Aaron Lopez Prof. Nikolaev

CSCI 49201 Spring 2019

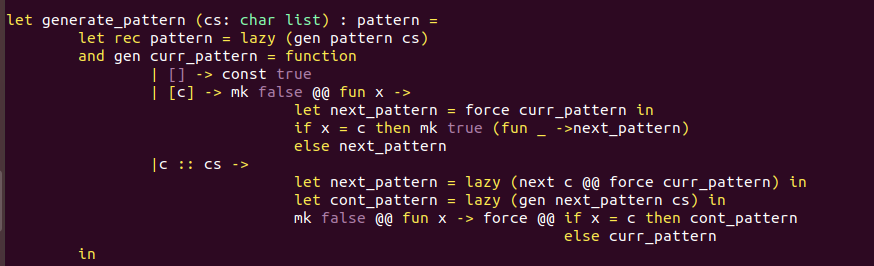
Knuth-Morris-Pratt algorithm in 100% Functional Ocaml

The original proposal for this project is followed as such:

The biggest issue with KMP is its use of a table for tracking positions and words already searched to prevent backtracking. Any backtracking would increase the rate of complexity to at least O(n^2). Ocaml does not allow tables or internal states. The goal is to functionally recreate the same purpose of the KMP table.

The Knuth-Morris-Pratt (KMP) algorithm is able to search for a single or multiple occurrence of a string within a larger volume of text in O(n) time. Being able to find the string in O(n) makes it an efficient algorithm. The main computational power behind the algorithm comes from a “pre-processing table”. This table looks at the pattern to be matched, and for each character calculates the length of the largest proper suffix and prefix of the substring up to that character. For example, if w = “aababaagoda”, and the table is currently processing the fifth “a” at w[6] if w was a 0-based char array, the length of the largest prefix and suffix within substring “aababaa” is seven. This is then associated with index 6 of the string.

By creating this “pre-processing” table, it eliminates the vast majority of backtracking, and removes worst case scenarios that cause naive algorithms to approach O(n^2). The best way to re-implement a similar paradigm while staying within functional coding is to use the “Lazy” module.

 The Lazy module allows the suspension of computation until called later by a different portion of the code. This allows the computation of a set of chars similar to how a table is constructed without relying on a non-functional structure. In the generate\_pattern function, the pattern structure type is used to hold both the flag for signaling the end of a pattern, and the function for the next pattern. The recursive pattern function creates an unevaluated thunk for forward referencing, effectively suspending the pattern. The function will match a list of characters depending upon three cases. The empty list is special, and will result in a pattern that always matches, the second case is a single character, where a forced evaluation of the current pattern occurs, thus branching it off dependinf on the evaluation of the next pattern, and the third case. The third case is if more than one character is matched, then it applies the first character to the current pattern, then recursively generates the next step in the pattern on the tail. This provides a much needed optimmization in solving string pattern matching algorithms in Ocaml.

Lazy is not safe for parallelization, and is not computationally faster than iterative programming algorithms. To make Lazy safe for parallelization, locks need to be added before and after any call made to lazy for suspension of computation, slowing down the computational process further. Currently, after five test runs per paradigm, the imperative scheme computes 10-40% faster than the functional scheme for both files and string arguments.

The executable should take two parameters alongside the run. These two parameters would be [string] [text]. [string] is the string to look for within the body of [test]. The parameters can either be strings, denoted by the quotation marks “ “ on each end, or with a text file ending in “.txt” .

For example :

``` {./OKMP “we are” “we are the crystal gems”} will be considered two strings.

``` {./OKMP we wearethecrystalgems } will be considered two strings as well.

``` {./OKMP we.txt werethere.txt } will be two text files

``` {./OKMP “we.txt” “we are there.txt” } will also be two text files.

```` {./OKMP we “we are there.txt” }will be one string and one text file.

Once the KMP algorithm is processed, the data will be outputted into the command line. A secondary goal will be to output the result into a text file as well, in case of larger inputs with multiple positions found. The output will only output the exact char location of the string. Another secondary goal will be to output the line as well by counting the number of new line characters there are in the text file.