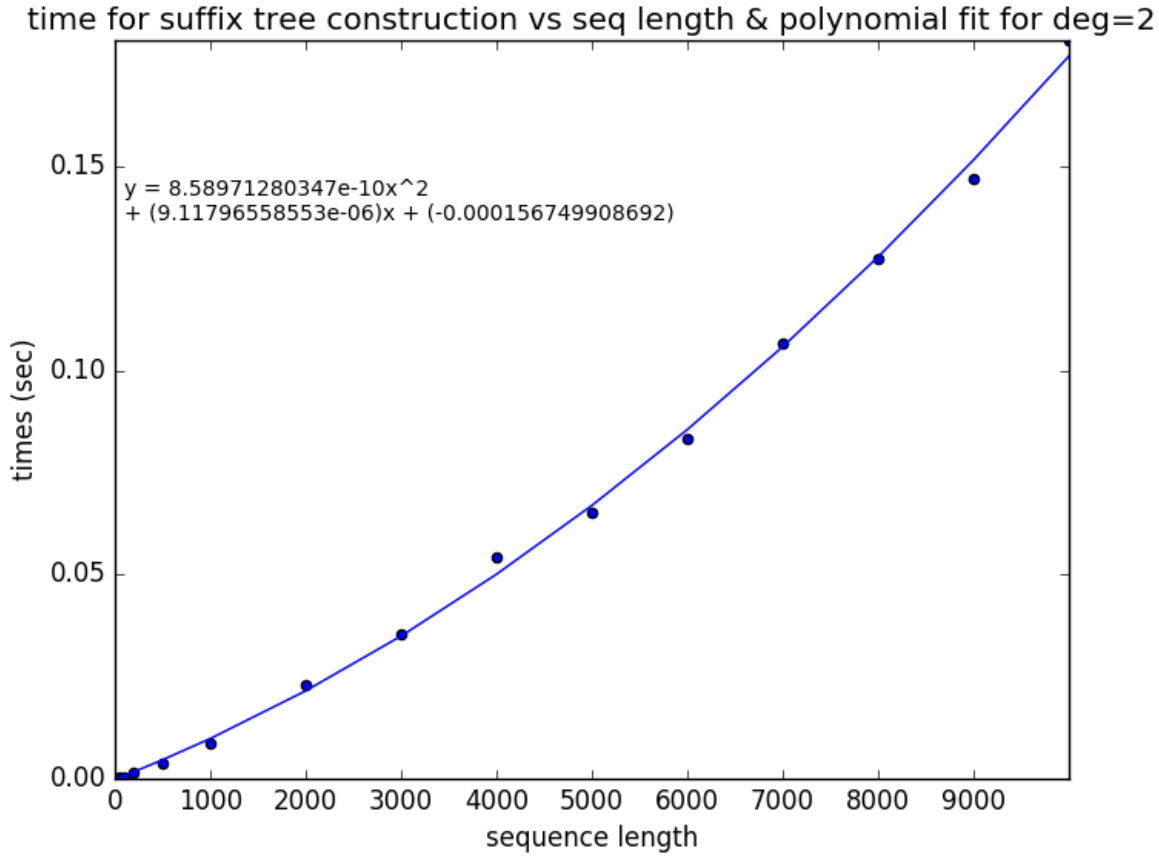
Computational Genomics – Suffix trees

Liza Dashevski 26/11/2016

1 Suffix tree construction - Task A



- The degree of the polynomial is 2.Thus, we can conclude that the running time of the procedure is $O(n^2)$, n is the size of a sequence the suffix tree is build from.
- estimated val:0.329881271952 vs actual val: 0.32433361040995806

2 Theoretical question - Mapping with mismatches - Task 3

We will solve the problem for k mismatches so all we have to do is then apply it for k = 2.

Problem definition

Input: a sequence T of length n, a pattern P of length mand a threshold k.

Output: All sub strings of T with length m matching P with k maximal number of mismatches.

General explanation

We will use the Kangaroo method described in the article "Efficient string with K mismatching, Landau, G.M., and Vishkin, U., Theoret. Comput Sci 43, 1986, pp. 239-249"

The Kangaroo method can be explained as follows, let a be a position in sequence T such that $t_1t_2t_3..T_a = p_1p_2p_3..P_a$ and such that for position $a+1$ $t_{a+1} \neq p_{a+1}$. Thus we don't have to examine $t_1...t_{a+1}$ with $p_1...p_{a+1}$ and continue to the suffix beginning with t_{a+2} and p_{a+2}. We will continue the process so that whenever we come across a perfect matching to pattern P we skip it. The process will stop when k+1 mismatches found.

First we will construct a suffix tree from sequence T and pattern P. Second we will find the lowest common ancestor of two leafs X and Y. Let the leaf node corresponding to the sub string of T be denoted as X. and the leaf of the pattern denoted as Y. Each time we find a mismatch we count it and do second step for the sub pattern after it with what is left of T.

Algorithm 1 Kangaroo pamming with mismatches

1: **procedure** KANGAROO(T,P,K)
2: $ST = constructST(T)$
3: $matchCounter = 0$
4: $resultMatches = []$
5: *while loop*:
6: **if** matchCounter = k **then**
7: **goto** *while loop*.
8: **close**;
9: $(P_{afterMismatch}, T_{afterMismatch}) = findCommonAncestor(ST, T, P)$
10: add to resultMatches
11: $P = P[P_{afterMismatch} :]$
12: $T = T[T_{afterMismatch} :]$
13: **if** $P \neq empty$ **then**
14: $matchCounter = matchCounter + 1$
15: **goto** *while loop*.

Algorithm 2 Find Comon Ancesrot

```
1:  procedure FINDCOMMONANCESTOR(ST,T,P)
2:      if ST.root = T or P  then
3:          returnST.root
4:      if otherwise then
5:          treeNodePtrleft =findCommonAncestor(ST.root->left , p , q)
6:          treeNodePtrright =findCommonAncestor(ST.root->right , p , q)
7:          if treeNodePtrleft and treeNodePtrright not null  then
8:              returnST.root
9:          if treeNodePtrleft not null  then
10:             return treeNodePtrleft
11:         if treeNodePtrright not null  then
12:             return treeNodePtrright
```