

LinksPlatform's Platform.Data.Doublets Class Library

./Converters/AddressToUnaryNumberConverter.cs

```

1  using System.Collections.Generic;
2  using Platform.Interfaces;
3  using Platform.Reflection;
4  using Platform.Numbers;
5
6  namespace Platform.Data.Doublets.Converters
7  {
8      public class
9          ↳ AddressToUnaryNumberConverter<TLink> :
10         ↳ LinksOperatorBase<TLink>,
11         ↳ IConverter<TLink>
12     {
13         private static readonly
14             ↳ EqualityComparer<TLink>
15             ↳ _equalityComparer =
16             ↳ EqualityComparer<TLink>.Default;
17
18         private readonly IConverter<int, TLink>
19             ↳ _powerOf2ToUnaryNumberConverter;
20
21         public AddressToUnaryNumberConverter(IL
22             ↳ links<TLink> links, IConverter<int,
23             ↳ TLink>
24             ↳ powerOf2ToUnaryNumberConverter) :
25             ↳ base(links) =>
26             ↳ _powerOf2ToUnaryNumberConverter =
27             ↳ powerOf2ToUnaryNumberConverter;
28
29         public TLink Convert(TLink
30             ↳ sourceAddress)
31         {
32             var number = sourceAddress;
33             var target = Links.Constants.Null;
34             for (int i = 0; i < CachedTypeInfo<
35                 ↳ TLink>.BitsLength;
36                 ↳ i++)
37             {
38                 if (_equalityComparer.Equals(Ar
39                     ↳ ithmeticHelpers.And(number,
40                     ↳ Integer<TLink>.One),
41                     ↳ Integer<TLink>.One))
42                 {
43                     target = _equalityComparer.
44                         ↳ Equals(target,
45                         ↳ Links.Constants.Null)
46                     ? _powerOf2ToUnaryNumbe
47                         ↳ rConverter.Convert(
48                         ↳ i)
49                     : Links.GetOrCreate(_po
50                         ↳ werOf2ToUnaryNumber
51                         ↳ Converter.Convert(i
52                         ↳ ),
53                         ↳ target);
54                 }
55                 number = (Integer<TLink>)((ulong
56                     ↳ g)(Integer<TLink>)number >>
57                     ↳ 1); // Should be BitwiseHel
58                     ↳ pers.ShiftRight(number,
59                     ↳ 1);
60                 if (_equalityComparer.Equals(nu
61                     ↳ mber,
62                     ↳ default))
63                 {
64                     break;
65                 }
66             }
67             return target;
68         }
69     }
70 }

```

./Converters/LinkToItsFrequencyNumberConveter.cs

```

1  using System;
2  using System.Collections.Generic;
3  using Platform.Interfaces;
4
5  namespace Platform.Data.Doublets.Converters
6  {
7      public class
8          ↳ LinkToItsFrequencyNumberConveter<TLink>
9          ↳ : LinksOperatorBase<TLink>,
10         ↳ IConverter<Doublet<TLink>, TLink>
11     {

```

```

9         private static readonly
10             ↳ EqualityComparer<TLink>
11             ↳ _equalityComparer =
12             ↳ EqualityComparer<TLink>.Default;
13
14         private readonly
15             ↳ ISpecificPropertyOperator<TLink,
16             ↳ TLink> _frequencyPropertyOperator;
17         private readonly IConverter<TLink>
18             ↳ _unaryNumberToAddressConverter;
19
20         public LinkToItsFrequencyNumberConveter(
21             ↳ ILinks<TLink> links,
22             ↳ ISpecificPropertyOperator<TLink,
23             ↳ TLink>
24             ↳ frequencyPropertyOperator,
25             ↳ IConverter<TLink>
26             ↳ unaryNumberToAddressConverter)
27             : base(links)
28         {
29             _frequencyPropertyOperator =
30                 ↳ frequencyPropertyOperator;
31             _unaryNumberToAddressConverter =
32                 ↳ unaryNumberToAddressConverter;
33         }
34
35         public TLink Convert(Doublet<TLink>
36             ↳ doublet)
37         {
38             var link = Links.SearchOrDefault(do
39                 ↳ ublet.Source,
40                 ↳ doublet.Target);
41             if (_equalityComparer.Equals(link,
42                 ↳ Links.Constants.Null))
43             {
44                 throw new
45                     ↳ ArgumentException($"Link
46                     ↳ with {doublet.Source}
47                     ↳ source and {doublet.Target}
48                     ↳ target not found.",
49                     ↳ nameof(doublet));
50             }
51             var frequency = _frequencyPropertyO
52                 ↳ perator.Get(link);
53             if (_equalityComparer.Equals(freque
54                 ↳ ncy,
55                 ↳ default))
56             {
57                 return default;
58             }
59             var frequencyNumber =
60                 ↳ Links.GetSource(frequency);
61             var number =
62                 ↳ _unaryNumberToAddressConverter.
63                 ↳ Convert(frequencyNumber);
64             return number;
65         }
66     }
67 }

```

./Converters/PowerOf2ToUnaryNumberConverter.cs

```

1  using System;
2  using System.Collections.Generic;
3  using Platform.Interfaces;
4
5  namespace Platform.Data.Doublets.Converters
6  {
7      public class
8          ↳ PowerOf2ToUnaryNumberConverter<TLink> :
9          ↳ LinksOperatorBase<TLink>,
10         ↳ IConverter<int, TLink>
11     {
12         private static readonly
13             ↳ EqualityComparer<TLink>
14             ↳ _equalityComparer =
15             ↳ EqualityComparer<TLink>.Default;
16
17         private readonly TLink[]
18             ↳ _unaryNumberPowersOf2;
19
20         public PowerOf2ToUnaryNumberConverter(I
21             ↳ links<TLink> links, TLink one) :
22             ↳ base(links)
23         {
24             _unaryNumberPowersOf2 = new
25                 ↳ TLink[64];

```

```

16         _unaryNumberPowersOf2[0] = one;
17     }
18
19     public TLink Convert(int power)
20     {
21         if (power < 0 || power >=
22             ↳ _unaryNumberPowersOf2.Length)
23         {
24             throw new ArgumentOutOfRangeException(
25                 ↳ eption(nameof(power)));
26         }
27         if (!_equalityComparer.Equals(_unaryN
28             ↳ yNumberPowersOf2[power],
29             ↳ default))
30         {
31             return _unaryNumberPowersOf2[po
32                 ↳ wer];
33         }
34         var previousPowerOf2 =
35             ↳ Convert(power - 1);
36         var powerOf2 = Links.GetOrCreate(pr
37             ↳ eviousPowerOf2,
38             ↳ previousPowerOf2);
39         _unaryNumberPowersOf2[power] =
40             ↳ powerOf2;
41         return powerOf2;
42     }
43 }
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
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
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2
```

```

26     var source = sourceNumber;
27     var target = Links.Constants.Null;
28     while (!_equalityComparer.Equals(source,
    ↳ urce,
    ↳ Links.Constants.Null))
29     {
30         if (_unaryNumberPowerOf2Indices
    ↳ s.TryGetValue(source, out
    ↳ int powerOf2Index))
31         {
32             source =
    ↳ Links.Constants.Null;
33         }
34         else
35         {
36             powerOf2Index = _unaryNumber
    ↳ rPowerOf2Indices[Links
    ↳ .GetSource(source)];
37             source =
    ↳ Links.GetTarget(source);
38         }
39         target = (Integer<TLink>)((Integer<TLink>)target | 1UL <<
    ↳ powerOf2Index); //
    ↳ MathHelpers.Or(target,
    ↳ MathHelpers.ShiftLeft(One,
    ↳ powerOf2Index))
40     }
41     return target;
42 }
43 }
44 }

```

./Decorators/LinksCascadeDependenciesResolver.cs

```

1  using System.Collections.Generic;
2  using Platform.Collections.Arrays;
3  using Platform.Numbers;
4
5  namespace Platform.Data.Doublets.Decorators
6  {
7      public class
    ↳ LinksCascadeDependenciesResolver<TLink>
    ↳ : LinksDecoratorBase<TLink>
8      {
9          private static readonly
    ↳ EqualityComparer<TLink>
    ↳ _equalityComparer =
    ↳ EqualityComparer<TLink>.Default;
10
11         public LinksCascadeDependenciesResolver
    ↳ (ILinks<TLink> links) : base(links)
    ↳ { }
12
13         public override void Delete(TLink link)
14         {
15             EnsureNoDependenciesOnDelete(link);
16             base.Delete(link);
17         }
18
19         public void
    ↳ EnsureNoDependenciesOnDelete(TLink
    ↳ link)
20         {
21             ulong referencesCount = (Integer<TLink>)Links.Count(Constants.Any,
    ↳ link);
22             var references = ArrayPool.Allocate
    ↳ <TLink>((long)referencesCount);
23             var referencesFiller = new
    ↳ ArrayFiller<TLink,
    ↳ TLink>(references,
    ↳ Constants.Continue);
24             Links.Each(referencesFiller.AddFirst
    ↳ tAndReturnConstant,
    ↳ Constants.Any, link);
25             //references.Sort() // TODO: Решить
    ↳ необходимо ли для корректного
    ↳ порядка отмены операций в
    ↳ транзакциях
26             for (var i = (long)referencesCount
    ↳ - 1; i >= 0; i--)
27             {
28                 if (_equalityComparer.Equals(references[i],
    ↳ link))
29                 {

```

```

30                 continue;
31             }
32             Links.Delete(references[i]);
33         }
34         ArrayPool.Free(references);
35     }
36 }
37 }

```

./Decorators/LinksCascadeUniquenessAndDependenciesResolver.cs

```

1  using System.Collections.Generic;
2  using Platform.Collections.Arrays;
3  using Platform.Numbers;
4
5  namespace Platform.Data.Doublets.Decorators
6  {
7      public class LinksCascadeUniquenessAndDependenciesResolver<TLink> :
    ↳ LinksUniquenessResolver<TLink>
8      {
9          private static readonly
    ↳ EqualityComparer<TLink>
    ↳ _equalityComparer =
    ↳ EqualityComparer<TLink>.Default;
10
11         public LinksCascadeUniquenessAndDependenciesResolver(ILinks<TLink> links)
    ↳ : base(links) { }
12
13         protected override TLink
    ↳ ResolveAddressChangeConflict(TLink
    ↳ oldLinkAddress, TLink
    ↳ newLinkAddress)
14         {
15             // TODO: Very similar to Merge
    ↳ (logic should be reused)
16             ulong referencesAsSourceCount =
    ↳ (Integer<TLink>)Links.Count(Constants.Any, oldLinkAddress,
    ↳ Constants.Any);
17             ulong referencesAsTargetCount =
    ↳ (Integer<TLink>)Links.Count(Constants.Any, Constants.Any,
    ↳ oldLinkAddress);
18             var references =
    ↳ ArrayPool.Allocate<TLink>((long)
    ↳ (referencesAsSourceCount +
    ↳ referencesAsTargetCount));
19             var referencesFiller = new
    ↳ ArrayFiller<TLink,
    ↳ TLink>(references,
    ↳ Constants.Continue);
20             Links.Each(referencesFiller.AddFirstAndReturnConstant,
    ↳ Constants.Any, oldLinkAddress,
    ↳ Constants.Any);
21             Links.Each(referencesFiller.AddFirstAndReturnConstant,
    ↳ Constants.Any, Constants.Any,
    ↳ oldLinkAddress);
22             for (ulong i = 0; i <
    ↳ referencesAsSourceCount; i++)
23             {
24                 var reference = references[i];
25                 if (!_equalityComparer.Equals(reference,
    ↳ oldLinkAddress))
26                 {
27                     Links.Update(reference,
    ↳ newLinkAddress, Links.GetTarget(reference));
28                 }
29             }
30             for (var i =
    ↳ (long)referencesAsSourceCount;
    ↳ i < references.Length; i++)
31             {
32                 var reference = references[i];
33                 if (!_equalityComparer.Equals(reference,
    ↳ oldLinkAddress))
34                 {
35                     Links.Update(reference,
    ↳ Links.GetSource(reference),
    ↳ newLinkAddress);

```

```

36         }
37     }
38     ArrayPool.Free(references);
39     return base.ResolveAddressChangeCon
        ↳ flict(oldLinkAddress,
        ↳ newLinkAddress);
40 }
41 }
42 }

```

./Decorators/LinksDecoratorBase.cs

```

1  using System;
2  using System.Collections.Generic;
3  using Platform.Data.Constants;
4
5  namespace Platform.Data.Doublets.Decorators
6  {
7      public abstract class LinksDecoratorBase<T>
8          ↳ : ILinks<T>
9      {
10         public LinksCombinedConstants<T, T,
11             ↳ int> Constants { get; }
12
13         public readonly ILinks<T> Links;
14
15         protected LinksDecoratorBase(ILinks<T>
16             ↳ links)
17         {
18             Links = links;
19             Constants = links.Constants;
20
21         public virtual T Count(IList<T>
22             ↳ restriction) =>
23             ↳ Links.Count(restriction);
24
25         public virtual T Each(Func<IList<T>, T>
26             ↳ handler, IList<T> restrictions) =>
27             ↳ Links.Each(handler, restrictions);
28
29         public virtual T Create() =>
30             ↳ Links.Create();
31
32         public virtual T Update(IList<T>
33             ↳ restrictions) =>
34             ↳ Links.Update(restrictions);
35
36         public virtual void Delete(T link) =>
37             ↳ Links.Delete(link);
38     }
39 }

```

./Decorators/LinksDependenciesValidator.cs

```

1  using System.Collections.Generic;
2
3  namespace Platform.Data.Doublets.Decorators
4  {
5      public class LinksDependenciesValidator<T>
6          ↳ : LinksDecoratorBase<T>
7      {
8         public LinksDependenciesValidator(ILink
9             ↳ s<T> links) : base(links) {
10         }
11
12         public override T Update(IList<T>
13             ↳ restrictions)
14         {
15             Links.EnsureNoDependencies(restrict
16                 ↳ ions[Constants.IndexPart]);
17             return base.Update(restrictions);
18         }
19
20         public override void Delete(T link)
21         {
22             Links.EnsureNoDependencies(link);
23             base.Delete(link);
24         }
25     }
26 }

```

./Decorators/LinksDisposableDecoratorBase.cs

```

1  using System;
2  using System.Collections.Generic;
3  using Platform.Disposables;
4  using Platform.Data.Constants;
5
6  namespace Platform.Data.Doublets.Decorators

```

```

7  {
8      public abstract class
9          ↳ LinksDisposableDecoratorBase<T> :
9          ↳ DisposableBase, ILinks<T>
10     {
11         public LinksCombinedConstants<T, T,
12             ↳ int> Constants { get; }
13
14         public readonly ILinks<T> Links;
15
16         protected LinksDisposableDecoratorBase(
17             ↳ ILinks<T>
18             ↳ links)
19         {
20             Links = links;
21             Constants = links.Constants;
22
23         public virtual T Count(IList<T>
24             ↳ restriction) =>
25             ↳ Links.Count(restriction);
26
27         public virtual T Each(Func<IList<T>, T>
28             ↳ handler, IList<T> restrictions) =>
29             ↳ Links.Each(handler, restrictions);
30
31         public virtual T Create() =>
32             ↳ Links.Create();
33
34         public virtual T Update(IList<T>
35             ↳ restrictions) =>
36             ↳ Links.Update(restrictions);
37
38         public virtual void Delete(T link) =>
39             ↳ Links.Delete(link);
40
41         protected override bool
42             ↳ AllowMultipleDisposeCalls => true;
43
44         protected override void
45             ↳ DisposeCore(bool manual, bool
46             ↳ wasDisposed) =>
47             ↳ Disposable.TryDispose(Links);
48     }
49 }

```

./Decorators/LinksInnerReferenceValidator.cs

```

1  using System;
2  using System.Collections.Generic;
3
4  namespace Platform.Data.Doublets.Decorators
5  {
6      // TODO: Make
7      ↳ LinksExternalReferenceValidator. A
8      ↳ layer that checks each link to exist or
9      ↳ to be external (hybrid link's raw
10     ↳ number).
11     public class
12         ↳ LinksInnerReferenceValidator<T> :
13         ↳ LinksDecoratorBase<T>
14     {
15         public LinksInnerReferenceValidator(ILi
16             ↳ nks<T> links) : base(links) {
17         }
18
19         public override T Each(Func<IList<T>,
20             ↳ T> handler, IList<T> restrictions)
21         {
22             Links.EnsureInnerReferenceExists(re
23                 ↳ strictions,
24                 ↳ nameof(restrictions));
25             return base.Each(handler,
26                 ↳ restrictions);
27         }
28
29         public override T Count(IList<T>
30             ↳ restriction)
31         {
32             Links.EnsureInnerReferenceExists(re
33                 ↳ striction,
34                 ↳ nameof(restriction));
35             return base.Count(restriction);
36         }
37
38         public override T Update(IList<T>
39             ↳ restrictions)
40         {

```

```

25         // TODO: Possible values: null,
        ↪ ExistingLink or NonExistentHybr
        ↪ id(ExternalReference)
26 Links.EnsureInnerReferenceExists(re
        ↪ strictions,
        ↪ nameof(restrictions));
27 return base.Update(restrictions);
28 }
29
30 public override void Delete(T link)
31 {
32     // TODO: Решить считать ли такое
        ↪ исключением, или лишь более
        ↪ конкретным требованием?
33 Links.EnsureLinkExists(link,
        ↪ nameof(link));
34 base.Delete(link);
35 }
36 }
37 }

```

./Decorators/LinksNonExistentReferencesCreator.cs

```

1 using System.Collections.Generic;
2
3 namespace Platform.Data.Doublets.Decorators
4 {
5     /// <remarks>
6     /// Not practical if newSource and
        ↪ newTarget are too big.
7     /// To be able to use practical version we
        ↪ should allow to create link at any
        ↪ specific location inside
        ↪ ResizableDirectMemoryLinks.
8     /// This in turn will require to implement
        ↪ not a list of empty links, but a list
        ↪ of ranges to store it more efficiently.
9     /// </remarks>
10 public class
        ↪ LinksNonExistentReferencesCreator<T> :
        ↪ LinksDecoratorBase<T>
11 {
12     public LinksNonExistentReferencesCreato
        ↪ r(ILinks<T> links) : base(links) {
        ↪ }
13
14     public override T Update(IList<T>
        ↪ restrictions)
15     {
16         Links.EnsureCreated(restrictions[Co
        ↪ nstants.SourcePart],
        ↪ restrictions[Constants.TargetPa
        ↪ rt]);
17         return base.Update(restrictions);
18     }
19 }
20 }

```

./Decorators/LinksNullToSelfReferenceResolver.cs

```

1 using System.Collections.Generic;
2
3 namespace Platform.Data.Doublets.Decorators
4 {
5     public class
        ↪ LinksNullToSelfReferenceResolver<TLink>
        ↪ : LinksDecoratorBase<TLink>
6     {
7         private static readonly
        ↪ EqualityComparer<TLink>
        ↪ _equalityComparer =
        ↪ EqualityComparer<TLink>.Default;
8
9         public LinksNullToSelfReferenceResolver
        ↪ (ILinks<TLink> links) : base(links)
        ↪ { }
10
11         public override TLink Create()
12         {
13             var link = base.Create();
14             return Links.Update(link, link,
        ↪ link);
15         }
16
17         public override TLink
        ↪ Update(IList<TLink> restrictions)
18         {

```

```

19         restrictions[Constants.SourcePart]
        = _equalityComparer.Equals(rest
        ↪ rictions[Constants.SourcePart],
        ↪ Constants.Null) ? restrictions[
        ↪ Constants.IndexPart] :
        ↪ restrictions[Constants.SourcePa
        ↪ rt];
20         restrictions[Constants.TargetPart]
        = _equalityComparer.Equals(rest
        ↪ rictions[Constants.TargetPart],
        ↪ Constants.Null) ? restrictions[
        ↪ Constants.IndexPart] :
        ↪ restrictions[Constants.TargetPa
        ↪ rt];
21         return base.Update(restrictions);
22     }
23 }
24 }

```

./Decorators/LinksSelfReferenceResolver.cs

```

1 using System;
2 using System.Collections.Generic;
3
4 namespace Platform.Data.Doublets.Decorators
5 {
6     public class
        ↪ LinksSelfReferenceResolver<TLink> :
        ↪ LinksDecoratorBase<TLink>
7     {
8         private static readonly
        ↪ EqualityComparer<TLink>
        ↪ _equalityComparer =
        ↪ EqualityComparer<TLink>.Default;
9
10         public LinksSelfReferenceResolver(ILink
        ↪ s<TLink> links) : base(links) {
        ↪ }
11
12         public override TLink
        ↪ Each(Func<IList<TLink>, TLink>
        ↪ handler, IList<TLink> restrictions)
13         {
14             if (!_equalityComparer.Equals(Const
        ↪ ants.Any,
        ↪ Constants.Itself)
        ↪ && ((restrictions.Count >
        ↪ Constants.IndexPart) &&
        ↪ _equalityComparer.Equals(restr
        ↪ ictions[Constants.IndexPart],
        ↪ Constants.Itself))
15             || (restrictions.Count >
        ↪ Constants.SourcePart) &&
        ↪ _equalityComparer.Equals(restr
        ↪ ictions[Constants.SourcePart],
        ↪ Constants.Itself))
16             || (restrictions.Count >
        ↪ Constants.TargetPart) &&
        ↪ _equalityComparer.Equals(restr
        ↪ ictions[Constants.TargetPart],
        ↪ Constants.Itself)))
17             {
        ↪         return Constants.Continue;
        ↪     }
        ↪     return base.Each(handler,
        ↪ restrictions);
18 }
19
20 public override TLink
        ↪ Update(IList<TLink> restrictions)
21 {
22     restrictions[Constants.SourcePart]
        = _equalityComparer.Equals(rest
        ↪ rictions[Constants.SourcePart],
        ↪ Constants.Itself) ? restriction
        ↪ s[Constants.IndexPart] :
        ↪ restrictions[Constants.SourcePa
        ↪ rt];
23     restrictions[Constants.TargetPart]
        = _equalityComparer.Equals(rest
        ↪ rictions[Constants.TargetPart],
        ↪ Constants.Itself) ? restriction
        ↪ s[Constants.IndexPart] :
        ↪ restrictions[Constants.TargetPa
        ↪ rt];
24 }
25 }
26 }
27 }

```

```

28         return base.Update(restrictions);
29     }
30 }
31 }

```

./Decorators/LinksUniquenessResolver.cs

```

1 using System.Collections.Generic;
2
3 namespace Platform.Data.Doublets.Decorators
4 {
5     public class LinksUniquenessResolver<TLink>
6         ↳ : LinksDecoratorBase<TLink>
7     {
8         private static readonly
9             ↳ EqualityComparer<TLink>
10             ↳ equalityComparer =
11             ↳ EqualityComparer<TLink>.Default;
12
13     public LinksUniquenessResolver(ILinks<T>
14         ↳ links) : base(links) {
15
16     public override TLink
17         ↳ Update(IList<TLink> restrictions)
18     {
19         var newLinkAddress =
20             ↳ Links.SearchOrDefault(restricti
21             ↳ ons[Constants.SourcePart],
22             ↳ restrictions[Constants.TargetPa
23             ↳ rt]);
24         if (equalityComparer.Equals(newLin
25             ↳ kAddress,
26             ↳ default))
27         {
28             return
29                 ↳ base.Update(restrictions);
30         }
31         return ResolveAddressChangeConflict
32             ↳ (restrictions[Constants.IndexPa
33             ↳ rt],
34             ↳ newLinkAddress);
35     }
36
37     protected virtual TLink
38         ↳ ResolveAddressChangeConflict(TLink
39         ↳ oldLinkAddress, TLink
40         ↳ newLinkAddress)
41     {
42         if (Links.Exists(oldLinkAddress))
43         {
44             Delete(oldLinkAddress);
45         }
46         return newLinkAddress;
47     }
48 }
49 }

```

./Decorators/LinksUniquenessValidator.cs

```

1 using System.Collections.Generic;
2
3 namespace Platform.Data.Doublets.Decorators
4 {
5     public class LinksUniquenessValidator<T> :
6         ↳ LinksDecoratorBase<T>
7     {
8     public
9         ↳ LinksUniquenessValidator(ILinks<T>
10         ↳ links) : base(links) { }
11
12     public override T Update(IList<T>
13         ↳ restrictions)
14     {
15         Links.EnsureDoesNotExists(restricti
16             ↳ ons[Constants.SourcePart],
17             ↳ restrictions[Constants.TargetPa
18             ↳ rt]);
19         return base.Update(restrictions);
20     }
21 }

```

./Decorators/NonNullContentsLinkDeletionResolver.cs

```

1 namespace Platform.Data.Doublets.Decorators
2 {
3     public class
4         ↳ NonNullContentsLinkDeletionResolver<T>
5         ↳ : LinksDecoratorBase<T>

```

```

6 {
7     public NonNullContentsLinkDeletionResol
8         ↳ ver(ILinks<T> links) : base(links)
9         ↳ { }
10
11     public override void Delete(T link)
12     {
13         Links.Update(link, Constants.Null,
14             ↳ Constants.Null);
15         base.Delete(link);
16     }
17 }

```

./Decorators/UInt64Links.cs

```

1 using System;
2 using System.Collections.Generic;
3 using Platform.Collections;
4 using Platform.Collections.Arrays;
5
6 namespace Platform.Data.Doublets.Decorators
7 {
8     /// <summary>
9     /// Представляет объект для работы с базой
10     /// данных (файлом) в формате Links
11     /// (массива взаимосвязей).
12     /// </summary>
13     /// <remarks>
14     /// Возможные оптимизации:
15     /// Объединение в одно поле Source и
16     /// Target с уменьшением до 32 бит.
17     /// + меньше объём БД
18     /// - меньше производительность
19     /// - больше ограничение на количество
20     /// связей в БД)
21     /// Ленивое хранение размеров поддеревьев
22     /// (расчитываемое по мере использования БД)
23     /// + меньше объём БД
24     /// - больше сложность
25     ///
26     /// AVL - высота дерева может позволить
27     /// точно рассчитать размер дерева, нет
28     /// необходимости в SBT.
29     /// AVL дерево можно прошить.
30     ///
31     /// Текущее теоретическое ограничение на
32     /// размер связей - long.MaxValue
33     /// Желательно реализовать поддержку
34     /// переключения между деревьями и битовыми
35     /// индексами (битовыми строками) - вариант
36     /// матрицы (выстраиваемой лениво).
37     ///
38     /// Решить отключать ли проверки при
39     /// компиляции под Release. Т.е. исключения
40     /// будут выбрасываться только при #if DEBUG
41     /// </remarks>
42     public class UInt64Links :
43         ↳ LinksDisposableDecoratorBase<ulong>
44     {
45     public UInt64Links(ILinks<ulong> links)
46         ↳ : base(links) { }
47
48     public override ulong
49         ↳ Each(Func<IList<ulong>, ulong>
50         ↳ handler, IList<ulong> restrictions)
51     {
52         this.EnsureLinkIsAnyOrExists(restri
53             ↳ ctions);
54         return Links.Each(handler,
55             ↳ restrictions);
56     }
57
58     public override ulong Create() =>
59         ↳ Links.CreatePoint();
60
61     public override ulong
62         ↳ Update(IList<ulong> restrictions)
63     {
64         if (restrictions.IsNullOrEmpty())
65         {
66             return Constants.Null;
67         }
68         // TODO: Remove usages of these
69         ↳ hacks (these should not be
70         ↳ backwards compatible)

```

```

48     if (restrictions.Count == 2)
49     {
50         return
            ↳ this.Merge(restrictions[0],
            ↳ restrictions[1]);
51     }
52     if (restrictions.Count == 4)
53     {
54         return this.UpdateOrCreateOrGet
            ↳ (restrictions[0],
            ↳ restrictions[1],
            ↳ restrictions[2],
            ↳ restrictions[3]);
55     }
56     // TODO: Looks like this is a
            ↳ common type of exceptions
            ↳ linked with restrictions support
57     if (restrictions.Count != 3)
58     {
59         throw new
            ↳ NotSupportedException();
60     }
61     var updatedLink = restrictions[Consta
            ↳ nts.IndexPart];
62     this.EnsureLinkExists(updatedLink,
            ↳ nameof(Constants.IndexPart));
63     var newSource = restrictions[Consta
            ↳ nts.SourcePart];
64     this.EnsureLinkIsItselfOrExists(new
            ↳ Source,
            ↳ nameof(Constants.SourcePart));
65     var newTarget = restrictions[Consta
            ↳ nts.TargetPart];
66     this.EnsureLinkIsItselfOrExists(new
            ↳ Target,
            ↳ nameof(Constants.TargetPart));
67     var existedLink = Constants.Null;
68     if (newSource != Constants.Itself
            ↳ && newTarget !=
            ↳ Constants.Itself)
69     {
70         existedLink = this.SearchOrDefa
            ↳ ult(newSource,
            ↳ newTarget);
71     }
72     if (existedLink == Constants.Null)
73     {
74         var before =
            ↳ Links.GetLink(updatedLink);
75         if (before[Constants.SourcePart]
            ↳ != newSource ||
            ↳ before[Constants.TargetPart]
            ↳ !=
            ↳ newTarget)
76         {
77             Links.Update(updatedLink,
                ↳ newSource ==
                ↳ Constants.Itself ?
                ↳ updatedLink : newSource,

```

78

79

80

81

82

83

84

85

86

87

88

89

```

        }
        return updatedLink;
    }
    else
    {
        // Replace one link with
        ↳ another (replaced link is
        ↳ deleted, children are
        ↳ updated or deleted), it is
        ↳ actually merge operation
        return this.Merge(updatedLink,
            ↳ existedLink);
    }
}

/// <summary>Удаляет связь с указанным
↳ индексом.</summary>

```

n

```

e
w
T
a
r
g
e
t
=
=
C
o
n
s
t
a
n
t
s
.
I
t
s
e
l
f
?
u
p
d
a
t
e
d
L
i
n
k
:
n
e
w
T
a
r
g
e
t
)
;

```



```

90     /// <param name="link">Индекс удаляемой
91     ↪ связи.</param>
92     public override void Delete(ulong link)
93     {
94         this.EnsureLinkExists(link);
95         Links.Update(link, Constants.Null,
96             ↪ Constants.Null);
97         var referencesCount =
98             ↪ Links.Count(Constants.Any,
99             ↪ link);
100         if (referencesCount > 0)
101         {
102             var references = new
103                 ↪ ulong[referencesCount];
104             var referencesFiller = new
105                 ↪ ArrayFiller<ulong,
106                 ↪ ulong>(references,
107                 ↪ Constants.Continue);
108             Links.Each(referencesFiller.Add,
109                 ↪ FirstAndReturnConstant,
110                 ↪ Constants.Any, link);
111             //references.Sort(); // TODO:
112             ↪ Решить необходимо ли для
113             ↪ корректного порядка отмены
114             ↪ операций в транзакциях
115             for (var i =
116                 ↪ (long)referencesCount - 1;
117                 ↪ i >= 0; i--)
118             {
119                 if (this.Exists(references[
120                     ↪ i]))
121                 {
122                     Delete(references[i]);
123                 }
124             }
125             //else
126             // TODO: Определить почему
127             ↪ здесь есть связи, которых
128             ↪ не существует
129         }
130         Links.Delete(link);
131     }
132 }
133 }
134 }

```

./Decorators/UniLinks.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using Platform.Collections;
5  using Platform.Collections.Arrays;
6  using Platform.Collections.Lists;
7  using Platform.Helpers.Scopes;
8  using Platform.Data.Constants;
9  using Platform.Data.Universal;
10 using System.Collections.ObjectModel;
11
12 namespace Platform.Data.Doublets.Decorators
13 {
14     /// <remarks>
15     /// What does empty pattern (for condition
16     ↪ or substitution) mean? Nothing or
17     ↪ Everything?
18     /// Now we go with nothing. And nothing is
19     ↪ something one, but empty, and cannot be
20     ↪ changed by itself. But can cause
21     ↪ creation (update from nothing) or
22     ↪ deletion (update to nothing).
23     ///
24     /// TODO: Decide to change to IDoubletLinks
25     ↪ or not to change. (Better to create
26     ↪ DefaultUniLinksBase, that contains
27     ↪ logic itself and can be implemented
28     ↪ using both IDoubletLinks and ILinks.)
29     /// </remarks>
30     internal class UniLinks<TLink> :
31         ↪ LinksDecoratorBase<TLink>,
32         ↪ IUniLinks<TLink>
33     {
34         private static readonly
35             ↪ EqualityComparer<TLink>
36             ↪ _equalityComparer =
37             ↪ EqualityComparer<TLink>.Default;
38
39         public UniLinks(ILinks<TLink> links) :
40             ↪ base(links) { }
41     }
42 }

```

```

26 private struct Transition
27 {
28     public IList<TLink> Before;
29     public IList<TLink> After;
30
31     public Transition(IList<TLink>
32         ↪ before, IList<TLink> after)
33     {
34         Before = before;
35         After = after;
36     }
37 }
38
39 public static readonly TLink
40     ↪ NullConstant =
41     ↪ Use<LinksCombinedConstants<TLink,
42     ↪ TLink, int>>.Single.Null;
43 public static readonly
44     ↪ IReadOnlyList<TLink> NullLink = new
45     ↪ ReadOnlyCollection<TLink>(new
46     ↪ List<TLink> { NullConstant,
47     ↪ NullConstant, NullConstant });
48
49 // TODO: Подумать о том, как
50 ↪ реализовать древовидный Restriction
51 ↪ и Substitution (Links-Expression)
52 public TLink Trigger(IList<TLink>
53     ↪ restriction, Func<IList<TLink>,
54     ↪ IList<TLink>, TLink>
55     ↪ matchedHandler, IList<TLink>
56     ↪ substitution, Func<IList<TLink>,
57     ↪ IList<TLink>, TLink>
58     ↪ substitutedHandler)
59 {
60     ///List<Transition> transitions =
61     ↪ null;
62     ///if
63     ↪ (!restriction.IsNullOrEmpty())
64     ///{
65     ///    // Есть причина делать
66     ↪ проход (чтение)
67     ///    if (matchedHandler != null)
68     ///    {
69     ///        if
70     ↪ (!substitution.IsNullOrEmpty())
71     ///        {
72     ///            // restriction => {
73     ↪ 0, 0, 0 } | { 0 } // Create
74     ///            // substitution =>
75     ↪ { itself, 0, 0 } | { itself,
76     ↪ itself, itself } // Create /
77     ↪ Update
78     ///            // substitution =>
79     ↪ { 0, 0, 0 } | { 0 } // Delete
80     ///            transitions = new
81     ↪ List<Transition>();
82     ///            if (Equals(substitu
83     ↪ tion[Constants.IndexPart],
84     ↪ Constants.Null))
85     ///            {
86     ///                // If index is
87     ↪ Null, that means we always
88     ↪ ignore every other value (they
89     ↪ are also Null by definition)
90     ///                var
91     ↪ matchDecision =
92     ↪ matchedHandler(, NullLink);
93     ///                if
94     ↪ (Equals(matchDecision,
95     ↪ Constants.Break))
96     ///                return
97     ↪ false;
98     ///                if
99     ↪ (!Equals(matchDecision,
100     ↪ Constants.Skip))
101     ///                transitions.Add(new
102     ↪ Transition(matchedLink,
103     ↪ newValue));
104     ///            }
105     ///            else
106     ///            {
107     ///                Func<T, bool>
108     ↪ handler;
109     ///                handler = link
110     ↪ =>

```



```

69         {
70         var
        ↪ matchedLink =
        ↪ Memory.GetLinkValue(link);
71         var
        ↪ newValue =
        ↪ Memory.GetLinkValue(link);
72         newValue[Constants.IndexPart] =
        ↪ Constants.Itself;
73         newValue[Constants.SourcePart]
        ↪ = Equals(substitution[Constants
        ↪ .SourcePart], Constants.Itself)
        ↪ ? matchedLink[Constants.IndexPa
        ↪ rt] :
        ↪ substitution[Constants.SourcePa
        ↪ rt];
74         newValue[Constants.TargetPart]
        ↪ = Equals(substitution[Constants
        ↪ .TargetPart], Constants.Itself)
        ↪ ? matchedLink[Constants.IndexPa
        ↪ rt] :
        ↪ substitution[Constants.TargetPa
        ↪ rt];
75         var
        ↪ matchDecision =
        ↪ matchedHandler(matchedLink,
        ↪ newValue);
76         if
        ↪ (Equals(matchDecision,
        ↪ Constants.Break))
77         return
        ↪ false;
78         if
        ↪ (!Equals(matchDecision,
        ↪ Constants.Skip))
79         transitions.Add(new
        ↪ Transition(matchedLink,
        ↪ newValue));
80         return true;
81         };
82         if
        ↪ (!Memory.Each(handler,
        ↪ restriction))
83         return
        ↪ Constants.Break;
84         }
85         }
86         else
87         {
88         Func<T, bool>
        ↪ handler = link =>
89         {
90         var matchedLink
        ↪ = Memory.GetLinkValue(link);
91         var
        ↪ matchDecision =
        ↪ matchedHandler(matchedLink,
        ↪ matchedLink);
92         return
        ↪ !Equals(matchDecision,
        ↪ Constants.Break);
93         };
94         if
        ↪ (!Memory.Each(handler,
        ↪ restriction))
95         return
        ↪ Constants.Break;
96         }
97         }
98         else
99         {
100         if (substitution !=
        ↪ null)
101         {
102         transitions = new
        ↪ List<IList<T>>();
103         Func<T, bool>
        ↪ handler = link =>
104         {
105         var matchedLink
        ↪ = Memory.GetLinkValue(link);
106         transitions.Add(matchedLink);
        ↪ return true;
107         };
108         if
        ↪ (!Memory.Each(handler,
        ↪ restriction))
109         return
        ↪ Constants.Break;
110         }
111         }
112         else
113         {
114         return
        ↪ Constants.Continue;
115         }
116         }
117     }
118     if (substitution != null)
119     {
120         // Есть причина делать
        ↪ замену (запись)
121         if (substitutedHandler !=
        ↪ null)
122         {
123         }
124         else
125         {
126         }
127     }
128     return Constants.Continue;
129
130     //if (restriction.IsNullOrEmpty())
131     //    // Create
132     //{
133         substitution[Constants.IndexP
        ↪ art] =
        ↪ Memory.AllocateLink();
        ↪ Memory.SetLinkValue(substitut
        ↪ ion);
134     //}
135     //else if
        ↪ (substitution.IsNullOrEmpty())
        ↪ // Delete
136     //{
137         Memory.FreeLink(restriction[C
        ↪ onstants.IndexPart]);
138     //}
139     //else if (restriction.EqualTo(subs
        ↪ titution)) // Read or ("repeat"
        ↪ the state) // Each
140     //{
141         // No need to collect links
        ↪ to list
142         // // Skip == Continue
143         // // No need to check
        ↪ substitutedHandler
        ↪ if (!Memory.Each(link =>
        ↪ !Equals(matchedHandler(Memory.G
        ↪ etLinkValue(link)),
        ↪ Constants.Break), restriction))
        ↪ return Constants.Break;
144     //}
145     //else // Update
146     //{
147         //List<IList<T>> matchedLinks
        ↪ = null;
148         // if (matchedHandler != null)
        ↪ {
149             matchedLinks = new
        ↪ List<IList<T>>();
150             Func<T, bool> handler =
        ↪ link =>
151             {
152                 var matchedLink =
        ↪ Memory.GetLinkValue(link);
        ↪ var matchDecision =
        ↪ matchedHandler(matchedLink);
153                 if
        ↪ (Equals(matchDecision,
        ↪ Constants.Break))
154                 return false;
155             }
156         }
157     }
158

```

```

159         // if
160         ↪ (!Equals(matchDecision,
161         ↪ Constants.Skip))
162         //
163         ↪ matchedLinks.Add(matchedLink);
164         ↪ return true;
165         // };
166         // if (!Memory.Each(handler,
167         ↪ restriction))
168         // return
169         ↪ Constants.Break;
170         // }
171         // if
172         ↪ (!matchedLinks.IsNullOrEmpty())
173         // {
174         ↪     var totalMatchedLinks =
175         ↪     matchedLinks.Count;
176         ↪     for (var i = 0; i <
177         ↪     totalMatchedLinks; i++)
178         // {
179         ↪     var matchedLink =
180         ↪     matchedLinks[i];
181         ↪     if
182         ↪     (substitutedHandler != null)
183         // {
184         ↪     var newValue =
185         ↪     new List<T>(); // TODO: Prepare
186         ↪     value to update here
187         ↪     // TODO: Decide
188         ↪     is it actually needed to use
189         ↪     Before and After substitution
190         ↪     handling.
191         // var
192         ↪     substitutedDecision =
193         ↪     substitutedHandler(matchedLink,
194         ↪     newValue);
195         // if
196         ↪     (Equals(substitutedDecision,
197         ↪     Constants.Break))
198         // return
199         ↪     Constants.Break;
200         // if
201         ↪     (Equals(substitutedDecision,
202         ↪     Constants.Continue))
203         // {
204         ↪     // Actual
205         ↪     update here
206         ↪     Memory.SetLinkValue(newValue);
207         ↪     }
208         // if
209         ↪     (Equals(substitutedDecision,
210         ↪     Constants.Skip))
211         // {
212         ↪     // Cancel the
213         ↪     update. TODO: decide use
214         ↪     separate Cancel constant or
215         ↪     Skip is enough?
216         // }
217         // }
218         // }
219         // }
220         // }
221         // }
222         // }
223         // }
224         // }
225         // }
226         // }
227         // }
228         // }
229         // }
230         // }
231         // }
232         // }
233         // }
234         // }
235         // }
236         // }
237         // }
238         // }
239         // }
240         // }
241         // }
242         // }
243         // }
244         // }
245         // }
246         // }
247         // }
248         // }
249         // }
250         // }
251         // }
252         // }
253         // }
254         // }
255         // }
256         // }
257         // }
258         // }
259         // }
260         // }
261         // }
262         // }
263         // }
264         // }
265         // }
266         // }
267         // }
268         // }
269         // }
270         // }
271         // }
272         // }
273         // }
274         // }
275         // }
276         // }
277         // }
278         // }
279         // }
280         // }
281         // }
282         // }
283         // }
284         // }
285         // }
286         // }
287         // }
288         // }
289         // }
290         // }
291         // }
292         // }
293         // }
294         // }
295         // }
296         // }
297         // }
298         // }
299         // }
300         // }
301         // }
302         // }
303         // }
304         // }
305         // }
306         // }
307         // }
308         // }
309         // }
310         // }
311         // }
312         // }
313         // }
314         // }
315         // }
316         // }
317         // }
318         // }
319         // }
320         // }
321         // }
322         // }
323         // }
324         // }
325         // }
326         // }
327         // }
328         // }
329         // }
330         // }
331         // }
332         // }
333         // }
334         // }
335         // }
336         // }
337         // }
338         // }
339         // }
340         // }
341         // }
342         // }
343         // }
344         // }
345         // }
346         // }
347         // }
348         // }
349         // }
350         // }
351         // }
352         // }
353         // }
354         // }
355         // }
356         // }
357         // }
358         // }
359         // }
360         // }
361         // }
362         // }
363         // }
364         // }
365         // }
366         // }
367         // }
368         // }
369         // }
370         // }
371         // }
372         // }
373         // }
374         // }
375         // }
376         // }
377         // }
378         // }
379         // }
380         // }
381         // }
382         // }
383         // }
384         // }
385         // }
386         // }
387         // }
388         // }
389         // }
390         // }
391         // }
392         // }
393         // }
394         // }
395         // }
396         // }
397         // }
398         // }
399         // }
400         // }
401         // }
402         // }
403         // }
404         // }
405         // }
406         // }
407         // }
408         // }
409         // }
410         // }
411         // }
412         // }
413         // }
414         // }
415         // }
416         // }
417         // }
418         // }
419         // }
420         // }
421         // }
422         // }
423         // }
424         // }
425         // }
426         // }
427         // }
428         // }
429         // }
430         // }
431         // }
432         // }
433         // }
434         // }
435         // }
436         // }
437         // }
438         // }
439         // }
440         // }
441         // }
442         // }
443         // }
444         // }
445         // }
446         // }
447         // }
448         // }
449         // }
450         // }
451         // }
452         // }
453         // }
454         // }
455         // }
456         // }
457         // }
458         // }
459         // }
460         // }
461         // }
462         // }
463         // }
464         // }
465         // }
466         // }
467         // }
468         // }
469         // }
470         // }
471         // }
472         // }
473         // }
474         // }
475         // }
476         // }
477         // }
478         // }
479         // }
480         // }
481         // }
482         // }
483         // }
484         // }
485         // }
486         // }
487         // }
488         // }
489         // }
490         // }
491         // }
492         // }
493         // }
494         // }
495         // }
496         // }
497         // }
498         // }
499         // }
500         // }
501         // }
502         // }
503         // }
504         // }
505         // }
506         // }
507         // }
508         // }
509         // }
510         // }
511         // }
512         // }
513         // }
514         // }
515         // }
516         // }
517         // }
518         // }
519         // }
520         // }
521         // }
522         // }
523         // }
524         // }
525         // }
526         // }
527         // }
528         // }
529         // }
530         // }
531         // }
532         // }
533         // }
534         // }
535         // }
536         // }
537         // }
538         // }
539         // }
540         // }
541         // }
542         // }
543         // }
544         // }
545         // }
546         // }
547         // }
548         // }
549         // }
550         // }
551         // }
552         // }
553         // }
554         // }
555         // }
556         // }
557         // }
558         // }
559         // }
560         // }
561         // }
562         // }
563         // }
564         // }
565         // }
566         // }
567         // }
568         // }
569         // }
570         // }
571         // }
572         // }
573         // }
574         // }
575         // }
576         // }
577         // }
578         // }
579         // }
580         // }
581         // }
582         // }
583         // }
584         // }
585         // }
586         // }
587         // }
588         // }
589         // }
590         // }
591         // }
592         // }
593         // }
594         // }
595         // }
596         // }
597         // }
598         // }
599         // }
600         // }
601         // }
602         // }
603         // }
604         // }
605         // }
606         // }
607         // }
608         // }
609         // }
610         // }
611         // }
612         // }
613         // }
614         // }
615         // }
616         // }
617         // }
618         // }
619         // }
620         // }
621         // }
622         // }
623         // }
624         // }
625         // }
626         // }
627         // }
628         // }
629         // }
630         // }
631         // }
632         // }
633         // }
634         // }
635         // }
636         // }
637         // }
638         // }
639         // }
640         // }
641         // }
642         // }
643         // }
644         // }
645         // }
646         // }
647         // }
648         // }
649         // }
650         // }
651         // }
652         // }
653         // }
654         // }
655         // }
656         // }
657         // }
658         // }
659         // }
660         // }
661         // }
662         // }
663         // }
664         // }
665         // }
666         // }
667         // }
668         // }
669         // }
670         // }
671         // }
672         // }
673         // }
674         // }
675         // }
676         // }
677         // }
678         // }
679         // }
680         // }
681         // }
682         // }
683         // }
684         // }
685         // }
686         // }
687         // }
688         // }
689         // }
690         // }
691         // }
692         // }
693         // }
694         // }
695         // }
696         // }
697         // }
698         // }
699         // }
700         // }
701         // }
702         // }
703         // }
704         // }
705         // }
706         // }
707         // }
708         // }
709         // }
710         // }
711         // }
712         // }
713         // }
714         // }
715         // }
716         // }
717         // }
718         // }
719         // }
720         // }
721         // }
722         // }
723         // }
724         // }
725         // }
726         // }
727         // }
728         // }
729         // }
730         // }
731         // }
732         // }
733         // }
734         // }
735         // }
736         // }
737         // }
738         // }
739         // }
740         // }
741         // }
742         // }
743         // }
744         // }
745         // }
746         // }
747         // }
748         // }
749         // }
750         // }
751         // }
752         // }
753         // }
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         // }
782         // }
783         // }
784         // }
785         // }
786         // }
787         // }
788         // }
789         // }
790         // }
791         // }
792         // }
793         // }
794         // }
795         // }
796         // }
797         // }
798         // }
799         // }
800         // }
801         // }
802         // }
803         // }
804         // }
805         // }
806         // }
807         // }
808         // }
809         // }
810         // }
811         // }
812         // }
813         // }
814         // }
815         // }
816         // }
817         // }
818         // }
819         // }
820         // }
821         // }
822         // }
823         // }
824         // }
825         // }
826         // }
827         // }
828         // }
829         // }
830         // }
831         // }
832         // }
833         // }
834         // }
835         // }
836         // }
837         // }
838         // }
839         // }
840         // }
841         // }
842         // }
843         // }
844         // }
845         // }
846         // }
847         // }
848         // }
849         // }
850         // }
851         // }
852         // }
853         // }
854         // }
855         // }
856         // }
857         // }
858         // }
859         // }
860         // }
861         // }
862         // }
863         // }
864         // }
865         // }
866         // }
867         // }
868         // }
869         // }
870         // }
871         // }
872         // }
873         // }
874         // }
875         // }
876         // }
877         // }
878         // }
879         // }
880         // }
881         // }
882         // }
883         // }
884         // }
885         // }
886         // }
887         // }
888         // }
889         // }
890         // }
891         // }
892         // }
893         // }
894         // }
895         // }
896         // }
897         // }
898         // }
899         // }
900         // }
901         // }
902         // }
903         // }
904         // }
905         // }
906         // }
907         // }
908         // }
909         // }
910         // }
911         // }
912         // }
913         // }
914         // }
915         // }
916         // }
917         // }
918         // }
919         // }
920         // }
921         // }
922         // }
923         // }
924         // }
925         // }
926         // }
927         // }
928         // }
929         // }
930         // }
931         // }
932         // }
933         // }
934         // }
935         // }
936         // }
937         // }
938         // }
939         // }
940         // }
941         // }
942         // }
943         // }
944         // }
945         // }
946         // }
947         // }
948         // }
949         // }
950         // }
951         // }
952         // }
953         // }
954         // }
955         // }
956         // }
957         // }
958         // }
959         // }
960         // }
961         // }
962         // }
963         // }
964         // }
965         // }
966         // }
967         // }
968         // }
969         // }
970         // }
971         // }
972         // }
973         // }
974         // }
975         // }
976         // }
977         // }
978         // }
979         // }
980         // }
981         // }
982         // }
983         // }
984         // }
985         // }
986         // }
987         // }
988         // }
989         // }
990         // }
991         // }
992         // }
993         // }
994         // }
995         // }
996         // }
997         // }
998         // }
999         // }
1000        // }

```

```

253         return substitutionHandler(
254             ↪ ler(before,
255             ↪ after);
256     }
257     return Constants.Continue;
258 }
259 else
260 {
261     throw new
262         ↪ NotSupportedException();
263 }
264 }
265 }
266 }
267 }
268 }
269 }
270 }
271 }
272 }
273 }
274 }
275 }
276 }
277 }
278 }
279 }
280 }
281 }
282 }
283 }
284 }
285 }
286 }
287 }
288 }
289 }
290 }
291 }
292 }
293 }
294 }
295 }
296 }
297 }
298 }
299 }
300 }
301 }
302 }
303 }
304 }
305 }
306 }
307 }
308 }
309 }
310 }

```

```

307 /// <remarks>
308 /// IList<IList<IList<T>>>
309 /// | | | | |
310 /// | | | | |

```

```

311 /// | | | | |
312 /// | | | | |
313 /// | | | | |
314 /// | | | | |
315 /// | | | | |
316 /// | | | | |
317 /// | | | | |
318 /// | | | | |
319 /// | | | | |
320 /// | | | | |
321 /// | | | | |
322 /// | | | | |
323 /// | | | | |
324 /// | | | | |
325 /// | | | | |
326 /// | | | | |
327 /// | | | | |
328 /// | | | | |
329 /// | | | | |
330 /// | | | | |
331 /// | | | | |

```

./DoubletComparer.cs

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3
4 namespace Platform.Data.Doublets
5 {
6     /// <remarks>
7     /// TODO: Может стоит попробовать ref во
8     ↪ всех методах (IRefEqualityComparer)
9     /// 2x faster with comparer
10    /// </remarks>
11    public class DoubletComparer<T> :
12        ↪ IEqualityComparer<Doublet<T>>
13    {
14        private static readonly
15            ↪ EqualityComparer<T>
16            ↪ _equalityComparer =
17            ↪ EqualityComparer<T>.Default;
18
19        public static readonly
20            ↪ DoubletComparer<T> Default = new
21            ↪ DoubletComparer<T>();
22
23        [MethodImpl(MethodImplOptions.AggressiveInlining)]
24        ↪ public bool Equals(Doublet<T> x,
25            ↪ Doublet<T> y) =>
26            ↪ _equalityComparer.Equals(x.Source,
27            ↪ y.Source) &&
28            ↪ _equalityComparer.Equals(x.Target,
29            ↪ y.Target);
30
31        [MethodImpl(MethodImplOptions.AggressiveInlining)]
32        ↪ public int GetHashCode(Doublet<T> obj)
33            ↪ =>
34            ↪ unchecked(obj.Source.GetHashCode()
35            ↪ << 15 ^ obj.Target.GetHashCode());
36    }
37 }

```

./Doublet.cs

```

1 using System;
2 using System.Collections.Generic;
3
4 namespace Platform.Data.Doublets
5 {
6     public struct Doublet<T> :
7         ↪ IEquatable<Doublet<T>>
8     {
9         private static readonly
10             ↪ EqualityComparer<T>
11             ↪ _equalityComparer =
12             ↪ EqualityComparer<T>.Default;
13
14         public T Source { get; set; }
15         public T Target { get; set; }
16     }

```

```

12 public Doublet(T source, T target)
13 {
14     Source = source;
15     Target = target;
16 }
17
18 public override string ToString() =>
19     $"{{Source}}->{{Target}}";
20
21 public bool Equals(Doublet<T> other) =>
22     _equalityComparer.Equals(Source,
23     other.Source) &&
24     _equalityComparer.Equals(Target,
25     other.Target);
26
27 }
28
29 }

```

./Hybrid.cs

```

1 using System;
2 using System.Reflection;
3 using Platform.Reflection;
4 using Platform.Converters;
5 using Platform.Numbers;
6
7 namespace Platform.Data.Doublets
8 {
9     public class Hybrid<T>
10     {
11         public readonly T Value;
12         public bool IsNothing =>
13             Convert.ToInt64(To.Signed(Value))
14             == 0;
15         public bool IsInternal =>
16             Convert.ToInt64(To.Signed(Value)) >
17             0;
18         public bool IsExternal =>
19             Convert.ToInt64(To.Signed(Value)) <
20             0;
21         public long AbsoluteValue => Math.Abs(C
22             onvert.ToInt64(To.Signed(Value)));
23
24         public Hybrid(T value)
25         {
26             if (CachedTypeInfo<T>.IsSigned)
27             {
28                 throw new
29                     ↳ NotSupportedException();
30             }
31             Value = value;
32         }
33
34         public Hybrid(object value) => Value =
35             To.UnsignedAs<T>(Convert.ChangeType
36             (value,
37             CachedTypeInfo<T>.SignedVersion));
38
39         public Hybrid(object value, bool
40             ↳ isExternal)
41         {
42             var signedType =
43                 ↳ CachedTypeInfo<T>.SignedVersion;
44             var signedValue =
45                 ↳ Convert.ChangeType(value,
46                 signedType);
47             var abs = typeof(MathHelpers).GetTy
48                 peInfo().GetMethod("Abs").MakeG
49                 enericMethod(signedType);
50             var negate = typeof(MathHelpers).Ge
51                 tTypeInfo().GetMethod("Negate")
52                 ↳ .MakeGenericMethod(signedType);
53             var absoluteValue =
54                 ↳ abs.Invoke(null, new[] {
55                 signedValue });
56             var resultValue = isExternal ?
57                 ↳ negate.Invoke(null, new[] {
58                 absoluteValue }) :
59                 ↳ absoluteValue;
60             Value =
61                 ↳ To.UnsignedAs<T>(resultValue);
62         }
63
64         public static implicit operator
65             ↳ Hybrid<T>(T integer) => new
66             ↳ Hybrid<T>(integer);
67
68     }
69
70 }

```

```

41 public static explicit operator
42     ↳ Hybrid<T>(ulong integer) => new
43     ↳ Hybrid<T>(integer);
44
45 public static explicit operator
46     ↳ Hybrid<T>(long integer) => new
47     ↳ Hybrid<T>(integer);
48
49 public static explicit operator
50     ↳ Hybrid<T>(uint integer) => new
51     ↳ Hybrid<T>(integer);
52
53 public static explicit operator
54     ↳ Hybrid<T>(int integer) => new
55     ↳ Hybrid<T>(integer);
56
57 public static explicit operator
58     ↳ Hybrid<T>(ushort integer) => new
59     ↳ Hybrid<T>(integer);
60
61 public static explicit operator
62     ↳ Hybrid<T>(short integer) => new
63     ↳ Hybrid<T>(integer);
64
65 public static explicit operator
66     ↳ Hybrid<T>(byte integer) => new
67     ↳ Hybrid<T>(integer);
68
69 public static explicit operator
70     ↳ Hybrid<T>(sbyte integer) => new
71     ↳ Hybrid<T>(integer);
72
73 public static implicit operator
74     ↳ T(Hybrid<T> hybrid) => hybrid.Value;
75
76 public static explicit operator
77     ↳ ulong(Hybrid<T> hybrid) =>
78     ↳ Convert.ToUInt64(hybrid.Value);
79
80 public static explicit operator
81     ↳ long(Hybrid<T> hybrid) =>
82     ↳ hybrid.AbsoluteValue;
83
84 public static explicit operator
85     ↳ uint(Hybrid<T> hybrid) =>
86     ↳ Convert.ToUInt32(hybrid.Value);
87
88 public static explicit operator
89     ↳ int(Hybrid<T> hybrid) => Convert.To
90     ↳ Int32(hybrid.AbsoluteValue);
91
92 public static explicit operator
93     ↳ ushort(Hybrid<T> hybrid) =>
94     ↳ Convert.ToUInt16(hybrid.Value);
95
96 public static explicit operator
97     ↳ short(Hybrid<T> hybrid) => Convert.
98     ↳ ToInt16(hybrid.AbsoluteValue);
99
100 public static explicit operator
101     ↳ byte(Hybrid<T> hybrid) =>
102     ↳ Convert.ToByte(hybrid.Value);
103
104 public static explicit operator
105     ↳ sbyte(Hybrid<T> hybrid) => Convert.
106     ↳ ToSByte(hybrid.AbsoluteValue);
107
108 public override string ToString() =>
109     IsNothing ? default(T) == null ?
110     ↳ "Nothing" : default(T).ToString() :
111     ↳ IsExternal ? $"<{AbsoluteValue}>" :
112     ↳ Value.ToString();
113
114 }
115
116 }

```

./ILinks.cs

```

1 using Platform.Data.Constants;
2
3 namespace Platform.Data.Doublets
4 {
5     public interface ILinks<TLink> :
6         ↳ ILinks<TLink,
7         ↳ LinksCombinedConstants<TLink, TLink,
8         ↳ int>>
9     {
10
11     }
12
13 }

```

```

7     }
8 }

./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.Numbers;
9 using Platform.Random;
10 using Platform.Helpers.Setters;
11 using Platform.Data.Exceptions;
12
13 namespace Platform.Data.Doublets
14 {
15     public static class ILinksExtensions
16     {
17         public static void
18             ↪ RunRandomCreations<TLink>(this
19             ↪ ILinks<TLink> links, long
20             ↪ amountOfCreations)
21         {
22             for (long i = 0; i <
23                 ↪ amountOfCreations; i++)
24             {
25                 var linksAddressRange = new
26                     ↪ Range<ulong>(0, (Integer<TL
27                     ↪ ink>))links.Count());
28                 Integer<TLink> source =
29                     ↪ RandomHelpers.Default.NextU
30                     ↪ Int64(linksAddressRange);
31                 Integer<TLink> target =
32                     ↪ RandomHelpers.Default.NextU
33                     ↪ Int64(linksAddressRange);
34                 links.CreateAndUpdate(source,
35                     ↪ target);
36             }
37         }
38
39         public static void
40             ↪ RunRandomSearches<TLink>(this
41             ↪ ILinks<TLink> links, long
42             ↪ amountOfSearches)
43         {
44             for (long i = 0; i <
45                 ↪ amountOfSearches; i++)
46             {
47                 var linkAddressRange = new
48                     ↪ Range<ulong>(1, (Integer<TL
49                     ↪ ink>))links.Count());
50                 Integer<TLink> source =
51                     ↪ RandomHelpers.Default.NextU
52                     ↪ Int64(linkAddressRange);
53                 Integer<TLink> target =
54                     ↪ RandomHelpers.Default.NextU
55                     ↪ Int64(linkAddressRange);
56                 links.SearchOrDefault(source,
57                     ↪ target);
58             }
59         }
60
61         public static void
62             ↪ RunRandomDeletions<TLink>(this
63             ↪ ILinks<TLink> links, long
64             ↪ amountOfDeletions)
65         {
66             var min = (ulong)amountOfDeletions
67                 ↪ > (Integer<TLink>)links.Count()
68                 ↪ ? 1 :
69                 ↪ (Integer<TLink>)links.Count() -
70                 ↪ (ulong)amountOfDeletions;
71             for (long i = 0; i <
72                 ↪ amountOfDeletions; i++)
73             {
74                 var linksAddressRange = new
75                     ↪ Range<ulong>(min, (Integer<
76                     ↪ TLink>))links.Count());
77                 Integer<TLink> link =
78                     ↪ RandomHelpers.Default.NextU
79                     ↪ Int64(linksAddressRange);
80                 links.Delete(link);
81             }
82         }
83     }
84 }
85
86 if ((Integer<TLink>)links.Count
87     ↪ () <
88     ↪ min)
89 {
90     break;
91 }
92
93 }
94
95 /// <remarks>
96 /// TODO: Возможно есть очень простой
97     ↪ способ это сделать.
98 /// (Например просто удалить файл, или
99     ↪ изменить его размер таким образом,
100    /// чтобы удалился весь контент)
101    /// Например через
102    ↪ _header->AllocatedLinks в
103    ↪ ResizableDirectMemoryLinks
104    /// </remarks>
105    public static void
106        ↪ DeleteAll<TLink>(this ILinks<TLink>
107        ↪ links)
108    {
109        var equalityComparer =
110            ↪ EqualityComparer<TLink>.Default;
111        var comparer =
112            ↪ Comparer<TLink>.Default;
113        for (var i = links.Count();
114            ↪ comparer.Compare(i, default) >
115            ↪ 0; i =
116            ↪ ArithmeticHelpers.Decrement(i))
117        {
118            links.Delete(i);
119            if (!equalityComparer.Equals(li
120                ↪ nks.Count(),
121                ↪ ArithmeticHelpers.Decrement
122                ↪ (i)))
123            {
124                i = links.Count();
125            }
126        }
127    }
128
129     public static TLink First<TLink>(this
130         ↪ ILinks<TLink> links)
131     {
132         TLink firstLink = default;
133         var equalityComparer =
134             ↪ EqualityComparer<TLink>.Default;
135         if (equalityComparer.Equals(links.C
136             ↪ ount(),
137             ↪ default))
138         {
139             throw new Exception("В
140                 ↪ хранилище нет связей.");
141         }
142         links.Each(links.Constants.Any,
143             ↪ links.Constants.Any, link =>
144         {
145             firstLink = link[links.Constant
146                 ↪ s.IndexPart];
147             return links.Constants.Break;
148         });
149         if (equalityComparer.Equals(firstLi
150             ↪ nk,
151             ↪ default))
152         {
153             throw new Exception("В процессе
154                 ↪ поиска по хранилищу не было
155                 ↪ найдено связей.");
156         }
157         return firstLink;
158     }
159
160     public static bool
161         ↪ IsInnerReference<TLink>(this
162         ↪ ILinks<TLink> links, TLink
163         ↪ reference)
164     {
165         var constants = links.Constants;
166         var comparer =
167             ↪ Comparer<TLink>.Default;

```

```

98         return comparer.Compare(constants.M
99         ↪ inPossibleIndex, reference) >=
100         ↪ 0 &&
101         ↪ comparer.Compare(reference,
102         ↪ constants.MaxPossibleIndex) <=
103         ↪ 0;
104     }
105
106     #region Paths
107
108     /// <remarks>
109     /// TODO: Как так? Как то что ниже
110     ↪ может быть корректно?
111     /// Скорее всего практически не
112     ↪ применимо
113     /// Предполагалось, что можно было
114     ↪ конвертировать формируемый в
115     ↪ проходе через SequenceWalker
116     /// Stack в конкретный путь из Source,
117     ↪ Target до связи, но это не всегда
118     ↪ так.
119     /// TODO: Возможно нужен метод, который
120     ↪ именно выбрасывает исключения
121     ↪ (EnsurePathExists)
122     /// </remarks>
123     public static bool
124     ↪ CheckPathExistence<TLink>(this
125     ↪ ILinks<TLink> links, params TLink[]
126     ↪ path)
127     {
128         var current = path[0];
129         //EnsureLinkExists(current, "path");
130         if (!links.Exists(current))
131         {
132             return false;
133         }
134         var equalityComparer =
135         ↪ EqualityComparer<TLink>.Default;
136         var constants = links.Constants;
137         for (var i = 1; i < path.Length;
138         ↪ i++)
139         {
140             var next = path[i];
141             var values =
142             ↪ links.GetLink(current);
143             var source = values[constants.S
144             ↪ ourcePart];
145             var target = values[constants.T
146             ↪ argetPart];
147             if (equalityComparer.Equals(sou
148             ↪ rce, target) &&
149             ↪ equalityComparer.Equals(sou
150             ↪ rce,
151             ↪ next))
152             {
153                 //throw new Exception(strin
154                 ↪ g.Format("Невозможно
155                 ↪ выбрать путь, так как и
156                 ↪ Source и Target
157                 ↪ совпадают с элементом
158                 ↪ пути {0}.", next));
159                 return false;
160             }
161             if (!equalityComparer.Equals(ne
162             ↪ xt, source) &&
163             ↪ !equalityComparer.Equals(ne
164             ↪ xt,
165             ↪ target))
166             {
167                 //throw new Exception(strin
168                 ↪ g.Format("Невозможно
169                 ↪ продолжить путь через
170                 ↪ элемент пути {0}",
171                 ↪ next));
172                 return false;
173             }
174             current = next;
175         }
176         return true;
177     }
178
179     /// <remarks>
180     /// Может потребовать дополнительного
181     ↪ стека для PathElement's при
182     ↪ использовании SequenceWalker.
183     /// </remarks>
184
185     public static TLink
186     ↪ GetByKeys<TLink>(this ILinks<TLink>
187     ↪ links, TLink root, params int[]
188     ↪ path)
189     {
190         links.EnsureLinkExists(root,
191         ↪ "root");
192         var currentLink = root;
193         for (var i = 0; i < path.Length;
194         ↪ i++)
195         {
196             currentLink = links.GetLink(cur
197             ↪ rentLink)[path[i]];
198         }
199         return currentLink;
200     }
201
202     public static TLink GetSquareMatrixSequ
203     ↪ enceElementByIndex<TLink>(this
204     ↪ ILinks<TLink> links, TLink root,
205     ↪ ulong size, ulong index)
206     {
207         var constants = links.Constants;
208         var source = constants.SourcePart;
209         var target = constants.TargetPart;
210         if (!MathHelpers.IsPowerOfTwo(size))
211         {
212             throw new ArgumentOutOfRangeException
213             ↪ (nameof(size),
214             ↪ "Sequences with sizes other
215             ↪ than powers of two are not
216             ↪ supported.");
217         }
218         var path = new BitArray(BitConverte
219         ↪ r.GetBytes(index));
220         var length = BitwiseHelpers.GetLowe
221         ↪ stBitPosition(size);
222         links.EnsureLinkExists(root,
223         ↪ "root");
224         var currentLink = root;
225         for (var i = length - 1; i >= 0;
226         ↪ i--)
227         {
228             currentLink = links.GetLink(cur
229             ↪ rentLink)[path[i] ? target
230             ↪ : source];
231         }
232         return currentLink;
233     }
234
235     #endregion
236
237     /// <summary>
238     /// Возвращает индекс указанной связи.
239     /// </summary>
240     /// <param name="links">Хранилище
241     ↪ связей.</param>
242     /// <param name="link">Связь
243     ↪ представленная списком, состоящим
244     ↪ из её адреса и содержимого.</param>
245     /// <returns>Индекс начальной связи для
246     ↪ указанной связи.</returns>
247     [MethodImpl(MethodImplOptions.Aggressive
248     ↪ eInlining)]
249     public static TLink
250     ↪ GetIndex<TLink>(this ILinks<TLink>
251     ↪ links, IList<TLink> link) =>
252     ↪ link[links.Constants.IndexPart];
253
254     /// <summary>
255     /// Возвращает индекс начальной
256     ↪ (Source) связи для указанной связи.
257     /// </summary>
258     /// <param name="links">Хранилище
259     ↪ связей.</param>
260     /// <param name="link">Индекс
261     ↪ связи.</param>
262     /// <returns>Индекс начальной связи для
263     ↪ указанной связи.</returns>
264     [MethodImpl(MethodImplOptions.Aggressive
265     ↪ eInlining)]
266     public static TLink
267     ↪ GetSource<TLink>(this ILinks<TLink>
268     ↪ links, TLink link) => links.GetLink
269     ↪ (link)[links.Constants.SourcePart];

```

```

194     /// <summary>
195     /// Возвращает индекс начальной
196     /// (Source) связи для указанной связи.
197     /// </summary>
198     /// <param name="links">Хранилище
199     /// связей.</param>
200     /// <param name="link">Связь
201     /// представленная списком, состоящим
202     /// из её адреса и содержимого.</param>
203     /// <returns>Индекс начальной связи для
204     /// указанной связи.</returns>
205     [MethodImpl(MethodImplOptions.AggressiveInlining)]
206     public static TLink
207     GetSource<TLink>(this ILinks<TLink>
208     links, TLink link) =>
209     link[links.Constants.SourcePart];
210
211     /// <summary>
212     /// Возвращает индекс конечной (Target)
213     /// связи для указанной связи.
214     /// </summary>
215     /// <param name="links">Хранилище
216     /// связей.</param>
217     /// <param name="link">Индекс
218     /// связи.</param>
219     /// <returns>Индекс конечной связи для
220     /// указанной связи.</returns>
221     [MethodImpl(MethodImplOptions.AggressiveInlining)]
222     public static TLink
223     GetTarget<TLink>(this ILinks<TLink>
224     links, TLink link) => links.GetLink
225     (link)[links.Constants.TargetPart];
226
227     /// <summary>
228     /// Возвращает индекс конечной (Target)
229     /// связи для указанной связи.
230     /// </summary>
231     /// <param name="links">Хранилище
232     /// связей.</param>
233     /// <param name="link">Связь
234     /// представленная списком, состоящим
235     /// из её адреса и содержимого.</param>
236     /// <returns>Индекс конечной связи для
237     /// указанной связи.</returns>
238     [MethodImpl(MethodImplOptions.AggressiveInlining)]
239     public static TLink
240     GetTarget<TLink>(this ILinks<TLink>
241     links, TLink link) => links.GetLink
242     (link)[links.Constants.TargetPart];
243
244     /// <summary>
245     /// Выполняет проход по всем связям,
246     /// соответствующим шаблону, вызывая
247     /// обработчик (handler) для каждой
248     /// подходящей связи.
249     /// </summary>
250     /// <param name="links">Хранилище
251     /// связей.</param>
252     /// <param name="source">Значение,
253     /// определяющее соответствующие
254     /// шаблону связи. (Constants.Null -
255     /// 0-я связь, обозначающая ссылку на
256     /// пустоту в качестве начала,
257     /// Constants.Any - любое начало, 1..∞
258     /// конкретное начало)</param>
259     /// <param name="target">Значение,
260     /// определяющее соответствующие
261     /// шаблону связи. (Constants.Null -
262     /// 0-я связь, обозначающая ссылку на
263     /// пустоту в качестве конца,
264     /// Constants.Any - любой конец, 1..∞
265     /// конкретный конец)</param>
266     /// <param name="handler">Обработчик
267     /// каждой подходящей связи.</param>
268     /// <returns>True, в случае если проход
269     /// по связям не был прерван и False в
270     /// обратном случае.</returns>
271     [MethodImpl(MethodImplOptions.AggressiveInlining)]
272     public static bool Each<TLink>(this
273     ILinks<TLink> links, TLink source,
274     TLink target, Func<TLink, bool>
275     handler)
276     {
277         var constants = links.Constants;
278         return links.Each(link => handler(l
279         ink[constants.IndexPart]) ?
280         constants.Continue :
281         constants.Break, constants.Any,
282         source, target);
283     }
284
285     /// <summary>
286     /// Выполняет проход по всем связям,
287     /// соответствующим шаблону, вызывая
288     /// обработчик (handler) для каждой
289     /// подходящей связи.
290     /// </summary>
291     /// <param name="links">Хранилище
292     /// связей.</param>
293     /// <param name="source">Значение,
294     /// определяющее соответствующие
295     /// шаблону связи. (Constants.Null -
296     /// 0-я связь, обозначающая ссылку на
297     /// пустоту в качестве начала,
298     /// Constants.Any - любое начало, 1..∞
299     /// конкретное начало)</param>
300     /// <param name="target">Значение,
301     /// определяющее соответствующие
302     /// шаблону связи. (Constants.Null -
303     /// 0-я связь, обозначающая ссылку на
304     /// пустоту в качестве конца,
305     /// Constants.Any - любой конец, 1..∞
306     /// конкретный конец)</param>
307     /// <param name="handler">Обработчик
308     /// каждой подходящей связи.</param>
309     /// <returns>True, в случае если проход
310     /// по связям не был прерван и False в
311     /// обратном случае.</returns>
312     [MethodImpl(MethodImplOptions.AggressiveInlining)]
313     public static bool Each<TLink>(this
314     ILinks<TLink> links, TLink source,
315     TLink target, Func<TLink, bool>
316     handler)
317     {
318         var constants = links.Constants;
319         return links.Each(handler,
320         constants.Any, source, target);
321     }
322
323     [MethodImpl(MethodImplOptions.AggressiveInlining)]
324     public static bool EqualityComparer<TLink>.Default.
325     Equals(links.Each(handler,
326     restrictions),
327     links.Constants.Continue);

```



```

264 public static IList<IList<TLink>>
    ↳ All<TLink>(this ILinks<TLink>
    ↳ links, params TLink[] restrictions)
265 {
266     var constants = links.Constants;
267     int listSize = (Integer<TLink>)link
    ↳ s.Count(restrictions);
268     var list = new
    ↳ IList<TLink>[listSize];
269     if (listSize > 0)
270     {
271         var filler = new
    ↳ ArrayFiller<IList<TLink>,
    ↳ TLink>(list,
    ↳ links.Constants.Continue);
272         links.Each(filler.AddAndReturnC
    ↳ onstant,
    ↳ restrictions);
273     }
274     return list;
275 }
276
277 /// <summary>
278 /// Возвращает значение, определяющее
    ↳ существует ли связь с указанными
    ↳ началом и концом в хранилище связей.
279 /// </summary>
280 /// <param name="links">Хранилище
    ↳ связей.</param>
281 /// <param name="source">Начало
    ↳ связи.</param>
282 /// <param name="target">Конец
    ↳ связи.</param>
283 /// <returns>Значение, определяющее
    ↳ существует ли связь.</returns>
284 [MethodImpl(MethodImplOptions.Aggressive
    ↳ eInlining)]
285 public static bool Exists<TLink>(this
    ↳ ILinks<TLink> links, TLink source,
    ↳ TLink target) =>
    ↳ Comparer<TLink>.Default.Compare(lin
    ↳ ks.Count(links.Constants.Any,
    ↳ source, target), default) > 0;
286
287 #region Ensure
288 // TODO: May be move to
    ↳ EnsureExtensions or make it both
    ↳ there and here
289
290 [MethodImpl(MethodImplOptions.Aggressive
    ↳ eInlining)]
291 public static void EnsureInnerReference
    ↳ Exists<TLink>(this ILinks<TLink>
    ↳ links, TLink reference, string
    ↳ argumentName)
292 {
293     if (links.IsInnerReference(referenc
    ↳ e) &&
    ↳ !links.Exists(reference))
294     {
295         throw new
    ↳ ArgumentLinkDoesNotExistsEx
    ↳ ception<TLink>(reference,
    ↳ argumentName);
296     }
297 }
298
299 [MethodImpl(MethodImplOptions.Aggressive
    ↳ eInlining)]
300 public static void EnsureInnerReference
    ↳ Exists<TLink>(this ILinks<TLink>
    ↳ links, IList<TLink> restrictions,
    ↳ string argumentName)
301 {
302     for (int i = 0; i <
    ↳ restrictions.Count; i++)
303     {
304         links.EnsureInnerReferenceExist
    ↳ s(restrictions[i],
    ↳ argumentName);
305     }
306 }
307
308 [MethodImpl(MethodImplOptions.Aggressive
    ↳ eInlining)]
309 public static void
    ↳ EnsureLinkIsAnyOrExists<TLink>(this
    ↳ ILinks<TLink> links, IList<TLink>
    ↳ restrictions)
310 {
311     for (int i = 0; i <
    ↳ restrictions.Count; i++)
312     {
313         links.EnsureLinkIsAnyOrExists(r
    ↳ estrictions[i],
    ↳ nameof(restrictions));
314     }
315 }
316
317 [MethodImpl(MethodImplOptions.Aggressive
    ↳ eInlining)]
318 public static void
    ↳ EnsureLinkIsAnyOrExists<TLink>(this
    ↳ ILinks<TLink> links, TLink link,
    ↳ string argumentName)
319 {
320     var equalityComparer =
    ↳ EqualityComparer<TLink>.Default;
321     if (!equalityComparer.Equals(link,
    ↳ links.Constants.Any) &&
    ↳ !links.Exists(link))
322     {
323         throw new ArgumentLinkDoesNotEx
    ↳ istsException<TLink>(link,
    ↳ argumentName);
324     }
325 }
326
327 [MethodImpl(MethodImplOptions.Aggressive
    ↳ eInlining)]
328 public static void EnsureLinkIsItselfOr
    ↳ Exists<TLink>(this ILinks<TLink>
    ↳ links, TLink link, string
    ↳ argumentName)
329 {
330     var equalityComparer =
    ↳ EqualityComparer<TLink>.Default;
331     if (!equalityComparer.Equals(link,
    ↳ links.Constants.Itself) &&
    ↳ !links.Exists(link))
332     {
333         throw new ArgumentLinkDoesNotEx
    ↳ istsException<TLink>(link,
    ↳ argumentName);
334     }
335 }
336
337 /// <param name="links">Хранилище
    ↳ связей.</param>
338 [MethodImpl(MethodImplOptions.Aggressive
    ↳ eInlining)]
339 public static void
    ↳ EnsureDoesNotExist<TLink>(this
    ↳ ILinks<TLink> links, TLink source,
    ↳ TLink target)
340 {
341     if (links.Exists(source, target))
342     {
343         throw new LinkWithSameValueAlre
    ↳ adyExistsException();
344     }
345 }
346
347 /// <param name="links">Хранилище
    ↳ связей.</param>
348 public static void
    ↳ EnsureNoDependencies<TLink>(this
    ↳ ILinks<TLink> links, TLink link)
349 {
350     if (links.DependenciesExist(link))
351     {
352         throw new
    ↳ ArgumentLinkHasDependencies
    ↳ Exception<TLink>(link);
353     }
354 }
355
356 /// <param name="links">Хранилище
    ↳ связей.</param>

```

```

357 public static void
    ↳ EnsureCreated<TLink>(this
    ↳ ILinks<TLink> links, params TLink[]
    ↳ addresses) =>
    ↳ links.EnsureCreated(links.Create,
    ↳ addresses);
358
359 /// <param name="links">Хранилище
    ↳ связей.</param>
360 public static void
    ↳ EnsurePointsCreated<TLink>(this
    ↳ ILinks<TLink> links, params TLink[]
    ↳ addresses) => links.EnsureCreated(l
    ↳ inks.CreatePoint,
    ↳ addresses);
361
362 /// <param name="links">Хранилище
    ↳ связей.</param>
363 public static void
    ↳ EnsureCreated<TLink>(this
    ↳ ILinks<TLink> links, Func<TLink>
    ↳ creator, params TLink[] addresses)
364 {
365     var constants = links.Constants;
366     var nonExistentAddresses = new Hash
    ↳
    ↳ Set<ulong>(addresses.Where(x =>
    ↳ !links.Exists(x)).Select(x =>
    ↳ (ulong)(Integer<TLink>)x));
367     if (nonExistentAddresses.Count > 0)
368     {
369         var max =
    ↳
    ↳ nonExistentAddresses.Max();
370         // TODO: Эту верхнюю границу
    ↳
    ↳ нужно разрешить
    ↳
    ↳ переопределять (проверить
    ↳
    ↳ применяется ли эта логика)
371         max = Math.Min(max,
    ↳
    ↳ (Integer<TLink>)constants.M
    ↳
    ↳ axPossibleIndex);
372         var createdLinks = new
    ↳
    ↳ List<TLink>();
373         var equalityComparer = Equality
    ↳
    ↳ Comparer<TLink>.Default;
374         TLink createdLink = creator();
375         while (!equalityComparer.Equals
    ↳
    ↳ (createdLink,
    ↳
    ↳ (Integer<TLink>)max))
376         {
377             createdLinks.Add(createdLin
    ↳
    ↳ k);
378         }
379         for (var i = 0; i <
    ↳
    ↳ createdLinks.Count; i++)
380         {
381             if (!nonExistentAddresses.C
    ↳
    ↳ ontains((Integer<TLink>
    ↳
    ↳ )createdLinks[i]))
382             {
383                 links.Delete(createdLin
    ↳
    ↳ ks[i]);
384             }
385         }
386     }
387 }
388 #endregion
389
390 /// <param name="links">Хранилище
    ↳ связей.</param>
391 public static ulong
    ↳
    ↳ DependenciesCount<TLink>(this
    ↳ ILinks<TLink> links, TLink link)
392 {
393     var constants = links.Constants;
394     var values = links.GetLink(link);
395     ulong referencesAsSource =
    ↳
    ↳ (Integer<TLink>)links.Count(con
    ↳
    ↳ stants.Any, link,
    ↳
    ↳ constants.Any);
396     var equalityComparer =
    ↳
    ↳ EqualityComparer<TLink>.Default;
397     if (equalityComparer.Equals(values
    ↳
    ↳ [constants.SourcePart],
    ↳
    ↳ link))
398     {
399
400         referencesAsSource--;
401     }
402     ulong referencesAsTarget =
    ↳
    ↳ (Integer<TLink>)links.Count(con
    ↳
    ↳ stants.Any, constants.Any,
    ↳
    ↳ link);
403     if (equalityComparer.Equals(values
    ↳
    ↳ [constants.TargetPart],
    ↳
    ↳ link))
404     {
405         referencesAsTarget--;
406     }
407     return referencesAsSource +
    ↳
    ↳ referencesAsTarget;
408 }
409
410 /// <param name="links">Хранилище
    ↳ связей.</param>
411 [MethodImpl(MethodImplOptions.Aggressive
    ↳
    ↳ eInlining)]
412 public static bool
    ↳
    ↳ DependenciesExist<TLink>(this
    ↳
    ↳ ILinks<TLink> links, TLink link) =>
    ↳
    ↳ links.DependenciesCount(link) > 0;
413
414 /// <param name="links">Хранилище
    ↳ связей.</param>
415 [MethodImpl(MethodImplOptions.Aggressive
    ↳
    ↳ eInlining)]
416 public static bool Equals<TLink>(this
    ↳
    ↳ ILinks<TLink> links, TLink link,
    ↳
    ↳ TLink source, TLink target)
417 {
418     var constants = links.Constants;
419     var values = links.GetLink(link);
420     var equalityComparer =
    ↳
    ↳ EqualityComparer<TLink>.Default;
421     return equalityComparer.Equals(valu
    ↳
    ↳ es[constants.SourcePart],
    ↳
    ↳ source) &&
    ↳
    ↳ equalityComparer.Equals(values
    ↳
    ↳ [constants.TargetPart],
    ↳
    ↳ target);
422 }
423
424 /// <summary>
425 /// Выполняет поиск связи с указанными
    ↳
    ↳ Source (началом) и Target (концом).
426 /// </summary>
427 /// <param name="links">Хранилище
    ↳ связей.</param>
428 /// <param name="source">Индекс связи,
    ↳
    ↳ которая является началом для
    ↳
    ↳ искомой связи.</param>
429 /// <param name="target">Индекс связи,
    ↳
    ↳ которая является концом для искомой
    ↳
    ↳ связи.</param>
430 /// <returns>Индекс искомой связи с
    ↳
    ↳ указанными Source (началом) и
    ↳
    ↳ Target (концом).</returns>
431 [MethodImpl(MethodImplOptions.Aggressive
    ↳
    ↳ eInlining)]
432 public static TLink
    ↳
    ↳ SearchOrDefault<TLink>(this
    ↳
    ↳ ILinks<TLink> links, TLink source,
    ↳
    ↳ TLink target)
433 {
434     var contants = links.Constants;
435     var setter = new Setter<TLink,
    ↳
    ↳ TLink>(contants.Continue,
    ↳
    ↳ contants.Break, default);
436     links.Each(setter.SetFirstAndReturn
    ↳
    ↳ False, contants.Any, source,
    ↳
    ↳ target);
437     return setter.Result;
438 }
439
440 /// <param name="links">Хранилище
    ↳ связей.</param>
441 [MethodImpl(MethodImplOptions.Aggressive
    ↳
    ↳ eInlining)]
442 public static TLink
    ↳
    ↳ CreatePoint<TLink>(this
    ↳
    ↳ ILinks<TLink> links)

```

```

443     {
444         var link = links.Create();
445         return links.Update(link, link,
            ↳ link);
446     }
447
448     /// <param name="links">Хранилище
449     ↳ связей.</param>
450     [MethodImpl(MethodImplOptions.Aggressive
    ↳ eInlining)]
451     public static TLink
452     CreateAndUpdate<TLink>(this
453     ↳ ILinks<TLink> links, TLink source,
454     ↳ TLink target) =>
455     links.Update(links.Create(),
456     ↳ source, target);
457
458     /// <summary>
459     /// Обновляет связь с указанными
460     ↳ началом (Source) и концом (Target)
461     /// на связь с указанными началом
462     ↳ (NewSource) и концом (NewTarget).
463     /// </summary>
464     /// <param name="links">Хранилище
465     ↳ связей.</param>
466     /// <param name="link">Индекс
467     ↳ обновляемой связи.</param>
468     /// <param name="newSource">Индекс
469     ↳ связи, которая является началом
470     ↳ связи, на которую выполняется
471     ↳ обновление.</param>
472     /// <param name="newTarget">Индекс
473     ↳ связи, которая является концом
474     ↳ связи, на которую выполняется
475     ↳ обновление.</param>
476     /// <returns>Индекс обновлённой
477     ↳ связи.</returns>
478     [MethodImpl(MethodImplOptions.Aggressive
    ↳ eInlining)]
479     public static TLink Update<TLink>(this
480     ↳ ILinks<TLink> links, TLink link,
481     ↳ TLink newSource, TLink newTarget)
482     => links.Update(new[] { link,
483     ↳ newSource, newTarget });
484
485     /// <summary>
486     /// Обновляет связь с указанными
487     ↳ началом (Source) и концом (Target)
488     /// на связь с указанными началом
489     ↳ (NewSource) и концом (NewTarget).
490     /// </summary>
491     /// <param name="links">Хранилище
492     ↳ связей.</param>
493     /// <param name="source">Индекс связи,
494     ↳ которая является началом на
495     ↳ создаваемой связи.</param>
496     /// <param name="target">Индекс связи,
497     ↳ которая является концом для
498     ↳ создаваемой связи.</param>
499     /// <returns>Индекс связи, с указанным
500     ↳ Source (началом) и Target
501     ↳ (концом)</returns>
502     [MethodImpl(MethodImplOptions.Aggressive
    ↳ eInlining)]
503     public static TLink
504     GetOrCreate<TLink>(this
505     ↳ ILinks<TLink> links, TLink source,
506     ↳ TLink target)
507     {
508         var link =
509         ↳ links.SearchOrDefault(source,
510         ↳ target);
511         if (EqualityComparer<TLink>.Default
512         ↳ .Equals(link,
513         ↳ default))
514         {
515             link = links.CreateAndUpdate(so
516             ↳ urce,
517             ↳ target);
518         }
519         return link;
520     }
521
522     /// <summary>
523     /// Обновляет связь с указанными
524     ↳ началом (Source) и концом (Target)
525     /// на связь с указанными началом
526     ↳ (NewSource) и концом (NewTarget).
527     /// </summary>
528     /// <param name="links">Хранилище
529     ↳ связей.</param>
530     /// <param name="source">Индекс связи,
531     ↳ которая является началом
532     ↳ обновляемой связи.</param>
533     /// <param name="target">Индекс связи,
534     ↳ которая является концом обновляемой
535     ↳ связи.</param>
536     /// <param name="newSource">Индекс
537     ↳ связи, которая является началом
538     ↳ связи, на которую выполняется
539     ↳ обновление.</param>
540     /// <param name="newTarget">Индекс
541     ↳ связи, которая является концом
542     ↳ связи, на которую выполняется
543     ↳ обновление.</param>
544     /// <returns>Индекс обновлённой
545     ↳ связи.</returns>
546     [MethodImpl(MethodImplOptions.Aggressive
    ↳ eInlining)]
547     public static TLink
548     UpdateOrCreateOrGet<TLink>(this
549     ↳ ILinks<TLink> links, TLink source,
550     ↳ TLink target, TLink newSource,
551     ↳ TLink newTarget)
552     {
553         var equalityComparer =
554         ↳ EqualityComparer<TLink>.Default;
555         var link =
556         ↳ links.SearchOrDefault(source,
557         ↳ target);
558         if (equalityComparer.Equals(link,
559         ↳ default))
560         {

```

```

523         return links.CreateAndUpdate(ne
            ↪ wSource,
            ↪ newTarget);
524     }
525     if (equalityComparer.Equals(newSour
            ↪ ce, source) &&
            ↪ equalityComparer.Equals(newTarg
            ↪ et,
            ↪ target))
526     {
527         return link;
528     }
529     return links.Update(link,
            ↪ newSource, newTarget);
530 }
531
532 /// <summary>Удаляет связь с указанными
            ↪ началом (Source) и концом
            ↪ (Target).</summary>
533 /// <param name="links">Хранилище
            ↪ связей.</param>
534 /// <param name="source">Индекс связи,
            ↪ которая является началом удаляемой
            ↪ связи.</param>
535 /// <param name="target">Индекс связи,
            ↪ которая является концом удаляемой
            ↪ связи.</param>
536 [MethodImpl(MethodImplOptions.Aggressive
            ↪ eInlining)]
537 public static TLink
            ↪ DeleteIfExists<TLink>(this
            ↪ ILinks<TLink> links, TLink source,
            ↪ TLink target)
538 {
539     var link =
            ↪ links.SearchOrDefault(source,
            ↪ target);
540     if (!EqualityComparer<TLink>.Defaul
            ↪ t.Equals(link,
            ↪ default))
541     {
542         links.Delete(link);
543         return link;
544     }
545     return default;
546 }
547
548 /// <summary>Удаляет несколько
            ↪ связей.</summary>
549 /// <param name="links">Хранилище
            ↪ связей.</param>
550 /// <param name="deletedLinks">Список
            ↪ адресов связей к удалению.</param>
551 [MethodImpl(MethodImplOptions.Aggressive
            ↪ eInlining)]
552 public static void
            ↪ DeleteMany<TLink>(this
            ↪ ILinks<TLink> links, IList<TLink>
            ↪ deletedLinks)
553 {
554     for (int i = 0; i <
            ↪ deletedLinks.Count; i++)
555     {
556         links.Delete(deletedLinks[i]);
557     }
558 }
559
560 // Replace one link with another
            ↪ (replaced link is deleted, children
            ↪ are updated or deleted)
561 public static TLink Merge<TLink>(this
            ↪ ILinks<TLink> links, TLink
            ↪ linkIndex, TLink newLink)
562 {
563     var equalityComparer =
            ↪ EqualityComparer<TLink>.Default;
564     if (equalityComparer.Equals(linkInd
            ↪ ex,
            ↪ newLink))
565     {
566         return newLink;
567     }
568     var constants = links.Constants;
569     ulong referencesAsSourceCount =
            ↪ (Integer<TLink>)links.Count(con
            ↪ stants.Any, linkIndex,
            ↪ constants.Any);
570     ulong referencesAsTargetCount =
            ↪ (Integer<TLink>)links.Count(con
            ↪ stants.Any, constants.Any,
            ↪ linkIndex);
571     var isStandalonePoint =
            ↪ Point<TLink>.IsFullPoint(links.
            ↪ GetLink(linkIndex)) &&
            ↪ referencesAsSourceCount == 1 &&
            ↪ referencesAsTargetCount == 1;
572     if (!isStandalonePoint)
573     {
574         var totalReferences =
            ↪ referencesAsSourceCount +
            ↪ referencesAsTargetCount;
575         if (totalReferences > 0)
576         {
577             var references = ArrayPool.
            ↪ Allocate<TLink>((long)t
            ↪ otalReferences);
578             var referencesFiller = new
            ↪ ArrayFiller<TLink,
            ↪ TLink>(references, link
            ↪ s.Constants.Continue);
579             links.Each(referencesFiller
            ↪ .AddFirstAndReturnConst
            ↪ ant, constants.Any,
            ↪ linkIndex,
            ↪ constants.Any);
580             links.Each(referencesFiller
            ↪ .AddFirstAndReturnConst
            ↪ ant, constants.Any,
            ↪ constants.Any,
            ↪ linkIndex);
581             for (ulong i = 0; i < refer
            ↪ encesAsSourceCount;
            ↪ i++)
582             {
583                 var reference =
            ↪ references[i];
584                 if (equalityComparer.Eq
            ↪ uals(reference,
            ↪ linkIndex))
585                 {
586                     continue;
587                 }
588                 links.Update(reference,
            ↪ newLink, links.GetT
            ↪ arget(reference));
589             }
590             for (var i = (long)referenc
            ↪ esAsSourceCount; i <
            ↪ references.Length; i++)
591             {
592                 var reference =
            ↪ references[i];
593                 if (equalityComparer.Eq
            ↪ uals(reference,
            ↪ linkIndex))
594                 {
595                     continue;
596                 }
597                 links.Update(reference,
            ↪ links.GetSource(ref
            ↪ erence),
            ↪ newLink);
598             }
599             ArrayPool.Free(references);
600         }
601     }
602     links.Delete(linkIndex);
603     return newLink;
604 }
605 }
606 }
607 }
608 }

```

```

./Incrementers/FrequencyIncrementer.cs
1 using System.Collections.Generic;
2 using Platform.Interfaces;
3

```

```

4 namespace Platform.Data.Doublets.Incremeters
5 {
6     public class FrequencyIncrementer<TLink> :
7         ↳ LinksOperatorBase<TLink>,
8         ↳ IIncrementer<TLink>
9     {
10         private static readonly
11             ↳ EqualityComparer<TLink>
12             ↳ _equalityComparer =
13             ↳ EqualityComparer<TLink>.Default;
14
15         private readonly TLink _frequencyMarker;
16         private readonly TLink _unaryOne;
17         private readonly IIncrementer<TLink>
18             ↳ _unaryNumberIncrementer;
19
20     public
21         ↳ FrequencyIncrementer(ILinks<TLink>
22         ↳ links, TLink frequencyMarker, TLink
23         ↳ unaryOne, IIncrementer<TLink>
24         ↳ unaryNumberIncrementer)
25             : base(links)
26     {
27         _frequencyMarker = frequencyMarker;
28         _unaryOne = unaryOne;
29         _unaryNumberIncrementer =
30             ↳ unaryNumberIncrementer;
31     }
32
33     public TLink Increment(TLink frequency)
34     {
35         if (_equalityComparer.Equals(frequency,
36             ↳ _unaryOne,
37             ↳ default))
38         {
39             return Links.GetOrCreate(_unaryOne,
40                 ↳ _frequencyMarker);
41         }
42         var source =
43             ↳ Links.GetSource(frequency);
44         var incrementedSource = _unaryNumberIncrementer.Increment(source);
45         return Links.GetOrCreate(incrementedSource,
46             ↳ _frequencyMarker);
47     }
48 }

```

./Incremeters/LinkFrequencyIncrementer.cs

```

1 using System.Collections.Generic;
2 using Platform.Interfaces;
3
4 namespace Platform.Data.Doublets.Incremeters
5 {
6     public class
7         ↳ LinkFrequencyIncrementer<TLink> :
8         ↳ LinksOperatorBase<TLink>,
9         ↳ IIncrementer<IList<TLink>>
10     {
11         private readonly
12             ↳ ISpecificPropertyOperator<TLink,
13             ↳ TLink> _frequencyPropertyOperator;
14         private readonly IIncrementer<TLink>
15             ↳ _frequencyIncrementer;
16
17     public LinkFrequencyIncrementer(ILinks<TLink> links,
18         ↳ ISpecificPropertyOperator<TLink,
19         ↳ TLink> frequencyPropertyOperator,
20         ↳ IIncrementer<TLink> frequencyIncrementer)
21         : base(links)
22     {
23         _frequencyPropertyOperator =
24             ↳ frequencyPropertyOperator;
25         _frequencyIncrementer =
26             ↳ frequencyIncrementer;
27     }
28
29     /// <remarks>Sequence itseft is not
30     ↳ changed, only frequency of its
31     ↳ doublets is incremented.</remarks>
32     public IList<TLink>
33         ↳ Increment(IList<TLink> sequence) //
34         ↳ TODO: May be move to
35         ↳ ILinksExtensions or make
36         ↳ SequenceDoubletsFrequencyIncrementer

```

```

20 {
21     for (var i = 1; i < sequence.Count;
22         ↳ i++)
23     {
24         Increment(Links.GetOrCreate(sequence[i - 1],
25             ↳ sequence[i]));
26     }
27     return sequence;
28 }
29
30 public void Increment(TLink link)
31 {
32     var previousFrequency = _frequencyPropertyOperator.Get(link);
33     var frequency = _frequencyIncrementer.Increment(previousFrequency);
34     _frequencyPropertyOperator.Set(link, frequency);
35 }

```

./Incremeters/UnaryNumberIncrementer.cs

```

1 using System.Collections.Generic;
2 using Platform.Interfaces;
3
4 namespace Platform.Data.Doublets.Incremeters
5 {
6     public class UnaryNumberIncrementer<TLink>
7         : LinksOperatorBase<TLink>,
8         ↳ IIncrementer<TLink>
9     {
10         private static readonly
11             ↳ EqualityComparer<TLink>
12             ↳ _equalityComparer =
13             ↳ EqualityComparer<TLink>.Default;
14
15         private readonly TLink _unaryOne;
16
17     public UnaryNumberIncrementer(ILinks<TLink> links, TLink unaryOne) :
18         ↳ base(links) => _unaryOne = unaryOne;
19
20     public TLink Increment(TLink unaryNumber)
21     {
22         if (_equalityComparer.Equals(unaryNumber,
23             ↳ _unaryOne))
24         {
25             return Links.GetOrCreate(_unaryOne,
26                 ↳ _unaryOne);
27         }
28         var source =
29             ↳ Links.GetSource(unaryNumber);
30         var target =
31             ↳ Links.GetTarget(unaryNumber);
32         if (_equalityComparer.Equals(source, target))
33         {
34             return Links.GetOrCreate(unaryNumber,
35                 ↳ _unaryOne);
36         }
37         else
38         {
39             return
40                 ↳ Links.GetOrCreate(source,
41                 ↳ Increment(target));
42         }
43     }
44 }

```

./ISynchronizedLinks.cs

```

1 using Platform.Data.Constants;
2
3 namespace Platform.Data.Doublets
4 {
5     public interface ISynchronizedLinks<TLink>
6         : ISynchronizedLinks<TLink,
7         ↳ ILinks<TLink>,
8         ↳ LinksCombinedConstants<TLink, TLink,
9         ↳ int>>, ILinks<TLink>

```

```

6      {
7      }
8  }

```

./Link.cs

```

1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  using Platform.Exceptions;
5  using Platform.Ranges;
6  using Platform.Helpers.Singletons;
7  using Platform.Data.Constants;
8
9  namespace Platform.Data.Doublets
10 {
11     /// <summary>
12     /// Структура описывающая уникальную связь.
13     /// </summary>
14     public struct Link<TLink> :
15     ↪ IEquatable<Link<TLink>>,
16     ↪ IReadOnlyList<TLink>, IList<TLink>
17     {
18         public static readonly Link<TLink> Null
19         ↪ = new Link<TLink>();
20
21         private static readonly
22         ↪ LinksCombinedConstants<bool, TLink,
23         ↪ int> _constants = Default<LinksComb
24         ↪ inedConstants<bool, TLink,
25         ↪ int>>.Instance;
26         private static readonly
27         ↪ EqualityComparer<TLink>
28         ↪ _equalityComparer =
29         ↪ EqualityComparer<TLink>.Default;
30
31         private const int Length = 3;
32
33         public readonly TLink Index;
34         public readonly TLink Source;
35         public readonly TLink Target;
36
37         public Link(params TLink[] values)
38         {
39             Index = values.Length >
40             ↪ _constants.IndexPart ?
41             ↪ values[_constants.IndexPart] :
42             ↪ _constants.Null;
43             Source = values.Length >
44             ↪ _constants.SourcePart ?
45             ↪ values[_constants.SourcePart] :
46             ↪ _constants.Null;
47             Target = values.Length >
48             ↪ _constants.TargetPart ?
49             ↪ values[_constants.TargetPart] :
50             ↪ _constants.Null;
51         }
52
53         public Link(IList<TLink> values)
54         {
55             Index = values.Count >
56             ↪ _constants.IndexPart ?
57             ↪ values[_constants.IndexPart] :
58             ↪ _constants.Null;
59             Source = values.Count >
60             ↪ _constants.SourcePart ?
61             ↪ values[_constants.SourcePart] :
62             ↪ _constants.Null;
63             Target = values.Count >
64             ↪ _constants.TargetPart ?
65             ↪ values[_constants.TargetPart] :
66             ↪ _constants.Null;
67         }
68
69         public Link(TLink index, TLink source,
70             ↪ TLink target)
71         {
72             Index = index;
73             Source = source;
74             Target = target;
75         }
76
77         public Link(TLink source, TLink target)
78         : this(_constants.Null, source,
79             ↪ target)
80         {
81             Source = source;
82             Target = target;
83         }
84     }
85 }

```

```

55     public static Link<TLink> Create(TLink
60     ↪ source, TLink target) => new
61     ↪ Link<TLink>(source, target);
62
63     public override int GetHashCode() =>
64     ↪ (Index, Source,
65     ↪ Target).GetHashCode();
66
67     public bool IsNull() =>
68     ↪ _equalityComparer.Equals(Index,
69     ↪ _constants.Null)
70     && _equalityCompar
71     ↪ er.Equals(Sour
72     ↪ ce,
73     ↪ _constants.Nul
74     ↪ l)
75     && _equalityCompar
76     ↪ er.Equals(Targ
77     ↪ et,
78     ↪ _constants.Nul
79     ↪ l);
80
81     public override bool Equals(object
82     ↪ other) => other is Link<TLink> &&
83     ↪ Equals((Link<TLink>)other);
84
85     public bool Equals(Link<TLink> other)
86     ↪ => _equalityComparer.Equals(Index,
87     ↪ other.Index)
88     &
89     ↪
90     ↪
91     ↪
92     ↪
93     ↪
94     ↪
95     ↪
96     ↪
97     ↪
98     ↪
99     ↪
100    ↪
101    ↪
102    ↪
103    ↪
104    ↪
105    ↪
106    ↪
107    ↪
108    ↪
109    ↪
110    ↪
111    ↪
112    ↪
113    ↪
114    ↪
115    ↪
116    ↪
117    ↪
118    ↪
119    ↪
120    ↪
121    ↪
122    ↪
123    ↪
124    ↪
125    ↪
126    ↪
127    ↪
128    ↪
129    ↪
130    ↪
131    ↪
132    ↪
133    ↪
134    ↪
135    ↪
136    ↪
137    ↪
138    ↪
139    ↪
140    ↪
141    ↪
142    ↪
143    ↪
144    ↪
145    ↪
146    ↪
147    ↪
148    ↪
149    ↪
150    ↪
151    ↪
152    ↪
153    ↪
154    ↪
155    ↪
156    ↪
157    ↪
158    ↪
159    ↪
160    ↪
161    ↪
162    ↪
163    ↪
164    ↪
165    ↪
166    ↪
167    ↪
168    ↪
169    ↪
170    ↪
171    ↪
172    ↪
173    ↪
174    ↪
175    ↪
176    ↪
177    ↪
178    ↪
179    ↪
180    ↪
181    ↪
182    ↪
183    ↪
184    ↪
185    ↪
186    ↪
187    ↪
188    ↪
189    ↪
190    ↪
191    ↪
192    ↪
193    ↪
194    ↪
195    ↪
196    ↪
197    ↪
198    ↪
199    ↪
200    ↪
201    ↪
202    ↪
203    ↪
204    ↪
205    ↪
206    ↪
207    ↪
208    ↪
209    ↪
210    ↪
211    ↪
212    ↪
213    ↪
214    ↪
215    ↪
216    ↪
217    ↪
218    ↪
219    ↪
220    ↪
221    ↪
222    ↪
223    ↪
224    ↪
225    ↪
226    ↪
227    ↪
228    ↪
229    ↪
230    ↪
231    ↪
232    ↪
233    ↪
234    ↪
235    ↪
236    ↪
237    ↪
238    ↪
239    ↪
240    ↪
241    ↪
242    ↪
243    ↪
244    ↪
245    ↪
246    ↪
247    ↪
248    ↪
249    ↪
250    ↪
251    ↪
252    ↪
253    ↪
254    ↪
255    ↪
256    ↪
257    ↪
258    ↪
259    ↪
260    ↪
261    ↪
262    ↪
263    ↪
264    ↪
265    ↪
266    ↪
267    ↪
268    ↪
269    ↪
270    ↪
271    ↪
272    ↪
273    ↪
274    ↪
275    ↪
276    ↪
277    ↪
278    ↪
279    ↪
280    ↪
281    ↪
282    ↪
283    ↪
284    ↪
285    ↪
286    ↪
287    ↪
288    ↪
289    ↪
290    ↪
291    ↪
292    ↪
293    ↪
294    ↪
295    ↪
296    ↪
297    ↪
298    ↪
299    ↪
300    ↪
301    ↪
302    ↪
303    ↪
304    ↪
305    ↪
306    ↪
307    ↪
308    ↪
309    ↪
310    ↪
311    ↪
312    ↪
313    ↪
314    ↪
315    ↪
316    ↪
317    ↪
318    ↪
319    ↪
320    ↪
321    ↪
322    ↪
323    ↪
324    ↪
325    ↪
326    ↪
327    ↪
328    ↪
329    ↪
330    ↪
331    ↪
332    ↪
333    ↪
334    ↪
335    ↪
336    ↪
337    ↪
338    ↪
339    ↪
340    ↪
341    ↪
342    ↪
343    ↪
344    ↪
345    ↪
346    ↪
347    ↪
348    ↪
349    ↪
350    ↪
351    ↪
352    ↪
353    ↪
354    ↪
355    ↪
356    ↪
357    ↪
358    ↪
359    ↪
360    ↪
361    ↪
362    ↪
363    ↪
364    ↪
365    ↪
366    ↪
367    ↪
368    ↪
369    ↪
370    ↪
371    ↪
372    ↪
373    ↪
374    ↪
375    ↪
376    ↪
377    ↪
378    ↪
379    ↪
380    ↪
381    ↪
382    ↪
383    ↪
384    ↪
385    ↪
386    ↪
387    ↪
388    ↪
389    ↪
390    ↪
391    ↪
392    ↪
393    ↪
394    ↪
395    ↪
396    ↪
397    ↪
398    ↪
399    ↪
400    ↪
401    ↪
402    ↪
403    ↪
404    ↪
405    ↪
406    ↪
407    ↪
408    ↪
409    ↪
410    ↪
411    ↪
412    ↪
413    ↪
414    ↪
415    ↪
416    ↪
417    ↪
418    ↪
419    ↪
420    ↪
421    ↪
422    ↪
423    ↪
424    ↪
425    ↪
426    ↪
427    ↪
428    ↪
429    ↪
430    ↪
431    ↪
432    ↪
433    ↪
434    ↪
435    ↪
436    ↪
437    ↪
438    ↪
439    ↪
440    ↪
441    ↪
442    ↪
443    ↪
444    ↪
445    ↪
446    ↪
447    ↪
448    ↪
449    ↪
450    ↪
451    ↪
452    ↪
453    ↪
454    ↪
455    ↪
456    ↪
457    ↪
458    ↪
459    ↪
460    ↪
461    ↪
462    ↪
463    ↪
464    ↪
465    ↪
466    ↪
467    ↪
468    ↪
469    ↪
470    ↪
471    ↪
472    ↪
473    ↪
474    ↪
475    ↪
476    ↪
477    ↪
478    ↪
479    ↪
480    ↪
481    ↪
482    ↪
483    ↪
484    ↪
485    ↪
486    ↪
487    ↪
488    ↪
489    ↪
490    ↪
491    ↪
492    ↪
493    ↪
494    ↪
495    ↪
496    ↪
497    ↪
498    ↪
499    ↪
500    ↪
501    ↪
502    ↪
503    ↪
504    ↪
505    ↪
506    ↪
507    ↪
508    ↪
509    ↪
510    ↪
511    ↪
512    ↪
513    ↪
514    ↪
515    ↪
516    ↪
517    ↪
518    ↪
519    ↪
520    ↪
521    ↪
522    ↪
523    ↪
524    ↪
525    ↪
526    ↪
527    ↪
528    ↪
529    ↪
530    ↪
531    ↪
532    ↪
533    ↪
534    ↪
535    ↪
536    ↪
537    ↪
538    ↪
539    ↪
540    ↪
541    ↪
542    ↪
543    ↪
544    ↪
545    ↪
546    ↪
547    ↪
548    ↪
549    ↪
550    ↪
551    ↪
552    ↪
553    ↪
554    ↪
555    ↪
556    ↪
557    ↪
558    ↪
559    ↪
560    ↪
561    ↪
562    ↪
563    ↪
564    ↪
565    ↪
566    ↪
567    ↪
568    ↪
569    ↪
570    ↪
571    ↪
572    ↪
573    ↪
574    ↪
575    ↪
576    ↪
577    ↪
578    ↪
579    ↪
580    ↪
581    ↪
582    ↪
583    ↪
584    ↪
585    ↪
586    ↪
587    ↪
588    ↪
589    ↪
590    ↪
591    ↪
592    ↪
593    ↪
594    ↪
595    ↪
596    ↪
597    ↪
598    ↪
599    ↪
600    ↪
601    ↪
602    ↪
603    ↪
604    ↪
605    ↪
606    ↪
607    ↪
608    ↪
609    ↪
610    ↪
611    ↪
612    ↪
613    ↪
614    ↪
615    ↪
616    ↪
617    ↪
618    ↪
619    ↪
620    ↪
621    ↪
622    ↪
623    ↪
624    ↪
625    ↪
626    ↪
627    ↪
628    ↪
629    ↪
630    ↪
631    ↪
632    ↪
633    ↪
634    ↪
635    ↪
636    ↪
637    ↪
638    ↪
639    ↪
640    ↪
641    ↪
642    ↪
643    ↪
644    ↪
645    ↪
646    ↪
647    ↪
648    ↪
649    ↪
650    ↪
651    ↪
652    ↪
653    ↪
654    ↪
655    ↪
656    ↪
657    ↪
658    ↪
659    ↪
660    ↪
661    ↪
662    ↪
663    ↪
664    ↪
665    ↪
666    ↪
667    ↪
668    ↪
669    ↪
670    ↪
671    ↪
672    ↪
673    ↪
674    ↪
675    ↪
676    ↪
677    ↪
678    ↪
679    ↪
680    ↪
681    ↪
682    ↪
683    ↪
684    ↪
685    ↪
686    ↪
687    ↪
688    ↪
689    ↪
690    ↪
691    ↪
692    ↪
693    ↪
694    ↪
695    ↪
696    ↪
697    ↪
698    ↪
699    ↪
700    ↪
701    ↪
702    ↪
703    ↪
704    ↪
705    ↪
706    ↪
707    ↪
708    ↪
709    ↪
710    ↪
711    ↪
712    ↪
713    ↪
714    ↪
715    ↪
716    ↪
717    ↪
718    ↪
719    ↪
720    ↪
721    ↪
722    ↪
723    ↪
724    ↪
725    ↪
726    ↪
727    ↪
728    ↪
729    ↪
730    ↪
731    ↪
732    ↪
733    ↪
734    ↪
735    ↪
736    ↪
737    ↪
738    ↪
739    ↪
740    ↪
741    ↪
742    ↪
743    ↪
744    ↪
745    ↪
746    ↪
747    ↪
748    ↪
749    ↪
750    ↪
751    ↪
752    ↪
753    ↪
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    ↪
782    ↪
783    ↪
784    ↪
785    ↪
786    ↪
787    ↪
788    ↪
789    ↪
790    ↪
791    ↪
792    ↪
793    ↪
794    ↪
795    ↪
796    ↪
797    ↪
798    ↪
799    ↪
800    ↪
801    ↪
802    ↪
803    ↪
804    ↪
805    ↪
806    ↪
807    ↪
808    ↪
809    ↪
810    ↪
811    ↪
812    ↪
813    ↪
814    ↪
815    ↪
816    ↪
817    ↪
818    ↪
819    ↪
820    ↪
821    ↪
822    ↪
823    ↪
824    ↪
825    ↪
826    ↪
827    ↪
828    ↪
829    ↪
830    ↪
831    ↪
832    ↪
833    ↪
834    ↪
835    ↪
836    ↪
837    ↪
838    ↪
839    ↪
840    ↪
841    ↪
842    ↪
843    ↪
844    ↪
845    ↪
846    ↪
847    ↪
848    ↪
849    ↪
850    ↪
851    ↪
852    ↪
853    ↪
854    ↪
855    ↪
856    ↪
857    ↪
858    ↪
859    ↪
860    ↪
861    ↪
862    ↪
863    ↪
864    ↪
865    ↪
866    ↪
867    ↪
868    ↪
869    ↪
870    ↪
871    ↪
872    ↪
873    ↪
874    ↪
875    ↪
876    ↪
877    ↪
878    ↪
879    ↪
880    ↪
881    ↪
882    ↪
883    ↪
884    ↪
885    ↪
886    ↪
887    ↪
888    ↪
889    ↪
890    ↪
891    ↪
892    ↪
893    ↪
894    ↪
895    ↪
896    ↪
897    ↪
898    ↪
899    ↪
900    ↪
901    ↪
902    ↪
903    ↪
904    ↪
905    ↪
906    ↪
907    ↪
908    ↪
909    ↪
910    ↪
911    ↪
912    ↪
913    ↪
914    ↪
915    ↪
916    ↪
917    ↪
918    ↪
919    ↪
920    ↪
921    ↪
922    ↪
923    ↪
924    ↪
925    ↪
926    ↪
927    ↪
928    ↪
929    ↪
930    ↪
931    ↪
932    ↪
933    ↪
934    ↪
935    ↪
936    ↪
937    ↪
938    ↪
939    ↪
940    ↪
941    ↪
942    ↪
943    ↪
944    ↪
945    ↪
946    ↪
947    ↪
948    ↪
949    ↪
950    ↪
951    ↪
952    ↪
953    ↪
954    ↪
955    ↪
956    ↪
957    ↪
958    ↪
959    ↪
960    ↪
961    ↪
962    ↪
963    ↪
964    ↪
965    ↪
966    ↪
967    ↪
968    ↪
969    ↪
970    ↪
971    ↪
972    ↪
973    ↪
974    ↪
975    ↪
976    ↪
977    ↪
978    ↪
979    ↪
980    ↪
981    ↪
982    ↪
983    ↪
984    ↪
985    ↪
986    ↪
987    ↪
988    ↪
989    ↪
990    ↪
991    ↪
992    ↪
993    ↪
994    ↪
995    ↪
996    ↪
997    ↪
998    ↪
999    ↪

```

84
 & j
 j
 e
 q
 u
 a
 i
 y
 C
 o
 m
 p
 a
 r
 e
 r
 98
 99
 E
 q
 u
 a
 s
 s
 ()
 T
 a
 r
 g
 e
 t
 11
 12
 o
 t
 h
 e
 r
 17
 18
 T
 a
 r
 g
 e
 t
 22
 23
 t
 24
) 25
 ;
 126
 127
)
) "
 128
 129
 130
 131
 132
 133
 134
 135
 136
 137
 138
 139
 140
 141

```

public override string ToString() =>
    _equalityComparer.Equals(Index,
    ↪ _constants.Null) ? ToString(Source,
    ↪ Target) : ToString(Index, Source,
    ↪ Target);

#region IList

public int Count => Length;

public bool IsReadOnly => true;

public TLink this[int index]
{
    get
    {
        Ensure.Always.ArgumentInRange(i
        ↪ ndex, new Range<int>(0,
        ↪ Length - 1), nameof(index));
        if (index ==
        ↪ _constants.IndexPart)
        {
            return Index;
        }
        if (index ==
        ↪ _constants.SourcePart)
        {
            return Source;
        }
        if (index ==
        ↪ _constants.TargetPart)
        {
            return Target;
        }
        throw new
        ↪ NotSupportedException(); //
        ↪ Impossible path due to
        ↪ Ensure.ArgumentInRange
    }
    set => throw new
    ↪ NotSupportedException();
}

IEnumerator IEnumerable.GetEnumerator()
    ↪ => GetEnumerator();

public IEnumerator<TLink>
    ↪ GetEnumerator()
{
    yield return Index;
    yield return Source;
    yield return Target;
}

public void Add(TLink item) => throw
    ↪ new NotSupportedException();

public void Clear() => throw new
    ↪ NotSupportedException();

public bool Contains(TLink item) =>
    ↪ IndexOf(item) >= 0;

public void CopyTo(TLink[] array, int
    ↪ arrayIndex)
{
    Ensure.Always.ArgumentNotNull(array
    ↪ ,
    ↪ nameof(array));
    Ensure.Always.ArgumentInRange(array
    ↪ Index, new Range<int>(0,
    ↪ array.Length - 1),
    ↪ nameof(arrayIndex));
    if (arrayIndex + Length >
    ↪ array.Length)
    {
        throw new
        ↪ InvalidOperationException();
    }
    array[arrayIndex++] = Index;
    array[arrayIndex++] = Source;
    array[arrayIndex] = Target;
}

```



```

142     public bool Remove(TLink item) =>
        ↪ Throw.A.NotSupportedExceptionAndRet
        ↪ urn<bool>();
143
144     public int IndexOf(TLink item)
145     {
146         if (_equalityComparer.Equals(Index,
        ↪ item))
147         {
148             return _constants.IndexPart;
149         }
150         if (_equalityComparer.Equals(Source,
        ↪ item))
151         {
152             return _constants.SourcePart;
153         }
154         if (_equalityComparer.Equals(Target,
        ↪ item))
155         {
156             return _constants.TargetPart;
157         }
158         return -1;
159     }
160
161     public void Insert(int index, TLink
        ↪ item) => throw new
        ↪ NotSupportedException();
162
163     public void RemoveAt(int index) =>
        ↪ throw new NotSupportedException();
164
165     #endregion
166 }
167 }

```

./LinkExtensions.cs

```

1 namespace Platform.Data.Doublets
2 {
3     public static class LinkExtensions
4     {
5         public static bool
        ↪ IsFullPoint<TLink>(this Link<TLink>
        ↪ link) =>
        ↪ Point<TLink>.IsFullPoint(link);
6         public static bool
        ↪ IsPartialPoint<TLink>(this
        ↪ Link<TLink> link) =>
        ↪ Point<TLink>.IsPartialPoint(link);
7     }
8 }

```

./LinksOperatorBase.cs

```

1 namespace Platform.Data.Doublets
2 {
3     public abstract class
        ↪ LinksOperatorBase<TLink>
4     {
5         protected readonly ILinks<TLink> Links;
6         protected
        ↪ LinksOperatorBase(ILinks<TLink>
        ↪ links) => Links = links;
7     }
8 }

```

./obj/Debug/netstandard2.0/Platform.Data.Doublets.AssemblyInfo.cs

```

1 //-----
2 // <auto-generated>
3 // Generated by the MSBuild
        ↪ WriteCodeFragment class.
4 // </auto-generated>
5 //-----
6
7 using System;
8 using System.Reflection;
9
10 [assembly: System.Reflection.AssemblyConfigurat
        ↪ ionAttribute("Debug")]
11 [assembly: System.Reflection.AssemblyCopyrightA
        ↪ ttribute("Konstantin
        ↪ Diachenko")]
12 [assembly: System.Reflection.AssemblyDescriptio
        ↪ nAttribute("LinksPlatform\'s
        ↪ Platform.Data.Doublets Class Library")]

```

```

13 [assembly: System.Reflection.AssemblyFileVersio
        ↪ nAttribute("0.0.1.0")]
14 [assembly: System.Reflection.AssemblyInformatio
        ↪ nalVersionAttribute("0.0.1")]
15 [assembly: System.Reflection.AssemblyTitleAttri
        ↪ bute("Platform.Data.Doublets")]
16 [assembly: System.Reflection.AssemblyVersionAtt
        ↪ ribute("0.0.1.0")]

```

./PropertyOperators/DefaultLinkPropertyOperator.cs

```

1 using System.Linq;
2 using System.Collections.Generic;
3 using Platform.Interfaces;
4
5 namespace
        ↪ Platform.Data.Doublets.PropertyOperators
6 {
7     public class
        ↪ DefaultLinkPropertyOperator<TLink> :
        ↪ LinksOperatorBase<TLink>,
        ↪ IPropertyOperator<TLink, TLink, TLink>
8     {
9         private static readonly
        ↪ EqualityComparer<TLink>
        ↪ _equalityComparer =
        ↪ EqualityComparer<TLink>.Default;
10
11         public DefaultLinkPropertyOperator(ILin
        ↪ ks<TLink> links) :
        ↪ base(links)
12         {
13         }
14
15         public TLink GetValue(TLink @object,
        ↪ TLink property)
16         {
17             var objectProperty =
        ↪ Links.SearchOrDefault(@object,
        ↪ property);
18             if (_equalityComparer.Equals(object
        ↪ Property,
        ↪ default))
19             {
20                 return default;
21             }
22             var valueLink =
        ↪ Links.All(Links.Constants.Any,
        ↪ objectProperty).SingleOrDefault
        ↪ ();
23             if (valueLink == null)
24             {
25                 return default;
26             }
27             var value = Links.GetTarget(valueLi
        ↪ nk[Links.Constants.IndexPart]);
28             return value;
29         }
30
31         public void SetValue(TLink @object,
        ↪ TLink property, TLink value)
32         {
33             var objectProperty =
        ↪ Links.GetOrCreate(@object,
        ↪ property);
34             Links.DeleteMany(Links.All(Links.Co
        ↪ nstants.Any,
        ↪ objectProperty).Select(link =>
        ↪ link[Links.Constants.IndexPart]
        ↪ ).ToList());
35             Links.GetOrCreate(objectProperty,
        ↪ value);
36         }
37     }
38 }

```

```

39
40 public void SetValue(TLink @object,
        ↪ TLink property, TLink value)
41 {
42     var objectProperty =
        ↪ Links.GetOrCreate(@object,
        ↪ property);
43     Links.DeleteMany(Links.All(Links.Co
        ↪ nstants.Any,
        ↪ objectProperty).Select(link =>
        ↪ link[Links.Constants.IndexPart]
        ↪ ).ToList());
44     Links.GetOrCreate(objectProperty,
        ↪ value);
45 }
46 }
47 }
48 }

```

./PropertyOperators/FrequencyPropertyOperator.cs

```

1 using System.Collections.Generic;
2 using Platform.Interfaces;
3
4 namespace
        ↪ Platform.Data.Doublets.PropertyOperators
5 {
6     public class
        ↪ FrequencyPropertyOperator<TLink> :
        ↪ LinksOperatorBase<TLink>,
        ↪ ISpecificPropertyOperator<TLink, TLink>

```



```

46     public static readonly int
        RightAsTargetOffset =
        ↪ Marshal.OffsetOf(typeof(Link),
        ↪ nameof(RightAsTarget)).ToInt32(
        ↪ );
47     public static readonly int
        SizeAsTargetOffset =
        ↪ Marshal.OffsetOf(typeof(Link),
        ↪ nameof(SizeAsTarget)).ToInt32();
48
49     public TLink Source;
50     public TLink Target;
51     public TLink LeftAsSource;
52     public TLink RightAsSource;
53     public TLink SizeAsSource;
54     public TLink LeftAsTarget;
55     public TLink RightAsTarget;
56     public TLink SizeAsTarget;
57
58     [MethodImpl(MethodImplOptions.AggressiveInlining)]
59     public static TLink
        GetSource(IntPtr pointer) =>
        ↪ (pointer +
        ↪ SourceOffset).GetValue<TLink>();
60     [MethodImpl(MethodImplOptions.AggressiveInlining)]
61     public static TLink
        GetTarget(IntPtr pointer) =>
        ↪ (pointer +
        ↪ TargetOffset).GetValue<TLink>();
62     [MethodImpl(MethodImplOptions.AggressiveInlining)]
63     public static TLink
        GetLeftAsSource(IntPtr pointer)
        ↪ => (pointer + LeftAsSourceOffset).GetValue<TLink>();
64     [MethodImpl(MethodImplOptions.AggressiveInlining)]
65     public static TLink
        GetRightAsSource(IntPtr
        ↪ pointer) => (pointer + RightAsSourceOffset).GetValue<TLink>();
66     [MethodImpl(MethodImplOptions.AggressiveInlining)]
67     public static TLink
        GetSizeAsSource(IntPtr pointer)
        ↪ => (pointer + SizeAsSourceOffset).GetValue<TLink>();
68     [MethodImpl(MethodImplOptions.AggressiveInlining)]
69     public static TLink
        GetLeftAsTarget(IntPtr pointer)
        ↪ => (pointer + LeftAsTargetOffset).GetValue<TLink>();
70     [MethodImpl(MethodImplOptions.AggressiveInlining)]
71     public static TLink
        GetRightAsTarget(IntPtr
        ↪ pointer) => (pointer + RightAsTargetOffset).GetValue<TLink>();
72     [MethodImpl(MethodImplOptions.AggressiveInlining)]
73     public static TLink
        GetSizeAsTarget(IntPtr pointer)
        ↪ => (pointer + SizeAsTargetOffset).GetValue<TLink>();
74     [MethodImpl(MethodImplOptions.AggressiveInlining)]
75     public static void SetSource(IntPtr
        ↪ pointer, TLink value) =>
        ↪ (pointer +
        ↪ SourceOffset).SetValue(value);
76     [MethodImpl(MethodImplOptions.AggressiveInlining)]
77     public static void SetTarget(IntPtr
        ↪ pointer, TLink value) =>
        ↪ (pointer +
        ↪ TargetOffset).SetValue(value);
78     [MethodImpl(MethodImplOptions.AggressiveInlining)]
79
80     public static void
        SetLeftAsSource(IntPtr pointer,
        ↪ TLink value) => (pointer + Left
        ↪ AsSourceOffset).SetValue(value);
81     [MethodImpl(MethodImplOptions.AggressiveInlining)]
82     public static void
        SetRightAsSource(IntPtr
        ↪ pointer, TLink value) =>
        ↪ (pointer + RightAsSourceOffset).SetValue(value);
83     [MethodImpl(MethodImplOptions.AggressiveInlining)]
84     public static void
        SetSizeAsSource(IntPtr pointer,
        ↪ TLink value) => (pointer + Size
        ↪ AsSourceOffset).SetValue(value);
85     [MethodImpl(MethodImplOptions.AggressiveInlining)]
86     public static void
        SetLeftAsTarget(IntPtr pointer,
        ↪ TLink value) => (pointer + Left
        ↪ AsTargetOffset).SetValue(value);
87     [MethodImpl(MethodImplOptions.AggressiveInlining)]
88     public static void
        SetRightAsTarget(IntPtr
        ↪ pointer, TLink value) =>
        ↪ (pointer + RightAsTargetOffset).SetValue(value);
89     [MethodImpl(MethodImplOptions.AggressiveInlining)]
90     public static void
        SetSizeAsTarget(IntPtr pointer,
        ↪ TLink value) => (pointer + Size
        ↪ AsTargetOffset).SetValue(value);
91     }
92
93     private struct LinksHeader
94     {
95         public static readonly int
            AllocatedLinksOffset = Marshal.
            ↪ OffsetOf(typeof(LinksHeader),
            ↪ nameof(AllocatedLinks)).ToInt32(
            ↪ );
96         public static readonly int
            ReservedLinksOffset = Marshal.
            ↪ OffsetOf(typeof(LinksHeader),
            ↪ nameof(ReservedLinks)).ToInt32(
            ↪ );
97         public static readonly int
            FreeLinksOffset = Marshal.
            ↪ OffsetOf(typeof(LinksHeader),
            ↪ nameof(FreeLinks)).ToInt32(
            ↪ );
98         public static readonly int
            FirstFreeLinkOffset = Marshal.
            ↪ OffsetOf(typeof(LinksHeader),
            ↪ nameof(FirstFreeLink)).ToInt32(
            ↪ );
99         public static readonly int
            FirstAsSourceOffset = Marshal.
            ↪ OffsetOf(typeof(LinksHeader),
            ↪ nameof(FirstAsSource)).ToInt32(
            ↪ );
100        public static readonly int
            FirstAsTargetOffset = Marshal.
            ↪ OffsetOf(typeof(LinksHeader),
            ↪ nameof(FirstAsTarget)).ToInt32(
            ↪ );
101        public static readonly int
            LastFreeLinkOffset = Marshal.
            ↪ OffsetOf(typeof(LinksHeader),
            ↪ nameof(LastFreeLink)).ToInt32(
            ↪ );
102
103        public TLink AllocatedLinks;
104        public TLink ReservedLinks;
105        public TLink FreeLinks;
106        public TLink FirstFreeLink;
107        public TLink FirstAsSource;
108        public TLink FirstAsTarget;
109        public TLink LastFreeLink;

```

```

110         public TLink Reserved8;
111
112         [MethodImpl(MethodImplOptions.AggressiveInlining)]
113         public static TLink
114             GetAllocatedLinks(IntPtr
115                 pointer) => (pointer + Allocate
116                 dLinksOffset).GetValue<TLink>();
117         [MethodImpl(MethodImplOptions.AggressiveInlining)]
118         public static TLink
119             GetReservedLinks(IntPtr
120                 pointer) => (pointer + Reserved
121                 LinksOffset).GetValue<TLink>();
122         [MethodImpl(MethodImplOptions.AggressiveInlining)]
123         public static TLink
124             GetFreeLinks(IntPtr pointer) =>
125             (pointer + FreeLinksOffset).Get
126             Value<TLink>();
127         [MethodImpl(MethodImplOptions.AggressiveInlining)]
128         public static TLink
129             GetFirstFreeLink(IntPtr
130                 pointer) => (pointer + FirstFre
131                 eLinkOffset).GetValue<TLink>();
132         [MethodImpl(MethodImplOptions.AggressiveInlining)]
133         public static TLink
134             GetFirstAsSource(IntPtr
135                 pointer) => (pointer + FirstAsS
136                 ourceOffset).GetValue<TLink>();
137         [MethodImpl(MethodImplOptions.AggressiveInlining)]
138         public static TLink
139             GetFirstAsTarget(IntPtr
140                 pointer) => (pointer + FirstAsT
141                 argetOffset).GetValue<TLink>();
142         [MethodImpl(MethodImplOptions.AggressiveInlining)]
143         public static TLink
144             GetLastFreeLink(IntPtr pointer)
145             => (pointer + LastFreeLinkOffse
146             t).GetValue<TLink>();
147
148         [MethodImpl(MethodImplOptions.AggressiveInlining)]
149         public static IntPtr
150             GetFirstAsSourcePointer(IntPtr
151                 pointer) => pointer +
152             FirstAsSourceOffset;
153         [MethodImpl(MethodImplOptions.AggressiveInlining)]
154         public static IntPtr
155             GetFirstAsTargetPointer(IntPtr
156                 pointer) => pointer +
157             FirstAsTargetOffset;
158
159         [MethodImpl(MethodImplOptions.AggressiveInlining)]
160         public static void
161             SetAllocatedLinks(IntPtr
162                 pointer, TLink value) =>
163             (pointer + AllocatedLinksOffset
164             ).SetValue(value);
165         [MethodImpl(MethodImplOptions.AggressiveInlining)]
166         public static void
167             SetReservedLinks(IntPtr
168                 pointer, TLink value) =>
169             (pointer + ReservedLinksOffset)
170             .SetValue(value);
171         [MethodImpl(MethodImplOptions.AggressiveInlining)]
172         public static void
173             SetFreeLinks(IntPtr pointer,
174                 TLink value) => (pointer + Free
175                 LinksOffset).SetValue(value);
176         [MethodImpl(MethodImplOptions.AggressiveInlining)]
177         public static void
178             SetFirstFreeLink(IntPtr
179                 pointer, TLink value) =>
180             (pointer + FirstFreeLinkOffset)
181             .SetValue(value);
182         [MethodImpl(MethodImplOptions.AggressiveInlining)]
183         public static void
184             SetFirstAsSource(IntPtr
185                 pointer, TLink value) =>
186             (pointer + FirstAsSourceOffset)
187             .SetValue(value);
188         [MethodImpl(MethodImplOptions.AggressiveInlining)]
189         public static void
190             SetFirstAsTarget(IntPtr
191                 pointer, TLink value) =>
192             (pointer + FirstAsTargetOffset)
193             .SetValue(value);
194         [MethodImpl(MethodImplOptions.AggressiveInlining)]
195         public static void
196             SetLastFreeLink(IntPtr pointer,
197                 TLink value) => (pointer + Last
198                 FreeLinkOffset).SetValue(value);
199     }
200
201     private readonly long
202         _memoryReservationStep;
203
204     private readonly IResizableDirectMemory
205         _memory;
206     private IntPtr _header;
207     private IntPtr _links;
208
209     private LinksTargetsTreeMethods
210         _targetsTreeMethods;
211     private LinksSourcesTreeMethods
212         _sourcesTreeMethods;
213
214     // TODO: Возможно чтобы гарантированно
215     // проверять на то, является ли связь
216     // удалённой, нужно использовать не
217     // список а дерево, так как так можно
218     // быстрее проверить на наличие связи
219     // внутри
220     private UnusedLinksListMethods
221         _unusedLinksListMethods;
222
223     /// <summary>
224     /// Возвращает общее число связей
225     /// находящихся в хранилище.
226     /// </summary>
227     private TLink Total => Subtract(LinksHe
228         ader.GetAllocatedLinks(_header),
229         LinksHeader.GetFreeLinks(_header));
230
231     public LinksCombinedConstants<TLink,
232         int> Constants { get; }
233
234     public
235         ResizableDirectMemoryLinks(string
236         address)
237         : this(address,
238             DefaultLinksSizeStep)
239     {
240     }
241
242     /// <summary>
243     /// Создаёт экземпляр базы данных Links
244     /// в файле по указанному адресу, с
245     /// указанным минимальным шагом
246     /// расширения базы данных.
247     /// </summary>
248     /// <param name="address">Полный путь
249     /// к файлу базы данных.</param>
250     /// <param name="memoryReservationStep">
251     /// >Минимальный шаг расширения базы
252     /// данных в байтах.</param>
253     public
254         ResizableDirectMemoryLinks(string
255         address, long memoryReservationStep)
256         : this(new FileMappedResizableDirec
257             tMemory(address,
258                 memoryReservationStep),
259                 memoryReservationStep)

```

```

179 {
180 }
181
182 public ResizableDirectMemoryLinks(IResi
    ↪ zableDirectMemory
    ↪ memory)
    : this(memory, DefaultLinksSizeStep)
183 {
184 }
185
186 public ResizableDirectMemoryLinks(IResi
    ↪ zableDirectMemory memory, long
    ↪ memoryReservationStep)
187 {
188     Constants = Default<LinksCombinedCo
    ↪ nstants<TLink, TLink,
    ↪ int>>.Instance;
189     _memory = memory;
    _memoryReservationStep =
    ↪ memoryReservationStep;
190 if (memory.ReservedCapacity <
    ↪ memoryReservationStep)
191 {
192     memory.ReservedCapacity =
    ↪ memoryReservationStep;
193 }
194 SetPointers(_memory);
    // Гарантия корректности
    ↪ _memory.UsedCapacity
    ↪ относительно
    ↪ _header->AllocatedLinks
195 _memory.UsedCapacity =
    ↪ ((long) (Integer<TLink>)LinksHea
    ↪ der.GetAllocatedLinks(_header)
    ↪ * LinkSizeInBytes) +
    ↪ LinkHeaderSizeInBytes;
196 // Гарантия корректности
    ↪ _header->ReservedLinks
    ↪ относительно
    ↪ _memory.ReservedCapacity
197 LinksHeader.SetReservedLinks(_heade
    ↪ r,
    ↪ (Integer<TLink>)((_memory.Reser
    ↪ vedCapacity -
    ↪ LinkHeaderSizeInBytes) /
    ↪ LinkSizeInBytes));
198 }
199
200 [MethodImpl(MethodImplOptions.Aggressive
    ↪ eInlining)]
201 public TLink Count(IList<TLink>
    ↪ restrictions)
202 {
203     // Если нет ограничений, тогда
    ↪ возвращаем общее число связей
    ↪ находящихся в хранилище.
204 if (restrictions.Count == 0)
205 {
206     return Total;
207 }
208 if (restrictions.Count == 1)
209 {
210     var index = restrictions[Consta
    ↪ nts.IndexPart];
211     if (_equalityComparer.Equals(in
    ↪ dex,
    ↪ Constants.Any))
212     {
213         return Total;
214     }
215     return Exists(index) ?
    ↪ Integer<TLink>.One :
    ↪ Integer<TLink>.Zero;
216 }
217 if (restrictions.Count == 2)
218 {
219     var index = restrictions[Consta
    ↪ nts.IndexPart];
220     var value = restrictions[1];
221     if (_equalityComparer.Equals(in
    ↪ dex,
    ↪ Constants.Any))
222     {
223         if (_equalityComparer.Equal
    ↪ s(value,
    ↪ Constants.Any))
224         {
225             return Total;
226         }
227     }
228     return Add(_sourcesTreeMeth
    ↪ ods.CalculateReferences
    ↪ (value),
    ↪ _targetsTreeMethods.Cal
    ↪ culateReferences(value)
    ↪ );
229 }
230 else
231 {
232     if (!Exists(index))
233     {
234         return
    ↪ Integer<TLink>.Zero;
235     }
236     if (_equalityComparer.Equal
    ↪ s(value,
    ↪ Constants.Any))
237     {
238         return
    ↪ Integer<TLink>.One;
239     }
240     var storedLinkValue =
    ↪ GetLinkUnsafe(index);
241     if (_equalityComparer.Equal
    ↪ s(Link.GetSource(stored
    ↪ LinkValue), value)
    ↪ ||
    ↪ _equalityComparer.Equal
    ↪ s(Link.GetTarget(st
    ↪ oredLinkValue),
    ↪ value))
242     {
243         return
    ↪ Integer<TLink>.One;
244     }
245     return Integer<TLink>.Zero;
246 }
247 }
248 if (restrictions.Count == 3)
249 {
250     var index = restrictions[Consta
    ↪ nts.IndexPart];
251     var source = restrictions[Const
    ↪ ants.SourcePart];
252     var target = restrictions[Const
    ↪ ants.TargetPart];
253     if (_equalityComparer.Equals(in
    ↪ dex,
    ↪ Constants.Any))
254     {
255         if (_equalityComparer.Equal
    ↪ s(source,
    ↪ Constants.Any) &&
    ↪ _equalityComparer.Equal
    ↪ s(target,
    ↪ Constants.Any))
256         {
257             return Total;
258         }
259         else if (_equalityComparer.
    ↪ Equals(source,
    ↪ Constants.Any))
260         {
261             return _targetsTreeMeth
    ↪ ods.CalculateRefere
    ↪ nces(target);
262         }
263         else if (_equalityComparer.
    ↪ Equals(target,
    ↪ Constants.Any))
264         {
265             return _sourcesTreeMeth
    ↪ ods.CalculateRefere
    ↪ nces(source);
266         }
267         else //if(source != Any &&
    ↪ target != Any)
268         {
269             return Total;
270         }
271     }
272 }

```

```

272 {
273     // Эквивалент
274     ↪ Exists(source,
275     ↪ target) =>
276     ↪ Count(Any, source,
277     ↪ target) > 0
278     var link =
279     ↪ _sourcesTreeMethods
280     ↪ .Search(source,
281     ↪ target);
282     return _equalityCompare
283     ↪ r.Equals(link,
284     ↪ Constants.Null) ?
285     ↪ Integer<TLink>.Zero
286     ↪ :
287     ↪ Integer<TLink>.One;
288 }
289 else
290 {
291     if (!Exists(index))
292     {
293         return
294         ↪ Integer<TLink>.Zero;
295     }
296     if (_equalityComparer.Equal
297     ↪ s(source,
298     ↪ Constants.Any) &&
299     ↪ _equalityComparer.Equal
300     ↪ s(target,
301     ↪ Constants.Any))
302     {
303         return
304         ↪ Integer<TLink>.One;
305     }
306     var storedLinkValue =
307     ↪ GetLinkUnsafe(index);
308     if (!_equalityComparer.Equa
309     ↪ ls(source,
310     ↪ Constants.Any) &&
311     ↪ !_equalityComparer.Equa
312     ↪ ls(target,
313     ↪ Constants.Any))
314     {
315         if (_equalityComparer.E
316         ↪ quals(Link.GetSourc
317         ↪ e(storedLinkValue),
318         ↪ source) &&
319         ↪ _equalityComparer.E
320         ↪ quals(Link.GetT
321         ↪ arget(storedLin
322         ↪ kValue),
323         ↪ target))
324         {
325             return Integer<TLin
326             ↪ k>.One;
327         }
328         return
329         ↪ Integer<TLink>.Zero;
330     }
331     var value = default(TLink);
332     if (_equalityComparer.Equal
333     ↪ s(source,
334     ↪ Constants.Any))
335     {
336         value = target;
337     }
338     if (_equalityComparer.Equal
339     ↪ s(target,
340     ↪ Constants.Any))
341     {
342         value = source;
343     }
344     if (_equalityComparer.Equal
345     ↪ s(Link.GetSource(stored
346     ↪ LinkValue), value)
347     ↪ ||
348     ↪ _equalityComparer.Equal
349     ↪ s(Link.GetTarget(st
350     ↪ oredLinkValue),
351     ↪ value))
352     {
353         return
354         ↪ Integer<TLink>.One;
355     }
356 }
357
358 return Integer<TLink>.Zero;
359 }
360
361 throw new
362 ↪ NotSupportedException("Другие
363 ↪ размеры и способы ограничений
364 ↪ не поддерживаются.");
365 }
366
367 [MethodImpl(MethodImplOptions.Aggressive
368 ↪ eInlining)]
369 public TLink Each(Func<IList<TLink>,
370 ↪ TLink> handler, IList<TLink>
371 ↪ restrictions)
372 {
373     if (restrictions.Count == 0)
374     {
375         for (TLink link =
376         ↪ Integer<TLink>.One;
377         ↪ _comparer.Compare(link, (In
378         ↪ teger<TLink>)LinksHeader.Ge
379         ↪ tAllocatedLinks(_header))
380         ↪ <= 0; link =
381         ↪ Increment(link))
382         {
383             if (Exists(link) &&
384             ↪ _equalityComparer.Equal
385             ↪ s(handler(GetLinkStruct
386             ↪ (link)),
387             ↪ Constants.Break))
388             {
389                 return Constants.Break;
390             }
391             return Constants.Continue;
392         }
393     }
394     if (restrictions.Count == 1)
395     {
396         var index = restrictions[Consta
397         ↪ nts.IndexPart];
398         if (_equalityComparer.Equals(in
399         ↪ dex,
400         ↪ Constants.Any))
401         {
402             return Each(handler,
403             ↪ ArrayPool<TLink>.Empty);
404         }
405         if (!Exists(index))
406         {
407             return Constants.Continue;
408         }
409         return handler(GetLinkStruct(in
410         ↪ dex));
411     }
412     if (restrictions.Count == 2)
413     {
414         var index = restrictions[Consta
415         ↪ nts.IndexPart];
416         var value = restrictions[1];
417         if (_equalityComparer.Equals(in
418         ↪ dex,
419         ↪ Constants.Any))
420         {
421             if (_equalityComparer.Equal
422             ↪ s(value,
423             ↪ Constants.Any))
424             {
425                 return Each(handler,
426                 ↪ ArrayPool<TLink>.Em
427                 ↪ pty);
428             }
429             if (_equalityComparer.Equal
430             ↪ s(Each(handler, new[] {
431             ↪ index, value,
432             ↪ Constants.Any })),
433             ↪ Constants.Break))
434             {
435                 return Constants.Break;
436             }
437             return Each(handler, new[]
438             ↪ { index, Constants.Any,
439             ↪ value });
440         }
441     }

```

```

362     else
363     {
364         if (!Exists(index))
365         {
366             return
367                 ↪ Constants.Continue;
368         }
369         if (_equalityComparer.Equal
370             ↪ s(value,
371             ↪ Constants.Any))
372         {
373             return handler(GetLinkS
374                 ↪ truct(index));
375         }
376         var storedLinkValue =
377             ↪ GetLinkUnsafe(index);
378         if (_equalityComparer.Equal
379             ↪ s(Link.GetSource(stored
380             ↪ LinkValue), value)
381             ↪ ||
382             ↪ _equalityComparer.Equal
383             ↪ s(Link.GetTarget(st
384             ↪ oredLinkValue),
385             ↪ value))
386         {
387             return handler(GetLinkS
388                 ↪ truct(index));
389         }
390         return Constants.Continue;
391     }
392 }
393 if (restrictions.Count == 3)
394 {
395     var index = restrictions[Consta
396         ↪ nts.IndexPart];
397     var source = restrictions[Const
398         ↪ ants.SourcePart];
399     var target = restrictions[Const
400         ↪ ants.TargetPart];
401     if (_equalityComparer.Equals(in
402         ↪ dex,
403         ↪ Constants.Any))
404     {
405         if (_equalityComparer.Equal
406             ↪ s(source,
407             ↪ Constants.Any) &&
408             ↪ _equalityComparer.Equal
409             ↪ s(target,
410             ↪ Constants.Any))
411         {
412             return Each(handler,
413                 ↪ ArrayPool<TLink>.Em
414                 ↪ pty);
415         }
416         else if (_equalityComparer.
417             ↪ Equals(source,
418             ↪ Constants.Any))
419         {
420             return _targetsTreeMeth
421                 ↪ ods.EachReference(t
422                 ↪ arget,
423                 ↪ handler);
424         }
425         else if (_equalityComparer.
426             ↪ Equals(target,
427             ↪ Constants.Any))
428         {
429             return _sourcesTreeMeth
430                 ↪ ods.EachReference(s
431                 ↪ ource,
432                 ↪ handler);
433         }
434         else //if(source != Any &&
435             ↪ target != Any)
436         {
437             var link =
438                 ↪ _sourcesTreeMethods
439                 ↪ .Search(source,
440                 ↪ target);
441         }
442     }
443     return _equalityCompare
444         ↪ r.Equals(link,
445         ↪ Constants.Null) ?
446         ↪ Constants.Continue
447         ↪ : handler(GetLinkSt
448             ↪ ruct(link));
449 }
450 }
451 else
452 {
453     if (!Exists(index))
454     {
455         return
456             ↪ Constants.Continue;
457     }
458     if (_equalityComparer.Equal
459         ↪ s(source,
460         ↪ Constants.Any) &&
461         ↪ _equalityComparer.Equal
462         ↪ s(target,
463         ↪ Constants.Any))
464     {
465         return handler(GetLinkS
466             ↪ truct(index));
467     }
468     var storedLinkValue =
469         ↪ GetLinkUnsafe(index);
470     if (!_equalityComparer.Equa
471         ↪ ls(source,
472         ↪ Constants.Any) &&
473         ↪ !_equalityComparer.Equa
474         ↪ ls(target,
475         ↪ Constants.Any))
476     {
477         if (_equalityComparer.E
478             ↪ quals(Link.GetSourc
479             ↪ e(storedLinkValue),
480             ↪ source) &&
481             ↪ _equalityComparer.E
482             ↪ quals(Link.GetT
483             ↪ arget(storedLin
484             ↪ kValue),
485             ↪ target))
486         {
487             return
488                 ↪ handler(GetLink
489                 ↪ Struct(index));
490         }
491         return
492             ↪ Constants.Continue;
493     }
494     var value = default(TLink);
495     if (_equalityComparer.Equal
496         ↪ s(source,
497         ↪ Constants.Any))
498     {
499         value = target;
500     }
501     if (_equalityComparer.Equal
502         ↪ s(target,
503         ↪ Constants.Any))
504     {
505         value = source;
506     }
507     if (_equalityComparer.Equal
508         ↪ s(Link.GetSource(stored
509         ↪ LinkValue), value)
510         ↪ ||
511         ↪ _equalityComparer.Equal
512         ↪ s(Link.GetTarget(st
513         ↪ oredLinkValue),
514         ↪ value))
515     {
516         return handler(GetLinkS
517             ↪ truct(index));
518     }
519     return Constants.Continue;
520 }
521 }
522 throw new
523     ↪ NotSupportedException("Другие
524     ↪ размеры и способы ограничений
525     ↪ не поддерживаются.");
526 }

```



```

445     /// <remarks>
446     /// TODO: Возможно можно перемещать
447     /// значения, если указан индекс, но
448     /// значение существует в другом месте
449     /// (но не в менеджере памяти, а в
450     /// логике Links)
451     /// </remarks>
452     [MethodImpl(MethodImplOptions.AggressiveInlining)]
453     public TLink Update(IList<TLink> values)
454     {
455         var linkIndex =
456             values[Constants.IndexPart];
457         var link = GetLinkUnsafe(linkIndex);
458         // Будет корректно работать только
459         // в том случае, если пространство
460         // выделенной связи предварительно
461         // заполнено нулями
462         if (!_equalityComparer.Equals(Link.
463             GetSource(link),
464             Constants.Null))
465         {
466             _sourcesTreeMethods.Detach(Link.
467                 sHeader.GetFirstAsSourcePoi
468                 nter(_header),
469                 linkIndex);
470         }
471         if (!_equalityComparer.Equals(Link.
472             GetTarget(link),
473             Constants.Null))
474         {
475             _targetsTreeMethods.Detach(Link.
476                 sHeader.GetFirstAsTargetPoi
477                 nter(_header),
478                 linkIndex);
479         }
480         Link.SetSource(link,
481             values[Constants.SourcePart]);
482         Link.SetTarget(link,
483             values[Constants.TargetPart]);
484         if (!_equalityComparer.Equals(Link.
485             GetSource(link),
486             Constants.Null))
487         {
488             _sourcesTreeMethods.Attach(Link.
489                 sHeader.GetFirstAsSourcePoi
490                 nter(_header),
491                 linkIndex);
492         }
493         if (!_equalityComparer.Equals(Link.
494             GetTarget(link),
495             Constants.Null))
496         {
497             _targetsTreeMethods.Attach(Link.
498                 sHeader.GetFirstAsTargetPoi
499                 nter(_header),
500                 linkIndex);
501         }
502         return linkIndex;
503     }
504     [MethodImpl(MethodImplOptions.AggressiveInlining)]
505     public Link<TLink> GetLinkStruct(TLink
506         linkIndex)
507     {
508         var link = GetLinkUnsafe(linkIndex);
509         return new Link<TLink>(linkIndex,
510             link.GetSource(link),
511             link.GetTarget(link));
512     }
513     [MethodImpl(MethodImplOptions.AggressiveInlining)]
514     private IntPtr GetLinkUnsafe(TLink
515         linkIndex) =>
516         _links.GetElement(LinkSizeInBytes,
517         linkIndex);
518     /// <remarks>
519     /// TODO: Возможно нужно будет
520     /// заполнение нулями, если внешнее API
521     /// ими не заполняет пространство
522
523     /// </remarks>
524     public TLink Create()
525     {
526         var freeLink = LinksHeader.GetFirst
527             FreeLink(_header);
528         if (!_equalityComparer.Equals(freeL
529             ink,
530             Constants.Null))
531         {
532             _unusedLinksListMethods.Detach(
533                 freeLink);
534         }
535         else
536         {
537             if (_comparer.Compare(LinksHead
538                 er.GetAllocatedLinks(_heade
539                 r),
540                 Constants.MaxPossibleIndex)
541                 > 0)
542             {
543                 throw new LinksLimitReached
544                     Exception((Integer<TLin
545                     k>)Constants.MaxPossibl
546                     eIndex);
547             }
548             if (_comparer.Compare(LinksHead
549                 er.GetAllocatedLinks(_heade
550                 r),
551                 Decrement(LinksHeader.GetRe
552                 servedLinks(_header))) >=
553                 0)
554             {
555                 _memory.ReservedCapacity +=
556                     _memory.ReservationStep;
557                 SetPointers(_memory);
558                 LinksHeader.SetReservedLink
559                     s(_header,
560                     (Integer<TLink>)(_memor
561                     y.ReservedCapacity /
562                     LinkSizeInBytes));
563             }
564             LinksHeader.SetAllocatedLinks(_
565                 header,
566                 Increment(LinksHeader.GetAl
567                 locatedLinks(_header)));
568             _memory.UsedCapacity +=
569                 LinkSizeInBytes;
570             freeLink = LinksHeader.GetAlloc
571                 atedLinks(_header);
572         }
573         return freeLink;
574     }
575     public void Delete(TLink link)
576     {
577         if (_comparer.Compare(link,
578             LinksHeader.GetAllocatedLinks(_
579             header)) <
580             0)
581         {
582             _unusedLinksListMethods.AttachA
583                 sFirst(link);
584         }
585         else if
586             (_equalityComparer.Equals(link,
587             LinksHeader.GetAllocatedLinks(_
588             header)))
589         {
590             LinksHeader.SetAllocatedLinks(_
591                 header,
592                 Decrement(LinksHeader.GetAl
593                 locatedLinks(_header)));
594             _memory.UsedCapacity -=
595                 LinkSizeInBytes;
596             // Убираем все связи,
597             // находящиеся в списке
598             // свободных в конце файла, до
599             // тех пор, пока не дойдём до
600             // первой существующей связи
601             // Позволяет оптимизировать
602             // количество выделенных
603             // связей (AllocatedLinks)

```

```

527         while ((_comparer.Compare(LinksHeader.GetAllocatedLinks(_header),
528             Integer<TLink>.Zero) > 0)
529             && IsUnusedLink(LinksHeader.GetAllocatedLinks(_header)))
530         {
531             _unusedLinksListMethods.Decrement(LinksHeader.GetAllocatedLinks(_header));
532             LinksHeader.SetAllocatedLinks(_header, Decrement(LinksHeader.GetAllocatedLinks(_header)));
533             _memory.UsedCapacity -= LinkSizeInBytes;
534         }
535     }
536     /// <remarks>
537     /// TODO: Возможно это должно быть событием, вызываемым из IMemory, в том случае, если адрес реально поменялся
538     ///
539     /// Указатель this.links может быть в том же месте,
540     /// так как 0-я связь не используется и имеет такой же размер как Header,
541     /// поэтому header размещается в том же месте, что и 0-я связь
542     /// </remarks>
543     private void SetPointers(IDirectMemory memory)
544     {
545         if (memory == null)
546         {
547             _links = IntPtr.Zero;
548             _header = _links;
549             _unusedLinksListMethods = null;
550             _targetsTreeMethods = null;
551             _unusedLinksListMethods = null;
552         }
553         else
554         {
555             _links = memory.Pointer;
556             _header = _links;
557             _sourcesTreeMethods = new LinksSourcesTreeMethods(this);
558             _targetsTreeMethods = new LinksTargetsTreeMethods(this);
559             _unusedLinksListMethods = new UnusedLinksListMethods(_links, _header);
560         }
561     }
562     [MethodImpl(MethodImplOptions.AggressiveInlining)]
563     private bool Exists(TLink link)
564     => (_comparer.Compare(link, Constants.MinPossibleIndex) >= 0)
565     && (_comparer.Compare(link, LinksHeader.GetAllocatedLinks(_header)) <= 0)
566     && !IsUnusedLink(link);
567     [MethodImpl(MethodImplOptions.AggressiveInlining)]
568     private bool IsUnusedLink(TLink link)
569     => _equalityComparer.Equals(LinksHeader.GetFirstFreeLink(_header), link)
570     || _equalityComparer.Equals(Link.GetSizeAsSource(GetLinkUnsafe(link)), Constants.Null)
571     && !_equalityComparer.Equals(Link.GetSource(GetLinkUnsafe(link)), Constants.Null));
572     #region DisposableBase
573     protected override bool
574     AllowMultipleDisposeCalls => true;
575     protected override void
576     DisposeCore(bool manual, bool wasDisposed)
577     {
578         if (!wasDisposed)
579         {
580             SetPointers(null);
581         }
582         Disposable.TryDispose(_memory);
583     }
584     #endregion
585 }
586 }
587 }
588 }
589 }
590 }
591 }
592 }
593 }
594 }
595 }
596 }
597 }
598 }
599 }
600 }
601 }
602 }
603 }
604 }
605 }
606 }
607 }
608 }
609 }
610 }
611 }
612 }
613 }
614 }
615 }
616 }
617 }
618 }
619 }
620 }
621 }
622 }
623 }
624 }
625 }
626 }
627 }
628 }
629 }
630 }
631 }
632 }
633 }
634 }
635 }
636 }
637 }
638 }
639 }
640 }
641 }
642 }
643 }
644 }
645 }
646 }
647 }
648 }
649 }
650 }
651 }
652 }
653 }
654 }
655 }
656 }
657 }
658 }
659 }
660 }
661 }
662 }
663 }
664 }
665 }
666 }
667 }
668 }
669 }
670 }
671 }
672 }
673 }
674 }
675 }
676 }
677 }
678 }
679 }
680 }
681 }
682 }
683 }
684 }
685 }
686 }
687 }
688 }
689 }
690 }
691 }
692 }
693 }
694 }
695 }
696 }
697 }
698 }
699 }
700 }
701 }
702 }
703 }
704 }
705 }
706 }
707 }
708 }
709 }
710 }
711 }
712 }
713 }
714 }
715 }
716 }
717 }
718 }
719 }
720 }
721 }
722 }
723 }
724 }
725 }
726 }
727 }
728 }
729 }
730 }
731 }
732 }
733 }
734 }
735 }
736 }
737 }
738 }
739 }
740 }
741 }
742 }
743 }
744 }
745 }
746 }
747 }
748 }
749 }
750 }
751 }
752 }
753 }
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 }
782 }
783 }
784 }
785 }
786 }
787 }
788 }
789 }
790 }
791 }
792 }
793 }
794 }
795 }
796 }
797 }
798 }
799 }
800 }
801 }
802 }
803 }
804 }
805 }
806 }
807 }
808 }
809 }
810 }
811 }
812 }
813 }
814 }
815 }
816 }
817 }
818 }
819 }
820 }
821 }
822 }
823 }
824 }
825 }
826 }
827 }
828 }
829 }
830 }
831 }
832 }
833 }
834 }
835 }
836 }
837 }
838 }
839 }
840 }
841 }
842 }
843 }
844 }
845 }
846 }
847 }
848 }
849 }
850 }
851 }
852 }
853 }
854 }
855 }
856 }
857 }
858 }
859 }
860 }
861 }
862 }
863 }
864 }
865 }
866 }
867 }
868 }
869 }
870 }
871 }
872 }
873 }
874 }
875 }
876 }
877 }
878 }
879 }
880 }
881 }
882 }
883 }
884 }
885 }
886 }
887 }
888 }
889 }
890 }
891 }
892 }
893 }
894 }
895 }
896 }
897 }
898 }
899 }
900 }
901 }
902 }
903 }
904 }
905 }
906 }
907 }
908 }
909 }
910 }
911 }
912 }
913 }
914 }
915 }
916 }
917 }
918 }
919 }
920 }
921 }
922 }
923 }
924 }
925 }
926 }
927 }
928 }
929 }
930 }
931 }
932 }
933 }
934 }
935 }
936 }
937 }
938 }
939 }
940 }
941 }
942 }
943 }
944 }
945 }
946 }
947 }
948 }
949 }
950 }
951 }
952 }
953 }
954 }
955 }
956 }
957 }
958 }
959 }
960 }
961 }
962 }
963 }
964 }
965 }
966 }
967 }
968 }
969 }
970 }
971 }
972 }
973 }
974 }
975 }
976 }
977 }
978 }
979 }
980 }
981 }
982 }
983 }
984 }
985 }
986 }
987 }
988 }
989 }
990 }
991 }
992 }
993 }
994 }
995 }
996 }
997 }
998 }
999 }

```

```

34         protected override void
           SetPrevious(TLink element,
           ↪ TLink previous) => (_links.GetElement(LinkSizeInBytes,
           ↪ element) + Link.SourceOffset).SetValue(previous);
35
36         protected override void
           SetNext(TLink element, TLink
           ↪ next) => (_links.GetElement(LinkSizeInBytes, element) +
           ↪ Link.TargetOffset).SetValue(next);
37
38         protected override void
           SetSize(TLink size) => (_header
           ↪ + LinksHeader.FreeLinksOffset).SetValue(size);
39     }
40 }
41 }

./ResizableDirectMemory/ResizableDirectMemoryLinks.TreeMethods.cs
1  using System;
2  using System.Text;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5  using Platform.Numbers;
6  using Platform.Unsafe;
7  using Platform.Collections.Methods.Trees;
8  using Platform.Data.Constants;
9
10 namespace Platform.Data.Doublets.ResizableDirectMemory
11 {
12     partial class ResizableDirectMemoryLinks<TLink>
13     {
14         private abstract class LinksTreeMethodsBase : SizedAndThre
           ↪ AddedAVLBalancedTreeMethods<TLink>
15         {
16             private readonly ResizableDirectMemoryLinks<TLink>
           ↪ _memory;
17             private readonly LinksCombinedConstants<TLink,
           ↪ TLink, int> _constants;
18             protected readonly IntPtr Links;
19             protected readonly IntPtr Header;
20
21             protected LinksTreeMethodsBase(ResizableDirectMemoryLinks<TLink>
           ↪ memory)
22             {
23                 Links = memory._links;
24                 Header = memory._header;
25                 _memory = memory;
26                 _constants = memory.Constants;
27             }
28
29             [MethodImpl(MethodImplOptions.AggressiveInlining)]
30             protected abstract TLink GetTreeRoot();
31
32             [MethodImpl(MethodImplOptions.AggressiveInlining)]
33             protected abstract TLink GetBasePartValue(TLink link);
34
35             public TLink this[TLink index]
36             {
37                 get
38                 {
39                     var root = GetTreeRoot();
40                     if (GreaterOrEqualThan(index,
           ↪ x,
           ↪ GetSize(root)))
41                     {
42                         return GetZero();
43                     }
44                     while (!EqualToZero(root))
45                     {
46                         var left = GetLeftOrDefault(root);
47
48                         var leftSize =
49                             ↪ GetSizeOrZero(left);
50                         if (LessThan(index,
51                             ↪ leftSize))
52                         {
53                             root = left;
54                             continue;
55                         }
56                         if (IsEquals(index,
57                             ↪ leftSize))
58                         {
59                             return root;
60                         }
61                         root = GetRightOrDefault(t(root));
62                         index = Subtract(index,
63                             ↪ Increment(leftSize));
64                     }
65                     return GetZero(); // TODO:
66                     ↪ Impossible situation
67                     ↪ exception (only if tree
68                     ↪ structure broken)
69                 }
70             }
71
72             // TODO: Return indices range
73             ↪ instead of references count
74             public TLink CalculateReferences(TLink link)
75             {
76                 var root = GetTreeRoot();
77                 var total = GetSize(root);
78                 var totalRightIgnore = GetZero();
79                 while (!EqualToZero(root))
80                 {
81                     var @base = GetBasePartValue(root);
82                     if (LessOrEqualThan(@base,
83                         ↪ link))
84                     {
85                         root = GetRightOrDefault(t(root));
86                     }
87                     else
88                     {
89                         totalRightIgnore = Add(totalRightIgnore,
90                             ↪ Increment(GetRightSize(root)));
91                         root = GetLeftOrDefault(root);
92                     }
93                 }
94                 root = GetTreeRoot();
95                 var totalLeftIgnore = GetZero();
96                 while (!EqualToZero(root))
97                 {
98                     var @base = GetBasePartValue(root);
99                     if (GreaterOrEqualThan(@base,
100                         ↪ link))
101                     {
102                         root = GetLeftOrDefault(root);
103                     }
104                     else
105                     {
106                         totalLeftIgnore = Add(totalLeftIgnore,
107                             ↪ Increment(GetLeftSize(root)));
108                         root = GetRightOrDefault(t(root));
109                     }
110                 }
111                 return Subtract(Subtract(total,
112                     ↪ totalRightIgnore),
113                     ↪ totalLeftIgnore);
114             }
115         }
116     }

```

```

102 public TLink EachReference(TLink 158 : base(memory)
    ↳ link, Func<TLink>, TLink> 159 {
    ↳ handler) 160 }
103 { 161
104     var root = GetTreeRoot(); 162
105     if (EqualToZero(root))
106     {
107         return _constants.Continue;
108     }
109     TLink first = GetZero(), 163
    ↳ current = root; 164
110     while (!EqualToZero(current))
111     {
112         var @base = GetBasePartValu 165
    ↳ e(current); 166
113         if (GreaterOrEqualThan(@bas 165
    ↳ e, 166
    ↳ link))
114         {
115             if (IsEquals(@base,
116             ↳ link))
117             {
118                 first = current; 167
119             } 168
120             current = GetLeftOrDefa 169
    ↳ ult(current); 170
121         }
122         else
123         {
124             current = GetRightOrDef 169
    ↳ ault(current); 170
125         }
126     } 171
127     if (!EqualToZero(first)) 172
128     {
129         current = first;
130         while (true)
131         {
132             if (IsEquals(handler(_m 173
    ↳ emory.GetLinkStruct 174
    ↳ (current)), 175
    ↳ _constants.Break)) 176
133             {
134                 return _constants.B 177
    ↳ reak; 178
135             }
136             current =
137             ↳ GetNext(current); 179
138             if (EqualToZero(current 180
    ↳ ) || 181
    ↳ !IsEquals(GetBasePa 182
    ↳ rtValue(current), 183
    ↳ link)) 184
139             {
140                 break; 185
141             } 186
142         } 187
143     } 188
144     return _constants.Continue;
145 }
146
147 protected override void
148 ↳ PrintNodeValue(TLink node,
149 ↳ StringBuilder sb)
150 {
151     sb.Append(' ');
152     sb.Append((Links.GetElement(Lin 183
    ↳ kSizeInBytes, node) + 184
    ↳ Link.SourceOffset).GetValue 185
    ↳ <TLink>()); 186
153     sb.Append('-');
154     sb.Append('>');
155     sb.Append((Links.GetElement(Lin 187
    ↳ kSizeInBytes, node) + 188
    ↳ Link.TargetOffset).GetValue 189
    ↳ <TLink>());
156 }
157
158 private class LinksSourcesTreeMethods : 189
    ↳ LinksTreeMethodsBase
159 {
160     public LinksSourcesTreeMethods(Resi 190
    ↳ zableDirectMemoryLinks<TLink> 191
    ↳ memory)
161     {
162         : base(memory)
163     {
164     }
165
166     protected override IntPtr
167     ↳ GetLeftPointer(TLink node) =>
168     ↳ Links.GetElement(LinkSizeInByte 169
    ↳ s, node) +
170     ↳ Link.LeftAsSourceOffset;
171
172     protected override IntPtr
173     ↳ GetRightPointer(TLink node) =>
174     ↳ Links.GetElement(LinkSizeInByte 175
    ↳ s, node) +
176     ↳ Link.RightAsSourceOffset;
177
178     protected override TLink
179     ↳ GetLeftValue(TLink node) => (Li 180
    ↳ nks.GetElement(LinkSizeInBytes, 181
    ↳ node) + Link.LeftAsSourceOffset 182
    ↳ ).GetValue<TLink>());
183
184     protected override TLink
185     ↳ GetRightValue(TLink node) =>
186     ↳ (Links.GetElement(LinkSizeInByt 187
    ↳ es, node) +
188     ↳ Link.RightAsSourceOffset).GetVa 189
    ↳ lue<TLink>());
190
191     protected override TLink
192     ↳ GetSize(TLink node)
193     {
194         var previousValue = (Links.GetE 195
    ↳ lement(LinkSizeInBytes, 196
    ↳ node) + Link.SizeAsSourceOf 197
    ↳ fset).GetValue<TLink>());
198         return BitwiseHelpers.PartialRe 199
    ↳ ad(previousValue, 5, 200
    ↳ -5);
199     }
200
201     protected override void
202     ↳ SetLeft(TLink node, TLink left)
203     ↳ => (Links.GetElement(LinkSizeIn 204
    ↳ Bytes, node) +
205     ↳ Link.LeftAsSourceOffset).SetVal 206
    ↳ ue(left);
207
208     protected override void
209     ↳ SetRight(TLink node, TLink
210     ↳ right) => (Links.GetElement(Lin 211
    ↳ kSizeInBytes, node) +
212     ↳ Link.RightAsSourceOffset).SetVa 213
    ↳ lue(right);
214
215     protected override void
216     ↳ SetSize(TLink node, TLink size)
217     {
218         var previousValue = (Links.GetE 219
    ↳ lement(LinkSizeInBytes, 220
    ↳ node) + Link.SizeAsSourceOf 221
    ↳ fset).GetValue<TLink>());
222         (Links.GetElement(LinkSizeInByt 223
    ↳ es, node) +
224     ↳ Link.SizeAsSourceOffset).Se 225
    ↳ tValue(BitwiseHelpers.Parti 226
    ↳ alWrite(previousValue, 227
    ↳ size, 5, -5));
228     }
229
230     protected override bool
231     ↳ GetLeftIsChild(TLink node)
232     {
233         var previousValue = (Links.GetE 234
    ↳ lement(LinkSizeInBytes, 235
    ↳ node) + Link.SizeAsSourceOf 236
    ↳ fset).GetValue<TLink>());
237         return (Integer<TLink>)BitwiseH 238
    ↳ elpers.PartialRead(previous 239
    ↳ Value, 4, 240
    ↳ 1);
241     }

```

```

192     protected override void                224         var modified = BitwiseHelpers.P
    ↪ SetLeftIsChild(TLink node, bool
    ↪ value)
193     {
194         var previousValue = (Links.GetE
    ↪ lement(LinkSizeInBytes,
    ↪ node) + Link.SizeAsSourceOf
    ↪ fset).GetValue<TLink>();
195         var modified = BitwiseHelpers.P
    ↪ artialWrite(previousValue,
    ↪ (TLink)(Integer<TLink>)valu
    ↪ e, 4,
    ↪ 1);
196         (Links.GetElement(LinkSizeInByt
    ↪ es, node) +
    ↪ Link.SizeAsSourceOffset).Se
    ↪ tValue(modified);
197     }
198
199     protected override bool
    ↪ GetRightIsChild(TLink node)
200     {
201         var previousValue = (Links.GetE
    ↪ lement(LinkSizeInBytes,
    ↪ node) + Link.SizeAsSourceOf
    ↪ fset).GetValue<TLink>();
202         return (Integer<TLink>)BitwiseH
    ↪ elpers.PartialRead(previous
    ↪ Value, 3,
    ↪ 1);
203     }
204
205     protected override void
    ↪ SetRightIsChild(TLink node,
    ↪ bool value)
206     {
207         var previousValue = (Links.GetE
    ↪ lement(LinkSizeInBytes,
    ↪ node) + Link.SizeAsSourceOf
    ↪ fset).GetValue<TLink>();
208         var modified = BitwiseHelpers.P
    ↪ artialWrite(previousValue,
    ↪ (TLink)(Integer<TLink>)valu
    ↪ e, 3,
    ↪ 1);
209         (Links.GetElement(LinkSizeInByt
    ↪ es, node) +
    ↪ Link.SizeAsSourceOffset).Se
    ↪ tValue(modified);
210     }
211
212     protected override sbyte
    ↪ GetBalance(TLink node)
213     {
214         var previousValue = (Links.GetE
    ↪ lement(LinkSizeInBytes,
    ↪ node) + Link.SizeAsSourceOf
    ↪ fset).GetValue<TLink>();
215         var value = (ulong)(Integer<TLi
    ↪ nk>)BitwiseHelpers.PartialR
    ↪ ead(previousValue, 0,
    ↪ 3);
216         var unpackedValue =
    ↪ (sbyte)((value & 4) > 0 ?
    ↪ ((value & 4) << 5) | value
    ↪ & 3 | 124 : value & 3);
217         return unpackedValue;
218     }
219
220     protected override void
    ↪ SetBalance(TLink node, sbyte
    ↪ value)
221     {
222         var previousValue = (Links.GetE
    ↪ lement(LinkSizeInBytes,
    ↪ node) + Link.SizeAsSourceOf
    ↪ fset).GetValue<TLink>();
223         var packagedValue = (TLink)(Int
    ↪ eger<TLink>)(((byte)value
    ↪ >> 5) & 4) | value & 3);
224         var modified = BitwiseHelpers.P
    ↪ artialWrite(previousValue,
    ↪ packagedValue, 0, 3);
    ↪ (Links.GetElement(LinkSizeInByt
    ↪ es, node) +
    ↪ Link.SizeAsSourceOffset).Se
    ↪ tValue(modified);
225     }
226
227     protected override bool
    ↪ FirstIsToTheLeftOfSecond(TLink
    ↪ first, TLink second)
228     {
229         var firstSource = (Links.GetEle
    ↪ ment(LinkSizeInBytes,
    ↪ first) + Link.SourceOffset)
    ↪ .GetValue<TLink>();
230         var secondSource = (Links.GetEl
    ↪ ement(LinkSizeInBytes,
    ↪ second) + Link.SourceOffset
    ↪ ).GetValue<TLink>();
231         return LessThan(firstSource,
    ↪ secondSource) ||
    ↪ (IsEquals(firstSource,
    ↪ secondSource) && Les
    ↪ sThan((Links.GetElem
    ↪ ent(LinkSizeInBytes,
    ↪ first) +
    ↪ Link.TargetOffset).G
    ↪ etValue<TLink>(),
    ↪ (Links.GetElement(Li
    ↪ nkSizeInBytes,
    ↪ second) +
    ↪ Link.TargetOffset).G
    ↪ etValue<TLink>()));
232     }
233
234     protected override bool
    ↪ FirstIsToTheRightOfSecond(TLink
    ↪ first, TLink second)
235     {
236         var firstSource = (Links.GetEle
    ↪ ment(LinkSizeInBytes,
    ↪ first) + Link.SourceOffset)
    ↪ .GetValue<TLink>();
237         var secondSource = (Links.GetEl
    ↪ ement(LinkSizeInBytes,
    ↪ second) + Link.SourceOffset
    ↪ ).GetValue<TLink>();
238         return GreaterThan(firstSource,
    ↪ secondSource) ||
    ↪ (IsEquals(firstSource,
    ↪ secondSource) &&
    ↪ GreaterThan((Links.G
    ↪ etElement(LinkSizeIn
    ↪ Bytes, first) +
    ↪ Link.TargetOffset).G
    ↪ etValue<TLink>(),
    ↪ (Links.GetElement(Li
    ↪ nkSizeInBytes,
    ↪ second) +
    ↪ Link.TargetOffset).G
    ↪ etValue<TLink>()));
239     }
240
241     protected override TLink
    ↪ GetTreeRoot() => (Header +
    ↪ LinksHeader.FirstAsSourceOffset
    ↪ ).GetValue<TLink>();
242
243     protected override TLink
    ↪ GetBasePartValue(TLink link) =>
    ↪ (Links.GetElement(LinkSizeInByt
    ↪ es, link) +
    ↪ Link.SourceOffset).GetValue<TLi
    ↪ nk>();
244
245     /// <summary>
246     /// Выполняет поиск и возвращает
    ↪ индекс связи с указанными
    ↪ Source (началом) и Target
    ↪ (концом)

```

```

250     /// по дереву (индексу) связей,
251     ↪ отсортированному по Source, а
252     ↪ затем по Target.
253     /// </summary>
254     /// <param name="source">Индекс
255     ↪ связи, которая является началом
256     ↪ на искомой связи.</param>
257     /// <param name="target">Индекс
258     ↪ связи, которая является концом
259     ↪ на искомой связи.</param>
260     /// <returns>Индекс искомой
261     ↪ связи.</returns>
262     public TLink Search(TLink source,
263     ↪ TLink target)
264     {
265         var root = GetTreeRoot();
266         while (!EqualToZero(root))
267         {
268             var rootSource =
269                 (Links.GetElement(LinkS
270                 ↪ izeInBytes, root) +
271                 ↪ Link.SourceOffset).GetV
272                 ↪ alue<TLink>();
273             var rootTarget =
274                 (Links.GetElement(LinkS
275                 ↪ izeInBytes, root) +
276                 ↪ Link.TargetOffset).GetV
277                 ↪ alue<TLink>();
278             if (FirstIsToTheLeftOfSecon
279             ↪ d(source, target,
280             ↪ rootSource,
281             ↪ rootTarget)) //
282             ↪ node.Key < root.Key
283             {
284                 root = GetLeftOrDefault
285                 ↪ (root);
286             }
287             else if (FirstIsToTheRightO
288             ↪ fSecond(source, target,
289             ↪ rootSource,
290             ↪ rootTarget)) //
291             ↪ node.Key > root.Key
292             {
293                 root = GetRightOrDefaul
294                 ↪ t(root);
295             }
296             else // node.Key == root.Key
297             {
298                 return root;
299             }
300         }
301         return GetZero();
302     }
303
304     [MethodImpl(MethodImplOptions.AggressiveInlining)]
305     private bool
306     FirstIsToTheLeftOfSecond(TLink
307     ↪ firstSource, TLink firstTarget,
308     ↪ TLink secondSource, TLink
309     ↪ secondTarget) =>
310     ↪ LessThan(firstSource,
311     ↪ secondSource) ||
312     ↪ (IsEquals(firstSource,
313     ↪ secondSource) &&
314     ↪ LessThan(firstTarget,
315     ↪ secondTarget));
316
317     [MethodImpl(MethodImplOptions.AggressiveInlining)]
318     private bool
319     FirstIsToTheRightOfSecond(TLink
320     ↪ firstSource, TLink firstTarget,
321     ↪ TLink secondSource, TLink
322     ↪ secondTarget) =>
323     ↪ GreaterThan(firstSource,
324     ↪ secondSource) ||
325     ↪ (IsEquals(firstSource,
326     ↪ secondSource) &&
327     ↪ GreaterThan(firstTarget,
328     ↪ secondTarget));
329
330     private class LinksTargetsTreeMethods :
331     ↪ LinksTreeMethodsBase
332     {
333         public LinksTargetsTreeMethods(Resi
334         ↪ zableDirectMemoryLinks<TLink>
335         ↪ memory)
336         : base(memory)
337         {
338         }
339
340         protected override IntPtr
341         ↪ GetLeftPointer(TLink node) =>
342         ↪ Links.GetElement(LinkSizeInByte
343         ↪ s, node) +
344         ↪ Link.LeftAsTargetOffset;
345
346         protected override IntPtr
347         ↪ GetRightPointer(TLink node) =>
348         ↪ Links.GetElement(LinkSizeInByte
349         ↪ s, node) +
350         ↪ Link.RightAsTargetOffset;
351
352         protected override TLink
353         ↪ GetLeftValue(TLink node) => (Li
354         ↪ nks.GetElement(LinkSizeInBytes,
355         ↪ node) + Link.LeftAsTargetOffset
356         ↪ ).GetValue<TLink>();
357
358         protected override TLink
359         ↪ GetRightValue(TLink node) =>
360         ↪ (Links.GetElement(LinkSizeInByt
361         ↪ es, node) +
362         ↪ Link.RightAsTargetOffset).GetVa
363         ↪ lue<TLink>();
364
365         protected override TLink
366         ↪ GetSize(TLink node)
367         {
368             var previousValue = (Links.GetE
369             ↪ lement(LinkSizeInBytes,
370             ↪ node) + Link.SizeAsTargetOf
371             ↪ fset).GetValue<TLink>();
372             return BitwiseHelpers.PartialRe
373             ↪ ad(previousValue, 5,
374             ↪ -5);
375         }
376
377         protected override void
378         ↪ SetLeft(TLink node, TLink left)
379         ↪ => (Links.GetElement(LinkSizeIn
380         ↪ Bytes, node) +
381         ↪ Link.LeftAsTargetOffset).SetVal
382         ↪ ue(left);
383
384         protected override void
385         ↪ SetRight(TLink node, TLink
386         ↪ right) => (Links.GetElement(Lin
387         ↪ kSizeInBytes, node) +
388         ↪ Link.RightAsTargetOffset).SetVa
389         ↪ lue(right);
390
391         protected override void
392         ↪ SetSize(TLink node, TLink size)
393         {
394             var previousValue = (Links.GetE
395             ↪ lement(LinkSizeInBytes,
396             ↪ node) + Link.SizeAsTargetOf
397             ↪ fset).GetValue<TLink>();
398             (Links.GetElement(LinkSizeInByt
399             ↪ es, node) +
400             ↪ Link.SizeAsTargetOffset).Se
401             ↪ tValue(BitwiseHelpers.Parti
402             ↪ alWrite(previousValue,
403             ↪ size, 5, -5));
404         }
405
406         protected override bool
407         ↪ GetLeftIsChild(TLink node)
408         {
409             var previousValue = (Links.GetE
410             ↪ lement(LinkSizeInBytes,
411             ↪ node) + Link.SizeAsTargetOf
412             ↪ fset).GetValue<TLink>();

```

```

319         return (Integer<TLink>)BitwiseH
320             ↪ elpers.PartialRead(previous
321             ↪ Value, 4,
322             ↪ 1);
323     }
324     protected override void
325     ↪ SetLeftIsChild(TLink node, bool
326     ↪ value)
327     {
328         var previousValue = (Links.GetE
329         ↪ lement(LinkSizeInBytes,
330         ↪ node) + Link.SizeAsTargetOf
331         ↪ fset).GetValue<TLink>();
332         var modified = BitwiseHelpers.P
333         ↪ artialWrite(previousValue,
334         ↪ (TLink)(Integer<TLink>)valu
335         ↪ e, 4,
336         ↪ 1);
337         (Links.GetElement(LinkSizeInByt
338         ↪ es, node) +
339         ↪ Link.SizeAsTargetOffset).Se
340         ↪ tValue(modified);
341     }
342     protected override bool
343     ↪ GetRightIsChild(TLink node)
344     {
345         var previousValue = (Links.GetE
346         ↪ lement(LinkSizeInBytes,
347         ↪ node) + Link.SizeAsTargetOf
348         ↪ fset).GetValue<TLink>();
349         return (Integer<TLink>)BitwiseH
350         ↪ elpers.PartialRead(previous
351         ↪ Value, 3,
352         ↪ 1);
353     }
354     protected override void
355     ↪ SetRightIsChild(TLink node,
356     ↪ bool value)
357     {
358         var previousValue = (Links.GetE
359         ↪ lement(LinkSizeInBytes,
360         ↪ node) + Link.SizeAsTargetOf
361         ↪ fset).GetValue<TLink>();
362         var modified = BitwiseHelpers.P
363         ↪ artialWrite(previousValue,
364         ↪ (TLink)(Integer<TLink>)valu
365         ↪ e, 3,
366         ↪ 1);
367         (Links.GetElement(LinkSizeInByt
368         ↪ es, node) +
369         ↪ Link.SizeAsTargetOffset).Se
370         ↪ tValue(modified);
371     }
372     protected override sbyte
373     ↪ GetBalance(TLink node)
374     {
375         var previousValue = (Links.GetE
376         ↪ lement(LinkSizeInBytes,
377         ↪ node) + Link.SizeAsTargetOf
378         ↪ fset).GetValue<TLink>();
379         var value = (ulong)(Integer<TLi
380         ↪ nk>)BitwiseHelpers.PartialR
381         ↪ ead(previousValue, 0,
382         ↪ 3);
383         var unpackedValue =
384         ↪ ((sbyte)((value & 4) > 0 ?
385         ↪ ((value & 4) << 5) | value
386         ↪ & 3 | 124 : value & 3);
387         return unpackedValue;
388     }
389     protected override void
390     ↪ SetBalance(TLink node, sbyte
391     ↪ value)
392     {
393         var previousValue = (Links.GetE
394         ↪ lement(LinkSizeInBytes,
395         ↪ node) + Link.SizeAsTargetOf
396         ↪ fset).GetValue<TLink>();
397         var modified = BitwiseHelpers.P
398         ↪ artialWrite(previousValue,
399         ↪ (TLink)(Integer<TLink>)valu
400         ↪ e, 4,
401         ↪ 1);
402         (Links.GetElement(LinkSizeInByt
403         ↪ es, node) +
404         ↪ Link.SizeAsTargetOffset).Se
405         ↪ tValue(modified);
406     }
407     protected override bool
408     ↪ FirstIsToTheLeftOfSecond(TLink
409     ↪ first, TLink second)
410     {
411         var firstTarget = (Links.GetEle
412         ↪ ment(LinkSizeInBytes,
413         ↪ first) + Link.TargetOffset)
414         ↪ .GetValue<TLink>();
415         var secondTarget = (Links.GetEl
416         ↪ ement(LinkSizeInBytes,
417         ↪ second) + Link.TargetOffset
418         ↪ ).GetValue<TLink>();
419         return LessThan(firstTarget,
420         ↪ secondTarget) ||
421         ↪ (IsEquals(firstTarget,
422         ↪ secondTarget) && Les
423         ↪ sThan((Links.GetElem
424         ↪ ent(LinkSizeInBytes,
425         ↪ first) +
426         ↪ Link.SourceOffset).G
427         ↪ etValue<TLink>(),
428         ↪ (Links.GetElement(Li
429         ↪ nkSizeInBytes,
430         ↪ second) +
431         ↪ Link.SourceOffset).G
432         ↪ etValue<TLink>()));
433     }
434     protected override bool
435     ↪ FirstIsToTheRightOfSecond(TLink
436     ↪ first, TLink second)
437     {
438         var firstTarget = (Links.GetEle
439         ↪ ment(LinkSizeInBytes,
440         ↪ first) + Link.TargetOffset)
441         ↪ .GetValue<TLink>();
442         var secondTarget = (Links.GetEl
443         ↪ ement(LinkSizeInBytes,
444         ↪ second) + Link.TargetOffset
445         ↪ ).GetValue<TLink>();
446         return GreaterThan(firstTarget,
447         ↪ secondTarget) ||
448         ↪ (IsEquals(firstTarget,
449         ↪ secondTarget) &&
450         ↪ GreaterThan((Links.G
451         ↪ etElement(LinkSizeIn
452         ↪ Bytes, first) +
453         ↪ Link.SourceOffset).G
454         ↪ etValue<TLink>(),
455         ↪ (Links.GetElement(Li
456         ↪ nkSizeInBytes,
457         ↪ second) +
458         ↪ Link.SourceOffset).G
459         ↪ etValue<TLink>()));
460     }
461     protected override TLink
462     ↪ GetTreeRoot() => (Header +
463     ↪ LinksHeader.FirstAsTargetOffset
464     ↪ ).GetValue<TLink>();
465     protected override TLink
466     ↪ GetBasePartValue(TLink link) =>
467     ↪ (Links.GetElement(LinkSizeInByt
468     ↪ es, link) +
469     ↪ Link.TargetOffset).GetValue<TLi
470     ↪ nk>();
471     }
472     }

```



```

379 }

./ResizableDirectMemory/UInt64ResizableDirectMemoryLinks.cs
1 using System;
2 using System.Collections.Generic;
3 using System.Runtime.CompilerServices;
4 using Platform.Disposables;
5 using Platform.Collections.Arrays;
6 using Platform.Helpers.Singletons;
7 using Platform.Memory;
8 using Platform.Data.Exceptions;
9 using Platform.Data.Constants;
10
11 // #define ENABLE_TREE_AUTO_DEBUG_AND_VALIDATION
12
13 #pragma warning disable 0649
14 #pragma warning disable 169
15
16 // ReSharper disable BuiltInTypeReferenceStyle
17
18 namespace
19 {
20     using id = UInt64;
21
22     public unsafe partial class
23     {
24         UInt64ResizableDirectMemoryLinks :
25         DisposableBase, ILinks<id>
26     {
27         /// <summary>Возвращает размер одной
28         /// связи в байтах.</summary>
29         /// <remarks>
30         /// Используется только во вне класса,
31         /// не рекомендуется использовать внутри.
32         /// Так как во вне не обязательно будет
33         /// доступен unsafe C#.
34         /// </remarks>
35         public static readonly int
36         LinkSizeInBytes = sizeof(Link);
37
38         public static readonly long
39         DefaultLinksSizeStep =
40         LinkSizeInBytes * 1024 * 1024;
41
42         private struct Link
43         {
44             public id Source;
45             public id Target;
46             public id LeftAsSource;
47             public id RightAsSource;
48             public id SizeAsSource;
49             public id LeftAsTarget;
50             public id RightAsTarget;
51             public id SizeAsTarget;
52         }
53
54         private struct LinksHeader
55         {
56             public id AllocatedLinks;
57             public id ReservedLinks;
58             public id FreeLinks;
59             public id FirstFreeLink;
60             public id FirstAsSource;
61             public id FirstAsTarget;
62             public id LastFreeLink;
63             public id Reserved8;
64         }
65
66         private readonly long
67         _memoryReservationStep;
68
69         private readonly IResizableDirectMemory
70         _memory;
71         private LinksHeader* _header;
72         private Link* _links;
73
74         private LinksTargetsTreeMethods
75         _targetsTreeMethods;
76         private LinksSourcesTreeMethods
77         _sourcesTreeMethods;
78
79         // TODO: Возможно чтобы гарантированно
80         // проверять на то, является ли связь
81         // удалённой, нужно использовать не
82         // список а дерево, так как так можно
83         // быстрее проверить на наличие связи
84         // внутри
85         private UnusedLinksListMethods
86         _unusedLinksListMethods;
87
88         /// <summary>
89         /// Возвращает общее число связей
90         /// находящихся в хранилище.
91         /// </summary>
92         private id Total =>
93         {
94             _header->AllocatedLinks -
95             _header->FreeLinks;
96
97             // TODO: Дать возможность
98             // переопределять в конструкторе
99             public LinksCombinedConstants<id, id,
100             int> Constants { get; }
101
102             public UInt64ResizableDirectMemoryLinks
103             (string address) : this(address,
104             DefaultLinksSizeStep) { }
105
106             /// <summary>
107             /// Создаёт экземпляр базы данных Links
108             /// в файле по указанному адресу, с
109             /// указанным минимальным шагом
110             /// расширения базы данных.
111             /// </summary>
112             /// <param name="address">Полный путь
113             /// к файлу базы данных.</param>
114             /// <param name="memoryReservationStep">
115             /// Минимальный шаг расширения базы
116             /// данных в байтах.</param>
117             public UInt64ResizableDirectMemoryLinks
118             (string address, long
119             memoryReservationStep) : this(new
120             FileMappedResizableDirectMemory(add
121             ress, memoryReservationStep),
122             memoryReservationStep) { }
123
124             public UInt64ResizableDirectMemoryLinks
125             (IResizableDirectMemory memory) :
126             this(memory, DefaultLinksSizeStep)
127             { }
128
129             public UInt64ResizableDirectMemoryLinks
130             (IResizableDirectMemory memory,
131             long memoryReservationStep)
132             {
133                 Constants = Default<LinksCombinedCo
134                 nstants<id, id,
135                 int>>.Instance;
136                 _memory = memory;
137                 _memoryReservationStep =
138                 memoryReservationStep;
139                 if (memory.ReservedCapacity <
140                 memoryReservationStep)
141                 {
142                     memory.ReservedCapacity =
143                     memoryReservationStep;
144                 }
145                 SetPointers(_memory);
146                 // Гарантия корректности
147                 _memory.UsedCapacity
148                 относительно
149                 _header->AllocatedLinks
150                 _memory.UsedCapacity =
151                 ((long) _header->AllocatedLinks
152                 * sizeof(Link)) +
153                 sizeof(LinksHeader);
154                 // Гарантия корректности
155                 _header->ReservedLinks
156                 относительно
157                 _memory.ReservedCapacity
158                 _header->ReservedLinks =
159                 (id)((_memory.ReservedCapacity
160                 - sizeof(LinksHeader)) /
161                 sizeof(Link));
162             }
163
164             [MethodImpl(MethodImplOptions.Aggressive
165            Inlining)]
166             public id Count(IList<id> restrictions)
167             {
168                 // Если нет ограничений, тогда
169                 // возвращаем общее число связей
170                 // находящихся в хранилище.
171                 if (restrictions.Count == 0)
172                 {
173                     return Total;
174                 }
175                 if (restrictions.Count == 1)
176

```

```

113 {
114     var index = restrictions[Consta
115         ↪ nts.IndexPart];
116     if (index == Constants.Any)
117     {
118         return Total;
119     }
120     return Exists(index) ? 1UL :
121         ↪ 0UL;
122 }
123 if (restrictions.Count == 2)
124 {
125     var index = restrictions[Consta
126         ↪ nts.IndexPart];
127     var value = restrictions[1];
128     if (index == Constants.Any)
129     {
130         if (value == Constants.Any)
131         {
132             return Total; // Any -
133                 ↪ как отсутствие
134                 ↪ ограничения
135         }
136         return
137             ↪ _sourcesTreeMethods.Cal
138             ↪ culateReferences(value)
139             ↪ + _targetsTreeMethods.
140             ↪ CalculateReference
141             ↪ s(value);
142     }
143     else
144     {
145         if (!Exists(index))
146         {
147             return 0;
148         }
149         if (value == Constants.Any)
150         {
151             return 1;
152         }
153         var storedLinkValue =
154             ↪ GetLinkUnsafe(index);
155         if (storedLinkValue->Source
156             ↪ == value ||
157             ↪ storedLinkValue->Target
158             ↪ == value)
159         {
160             return 1;
161         }
162         return 0;
163     }
164 }
165 if (restrictions.Count == 3)
166 {
167     var index = restrictions[Consta
168         ↪ nts.IndexPart];
169     var source = restrictions[Const
170         ↪ ants.SourcePart];
171     var target = restrictions[Const
172         ↪ ants.TargetPart];
173     if (index == Constants.Any)
174     {
175         if (source == Constants.Any
176             ↪ && target ==
177             ↪ Constants.Any)
178         {
179             return Total;
180         }
181         else if (source ==
182             ↪ Constants.Any)
183         {
184             return _targetsTreeMeth
185                 ↪ ods.CalculateRefere
186                 ↪ nces(target);
187         }
188         else if (target ==
189             ↪ Constants.Any)
190         {
191             return _sourcesTreeMeth
192                 ↪ ods.CalculateRefere
193                 ↪ nces(source);
194         }
195         else //if(source != Any &&
196             ↪ target != Any)
197     }
198 }
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
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
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
26
```

230	return	291	if (source == Constants.Any
	↳ Constants.Break;		↳
231	}		↳ Constants.Any)
232	}	292	{
233	}	293	return Each(handler,
234	return Constants.Continue;		↳ ArrayPool<ulong>.Em
235	}		↳ pty);
236	if (restrictions.Count == 1)	294	}
237	{	295	else if (source ==
238	var index = restrictions[Consta		↳ Constants.Any)
	↳ nts.IndexPart];	296	{
239	if (index == Constants.Any)	297	return _targetsTreeMeth
240	{		↳ ods.EachReference(t
241	return Each(handler,		↳ target,
	↳ ArrayPool<ulong>.Empty);		↳ handler);
242	}	298	}
243	if (!Exists(index))	299	else if (target ==
244	{		↳ Constants.Any)
245	return Constants.Continue;	300	{
246	}	301	return _sourcesTreeMeth
247	return handler(GetLinkStruct(in		↳ ods.EachReference(s
	↳ dex));		↳ ource,
248	}		↳ handler);
249	if (restrictions.Count == 2)	302	}
250	{	303	else //if(source != Any &&
251	var index = restrictions[Consta		↳ target != Any)
	↳ nts.IndexPart];	304	{
252	var value = restrictions[1];	305	var link =
253	if (index == Constants.Any)		↳ _sourcesTreeMethods
254	{		↳ .Search(source,
255	if (value == Constants.Any)		↳ target);
256	{	306	return link ==
257	return Each(handler,		↳ Constants.Null ?
	↳ ArrayPool<ulong>.Em		↳ Constants.Continue
	↳ pty);		: handler(GetLinkSt
258	}		↳ ruct(link));
259	if (Each(handler, new[] {	307	}
	↳ index, value,	308	} else
	↳ Constants.Any }) ==	309	{
	↳ Constants.Break)	310	if (!Exists(index))
260	{	311	{
261	return Constants.Break;	312	return
262	}	313	↳ Constants.Continue;
263	return Each(handler, new[]		}
	↳ { index, Constants.Any,	314	if (source == Constants.Any
	↳ value });	315	↳ && target ==
264	}		↳ Constants.Any)
265	else	316	{
266	{	317	return handler(GetLinkS
267	if (!Exists(index))		↳ truct(index));
268	{	318	}
269	return	319	var storedLinkValue =
	↳ Constants.Continue;		↳ GetLinkUnsafe(index);
270	}	320	if (source != Constants.Any
271	if (value == Constants.Any)		↳ && target !=
272	{		↳ Constants.Any)
273	return handler(GetLinkS	321	{
	↳ truct(index));	322	if (storedLinkValue->So
274	}		↳ urce == source
275	var storedLinkValue =		↳ &&
	↳ GetLinkUnsafe(index);		storedLinkValue->Ta
276	if (storedLinkValue->Source	323	↳ rget ==
	↳ == value		↳ target)
277	storedLinkValue->Target	324	{
	↳ == value)	325	return
278	{		↳ handler(GetLink
279	return handler(GetLinkS		↳ Struct(index));
	↳ truct(index));	326	}
280	}	327	return
281	return Constants.Continue;		↳ Constants.Continue;
282	}	328	}
283	}	329	var value = default(id);
284	if (restrictions.Count == 3)	330	if (source == Constants.Any)
285	{	331	{
286	var index = restrictions[Consta	332	value = target;
	↳ nts.IndexPart];	333	}
287	var source = restrictions[Const	334	if (target == Constants.Any)
	↳ ants.SourcePart];	335	{
288	var target = restrictions[Const	336	value = source;
	↳ ants.TargetPart];	337	}
289	if (index == Constants.Any)	338	if (storedLinkValue->Source
290	{	339	↳ == value
			storedLinkValue->Target
			↳ == value)

```

340         {
341             return handler(GetLinkS
                ↳ truct(index));
342         }
343         return Constants.Continue;
344     }
345 }
346 throw new
    ↳ NotSupportedException("Другие
    ↳ размеры и способы ограничений
    ↳ не поддерживаются.");
347 }
348
349 /// <remarks>
350 /// TODO: Возможно можно перемещать
    ↳ значения, если указан индекс, но
    ↳ значение существует в другом месте
    ↳ (но не в менеджере памяти, а в
    ↳ логике Links)
351 /// </remarks>
352 [MethodImpl(MethodImplOptions.Aggressive
    ↳ eInlining)]
353 public id Update(IList<id> values)
354 {
355     var linkIndex =
        ↳ values[Constants.IndexPart];
356     var link = GetLinkUnsafe(linkIndex);
357     // Будет корректно работать только
        ↳ в том случае, если пространство
        ↳ выделенной связи предварительно
        ↳ заполнено нулями
358     if (link->Source != Constants.Null)
359     {
360         _sourcesTreeMethods.Detach(new
            ↳ IntPtr(&_header->FirstAsSou
            ↳ rce),
            ↳ linkIndex);
361     }
362     if (link->Target != Constants.Null)
363     {
364         _targetsTreeMethods.Detach(new
            ↳ IntPtr(&_header->FirstAsTar
            ↳ get),
            ↳ linkIndex);
365     }
366 #if ENABLE_TREE_AUTO_DEBUG_AND_VALIDATION
367     var leftTreeSize =
        ↳ _sourcesTreeMethods.GetSize(new
        ↳ IntPtr(&_header->FirstAsSource)
        ↳ );
368     var rightTreeSize =
        ↳ _targetsTreeMethods.GetSize(new
        ↳ IntPtr(&_header->FirstAsTarget)
        ↳ );
369     if (leftTreeSize != rightTreeSize)
370     {
371         throw new Exception("One of the
            ↳ trees is broken.");
372     }
373 #endif
374     link->Source =
        ↳ values[Constants.SourcePart];
375     link->Target =
        ↳ values[Constants.TargetPart];
376     if (link->Source != Constants.Null)
377     {
378         _sourcesTreeMethods.Attach(new
            ↳ IntPtr(&_header->FirstAsSou
            ↳ rce),
            ↳ linkIndex);
379     }
380     if (link->Target != Constants.Null)
381     {
382         _targetsTreeMethods.Attach(new
            ↳ IntPtr(&_header->FirstAsTar
            ↳ get),
            ↳ linkIndex);
383     }
384 #if ENABLE_TREE_AUTO_DEBUG_AND_VALIDATION
385     leftTreeSize =
        ↳ _sourcesTreeMethods.GetSize(new
        ↳ IntPtr(&_header->FirstAsSource)
        ↳ );
386     rightTreeSize =
        ↳ _targetsTreeMethods.GetSize(new
        ↳ IntPtr(&_header->FirstAsTarget)
        ↳ );
387     if (leftTreeSize != rightTreeSize)
388     {
389         throw new Exception("One of the
            ↳ trees is broken.");
390     }
391 #endif
392     return linkIndex;
393 }
394
395 [MethodImpl(MethodImplOptions.Aggressive
    ↳ eInlining)]
396 private IList<id> GetLinkStruct(id
    ↳ linkIndex)
397 {
398     var link = GetLinkUnsafe(linkIndex);
399     return new UInt64Link(linkIndex,
        ↳ link->Source, link->Target);
400 }
401
402 [MethodImpl(MethodImplOptions.Aggressive
    ↳ eInlining)]
403 private Link* GetLinkUnsafe(id
    ↳ linkIndex) => &_links[linkIndex];
404
405 /// <remarks>
406 /// TODO: Возможно нужно будет
    ↳ заполнение нулями, если внешнее API
    ↳ ими не заполняет пространство
407 /// </remarks>
408 public id Create()
409 {
410     var freeLink =
        ↳ _header->FirstFreeLink;
411     if (freeLink != Constants.Null)
412     {
413         _unusedLinksListMethods.Detach(
            ↳ freeLink);
414     }
415     else
416     {
417         if (_header->AllocatedLinks >
            ↳ Constants.MaxPossibleIndex)
418         {
419             throw new LinksLimitReached
                ↳ Exception(Constants.Max
                ↳ PossibleIndex);
420         }
421         if (_header->AllocatedLinks >=
            ↳ _header->ReservedLinks - 1)
422         {
423             _memory.ReservedCapacity +=
                ↳ _memory.ReservationStep;
424             SetPointers(_memory);
425             _header->ReservedLinks =
                ↳ (id)(_memory.ReservedCa
                ↳ pacity /
                ↳ sizeof(Link));
426         }
427         _header->AllocatedLinks++;
428         _memory.UsedCapacity +=
            ↳ sizeof(Link);
429         freeLink =
            ↳ _header->AllocatedLinks;
430     }
431     return freeLink;
432 }
433
434 public void Delete(id link)
435 {
436     if (link < _header->AllocatedLinks)
437     {
438         _unusedLinksListMethods.AttachA
            ↳ sFirst(link);
439     }
440     else if (link ==
        ↳ _header->AllocatedLinks)
441     {
442         _header->AllocatedLinks--;
443         _memory.UsedCapacity -=
            ↳ sizeof(Link);

```

```

444 // Убираем все связи,
445     ↳ находящиеся в списке
446     ↳ свободных в конце файла, до
447     ↳ тех пор, пока не дойдём до
448     ↳ первой существующей связи
449 // Позволяет оптимизировать
450     ↳ количество выделенных
451     ↳ связей (AllocatedLinks)
452 while (_header->AllocatedLinks
453     ↳ > 0 && IsUnusedLink(_header
454     ↳ ->AllocatedLinks))
455 {
456     _unusedLinksListMethods.Det
457     ↳ ach(_header->AllocatedL
458     ↳ inks);
459     _header->AllocatedLinks--;
460     _memory.UsedCapacity -=
461     ↳ sizeof(Link);
462 }
463 }
464
465 /// <remarks>
466 /// TODO: Возможно это должно быть
467     ↳ событием, вызываемым из IMemory, в
468     ↳ том случае, если адрес реально
469     ↳ поменялся
470 ///
471 /// Указатель this.links может быть в
472     ↳ том же месте,
473 /// так как 0-я связь не используется и
474     ↳ имеет такой же размер как Header,
475 /// поэтому header размещается в том же
476     ↳ месте, что и 0-я связь
477 /// </remarks>
478 private void
479     ↳ SetPointers(IResizableDirectMemory
480     ↳ memory)
481 {
482     if (memory == null)
483     {
484         _header = null;
485         _links = null;
486         _unusedLinksListMethods = null;
487         _targetsTreeMethods = null;
488         _unusedLinksListMethods = null;
489     }
490     else
491     {
492         _header = (LinksHeader*)(void*)
493             ↳ memory.Pointer;
494         _links = (Link*)(void*)memory.P
495             ↳ ointer;
496         _sourcesTreeMethods = new Links
497             ↳ SourcesTreeMethods(this);
498         _targetsTreeMethods = new Links
499             ↳ TargetsTreeMethods(this);
500         _unusedLinksListMethods = new
501             ↳ UnusedLinksListMethods(_lin
502             ↳ ks,
503             ↳ _header);
504     }
505 }
506
507 [MethodImpl(MethodImplOptions.Aggressive
508     ↳ eInlining)]
509 private bool Exists(id link) => link >=
510     ↳ Constants.MinPossibleIndex && link
511     ↳ <= _header->AllocatedLinks &&
512     ↳ !IsUnusedLink(link);
513
514 [MethodImpl(MethodImplOptions.Aggressive
515     ↳ eInlining)]
516 private bool IsUnusedLink(id link) =>
517     ↳ _header->FirstFreeLink == link

```

487

488
489
490
491
492
493

#region Disposable

```

protected override bool
    ↳ AllowMultipleDisposeCalls => true;

protected override void
    ↳ DisposeCore(bool manual, bool
    ↳ wasDisposed)

```

```

|| (
|_l
|_i
|_n
|_k
|_s
|_l
|_i
|_n
|_k
|_l
|_.
|_S
|_i
|_z
|_e
|_A
|_s
|_S
|_o
|_u
|_r
|_c
|_e
|_=
|_C
|_o
|_n
|_s
|_t
|_a
|_n
|_t
|_s
|_.
|_N
|_u
|_l
|_l
|_&
|_&
|_-
|_l
|_i
|_n
|_k
|_s
|_l
|_l
|_i
|_n
|_k
|_l
|_.
|_S
|_o
|_u
|_r
|_c
|_e
|_!
|_=
|_C
|_o
|_n
|_s
|_t
|_a
|_n
|_t
|_s
|_.
|_N
|_u
|_l
|_l
|_)
;

```

```

494     {
495         if (!wasDisposed)
496         {
497             SetPointers(null);
498         }
499         Disposable.TryDispose(_memory);
500     }
501     #endregion
502 }
503 }
504 }

```

./ResizableDirectMemory/UInt64ResizableDirectMemoryLinks.ListMethods.cs

```

1  using Platform.Collections.Methods.Lists;
2
3  namespace
4  ↪ Platform.Data.Doublets.ResizableDirectMemory
5  {
6      unsafe partial class
7      ↪ UInt64ResizableDirectMemoryLinks
8      {
9          private class UnusedLinksListMethods :
10             ↪ CircularDoublyLinkedListMethods<ulong>
11             {
12                 private readonly Link* _links;
13                 private readonly LinksHeader*
14                     ↪ _header;
15
16                 public UnusedLinksListMethods(Link*
17                     ↪ links, LinksHeader* header)
18                 {
19                     _links = links;
20                     _header = header;
21                 }
22
23                 protected override ulong GetFirst()
24                     ↪ => _header->FirstFreeLink;
25
26                 protected override ulong GetLast()
27                     ↪ => _header->LastFreeLink;
28
29                 protected override ulong
30                     ↪ GetPrevious(ulong element) =>
31                     ↪ _links[element].Source;
32
33                 protected override ulong
34                     ↪ GetNext(ulong element) =>
35                     ↪ _links[element].Target;
36
37                 protected override ulong GetSize()
38                     ↪ => _header->FreeLinks;
39
40                 protected override void
41                     ↪ SetFirst(ulong element) =>
42                     ↪ _header->FirstFreeLink =
43                     ↪ element;
44
45                 protected override void
46                     ↪ SetLast(ulong element) =>
47                     ↪ _header->LastFreeLink = element;
48
49                 protected override void
50                     ↪ SetPrevious(ulong element,
51                     ↪ ulong previous) =>
52                     ↪ _links[element].Source =
53                     ↪ previous;
54
55                 protected override void
56                     ↪ SetNext(ulong element, ulong
57                     ↪ next) => _links[element].Target
58                     ↪ = next;
59
60                 protected override void
61                     ↪ SetSize(ulong size) =>
62                     ↪ _header->FreeLinks = size;
63
64             }
65     }
66 }

```

./ResizableDirectMemory/UInt64ResizableDirectMemoryLinks.TreeMethods.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using System.Text;
5  using Platform.Collections.Methods.Trees;
6  using Platform.Data.Constants;
7

```

```

8  namespace
9  ↪ Platform.Data.Doublets.ResizableDirectMemory
10 {
11     unsafe partial class
12     ↪ UInt64ResizableDirectMemoryLinks
13     {
14         private abstract class
15             ↪ LinksTreeMethodsBase : SizedAndThre
16             ↪ adedAVLBalancedTreeMethods<ulong>
17         {
18             private readonly UInt64ResizableDir
19                 ↪ ectMemoryLinks
20                 ↪ _memory;
21             private readonly
22                 ↪ LinksCombinedConstants<ulong,
23                 ↪ ulong, int> _constants;
24             protected readonly Link* Links;
25             protected readonly LinksHeader*
26                 ↪ Header;
27
28             protected LinksTreeMethodsBase(UInt
29                 ↪ 64ResizableDirectMemoryLinks
30                 ↪ memory)
31             {
32                 Links = memory._links;
33                 Header = memory._header;
34                 _memory = memory;
35                 _constants = memory.Constants;
36             }
37
38             [MethodImpl(MethodImplOptions.AggressiveIn
39                 ↪ lining)]
40             protected abstract ulong
41                 ↪ GetTreeRoot();
42
43             [MethodImpl(MethodImplOptions.AggressiveIn
44                 ↪ lining)]
45             protected abstract ulong
46                 ↪ GetBasePartValue(ulong link);
47
48             public ulong this[ulong index]
49             {
50                 get
51                 {
52                     var root = GetTreeRoot();
53                     if (index >= GetSize(root))
54                     {
55                         return 0;
56                     }
57                     while (root != 0)
58                     {
59                         var left = GetLeftOrDef
60                             ↪ ault(root);
61                         var leftSize =
62                             ↪ GetSizeOrZero(left);
63                         if (index < leftSize)
64                         {
65                             root = left;
66                             continue;
67                         }
68                         if (index == leftSize)
69                         {
70                             return root;
71                         }
72                         root = GetRightOrDefaul
73                             ↪ t(root);
74                         index -= leftSize + 1;
75                     }
76                     return 0; // TODO:
77                     ↪ Impossible situation
78                     ↪ exception (only if tree
79                     ↪ structure broken)
80                 }
81             }
82
83             // TODO: Return indices range
84             ↪ instead of references count
85             public ulong
86                 ↪ CalculateReferences(ulong link)
87             {
88                 var root = GetTreeRoot();
89                 var total = GetSize(root);
90                 var totalRightIgnore = 0UL;
91                 while (root != 0)
92                 {
93                     var @base =
94                         ↪ GetBasePartValue(root);
95                 }
96             }
97         }
98     }
99 }

```

```

71         if (@base <= link)
72         {
73             root = GetRightOrDefault
74                 ↳ t(root);
75         }
76         else
77         {
78             totalRightIgnore +=
79                 ↳ GetRightSize(root)
80                 ↳ + 1;
81             root = GetLeftOrDefault
82                 ↳ (root);
83         }
84     }
85     root = GetTreeRoot();
86     var totalLeftIgnore = 0UL;
87     while (root != 0)
88     {
89         var @base =
90             ↳ GetBasePartValue(root);
91         if (@base >= link)
92         {
93             root = GetLeftOrDefault
94                 ↳ (root);
95         }
96         else
97         {
98             totalLeftIgnore +=
99                 ↳ GetLeftSize(root) +
100                 ↳ 1;
101             root = GetRightOrDefault
102                 ↳ t(root);
103         }
104     }
105     return total - totalRightIgnore
106         ↳ - totalLeftIgnore;
107 }
108
109 public ulong EachReference(ulong
110     ↳ link, Func<IList<ulong>, ulong>
111     ↳ handler)
112 {
113     var root = GetTreeRoot();
114     if (root == 0)
115     {
116         return _constants.Continue;
117     }
118     ulong first = 0, current = root;
119     while (current != 0)
120     {
121         var @base = GetBasePartValu
122             ↳ e(current);
123         if (@base >= link)
124         {
125             if (@base == link)
126             {
127                 first = current;
128             }
129             current = GetLeftOrDefa
130                 ↳ ult(current);
131         }
132         else
133         {
134             current = GetRightOrDef
135                 ↳ ault(current);
136         }
137     }
138     if (first != 0)
139     {
140         current = first;
141         while (true)
142         {
143             if (handler(_memory.Get
144                 ↳ LinkStruct(current)
145                 ↳ ) ==
146                 ↳ _constants.Break)
147             {
148                 return _constants.B
149                     ↳ reak;
150             }
151             current =
152                 ↳ GetNext(current);
153             if (current == 0 ||
154                 ↳ GetBasePartValue(cu
155                     ↳ rrent) !=
156                 ↳ link)
157             {
158                 break;
159             }
160             return _constants.Continue;
161         }
162     }
163 }
164
165 private class LinksSourcesTreeMethods :
166     ↳ LinksTreeMethodsBase
167 {
168     public LinksSourcesTreeMethods(UInt
169         ↳ 64ResizableDirectMemoryLinks
170         ↳ memory)
171         : base(memory)
172     {
173     }
174
175     protected override IntPtr
176         ↳ GetLeftPointer(ulong node) =>
177         ↳ new IntPtr(&Links[node].LeftAsS
178             ↳ ource);
179
180     protected override IntPtr
181         ↳ GetRightPointer(ulong node) =>
182         ↳ new IntPtr(&Links[node].RightAs
183             ↳ Source);
184
185     protected override ulong
186         ↳ GetLeftValue(ulong node) =>
187         ↳ Links[node].LeftAsSource;
188
189     protected override ulong
190         ↳ GetRightValue(ulong node) =>
191         ↳ Links[node].RightAsSource;
192
193     protected override ulong
194         ↳ GetSize(ulong node)
195     {
196         var previousValue =
197             ↳ Links[node].SizeAsSource;
198         //return MathHelpers.PartialRea
199             ↳ d(previousValue, 5,
200             ↳ -5);
201         return (previousValue &
202             ↳ 4294967264) >> 5;
203     }
204
205     protected override void
206         ↳ SetLeft(ulong node, ulong left)
207         ↳ => Links[node].LeftAsSource =
208             ↳ left;
209
210     protected override void
211         ↳ SetRight(ulong node, ulong
212             ↳ right) =>
213             ↳ Links[node].RightAsSource =
214                 ↳ right;
215
216     protected override void
217         ↳ SetSize(ulong node, ulong size)
218     {
219         var previousValue =
220             ↳ Links[node].SizeAsSource;
221         //var modified = MathHelpers.Pa
222             ↳ rtialWrite(previousValue,
223             ↳ size, 5, -5);
224         var modified = (previousValue &
225             ↳ 31) | ((size & 134217727)
226             ↳ << 5);
227         Links[node].SizeAsSource =
228             ↳ modified;
229     }
230 }

```



```

185
186 protected override bool
187     ↳ GetLeftIsChild(ulong node)
188 {
189     var previousValue =
190         ↳ Links[node].SizeAsSource;
191     //return (Integer)MathHelpers.P
192     ↳ artialRead(previousValue,
193     ↳ 4, 1);
194     return (previousValue & 16) >>
195     ↳ 4 == 1UL;
196 }
197
198 protected override void
199     ↳ SetLeftIsChild(ulong node, bool
200     ↳ value)
201 {
202     var previousValue =
203     ↳ Links[node].SizeAsSource;
204     //var modified = MathHelpers.Pa
205     ↳ rtialWrite(previousValue,
206     ↳ (ulong)(Integer)value, 4,
207     ↳ 1);
208     var modified = (previousValue &
209     ↳ 4294967279) | ((value ? 1UL
210     ↳ : 0UL) << 4);
211     Links[node].SizeAsSource =
212     ↳ modified;
213 }
214
215 protected override bool
216     ↳ GetRightIsChild(ulong node)
217 {
218     var previousValue =
219     ↳ Links[node].SizeAsSource;
220     //return (Integer)MathHelpers.P
221     ↳ artialRead(previousValue,
222     ↳ 3, 1);
223     return (previousValue & 8) >> 3
224     ↳ == 1UL;
225 }
226
227 protected override void
228     ↳ SetRightIsChild(ulong node,
229     ↳ bool value)
230 {
231     var previousValue =
232     ↳ Links[node].SizeAsSource;
233     //var modified = MathHelpers.Pa
234     ↳ rtialWrite(previousValue,
235     ↳ (ulong)(Integer)value, 3,
236     ↳ 1);
237     var modified = (previousValue &
238     ↳ 4294967287) | ((value ? 1UL
239     ↳ : 0UL) << 3);
240     Links[node].SizeAsSource =
241     ↳ modified;
242 }
243
244 protected override sbyte
245     ↳ GetBalance(ulong node)
246 {
247     var previousValue =
248     ↳ Links[node].SizeAsSource;
249     //var value = MathHelpers.Parti
250     ↳ alRead(previousValue, 0,
251     ↳ 3);
252     var value = previousValue & 7;
253     var unpackedValue =
254     ↳ (sbyte)((value & 4) > 0 ?
255     ↳ ((value & 4) << 5) | value
256     ↳ & 3 | 124 : value & 3);
257     return unpackedValue;
258 }
259
260 protected override void
261     ↳ SetBalance(ulong node, sbyte
262     ↳ value)
263 {
264     var previousValue =
265     ↳ Links[node].SizeAsSource;
266     var packagedValue =
267     ↳ (ulong)((((byte)value >> 5)
268     ↳ & 4) | value & 3);
269
270     //var modified = MathHelpers.Pa
271     ↳ rtialWrite(previousValue,
272     ↳ packagedValue, 0, 3);
273     var modified = (previousValue &
274     ↳ 4294967288) |
275     ↳ (packagedValue & 7);
276     Links[node].SizeAsSource =
277     ↳ modified;
278 }
279
280 protected override bool
281     ↳ FirstIsToTheLeftOfSecond(ulong
282     ↳ first, ulong second)
283     => Links[first].Source <
284     ↳ Links[second].Source ||
285     ↳ (Links[first].Source ==
286     ↳ Links[second].Source &&
287     ↳ Links[first].Target <
288     ↳ Links[second].Target);
289
290 protected override bool
291     ↳ FirstIsToTheRightOfSecond(ulong
292     ↳ first, ulong second)
293     => Links[first].Source >
294     ↳ Links[second].Source ||
295     ↳ (Links[first].Source ==
296     ↳ Links[second].Source &&
297     ↳ Links[first].Target >
298     ↳ Links[second].Target);
299
300 protected override ulong
301     ↳ GetTreeRoot() =>
302     ↳ Header->FirstAsSource;
303
304 protected override ulong
305     ↳ GetBasePartValue(ulong link) =>
306     ↳ Links[link].Source;
307
308 /// <summary>
309 /// Выполняет поиск и возвращает
310 /// индекс связи с указанными
311 /// Source (началом) и Target
312 /// (концом)
313 /// по дереву (индексу) связей,
314 /// отсортированному по Source, а
315 /// затем по Target.
316 /// </summary>
317 /// <param name="source">Индекс
318 /// связи, которая является началом
319 /// на искомой связи.</param>
320 /// <param name="target">Индекс
321 /// связи, которая является концом
322 /// на искомой связи.</param>
323 /// <returns>Индекс искомой
324 /// связи.</returns>
325 public ulong Search(ulong source,
326     ↳ ulong target)
327 {
328     var root =
329     ↳ Header->FirstAsSource;
330     while (root != 0)
331     {
332         var rootSource =
333         ↳ Links[root].Source;
334         var rootTarget =
335         ↳ Links[root].Target;
336         if (FirstIsToTheLeftOfSecon
337         ↳ d(source, target,
338         ↳ rootSource,
339         ↳ rootTarget)) //
340         ↳ node.Key < root.Key
341         {
342             root = GetLeftOrDefault
343             ↳ (root);
344         }
345         else if (FirstIsToTheRightO
346         ↳ fSecond(source, target,
347         ↳ rootSource,
348         ↳ rootTarget)) //
349         ↳ node.Key > root.Key
350         {
351             root = GetRightOrDefaul
352             ↳ t(root);
353         }
354         else // node.Key == root.Key
355         {
356

```

```

270         return root;
271     }
272     }
273     return 0;
274 }
275
276 [MethodImpl(MethodImplOptions.AggressiveInlining)]
277 private static bool
278     FirstIsToTheLeftOfSecond(ulong
279         firstSource, ulong firstTarget,
280         ulong secondSource, ulong
281         secondTarget)
282     => firstSource < secondSource
283         || (firstSource ==
284             secondSource && firstTarget
285             < secondTarget);
286
287 [MethodImpl(MethodImplOptions.AggressiveInlining)]
288 private static bool
289     FirstIsToTheRightOfSecond(ulong
290         firstSource, ulong firstTarget,
291         ulong secondSource, ulong
292         secondTarget)
293     => firstSource > secondSource
294         || (firstSource ==
295             secondSource && firstTarget
296             > secondTarget);
297
298 [MethodImpl(MethodImplOptions.AggressiveInlining)]
299 protected override void
300     ClearNode(ulong node)
301 {
302     Links[node].LeftAsSource = OUL;
303     Links[node].RightAsSource = OUL;
304     Links[node].SizeAsSource = OUL;
305 }
306
307 [MethodImpl(MethodImplOptions.AggressiveInlining)]
308 protected override ulong GetZero()
309     => OUL;
310
311 [MethodImpl(MethodImplOptions.AggressiveInlining)]
312 protected override ulong GetOne()
313     => 1UL;
314
315 [MethodImpl(MethodImplOptions.AggressiveInlining)]
316 protected override ulong GetTwo()
317     => 2UL;
318
319 [MethodImpl(MethodImplOptions.AggressiveInlining)]
320 protected override bool
321     ValueEqualToZero(IntPtr
322         pointer) =>
323     *(ulong*)pointer.ToPointer() ==
324     OUL;
325
326 [MethodImpl(MethodImplOptions.AggressiveInlining)]
327 protected override bool
328     EqualToZero(ulong value) =>
329     value == OUL;
330
331 [MethodImpl(MethodImplOptions.AggressiveInlining)]
332 protected override bool
333     IsEquals(ulong first, ulong
334     second) => first == second;
335
336 [MethodImpl(MethodImplOptions.AggressiveInlining)]
337 protected override bool
338     GreaterThanZero(ulong value) =>
339     value > OUL;
340
341 [MethodImpl(MethodImplOptions.AggressiveInlining)]
342 protected override bool
343     GreaterThan(ulong first, ulong
344     second) => first > second;
345
346 [MethodImpl(MethodImplOptions.AggressiveInlining)]
347 protected override bool
348     GreaterOrEqualThan(ulong first,
349     ulong second) => first >=
350     second;
351
352 [MethodImpl(MethodImplOptions.AggressiveInlining)]
353 protected override bool
354     GreaterOrEqualThanZero(ulong
355     value) => value == 0; // value >= 0
356     is always true for ulong
357
358 [MethodImpl(MethodImplOptions.AggressiveInlining)]
359 protected override bool
360     LessOrEqualThanZero(ulong
361     value) => value == 0; // value
362     is always >= 0 for ulong
363
364 [MethodImpl(MethodImplOptions.AggressiveInlining)]
365 protected override bool
366     LessOrEqualThan(ulong first,
367     ulong second) => first <=
368     second;
369
370 [MethodImpl(MethodImplOptions.AggressiveInlining)]
371 protected override bool
372     LessThanZero(ulong value) =>
373     false; // value < 0 is always
374     false for ulong
375
376 [MethodImpl(MethodImplOptions.AggressiveInlining)]
377 protected override bool
378     LessThan(ulong first, ulong
379     second) => first < second;
380
381 [MethodImpl(MethodImplOptions.AggressiveInlining)]
382 protected override ulong
383     Increment(ulong value) =>
384     ++value;
385
386 [MethodImpl(MethodImplOptions.AggressiveInlining)]
387 protected override ulong
388     Decrement(ulong value) =>
389     --value;
390
391 [MethodImpl(MethodImplOptions.AggressiveInlining)]
392 protected override ulong Add(ulong
393     first, ulong second) => first +
394     second;
395
396 [MethodImpl(MethodImplOptions.AggressiveInlining)]
397 protected override ulong
398     Subtract(ulong first, ulong
399     second) => first - second;
400 }
401
402 private class LinksTargetsTreeMethods :
403     LinksTreeMethodsBase
404 {
405     public LinksTargetsTreeMethods(UInt
406         64ResizableDirectMemoryLinks
407         memory)
408         : base(memory)
409     {
410     }
411
412     //protected override IntPtr
413     GetLeft(ulong node) => new IntPtr
414     tr(&Links[node].LeftAsTarget);
415
416     //protected override IntPtr
417     GetRight(ulong node) => new IntPtr
418     tr(&Links[node].RightAsTarget);
419 }

```

```

357 //protected override ulong
358     ↳ GetSize(ulong node) =>
359     ↳ Links[node].SizeAsTarget;
360
361 //protected override void
362     ↳ SetLeft(ulong node, ulong left)
363     ↳ => Links[node].LeftAsTarget =
364     ↳ left;
365
366 //protected override void
367     ↳ SetRight(ulong node, ulong
368     ↳ right) =>
369     ↳ Links[node].RightAsTarget =
370     ↳ right;
371
372 //protected override void
373     ↳ SetSize(ulong node, ulong size)
374     ↳ => Links[node].SizeAsTarget =
375     ↳ size;
376
377 protected override IntPtr
378     ↳ GetLeftPointer(ulong node) =>
379     ↳ new IntPtr(&Links[node].LeftAsT
380     ↳ arget);
381
382 protected override IntPtr
383     ↳ GetRightPointer(ulong node) =>
384     ↳ new IntPtr(&Links[node].RightAs
385     ↳ Target);
386
387 protected override ulong
388     ↳ GetLeftValue(ulong node) =>
389     ↳ Links[node].LeftAsTarget;
390
391 protected override ulong
392     ↳ GetRightValue(ulong node) =>
393     ↳ Links[node].RightAsTarget;
394
395 protected override ulong
396     ↳ GetSize(ulong node)
397     {
398         var previousValue =
399             ↳ Links[node].SizeAsTarget;
400         //return MathHelpers.Pa
401             ↳ d(previousValue, 5,
402             ↳ -5);
403         return (previousValue &
404             ↳ 4294967264) >> 5;
405     }
406
407 protected override void
408     ↳ SetLeft(ulong node, ulong left)
409     ↳ => Links[node].LeftAsTarget =
410     ↳ left;
411
412 protected override void
413     ↳ SetRight(ulong node, ulong
414     ↳ right) =>
415     ↳ Links[node].RightAsTarget =
416     ↳ right;
417
418 protected override void
419     ↳ SetSize(ulong node, ulong size)
420     {
421         var previousValue =
422             ↳ Links[node].SizeAsTarget;
423         //var modified = MathHelpers.Pa
424             ↳ rtialWrite(previousValue,
425             ↳ size, 5, -5);
426         var modified = (previousValue &
427             ↳ 31) | ((size & 134217727)
428             ↳ << 5);
429         Links[node].SizeAsTarget =
430             ↳ modified;
431     }
432
433 protected override bool
434     ↳ GetLeftIsChild(ulong node)
435     {
436         var previousValue =
437             ↳ Links[node].SizeAsTarget;
438         //return (Integer)MathHelpers.P
439             ↳ artialRead(previousValue,
440             ↳ 4, 1);
441     }
442
443 return (previousValue & 16) >>
444     ↳ 4 == 1UL;
445 // TODO: Check if this is
446     ↳ possible to use
447 //var nodeSize = GetSize(node);
448 //var left = GetLeftValue(node);
449 //var leftSize =
450     ↳ GetSizeOrZero(left);
451 //return leftSize > 0 &&
452     ↳ nodeSize > leftSize;
453 }
454
455 protected override void
456     ↳ SetLeftIsChild(ulong node, bool
457     ↳ value)
458     {
459         var previousValue =
460             ↳ Links[node].SizeAsTarget;
461         //var modified = MathHelpers.Pa
462             ↳ rtialWrite(previousValue,
463             ↳ (ulong)(Integer)value, 4,
464             ↳ 1);
465         var modified = (previousValue &
466             ↳ 4294967279) | ((value ? 1UL
467             ↳ : 0UL) << 4);
468         Links[node].SizeAsTarget =
469             ↳ modified;
470     }
471
472 protected override bool
473     ↳ GetRightIsChild(ulong node)
474     {
475         var previousValue =
476             ↳ Links[node].SizeAsTarget;
477         //return (Integer)MathHelpers.P
478             ↳ artialRead(previousValue,
479             ↳ 3, 1);
480         return (previousValue & 8) >> 3
481             ↳ == 1UL;
482         // TODO: Check if this is
483             ↳ possible to use
484         //var nodeSize = GetSize(node);
485         //var right =
486             ↳ GetRightValue(node);
487         //var rightSize =
488             ↳ GetSizeOrZero(right);
489         //return rightSize > 0 &&
490             ↳ nodeSize > rightSize;
491     }
492
493 protected override void
494     ↳ SetRightIsChild(ulong node,
495     ↳ bool value)
496     {
497         var previousValue =
498             ↳ Links[node].SizeAsTarget;
499         //var modified = MathHelpers.Pa
500             ↳ rtialWrite(previousValue,
501             ↳ (ulong)(Integer)value, 3,
502             ↳ 1);
503         var modified = (previousValue &
504             ↳ 4294967287) | ((value ? 1UL
505             ↳ : 0UL) << 3);
506         Links[node].SizeAsTarget =
507             ↳ modified;
508     }
509
510 protected override sbyte
511     ↳ GetBalance(ulong node)
512     {
513         var previousValue =
514             ↳ Links[node].SizeAsTarget;
515         //var value = MathHelpers.Parti
516             ↳ alRead(previousValue, 0,
517             ↳ 3);
518         var value = previousValue & 7;
519         var unpackedValue =
520             ↳ (sbyte)((value & 4) > 0 ?
521             ↳ ((value & 4) << 5) | value
522             ↳ & 3 | 124 : value & 3);
523         return unpackedValue;
524     }

```

```

442     protected override void
        ↳ SetBalance(ulong node, sbyte
        ↳ value)
443     {
444         var previousValue =
        ↳ Links[node].SizeAsTarget;
445         var packagedValue =
        ↳ (ulong)((((byte)value >> 5)
        ↳ & 4) | value & 3);
446         //var modified = MathHelpers.Pa
        ↳ rtialWrite(previousValue,
        ↳ packagedValue, 0, 3);
447         var modified = (previousValue &
        ↳ 4294967288) |
        ↳ (packagedValue & 7);
448         Links[node].SizeAsTarget =
        ↳ modified;
449     }
450
451     protected override bool
        ↳ FirstIsToTheLeftOfSecond(ulong
        ↳ first, ulong second)
452     => Links[first].Target <
        ↳ Links[second].Target ||
453         (Links[first].Target ==
        ↳ Links[second].Target &&
        ↳ Links[first].Source <
        ↳ Links[second].Source);
454
455     protected override bool
        ↳ FirstIsToTheRightOfSecond(ulong
        ↳ first, ulong second)
456     => Links[first].Target >
        ↳ Links[second].Target ||
457         (Links[first].Target ==
        ↳ Links[second].Target &&
        ↳ Links[first].Source >
        ↳ Links[second].Source);
458
459     protected override ulong
        ↳ GetTreeRoot() =>
        ↳ Header->FirstAsTarget;
460
461     protected override ulong
        ↳ GetBasePartValue(ulong link) =>
        ↳ Links[link].Target;
462
463     [MethodImpl(MethodImplOptions.AggressiveInlining)]
464     protected override void
        ↳ ClearNode(ulong node)
465     {
466         Links[node].LeftAsTarget = OUL;
467         Links[node].RightAsTarget = OUL;
468         Links[node].SizeAsTarget = OUL;
469     }
470 }
471 }
472 }

```

```

./Sequences/Converters/BalancedVariantConverter.cs
1 using System.Collections.Generic;
2
3 namespace
    ↳ Platform.Data.Doublets.Sequences.Converters
4 {
5     public class
        ↳ BalancedVariantConverter<TLink> :
        ↳ LinksListToSequenceConverterBase<TLink>
6     {
7         public BalancedVariantConverter(ILinks<
        ↳ TLink> links) : base(links) {
        ↳ }
8
9         public override TLink
        ↳ Convert(ICollection<TLink> sequence)
10        {
11            var length = sequence.Count;
12            if (length < 1)
13            {
14                return default;
15            }
16            if (length == 1)
17            {
18                return sequence[0];
19            }

```

```

20 // Make copy of next layer
21 if (length > 2)
22 {
23     // TODO: Try to use stackalloc
        ↳ (which at the moment is not
        ↳ working with generics) but
        ↳ will be possible with Sigil
24     var halvedSequence = new
        ↳ TLink[(length / 2) +
        ↳ (length % 2)];
25     HalveSequence(halvedSequence,
        ↳ sequence, length);
26     sequence = halvedSequence;
27     length = halvedSequence.Length;
28 }
29 // Keep creating layer after layer
30 while (length > 2)
31 {
32     HalveSequence(sequence,
        ↳ sequence, length);
33     length = (length / 2) + (length
        ↳ % 2);
34 }
35 return
        ↳ Links.GetOrCreate(sequence[0],
        ↳ sequence[1]);
36 }
37
38 private void HalveSequence(ICollection<TLink>
        ↳ destination, ICollection<TLink> source,
        ↳ int length)
39 {
40     var loopedLength = length - (length
        ↳ % 2);
41     for (var i = 0; i < loopedLength; i
        ↳ += 2)
42     {
43         destination[i / 2] = Links.GetO
        ↳ rCreate(source[i], source[i
        ↳ + 1]);
44     }
45     if (length > loopedLength)
46     {
47         destination[length / 2] =
        ↳ source[length - 1];
48     }
49 }
50 }
51 }

```

./Sequences/Converters/CompressingConverter.cs

```

1 using System;
2 using System.Collections.Generic;
3 using System.Runtime.CompilerServices;
4 using Platform.Interfaces;
5 using Platform.Collections;
6 using Platform.Helpers.Singletons;
7 using Platform.Numbers;
8 using Platform.Data.Constants;
9 using Platform.Data.Doublets.Sequences.Frequenc
    ↳ ies.Cache;
10
11 namespace
    ↳ Platform.Data.Doublets.Sequences.Converters
12 {
13     /// <remarks>
14     /// TODO: Возможно будет лучше если
        ↳ алгоритм будет выполняться полностью
        ↳ изолированно от Links на этапе сжатия.
15     /// А именно будет создаваться
        ↳ временный список пар необходимых для
        ↳ выполнения сжатия, в таком случае тип
        ↳ значения элемента массива может быть
        ↳ любым, как char так и ulong.
16     /// Как только список/словарь пар был
        ↳ выявлен можно разом выполнить создание
        ↳ всех ЭТИХ пар, а так же разом выполнить
        ↳ замену.
17     /// </remarks>
18     public class CompressingConverter<TLink> :
        ↳ LinksListToSequenceConverterBase<TLink>
19     {
20         private static readonly
        ↳ LinksCombinedConstants<bool, TLink,
        ↳ long> _constants = Default<LinksCom
        ↳ binedConstants<bool, TLink,
        ↳ long>>.Instance;

```

```

21     private static readonly
    ↪ EqualityComparer<TLink>
    ↪ _equalityComparer =
    ↪ EqualityComparer<TLink>.Default;
22     private static readonly Comparer<TLink>
    ↪ _comparer = Comparer<TLink>.Default;
23
24     private readonly
    ↪ IConverter<IList<TLink>, TLink>
    ↪ _baseConverter;
25     private readonly
    ↪ LinkFrequenciesCache<TLink>
    ↪ _doubletFrequenciesCache;
26     private readonly TLink
    ↪ _minFrequencyToCompress;
27     private readonly bool
    ↪ _doInitialFrequenciesIncrement;
28     private Doublet<TLink> _maxDoublet;
29     private LinkFrequency<TLink>
    ↪ _maxDoubletData;
30
31     private struct HalfDoublet
32     {
33         public TLink Element;
34         public LinkFrequency<TLink>
    ↪ DoubletData;
35
36         public HalfDoublet(TLink element,
    ↪ LinkFrequency<TLink>
    ↪ doubletData)
37         {
38             Element = element;
39             DoubletData = doubletData;
40         }
41
42         public override string ToString()
    ↪ => $"{Element}:
    ↪ ({DoubletData})";
43     }
44
45     public
    ↪ CompressingConverter(ILinks<TLink>
    ↪ links, IConverter<IList<TLink>,
    ↪ TLink> baseConverter,
    ↪ LinkFrequenciesCache<TLink>
    ↪ doubletFrequenciesCache)
46     : this(links, baseConverter,
    ↪ doubletFrequenciesCache,
    ↪ Integer<TLink>.One, true)
47     {
48     }
49
50     public
    ↪ CompressingConverter(ILinks<TLink>
    ↪ links, IConverter<IList<TLink>,
    ↪ TLink> baseConverter,
    ↪ LinkFrequenciesCache<TLink>
    ↪ doubletFrequenciesCache, bool
    ↪ doInitialFrequenciesIncrement)
51     : this(links, baseConverter,
    ↪ doubletFrequenciesCache,
    ↪ Integer<TLink>.One,
    ↪ doInitialFrequenciesIncrement)
52     {
53     }
54
55     public
    ↪ CompressingConverter(ILinks<TLink>
    ↪ links, IConverter<IList<TLink>,
    ↪ TLink> baseConverter,
    ↪ LinkFrequenciesCache<TLink>
    ↪ doubletFrequenciesCache, TLink
    ↪ minFrequencyToCompress, bool
    ↪ doInitialFrequenciesIncrement)
56     : base(links)
57     {
58         _baseConverter = baseConverter;
59         _doubletFrequenciesCache =
    ↪ doubletFrequenciesCache;
60         if (_comparer.Compare(minFrequencyTo
    ↪ Compress, Integer<TLink>.One)
    ↪ < 0)
61         {
62             minFrequencyToCompress =
    ↪ Integer<TLink>.One;
63         }
64         _minFrequencyToCompress =
    ↪ minFrequencyToCompress;
65
66         _doInitialFrequenciesIncrement =
    ↪ doInitialFrequenciesIncrement;
67         ResetMaxDoublet();
68     }
69
70     public override TLink
    ↪ Convert(IList<TLink> source) =>
    ↪ _baseConverter.Convert(Compress(sou
    ↪ rce));
71
72     /// <remarks>
73     /// Original algorithm idea:
74     ↪ https://en.wikipedia.org/wiki/Byte_
    ↪ pair_encoding
75     /// Faster version (doublets'
    ↪ frequencies dictionary is not
    ↪ recreated).
76     /// </remarks>
77     private IList<TLink>
    ↪ Compress(IList<TLink> sequence)
78     {
79         if (sequence.IsNullOrEmpty())
80         {
81             return null;
82         }
83         if (sequence.Count == 1)
84         {
85             return sequence;
86         }
87         if (sequence.Count == 2)
88         {
89             return new[] { Links.GetOrCreat
    ↪ e(sequence[0], sequence[1])
    ↪ };
90         }
91         // TODO: arraypool with min size
92         ↪ (to improve cache locality) or
93         ↪ stackalloc with Sigil
94         var copy = new
    ↪ HalfDoublet[sequence.Count];
95         Doublet<TLink> doublet = default;
96         for (var i = 1; i < sequence.Count;
    ↪ i++)
97         {
98             doublet.Source = sequence[i -
    ↪ 1];
99             doublet.Target = sequence[i];
100             LinkFrequency<TLink> data;
101             if (_doInitialFrequenciesIncrem
    ↪ ent)
102             {
103                 data = _doubletFrequenciesC
    ↪ ache.IncrementFrequency
    ↪ (ref
    ↪ doublet);
104             }
105             else
106             {
107                 data = _doubletFrequenciesC
    ↪ ache.GetFrequency(ref
    ↪ doublet);
108                 if (data == null)
109                 {
110                     throw new NotSupportedE
    ↪ xception("If you
    ↪ ask not to
    ↪ increment
    ↪ frequencies, it is
    ↪ expected that all
    ↪ frequencies for the
    ↪ sequence are
    ↪ prepared.");
111                 }
112             }
113             copy[i - 1].Element =
    ↪ sequence[i - 1];
114             copy[i - 1].DoubletData = data;
115             UpdateMaxDoublet(ref doublet,
    ↪ data);
116         }
117         copy[sequence.Count - 1].Element =
    ↪ sequence[sequence.Count - 1];
118         copy[sequence.Count -
    ↪ 1].DoubletData = new
    ↪ LinkFrequency<TLink>();

```

```

115         if (_comparer.Compare(_maxDoubletDa_ 160
116             ↪ ta.Frequency, default) >
117             ↪ 0)
118         {
119             var newLength =
120                 ↪ ReplaceDoublets(copy);
121             sequence = new TLink[newLength];
122             for (int i = 0; i < newLength;
123                 ↪ i++)
124             {
125                 sequence[i] =
126                 ↪ copy[i].Element;
127             }
128         }
129         return sequence;
130     }
131
132     /// <remarks>
133     /// Original algorithm idea:
134     ↪ https://en.wikipedia.org/wiki/Byte_
135     ↪ pair_encoding
136     /// </remarks>
137     private int
138     ↪ ReplaceDoublets(HalfDoublet[] copy)
139     {
140         var oldLength = copy.Length;
141         var newLength = copy.Length;
142         while (_comparer.Compare(_maxDoublet_ 162
143             ↪ tData.Frequency, default) >
144             ↪ 0)
145         {
146             var maxDoubletSource =
147             ↪ _maxDoublet.Source;
148             var maxDoubletTarget =
149             ↪ _maxDoublet.Target;
150             if (_equalityComparer.Equals(_m_ 163
151             ↪ axDoubletData.Link,
152             ↪ _constants.Null))
153             {
154                 _maxDoubletData.Link =
155                 ↪ Links.GetOrCreate(maxDo_
156                 ↪ ubletSource,
157                 ↪ maxDoubletTarget);
158             }
159             var maxDoubletReplacementLink =
160             ↪ _maxDoubletData.Link;
161             oldLength--;
162             var oldLengthMinusTwo =
163             ↪ oldLength - 1;
164             // Substitute all usages
165             int w = 0, r = 0; // (r ==
166             ↪ read, w == write)
167             for (; r < oldLength; r++)
168             {
169                 if (_equalityComparer.Equal_ 164
170                 ↪ s(copy[r].Element,
171                 ↪ maxDoubletSource) &&
172                 ↪ _equalityComparer.Equal_
173                 ↪ s(copy[r + 1].Element,
174                 ↪ maxDoubletTarget))
175                 {
176                     if (r > 0)
177                     {
178                         var previous =
179                         ↪ copy[w -
180                         ↪ 1].Element;
181                         copy[w - 1].Doublet_ 165
182                         ↪ Data.DecrementF_
183                         ↪ requency();
184                         copy[w -
185                         ↪ 1].DoubletData
186                         ↪ = _doubletFrequ_
187                         ↪ enciesCache.Inc_
188                         ↪ rementFrequency_
189                         ↪ (previous,
190                         ↪ maxDoubletRepla_
191                         ↪ cementLink);
192                     }
193                     if (r <
194                         ↪ oldLengthMinusTwo)
195                     {
196                         var next = copy[r +
197                         ↪ 2].Element;
198                     }
199                 }
200             }
201             newLength--;
202             if (newLength == 0)
203             {
204                 return oldLength;
205             }
206             else
207             {
208                 copy[w++] = copy[r];
209             }
210             if (w < newLength)
211             {
212                 copy[w] = copy[r];
213             }
214             oldLength = newLength;
215             ResetMaxDoublet();
216             UpdateMaxDoublet(copy,
217                 ↪ newLength);
218         }
219         return newLength;
220     }
221
222     [MethodImpl(MethodImplOptions.Aggressive_ 166
223     ↪ eInlining)]
224     private void ResetMaxDoublet()
225     {
226         _maxDoublet = new Doublet<TLink>();
227         _maxDoubletData = new
228             ↪ LinkFrequency<TLink>();
229     }
230
231     [MethodImpl(MethodImplOptions.Aggressive_ 167
232     ↪ eInlining)]
233     private void
234     ↪ UpdateMaxDoublet(HalfDoublet[]
235     ↪ copy, int length)
236     {
237         Doublet<TLink> doublet = default;
238         for (var i = 1; i < length; i++)
239         {
240             doublet.Source = copy[i -
241             ↪ 1].Element;
242             doublet.Target =
243             ↪ copy[i].Element;
244             UpdateMaxDoublet(ref doublet,
245                 ↪ copy[i - 1].DoubletData);
246         }
247     }
248
249     [MethodImpl(MethodImplOptions.Aggressive_ 168
250     ↪ eInlining)]
251     private void UpdateMaxDoublet(ref
252     ↪ Doublet<TLink> doublet,
253     ↪ LinkFrequency<TLink> data)
254     {
255         var frequency = data.Frequency;
256         var maxFrequency =
257             ↪ _maxDoubletData.Frequency;
258         //if (frequency >
259             ↪ _minFrequencyToCompress &&
260             ↪ (maxFrequency < frequency ||
261             ↪ (maxFrequency == frequency &&
262             ↪ doublet.Source + doublet.Target
263             ↪ < /* gives better compression
264             ↪ string data (and gives
265             ↪ collisions quickly) */
266             ↪ _maxDoublet.Source +
267             ↪ _maxDoublet.Target)))
268         {
269             if (_comparer.Compare(frequency,
270                 ↪ _minFrequencyToCompress) > 0 &&
271             ↪ frequency < maxFrequency)
272             {
273                 maxFrequency = frequency;
274                 doublet.Source = doublet.Target = frequency;
275             }
276         }
277         _maxDoubletData.Frequency = maxFrequency;
278         _maxDoubletData.LinkFrequency = data;
279     }
280
281     [MethodImpl(MethodImplOptions.Aggressive_ 169
282     ↪ eInlining)]
283     private void UpdateMaxDoublet(ref
284     ↪ Doublet<TLink> doublet,
285     ↪ LinkFrequency<TLink> data)
286     {
287         var frequency = data.Frequency;
288         var maxFrequency =
289             ↪ _maxDoubletData.Frequency;
290         //if (frequency >
291             ↪ _minFrequencyToCompress &&
292             ↪ (maxFrequency < frequency ||
293             ↪ (maxFrequency == frequency &&
294             ↪ doublet.Source + doublet.Target
295             ↪ < /* gives better compression
296             ↪ string data (and gives
297             ↪ collisions quickly) */
298             ↪ _maxDoublet.Source +
299             ↪ _maxDoublet.Target)))
300         {
301             if (_comparer.Compare(frequency,
302                 ↪ _minFrequencyToCompress) > 0 &&
303             ↪ frequency < maxFrequency)
304             {
305                 maxFrequency = frequency;
306                 doublet.Source = doublet.Target = frequency;
307             }
308         }
309         _maxDoubletData.Frequency = maxFrequency;
310         _maxDoubletData.LinkFrequency = data;
311     }

```

```

209         (_comparer.Compare(maxFrequency,
210             frequency) < 0 ||
211             (_equalityComparer.Equals(maxFrequency, frequency) &&
212             _comparer.Compare(ArithmeticHelpers.Add(doublet.Source,
213             doublet.Target),
214             ArithmeticHelpers.Add(_maxDoublet.Source,
215             _maxDoublet.Target)) > 0)))
216         /* gives better stability
217         and better compression on
218         sequent data and even on
219         runder numbers data (but
220         gives collisions anyway) */
221     {
222         _maxDoublet = doublet;
223         _maxDoubletData = data;
224     }
225 }
226 }

./Sequences/Converters/LinksListToSequenceConverterBase.cs
1 using System.Collections.Generic;
2 using Platform.Interfaces;
3
4 namespace
5     Platform.Data.Doublets.Sequences.Converters
6 {
7     public abstract class
8         LinksListToSequenceConverterBase<TLink>
9         : IConverter<IList<TLink>, TLink>
10     {
11         protected readonly IList<TLink> Links;
12         public LinksListToSequenceConverterBase(
13             (IList<TLink> links) => Links =
14             links;
15         public abstract TLink
16             Convert(IList<TLink> source);
17     }
18 }

./Sequences/Converters/OptimalVariantConverter.cs
1 using System.Collections.Generic;
2 using System.Linq;
3 using Platform.Interfaces;
4
5 namespace
6     Platform.Data.Doublets.Sequences.Converters
7 {
8     public class OptimalVariantConverter<TLink>
9         : LinksListToSequenceConverterBase<TLink>
10     {
11         private static readonly
12             EqualityComparer<TLink>
13             _equalityComparer =
14             EqualityComparer<TLink>.Default;
15         private static readonly Comparer<TLink>
16             _comparer = Comparer<TLink>.Default;
17
18         private readonly
19             IConverter<IList<TLink>> _sequenceTo
20             oItsLocalElementLevelsConverter;
21
22         public OptimalVariantConverter(IList<T
23             Link> links,
24             IConverter<IList<TLink>> sequenceTo
25             ItsLocalElementLevelsConverter) :
26             base(links)
27         => _sequenceToItsLocalElementLevels
28             Converter =
29             sequenceToItsLocalElementLevels
30             Converter;
31
32         public override TLink
33             Convert(IList<TLink> sequence)
34         {
35             var length = sequence.Count;
36             if (length == 1)
37             {
38                 return sequence[0];
39             }
40             var links = Links;
41             if (length == 2)
42             {
43                 return links.GetOrCreate(sequence
44                     [0],
45                     sequence[1]);
46             }
47             sequence = sequence.ToArray();
48             var levels =
49                 _sequenceToItsLocalElementLevel
50                 sConverter.Convert(sequence);
51             while (length > 2)
52             {
53                 var levelRepeat = 1;
54                 var currentLevel = levels[0];
55                 var previousLevel = levels[0];
56                 var skipOnce = false;
57                 var w = 0;
58                 for (var i = 1; i < length; i++)
59                 {
60                     if (_equalityComparer.Equal
61                         s(currentLevel,
62                         levels[i]))
63                     {
64                         levelRepeat++;
65                         skipOnce = false;
66                         if (levelRepeat == 2)
67                         {
68                             sequence[w] =
69                                 links.GetOrCreate
70                                 (sequence[i -
71                                     1],
72                                     sequence[i]);
73                             var newLevel = i >=
74                                 length - 1 ?
75                                 GetPreviousLow
76                                 erThanCurrent
77                                 (currentLevel,
78                                     previousLev
79                                     el,
80                                     currentLeve
81                                     l)
82                                 :
83                                 i < 2 ?
84                                 GetNextLowerTha
85                                 nCurrentOrC
86                                 urrent(curr
87                                     entLevel,
88                                     levels[i +
89                                     1]) :
90                                 GetGreatestNeig
91                                 bourLowerTh
92                                 anCurrentOr
93                                 Current(pre
94                                     viousLevel,
95                                     currentLeve
96                                     l, levels[i
97                                     + 1]);
98                             levels[w] =
99                                 newLevel;
100                             previousLevel =
101                                 currentLevel;
102                             w++;
103                             levelRepeat = 0;
104                             skipOnce = true;
105                         }
106                     }
107                     else if (i == length -
108                         1)
109                     {
110                         sequence[w] =
111                             sequence[i];
112                         levels[w] =
113                             levels[i];
114                         w++;
115                     }
116                 }
117             }
118             else
119             {
120                 currentLevel =
121                     levels[i];
122                 levelRepeat = 1;
123                 if (skipOnce)
124                 {
125                     skipOnce = false;
126                 }
127                 else
128                 {
129                     sequence[w] =
130                         sequence[i - 1];

```



```

76         levels[w] =
77             ↳ levels[i - 1];
78         previousLevel =
79             ↳ levels[w];
80         w++;
81     }
82     if (i == length - 1)
83     {
84         sequence[w] =
85             ↳ sequence[i];
86         levels[w] =
87             ↳ levels[i];
88         w++;
89     }
90     length = w;
91     return
92     ↳ links.GetOrCreate(sequence[0],
93     ↳ sequence[1]);
94 }
95 private static TLink GetGreatestNeighbou
96     ↳ rLowerThanCurrentOrCurrent(TLink
97     ↳ previous, TLink current, TLink next)
98 {
99     return _comparer.Compare(previous,
100     ↳ next) > 0
101     ? _comparer.Compare(previous,
102     ↳ current) < 0 ? previous :
103     ↳ current
104     : _comparer.Compare(next,
105     ↳ current) < 0 ? next :
106     ↳ current;
107 }
108 private static TLink GetNextLowerThanCu
109     ↳ rrentOrCurrent(TLink current, TLink
110     ↳ next) => _comparer.Compare(next,
111     ↳ current) < 0 ? next : current;
112 private static TLink GetPreviousLowerTh
113     ↳ anCurrentOrCurrent(TLink previous,
114     ↳ TLink current) =>
115     ↳ _comparer.Compare(previous,
116     ↳ current) < 0 ? previous : current;
117 }
118 }
119 }

```

./Sequences/Converters/SequenceToItsLocalElementLevelsConverter.cs

```

1  using System.Collections.Generic;
2  using Platform.Interfaces;
3
4  namespace
5  ↳ Platform.Data.Doublets.Sequences.Converters
6  {
7      public class SequenceToItsLocalElementLevel
8          ↳ sConverter<TLink> :
9          ↳ LinksOperatorBase<TLink>,
10          ↳ IConverter<IList<TLink>>
11      {
12          private static readonly Comparer<TLink>
13          ↳ _comparer = Comparer<TLink>.Default;
14          private readonly
15          ↳ IConverter<Doublet<TLink>, TLink>
16          ↳ _linkToItsFrequencyToNumberConveter;
17          public SequenceToItsLocalElementLevelsC
18          ↳ onverter(ILinks<TLink> links,
19          ↳ IConverter<Doublet<TLink>, TLink>
20          ↳ linkToItsFrequencyToNumberConveter)
21          ↳ : base(links) =>
22          ↳ _linkToItsFrequencyToNumberConveter
23          ↳ =
24          ↳ linkToItsFrequencyToNumberConveter;
25          public IList<TLink>
26          ↳ Convert(IList<TLink> sequence)
27          {
28              var levels = new
29                  ↳ TLink[sequence.Count];
30              levels[0] =
31                  ↳ GetFrequencyNumber(sequence[0],
32                  ↳ sequence[1]);
33              for (var i = 1; i < sequence.Count
34                  ↳ - 1; i++)
35              {

```

```

17         var previous = GetFrequencyNumb
18         ↳ er(sequence[i - 1],
19         ↳ sequence[i]);
20         var next = GetFrequencyNumber(s
21         ↳ equence[i], sequence[i +
22         ↳ 1]);
23         levels[i] =
24         ↳ _comparer.Compare(previous,
25         ↳ next) > 0 ? previous : next;
26     }
27     levels[levels.Length - 1] =
28     ↳ GetFrequencyNumber(sequence[seq
29     ↳ uence.Count - 2],
30     ↳ sequence[sequence.Count - 1]);
31     return levels;
32 }
33
34 public TLink GetFrequencyNumber(TLink
35     ↳ source, TLink target) =>
36     ↳ _linkToItsFrequencyToNumberConveter
37     ↳ .Convert(new Doublet<TLink>(source,
38     ↳ target));
39 }
40 }

```

./Sequences/CreteriaMatchers/DefaultSequenceElementCreteria

```

1  using Platform.Interfaces;
2
3  namespace Platform.Data.Doublets.Sequences.Cret
4  ↳ eriaMatchers
5  {
6      public class DefaultSequenceElementCreteria
7          ↳ Matcher<TLink> :
8          ↳ LinksOperatorBase<TLink>,
9          ↳ ICreteriaMatcher<TLink>
10     {
11         public DefaultSequenceElementCreteriaMa
12         ↳ tcher(ILinks<TLink> links) :
13         ↳ base(links) { }
14         public bool IsMatched(TLink argument)
15         ↳ => Links.IsPartialPoint(argument);
16     }
17 }

```

./Sequences/CreteriaMatchers/MarkedSequenceCreteriaMatcher

```

1  using System.Collections.Generic;
2  using Platform.Interfaces;
3
4  namespace Platform.Data.Doublets.Sequences.Cret
5  ↳ eriaMatchers
6  {
7      public class
8          ↳ MarkedSequenceCreteriaMatcher<TLink> :
9          ↳ ICreteriaMatcher<TLink>
10     {
11         private static readonly
12         ↳ EqualityComparer<TLink>
13         ↳ _equalityComparer =
14         ↳ EqualityComparer<TLink>.Default;
15         private readonly ILinks<TLink> _links;
16         private readonly TLink
17         ↳ _sequenceMarkerLink;
18         public MarkedSequenceCreteriaMatcher(IL
19         ↳ inks<TLink> links, TLink
20         ↳ sequenceMarkerLink)
21     {
22         _links = links;
23         _sequenceMarkerLink =
24         ↳ sequenceMarkerLink;
25     }
26     public bool IsMatched(TLink
27     ↳ sequenceCandidate)
28     ↳ => _equalityComparer.Equals(_links.
29     ↳ GetSource(sequenceCandidate),
30     ↳ _sequenceMarkerLink)
31     || !_equalityComparer.Equals(_links
32     ↳ .SearchOrDefault(_sequenceMarke
33     ↳ rLink, sequenceCandidate),
34     ↳ _links.Constants.Null);
35 }

```

./Sequences/DefaultSequenceAppender.cs

```
1 using System.Collections.Generic;
2 using Platform.Collections.Stacks;
3 using Platform.Data.Doublets.Sequences.HeightPr
    ↳ oviders;
4 using Platform.Data.Sequences;
5
6 namespace Platform.Data.Doublets.Sequences
7 {
8     public class DefaultSequenceAppender<TLink>
9         ↳ : LinksOperatorBase<TLink>,
10         ↳ ISequenceAppender<TLink>
11     {
12         private static readonly
13             ↳ EqualityComparer<TLink>
14             ↳ _equalityComparer =
15             ↳ EqualityComparer<TLink>.Default;
16
17         private readonly IStack<TLink> _stack;
18         private readonly
19             ↳ ISequenceHeightProvider<TLink>
20             ↳ _heightProvider;
21
22         public DefaultSequenceAppender(ILinks<T
23             ↳ Link> links, IStack<TLink> stack,
24             ↳ ISequenceHeightProvider<TLink>
25             ↳ heightProvider)
26             ↳ : base(links)
27         {
28             _stack = stack;
29             _heightProvider = heightProvider;
30         }
31
32         public TLink Append(TLink sequence,
33             ↳ TLink appendant)
34         {
35             var cursor = sequence;
36             while (!_equalityComparer.Equals(_h
37                 ↳ eightProvider.Get(cursor),
38                 ↳ default))
39             {
40                 var source =
41                     ↳ Links.GetSource(cursor);
42                 var target =
43                     ↳ Links.GetTarget(cursor);
44                 if (_equalityComparer.Equals(_h
45                     ↳ eightProvider.Get(source),
46                     ↳ _heightProvider.Get(target))
47                 {
48                     break;
49                 }
50                 else
51                 {
52                     _stack.Push(source);
53                     cursor = target;
54                 }
55             }
56             var left = cursor;
57             var right = appendant;
58             while (!_equalityComparer.Equals(cu
59                 ↳ rsor = _stack.Pop(),
60                 ↳ Links.Constants.Null))
61             {
62                 right = Links.GetOrCreate(left,
63                     ↳ right);
64                 left = cursor;
65             }
66             return Links.GetOrCreate(left,
67                 ↳ right);
68         }
69     }
70 }
```

./Sequences/DuplicateSegmentsCounter.cs

```
1 using System.Collections.Generic;
2 using System.Linq;
3 using Platform.Interfaces;
4
5 namespace Platform.Data.Doublets.Sequences
6 {
7     public class
8         ↳ DuplicateSegmentsCounter<TLink> :
9         ↳ ICounter<int>
10     {
```

```
9         private readonly IProvider<IList<KeyVal
10             ↳ uePair<IList<TLink>,
11             ↳ IList<TLink>>>>
12             ↳ _duplicateFragmentsProvider;
13     public DuplicateSegmentsCounter(IProvid
14         ↳ er<IList<KeyValuePair<IList<TLink>,
15         ↳ IList<TLink>>>>
16         ↳ duplicateFragmentsProvider) =>
17         ↳ _duplicateFragmentsProvider =
18         ↳ duplicateFragmentsProvider;
19     public int Count() => _duplicateFragmen
20         ↳ tsProvider.Get().Sum(x =>
21         ↳ x.Value.Count);
22     }
23 }
```

./Sequences/DuplicateSegmentsProvider.cs

```
1 using System;
2 using System.Linq;
3 using System.Collections.Generic;
4 using Platform.Interfaces;
5 using Platform.Collections;
6 using Platform.Collections.Lists;
7 using Platform.Collections.Segments;
8 using Platform.Collections.Segments.Walkers;
9 using Platform.Helpers;
10 using Platform.Helpers.Singletons;
11 using Platform.Numbers;
12 using Platform.Data.Sequences;
13
14 namespace Platform.Data.Doublets.Sequences
15 {
16     public class
17         ↳ DuplicateSegmentsProvider<TLink> :
18         ↳ DictionaryBasedDuplicateSegmentsWalkerB
19         ↳ ase<TLink>,
20         ↳ IProvider<IList<KeyValuePair<IList<TLin
21         ↳ k>,
22         ↳ IList<TLink>>>>
23     {
24         private readonly ILinks<TLink> _links;
25         private readonly ISequences<TLink>
26             ↳ _sequences;
27         private
28             ↳ HashSet<KeyValuePair<IList<TLink>,
29             ↳ IList<TLink>>> _groups;
30         private BitString _visited;
31
32         private class ItemEquilityComparer :
33             ↳ IEquityComparer<KeyValuePair<ILis
34             ↳ t<TLink>,
35             ↳ IList<TLink>>>
36         {
37             private readonly
38                 ↳ IListEqualityComparer<TLink>
39                 ↳ _listComparer;
40             public ItemEquilityComparer() =>
41                 ↳ _listComparer = Default<IListEq
42                 ↳ ualityComparer<TLink>>.Instance;
43             public bool Equals(KeyValuePair<ILI
44                 ↳ st<TLink>, IList<TLink>> left,
45                 ↳ KeyValuePair<IList<TLink>,
46                 ↳ IList<TLink>> right) =>
47                 ↳ _listComparer.Equals(left.Key,
48                 ↳ right.Key) && _listComparer.Equ
49                 ↳ als(left.Value,
50                 ↳ right.Value);
51             public int GetHashCode(KeyValuePair
52                 ↳ <IList<TLink>, IList<TLink>>
53                 ↳ pair) =>
54                 ↳ HashHelpers.Generate(_listCompa
55                 ↳ rer.GetHashCode(pair.Key),
56                 ↳ _listComparer.GetHashCode(pair.
57                 ↳ Value));
58         }
59
60         private class ItemComparer : IComparer<
61             ↳ KeyValuePair<IList<TLink>,
62             ↳ IList<TLink>>>
63         {
64             private readonly
65                 ↳ IListComparer<TLink>
66                 ↳ _listComparer;
67             public ItemComparer() =>
68                 ↳ _listComparer = Default<IListCo
69                 ↳ mparer<TLink>>.Instance;
70         }
71     }
```

```

37     public int Compare(KeyValuePair<ILink> left,
38         ↪ KeyValuePair<ILink> right)
39     {
40         var intermediateResult = _listC
41         ↪ omparer.Compare(left.Key,
42         ↪ right.Key);
43         if (intermediateResult == 0)
44         {
45             intermediateResult =
46             ↪ _listComparer.Compare(l
47             ↪ eft.Value,
48             ↪ right.Value);
49         }
50         return intermediateResult;
51     }
52 }
53
54 public DuplicateSegmentsProvider(ILinks
55     ↪ <TLink> links, ISequences<TLink>
56     ↪ sequences)
57     : base(minimumStringSegmentLength:
58         ↪ 2)
59 {
60     _links = links;
61     _sequences = sequences;
62 }
63
64 public IList<KeyValuePair<IList<TLink>,
65     ↪ IList<TLink>>> Get()
66 {
67     _groups = new HashSet<KeyValuePair<
68     ↪ IList<TLink>,
69     ↪ IList<TLink>>>(Default<ItemEqui
70     ↪ lityComparer>.Instance);
71     var count = _links.Count();
72     _visited = new BitString((long)(Int
73     ↪ eger<TLink>)count +
74     ↪ 1);
75     _links.Each(link =>
76     {
77         var linkIndex =
78         ↪ _links.GetIndex(link);
79         var linkBitIndex = (long)(Integ
80         ↪ er<TLink>)linkIndex;
81         if (!_visited.Get(linkBitIndex))
82         {
83             var sequenceElements = new
84             ↪ List<TLink>();
85             _sequences.EachPart(sequenc
86             ↪ eElements.AddAndReturnT
87             ↪ rue,
88             ↪ linkIndex);
89             if (sequenceElements.Count
90             ↪ > 2)
91             {
92                 WalkAll(sequenceElement
93                 ↪ s);
94             }
95         }
96         return
97         ↪ _links.Constants.Continue;
98     });
99     var resultList = _groups.ToList();
100     var comparer =
101     ↪ Default<ItemComparer>.Instance;
102     resultList.Sort(comparer);
103
104     #if DEBUG
105     foreach (var item in resultList)
106     {
107         PrintDuplicates(item);
108     }
109     #endif
110     return resultList;
111 }
112
113 protected override Segment<TLink>
114     CreateSegment(IList<TLink>
115     ↪ elements, int offset, int length)
116     => new Segment<TLink>(elements,
117     ↪ offset, length);
118
119 protected override void
120     OnDuplicateFound(Segment<TLink>
121     ↪ segment)
122 {
123     var duplicates = CollectDuplicatesF
124     ↪ orSegment(segment);
125     if (duplicates.Count > 1)
126     {
127         _groups.Add(new
128         ↪ KeyValuePair<IList<TLink>,
129         ↪ IList<TLink>>(segment.ToArr
130         ↪ ay(),
131         ↪ duplicates));
132     }
133 }
134
135 private List<TLink> CollectDuplicatesFo
136     ↪ rSegment(Segment<TLink>
137     ↪ segment)
138 {
139     var duplicates = new List<TLink>();
140     var readAsElement = new
141     ↪ HashSet<TLink>();
142     _sequences.Each(sequence =>
143     {
144         duplicates.Add(sequence);
145         readAsElement.Add(sequence);
146         return true; // Continue
147     }, segment);
148     if (duplicates.Any(x => _visited.Ge
149     ↪ t((Integer<TLink>)x)))
150     {
151         return new List<TLink>();
152     }
153     foreach (var duplicate in
154     ↪ duplicates)
155     {
156         var duplicateBitIndex = (long)(
157         ↪ Integer<TLink>)duplicate;
158         _visited.Set(duplicateBitIndex);
159     }
160     if (_sequences is Sequences
161     ↪ sequencesExperiments)
162     {
163         var partiallyMatched =
164         ↪ sequencesExperiments.GetAll
165         ↪ PartiallyMatchingSequences4
166         ↪ ((HashSet<ulong>)(object)re
167         ↪ adAsElement,
168         ↪ (IList<ulong>)segment);
169         foreach (var
170         ↪ partiallyMatchedSequence in
171         ↪ partiallyMatched)
172         {
173             TLink sequenceIndex =
174             ↪ (Integer<TLink>)partial
175             ↪ lyMatchedSequence;
176             duplicates.Add(sequenceInde
177             ↪ x);
178         }
179     }
180     duplicates.Sort();
181     return duplicates;
182 }
183
184 private void PrintDuplicates(KeyValuePa
185     ↪ ir<IList<TLink>, IList<TLink>>
186     ↪ duplicatesItem)
187 {
188     if (!(_links is ILinks<ulong>
189     ↪ ulongLinks))
190     {
191         return;
192     }
193     var duplicatesKey =
194     ↪ duplicatesItem.Key;
195     var keyString =
196     ↪ UnicodeMap.FromLinksToString((I
197     ↪ List<ulong>)duplicatesKey);
198     Console.WriteLine($"{keyString}
199     ↪ ({string.Join(", ",
200     ↪ duplicatesKey)})");
201     var duplicatesList =
202     ↪ duplicatesItem.Value;
203     for (int i = 0; i <
204     ↪ duplicatesList.Count; i++)

```

```

141     {
142         ulong sequenceIndex = (Integer<
            ↳ TLink>)duplicatesList[i];
143         var formattedSequenceStructure =
            ↳ ulongLinks.FormatStructure(
            ↳ sequenceIndex, x =>
            ↳ Point<ulong>.IsPartialPoint
            ↳ (x), (sb, link) =>
            ↳ UnicodeMap.IsCharLink(link.
            ↳ Index) ?
            ↳ sb.Append(UnicodeMap.FromLi
            ↳ nkToChar(link.Index)) :
            ↳ sb.Append(link.Index));
144         Console.WriteLine(formattedSeque
            ↳ nceStructure);
145         var sequenceString =
            ↳ UnicodeMap.FromSequenceLink
            ↳ ToString(sequenceIndex,
            ↳ ulongLinks);
146         Console.WriteLine(sequenceStrin
            ↳ g);
147     }
148     Console.WriteLine();
149 }
150 }
151 }

```

./Sequences/Frequencies/Cache/FrequenciesCacheBasedLinkFrequencyIncrementer.cs

```

1 using System.Collections.Generic;
2 using Platform.Interfaces;
3
4 namespace Platform.Data.Doublets.Sequences.Freq
    ↳ uencies.Cache
5 {
6     public class FrequenciesCacheBasedLinkFreque
            ↳ encyIncrementer<TLink> :
            ↳ IIncrementer<IList<TLink>>
7     {
8         private readonly
            ↳ LinkFrequenciesCache<TLink> _cache;
9
10        public FrequenciesCacheBasedLinkFrequen
            ↳ cyIncrementer(LinkFrequenciesCache<
            ↳ TLink> cache) => _cache =
            ↳ cache;
11
12        /// <remarks>Sequence itseft is not
            ↳ changed, only frequency of its
            ↳ doublets is incremented.</remarks>
13        public IList<TLink>
            ↳ Increment(IList<TLink> sequence)
            ↳ {
14            _cache.IncrementFrequencies(sequenc
            ↳ e);
15            return sequence;
16        }
17    }
18 }
19 }

```

./Sequences/Frequencies/Cache/FrequenciesCacheBasedLinkToItsFrequencyNumberConverter.cs

```

1 using Platform.Interfaces;
2
3 namespace Platform.Data.Doublets.Sequences.Freq
    ↳ uencies.Cache
4 {
5     public class FrequenciesCacheBasedLinkToItsF
            ↳ requencyNumberConverter<TLink> :
            ↳ IConverter<Doublet<TLink>, TLink>
6     {
7         private readonly
            ↳ LinkFrequenciesCache<TLink> _cache;
8         public FrequenciesCacheBasedLinkToItsFr
            ↳ equencyNumberConverter(LinkFrequenc
            ↳ iesCache<TLink> cache) => _cache =
            ↳ cache;
9         public TLink Convert(Doublet<TLink>
            ↳ source) => _cache.GetFrequency(ref
            ↳ source).Frequency;
10    }
11 }

```

./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 namespace Platform.Data.Doublets.Sequences.Freq
    ↳ uencies.Cache
8 {
9     /// <remarks>
10    /// Can be used to operate with many
            ↳ CompressingConverters (to keep global
            ↳ frequencies data between them).
11    /// TODO: Extract interface to implement
            ↳ frequencies storage inside Links storage
12    /// </remarks>
13    public class LinkFrequenciesCache<TLink> :
            ↳ LinksOperatorBase<TLink>
14    {
15        private static readonly
            ↳ EqualityComparer<TLink>
            ↳ equalityComparer =
            ↳ EqualityComparer<TLink>.Default;
16        private static readonly Comparer<TLink>
            ↳ _comparer = Comparer<TLink>.Default;
17
18        private readonly
            ↳ Dictionary<Doublet<TLink>,
            ↳ LinkFrequency<TLink>>
            ↳ _doubletsCache;
19        private readonly ICounter<TLink, TLink>
            ↳ _frequencyCounter;
20    }
21    public

```

```

21    public
            ↳ LinkFrequenciesCache(ILinks<TLink>
            ↳ links, ICounter<TLink, TLink>
            ↳ frequencyCounter)
            ↳ : base(links)
22    {
23        _doubletsCache = new
            ↳ Dictionary<Doublet<TLink>,
            ↳ LinkFrequency<TLink>>(4096,
            ↳ DoubletComparer<TLink>.Default);
24        _frequencyCounter =
            ↳ frequencyCounter;
25    }
26
27    [MethodImpl(MethodImplOptions.Aggressive
            ↳ eInlining)]
28    public LinkFrequency<TLink>
            ↳ GetFrequency(TLink source, TLink
            ↳ target)
29    {
30        var doublet = new
            ↳ Doublet<TLink>(source, target);
31        return GetFrequency(ref doublet);
32    }
33
34    [MethodImpl(MethodImplOptions.Aggressive
            ↳ eInlining)]
35    public LinkFrequency<TLink>
            ↳ GetFrequency(ref Doublet<TLink>
            ↳ doublet)
36    {
37        TryGetValue(doublet,
            ↳ out LinkFrequency<TLink> data);
38        return data;
39    }
40
41    public void
            ↳ IncrementFrequencies(IList<TLink>
            ↳ sequence)
42    {
43        for (var i = 1; i < sequence.Count;
            ↳ i++)
44        {
45            IncrementFrequency(sequence[i -
            ↳ 1], sequence[i]);
46        }
47    }
48
49    [MethodImpl(MethodImplOptions.Aggressive
            ↳ eInlining)]
50    public LinkFrequency<TLink>
            ↳ IncrementFrequency(TLink source,
            ↳ TLink target)
51    {
52        var doublet = new
            ↳ Doublet<TLink>(source, target);
53    }

```

```

54         return IncrementFrequency(ref
           ↳ doublet);
55     }
56
57     public void
           ↳ PrintFrequencies(IList<TLink>
           ↳ sequence)
58     {
59         for (var i = 1; i < sequence.Count;
           ↳ i++)
60         {
61             PrintFrequency(sequence[i - 1],
           ↳ sequence[i]);
62         }
63     }
64
65     public void PrintFrequency(TLink
           ↳ source, TLink target)
66     {
67         var number = GetFrequency(source,
           ↳ target).Frequency;
68         Console.WriteLine("{0},{1} -
           ↳ {2}", source, target, number);
69     }
70
71     [MethodImpl(MethodImplOptions.Aggressive
           ↳ eInlining)]
72     public LinkFrequency<TLink>
           ↳ IncrementFrequency(ref
           ↳ Doublet<TLink> doublet)
73     {
74         if (_doubletsCache.TryGetValue(doub
           ↳ let, out LinkFrequency<TLink>
           ↳ data))
75         {
76             data.IncrementFrequency();
77         }
78         else
79         {
80             var link = Links.SearchOrDefaul
           ↳ t(doublet.Source,
           ↳ doublet.Target);
81             data = new LinkFrequency<TLink>
           ↳ (Integer<TLink>.One,
           ↳ link);
82             if (!_equalityComparer.Equals(l
           ↳ ink,
           ↳ default))
83             {
84                 data.Frequency =
           ↳ ArithmeticHelpers.Add(d
           ↳ ata.Frequency,
           ↳ _frequencyCounter.Count
           ↳ (link));
85             }
86             _doubletsCache.Add(doublet,
           ↳ data);
87         }
88         return data;
89     }
90
91     public void ValidateFrequencies()
92     {
93         foreach (var entry in
           ↳ _doubletsCache)
94         {
95             var value = entry.Value;
96             var linkIndex = value.Link;
97             if (!_equalityComparer.Equals(l
           ↳ inkIndex,
           ↳ default))
98             {
99                 var frequency =
           ↳ value.Frequency;
100                 var count = _frequencyCount
           ↳ er.Count(linkIndex);
101                 // TODO: Why `frequency`
           ↳ always greater than
           ↳ `count` by 1?
102
103                 if ((_comparer.Compare(fre
           ↳ quency, count) > 0) &&
           ↳ (_comparer.Compare(Arit
           ↳ hmeticHelpers.Subtract(
           ↳ frequency, count),
           ↳ Integer<TLink>.One) >
           ↳ 0))
           ↳ || ((_comparer.Compare(cou
           ↳ nt, frequency) > 0) &&
           ↳ (_comparer.Compare(Ari
           ↳ thmeticHelpers.Subtrac
           ↳ t(count, frequency),
           ↳ Integer<TLink>.One) >
           ↳ 0)))
104                 {
105                     throw new
           ↳ InvalidOperationException("Frequencies
           ↳ validation
           ↳ failed.");
106                 }
107             }
108             //else
109             //{
110             //    if (value.Frequency > 0)
111             //    {
112             //        var frequency =
           ↳ value.Frequency;
113             //        linkIndex = _createLi
           ↳ nk(entry.Key.Source,
           ↳ entry.Key.Target);
114             //        var count = _countLin
           ↳ kFrequency(linkIndex);
115             //        if ((frequency >
           ↳ count && frequency - count
           ↳ > 1) || (count > frequency
           ↳ && count - frequency > 1))
116             //        {
117             //            throw new
           ↳ Exception("Frequencies
           ↳ validation failed.");
118             //        }
119             //    }
120             //}
121         }
122     }
123 }

```

./Sequences/Frequencies/Cache/LinkFrequency.cs

```

1 using System.Runtime.CompilerServices;
2 using Platform.Numbers;
3
4 namespace Platform.Data.Doublets.Sequences.Freq
           ↳ uencies.Cache
5     {
6         public class LinkFrequency<TLink>
7         {
8             public TLink Frequency { get; set; }
9             public TLink Link { get; set; }
10
11             public LinkFrequency(TLink frequency,
           ↳ TLink link)
12             {
13                 Frequency = frequency;
14                 Link = link;
15             }
16
17             public LinkFrequency() { }
18
19             [MethodImpl(MethodImplOptions.Aggressive
           ↳ eInlining)]
20             public void IncrementFrequency() =>
           ↳ Frequency = ArithmeticHelpers<TLink>
           ↳ .Increment(Frequency);
21
22             [MethodImpl(MethodImplOptions.Aggressive
           ↳ eInlining)]
23             public void DecrementFrequency() =>
           ↳ Frequency = ArithmeticHelpers<TLink>
           ↳ .Decrement(Frequency);
24
25             public override string ToString() =>
           ↳ $"F: {Frequency}, L: {Link}";
26         }

```

```
27 }
```

```
./Sequences/Frequencies/Counters/MarkedSequenceSymbolFrequencyOneOffCounter.cs
```

```
1 using Platform.Interfaces;
2
3 namespace Platform.Data.Doublets.Sequences.Frequencies.Counters
4 {
5     public class MarkedSequenceSymbolFrequencyOneOffCounter<TLink> :
        SequenceSymbolFrequencyOneOffCounter<TLink>
6     {
7         private readonly
            ICriteriaMatcher<TLink>
            _markedSequenceMatcher;
8
9         public MarkedSequenceSymbolFrequencyOneOffCounter(ILinks<TLink> links,
            ICriteriaMatcher<TLink>
            markedSequenceMatcher, TLink
            sequenceLink, TLink symbol)
10         : base(links, sequenceLink, symbol)
11         => _markedSequenceMatcher =
            markedSequenceMatcher;
12
13         public override TLink Count()
14         {
15             if (!_markedSequenceMatcher.IsMatch(
                _sequenceLink))
16             {
17                 return default;
18             }
19             return base.Count();
20         }
21     }
22 }
```

```
./Sequences/Frequencies/Counters/SequenceSymbolFrequencyOneOffCounter.cs
```

```
1 using System.Collections.Generic;
2 using Platform.Interfaces;
3 using Platform.Numbers;
4 using Platform.Data.Sequences;
5
6 namespace Platform.Data.Doublets.Sequences.Frequencies.Counters
7 {
8     public class SequenceSymbolFrequencyOneOffCounter<TLink> :
        ICounter<TLink>
9     {
10         private static readonly
            EqualityComparer<TLink>
            _equalityComparer =
            EqualityComparer<TLink>.Default;
11         private static readonly Comparer<TLink>
            _comparer = Comparer<TLink>.Default;
12
13         protected readonly ILinks<TLink> _links;
14         protected readonly TLink _sequenceLink;
15         protected readonly TLink _symbol;
16         protected TLink _total;
17
18         public SequenceSymbolFrequencyOneOffCounter(ILinks<TLink> links, TLink
            sequenceLink, TLink symbol)
19         {
20             _links = links;
21             _sequenceLink = sequenceLink;
22             _symbol = symbol;
23             _total = default;
24         }
25
26         public virtual TLink Count()
27         {
28             if (_comparer.Compare(_total,
                default) > 0)
29             {
30                 return _total;
31             }
32             StopableSequenceWalker.WalkRight(_sequenceLink, _links.GetSource,
                _links.GetTarget, IsElement,
                VisitElement);
33             return _total;
34         }
35     }
```

```
36 private bool IsElement(TLink x) =>
    _equalityComparer.Equals(x,
        symbol) ||
    _links.IsPartialPoint(x); // TODO:
    Use SequenceElementCriteriaMatcher
    instead of IsPartialPoint
37
38 private bool VisitElement(TLink element)
39 {
40     if (_equalityComparer.Equals(element,
        _symbol))
41     {
42         _total = ArithmeticHelpers.Increment(_total);
43     }
44     return true;
45 }
46 }
47 }
```

```
./Sequences/Frequencies/Counters/TotalMarkedSequenceSymbolFrequencyOneOffCounter.cs
```

```
1 using Platform.Interfaces;
2
3 namespace Platform.Data.Doublets.Sequences.Frequencies.Counters
4 {
5     public class TotalMarkedSequenceSymbolFrequencyOneOffCounter<TLink> : ICounter<TLink>
6     {
7         private readonly ILinks<TLink> _links;
8         private readonly
            ICriteriaMatcher<TLink>
            _markedSequenceMatcher;
9
10        public TotalMarkedSequenceSymbolFrequencyOneOffCounter(ILinks<TLink> links,
            ICriteriaMatcher<TLink>
            markedSequenceMatcher)
11        {
12            _links = links;
13            _markedSequenceMatcher =
                markedSequenceMatcher;
14        }
15
16        public TLink Count(TLink argument) =>
            new TotalMarkedSequenceSymbolFrequencyOneOffCounter<TLink>(_links,
                _markedSequenceMatcher,
                argument).Count();
17    }
18 }
```

```
./Sequences/Frequencies/Counters/TotalMarkedSequenceSymbolFrequencyOneOffCounter.cs
```

```
1 using Platform.Interfaces;
2 using Platform.Numbers;
3
4 namespace Platform.Data.Doublets.Sequences.Frequencies.Counters
5 {
6     public class TotalMarkedSequenceSymbolFrequencyOneOffCounter<TLink> :
        TotalSequenceSymbolFrequencyOneOffCounter<TLink>
7     {
8         private readonly
            ICriteriaMatcher<TLink>
            _markedSequenceMatcher;
9
10        public TotalMarkedSequenceSymbolFrequencyOneOffCounter(ILinks<TLink>
            links, ICriteriaMatcher<TLink>
            markedSequenceMatcher, TLink
            symbol) : base(links, symbol)
11        => _markedSequenceMatcher =
            markedSequenceMatcher;
12
13        protected override void
            CountSequenceSymbolFrequency(TLink
            link)
14        {
15            var symbolFrequencyCounter = new
                MarkedSequenceSymbolFrequencyOneOffCounter<TLink>(_links,
                _markedSequenceMatcher, link,
                _symbol);
```

```

16         _total =
17             ↪ ArithmeticHelpers.Add(_total,
18             ↪ symbolFrequencyCounter.Count());
19     }
20 }
21 }

./Sequences/Frequencies/Counters/TotalSequenceSymbolFrequencyCounter.cs
1 using Platform.Interfaces;
2
3 namespace Platform.Data.Doublets.Sequences.Freq
4     ↪ uencies.Counters
5 {
6     public class TotalSequenceSymbolFrequencyCo
7         ↪ unter<TLink> : ICounter<TLink,
8         ↪ TLink>
9     {
10         private readonly ILinks<TLink> _links;
11         public TotalSequenceSymbolFrequencyCount
12             ↪ ter(ILinks<TLink> links) => _links
13             ↪ = links;
14         public TLink Count(TLink symbol) => new
15             ↪ TotalSequenceSymbolFrequencyOneOffC
16             ↪ ounter<TLink>(_links,
17             ↪ symbol).Count();
18     }
19 }

./Sequences/Frequencies/Counters/TotalSequenceSymbolFrequencyOneOffCounter.cs
1 using System.Collections.Generic;
2 using Platform.Interfaces;
3 using Platform.Numbers;
4
5 namespace Platform.Data.Doublets.Sequences.Freq
6     ↪ uencies.Counters
7 {
8     public class TotalSequenceSymbolFrequencyOn
9         ↪ eOffCounter<TLink> :
10         ↪ ICounter<TLink>
11     {
12         private static readonly
13             ↪ EqualityComparer<TLink>
14             ↪ _equalityComparer =
15             ↪ EqualityComparer<TLink>.Default;
16         private static readonly Comparer<TLink>
17             ↪ _comparer = Comparer<TLink>.Default;
18
19         protected readonly ILinks<TLink> _links;
20         protected readonly TLink _symbol;
21         protected readonly HashSet<TLink>
22             ↪ _visits;
23         protected TLink _total;
24
25         public TotalSequenceSymbolFrequencyOneO
26             ↪ ffCounter(ILinks<TLink> links,
27             ↪ TLink symbol)
28         {
29             _links = links;
30             _symbol = symbol;
31             _visits = new HashSet<TLink>();
32             _total = default;
33         }
34
35         public TLink Count()
36         {
37             if (_comparer.Compare(_total,
38                 ↪ default) > 0 || _visits.Count >
39                 ↪ 0)
40             {
41                 return _total;
42             }
43             CountCore(_symbol);
44             return _total;
45         }
46
47         private void CountCore(TLink link)
48         {
49             var any = _links.Constants.Any;
50             if (_equalityComparer.Equals(_links
51                 ↪ .Count(any, link),
52                 ↪ default))
53             {
54                 CountSequenceSymbolFrequency(li
55                 ↪ nk);
56             }
57             else
58             {
59             }
60         }
61     }
62 }

protected virtual void
    ↪ CountSequenceSymbolFrequency(TLink
    ↪ link)
{
    var symbolFrequencyCounter = new
        ↪ SequenceSymbolFrequencyOneOffCo
        ↪ unter<TLink>(_links, link,
        ↪ _symbol);
    _total =
        ↪ ArithmeticHelpers.Add(_total,
        ↪ symbolFrequencyCounter.Count());
}

private TLink
    ↪ EachElementHandler(IList<TLink>
    ↪ doublet)
{
    var constants = _links.Constants;
    var doubletIndex =
        ↪ doublet[constants.IndexPart];
    if (_visits.Add(doubletIndex))
    {
        CountCore(doubletIndex);
    }
    return constants.Continue;
}

./Sequences/HeightProviders/CachedSequenceHeightProvider.cs
1 using System.Collections.Generic;
2 using Platform.Interfaces;
3
4 namespace Platform.Data.Doublets.Sequences.Heig
5     ↪ htProviders
6 {
7     public class
8         ↪ CachedSequenceHeightProvider<TLink> :
9         ↪ LinksOperatorBase<TLink>,
10         ↪ ISequenceHeightProvider<TLink>
11     {
12         private static readonly
13             ↪ EqualityComparer<TLink>
14             ↪ _equalityComparer =
15             ↪ EqualityComparer<TLink>.Default;
16
17         private readonly TLink
18             ↪ _heightPropertyMarker;
19         private readonly
20             ↪ ISequenceHeightProvider<TLink>
21             ↪ _baseHeightProvider;
22         private readonly IConverter<TLink>
23             ↪ _addressToUnaryNumberConverter;
24         private readonly IConverter<TLink>
25             ↪ _unaryNumberToAddressConverter;
26         private readonly
27             ↪ IPropertyOperator<TLink, TLink,
28             ↪ TLink> _propertyOperator;
29
30         public CachedSequenceHeightProvider(
31             ↪ ILinks<TLink> links,
32             ↪ ISequenceHeightProvider<TLink>
33             ↪ baseHeightProvider,
34             ↪ IConverter<TLink>
35             ↪ addressToUnaryNumberConverter,
36             ↪ IConverter<TLink>
37             ↪ unaryNumberToAddressConverter,
38             ↪ TLink heightPropertyMarker,
39             ↪ IPropertyOperator<TLink, TLink,
40             ↪ TLink> propertyOperator)
41             : base(links)
42         {
43             _heightPropertyMarker =
44                 ↪ heightPropertyMarker;
45             _baseHeightProvider =
46                 ↪ baseHeightProvider;
47             _addressToUnaryNumberConverter =
48                 ↪ addressToUnaryNumberConverter;
49             _unaryNumberToAddressConverter =
50                 ↪ unaryNumberToAddressConverter;
51             _propertyOperator =
52                 ↪ propertyOperator;
53         }
54     }
55 }

```



```

31
32 public TLink Get(TLink sequence)
33 {
34     TLink height;
35     var heightValue = _propertyOperator
        ↪ .GetValue(sequence,
        ↪ _heightPropertyMarker);
36     if (_equalityComparer.Equals(height
        ↪ Value,
        ↪ default))
37     {
38         height = _baseHeightProvider.Ge
        ↪ t(sequence);
39         heightValue =
        ↪ _addressToUnaryNumberConver
        ↪ ter.Convert(height);
40         _propertyOperator.SetValue(sequ
        ↪ ence,
        ↪ _heightPropertyMarker,
        ↪ heightValue);
41     }
42     else
43     {
44         height =
        ↪ _unaryNumberToAddressConver
        ↪ ter.Convert(heightValue);
45     }
46     return height;
47 }
48 }
49 }

```

./Sequences/HeightProviders/DefaultSequenceRightHeightProvider.cs

```

1 using Platform.Interfaces;
2 using Platform.Numbers;
3
4 namespace Platform.Data.Doublets.Sequences.Heig
    ↪ htProviders
5 {
6     public class DefaultSequenceRightHeightProv
        ↪ ider<TLink> : LinksOperatorBase<TLink>,
        ↪ ISequenceHeightProvider<TLink>
7     {
8         private readonly
        ↪ ICriteriaMatcher<TLink>
        ↪ _elementMatcher;
9
10        public DefaultSequenceRightHeightProvid
        ↪ er(ILinks<TLink> links,
        ↪ ICriteriaMatcher<TLink>
        ↪ elementMatcher) : base(links) =>
        ↪ _elementMatcher = elementMatcher;
11
12        public TLink Get(TLink sequence)
13        {
14            var height = default(TLink);
15            var pairOrElement = sequence;
16            while (!_elementMatcher.IsMatched(p
        ↪ airOrElement))
17            {
18                pairOrElement = Links.GetTarget
        ↪ (pairOrElement);
19                height = ArithmeticHelpers.Incr
        ↪ ement(height);
20            }
21            return height;
22        }
23    }
24 }

```

./Sequences/HeightProviders/ISequenceHeightProvider.cs

```

1 using Platform.Interfaces;
2
3 namespace Platform.Data.Doublets.Sequences.Heig
    ↪ htProviders
4 {
5     public interface
        ↪ ISequenceHeightProvider<TLink> :
        ↪ IProvider<TLink, TLink>
6     {
7     }
8 }

```

./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.Threading.Synchronization;
8 using Platform.Helpers.Singletons;
9 using LinkIndex = System.UInt64;
10 using Platform.Data.Constants;
11 using Platform.Data.Sequences;
12 using Platform.Data.Doublets.Sequences.Walkers;
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 вместе, все числа вместе и
        ↪ т.п.
26     /// + использовать ровно сбалансированный
        ↪ вариант, чтобы уменьшать вложенность
        ↪ (глубину графа)
27     ///
        ↪ x*y - найти все связи между, в
        ↪ последовательностях любой формы, если
        ↪ не стоит ограничитель на то, что
        ↪ является последовательностью, а что нет,
28     /// то находятся любые структуры связей,
        ↪ которые содержат эти элементы именно в
        ↪ таком порядке.
29     ///
30     /// Рост последовательности слева и справа.
31     /// Поиск со звездочкой.
32     /// URL, PURL - реестр используемых во вне
        ↪ ссылок на ресурсы,
33     /// так же проблема может быть решена при
        ↪ реализации дистанционных триггеров.
34     /// Нужны ли уникальные указатели вообще?
35     /// Что если обращение к информации будет
        ↪ происходить через содержимое всегда?
36     ///
37     /// Писать тесты.
38     ///
39     ///
40     ///
41     /// Можно убрать зависимость от конкретной
        ↪ реализации Links,
42     /// на зависимость от абстрактного
        ↪ элемента, который может быть
        ↪ представлен несколькими 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 :
        ↪ ISequences<ulong> // IList<string>,
        ↪ IList<ulong[]> (после завершения
        ↪ реализации Sequences)
51     {
52         private static readonly
        ↪ LinksCombinedConstants<bool, ulong,
        ↪ long> _constants = Default<LinksCom
        ↪ binedConstants<bool, ulong,
        ↪ long>>.Instance;
53
54         /// <summary>Возвращает значение ulong,
        ↪ обозначающее любое количество
        ↪ связей.</summary>
55         public const ulong ZeroOrMany =
        ↪ ulong.MaxValue;
56
57         public SequencesOptions<ulong> Options;
58         public readonly
        ↪ SynchronizedLinks<ulong> Links;

```

```

59     public readonly ISynchronization Sync;
60
61     public
        ↳ Sequences(SynchronizedLinks<ulong>
        ↳ links)
62         : this(links, new
        ↳ SequencesOptions<ulong>())
63     {
64     }
65
66     public
        ↳ Sequences(SynchronizedLinks<ulong>
        ↳ links, SequencesOptions<ulong>
        ↳ options)
67     {
68         Links = links;
69         Sync = links.SyncRoot;
70         Options = options;
71
72         Options.ValidateOptions();
73         Options.InitOptions(Links);
74     }
75
76     public bool IsSequence(ulong sequence)
77     {
78         return Sync.ExecuteReadOperation(()
        ↳ =>
79         {
80             if (Options.UseSequenceMarker)
81             {
82                 return Options.MarkedSequen
        ↳ ceMatcher.IsMatched(seq
        ↳ uence);
83             }
84             return !Links.Unsync.IsPartialP
        ↳ oint(sequence);
85         });
86     }
87
88     [MethodImpl(MethodImplOptions.AggressiveI
        ↳ nlining)]
89     private ulong
        ↳ GetSequenceByElements(ulong
        ↳ sequence)
90     {
91         if (Options.UseSequenceMarker)
92         {
93             return Links.SearchOrDefault(Op
        ↳ tions.SequenceMarkerLink,
        ↳ sequence);
94         }
95         return sequence;
96     }
97
98     private ulong GetSequenceElements(ulong
        ↳ sequence)
99     {
100         if (Options.UseSequenceMarker)
101         {
102             var linkContents = new UInt64Li
        ↳ nk(Links.GetLink(sequence));
103             if (linkContents.Source ==
        ↳ Options.SequenceMarkerLink)
104             {
105                 return linkContents.Target;
106             }
107             if (linkContents.Target ==
        ↳ Options.SequenceMarkerLink)
108             {
109                 return linkContents.Source;
110             }
111         }
112         return sequence;
113     }
114
115     #region Count
116
117     public ulong Count(params ulong[]
        ↳ sequence)
118     {
119         if (sequence.Length == 0)
120         {
121             return
        ↳ Links.Count(_constants.Any,
        ↳ Options.SequenceMarkerLink,
        ↳ _constants.Any);

```

```

        }
        if (sequence.Length == 1) // Первая
        ↳ связь это адрес
        {
            if (sequence[0] ==
        ↳ _constants.Null)
            {
                return 0;
            }
            if (sequence[0] ==
        ↳ _constants.Any)
            {
                return Count();
            }
            if (Options.UseSequenceMarker)
            {
                return Links.Count(_constan
        ↳ ts.Any,
        ↳ Options.SequenceMarkerL
        ↳ ink,
        ↳ sequence[0]);
            }
            return
        ↳ Links.Exists(sequence[0]) ?
        ↳ 1UL : 0;
        }
        throw new NotImplementedException();
    }

    private ulong CountReferences(params
        ↳ ulong[] restrictions)
    {
        if (restrictions.Length == 0)
        {
            return 0;
        }
        if (restrictions.Length == 1) //
        ↳ Первая связь это адрес
        {
            if (restrictions[0] ==
        ↳ _constants.Null)
            {
                return 0;
            }
            if (Options.UseSequenceMarker)
            {
                var elementsLink =
        ↳ GetSequenceElements(res
        ↳ trictions[0]);
                var sequenceLink =
        ↳ GetSequenceByElements(e
        ↳ lementsLink);
                if (sequenceLink !=
        ↳ _constants.Null)
                {
                    return Links.Count(sequ
        ↳ enceLink) +
        ↳ Links.Count(element
        ↳ sLink) -
        ↳ 1;
                }
                return Links.Count(elements
        ↳ Link);
            }
            return Links.Count(restrictions
        ↳ [0]);
        }
        throw new NotImplementedException();
    }

    #endregion

    #region Create

    public ulong Create(params ulong[]
        ↳ sequence)
    {
        return
        ↳ Sync.ExecuteWriteOperation(() =>
        {
            if (sequence.IsNullOrEmpty())
            {
                return _constants.Null;
            }

```



```

301 private bool StepRight(Func<ulong,
    bool> handler, ulong left, ulong
    ↪ right) => Links.Unsync.Each(left,
    ↪ _constants.Any, rightStep =>
    ↪ TryStepRightUp(handler, right,
    ↪ rightStep));
302
303 private bool TryStepRightUp(Func<ulong,
    bool> handler, ulong right, ulong
    ↪ stepFrom)
304 {
305     var upStep = stepFrom;
306     var firstSource =
    ↪ Links.Unsync.GetTarget(upStep);
307     while (firstSource != right &&
    ↪ firstSource != upStep)
308     {
309         upStep = firstSource;
310         firstSource = Links.Unsync.GetS
    ↪ ource(upStep);
311     }
312     if (firstSource == right)
313     {
314         return handler(stepFrom);
315     }
316     return true;
317 }
318
319 private bool StepLeft(Func<ulong, bool>
    ↪ handler, ulong left, ulong right)
    ↪ =>
    ↪ Links.Unsync.Each(_constants.Any,
    ↪ right, leftStep =>
    ↪ TryStepLeftUp(handler, left,
    ↪ leftStep));
320
321 private bool TryStepLeftUp(Func<ulong,
    bool> handler, ulong left, ulong
    ↪ stepFrom)
322 {
323     var upStep = stepFrom;
324     var firstTarget =
    ↪ Links.Unsync.GetSource(upStep);
325     while (firstTarget != left &&
    ↪ firstTarget != upStep)
326     {
327         upStep = firstTarget;
328         firstTarget = Links.Unsync.GetT
    ↪ arget(upStep);
329     }
330     if (firstTarget == left)
331     {
332         return handler(stepFrom);
333     }
334     return true;
335 }
336
337 #endregion
338
339 #region Update
340
341 public ulong Update(ulong[] sequence,
    ↪ ulong[] newSequence)
342 {
343     if (sequence.IsNullOrEmpty() &&
    ↪ newSequence.IsNullOrEmpty())
344     {
345         return _constants.Null;
346     }
347     if (sequence.IsNullOrEmpty())
348     {
349         return Create(newSequence);
350     }
351     if (newSequence.IsNullOrEmpty())
352     {
353         Delete(sequence);
354         return _constants.Null;
355     }
356     return
    ↪ Sync.ExecuteWriteOperation(() =>
357     {
358         Links.EnsureEachLinkIsAnyOrExis
    ↪ ts(sequence);
359         Links.EnsureEachLinkExists(newS
    ↪ equence);
360         return UpdateCore(sequence,
    ↪ newSequence);
361     });
362 }
363
364 private ulong UpdateCore(ulong[]
    ↪ sequence, ulong[] newSequence)
365 {
366     ulong bestVariant;
367     if (Options.EnforceSingleSequenceVe
    ↪ rsionOnWriteBasedOnNew &&
    ↪ !sequence.EqualTo(newSequence))
368     {
369         bestVariant =
    ↪ CompactCore(newSequence);
370     }
371     else
372     {
373         bestVariant =
    ↪ CreateCore(newSequence);
374     }
375     // TODO: Check all options only
    ↪ ones before loop execution
    ↪ Возможно нужно две версии Each,
    ↪ возвращающий фактические
    ↪ последовательности и с маркером,
    ↪ или возможно даже возвращать и
    ↪ тот и тот вариант. С другой
    ↪ стороны все варианты можно
    ↪ получить имея только
    ↪ фактические последовательности.
376     foreach (var variant in
    ↪ Each(sequence))
377     {
378         if (variant != bestVariant)
379         {
380             UpdateOneCore(variant,
    ↪ bestVariant);
381         }
382     }
383     return bestVariant;
384 }
385
386 private void UpdateOneCore(ulong
    ↪ sequence, ulong newSequence)
387 {
388     if (Options.UseGarbageCollection)
389     {
390         var sequenceElements = GetSeque
    ↪ nceElements(sequence);
391         var sequenceElementsContents =
    ↪ new UInt64Link(Links.GetLin
    ↪ k(sequenceElements));
392         var sequenceLink = GetSequenceB
    ↪ yElements(sequenceElements);
393         var newSequenceElements = GetSe
    ↪ quenceElements(newSequence);
394         var newSequenceLink =
    ↪ GetSequenceByElements(newSe
    ↪ quenceElements);
395         if (Options.UseCascadeUpdate ||
    ↪ CountReferences(sequence)
    ↪ == 0)
396         {
397             if (sequenceLink !=
    ↪ _constants.Null)
398             {
399                 Links.Unsync.Merge(sequ
    ↪ enceLink,
400                 ↪ newSequenceLink);
401             }
402             Links.Unsync.Merge(sequence
    ↪ Elements,
403             ↪ newSequenceElements);
404         }
405         ClearGarbage(sequenceElementsCo
    ↪ ntents.Source);
406         ClearGarbage(sequenceElementsCo
    ↪ ntents.Target);
407     }
408     else
409     {
410         if (Options.UseSequenceMarker)
411         {

```

412	<code>var sequenceElements =</code>	462	<code>Links.Unsync.Delete(seq</code>
	<code>↳ GetSequenceElements(seq</code>		<code>↳ uenceLink);</code>
	<code>↳ uence);</code>	463	<code>}</code>
413	<code>var sequenceLink =</code>	464	<code>Links.Unsync.Delete(link);</code>
	<code>↳ GetSequenceByElements(s</code>	465	<code>}</code>
	<code>↳ equenceElements);</code>	466	<code>ClearGarbage(sequenceElementsCo</code>
414	<code>var newSequenceElements =</code>		<code>↳ ntents.Source);</code>
	<code>↳ GetSequenceElements(new</code>	467	<code>ClearGarbage(sequenceElementsCo</code>
	<code>↳ Sequence);</code>		<code>↳ ntents.Target);</code>
415	<code>var newSequenceLink =</code>	468	<code>}</code>
	<code>↳ GetSequenceByElements(n</code>	469	<code>else</code>
	<code>↳ ewSequenceElements);</code>	470	<code>{</code>
416	<code>if (Options.UseCascadeUpdat</code>	471	<code>if (Options.UseSequenceMarker)</code>
	<code>↳ e </code>	472	<code>{</code>
	<code>↳ CountReferences(sequenc</code>	473	<code>var sequenceElements = GetS</code>
	<code>↳ e) ==</code>		<code>↳ equenceElements(link);</code>
	<code>↳ 0)</code>	474	<code>var sequenceLink =</code>
	<code>{</code>		<code>↳ GetSequenceByElements(s</code>
417			<code>↳ equenceElements);</code>
418	<code>if (sequenceLink !=</code>	475	<code>if (Options.UseCascadeDelet</code>
	<code>↳ _constants.Null)</code>		<code>↳ e </code>
419	<code>{</code>		<code>↳ CountReferences(link)</code>
420	<code>Links.Unsync.Merge(</code>		<code>↳ == 0)</code>
	<code>↳ sequenceLink,</code>	476	<code>{</code>
	<code>↳ newSequenceLink</code>	477	<code>if (sequenceLink !=</code>
	<code>↳);</code>		<code>↳ _constants.Null)</code>
421	<code>}</code>	478	<code>{</code>
422	<code>Links.Unsync.Merge(sequ</code>	479	<code>Links.Unsync.Delete</code>
	<code>↳ enceElements,</code>		<code>↳ (sequenceLink);</code>
	<code>↳ newSequenceElements</code>	480	<code>}</code>
	<code>↳);</code>	481	<code>Links.Unsync.Delete(lin</code>
	<code>}</code>		<code>↳ k);</code>
423	<code>}</code>		<code>}</code>
424	<code>}</code>	482	<code>}</code>
425	<code>else</code>	483	<code>}</code>
426	<code>{</code>	484	<code>else</code>
427	<code>if (Options.UseCascadeUpdat</code>	485	<code>{</code>
	<code>↳ e </code>	486	<code>if (Options.UseCascadeDelet</code>
	<code>↳ CountReferences(sequenc</code>		<code>↳ e </code>
	<code>↳ e) ==</code>		<code>↳ CountReferences(link)</code>
	<code>↳ 0)</code>		<code>↳ == 0)</code>
428	<code>{</code>	487	<code>{</code>
429	<code>Links.Unsync.Merge(sequ</code>	488	<code>Links.Unsync.Delete(lin</code>
	<code>↳ ence,</code>		<code>↳ k);</code>
	<code>↳ newSequence);</code>	489	<code>}</code>
430	<code>}</code>	490	<code>}</code>
431	<code>}</code>	491	<code>}</code>
432	<code>}</code>	492	<code>}</code>
433	<code>}</code>	493	<code>}</code>
434	<code>}</code>	494	<code>#endregion</code>
435	<code>#endregion</code>	495	
436	<code>#region Delete</code>	496	<code>#region Compactification</code>
437		497	
438	<code>public void Delete(params ulong[]</code>	498	<code>/// <remarks></code>
439	<code>↳ sequence)</code>	499	<code>/// bestVariant можно выбирать по</code>
440	<code>{</code>	500	<code>↳ максимальному числу использований,</code>
441	<code>Sync.ExecuteWriteOperation(() =></code>	501	<code>/// но балансированный позволяет</code>
442	<code>{</code>		<code>↳ гарантировать уникальность (если</code>
443	<code>↳ // TODO: Check all options only</code>	502	<code>↳ есть возможность,</code>
444	<code>↳ ones before loop execution</code>	503	<code>/// гарантировать его использование в</code>
	<code>foreach (var linkToDelete in</code>		<code>↳ других местах).</code>
	<code>↳ Each(sequence))</code>	504	<code>///</code>
445	<code>{</code>	505	<code>/// Получается этот метод должен</code>
446	<code>↳ DeleteOneCore(linkToDelete);</code>		<code>↳ игнорировать Options.EnforceSingleS</code>
447	<code>}</code>		<code>↳ equenceVersionOnWrite</code>
448	<code>});</code>	506	<code>/// </remarks></code>
449	<code>}</code>	507	<code>public ulong Compact(params ulong[]</code>
450	<code>}</code>		<code>↳ sequence)</code>
451	<code>private void DeleteOneCore(ulong link)</code>	508	<code>{</code>
452	<code>{</code>	509	<code>return</code>
453	<code>if (Options.UseGarbageCollection)</code>	510	<code>↳ Sync.ExecuteWriteOperation(() =></code>
454	<code>{</code>	511	<code>{</code>
455	<code>var sequenceElements =</code>	512	<code>if (sequence.IsNullOrEmpty())</code>
456	<code>↳ GetSequenceElements(link);</code>		<code>{</code>
	<code>var sequenceElementsContents =</code>	513	<code>return _constants.Null;</code>
	<code>↳ new UInt64Link(Links.GetLin</code>	514	<code>}</code>
	<code>↳ k(sequenceElements));</code>	515	<code>Links.EnsureEachLinkExists(sequ</code>
457	<code>var sequenceLink = GetSequenceB</code>		<code>↳ ence);</code>
	<code>↳ yElements(sequenceElements);</code>	516	<code>return CompactCore(sequence);</code>
458	<code>if (Options.UseCascadeDelete </code>	517	<code>});</code>
	<code>↳ CountReferences(link) == 0)</code>	518	<code>}</code>
459	<code>{</code>		<code>[MethodImpl(MethodImplOptions.Aggressive</code>
460	<code>if (sequenceLink !=</code>		<code>↳ eInlining)]</code>
461	<code>↳ _constants.Null)</code>		
	<code>{</code>		

```

519     private ulong CompactCore(params
        ↳ ulong[] sequence) =>
        ↳ UpdateCore(sequence, sequence);
520
521     #endregion
522
523     #region Garbage Collection
524
525     /// <remarks>
526     /// TODO: Добавить дополнительный
        ↳ обработчик / событие CanBeDeleted
        ↳ которое можно определить извне или
        ↳ в унаследованном классе
527     /// </remarks>
528     [MethodImpl(MethodImplOptions.AggressiveI
        ↳ eInlining)]
529     private bool IsGarbage(ulong link) =>
        ↳ link != Options.SequenceMarkerLink
        ↳ &&
        ↳ !Links.Unsync.IsPartialPoint(link)
        ↳ && Links.Count(link) == 0;
530
531     private void ClearGarbage(ulong link)
532     {
533         if (IsGarbage(link))
534         {
535             var contents = new UInt64Link(L
        ↳ inks.GetLink(link));
536             Links.Unsync.Delete(link);
537             ClearGarbage(contents.Source);
538             ClearGarbage(contents.Target);
539         }
540     }
541
542     #endregion
543
544     #region Walkers
545
546     public bool EachPart(Func<ulong, bool>
        ↳ handler, ulong sequence)
547     {
548         return Sync.ExecuteReadOperation(()
        ↳ =>
549         {
550             var links = Links.Unsync;
551             var walker = new RightSequenceW
        ↳ alker<ulong>(links);
552             foreach (var part in
        ↳ walker.Walk(sequence))
553             {
554                 if (!handler(links.GetIndex
        ↳ (part)))
555                 {
556                     return false;
557                 }
558             }
559             return true;
560         });
561     }
562
563     public class Matcher :
        ↳ RightSequenceWalker<ulong>
564     {
565         private readonly Sequences
        ↳ _sequences;
566         private readonly IList<LinkIndex>
        ↳ _patternSequence;
567         private readonly HashSet<LinkIndex>
        ↳ _linksInSequence;
568         private readonly HashSet<LinkIndex>
        ↳ _results;
569         private readonly Func<ulong, bool>
        ↳ _stopableHandler;
570         private readonly HashSet<ulong>
        ↳ _readAsElements;
571         private int _filterPosition;
572
573         public Matcher(Sequences sequences,
        ↳ IList<LinkIndex>
        ↳ patternSequence,
        ↳ HashSet<LinkIndex> results,
        ↳ Func<LinkIndex, bool>
        ↳ stopableHandler,
        ↳ HashSet<LinkIndex>
        ↳ readAsElements = null)
574             : base(sequences.Links.Unsync)
575         {
576             _sequences = sequences;
577
        _patternSequence =
        ↳ _patternSequence;
578         _linksInSequence = new
        ↳ HashSet<LinkIndex>(patternS
        ↳ equence.Where(x => x !=
        ↳ _constants.Any && x !=
        ↳ ZeroOrMany));
579         _results = results;
580         _stopableHandler =
        ↳ stopableHandler;
581         _readAsElements =
        ↳ readAsElements;
582     }
583
584     protected override bool
        ↳ IsElement(IList<ulong> link) =>
        ↳ base.IsElement(link) ||
        ↳ (_readAsElements != null &&
        ↳ _readAsElements.Contains(Links.
        ↳ GetIndex(link))) ||
        ↳ _linksInSequence.Contains(Links
        ↳ .GetIndex(link));
585
586     public bool FullMatch(LinkIndex
        ↳ sequenceToMatch)
587     {
588         _filterPosition = 0;
589         foreach (var part in
        ↳ Walk(sequenceToMatch))
590         {
591             if (!FullMatchCore(Links.Ge
        ↳ tIndex(part)))
592             {
593                 break;
594             }
595         }
596         return _filterPosition ==
        ↳ _patternSequence.Count;
597     }
598
599     private bool
        ↳ FullMatchCore(LinkIndex element)
600     {
601         if (_filterPosition ==
        ↳ _patternSequence.Count)
602         {
603             _filterPosition = -2; //
        ↳ Длиннее чем нужно
604             return false;
605         }
606         if (_patternSequence[_filterPos
        ↳ ition] !=
        ↳ _constants.Any
        ↳ && element != _patternSequence
        ↳ [_filterPosition])
607         {
608             _filterPosition = -1;
609             return false; //
        ↳ Начинается/Продолжается
        ↳ иначе
610         }
611         _filterPosition++;
612         return true;
613     }
614
615     public void
        ↳ AddFullMatchedToResults(ulong
        ↳ sequenceToMatch)
616     {
617         if (FullMatch(sequenceToMatch))
618         {
619             _results.Add(sequenceToMatc
        ↳ h);
620         }
621     }
622
623     public bool HandleFullMatched(ulong
        ↳ sequenceToMatch)
624     {
625         if (FullMatch(sequenceToMatch)
        ↳ && _results.Add(sequenceToM
        ↳ atch))
626         {
627             return _stopableHandler(seq
        ↳ uenceToMatch);
628         }

```

```

629     }
630     return true;
631 }
632
633 public bool
    ↪ HandleFullMatchedSequence(ulong
    ↪ sequenceToMatch)
634 {
635     var sequence =
        ↪ _sequences.GetSequenceByEle
        ↪ ments(sequenceToMatch);
636     if (sequence != _constants.Null
        ↪ &&
        ↪ FullMatch(sequenceToMatch)
        ↪ && _results.Add(sequenceToM
        ↪ atch))
637     {
638         return _stopableHandler(seq
        ↪ uence);
639     }
640     return true;
641 }
642
643 /// <remarks>
644 /// TODO: Add support for
    ↪ LinksConstants.Any
645 /// </remarks>
646 public bool PartialMatch(LinkIndex
    ↪ sequenceToMatch)
647 {
648     _filterPosition = -1;
649     foreach (var part in
        ↪ Walk(sequenceToMatch))
650     {
651         if (!PartialMatchCore(Links
        ↪ .GetIndex(part)))
652         {
653             break;
654         }
655     }
656     return _filterPosition ==
        ↪ _patternSequence.Count - 1;
657 }
658
659 private bool
    ↪ PartialMatchCore(LinkIndex
    ↪ element)
660 {
661     if (_filterPosition ==
        ↪ (_patternSequence.Count -
        ↪ 1))
662     {
663         return false; // Нашлось
664     }
665     if (_filterPosition >= 0)
666     {
667         if (element == _patternSequ
        ↪ ence[_filterPosition +
        ↪ 1])
668         {
669             _filterPosition++;
670         }
671         else
672         {
673             _filterPosition = -1;
674         }
675     }
676     if (_filterPosition < 0)
677     {
678         if (element ==
        ↪ _patternSequence[0])
679         {
680             _filterPosition = 0;
681         }
682     }
683     return true; // Ищем дальше
684 }
685
686 public void AddPartialMatchedToResu
    ↪ lts(ulong
    ↪ sequenceToMatch)
687 {
688     if (PartialMatch(sequenceToMatc
    ↪ h))
689     {

```

```

690         _results.Add(sequenceToMatc
    ↪ h);
691     }
692 }
693
694 public bool
    ↪ HandlePartialMatched(ulong
    ↪ sequenceToMatch)
695 {
696     if (PartialMatch(sequenceToMatc
    ↪ h))
697     {
698         return _stopableHandler(seq
    ↪ uenceToMatch);
699     }
700     return true;
701 }
702
703 public void AddAllPartialMatchedToR
    ↪ esults(IEnumerable<ulong>
    ↪ sequencesToMatch)
704 {
705     foreach (var sequenceToMatch in
    ↪ sequencesToMatch)
706     {
707         if (PartialMatch(sequenceTo
    ↪ Match))
708         {
709             _results.Add(sequenceTo
    ↪ Match);
710         }
711     }
712 }
713
714 public void AddAllPartialMatchedToR
    ↪ esultsAndReadAsElements(IEnumer
    ↪ able<ulong>
    ↪ sequencesToMatch)
715 {
716     foreach (var sequenceToMatch in
    ↪ sequencesToMatch)
717     {
718         if (PartialMatch(sequenceTo
    ↪ Match))
719         {
720             _readAsElements.Add(seq
    ↪ uenceToMatch);
721             _results.Add(sequenceTo
    ↪ Match);
722         }
723     }
724 }
725
726 #endregion
727 }
728
729 }

```

./Sequences/Sequences.Experiments.cs

```

1  using System;
2  using LinkIndex = System.UInt64;
3  using System.Collections.Generic;
4  using Stack =
    ↪ System.Collections.Generic.Stack<ulong>;
5  using System.Linq;
6  using System.Text;
7  using Platform.Collections;
8  using Platform.Numbers;
9  using Platform.Data.Exceptions;
10 using Platform.Data.Sequences;
11 using Platform.Data.Doublets.Sequences.Frequenc
    ↪ ies.Counters;
12 using Platform.Data.Doublets.Sequences.Walkers;
13
14 namespace Platform.Data.Doublets.Sequences
15 {
16     partial class Sequences
17     {
18         #region Create All Variants (Not
    ↪ Practical)
19
20         /// <remarks>
21         /// Number of links that is needed to
    ↪ generate all variants for

```



```

22    /// sequence of length N corresponds to
23    ↪ https://oeis.org/A014143/list
24    ↪ sequence.
25    /// </remarks>
26    public ulong[]
27    ↪ CreateAllVariants2(ulong[] sequence)
28    {
29        return
30        ↪ Sync.ExecuteWriteOperation(() =>
31        {
32            if (sequence.IsNullOrEmpty())
33            {
34                return new ulong[0];
35            }
36            Links.EnsureEachLinkExists(sequence);
37            if (sequence.Length == 1)
38            {
39                return sequence;
40            }
41            return CreateAllVariants2Core(sequence, 0, sequence.Length - 1);
42        });
43    }
44    private ulong[]
45    ↪ CreateAllVariants2Core(ulong[]
46    ↪ sequence, long startAt, long stopAt)
47    {
48        #if DEBUG
49            if ((stopAt - startAt) < 0)
50            {
51                throw new ArgumentOutOfRangeException(
52                ↪ option(nameof(startAt),
53                ↪ "startAt должен быть меньше
54                ↪ или равен stopAt");
55            }
56            #endif
57            if ((stopAt - startAt) == 0)
58            {
59                return new[] {
60                ↪ sequence[startAt] };
61            }
62            if ((stopAt - startAt) == 1)
63            {
64                return new[] {
65                ↪ Links.Unsync.CreateAndUpdate(
66                ↪ sequence[startAt],
67                ↪ sequence[stopAt]) };
68            }
69            var variants = new ulong[(ulong)MathHelpers.Catalan(stopAt -
70            ↪ startAt)];
71            var last = 0;
72            for (var splitter = startAt;
73            ↪ splitter < stopAt; splitter++)
74            {
75                var left = CreateAllVariants2Core(sequence, startAt,
76                ↪ splitter);
77                var right = CreateAllVariants2Core(sequence, splitter + 1,
78                ↪ stopAt);
79                for (var i = 0; i <
80                ↪ left.Length; i++)
81                {
82                    for (var j = 0; j <
83                    ↪ right.Length; j++)
84                    {
85                        var variant =
86                        ↪ Links.Unsync.CreateAndUpdate(left[i],
87                        ↪ right[j]);
88                        if (variant ==
89                        ↪ _constants.Null)
90                        {
91                            throw new NotImplementedException(
92                            ↪ "Creation
93                            ↪ cancellation is not
94                            ↪ implemented.");
95                        }
96                    }
97                }
98            }
99            return variants;
100        }
101    }
102    private List<ulong>
103    ↪ CreateAllVariants1(params ulong[]
104    ↪ sequence)
105    {
106        return
107        ↪ Sync.ExecuteWriteOperation(() =>
108        {
109            if (sequence.IsNullOrEmpty())
110            {
111                return new List<ulong>();
112            }
113            Links.Unsync.EnsureEachLinkExists(sequence);
114            if (sequence.Length == 1)
115            {
116                return new List<ulong> {
117                ↪ sequence[0] };
118            }
119            var results = new
120            ↪ List<ulong>((int)MathHelpers.Catalan(sequence.Length));
121            return CreateAllVariants1Core(sequence, List<ulong> results);
122        });
123    }
124    private List<ulong>
125    ↪ CreateAllVariants1Core(ulong[]
126    ↪ sequence, List<ulong> results)
127    {
128        if (sequence.Length == 2)
129        {
130            var link = Links.Unsync.CreateAndUpdate(sequence[0],
131            ↪ sequence[1]);
132            if (link == _constants.Null)
133            {
134                throw new NotImplementedException(
135                ↪ "Creation
136                ↪ cancellation is not
137                ↪ implemented.");
138            }
139            results.Add(link);
140            return results;
141        }
142        var innerSequenceLength =
143        ↪ sequence.Length - 1;
144        var innerSequence = new
145        ↪ ulong[innerSequenceLength];
146        for (var li = 0; li <
147        ↪ innerSequenceLength; li++)
148        {
149            var link = Links.Unsync.CreateAndUpdate(sequence[li],
150            ↪ sequence[li + 1]);
151            if (link == _constants.Null)
152            {
153                throw new NotImplementedException(
154                ↪ "Creation
155                ↪ cancellation is not
156                ↪ implemented.");
157            }
158            innerSequence[li] = link;
159            for (var isi = li + 1; isi <
160            ↪ innerSequenceLength; isi++)
161            {
162                innerSequence[isi] =
163                ↪ sequence[isi + 1];
164            }
165        }
166    }

```

```

127         CreateAllVariants1Core(innerSeq
128         ↪ uence,
129         ↪ results);
130     }
131     return results;
132 }
133 #endregion
134 public HashSet<ulong> Each1(params
135 ↪ ulong[] sequence)
136 {
137     var visitedLinks = new
138     ↪ HashSet<ulong>(); // Заменить
139     ↪ на bitstring
140     Each1(link =>
141     {
142         if (!visitedLinks.Contains(link
143         ↪ ))
144         {
145             visitedLinks.Add(link); //
146             ↪ изучить почему
147             ↪ случаются повторы
148         }
149         return true;
150     }, sequence);
151     return visitedLinks;
152 }
153 private void Each1(Func<ulong, bool>
154 ↪ handler, params ulong[] sequence)
155 {
156     if (sequence.Length == 2)
157     {
158         Links.Unsync.Each(sequence[0],
159         ↪ sequence[1], handler);
160     }
161     else
162     {
163         var innerSequenceLength =
164         ↪ sequence.Length - 1;
165         for (var li = 0; li <
166         ↪ innerSequenceLength; li++)
167         {
168             var left = sequence[li];
169             var right = sequence[li +
170             ↪ 1];
171             if (left == 0 && right == 0)
172             {
173                 continue;
174             }
175             var linkIndex = li;
176             ulong[] innerSequence =
177             ↪ null;
178             Links.Unsync.Each(left,
179             ↪ right, doublet =>
180             {
181                 if (innerSequence ==
182                 ↪ null)
183                 {
184                     innerSequence = new
185                     ↪ ulong[innerSequ
186                     ↪ enceLength];
187                     for (var isi = 0;
188                     ↪ isi <
189                     ↪ linkIndex;
190                     ↪ isi++)
191                     {
192                         innerSequence[i
193                         ↪ ]
194                         ↪ si] =
195                         ↪ sequence[is
196                         ↪ i];
197                     }
198                     for (var isi =
199                     ↪ linkIndex + 1;
200                     ↪ isi < innerSequ
201                     ↪ enceLength;
202                     ↪ isi++)
203                     {
204                         innerSequence[i
205                         ↪ ]
206                         ↪ si] =
207                         ↪ sequence[is
208                         ↪ i +
209                         ↪ 1];
210                     }
211                 }
212                 if (innerSequence[linkIndex]
213                 ↪ == 0)
214                 {
215                     innerSequence[linkIndex]
216                     ↪ = doublet;
217                 }
218                 return true;
219             }
220             return false;
221         }
222     }
223 }
224 }
225 }
226 }
227 }
228 }
229 }
230 }
231 }
232 }
233 }
234 }
235 }
236 }
237 }
238 }
239 }
240 }
241 }
242 }
243 }
244 }
245 }
246 }
247 }
248 }
249 }
250 }
251 }
252 }
253 }
254 }
255 }
256 }
257 }
258 }
259 }
260 }
261 }
262 }
263 }
264 }
265 }
266 }
267 }
268 }
269 }
270 }
271 }
272 }
273 }
274 }
275 }
276 }
277 }
278 }
279 }
280 }
281 }
282 }
283 }
284 }
285 }
286 }
287 }
288 }
289 }
290 }
291 }
292 }
293 }
294 }
295 }
296 }
297 }
298 }
299 }
300 }
301 }
302 }
303 }
304 }
305 }
306 }
307 }
308 }
309 }
310 }
311 }
312 }
313 }
314 }
315 }
316 }
317 }
318 }
319 }
320 }
321 }
322 }
323 }
324 }
325 }
326 }
327 }
328 }
329 }
330 }
331 }
332 }
333 }
334 }
335 }
336 }
337 }
338 }
339 }
340 }
341 }
342 }
343 }
344 }
345 }
346 }
347 }
348 }
349 }
350 }
351 }
352 }
353 }
354 }
355 }
356 }
357 }
358 }
359 }
360 }
361 }
362 }
363 }
364 }
365 }
366 }
367 }
368 }
369 }
370 }
371 }
372 }
373 }
374 }
375 }
376 }
377 }
378 }
379 }
380 }
381 }
382 }
383 }
384 }
385 }
386 }
387 }
388 }
389 }
390 }
391 }
392 }
393 }
394 }
395 }
396 }
397 }
398 }
399 }
400 }
401 }
402 }
403 }
404 }
405 }
406 }
407 }
408 }
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }
465 }
466 }
467 }
468 }
469 }
470 }
471 }
472 }
473 }
474 }
475 }
476 }
477 }
478 }
479 }
480 }
481 }
482 }
483 }
484 }
485 }
486 }
487 }
488 }
489 }
490 }
491 }
492 }
493 }
494 }
495 }
496 }
497 }
498 }
499 }
500 }
501 }
502 }
503 }
504 }
505 }
506 }
507 }
508 }
509 }
510 }
511 }
512 }
513 }
514 }
515 }
516 }
517 }
518 }
519 }
520 }
521 }
522 }
523 }
524 }
525 }
526 }
527 }
528 }
529 }
530 }
531 }
532 }
533 }
534 }
535 }
536 }
537 }
538 }
539 }
540 }
541 }
542 }
543 }
544 }
545 }
546 }
547 }
548 }
549 }
550 }
551 }
552 }
553 }
554 }
555 }
556 }
557 }
558 }
559 }
560 }
561 }
562 }
563 }
564 }
565 }
566 }
567 }
568 }
569 }
570 }
571 }
572 }
573 }
574 }
575 }
576 }
577 }
578 }
579 }
580 }
581 }
582 }
583 }
584 }
585 }
586 }
587 }
588 }
589 }
590 }
591 }
592 }
593 }
594 }
595 }
596 }
597 }
598 }
599 }
600 }
601 }
602 }
603 }
604 }
605 }
606 }
607 }
608 }
609 }
610 }
611 }
612 }
613 }
614 }
615 }
616 }
617 }
618 }
619 }
620 }
621 }
622 }
623 }
624 }
625 }
626 }
627 }
628 }
629 }
630 }
631 }
632 }
633 }
634 }
635 }
636 }
637 }
638 }
639 }
640 }
641 }
642 }
643 }
644 }
645 }
646 }
647 }
648 }
649 }
650 }
651 }
652 }
653 }
654 }
655 }
656 }
657 }
658 }
659 }
660 }
661 }
662 }
663 }
664 }
665 }
666 }
667 }
668 }
669 }
670 }
671 }
672 }
673 }
674 }
675 }
676 }
677 }
678 }
679 }
680 }
681 }
682 }
683 }
684 }
685 }
686 }
687 }
688 }
689 }
690 }
691 }
692 }
693 }
694 }
695 }
696 }
697 }
698 }
699 }
700 }
701 }
702 }
703 }
704 }
705 }
706 }
707 }
708 }
709 }
710 }
711 }
712 }
713 }
714 }
715 }
716 }
717 }
718 }
719 }
720 }
721 }
722 }
723 }
724 }
725 }
726 }
727 }
728 }
729 }
730 }
731 }
732 }
733 }
734 }
735 }
736 }
737 }
738 }
739 }
740 }
741 }
742 }
743 }
744 }
745 }
746 }
747 }
748 }
749 }
750 }
751 }
752 }
753 }
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 }
782 }
783 }
784 }
785 }
786 }
787 }
788 }
789 }
790 }
791 }
792 }
793 }
794 }
795 }
796 }
797 }
798 }
799 }
800 }
801 }
802 }
803 }
804 }
805 }
806 }
807 }
808 }
809 }
810 }
811 }
812 }
813 }
814 }
815 }
816 }
817 }
818 }
819 }
820 }
821 }
822 }
823 }
824 }
825 }
826 }
827 }
828 }
829 }
830 }
831 }
832 }
833 }
834 }
835 }
836 }
837 }
838 }
839 }
840 }
841 }
842 }
843 }
844 }
845 }
846 }
847 }
848 }
849 }
850 }
851 }
852 }
853 }
854 }
855 }
856 }
857 }
858 }
859 }
860 }
861 }
862 }
863 }
864 }
865 }
866 }
867 }
868 }
869 }
870 }
871 }
872 }
873 }
874 }
875 }
876 }
877 }
878 }
879 }
880 }
881 }
882 }
883 }
884 }
885 }
886 }
887 }
888 }
889 }
890 }
891 }
892 }
893 }
894 }
895 }
896 }
897 }
898 }
899 }
900 }
901 }
902 }
903 }
904 }
905 }
906 }
907 }
908 }
909 }
910 }
911 }
912 }
913 }
914 }
915 }
916 }
917 }
918 }
919 }
920 }
921 }
922 }
923 }
924 }
925 }
926 }
927 }
928 }
929 }
930 }
931 }
932 }
933 }
934 }
935 }
936 }
937 }
938 }
939 }
940 }
941 }
942 }
943 }
944 }
945 }
946 }
947 }
948 }
949 }
950 }
951 }
952 }
953 }
954 }
955 }
956 }
957 }
958 }
959 }
960 }
961 }
962 }
963 }
964 }
965 }
966 }
967 }
968 }
969 }
970 }
971 }
972 }
973 }
974 }
975 }
976 }
977 }
978 }
979 }
980 }
981 }
982 }
983 }
984 }
985 }
986 }
987 }
988 }
989 }
990 }
991 }
992 }
993 }
994 }
995 }
996 }
997 }
998 }
999 }
1000 }

```

```

243         var match = Links.SearchOrD
244             ↳ default(sequence[0],
245                 ↳ doublet);
246         if (match !=
247             ↳ _constants.Null)
248         {
249             handler(match);
250         }
251         return true;
252     });
253     // |_x      ... x_o
254     // |_o      |___|
255     Links.Each(_constants.Any,
256         ↳ sequence[0], doublet =>
257     {
258         var match = Links.SearchOrD
259             ↳ default(doublet,
260                 ↳ sequence[1]);
261         if (match != 0)
262         {
263             handler(match);
264         }
265         return true;
266     });
267     // |_x o_ .
268     // |_o_ |
269     PartialStepRight(x =>
270         ↳ handler(x), sequence[0],
271         ↳ sequence[1]);
272 }
273 else
274 {
275     // TODO: Implement other
276     ↳ variants
277     return;
278 }
279 }
280
281 private void
282     ↳ PartialStepRight(Action<ulong>
283     ↳ handler, ulong left, ulong right)
284 {
285     Links.Unsync.Each(_constants.Any,
286         ↳ left, doublet =>
287     {
288         StepRight(handler, doublet,
289             ↳ right);
290         if (left != doublet)
291         {
292             PartialStepRight(handler,
293                 ↳ doublet, right);
294         }
295         return true;
296     });
297 }
298
299 private void StepRight(Action<ulong>
300     ↳ handler, ulong left, ulong right)
301 {
302     Links.Unsync.Each(left,
303         ↳ _constants.Any, rightStep =>
304     {
305         TryStepRightUp(handler, right,
306             ↳ rightStep);
307         return true;
308     });
309 }
310
311 private void
312     ↳ TryStepRightUp(Action<ulong>
313     ↳ handler, ulong right, ulong
314     ↳ stepFrom)
315 {
316     var upStep = stepFrom;
317     var firstSource =
318         ↳ Links.Unsync.GetTarget(upStep);
319     while (firstSource != right &&
320         ↳ firstSource != upStep)
321     {
322         upStep = firstSource;
323         firstSource = Links.Unsync.GetS
324             ↳ ource(upStep);
325     }
326     if (firstSource == right)
327     {
328         handler(stepFrom);
329     }
330 }
331
332 private bool StartsWith(ulong sequence,
333     ↳ ulong link)
334 {
335     var upStep = sequence;
336     var firstSource =
337         ↳ Links.Unsync.GetSource(upStep);
338     while (firstSource != link &&
339         ↳ firstSource != upStep)
340     {
341         upStep = firstSource;
342         firstSource = Links.Unsync.GetS
343             ↳ ource(upStep);
344     }
345     return firstSource == link;
346 }
347
348 private bool EndsWith(ulong sequence,
349     ↳ ulong link)
350 {
351     var upStep = sequence;
352     var lastTarget =
353         ↳ Links.Unsync.GetTarget(upStep);
354     while (lastTarget != link &&
355         ↳ lastTarget != upStep)
356     {
357         upStep = lastTarget;
358         lastTarget = Links.Unsync.GetTa
359             ↳ rget(upStep);
360     }
361     return lastTarget == link;
362 }
363
364 }
365
366 // TODO: Test
367 private void
368     ↳ PartialStepLeft(Action<ulong>
369     ↳ handler, ulong left, ulong right)
370 {
371     Links.Unsync.Each(right,
372         ↳ _constants.Any, doublet =>
373     {
374         StepLeft(handler, left,
375             ↳ doublet);
376         if (right != doublet)
377         {
378             PartialStepLeft(handler,
379                 ↳ left, doublet);
380         }
381         return true;
382     });
383 }
384
385 private void StepLeft(Action<ulong>
386     ↳ handler, ulong left, ulong right)
387 {
388     Links.Unsync.Each(_constants.Any,
389         ↳ right, leftStep =>
390     {
391         TryStepLeftUp(handler, left,
392             ↳ leftStep);
393         return true;
394     });
395 }
396
397 private void
398     ↳ TryStepLeftUp(Action<ulong>
399     ↳ handler, ulong left, ulong stepFrom)
400 {
401     var upStep = stepFrom;
402     var firstTarget =
403         ↳ Links.Unsync.GetSource(upStep);
404     while (firstTarget != left &&
405         ↳ firstTarget != upStep)
406     {
407         upStep = firstTarget;
408         firstTarget = Links.Unsync.GetT
409             ↳ arget(upStep);
410     }
411     if (firstTarget == left)
412     {
413         handler(stepFrom);
414     }
415 }
416
417 private bool StartsWith(ulong sequence,
418     ↳ ulong link)
419 {
420     var upStep = sequence;
421     var firstSource =
422         ↳ Links.Unsync.GetSource(upStep);
423     while (firstSource != link &&
424         ↳ firstSource != upStep)
425     {
426         upStep = firstSource;
427         firstSource = Links.Unsync.GetS
428             ↳ ource(upStep);
429     }
430     return firstSource == link;
431 }
432
433 private bool EndsWith(ulong sequence,
434     ↳ ulong link)
435 {
436     var upStep = sequence;
437     var lastTarget =
438         ↳ Links.Unsync.GetTarget(upStep);
439     while (lastTarget != link &&
440         ↳ lastTarget != upStep)
441     {
442         upStep = lastTarget;
443         lastTarget = Links.Unsync.GetTa
444             ↳ rget(upStep);
445     }
446     return lastTarget == link;
447 }
448
449 }

```

```

public List<ulong>
    GetAllMatchingSequences0(params
        ulong[] sequence)
{
    return Sync.ExecuteReadOperation(()
        =>
        {
            var results = new List<ulong>();
            if (sequence.Length > 0)
            {
                Links.EnsureEachLinkExists(
                    sequence);
                var firstElement =
                    sequence[0];
                if (sequence.Length == 1)
                {
                    results.Add(firstElement);
                    return results;
                }
                if (sequence.Length == 2)
                {
                    var doublet =
                        Links.SearchOrDefault(
                            firstElement,
                            sequence[1]);
                    if (doublet !=
                        _constants.Null)
                    {
                        results.Add(doublet);
                    }
                    return results;
                }
                var linksInSequence = new HashSet<ulong>(sequence);
                void handler(ulong result)
                {
                    var filterPosition = 0;
                    StopableSequenceWalker.WalkRight(result,
                        Links.Unsync.GetSource,
                        Links.Unsync.GetTarget,
                        x => linksInSequence.
                            Contains(x)
                            || Links.Unsync.
                                GetTarget(x)
                                == x, x =>
                    {
                        if (filterPosition ==
                            sequence.Length)
                        {
                            filterPosition =
                                -2; // Длиннее
                                чем
                                нужно
                            return
                                false;
                        }
                        if (x != sequence[filterPosition])
                        {
                            filterPosition =
                                -1;
                            return
                                false;
                            // Начи
                            // нается
                            // иначе
                        }
                        filterPosition++;
                    }
                    return true;
                }
                if (filterPosition ==
                    sequence.Length)

```

```

        {
            results.Add(result);
        }
    }
    if (sequence.Length >= 2)
    {
        StepRight(handler,
            ↪ sequence[0],
            ↪ sequence[1]);
    }
    var last = sequence.Length
    ↪ - 2;
    for (var i = 1; i < last;
    ↪ i++)
    {
        PartialStepRight(handler,
            ↪ r, sequence[i],
            ↪ sequence[i + 1]);
    }
    if (sequence.Length >= 3)
    {
        StepLeft(handler, sequen
            ↪ nce[sequence.Length
            ↪ - 2], sequence[sequ
            ↪ ence.Length -
            ↪ 1]);
    }
}
return results;
});
}

public HashSet<ulong>
    ↪ GetAllMatchingSequences1(params
    ↪ ulong[] sequence)
{
    return Sync.ExecuteReadOperation(()
    ↪ =>
    {
        var results = new
            ↪ HashSet<ulong>();
        if (sequence.Length > 0)
        {
            Links.EnsureEachLinkExists(
                ↪ sequence);
            var firstElement =
                ↪ sequence[0];
            if (sequence.Length == 1)
            {
                results.Add(firstElemen
                    ↪ t);
                return results;
            }
            if (sequence.Length == 2)
            {
                var doublet =
                    ↪ Links.SearchOrDefau
                    ↪ lt(firstElement,
                    ↪ sequence[1]);
                if (doublet !=
                    ↪ _constants.Null)
                {
                    results.Add(doublet
                        ↪ );
                }
                return results;
            }
            var matcher = new
                ↪ Matcher(this, sequence,
                ↪ results, null);
            if (sequence.Length >= 2)
            {
                StepRight(matcher.AddFu
                    ↪ llMatchedToResults,
                    ↪ sequence[0],
                    ↪ sequence[1]);
            }
            var last = sequence.Length
                ↪ - 2;
            for (var i = 1; i < last;
                ↪ i++)
            {

```

```

469         PartialStepRight(matcher.AddFullMatchedToResults,
470             results,
471             sequence[i],
472             sequence[i + 1]);
473     }
474     if (sequence.Length >= 3)
475     {
476         StepLeft(matcher.AddFullMatchedToResults,
477             results,
478             sequence[sequence.Length - 2],
479             sequence[sequence.Length - 1]);
480     }
481     return results;
482 }
483
484 public const int MaxSequenceFormatSize = 200;
485
486 public string FormatSequence(LinkIndex sequenceLink, params LinkIndex[]
487     knownElements) =>
488     FormatSequence(sequenceLink, (sb, x) => sb.Append(x), true,
489         knownElements);
490
491 public string FormatSequence(LinkIndex sequenceLink, Action<StringBuilder,
492     LinkIndex> elementToString, bool insertComma, params LinkIndex[]
493     knownElements) => Links.SyncRoot.ExecuteReadOperation(() =>
494     FormatSequence(Links.Unsync, sequenceLink, elementToString,
495         insertComma, knownElements));
496
497 private string
498     FormatSequence(ILinks<LinkIndex> links, LinkIndex sequenceLink,
499     Action<StringBuilder, LinkIndex> elementToString, bool insertComma,
500     params LinkIndex[] knownElements)
501 {
502     var linksInSequence = new
503         HashSet<ulong>(knownElements);
504     //var entered = new
505         HashSet<ulong>();
506     var sb = new StringBuilder();
507     sb.Append('{');
508     if (links.Exists(sequenceLink))
509     {
510         StopableSequenceWalker.WalkRight
511             (sequenceLink,
512             links.GetSource,
513             links.GetTarget,
514             x => linksInSequence.Contains(x) ||
515             links.IsPartialPoint(x),
516             element => //
517             entered.AddAndReturnVoid(d, x => { },
518             entered.DoNotContains
519             {
520                 if (insertComma &&
521                     sb.Length > 1)
522                 {
523                     sb.Append(',');
524                 }
525                 if (entered.Contains(element))
526                 {
527                     sb.Append('{');
528                     elementToString(sb,
529                         element);
530                     sb.Append('}');
531                 }
532                 else
533                 {
534                     elementToString(sb,
535                         element);
536                 }
537             }
538             if (sb.Length < MaxSequenceFormatSize)
539             {
540                 return true;
541             }
542             sb.Append(insertComma ?
543                 ", ..." : "...");
544             return false;
545         }
546     }
547     sb.Append('}');
548     return sb.ToString();
549 }
550
551 public string
552     SafeFormatSequence(LinkIndex sequenceLink, params LinkIndex[]
553     knownElements) =>
554     SafeFormatSequence(sequenceLink, (sb, x) => sb.Append(x), true,
555         knownElements);
556
557 public string
558     SafeFormatSequence(LinkIndex sequenceLink, Action<StringBuilder,
559     LinkIndex> elementToString, bool insertComma, params LinkIndex[]
560     knownElements) => Links.SyncRoot.ExecuteReadOperation(() =>
561     SafeFormatSequence(Links.Unsync, sequenceLink, elementToString,
562         insertComma, knownElements));
563
564 private string SafeFormatSequence(ILinks<LinkIndex> links, LinkIndex
565     sequenceLink, Action<StringBuilder, LinkIndex> elementToString, bool
566     insertComma, params LinkIndex[] knownElements)
567 {
568     var linksInSequence = new
569         HashSet<ulong>(knownElements);
570     var entered = new HashSet<ulong>();
571     var sb = new StringBuilder();
572     sb.Append('{');
573     if (links.Exists(sequenceLink))
574     {
575         StopableSequenceWalker.WalkRight
576             (sequenceLink,
577             links.GetSource,
578             links.GetTarget,
579             x => linksInSequence.Contains(x) ||
580             links.IsFullPoint(x),
581             entered.AddAndReturnVoid(d, x => { },
582             entered.DoNotContains,
583             element =>
584             {
585                 if (insertComma &&
586                     sb.Length > 1)
587                 {
588                     sb.Append(',');
589                 }
590                 if (entered.Contains(element))
591                 {
592                     sb.Append('{');
593                     elementToString(sb,
594                         element);
595                     sb.Append('}');
596                 }
597                 else
598                 {
599                     elementToString(sb,
600                         element);
601                 }
602             }
603             if (sb.Length < MaxSequenceFormatSize)
604             {
605                 return true;
606             }
607             sb.Append(insertComma ?
608                 ", ..." : "...");
609             return false;
610         }
611     }
612     sb.Append('}');
613     return sb.ToString();
614 }

```

```

556         });
557     }
558     sb.Append('}');
559     return sb.ToString();
560 }
561
562 public List<ulong> GetAllPartiallyMatch
563     ingSequences0(params ulong[]
564     sequence)
565 {
566     return Sync.ExecuteReadOperation(()
567     =>
568     {
569         if (sequence.Length > 0)
570         {
571             Links.EnsureEachLinkExists(
572                 sequence);
573             var results = new
574                 HashSet<ulong>();
575             for (var i = 0; i <
576                 sequence.Length; i++)
577             {
578                 AllUsagesCore(sequence[
579                     i],
580                     results);
581             }
582             var filteredResults = new
583                 List<ulong>();
584             var linksInSequence = new H
585                 ashSet<ulong>(sequence);
586             foreach (var result in
587                 results)
588             {
589                 var filterPosition = -1;
590                 StopableSequenceWalker.
591                     WalkRight(result,
592                         Links.Unsync.GetSou
593                         rce,
594                         Links.Unsync.GetTar
595                         get,
596                         x => linksInSequenc
597                         e.Contains(x)
598                         || Links.Unsync
599                         .GetTarget(x)
600                         == x, x =>
601
602                 if (filterPosit
603                     ion ==
604                     (sequence.L
605                     ength -
606                     1))
607                 {
608                     return
609                         false;
610                 }
611                 if (filterPosit
612                     ion >=
613                     0)
614                 {
615                     if (x == se
616                         quence[
617                         filterP
618                         osition
619                         + 1])
620                     {
621                         filterP
622                             osi
623                             tio
624                             n++;
625                     }
626                     else
627                     {
628                         return
629                             fal
630                             se;
631                     }
632                 }
633                 if (filterPosit
634                     ion <
635                     0)
636                 {
637                     if (x ==
638                         sequenc
639                         e[0])
640                     {
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
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
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
```

```

654         }
655     }
656     return true;
657 }
658 return true;
659 });
660 }
661
662 //public HashSet<ulong> GetAllPartiallyMa
663     MatchingSequences3(params ulong[]
664     sequence)
665 //{
666 //    return
667 //    Sync.ExecuteReadOperation(() =>
668 //    {
669 //        if (sequence.Length > 0)
670 //        {
671 //            _links.EnsureEachLinkIsAn
672 //            yOrExists(sequence);
673 //
674 //            var firstResults = new
675 //            HashSet<ulong>();
676 //            var lastResults = new
677 //            HashSet<ulong>();
678 //
679 //            var first =
680 //            sequence.First(x => x !=
681 //            LinksConstants.Any);
682 //            var last =
683 //            sequence.Last(x => x !=
684 //            LinksConstants.Any);
685 //
686 //            AllUsagesCore(first,
687 //            firstResults);
688 //            AllUsagesCore(last,
689 //            lastResults);
690 //
691 //            firstResults.IntersectWit
692 //            h(lastResults);
693 //
694 //            //for (var i = 0; i <
695 //            sequence.Length; i++)
696 //            //
697 //            AllUsagesCore(sequence[i], results);
698 //
699 //            var filteredResults = new
700 //            HashSet<ulong>();
701 //            var matcher = new
702 //            Matcher(this, sequence,
703 //            filteredResults, null);
704 //            matcher.AddAllPartialMatc
705 //            hedToResults(firstResults);
706 //            return filteredResults;
707 //        }
708 //    }
709 //    return new HashSet<ulong>();
710 //});
711
712 public HashSet<ulong> GetAllPartiallyMa
713     tchingSequences3(params ulong[]
714     sequence)
715 {
716     return Sync.ExecuteReadOperation(()
717     =>
718     {
719         if (sequence.Length > 0)
720         {
721             Links.EnsureEachLinkIsAnyOr
722             Exists(sequence);
723             var firstResults = new
724             HashSet<ulong>();
725             var lastResults = new
726             HashSet<ulong>();
727             var first =
728             sequence.First(x => x
729             != _constants.Any);
730             var last = sequence.Last(x
731             => x != _constants.Any);
732             AllUsagesCore(first,
733             firstResults);
734             AllUsagesCore(last,
735             lastResults);
736         }
737     }
738 }

```

```

707     firstResults.IntersectWith(
708     lastResults);
709     //for (var i = 0; i <
710     sequence.Length; i++)
711     //    AllUsagesCore(sequence
712     e[i],
713     results);
714     var filteredResults = new
715     HashSet<ulong>();
716     var matcher = new
717     Matcher(this, sequence,
718     filteredResults, null);
719     matcher.AddAllPartialMatche
720     dToResults(firstResults
721     );
722     return filteredResults;
723 }
724 return new HashSet<ulong>();
725 });
726 }
727
728 public HashSet<ulong> GetAllPartiallyMa
729     tchingSequences4(HashSet<ulong>
730     readAsElements, IList<ulong>
731     sequence)
732 {
733     return Sync.ExecuteReadOperation(()
734     =>
735     {
736         if (sequence.Count > 0)
737         {
738             Links.EnsureEachLinkExists(
739             sequence);
740             var results = new
741             HashSet<LinkIndex>();
742             //var nextResults = new
743             HashSet<ulong>();
744             //for (var i = 0; i <
745             sequence.Length; i++)
746             //{
747                 AllUsagesCore(sequence
748                 e[i],
749                 nextResults);
750                 //    if (results.IsNullOrEmpty()
751                 //    {
752                     results =
753                     nextResults;
754                     nextResults = new
755                     HashSet<ulong>();
756                 }
757                 //    else
758                 //    {
759                     results.Intersect
760                     With(nextResults);
761                 }
762                 nextResults.Clear();
763             }
764             //}
765             var collector1 = new
766             AllUsagesCollector1(Lin
767             ks.Unsync,
768             results);
769             collector1.Collect(Links.Un
770             sync.GetLink(sequence[0
771             ]));
772             var next = new
773             HashSet<ulong>();
774             for (var i = 1; i <
775             sequence.Count; i++)
776             {
777                 var collector = new
778                 AllUsagesCollector1
779                 (Links.Unsync,
780                 next);
781                 collector.Collect(Links
782                 .Unsync.GetLink(seq
783                 uence[i]));
784
785                 results.IntersectWith(n
786                 ext);
787                 next.Clear();
788             }
789         }
790     }
791 }

```



```

753         var filteredResults = new
754             ↳ HashSet<ulong>();
755         var matcher = new
756             ↳ Matcher(this, sequence,
757             ↳ filteredResults, null,
758             ↳ readAsElements);
759         matcher.AddAllPartialMatche
760             ↳ dToResultsAndReadAsElem
761             ↳ ents(results.OrderBy(x
762             ↳ => x)); // OrderBy is a
763             ↳ Hack
764         return filteredResults;
765     }
766     return new HashSet<ulong>();
767 });
768 }
769 // Does not work
770 public HashSet<ulong> GetAllPartiallyMa
771     ↳ tchingSequences5(HashSet<ulong>
772     ↳ readAsElements, params ulong[]
773     ↳ sequence)
774 {
775     var visited = new HashSet<ulong>();
776     var results = new HashSet<ulong>();
777     var matcher = new Matcher(this,
778     ↳ sequence, visited, x => {
779     ↳ results.Add(x); return true; },
780     ↳ readAsElements);
781     var last = sequence.Length - 1;
782     for (var i = 0; i < last; i++)
783     {
784         PartialStepRight(matcher.Partia
785             ↳ lMatch, sequence[i],
786             ↳ sequence[i + 1]);
787     }
788     return results;
789 }
790 public List<ulong> GetAllPartiallyMatch
791     ↳ ingSequences(params ulong[]
792     ↳ sequence)
793 {
794     return Sync.ExecuteReadOperation(()
795     ↳ =>
796     {
797         if (sequence.Length > 0)
798         {
799             Links.EnsureEachLinkExists(
800                 ↳ sequence);
801             //var firstElement =
802             ↳ sequence[0];
803             //if (sequence.Length == 1)
804             //{
805             //    //results.Add(firstEl
806             ↳ ement);
807             //    return results;
808             //}
809             //if (sequence.Length == 2)
810             //{
811             //    //var doublet =
812             ↳ _links.SearchCore(first
813             ↳ Element,
814             ↳ sequence[1]);
815             //    //if (doublet !=
816             ↳ Doublets.Links.Null)
817             //    //
818             ↳ results.Add(doublet);
819             //    return results;
820             //}
821             //var lastElement = sequenc
822             ↳ e[sequence.Length -
823             ↳ 1];
824             //Func<ulong, bool> handler
825             ↳ = x =>
826             //{
827             //    if (StartsWith(x,
828             ↳ firstElement) &&
829             ↳ EndsWith(x,
830             ↳ lastElement))
831             ↳ results.Add(x);
832             //    return true;
833             //};
834             //if (sequence.Length >= 2)
835
836         // StepRight(handler,
837         ↳ sequence[0],
838         ↳ sequence[1]);
839         //var last =
840         ↳ sequence.Length - 2;
841         //for (var i = 1; i < last;
842         ↳ i++)
843         //    PartialStepRight(hand
844         ↳ ler, sequence[i],
845         ↳ sequence[i + 1]);
846         //if (sequence.Length >= 3)
847         //    StepLeft(handler, seq
848         ↳ uence[sequence.Length -
849         ↳ 2], sequence[sequence.L
850         ↳ ength -
851         ↳ 1]);
852         //if (sequence.Length
853         ↳ == 1)
854         //{
855         //    throw new NotImpl
856         ↳ ementedException(); //
857         ↳ all sequences,
858         ↳ containing this element?
859         //}
860         //if (sequence.Length
861         ↳ == 2)
862         //{
863         //    var results = new
864         ↳ List<ulong>();
865         //    PartialStepRight(
866         ↳ results.Add,
867         ↳ sequence[0],
868         ↳ sequence[1]);
869         //    return results;
870         //}
871         //var matches = new
872         ↳ List<List<ulong>>();
873         //var last =
874         ↳ sequence.Length - 1;
875         //for (var i = 0; i <
876         ↳ last; i++)
877         //{
878         //    var results = new
879         ↳ List<ulong>();
880         //    //StepRight(resul
881         ↳ ts.Add, sequence[i],
882         ↳ sequence[i + 1]);
883         //    PartialStepRight(
884         ↳ results.Add,
885         ↳ sequence[i], sequence[i
886         ↳ + 1]);
887         //    if (results.Count
888         ↳ > 0)
889         //    {
890         //        matches.Add(results);
891         //    }
892         //    else
893         //    {
894         //        return
895         ↳ results;
896         //    }
897         //    if (matches.Count
898         ↳ == 2)
899         //    {
900         //        var merged =
901         ↳ new List<ulong>();
902         //        for (var j =
903         ↳ 0; j <
904         ↳ matches[0].Count; j++)
905         //        {
906         //            for (var
907         ↳ k = 0; k <
908         ↳ matches[1].Count; k++)
909         //            {
910         //                CloseInnerConnections(m
911         ↳ erged.Add,
912         ↳ matches[0][j],
913         ↳ matches[1][k]);
914         //            }
915         //            if
916         ↳ (merged.Count > 0)
917         //            {
918         //                matches =
919         ↳ new List<List<ulong>> {
920         ↳ merged };
921         //            }
922         //            else
923         //            {
924         //                return
925         ↳ new List<ulong>();
926         //            }
927         //        }
928     }
929 }

```

```

841         ///////////////
842         ///////////////if (matches.Count > 0)
843         ///////////////
844         ///////////////
845         // var usages = new
846         //     HashSet<ulong>();
847         // for (int i = 0; i
848         //     < sequence.Length; i++)
849         // {
850         //     AllUsagesCore(
851         //         (sequence[i],
852         //         usages);
853         // }
854         // for (int i = 0;
855         //     i < matches[0].Count;
856         //     i++)
857         // {
858         //     // AllUsagesCo
859         //     re(matches[0][i],
860         //     usages);
861         // }
862         // usages.UnionWith(
863         //     h(matches[0]));
864         // return
865         //     usages.ToList();
866         // }
867         // }
868         // }
869         // }
870         // }
871         // }
872         // }
873         // }
874         // }
875         // }
876         // }
877         // }
878         // }
879         // }
880         // }
881         // }
882         // }
883         // }
884         // }
885         // }
886         // }
887         // }
888         // }
889         // }
890         // }
891         // }
892         // }
893         // }
894         // }
895         // }
896         // }
897         // }
898         // }
899         // }
900         // }
901         // }
902         // }
903         // }
904         // }
905         // }
906         // }
907         // }
908         // }
909         // }
910         // }
911         // }
912         // }
913         // }
914         // }
915         // }
916         // }
917         // }
918         // }
919         // }
920         // }
921         // }
922         // }
923         // }
924         // }
925         // }
926         // }
927         // }
928         // }
929         // }
930         // }
931         // }
932         // }
933         // }
934         // }
935         // }
936         // }
937         // }
938         // }
939         // }
940         // }
941         // }
942         // }
943         // }
944         // }
945         // }
946         // }
947         // }
948         // }
949         // }
950         // }
951         // }
952         // }
953         // }
954         // }
955         // }
956         // }
957         // }
958         // }
959         // }
960         // }
961         // }
962         // }
963         // }
964         // }
965         // }
966         // }
967         // }
968         // }
969         // }
970         // }
971         // }
972         // }
973         // }
974         // }
975         // }
976         // }
977         // }
978         // }
979         // }
980         // }
981         // }
982         // }
983         // }
984         // }
985         // }
986         // }
987         // }
988         // }
989         // }
990         // }
991         // }
992         // }
993         // }
994         // }
995         // }
996         // }
997         // }
998         // }
999         // }
1000        // }

```

```

945 private bool AllUsagesCore1(ulong link, 1005
    ↳ HashSet<ulong> usages, Func<ulong, 1006
    ↳ bool> outerHandler)
946 {
947     bool handler(ulong doublet) 1007
948     {
949         if (usages.Add(doublet))
950         {
951             if (!outerHandler(doublet)) 1008
952             {
953                 return false; 1009
954             } 1010
955             if (!AllUsagesCore1(doublet, 1011
    ↳ , usages, 1012
    ↳ outerHandler)) 1013
956             {
957                 return false; 1014
958             } 1015
959         } 1016
960         return true; 1017
961     } 1018
962     return Links.Unsync.Each(link, 1019
    ↳ _constants.Any, handler) 1020
    ↳ && Links.Unsync.Each(_constants 1021
    ↳ .Any, link, 1022
    ↳ handler); 1023
964 } 1024
965 1025
966 public void CalculateAllUsages(ulong[] 1026
    ↳ totals) 1027
967 { 1028
968     var calculator = new 1029
    ↳ AllUsagesCalculator(Links, 1030
    ↳ totals); 1031
969     calculator.Calculate(); 1032
970 } 1033
971 1034
972 public void CalculateAllUsages2(ulong[] 1035
    ↳ totals) 1036
973 { 1037
974     var calculator = new 1038
    ↳ AllUsagesCalculator2(Links, 1039
    ↳ totals); 1040
975     calculator.Calculate(); 1041
976 } 1042
977 1043
978 private class AllUsagesCalculator 1044
979 { 1045
980     private readonly 1046
    ↳ SynchronizedLinks<ulong> _links; 1047
981     private readonly ulong[] _totals; 1048
982 1049
983     public AllUsagesCalculator(Synchron 1050
    ↳ izedLinks<ulong> links, ulong[] 1051
    ↳ totals) 1052
984     { 1053
985         _links = links; 1054
986         _totals = totals; 1055
987     } 1056
988 1057
989     public void Calculate() => 1058
    ↳ _links.Each(_constants.Any, 1059
    ↳ _constants.Any, CalculateCore); 1060
990 1061
991     private bool CalculateCore(ulong 1062
    ↳ link) 1063
992     { 1064
993         if (_totals[link] == 0) 1065
994         { 1066
995             var total = 1UL; 1067
996             _totals[link] = total; 1068
997             var visitedChildren = new 1069
    ↳ HashSet<ulong>(); 1070
998             bool linkCalculator(ulong 1071
    ↳ child) 1072
999             { 1073
1000                 if (link != child && 1074
    ↳ visitedChildren.Add 1075
    ↳ (child)) 1076
1001                 { 1077
1002                     total += 1078
    ↳ _totals[child] 1079
    ↳ == 0 ? 1 : 1080
    ↳ _totals[child]; 1081
1003                 } 1082
1004                 return true; 1083

```

```

    }
    _links.Unsync.Each(link,
    ↳ _constants.Any,
    ↳ linkCalculator);
    _links.Unsync.Each(_constan
    ↳ ts.Any, link,
    ↳ linkCalculator);
    _totals[link] = total;
    }
    return true;
}
}

private class AllUsagesCalculator2
{
    private readonly
    ↳ SynchronizedLinks<ulong> _links;
    private readonly ulong[] _totals;

    public AllUsagesCalculator2(Synchro
    ↳ nizedLinks<ulong> links,
    ↳ ulong[] totals)
    {
        _links = links;
        _totals = totals;
    }

    public void Calculate() =>
    ↳ _links.Each(_constants.Any,
    ↳ _constants.Any, CalculateCore);

    private bool IsElement(ulong link)
    {
        // _linksInSequence.Contains(lin
        ↳ k)
        ↳ ||
        return _links.Unsync.GetTarget(
        ↳ link) == link ||
        ↳ _links.Unsync.GetSource(lin
        ↳ k) ==
        ↳ link;
    }

    private bool CalculateCore(ulong
    ↳ link)
    {
        // TODO: Проработать защиту от
        ↳ зацикливания
        // Основано на
        ↳ SequenceWalker.WalkLeft
        Func<ulong, ulong> getSource =
        ↳ _links.Unsync.GetSource;
        Func<ulong, ulong> getTarget =
        ↳ _links.Unsync.GetTarget;
        Func<ulong, bool> isElement =
        ↳ IsElement;
        void visitLeaf(ulong parent)
        {
            if (link != parent)
            {
                _totals[parent]++;
            }
        }
        void visitNode(ulong parent)
        {
            if (link != parent)
            {
                _totals[parent]++;
            }
        }
        var stack = new Stack();
        var element = link;
        if (isElement(element))
        {
            visitLeaf(element);
        }
        else
        {
            while (true)
            {
                if (isElement(element))
                {
                    if (stack.Count ==
                    ↳ 0)
                    {
                        break;

```

```

1069     }
1070     element =
1071         ↪ stack.Pop();
1072         ↪ rce(element);
1073         ↪ var target = getTar
1074         ↪ get(element);
1075         ↪ // 06работка
1076         ↪ элемента
1077         ↪ if (isElement(target
1078         ↪ t))
1079         {
1080             visitLeaf(target
1081                 ↪ t);
1082         }
1083         ↪ if (isElement(sourc
1084         ↪ e))
1085         {
1086             visitLeaf(sourc
1087                 ↪ e);
1088         }
1089         ↪ element = source;
1090     }
1091     }
1092     _totals[link]++;
1093     return true;
1094 }
1095 }
1096 private class AllUsagesCollector
1097 {
1098     private readonly ILinks<ulong>
1099         ↪ _links;
1100     private readonly HashSet<ulong>
1101         ↪ _usages;
1102     public AllUsagesCollector(ILinks<ul
1103         ↪ ong> links, HashSet<ulong>
1104         ↪ usages)
1105     {
1106         _links = links;
1107         _usages = usages;
1108     }
1109     public bool Collect(ulong link)
1110     {
1111         if (_usages.Add(link))
1112         {
1113             _links.Each(link,
1114                 ↪ _constants.Any,
1115                 ↪ Collect);
1116             _links.Each(_constants.Any,
1117                 ↪ link, Collect);
1118         }
1119         return true;
1120     }
1121 }
1122 private class AllUsagesCollector1
1123 {
1124     private readonly ILinks<ulong>
1125         ↪ _links;
1126     private readonly HashSet<ulong>
1127         ↪ _usages;
1128     private readonly ulong _continue;
1129     public AllUsagesCollector1(ILinks<u
1130         ↪ long> links, HashSet<ulong>
1131         ↪ usages)
1132     {
1133         _links = links;
1134         _usages = usages;
1135         _continue =
1136             ↪ _links.Constants.Continue;
1137     }
1138     public ulong Collect(IList<ulong>
1139         ↪ link)
1140     {

```

```

1134     var linkIndex =
1135         ↪ _links.GetIndex(link);
1136     if (_usages.Add(linkIndex))
1137     {
1138         _links.Each(Collect,
1139             ↪ _constants.Any,
1140             ↪ linkIndex);
1141     }
1142     return _continue;
1143 }
1144 }
1145 private class AllUsagesCollector2
1146 {
1147     private readonly ILinks<ulong>
1148         ↪ _links;
1149     private readonly BitString _usages;
1150     public AllUsagesCollector2(ILinks<u
1151         ↪ long> links, BitString
1152         ↪ usages)
1153     {
1154         _links = links;
1155         _usages = usages;
1156     }
1157     public bool Collect(ulong link)
1158     {
1159         if (_usages.Add((long)link))
1160         {
1161             _links.Each(link,
1162                 ↪ _constants.Any,
1163                 ↪ Collect);
1164             _links.Each(_constants.Any,
1165                 ↪ link, Collect);
1166         }
1167         return true;
1168     }
1169 }
1170 }
1171 private class
1172     ↪ AllUsagesIntersectingCollector
1173 {
1174     private readonly
1175         ↪ SynchronizedLinks<ulong> _links;
1176     private readonly HashSet<ulong>
1177         ↪ _intersectWith;
1178     private readonly HashSet<ulong>
1179         ↪ _usages;
1180     private readonly HashSet<ulong>
1181         ↪ _enter;
1182     public AllUsagesIntersectingCollect
1183         ↪ or(SynchronizedLinks<ulong>
1184         ↪ links, HashSet<ulong>
1185         ↪ intersectWith, HashSet<ulong>
1186         ↪ usages)
1187     {
1188         _links = links;
1189         _intersectWith = intersectWith;
1190         _usages = usages;
1191         _enter = new HashSet<ulong>();
1192         ↪ // защита от зацикливания
1193     }
1194     public bool Collect(ulong link)
1195     {
1196         if (_enter.Add(link))
1197         {
1198             if (_intersectWith.Contains
1199                 ↪ (link))
1200             {
1201                 _usages.Add(link);
1202             }
1203             _links.Unsync.Each(link,
1204                 ↪ _constants.Any,
1205                 ↪ Collect);
1206             _links.Unsync.Each(_constan
1207                 ↪ ts.Any, link,
1208                 ↪ Collect);
1209         }
1210         return true;
1211     }
1212 }
1213 }

```

```

1195 private void
1196     ↳ CloseInnerConnections(Action<ulong>
1197     ↳ handler, ulong left, ulong right)
1198 {
1199     TryStepLeftUp(handler, left, right);
1200     TryStepRightUp(handler, right,
1201     ↳ left);
1202 }
1203
1204 private void
1205     ↳ AllCloseConnections(Action<ulong>
1206     ↳ handler, ulong left, ulong right)
1207 {
1208     // Direct
1209     if (left == right)
1210     {
1211         handler(left);
1212     }
1213     var doublet = Links.Unsync.SearchOr
1214     ↳ Default(left,
1215     ↳ right);
1216     if (doublet != _constants.Null)
1217     {
1218         handler(doublet);
1219     }
1220     // Inner
1221     CloseInnerConnections(handler,
1222     ↳ left, right);
1223     // Outer
1224     StepLeft(handler, left, right);
1225     StepRight(handler, left, right);
1226     PartialStepRight(handler, left,
1227     ↳ right);
1228     PartialStepLeft(handler, left,
1229     ↳ right);
1230 }
1231
1232 private HashSet<ulong> GetAllPartiallyM
1233     ↳ atchingSequencesCore(ulong[]
1234     ↳ sequence, HashSet<ulong>
1235     ↳ previousMatchings, long startAt)
1236 {
1237     if (startAt >= sequence.Length) // ?
1238     {
1239         return previousMatchings;
1240     }
1241     var secondLinkUsages = new
1242     ↳ HashSet<ulong>();
1243     AllUsagesCore(sequence[startAt],
1244     ↳ secondLinkUsages);
1245     secondLinkUsages.Add(sequence[start
1246     ↳ At]);
1247     var matchings = new
1248     ↳ HashSet<ulong>();
1249     //for (var i = 0; i <
1250     ↳ previousMatchings.Count; i++)
1251     foreach (var secondLinkUsage in
1252     ↳ secondLinkUsages)
1253     {
1254         foreach (var previousMatching
1255     ↳ in previousMatchings)
1256         {
1257             //AllCloseConnections(match
1258     ↳ ings.AddAndReturnVoid,
1259     ↳ previousMatching,
1260     ↳ secondLinkUsage);
1261             StepRight(matchings.AddAndR
1262     ↳ eturnVoid,
1263     ↳ previousMatching,
1264     ↳ secondLinkUsage);
1265             TryStepRightUp(matchings.Ad
1266     ↳ dAndReturnVoid,
1267     ↳ secondLinkUsage,
1268     ↳ previousMatching);
1269             //PartialStepRight(matching
1270     ↳ s.AddAndReturnVoid,
1271     ↳ secondLinkUsage,
1272     ↳ sequence[startAt]); //
1273     ↳ почему-то эта ошибочная
1274     ↳ запись приводит к
1275     ↳ желаемым результатам.
1276             PartialStepRight(matchings.
1277     ↳ AddAndReturnVoid,
1278     ↳ previousMatching,
1279     ↳ secondLinkUsage);

```

```

    }
    }
    if (matchings.Count == 0)
    {
        return matchings;
    }
    return GetAllPartiallyMatchingSeque
    ↳ ncesCore(sequence, matchings,
    ↳ startAt + 1); // ??
}

private static void
    ↳ EnsureEachLinkIsAnyOrZeroOrManyOrEx
    ↳ ists(SynchronizedLinks<ulong>
    ↳ links, params ulong[] sequence)
{
    if (sequence == null)
    {
        return;
    }
    for (var i = 0; i <
    ↳ sequence.Length; i++)
    {
        if (sequence[i] !=
    ↳ _constants.Any &&
    ↳ sequence[i] != ZeroOrMany
    ↳ &&
    ↳ !links.Exists(sequence[i]))
        {
            throw new ArgumentLinkDoesN
    ↳ otExistsException<ulong>
    ↳ >(sequence[i],
    ↳ $"patternSequence[{i}]"
    ↳ );
        }
    }
}

// Pattern Matching -> Key To Triggers
public HashSet<ulong>
    ↳ MatchPattern(params ulong[]
    ↳ patternSequence)
{
    return Sync.ExecuteReadOperation((
    ↳ =>
    {
        patternSequence =
    ↳ Simplify(patternSequence);
        if (patternSequence.Length > 0)
        {
            EnsureEachLinkIsAnyOrZeroOr
    ↳ ManyOrExists(Links,
    ↳ patternSequence);
            var uniqueSequenceElements
    ↳ = new HashSet<ulong>();
            for (var i = 0; i <
    ↳ patternSequence.Length;
    ↳ i++)
            {
                if (patternSequence[i]
    ↳ != _constants.Any
    ↳ &&
    ↳ patternSequence[i]
    ↳ != ZeroOrMany)
                {
                    uniqueSequenceEleme
    ↳ nts.Add(pattern
    ↳ Sequence[i]);
                }
            }
            var results = new
    ↳ HashSet<ulong>();
            foreach (var
    ↳ uniqueSequenceElement
    ↳ in uniqueSequenceElements)
            {
                AllUsagesCore(uniqueSeq
    ↳ uenceElement,
    ↳ results);
            }
            var filteredResults = new
    ↳ HashSet<ulong>();

```

```

1289         var matcher = new
1290             ↪ PatternMatcher(this,
1291             ↪ patternSequence,
1292             ↪ filteredResults);
1293         matcher.AddAllPatternMatche
1294             ↪ dToResults(results);
1295         return filteredResults;
1296     }
1297     ↪ return new HashSet<ulong>();
1298     ↪ });
1299     ↪ }
1300     ↪ // Найти все возможные связи между
1301     ↪ ↪ указанным списком связей.
1302     ↪ ↪ Находит связи между всеми указанными
1303     ↪ ↪ связями в любом порядке.
1304     ↪ ↪ TODO: решить что делать с повторами
1305     ↪ ↪ (когда одни и те же элементы
1306     ↪ ↪ встречаются несколько раз в
1307     ↪ ↪ последовательности)
1308     ↪ public HashSet<ulong>
1309     ↪ ↪ GetAllConnections(params ulong[]
1310     ↪ ↪ linksToConnect)
1311     ↪ {
1312     ↪     return Sync.ExecuteReadOperation(()
1313     ↪     ↪ =>
1314     ↪     {
1315     ↪         var results = new
1316     ↪         ↪ HashSet<ulong>();
1317     ↪         if (linksToConnect.Length > 0)
1318     ↪         {
1319     ↪             Links.EnsureEachLinkExists(
1320     ↪             ↪ linksToConnect);
1321     ↪             AllUsagesCore(linksToConnec
1322     ↪             ↪ t[0],
1323     ↪             ↪ results);
1324     ↪             for (var i = 1; i <
1325     ↪             ↪ linksToConnect.Length;
1326     ↪             ↪ i++)
1327     ↪             {
1328     ↪                 var next = new
1329     ↪                 ↪ HashSet<ulong>();
1330     ↪                 AllUsagesCore(linksToCo
1331     ↪                 ↪ nnect[i],
1332     ↪                 ↪ next);
1333     ↪                 results.IntersectWith(n
1334     ↪                 ↪ ext);
1335     ↪             }
1336     ↪         }
1337     ↪     }
1338     ↪     return results;
1339     ↪ });
1340     ↪ }
1341     ↪ public HashSet<ulong>
1342     ↪ ↪ GetAllConnections2(params ulong[]
1343     ↪ ↪ linksToConnect)
1344     ↪ {
1345     ↪     return Sync.ExecuteReadOperation(()
1346     ↪     ↪ =>
1347     ↪     {
1348     ↪         var results = new
1349     ↪         ↪ HashSet<ulong>();
1350     ↪         if (linksToConnect.Length > 0)
1351     ↪         {
1352     ↪             Links.EnsureEachLinkExists(
1353     ↪             ↪ linksToConnect);
1354     ↪             var collector1 = new AllUsa
1355     ↪             ↪ gesCollector(Links,
1356     ↪             ↪ results);
1357     ↪             collector1.Collect(linksToC
1358     ↪             ↪ onnect[0]);
1359     ↪             //AllUsagesCore(linksToConn
1360     ↪             ↪ ect[0],
1361     ↪             ↪ results);
1362     ↪             for (var i = 1; i <
1363     ↪             ↪ linksToConnect.Length;
1364     ↪             ↪ i++)
1365     ↪             {
1366     ↪                 var next = new
1367     ↪                 ↪ HashSet<ulong>();
1368     ↪                 var collector = new
1369     ↪                 ↪ AllUsagesIntersecti
1370     ↪                 ↪ ngCollector(Links,
1371     ↪                 ↪ results, next);
1372     ↪                 collector.Collect(links
1373     ↪                 ↪ ToConnect[i]);
1374     ↪                 //AllUsagesCore(linksTo
1375     ↪                 ↪ Connect[i],
1376     ↪                 ↪ next);
1377     ↪                 //results.IntersectWith
1378     ↪                 ↪ (next);
1379     ↪                 results = next;
1380     ↪             }
1381     ↪         }
1382     ↪     }
1383     ↪     return results;
1384     ↪ });
1385     ↪ }
1386     ↪ public List<ulong>
1387     ↪ ↪ GetAllConnections3(params ulong[]
1388     ↪ ↪ linksToConnect)
1389     ↪ {
1390     ↪     return Sync.ExecuteReadOperation(()
1391     ↪     ↪ =>
1392     ↪     {
1393     ↪         var results = new BitString((lo
1394     ↪         ↪ ng)Links.Unsync.Count() +
1395     ↪         ↪ 1); // new
1396     ↪         ↪ BitArray((int)_links.Total
1397     ↪         ↪ + 1);
1398     ↪         if (linksToConnect.Length > 0)
1399     ↪         {
1400     ↪             Links.EnsureEachLinkExists(
1401     ↪             ↪ linksToConnect);
1402     ↪             var collector1 = new
1403     ↪             ↪ AllUsagesCollector2(Lin
1404     ↪             ↪ ks.Unsync,
1405     ↪             ↪ results);
1406     ↪             collector1.Collect(linksToC
1407     ↪             ↪ onnect[0]);
1408     ↪             for (var i = 1; i <
1409     ↪             ↪ linksToConnect.Length;
1410     ↪             ↪ i++)
1411     ↪             {
1412     ↪                 var collector = new
1413     ↪                 ↪ AllUsagesCollector(
1414     ↪                 ↪ Links.Unsync,
1415     ↪                 ↪ next);
1416     ↪                 collector.Collect(links
1417     ↪                 ↪ ToConnect[i]);
1418     ↪             }
1419     ↪         }
1420     ↪     }
1421     ↪     return results;
1422     ↪ });
1423     ↪ }
1424     ↪ public List<ulong>
1425     ↪ ↪ GetAllConnections4(params ulong[]
1426     ↪ ↪ linksToConnect)
1427     ↪ {
1428     ↪     return Sync.ExecuteReadOperation(()
1429     ↪     ↪ =>
1430     ↪     {
1431     ↪         var results = new BitString((lo
1432     ↪         ↪ ng)Links.Unsync.Count() +
1433     ↪         ↪ 1); // new
1434     ↪         ↪ BitArray((int)_links.Total
1435     ↪         ↪ + 1);
1436     ↪         if (linksToConnect.Length > 0)
1437     ↪         {
1438     ↪             Links.EnsureEachLinkExists(
1439     ↪             ↪ linksToConnect);
1440     ↪             var collector1 = new
1441     ↪             ↪ AllUsagesCollector2(Lin
1442     ↪             ↪ ks.Unsync,
1443     ↪             ↪ results);
1444     ↪             collector1.Collect(linksToC
1445     ↪             ↪ onnect[0]);
1446     ↪             for (var i = 1; i <
1447     ↪             ↪ linksToConnect.Length;
1448     ↪             ↪ i++)
1449     ↪             {
1450     ↪                 var collector = new
1451     ↪                 ↪ AllUsagesCollector(
1452     ↪                 ↪ Links.Unsync,
1453     ↪                 ↪ next);
1454     ↪                 collector.Collect(links
1455     ↪                 ↪ ToConnect[i]);
1456     ↪             }
1457     ↪         }
1458     ↪     }
1459     ↪     return results;
1460     ↪ });
1461     ↪ }
1462     ↪ public List<ulong>
1463     ↪ ↪ GetAllConnections5(params ulong[]
1464     ↪ ↪ linksToConnect)
1465     ↪ {
1466     ↪     return Sync.ExecuteReadOperation(()
1467     ↪     ↪ =>
1468     ↪     {
1469     ↪         var results = new BitString((lo
1470     ↪         ↪ ng)Links.Unsync.Count() +
1471     ↪         ↪ 1); // new
1472     ↪         ↪ BitArray((int)_links.Total
1473     ↪         ↪ + 1);
1474     ↪         if (linksToConnect.Length > 0)
1475     ↪         {
1476     ↪             Links.EnsureEachLinkExists(
1477     ↪             ↪ linksToConnect);
1478     ↪             var collector1 = new
1479     ↪             ↪ AllUsagesCollector2(Lin
1480     ↪             ↪ ks.Unsync,
1481     ↪             ↪ results);
1482     ↪             collector1.Collect(linksToC
1483     ↪             ↪ onnect[0]);
1484     ↪             for (var i = 1; i <
1485     ↪             ↪ linksToConnect.Length;
1486     ↪             ↪ i++)
1487     ↪             {
1488     ↪                 var collector = new
1489     ↪                 ↪ AllUsagesCollector(
1490     ↪                 ↪ Links.Unsync,
1491     ↪                 ↪ next);
1492     ↪                 collector.Collect(links
1493     ↪                 ↪ ToConnect[i]);
1494     ↪             }
1495     ↪         }
1496     ↪     }
1497     ↪     return results;
1498     ↪ });
1499     ↪ }
1500     ↪ public List<ulong>
1501     ↪ ↪ GetAllConnections6(params ulong[]
1502     ↪ ↪ linksToConnect)
1503     ↪ {
1504     ↪     return Sync.ExecuteReadOperation(()
1505     ↪     ↪ =>
1506     ↪     {
1507     ↪         var results = new BitString((lo
1508     ↪         ↪ ng)Links.Unsync.Count() +
1509     ↪         ↪ 1); // new
1510     ↪         ↪ BitArray((int)_links.Total
1511     ↪         ↪ + 1);
1512     ↪         if (linksToConnect.Length > 0)
1513     ↪         {
1514     ↪             Links.EnsureEachLinkExists(
1515     ↪             ↪ linksToConnect);
1516     ↪             var collector1 = new
1517     ↪             ↪ AllUsagesCollector2(Lin
1518     ↪             ↪ ks.Unsync,
1519     ↪             ↪ results);
1520     ↪             collector1.Collect(linksToC
1521     ↪             ↪ onnect[0]);
1522     ↪             for (var i = 1; i <
1523     ↪             ↪ linksToConnect.Length;
1524     ↪             ↪ i++)
1525     ↪             {
1526     ↪                 var collector = new
1527     ↪                 ↪ AllUsagesCollector(
1528     ↪                 ↪ Links.Unsync,
1529     ↪                 ↪ next);
1530     ↪                 collector.Collect(links
1531     ↪                 ↪ ToConnect[i]);
1532     ↪             }
1533     ↪         }
1534     ↪     }
1535     ↪     return results;
1536     ↪ });
1537     ↪ }
1538     ↪ public List<ulong>
1539     ↪ ↪ GetAllConnections7(params ulong[]
1540     ↪ ↪ linksToConnect)
1541     ↪ {
1542     ↪     return Sync.ExecuteReadOperation(()
1543     ↪     ↪ =>
1544     ↪     {
1545     ↪         var results = new BitString((lo
1546     ↪         ↪ ng)Links.Unsync.Count() +
1547     ↪         ↪ 1); // new
1548     ↪         ↪ BitArray((int)_links.Total
1549     ↪         ↪ + 1);
1550     ↪         if (linksToConnect.Length > 0)
1551     ↪         {
1552     ↪             Links.EnsureEachLinkExists(
1553     ↪             ↪ linksToConnect);
1554     ↪             var collector1 = new
1555     ↪             ↪ AllUsagesCollector2(Lin
1556     ↪             ↪ ks.Unsync,
1557     ↪             ↪ results);
1558     ↪             collector1.Collect(linksToC
1559     ↪             ↪ onnect[0]);
1560     ↪             for (var i = 1; i <
1561     ↪             ↪ linksToConnect.Length;
1562     ↪             ↪ i++)
1563     ↪             {
1564     ↪                 var collector = new
1565     ↪                 ↪ AllUsagesCollector(
1566     ↪                 ↪ Links.Unsync,
1567     ↪                 ↪ next);
1568     ↪                 collector.Collect(links
1569     ↪                 ↪ ToConnect[i]);
1570     ↪             }
1571     ↪         }
1572     ↪     }
1573     ↪     return results;
1574     ↪ });
1575     ↪ }
1576     ↪ public List<ulong>
1577     ↪ ↪ GetAllConnections8(params ulong[]
1578     ↪ ↪ linksToConnect)
1579     ↪ {
1580     ↪     return Sync.ExecuteReadOperation(()
1581     ↪     ↪ =>
1582     ↪     {
1583     ↪         var results = new BitString((lo
1584     ↪         ↪ ng)Links.Unsync.Count() +
1585     ↪         ↪ 1); // new
1586     ↪         ↪ BitArray((int)_links.Total
1587     ↪         ↪ + 1);
1588     ↪         if (linksToConnect.Length > 0)
1589     ↪         {
1590     ↪             Links.EnsureEachLinkExists(
1591     ↪             ↪ linksToConnect);
1592     ↪             var collector1 = new
1593     ↪             ↪ AllUsagesCollector2(Lin
1594     ↪             ↪ ks.Unsync,
1595     ↪             ↪ results);
1596     ↪             collector1.Collect(linksToC
1597     ↪             ↪ onnect[0]);
1598     ↪             for (var i = 1; i <
1599     ↪             ↪ linksToConnect.Length;
1600     ↪             ↪ i++)
1601     ↪             {
1602     ↪                 var collector = new
1603     ↪                 ↪ AllUsagesCollector(
1604     ↪                 ↪ Links.Unsync,
1605     ↪                 ↪ next);
1606     ↪                 collector.Collect(links
1607     ↪                 ↪ ToConnect[i]);
1608     ↪             }
1609     ↪         }
1610     ↪     }
1611     ↪     return results;
1612     ↪ });
1613     ↪ }
1614     ↪ public List<ulong>
1615     ↪ ↪ GetAllConnections9(params ulong[]
1616     ↪ ↪ linksToConnect)
1617     ↪ {
1618     ↪     return Sync.ExecuteReadOperation(()
1619     ↪     ↪ =>
1620     ↪     {
1621     ↪         var results = new BitString((lo
1622     ↪         ↪ ng)Links.Unsync.Count() +
1623     ↪         ↪ 1); // new
1624     ↪         ↪ BitArray((int)_links.Total
1625     ↪         ↪ + 1);
1626     ↪         if (linksToConnect.Length > 0)
1627     ↪         {
1628     ↪             Links.EnsureEachLinkExists(
1629     ↪             ↪ linksToConnect);
1630     ↪             var collector1 = new
1631     ↪             ↪ AllUsagesCollector2(Lin
1632     ↪             ↪ ks.Unsync,
1633     ↪             ↪ results);
1634     ↪             collector1.Collect(linksToC
1635     ↪             ↪ onnect[0]);
1636     ↪             for (var i = 1; i <
1637     ↪             ↪ linksToConnect.Length;
1638     ↪             ↪ i++)
1639     ↪             {
1640     ↪                 var collector = new
1641     ↪                 ↪ AllUsagesCollector(
1642     ↪                 ↪ Links.Unsync,
1643     ↪                 ↪ next);
1644     ↪                 collector.Collect(links
1645     ↪                 ↪ ToConnect[i]);
1646     ↪             }
1647     ↪         }
1648     ↪     }
1649     ↪     return results;
1650     ↪ });
1651     ↪ }
1652     ↪ public List<ulong>
1653     ↪ ↪ GetAllConnections10(params ulong[]
1654     ↪ ↪ linksToConnect)
1655     ↪ {
1656     ↪     return Sync.ExecuteReadOperation(()
1657     ↪     ↪ =>
1658     ↪     {
1659     ↪         var results = new BitString((lo
1660     ↪         ↪ ng)Links.Unsync.Count() +
1661     ↪         ↪ 1); // new
1662     ↪         ↪ BitArray((int)_links.Total
1663     ↪         ↪ + 1);
1664     ↪         if (linksToConnect.Length > 0)
1665     ↪         {
1666     ↪             Links.EnsureEachLinkExists(
1667     ↪             ↪ linksToConnect);
1668     ↪             var collector1 = new
1669     ↪             ↪ AllUsagesCollector2(Lin
1670     ↪             ↪ ks.Unsync,
1671     ↪             ↪ results);
1672     ↪             collector1.Collect(linksToC
1673     ↪             ↪ onnect[0]);
1674     ↪             for (var i = 1; i <
1675     ↪             ↪ linksToConnect.Length;
1676     ↪             ↪ i++)
1677     ↪             {
1678     ↪                 var collector = new
1679     ↪                 ↪ AllUsagesCollector(
1680     ↪                 ↪ Links.Unsync,
1681     ↪                 ↪ next);
1682     ↪                 collector.Collect(links
1683     ↪                 ↪ ToConnect[i]);
1684     ↪             }
1685     ↪         }
1686     ↪     }
1687     ↪     return results;
1688     ↪ });
1689     ↪ }
1690     ↪ public List<ulong>
1691     ↪ ↪ GetAllConnections11(params ulong[]
1692     ↪ ↪ linksToConnect)
1693     ↪ {
1694     ↪     return Sync.ExecuteReadOperation(()
1695     ↪     ↪ =>
1696     ↪     {
1697     ↪         var results = new BitString((lo
1698     ↪         ↪ ng)Links.Unsync.Count() +
1699     ↪         ↪ 1); // new
1700     ↪         ↪ BitArray((int)_links.Total
1701     ↪         ↪ + 1);
1702     ↪         if (linksToConnect.Length > 0)
1703     ↪         {
1704     ↪             Links.EnsureEachLinkExists(
1705     ↪             ↪ linksToConnect);
1706     ↪             var collector1 = new
1707     ↪             ↪ AllUsagesCollector2(Lin
1708     ↪             ↪ ks.Unsync,
1709     ↪             ↪ results);
1710     ↪             collector1.Collect(linksToC
1711     ↪             ↪ onnect[0]);
1712     ↪             for (var i = 1; i <
1713     ↪             ↪ linksToConnect.Length;
1714     ↪             ↪ i++)
1715     ↪             {
1716     ↪                 var collector = new
1717     ↪                 ↪ AllUsagesCollector(
1718     ↪                 ↪ Links.Unsync,
1719     ↪                 ↪ next);
1720     ↪                 collector.Collect(links
1721     ↪                 ↪ ToConnect[i]);
1722     ↪             }
1723     ↪         }
1724     ↪     }
1725     ↪     return results;
1726     ↪ });
1727     ↪ }
1728     ↪ public List<ulong>
1729     ↪ ↪ GetAllConnections12(params ulong[]
1730     ↪ ↪ linksToConnect)
1731     ↪ {
1732     ↪     return Sync.ExecuteReadOperation(()
1733     ↪     ↪ =>
1734     ↪     {
1735     ↪         var results = new BitString((lo
1736     ↪         ↪ ng)Links.Unsync.Count() +
1737     ↪         ↪ 1); // new
1738     ↪         ↪ BitArray((int)_links.Total
1739     ↪         ↪ + 1);
1740     ↪         if (linksToConnect.Length > 0)
1741     ↪         {
1742     ↪             Links.EnsureEachLinkExists(
1743     ↪             ↪ linksToConnect);
1744     ↪             var collector1 = new
1745     ↪             ↪ AllUsagesCollector2(Lin
1746     ↪             ↪ ks.Unsync,
1747     ↪             ↪ results);
1748     ↪             collector1.Collect(linksToC
1749     ↪             ↪ onnect[0]);
1750     ↪             for (var i = 1; i <
1751     ↪             ↪ linksToConnect.Length;
1752     ↪             ↪ i++)
1753     ↪             {
1754     ↪                 var collector = new
1755     ↪                 ↪ AllUsagesCollector(
1756     ↪                 ↪ Links.Unsync,
1757     ↪                 ↪ next);
1758     ↪                 collector.Collect(links
1759     ↪                 ↪ ToConnect[i]);
1760     ↪             }
1761     ↪         }
1762     ↪     }
1763     ↪     return results;
1764     ↪ });
1765     ↪ }
1766     ↪ public List<ulong>
1767     ↪ ↪ GetAllConnections13(params ulong[]
1768     ↪ ↪ linksToConnect)
1769     ↪ {
1770     ↪     return Sync.ExecuteReadOperation(()
1771     ↪     ↪ =>
1772     ↪     {
1773     ↪         var results = new BitString((lo
1774     ↪         ↪ ng)Links.Unsync.Count() +
1775     ↪         ↪ 1); // new
1776     ↪         ↪ BitArray((int)_links.Total
1777     ↪         ↪ + 1);
1778     ↪         if (linksToConnect.Length > 0)
1779     ↪         {
1780     ↪             Links.EnsureEachLinkExists(
1781     ↪             ↪ linksToConnect);
1782     ↪             var collector1 = new
1783     ↪             ↪ AllUsagesCollector2(Lin
1784     ↪             ↪ ks.Unsync,
1785     ↪             ↪ results);
1786     ↪             collector1.Collect(linksToC
1787     ↪             ↪ onnect[0]);
1788     ↪             for (var i = 1; i <
1789     ↪             ↪ linksToConnect.Length;
1790     ↪             ↪ i++)
1791     ↪             {
1792     ↪                 var collector = new
1793     ↪                 ↪ AllUsagesCollector(
1794     ↪                 ↪ Links.Unsync,
1795     ↪                 ↪ next);
1796     ↪                 collector.Collect(links
1797     ↪                 ↪ ToConnect[i]);
1798     ↪             }
1799     ↪         }
1800     ↪     }
1801     ↪     return results;
1802     ↪ });
1803     ↪ }
1804     ↪ public List<ulong>
1805     ↪ ↪ GetAllConnections14(params ulong[]
1806     ↪ ↪ linksToConnect)
1807     ↪ {
1808     ↪     return Sync.ExecuteReadOperation(()
1809     ↪     ↪ =>
1810     ↪     {
1811     ↪         var results = new BitString((lo
1812     ↪         ↪ ng)Links.Unsync.Count() +
1813     ↪         ↪ 1); // new
1814     ↪         ↪ BitArray((int)_links.Total
1815     ↪         ↪ + 1);
1816     ↪         if (linksToConnect.Length > 0)
1817     ↪         {
1818     ↪             Links.EnsureEachLinkExists(
1819     ↪             ↪ linksToConnect);
1820     ↪             var collector1 = new
1821     ↪             ↪ AllUsagesCollector2(Lin
1822     ↪             ↪ ks.Unsync,
1823     ↪             ↪ results);
1824     ↪             collector1.Collect(linksToC
1825     ↪             ↪ onnect[0]);
1826     ↪             for (var i = 1; i <
1827     ↪             ↪ linksToConnect.Length;
1828     ↪             ↪ i++)
1829     ↪             {
1830     ↪                 var collector = new
1831     ↪                 ↪ AllUsagesCollector(
1832     ↪                 ↪ Links.Unsync,
1833     ↪                 ↪ next);
1834     ↪                 collector.Collect(links
1835     ↪                 ↪ ToConnect[i]);
1836     ↪             }
1837     ↪         }
1838     ↪     }
1839     ↪     return results;
1840     ↪ });
1841     ↪ }
1842     ↪ public List<ulong>
1843     ↪ ↪ GetAllConnections15(params ulong[]
1844     ↪ ↪ linksToConnect)
1845     ↪ {
1846     ↪     return Sync.ExecuteReadOperation(()
1847     ↪     ↪ =>
1848     ↪     {
1849     ↪         var results = new BitString((lo
1850     ↪         ↪ ng)Links.Unsync.Count() +
1851     ↪         ↪ 1); // new
1852     ↪         ↪ BitArray((int)_links.Total
1853     ↪         ↪ + 1);
1854     ↪         if (linksToConnect.Length > 0)
1855     ↪         {
1856     ↪             Links.EnsureEachLinkExists(
1857     ↪             ↪ linksToConnect);
1858     ↪             var collector1 = new
1859     ↪             ↪ AllUsagesCollector2(Lin
1860     ↪             ↪ ks.Unsync,
1861     ↪             ↪ results);
1862     ↪             collector1.Collect(linksToC
1863     ↪             ↪ onnect[0]);
1864     ↪             for (var i = 1; i <
1865     ↪             ↪ linksToConnect.Length;
1866     ↪             ↪ i++)
1867     ↪             {
1868     ↪                 var collector = new
1869     ↪                 ↪ AllUsagesCollector(
1870     ↪                 ↪ Links.Unsync,
1871     ↪                 ↪ next);
1872     ↪                 collector.Collect(links
1873     ↪                 ↪ ToConnect[i]);
1874     ↪             }
1875     ↪         }
1876     ↪     }
1877     ↪     return results;
1878     ↪ });
1879     ↪ }
1880     ↪ public List<ulong>
1881     ↪ ↪ GetAllConnections16(params ulong[]
1882     ↪ ↪ linksToConnect)
1883     ↪ {
1884     ↪     return Sync.ExecuteReadOperation(()
1885     ↪     ↪ =>
1886     ↪     {
1887     ↪         var results = new BitString((lo
1888     ↪         ↪ ng)Links.Unsync.Count() +
1889     ↪         ↪ 1); // new
1890     ↪         ↪ BitArray((int)_links.Total
1891     ↪         ↪ + 1);
1892     ↪         if (linksToConnect.Length > 0)
1893     ↪         {
1894     ↪             Links.EnsureEachLinkExists(
1895     ↪             ↪ linksToConnect);
1896     ↪             var collector1 = new
1897     ↪             ↪ AllUsagesCollector2(Lin
1898     ↪             ↪ ks.Unsync,
1899     ↪             ↪ results);
1900     ↪             collector1.Collect(linksToC
1901     ↪             ↪ onnect[0]);
1902     ↪             for (var i = 1; i <
1903     ↪             ↪ linksToConnect.Length;
1904     ↪             ↪ i++)
1905     ↪             {
1906     ↪                 var collector = new
1907     ↪                 ↪ AllUsagesCollector(
1908     ↪                 ↪ Links.Unsync,
1909     ↪                 ↪ next);
1910     ↪                 collector.Collect(links
1911     ↪                 ↪ ToConnect[i]);
1912     ↪             }
1913     ↪         }
1914     ↪     }
1915     ↪     return results;
1916     ↪ });
1917     ↪ }
1918     ↪ public List<ulong>
1919     ↪ ↪ GetAllConnections17(params ulong[]
1920     ↪ ↪ linksToConnect)
1921     ↪ {
1922     ↪     return Sync.ExecuteReadOperation(()
1923     ↪     ↪ =>
1924     ↪     {
1925     ↪         var results = new BitString((lo
1926     ↪         ↪ ng)Links.Unsync.Count() +
1927     ↪         ↪ 1); // new
1928     ↪         ↪ BitArray((int)_links.Total
1929     ↪         ↪ + 1);
1930     ↪         if (linksToConnect.Length > 0)
1931     ↪         {
1932     ↪             Links.EnsureEachLinkExists(
1933     ↪             ↪ linksToConnect);
1934     ↪             var collector1 = new
1935     ↪             ↪ AllUsagesCollector2(Lin
1936     ↪             ↪ ks.Unsync,
1937     ↪             ↪ results);
1938     ↪             collector1.Collect(linksToC
1939     ↪             ↪ onnect[0]);
1940     ↪             for (var i = 1; i <
1941     ↪             ↪ linksToConnect.Length;
1942     ↪             ↪ i++)
1943     ↪             {
1944     ↪                 var collector = new
1945     ↪                 ↪ AllUsagesCollector(
1946     ↪                 ↪ Links.Unsync,
1947     ↪                 ↪ next);
1948     ↪                 collector.Collect(links
1949     ↪                 ↪ ToConnect[i]);
1950     ↪             }
1951     ↪         }
1952     ↪     }
1953     ↪     return results;
1954     ↪ });
1955     ↪ }
1956     ↪ public List<ulong>
1957     ↪ ↪ GetAllConnections18(params ulong[]
1958     ↪ ↪ linksToConnect)
1959     ↪ {
1960     ↪     return Sync.ExecuteReadOperation(()
1961     ↪     ↪ =>
1962     ↪     {
1963     ↪         var results = new BitString((lo
1964     ↪         ↪ ng)Links.Unsync.Count() +
1965     ↪         ↪ 1); // new
1966     ↪         ↪ BitArray((int)_links.Total
1967     ↪         ↪ + 1);
1968     ↪         if (linksToConnect.Length > 0)
1969     ↪         {
1970     ↪             Links.EnsureEachLinkExists(
1971     ↪             ↪ linksToConnect);
1972     ↪             var collector1 = new
1973     ↪             ↪ AllUsagesCollector2(Lin
1974     ↪             ↪ ks.Unsync,
1975     ↪             ↪ results);
1976     ↪             collector1.Collect(linksToC
1977     ↪             ↪ onnect[0]);
1978     ↪             for (var i = 1; i <
1979     ↪             ↪ linksToConnect.Length;
1980     ↪             ↪ i++)
1981     ↪             {
1982     ↪                 var collector = new
1983     ↪                 ↪ AllUsagesCollector(
1984     ↪                 ↪ Links.Unsync,
1985     ↪                 ↪ next);
1986     ↪                 collector.Collect(links
1987     ↪                 ↪ ToConnect[i]);
1988     ↪             }
1989     ↪         }
1990     ↪     }
1991     ↪     return results;
1992     ↪ });
1993     ↪ }
1994     ↪ public List<ulong>
1995     ↪ ↪ GetAllConnections19(params ulong[]
1996     ↪ ↪ linksToConnect)
1997     ↪ {
1998     ↪     return Sync.ExecuteReadOperation(()
1999     ↪     ↪ =>
2000     ↪     {
2001     ↪         var results = new BitString((lo
2002     ↪         ↪ ng)Links.Unsync.Count() +
2003     ↪         ↪ 1); // new
2004     ↪         ↪ BitArray((int)_links.Total
2005     ↪         ↪ + 1);
2006     ↪         if (linksToConnect.Length > 0)
2007     ↪         {
2008     ↪             Links.EnsureEachLinkExists(
2009     ↪             ↪ linksToConnect);
2010     ↪             var collector1 = new
2011     ↪             ↪ AllUsagesCollector2(Lin
2012     ↪             ↪ ks.Unsync,
2013     ↪             ↪ results);
2014     ↪             collector1.Collect(linksToC
2015     ↪             ↪ onnect[0]);
2016     ↪             for (var i = 1; i <
2017     ↪             ↪ linksToConnect.Length;
2018     ↪             ↪ i++)
2019     ↪             {
2020     ↪                 var collector = new
2021     ↪                 ↪ AllUsagesCollector(
2022     ↪                 ↪ Links.Unsync,
2023     ↪                 ↪ next);
2024     ↪                 collector.Collect(links
2025     ↪                 ↪ ToConnect[i]);
2026     ↪             }
2027     ↪         }
2028     ↪     }
2029     ↪     return results;
2030     ↪ });
2031     ↪ }
2032     ↪ public List<ulong>
2033     ↪ ↪ GetAllConnections20(params ulong[]
2034     ↪ ↪ linksToConnect)
2035     ↪ {
2036     ↪     return Sync.ExecuteReadOperation(()
2037     ↪     ↪ =>
2038     ↪     {
2039     ↪         var results = new BitString((lo
2040     ↪         ↪ ng)Links.Unsync.Count() +
2041     ↪         ↪ 1); // new
2042     ↪         ↪ BitArray((int)_links.Total
2043     ↪         ↪ + 1);
2044     ↪         if (linksToConnect.Length > 0)
2045     ↪         {
2046     ↪             Links.EnsureEachLinkExists(
2047     ↪             ↪ linksToConnect);
2048     ↪             var collector1 = new
2049     ↪             ↪ AllUsagesCollector2(Lin
2050     ↪             ↪ ks.Unsync,
2051     ↪             ↪ results);
2052     ↪             collector1.Collect(linksToC
2053     ↪             ↪ onnect[0]);
2054     ↪             for (var i = 1; i <
2055     ↪             ↪ linksToConnect.Length;
2056     ↪             ↪ i++)
2057     ↪             {
2058     ↪                 var collector = new
2059     ↪                 ↪ AllUsagesCollector(
2060     ↪                 ↪ Links.Unsync,
2061     ↪                 ↪ next);
2062     ↪                 collector.Collect(links
2063     ↪                 ↪ ToConnect[i]);
2064     ↪             }
2065     ↪         }
2066     ↪     }
2067     ↪     return results;
2068     ↪ });
2069     ↪ }
2070     ↪ public List<ulong>
2071     ↪ ↪ GetAllConnections21(params ulong[]
2072     ↪ ↪ linksToConnect)
2073     ↪ {
2074     ↪     return Sync.ExecuteReadOperation(()
2075     ↪     ↪ =>
2076     ↪     {
2077     ↪         var results = new BitString((lo
2078     ↪         ↪ ng)Links.Unsync.Count() +
2079     ↪         ↪ 1); // new
2080     ↪         ↪ BitArray((int)_links.Total
2081     ↪         ↪ + 1);
2082     ↪         if (linksToConnect.Length > 0)
2083     ↪         {
2084     ↪             Links.EnsureEachLinkExists(
2085     ↪             ↪ linksToConnect);
2086     ↪             var collector1 = new
2087     ↪             ↪ AllUsagesCollector2(Lin
2088     ↪             ↪ ks.Unsync,
2089     ↪             ↪ results);
2090     ↪             collector1.Collect(linksToC
2091     ↪             ↪ onnect[0]);
2092     ↪             for (var i = 1; i <
2093     ↪             ↪ linksToConnect.Length;
2094     ↪             ↪ i++)
2095     ↪             {
2096     ↪                 var collector = new
2097     ↪                 ↪ AllUsagesCollector(
2098     ↪                 ↪ Links.Unsync,
2099     ↪                 ↪ next);
2100     ↪                 collector.Collect(links
2101     ↪                 ↪ ToConnect[i]);
2102     ↪             }
2103     ↪         }
2104     ↪     }
2105     ↪     return results;
2106     ↪ });
2107     ↪ }
2108     ↪ public List<ulong>
2109     ↪ ↪ GetAllConnections22(params ulong[]
2110     ↪ ↪ linksToConnect)
2111     ↪ {
2112     ↪     return Sync.ExecuteReadOperation(()
2113     ↪     ↪ =>
2114     ↪     {
2115     ↪         var results = new BitString((lo
2116     ↪         ↪ ng)Links.Unsync.Count() +
2117     ↪         ↪ 1); // new
2118     ↪         ↪ BitArray((int)_links.Total
2119     ↪         ↪ + 1);
2120     ↪         if (linksToConnect.Length > 0)
2121     ↪         {
2122     ↪             Links.EnsureEachLinkExists(
2123     ↪             ↪ linksToConnect);
2124     ↪             var collector1 = new
2125     ↪             ↪ AllUsagesCollector2(Lin
2126     ↪             ↪ ks.Unsync,
2127     ↪             ↪ results);
2128     ↪             collector1.Collect(linksToC
2129     ↪             ↪ onnect[0]);
2130     ↪             for (var i = 1; i <
2131     ↪             ↪ linksToConnect.Length;
2132     ↪             ↪ i++)
2133     ↪             {
2134     ↪                 var collector = new
2135     ↪                 ↪ AllUsagesCollector(
2136     ↪                 ↪ Links.Unsync,
2137     ↪                 ↪ next);
2138     ↪                 collector.Collect(links
2139     ↪                 ↪ ToConnect[i]);
2140     ↪             }
2141     ↪         }
2142     ↪     }
2143     ↪     return results;
2144     ↪ });
2145     ↪ }
2146     ↪ public List<ulong>
2147     ↪ ↪ GetAllConnections23(params ulong[]
2148     ↪ ↪ linksToConnect)
2149     ↪ {
2150     ↪     return Sync.ExecuteReadOperation(()
2151     ↪     ↪ =>
2152     ↪     {
2153     ↪         var results = new BitString((lo
2154     ↪         ↪ ng)Links.Unsync.Count() +
2155     ↪         ↪ 1); // new
2156     ↪         ↪ BitArray((int)_links.Total
2157     ↪         ↪ + 1);
2158     ↪         if (linksToConnect.Length > 0)
2159     ↪         {
2160     ↪             Links.EnsureEachLinkExists(
2161     ↪             ↪ linksToConnect);
2162     ↪             var collector1 = new
2163     ↪             ↪ AllUsages
```

```

1380         var next = new
1381             BitString((long)Links.Unsync.Count() +
1382                 1); //new BitArray(
1383                 (int)_links.Total +
1384                 1);
1385         var collector = new
1386             AllUsagesCollector2
1387             (Links.Unsync,
1388             next);
1389         collector.Collect(links
1390             ToConnect[i]);
1391         results =
1392             results.And(next);
1393     }
1394 }
1395 return results.GetSetUInt64Indices();
1396 }
1397
1398 private static ulong[] Simplify(ulong[]
1399     sequence)
1400 {
1401     // Считаем новый размер
1402     // последовательности
1403     long newLength = 0;
1404     var zeroOrManyStepped = false;
1405     for (var i = 0; i <
1406         sequence.Length; i++)
1407     {
1408         if (sequence[i] == ZeroOrMany)
1409         {
1410             if (zeroOrManyStepped)
1411             {
1412                 continue;
1413             }
1414             zeroOrManyStepped = true;
1415         }
1416         else
1417         {
1418             //if (zeroOrManyStepped) Is
1419             // it efficient?
1420             zeroOrManyStepped = false;
1421         }
1422         newLength++;
1423     }
1424     // Строим новую последовательность
1425     zeroOrManyStepped = false;
1426     var newSequence = new
1427         ulong[newLength];
1428     long j = 0;
1429     for (var i = 0; i <
1430         sequence.Length; i++)
1431     {
1432         //var current =
1433         // zeroOrManyStepped;
1434         //zeroOrManyStepped =
1435         // patternSequence[i] ==
1436         // zeroOrMany;
1437         //if (current &&
1438         // zeroOrManyStepped)
1439         // continue;
1440         //var newZeroOrManyStepped =
1441         // patternSequence[i] ==
1442         // zeroOrMany;
1443         //if (zeroOrManyStepped &&
1444         // newZeroOrManyStepped)
1445         // continue;
1446         //zeroOrManyStepped =
1447         // newZeroOrManyStepped;
1448         if (sequence[i] == ZeroOrMany)
1449         {
1450             if (zeroOrManyStepped)
1451             {
1452                 continue;
1453             }
1454             zeroOrManyStepped = true;
1455         }
1456         else
1457         {
1458             //if (zeroOrManyStepped) Is
1459             // it efficient?
1460             zeroOrManyStepped = false;
1461         }
1462         newSequence[j++] = sequence[i];
1463     }
1464 }
1465
1466 }
1467
1468 }
1469
1470 }
1471
1472 }
1473
1474 }
1475
1476 }
1477
1478 }
1479
1480 }
1481
1482 }
1483
1484 }
1485
1486 }
1487
1488 }
1489
1490 }
1491
1492 }
1493
1494 }
1495
1496 }
1497
1498 }
1499
1500 }
1501
1502 }
1503
1504 }
1505
1506 }
1507
1508 }
1509
1510 }
1511
1512 }
1513
1514 }
1515
1516 }
1517
1518 }
1519
1520 }
1521
1522 }
1523
1524 }
1525
1526 }
1527
1528 }
1529
1530 }
1531
1532 }
1533
1534 }
1535
1536 }
1537
1538 }
1539
1540 }
1541
1542 }
1543
1544 }
1545
1546 }
1547
1548 }
1549
1550 }
1551
1552 }
1553
1554 }
1555
1556 }
1557
1558 }
1559
1560 }
1561
1562 }
1563
1564 }
1565
1566 }
1567
1568 }
1569
1570 }
1571
1572 }
1573
1574 }
1575
1576 }
1577
1578 }
1579
1580 }
1581
1582 }
1583
1584 }
1585
1586 }
1587
1588 }
1589
1590 }
1591
1592 }
1593
1594 }
1595
1596 }
1597
1598 }
1599
1600 }
1601
1602 }
1603
1604 }
1605
1606 }
1607
1608 }
1609
1610 }
1611
1612 }
1613
1614 }
1615
1616 }
1617
1618 }
1619
1620 }
1621
1622 }
1623
1624 }
1625
1626 }
1627
1628 }
1629
1630 }
1631
1632 }
1633
1634 }
1635
1636 }
1637
1638 }
1639
1640 }
1641
1642 }
1643
1644 }
1645
1646 }
1647
1648 }
1649
1650 }
1651
1652 }
1653
1654 }
1655
1656 }
1657
1658 }
1659
1660 }
1661
1662 }
1663
1664 }
1665
1666 }
1667
1668 }
1669
1670 }
1671
1672 }
1673
1674 }
1675
1676 }
1677
1678 }
1679
1680 }
1681
1682 }
1683
1684 }
1685
1686 }
1687
1688 }
1689
1690 }
1691
1692 }
1693
1694 }
1695
1696 }
1697
1698 }
1699
1700 }
1701
1702 }
1703
1704 }
1705
1706 }
1707
1708 }
1709
1710 }
1711
1712 }
1713
1714 }
1715
1716 }
1717
1718 }
1719
1720 }
1721
1722 }
1723
1724 }
1725
1726 }
1727
1728 }
1729
1730 }
1731
1732 }
1733
1734 }
1735
1736 }
1737
1738 }
1739
1740 }
1741
1742 }
1743
1744 }
1745
1746 }
1747
1748 }
1749
1750 }
1751
1752 }
1753
1754 }
1755
1756 }
1757
1758 }
1759
1760 }
1761
1762 }
1763
1764 }
1765
1766 }
1767
1768 }
1769
1770 }
1771
1772 }
1773
1774 }
1775
1776 }
1777
1778 }
1779
1780 }
1781
1782 }
1783
1784 }
1785
1786 }
1787
1788 }
1789
1790 }
1791
1792 }
1793
1794 }
1795
1796 }
1797
1798 }
1799
1800 }
1801
1802 }
1803
1804 }
1805
1806 }
1807
1808 }
1809
1810 }
1811
1812 }
1813
1814 }
1815
1816 }
1817
1818 }
1819
1820 }
1821
1822 }
1823
1824 }
1825
1826 }
1827
1828 }
1829
1830 }
1831
1832 }
1833
1834 }
1835
1836 }
1837
1838 }
1839
1840 }
1841
1842 }
1843
1844 }
1845
1846 }
1847
1848 }
1849
1850 }
1851
1852 }
1853
1854 }
1855
1856 }
1857
1858 }
1859
1860 }
1861
1862 }
1863
1864 }
1865
1866 }
1867
1868 }
1869
1870 }
1871
1872 }
1873
1874 }
1875
1876 }
1877
1878 }
1879
1880 }
1881
1882 }
1883
1884 }
1885
1886 }
1887
1888 }
1889
1890 }
1891
1892 }
1893
1894 }
1895
1896 }
1897
1898 }
1899
1900 }
1901
1902 }
1903
1904 }
1905
1906 }
1907
1908 }
1909
1910 }
1911
1912 }
1913
1914 }
1915
1916 }
1917
1918 }
1919
1920 }
1921
1922 }
1923
1924 }
1925
1926 }
1927
1928 }
1929
1930 }
1931
1932 }
1933
1934 }
1935
1936 }
1937
1938 }
1939
1940 }
1941
1942 }
1943
1944 }
1945
1946 }
1947
1948 }
1949
1950 }
1951
1952 }
1953
1954 }
1955
1956 }
1957
1958 }
1959
1960 }
1961
1962 }
1963
1964 }
1965
1966 }
1967
1968 }
1969
1970 }
1971
1972 }
1973
1974 }
1975
1976 }
1977
1978 }
1979
1980 }
1981
1982 }
1983
1984 }
1985
1986 }
1987
1988 }
1989
1990 }
1991
1992 }
1993
1994 }
1995
1996 }
1997
1998 }
1999
2000 }

```



```

1494 CollectMatchingSequences(element, leftBound + 1, middleLinks, rightLink, rightBound, ref results);
1495 }
1496 }
1497 }
1498 else
1499 {
1500     for (var i = elements.Length - 1; i >= 0; i--)
1501     {
1502         var element = elements[i];
1503         if (element != 0)
1504         {
1505             results.Add(element);
1506         }
1507     }
1508 }
1509 }
1510 else
1511 {
1512     var nextRightLink = middleLinks[rightBound];
1513     var elements = GetLeftElements(rightLink, nextRightLink);
1514     if (leftBound <= rightBound)
1515     {
1516         for (var i = elements.Length - 1; i >= 0; i--)
1517         {
1518             var element = elements[i];
1519             if (element != 0)
1520             {
1521                 CollectMatchingSequences(leftLink, leftBound, middleLinks, elements[i], rightBound - 1, ref results);
1522             }
1523         }
1524     }
1525     else
1526     {
1527         for (var i = elements.Length - 1; i >= 0; i--)
1528         {
1529             var element = elements[i];
1530             if (element != 0)
1531             {
1532                 results.Add(element);
1533             }
1534         }
1535     }
1536 }
1537 }
1538 }
1539 public ulong[] GetRightElements(ulong startLink, ulong rightLink)
1540 {
1541     var result = new ulong[5];
1542     TryStepRight(startLink, rightLink, result, 0);
1543     Links.Each(_constants.Any, startLink, couple =>
1544     {
1545         if (couple != startLink)
1546         {
1547             if (TryStepRight(couple, rightLink, result, 2))
1548             {
1549                 return false;

```

```

1550     }
1551     }
1552     return true;
1553 });
1554 if (Links.GetTarget(Links.GetTarget(startLink)) == rightLink)
1555 {
1556     result[4] = startLink;
1557 }
1558 return result;
1559 }
1560 public bool TryStepRight(ulong startLink, ulong rightLink, ulong[] result, int offset)
1561 {
1562     var added = 0;
1563     Links.Each(startLink, _constants.Any, couple =>
1564     {
1565         if (couple != startLink)
1566         {
1567             var coupleTarget = Links.GetTarget(couple);
1568             if (coupleTarget == rightLink)
1569             {
1570                 result[offset] = couple;
1571                 if (++added == 2)
1572                 {
1573                     return false;
1574                 }
1575             }
1576             else if (Links.GetSource(coupleTarget) == rightLink) // coupleTarget.Linker == Net.And &&
1577             {
1578                 result[offset + 1] = couple;
1579                 if (++added == 2)
1580                 {
1581                     return false;
1582                 }
1583             }
1584         }
1585     }
1586     return true;
1587 });
1588 return added > 0;
1589 }
1590 public ulong[] GetLeftElements(ulong startLink, ulong leftLink)
1591 {
1592     var result = new ulong[5];
1593     TryStepLeft(startLink, leftLink, result, 0);
1594     Links.Each(startLink, _constants.Any, couple =>
1595     {
1596         if (couple != startLink)
1597         {
1598             if (TryStepLeft(couple, leftLink, result, 2))
1599             {
1600                 return false;
1601             }
1602         }
1603     }
1604     return true;
1605 });
1606 if (Links.GetSource(Links.GetSource(leftLink)) == startLink)
1607 {
1608     result[4] = leftLink;
1609 }
1610 return result;
1611 }
1612 public bool TryStepLeft(ulong startLink, ulong leftLink, ulong[] result, int offset)
1613 {

```

```

1615     var added = 0;
1616     Links.Each(_constants.Any,
1617         ↪ startLink, couple =>
1618     {
1619         if (couple != startLink)
1620         {
1621             var coupleSource =
1622                 ↪ Links.GetSource(couple);
1623             if (coupleSource ==
1624                 ↪ leftLink)
1625             {
1626                 result[offset] = couple;
1627                 if (++added == 2)
1628                 {
1629                     return false;
1630                 }
1631             }
1632             else if (Links.GetTarget(co
1633                 ↪ upleSource) ==
1634                 ↪ leftLink) //
1635                 ↪ coupleSource.Linker ==
1636                 ↪ Net.And &&
1637             {
1638                 result[offset + 1] =
1639                 ↪ couple;
1640                 if (++added == 2)
1641                 {
1642                     return false;
1643                 }
1644             }
1645         }
1646         return true;
1647     });
1648     return added > 0;
1649 }
1650
1651 #endregion
1652
1653 #region Walkers
1654
1655 public class PatternMatcher :
1656     ↪ RightSequenceWalker<ulong>
1657 {
1658     private readonly Sequences
1659     ↪ _sequences;
1660     private readonly ulong[]
1661     ↪ _patternSequence;
1662     private readonly HashSet<LinkIndex>
1663     ↪ _linksInSequence;
1664     private readonly HashSet<LinkIndex>
1665     ↪ _results;
1666
1667     #region Pattern Match
1668
1669     enum PatternBlockType
1670     {
1671         Undefined,
1672         Gap,
1673         Elements
1674     }
1675
1676     struct PatternBlock
1677     {
1678         public PatternBlockType Type;
1679         public long Start;
1680         public long Stop;
1681     }
1682
1683     private readonly List<PatternBlock>
1684     ↪ _pattern;
1685     private int _patternPosition;
1686     private long _sequencePosition;
1687
1688     #endregion
1689
1690     public PatternMatcher(Sequences
1691     ↪ sequences, LinkIndex[]
1692     ↪ patternSequence,
1693     ↪ HashSet<LinkIndex> results)
1694     : base(sequences.Links.Unsync)
1695     {
1696         _sequences = sequences;
1697         _patternSequence =
1698             ↪ patternSequence;
1699         _linksInSequence = new
1700             ↪ HashSet<LinkIndex>(patternS
1701             ↪ equence.Where(x => x !=
1702             ↪ _constants.Any && x !=
1703             ↪ ZeroOrMany));
1704
1705         _results = results;
1706         _pattern =
1707             ↪ CreateDetailedPattern();
1708     }
1709
1710     protected override bool
1711     IsElement(IList<ulong> link) =>
1712     ↪ _linksInSequence.Contains(Links
1713     ↪ .GetIndex(link)) ||
1714     ↪ base.IsElement(link);
1715
1716     public bool PatternMatch(LinkIndex
1717     ↪ sequenceToMatch)
1718     {
1719         _patternPosition = 0;
1720         _sequencePosition = 0;
1721         foreach (var part in
1722             ↪ Walk(sequenceToMatch))
1723         {
1724             if (!PatternMatchCore(Links
1725             ↪ .GetIndex(part)))
1726             {
1727                 break;
1728             }
1729         }
1730         return _patternPosition ==
1731             ↪ _pattern.Count ||
1732             ↪ (_patternPosition ==
1733             ↪ _pattern.Count - 1 && _patt
1734             ↪ ern[_patternPosition].Start
1735             ↪ == 0);
1736     }
1737
1738     private List<PatternBlock>
1739     ↪ CreateDetailedPattern()
1740     {
1741         var pattern = new
1742             ↪ List<PatternBlock>();
1743         var patternBlock = new
1744             ↪ PatternBlock();
1745         for (var i = 0; i <
1746             ↪ _patternSequence.Length;
1747             ↪ i++)
1748         {
1749             if (patternBlock.Type ==
1750                 ↪ PatternBlockType.Undefi
1751                 ↪ ned)
1752             {
1753                 if (_patternSequence[i]
1754                 ↪ == _constants.Any)
1755                 {
1756                     patternBlock.Type =
1757                     ↪ PatternBlockTyp
1758                     ↪ e.Gap;
1759                     patternBlock.Start
1760                     ↪ = 1;
1761                     patternBlock.Stop =
1762                     ↪ 1;
1763                 }
1764                 else if (_patternSequen
1765                 ↪ ce[i] ==
1766                 ↪ ZeroOrMany)
1767                 {
1768                     patternBlock.Type =
1769                     ↪ PatternBlockTyp
1770                     ↪ e.Gap;
1771                     patternBlock.Start
1772                     ↪ = 0;
1773                     patternBlock.Stop =
1774                     ↪ long.MaxValue;
1775                 }
1776                 else
1777                 {
1778                     patternBlock.Type =
1779                     ↪ PatternBlockTyp
1780                     ↪ e.Elements;
1781                     patternBlock.Start
1782                     ↪ = i;
1783                     patternBlock.Stop =
1784                     ↪ i;
1785                 }
1786             }
1787             else if (patternBlock.Type
1788             ↪ == PatternBlockType.Ele
1789             ↪ ments)
1790             {
1791             }
1792         }
1793     }

```

```

1730 {
1731     if (_patternSequence[i]
1732         == _constants.Any)
1733     {
1734         pattern.Add(pattern
1735             Block);
1736         patternBlock = new
1737             PatternBlock
1738         {
1739             Type =
1740                 PatternBloc
1741             kType.Gap,
1742             Start = 1,
1743             Stop = 1
1744         };
1745     }
1746     else if (_patternSequen
1747         ce[i] ==
1748         ZeroOrMany)
1749     {
1750         pattern.Add(pattern
1751             Block);
1752         patternBlock = new
1753             PatternBlock
1754         {
1755             Type =
1756                 PatternBloc
1757             kType.Gap,
1758             Start = 0,
1759             Stop = long.Max
1760             Value
1761         };
1762     }
1763     else
1764     {
1765         patternBlock.Stop =
1766             i;
1767     }
1768 }
1769 else // patternBlock.Type
1770     == PatternBlockType.Gap
1771 {
1772     if (_patternSequence[i]
1773         == _constants.Any)
1774     {
1775         patternBlock.Start+
1776             +;
1777         if (patternBlock.St
1778             op <
1779             patternBlock.St
1780             art)
1781         {
1782             patternBlock.St
1783                 op =
1784                 patternBloc
1785                 k.Start;
1786         }
1787     }
1788     else if (_patternSequen
1789         ce[i] ==
1790         ZeroOrMany)
1791     {
1792         patternBlock.Stop =
1793             long.MaxValue;
1794     }
1795     else
1796     {
1797         pattern.Add(pattern
1798             Block);
1799         patternBlock = new
1800             PatternBlock
1801         {
1802             Type = PatternB
1803             lockType.El
1804             ements,
1805             Start = i,
1806             Stop = i
1807         };
1808     }
1809 }
1810 }
1811 if (patternBlock.Type !=
1812     PatternBlockType.Undefined)
1813 {
1814     pattern.Add(patternBlock);
1815 }
1816
1817 return pattern;
1818
1819 /** match: search for regexp
1820     anywhere in text */
1821 //int match(char* regexp, char*
1822     text)
1823 //{
1824 //    do
1825 //    {
1826 //        while (*text++ != '\0');
1827 //        return 0;
1828 //    }
1829
1830 /** matchhere: search for regexp
1831     at beginning of text */
1832 //int matchhere(char* regexp, char*
1833     text)
1834 //{
1835 //    if (regexp[0] == '\0')
1836 //        return 1;
1837 //    if (regexp[1] == '*')
1838 //        return
1839 //        matchstar(regexp[0], regexp +
1840 //            2, text);
1841 //    if (regexp[0] == '$' &&
1842 //        regexp[1] == '\0')
1843 //        return *text == '\0';
1844 //    if (*text != '\0' &&
1845 //        (regexp[0] == '.' || regexp[0]
1846 //            == *text))
1847 //        return matchhere(regexp +
1848 //            1, text + 1);
1849 //    return 0;
1850 //}
1851
1852 /** matchstar: search for c*regexp
1853     at beginning of text */
1854 //int matchstar(int c, char*
1855     regexp, char* text)
1856 //{
1857 //    do
1858 //    {
1859 //        /* a * matches zero or
1860 //        more instances */
1861 //        if (matchhere(regexp,
1862 //            text))
1863 //            return 1;
1864 //        while (*text != '\0' &&
1865 //            (*text++ == c || c == '.'));
1866 //        return 0;
1867 //    }
1868
1869 //private void
1870     GetNextPatternElement(out
1871         LinkIndex element, out long
1872         mininumGap, out long maximumGap)
1873 //{
1874 //    mininumGap = 0;
1875 //    maximumGap = 0;
1876 //    element = 0;
1877 //    for (; _patternPosition <
1878 //        _patternSequence.Length;
1879 //        _patternPosition++)
1880 //    {
1881 //        if (_patternSequence[_pat
1882 //            ternPosition] ==
1883 //            Doublets.Links.Null)
1884 //            mininumGap++;
1885 //        else if (_patternSequence
1886 //            [_patternPosition] ==
1887 //            ZeroOrMany)
1888 //            maximumGap =
1889 //            long.MaxValue;
1890 //        else
1891 //            break;
1892 //    }
1893
1894 //    if (maximumGap < mininumGap)
1895 //        maximumGap = mininumGap;
1896 //}
1897
1898 private bool
1899     PatternMatchCore(LinkIndex
1900         element)

```

```

1843 {
1844     if (_patternPosition >=
1845         ↪ _pattern.Count)
1846     {
1847         _patternPosition = -2;
1848         return false;
1849     }
1850     var currentPatternBlock =
1851         ↪ _pattern[_patternPosition];
1852     if (currentPatternBlock.Type ==
1853         ↪ PatternBlockType.Gap)
1854     {
1855         //var currentMatchingBlockLength =
1856         ↪ (_sequencePosition -
1857         ↪ _lastMatchedBlockPosition);
1858         if (_sequencePosition < currentPatternBlock.Start)
1859         {
1860             _sequencePosition++;
1861             return true; // Двигаемся дальше
1862         }
1863         // Это последний блок
1864         if (_pattern.Count ==
1865             ↪ _patternPosition + 1)
1866         {
1867             _patternPosition++;
1868             _sequencePosition = 0;
1869             return false; // Полное
1870             ↪ соответствие
1871         }
1872         else
1873         {
1874             if (_sequencePosition >
1875                 ↪ currentPatternBlock
1876                 ↪ .Stop)
1877             {
1878                 return false; //
1879                 ↪ Соответствие
1880                 ↪ невозможно
1881             }
1882             var nextPatternBlock =
1883                 ↪ _pattern[_patternPosition +
1884                 ↪ 1];
1885             if (_patternSequence[nextPatternBlock.Start] ==
1886                 ↪ element)
1887             {
1888                 if (nextPatternBlock.Start <
1889                     ↪ nextPatternBlock
1890                     ↪ .Stop)
1891                 {
1892                     _patternPosition++;
1893                     _sequencePosition = 0;
1894                 }
1895                 else
1896                 {
1897                     _patternPosition++;
1898                     _sequencePosition = 0;
1899                 }
1900             }
1901         }
1902     }
1903     else //
1904         ↪ currentPatternBlock.Type ==
1905         ↪ PatternBlockType.Elements
1906     {
1907         var patternElementPosition = currentPatternBlock.Start +
1908             ↪ _sequencePosition;
1909         if (_patternSequence[patternElementPosition] !=
1910             ↪ element)
1911         {
1912             return false; //
1913             ↪ Соответствие
1914             ↪ невозможно
1915         }
1916         _patternPosition++;
1917         _sequencePosition++;
1918         return true;
1919     }
1920     //if (_patternSequence[_patternPosition] !=
1921     ↪ element)
1922     //    return false;
1923     //else
1924     //{
1925     //    _sequencePosition++;
1926     //    _patternPosition++;
1927     //    return true;
1928     //}
1929     //if (_filterPosition ==
1930     ↪ _patternSequence.Length)
1931     //{
1932     //    _filterPosition = -2; //
1933     ↪ Длиннее чем нужно
1934     //    return false;
1935     //}
1936     //if (element != _patternSequence[_filterPosition])
1937     //{
1938     //    _filterPosition = -1;
1939     //    return false; //
1940     ↪ Начинается иначе
1941     //}
1942     _filterPosition++;
1943     //if (_filterPosition ==
1944     ↪ (_patternSequence.Length -
1945     ↪ 1))
1946     //    return false;
1947     //if (_filterPosition >= 0)
1948     //{
1949     //    if (element == _patternSequence[_filterPosition +
1950     ↪ 1])
1951     //        _filterPosition++;
1952     //    else
1953     //        return false;
1954     //}
1955     //if (_filterPosition < 0)
1956     //{
1957     //    if (element ==
1958     ↪ _patternSequence[0])
1959     //        _filterPosition = 0;
1960     //}
1961 }
1962
1963 public void AddAllPatternMatchedToResults(
1964     ↪ IEnumerable<ulong>
1965     ↪ sequencesToMatch)
1966 {
1967     foreach (var sequenceToMatch in
1968         ↪ sequencesToMatch)
1969     {
1970         if (PatternMatch(sequenceToMatch,
1971             ↪ Match))
1972         {
1973             _results.Add(sequenceToMatch);
1974         }
1975     }
1976 }
1977
1978 #endregion
1979 }
1980

```

./Sequences/Sequences.Experiments.ReadSequence.cs

```
1  //define USEARRAYPOOL
2  using System;
3  using System.Runtime.CompilerServices;
4  #if USEARRAYPOOL
5  using Platform.Collections;
6  #endif
7
8  namespace Platform.Data.Doublets.Sequences
9  {
10     partial class Sequences
11     {
12         public ulong[] ReadSequenceCore(ulong
13             ↪ sequence, Func<ulong, bool>
14             ↪ isElement)
15         {
16             var links = Links.Unsync;
17             var length = 1;
18             var array = new ulong[length];
19             array[0] = sequence;
20
21             if (isElement(sequence))
22             {
23                 return array;
24             }
25
26             bool hasElements;
27             do
28             {
29                 length *= 2;
30                 #if USEARRAYPOOL
31                     var nextArray = ArrayPool.Alloc
32                     ↪ ate<ulong>(length);
33                 #else
34                     var nextArray = new
35                     ↪ ulong[length];
36                 #endif
37
38                 hasElements = false;
39                 for (var i = 0; i <
40                     ↪ array.Length; i++)
41                 {
42                     var candidate = array[i];
43                     if (candidate == 0)
44                     {
45                         continue;
46                     }
47                     var doubletOffset = i * 2;
48                     if (isElement(candidate))
49                     {
50                         nextArray[doubletOffset
51                             ↪ ] =
52                             ↪ candidate;
53                     }
54                     else
55                     {
56                         var link = links.GetLin
57                         ↪ k(candidate);
58                         var linkSource = links.
59                         ↪ GetSource(link);
60                         var linkTarget = links.
61                         ↪ GetTarget(link);
62                         nextArray[doubletOffset
63                             ↪ ] =
64                             ↪ linkSource;
65                         nextArray[doubletOffset
66                             ↪ + 1] = linkTarget;
67                         if (!hasElements)
68                         {
69                             hasElements =
70                             ↪ !(isElement(link
71                             ↪ Source) &&
72                             ↪ isElement(linkT
73                             ↪ arget));
74                         }
75                     }
76                 }
77             }
78             #if USEARRAYPOOL
79                 if (array.Length > 1)
80                 {
81                     ArrayPool.Free(array);
82                 }
83             #endif
84             array = nextArray;
85         }
86         while (hasElements);
87         var filledElementsCount =
88         ↪ CountFilledElements(array);
```

```
69         if (filledElementsCount ==
70             ↪ array.Length)
71         {
72             return array;
73         }
74         else
75         {
76             return
77             ↪ CopyFilledElements(array,
78             ↪ filledElementsCount);
79         }
80     }
81
82     [MethodImpl(MethodImplOptions.Aggressive
83     ↪ eInlining)]
84     private static ulong[]
85     ↪ CopyFilledElements(ulong[] array,
86     ↪ int filledElementsCount)
87     {
88         var finalArray = new
89         ↪ ulong[filledElementsCount];
90         for (int i = 0, j = 0; i <
91             ↪ array.Length; i++)
92         {
93             if (array[i] > 0)
94             {
95                 finalArray[j] = array[i];
96                 j++;
97             }
98         }
99         #if USEARRAYPOOL
100             ArrayPool.Free(array);
101         #endif
102         return finalArray;
103     }
104
105     [MethodImpl(MethodImplOptions.Aggressive
106     ↪ eInlining)]
107     private static int
108     ↪ CountFilledElements(ulong[] array)
109     {
110         var count = 0;
111         for (var i = 0; i < array.Length;
112             ↪ i++)
113         {
114             if (array[i] > 0)
115             {
116                 count++;
117             }
118         }
119         return count;
120     }
121 }
```

./Sequences/SequencesExtensions.cs

```
1  using Platform.Data.Sequences;
2  using System.Collections.Generic;
3
4  namespace Platform.Data.Doublets.Sequences
5  {
6     public static class SequencesExtensions
7     {
8         public static TLink Create<TLink>(this
9             ↪ ISequences<TLink> sequences,
10             ↪ IList<TLink[]> groupedSequence)
11         {
12             var finalSequence = new
13             ↪ TLink[groupedSequence.Count];
14             for (var i = 0; i <
15                 ↪ finalSequence.Length; i++)
16             {
17                 var part = groupedSequence[i];
18                 finalSequence[i] = part.Length
19                 ↪ == 1 ? part[0] :
20                 ↪ sequences.Create(part);
21             }
22             return
23             ↪ sequences.Create(finalSequence);
24         }
25     }
26 }
```

./Sequences/SequencesIndexer.cs

```
1  using System.Collections.Generic;
2
```

```

3 namespace Platform.Data.Doublets.Sequences
4 {
5     public class SequencesIndexer<TLink>
6     {
7         private static readonly
8             EqualityComparer<TLink>
9             equalityComparer =
10             EqualityComparer<TLink>.Default;
11
12         private readonly
13             ISynchronizedLinks<TLink> _links;
14         private readonly TLink _null;
15
16         public SequencesIndexer(ISynchronizedLi
17             nks<TLink>
18             links)
19         {
20             _links = links;
21             _null = _links.Constants.Null;
22         }
23
24         /// <summary>
25         /// Индексирует последовательность
26         /// глобально, и возвращает значение,
27         /// определяющие была ли запрошенная
28         /// последовательность проиндексирована
29         /// ранее.
30         /// </summary>
31         /// <param
32         /// name="sequence">Последовательность
33         /// для индексации.</param>
34         /// <returns>
35         /// True если последовательность уже
36         /// была проиндексирована ранее и
37         /// False если последовательность была
38         /// проиндексирована только что.
39         /// </returns>
40         public bool Index(TLink[] sequence)
41         {
42             var indexed = true;
43             var i = sequence.Length;
44             while (--i >= 1 && (indexed =
45                 !_equalityComparer.Equals(_link
46                 s.SearchOrDefault(sequence[i -
47                 1], sequence[i]), _null))) { }
48             for (; i >= 1; i--)
49             {
50                 _links.GetOrCreate(sequence[i -
51                     1], sequence[i]);
52             }
53             return indexed;
54         }
55
56         public bool BulkIndex(TLink[] sequence)
57         {
58             var indexed = true;
59             var i = sequence.Length;
60             var links = _links.Unsync;
61             _links.SyncRoot.ExecuteReadOperatio
62                 n(()
63                 =>
64                 {
65                     while (--i >= 1 && (indexed =
66                         !_equalityComparer.Equals(1
67                         nks.SearchOrDefault(sequen
68                         ce[i - 1], sequence[i]),
69                         _null))) { }
70                     });
71             if (indexed == false)
72             {
73                 _links.SyncRoot.ExecuteWriteOpe
74                     ration(()
75                     =>
76                     {
77                         for (; i >= 1; i--)
78                         {
79                             links.GetOrCreate(seque
80                                 nce[i - 1],
81                                 sequence[i]);
82                         }
83                     });
84             }
85             return indexed;
86         }
87
88         public bool BulkIndexUnsync(TLink[]
89             sequence)

```

```

62     {
63         var indexed = true;
64         var i = sequence.Length;
65         var links = _links.Unsync;
66         while (--i >= 1 && (indexed =
67             !_equalityComparer.Equals(links
68             .SearchOrDefault(sequence[i -
69             1], sequence[i]), _null))) { }
70         for (; i >= 1; i--)
71         {
72             links.GetOrCreate(sequence[i -
73                 1], sequence[i]);
74         }
75         return indexed;
76     }
77
78     public bool CheckIndex(IList<TLink>
79         sequence)
80     {
81         var indexed = true;
82         var i = sequence.Count;
83         while (--i >= 1 && (indexed =
84             !_equalityComparer.Equals(_link
85             s.SearchOrDefault(sequence[i -
86             1], sequence[i]), _null))) { }
87         return indexed;
88     }
89 }

```

./Sequences/SequencesOptions.cs

```

1 using System;
2 using System.Collections.Generic;
3 using Platform.Interfaces;
4 using Platform.Data.Doublets.Sequences.Frequenc
5     ies.Cache;
6 using Platform.Data.Doublets.Sequences.Frequenc
7     ies.Counters;
8 using
9     Platform.Data.Doublets.Sequences.Converters;
10 using Platform.Data.Doublets.Sequences.Creteria
11     Matchers;
12
13 namespace Platform.Data.Doublets.Sequences
14 {
15     public class SequencesOptions<TLink> //
16         TODO: To use type parameter <TLink> the
17         ILinks<TLink> must contain GetConstants
18         function.
19     {
20         private static readonly
21             EqualityComparer<TLink>
22             equalityComparer =
23             EqualityComparer<TLink>.Default;
24
25         public TLink SequenceMarkerLink { get;
26             set; }
27         public bool UseCascadeUpdate { get;
28             set; }
29         public bool UseCascadeDelete { get;
30             set; }
31         public bool UseIndex { get; set; } //
32         TODO: Update Index on sequence
33         update/delete.
34         public bool UseSequenceMarker { get;
35             set; }
36         public bool UseCompression { get; set; }
37         public bool UseGarbageCollection { get;
38             set; }
39         public bool EnforceSingleSequenceVersio
40             nOnWriteBasedOnExisting { get; set;
41             }
42         public bool EnforceSingleSequenceVersio
43             nOnWriteBasedOnNew { get; set;
44             }
45
46         public MarkedSequenceCriteriaMatcher<TL
47             ink> MarkedSequenceMatcher { get;
48             set; }
49         public IConverter<IList<TLink>, TLink>
50             LinksToSequenceConverter { get;
51             set; }
52         public SequencesIndexer<TLink> Indexer
53             { get; set; }
54     }
55 }

```

```

29 // TODO: Реализовать компактификацию
    ↪ при чтении
30 //public bool
    ↪ EnforceSingleSequenceVersionOnRead
    ↪ { get; set; }
31 //public bool UseRequestMarker { get;
    ↪ set; }
32 //public bool StoreRequestResults {
    ↪ get; set; }

33
34 public void InitOptions(ISynchronizedLi
    ↪ nks<TLink>
    ↪ links)
    {
35
36     if (UseSequenceMarker)
37     {
38         if (_equalityComparer.Equals(Se
    ↪ quenceMarkerLink,
    ↪ links.Constants.Null))
39     {
40         SequenceMarkerLink =
    ↪ links.CreatePoint();
41     }
42     else
43     {
44         if (!links.Exists(SequenceM
    ↪ arkerLink))
45     {
46         var link =
    ↪ links.CreatePoint();
47         if (!_equalityComparer.
    ↪ Equals(link,
    ↪ SequenceMarkerLink))
48     {
49         throw new InvalidOp
    ↪ erationExceptio
    ↪ n("Cannot
    ↪ recreate
    ↪ sequence marker
    ↪ link.");
50     }
51     }
52     }
53     if (MarkedSequenceMatcher ==
    ↪ null)
54     {
55         MarkedSequenceMatcher = new
    ↪ MarkedSequenceCreteriaM
    ↪ atcher<TLink>(links,
    ↪ SequenceMarkerLink);
56     }
57 }
58 var balancedVariantConverter = new
    ↪ BalancedVariantConverter<TLink>
    ↪ (links);
59 if (UseCompression)
    {
60     if (LinksToSequenceConverter ==
    ↪ null)
61     {
62         ICounter<TLink, TLink>
    ↪ totalSequenceSymbolFreq
    ↪ uencyCounter;
63         if (UseSequenceMarker)
        {
64             totalSequenceSymbolFreq
    ↪ uencyCounter = new
    ↪ TotalMarkedSequence
    ↪ SymbolFrequencyCoun
    ↪ ter<TLink>(links,
    ↪ MarkedSequenceMatch
    ↪ er);
65         }
66         else
        {
67             totalSequenceSymbolFreq
    ↪ uencyCounter = new
    ↪ TotalSequenceSymbol
    ↪ FrequencyCounter<TL
    ↪ ink>(links);
68         }
69     }
70 }
71

```

```

72 var doubletFrequenciesCache
    = new LinkFrequenciesCa
    ↪ che<TLink>(links,
    ↪ totalSequenceSymbolFreq
    ↪ uencyCounter);
73 var compressingConverter =
    ↪ new CompressingConverte
    ↪ r<TLink>(links,
    ↪ balancedVariantConverte
    ↪ r,
    ↪ doubletFrequenciesCache
    ↪ );
74 LinksToSequenceConverter =
    ↪ compressingConverter;
75 }
76 }
77 else
78 {
79     if (LinksToSequenceConverter ==
    ↪ null)
80     {
81         LinksToSequenceConverter =
    ↪ balancedVariantConverte
    ↪ r;
82     }
83 }
84 if (UseIndex && Indexer == null)
85 {
86     Indexer = new SequencesIndexer<
    ↪ TLink>(links);
87 }
88 }
89
90 public void ValidateOptions()
    {
91     if (UseGarbageCollection &&
    ↪ !UseSequenceMarker)
92     {
93         throw new
    ↪ NotSupportedException("To
    ↪ use garbage collection
    ↪ UseSequenceMarker option
    ↪ must be on.");
94     }
95 }
96 }
97 }
98 }

```

./Sequences/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 namespace Platform.Data.Doublets.Sequences
    {
9     {
10         public class UnicodeMap
11         {
12             public static readonly ulong
    ↪ FirstCharLink = 1;
13             public static readonly ulong
    ↪ LastCharLink = FirstCharLink +
    ↪ char.MaxValue;
14             public static readonly ulong MapSize =
    ↪ 1 + char.MaxValue;
15
16             private readonly ILinks<ulong> _links;
17             private bool _initialized;
18
19             public UnicodeMap(ILinks<ulong> links)
    ↪ => _links = links;
20
21             public static UnicodeMap
    ↪ InitNew(ILinks<ulong> links)
22             {
23                 var map = new UnicodeMap(links);
24                 map.Init();
25                 return map;
26             }
27
28             public void Init()
29             {
30                 if (_initialized)
31                 {
32                     return;
33                 }
34             }
35         }
36     }
37 }

```



```

34     _initialized = true;
35     var firstLink =
36         ↪ _links.CreatePoint();
37     if (firstLink != FirstCharLink)
38     {
39         _links.Delete(firstLink);
40     }
41     else
42     {
43         for (var i = FirstCharLink + 1;
44             ↪ i <= LastCharLink; i++)
45         {
46             // From NIL to It (NIL ->
47             ↪ Character)
48             ↪ transformation meaning,
49             ↪ (or infinite amount of
50             ↪ NIL characters before
51             ↪ actual Character)
52             var createdLink =
53             ↪ _links.CreatePoint();
54             _links.Update(createdLink,
55             ↪ firstLink, createdLink);
56             if (createdLink != i)
57             {
58                 throw new
59                 ↪ InvalidOperationExc
60                 ↪ eption("Unable to
61                 ↪ initialize UTF 16
62                 ↪ table.");
63             }
64         }
65     }
66     // 0 - null link
67     // 1 - nil character (0 character)
68     // ...
69     // 65536 (0(1) + 65535 = 65536 possible
70     ↪ values)
71     [MethodImpl(MethodImplOptions.Aggressive
72     ↪ eInlining)]
73     public static ulong FromCharToLink(char
74     ↪ character) => (ulong)character + 1;
75     [MethodImpl(MethodImplOptions.Aggressive
76     ↪ eInlining)]
77     public static char FromLinkToChar(ulong
78     ↪ link) => (char)(link - 1);
79     [MethodImpl(MethodImplOptions.Aggressive
80     ↪ eInlining)]
81     public static bool IsCharLink(ulong
82     ↪ link) => link <= MapSize;
83     public static string
84     ↪ FromLinksToString(IList<ulong>
85     ↪ linksList)
86     {
87         var sb = new StringBuilder();
88         for (int i = 0; i <
89             ↪ linksList.Count; i++)
90         {
91             sb.Append(FromLinkToChar(linksL
92             ↪ ist[i]));
93         }
94         return sb.ToString();
95     }
96     public static string
97     ↪ FromSequenceLinkToString(ulong
98     ↪ link, ILinks<ulong> links)
99     {
100         var sb = new StringBuilder();
101         if (links.Exists(link))
102         {
103             StopableSequenceWalker.WalkRigh
104             ↪ t(link, links.GetSource,
105             ↪ links.GetTarget,
106             ↪ x => x <= MapSize ||
107             ↪ links.GetSource(x) == x
108             ↪ || links.GetTarget(x)
109             ↪ == x, element =>
110             ↪
111             ↪
112             ↪
113             ↪
114             ↪
115             ↪
116             ↪
117             ↪
118             ↪
119             ↪
120             ↪
121             ↪
122             ↪
123             ↪
124             ↪
125             ↪
126             ↪
127             ↪
128             ↪
129             ↪
130             ↪
131             ↪
132             ↪
133             ↪
134             ↪
135             ↪
136             ↪
137             ↪
138             ↪
139             ↪
140             ↪
141             ↪
142             ↪
143             ↪
144             ↪
145             ↪
146             ↪
147             ↪
148             ↪
149             ↪
150             ↪
151             ↪
152             ↪
153             ↪
154             ↪
155             ↪
156             ↪
157             ↪
158             ↪
159             ↪
160             ↪
161             ↪
162             ↪
163             ↪
164             ↪
165             ↪
166             ↪
167             ↪
168             ↪
169             ↪
170             ↪
171             ↪
172             ↪
173             ↪
174             ↪
175             ↪
176             ↪
177             ↪
178             ↪
179             ↪
180             ↪
181             ↪
182             ↪
183             ↪
184             ↪
185             ↪
186             ↪
187             ↪
188             ↪
189             ↪
190             ↪
191             ↪
192             ↪
193             ↪
194             ↪
195             ↪
196             ↪
197             ↪
198             ↪
199             ↪
200             ↪
201             ↪
202             ↪
203             ↪
204             ↪
205             ↪
206             ↪
207             ↪
208             ↪
209             ↪
210             ↪
211             ↪
212             ↪
213             ↪
214             ↪
215             ↪
216             ↪
217             ↪
218             ↪
219             ↪
220             ↪
221             ↪
222             ↪
223             ↪
224             ↪
225             ↪
226             ↪
227             ↪
228             ↪
229             ↪
230             ↪
231             ↪
232             ↪
233             ↪
234             ↪
235             ↪
236             ↪
237             ↪
238             ↪
239             ↪
240             ↪
241             ↪
242             ↪
243             ↪
244             ↪
245             ↪
246             ↪
247             ↪
248             ↪
249             ↪
250             ↪
251             ↪
252             ↪
253             ↪
254             ↪
255             ↪
256             ↪
257             ↪
258             ↪
259             ↪
260             ↪
261             ↪
262             ↪
263             ↪
264             ↪
265             ↪
266             ↪
267             ↪
268             ↪
269             ↪
270             ↪
271             ↪
272             ↪
273             ↪
274             ↪
275             ↪
276             ↪
277             ↪
278             ↪
279             ↪
280             ↪
281             ↪
282             ↪
283             ↪
284             ↪
285             ↪
286             ↪
287             ↪
288             ↪
289             ↪
290             ↪
291             ↪
292             ↪
293             ↪
294             ↪
295             ↪
296             ↪
297             ↪
298             ↪
299             ↪
300             ↪
301             ↪
302             ↪
303             ↪
304             ↪
305             ↪
306             ↪
307             ↪
308             ↪
309             ↪
310             ↪
311             ↪
312             ↪
313             ↪
314             ↪
315             ↪
316             ↪
317             ↪
318             ↪
319             ↪
320             ↪
321             ↪
322             ↪
323             ↪
324             ↪
325             ↪
326             ↪
327             ↪
328             ↪
329             ↪
330             ↪
331             ↪
332             ↪
333             ↪
334             ↪
335             ↪
336             ↪
337             ↪
338             ↪
339             ↪
340             ↪
341             ↪
342             ↪
343             ↪
344             ↪
345             ↪
346             ↪
347             ↪
348             ↪
349             ↪
350             ↪
351             ↪
352             ↪
353             ↪
354             ↪
355             ↪
356             ↪
357             ↪
358             ↪
359             ↪
360             ↪
361             ↪
362             ↪
363             ↪
364             ↪
365             ↪
366             ↪
367             ↪
368             ↪
369             ↪
370             ↪
371             ↪
372             ↪
373             ↪
374             ↪
375             ↪
376             ↪
377             ↪
378             ↪
379             ↪
380             ↪
381             ↪
382             ↪
383             ↪
384             ↪
385             ↪
386             ↪
387             ↪
388             ↪
389             ↪
390             ↪
391             ↪
392             ↪
393             ↪
394             ↪
395             ↪
396             ↪
397             ↪
398             ↪
399             ↪
400             ↪
401             ↪
402             ↪
403             ↪
404             ↪
405             ↪
406             ↪
407             ↪
408             ↪
409             ↪
410             ↪
411             ↪
412             ↪
413             ↪
414             ↪
415             ↪
416             ↪
417             ↪
418             ↪
419             ↪
420             ↪
421             ↪
422             ↪
423             ↪
424             ↪
425             ↪
426             ↪
427             ↪
428             ↪
429             ↪
430             ↪
431             ↪
432             ↪
433             ↪
434             ↪
435             ↪
436             ↪
437             ↪
438             ↪
439             ↪
440             ↪
441             ↪
442             ↪
443             ↪
444             ↪
445             ↪
446             ↪
447             ↪
448             ↪
449             ↪
450             ↪
451             ↪
452             ↪
453             ↪
454             ↪
455             ↪
456             ↪
457             ↪
458             ↪
459             ↪
460             ↪
461             ↪
462             ↪
463             ↪
464             ↪
465             ↪
466             ↪
467             ↪
468             ↪
469             ↪
470             ↪
471             ↪
472             ↪
473             ↪
474             ↪
475             ↪
476             ↪
477             ↪
478             ↪
479             ↪
480             ↪
481             ↪
482             ↪
483             ↪
484             ↪
485             ↪
486             ↪
487             ↪
488             ↪
489             ↪
490             ↪
491             ↪
492             ↪
493             ↪
494             ↪
495             ↪
496             ↪
497             ↪
498             ↪
499             ↪
500             ↪
501             ↪
502             ↪
503             ↪
504             ↪
505             ↪
506             ↪
507             ↪
508             ↪
509             ↪
510             ↪
511             ↪
512             ↪
513             ↪
514             ↪
515             ↪
516             ↪
517             ↪
518             ↪
519             ↪
520             ↪
521             ↪
522             ↪
523             ↪
524             ↪
525             ↪
526             ↪
527             ↪
528             ↪
529             ↪
530             ↪
531             ↪
532             ↪
533             ↪
534             ↪
535             ↪
536             ↪
537             ↪
538             ↪
539             ↪
540             ↪
541             ↪
542             ↪
543             ↪
544             ↪
545             ↪
546             ↪
547             ↪
548             ↪
549             ↪
550             ↪
551             ↪
552             ↪
553             ↪
554             ↪
555             ↪
556             ↪
557             ↪
558             ↪
559             ↪
560             ↪
561             ↪
562             ↪
563             ↪
564             ↪
565             ↪
566             ↪
567             ↪
568             ↪
569             ↪
570             ↪
571             ↪
572             ↪
573             ↪
574             ↪
575             ↪
576             ↪
577             ↪
578             ↪
579             ↪
580             ↪
581             ↪
582             ↪
583             ↪
584             ↪
585             ↪
586             ↪
587             ↪
588             ↪
589             ↪
590             ↪
591             ↪
592             ↪
593             ↪
594             ↪
595             ↪
596             ↪
597             ↪
598             ↪
599             ↪
600             ↪
601             ↪
602             ↪
603             ↪
604             ↪
605             ↪
606             ↪
607             ↪
608             ↪
609             ↪
610             ↪
611             ↪
612             ↪
613             ↪
614             ↪
615             ↪
616             ↪
617             ↪
618             ↪
619             ↪
620             ↪
621             ↪
622             ↪
623             ↪
624             ↪
625             ↪
626             ↪
627             ↪
628             ↪
629             ↪
630             ↪
631             ↪
632             ↪
633             ↪
634             ↪
635             ↪
636             ↪
637             ↪
638             ↪
639             ↪
640             ↪
641             ↪
642             ↪
643             ↪
644             ↪
645             ↪
646             ↪
647             ↪
648             ↪
649             ↪
650             ↪
651             ↪
652             ↪
653             ↪
654             ↪
655             ↪
656             ↪
657             ↪
658             ↪
659             ↪
660             ↪
661             ↪
662             ↪
663             ↪
664             ↪
665             ↪
666             ↪
667             ↪
668             ↪
669             ↪
670             ↪
671             ↪
672             ↪
673             ↪
674             ↪
675             ↪
676             ↪
677             ↪
678             ↪
679             ↪
680             ↪
681             ↪
682             ↪
683             ↪
684             ↪
685             ↪
686             ↪
687             ↪
688             ↪
689             ↪
690             ↪
691             ↪
692             ↪
693             ↪
694             ↪
695             ↪
696             ↪
697             ↪
698             ↪
699             ↪
700             ↪
701             ↪
702             ↪
703             ↪
704             ↪
705             ↪
706             ↪
707             ↪
708             ↪
709             ↪
710             ↪
711             ↪
712             ↪
713             ↪
714             ↪
715             ↪
716             ↪
717             ↪
718             ↪
719             ↪
720             ↪
721             ↪
722             ↪
723             ↪
724             ↪
725             ↪
726             ↪
727             ↪
728             ↪
729             ↪
730             ↪
731             ↪
732             ↪
733             ↪
734             ↪
735             ↪
736             ↪
737             ↪
738             ↪
739             ↪
740             ↪
741             ↪
742             ↪
743             ↪
744             ↪
745             ↪
746             ↪
747             ↪
748             ↪
749             ↪
750             ↪
751             ↪
752             ↪
753             ↪
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             ↪
782             ↪
783             ↪
784             ↪
785             ↪
786             ↪
787             ↪
788             ↪
789             ↪
790             ↪
791             ↪
792             ↪
793             ↪
794             ↪
795             ↪
796             ↪
797             ↪
798             ↪
799             ↪
800             ↪
801             ↪
802             ↪
803             ↪
804             ↪
805             ↪
806             ↪
807             ↪
808             ↪
809             ↪
810             ↪
811             ↪
812             ↪
813             ↪
814             ↪
815             ↪
816             ↪
817             ↪
818             ↪
819             ↪
820             ↪
821             ↪
822             ↪
823             ↪
824             ↪
825             ↪
826             ↪
827             ↪
828             ↪
829             ↪
830             ↪
831             ↪
832             ↪
833             ↪
834             ↪
835             ↪
836             ↪
837             ↪
838             ↪
839             ↪
840             ↪
841             ↪
842             ↪
843             ↪
844             ↪
845             ↪
846             ↪
847             ↪
848             ↪
849             ↪
850             ↪
851             ↪
852             ↪
853             ↪
854             ↪
855             ↪
856             ↪
857             ↪
858             ↪
859             ↪
860             ↪
861             ↪
862             ↪
863             ↪
864             ↪
865             ↪
866             ↪
867             ↪
868             ↪
869             ↪
870             ↪
871             ↪
872             ↪
873             ↪
874             ↪
875             ↪
876             ↪
877             ↪
878             ↪
879             ↪
880             ↪
881             ↪
882             ↪
883             ↪
884             ↪
885             ↪
886             ↪
887             ↪
888             ↪
889             ↪
890             ↪
891             ↪
892             ↪
893             ↪
894             ↪
895             ↪
896             ↪
897             ↪
898             ↪
899             ↪
900             ↪
901             ↪
902             ↪
903             ↪
904             ↪
905             ↪
906             ↪
907             ↪
908             ↪
909             ↪
910             ↪
911             ↪
912             ↪
913             ↪
914             ↪
915             ↪
916             ↪
917             ↪
918             ↪
919             ↪
920             ↪
921             ↪
922             ↪
923             ↪
924             ↪
925             ↪
926             ↪
927             ↪
928             ↪
929             ↪
930             ↪
931             ↪
932             ↪
933             ↪
934             ↪
935             ↪
936             ↪
937             ↪
938             ↪
939             ↪
940             ↪
941             ↪
942             ↪
943             ↪
944             ↪
945             ↪
946             ↪
947             ↪
948             ↪
949             ↪
950             ↪
951             ↪
952             ↪
953             ↪
954             ↪
955             ↪
956             ↪
957             ↪
958             ↪
959             ↪
960             ↪
961             ↪
962             ↪
963             ↪
964             ↪
965             ↪
966             ↪
967             ↪
968             ↪
969             ↪
970             ↪
971             ↪
972             ↪
973             ↪
974             ↪
975             ↪
976             ↪
977             ↪
978             ↪
979             ↪
980             ↪
981             ↪
982             ↪
983             ↪
984             ↪
985             ↪
986             ↪
987             ↪
988             ↪
989             ↪
990             ↪
991             ↪
992             ↪
993             ↪
994             ↪
995             ↪
996             ↪
997             ↪
998             ↪
999             ↪
1000            ↪

```

```

145 public static List<ulong[]> FromLinkArr
146     ayToLinkArrayGroups(ulong[]
147     array)
148 {
149     var result = new List<ulong[]>();
150     var offset = 0;
151     while (offset < array.Length)
152     {
153         var relativeLength = 1;
154         if (array[offset] <=
155             LastCharLink)
156         {
157             var currentCategory =
158                 CharUnicodeInfo.GetUnic
159                 odeCategory(FromLinkToC
160                 har(array[offset]));
161             var absoluteLength = offset
162                 + relativeLength;
163             while (absoluteLength <
164                 array.Length &&
165                 array[absoluteLength]
166                     <=
167                     LastCharLink &&
168                     currentCategory ==
169                     CharUnicodeInfo.
170                     GetUnicodeCatego
171                     ry(FromLinkToCha
172                     r(array[absolute
173                     Length])))
174             {
175                 relativeLength++;
176                 absoluteLength++;
177             }
178         }
179         else
180         {
181             var absoluteLength = offset
182                 + relativeLength;
183             while (absoluteLength <
184                 array.Length &&
185                 array[absoluteLength] >
186                 LastCharLink)
187             {
188                 relativeLength++;
189                 absoluteLength++;
190             }
191         }
192         // copy array
193         var innerSequence = new
194             ulong[relativeLength];
195         var maxLength = offset +
196             relativeLength;
197         for (var i = offset; i <
198             maxLength; i++)
199         {
200             innerSequence[i - offset] =
201                 array[i];
202         }
203         result.Add(innerSequence);
204         offset += relativeLength;
205     }
206     return result;
207 }

```

./Sequences/Walkers/LeftSequenceWalker.cs

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3
4 namespace
5     Platform.Data.Doublets.Sequences.Walkers
6     {
7         public class LeftSequenceWalker<TLink> :
8             SequenceWalkerBase<TLink>
9         {
10             public LeftSequenceWalker(ILinks<TLink>
11                 links) : base(links) { }
12
13             [MethodImpl(MethodImplOptions.Aggressive
14                 Inlining)]
15             protected override IList<TLink>
16                 GetNextElementAfterPop(IList<TLink>
17                 element) => Links.GetLink(Links.Get
18                 Source(element));
19         }
20     }

```

```

12 [MethodImpl(MethodImplOptions.Aggressive
13     Inlining)]
14     protected override IList<TLink> GetNext
15         ElementAfterPush(IList<TLink>
16         element) => Links.GetLink(Links.Get
17         Target(element));
18
19     [MethodImpl(MethodImplOptions.Aggressive
20         Inlining)]
21     protected override
22         IEnumerable<IList<TLink>>
23         WalkContents(IList<TLink> element)
24     {
25         var start =
26             Links.Constants.IndexPart + 1;
27         for (var i = element.Count - 1; i
28             >= start; i--)
29         {
30             var partLink =
31                 Links.GetLink(element[i]);
32             if (IsElement(partLink))
33             {
34                 yield return partLink;
35             }
36         }
37     }
38 }

```

./Sequences/Walkers/RightSequenceWalker.cs

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3
4 namespace
5     Platform.Data.Doublets.Sequences.Walkers
6     {
7         public class RightSequenceWalker<TLink> :
8             SequenceWalkerBase<TLink>
9         {
10             public
11                 RightSequenceWalker(ILinks<TLink>
12                 links) : base(links) { }
13
14             [MethodImpl(MethodImplOptions.Aggressive
15                 Inlining)]
16             protected override IList<TLink>
17                 GetNextElementAfterPop(IList<TLink>
18                 element) => Links.GetLink(Links.Get
19                 Target(element));
20
21             [MethodImpl(MethodImplOptions.Aggressive
22                 Inlining)]
23             protected override IList<TLink> GetNext
24                 ElementAfterPush(IList<TLink>
25                 element) => Links.GetLink(Links.Get
26                 Source(element));
27
28             [MethodImpl(MethodImplOptions.Aggressive
29                 Inlining)]
30             protected override
31                 IEnumerable<IList<TLink>>
32                 WalkContents(IList<TLink> element)
33             {
34                 for (var i =
35                     Links.Constants.IndexPart + 1;
36                     i < element.Count; i++)
37                 {
38                     var partLink =
39                         Links.GetLink(element[i]);
40                     if (IsElement(partLink))
41                     {
42                         yield return partLink;
43                     }
44                 }
45             }
46         }
47     }

```

./Sequences/Walkers/SequenceWalkerBase.cs

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3 using Platform.Data.Sequences;
4

```

```

5 namespace
  ↳ Platform.Data.Doublets.Sequences.Walkers
6 {
7     public abstract class
8         ↳ SequenceWalkerBase<TLink> :
9         ↳ LinksOperatorBase<TLink>,
10        ↳ ISequenceWalker<TLink>
11    {
12        // TODO: Use IStack in stead of
13        ↳ System.Collections.Generic.Stack,
14        ↳ but IStack should contain IsEmpty
15        ↳ property
16        private readonly Stack<IList<TLink>>
17            ↳ _stack;
18
19        protected
20            ↳ SequenceWalkerBase(ILinks<TLink>
21            ↳ links) : base(links) => _stack =
22            ↳ new Stack<IList<TLink>>();
23
24        public IEnumerable<IList<TLink>>
25            ↳ Walk(TLink sequence)
26        {
27            if (_stack.Count > 0)
28            {
29                _stack.Clear(); // This can be
30                ↳ replaced with
31                ↳ while(!_stack.IsEmpty)
32                ↳ _stack.Pop()
33            }
34            var element =
35                ↳ Links.GetLink(sequence);
36            if (IsElement(element))
37            {
38                yield return element;
39            }
40            else
41            {
42                while (true)
43                {
44                    if (IsElement(element))
45                    {
46                        if (_stack.Count == 0)
47                        {
48                            break;
49                        }
50                        element = _stack.Pop();
51                        foreach (var output in
52                            ↳ WalkContents(element)
53                            ↳ t))
54                        {
55                            yield return output;
56                        }
57                        element = GetNextElementAfterPop(element);
58                    }
59                    else
60                    {
61                        _stack.Push(element);
62                        element = GetNextElementAfterPush(element);
63                    }
64                }
65            }
66        }
67    }
68
69    [MethodImpl(MethodImplOptions.AggressiveInlining)]
70    protected virtual bool
71        ↳ IsElement(IList<TLink> elementLink)
72        ↳ => Point<TLink>.IsPartialPointUnchecked(elementLink);
73
74    [MethodImpl(MethodImplOptions.AggressiveInlining)]
75    protected abstract IList<TLink>
76        ↳ GetNextElementAfterPop(IList<TLink>
77        ↳ element);
78
79    [MethodImpl(MethodImplOptions.AggressiveInlining)]
80    protected abstract IList<TLink> GetNext
81        ↳ ElementAfterPush(IList<TLink>
82        ↳ element);

```

```

60 [MethodImpl(MethodImplOptions.AggressiveInlining)]
61     ↳ eInlining)]
62     protected abstract
63         ↳ IEnumerable<IList<TLink>>
64         ↳ WalkContents(IList<TLink> element);
65     }
66 }
67
68 ./Stacks/Stack.cs
69 1 using System.Collections.Generic;
70 2 using Platform.Collections.Stacks;
71 3
72 4 namespace Platform.Data.Doublets.Stacks
73 5 {
74     6 public class Stack<TLink> : IStack<TLink>
75     7     {
76         8         private static readonly
77             ↳ EqualityComparer<TLink>
78             ↳ _equalityComparer =
79             ↳ EqualityComparer<TLink>.Default;
80
81         9         private readonly ILinks<TLink> _links;
82         10         private readonly TLink _stack;
83
84         11         public Stack(ILinks<TLink> links, TLink
85         12             ↳ stack)
86         13         {
87             14             _links = links;
88             15             _stack = stack;
89         16         }
90
91         17         private TLink GetStackMarker() =>
92             ↳ _links.GetSource(_stack);
93
94         18         private TLink GetTop() =>
95             ↳ _links.GetTarget(_stack);
96
97         19         public TLink Peek() =>
98             ↳ _links.GetTarget(GetTop());
99
100        20         public TLink Pop()
101        21         {
102            22             var element = Peek();
103            23             if (!_equalityComparer.Equals(element,
104            24                 ↳ nt,
105            25                 ↳ _stack))
106            26             {
107                27                 var top = GetTop();
108                28                 var previousTop =
109                    ↳ _links.GetSource(top);
110                29                 _links.Update(_stack,
111                30                     ↳ GetStackMarker(),
112                31                     ↳ previousTop);
113                32                 _links.Delete(top);
114            33             }
115            34             return element;
116        35         }
117
118        36         public void Push(TLink element) =>
119            ↳ _links.Update(_stack,
120            ↳ GetStackMarker(),
121            ↳ _links.GetOrCreate(GetTop(),
122            ↳ element));
123        37     }
124    }
125
126 ./Stacks/StackExtensions.cs
127 1 namespace Platform.Data.Doublets.Stacks
128 2 {
129     3     public static class StackExtensions
130     4     {
131         5         public static TLink
132             ↳ CreateStack<TLink>(this
133             ↳ ILinks<TLink> links, TLink
134             ↳ stackMarker)
135         6         {
136             7             var stackPoint =
137                 ↳ links.CreatePoint();
138             8             var stack =
139                 ↳ links.Update(stackPoint,
140                 ↳ stackMarker, stackPoint);
141             9             return stack;
142         10         }
143
144         11         public static void
145             ↳ DeleteStack<TLink>(this
146             ↳ ILinks<TLink> links, TLink stack)
147             ↳ => links.Delete(stack);
148     12     }

```

```

13     }
14 }

```

./SynchronizedLinks.cs

```

1  using System;
2  using System.Collections.Generic;
3  using Platform.Data.Constants;
4  using Platform.Data.Doublets;
5  using Platform.Threading.Synchronization;
6
7  namespace Platform.Data.Doublets
8  {
9      /// <remarks>
10     /// TODO: Autogeneration of synchronized
11     ///       wrapper (decorator).
12     /// TODO: Try to unfold code of each method
13     ///       using IL generation for performance
14     ///       improvements.
15     /// TODO: Or even to unfold multiple layers
16     ///       of implementations.
17     /// </remarks>
18     public class SynchronizedLinks<T> :
19         ISynchronizedLinks<T>
20     {
21         public LinksCombinedConstants<T, T,
22             int> Constants { get; }
23         public ISynchronization SyncRoot { get; }
24     }
25     public ILinks<T> Sync { get; }
26     public ILinks<T> Unsync { get; }
27
28     public SynchronizedLinks(ILinks<T>
29         links) : this(new
30         ReaderWriterLockSynchronization(),
31         links) { }
32
33     public
34         SynchronizedLinks(ISynchronization
35         synchronization, ILinks<T> links)
36     {
37         SyncRoot = synchronization;
38         Sync = this;
39         Unsync = links;
40         Constants = links.Constants;
41     }
42
43     public T Count(IList<T> restriction) =>
44         SyncRoot.ExecuteReadOperation(restr
45         iction,
46         Unsync.Count);
47     public T Each(Func<IList<T>, T>
48         handler, IList<T> restrictions) =>
49         SyncRoot.ExecuteReadOperation(handl
50         er, restrictions, (handler1,
51         restrictions1) =>
52         Unsync.Each(handler1,
53         restrictions1));
54     public T Create() => SyncRoot.ExecuteWr
55         iteOperation(Unsync.Create);
56     public T Update(IList<T> restrictions)
57         => SyncRoot.ExecuteWriteOperation(r
58         estrictions,
59         Unsync.Update);
60     public void Delete(T link) => SyncRoot.
61         ExecuteWriteOperation(link,
62         Unsync.Delete);
63
64     //public T Trigger(IList<T>
65     //    restriction, Func<IList<T>,
66     //    IList<T>, T> matchedHandler,
67     //    IList<T> substitution,
68     //    Func<IList<T>, IList<T>, T>
69     //    substitutedHandler)
70     //{
71     //    if (restriction != null &&
72     //        substitution != null &&
73     //        !substitution.EqualTo(restriction))
74     //        return SyncRoot.ExecuteWriteO
75     //        peration(restriction,
76     //        matchedHandler, substitution,
77     //        substitutedHandler, Unsync.Trigger);
78     //    return SyncRoot.ExecuteReadOperat
79     //    ion(restriction, matchedHandler,
80     //    substitution, substitutedHandler,
81     //    Unsync.Trigger);
82     //}

```

```

44     }
45 }

```

./UInt64Link.cs

```

1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  using Platform.Exceptions;
5  using Platform.Ranges;
6  using Platform.Helpers.Singletons;
7  using Platform.Data.Constants;
8
9  namespace Platform.Data.Doublets
10 {
11     /// <summary>
12     /// Структура описывающая уникальную связь.
13     /// </summary>
14     public struct UInt64Link :
15         IEquatable<UInt64Link>,
16         IReadOnlyList<ulong>, IList<ulong>
17     {
18         private static readonly
19             LinksCombinedConstants<bool, ulong,
20             int> _constants = Default<LinksComb
21             inedConstants<bool, ulong,
22             int>>.Instance;
23
24         private const int Length = 3;
25
26         public readonly ulong Index;
27         public readonly ulong Source;
28         public readonly ulong Target;
29
30         public static readonly UInt64Link Null
31             = new UInt64Link();
32
33         public UInt64Link(params ulong[] values)
34         {
35             Index = values.Length >
36                 _constants.IndexPart ?
37                 values[_constants.IndexPart] :
38                 _constants.Null;
39             Source = values.Length >
40                 _constants.SourcePart ?
41                 values[_constants.SourcePart] :
42                 _constants.Null;
43             Target = values.Length >
44                 _constants.TargetPart ?
45                 values[_constants.TargetPart] :
46                 _constants.Null;
47         }
48
49         public UInt64Link(IList<ulong> values)
50         {
51             Index = values.Count >
52                 _constants.IndexPart ?
53                 values[_constants.IndexPart] :
54                 _constants.Null;
55             Source = values.Count >
56                 _constants.SourcePart ?
57                 values[_constants.SourcePart] :
58                 _constants.Null;
59             Target = values.Count >
60                 _constants.TargetPart ?
61                 values[_constants.TargetPart] :
62                 _constants.Null;
63         }
64
65         public UInt64Link(ulong index, ulong
66             source, ulong target)
67         {
68             Index = index;
69             Source = source;
70             Target = target;
71         }
72
73         public UInt64Link(ulong source, ulong
74             target)
75             : this(_constants.Null, source,
76                 target)
77         {
78             Source = source;
79             Target = target;
80         }
81
82         public static UInt64Link Create(ulong
83             source, ulong target) => new
84             UInt64Link(source, target);
85     }

```

```

56 public override int GetHashCode() => 72
    ↪ (Index, Source,
    ↪ Target).GetHashCode();
57
58 public bool IsNull() => Index == 73
    ↪ _constants.Null 74
59
60         && Source ==
        ↪ _constants.Null 75
        && Target == _cons 76
        ↪ tants.Null; 77
61
62 public override bool Equals(object 78
    ↪ other) => other is UInt64Link && 79
    ↪ Equals((UInt64Link)other); 80
63
64 public bool Equals(UInt64Link other) => 81
    ↪ Index == other.Index 82
65
    &&
    ↪
    ↪ S 84
    ↪ o 85
    ↪ u 86
    ↪ r 87
    ↪ c 88
    ↪ e 89
    ↪
    ↪ 91
    ↪
    ↪ = 92
    ↪
    ↪
    ↪ o 93
    ↪ t 94
    ↪ h 95
    ↪ e 96
    ↪ r 97
    ↪
    ↪ 98
    ↪ · 99
    ↪ S 100
    ↪ o 101
    ↪ u 102
    ↪ r 103
    ↪ c 104
    ↪ e
66 &&
    ↪
    ↪ T 105
    ↪ a 106
    ↪ r 107
    ↪ g 108
    ↪ e 109
    ↪ t 110
    ↪
    ↪ 111
    ↪ 112
    ↪ = 113
    ↪
    ↪ 114
    ↪
    ↪ o 115
    ↪ t 116
    ↪ h 117
    ↪ e 118
    ↪ r 120
    ↪
    ↪ 121
    ↪ · 122
    ↪ T 123
    ↪ a 124
    ↪ r 125
    ↪ g 126
    ↪ e 127
    ↪ t
    ↪
    ↪ 128
67
68 public static string ToString(ulong
    ↪ index, ulong source, ulong target) 129
    ↪ => $"({index}: {source}->{target})"; 130
69
70 public static string ToString(ulong
    ↪ source, ulong target) => 131
    ↪ $"({source}->{target})";
71
    public static implicit operator
    ↪ ulong[] (UInt64Link link) =>
    ↪ link.ToArray();

    public static implicit operator
    ↪ UInt64Link(ulong[] linkArray) =>
    ↪ new UInt64Link(linkArray);

    public ulong[] ToArray()
    {
        var array = new ulong[Length];
        CopyTo(array, 0);
        return array;
    }

    public override string ToString() =>
    ↪ Index == _constants.Null ?
    ↪ ToString(Source, Target) :
    ↪ ToString(Index, Source, Target);

    #region IList

    public ulong this[int index]
    {
        get
        {
            Ensure.Always.ArgumentInRange(i
            ↪ ndex, new Range<int>(0,
            ↪ Length - 1), nameof(index));
            if (index ==
            ↪ _constants.IndexPart)
            {
                return Index;
            }
            if (index ==
            ↪ _constants.SourcePart)
            {
                return Source;
            }
            if (index ==
            ↪ _constants.TargetPart)
            {
                return Target;
            }
            throw new
            ↪ NotSupportedException(); //
            ↪ Impossible path due to
            ↪ Ensure.ArgumentInRange
        }
        set => throw new
            ↪ NotSupportedException();
    }

    public int Count => Length;

    public bool IsReadOnly => true;

    IEnumerator IEnumerable.GetEnumerator()
    ↪ => GetEnumerator();

    public IEnumerator<ulong>
    ↪ GetEnumerator()
    {
        yield return Index;
        yield return Source;
        yield return Target;
    }

    public void Add(ulong item) => throw
    ↪ new NotSupportedException();

    public void Clear() => throw new
    ↪ NotSupportedException();

    public bool Contains(ulong item) =>
    ↪ IndexOf(item) >= 0;

    public void CopyTo(ulong[] array, int
    ↪ arrayIndex)
    {
        Ensure.Always.ArgumentNotNull(array
        ↪
        ↪ nameof(array));
        Ensure.Always.ArgumentInRange(array
        ↪
        ↪ Index, new Range<int>(0,
        ↪ array.Length - 1),
        ↪ nameof(arrayIndex));
    }

```

```

132         if (arrayIndex + Length >
133             ↳ array.Length)
134         {
135             throw new ArgumentException();
136         }
137         array[arrayIndex++] = Index;
138         array[arrayIndex++] = Source;
139         array[arrayIndex] = Target;
140     }
141     public bool Remove(ulong item) =>
142     ↳ Throw.A.NotSupportedExceptionAndRet
143     ↳ urn<bool>();
144     public int IndexOf(ulong item)
145     {
146         if (Index == item)
147         {
148             return _constants.IndexPart;
149         }
150         if (Source == item)
151         {
152             return _constants.SourcePart;
153         }
154         if (Target == item)
155         {
156             return _constants.TargetPart;
157         }
158         return -1;
159     }
160     public void Insert(int index, ulong
161     ↳ item) => throw new
162     ↳ NotSupportedException();
163     public void RemoveAt(int index) =>
164     ↳ throw new NotSupportedException();
165     #endregion
166 }
167 }

```

./UInt64LinkExtensions.cs

```

1 namespace Platform.Data.Doublets
2 {
3     public static class UInt64LinkExtensions
4     {
5         public static bool IsFullPoint(this
6         ↳ UInt64Link link) =>
7         ↳ Point<ulong>.IsFullPoint(link);
8         public static bool IsPartialPoint(this
9         ↳ UInt64Link link) =>
10        ↳ Point<ulong>.IsPartialPoint(link);
11    }
12 }

```

./UInt64LinksExtensions.cs

```

1 using System;
2 using System.Text;
3 using System.Collections.Generic;
4 using Platform.Helpers.Singletons;
5 using Platform.Data.Constants;
6 using Platform.Data.Exceptions;
7 using Platform.Data.Doublets.Sequences;
8 namespace Platform.Data.Doublets
9 {
10     public static class UInt64LinksExtensions
11     {
12         public static readonly
13         ↳ LinksCombinedConstants<bool, ulong,
14         ↳ int> Constants = Default<LinksCombi
15         ↳ nedConstants<bool, ulong,
16         ↳ int>>.Instance;
17         public static void UseUnicode(this
18         ↳ ILinks<ulong> links) =>
19         ↳ UnicodeMap.InitNew(links);
20         public static void
21         ↳ EnsureEachLinkExists(this
22         ↳ ILinks<ulong> links, IList<ulong>
23         ↳ sequence)
24         {
25             if (sequence == null)
26             {
27                 return;
28             }
29         }
30     }
31 }

```

```

32 }
33 for (var i = 0; i < sequence.Count;
34     ↳ i++)
35 {
36     if (!links.Exists(sequence[i]))
37     {
38         throw new ArgumentLinkDoesN
39         ↳ otExistsException<ulong>
40         ↳ >(sequence[i],
41         ↳ $"sequence[{i}]");
42     }
43 }
44 public static void
45 ↳ EnsureEachLinkIsAnyOrExists(this
46 ↳ ILinks<ulong> links, IList<ulong>
47 ↳ sequence)
48 {
49     if (sequence == null)
50     {
51         return;
52     }
53     for (var i = 0; i < sequence.Count;
54         ↳ i++)
55     {
56         if (sequence[i] !=
57         ↳ Constants.Any &&
58         ↳ !links.Exists(sequence[i]))
59         {
60             throw new ArgumentLinkDoesN
61             ↳ otExistsException<ulong>
62             ↳ >(sequence[i],
63             ↳ $"sequence[{i}]");
64         }
65     }
66 }
67 public static bool AnyLinkIsAny(this
68 ↳ ILinks<ulong> links, params ulong[]
69 ↳ sequence)
70 {
71     if (sequence == null)
72     {
73         return false;
74     }
75     var constants = links.Constants;
76     for (var i = 0; i <
77         ↳ sequence.Length; i++)
78     {
79         if (sequence[i] ==
80         ↳ constants.Any)
81         {
82             return true;
83         }
84     }
85     return false;
86 }
87 public static string
88 ↳ FormatStructure(this ILinks<ulong>
89 ↳ links, ulong linkIndex,
90 ↳ Func<UInt64Link, bool> isElement,
91 ↳ bool renderIndex = false, bool
92 ↳ renderDebug = false)
93 {
94     var sb = new StringBuilder();
95     var visited = new HashSet<ulong>();
96     links.AppendStructure(sb, visited,
97         ↳ linkIndex, isElement, (innerSb,
98         ↳ link) =>
99         ↳ innerSb.Append(link.Index),
100         ↳ renderIndex, renderDebug);
101     return sb.ToString();
102 }
103 public static string
104 ↳ FormatStructure(this ILinks<ulong>
105 ↳ links, ulong linkIndex,
106 ↳ Func<UInt64Link, bool> isElement,
107 ↳ Action<StringBuilder, UInt64Link>
108 ↳ appendElement, bool renderIndex =
109 ↳ false, bool renderDebug = false)
110 {
111     var sb = new StringBuilder();
112 }

```

```

75     var visited = new HashSet<ulong>();
76     links.AppendStructure(sb, visited,
        ↪ linkIndex, isElement,
        ↪ appendElement, renderIndex,
        ↪ renderDebug);
77     return sb.ToString();
78 }
79
80 public static void AppendStructure(this
    ↪ ILinks<ulong> links, StringBuilder
    ↪ sb, HashSet<ulong> visited, ulong
    ↪ linkIndex, Func<UInt64Link, bool>
    ↪ isElement, Action<StringBuilder,
    ↪ UInt64Link> appendElement, bool
    ↪ renderIndex = false, bool
    ↪ renderDebug = false)
81 {
82     if (sb == null)
83     {
84         throw new ArgumentNullException
            ↪ (nameof(sb));
85     }
86     if (linkIndex == Constants.Null ||
        ↪ linkIndex == Constants.Any ||
        ↪ linkIndex == Constants.Itself)
87     {
88         return;
89     }
90     if (links.Exists(linkIndex))
91     {
92         if (visited.Add(linkIndex))
93         {
94             sb.Append('(');
95             var link = new
                ↪ UInt64Link(links.GetLin
                ↪ k(linkIndex));
96             if (renderIndex)
97             {
98                 sb.Append(link.Index);
99                 sb.Append(':');
100             }
101             if (link.Source ==
                ↪ link.Index)
102             {
103                 sb.Append(link.Index);
104             }
105             else
106             {
107                 var source = new
                    ↪ UInt64Link(links.Ge
                    ↪ tLink(link.Source));
108                 if (isElement(source))
109                 {
110                     appendElement(sb,
                        ↪ source);
111                 }
112                 else
113                 {
114                     links.AppendStructu
                        ↪ re(sb, visited,
                        ↪ source.Index,
                        ↪ isElement,
                        ↪ appendElement,
                        ↪ renderIndex);
115                 }
116             }
117             sb.Append(' ');
118             if (link.Target ==
                ↪ link.Index)
119             {
120                 sb.Append(link.Index);
121             }
122             else
123             {
124                 var target = new
                    ↪ UInt64Link(links.Ge
                    ↪ tLink(link.Target));
125                 if (isElement(target))
126                 {
127                     appendElement(sb,
                        ↪ target);
128                 }
129                 else
130                 {

```

```

131             links.AppendStructu
                ↪ re(sb, visited,
                ↪ target.Index,
                ↪ isElement,
                ↪ appendElement,
                ↪ renderIndex);
132         }
133     }
134     sb.Append(')');
135 }
136 else
137 {
138     if (renderDebug)
139     {
140         sb.Append('*');
141     }
142     sb.Append(linkIndex);
143 }
144 }
145 else
146 {
147     if (renderDebug)
148     {
149         sb.Append('~');
150     }
151     sb.Append(linkIndex);
152 }
153 }
154 }
155 }

```

./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
14 namespace Platform.Data.Doublets
15 {
16     public class UInt64LinksTransactionsLayer :
        ↪ LinksDisposableDecoratorBase<ulong>
        ↪ //-V3073
17     {
18         /// <remarks>
19         /// Альтернативные варианты хранения
20         /// трансформации (элемента транзакции):
21         ///
22         /// private enum TransitionType
23         /// {
24         ///     Creation,
25         ///     UpdateOf,
26         ///     UpdateTo,
27         ///     Deletion
28         /// }
29         /// private struct Transition
30         /// {
31         ///     public ulong TransactionId;
32         ///     public UniqueTimestamp
            ↪ Timestamp;
33         ///     public TransactionItemType Type;
34         ///     public Link Source;
35         ///     public Link Linker;
36         ///     public Link Target;
37         /// }
38         /// Или
39         ///
40         /// public struct TransitionHeader
41         /// {
42         ///     public ulong
            ↪ TransactionIdCombined;
43         ///     public ulong TimestampCombined;
44         ///
45         ///     public ulong TransactionId
46         ///     {
47         ///         get
48         ///         {
49             return (ulong) mask &
            ↪ TransactionIdCombined;
50         }

```



```

51     ///     }
52     /// }
53     ///
54     /// public UniqueTimestamp Timestamp
55     /// {
56     ///     get
57     ///     {
58     ///         return
59     ///         ↪ (UniqueTimestamp)mask &
60     ///         ↪ TransactionIdCombined;
61     ///     }
62     ///
63     ///     public TransactionItemType Type
64     ///     {
65     ///         get
66     ///         {
67     ///             // Использовать по
68     ///             ↪ одному биту из TransactionId и
69     ///             ↪ Timestamp,
70     ///             // для значения в 2
71     ///             ↪ бита, которое представляет тип
72     ///             ↪ операции
73     ///             throw new
74     ///             ↪ NotImplementedException();
75     ///         }
76     ///     }
77     /// }
78     /// private struct Transition
79     /// {
80     ///     public TransitionHeader Header;
81     ///     public Link Source;
82     ///     public Link Linker;
83     ///     public Link Target;
84     /// }
85     /// </remarks>
86     public struct Transition
87     {
88         public static readonly long Size =
89         ↪ StructureHelpers.SizeOf<Transit
90         ↪ ion>();
91
92         public readonly ulong TransactionId;
93         public readonly UInt64Link Before;
94         public readonly UInt64Link After;
95         public readonly Timestamp Timestamp;
96
97         public Transition(UniqueTimestampFa
98         ↪ ctory uniqueTimestampFactory,
99         ↪ ulong transactionId, UInt64Link
100        ↪ before, UInt64Link after)
101        {
102            TransactionId = transactionId;
103            Before = before;
104            After = after;
105            Timestamp = uniqueTimestampFact
106            ↪ ory.Create();
107        }
108
109        public Transition(UniqueTimestampFa
110        ↪ ctory uniqueTimestampFactory,
111        ↪ ulong transactionId, UInt64Link
112        ↪ before)
113        : this(uniqueTimestampFactory,
114        ↪ transactionId, before,
115        ↪ default)
116        {
117        }
118
119        public Transition(UniqueTimestampFa
120        ↪ ctory uniqueTimestampFactory,
121        ↪ ulong transactionId)
122        : this(uniqueTimestampFactory,
123        ↪ transactionId, default,
124        ↪ default)
125        {
126        }
127
128        public override string ToString()
129        ↪ => $"[{Timestamp}]
130        ↪ [{TransactionId}: {Before} =>
131        ↪ {After}]";
132    }
133
134    /// <remarks>
135    ///
136    /// Другие варианты реализации
137    /// ↪ транзакций (атомарности):
138    ///
139    /// 1. Разделение хранения значения
140    /// ↪ связи ((Source Target) или (Source
141    /// ↪ Linker Target)) и индексов.
142    ///
143    /// 2. Хранение
144    /// ↪ трансформаций/операций в отдельном
145    /// ↪ хранилище Links, но дополнительно
146    /// ↪ потребуется решить вопрос
147    /// ↪ со ссылками на внешние
148    /// ↪ идентификаторы, или как-то иначе
149    /// ↪ решить вопрос с пересечениями
150    /// ↪ идентификаторов.
151    ///
152    /// Где хранить промежуточный список
153    /// ↪ транзакций?
154    ///
155    /// В оперативной памяти:
156    /// Минусы:
157    /// 1. Может усложнить систему,
158    /// ↪ если она будет функционировать
159    /// ↪ самостоятельно,
160    /// ↪ так как нужно отдельно выделять
161    /// ↪ память под список трансформаций.
162    /// 2. Выделенной оперативной
163    /// ↪ памяти может не хватить, в том
164    /// ↪ случае,
165    /// ↪ если транзакция использует
166    /// ↪ слишком много трансформаций.
167    /// -> Можно использовать
168    /// ↪ жёсткий диск для слишком длинных
169    /// ↪ транзакций.
170    /// -> Максимальный размер
171    /// ↪ списка трансформаций можно
172    /// ↪ ограничить / задать константой.
173    /// 3. При подтверждении транзакции
174    /// ↪ (Commit) все трансформации
175    /// ↪ записываются разом создавая
176    /// ↪ задержку.
177    ///
178    /// На жёстком диске:
179    /// Минусы:
180    /// 1. Длительный отклик, на запись
181    /// ↪ каждой трансформации.
182    /// 2. Лог транзакций дополнительно
183    /// ↪ наполняется отменёнными
184    /// ↪ транзакциями.
185    /// -> Это может решаться
186    /// ↪ упаковкой/исключением дублирующих
187    /// ↪ операций.
188    /// -> Также это может решаться
189    /// ↪ тем, что короткие транзакции вообще
190    /// ↪ не будут записываться в
191    /// ↪ случае отката.
192    /// 3. Перед тем как выполнять
193    /// ↪ отмену операций транзакции нужно
194    /// ↪ дожидаться пока все операции
195    /// ↪ (трансформации)
196    /// ↪ будут записаны в лог.
197    ///
198    /// </remarks>
199    public class Transaction :
200    ↪ DisposableBase
201    {
202        private readonly Queue<Transition>
203        ↪ _transitions;
204        private readonly
205        ↪ UInt64LinksTransactionsLayer
206        ↪ _layer;
207        public bool IsCommitted { get;
208        ↪ private set; }
209        public bool IsReverted { get;
210        ↪ private set; }
211
212        public Transaction(UInt64LinksTrans
213        ↪ actionsLayer
214        ↪ layer)
215        {
216            _layer = layer;
217            if (_layer._currentTransactionI
218            ↪ d !=
219            ↪ 0)
220            {
221            }
222        }
223    }

```

```

153         throw new NotSupportedException(
154             ↪ tion("Nested
155             ↪ transactions not
156             ↪ supported.");
157     }
158     IsCommitted = false;
159     IsReverted = false;
160     _transitions = new
161     ↪ Queue<Transition>();
162     SetCurrentTransaction(layer,
163     ↪ this);
164 }
165 public void Commit()
166 {
167     EnsureTransactionAllowsWriteOperations(
168     ↪ this);
169     while (_transitions.Count > 0)
170     {
171         var transition =
172         ↪ _transitions.Dequeue();
173         _layer._transitions.Enqueue(
174         ↪ (transition);
175     }
176     _layer._lastCommittedTransactionId =
177     ↪ _layer._currentTransactionId;
178     IsCommitted = true;
179 }
180 private void Revert()
181 {
182     EnsureTransactionAllowsWriteOperations(
183     ↪ this);
184     var transitionsToRevert = new
185     ↪ Transition[_transitions.Count];
186     _transitions.CopyTo(transitionsToRevert,
187     ↪ 0);
188     for (var i =
189     ↪ transitionsToRevert.Length
190     ↪ - 1; i >= 0; i--)
191     {
192         _layer.RevertTransition(transitionsToRevert[i]);
193     }
194     IsReverted = true;
195 }
196 public static void
197     SetCurrentTransaction(UInt64LinksTransactionsLayer layer,
198     ↪ Transaction transaction)
199 {
200     layer._currentTransactionId =
201     ↪ layer._lastCommittedTransactionId +
202     ↪ 1;
203     layer._currentTransactionTransitions =
204     ↪ transaction._transitions;
205     layer._currentTransaction =
206     ↪ transaction;
207 }
208 public static void
209     EnsureTransactionAllowsWriteOperations(Transaction transaction)
210 {
211     if (transaction.IsReverted)
212     {
213         throw new InvalidOperationException(
214         ↪ exception("Transaction is
215         ↪ reverted.");
216     }
217     if (transaction.IsCommitted)
218     {
219         throw new InvalidOperationException(
220         ↪ exception("Transaction is
221         ↪ committed.");
222     }
223 }
224
225 protected override void
226     DisposeCore(bool manual, bool
227     ↪ wasDisposed)
228 {
229     if (!wasDisposed && !_layer !=
230     ↪ null && !_layer.IsDisposed)
231     {
232         if (!IsCommitted &&
233         ↪ !IsReverted)
234         {
235             Revert();
236         }
237         _layer.ResetCurrentTransaction();
238     }
239 }
240
241 // TODO: THIS IS EXCEPTION
242     WORKAROUND, REMOVE IT THEN
243     https://github.com/linksplatform/
244     ↪ m/Disposables/issues/13
245     FIXED
246 protected override bool
247     AllowMultipleDisposeCalls =>
248     ↪ true;
249 }
250
251 public static readonly TimeSpan
252     DefaultPushDelay =
253     ↪ TimeSpan.FromSeconds(0.1);
254
255 private readonly string _logAddress;
256 private readonly FileStream _log;
257 private readonly Queue<Transition>
258     ↪ _transitions;
259 private readonly UniqueTimestampFactory
260     ↪ _uniqueTimestampFactory;
261 private Task _transitionsPusher;
262 private Transition
263     ↪ _lastCommittedTransaction;
264 private ulong _currentTransactionId;
265 private Queue<Transition>
266     ↪ _currentTransactionTransitions;
267 private Transaction _currentTransaction;
268 private ulong
269     ↪ _lastCommittedTransactionId;
270
271 public UInt64LinksTransactionsLayer(ILink
272     ↪ nks<ulong> links, string
273     ↪ logAddress)
274     : base(links)
275 {
276     if (string.IsNullOrEmpty(logAddress))
277     {
278         throw new ArgumentNullException(
279         ↪ (nameof(logAddress)));
280     }
281     // В первой строке файла хранится
282     ↪ последняя закомиченная
283     ↪ транзакция.
284     // При запуске это используется для
285     ↪ проверки удачного закрытия
286     ↪ файла лога.
287     // In the first line of the file
288     ↪ the last committed transaction
289     ↪ is stored.
290     // On startup, this is used to
291     ↪ check that the log file is
292     ↪ successfully closed.
293     var lastCommittedTransition =
294     ↪ FileHelpers.ReadFirstOrDefault<
295     ↪ Transition>(logAddress);
296     var lastWrittenTransition =
297     ↪ FileHelpers.ReadLastOrDefault<T
298     ↪ ransition>(logAddress);
299     if (!lastCommittedTransition.Equals(
300     ↪ lastWrittenTransition))
301     {
302         Dispose();
303         throw new NotSupportedException(
304         ↪ ("Database is damaged,
305         ↪ autorecovery is not
306         ↪ supported yet.");
307     }
308 }

```

```

251         if (lastCommittedTransition.Equals(d_ 300             Transaction.EnsureTransactionAl
252             ↪ default(Transition)))             ↪ lowsWriteOperations(_curren
253         {                                     ↪ tTransaction);
254             FileHelpers.WriteFirst(logAddre_ 301         }
255             ↪ ss,                               302         var transitions =
256             ↪ lastCommittedTransition);         ↪ GetCurrentTransitions();
257         }                                     ↪ transitions.Enqueue(transition);
258         _lastCommittedTransition =           303     }
259         ↪ lastCommittedTransition;           304
260         // TODO: Think about a better way    305
261         ↪ to calculate or store this value    306     private void
262         var allTransitions = FileHelpers.Re_ 307         ↪ RevertTransition(Transition
263         ↪ adAll<Transition>(logAddress);        ↪ transition)
264         _lastCommittedTransactionId =       308         {
265         ↪ allTransitions.Max(x =>              309             if (transition.After.IsNull()) //
266         ↪ x.TransactionId);                   310             ↪ Revert Deletion with Creation
267         _uniqueTimestampFactory = new       311         {
268         ↪ UniqueTimestampFactory();           312             Links.Create();
269         _logAddress = logAddress;           313         }
270         _log =                               314         else if
271         ↪ FileHelpers.Append(logAddress);      315             (transition.Before.IsNull()) //
272         _transitions = new                  316             ↪ Revert Creation with Deletion
273         ↪ Queue<Transition>());               317         {
274         _transitionsPusher = new           318             Links.Delete(transition.After.I
275         ↪ Task(TransitionsPusher);            ↪ ndex);
276         _transitionsPusher.Start();         319         }
277     }                                     320         else // Revert Update
278                                     321         {
279         public IList<ulong> GetLinkValue(ulong 322             Links.Update(new[] {
280             ↪ link) => Links.GetLink(link);    ↪ transition.After.Index,
281                                     ↪ transition.Before.Source,
282                                     ↪ transition.Before.Target });
283         public override ulong Create()       323         }
284         {                                     324     }
285         var createdLinkIndex =             325     private void ResetCurrentTransation()
286         ↪ Links.Create();                     {
287         var createdLink = new UInt64Link(Li_ 326         _currentTransactionId = 0;
288         ↪ nks.GetLink(createdLinkIndex));      327         _currentTransactionTransitions =
289         CommitTransition(new Transition(_un_ 328         ↪ null;
290         ↪ iqueTimestampFactory,                329         _currentTransaction = null;
291         ↪ _currentTransactionId, default,      330     }
292         ↪ createdLink));                       331     private void PushTransitions()
293         return createdLinkIndex;            332     {
294     }                                     333         if (_log == null || _transitions ==
295         public override ulong               334         ↪ null)
296         ↪ Update(IList<ulong> parts)          335         {
297         {                                     336             return;
298             var beforeLink = new           337         }
299             ↪ UInt64Link(Links.GetLink(parts[_ 338         for (var i = 0; i <
300             ↪ Constants.IndexPart]));        ↪ _transitions.Count; i++)
301         parts[Constants.IndexPart] =       339         {
302         ↪ Links.Update(parts);               340             var transition =
303         var afterLink = new               341             ↪ _transitions.Dequeue();
304             ↪ UInt64Link(Links.GetLink(parts[_ 342             _log.Write(transition);
305             ↪ Constants.IndexPart]));        ↪ _lastCommittedTransition =
306         CommitTransition(new Transition(_un_ 343             ↪ transition;
307         ↪ iqueTimestampFactory,                344         }
308         ↪ _currentTransactionId,              345         }
309         ↪ beforeLink, afterLink));           346     }
310         return parts[Constants.IndexPart]; 347     private void TransitionsPusher()
311     }                                     348     {
312         public override void Delete(ulong link) 349         while (!IsDisposed &&
313         {                                     ↪ _transitionsPusher != null)
314             var deletedLink = new         350         {
315             ↪ UInt64Link(Links.GetLink(link)); 351             Thread.Sleep(DefaultPushDelay);
316             Links.Delete(link);           352             PushTransitions();
317         CommitTransition(new Transition(_un_ 353         }
318         ↪ iqueTimestampFactory,                354     }
319         ↪ _currentTransactionId,              355     public Transaction BeginTransaction()
320         ↪ deletedLink, default));           356         ↪ => new Transaction(this);
321     }                                     357     private void DisposeTransitions()
322     [MethodImpl(MethodImplOptions.Aggressive_ 358     {
323     ↪ eInlining)]                             359         try
324         private Queue<Transition>         360         {
325             ↪ GetCurrentTransitions() =>      361             var pusher = _transitionsPusher;
326             ↪ _currentTransactionTransitions ?? 362             if (pusher != null)
327             ↪ _transitions;                 {
328         private void                     363             _transitionsPusher = null;
329             ↪ CommitTransition(Transition      pusher.Wait();
330             ↪ transition)                   }
331         {                                     364             if (_transitions != null)
332             if (_currentTransaction != null) 365             {
333             {                               366

```

367	PushTransitions();	379	protected override void
368	}	380	↳ DisposeCore(bool manual, bool
369	Disposable.TryDispose(_log);	381	↳ wasDisposed)
370	FileHelpers.WriteFirst(_logAddr	382	{
	↳ ess,	383	if (!wasDisposed)
	↳ _lastCommittedTransition);	384	{
371	}	385	DisposeTransitions();
372	catch	386	}
373	{	387	base.DisposeCore(manual,
374	}	388	↳ wasDisposed);
375	}	389	}
376		390	#endregion
377	#region DisposalBase		
378			

Index

- ./Converters/AddressToUnaryNumberConverter.cs, 1
- ./Converters/LinkToltsFrequencyNumberConveter.cs, 1
- ./Converters/PowerOf2ToUnaryNumberConverter.cs, 1
- ./Converters/UnaryNumberToAddressAddOperationConverter.cs, 2
- ./Converters/UnaryNumberToAddressOrOperationConverter.cs, 2
- ./Decorators/LinksCascadeDependenciesResolver.cs, 3
- ./Decorators/LinksCascadeUniquenessAndDependenciesResolver.cs, 3
- ./Decorators/LinksDecoratorBase.cs, 4
- ./Decorators/LinksDependenciesValidator.cs, 4
- ./Decorators/LinksDisposableDecoratorBase.cs, 4
- ./Decorators/LinksInnerReferenceValidator.cs, 4
- ./Decorators/LinksNonExistentReferencesCreator.cs, 5
- ./Decorators/LinksNullToSelfReferenceResolver.cs, 5
- ./Decorators/LinksSelfReferenceResolver.cs, 5
- ./Decorators/LinksUniquenessResolver.cs, 6
- ./Decorators/LinksUniquenessValidator.cs, 6
- ./Decorators/NonNullContentsLinkDeletionResolver.cs, 6
- ./Decorators/UInt64Links.cs, 6
- ./Decorators/UniLinks.cs, 8
- ./Doublet.cs, 11
- ./DoubletComparer.cs, 11
- ./Hybrid.cs, 12
- ./ILinks.cs, 12
- ./ILinksExtensions.cs, 13
- ./ISynchronizedLinks.cs, 20
- ./Incrementers/FrequencyIncrementer.cs, 19
- ./Incrementers/LinkFrequencyIncrementer.cs, 20
- ./Incrementers/UnaryNumberIncrementer.cs, 20
- ./Link.cs, 21
- ./LinkExtensions.cs, 23
- ./LinksOperatorBase.cs, 23
- ./PropertyOperators/DefaultLinkPropertyOperator.cs, 23
- ./PropertyOperators/FrequencyPropertyOperator.cs, 23
- ./ResizableDirectMemory/ResizableDirectMemoryLinks.ListMethods.cs, 31
- ./ResizableDirectMemory/ResizableDirectMemoryLinks.TreeMethods.cs, 32
- ./ResizableDirectMemory/ResizableDirectMemoryLinks.cs, 24
- ./ResizableDirectMemory/UInt64ResizableDirectMemoryLinks.ListMethods.cs, 42
- ./ResizableDirectMemory/UInt64ResizableDirectMemoryLinks.TreeMethods.cs, 42
- ./ResizableDirectMemory/UInt64ResizableDirectMemoryLinks.cs, 37
- ./Sequences/Converters/BalancedVariantConverter.cs, 47
- ./Sequences/Converters/CompressingConverter.cs, 47
- ./Sequences/Converters/LinksListToSequenceConverterBase.cs, 50
- ./Sequences/Converters/OptimalVariantConverter.cs, 50
- ./Sequences/Converters/SequenceToltsLocalElementLevelsConverter.cs, 51
- ./Sequences/CreteriaMatchers/DefaultSequenceElementCreteriaMatcher.cs, 51
- ./Sequences/CreteriaMatchers/MarkedSequenceCreteriaMatcher.cs, 51
- ./Sequences/DefaultSequenceAppender.cs, 51
- ./Sequences/DuplicateSegmentsCounter.cs, 52
- ./Sequences/DuplicateSegmentsProvider.cs, 52
- ./Sequences/Frequencies/Cache/FrequenciesCacheBasedLinkFrequencyIncrementer.cs, 54
- ./Sequences/Frequencies/Cache/FrequenciesCacheBasedLinkToltsFrequencyNumberConverter.cs, 54
- ./Sequences/Frequencies/Cache/LinkFrequenciesCache.cs, 54
- ./Sequences/Frequencies/Cache/LinkFrequency.cs, 55
- ./Sequences/Frequencies/Counters/MarkedSequenceSymbolFrequencyOneOffCounter.cs, 56
- ./Sequences/Frequencies/Counters/SequenceSymbolFrequencyOneOffCounter.cs, 56
- ./Sequences/Frequencies/Counters/TotalMarkedSequenceSymbolFrequencyCounter.cs, 56
- ./Sequences/Frequencies/Counters/TotalMarkedSequenceSymbolFrequencyOneOffCounter.cs, 56
- ./Sequences/Frequencies/Counters/TotalSequenceSymbolFrequencyCounter.cs, 57
- ./Sequences/Frequencies/Counters/TotalSequenceSymbolFrequencyOneOffCounter.cs, 57
- ./Sequences/HeightProviders/CachedSequenceHeightProvider.cs, 57
- ./Sequences/HeightProviders/DefaultSequenceRightHeightProvider.cs, 58
- ./Sequences/HeightProviders/ISequenceHeightProvider.cs, 58
- ./Sequences/Sequences.Experiments.ReadSequence.cs, 83
- ./Sequences/Sequences.Experiments.cs, 64
- ./Sequences/Sequences.cs, 58
- ./Sequences/SequencesExtensions.cs, 83
- ./Sequences/SequencesIndexer.cs, 83
- ./Sequences/SequencesOptions.cs, 84
- ./Sequences/UnicodeMap.cs, 85
- ./Sequences/Walkers/LeftSequenceWalker.cs, 87
- ./Sequences/Walkers/RightSequenceWalker.cs, 87
- ./Sequences/Walkers/SequenceWalkerBase.cs, 87

./Stacks/Stack.cs, 88
./Stacks/StackExtensions.cs, 88
./SynchronizedLinks.cs, 89
./UInt64Link.cs, 89
./UInt64LinkExtensions.cs, 91
./UInt64LinksExtensions.cs, 91
./UInt64LinksTransactionsLayer.cs, 92
./obj/Debug/netstandard2.0/Platform.Data.Doublets.AssemblyInfo.cs, 23