

## 1. Resumen

Los algoritmos de búsqueda de subcadenas son una clase de algoritmos de cadenas que tratan de buscar un(os) *patrón(es)* en una cadena o texto. Una manera ingenua de hacerlo es mediante fuerza bruta; la idea es ir deslizando el patrón sobre el texto de izquierda a derecha, comparándolo con las subcadenas del mismo tamaño que empiezan en cada carácter del texto. Lo cual en el peor de los casos toma una complejidad de tiempo de  $O(nm)$ .

Normalmente cuando es una cadena grande se busca mejorar el desempeño del mismo. El algoritmo de Knuth-Morris-Pratt (KMP) es útil para buscar un solo patrón (con longitud  $m$ ) en un texto (de longitud  $n$ ) con una complejidad en tiempo de  $\Theta(n + m)$ .

En este trabajo se hará énfasis en el algoritmo KMP centrado en la programación funcional.

Primero se dará una especificación formal (en el lenguaje de programación Haskell) del algoritmo de fuerza bruta con complejidad  $O(nm)$  y por medio de razonamiento ecuacional lo que se busca es llegar a derivar el algoritmo KMP el cual tiene complejidad  $\Theta(n + m)$ .

A diferencia de la implementación del algoritmo en la versión imperativa, en la versión funcional se puede observar como se llega de una manera más *elegante* a la implementación del algoritmo. Pero en tanto en la versión imperativa y funcional algoritmo KMP se tiene la misma complejidad  $\Theta(n + m)$ .

En este análisis se desglosará la teoría necesaria para poder llevar a cabo este proceso en la programación funcional. También se hablará sobre una biblioteca en Haskell llamada QuickCheck para probar ambas versiones con cosas generados aleatoriamente.

Finalmente se hablará en qué consiste la programación competitiva y jueces en línea; en esta parte usando el algoritmo KMP y la función de error (función auxiliar usada en KMP) y se resolverán e implementarán algunos problemas famosos como encontrar el factor de repetición de una cadena, ver si una cadena es rotación cíclica de otra y como extender una cadena a un palíndromo de longitud mínima. Se implementarán tanto en Haskell como en C++ y se harán sus respectivas comparaciones.

## 2. Índice tentativo

1. **Fundamentos:** En esta sección se darán definiciones y teoremas necesarios usadas en la programación funcional para poder profundizar en el razonamiento ecuacional. También se darán propiedades y definiciones útiles respecto a *cadenas*, análisis de tiempo en algoritmos y un primer acercamiento a búsqueda de subcadenas.
  - 1.1. Programación funcional
  - 1.2. Listas, pliegos, principio universal de fusión, *Scan Lemma*, *Tuppling*, *Accumulating Parameters*
  - 1.3. Razonamiento ecuacional
  - 1.4. Notación asintótica
  - 1.5. Cadenas
2. **Algoritmos de búsqueda de subcadenas con perspectiva en la programación imperativa:** Se presentarán dos algoritmos; la versión ingenua y el algoritmo KMP y se analizarán algunos ejemplos y el algoritmo *per se*.
  - 2.1. Algoritmo de búsqueda de subcadenas ingenuo (naïve)
  - 2.2. Función de error
  - 2.3. De Morris-Pratt a Knuth-Morris-Pratt

## 2.4. Implementación en C++

### 3. Algoritmos de búsqueda de subcadenas con perspectiva en *la programación funcional*:

Aquí es donde se hará uso de todos los fundamentos presentados en el primer capítulo para hacer la *derivación* del algoritmo ingenuo al algoritmo Morris-Pratt y después a Knuth-Morris-Pratt. También se discutirá sobre la *función de error* que es usada en KMP. Se hará hincapié en los teoremas y *leyes* usadas en la programación funcional.

#### 3.1. Derivando Función de error

#### 3.2. Derivando Knuth-Morris-Pratt

4. **QuickCheck:** Se hablará acerca de esta biblioteca en Haskell para poder probar las propiedades que deberían de cumplir las funciones, es decir, cada función tiene propiedades deseables y usando QuickCheck se demuestra si se cumplen total o parcialmente estas propiedades. Una ventaja notoria es que una propiedad es probada con una gran cantidad de casos generados aleatoriamente.

#### 4.1. Motivación

#### 4.2. Uso básico

#### 4.3. Generadores

#### 4.4. Probando las implementaciones

5. **Jueces en línea:** Se hablará brevemente en qué consiste la programación competitiva y algunas “heurísticas” a seguir en la resolución de problemas. Al final se atacarán tres problemas en los lenguajes de programación Haskell y C++ usando el algoritmo KMP y la función de error.

#### 5.1. Diferentes jueces en línea

#### 5.2. Problemas: Encontrar el factor de repetición de una cadena, ver si una cadena es una rotación cíclica de otra, extender una cadena a un palíndromo de longitud mínima.

6. **Conclusiones:** Se centrará en las ventajas y desventajas del razonamiento ecuacional en la programación funcional, se hablará sobre algunas herramientas para poder verificar nuestros programas y se darán ideas a trabajos futuros.

7. **Apéndices\*:** Se abordarán cuestiones técnicas.

## 3. Bibliografía básica

- [1] Richard Bird y Jeremy Gibbons. Algorithm Design with Haskell. Cambridge University Press, 2020. doi: 10.1017/9781108869041.
- [2] Thomas H. Cormen y col. Introduction to Algorithms, Third Edition. 3rd. The MIT Press, 2009. isbn: 0262033844.
- [3] Graham Hutton. Programming in Haskell. 2nd. USA: Cambridge University Press, 2016. isbn: 1316626229.
- [4] Wojciech Rytter Maxime Crochemore. Jewels of stringology. 1st. World Scientific, 2002. isbn: 9789810247829,9789812778222,9810247826.
- [5] QuickCheck: An Automatic Testing Tool for Haskell. <http://www.cse.chalmers.se/~rjmh/QuickCheck/manual.html>.
- [6] Richard Bird. “On building cyclic and shared structures in Haskell”. En: Formal Aspects of Computing 24 (jul. de 2012).
- [7] Graham Hutton. “A Tutorial on the Universality and Expressiveness of Fold”. En: J. Funct. Program. 9.4 (jul. de 1999), págs. 355-372. <https://doi.org/10.1017/S0956796899003500>.