task45_Zhukov_vlad

November 16, 2016

0.0.1 See src.py, test.py files for algorithm code

```
In [50]: import src
         import test
```

1 Let's test algorithm

See test.py

```
In [51]: import unittest
         suite = unittest.TestLoader().loadTestsFromTestCase(test.Tests)
         unittest.TextTestRunner(verbosity=2).run(suite)
```

```
test_strings_overlap (test.Tests) ... ok

----------------------------------------------------------------------
Ran 1 test in 0.001s

OK
```

```
Out[51]: <unittest.runner.TextTestResult run=1 errors=0 failures=0>
```

1.1 Example of usage

```
In [52]: strings = ['cde', 'abc', 'eab', 'fgh', 'ghf', 'hed']
         algo = src.superstring4(strings)
         print('superstring4: ' + str(algo.solve()))
         print ('greedy:       ' + \
                str(src.greedy_min_max_contain_string(strings)))
         print ('answer:       ' + \
                str(src.min_max_contain_string(set(strings))))
                                #input must be set of strings
```

```
superstring4: cdeabcfghfhed
greedy:       cdeabchedfghf
answer:       fghfhedcdeabc
```

```
In [57]: from itertools import combinations
         from itertools import combinations_with_replacement
         from itertools import permutations
         from itertools import product

         def short_string_test(n_words, words):
             for c in combinations(words, n_words):
                 l1 = len(src.greedy_min_max_contain_string(c))
                 l2 = len(src.min_max_contain_string(set(c)))
                 res.append((1.0 *l1) / l2)
             return res
```

2 Let's "make sure" that aproximation ratio is equal 2 for short strings(a.r.>=2)

It takes some time to find right answers

```
In [62]: letters = 'abcd'
         tmp = [map(''.join, product(letters, repeat=length))\
                 for length in range(1, 4)]
         words = [x for n in tmp for x in n]
         res = []
         sst = short_string_test(3, words)
         print(max(sst))
```

```
1.4
```

For sentences with two words, where words consist of $1, 2, 3$ letters aproximation ratio is ~<= 1.5

Let's implement test described in: http://www.mimuw.edu.pl/~mucha/teaching/aa2008/ss.pdf (2.2 The greedy algorithm)

```
In [44]: def generate_worst_test(k):
             return ['a' + 'b' * k, 'b' * k + 'c', 'b' * (k + 1)]
         test = generate_worst_test(4)
         res = list()
         for c in [generate_worst_test(i) for i in range(4, 300)]:
             l1 = src.greedy_min_max_contain_string(c)
             l2 = src.min_max_contain_string(set(c))
             res.append(((1.0 *len(l1)) / len(l2)))
         print (max(res))
```

```
1.99
```

So we can see on $\{ab^k, b^kc, b^{k+1}\}$ tests algorithm's aproximation ratio converges to 2. We have a little bit better algorighm than in article(in article assumes that strings can not contain each other) that merges strings in one, if one contains another