1.1 ./csharp/Platform.Data.Sequences/ISequenceAppender.cs

```
using System.Runtime.CompilerServices;

#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.Data.Sequences
{
    public interface ISequenceAppender<TLinkAddress>
    {
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        TLinkAddress Append(TLinkAddress sequence, TLinkAddress appendant);
    }
}
```

1.2 ./csharp/Platform.Data.Sequences/ISequenceWalker.cs

```
using System.Collections.Generic;
using System.Runtime.CompilerServices;

#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.Data.Sequences
{
    public interface ISequenceWalker<TLinkAddress>
    {
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        IEnumerable<IList<TLinkAddress>?> Walk(TLinkAddress sequence);
    }
}
```

1.3 ./csharp/Platform.Data.Sequences/SequenceWalker.cs

```
using System;
using System.Collections.Generic;
using System.Runtime.CompilerServices;

#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.Data.Sequences
{
    /// <remarks>
    /// Реализованный внутри алгоритм наглядно показывает,
    /// что совершенно не обязательна рекурсивная реализация (с вложенным вызовом функцией самой себя),
    /// так как стэк можно использовать намного эффективнее при ручном управлении.
    ///
    /// Решить объединять ли логику в одну функцию, или оставить 4 отдельных реализации?
    /// Решить встраивать ли защиту от зацикливания.
    /// Альтернативой защиты от закливания может быть заранее известное ограничение на погружение вглубь.
    /// А так же качественное распознавание прохода по циклическому графу.
    /// Ограничение на уровень глубины рекурсии может позволить использовать уменьшенный размер стека.
    /// Можно использовать глобальный стек (или несколько глобальных стеков на каждый поток).
    /// </remarks>
    public static class SequenceWalker
    {
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void WalkRight<TLinkAddress>(TLinkAddress sequence, Func<TLinkAddress, TLinkAddress> getSource, Func<TLinkAddress, TLinkAddress> getTarget, Func<TLinkAddress, bool> isElement, Action<TLinkAddress> visit)
        {
            var stack = new Stack<TLinkAddress>();
            var element = sequence;
            if (isElement(element))
            {
                visit(element);
            }
            else
            {
                while (true)
                {
                    if (isElement(element))
                    {
                        if (stack.Count == 0)
                        {
                            break;
                        }
                        element = stack.Pop();
                        var source = getSource(element);
```

```
                        var target = getTarget(element);
                        if (isElement(source))
                        {
                            visit(source);
                        }
                        if (isElement(target))
                        {
                            visit(target);
                        }
                        element = target;
                    }
                    else
                    {
                        stack.Push(element);
                        element = getSource(element);
                    }
                }
            }
        }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void WalkLeft<TLinkAddress>(TLinkAddress sequence, Func<TLinkAddress,
        ↪  TLinkAddress> getSource, Func<TLinkAddress, TLinkAddress> getTarget,
        ↪  Func<TLinkAddress, bool> isElement, Action<TLinkAddress> visit)
        {
            var stack = new Stack<TLinkAddress>();
            var element = sequence;
            if (isElement(element))
            {
                visit(element);
            }
            else
            {
                while (true)
                {
                    if (isElement(element))
                    {
                        if (stack.Count == 0)
                        {
                            break;
                        }
                        element = stack.Pop();
                        var source = getSource(element);
                        var target = getTarget(element);
                        if (isElement(target))
                        {
                            visit(target);
                        }
                        if (isElement(source))
                        {
                            visit(source);
                        }
                        element = source;
                    }
                    else
                    {
                        stack.Push(element);
                        element = getTarget(element);
                    }
                }
            }
        }
    }
}
```

1.4 ./csharp/Platform.Data.Sequences/StopableSequenceWalker.cs

```
using System;
using System.Collections.Generic;
using System.Runtime.CompilerServices;

#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.Data.Sequences
{
    /// <remarks>
    /// Реализованный внутри алгоритм наглядно показывает,
    /// что совершенно не обязательна рекурсивная реализация (с вложенным вызовом функцией самой
    ↪  себя),
    /// так как стэк можно использовать намного эффективнее при ручном управлении.
```

```
13      ///
14      /// Решить объединять ли логику в одну функцию, или оставить 4 отдельных реализации?
15      /// Решить встраивать ли защиту от зацикливания.
16      /// Альтернативой защиты от закливания может быть заранее известное ограничение на
        ↪ погружение вглубь.
17      /// А так же качественное распознавание прохода по циклическому графу.
18      /// Ограничение на уровень глубины рекурсии может позволить использовать уменьшенный размер
        ↪ стека.
19      /// Можно использовать глобальный стек (или несколько глобальных стеков на каждый поток).
20      /// </remarks>
21      public static class StopableSequenceWalker
22      {
23          [MethodImpl(MethodImplOptions.AggressiveInlining)]
24          public static bool WalkRight<TLinkAddress>(TLinkAddress sequence, Func<TLinkAddress,
            ↪ TLinkAddress> getSource, Func<TLinkAddress, TLinkAddress> getTarget,
            ↪ Func<TLinkAddress, bool> isElement, Action<TLinkAddress> enter, Action<TLinkAddress>
            ↪ exit, Func<TLinkAddress, bool> canEnter, Func<TLinkAddress, bool> visit)
25          {
26              var exited = 0;
27              var stack = new Stack<TLinkAddress>();
28              var element = sequence;
29              if (isElement(element))
30              {
31                  return visit(element);
32              }
33              while (true)
34              {
35                  if (isElement(element))
36                  {
37                      if (stack.Count == 0)
38                      {
39                          return true;
40                      }
41                      element = stack.Pop();
42                      exit(element);
43                      exited++;
44                      var source = getSource(element);
45                      var target = getTarget(element);
46                      if ((isElement(source) || (exited == 1 && !canEnter(source))) &&
                        ↪ !visit(source))
47                      {
48                          return false;
49                      }
50                      if ((isElement(target) || !canEnter(target)) && !visit(target))
51                      {
52                          return false;
53                      }
54                      element = target;
55                  }
56                  else
57                  {
58                      if (canEnter(element))
59                      {
60                          enter(element);
61                          exited = 0;
62                          stack.Push(element);
63                          element = getSource(element);
64                      }
65                      else
66                      {
67                          if (stack.Count == 0)
68                          {
69                              return true;
70                          }
71                          element = stack.Pop();
72                          exit(element);
73                          exited++;
74                          var source = getSource(element);
75                          var target = getTarget(element);
76                          if ((isElement(source) || (exited == 1 && !canEnter(source))) &&
                            ↪ !visit(source))
77                          {
78                              return false;
79                          }
80                          if ((isElement(target) || !canEnter(target)) && !visit(target))
81                          {
82                              return false;
83                          }
84                          element = target;
```

```
85                  }
86              }
87          }
88      }
89
90      [MethodImpl(MethodImplOptions.AggressiveInlining)]
91      public static bool WalkRight<TLinkAddress>(TLinkAddress sequence, Func<TLinkAddress,
        ↪ TLinkAddress> getSource, Func<TLinkAddress, TLinkAddress> getTarget,
        ↪ Func<TLinkAddress, bool> isElement, Func<TLinkAddress, bool> visit)
92      {
93          var stack = new Stack<TLinkAddress>();
94          var element = sequence;
95          if (isElement(element))
96          {
97              return visit(element);
98          }
99          while (true)
100         {
101             if (isElement(element))
102             {
103                 if (stack.Count == 0)
104                 {
105                     return true;
106                 }
107                 element = stack.Pop();
108                 var source = getSource(element);
109                 var target = getTarget(element);
110                 if (isElement(source) && !visit(source))
111                 {
112                     return false;
113                 }
114                 if (isElement(target) && !visit(target))
115                 {
116                     return false;
117                 }
118                 element = target;
119             }
120             else
121             {
122                 stack.Push(element);
123                 element = getSource(element);
124             }
125         }
126     }
127
128     [MethodImpl(MethodImplOptions.AggressiveInlining)]
129     public static bool WalkLeft<TLinkAddress>(TLinkAddress sequence, Func<TLinkAddress,
        ↪ TLinkAddress> getSource, Func<TLinkAddress, TLinkAddress> getTarget,
        ↪ Func<TLinkAddress, bool> isElement, Func<TLinkAddress, bool> visit)
130     {
131         var stack = new Stack<TLinkAddress>();
132         var element = sequence;
133         if (isElement(element))
134         {
135             return visit(element);
136         }
137         while (true)
138         {
139             if (isElement(element))
140             {
141                 if (stack.Count == 0)
142                 {
143                     return true;
144                 }
145                 element = stack.Pop();
146                 var source = getSource(element);
147                 var target = getTarget(element);
148                 if (isElement(target) && !visit(target))
149                 {
150                     return false;
151                 }
152                 if (isElement(source) && !visit(source))
153                 {
154                     return false;
155                 }
156                 element = source;
157             }
158             else
159             {
```

```
                        stack.Push(element);
                        element = getTarget(element);
                    }
                }
            }
        }
    }
```

Index