DEPARTAMENTO
INGENIERÍA INFORMÁTICA
Y CIENCIAS DE LA COMPUTACIÓN
FACULTAD DE INGENIERÍA UNIVERSIDAD DE CONCEPCIÓN

Estructuras de Datos y Algoritmos Avanzados (2023-2)

Proyecto 3: Detección de patrones en múltiples documentos

Profesor: José Fuentes Sepúlveda
Ayudante: Oliver Brito Alarcón

Objetivos

Los objetivos de este proyecto son:

- Mejorar la programación, compilación y ejecución de programas escritos en lenguaje *C++* u otros.
- Estudiar e implementar algoritmos y estructuras de datos para la búsqueda de patrones en textos.
- Analizar experimental y teóricamente el desempeño de los algoritmos y estructuras de datos implementadas.

1. Descripción del problema

En este proyecto continuaremos con los resultados obtenidos en el proyecto 2, pero esta vez enfocados en la búsqueda de patrones en varios documentos. Para aplicar lo que sabemos sobre arreglos de sufijos, BWT y FM-index, consideraremos un gran texto resultante de la concatenación de todos los documentos sobre los que realizaremos las búsquedas. Por ejemplo, si tenemos los siguientes documentos `D1`="`Búsqueda binaria corre en tiempo logarítmico`", `D2`="`El árbol binario de búsqueda es una estructura de datos fundamental`" y `D3`="`Sólo quedan 6 semanas para fin de año`", consideraremos su concatenación `T=D1$D2$D3`="`Búsqueda binaria corre en tiempo logarítmico$El árbol binario de búsqueda es una estructura de datos fundamental$Sólo quedan 6 semanas para fin de año`", donde el símbolo `$` se usa como separador. Por ejemplo, si buscamos el patrón `bin`, nos interesa como respuesta los

documentos `D1` y `D2`, mientras que si buscamos el patrón `queda`, esperamos la respuesta `D1`, `D2` y `D3`. En general, si buscamos un patrón que aparece *occ* veces en *ndoc* documentos diferentes, ya conocemos soluciones que retornan los documentos que contienen dicho patrón en tiempo proporcional a *ndoc* (ver clase 9). No obstante, en este proyecto final relajaremos las cotas y trabajaremos con soluciones con tiempo proporcional a *occ*. En otras palabras, construiremos un arreglo de sufijos o un FM-index sobre el texto concatenado `T`, para decidir el documento que contiene un patrón según su posición en `T`.

En este proyecto se implementarán dos soluciones para soportar búsqueda de patrones en los documentos:

1. Búsqueda basada en arreglo de sufijos: Al igual que en el proyecto 2, utilizaremos como base el arreglo de sufijos del texto concatenado, para luego implementar la búsqueda de patrones usando búsqueda binaria. En esta oportunidad, se entregará como base un código que computa el arreglo de sufijos usando un algoritmo lineal (ver Sección *Código base*).

2. FM-index: Implementación compacta del auto-índice FM-index, usando la biblioteca SDSL. Al igual que para el arreglo de sufijos, se entregará como base un código que construye el FM-index de un texto. (ver Sección *Código base*).

2. Objetivos específicos

La entrega del proyecto, que consiste de un informe y el código fuente, debe satisfacer los siguientes objetivos:

- Cada estructura de datos y algoritmo deben proveer de la funcionalidad *doc_locate(*`T, p`*)*, la cual retorna los documentos que contienen al menos una ocurrencia del patrón `p`. Los documentos serán representado por medio de su concatenación `T`

- Describir las características principales de los algoritmos y estructuras de datos utilizadas.

- Describir las decisiones de implementación más importantes.

- Plantear varias hipótesis sobre el rendimiento de las soluciones a comparar en escenarios específicos. Por ejemplo, "La solución X es más rápida buscando patrones que la solución Y cuando los datos cumplen cierta característica C" o "la solución X usa menos espacio que la solución Y". Se deben plantear (por lo menos) tantas hipótesis como integrantes tenga el equipo.

- Diseñar un experimento que permita verificar cada hipótesis.

- Ejecutar los experimentos y discutir los resultados obtenidos.

3. Condiciones

- El proyecto se realizará en grupos de dos o tres estudiantes. El informe debe reflejar claramente los autores del proyecto.
- Las soluciones a través de algoritmos y estructuras de datos pueden ser obtenidas de una fuente externa, la cual debe ser referenciada en el informe. Además, el funcionamiento de las implementaciones obtenidas externamente deben estar detalladamente comentadas.
- Todas las implementaciones deben realizarse en el mismo lenguaje de programación y utilizar los mismos tipos de optimizaciones.
- Las implementaciones se pueden realizar en los lenguajes *C++*, *Java* o *Python*.
- Todos los experimentos deben ser llevados a cabo en la misma máquina.
- En el informe se debe describir el entorno de *hardware* y *software* en que se ejecutan los experimentos. Además, debe quedar clara la manera en que se generan u obtienen tanto los datos de entrada como los resultados reportados. A modo de ejemplo, se entregan tres datasets obtenidos del corpus de *Pizza&Chili*[1]. Cada dataset está compuesto por 10 archivos de 5MB cada uno. Una persona externa con la información descrita en el informe y el código fuente, debiera ser capaz de replicar los experimentos y obtener las mismas conclusiones.
- Con el objetivo de reducir ruido en los resultados, las mediciones realizadas deben contemplar al menos 30 repeticiones, de las cuales se reportará la media y varianza.

4. Evaluación

El proyecto se realizará en grupos de dos o tres personas. Se debe subir a Canvas lo siguiente:

1. Un informe que:
   a) Incluya portada, descripción de la tarea, descripción de las soluciones propuestas, detalles de implementación, análisis teórico y análisis experimental, considerando las condiciones previamente estipuladas.
   b) Sea claro y esté bien escrito. Un informe difícil de entender será mal evaluado, aunque todo esté bien implementado. Quien revise el documento debe poder entender su solución solo mirando el informe.
   c) Esté en formato PDF.

[1] `http://pizzachili.dcc.uchile.cl/`

2. Un archivo comprimido con todos los ficheros fuente implementados para solucionar la tarea. El informe debe hacer referencia a ellos y explicar en qué consiste cada uno.

Fecha de entrega: lunes 11 de diciembre de 2023 11:59PM

5. Código base

Este proyecto viene acompañado de dos archivos con código base:

- *sa.cpp*: Archivo que contiene un ejemplo de cómputo del arreglo de sufijos de un texto, así como su BWT. El ejemplo se basa en una implementación disponible en la biblioteca SDSL y trabaja en tiempo lineal. El objetivo de este código es que se use como base como implementar la búsqueda basada en arreglo de sufijos. Si su implementación actual (la usada en el proyecto 2) ya es lo suficientemente rápida y/o usa poco espacio durante el cómputo, puede ignorar este documento.
- *FM-index.cpp*: Archivo que contiene un ejemplo de un FM-index usando estructuras de datos compactas, además de un ejemplo para las operaciones *count()* y *locate()*. La implementación del FM-index se base en una estructura de datos compacta llamada *compressed suffix array* (CSA), el cual utiliza como base un wavelet tree. En la línea 27 del documento se aprecia como en el ejemplo se indica que use como base la implementación del wavelet tree llamada `wt_int<>`. Para el proyecto, se pide cambiar esa opción por alguna otra, como `wt_int<rrr_vector<≫`, `wt_huff<>` u otra. [2]

[2]Para más opciones, revisar la sección Wavelet Trees de `https://simongog.github.io/assets/data/sdsl-cheatsheet.pdf`