isUpperThreeWays

November 24, 2020

To check the time taken for 3 different 'isUpperCase strings' using list comprehenshion, recursion and normal python for loops

Mini in semester Jupyter project to understand the various algorthims - Mrinmoy

[4]:
```
import timeit
import matplotlib.pyplot as plt
import random
import string
```

[5]:
```
#Function to check for time-taken
def time(com):
    return timeit.timeit(com, number=1, globals=globals())
```

[6]:
```
#Method 1 list comprehension
def isUpperLC(string):
    return not any([not ((ord(i)>=65)&(ord(i)<=90)) for i in string])
```

[7]:
```
#Method 1 list comprehension
def isUpperLC_2(string):
    return all([((ord(i)>=65)&(ord(i)<=90)) for i in string])
```

[8]:
```
#Method 2 Recursive implementation
def isUpperRec(string):
    stringIsUpper = (ord(string[0])>=65)&(ord(string[0])<=90)
    if len(string) == 1 :
        return stringIsUpper
    return stringIsUpper&isUpperRec(string[1:])
```

[9]:
```
#Method 3 Normal implementation
def isUpperNormal(string):
        for i in string:
            if not (ord(i)>=65)&(ord(i)<=90):
                return False
        return True
```

[10]:
```
#Given a length of a string this function generates random uppercase strings
def get_random_string_upper(length):
    letters = string.ascii_uppercase
    result_str = ''.join(random.choice(letters) for i in range(length))
    #print("Random string of length", length, "is:", result_str)
    return result_str
```

[11]:
```
#Given a length of a string this function generates random lowercase strings
def get_random_string_lower(length):
    letters = string.ascii_lowercase
    result_str = ''.join(random.choice(letters) for i in range(length))
    #print("Random string of length", length, "is:", result_str)
```

Worst Case

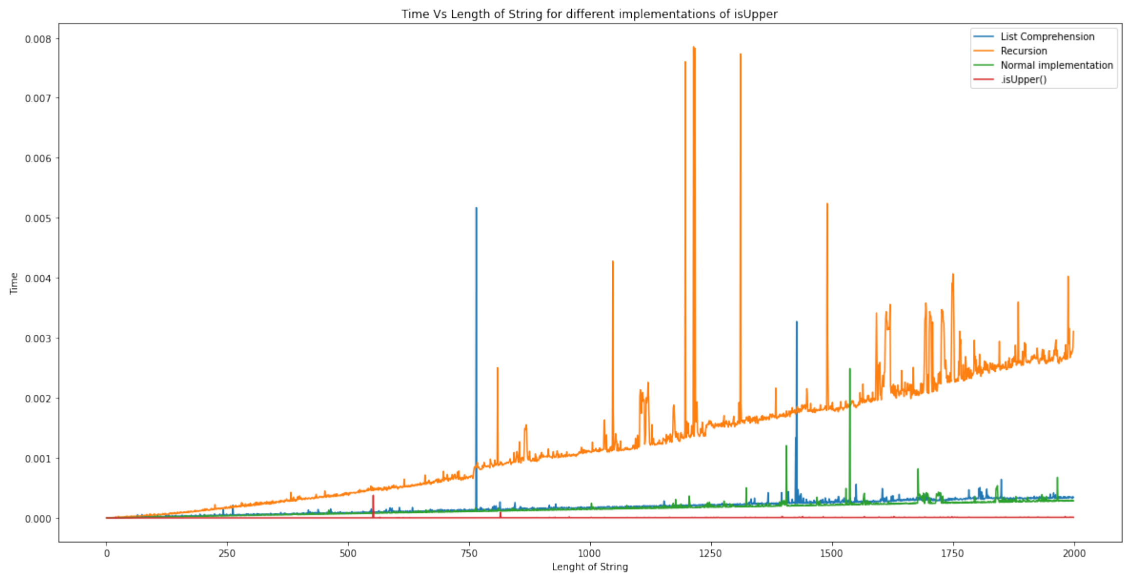
We shall now generate 100 upper case strings of increasing lengths from 1 to 100 and shall save it in the variable strings. Next we save the lengths of the string in lengths

[12]:
```
strings_LC = ['isUpperLC("{}")'.format(get_random_string_upper(i)) for i in␣
 ↪range(1,2000)]
strings_Rec = ['isUpperRec("{}")'.format(get_random_string_upper(i)) for i in␣
 ↪range(1,2000)]
strings_Norm = ['isUpperNormal("{}")'.format(get_random_string_upper(i)) for i␣
 ↪in range(1,2000)]
strings_dot_isupper = ['"{}".isupper()'.format(get_random_string_upper(i)) for␣
 ↪i in range(1,2000)]
lengths = [i for i in range(1,2000)]
```

[13]:
```
times_LC = [time(s) for s in strings_LC]
times_RC = [time(s) for s in strings_Rec]
times_Normal = [time(s) for s in strings_Norm]
times_dot_isupper = [time(s) for s in strings_dot_isupper]
```

[14]:
```
fig, ax = plt.subplots(figsize=(20, 10))
plt.title('Time Vs Length of String for different implementations of isUpper')
plt.xlabel('Lenght of String')
plt.ylabel('Time')
ax.plot(lengths,times_LC,label='List Comprehension')
ax.plot(lengths,times_RC,label='Recursion')
ax.plot(lengths,times_Normal,label='Normal implementation')
ax.plot(lengths,times_dot_isupper,label='.isUpper()')
ax.legend()
```

[14]:
```
<matplotlib.legend.Legend at 0x7f732db0af50>
```



After plotting the graph we can clearly see that .isupper() works in nearly constant time and is the lower bound for all worst cases

The recursive implementation is the most unstable out of all the above implementations, this is because we check if the string length is 1 multiple times.

Interestingly the normal implementation performed better than the List Comprehension, this could be because we are first creating a list of booleans and checking if all the elements are true.

BEST CASE

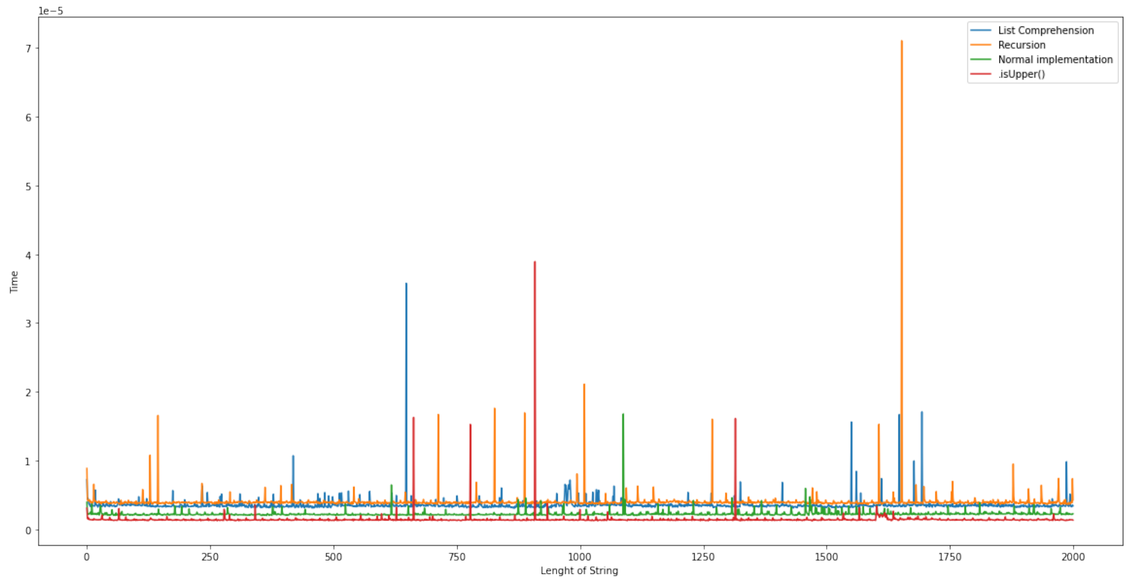
Our best case is when we take all lower case strings and run our functions on them and save the time taken

[21]:
```
strings_LC = ['isUpperLC("{}")'.format(get_random_string_lower(i)) for i in␣
 ↪range(1,2000)]
strings_Rec = ['isUpperRec("{}")'.format(get_random_string_lower(i)) for i in␣
 ↪range(1,2000)]
strings_Norm = ['isUpperNormal("{}")'.format(get_random_string_lower(i)) for i␣
 ↪in range(1,2000)]
strings_dot_isupper = ['"{}".isupper()'.format(get_random_string_lower(i)) for␣
 ↪i in range(1,2000)]
lengths = [i for i in range(1,2000)]
```

[22]:
```
times_LC = [time(s) for s in strings_LC]
times_RC = [time(s) for s in strings_Rec]
times_Normal = [time(s) for s in strings_Norm]
times_dot_isupper = [time(s) for s in strings_dot_isupper]
```

[23]:
```python
fig, ax = plt.subplots(figsize=(20, 10))
plt.xlabel('Lenght of String')
plt.ylabel('Time')
ax.plot(lengths,times_LC,label='List Comprehension')
ax.plot(lengths,times_RC,label='Recursion')
ax.plot(lengths,times_Normal,label='Normal implementation')
ax.plot(lengths,times_dot_isupper,label='.isUpper()')
ax.legend()
```

[23]:
```
<matplotlib.legend.Legend at 0x7f732d32a3d0>
```


1e−5
Time
Lenght of String
List Comprehension
Recursion
Normal implementation
.isUpper()