**Exercise: Red-Black Trees and AA-Trees**

This document defines the lab for ["Data Structures – Advanced (C#)" course @ Software University](https://softuni.bg/trainings/3113/data-structures-advanced-with-csharp-october-2020).

Please submit your solutions (**source code**) of all below described problems in [Judge](https://judge.softuni.bg/Contests/2604/04-Red-Black-Trees-and-AA-Trees-Exercise).

## Red-Black Tree

You are given class **RedBlackTree** your task is to implement all the methods with missing implementation:

* void **Insert**(Key key) – adds **new** **entry** to the tree if the key
* bool **Contains**(Key key) – returns **whether** the key is present
* int **Count** – returns the **size** of the tree
* **IBinarySearchTree<T> Search(T element)** - search for elements
* void **DeleteMin**() – removes the **min** element by key in the tree
* void **DeleteMax**() – removes the **max** element by key in the tree
* void **Delete**(Key key) – removes the wanted key
* Key **Floor**(Key key) - If a given key is less than the key at the root of a BST, then the floor of key (the largest key in the BST less than or equal to key) must be in the left subtree If key is greater than the key at the root, then the floor of key could be in the right subtree, but only if there is a key smaller than or equal to key in the right subtree if not (or if key is equal to the key at the root) then the key at the root is the floor of key.
* Key **Ceiling**(Key key) – finding the ceiling is similar to floor, interchanging right and left
* Key **Select** (int rank) – suppose that we seek the key of rank k (the key such that precisely k other keys in the BST are smaller). If the number of keys t in the left subtree is larger than k, we look (recursively) for the key of rank k in the left subtree; if t is equal to k, we return the key at the root; and if t is smaller than k, we look (recursively) for the key of rank k - t - 1 in the right subtree.
* int **Rank**(Key key) – if the given key is equal to the key at the root, we return the number of keys t in the left subtree; if the given key is less than the key at the root, we return the rank of the key in the left subtree; and if the given key is larger than the key at the root, we return t plus one (to count the key at the root) plus the rank of the key in the right subtree.
* IEnumerable<Key> **Range**(Key lo, Key hi) – returns the keys from **lower** bound to upper **bound** inside the tree

**Hints**: There are some private methods which should help try to implement them, however the tests won't require any of them.

## AA Tree

You are given class **AATree** your task is to implement all the methods with missing implementation, it should be pretty simple:

* bool **IsEmpty**() – returns **whether** the tree is empty or not
* void **Clear**() – **clears** the tree and makes it empty
* void **Insert**(T element) – **adds** an element
* int **CountNodes** – returns the **number** of nodes
* bool **Search**(T element) – returns **whether** the element is present or not
* void **InOrder**(Action<T> action) – the **action** **accepts** all the elements **in order** traversal
* void **PreOrder**(Action<T> action) – the **action** **accepts** all the elements **pre order** traversal
* void **PostOrder**(Action<T> action) – the **action** **accepts** all the elements **post order** traversal