Homework: Tree and Graph Traversal

This document defines the **homework assignments** for the "Data Structures" course @ Software University. Please submit a single **zip** / **rar** / **7z** archive holding the solutions (source code) of all below described problems.

Problem 1. Find the Root

You have a connected directed graph with **N nodes** and **M edges**. You are given the directed edges from one node to another. The nodes are numbered from **0** to **N-1** inclusive. Check **whether the graph is a tree** and **find its root**.

Input:

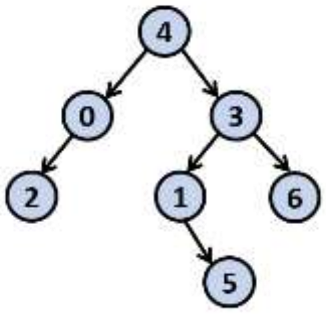
- Read the input data from the console.
- The first line holds the **number of nodes N**.
- The first line holds the **number of edges M**.
- Each of the following **M lines** hold the **edges**: two numbers separated by a space. The first number is the **parent node**; the second number is the **child node**.

Output:

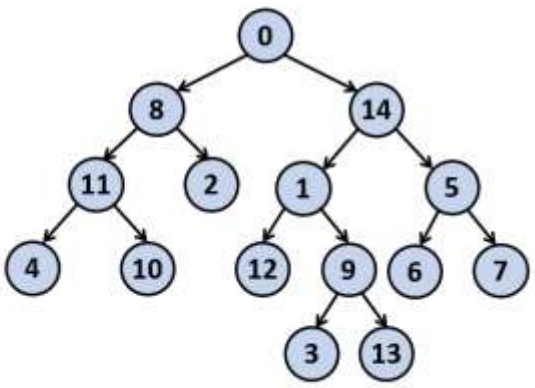
- Print at the console the **number of the root node** in case the graph is a **tree**.
- If there is no root, print "**No root!**" at the console.
- If multiple root nodes exist, print "**Multiple root nodes!**" at the console.

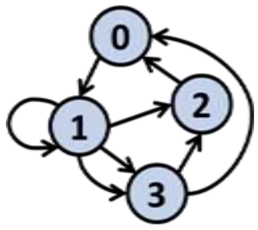
Examples:

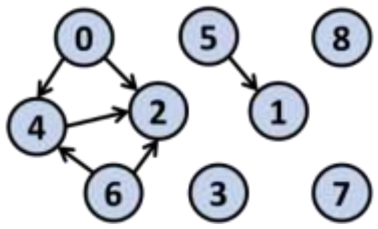
Input	Output	Tree

7
6
4 0
3 1
4 3
3 6
0 2
1 5
	4	The graph is a **tree** holding **7 nodes** (0…6) and **6 edges**. The **root** node is **4**.
		

15
14
11 4
11 10
8 11
8 2
9 3
	0	The graph is a **tree** holding **15 nodes** (0…14) and **14 edges**. The **root** node is **0**.

`9 13`
`1 9`
`14 1`
`0 8`
`0 14`
`5 6`
`5 7`
`14 5`
`1 12`
		

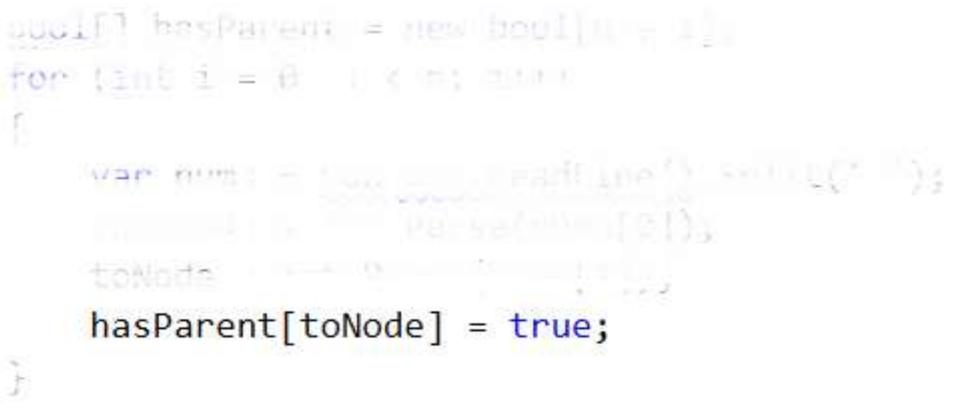
`4`
`8`
`0 1`
`1 1`
`1 2`
`1 3`
`1 3`
`2 0`
`3 0`
`3 2`
	`No root!`	The graph is **not a tree** (all nodes have parents). The graph has **4 nodes** (0…3) and **8 edges**. The **root** node is **0**.
		

`9`
`6`
`6 2`
`0 2`
`5 1`
`4 2`
`0 4`
`6 4`
	`Multiple root nodes!`	The graph is **not a tree** (it is a forest). The graph has **9 nodes** (0…8) and **6 edges**. There are several trees in the forest and their roots are: **0**, **3**, **5**, **6**, **7** and **8**.
		

Hints (Click on the arrow to show)

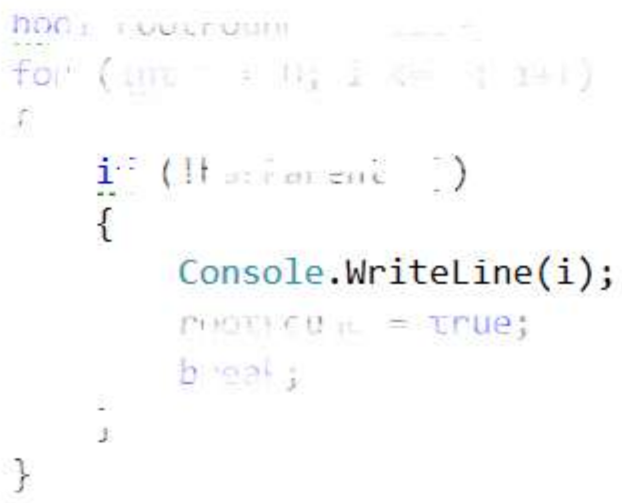
You need information for each node: whether it has a parent or not. Find the node without parent → it is the root.

1. Read the tree data from the console.
2. Collect **`hasParent`** information for each node.



```
bool[] hasParent = new bool[[illegible]];
for (int i = 0; i < [illegible])
{
    var nums = [illegible].ReadLine().Split(' ');
    [illegible] = [illegible].Parse(nums[2]);
    toNode [illegible]
    hasParent[toNode] = true;
}
```

3. Pass through each of the nodes and **find the node without parent**. It is the root.



```
bool rootFound [illegible]
for (int [illegible]; i <= [illegible]; i++)
{
    if (!hasParent[illegible])
    {
        Console.WriteLine(i);
        rootFound = true;
        break;
    }
}
```

4. If all nodes have parent, print “**No root!**” at the console.
5. If multiple nodes has no parent, print "**Multiple root nodes!**" at the console.

Problem 2. Round Dance

Build a round dance, where everybody is holding a friend. You are given the number the person who should lead the dance. Every person has a unique number. Every friendship is two-way. If **Stamat** is a friend of **Minka**, **Minka** is a friend of **Stamat**. There are **no circular friendships** (e.g. A is friend to B, C is friend to B and A is friend to C).

Your task is to **find the longest round dance**, such that it starts with K-person and every person in it is holding his/her friend.

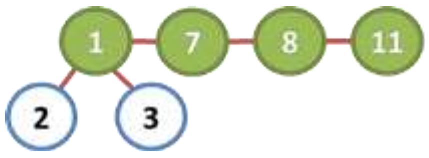
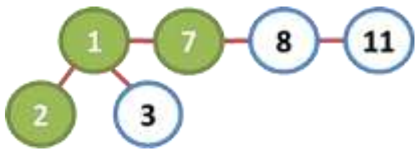
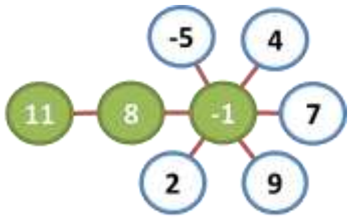
Input:

- At the first line the number **F** is given – the number of friendships.
- At the second line the number **K** is given – the man, who leads the dance.
- Each of the next **F** lines holds a friendship between two people, separated by a space.

Output:

- Print at the console the number of people in the longest dance.

Examples:

Input	Output	Explanation	Round Dance
5 1 11 8 2 1 8 7 1 3 7 1	4	The round dances starting from 1 are: • 1 → 2 • 1 → 3 • 1 → 7 → 8 → 11 The round dance with the most people, starting from 1 is **1 → 7 → 8 → 11** (**4** people).	
5 7 7 8 1 2 7 1 8 11 3 1	3	All round dances, starting from 7, consists of exactly 3 people: • 7 → 1 → 2 • 7 → 1 → 3 • 7 → 8 → 11 The longest round dance has **3** people.	
7 -1 -5 -1 11 8 -1 -7 9 -1 2 -1 -1 4 8 -1	3	The round dances, starting from -1 are: • -1 → -5 • -1 → 4 • -1 → 7 • -1 → 9 • -1 → 2 • -1 → 8 → 11 The longest round dance has **3** people.	

Hints (Click on the arrow to show)

- Use a dictionary (hash-table) to keep the list of child nodes for each node, e.g. **`Dictionary<int, List<int>>`**.
- Use **DFS starting from the leader** (node K). You need to find the longest path starting from a certain node.

Problem 3. Ride the Horse

You are given a matrix **N x M** and a start position. Your task is to traverse the matrix using the movements of the horse from the chess game and marking where you have gone.

1. Visit the start position and assign 1 in it.
2. If a position holds the value V, assign V+1 in all not-visited cells which can be reached by movement of the horse from this position.

3. Repeat the previous step until all positions are visited.

	V+1		V+1	
V+1				V+1
		V		
V+1				V+1
	V+1		V+1	

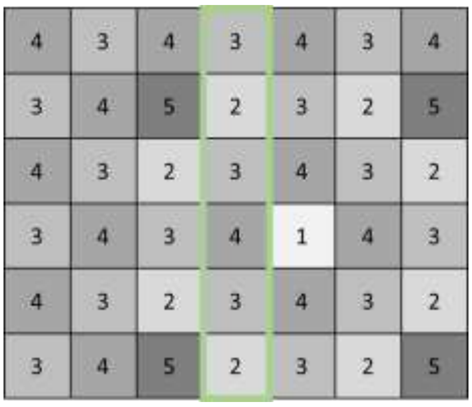
Input:

- The first line holds the number **N** – the number of rows in the matrix.
- The second line holds the number **M** – the number of the columns in the matrix.
- The third line holds the number **R** – the row of the start position of the horse.
- The fourth line holds the number **C** – the column of the start position of the horse.
- The cell at the top left corner is (0, 0) and the cell in the bottom right corner is (N-1, M-1).

Output:

- Print at the console the cells from column **M / 2** (integer division) in the matrix. Each cell should stay on separate line. If the horse has not visited some cell, it should hold 0.

Example:

Input	Output	Explanation
6 7 3 4	3 2 3 4 3 2	The matrix is as follows: 

4	3	4	3	4	3	4
3	4	5	2	3	2	5
4	3	2	3	4	3	2
3	4	3	4	1	4	3
4	3	2	3	4	3	2
3	4	5	2	3	2	5

Hints (Click on the arrow to show)

Use **BFS**. Initially put in the queue the starting cell. At each step take a cell from the queue and enqueue all unvisited neighbour cells. Stop when the queue becomes empty.

Problem 4. Longest Path in a Tree

You are given a tree holding **N** nodes. Each node holds a unique number. Each node can have child nodes. Find **the longest path by sum of the nodes from leaf to leaf**.

Input:

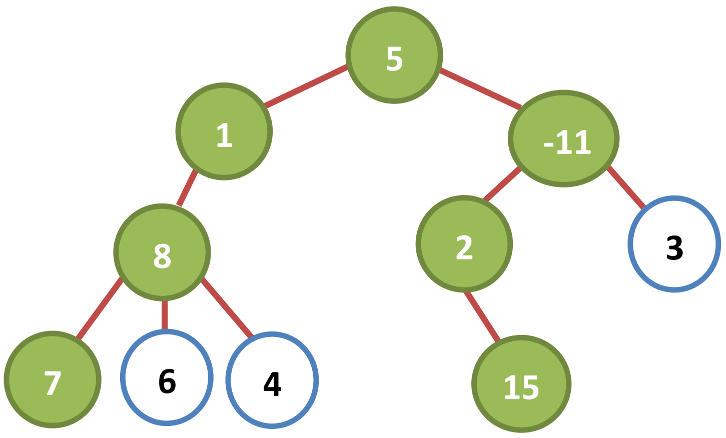
- Read the input from the console.
- The first line holds the number **N** – the count of the nodes of the tree.
- The second line holds the number **M** – the count of the edges between nodes in the tree.
- Each of the next **M** lines holds an edge is in format **"p_1 p_2"**.

Output:

- Print at the console the sum of the nodes of the largest path from leaf to leaf.

Example:

Input	Output	Explanation	Tree

`10`
`9`
`5 -11`
`1 8`
`-11 3`
`8 7`
`5 1`
`-11 2`
`8 6`
`2 15`
`8 4`
	`27`	The max sum path is:
		7 -> 8 -> 1 -> 5 -> -11 -> 2 -> 15
		
		`sum = 7+8+1+5-11+2+15`
		`= 27`
			

Hints (Click on the arrow to show)

1. Use a tree that holds for each node its **parent** and its **list of children**. Hold the nodes in a dictionary, e.g. **`Dictionary<int, List<int>>`**. Hold the parents in a dictionary, e.g. **`Dictionary<int, int?>`**.
2. Find the sum of the path from each node to the root. You may use recursive DFS traversal from the root.
3. For each couple of different nodes, **sum their paths to the nearest common parent**. You may use the following optimization: If B is parent of A, then sum(A … B) = sum(A … root) - sum (B … root) + weight(B).
4. Find the longest path (the greatest sum between two nodes).
5. If you implement efficiently the above algorithm, its complexity will be O(N^2) where N is the number of nodes.

Problem 5. * Sorting

You are given a list with numbers, holding a random permutation of the integers from **1** to **N**, inclusive.

At each step, it is allowed to make only one operation of the following type: take **K** consecutive elements from the list and reverse their order (e.g. if we have the numbers 1, 8, 3, 6, after their reversal they became in order 6, 3, 8, 1).

The list should be sorted in increasing order (1, 2, 3, …, **N**).

By given N, K and a list of numbers, find the smallest number of steps (operations) needed to sort the numbers in ascending order. If this is impossible, print **-1**.

Input:

- The first line holds the number **N** – count of the numbers.
- The second line holds the shuffled numbers (separated one from another by a space).
- The third line holds the number **K** – number of consecutive elements.

Output

- Print at the console the smallest number of operations, needed to sort the numbers in increasing order.

Examples

Input	Output	Explanation

3
1 2 3
3
	0	The list is already sorted. It is not necessary to perform any operations.

3
3 2 1
3
	1	Just reverse the whole list.

5
5 4 3 2 1
2
	10	4 operations to push 1 at left
		1 5 4 3 2
		3 operations to push 2 on second position
		1 **2** 5 4 3
		2 operations to push 3 on third position
		1 2 **3** 5 4
		1 operation to exchange 5 and 4

5
3 2 4 1 5
4
	-1	It is impossible to sort the list by reversing of 4 elements.

7
7 2 1 6 8 4 3
4
	7	The list can be sorted of at least 8 operations of reversing the order of 4 numbers at once.

Hints (Click on the arrow to show)

- Use **BFS**. Start from the initial sequence of numbers. Then put in the queue all sequences reachable by a single reverse operation. Then put in the queue all sequences reachable by two reverse operations, etc.
- Don't repeat sequences (obtain only non-repeating, unique sequences). Think how to check efficiently whether a sequence is already obtained at some previous step.
- Stop when no unique sequence can be obtained from the others in the queue.
- Note: this algorithm will not work for very big sequences of numbers due to exponential memory needed.