Code Jam 2022 - Round 2

Spiraling Into Control

Problem

As punishment for being naughty, Dante has been trapped in a strange house with many rooms. The house is an $\mathbf{N} \times \mathbf{N}$ grid of rooms, with $\mathbf{N}$ odd and greater than 1. The upper left room is numbered 1, and then the other rooms are numbered 2, 3, ..., $\mathbf{N}^2$, in a clockwise spiral pattern. That is, the numbering proceeds along the top row of the grid and then makes a 90 degree turn to the right whenever a grid boundary or an already numbered room is encountered, and finishes in the central room of the grid. Because $\mathbf{N}$ is odd, there is always a room in the exact center of the house, and it is always numbered $\mathbf{N}^2$.

For example, here are the room numberings for houses with $\mathbf{N} = 3$ and $\mathbf{N} = 5$:

1	2	3
8	9	4
7	6	5

1	2	3	4	5
16	17	18	19	6
15	24	25	20	7
14	23	22	21	8
13	12	11	10	9

Dante starts off in room 1 and is trying to reach the central room (room $\mathbf{N}^2$). Throughout his journey, he can only make moves from his current room to higher-numbered, adjacent rooms. (Two rooms must share an edge — not just a corner — to be adjacent.)

Dante knows that he could walk from room to room in consecutive numerical order — i.e., if he is currently in room x, he would move to room $x + 1$, and so on. This would take him exactly $\mathbf{N}^2 - 1$ moves. But Dante wants to do things his way! Specifically, he wants to reach the central room in exactly $\mathbf{K}$ moves, for some $\mathbf{K}$ strictly less than $\mathbf{N}^2 - 1$.

Dante can accomplish this by taking one or more *shortcuts*. A shortcut is a move between rooms that are not consecutively numbered.

For example, in the 5×5 house above,

- If Dante is at 1, he cannot move to 17, but he can move to 2 or to 16. The move to 2 is not a shortcut, since $1 + 1 = 2$. The move to 16 is a shortcut, since $1 + 1 \neq 16$.
- From 2, it is possible to move to 3 (not a shortcut) or to 17 (a shortcut), but not to 1, 16, or 18.
- From 24, Dante can only move to 25 (not a shortcut).
- It is not possible to move out of room 25.

As a specific example using the 5×5 house above, suppose that $\mathbf{K} = 4$. One option is for Dante to move from 1 to 2, then move from 2 to 17 (which is a shortcut), then move from 17 to 18, then move from 18 to 25 (which is another shortcut). This is illustrated below (the red arrows represent shortcuts):



Can you help Dante find a sequence of exactly $\mathbf{K}$ moves that gets him to the central room, or tell him that it is impossible?

Input

The first line of the input gives the number of test cases, $\mathbf{T}$. $\mathbf{T}$ test cases follow. Each test case consists of one line with two integers $\mathbf{N}$ and $\mathbf{K}$, where $\mathbf{N}$ is the dimension of the house (i.e. the number of rows of rooms, which is the same as the number of columns of rooms), and $\mathbf{K}$ is the exact number of moves that Dante wants to make while traveling from room 1 to room $\mathbf{N}^2$.

Output

For each test case, output one line containing `Case #`x`:` y, where x is the test case number (starting from 1).

If no valid sequence of exactly $\mathbf{K}$ moves will get Dante to the central room, y must be `IMPOSSIBLE`.

Otherwise, y must be an integer: the number of times that Dante takes a shortcut, as described above. (Notice that because Dante wants to finish in strictly less than $\mathbf{N}^2 - 1$ moves, he must always use at least one shortcut.) Then, output y more lines of two integers each. The i-th of these lines represents the i-th time in Dante's journey that he takes a shortcut, i.e., he moves from some room a_i to another room b_i such that $a_i + 1 < b_i$.

Notice that because these lines follow the order of the journey, $a_i < a_{i+1}$ for all $1 \leq i < y$.

Limits

Memory limit: 1 GB.
$1 \leq \mathbf{T} \leq 100$.
$1 \leq \mathbf{K} < \mathbf{N}^2 - 1$.
$\mathbf{N} \mod 2 \equiv 1$. ($\mathbf{N}$ is odd.)

Test Set 1 (Visible Verdict)

Time limit: 5 seconds.
$3 \leq \mathbf{N} \leq 9$.

Test Set 2 (Visible Verdict)

Time limit: 20 seconds.
$3 \leq \mathbf{N} \leq 39$.

Test Set 3 (Hidden Verdict)

Time limit: 20 seconds.
$3 \leq \mathbf{N} \leq 9999$.

Sample

Sample Input

```
4
5 4
5 3
5 12
3 1
```

Sample Output

```
Case #1: 2
2 17
18 25
Case #2: IMPOSSIBLE
Case #3: 2
11 22
22 25
Case #4: IMPOSSIBLE
```

Sample Case #1 is described in the problem statement. Dante's route is $1 \rightarrow 2 \rightarrow 17 \rightarrow 18 \rightarrow 25$. Because $1 \rightarrow 2$ and $17 \rightarrow 18$ are moves between consecutively numbered rooms, they are not included in the output. Only the shortcuts ($2 \rightarrow 17$ and $18 \rightarrow 25$) are included.

In Sample Case #2, there is no solution. (Recall that there is no way for Dante to move diagonally.)

In Sample Case #3, observe that 22 appears both as the end of one shortcut and the start of the next. It would not be valid to include the line `11 22 25` in the output; each line must represent a single shortcut.



There is another solution that uses only one shortcut: Dante can move from $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6$, then move from $6 \rightarrow 19$ (a shortcut), then move from $19 \rightarrow 20 \rightarrow 21 \rightarrow 22 \rightarrow 23 \rightarrow 24 \rightarrow 25$. This is also valid; there is no requirement to minimize (or maximize) the number of shortcuts taken.

In Sample Case #4, Dante cannot get to the central room (9, in this case) in just one move.

Code Jam 2022 - Round 2

Pixelated Circle

Problem

Typical computer images are matrices of pixels, with each pixel being a small square of a specific color. Drawing lines that are not perfectly parallel to the axes of the pixel matrix results in imperfections. Drawing circles is an extreme example where those imperfections arise.

Suppose we have a picture consisting of $2\mathbf{R}+1$ by $2\mathbf{R}+1$ pixels, and we number the rows and columns of pixels between $-\mathbf{R}$ and $\mathbf{R}$, such that the center pixel is at row 0 and column 0. Initially, all pixels are white. Then, a circle of radius $\mathbf{R}$ and centered in the picture can be drawn in black by the following pseudocode, where `set_pixel_to_black(x, y)` makes the pixel at row x and column y be colored black.

```
draw_circle_perimeter(R):
  for x between -R and R, inclusive {
    y = round(sqrt(R * R - x * x))   # round to nearest integer, breaking ties towards zero
    set_pixel_to_black(x, y)
    set_pixel_to_black(x, -y)
    set_pixel_to_black(y, x)
    set_pixel_to_black(-y, x)
  }
```

Notice that some pixels may be set to black more than once by the code, but the operation is idempotent (that is, calling `set_pixel_to_black` on a pixel that is already black changes nothing).

The following is pseudocode for a function to draw a filled circle (starting from an all-white picture).

```
draw_circle_filled(R):
  for x between -R and R, inclusive {
    for y between -R and R, inclusive {
      if round(sqrt(x * x + y * y)) <= R:
        set_pixel_to_black(x, y)
    }
  }
```

And finally, the following is pseudocode to incorrectly draw a filled circle:

```
draw_circle_filled_wrong(R):
  for r between 0 and R, inclusive {
    draw_circle_perimeter(r)
  }
```

Given $\mathbf{R}$, calculate the number of pixels that would have different colors between a picture in which `draw_circle_filled`($\mathbf{R}$) is called and another one in which `draw_circle_filled_wrong`($\mathbf{R}$) is called.

Input

The first line of the input gives the number of test cases, $\mathbf{T}$. $\mathbf{T}$ test cases follow. Each test case is described in a single line containing a single integer $\mathbf{R}$, the radius of the circle to draw.

Output

For each test case, output one line containing `Case #`x`:` y, where x is the test case number (starting from 1) and y is the number of pixels that would have different colors between a picture in which `draw_circle_filled`($\mathbf{R}$) is called and another one in which `draw_circle_filled_wrong`($\mathbf{R}$) is called.

Limits

Memory limit: 1 GB.
$1 \leq \mathbf{T} \leq 100$.

Test Set 1 (Visible Verdict)

Time limit: 10 seconds.
$1 \leq \mathbf{R} \leq 100$.

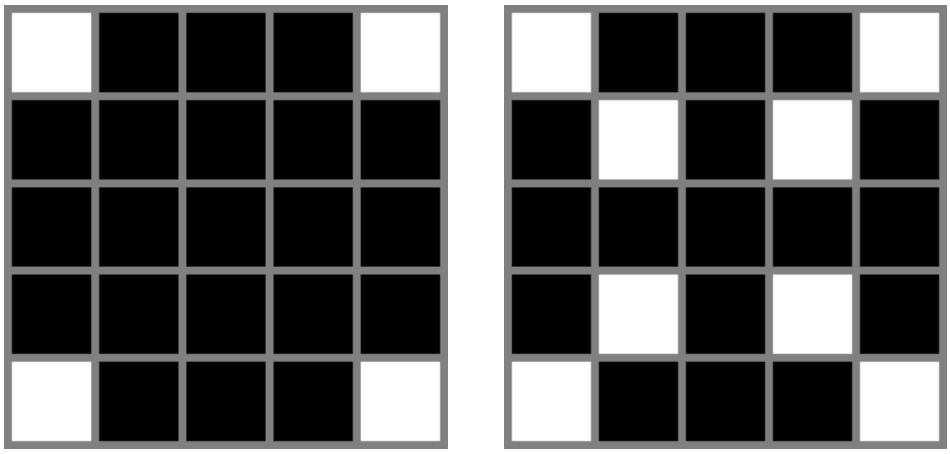
Test Set 2 (Hidden Verdict)

Time limit: 15 seconds.
$1 \leq \mathbf{R} \leq 10^5$.

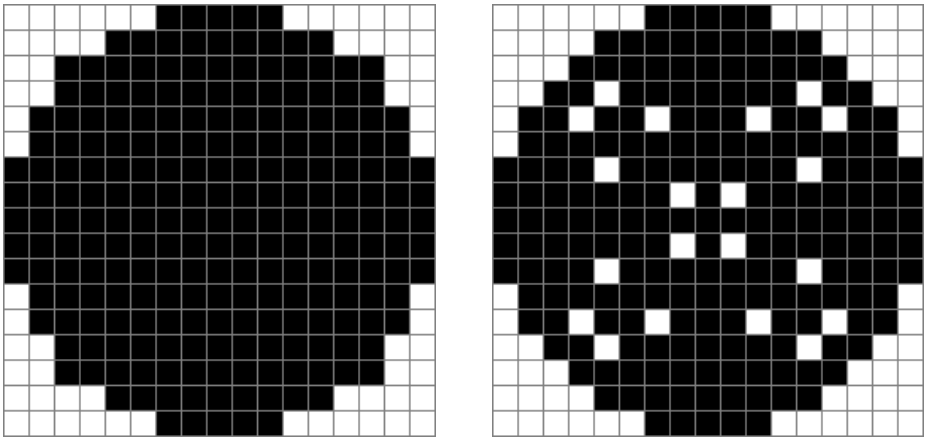
Sample

Sample Input	Sample Output
3 2 8 50	Case #1: 4 Case #2: 24 Case #3: 812

In Sample Case #1, 21 pixels are drawn in black by calling `draw_circle_filled(2)` (shown in the left picture). 17 pixels are drawn in black by calling `draw_circle_filled_wrong(2)` (shown in the right picture). Four pixels would have different colors between the two pictures: $(-1, -1)$, $(-1, 1)$, $(1, -1)$, and $(1, 1)$, where (x, y) represents the pixel at row x and column y, with the rows and columns numbered as described in the statement.



In Sample Case #2, the following pictures are the images generated by calling `draw_circle_filled(8)` (left) and `draw_circle_filled_wrong(8)` (right).



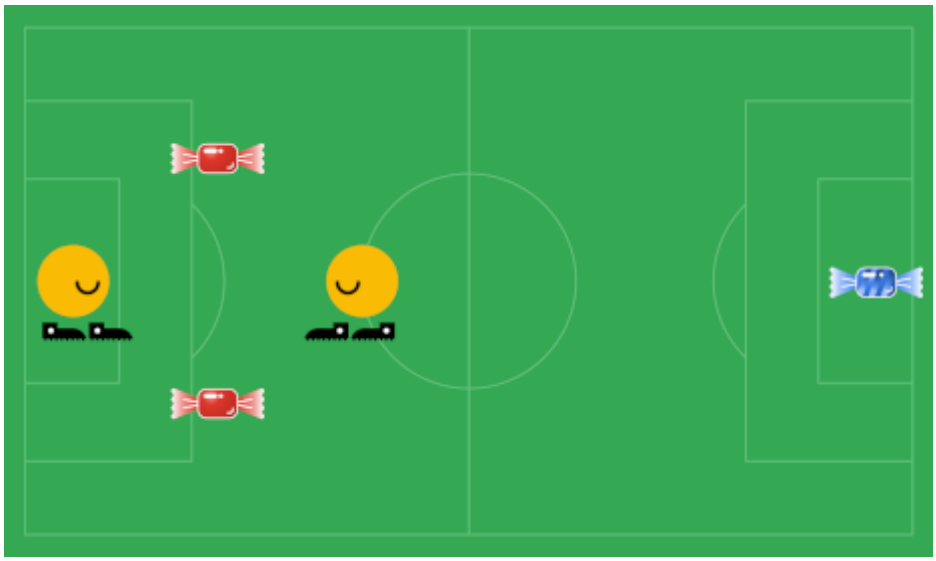
Code Jam 2022 - Round 2

Saving the Jelly

Problem

Mr. Jolly teaches football (or soccer, for US speakers) to $\mathbf{N}$ children numbered from 1 to $\mathbf{N}$. He has taken to leaving sweets on the field where the games take place, one for each child. After the game is finished, each child can grab and eat one sweet as their reward.

The children are tired after games, so each child wants to grab the sweet closest to them (using Euclidean distance). This could lead to fights — if the same sweet is closest to two or more children. To avoid that, after the game all the children stop where they are, and Mr. Jolly calls out their names, one by one. When a child's name is called, they grab the closest sweet to them (out of the ones that weren't already grabbed, of course). In the case where two or more sweets are tied for the smallest distance, Mr. Jolly can decide which one the child grabs.



This has worked very well for Mr. Jolly for a while now, but today disaster struck! While laying out the sweets, Mr. Jolly accidentally dropped his blueberry jelly that he planned to eat after all the children go home. So now there are $\mathbf{N}$ children on the field, and $\mathbf{N} + 1$ sweets. The sweets are numbered from 1 to $\mathbf{N} + 1$, with sweet 1 being Mr. Jolly's blueberry jelly. Is there a way for Mr. Jolly to save his blueberry jelly by calling the children's names in such an order that the blueberry jelly is the one sweet left over?

Input

The first line of the input gives the number of test cases, $\mathbf{T}$. $\mathbf{T}$ test cases follow. Each test begins with a line containing a single integer, $\mathbf{N}$, the number of children on the field. The next $\mathbf{N}$ lines describe the positions of the children. Each of these lines contains two integers, $\mathbf{X_i}$ and $\mathbf{Y_i}$, representing the position of the i-th child after the game ends. Then there are $\mathbf{N} + 1$ more lines that describe the positions of sweets after the game, where the first of the sweets is Mr. Jolly's blueberry jelly. Each of these lines contains two integers, $\mathbf{X_j}$ and $\mathbf{Y_j}$, representing the position of the j-th sweet.

Output

For each test case, output one line containing `Case #`x`: `y, where x is the test case number (starting from 1) and y is `IMPOSSIBLE` if there is no way Mr. Jolly can choose the children (and break ties for the closest sweet) to leave his blueberry jelly uneaten. Otherwise, if Mr. Jolly can save his blueberry jelly, y is `POSSIBLE`. If Mr. Jolly can save his jelly, output $\mathbf{N}$ additional lines representing the order the children will go and which jellies they will pick. The i-th line should contain two integers A_i and B_i representing that child A_i will go next and will pick sweet B_i. The sweet B_i must be the closest (or tied for the closest) sweet to child A_i when they go to pick their sweet.

Limits

Memory limit: 1 GB.
$1 \leq \mathbf{T} \leq 100$.
$-10^9 \leq \mathbf{X_i} \leq 10^9$, for all i.
$-10^9 \leq \mathbf{Y_i} \leq 10^9$, for all i.
$-10^9 \leq \mathbf{X_j} \leq 10^9$, for all j.
$-10^9 \leq \mathbf{Y_j} \leq 10^9$, for all j.

Test Set 1 (Visible Verdict)

Time limit: 10 seconds.
$1 \leq \mathbf{N} \leq 10$.

Test Set 2 (Hidden Verdict)

Time limit: 45 seconds.
$1 \leq \mathbf{N} \leq 1000$.

Sample

Sample Input

```
4
2
-3 0
-1 0
3 0
-2 -1
-2 1
1
0 0
1 1
2 2
3
10 0
-10 0
0 0
0 5
-1 0
5 0
0 -5
2
3 4
3 4
```

Sample Output

```
Case #1: POSSIBLE
2 2
1 3
Case #2: IMPOSSIBLE
Case #3: POSSIBLE
3 2
2 4
1 3
Case #4: POSSIBLE
1 2
2 3
```

```
5 7
3 4
5 7
```

Sample Case #1 is illustrated in the image above. Notice that each child is equally close to each of the two non-blueberry-jelly sweets. In our solution, Mr. Jolly assigns the second sweet to the second child and the third sweet to the first child, successfully leaving the first sweet (the blueberry jelly) for himself.

In Sample Case #2, the sole child is closer to the blueberry jelly than to the other sweet, so Mr. Jolly cannot prevent his precious blueberry jelly from being eaten.

In Sample Case #3, we present one of many solutions; it is actually possible to call the children in any order.

In Sample Case #4, note that children might share the same position, sweets might share the same position, and children and sweets might share the same position.

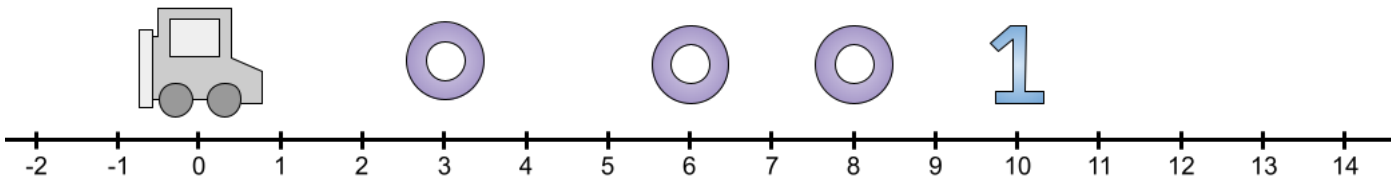
Code Jam 2022 - Round 2

I, O Bot

Problem

To welcome attendees to a developers' conference on Jupiter's moon of Io, the organizers inflated many giant beach balls. Each ball is in roughly the shape of either a 1 or a 0, since those look sort of like the letters I and O. The conference just ended, and so now the beach balls need to be cleaned up. Luckily, the beach ball cleanup robot, BALL-E, is on the job!

The conference was held on an infinite horizontal line, with station 0 in the middle, stations $1, 2, \ldots$ to the right, and stations $-1, -2, \ldots$ to the left. Station 0 contains the conference's only beach ball storage warehouse. Each other station contains at most one beach ball.



BALL-E has two storage compartments, each of which can hold a single beach ball. One compartment can only hold 1-shaped balls and the other can only hold 0-shaped balls. (The 1-shaped balls are more oblong than the 0-shaped balls, so neither shape of ball will fit in the other shape's compartment.)

BALL-E initially has both the 0 and 1 compartments empty, and it starts off at station 0. The robot can do the following things:

- Move left one station or right one station. This costs 1 unit of power.
- If there is a ball at the current station, and BALL-E is not already storing a ball of that shape, it can put the ball in the appropriate compartment. This takes 0 units of power.
- If there is a ball at the current station, BALL-E can compress it so that its shape becomes the other shape. That is, a 1-shaped ball becomes a 0-shaped ball, or vice versa. It takes $\mathbf{C}$ units of power to do this. Note that BALL-E may not change the shape of a ball that it has already put into one of its compartments.
- If BALL-E is at station 0 and is storing at least one ball, it can deposit all balls from its compartment(s) into the beach ball storage warehouse. This takes 0 units of power and leaves both compartments empty.

Notice that if BALL-E moves to a station and there is a ball there, BALL-E is not required to pick it up immediately, even if the robot has an open compartment for it. Also, if BALL-E moves to the station with the warehouse, it is not required to deposit any balls it has.

Find the minimum number of units of power needed for BALL-E to transfer all of the balls to the warehouse, using only the moves described above.

Input

The first line of the input gives the number of test cases, $\mathbf{T}$. $\mathbf{T}$ test cases follow.
The first line of each test case contains two integers, $\mathbf{N}$ and $\mathbf{C}$: the number of balls and the amount of power units needed to change the shape of a ball, respectively.
The next $\mathbf{N}$ lines describe the positions (i.e., station numbers) and the shapes of the balls. The i-th line contains two integers, $\mathbf{X_i}$ and $\mathbf{S_i}$: the position and the shape of the i-th ball, respectively.

Output

For each test case, output one line containing `Case #`x`: `y, where x is the test case number (starting from 1) and y is the *minimum* number of units of power needed to transfer all of the balls to the warehouse, as described above.

Limits

Time limit: 40 seconds.
Memory limit: 1 GB.
$1 \leq \mathbf{T} \leq 100$.
$0 \leq \mathbf{S_i} \leq 1$, for all i.
$-10^9 \leq \mathbf{X_i} \leq 10^9$, for all i.
$0 \leq \mathbf{C} \leq 10^9$.
$\mathbf{X_i} \neq 0$, for all i.
All $\mathbf{X_i}$ are distinct.

Test Set 1 (Visible Verdict)

For at most 15 cases:
$1 \leq \mathbf{N} \leq 5000$.
For the remaining cases:
$1 \leq \mathbf{N} \leq 100$.

Test Set 2 (Hidden Verdict)

For at most 15 cases:
$1 \leq \mathbf{N} \leq 10^5$.
For the remaining cases:
$1 \leq \mathbf{N} \leq 5000$.

Sample

Sample Input

```
4
5 0
3 0
6 0
8 0
10 1
15 1
5 10
3 0
6 0
8 0
10 1
15 1
5 1
3 0
6 0
8 0
10 1
15 1
2 0
1000000000 0
-1000000000 1
```

Sample Output

```
Case #1: 52
Case #2: 56
Case #3: 54
Case #4: 4000000000
```

In Sample Case #1 (illustrated in the statement), there are $\mathbf{N} = 5$ balls and $\mathbf{C} = 0$. One optimal strategy is to make three round trips from (and back to) the warehouse:

- First round trip: Move to station 3, pick up the 0 ball there and store it in the 0 compartment, move back to station 0, and deposit the ball in the warehouse. This takes 6 units of power.
- Second round trip: Move to station 8, pick up the 0 ball there, and store it in the 0 compartment. Move to station 6, change the 0 ball there into a 1 ball, and pick it up and store it in the 1 compartment. Move to station 0 and deposit both balls in the warehouse. This takes 16 units of power. (Recall that in this case, it takes 0 units of power to change a ball's shape.)
- Third round trip: Move to station 10. Change the 1 ball there into a 0 ball, and pick it up and store it in the 0 compartment. Move to station 15. Pick up the 1 ball there and store it in the 1 compartment. Move to station 0 and deposit both balls in the warehouse. This takes 30 units of power.

The total number of units of power needed to collect all the balls is 52.

Sample Case #2 is like Sample Case #1, but now with $\mathbf{C} = 10$. Now BALL-E has to use at least 56 units of power:

- First round trip: Get the ball from station 3. This takes 6 units of power.
- Second round trip: Get the differently-shaped balls from stations 6 and 10. (These are a 0 and a 1, respectively, so there is no need to change the shape of either of them.) This takes 20 units of power.
- Third round trip: Get the differently-shaped balls from stations 8 and 15. This takes 30 units of power.

Sample Case #3 is also like Sample Case #1, but now with $\mathbf{C} = 1$. Here, BALL-E needs at least 54 units of power:

- First round trip: Get the ball from station 3. This takes 6 units of power.
- Second round trip: Get the ball from station 8. When passing through station 6 on the way back, change the shape of the ball there and get it. This takes 17 units of power.
- Third round trip: Do the same with the balls at stations 15 and 10. This takes 31 units of power.

In Sample Case #4, one optimal strategy is for BALL-E to move to station -1000000000, get the 1 ball there, move to station 1000000000, get the 0 ball there, and then return to station 0 to deposit both of them.