**Тетрадь Смерти**

Вам досталась тетрадь, которая называется Тетрадь Смерти. В ней бесконечное количество страниц. На последней странице этой тетради (хех) написано правило. Оно гласит: «Вы должны записывать в эту тетрадь имена в течение *n* последовательных дней. В течение *i*-го дня вы должны записать ровно *ai* имен». Вы очень испугались (конечно вы испугались, кто бы не испугался, если бы он получил тетрадь, которая называется Тетрадь Смерти, с каким-то странным правилом, написанным в ней?).

Конечно же вы решили последовать этому правилу. Когда вы успокоились, вы разработали план, как именно вы будете записывать имена в тетрадь. Вы посчитали, что на каждой странице тетради можно записать ровно *m* имен. Вы начинаете записывать имена с первой страницы. Вы будете записывать имена на текущей страницу до тех пор, пока на ней умещается хотя бы еще одно имя. Когда текущая страница заканчивается, вы ее переворачиваете. Заметим, что вы всегда переворачиваете страницу, не важно, последний это день или нет. Если после какого-то дня на текущей странице умещается еще хотя бы одно имя, то на следующий день вы продолжите записывать имена с текущей страницы.

Теперь вас интересует следующий вопрос: сколько раз вы перевернете страницу в течение каждого дня? Вы заинтересованы в количестве переворотов страницы в течение каждого дня от 1 до *n*.

**Входные данные**

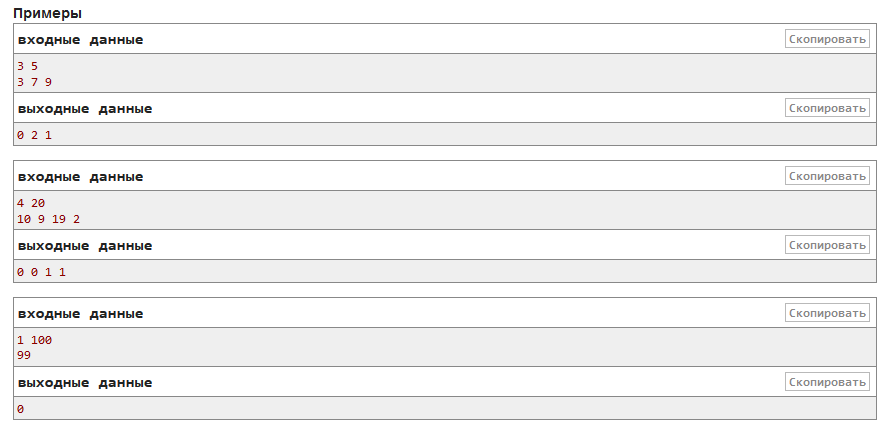
Первая строка входных данных содержит два целых числа *n*, *m* (1≤*n*≤2⋅105, 1≤*m*≤109) — количество дней, в течение которых вы будете записывать имена в тетрадь, и количество номеров, которое можно записать на одной странице тетради.

Вторая строка содержит *n* целых чисел *a*1, *a*2, …, *an* (1≤*ai*≤109), где *ai* означает количество имен, которое вы впишете в тетрадь в течение *i*-го дня.

**Выходные данные**

Выведите *n* целых чисел *t*1, *t*2, …, *tn*, где *ti* означает количество переворотов страницы в течение *i*-го дня.

*В первом тестовом примере* страницы Тетради Смерти будут выглядеть так: [1,1,1,2,2], [2,2,2,2,2], [3,3,3,3,3], [3,3,3,3]. Каждое число в массиве означает, в какой именно день будет написано имя на соответствующей позиции. Хорошо видно, что вы должны перевернуть первую и вторую страницу в течение второго дня и третью страницу в течение третьего дня.



**Драконы**

Кирито застрял на очередном уровне MMORPG, в которую он играет. Чтобы пройти игру дальше, ему нужно победить всех *n* драконов, которые живут на этом уровне. Как Кирито, так и драконы обладают силой, которая выражается целым числом. В схватке между двумя противниками исход схватки определяется их силой. Изначально сила Кирито равна *s*.

Если Кирито вступает в схватку с *i*-ым (1 ≤ *i* ≤ *n*) драконом и его сила не больше, чем сила этого дракона *xi*, то Кирито проигрывает схватку и умирает. Если же сила Кирито больше, чем сила дракона, то он побеждает дракона и получает бонус — его сила увеличивается на *yi*.

Кирито может сражаться с драконами в произвольном порядке. Определите, сможет ли он пройти на следующий уровень игры, то есть победить всех драконов, не умирая ни разу.

**Входные данные**

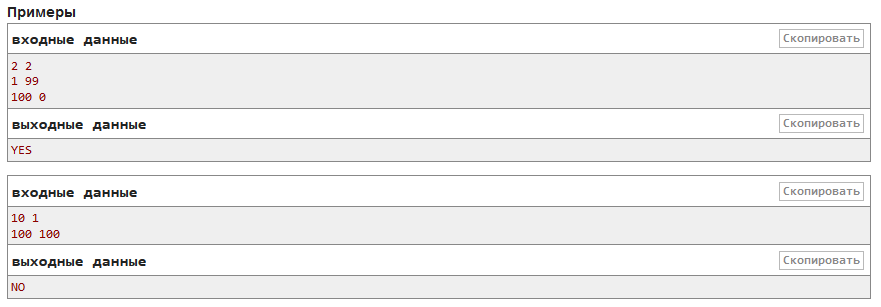
Первая строка содержит два целых числа *s* и *n* (1 ≤ *s* ≤ 104, 1 ≤ *n* ≤ 103), разделенные пробелом. Далее следуют *n* строк: *i*-ая строка содержит целые числа *xi* и *yi* (1 ≤ *xi* ≤ 104, 0 ≤ *yi* ≤ 104), разделенные пробелом — силу *i*-го дракона и бонус за победу над ним.

**Выходные данные**

В единственной строке выведите «YES» (без кавычек), если Кирито сможет пройти на следующий уровень, и «NO» (без кавычек), если не сможет.

*В первом примере* сила Кирито изначально равна 2. Поскольку сила первого дракона меньше 2, то Кирито может с ним сразиться и победить его. После этого он получает бонус, и его сила возрастает до 2 + 99 = 101. Теперь он может победить второго дракона и пройти на следующий уровень.

*Во втором примере* сила Кирито слишком мала, чтобы он мог сразиться с единственным драконом и победить.



[**OneHandSort**](https://community.topcoder.com/stat?c=problem_statement&pm=15416&rd=17457) **(**[**TCO19 SRM 755**](https://www.topcoder.com/stat?c=round_overview&rd=17457) **)**

|  |  |
| --- | --- |
|  | |
|  | Misof recently had an accident in which he managed to cut his left hand on some broken glass. He is now "all right" - meaning that he can only use his right hand for a while. Help him with some issues he has.  Misof needs to sort the objects on his shelf. The shelf has N+1 slots, numbered 0 through N from the left to the right. Slot N is currently empty and it should be empty once the shelf is sorted. Slots 0 through N-1 contain objects: for each i, slot i currently contains an object that belongs into slot **target**[i].  As Misof only has one working hand, he can only rearrange the shelf in one way: he can pick up any object and move it from its current slot into the one slot that is currently empty.  You are given the int [] **target** with at most 500 elements. Find any sequence of at most 1500 actions that will sort the shelf. Return a int[] containing instructions for Misof. Each element x of the return value represents the instruction "pick up the element that is currently in slot x and move it into the slot that is currently empty". |
|  | |
|  |  |
|  | |
| **Notes** | |
| - | While sorting the shelf Misof can (and often has to) move some of the objects more than once. |
| - | You are not required to minimize the number of moves. |
|  | |
| **Constraints** | |
| - | N will be between 1 and 500, inclusive. |
| - | **target** will contain exactly N elements. |
| - | Each element of **target** will be between 0 and N-1, inclusive. |
| - | All elements of **target** will be distinct. |
|  | |
| **Examples** | |
| 0) |  |
|  | |  |  | | --- | --- | | |  | | --- | | {0, 1, 2} | | | Returns: { } | | |  | | --- | | The shelf is already sorted, Misof can simply do nothing. | | |
| 1) |  |
|  | |  |  | | --- | --- | | |  | | --- | | {1, 2, 3, 0} | | | Returns: {2, 1, 0, 3, 4 } | | |  | | --- | | In the beginning, the shelf looks as follows:  | 1 | 2 | 3 | 0 | |  +---+---+---+---+---+  The return value describes the following solution:   1. Misof moves the object from slot 2. (This is the object that belongs into slot 3. Misof moves it to the empty slot 4.) 2. Misof moves the object from slot 1. (This is the object that belongs into slot 2. Misof moves it to the empty slot 2, where it belongs.) 3. Misof moves the object from slot 0. (This is the object that belongs into slot 1. Misof moves it to the empty slot 1, where it belongs.) 4. Misof moves the object from slot 3. (This is the object that belongs into slot 0. Misof moves it to the empty slot 0, where it belongs.) 5. Misof moves the object from slot 4. (This is the object that belongs into slot 3 - the one we already moved once at the beginning. Misof moves it to the empty slot 3, where it belongs.)   Other valid solutions exist. | | |
| 2) |  |
|  | |  |  | | --- | --- | | |  | | --- | | {1, 0, 3, 2} | | | Returns: {0, 1, 4, 2, 3, 4 } | |

|  |  |
| --- | --- |
| [**Missing Dwarf**](https://community.topcoder.com/stat?c=problem_statement&pm=15354&rd=17450) **(TCO19 SRM 754 )** | |
|  | One of Snow White's seven dwarves has gone missing! She is sure he will turn up eventually, but she needs to know his height because she is knitting him a sweater. Here is what she remembers about the heights:   * The missing dwarf is strictly taller than any of the other six dwarves. * The height of each dwarf is a positive integer. * The average of all seven heights is also a positive integer.   You are given the int [] **otherHeights**. The elements of **otherHeights** are the heights of the other six dwarves. Determine the height of the missing dwarf. As there are always multiple possibilitities, return the *smallest* one that matches the information you have. (After all, dwarves are small.) |
|  | |
| **Constraints** | |
| - | **otherHeights** will contain exactly 6 elements. |
| - | Each element of **otherHeights** will be between 1 and 1000, inclusive. |
|  | |
| **Examples** | |
| 0) |  |
|  | |  |  | | --- | --- | | |  | | --- | | {1,2,3,4,5,6} | | | Returns: 7 | | |  | | --- | | The height of the missing dwarf is 7. (The average height of a dwarf is exactly 4.) | | |
| 1) |  |
|  | |  |  | | --- | --- | | |  | | --- | | {6,5,4,3,2,1} | | | Returns: 7 | | |  | | --- | | The six heights can be given in any order. | | |
| 2) |  |
|  | |  |  | | --- | --- | | |  | | --- | | {10,10,20,20,30,30} | | | Returns: 34 | | |  | | --- | | Sometimes some of the six heights given as the input can be equal. | | |
| 3) |  |
|  | |  |  | | --- | --- | | |  | | --- | | {7,7,7,7,7,7} | | | Returns: 14 | | |  | | --- | | The missing dwarf must be taller than each of the six remaining dwarves. | | |