



哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

2019 年春季学期
计算机学院《软件构造》课程

Lab 2 实验报告

姓名	安天
学号	1170300627
班号	1703006
电子邮件	2507770493@qq.com
手机号码	14794410346

目录

1 实验目标概述	1
2 实验环境配置	1
3 实验过程	3
3.1 Poetic Walks	3
3.1.1 Get the code and prepare Git repository	3
3.1.2 Problem 1: Test Graph <String>	3
3.1.3 Problem 2: Implement Graph <String>	8
3.1.3.1 Implement ConcreteEdgesGraph	9
3.1.3.2 Implement ConcreteVerticesGraph	12
3.1.4 Problem 3: Implement generic Graph<L>	16
3.1.4.1 Make the implementations generic	16
3.1.4.2 Implement Graph.empty()	17
3.1.5 Problem 4: Poetic walks	17
3.1.5.1 Test GraphPoet	17
3.1.5.2 Implement GraphPoet	18
3.1.5.3 Graph poetry slam	20
3.1.6 Before you're done	21
3.2 Re-implement the Social Network in Lab1	21
3.2.1 FriendshipGraph 类	21
3.2.2 Person 类	22
3.2.3 客户端 main()	23
3.2.4 测试用例	25
3.2.5 提交至 Git 仓库	27
3.3 Playing Chess	28
3.3.1 ADT 设计/实现方案	28
3.3.2 主程序 ChessGame 设计/实现方案	30
3.3.3 ADT 和主程序的测试方案	32
3.4 Multi-Startup Set (MIT)	34
4 实验进度记录	35
5 实验过程中遇到的困难与解决途径	35
6 实验过程中收获的经验、教训、感想	35

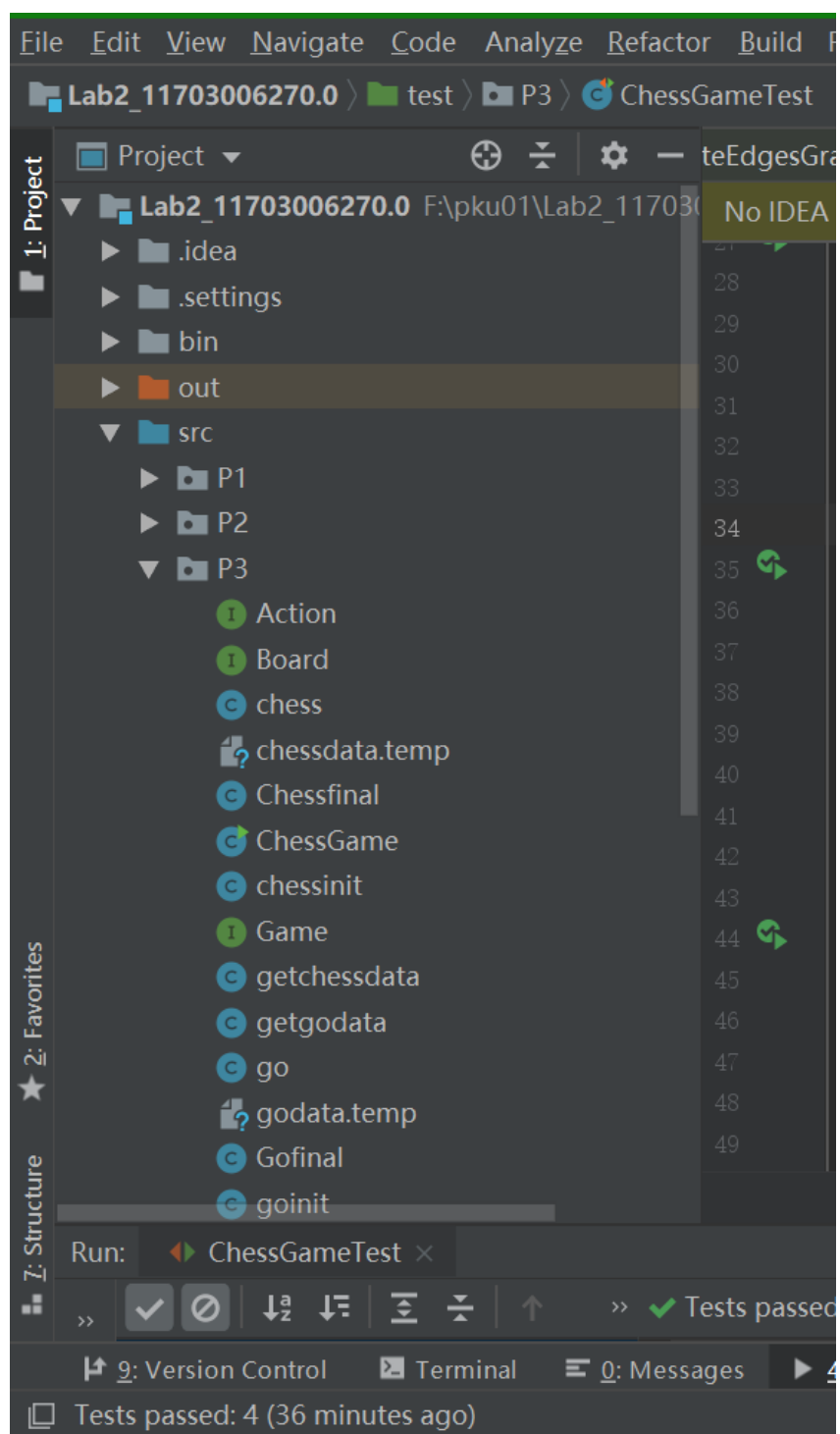
6.1 实验过程中收获的经验教训	35
6.2 针对以下方面的感受	35

1 实验目标概述

本次实验训练抽象数据类型（ADT）的设计、规约、测试，并使用面向对象编程（OOP）技术实现 ADT。具体来说：

- 针对给定的应用问题，从问题描述中识别所需的 ADT；
- 设计 ADT 规约（pre-condition、post-condition）并评估规约的质量；
- 根据 ADT 的规约设计测试用例；
- ADT 的泛型化；
- 根据规约设计 ADT 的多种不同的实现；针对每种实现，设计其表示（representation）、表示不变性（rep invariant）、抽象过程（abstraction function）
- 使用 OOP 实现 ADT，并判定表示不变性是否违反、各实现是否存在表示泄露（rep exposure）；
- 测试 ADT 的实现并评估测试的覆盖度；
- 使用 ADT 及其实现，为应用问题开发程序；
- 在测试代码中，能够写出 testing strategy 并据此设计测试用例。

2 实验环境配置



<https://github.com/ComputerScienceHIT/Lab2-1170300627>

3 实验过程

请仔细对照实验手册，针对三个问题中的每一项任务，在下面各节中记录你的实验过程、阐述你的设计思路和问题求解思路，可辅之以示意图或关键源代码加以说明（但千万不要把你的源代码全部粘贴过来！）。

3.1 Poetic Walks

我觉得该实验分成了三个部分，第一部分是完成以边为主的图结构创建以及实现图结构对应的方法和边类对应的方法，第二部分是实现以顶点为基础的图结构的创建以及实现顶点中的方法的实现。在最后一部分的诗意漫步中需要完成的是在之前已经实现的图结构的基础上寻找关联度最紧密的顶点，从而完成诗歌的创作。

3.1.1 Get the code and prepare Git repository

直接点击 download 下载就好了

此电脑 > 桌面 > 课件 > 软件构造 > Spring2019_HITCS_SC_Lab2-master >			
	名称	修改日期	类型
	P1	2019/3/4 10:44	文件夹
	P4	2019/3/4 10:44	文件夹
	每天_3次作业_version.p		

然后该文件夹就会被下载到本地
最后再解压一下，导入编译器
完成

3.1.2 Problem 1: Test Graph <String>

```
ConcreteEdgesGraph<String> graph = new ConcreteEdgesGraph<>();
String s = "ab5bc6cd5";
graph.set( source: "a", target: "b", weight: 5);
graph.set( source: "b", target: "c", weight: 6);
graph.set( source: "c", target: "d", weight: 5);
assertEquals(s, graph.toString());

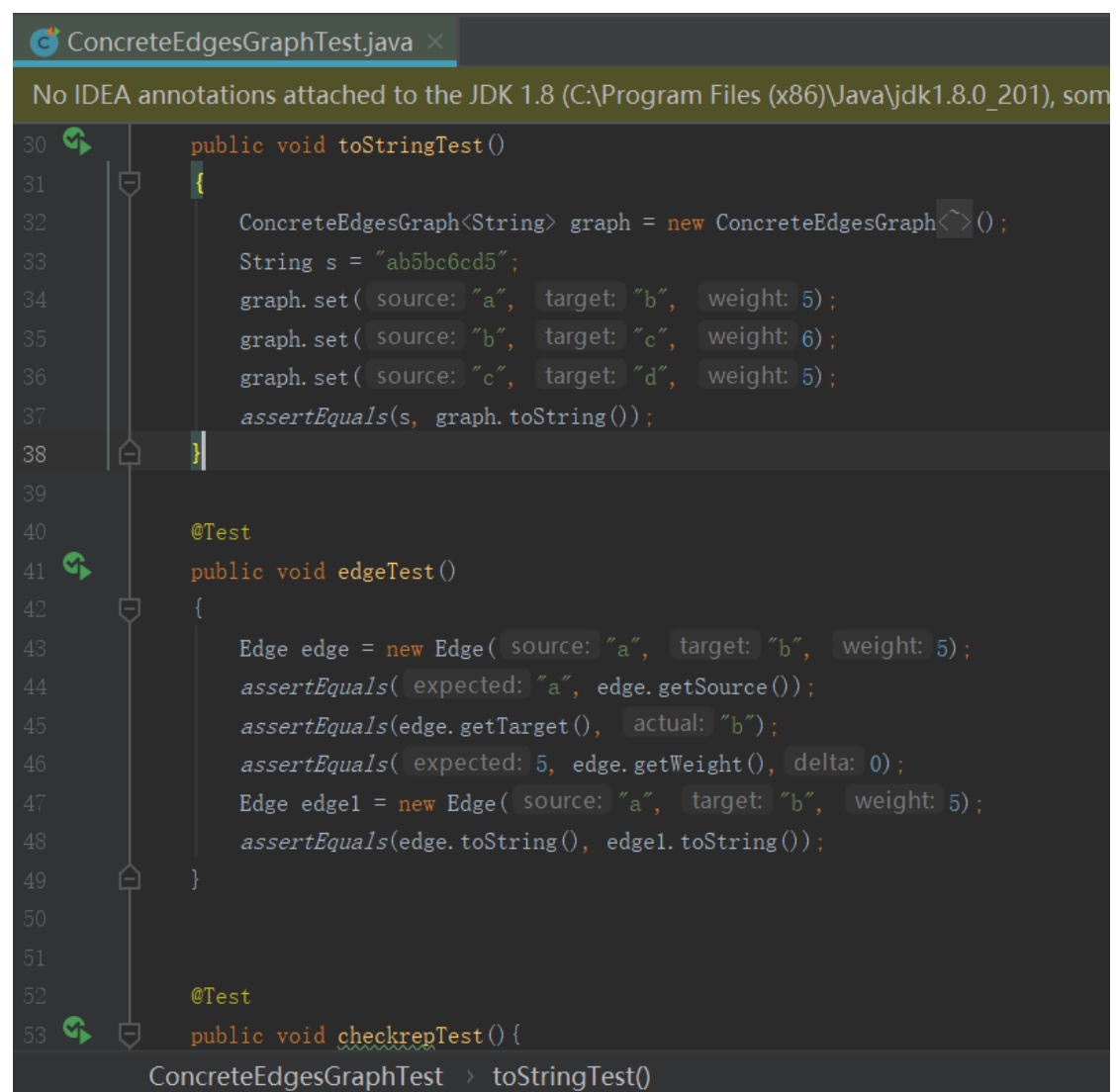
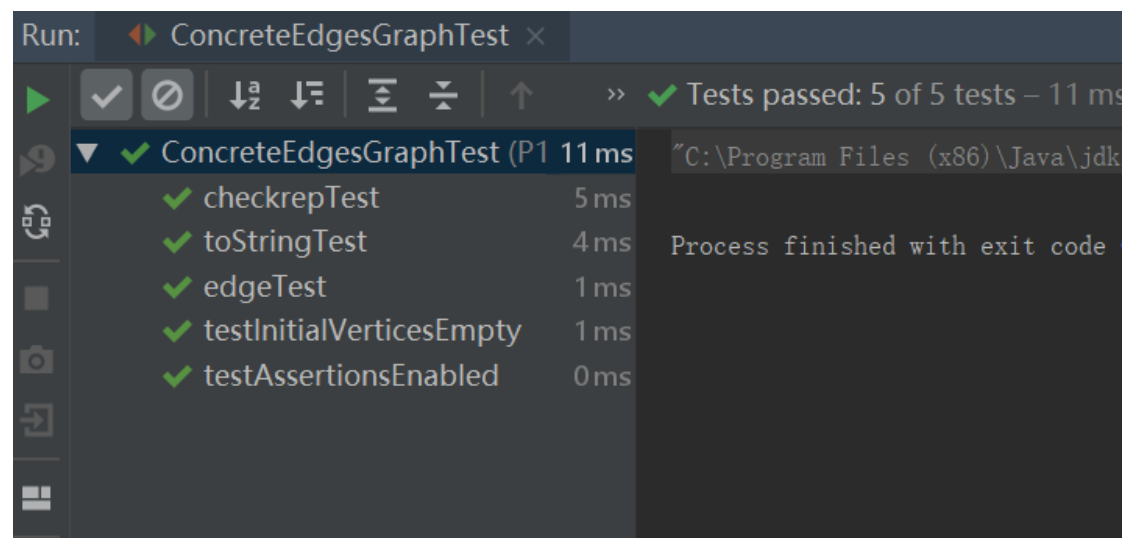
@Test
public void edgeTest()
{
    Edge edge = new Edge( source: "a", target: "b", weight: 5);
    assertEquals( expected: "a", edge.getSource());
    assertEquals(edge.getTarget(), actual: "b");
    assertEquals( expected: 5, edge.getWeight(), delta: 0);
    Edge edge1 = new Edge( source: "a", target: "b", weight: 5);
    assertEquals(edge.toString(), edge1.toString());
}

@Test
public void checkrepTest() {
    Edge edge = new Edge( source: "a", target: "b", weight: 5);
    assertFalse(!edge.checkRep());
}

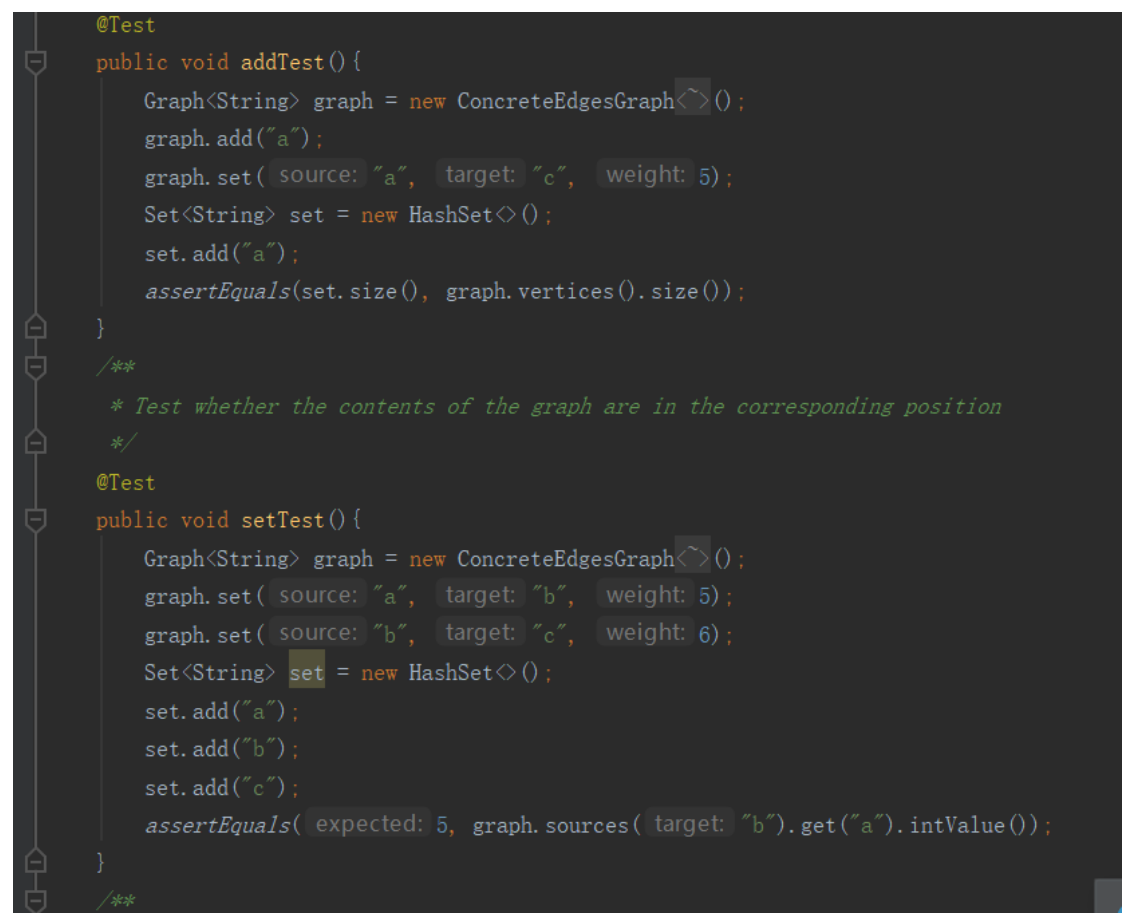
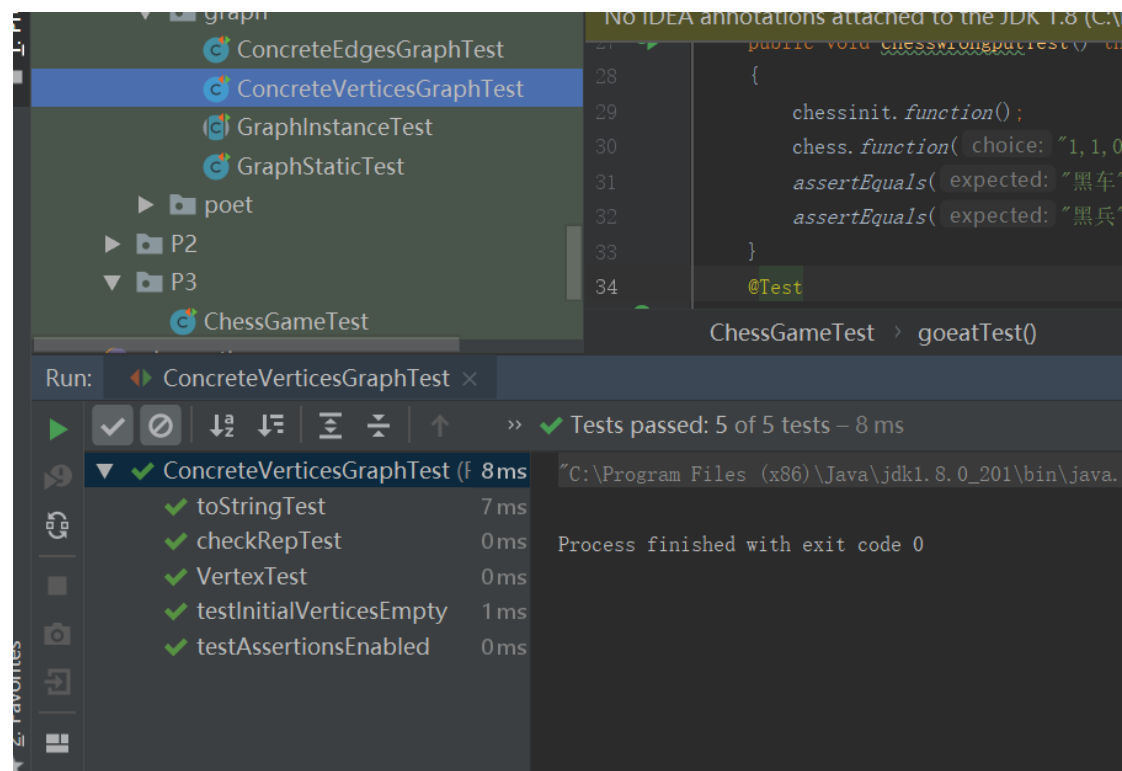
ConcreteEdgesGraphTest

@Test
public void toStringTest()
{
    ConcreteEdgesGraph<String> graph = new ConcreteEdgesGraph<>();
    String s = "ab5bc6cd5";
    graph.set( source: "a", target: "b", weight: 5);
    graph.set( source: "b", target: "c", weight: 6);
    graph.set( source: "c", target: "d", weight: 5);
    assertEquals(s, graph.toString());
}
```

在针对第一个测试函数进行完成的时候我对是否能够添加边, 以及不变量检查都进行了测试对于需要重写的 toString 方法也针对我写的测试用例进行了测试, 结果显示我的测试用例和我的程序能对应上



对于边的测试和上面顶点测试的测试内容相同, 测试的结果也是相同我选择的测试用例我的函数都能够实现功能



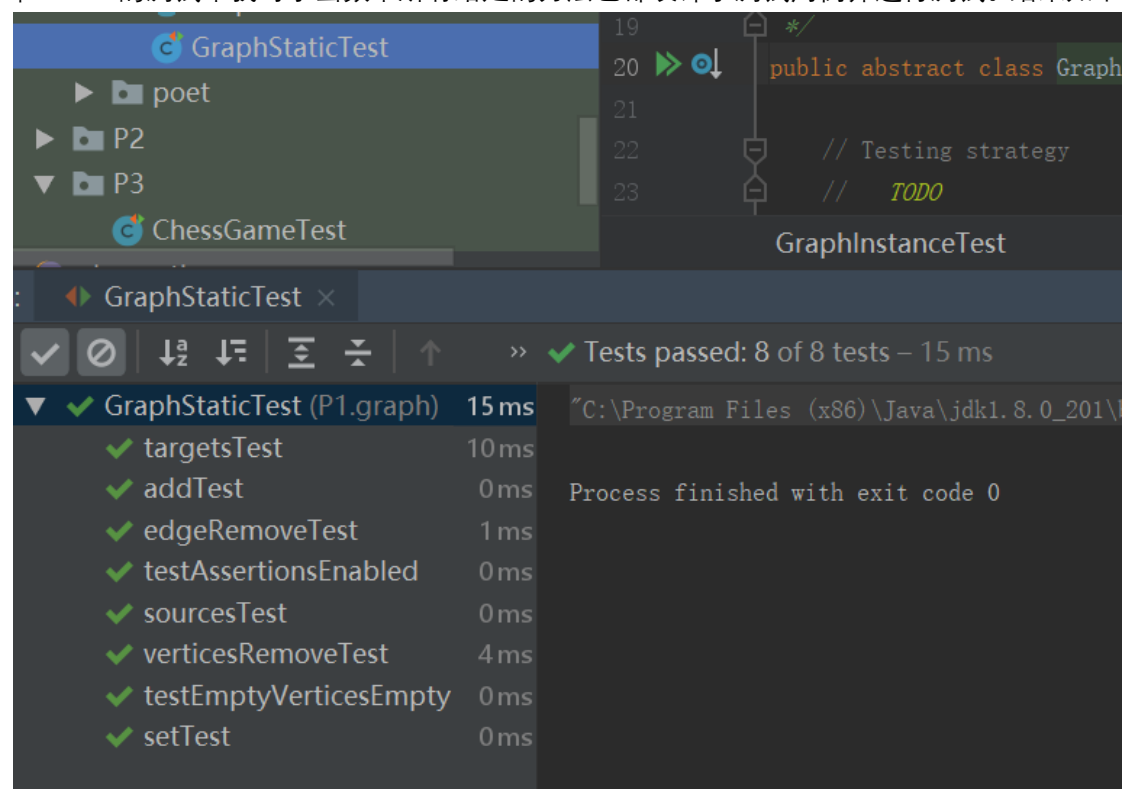
```
public void edgeRemoveTest() {
    Graph<String> graph = new ConcreteEdgesGraph<>();
    graph.add("a");
    graph.add("b");
    graph.remove(vertex: "a");
    graph.remove(vertex: "b");
    assertEquals(graph.vertices(), Collections.EMPTY_SET);
}

/**
 * Since the data type we set before is String, and the data type in the inter
 * we can only use the set method to add, and then add the node and then dele
 */

@Test
public void verticesRemoveTest() {
    Graph<String> graph = new ConcreteVerticesGraph<>();
    graph.set(source: "a", target: "b", weight: 5);
    graph.remove(vertex: "a");
    graph.remove(vertex: "b");
    assertEquals(graph.vertices(), Collections.EMPTY_SET);
}
```

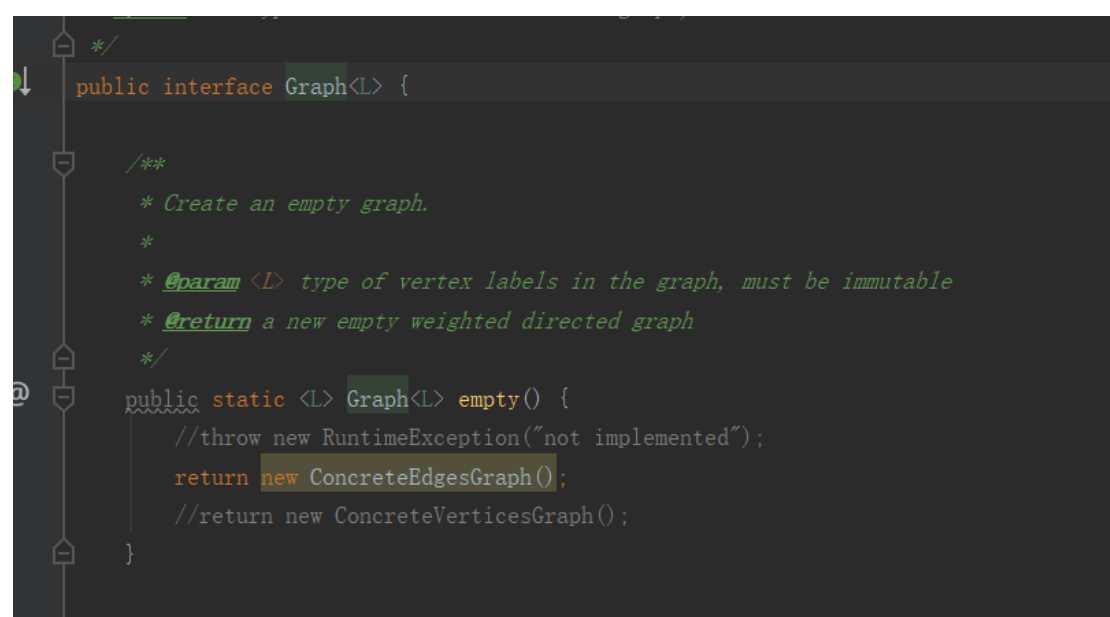
```
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
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
```

在 static 的测试中我对于函数中所有给定的方法也都设计了测试用例并进行测试。结果如下



以下各部分，请按照 MIT 页面上相应部分的要求，逐项列出你的设计和实现思路/过程/结果。

3.1.3 Problem 2: Implement Graph <String>



在进行该项功能的时候只需要把 L 都换成 String 即可，因为 java 在范围上可以向下兼容，但是不可以向上兼容。

3.1.3.1 Implement ConcreteEdgesGraph

Add 函数中只需要向 vertices 中 addvertex 就行了, 重要的我觉得就是对于重复情况的判断, 用来保证程序的健壮性

```
// TODO
@Override public boolean add(String vertex) {
    //throw new RuntimeException("not implemented");
    for(String it : vertices) {
        if(it.equals(vertex)) return false;
    }
    vertices.add(vertex);
    return true;
}
```

在 set 函数中和上一个问题一样, 主要的还是对于是否有重复情况的判断, 以及输入的 weight 情况的判断, 在判断之后直接保存结果就好了

```
@Override public int set(String source, String target, int weight) {
    //throw new RuntimeException("not implemented");
    if (source.equals(target)) throw new RuntimeException("Invalid input");
    for(int i = 0; i < edges.size(); i++) {
        if(edges.get(i).getSource().equals(edges.get(i).getTarget())) {
            if(weight == 0) {
                int ending = edges.get(i).getWeight();
                edges.remove(i);
                return ending;
            } else {
                int ending = edges.get(i).getWeight();
                edges.remove(i);
                edges.add(new Edge(source, target, weight));
                return ending;
            }
        }
    }
    edges.add(new Edge(source, target, weight));
    return 0;
}
```

Remove 在判断了是否存在之后直接删除就好了

```

@Override public boolean remove(String vertex) {
    //throw new RuntimeException("not implemented");
    if(!vertices.contains(vertex)) return false;
    Iterator<Edge> it=edges.iterator();
    while(it.hasNext()){
        Edge e=it.next();
        if(e.getSource().equals(vertex)||e.getTarget().equals(vertex)) it.remove();
    }
    vertices.remove(vertex);
    return true;
}

```

在 set 函数中注意防御性拷贝，把所有的元素都新弄一个然后再返回就好了，其实 String 并不需要如此操作，因为 string 是不可变类型，但是我想这写程序的时候规范一下，然后就做了一个无用的防御性拷贝来规范自己的编程习惯

```

@Override public Set<String> vertices() {
    //throw new RuntimeException("not implemented");
    HashSet<String> ending =new HashSet<>();
    for(String it : vertices) {
        ending.add(new String(it));
    }
    return ending;
}

```

Map 返回一个 map 在程序中便利一下所有的情况，装进去就好了

```

@Override public Map<String, Integer> sources(String target) {
    //throw new RuntimeException("not implemented");
    TreeMap<String, Integer> ending = new TreeMap<>();
    for(Edge it : edges) {
        if(it.getTarget().equals(target)) ending.put(new String(it.getSource()), new Integer(it.getWeight()));
    }
    return ending;
}

```

构造器，感觉没什么写的必要，因为默认的构造器就是下面我写的东西，但是老师让写我就把这个原型给写上吧权当加深自己对于 java 的理解了

```

// TODO constructor
public ConcreteEdgesGraph() {
}

```

Tostring 虽然编译器自带重写功能但是为了表示我写了作业我还是自己实现了这个函数，虽然在格式上不是很规整，但是所含信息都覆盖到了

```
// TODO toString()
@Override
public String toString() {
    String ending = new String();
    for(Edge it : edges) {
        ending = ending + it.getSource();
        ending = ending + it.getTarget();
        ending = ending + it.getWeight();
    }
    return ending;
}
```

在边的类中我为了保证程序的不变性，所以我做的第一点改进就是把所有的变量都弄成了 private final 的形式，有一份是程序里有并且提供给用户用的，还有一份是我留在类中的，作用只是核对另一份是否正确，所以 checkrep 函数就是在这个基础上实现的，检查是否有字段被修改

```
class Edge {

    // TODO fields
    // Representation invariant:
    // TODO

    private final String source;
    private final String target;
    private final String source1;
    private final String target1;
    private final Integer weight1;
    private final Integer weight;

    // TODO constructor
```

```

    }

    // TODO checkRep
    public boolean checkRep() {
        if(!source.equals(source1)) return false;
        if(!target.equals(target1)) return false;
        if(!weight.equals(weight1)) return false;
        return true;
    }

    // TODO methods

```

虽然 string 并不需要防御性拷贝但是我依旧进行了一下防御性拷贝
在 toString 中我也是自己写了一个 toString

```

// TODO
public Integer getWeight() { return new Integer(weight); //堆内存中 }
public String getSource() { return new String(source); }
public String getTarget() { return new String(target); }
// TODO toString()
@Override
public String toString() {
    return source.toString() + "spaceforsplit" + target.toString() + "spaceforsplit" + weight.toString() ;
}
}

```

3.1.3.2 Implement ConcreteVerticesGraph

图换一个方式实现

构造器，依旧是直接写出来了，虽然我也不知道老师设置这个问题到底有什么意义

```

// TODO constructor
public ConcreteVerticesGraph() {
    // TODO Auto-generated constructor stub
}

// TODO checkRep

```

Add 函数用于添加顶点，那就直接添加就好了

Map 找顶点和权之间的对应关系，依旧是遍历

```

@Override public boolean add(String vertex) {
    //throw new RuntimeException("not implemented");
    for(Vertex list:vertices)
    {
        if(list.getName().equals(vertex))
        {
            return false;
        }
    }

    Map<String, Integer> nextVertex = new HashMap<>();
    Vertex v = new Vertex(vertex, nextVertex);
    vertices.add(v);
    return true;
}

```

在这个问题中的 set

有一些复杂因为所给的结构并不能简介的完成任务，对于 wight 的情况，以及顶点是否有重复的情况都需要进行判断。还得看看是否给的两个点在我的顶点集中并没有。只有在判断结束之后，所有的要求都符合了之后才能将所给的条件装入。或者如果输入的 weight 是 0 那就代表该条边不存在了，就得删除此条边

```

@Override public int set(String source, String target, int weight) {
    //throw new RuntimeException("not implemented");
    if(weight<0)
        throw new RuntimeException("Invalid weight");
    if(source.equals(target))
        throw new RuntimeException("Invalid input");
    boolean targetisExist = true;
    for (Vertex list : vertices) {
        if (list.getName().equals(target))
            targetisExist = false;
        if (list.getName().equals(source)) {
            if (list.nextVertex.containsKey(target)) {
                list.nextVertex.remove(target);
                if (weight > 0) {
                    list.nextVertex.put(target, weight);
                    if (targetisExist) {
                        Map<String, Integer> map1 = new HashMap<>();
                        Vertex vertex1 = new Vertex(target, map1);
                        vertices.add(vertex1);
                    }
                }
                return weight;
            }
        }
        return weight;
    } else {

```



```

    } else {
        if (weight > 0) {
            if (targetisExist) {
                Map<String, Integer> map1 = new HashMap<>();
                Vertex vertex1 = new Vertex(target, map1);
                vertices.add(vertex1);
            }
            list.nextVertex.put(target, weight);
            return weight;
        }
        return weight;
    }
}

if (weight > 0) {
    Map<String, Integer> map = new HashMap<>();
    map.put(target, weight);
    Vertex vertex = new Vertex(source, map);
    vertices.add(vertex);
    if (targetisExist) {
        Map<String, Integer> map1 = new HashMap<>();
        Vertex vertex1 = new Vertex(target, map1);
        vertices.add(vertex1);
    }
}

```

Remove 直接删除就好了，如果没有对应的就返回一个 false

```

2 ①↑  @Override public boolean remove(String vertex) {
3      //throw new RuntimeException("not implemented");
4      for (Vertex list : vertices)
5      {
6          if(list.getName().equals(vertex))
7          {
8              vertices.remove(list);
9              return true;
10         }
11     }
12
13     return false;
14 }

```

在 set 的时候返回一个集合就好了，遍历一下集合中的元素，否则，如果直接返回集合的话，用户就可能对集合进行修改，违背了我们不变量的初衷

```
26 ① ↑ @Override public Set<String> vertices() {  
27      //throw new RuntimeException("not implemented");  
28      Set<String> set = new HashSet<>();  
29      for(Vertex list:vertices)  
30      {  
31          set.add(list.getName());  
32      }  
33  
34      return set;
```

下面的这两个函数其实在逻辑上是一样的, 只不过一个是找出来起始顶点一个是找出来终止顶点罢了。只不过是遍历一下而已

```
@Override public Map<String, Integer> sources(String target) {  
    //throw new RuntimeException("not implemented");  
    Map<String, Integer> map = new HashMap<>();  
    for(Vertex list:vertices)  
    {  
        for(int i=0;i<list.nextVertex.size();i++)  
            map.put(list.getName(), list.nextVertex.get(i));  
    }  
  
    return map;  
}  
  
@Override public Map<String, Integer> targets(String source) {  
    //throw new RuntimeException("not implemented");  
    Map<String, Integer> map = new HashMap<>();  
    for(Vertex list:vertices)  
    {  
        map.putAll(list.nextVertex);  
    }  
  
    return map;  
}
```

Tostring 在 string 中返回一下所有的信息就好了

```
// TODO toString()
@Override
public String toString() {
    String ending = new String();
    for(int i = 0; i < vertices.size(); i++) {
        ending = ending + vertices.get(i).getName();
    }
    return ending;
}

/**
 * TODO specification
 */
```

3.1.4 Problem 3: Implement generic Graph<L>

在 Graph 中只有这一个函数需要完成

```
19  * @param <L> type of vertex labels in this graph, must be immutable
20
21  public interface Graph<L> {
22
23      /**
24       * Create an empty graph.
25       *
26       * @param <L> type of vertex labels in the graph, must be immutable
27       * @return a new empty weighted directed graph
28       */
29      @ public static <L> Graph<L> empty() {
30          //throw new RuntimeException("not implemented");
31          return new ConcreteEdgesGraph();
32          //return new ConcreteVerticesGraph();
33      }
34
35      /**
```

3.1.4.1 Make the implementations generic

只需要把带函数中部分带 String 的地方都给改成 L 就好了

3.1.4.2 Implement Graph.empty()

直接在里面新建一个对象就好了，有两个 u 第项可供选择，用什么随意。因为都是实现了接口的类

```
public static <L> Graph<L> empty() {  
    //throw new RuntimeException("not implemented");  
    return new ConcreteEdgesGraph();  
    //return new ConcreteVerticesGraph();  
}
```

3.1.5 Problem 4: Poetic walks

3.1.5.1 Test GraphPoet

老师给的文件中自带的 Main 测试已经被我完成了，在此基础上我觉得多点的测试已经完成了，剩下的应该就是单点的测试了，所以我就写了一个针对于单点测试的函数，看看能给出什么样的结果

测试结果显示我的函数在单点和多点的情况下都唔那个完成任务

The screenshot displays an IDE with two main windows. The top window, titled 'Run: Main x', shows the output of a Java program. The output text is: 'C:\Program Files (x86)\Java\jdk1.8.0_201\bin\java.exe' ... Test the system. >>> Test of the system. Process finished with exit code 0. The bottom window shows the 'GraphPoetTest' class with two test methods: 'testAssertionsEnabled()' and 'testLengthOneInput()'. The test results at the bottom indicate that both tests passed: 'Tests passed: 2 of 2 tests - 17 ms'. The 'testLengthOneInput' test took 17 ms, and the 'testAssertionsEnabled' test took 0 ms. The process finished with exit code 0.

3.1.5.2 Implement GraphPoet

Graphpoet 函数先把我的语料库给读进来，弄到之前我写的图里去，为下一步的检索做铺垫

```
public GraphPoet(File corpus) throws IOException {
    //throw new RuntimeException("not implemented");
    Scanner sc = new Scanner(corpus);
    sc.useDelimiter("\\s|[.,]");
    List<String> list = new ArrayList<>();
    while(sc.hasNext()){
        String string = sc.next();
        if(string.length()>0)
            list.add(string.toLowerCase());
    }
    sc.close();
    int i=0;
    while(i<list.size()-1){
        String start = list.get(i);
        String end = list.get(i+1);
        int count = 1;
        i++;
        while(i<list.size()-1&&list.get(i+1).equals(end)){
            i++;count++;
        }
        graph.set(start, end, count);
    }
}
```

我的语料库已经被我导入到内存中了，所以剩下我需要做的就是检索了，针对由于所有的点我都两两遍历一下，寻找之间的权值最大的情况，然后再输出就好了

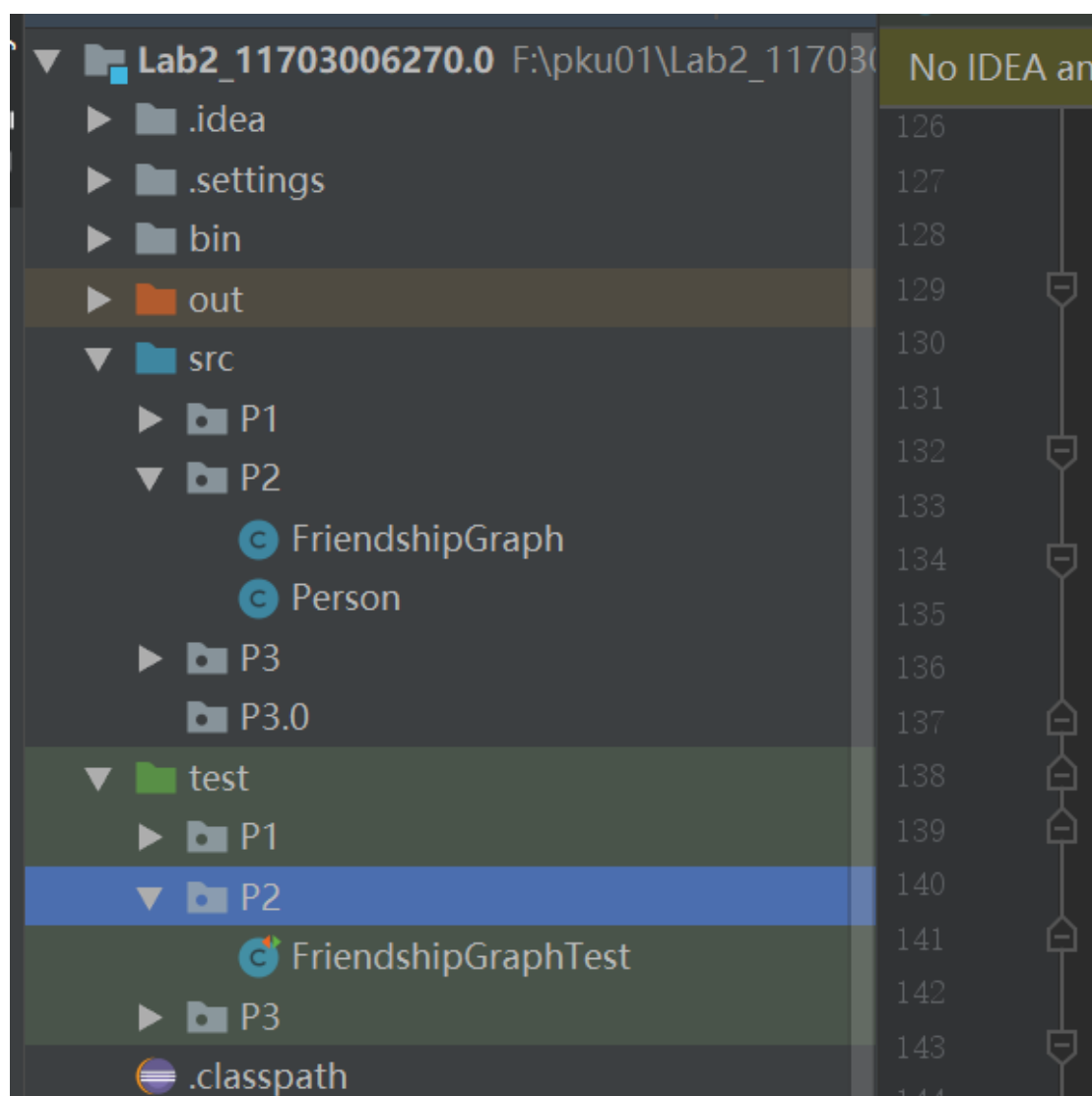
```
*/  
public String poem(String input) {  
    // throw new RuntimeException("not implemented");  
    String ending = new String();  
    if(input.length() <= 0) throw new RuntimeException("Illegal input");  
    Scanner sc = new Scanner(input);  
    sc.useDelimiter("\\s");  
    ArrayList<String> words = new ArrayList<>();  
    while(sc.hasNext()) {  
        String temp = sc.next();  
        if(temp.length() > 0)  
            words.add(temp);  
    }  
    sc.close();  
    for(int i = 0; i < words.size() - 1; i++) {  
        String source1 = words.get(i);  
        String target1 = words.get(i + 1);  
        int max = 0;  
        String addf = new String();  
        Map<String, Integer> start = graph.targets(source1.toLowerCase());  
        for(String it1 : start.keySet()) {  
            int inttempl = start.get(it1.toLowerCase());  
            Map<String, Integer> startt = graph.targets(it1.toLowerCase());  
        }  
    }  
}
```

```
int max = 0;
String addf = new String();
Map<String,Integer> start= graph.targets(source1.toLowerCase());
for(String it1 : start.keySet()) {
    int inttemp1 = start.get(it1.toLowerCase());
    Map<String,Integer> startt= graph.targets(it1.toLowerCase());
    for(String it2 : startt.keySet()) {
        int inttemp2 = startt.get(it2);
        if(inttemp1 + inttemp2 >=max&&it2.equals(target1)) {
            max = inttemp1 +inttemp2;
            addf = it1;
        }
    }
}
if(addf.length() !=0) words.add( index: i + 1, addf);
}
String ending0 = new String();
for(int i = 0 ;i < words.size();i++) {
    ending0 = ending0 + words.get(i);
    if(i!=words.size()-1) ending0 = ending0 + " ";
}
return ending0;
}
```

3.1.5.3 Graph poetry slam

上一问中说了

3.1.6 Before you're done



在这里给出你的项目的目录结构树状示意图。

3.2 Re-implement the Social Network in Lab1

就是在实验 1 的代码的基础上对不变量进行一下补充就好了

3.2.1 FriendshipGraph 类

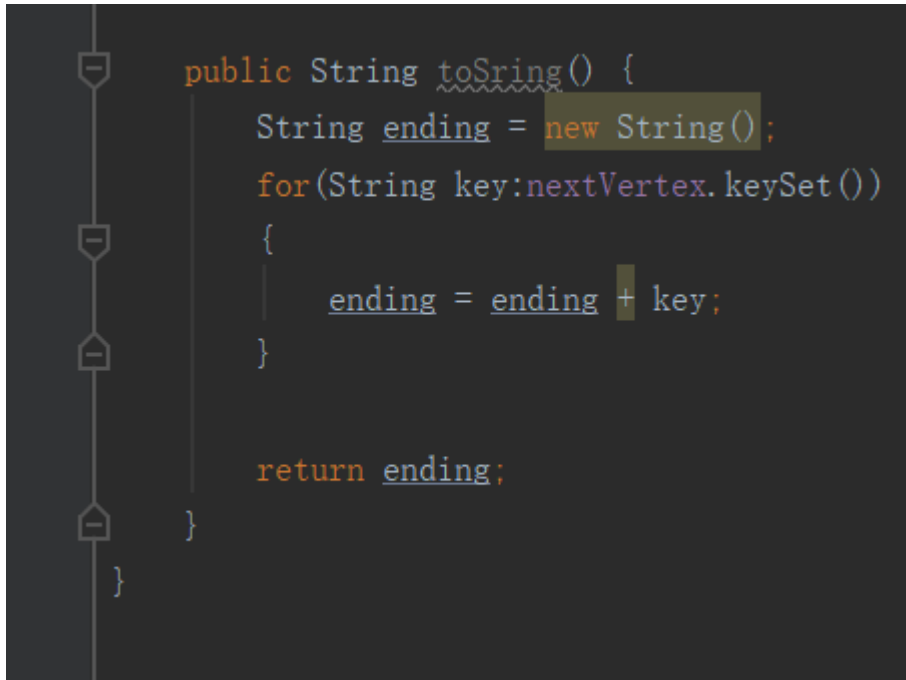
此次实验中在 `getdistance` 的功能上实现的更加谨慎，不改变原有数据是我在此次实验中的追求，也就是维持程序和变量的不变性


```
IDEA annotations attached to the JDK 1.8 (C:\Program Files (x86)\Java\jdk1.8.0_201), some  
  
String name = list.get(location);  
p = getPerson(name);  
p.isVisited = true;  
if(p.getName().equals(p2.getName()) || p.getName() == p2.getName()) {  
    isFinded = true;  
}  
else{  
    for(String key:p.nextVertex.keySet()) {  
        if(getPerson(key).isVisited==false) {  
            list1.add(key);  
        }  
    }  
}  
if(isFinded)  
    return flag;  
location++;  
if(location==list.size()) {  
    list = new ArrayList<>();  
    for(int i=0;i<list1.size();i++)  
        list.add(list1.get(i));  
    flag++;  
    list1 = new ArrayList<>();  
    location = 0;  
}
```

给出你的设计和实现思路/过程/结果。

3.2.2 Person 类

在上次的基础上加上了 toString 的功能函数



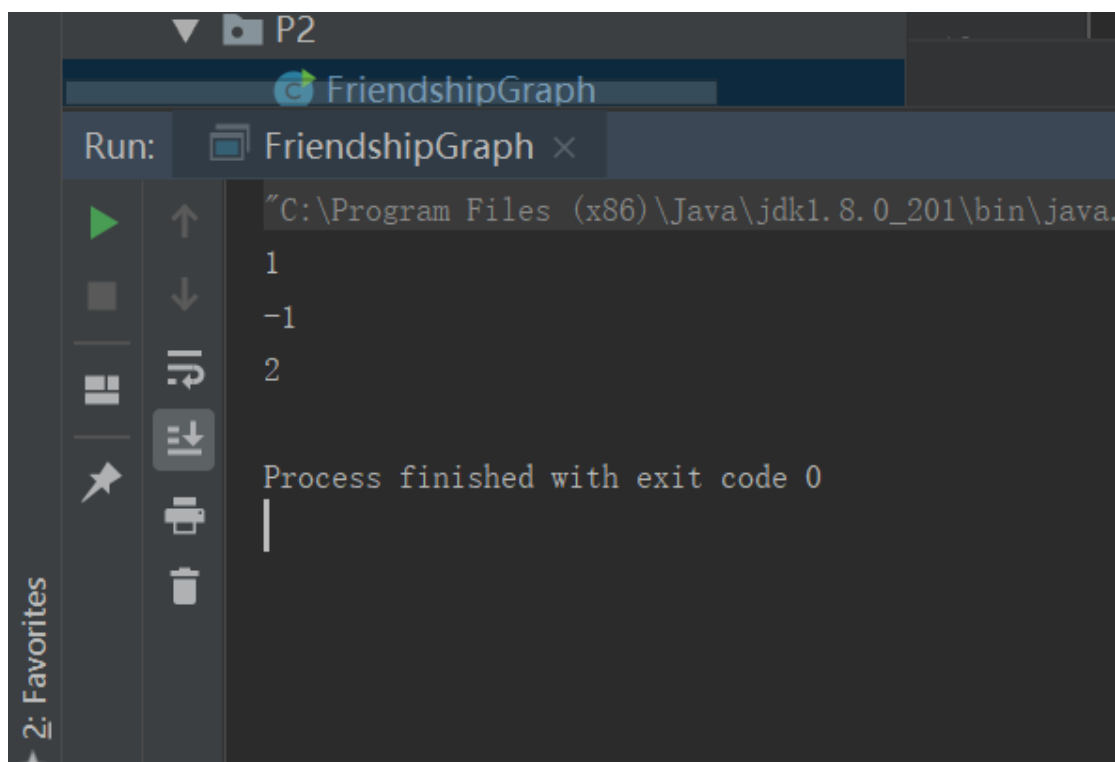
```
public String toSring() {  
    String ending = new String();  
    for(String key:nextVertex.keySet())  
    {  
        ending = ending + key;  
    }  
  
    return ending;  
}
```

3.2.3 客户端 main()

客户端也是上次实验中直接给出来的。。。

```
/* public static void main(String[] args) {  
    Map<String, Integer> map1 = new HashMap<>();  
    Map<String, Integer> map2 = new HashMap<>();  
    Map<String, Integer> map3 = new HashMap<>();  
    Map<String, Integer> map4 = new HashMap<>();  
  
    Person rachel = new Person("Rachel", map1);  
    Person ross = new Person("Ross", map2);  
    Person ben = new Person("Ben", map3);  
    Person kramer = new Person("Kramer", map4);  
  
    FriendshipGraph.addVertex(rachel);  
    FriendshipGraph.addVertex(ross);  
    FriendshipGraph.addVertex(ben);  
    FriendshipGraph.addVertex(kramer);  
  
    FriendshipGraph.addEdge(rachel, ross);  
    FriendshipGraph.addEdge(ross, rachel);  
    FriendshipGraph.addEdge(ross, ben);  
    FriendshipGraph.addEdge(ben, ross);  
}
```

运行结果如下



```
"C:\Program Files (x86)\Java\jdk1.8.0_201\bin\java.  
1  
-1  
2  
  
Process finished with exit code 0
```

3.2.4 测试用例

在测试测试用例的时候我的思路就是先弄出来几个我能算出来距离的测试用例, 再看看我的程序给出来的结果和实际的结果是否能对应上

```
FriendshipGraph. addEdge(P1, P2) ;  
FriendshipGraph. addEdge(P2, P1) ;  
FriendshipGraph. addEdge(P1, P3) ;  
FriendshipGraph. addEdge(P3, P1) ;  
FriendshipGraph. addEdge(P2, P4) ;  
FriendshipGraph. addEdge(P4, P2) ;  
FriendshipGraph. addEdge(P3, P4) ;  
FriendshipGraph. addEdge(P4, P3) ;  
FriendshipGraph. addEdge(P3, P5) ;  
FriendshipGraph. addEdge(P5, P3) ;  
FriendshipGraph. addEdge(P5, P6) ;  
FriendshipGraph. addEdge(P6, P5) ;
```

```
public void addVertexTest()  
{  
    Map<String, Integer> map1 = new HashMap<>() ;  
    Map<String, Integer> map2 = new HashMap<>() ;  
    Map<String, Integer> map3 = new HashMap<>() ;  
    Map<String, Integer> map4 = new HashMap<>() ;  
  
    Person rachel = new Person( name: "Rachel", map1) ;  
    Person ross = new Person( name: "Ross", map2) ;  
    Person ben = new Person( name: "Ben", map3) ;  
    Person kramer = new Person( name: "Kramer", map4) ;  
  
    ArrayList<Person> al = new ArrayList<>() ;  
    al.add(rachel) ;  
    al.add(ross) ;  
    al.add(ben) ;  
    al.add(kramer) ;  
}
```

FriendshipGraphTest

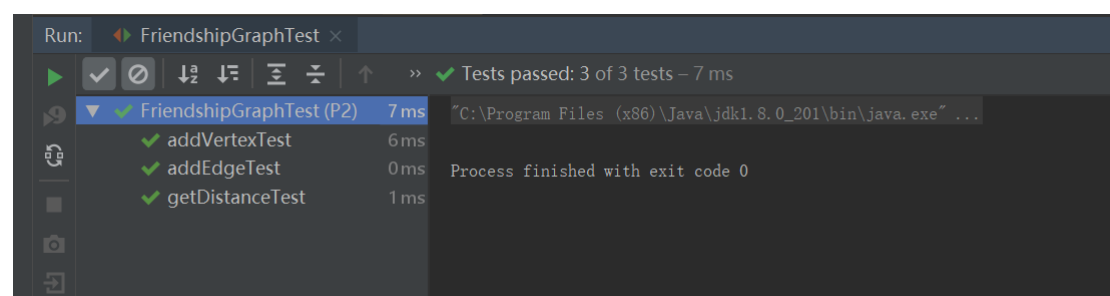
```
Map<String, Integer> map4 = new HashMap<>();

Person rachel = new Person( name: "Rachel", map1);
Person ross = new Person( name: "Ross", map2);
Person ben = new Person( name: "Ben", map3);
Person kramer = new Person( name: "Kramer", map4);

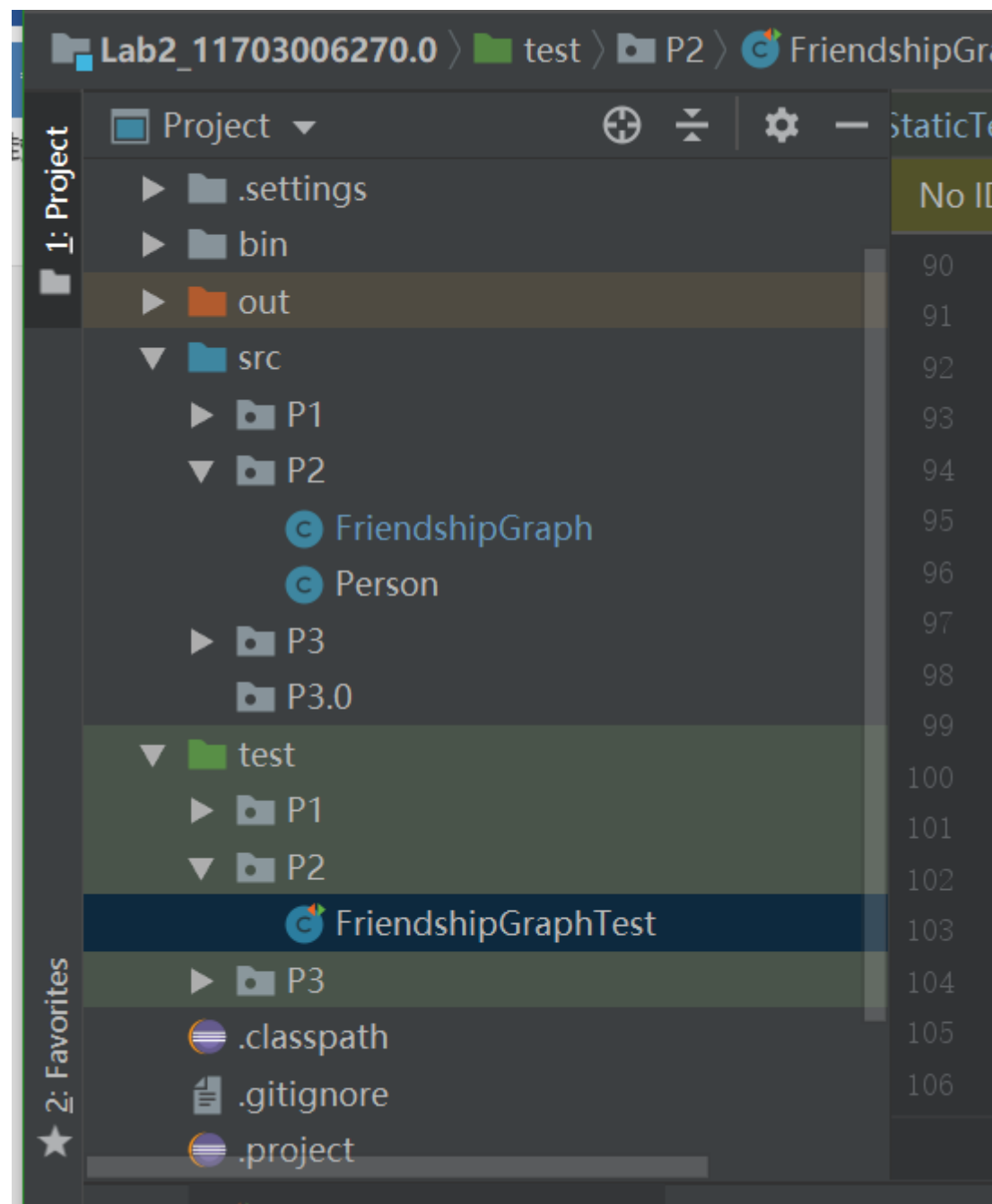
rachel.nextVertex.put(ross.getName(), 1);
ross.nextVertex.put(rachel.getName(), 1);
ross.nextVertex.put(ben.getName(), 1);
ben.nextVertex.put(ross.getName(), 1);

ArrayList<Person> al = new ArrayList<>();
al.add(rachel);
al.add(ross);
al.add(ben);
al.add(kramer);
```

FriendshipGraphTest

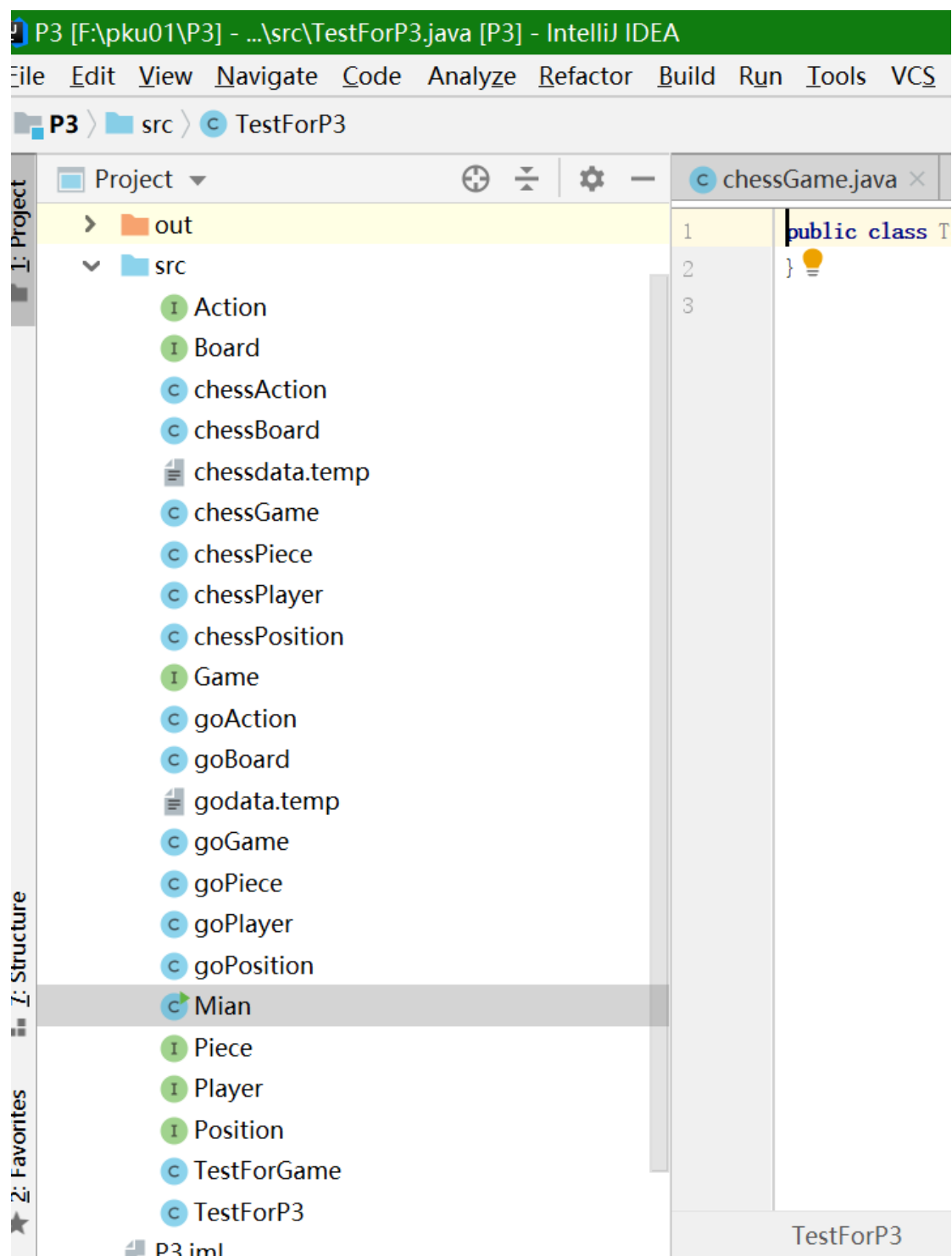


3.2.5 提交至 Git 仓库

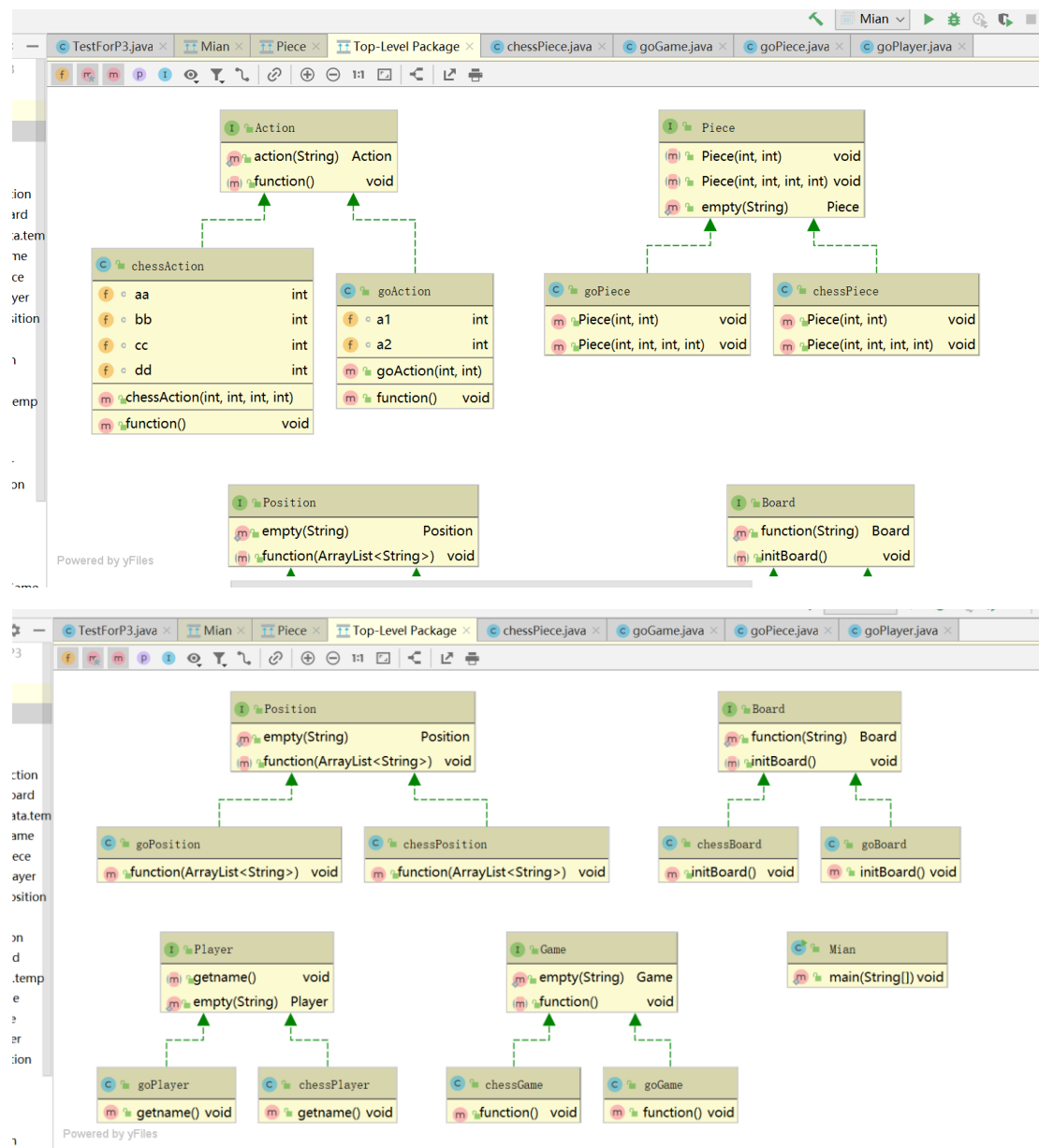


3.3 Playing Chess

3.3.1 ADT 设计/实现方案



下面的图中每个类都由两个接口实现。具体的依赖关系和函数请见下图



3.3.2 主程序 MyChessAndGoGame 设计/实现方案

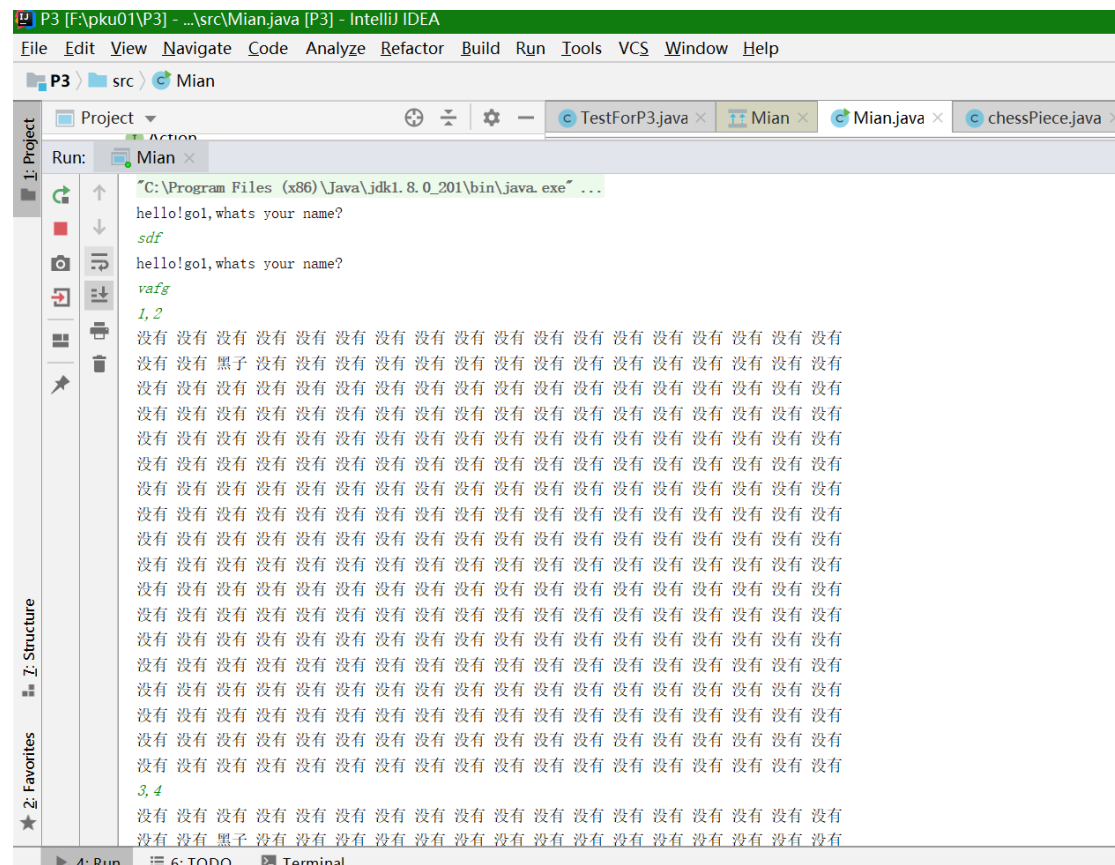
为了体现面向对象的思想，我把所有的函数都封装起来了，在用户端只要调用一下 `function` 就可以完成所有的功能了

```

1  import java.io.IOException;
2
3  public class Mian {
4      public static void main(String[] Zing) throws IOException { //别忘了最后在所有路径上都加上p3!!
5          //Game game1 = Game.empty("chess");
6          Game game1 = Game.empty("go");
7          game1.function(); //进行游戏
8      }
9  }
10

```

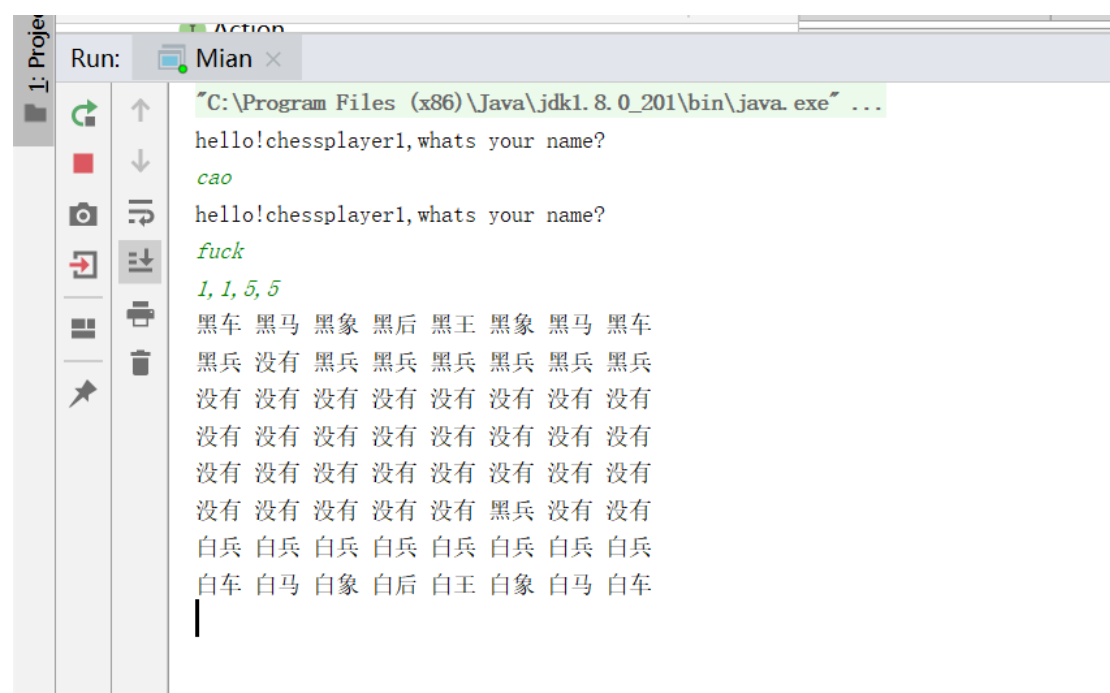
围棋运行过程如下



在下错了子的情况下会有如下的提示

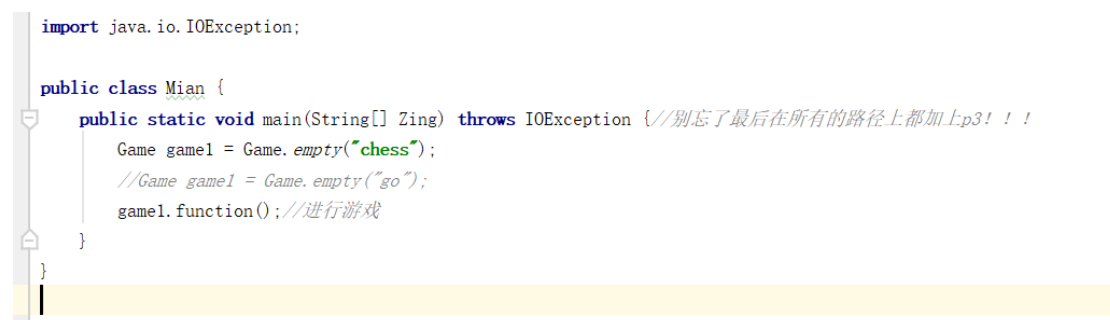


象棋的程序演示如下



```
Run: Mian x
"C:\Program Files (x86)\Java\jdk1.8.0_201\bin\java.exe" ...
hello!chessplayer1,whats your name?
cao
hello!chessplayer1,whats your name?
fuck
1, 1, 5, 5
黑车 黑马 黑象 黑后 黑王 黑象 黑马 黑车
黑兵 没有 黑兵 黑兵 黑兵 黑兵 黑兵 黑兵
没有 没有 没有 没有 没有 没有 没有 没有
没有 没有 没有 没有 没有 没有 没有 没有
没有 没有 没有 没有 没有 没有 没有 没有
没有 没有 没有 没有 没有 黑兵 没有 没有
白兵 白兵 白兵 白兵 白兵 白兵 白兵 白兵
白车 白马 白象 白后 白王 白象 白马 白车
```

此时的主程序如下



```
import java.io.IOException;

public class Mian {
    public static void main(String[] Zing) throws IOException { //别忘了最后在所有的路径上都加上p3! !!
        Game game1 = Game.empty("chess");
        //Game game1 = Game.empty("go");
        game1.function(); //进行游戏
    }
}
```

3.3.3 ADT 和主程序的测试方案

往里面放几组测试用例就好啦，看看得到的输出是不是我想要的输出

```
public void chesseatTest() throws IOException
{
    chessinit.function();
    chess.function("1, 1, 3, 4");
    chess.function("7, 7, 3, 4");
    chess.function("0, 0, 3, 4");
    chess.function("7, 6, 3, 4");
    chess.function("0, 1, 3, 4");
    chess.function("6, 6, 3, 4");
    assertEquals("没有", getchessdata.function().get(3*8+3));
}

@Test
public void chesswrongputTest() throws IOException
{
    chessinit.function();
    chess.function("1, 1, 0, 0");
    assertEquals("黑车", getchessdata.function().get(0));
    assertEquals("黑兵", getchessdata.function().get(1*8 + 1));
}

@Test
public void goeatTest() throws IOException
{
    goinit.function();
    go.function("1, 1");
    go.function("2, 2");
    go.function("2, 2");
    assertEquals("没有", getgodata.function().get(2*18+2));
}
```

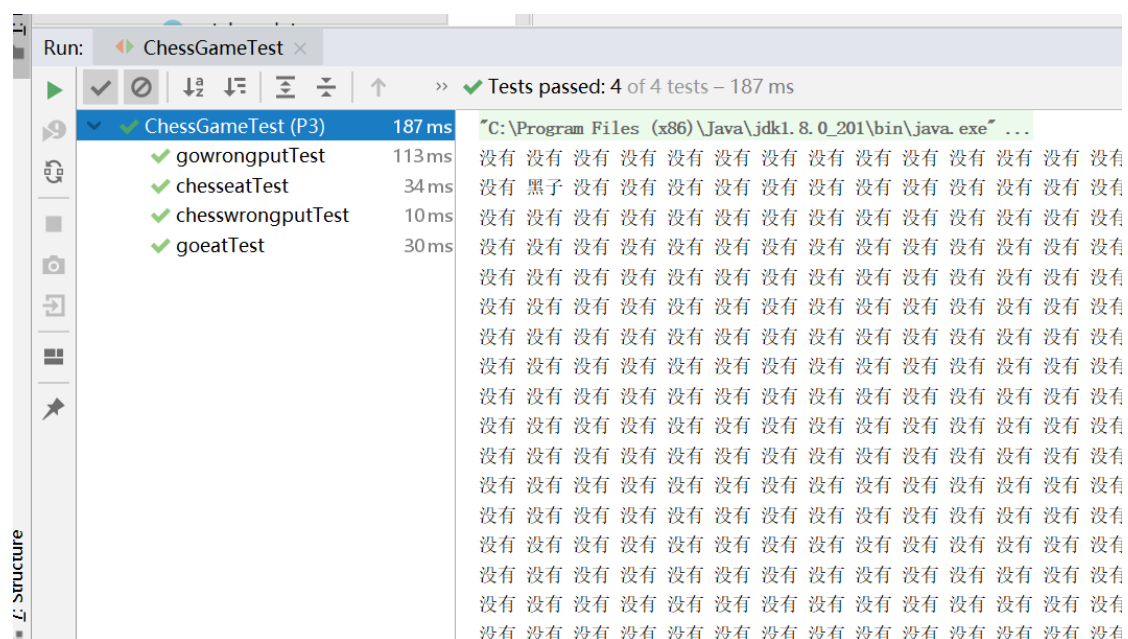
```

public void chesswrongputTest() throws IOException
{
    chessinit.function();
    chess.function("1,1,0,0");
    assertEquals("黑车", getchessdata.function().get(0));
    assertEquals("黑兵", getchessdata.function().get(1*8 + 1));
}

@Test
public void goeatTest() throws IOException
{
    goinit.function();
    go.function("1,1");
    go.function("2,2");
    go.function("2,2");
    assertEquals("没有", getgodata.function().get(2*18+2));
}

@Test
public void gowrongputTest() throws IOException
{
    goinit.function();
    go.function("1,1");
    go.function("10,10");
    go.function("1,1");
    assertEquals("黑子", getgodata.function().get(1*18 + 1));
}
}

```



介绍针对各 ADT 的各方法的测试方案和 testing strategy。

介绍你如何对该应用进行测试用例的设计，以及具体的测试过程。

3.4 Multi-Startup Set (MIT)

请自行设计目录结构。

注意：该任务为选做，不评判，不计分。

4 实验进度记录

请使用表格方式记录你的进度情况，以超过半小时的连续编程时间为一行。

每次结束编程时，请向该表格中增加一行。不要事后胡乱填写。

不要嫌烦，该表格可帮助你汇总你在每个任务上付出的时间和精力，发现自己不擅长的任务，后续有意识的弥补。

日期	时间段	计划任务	实际完成情况
3.21	14-19	实验 1	完成
3.22	14-19	实验 2	完成
3.33	14-19	实验 3	完成

5 实验过程中遇到的困难与解决途径

遇到的难点	解决途径
Empty 函数没法实现。。。题错了，得修改题目。。。	那咋整啊。。。我想了两周也没想明白。。。感觉按照 MIT 的课件老师给的题目是不可能实现的。老师说题错了我才改题目。。。希望以后出题谨慎点吧。别出那种无解的题了，别坑下一届学弟学妹了!!!

6 实验过程中收获的经验、教训、感想

6.1 实验过程中收获的经验教训

以后拖到最后两天再写软件构造作业。这样题目中的错误就会被别人发现了。在软件构造作业上赶进度是不可能的。

6.2 针对以下方面的感受

(1) 面向 ADT 的编程和直接面向应用场景编程，你体会到二者有何差异？

Adt 编程对于程序的服用更加方便

- (2) 使用泛型和不使用泛型的编程, 对你来说有何差异?

泛型可以兼容多种类型的对象, 使用起来更加方便

- (3) 在给出 ADT 的规约后就开始编写测试用例, 优势是什么? 你是否能够适应这种测试方式?

直接啊, 不用等到设计好程序签名之后再写测试用例

- (4) P1 设计的 ADT 在多个应用场景下使用, 这种复用带来什么好处?

直接提供了规范, 有一个统一化的规格。使用者使用起来更加方便, 直接按着说明书操作就好了

- (5) P3 要求你从 0 开始设计 ADT 并使用它们完成一个具体应用, 你是否已适应从具体应用场景到 ADT 的“抽象映射”? 相比起 P1 给出了 ADT 非常明确的 rep 和方法、ADT 之间的逻辑关系, P3 要求你自主设计这些内容, 你的感受如何?

感觉对于我的程序设计能力有着很好的提升作用。对于 adt 的理解更加深刻了

- (6) 为 ADT 撰写 specification, invariants, RI, AF, 时刻注意 ADT 是否有 rep exposure, 这些工作的意义是什么? 你是否愿意在以后编程中坚持这么做?

因为我们作为程序员我们需要保护类中的边类不被外界用户修改, 如果里面的数据大家都能修改的话, 那还怎么保护数据。

愿意, 因为用户的操作总是不能符合程序员的思想

- (7) 关于本实验的工作量、难度、deadline。

工作量不大, 没啥难度, 如果说有难度的话就应该是那道错题了

- (8) 《软件构造》课程进展到目前, 你对该课程有何体会和建议?

换个老师出题吧, 希望以后的题能没有错误