Purely functional palindromic trees
Iteration III results

Timur Khazhiev
Innopolis University
t.khazhiev@innopolis.ru

Nikolai Kudasov*
Innopolis University
n.kudasov@innopolis.ru

ABSTRACT

This is the results of BS-3 Project and it contains a general project results so far.

CCS CONCEPTS

• **Theory of computation → Data structures design and analysis**;

KEYWORDS

palindrome, eertree, purely functional, persistent

1 NAIVE IMPLEMENTATION OF INFINITE TREE

Simple (with basic suffix links) infinite tree was implemented. Basically structure is represented by two root nodes for odd and even palindromes. Node contains length, palindrome (in a form of string), list of pairs (symbol, node) where nodes adjacent to current node by that symbol (edges list) and a link (node) to max palindrome suffix.

Edges can be trivially defined. Link node is searched by value, to get that value we simply look up through all suffixes of current palindrome.

To visually check that it works we can print out ten top (lexicographically ordered) palindromes of tree, print their relations and so on. Code for that implementation can be found in GitHub repository[3].

2 STRING REPRESENTATION BASED ON INFINITE TREE

Eertree from original work[5] of some string S is actually a subtree of infinite tree. To build that subtree we do pretty much the same, but instead of building tree, we will refer to infinite tree and simply append found node to list of palindromes.

To check that it works, we can calculate number of rich palindromes of length n and compare it with actual sequence from OEIS [6]. A sequence of length n is called rich if it contains, as subsequences, exactly n distinct palindromes. That also shows that infinite tree is build properly (not only by observing, but using sequence).

Code for that implementation can be found in GitHub repository in older commits[1].

3 DOUBLE ENDED TREE

Double ended tree is an extension of basic tree explained in previous iteration[2] (which was called "two sided tree"). Remind that basic tree keeps information about maximum palindrome suffixes, that helps appending a new symbol. To allow prepend function we need to keep information maximum palindrome prefix.

That extension helps us to implement merge operation. That was also implemented a using merge algorithm described in previous iteration report and source code can be founded in GitHub repository [3]. That was also checked by comparing palindromes of $merge(tree(S_1), tree(S_2))$ and $tree(S_1 + S_2)$ where S_1, S_2 are string.

4 NEW MERGE ALGORITHM FOR MERGING TREES

During this iteration new approach of merging was observed. Previous merge approach was checking through all suffixes (maxSuffix, maxSuffix of maxSuffix, maxSuffix of maxSuffix of maxSuffix...) and same with prefixes. For example in case of string of form "$aaa \dots aaa$" number of suffixes (and prefixes) is length of that string. And for each suffix of first string (and prefix of second) we are looking for palindromes by expanding out of that suffix (or prefix). We will do a lot of checks.

Another approach comes from combining naive implementation (prepending/appending string to another string symbol by symbol) and fact that new palindromes are located around maxSuffix_1 and maxPrefix_2 ($_1$ can be read as "of first string"). In a simple way it can be described as followed (if we append to first string): do append and work with maxSuffix until maxSuffix_{new} went beyond maxPrefix_2 and $\text{maxSuffix}_{new} \not\supset \text{maxPrefix}_2$. As in previous algorithm we also add palindromes of second string to merged tree and update maxPrefix and maxSuffix.

In worst case it will operate as simple naive implementation with complexity $O(N)$. Also we can use fact from another Rubinchik's paper [4] that showed that random word of length n contains, on expectation, $\Theta(\sqrt{n})$ distinct palindromic factors. So if length of first string is n and length of second is m, on expectation, their concatenation gives us $\Theta(\sqrt{n+m})$ palindromes. From that we exclude palindromes that already processed separately. So $O(\sqrt{n+m})$ is complexity for average case and that is interesting for parallel computation of palindromes of chunks of some string. But it might be better.

5 OBSERVATION ON MAXSUFFIX LENGTH

We observed that average length of maximum palindromic suffix of a random string of length n for alphabets of size $\sigma \geq 2$ is $\Theta(1)$. That gives some room for optimization (e.g. logic branch). For example appending can be done $\Theta(1)$ on average and $O(\log n)$ worst case.

REFERENCES

[1] Timur Khazhiev. 2019. Iteration II detailed report. https://github.com/khazhix/ds-project/blob/iteration_ii/Iteration_II.pdf. (2019).

*Project supervisor.

[2] Timur Khazhiev. 2019. Iteration II report. https://github.com/khazhix/ds-project/blob/iteration_ii/Iteration_II.pdf. (2019).
[3] Timur Khazhiev. 2019. Naive implementation of infinite tree over binary alphabet. https://github.com/khazhix/ds-project/blob/sketch-2019-03-04/implementation/binary_eertree.hs. (2019).
[4] Mikhail Rubinchik and Arseny M Shur. 2016. The number of distinct subpalindromes in random words. *Fundamenta Informaticae* 145, 3 (2016), 371–384.
[5] Mikhail Rubinchik and Arseny M. Shur. 2018. EERTREE: An efficient data structure for processing palindromes in strings. *European Journal of Combinatorics* 68 (2018), 249 – 265. https://doi.org/10.1016/j.ejc.2017.07.021 Combinatorial Algorithms, Dedicated to the Memory of Mirka Miller.
[6] "Jeffrey Shallit". 2013. The On-Line Encycloped Integer Sequences. https://oeis.org/A216264. (Mar 2013). Number of rich binary words of length n.