Elevator

Time limit: 10 seconds
Memory limit: 64 MiB (per process)

This is a communication problem.

There is a building with floors numbered from 0 to N. Floor 0 is the ground floor, and above floor N is the rooftop. Thus, including the rooftop, the building has $N + 2$ levels.

Each floor f ($0 \leq f \leq N$) is occupied by exactly one resident and it has a secret integer v_f ($0 \leq v_f \leq 3$), known only to its resident. Also there is Karlsson who lives on the rooftop of this building. The objective is for Karlsson to determine as many of the values $v_0, v_1, \ldots, v_N$ as he can, while every resident **cooperates** to maximise the number of values that he can recover.

To communicate, the residents use an elevator of the building in the following way.

The elevator starts on floor 0 and travels only upwards. Inside the elevator there are buttons labelled with floors 1 through N. Initially, no buttons are pressed. Once a button is pressed, it remains pressed permanently. The elevator stops and opens on floor f if either $f = 0$ or the button corresponding to floor f has been pressed at an earlier stop. After visiting the highest floor whose button has been pressed (or floor 0 if no buttons are ever pressed), the elevator continues directly to the rooftop without stopping at any remaining floors.

When the elevator stops on floor f the following happens:

1. The resident sees the complete set of buttons that are currently pressed.
2. Based only on this information and on the value v_f, he can choose any subset of the currently unpressed buttons corresponding to floors **strictly above** their own floor. Resident presses these buttons.
3. Elevator goes up to the next lowest floor for which the corresponding button was pressed (or to the rooftop if all floors that corresponds to pressed buttons were visited).

If the elevator does not stop at some floor, the corresponding resident can't press any buttons.

When the elevator reaches the rooftop, Karlsson sees only the final set of pressed buttons. Using this information, he tries to recover as many of the values $v_0, v_1, \ldots, v_N$ as possible.

The values of v are fixed before your program starts and does not change during the process.

Implementation details

There will be T test cases.

Submit one file implementing these two functions.

The first function you should implement is the following:

```
std::vector<int> press_buttons(int subtask, int N,
                               int f, int v, std::vector<int> p)
```

- $\mathrm{subtask}$: the subtask for which this function is called for, where $0 \leq \mathrm{subtask} \leq 4$;
- N: the number of the last floor;
- f: the current floor, where $0 \leq f \leq N$;
- v: the value v_f written on floor f;
- p: the currently pressed buttons in increasing order.

This function is called when the elevator opens at floor f (if either $f = 0$ or the button corresponding to floor f has been pressed at an earlier stop). This function will be not called for specific floor if the corresponding button was not pressed before.

The return value is the list of new buttons to press. The order of the values in the returned array is not important. Every pressed button x must satisfy the following:

- $f < x \leq N$;
- x is not already in p and x appears exactly once in the returned array.

The second function you should implement is the following:

```
std::vector<int> answer(int subtask, int N, std::vector<int> p)
```

- $\mathrm{subtask}$: the subtask for which this function is called for, where $0 \leq \mathrm{subtask} \leq 4$;
- N: the number of the last floor;
- p: the final set of pressed buttons in increasing order.

This function is called exactly once per test case when the elevator reaches the rooftop.

The return array must have length of $N + 1$, the i-th element of it should be equal to v_i, while the values that Karlsson **can't recover** should be replaced with -1.

Important: The people in the building cannot communicate in any other way than through the pressed buttons. To ensure this, your program will be run as $N + 2$ separate processes, one for each floor, and one for the rooftop. In each process for the floors, there will be at most T calls of `press_buttons` function and in the process for the rooftop – exactly T calls of `answer` function. Hence your program should not assume any shared state (for example global variables). Each

process will have 64 MiB of memory available to it that is separate from the memory of other processes, and all those calls (up to $(N+1) \times T$ `press_buttons` and exactly T `answer`) need to complete in 10 seconds.

Constraints

- $N = 60$
- $T \leq 10\,000$
- $0 \leq v_i \leq 3$ for each $0 \leq i \leq N$

Example

Example test has 1 test case with the following floor values:

```
1 1 0 1 1 1 1 1 1 0 1 1 1 0 1 0 1 0 1 1 0 1 1 0 1 0 1 0 1 1 1
1 0 1 1 0 1 1 1 0 1 0 1 0 1 1 0 1 0 1 1 1 1 0 1 1 1 1 0 1 1
```

Here is an example interaction:

Participant program	Jury program
	`press_buttons(0, 60, 0, 1, {})`
`return {2}`	
	`press_buttons(0, 60, 2, 0, {2})`
`return {13, 42}`	
	`press_buttons(0, 60, 13, 0, {2, 13, 42})`
`return {}`	
	`press_buttons(0, 60, 42, 1, {2, 13, 42})`
`return {}`	
	`answer(0, 60, {2, 13, 42})`
`return {1, -1, 0, -1, -1, ..., -1}`	

This interaction recovers 0-th and 2-nd values correctly. All other returned values are equal to -1 as Karlsson did not recover them.

Subtasks

Subtask	Points	Additional constraints	Number of floors to recover for full points
0	0	The example.	–
1	15	$0 \le v_i \le 1$ $v_0 = v_N = 1$ If $v_i = 0$, then $v_{i+1} = 1$ for each $0 \le i < N$	60
2	35	$0 \le v_i \le 1$	40
3	15	$0 \le v_i \le 2$	30
4	35	$0 \le v_i \le 3$	25

Scoring

If your program returns an incorrecly recovered value or performs an invalid operation (for example returns a `vector` of the wrong length) in any of the test cases for a subtask, your submission will receive zero points for that subtask.

For a test case the *recovered count* of your submission is the number of positions whose returned value is not -1. Let K be the minimum recovered count between all test cases of a subtask (each subtask has only 1 test containing up to $10\,000$ test cases). Then, the points for each subtask is calculated according to the table below.

Subtask	Range of K	Points
0	–	0
1	$K < 60$	$0.25K$
	$K \ge 60$	15
2	$K < 30$	$0.5K$
	$30 \le K < 40$	$2K - 45$
	$K \ge 40$	35
3	$K < 30$	$0.5K$
	$K \ge 30$	15
4	$K < 20$	$0.7K$
	$20 \le K < 25$	$4K - 66$
	$K \ge 25$	35

Sample grader

The sample grader makes all function calls for all test cases in a single execution.

The input format is the following:

- line 1: three integers – the number of test cases T, the number of the last floor N and the subtask number S;
- line $1+i$: $N+1$ integers $v_0, v_1, \ldots, v_N$, where v_f is the value written on the floor f for the test case.

The output format is the following:

- line i: one integer – number of correctly recovered values of v in the test case i;
- line $T+1$: final points.

For more detailed feedback, change the macro `DETAILED` from `false` to `true` in the first line of the grader.

You can let the grader generate test cases (values of v_f) for you by changing the macro `AUTO_GENERATE` from `false` to `true` in the second line of the grader.