Taller 4: Análisis de eficiencia

Fernando González Rivero
Isabella Arrubla Orozco

23 de octubre de 2024

1. Análisis de `clasificarPeliculas()`

```
public static ST<String, BST<Integer, List<Pelicula>>>
clasificarPeliculas(Pelicula[] peliculas) {
    // Crear la tabla de simbolos principal donde la
    // llave es el genero
    ST<String, BST<Integer, List<Pelicula>>>
    clasificacion = new ST<>();

    for (Pelicula peli : peliculas) { // Ciclo: N
        // Obtener la lista de generos de la pelicula
        List<String> generos =
                parseGeneros(peli.getGenre_list());

        for (String genero: generos) { // Ciclo: g
            // Obtener la tabla de anios (tabla
            // secundaria) para el genero actual
            // (get() a un ST)
            BST<Integer, List<Pelicula>> stAnio
                    = clasificacion.get(genero);
            if (stAnio == null) {
                stAnio = new BST<>();
                // put() a un ST
                clasificacion.put(genero, stAnio);
            }

            // Obtener la lista de peliculas para el
            // anio actual (get() a un BST)
            List<Pelicula> listaPelis
                    = stAnio.get(peli.getYear());
            if (listaPelis == null) {
                listaPelis = new ArrayList<>();
                // put() a un BST
```

```
                    stAnio.put(
                            peli.getYear(), listaPelis
                        );
                }

                // Agregar la pelicula a la lista
                // correspondiente
                listaPelis.add(peli);
            }
        }

        return clasificacion;
    }
```

Las operaciones relevantes están dentro de dos ciclos: el más externo itera sobre el arreglo de películas, por lo que todo lo que esté dentro de este se repetirá N veces. Por su parte, el más interno itera sobre la cantidad de géneros que tenga una película en específico. Si bien esta cantidad es distinto para cada película, sabemos que no va a crecer hasta el infinito, cuando mucho, será la cantidad de géneros en el dataset (para nuestro caso, 19 [1]), si por alguna extraña razón una película pertenece a todos los géneros. En cualquier caso, utilizaremos una cota superior constante g para representar esta cantidad.

Dentro de estos dos ciclos, se realizan operaciones put() y get() a dos tipos de tablas de símbolos. Una de ellas es un ST implementado con arreglos, `clasificacion`, la cual relaciona a un género con un otra tabla de símbolos, `stAnio`, que las ordena por año, el segundo tipo que nos interesa, implementado como un árbol binario de búsqueda donde las llaves son los años y los valores corresponden a la lista de películas de ese género lanzadas ese año.

Tabla 1: Eficiencia de put() y get() para las dos tablas de símbolos relevantes.

Operación	`clasificacion`	`stAnio`
`get()`	$\sim lg(g)$	$\sim ln(y)$
`put()`	$\sim g$	$\sim ln(y)$

La tabla 1 lista la eficiencia media de estas operaciones, donde:

- g es el número total de géneros del dataset, previamente discutida.
- y es el número de años considerados por nuestro dataset.

Sin embargo, dado que el número de llaves de `clasificacion` está acotado por la constante g (número de géneros del dataset), despreciaremos todas sus operaciones. y, en cambio, tenderá a crecer junto al número de películas (lógicamente, a medida en que pasan los años, más películas se van lanzando). El

[1] Estrictamente hablando, será la máxima cantidad de géneros asociados a una película, que, para nuestro dataset, sería 7, de la película *Sherlock Gnomes (2018)*

algoritmo siempre hace un `get()` a `stAnio`, y, si no se encuentra el año en cuestión dentro del árbol, se le hace un `put()`. En el peor caso, por tanto, estos dos procedimientos son, en conjunto, tilde de $\sim 2ln(y)$.

Juntando todo esto, llegamos a la expresión:

$T(N, y) \sim 2Ngln(y)$

Que tiene un orden de crecimiento $O(Nln(y))$, linearitmético en base a las variables N y y.

2. Análisis de `topGeneroAnios()`

```
public static List<Pelicula> topGeneroAnios(
        ST<String, BST<Integer, List<Pelicula>>>
                pelisXgeneroXanio,
        String genero,
        int anoInicio,
        int M ) {

        // Obtener la tabla de anios para el genero
        // especificado
        BST<Integer, List<Pelicula>> stAnio
                = pelisXgeneroXanio.get(genero);
        if (stAnio == null) {
            // No hay peliculas para este genero
            return null;
        }

        Comparator<Pelicula> cmp
                = new PeliculaVoteComparator();
        MinPQ<Pelicula> pelipq = new MinPQ(M, cmp);

        // Obtener los anios dentro del rango especificado
        for (Integer ano : stAnio.keys(anoInicio, anoFin))
            for (Pelicula peli: stAnio.get(ano))
                if ( pelipq.size() < M )
                    pelipq.insert(peli);
                else if (cmp.compare(peli, pelipq.min()) > 0){
                    pelipq.delMin();
                    pelipq.insert(peli);
                }

        if (pelipq.isEmpty()) {
            // No hay peliculas en el rango especificado
            return null;
        }
```

```
        Pelicula[] peliarr = new Pelicula[M];
        for (Pelicula peli: pelipq)
            peliarr[--M] = peli;
        return Arrays.asList(peliarr);
    }
```

Este algoritmo sigue el procedimiento de obtener el top-M de un dataset utilizando una cola de mínima prioridad visto en clase. El comparador de películas utilizado, `cmp`, compara en base a la votación de películas, representada por un `Double`, lo cual la hace una operación $O(1)$.

Las variables que consideraremos para este análisis son:

- N El número de películas pertenecientes al género seleccionado.
- M La cantidad de películas mostradas en el top.

Además, consideraremos el peor caso que todos los años del dataset fueron seleccionados. Esto es, todas las N películas son candidatas para el top-M, en vez de únicamente las pertenecientes a un rango determinado de años.

La parte principal del algoritmo está contenida dentro de los ciclos anidados:

```
for (Integer ano : stAnio.keys(anoInicio, anoFin))
    for (Pelicula peli: stAnio.get(ano))
```

El externo itera sobre los años seleccionados, mientras que el interno itera sobre las películas del género seleccionado en ese año específico. En conjunto, esto hace que el cuerpo de los dos ciclos anidados se repita para cada una de las N películas seleccionadas.

Dentro de este, la operación más relevante es la eliminación del mínimo elemento en la cola (de eficiencia $\sim 2lg(M)$) y la inserción de un nuevo elemento dentro de ella (de eficiencia $\sim lg(M)$), para un total de $\sim 3lg(M)$ comparaciones.

El resto de operaciones en el ciclo son aritmético-lógicas o accesos a memoria, ambas de tiempo constante.

Por tanto, el número de comparaciones hecho dentro de estos dos ciclos es tilde de $\sim 3Nlg(M)$, de orden $O(Nlg(M))$ (linearitmético en base a las variables N y M) de acuerdo a lo visto en clase.

El último conjunto de operaciones:

```
Pelicula[] peliarr = new Pelicula[M];
for (Pelicula peli: pelipq)
    peliarr[--M] = peli;
return Arrays.asList(peliarr);
```

Son $O(M)$, un orden de crecimiento despreciable respecto al anterior, en primer lugar por ser únicamente lineal, en segundo lugar, y más importante, por ser en base a la variable M, la cual se espera que sea mucho menor que N.

Orden de crecimiento de `topGeneroAnios()`: $O(Nlg(M))$