https://codeforces.com/contest/1415/problem/A

A. Побег из тюрьмы

ограничение по времени на тест: 1 секунда
ограничение по памяти на тест: 256 мегабайт
ввод: стандартный ввод
вывод: стандартный вывод

Некоторая тюрьма может быть представлена в виде прямоугольной таблицы с n строками и m столбцами, каждая клетка которой — камера. Таким образом, всего есть $n \cdot m$ камер. В тюрьме $n \cdot m$ заключенных, по одному в каждой камере. Обозначим клетку-камеру в i-й строке и в j-м столбце как (i, j).

В клетке (r, c) заключенные прорыли тоннель, который можно использовать для побега! Чтобы не попасться, они будут убегать ночью.

В начале ночи каждый заключенный находится в своей клетке. Когда наступает ночь, они могут начать двигаться в соседние клетки. Формально, за одну секунду заключенный, находящийся в клетке (i, j), может переместиться в любую из клеток $(i-1, j)$, $(i+1, j)$, $(i, j-1)$ или $(i, j+1)$, если они находятся на территории тюрьмы. Он также может остаться в клетке (i, j).

Заключенные хотят знать минимальное необходимое время для того, чтобы все они смогли собраться в клетке (r, c). Обратите внимание, что в любой клетке одновременно могут находиться сколько угодно заключенных.

Входные данные
Первая строка содержит одно целое число t $(1 \le t \le 10^4)$ — количество тестовых случаев.

Каждая из следующих t строк содержит четыре целых числа n, m, r, c $(1 \le r \le n \le 10^9, 1 \le c \le m \le 10^9)$.

Выходные данные
Выведите t строк — ответы для каждого тестового случая.

Пример

входные данные	Скопировать
3 10 10 1 1 3 5 2 4 10 2 5 1	

выходные данные	Скопировать
18 4 6	

Разбор задачи A

1415A - Побег из тюрьмы

Задача эквивалентна нахождению расстояния до самой далекой клетки из (x, y). Легко видеть, что один из оптимальных путей из (i, j) в (x, y) имеет форму буквы L, и расстояние просто равно манхэттенскому расстоянию между двумя клетками.

Наибольшее расстояние, которое заключенный может пройти, меняя строчки, равно $max(x-1, n-x)$, а меняя строчки — $(y-1, m-y)$. Поэтому ответ просто равен $max(x-1, n-x) + max(y-1, m-y)$.

https://codeforces.com/contest/1415/problem/B

B. Перекраска улицы

ограничение по времени на тест: 1 секунда
ограничение по памяти на тест: 256 мегабайт
ввод: стандартный ввод
вывод: стандартный вывод

На некоторой улице построены n домов, по порядку пронумерованных от 1 до n. Дом номер i изначально покрашен в цвет c_i. Улица называется красивой, если все дома на ней покрашены в один и тот же цвет. Маляр Том — ответственный за перекраску улицы так, чтобы она стала красивой. Скорость работы Тома определяется целым числом k.

За один день Том может перекрасить некоторое количество домов, выполнив два следующих шага:

1. Сначала он выбирает два целых числа l и r так, что $1 \le l \le r \le n$ и $r - l + 1 = k$.
2. Затем для всех домов i таких, что $l \le i \le r$, он может либо перекрасить этот дом в любой цвет на свой выбор, либо пропустить и не перекрашивать этот дом.

Обратите внимание, что в один и тот же день Том может перекрашивать дома в разные цвета.

Том хочет знать минимальное количество дней, необходимое для того, чтобы сделать улицу красивой.

Входные данные

Первая строка содержит целое число t ($1 \le t \le 10^4$) — количество тестовых случаев. Далее следуют описания тестовых случаев.

Первая строка каждого тестового случая содержит одно целое число n и k ($1 \le k \le n \le 10^5$).

Вторая строка содержит n целых чисел. i-е из этих чисел равно c_i ($1 \le c_i \le 100$) — цвету, в который покрашен дом номе i изначально.

Гарантируется, что сумма n по всем тестовым случаям не превосходит 10^5.

Выходные данные

Выведите t строк, каждая из которых содержит одно целое число: для каждого тестового случая минимальное количество дней, необходимое Тому, чтобы сделать улицу красивой.

Пример

входные данные	Скопировать
3 10 2 1 1 2 2 1 1 2 2 2 1 7 1 1 2 3 4 5 6 7 10 3 1 3 3 3 3 1 2 1 3 3	

выходные данные	Скопировать
3 6 2	

Примечание

В первом тестовом случае Том может покрасить дома 1 и 2 в первый день в цвет 2, дома 5 и 6 во второй день в цвет 2, и последний дом в цвет 2 в третий день.

Во втором тестовом случае Том может, например, потратить 6 дней на покраску домов 1, 2, 4, 5, 6, 7 в цвет 3.

В третьем тестовом случае Том может покрасить первый дом в первый день, а дома 6, 7 и 8 во второй день в цвет 3.

Разбор задачи B

1415B - Перекраска улицы

Если мы хотим покрасить все дома на улице в цвет x, то легко видеть, что мы должны перекрасить все дома цветов, отличных от x, и не обязательно перекрашивать дома, уже имеющие цвет x. Мы можем использовать следующий жадный алгоритм для минимизации числа дней.

Найдем самый левый дом, еще не покрашенный в цвет x. Пусть этот дом имеет номер i. Тогда покрасим все дома в отрезке $[i, i+k-1]$ в цвет x. Повторим, пока все дома не станут покрашенными в цвет x. Почему это решение оптимально? Когда мы нашли самый левый дом, имеющий цвет x, понятно, что так или иначе мы должны его перекрасить. Так как он самый левый, то все дома левее уже имеют цвет x. Чтобы увеличить наши шансы сразу покрасить больше домов, которые нужно будет покрасить, мы позиционируем отрезок, который мы красим, максимально сдвинув его вправо. Такой жадный алгоритм может быть реализован одним линейным проходом.

Как же выбрать цвет x, в который мы покрасим все дома? Так как количество цветов ограничено, мы можем попробовать все варианты и выбрать минимальный ответ.

Итоговая сложность: $O(n \cdot max(c))$.

Использование памяти: $O(n)$.

https://codeforces.com/contest/1415/problem/C

C. Попрыгунчик

ограничение по времени на тест: 1 секунда
ограничение по памяти на тест: 256 мегабайт
ввод: стандартный ввод
вывод: стандартный вывод

Вы создаете уровень для некоторой мобильной игры. Уровень состоит из нескольких клеточек, выстроенных в ряд слева направо и пронумерованных последовательными натуральными числами, начиная с 1. Каждую клеточку вы можете оставить пустой или расположить там платформу.

Чтобы пройти уровень, игрок должен бросить мяч слева так, чтобы он сначала упал на платформу в некоторой клеточке p, отскочил от нее, затем отскочил от платформы в клеточке $(p+k)$, затем от платформы в клеточке $(p+2k)$, и так далее от платформы в каждой k-й клеточке до тех пор, пока он не перепрыгнет правее последней клеточки. Если хотя бы одна из этих клеточек не содержит платформы, то уровень с такими значениями p и k пройти нельзя.

У вас уже есть некоторый шаблон уровня, описываемый числами a_1, a_2, a_3, ..., a_n, где $a_i = 0$ означает, что в клеточке i нет платформы, а $a_i = 1$ означает, что платформа там есть. Вы хотите модифицировать этот шаблон так, чтобы уровень можно было пройти с заданными p и k. За x секунд вы можете добавить платформу в любую пустую клеточку. За y секунд вы можете полностью убрать первую клеточку, при этом количество клеточек уменьшится на один, а оставшиеся клетки пронумеруются заново, сохраняя порядок. Других изменений вы вносить не можете. Вы **не можете** уменьшить число клеток до меньше p.

Иллюстрация к третьему тестовому случаю. Крестами отмечены удаленные клеточки. Синим показана добавленная платформа.

Какое минимальное количество секунд вам требуется, чтобы сделать возможным прохождение уровня с заданными значениями p и k?

Входные данные
Первая строка содержит количество тестовых случаев t ($1 \le t \le 100$). Далее следуют описания тестовых случаев.

Первая строка каждого тестового случая содержит три целых числа n, p и k ($1 \le p \le n \le 10^5$, $1 \le k \le n$) — начальное количество клеточек, номер первой клеточки, которая должна содержать платформу, и требуемый период прыжков мяча.

Вторая строка каждого тестового случая содержит строку $a_1 a_2 a_3 \ldots a_n$ ($a_i = 0$ или $a_i = 1$) — начальный шаблон уровня, записанный **без пробелов**.

Последняя строка каждого тестового случая содержит два целых числа x и y ($1 \le x, y \le 10^4$) — время, необходимое для добавления платформы и время, необходимое для удаления первой клеточки, соответственно.

Сумма n по всем тестовым случаям не превосходит 10^5.

Выходные данные
Для каждого тестового случая выведите одно целое число — минимальное количество секунд, необходимое вам, чтобы изменить уровень соответствующим образом.

Можно показать, что всегда возможно изменить уровень так, чтобы его можно было пройти.

Пример

входные данные — Скопировать

```
3
10 3 2
0101010101
2 2
5 4 1
00000
2 10
11 2 3
10110011000
4 3
```

выходные данные — Скопировать

```
2
4
10
```

Примечание
В первом тестовом случае лучше всего просто убрать первую клеточку, после чего все необходимые платформы будут на своих местах: ~~0~~1**0**1**0**1**0**1**0**1. Зачеркнутая цифра удалена, цифры, выделенные жирным — места, где должны быть платформы. Необходимое время равно $y = 2$.

Во втором тестовом случае лучше всего добавить платформы в клетки 4 и 5: 000**00** → 000**11**. Необходимое время равно $x \cdot 2 = 4$.

В третьем тестовом случае лучше всего удалить первую клеточку дважды и затем добавить платформу в клеточку, которая изначально была 10-й: ~~10~~1**1**00**1**10**0**0 → ~~10~~1**1**00**1**10**1**0. Необходимое время равно $y \cdot 2 + x = 10$.

Разбор задачи C

1415C - Попрыгунчик

Заметим, что вместо удаления первых клеток мы можем за y секунд увеличивать необходимое значение p на единицу, эти операции эквивалентны. Давайте переберем, каким окажется итоговое значение p после удаления клеток из начала, пусть это q ($p \le q \le n$). Тогда нужно добавить платформы в те из клеток q, $(q + k)$, $(q + 2k)$, и так далее, в которых они отсутствуют.

Вычислим массив c_i — количество клеток, в которых отсутствует платформа, среди клеток i, $(i + k)$, $(i + 2k)$, и так далее. Его можно вычислить методом динамического программирования, проводя вычисления в порядке от больших i к меньшим: $c_i = c_{i+k} + (1 - a_i)$.

Тогда время, необходимое для добавления платформ при заданном значении q равно $c_q \cdot x$, а время, необходимое для увеличения p до значения q равно $(q - p) \cdot y$. Итоговое время равно $c_q \cdot x + (q - p) \cdot y$. Осталось лишь выбрать минимум по всем допустимым значениям q.

https://codeforces.com/contest/1415/problem/D

D. XOR-пушка

ограничение по времени на тест: 2 секунды
ограничение по памяти на тест: 256 мегабайт
ввод: стандартный ввод
вывод: стандартный вывод

У Аркадия есть **неубывающий** массив натуральных чисел $a_1, a_2, \ldots, a_n$. Вы завидуете ему и хотите уничтожить это свойство. У вас есть так называемая *XOR-пушка*, которую вы можете использовать один или несколько раз.

За один шаг вы можете выбрать два **последовательных** элемента в массиве, обозначим их за x и y, удалить их из массива и на их место вставить число $x \oplus y$, где $\oplus$ обозначает операцию побитового исключающего ИЛИ. Обратите внимание, что длина массива уменьшается на один в результате этой операции. Вы не можете выполнить эту операцию, если длина массива стала единицей.

Например, если массив равен $[2, 5, 6, 8]$, то вы можете выбрать 5 и 6 и заменить их на $5 \oplus 6 = 3$. Массив становится равен $[2, 3, 8]$.

Вы хотите, чтобы массив перестал быть неубывающим. Какое минимальное количество шагов вам для этого потребуется? Если массив остается неубывающим независимо от того, какие операции вы делаете, выведите -1.

Входные данные
Первая строка содержит одно целое число n ($2 \le n \le 10^5$) — начальную длину массива.

Вторая строка содержит n целых чисел $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$) — элементы массива. Гарантируется, что $a_i \le a_{i+1}$ для всех $1 \le i < n$.

Выходные данные
Выведите одно целое число — минимальное необходимое число шагов. Если решения не существует, выведите -1.

Примеры

входные данные (Скопировать)
```
4
2 5 6 8
```
выходные данные (Скопировать)
```
1
```

входные данные (Скопировать)
```
3
1 2 3
```
выходные данные (Скопировать)
```
-1
```

входные данные (Скопировать)
```
5
1 2 4 6 20
```
выходные данные (Скопировать)
```
2
```

Примечание
В первом примере можно выбрать 2 и 5, и массив станет равным $[7, 6, 8]$.

Во втором примере можно получить только массивы $[1, 1]$, $[3, 3]$ и $[0]$, которые все являются неубывающими.

В третьем примере можно выбрать 1 и 2 и массив станет равным $[3, 4, 6, 20]$. Затем вы можете, например, выбрать 3 и 4 и массив станет равным $[7, 6, 20]$. Этот массив не является неубывающим.

Разбор задачи D

1415D - XOR-пушка

Сначала вычислим массив $b_1, b_2, \ldots, b_n$, где b_i равно номеру самого старшего бита, равного 1 в двоичной записи числа a_i. По условию $b_i \le b_{i+1}$. Эти величины можно вычислить, проводя деление исходных чисел на 2, пока они не станут нулем.

Заметим, что если для некоторого i выполняется равенство $b_{i-1} = b_i = b_{i+1} = t$, то можно применить одну операцию к a_i и a_{i+1}, и результирующее число будет строго меньше, чем a_{i-1}. Действительно, ведь в a_{i-1} старший единичный бит имеет номер t, а в $a_i \oplus a_{i+1}$ бит номер t, как и все более старшие, равны нулю. Значит, если существует такое i, что легко проверить одним проходом по массиву b, то ответ равен 1.

Далее заметим, что если такого i не существует, то размер массива n не превосходит $2 \cdot \left(\lfloor \log_2 10^9 \rfloor + 1\right) = 60$! Действительно, существует не более двух чисел с одинаковым старшим единичным битом. В таких ограничениях решение задачи существенно упрощается.

Пусть решение существует. Тогда в итоговом массиве, обозначим его c, для некоторого i выполняется $c_i > c_{i+1}$. Заметим, что каждый элемент итогового массива равен побитовому исключающему ИЛИ некоего подотрезка исходного массива, причем каждый элемент исходного массива входит ровно в один такой подотрезок. Пусть для c_i это подотрезок $a_l, a_{l+1}, \ldots, a_m$, а для c_{i+1} это подотрезок $a_{m+1}, a_{m+2}, \ldots, a_r$. Тогда ясно, что для нахождения оптимального решения нам достаточно лишь перебрать все возможные значения l, m и r и проверить, что побитовое исключающее ИЛИ всех элементов на левом отрезке больше, чем на правом. Если неравенство выполняется, то обновим ответ значением $r - l - 1$. Асимптотика этой части решения $O(n^3)$ или $O(n^4)$ в зависимости от реализации.

https://codeforces.com/contest/1415/problem/E

E. Новая игра плюс!

ограничение по времени на тест: 2 секунды
ограничение по памяти на тест: 256 мегабайт
ввод: стандартный ввод
вывод: стандартный вывод

Кролик играет в игру, в которой есть n боссов, пронумерованных от 1 до n. Боссов можно побеждать в любом порядке. Каждого босса нужно победить **ровно один** раз. Также в игре присутствует параметр, называемый комбо-бонусом, изначально равный 0.

Когда игрок побеждает i-го босса, ко счету игрока добавляется текущее значение комбо-бонуса, а затем значение комбо-бонуса изменяется на величину c_i. Обратите внимание, что величина c_i может быть отрицательной, что означает, что дальнейшие боссы будут давать меньше очков.

Кролик нашел в игре лазейку. В любой момент времени он может сбросить прохождение игры и начать новое. Это сбросит текущее значение комбо-бонуса в 0, но все побежденные боссы останутся побежденными. Кроме того, текущий счет тоже сохраняется и **не** сбрасывается в ноль после этой операции. Эта лазейка может быть использована **не более** k раз. Можно сбрасывать игру после победы над любым количеством боссов, в том числе перед или после всех, также можно сбрасывать игру несколько раз подряд, не побеждая ни одного босса между сбросами.

Помогите кролику определить максимально возможный счет, который он может получить, победив **всех** n боссов.

Входные данные

В первой строке содержатся два целых числа n и k ($1 \le n \le 5 \cdot 10^5$, $0 \le k \le 5 \cdot 10^5$) — количество боссов и максимальное количество сбросов соответственно.

Вторая строка содержит n целых чисел $c_1, c_2, \ldots, c_n$ ($-10^6 \le c_i \le 10^6$) — влияние на комбо-бонус каждого из n боссов.

Выходные данные

Выведите одно целое число — максимальный счет, который кролик может получить, победив всех n боссов (обратите внимание, это число может быть отрицательным).

Примеры

входные данные	Скопировать
3 0 1 1 1	
выходные данные	Скопировать
3	

входные данные	Скопировать
5 1 -1 -2 -3 -4 5	
выходные данные	Скопировать
11	

входные данные	Скопировать
13 2 3 1 4 1 5 -9 -2 -6 -5 -3 -5 -8 -9	
выходные данные	Скопировать
71	

Примечание

В первом тесте запрещено делать сбросы. Оптимальное решение будет таким.

- Победить первого босса $(+0)$. Комбо-бонус становится равным 1.
- Победить второго босса $(+1)$. Комбо-бонус становится равным 2.
- Победить третьего босса $(+2)$. Комбо-бонус становится равным 3.

Поэтому ответ в первом примере равен $0 + 1 + 2 = 3$.

Во втором примере можно показать, что оптимальное решение будет таким:

- Победить пятого босса $(+0)$. Комбо-бонус становится равным 5.
- Победить первого босса $(+5)$. Комбо-бонус становится равным 4.
- Победить второго босса $(+4)$. Комбо-бонус становится равным 2.
- Победить третьего босса $(+2)$. Комбо-бонус становится равным -1.
- Сбросить. Комбо-бонус становится равным 0.
- Победить четвертого босса $(+0)$. Комбо-бонус становится равным -4.

Поэтому ответ будет равен $0 + 5 + 4 + 2 + 0 = 11$.

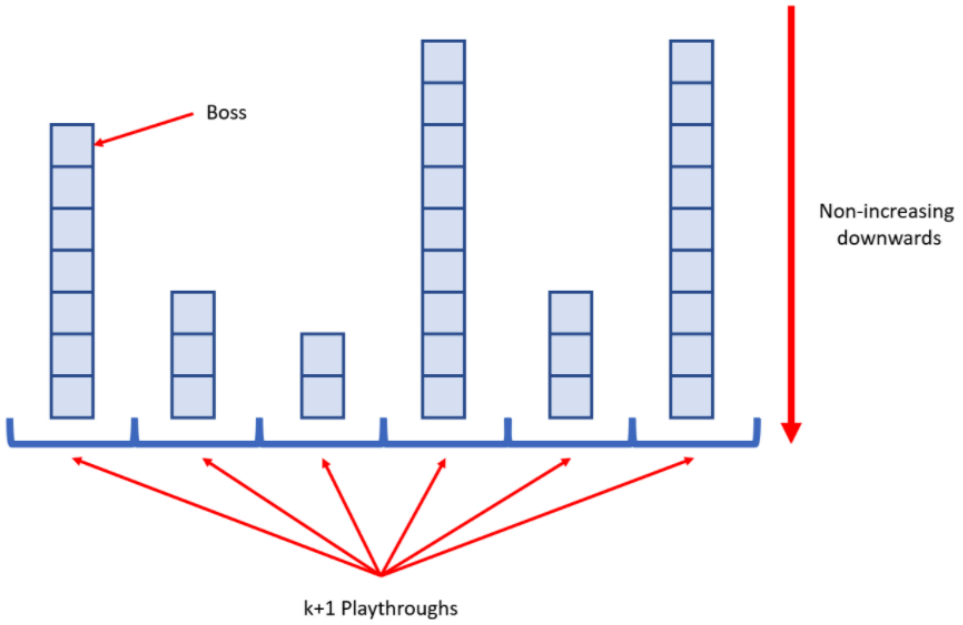
Разбор задачи E

1415E - Новая игра плюс!

Назовем победу над несколькими боссами между двух сбросов прохождением. Ясно, что прохождения не зависят друг от друга. Поэтому в задаче нужно разбить n боссов на $k+1$ прохождение.

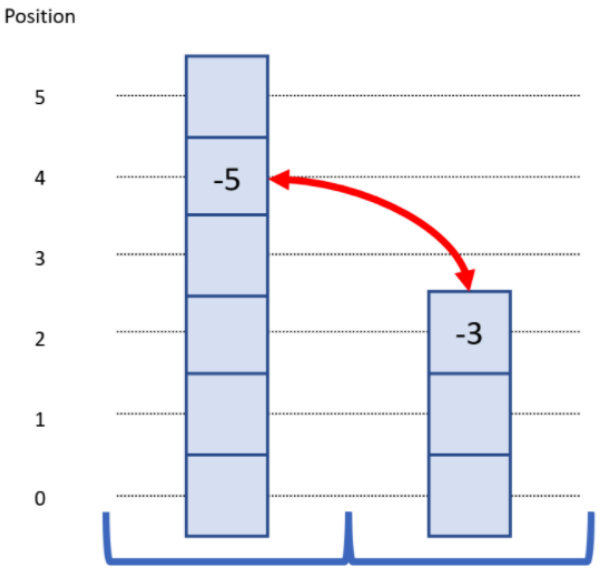
Рассмотрим прохождение с x боссами, влияние которых на комбо-бонус равно $a_1, a_2, \cdots, a_x$ в том порядке, в котором мы их проходим. Количество очков, которые мы получаем, равно $(x-1)a_1 + (x-2)a_2 + \ldots + (1)a_{x-1} + (0)a_x$. Легко показать, что внутри одного прохождения мы должны побеждать боссов в порядке невозрастания их влияния, то есть $a_1 \geq a_2 \geq \ldots \geq a_x$.

Мы можем представить решение как $k+1$ стек, каждый из которых представляет собой прохождение. Каждый стек содержит боссов, которые будут побеждены в этом прохождении. Стеки отсортированы по невозрастанию сверху вниз, что значит, что мы будем их побеждать в порядке сверху вниз.



Теперь легко понять, что случай $k = 0$ довольно прост: мы должны просто побеждать боссов в порядке невозрастания их влияния. Далее будем рассматривать только случай $k \geq 1$.

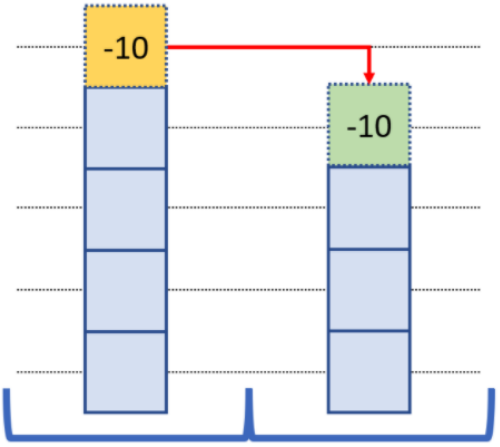
Для простоты скажем, что босс находится на месте p в некотором прохождении, если в стеке под ним находятся еще p боссов. Заметим, что если есть два босса в разных прохождениях с влияниями a и b на позициях i и j, то они дают нам $ia + jb$ очков. Поэтому ясно, что если $a \geq b$, то $i \geq j$, и наоборот. Например, данная ниже конфигурация не оптимальна, потому что если поменять местами -5 и -3, то получится ответ лучше.



Поэтому все боссы на позициях с меньшими номерами должны иметь меньшее влияние, чем все боссы на больших позициях. Это значит, что оптимальный ответ может быть получен добавлением боссов по одному в возрастающем порядке на верх стеков.

Назовем босса хорошим, если у него неотрицательное влияние, иначе назовем его плохим. Зафиксируем расположение плохих боссов и будем расставлять хороших боссов в порядке возрастания. Можно понять, что если мы поставим босса с влиянием a на стек высотой h, то к ответу добавится ah очков. Поэтому мы всегда выбираем стек с наибольшей высотой. Поэтому все хорошие боссы всегда попадут на один и тот же стек, и этот стек будет иметь наибольшую высоту. Назовем этот стек главным, а остальные k стеков побочными.

Если существуют два побочных стека, высота которых отличается не менее чем на 2, то мы можем переложить самого верхнего плохого босса с высокого стека на более короткий стек и уменьшить потерю очков. В примере ниже босса с влиянием -10 из левого стека (высоты 5) можно переложить на правый стек (высоты 3).



Поэтому высоты наименьшего и наибольшего из этих k побочных стеков отличаются не больше чем на 1.

Пусть наименьшая высота в побочных стеках равна h. Рассмотрим нижние h боссов во всех $k+1$ стеках, и вспомним, что все боссы с большим влиянием находятся выше чем боссы с меньшим влиянием. Поэтому рассмотренные $h(k+1)$ боссов должны иметь наименьшее влияние.

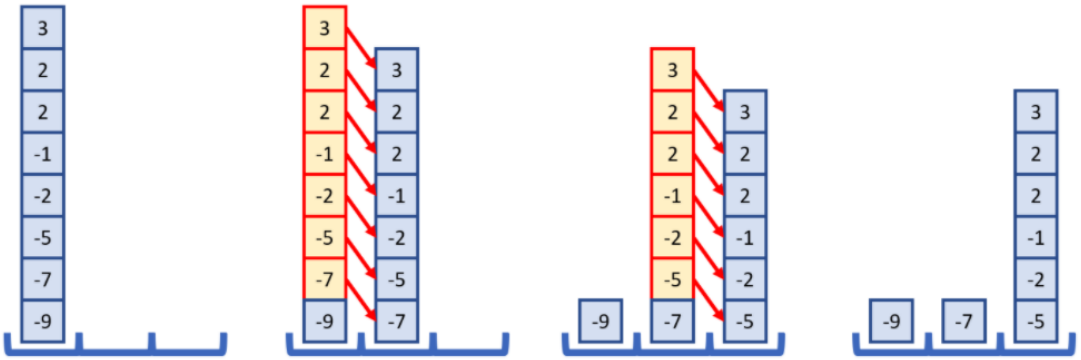
Простая жадность

Отсортируем боссов в порядке неубывания влияний. Для каждого возможного префикса P, который содержит только плохих боссов, рассмотрим их равномерное распределение в $k+1$ стек. Затем возьмем всех оставшихся боссов и положим их на самый высокий стек (если таких несколько, то на любой).

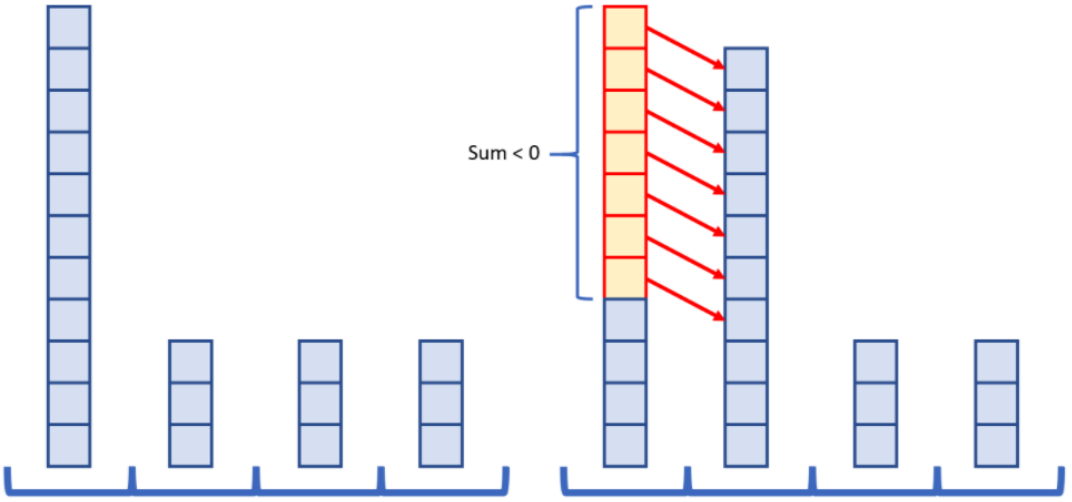
Наивное выполнение этих операций требует времени $O(n^2)$ но вычисления можно ускорить, предподсчитав взвешенные префиксные и суффиксные суммы, чтобы вычислить все ответы за $O(n)$. Итоговая асимптотика $O(n \log n)$ из-за сортировки.

Жадность посложнее

Возьмем все n влияний, сложим в один «скользящий» стек в неубывающем порядке снизу вверх. Далее будем перемещать этот стек по кругу по $k+1$ прохождениям и каждый раз будем оставлять самого нижнего босса на очередном стеке. Ниже показан пример с $n = 8$ и $k = 2$.



Каждый раз, когда мы перемещаем стек, мы уменьшаем ответ на сумму всех элементов, которые мы перемещаем вниз на одну позицию. Поэтому мы должны перемещать стек дальше, только если сумма этих величин меньше 0, и это даст нам оптимальный ответ.



Заметим, что влияния, которые мы перемещаем, всегда образуют суффикс влияний, отсортированных в порядке неубывания. Мы остановимся, когда сумма на очередном суффиксе станет неотрицательной. Поэтому мы можем найти правильную конфигурацию, сразу найдя правильный суффикс и расставив остальные элементы поровну.

Гениальная жадность (решение K0u1e)

Отсортируем влияния в порядке невозрастания $a_1, a_2, \ldots, a_n$. Будем поддерживать кучу, которая изначально содержит $k+1$ нулей. Мы будем рассматривать влияния в порядке неубывания.

Чтобы обработать очередное влияние a_i, выполним следующие операции:

- Найдем наибольшее число x в куче и уберем его из кучи
- Добавим x в наш счет
- Добавим в кучу $x + a_i$

Ответ будет равен счету в конце. Доказательство того факта, что это решение эквивалентно предыдущему, оставлено читателю в качестве упражнения.

https://codeforces.com/contest/1415/problem/F

F. Тортики для клонов

ограничение по времени на тест: 3 секунды
ограничение по памяти на тест: 256 мегабайт
ввод: стандартный ввод
вывод: стандартный вывод

Вы живете на числовой прямой и изначально (в момент времени $t = 0$) находитесь в точке $x = 0$. На прямой происходит n событий следующего вида: в момент времени t_i в координате x_i появляется небольшой тортик. Чтобы получить этот тортик, нужно в этот момент времени находиться в этой координате, иначе тортик сразу портится. Никакие два тортика не появляются в одной и той же точке.

Вы можете перемещаться на 1 единицу длины за единицу времени. Также вы обладаете возможностью мгновенно создавать своего клона в той же позиции, в которой вы сейчас находитесь. Клон не может двигаться, но он будет за вас собирать все тортики, появляющиеся в этой позиции. Клон **исчезнет**, когда вы создадите нового клона. Если новый клон создается в момент времени t, то старый клон может собирать тортики до момента t включительно, а новый — начиная с момента времени t включительно.

Можете ли вы собрать все тортики (сами или с помощью клонов)?

Входные данные
Первая строка содержит одно целое число n ($1 \le n \le 5000$) — количество тортиков.

Следующие n строк содержат по два целых числа t_i и x_i ($1 \le t_i \le 10^9$, $-10^9 \le x_i \le 10^9$) — время и координату появления очередного тортика.

Все времена появления тортиков различны и даны в порядке возрастания, а все координаты появления тортиков **различны**.

Выходные данные
Выведите «YES», если можно собрать все тортики, и «NO» иначе.

Примеры

входные данные | Скопировать
```
3
2 2
5 5
6 1
```
выходные данные | Скопировать
```
YES
```

входные данные | Скопировать
```
3
1 0
5 5
6 2
```
выходные данные | Скопировать
```
YES
```

входные данные | Скопировать
```
3
2 1
5 5
6 0
```
выходные данные | Скопировать
```
NO
```

Примечание
В первом примере нужно с самого начала двигаться в сторону координаты 5, оставив клона в позиции 1 в момент времени 1, и собрав тортик в позиции 2 в момент времени 2. В момент времени 5 вы окажетесь в позиции 5 и соберете там тортик, а клон соберет последний тортик в момент времени 6.

Во втором примере нужно оставить клона в позиции 0 и начать двигаться в сторону координаты 5. В момент времени 1 клон соберет тортик. В момент времени 2 нужно создать нового клона в текущей позиции 2, в будущем он соберет последний тортик. Вы сами же как раз успеете собрать второй тортик в позиции 5.

В третьем примере никак не получается успеть собрать все тортики.

Разбор задачи F

1415F - Тортики для клонов

Пусть $mintime_i$ — это минимальное время, в которое мы можем прийти в координату x_i, и при этом все предыдущие тортики уже собраны, а последний созданный нами клон нам уже не нужен. Также пусть $dp_{i,j}$ — хранит в себе достижима ли ситуация, в которой мы только что собрали i-й тортик, а клон ждёт события j. Пусть мы сейчас находимся в i-м событии и последний созданный клон нам уже не нужен, тогда тортик появляющийся в этой точке всегда выгодно забрать клоном, а дальше есть два возможных варианта развития:

1. Мы идём к следующему событию, тогда надо просто попробовать обновить $mintime_{i+1}$, не забыв подождать пока клон заберет тортик в i-м событии.
2. Мы хотим оставить клона ждать какого-то j-го события, и сами забрать тортик в $i+1$-м событии, тогда если у нас хватает времени это сделать, то надо $dp_{i+1,j}$ сделать достижимым.

Если же мы находимся в событии, в котором мы только что собрали i-й тортик, а клон ждёт j-го, тогда если $i+1 \neq j$, то мы просто должны пойти и сами забрать $i+1$-й тортик, а иначе есть два возможных варианта развития:

1. $j+1$-е событие заберет клон, тогда надо просто попробовать обновить $mintime_{j+1}$
2. Мы хотим оставить клона ждать какого-то более позднего события k, и самостоятельно забрать $j+1$-й тортик, тогда если у нас хватает времени это сделать, то надо $dp_{j+1,k}$ сделать достижимым.

Собрать все тортики возможно, если $mintime_n \leq t_n$, либо $dp_{n-1,n}$ — достижимо.
Итоговая асимптотика $O(n^2)$, так как мы перебираем куда поставить следующего клона только если предыдущий клон уже не нужен, либо предыдущий клон ждёт следующего события.