Bioinformatics I
WS 15/16
Tutor: Alexander Seitz

1	2	$\sum$ (6+2)

Jonas Ditz
& Benjamin Schroeder

Blatt 8

(Abgabe am 14. December 2015)

Theoretical Assignment - *Suffix tree construction and application*

a - *Suffix tree construction*

i

If one uses the naive approach to construct the suffix tree of “CTAGTAGCAG”, the result would look like Figure 1.

ii

b - *Main data table of WOTD*

c - *Suffix tree application*

Theoretical Assignment - *Runtime and space complexity of suffix trees*

a

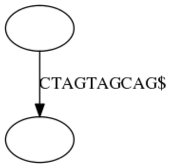
Assume a text T of length n with n times the letter “a”. If one build a suffix tree for T using WOTD, each node will have just one c-group with all remaining suffixes in it. So evaluating the root node, one has to compute the longest common prefix of n suffixes, of $n-1$ suffixes for the second node and so on. In each c-group the shortest suffix is of length 1, so for each node $\overline{u}$ there are just $|R_a\{\overline{u}\}|$ numbers of comparisons.

Since T is of length n this will lead to an overall runtime of $\sum_{i=1}^{n} i = \frac{1}{2}n(n+1)$, which is in $O(n^2)$. $\square$

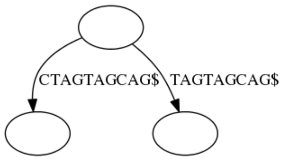
b

Suffix links are defined as an auxiliary edge that point from branching node $\overline{bw}$ to the branching node $\overline{w}$, if it exists and to the root otherwise. If we now defined a suffix link, such that it points into the other direction (i.e. from branching node $\overline{w}$ to branching node $\overline{bw}$) gives us an efficient method to find maximal unique matches. If we now find a branching node that indicates a unique match (for definition see script) and there is a suffix link that point to another branching node, we know that the current branching node is not a maximal unique match (because it is not left maximal). So we just have to follow the suffix links until we reach a branching node without an outgoing suffix link and we have found a maximal unique match.

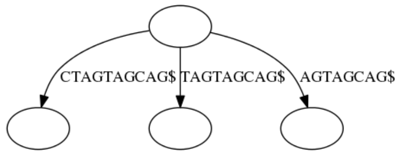
C



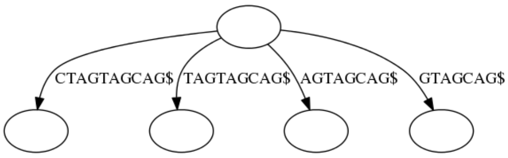
(a) step 1



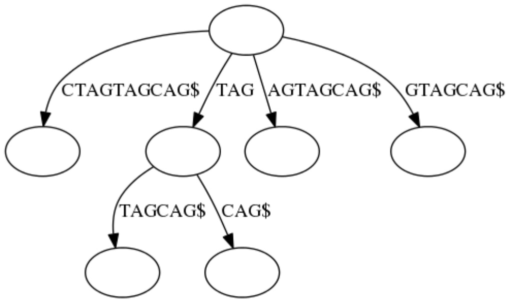
(b) step 2



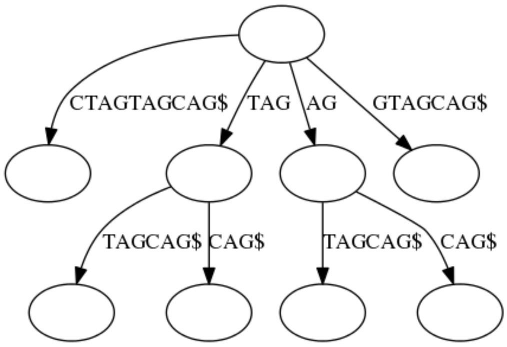
(c) step 3



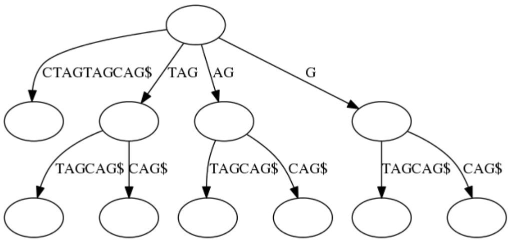
(d) step 4



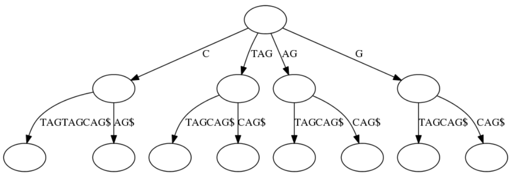
(e) step 5



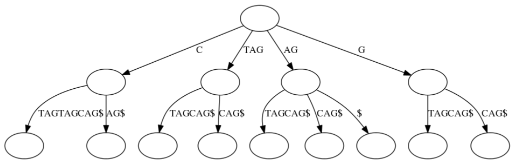
(f) step 6



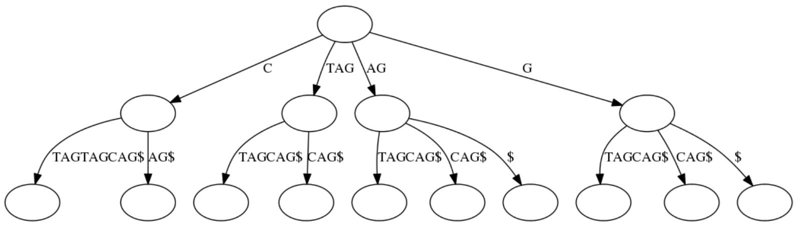
(g) step 7



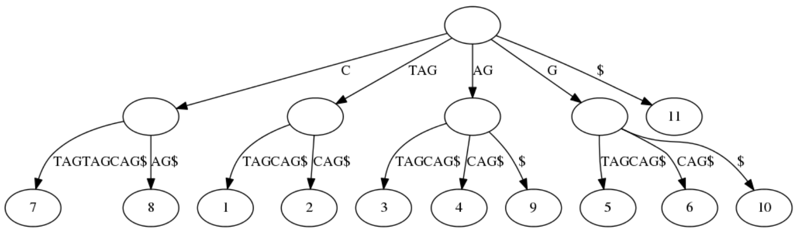
(h) step 8



(i) step 9



(j) step 10



(k) step 11

Figure 1: All steps of the naive implementation of suffix tree construction for the string "CTAGTAGCAG".