Projeto I de Algoritmos e Estruturas de Dados

Grupo 73

up202008169 – André Lima

up202006137 – Guilherme Almeida

up202004260 – Mariana Lobão

Descrição do problema

- Queria-se implementar um sistema que guardasse dados relativos a uma companhia aérea, desde informação acerca do aeroporto, aviões e locais de transporte público a informação sobre os bilhetes comprados, os passageiros e a respetiva bagagem.

A solução

- **Avião**
  - Para guardar informações sobre os voos a realizar usou-se uma lista, visto que a inserção é feita em tempo constante.
  - Para guardar informação sobre os serviços a realizar utilizou-se uma fila, visto que a ordem cronológica deve ser cumprida.
  - A informação sobre os serviços realizados foi guardada num vetor, visto que só é necessário adicionar elementos no fim ($\mathcal{O}(n)$) e que a pesquisa será linear ($\mathcal{O}(n)$). Guardar estes elementos numa lista iria ocupar mais espaço de memória, devido aos pointers armazenados.

- **Voo**
  - Em cada instância de um voo, é guardado uma referência para o avião onde esse voo se vai realizar, mantendo assim a integridade de classes.

A solução

- **Aeroporto**
  - Para guardar a informação sobre os locais de transporte público perto do aeroporto utilizou-se um set, uma vez que nunca existirá mais que uma instância de cada local transporte e a procura de cada um será realizada em tempo $\mathcal{O}(\log(n))$, no caso médio.

- **Carrinho de bagagens**
  - Para guardar informação do carrinho, guardaram-se as bagagens numa stack, pois são dispostas no carrinho umas em cima das outras, podendo apenas ser retirada a do topo. Foram posteriormente guardadas num deque, pois permite a procura por iteradores de acesso aleatório e acesso contanste a qualquer elemento, enquanto uma stack não o permite.
  - Estes decks foram armazenados num outro deque, que representa o carro na sua íntegra.

Diagrama de Classes

Airport

name: string
transportPlaceInfo: set<TransportPlace>

Airport(const string&)
const getName(): const string
const getTransportPlaceInfo(): const set<TransportPlace>
addTransportPlaceInfo(TransportPlace): void
removeAllTransportPLaceInfo(): void
removeTransportPlaceInfo(const string&): void
setName(string): const void
operator<<(ostream&, const Airport&): ostream&
const str(): string

TransportPlace

name: string
latitude: float
longitude: float
transport_type: transportType
airport_distance: float
schedule: set<Time>
operator<(const TransportPlace&): bool

{ TransportType é do tipo unsigned int e pode ter os valores 0, 1 ou 2 }

Flight

plane: Plane&
flight_id: string
departure_time: Datetime
duration: Time
origin: Airport&
destination: Airport&
tickets: vector<Ticket*>
luggage: vector<Luggage*>

Flight(const string&, const Datetime&, const Time&, Airport&, Airport&, Plane&
const getFlightId(): string
const getDepartureTime(): Datetime
const getDuration(): Time
const getOrigin(): Airport&
const getDestination(): Airport&
const getTickets(): vector<Ticket*>
const getLuggage(): vector<Luggage*>
const getPlane(): Plane&
setDepartureTime(Datetime&): void
setDuration(Time&): void
setOrigin(Airport&): void
setDestination(Airport&): void
addLuggage(Luggage&): void
const str(): string
addTicket(Ticket&): bool
removeTicket(const Ticket&): bool
removeFirstTicket(const function<bool(const Ticket&)>&): bool
clearTickets(): void
operator<<(ostream&, const Flight&): ostream&

Diagrama de Classes

Service
type: ServiceType datetime: Datetime worker: string plane: Plane&
Service(const ServiceType&, const Datetime&, const string&, Plane&) const getType(): ServiceType const getDatetime(): Datetime const getWorker(): string const getPlane(): Plane& operator<<(ostream&, const Service&): ostream&

{ ServiceType é do tipo unsigned int e pode ter os valores 0 ou 1 }

Plane
license_place: string type: string capacity: unsigned int flights: list<Flight*> scheduled_services: queue<Service*> finished_services: vector<Service*>
Plane(const string&, const string&, const unsigned int) const getLicensePlate(): string const getType(): string const getCapacity(): unsigned int const getFlights(): list<Flight*> const getScheduledServices(): queue<Service*> const getFinishedServices(): vector<Service*> setType(const string&): void setCapacity(const unsigned int&): void const str(): string addFlight(Flight&): void removeFlight(const Flight&): bool removeFirstFlight(const function<bool(const Flight&)>&): bool removeAllFlights(const function<bool(const Flight&)>&): bool scheduleService(Service&): void completeService(): bool operator<<(ostream&, const Plane&): ostream&

Diagrama de Classes

MenuBlock
options: vector<MenuOption>
addOption(const string&, const function<void()>&): void const getOptions(): vector<MenuOption> const&

Menu
title: string blocks: list<MenuBlock> special_block: MenuBlock const printOptions(): void const getSelectedOption(): MenuOption const&
Menu(const string&) addBlock(const MenuBlock&): void setSpecialBlock(const MenuBlock&): void const show(): void const show(const string&): void

{ MenuOption é do tipo pair<string, function<void()>> }

Diagrama de Classes

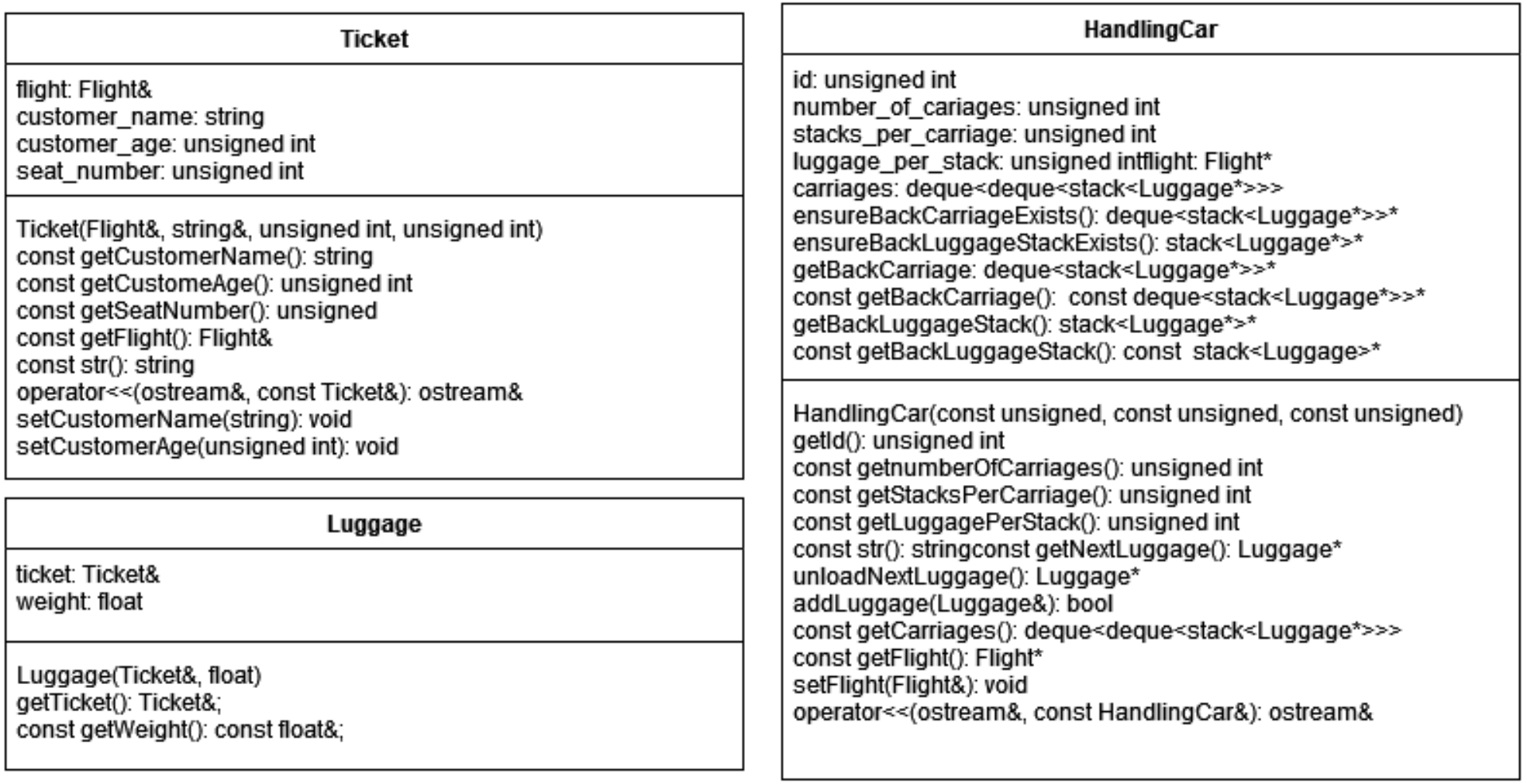
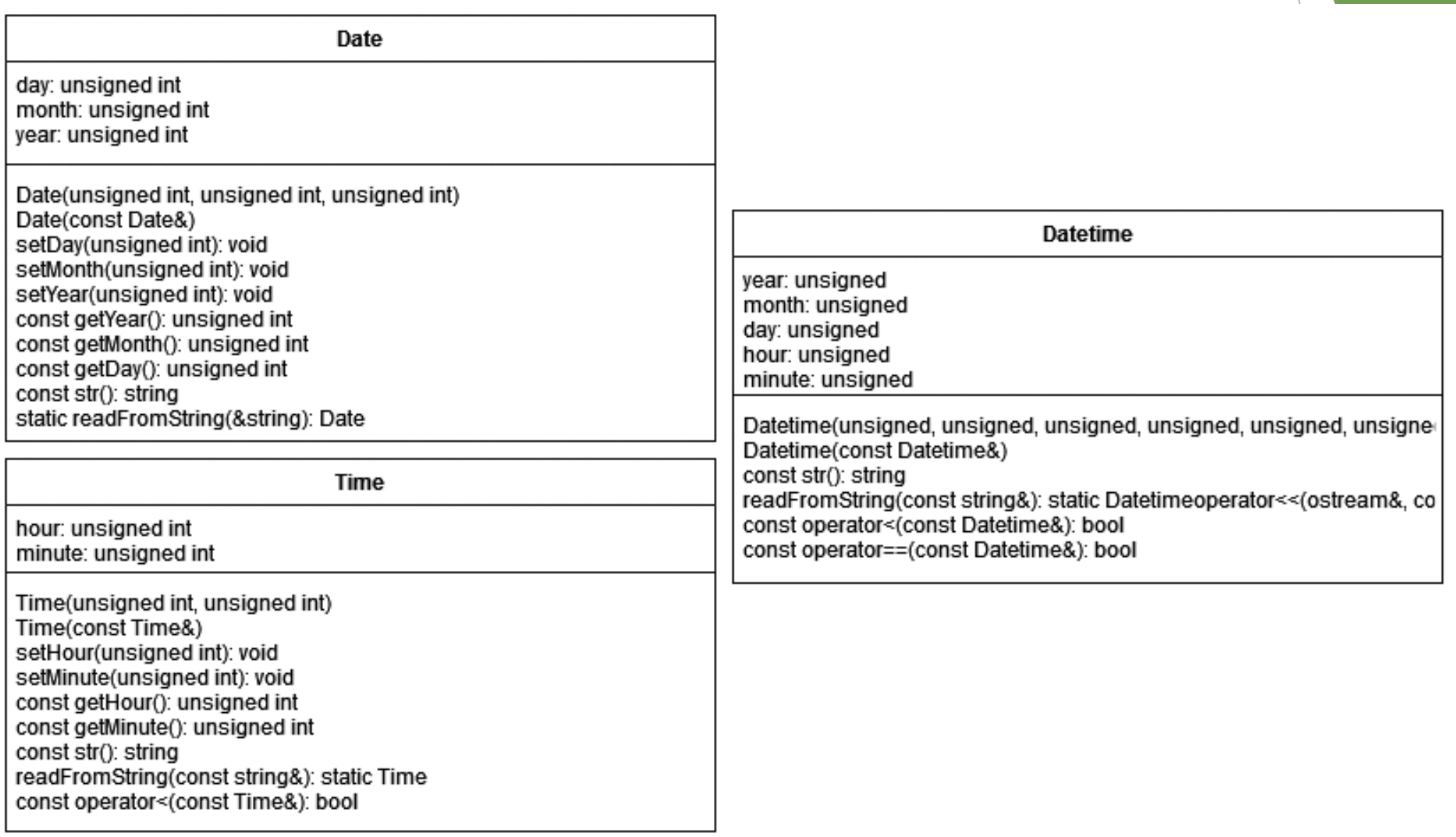


Diagrama de Classes



Estrutura de Ficheiros

- Para cada estrutura de dados guardada na primeira linha guardamos o número de elementos da mesma para, posteriormente, conseguirmos iterar por cada um deles (que se encontram em linhas separadas)
- As estruturas de dados guardadas representam:
  - Os aviões, voos, carrinhos de transporte de bagagem e aeroportos
  - Note-se que, nos casos em que estas contêm outras estruturas de dados, o procedimento é o mesmo, a primeira linha é reservada para o número de objetos que esta contém, e as restantes para os outro atributos da classe

Funcionalidades Implementadas

- Operações CRUD:
  - Para cada classe, damos a opção de criar, ler, atualizar e eliminar os dados dos objetos por via de um menu. Oferecemos também a possibilidade de obter um dado elemento ou elementos a partir da escolha do utilizador, como, por exemplo, escolher ler ou eliminar todos os voos com o mesmo número de serviços. O utilizador pode, também, ordenar os objetos pelo parâmetro que desejar.
- Leitura e escrita de ficheiros
  - Para a leitura de ficheiros, foi criado um namespace files, onde estão implementadas todas as funções de ler e escrever ficheiros.

Destaque de funcionalidade

- Apesar de o código estar mal estruturado, consideramos que a funcionalidade melhor implementada no nosso código são os filtros, que, de uma forma simples e intuitiva, permitem ao utilizador escolher como filtrar os objetos das classes através do menu, de forma completamente personalizada.

Dificuldades

- Organizar e estruturar o código no tempo que tivemos
- Suportar as diferentes operações de CRUD para todas as classes
- Implementar os filtros de escolha por parte do utilizador

- Consideramos que todos os elementos do grupo contribuíram igualmente para a realização do trabalho.

Exemplo de execução

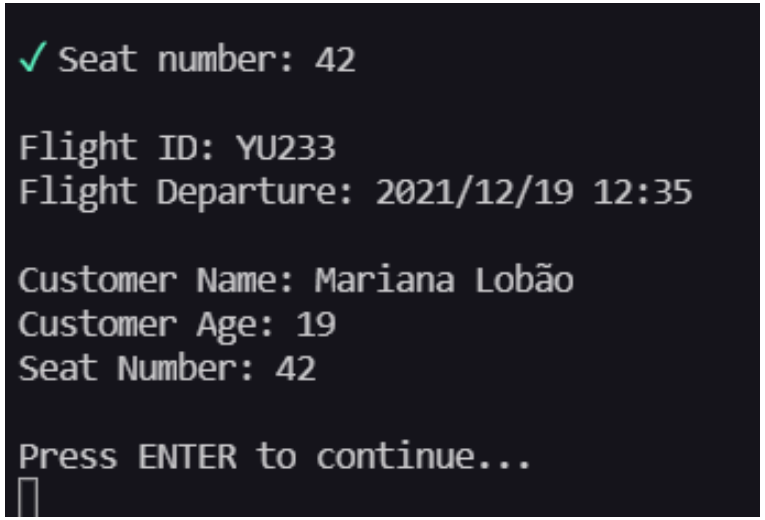


Figura 1 – Exemplo de output de um ticket filtrado pelo utilizador

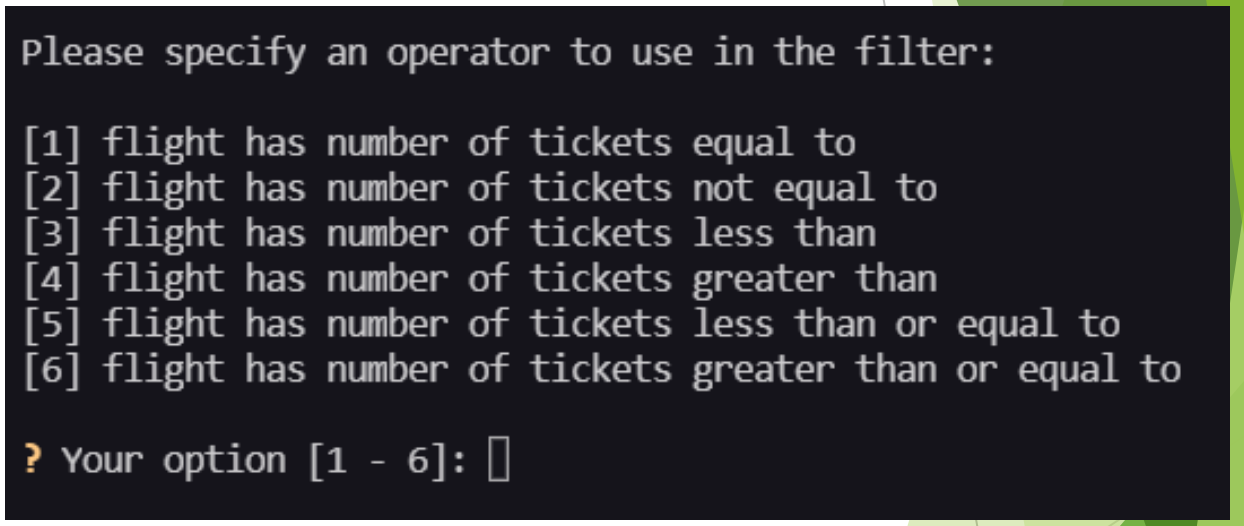


Figura 2 – Exemplo de filtro a partir do menu

Exemplo de execução

```
Please select an area you want to manage!

[1] Planes
[2] Flights
[3] Airports
[4] Handling Cars

[5] Exit

? Your option [1 - 5]:
```

Figura 3 – Menu em execução

```
Select one of the following operations:

[1] Create a new airport
[2] Update airport

[3] Read one airport
[4] Read all airports
[5] Read all airports with filters and sort

[6] Delete one airport
[7] Delete all airports
[8] Delete all airports with filters and sort

[9] Go back

? Your option [1 - 9]:
```

Figura 4 – Menu de operações CRUD em aeroportos

Exemplo de execução

```
Please specify a value to use as a filter:

[1] license plate
[2] type
[3] capacity
[4] number of flights
[5] number of scheduled services
[6] number of finished services

[7] all flights have
[8] any flights have

[9] all finished services have
[10] any finished services have
[11] the next scheduled service has

[12] not
[13] or
[14] and

? Your option [1 - 14]:
```

Figura 5 – Menu para escolha de filtros de aviões

```
Please specify an operator to use in the filter:

[1] (license plate greater than C) and (any flights have number of tickets equal to
[2] (license plate greater than C) and (any flights have number of tickets not equal to
[3] (license plate greater than C) and (any flights have number of tickets less than
[4] (license plate greater than C) and (any flights have number of tickets greater than
[5] (license plate greater than C) and (any flights have number of tickets less than or equal to
[6] (license plate greater than C) and (any flights have number of tickets greater than or equal to

✓ Your option [1 - 6]: 4

? (license plate greater than C) and (any flights have number of tickets greater than 2
```

Figura 6 – Construção de filtros para filtrar aviões

```
Please select an attribute to sort the selected planes:

[1] License Plate
[2] Type
[3] Capacity
[4] Number of flights
[5] Number of scheduled services
[6] Number of finished services

? Your option [1 - 6]:
```

Figura 7 – Menu para ordenação de aviões