Dungeon Crawling

Peepo loves adventure, so he decided to explore an ancient dungeon. Unfortunately, he got lost and doesn't remember where he is. Initially Peepo starts in an unknown room and doesn't know anything about the structure of the dungeon. He doesn't know how many rooms there are, or even which room he is in. Even so, he wants to explore the dungeon.

The dungeon has many rooms, each of which has some doors. Suppose there are k doors in a room. Then each door in the room has a unique number from 0 to $k-1$ written on it. Each door is connected to some other door (possibly in some other room, or in the same room) by a tunnel. We can traverse between the rooms by going through these doors and tunnels. It is known that each tunnel connects two distinct doors and that every door is attached to a tunnel. Each room also has a light, and Peepo can observe the current state of the light in the room he is in. Initially, some rooms may have their lights on, while other rooms may have theirs off.

Peepo can toggle the state of the light in the room he is in. He can also choose any door in the room and move through the tunnel attached to the door and arrive at the room at the other end of the tunnel. In this case, he can observe through which door he emerges in the new room. Your task is to help Peepo crawl the entire dungeon and make a map out of it.

Although Peepo doesn't know about the structure of the dungeon, he knows a few key facts:

- The dungeon has between 1 and 64 rooms and between 1 and 200 tunnels.
- It is possible to move from any room in the dungeon to any other room in the dungeon using one or more tunnels.

Implementation Details

You should implement the following function:

```
Dungeon guess()
```

- This function should interact with the judge via the functions provided below.
- This function should return the map of the dungeon in a `Dungeon` structure defined below.

Available Functions

Include the header `crawl.h`. Then you can call the following functions:

```
int number_of_doors()
```

- Returns the number of doors in the current room.

```
bool is_light_on()
```

- Returns `true` if the light in the current room is on.
- Returns `false` otherwise.

```
bool toggle_light()
```

- Peepo toggles the state of the light in the current room.
- Returns the new state: `true`, if the light is on, or `false`, if the light is off.

```
int go_through_door(int d)
```

- Here d should be between `0` and `number_of_doors() - 1`.
- Peepo moves through the tunnel attached to door d of the current room and arrives at the room at the other end of that tunnel.
- Returns the number of the door through which he emerges in the new room.

You can make as many calls to the first three functions as you want. However, the number of calls to `go_through_door` should not exceed Q. See subtasks for the value of Q.

The `Dungeon` Structure

The `Dungeon` structure defined in `crawl.h` represents the map of the dungeon. The structure is defined as follows:

```
struct Tunnel {
    int room1, door1, room2, door2;
};

struct Dungeon {
    int room;
    std::vector<int> doors;
    std::vector<Tunnel> tunnels;
};
```

Let n be the number of rooms in the dungeon. Then the rooms should be numbered from 0 to $n-1$. The member variables are:

- `room`: an integer between 0 and $n-1$ representing the number of the initial room Peepo starts in.
- `doors`: a list of n integers, where `doors[i]` indicates the number of doors in room i.

- `tunnels`: a list of tunnels connecting the doors. A tunnel is represented by the `Tunnel` structure described as follows:
  - The member variables are `room1`, `door1`, `room2`, and `door2`, representing a tunnel between the door numbered `door1` of room `room1` and the door numbered `door2` of room `room2`.
  - `room1` and `room2` should be integers between 0 and $n - 1$.
  - `door1` should be an integer between 0 and `doors[room1]` -1.
  - `door2` should be an integer between 0 and `doors[room2]` -1.

Each tunnel should connect exactly two distinct doors and each door should be adjacent to exactly one tunnel. You can describe a tunnel in either direction, and the list of tunnels can contain the tunnels in any order.

Your answer will be be considered correct if it is identical to the judge's answer up to a permutation of the rooms. In other words, your dungeon is considered correct if there is a one-to-one mapping between the rooms of your dungeon and the rooms of the judge's dungeon, such that the set of tunnels, the number of doors in each rooms, and the starting rooms become identical after applying the mapping.

Subtasks

1. (7 points) $Q = 16000$, and there is exactly one way to go from any room to another room without repeating rooms.
2. (6 points) $Q = 16000$, and every room has exactly two doors.
3. (24 points) $Q = 16000$, and every room has lights off at the beginning.
4. (63 points) $Q = 120000$ (see scoring section for details of scoring).

Scoring

For the last subtask, your points depend on the number of calls made to `go_through_door`. Suppose you make q calls to `go_through_door`. Then your score will be assigned as follows:

Range of q	Score
$q > 120000$	0
$70000 < q \leq 120000$	23
$16000 < q \leq 120000$	35
$q \leq 16000$	63

Examples

Example 1

Consider the following dungeon:

Here there are three rooms and two tunnels. One tunnel connects door 0 of room 0 to door 1 of room 1. The other tunnel connects door 0 of room 1 to door 0 of room 2. Room 0 has its lights on, while other two rooms are dark. Peepo starts in room 2.

An acceptable return value of `guess` is:

```
{ 2, {1,2,1}, {{0,0,1,1}, {1,0,2,0}} }
```

The following return value is also acceptable, as tunnels can be described in either direction and in any order.

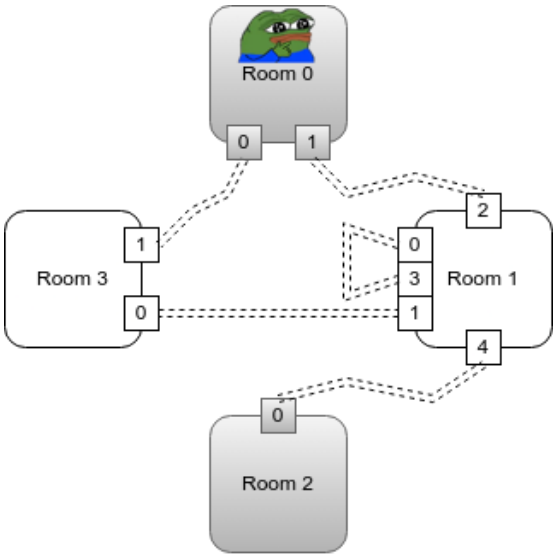
```
{ 2, {1,2,1}, {{2,0,1,0}, {1,1,0,0}} }
```

The following return value is also acceptable because it becomes identical to the dungeon after applying the mapping: $0 \mapsto 2; 1 \mapsto 0; 2 \mapsto 1$.

```
{ 0, {1,1,2}, {{2,1,1,0}, {0,0,2,0}} }
```

Example 2

Consider the following dungeon:



An acceptable return value of `guess` is:

```
{ 0, {2,5,1,2}, {{0,0,3,1}, {0,1,1,2}, {1,0,1,3}, {1,1,3,0}, {1,4,2,0}} }
```

The following return value is also acceptable because it becomes identical to the dungeon after applying the mapping: $0 \mapsto 0$; $2 \mapsto 1$; $3 \mapsto 2$; $1 \mapsto 3$.

```
{ 0, {2,2,5,1}, {{0,0,1,1}, {0,1,2,2}, {1,0,2,1}, {2,0,2,3}, {2,4,3,0}} }
```

Sample Grader

From the task page, you can download a sample grader. It consists of these files:

- `crawl.h`: the shared header file
- `crawl.cpp`: the file where you should write your solution
- `grader.cpp`: this is a grader which will simulate the judge. Your code in `crawl.cpp` needs to be compiled together with this file.
- `compile.sh`: for linux systems, run this file to compile `grader.cpp` and `crawl.cpp` together. Make sure you have g++ in system's `PATH` variable. It will produce an executable `crawl` in which you can give input to the sample grader.
- `compile.bat`: same as the above, but for windows.

The sample grader reads the input in the following format:

- line 1: n m s
- line 2: `doors[0] doors[1] ... doors[n-1]`
- line 3: `light[0] light[1] ... light[n-1]`
- line $4+i$ ($0 \le i < m$): r_1 d_1 r_2 d_2 describing a tunnel between the door numbered d_1 of room r_1 and the door numbered d_2 of room r_2.

Here, n is the number of rooms, m is the number of tunnels and s is the starting room. `doors[i]` is the number of rooms in room i. `light[i]` describes the state of the light in room i. It has value 0 if the light is off, or 1 if the light is on.

The sample grader writes the outcome (error messages, if any, query count etc.) to `stderr`. Note that the grader used in the judging server is different from the one provided here.