Trabajo Práctico 2

El Trabajo Práctico número 2 es de elaboración grupal, y tiene fecha de entrega para el **18/11**.

Contenido

- Introducción
  - Archivos
  - Tablero de despegues
  - Informacion de un vuelo
  - Prioridad de los vuelos
  - Siguientes vuelos a una fecha
  - Borrar datos de vuelos
- Interfaz
  - Agregar_archivo
  - Ver_tablero
  - Info_vuelo
  - Prioridad_vuelos
  - Siguiente_vuelo
  - Borrar
- Consideraciones adicionales
  - Archivos entrantes
  - Tiempos de ejecución
- Ejemplos
  - Agregar_archivo
  - Ver_tablero
  - Info_vuelo
  - Prioridad_vuelos
  - Siguiente_vuelo
  - Borrar
- Set de Pruebas
- Criterios de aprobación
  - Estructuras de datos
  - Complejidad de los comandos
  - Código del programa
  - Informe
  - Entrega

Introducción

El aeropuerto de Algueiza opera la entrada y salida de aviones; los operarios de dicho aeropuerto nos han pedido que implementemos un sistema de consulta de vuelos que les permita: ordenar, filtrar, analizar y obtener información de los distintos vuelos.

Archivos

El sistema que actualmente tiene Algueiza genera continuamente archivos en formato `.csv` que contienen la información de cada vuelo. Nuestro sistema deberá procesar estos archivos y responder a las consultas que los operarios necesiten realizar. El sistema tiene que ser capaz de recibir en cualquier momento un nuevo archivo conteniendo el detalle de vuelos (nuevos o viejos) y actualizar sus datos.

Tablero de despegues

El primer requerimiento que mencionaron los operarios fue que nuestro sistema permita ordenar los registros de despegues por fecha. Esto debe realizarse de forma ascendente o descendente según el criterio del usuario. Adicionalmente nos mencionaron que no están interesados en visualizar todo el listado, por lo que también es necesario que puedan ingresar un intervalo de fechas de despegue (min / max) y una cantidad de vuelos a listar.

Informacion de un vuelo

Existen ocasiones en las cuales un operario necesita ver información detallada de un vuelo. Por este motivo es necesario que, ingresando un código de vuelo, el sistema devuelva toda la información posible asociada al mismo.

Prioridad de los vuelos

Algueiza tiene un sistema de prioridades sobre los vuelos que necesitan atención especial, y los operarios necesitan saber cuales son esos vuelos. Nuestro sistema deberá ser capaz de devolver los *K* vuelos de mayor prioridad.

Siguientes vuelos a una fecha

Es posible que los usuarios, o administradores, deseen consultas por vuelos posteriores a una determinada fecha para hacer una conexión. Esto puede ser porque un adminsitrador necesita obtener información para reubicar pasajeros luego de una demora inesperada, o por usuarios que desean consultar en caso que necesiten cambiar fechas o simplemente consultar vuelos.

Borrar datos de vuelos

Finalmente, los operarios nos mencionaron que la información que maneja el sistema pierde validez con el paso del tiempo. Ciertas entradas pierden importancia y deben ser eliminadas periódicamente del sistema.

Con este fin se debe incorporar la posibilidad de eliminar información de vuelos que estén en un rango de fechas de despegue (min / max).

Interfaz

Es necesario implementar una interfaz del programa, que leerá por entrada estándar los siguientes comandos:

- `agregar_archivo <nombre_archivo>`: procesa de forma completa un archivo de `.csv` que contiene datos de vuelos.
- `ver_tablero <K cantidad vuelos> <modo: asc/desc> <desde> <hasta>`: muestra los *K* vuelos ordenados por fecha de forma ascendente (asc) o descendente (desc), cuya fecha de despegue esté dentro de el intervalo `<desde>` `<hasta>` (inclusive).
- `info_vuelo <código vuelo>`: muestra toda la información posible en sobre el vuelo que tiene el código pasado por parámetro.
- `prioridad_vuelos <K cantidad vuelos>`: muestra los códigos de los *K* vuelos que tienen mayor prioridad.
- `siguiente_vuelo <aeropuerto origen> <aeropuerto destino> <fecha>`: muestra la información del vuelo (tal cual en `info_vuelo`) del próximo vuelo directo que conecte los aeropuertos de origen y destino, a partir de la fecha indicada (inclusive). Si no hay un siguiente vuelo cargado, imprimir `No hay vuelo registrado desde <aeropuerto origen> hacia <aeropuerto destino> desde <fecha>` (con los valores que correspondan).
- `borrar <desde> <hasta>`: borra todos los vuelos cuya fecha de despegue estén dentro del intervalo `<desde>` `<hasta>` (inclusive).

Si un comando es válido deberá imprimir `OK` por salida estándar después de ser ejecutado (esto incluso en el caso que no haya un siguiente vuelo en `siguiente_vuelo`). Si un comando no pertenece a los listados previamente o tiene un error, se imprime `Error en comando <comando>` por `stderr` y continua la ejecución.

El programa no tendrá un comando para terminar. Este finaliza cuando no quedan más líneas para procesar por entrada estándar. Al finalizar, es importante que se cierren correctamente todos los archivos procesados.

Agregar_archivo

El comando se acompaña del nombre de un archivo `csv`, accesible desde el mismo directorio donde se ejecuta el programa.

Ejemplo: `agregar_archivo vuelos-algueiza-parte-01.csv`

Al ejecutarse, el programa deberá leer el archivo, parsear cada línea sabiendo que tiene el formato CSV, y guardar cada línea en las estructuras auxiliares correspondientes para ser capaz de ejecutar los otros comandos. En caso de que dos líneas en el archivo tengan el mismo código de vuelo, el sistema se deberá quedar solamente con la información de la última aparición.

Este comando se puede ejecutar más de una vez durante el ciclo de uso de nuestro programa, y por lo tanto puede recibir nuevos archivos. Si un nuevo archivo procesado contiene un código de vuelo que ya se encuentra en nuestro sistema, nuestro programa deberá quedarse solamente con la información de la última aparición.

Ver_tablero

El comando recibe varios parámetros:

- *K*, cantidad de vuelos: número entero mayor a 0, que indica la cantidad de vuelos a mostrar
- modo, `asc` o `desc`: cadena con “asc” o “desc” como únicos posibles valores, indican el ordenamiento a elegir utilizando el campo *fecha de despegue*.

- desde: cadena en formato `YYYY-MM-DDTHH:MM:SS` que indica el tiempo desde que se tienen que mostrar los vuelos, los vuelos con una fecha de despegue anteriores al tiempo ingresado no se tienen que mostrar.
- hasta: cadena en formato `YYYY-MM-DDTHH:MM:SS` que indica el tiempo hasta que se tienen que mostrar los vuelos, los vuelos con una fecha de despegue posteriores al tiempo ingresado no se tienen que mostrar.

Ejemplo: `ver_tablero 5 asc 2018-10-10T00:01:00 2018-10-19T00:12:00`

Al ejecutarse deberán imprimir por salida estandar las fechas de despegue y los códigos de los vuelos que cumplan con la lógica anteriomente mencionada, cada vuelo se imprime en una nueva línea. El formato de cada línea tiene que ser `YYYY-MM-DDTHH:MM:SS - <código de vuelo>`. En el caso que dos vuelos tengan la misma fecha de despegue, se deberá mostrar los vuelos comparando por número de vuelo (tomado como cadena).

Ejemplo de salida:

```
2018-10-10T08:51:32 - 1234
2018-10-11T10:20:12 - 11123
2018-10-11T10:20:22 - 9113
2018-10-12T17:20:12 - 991223
2018-10-18T21:32:59 - 98123
OK
```

Casos en los que el sistema tiene que devolver error (con el formato anteriormente mencionado):

- recibir un K igual o menor a 0
- recibir un modo que no sea asc o desc
- recibir una fecha `hasta` que sea anterior a la fecha `desde`

Info_vuelo

Este comando debe mostrar toda la informacion del vuelo cuyo código de vuelo coincida con el que fue pasado por parámetro, el formato que tiene que mostrar de la información es exactamente igual a la línea leída del archivo original, salvo que en vez de comas se tienen espacios.

Ejemplo: `info_vuelo 4608`

Ejemplo de salida:

```
4608 OO PDX SEA N812SK 8 2018-04-10T23:22:55 05 43 0
OK
```

En caso de ingresar un código de vuelo que no exista dentro de los códigos procesados, se deberá devolver un mensaje de error con el formato anteriormente mencionado.

Prioridad_vuelos

Este comando deberá mostrar de mayor a menor, los *K* vuelos con mayor prioridad que hayan sido cargados en el sistema, de la forma `<prioridad> - <código de vuelo>`. Si dos vuelos tienen la misma prioridad, se desempatará por el código de vuelo mostrándolos de menor a mayor (tomado como cadena).

Ejemplo: `prioridad_vuelos 3`

Ejemplo de salida:

```
10 - 1234
3 - 1624
3 - 325
OK
```

Siguiente_vuelo

Este comando deberá mostrar la información del siguiente vuelo que haya para la conexión definida, considerando una fecha inicial. El formato de la información a mostrar es exactamente la misma que en `info_vuelo`.

Ejemplo: `siguiente_vuelo LAX MIA 2018-10-28T15:02:01`

Ejemplo de salida:

```
115 AA LAX MIA N3CTAA 14 2018-10-29T15:02:01 -2 255 0
OK
```

Borrar

El comando recibe dos argumentos:

- desde: cadena en formato YYYY-MM-DDTHH:MM:SS
- hasta: cadena en formato YYYY-MM-DDTHH:MM:SS

Al ejecutarse, todos los vuelos cuya fecha de despegue sea igual o mayor a *desde* e igual o menor a *hasta* tienen que ser borrados del sistema y mostrados por salida estándar. En caso de que se reciba una fecha *hasta* que sea menor a *desde*, se deberá devolver un mensaje de error con el formato anteriormente mencionado.

Consideraciones adicionales

Archivos entrantes

Cada línea de los archivos entrantes `.csv` tienen la siguiente estructura:

- FLIGHT_NUMBER: 4608
- AIRLINE: OO
- ORIGIN_AIRPORT: PDX
- DESTINATION_AIRPORT: SEA
- TAIL_NUMBER: N812SK
- PRIORITY: 8
- DATE: 2018-04-10T23:22:55
- DEPARTURE_DELAY: 05
- AIR_TIME: 43
- CANCELLED: 0

Cada dato de una línea de log está separado por una coma.

Ejemplo:

```
4608,OO,PDX,SEA,N812SK,8,2018-04-10T23:22:55,05,43,0
```

Se provee una colección de archivos de ejemplo.

Tiempos de ejecución

- Agregar_archivo: El mantenimiento para actualizar los vuelos debe ser $\mathcal{O}(V \log n)$ siendo V la cantidad de vuelos en el nuevo archivo y n la cantidad total de vuelos en el sistema.
- Ver_tablero: debe ser $\mathcal{O}(v)$ en el peor caso (en el que se tenga que mostrar todos los vuelos del sistema), $\mathcal{O}(\log v)$ en un caso promedio (en el caso en el que no se pidan mostrar demasiados visitantes). v es la cantidad de vuelos.
- Info_vuelo: debe ser $\mathcal{O}(1)$.
- Prioridad_vuelos: debe ser $\mathcal{O}(n + K \log n)$. Siendo K la cantidad de vuelos a mostrar y n la cantidad de vuelos en el sistema.
- Siguiente_vuelo: debe ser $\mathcal{O}(\log F_{conexion})$, siendo $F_{conexion}$ la cantidad de fechas diferentes en las que se puede hacer dicho viaje (si todos los vuelos de esa conexión son en fechas diferentes, sería lo mismo que la cantidad de vuelos que hacen dicha conexión). Considerar que $F_{conexion}$ es mucho menor a la cantidad de vuelos totales y cantidad de fechas totales.
- Borrar: debe ser $\mathcal{O}(K \log n)$. Siendo K la cantidad de vuelos que hay en el rango de fechas ingresado y n la cantidad de vuelos en todo el sistema.

Ejemplos

Se proveen ejemplos completos para mostrar la salida esperada en cada instrucción.

Agregar_archivo

Ejemplo: `agregar_archivo vuelos-algueiza-parte-01.csv`

Salida esperada:

Si el comando se ejecuta correctamente deberá imprimir `OK` por salida estándar después de ser ejecutado. Si se produce un error al procesar el archivo se imprime `Error en comando <comando>` por `stderr` y se continua la ejecución.

Ver_tablero

Ejemplo: `ver_tablero 3 asc 2018-04-08T10:00:00 2018-04-21T10:12:00`

Salida esperada:

```
2018-04-08T11:15:29 - 6391
2018-04-10T23:22:55 - 4608
2018-04-21T07:41:48 - 5460
OK
```

Info_vuelo

Ejemplo: `info_vuelo 4608`

Salida esperada:

```
4608 OO PDX SEA N812SK 8 2018-04-10T23:22:55 05 43 0
OK
```

Prioridad_vuelos

Ejemplo: `prioridad_vuelos 4`

Ejemplo de salida:

```
26 - 6391
15 - 1086
12 - 4701
OK
```

Siguiente_vuelo

Ejemplo:

```
siguiente_vuelo LAX MIA 2018-10-28T15:02:01
siguiente_vuelo LAX MIA 2023-10-28T15:02:01
```

Ejemplo de salida:

```
115 AA LAX MIA N3CTAA 14 2018-10-29T15:02:01 -2 255 0
OK
No hay vuelo registrado desde LAS hacia MIA desde 2023-10-28T15:02:01
OK
```

Borrar

Ejemplo: `borrar 2018-10-04T02:01:01 2018-11-01T03:00:00`

Ejemplo de salida:

```
4701 EV EWR CMH N11150 12 2018-10-04T04:19:24 -10 55 0
2807 MQ SGF DFW N604MQ 00 2018-10-05T13:57:14 00 107 0
OK
```

Set de Pruebas

En el el sitio de descargas pueden encontrar un archivo zip con las pruebas que necesitan pasar. Allí se encuentran los diferentes archivos que se usan para agregar al programa, así como las entradas y salidas esperadas (incluyendo salidas de error). Para ejecutar las pruebas:

1. Compilar el programa del tp2 para generar el ejecutable.
2. Ir al directorio de las pruebas.
3. Ejecutar el comando `./pruebas.sh <PATH AL EJECUTABLE DEL TP2>`, indicando justamente el path al ejecutable en cuestión (pudiendo ser una ruta relativa o absoluta).

Criterios de aprobación

Los siguientes aspectos son condición necesaria para la aprobación del trabajo práctico.

Estructuras de datos

Es necesario emplear la estructura de datos más apropiada para cada función del programa, en particular **teniendo en cuenta la complejidad temporal**. También es necesario utilizar dicha estructura de la forma más apropiada posible para optimizar de manera correcta su funcionamiento.

Se puede (y no sólo alentamos fuertemente, sino que pedimos) implementar otros TDAs o estructuras que faciliten o mejoren la implementación de este Trabajo Práctico. Les recordamos que un TDA no es un simple `struct`donde se guarda información, sino que debe tener comportamiento. Esto será un punto muy importante en la evaluación del Trabajo Práctico. No implementar diseñando correctamente, implicará una reentrega directa y sin mayor corrección.

No deben modificar los TDAs ya implementados hasta ahora.

Complejidad de los comandos

Se debe cumplir con la complejidad indicada para los comandos

Código del programa

El código entregado debe:

- ser claro y legible, y estar estructurado en funciones lo más genéricas posible, adecuadamente documentadas.
- compilar sin advertencias ni errores.
- ajustarse a la especificación de la consigna y pasar todas las pruebas automáticas.

Informe

Este trabajo práctico tiene como principal foco el diseño del mismo (modularización, creación de nuevas estructuras y TDAs específicos del TP) así como el uso apropiado de las estructuras de datos vistas en clase. Por lo tanto, se les pide que escriban un informe de dicho diseño puesto que, además de ser capaces de escribir código correcto, también es necesario que aprendan a poder *transmitir* el porqué de sus decisiones, utilizando el lenguage técnico y justificaciones correspondientes.

El informe deberá consistir de las siguientes partes:

- carátula con los datos personales del grupo, y ayudante asignado.
- análisis y diseño de la solución, en particular: algoritmos y estructuras de datos utilizados, justificando conforme a los requisitos pedidos en este TP. Además los TDAs específicos para la elaboración del TP.
- Es importante que el informe **no debe** ni hablar de código, ni “si tal struct o tal función hace tal cosa”. Para entender el código, está el código (que debe ser legible y documentado donde corresponda). La idea del informe es describir la solución a partir de qué problemas se encontraron, por qué utilizan una estructura de datos y no otra, y por qué la definición de un TDA con **qué comportamiento**.

Entrega

La entrega incluye, obligatoriamente, los siguientes archivos:

- El código de los TDAs utilizados. El código del TP. Este debe incluir el archivo `go.mod`. Todo el código debe estar en una estructura de directorios que permita automáticamente compilarlo.
- El informe (en formato `.pdf`).

La entrega se realiza exclusivamente en forma digital a través del sistema de entregas, con todos los archivos mencionados en un único archivo ZIP.