Task 4: `Amusement Park`

People usually go to amusement parks in groups. But what happens when, during their turn to board a ride, they are told that there are not enough seats for all of them? Some groups are willing to split, whereas some aren't.

Stuart the Snail manages the queue in front of a popular attraction. Groups of people queue up to board. During the boarding period, Stuart first receives the number of seats that can be filled. Then, he approaches each group in the queue in order, starting from the group in front. He tells the group the number of seats remaining and asks if they would like to board. The group makes a decision like this:

- If there are enough seats for the whole group, the group will board together.
- Otherwise, if the group is willing to split, they send enough people to fill up the remaining seats while the rest of the group stays in their position in the queue.
- Otherwise, the entire group stays in their position in the queue together.

It is possible that not all available seats are filled by the end of this process. Regardless, the people that manage to board will enjoy the attraction and leave without rejoining the queue.

Unfortunately, this process is very time-consuming for a snail like Stuart as he needs to move along the queue repeatedly. Please help to write a program that keeps track of the status of the queue and quickly determines which groups should split up if necessary and board the attraction. Specifically, it should support the following three operations:

- *join*: A group joins the back of the queue.
  - You are given an integer s, the size of the group.
  - You are also given an integer w, which is either 0 or 1. If $w = 0$, then the group is unwilling to split. Otherwise, if $w = 1$, then the group is willing to split. You may assume that a group's willingness to split never changes.
  - Suppose this is the i-th *join* operation, then this group is assigned the ID i, where i starts counting from 1.
- *leave*: A group leaves the queue without boarding the attraction.
  - You are given an integer i, the ID of the group leaving the queue.
  - It is guaranteed that this group is in the queue. If the group is willing to split, it is possible that some of its members have already boarded the attraction and left.

- *board*: Allow some people to board the ride according to the rules.
  - You are given an integer b, the maximum number of people that can board during this period.
  - You should decide how many people from each group should board. For additional information, refer to the *Output format* section.

Input format

Your program must read from standard input.

Note that some values in the input may not fit in a 32-bit integer.

The first line of input contains a single integer q, representing the total number of *join*, *leave* and *board* operations.

The next q lines each contain either two or three integers, following one of the three formats listed below:

- $1 \;\; s \;\; w$ describing a *join* operation;
- $2 \;\; i$ describing a *leave* operation;
- $3 \;\; b$ describing a *board* operation.

Output format

Your program must print to standard output.

For *join* and *leave* operations, **do not output anything.**

For each *board* operation, let k be the number of groups where at least one person will board the attraction. You should output $k + 1$ lines as described below.

- The first line should contain only the integer k.
- If $k > 0$, each of the remaining k lines should contain two integers. The first is the group ID, whereas the second is the number of people from that group which will board the attraction. **You should output the k lines in order of increasing group ID.** (If $k = 0$, you may ignore this part.)

Subtasks

For all testcases, the input will satisfy the following bounds:

- $1 \leq q \leq 200\,000$
- For all *join* operations, $1 \leq s \leq 200\,000$ and $0 \leq w \leq 1$
- For all *leave* operations, the group with ID i is in the queue at the time of the operation
- For all *board* operations, $1 \leq b \leq 10^{12}$
- There is at least one *board* operation

Your program will be tested on input instances that satisfy the following restrictions:

Subtask	Marks	Additional Constraints
0	0	Sample Testcases
1	12	$q \leq 1000$
2	7	$s = 1$, $w = 0$, there are no *leave* operations
3	20	$s \leq 10$, $w = 0$, there are no *leave* operations
4	16	$s \leq 10$, there are no *leave* operations
5	10	$s \leq 10$
6	35	No additional restrictions

Sample Testcase 1

This testcase is valid for subtasks 1, 5 and 6.

Input	Output
7 1 2 0 1 6 0 1 6 1 3 5 2 2 1 3 0 3 123456789012	2 1 2 3 3 2 3 3 4 3

Sample Testcase 1 Explanation

After the first 3 operations, the groups from the front to the back of the queue are as follows:

- ID 1, size 2, not willing to split
- ID 2, size 6, not willing to split
- ID 3, size 6, willing to split

For the 4th operation, seats are assigned as follows:

- Initially, there are 5 seats available. The 2 people in the group with ID 1 can board.
- There are 3 seats remaining, which is not enough for the group with ID 2. They are not willing to split, so none of them board.
- There are still 3 seats remaining, which is not enough for the group with ID 3, but they are willing to split, so 3 of them board.

There are $k = 2$ groups where at least one person boards the attraction, corresponding to group IDs 1 and 3. The output for this operation is

```
2
1 2
3 3
```

Now the groups from the front to the back of the queue are as follows:

- ID 2, size 6, not willing to split
- ID 3, size 3, willing to split

In the 5th and 6th operations, the group with ID 2 leaves while a new group with ID 4 joins. The queue looks like this:

- ID 3, size 3, willing to split
- ID 4, size 3, not willing to split

In the 7th operation, there are clearly more than enough seats for everyone in the queue. There are $k = 2$ groups where at least one person boards the attraction, corresponding to group IDs 3 and 4. The output for this operation is

```
2
3 3
4 3
```

Sample Testcase 2

This testcase is valid for all subtasks.

Input	Output
5 1 1 0 1 1 0 1 1 0 3 2 1 1 0	2 1 1 2 1

Sample Testcase 3

This testcase is valid for subtasks 1 and 6.

Input	Output
4 1 19 1 3 10 3 10 3 10	1 1 10 1 1 9 0