наборе. Если какому-то набору b не соответствует ни одного c или какому-то набору c не соответствует ни одного b, то для такого набора строка, представляющая парный набор, должна содержать только 0. Порядок перечисления чисел в наборе и порядок перечисления пар наборов могут быть любыми.

Пример

Вход:	*Выход:*
5 4 4 1 2 3 4 6 2 10 5 9 7 17	2 1 3 1 9 2 2 4 0 0 5 2 10 5 7 17

Примечание: $(4^3 \bmod 5 = 64 \bmod 5) = (4^1 \bmod 5 = 4 \bmod 5) = 9 \bmod 5$; $4^2 \bmod 5 = 16 \bmod 5 = 1$, $4^4 \bmod 5 = 256 \bmod 5 = 1$, ни одно из остальных чисел c_j не даёт остаток 1 при делении на 5.

11.5. «Ночёвка на трассе». Летом Петя Торопыжкин в составе компании из n друзей поучаствовал в мотопробеге в Европу (каждый участник ехал на своём мотоцикле). К концу одного из дней пробега, когда пришло время вставать на ночёвку, они были на платной дороге, состоящей из l отрезков. Для i-го участника пробега известен номер x_i отрезка дороги, на котором он находится. Стоимость проезда по k-му отрезку равна d_k, не зависит от направления проезда и взимается при въезде на этот отрезок; дальнейшее движение по отрезку не требует затрат. Во время подготовки к поездке было выяснено, что на этой дороге имеется m мотелей. Мотель с номером j расположен на отрезке дороги с номером y_j (на одном участке дороги может быть один мотель, несколько мотелей или не быть вовсе) и может вместить p_j постояльцев. Общая вместимость всех мотелей достаточна, чтобы все друзья могли заночевать под крышей. По имеющимся данным нужно определить минимальные затраты на дорожные сборы, чтобы каждый из друзей доехал до какого-нибудь мотеля.

Формат входа: В первой строке через пробел указаны целые числа l, n и m — количества участков дороги, участников мотопробега и мотелей ($1 \leqslant l, n, m \leqslant 5000$). Во второй строке через пробел перечислено l целых чисел d_k — стоимости проезда по участкам дороги в порядке следования их на дороге ($1 \leqslant d_k \leqslant 10000$). В третьей строке через пробел перечислено n целых чисел x_i — номера отрезков дороги, на которых были участники пробега в момент начала распределения на ночёвку ($1 \leqslant x_i \leqslant l$). В следующих m строках перечислены описания мотелей — перечисленные через пробел два целых числа y_j и c_j ($1 \leqslant y_j \leqslant l$, $1 \leqslant c_j \leqslant 5000$).

Формат выхода: Выведите единственное целое число — минимально возможные дорожные расходы, связанные с достижением мотелей всеми участниками.

Пример

Вход:	*Выход:*
3 5 4 50 20 10 3 1 1 2 1 2 1 3 5 1 2 3 1	30

Муниципальный этап Всероссийской олимпиады школьников по информатике в 2017 – 2018 учебном году
11 класс. Разбор решений и идеи тестов

11.1. «Фантастический роман». *В фантастическом романе, который пишет Петя Торопыжкин, инопланетные существа используют алфавит, состоящий из двух символов ⊰ и ⊱, которые в рабочем варианте текста Петя представляет заглавными буквами* `F` *и* `G` *(для простоты). Петя даже составил словарь языка этих существ. Для быстроты поиска по словарю он выбрал целое число p и сопоставил каждому слову $w = \alpha_0\alpha_1\ldots\alpha_k$ целое число $h(w) = \sum_{i=0}^{k} a_i \cdot p^i$, где коэффициент a_i равен 0, если $\alpha_i =$* `F`*, и 1, если $\alpha_i =$* `G`*. Однако такое число может быть большим, поэтому Петя запоминает остаток от деления $h(w)$ на число $D = 10^9 + 7$.*

Такое число называется хешем слова w, а правило вычисления хеша — хеш-функцией. Вычислив один раз хеши слов из словаря, дальше очень просто проверять их на несовпадение: если хеши двух слов различаются, то и сами слова совпадать не могут. А вот если хеши двух слов совпадают (такую ситуацию называют коллизией), тогда для точной проверки эти слова надо сравнивать посимвольно.

Для работы со словарём нужна программа, по заданному слову находящую его хеш.

По мнению программного комитета, данная задача является очень простой. От уровня утешительной её отделяет две технические тонкости.

Первая техническая тонкость заключается в том, что нужно считать степени числа p. Здесь имеются следующие методы:

1) Можно при обработке каждого символа хранить предыдущую степень p и на её основе вычислять следующую.

2) Можно применить алгоритм *быстрого возведения в степень*:

$$a^n = \begin{cases} n = 2k, & (a^2)^k, \\ n = 2k+1, & (a^2)^k \cdot a. \end{cases}$$

3) Можно применить схему Горнера вычисления значения многочлена в точке:

$$\begin{aligned} P(x) &= \alpha_m x^m + \alpha_{m-1}x^{m-1} + \alpha_{m-2}x^{m-2} + \ldots + \alpha_2 x^2 + \alpha_1 x + \alpha_0 = \\ &= (\alpha_m x^{m-1} + \alpha_{m-1}x^{m-2} + \alpha_{m-2}x^{m-3} + \ldots + \alpha_2 x + \alpha_1)x + \alpha_0 = \\ &= \big((\alpha_m x^{m-2} + \alpha_{m-1}x^{m-3} + \alpha_{m-2}x^{m-4} + \ldots + \alpha_2)x + \alpha_1\big)x + \alpha_0 = \\ &= \ldots = \Bigg(\ldots\Big(\big((0 \cdot x + \alpha_m)x + \alpha_{m-1}\big)x + \alpha_{m-2}\Big)x + \ldots\Bigg)x + \alpha_0. \end{aligned}$$

То есть, чтобы вычислить значение многочлена в точке, можно, начав с нулевого значения аккумулятора, перебирать коэффициенты от старших к младшим (в нашем случае — с конца очередной строки к началу) и проводить итеративную процедуру:

предыдущее значение аккумулятора умножается на значение x (в нашем случае p) и к результату прибавляется очередной коэффициент.

Вторая техническая тонкость заключается в том, что все вычисления надо проводить по модулю D: $u + v \to (u + v) \bmod D$, $u \cdot v \to (u \cdot v) \bmod D$. А поскольку значение D достаточно большое, то вычисления следует проводить в 8-байтном целом типе `__int64` (или его аналогах в других языках).

Тесты подобраны так, что участник, не использующий длинный целый тип получит около 60% баллов.

Идеи тестов:

1. Слово `F`, $p = 2$.
2. Слово `G`, $p = 2$.
3. Слово `F`, $p = 10^9$.
4. Слово `G`, $p = 10^9$.
5. Слово из 10 символов `F`, $p = 2$.
6. Слово из 10 символов `G`, $p = 2$.
7. Слово из 30 символов `F`, $p = 2$.
8. Слово из 30 символов `G`, $p = 2$.

9–12. Случайные тесты для $p = 2$, длина слова не превосходит 30.

13–17. Случайные тесты, слова длинные.

18–20. Слова максимальной длины, различные p.

11.2. «Хитрая операция». *Петя Торопыжкин придумал новую операцию над целыми числами: от числа отделяется последняя цифра его десятичной записи и к получившемуся после отделения числу прибавляется эта отделённая цифра, умноженная на сто. Если предыдущее число было однозначным, то после отделения его последней цифры получается ноль. Для заданного начального числа a выдайте результат, который получится после k применений придуманной Петей операции.*

Задача, хотя и кажется очень простой, на самом деле содержит подвох: номер члена настолько велик, что последовательное вычисление членов последовательности не уложится в ограничения по времени. Однако несложное математическое рассуждение показывает, что долго считать не нужно.

Действительно, если число трёхзначное $\overline{a_1a_2a_3}$, то указанная операция просто переставляет последнюю цифру вперёд: $\overline{a_1a_2a_3} \to \overline{a_1a_2} + 100a_3 = \overline{a_3a_1a_2}$. Стало быть, если мы получили какое-то трёхзначное число $\overline{a_1a_2a_3}$ на шаге $m < k$, то с учётом того, что числа через три начнут повторяться, можно утверждать, что
1) если $(k - m) \bmod 3 = 0$, результат есть $\overline{a_1a_2a_3}$;
2) если $(k - m) \bmod 3 = 1$, результат есть $\overline{a_3a_1a_2}$;
3) если $(k - m) \bmod 3 = 2$, результат есть $\overline{a_2a_3a_1}$.

Остаётся вопрос: а дойдём ли мы до трёхзначного числа? Ответ — да. Если число

большое. то данная операция его уменьшает:

$$\overline{\alpha\beta}\ (\beta \in 0..9, \alpha \geqslant 1) \to \alpha + 100\beta;$$
$$\overline{\alpha\beta} - (\alpha + 100\beta) = (10\alpha + \beta) - (\alpha + 100\beta) = 9\alpha - 99\beta = 9(\alpha - 11\beta).$$

Если $\alpha > 99 \geqslant 11\beta$, то новое число будет меньше старого. А неравенство $\alpha > 99$ как раз и означает, что число $\overline{\alpha\beta}$ более чем трёхзначное. Вывод: четырёхзначные и более длинные числа при применнии данной операции уменьшаются, и мы обязательно дойдём до трёхзначного числа. Причём количество операций до достижения трёхзначного числа будет не слишком велико — не более 10–15: на каждом шаге число на 1 уменьшает свою разрядность.

Тесты подобраны так, что программа, построенная на «лобовом» вычислении получит около 45% баллов.

Идеи тестов:

1–4. $a = 0$, $k = 0$ / 10 / 10000 / 10^9.
5–8. $a = 10$, $k = 0$ / 10 / 10000 / 10^9.
9–12. $a = 123$, $k = 0$ / 10 / 10000 / 10^9.
13–15. Случайные тесты, a и k таковы, что мы не доходим до трёхзначного числа.
16–25. Случайные тесты, $k = 10^9$.

11.3. «Логистический центр». *В большом городе с квадратной застройкой введена координатная система так, что все прямые $x = a$ и $y = b$ для целых a и b — это улицы, по которым возможно передвижение транспорта. На некоторых перекрёстках расположены магазины. Владелец сети магазинов решил разместить на каком-то перекрёстке (возможно, на том, где уже есть магазин) логистический центр так, чтобы сумма расстояний (при движении по улицам) от него до всех магазинов была наименьшей. Напишите программу, которая будет находить подходящее место.*

Сложность этой задачи, по мнению программного комитета, является средней. С одной стороны, при знании специальных алгоритмов эта задача может быть быстро решена. С другой стороны, имеется путь, не требующий специальных знаний, но для его реализации необходимо провести математический анализ задачи.

Лобовое решение, связанное с последовательным перебором точек, возможных положений центра, в случае широко расположенных точек не будет укладываться в ограничения по времени.

Оптимальное решение, требующее знаний специальных алгоритмов, основывается на математическом рассмотрении сути задачи.

Расстояние между двумя точками, используемое в задаче, часто называется *манхеттенским* и вычисляется по формуле $d_1(a, b) = |a_x - b_x| + |a_y - b_y|$. Для фиксированной точки a и переменной второй точки (x, y) для расстояния между ними имеем выражение $f_a(x, y) = |x - a_x| + |y - a_y|$. Данная функция является *выпуклой* в том смысле,

что её надграфик — выпуклое множество. *Надграфиком* функции $f(x,y)$ двух переменных называют множество в трёхмерном пространстве $\operatorname{epi} f = \big\{(x,y,z) : z \geqslant f(x,y)\big\}$. Множество называется *выпуклым*, если вместе с любыми двумя своими точками оно содержит и отрезок с концами в этих точках.

Функция, которую нужно минимизировать есть сумма манхеттенских расстояний:

$$F(x,y) = \sum_{i=1}^{n} f_{a_i}(x,y) = \sum_{i=1}^{n} \big(|x - a_{i,x}| + |y - a_{i,y}|\big) \to \min_{(x,y)}.$$

Не очень сложно доказывается, что сумма любого конечного числа выпуклых функций является выпуклой функцией. То есть можно утверждать, что минимизируемая функция $F(x,y)$ выпукла.

Также из выпуклости функции двух переменных следует, что сужение этой функции на любую прямую вида $y = y^*$ является выпуклой функцией $F(x,y^*)$ одного аргумента x. При этом от минимизации по двум переменным можно перейти к *повторной* минимизации:

$$\min_{(x,y)} F(x,y) = \min_{y} \min_{x} F(x,y) = \min_{y} g(y). \qquad (*)$$

Наконец, выпуклость функции $F(x,y)$ влечёт выпуклость функции минимумов по первому аргументу $g(y) = \min\limits_{x} F(x,y)$.

Рассуждения о выпуклости различных функций важны, поскольку выпуклая функция одного переменного является *унимодальной*, то есть слева она имеет участок строгого убывания, затем отрезок (возможно, одноточечный), где функция принимает своё минимальное значения, после чего имеется участок строгого возрастания. Участки убывания и/или возрастания могут отсутствовать. Кроме того, важно, что функция не имеет участков постоянства, кроме участка минимума. В силу такого хорошего устройства унимодальные функции допускают очень быстрый алгоритм приближенного поиска минимума, который называется *тернарный (или троичный) поиск.*

Суть этого алгоритма в случае минимизации унимодальной функции по вещественному аргументу заключается в следующем. Пусть в начале нам указан отрезок $[\alpha_0, \beta_0] = [\alpha, \beta]$, на котором надо найти минимум унимодальной функции h. На каждой итерации алгоритма по имеющемуся отрезку $[\alpha_k, \beta_k]$ вычисляются две точки: $p_1 = (2\alpha_k + \beta)/3$ и $p_2 = (\alpha_k + 2\beta)/3$, делящие отрезок на три равные части. Затем сравниваются значения функции h в точках p_1 и p_2, и рассматриваемый отрезок сужается:

1. если $h(p_1) > h(p_2)$, то в качестве нового отрезка берём $[\alpha_{k+1}, \beta_{k+1}] = [p_1, b]$;
2. если $h(p_1) < h(p_2)$, то в качестве нового отрезка берём $[\alpha_{k+1}, \beta_{k+1}] = [a, p_2]$;
3. случай $h(p_1) = h(p_2)$ можно включить в любой из рассматриваемых случаев или же положить в этом случае $[\alpha_{k+1}, \beta_{k+1}] = [p_1, p_2]$.

Действительно, в первом случае из-за того, что $h(p_1) > h(p_2)$, отрезок $[p_1, p_2]$ пересекается с участком убывания функции h, и, значит, минимум находится правее точки p_1. Соответственным образом сужаем отрезок. Наоборот, во втором случае отрезок $[p_1, p_2]$ пересекается с участком возрастания функции, и минимум расположен левее точки p_2.

Данная операция повторяется до тех пор, пока длина отрезка $[\alpha_k, \beta_k]$ не станет меньше требуемой точности отыскания точки минимума. В качестве приближения точки минимума можно выдать любую точку этого маленького отрезка, например, середину или один из концов.

В случае поиска минимума по целому аргументу деления при вычислении точек p_1 и p_2 выполняются нацело, и операция повторяется до тех пор, пока длина отрезка не станет равной 2 (и, соответственно, перестанет содержать две целых внутренних точки). Из оставшихся трёх точек — концов отрезка и его середины — прямым сравнением находится точка минимума.

Видно, что на каждом шаге длина отрезка сокращается в полтора раза, а значит сложность тернарного поиска есть $O(\log l)$, где l — длина исходного отрезка.

Таким образом, вычисление минимума (*) связано с вычислением минимума унимодальной функции $g(y)$, осуществляемым при помощи тернарного поиска. При этом значение $g(y)$ для каждого требуемого значения y^* вычисляется как минимум унимодальной функции $F(x, y^*)$, что также производится при помощи тернарного поиска. Общая сложность алгоритма с учётом того, что для вычисления значения $F(x, y)$ требуется вычислить и просуммировать n слагаемых, есть $O(n \cdot \log \Delta x \cdot \log \Delta y)$, где Δx и Δy — размеры прямоугольника, на котором ищется минимум функции двух переменных.

Однако есть путь решения, не требующий знания указанного алгоритма поиска минимума функции. Рассмотрим минимизируемую функцию:

$$F(x, y) = \sum_{i=1}^{n} f_{a_i}(x, y) = \sum_{i=1}^{n} \big(|x - a_{i,x}| + |y - a_{i,y}|\big) = \sum_{i=1}^{n} \big(|x - a_{i,x}|\big) + \sum_{i=1}^{n} \big(|y - a_{i,y}|\big).$$

Во-первых, мы видим, что выражение распалось на два слагаемых, одно из которых зависит только от координаты x подбираемой точки, а второе — только от координаты y. Это значит, что мы можем независимо работать с подбором x и с подбором y. Такая ситуация была бы невозможной, если бы рассматривалось расстояние, в котором переменные были бы связаны, например, обычное евклидово расстояние $d_2(a, b) = \sqrt{(a_x - b_x)^2 + (a_y - b_y)^2}$ или «бесконечное» («шахматное») расстояние $d_\infty(a, b) = \max\big\{|a_x - b_x|, |a_y - b_y|\big\}$.

Рассмотрим теперь поведение каждого из этих слагаемых на примере первого (второе ведёт себя аналогично). Для начала рассмотрим ситуацию с двумя модулями: $g(x) = |x - \alpha| + |x - \beta|$. Если $\alpha = \beta$, то имеем функцию $g(x) = 2|x - \alpha|$, имеющую минимум при $x = \alpha$; или можно формально сказать, что при $x \in [\alpha, \beta]$. Если же $\alpha \neq \beta$ (для определённости, пусть $\alpha < \beta$), то форма этой функции известна: убывание при $x < \alpha$, плато при $x \in [\alpha, \beta]$, возрастание при $x > \beta$. Значит, такая функция в этом случае принимает минимальные значения также при $x \in [\alpha, \beta]$.

Вернёмся теперь обратно к большому количеству точек: $h(x) = \sum_{i=1}^{m} |x - \alpha_i|$. Пусть значения α_i упорядочены по возрастанию: $\alpha_1 \leqslant \alpha_2 \leqslant \alpha_3 \leqslant \ldots \leqslant \alpha_{m-1} \leqslant \alpha_m$. Рассмотрим два случая.

Первый — $m = 2k$, число чётное. Тогда мы можем разбить числа на пары (α_1, α_m), (α_2, α_{m-1}), (α_3, α_{m-2}), ... Сумма h разобьётся на пары модулей, как это было у функции g, причём плато пары слагаемых с меньшим номером первого из чисел α_i охватывает плато пары с бóльшим номером первого из чисел α_i:

$$[\alpha_1, \alpha_m] \supset [\alpha_2, \alpha_{m-1}] \supset [\alpha_3, \alpha_{m-2}] \supset \ldots$$

Значит, минимума функция h будет достигать на плато $[\alpha_{m/2}, \alpha_{m/2+1}]$ самой последней пары чисел $(\alpha_{m/2}, \alpha_{m/2+1})$.

Второй случай — $m = 2k + 1$, число нечётное. Тогда аналогичными рассуждениями можно понять, что минимум функция h получает в точке $\alpha_{(m+1)/2}$ — средней точке отсортированного набора α_i.

В математике значение, разделяющее некоторый набор величин пополам, называется *медианой* (это значение не обязательно принадлежит набору). Таким образом, задача минимизации функции h сводится к поиску медианы набора $\{\alpha_i\}$. Этот поиск производится посредством сортировки: набор $\{\alpha_i\}$ сортируется по возрастанию и, если количество чисел в нём нечётно, берётся средняя точка отсортированного набора, а если чётно, то любая точка из отрезка между двумя средними точками в отсортированном наборе.

Таким образом, задача может быть решена следующим образом: ищем x^* — медиану набора $\{a_{i,x}\}$, ищем y^* — медиану набора $\{a_{i,y}\}$, считаем значение функции $F(x^*, y^*)$, которое будет минимальным. Выдаём ответ.

Сложность этого решения $O(n \log n)$ — нужно провести две сортировки массивов объёмом n. Потенциально это работает быстрее в сравнении с тернарным поиском.

<u>Идеи тестов:</u>

1. Один магазин.
2. Два совпадающих магазина.
3. 100 совпадающих магазинов.
4. 10^5 совпадающих магазинов.
5. Три магазина, два совпадают.
6. 100 магазинов, 99 совпадают.
7. 10^5 магазинов, 99999 совпадают.
8. 4 магазина — две группы по 2 совпадающих.
9. 100 магазинов — две группы по 50 совпадающих.
10. 10^5 магазинов — две группы по 50000 совпадающих.
11. Небольшой квадрат со сторонами, параллельными осям, по 1 магазину в каждой вершине.
12. Небольшой квадрат со сторонами, параллельными осям, по 25 магазинов в каждой вершине.
13. Небольшой квадрат со сторонами, параллельными осям, по 25000 магазинов в каждой вершине.

14. Большой квадрат со сторонами, параллельными осям, по 1 магазину в каждой вершине.
15. Большой квадрат со сторонами, параллельными осям, по 25 магазинов в каждой вершине.
16. Большой квадрат со сторонами, параллельными осям, по 25000 магазинов в каждой вершине.
17. Небольшой квадрат со сторонами, наклонёнными под 45°, по 1 магазину в каждой вершине.
18. Небольшой квадрат со сторонами, наклонёнными под 45°, по 25 магазинов в каждой вершине.
19. Небольшой квадрат со сторонами, наклонёнными под 45°, по 25000 магазинов в каждой вершине.
20. Большой квадрат со сторонами, наклонёнными под 45°, по 1 магазину в каждой вершине.
21. Большой квадрат со сторонами, наклонёнными под 45°, по 25 магазинов в каждой вершине.
22. Большой квадрат со сторонами, наклонёнными под 45°, по 25000 магазинов в каждой вершине.

23–29. Случайные тесты: мало магазинов, сосредоточенных в небольшом квадрате.
30–36. Случайные тесты: много магазинов, сосредоточенных в небольшом квадрате.
37–43. Случайные тесты: мало магазинов, сосредоточенных в большом квадрате.
44–50. Случайные тесты: много магазинов, сосредоточенных в большом квадрате.

Видно, что лобовой алгоритм получит 40–50% от полного балла.

11.4. «Дискретное показательное уравнение». *Часто операции с числами ведутся по модулю какого-либо числа p, например, $a \uparrow_p b = a^b \bmod p$. Напишите программу, которая для заданных величин p и a будет находить все решения b_i и c_j уравнения $a \uparrow_p b_i = c_j \bmod p$, выбираемые из заданных множеств. Нужно разделить исходные множества чисел b и c на пары соответствующих друг другу наборов: любое число b и любое число c из пары наборов удовлетворяют указанному уравнению.*

Данная задача, по мнению программного комитета, является достаточно сложной, поскольку для своего оптимального решения требует знания специфических структур данных.

Идейно задача не слишком сложна. Нужно сгруппировать значения b_i с одинаковыми значениями $a \uparrow_p b_i$ (то есть с одинаковыми остатками $a^{b_i} \bmod p$), для чего для каждого b_i надо вычислить это значение. Понятно, что лобовое возведение в степень на основе многократного умножения не пройдёт по времени для указанных ограничений на b_i. Здесь поможет алгоритм *быстрого возведения в степень*:

$$a^n = \begin{cases} n = 2k, & (a^2)^k, \\ n = 2k+1, & (a^2)^k \cdot a. \end{cases}$$

На каждой итерации степень уменьшается вдвое, так что сложность одного такого возведения в степень есть $O(\log B)$, где B — ограничение на величину степени.

Второй вопрос: как быстро группировать каждое очередное значение b_i, для которого вычислено значение $a \uparrow_p b_i$. Из-за того, что p слишком велико, невозможно хранить списки b_i для каждого возможного значения остатка от деления на p. Здесь полезной будет структура данных под названием *словарь* (или *ассоциативный массив*). Она хранит значения, для доступа к которым используется индекс, также называемый *ключом*, вообще говоря, не являющийся последовательными целыми числами, а берущий свои значения из любого множества, на котором определён порядок (то есть имеется операция сравнения больше-меньше). Эта структура позволяет добавить, удалить элемент, а также проверить наличие элемента с заданным ключом (а следовательно, и получить доступ к такому элементу) за время $O(\log n)$, где n — количество элементов в хранилище. В языках программирования соответствующий тип называется `map`, или `Dictionary`, или `SortedDictionary`.

Соответственно, в нашем случае разумно воспользоваться словарём, ключом в котором является остаток от деления на p, а элемент хранит два списка (массива): список тех b_i, которые дают этот остаток при вычислении $a \uparrow_p b_i$, и список c_j, которые дают этот остаток при делении на p. После группировки всех b_i и c_j можно, перебирая элементы словаря, вывести соответствующие друг другу наборы чисел b и c.

Общая сложность такого алгоритма есть $O\big(n \log B + (n+m)\log(n+m)\big)$. Первое слагаемое есть время вычисления $a \uparrow_p b_i$ для всех b_i, второе — время помещения всех чисел b_i и c_j в словарь, а потом извлечение их оттуда и вывод.

Идеи тестов:

1. $m = n = 1$, $b_1 = 10$, нет решений.
2. $m = n = 1$, $b_1 = 10$, есть решение.
3. $m = n = 1$, $b_1 = 10^9$, нет решений.
4. $m = n = 1$, $b_1 = 10^9$, есть решение.
5. $m = n = 100$, $B = 10$, все b имеют парные c, все c имеют парные b.
6. $m = n = 100$, $B = 10$, есть значения b без пары c, все c имеют парные b.
7. $m = n = 100$, $B = 10$, все b имеют парные c, есть значения c без пары b.
8. $m = n = 100$, $B = 10$, есть значения b без пары c, есть значения c без пары b.

9–12. То же, что в тестах 5–8, только $m = 20000$.

13–16. То же, что в тестах 5–8, только $m = n = 20000$.

17–20. То же, что в тестах 5–8, только $m = 20000$, $B = 10^9$.

21–24. То же, что в тестах 5–8, только $m = n = 20000$, $B = 10^9$.

25. Максимальный тест: $n = m = 10^5$, $B = 10^9$.

11.5. «Ночёвка на трассе». *Летом Петя Торопыжкин в составе компании из n друзей поучаствовал в мотопробеге в Европу (каждый участник ехал на своём мотоцикле). К концу одного из дней пробега, когда пришло время вставать на ночёвку,*

они были на платной дороге, состоящей из l отрезков. Для i-го участника пробега известен номер x_i отрезка дороги, на котором он находится. Стоимость проезда по k-му отрезку равна d_k, не зависит от направления проезда и взимается при въезде на этот отрезок; дальнейшее движение по отрезку не требует затрат. Во время подготовки к поездке было выяснено, что на этой дороге имеется m мотелей. Мотель с номером j расположен на отрезке дороги с номером y_j (на одном участке дороги может быть один мотель, несколько мотелей или не быть вовсе) и может вместить p_j постояльцев. Общая вместимость всех мотелей достаточна, чтобы все друзья могли заночевать под крышей. По имеющимся данным нужно определить минимальные затраты на дорожные сборы, чтобы каждый из друзей доехал до какого-нибудь мотеля.

По мнению программного комитета данная задача является очень сложной, поскольку для своего решения требует весьма нетривиального применения принципа динамического программирования и достаточно необычных структур данных.

Обозначим через $(a, b]$ совокупность всех целых чисел, лежащих между числами a и b, включая b и исключая a. При $a < b$ имеем $(a, b] = \{a+1, a+2, \ldots, b-1, b\}$; при $a > b$ получаем $(a, b] = \{b, b+1, \ldots, a-2, a-1\}$; при $a = b$ положим $(a, b] = \varnothing$. Тогда стоимость переезда с участка дороги с номером a на участок дороги с номером y, рассмотренная как функция от y, есть

$$f_a(y) = \sum_{k \in (a,y]} d_k. \qquad (*)$$

Перед началом работы алгоритма отсортируем мотели и участников по возрастанию номеров участков дороги, на которых они находятся. Если несколько мотелей (или участников) находятся на одном участке дороги, то среди них нас устроит любое упорядочение. В дальнейшем, когда говорится о переборе объектов по возрастанию или по убыванию, имеется в виду именно этот порядок.

Пусть $\mathtt{dp}[i, j]$ — минимальные суммарные затраты первых i участников, которые разместились в первых j мотелях. (Каждая ячейка массива $\mathtt{dp}$ — 8-байтное целое число.) Если такое размещение невозможно, то есть если суммарная вместимость первых j мотелей меньше i, полагаем эту величину равной $+\infty$ (каким-то числом, заведомо бóльшим рассматриваемых затрат, например, $2 \cdot 10^{18}$); точнее, в процессе вычислений эта ячейка будет оставаться бесконечной. При этом считаем, что какое-то ненулевое количество последних по счету участников (хотя бы i-й) разместилось в j-м мотеле.

В качестве начальных условий считаем $\mathtt{dp}[0, j] = 0$ для всех $0 \leqslant j \leqslant m$: размещение нуля участников в любом числе мотелей не стоит ничего. Также считаем $\mathtt{dp}[i, 0] = +\infty$ для всех $1 \leqslant i \leqslant n$: размещение ненулевого количества участников в нуле мотелей невозможно. Остальные ячейки массива $\mathtt{dp}$ также положим равными $+\infty$.

Пересчёт на шаге динамического программирования:

$$\mathtt{dp}[i, j] = \min_{1 \leqslant p \leqslant \min\{c_j, i\}} \Big\{ \sum_{k=i-p+1}^{i} f_k(y_j) + \min_{0 \leqslant r \leqslant j-1} \mathtt{dp}[i-p, r] \Big\}. \qquad (**)$$

Смысл этого выражения в следующем. Мы ищем минимальное по стоимости размещение первых i участников по первым j мотелям так, чтобы в j-м мотеле хоть кто-то жил. Это внешний минимум. Переменная p, по которой этот минимум ищется — это количество участников, заселяемых в j-й мотель: она строго больше 0 — кто-то в j-м мотеле жить должен — и не превосходит ни вместимости j-го мотеля c_j, ни общего числа размещаемых участников i.

Минимизируемое выражение в фигурных скобках имеет следующий смысл. Первое слагаемое — суммарные затраты p последних участников на доезд до j-го мотеля, куда их селят. Второе слагаемое — наиболее дешёвое размещение остальных участников по первым $(j-1)$-му мотелю.

Пересчёт в (**) ведётся внешним циклом по j и для каждого j для всех i. После того, как цикл по j от 1 до m завершён, ответом будет $\min\big\{\mathtt{dp}[n,j] : 1 \leqslant j \leqslant m\big\}$.

Сложность «лобовой» реализации этого вычисления следующая: мы проводим вычисления для каждой из $n \cdot m$ ячеек массива $\mathtt{dp}$; каждое вычисление внешнего минимума требует $O(C)$ операций ($C = \max c_j$), каждая из которых требует C операций для вычисления суммы в первом слагаемом, причём $f_k(y_j)$ тоже требует до l сложений, и m операций для поиска минимума во втором слагаемом. То есть общая сложность есть $O\big(nmC(Cl+m)\big)$, что слишком много в указанных ограничениях. Лобовое вычисление нужно оптимизировать.

Достаточно просто оптимизируется вычисление выражения внутри минимума. Оптимизация второго слагаемого производится хранением дополнительного массива $\mathtt{MinDP}[i] = \min\big\{\mathtt{dp}[i,r] : 0 \leqslant r \leqslant j\big\}$. После того, как для какого-то очередного значения j элементы массива $\mathtt{dp}[i,j]$ просчитаны для всех i, делается ещё один проход по i и вычисляется $\mathtt{MinDP}[i] = \min\big\{\mathtt{MinDP}[i], \mathtt{dp}[i,j]\big\}$. Это снижает сложность вычисления второго слагаемого до константы: для $j-1$ значения максимума просчитаны и помещены в массив $\mathtt{MinDP}$.

Вычисление слагаемых $f_k(y_j)$ есть суммирование отрезка массива $d[j]$. Весьма известным методом оптимизации этой операции является введение нового массива

$$\mathtt{D}[j] = \sum_{k=1}^{j} d_k; \text{ рекурсивное определение: } \mathtt{D}[1] = d_1,\ \mathtt{D}[k+1] = \mathtt{D}[k] + d_{k+1},$$

который называется массивом *префиксных сумм набора* d_k. Тогда (*) можно переписать в виде

$$f_a(y) = \sum_{k \in (a,y]} d_k = \begin{cases} y > a, & d_{a+1} + d_{a+2} + \ldots + d_y = \mathtt{D}[y] - \mathtt{D}[a], \\ y = 0, & 0, \\ y < a, & d_{a-1} + d_{a-2} + \ldots + d_y = \mathtt{D}[a-1] - \mathtt{D}[y-1]. \end{cases}$$

Здесь удобно считать, что $\mathtt{D}[0] = 0$. Таким образом, вместо сложности $O(Cl)$ вычисления первого слагаемого в (**) имеем только $O(C)$.

То есть в итоге сложность уменьшена до $O\big(nmC(C+1)\big) = O(nmC^2)$.

В принципе, вычисление суммы $f_k(y_j)$ так же можно оптимизировать при помощи массива префиксных сумм. Перед началом цикла по i при очередном значении j можно предпросчитать массив $\mathtt{F}$, в котором $\mathtt{F}[k]$ есть сумма всех $f_q(y_j)$ для $1 \leqslant q \leqslant k$. Тогда сложность вычисления выражения под минимумом станет константной, а общая сложность алгоритма — $O(nmC)$. Впрочем для построения наиболее оптимального алгоритма это улучшение не требуется; оно будет включено в алгоритм автоматически.

Дальнейшая оптимизация связана одновременно с упрощением вычисления внешнего минимума и суммы слагаемых $f_k(y_j)$. Рассмотрения будем проводить для фиксированного j, поэтому в следующих рассуждениях для компактности записи будем опускать аргумент у f_k и аргумент, по которому берётся минимум. Рассмотрим выражение, стоящее внутри фигурных скобок.

Пусть сначала $i \leqslant c_j$. Если $p = 1$, то это выражение имеет вид

$$\sum_{k=i-1+1}^{i} f_k + \min \mathtt{dp}[i-1, r] = f_i + \mathtt{MinDP}[i-1]. \qquad (***)$$

При $2 \leqslant p \leqslant i$ можно проделать следующие преобразования:

$$\sum_{k=i-p+1}^{i} f_k + \min \mathtt{dp}[i-p, r] = f_i + \sum_{k=i-p+1}^{i-1} f_k + \min \mathtt{dp}[i-p, r] =$$

$$= f_i + \sum_{k=(i-1)-(p-1)+1}^{i-1} f_k + \min \mathtt{dp}[(i-1)-(p-1), r] = \Big[i' = i-1,\ p' = p-1\Big] =$$

$$= f_i + \sum_{k=i'-p'+1}^{i'} f_k + \min \mathtt{dp}[i'-p', r].$$

Отметим, что $1 \leqslant p' \leqslant i' = i - 1$.

Сравнивая начало преобразования и его конец, можно заметить, что форма выражений совпадает. Кроме того, диапазон изменения p' соответствует диапазону, по которому берётся внешний минимум в (**), для $i' = i - 1$. Стало быть, можно сделать вывод, что при вычислении минимума при каком-то i мы имеем дело с теми же самыми числами, что и при работе с $i - 1$, только увеличенными на f_i, к которым ещё добавлена величина (***), где также фигурирует слагаемое f_i.

Эти рассуждения приводят к необходимости иметь структуру данных, которая позволяет:

1) добавлять числа в себя;
2) быстро получать минимум из имеющегося в данный момент набора;
3) быстро увеличивать все числа в наборе на указанную величину.

Структура данных, реализующая пункты 1) и 2) известна, она называется *очередь с минимумами* (см., например, `http://e-maxx.ru/algo/stacks_for_minima`, раздел «Модификация очереди. Способ 2»).

Для реализации пункта 3) требуется следующая модификация. Дополнительно храним в структуре поле `off`, которое содержит аккумулированное значение прибавленных величин; в начале `off` $= 0$. При вызове метода увеличения всех элементов на величину `val` набора делаем `off` $\leftarrow$ `off` $+$ `val` без работы с элементами очереди. При добавлении элемента v в очередь реально заносится величина $v -$ `off`. При запросе минимума возвращается сумма минимума, полученного из модификации очереди, и текущего значения `off`. При извлечении элемента из очереди возвращается величина `head` $+$ `off`, где `head` — элемент из головы очереди. (Впрочем, в нашем случае нам будет достаточно операции удаления головного элемента из очереди без получения его значения.)

Тогда работа цикла по i, если $i \leqslant c_j$, выглядит следующим образом. Пусть `queue` — очередь, модифицированная для быстрого поиска минимума по всем своим элементам и добавления ко всем своим элементам заданной величины. В начале цикла по i она опустошается. При $i = 1$ имеем $\mathtt{dp}[1, j] = f_1(y_j)$. Эта же величина добавляется в `queue`. При переходе в цикле к следующему значению i в очередь добавляется величина $\mathtt{MinDP}[i-1]$, после чего вызывается метод, увеличивающий все элементы в очереди $f_i(y_j)$. После чего из очереди получаем значение минимума по всем её элементам, которое записывается в $\mathtt{dp}[i, j]$.

Теперь перейдём к анализу случая, когда $i \geqslant c_j$. В этом случае при переходе к $i+1$ не происходит увеличение количества величин, по которым в (**) берётся внешний минимум, но происходит изменение их состава. Так же в набор вводится величина (***), но при этом удаляется величина, которая соответствовала $p = c_j$ на предыдущем шаге, то есть величина, которая была введена в набор раньше остальных. Соответственно, нужно следующим образом изменить алгоритм, изложенный в предыдущем абзаце: при переходе в цикле к следующему значению i, если $i > c_j$, из очереди `queue` извлекается головной элемент, после чего в очередь `queue` добавляется величина $\mathtt{MinDP}[i-1]$, затем вызывается метод, увеличивающий все элементы в очереди на $f_i(y_j)$.

Таким образом, в итоге имеем алгоритм сложности $O(nm)$.

Весьма существенными являются затраты памяти на хранение массива `dp`: он содержит до $5000 \times 5000 = 25\,000\,000$ ячеек, каждая по 8 байт, то есть всего $200\,000\,000$ байт. Остальные структуры для своего хранения требуют существенно меньше памяти, так что ограничение по памяти в 256 Мб выполняется.

Идеи тестов:

1. $n = 1$, $m = 1$, $x_1 = y_1$.
2. $n = 1$, $m = 1$, $x_1 \neq y_1$.
3. $n = 20$, $m = 1$, все $x_i = y_1$.
4. $n = 20$, $m = 1$, некоторые $x_i = y_1$.
5. $n = 20$, $m = 1$, все $x_i \neq y_1$.
6. $n = 1$, $m = 20$, x_1 совпадает с несколькими y_j.
7. $n = 1$, $m = 20$, x_1 совпадает с одним y_j.
8. $n = 1$, $m = 20$, x_1 не совпадает ни с одним y_j.

9. $n = 20$, $m = 20$, все y_j совпадают, $c_j \geqslant 20$, все x_i совпадают.
10. $n = 20$, $m = 20$, все y_j различны, $c_j \geqslant 20$, все x_i совпадают.
11. $n = 20$, $m = 20$, все y_j совпадают, $c_j \geqslant 20$, все x_i различные.
12. $n = 20$, $m = 20$, все y_j различны, $c_j \geqslant 20$, все x_i различные.
13. $n = 20$, $m = 20$, все y_j совпадают, $c_j < 20$, все x_i совпадают.
14. $n = 20$, $m = 20$, все y_j различны, $c_j < 20$, все x_i совпадают.
15. $n = 20$, $m = 20$, все y_j совпадают, $c_j < 20$, все x_i различные.
16. $n = 20$, $m = 20$, все y_j различны, $c_j < 20$, все x_i различные.

17–22. Случайные тесты, которые обрабатываются наивной реализацией динамического программирования: $n, m, c_j < 50$.

23–30. Случайные тесты, которые обрабатываются реализацией с первичной оптимизацией минимизируемого выражения: $n, m, c_j < 170$.

31–38. Случайные тесты, которые обрабатываются реализацией с глубокой оптимизацией минимизируемого выражения: $n, m, c_j < 1000$.

39–45. Случайные тесты, которые обрабатываются оптимальным алгоритмом: $n, m, c_j < 4000$.

46–50. Максимальные случайные тесты $n, l, m = 5000$, $c_j \approx 50$.