Complexity Experiment

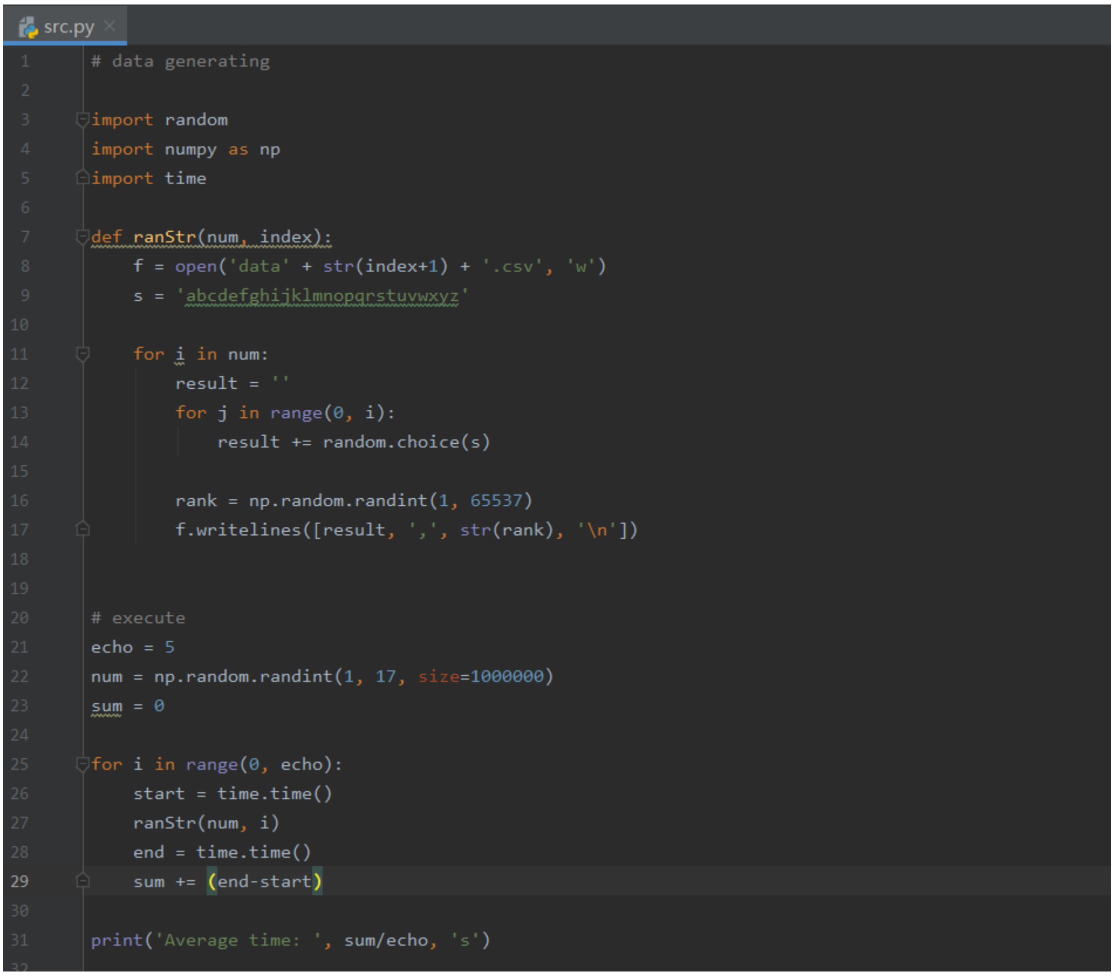
王雨阳 1809853Z-I011-0045

Formerly, we deduce that the main complexity of GUI Version is:

O(NlogN) [insert key] + O(N) [search] + O(0.01N^2) [sort] ~= O(0.01N^2)

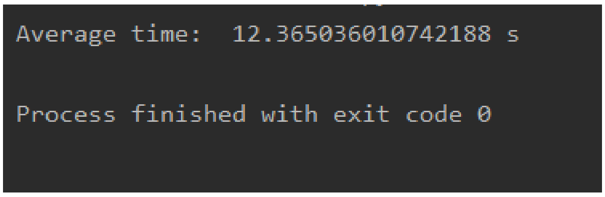
To verify it, I do more experiments by using larger size of data. I write a program in python which can generate 5 data sets which contain 1,000,000 lines of word-rank data.

Source code:



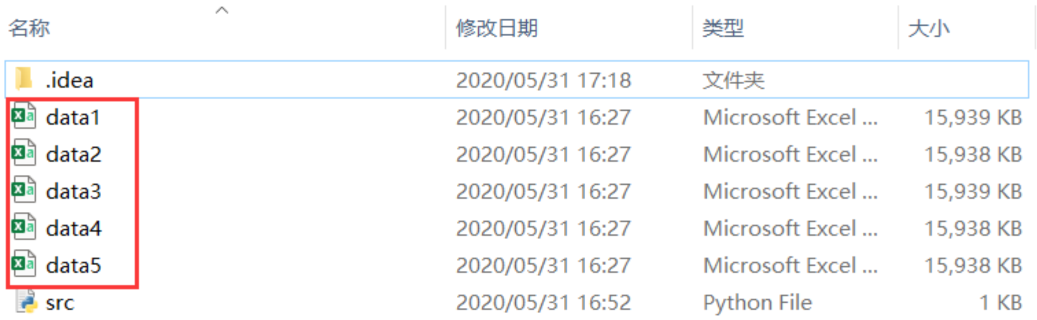
```python
# data generating

import random
import numpy as np
import time

def ranStr(num, index):
    f = open('data' + str(index+1) + '.csv', 'w')
    s = 'abcdefghijklmnopqrstuvwxyz'

    for i in num:
        result = ''
        for j in range(0, i):
            result += random.choice(s)

        rank = np.random.randint(1, 65537)
        f.writelines([result, ',', str(rank), '\n'])


# execute
echo = 5
num = np.random.randint(1, 17, size=1000000)
sum = 0

for i in range(0, echo):
    start = time.time()
    ranStr(num, i)
    end = time.time()
    sum += (end-start)

print('Average time: ', sum/echo, 's')
```

Average time cost:

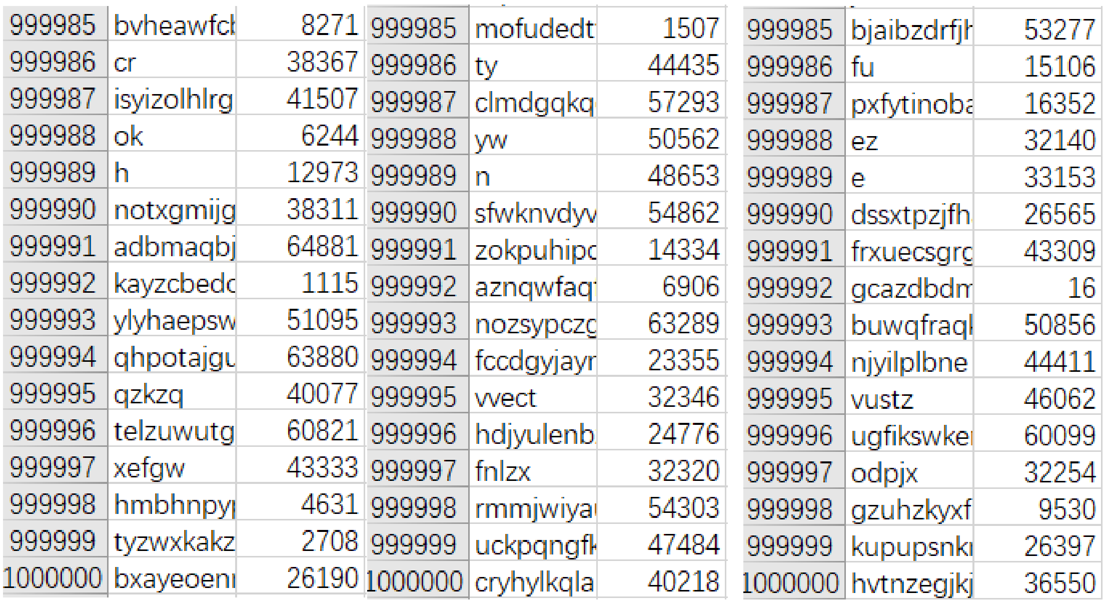


```
Average time:  12.365036010742188 s

Process finished with exit code 0
```

Here are the results:



名称	修改日期	类型	大小
.idea	2020/05/31 17:18	文件夹	
data1	2020/05/31 16:27	Microsoft Excel ...	15,939 KB
data2	2020/05/31 16:27	Microsoft Excel ...	15,938 KB
data3	2020/05/31 16:27	Microsoft Excel ...	15,939 KB
data4	2020/05/31 16:27	Microsoft Excel ...	15,938 KB
data5	2020/05/31 16:27	Microsoft Excel ...	15,938 KB
src	2020/05/31 16:52	Python File	1 KB



999985	bvheawfcl	8271	999985	mofudedt	1507	999985	bjaibzdrfjh	53277
999986	cr	38367	999986	ty	44435	999986	fu	15106
999987	isyizolhlrg	41507	999987	clmdgqkq	57293	999987	pxfytinoba	16352
999988	ok	6244	999988	yw	50562	999988	ez	32140
999989	h	12973	999989	n	48653	999989	e	33153
999990	notxgmijg	38311	999990	sfwknvdyv	54862	999990	dssxtpzjfh	26565
999991	adbmaqbj	64881	999991	zokpuhipc	14334	999991	frxuecsgrg	43309
999992	kayzcbedc	1115	999992	aznqwfaq	6906	999992	gcazdbdm	16
999993	ylyhaepsw	51095	999993	nozsypczg	63289	999993	buwqfraql	50856
999994	qhpotajgu	63880	999994	fccdgyjayr	23355	999994	njyilplbne	44411
999995	qzkzq	40077	999995	vvect	32346	999995	vustz	46062
999996	telzuwutg	60821	999996	hdjyulenb	24776	999996	ugfikswke	60099
999997	xefgw	43333	999997	fnlzx	32320	999997	odpjx	32254
999998	hmbhnpy	4631	999998	rmmjwiya	54303	999998	gzuhzkyxf	9530
999999	tyzwxkakz	2708	999999	uckpqngfk	47484	999999	kupupsnk	26397
1000000	bxayeoen	26190	1000000	cryhylkqla	40218	1000000	hvtnzegjkj	36550

Import Test: O(NlogN)

Test Data	data1	data2	data3	data4	data5
Time	8s	7.5s	7s	7.5s	7s
Size	825350	825354	825380	825381	825397
Percentage	82.5%	82.5%	82.5%	82.5%	82.5%

I combine the 2 arbitrary dicionaries and the program has collapsed, which means the map data structure has a limited storage. (s < = 1,000,000 is better)

Search Test: O(N) [search] + O(0.01N^2) [sort]

We still use character **E** to test:

Test Data	data1	data2	data3	data4	data5
E,1	6s	6s	6s	6s	5.5s
	31441(3.14%)	31696(3.17%)	31525(3.15%)	31803(3.18%)	31639(3.16%)
E,2	5s	5.5s	6s	6s	6s
	31752(3.18%)	31564(3.16%)	31748(3.17%)	31892(3.19%)	31722(3.17%)
E,3	5.5s	6s	5s	5.5s	6s
	31758(3.18%)	31706(3.17%)	31575(3.16%)	31904(3.19%)	31350(3.14%)
E,4	5s	5.5s	5s	5.5s	5.5s
	30923(3.09%)	30701(3.07%)	31160(3.12%)	31188(3.12%)	31057(3.11%)
E,5	5s	5s	4.5s	5s	5s
	28937(2.9%)	28702(2.87%)	28829(2.88%)	28643(2.86%)	28563(2.86%)
E,9	2.5s	3s	2.5s	2.5s	3s
	19355(1.94%)	15	19362(1.94%)	19044(1.9%)	19299(1.93%)
E,12	1.5s	1.5s	1s	1.5s	1.5s
	12191(1.22%)	15	12035(1.2%)	12221(1.2%)	12058(1.21%)
E,15	<1s	<1s	<1s	<1s	<1s
	4885(0.49%)	15	4975(0.5%)	4724(0.47%)	4869(0.49%)
E1,E4	<1s	<1s	<1s	<1s	<1s
	1256(0.13%)	15	1261(0.13%)	1236(0.12%)	1275(0.13%)
E2,E5	<1s	<1s	<1s	<1s	<1s
	1110(0.11%)	15	1166(0.12%)	1095(0.11%)	1074(0.11%)

E1~E5: simulate character who has high appearance rate

E9~E15: simulate character who has low appearance rate

E1, E4 & E2, E5: simulate more than one pattern

Conclusion:

According to experiment result, the percentage of remaining words is far less than 10%. If your dictionary is large enough, this algorithm will be faster a lot than before. The sort complexity will be less than O(0.0009N^2) because only 3% words remain.