**WORKGROUP**

* **Juan Felipe Jojoa Crespo A00382042**
* **Felipe Rojas Prado A00393918**

**COMPLEXITY AND SPATIAL COMPLEXITY ANALYSIS**

COMPLEXITY ANALYSIS

| 1. **We choose the function of receiving passenger information that has complexity O(1) and we can identify it as follows:**   **Function uploadPassengers:** | | |
| --- | --- | --- |
| File file = new File(archivo); - O(1) (crea un objeto de tipo File a partir del nombre de archivo proporcionado)  hashTable = new Plane<>(num); - O(1) (crea una instancia de la clase Plane con un tamaño determinado)  FileInputStream fis = new FileInputStream(file); - O(1) (crea un objeto FileInputStream a partir del archivo proporcionado)  BufferedReader reader = new BufferedReader(new InputStreamReader(fis)); - O(1) (crea un objeto BufferedReader a partir del objeto FileInputStream)  String line; - O(1) (crea una variable String para almacenar cada línea leída del archivo)  while ((line = reader.readLine()) != null) { - O(n) (lee cada línea del archivo hasta que se alcanza el final del archivo)  String[] atributs = line.split(";"); - O(n) (divide la línea actual en una matriz de cadenas usando ; como delimitador)  Passenger passenger = new Passenger(atributs[0], Integer.parseInt(atributs[1]), Integer.parseInt(atributs[2]),Integer.parseInt(atributs[3]), Integer.parseInt(atributs[4]),atributs[5],Boolean.parseBoolean(atributs[6])); - O(1) (crea un objeto Passenger a partir de los atributos en la línea actual)  hashTable.insert(Integer.parseInt(atributs[1]),passenger); - O(1) (inserta el objeto Passenger en la tabla hash en función de su atributo id)  fis.close(); - O(1) (cierra el objeto FileInputStream)  } catch (FileNotFoundException error) { - O(1) (maneja la excepción FileNotFoundException)  error.printStackTrace(); - O(1) (imprime el rastro de pila de la excepción)  } catch (IOException error) { - O(1) (maneja la excepción IOException)  error.printStackTrace(); - O(1) (imprime el rastro de pila de la excepción) | | |
|
|

| 1. **We choose the function of organizing the order of arrival of the passengers that has complexity O(n) and we can identify it as follows:**   **Function orderPassenger:** | | |
| --- | --- | --- |
| try (Stream<String> lines = Files.lines(Paths.get(filePath))) { - O(1) (crea un flujo de líneas a partir del archivo proporcionado)  lines.forEach(line -> { - O(n) (itera a través de cada línea del archivo)  String[] atributs = line.split(" "); - O(n) (divide la línea actual en una matriz de cadenas usando " " como delimitador)  Passenger passenger = hashTable.get(Integer.parseInt(atributs[0])); - O(1) (recupera el objeto Passenger de la tabla hash en función de su atributo id)  if (passenger != null) { - O(1) (verifica si el objeto Passenger existe en la tabla hash)  passenger.toString(); - O(1) (convierte el objeto Passenger a una cadena)  passenger.setTime(LocalTime.now()); - O(1) (establece la hora actual como la hora de llegada del pasajero)  if (passenger.getTime().isBefore(getTimeOfPlane())) { - O(1) (verifica si la hora de llegada del pasajero es anterior a la hora de salida del avión)  passenger.setMiles(passenger.getMiles() + 100); - O(1) (agrega 100 millas al pasajero)  if (passenger.getFirstClass()) { - O(1) (verifica si el pasajero está en primera clase)  firstClass.insert(passenger); - O(1) (inserta el pasajero en la cola de primera clase)  } else { - O(1)  turistClass.enqueue(passenger); - O(1) (inserta el pasajero en la cola de clase turista)  } else { - O(1) (maneja el caso en el que no se encuentra el objeto Passenger en la tabla hash)  System.out.println("No se encontro el pasajero, intenta nuevamente (Revisa que la direccion que estas poniendo este bien escrita y que en cada salto de arhcivo este un \\)"); - O(1) (imprime un mensaje de error)}); - O(1) | | |
|
|

**SPATIAL COMPLEXITY ANALYSIS**

SPATIAL COMPLEXITY ANALYSIS

| 1. **Function uploadPassengers** | | |
| --- | --- | --- |
| File file = new File(archivo); - O(1) (crea un objeto de tipo File a partir del nombre de archivo proporcionado)  hashTable = new Plane<>(num); - O(1) (crea una instancia de la clase Plane con un tamaño determinado)  FileInputStream fis = new FileInputStream(file); - O(1) (crea un objeto FileInputStream a partir del archivo proporcionado)  BufferedReader reader = new BufferedReader(new InputStreamReader(fis)); - O(1) (crea un objeto BufferedReader a partir del objeto FileInputStream)  String line; - O(1) (crea una variable String para almacenar cada línea leída del archivo)  while ((line = reader.readLine()) != null) { - O(n) (lee cada línea del archivo hasta que se alcanza el final del archivo)  String[] atributs = line.split(";"); - O(n) (divide la línea actual en una matriz de cadenas usando ; como delimitador)  Passenger passenger = new Passenger(atributs[0], Integer.parseInt(atributs[1]), Integer.parseInt(atributs[2]),Integer.parseInt(atributs[3]), Integer.parseInt(atributs[4]),atributs[5],Boolean.parseBoolean(atributs[6])); - O(1) (crea un objeto Passenger a partir de los atributos en la línea actual)  hashTable.insert(Integer.parseInt(atributs[1]),passenger); - O(1) (inserta el objeto Passenger en la tabla hash en función de su atributo id)  fis.close(); - O(1) (cierra el objeto FileInputStream)  } catch (FileNotFoundException error) { - O(1) (maneja la excepción FileNotFoundException)  error.printStackTrace(); - O(1) (imprime el rastro de pila de la excepción)  } catch (IOException error) { - O(1) (maneja la excepción IOException)  error.printStackTrace(); - O(1) (imprime el rastro de pila de la excepción) | | |
|
|

TYPE: Input

ARGUMENT: new Plane

SIZE OF BITS: 64 Bits

QUANTITY OF BITS: 1

TYPE: Auxiliary

ARGUMENT: hashTable

SIZE OF BITS: 32 Bits

QUANTITY OF BITS: 1

TYPE: Output

ARGUMENT: index

SIZE OF BITS: 32 Bits

QUANTITY OF BITS: 1

| 1. **Function orderPassenger** | | |
| --- | --- | --- |
| try (Stream<String> lines = Files.lines(Paths.get(filePath))) { - O(1) (crea un flujo de líneas a partir del archivo proporcionado)  lines.forEach(line -> { - O(n) (itera a través de cada línea del archivo)  String[] atributs = line.split(" "); - O(n) (divide la línea actual en una matriz de cadenas usando " " como delimitador)  Passenger passenger = hashTable.get(Integer.parseInt(atributs[0])); - O(1) (recupera el objeto Passenger de la tabla hash en función de su atributo id)  if (passenger != null) { - O(1) (verifica si el objeto Passenger existe en la tabla hash)  passenger.toString(); - O(1) (convierte el objeto Passenger a una cadena)  passenger.setTime(LocalTime.now()); - O(1) (establece la hora actual como la hora de llegada del pasajero)  if (passenger.getTime().isBefore(getTimeOfPlane())) { - O(1) (verifica si la hora de llegada del pasajero es anterior a la hora de salida del avión)  passenger.setMiles(passenger.getMiles() + 100); - O(1) (agrega 100 millas al pasajero)  if (passenger.getFirstClass()) { - O(1) (verifica si el pasajero está en primera clase)  firstClass.insert(passenger); - O(1) (inserta el pasajero en la cola de primera clase)  } else { - O(1)  turistClass.enqueue(passenger); - O(1) (inserta el pasajero en la cola de clase turista)  } else { - O(1) (maneja el caso en el que no se encuentra el objeto Passenger en la tabla hash)  System.out.println("No se encontro el pasajero, intenta nuevamente (Revisa que la direccion que estas poniendo este bien escrita y que en cada salto de arhcivo este un \\)"); - O(1) (imprime un mensaje de error)}); - O(1) | | |
|
|

TYPE: Input

ARGUMENT: new Plane

SIZE OF BITS: 64 Bits

QUANTITY OF BITS: 1

TYPE: Auxiliary

ARGUMENT: hashTable

SIZE OF BITS: 32 Bits

QUANTITY OF BITS: n

TYPE: Output

ARGUMENT: orderedPassenger

SIZE OF BITS: 32 Bits

QUANTITY OF BITS: 1

In conclusion we have total spatial complexity = Input + Auxiliary + Output = n + 2 = O(n)

Auxiliary spatial complexity = n = O(n)

Auxiliary + Output spatial complexity = n + 1 = O(n)