trabalho AED:
Gestão de informação em uma companhia aérea

Grupo 47

João Pinheiro up202008133
José Araújo up202007921
José Ribeiro up202007231

O problema proposto:

“Uma nova companhia aérea acabou de entrar no mercado e pretende implementar um sistema de gestão de informação inovador e personalizado às suas necessidades.

O sistema deve guardar e gerir informação relativa a aviões, voos, passageiros e bagagens.”

Descrição da solução

Algoritmos utilizados (dois exemplos):

Quick sort:

```
void insertionSort(vector<Flight> &f, unsigned left, unsigned right){
    for(unsigned p=left+1; p<right; p++){
        Flight tmp = f[p];
        unsigned j;
        for( j=p; j>0 && tmp<f[j-1]; j--)
            f[j] = f[j-1];
        f[j] = tmp;
    }
}


//Quick Sort
void Airplane::sortFLights(vector<Flight> &f, unsigned left, unsigned right) {
    if(right-left<=10)
        insertionSort(f, left, right);
    else {
        Flight x = median(f, left, right);
        int i = (int) left;
        int j = (int) right - 1;
        for (;;) {
            while (f[++i] < x);
            while (x < f[--j]);
            if (i < j)
                swap( & f[i],  & f[j]);
            else break;
        }
        swap( & f[i],  & f[right - 1]);
        sortFLights(f, left,  right: i - 1);
        sortFLights(f,  left: i + 1, right);
    }
}
```

Merge sort:

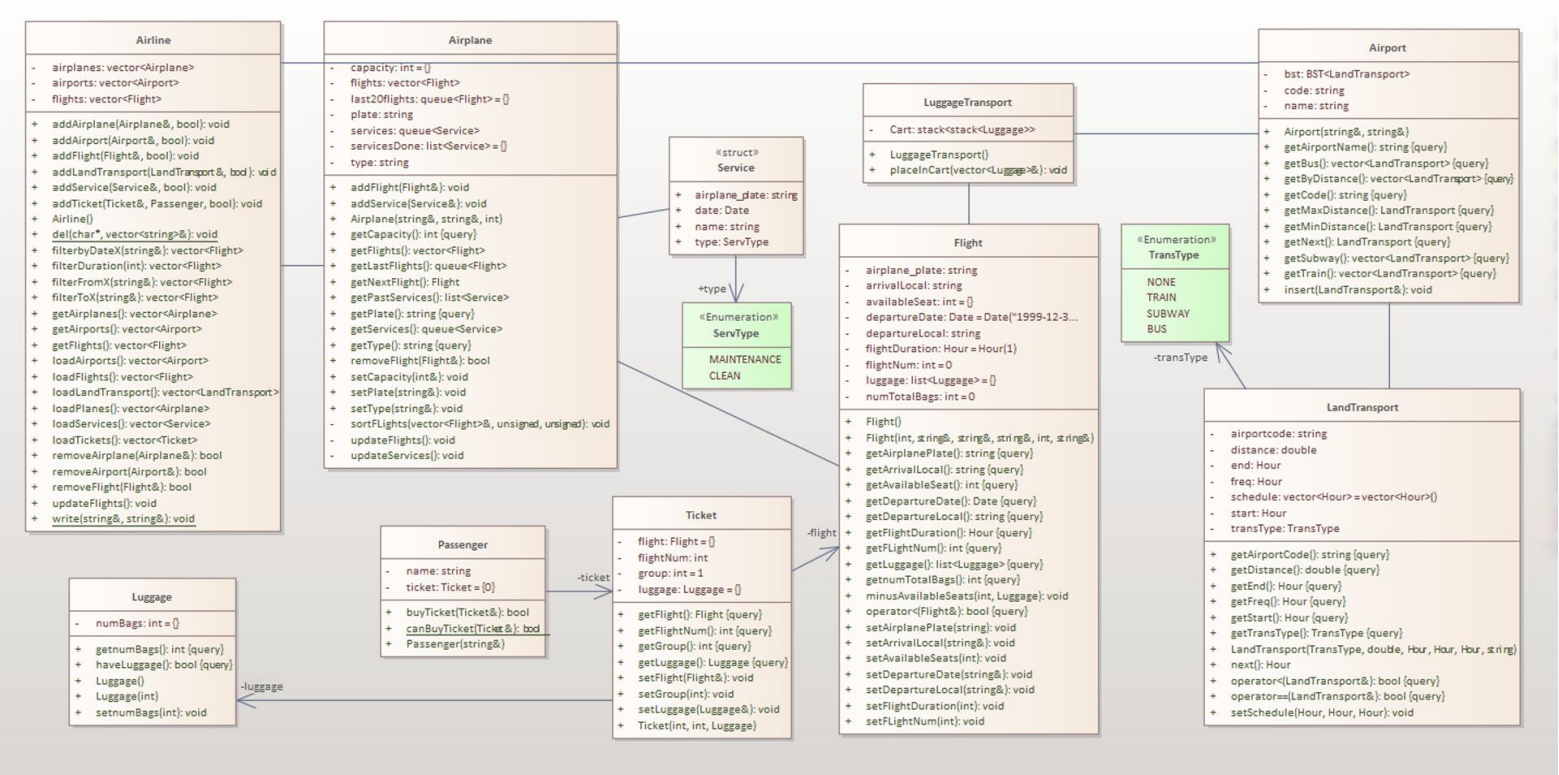
```
// Merge Sort
bool operator<= (const Flight& lhs, const Flight& rhs){
    if(lhs.getDepartureLocal()==rhs.getDepartureLocal()) return lhs.getArrivalLocal()<=rhs.getArrivalLocal();
    return lhs.getDepartureLocal()<rhs.getDepartureLocal();
}

void merge(vector<Flight> &v, vector<Flight> &tmpArr, int leftPos, int rightPos, int rightEnd) {
    int leftEnd = rightPos - 1, tmpPos = leftPos;
    int numElements = rightEnd - leftPos + 1;
    while ( leftPos <= leftEnd && rightPos <= rightEnd )
        if ( v[leftPos] <= v[rightPos] )
            tmpArr[tmpPos++] = v[leftPos++];
        else
            tmpArr[tmpPos++] = v[rightPos++];
    while ( leftPos <= leftEnd )
        tmpArr[tmpPos++] = v[leftPos++];
    while ( rightPos <= rightEnd )
        tmpArr[tmpPos++] = v[rightPos++];
    for ( int i = 0; i < numElements; i++, rightEnd-- )
        v[rightEnd] = tmpArr[rightEnd];
}

void mergeSort(vector<Flight> &v, vector<Flight> &tmpArr, int left, int right){
    if (left < right){
        int center = (left + right) / 2;
        mergeSort( & v,  & tmpArr, left, center);
        mergeSort( & v,  & tmpArr,  left: center + 1, right);
        merge( & v,  & tmpArr, left,  rightPos: center +1, right);
    }
}

void mergeSort(vector<Flight>& v) {
    vector<Flight> tmpArr(v.size());
    mergeSort( & v,  & tmpArr,  left: 0,  right: (int)v.size()-1);
}
```

Diagrama de classes



Estrutura de ficheiros

 Airline:
Classes:
Airline

- Airport:
  -LandTransport
  -LuggageTransport

- Airplane:
  -Service
  -Flight:
  Passenger:
  -Luggage
  -Ticket

- Date

Populate:

Airplane, Airport, Flight, Land Transport, Service e Ticket (.txt).

 Tests: Airplane, Airline, Airport, Luggage, Passenger e Ticket tests (.cpp).

 Outros (dentro da pasta do projeto): App.cpp, App.h (interface do menu) e main.cpp.

Lista de funcionalidades: 1.CRUD

Create:

```
void Airline::addAirplane(Airplane& airplane, bool write) {
    for(auto &f: airplane.getFlights()) flights.push_back(f);
    updateFlights();
    airplanes.push_back(airplane);
    if(write) {
        string content = airplane.getPlate() + " " + airplane.getType() + " " +
                         to_string(airplane.getCapacity());
        Airline::write( file: "../Populate/Airplane.txt", content);
    }
}
```

Read:

```
vector<Airplane> Airline::loadPlanes() {
    string namePlane, typePlane, platePlane, objPlane, name ;
    vector<Airplane> result;

    ifstream file_airplanes;
    file_airplanes.open( s: "../Populate/Airplane.txt");

    while(getline( &: file_airplanes,  &: namePlane)) {
        getline( &: file_airplanes,  &: typePlane);
        getline( &: file_airplanes,  &: platePlane);

        class Airplane plane(namePlane, typePlane,  capacity: stoi(platePlane));

        result.push_back(plane);
        this->addAirplane( &: plane,  write: false);
    }
    file_airplanes.close();
    return result;
}
```

Update & Delete:

```
void Airline::write(const string& file, const string& content) {
    ofstream file_generic( s: file,  mode: ios::app);

    if(file_generic.fail()) return;
    vector<string> strings;
    string word;
    for(auto i :char : content){
        if(i==' ') {
            strings.push_back(word);
            word="";
        }
        else word+=i;
    }
    strings.push_back(word);
    for(const auto& i :const string & :strings) file_generic << i << endl;
    file_generic.close();
}


void Airline::del(const char* file1, const vector<string>& del) {
    ifstream file( s: file1);
    ofstream o( s: "../Populate/temp.txt");
    int count=stoi( str: del[1]);
    string line;
    while(getline( &: file,  &: line)){
        if(count==-1) count = stoi( str: del[1]);
        if(del[0]==line) {
            count--;
        }
        if(count == stoi( str: del[1])) o << line << endl;
        else count--;
    }

    o.close();
    file.close();
    remove( filename: file1);
    rename( old: "../Populate/temp.txt",  new: file1);
}
```

Lista de funcionalidades: 2. Filtragem (parcial/total)

```
/**
 * Get Airport's LandTransport those that are subways
 * @return subways
 */
vector<LandTransport> getSubway() const;

/**
 * Get Airport's LandTransport those that are trains
 * @return trains
 */
vector<LandTransport> getTrain() const;

/**
 * Get Airport's LandTransport those that are bus
 * @return bus
 */
vector<LandTransport> getBus() const;

/**
 * Get Airport's LandTransport ordered by distance
 * @return LandTransports ordered by distance
 */
vector<LandTransport> getByDistance() const;

/**
 * Get closests transport
 * @return  Closest LandTransport
 */
LandTransport getMinDistance() const {return bst.findMin();};

/**
 * Get furthest transport
 * @return Fursthest LandTransport
 */
LandTransport getMaxDistance() const {return bst.findMax();};
/**
 * Get next transport
 * @return Next LandTransport to pass by
 */
LandTransport getNext() const;
```

O exemplo aqui presente de filtragem encontra-se dentro da classe Airport (Airport.h),

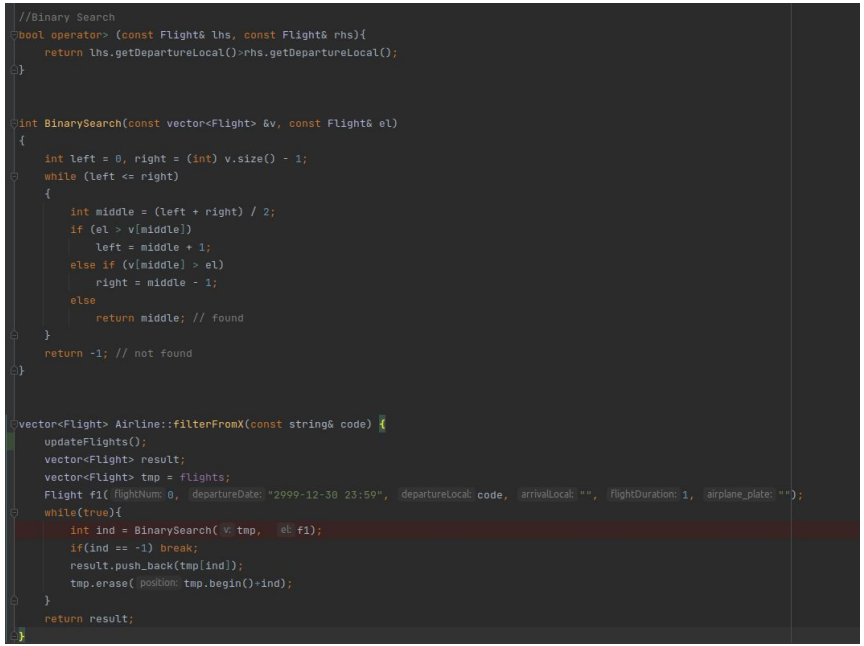
Aqui estão métodos que servem para obter objetos da classe LandTransport diferentes (Train, Subway ou Bus - parcial).

Também se encontram métodos para filtrar esses objetos, de acordo com a menor distância (getByDistance() - total), ou conseguir encontrar aquele que se encontra à maior (getMaxDistance - parcial) ou menor (getMinDistance - parcial) distância.

Lista de funcionalidades: 3. Pesquisa (parcial)

Binary search (pesquisa sequencial):

Para além da tradicional pesquisa sequencial, utilizámos também binary search num vetor previamente ordenado, neste caso em específico para a filtragem de voos provenientes do aeroporto X (filterFromX).



```
//Binary Search
bool operator> (const Flight& lhs, const Flight& rhs){
    return lhs.getDepartureLocal()>rhs.getDepartureLocal();
}


int BinarySearch(const vector<Flight> &v, const Flight& el)
{
    int left = 0, right = (int) v.size() - 1;
    while (left <= right)
    {
        int middle = (left + right) / 2;
        if (el > v[middle])
            left = middle + 1;
        else if (v[middle] > el)
            right = middle - 1;
        else
            return middle; // found
    }
    return -1; // not found
}


vector<Flight> Airline::filterFromX(const string& code) {
    updateFlights();
    vector<Flight> result;
    vector<Flight> tmp = flights;
    Flight f1( flightNum: 0,  departureDate: "2999-12-30 23:59",  departureLocal: code,  arrivalLocal: "",  flightDuration: 1,  airplane_plate: "");
    while(true){
        int ind = BinarySearch( v: tmp,  el: f1);
        if(ind == -1) break;
        result.push_back(tmp[ind]);
        tmp.erase( position: tmp.begin()+ind);
    }
    return result;
}
```

Destaque de funcionalidade:

Obter a próxima passagem de um transporte:

```
Hour LandTransport::next() {
    Date nowd( date: Date::getNow());
    Hour now{};
    now.setHour(nowd.getHour()); now.setMinute(nowd.getMinute());
    Hour min( date: "23:59");
    for(auto i :Hour :schedule){
        if(now<i && i<min){
            min = i;
        }
    }
    if(min.getHour()==23 && min.getMinute()==59) return Hour{};
    return min;
}
```

Em geral não houve nenhuma funcionalidade que se destacasse muito mais do que as outras, no entanto considerámos esta função para obter a próxima passagem de um transporte como uma solução interessante e simples para obter essa mesma passagem, merecendo assim uma nota de destaque.

Principais dificuldades:

1. Ler dados a partir dos ficheiros;

```
vector<Airplane> Airline::loadPlanes() {
    string namePlane, typePlane, platePlane, objPlane, name ;
    vector<Airplane> result;

    ifstream file_airplanes;
    file_airplanes.open( s: "../Populate/Airplane.txt");

    while(getline( &: file_airplanes,  &: namePlane)) {
        getline( &: file_airplanes,  &: typePlane);
        getline( &: file_airplanes,  &: platePlane);

        class Airplane plane(namePlane, typePlane,  capacity: stoi(platePlane));

        result.push_back(plane);
        this->addAirplane( &: plane,  write: false);
    }
    file_airplanes.close();
    return result;
}
```

2. Implementar o menu;

```
What are you?
1) Passenger
2) Worker
0) Exit
Option:
```

Obrigado pela atenção