

**algorimtos e estruturas de dados**

**MIEIC**

**Turma 2 – Grupo D**

Inês Teixeira – [up201404592@fe.up.pt](mailto:up201404592@fe.up.pt)

José Aleixo da Cruz – [up201403526@fe.up.pt](mailto:up201403526@fe.up.pt)

José Miguel da Costa – [up201402717@fe.up.pt](mailto:up201402717@fe.up.pt)

Dezembro 2015

Relatório de Projeto

Empresa de transporte de mercadorias

# Tema de trabalho

O nosso grupo trata do desenvolvimento de um programa em C++, que tem como propósito lidar com a informação de uma empresa de transporte de mercadorias.

No início, desenvolvemos métodos que se debruçavam sobre os clientes, serviços e meios de transporte da empresa, que apelidámos caridosamente de “Transportex”. Desta vez, coube-nos elaborar estruturas de dados que guardassem informação relativa a motoristas, reparações e clientes inativos desta empresa.

No âmbito da cadeira de Algoritmos e Estruturas de Dados (AEDA) foram lecionados três tipos importantes de contentores de dados: árvores binárias, filas de prioridade e tabelas de dispersão. É fazendo uso destes contentores que desenvolvemos a solução para esta segunda parte do projeto.

# Solução implementada

***Binary search tree***

A partir da definição de uma árvore de pesquisa binária (BST), presente no ficheiro *header* disponível no Moodle, desenvolvemos métodos capazes de adicionar, editar, atualizar e remover motoristas da empresa de uma BST.

Na classe “Empresa”, que foi elaborada na primeira parte do trabalho, adicionamos um membro privado denominado “motoristas”, que representa uma árvore binária de uma nova classe “Motorista”, cujo critério de ordenação é o número de horas de trabalho efetuado pelo motorista num determinado dia (ordem crescente).

A classe “Motorista” tem como atributos privados o nome, número de identificação fiscal (NIF) e horas de serviço diário já efetuado do motorista. Desenvolvemos o respetivo construtor e as funções que alteram estes atributos.

Sempre que um motorista da árvore binária é alterado, de forma a manter a ordenação da árvore, removemos os dados desse motorista da árvore, atualizamos os dados e voltamos a inseri-lo.

Temos funções que fazem uma listagem dos motoristas com menos e mais horas de trabalho, assim como listagem de todos os motoristas.

**Fila de prioridade**

Utilizando o contentor *priority\_queue* fornecido pela *standard library* do C++, gerámos uma fila de prioridades com o objetivo de controlar as eventuais reparações que os camiões da empresa necessitam de efetuar, atendendo à disponibilidade das oficinas e ao tipo de serviços (específicos ou casuais) que oferecem. Dessa forma, foram criadas novas classes como "Oficina" e "Lista de Oficinas".

Na classe "Oficina", foi desenvolvido os membros-função privados nome da oficina, disponibilidade da empresa (nº de dias que leva a terminar um serviço) e marcas especializadas. Todos estes atributos podem ser modificados e atualizados.

A empresa recorre a várias oficinas podendo escolher a oficina que tem maior disponibilidade. Desta forma, é criado na classe "ListaOficinas" uma priority\_queue "oficinas", onde é guardado todas as oficinas disponíveis para o conserto dos camiões da empresa.

Há dois tipos de serviços - serviço específico e normal. A função recebe um bool que indica o tipo de serviço que o camiao necessita. Se este tiver valor falso, o serviço é específico e para além de a oficina escolhida ter maior disponiblidade tem de ter como marca especilializada a marca do camião. Se o serviço for normal (valor verdadeiro do bool), então a função retorna apenas a oficina do topo da lista - a de maior disponibilidade.

Quando é associado um camião a uma oficina, a disponibilidade diminuí, ou seja, o número de dias que toma a oficina para arranjar um próximo camião aumenta, de 1 dia. Quando o serviço é finalizado, a disponibilidade da oficina tem de ser atualizada novamente, neste caso, diminuí em 1 dia.

Em qualquer altura uma oficina pode ser acrescentada e removida da priority\_queue, assim, esta lista pode ser sempre atualizada.

**Tabela de dispersão**

Para ajudar a secção de Marketing da Transportex, mantemos a informação relativa a clientes inativos (que não requisitam serviços da empresa há mais de 1 ano) atualizada numa tabela de dispersão, sob a forma de *unordered\_set*.

Na classe "Cliente Inativo" tem como atributos privados o nome do cliente inativo e o seu NIF.

# 3. Casos de utilização

A interface do programa complementa aquela que já tínhamos feito para a primeira parte do projeto com mais 3 menus. Um para atribuir serviços aos motoristas pertences à empresa, outro para enviar camiões para reparações e outro ainda para visualizar e alterar a lista de clientes inativos.

# 4. Dificuldades da realização do projeto

Ao longo do desenvolvimento desta segunda parte do projeto foram encontradas algumas dificuldades.

Em semelhança à primeira parte, a subjetividade do enunciado levou o nosso grupo a ter de refazer algumas partes do código e a reorganizar algumas classes e funções que julgávamos, numa primeira fase, o que era pedido para desenvolver.

No entanto, não foram encontradas mais dificuldades de fazer notar.