

```

1 # coding=utf-8
2
3 """
4 Unit tests for key handling
5 """
6
7 import os
8 from unittest import TestCase
9
10 from src.crypto import keys
11
12
13 class TestRSAKeyLoading(TestCase):
14     """Tests for RSA Keys"""
15
16     def setUp(self) -> None:
17         """Initialise loaders fresh between tests"""
18         self.loader = keys.RSAKeyLoader()
19         os.chdir("/home/tcassar/projects/settle/src")
20         print(os.system("pwd"))
21         self.key_path = "./crypto/sample_keys/
d_private-key.pem"
22         self.pub_key_path = "./crypto/sample_keys/
d_public-key.pe"
23
24     def test_file_loading(self):
25         """Tests that file is being loaded correctly
assuming correct file"""
26
27         expected_start = "modulus"
28         # test assumes if it starts off fine it will
continue being fine
29         received_start = self.loader.load(self.
key_path)
30         self.assertEqual(expected_start,
received_start[:7])
31
32     def test_parsing(self):
33         """Checks that keys are parsed correctly and
that errors are raised when inputs are incorrect"""

```

File - /home/tcassar/projects/settle/tests/test\_crypto/test\_keys.py

```
34     loader = self.loader
35
36     loader.load(self.key_path)
37     loader.parse()
38
39     # compare parsed file to known values from
40     # test keys.
41     # lay out known values
42     k_e = 65537
43
44     k_d =
45         4231860502610347555352698504423047563944256813822996
46         3925734038989576569874864579669508371562955146126547
47         8779552919144303233011283933461726857674426838187389
48         6721663847672076407494464781754007098969569658821095
49         1082558372250397158894945714319204774976709066583095
50         9656389449195952398626578010004019881519663179628663
51         1049344644422180019003905454847914081363056944472084
52         7689992964879750244948034517871603890465032993904165
53         5974782350016040380747036826804133887037575013624446
54         8012222270131756222483185448064466203435454227528122
55         731438704682552557037682427868460211954070416261823
56         5709118874694072838664235842554512315661905
57
58     k_n =
59
60         2161679203114375274630941557945232020251089312607308
61         7652383723408105063600070147761500936454570470862786
62         9702069833686361660030089751732511247633740543213460
63         3035445702142516535603778163693003610640441829541372
64         643100255683690006704986794499904312842155745102543
65         7279819137427553642625019821057158620227234339857381
66         3906769437847646651445567701079686709061678948557745
67         8637370869261774337704654931841775536224420241750390
68         3181827421441408785602795322818434551136750209831842
69         5250366913849245188288359620277590091100050269612943
70         3770297061845035322423193568722584809589006962060587
71         421954603201784497846158263582144177430642603
72
73     # build lists for testing
74     cases = [
75         "pub_exp",
76         "priv_exp",
```

```
50             "mod",
51             "prime1",
52             "prime2",
53             "exponent1",
54             "exponent2",
55             "coefficient",
56         ]
57
58     known = [k_e, k_d, k_n]
59
60     key = keys.RSAPrivateKey(loader)
61
62     received = [
63         key.e,
64         key.d,
65         key.n,
66         key.p,
67         key.q,
68         key.exp1,
69         key.exp2,
70         key.crt_coef,
71     ]
72
73     for test, known_val, rec_val in zip(cases,
74                                         known, received):
75         with self.subTest(test):
76             self.assertEqual(rec_val, known_val)
77             self.assertEqual(type(rec_val), type
78                             (known_val))
79
80     def test_file_not_found(self):
81         path = "./adsads"
82
83         with self.assertRaises(keys.RSAParserError):
84             loader = self.loader
85             loader.load(path)
86
87     def test_wrong_format(self):
88         """Checks that we can deal with files being
89         the wrong format"""
90         with self.assertRaises(keys.RSAParserError):
```

```
88             self.loader.load("./crypto/sample_keys/  
d_public-key.pem")  
89  
90     def test_unparsed_key(self):  
91         """Check accessing attributes"""  
92         loader = self.loader  
93         loader.load(self.key_path)  
94         key = keys.RSAPrivateKey(loader)  
95  
96         with self.assertRaises(keys.RSAKeyError):  
97             _ = key.asdf  
98  
99     def test_unloaded_key(self):  
100        """Checks that error is raised if parse()  
is called on a loader object which has not yet  
loaded a key"""  
101        ldr = self.loader  
102        with self.assertRaises(keys.RSAParserError  
):  
103            ldr.parse()  
104  
105    def test_public_key(self):  
106        """Checks that public keys will raise an  
exception if they are used for signing"""  
107        ldr = self.loader  
108        ldr.load(self.key_path)  
109        ldr.parse()  
110  
111        pub_key = keys.RSAPublicKey(ldr)  
112  
113        with self.subTest("allowed access"):  
114            self.assertEqual(pub_key.e, 65537)  
115  
116        with (self.subTest("deny access"), self.  
assertRaises(keys.RSAPublicKeyError)):  
117            _ = pub_key.p  
118
```

```

1 # coding=utf-8
2 import hashlib
3 import sys
4 from unittest import TestCase
5
6 from src.crypto import hashes
7
8
9 class TestHash(TestCase):
10     """
11         Test hash interfaces (functionality not aim of
12             testing as functionality comes from hashlib
13             1) Check initialisation as expected
14             2) Check update
15             3) Check digest
16             4) Check hexdigest == intdigest
17             """
18
19     def test_init(self) -> None:
20         """Checks that _hasher is initialised
21             correctly i.e. no strange start values"""
22         h = hashes.Hasher()
23         self.assertEqual(
24             h.digest().int_digest(),
25             int.from_bytes(hashlib.sha3_256(b"").digest(),
26                           byteorder=sys.byteorder),
27         ) # should be initialised empty
28
29     def test_hash(self) -> None:
30         """tests that hash object can validate hash
31             looking numbers"""
32         with self.subTest("valid"):
33             h_val = hashlib.sha3_256(b"1234").digest()
34             h = hashes.Hash(h_val) # create a
35             # definitely valid hash
36
37             with self.subTest("too short"):
38                 with self.assertRaises(hashes.HashError
39                 ):
40                     hashes.Hash(b"12")

```

```
35
36         with self.subTest("too long"):
37             with self.assertRaises(hashes.HashError):
38                 hashes.Hash(
39                     b"
40                         1157920892373161954235709850086879078532699846656405
41                         640394575840079131296399360"
42
43         with self.subTest("wrong type"):
44             with self.assertRaises(hashes.HashError):
45                 # noinspection PyTypeChecker
46                 hashes.Hash("dave")
47
48     def test_update_digest(self) -> None:
49         """Ensures that a hash with a given value
50         will digest the correct thing"""
51         h = hashes.Hasher(b"1234")
52         h_ = hashlib.sha3_256(b"1234")
53
54         with self.subTest("before update"):
55             self.assertEqual(h.digest().h, h_.digest())
56
57             update_msg = b"test hash update"
58             h.update(update_msg)
59             h_.update(update_msg)
60
61         with self.subTest("after update"):
62             self.assertEqual(h.digest().h, h_.digest())
63
64     def test_hasher_fails(self):
65         """Checks that hasher objects when not bytes
66         are passed into it"""
67         with self.assertRaises(hashes.HasherError):
68             # noinspection PyTypeChecker
69             hasher = hashes.Hasher("abd")
```

```

1 # coding=utf-8
2
3 """
4 Testing sign / verify through RSA working as
5 expected
6 """
7 import os
8 from unittest import TestCase
9
10 from src.crypto import keys
11 from src.crypto import rsa
12 from src.transactions.transaction import Signable
13
14 def setUpModule():
15     os.chdir("/home/tcassar/projects/settle/src")
16
17
18 class TestRSA(TestCase):
19     """Just tests RSA parts"""
20
21     def setUp(self) -> None:
22         # load keys
23         ldr = keys.RSAKeyLoader()
24         ldr.load("./crypto/sample_keys/d_private-key
25 .pem")
26         ldr.parse()
27
28         # build public and private
29         self.private = keys.RSAPrivateKey(ldr)
30         self.public = keys.RSAPublicKey(ldr)
31
32     def test_encryption(self):
33         """Checks to see if process is reversible,
34         and encrypted is different to how it started"""
35         message = " | maia"
36         m_bytes = bytes(message, encoding="utf8")
37         encrypted = rsa.RSA.encrypt(m_bytes, self.
38             public)
39         # TODO: actually fix byte overflow affair
40         decrypted = rsa.RSA.bytes_to_str(rsa.RSA.

```

```
37    naive_decrypt(encrypted, self.private))
38
39        with self.subTest("Catch Public Key"):
40            with self.assertRaises(rsa.
41                DecryptionError):
42                _ = rsa.RSA.naive_decrypt(encrypted
43                , self.public) # type: ignore
44
45        with self.subTest("encrypted"):
46            self.assertNotEqual(m_bytes, encrypted)
47
48        with self.subTest("Successful decryption"):
49            self.assertEqual(message, decrypted)
50
51
52    def test_RSA_sign(self):
53        """Checks for consistent creating /
54        verifying of a 'signature'"""
55
56        message = " | maia"
57        m_bytes = message.encode("utf8")
58        sig: bytes = rsa.RSA.sign(m_bytes, self.
59            private)
60        de_sign: bytes = rsa.RSA.inv_sig(sig, self.
61            public)
62
63        self.assertEqual(m_bytes, de_sign)
```

```

1 # coding=utf-8
2 from unittest import TestCase
3
4 from src.simplify.base_graph import Digraph
5 from src.simplify.flow_graph import FlowGraph
6 from src.simplify.graph_objects import Vertex
7 from src.simplify.path import Path, prev_map,
disc_map, BFSQueue
8 from src.simplify.weighted_digraph import
WeightedDigraph
9
10
11 class TestPath(TestCase):
12     """Tests searching for BFS"""
13
14     def setUp(self):
15         # make a graph with 5 nodes
16
17         labels = ["a", "b", "c", "d", "e", "f"]
18         self.vertices = [Vertex(ID, label=label) for
ID, label in enumerate(labels)]
19
20         self.g = Digraph(self.vertices)
21         a, b, c, d, e, f = self.vertices
22         self.g.add_edge(a, b, c)
23         self.g.add_edge(b, d)
24         self.g.add_edge(d, f)
25         self.g.add_edge(c, e)
26         self.g.add_edge(d, f)
27         self.g.add_edge(e, f, b)
28
29         # set up weighted_graph and flow graphs
30         self.weighted_graph = WeightedDigraph(self.
vertices)
31         self.flow_graph = FlowGraph(self.vertices)
32
33         for g in [self.weighted_graph, self.
flow_graph]:
34             g.add_edge(a, (b, 10), (c, 10))
35             g.add_edge(b, (d, 25))
36             g.add_edge(c, (e, 25))

```

```

37             g.add_edge(d, (f, 10))
38             g.add_edge(e, (f, 10), (b, 6))
39
40     def test_shortest_path(self):
41         """
42             Checks that we can in fact find shortest
43             path along
44             """
45
46             a, b, c, d, e, f = self.vertices
47
48             cases = ["digraph", "weighted_graph", "flow"]
49
50             # check precalculated shortest path
51             graph_cases = [self.g, self.weighted_graph,
52                             self.flow_graph]
53
54             for case, g in zip(cases, graph_cases):
55                 with self.subTest(case):
56                     calc_shorted: list[Vertex] = Path.
57                     shortest_path(
58                         self.g, a, f, self.g.neighbours
59                     )
60                     expected: list[Vertex] = [a, c, e, f
61 ]
62                     self.assertEqual(expected,
63                     calc_shorted)
64
65             def test_build_path(self):
66                 """Given a prev_map, check we build the
67                 right path"""
68
69                 # generate vertices, unpack into vars
70                 vertices = []
71                 for ID, name in enumerate(["A", "B", "C", "D",
72                     "E", "F"]):
73                     vertices.append(Vertex(ID, name))
74
75                 a, c, b, d, e, f = vertices

```

```

70          # build a prev map. for context, path is A
71          # -> B -> D -> F
72          prev: prev_map = {a: None, b: a, c: a, d: b
73          , e: c, f: d}
74          expected = [a, b, d, f]
75
76      def test_recursive_bfs(self):
77          """Check that BFS is finding paths along
78          graph correctly"""
79          graph = self.flow_graph
80          a, b, c, d, e, f = graph.nodes()
81
82          expected: prev_map = {a: None, b: a, c: a,
83          d: b, e: c, f: e}
84
85          # set up structures for BFS
86          # create queue, discovered list, previous
87          # list
88          queue = BFSQueue(next(iter(graph.graph)))
89
90          discovered: disc_map = {node: False for
91          node in graph.nodes()}
92          prev: prev_map = {node: None for node in
93          graph.nodes()}
94
95          self.assertEqual(
96              expected,
97              Path.BFS(
98                  graph=graph,
99                  queue=queue,
100                 discovered=discovered,
101                 target=None,
102                 previous=prev,
103                 neighbours=graph.neighbours,
104                 ),
105             )
106
107      def test_find_target(self):

```

```
103     """Checks that BFS can stop when its given
104     target is found"""
105     graph = self.flow_graph
106     a, b, c, *_ = graph.nodes()
107     expected = {node: None for node in graph.
108     nodes()}
109     expected[b] = a
110     expected[c] = a
111     queue, discovered, previous = Path.
112     build_bfs_structs(graph, a)
113     calculated = Path.BFS(
114         graph=graph,
115         queue=queue,
116         discovered=discovered,
117         previous=previous,
118         target=b,
119         neighbours=graph.neighbours,
120     )
121     self.assertEqual(expected, calculated)
122
123     def test_build_bfs_struct(self):
124         """Checks that structures that are built
125         for BFS are built correctly"""
126         with self.subTest("with initial value"):
127             queue, disc, prev = Path.
128             build_bfs_structs(
129                 self.flow_graph, self.vertices[0]
130             )
131             self.assertEqual(queue, BFSQueue(self.
132             vertices[0]))
133         with self.subTest("with initial value"):
134             queue, disc, prev = Path.
135             build_bfs_structs(self.flow_graph)
136             self.assertEqual(queue, BFSQueue())
```

```

1 # coding=utf-8
2 from unittest import TestCase
3
4 from src.simplify.flow_algorithms import
5     NoOptimisations, MaxFlow, Simplify
6 from src.simplify.flow_graph import *
7 from src.simplify.graph_objects import Vertex
8
9
9 class TestFlowEdge(TestCase):
10     """Tests for FlowEdge"""
11
12     def setUp(self) -> None:
13         self.edges = [FlowEdge(Vertex(0), Vertex(1),
14             5 * n) for n in range(2)]
14         self.edges[0].flow = -3 # => unused
15         capacity should be three
15
16     def test_unused_capacity(self):
17         """builds two edges, residual and non-
18         residual
19             non-residual has capacity 5"""
20
20         for edge, cap in zip(self.edges, [3, 5]):
21             with self.subTest(edge):
22                 self.assertEqual(edge.
22 unused_capacity(), cap)
23
24     def test_push_flow(self):
25         """Checks that we update flow by required
26         amount"""
26         # push three units of capacity through each
27         edge
27         for edge, exp in zip(self.edges, [0, 2]):
28             with self.subTest(edge):
29                 edge.push_flow(3)
30                 self.assertEqual(exp, edge.
30 unused_capacity())
31
32         # try to push excess amount of flow down an
32         edge

```

```

33         with self.subTest("exceed capacity"), self.
34             assertRaises(FlowEdgeError):
35                 self.edges[1].push_flow(3)
36
37     def test_adjust_edge(self):
38         """Tests that the adjust_edges function
39         works as expected
40             i.e when adjust is called, edges in the
41             graph morph to edges with a flow equal to their
42             unused capacity
43             and all edges of weight 0 are deleted"""
44
45         (
46             res,
47             fwd,
48         ) = self.edges
49         fwd.push_flow(3)
50         res.push_flow(-3)
51
52         new_fwd = FlowEdge(Vertex(0), Vertex(1), 2)
53         new_res = FlowEdge(Vertex(0), Vertex(1), 0)
54
55         fwd.adjust_edge()
56         res.adjust_edge()
57
58         self.assertEqual(new_fwd, fwd)
59         self.assertEqual(new_res, res)
60
61
62     class TestFlowGraph(TestCase):
63         def setUp(self) -> None:
64             # make some nodes for a graph
65             self.nodes = [Vertex(n, label=chr(n + 97))
66             for n in range(5)]
67             self.graph = FlowGraph(self.nodes)
68
69         def test_is_edge(self):

```

File - /home/tcassar/projects/settle/tests/test\_settling/test\_flow.py

```
69     a, b, c, d, e = self.nodes
70
71     with self.subTest("no edge"):
72         self.assertFalse(self.graph.is_edge(a,
73                           b))
74
75     with self.subTest("edge"):
76         self.graph.add_edge(a, (b, 4))
77         self.assertTrue(self.graph.is_edge(a, b))
78
79     def test_get_edge(self):
80         a, b, c, d, e = self.graph.nodes()
81         self.graph.add_edge(a, (b, 5))
82         self.assertEqual(FlowEdge(a, b, 5), self.
83             graph.get_edge(a, b))
84
85     def test_add_edge(self):
86         a, b, c, d, e = self.nodes
87         # check there aren't edges
88         with self.subTest("negative test"):
89             self.assertFalse(self.graph.is_edge(a,
90                           b))
91             self.assertFalse(self.graph.is_edge(b,
92                           a, residual=True))
93             self.graph.to_dot()
94             self.graph.add_edge(a, (b, 5))
95             self.graph.to_dot()
96             with self.subTest("added"):
97                 self.assertTrue(self.graph.is_edge(a, b))
98                 self.assertTrue(self.graph.is_edge(b, a,
99                               , residual=True))
100                self.assertFalse(self.graph.is_edge(b,
101                               a))
102
103    with self.subTest("Net debt (adding)"):
104        self.assertEqual(
105            {
106                Vertex(ID=0, label="a"): 5,
107                Vertex(ID=1, label="b"): -5,
```

```
102                     Vertex(ID=2, label="c"): 0,
103                     Vertex(ID=3, label="d"): 0,
104                     Vertex(ID=4, label="e"): 0,
105                 },
106                 self.graph.net_debt,
107             )
108
109         with self.subTest("Net debt (removing)"):
110             self.graph.pop_edge(a, b, update_debt=
True)
111             self.assertEqual(
112                 {
113                     Vertex(ID=0, label="a"): 0,
114                     Vertex(ID=1, label="b"): 0,
115                     Vertex(ID=2, label="c"): 0,
116                     Vertex(ID=3, label="d"): 0,
117                     Vertex(ID=4, label="e"): 0,
118                 },
119                 self.graph.net_debt,
120             )
121
122     def test_pop_edge(self):
123         a, b, c, d, e = self.nodes
124
125         self.graph.add_edge(a, (b, 5))
126
127         with self.subTest("negative"):
128             self.assertTrue(self.graph.is_edge(a, b))
129             self.assertTrue(self.graph.is_edge(b, a,
, residual=True))
130
131         self.graph.pop_edge(a, b)
132
133         with self.subTest("removed edge"):
134             self.assertFalse(self.graph.is_edge(a,
b))
135         with self.subTest("removed residual"):
136             self.assertFalse(self.graph.is_edge(b,
a, residual=True))
137
```

```

138     def test_flow_neighbours(self):
139         """Checks we get edges that have unused
140             capacity, including residual
141             These edges count as valid paths to
142             neighbouring nodes in Edmonds-Karp"""
143         graph = self.graph
144         a, b, c, d, *_ = graph.nodes()
145         graph.add_edge(a, (b, 10))
146         graph.add_edge(b, (c, 2))
147         graph.add_edge(c, (d, 10))
148         graph.add_edge(d, (a, 10))
149
150         MaxFlow.augment_flow(graph, [a, b, c, d], 2
151     )
152         c_flow_neighbours = graph.flow_neighbours(c
153     )
154         self.assertEqual([d, b], GenericDigraph.
155             nodes_from_edges(c_flow_neighbours))
156
157     def test_bool(self):
158         """Tests that dunder bool method returns
159             true when there are edges in the graph"""
160         # empty
161         print(self.graph.graph)
162         self.assertFalse(bool(self.graph))
163
164         # with an edge
165         a, b, c, d, e = self.nodes
166         self.graph.add_edge(a, (b, 10))
167
168         def test_add_existing(self):
169             """Tests that adding an existing edge will
170                 add capacity onto the existing edge,
171                 as opposed to adding a new distinct edge
172                 from src -> dest"""
173             a, b, c, d, e = self.graph.nodes()
174             self.graph.add_edge(a, (b, 5))
175             self.graph.add_edge(a, (b, 5))
176             print(self.graph)

```

```

171         self.assertEqual(self.graph.get_edge(a, b).
172                         capacity, 10)
173
174     def test_pop_existing(self):
175         """Tests that if an edge A -> B of weight [5] exists, adding B -> A [5] will remove
176             any edges between A and B, as there is net
177             0 between the two"""
178
179         a, b, c, d, e = self.graph.nodes()
180         self.graph.add_edge(a, (b, 5))
181         self.graph.add_edge(b, (a, 5))
182
183         self.assertFalse(self.graph.is_edge(a, b))
184
185     def test_add_edge_reverses_new_edge(self):
186         """Checks that if there exists an edge A
187             -> B of weight [5] exists, adding B -> A [10] will
188             reverse the direction of the edge such that
189             now an edge B -> A [5] exists"""
190
191         a, b, c, d, e = self.graph.nodes()
192         self.graph.add_edge(a, (b, 5))
193         self.graph.add_edge(b, (a, 10))
194
195         self.assertFalse(self.graph.is_edge(a, b))
196         self.assertEqual(self.graph.get_edge(b, a).
197                         capacity, 5)
198
199
200 class TestMaxFlow(TestCase):
201     def setUp(self) -> None:
202         graph = FlowGraph([Vertex(n, label=chr(n +
203                                     97)) for n in range(4)])
204
205         a, b, c, d, *_ = graph.nodes()
206         graph.add_edge(a, (b, 10))
207         graph.add_edge(b, (c, 2))
208         graph.add_edge(c, (d, 10))
209         graph.add_edge(d, (a, 10))
210
211         self.graph = graph
212
213     def test_bottleneck(self):

```

```

205         """Checks we get the correct bottleneck
206         value on a path"""
207         # build a chain with an obvious bottleneck
208         # on normal graphs
209         graph = self.graph
210         a, b, c, d, *_ = graph.nodes()
211
212         aug_path = [a, b, c, d]
213
214         graph.to_dot()
215
216     def test_nodes_to_path(self):
217         """Checks that we can build a path of edges
218         from nodes"""
219         a, b, c, d, *_ = self.graph.nodes()
220         edges = MaxFlow.nodes_to_path(self.graph, [
221             a, b, c, d])
222
223         self.assertEqual(
224             edges, [FlowEdge(a, b, 10), FlowEdge(b,
225                 c, 2), FlowEdge(c, d, 10)])
226
227     def test_augmenting_path(self):
228         """Tests that we find valid augmenting
229         paths"""
230         a, b, c, d, *_ = self.graph.nodes()
231         MaxFlow.augmenting_path(self.graph, a, d)
232
233     def test_augment_flow(self):
234         """Checks that augmenting flow works
235         exactly as described in Analysis"""
236         a, b, c, d, *_ = self.graph.nodes()
237
238         MaxFlow.augment_flow(self.graph, [a, b, c,
239             d], 2)
240
241         self.graph.to_dot()
242
243

```

File - /home/tcassar/projects/settle/tests/test\_settling/test\_flow.py

```

269         tg.to_dot()
270
271         max_flow = MaxFlow.edmonds_karp(tg, s, t)
272         self.assertEqual(max_flow, 20)
273
274     def test_old_edmonds(self):
275         """Tests a more complex graph for correct
maxflow"""
276         labels = ["a", "b", "c", "d", "e", "f"]
277         self.vertices = [Vertex(ID, label=label)
278             for ID, label in enumerate(labels)]
279         a, b, c, d, e, f = self.vertices
280
281         # set up weighted_graph and flow graphs
282         self.flow_graph = FlowGraph(self.vertices)
283         self.flow_graph.add_edge(a, (b, 10), (c, 10))
284         self.flow_graph.add_edge(b, (d, 25))
285         self.flow_graph.add_edge(c, (e, 25))
286         self.flow_graph.add_edge(d, (f, 10))
287         self.flow_graph.add_edge(e, (f, 10), (b, 6))
288         a, b, c, d, e, f = self.vertices
289
290         expected = 20
291         calculated = MaxFlow.edmonds_karp(self.
292             flow_graph, a, f)
293
294         self.assertEqual(expected, calculated)
295
296 class TestSimplify(TestCase): # type: ignore
297     def setUp(self) -> None:
298         self.graph = FlowGraph([Vertex(0, "d"),
299             Vertex(1, "m"), Vertex(2, "t")])
300         d, m, t = self.graph.nodes()
301         self.graph.add_edge(d, (m, 5), (t, 10))
302         self.graph.add_edge(m, (t, 5))
303
304     def test_simplify_debt(self):

```

```

304         """Uses graph from analysis to test that
305             the debt simplifying algorithm simplifies debt,
306             does not create or
307                 destroy debt"""
308
309     print(self.graph.net_debt)
310
311     people = ["dad", "tom", "maia"]
312     debt = FlowGraph([Vertex(ID, person) for ID
313 , person in enumerate(people)])
314
315     d, t, m = debt.nodes()
316
317     debt.add_edge(d, (t, 10), (m, 5))
318     debt.add_edge(m, (t, 5))
319
320     debt.to_dot()
321
322     clean = Simplify.simplify_debt(debt)
323
324     def test_adjust_edges(self):
325
326         """Checks that edges are adjusted
327             accordingly, generate graphs before and after"""
328
329         # saturate d -> m -> t
330         self.graph.to_dot(n=0)
331         d, m, t = self.graph.nodes()
332         MaxFlow.augment_flow(self.graph, [d, m, t
333 ], 5)
334         self.graph.to_dot(n=1)
335         self.graph.adjust_edges()
336         self.graph.to_dot(n=4)
337
338         # graph should have no edges in or out of m
339         # t should have a res edge to d, 0/0
340         # d should have a fwd to t, 0/10

```

```

339
340         self.assertFalse(self.graph[m])
341         self.assertEqual(self.graph[d], [FlowEdge(d
342             , t, 10)])
343         self.assertEqual(self.graph[t], [FlowEdge(t
344             , d, 0)])
345
346     def test_mithun_simplify(self):
347
348         """Checks more complex example, ensures
349         that graph is settled and that no debt is created
350         / destroyed"""
351
352         # gen vertices
353         people: list[Vertex] = []
354         for ID, person in enumerate(["b", "c", "d"
355             , "e", "f", "g"]):
356             people.append(Vertex(ID, label=person))
357         b, c, d, e, f, g = people
358
359         # build flow graph of transactions
360         messy = FlowGraph(people)
361
362         messy.add_edge(b, (c, 40))
363         messy.add_edge(c, (d, 20))
364         messy.add_edge(d, (e, 50))
365         messy.add_edge(f, (e, 10), (d, 10), (c, 30
366             ), (b, 10))
367         messy.add_edge(g, (b, 30), (d, 10))
368
369         messy.to_dot()
370
371         clean = Simplify.simplify_debt(messy)
372         clean.to_dot(n=1)
373
374         self.assertEqual(messy.net_debt, clean.
375             net_debt)
376
377     def test_simplest_graph(self):
378         """Shouldn't change"""
379         people = ["dad", "tom", "maia"]

```

```
373     debt = FlowGraph([Vertex(ID, person) for ID
374         , person in enumerate(people)])
375     d, t, m = debt.nodes()
376     debt.add_edge(d, (t, 5))
377     debt.add_edge(t, (m, 10))
378     debt.to_dot()
379
380     with self.assertRaises(NoOptimisations):
381         clean = Simplify.simplify_debt(debt)
382         clean.to_dot(n=1)
383
```

```

1 # coding=utf-8
2
3 from unittest import TestCase
4
5 from src.simplify.base_graph import Digraph
6 from src.simplify.flow_graph import *
7 from src.simplify.graph_objects import Edge
8 from src.simplify.weighted_digraph import
9     WeightedDigraph
10
11 class TestDigraph(TestCase):
12     """Generate a digraph and some vertices"""
13
14     def setUp(self) -> None:
15         """Build basic graph"""
16         labels = ["u", "v", "w"]
17         self.vertices = [Vertex(ID, label=label) for
18             ID, label in enumerate(labels)]
19
20         self.graph = Digraph(self.vertices)
21         u, v, w = self.vertices
22         self.graph.add_edge(u, v, w)
23         self.graph.add_edge(v, w)
24
25     def test_init(self):
26         expected = "U -> V\nV -> W\nW -> \n"
27
28         with self.subTest("init"):
29             self.assertEqual(expected, str(self.
graph))
30
31         with self.subTest("helpers"):
32             self.assertTrue(self.graph.is_node(self.
vertices[0]))
33             self.assertTrue(self.graph.sanitize(*
self.vertices))
34
35         with self.assertRaises(GraphError):
36             self.graph.edge_from_nodes(self.
vertices[2], [Edge(self.vertices[0])])

```

```
36
37     def test_is_edge(self):
38         u, v, w = self.vertices
39
40         with self.subTest("is edge"):
41             self.assertTrue(self.graph.is_edge(u, v))
42         with self.subTest("isn't edge"):
43             self.assertFalse(self.graph.is_edge(w, v))
44
45     def test_add_node(self):
46
47         new = Vertex(4, "b")
48         self.assertFalse(self.graph.is_node(new))
49
50         self.graph.add_node(new)
51         self.assertTrue(self.graph.is_node(new))
52
53     def test_pop_node(self):
54         u, v, w = self.vertices
55
56         print(self.graph)
57         self.graph.pop_node(u)
58         print(self.graph)
59         self.assertFalse(self.graph.is_node(u))
60         with self.assertRaises(GraphError):
61             self.graph.edge_from_nodes(u, self.graph
62 [v])
63
64     def test_pop_edge(self):
65         u, v, w = self.vertices
66         self.assertTrue(self.graph.is_edge(u, v))
67         self.graph.pop_edge(u, v)
68
69         self.assertFalse(self.graph.is_edge(u, v))
70
71     def test_add_edge(self):
72         u, v, w = self.graph.nodes()
73         self.assertFalse(self.graph.is_edge(w, v))
74         self.graph.add_edge(w, v)
```

```
74
75             self.assertTrue(self.graph.is_edge(w, v))
76
77     def test_nodes(self):
78         self.assertEqual(self.vertices, self.graph.
nodes())
79
80
81 class TestWeightedDigraph(TestCase):
82     """Artefact of prototyping process"""
83     def setUp(self) -> None:
84         """Build basic graph"""
85         labels = ["u", "v", "w"]
86         self.vertices = [Vertex(ID, label=label)
87             for ID, label in enumerate(labels)]
88
89         self.graph = WeightedDigraph(self.vertices)
90         u, v, w = self.vertices
91         self.graph.add_edge(u, (v, 1), (w, 2))
92         self.graph.add_edge(v, (w, 3))
93
94     def test_add_edge(self):
95         u, v, w = self.vertices
96         self.assertFalse(self.graph.is_edge(w, v))
97         self.graph.add_edge(w, (v, 4))
98         self.assertTrue(self.graph.is_edge(w, v))
99
100    def test_add_existing_edge(self):
101        u, v, w = self.graph.nodes()
102        self.assertEqual(self.graph.is_edge(v, w),
103            3)
104        self.graph.add_edge(v, (w, 2))
105        self.assertEqual(self.graph.is_edge(v, w),
106            5)
107
108    def test_flow_through(self):
109        for node, flow in zip(self.vertices, [3, 2,
110            -5]):
111            with self.subTest(node):
112                self.assertEqual(self.graph.
flow_through(node), flow)
```



```
1 digraph { dad -> tom [label=" 0/10  "]
2 dad -> maia [label=" 0/5  "]
3 tom -> dad [label=" 0/0  ", color=red]
4 tom -> maia [label=" 0/0  ", color=red]
5 maia -> tom [label=" 0/5  "]
6 maia -> dad [label=" 0/0  ", color=red]
7 }
8
```



File - /home/tcassar/projects/settle/tests/test\_settling/graph\_renders/graph1

```
1 digraph { dad -> tom [label=" 0/15  "]
2 tom -> dad [label=" 0/0  ", color=red]
3 maia
4 }
5
```



File - /home/tcassar/projects/settle/tests/test\_settling/graph\_renders/graph4

```
1 digraph { d -> t [label=" 0/10  "]
2 m
3 t -> d [label=" 0/0  ", color=red]
4 }
5
```



```
1 <?xml version="1.0" encoding="UTF-8" standalone="no"
?
2 <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
3   "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
4 <!-- Generated by graphviz version 2.43.0 (0)
5   -->
6 <!-- Title: %3 Pages: 1 -->
7 <svg width="223pt" height="218pt"
8   viewBox="0.00 0.00 223.20 218.00" xmlns="http://www
9   .w3.org/2000/svg" xmlns:xlink="http://www.w3.org/
10  1999/xlink">
11 <g id="graph0" class="graph" transform="scale(1 1)
12  rotate(0) translate(4 214)">
13 <title>%3</title>
14 <polygon fill="white" stroke="transparent" points="-
15  4,4 -4,-214 219.2,-214 219.2,4 -4,4"/>
16 <!-- dad -->
17 <g id="node1" class="node">
18 <title>dad</title>
19 <ellipse fill="none" stroke="black" cx="103.2" cy="-
20  192" rx="27.9" ry="18"/>
21 <text text-anchor="middle" x="103.2" y="-188.3" font-
22  -family="Times,serif" font-size="14.00">dad</text>
23 </g>
24 <!-- tom -->
25 <g id="node2" class="node">
26 <title>tom</title>
27 <ellipse fill="none" stroke="black" cx="38.2" cy="-
28  105" rx="28.7" ry="18"/>
29 <text text-anchor="middle" x="38.2" y="-101.3" font-
30  -family="Times,serif" font-size="14.00">tom</text>
31 </g>
32 <!-- dad->tom -->
33 <g id="edge1" class="edge">
34 <title>dad->tom</title>
35 <path fill="none" stroke="black" d="M75.1,-189.65C51
36  .59,-186.87 19.37,-178.8 3.2,-156 -3.87,-146.04 2.51
37  ,-134.92 11.67,-125.7"/>
38 <polygon fill="black" stroke="black" points="14.29, -
39  128.05 19.4,-118.77 9.62,-122.84 14.29,-128.05"/>
40 <text text-anchor="middle" x="28.2" y="-144.8" font-
```

```

29 family="Times,serif" font-size="14.00"> 0/10
  &#160;</text>
30 </g>
31 <!-- maia -->
32 <g id="node3" class="node">
33 <title>maia</title>
34 <ellipse fill="none" stroke="black" cx="103.2" cy="-18" rx="33.29" ry="18"/>
35 <text text-anchor="middle" x="103.2" y="-14.3" font-family="Times,serif" font-size="14.00">maia</text>
36 </g>
37 <!-- dad->maia -->
38 <g id="edge2" class="edge">
39 <title>dad->maia</title>
40 <path fill="none" stroke="black" d="M111.23,-174.66C113.65,-168.9 115.98,-162.3 117.2,-156 125.77,-111.48 125.77,-98.52 117.2,-54 116.64,-51.11 115.84,-48.15 114.91,-45.24"/>
41 <polygon fill="black" stroke="black" points="118.13,-43.85 111.37,-35.7 111.57,-46.29 118.13,-43.85"/>
42 <text text-anchor="middle" x="144.7" y="-101.3" font-family="Times,serif" font-size="14.00"> 0/5
  &#160;</text>
43 </g>
44 <!-- tom->dad -->
45 <g id="edge3" class="edge">
46 <title>tom->dad</title>
47 <path fill="none" stroke="red" d="M50.15,-121.79C57.62,-131.64 67.45,-144.57 76.2,-156 79.05,-159.73 82.09,-163.69 85.06,-167.54"/>
48 <polygon fill="red" stroke="red" points="82.46,-169.9 91.34,-175.69 88.01,-165.63 82.46,-169.9"/>
49 <text text-anchor="middle" x="96.7" y="-144.8" font-family="Times,serif" font-size="14.00"> 0/0
  &#160;</text>
50 </g>
51 <!-- tom->maia -->
52 <g id="edge4" class="edge">
53 <title>tom->maia</title>
54 <path fill="none" stroke="red" d="M22.24,-89.46C12.94,-79.19 4.44,-65.29 12.2,-54 23.19,-37.99 42.61,-

```

```
54 29.3 60.7,-24.58"/>
55 <polygon fill="red" stroke="red" points="61.57,-27.
56 98 70.55,-22.35 60.02,-21.15 61.57,-27.98"/>
57 <text text-anchor="middle" x="32.7" y="-57.8" font-
58   family="Times,serif" font-size="14.00"> &#160;0/0 &#
59   160;</text>
60 </g>
61 <!-- maia&gt;dad -->
62 <g id="edge6" class="edge">
63 <title>maia&gt;dad</title>
64 <path fill="none" stroke="red" d="M123,-32.56C138.86
65 , -44.72 159.92,-64.13 169.2,-87 175.21,-101.83 175.
66 21,-108.17 169.2,-123 161.16,-142.81 144.28,-160.03
67 129.6,-172.18"/>
68 <polygon fill="red" stroke="red" points="127.42,-169
69 .45 121.74,-178.4 131.76,-174.94 127.42,-169.45"/>
70 <text text-anchor="middle" x="194.7" y="-101.3" font-
71   family="Times,serif" font-size="14.00"> &#160;0/0
72   160;</text>
73 </g>
74 <!-- maia&gt;tom -->
75 <g id="edge5" class="edge">
76 <title>maia&gt;tom</title>
77 <path fill="none" stroke="black" d="M90.8,-35.01C86.
78 21,-40.97 80.95,-47.78 76.2,-54 69.7,-62.48 62.61,-
79 71.79 56.34,-80.06"/>
80 <polygon fill="black" stroke="black" points="53.4,-
81 78.13 50.15,-88.21 58.98,-82.36 53.4,-78.13"/>
82 <text text-anchor="middle" x="96.7" y="-57.8" font-
83   family="Times,serif" font-size="14.00"> &#160;0/5 &#
84   160;</text>
85 </g>
86 </g>
87 </svg>
```



```
1 <?xml version="1.0" encoding="UTF-8" standalone="no"
?
2 <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
3   "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
4 <!-- Generated by graphviz version 2.43.0 (0)
5   -->
6 <!-- Title: %3 Pages: 1 -->
7 <svg width="157pt" height="131pt"
8   viewBox="0.00 0.00 156.72 131.00" xmlns="http://www
9   .w3.org/2000/svg" xmlns:xlink="http://www.w3.org/
10  1999/xlink">
11 <g id="graph0" class="graph" transform="scale(1 1)
12  rotate(0) translate(4 127)">
13 <title>%3</title>
14 <polygon fill="white" stroke="transparent" points="-
15  4,4 -4,-127 152.72,-127 152.72,4 -4,4"/>
16 <!-- dad -->
17 <g id="node1" class="node">
18 <title>dad</title>
19 <ellipse fill="none" stroke="black" cx="36.57" cy="-
20  105" rx="27.9" ry="18"/>
21 <text text-anchor="middle" x="36.57" y="-101.3" font-
22  -family="Times,serif" font-size="14.00">dad</text>
23 </g>
24 <!-- tom -->
25 <g id="node2" class="node">
26 <title>tom</title>
27 <ellipse fill="none" stroke="black" cx="36.57" cy="-
28  18" rx="28.7" ry="18"/>
29 <text text-anchor="middle" x="36.57" y="-14.3" font-
30  -family="Times,serif" font-size="14.00">tom</text>
31 </g>
32 <!-- dad>tom -->
33 <g id="edge1" class="edge">
34 <title>dad>tom</title>
35 <path fill="none" stroke="black" d="M18.56,-91.03C11
36  .91,-85.14 5.12,-77.57 1.57,-69 -2.82,-58.4 2.77,-47
37  .82 10.67,-39.14"/>
38 <polygon fill="black" stroke="black" points="13.26, -
39  41.51 18.03,-32.05 8.4,-36.47 13.26,-41.51"/>
40 <text text-anchor="middle" x="26.57" y="-57.8" font-
```

```
29 family="Times,serif" font-size="14.00"> &#160;0/15  
  &#160;</text>  
30 </g>  
31 <!-- tom&#45;&gt;dad -->  
32 <g id="edge2" class="edge">  
33 <title>tom&#45;&gt;dad</title>  
34 <path fill="none" stroke="red" d="M45.17,-35.29C47.  
  77,-41.04 50.27,-47.65 51.57,-54 53.19,-61.9 51.87,-  
  70.35 49.46,-78.01"/>  
35 <polygon fill="red" stroke="red" points="46.13,-76.  
  93 45.82,-87.52 52.67,-79.44 46.13,-76.93"/>  
36 <text text-anchor="middle" x="72.07" y="-57.8" font-  
  family="Times,serif" font-size="14.00"> &#160;0/0 &#  
  160;</text>  
37 </g>  
38 <!-- maia -->  
39 <g id="node3" class="node">  
40 <title>maia</title>  
41 <ellipse fill="none" stroke="black" cx="115.57" cy  
  ="-105" rx="33.29" ry="18"/>  
42 <text text-anchor="middle" x="115.57" y="-101.3"  
  font-family="Times,serif" font-size="14.00">maia</  
  text>  
43 </g>  
44 </g>  
45 </svg>  
46
```

```
1 <?xml version="1.0" encoding="UTF-8" standalone="no"
?
2 <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
3   "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
4 <!-- Generated by graphviz version 2.43.0 (0)
5   -->
6 <!-- Title: %3 Pages: 1 -->
7 <svg width="144pt" height="131pt"
8   viewBox="0.00 0.00 143.59 131.00" xmlns="http://www
9   .w3.org/2000/svg" xmlns:xlink="http://www.w3.org/
10  1999/xlink">
11 <g id="graph0" class="graph" transform="scale(1 1)
12  rotate(0) translate(4 127)">
13 <title>%3</title>
14 <polygon fill="white" stroke="transparent" points="-
15  4,4 -4,-127 139.59,-127 139.59,4 -4,4"/>
16 <!-- d -->
17 <g id="node1" class="node">
18 <title>d</title>
19 <ellipse fill="none" stroke="black" cx="36.59" cy="-
20  105" rx="27" ry="18"/>
21 <text text-anchor="middle" x="36.59" y="-101.3" font-
22  -family="Times,serif" font-size="14.00">d</text>
23 </g>
24 <!-- t -->
25 <g id="node2" class="node">
26 <title>t</title>
27 <ellipse fill="none" stroke="black" cx="36.59" cy="-
28  18" rx="27" ry="18"/>
29 <text text-anchor="middle" x="36.59" y="-14.3" font-
30  -family="Times,serif" font-size="14.00">t</text>
31 </g>
32 <!-- d&#45;&gt;t -->
33 <g id="edge1" class="edge">
34 <title>d&#45;&gt;t</title>
35 <path fill="none" stroke="black" d="M18.95,-91.35C12
36  .17,-85.42 5.2,-77.73 1.59,-69 -2.87,-58.25 2.94,-47
37  .52 11.03,-38.77"/>
38 <polygon fill="black" stroke="black" points="13.69,
39  -41.07 18.52,-31.64 8.86,-36 13.69,-41.07"/>
40 <text text-anchor="middle" x="26.59" y="-57.8" font-
```

```
29 family="Times,serif" font-size="14.00"> &#160;0/10  
  &#160;</text>  
30 </g>  
31 <!-- t-->  
32 <g id="edge2" class="edge">  
33 <title>t>d</title>  
34 <path fill="none" stroke="red" d="M45.19,-35.29C47.  
  78,-41.04 50.28,-47.65 51.59,-54 53.25,-62.11 51.81  
  ,-70.81 49.28,-78.63"/>  
35 <polygon fill="red" stroke="red" points="46.01,-77.  
  39 45.63,-87.97 52.53,-79.93 46.01,-77.39"/>  
36 <text text-anchor="middle" x="72.09" y="-57.8" font-  
  family="Times,serif" font-size="14.00"> &#160;0/0 &#  
  160;</text>  
37 </g>  
38 <!-- m -->  
39 <g id="node3" class="node">  
40 <title>m</title>  
41 <ellipse fill="none" stroke="black" cx="108.59" cy  
  ="-105" rx="27" ry="18"/>  
42 <text text-anchor="middle" x="108.59" y="-101.3"  
  font-family="Times,serif" font-size="14.00">m</text>  
43 </g>  
44 </g>  
45 </svg>  
46
```

ID	src	dest	amount	group	src_n	src_e	dest_n	dest_e	src_d	dest_d
2	0,04,20,5,0,	2161679203114375274630941557945232020251089312607308 7652383723408105063600070147761500936454570470862786 9702069833686361660030089751732511247633740543213460 3035445702142516535603778163693003610640441829541372 643100255683690006704986794499904312842155745102543 7279819137427553642625019821057158620227234339857381 3906769437847646651445567701079686709061678948557745 8637370869261774337704654931841775536224420241750390 3181827421441408785602795322818434551136750209831842 5250366913849245188288359620277590091100050269612943 3770297061845035322423193568722584809589006962060587 421954603201784497846158263582144177430642603,65537, 2454352605322268592008900243138707108080875397916828 9339650707414109582770834242173009539282426745682722 9880553467004005448679109437851439095238915265836858 7308477851944847288700266021218281412607086300444035 1914152540555901097499930235793743416545610368040032 5960557898468091402415007872965823676993425321804474 4604285155918762905223816899981984372615636383003006 9139123051200441111945407276884741033640730462395861 9851439941769916099206947465717047070945326989108443 5772741978366746154234093028750651579356970414938478 4131969168222943901917462927710491717935881329934336 642631167278369058626398091411958822782486021,65537, 4231860502610347555352698504423047563944256813822996 3925734038989576569874864579669508371562955146126547 8779552919144303233011283933461726857674426838187389 6721663847672076407494464781754007098969569658821095 1082558372250397158894945714319204774976709066583095 9656389449195952398626578010004019881519663179628663 1049344644422180019003905454847914081363056944472084 7689992964879750244948034517871603890465032993904165 5974782350016040380747036826804133887037575013624446 8012222270131756222483185448064466203435454227528122 7314387046825525570376824278684460211954070416261823 5709118874694072838664235842554512315661905, 1745164279842191683897870688775252929437856379494395 9644593481018317996812806135240585387347011403463919								

2	4843123602886715197650891859619406775380829934052483 5872485281606266439620861446034635607786292042673482 7642392974806673957359914994399933521690425914379137 2656087371539940176580242716026174273279118970901055 4280600760265229078672123452862613189683248956255155 8690301697229495263298457688434655931885453437842161 1289899532022728673678378918475624105043819026629471 6820800459652950177227887446325164917035367452040210 6924748077589683971300268994119580437205561936946700 600893560778543423518093933835790069197852673
3	1, 04, 13, 10, 0, 2161679203114375274630941557945232020251089312607308 7652383723408105063600070147761500936454570470862786 9702069833686361660030089751732511247633740543213460 3035445702142516535603778163693003610640441829541372 643100255683690006704986794499904312842155745102543 7279819137427553642625019821057158620227234339857381 3906769437847646651445567701079686709061678948557745 8637370869261774337704654931841775536224420241750390 3181827421441408785602795322818434551136750209831842 5250366913849245188288359620277590091100050269612943 3770297061845035322423193568722584809589006962060587 421954603201784497846158263582144177430642603, 65537, 2256586403724745983208556687990030736021636869740873 3883964851808410874820858287921659997954743444212625 8889014725245633691567258383521676090489827618604882 3446599370109121575718012944124229461713591222435833 0826328284855986303335327620511972006453142090664344 6610551234084941258572108217677529623697862636846419 0409612760214589978439512179632414787894743946995271 6058077163515372836183503361380147895905992145482259 1898567781991786261932874106056739293869626830428217 3136210695524457478834244215547988454805806522190171 0928067881317959628155388903833923070328443328378726 386495866615509251768955425927155295624588823, 65537, 4231860502610347555352698504423047563944256813822996 3925734038989576569874864579669508371562955146126547 8779552919144303233011283933461726857674426838187389 6721663847672076407494464781754007098969569658821095 1082558372250397158894945714319204774976709066583095

3	9656389449195952398626578010004019881519663179628663 104934464422180019003905454847914081363056944472084 7689992964879750244948034517871603890465032993904165 5974782350016040380747036826804133887037575013624446 8012222270131756222483185448064466203435454227528122 7314387046825525570376824278684460211954070416261823 5709118874694072838664235842554512315661905, 7754142821899275148672764486249827147291951463534258 3131191306090515724089556837205820094594019006617381 7870914386153115146267705082153309665964404530415177 7471873569870060864117925980999686814744354231846889 6990846846049836207197701758385280007526246224538969 0963824067563800165256859035674194291114312004167623 5694762226761908445600472087203109098947447060087682 640572923907398373569449250780454435298169779944004 8112763789825376382626011524853438555391048758769078 0789232355207322596895153032599004522731021331970408 1395654861368829855439175850787537877943284782407352 5835250217870849030986866114641442612783873
4	2,20,13,5,0, 2454352605322268592008900243138707108080875397916828 9339650707414109582770834242173009539282426745682722 9880553467004005448679109437851439095238915265836858 730847785194484728870026602121828141260708630044035 1914152540555901097499930235793743416545610368040032 5960557898468091402415007872965823676993425321804474 4604285155918762905223816899981984372615636383003006 9139123051200441111945407276884741033640730462395861 9851439941769916099206947465717047070945326989108443 5772741978366746154234093028750651579356970414938478 4131969168222943901917462927710491717935881329934336 642631167278369058626398091411958822782486021,65537, 2256586403724745983208556687990030736021636869740873 3883964851808410874820858287921659997954743444212625 8889014725245633691567258383521676090489827618604882 3446599370109121575718012944124229461713591222435833 0826328284855986303335327620511972006453142090664344 6610551234084941258572108217677529623697862636846419 0409612760214589978439512179632414787894743946995271 6058077163515372836183503361380147895905992145482259

4

1898567781991786261932874106056739293869626830428217  
313621069552445747834244215547988454805806522190171  
0928067881317959628155388903833923070328443328378726  
386495866615509251768955425927155295624588823, 65537,  
1745164279842191683897870688775252929437856379494395  
9644593481018317996812806135240585387347011403463919  
4843123602886715197650891859619406775380829934052483  
5872485281606266439620861446034635607786292042673482  
7642392974806673957359914994399933521690425914379137  
2656087371539940176580242716026174273279118970901055  
4280600760265229078672123452862613189683248956255155  
8690301697229495263298457688434655931885453437842161  
1289899532022728673678378918475624105043819026629471  
6820800459652950177227887446325164917035367452040210  
6924748077589683971300268994119580437205561936946700  
600893560778543423518093933835790069197852673,  
7754142821899275148672764486249827147291951463534258  
3131191306090515724089556837205820094594019006617381  
7870914386153115146267705082153309665964404530415177  
7471873569870060864117925980999686814744354231846889  
699084684604983620719770175838528007526246224538969  
0963824067563800165256859035674194291114312004167623  
5694762226761908445600472087203109098947447060087682  
6405729239073983735694492507804544435298169779944004  
8112763789825376382626011524853438555391048758769078  
0789232355207322596895153032599004522731021331970408  
1395654861368829855439175850787537877943284782407352  
5835250217870849030986866114641442612783873

5 3, 04, 20, 5, 1,

2161679203114375274630941557945232020251089312607308  
7652383723408105063600070147761500936454570470862786  
9702069833686361660030089751732511247633740543213460  
3035445702142516535603778163693003610640441829541372  
643100255683690006704986794499904312842155745102543  
7279819137427553642625019821057158620227234339857381  
3906769437847646651445567701079686709061678948557745  
8637370869261774337704654931841775536224420241750390  
3181827421441408785602795322818434551136750209831842  
5250366913849245188288359620277590091100050269612943  
3770297061845035322423193568722584809589006962060587

5	421954603201784497846158263582144177430642603, 65537, 2454352605322268592008900243138707108080875397916828 9339650707414109582770834242173009539282426745682722 9880553467004005448679109437851439095238915265836858 7308477851944847288700266021218281412607086300444035 1914152540555901097499930235793743416545610368040032 5960557898468091402415007872965823676993425321804474 4604285155918762905223816899981984372615636383003006 9139123051200441111945407276884741033640730462395861 9851439941769916099206947465717047070945326989108443 5772741978366746154234093028750651579356970414938478 4131969168222943901917462927710491717935881329934336 642631167278369058626398091411958822782486021, 65537, 4231860502610347555352698504423047563944256813822996 3925734038989576569874864579669508371562955146126547 8779552919144303233011283933461726857674426838187389 6721663847672076407494464781754007098969569658821095 1082558372250397158894945714319204774976709066583095 9656389449195952398626578010004019881519663179628663 104934464422180019003905454847914081363056944472084 7689992964879750244948034517871603890465032993904165 5974782350016040380747036826804133887037575013624446 8012222270131756222483185448064466203435454227528122 7314387046825525570376824278684460211954070416261823 570911887469407283864235842554512315661905, 1745164279842191683897870688775252929437856379494395 9644593481018317996812806135240585387347011403463919 4843123602886715197650891859619406775380829934052483 5872485281606266439620861446034635607786292042673482 7642392974806673957359914994399933521690425914379137 2656087371539940176580242716026174273279118970901055 4280600760265229078672123452862613189683248956255155 8690301697229495263298457688434655931885453437842161 1289899532022728673678378918475624105043819026629471 6820800459652950177227887446325164917035367452040210 6924748077589683971300268994119580437205561936946700 600893560778543423518093933835790069197852673
6	4, 04, 13, 10, 1, 2161679203114375274630941557945232020251089312607308 7652383723408105063600070147761500936454570470862786 9702069833686361660030089751732511247633740543213460

6

3035445702142516535603778163693003610640441829541372  
6431002556836900067049867944499904312842155745102543  
7279819137427553642625019821057158620227234339857381  
3906769437847646651445567701079686709061678948557745  
8637370869261774337704654931841775536224420241750390  
3181827421441408785602795322818434551136750209831842  
5250366913849245188288359620277590091100050269612943  
3770297061845035322423193568722584809589006962060587  
421954603201784497846158263582144177430642603, 65537,  
2256586403724745983208556687990030736021636869740873  
388396485180841087482085828792165999795474344212625  
8889014725245633691567258383521676090489827618604882  
3446599370109121575718012944124229461713591222435833  
0826328284855986303335327620511972006453142090664344  
6610551234084941258572108217677529623697862636846419  
0409612760214589978439512179632414787894743946995271  
6058077163515372836183503361380147895905992145482259  
1898567781991786261932874106056739293869626830428217  
313621069552445747834244215547988454805806522190171  
0928067881317959628155388903833923070328443328378726  
386495866615509251768955425927155295624588823, 65537,  
4231860502610347555352698504423047563944256813822996  
3925734038989576569874864579669508371562955146126547  
8779552919144303233011283933461726857674426838187389  
6721663847672076407494464781754007098969569658821095  
1082558372250397158894945714319204774976709066583095  
9656389449195952398626578010004019881519663179628663  
104934464422180019003905454847914081363056944472084  
7689992964879750244948034517871603890465032993904165  
5974782350016040380747036826804133887037575013624446  
8012222270131756222483185448064466203435454227528122  
7314387046825525570376824278684460211954070416261823  
5709118874694072838664235842554512315661905,  
7754142821899275148672764486249827147291951463534258  
3131191306090515724089556837205820094594019006617381  
7870914386153115146267705082153309665964404530415177  
7471873569870060864117925980999686814744354231846889  
6990846846049836207197701758385280007526246224538969  
096382406756380016525685903567419429114312004167623  
5694762226761908445600472087203109098947447060087682

6	640572923907398373569449250780454435298169779944004 8112763789825376382626011524853438555391048758769078 0789232355207322596895153032599004522731021331970408 1395654861368829855439175850787537877943284782407352 5835250217870849030986866114641442612783873
7	5, 20, 13, 5, 1, 2454352605322268592008900243138707108080875397916828 9339650707414109582770834242173009539282426745682722 9880553467004005448679109437851439095238915265836858 7308477851944847288700266021218281412607086300444035 1914152540555901097499930235793743416545610368040032 5960557898468091402415007872965823676993425321804474 4604285155918762905223816899981984372615636383003006 9139123051200441111945407276884741033640730462395861 9851439941769916099206947465717047070945326989108443 5772741978366746154234093028750651579356970414938478 4131969168222943901917462927710491717935881329934336 642631167278369058626398091411958822782486021, 65537, 2256586403724745983208556687990030736021636869740873 3883964851808410874820858287921659997954743444212625 8889014725245633691567258383521676090489827618604882 3446599370109121575718012944124229461713591222435833 0826328284855986303335327620511972006453142090664344 6610551234084941258572108217677529623697862636846419 0409612760214589978439512179632414787894743946995271 6058077163515372836183503361380147895905992145482259 1898567781991786261932874106056739293869626830428217 3136210695524457478834244215547988454805806522190171 0928067881317959628155388903833923070328443328378726 386495866615509251768955425927155295624588823, 65537, 1745164279842191683897870688775252929437856379494395 9644593481018317996812806135240585387347011403463919 4843123602886715197650891859619406775380829934052483 5872485281606266439620861446034635607786292042673482 7642392974806673957359914994399933521690425914379137 2656087371539940176580242716026174273279118970901055 4280600760265229078672123452862613189683248956255155 8690301697229495263298457688434655931885453437842161 1289899532022728673678378918475624105043819026629471 6820800459652950177227887446325164917035367452040210

7	6924748077589683971300268994119580437205561936946700 600893560778543423518093933835790069197852673, 7754142821899275148672764486249827147291951463534258 3131191306090515724089556837205820094594019006617381 7870914386153115146267705082153309665964404530415177 7471873569870060864117925980999686814744354231846889 6990846846049836207197701758385280007526246224538969 0963824067563800165256859035674194291114312004167623 5694762226761908445600472087203109098947447060087682 640572923907398373569449250780454435298169779944004 8112763789825376382626011524853438555391048758769078 0789232355207322596895153032599004522731021331970408 1395654861368829855439175850787537877943284782407352 5835250217870849030986866114641442612783873
8	6,04,20,5,2, 2161679203114375274630941557945232020251089312607308 7652383723408105063600070147761500936454570470862786 9702069833686361660030089751732511247633740543213460 3035445702142516535603778163693003610640441829541372 6431002556836900067049867944499904312842155745102543 7279819137427553642625019821057158620227234339857381 3906769437847646651445567701079686709061678948557745 8637370869261774337704654931841775536224420241750390 3181827421441408785602795322818434551136750209831842 5250366913849245188288359620277590091100050269612943 3770297061845035322423193568722584809589006962060587 421954603201784497846158263582144177430642603,65537, 2454352605322268592008900243138707108080875397916828 9339650707414109582770834242173009539282426745682722 9880553467004005448679109437851439095238915265836858 7308477851944847288700266021218281412607086300444035 1914152540555901097499930235793743416545610368040032 5960557898468091402415007872965823676993425321804474 4604285155918762905223816899981984372615636383003006 9139123051200441111945407276884741033640730462395861 9851439941769916099206947465717047070945326989108443 5772741978366746154234093028750651579356970414938478 4131969168222943901917462927710491717935881329934336 642631167278369058626398091411958822782486021,65537, 4231860502610347555352698504423047563944256813822996

8	3925734038989576569874864579669508371562955146126547 8779552919144303233011283933461726857674426838187389 6721663847672076407494464781754007098969569658821095 1082558372250397158894945714319204774976709066583095 9656389449195952398626578010004019881519663179628663 1049344644422180019003905454847914081363056944472084 7689992964879750244948034517871603890465032993904165 5974782350016040380747036826804133887037575013624446 8012222270131756222483185448064466203435454227528122 7314387046825525570376824278684460211954070416261823 5709118874694072838664235842554512315661905, 1745164279842191683897870688775252929437856379494395 9644593481018317996812806135240585387347011403463919 4843123602886715197650891859619406775380829934052483 5872485281606266439620861446034635607786292042673482 7642392974806673957359914994399933521690425914379137 2656087371539940176580242716026174273279118970901055 4280600760265229078672123452862613189683248956255155 8690301697229495263298457688434655931885453437842161 1289899532022728673678378918475624105043819026629471 6820800459652950177227887446325164917035367452040210 6924748077589683971300268994119580437205561936946700 600893560778543423518093933835790069197852673
9	7, 04, 13, 10, 2, 65537, 6216167920311437527463094155794523202025108931260730 8765238372340810506360007014776150093645457047086278 6970206983368636166003008975173251124763374054321346 0303544570214251653560377816369300361064044182954137 2643100255683690006704986794449990431284215574510254 3727981913742755364262501982105715862022723433985738 1390676943784764665144556770107968670906167894855774 5863737086926177433770465493184177553622442024175039 0318182742144140878560279532281843455113675020983184 2525036691384924518828835962027759009110005026961294 3377029706184503532242319356872258480958900696206058 74219546032017844978461582635821441774306426035537, 2256586403724745983208556687990030736021636869740873 3883964851808410874820858287921659997954743444212625 8889014725245633691567258383521676090489827618604882 3446599370109121575718012944124229461713591222435833

9

0826328284855986303335327620511972006453142090664344  
6610551234084941258572108217677529623697862636846419  
0409612760214589978439512179632414787894743946995271  
6058077163515372836183503361380147895905992145482259  
1898567781991786261932874106056739293869626830428217  
313621069552445747834244215547988454805806522190171  
0928067881317959628155388903833923070328443328378726  
386495866615509251768955425927155295624588823, 65537,  
4231860502610347555352698504423047563944256813822996  
3925734038989576569874864579669508371562955146126547  
8779552919144303233011283933461726857674426838187389  
6721663847672076407494464781754007098969569658821095  
1082558372250397158894945714319204774976709066583095  
9656389449195952398626578010004019881519663179628663  
1049344644422180019003905454847914081363056944472084  
7689992964879750244948034517871603890465032993904165  
5974782350016040380747036826804133887037575013624446  
8012222270131756222483185448064466203435454227528122  
7314387046825525570376824278684460211954070416261823  
5709118874694072838664235842554512315661905,  
7754142821899275148672764486249827147291951463534258  
3131191306090515724089556837205820094594019006617381  
7870914386153115146267705082153309665964404530415177  
7471873569870060864117925980999686814744354231846889  
6990846846049836207197701758385280007526246224538969  
0963824067563800165256859035674194291114312004167623  
5694762226761908445600472087203109098947447060087682  
6405729239073983735694492507804544435298169779944004  
8112763789825376382626011524853438555391048758769078  
0789232355207322596895153032599004522731021331970408  
1395654861368829855439175850787537877943284782407352  
5835250217870849030986866114641442612783873

10 8, 20, 13, 5, 2,

2454352605322268592008900243138707108080875397916828  
9339650707414109582770834242173009539282426745682722  
9880553467004005448679109437851439095238915265836858  
7308477851944847288700266021218281412607086300444035  
1914152540555901097499930235793743416545610368040032  
5960557898468091402415007872965823676993425321804474  
4604285155918762905223816899981984372615636383003006

10

9139123051200441111945407276884741033640730462395861  
9851439941769916099206947465717047070945326989108443  
5772741978366746154234093028750651579356970414938478  
4131969168222943901917462927710491717935881329934336  
642631167278369058626398091411958822782486021, 65537,  
2256586403724745983208556687990030736021636869740873  
388396485180841087482085828792165999795474344212625  
8889014725245633691567258383521676090489827618604882  
3446599370109121575718012944124229461713591222435833  
0826328284855986303335327620511972006453142090664344  
6610551234084941258572108217677529623697862636846419  
0409612760214589978439512179632414787894743946995271  
6058077163515372836183503361380147895905992145482259  
1898567781991786261932874106056739293869626830428217  
3136210695524457478834244215547988454805806522190171  
092806788131795962815538890383392307032843328378726  
386495866615509251768955425927155295624588823, 65537,  
1745164279842191683897870688775252929437856379494395  
9644593481018317996812806135240585387347011403463919  
4843123602886715197650891859619406775380829934052483  
5872485281606266439620861446034635607786292042673482  
7642392974806673957359914994399933521690425914379137  
2656087371539940176580242716026174273279118970901055  
4280600760265229078672123452862613189683248956255155  
8690301697229495263298457688434655931885453437842161  
1289899532022728673678378918475624105043819026629471  
6820800459652950177227887446325164917035367452040210  
6924748077589683971300268994119580437205561936946700  
600893560778543423518093933835790069197852673,  
7754142821899275148672764486249827147291951463534258  
3131191306090515724089556837205820094594019006617381  
7870914386153115146267705082153309665964404530415177  
7471873569870060864117925980999686814744354231846889  
6990846846049836207197701758385280007526246224538969  
0963824067563800165256859035674194291114312004167623  
5694762226761908445600472087203109098947447060087682  
6405729239073983735694492507804544435298169779944004  
81127637898253763826011524853438555391048758769078  
0789232355207322596895153032599004522731021331970408  
1395654861368829855439175850787537877943284782407352

10 5835250217870849030986866114641442612783873

11

```

1 # coding=utf-8
2 import unittest
3
4 import src.simplify.flow_graph
5 import src.simplify.graph_objects
6 from src.crypto import keys
7 from src.transactions.ledger import *
8 from src.transactions.transaction import
    VerificationError
9
10
11 def setUpModule():
12     print(os.getcwd())
13     os.chdir("/home/tcassar/projects/settle")
14
15
16 def key_path(usr: str, keytype="private") -> str:
17     assert usr == "d" or usr == "m" or usr == "t"
18     return f"./src/crypto/sample_keys/{usr}_{keytype}
    }-key.pem' if keytype == 'private' else ''}"
19
20
21 class TestLedger(unittest.TestCase):
22     def setUp(self) -> None:
23
        """Load in sample data from mock db"""
24
25
26     self.valid: Ledger
27     self.missing_key: Ledger
28     self.invalid: Ledger
29
30     # load in raw copies of d, m, t test keys
31     ldr = keys.RSAKeyLoader()
32     self.d_m_t_keys: dict[int, keys.
        RSAPrivateKey] = {}
33     pub: list[keys.RSAPublicKey] = []
34
35     for person, id in zip(["d", "m", "t"], [4,
36         13, 20]):
37         ldr.load(key_path(person))
            ldr.parse()

```

```

38             self.d_m_t_keys[id] = keys.RSAPrivateKey
39             ldr)
40             pubs.append(keys.RSAPublicKey(ldr))
41             self.d_pub, self.m_pub, self.t_pub = pubs
42             self.d_priv, self.m_priv, self.t_priv = self
43             .d_m_t_keys.values()
44             self.valid, self.missing_key, self.invalid
45             = LedgerLoader.load_from_csv(
46                 "./tests/test_transactions/mock_db.csv"
47             )
48             def test_add(self):
49                 """test adding transactions to a ledger"""
50
51                 with self.subTest("Add"):
52                     ledger = Ledger()
53                     self.assertFalse(not not ledger)
54                     ledger.append(Transaction(0, 0, 0, self.
55                     d_priv, self.m_priv))
56                     self.assertTrue(not not ledger)
57
58                 with self.subTest("Catch non transaction"),
59                 self.assertRaises(LedgerBuildError):
60                     ledger.append(6) # type: ignore
61
62             def test_load_from_csv(self):
63                 """Test loading from mock db works as
64                 expected"""
65                 ledger_list = LedgerLoader.load_from_csv(
66                     "./tests/test_transactions/mock_db.csv"
67                 )
68                 self.assertEqual(len(ledger_list), 3)
69
70             def test_verify_transactions(self):
71                 """
72                     Make three ledgers:
73                     1) Valid transactions
74                     2) An unsigned transaction
75                     3) An invalid signature

```

```
73
74         Check that first one goes through no issues
75         , and that other two are caught
76         """
77
78         with self.subTest("unsigned"), self.
79             assertRaises(VerificationError):
80                 self.valid._verify_transactions()
81
82
83         with self.subTest("signed"):
84             self.valid._verify_transactions()
85
86         with self.subTest("missing key"), self.
87             assertRaises(VerificationError):
88                 self.missing_key._verify_transactions()
89
90         with self.subTest("invalid key"), self.
91             assertRaises(VerificationError):
92                 self.invalid._verify_transactions()
93
94     def test_as_flow(self):
95         """Test that we build flow graphs from
96         ledgers as expected"""
97         # sign ledger
98         self.sign()
99
100        Vertex = src.simplify.graph_objects.Vertex
101
102        exp = src.simplify.flow_graph.FlowGraph(
103            [Vertex(ID=4), Vertex(ID=13), Vertex(ID
104            =20)])
105
106        d, m, t = exp.nodes()
107        exp.add_edge(d, (m, 10), (t, 5))
108        exp.add_edge(t, (m, 5))
109
110        as_flow = self.valid._as_flow()
```

```

107
108         with self.subTest("nodes"):
109             self.assertEqual(exp.nodes(), self.
110                           valid.nodes)
111
112         with self.subTest("to flow graph"):
113             self.assertEqual(exp, as_flow)
114
115     def sign(self):
116         """Checks that signing a ledger is a
117         reliable process"""
118         for trn in self.valid.ledger:
119             trn.sign(self.d_m_t_keys[trn.src],
120                      origin="src")
121             trn.sign(self.d_m_t_keys[trn.dest],
122                      origin="dest")
123             print("verifying...")
124             trn.verify()
125             print("verified")
126
127     def test_flow_to_transactions(self):
128         """Checks that flow graph data structures
129         can be effectively converted to ledgers"""
130         self.maxDiff = None
131         self.sign()
132         as_flow = self.valid._as_flow()
133
134         # remove sigs, ID, for comparison
135         trn: Transaction
136         for trn in self.valid.ledger:
137             trn.ID = 0
138             trn.signatures = {}
139
140         with self.subTest("to transactions"):
141             self.valid.ledger.sort(key=lambda trn:
142                                   trn.amount)
143             calc: list[Transaction] = self.valid.
144             _flow_to_transactions(as_flow)
145             calc.sort(key=lambda trn: trn.amount)
146
147             self.assertEqual(calc, self.valid.

```

```
140 ledger)
141
142     def test_simplify_ledger(self):
143         """Integration test, checks that calling
144             simplify() on a ledger displays expected behaviour
145         """
146
147         self.sign()
148         self.valid.simplify_ledger()
149
150         trn = Transaction(4, 13, 15, self.d_pub,
151                           self.m_pub)
152
153         self.assertEqual(self.valid.ledger, [trn])
```



File - /home/tcassar/projects/settle/tests/test\_transactions/test\_transaction.py

```
1 # coding=utf-8
2 import os
3 import unittest
4
5 from src.transactions.transaction import *
6
7
8 class TestTransaction(unittest.TestCase):
9     def setUp(self) -> None:
10         # Load keys
11         os.chdir("/home/tcassar/projects/settle/src")
12         ldr = keys.RSAKeyLoader()
13         ldr.load("./crypto/sample_keys/d_private-key.pem")
14         ldr.parse()
15
16         self.key = keys.RSAPrivateKey(ldr)
17         self.pub_key = keys.RSAPublicKey(ldr)
18
19         self.trn = Transaction(0, 0, 0, self.pub_key,
20                               self.pub_key)
21         self.trn.time = 0 # type: ignore
22
23     def test_hash(self):
24         """Tests that calling .hash() on a
25         Transaction displays expected behaviour"""
26         self.assertEqual(
27             self.trn.hash(),
28             b"\xb9\x8f\xe6\xde\x08\xd4\x1f\xdd\x01\x
29             \xa3\xb4\xc5\x1d\x98\xd4\xac\x1b\x02\xd3\x
30             \xa0\x01!\xf3\x99\xdf\xd5\\x85",
31             )
32         self.trn.time = 1
33         self.assertNotEqual(
34             self.trn.hash(),
35             b"\xb9\x8f\xe6\xde\x08\xd4\x1f\xdd\x01\x
36             \xa3\xb4\xc5\x1d\x98\xd4\xac\x1b\x02\xd3\x
37             \xa0\x01!\xf3\x99\xdf\xd5\\x85",
38             )
```

```

34     def test_sign(self):
35         """
36             Checks that signing a Transaction displays
37             expected behaviour
38             Working on assumption that rsa.Notary is
39             working; tested in settle/tests/test_crypto"""
40
41         self.trn.sign(self.key, origin="src")
42
43         with self.subTest("catch invalid origin"),
44             self.assertRaises(ValueError):
45                 self.trn.sign(self.key, origin="no")
46
47         with self.subTest("invalid key types"), self.
48             .assertRaises(TransactionError):
49                 self.trn.sign(12, origin="src")
50                 self.trn.sign(self.pub_key, origin="dest")
51
52         with self.subTest("sig_overwrite"), self.
53             .assertRaises(TransactionError):
54                 self.trn.sign(self.key, origin="src")
55
56         with self.subTest("Right sig"):
57             self.assertEqual(
58                 b"\xc1~\xcb\u\xec\x01E*\xf2;0\xe3\
59                 \xf3\x08x\xee\x84\xfc\xe1\xca\x8b\xa2\xed\xec\x9b\
60                 \xfd\xe5$7\x88n\xb7\x86\xfc~\x98\x91\x80Z\xcd\x1e\xf5\
61                 }\xc2<JS\xefY\xe5UW\xfc\x0e\xd2\xbe\xc9\xea\xfbZm\
62                 \xf2\xa7\x08\x8d\x05\x16\xe4@\xf40\x03I\xca\xbc.\x95\
63                 \xbb\xd3\n\xfd9\xb0Wk\xf4\x03\x96\xdbF\xcd\xa0E\xd1\
64                 \xac\xa6(\xb9\x92\xdb\x841\xe0U"\xe4\x7f\xeb U\xc9Z\
65                 \xe5\xf6\x19\xc1Wn\x17&\x17\x84Dt\xb6Z\xc0\x02\x85\
66                 \xf3\xd3\xd5\x98t7\xcd_\xfc\xa7\\xa7t\xe8\xb1\xafL\
67                 \x05\xb3\x07a\xd0jr\xc4}\xd8\xb58\xf40\x9f\x02\xa2\
68                 \xe6?3B\xf6\xe1\xe6\xb2\x02d\xfd\xd5\x83\xcf\xcc\xb6\
69                 \x06m\xc2p\xd7\x00@\\x93H\xc6]\x0c\x98\x1e\xdf\xda\x00\
70                 \x86\xd7\xec\xc7\x10\x025eiry\xd6\x80\xfe\xe2+\xb8\
71                 \x1dn\noB\xc0\x8a\xc8t\xbfg\xbb\xca(Aj\x85\xeb\x94\
72                 \x0c-\xacr'\xfe\n^Z\x13\x80'\xaa\x96{\xf8x\xce\xd6\
73                 \x90",

```

```
54                     self.trn.signatures[self.trn.src],  
55                 )  
56  
57     def test_verify(self):  
58         """Checks that we are able to verify  
transactions"""  
59         # check that we are complained at if no  
valid keys are passed in  
60         with self.subTest("catch invalid keys"),  
self.assertRaises(VerificationError):  
61             self.trn.verify()  
62             self.trn.verify() # wrong type # type  
: ignore  
63  
64         # check works with priv and public keys  
65  
66         with self.subTest("good verif"):  
67             self.trn.sign(self.key, origin="src")  
68             self.trn.verify()  
69  
70         with self.subTest("priv/pub keys"):  
71             self.trn.verify()  
72             self.trn.verify()  
73  
74         with self.subTest("verify src, dest"):  
75             self.trn.verify()  
76             self.trn.verify()  
77  
78         with self.subTest("verify >1 param"):  
79             self.trn.verify()  
80  
81         with self.subTest("bad key"), self.  
assertRaises(VerificationError):  
82             # edit pub key, thus should fail  
83             self.pub_key.lookup["n"] = 3  
84             self.trn.verify()  
85
```



File - /home/tcassar/projects/settle/tests/test\_transactions/graph\_renders/settled0

```
1 digraph { 4 -> 13 [label=" 0/15  "]
2 13 -> 4 [label=" 0/0  ", color=red]
3 20
4 }
5
```



```
1 digraph { 4 -> 13 [label=" 0/10  "]
2 4 -> 20 [label=" 0/5   "]
3 13 -> 4 [label=" 0/0   ", color=red]
4 13 -> 20 [label=" 0/0   ", color=red]
5 20 -> 13 [label=" 0/5   "]
6 20 -> 4 [label=" 0/0   ", color=red]
7 }
8
```



```

1 <?xml version="1.0" encoding="UTF-8" standalone="no"
?
2 <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
3   "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
4 <!-- Generated by graphviz version 2.43.0 (0)
5   -->
6 <!-- Title: %3 Pages: 1 -->
7 <svg width="144pt" height="131pt"
8   viewBox="0.00 0.00 143.59 131.00" xmlns="http
9   ://www.w3.org/2000/svg">
9 <g id="graph0" class="graph" transform="scale(1 1)
10 rotate(0) translate(4 127)">
10 <title>%3</title>
11 <polygon fill="white" stroke="transparent" points="-
12   4,4 -4,-127 139.59,-127 139.59,4 -4,4"/>
12 <!-- 4 -->
13 <g id="node1" class="node">
14 <title>4</title>
15 <ellipse fill="none" stroke="black" cx="36.59" cy="-
16   105" rx="27" ry="18"/>
16 <text text-anchor="middle" x="36.59" y="-101.3" font
17   -family="Times,serif" font-size="14.00">4</text>
17 </g>
18 <!-- 13 -->
19 <g id="node2" class="node">
20 <title>13</title>
21 <ellipse fill="none" stroke="black" cx="36.59" cy="-
22   18" rx="27" ry="18"/>
22 <text text-anchor="middle" x="36.59" y="-14.3" font-
23   family="Times,serif" font-size="14.00">13</text>
23 </g>
24 <!-- 4&#45;&gt;13 -->
25 <g id="edge1" class="edge">
26 <title>4&#45;&gt;13</title>
27 <path fill="none" stroke="black" d="M18.95,-91.35C12
28   .17,-85.42 5.2,-77.73 1.59,-69 -2.87,-58.25 2.94,-47
29   .52 11.03,-38.77"/>
29 <polygon fill="black" stroke="black" points="13.69, -
30   41.07 18.52,-31.64 8.86,-36 13.69,-41.07"/>
30 <text text-anchor="middle" x="26.59" y="-57.8" font-
31   family="Times,serif" font-size="14.00"> &#160;0/15

```

```
29 &#160;</text>
30 </g>
31 <!-- 13-->
32 <g id="edge2" class="edge">
33 <title>13</title>
34 <path fill="none" stroke="red" d="M45.19,-35.29C47.
    78,-41.04 50.28,-47.65 51.59,-54 53.25,-62.11 51.81
    ,-70.81 49.28,-78.63"/>
35 <polygon fill="red" stroke="red" points="46.01,-77.
    39 45.63,-87.97 52.53,-79.93 46.01,-77.39"/>
36 <text text-anchor="middle" x="72.09" y="-57.8" font-
    family="Times,serif" font-size="14.00"> &#160;0/0 &#
    160;</text>
37 </g>
38 <!-- 20 -->
39 <g id="node3" class="node">
40 <title>20</title>
41 <ellipse fill="none" stroke="black" cx="108.59" cy
    ="-105" rx="27" ry="18"/>
42 <text text-anchor="middle" x="108.59" y="-101.3"
    font-family="Times,serif" font-size="14.00">20</text>
    >
43 </g>
44 </g>
45 </svg>
46
```

```

1 <?xml version="1.0" encoding="UTF-8" standalone="no"
?
2 <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
3   "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
4 <!-- Generated by graphviz version 2.43.0 (0)
5   -->
6 <!-- Title: %3 Pages: 1 -->
7 <svg width="221pt" height="218pt"
8   viewBox="0.00 0.00 221.18 218.00" xmlns="http
9   ://www.w3.org/2000/svg">
9 <g id="graph0" class="graph" transform="scale(1 1)
10 rotate(0) translate(4 214)">
10 <title>%3</title>
11 <polygon fill="white" stroke="transparent" points="-
12   4,4 -4,-214 217.18,-214 217.18,4 -4,4"/>
12 <!-- 4 -->
13 <g id="node1" class="node">
14 <title>4</title>
15 <ellipse fill="none" stroke="black" cx="103.18" cy
16   ="-192" rx="27" ry="18"/>
16 <text text-anchor="middle" x="103.18" y="-188.3"
17   font-family="Times,serif" font-size="14.00">4</text>
17 </g>
18 <!-- 13 -->
19 <g id="node2" class="node">
20 <title>13</title>
21 <ellipse fill="none" stroke="black" cx="39.18" cy="-
22   105" rx="27" ry="18"/>
22 <text text-anchor="middle" x="39.18" y="-101.3" font
23   -family="Times,serif" font-size="14.00">13</text>
23 </g>
24 <!-- 4&#45;&gt;13 -->
25 <g id="edge1" class="edge">
26 <title>4&#45;&gt;13</title>
27 <path fill="none" stroke="black" d="M76.01,-189.
28   76C52.42,-187.08 19.57,-179.1 3.18,-156 -4.12,-145.
29   71 2.94,-134.29 12.69,-124.96"/>
29 <polygon fill="black" stroke="black" points="15.15,
30   127.46 20.47,-118.29 10.6,-122.15 15.15,-127.46"/>
30 <text text-anchor="middle" x="28.18" y="-144.8" font
31   -family="Times,serif" font-size="14.00"> &#160;0/10
31

```

```

29 &#160;</text>
30 </g>
31 <!-- 20 -->
32 <g id="node3" class="node">
33 <title>20</title>
34 <ellipse fill="none" stroke="black" cx="103.18" cy
   ="-18" rx="27" ry="18"/>
35 <text text-anchor="middle" x="103.18" y="-14.3" font
   -family="Times,serif" font-size="14.00">20</text>
36 </g>
37 <!-- 4&#45;&gt;20 -->
38 <g id="edge2" class="edge">
39 <title>4&#45;&gt;20</title>
40 <path fill="none" stroke="black" d="M111.21,-174.
  66C113.63,-168.9 115.96,-162.3 117.18,-156 125.75,-
  111.48 125.75,-98.52 117.18,-54 116.59,-50.95 115.74
  ,-47.83 114.74,-44.76"/>
41 <polygon fill="black" stroke="black" points="118,-43
  .48 111.21,-35.34 111.44,-45.93 118,-43.48"/>
42 <text text-anchor="middle" x="143.68" y="-101.3"
   font-family="Times,serif" font-size="14.00"> &#160;0
  /5 &#160;</text>
43 </g>
44 <!-- 13&#45;&gt;4 -->
45 <g id="edge3" class="edge">
46 <title>13&#45;&gt;4</title>
47 <path fill="none" stroke="red" d="M50.43,-121.38C57.
  72,-131.3 67.45,-144.45 76.18,-156 79.01,-159.75 82.
  04,-163.71 85,-167.57"/>
48 <polygon fill="red" stroke="red" points="82.4,-169.
  94 91.28,-175.72 87.95,-165.66 82.4,-169.94"/>
49 <text text-anchor="middle" x="96.68" y="-144.8" font
   -family="Times,serif" font-size="14.00"> &#160;0/0
  &#160;</text>
50 </g>
51 <!-- 13&#45;&gt;20 -->
52 <g id="edge4" class="edge">
53 <title>13&#45;&gt;20</title>
54 <path fill="none" stroke="red" d="M23.16,-89.95C13.
  42,-79.67 4.24,-65.55 12.18,-54 24.33,-36.3 46.8,-27
  .54 66.36,-23.22"/>

```

```
55 <polygon fill="red" stroke="red" points="67.09,-26.  
64 76.26,-21.34 65.78,-19.77 67.09,-26.64"/>  
56 <text text-anchor="middle" x="32.68" y="-57.8" font-  
family="Times,serif" font-size="14.00"> &#160;0/0 &  
160;</text>  
57 </g>  
58 <!-- 20&#45;&gt;4 -->  
59 <g id="edge6" class="edge">  
60 <title>20&#45;&gt;4</title>  
61 <path fill="none" stroke="red" d="M121.45,-31.67C137  
.19,-43.79 158.81,-63.66 168.18,-87 174.13,-101.85  
174.13,-108.15 168.18,-123 160.35,-142.51 143.96,-  
159.6 129.59,-171.76"/>  
62 <polygon fill="red" stroke="red" points="127.03,-169  
.33 121.45,-178.33 131.43,-174.78 127.03,-169.33"/>  
63 <text text-anchor="middle" x="192.68" y="-101.3"  
font-family="Times,serif" font-size="14.00"> &#160;0  
/0 &#160;</text>  
64 </g>  
65 <!-- 20&#45;&gt;13 -->  
66 <g id="edge5" class="edge">  
67 <title>20&#45;&gt;13</title>  
68 <path fill="none" stroke="black" d="M91.28,-34.28C86  
.55,-40.39 81.08,-47.51 76.18,-54 69.7,-62.57 62.67  
, -72.03 56.5,-80.39"/>  
69 <polygon fill="black" stroke="black" points="53.55,-  
78.5 50.43,-88.62 59.18,-82.65 53.55,-78.5"/>  
70 <text text-anchor="middle" x="96.68" y="-57.8" font-  
family="Times,serif" font-size="14.00"> &#160;0/5 &  
160;</text>  
71 </g>  
72 </g>  
73 </svg>  
74
```