Задача Bugatti. Разрезание торта

Имя входного файла:	`input.txt` или стандартный поток ввода
Имя выходного файла:	`output.txt` или стандартный поток вывода
Ограничение по времени:	2 секунды
Ограничение по памяти:	512 мегабайт

Данную задачу можно решать только на языке программирования C++.

Организаторы Закрытой олимпиады школьников по программированию решили заказать большой торт на вечерний банкет. Для этого они заказали торт с n свечками, пронумерованными от 0 до $n-1$. Известно, что свечки находятся в различных точках на плоскости, никакие три свечки не лежат на одной прямой, никакие четыре свечки не образуют трапецию (четырехугольник с двумя параллельными сторонами).

Чтобы всем достался кусочек торта, организаторы хотят заранее построить способ разрезать торт на максимальное число кусочков. К сожалению, вам неизвестно расположение свечек на торте, а единственный доступный способ узнать, как выглядит торт — переписка с кондитерской по электронной почте.

В одном сообщении вы можете попросить взвесить не более четырех треугольных кусочков, с вершинами в свечках:

- Вы предоставляете два списка треугольников `left` и `right`, суммарно содержащих не более четырех треугольников. Треугольники могут пересекаться, содержать общие свечки и даже совпадать.
- В кондитерской вырежут треугольные кусочки в соответствии с запрошенными списками из нескольких копий торта и положат кусочки из `left` на левую чашу весов, кусочки из `right` на правую чашу весов.
- Вам отправят результат взвешивания. Будем считать, что вес кусочка равен его площади. В результате вы узнаете, в каком из списков треугольников суммарная площадь больше, или что площади совпадают.

После нескольких взвешиваний вы должны найти способ разрезать торт. Каждый кусочек должен быть выпуклым многоугольником с вершинами в свечках. Вершины кусочка должны быть перечислены в порядке обхода по или против часовой стрелки. Любые два кусочка не должны пересекаться по внутренним точкам. Вы должны вернуть список кусочков, суммарная площадь которых максимально возможная. Также в некоторых группах требуется максимизировать число кусочков.

Формат решения

Это необычная задача. Она имеет формат тестирования с грейдером, где вам необходимо реализовать только функцию `solve` с решением. Эта функция будет вызвана тестирующей программой жюри (грейдером), и возвращаемое значение функции будет принято как решение задачи.

В частности, это означает, что в отправленном вами коде **не должно быть ввода или вывода**. Ваш код **не должен** содержать функцию `main`. При необходимости вы можете реализовать произвольное количество вспомогательных функций, структур, классов и глобальных переменных, но весь код вашего решения должен находиться в одном файле.

Вы должны реализовать следующую функцию:

```
std::vector<std::vector<int>> solve(int n);
```

Функция `solve` принимает на вход единственное число n — количество свечек.

В реализации функции `solve` вы можете использовать функцию `compare`, которую реализует грейдер:

```
int compare(const std::vector<Triangle>& left, const std::vector<Triangle>& right);
```

Данная функция принимает на вход два непустых списка треугольников суммарного размера не более 4. В качестве результата она возвращает -1, если суммарная площадь треугольников в `left` меньше суммарной площади треугольников в `right`, 0, если площади равны и 1, иначе. Если сделать

некорректный запрос или исчерпать лимит на количество запросов, то программа автоматически завершит свою работу.

Для того, чтобы получить доступ к функции `compare` в решении, первой строчкой вашего кода нужно подключить заголовочный файл следующей строчкой:

```
#include "triangles.h"
```

В этом же файле определена используемая в функции `compare` структура `Triangle`:

```
struct Triangle {
    int i, j, k;

    Triangle() = default;
    Triangle(int i_, int j_, int k_) : i(i_), j(j_), k(k_) {}
};
```

Данная структура описывает один треугольный кусочек, параметры `i`, `j` и `k` описывают номера свечек, образующих вершины треугольного кусочка. Номера свечек должны быть различными для одного треугольного кусочка, но могут повторяться в нескольких треугольниках.

В код отправляемого решения вам не надо включать определение структуры `Triangle`, оно будет автоматически браться из заголовочного файла.

Все параметры (индексы свечек в запросах, а также в возвращаемом вами результате) задаются в **0-индексации**.

Ваша функция `solve` с помощью вызовов функции `compare` должна найти любое разбиение торта на выпуклые кусочки, в котором любые два кусочка не пересекаются по внутренним точкам, суммарная площадь кусочков максимальна, и, возможно, количество кусочков максимально.

Функция `solve` должна вернуть список кусочков, на которые разбивается торт. Каждый кусочек описывается структурой `std::vector<int>`, состоящей из свечек, образующих выпуклый многоугольник. Свечки должны быть перечислены **в порядке следования по границе многоугольника** (по часовой стрелке или против часовой стрелки). Любые два кусочка не должны пересекаться по внутренним точкам. Вы должны вернуть список кусочков, суммарная площадь которых максимально возможная, а в некоторых группах дополнительно требуется максимизировать количество кусочков, при этом сохраняя условие на максимальную возможную суммарную площадь.

За один запуск грейдера **жюри сделает ровно один вызов функции** `solve`. **Грейдер не является адаптивным**, то есть положения свечек фиксируются заранее и не зависят от реализации функции `solve`.

Тестирование

Вам предоставлен шаблон решения `triangles.cpp`, а так же заголовочный файл `triangles.h`, содержащий определение функций `solve` и `compare`. Для удобства тестирования вам предоставлен грейдер — файл `grader.cpp`. В этом файле реализован ввод входных данных со стандартного потока ввода, запуск функции `solve` и вывод на стандартный поток вывода возвращаемого значения функции `solve`. В тестирующей системе грейдер может отличаться.

Чтобы скомпилировать ваш код `triangles.cpp` на языке C++, используйте команду

```
g++ -std=c++20 grader.cpp triangles.cpp -o grader
```

После выполнения данной команды будет создан исполняемый файл грейдера `grader` или `grader.exe`, в зависимости от вашей операционной системы, запустив который можно ввести тест в формате, указанном далее.

Если компиляция через команды вызывает у вас трудности, для локального тестирования вы можете скопировать реализацию функции `solve` в файл `grader.cpp` (ее необходимо вставить перед функцией `main`) и запустить файл `grader.cpp`. При этом перед отправкой решения в тестирующую систему вам нужно будет оставить только реализацию функции `solve`, **не забыв подключить заголовочный файл в начале кода** (это делается с помощью добавления строки `#include "triangles.h"` в начало сдаваемого вами кода). Если вы используете какие либо библиотеки языка C++, их подключение также необходимо добавить в начало отправляемого кода.

В случае получения вердикта ошибка компиляции, убедитесь, что **в отправляемом вами коде не содержится функции `main`, а также не содержатся определения функции `compare` и структуры `Triangle`**. Функцию `compare` и структуру `Triangle` вы можете только использовать в отправляемом вами коде.

Формат входных данных

Грейдер читает тест в следующем формате:

В первой строке находятся целое число n ($4 \leqslant n \leqslant 10\,000$) — количество свечек.

В следующих n строках находится по два целых числа x_i, y_i ($-10^9 \leqslant x_i, y_i \leqslant 10^9$) — координаты i-й свечки.

Формат выходных данных

Грейдер выводит результаты функции `solve` — список найденных кусочков.

В первой строке выводится число найденных многоугольников (размер вектора, возвращаемого функцией `solve`).

В следующих строках для каждого многоугольника (элемента вектора векторов, возвращаемого функцией `solve`) в начале выводится размер многоугольника, а в следующей строке номера свечек, являющихся вершинами многоугольника (элементы каждого из `std::vector<int>`, входящих в возвращаемое функцией `solve` значение). Каждый многоугольник должен быть выпуклым, его вершины должны быть перечислены в порядке следования (по часовой стрелке или против часовой стрелки). Никакие два многоугольника не должны пересекаться во внутренних точках.

В файле `grader.cpp` есть переменная `verbose`, которая исходно равна 0. При увеличении ее значения грейдер будет более подробно писать информацию о вашем решении и его запросах.

Примеры

ввод	вывод
4 1 2 1 4 0 0 3 -1	3 3 0 1 2 3 0 2 3 3 0 1 3
5 -1 -1 4 4 4 -2 1 2 -2 2	1 4 0 4 1 2
6 2 2 0 -2 -1 3 -2 0 7 0 2 -3	4 3 2 4 3 3 3 4 5 3 1 3 5 3 0 2 4

Пояснение

В первом примере возможная последовательность вызовов функций могла выглядеть так:

В начале вызывается функция `solve(4)`.

Из функции `solve` вызывается функция

```
compare({Triangle(0, 1, 2)}, {Triangle(1, 2, 3)})}
```

Во время выполнения функции происходит сравнения размеров треугольных кусочков, образованных треугольником с вершинами в свечках с номерами 0, 1 и 2 и треугольником с вершинами в свечках 1, 2, 3.

Так как первый треугольник меньше второго, то функция возвращает -1.

Далее из функции `solve` вызывается функция

```
compare({Triangle(1, 2, 3)}, {Triangle(0, 1, 2), Triangle(0, 1, 3)})}
```

В ответ будет возвращено 1, так как площадь треугольника с вершинами в свечках 1, 2 и 3 больше суммарной площади треугольника с вершинами в свечках 0, 1 и 2 и треугольника с вершинами в свечках 0, 1 и 3.

Далее из функции `solve` вызывается функция

```
compare({Triangle(1, 2, 3)}, {Triangle(0, 1, 2), Triangle(2, 3, 0), Triangle(0, 1, 3)})}
```

В ответ будет возвращен 0.

В конце функция `solve` возвращает `{{0, 1, 2}, {0, 2, 3}, {0, 1, 3}}` — описание разделения торта на выпуклые кусочки, имеющие максимальную возможную суммарную площадь и не пересекающихся во внутренних точках.

В первом и третьем примере, в найденном способе разрезать торт количество кусочков максимально возможное, поэтому такие ответы считались бы корректными в группах с максимизацией. Во втором тесте количество кусочков не максимально возможное, поэтому такой ответ считался бы корректным в группах без максимизации, но получил бы 0 баллов в группах с максимизацией.

Обратите внимание, что первый тест подходит под дополнительные ограничения групп 1 и 2, а третий тест подходит под дополнительные ограничения группы 5.

Система оценки

Тесты к этой задаче состоят из 11 групп. Правила, по которым выставляются баллы за группы описаны ниже. Обратите внимание, что прохождение тестов из условия не требуется для некоторых групп. Итоговый балл за каждую группу равняется максимальному баллу, полученному за эту группу тестов по всем отправленным посылкам.

Балл решения за группу тестов равен минимальному баллу, полученному по всем тестам группы.

- Если ваше решение сделает некорректный вызов `compare` или вернет ответ, не удовлетворяющий обязательным условиям, оно получит 0 баллов за тест.
- В каждом тесте вашему решению разрешается выполнить не более $2 \cdot 10^6$ вызовов функции `compare`. В случае превышения лимита ваше решение получит 0 баллов за тест.
- В части групп требуется максимизировать число кусочков в ответе. В таких группах, если число кусочков будет не максимально возможным, ваше решение получит 0 баллов за тест.

Если ни одно из описанных условий для теста не нарушено, то ответ за тест считается корректным. Пусть полный балл за группу тестов равен p. Часть групп оценивается с использованием формулы, часть оценивается без использования формулы:

- Если группа оценивается без использования формулы, балл за тест равен p.
- Иначе, если группа оценивается с использованием формулы, пусть q это количество вызовов `compare`, которое сделало ваше решение. Тогда балл за тест равен $\left\lfloor p \cdot \frac{20n}{\max(q, 20n)} \right\rfloor$.

Группа	Полный балл	Оценка		Доп. ограничения	Необх. группы	Комментарий
		Макс.	Формула	n		
0	0	нет	нет	–	–	Тесты из условия.
1	13	нет	нет	$n = 4$	–	
2	4	да	нет	$n = 4$	1	
3	13	нет	нет	–	–	$(10^9, -10^9)$, $(-10^9, 10^9)$, $(10^9, 10^9)$ есть в множестве, остальные точки случайные внутри этого треугольника
4	8	да	нет	–	3	$(10^9, -10^9)$, $(-10^9, 10^9)$, $(10^9, 10^9)$ есть в множестве, остальные точки случайные внутри этого треугольника
5	11	да	да	–	–	точки являются вершинами выпуклого многоугольника
6	9	нет	нет	$n \leqslant 100$	–	точки случайные
7	8	да	нет	$n \leqslant 100$	6	точки случайные
8	9	нет	да	–	–	точки случайные
9	8	да	да	–	–	точки случайные
10	9	нет	да	–	–	
11	8	да	да	–	–	

В группах 6 – 9 все точки были выбраны случайно и независимо из квадрата $[-10^9, 10^9] \times [-10^9, 10^9]$.

В группах 3 – 4 точки, кроме $(10^9, -10^9)$, $(-10^9, 10^9)$, $(10^9, 10^9)$, были выбраны случайно и независимо из треугольника с вершинами $(10^9, -10^9)$, $(-10^9, 10^9)$, $(10^9, 10^9)$.

В этих группах среди таких случайных тестов оставлены только те, в которых все точки различные, никакие три не лежат на одной прямой, никакие четыре не образуют трапецию.

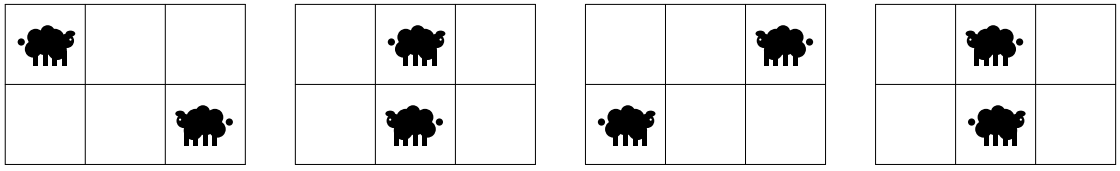
Задача Ferrari. Марш овец

Имя входного файла:	`input.txt` или стандартный поток ввода
Имя выходного файла:	`output.txt` или стандартный поток вывода
Ограничение по времени:	1 секунда
Ограничение по памяти:	512 мегабайт

Саша устал от тяжелой городской жизни и решил уехать в деревню. Чтобы чем-то занять свой быт, он решил заняться разведением овец. Для этого он приобрел в деревне прямоугольный участок, который можно представить в виде клетчатого поля $n \times m$, в котором строки пронумерованы сверху вниз числами от 1 до n, а столбцы пронумерованы слева направо числами от 1 до m.

На свой участок Саша приобрел n овец, каждой из которых он выделил целую строку. Исходно i-я овца была помещена в клетку с координатами (i, a_i) (в строку с номером i и столбец с номером a_i). Саша выяснил, что овцы перемещаются только горизонтально по следующим правилам:

- Если на участке ровно один столбец, то овцы не двигаются.
- Изначально i-я овца находится в клетке (i, a_i), а также у каждой из овец есть изначальное направление, куда она двигается — либо влево, либо вправо. При том не существует овцы, которая находится в первом столбце и идет влево, а также которая находится в последнем столбце и идет вправо.
- Каждую секунду каждая из овец перемещается на один столбец левее, если она движется влево, иначе она перемещается на один столбец правее.
- Если после перемещения овца оказывается в столбце 1 и движется влево, либо же если овца оказывается в последнем столбце и движется вправо, то она меняет направление движения на противоположное. Таким образом, овцы никогда не выходят за пределы участка.



Здесь нарисован пример движения овец, если изначально первая овца находилась в столбце 1 и двигалась вправо, а вторая находилась в столбце 3 и двигалась влево. На первой картинке нарисовано состояние в начале, на второй через одну секунду, на третьей через 2 секунды после начала движения, на четвертой через 3 секунды после начала движения.

Саше не нравится, что овцы так хаотично двигаются, он хочет добиться того, чтобы все овцы двигались одинаково. Это означает, что все овцы должны находиться в одном и том же столбце и иметь одинаковое направление движения. Чтобы этого добиться, Саша может обрезать свой участок несколько (возможно, ноль) раз следующим образом:

- Он выбирает время t (сколько секунд прошло от исходного начала движения овец) и количество столбцов x, которое останется у участка после обрезания.
- Все овцы во время t должны строго лежать внутри участка размера $n \times x$. Это означает, что для любого i от 1 до n, i-я овца во время t должна находиться в клетке (i, y_i), где $1 \leqslant y_i \leqslant x$. Иначе такое обрезание участка недопустимо.
- После этой операции начиная со времени t, количество столбцов у участка уменьшается до x.
- Каждая овца, которая после этого находится в последнем столбце и движется вправо, меняет направление движения на противоположное.

Подробные примеры процесса обрезания участка описаны в описании к тестам из условия.

Саша хочет в первую очередь узнать, можно ли, обрезая участок несколько раз, добиться того, чтобы все овцы двигались одинаково. Если такое возможно, то Саша хочет знать, какое максимальное количество столбцов может остаться на участке и как именно ему этого добиться. Помогите Саше решить эту задачу!

Формат входных данных

В первой строке дано одно целое число T ($1 \leqslant T \leqslant 3$) — параметр, обозначающий какую именно информацию об ответе необходимо вывести. Подробное описание описано в «Формате выходных данных».

Во второй строке записаны два целых числа n и m ($2 \leqslant n \leqslant 200\,000$, $2 \leqslant m \leqslant 10^9$) — количество строк и столбцов у участка.

В третьей строке записано n чисел $a_1, a_2, \ldots, a_n$ ($1 \leqslant a_i \leqslant m$) — номера столбцов, где изначально находятся овцы.

В четвертой строке записана строка s длины n, состоящая только из символов `L`, `R`, где i-й символ равен `L`, если i-я овца изначально двигается влево, и `R`, если i-я овца изначально двигается вправо. Гарантируется, что никакая овца в первом столбце не двигается влево, и никакая овца в m-м столбце не двигается вправо.

Формат выходных данных

В первой строке выведите «`No`» (без кавычек), если Саша не сможет добиться того, чтобы овцы двигались одинаково. Иначе выведите «`Yes`» (без кавычек).

Если $T = 2$ или $T = 3$, то во второй строке выведите максимальное количество столбцов, которое Саша может оставить у участка так, чтобы все овцы двигались одинаково. В случае, если $T = 1$, то **это выводить не нужно**.

Если $T = 3$, в третьей строке выведите число q ($0 \leqslant q \leqslant 10^6$) — сколько раз Саше нужно будет обрезать участок. В следующих q строках выведите описание обрезаний участка.

Для описания каждого обрезания участка выведите в одной строке два целых числа t и x ($0 \leqslant t \leqslant 10^{18}$, $1 \leqslant x < m$), где t — это время, в которое Саше необходимо обрезать участок (от самого начала движения овец), а x — количество столбцов участка, которое останется у участка после этой операции.

Операции необходимо выводить в порядке неубывания t. В каждом следующем обрезании x должен быть меньше, чем в предыдущем.

Если подходящих последовательностей обрезаний несколько, можно вывести любую из них.

Можно показать, что при заданных ограничениях, если ответ существует, то и существует ответ, который укладывается в данные ограничения.

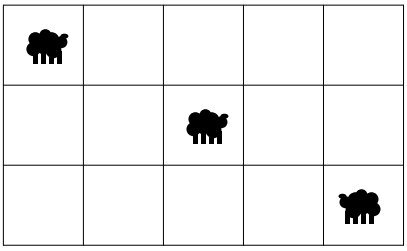
В случае, если $T = 1$ или $T = 2$, то **описание обрезаний участка выводить не нужно**.

Примеры

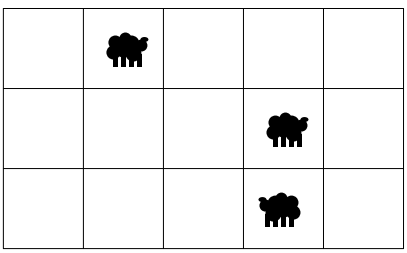
ввод	вывод
3 2 3 1 3 RL	Yes 2 1 3 2
3 2 3 1 2 RL	No
3 3 5 1 3 5 RRL	Yes 3 2 1 4 2 3
2 3 5 1 3 5 RRL	Yes 3
1 3 5 1 3 5 RRL	Yes
3 3 7 3 3 5 RRL	Yes 4 3 0 6 0 5 1 4

Пояснение

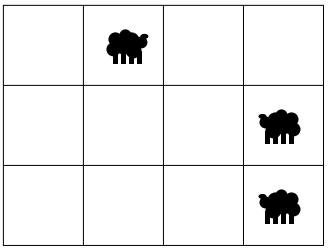
Рассмотрим третий пример:
Состояния овец в нулевую секунду.



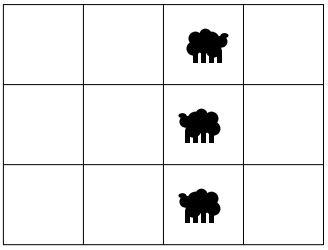
Состояние овец в первую секунду.



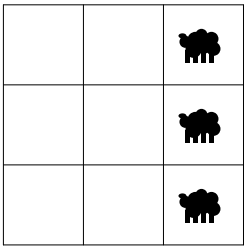
Обрезали участок до 4 клеток.



Состояние овец во вторую секунду.



Обрезали участок до 3 клеток.



Система оценки

Тесты к этой задаче состоят из десяти групп. Баллы за каждую группу ставятся только при прохождении всех тестов группы и всех тестов некоторых из предыдущих групп. Обратите внимание, что прохождение тестов из условия не требуется для некоторых групп. **Offline-проверка** означает, что результаты тестирования вашего решения на данной группе станут доступны только после окончания соревнования. Итоговый балл за каждую группу равняется максимальному баллу, полученному за эту группу тестов по всем отправленным посылкам.

Группа	Баллы	Доп. ограничения			Необх. группы	Комментарий
		T	n	m		
0	0	–	–	–	–	Тесты из условия.
1	8	$T \leqslant 1$	–	–	–	
2	11	$T \leqslant 2$	$n \leqslant 3$	$m \leqslant 4$	–	
3	9	–	$n = 2$	–	–	$a_1 = 1, a_2 = m$
4	12	–	$n = 2$	$m \leqslant 200\,000$	–	
5	8	–	$n = 2$	–	3, 4	
6	14	$T \leqslant 2$	$n \leqslant 1000$	$m \leqslant 1000$	2	
7	10	$T \leqslant 2$	–	–	1, 2, 6	
8	9	–	–		–	Строка s содержит только символы `R`.
9	12	–	$n \leqslant 1000$	$m \leqslant 1000$	0, 2, 6	
10	7	–	–	–	0 – 9	**Offline-проверка.**

Задача Lamborghini. Украшение

Имя входного файла:	`input.txt` или стандартный поток ввода
Имя выходного файла:	`output.txt` или стандартный поток вывода
Ограничение по времени:	1 секунда
Ограничение по памяти:	512 мегабайт

В процессе подготовки организаторы Закрытой олимпиады школьников по программированию решили украсить аудиторию. Для этого на стене на одной прямой было вбито $2n$ колышков. Все колышки находятся в различных позициях. Между ними было натянуто n ниточек, i-я из них была протянута между колышками на расстоянии l_i и r_i от начала стены ($l_i < r_i$). Известно, что на каждый колышек натянута ровно одна ниточка.

С ниточками можно производить операции *перенатягивания*. За одну операцию *перенатягивания* можно выбрать две ниточки (l_i, r_i) и (l_j, r_j), снять их с колышков, а затем повесить две новые ниточки, использовав каждый из четырёх освободившихся колышков ровно один раз. То есть из четырёх освободившихся колышков составляются две новые пары, между которыми натягиваются новые ниточки.

Ниточки, натянутые между колышками (l_i, r_i) и (l_j, r_j), пересекаются, если отрезки $[l_i, r_i]$ и $[l_j, r_j]$ имеют хотя бы одну общую точку. Считается, что конфигурация из ниточек имеет красоту хотя-бы k, если существует множество из k ниточек, каждая пара из которых пересекается. Обратите внимание, что если конфигурация имеет красоту k, то она в том числе имеет красоту хотя-бы $k-1,\ k-2,\ \ldots,\ 0$.

У организаторов олимпиады есть q запросов получения конфигурации некоторой красоты в результате применения нескольких операций *перенатягивания*. В i-м запросе они бы хотели получить красоту хотя-бы k_i. Для каждого из запросов организаторам хочется узнать, какое минимальное количество операций *перенатягивания* необходимо совершить, чтобы этого добиться. Запросы независимы между собой. То есть операции *перенатягивания* между запросами не сохраняются.

Формат входных данных

В первой строке даны два целых числа n и q ($1 \leqslant q \leqslant n \leqslant 200\,000$) — количество ниточек и количество запросов.

В следующих n строках описываются ниточки. В i-й из них даны два целых числа l_i и r_i ($1 \leqslant l_i < r_i \leqslant 10^9$) — номера колышков, соединённых i-й ниточкой. Гарантируется, что каждый колышек встречается ровно один раз.

В следующей строке даны q целых чисел $k_1, k_2, \ldots, k_q$ ($1 \leqslant k_i \leqslant n$) — размеры желаемых красот в запросах от организаторов. Гарантируется, что все k_i различны.

Формат выходных данных

Для каждого запроса выведите одно целое неотрицательное число — минимальное число операций *перенатягивания*, необходимых для получения конфигурации ниточек, имеющих необходимую в запросе красоту.

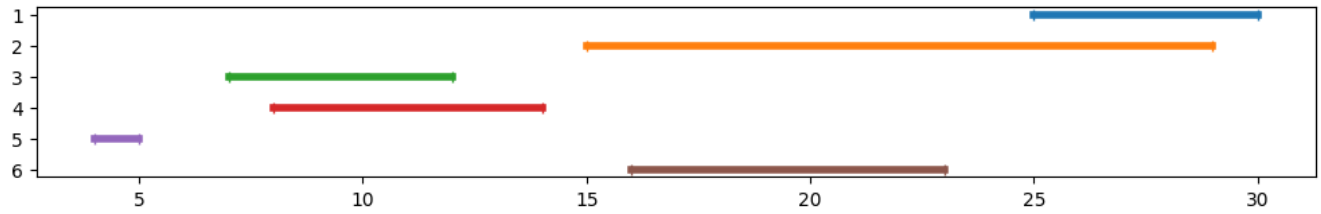
Гарантируется, что для каждого запроса можно совершить несколько перенатягиваний, чтобы получить конфигурацию необходимой красоты.

Пример

ввод	вывод
6 6 25 30 15 29 7 12 8 14 4 5 16 23 1 2 3 4 5 6	0 0 1 1 2 3

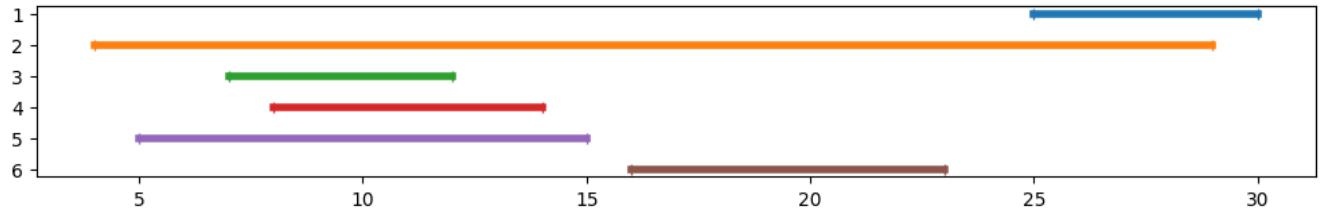
Пояснение

В первом примере ниточки исходно имеют следующую конфигурацию:

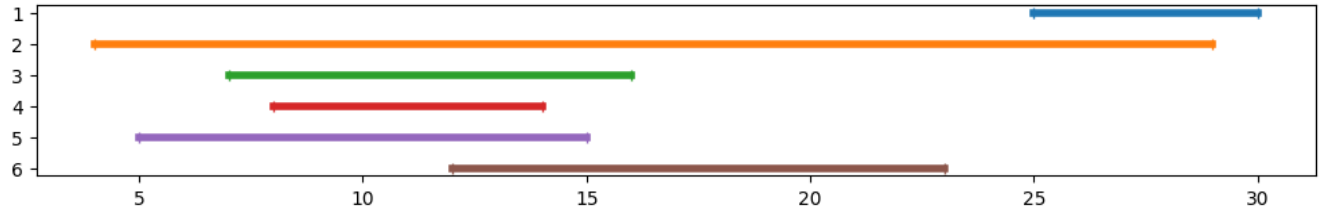


Так как ниточки 3 и 4 уже пересекаются, то для получения красоты хотя-бы 1 и красоты хотя-бы 2 не нужно совершать ни одной операции.

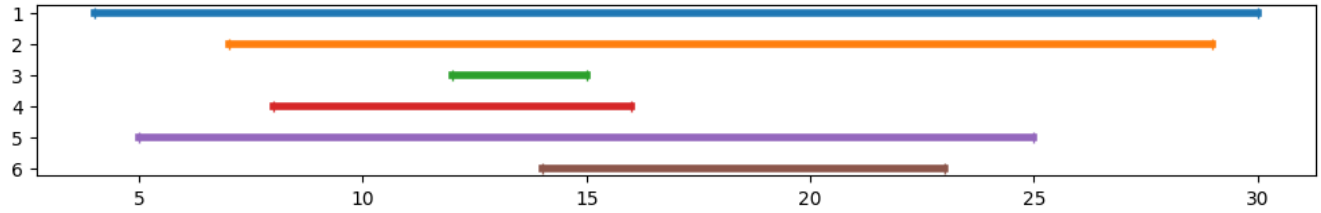
Для получения красот 3 и 4 можно применить операцию *перенатягивания* к ниточкам 2 и 5 и получить ниточки $(4, 29)$ и $(5, 15)$



Для получения красоты 5 можно дополнительно применить операцию к ниточкам 3, 6 и получить $(7, 16)$ и $(12, 23)$.



Для того, чтобы все 6 ниточек пересекались, можно применить операции к ниточкам 1 и 5, 2 и 3, 4 и 6 и получить следующую конфигурацию:



Система оценки

Тесты к этой задаче состоят из шести групп. Баллы за каждую группу ставятся только при прохождении всех тестов группы и всех тестов некоторых из предыдущих групп. Обратите внимание, прохождение тестов из условия не требуется для некоторых групп.

Группа	Баллы	Доп. ограничения			Необх. группы	Комментарий
		n	q	k_i		
0	0	–	–	–	–	Тесты из условия.
1	14	$n \leqslant 100$	–	–	0	–
2	16	$n \leqslant 3000$	–	–	0, 1	–
3	13	–	–	–	–	Ниточки попарно не пересекаются
4	25	–	$q = 1$	$k_i = n$	–	–
5	17	–	$q = 1$	$k_i \leqslant 10$	–	–
6	15	–	–	–	0 – 5	–

Задача McLaren. Монстры и мечи

Имя входного файла:	`input.txt` или стандартный поток ввода
Имя выходного файла:	`output.txt` или стандартный поток вывода
Ограничение по времени:	1 секунда
Ограничение по памяти:	512 мегабайт

Дано n монстров, выстроенных в ряд, для каждого монстра известно два значения: h_i — здоровье монстра и r_i — награда за победу над данным монстром, выраженная в монетах. Рыцарю необходимо победить всех монстров.

Для сражения с монстрами у рыцаря есть m типов мечей, для каждого типа меча известны его характеристики: s_j — сила меча и c_j — цена меча в монетах. Для покупки меча с ценой c_j у рыцаря должно быть хотя бы c_j монет. После покупки меча количество монет у рыцаря уменьшается на c_j. Изначально у рыцаря имеется x монет.

После покупки меча его можно использовать не более k раз при сражениях с монстрами. Меч каждого типа можно приобрести произвольное количество раз. Мечом силы s_j можно убить монстра со здоровьем h_i, если $s_j \geqslant h_i$. В каждый момент времени рыцарь может владеть только одним мечом, то есть после покупки нового меча старый меч уже нельзя будет использовать (но можно будет в будущем снова купить меч прошлого типа).

Монстров необходимо побеждать в фиксированном порядке: от первого до последнего. Требуется выяснить, сможет ли рыцарь это сделать.

Формат входных данных

Первая строка содержит четыре целых числа n, m, k и x ($1 \leqslant n, m \leqslant 500\,000$, $1 \leqslant k \leqslant n$, $1 \leqslant x \leqslant 10^9$) — количество монстров, количество типов мечей, максимальное количество раз, которое можно использовать меч и изначальное количество монет у рыцаря.

В следующих n строках даны описания монстров. В i-й строке находятся два целых числа: h_i и r_i ($1 \leqslant h_i, r_i \leqslant 10^9$) — здоровье i-го монстра и награда за победу над ним.

В следующие m строк дано описание характеристик мечей. В j-й строке находятся два целых числа s_j и c_j ($1 \leqslant s_j, c_j \leqslant 10^9$) — сила и цена j-го меча.

Формат выходных данных

Выведите «`Yes`» (без кавычек), если рыцарь может победить всех монстров, и «`No`» (без кавычек) иначе.

Примеры

ввод	вывод
3 3 2 5 1 1 5 5 9 5 9 10 1 1 5 1	Yes
5 5 5 6 3 6 2 4 1 1 4 4 8 4 2 7 7 7 3 4 5 9 6 4	No

Пояснение

В первом примере рыцарь может сначала купить третий меч, с его помощью победить первого и второго монстров. После чего у него станет $5 - 1 + 1 + 5 = 10$ монет. Благодаря чему он сможет купить первый меч и победить последнего монстра.

Во втором примере рыцарь не сможет победить всех монстров, потому что нет типа меча, при помощи которого можно было бы победить пятого монстра.

Система оценки

Тесты к этой задаче состоят из 9 групп. Баллы за каждую группу ставятся только при прохождении всех тестов группы и всех тестов некоторых из предыдущих групп. Обратите внимание, что прохождение тестов из условия не требуется для некоторых групп. **Offline-проверка** означает, что результаты тестирования вашего решения на данной группе станут доступны только после окончания соревнования. Итоговый балл за каждую группу равняется максимальному баллу, полученному за эту группу тестов по всем отправленным посылкам.

Группа	Баллы	Доп. ограничения			Необх. группы	Комментарий
		n	m	k		
0	0	–	–	–	–	Тесты из условия.
1	11	–	–	$k = 1$	–	–
2	9	$n \leqslant 100$	$m \leqslant 100$	$k = n$	–	–
3	14	$n \leqslant 100\,000$	$m \leqslant 3000$	$k = n$	2	–
4	16	–	–	$k = n$	2, 3	–
5	7	$n \leqslant 400$	$m \leqslant 400$	–	0, 2	–
6	8	$n \leqslant 3000$	$m \leqslant 3000$	–	0, 2, 5	–
7	10	$n \leqslant 150\,000$	$m \leqslant 150\,000$	–	0, 2, 3, 5, 6	–
8	12	$n \leqslant 300\,000$	$m \leqslant 300\,000$	–	0, 2, 3, 5 – 7	–
9	13	–	–	–	0 – 8	**Offline-проверка.**