Lab #9: MiniMax and Alpha Beta Pruning algorithms (cont.)

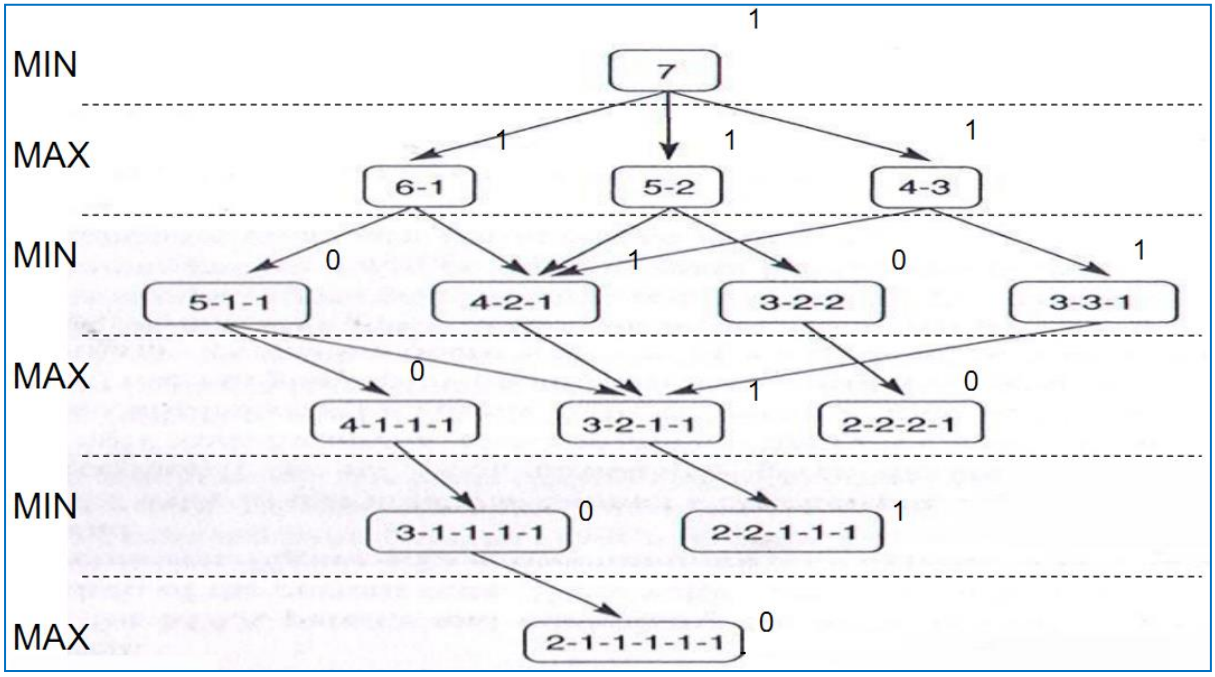
The main aim of this lab is to apply Minimax algorithm to **NIM game**.

Deadline: 23h59, 04/12/2023

The Game of Nim: A number of tokens are placed on a table between two opponents. At each move, the player must divide a pile of tokens into two nonempty piles of different sizes (unequal size). Thus, 6 tokens may be divided into 5 and 1, 4 and 2, but ***not 3 and 3***. The first player who is unable to make a move loses the game.

For a small number of tokens, the search space can be searched exhaustively. The next figure gives the complete space for a 7-token game. In this case, for each move, the player can divide the pile of tokens into smaller piles containing 1, 2, or 3 (if possible) tokens. Particularly, 7 tokens can be divided into 2 piles having unequal size such as (6, 1), (5, 2), and (4, 3).

Notice that, a utility function of 1 is assigned to a state if MAX wins the game, 0 if MIN wins.



The Node structure is defined as follows:

```
public class Node {
	private List<Integer> data = new ArrayList<Integer>();
	private List<Node> children = new ArrayList<Node>();
```

```
	public void add(Integer val) {
		this.data.add(val);
	}

	public void addAll(List<Integer> data) {
		this.data.addAll(data);
	}

	//Get children of the current nodes
	public List<Node> getSuccessors() {
		//Enter your code here
		return null;
	}
	//Check whether a node is terminal or not
	public boolean isTerminal() {
		//Enter your code here
		return false;
	}

	public static final Comparator<Integer> DESCOMPARATOR = new
Comparator<Integer>() {
		@Override
		public int compare(Integer o1, Integer o2) {
			return o2.compareTo(o1);
		}

	};

	@Override
	public String toString() {
		Collections.sort(this.data, DESCOMPARATOR);
		return this.data.toString();
	}
}
```

Task 1. Implement ***isTerminal()*** to check whether a node is terminal or not. A node is called a terminal if the tokens of each pile cannot be divided.

Task 2. Implement ***getSuccessors()*** to generate the children (successors) of the current node.

Notice that, with this method, each node cannot have 2 identical successors.

Task 3. Implement Minimax algorithm to the Nim game to determine the value at the root node (using the given pseudo-code in **MinimaxAlgo.java** class).

Task 4. Modify the implemented code to show the best move for MIN player at the root of the game tree.

Task 5 (Advanced). Test the implemented algorithm with other numbers of tokens such as 8, 9, ...