A. Старт коллайдера

ограничение по времени на тест

2 секунды

ограничение по памяти на тест

256 мегабайт

ввод

стандартный ввод

вывод

стандартный вывод

Совсем скоро произойдёт запуск самого нового, мощного и необычного коллайдера, который расположен вдоль прямой. В нём будут запущены *n* частиц. Все они расположены на одной прямой, причем в одной точке не может быть двух или более частиц. Координаты частиц совпадают с расстоянием в метрах от центра коллайдера, то есть *xi* — это и координата *i*-й частицы, и одновременно её позиция в коллайдере. Все координаты частиц — **целые чётные числа**.

Про каждую частицу известно направление её движения — она будет двигаться либо влево, либо вправо после старта коллайдера. Все частицы начнут двигаться одновременно в момент старта коллайдера. Каждая частица будет двигаться ровно влево или ровно вправо с постоянной скоростью 1 метр в микросекунду. Считайте, что коллайдер достаточно большой, и частицы не могут вылететь из него за обозримое время.

Напишите программу, которая найдет момент первого столкновения двух каких-либо частиц коллайдера. Иными словами, требуется найти количество микросекунд до первого такого момента, когда какие-либо две частицы окажутся в одной точке.

**Входные данные**

В первой строке следует целое положительное число *n* (1 ≤ *n* ≤ 200 000) — количество частиц.

Во второй строке следует *n* символов «L» и «R», причём если *i*-й символ равен «L», то *i*-я частица будет двигаться влево, иначе *i*-й символ равен «R», и *i*-я частица будет двигаться вправо.

В третьей строке следует последовательность различных целых **чётных** чисел *x*1, *x*2, ..., *xn* (0 ≤ *xi* ≤ 109) — координаты частиц в порядке слева направо. Гарантируется, что координаты частиц заданы в порядке возрастания.

**Выходные данные**

В первой строке выведите единственное целое число — первый момент времени (в микросекундах), в который две частицы окажутся в одной точке и произойдет взрыв.

Выведите единственное целое число -1, если столкновения частиц не произойдет.

**Примеры**

**входные данные**

4  
RLRL  
2 4 6 10

**выходные данные**

1

**входные данные**

3  
LLR  
40 50 60

**выходные данные**

-1

**Примечание**

В первом примере из условия первый взрыв произойдёт через 1 микросекунду, так как частицы с номерами 1 и 2 одновременно окажутся в точке с координатой 3.

Во втором примере из условия взрыв никогда не произойдёт, так как никакие две частицы никогда не окажутся в одной точке одновременно.

B. Одна бомба

ограничение по времени на тест

1 секунда

ограничение по памяти на тест

256 мегабайт

ввод

стандартный ввод

вывод

стандартный вывод

Вам задано описание склада, который представляет из себя прямоугольное клетчатое поле размера *n* × *m*. Каждая клетка склада может быть либо пустой («.»), либо занятой стеной («\*»).

У вас есть одна бомба. Если её заложить в клетке (*x*, *y*), то при срабатывании она уничтожит все препятствия, находящиеся в строке *x*, и все препятствия, находящиеся в столбце *y*.

Перед вами стоит задача определить, можно ли с помощью взрыва **ровно** одной бомбы уничтожить все стены на складе. Бомбу разрешается закладывать как в пустую клетку, так и в клетку со стеной.

**Входные данные**

В первой строке следует два целых положительных числа *n* и *m* (1 ≤ *n*, *m* ≤ 1000) — количество строк и столбцов поля.

Следующие *n* строк содержат по *m* символов «.» и «\*» — описание склада. *j*-й символ в *i*-й из них соответствует клетке (*i*, *j*).Если символ равен «.», то соответствующая клетка пустая. В противном случае символ равен «\*», и в соответствующей клетке находится стена.

**Выходные данные**

Если невозможно уничтожить все стены с помощью одной бомбы, выведите в первую строку «NO» (без кавычек).

В противном случае выведите в первую строку «YES» (без кавычек), а во вторую два целых числа — координаты клетки, в которую нужно заложить бомбу. Если вариантов ответа несколько, разрешается вывести любой из них.

**Примеры**

**входные данные**

3 4  
.\*..  
....  
.\*..

**выходные данные**

YES  
1 2

**входные данные**

3 3  
..\*  
.\*.  
\*..

**выходные данные**

NO

**входные данные**

6 5  
..\*..  
..\*..  
\*\*\*\*\*  
..\*..  
..\*..  
..\*..

**выходные данные**

YES  
3 3

C. Каникулы

ограничение по времени на тест

1 секунда

ограничение по памяти на тест

256 мегабайт

ввод

стандартный ввод

вывод

стандартный вывод

Каникулы Васи будут длиться *n* дней! И Вася решил за это время улучшить свои навыки программирования, а также позаниматься спортом. Про каждый из *n* дней каникул Вася знает, будет ли открыт в этот день спортзал, а также будет ли в интернете в этот день проводиться контест. Для *i*-го дня возможны четыре варианта:

1. в этот день закрыт спортзал и не проводится контест;
2. в этот день закрыт спортзал и проводится контест;
3. в этот день открыт спортзал и не проводится контест;
4. в этот день открыт спортзал и проводится контест.

В каждый из дней Вася может либо отдыхать, либо писать контест (если он проводится в этот день), либо заниматься спортом (если открыт спортзал).

Перед вами стоит задача найти минимальное количество дней, в которые Вася будет отдыхать (то есть не будет заниматься спортом и не будет писать контест одновременно). Единственное ограничение от Васи — *он не хочет два дня подряд заниматься одним и тем же видом активности, то есть он не будет заниматься спортом два дня подряд и он не будет писать контесты два дня подряд*.

**Входные данные**

В первой строке следует целое положительное число *n* (1 ≤ *n* ≤ 100) — количество дней в каникулах Васи.

Во второй строке следует через пробел последовательность целых чисел *a*1, *a*2, ..., *an* (0 ≤ *ai* ≤ 3), где:

* *ai* равно 0, если в *i*-й день каникул не работает спортзал и не проводится контест;
* *ai* равно 1, если в *i*-й день каникул не работает спортзал, но проводится контест;
* *ai* равно 2, если в *i*-й день каникул работает спортзал и не проводится контест;
* *ai* равно 3, если в *i*-й день каникул работает спортзал и проводится контест.

**Выходные данные**

Выведите минимально возможное количество дней, в которые Вася будет отдыхать. Помните, что Вася отказывается:

* заниматься спортом в какие-либо два подряд идущих дня,
* писать контест в какие-либо два подряд идущих дня.

**Примеры**

**входные данные**

4  
1 3 2 0

**выходные данные**

2

**входные данные**

7  
1 3 3 2 1 2 3

**выходные данные**

0

**входные данные**

2  
2 2

**выходные данные**

1

**Примечание**

В первом тестовом примере Вася может написать контест в день номер 1 и позаниматься спортом в день номер 3. Таким образом, он будет отдыхать всего два дня.

Во втором тестовом примере Вася должен писать контесты в дни с номерами 1, 3, 5 и 7, а в остальные дни заниматься спортом. Таким образом, он не будет отдыхать ни одного дня.

В третьем тестовом примере Вася может позаниматься спортом либо в день номер 1, либо в день номер 2. Он не может заниматься спортом оба дня, так как это будет противоречить его ограничению. Таким образом, он будет отдыхать один день.

D. Исправление дерева

ограничение по времени на тест

2 секунды

ограничение по памяти на тест

256 мегабайт

ввод

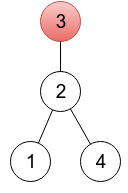
стандартный ввод

вывод

стандартный вывод

Дерево — это неориентированный связный граф без циклов.

Рассмотрим корневое неориентированное дерево, состоящее из *n* вершин, пронумерованных от 1 до *n*. Существует несколько способов представлять такие деревья. Одним из наиболее популярных является массив предков, состоящий из *n* чисел *p*1, *p*2, ..., *pn*, где *pi* означает предка вершины *i* (в рамках данной задачи корень дерева считается предком самого себя).

Для данного корневого дерева массив *p* равен [2, 3, 3, 2].

По данной последовательности *p*1, *p*2, ..., *pn* легко восстановить дерево:

1. Существует ровно один индекс *r*, такой что *pr* = *r*. Вершина *r* является корнем дерева.
2. Для всех остальных *n* - 1 вершины *i* существует ребро между вершинами *i* и *pi*.

Последовательность *p*1, *p*2, ..., *pn* является корректной, если описанная выше процедура определяет некоторое (произвольное) корневое дерево. Например, для *n* = 3 последовательности (1,2,2), (2,3,1) и (2,1,3) **не являются** корректными.

Вам дана последовательность *a*1, *a*2, ..., *an*, не обязательно являющаяся корректной. Вам требуется изменить минимальное количество элементов, чтобы последовательность стала корректной. Выведите минимальное необходимое количество изменений и пример любого ответа, который можно получить с их помощью. Если существует несколько корректных последовательностей, достижимых за минимальное количество изменений, то разрешается вывести любую.

**Входные данные**

В первой строке входных данных записано число *n* (2 ≤ *n* ≤ 200 000) — количество вершин в дереве (а значит, и в последовательности).

Во второй строке записаны *n* целых чисел *a*1, *a*2, ..., *an* (1 ≤ *ai* ≤ *n*).

**Выходные данные**

В первой строке выведите минимальное количество изменений, которое надо сделать, чтобы получить корректную последовательность.

Во второй строке выведите любую корректную последовательность, которая может быть получена из (*a*1, *a*2, ..., *an*) за минимальное количество изменений. Если существует несколько таких последовательностей, то выведите любую.

**Примеры**

**входные данные**

4  
2 3 3 4

**выходные данные**

1  
2 3 4 4

**входные данные**

5  
3 2 2 5 3

**выходные данные**

0  
3 2 2 5 3

**входные данные**

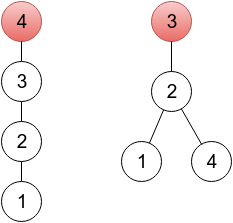
8  
2 3 5 4 1 6 6 7

**выходные данные**

2  
2 3 7 8 1 6 6 7

**Примечание**

В первом примере достаточно изменить один элемент. В приведённом примере вывода последовательность представляет корневое дерево с корнем в вершине 4 (потому что *p*4 = 4), которое можно увидеть на левом рисунке. Одним из корректных решений будет являться последовательность 2 3 3 2, определяющая дерево с корнем в вершине 3 (правый рисунок). На обоих рисунках корни выделены красным.



Во втором примере данная во входных данных последовательность уже является корректной.

E. LRU

ограничение по времени на тест

2 секунды

ограничение по памяти на тест

256 мегабайт

ввод

стандартный ввод

вывод

стандартный вывод

При создании высоконагруженных систем важную роль играет кэширование данных. В данной задаче пойдёт речь о схеме кэширования LRU (Least Recently Used).

Допустим, размер кэша позволяет хранить не более *k* объектов. В начале работы программы кэш пуст. При поступлении запроса на обращение к некоторому объекту проверяется его наличие в кэше, и в случае отсутствия объект помещается в кэш. Если после этого в кэше становится больше чем *k* объектов, то из него удаляется тот объект, который последний раз запрашивался раньше всех, то есть дольше всех не использовался.

Пусть на некотором сервере хранится *n* видеороликов, причём все видеоролики одинакового размера. Кэш сервера вмещает *k*видеороликов, и используется схема кэширования, описанная в предыдущем абзаце. Известно, что когда пользователь заходит на сервер и выбирает какой-нибудь ролик для просмотра, вероятность выбрать видеоролик *i* составляет *pi*. При этом выбор очередного видеоролика никак не зависит от того, что происходило ранее.

Требуется для каждого видеоролика посчитать вероятность того, что он будет находиться в кэше через 10100 запросов пользователей.

**Входные данные**

В первой строке входных данных записаны два числа *n* и *k* (1 ≤ *k* ≤ *n* ≤ 20) — количество видеороликов и размер кэша соответственно. В следующей строке находятся *n* вещественных чисел *pi* (0 ≤ *pi* ≤ 1), заданных с не более чем двумя знаками после запятой.

Гарантируется, что сумма всех *pi* равняется 1.

**Выходные данные**

Выведите *n* вещественных чисел, *i*-е из которых должно равняться вероятности того, что *i*-й видеоролик будет находиться в кэше через 10100 операций. Ваш ответ будет считаться правильным, если его абсолютная или относительная ошибка не будет превосходить 10- 6.

А именно: пусть ваш ответ равен *a*, а ответ жюри — *b*. Проверяющая программа будет считать ваш ответ правильным, если http://codeforces.com/predownloaded/c6/2e/c62ea64d4651240724c5ac4779b671c741edec24.png.

**Примеры**

**входные данные**

3 1  
0.3 0.2 0.5

**выходные данные**

0.3 0.2 0.5

**входные данные**

2 1  
0.0 1.0

**выходные данные**

0.0 1.0

**входные данные**

3 2  
0.3 0.2 0.5

**выходные данные**

0.675 0.4857142857142857 0.8392857142857143

**входные данные**

3 3  
0.2 0.3 0.5

**выходные данные**

1.0 1.0 1.0

F. Медвежонок и позиции для стрельбы

ограничение по времени на тест

3 секунды

ограничение по памяти на тест

256 мегабайт

ввод

стандартный ввод

вывод

стандартный вывод

Беарляндия — очень опасное место, поэтому Лимак никогда не ходит пешком. Вместо этого он использует камни телепортации. Всего у него *k* таких камней, *i*-й из которых позволяет мгновенно переместиться в точку (*axi*, *ayi*). Каждый камень может быть использован **не более одного раза**, при этом использовать камни можно **в любом порядке**.

В Берляндии живут *n* монстров. Монстр с номером *i* расположен в точке с координатами (*mxi*, *myi*). Гарантируется, что позиции всех *k* + *n* точек различны.

После каждой телепортации Лимак может выпустить стрелу в произвольном направлении. Стрела убьёт первого монстра, который будет на её пути, после чего и стрела, и монстр исчезнут с карты. Поскольку оставаться на одном месте опасно, Лимак может выпустить не более одной стрелы с каждой позиции.

Монстр опасается Лимака, если существует последовательность действий, при которой Лимак убивает данного конкретного монстра. Вычислите, сколько монстров должны опасаться Лимака?

**Входные данные**

В первой строке входных данных записаны два числа *k* и *n* (1 ≤ *k* ≤ 7, 1 ≤ *n* ≤ 1000) — количество камней телепортации и монстров на карте соответственно.

В *i*-й из последующих *k* строк записаны два целых числа *axi* и *ayi* ( - 109 ≤ *axi*, *ayi* ≤ 109) — координаты точки, в которую переместится Лимак при использовании *i*-го камня.

В *i*-й из последующих *n* строк записаны два целых числа *mxi* и *myi* ( - 109 ≤ *mxi*, *myi* ≤ 109) — координаты *i*-го монстра.

Гарантируется, что все *k* + *n* точек попарно различны.

**Выходные данные**

Выведите количество монстров, которым стоит опасаться Лимака.

**Примеры**

**входные данные**

2 4  
-2 -1  
4 5  
4 2  
2 1  
4 -1  
1 -1

**выходные данные**

3

**входные данные**

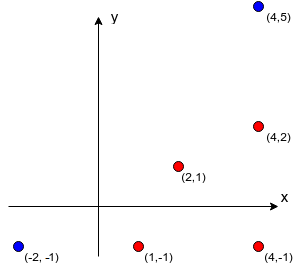
3 8  
10 20  
0 0  
20 40  
300 600  
30 60  
170 340  
50 100  
28 56  
90 180  
-4 -8  
-1 -2

**выходные данные**

5

**Примечание**

В первом примере имеются два камня телепортации и четыре монстра. С помощью камней можно переместиться в точки ( - 2,  - 1) и (4, 5), отмеченные синим на рисунке ниже. Монстры находятся в точках (4, 2), (2, 1), (4,  - 1) и (1,  - 1), отмеченных красным. Опасаться Лимака стоит всем монстрам, кроме расположенного в точке (4,  - 1), поэтому ответ равен 3.



Во втором примере опасаться нужно пятерым монстрам. В безопасности себя могут чувствовать только монстры в точках (300, 600), (170, 340) и (90, 180).