```
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()
```