Análisis de resultados reto 3

Alejandra Melo -> req. 2

a.melo4@uniandes.edu.co

202021526

Obed Cobanzo

jo.cobanzo@uniandes.edu.co

201911749

*las complejidades teóricas de tiempo y memoria se obtuvieron con el small

Carga de datos

Complejidad de tiempo: 16872.981 ms

Complejidad de memoria: 160793.050 kb

-La complejidad espacial de los árboles balanceados se define de acuerdo con su altura, la cual corresponde a O(logN)

Requerimiento 1

```
def getReproducciones(analyzer, contenido, min, max):
    """
    Retorna el total de reproducciones (eventos de escucha) que se tienen en el sistema de
    recomendación basado en una característica de contenido y con un rango determinado
    """
    num_eventos = 0
    map_art_cumple = om.newMap(omaptype='RBT',comparefunction=compareArtistas)

    for evento in lt.iterator(analyzer["eventos"]):
        if float(evento[contenido]) <= max and float(evento[contenido]) >= min :
            num_eventos += 1
            om.put(map_art_cumple, evento['artist_id'],"")

    num_artistas = om.size(map_art_cumple)

    if num_eventos == 0:
        return None

    return (num_eventos, num_artistas)
```

Complejidad temporal: O(N)

La complejidad en el peor caso es de N porque se haría comparación de todos los datos de eventos.

Complejidad de tiempo: 958.924 ms

Complejidad de memoria: 5.868 kb

Requerimiento 2

```
def getMusicapara(analyzer, c1, c2, min_c1, max_c1, min_c2, max_c2):
    """
    Retorna las pistas en el sistema de recomendación que cumplen dos características
    """
    map_pista_cumple = om.newMap(omaptype='RBT',comparefunction=compareIds)

    for pista in lt.iterator(analyzer["eventos"]):
        if float(pista[c1]) <= max_c1 and float(pista[c1]) >= min_c1:
            if float(pista[c2]) <= max_c2 and float(pista[c2]) >= min_c2:
                om.put(map_pista_cumple, pista["track_id"], (pista[c1], pista[c2]))

    num_unicas = om.size(map_pista_cumple)

    if num_unicas == 0:
        return None

    return (num_unicas, map_pista_cumple)
```

Complejidad temporal: O(N)

La complejidad en el peor caso es de N porque se haría comparación de todos los datos de eventos.

Complejidad de tiempo: 1037.870 ms

Complejidad de memoria: 681.20 kb

Requerimiento 3

```
def getMusicapara(analyzer, c1, c2, min_c1, max_c1, min_c2, max_c2):
    """
    Retorna las pistas en el sistema de recomendación que cumplen dos características
    """
    map_pista_cumple = om.newMap(omaptype='RBT',comparefunction=compareIds)

    for pista in lt.iterator(analyzer["eventos"]):
        if float(pista[c1]) <= max_c1 and float(pista[c1]) >= min_c1:
            if float(pista[c2]) <= max_c2 and float(pista[c2]) >= min_c2:
                om.put(map_pista_cumple, pista["track_id"], (pista[c1], pista[c2]))

    num_unicas = om.size(map_pista_cumple)

    if num_unicas == 0:
        return None

    return (num_unicas, map_pista_cumple)
```

Complejidad temporal: O(N)

La complejidad en el peor caso es de N porque se haría comparación de todos los datos de eventos.

Complejidad de tiempo: 413.395 ms

Complejidad de memoria: 0.172 kb

Requerimiento 4

Requerimiento 5