

LinksPlatform's Platform.Data.Doublets Class Library

1.1 ./Platform.Data.Doublets/Decorators/LinksCascadeUniquenessAndUsagesResolver.cs

```
1 using System.Runtime.CompilerServices;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Doublets.Decorators
6 {
7     public class LinksCascadeUniquenessAndUsagesResolver<TLink> : LinksUniquenessResolver<TLink>
8     {
9         [MethodImpl(MethodImplOptions.AggressiveInlining)]
10         public LinksCascadeUniquenessAndUsagesResolver(ILinks<TLink> links) : base(links) { }
11
12         [MethodImpl(MethodImplOptions.AggressiveInlining)]
13         protected override TLink ResolveAddressChangeConflict(TLink oldLinkAddress, TLink
14             ↪ newLinkAddress)
15         {
16             // Use Facade (the last decorator) to ensure recursion working correctly
17             Facade.MergeUsages(oldLinkAddress, newLinkAddress);
18             return base.ResolveAddressChangeConflict(oldLinkAddress, newLinkAddress);
19         }
20     }
```

1.2 ./Platform.Data.Doublets/Decorators/LinksCascadeUsagesResolver.cs

```
1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Decorators
7 {
8     /// <remarks>
9     /// <para>Must be used in conjunction with NonNullContentsLinkDeletionResolver.</para>
10    /// <para>Должен использоваться вместе с NonNullContentsLinkDeletionResolver.</para>
11    /// </remarks>
12    public class LinksCascadeUsagesResolver<TLink> : LinksDecoratorBase<TLink>
13    {
14        [MethodImpl(MethodImplOptions.AggressiveInlining)]
15        public LinksCascadeUsagesResolver(ILinks<TLink> links) : base(links) { }
16
17        [MethodImpl(MethodImplOptions.AggressiveInlining)]
18        public override void Delete(IList<TLink> restrictions)
19        {
20            var linkIndex = restrictions[Constants.IndexPart];
21            // Use Facade (the last decorator) to ensure recursion working correctly
22            Facade.DeleteAllUsages(linkIndex);
23            Links.Delete(linkIndex);
24        }
25    }
26 }
```

1.3 ./Platform.Data.Doublets/Decorators/LinksDecoratorBase.cs

```
1 using System;
2 using System.Collections.Generic;
3 using System.Runtime.CompilerServices;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data.Doublets.Decorators
8 {
9     public abstract class LinksDecoratorBase<TLink> : LinksOperatorBase<TLink>, ILinks<TLink>
10    {
11        private ILinks<TLink> _facade;
12
13        public LinksConstants<TLink> Constants
14        {
15            [MethodImpl(MethodImplOptions.AggressiveInlining)]
16            get;
17        }
18
19        public ILinks<TLink> Facade
20        {
21            [MethodImpl(MethodImplOptions.AggressiveInlining)]
22            get => _facade;
23            [MethodImpl(MethodImplOptions.AggressiveInlining)]
24            set
25            {
26                _facade = value;
27            }
28        }
29    }
```

```

27         if (Links is LinksDecoratorBase<TLink> decorator)
28         {
29             decorator.Facade = value;
30         }
31     }
32 }
33
34 [MethodImpl(MethodImplOptions.AggressiveInlining)]
35 protected LinksDecoratorBase(ILinks<TLink> links) : base(links)
36 {
37     Constants = links.Constants;
38     Facade = this;
39 }
40
41 [MethodImpl(MethodImplOptions.AggressiveInlining)]
42 public virtual TLink Count(IList<TLink> restrictions) => Links.Count(restrictions);
43
44 [MethodImpl(MethodImplOptions.AggressiveInlining)]
45 public virtual TLink Each(Func<IList<TLink>, TLink> handler, IList<TLink> restrictions)
46     => Links.Each(handler, restrictions);
47
48 [MethodImpl(MethodImplOptions.AggressiveInlining)]
49 public virtual TLink Create(IList<TLink> restrictions) => Links.Create(restrictions);
50
51 [MethodImpl(MethodImplOptions.AggressiveInlining)]
52 public virtual TLink Update(IList<TLink> restrictions, IList<TLink> substitution) =>
53     Links.Update(restrictions, substitution);
54
55 [MethodImpl(MethodImplOptions.AggressiveInlining)]
56 public virtual void Delete(IList<TLink> restrictions) => Links.Delete(restrictions);
57 }
58 }

```

1.4 ./Platform.Data.Doublets/Decorators/LinksDisposableDecoratorBase.cs

```

1  using System.Runtime.CompilerServices;
2  using Platform.Disposables;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5  #pragma warning disable CA1063 // Implement IDisposable Correctly
6
7  namespace Platform.Data.Doublets.Decorators
8  {
9      public abstract class LinksDisposableDecoratorBase<TLink> : LinksDecoratorBase<TLink>,
10         ↳ ILinks<TLink>, System.IDisposable
11     {
12         protected class DisposableWithMultipleCallsAllowed : Disposable
13         {
14             [MethodImpl(MethodImplOptions.AggressiveInlining)]
15             public DisposableWithMultipleCallsAllowed(Disposal disposal) : base(disposal) { }
16
17             protected override bool AllowMultipleDisposeCalls
18             {
19                 [MethodImpl(MethodImplOptions.AggressiveInlining)]
20                 get => true;
21             }
22         }
23
24         protected readonly DisposableWithMultipleCallsAllowed Disposable;
25
26         [MethodImpl(MethodImplOptions.AggressiveInlining)]
27         protected LinksDisposableDecoratorBase(ILinks<TLink> links) : base(links) => Disposable
28             ↳ = new DisposableWithMultipleCallsAllowed(Dispose);
29
30         [MethodImpl(MethodImplOptions.AggressiveInlining)]
31         ~LinksDisposableDecoratorBase() => Disposable.Destruct();
32
33         [MethodImpl(MethodImplOptions.AggressiveInlining)]
34         public void Dispose() => Disposable.Dispose();
35
36         [MethodImpl(MethodImplOptions.AggressiveInlining)]
37         protected virtual void Dispose(bool manual, bool wasDisposed)
38         {
39             if (!wasDisposed)
40             {
41                 Links.DisposeIfPossible();
42             }
43         }
44     }
45 }

```

1.5 ./Platform.Data.Doublets/Decorators/LinksInnerReferenceExistenceValidator.cs

```

1 using System;
2 using System.Collections.Generic;
3 using System.Runtime.CompilerServices;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data.Doublets.Decorators
8 {
9     // TODO: Make LinksExternalReferenceValidator. A layer that checks each link to exist or to
10     ↪ be external (hybrid link's raw number).
11     public class LinksInnerReferenceExistenceValidator<TLink> : LinksDecoratorBase<TLink>
12     {
13         [MethodImpl(MethodImplOptions.AggressiveInlining)]
14         public LinksInnerReferenceExistenceValidator(ILinks<TLink> links) : base(links) { }
15
16         [MethodImpl(MethodImplOptions.AggressiveInlining)]
17         public override TLink Each(Func<IList<TLink>, TLink> handler, IList<TLink> restrictions)
18         {
19             Links.EnsureInnerReferenceExists(restrictions, nameof(restrictions));
20             return Links.Each(handler, restrictions);
21         }
22
23         [MethodImpl(MethodImplOptions.AggressiveInlining)]
24         public override TLink Update(IList<TLink> restrictions, IList<TLink> substitution)
25         {
26             // TODO: Possible values: null, ExistentLink or NonExistentHybrid(ExternalReference)
27             Links.EnsureInnerReferenceExists(restrictions, nameof(restrictions));
28             Links.EnsureInnerReferenceExists(substitution, nameof(substitution));
29             return Links.Update(restrictions, substitution);
30         }
31
32         [MethodImpl(MethodImplOptions.AggressiveInlining)]
33         public override void Delete(IList<TLink> restrictions)
34         {
35             var link = restrictions[Constants.IndexPart];
36             Links.EnsureLinkExists(link, nameof(link));
37             Links.Delete(link);
38         }
39     }
40 }

```

1.6 ./Platform.Data.Doublets/Decorators/LinksItselfConstantToSelfReferenceResolver.cs

```

1 using System;
2 using System.Collections.Generic;
3 using System.Runtime.CompilerServices;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data.Doublets.Decorators
8 {
9     public class LinksItselfConstantToSelfReferenceResolver<TLink> : LinksDecoratorBase<TLink>
10     {
11         private static readonly EqualityComparer<TLink> _equalityComparer =
12         ↪ EqualityComparer<TLink>.Default;
13
14         [MethodImpl(MethodImplOptions.AggressiveInlining)]
15         public LinksItselfConstantToSelfReferenceResolver(ILinks<TLink> links) : base(links) { }
16
17         [MethodImpl(MethodImplOptions.AggressiveInlining)]
18         public override TLink Each(Func<IList<TLink>, TLink> handler, IList<TLink> restrictions)
19         {
20             var constants = Constants;
21             var itselfConstant = constants.Itself;
22             var indexPartConstant = constants.IndexPart;
23             var sourcePartConstant = constants.SourcePart;
24             var targetPartConstant = constants.TargetPart;
25             var restrictionsCount = restrictions.Count;
26             if (!_equalityComparer.Equals(constants.Any, itselfConstant)
27                 && (((restrictionsCount > indexPartConstant) &&
28                 ↪ _equalityComparer.Equals(restrictions[indexPartConstant], itselfConstant))
29                 || ((restrictionsCount > sourcePartConstant) &&
30                 ↪ _equalityComparer.Equals(restrictions[sourcePartConstant], itselfConstant))
31                 || ((restrictionsCount > targetPartConstant) &&
32                 ↪ _equalityComparer.Equals(restrictions[targetPartConstant], itselfConstant))))
33             {
34                 // Itself constant is not supported for Each method right now, skipping execution
35                 return constants.Continue;
36             }
37         }
38     }
39 }

```

```

33         return Links.Each(handler, restrictions);
34     }
35
36     [MethodImpl(MethodImplOptions.AggressiveInlining)]
37     public override TLink Update(IList<TLink> restrictions, IList<TLink> substitution) =>
38     {
39         Links.Update(restrictions, Links.ResolveConstantAsSelfReference(Constants.Itself,
40             ↪ restrictions, substitution));
41     }
42 }

```

1.7 ./Platform.Data.Doublets/Decorators/LinksNonExistentDependenciesCreator.cs

```

1  using System.Collections.Generic;
2  using System.Runtime.CompilerServices;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Decorators
7  {
8      /// <remarks>
9      /// Not practical if newSource and newTarget are too big.
10     /// To be able to use practical version we should allow to create link at any specific
11     ↪ location inside ResizableDirectMemoryLinks.
12     /// This in turn will require to implement not a list of empty links, but a list of ranges
13     ↪ to store it more efficiently.
14     /// </remarks>
15     public class LinksNonExistentDependenciesCreator<TLink> : LinksDecoratorBase<TLink>
16     {
17         [MethodImpl(MethodImplOptions.AggressiveInlining)]
18         public LinksNonExistentDependenciesCreator(ILinks<TLink> links) : base(links) { }
19
20         [MethodImpl(MethodImplOptions.AggressiveInlining)]
21         public override TLink Update(IList<TLink> restrictions, IList<TLink> substitution)
22         {
23             var constants = Constants;
24             Links.EnsureCreated(substitution[constants.SourcePart],
25                 ↪ substitution[constants.TargetPart]);
26             return Links.Update(restrictions, substitution);
27         }
28     }
29 }

```

1.8 ./Platform.Data.Doublets/Decorators/LinksNullConstantToSelfReferenceResolver.cs

```

1  using System.Collections.Generic;
2  using System.Runtime.CompilerServices;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Decorators
7  {
8      public class LinksNullConstantToSelfReferenceResolver<TLink> : LinksDecoratorBase<TLink>
9      {
10         [MethodImpl(MethodImplOptions.AggressiveInlining)]
11         public LinksNullConstantToSelfReferenceResolver(ILinks<TLink> links) : base(links) { }
12
13         [MethodImpl(MethodImplOptions.AggressiveInlining)]
14         public override TLink Create(IList<TLink> restrictions) => Links.CreatePoint();
15
16         [MethodImpl(MethodImplOptions.AggressiveInlining)]
17         public override TLink Update(IList<TLink> restrictions, IList<TLink> substitution) =>
18         {
19             Links.Update(restrictions, Links.ResolveConstantAsSelfReference(Constants.Null,
20                 ↪ restrictions, substitution));
21         }
22     }
23 }

```

1.9 ./Platform.Data.Doublets/Decorators/LinksUniquenessResolver.cs

```

1  using System.Collections.Generic;
2  using System.Runtime.CompilerServices;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Decorators
7  {
8      public class LinksUniquenessResolver<TLink> : LinksDecoratorBase<TLink>
9      {
10         private static readonly EqualityComparer<TLink> _equalityComparer =
11         ↪ EqualityComparer<TLink>.Default;
12
13         [MethodImpl(MethodImplOptions.AggressiveInlining)]
14         public LinksUniquenessResolver(ILinks<TLink> links) : base(links) { }
15     }
16 }

```

```

14     [MethodImpl(MethodImplOptions.AggressiveInlining)]
15     public override TLink Update(IList<TLink> restrictions, IList<TLink> substitution)
16     {
17         var constants = Constants;
18         var newLinkAddress = Links.SearchOrDefault(substitution[constants.SourcePart],
19             ↪ substitution[constants.TargetPart]);
20         if (!_equalityComparer.Equals(newLinkAddress, default))
21         {
22             return Links.Update(restrictions, substitution);
23         }
24         return ResolveAddressChangeConflict(restrictions[constants.IndexPart],
25             ↪ newLinkAddress);
26     }
27     [MethodImpl(MethodImplOptions.AggressiveInlining)]
28     protected virtual TLink ResolveAddressChangeConflict(TLink oldLinkAddress, TLink
29     ↪ newLinkAddress)
30     {
31         if (!_equalityComparer.Equals(oldLinkAddress, newLinkAddress) &&
32             ↪ Links.Exists(oldLinkAddress))
33         {
34             Facade.Delete(oldLinkAddress);
35         }
36         return newLinkAddress;
37     }

```

1.10 ./Platform.Data.Doublets/Decorators/LinksUniquenessValidator.cs

```

1  using System.Collections.Generic;
2  using System.Runtime.CompilerServices;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Decorators
7  {
8      public class LinksUniquenessValidator<TLink> : LinksDecoratorBase<TLink>
9      {
10         [MethodImpl(MethodImplOptions.AggressiveInlining)]
11         public LinksUniquenessValidator(ILinks<TLink> links) : base(links) { }
12
13         [MethodImpl(MethodImplOptions.AggressiveInlining)]
14         public override TLink Update(IList<TLink> restrictions, IList<TLink> substitution)
15         {
16             Links.EnsureDoesNotExists(substitution[Constants.SourcePart],
17                 ↪ substitution[Constants.TargetPart]);
18             return Links.Update(restrictions, substitution);
19         }
20     }

```

1.11 ./Platform.Data.Doublets/Decorators/LinksUsagesValidator.cs

```

1  using System.Collections.Generic;
2  using System.Runtime.CompilerServices;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Decorators
7  {
8      public class LinksUsagesValidator<TLink> : LinksDecoratorBase<TLink>
9      {
10         [MethodImpl(MethodImplOptions.AggressiveInlining)]
11         public LinksUsagesValidator(ILinks<TLink> links) : base(links) { }
12
13         [MethodImpl(MethodImplOptions.AggressiveInlining)]
14         public override TLink Update(IList<TLink> restrictions, IList<TLink> substitution)
15         {
16             Links.EnsureNoUsages(restrictions[Constants.IndexPart]);
17             return Links.Update(restrictions, substitution);
18         }
19
20         [MethodImpl(MethodImplOptions.AggressiveInlining)]
21         public override void Delete(IList<TLink> restrictions)
22         {
23             var link = restrictions[Constants.IndexPart];
24             Links.EnsureNoUsages(link);
25             Links.Delete(link);

```

```

26     }
27 }
28 }

```

1.12 ./Platform.Data.Doublets/Decorators/NonNullContentsLinkDeletionResolver.cs

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Decorators
7 {
8     public class NonNullContentsLinkDeletionResolver<TLink> : LinksDecoratorBase<TLink>
9     {
10         [MethodImpl(MethodImplOptions.AggressiveInlining)]
11         public NonNullContentsLinkDeletionResolver(ILinks<TLink> links) : base(links) { }
12
13         [MethodImpl(MethodImplOptions.AggressiveInlining)]
14         public override void Delete(IList<TLink> restrictions)
15         {
16             var linkIndex = restrictions[Constants.IndexPart];
17             Links.EnforceResetValues(linkIndex);
18             Links.Delete(linkIndex);
19         }
20     }
21 }

```

1.13 ./Platform.Data.Doublets/Decorators/UInt64Links.cs

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Decorators
7 {
8     /// <summary>
9     /// Представляет объект для работы с базой данных (файлом) в формате Links (массива связей).
10    /// </summary>
11    /// <remarks>
12    /// Возможные оптимизации:
13    /// Объединение в одном поле Source и Target с уменьшением до 32 бит.
14    ///     + меньше объём БД
15    ///     - меньше производительность
16    ///     - больше ограничение на количество связей в БД)
17    /// Ленивое хранение размеров поддеревьев (расчитываемое по мере использования БД)
18    ///     + меньше объём БД
19    ///     - больше сложность
20    ///
21    /// Текущее теоретическое ограничение на индекс связи, из-за использования 5 бит в размере
22    ///     ↳ поддеревьев для AVL баланса и флагов нитей: 2 в степени(64 минус 5 равно 59 ) равно 576
23    ///     ↳ 460 752 303 423 488
24    /// Желательно реализовать поддержку переключения между деревьями и битовыми индексами
25    ///     ↳ (битовыми строками) - вариант матрицы (выстраиваемой лениво).
26    ///
27    /// Решить отключать ли проверки при компиляции под Release. Т.е. исключения будут
28    ///     ↳ выбрасываться только при #if DEBUG
29    /// </remarks>
30    public class UInt64Links : LinksDisposableDecoratorBase<ulong>
31    {
32        [MethodImpl(MethodImplOptions.AggressiveInlining)]
33        public UInt64Links(ILinks<ulong> links) : base(links) { }
34
35        [MethodImpl(MethodImplOptions.AggressiveInlining)]
36        public override ulong Create(IList<ulong> restrictions) => Links.CreatePoint();
37
38        [MethodImpl(MethodImplOptions.AggressiveInlining)]
39        public override ulong Update(IList<ulong> restrictions, IList<ulong> substitution)
40        {
41            var constants = Constants;
42            var indexPartConstant = constants.IndexPart;
43            var sourcePartConstant = constants.SourcePart;
44            var targetPartConstant = constants.TargetPart;
45            var nullConstant = constants.Null;
46            var itselfConstant = constants.Itself;
47            var existedLink = nullConstant;
48            var updatedLink = restrictions[indexPartConstant];
49            var newSource = substitution[sourcePartConstant];
50            var newTarget = substitution[targetPartConstant];
51            if (newSource != itselfConstant && newTarget != itselfConstant)

```

```

48     {
49         existedLink = Links.SearchOrDefault(newSource, newTarget);
50     }
51     if (existedLink == nullConstant)
52     {
53         var before = Links.GetLink(updatedLink);
54         if (before[sourcePartConstant] != newSource || before[targetPartConstant] !=
55             ↪ newTarget)
56         {
57             Links.Update(updatedLink, newSource == itselfConstant ? updatedLink :
58                 ↪ newSource,
59                 newTarget == itselfConstant ? updatedLink :
60                     ↪ newTarget);
61         }
62         return updatedLink;
63     }
64     else
65     {
66         return Facade.MergeAndDelete(updatedLink, existedLink);
67     }
68 }
69
70 [MethodImpl(MethodImplOptions.AggressiveInlining)]
71 public override void Delete(ICollection<ulong> restrictions)
72 {
73     var linkIndex = restrictions[Constants.IndexPart];
74     Links.EnforceResetValues(linkIndex);
75     Facade.DeleteAllUsages(linkIndex);
76     Links.Delete(linkIndex);
77 }
78 }
79
80 }

```

1.14 ./Platform.Data.Doublets/Decorators/UniLinks.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using Platform.Collections;
5  using Platform.Collections.Lists;
6  using Platform.Data.Universal;
7
8  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
9
10 namespace Platform.Data.Doublets.Decorators
11 {
12     /// <remarks>
13     /// What does empty pattern (for condition or substitution) mean? Nothing or Everything?
14     /// Now we go with nothing. And nothing is something one, but empty, and cannot be changed
15     ↪ by itself. But can cause creation (update from nothing) or deletion (update to nothing).
16     ///
17     /// TODO: Decide to change to IDoubletLinks or not to change. (Better to create
18     ↪ DefaultUniLinksBase, that contains logic itself and can be implemented using both
19     ↪ IDoubletLinks and ILinks.)
20     /// </remarks>
21     internal class UniLinks<TLink> : LinksDecoratorBase<TLink>, IUniLinks<TLink>
22     {
23         private static readonly EqualityComparer<TLink> _equalityComparer =
24             ↪ EqualityComparer<TLink>.Default;
25
26         public UniLinks(ILinks<TLink> links) : base(links) { }
27
28         private struct Transition
29         {
30             public IList<TLink> Before;
31             public IList<TLink> After;
32
33             public Transition(IList<TLink> before, IList<TLink> after)
34             {
35                 Before = before;
36                 After = after;
37             }
38         }
39
40         //public static readonly TLink NullConstant = Use<LinksConstants<TLink>>.Single.Null;
41         //public static readonly IReadOnlyList<TLink> NullLink = new
42         ↪ ReadOnlyCollection<TLink>(new List<TLink> { NullConstant, NullConstant, NullConstant
43         ↪ });
44
45         // TODO: Подумать о том, как реализовать древовидный Restriction и Substitution
46         ↪ (Links-Expression)

```

```

40 public TLink Trigger(IList<TLink> restriction, Func<IList<TLink>, IList<TLink>, TLink>
    ↳ matchedHandler, IList<TLink> substitution, Func<IList<TLink>, IList<TLink>, TLink>
    ↳ substitutedHandler)
41 {
42     /////List<Transition> transitions = null;
43     /////if (!restriction.IsNullOrEmpty())
44     /////{
45     /////    // Есть причина делать проход (чтение)
46     /////    if (matchedHandler != null)
47     /////    {
48     /////        if (!substitution.IsNullOrEmpty())
49     /////        {
50     /////            // restriction => { 0, 0, 0 } | { 0 } // Create
51     /////            // substitution => { itself, 0, 0 } | { itself, itself, itself } //
    ↳ Create / Update
52     /////            // substitution => { 0, 0, 0 } | { 0 } // Delete
53     /////            transitions = new List<Transition>();
54     /////            if (Equals(substitution[Constants.IndexPart], Constants.Null))
55     /////            {
56     /////                // If index is Null, that means we always ignore every other
    ↳ value (they are also Null by definition)
57     /////                var matchDecision = matchedHandler(, NullLink);
58     /////                if (Equals(matchDecision, Constants.Break))
59     /////                    return false;
60     /////                if (!Equals(matchDecision, Constants.Skip))
61     /////                    transitions.Add(new Transition(matchedLink, newValue));
62     /////            }
63     /////            else
64     /////            {
65     /////                Func<T, bool> handler;
66     /////                handler = link =>
67     /////                {
68     /////                    var matchedLink = Memory.GetLinkValue(link);
69     /////                    var newValue = Memory.GetLinkValue(link);
70     /////                    newValue[Constants.IndexPart] = Constants.Itself;
71     /////                    newValue[Constants.SourcePart] =
    ↳ Equals(substitution[Constants.SourcePart], Constants.Itself) ?
    ↳ matchedLink[Constants.IndexPart] : substitution[Constants.SourcePart];
72     /////                    newValue[Constants.TargetPart] =
    ↳ Equals(substitution[Constants.TargetPart], Constants.Itself) ?
    ↳ matchedLink[Constants.IndexPart] : substitution[Constants.TargetPart];
73     /////                    var matchDecision = matchedHandler(matchedLink, newValue);
74     /////                    if (Equals(matchDecision, Constants.Break))
75     /////                        return false;
76     /////                    if (!Equals(matchDecision, Constants.Skip))
77     /////                        transitions.Add(new Transition(matchedLink, newValue));
78     /////                    return true;
79     /////                };
80     /////                if (!Memory.Each(handler, restriction))
81     /////                    return Constants.Break;
82     /////            }
83     /////        }
84     /////        else
85     /////        {
86     /////            Func<T, bool> handler = link =>
87     /////            {
88     /////                var matchedLink = Memory.GetLinkValue(link);
89     /////                var matchDecision = matchedHandler(matchedLink, matchedLink);
90     /////                return !Equals(matchDecision, Constants.Break);
91     /////            };
92     /////            if (!Memory.Each(handler, restriction))
93     /////                return Constants.Break;
94     /////        }
95     /////    }
96     /////    else
97     /////    {
98     /////        if (substitution != null)
99     /////        {
100     /////            transitions = new List<IList<T>>();
101     /////            Func<T, bool> handler = link =>
102     /////            {
103     /////                var matchedLink = Memory.GetLinkValue(link);
104     /////                transitions.Add(matchedLink);
105     /////                return true;
106     /////            };
107     /////            if (!Memory.Each(handler, restriction))
108     /////                return Constants.Break;

```



```

109         ///}
110         ///} else
111         ///{
112         ///    return Constants.Continue;
113         ///}
114     ///}
115     ///}
116     ///if (substitution != null)
117     ///{
118     ///    // Есть причина делать замену (запись)
119     ///    if (substitutedHandler != null)
120     ///    {
121     ///    }
122     ///    else
123     ///    {
124     ///    }
125     ///}
126     ///return Constants.Continue;
127
128     ///if (restriction.IsNullOrEmpty()) // Create
129     ///{
130     ///    substitution[Constants.IndexPart] = Memory.AllocateLink();
131     ///    Memory.SetLinkValue(substitution);
132     ///}
133     ///else if (substitution.IsNullOrEmpty()) // Delete
134     ///{
135     ///    Memory.FreeLink(restriction[Constants.IndexPart]);
136     ///}
137     ///else if (restriction.EqualTo(substitution)) // Read or ("repeat" the state) // Each
138     ///{
139     ///    // No need to collect links to list
140     ///    // Skip == Continue
141     ///    // No need to check substitutedHandler
142     ///    if (!Memory.Each(link => !Equals(matchedHandler(Memory.GetLinkValue(link)),
143     ↪ Constants.Break), restriction))
144     ///    return Constants.Break;
145     ///}
146     ///else // Update
147     ///{
148     ///    //List<IList<T>> matchedLinks = null;
149     ///    if (matchedHandler != null)
150     ///    {
151     ///        matchedLinks = new List<IList<T>>>();
152     ///        Func<T, bool> handler = link =>
153     ///        {
154     ///            var matchedLink = Memory.GetLinkValue(link);
155     ///            var matchDecision = matchedHandler(matchedLink);
156     ///            if (Equals(matchDecision, Constants.Break))
157     ///                return false;
158     ///            if (!Equals(matchDecision, Constants.Skip))
159     ///                matchedLinks.Add(matchedLink);
160     ///            return true;
161     ///        };
162     ///        if (!Memory.Each(handler, restriction))
163     ///            return Constants.Break;
164     ///    }
165     ///    if (!matchedLinks.IsNullOrEmpty())
166     ///    {
167     ///        var totalMatchedLinks = matchedLinks.Count;
168     ///        for (var i = 0; i < totalMatchedLinks; i++)
169     ///        {
170     ///            var matchedLink = matchedLinks[i];
171     ///            if (substitutedHandler != null)
172     ///            {
173     ///                var newValue = new List<T>(); // TODO: Prepare value to update here
174     ///                // TODO: Decide is it actually needed to use Before and After
175     ↪ substitution handling.
176     ///                var substitutedDecision = substitutedHandler(matchedLink,
177     ↪ newValue);
178     ///                if (Equals(substitutedDecision, Constants.Break))
179     ///                    return Constants.Break;
180     ///                if (Equals(substitutedDecision, Constants.Continue))
181     ///                {
182     ///                    // Actual update here
183     ///                    Memory.SetLinkValue(newValue);
184     ///                }
185     ///                if (Equals(substitutedDecision, Constants.Skip))

```

```

183         //          {
184         //          // Cancel the update. TODO: decide use separate Cancel
185         //          constant or Skip is enough?
186         //          }
187         //      }
188         //  }
189     //}
190     return Constants.Continue;
191 }
192
193 public TLink Trigger(IList<TLink> patternOrCondition, Func<IList<TLink>, TLink>
194     ↪ matchHandler, IList<TLink> substitution, Func<IList<TLink>, IList<TLink>, TLink>
195     ↪ substitutionHandler)
196 {
197     if (patternOrCondition.IsNullOrEmpty() && substitution.IsNullOrEmpty())
198     {
199         return Constants.Continue;
200     }
201     else if (patternOrCondition.EqualTo(substitution)) // Should be Each here TODO:
202     ↪ Check if it is a correct condition
203     {
204         // Or it only applies to trigger without matchHandler.
205         throw new NotImplementedException();
206     }
207     else if (!substitution.IsNullOrEmpty()) // Creation
208     {
209         var before = Array.Empty<TLink>();
210         // Что должно означать False здесь? Остановиться (перестать идти) или пропустить
211         ↪ (пройти мимо) или пустить (взять)?
212         if (matchHandler != null && _equalityComparer.Equals(matchHandler(before),
213             ↪ Constants.Break))
214         {
215             return Constants.Break;
216         }
217         var after = (IList<TLink>)substitution.ToArray();
218         if (_equalityComparer.Equals(after[0], default))
219         {
220             var newLink = Links.Create();
221             after[0] = newLink;
222         }
223         if (substitution.Count == 1)
224         {
225             after = Links.GetLink(substitution[0]);
226         }
227         else if (substitution.Count == 3)
228         {
229             //Links.Create(after);
230         }
231         else
232         {
233             throw new NotSupportedException();
234         }
235         if (matchHandler != null)
236         {
237             return substitutionHandler(before, after);
238         }
239         return Constants.Continue;
240     }
241     else if (!patternOrCondition.IsNullOrEmpty()) // Deletion
242     {
243         if (patternOrCondition.Count == 1)
244         {
245             var linkToDelete = patternOrCondition[0];
246             var before = Links.GetLink(linkToDelete);
247             if (matchHandler != null && _equalityComparer.Equals(matchHandler(before),
248                 ↪ Constants.Break))
249             {
250                 return Constants.Break;
251             }
252             var after = Array.Empty<TLink>();
253             Links.Update(linkToDelete, Constants.Null, Constants.Null);
254             Links.Delete(linkToDelete);
255             if (matchHandler != null)
256             {
257                 return substitutionHandler(before, after);
258             }
259             return Constants.Continue;
260         }
261     }
262 }

```

```

254     }
255     else
256     {
257         throw new NotSupportedException();
258     }
259 }
260 else // Replace / Update
261 {
262     if (patternOrCondition.Count == 1) //-V3125
263     {
264         var linkToUpdate = patternOrCondition[0];
265         var before = Links.GetLink(linkToUpdate);
266         if (matchHandler != null && _equalityComparer.Equals(matchHandler(before),
267             ↪ Constants.Break))
268         {
269             return Constants.Break;
270         }
271         var after = (IList<TLink>)substitution.ToArray(); //-V3125
272         if (_equalityComparer.Equals(after[0], default))
273         {
274             after[0] = linkToUpdate;
275         }
276         if (substitution.Count == 1)
277         {
278             if (!_equalityComparer.Equals(substitution[0], linkToUpdate))
279             {
280                 after = Links.GetLink(substitution[0]);
281                 Links.Update(linkToUpdate, Constants.Null, Constants.Null);
282                 Links.Delete(linkToUpdate);
283             }
284         }
285         else if (substitution.Count == 3)
286         {
287             //Links.Update(after);
288         }
289         else
290         {
291             throw new NotSupportedException();
292         }
293         if (matchHandler != null)
294         {
295             return substitutionHandler(before, after);
296         }
297         return Constants.Continue;
298     }
299     else
300     {
301         throw new NotSupportedException();
302     }
303 }
304
305 /// <remarks>
306 /// IList[IList[IList[T]]]
307 /// |         |         |         ||
308 /// |         |         ----- ||
309 /// |         |         link    ||
310 /// |         |         ----- |
311 /// |         change          |
312 /// -----
313 /// changes
314 /// </remarks>
315 public IList<IList<IList<TLink>>> Trigger(IList<TLink> condition, IList<TLink>
316 ↪ substitution)
317 {
318     var changes = new List<IList<IList<TLink>>>();
319     Trigger(condition, AlwaysContinue, substitution, (before, after) =>
320     {
321         var change = new[] { before, after };
322         changes.Add(change);
323         return Constants.Continue;
324     });
325     return changes;
326 }
327 private TLink AlwaysContinue(IList<TLink> linkToMatch) => Constants.Continue;
328 }
329 }

```

1.15 ./Platform.Data.Doublets/Doublet.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets
8  {
9      public struct Doublet<T> : IEquatable<Doublet<T>>
10     {
11         private static readonly EqualityComparer<T> _equalityComparer =
12             ↳ EqualityComparer<T>.Default;
13
14         public T Source
15         {
16             [MethodImpl(MethodImplOptions.AggressiveInlining)]
17             get;
18             [MethodImpl(MethodImplOptions.AggressiveInlining)]
19             set;
20         }
21         public T Target
22         {
23             [MethodImpl(MethodImplOptions.AggressiveInlining)]
24             get;
25             [MethodImpl(MethodImplOptions.AggressiveInlining)]
26             set;
27         }
28
29         [MethodImpl(MethodImplOptions.AggressiveInlining)]
30         public Doublet(T source, T target)
31         {
32             Source = source;
33             Target = target;
34         }
35
36         [MethodImpl(MethodImplOptions.AggressiveInlining)]
37         public override string ToString() => $"{Source}->{Target}";
38
39         [MethodImpl(MethodImplOptions.AggressiveInlining)]
40         public bool Equals(Doublet<T> other) => _equalityComparer.Equals(Source, other.Source)
41             ↳ && _equalityComparer.Equals(Target, other.Target);
42
43         [MethodImpl(MethodImplOptions.AggressiveInlining)]
44         public override bool Equals(object obj) => obj is Doublet<T> doublet ?
45             ↳ base.Equals(doublet) : false;
46
47         [MethodImpl(MethodImplOptions.AggressiveInlining)]
48         public override int GetHashCode() => (Source, Target).GetHashCode();
49
50         [MethodImpl(MethodImplOptions.AggressiveInlining)]
51         public static bool operator ==(Doublet<T> left, Doublet<T> right) => left.Equals(right);
52
53         [MethodImpl(MethodImplOptions.AggressiveInlining)]
54         public static bool operator !=(Doublet<T> left, Doublet<T> right) => !(left == right);
55     }
56 }

```

1.16 ./Platform.Data.Doublets/DoubletComparer.cs

```

1  using System.Collections.Generic;
2  using System.Runtime.CompilerServices;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets
7  {
8      /// <remarks>
9      /// TODO: Может стоит попробовать ref во всех методах (IRefEqualityComparer)
10     /// 2x faster with comparer
11     /// </remarks>
12     public class DoubletComparer<T> : IEquityComparer<Doublet<T>>
13     {
14         public static readonly DoubletComparer<T> Default = new DoubletComparer<T>();
15
16         [MethodImpl(MethodImplOptions.AggressiveInlining)]
17         public bool Equals(Doublet<T> x, Doublet<T> y) => x.Equals(y);
18
19         [MethodImpl(MethodImplOptions.AggressiveInlining)]
20         public int GetHashCode(Doublet<T> obj) => obj.GetHashCode();
21     }
22 }

```

```
22 }
```

1.17 ./Platform.Data.Doublets/ILinks.cs

```
1  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
2
3  using System.Collections.Generic;
4
5  namespace Platform.Data.Doublets
6  {
7      public interface ILinks<TLink> : ILinks<TLink, LinksConstants<TLink>>
8      {
9      }
10 }
```

1.18 ./Platform.Data.Doublets/ILinksExtensions.cs

```
1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Runtime.CompilerServices;
6  using Platform.Ranges;
7  using Platform.Collections.Arrays;
8  using Platform.Random;
9  using Platform.Setters;
10 using Platform.Converters;
11 using Platform.Numbers;
12 using Platform.Data.Exceptions;
13 using Platform.Data.Doublets.Decorators;
14
15 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
16
17 namespace Platform.Data.Doublets
18 {
19     public static class ILinksExtensions
20     {
21         [MethodImpl(MethodImplOptions.AggressiveInlining)]
22         public static void RunRandomCreations<TLink>(this ILinks<TLink> links, ulong
23             ↳ amountOfCreations)
24         {
25             var random = RandomHelpers.Default;
26             var addressToUInt64Converter = UncheckedConverter<TLink, ulong>.Default;
27             var uInt64ToAddressConverter = UncheckedConverter<ulong, TLink>.Default;
28             for (var i = 0UL; i < amountOfCreations; i++)
29             {
30                 var linksAddressRange = new Range<ulong>(0,
31                     ↳ addressToUInt64Converter.Convert(links.Count()));
32                 var source =
33                     ↳ uInt64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
34                 var target =
35                     ↳ uInt64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
36                 links.GetOrCreate(source, target);
37             }
38
39             [MethodImpl(MethodImplOptions.AggressiveInlining)]
40             public static void RunRandomSearches<TLink>(this ILinks<TLink> links, ulong
41                 ↳ amountOfSearches)
42             {
43                 var random = RandomHelpers.Default;
44                 var addressToUInt64Converter = UncheckedConverter<TLink, ulong>.Default;
45                 var uInt64ToAddressConverter = UncheckedConverter<ulong, TLink>.Default;
46                 for (var i = 0UL; i < amountOfSearches; i++)
47                 {
48                     var linksAddressRange = new Range<ulong>(0,
49                         ↳ addressToUInt64Converter.Convert(links.Count()));
50                     var source =
51                         ↳ uInt64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
52                     var target =
53                         ↳ uInt64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
54                     links.SearchOrDefault(source, target);
55                 }
56
57                 [MethodImpl(MethodImplOptions.AggressiveInlining)]
58                 public static void RunRandomDeletions<TLink>(this ILinks<TLink> links, ulong
59                     ↳ amountOfDeletions)
60                 {
61                     var random = RandomHelpers.Default;
62                     var addressToUInt64Converter = UncheckedConverter<TLink, ulong>.Default;
63                     var uInt64ToAddressConverter = UncheckedConverter<ulong, TLink>.Default;
```

```

57     var linksCount = addressToUInt64Converter.Convert(links.Count());
58     var min = amountOfDeletions > linksCount ? OUL : linksCount - amountOfDeletions;
59     for (var i = OUL; i < amountOfDeletions; i++)
60     {
61         linksCount = addressToUInt64Converter.Convert(links.Count());
62         if (linksCount <= min)
63         {
64             break;
65         }
66         var linksAddressRange = new Range<ulong>(min, linksCount);
67         var link =
68             ↪ uInt64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
69         links.Delete(link);
70     }
71
72     [MethodImpl(MethodImplOptions.AggressiveInlining)]
73     public static void Delete<TLink>(this ILinks<TLink> links, TLink linkToDelete) =>
74         ↪ links.Delete(new LinkAddress<TLink>(linkToDelete));
75
76     /// <remarks>
77     /// TODO: Возможно есть очень простой способ это сделать.
78     /// (Например просто удалить файл, или изменить его размер таким образом,
79     /// чтобы удалился весь контент)
80     /// Например через _header->AllocatedLinks в ResizableDirectMemoryLinks
81     /// </remarks>
82     [MethodImpl(MethodImplOptions.AggressiveInlining)]
83     public static void DeleteAll<TLink>(this ILinks<TLink> links)
84     {
85         var equalityComparer = EqualityComparer<TLink>.Default;
86         var comparer = Comparer<TLink>.Default;
87         for (var i = links.Count(); comparer.Compare(i, default) > 0; i =
88             ↪ Arithmetic.Decrement(i))
89         {
90             links.Delete(i);
91             if (!equalityComparer.Equals(links.Count(), Arithmetic.Decrement(i)))
92             {
93                 i = links.Count();
94             }
95         }
96     }
97
98     [MethodImpl(MethodImplOptions.AggressiveInlining)]
99     public static TLink First<TLink>(this ILinks<TLink> links)
100     {
101         TLink firstLink = default;
102         var equalityComparer = EqualityComparer<TLink>.Default;
103         if (equalityComparer.Equals(links.Count(), default))
104         {
105             throw new InvalidOperationException("В хранилище нет связей.");
106         }
107         links.Each(links.Constants.Any, links.Constants.Any, link =>
108         {
109             firstLink = link[links.Constants.IndexPart];
110             return links.Constants.Break;
111         });
112         if (equalityComparer.Equals(firstLink, default))
113         {
114             throw new InvalidOperationException("В процессе поиска по хранилищу не было
115                 ↪ найдено связей.");
116         }
117         return firstLink;
118     }
119
120     #region Paths
121
122     /// <remarks>
123     /// TODO: Как так? Как то что ниже может быть корректно?
124     /// Скорее всего практически не применимо
125     /// Предполагалось, что можно было конвертировать формируемый в проходе через
126     /// ↪ SequenceWalker
127     /// Stack в конкретный путь из Source, Target до связи, но это не всегда так.
128     /// TODO: Возможно нужен метод, который именно выбрасывает исключения (EnsurePathExists)
129     /// </remarks>
130     [MethodImpl(MethodImplOptions.AggressiveInlining)]
131     public static bool CheckPathExistance<TLink>(this ILinks<TLink> links, params TLink[]
132         ↪ path)
133     {
134

```

```

129     var current = path[0];
130     //EnsureLinkExists(current, "path");
131     if (!links.Exists(current))
132     {
133         return false;
134     }
135     var equalityComparer = EqualityComparer<TLink>.Default;
136     var constants = links.Constants;
137     for (var i = 1; i < path.Length; i++)
138     {
139         var next = path[i];
140         var values = links.GetLink(current);
141         var source = values[constants.SourcePart];
142         var target = values[constants.TargetPart];
143         if (equalityComparer.Equals(source, target) && equalityComparer.Equals(source,
144             ↪ next))
145         {
146             //throw new InvalidOperationException(string.Format("Невозможно выбрать
147             ↪ путь, так как и Source и Target совпадают с элементом пути {0}.", next));
148             return false;
149         }
150         if (!equalityComparer.Equals(next, source) && !equalityComparer.Equals(next,
151             ↪ target))
152         {
153             //throw new InvalidOperationException(string.Format("Невозможно продолжить
154             ↪ путь через элемент пути {0}", next));
155             return false;
156         }
157         current = next;
158     }
159     return true;
160 }
161
162 /// <remarks>
163 /// Может потребовать дополнительного стека для PathElement's при использовании
164 ↪ SequenceWalker.
165 /// </remarks>
166 [MethodImpl(MethodImplOptions.AggressiveInlining)]
167 public static TLink GetByKeyes<TLink>(this ILinks<TLink> links, TLink root, params int[])
168 ↪ path)
169 {
170     links.EnsureLinkExists(root, "root");
171     var currentLink = root;
172     for (var i = 0; i < path.Length; i++)
173     {
174         currentLink = links.GetLink(currentLink)[path[i]];
175     }
176     return currentLink;
177 }
178
179 [MethodImpl(MethodImplOptions.AggressiveInlining)]
180 public static TLink GetSquareMatrixSequenceElementByIndex<TLink>(this ILinks<TLink>
181 ↪ links, TLink root, ulong size, ulong index)
182 {
183     var constants = links.Constants;
184     var source = constants.SourcePart;
185     var target = constants.TargetPart;
186     if (!Platform.Numbers.Math.IsPowerOfTwo(size))
187     {
188         throw new ArgumentOutOfRangeException(nameof(size), "Sequences with sizes other
189         ↪ than powers of two are not supported.");
190     }
191     var path = new BitArray(BitConverter.GetBytes(index));
192     var length = Bit.GetLowestPosition(size);
193     links.EnsureLinkExists(root, "root");
194     var currentLink = root;
195     for (var i = length - 1; i >= 0; i--)
196     {
197         currentLink = links.GetLink(currentLink)[path[i] ? target : source];
198     }
199     return currentLink;
200 }
201
202 #endregion
203
204 /// <summary>
205 /// Возвращает индекс указанной связи.
206 /// </summary>
207 /// <param name="links">Хранилище связей.</param>

```

```

200 /// <param name="link">Связь представленная списком, состоящим из её адреса и
201   ↳ содержимого.</param>
202 /// <returns>Индекс начальной связи для указанной связи.</returns>
203 [MethodImpl(MethodImplOptions.AggressiveInlining)]
204 public static TLink GetIndex<TLink>(this ILinks<TLink> links, IList<TLink> link) =>
205   ↳ link[links.Constants.IndexPart];
206
207 /// <summary>
208 /// Возвращает индекс начальной (Source) связи для указанной связи.
209 /// </summary>
210 /// <param name="links">Хранилище связей.</param>
211 /// <param name="link">Индекс связи.</param>
212 /// <returns>Индекс начальной связи для указанной связи.</returns>
213 [MethodImpl(MethodImplOptions.AggressiveInlining)]
214 public static TLink GetSource<TLink>(this ILinks<TLink> links, TLink link) =>
215   ↳ links.GetLink(link)[links.Constants.SourcePart];
216
217 /// <summary>
218 /// Возвращает индекс начальной (Source) связи для указанной связи.
219 /// </summary>
220 /// <param name="links">Хранилище связей.</param>
221 /// <param name="link">Связь представленная списком, состоящим из её адреса и
222   ↳ содержимого.</param>
223 /// <returns>Индекс начальной связи для указанной связи.</returns>
224 [MethodImpl(MethodImplOptions.AggressiveInlining)]
225 public static TLink GetSource<TLink>(this ILinks<TLink> links, IList<TLink> link) =>
226   ↳ link[links.Constants.SourcePart];
227
228 /// <summary>
229 /// Возвращает индекс конечной (Target) связи для указанной связи.
230 /// </summary>
231 /// <param name="links">Хранилище связей.</param>
232 /// <param name="link">Индекс связи.</param>
233 /// <returns>Индекс конечной связи для указанной связи.</returns>
234 [MethodImpl(MethodImplOptions.AggressiveInlining)]
235 public static TLink GetTarget<TLink>(this ILinks<TLink> links, TLink link) =>
236   ↳ links.GetLink(link)[links.Constants.TargetPart];
237
238 /// <summary>
239 /// Возвращает индекс конечной (Target) связи для указанной связи.
240 /// </summary>
241 /// <param name="links">Хранилище связей.</param>
242 /// <param name="link">Связь представленная списком, состоящим из её адреса и
243   ↳ содержимого.</param>
244 /// <returns>Индекс конечной связи для указанной связи.</returns>
245 [MethodImpl(MethodImplOptions.AggressiveInlining)]
246 public static TLink GetTarget<TLink>(this ILinks<TLink> links, IList<TLink> link) =>
247   ↳ link[links.Constants.TargetPart];
248
249 /// <summary>
250 /// Выполняет проход по всем связям, соответствующим шаблону, вызывая обработчик
251   ↳ (handler) для каждой подходящей связи.
252 /// </summary>
253 /// <param name="links">Хранилище связей.</param>
254 /// <param name="handler">Обработчик каждой подходящей связи.</param>
255 /// <param name="restrictions">Ограничения на содержимое связей. Каждое ограничение
256   ↳ может иметь значения: Constants.Null - 0-я связь, обозначающая ссылку на пустоту,
257   ↳ Any - отсутствие ограничения, 1..∞ конкретный адрес связи.</param>
258 /// <returns>True, в случае если проход по связям не был прерван и False в обратном
259   ↳ случае.</returns>
260 [MethodImpl(MethodImplOptions.AggressiveInlining)]
261 public static bool Each<TLink>(this ILinks<TLink> links, Func<IList<TLink>, TLink>
262   ↳ handler, params TLink[] restrictions)
263   => EqualityComparer<TLink>.Default.Equals(links.Each(handler, restrictions),
264     ↳ links.Constants.Continue);
265
266 /// <summary>
267 /// Выполняет проход по всем связям, соответствующим шаблону, вызывая обработчик
268   ↳ (handler) для каждой подходящей связи.
269 /// </summary>
270 /// <param name="links">Хранилище связей.</param>
271 /// <param name="source">Значение, определяющее соответствующие шаблону связи.
272   ↳ (Constants.Null - 0-я связь, обозначающая ссылку на пустоту в качестве начала,
273   ↳ Constants.Any - любое начало, 1..∞ конкретное начало)</param>
274 /// <param name="target">Значение, определяющее соответствующие шаблону связи.
275   ↳ (Constants.Null - 0-я связь, обозначающая ссылку на пустоту в качестве конца,
276   ↳ Constants.Any - любой конец, 1..∞ конкретный конец)</param>

```



```

258 /// <param name="handler">Обработчик каждой подходящей связи.</param>
259 /// <returns>True, в случае если проход по связям не был прерван и False в обратном
    ↳ случае.</returns>
260 [MethodImpl(MethodImplOptions.AggressiveInlining)]
261 public static bool Each<TLink>(this ILinks<TLink> links, TLink source, TLink target,
    ↳ Func<TLink, bool> handler)
262 {
263     var constants = links.Constants;
264     return links.Each(link => handler(link[constants.IndexPart]) ? constants.Continue :
    ↳ constants.Break, constants.Any, source, target);
265 }
266
267 /// <summary>
268 /// Выполняет проход по всем связям, соответствующим шаблону, вызывая обработчик
    ↳ (handler) для каждой подходящей связи.
269 /// </summary>
270 /// <param name="links">Хранилище связей.</param>
271 /// <param name="source">Значение, определяющее соответствующие шаблону связи.
    ↳ (Constants.Null - 0-я связь, обозначающая ссылку на пустоту в качестве начала,
    ↳ Constants.Any - любое начало, 1..∞ конкретное начало)</param>
272 /// <param name="target">Значение, определяющее соответствующие шаблону связи.
    ↳ (Constants.Null - 0-я связь, обозначающая ссылку на пустоту в качестве конца,
    ↳ Constants.Any - любой конец, 1..∞ конкретный конец)</param>
273 /// <param name="handler">Обработчик каждой подходящей связи.</param>
274 /// <returns>True, в случае если проход по связям не был прерван и False в обратном
    ↳ случае.</returns>
275 [MethodImpl(MethodImplOptions.AggressiveInlining)]
276 public static bool Each<TLink>(this ILinks<TLink> links, TLink source, TLink target,
    ↳ Func<IList<TLink>, TLink> handler) => links.Each(handler, links.Constants.Any,
    ↳ source, target);
277
278 [MethodImpl(MethodImplOptions.AggressiveInlining)]
279 public static IList<IList<TLink>> All<TLink>(this ILinks<TLink> links, params TLink[]
    ↳ restrictions)
280 {
281     var arraySize = CheckedConverter<TLink,
    ↳ long>.Default.Convert(links.Count(restrictions));
282     if (arraySize > 0)
283     {
284         var array = new IList<TLink>[arraySize];
285         var filler = new ArrayFiller<IList<TLink>, TLink>(array,
    ↳ links.Constants.Continue);
286         links.Each(filler.AddAndReturnConstant, restrictions);
287         return array;
288     }
289     else
290     {
291         return Array.Empty<IList<TLink>>();
292     }
293 }
294
295 [MethodImpl(MethodImplOptions.AggressiveInlining)]
296 public static IList<TLink> AllIndices<TLink>(this ILinks<TLink> links, params TLink[]
    ↳ restrictions)
297 {
298     var arraySize = CheckedConverter<TLink,
    ↳ long>.Default.Convert(links.Count(restrictions));
299     if (arraySize > 0)
300     {
301         var array = new TLink[arraySize];
302         var filler = new ArrayFiller<TLink, TLink>(array, links.Constants.Continue);
303         links.Each(filler.AddFirstAndReturnConstant, restrictions);
304         return array;
305     }
306     else
307     {
308         return Array.Empty<TLink>();
309     }
310 }
311
312 /// <summary>
313 /// Возвращает значение, определяющее существует ли связь с указанными началом и концом
    ↳ в хранилище связей.
314 /// </summary>
315 /// <param name="links">Хранилище связей.</param>
316 /// <param name="source">Начало связи.</param>
317 /// <param name="target">Конец связи.</param>

```

```

318 /// <returns>Значение, определяющее существует ли связь.</returns>
319 [MethodImpl(MethodImplOptions.AggressiveInlining)]
320 public static bool Exists<TLink>(this ILinks<TLink> links, TLink source, TLink target)
    ↳ => Comparer<TLink>.Default.Compare(links.Count(links.Constants.Any, source, target),
    ↳ default) > 0;

321 #region Ensure
322 // TODO: May be move to EnsureExtensions or make it both there and here
323
324 [MethodImpl(MethodImplOptions.AggressiveInlining)]
325 public static void EnsureLinkExists<TLink>(this ILinks<TLink> links, IList<TLink>
    ↳ restrictions)
326 {
327     for (var i = 0; i < restrictions.Count; i++)
328     {
329         if (!links.Exists(restrictions[i]))
330         {
331             throw new ArgumentLinkDoesNotExistsException<TLink>(restrictions[i],
332                 ↳ $"sequence[{i}]");
333         }
334     }
335 }
336
337 [MethodImpl(MethodImplOptions.AggressiveInlining)]
338 public static void EnsureInnerReferenceExists<TLink>(this ILinks<TLink> links, TLink
    ↳ reference, string argumentName)
339 {
340     if (links.Constants.IsInternalReference(reference) && !links.Exists(reference))
341     {
342         throw new ArgumentLinkDoesNotExistsException<TLink>(reference, argumentName);
343     }
344 }
345
346 [MethodImpl(MethodImplOptions.AggressiveInlining)]
347 public static void EnsureInnerReferenceExists<TLink>(this ILinks<TLink> links,
    ↳ IList<TLink> restrictions, string argumentName)
348 {
349     for (int i = 0; i < restrictions.Count; i++)
350     {
351         links.EnsureInnerReferenceExists(restrictions[i], argumentName);
352     }
353 }
354
355 [MethodImpl(MethodImplOptions.AggressiveInlining)]
356 public static void EnsureLinkIsAnyOrExists<TLink>(this ILinks<TLink> links, IList<TLink>
    ↳ restrictions)
357 {
358     var equalityComparer = EqualityComparer<TLink>.Default;
359     var any = links.Constants.Any;
360     for (var i = 0; i < restrictions.Count; i++)
361     {
362         if (!equalityComparer.Equals(restrictions[i], any) &&
363             ↳ !links.Exists(restrictions[i]))
364         {
365             throw new ArgumentLinkDoesNotExistsException<TLink>(restrictions[i],
366                 ↳ $"sequence[{i}]");
367         }
368     }
369 }
370
371 [MethodImpl(MethodImplOptions.AggressiveInlining)]
372 public static void EnsureLinkIsAnyOrExists<TLink>(this ILinks<TLink> links, TLink link,
    ↳ string argumentName)
373 {
374     var equalityComparer = EqualityComparer<TLink>.Default;
375     if (!equalityComparer.Equals(link, links.Constants.Any) && !links.Exists(link))
376     {
377         throw new ArgumentLinkDoesNotExistsException<TLink>(link, argumentName);
378     }
379 }
380
381 [MethodImpl(MethodImplOptions.AggressiveInlining)]
382 public static void EnsureLinkIsItselfOrExists<TLink>(this ILinks<TLink> links, TLink
    ↳ link, string argumentName)
383 {
384     var equalityComparer = EqualityComparer<TLink>.Default;
385     if (!equalityComparer.Equals(link, links.Constants.Itself) && !links.Exists(link))

```

```

384     {
385         throw new ArgumentLinkDoesNotExistsException<TLink>(link, argumentName);
386     }
387 }
388
389 /// <param name="links">Хранилище связей.</param>
390 [MethodImpl(MethodImplOptions.AggressiveInlining)]
391 public static void EnsureDoesNotExists<TLink>(this ILinks<TLink> links, TLink source,
    ↪ TLink target)
392 {
393     if (links.Exists(source, target))
394     {
395         throw new LinkWithSameValueAlreadyExistsException();
396     }
397 }
398
399 /// <param name="links">Хранилище связей.</param>
400 [MethodImpl(MethodImplOptions.AggressiveInlining)]
401 public static void EnsureNoUsages<TLink>(this ILinks<TLink> links, TLink link)
402 {
403     if (links.HasUsages(link))
404     {
405         throw new ArgumentLinkHasDependenciesException<TLink>(link);
406     }
407 }
408
409 /// <param name="links">Хранилище связей.</param>
410 [MethodImpl(MethodImplOptions.AggressiveInlining)]
411 public static void EnsureCreated<TLink>(this ILinks<TLink> links, params TLink[]
    ↪ addresses) => links.EnsureCreated(links.Create, addresses);
412
413 /// <param name="links">Хранилище связей.</param>
414 [MethodImpl(MethodImplOptions.AggressiveInlining)]
415 public static void EnsurePointsCreated<TLink>(this ILinks<TLink> links, params TLink[]
    ↪ addresses) => links.EnsureCreated(links.CreatePoint, addresses);
416
417 /// <param name="links">Хранилище связей.</param>
418 [MethodImpl(MethodImplOptions.AggressiveInlining)]
419 public static void EnsureCreated<TLink>(this ILinks<TLink> links, Func<TLink> creator,
    ↪ params TLink[] addresses)
420 {
421     var addressToUInt64Converter = CheckedConverter<TLink, ulong>.Default;
422     var uInt64ToAddressConverter = CheckedConverter<ulong, TLink>.Default;
423     var nonExistentAddresses = new HashSet<TLink>(addresses.Where(x =>
    ↪ !links.Exists(x)));
424     if (nonExistentAddresses.Count > 0)
425     {
426         var max = nonExistentAddresses.Max();
427         max = uInt64ToAddressConverter.Convert(System.Math.Min(addressToUInt64Converter.
    ↪ Convert(max),
    ↪ addressToUInt64Converter.Convert(links.Constants.InternalReferencesRange.Max
    ↪ imum)));
428         var createdLinks = new List<TLink>();
429         var equalityComparer = EqualityComparer<TLink>.Default;
430         TLink createdLink = creator();
431         while (!equalityComparer.Equals(createdLink, max))
432         {
433             createdLinks.Add(createdLink);
434         }
435         for (var i = 0; i < createdLinks.Count; i++)
436         {
437             if (!nonExistentAddresses.Contains(createdLinks[i]))
438             {
439                 links.Delete(createdLinks[i]);
440             }
441         }
442     }
443 }
444
445 #endregion
446
447 /// <param name="links">Хранилище связей.</param>
448 [MethodImpl(MethodImplOptions.AggressiveInlining)]
449 public static TLink CountUsages<TLink>(this ILinks<TLink> links, TLink link)
450 {
451     var constants = links.Constants;
452     var values = links.GetLink(link);

```

```

453     TLink usagesAsSource = links.Count(new Link<TLink>(constants.Any, link,
454         ↪ constants.Any));
455     var equalityComparer = EqualityComparer<TLink>.Default;
456     if (equalityComparer.Equals(values[constants.SourcePart], link))
457     {
458         usagesAsSource = Arithmetic<TLink>.Decrement(usagesAsSource);
459     }
460     TLink usagesAsTarget = links.Count(new Link<TLink>(constants.Any, constants.Any,
461         ↪ link));
462     if (equalityComparer.Equals(values[constants.TargetPart], link))
463     {
464         usagesAsTarget = Arithmetic<TLink>.Decrement(usagesAsTarget);
465     }
466     return Arithmetic<TLink>.Add(usagesAsSource, usagesAsTarget);
467 }
468
469 /// <param name="links">Хранилище связей.</param>
470 [MethodImpl(MethodImplOptions.AggressiveInlining)]
471 public static bool HasUsages<TLink>(this ILinks<TLink> links, TLink link) =>
472     ↪ Comparer<TLink>.Default.Compare(links.CountUsages(link), default) > 0;
473
474 /// <param name="links">Хранилище связей.</param>
475 [MethodImpl(MethodImplOptions.AggressiveInlining)]
476 public static bool Equals<TLink>(this ILinks<TLink> links, TLink link, TLink source,
477     ↪ TLink target)
478 {
479     var constants = links.Constants;
480     var values = links.GetLink(link);
481     var equalityComparer = EqualityComparer<TLink>.Default;
482     return equalityComparer.Equals(values[constants.SourcePart], source) &&
483         ↪ equalityComparer.Equals(values[constants.TargetPart], target);
484 }
485
486 /// <summary>
487 /// Выполняет поиск связи с указанными Source (началом) и Target (концом).
488 /// </summary>
489 /// <param name="links">Хранилище связей.</param>
490 /// <param name="source">Индекс связи, которая является началом для искомой
491     ↪ связи.</param>
492 /// <param name="target">Индекс связи, которая является концом для искомой связи.</param>
493 /// <returns>Индекс искомой связи с указанными Source (началом) и Target
494     ↪ (концом).</returns>
495 [MethodImpl(MethodImplOptions.AggressiveInlining)]
496 public static TLink SearchOrDefault<TLink>(this ILinks<TLink> links, TLink source, TLink
497     ↪ target)
498 {
499     var constants = links.Constants;
500     var setter = new Setter<TLink, TLink>(constants.Continue, constants.Break, default);
501     links.Each(setter.SetFirstAndReturnFalse, constants.Any, source, target);
502     return setter.Result;
503 }
504
505 /// <param name="links">Хранилище связей.</param>
506 [MethodImpl(MethodImplOptions.AggressiveInlining)]
507 public static TLink Create<TLink>(this ILinks<TLink> links) => links.Create(null);
508
509 /// <param name="links">Хранилище связей.</param>
510 [MethodImpl(MethodImplOptions.AggressiveInlining)]
511 public static TLink CreatePoint<TLink>(this ILinks<TLink> links)
512 {
513     var link = links.Create();
514     return links.Update(link, link, link);
515 }
516
517 /// <param name="links">Хранилище связей.</param>
518 [MethodImpl(MethodImplOptions.AggressiveInlining)]
519 public static TLink CreateAndUpdate<TLink>(this ILinks<TLink> links, TLink source, TLink
520     ↪ target) => links.Update(links.Create(), source, target);
521
522 /// <summary>
523 /// Обновляет связь с указанными началом (Source) и концом (Target)
524 /// на связь с указанными началом (NewSource) и концом (NewTarget).
525 /// </summary>
526 /// <param name="links">Хранилище связей.</param>
527 /// <param name="link">Индекс обновляемой связи.</param>
528 /// <param name="newSource">Индекс связи, которая является началом связи, на которую
529     ↪ выполняется обновление.</param>

```

```

520  /// <param name="newTarget">Индекс связи, которая является концом связи, на которую
    ↳ выполняется обновление.</param>
521  /// <returns>Индекс обновлённой связи.</returns>
522  [MethodImpl(MethodImplOptions.AggressiveInlining)]
523  public static TLink Update<TLink>(this ILinks<TLink> links, TLink link, TLink newSource,
    ↳ TLink newTarget) => links.Update(new LinkAddress<TLink>(link), new Link<TLink>(link,
    ↳ newSource, newTarget));

524
525  /// <summary>
526  /// Обновляет связь с указанными началом (Source) и концом (Target)
527  /// на связь с указанными началом (NewSource) и концом (NewTarget).
528  /// </summary>
529  /// <param name="links">Хранилище связей.</param>
530  /// <param name="restrictions">Ограничения на содержимое связей. Каждое ограничение
    ↳ может иметь значения: Constants.Null - 0-я связь, обозначающая ссылку на пустоту,
    ↳ Itself - требование установить ссылку на себя, 1.. $\infty$  конкретный адрес другой
    ↳ связи.</param>
531  /// <returns>Индекс обновлённой связи.</returns>
532  [MethodImpl(MethodImplOptions.AggressiveInlining)]
533  public static TLink Update<TLink>(this ILinks<TLink> links, params TLink[] restrictions)
534  {
535      if (restrictions.Length == 2)
536      {
537          return links.MergeAndDelete(restrictions[0], restrictions[1]);
538      }
539      if (restrictions.Length == 4)
540      {
541          return links.UpdateOrCreateOrGet(restrictions[0], restrictions[1],
    ↳ restrictions[2], restrictions[3]);
542      }
543      else
544      {
545          return links.Update(new LinkAddress<TLink>(restrictions[0]), restrictions);
546      }
547  }

548
549  [MethodImpl(MethodImplOptions.AggressiveInlining)]
550  public static IList<TLink> ResolveConstantAsSelfReference<TLink>(this ILinks<TLink>
    ↳ links, TLink constant, IList<TLink> restrictions, IList<TLink> substitution)
551  {
552      var equalityComparer = EqualityComparer<TLink>.Default;
553      var constants = links.Constants;
554      var restrictionsIndex = restrictions[constants.IndexPart];
555      var substitutionIndex = substitution[constants.IndexPart];
556      if (equalityComparer.Equals(substitutionIndex, default))
557      {
558          substitutionIndex = restrictionsIndex;
559      }
560      var source = substitution[constants.SourcePart];
561      var target = substitution[constants.TargetPart];
562      source = equalityComparer.Equals(source, constant) ? substitutionIndex : source;
563      target = equalityComparer.Equals(target, constant) ? substitutionIndex : target;
564      return new Link<TLink>(substitutionIndex, source, target);
565  }

566
567  /// <summary>
568  /// Создаёт связь (если она не существовала), либо возвращает индекс существующей связи
    ↳ с указанными Source (началом) и Target (концом).
569  /// </summary>
570  /// <param name="links">Хранилище связей.</param>
571  /// <param name="source">Индекс связи, которая является началом на создаваемой
    ↳ связи.</param>
572  /// <param name="target">Индекс связи, которая является концом для создаваемой
    ↳ связи.</param>
573  /// <returns>Индекс связи, с указанным Source (началом) и Target (концом)</returns>
574  [MethodImpl(MethodImplOptions.AggressiveInlining)]
575  public static TLink GetOrCreate<TLink>(this ILinks<TLink> links, TLink source, TLink
    ↳ target)
576  {
577      var link = links.SearchOrDefault(source, target);
578      if (EqualityComparer<TLink>.Default.Equals(link, default))
579      {
580          link = links.CreateAndUpdate(source, target);
581      }
582      return link;
583  }

584
585  /// <summary>

```

```

586 /// Обновляет связь с указанными началом (Source) и концом (Target)
587 /// на связь с указанными началом (NewSource) и концом (NewTarget).
588 /// </summary>
589 /// <param name="links">Хранилище связей.</param>
590 /// <param name="source">Индекс связи, которая является началом обновляемой
    ↳ связи.</param>
591 /// <param name="target">Индекс связи, которая является концом обновляемой связи.</param>
592 /// <param name="newSource">Индекс связи, которая является началом связи, на которую
    ↳ выполняется обновление.</param>
593 /// <param name="newTarget">Индекс связи, которая является концом связи, на которую
    ↳ выполняется обновление.</param>
594 /// <returns>Индекс обновлённой связи.</returns>
595 [MethodImpl(MethodImplOptions.AggressiveInlining)]
596 public static TLink UpdateOrCreateOrGet<TLink>(this ILinks<TLink> links, TLink source,
    ↳ TLink target, TLink newSource, TLink newTarget)
597 {
598     var equalityComparer = EqualityComparer<TLink>.Default;
599     var link = links.SearchOrDefault(source, target);
600     if (equalityComparer.Equals(link, default))
601     {
602         return links.CreateAndUpdate(newSource, newTarget);
603     }
604     if (equalityComparer.Equals(newSource, source) && equalityComparer.Equals(newTarget,
    ↳ target))
605     {
606         return link;
607     }
608     return links.Update(link, newSource, newTarget);
609 }
610
611 /// <summary>Удаляет связь с указанными началом (Source) и концом (Target).</summary>
612 /// <param name="links">Хранилище связей.</param>
613 /// <param name="source">Индекс связи, которая является началом удаляемой связи.</param>
614 /// <param name="target">Индекс связи, которая является концом удаляемой связи.</param>
615 [MethodImpl(MethodImplOptions.AggressiveInlining)]
616 public static TLink DeleteIfExists<TLink>(this ILinks<TLink> links, TLink source, TLink
    ↳ target)
617 {
618     var link = links.SearchOrDefault(source, target);
619     if (!EqualityComparer<TLink>.Default.Equals(link, default))
620     {
621         links.Delete(link);
622         return link;
623     }
624     return default;
625 }
626
627 /// <summary>Удаляет несколько связей.</summary>
628 /// <param name="links">Хранилище связей.</param>
629 /// <param name="deletedLinks">Список адресов связей к удалению.</param>
630 [MethodImpl(MethodImplOptions.AggressiveInlining)]
631 public static void DeleteMany<TLink>(this ILinks<TLink> links, IList<TLink> deletedLinks)
632 {
633     for (int i = 0; i < deletedLinks.Count; i++)
634     {
635         links.Delete(deletedLinks[i]);
636     }
637 }
638
639 /// <remarks>Before execution of this method ensure that deleted link is detached (all
    ↳ values - source and target are reset to null) or it might enter into infinite
    ↳ recursion.</remarks>
640 [MethodImpl(MethodImplOptions.AggressiveInlining)]
641 public static void DeleteAllUsages<TLink>(this ILinks<TLink> links, TLink linkIndex)
642 {
643     var anyConstant = links.Constants.Any;
644     var usagesAsSourceQuery = new Link<TLink>(anyConstant, linkIndex, anyConstant);
645     links.DeleteByQuery(usagesAsSourceQuery);
646     var usagesAsTargetQuery = new Link<TLink>(anyConstant, linkIndex, anyConstant);
647     links.DeleteByQuery(usagesAsTargetQuery);
648 }
649
650 [MethodImpl(MethodImplOptions.AggressiveInlining)]
651 public static void DeleteByQuery<TLink>(this ILinks<TLink> links, Link<TLink> query)
652 {
653     var count = CheckedConverter<TLink, long>.Default.Convert(links.Count(query));
654     if (count > 0)
655     {

```

```

656     var queryResult = new TLink[count];
657     var queryResultFiller = new ArrayFiller<TLink, TLink>(queryResult,
        ↪ links.Constants.Continue);
658     links.Each(queryResultFiller.AddFirstAndReturnConstant, query);
659     for (var i = count - 1; i >= 0; i--)
660     {
661         links.Delete(queryResult[i]);
662     }
663 }
664 }
665
666 // TODO: Move to Platform.Data
667 [MethodImpl(MethodImplOptions.AggressiveInlining)]
668 public static bool AreValuesReset<TLink>(this ILinks<TLink> links, TLink linkIndex)
669 {
670     var nullConstant = links.Constants.Null;
671     var equalityComparer = EqualityComparer<TLink>.Default;
672     var link = links.GetLink(linkIndex);
673     for (int i = 1; i < link.Count; i++)
674     {
675         if (!equalityComparer.Equals(link[i], nullConstant))
676         {
677             return false;
678         }
679     }
680     return true;
681 }
682
683 // TODO: Create a universal version of this method in Platform.Data (with using of for
        ↪ loop)
684 [MethodImpl(MethodImplOptions.AggressiveInlining)]
685 public static void ResetValues<TLink>(this ILinks<TLink> links, TLink linkIndex)
686 {
687     var nullConstant = links.Constants.Null;
688     var updateRequest = new Link<TLink>(linkIndex, nullConstant, nullConstant);
689     links.Update(updateRequest);
690 }
691
692 // TODO: Create a universal version of this method in Platform.Data (with using of for
        ↪ loop)
693 [MethodImpl(MethodImplOptions.AggressiveInlining)]
694 public static void EnforceResetValues<TLink>(this ILinks<TLink> links, TLink linkIndex)
695 {
696     if (!links.AreValuesReset(linkIndex))
697     {
698         links.ResetValues(linkIndex);
699     }
700 }
701
702 /// <summary>
703 /// Merging two usages graphs, all children of old link moved to be children of new link
        ↪ or deleted.
704 /// </summary>
705 [MethodImpl(MethodImplOptions.AggressiveInlining)]
706 public static TLink MergeUsages<TLink>(this ILinks<TLink> links, TLink oldLinkIndex,
        ↪ TLink newLinkIndex)
707 {
708     var addressToInt64Converter = CheckedConverter<TLink, long>.Default;
709     var equalityComparer = EqualityComparer<TLink>.Default;
710     if (!equalityComparer.Equals(oldLinkIndex, newLinkIndex))
711     {
712         var constants = links.Constants;
713         var usagesAsSourceQuery = new Link<TLink>(constants.Any, oldLinkIndex,
            ↪ constants.Any);
714         var usagesAsSourceCount =
            ↪ addressToInt64Converter.Convert(links.Count(usagesAsSourceQuery));
715         var usagesAsTargetQuery = new Link<TLink>(constants.Any, constants.Any,
            ↪ oldLinkIndex);
716         var usagesAsTargetCount =
            ↪ addressToInt64Converter.Convert(links.Count(usagesAsTargetQuery));
717         var isStandalonePoint = Point<TLink>.IsFullPoint(links.GetLink(oldLinkIndex)) &&
            ↪ usagesAsSourceCount == 1 && usagesAsTargetCount == 1;
718         if (!isStandalonePoint)
719         {
720             var totalUsages = usagesAsSourceCount + usagesAsTargetCount;
721             if (totalUsages > 0)
722             {
723                 var usages = ArrayPool.Allocate<TLink>(totalUsages);

```

```

724     var usagesFiller = new ArrayFiller<TLink, TLink>(usages,
725         ↪ links.Constants.Continue);
726     var i = 0L;
727     if (usagesAsSourceCount > 0)
728     {
729         links.Each(usagesFiller.AddFirstAndReturnConstant,
730             ↪ usagesAsSourceQuery);
731         for (; i < usagesAsSourceCount; i++)
732         {
733             var usage = usages[i];
734             if (!equalityComparer.Equals(usage, oldLinkIndex))
735             {
736                 links.Update(usage, newLinkIndex, links.GetTarget(usage));
737             }
738         }
739     }
740     if (usagesAsTargetCount > 0)
741     {
742         links.Each(usagesFiller.AddFirstAndReturnConstant,
743             ↪ usagesAsTargetQuery);
744         for (; i < usages.Length; i++)
745         {
746             var usage = usages[i];
747             if (!equalityComparer.Equals(usage, oldLinkIndex))
748             {
749                 links.Update(usage, links.GetSource(usage), newLinkIndex);
750             }
751         }
752     }
753     ArrayPool.Free(usages);
754 }
755 }
756 }
757 }
758 }
759 }
760 }
761 }
762 }
763 }
764 }
765 }
766 }
767 }
768 }
769 }
770 }
771 }
772 }
773 }
774 }
775 }
776 }
777 }
778 }
779 }
780 }
781 }

```

1.19 ./Platform.Data.Doublets/ISynchronizedLinks.cs

```

1  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
2
3  namespace Platform.Data.Doublets
4  {
5      public interface ISynchronizedLinks<TLink> : ISynchronizedLinks<TLink, ILinks<TLink>,
6          ↪ LinksConstants<TLink>>, ILinks<TLink>
7      {
8      }
9  }

```


1.20 ./Platform.Data.Doublets/Incrementers/FrequencyIncrementer.cs

```
1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3 using Platform.Incrementers;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data.Doublets.Incrementers
8 {
9     public class FrequencyIncrementer<TLink> : LinksOperatorBase<TLink>, IIncrementer<TLink>
10    {
11        private static readonly EqualityComparer<TLink> _equalityComparer =
12            ↪ EqualityComparer<TLink>.Default;
13
14        private readonly TLink _frequencyMarker;
15        private readonly TLink _unaryOne;
16        private readonly IIncrementer<TLink> _unaryNumberIncrementer;
17
18        [MethodImpl(MethodImplOptions.AggressiveInlining)]
19        public FrequencyIncrementer(ILinks<TLink> links, TLink frequencyMarker, TLink unaryOne,
20            ↪ IIncrementer<TLink> unaryNumberIncrementer)
21            : base(links)
22        {
23            _frequencyMarker = frequencyMarker;
24            _unaryOne = unaryOne;
25            _unaryNumberIncrementer = unaryNumberIncrementer;
26        }
27
28        [MethodImpl(MethodImplOptions.AggressiveInlining)]
29        public TLink Increment(TLink frequency)
30        {
31            if (_equalityComparer.Equals(frequency, default))
32            {
33                return Links.GetOrCreate(_unaryOne, _frequencyMarker);
34            }
35            var incrementedSource =
36                ↪ _unaryNumberIncrementer.Increment(Links.GetSource(frequency));
37            return Links.GetOrCreate(incrementedSource, _frequencyMarker);
38        }
39    }
40 }
```

1.21 ./Platform.Data.Doublets/Incrementers/UnaryNumberIncrementer.cs

```
1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3 using Platform.Incrementers;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data.Doublets.Incrementers
8 {
9     public class UnaryNumberIncrementer<TLink> : LinksOperatorBase<TLink>, IIncrementer<TLink>
10    {
11        private static readonly EqualityComparer<TLink> _equalityComparer =
12            ↪ EqualityComparer<TLink>.Default;
13
14        private readonly TLink _unaryOne;
15
16        [MethodImpl(MethodImplOptions.AggressiveInlining)]
17        public UnaryNumberIncrementer(ILinks<TLink> links, TLink unaryOne) : base(links) =>
18            ↪ _unaryOne = unaryOne;
19
20        [MethodImpl(MethodImplOptions.AggressiveInlining)]
21        public TLink Increment(TLink unaryNumber)
22        {
23            if (_equalityComparer.Equals(unaryNumber, _unaryOne))
24            {
25                return Links.GetOrCreate(_unaryOne, _unaryOne);
26            }
27            var source = Links.GetSource(unaryNumber);
28            var target = Links.GetTarget(unaryNumber);
29            if (_equalityComparer.Equals(source, target))
30            {
31                return Links.GetOrCreate(unaryNumber, _unaryOne);
32            }
33            else
34            {
35                return Links.GetOrCreate(source, Increment(target));
36            }
37        }
38    }
39 }
```

```
36     }
37 }
```

1.22 ./Platform.Data.Doublets/Link.cs

```
1  using Platform.Collections.Lists;
2  using Platform.Exceptions;
3  using Platform.Ranges;
4  using Platform.Singletons;
5  using System;
6  using System.Collections;
7  using System.Collections.Generic;
8  using System.Runtime.CompilerServices;
9
10 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
11
12 namespace Platform.Data.Doublets
13 {
14     /// <summary>
15     /// Структура описывающая уникальную связь.
16     /// </summary>
17     public struct Link<TLink> : IEquatable<Link<TLink>>, IReadOnlyList<TLink>, IList<TLink>
18     {
19         public static readonly Link<TLink> Null = new Link<TLink>();
20
21         private static readonly LinksConstants<TLink> _constants =
22             ↪ Default<LinksConstants<TLink>>.Instance;
23         private static readonly EqualityComparer<TLink> _equalityComparer =
24             ↪ EqualityComparer<TLink>.Default;
25
26         private const int Length = 3;
27
28         public readonly TLink Index;
29         public readonly TLink Source;
30         public readonly TLink Target;
31
32         [MethodImpl(MethodImplOptions.AggressiveInlining)]
33         public Link(params TLink[] values) => SetValues(values, out Index, out Source, out
34             ↪ Target);
35
36         [MethodImpl(MethodImplOptions.AggressiveInlining)]
37         public Link(IList<TLink> values) => SetValues(values, out Index, out Source, out Target);
38
39         [MethodImpl(MethodImplOptions.AggressiveInlining)]
40         public Link(object other)
41         {
42             if (other is Link<TLink> otherLink)
43             {
44                 SetValues(ref otherLink, out Index, out Source, out Target);
45             }
46             else if (other is IList<TLink> otherList)
47             {
48                 SetValues(otherList, out Index, out Source, out Target);
49             }
50             else
51             {
52                 throw new NotSupportedException();
53             }
54         }
55
56         [MethodImpl(MethodImplOptions.AggressiveInlining)]
57         public Link(ref Link<TLink> other) => SetValues(ref other, out Index, out Source, out
58             ↪ Target);
59
60         [MethodImpl(MethodImplOptions.AggressiveInlining)]
61         public Link(TLink index, TLink source, TLink target)
62         {
63             Index = index;
64             Source = source;
65             Target = target;
66         }
67
68         [MethodImpl(MethodImplOptions.AggressiveInlining)]
69         private static void SetValues(ref Link<TLink> other, out TLink index, out TLink source,
70             ↪ out TLink target)
71         {
72             index = other.Index;
73             source = other.Source;
74             target = other.Target;
75         }
76     }
77 }
```

```

72 [MethodImpl(MethodImplOptions.AggressiveInlining)]
73 private static void SetValues(TList<TLink> values, out TLink index, out TLink source,
74     ↪ out TLink target)
75 {
76     switch (values.Count)
77     {
78         case 3:
79             index = values[0];
80             source = values[1];
81             target = values[2];
82             break;
83         case 2:
84             index = values[0];
85             source = values[1];
86             target = default;
87             break;
88         case 1:
89             index = values[0];
90             source = default;
91             target = default;
92             break;
93         default:
94             index = default;
95             source = default;
96             target = default;
97             break;
98     }
99
100 [MethodImpl(MethodImplOptions.AggressiveInlining)]
101 public override int GetHashCode() => (Index, Source, Target).GetHashCode();
102
103 [MethodImpl(MethodImplOptions.AggressiveInlining)]
104 public bool IsNull() => _equalityComparer.Equals(Index, _constants.Null)
105     && _equalityComparer.Equals(Source, _constants.Null)
106     && _equalityComparer.Equals(Target, _constants.Null);
107
108 [MethodImpl(MethodImplOptions.AggressiveInlining)]
109 public override bool Equals(object other) => other is Link<TLink> &&
110     ↪ Equals((Link<TLink>)other);
111
112 [MethodImpl(MethodImplOptions.AggressiveInlining)]
113 public bool Equals(Link<TLink> other) => _equalityComparer.Equals(Index, other.Index)
114     && _equalityComparer.Equals(Source, other.Source)
115     && _equalityComparer.Equals(Target, other.Target);
116
117 [MethodImpl(MethodImplOptions.AggressiveInlining)]
118 public static string ToString(TLink index, TLink source, TLink target) => $"{index}:
119     ↪ {source}->{target}";
120
121 [MethodImpl(MethodImplOptions.AggressiveInlining)]
122 public static string ToString(TLink source, TLink target) => $"{source}->{target}";
123
124 [MethodImpl(MethodImplOptions.AggressiveInlining)]
125 public static implicit operator TLink[] (Link<TLink> link) => link.ToArray();
126
127 [MethodImpl(MethodImplOptions.AggressiveInlining)]
128 public static implicit operator Link<TLink>(TLink[] linkArray) => new
129     ↪ Link<TLink>(linkArray);
130
131 [MethodImpl(MethodImplOptions.AggressiveInlining)]
132 public override string ToString() => _equalityComparer.Equals(Index, _constants.Null) ?
133     ↪ ToString(Source, Target) : ToString(Index, Source, Target);
134
135 #region IList
136
137 public int Count
138 {
139     [MethodImpl(MethodImplOptions.AggressiveInlining)]
140     get => Length;
141 }
142
143 public bool IsReadOnly
144 {
145     [MethodImpl(MethodImplOptions.AggressiveInlining)]
146     get => true;
147 }
148
149 public TLink this[int index]
150 {

```

```

147 [MethodImpl(MethodImplOptions.AggressiveInlining)]
148 get
149 {
150     Ensure.OnDebug.ArgumentInRange(index, new Range<int>(0, Length - 1),
151         ↳ nameof(index));
152     if (index == _constants.IndexPart)
153     {
154         return Index;
155     }
156     if (index == _constants.SourcePart)
157     {
158         return Source;
159     }
160     if (index == _constants.TargetPart)
161     {
162         return Target;
163     }
164     throw new NotSupportedException(); // Impossible path due to
165     ↳ Ensure.ArgumentInRange
166 }
167 [MethodImpl(MethodImplOptions.AggressiveInlining)]
168 set => throw new NotSupportedException();
169
170 [MethodImpl(MethodImplOptions.AggressiveInlining)]
171 IEnumerator IEnumerable.GetEnumerator() => GetEnumerator();
172
173 [MethodImpl(MethodImplOptions.AggressiveInlining)]
174 public IEnumerator<TLink> GetEnumerator()
175 {
176     yield return Index;
177     yield return Source;
178     yield return Target;
179 }
180
181 [MethodImpl(MethodImplOptions.AggressiveInlining)]
182 public void Add(TLink item) => throw new NotSupportedException();
183
184 [MethodImpl(MethodImplOptions.AggressiveInlining)]
185 public void Clear() => throw new NotSupportedException();
186
187 [MethodImpl(MethodImplOptions.AggressiveInlining)]
188 public bool Contains(TLink item) => IndexOf(item) >= 0;
189
190 [MethodImpl(MethodImplOptions.AggressiveInlining)]
191 public void CopyTo(TLink[] array, int arrayIndex)
192 {
193     Ensure.OnDebug.ArgumentNotNull(array, nameof(array));
194     Ensure.OnDebug.ArgumentInRange(arrayIndex, new Range<int>(0, array.Length - 1),
195         ↳ nameof(arrayIndex));
196     if (arrayIndex + Length > array.Length)
197     {
198         throw new InvalidOperationException();
199     }
200     array[arrayIndex++] = Index;
201     array[arrayIndex++] = Source;
202     array[arrayIndex] = Target;
203 }
204
205 [MethodImpl(MethodImplOptions.AggressiveInlining)]
206 public bool Remove(TLink item) => Throw.A.NotSupportedExceptionAndReturn<bool>();
207
208 [MethodImpl(MethodImplOptions.AggressiveInlining)]
209 public int IndexOf(TLink item)
210 {
211     if (_equalityComparer.Equals(Index, item))
212     {
213         return _constants.IndexPart;
214     }
215     if (_equalityComparer.Equals(Source, item))
216     {
217         return _constants.SourcePart;
218     }
219     if (_equalityComparer.Equals(Target, item))
220     {
221         return _constants.TargetPart;
222     }
223     return -1;
224 }

```

```

223
224     [MethodImpl(MethodImplOptions.AggressiveInlining)]
225     public void Insert(int index, TLink item) => throw new NotSupportedException();
226
227     [MethodImpl(MethodImplOptions.AggressiveInlining)]
228     public void RemoveAt(int index) => throw new NotSupportedException();
229
230     [MethodImpl(MethodImplOptions.AggressiveInlining)]
231     public static bool operator ==(Link<TLink> left, Link<TLink> right) =>
232         ↪ left.Equals(right);
233
234     [MethodImpl(MethodImplOptions.AggressiveInlining)]
235     public static bool operator !=(Link<TLink> left, Link<TLink> right) => !(left == right);
236
237     #endregion
238 }

```

1.23 ./Platform.Data.Doublets/LinkExtensions.cs

```

1 using System.Runtime.CompilerServices;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Doublets
6 {
7     public static class LinkExtensions
8     {
9         [MethodImpl(MethodImplOptions.AggressiveInlining)]
10        public static bool IsFullPoint<TLink>(this Link<TLink> link) =>
11            ↪ Point<TLink>.IsFullPoint(link);
12
13        [MethodImpl(MethodImplOptions.AggressiveInlining)]
14        public static bool IsPartialPoint<TLink>(this Link<TLink> link) =>
15            ↪ Point<TLink>.IsPartialPoint(link);
16    }
17 }

```

1.24 ./Platform.Data.Doublets/LinksOperatorBase.cs

```

1 using System.Runtime.CompilerServices;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Doublets
6 {
7     public abstract class LinksOperatorBase<TLink>
8     {
9         public ILinks<TLink> Links
10        {
11            [MethodImpl(MethodImplOptions.AggressiveInlining)]
12            get;
13        }
14
15        [MethodImpl(MethodImplOptions.AggressiveInlining)]
16        protected LinksOperatorBase(ILinks<TLink> links) => Links = links;
17    }
18 }

```

1.25 ./Platform.Data.Doublets/Numbers/Unary/AddressToUnaryNumberConverter.cs

```

1 using System.Collections.Generic;
2 using Platform.Reflection;
3 using Platform.Converters;
4 using Platform.Numbers;
5 using System.Runtime.CompilerServices;
6
7 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9 namespace Platform.Data.Doublets.Numbers.Unary
10 {
11     public class AddressToUnaryNumberConverter<TLink> : LinksOperatorBase<TLink>,
12         ↪ IConverter<TLink>
13     {
14         private static readonly EqualityComparer<TLink> _equalityComparer =
15             ↪ EqualityComparer<TLink>.Default;
16         private static readonly TLink _zero = default;
17         private static readonly TLink _one = Arithmetic.Increment(_zero);
18
19         private readonly IConverter<int, TLink> _powerOf2ToUnaryNumberConverter;
20
21         [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

20     public AddressToUnaryNumberConverter(ILinks<TLink> links, IConverter<int, TLink>
    ↪ powerOf2ToUnaryNumberConverter) : base(links) => _powerOf2ToUnaryNumberConverter =
    ↪ powerOf2ToUnaryNumberConverter;
21
22     [MethodImpl(MethodImplOptions.AggressiveInlining)]
23     public TLink Convert(TLink number)
24     {
25         var nullConstant = Links.Constants.Null;
26         var target = nullConstant;
27         for (var i = 0; !_equalityComparer.Equals(number, _zero) && i <
    ↪ NumericType<TLink>.BitsSize; i++)
28         {
29             if (_equalityComparer.Equals(Bit.And(number, _one), _one))
30             {
31                 target = _equalityComparer.Equals(target, nullConstant)
32                     ? _powerOf2ToUnaryNumberConverter.Convert(i)
33                     : Links.GetOrCreate(_powerOf2ToUnaryNumberConverter.Convert(i), target);
34             }
35             number = Bit.ShiftRight(number, 1);
36         }
37         return target;
38     }
39 }
40 }

```

1.26 ./Platform.Data.Doublets/Numbers/Unary/LinkToItsFrequencyNumberConveter.cs

```

1  using System;
2  using System.Collections.Generic;
3  using Platform.Interfaces;
4  using Platform.Converters;
5  using System.Runtime.CompilerServices;
6
7  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9  namespace Platform.Data.Doublets.Numbers.Unary
10 {
11     public class LinkToItsFrequencyNumberConveter<TLink> : LinksOperatorBase<TLink>,
    ↪ IConverter<Doublet<TLink>, TLink>
12     {
13         private static readonly EqualityComparer<TLink> _equalityComparer =
    ↪ EqualityComparer<TLink>.Default;
14
15         private readonly IProperty<TLink, TLink> _frequencyPropertyOperator;
16         private readonly IConverter<TLink> _unaryNumberToAddressConverter;
17
18         [MethodImpl(MethodImplOptions.AggressiveInlining)]
19         public LinkToItsFrequencyNumberConveter(
20             ILinks<TLink> links,
21             IProperty<TLink, TLink> frequencyPropertyOperator,
22             IConverter<TLink> unaryNumberToAddressConverter)
23             : base(links)
24         {
25             _frequencyPropertyOperator = frequencyPropertyOperator;
26             _unaryNumberToAddressConverter = unaryNumberToAddressConverter;
27         }
28
29         [MethodImpl(MethodImplOptions.AggressiveInlining)]
30         public TLink Convert(Doublet<TLink> doublet)
31         {
32             var link = Links.SearchOrDefault(doublet.Source, doublet.Target);
33             if (_equalityComparer.Equals(link, default))
34             {
35                 throw new ArgumentException($"Link ({doublet}) not found.", nameof(doublet));
36             }
37             var frequency = _frequencyPropertyOperator.Get(link);
38             if (_equalityComparer.Equals(frequency, default))
39             {
40                 return default;
41             }
42             var frequencyNumber = Links.GetSource(frequency);
43             return _unaryNumberToAddressConverter.Convert(frequencyNumber);
44         }
45     }
46 }

```

1.27 ./Platform.Data.Doublets/Numbers/Unary/PowerOf2ToUnaryNumberConverter.cs

```

1  using System.Collections.Generic;
2  using Platform.Exceptions;
3  using Platform.Ranges;

```

```

4 using Platform.Converters;
5 using System.Runtime.CompilerServices;
6
7 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9 namespace Platform.Data.Doublets.Numbers.Unary
10 {
11     public class PowerOf2ToUnaryNumberConverter<TLink> : LinksOperatorBase<TLink>,
12         ↪ IConverter<int, TLink>
13     {
14         private static readonly EqualityComparer<TLink> _equalityComparer =
15             ↪ EqualityComparer<TLink>.Default;
16
17         private readonly TLink[] _unaryNumberPowersOf2;
18
19         [MethodImpl(MethodImplOptions.AggressiveInlining)]
20         public PowerOf2ToUnaryNumberConverter(ILinks<TLink> links, TLink one) : base(links)
21         {
22             _unaryNumberPowersOf2 = new TLink[64];
23             _unaryNumberPowersOf2[0] = one;
24         }
25
26         [MethodImpl(MethodImplOptions.AggressiveInlining)]
27         public TLink Convert(int power)
28         {
29             Ensure.Always.ArgumentInRange(power, new Range<int>(0, _unaryNumberPowersOf2.Length
30                 ↪ - 1), nameof(power));
31             if (!_equalityComparer.Equals(_unaryNumberPowersOf2[power], default))
32             {
33                 return _unaryNumberPowersOf2[power];
34             }
35             var previousPowerOf2 = Convert(power - 1);
36             var powerOf2 = Links.GetOrCreate(previousPowerOf2, previousPowerOf2);
37             _unaryNumberPowersOf2[power] = powerOf2;
38             return powerOf2;
39         }
40     }
41 }

```

1.28 ./Platform.Data.Doublets/Numbers/Unary/UnaryNumberToAddressAddOperationConverter.cs

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3 using Platform.Converters;
4 using Platform.Numbers;
5
6 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8 namespace Platform.Data.Doublets.Numbers.Unary
9 {
10     public class UnaryNumberToAddressAddOperationConverter<TLink> : LinksOperatorBase<TLink>,
11         ↪ IConverter<TLink>
12     {
13         private static readonly EqualityComparer<TLink> _equalityComparer =
14             ↪ EqualityComparer<TLink>.Default;
15         private static readonly UncheckedConverter<TLink, ulong> _addressToUInt64Converter =
16             ↪ UncheckedConverter<TLink, ulong>.Default;
17         private static readonly UncheckedConverter<ulong, TLink> _uInt64ToAddressConverter =
18             ↪ UncheckedConverter<ulong, TLink>.Default;
19         private static readonly TLink _zero = default;
20         private static readonly TLink _one = Arithmetic.Increment(_zero);
21
22         private readonly Dictionary<TLink, TLink> _unaryToUInt64;
23         private readonly TLink _unaryOne;
24
25         [MethodImpl(MethodImplOptions.AggressiveInlining)]
26         public UnaryNumberToAddressAddOperationConverter(ILinks<TLink> links, TLink unaryOne)
27             : base(links)
28         {
29             _unaryOne = unaryOne;
30             _unaryToUInt64 = CreateUnaryToUInt64Dictionary(links, unaryOne);
31         }
32
33         [MethodImpl(MethodImplOptions.AggressiveInlining)]
34         public TLink Convert(TLink unaryNumber)
35         {
36             if (_equalityComparer.Equals(unaryNumber, default))
37             {
38                 return default;
39             }
40             if (_equalityComparer.Equals(unaryNumber, _unaryOne))

```

```

37     {
38         return _one;
39     }
40     var source = Links.GetSource(unaryNumber);
41     var target = Links.GetTarget(unaryNumber);
42     if (_equalityComparer.Equals(source, target))
43     {
44         return _unaryToUInt64[unaryNumber];
45     }
46     else
47     {
48         var result = _unaryToUInt64[source];
49         TLink lastValue;
50         while (!_unaryToUInt64.TryGetValue(target, out lastValue))
51         {
52             source = Links.GetSource(target);
53             result = Arithmetic<TLink>.Add(result, _unaryToUInt64[source]);
54             target = Links.GetTarget(target);
55         }
56         result = Arithmetic<TLink>.Add(result, lastValue);
57         return result;
58     }
59 }
60
61 [MethodImpl(MethodImplOptions.AggressiveInlining)]
62 private static Dictionary<TLink, TLink> CreateUnaryToUInt64Dictionary(ILinks<TLink>
↪ links, TLink unaryOne)
63 {
64     var unaryToUInt64 = new Dictionary<TLink, TLink>
65     {
66         { unaryOne, _one }
67     };
68     var unary = unaryOne;
69     var number = _one;
70     for (var i = 1; i < 64; i++)
71     {
72         unary = links.GetOrCreate(unary, unary);
73         number = Double(number);
74         unaryToUInt64.Add(unary, number);
75     }
76     return unaryToUInt64;
77 }
78
79 [MethodImpl(MethodImplOptions.AggressiveInlining)]
80 private static TLink Double(TLink number) =>
↪ _uInt64ToAddressConverter.Convert(_addressToUInt64Converter.Convert(number) * 2UL);
81 }
82 }

```

1.29 ./Platform.Data.Doublets/Numbers/Unary/UnaryNumberToAddressOrOperationConverter.cs

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3 using Platform.Reflection;
4 using Platform.Converters;
5 using Platform.Numbers;
6
7 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9 namespace Platform.Data.Doublets.Numbers.Unary
10 {
11     public class UnaryNumberToAddressOrOperationConverter<TLink> : LinksOperatorBase<TLink>,
↪ IConverter<TLink>
12     {
13         private static readonly EqualityComparer<TLink> _equalityComparer =
↪ EqualityComparer<TLink>.Default;
14         private static readonly TLink _zero = default;
15         private static readonly TLink _one = Arithmetic.Increment(_zero);
16
17         private readonly IDictionary<TLink, int> _unaryNumberPowerOf2Indicies;
18
19         [MethodImpl(MethodImplOptions.AggressiveInlining)]
20         public UnaryNumberToAddressOrOperationConverter(ILinks<TLink> links, IConverter<int,
↪ TLink> powerOf2ToUnaryNumberConverter) : base(links) => _unaryNumberPowerOf2Indicies
↪ = CreateUnaryNumberPowerOf2IndiciesDictionary(powerOf2ToUnaryNumberConverter);
21
22         [MethodImpl(MethodImplOptions.AggressiveInlining)]
23         public TLink Convert(TLink sourceNumber)
24         {
25             var links = Links;
26             var nullConstant = links.Constants.Null;

```



```

27     var source = sourceNumber;
28     var target = nullConstant;
29     if (!_equalityComparer.Equals(source, nullConstant))
30     {
31         while (true)
32         {
33             if (_unaryNumberPowerOf2Indicies.TryGetValue(source, out int powerOf2Index))
34             {
35                 SetBit(ref target, powerOf2Index);
36                 break;
37             }
38             else
39             {
40                 powerOf2Index = _unaryNumberPowerOf2Indicies[links.GetSource(source)];
41                 SetBit(ref target, powerOf2Index);
42                 source = links.GetTarget(source);
43             }
44         }
45     }
46     return target;
47 }
48
49 [MethodImpl(MethodImplOptions.AggressiveInlining)]
50 private static Dictionary<TLink, int>
    ↪ CreateUnaryNumberPowerOf2IndiciesDictionary(IConverter<int, TLink>
    ↪ powerOf2ToUnaryNumberConverter)
51 {
52     var unaryNumberPowerOf2Indicies = new Dictionary<TLink, int>();
53     for (int i = 0; i < NumericType<TLink>.BitsSize; i++)
54     {
55         unaryNumberPowerOf2Indicies.Add(powerOf2ToUnaryNumberConverter.Convert(i), i);
56     }
57     return unaryNumberPowerOf2Indicies;
58 }
59
60 [MethodImpl(MethodImplOptions.AggressiveInlining)]
61 private static void SetBit(ref TLink target, int powerOf2Index) => target =
    ↪ Bit.Or(target, Bit.ShiftLeft(_one, powerOf2Index));
62 }
63 }

```

1.30 ./Platform.Data.Doublents/PropertyOperators/PropertiesOperator.cs

```

1  using System.Linq;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Interfaces;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublents.PropertyOperators
9  {
10     public class PropertiesOperator<TLink> : LinksOperatorBase<TLink>, IProperties<TLink, TLink,
    ↪ TLink>
11     {
12         private static readonly EqualityComparer<TLink> _equalityComparer =
    ↪ EqualityComparer<TLink>.Default;
13
14         [MethodImpl(MethodImplOptions.AggressiveInlining)]
15         public PropertiesOperator(ILinks<TLink> links) : base(links) { }
16
17         [MethodImpl(MethodImplOptions.AggressiveInlining)]
18         public TLink GetValue(TLink @object, TLink property)
19         {
20             var objectProperty = Links.SearchOrDefault(@object, property);
21             if (_equalityComparer.Equals(objectProperty, default))
22             {
23                 return default;
24             }
25             var valueLink = Links.All(Links.Constants.Any, objectProperty).SingleOrDefault();
26             if (valueLink == null)
27             {
28                 return default;
29             }
30             return Links.GetTarget(valueLink[Links.Constants.IndexPart]);
31         }
32
33         [MethodImpl(MethodImplOptions.AggressiveInlining)]
34         public void SetValue(TLink @object, TLink property, TLink value)
35         {

```

```

36         var objectProperty = Links.GetOrCreate(@Object, property);
37         Links.DeleteMany(Links.AllIndices(Links.Constants.Any, objectProperty));
38         Links.GetOrCreate(objectProperty, value);
39     }
40 }
41 }

```

1.31 ./Platform.Data.Doublets/PropertyOperators/PropertyOperator.cs

```

1  using System.Collections.Generic;
2  using System.Runtime.CompilerServices;
3  using Platform.Interfaces;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.PropertyOperators
8  {
9      public class PropertyOperator<TLink> : LinksOperatorBase<TLink>, IProperty<TLink, TLink>
10     {
11         private static readonly EqualityComparer<TLink> _equalityComparer =
12             ↳ EqualityComparer<TLink>.Default;
13
14         private readonly TLink _propertyMarker;
15         private readonly TLink _propertyValueMarker;
16
17         [MethodImpl(MethodImplOptions.AggressiveInlining)]
18         public PropertyOperator(ILinks<TLink> links, TLink propertyMarker, TLink
19             ↳ propertyValueMarker) : base(links)
20         {
21             _propertyMarker = propertyMarker;
22             _propertyValueMarker = propertyValueMarker;
23         }
24
25         [MethodImpl(MethodImplOptions.AggressiveInlining)]
26         public TLink Get(TLink link)
27         {
28             var property = Links.SearchOrDefault(link, _propertyMarker);
29             return GetValue(GetContainer(property));
30         }
31
32         [MethodImpl(MethodImplOptions.AggressiveInlining)]
33         private TLink GetContainer(TLink property)
34         {
35             var valueContainer = default(TLink);
36             if (_equalityComparer.Equals(property, default))
37             {
38                 return valueContainer;
39             }
40             var links = Links;
41             var constants = links.Constants;
42             var countinueConstant = constants.Continue;
43             var breakConstant = constants.Break;
44             var anyConstant = constants.Any;
45             var query = new Link<TLink>(anyConstant, property, anyConstant);
46             links.Each(candidate =>
47             {
48                 var candidateTarget = links.GetTarget(candidate);
49                 var valueTarget = links.GetTarget(candidateTarget);
50                 if (_equalityComparer.Equals(valueTarget, _propertyValueMarker))
51                 {
52                     valueContainer = links.GetIndex(candidate);
53                     return breakConstant;
54                 }
55                 return countinueConstant;
56             }, query);
57             return valueContainer;
58         }
59
60         [MethodImpl(MethodImplOptions.AggressiveInlining)]
61         private TLink GetValue(TLink container) => _equalityComparer.Equals(container, default)
62             ↳ ? default : Links.GetTarget(container);
63
64         [MethodImpl(MethodImplOptions.AggressiveInlining)]
65         public void Set(TLink link, TLink value)
66         {
67             var links = Links;
68             var property = links.GetOrCreate(link, _propertyMarker);
69             var container = GetContainer(property);
70             if (_equalityComparer.Equals(container, default))
71             {
72                 links.GetOrCreate(property, value);
73             }
74         }
75     }
76 }

```

```

70     }
71     else
72     {
73         links.Update(container, property, value);
74     }
75 }
76 }
77 }

```

1.32 ./Platform.Data.Doublets/ResizableDirectMemory/Generic/LinksAvlBalancedTreeMethodsBase.cs

```

1  using System;
2  using System.Text;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5  using Platform.Collections.Methods.Trees;
6  using Platform.Converters;
7  using Platform.Numbers;
8  using static System.Runtime.CompilerServices.Unsafe;
9
10 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
11
12 namespace Platform.Data.Doublets.ResizableDirectMemory.Generic
13 {
14     public unsafe abstract class LinksAvlBalancedTreeMethodsBase<TLink> :
15         ↳ SizedAndThreadedAVLBalancedTreeMethods<TLink>, ILinksTreeMethods<TLink>
16     {
17         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
18             ↳ UncheckedConverter<TLink, long>.Default;
19         private static readonly UncheckedConverter<TLink, int> _addressToInt32Converter =
20             ↳ UncheckedConverter<TLink, int>.Default;
21         private static readonly UncheckedConverter<bool, TLink> _boolToAddressConverter =
22             ↳ UncheckedConverter<bool, TLink>.Default;
23         private static readonly UncheckedConverter<int, TLink> _int32ToAddressConverter =
24             ↳ UncheckedConverter<int, TLink>.Default;
25
26         protected readonly TLink Break;
27         protected readonly TLink Continue;
28         protected readonly byte* Links;
29         protected readonly byte* Header;
30
31         [MethodImpl(MethodImplOptions.AggressiveInlining)]
32         protected LinksAvlBalancedTreeMethodsBase(LinksConstants<TLink> constants, byte* links,
33             ↳ byte* header)
34         {
35             Links = links;
36             Header = header;
37             Break = constants.Break;
38             Continue = constants.Continue;
39         }
40
41         [MethodImpl(MethodImplOptions.AggressiveInlining)]
42         protected abstract TLink GetTreeRoot();
43
44         [MethodImpl(MethodImplOptions.AggressiveInlining)]
45         protected abstract TLink GetBasePartValue(TLink link);
46
47         [MethodImpl(MethodImplOptions.AggressiveInlining)]
48         protected abstract bool FirstIsToTheRightOfSecond(TLink source, TLink target, TLink
49             ↳ rootSource, TLink rootTarget);
50
51         [MethodImpl(MethodImplOptions.AggressiveInlining)]
52         protected abstract bool FirstIsToTheLeftOfSecond(TLink source, TLink target, TLink
53             ↳ rootSource, TLink rootTarget);
54
55         [MethodImpl(MethodImplOptions.AggressiveInlining)]
56         protected virtual ref LinksHeader<TLink> GetHeaderReference() => ref
57             ↳ AsRef<LinksHeader<TLink>>(Header);
58
59         [MethodImpl(MethodImplOptions.AggressiveInlining)]
60         protected virtual ref RawLink<TLink> GetLinkReference(TLink link) => ref
61             ↳ AsRef<RawLink<TLink>>(Links + (RawLink<TLink>.SizeInBytes *
62             ↳ _addressToInt64Converter.Convert(link)));
63
64         [MethodImpl(MethodImplOptions.AggressiveInlining)]
65         protected virtual IList<TLink> GetLinkValues(TLink linkIndex)
66         {
67             ref var link = ref GetLinkReference(linkIndex);
68             return new Link<TLink>(linkIndex, link.Source, link.Target);
69         }
70     }
71 }

```

```

60 [MethodImpl(MethodImplOptions.AggressiveInlining)]
61 protected override bool FirstIsToTheLeftOfSecond(TLink first, TLink second)
62 {
63     ref var firstLink = ref GetLinkReference(first);
64     ref var secondLink = ref GetLinkReference(second);
65     return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
        ↪ secondLink.Source, secondLink.Target);
66 }
67
68 [MethodImpl(MethodImplOptions.AggressiveInlining)]
69 protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second)
70 {
71     ref var firstLink = ref GetLinkReference(first);
72     ref var secondLink = ref GetLinkReference(second);
73     return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
        ↪ secondLink.Source, secondLink.Target);
74 }
75
76 [MethodImpl(MethodImplOptions.AggressiveInlining)]
77 protected virtual TLink GetSizeValue(TLink value) => Bit<TLink>.PartialRead(value, 5,
    ↪ -5);
78
79 [MethodImpl(MethodImplOptions.AggressiveInlining)]
80 protected virtual void SetSizeValue(ref TLink storedValue, TLink size) => storedValue =
    ↪ Bit<TLink>.PartialWrite(storedValue, size, 5, -5);
81
82 [MethodImpl(MethodImplOptions.AggressiveInlining)]
83 protected virtual bool GetLeftIsChildValue(TLink value)
84 {
85     unchecked
86     {
87         //return _addressToBoolConverter.Convert(Bit<TLink>.PartialRead(previousValue,
            ↪ 4, 1));
88         return !EqualityComparer.Equals(Bit<TLink>.PartialRead(value, 4, 1), default);
89     }
90 }
91
92 [MethodImpl(MethodImplOptions.AggressiveInlining)]
93 protected virtual void SetLeftIsChildValue(ref TLink storedValue, bool value)
94 {
95     unchecked
96     {
97         var previousValue = storedValue;
98         var modified = Bit<TLink>.PartialWrite(previousValue,
            ↪ _boolToAddressConverter.Convert(value), 4, 1);
99         storedValue = modified;
100     }
101 }
102
103 [MethodImpl(MethodImplOptions.AggressiveInlining)]
104 protected virtual bool GetRightIsChildValue(TLink value)
105 {
106     unchecked
107     {
108         //return _addressToBoolConverter.Convert(Bit<TLink>.PartialRead(previousValue,
            ↪ 3, 1));
109         return !EqualityComparer.Equals(Bit<TLink>.PartialRead(value, 3, 1), default);
110     }
111 }
112
113 [MethodImpl(MethodImplOptions.AggressiveInlining)]
114 protected virtual void SetRightIsChildValue(ref TLink storedValue, bool value)
115 {
116     unchecked
117     {
118         var previousValue = storedValue;
119         var modified = Bit<TLink>.PartialWrite(previousValue,
            ↪ _boolToAddressConverter.Convert(value), 3, 1);
120         storedValue = modified;
121     }
122 }
123
124 [MethodImpl(MethodImplOptions.AggressiveInlining)]
125 protected bool IsChild(TLink parent, TLink possibleChild)
126 {
127     var parentSize = GetSize(parent);
128     var childSize = GetSizeOrZero(possibleChild);
129     return GreaterThanZero(childSize) && LessOrEqualThan(childSize, parentSize);

```

```

130 }
131
132 [MethodImpl(MethodImplOptions.AggressiveInlining)]
133 protected virtual sbyte GetBalanceValue(TLink storedValue)
134 {
135     unchecked
136     {
137         var value = _addressToInt32Converter.Convert(Bit<TLink>.PartialRead(storedValue,
138             ↪ 0, 3));
139         value |= 0xF8 * ((value & 4) >> 2); // if negative, then continue ones to the
140             ↪ end of sbyte
141         return (sbyte)value;
142     }
143 }
144
145 [MethodImpl(MethodImplOptions.AggressiveInlining)]
146 protected virtual void SetBalanceValue(ref TLink storedValue, sbyte value)
147 {
148     unchecked
149     {
150         var packagedValue = _int32ToAddressConverter.Convert((byte)value >> 5 & 4 |
151             ↪ value & 3);
152         var modified = Bit<TLink>.PartialWrite(storedValue, packagedValue, 0, 3);
153         storedValue = modified;
154     }
155 }
156
157 public TLink this[TLink index]
158 {
159     [MethodImpl(MethodImplOptions.AggressiveInlining)]
160     get
161     {
162         var root = GetTreeRoot();
163         if (GreaterOrEqualThan(index, GetSize(root)))
164         {
165             return Zero;
166         }
167         while (!EqualToZero(root))
168         {
169             var left = GetLeftOrDefault(root);
170             var leftSize = GetSizeOrZero(left);
171             if (LessThan(index, leftSize))
172             {
173                 root = left;
174                 continue;
175             }
176             if (AreEqual(index, leftSize))
177             {
178                 return root;
179             }
180             root = GetRightOrDefault(root);
181             index = Subtract(index, Increment(leftSize));
182         }
183         return Zero; // TODO: Impossible situation exception (only if tree structure
184             ↪ broken)
185     }
186 }
187
188 /// <summary>
189 /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
190 ↪ (концом).
191 /// </summary>
192 /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
193 /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
194 /// <returns>Индекс искомой связи.</returns>
195 [MethodImpl(MethodImplOptions.AggressiveInlining)]
196 public TLink Search(TLink source, TLink target)
197 {
198     var root = GetTreeRoot();
199     while (!EqualToZero(root))
200     {
201         ref var rootLink = ref GetLinkReference(root);
202         var rootSource = rootLink.Source;
203         var rootTarget = rootLink.Target;
204         if (FirstIsToTheLeftOfSecond(source, target, rootSource, rootTarget)) //
205             ↪ node.Key < root.Key
206         {
207             root = GetLeftOrDefault(root);
208         }
209     }
210 }

```

```

202     }
203     else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
        ↪ node.Key > root.Key
204     {
205         root = GetRightOrDefault(root);
206     }
207     else // node.Key == root.Key
208     {
209         return root;
210     }
211 }
212 return Zero;
213 }
214
215 // TODO: Return indices range instead of references count
216 [MethodImpl(MethodImplOptions.AggressiveInlining)]
217 public TLink CountUsages(TLink link)
218 {
219     var root = GetTreeRoot();
220     var total = GetSize(root);
221     var totalRightIgnore = Zero;
222     while (!EqualToZero(root))
223     {
224         var @base = GetBasePartValue(root);
225         if (LessOrEqualThan(@base, link))
226         {
227             root = GetRightOrDefault(root);
228         }
229         else
230         {
231             totalRightIgnore = Add(totalRightIgnore, Increment(GetRightSize(root)));
232             root = GetLeftOrDefault(root);
233         }
234     }
235     root = GetTreeRoot();
236     var totalLeftIgnore = Zero;
237     while (!EqualToZero(root))
238     {
239         var @base = GetBasePartValue(root);
240         if (GreaterOrEqualThan(@base, link))
241         {
242             root = GetLeftOrDefault(root);
243         }
244         else
245         {
246             totalLeftIgnore = Add(totalLeftIgnore, Increment(GetLeftSize(root)));
247             root = GetRightOrDefault(root);
248         }
249     }
250     return Subtract(Subtract(total, totalRightIgnore), totalLeftIgnore);
251 }
252
253 [MethodImpl(MethodImplOptions.AggressiveInlining)]
254 public TLink EachUsage(TLink link, Func<IList<TLink>, TLink> handler)
255 {
256     var root = GetTreeRoot();
257     if (EqualToZero(root))
258     {
259         return Continue;
260     }
261     TLink first = Zero, current = root;
262     while (!EqualToZero(current))
263     {
264         var @base = GetBasePartValue(current);
265         if (GreaterOrEqualThan(@base, link))
266         {
267             if (AreEqual(@base, link))
268             {
269                 first = current;
270             }
271             current = GetLeftOrDefault(current);
272         }
273         else
274         {
275             current = GetRightOrDefault(current);
276         }
277     }
278     if (!EqualToZero(first))
279

```

```

280     {
281         current = first;
282         while (true)
283         {
284             if (AreEqual(handler(GetLinkValues(current)), Break))
285             {
286                 return Break;
287             }
288             current = GetNext(current);
289             if (EqualToZero(current) || !AreEqual(GetBasePartValue(current), link))
290             {
291                 break;
292             }
293         }
294     }
295     return Continue;
296 }
297
298 [MethodImpl(MethodImplOptions.AggressiveInlining)]
299 protected override void PrintNodeValue(TLink node, StringBuilder sb)
300 {
301     ref var link = ref GetLinkReference(node);
302     sb.Append(' ');
303     sb.Append(link.Source);
304     sb.Append('-');
305     sb.Append('>');
306     sb.Append(link.Target);
307 }
308 }
309 }

```

1.33 ./Platform.Data.Doublets/ResizableDirectMemory/Generic/LinksSizeBalancedTreeMethodsBase.cs

```

1  using System;
2  using System.Text;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5  using Platform.Collections.Methods.Trees;
6  using Platform.Converters;
7  using static System.Runtime.CompilerServices.Unsafe;
8
9  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
10
11 namespace Platform.Data.Doublets.ResizableDirectMemory.Generic
12 {
13     public unsafe abstract class LinksSizeBalancedTreeMethodsBase<TLink> :
14         ↳ SizeBalancedTreeMethods<TLink>, ILinksTreeMethods<TLink>
15     {
16         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
17             ↳ UncheckedConverter<TLink, long>.Default;
18
19         protected readonly TLink Break;
20         protected readonly TLink Continue;
21         protected readonly byte* Links;
22         protected readonly byte* Header;
23
24         [MethodImpl(MethodImplOptions.AggressiveInlining)]
25         protected LinksSizeBalancedTreeMethodsBase(LinksConstants<TLink> constants, byte* links,
26             ↳ byte* header)
27         {
28             Links = links;
29             Header = header;
30             Break = constants.Break;
31             Continue = constants.Continue;
32         }
33
34         [MethodImpl(MethodImplOptions.AggressiveInlining)]
35         protected abstract TLink GetTreeRoot();
36
37         [MethodImpl(MethodImplOptions.AggressiveInlining)]
38         protected abstract TLink GetBasePartValue(TLink link);
39
40         [MethodImpl(MethodImplOptions.AggressiveInlining)]
41         protected abstract bool FirstIsToTheRightOfSecond(TLink source, TLink target, TLink
42             ↳ rootSource, TLink rootTarget);
43
44         [MethodImpl(MethodImplOptions.AggressiveInlining)]
45         protected abstract bool FirstIsToTheLeftOfSecond(TLink source, TLink target, TLink
46             ↳ rootSource, TLink rootTarget);
47
48         [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

44     protected virtual ref LinksHeader<TLink> GetHeaderReference() => ref
45         ↪ AsRef<LinksHeader<TLink>>(Header);
46
47     [MethodImpl(MethodImplOptions.AggressiveInlining)]
48     protected virtual ref RawLink<TLink> GetLinkReference(TLink link) => ref
49         ↪ AsRef<RawLink<TLink>>(Links + (RawLink<TLink>.SizeInBytes *
50         ↪ _addressToInt64Converter.Convert(link)));
51
52     [MethodImpl(MethodImplOptions.AggressiveInlining)]
53     protected virtual IList<TLink> GetLinkValues(TLink linkIndex)
54     {
55         ref var link = ref GetLinkReference(linkIndex);
56         return new Link<TLink>(linkIndex, link.Source, link.Target);
57     }
58
59     [MethodImpl(MethodImplOptions.AggressiveInlining)]
60     protected override bool FirstIsToTheLeftOfSecond(TLink first, TLink second)
61     {
62         ref var firstLink = ref GetLinkReference(first);
63         ref var secondLink = ref GetLinkReference(second);
64         return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
65         ↪ secondLink.Source, secondLink.Target);
66     }
67
68     [MethodImpl(MethodImplOptions.AggressiveInlining)]
69     protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second)
70     {
71         ref var firstLink = ref GetLinkReference(first);
72         ref var secondLink = ref GetLinkReference(second);
73         return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
74         ↪ secondLink.Source, secondLink.Target);
75     }
76
77     public TLink this[TLink index]
78     {
79         [MethodImpl(MethodImplOptions.AggressiveInlining)]
80         get
81         {
82             var root = GetTreeRoot();
83             if (GreaterOrEqualThan(index, GetSize(root)))
84             {
85                 return Zero;
86             }
87             while (!EqualToZero(root))
88             {
89                 var left = GetLeftOrDefault(root);
90                 var leftSize = GetSizeOrZero(left);
91                 if (LessThan(index, leftSize))
92                 {
93                     root = left;
94                     continue;
95                 }
96                 if (AreEqual(index, leftSize))
97                 {
98                     return root;
99                 }
100                 root = GetRightOrDefault(root);
101                 index = Subtract(index, Increment(leftSize));
102             }
103             return Zero; // TODO: Impossible situation exception (only if tree structure
104             ↪ broken)
105         }
106     }
107
108     /// <summary>
109     /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
110     ↪ (концом).
111     /// </summary>
112     /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
113     /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
114     /// <returns>Индекс искомой связи.</returns>
115     [MethodImpl(MethodImplOptions.AggressiveInlining)]
116     public TLink Search(TLink source, TLink target)
117     {
118         var root = GetTreeRoot();
119         while (!EqualToZero(root))
120         {
121             ref var rootLink = ref GetLinkReference(root);

```



```

115     var rootSource = rootLink.Source;
116     var rootTarget = rootLink.Target;
117     if (FirstIsToTheLeftOfSecond(source, target, rootSource, rootTarget)) //
118         ↪ node.Key < root.Key
119     {
120         root = GetLeftOrDefault(root);
121     }
122     else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
123         ↪ node.Key > root.Key
124     {
125         root = GetRightOrDefault(root);
126     }
127     else // node.Key == root.Key
128     {
129         return root;
130     }
131 }
132 return Zero;
133 }
134 // TODO: Return indices range instead of references count
135 [MethodImpl(MethodImplOptions.AggressiveInlining)]
136 public TLink CountUsages(TLink link)
137 {
138     var root = GetTreeRoot();
139     var total = GetSize(root);
140     var totalRightIgnore = Zero;
141     while (!EqualToZero(root))
142     {
143         var @base = GetBasePartValue(root);
144         if (LessOrEqualThan(@base, link))
145         {
146             root = GetRightOrDefault(root);
147         }
148         else
149         {
150             totalRightIgnore = Add(totalRightIgnore, Increment(GetRightSize(root)));
151             root = GetLeftOrDefault(root);
152         }
153     }
154     root = GetTreeRoot();
155     var totalLeftIgnore = Zero;
156     while (!EqualToZero(root))
157     {
158         var @base = GetBasePartValue(root);
159         if (GreaterOrEqualThan(@base, link))
160         {
161             root = GetLeftOrDefault(root);
162         }
163         else
164         {
165             totalLeftIgnore = Add(totalLeftIgnore, Increment(GetLeftSize(root)));
166             root = GetRightOrDefault(root);
167         }
168     }
169     return Subtract(Subtract(total, totalRightIgnore), totalLeftIgnore);
170 }
171
172 [MethodImpl(MethodImplOptions.AggressiveInlining)]
173 public TLink EachUsage(TLink @base, Func<IList<TLink>, TLink> handler) =>
174     ↪ EachUsageCore(@base, GetTreeRoot(), handler);
175
176 // TODO: 1. Move target, handler to separate object. 2. Use stack or walker 3. Use
177     ↪ low-level MSIL stack.
178 [MethodImpl(MethodImplOptions.AggressiveInlining)]
179 private TLink EachUsageCore(TLink @base, TLink link, Func<IList<TLink>, TLink> handler)
180 {
181     var @continue = Continue;
182     if (EqualToZero(link))
183     {
184         return @continue;
185     }
186     var linkBasePart = GetBasePartValue(link);
187     var @break = Break;
188     if (GreaterThan(linkBasePart, @base))
189     {
190         if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
191         {

```

```

190         return @break;
191     }
192 }
193 else if (LessThan(linkBasePart, @base))
194 {
195     if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
196     {
197         return @break;
198     }
199 }
200 else //if (linkBasePart == @base)
201 {
202     if (AreEqual(handler(GetLinkValues(link)), @break))
203     {
204         return @break;
205     }
206     if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
207     {
208         return @break;
209     }
210     if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
211     {
212         return @break;
213     }
214 }
215 return @continue;
216 }
217
218 [MethodImpl(MethodImplOptions.AggressiveInlining)]
219 protected override void PrintNodeValue(TLink node, StringBuilder sb)
220 {
221     ref var link = ref GetLinkReference(node);
222     sb.Append(' ');
223     sb.Append(link.Source);
224     sb.Append('-');
225     sb.Append('>');
226     sb.Append(link.Target);
227 }
228 }
229 }

```

1.34 ./Platform.Data.Doublets/ResizableDirectMemory/Generic/LinksSourcesAvlBalancedTreeMethods.cs

```

1 using System.Runtime.CompilerServices;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Doublets.ResizableDirectMemory.Generic
6 {
7     public unsafe class LinksSourcesAvlBalancedTreeMethods<TLink> :
8         ↳ LinksAvlBalancedTreeMethodsBase<TLink>
9     {
10         [MethodImpl(MethodImplOptions.AggressiveInlining)]
11         public LinksSourcesAvlBalancedTreeMethods(LinksConstants<TLink> constants, byte* links,
12             ↳ byte* header) : base(constants, links, header) { }
13
14         [MethodImpl(MethodImplOptions.AggressiveInlining)]
15         protected unsafe override ref TLink GetLeftReference(TLink node) => ref
16             ↳ GetLinkReference(node).LeftAsSource;
17
18         [MethodImpl(MethodImplOptions.AggressiveInlining)]
19         protected unsafe override ref TLink GetRightReference(TLink node) => ref
20             ↳ GetLinkReference(node).RightAsSource;
21
22         [MethodImpl(MethodImplOptions.AggressiveInlining)]
23         protected override TLink GetLeft(TLink node) => GetLinkReference(node).LeftAsSource;
24
25         [MethodImpl(MethodImplOptions.AggressiveInlining)]
26         protected override TLink GetRight(TLink node) => GetLinkReference(node).RightAsSource;
27
28         [MethodImpl(MethodImplOptions.AggressiveInlining)]
29         protected override void SetLeft(TLink node, TLink left) =>
30             ↳ GetLinkReference(node).LeftAsSource = left;
31
32         [MethodImpl(MethodImplOptions.AggressiveInlining)]
33         protected override void SetRight(TLink node, TLink right) =>
34             ↳ GetLinkReference(node).RightAsSource = right;
35
36         [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

31     protected override TLink GetSize(TLink node) =>
32         ↳ GetSizeValue(GetLinkReference(node).SizeAsSource);
33
34     [MethodImpl(MethodImplOptions.AggressiveInlining)]
35     protected override void SetSize(TLink node, TLink size) => SetSizeValue(ref
36         ↳ GetLinkReference(node).SizeAsSource, size);
37
38     [MethodImpl(MethodImplOptions.AggressiveInlining)]
39     protected override bool GetLeftIsChild(TLink node) =>
40         ↳ GetLeftIsChildValue(GetLinkReference(node).SizeAsSource);
41
42     [MethodImpl(MethodImplOptions.AggressiveInlining)]
43     protected override void SetLeftIsChild(TLink node, bool value) =>
44         ↳ SetLeftIsChildValue(ref GetLinkReference(node).SizeAsSource, value);
45
46     [MethodImpl(MethodImplOptions.AggressiveInlining)]
47     protected override bool GetRightIsChild(TLink node) =>
48         ↳ GetRightIsChildValue(GetLinkReference(node).SizeAsSource);
49
50     [MethodImpl(MethodImplOptions.AggressiveInlining)]
51     protected override void SetRightIsChild(TLink node, bool value) =>
52         ↳ SetRightIsChildValue(ref GetLinkReference(node).SizeAsSource, value);
53
54     [MethodImpl(MethodImplOptions.AggressiveInlining)]
55     protected override sbyte GetBalance(TLink node) =>
56         ↳ GetBalanceValue(GetLinkReference(node).SizeAsSource);
57
58     [MethodImpl(MethodImplOptions.AggressiveInlining)]
59     protected override void SetBalance(TLink node, sbyte value) => SetBalanceValue(ref
60         ↳ GetLinkReference(node).SizeAsSource, value);
61
62     [MethodImpl(MethodImplOptions.AggressiveInlining)]
63     protected override TLink GetTreeRoot() => GetHeaderReference().FirstAsSource;
64
65     [MethodImpl(MethodImplOptions.AggressiveInlining)]
66     protected override TLink GetBasePartValue(TLink link) => GetLinkReference(link).Source;
67
68     [MethodImpl(MethodImplOptions.AggressiveInlining)]
69     protected override bool FirstIsToTheLeftOfSecond(TLink firstSource, TLink firstTarget,
70         ↳ TLink secondSource, TLink secondTarget) => LessThan(firstSource, secondSource) ||
71         ↳ (AreEqual(firstSource, secondSource) && LessThan(firstTarget, secondTarget));
72
73     [MethodImpl(MethodImplOptions.AggressiveInlining)]
74     protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
75         ↳ TLink secondSource, TLink secondTarget) => GreaterThan(firstSource, secondSource) ||
76         ↳ (AreEqual(firstSource, secondSource) && GreaterThan(firstTarget, secondTarget));
77
78     [MethodImpl(MethodImplOptions.AggressiveInlining)]
79     protected override void ClearNode(TLink node)
80     {
81         ref var link = ref GetLinkReference(node);
82         link.LeftAsSource = Zero;
83         link.RightAsSource = Zero;
84         link.SizeAsSource = Zero;
85     }
86 }

```

1.35 ./Platform.Data.Doublets/ResizableDirectMemory/Generic/LinksSourcesSizeBalancedTreeMethods.cs

```

1 using System.Runtime.CompilerServices;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Doublets.ResizableDirectMemory.Generic
6 {
7     public unsafe class LinksSourcesSizeBalancedTreeMethods<TLink> :
8         ↳ LinksSizeBalancedTreeMethodsBase<TLink>
9     {
10         [MethodImpl(MethodImplOptions.AggressiveInlining)]
11         public LinksSourcesSizeBalancedTreeMethods(LinksConstants<TLink> constants, byte* links,
12             ↳ byte* header) : base(constants, links, header) { }
13
14         [MethodImpl(MethodImplOptions.AggressiveInlining)]
15         protected unsafe override ref TLink GetLeftReference(TLink node) => ref
16             ↳ GetLinkReference(node).LeftAsSource;
17
18         [MethodImpl(MethodImplOptions.AggressiveInlining)]
19         protected unsafe override ref TLink GetRightReference(TLink node) => ref
20             ↳ GetLinkReference(node).RightAsSource;
21     }
22 }

```

```

17     [MethodImpl(MethodImplOptions.AggressiveInlining)]
18     protected override TLink GetLeft(TLink node) => GetLinkReference(node).LeftAsSource;
19
20     [MethodImpl(MethodImplOptions.AggressiveInlining)]
21     protected override TLink GetRight(TLink node) => GetLinkReference(node).RightAsSource;
22
23     [MethodImpl(MethodImplOptions.AggressiveInlining)]
24     protected override void SetLeft(TLink node, TLink left) =>
25     ↪ GetLinkReference(node).LeftAsSource = left;
26
27     [MethodImpl(MethodImplOptions.AggressiveInlining)]
28     protected override void SetRight(TLink node, TLink right) =>
29     ↪ GetLinkReference(node).RightAsSource = right;
30
31     [MethodImpl(MethodImplOptions.AggressiveInlining)]
32     protected override TLink GetSize(TLink node) => GetLinkReference(node).SizeAsSource;
33
34     [MethodImpl(MethodImplOptions.AggressiveInlining)]
35     protected override void SetSize(TLink node, TLink size) =>
36     ↪ GetLinkReference(node).SizeAsSource = size;
37
38     [MethodImpl(MethodImplOptions.AggressiveInlining)]
39     protected override TLink GetTreeRoot() => GetHeaderReference().FirstAsSource;
40
41     [MethodImpl(MethodImplOptions.AggressiveInlining)]
42     protected override TLink GetBasePartValue(TLink link) => GetLinkReference(link).Source;
43
44     [MethodImpl(MethodImplOptions.AggressiveInlining)]
45     protected override bool FirstIsToTheLeftOfSecond(TLink firstSource, TLink firstTarget,
46     ↪ TLink secondSource, TLink secondTarget) => LessThan(firstSource, secondSource) ||
47     ↪ (AreEqual(firstSource, secondSource) && LessThan(firstTarget, secondTarget));
48
49     [MethodImpl(MethodImplOptions.AggressiveInlining)]
50     protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
51     ↪ TLink secondSource, TLink secondTarget) => GreaterThan(firstSource, secondSource) ||
52     ↪ (AreEqual(firstSource, secondSource) && GreaterThan(firstTarget, secondTarget));
53
54     [MethodImpl(MethodImplOptions.AggressiveInlining)]
55     protected override void ClearNode(TLink node)
56     {
57         ref var link = ref GetLinkReference(node);
58         link.LeftAsSource = Zero;
59         link.RightAsSource = Zero;
60         link.SizeAsSource = Zero;
61     }
62 }

```

1.36 ./Platform.Data.Doublets/ResizableDirectMemory/Generic/LinksTargetsAvlBalancedTreeMethods.cs

```

1 using System.Runtime.CompilerServices;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Doublets.ResizableDirectMemory.Generic
6 {
7     public unsafe class LinksTargetsAvlBalancedTreeMethods<TLink> :
8     ↪ LinksAvlBalancedTreeMethodsBase<TLink>
9     {
10         [MethodImpl(MethodImplOptions.AggressiveInlining)]
11         public LinksTargetsAvlBalancedTreeMethods(LinksConstants<TLink> constants, byte* links,
12         ↪ byte* header) : base(constants, links, header) { }
13
14         [MethodImpl(MethodImplOptions.AggressiveInlining)]
15         protected unsafe override ref TLink GetLeftReference(TLink node) => ref
16         ↪ GetLinkReference(node).LeftAsTarget;
17
18         [MethodImpl(MethodImplOptions.AggressiveInlining)]
19         protected unsafe override ref TLink GetRightReference(TLink node) => ref
20         ↪ GetLinkReference(node).RightAsTarget;
21
22         [MethodImpl(MethodImplOptions.AggressiveInlining)]
23         protected override TLink GetLeft(TLink node) => GetLinkReference(node).LeftAsTarget;
24
25         [MethodImpl(MethodImplOptions.AggressiveInlining)]
26         protected override TLink GetRight(TLink node) => GetLinkReference(node).RightAsTarget;
27
28         [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

25     protected override void SetLeft(TLink node, TLink left) =>
26         ↪ GetLinkReference(node).LeftAsTarget = left;
27
28     [MethodImpl(MethodImplOptions.AggressiveInlining)]
29     protected override void SetRight(TLink node, TLink right) =>
30         ↪ GetLinkReference(node).RightAsTarget = right;
31
32     [MethodImpl(MethodImplOptions.AggressiveInlining)]
33     protected override TLink GetSize(TLink node) =>
34         ↪ GetSizeValue(GetLinkReference(node).SizeAsTarget);
35
36     [MethodImpl(MethodImplOptions.AggressiveInlining)]
37     protected override void SetSize(TLink node, TLink size) => SetSizeValue(ref
38         ↪ GetLinkReference(node).SizeAsTarget, size);
39
40     [MethodImpl(MethodImplOptions.AggressiveInlining)]
41     protected override bool GetLeftIsChild(TLink node) =>
42         ↪ GetLeftIsChildValue(GetLinkReference(node).SizeAsTarget);
43
44     [MethodImpl(MethodImplOptions.AggressiveInlining)]
45     protected override void SetLeftIsChild(TLink node, bool value) =>
46         ↪ SetLeftIsChildValue(ref GetLinkReference(node).SizeAsTarget, value);
47
48     [MethodImpl(MethodImplOptions.AggressiveInlining)]
49     protected override bool GetRightIsChild(TLink node) =>
50         ↪ GetRightIsChildValue(GetLinkReference(node).SizeAsTarget);
51
52     [MethodImpl(MethodImplOptions.AggressiveInlining)]
53     protected override void SetRightIsChild(TLink node, bool value) =>
54         ↪ SetRightIsChildValue(ref GetLinkReference(node).SizeAsTarget, value);
55
56     [MethodImpl(MethodImplOptions.AggressiveInlining)]
57     protected override sbyte GetBalance(TLink node) =>
58         ↪ GetBalanceValue(GetLinkReference(node).SizeAsTarget);
59
60     [MethodImpl(MethodImplOptions.AggressiveInlining)]
61     protected override void SetBalance(TLink node, sbyte value) => SetBalanceValue(ref
62         ↪ GetLinkReference(node).SizeAsTarget, value);
63
64     [MethodImpl(MethodImplOptions.AggressiveInlining)]
65     protected override TLink GetTreeRoot() => GetHeaderReference().FirstAsTarget;
66
67     [MethodImpl(MethodImplOptions.AggressiveInlining)]
68     protected override TLink GetBasePartValue(TLink link) => GetLinkReference(link).Target;
69
70     [MethodImpl(MethodImplOptions.AggressiveInlining)]
71     protected override bool FirstIsToTheLeftOfSecond(TLink firstSource, TLink firstTarget,
72         ↪ TLink secondSource, TLink secondTarget) => LessThan(firstTarget, secondTarget) ||
73         ↪ (AreEqual(firstTarget, secondTarget) && LessThan(firstSource, secondSource));
74
75     [MethodImpl(MethodImplOptions.AggressiveInlining)]
76     protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
77         ↪ TLink secondSource, TLink secondTarget) => GreaterThan(firstTarget, secondTarget) ||
78         ↪ (AreEqual(firstTarget, secondTarget) && GreaterThan(firstSource, secondSource));
79
80     [MethodImpl(MethodImplOptions.AggressiveInlining)]
81     protected override void ClearNode(TLink node)
82     {
83         ref var link = ref GetLinkReference(node);
84         link.LeftAsTarget = Zero;
85         link.RightAsTarget = Zero;
86         link.SizeAsTarget = Zero;
87     }
88 }

```

1.37 ./Platform.Data.Doublets/ResizableDirectMemory/Generic/LinksTargetsSizeBalancedTreeMethods.cs

```

1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.ResizableDirectMemory.Generic
6  {
7      public unsafe class LinksTargetsSizeBalancedTreeMethods<TLink> :
8          ↪ LinksSizeBalancedTreeMethodsBase<TLink>
9      {
10         [MethodImpl(MethodImplOptions.AggressiveInlining)]
11         public LinksTargetsSizeBalancedTreeMethods(LinksConstants<TLink> constants, byte* links,
12             ↪ byte* header) : base(constants, links, header) { }
13     }
14 }

```

```

11     [MethodImpl(MethodImplOptions.AggressiveInlining)]
12     protected unsafe override ref TLink GetLeftReference(TLink node) => ref
13     ↪ GetLinkReference(node).LeftAsTarget;
14
15     [MethodImpl(MethodImplOptions.AggressiveInlining)]
16     protected unsafe override ref TLink GetRightReference(TLink node) => ref
17     ↪ GetLinkReference(node).RightAsTarget;
18
19     [MethodImpl(MethodImplOptions.AggressiveInlining)]
20     protected override TLink GetLeft(TLink node) => GetLinkReference(node).LeftAsTarget;
21
22     [MethodImpl(MethodImplOptions.AggressiveInlining)]
23     protected override TLink GetRight(TLink node) => GetLinkReference(node).RightAsTarget;
24
25     [MethodImpl(MethodImplOptions.AggressiveInlining)]
26     protected override void SetLeft(TLink node, TLink left) =>
27     ↪ GetLinkReference(node).LeftAsTarget = left;
28
29     [MethodImpl(MethodImplOptions.AggressiveInlining)]
30     protected override void SetRight(TLink node, TLink right) =>
31     ↪ GetLinkReference(node).RightAsTarget = right;
32
33     [MethodImpl(MethodImplOptions.AggressiveInlining)]
34     protected override TLink GetSize(TLink node) => GetLinkReference(node).SizeAsTarget;
35
36     [MethodImpl(MethodImplOptions.AggressiveInlining)]
37     protected override void SetSize(TLink node, TLink size) =>
38     ↪ GetLinkReference(node).SizeAsTarget = size;
39
40     [MethodImpl(MethodImplOptions.AggressiveInlining)]
41     protected override TLink GetTreeRoot() => GetHeaderReference().FirstAsTarget;
42
43     [MethodImpl(MethodImplOptions.AggressiveInlining)]
44     protected override bool FirstIsToTheLeftOfSecond(TLink firstSource, TLink firstTarget,
45     ↪ TLink secondSource, TLink secondTarget) => LessThan(firstTarget, secondTarget) ||
46     ↪ (AreEqual(firstTarget, secondTarget) && LessThan(firstSource, secondSource));
47
48     [MethodImpl(MethodImplOptions.AggressiveInlining)]
49     protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
50     ↪ TLink secondSource, TLink secondTarget) => GreaterThan(firstTarget, secondTarget) ||
51     ↪ (AreEqual(firstTarget, secondTarget) && GreaterThan(firstSource, secondSource));
52
53     [MethodImpl(MethodImplOptions.AggressiveInlining)]
54     protected override void ClearNode(TLink node)
55     {
56         ref var link = ref GetLinkReference(node);
57         link.LeftAsTarget = Zero;
58         link.RightAsTarget = Zero;
59         link.SizeAsTarget = Zero;
60     }
61 }

```

1.38 ./Platform.Data.Doublets/ResizableDirectMemory/Generic/ResizableDirectMemoryLinks.cs

```

1  using System;
2  using System.Runtime.CompilerServices;
3  using Platform.Singletons;
4  using Platform.Memory;
5  using static System.Runtime.CompilerServices.Unsafe;
6
7  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9  namespace Platform.Data.Doublets.ResizableDirectMemory.Generic
10 {
11     public unsafe partial class ResizableDirectMemoryLinks<TLink> :
12     ↪ ResizableDirectMemoryLinksBase<TLink>
13     {
14         private readonly Func<ILinksTreeMethods<TLink>> _createSourceTreeMethods;
15         private readonly Func<ILinksTreeMethods<TLink>> _createTargetTreeMethods;
16         private byte* _header;
17         private byte* _links;
18
19         [MethodImpl(MethodImplOptions.AggressiveInlining)]
20         public ResizableDirectMemoryLinks(string address) : this(address, DefaultLinksSizeStep)
21         ↪ { }
22     }
23 }

```

```

20
21 /// <summary>
22 /// Создаёт экземпляр базы данных Links в файле по указанному адресу, с указанным
23   ↳ минимальным шагом расширения базы данных.
24 /// </summary>
25 /// <param name="address">Полный путь к файлу базы данных.</param>
26 /// <param name="memoryReservationStep">Минимальный шаг расширения базы данных в
27   ↳ байтах.</param>
28 [MethodImpl(MethodImplOptions.AggressiveInlining)]
29 public ResizableDirectMemoryLinks(string address, long memoryReservationStep) : this(new
30   ↳ FileMappedResizableDirectMemory(address, memoryReservationStep),
31   ↳ memoryReservationStep) { }
32
33 [MethodImpl(MethodImplOptions.AggressiveInlining)]
34 public ResizableDirectMemoryLinks(IResizableDirectMemory memory) : this(memory,
35   ↳ DefaultLinksSizeStep) { }
36
37 [MethodImpl(MethodImplOptions.AggressiveInlining)]
38 public ResizableDirectMemoryLinks(IResizableDirectMemory memory, long
39   ↳ memoryReservationStep) : this(memory, memoryReservationStep,
40   ↳ Default<LinksConstants<TLink>>.Instance, true) { }
41
42 [MethodImpl(MethodImplOptions.AggressiveInlining)]
43 public ResizableDirectMemoryLinks(IResizableDirectMemory memory, long
44   ↳ memoryReservationStep, LinksConstants<TLink> constants, bool useAvlBasedIndex) :
45   ↳ base(memory, memoryReservationStep, constants)
46 {
47     if (useAvlBasedIndex)
48     {
49         _createSourceTreeMethods = () => new
50             ↳ LinksSourcesAvlBalancedTreeMethods<TLink>(Constants, _links, _header);
51         _createTargetTreeMethods = () => new
52             ↳ LinksTargetsAvlBalancedTreeMethods<TLink>(Constants, _links, _header);
53     }
54     else
55     {
56         _createSourceTreeMethods = () => new
57             ↳ LinksSourcesSizeBalancedTreeMethods<TLink>(Constants, _links, _header);
58         _createTargetTreeMethods = () => new
59             ↳ LinksTargetsSizeBalancedTreeMethods<TLink>(Constants, _links, _header);
60     }
61     Init(memory, memoryReservationStep);
62 }
63
64 [MethodImpl(MethodImplOptions.AggressiveInlining)]
65 protected override void SetPointers(IResizableDirectMemory memory)
66 {
67     _links = (byte*)memory.Pointer;
68     _header = _links;
69     SourcesTreeMethods = _createSourceTreeMethods();
70     TargetsTreeMethods = _createTargetTreeMethods();
71     UnusedLinksListMethods = new UnusedLinksListMethods<TLink>(_links, _header);
72 }
73
74 [MethodImpl(MethodImplOptions.AggressiveInlining)]
75 protected override void ResetPointers()
76 {
77     base.ResetPointers();
78     _links = null;
79     _header = null;
80 }
81
82 [MethodImpl(MethodImplOptions.AggressiveInlining)]
83 protected override ref LinksHeader<TLink> GetHeaderReference() => ref
84   ↳ AsRef<LinksHeader<TLink>>(_header);
85
86 [MethodImpl(MethodImplOptions.AggressiveInlining)]
87 protected override ref RawLink<TLink> GetLinkReference(TLink linkIndex) => ref
88   ↳ AsRef<RawLink<TLink>>(_links + (LinkSizeInBytes * Convert.ToInt64(linkIndex)));
89 }
90 }

```

1.39 ./Platform.Data.Doublets/ResizableDirectMemory/Generic/ResizableDirectMemoryLinksBase.cs

```

1 using System;
2 using System.Collections.Generic;
3 using System.Runtime.CompilerServices;
4 using Platform.Disposables;
5 using Platform.Singletons;

```

```

6 using Platform.Converters;
7 using Platform.Numbers;
8 using Platform.Memory;
9 using Platform.Data.Exceptions;
10
11 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
12
13 namespace Platform.Data.Doublets.ResizableDirectMemory.Generic
14 {
15     public abstract class ResizableDirectMemoryLinksBase<TLink> : DisposableBase, ILinks<TLink>
16     {
17         private static readonly EqualityComparer<TLink> _equalityComparer =
18             ↳ EqualityComparer<TLink>.Default;
19         private static readonly Comparer<TLink> _comparer = Comparer<TLink>.Default;
20         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
21             ↳ UncheckedConverter<TLink, long>.Default;
22         private static readonly UncheckedConverter<long, TLink> _int64ToAddressConverter =
23             ↳ UncheckedConverter<long, TLink>.Default;
24
25         private static readonly TLink _zero = default;
26         private static readonly TLink _one = Arithmetic.Increment(_zero);
27
28         /// <summary>Возвращает размер одной связи в байтах.</summary>
29         /// <remarks>
30         ///     Используется только во вне класса, не рекомендуется использовать внутри.
31         ///     Так как во вне не обязательно будет доступен unsafe C#.
32         /// </remarks>
33         public static readonly long LinkSizeInBytes = RawLink<TLink>.SizeInBytes;
34
35         public static readonly long LinkHeaderSizeInBytes = LinksHeader<TLink>.SizeInBytes;
36
37         public static readonly long DefaultLinksSizeStep = LinkSizeInBytes * 1024 * 1024;
38
39         protected readonly IResizableDirectMemory _memory;
40         protected readonly long _memoryReservationStep;
41
42         protected ILinksTreeMethods<TLink> TargetsTreeMethods;
43         protected ILinksTreeMethods<TLink> SourcesTreeMethods;
44         // TODO: Возможно чтобы гарантированно проверять на то, является ли связь удалённой,
45         // ↳ нужно использовать не список а дерево, так как так можно быстрее проверить на
46         // ↳ наличие связи внутри
47         protected ILinksListMethods<TLink> UnusedLinksListMethods;
48
49         /// <summary>
50         ///     Возвращает общее число связей находящихся в хранилище.
51         /// </summary>
52         protected virtual TLink Total
53         {
54             [MethodImpl(MethodImplOptions.AggressiveInlining)]
55             get
56             {
57                 ref var header = ref GetHeaderReference();
58                 return Subtract(header.AllocatedLinks, header.FreeLinks);
59             }
60         }
61
62         public virtual LinksConstants<TLink> Constants
63         {
64             [MethodImpl(MethodImplOptions.AggressiveInlining)]
65             get;
66         }
67
68         [MethodImpl(MethodImplOptions.AggressiveInlining)]
69         protected ResizableDirectMemoryLinksBase(IResizableDirectMemory memory, long
70             ↳ memoryReservationStep, LinksConstants<TLink> constants)
71         {
72             _memory = memory;
73             _memoryReservationStep = memoryReservationStep;
74             Constants = constants;
75         }
76
77         [MethodImpl(MethodImplOptions.AggressiveInlining)]
78         protected ResizableDirectMemoryLinksBase(IResizableDirectMemory memory, long
79             ↳ memoryReservationStep) : this(memory, memoryReservationStep,
80             ↳ Default<LinksConstants<TLink>>.Instance) { }
81
82         [MethodImpl(MethodImplOptions.AggressiveInlining)]
83         protected virtual void Init(IResizableDirectMemory memory, long memoryReservationStep)
84         {
85             if (memory.ReservedCapacity < memoryReservationStep)
86             {

```



```

79         memory.ReservedCapacity = memoryReservationStep;
80     }
81     SetPointers(_memory);
82     ref var header = ref GetHeaderReference();
83     // Гарантия корректности _memory.UsedCapacity относительно _header->AllocatedLinks
84     _memory.UsedCapacity = (ConvertToInt64(header.AllocatedLinks) * LinkSizeInBytes) +
85     ↪ LinkHeaderSizeInBytes;
86     // Гарантия корректности _header->ReservedLinks относительно _memory.ReservedCapacity
87     header.ReservedLinks = ConvertToAddress((_memory.ReservedCapacity -
88     ↪ LinkHeaderSizeInBytes) / LinkSizeInBytes);
89 }
90 [MethodImpl(MethodImplOptions.AggressiveInlining)]
91 public virtual TLink Count(IList<TLink> restrictions)
92 {
93     // Если нет ограничений, тогда возвращаем общее число связей находящихся в хранилище.
94     if (restrictions.Count == 0)
95     {
96         return Total;
97     }
98     var constants = Constants;
99     var any = constants.Any;
100     var index = restrictions[constants.IndexPart];
101     if (restrictions.Count == 1)
102     {
103         if (AreEqual(index, any))
104         {
105             return Total;
106         }
107         return Exists(index) ? GetOne() : GetZero();
108     }
109     if (restrictions.Count == 2)
110     {
111         var value = restrictions[1];
112         if (AreEqual(index, any))
113         {
114             if (AreEqual(value, any))
115             {
116                 return Total; // Any - как отсутствие ограничения
117             }
118             return Add(SourcesTreeMethods.CountUsages(value),
119             ↪ TargetsTreeMethods.CountUsages(value));
120         }
121         else
122         {
123             if (!Exists(index))
124             {
125                 return GetZero();
126             }
127             if (AreEqual(value, any))
128             {
129                 return GetOne();
130             }
131             ref var storedLinkValue = ref GetLinkReference(index);
132             if (AreEqual(storedLinkValue.Source, value) ||
133             ↪ AreEqual(storedLinkValue.Target, value))
134             {
135                 return GetOne();
136             }
137             return GetZero();
138         }
139     }
140     if (restrictions.Count == 3)
141     {
142         var source = restrictions[constants.SourcePart];
143         var target = restrictions[constants.TargetPart];
144         if (AreEqual(index, any))
145         {
146             if (AreEqual(source, any) && AreEqual(target, any))
147             {
148                 return Total;
149             }
150             else if (AreEqual(source, any))
151             {
152                 return TargetsTreeMethods.CountUsages(target);
153             }
154             else if (AreEqual(target, any))
155             {
156

```

```

153         return SourcesTreeMethods.CountUsages(source);
154     }
155     else //if(source != Any && target != Any)
156     {
157         // Эквивалент Exists(source, target) => Count(Any, source, target) > 0
158         var link = SourcesTreeMethods.Search(source, target);
159         return AreEqual(link, constants.Null) ? GetZero() : GetOne();
160     }
161 }
162 else
163 {
164     if (!Exists(index))
165     {
166         return GetZero();
167     }
168     if (AreEqual(source, any) && AreEqual(target, any))
169     {
170         return GetOne();
171     }
172     ref var storedLinkValue = ref GetLinkReference(index);
173     if (!AreEqual(source, any) && !AreEqual(target, any))
174     {
175         if (AreEqual(storedLinkValue.Source, source) &&
176             ↪ AreEqual(storedLinkValue.Target, target))
177         {
178             return GetOne();
179         }
180         return GetZero();
181     }
182     var value = default(TLink);
183     if (AreEqual(source, any))
184     {
185         value = target;
186     }
187     if (AreEqual(target, any))
188     {
189         value = source;
190     }
191     if (AreEqual(storedLinkValue.Source, value) ||
192         ↪ AreEqual(storedLinkValue.Target, value))
193     {
194         return GetOne();
195     }
196     return GetZero();
197 }
198 }
199
200 [MethodImpl(MethodImplOptions.AggressiveInlining)]
201 public virtual TLink Each(Func<IList<TLink>, TLink> handler, IList<TLink> restrictions)
202 {
203     var constants = Constants;
204     var @break = constants.Break;
205     if (restrictions.Count == 0)
206     {
207         for (var link = GetOne(); LessOrEqualThan(link,
208             ↪ GetHeaderReference().AllocatedLinks); link = Increment(link))
209         {
210             if (Exists(link) && AreEqual(handler(GetLinkStruct(link)), @break))
211             {
212                 return @break;
213             }
214         }
215         return @break;
216     }
217     var @continue = constants.Continue;
218     var any = constants.Any;
219     var index = restrictions[constants.IndexPart];
220     if (restrictions.Count == 1)
221     {
222         if (AreEqual(index, any))
223         {
224             return Each(handler, GetEmptyList());
225         }
226         if (!Exists(index))
227         {

```

```

227         return @continue;
228     }
229     return handler(GetLinkStruct(index));
230 }
231 if (restrictions.Count == 2)
232 {
233     var value = restrictions[1];
234     if (AreEqual(index, any))
235     {
236         if (AreEqual(value, any))
237         {
238             return Each(handler, GetEmptyList());
239         }
240         if (AreEqual(Each(handler, new Link<TLink>(index, value, any)), @break))
241         {
242             return @break;
243         }
244         return Each(handler, new Link<TLink>(index, any, value));
245     }
246     else
247     {
248         if (!Exists(index))
249         {
250             return @continue;
251         }
252         if (AreEqual(value, any))
253         {
254             return handler(GetLinkStruct(index));
255         }
256         ref var storedLinkValue = ref GetLinkReference(index);
257         if (AreEqual(storedLinkValue.Source, value) ||
258             AreEqual(storedLinkValue.Target, value))
259         {
260             return handler(GetLinkStruct(index));
261         }
262         return @continue;
263     }
264 }
265 if (restrictions.Count == 3)
266 {
267     var source = restrictions[constants.SourcePart];
268     var target = restrictions[constants.TargetPart];
269     if (AreEqual(index, any))
270     {
271         if (AreEqual(source, any) && AreEqual(target, any))
272         {
273             return Each(handler, GetEmptyList());
274         }
275         else if (AreEqual(source, any))
276         {
277             return TargetsTreeMethods.EachUsage(target, handler);
278         }
279         else if (AreEqual(target, any))
280         {
281             return SourcesTreeMethods.EachUsage(source, handler);
282         }
283         else //if(source != Any && target != Any)
284         {
285             var link = SourcesTreeMethods.Search(source, target);
286             return AreEqual(link, constants.Null) ? @continue :
287                 ↪ handler(GetLinkStruct(link));
288         }
289     }
290     else
291     {
292         if (!Exists(index))
293         {
294             return @continue;
295         }
296         if (AreEqual(source, any) && AreEqual(target, any))
297         {
298             return handler(GetLinkStruct(index));
299         }
300         ref var storedLinkValue = ref GetLinkReference(index);
301         if (!AreEqual(source, any) && !AreEqual(target, any))
302         {
303             if (AreEqual(storedLinkValue.Source, source) &&
304                 AreEqual(storedLinkValue.Target, target))

```

```

304         {
305             return handler(GetLinkStruct(index));
306         }
307         return @continue;
308     }
309     var value = default(TLink);
310     if (AreEqual(source, any))
311     {
312         value = target;
313     }
314     if (AreEqual(target, any))
315     {
316         value = source;
317     }
318     if (AreEqual(storedLinkValue.Source, value) ||
319         AreEqual(storedLinkValue.Target, value))
320     {
321         return handler(GetLinkStruct(index));
322     }
323     return @continue;
324 }
325 }
326 throw new NotSupportedException("Другие размеры и способы ограничений не
    ↳ поддерживаются.");
327 }
328
329 /// <remarks>
330 /// TODO: Возможно можно перемещать значения, если указан индекс, но значение существует
331 ↳ в другом месте (но не в менеджере памяти, а в логике Links)
332 /// </remarks>
333 [MethodImpl(MethodImplOptions.AggressiveInlining)]
334 public virtual TLink Update(IList<TLink> restrictions, IList<TLink> substitution)
335 {
336     var constants = Constants;
337     var @null = constants.Null;
338     var linkIndex = restrictions[constants.IndexPart];
339     ref var link = ref GetLinkReference(linkIndex);
340     ref var header = ref GetHeaderReference();
341     ref var firstAsSource = ref header.FirstAsSource;
342     ref var firstAsTarget = ref header.FirstAsTarget;
343     // Будет корректно работать только в том случае, если пространство выделенной связи
344     ↳ предварительно заполнено нулями
345     if (!AreEqual(link.Source, @null))
346     {
347         SourcesTreeMethods.Detach(ref firstAsSource, linkIndex);
348     }
349     if (!AreEqual(link.Target, @null))
350     {
351         TargetsTreeMethods.Detach(ref firstAsTarget, linkIndex);
352     }
353     link.Source = substitution[constants.SourcePart];
354     link.Target = substitution[constants.TargetPart];
355     if (!AreEqual(link.Source, @null))
356     {
357         SourcesTreeMethods.Attach(ref firstAsSource, linkIndex);
358     }
359     if (!AreEqual(link.Target, @null))
360     {
361         TargetsTreeMethods.Attach(ref firstAsTarget, linkIndex);
362     }
363     return linkIndex;
364 }
365
366 /// <remarks>
367 /// TODO: Возможно нужно будет заполнение нулями, если внешнее API ими не заполняет
368 ↳ пространство
369 /// </remarks>
370 [MethodImpl(MethodImplOptions.AggressiveInlining)]
371 public virtual TLink Create(IList<TLink> restrictions)
372 {
373     ref var header = ref GetHeaderReference();
374     var freeLink = header.FirstFreeLink;
375     if (!AreEqual(freeLink, Constants.Null))
376     {
377         UnusedLinksListMethods.Detach(freeLink);
378     }
379     else
380     {
381         var maximumPossibleInnerReference = Constants.InternalReferencesRange.Maximum;

```

```

379         if (GreaterThan(header.AllocatedLinks, maximumPossibleInnerReference))
380         {
381             throw new LinksLimitReachedException<TLink>(maximumPossibleInnerReference);
382         }
383         if (GreaterOrEqualThan(header.AllocatedLinks, Decrement(header.ReservedLinks)))
384         {
385             _memory.ReservedCapacity += _memory.ReservationStep;
386             SetPointers(_memory);
387             header.ReservedLinks = ConvertToAddress(_memory.ReservedCapacity /
388                 ↳ LinkSizeInBytes);
389         }
390         header.AllocatedLinks = Increment(header.AllocatedLinks);
391         _memory.UsedCapacity += LinkSizeInBytes;
392         freeLink = header.AllocatedLinks;
393     }
394     return freeLink;
395 }
396 [MethodImpl(MethodImplOptions.AggressiveInlining)]
397 public virtual void Delete(ICollection<TLink> restrictions)
398 {
399     ref var header = ref GetHeaderReference();
400     var link = restrictions[Constants.IndexPart];
401     if (LessThan(link, header.AllocatedLinks))
402     {
403         UnusedLinksListMethods.AttachAsFirst(link);
404     }
405     else if (AreEqual(link, header.AllocatedLinks))
406     {
407         header.AllocatedLinks = Decrement(header.AllocatedLinks);
408         _memory.UsedCapacity -= LinkSizeInBytes;
409         // Убираем все связи, находящиеся в списке свободных в конце файла, до тех пор,
410         ↳ пока не дойдём до первой существующей связи
411         // Позволяет оптимизировать количество выделенных связей (AllocatedLinks)
412         while (GreaterThan(header.AllocatedLinks, GetZero()) &&
413             ↳ IsUnusedLink(header.AllocatedLinks))
414         {
415             UnusedLinksListMethods.Detach(header.AllocatedLinks);
416             header.AllocatedLinks = Decrement(header.AllocatedLinks);
417             _memory.UsedCapacity -= LinkSizeInBytes;
418         }
419     }
420 }
421 [MethodImpl(MethodImplOptions.AggressiveInlining)]
422 public IList<TLink> GetLinkStruct(TLink linkIndex)
423 {
424     ref var link = ref GetLinkReference(linkIndex);
425     return new Link<TLink>(linkIndex, link.Source, link.Target);
426 }
427 /// <remarks>
428 /// TODO: Возможно это должно быть событием, вызываемым из IMemory, в том случае, если
429 ↳ адрес реально поменялся
430 ///
431 /// Указатель this.links может быть в том же месте,
432 /// так как 0-я связь не используется и имеет такой же размер как Header,
433 /// поэтому header размещается в том же месте, что и 0-я связь
434 /// </remarks>
435 [MethodImpl(MethodImplOptions.AggressiveInlining)]
436 protected abstract void SetPointers(IResizableDirectMemory memory);
437 [MethodImpl(MethodImplOptions.AggressiveInlining)]
438 protected virtual void ResetPointers()
439 {
440     SourcesTreeMethods = null;
441     TargetsTreeMethods = null;
442     UnusedLinksListMethods = null;
443 }
444 [MethodImpl(MethodImplOptions.AggressiveInlining)]
445 protected abstract ref LinksHeader<TLink> GetHeaderReference();
446 [MethodImpl(MethodImplOptions.AggressiveInlining)]
447 protected abstract ref RawLink<TLink> GetLinkReference(TLink linkIndex);
448 [MethodImpl(MethodImplOptions.AggressiveInlining)]
449 protected virtual bool Exists(TLink link)
450 => GreaterOrEqualThan(link, Constants.InternalReferencesRange.Minimum)

```

```

454         && LessOrEqualThan(link, GetHeaderReference().AllocatedLinks)
455         && !IsUnusedLink(link);
456
457     [MethodImpl(MethodImplOptions.AggressiveInlining)]
458     protected virtual bool IsUnusedLink(TLink linkIndex)
459     {
460         if (!AreEqual(GetHeaderReference().FirstFreeLink, linkIndex)) // May be this check
461             ↪ is not needed
462         {
463             ref var link = ref GetLinkReference(linkIndex);
464             return AreEqual(link.SizeAsSource, default) && !AreEqual(link.Source, default);
465         }
466         else
467         {
468             return true;
469         }
470     }
471
472     [MethodImpl(MethodImplOptions.AggressiveInlining)]
473     protected virtual TLink GetOne() => _one;
474
475     [MethodImpl(MethodImplOptions.AggressiveInlining)]
476     protected virtual TLink GetZero() => default;
477
478     [MethodImpl(MethodImplOptions.AggressiveInlining)]
479     protected virtual bool AreEqual(TLink first, TLink second) =>
480         ↪ _equalityComparer.Equals(first, second);
481
482     [MethodImpl(MethodImplOptions.AggressiveInlining)]
483     protected virtual bool LessThan(TLink first, TLink second) => _comparer.Compare(first,
484         ↪ second) < 0;
485
486     [MethodImpl(MethodImplOptions.AggressiveInlining)]
487     protected virtual bool LessOrEqualThan(TLink first, TLink second) =>
488         ↪ _comparer.Compare(first, second) <= 0;
489
490     [MethodImpl(MethodImplOptions.AggressiveInlining)]
491     protected virtual bool GreaterThan(TLink first, TLink second) =>
492         ↪ _comparer.Compare(first, second) > 0;
493
494     [MethodImpl(MethodImplOptions.AggressiveInlining)]
495     protected virtual bool GreaterOrEqualThan(TLink first, TLink second) =>
496         ↪ _comparer.Compare(first, second) >= 0;
497
498     [MethodImpl(MethodImplOptions.AggressiveInlining)]
499     protected virtual long ConvertToInt64(TLink value) =>
500         ↪ _addressToInt64Converter.Convert(value);
501
502     [MethodImpl(MethodImplOptions.AggressiveInlining)]
503     protected virtual TLink ConvertToAddress(long value) =>
504         ↪ _int64ToAddressConverter.Convert(value);
505
506     [MethodImpl(MethodImplOptions.AggressiveInlining)]
507     protected virtual TLink Add(TLink first, TLink second) => Arithmetic<TLink>.Add(first,
508         ↪ second);
509
510     [MethodImpl(MethodImplOptions.AggressiveInlining)]
511     protected virtual TLink Subtract(TLink first, TLink second) =>
512         ↪ Arithmetic<TLink>.Subtract(first, second);
513
514     [MethodImpl(MethodImplOptions.AggressiveInlining)]
515     protected virtual TLink Increment(TLink link) => Arithmetic<TLink>.Increment(link);
516
517     [MethodImpl(MethodImplOptions.AggressiveInlining)]
518     protected virtual TLink Decrement(TLink link) => Arithmetic<TLink>.Decrement(link);
519
520     [MethodImpl(MethodImplOptions.AggressiveInlining)]
521     protected virtual IList<TLink> GetEmptyList() => Array.Empty<TLink>();
522
523     #region Disposable
524
525     protected override bool AllowMultipleDisposeCalls
526     {
527         [MethodImpl(MethodImplOptions.AggressiveInlining)]
528         get => true;
529     }
530
531     [MethodImpl(MethodImplOptions.AggressiveInlining)]
532     protected override void Dispose(bool manual, bool wasDisposed)

```

```

523     {
524         if (!wasDisposed)
525         {
526             ResetPointers();
527             _memory.DisposeIfPossible();
528         }
529     }
530
531     #endregion
532 }
533 }

```

1.40 ./Platform.Data.Doublets/ResizableDirectMemory/Generic/UnusedLinksListMethods.cs

```

1  using System.Runtime.CompilerServices;
2  using Platform.Collections.Methods.Lists;
3  using Platform.Converters;
4  using static System.Runtime.CompilerServices.Unsafe;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.ResizableDirectMemory.Generic
9  {
10     public unsafe class UnusedLinksListMethods<TLink> : CircularDoublyLinkedListMethods<TLink>,
11         ↳ ILinksListMethods<TLink>
12     {
13         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
14             ↳ UncheckedConverter<TLink, long>.Default;
15
16         private readonly byte* _links;
17         private readonly byte* _header;
18
19         [MethodImpl(MethodImplOptions.AggressiveInlining)]
20         public UnusedLinksListMethods(byte* links, byte* header)
21         {
22             _links = links;
23             _header = header;
24         }
25
26         [MethodImpl(MethodImplOptions.AggressiveInlining)]
27         protected virtual ref LinksHeader<TLink> GetHeaderReference() => ref
28             ↳ AsRef<LinksHeader<TLink>>(_header);
29
30         [MethodImpl(MethodImplOptions.AggressiveInlining)]
31         protected virtual ref RawLink<TLink> GetLinkReference(TLink link) => ref
32             ↳ AsRef<RawLink<TLink>>(_links + (RawLink<TLink>.SizeInBytes *
33             ↳ _addressToInt64Converter.Convert(link)));
34
35         [MethodImpl(MethodImplOptions.AggressiveInlining)]
36         protected override TLink GetFirst() => GetHeaderReference().FirstFreeLink;
37
38         [MethodImpl(MethodImplOptions.AggressiveInlining)]
39         protected override TLink GetLast() => GetHeaderReference().LastFreeLink;
40
41         [MethodImpl(MethodImplOptions.AggressiveInlining)]
42         protected override TLink GetPrevious(TLink element) => GetLinkReference(element).Source;
43
44         [MethodImpl(MethodImplOptions.AggressiveInlining)]
45         protected override TLink GetNext(TLink element) => GetLinkReference(element).Target;
46
47         [MethodImpl(MethodImplOptions.AggressiveInlining)]
48         protected override TLink GetSize() => GetHeaderReference().FreeLinks;
49
50         [MethodImpl(MethodImplOptions.AggressiveInlining)]
51         protected override void SetFirst(TLink element) => GetHeaderReference().FirstFreeLink =
52             ↳ element;
53
54         [MethodImpl(MethodImplOptions.AggressiveInlining)]
55         protected override void SetLast(TLink element) => GetHeaderReference().LastFreeLink =
56             ↳ element;
57
58         [MethodImpl(MethodImplOptions.AggressiveInlining)]
59         protected override void SetPrevious(TLink element, TLink previous) =>
60             ↳ GetLinkReference(element).Source = previous;
61
62         [MethodImpl(MethodImplOptions.AggressiveInlining)]
63         protected override void SetNext(TLink element, TLink next) =>
64             ↳ GetLinkReference(element).Target = next;
65
66         [MethodImpl(MethodImplOptions.AggressiveInlining)]
67         protected override void SetSize(TLink size) => GetHeaderReference().FreeLinks = size;

```

```

59     }
60 }

```

1.41 ./Platform.Data.Doublets/ResizableDirectMemory/ILinksListMethods.cs

```

1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.ResizableDirectMemory
6  {
7      public interface ILinksListMethods<TLink>
8      {
9          [MethodImpl(MethodImplOptions.AggressiveInlining)]
10         void Detach(TLink freeLink);
11
12         [MethodImpl(MethodImplOptions.AggressiveInlining)]
13         void AttachAsFirst(TLink link);
14     }
15 }

```

1.42 ./Platform.Data.Doublets/ResizableDirectMemory/ILinksTreeMethods.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.ResizableDirectMemory
8  {
9      public interface ILinksTreeMethods<TLink>
10     {
11         [MethodImpl(MethodImplOptions.AggressiveInlining)]
12         TLink CountUsages(TLink link);
13
14         [MethodImpl(MethodImplOptions.AggressiveInlining)]
15         TLink Search(TLink source, TLink target);
16
17         [MethodImpl(MethodImplOptions.AggressiveInlining)]
18         TLink EachUsage(TLink source, Func<IList<TLink>, TLink> handler);
19
20         [MethodImpl(MethodImplOptions.AggressiveInlining)]
21         void Detach(ref TLink firstAsSource, TLink linkIndex);
22
23         [MethodImpl(MethodImplOptions.AggressiveInlining)]
24         void Attach(ref TLink firstAsSource, TLink linkIndex);
25     }
26 }

```

1.43 ./Platform.Data.Doublets/ResizableDirectMemory/LinksHeader.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Unsafe;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.ResizableDirectMemory
9  {
10     public struct LinksHeader<TLink> : IEquatable<LinksHeader<TLink>>
11     {
12         private static readonly EqualityComparer<TLink> _equalityComparer =
13             ⇨ EqualityComparer<TLink>.Default;
14
15         public static readonly long SizeInBytes = Structure<LinksHeader<TLink>>.Size;
16
17         public TLink AllocatedLinks;
18         public TLink ReservedLinks;
19         public TLink FreeLinks;
20         public TLink FirstFreeLink;
21         public TLink FirstAsSource;
22         public TLink FirstAsTarget;
23         public TLink LastFreeLink;
24         public TLink Reserved8;
25
26         [MethodImpl(MethodImplOptions.AggressiveInlining)]
27         public override bool Equals(object obj) => obj is LinksHeader<TLink> linksHeader ?
28             ⇨ Equals(linksHeader) : false;
29
30         [MethodImpl(MethodImplOptions.AggressiveInlining)]
31         public bool Equals(LinksHeader<TLink> other)

```



```

30         => _equalityComparer.Equals(AllocatedLinks, other.AllocatedLinks)
31         && _equalityComparer.Equals(ReservedLinks, other.ReservedLinks)
32         && _equalityComparer.Equals(FreeLinks, other.FreeLinks)
33         && _equalityComparer.Equals(FirstFreeLink, other.FirstFreeLink)
34         && _equalityComparer.Equals(FirstAsSource, other.FirstAsSource)
35         && _equalityComparer.Equals(FirstAsTarget, other.FirstAsTarget)
36         && _equalityComparer.Equals>LastFreeLink, other.LastFreeLink)
37         && _equalityComparer.Equals(Reserved8, other.Reserved8);
38
39     [MethodImpl(MethodImplOptions.AggressiveInlining)]
40     public override int GetHashCode() => (AllocatedLinks, ReservedLinks, FreeLinks,
41     ↪ FirstFreeLink, FirstAsSource, FirstAsTarget, LastFreeLink, Reserved8).GetHashCode();
42
43     [MethodImpl(MethodImplOptions.AggressiveInlining)]
44     public static bool operator ==(LinksHeader<TLink> left, LinksHeader<TLink> right) =>
45     ↪ left.Equals(right);
46
47     [MethodImpl(MethodImplOptions.AggressiveInlining)]
48     public static bool operator !=(LinksHeader<TLink> left, LinksHeader<TLink> right) =>
49     ↪ !(left == right);
50 }

```

1.44 ./Platform.Data.Doublets/ResizableDirectMemory/RawLink.cs

```

1  using Platform.Unsafe;
2  using System;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.ResizableDirectMemory
9  {
10     public struct RawLink<TLink> : IEquatable<RawLink<TLink>>
11     {
12         private static readonly EqualityComparer<TLink> _equalityComparer =
13         ↪ EqualityComparer<TLink>.Default;
14
15         public static readonly long SizeInBytes = Structure<RawLink<TLink>>.Size;
16
17         public TLink Source;
18         public TLink Target;
19         public TLink LeftAsSource;
20         public TLink RightAsSource;
21         public TLink SizeAsSource;
22         public TLink LeftAsTarget;
23         public TLink RightAsTarget;
24         public TLink SizeAsTarget;
25
26         [MethodImpl(MethodImplOptions.AggressiveInlining)]
27         public override bool Equals(object obj) => obj is RawLink<TLink> link ? Equals(link) :
28         ↪ false;
29
30         [MethodImpl(MethodImplOptions.AggressiveInlining)]
31         public bool Equals(RawLink<TLink> other)
32         => _equalityComparer.Equals(Source, other.Source)
33         && _equalityComparer.Equals(Target, other.Target)
34         && _equalityComparer.Equals(LeftAsSource, other.LeftAsSource)
35         && _equalityComparer.Equals(RightAsSource, other.RightAsSource)
36         && _equalityComparer.Equals(SizeAsSource, other.SizeAsSource)
37         && _equalityComparer.Equals(LeftAsTarget, other.LeftAsTarget)
38         && _equalityComparer.Equals(RightAsTarget, other.RightAsTarget)
39         && _equalityComparer.Equals(SizeAsTarget, other.SizeAsTarget);
40
41         [MethodImpl(MethodImplOptions.AggressiveInlining)]
42         public override int GetHashCode() => (Source, Target, LeftAsSource, RightAsSource,
43         ↪ SizeAsSource, LeftAsTarget, RightAsTarget, SizeAsTarget).GetHashCode();
44
45         [MethodImpl(MethodImplOptions.AggressiveInlining)]
46         public static bool operator ==(RawLink<TLink> left, RawLink<TLink> right) =>
47         ↪ left.Equals(right);
48
49         [MethodImpl(MethodImplOptions.AggressiveInlining)]
50         public static bool operator !=(RawLink<TLink> left, RawLink<TLink> right) => !(left ==
51         ↪ right);
52     }
53 }

```

1.45 ./Platform.Data.Doublets/ResizableDirectMemory/Specific/UInt64LinksAvlBalancedTreeMethodsBase.cs

```

1  using System.Runtime.CompilerServices;
2  using Platform.Data.Doublets.ResizableDirectMemory.Generic;
3  using static System.Runtime.CompilerServices.Unsafe;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.ResizableDirectMemory.Specific
8  {
9      public unsafe abstract class UInt64LinksAvlBalancedTreeMethodsBase :
10         ↳ LinksAvlBalancedTreeMethodsBase<ulong>
11     {
12         protected new readonly RawLink<ulong>* Links;
13         protected new readonly LinksHeader<ulong>* Header;
14
15         protected UInt64LinksAvlBalancedTreeMethodsBase(LinksConstants<ulong> constants,
16             ↳ RawLink<ulong>* links, LinksHeader<ulong>* header)
17             : base(constants, (byte*)links, (byte*)header)
18         {
19             Links = links;
20             Header = header;
21         }
22
23         [MethodImpl(MethodImplOptions.AggressiveInlining)]
24         protected override ulong GetZero() => OUL;
25
26         [MethodImpl(MethodImplOptions.AggressiveInlining)]
27         protected override bool EqualToZero(ulong value) => value == OUL;
28
29         [MethodImpl(MethodImplOptions.AggressiveInlining)]
30         protected override bool AreEqual(ulong first, ulong second) => first == second;
31
32         [MethodImpl(MethodImplOptions.AggressiveInlining)]
33         protected override bool GreaterThanZero(ulong value) => value > OUL;
34
35         [MethodImpl(MethodImplOptions.AggressiveInlining)]
36         protected override bool GreaterThan(ulong first, ulong second) => first > second;
37
38         [MethodImpl(MethodImplOptions.AggressiveInlining)]
39         protected override bool GreaterOrEqualThan(ulong first, ulong second) => first >= second;
40
41         [MethodImpl(MethodImplOptions.AggressiveInlining)]
42         protected override bool GreaterOrEqualThanZero(ulong value) => true; // value >= 0 is
43             ↳ always true for ulong
44
45         [MethodImpl(MethodImplOptions.AggressiveInlining)]
46         protected override bool LessOrEqualThanZero(ulong value) => value == OUL; // value is
47             ↳ always >= 0 for ulong
48
49         [MethodImpl(MethodImplOptions.AggressiveInlining)]
50         protected override bool LessOrEqualThan(ulong first, ulong second) => first <= second;
51
52         [MethodImpl(MethodImplOptions.AggressiveInlining)]
53         protected override bool LessThanZero(ulong value) => false; // value < 0 is always false
54             ↳ for ulong
55
56         [MethodImpl(MethodImplOptions.AggressiveInlining)]
57         protected override bool LessThan(ulong first, ulong second) => first < second;
58
59         [MethodImpl(MethodImplOptions.AggressiveInlining)]
60         protected override ulong Increment(ulong value) => ++value;
61
62         [MethodImpl(MethodImplOptions.AggressiveInlining)]
63         protected override ulong Decrement(ulong value) => --value;
64
65         [MethodImpl(MethodImplOptions.AggressiveInlining)]
66         protected override ulong Add(ulong first, ulong second) => first + second;
67
68         [MethodImpl(MethodImplOptions.AggressiveInlining)]
69         protected override ulong Subtract(ulong first, ulong second) => first - second;
70
71         [MethodImpl(MethodImplOptions.AggressiveInlining)]
72         protected override bool FirstIsToTheLeftOfSecond(ulong first, ulong second)
73         {
74             ref var firstLink = ref Links[first];
75             ref var secondLink = ref Links[second];
76             return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
77                 ↳ secondLink.Source, secondLink.Target);
78         }
79     }
80 }

```

```

74 [MethodImpl(MethodImplOptions.AggressiveInlining)]
75 protected override bool FirstIsToTheRightOfSecond(ulong first, ulong second)
76 {
77     ref var firstLink = ref Links[first];
78     ref var secondLink = ref Links[second];
79     return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
    ↪ secondLink.Source, secondLink.Target);
80 }
81
82 [MethodImpl(MethodImplOptions.AggressiveInlining)]
83 protected override ulong GetSizeValue(ulong value) => unchecked((value & 4294967264UL)
    ↪ >> 5);
84
85 [MethodImpl(MethodImplOptions.AggressiveInlining)]
86 protected override void SetSizeValue(ref ulong storedValue, ulong size) => storedValue =
    ↪ unchecked(storedValue & 31UL | (size & 134217727UL) << 5);
87
88 [MethodImpl(MethodImplOptions.AggressiveInlining)]
89 protected override bool GetLeftIsChildValue(ulong value) => unchecked((value & 16UL) >>
    ↪ 4 == 1UL);
90
91 [MethodImpl(MethodImplOptions.AggressiveInlining)]
92 protected override void SetLeftIsChildValue(ref ulong storedValue, bool value) =>
    ↪ storedValue = unchecked(storedValue & 4294967279UL | (As<bool, byte>(ref value) &
    ↪ 1UL) << 4);
93
94 [MethodImpl(MethodImplOptions.AggressiveInlining)]
95 protected override bool GetRightIsChildValue(ulong value) => unchecked((value & 8UL) >>
    ↪ 3 == 1UL);
96
97 [MethodImpl(MethodImplOptions.AggressiveInlining)]
98 protected override void SetRightIsChildValue(ref ulong storedValue, bool value) =>
    ↪ storedValue = unchecked(storedValue & 4294967287UL | (As<bool, byte>(ref value) &
    ↪ 1UL) << 3);
99
100 [MethodImpl(MethodImplOptions.AggressiveInlining)]
101 protected override sbyte GetBalanceValue(ulong value) => unchecked((sbyte)(value & 7UL |
    ↪ 0xF8UL * ((value & 4UL) >> 2))); // if negative, then continue ones to the end of
    ↪ sbyte
102
103 [MethodImpl(MethodImplOptions.AggressiveInlining)]
104 protected override void SetBalanceValue(ref ulong storedValue, sbyte value) =>
    ↪ storedValue = unchecked(storedValue & 4294967288UL | (ulong)((byte)value >> 5 & 4 |
    ↪ value & 3) & 7UL);
105
106 [MethodImpl(MethodImplOptions.AggressiveInlining)]
107 protected override ref LinksHeader<ulong> GetHeaderReference() => ref *Header;
108
109 [MethodImpl(MethodImplOptions.AggressiveInlining)]
110 protected override ref RawLink<ulong> GetLinkReference(ulong link) => ref Links[link];
111 }
112 }

```

1.46 ./Platform.Data.Doublets/ResizableDirectMemory/Specific/UInt64LinksSizeBalancedTreeMethodsBase.cs

```

1 using System.Runtime.CompilerServices;
2 using Platform.Data.Doublets.ResizableDirectMemory.Generic;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.ResizableDirectMemory.Specific
7 {
8     public unsafe abstract class UInt64LinksSizeBalancedTreeMethodsBase :
    ↪ LinksSizeBalancedTreeMethodsBase<ulong>
9     {
10         protected new readonly RawLink<ulong>* Links;
11         protected new readonly LinksHeader<ulong>* Header;
12
13         protected UInt64LinksSizeBalancedTreeMethodsBase(LinksConstants<ulong> constants,
    ↪ RawLink<ulong>* links, LinksHeader<ulong>* header)
14             : base(constants, (byte*)links, (byte*)header)
15         {
16             Links = links;
17             Header = header;
18         }
19
20         [MethodImpl(MethodImplOptions.AggressiveInlining)]
21         protected override ulong GetZero() => 0UL;
22
23         [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

24     protected override bool EqualToZero(ulong value) => value == OUL;
25
26     [MethodImpl(MethodImplOptions.AggressiveInlining)]
27     protected override bool AreEqual(ulong first, ulong second) => first == second;
28
29     [MethodImpl(MethodImplOptions.AggressiveInlining)]
30     protected override bool GreaterThanZero(ulong value) => value > OUL;
31
32     [MethodImpl(MethodImplOptions.AggressiveInlining)]
33     protected override bool GreaterThan(ulong first, ulong second) => first > second;
34
35     [MethodImpl(MethodImplOptions.AggressiveInlining)]
36     protected override bool GreaterOrEqualThan(ulong first, ulong second) => first >= second;
37
38     [MethodImpl(MethodImplOptions.AggressiveInlining)]
39     protected override bool GreaterOrEqualThanZero(ulong value) => true; // value >= 0 is
    ↪ always true for ulong
40
41     [MethodImpl(MethodImplOptions.AggressiveInlining)]
42     protected override bool LessOrEqualThanZero(ulong value) => value == OUL; // value is
    ↪ always >= 0 for ulong
43
44     [MethodImpl(MethodImplOptions.AggressiveInlining)]
45     protected override bool LessOrEqualThan(ulong first, ulong second) => first <= second;
46
47     [MethodImpl(MethodImplOptions.AggressiveInlining)]
48     protected override bool LessThanZero(ulong value) => false; // value < 0 is always false
    ↪ for ulong
49
50     [MethodImpl(MethodImplOptions.AggressiveInlining)]
51     protected override bool LessThan(ulong first, ulong second) => first < second;
52
53     [MethodImpl(MethodImplOptions.AggressiveInlining)]
54     protected override ulong Increment(ulong value) => ++value;
55
56     [MethodImpl(MethodImplOptions.AggressiveInlining)]
57     protected override ulong Decrement(ulong value) => --value;
58
59     [MethodImpl(MethodImplOptions.AggressiveInlining)]
60     protected override ulong Add(ulong first, ulong second) => first + second;
61
62     [MethodImpl(MethodImplOptions.AggressiveInlining)]
63     protected override ulong Subtract(ulong first, ulong second) => first - second;
64
65     [MethodImpl(MethodImplOptions.AggressiveInlining)]
66     protected override bool FirstIsToLeftOfSecond(ulong first, ulong second)
67     {
68         ref var firstLink = ref Links[first];
69         ref var secondLink = ref Links[second];
70         return FirstIsToLeftOfSecond(firstLink.Source, firstLink.Target,
    ↪ secondLink.Source, secondLink.Target);
71     }
72
73     [MethodImpl(MethodImplOptions.AggressiveInlining)]
74     protected override bool FirstIsToTheRightOfSecond(ulong first, ulong second)
75     {
76         ref var firstLink = ref Links[first];
77         ref var secondLink = ref Links[second];
78         return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
    ↪ secondLink.Source, secondLink.Target);
79     }
80
81     [MethodImpl(MethodImplOptions.AggressiveInlining)]
82     protected override ref LinksHeader<ulong> GetHeaderReference() => ref *Header;
83
84     [MethodImpl(MethodImplOptions.AggressiveInlining)]
85     protected override ref RawLink<ulong> GetLinkReference(ulong link) => ref Links[link];
86 }
87 }

```

1.47 ./Platform.Data.Doublets/ResizableDirectMemory/Specific/UInt64LinksSourcesAvlBalancedTreeMethods.cs

```

1 using System.Runtime.CompilerServices;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Doublets.ResizableDirectMemory.Specific
6 {
7     public unsafe class UInt64LinksSourcesAvlBalancedTreeMethods :
    ↪ UInt64LinksAvlBalancedTreeMethodsBase

```

```

8 {
9     public UInt64LinksSourcesAvlBalancedTreeMethods(LinksConstants<ulong> constants,
10         ↳ RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants, links, header)
11         ↳ { }
12
13     [MethodImpl(MethodImplOptions.AggressiveInlining)]
14     protected override ref ulong GetLeftReference(ulong node) => ref
15         ↳ Links[node].LeftAsSource;
16
17     [MethodImpl(MethodImplOptions.AggressiveInlining)]
18     protected override ref ulong GetRightReference(ulong node) => ref
19         ↳ Links[node].RightAsSource;
20
21     [MethodImpl(MethodImplOptions.AggressiveInlining)]
22     protected override ulong GetLeft(ulong node) => Links[node].LeftAsSource;
23
24     [MethodImpl(MethodImplOptions.AggressiveInlining)]
25     protected override void SetLeft(ulong node, ulong left) => Links[node].LeftAsSource =
26         ↳ left;
27
28     [MethodImpl(MethodImplOptions.AggressiveInlining)]
29     protected override void SetRight(ulong node, ulong right) => Links[node].RightAsSource =
30         ↳ right;
31
32     [MethodImpl(MethodImplOptions.AggressiveInlining)]
33     protected override ulong GetSize(ulong node) => GetSizeValue(Links[node].SizeAsSource);
34
35     [MethodImpl(MethodImplOptions.AggressiveInlining)]
36     protected override void SetSize(ulong node, ulong size) => SetSizeValue(ref
37         ↳ Links[node].SizeAsSource, size);
38
39     [MethodImpl(MethodImplOptions.AggressiveInlining)]
40     protected override bool GetLeftIsChild(ulong node) =>
41         ↳ GetLeftIsChildValue(Links[node].SizeAsSource);
42
43     //[MethodImpl(MethodImplOptions.AggressiveInlining)]
44     //protected override bool GetLeftIsChild(ulong node) => IsChild(node, GetLeft(node));
45
46     [MethodImpl(MethodImplOptions.AggressiveInlining)]
47     protected override void SetLeftIsChild(ulong node, bool value) =>
48         ↳ SetLeftIsChildValue(ref Links[node].SizeAsSource, value);
49
50     [MethodImpl(MethodImplOptions.AggressiveInlining)]
51     protected override bool GetRightIsChild(ulong node) =>
52         ↳ GetRightIsChildValue(Links[node].SizeAsSource);
53
54     //[MethodImpl(MethodImplOptions.AggressiveInlining)]
55     //protected override bool GetRightIsChild(ulong node) => IsChild(node, GetRight(node));
56
57     [MethodImpl(MethodImplOptions.AggressiveInlining)]
58     protected override void SetRightIsChild(ulong node, bool value) =>
59         ↳ SetRightIsChildValue(ref Links[node].SizeAsSource, value);
60
61     [MethodImpl(MethodImplOptions.AggressiveInlining)]
62     protected override sbyte GetBalance(ulong node) =>
63         ↳ GetBalanceValue(Links[node].SizeAsSource);
64
65     [MethodImpl(MethodImplOptions.AggressiveInlining)]
66     protected override void SetBalance(ulong node, sbyte value) => SetBalanceValue(ref
67         ↳ Links[node].SizeAsSource, value);
68
69     [MethodImpl(MethodImplOptions.AggressiveInlining)]
70     protected override ulong GetTreeRoot() => Header->FirstAsSource;
71
72     [MethodImpl(MethodImplOptions.AggressiveInlining)]
73     protected override ulong GetBasePartValue(ulong link) => Links[link].Source;
74
75     [MethodImpl(MethodImplOptions.AggressiveInlining)]
76     protected override bool FirstIsToTheLeftOfSecond(ulong firstSource, ulong firstTarget,
77         ↳ ulong secondSource, ulong secondTarget)
78         ↳ => firstSource < secondSource || (firstSource == secondSource && firstTarget <
79             ↳ secondTarget);
80
81     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

70     protected override bool FirstIsToTheRightOfSecond(ulong firstSource, ulong firstTarget,
71     ↪     ulong secondSource, ulong secondTarget)
72     => firstSource > secondSource || (firstSource == secondSource && firstTarget >
73     ↪     secondTarget);
74
75     [MethodImpl(MethodImplOptions.AggressiveInlining)]
76     protected override void ClearNode(ulong node)
77     {
78         ref var link = ref Links[node];
79         link.LeftAsSource = OUL;
80         link.RightAsSource = OUL;
81         link.SizeAsSource = OUL;
82     }
83 }

```

1.48 ./Platform.Data.Doublets/ResizableDirectMemory/Specific/UInt64LinksSourcesSizeBalancedTreeMethods.cs

```

1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.ResizableDirectMemory.Specific
6  {
7      public unsafe class UInt64LinksSourcesSizeBalancedTreeMethods :
8      ↪     UInt64LinksSizeBalancedTreeMethodsBase
9      {
10         public UInt64LinksSourcesSizeBalancedTreeMethods(LinksConstants<ulong> constants,
11         ↪     RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants, links, header)
12         ↪     { }
13
14         [MethodImpl(MethodImplOptions.AggressiveInlining)]
15         protected override ref ulong GetLeftReference(ulong node) => ref
16         ↪     Links[node].LeftAsSource;
17
18         [MethodImpl(MethodImplOptions.AggressiveInlining)]
19         protected override ref ulong GetRightReference(ulong node) => ref
20         ↪     Links[node].RightAsSource;
21
22         [MethodImpl(MethodImplOptions.AggressiveInlining)]
23         protected override ulong GetLeft(ulong node) => Links[node].LeftAsSource;
24
25         [MethodImpl(MethodImplOptions.AggressiveInlining)]
26         protected override ulong GetRight(ulong node) => Links[node].RightAsSource;
27
28         [MethodImpl(MethodImplOptions.AggressiveInlining)]
29         protected override void SetLeft(ulong node, ulong left) => Links[node].LeftAsSource =
30         ↪     left;
31
32         [MethodImpl(MethodImplOptions.AggressiveInlining)]
33         protected override void SetRight(ulong node, ulong right) => Links[node].RightAsSource =
34         ↪     right;
35
36         [MethodImpl(MethodImplOptions.AggressiveInlining)]
37         protected override ulong GetSize(ulong node) => Links[node].SizeAsSource;
38
39         [MethodImpl(MethodImplOptions.AggressiveInlining)]
40         protected override void SetSize(ulong node, ulong size) => Links[node].SizeAsSource =
41         ↪     size;
42
43         [MethodImpl(MethodImplOptions.AggressiveInlining)]
44         protected override ulong GetTreeRoot() => Header->FirstAsSource;
45
46         [MethodImpl(MethodImplOptions.AggressiveInlining)]
47         protected override ulong GetBasePartValue(ulong link) => Links[link].Source;
48
49         [MethodImpl(MethodImplOptions.AggressiveInlining)]
50         protected override bool FirstIsToTheLeftOfSecond(ulong firstSource, ulong firstTarget,
51         ↪     ulong secondSource, ulong secondTarget)
52         => firstSource < secondSource || (firstSource == secondSource && firstTarget <
53         ↪     secondTarget);
54
55         [MethodImpl(MethodImplOptions.AggressiveInlining)]
56         protected override bool FirstIsToTheRightOfSecond(ulong firstSource, ulong firstTarget,
57         ↪     ulong secondSource, ulong secondTarget)
58         => firstSource > secondSource || (firstSource == secondSource && firstTarget >
59         ↪     secondTarget);
60
61         [MethodImpl(MethodImplOptions.AggressiveInlining)]
62         protected override void ClearNode(ulong node)

```

```

51     {
52         ref var link = ref Links[node];
53         link.LeftAsSource = OUL;
54         link.RightAsSource = OUL;
55         link.SizeAsSource = OUL;
56     }
57 }
58 }

```

1.49 ./Platform.Data.Doublets/ResizableDirectMemory/Specific/UInt64LinksTargetsAvlBalancedTreeMethods.cs

```

1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.ResizableDirectMemory.Specific
6  {
7      public unsafe class UInt64LinksTargetsAvlBalancedTreeMethods :
8          ↳ UInt64LinksAvlBalancedTreeMethodsBase
9      {
10
11         public UInt64LinksTargetsAvlBalancedTreeMethods(LinksConstants<ulong> constants,
12             ↳ RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants, links, header)
13             ↳ { }
14
15         [MethodImpl(MethodImplOptions.AggressiveInlining)]
16         protected override ref ulong GetLeftReference(ulong node) => ref
17             ↳ Links[node].LeftAsTarget;
18
19         [MethodImpl(MethodImplOptions.AggressiveInlining)]
20         protected override ref ulong GetRightReference(ulong node) => ref
21             ↳ Links[node].RightAsTarget;
22
23         [MethodImpl(MethodImplOptions.AggressiveInlining)]
24         protected override ulong GetLeft(ulong node) => Links[node].LeftAsTarget;
25
26         [MethodImpl(MethodImplOptions.AggressiveInlining)]
27         protected override ulong GetRight(ulong node) => Links[node].RightAsTarget;
28
29         [MethodImpl(MethodImplOptions.AggressiveInlining)]
30         protected override void SetLeft(ulong node, ulong left) => Links[node].LeftAsTarget =
31             ↳ left;
32
33         [MethodImpl(MethodImplOptions.AggressiveInlining)]
34         protected override void SetRight(ulong node, ulong right) => Links[node].RightAsTarget =
35             ↳ right;
36
37         [MethodImpl(MethodImplOptions.AggressiveInlining)]
38         protected override ulong GetSize(ulong node) => GetSizeValue(Links[node].SizeAsTarget);
39
40         [MethodImpl(MethodImplOptions.AggressiveInlining)]
41         protected override void SetSize(ulong node, ulong size) => SetSizeValue(ref
42             ↳ Links[node].SizeAsTarget, size);
43
44         [MethodImpl(MethodImplOptions.AggressiveInlining)]
45         protected override bool GetLeftIsChild(ulong node) =>
46             ↳ GetLeftIsChildValue(Links[node].SizeAsTarget);
47
48         [MethodImpl(MethodImplOptions.AggressiveInlining)]
49         protected override void SetLeftIsChild(ulong node, bool value) =>
50             ↳ SetLeftIsChildValue(ref Links[node].SizeAsTarget, value);
51
52         [MethodImpl(MethodImplOptions.AggressiveInlining)]
53         protected override bool GetRightIsChild(ulong node) =>
54             ↳ GetRightIsChildValue(Links[node].SizeAsTarget);
55
56         [MethodImpl(MethodImplOptions.AggressiveInlining)]
57         protected override void SetRightIsChild(ulong node, bool value) =>
58             ↳ SetRightIsChildValue(ref Links[node].SizeAsTarget, value);
59
60         [MethodImpl(MethodImplOptions.AggressiveInlining)]
61         protected override sbyte GetBalance(ulong node) =>
62             ↳ GetBalanceValue(Links[node].SizeAsTarget);
63
64         [MethodImpl(MethodImplOptions.AggressiveInlining)]
65         protected override void SetBalance(ulong node, sbyte value) => SetBalanceValue(ref
66             ↳ Links[node].SizeAsTarget, value);
67
68         [MethodImpl(MethodImplOptions.AggressiveInlining)]
69         protected override ulong GetTreeRoot() => Header->FirstAsTarget;
70
71     }
72 }

```

```

56 [MethodImpl(MethodImplOptions.AggressiveInlining)]
57 protected override ulong GetBasePartValue(ulong link) => Links[link].Target;
58
59 [MethodImpl(MethodImplOptions.AggressiveInlining)]
60 protected override bool FirstIsToTheLeftOfSecond(ulong firstSource, ulong firstTarget,
61     ↳ ulong secondSource, ulong secondTarget)
62     => firstTarget < secondTarget || (firstTarget == secondTarget && firstSource <
63     ↳ secondSource);
64
65 [MethodImpl(MethodImplOptions.AggressiveInlining)]
66 protected override bool FirstIsToTheRightOfSecond(ulong firstSource, ulong firstTarget,
67     ↳ ulong secondSource, ulong secondTarget)
68     => firstTarget > secondTarget || (firstTarget == secondTarget && firstSource >
69     ↳ secondSource);
70
71 [MethodImpl(MethodImplOptions.AggressiveInlining)]
72 protected override void ClearNode(ulong node)
73 {
74     ref var link = ref Links[node];
75     link.LeftAsTarget = OUL;
76     link.RightAsTarget = OUL;
77     link.SizeAsTarget = OUL;
78 }
79 }
80 }

```

1.50 ./Platform.Data.Doublets/ResizableDirectMemory/Specific/UInt64LinksTargetsSizeBalancedTreeMethods.cs

```

1 using System.Runtime.CompilerServices;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Doublets.ResizableDirectMemory.Specific
6 {
7     public unsafe class UInt64LinksTargetsSizeBalancedTreeMethods :
8     ↳ UInt64LinksSizeBalancedTreeMethodsBase
9     {
10         public UInt64LinksTargetsSizeBalancedTreeMethods(LinksConstants<ulong> constants,
11     ↳ RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants, links, header)
12     ↳ { }
13
14         [MethodImpl(MethodImplOptions.AggressiveInlining)]
15         protected override ref ulong GetLeftReference(ulong node) => ref
16     ↳ Links[node].LeftAsTarget;
17
18         [MethodImpl(MethodImplOptions.AggressiveInlining)]
19         protected override ref ulong GetRightReference(ulong node) => ref
20     ↳ Links[node].RightAsTarget;
21
22         [MethodImpl(MethodImplOptions.AggressiveInlining)]
23         protected override ulong GetLeft(ulong node) => Links[node].LeftAsTarget;
24
25         [MethodImpl(MethodImplOptions.AggressiveInlining)]
26         protected override ulong GetRight(ulong node) => Links[node].RightAsTarget;
27
28         [MethodImpl(MethodImplOptions.AggressiveInlining)]
29         protected override void SetLeft(ulong node, ulong left) => Links[node].LeftAsTarget =
30     ↳ left;
31
32         [MethodImpl(MethodImplOptions.AggressiveInlining)]
33         protected override void SetRight(ulong node, ulong right) => Links[node].RightAsTarget =
34     ↳ right;
35
36         [MethodImpl(MethodImplOptions.AggressiveInlining)]
37         protected override ulong GetSize(ulong node) => Links[node].SizeAsTarget;
38
39         [MethodImpl(MethodImplOptions.AggressiveInlining)]
40         protected override void SetSize(ulong node, ulong size) => Links[node].SizeAsTarget =
41     ↳ size;
42
43         [MethodImpl(MethodImplOptions.AggressiveInlining)]
44         protected override ulong GetTreeRoot() => Header->FirstAsTarget;
45
46         [MethodImpl(MethodImplOptions.AggressiveInlining)]
47         protected override ulong GetBasePartValue(ulong link) => Links[link].Target;
48
49         [MethodImpl(MethodImplOptions.AggressiveInlining)]
50         protected override bool FirstIsToTheLeftOfSecond(ulong firstSource, ulong firstTarget,
51     ↳ ulong secondSource, ulong secondTarget)

```



```

43     => firstTarget < secondTarget || (firstTarget == secondTarget && firstSource <
44         ↪ secondSource);
45
46     [MethodImpl(MethodImplOptions.AggressiveInlining)]
47     protected override bool FirstIsToTheRightOfSecond(ulong firstSource, ulong firstTarget,
48         ↪ ulong secondSource, ulong secondTarget)
49         => firstTarget > secondTarget || (firstTarget == secondTarget && firstSource >
50             ↪ secondSource);
51
52     [MethodImpl(MethodImplOptions.AggressiveInlining)]
53     protected override void ClearNode(ulong node)
54     {
55         ref var link = ref Links[node];
56         link.LeftAsTarget = OUL;
57         link.RightAsTarget = OUL;
58         link.SizeAsTarget = OUL;
59     }
60 }

```

1.51 ./Platform.Data.Doublets/ResizableDirectMemory/Specific/UInt64ResizableDirectMemoryLinks.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Memory;
5  using Platform.Data.Doublets.ResizableDirectMemory.Generic;
6  using Platform.Singletons;
7
8  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
9
10 namespace Platform.Data.Doublets.ResizableDirectMemory.Specific
11 {
12     public unsafe class UInt64ResizableDirectMemoryLinks : ResizableDirectMemoryLinksBase<ulong>
13     {
14         private readonly Func<ILinksTreeMethods<ulong>> _createSourceTreeMethods;
15         private readonly Func<ILinksTreeMethods<ulong>> _createTargetTreeMethods;
16         private LinksHeader<ulong>* _header;
17         private RawLink<ulong>* _links;
18
19         [MethodImpl(MethodImplOptions.AggressiveInlining)]
20         public UInt64ResizableDirectMemoryLinks(string address) : this(address,
21             ↪ DefaultLinksSizeStep) { }
22
23         /// <summary>
24         /// Создаёт экземпляр базы данных Links в файле по указанному адресу, с указанным
25         /// ↪ минимальным шагом расширения базы данных.
26         /// </summary>
27         /// <param name="address">Полный путь к файлу базы данных.</param>
28         /// <param name="memoryReservationStep">Минимальный шаг расширения базы данных в
29         /// ↪ байтах.</param>
30         [MethodImpl(MethodImplOptions.AggressiveInlining)]
31         public UInt64ResizableDirectMemoryLinks(string address, long memoryReservationStep) :
32             ↪ this(new FileMappedResizableDirectMemory(address, memoryReservationStep),
33                 ↪ memoryReservationStep) { }
34
35         [MethodImpl(MethodImplOptions.AggressiveInlining)]
36         public UInt64ResizableDirectMemoryLinks(IResizableDirectMemory memory) : this(memory,
37             ↪ DefaultLinksSizeStep) { }
38
39         [MethodImpl(MethodImplOptions.AggressiveInlining)]
40         public UInt64ResizableDirectMemoryLinks(IResizableDirectMemory memory, long
41             ↪ memoryReservationStep) : this(memory, memoryReservationStep,
42             ↪ Default<LinksConstants<ulong>>.Instance, true) { }
43
44         [MethodImpl(MethodImplOptions.AggressiveInlining)]
45         public UInt64ResizableDirectMemoryLinks(IResizableDirectMemory memory, long
46             ↪ memoryReservationStep, LinksConstants<ulong> constants, bool useAvlBasedIndex) :
47             ↪ base(memory, memoryReservationStep, constants)
48         {
49             if (useAvlBasedIndex)
50             {
51                 _createSourceTreeMethods = () => new
52                     ↪ UInt64LinksSourcesAvlBalancedTreeMethods(Constants, _links, _header);
53                 _createTargetTreeMethods = () => new
54                     ↪ UInt64LinksTargetsAvlBalancedTreeMethods(Constants, _links, _header);
55             }
56             else
57             {
58

```

```

46         _createSourceTreeMethods = () => new
47         ↪ UInt64LinksSourcesSizeBalancedTreeMethods(Constants, _links, _header);
48     _createTargetTreeMethods = () => new
49     ↪ UInt64LinksTargetsSizeBalancedTreeMethods(Constants, _links, _header);
50 }
51 Init(memory, memoryReservationStep);
52 }
53 [MethodImpl(MethodImplOptions.AggressiveInlining)]
54 protected override void SetPointers(IResizableDirectMemory memory)
55 {
56     _header = (LinksHeader<ulong>*)memory.Pointer;
57     _links = (RawLink<ulong>*)memory.Pointer;
58     SourcesTreeMethods = _createSourceTreeMethods();
59     TargetsTreeMethods = _createTargetTreeMethods();
60     UnusedLinksListMethods = new UInt64UnusedLinksListMethods(_links, _header);
61 }
62 [MethodImpl(MethodImplOptions.AggressiveInlining)]
63 protected override void ResetPointers()
64 {
65     base.ResetPointers();
66     _links = null;
67     _header = null;
68 }
69 [MethodImpl(MethodImplOptions.AggressiveInlining)]
70 protected override ref LinksHeader<ulong> GetHeaderReference() => ref *_header;
71 [MethodImpl(MethodImplOptions.AggressiveInlining)]
72 protected override ref RawLink<ulong> GetLinkReference(ulong linkIndex) => ref
73 ↪ _links[linkIndex];
74 [MethodImpl(MethodImplOptions.AggressiveInlining)]
75 protected override bool AreEqual(ulong first, ulong second) => first == second;
76 [MethodImpl(MethodImplOptions.AggressiveInlining)]
77 protected override bool LessThan(ulong first, ulong second) => first < second;
78 [MethodImpl(MethodImplOptions.AggressiveInlining)]
79 protected override bool LessOrEqualThan(ulong first, ulong second) => first <= second;
80 [MethodImpl(MethodImplOptions.AggressiveInlining)]
81 protected override bool GreaterThan(ulong first, ulong second) => first > second;
82 [MethodImpl(MethodImplOptions.AggressiveInlining)]
83 protected override bool GreaterOrEqualThan(ulong first, ulong second) => first >= second;
84 [MethodImpl(MethodImplOptions.AggressiveInlining)]
85 protected override bool GetZero() => 0UL;
86 [MethodImpl(MethodImplOptions.AggressiveInlining)]
87 protected override bool GetOne() => 1UL;
88 [MethodImpl(MethodImplOptions.AggressiveInlining)]
89 protected override long ConvertToInt64(ulong value) => (long)value;
90 [MethodImpl(MethodImplOptions.AggressiveInlining)]
91 protected override ulong ConvertToAddress(long value) => (ulong)value;
92 [MethodImpl(MethodImplOptions.AggressiveInlining)]
93 protected override ulong Add(ulong first, ulong second) => first + second;
94 [MethodImpl(MethodImplOptions.AggressiveInlining)]
95 protected override ulong Subtract(ulong first, ulong second) => first - second;
96 [MethodImpl(MethodImplOptions.AggressiveInlining)]
97 protected override ulong Increment(ulong link) => ++link;
98 [MethodImpl(MethodImplOptions.AggressiveInlining)]
99 protected override ulong Decrement(ulong link) => --link;
100 }
101 }
102 }
103 }
104 }
105 }
106 }
107 }
108 }
109 }
110 }
111 }
112 }
113 }
114 }
115 }

```

1.52 ./Platform.Data.Doublets/ResizableDirectMemory/Specific/UInt64UnusedLinksListMethods.cs

```

1 using System.Runtime.CompilerServices;
2 using Platform.Data.Doublets.ResizableDirectMemory.Generic;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

```

```

5
6 namespace Platform.Data.Doublets.ResizableDirectMemory.Specific
7 {
8     public unsafe class UInt64UnusedLinksListMethods : UnusedLinksListMethods<ulong>
9     {
10         private readonly RawLink<ulong>* _links;
11         private readonly LinksHeader<ulong>* _header;
12
13         [MethodImpl(MethodImplOptions.AggressiveInlining)]
14         public UInt64UnusedLinksListMethods(RawLink<ulong>* links, LinksHeader<ulong>* header)
15             : base((byte*)links, (byte*)header)
16         {
17             _links = links;
18             _header = header;
19         }
20
21         [MethodImpl(MethodImplOptions.AggressiveInlining)]
22         protected override ref RawLink<ulong> GetLinkReference(ulong link) => ref _links[link];
23
24         [MethodImpl(MethodImplOptions.AggressiveInlining)]
25         protected override ref LinksHeader<ulong> GetHeaderReference() => ref *_header;
26     }
27 }

```

1.53 ./Platform.Data.Doublets/Sequences/Converters/BalancedVariantConverter.cs

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Sequences.Converters
7 {
8     public class BalancedVariantConverter<TLink> : LinksListToSequenceConverterBase<TLink>
9     {
10         [MethodImpl(MethodImplOptions.AggressiveInlining)]
11         public BalancedVariantConverter(ILinks<TLink> links) : base(links) { }
12
13         [MethodImpl(MethodImplOptions.AggressiveInlining)]
14         public override TLink Convert(ICollection<TLink> sequence)
15         {
16             var length = sequence.Count;
17             if (length < 1)
18             {
19                 return default;
20             }
21             if (length == 1)
22             {
23                 return sequence[0];
24             }
25             // Make copy of next layer
26             if (length > 2)
27             {
28                 // TODO: Try to use stackalloc (which at the moment is not working with
29                 // ↪ generics) but will be possible with Sigil
30                 var halvedSequence = new TLink[(length / 2) + (length % 2)];
31                 HalveSequence(halvedSequence, sequence, length);
32                 sequence = halvedSequence;
33                 length = halvedSequence.Length;
34             }
35             // Keep creating layer after layer
36             while (length > 2)
37             {
38                 HalveSequence(sequence, sequence, length);
39                 length = (length / 2) + (length % 2);
40             }
41             return Links.GetOrCreate(sequence[0], sequence[1]);
42         }
43
44         [MethodImpl(MethodImplOptions.AggressiveInlining)]
45         private void HalveSequence(ICollection<TLink> destination, ICollection<TLink> source, int length)
46         {
47             var loopedLength = length - (length % 2);
48             for (var i = 0; i < loopedLength; i += 2)
49             {
50                 destination[i / 2] = Links.GetOrCreate(source[i], source[i + 1]);
51             }
52             if (length > loopedLength)
53             {
54                 destination[length / 2] = source[length - 1];
55             }
56         }
57     }
58 }

```

```

54     }
55 }
56 }
57 }

```

1.54 ./Platform.Data.Doublets/Sequences/Converters/CompressingConverter.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Collections;
5  using Platform.Converters;
6  using Platform.Singletons;
7  using Platform.Numbers;
8  using Platform.Data.Doublets.Sequences.Frequencies.Cache;
9
10 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
11
12 namespace Platform.Data.Doublets.Sequences.Converters
13 {
14     /// <remarks>
15     /// TODO: Возможно будет лучше если алгоритм будет выполняться полностью изолированно от
16     ///     Links на этапе сжатия.
17     ///     А именно будет создаваться временный список пар необходимых для выполнения сжатия, в
18     ///     таком случае тип значения элемента массива может быть любым, как char так и ulong.
19     ///     Как только список/словарь пар был выявлен можно разом выполнить создание всех этих
20     ///     пар, а так же разом выполнить замену.
21     /// </remarks>
22     public class CompressingConverter<TLink> : LinksListToSequenceConverterBase<TLink>
23     {
24         private static readonly LinksConstants<TLink> _constants =
25             ↪ Default<LinksConstants<TLink>>.Instance;
26         private static readonly EqualityComparer<TLink> _equalityComparer =
27             ↪ EqualityComparer<TLink>.Default;
28         private static readonly Comparer<TLink> _comparer = Comparer<TLink>.Default;
29
30         private static readonly TLink _zero = default;
31         private static readonly TLink _one = Arithmetic.Increment(_zero);
32
33         private readonly IConverter<IList<TLink>, TLink> _baseConverter;
34         private readonly LinkFrequenciesCache<TLink> _doubletFrequenciesCache;
35         private readonly TLink _minFrequencyToCompress;
36         private readonly bool _doInitialFrequenciesIncrement;
37         private Doublet<TLink> _maxDoublet;
38         private LinkFrequency<TLink> _maxDoubletData;
39
40         private struct HalfDoublet
41         {
42             public TLink Element;
43             public LinkFrequency<TLink> DoubletData;
44
45             [MethodImpl(MethodImplOptions.AggressiveInlining)]
46             public HalfDoublet(TLink element, LinkFrequency<TLink> doubletData)
47             {
48                 Element = element;
49                 DoubletData = doubletData;
50             }
51
52             public override string ToString() => $"{Element}: ({DoubletData})";
53         }
54
55         [MethodImpl(MethodImplOptions.AggressiveInlining)]
56         public CompressingConverter(ILinks<TLink> links, IConverter<IList<TLink>, TLink>
57             ↪ baseConverter, LinkFrequenciesCache<TLink> doubletFrequenciesCache)
58             : this(links, baseConverter, doubletFrequenciesCache, _one, true)
59         {
60         }
61
62         [MethodImpl(MethodImplOptions.AggressiveInlining)]
63         public CompressingConverter(ILinks<TLink> links, IConverter<IList<TLink>, TLink>
64             ↪ baseConverter, LinkFrequenciesCache<TLink> doubletFrequenciesCache, bool
65             ↪ doInitialFrequenciesIncrement)
66             : this(links, baseConverter, doubletFrequenciesCache, _one,
67                 ↪ doInitialFrequenciesIncrement)
68         {
69         }
70
71         [MethodImpl(MethodImplOptions.AggressiveInlining)]
72         public CompressingConverter(ILinks<TLink> links, IConverter<IList<TLink>, TLink>
73             ↪ baseConverter, LinkFrequenciesCache<TLink> doubletFrequenciesCache, TLink
74             ↪ minFrequencyToCompress, bool doInitialFrequenciesIncrement)

```

```

64         : base(links)
65     {
66         _baseConverter = baseConverter;
67         _doubletFrequenciesCache = doubletFrequenciesCache;
68         if (_comparer.Compare(minFrequencyToCompress, _one) < 0)
69         {
70             minFrequencyToCompress = _one;
71         }
72         _minFrequencyToCompress = minFrequencyToCompress;
73         _doInitialFrequenciesIncrement = doInitialFrequenciesIncrement;
74         ResetMaxDoublet();
75     }
76
77     [MethodImpl(MethodImplOptions.AggressiveInlining)]
78     public override TLink Convert(ICollection<TLink> source) =>
79     {
80         // <remarks>
81         // Original algorithm idea: https://en.wikipedia.org/wiki/Byte\_pair\_encoding .
82         // Faster version (doublets' frequencies dictionary is not recreated).
83         // </remarks>
84         [MethodImpl(MethodImplOptions.AggressiveInlining)]
85         private ICollection<TLink> Compress(ICollection<TLink> sequence)
86         {
87             if (sequence.IsNullOrEmpty())
88             {
89                 return null;
90             }
91             if (sequence.Count == 1)
92             {
93                 return sequence;
94             }
95             if (sequence.Count == 2)
96             {
97                 return new[] { Links.GetOrCreate(sequence[0], sequence[1]) };
98             }
99             // TODO: arraypool with min size (to improve cache locality) or stackalloc with Sigil
100             var copy = new HalfDoublet[sequence.Count];
101             Doublet<TLink> doublet = default;
102             for (var i = 1; i < sequence.Count; i++)
103             {
104                 doublet.Source = sequence[i - 1];
105                 doublet.Target = sequence[i];
106                 LinkFrequency<TLink> data;
107                 if (_doInitialFrequenciesIncrement)
108                 {
109                     data = _doubletFrequenciesCache.IncrementFrequency(ref doublet);
110                 }
111                 else
112                 {
113                     data = _doubletFrequenciesCache.GetFrequency(ref doublet);
114                     if (data == null)
115                     {
116                         throw new NotSupportedException("If you ask not to increment
117                             ↪ frequencies, it is expected that all frequencies for the sequence
118                             ↪ are prepared.");
119                     }
120                 }
121                 copy[i - 1].Element = sequence[i - 1];
122                 copy[i - 1].DoubletData = data;
123                 UpdateMaxDoublet(ref doublet, data);
124             }
125             copy[sequence.Count - 1].Element = sequence[sequence.Count - 1];
126             copy[sequence.Count - 1].DoubletData = new LinkFrequency<TLink>();
127             if (_comparer.Compare(_maxDoubletData.Frequency, default) > 0)
128             {
129                 var newLength = ReplaceDoublets(copy);
130                 sequence = new TLink[newLength];
131                 for (int i = 0; i < newLength; i++)
132                 {
133                     sequence[i] = copy[i].Element;
134                 }
135             }
136             return sequence;
137         }
138     }
139
140     // <remarks>
141     // Original algorithm idea: https://en.wikipedia.org/wiki/Byte\_pair\_encoding
142     // </remarks>

```

```

140 [MethodImpl(MethodImplOptions.AggressiveInlining)]
141 private int ReplaceDoublets(HalfDoublet[] copy)
142 {
143     var oldLength = copy.Length;
144     var newLength = copy.Length;
145     while (_comparer.Compare(_maxDoubletData.Frequency, default) > 0)
146     {
147         var maxDoubletSource = _maxDoublet.Source;
148         var maxDoubletTarget = _maxDoublet.Target;
149         if (_equalityComparer.Equals(_maxDoubletData.Link, _constants.Null))
150         {
151             _maxDoubletData.Link = Links.GetOrCreate(maxDoubletSource, maxDoubletTarget);
152         }
153         var maxDoubletReplacementLink = _maxDoubletData.Link;
154         oldLength--;
155         var oldLengthMinusTwo = oldLength - 1;
156         // Substitute all usages
157         int w = 0, r = 0; // (r == read, w == write)
158         for (; r < oldLength; r++)
159         {
160             if (_equalityComparer.Equals(copy[r].Element, maxDoubletSource) &&
161                 ↪ _equalityComparer.Equals(copy[r + 1].Element, maxDoubletTarget))
162             {
163                 if (r > 0)
164                 {
165                     var previous = copy[w - 1].Element;
166                     copy[w - 1].DoubletData.DecrementFrequency();
167                     copy[w - 1].DoubletData =
168                         ↪ _doubletFrequenciesCache.IncrementFrequency(previous,
169                         ↪ maxDoubletReplacementLink);
170                 }
171                 if (r < oldLengthMinusTwo)
172                 {
173                     var next = copy[r + 2].Element;
174                     copy[r + 1].DoubletData.DecrementFrequency();
175                     copy[w].DoubletData = _doubletFrequenciesCache.IncrementFrequency(maxDoubletReplacementLink,
176                     ↪ next);
177                 }
178                 copy[w++].Element = maxDoubletReplacementLink;
179                 r++;
180                 newLength--;
181             }
182             else
183             {
184                 copy[w++] = copy[r];
185             }
186         }
187         if (w < newLength)
188         {
189             copy[w] = copy[r];
190         }
191         oldLength = newLength;
192         ResetMaxDoublet();
193         UpdateMaxDoublet(copy, newLength);
194     }
195     return newLength;
196 }
197
198 [MethodImpl(MethodImplOptions.AggressiveInlining)]
199 private void ResetMaxDoublet()
200 {
201     _maxDoublet = new Doublet<TLink>();
202     _maxDoubletData = new LinkFrequency<TLink>();
203 }
204
205 [MethodImpl(MethodImplOptions.AggressiveInlining)]
206 private void UpdateMaxDoublet(HalfDoublet[] copy, int length)
207 {
208     Doublet<TLink> doublet = default;
209     for (var i = 1; i < length; i++)
210     {
211         doublet.Source = copy[i - 1].Element;
212         doublet.Target = copy[i].Element;
213         UpdateMaxDoublet(ref doublet, copy[i - 1].DoubletData);
214     }
215 }
216
217 [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

214 private void UpdateMaxDoublet(ref Doublet<TLink> doublet, LinkFrequency<TLink> data)
215 {
216     var frequency = data.Frequency;
217     var maxFrequency = _maxDoubletData.Frequency;
218     //if (frequency > _minFrequencyToCompress && (maxFrequency < frequency ||
    ↪ (maxFrequency == frequency && doublet.Source + doublet.Target < /* gives better
    ↪ compression string data (and gives collisions quickly) */ _maxDoublet.Source +
    ↪ _maxDoublet.Target)))
219 if (_comparer.Compare(frequency, _minFrequencyToCompress) > 0 &&
220     (_comparer.Compare(maxFrequency, frequency) < 0 ||
    ↪ (_equalityComparer.Equals(maxFrequency, frequency) &&
    ↪ _comparer.Compare(Arithmetic.Add(doublet.Source, doublet.Target),
    ↪ Arithmetic.Add(_maxDoublet.Source, _maxDoublet.Target)) > 0))) /* gives
    ↪ better stability and better compression on sequent data and even on random
    ↪ numbers data (but gives collisions anyway) */
221 {
222     _maxDoublet = doublet;
223     _maxDoubletData = data;
224 }
225 }
226 }
227 }

```

1.55 ./Platform.Data.Doublets/Sequences/Converters/LinksListToSequenceConverterBase.cs

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3 using Platform.Converters;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data.Doublets.Sequences.Converters
8 {
9     public abstract class LinksListToSequenceConverterBase<TLink> : IConverter<IList<TLink>,
    ↪ TLink>
10     {
11         protected readonly ILinks<TLink> Links;
12
13         [MethodImpl(MethodImplOptions.AggressiveInlining)]
14         protected LinksListToSequenceConverterBase(ILinks<TLink> links) => Links = links;
15
16         [MethodImpl(MethodImplOptions.AggressiveInlining)]
17         public abstract TLink Convert(IList<TLink> source);
18     }
19 }

```

1.56 ./Platform.Data.Doublets/Sequences/Converters/OptimalVariantConverter.cs

```

1 using System.Collections.Generic;
2 using System.Linq;
3 using System.Runtime.CompilerServices;
4 using Platform.Converters;
5
6 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8 namespace Platform.Data.Doublets.Sequences.Converters
9 {
10     public class OptimalVariantConverter<TLink> : LinksListToSequenceConverterBase<TLink>
11     {
12         private static readonly EqualityComparer<TLink> _equalityComparer =
    ↪ EqualityComparer<TLink>.Default;
13         private static readonly Comparer<TLink> _comparer = Comparer<TLink>.Default;
14
15         private readonly IConverter<IList<TLink>> _sequenceToItsLocalElementLevelsConverter;
16
17         [MethodImpl(MethodImplOptions.AggressiveInlining)]
18         public OptimalVariantConverter(ILinks<TLink> links, IConverter<IList<TLink>>
    ↪ sequenceToItsLocalElementLevelsConverter) : base(links)
19         => _sequenceToItsLocalElementLevelsConverter =
    ↪ sequenceToItsLocalElementLevelsConverter;
20
21         [MethodImpl(MethodImplOptions.AggressiveInlining)]
22         public override TLink Convert(IList<TLink> sequence)
23         {
24             var length = sequence.Count;
25             if (length == 1)
26             {
27                 return sequence[0];
28             }
29             var links = Links;
30             if (length == 2)

```

```

31     {
32         return links.GetOrCreate(sequence[0], sequence[1]);
33     }
34     sequence = sequence.ToArray();
35     var levels = _sequenceToItsLocalElementLevelsConverter.Convert(sequence);
36     while (length > 2)
37     {
38         var levelRepeat = 1;
39         var currentLevel = levels[0];
40         var previousLevel = levels[0];
41         var skipOnce = false;
42         var w = 0;
43         for (var i = 1; i < length; i++)
44         {
45             if (_equalityComparer.Equals(currentLevel, levels[i]))
46             {
47                 levelRepeat++;
48                 skipOnce = false;
49                 if (levelRepeat == 2)
50                 {
51                     sequence[w] = links.GetOrCreate(sequence[i - 1], sequence[i]);
52                     var newLevel = i >= length - 1 ?
53                         GetPreviousLowerThanCurrentOrCurrent(previousLevel,
54                             ↪ currentLevel) :
55                         GetNextLowerThanCurrentOrCurrent(currentLevel, levels[i + 1]) :
56                         GetGreatestNeighbourLowerThanCurrentOrCurrent(previousLevel,
57                             ↪ currentLevel, levels[i + 1]);
58                     levels[w] = newLevel;
59                     previousLevel = currentLevel;
60                     w++;
61                     levelRepeat = 0;
62                     skipOnce = true;
63                 }
64                 else if (i == length - 1)
65                 {
66                     sequence[w] = sequence[i];
67                     levels[w] = levels[i];
68                     w++;
69                 }
70             }
71             else
72             {
73                 currentLevel = levels[i];
74                 levelRepeat = 1;
75                 if (skipOnce)
76                 {
77                     skipOnce = false;
78                 }
79                 else
80                 {
81                     sequence[w] = sequence[i - 1];
82                     levels[w] = levels[i - 1];
83                     previousLevel = levels[w];
84                     w++;
85                 }
86                 if (i == length - 1)
87                 {
88                     sequence[w] = sequence[i];
89                     levels[w] = levels[i];
90                     w++;
91                 }
92             }
93             length = w;
94         }
95         return links.GetOrCreate(sequence[0], sequence[1]);
96     }
97
98     [MethodImpl(MethodImplOptions.AggressiveInlining)]
99     private static TLink GetGreatestNeighbourLowerThanCurrentOrCurrent(TLink previous, TLink
100     ↪ current, TLink next)
101     {
102         return _comparer.Compare(previous, next) > 0
103             ? _comparer.Compare(previous, current) < 0 ? previous : current
104             : _comparer.Compare(next, current) < 0 ? next : current;
105     }
106
107     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```



```

107     private static TLink GetNextLowerThanCurrentOrCurrent(TLink current, TLink next) =>
108         ↪ _comparer.Compare(next, current) < 0 ? next : current;
109
110     [MethodImpl(MethodImplOptions.AggressiveInlining)]
111     private static TLink GetPreviousLowerThanCurrentOrCurrent(TLink previous, TLink current)
112         ↪ => _comparer.Compare(previous, current) < 0 ? previous : current;
113 }
114 }

```

1.57 ./Platform.Data.Doublets/Sequences/Converters/SequenceToItsLocalElementLevelsConverter.cs

```

1  using System.Collections.Generic;
2  using System.Runtime.CompilerServices;
3  using Platform.Converters;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.Sequences.Converters
8  {
9      public class SequenceToItsLocalElementLevelsConverter<TLink> : LinksOperatorBase<TLink>,
10         ↪ IConverter<IList<TLink>>
11     {
12         private static readonly Comparer<TLink> _comparer = Comparer<TLink>.Default;
13
14         private readonly IConverter<Doublet<TLink>, TLink> _linkToItsFrequencyToNumberConveter;
15
16         [MethodImpl(MethodImplOptions.AggressiveInlining)]
17         public SequenceToItsLocalElementLevelsConverter(ILinks<TLink> links,
18             ↪ IConverter<Doublet<TLink>, TLink> linkToItsFrequencyToNumberConveter) : base(links)
19             ↪ => _linkToItsFrequencyToNumberConveter = linkToItsFrequencyToNumberConveter;
20
21         [MethodImpl(MethodImplOptions.AggressiveInlining)]
22         public IList<TLink> Convert(IList<TLink> sequence)
23         {
24             var levels = new TLink[sequence.Count];
25             levels[0] = GetFrequencyNumber(sequence[0], sequence[1]);
26             for (var i = 1; i < sequence.Count - 1; i++)
27             {
28                 var previous = GetFrequencyNumber(sequence[i - 1], sequence[i]);
29                 var next = GetFrequencyNumber(sequence[i], sequence[i + 1]);
30                 levels[i] = _comparer.Compare(previous, next) > 0 ? previous : next;
31             }
32             levels[levels.Length - 1] = GetFrequencyNumber(sequence[sequence.Count - 2],
33                 ↪ sequence[sequence.Count - 1]);
34             return levels;
35         }
36
37         [MethodImpl(MethodImplOptions.AggressiveInlining)]
38         public TLink GetFrequencyNumber(TLink source, TLink target) =>
39             ↪ _linkToItsFrequencyToNumberConveter.Convert(new Doublet<TLink>(source, target));
40     }
41 }

```

1.58 ./Platform.Data.Doublets/Sequences/CriterionMatchers/DefaultSequenceElementCriterionMatcher.cs

```

1  using System.Runtime.CompilerServices;
2  using Platform.Interfaces;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Sequences.CriterionMatchers
7  {
8      public class DefaultSequenceElementCriterionMatcher<TLink> : LinksOperatorBase<TLink>,
9         ↪ ICriterionMatcher<TLink>
10     {
11         [MethodImpl(MethodImplOptions.AggressiveInlining)]
12         public DefaultSequenceElementCriterionMatcher(ILinks<TLink> links) : base(links) {}
13
14         [MethodImpl(MethodImplOptions.AggressiveInlining)]
15         public bool IsMatched(TLink argument) => Links.IsPartialPoint(argument);
16     }
17 }

```

1.59 ./Platform.Data.Doublets/Sequences/CriterionMatchers/MarkedSequenceCriterionMatcher.cs

```

1  using System.Collections.Generic;
2  using System.Runtime.CompilerServices;
3  using Platform.Interfaces;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.Sequences.CriterionMatchers

```

```

8 {
9     public class MarkedSequenceCriterionMatcher<TLink> : ICriterionMatcher<TLink>
10    {
11        private static readonly EqualityComparer<TLink> _equalityComparer =
12            ↳ EqualityComparer<TLink>.Default;
13
14        private readonly ILinks<TLink> _links;
15        private readonly TLink _sequenceMarkerLink;
16
17        [MethodImpl(MethodImplOptions.AggressiveInlining)]
18        public MarkedSequenceCriterionMatcher(ILinks<TLink> links, TLink sequenceMarkerLink)
19        {
20            _links = links;
21            _sequenceMarkerLink = sequenceMarkerLink;
22        }
23
24        [MethodImpl(MethodImplOptions.AggressiveInlining)]
25        public bool IsMatched(TLink sequenceCandidate)
26            => _equalityComparer.Equals(_links.GetSource(sequenceCandidate), _sequenceMarkerLink)
27            || !_equalityComparer.Equals(_links.SearchOrDefault(_sequenceMarkerLink,
28                ↳ sequenceCandidate), _links.Constants.Null);
29    }
30 }

```

1.60 ./Platform.Data.Doublets/Sequences/DefaultSequenceAppender.cs

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3 using Platform.Collections.Stacks;
4 using Platform.Data.Doublets.Sequences.HeightProviders;
5 using Platform.Data.Sequences;
6
7 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9 namespace Platform.Data.Doublets.Sequences
10 {
11     public class DefaultSequenceAppender<TLink> : LinksOperatorBase<TLink>,
12         ↳ ISequenceAppender<TLink>
13     {
14         private static readonly EqualityComparer<TLink> _equalityComparer =
15             ↳ EqualityComparer<TLink>.Default;
16
17         private readonly IStack<TLink> _stack;
18         private readonly ISequenceHeightProvider<TLink> _heightProvider;
19
20         [MethodImpl(MethodImplOptions.AggressiveInlining)]
21         public DefaultSequenceAppender(ILinks<TLink> links, IStack<TLink> stack,
22             ↳ ISequenceHeightProvider<TLink> heightProvider)
23             : base(links)
24         {
25             _stack = stack;
26             _heightProvider = heightProvider;
27         }
28
29         [MethodImpl(MethodImplOptions.AggressiveInlining)]
30         public TLink Append(TLink sequence, TLink appendant)
31         {
32             var cursor = sequence;
33             while (!_equalityComparer.Equals(_heightProvider.Get(cursor), default))
34             {
35                 var source = Links.GetSource(cursor);
36                 var target = Links.GetTarget(cursor);
37                 if (_equalityComparer.Equals(_heightProvider.Get(source),
38                     ↳ _heightProvider.Get(target)))
39                 {
40                     break;
41                 }
42                 else
43                 {
44                     _stack.Push(source);
45                     cursor = target;
46                 }
47             }
48             var left = cursor;
49             var right = appendant;
50             while (!_equalityComparer.Equals(cursor = _stack.Pop(), Links.Constants.Null))
51             {
52                 right = Links.GetOrCreate(left, right);
53                 left = cursor;
54             }
55             return Links.GetOrCreate(left, right);
56         }
57     }
58 }

```

```

52     }
53 }
54 }

```

1.61 ./Platform.Data.Doublets/Sequences/DuplicateSegmentsCounter.cs

```

1  using System.Collections.Generic;
2  using System.Linq;
3  using System.Runtime.CompilerServices;
4  using Platform.Interfaces;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Sequences
9  {
10     public class DuplicateSegmentsCounter<TLink> : ICounter<int>
11     {
12         private readonly IProvider<IList<KeyValuePair<IList<TLink>, IList<TLink>>>>
13             ↪ _duplicateFragmentsProvider;
14
15         [MethodImpl(MethodImplOptions.AggressiveInlining)]
16         public DuplicateSegmentsCounter(IProvider<IList<KeyValuePair<IList<TLink>,
17             ↪ IList<TLink>>>> duplicateFragmentsProvider) => _duplicateFragmentsProvider =
18             ↪ duplicateFragmentsProvider;
19
20         [MethodImpl(MethodImplOptions.AggressiveInlining)]
21         public int Count() => _duplicateFragmentsProvider.Get().Sum(x => x.Value.Count);
22     }
23 }

```

1.62 ./Platform.Data.Doublets/Sequences/DuplicateSegmentsProvider.cs

```

1  using System;
2  using System.Linq;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5  using Platform.Interfaces;
6  using Platform.Collections;
7  using Platform.Collections.Lists;
8  using Platform.Collections.Segments;
9  using Platform.Collections.Segments.Walkers;
10 using Platform.Singletons;
11 using Platform.Converters;
12 using Platform.Data.Doublets.Unicode;
13
14 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
15
16 namespace Platform.Data.Doublets.Sequences
17 {
18     public class DuplicateSegmentsProvider<TLink> :
19         ↪ DictionaryBasedDuplicateSegmentsWalkerBase<TLink>,
20         ↪ IProvider<IList<KeyValuePair<IList<TLink>, IList<TLink>>>>
21     {
22         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
23             ↪ UncheckedConverter<TLink, long>.Default;
24         private static readonly UncheckedConverter<TLink, ulong> _addressToUInt64Converter =
25             ↪ UncheckedConverter<TLink, ulong>.Default;
26         private static readonly UncheckedConverter<ulong, TLink> _uInt64ToAddressConverter =
27             ↪ UncheckedConverter<ulong, TLink>.Default;
28
29         private readonly ILinks<TLink> _links;
30         private readonly ILinks<TLink> _sequences;
31         private HashSet<KeyValuePair<IList<TLink>, IList<TLink>>> _groups;
32         private BitString _visited;
33
34         private class ItemEquilityComparer : IEqualityComparer<KeyValuePair<IList<TLink>,
35             ↪ IList<TLink>>>
36         {
37             private readonly IListEqualityComparer<TLink> _listComparer;
38
39             public ItemEquilityComparer() => _listComparer =
40                 ↪ Default<IListEqualityComparer<TLink>>.Instance;
41
42             [MethodImpl(MethodImplOptions.AggressiveInlining)]
43             public bool Equals(KeyValuePair<IList<TLink>, IList<TLink>> left,
44                 ↪ KeyValuePair<IList<TLink>, IList<TLink>> right) =>
45                 ↪ _listComparer.Equals(left.Key, right.Key) && _listComparer.Equals(left.Value,
46                 ↪ right.Value);
47
48             [MethodImpl(MethodImplOptions.AggressiveInlining)]
49             public int GetHashCode(KeyValuePair<IList<TLink>, IList<TLink>> pair) =>
50                 ↪ (_listComparer.GetHashCode(pair.Key),
51                 ↪ _listComparer.GetHashCode(pair.Value)).GetHashCode();
52         }
53     }
54 }

```

```

40     }
41
42     private class ItemComparer : IComparer<KeyValuePair<IList<TLink>, IList<TLink>>>
43     {
44         private readonly IListComparer<TLink> _listComparer;
45
46         [MethodImpl(MethodImplOptions.AggressiveInlining)]
47         public ItemComparer() => _listComparer = Default<IListComparer<TLink>>.Instance;
48
49         [MethodImpl(MethodImplOptions.AggressiveInlining)]
50         public int Compare(KeyValuePair<IList<TLink>, IList<TLink>> left,
51             ↪ KeyValuePair<IList<TLink>, IList<TLink>> right)
52         {
53             var intermediateResult = _listComparer.Compare(left.Key, right.Key);
54             if (intermediateResult == 0)
55             {
56                 intermediateResult = _listComparer.Compare(left.Value, right.Value);
57             }
58             return intermediateResult;
59         }
60     }
61
62     [MethodImpl(MethodImplOptions.AggressiveInlining)]
63     public DuplicateSegmentsProvider(ILinks<TLink> links, ILinks<TLink> sequences)
64         : base(minimumStringSegmentLength: 2)
65     {
66         _links = links;
67         _sequences = sequences;
68     }
69
70     [MethodImpl(MethodImplOptions.AggressiveInlining)]
71     public IList<KeyValuePair<IList<TLink>, IList<TLink>>> Get()
72     {
73         _groups = new HashSet<KeyValuePair<IList<TLink>,
74             ↪ IList<TLink>>>(Default<ItemEqualityComparer>.Instance);
75         var count = _links.Count();
76         _visited = new BitString(_addressToInt64Converter.Convert(count) + 1L);
77         _links.Each(link =>
78         {
79             var linkIndex = _links.GetIndex(link);
80             var linkBitIndex = _addressToInt64Converter.Convert(linkIndex);
81             if (!_visited.Get(linkBitIndex))
82             {
83                 var sequenceElements = new List<TLink>();
84                 var filler = new ListFiller<TLink, TLink>(sequenceElements,
85                     ↪ _sequences.Constants.Break);
86                 _sequences.Each(filler.AddSkipFirstAndReturnConstant, new
87                     ↪ LinkAddress<TLink>(linkIndex));
88                 if (sequenceElements.Count > 2)
89                 {
90                     WalkAll(sequenceElements);
91                 }
92             }
93             return _links.Constants.Continue;
94         });
95         var resultList = _groups.ToList();
96         var comparer = Default<ItemComparer>.Instance;
97         resultList.Sort(comparer);
98
99         #if DEBUG
100         foreach (var item in resultList)
101         {
102             PrintDuplicates(item);
103         }
104         #endif
105         return resultList;
106     }
107
108     [MethodImpl(MethodImplOptions.AggressiveInlining)]
109     protected override Segment<TLink> CreateSegment(IList<TLink> elements, int offset, int
110         ↪ length) => new Segment<TLink>(elements, offset, length);
111
112     [MethodImpl(MethodImplOptions.AggressiveInlining)]
113     protected override void OnDuplicateFound(Segment<TLink> segment)
114     {
115         var duplicates = CollectDuplicatesForSegment(segment);
116         if (duplicates.Count > 1)
117         {
118             _groups.Add(new KeyValuePair<IList<TLink>, IList<TLink>>(segment.ToArray(),
119                 ↪ duplicates));
120         }
121     }

```

```

113     }
114 }
115
116 [MethodImpl(MethodImplOptions.AggressiveInlining)]
117 private List<TLink> CollectDuplicatesForSegment(Segment<TLink> segment)
118 {
119     var duplicates = new List<TLink>();
120     var readAsElement = new HashSet<TLink>();
121     var restrictions = segment.ShiftRight();
122     restrictions[0] = _sequences.Constants.Any;
123     _sequences.Each(sequence =>
124     {
125         var sequenceIndex = sequence[_sequences.Constants.IndexPart];
126         duplicates.Add(sequenceIndex);
127         readAsElement.Add(sequenceIndex);
128         return _sequences.Constants.Continue;
129     }, restrictions);
130     if (duplicates.Any(x => _visited.Get(_addressToInt64Converter.Convert(x))))
131     {
132         return new List<TLink>();
133     }
134     foreach (var duplicate in duplicates)
135     {
136         var duplicateBitIndex = _addressToInt64Converter.Convert(duplicate);
137         _visited.Set(duplicateBitIndex);
138     }
139     if (_sequences is Sequences sequencesExperiments)
140     {
141         var partiallyMatched = sequencesExperiments.GetAllPartiallyMatchingSequences4((H
142             ↪ ashSet<ulong>)(object)readAsElement,
143             ↪ (IList<ulong>)segment);
144         foreach (var partiallyMatchedSequence in partiallyMatched)
145         {
146             var sequenceIndex =
147                 ↪ _uInt64ToAddressConverter.Convert(partiallyMatchedSequence);
148             duplicates.Add(sequenceIndex);
149         }
150     }
151     duplicates.Sort();
152     return duplicates;
153 }
154
155 [MethodImpl(MethodImplOptions.AggressiveInlining)]
156 private void PrintDuplicates(KeyValuePair<IList<TLink>, IList<TLink>> duplicatesItem)
157 {
158     if (!(_links is ILinks<ulong> ulongLinks))
159     {
160         return;
161     }
162     var duplicatesKey = duplicatesItem.Key;
163     var keyString = UnicodeMap.FromLinksToString((IList<ulong>)duplicatesKey);
164     Console.WriteLine($"> {keyString} ({string.Join(", ", duplicatesKey)})");
165     var duplicatesList = duplicatesItem.Value;
166     for (int i = 0; i < duplicatesList.Count; i++)
167     {
168         var sequenceIndex = _addressToUInt64Converter.Convert(duplicatesList[i]);
169         var formattedSequenceStructure = ulongLinks.FormatStructure(sequenceIndex, x =>
170             ↪ Point<ulong>.IsPartialPoint(x), (sb, link) => _ =
171             ↪ UnicodeMap.IsCharLink(link.Index) ?
172             ↪ sb.Append(UnicodeMap.FromLinkToChar(link.Index)) : sb.Append(link.Index));
173         Console.WriteLine(formattedSequenceStructure);
174         var sequenceString = UnicodeMap.FromSequenceLinkToString(sequenceIndex,
175             ↪ ulongLinks);
176         Console.WriteLine(sequenceString);
177     }
178     Console.WriteLine();
179 }
180 }
181 }
182 }
183 }

```

1.63 ./Platform.Data.Doublets/Sequences/Frequencies/Cache/LinkFrequenciesCache.cs

```

1 using System;
2 using System.Collections.Generic;
3 using System.Runtime.CompilerServices;
4 using Platform.Interfaces;
5 using Platform.Numbers;
6
7 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

```

```

8
9 namespace Platform.Data.Doublets.Sequences.Frequencies.Cache
10 {
11     /// <remarks>
12     /// Can be used to operate with many CompressingConverters (to keep global frequencies data
13     /// ↪ between them).
14     /// TODO: Extract interface to implement frequencies storage inside Links storage
15     /// </remarks>
16     public class LinkFrequenciesCache<TLink> : LinksOperatorBase<TLink>
17     {
18         private static readonly EqualityComparer<TLink> _equalityComparer =
19             ↪ EqualityComparer<TLink>.Default;
20         private static readonly Comparer<TLink> _comparer = Comparer<TLink>.Default;
21
22         private static readonly TLink _zero = default;
23         private static readonly TLink _one = Arithmetic.Increment(_zero);
24
25         private readonly Dictionary<Doublet<TLink>, LinkFrequency<TLink>> _doubletsCache;
26         private readonly ICounter<TLink, TLink> _frequencyCounter;
27
28         [MethodImpl(MethodImplOptions.AggressiveInlining)]
29         public LinkFrequenciesCache(ILinks<TLink> links, ICounter<TLink, TLink> frequencyCounter)
30             : base(links)
31         {
32             _doubletsCache = new Dictionary<Doublet<TLink>, LinkFrequency<TLink>>(4096,
33                 ↪ DoubletComparer<TLink>.Default);
34             _frequencyCounter = frequencyCounter;
35         }
36
37         [MethodImpl(MethodImplOptions.AggressiveInlining)]
38         public LinkFrequency<TLink> GetFrequency(TLink source, TLink target)
39         {
40             var doublet = new Doublet<TLink>(source, target);
41             return GetFrequency(ref doublet);
42         }
43
44         [MethodImpl(MethodImplOptions.AggressiveInlining)]
45         public LinkFrequency<TLink> GetFrequency(ref Doublet<TLink> doublet)
46         {
47             _doubletsCache.TryGetValue(doublet, out LinkFrequency<TLink> data);
48             return data;
49         }
50
51         [MethodImpl(MethodImplOptions.AggressiveInlining)]
52         public void IncrementFrequencies(ICollection<TLink> sequence)
53         {
54             for (var i = 1; i < sequence.Count; i++)
55             {
56                 IncrementFrequency(sequence[i - 1], sequence[i]);
57             }
58         }
59
60         [MethodImpl(MethodImplOptions.AggressiveInlining)]
61         public LinkFrequency<TLink> IncrementFrequency(TLink source, TLink target)
62         {
63             var doublet = new Doublet<TLink>(source, target);
64             return IncrementFrequency(ref doublet);
65         }
66
67         [MethodImpl(MethodImplOptions.AggressiveInlining)]
68         public void PrintFrequencies(ICollection<TLink> sequence)
69         {
70             for (var i = 1; i < sequence.Count; i++)
71             {
72                 PrintFrequency(sequence[i - 1], sequence[i]);
73             }
74         }
75
76         [MethodImpl(MethodImplOptions.AggressiveInlining)]
77         public void PrintFrequency(TLink source, TLink target)
78         {
79             var number = GetFrequency(source, target).Frequency;
80             Console.WriteLine("{0},{1} - {2}", source, target, number);
81         }
82
83         [MethodImpl(MethodImplOptions.AggressiveInlining)]
84         public LinkFrequency<TLink> IncrementFrequency(ref Doublet<TLink> doublet)
85         {
86             if (_doubletsCache.TryGetValue(doublet, out LinkFrequency<TLink> data))

```

```

84     {
85         data.IncrementFrequency();
86     }
87     else
88     {
89         var link = Links.SearchOrDefault(doublet.Source, doublet.Target);
90         data = new LinkFrequency<TLink>(_one, link);
91         if (!_equalityComparer.Equals(link, default))
92         {
93             data.Frequency = Arithmetic.Add(data.Frequency,
94                 ↪ _frequencyCounter.Count(link));
95         }
96         _doubletsCache.Add(doublet, data);
97     }
98     return data;
99 }
100 [MethodImpl(MethodImplOptions.AggressiveInlining)]
101 public void ValidateFrequencies()
102 {
103     foreach (var entry in _doubletsCache)
104     {
105         var value = entry.Value;
106         var linkIndex = value.Link;
107         if (!_equalityComparer.Equals(linkIndex, default))
108         {
109             var frequency = value.Frequency;
110             var count = _frequencyCounter.Count(linkIndex);
111             // TODO0: Why `frequency` always greater than `count` by 1?
112             if (((_comparer.Compare(frequency, count) > 0) &&
113                 ↪ (_comparer.Compare(Arithmetic.Subtract(frequency, count), _one) > 0))
114                 || ((_comparer.Compare(count, frequency) > 0) &&
115                 ↪ (_comparer.Compare(Arithmetic.Subtract(count, frequency), _one) > 0)))
116             {
117                 throw new InvalidOperationException("Frequencies validation failed.");
118             }
119             //else
120             //{
121             //    if (value.Frequency > 0)
122             //    {
123             //        var frequency = value.Frequency;
124             //        linkIndex = _createLink(entry.Key.Source, entry.Key.Target);
125             //        var count = _countLinkFrequency(linkIndex);
126             //        if ((frequency > count && frequency - count > 1) || (count > frequency
127             //            ↪ && count - frequency > 1))
128             //            throw new Exception("Frequencies validation failed.");
129             //    }
130             //}
131         }
132     }
133 }

```

1.64 ./Platform.Data.Doublets/Sequences/Frequencies/Cache/LinkFrequency.cs

```

1  using System.Runtime.CompilerServices;
2  using Platform.Numbers;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Sequences.Frequencies.Cache
7  {
8      public class LinkFrequency<TLink>
9      {
10         public TLink Frequency { get; set; }
11         public TLink Link { get; set; }
12
13         [MethodImpl(MethodImplOptions.AggressiveInlining)]
14         public LinkFrequency(TLink frequency, TLink link)
15         {
16             Frequency = frequency;
17             Link = link;
18         }
19
20         [MethodImpl(MethodImplOptions.AggressiveInlining)]
21         public LinkFrequency() { }
22
23         [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

24     public void IncrementFrequency() => Frequency = Arithmetic<TLink>.Increment(Frequency);
25
26     [MethodImpl(MethodImplOptions.AggressiveInlining)]
27     public void DecrementFrequency() => Frequency = Arithmetic<TLink>.Decrement(Frequency);
28
29     [MethodImpl(MethodImplOptions.AggressiveInlining)]
30     public override string ToString() => $"F: {Frequency}, L: {Link}";
31 }
32 }

```

1.65 ./Platform.Data.Doublets/Sequences/Frequencies/Cache/LinkToItsFrequencyValueConverter.cs

```

1  using System.Runtime.CompilerServices;
2  using Platform.Converters;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Sequences.Frequencies.Cache
7  {
8      public class FrequenciesCacheBasedLinkToItsFrequencyNumberConverter<TLink> :
9          ↳ IConverter<Doublet<TLink>, TLink>
10     {
11         private readonly LinkFrequenciesCache<TLink> _cache;
12
13         [MethodImpl(MethodImplOptions.AggressiveInlining)]
14         public
15             ↳ FrequenciesCacheBasedLinkToItsFrequencyNumberConverter(LinkFrequenciesCache<TLink>
16             ↳ cache) => _cache = cache;
17
18         [MethodImpl(MethodImplOptions.AggressiveInlining)]
19         public TLink Convert(Doublet<TLink> source) => _cache.GetFrequency(ref source).Frequency;
20     }
21 }

```

1.66 ./Platform.Data.Doublets/Sequences/Frequencies/Counters/MarkedSequenceSymbolFrequencyOneOffCounter.cs

```

1  using System.Runtime.CompilerServices;
2  using Platform.Interfaces;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Sequences.Frequencies.Counters
7  {
8      public class MarkedSequenceSymbolFrequencyOneOffCounter<TLink> :
9          ↳ SequenceSymbolFrequencyOneOffCounter<TLink>
10     {
11         private readonly ICriterionMatcher<TLink> _markedSequenceMatcher;
12
13         [MethodImpl(MethodImplOptions.AggressiveInlining)]
14         public MarkedSequenceSymbolFrequencyOneOffCounter(ILinks<TLink> links,
15             ↳ ICriterionMatcher<TLink> markedSequenceMatcher, TLink sequenceLink, TLink symbol)
16             : base(links, sequenceLink, symbol)
17             => _markedSequenceMatcher = markedSequenceMatcher;
18
19         [MethodImpl(MethodImplOptions.AggressiveInlining)]
20         public override TLink Count()
21         {
22             if (!_markedSequenceMatcher.IsMatched(_sequenceLink))
23             {
24                 return default;
25             }
26             return base.Count();
27         }
28     }
29 }

```

1.67 ./Platform.Data.Doublets/Sequences/Frequencies/Counters/SequenceSymbolFrequencyOneOffCounter.cs

```

1  using System.Collections.Generic;
2  using System.Runtime.CompilerServices;
3  using Platform.Interfaces;
4  using Platform.Numbers;
5  using Platform.Data.Sequences;
6
7  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9  namespace Platform.Data.Doublets.Sequences.Frequencies.Counters
10 {
11     public class SequenceSymbolFrequencyOneOffCounter<TLink> : ICounter<TLink>
12     {
13         private static readonly EqualityComparer<TLink> _equalityComparer =
14             ↳ EqualityComparer<TLink>.Default;
15         private static readonly Comparer<TLink> _comparer = Comparer<TLink>.Default;
16     }
17 }

```



```

15
16     protected readonly ILinks<TLink> _links;
17     protected readonly TLink _sequenceLink;
18     protected readonly TLink _symbol;
19     protected TLink _total;
20
21     [MethodImpl(MethodImplOptions.AggressiveInlining)]
22     public SequenceSymbolFrequencyOneOffCounter(ILinks<TLink> links, TLink sequenceLink,
23         ↪ TLink symbol)
24     {
25         _links = links;
26         _sequenceLink = sequenceLink;
27         _symbol = symbol;
28         _total = default;
29     }
30
31     [MethodImpl(MethodImplOptions.AggressiveInlining)]
32     public virtual TLink Count()
33     {
34         if (_comparer.Compare(_total, default) > 0)
35         {
36             return _total;
37         }
38         StopableSequenceWalker.WalkRight(_sequenceLink, _links.GetSource, _links.GetTarget,
39             ↪ IsElement, VisitElement);
40         return _total;
41     }
42
43     [MethodImpl(MethodImplOptions.AggressiveInlining)]
44     private bool IsElement(TLink x) => _equalityComparer.Equals(x, _symbol) ||
45         ↪ _links.IsPartialPoint(x); // TODO: Use SequenceElementCriteriaMatcher instead of
46         ↪ IsPartialPoint
47
48     [MethodImpl(MethodImplOptions.AggressiveInlining)]
49     private bool VisitElement(TLink element)
50     {
51         if (_equalityComparer.Equals(element, _symbol))
52         {
53             _total = Arithmetic.Increment(_total);
54         }
55         return true;
56     }
57 }

```

1.68 ./Platform.Data.Doublets/Sequences/Frequencies/Counters/TotalMarkedSequenceSymbolFrequencyCounter.

```

1 using System.Runtime.CompilerServices;
2 using Platform.Interfaces;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Sequences.Frequencies.Counters
7 {
8     public class TotalMarkedSequenceSymbolFrequencyCounter<TLink> : ICounter<TLink, TLink>
9     {
10         private readonly ILinks<TLink> _links;
11         private readonly ICriterionMatcher<TLink> _markedSequenceMatcher;
12
13         [MethodImpl(MethodImplOptions.AggressiveInlining)]
14         public TotalMarkedSequenceSymbolFrequencyCounter(ILinks<TLink> links,
15             ↪ ICriterionMatcher<TLink> markedSequenceMatcher)
16         {
17             _links = links;
18             _markedSequenceMatcher = markedSequenceMatcher;
19         }
20
21         [MethodImpl(MethodImplOptions.AggressiveInlining)]
22         public TLink Count(TLink argument) => new
23             ↪ TotalMarkedSequenceSymbolFrequencyOneOffCounter<TLink>(_links,
24             ↪ _markedSequenceMatcher, argument).Count();
25     }
26 }

```

1.69 ./Platform.Data.Doublets/Sequences/Frequencies/Counters/TotalMarkedSequenceSymbolFrequencyOneOffCounter.

```

1 using System.Runtime.CompilerServices;
2 using Platform.Interfaces;
3 using Platform.Numbers;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6

```

```

7 namespace Platform.Data.Doublets.Sequences.Frequencies.Counters
8 {
9     public class TotalMarkedSequenceSymbolFrequencyOneOffCounter<TLink> :
10         ↪ TotalSequenceSymbolFrequencyOneOffCounter<TLink>
11     {
12         private readonly ICriterionMatcher<TLink> _markedSequenceMatcher;
13
14         [MethodImpl(MethodImplOptions.AggressiveInlining)]
15         public TotalMarkedSequenceSymbolFrequencyOneOffCounter(ILinks<TLink> links,
16             ↪ ICriterionMatcher<TLink> markedSequenceMatcher, TLink symbol)
17             : base(links, symbol)
18             => _markedSequenceMatcher = markedSequenceMatcher;
19
20         [MethodImpl(MethodImplOptions.AggressiveInlining)]
21         protected override void CountSequenceSymbolFrequency(TLink link)
22         {
23             var symbolFrequencyCounter = new
24                 ↪ MarkedSequenceSymbolFrequencyOneOffCounter<TLink>(_links,
25                 ↪ _markedSequenceMatcher, link, _symbol);
26             _total = Arithmetic.Add(_total, symbolFrequencyCounter.Count());
27         }
28     }
29 }

```

1.70 ./Platform.Data.Doublets/Sequences/Frequencies/Counters/TotalSequenceSymbolFrequencyCounter.cs

```

1 using System.Runtime.CompilerServices;
2 using Platform.Interfaces;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Sequences.Frequencies.Counters
7 {
8     public class TotalSequenceSymbolFrequencyCounter<TLink> : ICounter<TLink, TLink>
9     {
10         private readonly ILinks<TLink> _links;
11
12         [MethodImpl(MethodImplOptions.AggressiveInlining)]
13         public TotalSequenceSymbolFrequencyCounter(ILinks<TLink> links) => _links = links;
14
15         [MethodImpl(MethodImplOptions.AggressiveInlining)]
16         public TLink Count(TLink symbol) => new
17             ↪ TotalSequenceSymbolFrequencyOneOffCounter<TLink>(_links, symbol).Count();
18     }
19 }

```

1.71 ./Platform.Data.Doublets/Sequences/Frequencies/Counters/TotalSequenceSymbolFrequencyOneOffCounter.

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3 using Platform.Interfaces;
4 using Platform.Numbers;
5
6 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8 namespace Platform.Data.Doublets.Sequences.Frequencies.Counters
9 {
10     public class TotalSequenceSymbolFrequencyOneOffCounter<TLink> : ICounter<TLink>
11     {
12         private static readonly EqualityComparer<TLink> _equalityComparer =
13             ↪ EqualityComparer<TLink>.Default;
14         private static readonly Comparer<TLink> _comparer = Comparer<TLink>.Default;
15
16         protected readonly ILinks<TLink> _links;
17         protected readonly TLink _symbol;
18         protected readonly HashSet<TLink> _visits;
19         protected TLink _total;
20
21         [MethodImpl(MethodImplOptions.AggressiveInlining)]
22         public TotalSequenceSymbolFrequencyOneOffCounter(ILinks<TLink> links, TLink symbol)
23         {
24             _links = links;
25             _symbol = symbol;
26             _visits = new HashSet<TLink>();
27             _total = default;
28         }
29
30         [MethodImpl(MethodImplOptions.AggressiveInlining)]
31         public TLink Count()
32         {
33             if (_comparer.Compare(_total, default) > 0 || _visits.Count > 0)
34             {

```

```

34         return _total;
35     }
36     CountCore(_symbol);
37     return _total;
38 }
39
40 [MethodImpl(MethodImplOptions.AggressiveInlining)]
41 private void CountCore(TLink link)
42 {
43     var any = _links.Constants.Any;
44     if (_equalityComparer.Equals(_links.Count(any, link), default))
45     {
46         CountSequenceSymbolFrequency(link);
47     }
48     else
49     {
50         _links.Each(EachElementHandler, any, link);
51     }
52 }
53
54 [MethodImpl(MethodImplOptions.AggressiveInlining)]
55 protected virtual void CountSequenceSymbolFrequency(TLink link)
56 {
57     var symbolFrequencyCounter = new SequenceSymbolFrequencyOneOffCounter<TLink>(_links,
58     ↪ link, _symbol);
59     _total = Arithmetic.Add(_total, symbolFrequencyCounter.Count());
60 }
61
62 [MethodImpl(MethodImplOptions.AggressiveInlining)]
63 private TLink EachElementHandler(IList<TLink> doublet)
64 {
65     var constants = _links.Constants;
66     var doubletIndex = doublet[constants.IndexPart];
67     if (_visits.Add(doubletIndex))
68     {
69         CountCore(doubletIndex);
70     }
71     return constants.Continue;
72 }
73 }

```

1.72 ./Platform.Data.Doublets/Sequences/HeightProviders/CachedSequenceHeightProvider.cs

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3 using Platform.Interfaces;
4 using Platform.Converters;
5
6 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8 namespace Platform.Data.Doublets.Sequences.HeightProviders
9 {
10     public class CachedSequenceHeightProvider<TLink> : LinksOperatorBase<TLink>,
11     ↪ ISequenceHeightProvider<TLink>
12     {
13         private static readonly EqualityComparer<TLink> _equalityComparer =
14         ↪ EqualityComparer<TLink>.Default;
15
16         private readonly TLink _heightPropertyMarker;
17         private readonly ISequenceHeightProvider<TLink> _baseHeightProvider;
18         private readonly IConverter<TLink> _addressToUnaryNumberConverter;
19         private readonly IConverter<TLink> _unaryNumberToAddressConverter;
20         private readonly IProperties<TLink, TLink, TLink> _propertyOperator;
21
22         [MethodImpl(MethodImplOptions.AggressiveInlining)]
23         public CachedSequenceHeightProvider(
24             ILinks<TLink> links,
25             ISequenceHeightProvider<TLink> baseHeightProvider,
26             IConverter<TLink> addressToUnaryNumberConverter,
27             IConverter<TLink> unaryNumberToAddressConverter,
28             TLink heightPropertyMarker,
29             IProperties<TLink, TLink, TLink> propertyOperator)
30             : base(links)
31         {
32             _heightPropertyMarker = heightPropertyMarker;
33             _baseHeightProvider = baseHeightProvider;
34             _addressToUnaryNumberConverter = addressToUnaryNumberConverter;
35             _unaryNumberToAddressConverter = unaryNumberToAddressConverter;
36             _propertyOperator = propertyOperator;
37         }
38     }
39 }

```

```

37     [MethodImpl(MethodImplOptions.AggressiveInlining)]
38     public TLink Get(TLink sequence)
39     {
40         TLink height;
41         var heightValue = _propertyOperator.GetValue(sequence, _heightPropertyMarker);
42         if (_equalityComparer.Equals(heightValue, default))
43         {
44             height = _baseHeightProvider.Get(sequence);
45             heightValue = _addressToUnaryNumberConverter.Convert(height);
46             _propertyOperator.SetValue(sequence, _heightPropertyMarker, heightValue);
47         }
48         else
49         {
50             height = _unaryNumberToAddressConverter.Convert(heightValue);
51         }
52         return height;
53     }
54 }
55 }

```

1.73 ./Platform.Data.Doublets/Sequences/HeightProviders/DefaultSequenceRightHeightProvider.cs

```

1  using System.Runtime.CompilerServices;
2  using Platform.Interfaces;
3  using Platform.Numbers;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.Sequences.HeightProviders
8  {
9      public class DefaultSequenceRightHeightProvider<TLink> : LinksOperatorBase<TLink>,
10         ↳ ISequenceHeightProvider<TLink>
11      {
12          private readonly ICriterionMatcher<TLink> _elementMatcher;
13
14          [MethodImpl(MethodImplOptions.AggressiveInlining)]
15          public DefaultSequenceRightHeightProvider(ILinks<TLink> links, ICriterionMatcher<TLink>
16             ↳ elementMatcher) : base(links) => _elementMatcher = elementMatcher;
17
18          [MethodImpl(MethodImplOptions.AggressiveInlining)]
19          public TLink Get(TLink sequence)
20          {
21              var height = default(TLink);
22              var pairOrElement = sequence;
23              while (!_elementMatcher.IsMatched(pairOrElement))
24              {
25                  pairOrElement = Links.GetTarget(pairOrElement);
26                  height = Arithmetic.Increment(height);
27              }
28              return height;
29          }
30      }
31 }

```

1.74 ./Platform.Data.Doublets/Sequences/HeightProviders/ISequenceHeightProvider.cs

```

1  using Platform.Interfaces;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.Sequences.HeightProviders
6  {
7      public interface ISequenceHeightProvider<TLink> : IProvider<TLink, TLink>
8      {
9      }
10 }

```

1.75 ./Platform.Data.Doublets/Sequences/Indexes/CachedFrequencyIncrementingSequenceIndex.cs

```

1  using System.Collections.Generic;
2  using System.Runtime.CompilerServices;
3  using Platform.Data.Doublets.Sequences.Frequencies.Cache;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.Sequences.Indexes
8  {
9      public class CachedFrequencyIncrementingSequenceIndex<TLink> : ISequenceIndex<TLink>
10     {
11         private static readonly EqualityComparer<TLink> _equalityComparer =
12             ↳ EqualityComparer<TLink>.Default;

```

```

13     private readonly LinkFrequenciesCache<TLink> _cache;
14
15     [MethodImpl(MethodImplOptions.AggressiveInlining)]
16     public CachedFrequencyIncrementingSequenceIndex(LinkFrequenciesCache<TLink> cache) =>
17         ↪ _cache = cache;
18
19     [MethodImpl(MethodImplOptions.AggressiveInlining)]
20     public bool Add(ICollection<TLink> sequence)
21     {
22         var indexed = true;
23         var i = sequence.Count;
24         while (--i >= 1 && (indexed = IsIndexedWithIncrement(sequence[i - 1], sequence[i])))
25             ↪ { }
26         for (; i >= 1; i--)
27         {
28             _cache.IncrementFrequency(sequence[i - 1], sequence[i]);
29         }
30         return indexed;
31     }
32
33     [MethodImpl(MethodImplOptions.AggressiveInlining)]
34     private bool IsIndexedWithIncrement(TLink source, TLink target)
35     {
36         var frequency = _cache.GetFrequency(source, target);
37         if (frequency == null)
38         {
39             return false;
40         }
41         var indexed = !_equalityComparer.Equals(frequency.Frequency, default);
42         if (indexed)
43         {
44             _cache.IncrementFrequency(source, target);
45         }
46         return indexed;
47     }
48
49     [MethodImpl(MethodImplOptions.AggressiveInlining)]
50     public bool MightContain(ICollection<TLink> sequence)
51     {
52         var indexed = true;
53         var i = sequence.Count;
54         while (--i >= 1 && (indexed = IsIndexed(sequence[i - 1], sequence[i]))) { }
55         return indexed;
56     }
57
58     [MethodImpl(MethodImplOptions.AggressiveInlining)]
59     private bool IsIndexed(TLink source, TLink target)
60     {
61         var frequency = _cache.GetFrequency(source, target);
62         if (frequency == null)
63         {
64             return false;
65         }
66         return !_equalityComparer.Equals(frequency.Frequency, default);
67     }
68 }

```

1.76 ./Platform.Data.Doublets/Sequences/Indexes/FrequencyIncrementingSequenceIndex.cs

```

1  using System.Collections.Generic;
2  using System.Runtime.CompilerServices;
3  using Platform.Interfaces;
4  using Platform.Incrementers;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Sequences.Indexes
9  {
10     public class FrequencyIncrementingSequenceIndex<TLink> : SequenceIndex<TLink>,
11         ↪ ISequenceIndex<TLink>
12     {
13         private static readonly EqualityComparer<TLink> _equalityComparer =
14             ↪ EqualityComparer<TLink>.Default;
15
16         private readonly IProperty<TLink, TLink> _frequencyPropertyOperator;
17         private readonly IIncrementer<TLink> _frequencyIncrementer;
18
19         [MethodImpl(MethodImplOptions.AggressiveInlining)]
20         public FrequencyIncrementingSequenceIndex(ICollection<TLink> links, IProperty<TLink, TLink>
21             ↪ frequencyPropertyOperator, IIncrementer<TLink> frequencyIncrementer)

```

```

19         : base(links)
20     {
21         _frequencyPropertyOperator = frequencyPropertyOperator;
22         _frequencyIncrementer = frequencyIncrementer;
23     }
24
25     [MethodImpl(MethodImplOptions.AggressiveInlining)]
26     public override bool Add(ICollection<TLink> sequence)
27     {
28         var indexed = true;
29         var i = sequence.Count;
30         while (--i >= 1 && (indexed = IsIndexedWithIncrement(sequence[i - 1], sequence[i])))
31             ↳ { }
32         for (; i >= 1; i--)
33         {
34             Increment(Links.GetOrCreate(sequence[i - 1], sequence[i]));
35         }
36         return indexed;
37     }
38
39     [MethodImpl(MethodImplOptions.AggressiveInlining)]
40     private bool IsIndexedWithIncrement(TLink source, TLink target)
41     {
42         var link = Links.SearchOrCreate(source, target);
43         var indexed = !_equalityComparer.Equals(link, default);
44         if (indexed)
45         {
46             Increment(link);
47         }
48         return indexed;
49     }
50
51     [MethodImpl(MethodImplOptions.AggressiveInlining)]
52     private void Increment(TLink link)
53     {
54         var previousFrequency = _frequencyPropertyOperator.Get(link);
55         var frequency = _frequencyIncrementer.Increment(previousFrequency);
56         _frequencyPropertyOperator.Set(link, frequency);
57     }
58 }

```

1.77 ./Platform.Data.Doublets/Sequences/Indexes/ISequenceIndex.cs

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Sequences.Indexes
7 {
8     public interface ISequenceIndex<TLink>
9     {
10         /// <summary>
11         /// Индексирует последовательность глобально, и возвращает значение,
12         /// определяющие была ли запрошенная последовательность проиндексирована ранее.
13         /// </summary>
14         /// <param name="sequence">Последовательность для индексации.</param>
15         [MethodImpl(MethodImplOptions.AggressiveInlining)]
16         bool Add(ICollection<TLink> sequence);
17
18         [MethodImpl(MethodImplOptions.AggressiveInlining)]
19         bool MightContain(ICollection<TLink> sequence);
20     }
21 }

```

1.78 ./Platform.Data.Doublets/Sequences/Indexes/SequenceIndex.cs

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Sequences.Indexes
7 {
8     public class SequenceIndex<TLink> : LinksOperatorBase<TLink>, ISequenceIndex<TLink>
9     {
10         private static readonly EqualityComparer<TLink> _equalityComparer =
11             ↳ EqualityComparer<TLink>.Default;
12
13         [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

13 public SequenceIndex(ILinks<TLink> links) : base(links) { }
14
15 [MethodImpl(MethodImplOptions.AggressiveInlining)]
16 public virtual bool Add(IList<TLink> sequence)
17 {
18     var indexed = true;
19     var i = sequence.Count;
20     while (--i >= 1 && (indexed =
21         ↪ !_equalityComparer.Equals(Links.SearchOrDefault(sequence[i - 1], sequence[i]),
22         ↪ default))) { }
23     for (; i >= 1; i--)
24     {
25         Links.GetOrCreate(sequence[i - 1], sequence[i]);
26     }
27     return indexed;
28 }
29
30 [MethodImpl(MethodImplOptions.AggressiveInlining)]
31 public virtual bool MightContain(IList<TLink> sequence)
32 {
33     var indexed = true;
34     var i = sequence.Count;
35     while (--i >= 1 && (indexed =
36         ↪ !_equalityComparer.Equals(Links.SearchOrDefault(sequence[i - 1], sequence[i]),
37         ↪ default))) { }
38     return indexed;
39 }
40 }
41 }

```

1.79 ./Platform.Data.Doublets/Sequences/Indexes/SynchronizedSequenceIndex.cs

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Sequences.Indexes
7 {
8     public class SynchronizedSequenceIndex<TLink> : ISequenceIndex<TLink>
9     {
10         private static readonly EqualityComparer<TLink> _equalityComparer =
11             ↪ EqualityComparer<TLink>.Default;
12
13         private readonly ISynchronizedLinks<TLink> _links;
14
15         [MethodImpl(MethodImplOptions.AggressiveInlining)]
16         public SynchronizedSequenceIndex(ISynchronizedLinks<TLink> links) => _links = links;
17
18         [MethodImpl(MethodImplOptions.AggressiveInlining)]
19         public bool Add(IList<TLink> sequence)
20         {
21             var indexed = true;
22             var i = sequence.Count;
23             var links = _links.Unsync;
24             _links.SyncRoot.ExecuteReadOperation(() =>
25             {
26                 while (--i >= 1 && (indexed =
27                     ↪ !_equalityComparer.Equals(links.SearchOrDefault(sequence[i - 1],
28                     ↪ sequence[i]), default))) { }
29             });
30             if (!indexed)
31             {
32                 _links.SyncRoot.ExecuteWriteOperation(() =>
33                 {
34                     for (; i >= 1; i--)
35                     {
36                         links.GetOrCreate(sequence[i - 1], sequence[i]);
37                     }
38                 });
39             }
40             return indexed;
41         }
42
43         [MethodImpl(MethodImplOptions.AggressiveInlining)]
44         public bool MightContain(IList<TLink> sequence)
45         {
46             var links = _links.Unsync;
47             return _links.SyncRoot.ExecuteReadOperation(() =>
48             {

```

```

46         var indexed = true;
47         var i = sequence.Count;
48         while (--i >= 1 && (indexed =
            ↳ !_equalityComparer.Equals(links.SearchOrDefault(sequence[i - 1],
            ↳ sequence[i]), default))) { }
49         return indexed;
50     });
51 }
52 }
53 }

```

1.80 ./Platform.Data.Doublets/Sequences/Indexes/Unindex.cs

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Sequences.Indexes
7 {
8     public class Unindex<TLink> : ISequenceIndex<TLink>
9     {
10         [MethodImpl(MethodImplOptions.AggressiveInlining)]
11         public virtual bool Add(IList<TLink> sequence) => false;
12
13         [MethodImpl(MethodImplOptions.AggressiveInlining)]
14         public virtual bool MightContain(IList<TLink> sequence) => true;
15     }
16 }

```

1.81 ./Platform.Data.Doublets/Sequences/Sequences.Experiments.cs

```

1 using System;
2 using System.Collections.Generic;
3 using System.Runtime.CompilerServices;
4 using System.Linq;
5 using System.Text;
6 using Platform.Collections;
7 using Platform.Collections.Sets;
8 using Platform.Collections.Stacks;
9 using Platform.Data.Exceptions;
10 using Platform.Data.Sequences;
11 using Platform.Data.Doublets.Sequences.Frequencies.Counters;
12 using Platform.Data.Doublets.Sequences.Walkers;
13 using LinkIndex = System.UInt64;
14 using Stack = System.Collections.Generic.Stack<ulong>;
15
16 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
17
18 namespace Platform.Data.Doublets.Sequences
19 {
20     partial class Sequences
21     {
22         #region Create All Variants (Not Practical)
23
24         /// <remarks>
25         /// Number of links that is needed to generate all variants for
26         /// sequence of length N corresponds to https://oeis.org/A014143/list sequence.
27         /// </remarks>
28         [MethodImpl(MethodImplOptions.AggressiveInlining)]
29         public ulong[] CreateAllVariants2(ulong[] sequence)
30         {
31             return _sync.ExecuteWriteOperation(() =>
32             {
33                 if (sequence.IsNullOrEmpty())
34                 {
35                     return Array.Empty<ulong>();
36                 }
37                 Links.EnsureLinkExists(sequence);
38                 if (sequence.Length == 1)
39                 {
38                     return sequence;
41                 }
42                 return CreateAllVariants2Core(sequence, 0, sequence.Length - 1);
43             });
44         }
45
46         [MethodImpl(MethodImplOptions.AggressiveInlining)]
47         private ulong[] CreateAllVariants2Core(ulong[] sequence, long startAt, long stopAt)
48         {
49             #if DEBUG
50                 if ((stopAt - startAt) < 0)

```



```

51     {
52         throw new ArgumentOutOfRangeException(nameof(startAt), "startAt должен быть
53         ↪ меньше или равен stopAt");
54     }
55 #endif
56     if ((stopAt - startAt) == 0)
57     {
58         return new[] { sequence[startAt] };
59     }
60     if ((stopAt - startAt) == 1)
61     {
62         return new[] { Links.Unsync.GetOrCreate(sequence[startAt], sequence[stopAt]) };
63     }
64     var variants = new ulong[(ulong)Platform.Numbers.Math.Catalan(stopAt - startAt)];
65     var last = 0;
66     for (var splitter = startAt; splitter < stopAt; splitter++)
67     {
68         var left = CreateAllVariants2Core(sequence, startAt, splitter);
69         var right = CreateAllVariants2Core(sequence, splitter + 1, stopAt);
70         for (var i = 0; i < left.Length; i++)
71         {
72             for (var j = 0; j < right.Length; j++)
73             {
74                 var variant = Links.Unsync.GetOrCreate(left[i], right[j]);
75                 if (variant == Constants.Null)
76                 {
77                     throw new NotImplementedException("Creation cancellation is not
78                     ↪ implemented.");
79                 }
80                 variants[last++] = variant;
81             }
82         }
83     }
84     return variants;
85 }
86 [MethodImpl(MethodImplOptions.AggressiveInlining)]
87 public List<ulong> CreateAllVariants1(params ulong[] sequence)
88 {
89     return _sync.ExecuteWriteOperation(() =>
90     {
91         if (sequence.IsNullOrEmpty())
92         {
93             return new List<ulong>();
94         }
95         Links.Unsync.EnsureLinkExists(sequence);
96         if (sequence.Length == 1)
97         {
98             return new List<ulong> { sequence[0] };
99         }
100         var results = new
101         ↪ List<ulong>((int)Platform.Numbers.Math.Catalan(sequence.Length));
102         return CreateAllVariants1Core(sequence, results);
103     });
104 }
105 [MethodImpl(MethodImplOptions.AggressiveInlining)]
106 private List<ulong> CreateAllVariants1Core(ulong[] sequence, List<ulong> results)
107 {
108     if (sequence.Length == 2)
109     {
110         var link = Links.Unsync.GetOrCreate(sequence[0], sequence[1]);
111         if (link == Constants.Null)
112         {
113             throw new NotImplementedException("Creation cancellation is not
114             ↪ implemented.");
115         }
116         results.Add(link);
117         return results;
118     }
119     var innerSequenceLength = sequence.Length - 1;
120     var innerSequence = new ulong[innerSequenceLength];
121     for (var li = 0; li < innerSequenceLength; li++)
122     {
123         var link = Links.Unsync.GetOrCreate(sequence[li], sequence[li + 1]);
124         if (link == Constants.Null)
125         {

```

```

124         throw new NotImplementedException("Creation cancellation is not
125         ↪ implemented.");
126     }
127     for (var isi = 0; isi < li; isi++)
128     {
129         innerSequence[isi] = sequence[isi];
130     }
131     innerSequence[li] = link;
132     for (var isi = li + 1; isi < innerSequenceLength; isi++)
133     {
134         innerSequence[isi] = sequence[isi + 1];
135     }
136     CreateAllVariants1Core(innerSequence, results);
137 }
138 return results;
139 }
140 #endregion
141
142 [MethodImpl(MethodImplOptions.AggressiveInlining)]
143 public HashSet<ulong> Each1(params ulong[] sequence)
144 {
145     var visitedLinks = new HashSet<ulong>(); // Заменить на bitstring
146     Each1(link =>
147     {
148         if (!visitedLinks.Contains(link))
149         {
150             visitedLinks.Add(link); // изучить почему случаются повторы
151         }
152         return true;
153     }, sequence);
154     return visitedLinks;
155 }
156
157 [MethodImpl(MethodImplOptions.AggressiveInlining)]
158 private void Each1(Func<ulong, bool> handler, params ulong[] sequence)
159 {
160     if (sequence.Length == 2)
161     {
162         Links.Unsync.Each(sequence[0], sequence[1], handler);
163     }
164     else
165     {
166         var innerSequenceLength = sequence.Length - 1;
167         for (var li = 0; li < innerSequenceLength; li++)
168         {
169             var left = sequence[li];
170             var right = sequence[li + 1];
171             if (left == 0 && right == 0)
172             {
173                 continue;
174             }
175             var linkIndex = li;
176             ulong[] innerSequence = null;
177             Links.Unsync.Each(doublet =>
178             {
179                 if (innerSequence == null)
180                 {
181                     innerSequence = new ulong[innerSequenceLength];
182                     for (var isi = 0; isi < linkIndex; isi++)
183                     {
184                         innerSequence[isi] = sequence[isi];
185                     }
186                     for (var isi = linkIndex + 1; isi < innerSequenceLength; isi++)
187                     {
188                         innerSequence[isi] = sequence[isi + 1];
189                     }
190                 }
191                 innerSequence[linkIndex] = doublet[Constants.IndexPart];
192                 Each1(handler, innerSequence);
193                 return Constants.Continue;
194             }, Constants.Any, left, right);
195         }
196     }
197 }
198
199 [MethodImpl(MethodImplOptions.AggressiveInlining)]
200 public HashSet<ulong> EachPart(params ulong[] sequence)
201 {

```

```

202     var visitedLinks = new HashSet<ulong>(); // Заменить на bitstring
203     EachPartCore(link =>
204     {
205         var linkIndex = link[Constants.IndexPart];
206         if (!visitedLinks.Contains(linkIndex))
207         {
208             visitedLinks.Add(linkIndex); // изучить почему случаются повторы
209         }
210         return Constants.Continue;
211     }, sequence);
212     return visitedLinks;
213 }
214
215 [MethodImpl(MethodImplOptions.AggressiveInlining)]
216 public void EachPart(Func<IList<LinkIndex>, LinkIndex> handler, params ulong[] sequence)
217 {
218     var visitedLinks = new HashSet<ulong>(); // Заменить на bitstring
219     EachPartCore(link =>
220     {
221         var linkIndex = link[Constants.IndexPart];
222         if (!visitedLinks.Contains(linkIndex))
223         {
224             visitedLinks.Add(linkIndex); // изучить почему случаются повторы
225             return handler(new LinkAddress<LinkIndex>(linkIndex));
226         }
227         return Constants.Continue;
228     }, sequence);
229 }
230
231 [MethodImpl(MethodImplOptions.AggressiveInlining)]
232 private void EachPartCore(Func<IList<LinkIndex>, LinkIndex> handler, params ulong[]
233 ↪ sequence)
234 {
235     if (sequence.IsNullOrEmpty())
236     {
237         return;
238     }
239     Links.EnsureLinkIsAnyOrExists(sequence);
240     if (sequence.Length == 1)
241     {
242         var link = sequence[0];
243         if (link > 0)
244         {
245             handler(new LinkAddress<LinkIndex>(link));
246         }
247         else
248         {
249             Links.Each(Constants.Any, Constants.Any, handler);
250         }
251     }
252     else if (sequence.Length == 2)
253     {
254         //_links.Each(sequence[0], sequence[1], handler);
255         //  o_|      x_o ...
256         // x_|      |__|
257         Links.Each(sequence[1], Constants.Any, doublet =>
258         {
259             var match = Links.SearchOrDefault(sequence[0], doublet);
260             if (match != Constants.Null)
261             {
262                 handler(new LinkAddress<LinkIndex>(match));
263             }
264             return true;
265         });
266         // |_x      ... x_o
267         // |_o      |__|
268         Links.Each(Constants.Any, sequence[0], doublet =>
269         {
270             var match = Links.SearchOrDefault(doublet, sequence[1]);
271             if (match != 0)
272             {
273                 handler(new LinkAddress<LinkIndex>(match));
274             }
275             return true;
276         });
277         //      .x o_.
278         //      |__|
279         PartialStepRight(x => handler(x), sequence[0], sequence[1]);

```

```

279     }
280     else
281     {
282         throw new NotImplementedException();
283     }
284 }
285
286 [MethodImpl(MethodImplOptions.AggressiveInlining)]
287 private void PartialStepRight(Action<IList<LinkIndex>> handler, ulong left, ulong right)
288 {
289     Links.Unsync.Each(Constants.Any, left, doublet =>
290     {
291         StepRight(handler, doublet, right);
292         if (left != doublet)
293         {
294             PartialStepRight(handler, doublet, right);
295         }
296         return true;
297     });
298 }
299
300 [MethodImpl(MethodImplOptions.AggressiveInlining)]
301 private void StepRight(Action<IList<LinkIndex>> handler, ulong left, ulong right)
302 {
303     Links.Unsync.Each(left, Constants.Any, rightStep =>
304     {
305         TryStepRightUp(handler, right, rightStep);
306         return true;
307     });
308 }
309
310 [MethodImpl(MethodImplOptions.AggressiveInlining)]
311 private void TryStepRightUp(Action<IList<LinkIndex>> handler, ulong right, ulong
↪ stepFrom)
312 {
313     var upStep = stepFrom;
314     var firstSource = Links.Unsync.GetTarget(upStep);
315     while (firstSource != right && firstSource != upStep)
316     {
317         upStep = firstSource;
318         firstSource = Links.Unsync.GetSource(upStep);
319     }
320     if (firstSource == right)
321     {
322         handler(new LinkAddress<LinkIndex>(stepFrom));
323     }
324 }
325
326 // TODO: Test
327 [MethodImpl(MethodImplOptions.AggressiveInlining)]
328 private void PartialStepLeft(Action<IList<LinkIndex>> handler, ulong left, ulong right)
329 {
330     Links.Unsync.Each(right, Constants.Any, doublet =>
331     {
332         StepLeft(handler, left, doublet);
333         if (right != doublet)
334         {
335             PartialStepLeft(handler, left, doublet);
336         }
337         return true;
338     });
339 }
340
341 [MethodImpl(MethodImplOptions.AggressiveInlining)]
342 private void StepLeft(Action<IList<LinkIndex>> handler, ulong left, ulong right)
343 {
344     Links.Unsync.Each(Constants.Any, right, leftStep =>
345     {
346         TryStepLeftUp(handler, left, leftStep);
347         return true;
348     });
349 }
350
351 [MethodImpl(MethodImplOptions.AggressiveInlining)]
352 private void TryStepLeftUp(Action<IList<LinkIndex>> handler, ulong left, ulong stepFrom)
353 {
354     var upStep = stepFrom;
355     var firstTarget = Links.Unsync.GetSource(upStep);
356     while (firstTarget != left && firstTarget != upStep)

```

```

357     {
358         upStep = firstTarget;
359         firstTarget = Links.Unsync.GetTarget(upStep);
360     }
361     if (firstTarget == left)
362     {
363         handler(new LinkAddress<LinkIndex>(stepFrom));
364     }
365 }
366
367 [MethodImpl(MethodImplOptions.AggressiveInlining)]
368 private bool StartsWith(ulong sequence, ulong link)
369 {
370     var upStep = sequence;
371     var firstSource = Links.Unsync.GetSource(upStep);
372     while (firstSource != link && firstSource != upStep)
373     {
374         upStep = firstSource;
375         firstSource = Links.Unsync.GetSource(upStep);
376     }
377     return firstSource == link;
378 }
379
380 [MethodImpl(MethodImplOptions.AggressiveInlining)]
381 private bool EndsWith(ulong sequence, ulong link)
382 {
383     var upStep = sequence;
384     var lastTarget = Links.Unsync.GetTarget(upStep);
385     while (lastTarget != link && lastTarget != upStep)
386     {
387         upStep = lastTarget;
388         lastTarget = Links.Unsync.GetTarget(upStep);
389     }
390     return lastTarget == link;
391 }
392
393 [MethodImpl(MethodImplOptions.AggressiveInlining)]
394 public List<ulong> GetAllMatchingSequences0(params ulong[] sequence)
395 {
396     return _sync.ExecuteReadOperation(() =>
397     {
398         var results = new List<ulong>();
399         if (sequence.Length > 0)
400         {
401             Links.EnsureLinkExists(sequence);
402             var firstElement = sequence[0];
403             if (sequence.Length == 1)
404             {
405                 results.Add(firstElement);
406                 return results;
407             }
408             if (sequence.Length == 2)
409             {
410                 var doublet = Links.SearchOrDefault(firstElement, sequence[1]);
411                 if (doublet != Constants.Null)
412                 {
413                     results.Add(doublet);
414                 }
415                 return results;
416             }
417             var linksInSequence = new HashSet<ulong>(sequence);
418             void handler(ICollection<LinkIndex> result)
419             {
420                 var resultIndex = result[Links.Constants.IndexPart];
421                 var filterPosition = 0;
422                 StopableSequenceWalker.WalkRight(resultIndex, Links.Unsync.GetSource,
423                     ↪ Links.Unsync.GetTarget,
424                     ↪ x => linksInSequence.Contains(x) || Links.Unsync.GetTarget(x) == x,
425                     ↪ x =>
426                     {
427                         if (filterPosition == sequence.Length)
428                         {
429                             filterPosition = -2; // Длиннее чем нужно
430                             return false;
431                         }
432                     }
433                     if (x != sequence[filterPosition])
434                     {
435                         filterPosition = -1;
436                         return false; // Начинается иначе
437                     }
438                 }
439             }
440         }
441     });
442 }

```

```

434         }
435         filterPosition++;
436
437         return true;
438     });
439     if (filterPosition == sequence.Length)
440     {
441         results.Add(resultIndex);
442     }
443 }
444 if (sequence.Length >= 2)
445 {
446     StepRight(handler, sequence[0], sequence[1]);
447 }
448 var last = sequence.Length - 2;
449 for (var i = 1; i < last; i++)
450 {
451     PartialStepRight(handler, sequence[i], sequence[i + 1]);
452 }
453 if (sequence.Length >= 3)
454 {
455     StepLeft(handler, sequence[sequence.Length - 2],
456         ↪ sequence[sequence.Length - 1]);
457 }
458 return results;
459 });
460 }
461
462 [MethodImpl(MethodImplOptions.AggressiveInlining)]
463 public HashSet<ulong> GetAllMatchingSequences1(params ulong[] sequence)
464 {
465     return _sync.ExecuteReadOperation(() =>
466     {
467         var results = new HashSet<ulong>();
468         if (sequence.Length > 0)
469         {
470             Links.EnsureLinkExists(sequence);
471             var firstElement = sequence[0];
472             if (sequence.Length == 1)
473             {
474                 results.Add(firstElement);
475                 return results;
476             }
477             if (sequence.Length == 2)
478             {
479                 var doublet = Links.SearchOrDefault(firstElement, sequence[1]);
480                 if (doublet != Constants.Null)
481                 {
482                     results.Add(doublet);
483                 }
484                 return results;
485             }
486             var matcher = new Matcher(this, sequence, results, null);
487             if (sequence.Length >= 2)
488             {
489                 StepRight(matcher.AddFullMatchedToResults, sequence[0], sequence[1]);
490             }
491             var last = sequence.Length - 2;
492             for (var i = 1; i < last; i++)
493             {
494                 PartialStepRight(matcher.AddFullMatchedToResults, sequence[i],
495                     ↪ sequence[i + 1]);
496             }
497             if (sequence.Length >= 3)
498             {
499                 StepLeft(matcher.AddFullMatchedToResults, sequence[sequence.Length - 2],
500                     ↪ sequence[sequence.Length - 1]);
501             }
502             return results;
503         }
504     });
505 }
506
507 public const int MaxSequenceFormatSize = 200;
508
509 [MethodImpl(MethodImplOptions.AggressiveInlining)]
510 public string FormatSequence(LinkIndex sequenceLink, params LinkIndex[] knownElements)
511     ↪ => FormatSequence(sequenceLink, (sb, x) => sb.Append(x), true, knownElements);

```

```

509 [MethodImpl(MethodImplOptions.AggressiveInlining)]
510 public string FormatSequence(LinkIndex sequenceLink, Action<StringBuilder, LinkIndex>
511     ↪ elementToString, bool insertComma, params LinkIndex[] knownElements) =>
    ↪ Links.SyncRoot.ExecuteReadOperation(() => FormatSequence(Links.Unsync, sequenceLink,
    ↪ elementToString, insertComma, knownElements));

512 [MethodImpl(MethodImplOptions.AggressiveInlining)]
513 private string FormatSequence(ILinks<LinkIndex> links, LinkIndex sequenceLink,
514     ↪ Action<StringBuilder, LinkIndex> elementToString, bool insertComma, params
    ↪ LinkIndex[] knownElements)
515 {
516     var linksInSequence = new HashSet<ulong>(knownElements);
517     //var entered = new HashSet<ulong>();
518     var sb = new StringBuilder();
519     sb.Append('{');
520     if (links.Exists(sequenceLink))
521     {
522         StopableSequenceWalker.WalkRight(sequenceLink, links.GetSource, links.GetTarget,
523             x => linksInSequence.Contains(x) || links.IsPartialPoint(x), element => //
    ↪     entered.AddAndReturnVoid, x => { }, entered.DoNotContains
524         {
525             if (insertComma && sb.Length > 1)
526             {
527                 sb.Append(',');
528             }
529             //if (entered.Contains(element))
530             //{
531                 sb.Append('{');
532                 elementToString(sb, element);
533                 sb.Append('}');
534             //}
535             //else
536             elementToString(sb, element);
537             if (sb.Length < MaxSequenceFormatSize)
538             {
539                 return true;
540             }
541             sb.Append(insertComma ? ", ..." : "...");
542             return false;
543         });
544     }
545     sb.Append('}');
546     return sb.ToString();
547 }

548 [MethodImpl(MethodImplOptions.AggressiveInlining)]
549 public string SafeFormatSequence(LinkIndex sequenceLink, params LinkIndex[]
550     ↪ knownElements) => SafeFormatSequence(sequenceLink, (sb, x) => sb.Append(x), true,
    ↪ knownElements);

551 [MethodImpl(MethodImplOptions.AggressiveInlining)]
552 public string SafeFormatSequence(LinkIndex sequenceLink, Action<StringBuilder,
553     ↪ LinkIndex> elementToString, bool insertComma, params LinkIndex[] knownElements) =>
    ↪ Links.SyncRoot.ExecuteReadOperation(() => SafeFormatSequence(Links.Unsync,
    ↪ sequenceLink, elementToString, insertComma, knownElements));

554 [MethodImpl(MethodImplOptions.AggressiveInlining)]
555 private string SafeFormatSequence(ILinks<LinkIndex> links, LinkIndex sequenceLink,
556     ↪ Action<StringBuilder, LinkIndex> elementToString, bool insertComma, params
    ↪ LinkIndex[] knownElements)
557 {
558     var linksInSequence = new HashSet<ulong>(knownElements);
559     var entered = new HashSet<ulong>();
560     var sb = new StringBuilder();
561     sb.Append('{');
562     if (links.Exists(sequenceLink))
563     {
564         StopableSequenceWalker.WalkRight(sequenceLink, links.GetSource, links.GetTarget,
565             x => linksInSequence.Contains(x) || links.IsFullPoint(x),
    ↪     entered.AddAndReturnVoid, x => { }, entered.DoNotContains, element =>
566         {
567             if (insertComma && sb.Length > 1)
568             {
569                 sb.Append(',');
570             }
571             if (entered.Contains(element))

```

```

572         {
573             sb.Append('{');
574             elementToString(sb, element);
575             sb.Append('}');
576         }
577         else
578         {
579             elementToString(sb, element);
580         }
581         if (sb.Length < MaxSequenceFormatSize)
582         {
583             return true;
584         }
585         sb.Append(insertComma ? ", ..." : "...");
586         return false;
587     });
588 }
589 sb.Append('}');
590 return sb.ToString();
591 }
592
593 [MethodImpl(MethodImplOptions.AggressiveInlining)]
594 public List<ulong> GetAllPartiallyMatchingSequences0(params ulong[] sequence)
595 {
596     return _sync.ExecuteReadOperation(() =>
597     {
598         if (sequence.Length > 0)
599         {
600             Links.EnsureLinkExists(sequence);
601             var results = new HashSet<ulong>();
602             for (var i = 0; i < sequence.Length; i++)
603             {
604                 AllUsagesCore(sequence[i], results);
605             }
606             var filteredResults = new List<ulong>();
607             var linksInSequence = new HashSet<ulong>(sequence);
608             foreach (var result in results)
609             {
610                 var filterPosition = -1;
611                 StopableSequenceWalker.WalkRight(result, Links.Unsync.GetSource,
612                     ↪ Links.Unsync.GetTarget,
613                     ↪ x => linksInSequence.Contains(x) || Links.Unsync.GetTarget(x) == x,
614                     {
615                         if (filterPosition == (sequence.Length - 1))
616                         {
617                             return false;
618                         }
619                         if (filterPosition >= 0)
620                         {
621                             if (x == sequence[filterPosition + 1])
622                             {
623                                 filterPosition++;
624                             }
625                             else
626                             {
627                                 return false;
628                             }
629                         }
630                         if (filterPosition < 0)
631                         {
632                             if (x == sequence[0])
633                             {
634                                 filterPosition = 0;
635                             }
636                         }
637                         return true;
638                     }
639                 ));
640             if (filterPosition == (sequence.Length - 1))
641             {
642                 filteredResults.Add(result);
643             }
644         }
645         return filteredResults;
646     }
647     return new List<ulong>();
648 });

```



```

649 [MethodImpl(MethodImplOptions.AggressiveInlining)]
650 public HashSet<ulong> GetAllPartiallyMatchingSequences1(params ulong[] sequence)
651 {
652     return _sync.ExecuteReadOperation(() =>
653     {
654         if (sequence.Length > 0)
655         {
656             Links.EnsureLinkExists(sequence);
657             var results = new HashSet<ulong>();
658             for (var i = 0; i < sequence.Length; i++)
659             {
660                 AllUsagesCore(sequence[i], results);
661             }
662             var filteredResults = new HashSet<ulong>();
663             var matcher = new Matcher(this, sequence, filteredResults, null);
664             matcher.AddAllPartialMatchedToResults(results);
665             return filteredResults;
666         }
667         return new HashSet<ulong>();
668     });
669 }
670
671 [MethodImpl(MethodImplOptions.AggressiveInlining)]
672 public bool GetAllPartiallyMatchingSequences2(Func<IList<LinkIndex>, LinkIndex> handler,
673     ↪ params ulong[] sequence)
674 {
675     return _sync.ExecuteReadOperation(() =>
676     {
677         if (sequence.Length > 0)
678         {
679             Links.EnsureLinkExists(sequence);
680
681             var results = new HashSet<ulong>();
682             var filteredResults = new HashSet<ulong>();
683             var matcher = new Matcher(this, sequence, filteredResults, handler);
684             for (var i = 0; i < sequence.Length; i++)
685             {
686                 if (!AllUsagesCore1(sequence[i], results, matcher.HandlePartialMatched))
687                 {
688                     return false;
689                 }
690             }
691             return true;
692         }
693         return true;
694     });
695 }
696
697 //public HashSet<ulong> GetAllPartiallyMatchingSequences3(params ulong[] sequence)
698 //{
699 //    return Sync.ExecuteReadOperation(() =>
700 //    {
701 //        if (sequence.Length > 0)
702 //        {
703 //            _links.EnsureEachLinkIsAnyOrExists(sequence);
704 //
705 //            var firstResults = new HashSet<ulong>();
706 //            var lastResults = new HashSet<ulong>();
707 //
708 //            var first = sequence.First(x => x != LinksConstants.Any);
709 //            var last = sequence.Last(x => x != LinksConstants.Any);
710 //
711 //            AllUsagesCore(first, firstResults);
712 //            AllUsagesCore(last, lastResults);
713 //
714 //            firstResults.IntersectWith(lastResults);
715 //
716 //            //for (var i = 0; i < sequence.Length; i++)
717 //            //    AllUsagesCore(sequence[i], results);
718 //
719 //            var filteredResults = new HashSet<ulong>();
720 //            var matcher = new Matcher(this, sequence, filteredResults, null);
721 //            matcher.AddAllPartialMatchedToResults(firstResults);
722 //            return filteredResults;
723 //        }
724 //
725 //        return new HashSet<ulong>();
726 //    });
727 //}

```

```

727 [MethodImpl(MethodImplOptions.AggressiveInlining)]
728 public HashSet<ulong> GetAllPartiallyMatchingSequences3(params ulong[] sequence)
729 {
730     return _sync.ExecuteReadOperation((Func<HashSet<ulong>>)(() =>
731     {
732         if (sequence.Length > 0)
733         {
734             ILinksExtensions.EnsureLinkIsAnyOrExists<ulong>(Links,
735                 ↪ (IList<ulong>)sequence);
736             var firstResults = new HashSet<ulong>();
737             var lastResults = new HashSet<ulong>();
738             var first = sequence.First(x => x != Constants.Any);
739             var last = sequence.Last(x => x != Constants.Any);
740             AllUsagesCore(first, firstResults);
741             AllUsagesCore(last, lastResults);
742             firstResults.IntersectWith(lastResults);
743             //for (var i = 0; i < sequence.Length; i++)
744             //    AllUsagesCore(sequence[i], results);
745             var filteredResults = new HashSet<ulong>();
746             var matcher = new Matcher(this, sequence, filteredResults, null);
747             matcher.AddAllPartialMatchedToResults(firstResults);
748             return filteredResults;
749         }
750         return new HashSet<ulong>();
751     }));
752 }
753
754 [MethodImpl(MethodImplOptions.AggressiveInlining)]
755 public HashSet<ulong> GetAllPartiallyMatchingSequences4(HashSet<ulong> readAsElements,
756     ↪ IList<ulong> sequence)
757 {
758     return _sync.ExecuteReadOperation(() =>
759     {
760         if (sequence.Count > 0)
761         {
762             Links.EnsureLinkExists(sequence);
763             var results = new HashSet<LinkIndex>();
764             //var nextResults = new HashSet<ulong>();
765             //for (var i = 0; i < sequence.Length; i++)
766             //{
767             //    AllUsagesCore(sequence[i], nextResults);
768             //    if (results.IsNullOrEmpty())
769             //    {
770             //        results = nextResults;
771             //        nextResults = new HashSet<ulong>();
772             //    }
773             //    else
774             //    {
775             //        results.IntersectWith(nextResults);
776             //        nextResults.Clear();
777             //    }
778             //}
779             var collector1 = new AllUsagesCollector1(Links.Unsync, results);
780             collector1.Collect(Links.Unsync.GetLink(sequence[0]));
781             var next = new HashSet<ulong>();
782             for (var i = 1; i < sequence.Count; i++)
783             {
784                 var collector = new AllUsagesCollector1(Links.Unsync, next);
785                 collector.Collect(Links.Unsync.GetLink(sequence[i]));
786
787                 results.IntersectWith(next);
788                 next.Clear();
789             }
790             var filteredResults = new HashSet<ulong>();
791             var matcher = new Matcher(this, sequence, filteredResults, null,
792                 ↪ readAsElements);
793             matcher.AddAllPartialMatchedToResultsAndReadAsElements(results.OrderBy(x =>
794                 ↪ x)); // OrderBy is a Hack
795             return filteredResults;
796         }
797         return new HashSet<ulong>();
798     }));
799 }
800
801 // Does not work
802 //public HashSet<ulong> GetAllPartiallyMatchingSequences5(HashSet<ulong> readAsElements,
803     ↪ params ulong[] sequence)

```

```

800 // {
801 //     var visited = new HashSet<ulong>();
802 //     var results = new HashSet<ulong>();
803 //     var matcher = new Matcher(this, sequence, visited, x => { results.Add(x); return
    ↪ true; }, readAsElements);
804 //     var last = sequence.Length - 1;
805 //     for (var i = 0; i < last; i++)
806 //     {
807 //         PartialStepRight(matcher.PartialMatch, sequence[i], sequence[i + 1]);
808 //     }
809 //     return results;
810 // }
811
812 [MethodImpl(MethodImplOptions.AggressiveInlining)]
813 public List<ulong> GetAllPartiallyMatchingSequences(params ulong[] sequence)
814 {
815     return _sync.ExecuteReadOperation(() =>
816     {
817         if (sequence.Length > 0)
818         {
819             Links.EnsureLinkExists(sequence);
820             //var firstElement = sequence[0];
821             //if (sequence.Length == 1)
822             //{
823             //    //results.Add(firstElement);
824             //    return results;
825             //}
826             //if (sequence.Length == 2)
827             //{
828             //    //var doublet = _links.SearchCore(firstElement, sequence[1]);
829             //    //if (doublet != Doublets.Links.Null)
830             //    //    results.Add(doublet);
831             //    return results;
832             //}
833             //var lastElement = sequence[sequence.Length - 1];
834             //Func<ulong, bool> handler = x =>
835             //{
836             //    if (StartsWith(x, firstElement) && EndsWith(x, lastElement))
837             //        ↪ results.Add(x);
838             //    return true;
839             //};
840             //if (sequence.Length >= 2)
841             //    StepRight(handler, sequence[0], sequence[1]);
842             //var last = sequence.Length - 2;
843             //for (var i = 1; i < last; i++)
844             //    PartialStepRight(handler, sequence[i], sequence[i + 1]);
845             //if (sequence.Length >= 3)
846             //    StepLeft(handler, sequence[sequence.Length - 2],
847             //        ↪ sequence[sequence.Length - 1]);
848             //if (sequence.Length == 1)
849             //{
850             //    throw new NotImplementedException(); // all sequences, containing
851             //        ↪ this element?
852             //}
853             //if (sequence.Length == 2)
854             //{
855             //    var results = new List<ulong>();
856             //    PartialStepRight(results.Add, sequence[0], sequence[1]);
857             //    return results;
858             //}
859             //var matches = new List<List<ulong>>();
860             //var last = sequence.Length - 1;
861             //for (var i = 0; i < last; i++)
862             //{
863             //    var results = new List<ulong>();
864             //    //StepRight(results.Add, sequence[i], sequence[i + 1]);
865             //    PartialStepRight(results.Add, sequence[i], sequence[i + 1]);
866             //    if (results.Count > 0)
867             //        matches.Add(results);
868             //    else
869             //        return results;
870             //    if (matches.Count == 2)
871             //    {
872                 var merged = new List<ulong>();
873                 for (var j = 0; j < matches[0].Count; j++)
874                     for (var k = 0; k < matches[1].Count; k++)

```

```

872         CloseInnerConnections(merged.Add, matches[0][j],
873         ↪ matches[1][k]);
874         if (merged.Count > 0)
875             matches = new List<List<ulong>> { merged };
876         else
877             return new List<ulong>();
878     }
879     if (matches.Count > 0)
880     {
881         var usages = new HashSet<ulong>();
882         for (int i = 0; i < sequence.Length; i++)
883         {
884             AllUsagesCore(sequence[i], usages);
885         }
886         //for (int i = 0; i < matches[0].Count; i++)
887         //    AllUsagesCore(matches[0][i], usages);
888         //usages.UnionWith(matches[0]);
889         return usages.ToList();
890     }
891     var firstLinkUsages = new HashSet<ulong>();
892     AllUsagesCore(sequence[0], firstLinkUsages);
893     firstLinkUsages.Add(sequence[0]);
894     //var previousMatchings = firstLinkUsages.ToList(); //new List<ulong>() {
895     ↪ sequence[0] }; // or all sequences, containing this element?
896     //return GetAllPartiallyMatchingSequencesCore(sequence, firstLinkUsages,
897     ↪ 1).ToList();
898     var results = new HashSet<ulong>();
899     foreach (var match in GetAllPartiallyMatchingSequencesCore(sequence,
900     ↪ firstLinkUsages, 1))
901     {
902         AllUsagesCore(match, results);
903     }
904     return results.ToList();
905 }
906 return new List<ulong>();
907 });
908 }
909
910 /// <remarks>
911 /// TODO: Может потребоваться ограничение на уровень глубины рекурсии
912 /// </remarks>
913 [MethodImpl(MethodImplOptions.AggressiveInlining)]
914 public HashSet<ulong> AllUsages(ulong link)
915 {
916     return _sync.ExecuteReadOperation(() =>
917     {
918         var usages = new HashSet<ulong>();
919         AllUsagesCore(link, usages);
920         return usages;
921     });
922 }
923
924 // При сборе всех использований (последовательностей) можно сохранять обратный путь к
925 ↪ той связи с которой начинался поиск (STTTSSSTT),
926 // причём достаточно одного бита для хранения перехода влево или вправо
927 [MethodImpl(MethodImplOptions.AggressiveInlining)]
928 private void AllUsagesCore(ulong link, HashSet<ulong> usages)
929 {
930     bool handler(ulong doublet)
931     {
932         if (usages.Add(doublet))
933         {
934             AllUsagesCore(doublet, usages);
935         }
936         return true;
937     }
938     Links.Unsync.Each(link, Constants.Any, handler);
939     Links.Unsync.Each(Constants.Any, link, handler);
940 }
941
942 [MethodImpl(MethodImplOptions.AggressiveInlining)]
943 public HashSet<ulong> AllBottomUsages(ulong link)
944 {
945     return _sync.ExecuteReadOperation(() =>
946     {
947         var visits = new HashSet<ulong>();

```

```

944     var usages = new HashSet<ulong>();
945     AllBottomUsagesCore(link, visits, usages);
946     return usages;
947 });
948 }
949
950 [MethodImpl(MethodImplOptions.AggressiveInlining)]
951 private void AllBottomUsagesCore(ulong link, HashSet<ulong> visits, HashSet<ulong>
    ↳ usages)
952 {
953     bool handler(ulong doublet)
954     {
955         if (visits.Add(doublet))
956         {
957             AllBottomUsagesCore(doublet, visits, usages);
958         }
959         return true;
960     }
961     if (Links.Unsync.Count(Constants.Any, link) == 0)
962     {
963         usages.Add(link);
964     }
965     else
966     {
967         Links.Unsync.Each(link, Constants.Any, handler);
968         Links.Unsync.Each(Constants.Any, link, handler);
969     }
970 }
971
972 [MethodImpl(MethodImplOptions.AggressiveInlining)]
973 public ulong CalculateTotalSymbolFrequencyCore(ulong symbol)
974 {
975     if (Options.UseSequenceMarker)
976     {
977         var counter = new TotalMarkedSequenceSymbolFrequencyOneOffCounter<ulong>(Links,
    ↳ Options.MarkedSequenceMatcher, symbol);
978         return counter.Count();
979     }
980     else
981     {
982         var counter = new TotalSequenceSymbolFrequencyOneOffCounter<ulong>(Links,
    ↳ symbol);
983         return counter.Count();
984     }
985 }
986
987 [MethodImpl(MethodImplOptions.AggressiveInlining)]
988 private bool AllUsagesCore1(ulong link, HashSet<ulong> usages, Func<IList<LinkIndex>,
    ↳ LinkIndex> outerHandler)
989 {
990     bool handler(ulong doublet)
991     {
992         if (usages.Add(doublet))
993         {
994             if (outerHandler(new LinkAddress<LinkIndex>(doublet)) != Constants.Continue)
995             {
996                 return false;
997             }
998             if (!AllUsagesCore1(doublet, usages, outerHandler))
999             {
1000                 return false;
1001             }
1002         }
1003         return true;
1004     }
1005     return Links.Unsync.Each(link, Constants.Any, handler)
1006         && Links.Unsync.Each(Constants.Any, link, handler);
1007 }
1008
1009 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1010 public void CalculateAllUsages(ulong[] totals)
1011 {
1012     var calculator = new AllUsagesCalculator(Links, totals);
1013     calculator.Calculate();
1014 }
1015
1016 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1017 public void CalculateAllUsages2(ulong[] totals)

```

```

1018 {
1019     var calculator = new AllUsagesCalculator2(Links, totals);
1020     calculator.Calculate();
1021 }
1022
1023 private class AllUsagesCalculator
1024 {
1025     private readonly SynchronizedLinks<ulong> _links;
1026     private readonly ulong[] _totals;
1027
1028     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1029     public AllUsagesCalculator(SynchronizedLinks<ulong> links, ulong[] totals)
1030     {
1031         _links = links;
1032         _totals = totals;
1033     }
1034
1035     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1036     public void Calculate() => _links.Each(_links.Constants.Any, _links.Constants.Any,
        ↪ CalculateCore);
1037
1038     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1039     private bool CalculateCore(ulong link)
1040     {
1041         if (_totals[link] == 0)
1042         {
1043             var total = 1UL;
1044             _totals[link] = total;
1045             var visitedChildren = new HashSet<ulong>();
1046             bool linkCalculator(ulong child)
1047             {
1048                 if (link != child && visitedChildren.Add(child))
1049                 {
1050                     total += _totals[child] == 0 ? 1 : _totals[child];
1051                 }
1052                 return true;
1053             }
1054             _links.Unsync.Each(link, _links.Constants.Any, linkCalculator);
1055             _links.Unsync.Each(_links.Constants.Any, link, linkCalculator);
1056             _totals[link] = total;
1057         }
1058         return true;
1059     }
1060 }
1061
1062 private class AllUsagesCalculator2
1063 {
1064     private readonly SynchronizedLinks<ulong> _links;
1065     private readonly ulong[] _totals;
1066
1067     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1068     public AllUsagesCalculator2(SynchronizedLinks<ulong> links, ulong[] totals)
1069     {
1070         _links = links;
1071         _totals = totals;
1072     }
1073
1074     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1075     public void Calculate() => _links.Each(_links.Constants.Any, _links.Constants.Any,
        ↪ CalculateCore);
1076
1077     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1078     private bool IsElement(ulong link)
1079     {
1080         // _linksInSequence.Contains(link) ||
1081         return _links.Unsync.GetTarget(link) == link || _links.Unsync.GetSource(link) ==
            ↪ link;
1082     }
1083
1084     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1085     private bool CalculateCore(ulong link)
1086     {
1087         // TODO: Проработать защиту от заикливания
1088         // Основано на SequenceWalker.WalkLeft
1089         Func<ulong, ulong> getSource = _links.Unsync.GetSource;
1090         Func<ulong, ulong> getTarget = _links.Unsync.GetTarget;
1091         Func<ulong, bool> isElement = IsElement;
1092         void visitLeaf(ulong parent)
1093     {

```

```

1094         if (link != parent)
1095         {
1096             _totals[parent]++;
1097         }
1098     }
1099     void visitNode(ulong parent)
1100     {
1101         if (link != parent)
1102         {
1103             _totals[parent]++;
1104         }
1105     }
1106     var stack = new Stack();
1107     var element = link;
1108     if (isElement(element))
1109     {
1110         visitLeaf(element);
1111     }
1112     else
1113     {
1114         while (true)
1115         {
1116             if (isElement(element))
1117             {
1118                 if (stack.Count == 0)
1119                 {
1120                     break;
1121                 }
1122                 element = stack.Pop();
1123                 var source = getSource(element);
1124                 var target = getTarget(element);
1125                 // Обработка элемента
1126                 if (isElement(target))
1127                 {
1128                     visitLeaf(target);
1129                 }
1130                 if (isElement(source))
1131                 {
1132                     visitLeaf(source);
1133                 }
1134                 element = source;
1135             }
1136             else
1137             {
1138                 stack.Push(element);
1139                 visitNode(element);
1140                 element = getTarget(element);
1141             }
1142         }
1143     }
1144     _totals[link]++;
1145     return true;
1146 }
1147 }
1148
1149 private class AllUsagesCollector
1150 {
1151     private readonly ILinks<ulong> _links;
1152     private readonly HashSet<ulong> _usages;
1153
1154     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1155     public AllUsagesCollector(ILinks<ulong> links, HashSet<ulong> usages)
1156     {
1157         _links = links;
1158         _usages = usages;
1159     }
1160
1161     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1162     public bool Collect(ulong link)
1163     {
1164         if (_usages.Add(link))
1165         {
1166             _links.Each(link, _links.Constants.Any, Collect);
1167             _links.Each(_links.Constants.Any, link, Collect);
1168         }
1169         return true;
1170     }
1171 }
1172

```

```

1173 private class AllUsagesCollector1
1174 {
1175     private readonly ILinks<ulong> _links;
1176     private readonly HashSet<ulong> _usages;
1177     private readonly ulong _continue;
1178
1179     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1180     public AllUsagesCollector1(ILinks<ulong> links, HashSet<ulong> usages)
1181     {
1182         _links = links;
1183         _usages = usages;
1184         _continue = _links.Constants.Continue;
1185     }
1186
1187     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1188     public ulong Collect(IList<ulong> link)
1189     {
1190         var linkIndex = _links.GetIndex(link);
1191         if (_usages.Add(linkIndex))
1192         {
1193             _links.Each(Collect, _links.Constants.Any, linkIndex);
1194         }
1195         return _continue;
1196     }
1197 }
1198
1199 private class AllUsagesCollector2
1200 {
1201     private readonly ILinks<ulong> _links;
1202     private readonly BitString _usages;
1203
1204     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1205     public AllUsagesCollector2(ILinks<ulong> links, BitString usages)
1206     {
1207         _links = links;
1208         _usages = usages;
1209     }
1210
1211     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1212     public bool Collect(ulong link)
1213     {
1214         if (_usages.Add((long)link))
1215         {
1216             _links.Each(link, _links.Constants.Any, Collect);
1217             _links.Each(_links.Constants.Any, link, Collect);
1218         }
1219         return true;
1220     }
1221 }
1222
1223 private class AllUsagesIntersectingCollector
1224 {
1225     private readonly SynchronizedLinks<ulong> _links;
1226     private readonly HashSet<ulong> _intersectWith;
1227     private readonly HashSet<ulong> _usages;
1228     private readonly HashSet<ulong> _enter;
1229
1230     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1231     public AllUsagesIntersectingCollector(SynchronizedLinks<ulong> links, HashSet<ulong>
↵ intersectWith, HashSet<ulong> usages)
1232     {
1233         _links = links;
1234         _intersectWith = intersectWith;
1235         _usages = usages;
1236         _enter = new HashSet<ulong>(); // защита от зацикливания
1237     }
1238
1239     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1240     public bool Collect(ulong link)
1241     {
1242         if (_enter.Add(link))
1243         {
1244             if (_intersectWith.Contains(link))
1245             {
1246                 _usages.Add(link);
1247             }
1248             _links.Unsync.Each(link, _links.Constants.Any, Collect);
1249             _links.Unsync.Each(_links.Constants.Any, link, Collect);
1250         }
1251         return true;

```



```

1252     }
1253 }
1254
1255 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1256 private void CloseInnerConnections(Action<IList<LinkIndex>> handler, ulong left, ulong
    ↪ right)
1257 {
1258     TryStepLeftUp(handler, left, right);
1259     TryStepRightUp(handler, right, left);
1260 }
1261
1262 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1263 private void AllCloseConnections(Action<IList<LinkIndex>> handler, ulong left, ulong
    ↪ right)
1264 {
1265     // Direct
1266     if (left == right)
1267     {
1268         handler(new LinkAddress<LinkIndex>(left));
1269     }
1270     var doublet = Links.Unsync.SearchOrDefault(left, right);
1271     if (doublet != Constants.Null)
1272     {
1273         handler(new LinkAddress<LinkIndex>(doublet));
1274     }
1275     // Inner
1276     CloseInnerConnections(handler, left, right);
1277     // Outer
1278     StepLeft(handler, left, right);
1279     StepRight(handler, left, right);
1280     PartialStepRight(handler, left, right);
1281     PartialStepLeft(handler, left, right);
1282 }
1283
1284 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1285 private HashSet<ulong> GetAllPartiallyMatchingSequencesCore(ulong[] sequence,
    ↪ HashSet<ulong> previousMatchings, long startAt)
1286 {
1287     if (startAt >= sequence.Length) // ?
1288     {
1289         return previousMatchings;
1290     }
1291     var secondLinkUsages = new HashSet<ulong>();
1292     AllUsagesCore(sequence[startAt], secondLinkUsages);
1293     secondLinkUsages.Add(sequence[startAt]);
1294     var matchings = new HashSet<ulong>();
1295     var filler = new SetFiller<LinkIndex, LinkIndex>(matchings, Constants.Continue);
1296     //for (var i = 0; i < previousMatchings.Count; i++)
1297     foreach (var secondLinkUsage in secondLinkUsages)
1298     {
1299         foreach (var previousMatching in previousMatchings)
1300         {
1301             //AllCloseConnections(matchings.AddAndReturnVoid, previousMatching,
1302             ↪ secondLinkUsage);
1303             StepRight(filler.AddFirstAndReturnConstant, previousMatching,
1304             ↪ secondLinkUsage);
1305             TryStepRightUp(filler.AddFirstAndReturnConstant, secondLinkUsage,
1306             ↪ previousMatching);
1307             //PartialStepRight(matchings.AddAndReturnVoid, secondLinkUsage,
1308             ↪ sequence[startAt]); // почему-то эта ошибочная запись приводит к
1309             ↪ желаемым результатам.
1310             PartialStepRight(filler.AddFirstAndReturnConstant, previousMatching,
1311             ↪ secondLinkUsage);
1312         }
1313     }
1314     if (matchings.Count == 0)
1315     {
1316         return matchings;
1317     }
1318     return GetAllPartiallyMatchingSequencesCore(sequence, matchings, startAt + 1); // ??
1319 }
1320
1321 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1322 private static void EnsureEachLinkIsAnyOrZeroOrManyOrExists(SynchronizedLinks<ulong>
    ↪ links, params ulong[] sequence)
1323 {
1324     if (sequence == null)

```

```

1319     {
1320         return;
1321     }
1322     for (var i = 0; i < sequence.Length; i++)
1323     {
1324         if (sequence[i] != links.Constants.Any && sequence[i] != ZeroOrMany &&
1325             ↪ !links.Exists(sequence[i]))
1326         {
1327             throw new ArgumentLinkDoesNotExistsException<ulong>(sequence[i],
1328                 ↪ $"patternSequence[{i}]");
1329         }
1330     }
1331 }
1332
1333 // Pattern Matching -> Key To Triggers
1334 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1335 public HashSet<ulong> MatchPattern(params ulong[] patternSequence)
1336 {
1337     return _sync.ExecuteReadOperation(() =>
1338     {
1339         patternSequence = Simplify(patternSequence);
1340         if (patternSequence.Length > 0)
1341         {
1342             EnsureEachLinkIsAnyOrZeroOrManyOrExists(Links, patternSequence);
1343             var uniqueSequenceElements = new HashSet<ulong>();
1344             for (var i = 0; i < patternSequence.Length; i++)
1345             {
1346                 if (patternSequence[i] != Constants.Any && patternSequence[i] !=
1347                     ↪ ZeroOrMany)
1348                 {
1349                     uniqueSequenceElements.Add(patternSequence[i]);
1350                 }
1351             }
1352             var results = new HashSet<ulong>();
1353             foreach (var uniqueSequenceElement in uniqueSequenceElements)
1354             {
1355                 AllUsagesCore(uniqueSequenceElement, results);
1356             }
1357             var filteredResults = new HashSet<ulong>();
1358             var matcher = new PatternMatcher(this, patternSequence, filteredResults);
1359             matcher.AddAllPatternMatchedToResults(results);
1360             return filteredResults;
1361         }
1362         return new HashSet<ulong>();
1363     });
1364 }
1365
1366 // Найти все возможные связи между указанным списком связей.
1367 // Находит связи между всеми указанными связями в любом порядке.
1368 // TODO: решить что делать с повторами (когда одни и те же элементы встречаются
1369 ↪ несколько раз в последовательности)
1370 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1371 public HashSet<ulong> GetAllConnections(params ulong[] linksToConnect)
1372 {
1373     return _sync.ExecuteReadOperation(() =>
1374     {
1375         var results = new HashSet<ulong>();
1376         if (linksToConnect.Length > 0)
1377         {
1378             Links.EnsureLinkExists(linksToConnect);
1379             AllUsagesCore(linksToConnect[0], results);
1380             for (var i = 1; i < linksToConnect.Length; i++)
1381             {
1382                 var next = new HashSet<ulong>();
1383                 AllUsagesCore(linksToConnect[i], next);
1384                 results.IntersectWith(next);
1385             }
1386             return results;
1387         }
1388     });
1389 }
1390
1391 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1392 public HashSet<ulong> GetAllConnections1(params ulong[] linksToConnect)
1393 {
1394     return _sync.ExecuteReadOperation(() =>
1395     {
1396         var results = new HashSet<ulong>();

```

```

1393     if (linksToConnect.Length > 0)
1394     {
1395         Links.EnsureLinkExists(linksToConnect);
1396         var collector1 = new AllUsagesCollector(Links.Unsync, results);
1397         collector1.Collect(linksToConnect[0]);
1398         var next = new HashSet<ulong>();
1399         for (var i = 1; i < linksToConnect.Length; i++)
1400         {
1401             var collector = new AllUsagesCollector(Links.Unsync, next);
1402             collector.Collect(linksToConnect[i]);
1403             results.IntersectWith(next);
1404             next.Clear();
1405         }
1406     }
1407     return results;
1408 });
1409 }
1410
1411 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1412 public HashSet<ulong> GetAllConnections2(params ulong[] linksToConnect)
1413 {
1414     return _sync.ExecuteReadOperation(() =>
1415     {
1416         var results = new HashSet<ulong>();
1417         if (linksToConnect.Length > 0)
1418         {
1419             Links.EnsureLinkExists(linksToConnect);
1420             var collector1 = new AllUsagesCollector(Links, results);
1421             collector1.Collect(linksToConnect[0]);
1422             //AllUsagesCore(linksToConnect[0], results);
1423             for (var i = 1; i < linksToConnect.Length; i++)
1424             {
1425                 var next = new HashSet<ulong>();
1426                 var collector = new AllUsagesIntersectingCollector(Links, results, next);
1427                 collector.Collect(linksToConnect[i]);
1428                 //AllUsagesCore(linksToConnect[i], next);
1429                 //results.IntersectWith(next);
1430                 results = next;
1431             }
1432         }
1433         return results;
1434     });
1435 }
1436
1437 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1438 public List<ulong> GetAllConnections3(params ulong[] linksToConnect)
1439 {
1440     return _sync.ExecuteReadOperation(() =>
1441     {
1442         var results = new BitString((long)Links.Unsync.Count() + 1); // new
1443         ↪ BitArray((int)_links.Total + 1);
1444         if (linksToConnect.Length > 0)
1445         {
1446             Links.EnsureLinkExists(linksToConnect);
1447             var collector1 = new AllUsagesCollector2(Links.Unsync, results);
1448             collector1.Collect(linksToConnect[0]);
1449             for (var i = 1; i < linksToConnect.Length; i++)
1450             {
1451                 var next = new BitString((long)Links.Unsync.Count() + 1); //new
1452                 ↪ BitArray((int)_links.Total + 1);
1453                 var collector = new AllUsagesCollector2(Links.Unsync, next);
1454                 collector.Collect(linksToConnect[i]);
1455                 results = results.And(next);
1456             }
1457             return results.GetSetUInt64Indices();
1458         }
1459     });
1460 }
1461
1462 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1463 private static ulong[] Simplify(ulong[] sequence)
1464 {
1465     // Считаем новый размер последовательности
1466     long newLength = 0;
1467     var zeroOrManyStepped = false;
1468     for (var i = 0; i < sequence.Length; i++)
1469     {
1470         if (sequence[i] == ZeroOrMany)

```

```

1469     {
1470         if (zeroOrManyStepped)
1471         {
1472             continue;
1473         }
1474         zeroOrManyStepped = true;
1475     }
1476     else
1477     {
1478         //if (zeroOrManyStepped) Is it efficient?
1479         zeroOrManyStepped = false;
1480     }
1481     newLength++;
1482 }
1483 // Строим новую последовательность
1484 zeroOrManyStepped = false;
1485 var newSequence = new ulong[newLength];
1486 long j = 0;
1487 for (var i = 0; i < sequence.Length; i++)
1488 {
1489     //var current = zeroOrManyStepped;
1490     //zeroOrManyStepped = patternSequence[i] == zeroOrMany;
1491     //if (current && zeroOrManyStepped)
1492     //    continue;
1493     //var newZeroOrManyStepped = patternSequence[i] == zeroOrMany;
1494     //if (zeroOrManyStepped && newZeroOrManyStepped)
1495     //    continue;
1496     //zeroOrManyStepped = newZeroOrManyStepped;
1497     if (sequence[i] == ZeroOrMany)
1498     {
1499         if (zeroOrManyStepped)
1500         {
1501             continue;
1502         }
1503         zeroOrManyStepped = true;
1504     }
1505     else
1506     {
1507         //if (zeroOrManyStepped) Is it efficient?
1508         zeroOrManyStepped = false;
1509     }
1510     newSequence[j++] = sequence[i];
1511 }
1512 return newSequence;
1513 }
1514
1515 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1516 public static void TestSimplify()
1517 {
1518     var sequence = new ulong[] { ZeroOrMany, ZeroOrMany, 2, 3, 4, ZeroOrMany,
1519     ↪ ZeroOrMany, ZeroOrMany, 4, ZeroOrMany, ZeroOrMany, ZeroOrMany };
1520     var simplifiedSequence = Simplify(sequence);
1521 }
1522
1523 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1524 public List<ulong> GetSimilarSequences() => new List<ulong>();
1525
1526 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1527 public void Prediction()
1528 {
1529     //_links
1530     //_sequences
1531 }
1532
1533 #region From Triplets
1534
1535 //public static void DeleteSequence(Link sequence)
1536 //{
1537 //}
1538
1539 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1540 public List<ulong> CollectMatchingSequences(ulong[] links)
1541 {
1542     if (links.Length == 1)
1543     {
1544         throw new Exception("Подпоследовательности с одним элементом не
1545         ↪ поддерживаются.");
1546     }
1547     var leftBound = 0;

```

```

1546     var rightBound = links.Length - 1;
1547     var left = links[leftBound++];
1548     var right = links[rightBound--];
1549     var results = new List<ulong>();
1550     CollectMatchingSequences(left, leftBound, links, right, rightBound, ref results);
1551     return results;
1552 }
1553
1554 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1555 private void CollectMatchingSequences(ulong leftLink, int leftBound, ulong[]
    ↪ middleLinks, ulong rightLink, int rightBound, ref List<ulong> results)
1556 {
1557     var leftLinkTotalReferers = Links.Unsync.Count(leftLink);
1558     var rightLinkTotalReferers = Links.Unsync.Count(rightLink);
1559     if (leftLinkTotalReferers <= rightLinkTotalReferers)
1560     {
1561         var nextLeftLink = middleLinks[leftBound];
1562         var elements = GetRightElements(leftLink, nextLeftLink);
1563         if (leftBound <= rightBound)
1564         {
1565             for (var i = elements.Length - 1; i >= 0; i--)
1566             {
1567                 var element = elements[i];
1568                 if (element != 0)
1569                 {
1570                     CollectMatchingSequences(element, leftBound + 1, middleLinks,
1571                         ↪ rightLink, rightBound, ref results);
1572                 }
1573             }
1574         }
1575         else
1576         {
1577             for (var i = elements.Length - 1; i >= 0; i--)
1578             {
1579                 var element = elements[i];
1580                 if (element != 0)
1581                 {
1582                     results.Add(element);
1583                 }
1584             }
1585         }
1586     }
1587     else
1588     {
1589         var nextRightLink = middleLinks[rightBound];
1590         var elements = GetLeftElements(rightLink, nextRightLink);
1591         if (leftBound <= rightBound)
1592         {
1593             for (var i = elements.Length - 1; i >= 0; i--)
1594             {
1595                 var element = elements[i];
1596                 if (element != 0)
1597                 {
1598                     CollectMatchingSequences(leftLink, leftBound, middleLinks,
1599                         ↪ elements[i], rightBound - 1, ref results);
1600                 }
1601             }
1602         }
1603         else
1604         {
1605             for (var i = elements.Length - 1; i >= 0; i--)
1606             {
1607                 var element = elements[i];
1608                 if (element != 0)
1609                 {
1610                     results.Add(element);
1611                 }
1612             }
1613         }
1614     }
1615 }
1616
1617 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1618 public ulong[] GetRightElements(ulong startLink, ulong rightLink)
1619 {
1620     var result = new ulong[5];
1621     TryStepRight(startLink, rightLink, result, 0);
1622     Links.Each(Constants.Any, startLink, couple =>

```

```

1621     {
1622         if (couple != startLink)
1623         {
1624             if (TryStepRight(couple, rightLink, result, 2))
1625             {
1626                 return false;
1627             }
1628         }
1629         return true;
1630     });
1631     if (Links.GetTarget(Links.GetTarget(startLink)) == rightLink)
1632     {
1633         result[4] = startLink;
1634     }
1635     return result;
1636 }
1637
1638 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1639 public bool TryStepRight(ulong startLink, ulong rightLink, ulong[] result, int offset)
1640 {
1641     var added = 0;
1642     Links.Each(startLink, Constants.Any, couple =>
1643     {
1644         if (couple != startLink)
1645         {
1646             var coupleTarget = Links.GetTarget(couple);
1647             if (coupleTarget == rightLink)
1648             {
1649                 result[offset] = couple;
1650                 if (++added == 2)
1651                 {
1652                     return false;
1653                 }
1654             }
1655             else if (Links.GetSource(coupleTarget) == rightLink) // coupleTarget.Linker
1656                 ↪ == Net.And &&
1657             {
1658                 result[offset + 1] = couple;
1659                 if (++added == 2)
1660                 {
1661                     return false;
1662                 }
1663             }
1664             return true;
1665         });
1666     return added > 0;
1667 }
1668
1669 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1670 public ulong[] GetLeftElements(ulong startLink, ulong leftLink)
1671 {
1672     var result = new ulong[5];
1673     TryStepLeft(startLink, leftLink, result, 0);
1674     Links.Each(startLink, Constants.Any, couple =>
1675     {
1676         if (couple != startLink)
1677         {
1678             if (TryStepLeft(couple, leftLink, result, 2))
1679             {
1680                 return false;
1681             }
1682         }
1683         return true;
1684     });
1685     if (Links.GetSource(Links.GetSource(leftLink)) == startLink)
1686     {
1687         result[4] = leftLink;
1688     }
1689     return result;
1690 }
1691
1692 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1693 public bool TryStepLeft(ulong startLink, ulong leftLink, ulong[] result, int offset)
1694 {
1695     var added = 0;
1696     Links.Each(Constants.Any, startLink, couple =>
1697     {
1698         if (couple != startLink)

```

```

1699     {
1700         var coupleSource = Links.GetSource(couple);
1701         if (coupleSource == leftLink)
1702         {
1703             result[offset] = couple;
1704             if (++added == 2)
1705             {
1706                 return false;
1707             }
1708         }
1709         else if (Links.GetTarget(coupleSource) == leftLink) // coupleSource.Linker
1710             ↪ == Net.And &&
1711         {
1712             result[offset + 1] = couple;
1713             if (++added == 2)
1714             {
1715                 return false;
1716             }
1717         }
1718         return true;
1719     });
1720     return added > 0;
1721 }
1722
1723 #endregion
1724
1725 #region Walkers
1726
1727 public class PatternMatcher : RightSequenceWalker<ulong>
1728 {
1729     private readonly Sequences _sequences;
1730     private readonly ulong[] _patternSequence;
1731     private readonly HashSet<LinkIndex> _linksInSequence;
1732     private readonly HashSet<LinkIndex> _results;
1733
1734     #region Pattern Match
1735
1736     enum PatternBlockType
1737     {
1738         Undefined,
1739         Gap,
1740         Elements
1741     }
1742
1743     struct PatternBlock
1744     {
1745         public PatternBlockType Type;
1746         public long Start;
1747         public long Stop;
1748     }
1749
1750     private readonly List<PatternBlock> _pattern;
1751     private int _patternPosition;
1752     private long _sequencePosition;
1753
1754     #endregion
1755
1756     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1757     public PatternMatcher(Sequences sequences, LinkIndex[] patternSequence,
1758         ↪ HashSet<LinkIndex> results)
1759         : base(sequences.Links.Unsync, new DefaultStack<ulong>())
1760     {
1761         _sequences = sequences;
1762         _patternSequence = patternSequence;
1763         _linksInSequence = new HashSet<LinkIndex>(patternSequence.Where(x => x !=
1764             ↪ _sequences.Constants.Any && x != ZeroOrMany));
1765         _results = results;
1766         _pattern = CreateDetailedPattern();
1767     }
1768
1769     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1770     protected override bool IsElement(ulong link) => _linksInSequence.Contains(link) ||
1771         ↪ base.IsElement(link);
1772
1773     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1774     public bool PatternMatch(LinkIndex sequenceToMatch)
1775     {
1776         _patternPosition = 0;
1777         _sequencePosition = 0;
1778         foreach (var part in Walk(sequenceToMatch))

```

```

1776     {
1777         if (!PatternMatchCore(part))
1778         {
1779             break;
1780         }
1781     }
1782     return _patternPosition == _pattern.Count || (_patternPosition == _pattern.Count
        ↪ - 1 && _pattern[_patternPosition].Start == 0);
1783 }
1784
1785 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1786 private List<PatternBlock> CreateDetailedPattern()
1787 {
1788     var pattern = new List<PatternBlock>();
1789     var patternBlock = new PatternBlock();
1790     for (var i = 0; i < _patternSequence.Length; i++)
1791     {
1792         if (patternBlock.Type == PatternBlockType.Undefined)
1793         {
1794             if (_patternSequence[i] == _sequences.Constants.Any)
1795             {
1796                 patternBlock.Type = PatternBlockType.Gap;
1797                 patternBlock.Start = 1;
1798                 patternBlock.Stop = 1;
1799             }
1800             else if (_patternSequence[i] == ZeroOrMany)
1801             {
1802                 patternBlock.Type = PatternBlockType.Gap;
1803                 patternBlock.Start = 0;
1804                 patternBlock.Stop = long.MaxValue;
1805             }
1806             else
1807             {
1808                 patternBlock.Type = PatternBlockType.Elements;
1809                 patternBlock.Start = i;
1810                 patternBlock.Stop = i;
1811             }
1812         }
1813         else if (patternBlock.Type == PatternBlockType.Elements)
1814         {
1815             if (_patternSequence[i] == _sequences.Constants.Any)
1816             {
1817                 pattern.Add(patternBlock);
1818                 patternBlock = new PatternBlock
1819                 {
1820                     Type = PatternBlockType.Gap,
1821                     Start = 1,
1822                     Stop = 1
1823                 };
1824             }
1825             else if (_patternSequence[i] == ZeroOrMany)
1826             {
1827                 pattern.Add(patternBlock);
1828                 patternBlock = new PatternBlock
1829                 {
1830                     Type = PatternBlockType.Gap,
1831                     Start = 0,
1832                     Stop = long.MaxValue
1833                 };
1834             }
1835             else
1836             {
1837                 patternBlock.Stop = i;
1838             }
1839         }
1840         else // patternBlock.Type == PatternBlockType.Gap
1841         {
1842             if (_patternSequence[i] == _sequences.Constants.Any)
1843             {
1844                 patternBlock.Start++;
1845                 if (patternBlock.Stop < patternBlock.Start)
1846                 {
1847                     patternBlock.Stop = patternBlock.Start;
1848                 }
1849             }
1850             else if (_patternSequence[i] == ZeroOrMany)
1851             {
1852                 patternBlock.Stop = long.MaxValue;
1853             }
1854             else

```



```

1855         {
1856             pattern.Add(patternBlock);
1857             patternBlock = new PatternBlock
1858             {
1859                 Type = PatternBlockType.Elements,
1860                 Start = i,
1861                 Stop = i
1862             };
1863         }
1864     }
1865 }
1866 if (patternBlock.Type != PatternBlockType.Undefined)
1867 {
1868     pattern.Add(patternBlock);
1869 }
1870 return pattern;
1871 }
1872
1873 // match: search for regexp anywhere in text
1874 //int match(char* regexp, char* text)
1875 //{
1876 //    do
1877 //    {
1878 //        } while (*text++ != '\0');
1879 //    return 0;
1880 //}
1881
1882 // matchhere: search for regexp at beginning of text
1883 //int matchhere(char* regexp, char* text)
1884 //{
1885 //    if (regexp[0] == '\0')
1886 //        return 1;
1887 //    if (regexp[1] == '*')
1888 //        return matchstar(regexp[0], regexp + 2, text);
1889 //    if (regexp[0] == '$' && regexp[1] == '\0')
1890 //        return *text == '\0';
1891 //    if (*text != '\0' && (regexp[0] == '.' || regexp[0] == *text))
1892 //        return matchhere(regexp + 1, text + 1);
1893 //    return 0;
1894 //}
1895
1896 // matchstar: search for c*regexp at beginning of text
1897 //int matchstar(int c, char* regexp, char* text)
1898 //{
1899 //    do
1900 //    {
1901 //        /* a * matches zero or more instances */
1902 //        if (matchhere(regexp, text))
1903 //            return 1;
1904 //    } while (*text != '\0' && (*text++ == c || c == '.'));
1905 //    return 0;
1906 //}
1907
1908 //private void GetNextPatternElement(out LinkIndex element, out long mininumGap, out
1909 ↪ long maximumGap)
1910 //{
1911 //    mininumGap = 0;
1912 //    maximumGap = 0;
1913 //    element = 0;
1914 //    for (; _patternPosition < _patternSequence.Length; _patternPosition++)
1915 //    {
1916 //        if (_patternSequence[_patternPosition] == Doublets.Links.Null)
1917 //            mininumGap++;
1918 //        else if (_patternSequence[_patternPosition] == ZeroOrMany)
1919 //            maximumGap = long.MaxValue;
1920 //        else
1921 //            break;
1922 //    }
1923
1924 //    if (maximumGap < mininumGap)
1925 //        maximumGap = mininumGap;
1926 //}
1927
1928 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1929 private bool PatternMatchCore(LinkIndex element)
1930 {
1931     if (_patternPosition >= _pattern.Count)
1932     {
1933         _patternPosition = -2;
1934         return false;
1935     }

```

```

1933 }
1934 var currentPatternBlock = _pattern[_patternPosition];
1935 if (currentPatternBlock.Type == PatternBlockType.Gap)
1936 {
1937     //var currentMatchingBlockLength = (_sequencePosition -
1938     ↪ _lastMatchedBlockPosition);
1939     if (_sequencePosition < currentPatternBlock.Start)
1940     {
1941         _sequencePosition++;
1942         return true; // Двигаемся дальше
1943     }
1944     // Это последний блок
1945     if (_pattern.Count == _patternPosition + 1)
1946     {
1947         _patternPosition++;
1948         _sequencePosition = 0;
1949         return false; // Полное соответствие
1950     }
1951     else
1952     {
1953         if (_sequencePosition > currentPatternBlock.Stop)
1954         {
1955             return false; // Соответствие невозможно
1956         }
1957         var nextPatternBlock = _pattern[_patternPosition + 1];
1958         if (_patternSequence[nextPatternBlock.Start] == element)
1959         {
1960             if (nextPatternBlock.Start < nextPatternBlock.Stop)
1961             {
1962                 _patternPosition++;
1963                 _sequencePosition = 1;
1964             }
1965             else
1966             {
1967                 _patternPosition += 2;
1968                 _sequencePosition = 0;
1969             }
1970         }
1971     }
1972 }
1973 else // currentPatternBlock.Type == PatternBlockType.Elements
1974 {
1975     var patternElementPosition = currentPatternBlock.Start + _sequencePosition;
1976     if (_patternSequence[patternElementPosition] != element)
1977     {
1978         return false; // Соответствие невозможно
1979     }
1980     if (patternElementPosition == currentPatternBlock.Stop)
1981     {
1982         _patternPosition++;
1983         _sequencePosition = 0;
1984     }
1985     else
1986     {
1987         _sequencePosition++;
1988     }
1989 }
1990 return true;
1991 //if (_patternSequence[_patternPosition] != element)
1992 //    return false;
1993 //else
1994 //    {
1995 //        _sequencePosition++;
1996 //        _patternPosition++;
1997 //        return true;
1998 //    }
1999 //if (_filterPosition == _patternSequence.Length)
2000 //    {
2001 //        _filterPosition = -2; // Длиннее чем нужно
2002 //        return false;
2003 //    }
2004 //if (element != _patternSequence[_filterPosition])
2005 //    {
2006 //        _filterPosition = -1;
2007 //        return false; // Начинается иначе
2008 //    }
2009 //if (_filterPosition == (_patternSequence.Length - 1))
2010

```

```

2011         //      return false;
2012         //if (_filterPosition >= 0)
2013         //{
2014         //      if (element == _patternSequence[_filterPosition + 1])
2015         //          _filterPosition++;
2016         //      else
2017         //          return false;
2018         //}
2019         //if (_filterPosition < 0)
2020         //{
2021         //      if (element == _patternSequence[0])
2022         //          _filterPosition = 0;
2023         //}
2024     }
2025
2026     [MethodImpl(MethodImplOptions.AggressiveInlining)]
2027     public void AddAllPatternMatchedToResults(IEnumerable<ulong> sequencesToMatch)
2028     {
2029         foreach (var sequenceToMatch in sequencesToMatch)
2030         {
2031             if (PatternMatch(sequenceToMatch))
2032             {
2033                 _results.Add(sequenceToMatch);
2034             }
2035         }
2036     }
2037 }
2038
2039 #endregion
2040 }
2041 }

```

1.82 ./Platform.Data.Doublets/Sequences/Sequences.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Runtime.CompilerServices;
5  using Platform.Collections;
6  using Platform.Collections.Lists;
7  using Platform.Collections.Stacks;
8  using Platform.Threading.Synchronization;
9  using Platform.Data.Doublets.Sequences.Walkers;
10 using LinkIndex = System.UInt64;
11
12 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
13
14 namespace Platform.Data.Doublets.Sequences
15 {
16     /// <summary>
17     /// Представляет коллекцию последовательностей связей.
18     /// </summary>
19     /// <remarks>
20     /// Обязательно реализовать атомарность каждого публичного метода.
21     ///
22     /// TODO:
23     ///
24     /// !!! Повышение вероятности повторного использования групп (подпоследовательностей),
25     /// через естественную группировку по unicode типам, все whitespace вместе, все символы
26     /// ↪ вместе, все числа вместе и т.п.
27     /// + использовать ровно сбалансированный вариант, чтобы уменьшать вложенность (глубину
28     /// ↪ графа)
29     ///
30     /// х*у - найти все связи между, в последовательностях любой формы, если не стоит
31     /// ↪ ограничитель на то, что является последовательностью, а что нет,
32     /// то находятся любые структуры связей, которые содержат эти элементы именно в таком
33     /// ↪ порядке.
34     ///
35     /// Рост последовательности слева и справа.
36     /// Поиск со звездочкой.
37     /// URL, PURL - реестр используемых во вне ссылок на ресурсы,
38     /// так же проблема может быть решена при реализации дистанционных триггеров.
39     /// Нужны ли уникальные указатели вообще?
40     /// Что если обращение к информации будет происходить через содержимое всегда?
41     ///
42     /// Писать тесты.
43     ///
44     ///
45     /// Можно убрать зависимость от конкретной реализации Links,
46     /// на зависимость от абстрактного элемента, который может быть представлен несколькими
47     /// ↪ способами.

```

```

43  ///
44  /// Можно ли как-то сделать один общий интерфейс
45  ///
46  ///
47  /// Блокчейн и/или гит для распределённой записи транзакций.
48  ///
49  /// </remarks>
50  public partial class Sequences : ILinks<LinkIndex> // IList<string>, IList<LinkIndex[]>
    ↪ (после завершения реализации Sequences)
51  {
52      /// <summary>Возвращает значение LinkIndex, обозначающее любое количество
    ↪ связей.</summary>
53      public const LinkIndex ZeroOrMany = LinkIndex.MaxValue;
54
55      public SequencesOptions<LinkIndex> Options { get; }
56      public SynchronizedLinks<LinkIndex> Links { get; }
57      private readonly ISynchronization _sync;
58
59      public LinksConstants<LinkIndex> Constants { get; }
60
61      [MethodImpl(MethodImplOptions.AggressiveInlining)]
62      public Sequences(SynchronizedLinks<LinkIndex> links, SequencesOptions<LinkIndex> options)
63      {
64          Links = links;
65          _sync = links.SyncRoot;
66          Options = options;
67          Options.ValidateOptions();
68          Options.InitOptions(Links);
69          Constants = links.Constants;
70      }
71
72      [MethodImpl(MethodImplOptions.AggressiveInlining)]
73      public Sequences(SynchronizedLinks<LinkIndex> links) : this(links, new
    ↪ SequencesOptions<LinkIndex>()) { }
74
75      [MethodImpl(MethodImplOptions.AggressiveInlining)]
76      public bool IsSequence(LinkIndex sequence)
77      {
78          return _sync.ExecuteReadOperation(() =>
79          {
80              if (Options.UseSequenceMarker)
81              {
82                  return Options.MarkedSequenceMatcher.IsMatched(sequence);
83              }
84              return !Links.Unsync.IsPartialPoint(sequence);
85          });
86      }
87
88      [MethodImpl(MethodImplOptions.AggressiveInlining)]
89      private LinkIndex GetSequenceByElements(LinkIndex sequence)
90      {
91          if (Options.UseSequenceMarker)
92          {
93              return Links.SearchOrDefault(Options.SequenceMarkerLink, sequence);
94          }
95          return sequence;
96      }
97
98      [MethodImpl(MethodImplOptions.AggressiveInlining)]
99      private LinkIndex GetSequenceElements(LinkIndex sequence)
100     {
101         if (Options.UseSequenceMarker)
102         {
103             var linkContents = new Link<ulong>(Links.GetLink(sequence));
104             if (linkContents.Source == Options.SequenceMarkerLink)
105             {
106                 return linkContents.Target;
107             }
108             if (linkContents.Target == Options.SequenceMarkerLink)
109             {
110                 return linkContents.Source;
111             }
112         }
113         return sequence;
114     }
115
116     #region Count
117
118     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

119 public LinkIndex Count(ICollection<LinkIndex> restrictions)
120 {
121     if (restrictions.IsNullOrEmpty())
122     {
123         return Links.Count(Constants.Any, Options.SequenceMarkerLink, Constants.Any);
124     }
125     if (restrictions.Count == 1) // Первая связь это адрес
126     {
127         var sequenceIndex = restrictions[0];
128         if (sequenceIndex == Constants.Null)
129         {
130             return 0;
131         }
132         if (sequenceIndex == Constants.Any)
133         {
134             return Count(null);
135         }
136         if (Options.UseSequenceMarker)
137         {
138             return Links.Count(Constants.Any, Options.SequenceMarkerLink, sequenceIndex);
139         }
140         return Links.Exists(sequenceIndex) ? 1UL : 0;
141     }
142     throw new NotImplementedException();
143 }
144
145 [MethodImpl(MethodImplOptions.AggressiveInlining)]
146 private LinkIndex CountUsages(params LinkIndex[] restrictions)
147 {
148     if (restrictions.Length == 0)
149     {
150         return 0;
151     }
152     if (restrictions.Length == 1) // Первая связь это адрес
153     {
154         if (restrictions[0] == Constants.Null)
155         {
156             return 0;
157         }
158         var any = Constants.Any;
159         if (Options.UseSequenceMarker)
160         {
161             var elementsLink = GetSequenceElements(restrictions[0]);
162             var sequenceLink = GetSequenceByElements(elementsLink);
163             if (sequenceLink != Constants.Null)
164             {
165                 return Links.Count(any, sequenceLink) + Links.Count(any, elementsLink) -
166                     ↪ 1;
167             }
168             return Links.Count(any, elementsLink);
169         }
170         return Links.Count(any, restrictions[0]);
171     }
172     throw new NotImplementedException();
173 }
174 #endregion
175
176 #region Create
177
178 [MethodImpl(MethodImplOptions.AggressiveInlining)]
179 public LinkIndex Create(ICollection<LinkIndex> restrictions)
180 {
181     return _sync.ExecuteWriteOperation(() =>
182     {
183         if (restrictions.IsNullOrEmpty())
184         {
185             return Constants.Null;
186         }
187         Links.EnsureInnerReferenceExists(restrictions, nameof(restrictions));
188         return CreateCore(restrictions);
189     });
190 }
191
192 [MethodImpl(MethodImplOptions.AggressiveInlining)]
193 private LinkIndex CreateCore(ICollection<LinkIndex> restrictions)
194 {
195     LinkIndex[] sequence = restrictions.SkipFirst();
196     if (Options.UseIndex)

```

```

197     {
198         Options.Index.Add(sequence);
199     }
200     var sequenceRoot = default(LinkIndex);
201     if (Options.EnforceSingleSequenceVersionOnWriteBasedOnExisting)
202     {
203         var matches = Each(restrictions);
204         if (matches.Count > 0)
205         {
206             sequenceRoot = matches[0];
207         }
208     }
209     else if (Options.EnforceSingleSequenceVersionOnWriteBasedOnNew)
210     {
211         return CompactCore(sequence);
212     }
213     if (sequenceRoot == default)
214     {
215         sequenceRoot = Options.LinksToSequenceConverter.Convert(sequence);
216     }
217     if (Options.UseSequenceMarker)
218     {
219         return Links.Unsync.GetOrCreate(Options.SequenceMarkerLink, sequenceRoot);
220     }
221     return sequenceRoot; // Возвращаем корень последовательности (т.е. сами элементы)
222 }
223
224 #endregion
225
226 #region Each
227
228 [MethodImpl(MethodImplOptions.AggressiveInlining)]
229 public List<LinkIndex> Each(IList<LinkIndex> sequence)
230 {
231     var results = new List<LinkIndex>();
232     var filler = new ListFiller<LinkIndex, LinkIndex>(results, Constants.Continue);
233     Each(filler.AddFirstAndReturnConstant, sequence);
234     return results;
235 }
236
237 [MethodImpl(MethodImplOptions.AggressiveInlining)]
238 public LinkIndex Each(Func<IList<LinkIndex>, LinkIndex> handler, IList<LinkIndex>
    ↪ restrictions)
239 {
240     return _sync.ExecuteReadOperation(() =>
241     {
242         if (restrictions.IsNullOrEmpty())
243         {
244             return Constants.Continue;
245         }
246         Links.EnsureInnerReferenceExists(restrictions, nameof(restrictions));
247         if (restrictions.Count == 1)
248         {
249             var link = restrictions[0];
250             var any = Constants.Any;
251             if (link == any)
252             {
253                 if (Options.UseSequenceMarker)
254                 {
255                     return Links.Unsync.Each(handler, new Link<LinkIndex>(any,
256                         ↪ Options.SequenceMarkerLink, any));
257                 }
258                 else
259                 {
260                     return Links.Unsync.Each(handler, new Link<LinkIndex>(any, any,
261                         ↪ any));
262                 }
263             }
264             if (Options.UseSequenceMarker)
265             {
266                 var sequenceLinkValues = Links.Unsync.GetLink(link);
267                 if (sequenceLinkValues[Constants.SourcePart] ==
268                     ↪ Options.SequenceMarkerLink)
269                 {
270                     link = sequenceLinkValues[Constants.TargetPart];
271                 }
272             }
273             var sequence = Options.Walker.Walk(link).ToArray().ShiftRight();

```

```

271         sequence[0] = link;
272         return handler(sequence);
273     }
274     else if (restrictions.Count == 2)
275     {
276         throw new NotImplementedException();
277     }
278     else if (restrictions.Count == 3)
279     {
280         return Links.Unsync.Each(handler, restrictions);
281     }
282     else
283     {
284         var sequence = restrictions.SkipFirst();
285         if (Options.UseIndex && !Options.Index.MightContain(sequence))
286         {
287             return Constants.Break;
288         }
289         return EachCore(handler, sequence);
290     }
291     });
292 }
293
294 [MethodImpl(MethodImplOptions.AggressiveInlining)]
295 private LinkIndex EachCore(Func<IList<LinkIndex>, LinkIndex> handler, IList<LinkIndex>
    ↪ values)
296 {
297     var matcher = new Matcher(this, values, new HashSet<LinkIndex>(), handler);
298     // TODO: Find out why matcher.HandleFullMatched executed twice for the same sequence
    ↪ Id.
299     Func<IList<LinkIndex>, LinkIndex> innerHandler = Options.UseSequenceMarker ?
    ↪ (Func<IList<LinkIndex>, LinkIndex>)matcher.HandleFullMatchedSequence :
    ↪ matcher.HandleFullMatched;
300     //if (sequence.Length >= 2)
301     if (StepRight(innerHandler, values[0], values[1]) != Constants.Continue)
302     {
303         return Constants.Break;
304     }
305     var last = values.Count - 2;
306     for (var i = 1; i < last; i++)
307     {
308         if (PartialStepRight(innerHandler, values[i], values[i + 1]) !=
    ↪ Constants.Continue)
309         {
310             return Constants.Break;
311         }
312     }
313     if (values.Count >= 3)
314     {
315         if (StepLeft(innerHandler, values[values.Count - 2], values[values.Count - 1])
    ↪ != Constants.Continue)
316         {
317             return Constants.Break;
318         }
319     }
320     return Constants.Continue;
321 }
322
323 [MethodImpl(MethodImplOptions.AggressiveInlining)]
324 private LinkIndex PartialStepRight(Func<IList<LinkIndex>, LinkIndex> handler, LinkIndex
    ↪ left, LinkIndex right)
325 {
326     return Links.Unsync.Each(doublet =>
327     {
328         var doubletIndex = doublet[Constants.IndexPart];
329         if (StepRight(handler, doubletIndex, right) != Constants.Continue)
330         {
331             return Constants.Break;
332         }
333         if (left != doubletIndex)
334         {
335             return PartialStepRight(handler, doubletIndex, right);
336         }
337         return Constants.Continue;
338     }, new Link<LinkIndex>(Constants.Any, Constants.Any, left));
339 }
340
341 [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

342 private LinkIndex StepRight(Func<IList<LinkIndex>, LinkIndex> handler, LinkIndex left,
    ↳ LinkIndex right) => Links.Unsync.Each(rightStep => TryStepRightUp(handler, right,
    ↳ rightStep[Constants.IndexPart]), new Link<LinkIndex>(Constants.Any, left,
    ↳ Constants.Any));
343
344 [MethodImpl(MethodImplOptions.AggressiveInlining)]
345 private LinkIndex TryStepRightUp(Func<IList<LinkIndex>, LinkIndex> handler, LinkIndex
    ↳ right, LinkIndex stepFrom)
346 {
347     var upStep = stepFrom;
348     var firstSource = Links.Unsync.GetTarget(upStep);
349     while (firstSource != right && firstSource != upStep)
350     {
351         upStep = firstSource;
352         firstSource = Links.Unsync.GetSource(upStep);
353     }
354     if (firstSource == right)
355     {
356         return handler(new LinkAddress<LinkIndex>(stepFrom));
357     }
358     return Constants.Continue;
359 }
360
361 [MethodImpl(MethodImplOptions.AggressiveInlining)]
362 private LinkIndex StepLeft(Func<IList<LinkIndex>, LinkIndex> handler, LinkIndex left,
    ↳ LinkIndex right) => Links.Unsync.Each(leftStep => TryStepLeftUp(handler, left,
    ↳ leftStep[Constants.IndexPart]), new Link<LinkIndex>(Constants.Any, Constants.Any,
    ↳ right));
363
364 [MethodImpl(MethodImplOptions.AggressiveInlining)]
365 private LinkIndex TryStepLeftUp(Func<IList<LinkIndex>, LinkIndex> handler, LinkIndex
    ↳ left, LinkIndex stepFrom)
366 {
367     var upStep = stepFrom;
368     var firstTarget = Links.Unsync.GetSource(upStep);
369     while (firstTarget != left && firstTarget != upStep)
370     {
371         upStep = firstTarget;
372         firstTarget = Links.Unsync.GetTarget(upStep);
373     }
374     if (firstTarget == left)
375     {
376         return handler(new LinkAddress<LinkIndex>(stepFrom));
377     }
378     return Constants.Continue;
379 }
380
381 #endregion
382
383 #region Update
384
385 [MethodImpl(MethodImplOptions.AggressiveInlining)]
386 public LinkIndex Update(IList<LinkIndex> restrictions, IList<LinkIndex> substitution)
387 {
388     var sequence = restrictions.SkipFirst();
389     var newSequence = substitution.SkipFirst();
390     if (sequence.IsNullOrEmpty() && newSequence.IsNullOrEmpty())
391     {
392         return Constants.Null;
393     }
394     if (sequence.IsNullOrEmpty())
395     {
396         return Create(substitution);
397     }
398     if (newSequence.IsNullOrEmpty())
399     {
400         Delete(restrictions);
401         return Constants.Null;
402     }
403     return _sync.ExecuteWriteOperation((Func<ulong>)((() =>
404     {
405         ILinksExtensions.EnsureLinkIsAnyOrExists<ulong>(Links, (IList<ulong>)sequence);
406         Links.EnsureLinkExists(newSequence);
407         return UpdateCore(sequence, newSequence);
408     })));
409 }
410
411 [MethodImpl(MethodImplOptions.AggressiveInlining)]

```



```

412 private LinkIndex UpdateCore(IList<LinkIndex> sequence, IList<LinkIndex> newSequence)
413 {
414     LinkIndex bestVariant;
415     if (Options.EnforceSingleSequenceVersionOnWriteBasedOnNew &&
416         ↪ !sequence.EqualTo(newSequence))
417     {
418         bestVariant = CompactCore(newSequence);
419     }
420     else
421     {
422         bestVariant = CreateCore(newSequence);
423     }
424     // TODO: Check all options only ones before loop execution
425     // Возможно нужно две версии Each, возвращающий фактические последовательности и с
426     ↪ маркером,
427     // или возможно даже возвращать и тот и тот вариант. С другой стороны все варианты
428     ↪ можно получить имея только фактические последовательности.
429     foreach (var variant in Each(sequence))
430     {
431         if (variant != bestVariant)
432         {
433             UpdateOneCore(variant, bestVariant);
434         }
435     }
436     return bestVariant;
437 }
438
439 [MethodImpl(MethodImplOptions.AggressiveInlining)]
440 private void UpdateOneCore(LinkIndex sequence, LinkIndex newSequence)
441 {
442     if (Options.UseGarbageCollection)
443     {
444         var sequenceElements = GetSequenceElements(sequence);
445         var sequenceElementsContents = new Link<ulong>(Links.GetLink(sequenceElements));
446         var sequenceLink = GetSequenceByElements(sequenceElements);
447         var newSequenceElements = GetSequenceElements(newSequence);
448         var newSequenceLink = GetSequenceByElements(newSequenceElements);
449         if (Options.UseCascadeUpdate || CountUsages(sequence) == 0)
450         {
451             if (sequenceLink != Constants.Null)
452             {
453                 Links.Unsync.MergeAndDelete(sequenceLink, newSequenceLink);
454             }
455             Links.Unsync.MergeAndDelete(sequenceElements, newSequenceElements);
456         }
457         ClearGarbage(sequenceElementsContents.Source);
458         ClearGarbage(sequenceElementsContents.Target);
459     }
460     else
461     {
462         if (Options.UseSequenceMarker)
463         {
464             var sequenceElements = GetSequenceElements(sequence);
465             var sequenceLink = GetSequenceByElements(sequenceElements);
466             var newSequenceElements = GetSequenceElements(newSequence);
467             var newSequenceLink = GetSequenceByElements(newSequenceElements);
468             if (Options.UseCascadeUpdate || CountUsages(sequence) == 0)
469             {
470                 if (sequenceLink != Constants.Null)
471                 {
472                     Links.Unsync.MergeAndDelete(sequenceLink, newSequenceLink);
473                 }
474                 Links.Unsync.MergeAndDelete(sequenceElements, newSequenceElements);
475             }
476         }
477         else
478         {
479             if (Options.UseCascadeUpdate || CountUsages(sequence) == 0)
480             {
481                 Links.Unsync.MergeAndDelete(sequence, newSequence);
482             }
483         }
484     }
485 }
486
487 #endregion
488
489 #region Delete

```

```

487 [MethodImpl(MethodImplOptions.AggressiveInlining)]
488 public void Delete(ICollection<LinkIndex> restrictions)
489 {
490     _sync.ExecuteWriteOperation(() =>
491     {
492         var sequence = restrictions.SkipFirst();
493         // TODO: Check all options only ones before loop execution
494         foreach (var linkToDelete in Each(sequence))
495         {
496             DeleteOneCore(linkToDelete);
497         }
498     });
499 }
500
501 [MethodImpl(MethodImplOptions.AggressiveInlining)]
502 private void DeleteOneCore(LinkIndex link)
503 {
504     if (Options.UseGarbageCollection)
505     {
506         var sequenceElements = GetSequenceElements(link);
507         var sequenceElementsContents = new Link<ulong>(Links.GetLink(sequenceElements));
508         var sequenceLink = GetSequenceByElements(sequenceElements);
509         if (Options.UseCascadeDelete || CountUsages(link) == 0)
510         {
511             if (sequenceLink != Constants.Null)
512             {
513                 Links.Unsync.Delete(sequenceLink);
514             }
515             Links.Unsync.Delete(link);
516         }
517         ClearGarbage(sequenceElementsContents.Source);
518         ClearGarbage(sequenceElementsContents.Target);
519     }
520     else
521     {
522         if (Options.UseSequenceMarker)
523         {
524             var sequenceElements = GetSequenceElements(link);
525             var sequenceLink = GetSequenceByElements(sequenceElements);
526             if (Options.UseCascadeDelete || CountUsages(link) == 0)
527             {
528                 if (sequenceLink != Constants.Null)
529                 {
530                     Links.Unsync.Delete(sequenceLink);
531                 }
532                 Links.Unsync.Delete(link);
533             }
534         }
535         else
536         {
537             if (Options.UseCascadeDelete || CountUsages(link) == 0)
538             {
539                 Links.Unsync.Delete(link);
540             }
541         }
542     }
543 }
544
545 #endregion
546
547 #region Compactification
548
549 [MethodImpl(MethodImplOptions.AggressiveInlining)]
550 public void CompactAll()
551 {
552     _sync.ExecuteWriteOperation(() =>
553     {
554         var sequences = Each((LinkAddress<LinkIndex>)Constants.Any);
555         for (int i = 0; i < sequences.Count; i++)
556         {
557             var sequence = this.ToList(sequences[i]);
558             Compact(sequence.ShiftRight());
559         }
560     });
561 }
562
563 /// <remarks>
564

```

```

565 /// bestVariant можно выбирать по максимальному числу использований,
566 /// но сбалансированный позволяет гарантировать уникальность (если есть возможность,
567 /// гарантировать его использование в других местах).
568 ///
569 /// Получается этот метод должен игнорировать Options.EnforceSingleSequenceVersionOnWrite
570 /// </remarks>
571 [MethodImpl(MethodImplOptions.AggressiveInlining)]
572 public LinkIndex Compact(IList<LinkIndex> sequence)
573 {
574     return _sync.ExecuteWriteOperation(() =>
575     {
576         if (sequence.IsNullOrEmpty())
577         {
578             return Constants.Null;
579         }
580         Links.EnsureInnerReferenceExists(sequence, nameof(sequence));
581         return CompactCore(sequence);
582     });
583 }
584
585 [MethodImpl(MethodImplOptions.AggressiveInlining)]
586 private LinkIndex CompactCore(IList<LinkIndex> sequence) => UpdateCore(sequence,
    ↪ sequence);
587
588 #endregion
589
590 #region Garbage Collection
591
592 /// <remarks>
593 /// TODO: Добавить дополнительный обработчик / событие CanBeDeleted которое можно
    ↪ определить извне или в унаследованном классе
594 /// </remarks>
595 [MethodImpl(MethodImplOptions.AggressiveInlining)]
596 private bool IsGarbage(LinkIndex link) => link != Options.SequenceMarkerLink &&
    ↪ !Links.Unsync.IsPartialPoint(link) && Links.Count(Constants.Any, link) == 0;
597
598 [MethodImpl(MethodImplOptions.AggressiveInlining)]
599 private void ClearGarbage(LinkIndex link)
600 {
601     if (IsGarbage(link))
602     {
603         var contents = new Link<ulong>(Links.GetLink(link));
604         Links.Unsync.Delete(link);
605         ClearGarbage(contents.Source);
606         ClearGarbage(contents.Target);
607     }
608 }
609
610 #endregion
611
612 #region Walkers
613
614 [MethodImpl(MethodImplOptions.AggressiveInlining)]
615 public bool EachPart(Func<LinkIndex, bool> handler, LinkIndex sequence)
616 {
617     return _sync.ExecuteReadOperation(() =>
618     {
619         var links = Links.Unsync;
620         foreach (var part in Options.Walker.Walk(sequence))
621         {
622             if (!handler(part))
623             {
624                 return false;
625             }
626         }
627         return true;
628     });
629 }
630
631 public class Matcher : RightSequenceWalker<LinkIndex>
632 {
633     private readonly Sequences _sequences;
634     private readonly IList<LinkIndex> _patternSequence;
635     private readonly HashSet<LinkIndex> _linksInSequence;
636     private readonly HashSet<LinkIndex> _results;
637     private readonly Func<IList<LinkIndex>, LinkIndex> _stopableHandler;
638     private readonly HashSet<LinkIndex> _readAsElements;
639     private int _filterPosition;
640
641     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

642 public Matcher(Sequences sequences, IList<LinkIndex> patternSequence,
    ↳ HashSet<LinkIndex> results, Func<IList<LinkIndex>, LinkIndex> stopableHandler,
    ↳ HashSet<LinkIndex> readAsElements = null)
643 : base(sequences.Links.Unsync, new DefaultStack<LinkIndex>())
644 {
645     _sequences = sequences;
646     _patternSequence = patternSequence;
647     _linksInSequence = new HashSet<LinkIndex>(patternSequence.Where(x => x !=
    ↳ Links.Constants.Any && x != ZeroOrMany));
648     _results = results;
649     _stopableHandler = stopableHandler;
650     _readAsElements = readAsElements;
651 }
652
653 [MethodImpl(MethodImplOptions.AggressiveInlining)]
654 protected override bool IsElement(LinkIndex link) => base.IsElement(link) ||
    ↳ (_readAsElements != null && _readAsElements.Contains(link)) ||
    ↳ _linksInSequence.Contains(link);
655
656 [MethodImpl(MethodImplOptions.AggressiveInlining)]
657 public bool FullMatch(LinkIndex sequenceToMatch)
658 {
659     _filterPosition = 0;
660     foreach (var part in Walk(sequenceToMatch))
661     {
662         if (!FullMatchCore(part))
663         {
664             break;
665         }
666     }
667     return _filterPosition == _patternSequence.Count;
668 }
669
670 [MethodImpl(MethodImplOptions.AggressiveInlining)]
671 private bool FullMatchCore(LinkIndex element)
672 {
673     if (_filterPosition == _patternSequence.Count)
674     {
675         _filterPosition = -2; // Длиннее чем нужно
676         return false;
677     }
678     if (_patternSequence[_filterPosition] != Links.Constants.Any
        && element != _patternSequence[_filterPosition])
679     {
680         _filterPosition = -1;
681         return false; // Начинается/Продолжается иначе
682     }
683     _filterPosition++;
684     return true;
685 }
686
687 [MethodImpl(MethodImplOptions.AggressiveInlining)]
688 public void AddFullMatchedToResults(IList<LinkIndex> restrictions)
689 {
690     var sequenceToMatch = restrictions[Links.Constants.IndexPart];
691     if (FullMatch(sequenceToMatch))
692     {
693         _results.Add(sequenceToMatch);
694     }
695 }
696
697 [MethodImpl(MethodImplOptions.AggressiveInlining)]
698 public LinkIndex HandleFullMatched(IList<LinkIndex> restrictions)
699 {
700     var sequenceToMatch = restrictions[Links.Constants.IndexPart];
701     if (FullMatch(sequenceToMatch) && _results.Add(sequenceToMatch))
702     {
703         return _stopableHandler(new LinkAddress<LinkIndex>(sequenceToMatch));
704     }
705     return Links.Constants.Continue;
706 }
707
708 [MethodImpl(MethodImplOptions.AggressiveInlining)]
709 public LinkIndex HandleFullMatchedSequence(IList<LinkIndex> restrictions)
710 {
711     var sequenceToMatch = restrictions[Links.Constants.IndexPart];
712     var sequence = _sequences.GetSequenceByElements(sequenceToMatch);
713     if (sequence != Links.Constants.Null && FullMatch(sequenceToMatch) &&
    ↳ _results.Add(sequenceToMatch))
714

```

```

715     {
716         return _stopableHandler(new LinkAddress<LinkIndex>(sequence));
717     }
718     return Links.Constants.Continue;
719 }
720
721 /// <remarks>
722 /// TODO: Add support for LinksConstants.Any
723 /// </remarks>
724 [MethodImpl(MethodImplOptions.AggressiveInlining)]
725 public bool PartialMatch(LinkIndex sequenceToMatch)
726 {
727     _filterPosition = -1;
728     foreach (var part in Walk(sequenceToMatch))
729     {
730         if (!PartialMatchCore(part))
731         {
732             break;
733         }
734     }
735     return _filterPosition == _patternSequence.Count - 1;
736 }
737
738 [MethodImpl(MethodImplOptions.AggressiveInlining)]
739 private bool PartialMatchCore(LinkIndex element)
740 {
741     if (_filterPosition == (_patternSequence.Count - 1))
742     {
743         return false; // Нашлось
744     }
745     if (_filterPosition >= 0)
746     {
747         if (element == _patternSequence[_filterPosition + 1])
748         {
749             _filterPosition++;
750         }
751         else
752         {
753             _filterPosition = -1;
754         }
755     }
756     if (_filterPosition < 0)
757     {
758         if (element == _patternSequence[0])
759         {
760             _filterPosition = 0;
761         }
762     }
763     return true; // Ищем дальше
764 }
765
766 [MethodImpl(MethodImplOptions.AggressiveInlining)]
767 public void AddPartialMatchedToResults(LinkIndex sequenceToMatch)
768 {
769     if (PartialMatch(sequenceToMatch))
770     {
771         _results.Add(sequenceToMatch);
772     }
773 }
774
775 [MethodImpl(MethodImplOptions.AggressiveInlining)]
776 public LinkIndex HandlePartialMatched(ICollection<LinkIndex> restrictions)
777 {
778     var sequenceToMatch = restrictions[Links.Constants.IndexPart];
779     if (PartialMatch(sequenceToMatch))
780     {
781         return _stopableHandler(new LinkAddress<LinkIndex>(sequenceToMatch));
782     }
783     return Links.Constants.Continue;
784 }
785
786 [MethodImpl(MethodImplOptions.AggressiveInlining)]
787 public void AddAllPartialMatchedToResults(IEnumerable<LinkIndex> sequencesToMatch)
788 {
789     foreach (var sequenceToMatch in sequencesToMatch)
790     {
791         if (PartialMatch(sequenceToMatch))
792         {
793             _results.Add(sequenceToMatch);
794         }
795     }
796 }

```

```

794     }
795     }
796 }
797
798 [MethodImpl(MethodImplOptions.AggressiveInlining)]
799 public void AddAllPartialMatchedToResultsAndReadAsElements(IEnumerable<LinkIndex>
    ↪ sequencesToMatch)
800 {
801     foreach (var sequenceToMatch in sequencesToMatch)
802     {
803         if (PartialMatch(sequenceToMatch))
804         {
805             _readAsElements.Add(sequenceToMatch);
806             _results.Add(sequenceToMatch);
807         }
808     }
809 }
810 }
811
812 #endregion
813 }
814 }

```

1.83 ./Platform.Data.Doublets/Sequences/SequencesExtensions.cs

```

1  using System.Collections.Generic;
2  using System.Runtime.CompilerServices;
3  using Platform.Collections.Lists;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.Sequences
8  {
9      public static class SequencesExtensions
10     {
11         [MethodImpl(MethodImplOptions.AggressiveInlining)]
12         public static TLink Create<TLink>(this ILinks<TLink> sequences, IList<TLink[]>
            ↪ groupedSequence)
13         {
14             var finalSequence = new TLink[groupedSequence.Count];
15             for (var i = 0; i < finalSequence.Length; i++)
16             {
17                 var part = groupedSequence[i];
18                 finalSequence[i] = part.Length == 1 ? part[0] :
                    ↪ sequences.Create(part.ShiftRight());
19             }
20             return sequences.Create(finalSequence.ShiftRight());
21         }
22
23         [MethodImpl(MethodImplOptions.AggressiveInlining)]
24         public static IList<TLink> ToList<TLink>(this ILinks<TLink> sequences, TLink sequence)
25         {
26             var list = new List<TLink>();
27             var filler = new ListFiller<TLink, TLink>(list, sequences.Constants.Break);
28             sequences.Each(filler.AddSkipFirstAndReturnConstant, new
                ↪ LinkAddress<TLink>(sequence));
29             return list;
30         }
31     }
32 }

```

1.84 ./Platform.Data.Doublets/Sequences/SequencesOptions.cs

```

1  using System;
2  using System.Collections.Generic;
3  using Platform.Interfaces;
4  using Platform.Collections.Stacks;
5  using Platform.Converters;
6  using Platform.Data.Doublets.Sequences.Frequencies.Cache;
7  using Platform.Data.Doublets.Sequences.Frequencies.Counters;
8  using Platform.Data.Doublets.Sequences.Converters;
9  using Platform.Data.Doublets.Sequences.Walkers;
10 using Platform.Data.Doublets.Sequences.Indexes;
11 using Platform.Data.Doublets.Sequences.CriterionMatchers;
12 using System.Runtime.CompilerServices;
13
14 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
15
16 namespace Platform.Data.Doublets.Sequences
17 {
18     public class SequencesOptions<TLink> // TODO: To use type parameter <TLink> the
        ↪ ILinks<TLink> must contain GetConstants function.

```

```

19 {
20     private static readonly EqualityComparer<TLink> _equalityComparer =
        ↳ EqualityComparer<TLink>.Default;
21
22     public TLink SequenceMarkerLink
23     {
24         [MethodImpl(MethodImplOptions.AggressiveInlining)]
25         get;
26         [MethodImpl(MethodImplOptions.AggressiveInlining)]
27         set;
28     }
29
30     public bool UseCascadeUpdate
31     {
32         [MethodImpl(MethodImplOptions.AggressiveInlining)]
33         get;
34         [MethodImpl(MethodImplOptions.AggressiveInlining)]
35         set;
36     }
37
38     public bool UseCascadeDelete
39     {
40         [MethodImpl(MethodImplOptions.AggressiveInlining)]
41         get;
42         [MethodImpl(MethodImplOptions.AggressiveInlining)]
43         set;
44     }
45
46     public bool UseIndex
47     {
48         [MethodImpl(MethodImplOptions.AggressiveInlining)]
49         get;
50         [MethodImpl(MethodImplOptions.AggressiveInlining)]
51         set;
52     } // TODO: Update Index on sequence update/delete.
53
54     public bool UseSequenceMarker
55     {
56         [MethodImpl(MethodImplOptions.AggressiveInlining)]
57         get;
58         [MethodImpl(MethodImplOptions.AggressiveInlining)]
59         set;
60     }
61
62     public bool UseCompression
63     {
64         [MethodImpl(MethodImplOptions.AggressiveInlining)]
65         get;
66         [MethodImpl(MethodImplOptions.AggressiveInlining)]
67         set;
68     }
69
70     public bool UseGarbageCollection
71     {
72         [MethodImpl(MethodImplOptions.AggressiveInlining)]
73         get;
74         [MethodImpl(MethodImplOptions.AggressiveInlining)]
75         set;
76     }
77
78     public bool EnforceSingleSequenceVersionOnWriteBasedOnExisting
79     {
80         [MethodImpl(MethodImplOptions.AggressiveInlining)]
81         get;
82         [MethodImpl(MethodImplOptions.AggressiveInlining)]
83         set;
84     }
85
86     public bool EnforceSingleSequenceVersionOnWriteBasedOnNew
87     {
88         [MethodImpl(MethodImplOptions.AggressiveInlining)]
89         get;
90         [MethodImpl(MethodImplOptions.AggressiveInlining)]
91         set;
92     }
93
94     public MarkedSequenceCriterionMatcher<TLink> MarkedSequenceMatcher
95     {
96         [MethodImpl(MethodImplOptions.AggressiveInlining)]
97         get;
98         [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

    set;
}

public IConverter<IList<TLink>, TLink> LinksToSequenceConverter
{
    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    get;
    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    set;
}

public ISequenceIndex<TLink> Index
{
    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    get;
    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    set;
}

public ISequenceWalker<TLink> Walker
{
    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    get;
    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    set;
}

public bool ReadFullSequence
{
    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    get;
    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    set;
}

// TODO: Реализовать компактификацию при чтении
//public bool EnforceSingleSequenceVersionOnRead { get; set; }
//public bool UseRequestMarker { get; set; }
//public bool StoreRequestResults { get; set; }

[MethodImpl(MethodImplOptions.AggressiveInlining)]
public void InitOptions(ISynchronizedLinks<TLink> links)
{
    if (UseSequenceMarker)
    {
        if (_equalityComparer.Equals(SequenceMarkerLink, links.Constants.Null))
        {
            SequenceMarkerLink = links.CreatePoint();
        }
        else
        {
            if (!links.Exists(SequenceMarkerLink))
            {
                var link = links.CreatePoint();
                if (!_equalityComparer.Equals(link, SequenceMarkerLink))
                {
                    throw new InvalidOperationException("Cannot recreate sequence marker
↪ link.");
                }
            }
        }
        if (MarkedSequenceMatcher == null)
        {
            MarkedSequenceMatcher = new MarkedSequenceCriterionMatcher<TLink>(links,
↪ SequenceMarkerLink);
        }
    }
    var balancedVariantConverter = new BalancedVariantConverter<TLink>(links);
    if (UseCompression)
    {
        if (LinksToSequenceConverter == null)
        {
            ICounter<TLink, TLink> totalSequenceSymbolFrequencyCounter;
            if (UseSequenceMarker)
            {
                totalSequenceSymbolFrequencyCounter = new
↪ TotalMarkedSequenceSymbolFrequencyCounter<TLink>(links,
↪ MarkedSequenceMatcher);
            }
        }
    }
}

```



```

174         else
175         {
176             totalSequenceSymbolFrequencyCounter = new
177                 ↪ TotalSequenceSymbolFrequencyCounter<TLink>(links);
178         }
179         var doubletFrequenciesCache = new LinkFrequenciesCache<TLink>(links,
180             ↪ totalSequenceSymbolFrequencyCounter);
181         var compressingConverter = new CompressingConverter<TLink>(links,
182             ↪ balancedVariantConverter, doubletFrequenciesCache);
183         LinksToSequenceConverter = compressingConverter;
184     }
185 }
186 else
187 {
188     if (LinksToSequenceConverter == null)
189     {
190         LinksToSequenceConverter = balancedVariantConverter;
191     }
192 }
193 if (UseIndex && Index == null)
194 {
195     Index = new SequenceIndex<TLink>(links);
196 }
197 if (Walker == null)
198 {
199     Walker = new RightSequenceWalker<TLink>(links, new DefaultStack<TLink>());
200 }
201 }
202 }
203 }
204 }
205 }
206 }
207 }
208 }
209 }

```

1.85 ./Platform.Data.Doublets/Sequences/Walkers/ISequenceWalker.cs

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Sequences.Walkers
7 {
8     public interface ISequenceWalker<TLink>
9     {
10         [MethodImpl(MethodImplOptions.AggressiveInlining)]
11         IEnumerable<TLink> Walk(TLink sequence);
12     }
13 }

```

1.86 ./Platform.Data.Doublets/Sequences/Walkers/LeftSequenceWalker.cs

```

1 using System;
2 using System.Collections.Generic;
3 using System.Runtime.CompilerServices;
4 using Platform.Collections.Stacks;
5
6 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8 namespace Platform.Data.Doublets.Sequences.Walkers
9 {
10     public class LeftSequenceWalker<TLink> : SequenceWalkerBase<TLink>
11     {
12         [MethodImpl(MethodImplOptions.AggressiveInlining)]
13         public LeftSequenceWalker(ILinks<TLink> links, IStack<TLink> stack, Func<TLink, bool>
14             ↪ isElement) : base(links, stack, isElement) { }
15
16         [MethodImpl(MethodImplOptions.AggressiveInlining)]
17         public LeftSequenceWalker(ILinks<TLink> links, IStack<TLink> stack) : base(links, stack,
18             ↪ links.IsPartialPoint) { }
19
20         [MethodImpl(MethodImplOptions.AggressiveInlining)]
21         protected override TLink GetNextElementAfterPop(TLink element) =>
22             ↪ Links.GetSource(element);
23     }
24 }

```

```

20
21     [MethodImpl(MethodImplOptions.AggressiveInlining)]
22     protected override TLink GetNextElementAfterPush(TLink element) =>
        ↳ Links.GetTarget(element);
23
24     [MethodImpl(MethodImplOptions.AggressiveInlining)]
25     protected override IEnumerable<TLink> WalkContents(TLink element)
26     {
27         var parts = Links.GetLink(element);
28         var start = Links.Constants.IndexPart + 1;
29         for (var i = parts.Count - 1; i >= start; i--)
30         {
31             var part = parts[i];
32             if (IsElement(part))
33             {
34                 yield return part;
35             }
36         }
37     }
38 }
39 }

```

1.87 ./Platform.Data.Doublets/Sequences/Walkers/LeveledSequenceWalker.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  //#define USEARRAYPOOL
8  #if USEARRAYPOOL
9  using Platform.Collections;
10 #endif
11
12 namespace Platform.Data.Doublets.Sequences.Walkers
13 {
14     public class LeveledSequenceWalker<TLink> : LinksOperatorBase<TLink>, ISequenceWalker<TLink>
15     {
16         private static readonly EqualityComparer<TLink> _equalityComparer =
            ↳ EqualityComparer<TLink>.Default;
17
18         private readonly Func<TLink, bool> _isElement;
19
20         [MethodImpl(MethodImplOptions.AggressiveInlining)]
21         public LeveledSequenceWalker(ILinks<TLink> links, Func<TLink, bool> isElement) :
            ↳ base(links) => _isElement = isElement;
22
23         [MethodImpl(MethodImplOptions.AggressiveInlining)]
24         public LeveledSequenceWalker(ILinks<TLink> links) : base(links) => _isElement =
            ↳ Links.IsPartialPoint;
25
26         [MethodImpl(MethodImplOptions.AggressiveInlining)]
27         public IEnumerable<TLink> Walk(TLink sequence) => ToArray(sequence);
28
29         [MethodImpl(MethodImplOptions.AggressiveInlining)]
30         public TLink[] ToArray(TLink sequence)
31         {
32             var length = 1;
33             var array = new TLink[length];
34             array[0] = sequence;
35             if (_isElement(sequence))
36             {
37                 return array;
38             }
39             bool hasElements;
40             do
41             {
42                 length *= 2;
43 #if USEARRAYPOOL
44                 var nextArray = ArrayPool.Allocate<ulong>(length);
45 #else
46                 var nextArray = new TLink[length];
47 #endif
48                 hasElements = false;
49                 for (var i = 0; i < array.Length; i++)
50                 {
51                     var candidate = array[i];
52                     if (_equalityComparer.Equals(array[i], default))
53                     {
54                         continue;

```

```

55     }
56     var doubletOffset = i * 2;
57     if (!_isElement(candidate))
58     {
59         nextArray[doubletOffset] = candidate;
60     }
61     else
62     {
63         var link = Links.GetLink(candidate);
64         var linkSource = Links.GetSource(link);
65         var linkTarget = Links.GetTarget(link);
66         nextArray[doubletOffset] = linkSource;
67         nextArray[doubletOffset + 1] = linkTarget;
68         if (!hasElements)
69         {
70             hasElements = !(_isElement(linkSource) && _isElement(linkTarget));
71         }
72     }
73 }
74 #if USEARRAYPOOL
75     if (array.Length > 1)
76     {
77         ArrayPool.Free(array);
78     }
79 #endif
80     array = nextArray;
81 }
82 while (hasElements);
83 var filledElementsCount = CountFilledElements(array);
84 if (filledElementsCount == array.Length)
85 {
86     return array;
87 }
88 else
89 {
90     return CopyFilledElements(array, filledElementsCount);
91 }
92 }
93
94 [MethodImpl(MethodImplOptions.AggressiveInlining)]
95 private static TLink[] CopyFilledElements(TLink[] array, int filledElementsCount)
96 {
97     var finalArray = new TLink[filledElementsCount];
98     for (int i = 0, j = 0; i < array.Length; i++)
99     {
100         if (!_equalityComparer.Equals(array[i], default))
101         {
102             finalArray[j] = array[i];
103             j++;
104         }
105     }
106     #if USEARRAYPOOL
107         ArrayPool.Free(array);
108     #endif
109     return finalArray;
110 }
111
112 [MethodImpl(MethodImplOptions.AggressiveInlining)]
113 private static int CountFilledElements(TLink[] array)
114 {
115     var count = 0;
116     for (var i = 0; i < array.Length; i++)
117     {
118         if (!_equalityComparer.Equals(array[i], default))
119         {
120             count++;
121         }
122     }
123     return count;
124 }
125 }
126 }

```

1.88 ./Platform.Data.Doublers/Sequences/Walkers/RightSequenceWalker.cs

```

1 using System;
2 using System.Collections.Generic;
3 using System.Runtime.CompilerServices;
4 using Platform.Collections.Stacks;
5

```

```

6 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8 namespace Platform.Data.Doublets.Sequences.Walkers
9 {
10     public class RightSequenceWalker<TLink> : SequenceWalkerBase<TLink>
11     {
12         [MethodImpl(MethodImplOptions.AggressiveInlining)]
13         public RightSequenceWalker(ILinks<TLink> links, IStack<TLink> stack, Func<TLink, bool>
14             ↪ isElement) : base(links, stack, isElement) { }
15
16         [MethodImpl(MethodImplOptions.AggressiveInlining)]
17         public RightSequenceWalker(ILinks<TLink> links, IStack<TLink> stack) : base(links,
18             ↪ stack, links.IsPartialPoint) { }
19
20         [MethodImpl(MethodImplOptions.AggressiveInlining)]
21         protected override TLink GetNextElementAfterPop(TLink element) =>
22             ↪ Links.GetTarget(element);
23
24         [MethodImpl(MethodImplOptions.AggressiveInlining)]
25         protected override TLink GetNextElementAfterPush(TLink element) =>
26             ↪ Links.GetSource(element);
27
28         [MethodImpl(MethodImplOptions.AggressiveInlining)]
29         protected override IEnumerable<TLink> WalkContents(TLink element)
30         {
31             var parts = Links.GetLink(element);
32             for (var i = Links.Constants.IndexPart + 1; i < parts.Count; i++)
33             {
34                 var part = parts[i];
35                 if (IsElement(part))
36                 {
37                     yield return part;
38                 }
39             }
40         }
41     }
42 }

```

1.89 ./Platform.Data.Doublets/Sequences/Walkers/SequenceWalkerBase.cs

```

1 using System;
2 using System.Collections.Generic;
3 using System.Runtime.CompilerServices;
4 using Platform.Collections.Stacks;
5
6 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8 namespace Platform.Data.Doublets.Sequences.Walkers
9 {
10     public abstract class SequenceWalkerBase<TLink> : LinksOperatorBase<TLink>,
11         ↪ ISequenceWalker<TLink>
12     {
13         private readonly IStack<TLink> _stack;
14         private readonly Func<TLink, bool> _isElement;
15
16         [MethodImpl(MethodImplOptions.AggressiveInlining)]
17         protected SequenceWalkerBase(ILinks<TLink> links, IStack<TLink> stack, Func<TLink, bool>
18             ↪ isElement) : base(links)
19         {
20             _stack = stack;
21             _isElement = isElement;
22         }
23
24         [MethodImpl(MethodImplOptions.AggressiveInlining)]
25         protected SequenceWalkerBase(ILinks<TLink> links, IStack<TLink> stack) : this(links,
26             ↪ stack, links.IsPartialPoint) { }
27
28         [MethodImpl(MethodImplOptions.AggressiveInlining)]
29         public IEnumerable<TLink> Walk(TLink sequence)
30         {
31             _stack.Clear();
32             var element = sequence;
33             if (IsElement(element))
34             {
35                 yield return element;
36             }
37             else
38             {
39                 while (true)
40                 {

```

```

38         if (IsElement(element))
39         {
40             if (_stack.IsEmpty)
41             {
42                 break;
43             }
44             element = _stack.Pop();
45             foreach (var output in WalkContents(element))
46             {
47                 yield return output;
48             }
49             element = GetNextElementAfterPop(element);
50         }
51     else
52     {
53         _stack.Push(element);
54         element = GetNextElementAfterPush(element);
55     }
56 }
57 }
58 }
59
60 [MethodImpl(MethodImplOptions.AggressiveInlining)]
61 protected virtual bool IsElement(TLink elementLink) => _isElement(elementLink);
62
63 [MethodImpl(MethodImplOptions.AggressiveInlining)]
64 protected abstract TLink GetNextElementAfterPop(TLink element);
65
66 [MethodImpl(MethodImplOptions.AggressiveInlining)]
67 protected abstract TLink GetNextElementAfterPush(TLink element);
68
69 [MethodImpl(MethodImplOptions.AggressiveInlining)]
70 protected abstract IEnumerable<TLink> WalkContents(TLink element);
71 }
72 }

```

1.90 ./Platform.Data.Doublets/Stacks/Stack.cs

```

1  using System.Collections.Generic;
2  using System.Runtime.CompilerServices;
3  using Platform.Collections.Stacks;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.Stacks
8  {
9      public class Stack<TLink> : IStack<TLink>
10     {
11         private static readonly EqualityComparer<TLink> _equalityComparer =
12             ↳ EqualityComparer<TLink>.Default;
13
14         private readonly ILinks<TLink> _links;
15         private readonly TLink _stack;
16
17         public bool IsEmpty
18         {
19             [MethodImpl(MethodImplOptions.AggressiveInlining)]
20             get => _equalityComparer.Equals(Peek(), _stack);
21         }
22
23         [MethodImpl(MethodImplOptions.AggressiveInlining)]
24         public Stack(ILinks<TLink> links, TLink stack)
25         {
26             _links = links;
27             _stack = stack;
28         }
29
30         [MethodImpl(MethodImplOptions.AggressiveInlining)]
31         private TLink GetStackMarker() => _links.GetSource(_stack);
32
33         [MethodImpl(MethodImplOptions.AggressiveInlining)]
34         private TLink GetTop() => _links.GetTarget(_stack);
35
36         [MethodImpl(MethodImplOptions.AggressiveInlining)]
37         public TLink Peek() => _links.GetTarget(GetTop());
38
39         [MethodImpl(MethodImplOptions.AggressiveInlining)]
40         public TLink Pop()
41         {
42             var element = Peek();
43             if (!_equalityComparer.Equals(element, _stack))

```

```

43     {
44         var top = GetTop();
45         var previousTop = _links.GetSource(top);
46         _links.Update(_stack, GetStackMarker(), previousTop);
47         _links.Delete(top);
48     }
49     return element;
50 }
51
52 [MethodImpl(MethodImplOptions.AggressiveInlining)]
53 public void Push(TLink element) => _links.Update(_stack, GetStackMarker(),
    ↪ _links.GetOrCreate(GetTop(), element));
54 }
55 }

```

1.91 ./Platform.Data.Doublets/Stacks/StackExtensions.cs

```

1 using System.Runtime.CompilerServices;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Doublets.Stacks
6 {
7     public static class StackExtensions
8     {
9         [MethodImpl(MethodImplOptions.AggressiveInlining)]
10        public static TLink CreateStack<TLink>(this ILinks<TLink> links, TLink stackMarker)
11        {
12            var stackPoint = links.CreatePoint();
13            var stack = links.Update(stackPoint, stackMarker, stackPoint);
14            return stack;
15        }
16    }
17 }

```

1.92 ./Platform.Data.Doublets/SynchronizedLinks.cs

```

1 using System;
2 using System.Collections.Generic;
3 using System.Runtime.CompilerServices;
4 using Platform.Data.Doublets;
5 using Platform.Threading.Synchronization;
6
7 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9 namespace Platform.Data.Doublets
10 {
11     /// <remarks>
12     /// TODO: Autogeneration of synchronized wrapper (decorator).
13     /// TODO: Try to unfold code of each method using IL generation for performance improvements.
14     /// TODO: Or even to unfold multiple layers of implementations.
15     /// </remarks>
16     public class SynchronizedLinks<TLinkAddress> : ISynchronizedLinks<TLinkAddress>
17     {
18         public LinksConstants<TLinkAddress> Constants
19         {
20             [MethodImpl(MethodImplOptions.AggressiveInlining)]
21             get;
22         }
23
24         public ISynchronization SyncRoot
25         {
26             [MethodImpl(MethodImplOptions.AggressiveInlining)]
27             get;
28         }
29
30         public ILinks<TLinkAddress> Sync
31         {
32             [MethodImpl(MethodImplOptions.AggressiveInlining)]
33             get;
34         }
35
36         public ILinks<TLinkAddress> Unsync
37         {
38             [MethodImpl(MethodImplOptions.AggressiveInlining)]
39             get;
40         }
41
42         [MethodImpl(MethodImplOptions.AggressiveInlining)]
43         public SynchronizedLinks(ILinks<TLinkAddress> links) : this(new
            ↪ ReaderWriterLockSynchronization(), links) { }
44

```

```

45     [MethodImpl(MethodImplOptions.AggressiveInlining)]
46     public SynchronizedLinks(ISynchronization synchronization, ILinks<TLinkAddress> links)
47     {
48         SyncRoot = synchronization;
49         Sync = this;
50         Unsync = links;
51         Constants = links.Constants;
52     }
53
54     [MethodImpl(MethodImplOptions.AggressiveInlining)]
55     public TLinkAddress Count(IList<TLinkAddress> restriction) =>
56         ↳ SyncRoot.ExecuteReadOperation(restriction, Unsync.Count);
57
58     [MethodImpl(MethodImplOptions.AggressiveInlining)]
59     public TLinkAddress Each(Func<IList<TLinkAddress>, TLinkAddress> handler,
60         ↳ IList<TLinkAddress> restrictions) => SyncRoot.ExecuteReadOperation(handler,
61         ↳ restrictions, (handler1, restrictions1) => Unsync.Each(handler1, restrictions1));
62
63     [MethodImpl(MethodImplOptions.AggressiveInlining)]
64     public TLinkAddress Create(IList<TLinkAddress> restrictions) =>
65         ↳ SyncRoot.ExecuteWriteOperation(restrictions, Unsync.Create);
66
67     [MethodImpl(MethodImplOptions.AggressiveInlining)]
68     public TLinkAddress Update(IList<TLinkAddress> restrictions, IList<TLinkAddress>
69         ↳ substitution) => SyncRoot.ExecuteWriteOperation(restrictions, substitution,
70         ↳ Unsync.Update);
71
72     [MethodImpl(MethodImplOptions.AggressiveInlining)]
73     public void Delete(IList<TLinkAddress> restrictions) =>
74         ↳ SyncRoot.ExecuteWriteOperation(restrictions, Unsync.Delete);
75
76     //public T Trigger(IList<T> restriction, Func<IList<T>, IList<T>, T> matchedHandler,
77     //    ↳ IList<T> substitution, Func<IList<T>, IList<T>, T> substitutedHandler)
78     //{
79     //    if (restriction != null && substitution != null &&
80     //        ↳ !substitution.EqualTo(restriction))
81     //        return SyncRoot.ExecuteWriteOperation(restriction, matchedHandler,
82     //        ↳ substitution, substitutedHandler, Unsync.Trigger);
83     //    return SyncRoot.ExecuteReadOperation(restriction, matchedHandler, substitution,
84     //        ↳ substitutedHandler, Unsync.Trigger);
85     //}
86 }

```

1.93 ./Platform.Data.Doublets/UInt64LinksExtensions.cs

```

1  using System;
2  using System.Text;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5  using Platform.Singletons;
6  using Platform.Data.Doublets.Unicode;
7
8  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
9
10 namespace Platform.Data.Doublets
11 {
12     public static class UInt64LinksExtensions
13     {
14         public static readonly LinksConstants<ulong> Constants =
15             ↳ Default<LinksConstants<ulong>>.Instance;
16
17         [MethodImpl(MethodImplOptions.AggressiveInlining)]
18         public static void UseUnicode(this ILinks<ulong> links) => UnicodeMap.InitNew(links);
19
20         [MethodImpl(MethodImplOptions.AggressiveInlining)]
21         public static bool AnyLinkIsAny(this ILinks<ulong> links, params ulong[] sequence)
22         {
23             if (sequence == null)
24             {
25                 return false;
26             }
27             var constants = links.Constants;
28             for (var i = 0; i < sequence.Length; i++)
29             {
30                 if (sequence[i] == constants.Any)
31                 {
32                     return true;
33                 }
34             }
35         }
36     }
37 }

```

```

33     }
34     return false;
35 }
36
37 [MethodImpl(MethodImplOptions.AggressiveInlining)]
38 public static string FormatStructure(this ILinks<ulong> links, ulong linkIndex,
    ↪ Func<Link<ulong>, bool> isElement, bool renderIndex = false, bool renderDebug =
    ↪ false)
39 {
40     var sb = new StringBuilder();
41     var visited = new HashSet<ulong>();
42     links.AppendStructure(sb, visited, linkIndex, isElement, (innerSb, link) =>
    ↪ innerSb.Append(link.Index), renderIndex, renderDebug);
43     return sb.ToString();
44 }
45
46 [MethodImpl(MethodImplOptions.AggressiveInlining)]
47 public static string FormatStructure(this ILinks<ulong> links, ulong linkIndex,
    ↪ Func<Link<ulong>, bool> isElement, Action<StringBuilder, Link<ulong>> appendElement,
    ↪ bool renderIndex = false, bool renderDebug = false)
48 {
49     var sb = new StringBuilder();
50     var visited = new HashSet<ulong>();
51     links.AppendStructure(sb, visited, linkIndex, isElement, appendElement, renderIndex,
    ↪ renderDebug);
52     return sb.ToString();
53 }
54
55 [MethodImpl(MethodImplOptions.AggressiveInlining)]
56 public static void AppendStructure(this ILinks<ulong> links, StringBuilder sb,
    ↪ HashSet<ulong> visited, ulong linkIndex, Func<Link<ulong>, bool> isElement,
    ↪ Action<StringBuilder, Link<ulong>> appendElement, bool renderIndex = false, bool
    ↪ renderDebug = false)
57 {
58     if (sb == null)
59     {
60         throw new ArgumentNullException(nameof(sb));
61     }
62     if (linkIndex == Constants.Null || linkIndex == Constants.Any || linkIndex ==
    ↪ Constants.Itself)
63     {
64         return;
65     }
66     if (links.Exists(linkIndex))
67     {
68         if (visited.Add(linkIndex))
69         {
70             sb.Append('(');
71             var link = new Link<ulong>(links.GetLink(linkIndex));
72             if (renderIndex)
73             {
74                 sb.Append(link.Index);
75                 sb.Append(':');
76             }
77             if (link.Source == link.Index)
78             {
79                 sb.Append(link.Index);
80             }
81             else
82             {
83                 var source = new Link<ulong>(links.GetLink(link.Source));
84                 if (isElement(source))
85                 {
86                     appendElement(sb, source);
87                 }
88                 else
89                 {
90                     links.AppendStructure(sb, visited, source.Index, isElement,
    ↪ appendElement, renderIndex);
91                 }
92             }
93             sb.Append(' ');
94             if (link.Target == link.Index)
95             {
96                 sb.Append(link.Index);
97             }
98             else
99             {

```



```

100         var target = new Link<ulong>(links.GetLink(link.Target));
101         if (isElement(target))
102         {
103             appendElement(sb, target);
104         }
105         else
106         {
107             links.AppendStructure(sb, visited, target.Index, isElement,
108                 ↪ appendElement, renderIndex);
109         }
110         sb.Append(' ');
111     }
112     else
113     {
114         if (renderDebug)
115         {
116             sb.Append('*');
117         }
118         sb.Append(linkIndex);
119     }
120 }
121 else
122 {
123     if (renderDebug)
124     {
125         sb.Append('~');
126     }
127     sb.Append(linkIndex);
128 }
129 }
130 }
131 }

```

1.94 ./Platform.Data.Doublets/UInt64LinksTransactionsLayer.cs

```

1  using System;
2  using System.Linq;
3  using System.Collections.Generic;
4  using System.IO;
5  using System.Runtime.CompilerServices;
6  using System.Threading;
7  using System.Threading.Tasks;
8  using Platform.Disposables;
9  using Platform.Timestamps;
10 using Platform.Unsafe;
11 using Platform.IO;
12 using Platform.Data.Doublets.Decorators;
13 using Platform.Exceptions;
14
15 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
16
17 namespace Platform.Data.Doublets
18 {
19     public class UInt64LinksTransactionsLayer : LinksDisposableDecoratorBase<ulong> //-V3073
20     {
21         /// <remarks>
22         /// Альтернативные варианты хранения трансформации (элемента транзакции):
23         ///
24         /// private enum TransitionType
25         /// {
26         ///     Creation,
27         ///     UpdateOf,
28         ///     UpdateTo,
29         ///     Deletion
30         /// }
31         ///
32         /// private struct Transition
33         /// {
34         ///     public ulong TransactionId;
35         ///     public UniqueTimestamp Timestamp;
36         ///     public TransactionItemType Type;
37         ///     public Link Source;
38         ///     public Link Linker;
39         ///     public Link Target;
40         /// }
41         ///
42         /// Или
43         ///
44         /// public struct TransitionHeader

```

```

45     /// {
46     ///     public ulong TransactionIdCombined;
47     ///     public ulong TimestampCombined;
48     ///
49     ///     public ulong TransactionId
50     ///     {
51     ///         get
52     ///         {
53     ///             return (ulong) mask & TransactionIdCombined;
54     ///         }
55     ///     }
56     ///
57     ///     public UniqueTimestamp Timestamp
58     ///     {
59     ///         get
60     ///         {
61     ///             return (UniqueTimestamp)mask & TransactionIdCombined;
62     ///         }
63     ///     }
64     ///
65     ///     public TransactionItemType Type
66     ///     {
67     ///         get
68     ///         {
69     ///             // Использовать по одному биту из TransactionId и Timestamp,
70     ///             // для значения в 2 бита, которое представляет тип операции
71     ///             throw new NotImplementedException();
72     ///         }
73     ///     }
74     /// }
75     ///
76     /// private struct Transition
77     /// {
78     ///     public TransitionHeader Header;
79     ///     public Link Source;
80     ///     public Link Linker;
81     ///     public Link Target;
82     /// }
83     ///
84     </remarks>
85     public struct Transition : IEquatable<Transition>
86     {
87         public static readonly long Size = Structure<Transition>.Size;
88
89         public readonly ulong TransactionId;
90         public readonly Link<ulong> Before;
91         public readonly Link<ulong> After;
92         public readonly Timestamp Timestamp;
93
94         [MethodImpl(MethodImplOptions.AggressiveInlining)]
95         public Transition(UniqueTimestampFactory uniqueTimestampFactory, ulong
96         ↪ transactionId, Link<ulong> before, Link<ulong> after)
97         {
98             TransactionId = transactionId;
99             Before = before;
100             After = after;
101             Timestamp = uniqueTimestampFactory.Create();
102         }
103
104         [MethodImpl(MethodImplOptions.AggressiveInlining)]
105         public Transition(UniqueTimestampFactory uniqueTimestampFactory, ulong
106         ↪ transactionId, Link<ulong> before) : this(uniqueTimestampFactory, transactionId,
107         ↪ before, default) { }
108
109         [MethodImpl(MethodImplOptions.AggressiveInlining)]
110         public Transition(UniqueTimestampFactory uniqueTimestampFactory, ulong
111         ↪ transactionId) : this(uniqueTimestampFactory, transactionId, default, default) {
112         ↪ }
113
114         [MethodImpl(MethodImplOptions.AggressiveInlining)]
115         public override string ToString() => $"{Timestamp} {TransactionId}: {Before} =>
116         ↪ {After}";
117
118         [MethodImpl(MethodImplOptions.AggressiveInlining)]
119         public override bool Equals(object obj) => obj is Transition transition ?
120         ↪ Equals(transition) : false;
121
122         [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

116     public override int GetHashCode() => (TransactionId, Before, After,
117         ↳ Timestamp).GetHashCode();
118
119     [MethodImpl(MethodImplOptions.AggressiveInlining)]
120     public bool Equals(Transition other) => TransactionId == other.TransactionId &&
121         ↳ Before == other.Before && After == other.After && Timestamp == other.Timestamp;
122
123     [MethodImpl(MethodImplOptions.AggressiveInlining)]
124     public static bool operator ==(Transition left, Transition right) =>
125         ↳ left.Equals(right);
126
127     [MethodImpl(MethodImplOptions.AggressiveInlining)]
128     public static bool operator !=(Transition left, Transition right) => !(left ==
129         ↳ right);
130 }
131
132 /// <remarks>
133 /// Другие варианты реализации транзакций (атомарности):
134 /// 1. Разделение хранения значения связи ((Source Target) или (Source Linker
135   ↳ Target)) и индексов.
136 /// 2. Хранение трансформаций/операций в отдельном хранилище Links, но дополнительно
137   ↳ потребуется решить вопрос
138   ↳ со ссылками на внешние идентификаторы, или как-то иначе решить вопрос с
139   ↳ пересечениями идентификаторов.
140 ///
141 /// Где хранить промежуточный список транзакций?
142 ///
143 /// В оперативной памяти:
144 /// Минусы:
145 /// 1. Может усложнить систему, если она будет функционировать самостоятельно,
146   ↳ так как нужно отдельно выделять память под список трансформаций.
147 /// 2. Выделенной оперативной памяти может не хватить, в том случае,
148   ↳ если транзакция использует слишком много трансформаций.
149   ↳ -> Можно использовать жёсткий диск для слишком длинных транзакций.
150   ↳ -> Максимальный размер списка трансформаций можно ограничить / задать
151   ↳ константой.
152 /// 3. При подтверждении транзакции (Commit) все трансформации записываются разом
153   ↳ создавая задержку.
154 ///
155 /// На жёстком диске:
156 /// Минусы:
157 /// 1. Длительный отклик, на запись каждой трансформации.
158 /// 2. Лог транзакций дополнительно наполняется отменёнными транзакциями.
159   ↳ -> Это может решаться упаковкой/исключением дублирующих операций.
160   ↳ -> Также это может решаться тем, что короткие транзакции вообще
161   ↳ не будут записываться в случае отката.
162 /// 3. Перед тем как выполнять отмену операций транзакции нужно дождаться пока все
163   ↳ операции (трансформации)
164   ↳ будут записаны в лог.
165 /// </remarks>
166 public class Transaction : DisposableBase
167 {
168     private readonly Queue<Transition> _transitions;
169     private readonly UInt64LinksTransactionsLayer _layer;
170     public bool IsCommitted { get; private set; }
171     public bool IsReverted { get; private set; }
172
173     [MethodImpl(MethodImplOptions.AggressiveInlining)]
174     public Transaction(UInt64LinksTransactionsLayer layer)
175     {
176         _layer = layer;
177         if (_layer._currentTransactionId != 0)
178         {
179             throw new NotSupportedException("Nested transactions not supported.");
180         }
181         IsCommitted = false;
182         IsReverted = false;
183         _transitions = new Queue<Transition>();
184         SetCurrentTransaction(layer, this);
185     }
186
187     [MethodImpl(MethodImplOptions.AggressiveInlining)]
188     public void Commit()
189     {
190         EnsureTransactionAllowsWriteOperations(this);
191         while (_transitions.Count > 0)
192         {
193

```

```

184         var transition = _transitions.Dequeue();
185         _layer._transitions.Enqueue(transition);
186     }
187     _layer._lastCommittedTransactionId = _layer._currentTransactionId;
188     IsCommitted = true;
189 }
190
191 [MethodImpl(MethodImplOptions.AggressiveInlining)]
192 private void Revert()
193 {
194     EnsureTransactionAllowsWriteOperations(this);
195     var transitionsToRevert = new Transition[_transitions.Count];
196     _transitions.CopyTo(transitionsToRevert, 0);
197     for (var i = transitionsToRevert.Length - 1; i >= 0; i--)
198     {
199         _layer.RevertTransition(transitionsToRevert[i]);
200     }
201     IsReverted = true;
202 }
203
204 [MethodImpl(MethodImplOptions.AggressiveInlining)]
205 public static void SetCurrentTransaction(UInt64LinksTransactionsLayer layer,
    ↳ Transaction transaction)
206 {
207     layer._currentTransactionId = layer._lastCommittedTransactionId + 1;
208     layer._currentTransactionTransitions = transaction._transitions;
209     layer._currentTransaction = transaction;
210 }
211
212 [MethodImpl(MethodImplOptions.AggressiveInlining)]
213 public static void EnsureTransactionAllowsWriteOperations(Transaction transaction)
214 {
215     if (transaction.IsReverted)
216     {
217         throw new InvalidOperationException("Transation is reverted.");
218     }
219     if (transaction.IsCommitted)
220     {
221         throw new InvalidOperationException("Transation is committed.");
222     }
223 }
224
225 [MethodImpl(MethodImplOptions.AggressiveInlining)]
226 protected override void Dispose(bool manual, bool wasDisposed)
227 {
228     if (!wasDisposed && _layer != null && !_layer.Disposable.IsDisposed)
229     {
230         if (!IsCommitted && !IsReverted)
231         {
232             Revert();
233         }
234         _layer.ResetCurrentTransation();
235     }
236 }
237
238
239 public static readonly TimeSpan DefaultPushDelay = TimeSpan.FromSeconds(0.1);
240
241 private readonly string _logAddress;
242 private readonly FileStream _log;
243 private readonly Queue<Transition> _transitions;
244 private readonly UniqueTimestampFactory _uniqueTimestampFactory;
245 private Task _transitionsPusher;
246 private Transition _lastCommittedTransition;
247 private ulong _currentTransactionId;
248 private Queue<Transition> _currentTransactionTransitions;
249 private Transaction _currentTransaction;
250 private ulong _lastCommittedTransactionId;
251
252 [MethodImpl(MethodImplOptions.AggressiveInlining)]
253 public UInt64LinksTransactionsLayer(ILinks<ulong> links, string logAddress)
254     : base(links)
255 {
256     if (string.IsNullOrEmpty(logAddress))
257     {
258         throw new ArgumentNullException(nameof(logAddress));
259     }
260     // В первой строке файла хранится последняя закоммиченную транзакцию.
261     // При запуске это используется для проверки удачного закрытия файла лога.

```

```

262 // In the first line of the file the last committed transaction is stored.
263 // On startup, this is used to check that the log file is successfully closed.
264 var lastCommittedTransition = FileHelpers.ReadFirstOrDefault<Transition>(logAddress);
265 var lastWrittenTransition = FileHelpers.ReadLastOrDefault<Transition>(logAddress);
266 if (!lastCommittedTransition.Equals(lastWrittenTransition))
267 {
268     Dispose();
269     throw new NotSupportedException("Database is damaged, autorecovery is not
    ↳ supported yet.");
270 }
271 if (lastCommittedTransition == default)
272 {
273     FileHelpers.WriteFirst(logAddress, lastCommittedTransition);
274 }
275 _lastCommittedTransition = lastCommittedTransition;
276 // TODO: Think about a better way to calculate or store this value
277 var allTransitions = FileHelpers.ReadAll<Transition>(logAddress);
278 _lastCommittedTransactionId = allTransitions.Length > 0 ? allTransitions.Max(x =>
    ↳ x.TransactionId) : 0;
279 _uniqueTimestampFactory = new UniqueTimestampFactory();
280 _logAddress = logAddress;
281 _log = FileHelpers.Append(logAddress);
282 _transitions = new Queue<Transition>();
283 _transitionsPusher = new Task(TransitionsPusher);
284 _transitionsPusher.Start();
285 }
286
287 [MethodImpl(MethodImplOptions.AggressiveInlining)]
288 public IList<ulong> GetLinkValue(ulong link) => Links.GetLink(link);
289
290 [MethodImpl(MethodImplOptions.AggressiveInlining)]
291 public override ulong Create(IList<ulong> restrictions)
292 {
293     var createdLinkIndex = Links.Create();
294     var createdLink = new Link<ulong>(Links.GetLink(createdLinkIndex));
295     CommitTransition(new Transition(_uniqueTimestampFactory, _currentTransactionId,
    ↳ default, createdLink));
296     return createdLinkIndex;
297 }
298
299 [MethodImpl(MethodImplOptions.AggressiveInlining)]
300 public override ulong Update(IList<ulong> restrictions, IList<ulong> substitution)
301 {
302     var linkIndex = restrictions[Constants.IndexPart];
303     var beforeLink = new Link<ulong>(Links.GetLink(linkIndex));
304     linkIndex = Links.Update(restrictions, substitution);
305     var afterLink = new Link<ulong>(Links.GetLink(linkIndex));
306     CommitTransition(new Transition(_uniqueTimestampFactory, _currentTransactionId,
    ↳ beforeLink, afterLink));
307     return linkIndex;
308 }
309
310 [MethodImpl(MethodImplOptions.AggressiveInlining)]
311 public override void Delete(IList<ulong> restrictions)
312 {
313     var link = restrictions[Constants.IndexPart];
314     var deletedLink = new Link<ulong>(Links.GetLink(link));
315     Links.Delete(link);
316     CommitTransition(new Transition(_uniqueTimestampFactory, _currentTransactionId,
    ↳ deletedLink, default));
317 }
318
319 [MethodImpl(MethodImplOptions.AggressiveInlining)]
320 private Queue<Transition> GetCurrentTransitions() => _currentTransactionTransitions ??
    ↳ _transitions;
321
322 [MethodImpl(MethodImplOptions.AggressiveInlining)]
323 private void CommitTransition(Transition transition)
324 {
325     if (_currentTransaction != null)
326     {
327         Transaction.EnsureTransactionAllowsWriteOperations(_currentTransaction);
328     }
329     var transitions = GetCurrentTransitions();
330     transitions.Enqueue(transition);
331 }
332
333 [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

334 private void RevertTransition(Transition transition)
335 {
336     if (transition.After.IsNull()) // Revert Deletion with Creation
337     {
338         Links.Create();
339     }
340     else if (transition.Before.IsNull()) // Revert Creation with Deletion
341     {
342         Links.Delete(transition.After.Index);
343     }
344     else // Revert Update
345     {
346         Links.Update(new[] { transition.After.Index, transition.Before.Source,
347                               ↪ transition.Before.Target });
348     }
349 }
350 [MethodImpl(MethodImplOptions.AggressiveInlining)]
351 private void ResetCurrentTransation()
352 {
353     _currentTransactionId = 0;
354     _currentTransactionTransitions = null;
355     _currentTransaction = null;
356 }
357 [MethodImpl(MethodImplOptions.AggressiveInlining)]
358 private void PushTransitions()
359 {
360     if (_log == null || _transitions == null)
361     {
362         return;
363     }
364     for (var i = 0; i < _transitions.Count; i++)
365     {
366         var transition = _transitions.Dequeue();
367
368         _log.Write(transition);
369         _lastCommittedTransition = transition;
370     }
371 }
372 }
373 [MethodImpl(MethodImplOptions.AggressiveInlining)]
374 private void TransitionsPusher()
375 {
376     while (!Disposable.IsDisposed && _transitionsPusher != null)
377     {
378         Thread.Sleep(DefaultPushDelay);
379         PushTransitions();
380     }
381 }
382 }
383 [MethodImpl(MethodImplOptions.AggressiveInlining)]
384 public Transaction BeginTransaction() => new Transaction(this);
385 [MethodImpl(MethodImplOptions.AggressiveInlining)]
386 private void DisposeTransitions()
387 {
388     try
389     {
390         {
391             var pusher = _transitionsPusher;
392             if (pusher != null)
393             {
394                 _transitionsPusher = null;
395                 pusher.Wait();
396             }
397             if (_transitions != null)
398             {
399                 PushTransitions();
400             }
401             _log.DisposeIfPossible();
402             FileHelpers.WriteFirst(_logAddress, _lastCommittedTransition);
403         }
404         catch (Exception ex)
405         {
406             ex.Ignore();
407         }
408     }
409 }
410 #region DisposalBase
411

```

```

412     [MethodImpl(MethodImplOptions.AggressiveInlining)]
413     protected override void Dispose(bool manual, bool wasDisposed)
414     {
415         if (!wasDisposed)
416         {
417             DisposeTransitions();
418         }
419         base.Dispose(manual, wasDisposed);
420     }
421 }
422 #endregion
423 }
424 }
425 }

```

1.95 ./Platform.Data.Doublets/Unicode/CharToUnicodeSymbolConverter.cs

```

1  using System.Runtime.CompilerServices;
2  using Platform.Converters;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Unicode
7  {
8      public class CharToUnicodeSymbolConverter<TLink> : LinksOperatorBase<TLink>,
9          ↪ IConverter<char, TLink>
10     {
11         private static readonly UncheckedConverter<char, TLink> _charToAddressConverter =
12             ↪ UncheckedConverter<char, TLink>.Default;
13
14         private readonly IConverter<TLink> _addressToNumberConverter;
15         private readonly TLink _unicodeSymbolMarker;
16
17         [MethodImpl(MethodImplOptions.AggressiveInlining)]
18         public CharToUnicodeSymbolConverter(ILinks<TLink> links, IConverter<TLink>
19             ↪ addressToNumberConverter, TLink unicodeSymbolMarker) : base(links)
20         {
21             _addressToNumberConverter = addressToNumberConverter;
22             _unicodeSymbolMarker = unicodeSymbolMarker;
23         }
24
25         [MethodImpl(MethodImplOptions.AggressiveInlining)]
26         public TLink Convert(char source)
27         {
28             var unaryNumber =
29                 ↪ _addressToNumberConverter.Convert(_charToAddressConverter.Convert(source));
30             return Links.GetOrCreate(unaryNumber, _unicodeSymbolMarker);
31         }
32     }
33 }

```

1.96 ./Platform.Data.Doublets/Unicode/StringToUnicodeSequenceConverter.cs

```

1  using System.Collections.Generic;
2  using System.Runtime.CompilerServices;
3  using Platform.Converters;
4  using Platform.Data.Doublets.Sequences.Indexes;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Unicode
9  {
10     public class StringToUnicodeSequenceConverter<TLink> : LinksOperatorBase<TLink>,
11         ↪ IConverter<string, TLink>
12     {
13         private readonly IConverter<char, TLink> _charToUnicodeSymbolConverter;
14         private readonly ISequenceIndex<TLink> _index;
15         private readonly IConverter<IList<TLink>, TLink> _listToSequenceLinkConverter;
16         private readonly TLink _unicodeSequenceMarker;
17
18         [MethodImpl(MethodImplOptions.AggressiveInlining)]
19         public StringToUnicodeSequenceConverter(ILinks<TLink> links, IConverter<char, TLink>
20             ↪ charToUnicodeSymbolConverter, ISequenceIndex<TLink> index, IConverter<IList<TLink>,
21             ↪ TLink> listToSequenceLinkConverter, TLink unicodeSequenceMarker) : base(links)
22         {
23             _charToUnicodeSymbolConverter = charToUnicodeSymbolConverter;
24             _index = index;
25             _listToSequenceLinkConverter = listToSequenceLinkConverter;
26             _unicodeSequenceMarker = unicodeSequenceMarker;
27         }
28
29         [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

27     public TLink Convert(string source)
28     {
29         var elements = new TLink[source.Length];
30         for (int i = 0; i < source.Length; i++)
31         {
32             elements[i] = _charToUnicodeSymbolConverter.Convert(source[i]);
33         }
34         _index.Add(elements);
35         var sequence = _listToSequenceLinkConverter.Convert(elements);
36         return Links.GetOrCreate(sequence, _unicodeSequenceMarker);
37     }
38 }
39 }

```

1.97 ./Platform.Data.Doublets/Unicode/UnicodeMap.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Globalization;
4  using System.Runtime.CompilerServices;
5  using System.Text;
6  using Platform.Data.Sequences;
7
8  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
9
10 namespace Platform.Data.Doublets.Unicode
11 {
12     public class UnicodeMap
13     {
14         public static readonly ulong FirstCharLink = 1;
15         public static readonly ulong LastCharLink = FirstCharLink + char.MaxValue;
16         public static readonly ulong MapSize = 1 + char.MaxValue;
17
18         private readonly ILinks<ulong> _links;
19         private bool _initialized;
20
21         [MethodImpl(MethodImplOptions.AggressiveInlining)]
22         public UnicodeMap(ILinks<ulong> links) => _links = links;
23
24         [MethodImpl(MethodImplOptions.AggressiveInlining)]
25         public static UnicodeMap InitNew(ILinks<ulong> links)
26         {
27             var map = new UnicodeMap(links);
28             map.Init();
29             return map;
30         }
31
32         [MethodImpl(MethodImplOptions.AggressiveInlining)]
33         public void Init()
34         {
35             if (_initialized)
36             {
37                 return;
38             }
39             _initialized = true;
40             var firstLink = _links.CreatePoint();
41             if (firstLink != FirstCharLink)
42             {
43                 _links.Delete(firstLink);
44             }
45             else
46             {
47                 for (var i = FirstCharLink + 1; i <= LastCharLink; i++)
48                 {
49                     // From NIL to It (NIL -> Character) transformation meaning, (or infinite
50                     //   ↳ amount of NIL characters before actual Character)
51                     var createdLink = _links.CreatePoint();
52                     _links.Update(createdLink, firstLink, createdLink);
53                     if (createdLink != i)
54                     {
55                         throw new InvalidOperationException("Unable to initialize UTF 16
56                         ↳ table.");
57                     }
58                 }
59             }
60         }
61
62         // 0 - null link
63         // 1 - nil character (0 character)
64         // ...
65         // 65536 (0(1) + 65535 = 65536 possible values)

```



```

64 [MethodImpl(MethodImplOptions.AggressiveInlining)]
65 public static ulong FromCharToLink(char character) => (ulong)character + 1;
66
67 [MethodImpl(MethodImplOptions.AggressiveInlining)]
68 public static char FromLinkToChar(ulong link) => (char)(link - 1);
69
70 [MethodImpl(MethodImplOptions.AggressiveInlining)]
71 public static bool IsCharLink(ulong link) => link <= MapSize;
72
73 [MethodImpl(MethodImplOptions.AggressiveInlining)]
74 public static string FromLinksToString(IList<ulong> linksList)
75 {
76     var sb = new StringBuilder();
77     for (int i = 0; i < linksList.Count; i++)
78     {
79         sb.Append(FromLinkToChar(linksList[i]));
80     }
81     return sb.ToString();
82 }
83
84 [MethodImpl(MethodImplOptions.AggressiveInlining)]
85 public static string FromSequenceLinkToString(ulong link, ILinks<ulong> links)
86 {
87     var sb = new StringBuilder();
88     if (links.Exists(link))
89     {
90         StopableSequenceWalker.WalkRight(link, links.GetSource, links.GetTarget,
91             x => x <= MapSize || links.GetSource(x) == x || links.GetTarget(x) == x,
92             ↪ element =>
93             {
94                 sb.Append(FromLinkToChar(element));
95                 return true;
96             }
97         );
98     }
99     return sb.ToString();
100 }
101
102 [MethodImpl(MethodImplOptions.AggressiveInlining)]
103 public static ulong[] FromCharsToLinkArray(char[] chars) => FromCharsToLinkArray(chars,
104     ↪ chars.Length);
105
106 [MethodImpl(MethodImplOptions.AggressiveInlining)]
107 public static ulong[] FromCharsToLinkArray(char[] chars, int count)
108 {
109     // char array to ulong array
110     var linksSequence = new ulong[count];
111     for (var i = 0; i < count; i++)
112     {
113         linksSequence[i] = FromCharToLink(chars[i]);
114     }
115     return linksSequence;
116 }
117
118 [MethodImpl(MethodImplOptions.AggressiveInlining)]
119 public static ulong[] FromStringToLinkArray(string sequence)
120 {
121     // char array to ulong array
122     var linksSequence = new ulong[sequence.Length];
123     for (var i = 0; i < sequence.Length; i++)
124     {
125         linksSequence[i] = FromCharToLink(sequence[i]);
126     }
127     return linksSequence;
128 }
129
130 [MethodImpl(MethodImplOptions.AggressiveInlining)]
131 public static List<ulong[]> FromStringToLinkArrayGroups(string sequence)
132 {
133     var result = new List<ulong[]>();
134     var offset = 0;
135     while (offset < sequence.Length)
136     {
137         var currentCategory = CharUnicodeInfo.GetUnicodeCategory(sequence[offset]);
138         var relativeLength = 1;
139         var absoluteLength = offset + relativeLength;
140         while (absoluteLength < sequence.Length &&
141             currentCategory ==
142             ↪ CharUnicodeInfo.GetUnicodeCategory(sequence[absoluteLength]))

```

```

140         {
141             relativeLength++;
142             absoluteLength++;
143         }
144         // char array to ulong array
145         var innerSequence = new ulong[relativeLength];
146         var maxLength = offset + relativeLength;
147         for (var i = offset; i < maxLength; i++)
148         {
149             innerSequence[i - offset] = FromCharToLink(sequence[i]);
150         }
151         result.Add(innerSequence);
152         offset += relativeLength;
153     }
154     return result;
155 }
156
157 [MethodImpl(MethodImplOptions.AggressiveInlining)]
158 public static List<ulong[]> FromLinkArrayToLinkArrayGroups(ulong[] array)
159 {
160     var result = new List<ulong[]>();
161     var offset = 0;
162     while (offset < array.Length)
163     {
164         var relativeLength = 1;
165         if (array[offset] <= LastCharLink)
166         {
167             var currentCategory =
168                 ↳ CharUnicodeInfo.GetUnicodeCategory(FromLinkToChar(array[offset]));
169             var absoluteLength = offset + relativeLength;
170             while (absoluteLength < array.Length &&
171                 array[absoluteLength] <= LastCharLink &&
172                 currentCategory == CharUnicodeInfo.GetUnicodeCategory(FromLinkToChar(
173                     ↳ array[absoluteLength])))
174             {
175                 relativeLength++;
176                 absoluteLength++;
177             }
178         }
179         else
180         {
181             var absoluteLength = offset + relativeLength;
182             while (absoluteLength < array.Length && array[absoluteLength] > LastCharLink)
183             {
184                 relativeLength++;
185                 absoluteLength++;
186             }
187         }
188         // copy array
189         var innerSequence = new ulong[relativeLength];
190         var maxLength = offset + relativeLength;
191         for (var i = offset; i < maxLength; i++)
192         {
193             innerSequence[i - offset] = array[i];
194         }
195         result.Add(innerSequence);
196         offset += relativeLength;
197     }
198     return result;
199 }

```

1.98 ./Platform.Data.Doublets/Unicode/UnicodeSequenceCriterionMatcher.cs

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3 using Platform.Interfaces;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data.Doublets.Unicode
8 {
9     public class UnicodeSequenceCriterionMatcher<TLink> : LinksOperatorBase<TLink>,
10         ↳ ICriterionMatcher<TLink>
11     {
12         private static readonly EqualityComparer<TLink> _equalityComparer =
13             ↳ EqualityComparer<TLink>.Default;
14
15         private readonly TLink _unicodeSequenceMarker;
16     }
17 }

```

```

15     [MethodImpl(MethodImplOptions.AggressiveInlining)]
16     public UnicodeSequenceCriterionMatcher(ILinks<TLink> links, TLink unicodeSequenceMarker)
17         ↪ : base(links) => _unicodeSequenceMarker = unicodeSequenceMarker;
18
19     [MethodImpl(MethodImplOptions.AggressiveInlining)]
20     public bool IsMatched(TLink link) => _equalityComparer.Equals(Links.GetTarget(link),
21         ↪ _unicodeSequenceMarker);

```

1.99 ./Platform.Data.Doublets/Unicode/UnicodeSequenceToStringConverter.cs

```

1  using System;
2  using System.Linq;
3  using System.Runtime.CompilerServices;
4  using Platform.Interfaces;
5  using Platform.Converters;
6  using Platform.Data.Doublets.Sequences.Walkers;
7
8  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
9
10 namespace Platform.Data.Doublets.Unicode
11 {
12     public class UnicodeSequenceToStringConverter<TLink> : LinksOperatorBase<TLink>,
13         ↪ IConverter<TLink, string>
14     {
15         private readonly ICriterionMatcher<TLink> _unicodeSequenceCriterionMatcher;
16         private readonly ISequenceWalker<TLink> _sequenceWalker;
17         private readonly IConverter<TLink, char> _unicodeSymbolToCharConverter;
18
19         [MethodImpl(MethodImplOptions.AggressiveInlining)]
20         public UnicodeSequenceToStringConverter(ILinks<TLink> links, ICriterionMatcher<TLink>
21             ↪ unicodeSequenceCriterionMatcher, ISequenceWalker<TLink> sequenceWalker,
22             ↪ IConverter<TLink, char> unicodeSymbolToCharConverter) : base(links)
23         {
24             _unicodeSequenceCriterionMatcher = unicodeSequenceCriterionMatcher;
25             _sequenceWalker = sequenceWalker;
26             _unicodeSymbolToCharConverter = unicodeSymbolToCharConverter;
27         }
28
29         [MethodImpl(MethodImplOptions.AggressiveInlining)]
30         public string Convert(TLink source)
31         {
32             if(!_unicodeSequenceCriterionMatcher.IsMatched(source))
33             {
34                 throw new ArgumentOutOfRangeException(nameof(source), source, "Specified link is
35                     ↪ not a unicode sequence.");
36             }
37             var sequence = Links.GetSource(source);
38             var charArray = _sequenceWalker.Walk(sequence).Select(_unicodeSymbolToCharConverter.
39                 ↪ Convert).ToArray();
40             return new string(charArray);
41         }
42     }
43 }

```

1.100 ./Platform.Data.Doublets/Unicode/UnicodeSymbolCriterionMatcher.cs

```

1  using System.Collections.Generic;
2  using System.Runtime.CompilerServices;
3  using Platform.Interfaces;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.Unicode
8  {
9      public class UnicodeSymbolCriterionMatcher<TLink> : LinksOperatorBase<TLink>,
10         ↪ ICriterionMatcher<TLink>
11     {
12         private static readonly EqualityComparer<TLink> _equalityComparer =
13             ↪ EqualityComparer<TLink>.Default;
14
15         private readonly TLink _unicodeSymbolMarker;
16
17         [MethodImpl(MethodImplOptions.AggressiveInlining)]
18         public UnicodeSymbolCriterionMatcher(ILinks<TLink> links, TLink unicodeSymbolMarker) :
19             ↪ base(links) => _unicodeSymbolMarker = unicodeSymbolMarker;
20
21         [MethodImpl(MethodImplOptions.AggressiveInlining)]
22         public bool IsMatched(TLink link) => _equalityComparer.Equals(Links.GetTarget(link),
23             ↪ _unicodeSymbolMarker);

```

```

20     }
21 }

```

1.101 ./Platform.Data.Doublets/Unicode/UnicodeSymbolToCharConverter.cs

```

1  using System;
2  using System.Runtime.CompilerServices;
3  using Platform.Interfaces;
4  using Platform.Converters;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Unicode
9  {
10     public class UnicodeSymbolToCharConverter<TLink> : LinksOperatorBase<TLink>,
        ↳ IConverter<TLink, char>
11     {
12         private static readonly UncheckedConverter<TLink, char> _addressToCharConverter =
        ↳ UncheckedConverter<TLink, char>.Default;
13
14         private readonly IConverter<TLink> _numberToAddressConverter;
15         private readonly ICriterionMatcher<TLink> _unicodeSymbolCriterionMatcher;
16
17         [MethodImpl(MethodImplOptions.AggressiveInlining)]
18         public UnicodeSymbolToCharConverter(ILinks<TLink> links, IConverter<TLink>
        ↳ numberToAddressConverter, ICriterionMatcher<TLink> unicodeSymbolCriterionMatcher) :
        ↳ base(links)
19         {
20             _numberToAddressConverter = numberToAddressConverter;
21             _unicodeSymbolCriterionMatcher = unicodeSymbolCriterionMatcher;
22         }
23
24         [MethodImpl(MethodImplOptions.AggressiveInlining)]
25         public char Convert(TLink source)
26         {
27             if (!_unicodeSymbolCriterionMatcher.IsMatched(source))
28             {
29                 throw new ArgumentOutOfRangeException(nameof(source), source, "Specified link is
        ↳ not a unicode symbol.");
30             }
31             return _addressToCharConverter.Convert(_numberToAddressConverter.Convert(Links.GetSo_
        ↳ urce(source)));
32         }
33     }
34 }

```

1.102 ./Platform.Data.Doublets.Tests/ComparisonTests.cs

```

1  using System;
2  using System.Collections.Generic;
3  using Xunit;
4  using Platform.Diagnostics;
5
6  namespace Platform.Data.Doublets.Tests
7  {
8     public static class ComparisonTests
9     {
10         private class UInt64Comparer : IComparer<ulong>
11         {
12             public int Compare(ulong x, ulong y) => x.CompareTo(y);
13         }
14
15         private static int Compare(ulong x, ulong y) => x.CompareTo(y);
16
17         [Fact]
18         public static void GreaterOrEqualPerformanceTest()
19         {
20             const int N = 1000000;
21
22             ulong x = 10;
23             ulong y = 500;
24
25             bool result = false;
26
27             var ts1 = Performance.Measure(() =>
28             {
29                 for (int i = 0; i < N; i++)
30                 {
31                     result = Compare(x, y) >= 0;
32                 }
33             });
34 }

```

```

35     var comparer1 = Comparer<ulong>.Default;
36
37     var ts2 = Performance.Measure(() =>
38     {
39         for (int i = 0; i < N; i++)
40         {
41             result = comparer1.Compare(x, y) >= 0;
42         }
43     });
44
45     Func<ulong, ulong, int> compareReference = comparer1.Compare;
46
47     var ts3 = Performance.Measure(() =>
48     {
49         for (int i = 0; i < N; i++)
50         {
51             result = compareReference(x, y) >= 0;
52         }
53     });
54
55     var comparer2 = new UInt64Comparer();
56
57     var ts4 = Performance.Measure(() =>
58     {
59         for (int i = 0; i < N; i++)
60         {
61             result = comparer2.Compare(x, y) >= 0;
62         }
63     });
64
65     Console.WriteLine($"{ts1} {ts2} {ts3} {ts4} {result}");
66 }
67
68 }

```

1.103 ./Platform.Data.Doublets.Tests/EqualityTests.cs

```

1  using System;
2  using System.Collections.Generic;
3  using Xunit;
4  using Platform.Diagnostics;
5
6  namespace Platform.Data.Doublets.Tests
7  {
8      public static class EqualityTests
9      {
10         protected class UInt64EqualityComparer : IEqualityComparer<ulong>
11         {
12             public bool Equals(ulong x, ulong y) => x == y;
13
14             public int GetHashCode(ulong obj) => obj.GetHashCode();
15         }
16
17         private static bool Equals1<T>(T x, T y) => Equals(x, y);
18
19         private static bool Equals2<T>(T x, T y) => x.Equals(y);
20
21         private static bool Equals3(ulong x, ulong y) => x == y;
22
23         [Fact]
24         public static void EqualsPerfomanceTest()
25         {
26             const int N = 1000000;
27
28             ulong x = 10;
29             ulong y = 500;
30
31             bool result = false;
32
33             var ts1 = Performance.Measure(() =>
34             {
35                 for (int i = 0; i < N; i++)
36                 {
37                     result = Equals1(x, y);
38                 }
39             });
40
41             var ts2 = Performance.Measure(() =>
42             {
43                 for (int i = 0; i < N; i++)
44                 {

```

```

45         result = Equals2(x, y);
46     }
47 });
48
49 var ts3 = Performance.Measure(() =>
50 {
51     for (int i = 0; i < N; i++)
52     {
53         result = Equals3(x, y);
54     }
55 });
56
57 var equalityComparer1 = EqualityComparer<ulong>.Default;
58
59 var ts4 = Performance.Measure(() =>
60 {
61     for (int i = 0; i < N; i++)
62     {
63         result = equalityComparer1.Equals(x, y);
64     }
65 });
66
67 var equalityComparer2 = new UInt64EqualityComparer();
68
69 var ts5 = Performance.Measure(() =>
70 {
71     for (int i = 0; i < N; i++)
72     {
73         result = equalityComparer2.Equals(x, y);
74     }
75 });
76
77 Func<ulong, ulong, bool> equalityComparer3 = equalityComparer2.Equals;
78
79 var ts6 = Performance.Measure(() =>
80 {
81     for (int i = 0; i < N; i++)
82     {
83         result = equalityComparer3(x, y);
84     }
85 });
86
87 var comparer = Comparer<ulong>.Default;
88
89 var ts7 = Performance.Measure(() =>
90 {
91     for (int i = 0; i < N; i++)
92     {
93         result = comparer.Compare(x, y) == 0;
94     }
95 });
96
97 Assert.True(ts2 < ts1);
98 Assert.True(ts3 < ts2);
99 Assert.True(ts5 < ts4);
100 Assert.True(ts5 < ts6);
101
102 Console.WriteLine($"{ts1} {ts2} {ts3} {ts4} {ts5} {ts6} {ts7} {result}");
103 }
104 }
105 }

```

1.104 ./Platform.Data.Doublets.Tests/GenericLinksTests.cs

```

1  using System;
2  using Xunit;
3  using Platform.Reflection;
4  using Platform.Memory;
5  using Platform.Scopes;
6  using Platform.Data.Doublets.ResizableDirectMemory.Generic;
7
8  namespace Platform.Data.Doublets.Tests
9  {
10     public unsafe static class GenericLinksTests
11     {
12         [Fact]
13         public static void CRUDTest()
14         {
15             Using<byte>(links => links.TestCRUDOperations());
16             Using<ushort>(links => links.TestCRUDOperations());
17             Using<uint>(links => links.TestCRUDOperations());

```

```

18         Using<ulong>(links => links.TestCRUDOperations());
19     }
20
21     [Fact]
22     public static void RawNumbersCRUDTest()
23     {
24         Using<byte>(links => links.TestRawNumbersCRUDOperations());
25         Using<ushort>(links => links.TestRawNumbersCRUDOperations());
26         Using<uint>(links => links.TestRawNumbersCRUDOperations());
27         Using<ulong>(links => links.TestRawNumbersCRUDOperations());
28     }
29
30     [Fact]
31     public static void MultipleRandomCreationsAndDeletionsTest()
32     {
33         Using<byte>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Test
34             ↪ MultipleRandomCreationsAndDeletions(16)); // Cannot use more because current
35             ↪ implementation of tree cuts out 5 bits from the address space.
36         Using<ushort>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Te
37             ↪ stMultipleRandomCreationsAndDeletions(100));
38         Using<uint>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Test
39             ↪ MultipleRandomCreationsAndDeletions(100));
40         Using<ulong>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Tes
41             ↪ tMultipleRandomCreationsAndDeletions(100));
42     }
43
44     private static void Using<TLink>(Action<ILinks<TLink>> action)
45     {
46         using (var scope = new Scope<Types<HeapResizableDirectMemory,
47             ↪ ResizableDirectMemoryLinks<TLink>>>())
48         {
49             action(scope.Use<ILinks<TLink>>());
50         }
51     }
52 }

```

1.105 ./Platform.Data.Doublets.Tests/LinksConstantsTests.cs

```

1  using Xunit;
2
3  namespace Platform.Data.Doublets.Tests
4  {
5      public static class LinksConstantsTests
6      {
7          [Fact]
8          public static void ExternalReferencesTest()
9          {
10             LinksConstants<ulong> constants = new LinksConstants<ulong>((1, long.MaxValue),
11                 ↪ (long.MaxValue + 1UL, ulong.MaxValue));
12
13             //var minimum = new Hybrid<ulong>(0, isExternal: true);
14             var minimum = new Hybrid<ulong>(1, isExternal: true);
15             var maximum = new Hybrid<ulong>(long.MaxValue, isExternal: true);
16
17             Assert.True(constants.IsExternalReference(minimum));
18             Assert.True(constants.IsExternalReference(maximum));
19         }
20     }

```

1.106 ./Platform.Data.Doublets.Tests/OptimalVariantSequenceTests.cs

```

1  using System;
2  using System.Linq;
3  using Xunit;
4  using Platform.Collections.Stacks;
5  using Platform.Collections.Arrays;
6  using Platform.Memory;
7  using Platform.Data.Numbers.Raw;
8  using Platform.Data.Doublets.Sequences;
9  using Platform.Data.Doublets.Sequences.Frequencies.Cache;
10 using Platform.Data.Doublets.Sequences.Frequencies.Counters;
11 using Platform.Data.Doublets.Sequences.Converters;
12 using Platform.Data.Doublets.PropertyOperators;
13 using Platform.Data.Doublets.Incrementers;
14 using Platform.Data.Doublets.Sequences.Walkers;
15 using Platform.Data.Doublets.Sequences.Indexes;
16 using Platform.Data.Doublets.Unicode;
17 using Platform.Data.Doublets.Numbers.Unary;
18 using Platform.Data.Doublets.Decorators;

```

```

19 using Platform.Data.Doubllets.ResizableDirectMemory.Specific;
20
21 namespace Platform.Data.Doubllets.Tests
22 {
23     public static class OptimalVariantSequenceTests
24     {
25         private static readonly string _sequenceExample = "зеленела зелёная зелень";
26         private static readonly string _loremIpsumExample = @"Lorem ipsum dolor sit amet,
        ↳ consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore
        ↳ magna aliqua.
Facilisi nullam vehicula ipsum a arcu cursus vitae congue mauris.
Et malesuada fames ac turpis egestas sed.
Eget velit aliquet sagittis id consectetur purus.
Dignissim cras tincidunt lobortis feugiat vivamus.
Vitae aliquet nec ullamcorper sit.
Lectus quam id leo in vitae.
Tortor dignissim convallis aenean et tortor at risus viverra adipiscing.
Sed risus ultricies tristique nulla aliquet enim tortor at auctor.
Integer eget aliquet nibh praesent tristique.
Vitae congue eu consequat ac felis donec et odio.
Tristique et egestas quis ipsum suspendisse.
Suspendisse potenti nullam ac tortor vitae purus faucibus ornare.
Nulla facilisi etiam dignissim diam quis enim lobortis scelerisque.
Imperdiet proin fermentum leo vel orci.
In ante metus dictum at tempor commodo.
Nisi lacus sed viverra tellus in.
Quam vulputate dignissim suspendisse in.
Elit scelerisque mauris pellentesque pulvinar pellentesque habitant morbi tristique senectus.
Gravida cum sociis natoque penatibus et magnis dis parturient.
Risus quis varius quam quisque id diam.
Congue nisi vitae suscipit tellus mauris a diam maecenas.
Eget nunc scelerisque viverra mauris in aliquam sem fringilla.
Pharetra vel turpis nunc eget lorem dolor sed viverra.
Mattis pellentesque id nibh tortor id aliquet.
Purus non enim praesent elementum facilisis leo vel.
Etiam sit amet nisl purus in mollis nunc sed.
Tortor at auctor urna nunc id cursus metus aliquam.
Volutpat odio facilisis mauris sit amet.
Turpis egestas pretium aenean pharetra magna ac placerat.
Fermentum dui faucibus in ornare quam viverra orci sagittis eu.
Porttitor leo a diam sollicitudin tempor id eu.
Volutpat sed cras ornare arcu dui.
Ut aliquam purus sit amet luctus venenatis lectus magna.
Aliquet risus feugiat in ante metus dictum at.
Mattis nunc sed blandit libero.
Elit pellentesque habitant morbi tristique senectus et netus.
Nibh sit amet commodo nulla facilisi nullam vehicula ipsum a.
Enim sit amet venenatis urna cursus eget nunc scelerisque viverra.
Amet venenatis urna cursus eget nunc scelerisque viverra mauris in.
Diam donec adipiscing tristique risus nec feugiat.
Pulvinar mattis nunc sed blandit libero volutpat.
Cras fermentum odio eu feugiat pretium nibh ipsum.
In nulla posuere sollicitudin aliquam ultrices sagittis orci a.
Mauris pellentesque pulvinar pellentesque habitant morbi tristique senectus et.
A iaculis at erat pellentesque.
Morbi blandit cursus risus at ultrices mi tempus imperdiet nulla.
Eget lorem dolor sed viverra ipsum nunc.
Leo a diam sollicitudin tempor id eu.
Interdum consectetur libero id faucibus nisl tincidunt eget nullam non.";
76
77     [Fact]
78     public static void LinksBasedFrequencyStoredOptimalVariantSequenceTest()
79     {
80         using (var scope = new TempLinksTestScope(useSequences: false))
81         {
82             var links = scope.Links;
83             var constants = links.Constants;
84
85             links.UseUnicode();
86
87             var sequence = UnicodeMap.FromStringToLinkArray(_sequenceExample);
88
89             var meaningRoot = links.CreatePoint();
90             var unaryOne = links.CreateAndUpdate(meaningRoot, constants.Itself);
91             var frequencyMarker = links.CreateAndUpdate(meaningRoot, constants.Itself);
92             var frequencyPropertyMarker = links.CreateAndUpdate(meaningRoot,
        ↳ constants.Itself);
93
94             var unaryNumberToAddressConverter = new
        ↳ UnaryNumberToAddressAddOperationConverter<ulong>(links, unaryOne);
95             var unaryNumberIncrementer = new UnaryNumberIncrementer<ulong>(links, unaryOne);
96             var frequencyIncrementer = new FrequencyIncrementer<ulong>(links,
        ↳ frequencyMarker, unaryOne, unaryNumberIncrementer);

```



```

var frequencyPropertyOperator = new PropertyOperator<ulong>(links,
    ↪ frequencyPropertyMarker, frequencyMarker);
var index = new FrequencyIncrementingSequenceIndex<ulong>(links,
    ↪ frequencyPropertyOperator, frequencyIncrementer);
var linkToItsFrequencyNumberConverter = new
    ↪ LinkToItsFrequencyNumberConverter<ulong>(links, frequencyPropertyOperator,
    ↪ unaryNumberToAddressConverter);
var sequenceToItsLocalElementLevelsConverter = new
    ↪ SequenceToItsLocalElementLevelsConverter<ulong>(links,
    ↪ linkToItsFrequencyNumberConverter);
var optimalVariantConverter = new OptimalVariantConverter<ulong>(links,
    ↪ sequenceToItsLocalElementLevelsConverter);

var sequences = new Sequences.Sequences(links, new SequencesOptions<ulong>() {
    ↪ Walker = new LeveledSequenceWalker<ulong>(links) });

ExecuteTest(sequences, sequence, sequenceToItsLocalElementLevelsConverter,
    ↪ index, optimalVariantConverter);
}
}

[Fact]
public static void DictionaryBasedFrequencyStoredOptimalVariantSequenceTest()
{
    using (var scope = new TempLinksTestScope(useSequences: false))
    {
        var links = scope.Links;

        links.UseUnicode();

        var sequence = UnicodeMap.FromStringToLinkArray(_sequenceExample);

        var totalSequenceSymbolFrequencyCounter = new
            ↪ TotalSequenceSymbolFrequencyCounter<ulong>(links);

        var linkFrequenciesCache = new LinkFrequenciesCache<ulong>(links,
            ↪ totalSequenceSymbolFrequencyCounter);

        var index = new
            ↪ CachedFrequencyIncrementingSequenceIndex<ulong>(linkFrequenciesCache);
        var linkToItsFrequencyNumberConverter = new FrequenciesCacheBasedLinkToItsFrequencyNumberConverter<ulong>(linkFrequenciesCache);

        var sequenceToItsLocalElementLevelsConverter = new
            ↪ SequenceToItsLocalElementLevelsConverter<ulong>(links,
            ↪ linkToItsFrequencyNumberConverter);
        var optimalVariantConverter = new OptimalVariantConverter<ulong>(links,
            ↪ sequenceToItsLocalElementLevelsConverter);

        var sequences = new Sequences.Sequences(links, new SequencesOptions<ulong>() {
            ↪ Walker = new LeveledSequenceWalker<ulong>(links) });

        ExecuteTest(sequences, sequence, sequenceToItsLocalElementLevelsConverter,
            ↪ index, optimalVariantConverter);
    }
}

private static void ExecuteTest(Sequences.Sequences sequences, ulong[] sequence,
    ↪ SequenceToItsLocalElementLevelsConverter<ulong>
    ↪ sequenceToItsLocalElementLevelsConverter, ISequenceIndex<ulong> index,
    ↪ OptimalVariantConverter<ulong> optimalVariantConverter)
{
    index.Add(sequence);

    var optimalVariant = optimalVariantConverter.Convert(sequence);

    var readSequence1 = sequences.ToList(optimalVariant);

    Assert.True(sequence.SequenceEqual(readSequence1));
}

[Fact]
public static void SavedSequencesOptimizationTest()
{
    LinksConstants<ulong> constants = new LinksConstants<ulong>((1, long.MaxValue),
        ↪ (long.MaxValue + 1UL, ulong.MaxValue));

    using (var memory = new HeapResizableDirectMemory())

```

```

153 using (var disposableLinks = new UInt64ResizableDirectMemoryLinks(memory,
    ↳ UInt64ResizableDirectMemoryLinks.DefaultLinksSizeStep, constants,
    ↳ useAvlBasedIndex: false))
154 {
155     var links = new UInt64Links(disposableLinks);
156
157     var root = links.CreatePoint();
158
159     //var numberToAddressConverter = new RawNumberToAddressConverter<ulong>();
160     var addressToNumberConverter = new AddressToRawNumberConverter<ulong>();
161
162     var unicodeSymbolMarker = links.GetOrCreate(root,
    ↳ addressToNumberConverter.Convert(1));
163     var unicodeSequenceMarker = links.GetOrCreate(root,
    ↳ addressToNumberConverter.Convert(2));
164
165     var totalSequenceSymbolFrequencyCounter = new
    ↳ TotalSequenceSymbolFrequencyCounter<ulong>(links);
166     var linkFrequenciesCache = new LinkFrequenciesCache<ulong>(links,
    ↳ totalSequenceSymbolFrequencyCounter);
167     var index = new
    ↳ CachedFrequencyIncrementingSequenceIndex<ulong>(linkFrequenciesCache);
168     var linkToItsFrequencyNumberConverter = new FrequenciesCacheBasedLinkToItsFreque
    ↳ ncyNumberConverter<ulong>(linkFrequenciesCache);
169     var sequenceToItsLocalElementLevelsConverter = new
    ↳ SequenceToItsLocalElementLevelsConverter<ulong>(links,
    ↳ linkToItsFrequencyNumberConverter);
170     var optimalVariantConverter = new OptimalVariantConverter<ulong>(links,
    ↳ sequenceToItsLocalElementLevelsConverter);
171
172     var walker = new RightSequenceWalker<ulong>(links, new DefaultStack<ulong>(),
    ↳ (link) => constants.IsExternalReference(link) || links.IsPartialPoint(link));
173
174     var unicodeSequencesOptions = new SequencesOptions<ulong>()
175     {
176         UseSequenceMarker = true,
177         SequenceMarkerLink = unicodeSequenceMarker,
178         UseIndex = true,
179         Index = index,
180         LinksToSequenceConverter = optimalVariantConverter,
181         Walker = walker,
182         UseGarbageCollection = true
183     };
184
185     var unicodeSequences = new Sequences.Sequences(new
    ↳ SynchronizedLinks<ulong>(links), unicodeSequencesOptions);
186
187     // Create some sequences
188     var strings = _loremIpsumExample.Split(new[] { '\n', '\r' },
    ↳ StringSplitOptions.RemoveEmptyEntries);
189     var arrays = strings.Select(x => x.Select(y =>
    ↳ addressToNumberConverter.Convert(y)).ToArray()).ToArray();
190     for (int i = 0; i < arrays.Length; i++)
191     {
192         unicodeSequences.Create(arrays[i].ShiftRight());
193     }
194
195     var linksCountAfterCreation = links.Count();
196
197     // get list of sequences links
198     // for each sequence link
199     // create new sequence version
200     // if new sequence is not the same as sequence link
201     // delete sequence link
202     // collect garbage
203     unicodeSequences.CompactAll();
204
205     var linksCountAfterCompactification = links.Count();
206
207     Assert.True(linksCountAfterCompactification < linksCountAfterCreation);
208 }
209 }
210 }
211 }

```

1.107 ./Platform.Data.Doublets.Tests/ReadSequenceTests.cs

```

1 using System;
2 using System.Collections.Generic;
3 using System.Diagnostics;

```

```

4 using System.Linq;
5 using Xunit;
6 using Platform.Data.Sequences;
7 using Platform.Data.Doublets.Sequences.Converters;
8 using Platform.Data.Doublets.Sequences.Walkers;
9 using Platform.Data.Doublets.Sequences;
10
11 namespace Platform.Data.Doublets.Tests
12 {
13     public static class ReadSequenceTests
14     {
15         [Fact]
16         public static void ReadSequenceTest()
17         {
18             const long sequenceLength = 2000;
19
20             using (var scope = new TempLinksTestScope(useSequences: false))
21             {
22                 var links = scope.Links;
23                 var sequences = new Sequences.Sequences(links, new SequencesOptions

```

1.108 ./Platform.Data.Doublets.Tests/ResizableDirectMemoryLinksTests.cs

```

1 using System.IO;
2 using Xunit;
3 using Platform.Singletons;
4 using Platform.Memory;
5 using Platform.Data.Doublets.ResizableDirectMemory.Specific;
6
7 namespace Platform.Data.Doublets.Tests
8 {
9     public static class ResizableDirectMemoryLinksTests
10     {
11         private static readonly LinksConstants

```

```

16     var tempFilename = Path.GetTempFileName();
17     using (var memoryAdapter = new UInt64ResizableDirectMemoryLinks(tempFilename))
18     {
19         memoryAdapter.TestBasicMemoryOperations();
20     }
21     File.Delete(tempFilename);
22 }
23
24 [Fact]
25 public static void BasicHeapMemoryTest()
26 {
27     using (var memory = new
28         ↪ HeapResizableDirectMemory(UInt64ResizableDirectMemoryLinks.DefaultLinksSizeStep))
29     using (var memoryAdapter = new UInt64ResizableDirectMemoryLinks(memory,
30         ↪ UInt64ResizableDirectMemoryLinks.DefaultLinksSizeStep))
31     {
32         memoryAdapter.TestBasicMemoryOperations();
33     }
34 }
35
36 private static void TestBasicMemoryOperations(this ILinks<ulong> memoryAdapter)
37 {
38     var link = memoryAdapter.Create();
39     memoryAdapter.Delete(link);
40 }
41
42 [Fact]
43 public static void NonexistentReferencesHeapMemoryTest()
44 {
45     using (var memory = new
46         ↪ HeapResizableDirectMemory(UInt64ResizableDirectMemoryLinks.DefaultLinksSizeStep))
47     using (var memoryAdapter = new UInt64ResizableDirectMemoryLinks(memory,
48         ↪ UInt64ResizableDirectMemoryLinks.DefaultLinksSizeStep))
49     {
50         memoryAdapter.TestNonexistentReferences();
51     }
52 }
53
54 private static void TestNonexistentReferences(this ILinks<ulong> memoryAdapter)
55 {
56     var link = memoryAdapter.Create();
57     memoryAdapter.Update(link, ulong.MaxValue, ulong.MaxValue);
58     var resultLink = _constants.Null;
59     memoryAdapter.Each(foundLink =>
60     {
61         resultLink = foundLink[_constants.IndexPart];
62         return _constants.Break;
63     }, _constants.Any, ulong.MaxValue, ulong.MaxValue);
64     Assert.True(resultLink == link);
65     Assert.True(memoryAdapter.Count(ulong.MaxValue) == 0);
66     memoryAdapter.Delete(link);
67 }
68 }
69 }

```

1.109 ./Platform.Data.Doublets.Tests/ScopeTests.cs

```

1  using Xunit;
2  using Platform.Scopes;
3  using Platform.Memory;
4  using Platform.Data.Doublets.Decorators;
5  using Platform.Reflection;
6  using Platform.Data.Doublets.ResizableDirectMemory.Generic;
7  using Platform.Data.Doublets.ResizableDirectMemory.Specific;
8
9  namespace Platform.Data.Doublets.Tests
10 {
11     public static class ScopeTests
12     {
13         [Fact]
14         public static void SingleDependencyTest()
15         {
16             using (var scope = new Scope())
17             {
18                 scope.IncludeAssemblyOf<IMemory>();
19                 var instance = scope.Use<IDirectMemory>();
20                 Assert.IsType<HeapResizableDirectMemory>(instance);
21             }
22         }
23     }
24 }

```

```

24 [Fact]
25 public static void CascadeDependencyTest()
26 {
27     using (var scope = new Scope())
28     {
29         scope.Include<TemporaryFileMappedResizableDirectMemory>();
30         scope.Include<UInt64ResizableDirectMemoryLinks>();
31         var instance = scope.Use<ILinks<ulong>>();
32         Assert.IsType<UInt64ResizableDirectMemoryLinks>(instance);
33     }
34 }
35
36 [Fact]
37 public static void FullAutoResolutionTest()
38 {
39     using (var scope = new Scope(autoInclude: true, autoExplore: true))
40     {
41         var instance = scope.Use<UInt64Links>();
42         Assert.IsType<UInt64Links>(instance);
43     }
44 }
45
46 [Fact]
47 public static void TypeParametersTest()
48 {
49     using (var scope = new Scope<Types<HeapResizableDirectMemory,
50 ↪ ResizableDirectMemoryLinks<ulong>>>())
51     {
52         var links = scope.Use<ILinks<ulong>>();
53         Assert.IsType<ResizableDirectMemoryLinks<ulong>>(links);
54     }
55 }
56 }

```

1.110 ./Platform.Data.Doublets.Tests/SequencesTests.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Diagnostics;
4  using System.Linq;
5  using Xunit;
6  using Platform.Collections;
7  using Platform.Collections.Arrays;
8  using Platform.Random;
9  using Platform.IO;
10 using Platform.Singletons;
11 using Platform.Data.Doublets.Sequences;
12 using Platform.Data.Doublets.Sequences.Frequencies.Cache;
13 using Platform.Data.Doublets.Sequences.Frequencies.Counters;
14 using Platform.Data.Doublets.Sequences.Converters;
15 using Platform.Data.Doublets.Unicode;
16
17 namespace Platform.Data.Doublets.Tests
18 {
19     public static class SequencesTests
20     {
21         private static readonly LinksConstants<ulong> _constants =
22             ↪ Default<LinksConstants<ulong>>.Instance;
23
24         static SequencesTests()
25         {
26             // Trigger static constructor to not mess with performance measurements
27             _ = BitString.GetBitMaskFromIndex(1);
28         }
29
30         [Fact]
31         public static void CreateAllVariantsTest()
32         {
33             const long sequenceLength = 8;
34
35             using (var scope = new TempLinksTestScope(useSequences: true))
36             {
37                 var links = scope.Links;
38                 var sequences = scope.Sequences;
39
40                 var sequence = new ulong[sequenceLength];
41                 for (var i = 0; i < sequenceLength; i++)
42                 {
43                     sequence[i] = links.Create();
44                 }
45             }
46         }
47     }
48 }

```

```

44
45     var sw1 = Stopwatch.StartNew();
46     var results1 = sequences.CreateAllVariants1(sequence); sw1.Stop();
47
48     var sw2 = Stopwatch.StartNew();
49     var results2 = sequences.CreateAllVariants2(sequence); sw2.Stop();
50
51     Assert.True(results1.Count > results2.Length);
52     Assert.True(sw1.Elapsed > sw2.Elapsed);
53
54     for (var i = 0; i < sequenceLength; i++)
55     {
56         links.Delete(sequence[i]);
57     }
58
59     Assert.True(links.Count() == 0);
60 }
61 }
62
63 // [Fact]
64 // public void CUDTest()
65 // {
66 //     var tempFilename = Path.GetTempFileName();
67
68 //     const long sequenceLength = 8;
69
70 //     const ulong itself = LinksConstants.Itself;
71
72 //     using (var memoryAdapter = new ResizableDirectMemoryLinks(tempFilename,
73 //         ↪ DefaultLinksSizeStep))
74 //     using (var links = new Links(memoryAdapter))
75 //     {
76 //         var sequence = new ulong[sequenceLength];
77 //         for (var i = 0; i < sequenceLength; i++)
78 //             sequence[i] = links.Create(itself, itself);
79
80 //         SequencesOptions o = new SequencesOptions();
81
82 //         // TODO: Из числа в bool значения o.UseSequenceMarker = ((value & 1) != 0)
83 //         o.
84
85 //         var sequences = new Sequences(links);
86
87 //         var sw1 = Stopwatch.StartNew();
88 //         var results1 = sequences.CreateAllVariants1(sequence); sw1.Stop();
89
90 //         var sw2 = Stopwatch.StartNew();
91 //         var results2 = sequences.CreateAllVariants2(sequence); sw2.Stop();
92
93 //         Assert.True(results1.Count > results2.Length);
94 //         Assert.True(sw1.Elapsed > sw2.Elapsed);
95
96 //         for (var i = 0; i < sequenceLength; i++)
97 //             links.Delete(sequence[i]);
98 //     }
99
100 //     File.Delete(tempFilename);
101 // }
102
103 [Fact]
104 public static void AllVariantsSearchTest()
105 {
106     const long sequenceLength = 8;
107
108     using (var scope = new TempLinksTestScope(useSequences: true))
109     {
110         var links = scope.Links;
111         var sequences = scope.Sequences;
112
113         var sequence = new ulong[sequenceLength];
114         for (var i = 0; i < sequenceLength; i++)
115         {
116             sequence[i] = links.Create();
117         }
118
119         var createResults = sequences.CreateAllVariants2(sequence).Distinct().ToArray();
120
121         // for (int i = 0; i < createResults.Length; i++)
122         //     sequences.Create(createResults[i]);

```

```

123     var sw0 = Stopwatch.StartNew();
124     var searchResults0 = sequences.GetAllMatchingSequences0(sequence); sw0.Stop();
125
126     var sw1 = Stopwatch.StartNew();
127     var searchResults1 = sequences.GetAllMatchingSequences1(sequence); sw1.Stop();
128
129     var sw2 = Stopwatch.StartNew();
130     var searchResults2 = sequences.Each1(sequence); sw2.Stop();
131
132     var sw3 = Stopwatch.StartNew();
133     var searchResults3 = sequences.Each(sequence.ShiftRight()); sw3.Stop();
134
135     var intersection0 = createResults.Intersect(searchResults0).ToList();
136     Assert.True(intersection0.Count == searchResults0.Count);
137     Assert.True(intersection0.Count == createResults.Length);
138
139     var intersection1 = createResults.Intersect(searchResults1).ToList();
140     Assert.True(intersection1.Count == searchResults1.Count);
141     Assert.True(intersection1.Count == createResults.Length);
142
143     var intersection2 = createResults.Intersect(searchResults2).ToList();
144     Assert.True(intersection2.Count == searchResults2.Count);
145     Assert.True(intersection2.Count == createResults.Length);
146
147     var intersection3 = createResults.Intersect(searchResults3).ToList();
148     Assert.True(intersection3.Count == searchResults3.Count);
149     Assert.True(intersection3.Count == createResults.Length);
150
151     for (var i = 0; i < sequenceLength; i++)
152     {
153         links.Delete(sequence[i]);
154     }
155 }
156
157 [Fact]
158 public static void BalancedVariantSearchTest()
159 {
160     const long sequenceLength = 200;
161
162     using (var scope = new TempLinksTestScope(useSequences: true))
163     {
164         var links = scope.Links;
165         var sequences = scope.Sequences;
166
167         var sequence = new ulong[sequenceLength];
168         for (var i = 0; i < sequenceLength; i++)
169         {
170             sequence[i] = links.Create();
171         }
172
173         var balancedVariantConverter = new BalancedVariantConverter<ulong>(links);
174
175         var sw1 = Stopwatch.StartNew();
176         var balancedVariant = balancedVariantConverter.Convert(sequence); sw1.Stop();
177
178         var sw2 = Stopwatch.StartNew();
179         var searchResults2 = sequences.GetAllMatchingSequences0(sequence); sw2.Stop();
180
181         var sw3 = Stopwatch.StartNew();
182         var searchResults3 = sequences.GetAllMatchingSequences1(sequence); sw3.Stop();
183
184         // На количестве в 200 элементов это будет занимать вечность
185         //var sw4 = Stopwatch.StartNew();
186         //var searchResults4 = sequences.Each(sequence); sw4.Stop();
187
188         Assert.True(searchResults2.Count == 1 && balancedVariant == searchResults2[0]);
189
190         Assert.True(searchResults3.Count == 1 && balancedVariant ==
191             ↪ searchResults3.First());
192
193         //Assert.True(sw1.Elapsed < sw2.Elapsed);
194
195         for (var i = 0; i < sequenceLength; i++)
196         {
197             links.Delete(sequence[i]);
198         }
199     }
200 }
201

```

```

202 [Fact]
203 public static void AllPartialVariantsSearchTest()
204 {
205     const long sequenceLength = 8;
206
207     using (var scope = new TempLinksTestScope(useSequences: true))
208     {
209         var links = scope.Links;
210         var sequences = scope.Sequences;
211
212         var sequence = new ulong[sequenceLength];
213         for (var i = 0; i < sequenceLength; i++)
214         {
215             sequence[i] = links.Create();
216         }
217
218         var createResults = sequences.CreateAllVariants2(sequence);
219
220         //var createResultsStrings = createResults.Select(x => x + ": " +
221         ↪ sequences.FormatSequence(x)).ToList();
222         //Global.Trash = createResultsStrings;
223
224         var partialSequence = new ulong[sequenceLength - 2];
225         Array.Copy(sequence, 1, partialSequence, 0, (int)sequenceLength - 2);
226
227         var sw1 = Stopwatch.StartNew();
228         var searchResults1 =
229         ↪ sequences.GetAllPartiallyMatchingSequences0(partialSequence); sw1.Stop();
230
231         var sw2 = Stopwatch.StartNew();
232         var searchResults2 =
233         ↪ sequences.GetAllPartiallyMatchingSequences1(partialSequence); sw2.Stop();
234
235         //var sw3 = Stopwatch.StartNew();
236         //var searchResults3 =
237         ↪ sequences.GetAllPartiallyMatchingSequences2(partialSequence); sw3.Stop();
238
239         var sw4 = Stopwatch.StartNew();
240         var searchResults4 =
241         ↪ sequences.GetAllPartiallyMatchingSequences3(partialSequence); sw4.Stop();
242
243         //Global.Trash = searchResults3;
244
245         //var searchResults1Strings = searchResults1.Select(x => x + ": " +
246         ↪ sequences.FormatSequence(x)).ToList();
247         //Global.Trash = searchResults1Strings;
248
249         var intersection1 = createResults.Intersect(searchResults1).ToList();
250         Assert.True(intersection1.Count == createResults.Length);
251
252         var intersection2 = createResults.Intersect(searchResults2).ToList();
253         Assert.True(intersection2.Count == createResults.Length);
254
255         var intersection4 = createResults.Intersect(searchResults4).ToList();
256         Assert.True(intersection4.Count == createResults.Length);
257
258         for (var i = 0; i < sequenceLength; i++)
259         {
260             links.Delete(sequence[i]);
261         }
262     }
263 }
264
265 [Fact]
266 public static void BalancedPartialVariantsSearchTest()
267 {
268     const long sequenceLength = 200;
269
270     using (var scope = new TempLinksTestScope(useSequences: true))
271     {
272         var links = scope.Links;
273         var sequences = scope.Sequences;
274
275         var sequence = new ulong[sequenceLength];
276         for (var i = 0; i < sequenceLength; i++)
277         {
278             sequence[i] = links.Create();
279         }
280     }
281 }

```



```

276     var balancedVariantConverter = new BalancedVariantConverter<ulong>(links);
277
278     var balancedVariant = balancedVariantConverter.Convert(sequence);
279
280     var partialSequence = new ulong[sequenceLength - 2];
281
282     Array.Copy(sequence, 1, partialSequence, 0, (int)sequenceLength - 2);
283
284     var sw1 = Stopwatch.StartNew();
285     var searchResults1 =
286         ↪ sequences.GetAllPartiallyMatchingSequences0(partialSequence); sw1.Stop();
287
288     var sw2 = Stopwatch.StartNew();
289     var searchResults2 =
290         ↪ sequences.GetAllPartiallyMatchingSequences1(partialSequence); sw2.Stop();
291
292     Assert.True(searchResults1.Count == 1 && balancedVariant == searchResults1[0]);
293
294     Assert.True(searchResults2.Count == 1 && balancedVariant ==
295         ↪ searchResults2.First());
296
297     for (var i = 0; i < sequenceLength; i++)
298     {
299         links.Delete(sequence[i]);
300     }
301 }
302
303 [Fact(Skip = "Correct implementation is pending")]
304 public static void PatternMatchTest()
305 {
306     var zeroOrMany = Sequences.Sequences.ZeroOrMany;
307
308     using (var scope = new TempLinksTestScope(useSequences: true))
309     {
310         var links = scope.Links;
311         var sequences = scope.Sequences;
312
313         var e1 = links.Create();
314         var e2 = links.Create();
315
316         var sequence = new[]
317         {
318             e1, e2, e1, e2 // mama / papa
319         };
320
321         var balancedVariantConverter = new BalancedVariantConverter<ulong>(links);
322
323         var balancedVariant = balancedVariantConverter.Convert(sequence);
324
325         // 1: [1]
326         // 2: [2]
327         // 3: [1,2]
328         // 4: [1,2,1,2]
329
330         var doublet = links.GetSource(balancedVariant);
331
332         var matchedSequences1 = sequences.MatchPattern(e2, e1, zeroOrMany);
333
334         Assert.True(matchedSequences1.Count == 0);
335
336         var matchedSequences2 = sequences.MatchPattern(zeroOrMany, e2, e1);
337
338         Assert.True(matchedSequences2.Count == 0);
339
340         var matchedSequences3 = sequences.MatchPattern(e1, zeroOrMany, e1);
341
342         Assert.True(matchedSequences3.Count == 0);
343
344         var matchedSequences4 = sequences.MatchPattern(e1, zeroOrMany, e2);
345
346         Assert.Contains(doublet, matchedSequences4);
347         Assert.Contains(balancedVariant, matchedSequences4);
348
349         for (var i = 0; i < sequence.Length; i++)
350         {
351             links.Delete(sequence[i]);
352         }
353     }
354 }

```

```

353 [Fact]
354 public static void IndexTest()
355 {
356     using (var scope = new TempLinksTestScope(new SequencesOptions<ulong> { UseIndex =
357         ↳ true }, useSequences: true))
358     {
359         var links = scope.Links;
360         var sequences = scope.Sequences;
361         var index = sequences.Options.Index;
362
363         var e1 = links.Create();
364         var e2 = links.Create();
365
366         var sequence = new[]
367         {
368             e1, e2, e1, e2 // mama / papa
369         };
370
371         Assert.False(index.MightContain(sequence));
372
373         index.Add(sequence);
374
375         Assert.True(index.MightContain(sequence));
376     }
377 }
378
379 /// <summary>Imported from https://raw.githubusercontent.com/Konard/LinksPlatform/%
380 ↳ DO%9E-%D1%82%D0%BE%D0%BC%2C-%D0%BA%D0%B0%D0%BA-%D0%B2%D1%81%D1%91-%D0%BD%D0%B0%D1%87
381 ↳ %D0%B8%D0%BD%D0%B0%D0%BB%D0%BE%D1%81%D1%8C.md</summary>
382 private static readonly string _exampleText =
383     @"([english
384     ↳ version] (https://github.com/Konard/LinksPlatform/wiki/About-the-beginning))
385
386 Обозначение пустоты, какое оно? Темнота ли это? Там где отсутствие света, отсутствие фотонов
387 ↳ (носителей света)? Или это то, что полностью отражает свет? Пустой белый лист бумаги? Там
388 ↳ где есть место для нового начала? Разве пустота это не характеристика пространства?
389 ↳ Пространство это то, что можно чем-то наполнить?
390
391 ![чёрное пространство, белое
392 ↳ пространство] (https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/1.png
393 ↳ "чёрное пространство, белое пространство") (https://raw.githubusercontent.com/Konard/Links
394 ↳ Platform/master/doc/Intro/1.png)
395
396 Что может быть минимальным рисунком, образом, графикой? Может быть это точка? Это ли простейшая
397 ↳ форма? Но есть ли у точки размер? Цвет? Масса? Координаты? Время существования?
398
399 ![чёрное пространство, чёрная
400 ↳ точка] (https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/2.png
401 ↳ "чёрное пространство, чёрная
402 ↳ точка") (https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/2.png)
403
404 А что если повторить? Сделать копию? Создать дубликат? Из одного сделать два? Может это быть
405 ↳ так? Инверсия? Отражение? Сумма?
406
407 ![белая точка, чёрная
408 ↳ точка] (https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/3.png "белая
409 ↳ точка, чёрная
410 ↳ точка") (https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/3.png)
411
412 А что если мы вообразим движение? Нужно ли время? Каким самым коротким будет путь? Что будет
413 ↳ если этот путь зафиксировать? Запомнить след? Как две точки становятся линией? Чертой?
414 ↳ Гранью? Разделителем? Единицей?
415
416 ![две белые точки, чёрная вертикальная
417 ↳ линия] (https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/4.png "две
418 ↳ белые точки, чёрная вертикальная
419 ↳ линия") (https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/4.png)
420
421 Можно ли замкнуть движение? Может ли это быть кругом? Можно ли замкнуть время? Или остаётся
422 ↳ только спираль? Но что если замкнуть предел? Создать ограничение, разделение? Получится
423 ↳ замкнутая область? Полностью отделённая от всего остального? Но что это всё остальное? Что
424 ↳ можно делить? В каком направлении? Ничего или всё? Пустота или полнота? Начало или конец?
425 ↳ Или может быть это единица и ноль? Дуальность? Противоположность? А что будет с кругом если
426 ↳ у него нет размера? Будет ли круг точкой? Точка состоящая из точек?
427
428 ![белая вертикальная линия, чёрный
429 ↳ круг] (https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/5.png "белая
430 ↳ вертикальная линия, чёрный
431 ↳ круг") (https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/5.png)

```

402
403 Как ещё можно использовать грань, черту, линию? А что если она может что-то соединять, может
→ тогда её нужно повернуть? Почему то, что перпендикулярно вертикальному горизонтально?
→ Горизонт? Инвертирует ли это смысл? Что такое смысл? Из чего состоит смысл? Существует ли
→ элементарная единица смысла?

404
405 `[[белый круг, чёрная горизонтальная`
→ `линия](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/6.png` `"белый`
→ `круг, чёрная горизонтальная`
→ `линия")](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/6.png)`

406
407 Соединять, допустим, а какой смысл в этом есть ещё? Что если помимо смысла "соединить,
→ связать", есть ещё и смысл направления "от начала к концу"? От предка к потомку? От
→ родителя к ребёнку? От общего к частному?

408
409 `[[белая горизонтальная линия, чёрная горизонтальная`
→ `стрелка](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/7.png`
→ `"белая горизонтальная линия, чёрная горизонтальная`
→ `стрелка")](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/7.png)`

410
411 Шаг назад. Возьмём опять отделённую область, которая лишь та же замкнутая линия, что ещё она
→ может представлять собой? Объект? Но в чём его суть? Разве не в том, что у него есть
→ граница, разделяющая внутреннее и внешнее? Допустим связь, стрелка, линия соединяет два
→ объекта, как бы это выглядело?

412
413 `[[белая связь, чёрная направленная`
→ `связь](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/8.png` `"белая`
→ `связь, чёрная направленная`
→ `связь")](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/8.png)`

414
415 Допустим у нас есть смысл "связать" и смысл "направления", много ли это нам даёт? Много ли
→ вариантов интерпретации? А что если уточнить, каким именно образом выполнена связь? Что если
→ можно задать ей чёткий, конкретный смысл? Что это будет? Тип? Глагол? Связка? Действие?
→ Трансформация? Переход из состояния в состояние? Или всё это и есть объект, суть которого в
→ его конечном состоянии, если конечно конец определён направлением?

416
417 `[[белая обычная и направленная связи, чёрная типизированная`
→ `связь](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/9.png` `"белая`
→ `обычная и направленная связи, чёрная типизированная`
→ `связь")](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/9.png)`

418
419 А что если всё это время, мы смотрели на суть как бы снаружи? Можно ли взглянуть на это изнутри?
→ Что будет внутри объектов? Объекты ли это? Или это связи? Может ли эта структура описать
→ сама себя? Но что тогда получится, разве это не рекурсия? Может это фрактал?

420
421 `[[белая обычная и направленная связи с рекурсивной внутренней структурой, чёрная типизированная`
→ `связь с рекурсивной внутренней`
→ `структурой](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/10.png`
→ `"белая обычная и направленная связи с рекурсивной внутренней структурой, чёрная`
→ `типизированная связь с рекурсивной внутренней структурой")](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/10.png)`

422
423 На один уровень внутрь (вниз)? Или на один уровень во вне (вверх)? Или это можно назвать шагом
→ рекурсии или фрактала?

424
425 `[[белая обычная и направленная связи с двойной рекурсивной внутренней структурой, чёрная`
→ `типизированная связь с двойной рекурсивной внутренней`
→ `структурой](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/11.png`
→ `"белая обычная и направленная связи с двойной рекурсивной внутренней структурой, чёрная`
→ `типизированная связь с двойной рекурсивной внутренней структурой")](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/11.png)`

426
427 Последовательность? Массив? Список? Множество? Объект? Таблица? Элементы? Цвета? Символы? Буквы?
→ Слово? Цифры? Число? Алфавит? Дерево? Сеть? Граф? Гиперграф?

428
429 `[[белая обычная и направленная связи со структурой из 8 цветных элементов последовательности,`
→ `чёрная типизированная связь со структурой из 8 цветных элементов последовательности](https://`
→ `/raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/12.png` `"белая обычная и`
→ `направленная связи со структурой из 8 цветных элементов последовательности, чёрная`
→ `типизированная связь со структурой из 8 цветных элементов последовательности")](https://raw`
→ `.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/12.png)`

430
431 ...

432
433 `[[анимация](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/intro-anim`
→ `ation-500.gif`
→ `"анимация")](https://raw.githubusercontent.com/Konard/LinksPlatform/master/doc/Intro/intro`
→ `-animation-500.gif)";`

434

```

435     private static readonly string _exampleLoremIpsumText =
436         @"Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor
        ↳ incididunt ut labore et dolore magna aliqua.
437     Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
        ↳ consequat.";
438
439     [Fact]
440     public static void CompressionTest()
441     {
442         using (var scope = new TempLinksTestScope(useSequences: true))
443         {
444             var links = scope.Links;
445             var sequences = scope.Sequences;
446
447             var e1 = links.Create();
448             var e2 = links.Create();
449
450             var sequence = new[]
451             {
452                 e1, e2, e1, e2 // mama / papa / template [(m/p), a] { [1] [2] [1] [2] }
453             };
454
455             var balancedVariantConverter = new BalancedVariantConverter<ulong>(links.Unsync);
456             var totalSequenceSymbolFrequencyCounter = new
457                 ↳ TotalSequenceSymbolFrequencyCounter<ulong>(links.Unsync);
458             var doubletFrequenciesCache = new LinkFrequenciesCache<ulong>(links.Unsync,
459                 ↳ totalSequenceSymbolFrequencyCounter);
460             var compressingConverter = new CompressingConverter<ulong>(links.Unsync,
461                 ↳ balancedVariantConverter, doubletFrequenciesCache);
462
463             var compressedVariant = compressingConverter.Convert(sequence);
464
465             // 1: [1]          (1->1) point
466             // 2: [2]          (2->2) point
467             // 3: [1,2]        (1->2) doublet
468             // 4: [1,2,1,2]    (3->3) doublet
469
470             Assert.True(links.GetSource(links.GetSource(compressedVariant)) == sequence[0]);
471             Assert.True(links.GetTarget(links.GetSource(compressedVariant)) == sequence[1]);
472             Assert.True(links.GetSource(links.GetTarget(compressedVariant)) == sequence[2]);
473             Assert.True(links.GetTarget(links.GetTarget(compressedVariant)) == sequence[3]);
474
475             var source = _constants.SourcePart;
476             var target = _constants.TargetPart;
477
478             Assert.True(links.GetByKeys(compressedVariant, source, source) == sequence[0]);
479             Assert.True(links.GetByKeys(compressedVariant, source, target) == sequence[1]);
480             Assert.True(links.GetByKeys(compressedVariant, target, source) == sequence[2]);
481             Assert.True(links.GetByKeys(compressedVariant, target, target) == sequence[3]);
482
483             // 4 - length of sequence
484             Assert.True(links.GetSquareMatrixSequenceElementByIndex(compressedVariant, 4, 0)
485                 ↳ == sequence[0]);
486             Assert.True(links.GetSquareMatrixSequenceElementByIndex(compressedVariant, 4, 1)
487                 ↳ == sequence[1]);
488             Assert.True(links.GetSquareMatrixSequenceElementByIndex(compressedVariant, 4, 2)
489                 ↳ == sequence[2]);
490             Assert.True(links.GetSquareMatrixSequenceElementByIndex(compressedVariant, 4, 3)
491                 ↳ == sequence[3]);
492         }
493     }
494
495     [Fact]
496     public static void CompressionEfficiencyTest()
497     {
498         var strings = _exampleLoremIpsumText.Split(new[] { '\n', '\r' },
499             ↳ StringSplitOptions.RemoveEmptyEntries);
500         var arrays = strings.Select(UnicodeMap.FromStringToLinkArray).ToArray();
501         var totalCharacters = arrays.Select(x => x.Length).Sum();
502
503         using (var scope1 = new TempLinksTestScope(useSequences: true))
504         using (var scope2 = new TempLinksTestScope(useSequences: true))
505         using (var scope3 = new TempLinksTestScope(useSequences: true))
506         {
507             scope1.Links.Unsync.UseUnicode();
508             scope2.Links.Unsync.UseUnicode();
509             scope3.Links.Unsync.UseUnicode();
510         }
511     }

```

```

503     var balancedVariantConverter1 = new
504         ↳ BalancedVariantConverter<ulong>(scope1.Links.Unsync);
505     var totalSequenceSymbolFrequencyCounter = new
506         ↳ TotalSequenceSymbolFrequencyCounter<ulong>(scope1.Links.Unsync);
507     var linkFrequenciesCache1 = new LinkFrequenciesCache<ulong>(scope1.Links.Unsync,
508         ↳ totalSequenceSymbolFrequencyCounter);
509     var compressor1 = new CompressingConverter<ulong>(scope1.Links.Unsync,
510         ↳ balancedVariantConverter1, linkFrequenciesCache1,
511         ↳ doInitialFrequenciesIncrement: false);
512
513     //var compressor2 = scope2.Sequences;
514     var compressor3 = scope3.Sequences;
515
516     var constants = Default<LinksConstants<ulong>>.Instance;
517
518     var sequences = compressor3;
519     //var meaningRoot = links.CreatePoint();
520     //var unaryOne = links.CreateAndUpdate(meaningRoot, constants.Itself);
521     //var frequencyMarker = links.CreateAndUpdate(meaningRoot, constants.Itself);
522     //var frequencyPropertyMarker = links.CreateAndUpdate(meaningRoot,
523         ↳ constants.Itself);
524
525     //var unaryNumberToAddressConverter = new
526         ↳ UnaryNumberToAddressAddOperationConverter<ulong>(links, unaryOne);
527     //var unaryNumberIncrementer = new UnaryNumberIncrementer<ulong>(links,
528         ↳ unaryOne);
529     //var frequencyIncrementer = new FrequencyIncrementer<ulong>(links,
530         ↳ frequencyMarker, unaryOne, unaryNumberIncrementer);
531     //var frequencyPropertyOperator = new FrequencyPropertyOperator<ulong>(links,
532         ↳ frequencyPropertyMarker, frequencyMarker);
533     //var linkFrequencyIncrementer = new LinkFrequencyIncrementer<ulong>(links,
534         ↳ frequencyPropertyOperator, frequencyIncrementer);
535     //var linkToItsFrequencyNumberConverter = new
536         ↳ LinkToItsFrequencyNumberConveter<ulong>(links, frequencyPropertyOperator,
537         ↳ unaryNumberToAddressConverter);
538
539     var linkFrequenciesCache3 = new LinkFrequenciesCache<ulong>(scope3.Links.Unsync,
540         ↳ totalSequenceSymbolFrequencyCounter);
541
542     var linkToItsFrequencyNumberConverter = new FrequenciesCacheBasedLinkToItsFrequencyNumberConverter<ulong>(linkFrequenciesCache3);
543
544     var sequenceToItsLocalElementLevelsConverter = new
545         ↳ SequenceToItsLocalElementLevelsConverter<ulong>(scope3.Links.Unsync,
546         ↳ linkToItsFrequencyNumberConverter);
547     var optimalVariantConverter = new
548         ↳ OptimalVariantConverter<ulong>(scope3.Links.Unsync,
549         ↳ sequenceToItsLocalElementLevelsConverter);
550
551     var compressed1 = new ulong[arrays.Length];
552     var compressed2 = new ulong[arrays.Length];
553     var compressed3 = new ulong[arrays.Length];
554
555     var START = 0;
556     var END = arrays.Length;
557
558     //for (int i = START; i < END; i++)
559     //    linkFrequenciesCache1.IncrementFrequencies(arrays[i]);
560
561     var initialCount1 = scope2.Links.Unsync.Count();
562
563     var sw1 = Stopwatch.StartNew();
564
565     for (int i = START; i < END; i++)
566     {
567         linkFrequenciesCache1.IncrementFrequencies(arrays[i]);
568         compressed1[i] = compressor1.Convert(arrays[i]);
569     }
570
571     var elapsed1 = sw1.Elapsed;
572
573     var balancedVariantConverter2 = new
574         ↳ BalancedVariantConverter<ulong>(scope2.Links.Unsync);
575
576     var initialCount2 = scope2.Links.Unsync.Count();
577
578     var sw2 = Stopwatch.StartNew();

```

```

561     for (int i = START; i < END; i++)
562     {
563         compressed2[i] = balancedVariantConverter2.Convert(arrays[i]);
564     }
565
566     var elapsed2 = sw2.Elapsed;
567
568     for (int i = START; i < END; i++)
569     {
570         linkFrequenciesCache3.IncrementFrequencies(arrays[i]);
571     }
572
573     var initialCount3 = scope3.Links.Unsync.Count();
574
575     var sw3 = Stopwatch.StartNew();
576
577     for (int i = START; i < END; i++)
578     {
579         //linkFrequenciesCache3.IncrementFrequencies(arrays[i]);
580         compressed3[i] = optimalVariantConverter.Convert(arrays[i]);
581     }
582
583     var elapsed3 = sw3.Elapsed;
584
585     Console.WriteLine($"Compressor: {elapsed1}, Balanced variant: {elapsed2},
586         ↳ Optimal variant: {elapsed3}");
587
588     // Assert.True(elapsed1 > elapsed2);
589
590     // Checks
591     for (int i = START; i < END; i++)
592     {
593         var sequence1 = compressed1[i];
594         var sequence2 = compressed2[i];
595         var sequence3 = compressed3[i];
596
597         var decompress1 = UnicodeMap.FromSequenceLinkToString(sequence1,
598             ↳ scope1.Links.Unsync);
599
600         var decompress2 = UnicodeMap.FromSequenceLinkToString(sequence2,
601             ↳ scope2.Links.Unsync);
602
603         var decompress3 = UnicodeMap.FromSequenceLinkToString(sequence3,
604             ↳ scope3.Links.Unsync);
605
606         var structure1 = scope1.Links.Unsync.FormatStructure(sequence1, link =>
607             ↳ link.IsPartialPoint());
608         var structure2 = scope2.Links.Unsync.FormatStructure(sequence2, link =>
609             ↳ link.IsPartialPoint());
610         var structure3 = scope3.Links.Unsync.FormatStructure(sequence3, link =>
611             ↳ link.IsPartialPoint());
612
613         //if (sequence1 != Constants.Null && sequence2 != Constants.Null &&
614             ↳ arrays[i].Length > 3)
615         //    Assert.False(structure1 == structure2);
616         //if (sequence3 != Constants.Null && sequence2 != Constants.Null &&
617             ↳ arrays[i].Length > 3)
618         //    Assert.False(structure3 == structure2);
619
620         Assert.True(strings[i] == decompress1 && decompress1 == decompress2);
621         Assert.True(strings[i] == decompress3 && decompress3 == decompress2);
622     }
623
624     Assert.True((int)(scope1.Links.Unsync.Count() - initialCount1) <
625         ↳ totalCharacters);
626     Assert.True((int)(scope2.Links.Unsync.Count() - initialCount2) <
627         ↳ totalCharacters);
628     Assert.True((int)(scope3.Links.Unsync.Count() - initialCount3) <
629         ↳ totalCharacters);
630
631     Console.WriteLine($"{{(double)(scope1.Links.Unsync.Count() - initialCount1) /
632         ↳ totalCharacters}} | {{(double)(scope2.Links.Unsync.Count() - initialCount2) /
633         ↳ totalCharacters}} | {{(double)(scope3.Links.Unsync.Count() - initialCount3) /
634         ↳ totalCharacters}}");
635
636     Assert.True(scope1.Links.Unsync.Count() - initialCount1 <
637         ↳ scope2.Links.Unsync.Count() - initialCount2);

```

```

622     Assert.True(scope3.Links.Unsync.Count() - initialCount3 <
        ↳ scope2.Links.Unsync.Count() - initialCount2);
623
624     var duplicateProvider1 = new
        ↳ DuplicateSegmentsProvider<ulong>(scope1.Links.Unsync, scope1.Sequences);
625     var duplicateProvider2 = new
        ↳ DuplicateSegmentsProvider<ulong>(scope2.Links.Unsync, scope2.Sequences);
626     var duplicateProvider3 = new
        ↳ DuplicateSegmentsProvider<ulong>(scope3.Links.Unsync, scope3.Sequences);
627
628     var duplicateCounter1 = new DuplicateSegmentsCounter<ulong>(duplicateProvider1);
629     var duplicateCounter2 = new DuplicateSegmentsCounter<ulong>(duplicateProvider2);
630     var duplicateCounter3 = new DuplicateSegmentsCounter<ulong>(duplicateProvider3);
631
632     var duplicates1 = duplicateCounter1.Count();
633
634     ConsoleHelpers.Debug("-----");
635
636     var duplicates2 = duplicateCounter2.Count();
637
638     ConsoleHelpers.Debug("-----");
639
640     var duplicates3 = duplicateCounter3.Count();
641
642     Console.WriteLine($"{duplicates1} | {duplicates2} | {duplicates3}");
643
644     linkFrequenciesCache1.ValidateFrequencies();
645     linkFrequenciesCache3.ValidateFrequencies();
646 }
647 }
648
649 [Fact]
650 public static void CompressionStabilityTest()
651 {
652     // TODO: Fix bug (do a separate test)
653     //const ulong minNumbers = 0;
654     //const ulong maxNumbers = 1000;
655
656     const ulong minNumbers = 10000;
657     const ulong maxNumbers = 12500;
658
659     var strings = new List<string>();
660
661     for (ulong i = minNumbers; i < maxNumbers; i++)
662     {
663         strings.Add(i.ToString());
664     }
665
666     var arrays = strings.Select(UnicodeMap.FromStringToLinkArray).ToArray();
667     var totalCharacters = arrays.Select(x => x.Length).Sum();
668
669     using (var scope1 = new TempLinksTestScope(useSequences: true, sequencesOptions: new
        ↳ SequencesOptions<ulong> { UseCompression = true,
        ↳ EnforceSingleSequenceVersionOnWriteBasedOnExisting = true }))
        using (var scope2 = new TempLinksTestScope(useSequences: true))
        {
670         scope1.Links.UseUnicode();
671         scope2.Links.UseUnicode();
672
673         //var compressor1 = new Compressor(scope1.Links.Unsync, scope1.Sequences);
674         var compressor1 = scope1.Sequences;
675         var compressor2 = scope2.Sequences;
676
677         var compressed1 = new ulong[arrays.Length];
678         var compressed2 = new ulong[arrays.Length];
679
680         var sw1 = Stopwatch.StartNew();
681
682         var START = 0;
683         var END = arrays.Length;
684
685         // Collisions proved (cannot be solved by max doublet comparison, no stable rule)
686         // Stability issue starts at 10001 or 11000
687         //for (int i = START; i < END; i++)
688         //{
689             var first = compressor1.Compress(arrays[i]);
690             var second = compressor1.Compress(arrays[i]);
691
692             if (first == second)
693                 compressed1[i] = first;
694         }
695

```

```

696 //     else
697 //     {
698 //         // TODO: Find a solution for this case
699 //     }
700 //}
701
702 for (int i = START; i < END; i++)
703 {
704     var first = compressor1.Create(arrays[i].ShiftRight());
705     var second = compressor1.Create(arrays[i].ShiftRight());
706
707     if (first == second)
708     {
709         compressed1[i] = first;
710     }
711     else
712     {
713         // TODO: Find a solution for this case
714     }
715 }
716
717 var elapsed1 = sw1.Elapsed;
718
719 var balancedVariantConverter = new BalancedVariantConverter<ulong>(scope2.Links);
720
721 var sw2 = Stopwatch.StartNew();
722
723 for (int i = START; i < END; i++)
724 {
725     var first = balancedVariantConverter.Convert(arrays[i]);
726     var second = balancedVariantConverter.Convert(arrays[i]);
727
728     if (first == second)
729     {
730         compressed2[i] = first;
731     }
732 }
733
734 var elapsed2 = sw2.Elapsed;
735
736 Debug.WriteLine($"Compressor: {elapsed1}, Balanced sequence creator:
737 ↪ {elapsed2}");
738
739 Assert.True(elapsed1 > elapsed2);
740
741 // Checks
742 for (int i = START; i < END; i++)
743 {
744     var sequence1 = compressed1[i];
745     var sequence2 = compressed2[i];
746
747     if (sequence1 != _constants.Null && sequence2 != _constants.Null)
748     {
749         var decompress1 = UnicodeMap.FromSequenceLinkToString(sequence1,
750 ↪ scope1.Links);
751
752         var decompress2 = UnicodeMap.FromSequenceLinkToString(sequence2,
753 ↪ scope2.Links);
754
755         //var structure1 = scope1.Links.FormatStructure(sequence1, link =>
756 ↪ link.IsPartialPoint());
757         //var structure2 = scope2.Links.FormatStructure(sequence2, link =>
758 ↪ link.IsPartialPoint());
759
760         //if (sequence1 != Constants.Null && sequence2 != Constants.Null &&
761 ↪ arrays[i].Length > 3)
762         //    Assert.False(structure1 == structure2);
763
764         Assert.True(strings[i] == decompress1 && decompress1 == decompress2);
765     }
766 }
767
768 Assert.True((int)(scope1.Links.Count() - UnicodeMap.MapSize) < totalCharacters);
769 Assert.True((int)(scope2.Links.Count() - UnicodeMap.MapSize) < totalCharacters);
770
771 Debug.WriteLine($"{{(double)(scope1.Links.Count() - UnicodeMap.MapSize) /
772 ↪ totalCharacters}} | {{(double)(scope2.Links.Count() - UnicodeMap.MapSize) /
773 ↪ totalCharacters}}");

```



```

767         Assert.True(scope1.Links.Count() <= scope2.Links.Count());
768
769         //compressor1.ValidateFrequencies();
770     }
771 }
772
773 [Fact]
774 public static void RandomNumbersCompressionQualityTest()
775 {
776     const ulong N = 500;
777
778     //const ulong minNumbers = 10000;
779     //const ulong maxNumbers = 20000;
780
781     //var strings = new List<string>();
782
783     //for (ulong i = 0; i < N; i++)
784     //    strings.Add(RandomHelpers.DefaultFactory.NextUInt64(minNumbers,
785     //        ↪ maxNumbers).ToString());
786
787     var strings = new List<string>();
788
789     for (ulong i = 0; i < N; i++)
790     {
791         strings.Add(RandomHelpers.Default.NextUInt64().ToString());
792     }
793
794     strings = strings.Distinct().ToList();
795
796     var arrays = strings.Select(UnicodeMap.FromStringToLinkArray).ToArray();
797     var totalCharacters = arrays.Select(x => x.Length).Sum();
798
799     using (var scope1 = new TempLinksTestScope(useSequences: true, sequencesOptions: new
800     ↪ SequencesOptions<ulong> { UseCompression = true,
801     ↪ EnforceSingleSequenceVersionOnWriteBasedOnExisting = true }))
802     using (var scope2 = new TempLinksTestScope(useSequences: true))
803     {
804         scope1.Links.UseUnicode();
805         scope2.Links.UseUnicode();
806
807         var compressor1 = scope1.Sequences;
808         var compressor2 = scope2.Sequences;
809
810         var compressed1 = new ulong[arrays.Length];
811         var compressed2 = new ulong[arrays.Length];
812
813         var sw1 = Stopwatch.StartNew();
814
815         var START = 0;
816         var END = arrays.Length;
817
818         for (int i = START; i < END; i++)
819         {
820             compressed1[i] = compressor1.Create(arrays[i].ShiftRight());
821         }
822
823         var elapsed1 = sw1.Elapsed;
824
825         var balancedVariantConverter = new BalancedVariantConverter<ulong>(scope2.Links);
826
827         var sw2 = Stopwatch.StartNew();
828
829         for (int i = START; i < END; i++)
830         {
831             compressed2[i] = balancedVariantConverter.Convert(arrays[i]);
832         }
833
834         var elapsed2 = sw2.Elapsed;
835
836         Debug.WriteLine($"Compressor: {elapsed1}, Balanced sequence creator:
837         ↪ {elapsed2}");
838
839         Assert.True(elapsed1 > elapsed2);
840
841         // Checks
842         for (int i = START; i < END; i++)
843         {
844             var sequence1 = compressed1[i];
845             var sequence2 = compressed2[i];

```

```

843         if (sequence1 != _constants.Null && sequence2 != _constants.Null)
844         {
845             var decompress1 = UnicodeMap.FromSequenceLinkToString(sequence1,
846                 ↳ scope1.Links);
847
848             var decompress2 = UnicodeMap.FromSequenceLinkToString(sequence2,
849                 ↳ scope2.Links);
850
851             Assert.True(strings[i] == decompress1 && decompress1 == decompress2);
852         }
853     }
854
855     Assert.True((int)(scope1.Links.Count() - UnicodeMap.MapSize) < totalCharacters);
856     Assert.True((int)(scope2.Links.Count() - UnicodeMap.MapSize) < totalCharacters);
857
858     Debug.WriteLine($"{{(double)(scope1.Links.Count() - UnicodeMap.MapSize) /
859         ↳ totalCharacters}} | {{(double)(scope2.Links.Count() - UnicodeMap.MapSize) /
860         ↳ totalCharacters}}");
861
862     // Can be worse than balanced variant
863     //Assert.True(scope1.Links.Count() <= scope2.Links.Count());
864
865     //compressor1.ValidateFrequencies();
866 }
867
868 [Fact]
869 public static void AllTreeBreakDownAtSequencesCreationBugTest()
870 {
871     // Made out of AllPossibleConnectionsTest test.
872
873     //const long sequenceLength = 5; //100% bug
874     const long sequenceLength = 4; //100% bug
875     //const long sequenceLength = 3; //100% _no_bug (ok)
876
877     using (var scope = new TempLinksTestScope(useSequences: true))
878     {
879         var links = scope.Links;
880         var sequences = scope.Sequences;
881
882         var sequence = new ulong[sequenceLength];
883         for (var i = 0; i < sequenceLength; i++)
884         {
885             sequence[i] = links.Create();
886         }
887
888         var createResults = sequences.CreateAllVariants2(sequence);
889
890         Global.Trash = createResults;
891
892         for (var i = 0; i < sequenceLength; i++)
893         {
894             links.Delete(sequence[i]);
895         }
896     }
897 }
898
899 [Fact]
900 public static void AllPossibleConnectionsTest()
901 {
902     const long sequenceLength = 5;
903
904     using (var scope = new TempLinksTestScope(useSequences: true))
905     {
906         var links = scope.Links;
907         var sequences = scope.Sequences;
908
909         var sequence = new ulong[sequenceLength];
910         for (var i = 0; i < sequenceLength; i++)
911         {
912             sequence[i] = links.Create();
913         }
914
915         var createResults = sequences.CreateAllVariants2(sequence);
916         var reverseResults = sequences.CreateAllVariants2(sequence.Reverse().ToArray());
917
918         for (var i = 0; i < 1; i++)
919         {
920             var sw1 = Stopwatch.StartNew();

```

```

918         var searchResults1 = sequences.GetAllConnections(sequence); sw1.Stop();
919
920         var sw2 = Stopwatch.StartNew();
921         var searchResults2 = sequences.GetAllConnections1(sequence); sw2.Stop();
922
923         var sw3 = Stopwatch.StartNew();
924         var searchResults3 = sequences.GetAllConnections2(sequence); sw3.Stop();
925
926         var sw4 = Stopwatch.StartNew();
927         var searchResults4 = sequences.GetAllConnections3(sequence); sw4.Stop();
928
929         Global.Trash = searchResults3;
930         Global.Trash = searchResults4; //-V3008
931
932         var intersection1 = createResults.Intersect(searchResults1).ToList();
933         Assert.True(intersection1.Count == createResults.Length);
934
935         var intersection2 = reverseResults.Intersect(searchResults1).ToList();
936         Assert.True(intersection2.Count == reverseResults.Length);
937
938         var intersection0 = searchResults1.Intersect(searchResults2).ToList();
939         Assert.True(intersection0.Count == searchResults2.Count);
940
941         var intersection3 = searchResults2.Intersect(searchResults3).ToList();
942         Assert.True(intersection3.Count == searchResults3.Count);
943
944         var intersection4 = searchResults3.Intersect(searchResults4).ToList();
945         Assert.True(intersection4.Count == searchResults4.Count);
946     }
947
948     for (var i = 0; i < sequenceLength; i++)
949     {
950         links.Delete(sequence[i]);
951     }
952 }
953
954 [Fact(Skip = "Correct implementation is pending")]
955 public static void CalculateAllUsagesTest()
956 {
957     const long sequenceLength = 3;
958
959     using (var scope = new TempLinksTestScope(useSequences: true))
960     {
961         var links = scope.Links;
962         var sequences = scope.Sequences;
963
964         var sequence = new ulong[sequenceLength];
965         for (var i = 0; i < sequenceLength; i++)
966         {
967             sequence[i] = links.Create();
968         }
969
970         var createResults = sequences.CreateAllVariants2(sequence);
971
972         //var reverseResults =
973         ↪ sequences.CreateAllVariants2(sequence.Reverse().ToArray());
974
975         for (var i = 0; i < 1; i++)
976         {
977             var linksTotalUsages1 = new ulong[links.Count() + 1];
978
979             sequences.CalculateAllUsages(linksTotalUsages1);
980
981             var linksTotalUsages2 = new ulong[links.Count() + 1];
982
983             sequences.CalculateAllUsages2(linksTotalUsages2);
984
985             var intersection1 = linksTotalUsages1.Intersect(linksTotalUsages2).ToList();
986             Assert.True(intersection1.Count == linksTotalUsages2.Length);
987         }
988
989         for (var i = 0; i < sequenceLength; i++)
990         {
991             links.Delete(sequence[i]);
992         }
993     }
994 }
995 }
996 }

```

1.111 ./Platform.Data.Doublets.Tests/TempLinksTestScope.cs

```

1  using System.IO;
2  using Platform.Disposables;
3  using Platform.Data.Doublets.Sequences;
4  using Platform.Data.Doublets.Decorators;
5  using Platform.Data.Doublets.ResizableDirectMemory.Specific;
6
7  namespace Platform.Data.Doublets.Tests
8  {
9      public class TempLinksTestScope : DisposableBase
10     {
11         public ILinks<ulong> MemoryAdapter { get; }
12         public SynchronizedLinks<ulong> Links { get; }
13         public Sequences.Sequences Sequences { get; }
14         public string TempFilename { get; }
15         public string TempTransactionLogFilename { get; }
16         private readonly bool _deleteFiles;
17
18         public TempLinksTestScope(bool deleteFiles = true, bool useSequences = false, bool
            ↪ useLog = false) : this(new SequencesOptions<ulong>(), deleteFiles, useSequences,
            ↪ useLog) { }
19
20         public TempLinksTestScope(SequencesOptions<ulong> sequencesOptions, bool deleteFiles =
            ↪ true, bool useSequences = false, bool useLog = false)
21         {
22             _deleteFiles = deleteFiles;
23             TempFilename = Path.GetTempFileName();
24             TempTransactionLogFilename = Path.GetTempFileName();
25             var coreMemoryAdapter = new UInt64ResizableDirectMemoryLinks(TempFilename);
26             MemoryAdapter = useLog ? (ILinks<ulong>)new
                ↪ UInt64LinksTransactionsLayer(coreMemoryAdapter, TempTransactionLogFilename) :
                ↪ coreMemoryAdapter;
27             Links = new SynchronizedLinks<ulong>(new UInt64Links(MemoryAdapter));
28             if (useSequences)
29             {
30                 Sequences = new Sequences.Sequences(Links, sequencesOptions);
31             }
32         }
33
34         protected override void Dispose(bool manual, bool wasDisposed)
35         {
36             if (!wasDisposed)
37             {
38                 Links.Unsync.DisposeIfPossible();
39                 if (_deleteFiles)
40                 {
41                     DeleteFiles();
42                 }
43             }
44         }
45
46         public void DeleteFiles()
47         {
48             File.Delete(TempFilename);
49             File.Delete(TempTransactionLogFilename);
50         }
51     }
52 }

```

1.112 ./Platform.Data.Doublets.Tests/TestExtensions.cs

```

1  using System.Collections.Generic;
2  using Xunit;
3  using Platform.Ranges;
4  using Platform.Numbers;
5  using Platform.Random;
6  using Platform.Setters;
7  using Platform.Converters;
8
9  namespace Platform.Data.Doublets.Tests
10 {
11     public static class TestExtensions
12     {
13         public static void TestCRUDOperations<T>(this ILinks<T> links)
14         {
15             var constants = links.Constants;
16
17             var equalityComparer = EqualityComparer<T>.Default;
18
19             var zero = default(T);
20             var one = Arithmetic.Increment(zero);

```

```

21
22 // Create Link
23 Assert.True(equalityComparer.Equals(links.Count(), zero));
24
25 var setter = new Setter<T>(constants.Null);
26 links.Each(constants.Any, constants.Any, setter.SetAndReturnTrue);
27
28 Assert.True(equalityComparer.Equals(setter.Result, constants.Null));
29
30 var linkAddress = links.Create();
31
32 var link = new Link<T>(links.GetLink(linkAddress));
33
34 Assert.True(link.Count == 3);
35 Assert.True(equalityComparer.Equals(link.Index, linkAddress));
36 Assert.True(equalityComparer.Equals(link.Source, constants.Null));
37 Assert.True(equalityComparer.Equals(link.Target, constants.Null));
38
39 Assert.True(equalityComparer.Equals(links.Count(), one));
40
41 // Get first link
42 setter = new Setter<T>(constants.Null);
43 links.Each(constants.Any, constants.Any, setter.SetAndReturnFalse);
44
45 Assert.True(equalityComparer.Equals(setter.Result, linkAddress));
46
47 // Update link to reference itself
48 links.Update(linkAddress, linkAddress, linkAddress);
49
50 link = new Link<T>(links.GetLink(linkAddress));
51
52 Assert.True(equalityComparer.Equals(link.Source, linkAddress));
53 Assert.True(equalityComparer.Equals(link.Target, linkAddress));
54
55 // Update link to reference null (prepare for delete)
56 var updated = links.Update(linkAddress, constants.Null, constants.Null);
57
58 Assert.True(equalityComparer.Equals(updated, linkAddress));
59
60 link = new Link<T>(links.GetLink(linkAddress));
61
62 Assert.True(equalityComparer.Equals(link.Source, constants.Null));
63 Assert.True(equalityComparer.Equals(link.Target, constants.Null));
64
65 // Delete link
66 links.Delete(linkAddress);
67
68 Assert.True(equalityComparer.Equals(links.Count(), zero));
69
70 setter = new Setter<T>(constants.Null);
71 links.Each(constants.Any, constants.Any, setter.SetAndReturnTrue);
72
73 Assert.True(equalityComparer.Equals(setter.Result, constants.Null));
74 }
75
76 public static void TestRawNumbersCRUDOperations<T>(this ILinks<T> links)
77 {
78 // Constants
79 var constants = links.Constants;
80 var equalityComparer = EqualityComparer<T>.Default;
81
82 var zero = default(T);
83 var one = Arithmetic.Increment(zero);
84 var two = Arithmetic.Increment(one);
85
86 var h106E = new Hybrid<T>(106L, isExternal: true);
87 var h107E = new Hybrid<T>(-char.ConvertFromUtf32(107)[0]);
88 var h108E = new Hybrid<T>(-108L);
89
90 Assert.Equal(106L, h106E.AbsoluteValue);
91 Assert.Equal(107L, h107E.AbsoluteValue);
92 Assert.Equal(108L, h108E.AbsoluteValue);
93
94 // Create Link (External -> External)
95 var linkAddress1 = links.Create();
96
97 links.Update(linkAddress1, h106E, h108E);
98
99 var link1 = new Link<T>(links.GetLink(linkAddress1));
100

```

```

101 Assert.True(equalityComparer.Equals(link1.Source, h106E));
102 Assert.True(equalityComparer.Equals(link1.Target, h108E));
103
104 // Create Link (Internal -> External)
105 var linkAddress2 = links.Create();
106
107 links.Update(linkAddress2, linkAddress1, h108E);
108
109 var link2 = new Link<T>(links.GetLink(linkAddress2));
110
111 Assert.True(equalityComparer.Equals(link2.Source, linkAddress1));
112 Assert.True(equalityComparer.Equals(link2.Target, h108E));
113
114 // Create Link (Internal -> Internal)
115 var linkAddress3 = links.Create();
116
117 links.Update(linkAddress3, linkAddress1, linkAddress2);
118
119 var link3 = new Link<T>(links.GetLink(linkAddress3));
120
121 Assert.True(equalityComparer.Equals(link3.Source, linkAddress1));
122 Assert.True(equalityComparer.Equals(link3.Target, linkAddress2));
123
124 // Search for created link
125 var setter1 = new Setter<T>(constants.Null);
126 links.Each(h106E, h108E, setter1.SetAndReturnFalse);
127
128 Assert.True(equalityComparer.Equals(setter1.Result, linkAddress1));
129
130 // Search for nonexistent link
131 var setter2 = new Setter<T>(constants.Null);
132 links.Each(h106E, h107E, setter2.SetAndReturnFalse);
133
134 Assert.True(equalityComparer.Equals(setter2.Result, constants.Null));
135
136 // Update link to reference null (prepare for delete)
137 var updated = links.Update(linkAddress3, constants.Null, constants.Null);
138
139 Assert.True(equalityComparer.Equals(updated, linkAddress3));
140
141 link3 = new Link<T>(links.GetLink(linkAddress3));
142
143 Assert.True(equalityComparer.Equals(link3.Source, constants.Null));
144 Assert.True(equalityComparer.Equals(link3.Target, constants.Null));
145
146 // Delete link
147 links.Delete(linkAddress3);
148
149 Assert.True(equalityComparer.Equals(links.Count(), two));
150
151 var setter3 = new Setter<T>(constants.Null);
152 links.Each(constants.Any, constants.Any, setter3.SetAndReturnTrue);
153
154 Assert.True(equalityComparer.Equals(setter3.Result, linkAddress2));
155 }
156
157 public static void TestMultipleRandomCreationsAndDeletions<TLink>(this ILinks<TLink>
→ links, int maximumOperationsPerCycle)
158 {
159     var comparer = Comparer<TLink>.Default;
160     var addressToUInt64Converter = CheckedConverter<TLink, ulong>.Default;
161     var uInt64ToAddressConverter = CheckedConverter<ulong, TLink>.Default;
162     for (var N = 1; N < maximumOperationsPerCycle; N++)
163     {
164         var random = new System.Random(N);
165         var created = 0UL;
166         var deleted = 0UL;
167         for (var i = 0; i < N; i++)
168         {
169             var linksCount = addressToUInt64Converter.Convert(links.Count());
170             var createPoint = random.NextBoolean();
171             if (linksCount > 2 && createPoint)
172             {
173                 var linksAddressRange = new Range<ulong>(1, linksCount);
174                 TLink source = uInt64ToAddressConverter.Convert(random.NextUInt64(linksA_
→ ddressRange));
175                 TLink target = uInt64ToAddressConverter.Convert(random.NextUInt64(linksA_
→ ddressRange));
176                 //--V3086

```

```

176         var resultLink = links.GetOrCreate(source, target);
177         if (comparer.Compare(resultLink,
178             ↪ uInt64ToAddressConverter.Convert(linksCount)) > 0)
179         {
180             created++;
181         }
182     else
183     {
184         links.Create();
185         created++;
186     }
187 }
188 Assert.True(created == addressToUInt64Converter.Convert(links.Count()));
189 for (var i = 0; i < N; i++)
190 {
191     TLink link = uInt64ToAddressConverter.Convert((ulong)i + 1UL);
192     if (links.Exists(link))
193     {
194         links.Delete(link);
195         deleted++;
196     }
197 }
198 Assert.True(addressToUInt64Converter.Convert(links.Count()) == 0L);
199 }
200 }
201 }
202 }

```

1.113 ./Platform.Data.Doublets.Tests/UInt64LinksTests.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Diagnostics;
4  using System.IO;
5  using System.Text;
6  using System.Threading;
7  using System.Threading.Tasks;
8  using Xunit;
9  using Platform.Disposables;
10 using Platform.Ranges;
11 using Platform.Random;
12 using Platform.Timestamps;
13 using Platform.Reflection;
14 using Platform.Singletons;
15 using Platform.Scopes;
16 using Platform.Counters;
17 using Platform.Diagnostics;
18 using Platform.IO;
19 using Platform.Memory;
20 using Platform.Data.Doublets.Decorators;
21 using Platform.Data.Doublets.ResizableDirectMemory.Specific;
22
23 namespace Platform.Data.Doublets.Tests
24 {
25     public static class UInt64LinksTests
26     {
27         private static readonly LinksConstants<ulong> _constants =
28             ↪ Default<LinksConstants<ulong>>.Instance;
29
30         private const long Iterations = 10 * 1024;
31
32         #region Concept
33
34         [Fact]
35         public static void MultipleCreateAndDeleteTest()
36         {
37             using (var scope = new Scope<Types<HeapResizableDirectMemory,
38                 ↪ UInt64ResizableDirectMemoryLinks>>())
39             {
40                 new UInt64Links(scope.Use<ILinks<ulong>>()).TestMultipleRandomCreationsAndDeletions(100);
41             }
42         }
43
44         [Fact]
45         public static void CascadeUpdateTest()
46         {
47             var itself = _constants.Itself;
48             using (var scope = new TempLinksTestScope(useLog: true))
49             {
40                 var links = scope.Links;

```

```

49
50     var l1 = links.Create();
51     var l2 = links.Create();
52
53     l2 = links.Update(l2, l2, l1, l2);
54
55     links.CreateAndUpdate(l2, itself);
56     links.CreateAndUpdate(l2, itself);
57
58     l2 = links.Update(l2, l1);
59
60     links.Delete(l2);
61
62     Global.Trash = links.Count();
63
64     links.Unsync.DisposeIfPossible(); // Close links to access log
65
66     Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(scope
        ↪ e.TempTransactionLogFilename);
67     }
68 }
69
70 [Fact]
71 public static void BasicTransactionLogTest()
72 {
73     using (var scope = new TempLinksTestScope(useLog: true))
74     {
75         var links = scope.Links;
76         var l1 = links.Create();
77         var l2 = links.Create();
78
79         Global.Trash = links.Update(l2, l2, l1, l2);
80
81         links.Delete(l1);
82
83         links.Unsync.DisposeIfPossible(); // Close links to access log
84
85         Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(scope
        ↪ e.TempTransactionLogFilename);
86     }
87 }
88
89 [Fact]
90 public static void TransactionAutoRevertedTest()
91 {
92     // Auto Reverted (Because no commit at transaction)
93     using (var scope = new TempLinksTestScope(useLog: true))
94     {
95         var links = scope.Links;
96         var transactionsLayer = (UInt64LinksTransactionsLayer)scope.MemoryAdapter;
97         using (var transaction = transactionsLayer.BeginTransaction())
98         {
99             var l1 = links.Create();
100             var l2 = links.Create();
101
102             links.Update(l2, l2, l1, l2);
103         }
104
105         Assert.Equal(0UL, links.Count());
106
107         links.Unsync.DisposeIfPossible();
108
109         var transitions = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(s
        ↪ cope.TempTransactionLogFilename);
110         Assert.Single(transitions);
111     }
112 }
113
114 [Fact]
115 public static void TransactionUserCodeErrorNoDataSavedTest()
116 {
117     // User Code Error (Autoreverted), no data saved
118     var itself = _constants.Itself;
119
120     TempLinksTestScope lastScope = null;
121     try
122     {
123         using (var scope = lastScope = new TempLinksTestScope(deleteFiles: false,
        ↪ useLog: true))

```



```

124     {
125         var links = scope.Links;
126         var transactionsLayer = (UInt64LinksTransactionsLayer)((LinksDisposableDecor_
        ↳ atorBase<ulong>>links.Unsync).Links;
127         using (var transaction = transactionsLayer.BeginTransaction())
128         {
129             var l1 = links.CreateAndUpdate(itself, itself);
130             var l2 = links.CreateAndUpdate(itself, itself);
131
132             l2 = links.Update(l2, l2, l1, l2);
133
134             links.CreateAndUpdate(l2, itself);
135             links.CreateAndUpdate(l2, itself);
136
137             //Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transi_
        ↳ tion>(scope.TempTransactionLogFilename);
138
139             l2 = links.Update(l2, l1);
140
141             links.Delete(l2);
142
143             ExceptionThrower();
144
145             transaction.Commit();
146         }
147
148         Global.Trash = links.Count();
149     }
150 }
151 catch
152 {
153     Assert.False(lastScope == null);
154
155     var transitions = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(l_
        ↳ astScope.TempTransactionLogFilename);
156
157     Assert.True(transitions.Length == 1 && transitions[0].Before.IsNull() &&
        ↳ transitions[0].After.IsNull());
158
159     lastScope.DeleteFiles();
160 }
161 }
162
163 [Fact]
164 public static void TransactionUserCodeErrorSomeDataSavedTest()
165 {
166     // User Code Error (Autoreverted), some data saved
167     var itself = _constants.Itself;
168
169     TempLinksTestScope lastScope = null;
170     try
171     {
172         ulong l1;
173         ulong l2;
174
175         using (var scope = new TempLinksTestScope(useLog: true))
176         {
177             var links = scope.Links;
178             l1 = links.CreateAndUpdate(itself, itself);
179             l2 = links.CreateAndUpdate(itself, itself);
180
181             l2 = links.Update(l2, l2, l1, l2);
182
183             links.CreateAndUpdate(l2, itself);
184             links.CreateAndUpdate(l2, itself);
185
186             links.Unsync.DisposeIfPossible();
187
188             Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(
        ↳ scope.TempTransactionLogFilename);
189         }
190
191         using (var scope = lastScope = new TempLinksTestScope(deleteFiles: false,
        ↳ useLog: true))
192         {
193             var links = scope.Links;
194             var transactionsLayer = (UInt64LinksTransactionsLayer)links.Unsync;
195             using (var transaction = transactionsLayer.BeginTransaction())
196             {

```

```

197         l2 = links.Update(l2, l1);
198
199         links.Delete(l2);
200
201         ExceptionThrower();
202
203         transaction.Commit();
204     }
205
206     Global.Trash = links.Count();
207 }
208
209 catch
210 {
211     Assert.False(lastScope == null);
212
213     Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(last
        ↳ Scope.TempTransactionLogFilename);
214
215     lastScope.DeleteFiles();
216 }
217
218
219 [Fact]
220 public static void TransactionCommit()
221 {
222     var itself = _constants.Itself;
223
224     var tempDatabaseFilename = Path.GetTempFileName();
225     var tempTransactionLogFilename = Path.GetTempFileName();
226
227     // Commit
228     using (var memoryAdapter = new UInt64LinksTransactionsLayer(new
        ↳ UInt64ResizableDirectMemoryLinks(tempDatabaseFilename),
        ↳ tempTransactionLogFilename))
229     using (var links = new UInt64Links(memoryAdapter))
230     {
231         using (var transaction = memoryAdapter.BeginTransaction())
232         {
233             var l1 = links.CreateAndUpdate(itself, itself);
234             var l2 = links.CreateAndUpdate(itself, itself);
235
236             Global.Trash = links.Update(l2, l2, l1, l2);
237
238             links.Delete(l1);
239
240             transaction.Commit();
241         }
242
243         Global.Trash = links.Count();
244     }
245
246     Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(tempTran
        ↳ sactionLogFilename);
247 }
248
249 [Fact]
250 public static void TransactionDamage()
251 {
252     var itself = _constants.Itself;
253
254     var tempDatabaseFilename = Path.GetTempFileName();
255     var tempTransactionLogFilename = Path.GetTempFileName();
256
257     // Commit
258     using (var memoryAdapter = new UInt64LinksTransactionsLayer(new
        ↳ UInt64ResizableDirectMemoryLinks(tempDatabaseFilename),
        ↳ tempTransactionLogFilename))
259     using (var links = new UInt64Links(memoryAdapter))
260     {
261         using (var transaction = memoryAdapter.BeginTransaction())
262         {
263             var l1 = links.CreateAndUpdate(itself, itself);
264             var l2 = links.CreateAndUpdate(itself, itself);
265
266             Global.Trash = links.Update(l2, l2, l1, l2);
267
268             links.Delete(l1);
269

```

```

270         transaction.Commit();
271     }
272
273     Global.Trash = links.Count();
274 }
275
276 Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(tempTran_
    ↪ sactionLogFilename);
277
278 // Damage database
279
280 FileHelpers.WriteFirst(tempTransactionLogFilename, new
    ↪ UInt64LinksTransactionsLayer.Transition(new UniqueTimestampFactory(), 555));
281
282 // Try load damaged database
283 try
284 {
285     // TODO: Fix
286     using (var memoryAdapter = new UInt64LinksTransactionsLayer(new
        ↪ UInt64ResizableDirectMemoryLinks(tempDatabaseFilename),
        ↪ tempTransactionLogFilename))
        using (var links = new UInt64Links(memoryAdapter))
        {
287         Global.Trash = links.Count();
288     }
289 }
290
291 catch (NotSupportedException ex)
292 {
293     Assert.True(ex.Message == "Database is damaged, autorecovery is not supported
294         ↪ yet.");
295 }
296
297 Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(tempTran_
    ↪ sactionLogFilename);
298
299 File.Delete(tempDatabaseFilename);
300 File.Delete(tempTransactionLogFilename);
301 }
302
303 [Fact]
304 public static void Bug1Test()
305 {
306     var tempDatabaseFilename = Path.GetTempFileName();
307     var tempTransactionLogFilename = Path.GetTempFileName();
308
309     var itself = _constants.Itself;
310
311     // User Code Error (Autoreverted), some data saved
312     try
313     {
314         ulong l1;
315         ulong l2;
316
317         using (var memory = new UInt64ResizableDirectMemoryLinks(tempDatabaseFilename))
318         using (var memoryAdapter = new UInt64LinksTransactionsLayer(memory,
            ↪ tempTransactionLogFilename))
            using (var links = new UInt64Links(memoryAdapter))
            {
319             l1 = links.CreateAndUpdate(itself, itself);
320             l2 = links.CreateAndUpdate(itself, itself);
321
322             l2 = links.Update(l2, l2, l1, l2);
323
324             links.CreateAndUpdate(l2, itself);
325             links.CreateAndUpdate(l2, itself);
326         }
327
328         Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(temp_
            ↪ TransactionLogFilename);
329
330         using (var memory = new UInt64ResizableDirectMemoryLinks(tempDatabaseFilename))
331         using (var memoryAdapter = new UInt64LinksTransactionsLayer(memory,
            ↪ tempTransactionLogFilename))
            using (var links = new UInt64Links(memoryAdapter))
            {
332             using (var transaction = memoryAdapter.BeginTransaction())
333             {
334                 l2 = links.Update(l2, l1);
335             }
336         }
337     }
338 }
339

```

```

340         links.Delete(l2);
341
342         ExceptionThrower();
343
344         transaction.Commit();
345     }
346
347     Global.Trash = links.Count();
348 }
349 }
350 catch
351 {
352     Global.Trash = FileHelpers.ReadAll<UInt64LinksTransactionsLayer.Transition>(temp ↵
        ↵ TransactionLogFilename);
353 }
354
355 File.Delete(tempDatabaseFilename);
356 File.Delete(tempTransactionLogFilename);
357 }
358
359 private static void ExceptionThrower() => throw new InvalidOperationException();
360
361 [Fact]
362 public static void PathsTest()
363 {
364     var source = _constants.SourcePart;
365     var target = _constants.TargetPart;
366
367     using (var scope = new TempLinksTestScope())
368     {
369         var links = scope.Links;
370         var l1 = links.CreatePoint();
371         var l2 = links.CreatePoint();
372
373         var r1 = links.GetByKeys(l1, source, target, source);
374         var r2 = links.CheckPathExistence(l2, l2, l2, l2);
375     }
376 }
377
378 [Fact]
379 public static void RecursiveStringFormattingTest()
380 {
381     using (var scope = new TempLinksTestScope(useSequences: true))
382     {
383         var links = scope.Links;
384         var sequences = scope.Sequences; // TODO: Auto use sequences on Sequences getter.
385
386         var a = links.CreatePoint();
387         var b = links.CreatePoint();
388         var c = links.CreatePoint();
389
390         var ab = links.GetOrCreate(a, b);
391         var cb = links.GetOrCreate(c, b);
392         var ac = links.GetOrCreate(a, c);
393
394         a = links.Update(a, c, b);
395         b = links.Update(b, a, c);
396         c = links.Update(c, a, b);
397
398         Debug.WriteLine(links.FormatStructure(ab, link => link.IsFullPoint(), true));
399         Debug.WriteLine(links.FormatStructure(cb, link => link.IsFullPoint(), true));
400         Debug.WriteLine(links.FormatStructure(ac, link => link.IsFullPoint(), true));
401
402         Assert.True(links.FormatStructure(cb, link => link.IsFullPoint(), true) ==
            ↵ "(5:(4:5 (6:5 4)) 6)");
403         Assert.True(links.FormatStructure(ac, link => link.IsFullPoint(), true) ==
            ↵ "(6:(5:(4:5 6) 6) 4)");
404         Assert.True(links.FormatStructure(ab, link => link.IsFullPoint(), true) ==
            ↵ "(4:(5:4 (6:5 4)) 6)");
405
406         // TODO: Think how to build balanced syntax tree while formatting structure (eg.
            ↵ "(4:(5:4 6) (6:5 4))" instead of "(4:(5:4 (6:5 4)) 6)"
407
408         Assert.True(sequences.SafeFormatSequence(cb, DefaultFormatter, false) ==
            ↵ "{5}{5}{4}{6}");
409         Assert.True(sequences.SafeFormatSequence(ac, DefaultFormatter, false) ==
            ↵ "{5}{6}{6}{4}");
410         Assert.True(sequences.SafeFormatSequence(ab, DefaultFormatter, false) ==
            ↵ "{4}{5}{4}{6}");

```

```

411     }
412 }
413
414 private static void DefaultFormatter(StringBuilder sb, ulong link)
415 {
416     sb.Append(link.ToString());
417 }
418
419 #endregion
420
421 #region Performance
422
423 /*
424 public static void RunAllPerformanceTests()
425 {
426     try
427     {
428         links.TestLinksInSteps();
429     }
430     catch (Exception ex)
431     {
432         ex.WriteToConsole();
433     }
434
435     return;
436
437     try
438     {
439         //ThreadPool.SetMaxThreads(2, 2);
440
441         // Запускаем все тесты дважды, чтобы первоначальная инициализация не повлияла на
442         // Также это дополнительно помогает в отладке
443         // Увеличивает вероятность попадания информации в кэши
444         for (var i = 0; i < 10; i++)
445         {
446             //0 - 10 ГБ
447             //Каждые 100 МБ срез цифр
448
449             //links.TestGetSourceFunction();
450             //links.TestGetSourceFunctionInParallel();
451             //links.TestGetTargetFunction();
452             //links.TestGetTargetFunctionInParallel();
453             links.Create64BillionLinks();
454
455             links.TestRandomSearchFixed();
456             //links.Create64BillionLinksInParallel();
457             links.TestEachFunction();
458             //links.TestForeach();
459             //links.TestParallelForeach();
460         }
461
462         links.TestDeletionOfAllLinks();
463
464     }
465     catch (Exception ex)
466     {
467         ex.WriteToConsole();
468     }
469 }*/
470
471 /*
472 public static void TestLinksInSteps()
473 {
474     const long gibibyte = 1024 * 1024 * 1024;
475     const long mebibyte = 1024 * 1024;
476
477     var totalLinksToCreate = gibibyte /
478     Platform.Links.Data.Core.Doublets.Links.LinkSizeInBytes;
479     var linksStep = 102 * mebibyte /
480     Platform.Links.Data.Core.Doublets.Links.LinkSizeInBytes;
481
482     var creationMeasurements = new List<TimeSpan>();
483     var searchMeasurements = new List<TimeSpan>();
484     var deletionMeasurements = new List<TimeSpan>();
485
486     GetBaseRandomLoopOverhead(linksStep);
487     GetBaseRandomLoopOverhead(linksStep);
488
489     var stepLoopOverhead = GetBaseRandomLoopOverhead(linksStep);

```

```

488     ConsoleHelpers.Debug("Step loop overhead: {0}.", stepLoopOverhead);
489
490     var loops = totalLinksToCreate / linksStep;
491
492     for (int i = 0; i < loops; i++)
493     {
494         creationMeasurements.Add(Measure(() => links.RunRandomCreations(linksStep)));
495         searchMeasurements.Add(Measure(() => links.RunRandomSearches(linksStep)));
496
497         Console.WriteLine("\rC + S {0}/{1}", i + 1, loops);
498     }
499
500     ConsoleHelpers.Debug();
501
502     for (int i = 0; i < loops; i++)
503     {
504         deletionMeasurements.Add(Measure(() => links.RunRandomDeletions(linksStep)));
505
506         Console.WriteLine("\rD {0}/{1}", i + 1, loops);
507     }
508
509     ConsoleHelpers.Debug();
510
511     ConsoleHelpers.Debug("C S D");
512
513     for (int i = 0; i < loops; i++)
514     {
515         ConsoleHelpers.Debug("{0} {1} {2}", creationMeasurements[i],
516 ↪ searchMeasurements[i], deletionMeasurements[i]);
517     }
518
519     ConsoleHelpers.Debug("C S D (no overhead)");
520
521     for (int i = 0; i < loops; i++)
522     {
523         ConsoleHelpers.Debug("{0} {1} {2}", creationMeasurements[i] - stepLoopOverhead,
524 ↪ searchMeasurements[i] - stepLoopOverhead, deletionMeasurements[i] - stepLoopOverhead);
525     }
526
527     ConsoleHelpers.Debug("All tests done. Total links left in database: {0}.",
528 ↪ links.Total);
529
530     private static void CreatePoints(this Platform.Links.Data.Core.Doublets.Links links, long
531 ↪ amountToCreate)
532     {
533         for (long i = 0; i < amountToCreate; i++)
534             links.Create(0, 0);
535     }
536
537     private static TimeSpan GetBaseRandomLoopOverhead(long loops)
538     {
539         return Measure(() =>
540         {
541             ulong maxValue = RandomHelpers.DefaultFactory.NextUInt64();
542             ulong result = 0;
543             for (long i = 0; i < loops; i++)
544             {
545                 var source = RandomHelpers.DefaultFactory.NextUInt64(maxValue);
546                 var target = RandomHelpers.DefaultFactory.NextUInt64(maxValue);
547
548                 result += maxValue + source + target;
549             }
550             Global.Trash = result;
551         });
552     }
553
554     [Fact(Skip = "performance test")]
555     public static void GetSourceTest()
556     {
557         using (var scope = new TempLinksTestScope())
558         {
559             var links = scope.Links;
560             ConsoleHelpers.Debug("Testing GetSource function with {0} Iterations.",
561 ↪ Iterations);
562
563             ulong counter = 0;

```

```

562 //var firstLink = links.First();
563 // Создаём одну связь, из которой будет производить считывание
564 var firstLink = links.Create();
565
566
567 var sw = Stopwatch.StartNew();
568
569 // Тестируем саму функцию
570 for (ulong i = 0; i < Iterations; i++)
571 {
572     counter += links.GetSource(firstLink);
573 }
574
575 var elapsedTime = sw.Elapsed;
576
577 var iterationsPerSecond = Iterations / elapsedTime.TotalSeconds;
578
579 // Удаляем связь, из которой производилось считывание
580 links.Delete(firstLink);
581
582 ConsoleHelpers.Debug(
583     "{0} Iterations of GetSource function done in {1} ({2} Iterations per
    ↪ second), counter result: {3}",
    Iterations, elapsedTime, (long)iterationsPerSecond, counter);
584 }
585
586 }
587
588 [Fact(Skip = "performance test")]
589 public static void GetSourceInParallel()
590 {
591     using (var scope = new TempLinksTestScope())
592     {
593         var links = scope.Links;
594         ConsoleHelpers.Debug("Testing GetSource function with {0} Iterations in
    ↪ parallel.", Iterations);
595
596         long counter = 0;
597
598         //var firstLink = links.First();
599         var firstLink = links.Create();
600
601         var sw = Stopwatch.StartNew();
602
603         // Тестируем саму функцию
604         Parallel.For(0, Iterations, x =>
605         {
606             Interlocked.Add(ref counter, (long)links.GetSource(firstLink));
607             //Interlocked.Increment(ref counter);
608         });
609
610         var elapsedTime = sw.Elapsed;
611
612         var iterationsPerSecond = Iterations / elapsedTime.TotalSeconds;
613
614         links.Delete(firstLink);
615
616         ConsoleHelpers.Debug(
617             "{0} Iterations of GetSource function done in {1} ({2} Iterations per
    ↪ second), counter result: {3}",
            Iterations, elapsedTime, (long)iterationsPerSecond, counter);
618     }
619 }
620
621
622 [Fact(Skip = "performance test")]
623 public static void TestGetTarget()
624 {
625     using (var scope = new TempLinksTestScope())
626     {
627         var links = scope.Links;
628         ConsoleHelpers.Debug("Testing GetTarget function with {0} Iterations.",
    ↪ Iterations);
629
630         ulong counter = 0;
631
632         //var firstLink = links.First();
633         var firstLink = links.Create();
634
635         var sw = Stopwatch.StartNew();
636

```

```

637     for (ulong i = 0; i < Iterations; i++)
638     {
639         counter += links.GetTarget(firstLink);
640     }
641
642     var elapsedTime = sw.Elapsed;
643
644     var iterationsPerSecond = Iterations / elapsedTime.TotalSeconds;
645
646     links.Delete(firstLink);
647
648     ConsoleHelpers.Debug(
649         "{0} Iterations of GetTarget function done in {1} ({2} Iterations per
        ↳ second), counter result: {3}",
        Iterations, elapsedTime, (long)iterationsPerSecond, counter);
650
651     }
652 }
653
654 [Fact(Skip = "performance test")]
655 public static void TestGetTargetInParallel()
656 {
657     using (var scope = new TempLinksTestScope())
658     {
659         var links = scope.Links;
660         ConsoleHelpers.Debug("Testing GetTarget function with {0} Iterations in
        ↳ parallel.", Iterations);
661
662         long counter = 0;
663
664         //var firstLink = links.First();
665         var firstLink = links.Create();
666
667         var sw = Stopwatch.StartNew();
668
669         Parallel.For(0, Iterations, x =>
670         {
671             Interlocked.Add(ref counter, (long)links.GetTarget(firstLink));
672             //Interlocked.Increment(ref counter);
673         });
674
675         var elapsedTime = sw.Elapsed;
676
677         var iterationsPerSecond = Iterations / elapsedTime.TotalSeconds;
678
679         links.Delete(firstLink);
680
681         ConsoleHelpers.Debug(
682             "{0} Iterations of GetTarget function done in {1} ({2} Iterations per
            ↳ second), counter result: {3}",
            Iterations, elapsedTime, (long)iterationsPerSecond, counter);
683     }
684 }
685
686 // TODO: Заполнить базу данных перед тестом
687 /*
688 [Fact]
689 public void TestRandomSearchFixed()
690 {
691     var tempFilename = Path.GetTempFileName();
692
693     using (var links = new Platform.Links.Data.Core.Doublets.Links(tempFilename,
694 ↳ DefaultLinksSizeStep))
695     {
696         long iterations = 64 * 1024 * 1024 /
697 ↳ Platform.Links.Data.Core.Doublets.Links.LinkSizeInBytes;
698
699         ulong counter = 0;
700         var maxLink = links.Total;
701
702         ConsoleHelpers.Debug("Testing Random Search with {0} Iterations.", iterations);
703
704         var sw = Stopwatch.StartNew();
705
706         for (var i = iterations; i > 0; i--)
707         {
708             var source =
709 ↳ RandomHelpers.DefaultFactory.NextUInt64(LinksConstants.MinPossibleIndex, maxLink);
710             var target =
711 ↳ RandomHelpers.DefaultFactory.NextUInt64(LinksConstants.MinPossibleIndex, maxLink);

```



```

709         counter += links.Search(source, target);
710     }
711
712     var elapsedTime = sw.Elapsed;
713
714     var iterationsPerSecond = iterations / elapsedTime.TotalSeconds;
715
716     ConsoleHelpers.Debug("{0} Iterations of Random Search done in {1} ({2}
↵ Iterations per second), c: {3}", iterations, elapsedTime, (long)iterationsPerSecond,
↵ counter);
718 }
719
720     File.Delete(tempFilename);
721 }*/
722
723 [Fact(Skip = "useless: 0(0), was dependent on creation tests")]
724 public static void TestRandomSearchAll()
725 {
726     using (var scope = new TempLinksTestScope())
727     {
728         var links = scope.Links;
729         ulong counter = 0;
730
731         var maxLink = links.Count();
732
733         var iterations = links.Count();
734
735         ConsoleHelpers.Debug("Testing Random Search with {0} Iterations.",
↵ links.Count());
736
737         var sw = Stopwatch.StartNew();
738
739         for (var i = iterations; i > 0; i--)
740         {
741             var linksAddressRange = new
↵ Range<ulong>(_constants.InternalReferencesRange.Minimum, maxLink);
742
743             var source = RandomHelpers.Default.NextUInt64(linksAddressRange);
744             var target = RandomHelpers.Default.NextUInt64(linksAddressRange);
745
746             counter += links.SearchOrDefault(source, target);
747         }
748
749         var elapsedTime = sw.Elapsed;
750
751         var iterationsPerSecond = iterations / elapsedTime.TotalSeconds;
752
753         ConsoleHelpers.Debug("{0} Iterations of Random Search done in {1} ({2}
↵ Iterations per second), c: {3}",
↵ iterations, elapsedTime, (long)iterationsPerSecond, counter);
754     }
755 }
756
757 [Fact(Skip = "useless: 0(0), was dependent on creation tests")]
758 public static void TestEach()
759 {
760     using (var scope = new TempLinksTestScope())
761     {
762         var links = scope.Links;
763
764         var counter = new Counter<IList<ulong>, ulong>(links.Constants.Continue);
765
766         ConsoleHelpers.Debug("Testing Each function.");
767
768         var sw = Stopwatch.StartNew();
769
770         links.Each(counter.IncrementAndReturnTrue);
771
772         var elapsedTime = sw.Elapsed;
773
774         var linksPerSecond = counter.Count / elapsedTime.TotalSeconds;
775
776         ConsoleHelpers.Debug("{0} Iterations of Each's handler function done in {1} ({2}
↵ links per second)",
↵ counter, elapsedTime, (long)linksPerSecond);
777     }
778 }
779
780 }
781
782 /*

```

```

783     [Fact]
784     public static void TestForeach()
785     {
786         var tempFilename = Path.GetTempFileName();
787
788         using (var links = new Platform.Links.Data.Core.Doublets.Links(tempFilename,
↵ DefaultLinksSizeStep))
789         {
790             ulong counter = 0;
791
792             ConsoleHelpers.Debug("Testing foreach through links.");
793
794             var sw = Stopwatch.StartNew();
795
796             //foreach (var link in links)
797             //{
798                 //    counter++;
799             //}
800
801             var elapsedTime = sw.Elapsed;
802
803             var linksPerSecond = (double)counter / elapsedTime.TotalSeconds;
804
805             ConsoleHelpers.Debug("{0} Iterations of Foreach's handler block done in {1} ({2}
↵ links per second)", counter, elapsedTime, (long)linksPerSecond);
806         }
807
808         File.Delete(tempFilename);
809     }
810     */
811
812     /*
813     [Fact]
814     public static void TestParallelForeach()
815     {
816         var tempFilename = Path.GetTempFileName();
817
818         using (var links = new Platform.Links.Data.Core.Doublets.Links(tempFilename,
↵ DefaultLinksSizeStep))
819         {
820             long counter = 0;
821
822             ConsoleHelpers.Debug("Testing parallel foreach through links.");
823
824             var sw = Stopwatch.StartNew();
825
826             //Parallel.ForEach((IEnumerable<ulong>)links, x =>
827             //{
828                 //    Interlocked.Increment(ref counter);
829             //});
830
831             var elapsedTime = sw.Elapsed;
832
833             var linksPerSecond = (double)counter / elapsedTime.TotalSeconds;
834
835             ConsoleHelpers.Debug("{0} Iterations of Parallel Foreach's handler block done in
↵ {1} ({2} links per second)", counter, elapsedTime, (long)linksPerSecond);
836         }
837
838         File.Delete(tempFilename);
839     }
840     */
841
842     [Fact(Skip = "performance test")]
843     public static void Create64BillionLinks()
844     {
845         using (var scope = new TempLinksTestScope())
846         {
847             var links = scope.Links;
848             var linksBeforeTest = links.Count();
849
850             long linksToCreate = 64 * 1024 * 1024 /
↵ UInt64ResizableDirectMemoryLinks.LinkSizeInBytes;
851
852             ConsoleHelpers.Debug("Creating {0} links.", linksToCreate);
853
854             var elapsedTime = Performance.Measure(() =>
855             {
856                 for (long i = 0; i < linksToCreate; i++)

```

```

858         {
859             links.Create();
860         }
861     });
862
863     var linksCreated = links.Count() - linksBeforeTest;
864     var linksPerSecond = linksCreated / elapsedTime.TotalSeconds;
865
866     ConsoleHelpers.Debug("Current links count: {0}.", links.Count());
867
868     ConsoleHelpers.Debug("{0} links created in {1} ({2} links per second)",
869         ↪ linksCreated, elapsedTime,
870         (long)linksPerSecond);
871 }
872
873 [Fact(Skip = "performance test")]
874 public static void Create64BillionLinksInParallel()
875 {
876     using (var scope = new TempLinksTestScope())
877     {
878         var links = scope.Links;
879         var linksBeforeTest = links.Count();
880
881         var sw = Stopwatch.StartNew();
882
883         long linksToCreate = 64 * 1024 * 1024 /
884             ↪ UInt64ResizableDirectMemoryLinks.LinkSizeInBytes;
885
886         ConsoleHelpers.Debug("Creating {0} links in parallel.", linksToCreate);
887
888         Parallel.For(0, linksToCreate, x => links.Create());
889
890         var elapsedTime = sw.Elapsed;
891
892         var linksCreated = links.Count() - linksBeforeTest;
893         var linksPerSecond = linksCreated / elapsedTime.TotalSeconds;
894
895         ConsoleHelpers.Debug("{0} links created in {1} ({2} links per second)",
896             ↪ linksCreated, elapsedTime,
897             (long)linksPerSecond);
898     }
899
900     [Fact(Skip = "useless: 0(0), was dependent on creation tests")]
901     public static void TestDeletionOfAllLinks()
902     {
903         using (var scope = new TempLinksTestScope())
904         {
905             var links = scope.Links;
906             var linksBeforeTest = links.Count();
907
908             ConsoleHelpers.Debug("Deleting all links");
909
910             var elapsedTime = Performance.Measure(links.DeleteAll);
911
912             var linksDeleted = linksBeforeTest - links.Count();
913             var linksPerSecond = linksDeleted / elapsedTime.TotalSeconds;
914
915             ConsoleHelpers.Debug("{0} links deleted in {1} ({2} links per second)",
916                 ↪ linksDeleted, elapsedTime,
917                 (long)linksPerSecond);
918         }
919     }
920 }
921 }

```

1.114 ./Platform.Data.Doublets.Tests/UnaryNumberConvertersTests.cs

```

1  using Xunit;
2  using Platform.Random;
3  using Platform.Data.Doublets.Numbers.Unary;
4
5  namespace Platform.Data.Doublets.Tests
6  {
7      public static class UnaryNumberConvertersTests
8      {
9          [Fact]
10         public static void ConvertersTest()

```

```

11 {
12     using (var scope = new TempLinksTestScope())
13     {
14         const int N = 10;
15         var links = scope.Links;
16         var meaningRoot = links.CreatePoint();
17         var one = links.CreateAndUpdate(meaningRoot, links.Constants.Itself);
18         var powerOf2ToUnaryNumberConverter = new
19             ↪ PowerOf2ToUnaryNumberConverter<ulong>(links, one);
20         var toUnaryNumberConverter = new AddressToUnaryNumberConverter<ulong>(links,
21             ↪ powerOf2ToUnaryNumberConverter);
22         var random = new System.Random(0);
23         ulong[] numbers = new ulong[N];
24         ulong[] unaryNumbers = new ulong[N];
25         for (int i = 0; i < N; i++)
26         {
27             numbers[i] = random.NextUInt64();
28             unaryNumbers[i] = toUnaryNumberConverter.Convert(numbers[i]);
29         }
30         var fromUnaryNumberConverterUsingOrOperation = new
31             ↪ UnaryNumberToAddressOrOperationConverter<ulong>(links,
32             ↪ powerOf2ToUnaryNumberConverter);
33         var fromUnaryNumberConverterUsingAddOperation = new
34             ↪ UnaryNumberToAddressAddOperationConverter<ulong>(links, one);
35         for (int i = 0; i < N; i++)
36         {
37             Assert.Equal(numbers[i],
38                 ↪ fromUnaryNumberConverterUsingOrOperation.Convert(unaryNumbers[i]));
39             Assert.Equal(numbers[i],
40                 ↪ fromUnaryNumberConverterUsingAddOperation.Convert(unaryNumbers[i]));
41         }
42     }
43 }
44 }
45 }

```

1.115 ./Platform.Data.Doublets.Tests/UnicodeConvertersTests.cs

```

1 using Xunit;
2 using Platform.Converters;
3 using Platform.Memory;
4 using Platform.Reflection;
5 using Platform.Scopes;
6 using Platform.Data.Numbers.Raw;
7 using Platform.Data.Doublets.Incremeters;
8 using Platform.Data.Doublets.Numbers.Unary;
9 using Platform.Data.Doublets.PropertyOperators;
10 using Platform.Data.Doublets.Sequences.Converters;
11 using Platform.Data.Doublets.Sequences.Indexes;
12 using Platform.Data.Doublets.Sequences.Walkers;
13 using Platform.Data.Doublets.Unicode;
14 using Platform.Data.Doublets.ResizableDirectMemory.Generic;
15
16 namespace Platform.Data.Doublets.Tests
17 {
18     public static class UnicodeConvertersTests
19     {
20         [Fact]
21         public static void CharAndUnaryNumberUnicodeSymbolConvertersTest()
22         {
23             using (var scope = new TempLinksTestScope())
24             {
25                 var links = scope.Links;
26                 var meaningRoot = links.CreatePoint();
27                 var one = links.CreateAndUpdate(meaningRoot, links.Constants.Itself);
28                 var powerOf2ToUnaryNumberConverter = new
29                     ↪ PowerOf2ToUnaryNumberConverter<ulong>(links, one);
30                 var addressToUnaryNumberConverter = new
31                     ↪ AddressToUnaryNumberConverter<ulong>(links, powerOf2ToUnaryNumberConverter);
32                 var unaryNumberToAddressConverter = new
33                     ↪ UnaryNumberToAddressOrOperationConverter<ulong>(links,
34                     ↪ powerOf2ToUnaryNumberConverter);
35                 TestCharAndUnicodeSymbolConverters(links, meaningRoot,
36                     ↪ addressToUnaryNumberConverter, unaryNumberToAddressConverter);
37             }
38         }
39
40         [Fact]
41         public static void CharAndRawNumberUnicodeSymbolConvertersTest()
42         {

```

```

38     using (var scope = new Scope<Types<HeapResizableDirectMemory,
39         ↳ ResizableDirectMemoryLinks<ulong>>>())
40     {
41         var links = scope.Use<ILinks<ulong>>();
42         var meaningRoot = links.CreatePoint();
43         var addressToRawNumberConverter = new AddressToRawNumberConverter<ulong>();
44         var rawNumberToAddressConverter = new RawNumberToAddressConverter<ulong>();
45         TestCharAndUnicodeSymbolConverters(links, meaningRoot,
46             ↳ addressToRawNumberConverter, rawNumberToAddressConverter);
47     }
48
49 private static void TestCharAndUnicodeSymbolConverters(ILinks<ulong> links, ulong
50     ↳ meaningRoot, IConverter<ulong> addressToNumberConverter, IConverter<ulong>
51     ↳ numberToAddressConverter)
52 {
53     var unicodeSymbolMarker = links.CreateAndUpdate(meaningRoot, links.Constants.Itself);
54     var charToUnicodeSymbolConverter = new CharToUnicodeSymbolConverter<ulong>(links,
55         ↳ addressToNumberConverter, unicodeSymbolMarker);
56     var originalCharacter = 'H';
57     var characterLink = charToUnicodeSymbolConverter.Convert(originalCharacter);
58     var unicodeSymbolCriterionMatcher = new UnicodeSymbolCriterionMatcher<ulong>(links,
59         ↳ unicodeSymbolMarker);
60     var unicodeSymbolToCharConverter = new UnicodeSymbolToCharConverter<ulong>(links,
61         ↳ numberToAddressConverter, unicodeSymbolCriterionMatcher);
62     var resultingCharacter = unicodeSymbolToCharConverter.Convert(characterLink);
63     Assert.Equal(originalCharacter, resultingCharacter);
64 }
65
66 [Fact]
67 public static void StringAndUnicodeSequenceConvertersTest()
68 {
69     using (var scope = new TempLinksTestScope())
70     {
71         var links = scope.Links;
72
73         var itself = links.Constants.Itself;
74
75         var meaningRoot = links.CreatePoint();
76         var unaryOne = links.CreateAndUpdate(meaningRoot, itself);
77         var unicodeSymbolMarker = links.CreateAndUpdate(meaningRoot, itself);
78         var unicodeSequenceMarker = links.CreateAndUpdate(meaningRoot, itself);
79         var frequencyMarker = links.CreateAndUpdate(meaningRoot, itself);
80         var frequencyPropertyMarker = links.CreateAndUpdate(meaningRoot, itself);
81
82         var powerOf2ToUnaryNumberConverter = new
83             ↳ PowerOf2ToUnaryNumberConverter<ulong>(links, unaryOne);
84         var addressToUnaryNumberConverter = new
85             ↳ AddressToUnaryNumberConverter<ulong>(links, powerOf2ToUnaryNumberConverter);
86         var charToUnicodeSymbolConverter = new
87             ↳ CharToUnicodeSymbolConverter<ulong>(links, addressToUnaryNumberConverter,
88             ↳ unicodeSymbolMarker);
89
90         var unaryNumberToAddressConverter = new
91             ↳ UnaryNumberToAddressOrOperationConverter<ulong>(links,
92             ↳ powerOf2ToUnaryNumberConverter);
93         var unaryNumberIncrementer = new UnaryNumberIncrementer<ulong>(links, unaryOne);
94         var frequencyIncrementer = new FrequencyIncrementer<ulong>(links,
95             ↳ frequencyMarker, unaryOne, unaryNumberIncrementer);
96         var frequencyPropertyOperator = new PropertyOperator<ulong>(links,
97             ↳ frequencyPropertyMarker, frequencyMarker);
98         var index = new FrequencyIncrementingSequenceIndex<ulong>(links,
99             ↳ frequencyPropertyOperator, frequencyIncrementer);
100        var linkToItsFrequencyNumberConverter = new
101            ↳ LinkToItsFrequencyNumberConverter<ulong>(links, frequencyPropertyOperator,
102            ↳ unaryNumberToAddressConverter);
103        var sequenceToItsLocalElementLevelsConverter = new
104            ↳ SequenceToItsLocalElementLevelsConverter<ulong>(links,
105            ↳ linkToItsFrequencyNumberConverter);
106        var optimalVariantConverter = new OptimalVariantConverter<ulong>(links,
107            ↳ sequenceToItsLocalElementLevelsConverter);
108
109        var stringToUnicodeSequenceConverter = new
110            ↳ StringToUnicodeSequenceConverter<ulong>(links, charToUnicodeSymbolConverter,
111            ↳ index, optimalVariantConverter, unicodeSequenceMarker);
112
113        var originalString = "Hello";

```

```

92     var unicodeSequenceLink =
93         ↳ stringToUnicodeSequenceConverter.Convert(originalString);
94
95     var unicodeSymbolCriterionMatcher = new
96         ↳ UnicodeSymbolCriterionMatcher<ulong>(links, unicodeSymbolMarker);
97     var unicodeSymbolToCharConverter = new
98         ↳ UnicodeSymbolToCharConverter<ulong>(links, unaryNumberToAddressConverter,
99         ↳ unicodeSymbolCriterionMatcher);
100
101     var unicodeSequenceCriterionMatcher = new
102         ↳ UnicodeSequenceCriterionMatcher<ulong>(links, unicodeSequenceMarker);
103
104     var sequenceWalker = new LeveledSequenceWalker<ulong>(links,
105         ↳ unicodeSymbolCriterionMatcher.IsMatched);
106
107     var unicodeSequenceToStringConverter = new
108         ↳ UnicodeSequenceToStringConverter<ulong>(links,
109         ↳ unicodeSequenceCriterionMatcher, sequenceWalker,
110         ↳ unicodeSymbolToCharConverter);
111
112     var resultingString =
113         ↳ unicodeSequenceToStringConverter.Convert(unicodeSequenceLink);
114
115     Assert.Equal(originalString, resultingString);
116 }
117 }
118 }
119 }
120 }

```

Index

- ./Platform.Data.Doublets.Tests/ComparisonTests.cs, 148
- ./Platform.Data.Doublets.Tests/EqualityTests.cs, 149
- ./Platform.Data.Doublets.Tests/GenericLinksTests.cs, 150
- ./Platform.Data.Doublets.Tests/LinksConstantsTests.cs, 151
- ./Platform.Data.Doublets.Tests/OptimalVariantSequenceTests.cs, 151
- ./Platform.Data.Doublets.Tests/ReadSequenceTests.cs, 154
- ./Platform.Data.Doublets.Tests/ResizableDirectMemoryLinksTests.cs, 155
- ./Platform.Data.Doublets.Tests/ScopeTests.cs, 156
- ./Platform.Data.Doublets.Tests/SequencesTests.cs, 157
- ./Platform.Data.Doublets.Tests/TempLinksTestScope.cs, 172
- ./Platform.Data.Doublets.Tests/TestExtensions.cs, 172
- ./Platform.Data.Doublets.Tests/UInt64LinksTests.cs, 175
- ./Platform.Data.Doublets.Tests/UnaryNumberConvertersTests.cs, 187
- ./Platform.Data.Doublets.Tests/UnicodeConvertersTests.cs, 188
- ./Platform.Data.Doublets/Decorators/LinksCascadeUniquenessAndUsagesResolver.cs, 1
- ./Platform.Data.Doublets/Decorators/LinksCascadeUsagesResolver.cs, 1
- ./Platform.Data.Doublets/Decorators/LinksDecoratorBase.cs, 1
- ./Platform.Data.Doublets/Decorators/LinksDisposableDecoratorBase.cs, 2
- ./Platform.Data.Doublets/Decorators/LinksInnerReferenceExistenceValidator.cs, 2
- ./Platform.Data.Doublets/Decorators/LinksItselfConstantToSelfReferenceResolver.cs, 3
- ./Platform.Data.Doublets/Decorators/LinksNonExistentDependenciesCreator.cs, 4
- ./Platform.Data.Doublets/Decorators/LinksNullConstantToSelfReferenceResolver.cs, 4
- ./Platform.Data.Doublets/Decorators/LinksUniquenessResolver.cs, 4
- ./Platform.Data.Doublets/Decorators/LinksUniquenessValidator.cs, 5
- ./Platform.Data.Doublets/Decorators/LinksUsagesValidator.cs, 5
- ./Platform.Data.Doublets/Decorators/NonNullContentsLinkDeletionResolver.cs, 6
- ./Platform.Data.Doublets/Decorators/UInt64Links.cs, 6
- ./Platform.Data.Doublets/Decorators/UniLinks.cs, 7
- ./Platform.Data.Doublets/Doublet.cs, 12
- ./Platform.Data.Doublets/DoubletComparer.cs, 12
- ./Platform.Data.Doublets/ILinks.cs, 13
- ./Platform.Data.Doublets/ILinksExtensions.cs, 13
- ./Platform.Data.Doublets/ISynchronizedLinks.cs, 24
- ./Platform.Data.Doublets/Incrementers/FrequencyIncrementer.cs, 24
- ./Platform.Data.Doublets/Incrementers/UnaryNumberIncrementer.cs, 25
- ./Platform.Data.Doublets/Link.cs, 26
- ./Platform.Data.Doublets/LinkExtensions.cs, 29
- ./Platform.Data.Doublets/LinksOperatorBase.cs, 29
- ./Platform.Data.Doublets/Numbers/Unary/AddressToUnaryNumberConverter.cs, 29
- ./Platform.Data.Doublets/Numbers/Unary/LinkToItsFrequencyNumberConverter.cs, 30
- ./Platform.Data.Doublets/Numbers/Unary/PowerOf2ToUnaryNumberConverter.cs, 30
- ./Platform.Data.Doublets/Numbers/Unary/UnaryNumberToAddressAddOperationConverter.cs, 31
- ./Platform.Data.Doublets/Numbers/Unary/UnaryNumberToAddressOrOperationConverter.cs, 32
- ./Platform.Data.Doublets/PropertyOperators/PropertiesOperator.cs, 33
- ./Platform.Data.Doublets/PropertyOperators/PropertyOperator.cs, 34
- ./Platform.Data.Doublets/ResizableDirectMemory/Generic/LinksAvlBalancedTreeMethodsBase.cs, 35
- ./Platform.Data.Doublets/ResizableDirectMemory/Generic/LinksSizeBalancedTreeMethodsBase.cs, 39
- ./Platform.Data.Doublets/ResizableDirectMemory/Generic/LinksSourcesAvlBalancedTreeMethods.cs, 42
- ./Platform.Data.Doublets/ResizableDirectMemory/Generic/LinksSourcesSizeBalancedTreeMethods.cs, 43
- ./Platform.Data.Doublets/ResizableDirectMemory/Generic/LinksTargetsAvlBalancedTreeMethods.cs, 44
- ./Platform.Data.Doublets/ResizableDirectMemory/Generic/LinksTargetsSizeBalancedTreeMethods.cs, 45
- ./Platform.Data.Doublets/ResizableDirectMemory/Generic/ResizableDirectMemoryLinks.cs, 46
- ./Platform.Data.Doublets/ResizableDirectMemory/Generic/ResizableDirectMemoryLinksBase.cs, 47
- ./Platform.Data.Doublets/ResizableDirectMemory/Generic/UnusedLinksListMethods.cs, 55
- ./Platform.Data.Doublets/ResizableDirectMemory/ILinksListMethods.cs, 56
- ./Platform.Data.Doublets/ResizableDirectMemory/ILinksTreeMethods.cs, 56
- ./Platform.Data.Doublets/ResizableDirectMemory/LinksHeader.cs, 56
- ./Platform.Data.Doublets/ResizableDirectMemory/RawLink.cs, 57
- ./Platform.Data.Doublets/ResizableDirectMemory/Specific/UInt64LinksAvlBalancedTreeMethodsBase.cs, 57
- ./Platform.Data.Doublets/ResizableDirectMemory/Specific/UInt64LinksSizeBalancedTreeMethodsBase.cs, 59
- ./Platform.Data.Doublets/ResizableDirectMemory/Specific/UInt64LinksSourcesAvlBalancedTreeMethods.cs, 60
- ./Platform.Data.Doublets/ResizableDirectMemory/Specific/UInt64LinksSourcesSizeBalancedTreeMethods.cs, 62
- ./Platform.Data.Doublets/ResizableDirectMemory/Specific/UInt64LinksTargetsAvlBalancedTreeMethods.cs, 63
- ./Platform.Data.Doublets/ResizableDirectMemory/Specific/UInt64LinksTargetsSizeBalancedTreeMethods.cs, 64
- ./Platform.Data.Doublets/ResizableDirectMemory/Specific/UInt64ResizableDirectMemoryLinks.cs, 65
- ./Platform.Data.Doublets/ResizableDirectMemory/Specific/UInt64UnusedLinksListMethods.cs, 66

./Platform.Data.Doublets/Sequences/Converters/BalancedVariantConverter.cs, 67
./Platform.Data.Doublets/Sequences/Converters/CompressingConverter.cs, 68
./Platform.Data.Doublets/Sequences/Converters/LinksListToSequenceConverterBase.cs, 71
./Platform.Data.Doublets/Sequences/Converters/OptimalVariantConverter.cs, 71
./Platform.Data.Doublets/Sequences/Converters/SequenceToltsLocalElementLevelsConverter.cs, 73
./Platform.Data.Doublets/Sequences/CriterionMatchers/DefaultSequenceElementCriterionMatcher.cs, 73
./Platform.Data.Doublets/Sequences/CriterionMatchers/MarkedSequenceCriterionMatcher.cs, 73
./Platform.Data.Doublets/Sequences/DefaultSequenceAppender.cs, 74
./Platform.Data.Doublets/Sequences/DuplicateSegmentsCounter.cs, 75
./Platform.Data.Doublets/Sequences/DuplicateSegmentsProvider.cs, 75
./Platform.Data.Doublets/Sequences/Frequencies/Cache/LinkFrequenciesCache.cs, 77
./Platform.Data.Doublets/Sequences/Frequencies/Cache/LinkFrequency.cs, 79
./Platform.Data.Doublets/Sequences/Frequencies/Cache/LinkToltsFrequencyValueConverter.cs, 80
./Platform.Data.Doublets/Sequences/Frequencies/Counters/MarkedSequenceSymbolFrequencyOneOffCounter.cs, 80
./Platform.Data.Doublets/Sequences/Frequencies/Counters/SequenceSymbolFrequencyOneOffCounter.cs, 80
./Platform.Data.Doublets/Sequences/Frequencies/Counters/TotalMarkedSequenceSymbolFrequencyCounter.cs, 81
./Platform.Data.Doublets/Sequences/Frequencies/Counters/TotalMarkedSequenceSymbolFrequencyOneOffCounter.cs, 81
./Platform.Data.Doublets/Sequences/Frequencies/Counters/TotalSequenceSymbolFrequencyCounter.cs, 82
./Platform.Data.Doublets/Sequences/Frequencies/Counters/TotalSequenceSymbolFrequencyOneOffCounter.cs, 82
./Platform.Data.Doublets/Sequences/HeightProviders/CachedSequenceHeightProvider.cs, 83
./Platform.Data.Doublets/Sequences/HeightProviders/DefaultSequenceRightHeightProvider.cs, 84
./Platform.Data.Doublets/Sequences/HeightProviders/ISequenceHeightProvider.cs, 84
./Platform.Data.Doublets/Sequences/Indexes/CachedFrequencyIncrementingSequenceIndex.cs, 84
./Platform.Data.Doublets/Sequences/Indexes/FrequencyIncrementingSequenceIndex.cs, 85
./Platform.Data.Doublets/Sequences/Indexes/ISequenceIndex.cs, 86
./Platform.Data.Doublets/Sequences/Indexes/SequenceIndex.cs, 86
./Platform.Data.Doublets/Sequences/Indexes/SynchronizedSequenceIndex.cs, 87
./Platform.Data.Doublets/Sequences/Indexes/Unindex.cs, 88
./Platform.Data.Doublets/Sequences/Sequences.Experiments.cs, 88
./Platform.Data.Doublets/Sequences/Sequences.cs, 115
./Platform.Data.Doublets/Sequences/SequencesExtensions.cs, 126
./Platform.Data.Doublets/Sequences/SequencesOptions.cs, 126
./Platform.Data.Doublets/Sequences/Walkers/ISequenceWalker.cs, 129
./Platform.Data.Doublets/Sequences/Walkers/LeftSequenceWalker.cs, 129
./Platform.Data.Doublets/Sequences/Walkers/LeveledSequenceWalker.cs, 130
./Platform.Data.Doublets/Sequences/Walkers/RightSequenceWalker.cs, 131
./Platform.Data.Doublets/Sequences/Walkers/SequenceWalkerBase.cs, 132
./Platform.Data.Doublets/Stacks/Stack.cs, 133
./Platform.Data.Doublets/Stacks/StackExtensions.cs, 134
./Platform.Data.Doublets/SynchronizedLinks.cs, 134
./Platform.Data.Doublets/UInt64LinksExtensions.cs, 135
./Platform.Data.Doublets/UInt64LinksTransactionsLayer.cs, 137
./Platform.Data.Doublets/Unicode/CharToUnicodeSymbolConverter.cs, 143
./Platform.Data.Doublets/Unicode/StringToUnicodeSequenceConverter.cs, 143
./Platform.Data.Doublets/Unicode/UnicodeMap.cs, 144
./Platform.Data.Doublets/Unicode/UnicodeSequenceCriterionMatcher.cs, 146
./Platform.Data.Doublets/Unicode/UnicodeSequenceToStringConverter.cs, 147
./Platform.Data.Doublets/Unicode/UnicodeSymbolCriterionMatcher.cs, 147
./Platform.Data.Doublets/Unicode/UnicodeSymbolToCharConverter.cs, 148