Finding longest substring in alphabetical order

EDIT: I am aware that a question with similar task was already asked in SO but I'm interested to find out the problem in this specific piece of code. I am also aware that this problem can be solved without using recursion.

The task is to write a program which will find (and print) the longest sub-string in which the letters occur in alphabetical order. If more than 1 equally long sequences were found, then the first one should be printed. For example, the output for a string `abczabcd` will be `abcz` .

I have solved this problem with recursion which seemed to pass my manual tests. However when I run an automated tests set which generate random strings, I have noticed that in some cases, the output is incorrect. For example:

if `s = 'hixwluvyhzzzdgd'` , the output is `hix` instead of `luvy`

if `s = 'eseoojlsuai'` , the output is `eoo` instead of `jlsu`

if `s = 'drurotsxjehlwfwgygygxz'` , the output is `dru` instead of `ehlw`

After some time struggling, I couldn't figure out what is so special about these strings that causes the bug.

This is my code:

```
pos = 0
maxLen = 0
startPos = 0
endPos = 0
```

```
def last_pos(pos):
    if pos < (len(s) - 1):
        if s[pos + 1] >= s[pos]:
            pos += 1
            if pos == len(s)-1:
                return len(s)
            else:
                return last_pos(pos)
        return pos


for i in range(len(s)):
    if last_pos(i+1) != None:
        diff = last_pos(i) - i
    if diff - 1 > maxLen:
        maxLen = diff
        startPos = i
        endPos = startPos + diff

print s[startPos:endPos+1]
```

BTW, this is a homework. I wanted to tag it as one but I couldn't find the homework tag.

python recursion

edited Oct 27 '13 at 14:22

asked Oct 27 '13 at 13:24

Eugene S
915 1 7 27

1 The `homework` tag is now deprecated... do you need recursion? There's plenty of "find the longest in-order substrings from a string" going around at the moment... – Jon Clements Oct 27 '13 at 13:36

@JonClements Thanks for pointing that out. And to your question, no, I don't need it to be recursion but I wanted to try this way. I wonder why it doesn't work in all cases. – Eugene S Oct 27 '13 at 13:51

possible duplicate of Find the longest substring in alphabetical order – Games Brainiac Oct 27 '13 at 13:55

`diff` should be `last_pos(i) - i + 1` . And why do you compare `maxLen` with `diff - 1` ? – ajay

Oct 27 '13 at 13:57

4 Answers

There are many things to improve in your code but making minimum changes so as to make it work. The problem is you should have `if last_pos(i) != None:` in your `for` loop (`i` instead of `i+1`) and you should compare `diff` (not `diff - 1`) against `maxLen` . Please read other answers to learn how to do it better.

```
for i in range(len(s)):
    if last_pos(i) != None:
        diff = last_pos(i) - i + 1
    if diff > maxLen:
        maxLen = diff
        startPos = i
        endPos = startPos + diff - 1
```

edited Oct 27 '13 at 14:20

answered Oct 27 '13 at 14:11

ajay
4,841 2 10 33

1 Yes! You are right about the `diff - 1 > maxLen` comparison being wrong. It forgot to change it after playing with the code. BTW, you can delete the `+1` and `-1` and the result will not change. – Eugene S Oct 27 '13 at 14:19

yes but just for making out the sense of what they are, I kept `+1` and `-1` . – ajay Oct 27 '13 at 14:22

Here. This does what you want. One pass, no need for recursion.

```
def find_longest_substring_in_alphabetical_order(s):
    groups = []
    cur_longest = ''
    prev_char = ''
    for c in s.lower():
        if prev_char and c < prev_char:
            groups.append(cur_longest)
            cur_longest = c
        else:
            cur_longest += c
        prev_char = c
    return max(groups, key=len)
```

Using it:

```
>>> find_longest_substring_in_alphabetical_order('hixwluvyhzzzdgd')
'luvy'
>>> find_longest_substring_in_alphabetical_order('eseoojlsuai')
'jlsu'
>>> find_longest_substring_in_alphabetical_order('drurotsxjehlwfwgygygxz')
'ehlw'
```

Note: It will probably break on strange characters, has only been tested with the inputs you suggested. Since this is a "homework" question, I will leave you with the solution as is, though there is still some optimization to be done, I wanted to leave it a little bit understandable.

edited Oct 30 '13 at 13:33

answered Oct 27 '13 at 13:36

Inbar Rose
13.5k 10 34 68

1 η-reduction: instead of `lambda x: len(x)` , just write `len` . – Gareth Rees Oct 30 '13 at 13:31

@GarethRees You are right, I am surprised I didn't notice that one. Thanks. – Inbar Rose Oct 30 '13 at 13:33

You can use nested `for` loops, slicing and `sorted` . If the string is not all lower-case then you can convert the sub-strings to lower-case before comparing using `str.lower` :

```
def solve(strs):
  maxx = ''
  for i in xrange(len(strs)):
      for j in xrange(i+1, len(strs)):
          s = strs[i:j+1]
          if ''.join(sorted(s)) == s:
              maxx = max(maxx, s, key=len)
          else:
              break
  return maxx
```

Output:

```
>>> solve('hixwluvyhzzzdgd')
'luvy'
>>> solve('eseoojlsuai')
'jlsu'
>>> solve('drurotsxjehlwfwgygygxz')
'ehlw'
```

answered Oct 27 '13 at 13:42

Ashwini Chaudhary

103k 10 107 172

Python has a powerful builtin package itertools and a wonderful function within groupby

An intuitive use of the Key function can give immense mileage.

In this particular case, you just have to keep a track of order change and group the sequence accordingly. The only exception is the boundary case which you have to handle separately

Code

```
def find_long_cons_sub(s):

    class Key(object):
        '''
        The Key function returns
            1: For Increasing Sequence
            0: For Decreasing Sequence
        '''
        def __init__(self):
            self.last_char = None
        def __call__(self, char):
            resp = True
            if self.last_char:
```

```
            resp = self.last_char < char
        self.last_char = char
        return resp
def find_substring(groups):
    '''
    The Boundary Case is when an increasing sequence
    starts just after the Decresing Sequence. This causes
    the first character to be in the previous group.
    If you do not want to handle the Boundary Case
    seperately, you have to mak the Key function a bit
    complicated to flag the start of increasing sequence'''
    yield next(groups)
    try:
        while True:
            yield next(groups)[-1:] + next(groups)
    except StopIteration:
        pass
groups = (list(g) for k, g in groupby(s, key = Key()) if k)
#Just determine the maximum sequence based on length
return ''.join(max(find_substring(groups), key = len))
```

Result

```
>>> find_long_cons_sub('drurotsxjehlwfwgygygxz')
'ehlw'
>>> find_long_cons_sub('eseoojlsuai')
'jlsu'
>>> find_long_cons_sub('hixwluvyhzzzdgd')
'luvy'
```

edited Oct 27 '13 at 14:27

answered Oct 27 '13 at 14:11

Abhijit
26.4k 2 28 79