Miniproyecto FADA - Universidad del Valle

David Santiago Cortés, Alejandro Orozco, Brayan Rincones

1 Soluciones Planteadas

Idea General de la solución:
Almacenar los animales y sus grandezas en una estructura de datos tipo lista, arreglo o diccionario, se ordena esta estructura de acuerdo a los valores de las grandezas y se guarda utiliza para armar las escenas de todas las partes del evento.

1.1 $O(n^2)$

Idea de la solución:
Estructuras de datos utilizadas: Listas, Diccionarios
Algoritmo de ordenamiento: Bucles
Lenguaje en el que se implementó: Python

1.2 $O(n * \log(n))$

Idea de la solución: En la descripción del problema tenemos una serie de elementos que, de alguna manera, hay que ordenarlos para dar con el espectáculo final. Al tener que organizar ascendentemente los escenarios de acuerdo a las grandezas de cada animal, además de organizar las partes acorde a las grandezas de las escenas que conlleva, se considera que este problema se puede llevar a cabo mediante el ordenamiento de Merge Sort, cuando ya tengamos definido las características de los animales, estos serán almacenados en arreglos, tanto nombres en un arreglo, como grandezas respectivas en otro arreglo, los cuales serán los datos de entrada.
Ya organizados los animales en un escenario por una función auxiliar, esta misma es llamada dentro del Merge Sort, con esto estamos organizando las partes finales acorde a operaciones como tamaños de medias de grandezas de los escenarios. Extrayendo los datos, podemos obtener la media total de grandeza, el animal que mas participo, escena de mayor grandeza, menor grandeza así como el que menos lo hizo mediante contadores, abstracción y almacenamiento en nuevos arreglos de los resultados finales, cada operación con su respectiva función dentro del algoritmo.

Estructuras de datos utilizadas:
Algoritmo de ordenamiento: Merge Sort
Lenguaje en el que se implementó: Python

1.3 $O(n)$

Idea de la solución: Después de recolectar los datos de entrada se construye un diccionario que mapea animales a sus respectivas grandezas a partir de la lista de animales y grandezas que se pasaron como parametro de entrada. Este diccionario se usará más adelante para apoyar distintas operaciones.
Para atacar el problema se utiliza una estrategia *bottom-up*

1. Ordenar las escenas localmente: para que todas las escenas del evento queden ordenadas localmente solo basta con ordenar las escenas de la apertura. El ordenamiento se hace con una implementación

de Counting Sort que ordena una lista de tuplas (animal,grandeza) en orden ascendente de acuerdo a las grandezas.

2. Ordenar las escenas de las demás partes (localmente): con el resultado del paso anterior se construye un diccionario/tabla hash que relaciona escenas desordenadas con escenas ordenadas, se usa un ciclo que itera sobre cada escena de las m-1 partes y se reemplaza cada escena en cada parte haciendo la consulta en la tabla hash.

3. Ordenar todas las escenas: teniendo todas las escenas ordenadas localmente en cada parte, lo que resta es ordenar las escenas dentro de sus partes de acuerdo a sus grandezas. El proceso se divide en dos, la primera parte se encarga de constuir un diccionario/tabla hash que relacione escenas con su grandeza, para ello se itera sobre apertura y se calcula la grandeza de cada escena, con esta información ordenamos apertura utilizando Counting Sort.
   En la segunda parte se itera sobre las siguientes $m-1$ partes y sobre las k escenas de tales partes, se consulta cada escena en el diccionario para hallar su grandeza y antes de cambiar a la siguiente parte, se ordenan las escenas de la parte actual con Counting Sort. Al finalizar el ciclo externo las escenas dentro de cada parte están ordenadas ascendentemente.

   (a) Empates: se buscan y se resuelven cada que se ordenan las escenas de una parte. Para detectarlos se usa una función que detecta duplicados en una lista utilizando la librería `collections`. Una vez se saben los indices de los elementos repetidos, se utiliza la función `remove_duplicates` que remueve los duplicados con un `while` que va "en reversa" empezando en $i = 2$ hasta 0, i se usa para acceder una tabla hash que mapea animales a grandezas.

4. Ordenar las partes: para ordenar las partes es necesario saber las grandezas de cada parte, lo cual se hace en el paso anterior cada que se va iterando sobre cada parte y se van almacenando los valores una en una lista. `sort_parts` entonces fusiona las $m-1$ partes con sus grandezas, generando una lista de tuplas (parte, grandeza) que se le pasa como parametro a la función de ordenar.

5. Estadísticas del evento:

   (a) Animal más popular y menos popular: se utiliza una lista de todas las escenas que ocurren en el evento (con apertura y las demás partes) y la lista de animales del evento. Con ambas se construye una tabla hash que mapea animales a número de veces que aparecieron en escena, como candidato para animal más popular y menos popular se tiene obviamente al elemento máximo y mínimo de los valores de la tabla hash, para no dejar por fuera ningún otro animal se itera sobre la tabla hash buscando animales distintos al candidato y cuyas apariciones sean iguales a las del candidato.

   (b) Promedio de grandeza del evento: se itera sobre la lista de todas las escenas que tuvieron lugar, se suman sus grandezas y se divide entre el número de escenas que se mostraron. Lo mismo también puede lograrse con las partes y sus grandezas.

   (c) Escenas de mayor y menor grandeza: se calcula el elemento más grande y más pequeño en la tabla hash que mapea escenas a sus grandezas.

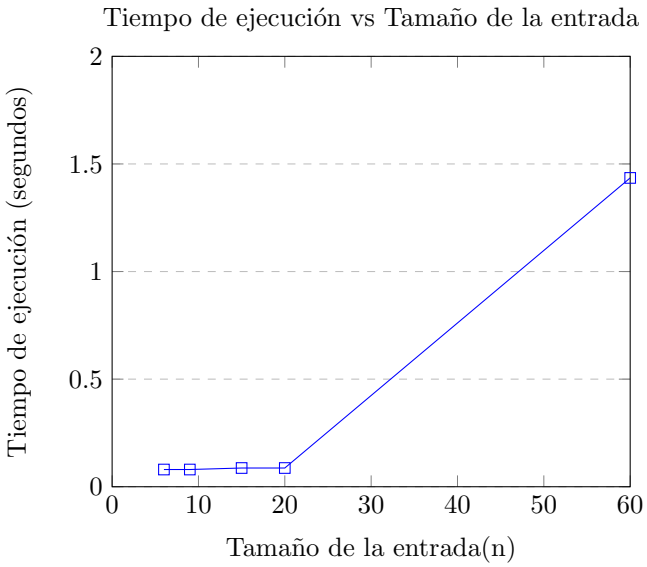
Estructuras de datos utilizadas: Listas, tablas hash, tuplas
Algoritmo de ordenamiento: Counting Sort para listas de tuplas (animal, grandeza)
Lenguaje en el que se implementó: Python

2 Análisis de Resultados

2.1 $O(n^2)$

2.2 $O(n * \log(n))$

2.3 $O(n)$



3 Instrucciones para la ejecución

3.1 $O(n^2)$

En la linea de comandos o al final del programa escribir zoo("nombre del archivo a ejectura")

3.2 $O(n * \log(n))$

Ya por defecto existe una entrada en la carpeta de archivos, `entrada.txt`, con datos muy grandes para la prueba del programa solZoonlogn.py, puesto que no se notan cambios muy bruscos con respecto al tiempo de ejecución final al probar con datos de entrada pequeños. Por ende, como ya hay un archivo de entrada, se procede a ejecutar el programa (run) `solZoonlogn.py` desde una ide de python3 o por medio de terminal pasandole como argumento el nombre y la ruta al archivo de entrada y este retorna los resultados por consola.

Si desea cambiar y probar con otra entrada, el archivo `file.py` es un programa que modifica a entrada.txt, con los valores para n,m,k, animales y grandezas que usted elija, inclusive en este documento encontrará tests con los cuales hemos probado, con sus respectivas medidas de tiempo de ejecución. Si quiere modificar la entrada manualmente, aquí hay algunos ejemplos de pruebas realizadas con anterioridad, solo tiene que copiar alguno de estos ejemplos en el archivo de entrada.txt y así reemplazar el contenido que esté ya tenía.

3.3 $O(n)$

`./run.py <nombre-archivo-entrada>`

3.4 Instrucciones para generar entradas

- Para generar entradas de la solución cuadrática y lineal, ejecutar el archivo `./generar.py <valor-n> <valor-m> <valor-k>`

- Para generar entradas de la solución $n\log(n)$ ejecutar el archivo `file.py` dentro de la carpeta `nlgn`

4 Sets de prueba

Se encuentran dentro de la carpeta `/entradas`

5 Conclusiones del proyecto