Lab #8: Minimax and Alpha Beta Pruning algorithms

The main aim of this lab is to deal with the implementations of Minimax and Alpha-beta pruning algorithms.

The structure of a node in the tree is defined as follows:

```
public class Node {
	private String label;
	private int value;
	private List<Node> children = new ArrayList<Node>();

	// use for non-terminal node
	public Node(String label) {
		super();
		this.label = label;
	}

	// use for terminal node
	public Node(String label, int value) {
		super();
		this.label = label;
		this.value = value;
	}

	//...
	//add a child to this node
	public void addChild(Node that) {
		this.children.add(that);
	}

	// check whether this node is terminal or not. The terminal node is
assigned a
	// value.
	public boolean isTerminal() {
		return this.children.size() == 0;
	}

	// Defined comparator which is used for sorting children by
alphabetical order
	public static Comparator<Node> LabelComparator = new
Comparator<Node>() {
		@Override
		public int compare(Node o1, Node o2) {
			return o1.getLabel().compareTo(o2.getLabel());
		}
	};
}
```

ISearchAlgo.java:

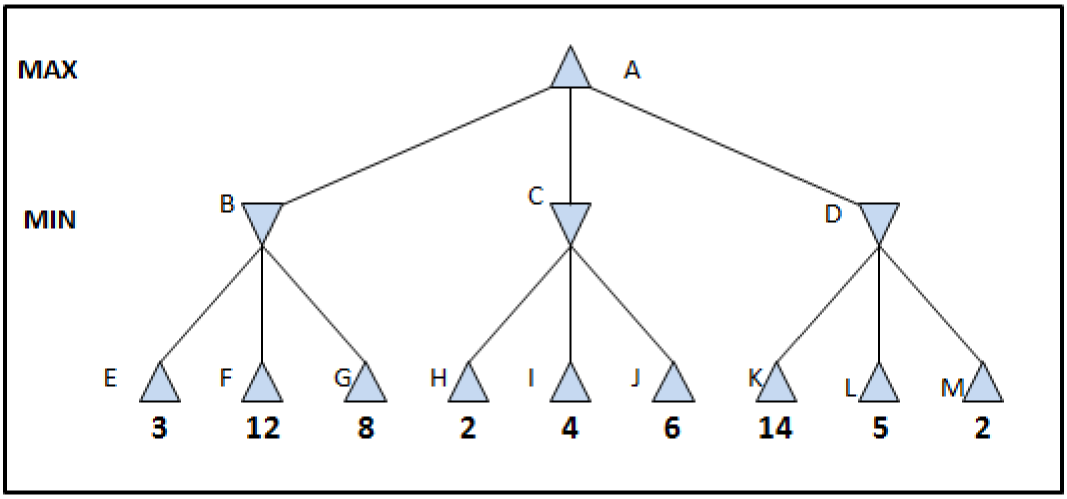
```
public interface ISearchAlgo {
	public void execute(Node node);
}
```

Task 1. Implement **MiniMaxSearchAlgo** with the given pseudo code in the comments.

Note, use the defined comparator in the Node class to sort children before invoking recursively for each child node to ensure the traversal according to the alphabetical order.

```
public class MiniMaxSearchAlgo implements ISearchAlgo {

	@Override
	public void execute(Node node) {
		// Enter your code here
	}
	public int maxValue(Node node) {
		// Enter your code here
		return Integer.MIN_VALUE;
	}
	public int minValue(Node node) {
		// Enter your code here
		return Integer.MAX_VALUE;
	}
}
```

The result using the MiniMaxSearchAlgo for the following tree is **3** at node A. Then modify the code so that we can get values at nodes B, C, and D (after invoking execute method). In this case, the values at nodes B, C, and D are 3, 2, and 2, respectively.



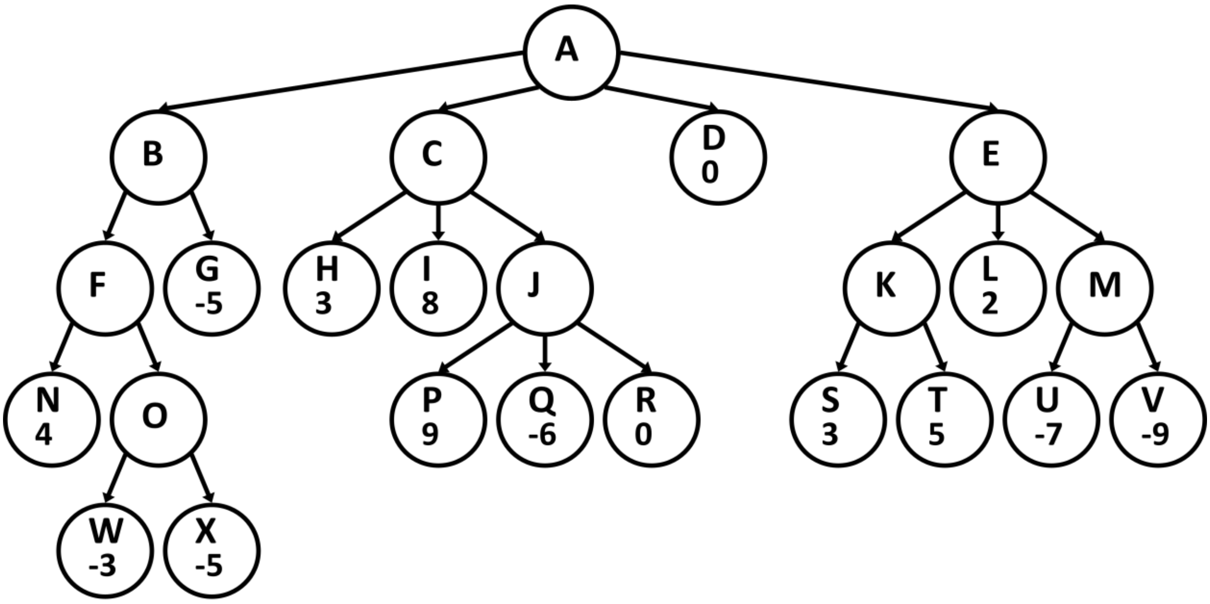
Task 2. Implement **AlphaBetaSearchAlgo** with the given pseudo code in the comments

```
public class AlphaBetaSearchAlgo implements ISearchAlgo {

	@Override
	public void execute(Node node) {
		// Enter your code here
	}


	public int maxValue(Node node, int alpha, int beta) {
		// Enter your code here
```

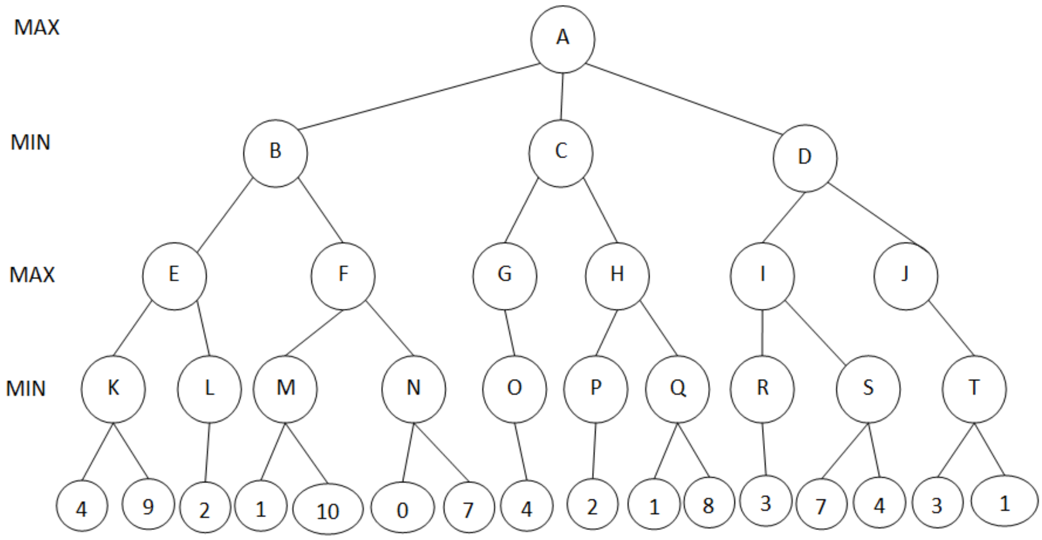
```
            return Integer.MIN_VALUE;
        }

        public int minValue(Node node, int alpha, int beta) {
            // Enter your code here
            return Integer.MAX_VALUE;
        }
}
```

The result using the AlphaBetaSearchAlgo for the following tree is **3** at node A. Similar to Task 1, modify the code so that we can get values at non-terminal nodes.

Then, continue modifying the code to show **the nodes that pruned**. With this tree, the pruned nodes will be **X, Q, R, M, U,** and **V**.



Additional task: test the implemented algorithms with the following tree (remember assigning labels for the nodes).



Task 3. Modify the implemented algorithm in Task 2, named **AlphaBetaRightToLeftSearchAlgo** to deal with the alpha-beta pruning algorithm so that the expanding order is right to left mode. Test with the previous game tree to check the correctness of the implemented algorithm.

Task 4. Modify the implemented algorithms so that we can export the best move for a player at any node in the game tree. For example: in Task 2, the best move for MAX at A will be **B**.