

Задача А. Мы ждём перемен

Имя входного файла: *стандартный ввод*
 Имя выходного файла: *стандартный вывод*
 Ограничение по времени: *1 секунда*
 Ограничение по памяти: *256 мегабайт*

Виктор любит переменные, но не любит уроки. Также он является школьным диспетчером расписания, и его постоянно просят расставить K уроков длительностью N минут каждый на промежуток в T часов. Определите максимальное количество M перемен и их максимальную длительность L , так, чтобы их продолжительность не отличалась более чем на 1 минуту, либо выведите, что подобное распределение невозможно.

Формат входных данных

На вход программе поступают три числа: K ($1 \leq K \leq 50$) — количество уроков, которое нужно расставить, N ($1 \leq N \leq 60$) — длительность урока в минутах и T ($1 \leq T \leq 24$) — промежуток, на котором нужно расставить уроки.

Формат выходных данных

Программа должна вывести два значения (каждое на отдельной строке): M — максимальное количество перемен и их максимальную длительность L , так, чтобы продолжительность перемен не отличалась более чем на 1 минуту. Если это невозможно (если длительность хотя бы одной переменной получается отрицательной), то нужно вывести число -1 в качестве значения каждого из выходных параметров.

Система оценки

В задаче каждый тест оценивается отдельно.

Примеры

стандартный ввод	стандартный вывод
6 36 7	5 41
9 5 17	8 122

Решение:

Решение довольно очевидное — если у нас запланировано K уроков, то перемен будет $K - 1$. N уроков займут по времени $K * T$ минут, а всего нам доступно $60 * T$ минут. Итого, на переменные у нас останется $60 * T - K * T$ минут, а перемен всего $K - 1$. Далее у нас два варианта развития событий: все переменные будут одинаковыми по времени (если оставшиеся минуты нацело делятся на их количество) или какая-то переменная будет на минуту дольше. А нам как раз и нужно найти длительность этой максимальной переменной. Значит, получается нехитрая формула

$$\frac{60 * T - K * T}{K - 1}$$

Чтобы пройти все тесты нужно внимательно прочитать формат выходных данных — если не получается провести ни одной переменной хотя бы на минуту, то выводим две -1



Задача В. Время магического числа

Имя входного файла:	стандартный ввод
Имя выходного файла:	стандартный вывод
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

Давным-давно в королевстве Тайм жил добрый волшебник Алексей. Он обожал магическое число XYZ (где X, Y, Z , соответственно, первая, вторая и третья цифры трёхзначного десятичного числа), которое приносило ему удачу и радость каждый раз, когда он видел его на волшебных электронных часах королевства, расположенных на главной башне королевского замка. Эти чудесные часы показывали время, состоящее из четырёх цифр: две цифры для часов и две для минут. Если число часов или минут было меньше 10, то первой цифрой шёл 0. Алексей может заколдовать время, чтобы в каждый сутках стало N часов, пронумерованных от 0 до $N - 1$, а в каждом часе M минут, пронумерованных от 0 до $M - 1$. Для этого заклинания Алексей может выбрать любые N и M от 10 до 100, включительно. Часы показывают счастливое время, если цифры на часах похожи на заветное сочетание XYZ :

1. Когда часы показывают $TX : YZ$, где первая цифра T может принимать значение от 0 до 9.
2. Или $XT : YZ$, где цифра T может принимать значение 0, X или Y (так как 0 не мешает восприятию магического числа).
3. Или $XY : TZ$, где цифра T может принимать значение 0, Y или Z .
4. Или даже $XY : ZT$, где цифра T может принимать значение от 0 до 9.

Однажды перед Алексеем встал вопрос: если известны N, M, XYZ сколько минут за одни полные сутки часы показывают счастливое время? Помогите Алексею найти это число.

Формат входных данных

В единственной строке через пробел записаны целые числа N, M, XYZ — число часов в сутках, число минут в часе ($10 \leq N, M \leq 99$) и магическое число XYZ ($100 \leq XYZ \leq 999$).

Формат выходных данных

Выведите одно число: сколько минут за полные сутки часы показывают счастливое время.

Система оценки

В задаче каждый тест оценивается отдельно.

Примеры

стандартный ввод	стандартный вывод
61 66 407	7
44 24 251	11

Решение:

В этой задаче нужно было аккуратно реализовать поминутный перебор целого дня. На это по-требуется $N * M$ операций. Учитывайте тот факт, что в записи каждого значения часа и минуты могут быть незначимые нули (если значение однозначное). А далее нужно разделить каждое значение часов и минут на десятки и единицы и прописать проверки из условия задачи. Вычислительная сложность $O(N * M)$

Задача С. Ленивый огородник

Имя входного файла: *стандартный ввод*
Имя выходного файла: *стандартный вывод*
Ограничение по времени: *1 секунда*
Ограничение по памяти: *256 мегабайт*

Коля приехал на лето в деревню к бабушке. После душных городских улиц что может быть лучше, чем полежать под раскидистым абрикосом в прохладной тени на гамаке...Как бы не так! На следующий день бабушка отправила Колю сажать картошку. Ее огород представлен в виде непре-рывной прямой линии длиной N метров, расположенной вдоль небольшой речки. Линия размечена на ячейки 1×1 метр и в каждой такой ячейке можно посадить не более 1 картошки. По ГОСТам, принятым еще в незапамятные времена, никакие две и более картошки не должны расти в соседних ячейках огорода (иначе они будут недополучать воду и удобрения). Но на некоторых участках огорода почва довольно каменистая и в них, ровно как и в ячейках справа и слева от этих, картошка расти не сможет. Коля хочет как можно быстрее закончить с посадкой корнеплодов и пойти купаться на речку! Помогите ему засеять огород так, чтобы все ГОСТы и правила были соблюдены, было посажено минимальное количество картофеля, но заполнить свободные места без нарушения регламента было невозможно.

Формат входных данных

В первой строке входных данных записано целое число N ($1 \leq N \leq 10^5$) — длина огорода. Во второй строке записана последовательность из символов S, F , где S означает сектор с каменистой почвой, а F — сектор пригодный для посадки.

Формат выходных данных

Выведите ответ на задачу — минимальное количество картофеля, которое придется посадить.

Система оценки

В задаче каждый тест оценивается отдельно.

Пример

стандартный ввод	стандартный вывод
8 SFFFFSFS	1

Решение:

Первое, что нужно было заметить — нас не интересуют позиции, на которых каменистая почва (стоит буква S в строке), а интересно только то, что между ними, до них или после них. Было бы неплохо избавиться от букв S и разделить строку на подряд идущие символы F . Добиться такого эффекта можно, к примеру, с помощью метода `split` в Python. Далее нас интересуют 3 ситуации:

1. В строке вообще нет символов S .
2. Строка имеет вид $SF...F$ или $F...FS$.
3. Строка имеет вид $SF...FS$.

Для каждой такой ситуации нужно заметить закономерность какое минимальное количество са-женцев нужно посадить, чтобы ни один из них не рос рядом с другим или рядом с камнем, а также ни в одно свободное место после этого нельзя будет ничего добавить. Если строка попадает под первую ситуацию, то после применения метода `split` к ней у нас образуется список, состоящий из одного элемента. Во всех остальных ситуациях будет получаться список хотя бы из двух элементов. И в таком списке первый и последний элементы попадают под вторую ситуацию, а все остальные под третью. Будьте аккуратны с обработкой пустых подстрок в списке после разделения. Все выше-перечисленные ситуации разбиты на подзадачи в тестовом наборе. Если вы правильно обработали только первый случай — вы получите 20 баллов, если обработали второй случай — получите еще 40 баллов, если третий случай — еще 20 баллов и, наконец, если вы совместите второй и третий случай, то получаете оставшиеся 20 баллов и задача решена!

Задача D. Мемная сортировка

Имя входного файла:	<i>стандартный ввод</i>
Имя выходного файла:	<i>стандартный вывод</i>
Ограничение по времени:	<i>1 секунда</i>
Ограничение по памяти:	<i>256 мегабайт</i>

Однажды Игорь Борисович дал своим ученикам такую задачку: пусть у нас есть массив $[-7, -5, -20, -14, -10]$. Давайте применим для каждого элемента следующее правило: если элемент четный, то сокращаем его в 2 раза, а в противном случае увеличиваем в 3 раза и прибавляем 1. Ребята ответственно подошли к заданию и сделали все, что просил Игорь Борисович, и, о чудо - массив отсортировался! Получилось $[-20, -14, -10, -7, -5]$. Круто? — Да это же круто! Получается, что все—таки есть сортировка, всегда работающая за линейное время! Но, применив ее к следующему массиву $[100, 21, 30]$, ребята очень разочаровались — ведь получилось $[50, 64, 15]$, это далеко не то, что они ожидали. Но на доске же все работало!

Важное уточнение — занятие проходило 1 - апреля, поэтому Игорь Борисович просто над ребятами пошутил. Или нет? Для проведения следующих занятий и демонстрации фокуса с новыми

массивами было бы неплохо научиться генерировать такой массив произвольной длины, который бы корректно сортировался таким способом (т.е. чтобы каждый следующий элемент в нем после сортировки был не меньше, чем предыдущий и отсортированный массив получался перестановкой элементов исходного). Давайте ему в этом поможем! Единственное ограничение — в массиве нельзя использовать 0, иначе это будет совсем неинтересная задача.

Формат входных данных

На вход программе подается натуральное число N ($2 \leq N \leq 10^5$) — длина массива, которую хочет получить Игорь Борисович.

Формат выходных данных

В качестве ответа на первой строке выведите размер нового массива (должен совпадать с N), а затем в следующей строке выведите N целых чисел (кроме 0), для которых сортировка будет правильно работать. Если решения нет — выведите *нет*. Если подходящих массивов несколько — выведите любой из них.

Система оценки

В задаче каждый тест оценивается отдельно.

Решение:

Грубо говоря, наша задача найти некий цикл из чисел, которые при применении этого правила не образуют никаких других чисел кроме самих себя (причем итоговые числа должны быть еще и перестановкой исходных). Очевидно, что далеко не любые числа могут образовать такой цикл, а уж тем более думать о том, как составить циклы желаемой длины вообще не хочется. Но, если мы найдем такие циклы маленькой длины, то из них можно будет составить и циклы большей длины, продублировав их нужное количество раз. Давайте придумаем цикл длины 2. Он есть, это числа $-1, -2$. Действительно — $-3 + 1 = -2$, а $-2/2 = -1$. А есть ли цикл длины 3? - Да, есть! Это числа $2, 4, 1$ — $2/2 = 1$, $4/2 = 2$, а $1 * 3 + 1 = 4$. Бинго! У нас есть цикл длины 2 и цикл длины 3 и с помощью них можно составить абсолютно любой цикл!

Задача Е. Городская эстафета

Имя входного файла: *стандартный ввод*
Имя выходного файла: *стандартный вывод*
Ограничение по времени: *1 секунда*
Ограничение по памяти: *256 мегабайт*

Форест Гамп заявился на очередную эстафету. Суть ее очень проста — нужно пробежаться по городским достопримечательностям и в каждой локации забрать флажок с определенным номером. В начале забега спортсмену выдают задание, в котором указано в каком порядке нужно собирать флажки. Менять порядок флажков нельзя — за это сразу дисквалифицируют! Но кроме задания на эстафету у спортсменов есть возможность ознакомиться с картой, на которой обозначены все городские локации и какие флажки на них находятся. Важно, что каждой локации соответствует флажок с единственным номером. На каждой площадке спортсмена поджидает волонтер, который отвечает за раздачу флажков. Но, если волонтер видит, что спортсмен пробежал мимо него и не забрал флажок, то он расстраивается, забирает все флажки и уходит пить раф на кокосовом молоке. Так получилось, что все городские локации в этом замечательном городе расположены вдоль одной длинной улицы. Эстафета начинается с левого края улицы. Форест получил свое задание, наизусть запомнил карту, но он сразу заметил, что собрать флажки в нужном порядке можно не единственным образом. А давайте поможем ему выяснить наверняка сколькими способами он может пройти эстафету!

Формат входных данных

В первой строке входных данных подается два целых числа N, K ($1 \leq K \leq N \leq 10^5$) — количество городских достопримечательностей и количество флажков, которые нужно собрать Форесту.

Во второй строке через пробел записаны N целых чисел f ($0 \leq f \leq 10^9$) — карта города с расстановкой флажков.

В третьей строке через пробел записаны K целых чисел g ($0 \leq g \leq 10^9$) — задание Фореста на эстафету. Гарантируется, что все флажки в задании различные и каждый из них можно найти в городе.

Формат выходных данных

Выведите количество способов, которыми можно пройти эстафету. Т.к. оно может быть достаточно большим, то выведите его по модулю $10^9 + 7$

Система оценки

В задаче каждый тест оценивается отдельно.

За решения, правильно работающие при $N \leq 1000, K \leq 2$, можно получить не более 10 баллов. За решения, правильно работающие при любых N и $K = 2$, можно получить не более 20 баллов.

За решения, правильно работающие при $N \leq 100$ и любых K , можно получить не более 30 баллов.

Пример

стандартный ввод	стандартный вывод
5 2 0 0 0 1 1 0 1	6

Решение:

В случае, когда $k = 1$ ответом будет просто количество единственного флажка из задания в начальной последовательности.

Если, $k = 2$ тут уже интереснее. Представим, что у нас последовательность флажков 0,1 и мы забираем из текущего пункта флажок с номером 1. Значит сколько у нас будет способов до него добраться? — Правильно, из любого предыдущего флажка с номером 0. А сколько таких флажков было до этого? — Можно каждый раз вызывать метод count для всего префикса массива (такое решение будет работать за время $O(N^2)$ и наберет 10 баллов), а можно делать подсчет нулей на префиксе (такое решение будет работать за время $O(N)$ и наберет еще 10 баллов, потому что покрывает случаи с маленькими N).

А вот если $k = 3$, что тогда? Представим последовательность 0,1,2. Если мы забираем флажок 2, то сколько будет способов в него попасть? — Столько, сколько до этого было подпоследовательностей вида 0,1. Теперь нужно разобраться как их эффективно считать. Давайте введем понятие $pred[i]$, которое будет обозначать количество подпоследовательностей на префиксе, оканчивающихся на i . Значит, если мы встретили 2, то $pred[2] += pred[1]$. А если мы встретили 1, то $pred[1] += pred[0]$, а если 0, то просто увеличиваем $pred[0]$ на 1. И далее нетрудно догадаться, что ответ у нас будет лежать в ячейке $pred[2]$. Таким образом, если обобщить этот подход, то для флажков, которые не являются первыми в задании, нужно прибавить количество способов попасть в них на префиксе из предыдущего флажка, а если мы встретили первый флажок, то просто увеличиваем его на 1. Еще обратите внимание, что i это не номер флажка, а его значение. При аккуратной реализации это будет работать за $O(N)$.



Задача F. Просчеты архитектора

Имя входного файла: *стандартный ввод*
Имя выходного файла: *стандартный вывод*
Ограничение по времени: *1 секунда*
Ограничение по памяти: *256 мегабайт*

Матвей выиграл тендер на постройку небоскреба в Санкт—Петербурге. Северная столица, Бал-тийское море, Нева... Здание получилось красивым, современным, технологичным, но была одна проблема — в разработке дизайна Матвею помогал MegaChat. LLM все продумала, кроме климати-ческих особенностей региона — всем кроме нее известно, что в городе над вольной Невой постоянно идут дожди. Матвей, разумеется, тоже об этом не подумал, и, когда здание было достроено, после первых осадков вскрылся главный его недостаток. Если посмотреть на здание сбоку, то оно похоже на пирамиду. Когда идет дождь вся вода должна спускаться по фасаду здания вниз, но на некоторых участках она задерживается, образуя своего рода озера. Потолок в этих местах начинает протекать, жильцы недовольны. Тут уж искусственный интеллект не поможет и Матвею срочно нужно провести аудит — сколько в его здании есть мест, где может скапливаться вода, а также найти объем самого большого такого места. Помогите горе—архитектору исправить свои ошибки!

Формат входных данных

В первой строке входных данных записано целое число N ($1 \leq N \leq 10^5$) — количество участков, в которых были сделаны замеры высоты здания.

Во второй строке через пробел записаны N целых чисел h ($1 \leq h \leq 10^9$) — высота каждого участка здания.

Формат выходных данных

В качестве ответа выведите два целых числа: в первой строке количество проблемных участков в здании, а во второй объем самого большого из них.

Система оценки

В задаче каждый тест оценивается отдельно.

За решения, правильно работающие при $N \leq 1000$, можно получить не более 30 баллов.

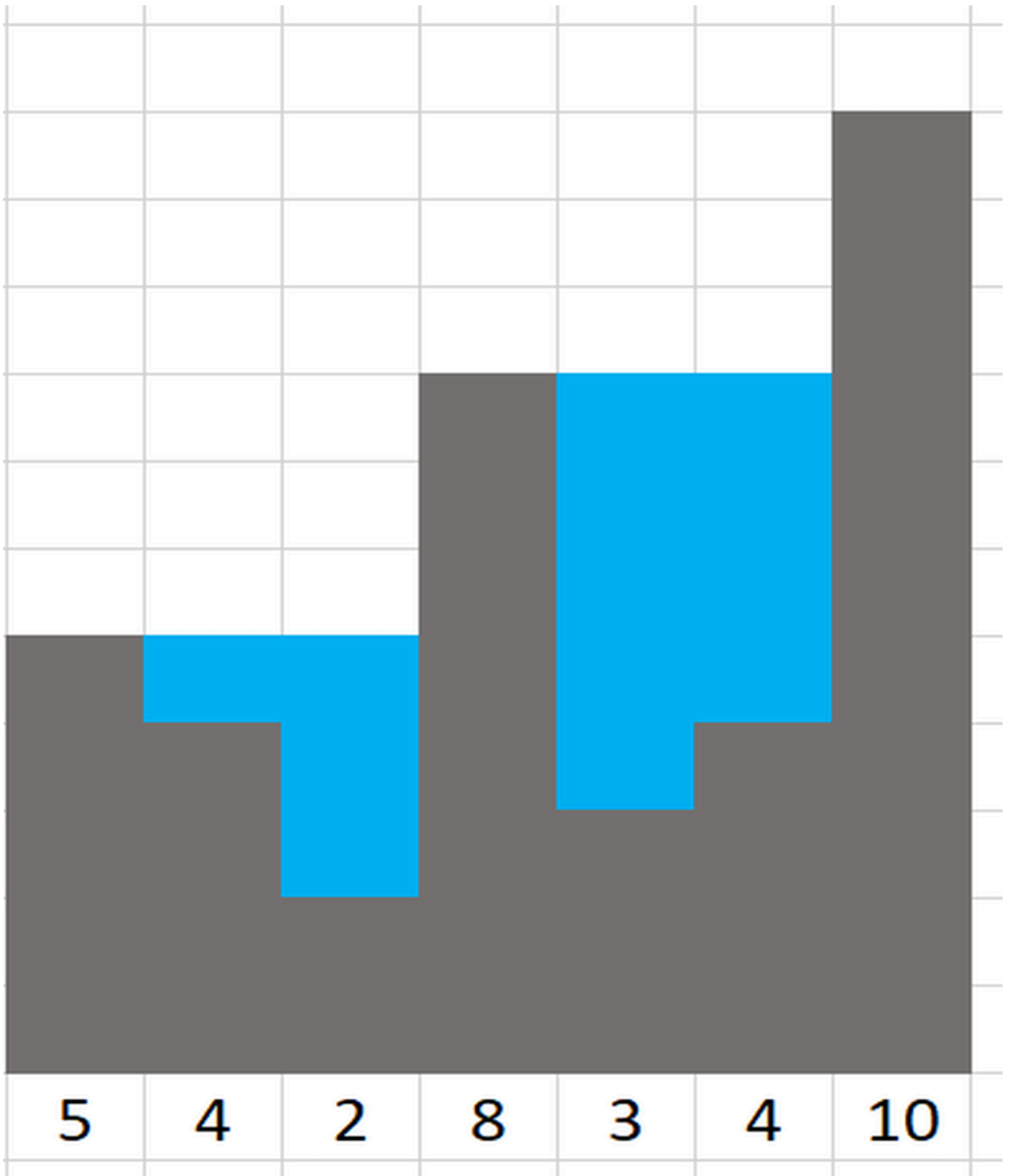
Пример

стандартный ввод	стандартный вывод
7 5 4 2 8 3 4 10	2 9

Замечание

В тестовом примере в здании образуется два озера (выделены на изображении синим цветом).

Объем озера — количество закрасенных клеток, которые в нем находятся.



Решение:

Представим, что мы сейчас находимся на участке здания, высота которого равна H . В каком озере он может находиться? Ну как минимум границы этого озера должны быть выше, чем H . Вероятно, что таких границ (уровень которых выше текущего участка) будет несколько, но в первую очередь нас интересуют именно ближайšie. А те, в свою очередь, могут ограничиваться другими участками. И так далее, и так далее. Т.е. каждый участок озера будет характеризоваться некоторым отрезком, в котором он точно может существовать как часть озера. А как понять, что несколько отрезков можно объединить в озеро? — Правильно, их объединение должно образовывать непре-рывный отрезок. А если участок не может входить ни в одно озеро, то как мы это поймем? — А для него ни справа, ни слева не окажется участков повыше. Итого, нам необходимо:

1. Найти для каждого элемента списка ближайший больший справа и слева от него.
2. Объединить получившиеся отрезки.
3. Для каждого объединения найти уровень, на котором будет стоять вода, и рассчитать объем.

Очевидным решением для пункта 1 будет запуск из каждого элемента вложенного цикла с целью найти ближайший больший от него элемент. Но такое решение будет работать за $O(N^2)$ и наберет не более 30 баллов. На самом деле это классическая задача на алгоритмы со стеками и ее можно решить за $O(N)$ двумя проходами по списку.

А чтобы объединить отрезки все еще проще. Как мы понимаем, что объединение закончилось? — Когда мы увидим отрезок, для которого не нашлась левая или правая границы (или даже обе). А для всех отрезков, которые можно было объединить, мы должны рассчитать самую дальнюю левую и правую границы. Это и будет озером. Очевидно, что уровень воды, который будет в нем стоять, это минимум из высот левой и правой его границ. А чтобы рассчитать объем, нужно пройтись по всему озеру и найти разницу высот каждого участка и уровня воды. Итоговая сложность решения $O(N)$.

Задача G. Встреча на карте

Имя входного файла: *стандартный ввод*
 Имя выходного файла: *стандартный вывод*
 Ограничение по времени: *1 секунда*
 Ограничение по памяти: *256 мегабайт*

Петя и Ваня играют в настольную игру на линейной карте из N клеток, пронумерованных от 1 до N . Петя начинает на клетке 1, Ваня — на клетке N .

Для перемещения по карте используется необычный игральный кубик с K гранями. Каждая грань задаёт операцию из множества $\{+, -, *, /\}$ и натуральное число x ($1 \leq x \leq N$). Если игрок находится на клетке с номером p , то при выпадении грани (op, x) он **пытается** перейти на клетку с номером p *op* x (деление целочисленное).

Игроки ходят по очереди, первым ходит Петя. Считайте, что на каждом ходе может выпасть любая из K граней кубика, и вас интересует **минимально возможное** число ходов до завершения игры и при этом в каждую клетку каждый игрок стремится попасть как можно быстрее.

Игроки движутся навстречу друг другу.

Ход считается **недопустимым**, если игрок окажется в клетке, которая не принадлежит диапа-зону $[1, N]$.

Игра завершается в тот момент, когда после очередного хода оба игрока оказываются на одной и той же клетке.

Формат входных данных

В первой строке заданы два целых числа N и K ($1 \leq N \leq 10^5, 1 \leq K \leq 100$).

В следующих K строках задано описание граней кубика: операция *op* и число x . Операция *op* — один символ из $+, -, *, /$. Число x удовлетворяет $1 \leq x \leq N$.

Формат выходных данных

Выведите одно целое число — количество клеток, на которых может закончиться игра.

Система оценки

В задаче каждый тест оценивается отдельно.

Пример

стандартный ввод	стандартный вывод
10 4 + 5 / 2 - 2 - 1	2

Решение:

По условию задачи мы хотим попасть в каждую клетку поля как можно быстрее. У нас есть определенные команды (игральный кубик). На первый взгляд, задача похожа на классическую за-дачу о Кузнечике, а она решается методом динамического программирования. Но тогда не совсем понятно в каком порядке делать обход клеток, т.к. в задаче о Кузнечике он довольно очевидный (с начала в конец или наоборот). Ведь иногда, чтобы попасть в клетку с координатой X , нужно сделать шаг назад из клетки с большей координатой (и возможно по другому в нее в принципе никак не получится попасть). Допустим, что у нас есть команды $+1, *2, /2$ и мы стоим в клетке с координатой X . Значит, из нее мы можем попасть в клетки с координатами $X+1, X*2, X/2$ (конечно учитывая, что мы не покинем пределы поля). Это будет кратчайший путь. А чтобы найти кратчай-ший путь до следующих клеток, то придется делать такие же ходы по очереди из $X + 1, X * 2, X/2$. Очевидно, что нам неинтересно заходить в одну и ту же клетку дважды (иначе второй заход не будет кратчайшим путем). Итого, нам нужно: обрабатывать клетки в том порядке, в

котором мы их обходим (т.е. в порядке увеличения расстояния от стартовой) и запоминать в каких клетках мы побывали. А это буквально описание алгоритма BFS — поиск в ширину. Обычно он применяется на графах, но на самом деле эту задачу тоже можно представить как граф, где вершины — числа, а ребра — операции, которые над ними можно совершить. Чтобы хранить расстояние до каждого числа можно ввести понятие $dist[i]$, обозначающее расстояние до i -го числа. Изначально все $dist[i]$ можно пометить -1 и таким образом мы будем понимать были мы в вершине или нет (очевидно, что расстояние до вершины не может быть отрицательным). Чтобы обрабатывать вершины в порядке увеличения расстояния от стартовой, нам потребуется очередь (queue). Если мы стоим в вершине i и хотим сделать из нее ход $*2$, то $dist[i * 2] = dist[i] + 1$. И $dist$ для стартового значения, очевидно, что надо обозначить 0. Итого, для каждого игрока придется сделать свой запуск BFS, посчитать для каждой клетки кратчайшее расстояние от стартовой и те клетки, в которых это расстояние совпадет для первого игрока и для второго и будут нам подходить. Из-за того, что мы используем $dist$ мы гарантируем, что в каждую клетку мы зайдём ровно 1 раз в рамках одного запуска BFS. И из каждой клетки мы попытаемся сделать K различных операций. Итоговая сложность $O(N * K)$.

Задача Н. Блокировка серверов

Имя входного файла: стандартный ввод
 Имя выходного файла: стандартный вывод
 Ограничение по времени: 2 секунды
 Ограничение по памяти: 256 мегабайт

Роскомнадзор (РКН) обнаружил новую децентрализованную сеть VPN-сервисов. Сеть устроена иерархически и представляет собой корневое дерево, где вершина №1 — это главный координационный сервер. У каждого сервера i есть определенный показатель паразитного трафика W_i . Механизм блокировки работает следующим образом: если РКН блокирует сервер v , то автоматически отключаются и сам сервер v , и всё его поддереву (все сервера, для которых v является прямым или косвенным предком). При этом нагрузка на оборудование РКН, необходимая для блокировки сервера v , равна суммарному паразитному трафику всего поддерева вершины v . Отдел мониторинга перехватил лог активности — последовательность из M идентификаторов серверов P_1, P_2, \dots, P_M , которые проявляли активность в хронологическом порядке. Инженеры РКН хотят выбрать непрерывный отрезок времени (подстроку лога $P [L..R]$), чтобы заблокировать все серверы из этого отрезка по очереди. Однако оборудование имеет предел мощности: суммарная нагрузка от блокировок на выбранном отрезке не должна превышать K . Важное уточнение: Система блокировки $\ll \gg$. Если в выбранном отрезке лога есть сервер A , а затем сервер B , который находится в поддереве A , то система потратит ресурсы на блокировку A (отключив всё поддерево), а затем снова потратит ресурсы на попытку блокировки B (хотя он уже оффлайн). То есть нагрузки просто суммируются. Ваша задача: Найти максимальную длину непрерывного отрезка в логе, который можно обработать, не превысив лимит нагрузки K .

Формат входных данных

Первая строка содержит три целых числа N, M, K ($1 \leq N, M \leq 2 \cdot 10^5, 1 \leq K \leq 10^{18}$) — количество серверов, длина лога и предел мощности.

Вторая строка содержит N чисел W_i ($1 \leq W_i \leq 10^9$) — трафик i сервера.

Следующие $N - 1$ строк описывают связи в сети. Каждая строка содержит два числа u, v , означающие, что u является родителем v . (Гарантируется, что граф — дерево с корнем в вершине 1).

Последняя строка содержит M чисел P_i ($1 \leq P_i \leq N$) — идентификаторы серверов в логе активности.

Формат выходных данных

Выведите одно число — максимальную длину непрерывного отрезка в логе, который можно обработать, не превысив лимит нагрузки K .

Система оценки

В задаче каждый тест оценивается отдельно.

Пример

стандартный ввод	стандартный вывод
5 6 20 2 3 14 5 1 2 1 3 2 4 2 5 4 5 2 3 4 1	3

Решение:

Если мы посмотрим на основной вопрос задачи, то он звучит буквально так: найдите подотрезок максимальной длины, сумма которого не превышает K . Классическая задача на два указателя. Но откуда взять значения для этого отрезка? Каждое значение отрезка по условию задачи это номер сервера и, если мы его блокируем, то он блокируется сам и блокирует все серверы, которые от него зависят. Конечно, мы можем в процессе обработки подотрезка для каждого элемента обсчитывать нагрузку, но тогда сложность будет порядка $O(N^2)$. В условии задачи сказано, что все блокировки не зависят друг от друга, поэтому разумно будет их заранее предподсчитать и потом обращаться к уже готовым значениям. На этом этапе нам нужно будет запомнить для каждого сервера (вершины) — количество серверов, которые от него зависят (количество вершин его поддеревя). Представим, что мы находимся в вершине V и от нее зависят вершины A и B , от которых зависят еще какие-то вершины. Но то, что зависит от A, B нам уже неинтересно, ведь оно посчитано и ответ для V это сумма ответов для A, B . Это классическая задача динамического программирования на поддеревьях и решается она с помощью алгоритма DFS — поиск в глубину. Он работает за $O(N)$ и нахождение нужного подотрезка тоже за $O(N)$.

Задача I. Генерация ландшафта

Имя входного файла:	стандартный ввод
Имя выходного файла:	стандартный вывод
Ограничение по времени:	1 секунда
Ограничение по памяти:	1024 мегабайта

Вы участвуете в разработке новой 2D-игры в жанре «песочница». Дизайнеры уровней нарисовали схематичную карту мира, но она получилась слишком угловатой и короткой. Ваша задача — написать алгоритм процедурной генерации, который расширит карту и сделает ландшафт более естественным.

Карта мира состоит из n независимых горизонтальных слоев (например, слой облаков, слой поверхности, слой подземелья). Исходно каждый слой состоит из m блоков, идущих подряд слева направо. Высота каждого блока задается целым числом от 0 до 9.

Вам необходимо увеличить ширину мира в k раз. Это происходит по следующим правилам:

- Каждый исходный блок превращается в участок ландшафта из k новых блоков.
- Порядок участков сохраняется: если в исходном слое блок A шел перед блоком B , то и участок, сгенерированный из A , будет идти перед участком из B .
- Высота каждого нового блока должна быть целым числом от 0 до 9.
- Чтобы сохранить общий рельеф, задуманный дизайнером, средняя высота участка должна соответствовать исходной. Формально: сумма высот k новых блоков должна быть равна высоте исходного блока, умноженной на k .

Игрокам не нравится перемещаться по слишком резким склонам. «Неудобство» перехода между двумя соседними по горизонтали блоками с высотами h_1 и h_2 обычно вычисляется по формуле $(h_1 - h_2)^2$.

Однако в игре есть специальные механики (например, лестницы или трамплины). Для t особых

упорядоченных пар высот (a, b) существует фиксированное значение неудобства c , которое используется вместо стандартной формулы квадрата разности.

Ваша задача — сгенерировать расширенный ландшафт так, чтобы суммарное неудобство переходов между всеми соседними блоками во всех слоях было минимальным. Слои независимы друг от друга, переходы между блоками разных слоев учитывать не нужно.

Формат входных данных

В первой строке заданы три целых числа n , m и k — количество слоев, количество исходных блоков в слое и коэффициент растяжения ($1 \leq n, m \leq 50, 2 \leq k \leq 50$).

Далее следует n строк по m символов в каждой — описание исходной карты. Каждый символ является цифрой от 0 до 9 и обозначает высоту соответствующего блока.

В следующей строке записано целое число t — количество исключений в правилах подсчета неудобства ($0 \leq t \leq 10$).

Затем следуют t строк, содержащих по три целых числа a_i, b_i, c_i — описание исключения ($0 \leq a_i, b_i \leq 9, 0 \leq c_i \leq 100$). Это означает, что при переходе с высоты a_i на высоту b_i штраф равен c_i . Гарантируется, что для каждой упорядоченной пары высот (a, b) существует не более одного исключения.

Формат выходных данных

Выведите одно число — минимальное возможное суммарное неудобство, которое можно получить после расширения карты.

Система оценки

В задаче каждый тест оценивается отдельно.

Примеры

стандартный ввод	стандартный вывод
3 3 2 010 111 23 4 0	4
3 3 2 010 111 23 4 1 2 3 40	6
6 10 2 881111188 1881111881 1188888811 1111881111 1111881111 1111881111 1 1 9 1	74

Замечание

Один из оптимальных способов растянуть карту в последнем примере выглядит следующим образом:

```
887911111111119788
1197791111111977911
11119788888888791111
111111197791111111
111111197791111111
111111197791111111
```

Решение

Задача состоит в том, чтобы расширить заданную 2D-карту, увеличив её ширину в k раз, и при этом минимизировать суммарный «штраф за неудобство» переходов между соседними блоками.

Ключевые моменты:

- **Независимость слоев:** Каждый из n слоев генерируется независимо от других. Это означает, что мы можем решить задачу для одного слоя и просто просуммировать полученные минимальные штрафы для всех n слоев.
- **Расширение блоков:** Каждый исходный блок с высотой h превращается в сегмент из k новых блоков.
- **Сохранение средней высоты:** Сумма высот k новых блоков в сегменте должна быть строго равна $h \times k$.
- **Минимизация штрафа:** Цель — минимизировать сумму штрафов за переходы между всеми соседними блоками. Штрафы бывают двух видов:

1. Внутри одного сегмента (между $k - 1$ парой соседних блоков).
2. Между сегментами (между последним блоком одного сегмента и первым блоком следующего).

Такая структура задачи, где нужно принять последовательность решений (выбрать высоты блоков) для минимизации итоговой функции, является классическим признаком задачи на динамическое программирование (ДП).

2. Идея решения

Решение можно разбить на два основных этапа, каждый из которых использует свой метод ДП.

2.1 ДП 1: Предподсчёт стоимостей внутри сегмента

Сначала нам нужно научиться отвечать на вопрос: какова минимальная стоимость переходов внутри одного сегмента длины k , если мы знаем его суммарную высоту, высоту первого и последнего блока?

Введём состояние ДП: $DP1[l][s][hfirst][hlast]$ — минимально возможный штраф за переходы внутри сегмента из l блоков, сумма высот которых равна s , первый блок имеет высоту $hfirst$, а последний — $hlast$.

Параметры состояния:

- l : длина сегмента, $1 \leq l \leq k$.
- s : сумма высот в сегменте, $0 \leq s \leq 9 \times l$.
- $hfirst$: высота первого блока, $0 \leq hfirst \leq 9$.
- $hlast$: высота последнего блока, $0 \leq hlast \leq 9$.

База ДП: Для сегмента длиной $l = 1$, нет внутренних переходов. Поэтому штраф равен 0. $DP1[1][h][h][h] = 0$ для всех $h \in [0,9]$. Остальные значения инициализируем бесконечно-стью.

Переходы: Чтобы вычислить значение для $DP1[l][s][hfirst][hlast]$, мы можем взять уже посчитанный оптимальный сегмент длины $l - 1$ и добавить к нему справа новый блок высотой $hlast$. Этот предыдущий сегмент должен был иметь сумму высот $s - hlast$, начинаться с $hfirst$ и заканчиваться на какой-то высоте $hprev_last$. Формула перехода:

$$DP1[l][s][hfirst][hlast] = \min_{h \in [0,9]} (DP1[l-1][s-hlast][hfirst][hprev_last] + Cost(hprev_last, hlast))$$

где $Cost(h1, h2)$ — это штраф за переход с высоты $h1$ на $h2$, который мы заранее вычисляем в матрице.

Мы итеративно вычисляем это ДП для l от 2 до k .

2.2 ДП 2: Расчёт оптимального ландшафта для строки

Теперь, когда мы умеем быстро находить стоимость любого сегмента, мы можем найти оптимальное решение для всей строки, состоящей из m исходных блоков.

Введём второе состояние ДП: $DP2[i][hlast]$ — минимальный суммарный штраф для первых i сегментов (сгенерированных из первых i исходных блоков), при условии, что последний блок i -го сегмента имеет высоту $hlast$.

Параметры состояния:

- i : количество рассмотренных исходных блоков, $1 \leq i \leq m$.
- h_{last} : высота последнего блока в последнем, i -м сегменте, $0 \leq h_{last} \leq 9$.

База ДП ($i = 1$): Для первого сегмента нет переходов извне. Его стоимость равна внутренней стоимости, которую мы уже рассчитали. Пусть H_1 — высота первого блока в исходной строке, тогда требуемая сумма $S_1 = H_1 \times k$.

$$DP_2[1][h] = \min_{hf \in [0,9]} (DP_1[k][S_1][h][hf])$$

Переходы ($i > 1$): Чтобы рассчитать $DP_2[i][h_{curr_l}]$, мы должны рассмотреть i -й сегмент. Он должен иметь сумму высот $S_i = H_i \times k$, где H_i — высота i -го исходного блока. Этот сегмент может начинаться с любой высоты h_{curr_f} и заканчиваться на h_{curr_l} . Стоимость этого сегмента складывается из:

1. Минимальной стоимости для первых $i-1$ сегментов, заканчивающихся на некоторую высоту h_{prev_l} (это $DP_2[i-1][h_{prev_l}]$).
2. Штрафа за переход между $(i-1)$ -м и i -м сегментами: $Cost(h_{prev_l}, h_{curr_f})$.
3. Внутренней стоимости i -го сегмента: $DP_1[k][S_i][h_{curr_f}][h_{curr_l}]$.

Формула перехода:

$$DP_2[i][h_{cl}] = \min_{h_{pl}, h_{cf} \in [0,9]} (DP_2[i-1][h_{pl}] + Cost(h_{pl}, h_{cf}) + DP_1[k][S_i][h_{cf}][h_{cl}])$$

Итоговый ответ для строки: После вычисления ДП для всех m блоков, минимальный штраф для всей строки будет минимальным значением в последнем столбце ДП:

$$MinPenalty_{row} = \min_{h \in [0,9]} DP[m][h]$$

Общий ответ на задачу — сумма таких минимальных штрафов по всем n строкам.

3 Асимптотика

Обозначим количество уровней высоты (в данном случае 10) как N .

Временная сложность:

1. ДП 1 ($segment_penalty$): Вычисляется один раз. Количество состояний: $k \times (9k) \times N \times N$. Переход для каждого состояния требует цикла по N предыдущим высотам. Сложность: $O(k \times (9k) \times N \times N \times N) = O(k^2 \cdot N^5)$.
2. ДП 2 (для одной строки): Этот этап выполняется n раз, по одному для каждой строки. Для каждой строки, состоящей из m сегментов, мы последовательно вычисляем состояния ДП. Наивный переход для i -го сегмента выглядел бы так:

$$DP_2[i][h_{cl}] = \min_{h_{pl}, h_{cf}} (DP_2[i-1][h_{pl}] + Cost(h_{pl}, h_{cf}) + DP_1[k][S_i][h_{cf}][h_{cl}])$$

Такой тройной вложенный перебор по высотам (hpl, hcf, hcl) дал бы сложность $O(m \cdot N^3)$ на одну строку.

Однако можно заметить, что часть $\min_{hcl} (DP2[i-1][hpl] + Cost(hpl, hcf))$ зависит только от hcf . Эту часть можно вычислить заранее для всех hcf за $O(N^2)$. После этого вычисление $DP2[i][hcl]$ для всех hcl также потребует $O(N^2)$. Таким образом, сложность перехода для одного сегмента снижается до $O(N^2)$, а общая сложность на одну строку — до $O(m \cdot N^2)$.

Итоговая временная сложность: $O(k^2 \cdot N^5 + n \cdot m \cdot N^2)$. Учитывая, что $N = 10$ — небольшая константа, асимптотика для ограничений задачи ($n, m, k \leq 50$) выглядит как $O(k^2 + n \cdot m)$, что является очень эффективным.

Пространственная сложность:

1. ДП 1 (`segment_penalty`): Хранит всю таблицу. Сложность: $O(k \times (9k) \times N \times N) = O(k^2 \cdot N^2)$.
2. ДП 2 (`current_row_dp`): Для вычисления состояний для i -го сегмента нам нужны только состояния для $(i - 1)$ -го. Поэтому мы можем хранить только два слоя ДП (текущий и предыдущий). Сложность: $O(N)$.

Итоговая пространственная сложность определяется таблицей для ДП 1: $O(k^2 \cdot N^2)$. При $N = 10$ это $O(k^2)$.

Задача J. Идеальный баланс

Имя входного файла: *стандартный ввод*
 Имя выходного файла: *стандартный вывод*
 Ограничение по времени: *1 секунда*
 Ограничение по памяти: *256 мегабайт*

Последовательность Морса—Туэ — бесконечная битовая последовательность, состоящая из 0 и 1. Начинается она всегда с 0, а далее каждый раз к получившейся на прошлом шаге строке дописывается ее инвертированная версия. Нетрудно догадаться, что после N итераций получится строка длины 2^N . Ваша задача заключается в следующем — по заданному номеру бита в получившейся строке определите его значение.

Формат входных данных

На вход программе поступает число N ($1 \leq N \leq 60$) — количество итераций при составлении последовательности.

На второй строке записано натуральное число Q ($1 \leq Q \leq 10^5$) — количество запросов.

Далее в Q строках записаны натуральные числа x ($0 \leq x \leq 2^{N-1}$) — номер бита, для которого нужно определить значение.

Формат выходных данных

Для каждого запроса выведите значение бита 0 или 1, который стоит под этим номером в последовательности Морса—Туэ

Система оценки

В задаче каждый тест оценивается отдельно.

За решения, правильно работающие при $N \leq 17$ можно получить не более 25 баллов.

Пример

стандартный ввод	стандартный вывод
3	1
4	0
1	0
5	1
3	
2	

Решение:

Решение на 25 баллов довольно очевидное — нужно составить последовательность Морса—Туэ и результатом каждого запроса будет символ, находящийся под этим номером в получившейся последовательности. В остальных тестах значение N достаточно большие. Заметим, что значение каждого i -го бита совпадает со значением бита на позиции $2i$ и не совпадает со значением бита на позиции $2i + 1$. Биты на позициях 0, 1 равны 0 и 1 соответственно.

Допустим, что мы стоим в бите на позиции i . Если она четная, то позиция $\frac{i}{2}$ совпадает с

ним по значению. А если позиция нечетная, то бит на позиции $\frac{i}{2}$ будет противоположным. Таким образом, мы обязательно дойдем до позиций под номерами 0, 1, а их значения заранее известны. И рекурсивно восстановим ответ для позиции i . Сложность решения $O(Q * N)$. Также, можно заметить, что значение i -го разряда в последовательности Морса—Туэ совпадает с остатком от деления на 2 количества единиц в двоичной записи числа i . Этот факт был замечен абсолютно случайно до рекурсивного решения. В таком случае нам нужно лишь уметь переводить

в двоичную запись номер запрашиваемого разряда и считать в ней количество единиц. Сложность решения аналогичная $O(Q * N)$.