Universidad de los Andes

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

Taller 3: Pilas y Colas

Objetivos

- Crear implementaciones de estructuras de datos de los tipos de datos abstractos Pila y Cola.
- Utilizar adecuadamente herramientas para el desarrollo de software en equipos

Lectura Previa

Pila

Una pila es un tipo de dato abstracto que representa un contenedor lineal de elementos. Una pila sigue el principio “el último en entrar, el primero en salir” (Last In, First Out en inglés, por lo que se llaman LIFO). La pila ofrece dos operaciones *push* y *pop.* La operación *push* inserta un nuevo elemento en el tope de la pila (elemento más reciente) y la operación *pop* obtiene/saca el último elemento insertado.

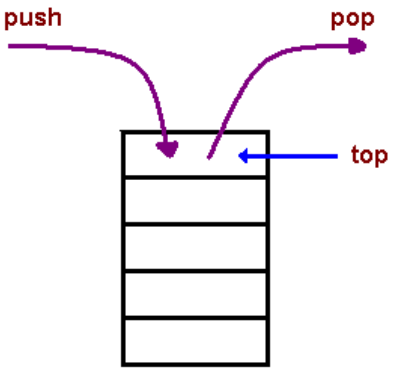

Ilustración 1 Acceso de elementos en una Pila . Imagen tomada de https://www.cs.cmu.edu/~adamchik/15-121/lectures/Stacks%20and%20Queues/Stacks%20and%20Queues.html

Cola

Una cola es un tipo de dato abstracto que representa un contenedor lineal de elementos. Una cola sigue el principio “el primero en entrar, el primero en salir” (First In, First Out en inglés, por lo que se llaman FIFO). La cola ofrece dos operaciones *enqueue* y *dequeue.* La operación *enqueue* inserta un nuevo elemento al "final" de la cola (elemento más reciente) y la operación *dequeue* obtiene/saca el elemento del "principio" de la cola (elemento menos reciente).

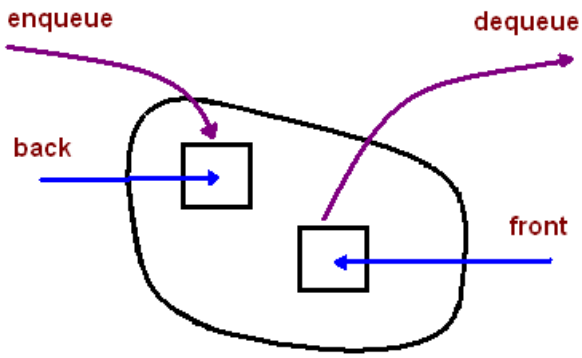

Ilustración 2 Acceso de elementos en Colas Imagen tomada de https://www.cs.cmu.edu/~adamchik/15-121/lectures/Stacks%20and%20Queues/Stacks%20and%20Queues.html

Descripción General

"La accidentalidad vial en Colombia se ha convertido en la segunda manera de muerte violenta y según la Organización Mundial de la Salud (OMS), en el mundo cerca de seis millones de personas mueren por este evento, convirtiéndose así el trafico automotor en la primera manera de muerte violenta en el nivel mundial.[1]" Medicina Legal, Lesiones no Intencionales.

Para prevenir los accidentes viales los gobiernos usan policías de tránsito, cámaras, etc. los cuales producen mucha información. Para que las políticas del gobierno sean más efectivas esta información debe ser tenida en cuenta en la toma de decisiones. Sin embargo, el problema es que la información no siempre es accesible al encargado de tomar las decisiones.

El tema del proyecto está relacionado con la información de infracciones en movimiento emitidas por el Departamento de Policía Metropolitana (MPD) del Distrito de Columbia y agencias asociadas con la autoridad. Esta ciudad está a la vanguardia en el registro de información en diferentes aspectos de su funcionamiento (portal oficial de datos *Open Data DC* http://opendata.dc.gov/). Para el análisis del sistema de información de infracciones en movimiento utilizaremos como fuente de información, el portal Web de consulta http://opendata.dc.gov/datasets?q=moving%20violations.

En este proyecto, vamos a construir una aplicación que le permita entender a los administradores y autoridades de la ciudad de Washington D.C. un conjunto de consultas importantes sobre la información de infracciones en movimiento.

Lo que usted debe hacer

Parte 1 – Trabajo en casa

[1] Tomado de: http://www.medicinalegal.gov.co/documents/20143/49484/Muertes+Transito.pdf/ad2ae841-ed99-d524-66bf-1b70dec0b44a

1. Haga un fork del taller, disponible en el sitio (https://isis1206@bitbucket.org/talleres/esqueleto_t3_201910.git o https://github.com/le99/esqueleto_t3_201910.git). Su taller debe llamarse T3_201910. Si tiene dudas de cómo realizar este paso consulte la guía del taller1.
2. El taller se debe trabajar en los grupos del proyecto. Su primer *commit* al repositorio debe ser un archivo README.txt con los nombres y códigos de los estudiantes conformando su grupo.
3. Descargue la información del archivo de infracciones para el mes de enero y febrero de 2018 (formato CSV) que se encuentra en el siguiente enlace: http://opendata.dc.gov/datasets/moving-violations-issued-in-january-2018 y http://opendata.dc.gov/datasets/moving-violations-issued-in-february-2018
4. Verifique la estructura de los archivos de datos. En estos archivos cada línea tiene la estructura:
    - OBJECTID: Identificador único de la infracción.
    - ROW_:
    - LOCATION: Dirección en formato de texto.
    - ADDRESS_ID: ID de la dirección.
    - STREETSEGID: ID del segmento de la calle.
    - XCOORD: Coordenada X donde ocurrió (No corresponde a longitud geográfica).
    - YCOORD: Coordenada Y donde ocurrió (No corresponde a latitud geográfica).
    - TICKETTYPE:
    - FINEAMT: Cantidad a pagar por la infracción USD.
    - TOTALPAID: Cuanto dinero efectivamente pagó el que recibió la infracción en USD.
    - PENALTY1: Dinero extra que debe pagar el conductor.
    - PENALTY2: Dinero extra que debe pagar el conductor.
    - ACCIDENTINDICATOR: Si hubo un accidente o no.
    - AGENCYID:
    - TICKETISSUEDATE: Fecha cuando se puso la infracción.
    - VIOLATIONCODE: código de la infracción.
    - VIOLATIONDESC: descripción textual de la infracción.
    - ROW_ID:
5. Implementar APIs para Pila (*Stack*) y Cola (*Queue*) dentro del paquete *model.data_structures*. Tenga en cuenta que estas estructuras deben ser utilizadas con cualquier tipo de datos (i.e., deben ser estructuras genéricas). Puede implementar cada una de las estructuras con una lista encadenada o con un arreglo dinámico.
   Coloque en la carpeta docs el diseño UML de las dos estructuras de datos.
6. Cree (en la carpeta test/src) las pruebas unitarias de las implementaciones de las estructuras de datos Pila y Cola, pruebe cada uno de los servicios de las estructuras.
7. Consulte cómo resolver conflictos en git en el siguiente enlace https://githowto.com/resolving_conflicts

Parte 2 – Trabajo en clase

Se desea crear una aplicación que permita entender a los administradores y autoridades de la ciudad de Washington un conjunto de consultas importantes sobre las infracciones en movimiento. En ese sentido usted debe:

1. Implemente el método `loadMovingViolations( )` en la clase `MovingViolations`Manager del paquete **model.logic**. Las infracciones se deben cargar en una Pila (Stack) y en una Cola (Queue) en el orden que aparecen en el archivo *Moving_Violations_Issued_in_January_2018_ordered.csv* y *Moving_Violations_Issued_in_February_2018_ordered.csv* (están ordenada por TICKETISSUEDATE).
2. Implemente un método *getDalyStatistics()* que usa la Cola para retornar las siguientes estadísticas de todos los días: fecha del día, número de accidentes, número de infracciones y suma total de FINEAMT de las infracciones ese día. La respuesta debe estar ordenada por la fecha del día y debe ser retornada en una cola.
3. Implementar un método *nLastAccidents(int n)* que usa la Pila para retornar la información de las ultimas "n" infracciones que tuvieron un accidente (*nAccidents(1)* retorna el último accidente, *nAccidents(2)* retorna el último y penúltimo accidente, y así sucesivamente). En consola se debe mostrar de cada infracción el OBJECTID, TICKETISSUEDATE, LOCATION y VIOLATIONDESC.

Entrega

1. Para hacer la entrega del taller usted debe agregar a su repositorio los usuarios de los monitores y su profesor con acceso de Lectura (Read), siguiendo las instrucciones del documento "Guía Creación de Repositorios para Talleres y Proyectos.docx".
2. Entregue el taller en su cuenta BitBucket/GitHub. Recuerde, si su cuenta no contiene el taller o está vacío, su taller no será calificado por más de que lo haya desarrollado.