Gestão de informação em uma companhia aérea

Faculdade de Engenharia da Universidade do Porto

Algoritmos e Estruturas de Dados 2021/22 **L.EIC – 2ºano**

Trabalho prático 1

Trabalho realizado por: Pedro Macedo (up202007531), Pedro Balazeiro (up202005097) e Rúben Viana (up202005108) - Grupo 11

Descrição do problema

Elaboração de um sistema personalizado e inovador que consiga guardar e gerir informações relativas a uma nova companhia aérea, que acabou de surgir no mercado. O principal objetivo é atender às suas necessidades principais e torná-la funcional e produtiva.

Descrição da solução

Criação de um sistema com a ajuda de um IDE, o CLion. Basicamente, criou-se um conjunto de classes que interligadas pelas suas relações pertinentes permitem gerir e guardar tudo o que é essencial na base de dados da companhia. Tal processo ajuda a companhia na sua gestão e permite aos seus clientes terem a seu dispor o que for necessário dentro do que a própria companhia lhes possa disponibilizar.

Identificação de algoritmos relevantes

Neste sistema utilizaram-se alguns algoritmos:

Algoritmo de pesquisa em vetores: Binary Search

```
int BinarySearch(vector<Flight> v, int el){
    int left = 0, right = v.size() - 1;
    while (left <= right)
    {
        int middle = (left + right) / 2;
        if (v[middle].getFlightNumber() < el) {
            left = middle + 1;
        }
        else if (el < v[middle].getFlightNumber()) {
            right = middle -1;
        }
        else{
            return middle;
        } // found
    }
    return 0;
}
```

Algoritmo de ordenação em vetores: Quick Sort

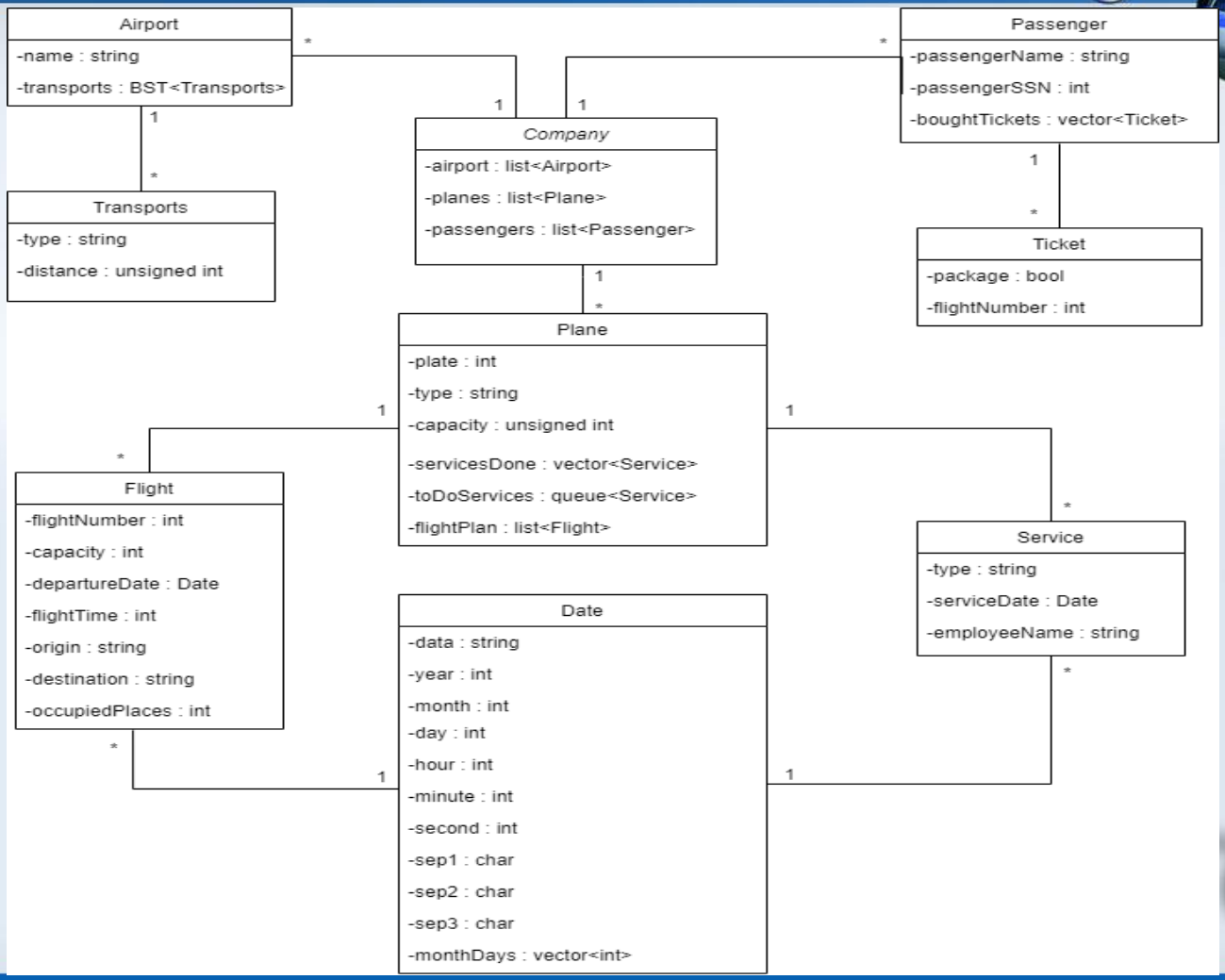
```
void sort(vector<Flight> &v, int low, int high)
{
    if (low < high)
    {
        //partition the array
        int pivot = partition( &: v, low, high);

        //sort the sub arrays independently
        sort( &: v, low , high: pivot - 1);
        sort( &: v, low: pivot + 1, high);
    }
}
```

```
// partition the array using last element as pivot
int partition (vector<Flight> &v, int low, int high)
```

```
void swap(Flight* a, Flight* b)
```

Diagrama de classes



Estrutura de ficheiros

AIRPORT.txt:

- 1ª linha - número de aeroportos;
- 2ª linha - Aeroporto - número de transportes do aeroporto;
- Seguintes linhas - cada uma com um transporte do aeroporto e a sua distância em km;
- Depois de acabarem os transportes desse aeroporto a seguinte linha tem um novo aeroporto;
- Repete-se o mesmo processo feito anteriormente até atingir-se o número de aeroportos indicado inicialmente.

PASSENGERS.txt

- 1ª linha - número de clientes registados;
- Linhas seguintes - Nome do cliente - SSN - e número de bilhetes do cliente;
- Bilhetes comprados (um bilhete por linha);
- Este processo repete-se até o número de clientes registados indicado inicialmente acabar (é de notar que fica um cliente por linha).

PLANES.txt

- 1ª linha - número de aviões;
- 2ª linha - Placa do avião - tipo - capacidade;
- 3ª linha - número de voos;
- Linhas seguintes contêm os respetivos voos e suas caraterísticas, estando um voo por linha;
- Depois de acabarem os voos, a linha seguinte contêm o número de serviços feitos;
- Linhas posteriores contêm os respetivos serviços feitos e suas caraterísticas, estando um serviço por linha, até acabar o número de serviços feitos;
- Linha seguinte indica o número de serviços por fazer;
- Linhas posteriores contêm os respetivos serviços por fazer e suas caraterísticas, estando um serviço por linha, até acabar o número de serviços por fazer;
- Repete-se o mesmo processo feito anteriormente a partir da 2ª linha até atingir-se o número de aviões indicado inicialmente.

Lista de funcionalidades implementadas

Create Flight:

```
void Company::addFlight() {
```

Create Service:

```
void Company::addService() {
```

Create Ticket:

```
void Company::buyTicket(Passenger &p) {
```

Create Plane:

```
void Company::addPlane() {
```

Create Passenger:

```
void Company::addPassenger() {
```

Create Transport:

```
void addTransport(string typ, unsigned int dis);
```

Delete Flight:

```
void Company::removeFlight() {
```

Delete Service:

```
void Company::removeService() {
```

Delete Passenger:

```
void Company::removePassenger() {
```

Delete Plane:

```
void Company::removePlane() {
```

Delete Transport:

```
    void removeTransport(string typ, unsigned int d);
```

Show Transport:

```
    void showTransports() const;
```

Show Flights:

```
void Company::showAllFlights() {
```

Show Services:

```
void Company::showAllServices() {
```

Show Passengers:

```
void Company::showAllPassengers() {
```

Show Planes:

```
void Company::showAllPlanes() {
```

Lista de funcionalidades implementadas

Vector Flights available to check-in:

```
vector<Flight> Company::getFlightsToCheckIn() const {
```

Check-in:

```
void Company::checkIn(Passenger &p) {
```

Record Airports, Passengers and Planes:

```
void Company::record(ofstream &dataPl, ofstream &dataPs, ofstream &dataAir) {
```

Main menu:

```
void Company::mainMenu(){
```

Settings menu:

```
void Company::settingsMenu(){
```

User menu:

```
void Company::userMenu() {
```

Check if passenger registered in data base:

```
bool Company::checkPassenger(Passenger &p){
```

Read Airports, Passengers and Planes:

```
Company::Company(ifstream &dataPl, ifstream &dataPs, ifstream &dataAir) {
```

Update Flights and Services:

```
void Company::update() {
```

Helper to cliente (continue when he/she wants):

```
void waitEnter(){
```

Check if Plane is available to fly:

```
bool checkIfIsAvailable(Date maintenanceDay, Date wantedDay);
```

Check if Ticket exist when check-in:

```
bool ticketExist(int numFlight);
```

Available places:

```
unsigned int getAvailablePlaces();
```

Check difference between days:

```
int daysBetweenDates(Date &date2);
```

Lista de funcionalidades implementadas

List Filghts:

```
list<Flight> &getFlights();
```

Queue Services not done:

```
queue<Service> &getToDoServ();
```

Vector Services done:

```
vector<Service> &getDoneServ();
```

Vector Tickects bought by Passenger:

```
vector<Ticket> boughtTickets;
```

List Airports:

```
list<Airport> airports;
```

List Planes:

```
list<Plane> planes;
```

List Passengers:

```
list<Passenger> passengers;
```

Actual Date:

```
 int daysSince2020() const;
```

Binary Search:

```
int BinarySearch(vector<Flight> v, int el){
```

Quick sort:

```
void sort(vector<Flight> &v, int low, int high)
```

Destaque de funcionalidade

Check-in:

Esta funcionalidade permite ao cliente, tendo comprado bilhetes, saber se tem algum voo no qual possa realizar o check-in, apresentando o número do voo caso exista um voo correspondente. Caso contrário dá a informação que não existe nenhum voo no qual possa fazer check-in.

Na possibilidade de fazer check-in o cliente é questionado se o quer fazer ou cancelar. No primeiro caso, depois de confirmado o check-in, informação sobre o destino é demonstrada para ajudar o cliente aquando da sua chegada no aeroporto final.

Principais dificuldades encontradas

Na visão do grupo, encontrou-se um enunciado ambíguo o que criou algumas dificuldades, visto que não era claro do que se pretendia em certas situações fazendo com que a liberdade fosse e não fosse restringida ao mesmo tempo. Além disso, com esta confusão o tempo foi se tornando um pouco reduzido pois era necessário umas horas extra para tentar cumprir dentro dos possíveis o que o trabalho exigia, mesmo muitas vezes certas situações não fazerem muito sentido.