Helena Vegalara Correa 201823328 s.arangoa

Sergio Arango Arango 201921814 h.vegalara

Observaciones del Reto 2

Análisis de Resultados:

- Rendimiento (tiempo de ejecución y consumo de memoria)

Al analizar el rendimiento del Reto 1 y del Reto 2 con los laboratorios realizados a lo largo del curso, se pudo determinar primero que, para la utilización de listas (Reto 1), era más eficiente en términos de tiempo de ejecución la estructura ARRAY_LIST. Posteriormente, con los laboratorios se pudo determinar que la estructura de datos para la utilización de mapas (Reto 2) que resultó ser más eficiente en términos de tiempo de ejecución fue CHAINING. Con esta información, se procedió a comparar la eficiencia en tiempo del Reto 1 con el Reto 2.

A continuación, podemos ver el tiempo de ejecución y el consumo de memoria de cada requerimiento, para cada reto:

Reto 1:

Tiempo de ejecución[mS]:

Carga	Requerimiento 1	Requerimiento 2	Requerimiento 3	Requerimiento 4
67564.12	**1166.97**	**4575.95**	N.A	**1473.56**

Consumo de memoria[kB]:

Carga	Requerimiento 1	Requerimiento 2	Requerimiento 3	Requerimiento 4
1034501.37	**1018.25**	**15479.45**	N.A	**2204.26**

Reto 2:

Tiempo de ejecución[mS]:

Carga	Requerimiento 1	Requerimiento 2	Requerimiento 3	Requerimiento 4
107483.76	**434.43**	**4337.79**	**3539.74**	**591.29**

Consumo de memoria[kB]:

Carga	Requerimiento 1	Requerimiento 2	Requerimiento 3	Requerimiento 4
1026527.65	**1009.31**	**10.02**	**10096.91**	**2194.05**

Estas pruebas son realizadas con la estructura ARRAY en el reto 1 y con CHAINING con factor de carga de 4.0 en el reto 2.

Helena Vegalara Correa 201823328 s.arangoa

Sergio Arango Arango 201921814 h.vegalara

Se observa que el reto 2 es más rápido en todos los requerimientos y bastante más rápido en algunos (por ejemplo, Requerimiento 1 y Requerimiento 4). El consumo de memoria también resulta menor para cada requerimiento. Esto se debe a que el acceso a la información es extremadamente más eficiente en los mapas, al ser de complejidad constante, que en las listas donde se requieren recorridos lineales para crear sublistas por criterios arbitrarios. Es decir, ambos programas resuelven los requerimientos de la misma forma en general, que resulta ser: acceder a la información deseada y luego ordenar por vistas o likes en el caso de Requerimiento 1 y Requerimiento 4 ó contar repeticiones de forma inteligente en Requerimiento 2 y Requerimiento 3. Dado lo anterior, las complejidades de los requerimientos corresponden a complejidades parecidas a las del reto anterior remplazando recorridos lineales iniciales (con los que se creaban las sublistas) por accesos directos a la memoria (complejidad constante):

Sea N la cantidad total de datos(videos) cargados.

Requerimiento 1: O(1) + O(1) + O(JLogJ) + O(J) + O(RLogR); N>J>R

Primero se hace el primer acceso al mapa compuesto por país, que retorna el mapa de las categorías de ese país como llaves y cada categoría proporciona una lista de los videos del país y de la categoría. El segundo acceso es en este mapa interno y retorna los J videos del país y categoría. En la función ObtenerVideos distintos primero se hace un ordenamiento alfabético por video_id (asumiremos que se hace con MergeSort) y tiene complejidad JLogJ y esta función vuelve a hacer una sublista colapsando los videos por video_id lo cual tiene complejidad J y retorna R<J datos. Finalmente, como respuesta se vuelven a ordenar por views, pero esta vez los R datos finales, se vuelve a asumir que el algoritmo es MergeSort por lo que tiene complejidad RLogR.

Requerimiento 2: O(1) + O(JLogJ) + O(J) + O(R) ; N>J>R.

Primero se obtienen los videos del país accediendo al mapa lo cual tiene complejidad constante, aquí J<N es la cantidad de videos del país. En la función ObtenerVideos distintos primero se hace un ordenamiento alfabético por video_id (asumiremos que se hace con MergeSort) y tiene complejidad JLogJ y esta función vuelve a hacer una sublista colapsando los videos por video_id lo cual tiene complejidad J y retorna R<J datos. Luego se hace un recorrido lineal sobre los J videos distintos del país para ver cuál tiene el máximo valor en el atributo ‘repeticiones’.

Requerimiento 3: O(1) + O(JLogJ) + O(J) + O(R) ; N>J>R.

Primero se obtienen los videos de la categoría accediendo al mapa lo cual tiene complejidad constante, aquí J<N es la cantidad de videos de la categoría. En la función ObtenerVideos distintos primero se hace un ordenamiento alfabético por video_id (asumiremos que se hace con MergeSort) y tiene complejidad JLogJ y esta función vuelve a hacer una sublista colapsando los videos por video_id lo cual tiene complejidad J y retorna R<J datos. Luego se hace un recorrido lineal sobre los J videos distintos de la categoría para ver cuál tiene el máximo valor en el atributo ‘repeticiones’.

Requerimiento 4: O(1) + O(JLogJ) + O(J) + O(RLogR); N>J>R.

Helena Vegalara Correa 201823328 s.arangoa

Sergio Arango Arango 201921814 h.vegalara

Primero se hace el único acceso de esta solución al mapa por país, que retorna la lista de los videos del país de tamaño J. En la función ObtenerVideos distintos primero se hace un ordenamiento alfabético por video_id (asumiremos que se hace con MergeSort) y tiene complejidad JLogJ y esta función vuelve a hacer una sublista colapsando los videos por video_id lo cual tiene complejidad J y retorna R<J datos. Finalmente, como respuesta se vuelven a ordenar por views, pero esta vez los R datos finales, se vuelve a asumir que el algoritmo es MergeSort por lo que tiene complejidad RLogR.

Se concluye que la implementación de este mismo programa es más eficiente con las nuevas estructuras de datos del nuevo TAD mapa ya que hace que el acceso a cierta información sea directo si se construyen los mapas desde la carga de forma que no se necesiten recorridos lineales para obtener subconjuntos de los datos totales.