Exercise 3

Ewa Szczurek
MIM UW

October 19, 2015

Note 1. How to hand in homework:

- Format the title of your email: HOMEWORK; your first and last name; lab number
- Put all exercises for this homework into one single email
- Run all programming code checking that it executes correctly before submitting
- Provide calling examples: examples showing how to call your functions, showing that the program does what it is supposed to do
- Run your calling examples before submitting
- Comment your code
- All solutions which do execute and which do what they are supposed to count, although code elegance and efficiency is much appreciated
- Program either in R or in python
- Submit to `szczurek@mimuw.edu.pl` by the next week.

Last homework 1. Using suffix arrays to find overlaps.

1. Inspect a Python implementation of a suffix tree
2. Using this implementation write a function $overlap(a, b, min\_length)$, that returns the length of longest suffix of a matching a prefix of b that is at least min_length characters long. If no such overlap exists, return 0.
3. What should be an efficient procedure if we wished to use suffix tree for finding overlaps of multiple strings?

Solution example 1. Worst case $O(m^2)$, where m is the length of the longest overlap.

```
def overlap(a,b,min_length):
   tree = SuffixTree(a)
   overlap = 0
   prefix = ''
   for i in b:
      prefix += i
      if tree.hasSuffix(prefix)==True and len(prefix)>=min_length:
         overlap = len(prefix)
   return overlap
```

Solution example 2. Worst case $O(m)$, where m is the length of the longest overlap.

Add one more method to the class SuffixTree:

```
def overlap(self, s):
        """Find the length of a longest suffix
        matching a prefix from string s."""
        cur = self.root
        i = 0
        """Variable level stores the length of the longest suffix
        matched so far to a prefix of s,
        i.e., where was the last '$' passed by """
        level = 0
        while i < len(s):
            c = s[i]
            if c not in cur.out:
                """Can't match the next character in the string s.
                We are in some node of the suffix tree """
                if '$' in cur.out: #A suffix was matched with a prefix of s
                    return i
                else:
                    """Return the stored length of the longest suffix
                    which matched a prefix of s. """
                    return level
            child = cur.out[s[i]]
            lab = child.lab
            j = i+1
            while j-i < len(lab) and j < len(s) and s[j] == lab[j-i]:
                # Go down the label.
                j += 1
            if j-i == len(lab): #Finished processing the entire edge label
                cur = child
                if '$' in cur.out:
                    # If there is a terminator, update the level value.
                    level = i
                i = j
```

```
            else:
                """Can't match the next character in the string s.
                We are somewhere in the edge label. """
                if lab[j-i] == '$':
                    """The next character on the label is a '$'.
                    A suffix was matched with a prefix of s. """
                    return j
                """Return the stored length of the longest suffix
                which matched a prefix of s."""
                return level
        """Finished processing the string s.
        Return the stored length of the longest suffix
        which matched a prefix of s."""
        if '$' in cur.out:
            return i
        return level
```

And the function *overlap* becomes

```
def overlap(a,b,min_length):
    """Returns the length of longest suffix of a matching a prefix of b
    that is at least min length characters long.
    If no such overlap exists, return 0
    """
    aTree = SuffixTree(a)
    length = aTree.overlap(b)
    if length < min_length:
        return 0
    return length
```

Exercise 1. Assembly with de Bruijn graphs.

1. Inspect the code for building, visualising, and using de Bruijn graphs over strings
2. Build a de Bruijn graph for the string "to_every_thing_turn_turn_turn_there_is_a_season" using k-mer length 3
3. What is the superstring that can be read off an eulerean walk over this graph?
4. Build a de Bruijn graph for the same string using 4-mers, and print the superstring.
5. Generate a dot file for this graph using the *toDot* function
6. Generate a picture of this graph
    - dot -Tpdf dotfile.dot -o picture.pdf
7. Now generate a picture with weighted edges (not a multigraph).

Homework 1. Shortest common superstring with repeats in Python

1. Implement the Greedy-SCS algorithm in Python
2. For a string it_was_the_best_of_times_it_was_the_worst_of_times generate all substrings of length k, and run the algorithm for min overlap length l where

   a) $l = 3$, $k = 7$

   b) $l = 3$, $k = 10$

   c) $l = 3$, $k = 13$

Hint: you may wish to inspect the example implementation of the brute-force SCS algorithm.