Problem G. Combostone

Time limit:	2 seconds
Memory limit:	512 megabytes

One big company is developing a computer game named "Combostone", that should blast the market. The rules of the game are quite complicated, so implementation of the server engine, which should adhere to the rules, was outsourced. You are asked to implement such server engine.

The players of the game are able to summon creatures to the arena, and then they can improve these creatures. Every creature has two parameters — an integer value of its attack a and an integer value of its remaining health points h. Let us denote parameters of a creature as (a, h). There are no creatures in the arena at the beginning of the game.

Players are able to use the following spells:

- *Summon Creature*: Summon new creature with parameters $(1, 1)$. If k creatures have already been summoned, the new creature gets an id $k + 1$.
- *Blessed Champion*: Double the attack of the chosen creature. If the creature have had parameters (a, h) before this spell, after the spell the creature has parameters $(2a, h)$.
- *Divine Spirit*: Double the health points of the chosen creature. If the creature have had parameters (a, h) before this spell, after the spell the creature has parameters $(a, 2h)$.
- *Molten Reflection*: Summon the new creature, which has the same parameters as the chosen creature. If k creatures have already been summoned in the game, the new creature gets an id $k + 1$.
- *Attack!*: Make a fight between two different creatures. In the fight both creatures at the same time make one hit to the enemy. This hit decreases opponent's health points by the attack value of the creature. In other words, after the fight between creatures with parameters (a_1, h_1) and (a_2, h_2) their parameters become $(a_1, h_1 - a_2)$ and $(a_2, h_2 - a_1)$, respectively. If after the fight the creature is left with 0 or less health points, it dies and can't participate in following actions of the game.

The server engine that you should implement must be able to process all the events, and for every summoned creature it must determine the number of the turn when this creature dies, or find out that it is still alive at the end of the game.

Also, the engine should correctly process the cases when the player tries to interact with the dead creatures: if the spell *Blessed Champion*, *Divine Spirit* or *Attack!* is applied to the dead creature, nothing should happen. If the spell *Molten Reflection* is applied to the dead creature, a new creature with the same parameters is summoned, but it is assumed to be killed at the turn of its creation.

Input

The first line of input contains an integer n — the number of turns in the game ($1 \le n \le 250\,000$).

The next n lines contain turn descriptions in the following format:

- 1 — apply a spell *Summon Creature*;
- 2 i — apply a spell *Blessed Champion* to the creature with id i;
- 3 i — apply a spell *Divine Spirit* to the creature with id i;
- 4 i — apply a spell *Molten Reflection* to the creature with id i;
- 5 i j — apply a spell *Attack!* to the creatures with ids i and j.

It is guaranteed that any creatures mentioned in the queries have already been summoned at the time of the query, but they can be already dead.

Output

The first line of output must contain one integer k — the number of creatures summoned in the game.

The next line must contain k integers $t_1, t_2, \ldots, t_k$. If the creature with id i is alive at the end of the game, t_i should be equal to -1, otherwise t_i should be equal to the number of the turn, when this creature was killed.

Example

standard input	standard output
16 1 2 1 3 1 1 5 1 2 3 1 1 3 3 3 3 4 1 5 1 3 3 3 5 1 3 5 4 3 5 4 3 4 1	5 13 5 14 -1 16

Note

The table below shows how the parameters of the creatures changed in the sample test.

turn	1	2	3	4	5
0	-	-	-	-	-
1	(1, 1)	-	-	-	-
2	(2, 1)	-	-	-	-
3	(2, 2)	-	-	-	-
4	(2, 2)	(1, 1)	-	-	-
5	(2, 1)	dead	-	-	-
6	(2, 2)	dead	-	-	-
7	(2, 2)	dead	(1, 1)	-	-
8	(2, 2)	dead	(1, 2)	-	-
9	(2, 2)	dead	(1, 4)	-	-
10	(2, 2)	dead	(1, 4)	(2, 2)	-
11	(2, 1)	dead	(1, 2)	(2, 2)	-
12	(2, 1)	dead	(1, 4)	(2, 2)	-
13	dead	dead	(1, 2)	(2, 2)	-
14	dead	dead	dead	(2, 1)	-
15	dead	dead	dead	(2, 1)	-
16	dead	dead	dead	(2, 1)	dead