Proyecto 2: Compresión Re-Pair

Estructuras de Datos, Primer Semestre 2020

Prof. Diego Seco

Ayudantes: Catalina Pezo y Alexis Espinoza

1. Descripción del Problema

La compresión de datos consiste en la reducción del espacio empleado para representar una determinada información. Existen muchas técnicas para lograr dicho objetivo. En el módulo 12 se ven las más básicas y en la asignatura Estructuras de Datos y Algoritmos Avanzados, se ven otras complementarias y más avanzadas. Generalmente un compresor combina varias técnicas para lograr mayor efectividad. Estas técnicas además se suelen clasificar en familias. En este proyecto se verá una técnica importante en la familia de compresión basada en gramáticas. La técnica se conoce como Re-Pair (*Recursive Pairing*).

La idea básica de Re-Pair consiste en, dada una secuencia de símbolos de entrada (por ejemplo, un texto) construir una gramática que genere dicha secuencia. Para ello, recursivamente reemplaza el par de símbolos más frecuente, digamos xy por un nuevo símbolo Z. En ese momento se crea una regla de la forma $Z \rightarrow xy$. El proceso termina cuando ningún par de símbolos se repite 2 veces. La secuencia resultante de ese proceso se conoce como axioma de la gramática. Tomando como ejemplo la secuencia $w =$ "$xabcabcy123123zabc$", el par de símbolos más frecuente es ab, por lo que en la primera iteración se crea la regla $R_1 \rightarrow ab$ y la secuencia queda como $w_1 =$ "$xR_1cR_1cy123123zR_1c$". El par más frecuente en w_1 es R_1c, por lo que se crea una regla $R_2 \rightarrow R_1c$ y la secuencia queda como $w_2 =$ "$xR_2R_2y123123zR_2$". Se puede ver el resto del ejemplo en `https://en.wikipedia.org/wiki/Re-Pair`.

Si bien la idea es sencilla, obtener una implementación eficiente de la misma no es trivial. Esto incluso sin considerar el cómo se almacenan las reglas para que ocupen poco espacio (lo cual dejaremos fuera de este proyecto). En este proyecto se comparará una versión directa con otra más avanzada (sin llegar todavía a la solución óptima).

2. Implementación

Para ambas versiones, la secuencia a comprimir será almacenada como una lista doblemente enlazada cuya implementación forma parte del proyecto. Además, ambas versiones pueden hacer uso de la clase map de la librería estándar (como caja negra). Por simplicidad, las secuencias serán de números enteros entre 1 y un parámetro σ. Los símbolos que se irán creando también serán enteros secuenciales, comenzando en $\sigma + 1$.

La versión directa hará iteraciones sobre la secuencia mientras que en la misma haya algún par con frecuencia mayor a 1. Cada iteración realizará varias pasadas sobre la secuencia. En la primera pasada, para contar la frecuencia de los pares se utilizará map (pista: `https://www.geeksforgeeks.org/map-pairs-stl/`). Una vez detectado el par de mayor frecuencia, se realizará una segunda pasada reemplazando dicho par por un símbolo nuevo y se pasará a la siguiente iteración. Notar que en cada iteración el map se debe vaciar.

En la versión avanzada, cada nodo de la lista doblemente enlazada contendrá dos punteros adicionales, uno a la ocurrencia previa del mismo par y otro a la ocurrencia siguiente a dicho par. El map, además de la frecuencia del par, almacenará un puntero a la primera ocurrencia del par y un puntero a la última ocurrencia del par. Por último, se implementará un MAX-heap cuya clave será la frecuencia y, por cada nodo, el valor será el par correspondiente (notar que estos nodos son conceptuales, ya que MAX-heap se puede implementar sobre un arreglo o vector). El heap implementado debe permitir una operación adicional a las habituales la cual, dado un nodo, permitirá modificar su clave (notar que esto puede subir o bajar al nodo en el heap). También es posible que un clave llegue a valer 0, pero no es necesario eliminar el nodo correspondiente. El algoritmo funcionará del siguiente modo: en una primera pasada se inicializará el map, el heap y los punteros de la lista doblemente enlazada. Después, iterativamente se solicitará al heap el par de mayor frecuencia. Si la frecuencia es menor que 2, se termina el algoritmo. Mientras que la frecuencia obtenida sea 2 o más, se creará un nuevo símbolo (incremental sobre el anterior) que reemplazará a dicho par en la secuencia. Por cada substitución en la secuencia se puede tener que: crear un nuevo par (con el nuevo símbolo y quien lo precede/sucede), modificar la frecuencia de un par y/o eliminar un par. Además, se deben mantener los punteros en la lista doblemente enlazada. Estos detalles de implementación pueden complicar el proyecto, por lo que se recomienda realizar un ejemplo completo en papel en el que se analicen todos los casos.

3. Evaluación

El proyecto se realizará en parejas. Subir en Canvas lo siguiente:

1. Un informe en pdf que:
   a) Incluya portada, descripción de la tarea, descripción de la solución propuesta (puede emplear diagramas de clase para explicar la solución) y detalles de implementación (debe incluir un ejemplo completo con al menos 3 pasos donde se vea cómo se van actualizando los punteros y estructuras de datos, para la versión avanzada). Además, debe incluir una evaluación preliminar de tiempo que compare ambos algoritmos. Para eso se generarán secuencias aleatorias de tamaño n (para varios valores de n creciente). El tamaño de $\sigma = 27$ será fijo. Se incluirá un gráfico donde el eje X se corresponderá con n y el eje Y con tiempo consumido por cada algoritmo.
   b) Sea claro y esté bien escrito. Un informe difícil de entender es un informe que será mal evaluado aunque todo esté bien implementado. La persona que revise el documento debe poder entender su solución sólo mirando el informe.
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: viernes 10 de Julio 11:59PM