```python
# coding=utf-8

"""
Unit tests for key handling
"""

import os
from unittest import TestCase

from src.crypto import keys


class TestRSAKeyLoading(TestCase):
    """Tests for RSA Keys"""

    def setUp(self) -> None:
        """Initialise loaders fresh between tests"""
        self.loader = keys.RSAKeyLoader()
        os.chdir("/home/tcassar/projects/settle/src")
        print(os.system("pwd"))
        self.key_path = "./crypto/sample_keys/d_private-key.pem"
        self.pub_key_path = "./crypto/sample_keys/d_public-key.pe"

    def test_file_loading(self):
        """Tests that file is being loaded correctly assuming correct file"""

        expected_start = "modulus"
        # test assumes if it starts off fine it will continue being fine
        received_start = self.loader.load(self.key_path)
        self.assertEqual(expected_start, received_start[:7])

    def test_parsing(self):
        loader = self.loader

        loader.load(self.key_path)
        loader.parse()

        # compare parsed file to known values from test keys.
        # lay out known values
        k_e = 65537

        k_d = 42318605026103475553526985044230475639442568138229963925734038989576569874864579669508371562955146126547877955291914430323300
```

```
42
   112839334617268576744268381873896721663847672076407494464781754007098969569658821095108255837225039715889494571431920477497670906658309596563894491959523986265780100040198815196631796286631049344644422180019003905454847914081363056944472084768999296487975024494803451787160389046503299390416559747823500160403807470368268041338870375750136244468012222270131756222483185448064466203435454227528122731438704682552557037682427868446021195407041626182357091188746940728386642358425545123156619054320          k_n =
   216167920311437527463094155794523202025108931260730876523837234081050636000701477615009364545704708627869702069833686361660030089751732511247633740543213460303544570214251653560377816369300361064044182954137264310025568369000670498679444999043128421557451025437279819137427553642625019821057158620227234339857381390676943784764665144556770107968670906167894855774586373708692617743377046549318417755362244202417503903181827421441408785602795322818434551136750209831842525036691384924518828835962027759009110005026961294337702970618450353224231935687225848095890069620605874219546032017844978461582635821441774306426034
44
45          # build lists for testing
46          cases = [
47              "pub_exp",
48              "priv_exp",
49              "mod",
50              "prime1",
51              "prime2",
52              "exponent1",
53              "exponent2",
54              "coefficient",
55          ]
56
57          known = [k_e, k_d, k_n]
58
59          key = keys.RSAPrivateKey(loader)
60
61          received = [
62              key.e,
63              key.d,
64              key.n,
65              key.p,
66              key.q,
67              key.exp1,
68              key.exp2,
69              key.crt_coef,
70          ]
71
```

```
 72         for test, known_val, rec_val in zip(cases, known,
    received):
 73             with self.subTest(test):
 74                 self.assertEqual(rec_val, known_val)
 75                 self.assertEqual(type(rec_val), type(
    known_val))
 76
 77     def test_file_not_found(self):
 78         path = "./adsads"
 79
 80         with self.assertRaises(keys.RSAParserError):
 81             loader = self.loader
 82             loader.load(path)
 83
 84     def test_wrong_format(self):
 85         """Checks that we can deal with files being the wrong
     format"""
 86         with self.assertRaises(keys.RSAParserError):
 87             self.loader.load("./crypto/sample_keys/d_public-
    key.pe")
 88
 89     def test_unparsed_key(self):
 90         """Check accessing attributes"""
 91         loader = self.loader
 92         loader.load(self.key_path)
 93         key = keys.RSAPrivateKey(loader)
 94
 95         with self.assertRaises(keys.RSAKeyError):
 96             _ = key.asdf
 97
 98     def test_unloaded_key(self):
 99         ldr = self.loader
100         with self.assertRaises(keys.RSAParserError):
101             ldr.parse()
102
103     def test_public_key(self):
104         ldr = self.loader
105         ldr.load(self.key_path)
106         ldr.parse()
107
108         pub_key = keys.RSAPublicKey(ldr)
109
110         with self.subTest("allowed access"):
111             self.assertEqual(pub_key.e, 65537)
112
113         with (self.subTest("deny access"), self.assertRaises(
    keys.RSAPublicKeyError)):
114             _ = pub_key.p
```

```python
# coding=utf-8
import hashlib
import sys
from unittest import TestCase

from src.crypto import hashes


class TestHash(TestCase):
    """
    Test hash interfaces (functionality not aim of testing as functionality comes from hashlib
    1) Check initialisation as expected
    2) Check update
    3) Check digest
    4) Check hexdigest == intdigest
    """

    def test_init(self) -> None:
        """Checks that _hasher is initialised correctly i.e. no strange start values"""
        h = hashes.Hasher()
        self.assertEqual(
            h.digest().int_digest(),
            int.from_bytes(hashlib.sha3_256(b"").digest(), byteorder=sys.byteorder),
        )  # should be initialised empty

    def test_hash(self) -> None:
        """tests that hash object can validate hash looking numbers"""
        with self.subTest("valid"):
            h_val = hashlib.sha3_256(b"1234").digest()
            h = hashes.Hash(h_val)  # create a definitely valid hash

        with self.subTest("too short"):
            with self.assertRaises(hashes.HashError):
                hashes.Hash(b"12")

        with self.subTest("too long"):
            with self.assertRaises(hashes.HashError):
                hashes.Hash(
                    b"115792089237316195423570985008687907853269984665640564039457584007913129639936O"
                )

```

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

File - /home/tcassar/projects/settle/tests/test_crypto/test_sign_verify.py

```python
# coding=utf-8

"""
Testing sign / verify through RSA working as expected
"""
import os
from unittest import TestCase

from src.crypto import keys
from src.crypto import rsa
from src.transactions.transaction import Signable


def setUpModule():
    os.chdir("/home/tcassar/projects/settle/src")


class TestRSA(TestCase):
    """Just tests RSA parts"""

    def setUp(self) -> None:
        # load keys
        ldr = keys.RSAKeyLoader()
        ldr.load("./crypto/sample_keys/d_private-key.pem")
        ldr.parse()

        # build public and private
        self.private = keys.RSAPrivateKey(ldr)
        self.public = keys.RSAPublicKey(ldr)

    def test_encryption(self):
        """Checks to see if process is reversible, and encrypted is different to how it started"""
        message = " | maia"
        m_bytes = bytes(message, encoding="utf8")
        encrypted = rsa.RSA.encrypt(m_bytes, self.public)
        # TODO: actually fix byte overflow affair
        decrypted = rsa.RSA.bytes_to_str(rsa.RSA.naive_decrypt(encrypted, self.private))

        with self.subTest("Catch Public Key"):
            with self.assertRaises(rsa.DecryptionError):
                _ = rsa.RSA.naive_decrypt(encrypted, self.public)  # type: ignore

        with self.subTest("encrypted"):
            self.assertNotEqual(m_bytes, encrypted)

```

```
46          with self.subTest("Successful decryption"):
47              self.assertEqual(message, decrypted)
48
49      def test_RSA_sign(self):
50          """Checks for consistent creating / verifying of a '
    signature'"""
51
52          message = " | maia"
53          m_bytes = message.encode("utf8")
54          sig: bytes = rsa.RSA.sign(m_bytes, self.private)
55          de_sign: bytes = rsa.RSA.inv_sig(sig, self.public)
56
57          self.assertEqual(m_bytes, de_sign)
58
```

```python
# coding=utf-8
from unittest import TestCase

from src.simplify.base_graph import Digraph
from src.simplify.flow_graph import FlowGraph
from src.simplify.graph_objects import Vertex
from src.simplify.path import Path, prev_map, disc_map, BFSQueue
from src.simplify.weighted_digraph import WeightedDigraph


class TestPath(TestCase):
    """Tests searching for BFS"""

    def setUp(self):
        # make a graph with 5 nodes

        labels = ["a", "b", "c", "d", "e", "f"]
        self.vertices = [Vertex(ID, label=label) for ID, label in enumerate(labels)]

        self.g = Digraph(self.vertices)
        a, b, c, d, e, f = self.vertices
        self.g.add_edge(a, b, c)
        self.g.add_edge(b, d)
        self.g.add_edge(d, f)
        self.g.add_edge(c, e)
        self.g.add_edge(d, f)
        self.g.add_edge(e, f, b)

        # set up weighted_graph and flow graphs
        self.weighted_graph = WeightedDigraph(self.vertices)
        self.flow_graph = FlowGraph(self.vertices)

        for g in [self.weighted_graph, self.flow_graph]:
            g.add_edge(a, (b, 10), (c, 10))
            g.add_edge(b, (d, 25))
            g.add_edge(c, (e, 25))
            g.add_edge(d, (f, 10))
            g.add_edge(e, (f, 10), (b, 6))

    def test_shortest_path(self):
        """
        Checks that we can in fact find shortest path along
        """

        a, b, c, d, e, f = self.vertices

```

```
47          cases = ["digraph", "weighted_graph", "flow"]
48
49          # check precalulated shortest path
50          graph_cases = [self.g, self.weighted_graph, self.
    flow_graph]
51
52          for case, g in zip(cases, graph_cases):
53              with self.subTest(case):
54                  calc_shorted: list[Vertex] = Path.
    shortest_path(
55                      self.g, a, f, self.g.neighbours
56                  )
57                  expected: list[Vertex] = [a, c, e, f]
58                  self.assertEqual(expected, calc_shorted)
59
60      def test_build_path(self):
61          """Given a prev_map, check we build the right path"""
62
63          # generate vertices, unpack into vars
64          vertices = []
65          for ID, name in enumerate(["A", "B", "C", "D", "E", "F
    "]):
66              vertices.append(Vertex(ID, name))
67
68          a, c, b, d, e, f = vertices
69
70          # build a prev map. for context, path is A -> B -> D
     -> F
71          prev: prev_map = {a: None, b: a, c: a, d: b, e: c, f:
    d}
72          expected = [a, b, d, f]
73
74          self.assertEqual(expected, Path._build_path(prev, f))
75
76      def test_recursive_bfs(self):
77          """Check that BFS is finding paths along graph
    correctly"""
78          graph = self.flow_graph
79          a, b, c, d, e, f = graph.nodes()
80
81          expected: prev_map = {a: None, b: a, c: a, d: b, e: c
    , f: e}
82
83          # set up structures for BFS
84          # create queue, discovered list, previous list
85          queue = BFSQueue(next(iter(graph.graph)))
86
87          discovered: disc_map = {node: False for node in graph.
```

```
 87 nodes()}
 88         prev: prev_map = {node: None for node in graph.nodes
    ()}
 89
 90         self.assertEqual(
 91             expected,
 92             Path.BFS(
 93                 graph=graph,
 94                 queue=queue,
 95                 discovered=discovered,
 96                 target=None,
 97                 previous=prev,
 98                 neighbours=graph.neighbours,
 99             ),
100         )
101
102     def test_find_target(self):
103         graph = self.flow_graph
104         a, b, c, *_ = graph.nodes()
105
106         expected = {node: None for node in graph.nodes()}
107         expected[b] = a
108         expected[c] = a
109
110         queue, discovered, previous = Path.build_bfs_structs(
    graph, a)
111
112         calculated = Path.BFS(
113             graph=graph,
114             queue=queue,
115             discovered=discovered,
116             previous=previous,
117             target=b,
118             neighbours=graph.neighbours,
119         )
120         self.assertEqual(expected, calculated)
121
122     def test_build_bfs_struct(self):
123         with self.subTest("with initial value"):
124             queue, disc, prev = Path.build_bfs_structs(
125                 self.flow_graph, self.vertices[0]
126             )
127             self.assertEqual(queue, BFSQueue(self.vertices[0
    ]))
128
129         with self.subTest("with initial value"):
130             queue, disc, prev = Path.build_bfs_structs(self.
    flow_graph)
```

```
131             self.assertEqual(queue, BFSQueue())
132
```

File - /home/tcassar/projects/settle/tests/test_settling/test_flow.py

```python
# coding=utf-8
from unittest import TestCase

from src.simplify.flow_algorithms import NoOptimisations, MaxFlow, Simplify
from src.simplify.flow_graph import *
from src.simplify.graph_objects import Vertex


class TestFlowEdge(TestCase):
    """Tests for FlowEdge"""

    def setUp(self) -> None:
        self.edges = [FlowEdge(Vertex(0), Vertex(1), 5 * n) for n in range(2)]
        self.edges[0].flow = -3  # => unused capacity should be three

    def test_unused_capacity(self):
        """builds two edges, residual and non-residual
        non-residual has capacity 5"""

        for edge, cap in zip(self.edges, [3, 5]):
            with self.subTest(edge):
                self.assertEqual(edge.unused_capacity(), cap)

    def test_push_flow(self):
        """Checks that we update flow by required amount"""
        # push three units of capacity through each edge
        for edge, exp in zip(self.edges, [0, 2]):
            with self.subTest(edge):
                edge.push_flow(3)
                self.assertEqual(exp, edge.unused_capacity())

        # try to push excess amount of flow down an edge
        with self.subTest("exceed capacity"), self.assertRaises(FlowEdgeError):
            self.edges[1].push_flow(3)

    def test_adjust_edge(self):
        (
            res,
            fwd,
        ) = self.edges
        fwd.push_flow(3)
        res.push_flow(-3)

        new_fwd = FlowEdge(Vertex(0), Vertex(1), 2)
```

```
45          new_res = FlowEdge(Vertex(0), Vertex(1), 0)
46
47          fwd.adjust_edge()
48          res.adjust_edge()
49
50          self.assertEqual(new_fwd, fwd)
51          self.assertEqual(new_res, res)
52
53          # catch EdgeCapacityZero
54          with self.assertRaises(EdgeCapacityZero):
55              fwd.push_flow(2)
56              fwd.adjust_edge()
57
58
59  class TestFlowGraph(TestCase):
60      def setUp(self) -> None:
61          # make some nodes for a graph
62          self.nodes = [Vertex(n, label=chr(n + 97)) for n in
    range(5)]
63          self.graph = FlowGraph(self.nodes)
64
65      def test_is_edge(self):
66          a, b, c, d, e = self.nodes
67
68          with self.subTest("no edge"):
69              self.assertFalse(self.graph.is_edge(a, b))
70
71          with self.subTest("edge"):
72              self.graph.add_edge(a, (b, 4))
73              self.assertTrue(self.graph.is_edge(a, b))
74
75      def test_get_edge(self):
76          a, b, c, d, e = self.graph.nodes()
77          self.graph.add_edge(a, (b, 5))
78          self.assertEqual(FlowEdge(a, b, 5), self.graph.
    get_edge(a, b))
79
80      def test_add_edge(self):
81          a, b, c, d, e = self.nodes
82          # check there aren't edges
83          with self.subTest("negative test"):
84              self.assertFalse(self.graph.is_edge(a, b))
85              self.assertFalse(self.graph.is_edge(b, a, residual
    =True))
86          self.graph.to_dot()
87          self.graph.add_edge(a, (b, 5))
88          self.graph.to_dot()
89          with self.subTest("added"):
```

```
 90             self.assertTrue(self.graph.is_edge(a, b))
 91             self.assertTrue(self.graph.is_edge(b, a, residual
    =True))
 92             self.assertFalse(self.graph.is_edge(b, a))
 93
 94         with self.subTest("Net debt (adding)"):
 95             self.assertEqual(
 96                 {
 97                     Vertex(ID=0, label="a"): 5,
 98                     Vertex(ID=1, label="b"): -5,
 99                     Vertex(ID=2, label="c"): 0,
100                     Vertex(ID=3, label="d"): 0,
101                     Vertex(ID=4, label="e"): 0,
102                 },
103                 self.graph.net_debt,
104             )
105
106         with self.subTest("Net debt (removing)"):
107             self.graph.pop_edge(a, b, update_debt=True)
108             self.assertEqual(
109                 {
110                     Vertex(ID=0, label="a"): 0,
111                     Vertex(ID=1, label="b"): 0,
112                     Vertex(ID=2, label="c"): 0,
113                     Vertex(ID=3, label="d"): 0,
114                     Vertex(ID=4, label="e"): 0,
115                 },
116                 self.graph.net_debt,
117             )
118
119     def test_pop_edge(self):
120         a, b, c, d, e = self.nodes
121
122         self.graph.add_edge(a, (b, 5))
123
124         with self.subTest("negative"):
125             self.assertTrue(self.graph.is_edge(a, b))
126             self.assertTrue(self.graph.is_edge(b, a, residual
    =True))
127
128         self.graph.pop_edge(a, b)
129
130         with self.subTest("removed edge"):
131             self.assertFalse(self.graph.is_edge(a, b))
132         with self.subTest("removed residual"):
133             self.assertFalse(self.graph.is_edge(b, a,
    residual=True))
134
```

```
135     def test_flow_neighbours(self):
136         """Checks we get edges that have unused capacity,
    including residual"""
137         graph = self.graph
138         a, b, c, d, *_ = graph.nodes()
139         graph.add_edge(a, (b, 10))
140         graph.add_edge(b, (c, 2))
141         graph.add_edge(c, (d, 10))
142         graph.add_edge(d, (a, 10))
143 
144         graph.to_dot()
145 
146         MaxFlow.augment_flow(graph, [a, b, c, d], 2)
147         c_flow_neighbours = graph.flow_neighbours(c)
148         self.assertEqual([d, b], GenericDigraph.
    nodes_from_edges(c_flow_neighbours))
149 
150     def test_bool(self):
151         # empty
152         print(self.graph.graph)
153         self.assertFalse(bool(self.graph))
154 
155         # with an edge
156         a, b, c, d, e = self.nodes
157         self.graph.add_edge(a, (b, 10))
158 
159 
160 class TestMaxFlow(TestCase):
161     def setUp(self) -> None:
162         graph = FlowGraph([Vertex(n, label=chr(n + 97)) for n
     in range(4)])
163         a, b, c, d, *_ = graph.nodes()
164         graph.add_edge(a, (b, 10))
165         graph.add_edge(b, (c, 2))
166         graph.add_edge(c, (d, 10))
167         graph.add_edge(d, (a, 10))
168 
169         self.graph = graph
170 
171     def test_bottleneck(self):
172         # build a chain with an obvious bottleneck on normal
    graphs
173         graph = self.graph
174         a, b, c, d, *_ = graph.nodes()
175 
176         aug_path = [a, b, c, d]
177 
178         graph.to_dot()
```

```
179
180         self.assertEqual(2, MaxFlow.bottleneck(graph,
    aug_path))
181
182     def test_nodes_to_path(self):
183         a, b, c, d, *_ = self.graph.nodes()
184         edges = MaxFlow.nodes_to_path(self.graph, [a, b, c, d
    ])
185
186         self.assertEqual(
187             edges, [FlowEdge(a, b, 10), FlowEdge(b, c, 2),
    FlowEdge(c, d, 10)]
188         )
189
190     def test_augmenting_path(self):
191         a, b, c, d, *_ = self.graph.nodes()
192         MaxFlow.augmenting_path(self.graph, a, d)
193
194     def test_augment_flow(self):
195         a, b, c, d, *_ = self.graph.nodes()
196
197         MaxFlow.augment_flow(self.graph, [a, b, c, d], 2)
198         self.graph.to_dot()
199
200         # check that the edges and residual edges all now
    have flow 2
201         node_path = [a, b, c, d]
202         flow = 2
203         for _ in range(2):
204             # one for normal one for residual
205             for src, dest in zip(node_path, node_path[1:]):
206                 with self.subTest(f"{src} -> {dest}"):
207                     self.assertEqual(flow, self.graph.
    get_edge(src, dest).flow)
208             # change path to be residual, flow changes
    accordingly
209             node_path.reverse()
210             flow *= -1
211
212     def test_edmonds_karp(self):
213         # build slightly more involved graph
214         nodes = [Vertex(0, label="src"), Vertex(10, label="
    sink")]
215         nodes += [Vertex(n, label=chr(n + 96)) for n in range
    (1, 10)]
216         tg = FlowGraph(nodes)
217
218         s, t, a, b, c, d, e, f, g, h, i = tg.nodes()
```

```
219
220         tg.add_edge(s, (a, 5), (b, 10), (c, 5))
221         tg.add_edge(a, (d, 10))
222         tg.add_edge(b, (a, 15), (e, 20))
223         tg.add_edge(c, (f, 10))
224         tg.add_edge(d, (e, 25), (g, 10))
225         tg.add_edge(e, (c, 5), (h, 30))
226         tg.add_edge(f, (h, 5), (i, 5))
227         tg.add_edge(g, (t, 5))
228         tg.add_edge(h, (t, 15), (i, 5))
229         tg.add_edge(i, (t, 10))
230
231         tg.to_dot()
232
233         max_flow = MaxFlow.edmonds_karp(tg, s, t)
234         self.assertEqual(max_flow, 20)
235
236     def test_old_edmonds(self):
237         labels = ["a", "b", "c", "d", "e", "f"]
238         self.vertices = [Vertex(ID, label=label) for ID,
    label in enumerate(labels)]
239         a, b, c, d, e, f = self.vertices
240
241         # set up weighted_graph and flow graphs
242         self.flow_graph = FlowGraph(self.vertices)
243         self.flow_graph.add_edge(a, (b, 10), (c, 10))
244         self.flow_graph.add_edge(b, (d, 25))
245         self.flow_graph.add_edge(c, (e, 25))
246         self.flow_graph.add_edge(d, (f, 10))
247         self.flow_graph.add_edge(e, (f, 10), (b, 6))
248
249         a, b, c, d, e, f = self.vertices
250
251         expected = 20
252         calculated = MaxFlow.edmonds_karp(self.flow_graph, a
    , f)
253
254         self.assertEqual(expected, calculated)
255
256
257 class TestSimplify(TestCase):  # type: ignore
258     def setUp(self) -> None:
259         self.graph = FlowGraph([Vertex(0, "d"), Vertex(1, "m"
    ), Vertex(2, "t")])
260         d, m, t = self.graph.nodes()
261         self.graph.add_edge(d, (m, 5), (t, 10))
262         self.graph.add_edge(m, (t, 5))
263
```

```
264     def test_simplify_debt(self):
265         print(self.graph.net_debt)
266 
267         people = ["dad", "tom", "maia"]
268         debt = FlowGraph([Vertex(ID, person) for ID, person
    in enumerate(people)])
269 
270         d, t, m = debt.nodes()
271 
272         debt.add_edge(d, (t, 10), (m, 5))
273         debt.add_edge(m, (t, 5))
274 
275         debt.to_dot()
276 
277         clean = Simplify.simplify_debt(debt)
278 
279         clean.to_dot(n=1)
280         self.assertEqual(debt.net_debt, clean.net_debt)
281 
282     def test_adjust_edges(self):
283         # saturate d -> m -> t
284         self.graph.to_dot(n=0)
285         d, m, t = self.graph.nodes()
286         MaxFlow.augment_flow(self.graph, [d, m, t], 5)
287         self.graph.to_dot(n=1)
288         self.graph.adjust_edges()
289         self.graph.to_dot(n=4)
290 
291         # graph should have no edges in or out of m
292         # t should have a res edge to d, 0/0
293         # d should have a fwd to t, 0/10
294 
295         self.assertFalse(self.graph[m])
296         self.assertEqual(self.graph[d], [FlowEdge(d, t, 10)])
297         self.assertEqual(self.graph[t], [FlowEdge(t, d, 0)])
298 
299     def test_netflow(self):
300         """Checking people owed same before and after"""
301 
302     def test_mithun_simplify(self):
303 
304         # gen vertices
305         people: list[Vertex] = []
306         for ID, person in enumerate(["b", "c", "d", "e", "f"
    , "g"]):
307             people.append(Vertex(ID, label=person))
308         b, c, d, e, f, g = people
309 
```

```
310         # build flow graph of transactions
311         messy = FlowGraph(people)
312 
313         messy.add_edge(b, (c, 40))
314         messy.add_edge(c, (d, 20))
315         messy.add_edge(d, (e, 50))
316         messy.add_edge(f, (e, 10), (d, 10), (c, 30), (b, 10))
317         messy.add_edge(g, (b, 30), (d, 10))
318 
319         messy.to_dot()
320 
321         clean = Simplify.simplify_debt(messy)
322         clean.to_dot(n=1)
323 
324         self.assertEqual(messy.net_debt, clean.net_debt)
325 
326     def test_simplest_graph(self):
327         """Shouldn't change"""
328         people = ["dad", "tom", "maia"]
329         debt = FlowGraph([Vertex(ID, person) for ID, person
    in enumerate(people)])
330         d, t, m = debt.nodes()
331         debt.add_edge(d, (t, 5))
332         debt.add_edge(t, (m, 10))
333 
334         debt.to_dot()
335 
336         with self.assertRaises(NoOptimisations):
337             clean = Simplify.simplify_debt(debt)
338             clean.to_dot(n=1)
339 
```

File - /home/tcassar/projects/settle/tests/test_settling/test_graph.py

```python
# coding=utf-8

from unittest import TestCase

from src.simplify.base_graph import Digraph
from src.simplify.flow_graph import *
from src.simplify.graph_objects import Edge
from src.simplify.weighted_digraph import WeightedDigraph


class TestDigraph(TestCase):
    """Generate a digraph and some vertices"""

    def setUp(self) -> None:
        """Build basic graph"""
        labels = ["u", "v", "w"]
        self.vertices = [Vertex(ID, label=label) for ID, label in enumerate(labels)]

        self.graph = Digraph(self.vertices)
        u, v, w = self.vertices
        self.graph.add_edge(u, v, w)
        self.graph.add_edge(v, w)

    def test_init(self):
        expected = "U -> VW\nV -> W\nW -> \n"

        with self.subTest("init"):
            self.assertEqual(expected, str(self.graph))

        with self.subTest("helpers"):
            self.assertTrue(self.graph.is_node(self.vertices[0]))
            self.assertTrue(self.graph.sanitize(*self.vertices))

            with self.assertRaises(GraphError):
                self.graph.edge_from_nodes(self.vertices[2], [Edge(self.vertices[0])])

    def test_is_edge(self):
        u, v, w = self.vertices

        with self.subTest("is edge"):
            self.assertTrue(self.graph.is_edge(u, v))
        with self.subTest("isn't edge"):
            self.assertFalse(self.graph.is_edge(w, v))

```

```
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[v])
62
63      def test_pop_edge(self):
64          u, v, w = self.vertices
65          self.assertTrue(self.graph.is_edge(u, v))
66          self.graph.pop_edge(u, v)
67
68          self.assertFalse(self.graph.is_edge(u, v))
69
70      def test_add_edge(self):
71          u, v, w = self.graph.nodes()
72          self.assertFalse(self.graph.is_edge(w, v))
73          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      def setUp(self) -> None:
83          """Build basic graph"""
84          labels = ["u", "v", "w"]
85          self.vertices = [Vertex(ID, label=label) for ID, label
     in enumerate(labels)]
86
87          self.graph = WeightedDigraph(self.vertices)
88          u, v, w = self.vertices
89          self.graph.add_edge(u, (v, 1), (w, 2))
90          self.graph.add_edge(v, (w, 3))
91
```

```
 92     def test_add_edge(self):
 93         u, v, w = self.vertices
 94         self.assertFalse(self.graph.is_edge(w, v))
 95         self.graph.add_edge(w, (v, 4))
 96         self.assertTrue(self.graph.is_edge(w, v))
 97 
 98     def test_add_existing_edge(self):
 99         u, v, w = self.graph.nodes()
100         self.assertEqual(self.graph.is_edge(v, w), 3)
101         self.graph.add_edge(v, (w, 2))
102         self.assertEqual(self.graph.is_edge(v, w), 5)
103 
104     def test_flow_through(self):
105         for node, flow in zip(self.vertices, [3, 2, -5]):
106             with self.subTest(node):
107                 self.assertEqual(self.graph.flow_through(node
    ), flow)
108 
```

```
1 ID,src,dest,amount,group,src_n,src_e,dest_n,dest_e,src_d,
  dest_d
2 0,04,20,5,0,
  216167920311437527463094155794523202025108931260730876523837234081050636000070147761500936454570470862786970206983368636166003008975173251124763374054321346030354457021425165356037781636930036106404418295413726431002556836900067049867944499904312842155745102543727981913742755364262501982105715862022723433985738139067694378476466514455677010796867090616789485577458637370869261774337704654931841775536224420241750390318182742144140878560279532281843455113675020983184252503669138492451882883596202775900911000502696129433770297061845035322423193568722584809589006962060587421954603201784497846158263582144177430642603,
  65537,
  245435260532226859200890024313870710808087539791682893396507074141095827708342421730095392824267456827229880553467004005448679109437851439095238915265836858730847785194484728870026602121828141260708630044403519141525405559010974999302357937434165456103680400325960557898468091402415007872965823676993425321804474460428515591876290522381689998198437261563638300300691391230512004411119454072768847410336407304623958619851439941769916099206947465717047070945326989108443577274197836674615423409302875065157935697041493847841319691682229439019174629277104917717935881329934336642631167278369058626398091411958822782486021,
  65537,
  42318605026103475553526985044230475639442568138229963925734038989576569874864579669508371562955146126547877955291914430323301128393346172685767442683818738967216638476720764074944647817540070989695696588210951082558372250397158894945714319204774976709066583095965638944919595239862657801000401988151966317962866310493446444221800190039054548479140813630569444720847689992964879750244948034517871603890465032993904165597478235001604038074703682680413388703757501362444680122222701317562224831854480644662034354542275281227314387046825525570376824278684460211954070416261823570911887469407283866423584255451231566190517451642798421916838978706887752529294378563794943959644593481018317996812806135240585387347011403463919484312360288671519765089185961940677538082993405248358724852816062664396208614460346356077862920426734827642392974806673957359914994399933521690425914379137265608737153994017658024271602617427327911897090105542806007602652290786721234528626131896832489562551558690301697229495263298457688434655931885453437842161128989953202272867367837891847562410504381902662947168208004596529501772278874463251649170353674520402106924748077589683971300268994119580437205561936946700600893560778543423518093933835790069197852673
3 1,04,13,10,0,
  216167920311437527463094155794523202025108931260730876523837234081050636000070147761500936454570470862786970206983368636166003
```

```
3
  300897517325112476337405432134603035445702142516535603778163693
  300361064044182954137264310025568369000670498679444999043128422
  155745102543727981913742755364262501982105715862022723433985733
  813906769437847646651445567701079686709061678948557745863737083
  692617743377046549318417755362244202417503903181827421441408783
  560279532281843455113675020983184252503669138492451882883596203
  277590091100050269612943377029706184503532242319356872258480953
  890069620605874219546032017844978461582635821441774306426033,
  65537,
  225658640372474598320855668799003073602163686974087338839648513
  808410874820858287921659997954743444212625888901472524563369153
  672583835216760904898276186048823446599370109121575718012944123
  422946171359122243583308263282848559863033353276205119720064533
  142090664344661055123408494125857210821767752962369786263684643
  190409612760214589978439512179632414787894743946995271605807713
  635153728361835033613801478959059921454822591898567781991786263
  193287410605673929386962683042821731362106955244574788342442153
  547988454805806522190171092806788131795962815538890383392307033
  284433283787263864958666155092517689554259271552956245888233,
  65537,
  423186050261034755535269850442304756394425681382299639257340389
  895765698748645796695083715629551461265478779552919144303233011
  128393346172685767442683818738967216638476720764074944647817540
  400709896956965882109510825583722503971588949457143192047749767
  709066583095965638944919595239862657801000401988151966317962866
  631049344644422180019003905454847914081363056944472084768999296
  648797502449480345178716038904650329939041655974782350016040380
  074703682680413388703757501362444680122222701317562224831854480
  064466203435454227528122731438704682552557037682427868446021195
  540704162618235709118874694072838664235842554512315661905,
  775414282189927514867276448624982714729195146353425831311913060
  090515724089556837205820094594019006617381787091438615311514627
  677050821533096659644045304151777471873569870060864117925980999
  968681474435423184688969908468460498362071977017583852800075266
  246224538969096382406756380016525685903567419429111431200416762
  235694762226761908445600472087203109098947447060087682640572923
  907398373569449250780454443529816977994400481127637898253763826
  262601152485343855539104875876907807892323552073225968951530325
  599004522731021331970408139565486136882985543917585078753787794
  432847824073525835250217870849030986866114641442612783873
4 2,20,13,5,0,
  245435260532226859200890024313870710808087539791682893396507074
  141409582770834242173009539282426745682722988055346700400544867
  791094378514390952389152658368587308477851944847288700266021218
  828141260708630044403519141525405559010974999302357937434165456
  610368040032596055789846809140241500787296582367699342532180447
  744604285155918762905223816899981984372615636383003006913912306
```

```
4
  5120044111194540727688474103364073046239586198514399417699160 9
  9206947465717047070945326989108443577274197836674615423409302 8
  7506515793569704149384784131969168222943901917462927710491717 9
  3588132993433664263116727836905862639809141195882278248602 1,
  65537,
  2256586403724745983208556687990030736021636869740873388396485 1
  8084108748208582879216599979547434442126258889014725245633691 5
  6725838352167609048982761860488234465993701091215757180129441 2
  4229461713591222435833082632828485598630333532762051197200645 3
  1420906643446610551234084941258572108217677529623697862636846 4
  1904096127602145899784395121796324147878947439469952716058077 1
  6351537283618350336138014789590599214548225918985677819917862 6
  1932874106056739293869626830428217313621069552445747883424421 5
  5479884548058065221901710928067881317959628155388903833923070 3
  2844332837872638649586661550925176895542592715529562458882 3,
  65537,
  1745164279842191683897870688775252929437856379494395964459348 1
  0183179968128061352405853873470114034639194843123602886715197 6
  5089185961940677538082993405248358724852816062664396208614460 3
  4635607786292042673482764239297480667395735991499439993352169 0
  4259143791372656087371539940176580242716026174273279118970901 0
  5542806007602652290786721234528626131896832489562551558690301 6
  9722949526329845768834655931885453437842161128989953202272867 3
  6783789184756241050438190266294716820800459652950177227887446 3
  2516491703536745204021069247480775896839713002689941195804372 0
  5561936946700600893560778543423518093933835790069197852673,
  7754142821899275148672764486249827147291951463534258313119130 6
  0905157240895568372058200945940190066173817870914386153115146 2
  6770508215330966596440453041517774718735698700608641179259809 9
  9686814744354231846889699084684604983620719770175838528000752 6
  2462245389690963824067563800165256859035674194291114312004167 6
  2356947622267619084456004720872031090989474470600876826405729 2
  3907398373569449250780454443529816977994400481127637898253763 8
  2626011524853438555391048758769078078923235520732259689515303 2
  5990045227310213319704081395654861368829855439175850787537877 9
  4328478240735258352502178708490309868661146414426127838 73
5 3,04,20,5,1,
  2161679203114375274630941557945232020251089312607308765238372 3
  4081050636000701477615009364545704708627869702069833686361660 0
  3008975173251124763374054321346030354457021425165356037781636 9
  3003610640441829541372643100255683690006704986794449990431284 2
  1557451025437279819137427553642625019821057158620227234339857 3
  8139067694378476466514455677010796867090616789485577458637370 8
  6926177433770465493184177553622442024175039031818274214414087 8
  5602795322818434551136750209831842525036691384924518828835962 0
  2775900911000502696129433770297061845035322423193568722584809 5
  8900696206058742195460320178449784615826358214417743064260 3,
```

```
5 65537,
  24543526053222685920089002431387071080808753979168289339650707
  41410958277083424217300953928242674568272298805534670040054486
  79109437851439095238915265836858730847785194484728870026602121
  82814126070863004440351914152540555901097499930235793743416545
  61036804003259605578984680914024150078729658236769934253218044
  74460428515591876290522381689998198437261563638300300691391230
  51200441111945407276884741033640730462395861985143994176991609
  92069474657170470709453269891084435772741978366746154234093028
  75065157935697041493847841319691682229439019174629277104917179
  35881329934336642631167278369058626398091411958822782486021,
  65537,
  42318605026103475553526985044230475639442568138229963925734038
  98957656987486457966950837156295514612654787795529191443032330
  11283933461726857674426838187389672166384767207640749446478175
  40070989695696588210951082558372250397158894945714319204774976
  70906658309596563894491959523986265780100040198815196631796286
  63104934464442218001900390545484791408136305694447208476899929
  64879750244948034517871603890465032993904165597478235001604038
  07470368268041338870375750136244468012222270131756222483185448
  06446620343545422752812273143870468255255703768242786844602119
  54070416261823570911887469407283866423584255451231566190 5,
  17451642798421916838978706887752529294378563794943959644593481
  01831799681280613524058538734701140346391948431236028867151976
  50891859619406775380829934052483587248528160626643962086144603
  46356077862920426734827642392974806673957359914994399933521690
  42591437913726560873715399401765802427160261742732791189709010
  55428060076026522907867212345286261318968324895625515586903016
  97229495263298457688434655931885453437842161128989953202272867
  36783789184756241050438190266294716820800459652950177227887446
  32516491703536745204021069247480775896839713002689941195804372
  05561936946700600893560778543423518093933835790069197852673
6 4,04,13,10,1,
  21616792031143752746309415579452320202510893126073087652383723
  40810506360007014776150093645457047086278697020698336863616600
  30089751732511247633740543213460303544570214251653560377816369
  30036106404418295413726431002556836900067049867944499904312842
  15574510254372798191374275536426250198210571586202272343398573
  81390676943784764665144556770107968670906167894855774586373708
  69261774337704654931841775536224420241750390318182742144140878
  56027953228184345511367502098318425250366913849245188288359620
  27759009110005026961294337702970618450353224231935687225848095
  89006962060587421954603201784497846158263582144177430642603,
  65537,
  22565864037247459832085566879900307360216368697408733883964851
  80841087482085828792165999795474344421262588890147252456336915
  67258383521676090489827618604882344659937010912157571801294412
  42294617135912224358330826328284855986303335327620511972006453
```

```
6
  14209066434466105512340849412585721082176775296236978626368464
  19040961276021458997843951217963241478789474394699527160580771
  63515372836183503361380147895905992145482259189856778199178626
  19328741060567392938696268304282173136210695524457478834244215
  54798845480580652219017109280678813179596281553889038339230703
  28443328378726386495866615509251768955425927155295624588823,
  65537,
  42318605026103475553526985044230475639442568138229963925734038
  98957656987486457966950837156295514612654787795529191443032330
  11283933461726857674426838187389672166384767207640749446478175
  40070989695696588210951082558372250397158894945714319204774976
  70906658309596563894491959523986265780100040198815196631796286
  63104934464442218001900390545484791408136305694447208476899929
  64879750244948034517871603890465032993904165597478235001604038
  07470368268041338870375750136244468012222270131756222483185448
  06446620343545422752812273143870468255255703768242786844602119
  54070416261823570911887469407283866423584255451231566190 5,
  77541428218992751486727644862498271472919514635342583131191306
  09051572408955683720582009459401900661738178709143861531151462
  67705082153309665964404530415177747187356987006086411792598099
  96868147443542318468896990846846049836207197701758385280007526
  24622453896909638240675638001652568590356741942911143120041676
  23569476222676190844560047208720310909894744706008768264057292
  39073983735694492507804544435298169779944004811276378982537638
  26260115248534385553910487587690780789232355207322596895153032
  59900452273102133197040813956548613688298554391758507875378779
  43284782407352583525021787084903098686611464144261278387 3
7 5,20,13,5,1,
  24543526053222685920089002431387071080808753979168289339650707
  41410958277083424217300953928242674568272298805534670040054486
  79109437851439095238915265836858730847785194484728870026602121
  82814126070863004440351914152540555901097499930235793743416545
  61036804003259605578984680914024150078729658236769934253218044
  74460428515591876290522381689998198437261563638300300691391230
  51200441111945407276884741033640730462395861985143994176991609
  92069474657170470709453269891084435772741978366746154234093028
  75065157935697041493847841319691682229439019174629277104917179
  35881329934336642631167278369058626398091411958822782486021,
  65537,
  22565864037247459832085566879900307360216368697408733883964851
  80841087482085828792165999795474344421262588890147252456336915
  67258383521676090489827618604882344659937010912157571801294412
  42294617135912224358330826328284855986303335327620511972006453
  14209066434466105512340849412585721082176775296236978626368464
  19040961276021458997843951217963241478789474394699527160580771
  63515372836183503361380147895905992145482259189856778199178626
  19328741060567392938696268304282173136210695524457478834244215
```

```
7
  54798845480580652219017109280678813179596281553889038339230703
  28443328378726386495866615509251768955425927155295624588823,
  65537,
  17451642798421916838978706887752529294378563794943959644593481
  01831799681280613524058538734701140346391948431236028867151976
  50891859619406775380829934052483587248528160626643962086144603
  46356077862920426734827642392974806673957359914994399933521690
  42591437913726560873715399401765802427160261742732791189709010
  55428060076026522907867212345286261318968324895625515586903016
  97229495263298457688434655931885453437842161128989953202272867
  36783789184756241050438190266294716820800459652950177227887446
  32516491703536745204021069247480775896839713002689941195804372
  05561936946700600893560778543423518093933835790069197852673,
  77541428218992751486727644862498271472919514635342583131191306
  09051572408955683720582009459401900661738178709143861531151462
  67705082153309665964404530415177747187356987006086411792598099
  96868147443542318468896990846846049836207197701758385280007526
  24622453896909638240675638001652568590356741942911143120041676
  23569476222676190844560047208720310909894744706008768264057292
  39073983735694492507804544435298169779944004811276378982537638
  26260115248534385553910487587690780789232355207322596895153032
  59900452273102133197040813956548613688298554391758507875378779
  43284782407352583525021787084903098686611464144261278387 3
8 6,04,20,5,2,
  21616792031143752746309415579452320202510893126073087652383723
  40810506360007014776150093645457047086278697020698336863616600
  30089751732511247633740543213460303544570214251653560377816369
  30036106404418295413726431002556836900067049867944499904312842
  15574510254372798191374275536426250198210571586202272343398573
  81390676943784764665144556770107968670906167894855774586373708
  69261774337704654931841775536224420241750390318182742144140878
  56027953228184345511367502098318425250366913849245188288359620
  27759009110005026961294337702970618450353224231935687225848095
  89006962060587421954603201784497846158263582144177430642603,
  65537,
  24543526053222685920089002431387071080808753979168289339650707
  41410958277083424217300953928242674568272298805534670040054486
  79109437851439095238915265836858730847785194484728870026602121
  82814126070863004440351914152540555901097499930235793743416545
  61036804003259605578984680914024150078729658236769934253218044
  74460428515591876290522381689998198437261563638300300691391230
  51200441111945407276884741033640730462395861985143994176991609
  92069474657170470709453269891084435772741978366746154234093028
  75065157935697041493847841319691682229439019174629277104917179
  35881329934336642631167278369058626398091411958822782486021,
  65537,
  42318605026103475553526985044230475639442568138229963925734038
```

```
8
  98957656987486457966950837156295514612654787795529191443032330
  11283933461726857674426838187389672166384767207640749446478175
  40070989695696588210951082558372250397158894945714319204774976
  70906658309596563894491959523986265780100040198815196631796286
  63104934464442218001900390545484791408136305694447208476899929
  64879750244948034517871603890465032993904165597478235001604038
  07470368268041338870375750136244468012222270131756222483185448
  06446620343545422752812273143870468255255703768242786844602119
  54070416261823570911887469407283866423584255451231566190 5,
  17451642798421916838978706887752529294378563794943959644593481
  01831799681280613524058538734701140346391948431236028867151976
  50891859619406775380829934052483587248528160626643962086144603
  46356077862920426734827642392974806673957359914994399933521690
  42591437913726560873715399401765802427160261742732791189709010
  55428060076026522907867212345286261318968324895625515586903016
  97229495263298457688434655931885453437842161128989953202272867
  36783789184756241050438190266294716820800459652950177227887446
  32516491703536745204021069247480775896839713002689941195804372
  05561936946700600893560778543423518093933835790069197852673
9 7,04,13,10,2,65537,
  62161679203114375274630941557945232020251089312607308765238372
  34081050636000701477615009364545704708627869702069833686361660
  03008975173251124763374054321346030354457021425165356037781636
  93003610640441829541372643100255683690006704986794449990431284
  21557451025437279819137427553642625019821057158620227234339857
  38139067694378476466514455677010796867090616789485577458637370
  86926177433770465493184177553622442024175039031818274214414087
  85602795322818434551136750209831842525036691384924518828835962
  02775900911000502696129433770297061845035322423193568722584809
  58900696206058742195460320178449784615826358214417743064260355
  37,
  22565864037247459832085566879900307360216368697408733883964851
  80841087482085828792165999795474344421262588890147252456336915
  67258383521676090489827618604882344659937010912157571801294412
  42294617135912224358330826328284855986303335327620511972006453
  14209066434466105512340849412585721082176775296236978626368464
  19040961276021458997843951217963241478789474394699527160580771
  63515372836183503361380147895905992145482259189856778199178626
  19328741060567392938696268304282173136210695524457478834244215
  54798845480580652219017109280678813179596281553889038339230703
  28443328378726386495866615509251768955425927155295624588823,
  65537,
  42318605026103475553526985044230475639442568138229963925734038
  98957656987486457966950837156295514612654787795529191443032330
  11283933461726857674426838187389672166384767207640749446478175
  40070989695696588210951082558372250397158894945714319204774976
  70906658309596563894491959523986265780100040198815196631796286
```

```
 9
   631049344644422180019003905454847914081363056944472084768999296487975024494803451787160389046503299390416559747823500160403807470368268041338870375750136244468012222270131756222483185448064466203435454227528122731438704682552557037682427868446021195407041626182357091188746940728386642358425545123156619057754142821899275148672764486249827147291951463534258313119130609051572408955683720582009459401900661738178709143861531151462677050821533096659644045304151777471873569870060864117925980999686814744354231846889699084684604983620719770175838528000752624622453896909638240675638001652568590356741942911143120041676235694762226761908445600472087203109098947447060087682640572923907398373569449250780454443529816977994400481127637898253763826260115248534385553910487587690780789232355207322596895153032599004522731021331970408139565486136882985543917585078753787794328478240735258352502178708490309868661146414426127838738,20,13,5,2,24543526053222685920089002431387071080808753979168289339650707414109582770834242173009539282426745682722988055346700400544867910943785143909523891526583685873084778519448472887002660212182814126070863004440351914152540555901097499930235793743416545610368040032596055789846809140241500787296582367699342532180447446042851559187629052238168999819843726156363830030069139123051200441111945407276884741033640730462395861985143994176991609920694746571704707094532698910844357727419783667461542340930287506515793569704149384784131969168222943901917462927710491717935881329934336642631167278369058626398091411958822782486021,65537,225658640372474598320855668799003073602163686974087338839648518084108748208582879216599979547434442126258889014725245633691567258383521676090489827618604882344659937010912157571801294412422946171359122243583308263282848559863033353276205119720064531420906643446610551234084941258572108217677529623697862636846419040961276021458997843951217963241478789474394699527160580771635153728361835033613801478959059921454822591898567781991786261932874106056739293869626830428217313621069552445747883424421554798845480580652219017109280678813179596281553889038339230703284433283787263864958666155092517689554259271552956245888236517451642798421916838978706887752529294378563794943959644593481018317996812806135240585387347011403463919484312360288671519765089185961940677538082993405248358724852816062664396208614460346356077862920426734827642392974806673957359914994399933521690425914379137265608737153994017658024271602617427327911897090105542806007602652290786721234528626131896832489562551558690301697229495263298457688434655931885453437842161128989953202272867367837891847562410504381902662947168208004596529501772278874463251649170353674520402106924748077589683971300268994119580437 2
```

```
10 055619369467006008935607785434235180939338357900691978526 73,
   775414282189927514867276448624982714729195146353425831311913 06
   090515724089556837205820094594019006617381787091438615311514 62
   677050821533096659644045304151777471873569870060864117925980 99
   968681474435423184688969908468460498362071977017583852800075 26
   246224538969096382406756380016525685903567419429111431200416 76
   235694762226761908445600472087203109098947447060087682640572 92
   390739837356944925078045444352981697799440048112763789825376 38
   262601152485343855539104875876907807892323552073225968951530 32
   599004522731021331970408139565486136882985543917585078753787 79
   432847824073525835250217870849030986866114641442612783873
11
```

```python
# coding=utf-8
import unittest

import src.simplify.flow_graph
import src.simplify.graph_objects
from src.crypto import keys
from src.transactions.ledger import *
from src.transactions.transaction import VerificationError


def setUpModule():
    print(os.getcwd())
    os.chdir("/home/tcassar/projects/settle")


def key_path(usr: str, keytype="private") -> str:
    assert usr == "d" or usr == "m" or usr == "t"
    return f"./src/crypto/sample_keys/{usr}_{keytype}-key.pe{'m' if keytype == 'private' else ''}"


class TestLedger(unittest.TestCase):
    def setUp(self) -> None:
        self.valid: Ledger
        self.missing_key: Ledger
        self.invalid: Ledger

        # load in raw copies of d, m, t test keys
        ldr = keys.RSAKeyLoader()
        self.d_m_t_keys: dict[int, keys.RSAPrivateKey] = {}
        pubs: list[keys.RSAPublicKey] = []

        for person, id in zip(["d", "m", "t"], [4, 13, 20]):
            ldr.load(key_path(person))
            ldr.parse()
            self.d_m_t_keys[id] = keys.RSAPrivateKey(ldr)
            pubs.append(keys.RSAPublicKey(ldr))

        self.d_pub, self.m_pub, self.t_pub = pubs
        self.d_priv, self.m_priv, self.t_priv = self.d_m_t_keys.values()

        self.valid, self.missing_key, self.invalid = LedgerLoader.load_from_csv(
            "./tests/test_transactions/database.csv"
        )

    def test_add(self):
```

```
46
47          with self.subTest("Add"):
48              ledger = Ledger()
49              self.assertFalse(not not ledger)
50              ledger.append(Transaction(0, 0, 0, self.d_priv,
    self.m_priv))
51              self.assertTrue(not not ledger)
52
53          with self.subTest("Catch non transaction"), self.
    assertRaises(LedgerBuildError):
54              ledger.append(6)  # type: ignore
55
56      def test_load_from_csv(self):
57          ledger_list = LedgerLoader.load_from_csv(
58              "./tests/test_transactions/database.csv"
59          )
60          self.assertEqual(len(ledger_list), 3)
61
62      def test_verify_transactions(self):
63          """
64          Make three ledgers:
65              1) Valid transactions
66              2) An unsigned transaction
67              3) An invalid signature
68
69          Check that first one goes through no issues, and that
    other two are caught
70          """
71
72          with self.subTest("unsigned"), self.assertRaises(
    VerificationError):
73              self.valid._verify_transactions()
74
75          # sign transactions in ledger; not normal procedure so
     contrived
76          self.sign()
77
78          with self.subTest("signed"):
79              self.valid._verify_transactions()
80
81          with self.subTest("missing key"), self.assertRaises(
    VerificationError):
82              self.missing_key._verify_transactions()
83
84          with self.subTest("invalid key"), self.assertRaises(
    VerificationError):
85              self.invalid._verify_transactions()
86
```

```
 87     def test_as_flow(self):
 88         # sign ledger
 89         self.sign()
 90 
 91         Vertex = src.simplify.graph_objects.Vertex
 92 
 93         exp = src.simplify.flow_graph.FlowGraph(
 94             [Vertex(ID=4), Vertex(ID=13), Vertex(ID=20)]
 95         )
 96         d, m, t = exp.nodes()
 97         exp.add_edge(d, (m, 10), (t, 5))
 98         exp.add_edge(t, (m, 5))
 99 
100         as_flow = self.valid._as_flow()
101 
102         with self.subTest("nodes"):
103             self.assertEqual(exp.nodes(), self.valid.nodes)
104 
105         with self.subTest("to flow graph"):
106             self.assertEqual(exp, as_flow)
107 
108     def sign(self):
109         for trn in self.valid.ledger:
110             trn.sign(self.d_m_t_keys[trn.src], origin="src")
111             trn.sign(self.d_m_t_keys[trn.dest], origin="dest"
    )
112             print("verifying...")
113             trn.verify()
114             print("verified")
115 
116     def test_flow_to_transactions(self):
117         self.maxDiff = None
118         self.sign()
119         as_flow = self.valid._as_flow()
120 
121         # remove sigs, ID, for comparison
122         trn: Transaction
123         for trn in self.valid.ledger:
124             trn.ID = 0
125             trn.signatures = {}
126 
127         with self.subTest("to transactions"):
128             self.valid.ledger.sort(key=lambda trn: trn.amount
    )
129             calc: list[Transaction] = self.valid.
    _flow_to_transactions(as_flow)
130             calc.sort(key=lambda trn: trn.amount)
131 
```

```
132             self.assertEqual(calc, self.valid.ledger)
133
134     def test_simplify_ledger(self):
135         self.sign()
136         self.valid.simplify_ledger()
137
138         trn = Transaction(4, 13, 15, self.d_pub, self.m_pub)
139
140         self.assertEqual(self.valid.ledger, [trn])
141
```

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, self.
   pub_key)
20         self.trn.time = 0  # type: ignore
21
22     def test_hash(self):
23         self.assertEqual(
24             self.trn.hash(),
25             b"\xb9y\x8f\xe6\xdeo\x08\xd4\x1f\xdd\x01\xa3\xb4\
   xc5\x1d\x98\xd4\xac\x1bJ\x02\xd3\xa0\x01!\xf3\x99\xdf\xd5\\\
   x85v",
26         )
27         self.trn.time = 1
28         self.assertNotEqual(
29             self.trn.hash(),
30             b"\xb9y\x8f\xe6\xdeo\x08\xd4\x1f\xdd\x01\xa3\xb4\
   xc5\x1d\x98\xd4\xac\x1bJ\x02\xd3\xa0\x01!\xf3\x99\xdf\xd5\\\
   x85v",
31         )
32
33     def test_sign(self):
34         """Working on assumption that rsa.Notary is working;
   tested in settle/tests/test_crypto"""
35
36         self.trn.sign(self.key, origin="src")
37
38         with (self.subTest("catch invalid origin"), self.
   assertRaises(ValueError)):
39             self.trn.sign(self.key, origin="no")
40
41         with self.subTest("invalid key types"), self.
```

```
41 assertRaises(TransactionError):
42             self.trn.sign(12, origin="src")
43             self.trn.sign(self.pub_key, origin="dest")
44 
45         with self.subTest("sig_overwrite"), self.assertRaises(
   TransactionError):
46             self.trn.sign(self.key, origin="src")
47 
48         with self.subTest("Right sig"):
49             self.assertEqual(
50                 b"\xc1~\xcbu\xecX\x01E*\xf2;O\xe3\xf3\x08x\xee
   \x84\xfc\xe1\xca\x8b\xa2\xedj\xec\x9b\xfd\xe5$7\x88n\xb7\x86\
   xfc~\x98\x91\x80Z\xcd\x1e\xf5}\xc2<JS\xefY\xe5UW\xfc\x0e\xd2\
   xbe\xc9\xea\xfbzm\xf2\xa7\x08\x8d\x05\x16\xe4@\xf40\x03I\xca\
   xbc.\x95\xbb\xd3\n\xfd9\xb0Wk\xf4\x03\x96\xdbF\xcd\xa0E\xd1\
   xac\xa6(\xb9\x92\xdb\x841\xe0U\"\xe4\x7f\xeb U\xc9Z\xe5\xf6\
   x19\xc1Wn\x17&\x17\x84Dt\xb6z\xc0\x02\x85`\xf3\xd3\xd5\x98t7\
   xcd_\xfc\xa7\\\xa7t\xe8\xb1\xafl\x05\xb3\x07a\xd0jr\xc4}\xd8\
   xb58\xf4o\x9f\x02\xa2\xe6?3B\xf6\xe1\xe6\xb2\x02d\xfd\xd5\x83\
   xcf\xcc\xb6\x06m\xc2p\xd7\x00@\x93H\xc6]\x0c\x98\x1e\xdf\xda\
   x00\x86\xd7\xec\xc7\x10\x025eiry\xd6\x80\xfe\xe2+\xb8\x1dn\noB
   \xc0\xa8a\xc8t\xbfg\xbb\xca(Aj\xa5\xeb\x94\x0c-\xacr'\xfe\n^Z\
   x13\xa0'\xaa\x96{\xf8x\xce\xd6\x90",
51                 self.trn.signatures[self.trn.src],
52             )
53 
54     def test_verify(self):
55         # check that we are complained at if no valid keys are
    passed in
56         with self.subTest("catch invalid keys"), self.
   assertRaises(VerificationError):
57             self.trn.verify()
58             self.trn.verify()  # wrong type  # type: ignore
59 
60         # check works with priv and public keys
61 
62         with self.subTest("good verif"):
63             self.trn.sign(self.key, origin="src")
64             self.trn.verify()
65 
66         with self.subTest("priv/pub keys"):
67             self.trn.verify()
68             self.trn.verify()
69 
70         with self.subTest("verify src, dest"):
71             self.trn.verify()
72             self.trn.verify()
73 
```

```
74          with self.subTest("verify >1 param"):
75              self.trn.verify()
76
77          with self.subTest("bad key"), self.assertRaises(
   VerificationError):
78              # edit pub key, thus should fail
79              self.pub_key.lookup["n"] = 3
80              self.trn.verify()
81
```

```
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
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
 "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<!-- Generated by graphviz version 2.43.0 (0)
 -->
<!-- Title: %3 Pages: 1 -->
<svg width="144pt" height="131pt"
     viewBox="0.00 0.00 143.59 131.00" xmlns="http://www.w3.org/2000/svg">
<g id="graph0" class="graph" transform="scale(1 1) rotate(0) translate(4 127)">
<title>%3</title>
<polygon fill="white" stroke="transparent" points="-4,4 -4,-127 139.59,-127 139.59,4 -4,4"/>
<!-- 4 -->
<g id="node1" class="node">
<title>4</title>
<ellipse fill="none" stroke="black" cx="36.59" cy="-105" rx="27" ry="18"/>
<text text-anchor="middle" x="36.59" y="-101.3" font-family="Times,serif" font-size="14.00">4</text>
</g>
<!-- 13 -->
<g id="node2" class="node">
<title>13</title>
<ellipse fill="none" stroke="black" cx="36.59" cy="-18" rx="27" ry="18"/>
<text text-anchor="middle" x="36.59" y="-14.3" font-family="Times,serif" font-size="14.00">13</text>
</g>
<!-- 4&#45;&gt;13 -->
<g id="edge1" class="edge">
<title>4&#45;&gt;13</title>
<path fill="none" stroke="black" d="M18.95,-91.35C12.17,-85.42 5.2,-77.73 1.59,-69 -2.87,-58.25 2.94,-47.52 11.03,-38.77"/>
<polygon fill="black" stroke="black" points="13.69,-41.07 18.52,-31.64 8.86,-36 13.69,-41.07"/>
<text text-anchor="middle" x="26.59" y="-57.8" font-family="Times,serif" font-size="14.00">  0/15  </text>
</g>
<!-- 13&#45;&gt;4 -->
<g id="edge2" class="edge">
<title>13&#45;&gt;4</title>
<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" />
<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">  0/0  </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
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
 "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<!-- Generated by graphviz version 2.43.0 (0)
 -->
<!-- Title: %3 Pages: 1 -->
<svg width="221pt" height="218pt"
     viewBox="0.00 0.00 221.18 218.00" xmlns="http://www.w3.org/2000/svg">
<g id="graph0" class="graph" transform="scale(1 1) rotate(0) translate(4 214)">
<title>%3</title>
<polygon fill="white" stroke="transparent" points="-4,4 -4,-214 217.18,-214 217.18,4 -4,4"/>
<!-- 4 -->
<g id="node1" class="node">
<title>4</title>
<ellipse fill="none" stroke="black" cx="103.18" cy="-192" rx="27" ry="18"/>
<text text-anchor="middle" x="103.18" y="-188.3" font-family="Times,serif" font-size="14.00">4</text>
</g>
<!-- 13 -->
<g id="node2" class="node">
<title>13</title>
<ellipse fill="none" stroke="black" cx="39.18" cy="-105" rx="27" ry="18"/>
<text text-anchor="middle" x="39.18" y="-101.3" font-family="Times,serif" font-size="14.00">13</text>
</g>
<!-- 4&#45;&gt;13 -->
<g id="edge1" class="edge">
<title>4&#45;&gt;13</title>
<path fill="none" stroke="black" d="M76.01,-189.76C52.42,-187.08 19.57,-179.1 3.18,-156 -4.12,-145.71 2.94,-134.29 12.69,-124.96"/>
<polygon fill="black" stroke="black" points="15.15,-127.46 20.47,-118.29 10.6,-122.15 15.15,-127.46"/>
<text text-anchor="middle" x="28.18" y="-144.8" font-family="Times,serif" font-size="14.00">  0/10  </text>
</g>
<!-- 20 -->
<g id="node3" class="node">
<title>20</title>
<ellipse fill="none" stroke="black" cx="103.18" cy="-18" rx="27" ry="18"/>
<text text-anchor="middle" x="103.18" y="-14.3" font-family="Times,serif" font-size="14.00">20</text>
```

```
</g>
<!-- 4&#45;&gt;20 -->
<g id="edge2" class="edge">
<title>4&#45;&gt;20</title>
<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"/>
<polygon fill="black" stroke="black" points="118,-43.48 111.21,-35.34 111.44,-45.93 118,-43.48"/>
<text text-anchor="middle" x="143.68" y="-101.3" font-family="Times,serif" font-size="14.00">  0/5  </text>
</g>
<!-- 13&#45;&gt;4 -->
<g id="edge3" class="edge">
<title>13&#45;&gt;4</title>
<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"/>
<polygon fill="red" stroke="red" points="82.4,-169.94 91.28,-175.72 87.95,-165.66 82.4,-169.94"/>
<text text-anchor="middle" x="96.68" y="-144.8" font-family="Times,serif" font-size="14.00">  0/0  </text>
</g>
<!-- 13&#45;&gt;20 -->
<g id="edge4" class="edge">
<title>13&#45;&gt;20</title>
<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"/>
<polygon fill="red" stroke="red" points="67.09,-26.64 76.26,-21.34 65.78,-19.77 67.09,-26.64"/>
<text text-anchor="middle" x="32.68" y="-57.8" font-family="Times,serif" font-size="14.00">  0/0  </text>
</g>
<!-- 20&#45;&gt;4 -->
<g id="edge6" class="edge">
<title>20&#45;&gt;4</title>
<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"/>
<polygon fill="red" stroke="red" points="127.03,-169.33 121.45,-178.33 131.43,-174.78 127.03,-169.33"/>
<text text-anchor="middle" x="192.68" y="-101.3" font-family="Times,serif" font-size="14.00">  0/0  </text>
</g>
<!-- 20&#45;&gt;13 -->
<g id="edge5" class="edge">
<title>20&#45;&gt;13</title>
<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"
/>
<polygon fill="black" stroke="black" points="53.55,-78.5 50.
43,-88.62 59.18,-82.65 53.55,-78.5"/>
<text text-anchor="middle" x="96.68" y="-57.8" font-family="
Times,serif" font-size="14.00">  0/5  </text>
</g>
</g>
</svg>

```