Universidad de los Andes

Ingeniería de Sistemas y Computación
ISIS1206 – Estructura de Datos
Taller 4

Taller 4: Ordenamiento

Objetivos

- Estudiar, implementar, probar y documentar dos 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.

Lo que su grupo (en parejas) debe hacer

Parte 1 – Trabajo en casa

1. Haga un fork del taller, disponible en el sitio (https://isis1206@bitbucket.org/isis1206/esqueleto_t4_201910.git o https://github.com/le99/esqueleto_t4_201910.git). Su taller debe llamarse T4_201910.
   Si tiene dudas de cómo realizar este paso consulte la guía del taller1.
2. 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. El taller se debe trabajar en los grupos del proyecto. Cree 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 infracciones para el mes de Enero, Febrero y Marzo de 2018 (formato CSV) que se encuentra en el siguiente enlace:
   - http://opendata.dc.gov/datasets/moving-violations-issued-in-january-2018/data
   - http://opendata.dc.gov/datasets/moving-violations-issued-in-february-2018/data
   - http://opendata.dc.gov/datasets/moving-violations-issued-in-march-2018/data

   Copie las versiones de estos archivos (CSV) en la carpeta data del repositorio.

6. Definir e implementar la(s) Estructura de Datos (su interface y su implementación) genérica de tipo T extends Comparable <T>. En esta estructura se van a almacenar las infracciones. Definirla en el paquete model.data_structures.
7. Completar la definición e implementación de la clase que representa las infracciones, clase VOMovingViolation implements Comparable<VOMovingViolation> en el paquete model.vo. La comparación "natural" (método compareTo(…)) de las infracciones debe hacerse por su TicketIssueDate. Si el TicketIssueDate de las infracciones comparadas es igual, la comparación la resuelve su ObjectId.
8. Implementar el método loadMovingViolations() en la clase Controller que carga las infracciones registradas en los archivos CSV para los meses: Enero, Febrero y Marzo. Las infracciones deben almacenarse en la estructura de datos definida.
9. Realice el procedimiento para confirmar los cambios y subir el desarrollo de la primera parte del taller al repositorio remoto T4_201910 en su cuenta Bitbucket.

Parte 2 – Trabajo en clase

1. Implementar el método generarMuestra(…) en la clase Controller que permite generar una muestra aleatoria de tamaño N de infracciones. El valor N es un dato de entrada del requerimiento (valor máximo 240.000).
    a. El método debe generar la muestra en una representación arreglo de objetos comparables.
    b. Las infracciones usadas en la muestra deben mantenerse en la estructura de datos.
2. Implementar el método invertirMuestra(…) en la clase Controller que permite invertir una muestra de datos.
3. Implementar el método ordenarShellSort(…) en la clase Sort que ordena ascendentemente una muestra utilizando el algoritmo ShellSort.
    a. Definir las pruebas unitarias del ordenamiento en la clase SortTest para diferentes muestras de datos.
4. Implementar el método ordenarMergeSort(…) en la clase Sort que ordena ascendentemente una muestra utilizando el algoritmo MergeSort.
    a. Definir las pruebas unitarias del ordenamiento en la clase SortTest para diferentes muestras de datos.
5. Implementar el método ordenarQuickSort(…) en la clase Sort que ordena ascendentemente una muestra utilizando el algoritmo QuickSort.
    a. Definir las pruebas unitarias del ordenamiento en la clase SortTest para diferentes muestras de datos.

Análisis de Algoritmos

6. Caso general de datos (**datos aleatorios**): Ejecutar cada algoritmo de ordenamiento para una misma muestra aleatoria de tamaño de 30.000 infracciones. Anote los tiempos de ejecución de la aplicación de cada algoritmo.

7. Caso con **datos ordenados ascendentemente**: Utilizar la muestra resultado de un ordenamiento (paso anterior) como muestra para realizar un ordenamiento (ejecutar la opción 8). Ejecutar cada algoritmo de ordenamiento para la misma muestra ordenada. Anote los tiempos de ejecución de la aplicación de cada algoritmo de ordenamiento.
8. Caso con **datos ordenados descendentemente**: Invertir la muestra resultado de un ordenamiento (paso anterior) para generar una nueva muestra ordenada descendentemente (ejecutar la opción 9). Ejecutar cada algoritmo de ordenamiento (ascendente) para la misma muestra ordenada descendentemente. Anote los tiempos de ejecución de la aplicación de cada algoritmo de ordenamiento.
9. Realice los 3 pasos anteriores para una muestra de tamaño 60.000, 90.000, 120.000, 150.000, 180.000, 210.000 y 240.000 infracciones.

10. Preparar en grupo un único documento (.docx o pdf) con:
    a. Para cada algoritmo de ordenamiento (ShellSort, MergeSort, QuickSort), explicar cuando se presenta su peor caso y su mejor caso. Para estos casos dar la complejidad teórica en tiempo usando la notación O(…) y la notación ~.
    b. Indicar para cada algoritmo si cumple las siguientes propiedades: algoritmo *InPlace*, algoritmo Adaptativo y algoritmo Estable.
    c. Incluir una tabla de comparación de los tiempos de ejecución de los algoritmos para las 8 muestras del **Caso General**.
    Dar una conclusión que precise cual de los tres algoritmos implementados es mejor en tiempo de ejecución en el caso general.
    La tabla de comparación de tiempos es de la forma:

Tamaño muestra	Shellsort (mseg)	Mergesort (mseg)	Quicksort (mseg)
30000			
60000			
90000			
...			

    d. Incluir una tabla de comparación de los tiempos de ejecución de los algoritmos para las 8 muestras con **datos ordenados (ascendentemente)**.
    Dar una conclusión que precise cual de los tres algoritmos implementados es mejor en tiempo de ejecución en este caso.
    e. Incluir una tabla de comparación de los tiempos de ejecución de los algoritmos para las 8 muestras con **datos ordenados descendentemente**.
    Dar una conclusión que precise cual de los tres algoritmos implementados es mejor en tiempo de ejecución en este caso.
    f. Incluir el documento preparado en la carpeta docs de su proyecto Eclipse.

11. Hacer la actualización de su repositorio T4_201910 en su computador y en su cuenta Bitbucket.

Entrega

1. Para hacer la entrega del taller usted debe agregar a su repositorio los usuarios de los monitores y su profesor, siguiendo las instrucciones del documento “Guía Creación de Repositorios para Talleres y Proyectos.docx”.
2. Entregue su taller por medio de BitBucket. Recuerde, dar acceso a su profesor y monitores (modo Read) para que pueda ser calificado.