Universidad de los Andes

Ingeniería de Sistemas y Computación
ISIS1206 – Estructura de Datos
Taller 2

Taller 2: Pilas y Colas

Objetivos

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

Lectura Previa

Pila (Stack)

Una pila (*stack*) es un tipo abstracto de dato 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 las siguientes operaciones básicas:

- *push:* Inserta un nuevo elemento en el tope de la pila (elemento más reciente)
- *pop:* Saca/elimina el elemento tope de la pila (elemento más reciente) y lo retorna
- Consultar el tamaño (número de elementos) de la pila
- Consultar si la pila está vacía
- Consultar el elemento tope de la pila sin eliminarlo

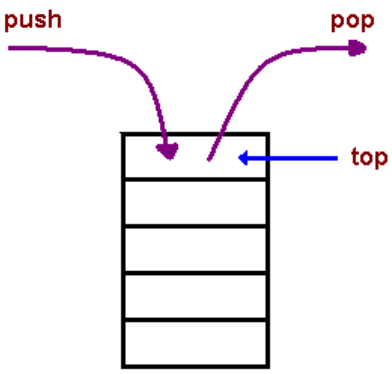

Ilustración 1 Acceso de elementos en una Pila . Imagen tomada de https://www.cs.cmu.edu/~adamchik/15-121/lectures/Stacks%20and%20Queues/Stacks%20and%20Queues.html

Cola (Queue)

Una cola (*queue*) es un tipo abstracto de dato 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 las siguientes operaciones básicas:

- *enqueue:* inserta un nuevo elemento al "final" de la cola (elemento más reciente)
- *dequeue:* saca/elimina el elemento del "principio" de la cola (elemento menos reciente) y lo retorna.
- Consultar el tamaño (número de elementos) de la cola
- Consultar si la cola está vacía
- Consultar el elemento del "principio" de la cola sin eliminarlo

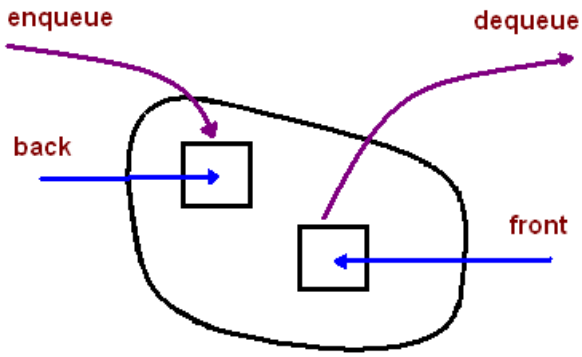

Ilustración 2 Acceso de elementos en Colas Imagen tomada de https://www.cs.cmu.edu/~adamchik/15-121/lectures/Stacks%20and%20Queues/Stacks%20and%20Queues.html

Descripción General

El transporte es una de las problemáticas importantes en una ciudad. En particular, el transporte público debe ofrecer soluciones eficientes para transportar un alto volumen de la población, con buena calidad, para así aumentar la productividad de la ciudad y la calidad de vida de sus ciudadanos. Además, un buen servicio de transporte público puede incentivar la reducción del transporte privado. En este aspecto, el servicio de taxis se viene transformando y mejorando con la prestación del servicio a través de plataformas tecnológicas. Entre las empresas que vienen impulsando este cambio a nivel nacional están Uber, Beat, Cabify, Didi, Picap, Emobi entre otras.

El tema del proyecto está relacionado con el manejo de información de la prestación del servicio de Uber en Bogotá. Esta plataforma ofrece información de su servicio en diferentes ciudades del mundo. Para el análisis de su servicio utilizaremos como fuente de información, el portal web de consulta disponible en https://movement.uber.com.

En este proyecto, les suministraremos la información obtenida a través del portal Web.

El propósito es construir una aplicación que nos permita entender un conjunto de consultas importantes sobre el servicio de Uber en Bogotá.

Lo que usted debe hacer

Parte 1 – Trabajo en laboratorio

1. Haga un *fork* del taller de base (taller 0) disponible en el URL bitbucket https://bitbucket.org/isis1206/esqueleto_t0_201920.git. Su taller debe llamarse T2_201920. Si tiene dudas de cómo realizar este paso consulte la guía del Taller 0.
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 primer trimestre de 2018 (formato CSV) que se encuentran publicados en el aula Sicua+ Unificada del curso. Archivo: *bogota-cadastral-2018-1-All-HourlyAggregate.csv*
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. Consulte cómo resolver conflictos en git en los siguientes enlaces:
   - https://githowto.com/resolving_conflicts
   - https://backlog.com/git-tutorial/syncing-repositories/

Parte 2 – Trabajo en laboratorio y en casa

1. Diseñe e Implemente los APIs de Pila (*Stack*) y Cola (*Queue*) propuestos con las operaciones básicas. Esta implementación debe realizarse dentro del paquete ***model.data_structures*** separando la interfaz de la implementación de cada estructura de datos. Tenga en cuenta que estas estructuras pueden ser utilizadas con cualquier tipo de datos (i.e., deben ser **estructuras genéricas**).

La implementación de cada estructura de datos puede utilizar una lista encadenada o un arreglo dinámico.

2. Cree la carpeta (*Folder*) docs en el proyecto y agregue el diseño de cada estructura de datos (imagen UML del diagrama de clases).
3. Implemente las pruebas unitarias de las estructuras de datos Pila y Cola en el paquete **test.data_structures**; pruebe cada uno de sus servicios (métodos).

4. Cree la carpeta (*Folder*) **data** en el proyecto y copie el archivo de viajes agregados por hora del primer trimestre del año (*bogota-cadastral-2018-1-All-HourlyAggregate.csv*)
5. Modifique el menú de servicios que ofrece la clase **Controller** del paquete **controller**. Los servicios que debe ofrecer la aplicación son:

- Realizar la carga del archivo de los viajes UBER del primer trimestre del año manteniendo el orden en el que aparecen en el archivo.
  La carga debe implementarse en un método en la clase **MVCModelo** del paquete **model.logic**. Los viajes deben quedar almacenados en una cola y en una pila.
  Como información al usuario debe mostrarse el total de viajes (líneas) leídos para el primer trimestre del año. 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.
- Procesar la cola resultante para buscar el grupo de viajes consecutivos (*cluster*) más grande que se encuentran ordenados ascendentemente por la hora a partir de una hora dada por el usuario. La respuesta acepta viajes consecutivos en la misma hora. La cola debe quedar vacía al final del procesamiento.
  Por ejemplo si se tiene la siguiente secuencia de viajes en la cola (representados como objetos donde la información de la primera línea es el primer viaje de la cola y la información de la última línea es el último viaje en la cola. Adicionalmente la hora está dada por el tercer dato en cada viaje.):

4,5,**13**,1440.57,444.47,1381.48,1.33
4,6,**20**,1125.34,531.04,1062.68,1.35
4,9,**3**,901.89,198.16,878.76,1.26
5,6,**9**,2757.13,467.44,2716.74,1.19
2,8,**9**,1834.13,347.44,1814.74,1.09
6,4,**22**,1111.97,656.62,1029.94,1.41
6,8,**12**,1781.58,609.61,1675.14,1.43
6,9,**19**,1116.92,436.53,1053.76,1.39
7,3,**21**,976.92,386.53,953.76,1.19

  En este ejemplo, se encuentran tres grupos consecutivos de viajes (clústeres) agrupados por hora (tercera columna) de manera ascendente: el grupo rojo (en su orden con hora 13 y 20), el grupo azul (en su orden con hora 3, 9, 9 y 22) y el grupo verde (en su orden con hora 12, 19 y 21).

4,5,**13**,1440.57,444.47,1381.48,1.33
4,6,**20**,1125.34,531.04,1062.68,1.35

4,9,**3**,901.89,198.16,878.76,1.26
5,6,**9**,2757.13,467.44,2716.74,1.19
2,8,**9**,1834.13,347.44,1814.74,1.09
6,4,**22**,1111.97,656.62,1029.94,1.41
6,8,**12**,1781.58,609.61,1675.14,1.43
6,9,**19**,1116.92,436.53,1053.76,1.39
7,3,**21**,976.92,386.53,953.76,1.19

El grupo de viajes consecutivos más grande es el grupo azul porque tiene 4 viajes ordenados por hora.

Esta consulta debe implementarse en un método en la clase **MVCModelo** del paquete **model.logic** y debe retornar una cola con los viajes resultantes. La hora inicial de referencia debe ser un parámetro del método.
Como información al usuario debe mostrarse el número de viajes del grupo resultado y por cada viaje debe informarse (en orden): la hora, la zona origen, la zona destino y el tiempo promedio.

- Procesar la pila resultante para reportar los últimos N viajes para una hora dada. La pila se procesa hasta resolver la consulta o hasta que la pila quede vacía.
  Esta consulta debe implementarse en un método en la clase **MVCModelo** del paquete **model.logic** y debe retornar una cola con los viajes resultantes. El valor de N y la hora de consulta deben ser parámetros del método.
  Como información al usuario debe mostrarse por cada viaje de la solución (en orden): la hora, la zona origen, la zona destino y el tiempo promedio.

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.