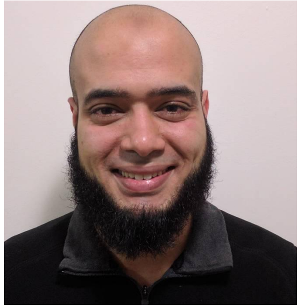
Algorithms

Topological Homework 2

Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

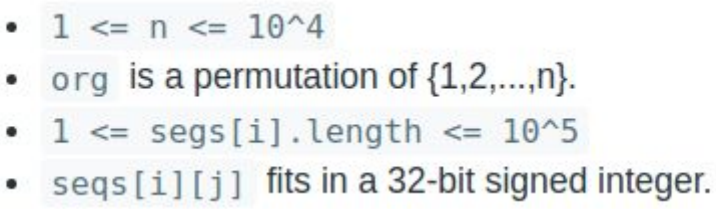


Problem 1#: LeetCode 444 - Sequence Reconstruction

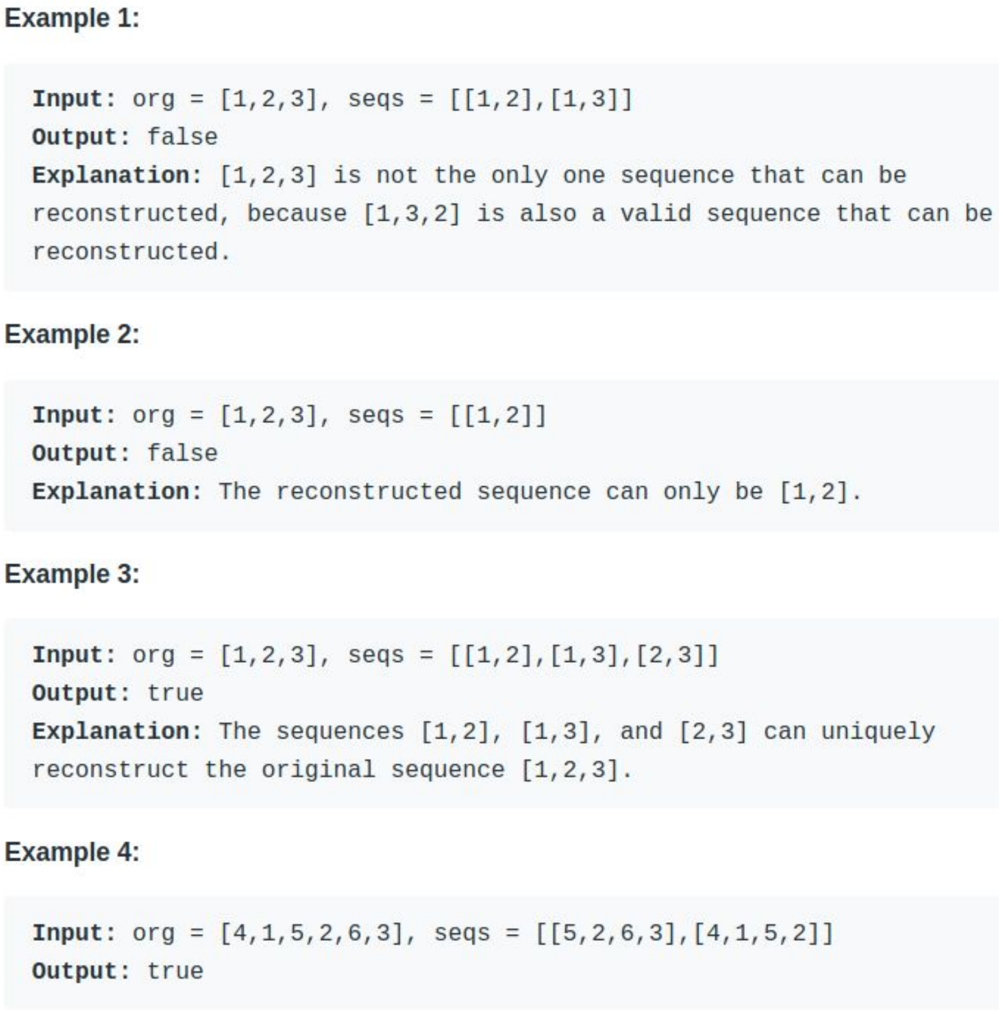
- *Rewriting the statement*
- Recall: a **subsequence** of a given sequence is a sequence that can be derived from the given sequence by **deleting** some or no elements **without** changing the **order** of the remaining elements.
  - [2, 4, 7] is subsequence of [1, 2, 3, 4, 5, 6, 7]
  - [7, 4, 2] is NOT subsequence of [1, 2, 3, 4, 5, 6, 7] as the order is maintained
- Given a list of sequences *seqs*, a sequence S is called supersequence if it contains ALL the sequences in *seqs*
  - S = [1,2,3] is supersequence for seqs = [[1,2],[1,3],[2,3]]
- Given *seq* and *org* (a permutation of the integers from 1 to n, with $1 \leq n \leq 10^4$) return true IFF org is the **shortest and unique** supersequence for seq
  - Only **one shortest** super-sequence is possible

Signature

- C++: bool sequenceReconstruction(vector<int> &org, vector<vector<int>> &seqs)
- Java: public boolean sequenceReconstruction(int[] org, List<List<Integer>> seqs)
- Python: def sequenceReconstruction(self, org: List[int], seqs: List[List[int]]) -> bool
- Javascript: var sequenceReconstruction = function(org, seqs)



- `1 <= n <= 10^4`
- `org` is a permutation of {1,2,...,n}.
- `1 <= segs[i].length <= 10^5`
- `seqs[i][j]` fits in a 32-bit signed integer.



Example 1:

```
Input: org = [1,2,3], seqs = [[1,2],[1,3]]
Output: false
Explanation: [1,2,3] is not the only one sequence that can be
reconstructed, because [1,3,2] is also a valid sequence that can be
reconstructed.
```

Example 2:

```
Input: org = [1,2,3], seqs = [[1,2]]
Output: false
Explanation: The reconstructed sequence can only be [1,2].
```

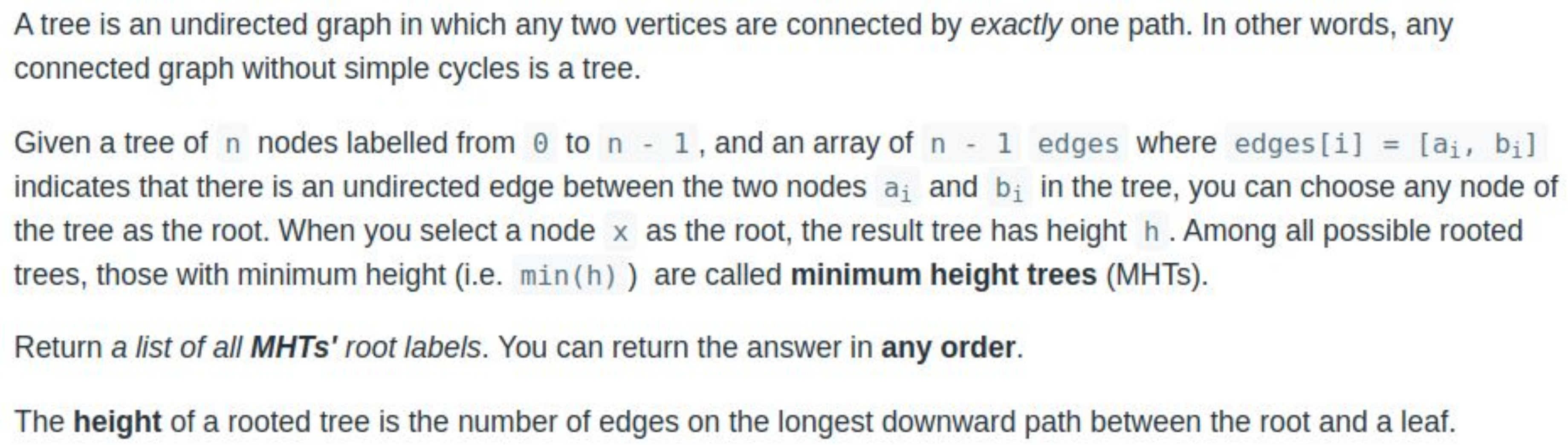
Example 3:

```
Input: org = [1,2,3], seqs = [[1,2],[1,3],[2,3]]
Output: true
Explanation: The sequences [1,2], [1,3], and [2,3] can uniquely
reconstruct the original sequence [1,2,3].
```

Example 4:

```
Input: org = [4,1,5,2,6,3], seqs = [[5,2,6,3],[4,1,5,2]]
Output: true
```

Problem #2: LeetCode 310 - Minimum Height Trees



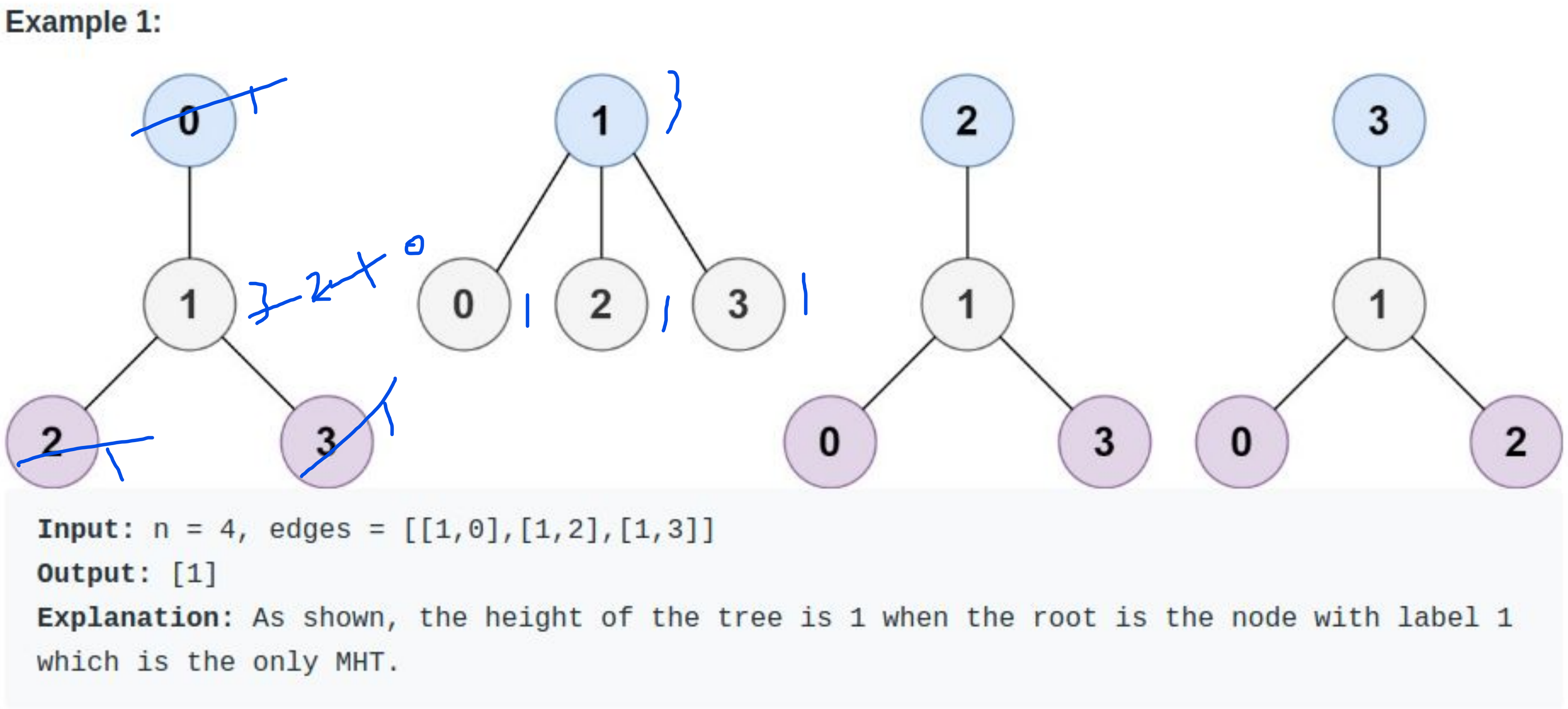
A tree is an undirected graph in which any two vertices are connected by *exactly* one path. In other words, any connected graph without simple cycles is a tree.

Given a tree of `n` nodes labelled from `0` to `n - 1`, and an array of `n - 1` `edges` where edges[i] = [a_i, b_i] indicates that there is an undirected edge between the two nodes a_i and b_i in the tree, you can choose any node of the tree as the root. When you select a node `x` as the root, the result tree has height `h`. Among all possible rooted trees, those with minimum height (i.e. `min(h)`) are called **minimum height trees** (MHTs).

Return *a list of all **MHTs'** root labels*. You can return the answer in **any order**.

The **height** of a rooted tree is the number of edges on the longest downward path between the root and a leaf.

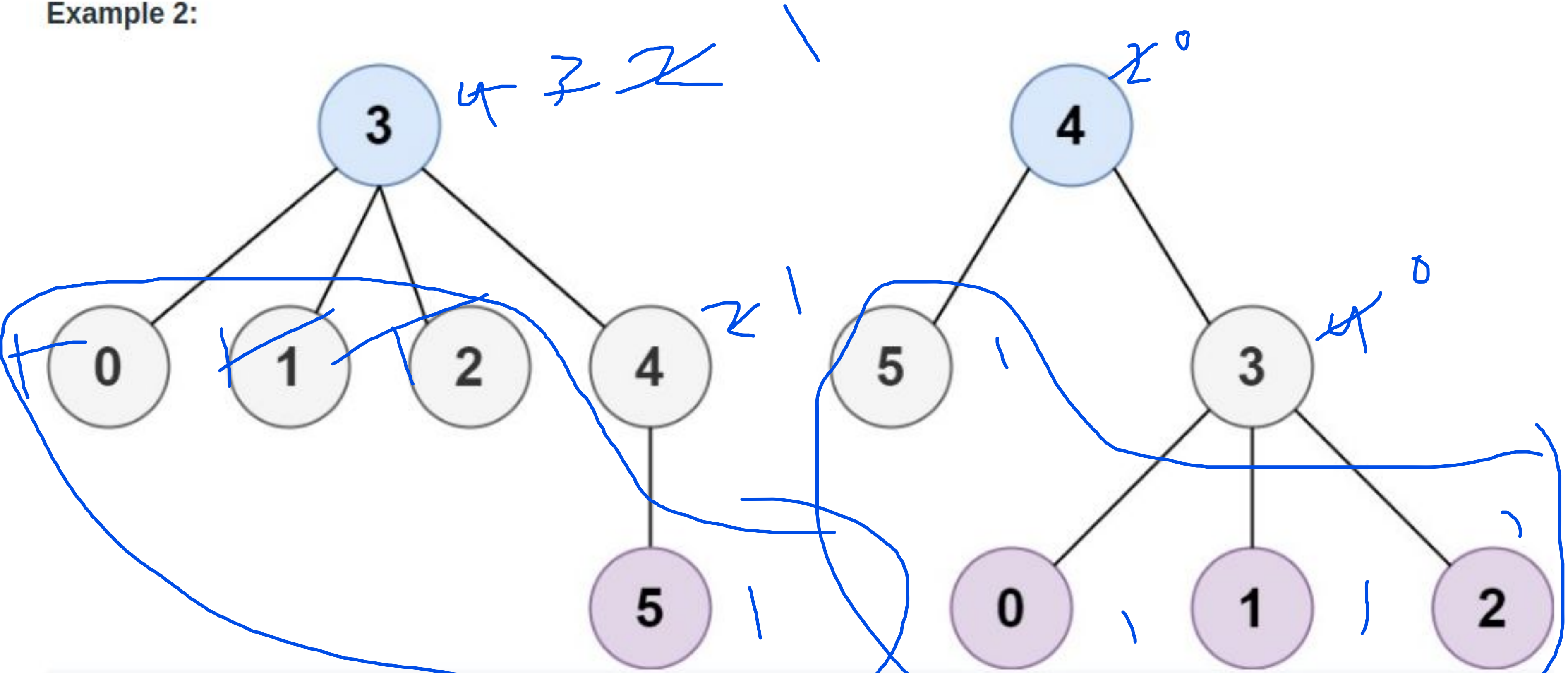
- C++: vector<int> findMinHeightTrees(int n, vector<vector<int>> &edges)
- Java: public List<Integer> findMinHeightTrees(int n, int[][] edges)
- Python: def findMinHeightTrees(self, n: int, edges: List[List[int]]) -> List[int]:
- Javascript: var findMinHeightTrees = function(n, edges)



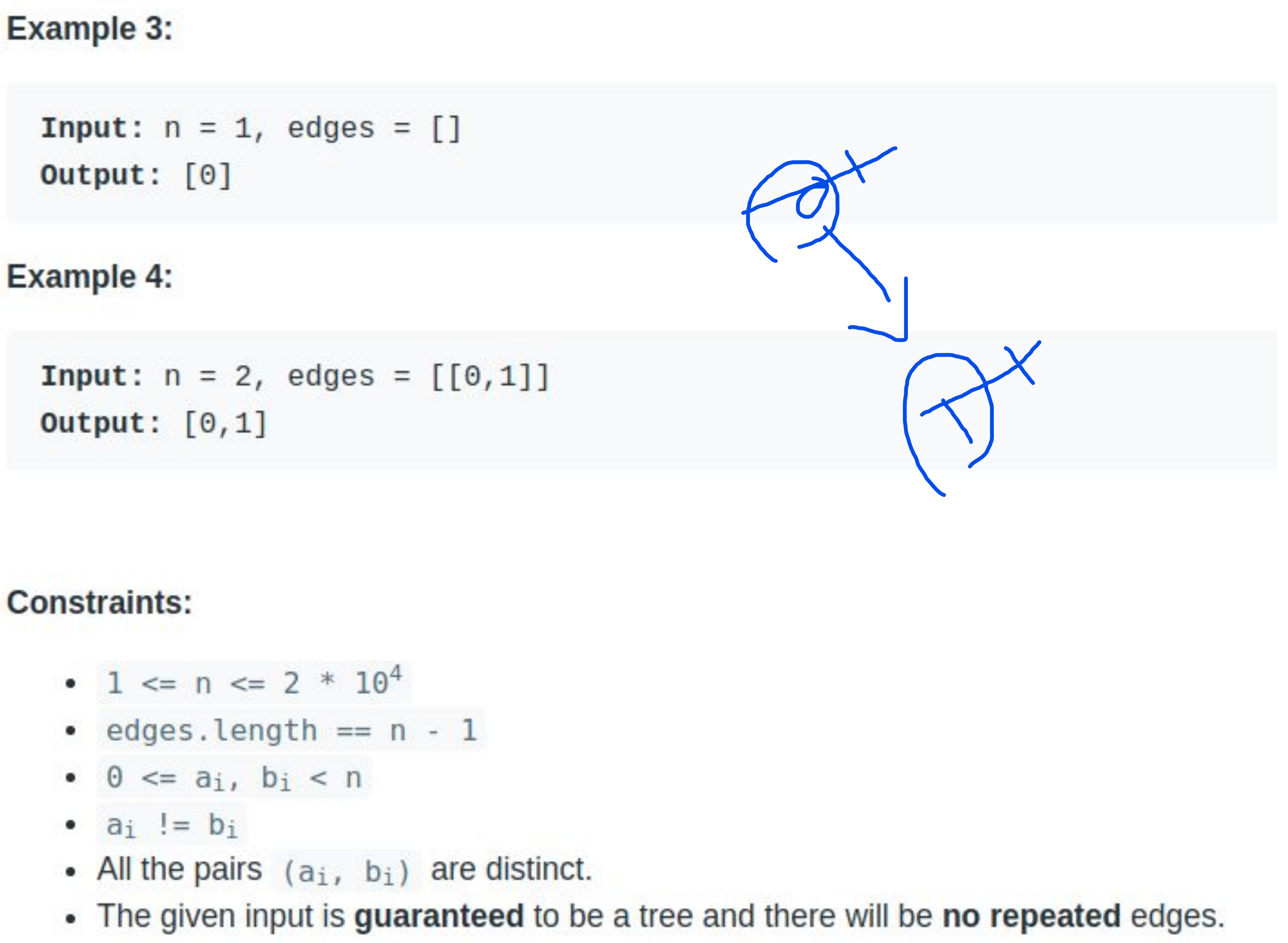
Example 1:

```
Input: n = 4, edges = [[1,0],[1,2],[1,3]]
Output: [1]
Explanation: As shown, the height of the tree is 1 when the root is the node with label 1
which is the only MHT.
```

Example 2:



```
Input: n = 6, edges = [[3,0],[3,1],[3,2],[3,4],[5,4]]
Output: [3,4]
```



Example 3:

```
Input: n = 1, edges = []
Output: [0]
```

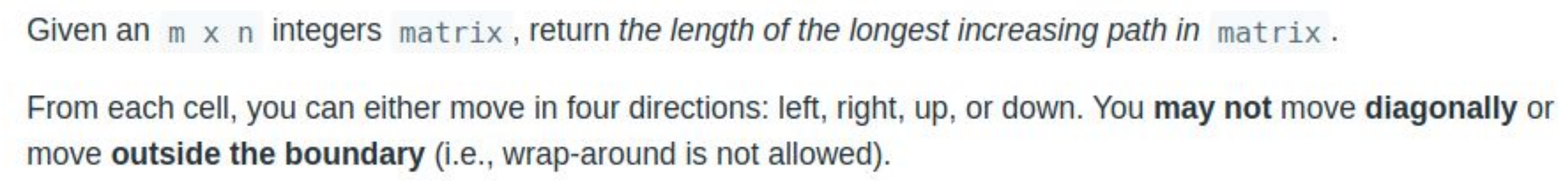
Example 4:

```
Input: n = 2, edges = [[0,1]]
Output: [0,1]
```

Constraints:

- `1 <= n <= 2 * 10`^{4}
- `edges.length == n - 1`
- `0 <=` a_i, b_i `< n`
- a_i `!=` b_i
- All the pairs (a_i, b_i) are distinct.
- The given input is **guaranteed** to be a tree and there will be **no repeated** edges.

Problem #3: LeetCode 329 - Longest Increasing Path in a Matrix

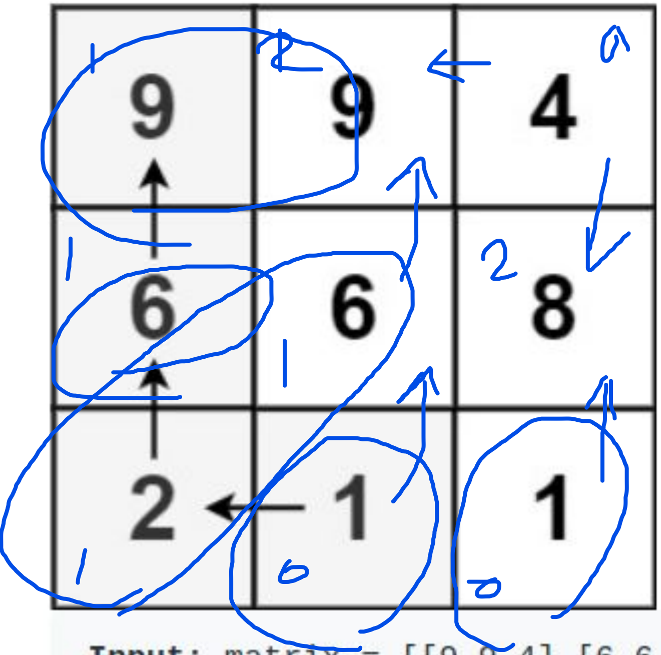


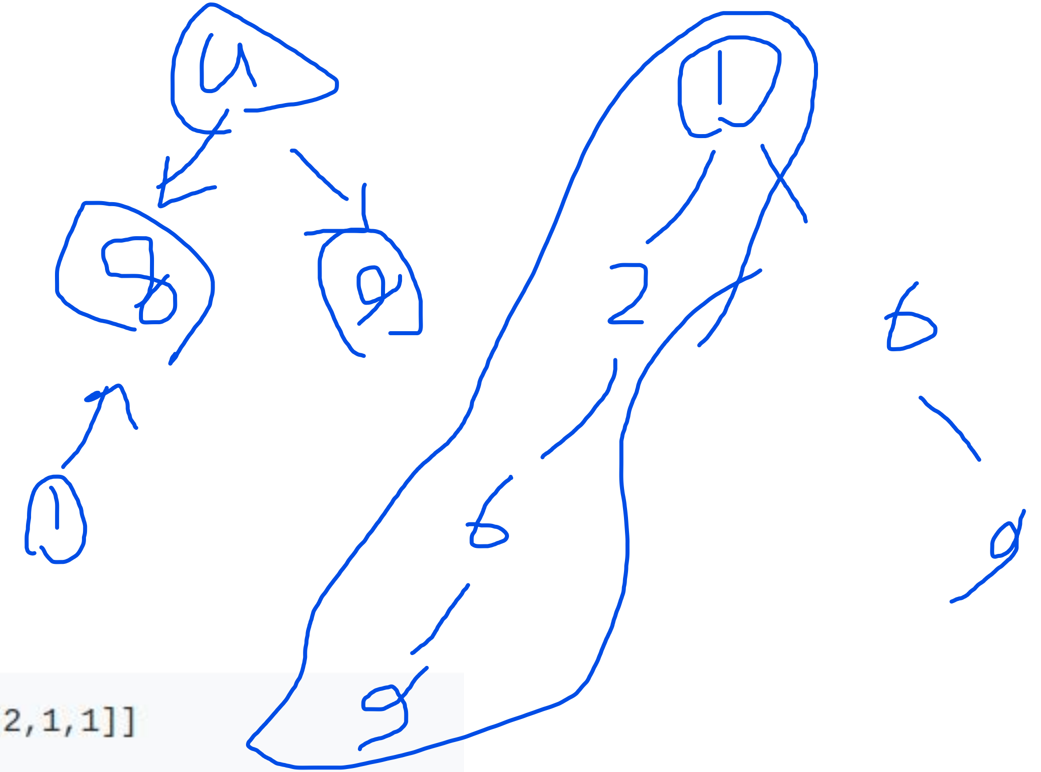
Given an `m x n` integers `matrix`, return *the length of the longest increasing path in* `matrix`.

From each cell, you can either move in four directions: left, right, up, or down. You **may not** move **diagonally** or move **outside the boundary** (i.e., wrap-around is not allowed).

- C++: int longestIncreasingPath(vector<vector<int>>& matrix)
- Java: public int longestIncreasingPath(int[][] matrix)
- Python: def longestIncreasingPath(self, matrix: List[List[int]]) -> int
- Javascript: var longestIncreasingPath = function(matrix)
- 1 <= m, n <= 200
- 0 <= matrix[i][j] <= 2^{31} - 1

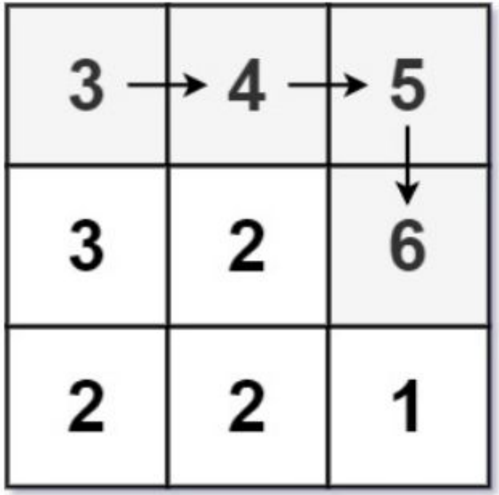
Example 1:





```
Input: matrix = [[9,9,4],[6,6,8],[2,1,1]]
Output: 4
Explanation: The longest increasing path is [1, 2, 6, 9].
```

Example 2:



```
Input: matrix = [[3,4,5],[3,2,6],[2,2,1]]
Output: 4
Explanation: The longest increasing path is [3, 4, 5, 6]. Moving diagonally is not allowed.
```

Example 3:

```
Input: matrix = [[1]]
Output: 1
```

“Acquire knowledge and impart it to the people.”

“Seek knowledge from the Cradle to the Grave.”