import unittest

from braille\_autocorrect import qwerty\_to\_pattern, levenshtein, BrailleAutoCorrect

class TestQwertyToPattern(unittest.TestCase):

    def test\_single\_key(self):

        self.assertEqual(qwerty\_to\_pattern('s'), '1')

        self.assertEqual(qwerty\_to\_pattern('k'), '5')

    def test\_multiple\_keys(self):

        self.assertEqual(qwerty\_to\_pattern('sd'), '12')

        self.assertEqual(qwerty\_to\_pattern('ds'), '12')  # order doesn't matter

        self.assertEqual(qwerty\_to\_pattern('sdfj'), '1234')

    def test\_invalid\_keys\_ignored(self):

        self.assertEqual(qwerty\_to\_pattern('sxz'), '1')

        self.assertEqual(qwerty\_to\_pattern(''), '')

class TestLevenshtein(unittest.TestCase):

    def test\_same\_string(self):

        self.assertEqual(levenshtein('abc', 'abc'), 0)

    def test\_simple\_insert\_delete(self):

        self.assertEqual(levenshtein('abc', 'ab'), 1)

        self.assertEqual(levenshtein('ab', 'abc'), 1)

    def test\_substitution(self):

        self.assertEqual(levenshtein('kitten', 'sitten'), 1)

        self.assertEqual(levenshtein('kitten', 'sitting'), 3)

    def test\_max\_dist\_abort(self):

        # should abort once distance > max\_dist

        self.assertGreater(levenshtein('abcdef', 'xyz', max\_dist=2), 2)

        # if within max\_dist

        self.assertEqual(levenshtein('abc', 'adc', max\_dist=1), 1)

class TestBrailleAutoCorrect(unittest.TestCase):

    def setUp(self):

        # dictionary words in QWERTY format

        self.words = ['sad', 'fat', 'cat', 'kitty', 'ads']

        self.ac = BrailleAutoCorrect(self.words)

    def test\_suggest\_exact\_match(self):

        # input matches 'sad' pattern exactly

        suggestions = self.ac.suggest('sd', max\_dist=0)

        self.assertIn('sad', suggestions)

    def test\_suggest\_distance(self):

        # 'sdfj' pattern close to 'sad' and 'fat'

        suggestions = self.ac.suggest('sdfj', max\_dist=2, top\_k=2)

        # ensure at most top\_k returned and sorted by dist then alpha

        self.assertEqual(len(suggestions), 2)

        self.assertTrue(all(w in self.words for w in suggestions))

    def test\_no\_suggestions(self):

        # input pattern too far from any word

        suggestions = self.ac.suggest('llll', max\_dist=1)

        self.assertEqual(suggestions, [])

    def test\_top\_k(self):

        # even if more candidates, only top\_k returned

        # use a small dictionary to force more than top\_k matches

        ac\_small = BrailleAutoCorrect(['sad', 'ads', 'das', 'sda'])

        suggestions = ac\_small.suggest('sd', max\_dist=1, top\_k=3)

        self.assertEqual(len(suggestions), 3)

        # suggestions should be unique

        self.assertEqual(len(set(suggestions)), len(suggestions))

if \_\_name\_\_ == '\_\_main\_\_':

    unittest.main()