Universidad de los Andes

Ingeniería de Sistemas y Computación
ISIS1206 – Estructura de Datos
Taller 3

Taller 3: Ordenamiento

Objetivos

- Estudiar, implementar, probar y documentar tres algoritmos de ordenamiento
- Utilizar adecuadamente herramientas para el desarrollo de software en equipos

Lectura Previa

Estudiar la teoría de los algoritmos de ordenamiento: *Selection sort, Insertion sort, Shell sort, Merge sort, y Quick sort*. Consultar las secciones 2.1, 2.2 y 2.3 del libro guía "*Algorithms*" de Sedgewick y Wayne.

Manejo de Conflictos en Repositorios (en caso de múltiples usuarios modificando partes comunes en un repositorio)

Consulte cómo resolver conflictos en git en los siguientes enlaces:

- https://githowto.com/resolving_conflicts
- https://backlog.com/git-tutorial/syncing-repositories/

Lo que su grupo (en parejas) debe hacer

Parte 1 – Configuración Repositorio

1. Haga un *fork* del taller 0 de base, disponible en el URL https://bitbucket.org/isis1206/esqueleto_t0_201920.git. Su taller debe llamarse T3_201920.
   Si tiene dudas de cómo realizar este paso consulte la guía del taller1.
2. El taller se debe trabajar en los **grupos del proyecto**. Configurar el acceso al repositorio al compañero de grupo en modo Administrador (Admin) y al profesor y los monitores en modo Lectura (*Read*).
3. Clone el repositorio remoto de su cuenta en su computador (repositorio local).
4. Su primer *commit* al repositorio debe ser el archivo README.txt del repositorio donde aparezcan los nombres y códigos de los miembros del grupo de trabajo.

5. Descargue la información del archivo de viajes UBER en Bogotá agregados por hora para el segundo trimestre de 2018 (formato CSV) que se encuentran publicados en el aula Sicua+ Unificada del curso. Archivo: *bogota-cadastral-2018-2-All-HourlyAggregate.csv.* Cree la carpeta (Folder) *data* en el proyecto y copie este archivo (CSV) en esta carpeta.
6. Verifique la estructura del archivo de datos. En este archivo cada viaje (línea) tiene la estructura:
   sourceid,dstid,hod,mean_travel_time,standard_deviation_travel_time,geometric_mean_travel_time,geometric_standard_deviation_travel_time

   Cada línea contiene la información agregada de los viajes UBER entre una zona origen y una zona destino para una hora dada. Los campos de información contenidos para los viajes en una hora (una línea en el archivo) son:
   - *sourceid*: Identificador único de la zona de origen de los viajes relacionados
   - *dstid*: Identificador único de la zona de destino de los viajes relacionados
   - *hod (hour of day)*: hora (entera) del día del servicio relacionado
   - *mean_travel_time*: tiempo promedio en segundos de los viajes relacionados
   - *standard_deviation_travel_time*: desviación estándar de los viajes relacionados
   - *geometric_mean_travel_time*: tiempo promedio geométrico en segundos de los viajes relacionados
   - *geometric_standard_deviation_travel_time*: desviación estándar geométrica de los viajes relacionados

7. Configurar el API (jar) para la lectura de archivos CSV en el proyecto eclipse.
8. Realizar el procedimiento para confirmar los cambios y subir el desarrollo de la primera parte del taller al repositorio remoto T3_201920 en su cuenta Bitbucket/Github.

Parte 2 – Desarrollo

1. Definir la clase que representa los viajes, clase **UBERTrip implements Comparable<UBERTrip>** en el paquete **model.logic**. La comparación "natural" (método compareTo(…)) de esta clase debe hacerse por su *mean_travel_time*. Si el *mean_travel_time* de los viajes comparados es igual, la comparación la resuelve su *standard_deviation_travel_time*.

A continuación se listan los requerimientos a resolver, cada requerimiento debe implementarse en un método de la clase **MVCModelo** del paquete **model.logic**:

2. Realizar la carga del archivo descargado de los viajes UBER manteniendo el orden en el que aparecen en el archivo.
   Los viajes deben quedar almacenados como objetos **UBERTrip** en una estructura lineal de su preferencia (ArregloDinamico<UBERTrip> o Lista<UBERTrip>). La estructura lineal escogida debe ser genérica de objetos Comparables.
   Como información al usuario debe mostrarse el total de viajes (líneas) leídos del archivo. Adicionalmente debe mostrarse la zona origen, la zona destino, la hora y el tiempo promedio para el primer viaje y para el último viaje en el trimestre.

3. Consultar los viajes de una hora dada.
   a. El método debe retornar los viajes resultantes en una representación tipo arreglo de objetos Comparables.
   b. Los viajes consultados deben mantenerse en la estructura de datos de viajes.
   c. Como información al usuario debe mostrarse el número de viajes resultantes de la consulta.

4. Ordenar ascendentemente los viajes resultantes de la consulta del punto 3 (en el arreglo de objetos Comparables) usando el algoritmo ShellSort.
   a. Implementar el método de ordenamiento ShellSort de un arreglo de objetos Comparables. El criterio de comparación está definido por el método compareTo(…) de los objetos a ordenar.
   b. Definir las pruebas unitarias del ordenamiento ShellSort para tres conjuntos de datos de prueba: datos desordenados, datos ordenados ascendentemente y datos ordenados descendentemente.
   c. Como información al usuario debe mostrarse: el tiempo en milisegundos que tomó el algoritmo realizando el ordenamiento, los 10 primeros viajes y los 10 últimos viajes resultado del ordenamiento.

5. Ordenar ascendentemente los viajes resultantes de la consulta del punto 3 (en el arreglo de objetos comparables) usando el algoritmo MergeSort.
   a. Implementar el método de ordenamiento MergeSort de un arreglo de objetos Comparables. El criterio de comparación está definido por el método compareTo(…) de los objetos a ordenar.
   b. Definir las pruebas unitarias del ordenamiento MergeSort para tres conjuntos de datos de prueba: datos desordenados, datos ordenados ascendentemente y datos ordenados descendentemente.
   c. Como información al usuario debe mostrarse: el tiempo en milisegundos que se tomó el algoritmo realizando el ordenamiento, los 10 primeros viajes y los 10 últimos viajes resultado del ordenamiento.

6. Ordenar ascendentemente los viajes resultantes de la consulta del punto 3 (en el arreglo de objetos comparables) usando el algoritmo QuickSort.
   a. Implementar el método de ordenamiento QuickSort de un arreglo de objetos Comparables. El criterio de comparación está definido por el método compareTo(…) de los objetos a ordenar.
   b. Definir las pruebas unitarias del ordenamiento QuickSort para para tres conjuntos de datos de prueba: datos desordenados, datos ordenados ascendentemente y datos ordenados descendentemente.
   c. Como información al usuario debe mostrarse: el tiempo en milisegundos que se tomó el algoritmo realizando el ordenamiento, los 10 primeros viajes y los 10 últimos viajes resultado del ordenamiento.

Características y Análisis de Algoritmos de Ordenamiento

7. **Caso General de datos**: Los datos obtenidos de una consulta por la hora del día (valor entero en el rango [0, 23]) están desordenados. Se quiere comparar la eficiencia en tiempo de los tres algoritmos de ordenamiento para los viajes UBER resultado de una misma consulta por la hora del día. Para este objetivo, hay que cronometrar el tiempo de ejecución (en milisegundos) de cada algoritmo de ordenamiento.

   El tiempo de ejecución de un algoritmo de ordenamiento se puede medir con las siguientes instrucciones:

```
long startTime = System.currentTimeMillis();     // medición tiempo actual
this.algoritmoOrdenarXXXXX(viajes_A_Ordenar);
long endTime = System.currentTimeMillis();     // medición tiempo actual
long duration = endTime - startTime;                // duracion de ejecucion del algoritmo
view.printMensage("Tiempo de ordenamiento XXXXX: " + duration + " milisegundos");
```

Preparar un documento de texto (.docx o pdf) con:

8. Documentación de las características teóricas de los algoritmos de ordenamiento
   a. Para cada algoritmo explicar cuando se presenta su peor caso y su mejor caso. Para cada caso, dar la complejidad teórica en tiempo usando la notación O(…).

   b. Indicar para cada algoritmo si cumple las siguientes propiedades: algoritmo *InPlace*, algoritmo Adaptativo y algoritmo Estable.

      Resuma la información anterior para cada algoritmo (ShellSort, MergeSort y QuickSort) en la siguiente tabla:

Resumen de Información de un Algoritmo de Ordenamiento

Nombre del Algoritmo	…<completar>
Mejor caso	Este algoritmo presenta su mejor caso cuando … <completar>
Complejidad en el mejor caso (notación O)	O(…)
Peor caso	Este algoritmo presenta su peor caso cuando … <completar>
Complejidad en el peor caso (notación O)	O(…)
Algoritmo *Inplace*	Si / No
Algoritmo Adaptativo	Si / No
Algoritmo Estable	Si / No

c. Incluir una tabla de comparación de los tiempos de ejecución de los tres algoritmos para 6 consultas de viajes, cada una correspondiente a una hora del día, del **Caso General**.

Tabla de Comparación de Tiempos para seis conjuntos de datos

Hora de consulta de viajes	Número de viajes a ordenar	Shellsort (mseg)	Mergesort (mseg)	Quicksort (mseg)
	#Viajes Promedio: ¿?	Tiempo Promedio: ¿?	Tiempo Promedio: ¿?	Tiempo Promedio: ¿?

d. Dar una conclusión que precise cuál de los tres algoritmos implementados es mejor en tiempo de ejecución en el caso general.

Conclusión: Por el tiempo promedio de ejecución, para el caso general, el algoritmo más eficiente es <…>. El siguiente algoritmo en eficiencia es <…>. El algoritmo menos eficiente es <…>.

Incluir el documento preparado en la carpeta docs de su proyecto Eclipse.

9. Hacer la actualización de su repositorio T3_201920 en su computador y en su cuenta Bitbucket/Github.

Entrega

Para hacer la entrega del taller usted debe agregar a su repositorio BitBucket/Github los correos de los monitores y su profesor (modo Read), siguiendo las instrucciones del documento “Guía Creación de Repositorios para Talleres y Proyectos.docx”.
Si No da acceso a su repositorio a los monitores y al profesor, el taller No podrá ser calificado.