Task 2: Archaeologist (`archaeologist`)

A team of archaeologists are exploring some ruins, but many of them are running late, hence the earlier archaeologists are entering the ruins before the others arrive.

The K archaeologists each have a map of the ruins, and they note that there are N rooms in the ruins, labelled from 0 to $N-1$ inclusive, and some pairs of rooms are directly connected by bidirectional corridors. Furthermore, there are no cycles in the ruins, and all rooms are reachable from the entrance, which is directly connected to room 0.

Each archaeologist does not have time to backtrack (i.e. they cannot take a corridor that they have previously used in the opposite direction), nor do they have time to check how many archaeologists have already entered the ruins. However, they can change the illumination level of the room that they are standing in to any level from 0 to L inclusive before taking a corridor to the next room. All rooms in the ruins have an illumination level of 0 before the first archaeologist arrives, and will remain so until some archaeologist changes it. When currently inside a room, they cannot tell which room each corridor leads to (apart from the corridor from which they came, which they are not allowed to take), but they can tell which room they are currently in, and can see the illumination level of the room at the end of each corridor.

You may assume that each archaeologist arrives at the final room that they want to visit before any subsequent archaeologist enters the ruins.

Help the archaeologists come up with a strategy to be able to explore all rooms of the ruins!

Implementation Details

This is a communication task. You need to implement the strategy that each archaeologist should follow in order to collectively explore all the rooms. Remember that each archaeologist does not know how many other archaeologists have already entered the ruins.

You should implement the following procedure:

```
void archaeologist(int N, int K, int L, int[] map, int lightlevel, int[]
    paths)
```

- N: the number of rooms in the ruins
- K: the number of archaeologists
- L: the maximum illumination level
- map: an array of size N, where $map[0] = -1$ and for all $i \geq 1$ room i is directly connected to room $map[i]$
- $lightlevel$: the current illumination level of room 0

- $paths$: an array containing the illumination level at the end of each corridor of room 0 (other than the corridor from which the archaeologist entered room 0) — this array is shuffled arbitrarily before being given to you

Each invocation of the `archaeologist` procedure represents the behaviour of one archaeologist — all archaeologists start from room 0. The `archaeologist` procedure will be called exactly K times in sequence.

From within the `archaeologist` procedure, you may call the following two procedures, which are implemented for you in the grader:

```
void set_light(int level)
```

- This procedure sets the illumination level of the current room.
- $level$: the new illumination level, which must be between 0 and L inclusive

```
(int, int[]) take_path(int corridor)
```

- This procedure is called to make the archaeologist take a corridor into an adjacent room.
- $corridor$: the index of the corridor to take
- This procedure returns two values — the first value is the label of the new room that the archaeologist ends up in, and the second value is an array containing the illumination level at the end of each corridor of this new room (other than the corridor from which the archaeologist came from). Similarly, the second value is an array that is shuffled arbitrarily before being given to you.

The index of the corridor that is passed into `take_path` is the index into the $paths$ array of `archaeologist` (if we are currently in room 0) or array returned by the previous invocation of `take_path` (if we are currently in some room other than room 0).

Note: Invocations of the `archaeologist` procedure may be made on different processes, but we do not guarantee that every invocation is made on a different process from all other invocations. You should not rely on data that may be modified by previous invocations of the `archaeologist` procedure, such as global variables. Furthermore, you must not prematurely end execution of the process, e.g. by calling `exit()`.

Implementation Note

C++ and Java templates have been provided in the attachment. A grader file that reads from standard input is also provided, to aid you in your testing. We use a different grader implementation for grading, but with the same interface between our grader and your solution.

You are recommended to use C++ to solve this problem, even though the scientific committee has solutions in both C++ and Java. **It is not possible to make any submissions in Python.**

If you are implementing your solution in Java, please name your file `Archaeologist.java` and place your `archaeologist` function inside `class Archaeologist`.

Grader Input

A grader is provided for both C++ and Java (called `stub.cpp` and `stub.java` respectively), which reads from standard input. This grader is meant only to aid you when implementing your solution.

The input format for this grader is described below.

The first line contains three integers, N, K, and L, as described in the statement.

The second line contains $N - 1$ integers. The i^{th} integer is $map[i]$. $map[0]$ is implicitly set to -1, and hence is not part of the input file.

The grader prints debugging output when each archaeologist is invoked, and when `set_light` or `take_path` is called. The final line of the grader output will be `All rooms explored successfully` if your strategy managed to explore all rooms, or `Not all rooms were explored` otherwise.

Subtasks

The maximum execution time on each instance is 12.0s, and the maximum memory usage on each instance is 1GiB. For all testcases, the input will satisfy the following bounds:

- $1 \leq N \leq 1000$
- K is equal to the number of dead-end rooms (i.e. rooms with no corridors from it apart from the corridor from which the archaeologist entered the room)

Note that the time limit is sufficient for each archaeologist to call `set_light` once in every room that they visit.

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

Subtask	Points	Value of L	Additional Constraints
1	6	$L = 1$	$map[i] = 0$ for all $1 \leq i < N$
2	14	$L = N$	-
3	15	L is the number of corridors between room 0 and the furthest room from it	The map forms a perfect binary tree
4	18		The map array has no unique elements except for the value -1
5	47		-

Note: The distance of a room x from room 0 is the number of corridors along the shortest path from room 0 to room x. The furthest room from room 0 is the room with largest distance from room 0. When we say that the map forms a perfect binary tree, we mean that each dead-end room has the same distance from room 0, and at each non-dead-end room the archaeologist has exactly two corridors to choose from (apart from the corridor from which they entered the room).

Sample Interaction

This testcase is valid for subtasks 3, 4, and 5.

Grader input

Input

`7 4 2`
`0 0 1 1 2 2`

Sample interaction for archaeologist #1

Grader calls `archaeologist(7, 4, 2, [-1, 0, 0, 1, 1, 2, 2], 0, [0, 0])`	
Function call	**Return value**
`set_light(1)`	-
`take_path(0)`	`(1, [0, 0])`
`take_path(1)`	`(4, [])`
`set_light(2)`	-

Sample interaction for archaeologist #2

Grader calls `archaeologist(7, 4, 2, [-1, 0, 0, 1, 1, 2, 2], 1, [0, 0])`	
Function call	**Return value**
`set_light(0)`	-
`take_path(0)`	`(2, [0, 0])`
`set_light(1)`	-
`take_path(1)`	`(6, [])`
`set_light(1)`	-

Sample interaction for archaeologist #3

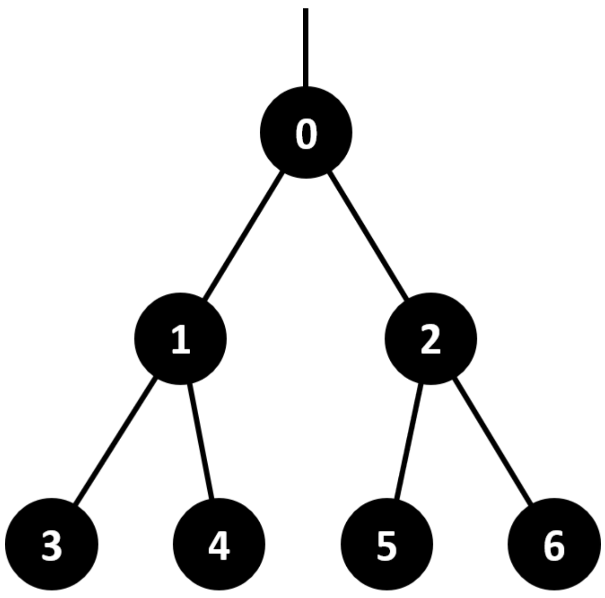
Grader calls `archaeologist(7, 4, 2, [-1, 0, 0, 1, 1, 2, 2], 0, [1, 0])`	
Function call	**Return value**
`set_light(2)`	-
`take_path(1)`	`(1, [2, 0])`
`set_light(0)`	-
`take_path(1)`	`(3, [])`

Sample interaction for archaeologist #4

Grader calls `archaeologist(7, 4, 2, [-1, 0, 0, 1, 1, 2, 2], 2, [1, 0])`	
Function call	**Return value**
`set_light(1)`	-
`take_path(0)`	`(2, [0, 1])`
`take_path(0)`	`(5, [])`
`set_light(1)`	-

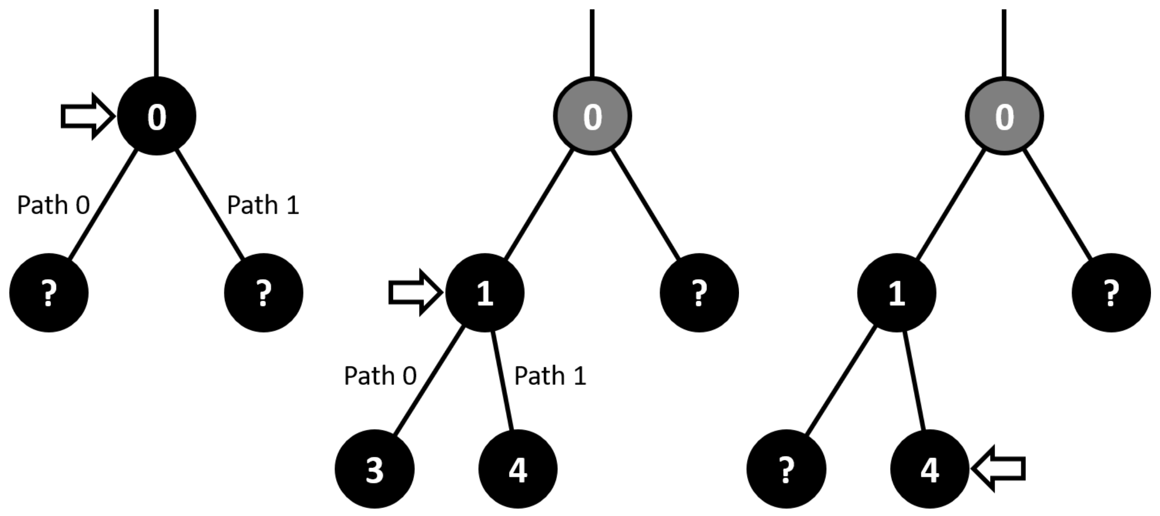
Sample Interaction Explanation

The map of the ruins looks like this, where the entrance is directly connected to room 0:

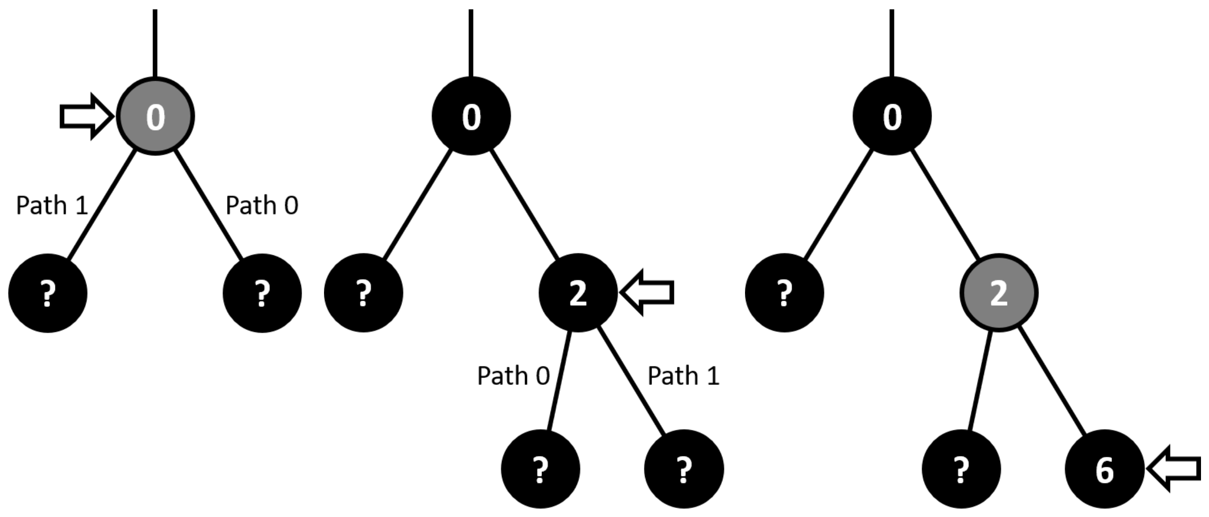


The diagrams that follow show, for each archaeologist, initially and after each call of `take_path`, where the archaeologist is and the information that they have been given. The diagrams are ordered from left to right, as the archaeologist goes deeper into the ruins. The arrow denotes the current location of the archaeologist. Black circles represent a room with an illumination level of 0, gray circles represent a room with an illumination level of 1, and white circles represent a room with an illumination level of 2, the maximum allowed. The numbers in the circles represent the room number, with question marks representing room numbers that the archaeologist does not yet know. Remember that the list of illumination levels presented to your program may be shuffled arbitrarily.

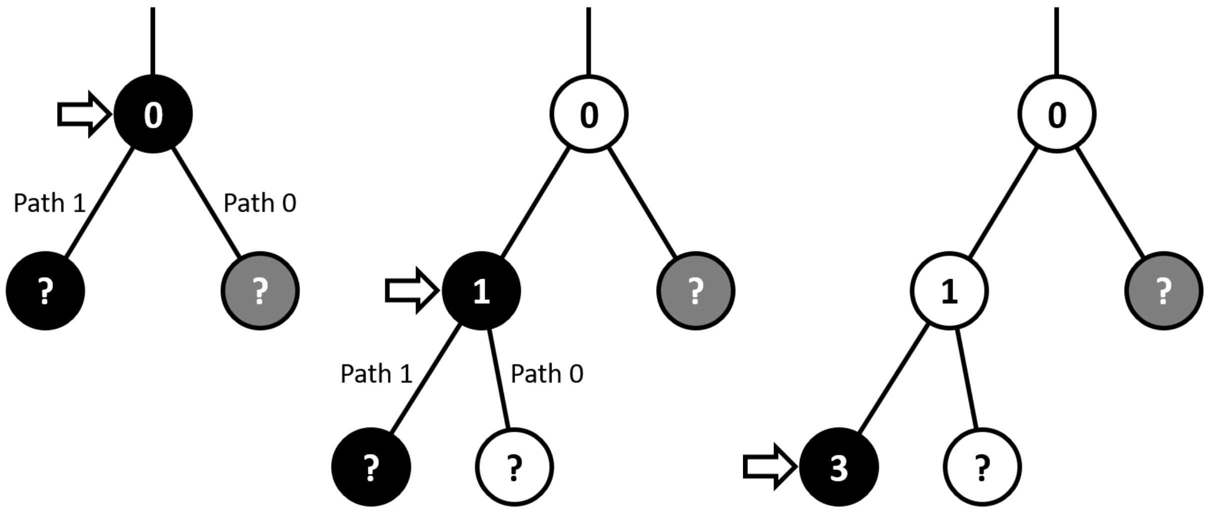
Diagrams for archaeologist #1



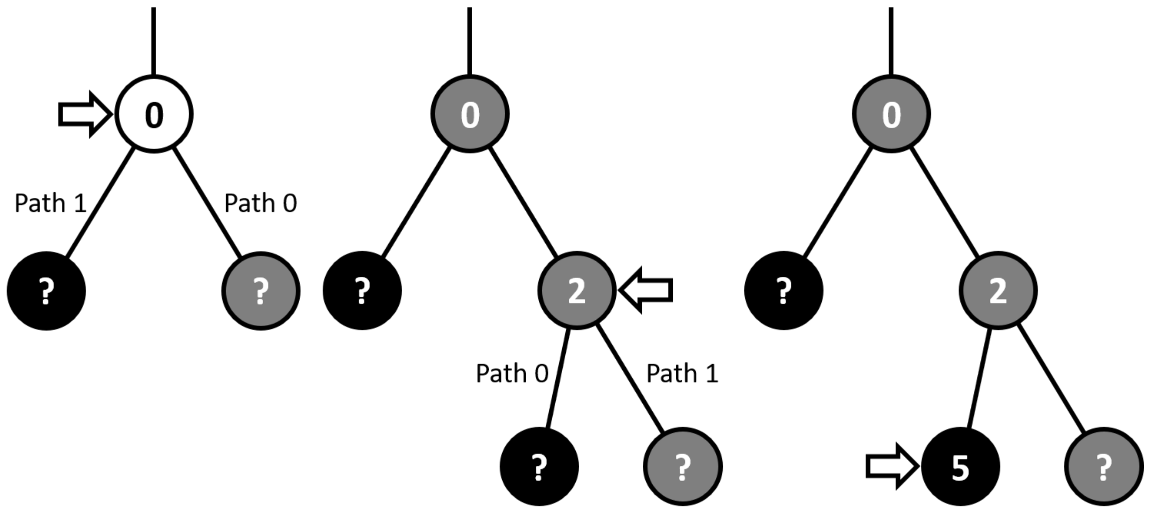
Diagrams for archaeologist #2



Diagrams for archaeologist #3



Diagrams for archaeologist #4



After all four archaeologists have executed their strategies, all rooms will have been explored by at least one archaeologist.