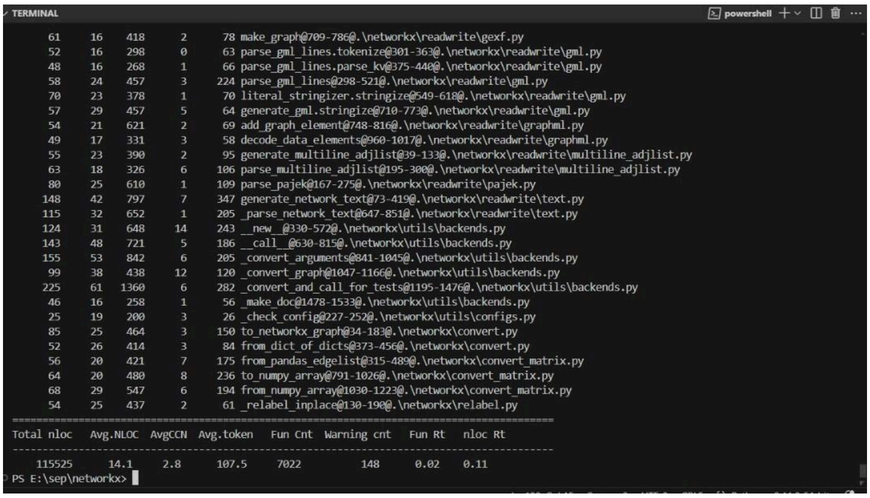
Report for Assignment 1

Project: NetworkX

URL: https://github.com/networkx/networkx

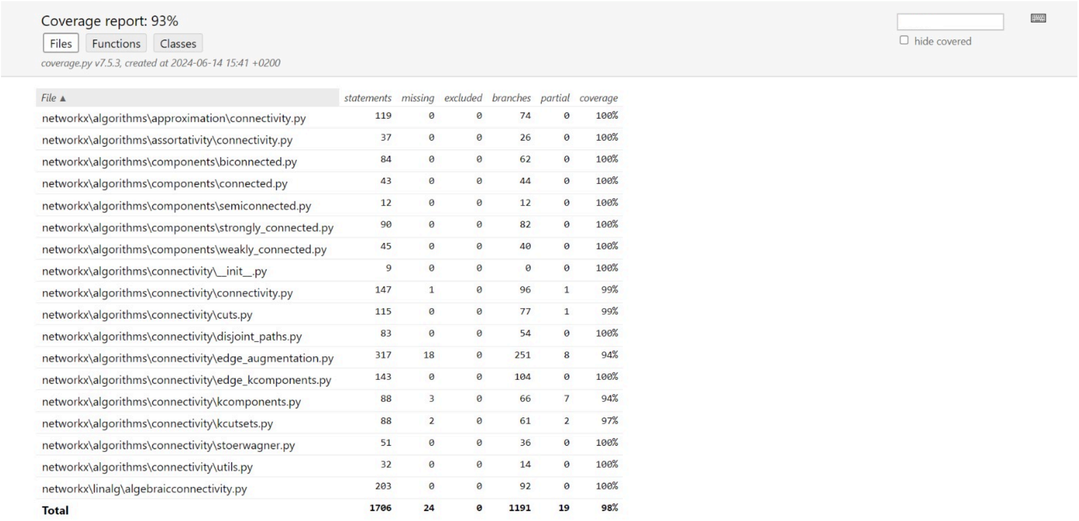
Lizard: 115,525 lines of code.



```
       61     16    418      2      78 make_graph@709-786@.\networkx\readwrite\gexf.py
       52     16    298      0      63 parse_gml_lines.tokenize@301-363@.\networkx\readwrite\gml.py
       48     16    268      1      66 parse_gml_lines.parse_kv@375-440@.\networkx\readwrite\gml.py
       58     24    457      3     224 parse_gml_lines@298-521@.\networkx\readwrite\gml.py
       70     23    378      1      70 literal_stringizer.stringize@549-618@.\networkx\readwrite\gml.py
       57     29    457      5      64 generate_gml.stringize@710-773@.\networkx\readwrite\gml.py
       54     21    621      2      69 add_graph_element@748-816@.\networkx\readwrite\graphml.py
       49     17    331      3      58 decode_data_elements@960-1017@.\networkx\readwrite\graphml.py
       55     23    390      2      95 generate_multiline_adjlist@39-133@.\networkx\readwrite\multiline_adjlist.py
       63     18    326      6     106 parse_multiline_adjlist@195-300@.\networkx\readwrite\multiline_adjlist.py
       80     25    610      1     109 parse_pajek@167-275@.\networkx\readwrite\pajek.py
      148     42    797      7     347 generate_network_text@73-419@.\networkx\readwrite\text.py
      115     32    652      1     205 _parse_network_text@647-851@.\networkx\readwrite\text.py
      124     31    648     14     243 __new__@330-572@.\networkx\utils\backends.py
      143     48    721      5     186 __call__@630-815@.\networkx\utils\backends.py
      155     53    842      6     205 _convert_arguments@841-1045@.\networkx\utils\backends.py
       99     38    438     12     120 _convert_graph@1047-1166@.\networkx\utils\backends.py
      225     61   1360      6     282 _convert_and_call_for_tests@1195-1476@.\networkx\utils\backends.py
       46     16    258      1      56 _make_doc@1478-1533@.\networkx\utils\backends.py
       25     19    200      3      26 _check_config@227-252@.\networkx\utils\configs.py
       85     25    464      3     150 to_networkx_graph@34-183@.\networkx\convert.py
       52     26    414      3      84 from_dict_of_dicts@373-456@.\networkx\convert.py
       56     20    421      7     175 from_pandas_edgelist@315-489@.\networkx\convert_matrix.py
       64     20    480      8     236 to_numpy_array@791-1026@.\networkx\convert_matrix.py
       68     29    547      6     194 from_numpy_array@1030-1223@.\networkx\convert_matrix.py
       54     25    437      2      61 _relabel_inplace@130-190@.\networkx\relabel.py
=========================================================================================
Total nloc   Avg.NLOC  AvgCCN  Avg.token   Fun Cnt  Warning cnt   Fun Rt   nloc Rt
-----------------------------------------------------------------------------------------
    115525      14.1     2.8      107.5     7022          148      0.02     0.11
PS E:\sep\networkx>
```

Programming language: Python

The coverage tool we utilised for NetworkX is "coverage.py". To run it on our project, we first forked the original branch of NetworkX from github into our own repository, then after making sure every dependency of the project was met on our device, we installed the coverage tool following the instructions in the README of the github page. Afterwards, we ran the test on the project. The results can be seen in the screenshot below:



Coverage report: 93%

coverage.py v7.5.3, created at 2024-06-14 15:41 +0200

File ▲	statements	missing	excluded	branches	partial	coverage
networkx\algorithms\approximation\connectivity.py	119	0	0	74	0	100%
networkx\algorithms\assortativity\connectivity.py	37	0	0	26	0	100%
networkx\algorithms\components\biconnected.py	84	0	0	62	0	100%
networkx\algorithms\components\connected.py	43	0	0	44	0	100%
networkx\algorithms\components\semiconnected.py	12	0	0	12	0	100%
networkx\algorithms\components\strongly_connected.py	90	0	0	82	0	100%
networkx\algorithms\components\weakly_connected.py	45	0	0	40	0	100%
networkx\algorithms\connectivity__init__.py	9	0	0	0	0	100%
networkx\algorithms\connectivity\connectivity.py	147	1	0	96	1	99%
networkx\algorithms\connectivity\cuts.py	115	0	0	77	1	99%
networkx\algorithms\connectivity\disjoint_paths.py	83	0	0	54	0	100%
networkx\algorithms\connectivity\edge_augmentation.py	317	18	0	251	8	94%
networkx\algorithms\connectivity\edge_kcomponents.py	143	0	0	104	0	100%
networkx\algorithms\connectivity\kcomponents.py	88	3	0	66	7	94%
networkx\algorithms\connectivity\kcutsets.py	88	2	0	61	2	97%
networkx\algorithms\connectivity\stoerwagner.py	51	0	0	36	0	100%
networkx\algorithms\connectivity\utils.py	32	0	0	14	0	100%
networkx\linalg\algebraicconnectivity.py	203	0	0	92	0	100%
Total	**1706**	**24**	**0**	**1191**	**19**	**98%**

Additional Coverage Tool

Name Teodor:

(NOTE: both of the branch coverage checks were done at the same time, which is why the link showing the github commit is the same and the screenshot is the same. Additionally, I did work on 3 functions, but managed to boost only 2 to above 80% coverage. I decide to include the one i did not fully boost, because the function is pretty big and the improvement is still not bad, from 0 to 35%)

Find_optimum (NOTE: the function has over 50 flags, so i will only post a small example of the instrumentation):

https://github.com/networkx/networkx/commit/efc292f9b11706344c786128246cd1fef9a62e5d

8 coverage report branchings.txt

```
@@ -0,0 +1,8 @@
+ -=-=-=-coverage report for: branchings.txt-=-=-=-
+ total branch coverage for find_optimum: 0.0%
+
+ 1: 1
+ total branch coverage for _init: 9.090909090909092%
+
+ -=-=-=--=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
+
```

```
@@ -461,6 +479,7 @@ def find_optimum(
461 479             The branching.
462 480
463 481         """
482 +           findOptimumCoverer = branchCoverer.branch_function(branchCoverer, "find_optimum", 59)
464 483         self._init(attr, default, kind, style, preserve_attrs, seed, partition)
465 484         uf = self.uf
466 485
@@ -483,36 +502,45 @@ def desired_edge(v):
483 502             edge = None
484 503             weight = -INF
485 504             for u, _, key, data in G.in_edges(v, data=True, keys=True):
505 +               findOptimumCoverer.addFlag("1")
486 506                 # Skip excluded edges
487 507                 if data.get(partition) == nx.EdgePartition.EXCLUDED:
508 +                   findOptimumCoverer.addFlag("3")
488 509                     continue
489 510                 new_weight = data[attr]
490 511                 # Return the included edge
491 512                 if data.get(partition) == nx.EdgePartition.INCLUDED:
513 +                   findOptimumCoverer.addFlag("4")
492 514                     weight = new_weight
493 515                     edge = (u, v, key, new_weight, data)
494 516                     return edge, weight
495 517                 # Find the best open edge
496 518                 if new_weight > weight:
519 +                   findOptimumCoverer.addFlag("5")
497 520                     weight = new_weight
498 521                     edge = (u, v, key, new_weight, data)
```

```
522 +               findOptimumCoverer.addFlag("2")
500 523             return edge, weight
501 524
502 525         while True:
526 +               findOptimumCoverer.addFlag("6")
503 527             # (I1): Choose a node v in G^i not in D^i.
504 528             try:
529 +                   findOptimumCoverer.addFlag("8")
505 530                 v = next(nodes)
506 531             except StopIteration:
532 +                   findOptimumCoverer.addFlag("9")
507 533                 # If there are no more new nodes to consider, then we *should*
508 534                 # meet the break condition (b) from the paper:
509 535                 #   (b) every node of G^i is in D^i and E^i is a branching
510 536                 # Construction guarantees that it's a branching.
511 537                 assert len(G) == len(B)
512 538                 if len(B):
539 +                   findOptimumCoverer.addFlag("10")
513 540                     assert is_branching(B)
514 541
515 542                 if self.store:
543 +                   findOptimumCoverer.addFlag("11")
516 544                     self.graphs.append(G.copy())
517 545                     self.branchings.append(B.copy())
518 546
@@ -523,10 +551,12 @@ def desired_edge(v):
523 551                     self.minedge_circuit.append(None)
524 552                 break
525 553             else:
554 +                   findOptimumCoverer.addFlag("12")
526 555                 if v in D:
556 +                   findOptimumCoverer.addFlag("13")
527 557                     # print("v in D", v)
528 558                     continue
```

```diff
+            findOptimumCoverer.addFlag("7")
             # Put v into bucket D^i.
             # print(f"Adding node {v}")
             D.add(v)
@@ -537,21 +567,25 @@ def desired_edge(v):
             edge, weight = desired_edge(v)
             # print(f"Max edge is {edge!r}")
             if edge is None:
+                findOptimumCoverer.addFlag("14")
                 # If there is no edge, continue with a new node at (I1).
                 continue
             else:
+                findOptimumCoverer.addFlag("15")
                 # Determine if adding the edge to E^i would mean its no longer
                 # a branching. Presently, v has indegree 0 in B---it is a root.
                 u = edge[0]

                 if uf[u] == uf[v]:
+                    findOptimumCoverer.addFlag("16")
                     # Then adding the edge will create a circuit. Then B
                     # contains a unique path P from v to u. So condition (a)
                     # from the paper does hold. We need to store the circuit
                     # for future reference.
                     Q_nodes, Q_edges = get_path(B, v, u)
                     Q_edges.append(edge[2])  # Edge key
                 else:
+                    findOptimumCoverer.addFlag("17")
                     # Then B with the edge is still a branching and condition
                     # (a) from the paper does not hold.
                     Q_nodes, Q_edges = None, None
@@ -562,19 +596,24 @@ def desired_edge(v):
             # If weight < 0, then it cannot help in finding a maximum branching.
             # This is the root of the problem with minimum branching.
             if self.style == "branching" and weight <= 0:
+                findOptimumCoverer.addFlag("18")
                 acceptable = False
             else:
+                findOptimumCoverer.addFlag("19")
                 acceptable = True

             # print(f"Edge is acceptable: {acceptable}")
             if acceptable:
+                findOptimumCoverer.addFlag("20")
                 dd = {attr: weight}
                 if edge[4].get(partition) is not None:
                     dd[partition] = edge[4].get(partition)
                 B.add_edge(u, v, edge[2], **dd)
                 G[u][v][edge[2]][self.candidate_attr] = True
                 uf.union(u, v)
                 if Q_edges is not None:
+                    findOptimumCoverer.addFlag("22")
                     # print("Edge introduced a simple cycle:")
                     # print(Q_nodes, Q_edges)

@@ -588,21 +627,25 @@ def desired_edge(v):
                 minedge = None
                 Q_incoming_weight = {}
                 for edge_key in Q_edges:
+                    findOptimumCoverer.addFlag("23")
                     u, v, data = B.edge_index[edge_key]
                     # We cannot remove an included edges, even if it is
                     # the minimum edge in the circuit
                     w = data[attr]
                     Q_incoming_weight[v] = w
                     if data.get(partition) == nx.EdgePartition.INCLUDED:
```

Remove_node (NOTE: this function was not tested in the first place, which means that its initial coverage is 0 and not shown on the file):

https://github.com/networkx/networkx/commit/efc292f9b11706344c786128246cd1fef9a62e5d

8 coverage report branchings.txt

```diff
@@ -0,0 +1,8 @@
+ -=-=-=-coverage report for: branchings.txt-=-=-=-
+ total branch coverage for find_optimum: 0.0%
+
+ 1: 1
+ total branch coverage for _init: 9.090909090909092%
+
+ -=-=-=--=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
+
```

```
217  217
218  218        def remove_node(self, n):
     219  +         removeNodeCoverage = branchCoverer.branch_function(branchCoverer, "remove_node", 6)
219  220            keys = set()
220  221            for keydict in self.pred[n].values():
     222  +             removeNodeCoverage.addFlag("1")
221  223                keys.update(keydict)
     224  +         removeNodeCoverage.addFlag("2")
222  225            for keydict in self.succ[n].values():
     226  +             removeNodeCoverage.addFlag("3")
223  227                keys.update(keydict)
     228  +         removeNodeCoverage.addFlag("4")
224  229
225  230            for key in keys:
     231  +             removeNodeCoverage.addFlag("5")
226  232                del self.edge_index[key]
     233  +         removeNodeCoverage.addFlag("6")
227  234
```

_init:

https://github.com/networkx/networkx/commit/efc292f9b11706344c786128246cd1fef9a62e5d

```
346  355            # Store inputs.
@@ -351,12 +360,15 @@ def _init(self, attr, default, kind, style, preserve_attrs, seed, partition):
351  360
352  361            # Determine how we are going to transform the weights.
353  362            if kind == "min":
     363  +             initCoverage.addFlag("2")
354  364                self.trans = trans = _min_weight
355  365            else:
     366  +             initCoverage.addFlag("3")
356  367                self.trans = trans = _max_weight
357  368
358  369            if attr is None:
359  370                # Generate a random attr the graph probably won't have.
     371  +             initCoverage.addFlag("4")
360  372                attr = random_string(seed=seed)
361  373
362  374            # This is the actual attribute used by the algorithm.
@@ -371,18 +383,24 @@ def _init(self, attr, default, kind, style, preserve_attrs, seed, partition):
371  383            self.G = G = MultiDiGraph_EdgeKey()
372  384            self.G.__networkx_cache__ = None  # Disable caching
373  385            for key, (u, v, data) in enumerate(self.G_original.edges(data=True)):
372  384                self.G.__networkx_cache__ = None  # Disable caching
373  385                for key, (u, v, data) in enumerate(self.G_original.edges(data=True)):
     386  +                 initCoverage.addFlag("5")
374  387                    d = {attr: trans(data.get(attr, default))}
375  388
376  389                    if data.get(partition) is not None:
     390  +                     initCoverage.addFlag("7")
377  391                        d[partition] = data.get(partition)
378  392
379  393                    if preserve_attrs:
     394  +                     initCoverage.addFlag("8")
380  395                        for d_k, d_v in data.items():
     396  +                         initCoverage.addFlag("9")
381  397                            if d_k != attr:
     398  +                             initCoverage.addFlag("11")
382  399                                d[d_k] = d_v
     400  +                 initCoverage.addFlag("10")
383  401
384  402                    G.add_edge(u, v, key, **d)
385      -
     403  +         initCoverage.addFlag("6")
386  404                self.level = 0
387  405
```

Coverage improvement

Individual tests (NOTE: The link is again the same, as they were pushed at the same time)

Test_remove_node:

https://github.com/networkx/networkx/commit/0f871ccd6e815c95ab1426576270d3d4454144a6

8 coverage report branchings.txt

```
@@ -0,0 +1,8 @@
+ -=-=-=-coverage report for: branchings.txt-=-=-=-
+ total branch coverage for find_optimum: 0.0%
+
+ 1: 1
+ total branch coverage for _init: 9.090909090909092%
+
+ -=-=-=--=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
+
```

```
+
+ 1: 1
+ 2: 1
+ 3: 1
+ 4: 1
+ 5: 2
+ 6: 1
+ total branch coverage for remove_node: 100.0%
```

This is the improvement on the remove_node branch coverage. It has increased to a full coverage of a hundred percent, as the function was not being tested in the first place

Test_edmonds_init:

https://github.com/networkx/networkx/commit/0f871ccd6e815c95ab1426576270d3d4454144a6

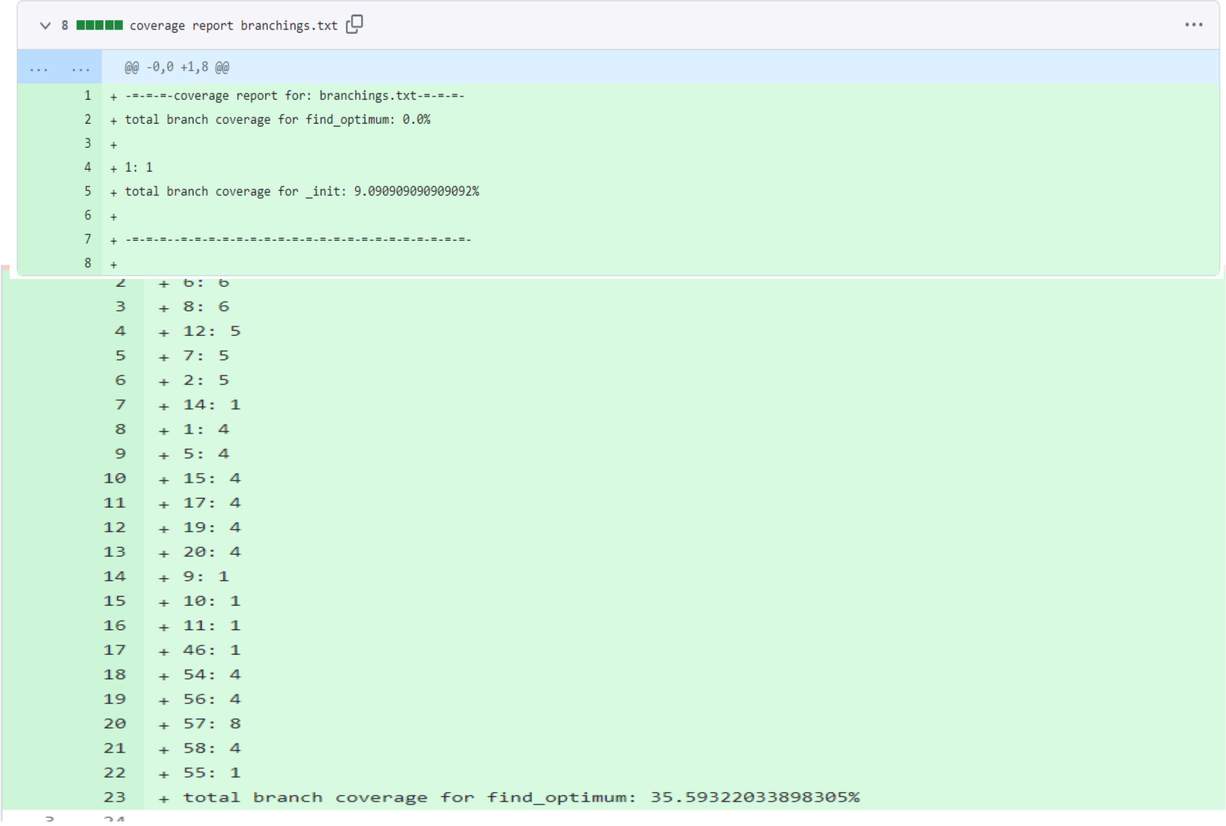
8 coverage report branchings.txt

```
@@ -0,0 +1,8 @@
+ -=-=-=-coverage report for: branchings.txt-=-=-=-
+ total branch coverage for find_optimum: 0.0%
+
+ 1: 1
+ total branch coverage for _init: 9.090909090909092%
+
+ -=-=-=--=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
+
```

```
+ 2: 1
+ 5: 8
+ 7: 4
+ 8: 8
+ 9: 12
+ 11: 4
+ 10: 8
+ 6: 2
+ 3: 1
+ total branch coverage for _init: 90.9090909090909%
```

The test, again, almost fully upgraded the coverage to a hundred, because it was mostly not being run by any test.

Test_find_optimum_max_absorescence: https://github.com/networkx/networkx/commit/0f871ccd6e815c95ab1426576270d3d4454144a6



```
8 coverage report branchings.txt
@@ -0,0 +1,8 @@
+ -=-=-=-coverage report for: branchings.txt-=-=-=-
+ total branch coverage for find_optimum: 0.0%
+
+ 1: 1
+ total branch coverage for _init: 9.090909090909092%
+
+ -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
+
```

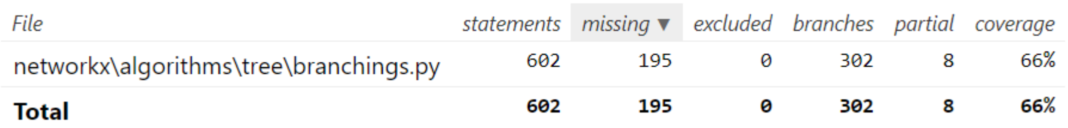
```
+ 6: 6
+ 8: 6
+ 12: 5
+ 7: 5
+ 2: 5
+ 14: 1
+ 1: 4
+ 5: 4
+ 15: 4
+ 17: 4
+ 19: 4
+ 20: 4
+ 9: 1
+ 10: 1
+ 11: 1
+ 46: 1
+ 54: 4
+ 56: 4
+ 57: 8
+ 58: 4
+ 55: 1
+ total branch coverage for find_optimum: 35.59322033898305%
```

For this function ,sadly, I could not manage to raise it above 80%, because there are a lot of if statements which are always true and I could not manage to simulate a scenario, where it checks the else statement.

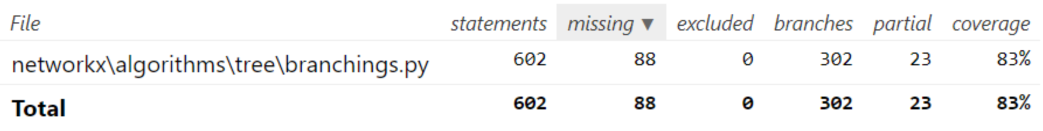
###Overview

Here is the overall difference:

The tests boosted the file by around 20%, which itself boosted the whole project by a few percent as well. Here is the statement/line coverage



File	statements	missing ▼	excluded	branches	partial	coverage
networkx\algorithms\tree\branchings.py	602	195	0	302	8	66%
Total	**602**	**195**	**0**	**302**	**8**	**66%**



File	statements	missing ▼	excluded	branches	partial	coverage
networkx\algorithms\tree\branchings.py	602	88	0	302	23	83%
Total	**602**	**88**	**0**	**302**	**23**	**83%**

Additional Coverage Tool

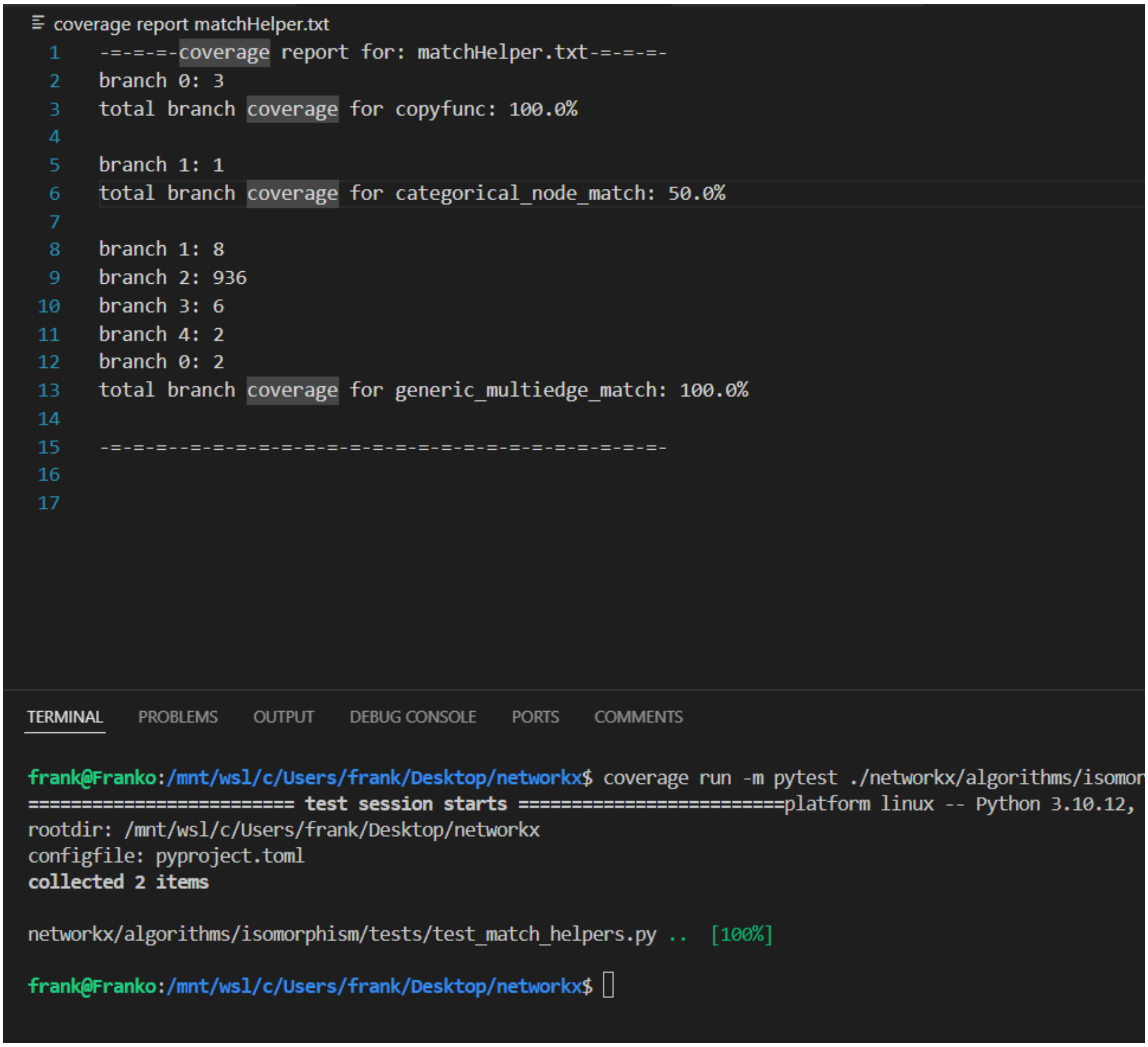
Name Franko:

Function 1 Name: numerical_multiedge_match

https://github.com/TeodorAL2003/networkx/commit/f4a2705378b185bd307238cb9139f995300369fd (there is only one link for the commit for both of my functions - I worked mostly locally and then uploaded everything once I reached the desired goals)

networkx/algorithms/isomorphism/matchhelpers.py: 70% 87 31 0 2

```
181
182
183 def numerical_multiedge_match(attr, default, rtol=1.0000000000000001e-05, atol=1e-08):
184     if isinstance(attr, str):
185
186         def match(datasets1, datasets2):
187             values1 = sorted(data.get(attr, default) for data in datasets1.values())
188             values2 = sorted(data.get(attr, default) for data in datasets2.values())
189             return allclose(values1, values2, rtol=rtol, atol=atol)
190
191     else:
192         attrs = list(zip(attr, default))  # Python 3
193
194         def match(datasets1, datasets2):
195             values1 = []
196             for data1 in datasets1.values():
197                 x = tuple(data1.get(attr, d) for attr, d in attrs)
198                 values1.append(x)
199             values2 = []
200             for data2 in datasets2.values():
201                 x = tuple(data2.get(attr, d) for attr, d in attrs)
202                 values2.append(x)
203             values1.sort()
204             values2.sort()
205             for xi, yi in zip(values1, values2):
206                 if not allclose(xi, yi, rtol=rtol, atol=atol):
207                     return False
208             else:
209                 return True
210
211     return match
212
```

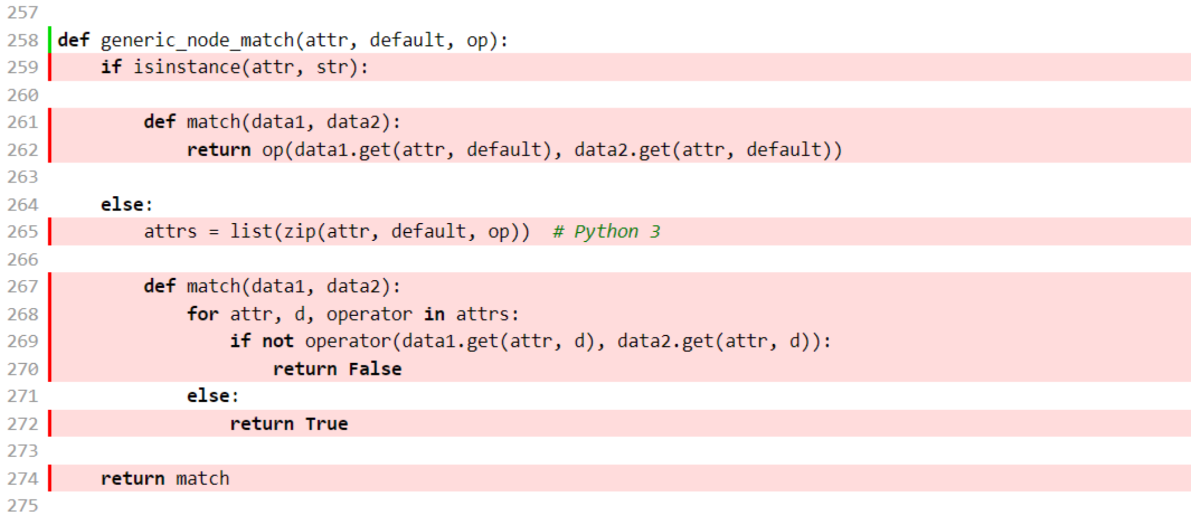


```
-=-=-=-coverage report for: matchHelper.txt-=-=-=-
branch 0: 3
total branch coverage for copyfunc: 100.0%

branch 1: 1
total branch coverage for categorical_node_match: 50.0%

branch 1: 8
branch 2: 936
branch 3: 6
branch 4: 2
branch 0: 2
total branch coverage for generic_multiedge_match: 100.0%

-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
```

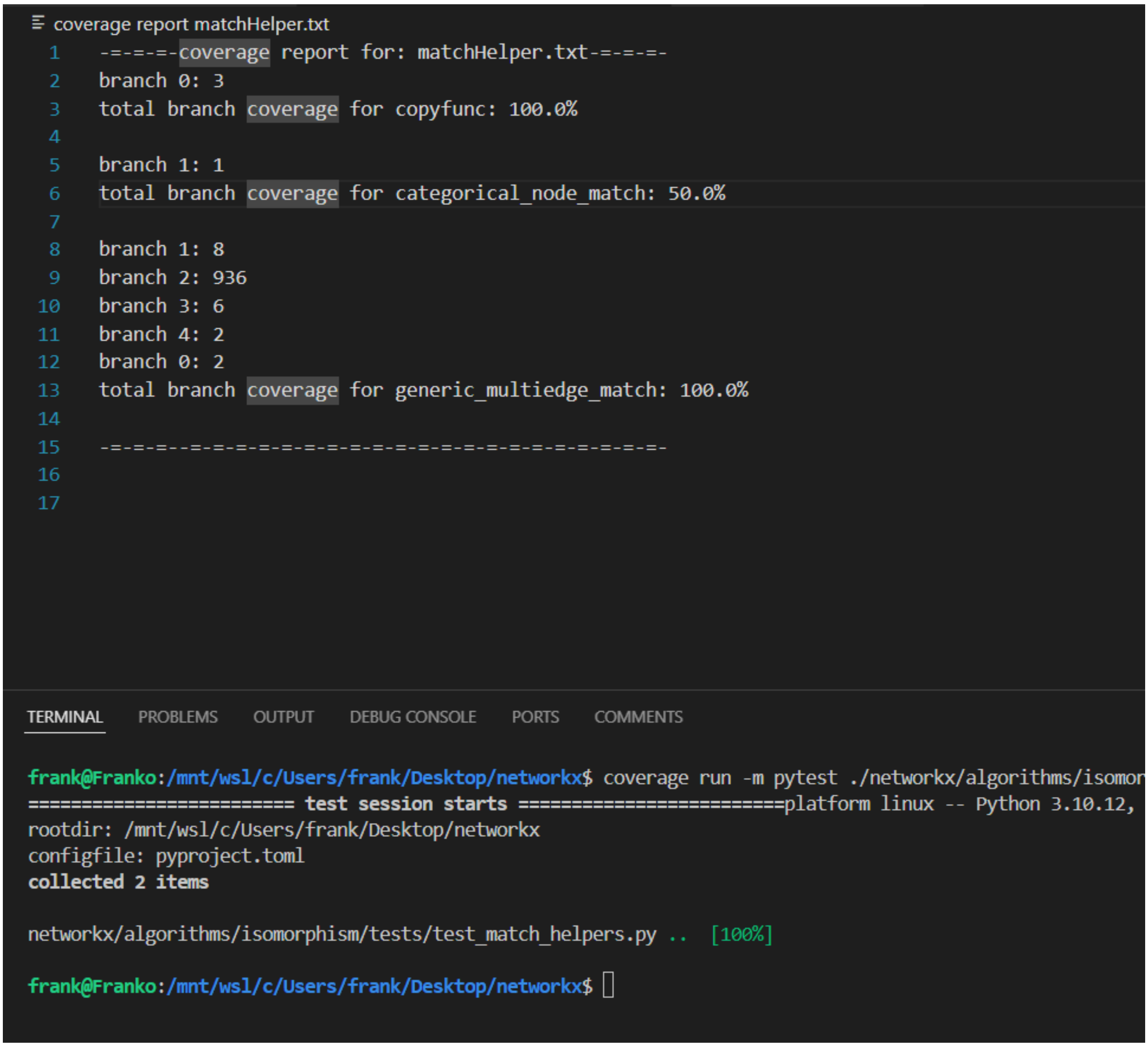
```
frank@Franko:/mnt/wsl/c/Users/frank/Desktop/networkx$ coverage run -m pytest ./networkx/algorithms/isomor
========================= test session starts ========================platform linux -- Python 3.10.12,
rootdir: /mnt/wsl/c/Users/frank/Desktop/networkx
configfile: pyproject.toml
collected 2 items

networkx/algorithms/isomorphism/tests/test_match_helpers.py ..  [100%]

frank@Franko:/mnt/wsl/c/Users/frank/Desktop/networkx$ 
```

After running our coverage tool, it is possible to see that the function has 0% branch coverage.

Function 2 Name: generic_node_match



```
def generic_node_match(attr, default, op):
    if isinstance(attr, str):

        def match(data1, data2):
            return op(data1.get(attr, default), data2.get(attr, default))

    else:
        attrs = list(zip(attr, default, op))  # Python 3

        def match(data1, data2):
            for attr, d, operator in attrs:
                if not operator(data1.get(attr, d), data2.get(attr, d)):
                    return False
            else:
                return True

    return match
```



coverage report matchHelper.txt

```
-=-=-=-coverage report for: matchHelper.txt-=-=-=-
branch 0: 3
total branch coverage for copyfunc: 100.0%

branch 1: 1
total branch coverage for categorical_node_match: 50.0%

branch 1: 8
branch 2: 936
branch 3: 6
branch 4: 2
branch 0: 2
total branch coverage for generic_multiedge_match: 100.0%

-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
```

```
frank@Franko:/mnt/wsl/c/Users/frank/Desktop/networkx$ coverage run -m pytest ./networkx/algorithms/isomor
======================== test session starts ========================platform linux -- Python 3.10.12,
rootdir: /mnt/wsl/c/Users/frank/Desktop/networkx
configfile: pyproject.toml
collected 2 items

networkx/algorithms/isomorphism/tests/test_match_helpers.py ..  [100%]

frank@Franko:/mnt/wsl/c/Users/frank/Desktop/networkx$ 
```

Also for this second function it is possible to see through our coverage tool that the branch coverage is 0%.

Coverage improvement

Individual tests

Test 1:

networkx/algorithms/isomorphism/matchhelpers.py: 70% 87 31 0 2

```
181
182
183 def numerical_multiedge_match(attr, default, rtol=1.0000000000000001e-05, atol=1e-08):
184     if isinstance(attr, str):
185
186         def match(datasets1, datasets2):
187             values1 = sorted(data.get(attr, default) for data in datasets1.values())
188             values2 = sorted(data.get(attr, default) for data in datasets2.values())
189             return allclose(values1, values2, rtol=rtol, atol=atol)
190
191     else:
192         attrs = list(zip(attr, default))  # Python 3
193
194         def match(datasets1, datasets2):
195             values1 = []
196             for data1 in datasets1.values():
197                 x = tuple(data1.get(attr, d) for attr, d in attrs)
198                 values1.append(x)
199             values2 = []
200             for data2 in datasets2.values():
201                 x = tuple(data2.get(attr, d) for attr, d in attrs)
202                 values2.append(x)
203             values1.sort()
204             values2.sort()
205             for xi, yi in zip(values1, values2):
206                 if not allclose(xi, yi, rtol=rtol, atol=atol):
207                     return False
208             else:
209                 return True
210
211     return match
212
```

networkx/algorithms/isomorphism/matchhelpers.py: 91% 108 10 0 0

```python

def numerical_multiedge_match(attr, default, rtol=1.0000000000000001e-05, atol=1e-08):
    if isinstance(attr, str):

        def match(datasets1, datasets2):
            values1 = sorted(data.get(attr, default) for data in datasets1.values())
            values2 = sorted(data.get(attr, default) for data in datasets2.values())
            return allclose(values1, values2, rtol=rtol, atol=atol)

    else:
        attrs = list(zip(attr, default))  # Python 3

        def match(datasets1, datasets2):
            values1 = []
            for data1 in datasets1.values():
                x = tuple(data1.get(attr, d) for attr, d in attrs)
                values1.append(x)
            values2 = []
            for data2 in datasets2.values():
                x = tuple(data2.get(attr, d) for attr, d in attrs)
                values2.append(x)
            values1.sort()
            values2.sort()
            for xi, yi in zip(values1, values2):
                if not allclose(xi, yi, rtol=rtol, atol=atol):
                    return False
            else:
                return True

    return match
```

As shown in these two screenshots, it is possible to see that the coverage of the first function has increased, with my tests I was able to cover the else branch of this function which didn't have any coverage. I was able to increase the coverage by 21%.

```
-=-=-=-coverage report for: matchHelper.txt-=-=-=-
branch 0: 3
total branch coverage for copyfunc: 100.0%

branch 1: 1
total branch coverage for categorical_node_match: 50.0%

branch 0: 1
branch 1: 1
branch 3: 1
branch 2: 2
total branch coverage for numerical_multiedge_match: 100.0%

branch 0: 8
total branch coverage for allclose: 100.0%

branch 1: 8
branch 2: 936
branch 3: 6
branch 4: 2
branch 0: 2
total branch coverage for generic_multiedge_match: 100.0%
```

```
MINAL    PROBLEMS    OUTPUT    DEBUG CONSOLE    PORTS    COMMENTS

=====================platform linux -- Python 3.10.12, pytest-8.2.2, pluggy-1.5.0
tdir: /mnt/wsl/c/Users/frank/Desktop/networkx
figfile: pyproject.toml
lected 3 items

workx/algorithms/isomorphism/tests/test_match_helpers.py ...                    [100%]

================================= 3 passed in 3.10s =====================================
nk@Franko:/mnt/wsl/c/Users/frank/Desktop/networkx$
```

Also after running again our tool, it is possible to see that the number of tests increased from 2 to 3 and also the function of interest has reach 100% branch coverage

Test 2:

networkx/algorithms/isomorphism/matchhelpers.py: 91% 108 10 0 0

```
>>> nm = generic_node_match(["weight", "color"], [1.0, "red"], [isclose, eq])

"""


def generic_node_match(attr, default, op):
    if isinstance(attr, str):

        def match(data1, data2):
            return op(data1.get(attr, default), data2.get(attr, default))

    else:
        attrs = list(zip(attr, default, op))  # Python 3

        def match(data1, data2):
            for attr, d, operator in attrs:
                if not operator(data1.get(attr, d), data2.get(attr, d)):
                    return False
            else:
                return True

    return match
```

networkx/algorithms/isomorphism/matchhelpers.py: 100% 118 0 0 0

```

"""


def generic_node_match(attr, default, op):
    if isinstance(attr, str):

        def match(data1, data2):
            return op(data1.get(attr, default), data2.get(attr, default))

    else:
        attrs = list(zip(attr, default, op))  # Python 3

        def match(data1, data2):
            for attr, d, operator in attrs:
                if not operator(data1.get(attr, d), data2.get(attr, d)):
                    return False
            else:
                return True

    return match
```

With this last function, I was able to reach a 100% coverage of the file - adding an additional 9% to the previous statement.

```
-=-=-=-coverage report for: matchHelper.txt-=-=-=-
branch 0: 3
total branch coverage for copyfunc: 100.0%

branch 1: 1
total branch coverage for categorical_node_match: 50.0%

branch 0: 1
branch 1: 1
branch 3: 2
branch 2: 4
total branch coverage for generic_node_match: 100.0%

branch 0: 1
branch 1: 1
branch 3: 1
branch 2: 2
total branch coverage for numerical_multiedge_match: 100.0%

branch 0: 8
total branch coverage for allclose: 100.0%

branch 1: 8
```

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE PORTS COMMENTS

```
frank@Franko:/mnt/wsl/c/Users/frank/Desktop/networkx$ coverage run -m pytest ./networkx/algorithms/isomorphism/tests/test_match_helpers.p
========================================================================== test session starts ==========================================
platform linux -- Python 3.10.12, pytest-8.2.2, pluggy-1.5.0
rootdir: /mnt/wsl/c/Users/frank/Desktop/networkx
configfile: pyproject.toml
collected 4 items

networkx/algorithms/isomorphism/tests/test_match_helpers.py ....

========================================================================== 4 passed in 3.15s ============================================
frank@Franko:/mnt/wsl/c/Users/frank/Desktop/networkx$
```

After running our coverage tool, it is possible to see that the branch coverage of this function went from 0% to 100%

###Overview

File ▲	statements	missing	excluded	branches	partial	coverage
networkx/algorithms/isomorphism/matchhelpers.py	118	31	0	67	2	70%
Total	**118**	**31**	**0**	**67**	**2**	**70%**

File ▲	statements	missing	excluded	branches	partial	coverage
networkx/algorithms/isomorphism/matchhelpers.py	118	0	0	67	0	100%
Total	**118**	**0**	**0**	**67**	**0**	**100%**

The overall coverage has been significantly increased, reaching the desired goals.

Additional Coverage Tool

Name Jimmy:

I covered the functions in the file “maxflow.py”. These included the functions

- maximum_flow()
- minimum_cut()
- maximum_flow_value()
- minimum_cut_value()

All changes to the source code can be viewed here:
https://github.com/TeodorAL2003/networkx/commit/216d817dbaa0f956e584f5ba0fcfb70db99d0678

This is the branch coverage for this file before the improvements, not that the function maximum_flow() is not tested at all.

maxflow.py | test_maxflow.py M | coverage report maxflow.txt M

networkx > coverage report maxflow.txt

```
-=-=-=-coverage report for: maxflow.txt-=-=-=-
branch 5: 65
branch 6: 65
branch 4: 1
total branch coverage for minimum_cut: 50.0%

branch 1: 1
branch 4: 13
total branch coverage for maximum_flow_value: 50.0%

branch 4: 1
total branch coverage for minimum_cut_value: 20.0%
-=-=-=--=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
```

Coverage improvement

I improved the coverage for the functions maximum_flow() and maximum_flow_value():

maxflow.py | test_maxflow.py M | coverage report maxflow.txt M

networkx > coverage report maxflow.txt

```
-=-=-=-coverage report for: maxflow.txt-=-=-=-
branch 0: 3
branch 1: 2
branch 4: 1
branch 2: 1
branch 3: 1
total branch coverage for maximum_flow: 100.0%

branch 1: 2
branch 2: 1
branch 3: 1
branch 4: 13
total branch coverage for maximum_flow_value: 100.0%

branch 5: 65
branch 6: 65
branch 4: 1
total branch coverage for minimum_cut: 50.0%

branch 4: 1
total branch coverage for minimum_cut_value: 20.0%
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
```

Individual tests

I improved the branch coverage by adding some tests:

```
def test_max_flow_directly(self):
        G = nx.Graph()
        G.add_edge('a', 'b', capacity=2)

        flowVal, flowDict = nx.maximum_flow(G, 'a', 'b')
```

```
        assert(flowVal == 2)
        assert(flowDict['a']['b'] == 2)

    def test_max_flow_exceptions(self):
        G = nx.Graph()
        G.add_edge('a', 'b', capacity=2)

        try:
            flowVal, flowDict = nx.maximum_flow(G, 'a', 'b',
kwargs=["someKwarg"])
        except nx.NetworkXError as e:
            assert(e.args[0] == "You have to explicitly set a flow_func
if"
                " you need to pass parameters via kwargs.")

        try:
            uncallableParam = 0
            flowVal, flowDict = nx.maximum_flow(G, 'a', 'b',
flow_func=uncallableParam)
        except nx.NetworkXError as e:
            assert(e.args[0] == "flow_func has to be callable.")

    def test_max_flow_value_exceptions(self):
        G = nx.Graph()
        G.add_edge('a', 'b', capacity=2)

        try:
            flowVal, flowDict = nx.maximum_flow_value(G, 'a', 'b',
kwargs=["someKwarg"])
        except nx.NetworkXError as e:
            assert(e.args[0] == "You have to explicitly set a flow_func
if"
                " you need to pass parameters via kwargs.")

        try:
            uncallableParam = 0
            flowVal, flowDict = nx.maximum_flow_value(G, 'a', 'b',
flow_func=uncallableParam)
        except nx.NetworkXError as e:
            assert(e.args[0] == "flow_func has to be callable.")
```

###Overview

Coverage report: 15%

Files | Functions | Classes

coverage.py v7.5.3, created at 2024-06-27 14:24 +0200

File ▲	statements	missing	excluded	branches	partial	coverage
networkx\algorithms\flow\maxflow.py	86	36	0	32	8	54%
Total	**86**	**36**	**0**	**32**	**8**	**54%**

coverage.py v7.5.3, created at 2024-06-27 14:24 +0200

Coverage report: 15%

Files | Functions | Classes

coverage.py v7.5.3, created at 2024-06-27 14:25 +0200

File ▲	statements	missing	excluded	branches	partial	coverage
networkx\algorithms\flow\maxflow.py	86	17	0	32	6	77%
Total	**86**	**17**	**0**	**32**	**6**	**77%**

coverage.py v7.5.3, created at 2024-06-27 14:25 +0200

I was able to improve the coverage of the whole file from 54% to 77%

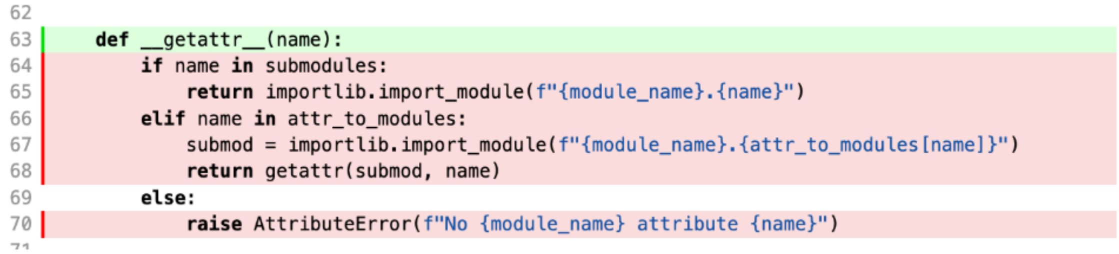
Additional Coverage Tool

Name Alessandro:

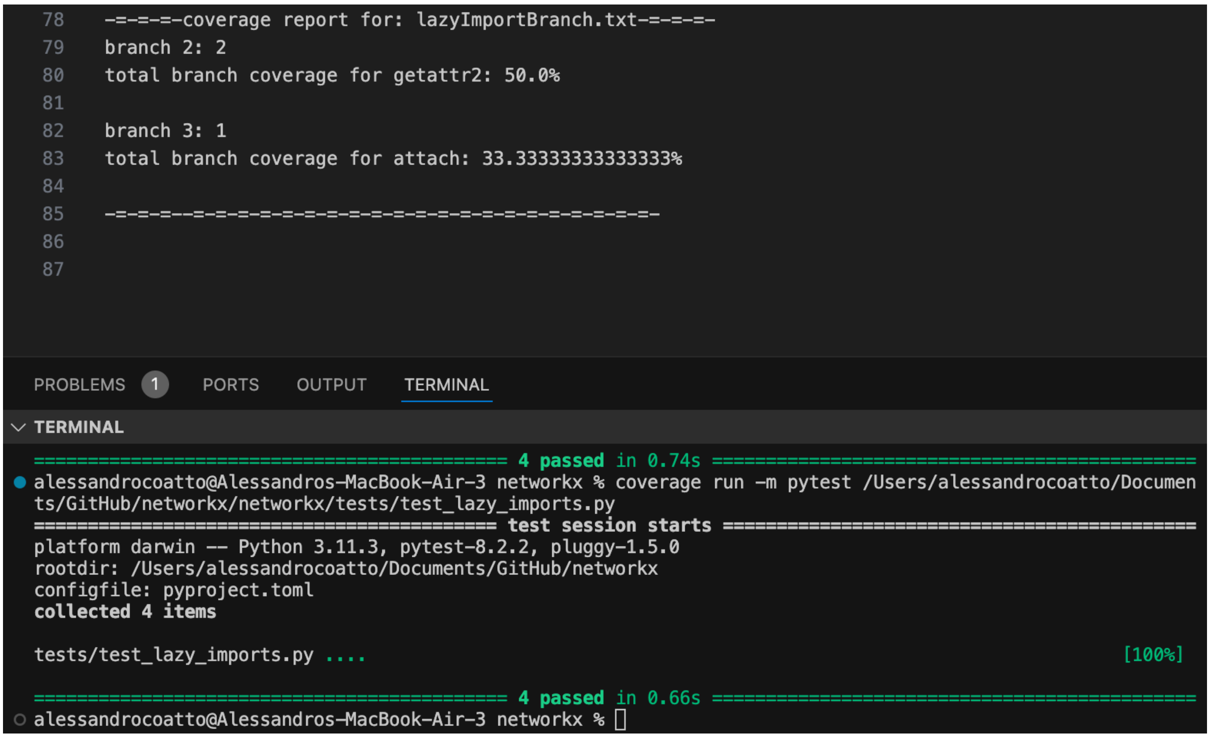
All of the changes can be seen here:

https://github.com/TeodorAL2003/networkx/tree/Alessandro

__getattr__:



```
    def __getattr__(name):
        if name in submodules:
            return importlib.import_module(f"{module_name}.{name}")
        elif name in attr_to_modules:
            submod = importlib.import_module(f"{module_name}.{attr_to_modules[name]}")
            return getattr(submod, name)
        else:
            raise AttributeError(f"No {module_name} attribute {name}")
```



```
-=-=-=-coverage report for: lazyImportBranch.txt-=-=-=-
branch 2: 2
total branch coverage for getattr2: 50.0%

branch 3: 1
total branch coverage for attach: 33.33333333333333%

-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
```

```
============================================ 4 passed in 0.74s ============================================
alessandrocoatto@Alessandros-MacBook-Air-3 networkx % coverage run -m pytest /Users/alessandrocoatto/Documen
ts/GitHub/networkx/networkx/tests/test_lazy_imports.py
=========================================== test session starts ===========================================
platform darwin -- Python 3.11.3, pytest-8.2.2, pluggy-1.5.0
rootdir: /Users/alessandrocoatto/Documents/GitHub/networkx
configfile: pyproject.toml
collected 4 items

tests/test_lazy_imports.py ....                                                                      [100%]

============================================ 4 passed in 0.66s ============================================
alessandrocoatto@Alessandros-MacBook-Air-3 networkx % 
```

Currently, the coverage for the function __getattr__ is not even covered.

Lazy_imports:

networkx/lazy_imports.py: 61% 37 17 0 5

```
143     of the library.
144
145     Parameters
146     ----------
147     fullname : str
148         The full name of the package or subpackage to import.  For example::
149
150           sp = lazy.load("scipy")  # import scipy as sp
151           spla = lazy.load("scipy.linalg")  # import scipy.linalg as spla
152
153     Returns
154     -------
155     pm : importlib.util._LazyModule
156         Proxy module. Can be used like any regularly imported module.
157         Actual loading of the module occurs upon first attribute request.
158
159     """
160     try:
161         return sys.modules[fullname]
162     except:
163         pass
164
165     # Not previously loaded -- look it up
166     spec = importlib.util.find_spec(fullname)
167
168     if spec is None:                                                    168 →✓182
169         try:
170             parent = inspect.stack()[1]
171             frame_data = {
172                 "spec": fullname,
173                 "filename": parent.filename,
174                 "lineno": parent.lineno,
175                 "function": parent.function,
176                 "code_context": parent.code_context,
177             }
178             return DelayedImportErrorModule(frame_data, "DelayedImportErrorModule")
179         finally:
180             del parent
181
182     module = importlib.util.module_from_spec(spec)
183     sys.modules[fullname] = module
184
185     loader = importlib.util.LazyLoader(spec.loader)
186     loader.exec_module(module)
187
188     return module
```

« prev ^ index » next coverage.py v7.5.3, created at 2024-06-14 16:32 +0200

```
78    -=-=-=-coverage report for: lazyImportBranch.txt-=-=-=-
79    branch 2: 2
80    total branch coverage for getattr2: 50.0%
81
82    branch 3: 1
83    total branch coverage for attach: 33.33333333333333%
84
85    -=-=-=--=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
86
87
```

PROBLEMS 1 PORTS OUTPUT TERMINAL

TERMINAL

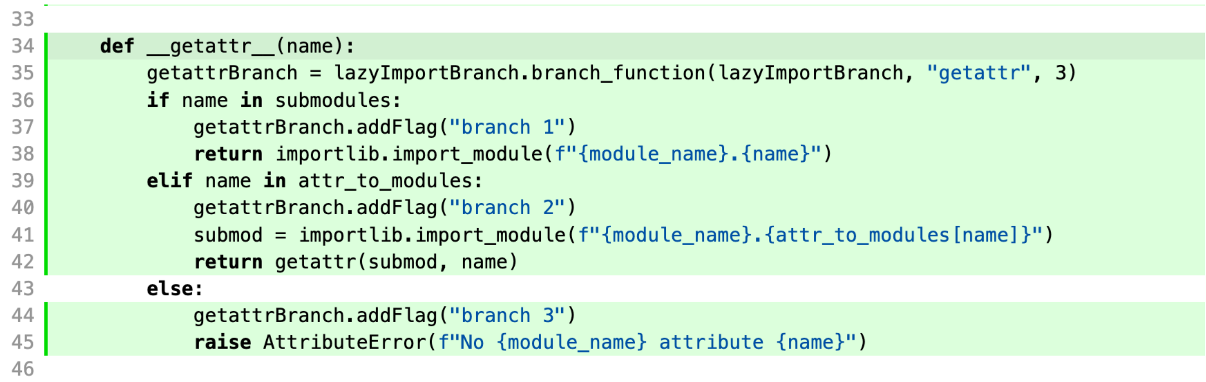
```
========================================== 4 passed in 0.74s ==========================================
alessandrocoatto@Alessandros-MacBook-Air-3 networkx % coverage run -m pytest /Users/alessandrocoatto/Documen
ts/GitHub/networkx/networkx/tests/test_lazy_imports.py
========================================= test session starts =========================================
platform darwin -- Python 3.11.3, pytest-8.2.2, pluggy-1.5.0
rootdir: /Users/alessandrocoatto/Documents/GitHub/networkx
configfile: pyproject.toml
collected 4 items

tests/test_lazy_imports.py ....                                                              [100%]

========================================== 4 passed in 0.66s ==========================================
alessandrocoatto@Alessandros-MacBook-Air-3 networkx %
```

Similarly to __getattr__, also this function does not have any coverage.

Coverage improvement

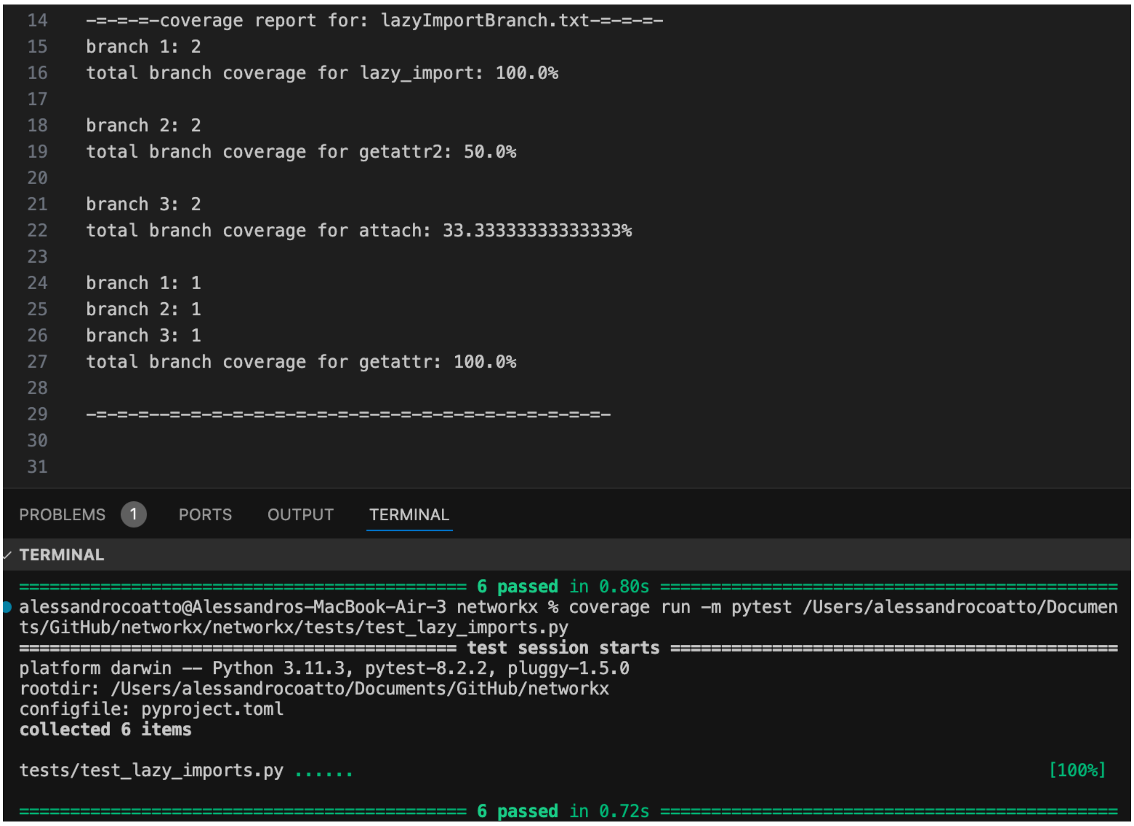
Individual tests

__getattr__:



```
def __getattr__(name):
    getattrBranch = lazyImportBranch.branch_function(lazyImportBranch, "getattr", 3)
    if name in submodules:
        getattrBranch.addFlag("branch 1")
        return importlib.import_module(f"{module_name}.{name}")
    elif name in attr_to_modules:
        getattrBranch.addFlag("branch 2")
        submod = importlib.import_module(f"{module_name}.{attr_to_modules[name]}")
        return getattr(submod, name)
    else:
        getattrBranch.addFlag("branch 3")
        raise AttributeError(f"No {module_name} attribute {name}")
```

Originally, the entire *function* was not completely untested. After the implementation of my tests, the overall coverage for the functions has increased, covering them completely.



```
-=-=-=-coverage report for: lazyImportBranch.txt-=-=-=-
branch 1: 2
total branch coverage for lazy_import: 100.0%

branch 2: 2
total branch coverage for getattr2: 50.0%

branch 3: 2
total branch coverage for attach: 33.33333333333333%

branch 1: 1
branch 2: 1
branch 3: 1
total branch coverage for getattr: 100.0%

-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-


```

```
============================================ 6 passed in 0.80s ============================================
alessandrocoatto@Alessandros-MacBook-Air-3 networkx % coverage run -m pytest /Users/alessandrocoatto/Documen
ts/GitHub/networkx/networkx/tests/test_lazy_imports.py
=========================================== test session starts ===========================================
platform darwin -- Python 3.11.3, pytest-8.2.2, pluggy-1.5.0
rootdir: /Users/alessandrocoatto/Documents/GitHub/networkx
configfile: pyproject.toml
collected 6 items

tests/test_lazy_imports.py ......                                                                    [100%]

============================================ 6 passed in 0.72s ============================================
```

After the tests, the coverage went up by 100%.

lazy_imports:

networkx/lazy_imports.py: 92% 51 3 0 3

```
    Parameters
    ----------
    fullname : str
        The full name of the package or subpackage to import.  For example::

          sp = lazy.load("scipy")  # import scipy as sp
          spla = lazy.load("scipy.linalg")  # import scipy.linalg as spla

    Returns
    -------
    pm : importlib.util._LazyModule
        Proxy module. Can be used like any regularly imported module.
        Actual loading of the module occurs upon first attribute request.

    """
    try:
        return sys.modules[fullname]
    except:
        pass

    # Not previously loaded -- look it up
    spec = importlib.util.find_spec(fullname)

    if spec is None:
        #lazyImportBranch.addFlag("branch 1")
        try:
            parent = inspect.stack()[1]
            frame_data = {
                "spec": fullname,
                "filename": parent.filename,
                "lineno": parent.lineno,
                "function": parent.function,
                "code_context": parent.code_context,
            }
            return DelayedImportErrorModule(frame_data, "DelayedImportErrorModule")
        finally:
            del parent

    module = importlib.util.module_from_spec(spec)
    sys.modules[fullname] = module

    loader = importlib.util.LazyLoader(spec.loader)
    loader.exec_module(module)

    return module
```

« prev ^ index » next coverage.py v7.5.3, created at 2024-06-20 09:29 +0200

```
-=-=-=-coverage report for: lazyImportBranch.txt-=-=-=-
branch 1: 2
total branch coverage for lazy_import: 100.0%

branch 2: 2
total branch coverage for getattr2: 50.0%

branch 3: 2
total branch coverage for attach: 33.33333333333333%

branch 1: 1
branch 2: 1
branch 3: 1
total branch coverage for getattr: 100.0%

-=-=-=--=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
```

PROBLEMS 1 PORTS OUTPUT TERMINAL

TERMINAL

```
========================================== 6 passed in 0.80s ==========================================
alessandrocoatto@Alessandros-MacBook-Air-3 networkx % coverage run -m pytest /Users/alessandrocoatto/Documen
ts/GitHub/networkx/networkx/tests/test_lazy_imports.py
========================================== test session starts ==========================================
platform darwin -- Python 3.11.3, pytest-8.2.2, pluggy-1.5.0
rootdir: /Users/alessandrocoatto/Documents/GitHub/networkx
configfile: pyproject.toml
collected 6 items

tests/test_lazy_imports.py ......                                                                  [100%]

========================================== 6 passed in 0.72s ==========================================
```

After the tests, the coverage went up by 100%.

###Overview

Coverage report: 17%

Files | Functions | Classes

coverage.py v7.5.3, created at 2024-06-27 15:01 +0200

File	*statements*	*missing*	*excluded*	*coverage* ▲
lazy_imports.py	55	17	0	69%
tests/test_lazy_imports.py	66	12	0	82%
Total	**121**	**29**	**0**	**76%**

Coverage report: 17%

Files | Functions | Classes

coverage.py v7.5.3, created at 2024-06-27 14:59 +0200

File	*statements*	*missing*	*excluded*	*coverage* ▲
lazy_imports.py	71	11	0	85%
tests/test_lazy_imports.py	100	13	0	87%
Total	**171**	**24**	**0**	**86%**

Overall, the coverage for lazy_imports went up from 69% to 85%, resulting in a 16% increase in coverage.

Contributions

While Jimmy was working on the script for the additional branch coverage, Alessandro, Teodor and Franko were all doing more research on setting up the project, creating additional tests for functions and installing pytest to get an initial overview of the project. After this, we all picked 2 functions to improve and test based on the pytest we performed on the entire file. It's important to note that several projects have been changed throughout this assignment as we needed one that met all of the requirements. Once we all agreed on what to do and how, we started implementing everything together. We tried to always work in pairs to avoid having a team member stuck on a problem someone else previously solved.