Algorithms for Massive Data Sets

Student name and id: Alexander R. Johansen (s145706)

Collaborator name(s) and id(s): Jonas (s142957)

Hand-in for week: 6

---

1 Sort problem

The $Search?(P_1, P_2)$ Problem can be solved in $O(nlgn)$ space using two suffix trees and a 2-D range reporting data structure.. The query time, in a format of $m + f(n) + occ$ can be solved in $f(n) = O(lgn)$ which gives an algorithm that runs in $O(m + lgn + occ)$ that validly can solve the "search?" problem.

1.1 Analysis of datastructure and algorithm

Suppose two suffix trees, built on S and S^R where ^{R} stands for reversed. The first one indicating the *startingPosition* of a given substring, the reversed indicating an *endPosition* with an added character of a given substring. The point in which these positions collide, $endPosition = startingPosition$ has to be the *mergingPlace* in the text where two sub strings merge, denoted here by a "|", such that $P_1?|P_2$.
In order to find the starting position of the $P_1?P_2$, given the *mergingPlace* at "|" in $P_1?|P_2$ this can be found be subtracting $P_1.length + 1$, where +1 denotes the ? letter. Such that *startingplace* of the concatenated string $P_1?P_2$ is $mergingPlace - (P_1.length + 1)$
Suppose the *mergePlace* could be represented as a point in a X,Y graph in such case a 2D range reporting data structure could evaluate were concatenated patterns starts.
Suppose each sorted suffix from the first suffix tree built on S was a X-value, and each reverse sorted suffix from tree S^R was a Y-value and at their original *startingPosition* or *endPosition*, they intersect on the X,Y graph.
Appendix A illustrates the $BANANAS\$$ example.
Note: In order to make it fit with an extra character, the reversed tree started with an extra \$.

1.2 Specific data structure

Given a string S
Compute the reverse of the String with a padded \$ to match for the ? letter. Build suffix tries on both keeping track of the sorted order of the suffix'es. This will leave all of the end notes in the suffix tree, \$ with two values. Their original placement in the string and their sorted placement. Use the sorted placement as X,Y values in a 2D range tree, where the points are given by the matching placement in the string between S and S^R, illustrated in appendix A. Further each node in the suffix contains the range of the sorted leaves below it. Space: $O(nlgn)$

1.2.1 Evaluation data structure

2 x Suffix trees: $O(n)$
2-D Range tree: $O(nlgn)$
Total: $O(nlgn)$

1.3 Specific algorithm

Given two strings, P_1, P_2.
Match P_2 down the suffix tree for S, if it does not match the algorithm terminates with [], if it does then at the node of being fully matched; extracted the indexes of the sorted order of the suffix'es below the node (as pointed earlier would be held in the nodes), do the same for P_1 in S^R. The extracted indexes are now the X and Y range, use these in the 2-D range tree, report all occurences.

1.3.1 Evaluating algorithm

Suffix search S: $O(|P_2|)$
Suffix search S^R: $O(|P_1|)$
2-D range tree: Step 2: $O(lgn + occ)$
Total: $O(m + lgn + occ)$

1.4 Appendix A

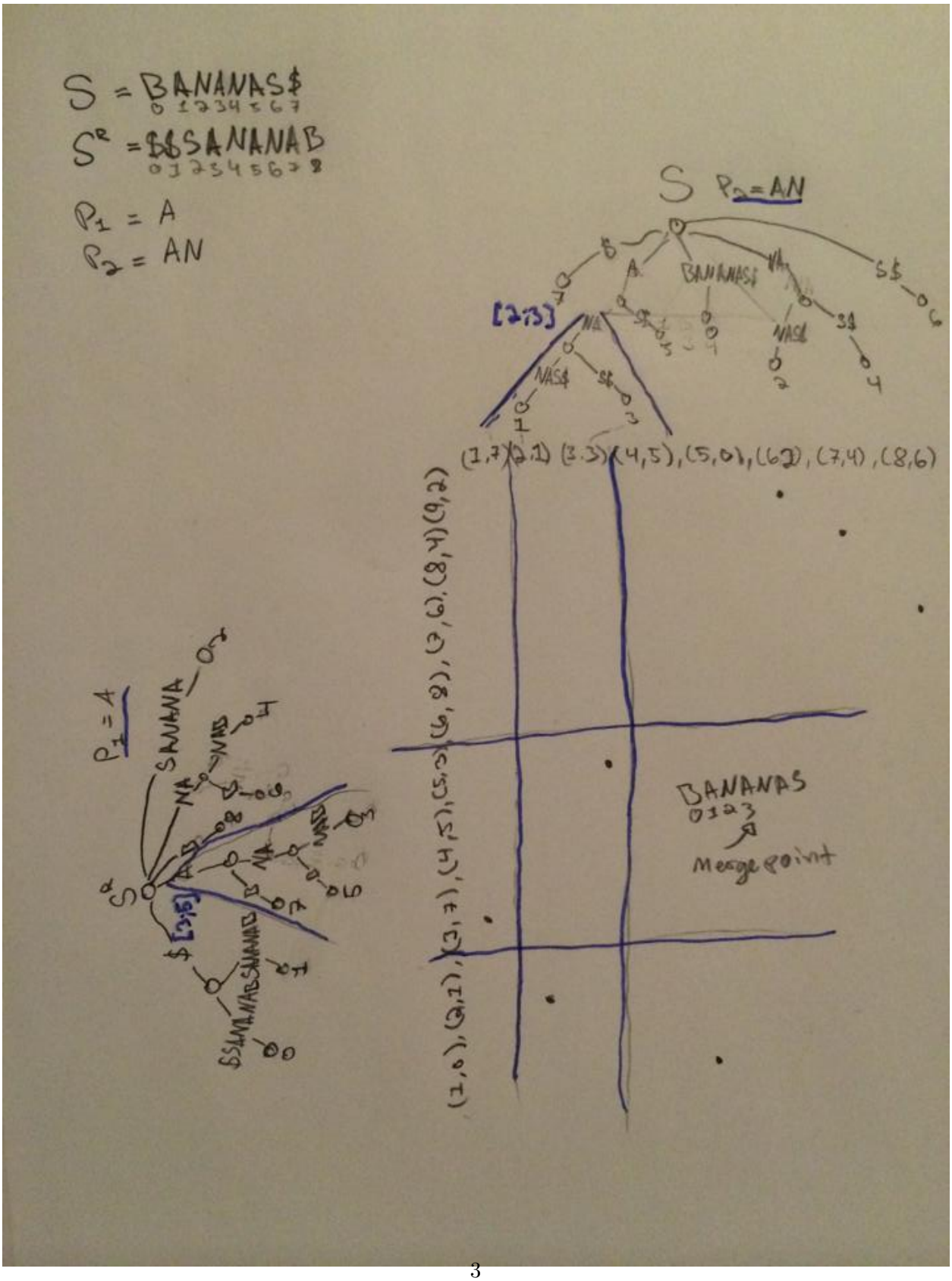

Figure 1: figure A.