SP 2023 Graph

Question 1

Problem statement

In the land of Aria, there are **n** islands connected by unique bridges. Each island has a unique identifier from 0 to n-1. The main island, where the Palace of Aria stands, has an identifier **r1**. The bridge system in Aria is such that there's a unique bridge path from the main island to any other island, resulting in a tree-like bridge map. The bridge map is recorded in the Arian scrolls **p**, such that for each island **i**, the scroll mentions an integer $\mathbf{p_i}$, representing the immediate island one would pass through when traveling from the main island to island **i**. (The only exception is, for the main island, this integer $\mathbf{p_i}$ would be -1, because it doesn't have to go anywhere)

The ruler of Aria, Queen Elara, has decided to shift the Palace of Aria from island **r1** to island **r2**. This means the main island will now be **r2**. Due to this shift, the current bridge map in the Arian scrolls is no longer valid. Your task is to assist Queen Elara in deriving the new bridge map based on the Palace shift.

Class Definition

Class: `Question1`

Method: `bridgeMap`

The class `Question1` includes the method `bridgeMap` which accepts one `int` array, two `int` parameters and returns an `int` array.

```
//java
public int[] bridgeMap(int[] p, int r1, int r2)
```

```
//cpp
vector<int> bridgeMap(vector<int> p, int r1, int r2)
```

```
#python
def bridgeMap(p: List[int], r1: int, r2: int) -> List[int]
```

Constraints

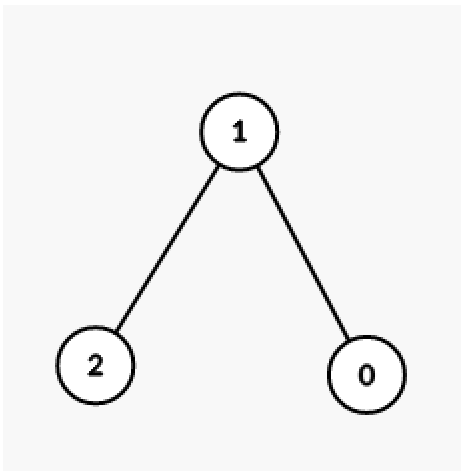
- `p.length` = `n` ; 2 ≤ `n` ≤ 1000 (n is the number of islands)
- 0 ≤ `r1` ≠ `r2` < `n`.

Examples

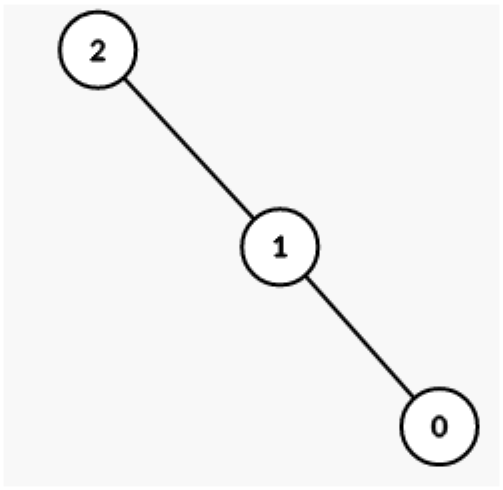
Example 0

```
Question1 q1 = new Question1();
//Returns: [1, 2, -1]
System.out.println(Arrays.toString(q1.bridgeMap({1, -1, 1}, 1, 2)));
```

There are 2 versions of the graph (Notice that, the edges and vertices are not changed, the only difference, is the presentation of the graph)



Node 1 is the root: node 1 is the closest node to both nodes 0 and 2, when reaching from these nodes to root.

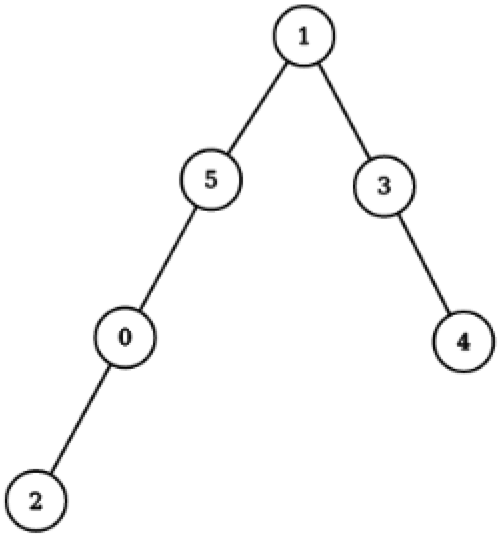


Node 2 is the root: node 2 is the closest node to node 1, and 1 is the closest node to 0, when reaching from these nodes to root.

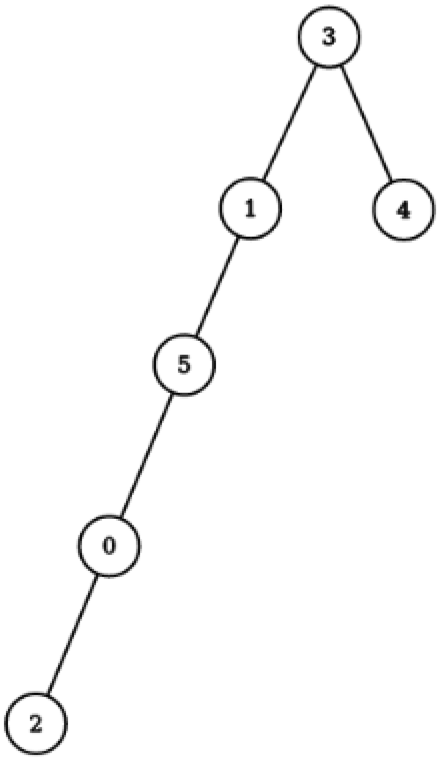
Example 1

```
Question1 q1 = new Question1();
// Returns: [5, 3, 0, -1, 3, 1]
System.out.println(Arrays.toString(q1.bridgeMap({5, -1, 0, 1, 3, 1}, 1, 3)));
```

2 versions of the graph:



Node 1 is root: Same logic with previous example, 3→1, 4→3, 5→1, 0→5, 2→0



Node 3 is root: 4→3, 1→3, 5→1, 0→5, 2→0

Test Cases

p	r1	r2	Expected Result
[1, -1, 1]	1	2	[1, 2, -1]
[5, -1, 0, 1, 3, 1]	1	3	[5, 3, 0, -1, 3, 1]
[-1, 0, 0, 1, 1]	0	3	[1, 3, 0, -1, 1]
[-1, 0, 0, 1, 1]	0	4	
[-1, 0, 0, 1, 1]	0	1	
[-1, 0, 0, 2, 2, 4, 4, 6, 6]	0	5	
[26, 2, 32, 36, 40, 19, 43, 24, 30, 13, 21, 14, 24, 21, 19, 4, 30, 10, 44, 12, 7, 32, 17, 43, 35, 18, 7, 36, 10, 16, 5, 38,	35	19	

35, 4, 13, -1, 16, 26, 1, 12, 2, 5, 18, 40, 1, 17, 38, 44, 14]			
[6, 5, 6, -1, 3, 3, 5]	3	0	
[5, 3, 6, 6, 3, -1, 5]	5	3	
[3, 2, 4, 2, -1, 7, 7, 4, 3]	4	8	
[9, 14, 13, 2, 2, -1, 14, 9, 13, 11, 11, 5, 10, 5, 10]	5	7	
[28, 6, 0, 16, 9, 18, 27, 15, 13, 34, 6, 18, 19, -1, 9, 19, 8, 0, 1, 32, 4, 1, 4, 28, 32, 34, 27, 13, 15, 26, 3, 26, 8, 3, 16]	13	17	
[-1, 4, 36, 24, 10, 29, 19, 31, 7, 0, 26, 31, 4, 18, 7, 6, 14, 24, 23, 0, 29, 14, 10, 35, 26, 1, 19, 1, 9, 36, 18, 6, 22, 35, 22, 9, 23]	0	2	
[38, 20, 34, 7, 6, 37, 5, 8, 10, 30, 6, 37, 20, 2, 16, 7, 21, 23, 22, 23, 8, 33, 11, 21, 4, 30, 15, 34, 16, 15, 10, 36, 38, 5, 4, 33, 11, -1, 22, 2, 36]	37	38	
[25, 34, 34, 27, 21, 21, 24, 36, 31, 2, 27, 8, 26, 8, 9, 35, 38, 30, 23, 26, 36, 9, 24, 31, 15, 1, 0, 30, 0, 35, 25, 38, -1, 18, 32, 1, 23, 2, 32, 18, 15]	32	4	
[24, 42, 4, 30, 29, 43, 22, 15, 26, 36, 26, 16, 3, 22, 21, 41, 18, 16, 34, 41, 12, 29, 32, 30, 43, 15, 4, 38, 36, -1, 24, 42, 18, 6, 21, 38, 6, 17, 32, 17, 3, 34, 12, 14, 14]	29	18	
[-1, 21, 24, 30, 46, 1, 16, 29, 30, 41, 18, 33, 26, 31, 12, 45, 29, 7, 16, 45, 27, 32, 41, 18, 35, 6, 9, 1, 19, 32, 22, 33, 0, 9, 6, 0, 12, 19, 21, 35, 46, 24, 31, 27, 26, 22, 7]	0	31	

Please note that the expected outputs for some test cases are not provided.

Question 2

Problem Statement

In the intergalactic realm, there exists a unique communication system between **n** planets. The system is maintained by **m** hyper-channels that connect some pairs of planets. The efficiency of communication between two planets is determined by the number of hyper-channels one needs to traverse to relay a message between them. We define the "communication delay" as the maximum number of hyper-channels required to relay a message between any two planets. Given information of **n** planets and **m** channels, your task is to calculate this delay, given the structure of hyper-channels.

Class Definition

Class: `Question2`

Method: `communicationLatency`

The class `Question2` includes the method `communicationLatency` which accepts an `int` , a 2D `int` array parameter and returns an `int`.

```
//java
public int communicationLatency(int n, int[][] channels)
```

```
//cpp
int communicationLatency(int n, vector<vector<int>> channels)
```

```
#python
def communicationLatency(n: int, channels: List[List[int]]) -> int
```

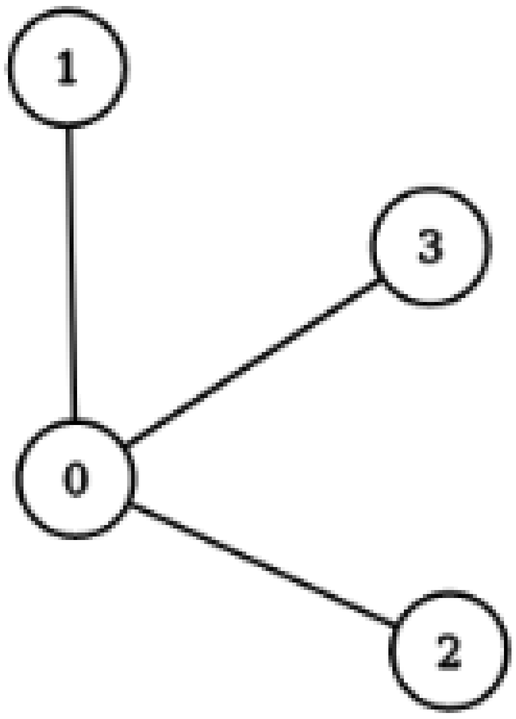
Constraints

- 1 ≤ `n` , `m` ≤ 100000
- `channels[i].length` = 2, which consists of 2 planets that can be communicated directly.
- 0 ≤ `channels[i][j]` < n; `channels[i][0]` ≠ `channels[i][1]`

Examples

Example 0

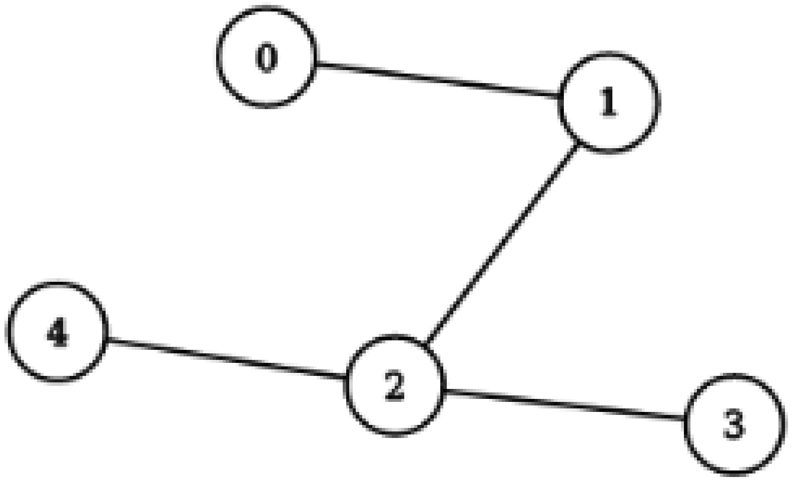
```
Question2 q2 = new Question2();
//Returns 2
System.out.println(q2.communicationLatency(4, {{0, 1}, {0, 2}, {0, 3}}));
```



The longest path is from any one node to another node, between nodes 1, 2 and 3.

Example 1

```
Question2 q2 = new Question2();
//Returns 3
System.out.println(q2.communicationLatency(5, {{0, 1}, {1, 2}, {2, 3}, {2, 4}}));
```



The longest path is the path between node 0 to either node 4, or node 3.

Test Cases

n	edges	Expected result
4	[[0, 1], [0, 2], [0, 3]]	2
5	[[0, 1], [1, 2], [2, 3], [2, 4]]	3
4	[[0, 1], [0, 2], [0, 3], [1, 2], [1, 3], [2, 3]]	
5	[[0, 1], [0, 2], [0, 3], [0, 4], [1, 2], [1, 3], [1, 4], [2, 3], [2, 4], [3, 4]]	
6	[[0, 1], [0, 2], [0, 3], [0, 4], [0, 5], [1, 2], [1, 3], [1, 4], [1, 5], [2, 3], [2, 4], [2, 5], [3, 4], [3, 5], [4, 5]]	
7	[[0, 1], [0, 2], [0, 3], [0, 4], [0, 5], [0, 6], [1, 2], [1, 3], [2, 4], [3, 5], [4, 6], [5, 6]]	
8	[[0, 1], [0, 2], [0, 3], [0, 4], [0, 5], [0, 6], [0, 7], [1, 2], [1, 3], [1, 4], [2, 5], [2, 6], [3, 7], [4, 5], [4, 6], [5, 7]]	
9	[[0, 1], [0, 2], [0, 3], [0, 4], [0, 5], [0, 6], [0, 7], [0, 8], [1, 2], [1, 3], [1, 4], [1, 5], [1, 6], [1, 7], [1, 8], [2, 3], [2, 4], [3, 5], [4, 6], [7, 8]]	
10	[[0, 1], [0, 2], [0, 3], [0, 4], [0, 5], [0, 6], [0, 7], [0, 8], [0, 9], [1, 2], [1, 3], [1, 4], [1, 5], [1, 6], [1, 7], [1, 8], [1, 9], [2, 3], [2, 4], [3, 5]]	

11	[[0, 1], [0, 2], [0, 3], [0, 4], [0, 5], [0, 6], [0, 7], [0, 8], [0, 9], [0, 10], [1, 2], [1, 3], [1, 4], [1, 5], [1, 6], [1, 7], [1, 8], [1, 9], [1, 10], [2, 3]]	
6	[[0, 1], [0, 2], [0, 3], [0, 4], [0, 5], [1, 2], [1, 3], [1, 4], [1, 5]]	
7	[[0, 1], [0, 2], [0, 3], [0, 4], [0, 5], [0, 6], [1, 2], [1, 3], [1, 4], [1, 5], [1, 6], [2, 3], [2, 4], [2, 5]]	
8	[[0, 1], [0, 2], [0, 3], [0, 4], [0, 5], [0, 6], [0, 7], [1, 2], [1, 3], [1, 4], [1, 5], [1, 6], [1, 7], [2, 3], [2, 4], [2, 5], [2, 6], [2, 7]]	
5	[[0, 1], [0, 2], [0, 3], [0, 4], [1, 2], [1, 3], [1, 4], [2, 3]]	
9	[[0, 1], [0, 2], [0, 3], [0, 4], [0, 5], [0, 6], [0, 7], [0, 8], [1, 2], [1, 3], [1, 4], [1, 5], [1, 6], [1, 7], [1, 8], [2, 3], [2, 4], [2, 5], [2, 6]]	
10	[[0, 1], [0, 2], [0, 3], [0, 4], [0, 5], [0, 6], [0, 7], [0, 8], [0, 9], [1, 2], [1, 3], [1, 4], [1, 5], [1, 6], [1, 7], [1, 8], [1, 9], [2, 3], [2, 4]]	
11	[[0, 1], [0, 2], [0, 3], [0, 4], [0, 5], [0, 6], [0, 7], [0, 8], [0, 9], [0, 10], [1, 2], [1, 3], [1, 4], [1, 5], [1, 6], [1, 7], [1, 8], [1, 9], [1, 10]]	

Please note that the expected outputs for some test cases are not provided.