```
LinksPlatform's Platform Collections Methods Class Library
./GenericCollectionMethodsBase.cs
   using System;
   using System.Collections.Generic;
using System.Runtime.CompilerServices;
2
   using Platform. Numbers;
4
   #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
   namespace Platform.Collections.Methods
       public unsafe abstract class GenericCollectionMethodsBase<TElement>
10
            private static readonly EqualityComparer<TElement> _equalityComparer =
12
               EqualityComparer<TElement>.Default;
            private static readonly Comparer<TElement> _comparer = Comparer<TElement>.Default;
13
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
15
            protected virtual TElement GetZero() => Integer<TElement>.Zero;
16
17
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
18
            protected virtual TElement GetOne() => Integer<TElement>.One;
19
20
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
21
            protected virtual TElement GetTwo() => Integer<TElement>.Two;
23
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
24
            protected virtual bool ValueEqualToZero(IntPtr pointer) => _equalityComparer.Equals(Syst | 
25
            em.Runtime.CompilerServices.Unsafe.Read<TElement>((void*)pointer),

   GetZero());
26
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            protected virtual bool EqualToZero(TElement value) => _equalityComparer.Equals(value,
28

   GetZero());
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            protected virtual bool IsEquals(TElement first, TElement second) =>
31
               _equalityComparer.Equals(first, second);
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
33
            protected virtual bool GreaterThanZero(TElement value) => _comparer.Compare(value,
34
            \rightarrow GetZero()) > 0;
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
36
            protected virtual bool GreaterThan(TElement first, TElement second) =>
37
                _comparer.Compare(first, second) > 0;
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
39
            protected virtual bool GreaterOrEqualThanZero(TElement value) =>
                _comparer.Compare(value, GetZero()) >= 0;
41
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
42
            protected virtual bool GreaterOrEqualThan(TElement first, TElement second) =>
            → _comparer.Compare(first, second) >= 0;
44
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
45
            protected virtual bool LessOrEqualThanZero(TElement value) => _comparer.Compare(value,
            \rightarrow GetZero()) <= 0;
47
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            protected virtual bool LessOrEqualThan(TElement first, TElement second) =>
49
                _comparer.Compare(first, second) <= 0;</pre>
50
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            protected virtual bool LessThanZero(TElement value) => _comparer.Compare(value,
52
            \rightarrow GetZero()) < 0;
53
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            protected virtual bool LessThan(TElement first, TElement second) =>
5.5
               _comparer.Compare(first, second) < 0;
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
57
            protected virtual TElement Increment(TElement value) =>
58
               Arithmetic<TElement>.Increment(value);
59
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
60
            protected virtual TElement Decrement(TElement value) =>
61
               Arithmetic<TElement>.Decrement(value);
```

```
62
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            protected virtual TElement Add(TElement first, TElement second) =>
64
               Arithmetic<TElement>.Add(first, second);
65
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
66
            protected virtual TElement Subtract(TElement first, TElement second) =>
67
               Arithmetic<TElement>.Subtract(first, second);
68
   }
69
./Lists/CircularDoublyLinkedListMethods.cs
   #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
   namespace Platform.Collections.Methods.Lists
3
       public abstract class CircularDoublyLinkedListMethods<TElement> :
5
           DoublyLinkedListMethodsBase<TElement>
            public void AttachBefore(TElement baseElement, TElement newElement)
                var baseElementPrevious = GetPrevious(baseElement);
10
                SetPrevious(newElement, baseElementPrevious);
                SetNext(newElement, baseElement);
11
                if (IsEquals(baseElement, GetFirst()))
12
13
                    SetFirst(newElement);
14
15
                SetNext(baseElementPrevious, newElement);
                SetPrevious(baseElement, newElement);
17
                IncrementSize();
18
            }
19
20
            public void AttachAfter(TElement baseElement, TElement newElement)
21
                var baseElementNext = GetNext(baseElement);
23
                SetPrevious(newElement, baseElement);
24
                SetNext(newElement, baseElementNext);
25
                if (IsEquals(baseElement, GetLast()))
26
                {
27
                    SetLast(newElement);
28
                SetPrevious(baseElementNext, newElement);
30
                SetNext(baseElement, newElement);
31
32
                IncrementSize();
            }
33
34
            public void AttachAsFirst(TElement element)
36
                var first = GetFirst();
37
                if (EqualToZero(first))
38
39
                    SetFirst(element);
40
                    SetLast(element);
41
                    SetPrevious(element, element);
                    SetNext(element, element);
43
                    IncrementSize();
44
                }
45
                else
46
                {
47
                    AttachBefore(first, element);
49
            }
50
51
            public void AttachAsLast(TElement element)
52
5.3
                var last = GetLast();
54
                if (EqualToZero(last))
55
                {
56
                    AttachAsFirst(element);
                }
58
                else
                {
60
                     AttachAfter(last, element);
61
                }
62
            }
64
            public void Detach(TElement element)
```

```
66
                            var elementPrevious = GetPrevious(element);
                            var elementNext = GetNext(element)
                            if (IsEquals(elementNext, element))
69
                                   SetFirst(GetZero());
7.1
                                   SetLast(GetZero());
72
73
                            else
74
                            {
75
                                   SetNext(elementPrevious, elementNext);
                                   SetPrevious(elementNext, elementPrevious);
                                   if (IsEquals(element, GetFirst()))
78
79
80
                                           SetFirst(elementNext);
81
                                        (IsEquals(element, GetLast()))
82
                                           SetLast(elementPrevious);
85
86
                            SetPrevious(element, GetZero());
                            SetNext(element, GetZero());
88
                            DecrementSize();
89
                     }
             }
91
92
./Lists/DoublyLinkedListMethodsBase.cs
      using System.Runtime.CompilerServices;
      #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
 3
      namespace Platform.Collections.Methods.Lists
 5
 6
              /// <remarks>
             /// Based on <a href="https://en.wikipedia.org/wiki/Doubly_linked_list">doubly_linked_list">doubly_linked_list">doubly_linked_list">doubly_linked_list">doubly_linked_list">doubly_linked_list">doubly_linked_list">doubly_linked_list">doubly_linked_list">doubly_linked_list">doubly_linked_list">doubly_linked_list">doubly_linked_list">doubly_linked_list">doubly_linked_list">doubly_linked_list">doubly_linked_list">doubly_linked_list">doubly_linked_list">doubly_linked_list">doubly_linked_list">doubly_linked_list">doubly_linked_list">doubly_linked_list">doubly_linked_list">doubly_linked_list">doubly_linked_list">doubly_linked_list">doubly_linked_list">doubly_linked_list">doubly_linked_list">doubly_linked_list">doubly_linked_list">doubly_linked_list">doubly_linked_list">doubly_linked_list">doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</d>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked_list</dd>doubly_linked
                    list</a> implementation.
              /// </remarks>
             public abstract class DoublyLinkedListMethodsBase<TElement> :
10
                    GenericCollectionMethodsBase<TElement>
11
12
                     [MethodImpl(MethodImplOptions.AggressiveInlining)]
                     protected abstract TElement GetFirst();
13
                     [MethodImpl(MethodImplOptions.AggressiveInlining)]
14
                    protected abstract TElement GetLast();
                     [MethodImpl(MethodImplOptions.AggressiveInlining)]
                     protected abstract TElement GetPrevious(TElement element);
17
                     [MethodImpl(MethodImplOptions.AggressiveInlining)]
18
                     protected abstract TElement GetNext(TElement element);
19
                     [MethodImpl(MethodImplOptions.AggressiveInlining)]
20
                     protected abstract TElement GetSize();
21
                     [MethodImpl(MethodImplOptions.AggressiveInlining)]
                     protected abstract void SetFirst(TElement element);
23
                     [MethodImpl(MethodImplOptions.AggressiveInlining)]
24
                     protected abstract void SetLast(TElement element)
25
                     [MethodImpl(MethodImplOptions.AggressiveInlining)]
                     protected abstract void SetPrevious(TElement element, TElement previous);
27
                     [MethodImpl(MethodImplOptions.AggressiveInlining)]
28
                     protected abstract void SetNext(TElement element,
                                                                                                               TElement next);
                     [MethodImpl(MethodImplOptions.AggressiveInlining)]
                     protected abstract void SetSize(TElement size);
31
                     [MethodImpl(MethodImplOptions.AggressiveInlining)]
32
                     protected void IncrementSize() => SetSize(Increment(GetSize()));
33
                     [MethodImpl(MethodImplOptions.AggressiveInlining)]
34
                    protected void DecrementSize() => SetSize(Decrement(GetSize()));
35
             }
36
      }
37
./Lists/OpenDoublyLinkedListMethods.cs
      #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
      namespace Platform.Collections.Methods.Lists
 3
 4
             public abstract class OpenDoublyLinkedListMethods<TElement> :
 5
                    DoublyLinkedListMethodsBase<TElement>
                    public void AttachBefore(TElement baseElement, TElement newElement)
```

```
var baseElementPrevious = GetPrevious(baseElement);
    SetPrevious(newElement, baseElementPrevious);
    SetNext(newElement, baseElement);
    if (EqualToZero(baseElementPrevious))
    {
        SetFirst(newElement);
    }
    else
    {
        SetNext(baseElementPrevious, newElement);
    SetPrevious(baseElement, newElement);
    IncrementSize();
}
public void AttachAfter(TElement baseElement, TElement newElement)
    var baseElementNext = GetNext(baseElement);
    SetPrevious(newElement, baseElement);
    SetNext(newElement, baseElementNext);
    if (EqualToZero(baseElementNext))
    {
        SetLast(newElement);
    }
    else
    {
        SetPrevious(baseElementNext, newElement);
    SetNext(baseElement, newElement);
    IncrementSize();
public void AttachAsFirst(TElement element)
    var first = GetFirst();
    if (EqualToZero(first))
        SetFirst(element);
        SetLast(element);
        SetPrevious(element, GetZero());
        SetNext(element, GetZero());
        IncrementSize();
    }
    else
    {
        AttachBefore(first, element);
    }
}
public void AttachAsLast(TElement element)
    var last = GetLast();
    if (EqualToZero(last))
        AttachAsFirst(element);
    }
    else
    {
        AttachAfter(last, element);
    }
}
public void Detach(TElement element)
    var elementPrevious = GetPrevious(element);
    var elementNext = GetNext(element);
    if (EqualToZero(elementPrevious))
        SetFirst(elementNext);
    }
    else
        SetNext(elementPrevious, elementNext);
    if (EqualToZero(elementNext))
        SetLast(elementPrevious);
```

11

13

14

15

16

17

18

 $\frac{20}{21}$

22 23

25

26

27

28

29

30

32 33

34

35 36

38 39 40

41

43

44 45 46

47

48

49

51

52

53

54

57

59

60

62

63

64

65

66

69 70

71

74

75 76

77

78 79

80

81 82

```
}
86
                else
87
                {
88
                    SetPrevious(elementNext, elementPrevious);
90
                SetPrevious(element, GetZero());
91
                SetNext(element, GetZero());
92
                DecrementSize();
93
            }
94
        }
95
   }
./Trees/SizeBalancedTreeMethods2.cs
   using System;
1
2
   #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
3
   namespace Platform.Collections.Methods.Trees
5
        /// <summary>
        /// Experimental implementation, don't use it yet.
        /// </summary>
       public unsafe abstract class SizeBalancedTreeMethods2<TElement> :
10

→ SizedBinaryTreeMethodsBase<TElement>

            protected override void AttachCore(IntPtr root, TElement newNode)
{
12
13
                if
                   (ValueEqualToZero(root))
                {
15
                    System.Runtime.CompilerServices.Unsafe.Write((void*)root, newNode);
16
                    IncrementSize(System.Runtime.CompilerServices.Unsafe.Read<TElement>((void*)root) |
17
                }
                else
19
20
                    IncrementSize(System.Runtime.CompilerServices.Unsafe.Read<TElement>((void*)root)
                    if (FirstIsToTheLeftOfSecond(newNode,
22
                        System.Runtime.CompilerServices.Unsafe.Read<TElement>((void*)root)))
23
                         {	t AttachCore}({	t GetLeftPointer}({	t System.Runtime.CompilerServices.Unsafe.Read<{	t TEleme}_1
                         → nt>((void*)root)),
                            newNode)
                        LeftMaintain(root);
25
                    }
                    else
2.7
                        AttachCore(GetRightPointer(System.Runtime.CompilerServices.Unsafe.Read<TElem_
29

→ ent>((void*)root)),
                            newNode);
                        RightMaintain(root);
                    }
31
                }
32
            }
33
34
            protected override void DetachCore(IntPtr root, TElement nodeToDetach)
35
                if (ValueEqualToZero(root))
37
                {
38
                    return;
39
40
                var currentNode = root;
41
                var parent = IntPtr.Zero; /* Изначально зануление, так как родителя может и не быть
42
                    (Корень дерева). */
                var replacementNode = GetZero();
43
                while (!IsEquals(System.Runtime.CompilerServices.Unsafe.Read<TElement>((void*)curren_
                    tNode)
                    nodeToDetach))
                {
45
                    SetSize(System.Runtime.CompilerServices.Unsafe.Read<TElement>((void*)currentNode
                        Decrement(GetSize(System.Runtime.CompilerServices.Unsafe.Read<TElement>((voi
                        d*)currentNode))));
                    if (FirstIsToTheLeftOfSecond(nodeToDetach,
                        System.Runtime.CompilerServices.Unsafe.Read<TElement>((void*)currentNode)))
                        parent = currentNode;
49
```

```
currentNode = GetLeftPointer(System.Runtime.CompilerServices.Unsafe.Read<TEl
                ement>((void*)currentNode));
        }
        else if (FirstIsToTheRightOfSecond(nodeToDetach,
            System.Runtime.CompilerServices.Unsafe.Read<TElement>((void*)currentNode)))
        {
            parent = currentNode;
            currentNode = GetRightPointer(System.Runtime.CompilerServices.Unsafe.Read<TE
            → lement>((void*)currentNode));
        else
        {
            throw new InvalidOperationException("Duplicate link found in the tree.");
    }
    if (!ValueEqualToZero(GetLeftPointer(nodeToDetach)) &&
        !ValueEqualToZero(GetRightPointer(nodeToDetach)))
        var minNode = GetRightValue(nodeToDetach)
        while (!EqualToZero(GetLeftValue(minNode)))
            minNode = GetLeftValue(minNode); /* Передвигаемся до минимума */
        DetachCore(GetRightPointer(nodeToDetach), minNode);
        SetLeft(minNode, GetLeftValue(nodeToDetach));
        if (!ValueEqualToZero(GetRightPointer(nodeToDetach)))
        {
            SetRight(minNode, GetRightValue(nodeToDetach));
            SetSize(minNode, Increment(Add(GetSize(GetLeftValue(nodeToDetach)),

→ GetSize(GetRightValue(nodeToDetach))));
        }
        else
        {
            SetSize(minNode, Increment(GetSize(GetLeftValue(nodeToDetach))));
        replacementNode = minNode;
    else if (!ValueEqualToZero(GetLeftPointer(nodeToDetach)))
        replacementNode = GetLeftValue(nodeToDetach);
    else if (!ValueEqualToZero(GetRightPointer(nodeToDetach)))
        replacementNode = GetRightValue(nodeToDetach);
    if
      (parent == IntPtr.Zero)
    {
        System.Runtime.CompilerServices.Unsafe.Write((void*)root, replacementNode);
    else if (IsEquals(GetLeftValue(System.Runtime.CompilerServices.Unsafe.Read<TElement>_
        ((void*)parent)),
        nodeToDetach))
        SetLeft(System.Runtime.CompilerServices.Unsafe.Read<TElement>((void*)parent),

→ replacementNode);

    else if (IsEquals(GetRightValue(System.Runtime.CompilerServices.Unsafe.Read<TElement
        >((void*)parent)),
    \hookrightarrow
        nodeToDetach))
    {
        SetRight(System.Runtime.CompilerServices.Unsafe.Read<TElement>((void*)parent),

→ replacementNode);

    ClearNode(nodeToDetach);
}
private void LeftMaintain(IntPtr root)
    if (!ValueEqualToZero(root))
        var rootLeftNode = GetLeftPointer(System.Runtime.CompilerServices.Unsafe.Read<TE |</pre>
            lement>((void*)root));
           (!ValueEqualToZero(rootLeftNode))
            var rootRightNode = GetRightPointer(System.Runtime.CompilerServices.Unsafe.R_

→ ead<TElement>((void*)root));
```

53

55

56

57

58

59

61

62

63

65 66

69

70

71

72

73

76

77

78 79

80 81

82 83

84 85

86 87

88 89

90

91

92

94

95

99

100

101

102

103 104

105 106

107 108

109

111

```
var rootLeftNodeLeftNode = GetLeftPointer(System.Runtime.CompilerServices.Un | 
113
                              safe.Read<TElement>((void*)rootLeftNode));
                              (!ValueEqualToZero(rootLeftNodeLeftNode) &&
114
                               (Value Equal To Zero(root Right Node) \mid \mid Greater Than(Get Size(System.Runtime.C_1))
115
                                   ompilerServices.Unsafe.Read<TElement>((void*)rootLeftNodeLeftNode))
                                   GetSize(System.Runtime.CompilerServices.Unsafe.Read<TElement>((void*
                                   )rootRightNode)))))
                          {
116
                               RightRotate(root);
117
                          }
                          else
119
120
                               var rootLeftNodeRightNode = GetRightPointer(System.Runtime.CompilerServi | 
                                   ces.Unsafe.Read<TElement>((void*)rootLeftNode));
                               if (!ValueEqualToZero(rootLeftNodeRightNode) &&
122
                                   (ValueEqualToZero(rootRightNode) |
123
                                        GreaterThan(GetSize(System.Runtime.CompilerServices.Unsafe.Read
                                       TElement>((void*)rootLeftNodeRightNode)),
                                       GetSize(System.Runtime.CompilerServices.Unsafe.Read<TElement>((v)
                                       oid*)rootRightNode)))))
                               {
124
                                   LeftRotate(GetLeftPointer(System.Runtime.CompilerServices.Unsafe.Rea
                                      d<TElement>((void*)root)));
                                   RightRotate(root);
126
                               }
127
                               else
128
                               {
129
                                   return;
131
132
                          \texttt{LeftMaintain}(\texttt{GetLeftPointer}(\texttt{System.Runtime.CompilerServices.Unsafe.Read} < \texttt{TEle}_1
133
                              ment>((void*)root)));
                          RightMaintain(GetRightPointer(System.Runtime.CompilerServices.Unsafe.Read<TE
                               lement>((void*)root)));
                          LeftMaintain(root);
135
                          RightMaintain(root);
136
                      }
                 }
138
             }
139
140
             private void RightMaintain(IntPtr root)
141
142
                 if (!ValueEqualToZero(root))
144
                      var rootRightNode = GetRightPointer(System.Runtime.CompilerServices.Unsafe.Read
145
                          TElement>((void*)root));
                         (!ValueEqualToZero(rootRightNode))
146
147
                          {\tt var} \ \ {\tt rootLeftNode} \ = \ {\tt GetLeftPointer} ( {\tt System.Runtime.CompilerServices.Unsafe.Rea} \ )

    d<TElement>((void*)root));
                          var rootRightNodeRightNode = GetRightPointer(System.Runtime.CompilerServices | 
149
                               .Unsafe.Read<TElement>((void*)rootRightNode));
                          if (!ValueEqualToZero(rootRightNodeRightNode) &&
150
                               (ValueEqualToZero(rootLeftNode) ||
                                   GreaterThan(GetSize(System.Runtime.CompilerServices.Unsafe.Read<TEle
                                   ment>((void*)rootRightNodeRightNode)),
                                   GetSize(System.Runtime.CompilerServices.Unsafe.Read<TElement>((void*
                                   )rootLeftNode)))))
                          {
152
                               LeftRotate(root);
153
                          }
154
                          else
155
                          {
156
                               var rootRightNodeLeftNode = GetLeftPointer(System.Runtime.CompilerServic | 
157
                                   es.Unsafe.Read<TElement>((void*)rootRightNode));
158
                               if (!ValueEqualToZero(rootRightNodeLeftNode) &&
                                   (ValueEqualToZero(rootLeftNode) ||
159
                                        {\tt GreaterThan}({\tt GetSize}({\tt System.Runtime.CompilerServices.Unsafe.Read<_{|}}
                                       TElement>((void*)rootRightNodeLeftNode)),
                                       GetSize(System.Runtime.CompilerServices.Unsafe.Read<TElement>((v
                                       oid*)rootLeftNode)))))
160
                                   {\tt RightRotate}({\tt GetRightPointer}({\tt System.Runtime.CompilerServices.Unsafe.R}_1
                                       ead<TElement>((void*)root)));
                                   LeftRotate(root);
                               }
163
```

```
else
164
                                                                        return;
166
                                                               }
168
                                                      LeftMaintain(GetLeftPointer(System.Runtime.CompilerServices.Unsafe.Read<TEle
169
                                                              ment>((void*)root)));
                                                      \label{lem:reduced_reduced_reduced_reduced} Right \texttt{Maintain} (\texttt{GetRightPointer}(\texttt{System}. \texttt{Runtime}. \texttt{CompilerServices}. \texttt{Unsafe}. \texttt{Read} \\ < \texttt{TE}_{\perp} (\texttt{System}, \texttt{Runtime}) \\ = \texttt{TE}_{\perp} (\texttt{System}, \texttt{Runtime}, \texttt{CompilerServices}) \\ = \texttt{TE}_{\perp} (\texttt{System}, \texttt{CompilerServices}) \\ 
170
                                                               lement>((void*)root)));
                                                      LeftMaintain(root);
                                                      RightMaintain(root);
172
                                             }
173
                                   }
174
                          }
                  }
176
177
 ./Trees/SizeBalancedTreeMethods.cs
        using System;
         #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
         namespace Platform.Collections.Methods.Trees
  5
   6
                  public unsafe abstract class SizeBalancedTreeMethods<TElement> :
   7
                          SizedBinaryTreeMethodsBase<TElement>
                           protected override void AttachCore(IntPtr root, TElement node)
   9
 1.0
                                    while (true)
 11
 12
                                             var left = GetLeftPointer(System.Runtime.CompilerServices.Unsafe.Read<TElement>( |
 13
                                                      (void*)root));
                                             var leftSize = GetSizeOrZero(System.Runtime.CompilerServices.Unsafe.Read<TElemen_</pre>
 14

    t>((void*)left));
                                             var right = GetRightPointer(System.Runtime.CompilerServices.Unsafe.Read<TElement_</pre>
 15
                                              → >((void*)root));
                                             var rightSize = GetSizeOrZero(System.Runtime.CompilerServices.Unsafe.Read<TEleme | </pre>
                                                   nt>((void*)right));
                                             if (FirstIsToTheLeftOfSecond(node,
                                                     System.Runtime.CompilerServices.Unsafe.Read<TElement>((void*)root))) //
                                                     node.Key less than root.Key
                                                      if (EqualToZero(System.Runtime.CompilerServices.Unsafe.Read<TElement>((void* |
                                                               )left)))
                                                       {
 20
                                                               IncrementSize(System.Runtime.CompilerServices.Unsafe.Read<TElement>((voi
                                                                 \rightarrow d*)root));
                                                               SetSize(node, GetOne());
                                                               System.Runtime.CompilerServices.Unsafe.Write((void*)left, node);
 23
                                                      }
 25
                                                      if (FirstIsToTheRightOfSecond(node,
 26
                                                               System.Runtime.CompilerServices.Unsafe.Read<TElement>((void*)left))) //
                                                               node.Key greater than left.Key
 27
                                                               var leftRight = GetRightValue(System.Runtime.CompilerServices.Unsafe.Rea_
 2.8

→ d<TElement>((void*)left));
                                                               var leftRightSize = GetSizeOrZero(leftRight);
 29
                                                               if (GreaterThan(Increment(leftRightSize), rightSize))
 30
 31
                                                                         if (EqualToZero(leftRightSize) && EqualToZero(rightSize))
 32
                                                                         {
                                                                                  SetLeft(node, System.Runtime.CompilerServices.Unsafe.Read<TEleme
 34
                                                                                  → nt>((void*)left));
                                                                                  {\tt SetRight(node, System.Runtime.CompilerServices.Unsafe.Read < TElem_{\bot}}
 35

→ ent>((void*)root));
                                                                                  {\tt SetSize(node,\ Add(GetSize(System.Runtime.CompilerServices.Unsafe\_lambda))} \\
 36
                                                                                           .Read<TElement>((void*)left)), GetTwo())); // Two (2) -
                                                                                           размер ветки *root (right) и самого node
                                                                                  SetLeft(System.Runtime.CompilerServices.Unsafe.Read<TElement>((v<sub>|</sub>
 37
                                                                                          oid*)root)
                                                                                         GetZero());
                                                                                  SetSize(System.Runtime.CompilerServices.Unsafe.Read<TElement>((v)
                                                                                          oid*)root),
                                                                                          GetOne());
                                                                                  System.Runtime.CompilerServices.Unsafe.Write((void*)root, node);
```

```
break;
           LeftRotate(left);
           RightRotate(root);
       }
       else
            IncrementSize(System.Runtime.CompilerServices.Unsafe.Read<TElement>(
            root = left;
   else // node.Key less than left.Key
       var leftLeft = GetLeftValue(System.Runtime.CompilerServices.Unsafe.Read
           TElement>((void*)left));
       var leftLeftSize = GetSizeOrZero(leftLeft);
       if (GreaterThan(Increment(leftLeftSize), rightSize))
           RightRotate(root);
       }
       else
        {
            IncrementSize(System.Runtime.CompilerServices.Unsafe.Read<TElement>(
               (void*)root));
           root = left;
       }
   }
}
else // node.Key greater than root.Key
    if (EqualToZero(System.Runtime.CompilerServices.Unsafe.Read<TElement>((void*)
       )right)))
    {
       IncrementSize(System.Runtime.CompilerServices.Unsafe.Read<TElement>((voi

→ d*)root));
       SetSize(node, GetOne());
       System.Runtime.CompilerServices.Unsafe.Write((void*)right, node);
       break;
    if (FirstIsToTheRightOfSecond(node,
       System.Runtime.CompilerServices.Unsafe.Read<TElement>((void*)right))) //
       node.Key greater than right.Key
       var rightRight = GetRightValue(System.Runtime.CompilerServices.Unsafe.Re | 
           ad<TElement>((void*)right));
       var rightRightSize = GetSizeOrZero(rightRight);
       if (GreaterThan(Increment(rightRightSize), leftSize))
           LeftRotate(root);
       }
       else
        {
           IncrementSize(System.Runtime.CompilerServices.Unsafe.Read<TElement>(
            root = right;
       }
    else // node.Key less than right.Key
       var rightLeft = GetLeftValue(System.Runtime.CompilerServices.Unsafe.Read |
           <TElement>((void*)right));
       var rightLeftSize = GetSizeOrZero(rightLeft);
       if (GreaterThan(Increment(rightLeftSize), leftSize))
            if (EqualToZero(rightLeftSize) && EqualToZero(leftSize))
                SetLeft(node, System.Runtime.CompilerServices.Unsafe.Read<TEleme |
                → nt>((void*)root));
               SetRight(node, System.Runtime.CompilerServices.Unsafe.Read<TElem |

→ ent>((void*)right));
               SetSize(node, Add(GetSize(System.Runtime.CompilerServices.Unsafe,
                . Read<TElement>((void*)right)), GetTwo())); // Two (2) -
                → размер ветки *root (left) и самого node
               SetRight(System.Runtime.CompilerServices.Unsafe.Read<TElement>((
                   void*)root),
                   GetZero());
```

42

43

45 46

47

48 49

51 52

55 56

58

60

61

62

63

64

66 67

68

70

71

72

73 74

75

76

77

79

81

82

83

84

85

87 88

90

91

94

95

97

98

```
SetSize(System.Runtime.CompilerServices.Unsafe.Read<TElement>((v)
                             oid*)root),
                             GetOne());
                         System.Runtime.CompilerServices.Unsafe.Write((void*)root, node);
                         break;
                     RightRotate(right);
                    LeftRotate(root);
                }
                else
                {
                     IncrementSize(System.Runtime.CompilerServices.Unsafe.Read<TElement>(
                        (void*)root));
                     root = right;
                }
            }
        }
    }
}
protected override void DetachCore(IntPtr root, TElement node)
    while (true)
    {
        var left = GetLeftPointer(System.Runtime.CompilerServices.Unsafe.Read<TElement>( |
            (void*)root));
        var leftSize = GetSizeOrZero(System.Runtime.CompilerServices.Unsafe.Read<TElemen_</pre>
            t>((void*)left)):
        var right = GetRightPointer(System.Runtime.CompilerServices.Unsafe.Read<TElement | </pre>
            >((void*)root));
        var rightSize = GetSizeOrZero(System.Runtime.CompilerServices.Unsafe.Read<TEleme_</pre>
            nt>((void*)right))
        if (FirstIsToTheLeftOfSecond(node,
            System.Runtime.CompilerServices.Unsafe.Read<TElement>((void*)root))) //
            node. Key less than root. Key
            EnsureNodeInTheTree(node, left);
            var rightLeft = GetLeftValue(System.Runtime.CompilerServices.Unsafe.Read<TEl
                ement>((void*)right));
            var rightLeftSize = GetSizeOrZero(rightLeft);
            var rightRight = GetRightValue(System.Runtime.CompilerServices.Unsafe.Read<T |</pre>
                Element>((void*)right));
            var rightRightSize = GetSizeOrZero(rightRight);
            if (GreaterThan(rightRightSize, Decrement(leftSize)))
                LeftRotate(root);
            }
            else if (GreaterThan(rightLeftSize, Decrement(leftSize)))
                RightRotate(right);
                LeftRotate(root);
            else
                DecrementSize(System.Runtime.CompilerServices.Unsafe.Read<TElement>((voi
                 \rightarrow d*)root));
                root = left;
        else if (FirstIsToTheRightOfSecond(node,
            System.Runtime.CompilerServices.Unsafe.Read<TElement>((void*)root))) //
            node.Key greater than root.Key
            EnsureNodeInTheTree(node, right);
            var leftLeft = GetLeftValue(System.Runtime.CompilerServices.Unsafe.Read<TEle | </pre>

→ ment>((void*)left));
            var leftLeftSize = GetSizeOrZero(leftLeft);
            var leftRight = GetRightValue(System.Runtime.CompilerServices.Unsafe.Read<TE;</pre>
                lement>((void*)left));
                leftRightSize = GetSizeOrZero(leftRight);
            var
               (GreaterThan(leftLeftSize, Decrement(rightSize)))
            {
                RightRotate(root);
            else if (GreaterThan(leftRightSize, Decrement(rightSize)))
                LeftRotate(left);
```

102

103 104

105

107

108

109

110

112

113

116 117

118

121

122

123

125

126

127

128

129

130

131

132

133 134 135

136 137

138

139

140

142 143

144

145

147

148

149

150

152

153

154

155

156

157 158

159 160

```
RightRotate(root);
162
                                                  }
                                                  else
164
                                                  {
                                                          DecrementSize(System.Runtime.CompilerServices.Unsafe.Read<TElement>((voi
166
                                                           \rightarrow d*)root));
                                                          root = right;
167
                                                  }
169
                                         else // key equals to root. Key
170
                                                       (GreaterThanZero(leftSize) && GreaterThanZero(rightSize))
172
173
                                                               (GreaterThan(leftSize, rightSize))
174
175
                                                                   var replacement = System.Runtime.CompilerServices.Unsafe.Read<TEleme</pre>
176
                                                                         nt>((void*)left);
                                                                  while (!EqualToZero(GetRightValue(replacement)))
                                                                   {
178
                                                                           replacement = GetRightValue(replacement);
179
180
                                                                   DetachCore(left, replacement);
                                                                  SetLeft(replacement, System.Runtime.CompilerServices.Unsafe.Read<TEl
182
                                                                          ement>((void*)left));
                                                                  SetRight(replacement, System.Runtime.CompilerServices.Unsafe.Read<TE
183
                                                                         lement>((void*)right));
                                                                  FixSize(replacement);
184
                                                                  System.Runtime.CompilerServices.Unsafe.Write((void*)root,

→ replacement);

                                                          }
186
                                                          else
                                                                   var replacement = System.Runtime.CompilerServices.Unsafe.Read<TEleme |</pre>
189
                                                                         nt>((void*)right);
                                                                  while (!EqualToZero(GetLeftValue(replacement)))
190
                                                                   {
191
                                                                          replacement = GetLeftValue(replacement);
192
193
                                                                  DetachCore(right, replacement);
                                                                  {\tt SetLeft(replacement, System.Runtime.CompilerServices.Unsafe.Read < TEl_{+} and a substitution of the compiler compilers and a substitution of the compilers and a substitution of the compiler compilers and a substitution of the compiler compilers and a substitution of the compilers and a substitution of 
195

→ ement>((void*)left));
                                                                  SetRight(replacement, System.Runtime.CompilerServices.Unsafe.Read<TE
196
                                                                         lement>((void*)right));
                                                                  FixSize(replacement);
                                                                  System.Runtime.CompilerServices.Unsafe.Write((void*)root,
                                                                        replacement);
199
200
                                                  else if (GreaterThanZero(leftSize))
201
202
                                                          System.Runtime.CompilerServices.Unsafe.Write((void*)root,
203
                                                                  System.Runtime.CompilerServices.Unsafe.Read<TElement>((void*)left));
204
                                                  else if (GreaterThanZero(rightSize))
206
                                                          System.Runtime.CompilerServices.Unsafe.Write((void*)root,
207
                                                                  System.Runtime.CompilerServices.Unsafe.Read<TElement>((void*)right));
208
                                                  else
20.9
                                                  {
210
                                                          System.Runtime.CompilerServices.Unsafe.Write((void*)root, GetZero());
212
                                                  ClearNode(node);
213
                                                  break;
214
                                          }
215
                                 }
216
                         }
217
218
                         private void EnsureNodeInTheTree(TElement node, IntPtr branch)
219
220
                                 if (EqualToZero(System.Runtime.CompilerServices.Unsafe.Read<TElement>((void*)branch) |
221
                                         ))
                                 {
222
                                          throw new InvalidOperationException($"Элемент {node} не содержится в дереве.");
223
                                 }
                         }
225
                 }
226
```

```
}
227
./Trees/Sized And Threaded AVL Balanced Tree Methods.cs\\
    using System;
   using System.Runtime.CompilerServices;
using System.Text;
#if USEARRAYPOOL
 3
 4
    using Platform.Collections;
 5
    #endif
 6
    #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
    namespace Platform.Collections.Methods.Trees
10
11
        /// <summary>
12
        /// Combination of Size, Height (AVL), and threads.
13
        /// </summary>
14
        /// <remarks>
15
        /// Based on: <a href="https://github.com/programmatom/TreeLib/blob/master/TreeLib/TreeLib/G<sub>|</sub>
16
            enerated/AVLTreeList.cs">TreeLib.AVLTreeList</a>.
        /// Which itself based on: <a
17
           href="https://github.com/GNOME/glib/blob/master/glib/gtree.c">GNOME/glib/gtree</a>.
        /// </remarks>
18
        public unsafe abstract class SizedAndThreadedAVLBalancedTreeMethods<TElement> :
19
            SizedBinaryTreeMethodsBase<TElement>
20
             // TODO: Link with size of TElement
21
            private const int MaxPath = 92;
23
            protected override void PrintNode(TElement node, StringBuilder sb, int level)
24
25
                 base.PrintNode(node, sb, level);
26
                 sb.Append(' ');
2.7
                 sb.Append(GetLeftIsChild(node) ? 'l' : 'L');
                 \verb|sb.Append(GetRightIsChild(node)|? 'r' : 'R');\\
29
                 sb.Append(' ');
30
                 sb.Append(GetBalance(node));
31
            }
33
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
            protected void IncrementBalance(TElement node) => SetBalance(node,
35
                (sbyte)(GetBalance(node) + 1));
36
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
37
            protected void DecrementBalance(TElement node) => SetBalance(node,
38
                (sbyte)(GetBalance(node) - 1));
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
            protected override TElement GetLeftOrDefault(TElement node) => GetLeftIsChild(node) ?
41
             → base.GetLeftOrDefault(node) : default;
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
43
            protected override TElement GetRightOrDefault(TElement node) => GetRightIsChild(node) ?
44
             → base.GetRightOrDefault(node) : default;
            protected abstract bool GetLeftIsChild(TElement node);
46
            protected abstract void SetLeftIsChild(TElement node, bool value);
47
            protected abstract bool GetRightIsChild(TElement node);
48
            protected abstract void SetRightIsChild(TElement node, bool value);
49
            protected abstract sbyte GetBalance(TElement node);
50
            protected abstract void SetBalance(TElement node, sbyte value);
52
            protected override void AttachCore(IntPtr root, TElement node)
53
                 unchecked
55
                 {
                     // TODO: Check what is faster to use simple array or array from array pool
57
                     // TODO: Try to use stackalloc as an optimization (requires code generation,
58

→ because of generics)

    #if USEARRAYPOOL
59
                     var path = ArrayPool.Allocate<TElement>(MaxPath);
60
                     var pathPosition = 0;
                     path[pathPosition++] = default;
62
    #else
63
                     var path = new TElement[MaxPath];
64
                     var pathPosition = 1;
65
    #endif
66
                     var rootPointer = (void*)root;
67
```

```
var currentNode =
   System.Runtime.CompilerServices.Unsafe.Read<TElement>(rootPointer);
while (true)
    if (FirstIsToTheLeftOfSecond(node, currentNode))
        if (GetLeftIsChild(currentNode))
            IncrementSize(currentNode);
            path[pathPosition++] = currentNode;
            currentNode = GetLeftValue(currentNode);
        }
        else
        {
            // Threads
            SetLeft(node, GetLeftValue(currentNode));
            SetRight(node, currentNode);
            SetLeft(currentNode, node);
            SetLeftIsChild(currentNode, true);
            DecrementBalance(currentNode);
            SetSize(node, GetOne());
            FixSize(currentNode); // Should be incremented already
            break;
    else if (FirstIsToTheRightOfSecond(node, currentNode))
        if (GetRightIsChild(currentNode))
            IncrementSize(currentNode);
            path[pathPosition++] = currentNode;
            currentNode = GetRightValue(currentNode);
        }
        else
            // Threads
            SetRight(node, GetRightValue(currentNode));
            SetLeft(node, currentNode);
            SetRight(currentNode, node);
            SetRightIsChild(currentNode, true);
            IncrementBalance(currentNode);
            SetSize(node, GetOne());
            FixSize(currentNode); // Should be incremented already
            break;
        }
    }
    else
    {
        throw new InvalidOperationException("Node with the same key already
        → attached to a tree.");
    }
}
// Restore balance. This is the goodness of a non-recursive
  implementation, when we are done with balancing we 'break'
// the loop and we are done.
while (true)
    var parent = path[--pathPosition];
    var isLeftNode = !IsEquals(parent, default) && IsEquals(currentNode,

   GetLeftValue(parent));
    var currentNodeBalance = GetBalance(currentNode);
    if (currentNodeBalance < -1 || currentNodeBalance > 1)
        currentNode = Balance(currentNode);
        if (IsEquals(parent, default))
            System.Runtime.CompilerServices.Unsafe.Write((void*)root,
            else if (isLeftNode)
            SetLeft(parent, currentNode);
            FixSize(parent);
        }
        else
            SetRight(parent, currentNode);
            FixSize(parent);
```

70

71 72

7.4

7.5

76

77

78

80

81

82

83

84

87

88

90

92 93

94

96

97

99

100

102

103

105

106 107

108

109

111

112

113

114

115

117

118 119

120

121 122

124

125

126 127

128

129

131

132

133 134

135

136

138 139

140

```
}
142
                           }
                           currentNodeBalance = GetBalance(currentNode);
144
                           if (currentNodeBalance == 0 || IsEquals(parent, default))
145
                                break;
147
148
                              (isLeftNode)
149
                           {
150
                                DecrementBalance(parent);
151
                           }
152
                           else
153
                           {
154
                                IncrementBalance(parent);
156
                           currentNode = parent;
157
158
    #if USEARRAYPOOL
159
                       ArrayPool.Free(path);
160
    #endif
161
                  }
162
              }
163
164
             private TElement Balance(TElement node)
165
166
167
                  unchecked
168
                       var rootBalance = GetBalance(node);
169
                       if (rootBalance < -1)</pre>
170
171
                           var left = GetLeftValue(node);
172
                           if (GetBalance(left) > 0)
                                SetLeft(node, LeftRotateWithBalance(left));
175
                                FixSize(node);
176
                           }
177
                           node = RightRotateWithBalance(node);
178
179
                       else if (rootBalance > 1)
181
                           var right = GetRightValue(node);
182
                           if (GetBalance(right) < 0)</pre>
183
                                SetRight(node, RightRotateWithBalance(right));
185
                                FixSize(node);
186
                           node = LeftRotateWithBalance(node);
188
189
190
                       return node;
                  }
191
              }
192
193
             protected TElement LeftRotateWithBalance(TElement node)
194
                  unchecked
196
                  {
197
                       var right = GetRightValue(node);
198
                       if (GetLeftIsChild(right))
199
                       {
200
                           SetRight(node, GetLeftValue(right));
201
                       }
202
                       else
203
                       {
204
                           SetRightIsChild(node, false);
205
                           SetLeftIsChild(right, true);
206
207
                       SetLeft(right, node);
208
                       // Fix size
209
                       SetSize(right, GetSize(node));
210
211
                       FixSize(node);
                       // Fix balance
212
                       var rootBalance = GetBalance(node);
213
                       var rightBalance = GetBalance(right);
                       if (rightBalance <= 0)</pre>
215
216
217
                           if (rootBalance >= 1)
                           {
218
                                SetBalance(right, (sbyte)(rightBalance - 1));
219
```

```
}
            else
            {
                SetBalance(right, (sbyte)(rootBalance + rightBalance - 2));
            SetBalance(node, (sbyte)(rootBalance - 1));
        else
               (rootBalance <= rightBalance)</pre>
                SetBalance(right, (sbyte)(rootBalance - 2));
            else
            {
                SetBalance(right, (sbyte)(rightBalance - 1));
            SetBalance(node, (sbyte)(rootBalance - rightBalance - 1));
        return right;
    }
}
protected TElement RightRotateWithBalance(TElement node)
    unchecked
    {
        var left = GetLeftValue(node);
        if (GetRightIsChild(left))
            SetLeft(node, GetRightValue(left));
        }
        else
        {
            SetLeftIsChild(node, false);
            SetRightIsChild(left, true);
        SetRight(left, node);
        // Fix size
        SetSize(left, GetSize(node));
        FixSize(node);
        // Fix balance
        var rootBalance = GetBalance(node);
        var leftBalance = GetBalance(left);
        if (leftBalance <= 0)</pre>
            if (leftBalance > rootBalance)
                SetBalance(left, (sbyte)(leftBalance + 1));
            }
            else
            {
                SetBalance(left, (sbyte)(rootBalance + 2));
            SetBalance(node, (sbyte)(rootBalance - leftBalance + 1));
        else
        {
            if (rootBalance <= -1)</pre>
            {
                SetBalance(left, (sbyte)(leftBalance + 1));
            }
            else
            {
                SetBalance(left, (sbyte)(rootBalance + leftBalance + 2));
            SetBalance(node, (sbyte)(rootBalance + 1));
        return left;
    }
}
protected TElement GetNext(TElement node)
    unchecked
        var current = GetRightValue(node);
        if (GetRightIsChild(node))
```

221

222

 $\frac{223}{224}$

 $\frac{225}{226}$

 $\frac{227}{228}$

230

231 232

233

234

236

237

239

240

 $\frac{242}{243}$

244

 245

 246

247

248 249

250

251 252

253

254

256

257

259

260

261

262

263

 $\frac{264}{265}$

266 267

268

269

270

 $\frac{272}{273}$

 $\frac{274}{275}$

276

277

278

279

280

281

282 283

284 285

287

289

290 291

292 293

294 295

296

```
while (GetLeftIsChild(current))
299
                               current = GetLeftValue(current);
301
302
                      return current;
304
                 }
305
             }
307
             protected TElement GetPrevious(TElement node)
309
                 unchecked
310
311
                      var current = GetLeftValue(node);
312
                      if (GetLeftIsChild(node))
313
                          while (GetRightIsChild(current))
315
316
                               current = GetRightValue(current);
317
                          }
318
319
                      return current;
                 }
321
             }
322
323
             protected override void DetachCore(IntPtr root, TElement node)
324
325
                 unchecked
326
327
    #if USEARRAYPOOL
328
                      var path = ArrayPool.Allocate<TElement>(MaxPath);
329
                      var pathPosition = 0;
330
                      path[pathPosition++] = default;
331
    #else
332
                      var path = new TElement[MaxPath];
333
                      var pathPosition = 1;
334
    #endif
335
                      var rootPointer = (void*)root;
336
                      var currentNode =
337
                         System.Runtime.CompilerServices.Unsafe.Read<TElement>(rootPointer);
                      while (true)
338
339
                          if (FirstIsToTheLeftOfSecond(node, currentNode))
340
341
                               if (!GetLeftIsChild(currentNode))
                               {
343
                                   throw new InvalidOperationException("Cannot find a node.");
344
345
                               DecrementSize(currentNode);
                               path[pathPosition++] = currentNode;
347
                               currentNode = GetLeftValue(currentNode);
348
                          else if (FirstIsToTheRightOfSecond(node, currentNode))
350
351
                               if (!GetRightIsChild(currentNode))
352
                               {
353
                                   throw new InvalidOperationException("Cannot find a node.");
354
355
                               DecrementSize(currentNode);
356
                               path[pathPosition++] = currentNode;
357
                               currentNode = GetRightValue(currentNode);
358
                          }
359
                          else
360
                          {
361
                               break;
362
                          }
363
                      var parent = path[--pathPosition];
365
                      var balanceNode = parent;
366
                      var isLeftNode = !IsEquals(parent, default) && IsEquals(currentNode,
367
                          GetLeftValue(parent));
                      if (!GetLeftIsChild(currentNode))
368
369
                          if (!GetRightIsChild(currentNode)) // node has no children
370
371
                               if (IsEquals(parent, default))
372
373
                                   System.Runtime.CompilerServices.Unsafe.Write(rootPointer, GetZero());
374
                               }
375
```

```
else if (isLeftNode)
            SetLeftIsChild(parent, false);
            SetLeft(parent, GetLeftValue(currentNode));
            IncrementBalance(parent);
        else
            SetRightIsChild(parent, false);
            SetRight(parent, GetRightValue(currentNode));
            DecrementBalance(parent);
    }
    else // node has a right child
        var successor = GetNext(currentNode);
        SetLeft(successor, GetLeftValue(currentNode));
        var right = GetRightValue(currentNode);
        if (IsEquals(parent, default))
            System.Runtime.CompilerServices.Unsafe.Write(rootPointer, right);
        else if (isLeftNode)
            SetLeft(parent, right);
            IncrementBalance(parent);
        }
        else
        {
            SetRight(parent, right);
            DecrementBalance(parent);
else // node has a left child
    if (!GetRightIsChild(currentNode))
    {
        var predecessor = GetPrevious(currentNode);
        SetRight(predecessor, GetRightValue(currentNode));
        var leftValue = GetLeftValue(currentNode);
        if (IsEquals(parent, default))
            System.Runtime.CompilerServices.Unsafe.Write(rootPointer, leftValue);
        else if (isLeftNode)
            SetLeft(parent, leftValue);
            IncrementBalance(parent);
        }
        else
        {
            SetRight(parent, leftValue);
            DecrementBalance(parent);
        }
    else // node has a both children (left and right)
        var predecessor = GetLeftValue(currentNode);
        var successor = GetRightValue(currentNode);
var successorParent = currentNode;
        int previousPathPosition = ++pathPosition;
        // find the immediately next node (and its parent)
        while (GetLeftIsChild(successor))
            path[++pathPosition] = successorParent = successor;
            successor = GetLeftValue(successor);
            if (!IsEquals(successorParent, currentNode))
            {
                DecrementSize(successorParent);
        path[previousPathPosition] = successor;
        balanceNode = path[pathPosition];
        // remove 'successor' from the tree
        if (!IsEquals(successorParent, currentNode))
        {
            if (!GetRightIsChild(successor))
```

379

381

382 383

384

385

386 387

388

389

391

392

394 395

396 397

398 399

401

402 403

404

405

407 408 409

410

412

413

414

415 416

417

418

419

421

422

423

424

425

427

428 429

430 431

432 433

434

435 436

437

438

439 440

442

443

444

449

450

452

```
{
                SetLeftIsChild(successorParent, false);
            }
            else
            {
                SetLeft(successorParent, GetRightValue(successor));
            IncrementBalance(successorParent);
            SetRightIsChild(successor, true);
            SetRight(successor, GetRightValue(currentNode));
        }
        else
        {
            DecrementBalance(currentNode);
        // set the predecessor's successor link to point to the right place
        while (GetRightIsChild(predecessor))
            predecessor = GetRightValue(predecessor);
        SetRight(predecessor, successor);
        // prepare 'successor' to replace 'node'
        var left = GetLeftValue(currentNode);
        SetLeftIsChild(successor, true);
        SetLeft(successor, left);
        SetBalance(successor, GetBalance(currentNode));
        FixSize(successor);
        if (IsEquals(parent, default))
        {
            System.Runtime.CompilerServices.Unsafe.Write(rootPointer, successor);
        }
        else if (isLeftNode)
            SetLeft(parent, successor);
        }
        else
        {
            SetRight(parent, successor);
    }
}
// restore balance
if (!IsEquals(balanceNode, default))
    while (true)
        var balanceParent = path[--pathPosition];
        isLeftNode = !IsEquals(balanceParent, default) && IsEquals(balanceNode,

   GetLeftValue(balanceParent));
        var currentNodeBalance = GetBalance(balanceNode);
        if (currentNodeBalance < -1 || currentNodeBalance > 1)
            balanceNode = Balance(balanceNode);
            if (IsEquals(balanceParent, default))
            {
                System.Runtime.CompilerServices.Unsafe.Write(rootPointer,

→ balanceNode);

            }
            else if (isLeftNode)
                SetLeft(balanceParent, balanceNode);
            else
            {
                SetRight(balanceParent, balanceNode);
            }
        currentNodeBalance = GetBalance(balanceNode);
        if (currentNodeBalance != 0 || IsEquals(balanceParent, default))
        {
            break;
        }
        if (isLeftNode)
            IncrementBalance(balanceParent);
        }
        else
        {
```

456

458

459 460

461

462

463

464

465

466 467

468

469

471

472 473

474

475

476

478

479

480 481

482

483

485 486 487

488

489

491 492

494

495

496 497

498 499

500

501

502

503

505

506

507

508

509

510 511

512 513

514

515

516

518

519

520

521

522

523

524 525 526

527

528

```
DecrementBalance(balanceParent);
530
                              balanceNode = balanceParent;
532
                         }
534
                     ClearNode(node);
535
    #if USEARRAYPOOL
536
                     ArrayPool.Free(path);
537
    #endif
                 }
539
             }
540
541
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
542
            protected override void ClearNode(TElement node)
543
544
                 SetLeft(node, GetZero());
545
                 SetRight(node, GetZero());
SetSize(node, GetZero());
546
547
                 SetLeftIsChild(node, false);
548
                 SetRightIsChild(node, false);
549
                 SetBalance(node, 0);
550
             }
        }
552
553
./Trees/SizedBinaryTreeMethodsBase.cs
    using System;
    using System.Runtime.CompilerServices;
 2
    using System.Text;
 3
    using Platform. Numbers;
 4
    //#define ENABLE_TREE_AUTO_DEBUG_AND_VALIDATION
    #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
    namespace Platform.Collections.Methods.Trees
    {
10
        public unsafe abstract class SizedBinaryTreeMethodsBase<TElement> :
11
            GenericCollectionMethodsBase<TElement>
12
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
13
            protected abstract IntPtr GetLeftPointer(TElement node);
14
15
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
16
             protected abstract IntPtr GetRightPointer(TElement node);
17
1.8
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
            protected abstract TElement GetLeftValue(TElement node);
20
21
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
22
            protected abstract TElement GetRightValue(TElement node);
23
24
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
25
            protected abstract TElement GetSize(TElement node);
26
27
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
2.8
            protected abstract void SetLeft(TElement node, TElement left);
30
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
31
            protected abstract void SetRight(TElement node, TElement right);
33
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
             protected abstract void SetSize(TElement node, TElement size);
35
36
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
37
            protected abstract bool FirstIsToTheLeftOfSecond(TElement first, TElement second);
38
40
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
            protected abstract bool FirstIsToTheRightOfSecond(TElement first, TElement second);
41
42
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
43
            protected virtual TElement GetLeftOrDefault(TElement node) => GetLeftPointer(node) !=
                IntPtr.Zero ? GetLeftValue(node) : default;
45
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
46
             protected virtual TElement GetRightOrDefault(TElement node) => GetRightPointer(node) !=
                IntPtr.Zero ? GetRightValue(node) : default;
48
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
49
            protected void IncrementSize(TElement node) => SetSize(node, Increment(GetSize(node)));
```

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
52
             protected void DecrementSize(TElement node) => SetSize(node, Decrement(GetSize(node)));
54
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
             protected TElement GetLeftSize(TElement node) => GetSizeOrZero(GetLeftOrDefault(node));
56
57
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
            protected TElement GetRightSize(TElement node) => GetSizeOrZero(GetRightOrDefault(node));
59
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
61
             protected TElement GetSizeOrZero(TElement node) => EqualToZero(node) ? GetZero() :
62

→ GetSize(node);

             [MethodImpl(MethodImplOptions.AggressiveInlining)]
            protected void FixSize(TElement node) => SetSize(node, Increment(Add(GetLeftSize(node),
65
                 GetRightSize(node))));
66
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
67
            protected void LeftRotate(IntPtr root)
69
                 var rootPointer = (void*)root;
7.0
                 System.Runtime.CompilerServices.Unsafe.Write(rootPointer,
                 LeftRotate(System.Runtime.CompilerServices.Unsafe.Read<TElement>(rootPointer)));
73
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
             protected TElement LeftRotate(TElement root)
75
76
                 var right = GetRightValue(root);
77
    #if ENABLE_TREE_AUTO_DEBUG_AND_VALIDATION
78
                 if (EqualToZero(right))
79
80
                     throw new Exception("Right is null.");
81
                 }
82
    #endif
83
                 SetRight(root, GetLeftValue(right));
84
                 SetLeft(right, root);
SetSize(right, GetSize(root));
85
86
87
                 FixSize(root);
                 return right;
88
             }
90
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
            protected void RightRotate(IntPtr root)
92
93
                 var rootPointer = (void*)root;
                 System.Runtime.CompilerServices.Unsafe.Write(rootPointer,
95
                 RightRotate(System.Runtime.CompilerServices.Unsafe.Read<TElement>(rootPointer)));
96
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
98
             protected TElement RightRotate(TElement root)
99
100
    var left = GetLeftValue(root);
#if ENABLE_TREE_AUTO_DEBUG_AND_VALIDATION
101
102
                 if (EqualToZero(left))
                 {
104
                     throw new Exception("Left is null.");
105
                 }
106
    #endif
107
                 SetLeft(root, GetRightValue(left));
108
                 SetRight(left, root);
109
                 SetSize(left, GetSize(root));
110
                 FixSize(root);
111
112
                 return left;
113
114
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
115
            public bool Contains(TElement node, TElement root)
116
                 while (!EqualToZero(root))
118
119
                     if (FirstIsToTheLeftOfSecond(node, root)) // node.Key < root.Key</pre>
120
                     {
121
                         root = GetLeftOrDefault(root);
122
123
                     else if (FirstIsToTheRightOfSecond(node, root)) // node.Key > root.Key
125
```

```
root = GetRightOrDefault(root);
126
                      }
                      else // node.Key == root.Key
128
129
                          return true;
130
131
132
                 return false;
133
134
135
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
136
             protected virtual void ClearNode(TElement node)
137
138
                 SetLeft(node, GetZero());
139
                 SetRight(node, GetZero());
140
                 SetSize(node, GetZero());
142
143
             public void Attach(IntPtr root, TElement node)
144
145
    #if ENABLE_TREE_AUTO_DEBUG_AND_VALIDATION
                 ValidateSizes(root);
147
                 Debug.WriteLine("--BeforeAttach--");
148
                 Debug.WriteLine(PrintNodes(root));
                 Debug.WriteLine("----"):
150
                 var sizeBefore = GetSize(root);
151
    #endif
152
153
                 if (ValueEqualToZero(root))
154
                      SetSize(node, GetOne());
155
                     System.Runtime.CompilerServices.Unsafe.Write((void*)root, node);
156
                      return:
158
    AttachCore(root, node);
#if ENABLE_TREE_AUTO_DEBUG_AND_VALIDATION
159
160
                 Debug.WriteLine("--AfterAttach--");
161
                 Debug.WriteLine(PrintNodes(root));
162
                 Debug.WriteLine("----");
                 ValidateSizes(root);
164
                 var sizeAfter = GetSize(root);
165
                 if (!IsEquals(MathHelpers.Increment(sizeBefore), sizeAfter))
166
167
                      throw new Exception("Tree was broken after attach.");
168
                 }
169
    #endif
170
171
172
             protected abstract void AttachCore(IntPtr root, TElement node);
173
174
             public void Detach(IntPtr root, TElement node)
175
176
    #if ENABLE_TREE_AUTO_DEBUG_AND_VALIDATION
177
                 ValidateSizes(root);
178
                 Debug.WriteLine("--BeforeDetach--");
179
                 Debug.WriteLine(PrintNodes(root));
                 Debug.WriteLine("-----");
181
                 var sizeBefore = GetSize(root);
182
                 if (ValueEqualToZero(root))
                 {
184
                      throw new Exception($"Элемент с {node} не содержится в дереве.");
185
                 }
186
    #endif
187
    DetachCore(root, node); #if ENABLE_TREE_AUTO_DEBUG_AND_VALIDATION
188
189
                 Debug.WriteLine("--AfterDetach--");
190
                 Debug.WriteLine(PrintNodes(root));
191
                 Debug.WriteLine("----");
                 ValidateSizes(root);
193
                 var sizeAfter = GetSize(root);
194
                 if (!IsEquals(MathHelpers.Decrement(sizeBefore), sizeAfter))
195
                 {
                      throw new Exception("Tree was broken after detach.");
197
                 }
198
    #endif
199
200
201
             protected abstract void DetachCore(IntPtr root, TElement node);
202
```

```
public TElement GetSize(IntPtr root) => root == IntPtr.Zero ? GetZero()
204
                 GetSizeOrZero(System.Runtime.CompilerServices.Unsafe.Read<TElement>((void*)root));
205
             public void FixSizes(IntPtr root)
206
207
                 if (root != IntPtr.Zero)
208
209
                      FixSizes(System.Runtime.CompilerServices.Unsafe.Read<TElement>((void*)root));
210
             }
212
213
             public void FixSizes(TElement node)
214
215
                 if (IsEquals(node, default))
216
                 {
217
                     return;
218
                 FixSizes(GetLeftOrDefault(node))
220
                 FixSizes(GetRightOrDefault(node));
221
                 FixSize(node);
222
             }
223
224
             public void ValidateSizes(IntPtr root)
225
226
                 if (root != IntPtr.Zero)
227
228
                      ValidateSizes(System.Runtime.CompilerServices.Unsafe.Read<TElement>((void*)root)
229
                      → );
                 }
230
             }
231
232
             public void ValidateSizes(TElement node)
233
234
235
                 if (IsEquals(node, default))
                 {
236
                     return;
238
                 var size = GetSize(node);
239
240
                 var leftSize = GetLeftSize(node);
                 var rightSize = GetRightSize(node);
241
                 var expectedSize = Arithmetic.Increment(Arithmetic.Add(leftSize, rightSize));
242
                 if (!IsEquals(size, expectedSize))
243
                 {
                      throw new InvalidOperationException($\sigma"Size of \{node\} is not valid. Expected
245

    size: {expectedSize}, actual size: {size}.");
246
                 ValidateSizes(GetLeftOrDefault(node));
247
                 ValidateSizes(GetRightOrDefault(node));
248
             }
249
250
             public void ValidateSize(TElement node)
252
                 var size = GetSize(node);
253
254
                 var leftSize = GetLeftSize(node);
                 var rightSize = GetRightSize(node);
255
                 var expectedSize = Arithmetic.Increment(Arithmetic.Add(leftSize, rightSize));
256
                 if (!IsEquals(size, expectedSize))
258
                      throw new InvalidOperationException($ "Size of {node} is not valid. Expected
259

    size: {expectedSize}, actual size: {size}.");
260
             }
261
262
             public string PrintNodes(IntPtr root)
263
264
                 if (root != IntPtr.Zero)
266
267
                      var sb = new StringBuilder();
                     PrintNodes(System.Runtime.CompilerServices.Unsafe.Read<TElement>((void*)root),
268
                         sb):
                     return sb.ToString();
270
                 return "";
272
273
             public string PrintNodes(TElement node)
274
275
                 var sb = new StringBuilder();
```

```
PrintNodes(node, sb);
277
278
                 return sb.ToString();
279
             public void PrintNodes(TElement node, StringBuilder sb) => PrintNodes(node, sb, 0);
281
282
             public void PrintNodes(TElement node, StringBuilder sb, int level)
283
284
                 if (IsEquals(node, default))
285
                 {
286
                     return;
287
288
                 PrintNodes(GetLeftOrDefault(node), sb, level + 1);
289
                 PrintNode(node, sb, level);
290
                 sb.AppendLine();
291
                 PrintNodes(GetRightOrDefault(node), sb, level + 1);
             }
293
294
             public string PrintNode(TElement node)
295
296
                 var sb = new StringBuilder();
297
                 PrintNode(node, sb);
298
                 return sb.ToString();
299
300
301
             protected void PrintNode(TElement node, StringBuilder sb) => PrintNode(node, sb, 0);
302
303
             protected virtual void PrintNode(TElement node, StringBuilder sb, int level)
304
305
                 sb.Append('\t', level);
306
                 sb.Append(node);
307
                 PrintNodeValue(node, sb);
308
                 sb.Append(' ');
309
                 sb.Append('s')
310
                 sb.Append(GetSize(node));
311
312
313
            protected abstract void PrintNodeValue(TElement node, StringBuilder sb);
314
        }
315
    }
316
```

Index

- ./GenericCollectionMethodsBase.cs, 1 ./Lists/CircularDoublyLinkedListMethods.cs, 2 ./Lists/DoublyLinkedListMethodsBase.cs, 3 ./Lists/OpenDoublyLinkedListMethods.cs, 3 ./Trees/SizeBalancedTreeMethods.cs, 8 ./Trees/SizeBalancedTreeMethods2.cs, 5
- ./Trees/SizedAndThreadedAVLBalancedTreeMethods.cs, 12
- ./Trees/SizedBinaryTreeMethodsBase.cs, 19