

Estruturas de Dados e Algoritmos Projeto prático 2 - 2019/2020

 ***Trabalho realizado por:***

Pedro Rodrigues - 2046619

Diogo Santos - 2042919

João Cupido - 2046519

Michael Costa - 2048919

Paulo Drumond - 2048719

***1º Ano de Engenharia Informática***

Índice

[**Introdução**](#_heading=h.39pr77j8gb98) **3**

[**Implementação**](#_heading=h.2et92p0) **4**

[**main.cpp**](#_heading=h.tyjcwt) **4**

[**Inicio.h**](#_heading=h.164qsgh6dtuo) **5**

[**Inicio.cpp**](#_heading=h.1t3h5sf) **5**

[**iteracao.h**](#_heading=h.9yw16bnuctrk) **6**

[**iteracao.cpp**](#_heading=h.e2f90srengdh) **6**

[**Fila.h**](#_heading=h.l7anmj7u8pkb) **6**

[**Fila.cpp**](#_heading=h.767d577gvyun) **7**

[**Percurso.h**](#_heading=h.17dp8vu) **8**

[**Percurso.cpp**](#_heading=h.isn8zzpgvebh) **8**

[**Conclusão**](#_heading=h.35nkun2) **11**

# 

# 

# 

# Introdução

Neste projeto, nós criámos um programa em C++ através do Visual Studio com o objetivo final de simular o funcionamento da empresa “​ExpressoEDA​”. Esta é uma empresa de transporte de passageiros de ponto a ponto na qual não podem entrar passageiros nas paragens intermédias, apenas sair.

O projeto deverá simular todas as funcionalidades relativas ao funcionamento desta empresa, de uma forma geral a entrada de passageiros na origem e a respetiva saída de passageiros nos seus destinos.

Tivemos contudo certas restrições relativas à construção do projeto. Foi-nos proibido o uso da classe/biblioteca *vector*, e pedido o uso dos tipos de dados definidos pelos alunos, structs, listas ligadas e árvores de pesquisa binária.

É também importante mencionar que cada viagem é levada a cabo por um autocarro com uma capacidade de passageiros variável entre 5 e 10 inclusive, um motorista, e uma matrícula. O motorista é identificado pelo seu primeiro e último nome. A matrícula é composta por um valor hexadecimal aleatório de 4 dígitos. Cada autocarro encontra-se associado a uma lista de passageiros, todos os autocarros iniciam a sua viagem com todas as vagas preenchidas passando por um número variável de paragens entre 4 e 9 inclusive.

# 

# Implementação

## main.cpp

O ficheiro main.cpp é constituído só por uma função, que é a função *main*, que tem como objetivo unir os restantes itens e controlá-los de acordo com as indicações e os desejos do utilizador.

O primeiro bloco de comandos da função *main* começa por definir uma semente que será usada para gerar valores aleatórios através do *srand((unsigned)time(NULL))*.   
  
O segundo bloco de comando define 3 listas que armazenam, respetivamente, os primeiros nomes dos passageiros e condutores, os últimos nomes dos mesmos e as paragens existentes no percurso. Todos os nomes estão armazenados em ficheiros separados. Por isso, definimos em seguida as funções *lerFicheiroPrimeiroNome*, *lerFicheiroUltimoNome* e *lerFicheiroParagens*, que têm como função ler os nomes armazenados nos ficheiros e guardá-los na lista correspondente (*listaPrimeironome*, *listaUltimonome* e *listaParagens*, respetivamente).  
  
O terceiro bloco de comandos define uma lista ligada denominada *Fila* que se encontra vazia inicialmente, até à chamada da função *inicializaFila*. Esta função recorre a 2 outras funções (*Criapessoa* e *entraFila*)que têm como principal objetivo criar uma nova pessoa, definida pelo seu bilhete de autocarro, pelo seu primeiro nome e pelo seu último nome e, em seguida, ocupar a última posição da lista ligada *Fila*. Esta fila representa a fila de espera para a entrada no autocarro.

O quarto bloco de comandos começa por chamar a *função inicializaFila* que tem como principal objetivo criar a fila de espera para o autocarro (representada por uma estrutura ligada) através de outras duas funções, a função *criaPessoa*, criando assim uma nova pessoa que será posteriormente adicionada pela função *entraFila* à fila de espera. Em seguida, há uma criação da árvore binária de bilhetes em que os elementos são os bilhetes que já foram usados pelas pessoas e pelos passageiros, com o objetivo de estar numa função em que não há dois ou mais pessoas com o mesmo número de bilhete. Finalmente, a criação da árvore binária de bilhetes usados não começa com NULL, mas sim com um valor negativo. Como o número de bilhete é sempre positivo, criámos uma sub-árvore direita chamada *BilhetesUsados* e a árvore original é denominado por *aux*. Logo, removemos o valor negativo , eliminando a árvore de bilhetes usados original (aux).

O quinto bloco de comando tem como objetivo inicializar um percurso que será definido de *pe* e, terá como seu primeiro elemento a primeira paragem. Em seguida temos a chamada das funções *criaPercurso* e *imprimirPercurso*,que servem para criar o percurso com localidades escolhidas de modo aleatório e, depois, imprimir o percurso.

Os restantes blocos de comandos define uma variável (*escolha\_do\_utilizador*), que servirá para armazenar um char escolhido pelo utilizador. Logo após a definição da variável, o programa pede ao utilizador que insira o valor da variável *escolha\_do\_utilizador*.

Caso a variável *escolha\_do\_utilizador* seja diferente de ‘o’ e diferente de ‘s’, o programa fecha. Caso a variável *escolha\_do\_utilizador* seja ‘o’, o programa irá imprimir as operações disponíveis para o utilizador. Caso a mesma seja ‘s’, o programa vai executar a iteração. A cada iteração o programa; movimenta o autocarro para a paragem seguinte, remove o autocarro que partiu da última paragem, cria 15 novos passageiros e coloca-os na fila de espera, remove um conjunto aleatório de passageiros de cada autocarro, coloca os números dos bilhetes de cada passageiro na árvore de cada paragem e volta a apresentar o estado do programa.

Por fim, o programa utiliza um comando while para que o programa seja corrido mais que uma vez caso a opção escolhida seja diferente de ‘o’ ou de ‘s’.

## Inicio.h

Neste ficheiro *header* está presente cinco protótipos de funções. Nomeadamente, *Pessoa* *criaPessoa*, *inicializaFila*, *lerFicheiroPrimeiroNome*, *lerFicheiroUltimoNome* e *lerFicheiroParagens*.

Neste header também está definido duas constantes; NUM\_P\_NOMES 44 e NUM\_U\_NOMES 97, que representam a quantidade dos nomes e sobrenomes presentes nos ficheiros.

## Inicio.cpp

A primeira função neste ficheiro é o *Pessoa criaPessoa*, que recebe 2 arrays de strings com os seus tamanhos definidos pelas constantes declaradas anteriormente. Esta também recebe um apontador para os *bilhetesUsados*.

Após terem sido colocados os nomes e sobrenomes nos devidos arrays, estes são selecionados aleatoriamente e depois são armazenados na estrutura Pessoa. Os bilhetes são também números gerados de forma aleatória (entre 1000 e 10999). Depois, estes são comparados com os números já existentes. Se não for único, ele vai entrar no loop while e criar um novo número de bilhete até ela ser única, isto é, se não for encontrada na árvore de bilhetesUsados. Finalmente, o novo número de bilhete vai ser inserido na árvore de bilhetesUsados e vai ser inserido na estrutura novaPessoa.

Na função *inicializaFila*, esta recebe como argumento um apontador para o primeiro elemento da fila, os arrays dinâmicos listaPnomes e listaUnomes e um apontador para bilhetesUsados. É usado um ciclo for, em que um inteiro i, que inicializa no 1, incrementa até ao valor 30 para que seja possível a criação da lista de espera. Em cada incrementação é “chamado” primeira função, que cria as pessoas e coloca-as na fila de espera.

As três últimas funções são respetivamente void lerFicheiroPrimeiroNome, void lerFicheiroUltimoNome e void lerFicheiroParagens, que recebem arrays dinâmicos vazios e estas funções leem os ficheiros .txt e fazem a conversão para os devidos arrays dinâmicos.

## iteracao.h

Neste ficheiro header foi implementado seis funções protótipos o *Item\* menuOpcoes*, *iteraFila*, *autocarro cria\_autocarro*, *Item\* inserir\_autocarro*, *bool verifica\_saida\_passageiro* e por fim o *saida\_passageiros*.

## iteracao.cpp

O *Item\* menuOpcoes* recebe um percurso pe e um apontador para uma fila. Primeiramente, é declarado um inteiro num\_opcao que vai possibilitar ao utilizador escolher de entre 5 opções, nomeadamente, o remover passageiros nos autocarros, remover pessoas em fila de espera, apresentar bilhetes numa paragem,alterar o motorista e por fim remover bilhete da paragem.

Para a primeira opção é executado a função *RemoverPessoaAutocarro*, que recebe como argumento o início do percurso. Para a segunda opção é utilizado *RemoverPessoaFiladeEspera*, esta função recebe como argumento uma fila. Para a terceira opção é usado uma outra função, *apresentarBilhetes* que tem como argumento um percurso, na quarta opção é usado o *alterarMotorista*, e na última opção é utilizado o *RemoverBilheteParagem*.

Na função *iterafila* os argumentos são um apontador para o primeiro elemento de um item, dois apontadores para arrays de strings (primeiro e último nome), um inteiro e um apontador para bilhete. Aqui é usado um ciclo for que irá adicionar pessoas a fila de espera.

A função *autocarro cria\_autocarro*, tal como o nome indica, cria um novo autocarro. Recebe dois array de strings para o primeiro e último nome, dos quais são gerados aleatoriamente o nome e sobrenome do motorista. É criada uma matrícula com valores hexadecimais aleatório de quatro caracteres. A capacidade do autocarro é também criada como um número aleatório entre 5 e 10.

Na função *Item\* inserir\_autocarro* é criado um autocarro novo. Se houver autocarros, serão todos passados para a próxima paragem. Quando a primeira paragem estiver sem autocarro, é inserido esse novo autocarro, retirando também o número correto de passageiros da fila.

Na função *bool verifica\_saida\_passageiro*, é gerado um número de 1 a 4, de modo que se sair o 1 (que representa 25% de chance) o passageiro sai do autocarro.

Na última função (*saida\_passageiro*) é percorrido cada passageiro e executado ao mesmo tempo a função anterior que verifica se este sai ou permanece no autocarro.

## Fila.h

Neste item temos a definição de duas estruturas denominadas Pessoa e Item. A primeira estrutura (estrutura Pessoa) armazena o primeiro e último nome de cada passageiro, assim como o seu número de bilhete, enquanto que a segunda (estrutura Item) define um apontador para o seguinte, inicializado em NULL, e pela sua informação pessoal armazenada na estrutura Pessoa. Ainda dentro deste ficheiro temos a chamada de três funções: *printItem*, *entraFila* e *imprimeFila* que serão explicadas mais à frente. Por fim, temos a chamada de três funções; *removerInicio*, *removerFim* e *removerMeio*, que irão remover um elemento do array relacionado à estrutura Item de acordo com a posição do elemento.

## Fila.cpp

A primeira função deste item, denominada *printItem*, imprime um nome (name) e ocupa o restante espaço (tam) com espaços vazios. Ao chamar esta função, o tam terá obviamente de ser sempre superior ao tamanho da string.

Em seguida, a função *imprimeFila* cria um apontador temporário para o primeiro elemento da fila e o tamanho que vai ser igual ao tamanho da fila (com a ajuda da função *ContaFila*). Também vai ser criado dois arrays (arrayUnomes e arrayNumBilhete) onde vão ser guardadas os últimos nomes das pessoas e os seus números de bilhetes. Finalmente, vai ser imprimido a fila de espera com a ajuda da função *printItem* e da função *ContaFila*.

A função entraFila insere uma nova pessoa no fim da fila de espera representada através de um array, caso a fila se encontre vazia, uma nova fila deve ser criada, na qual a nova pessoa será obviamente a primeira pessoa na sua fila.  
  
A função ContaFila tem como objetivo contar o número de pessoas presentes na fila, através de um comando do e while, a vaiável conta deverá ser atualizada em 1 valor por cada elemento no array que representa a fila de espera.

A função removerInicio recebe a fila de espera. É criado um apontador temporário para o segundo elemento da fila de espera, é removido a fila original e retorna o apontador temporário (que é uma fila de espera em que o primeiro elemento da fila original foi removido).

A função removerFim recebe a fila de espera. É criado um apontador temporário para o primeiro elemento da fila de espera e o apontador vai passar de elemento a elemento da fila até chegar a apontar ao penúltimo elemento da fila. Aí, o apontador temporário para o seguinte vai ser removido e alterado para NULL. Por fim, vai retornar a fila de espera alterada.

A função removerMeio recebe a fila de espera e o número de bilhete a remover. É criado um apontador temporário para o primeiro elemento da fila de espera e vai percorrer de elemento a elemento da fila e verificar se o número de bilhete da pessoa seguinte do apontador temporário é igual ao número de bilhete a remover. Se o número de bilhete a remover não está na fila de espera, a fila não vai ser alterada. Se o número de bilhete a remover está na fila de espera, o apontador temporário para o seguinte (em que aponta para a pessoa que tem o número de bilhete a remover) vai ser igual ao apontador temporário para o seguinte do seguinte, isto é, vai saltar para o elemento seguinte. Assim, a pessoa que tem o número de bilhete a remover vai ser removido da fila de espera. Finalmente, esta função retorna a fila de espera alterada.

## Percurso.h

Neste ficheiro header, foi definido uma estruturaautocarro que vai receber um apontador para o primeiro elemento da lista ligada, um int matricula , um int capacidade, um string para o primeiro nome e um string para o último nome.

Também está definido uma struct bilhete que cria uma árvore de procura binária vazia, que recebe um int num e os apontadores esquerda e direita inicializados como NULL.

É definido posteriormente uma struct percurso que contém outra struct paragem. A struct paragem recebe um apontador para o início da árvore binária, uma string nome, um apontador para a paragem seguinte (que é definido como NULL), um int index e por fim cria-se um autocarro. De seguida existe um int numParagensTotal e um apontador para o início da paragem.

Existe, de seguida, 18 funções protótipos, nomeadamente: *bilhete\* novoNodoArvoreBilhetes*, *bilhete\* insereArvoreBilhetes*, *bool procuraArvoreBilhetes*, *bilhete\* removerArvoreBilhetes* (esta função é uma função de procura), *bilhete\* removerArvoreBilhetes* (esta função é uma função de remoção), *void imprimeArvoreBilhetes*, *void imprimeArvoreBilhetesAux*, *void imprimeArvoreBilhetesInfixa*, *void apresentarBilhetes*, *void criarPercurso*, *void imprimirPercurso*, *bool autocarroExiste*, *void RemoverPessoaAutocarro*, *void alteraMotorista*, *void inserirBilhetePassageiro*, *void RemoverBilheteParagem*, *Item\* RemoverPessoaFiladeEspera* e *void TextoMenu*.

Também está definida a constante NUM\_PARAGENS 53 que representa a quantidade de paragens total.

## Percurso.cpp

No void alteramotorista é pedido para o utilizador indicar onde o motorista se encontra e depois através de um while e pedido ao utilizador introduzir o primeiro e último nome do motorista. Caso não encontre é incrementado o apontador *temp* definido no início até encontre o motorista ou quando percorre os autocarros todos.

Na função bilhete\* novoNodoArvoreBilhete é criado um novo apontador *novo* para um novo bilhete e depois é definido os valores esquerda e direita, como NULL.

Na próxima função *bilhete\*insereArvoreBilhete* recebe um int e um apontador para a raiz e se o valor a ser inserido for maior do que o valor da raiz é colocado à direita caso contrário é colocado à esquerda.

A função *bool procuraArvoresBilhetes* recebe como argumentos um int *num\_bilhetes* e um apontador para a raiz. Se um o valor inserido for encontrado é atualizado uma variável existe para *true,* caso contrário será *false.* Se o num\_billhetes não for encontrado na raiz, será procurado na direita se for maior que o valor da raiz, caso contrário será procurado na direita.

Para a remoção de bilhetes nas paragens, foram criadas duas funções bilhete\* *removerArvoreBilhetes,* baseadas no código explicado nas aulas para a remoção por cópia. Neste algoritmo de remoção, primeiro encontra-se o nodo a remover. Depois de encontrado, trocamos ele com o nodo mais à direita da sua subárvore esquerda (pois será o maior elemento na árvore inferior a ele mesmo). Após trocado, removemos o elemento.

A função void *imprimeArvoreBilhetes* juntamente com a void *imprimeArvoreBilhetesAux*, têm o propósito de imprimir os elementos da árvore de uma forma mais visual, onde a árvore é apresentada de lado. Nodos acima são os nodos à direita (maiores), e os nodos abaixo são os nodos à esquerda (menores). Na *imprimeArvoreBilhetes*, recebe-se a raíz da árvore, que depois é usada para inicializar *imprimeArvoreBilhetesAux*, juntamente com um inteiro n, que será a altura do nodo atual (inicializado com o valor 0) e um booleano que diz se a posição atual se encontra à esquerda de uma anterior (inicializado como false). Esta função auxiliar faz a impressão da árvore propriamente dita. Começa-se a impressão do maior elemento para o menor, ou seja, do mais à direita até o que está mais à esquerda. Para cada nodo, há um espaçamento proporcional à sua altura na árvore. Quando um nodo tem filhos à esquerda, é posto um “v” à frente do seu valor para indicar que se desloca para baixo. Quando tem filhos à direita, este símbolo será um “^”. Se o nodo atual tiver filhos à esquerda e à direita, então este símbolo será um “<”.

A função *void imprimeArvoreBilhetesInfixa* verifica primeiro se a árvore não tem nodos. Se não tiver nodos na árvore, retorna vazio. Se tiver nodos na árvore, a função vai imprimir em ordem crescente, isto é, vai primeiro para a subárvore esquerda, depois vai para a raiz da árvore e, finalmente, vai para a subárvore direita.

Na função void *apresentarBilhetes* é declarado um int, que representa o número da paragem, de seguida é pedido ao utilizador introduzir o número deste. Depois percorremos as paragens até ser encontrada essa paragem, que de seguida é imprimida a árvore, caso não encontre será incrementada o apontador *temp* para a seguinte paragem. Caso for encontrada a paragem pretendida e esta conter uma árvore não vazia será pedido ao utilizador como quer que seja apresentada a árvore: por travessia infixa (ordem numérica) ou como árvore (representação mais visual da estrutura de dados utilizada).

Já na função void *criarPercurso*, esta recebe como argumento uma lista das paragens e recebe o valor de memória contido no percurso. Durante o decorrer da função é criado uma variável numParagensIntermedias que cria um número aleatório entre 4 e 9, também é criado um numParagensTotal no percurso em que é adicionado 2 ao numParagensIntermedias, é feito também um apontador auxiliar dinâmico do percurso, e é dito de seguida que o aux é o início do percurso, também foi criado um índice que representa as paragens(int i= 1) após isto, é feito a criação das paragens, em que primeiramente é criado o nome da primeira paragem e definida a mesma com índice i, depois passamos para a próxima antes de entrar no ciclo for. No ciclo é criado um índice j e enquanto verificar a condição j <= numParagensIntermedias, irá incrementar o valor de j. Dentro do ciclo for é criado um apontador auxiliar para o nome e é definido o mesmo como listaParagens[j-1], e é incrementado as variáveis pretendidas, finalmente é dado um nome à última paragem e também é feito um ciclo while que verifica se o nome da paragem for igual à primeira, repete a atribuição anterior e verifica de novo.

A função *imprimirPercurso* tem como objetivo imprimir o percurso dos autocarros. Esta função lê as paragens existentes através do destino de cada autocarro representado pela estrutura previamente definida autocarro. Esta função imprime ainda a capacidade de cada autocarro, o número de bilhete de cada passageiro assim como o primeiro nome e último nome de cada passageiro com o auxílio da função *autocarroExiste*.

A função *removerPessoaAutocarro* recebe o apontador para o início da paragem. A função pede ao utilizador para inserir o número de bilhete do passageiro a ser removido do autocarro. É criado um apontador temporário no início do percurso. Se a paragem não tiver autocarro, assumimos que não existem autocarros no percurso. Em outro caso, percorremos as paragens, verificando a lista de passageiros em cada autocarro de cada paragem à procura de um passageiro com esse número de bilhete. Se encontrado, remove-o. Se não encontrarmos até ao fim do percurso ou até não ser encontrado um próximo autocarro, conclui-se que esse número de bilhete não corresponde a nenhum passageiro, e a remoção não acontece.

Na função bool *autocarroExiste*, que recebe um autocarro, retorna *true* caso exista uma matrícula, *false* caso contrário. Se for *false* significa que este autocarro não existe.

Na função void *inserirBilhetePassageiro*, recebe o número de bilhete e a paragem, com o objetivo de atualizar a árvore de bilhetes da paragem, isto é, inserir o número de bilhete na árvore de bilhetes da paragem pretendida.

A função *RemoverBilheteParagem* recebe um percurso, e depois pergunta ao utilizador qual a paragem de onde quer remover um bilhete. De seguida é criado um apontador *temp,* que vai percorrer as paragens e verifica se corresponde à paragem indicada anteriormente se contém uma árvore não vazia. Caso não encontre será incrementado o *temp* para a próxima paragem*.* Se não encontrar a paragem desejada, avisa o utilizador e volta ao menu anterior. Se a paragem dada existir, será apresentada a árvore de bilhetes dessa paragem. Pergunta-se ao utilizador qual o bilhete que deseja remover dessa árvore. Se esse bilhete existir, remove-o utilizando a função de remoção por cópia: *removerArvoreBilhetes*.

A função *RemoverPessoaFiladeEspera* pede ao utilizador um número de bilhete a remover da fila de espera, e depois procura-o e, se encontrado, remove-o.

A função *TextoMenu* imprime o menu principal, isto é, pede ao utilizador para inserir ‘s’ para fazer a próxima iteração, ‘o’ para ver as opções disponíveis ou inserir outra tecla para fechar o programa e terminar a execução.

# Conclusão

Neste trabalho prático, conseguimos implementar todas as funcionalidades que nos tinham sido propostas no enunciado. Aprendemos muito acerca do funcionamento das estruturas de dados usadas ao longo do projeto, e de como definir e seguidamente trabalhar com elas.

Apesar disso, houve um problema estranho de implementação com a árvore de bilhetes que usamos para saber se um bilhete é único (para não repetir bilhetes na criação de uma nova pessoa). Essa árvore estranhamente era sempre inicializada com um valor negativo após ter sido posto nela um valor qualquer. Para resolver o problema, e como este valor seria sempre negativo, decidimos simplesmente passar a raiz para a direita desse elemento e depois removê-lo a si mesmo. De qualquer forma, é possível que houvesse uma maneira melhor de resolver este problema para que não aparecesse esse valor negativo de todo.

Também é de notar que utilizámos código fornecido pelo professor Filipe Quintal, no canal de Discord da cadeira, para imprimir a Fila de Espera sem problemas de espaçamento usando o setw, devido a uma iteração inesperada entre o setw e caracteres especiais que diminuem o espaçamento quando surgem. A função que usamos chama-se *printItem*.

Abaixo encontram-se vários links que nós usamos como referência para algumas partes do nosso código (adaptando-o):

*Estrutura das Listas ligadas*

<https://github.com/kbaras2020/Lab7_2020_1/blob/master/Lab7_2020_1/Fila.h>

*Remoção por cópia nas Árvores de Procura Binária*

<https://github.com/kbaras2020/Lab9_2020/blob/master/Lab9_2020/ArvBinPesquisa.cpp>