Project 2 — Suffix tree construction

You should implement a suffix tree construction algorithm. You can chose to implement the naive O(n^2)-time construction algorithm as discussed in class or McCreight's O(n) construction algorithm. Write a program, `search-st` using the suffix tree exact pattern search algorithm (similar to slowscan) to report all indices in a string where a given pattern occurs.

The `search-st` program should take the same input as the programs from project 1 and should output (almost) the same SAM files. Because a search in a suffix tree is not done from the start to the end of the string the output will be different, but if you sort the output from the previous project and for this program, they should be identical.

Consider how you would serialise a constructed suffix tree such that you can load it again in a search application. You do not have to implement this, but you might want to do this in project 5; project 2 is one way you might use, but it is worth considering alternatives.

Evaluation

You should upload your solution to Blackboard before the lecture **5/3**.

The solution you upload should contain code for building the `search-st` program (with the appropriate build tools for the programming language you have chosen), and a one-page PDF file where you address the following questions:

- Insights you may have had while implementing and comparing the algorithms.
- Problems encountered, if any.
- An experiment that verifies the correctness of your implementations.
- An experiment that verifies that your implementation of `search-st` uses no more time than O(n) or O(n^2) (depending on algorithm) for constructing the suffix tree and no more than O(m) for searching for a given read in it. Remember to explain your choice of test data. What are "best" and "worst" case

inputs?

Handin of project 2