Extending Range Shortest Unique Substring queries

Gary Hoppenworth

What paper am I extending?

- Range Shortest Unique Substring Queries by Abedin, Ganguly, Pissis, and Thankachan (SPIRE 2019)
- https://ir.cwi.nl/pub/29113

What am I trying to do?

- The **Range Shortest Unique Substring** queries paper gives an algorithm that answers queries about the existence of unique substrings within a given *range* of a text T.
- I want to use the ideas in this paper as well as the methods learned in this class to solve the shortest unique substring query problem where, given a position t in the text T, you must find the shortest unique substring covering T.
- I call this problem **Position Shortest Unique Substring**.
- This problem is interesting because you may be interested in a specific position in the text as opposed to a range.
- **I want to design an algorithm (randomized or nonrandomized) that is better than the naive solution to this problem.**

How is SUS done today?

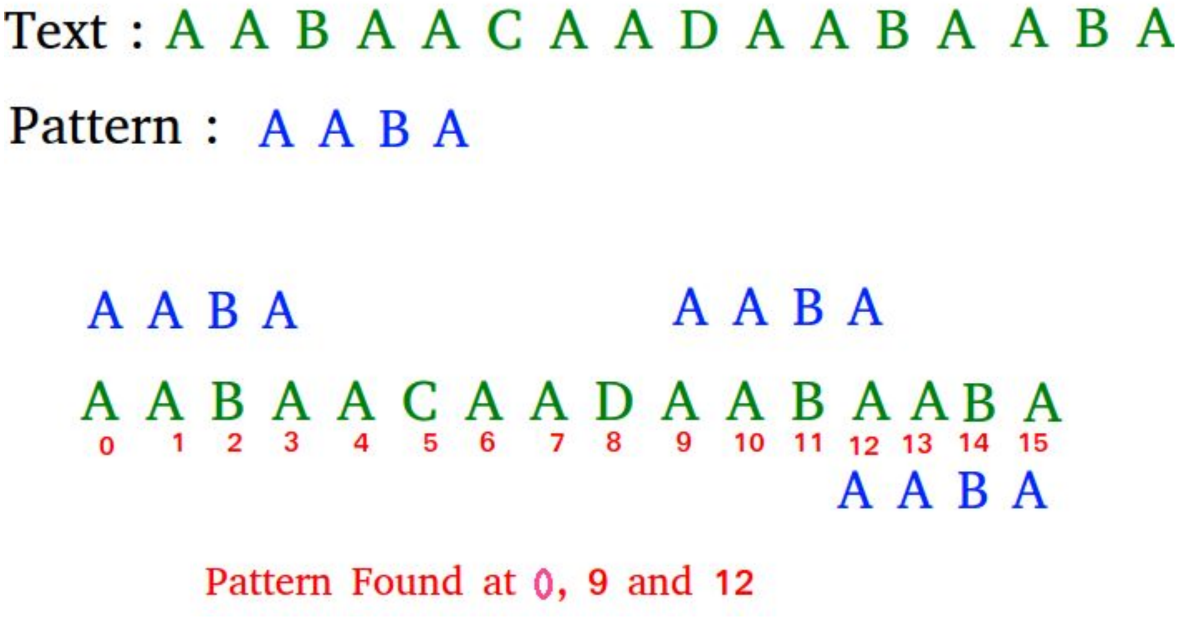
- The classic shortest unique substring problem is typically solved with a suffix tree data structure in O(n) time and O(n) space.
- The algorithm described in the paper I am extending makes use of
    - Range LCP data structures
    - Heavy path decomposition
- These are two tools widely used in algorithm design and analysis, and specifically string algorithms.
- These data structures have not yet been applied to the Shortest unique substring problem covering a specific position in the text. Current techniques will require generalization to solve this new problem.

What is new in my approach?

- The problem I am considering was not considered in the literature before.
- The algorithm for Range shortest unique substring does not imply an algorithm for the Position shortest unique substring problem.
- I will now describe some of the algorithmic techniques I will use in my project.

Rabin-Karp Rolling Hash

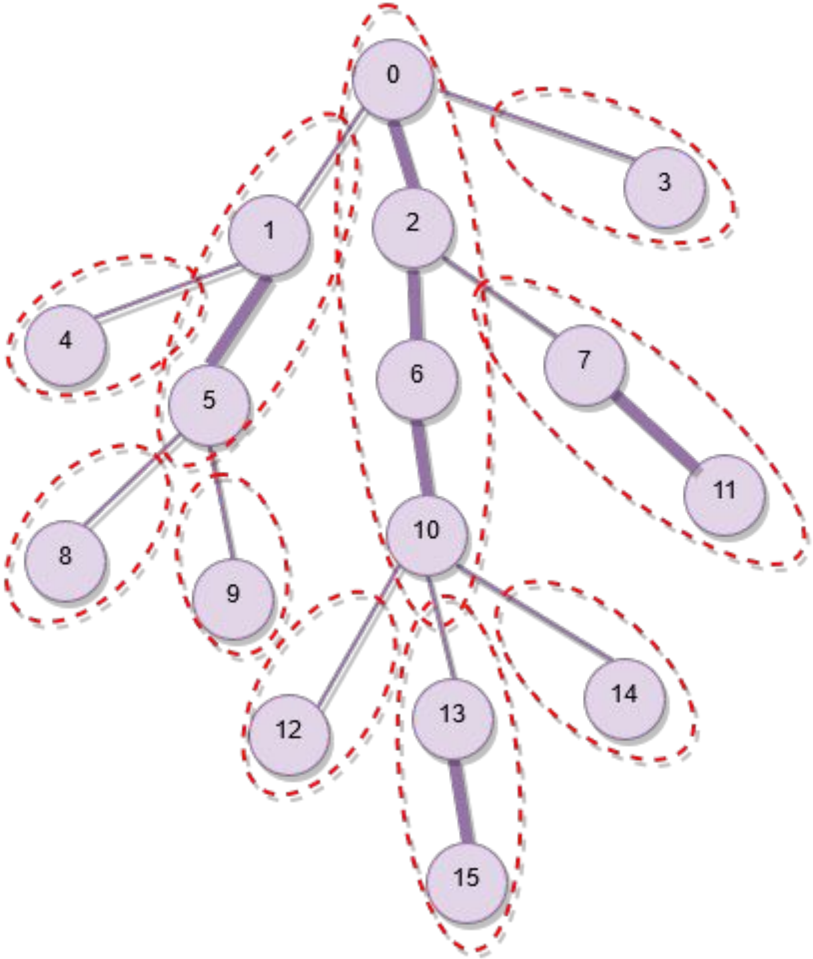
- The Rabin-Karp rolling hash allows us to match patterns in a text in linear time.
- This randomized algorithm is a powerful tool for pattern matching problems like shortest unique substring.
- Takes O(nm) time where n is the length of string and m is length of the pattern



Pattern Found at 0, 9 and 12

Heavy Path Decomposition

- Heavy path decomposition can decompose a tree into heavy and light paths, and gives a stronger upper bound on the number of heavy (or light nodes) on any given root to leaf path.
- This algorithmic strategy combined with the suffix tree has many uses in string algorithms.



Who cares?

- **Computational Biology:** The shortest unique substring problem has important applications to computational biology. An algorithmic solution to my problem would also be useful to people in the field of bioinformatics.
- **String Algorithms Theory:** The shortest unique substring problem is of interest to researchers studying string algorithm design. My new problem will add to the literature on this fascinating problem.

What are the risks?

- Because this is a theory paper, there are little to no risks in this research.
- Of course, I may not be able to solve it, but in string algorithms there is always the right data structure or strategy, you just have to find it.

How much will it cost?

- It won’t cost anything besides my time and effort.

How long will it take?

- It will take 1-2 hours to read the paper fully and understand the required background.
- It will probably take 5-10 hours to try to solve the Position Shortest Unique Substring Problem.

Mid-term check for success

- I should be able to present a naive solution to this algorithm.
- I should present straightforward applications of existing data structures to this problem.
- I should have read the paper and background paper and have written in my report descriptions that reveal my understanding / knowledge of these ideas.

Final checks for success

- A complete, sophisticated algorithm (randomized or nonrandomized) that solves the Position Shortest Unique Substring problem in a competitive time.
- I should give a literature review that displays existing algorithms that can solve this problem and their complexity, in order to compare the performance of my algorithmic solution.
- My project will be a success if my solution is faster or as fast as existing algorithmic solutions to the Position Shortest Unique Substring problem.