Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia

# Relatório do trabalho prático de AED

Sistema de Suporte de uma Rede Ferroviária

*Autores*
69720 Tomás Silva
72360 Jorge Dias

6 de dezembro de 2024

# Conteúdo

# Capítulo 1

# Introdução

Para este trabalho, é proposta a criação dum sistema de suporte de uma rede ferroviária aos alunos, usando Java. Este sistema deve seguir as especificações providenciadas no enunciado e implementar certas operações, abordadas em maior detalhe na secção 2.2.

Os autores, `Jorge Dias` e `Tomás Silva`, alunos do turno prático P7, decidiram implementar este projecto de acordo com as regras da `versão I`, não utilizando o pacote java.util, à excepção da classe Scanner. Salienta-se ainda que os autores ao desenvolver este sistema consideraram as seguinte prioridades de design:

- O número de consultas ao sistema será superior ao de inserções e remoções, pelo que se deu prioridade à eficiência dos métodos de consulta em prol de inserção/remoção;
- Complexidade temporal eficiente > > Complexidade espacial eficiente > Legibilidade/elegância do código

Para contexto do leitor, apresenta-se de seguida a estrutura geral do sistema:

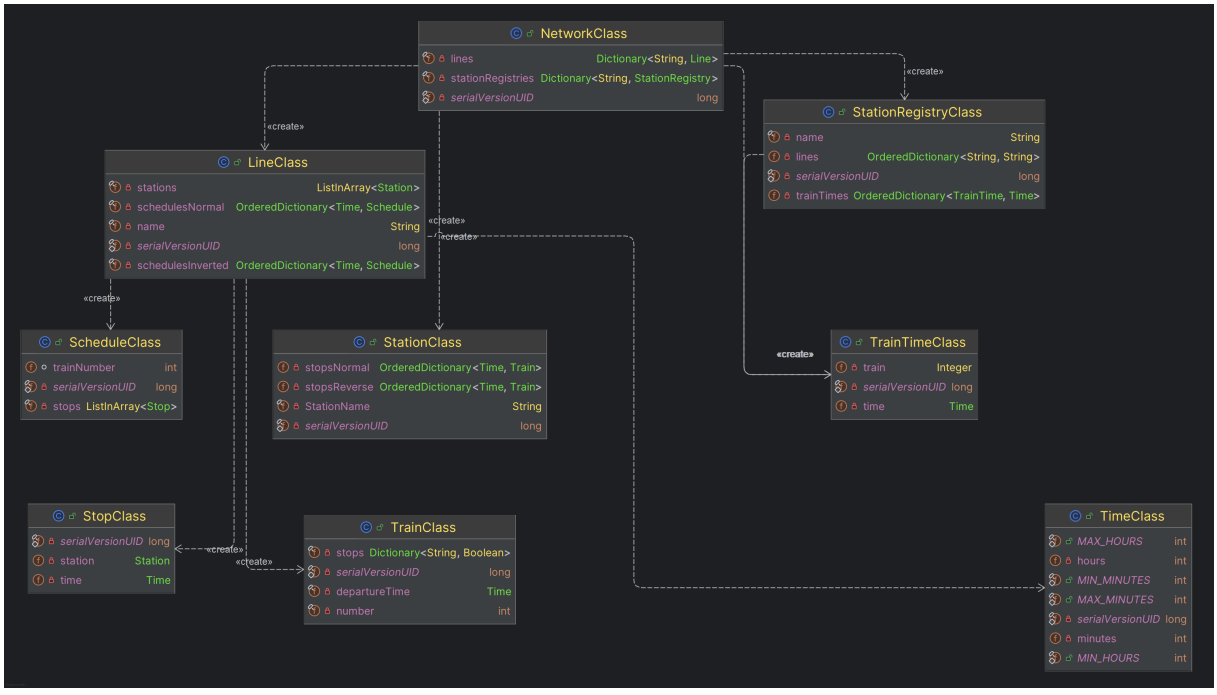


Figura 1.1. Diagrama das principais classes do sistema desenvolvido.

# Capítulo 2

# Detalhes do Sistema

## 2.1 Tipos Abstratos de Dados

Esta secção aborda cada um dos Tipos Abstratos de Dados utilizados, descrevendo as motivações que levaram às escolhas de estruturas de dados para as implementações de cada um deles.

### 2.1.1 Network

A Network serve de interface entre o input da consola e o output dos dados internos e lida diretamente com as operações do sistema, que baseiam-se em uma linha (Line) ou uma estação (StationRegistry). Logo, são mantidas duas coleções, uma para Lines e outra para StationRegestries.

Dado que:

- haverá inserções e remoções;
- tanto uma Line como uma StationRegistry são identificáveis com um nome, que é único no sistema todo;
- é necessário consultar pelo menos uma destas coleções para todas as operações realizadas;
- é expectável haver uma quantia elevada de entradas (500 linhas, 1000 estações);
- não nos é relevante a ordem entre estes dados.

Utilizando o nome da linha ou de estação como chave, escolhemos SepChainHashTables, instanciadas como <String, Line> e <String, StationRegistry>, que permitem-nos ter inserções, remoções e pesquisas rápidas, de complexidade temporal espectável O(ocupação).

Dado que ordem é irrelevante, a dispersão dos elementos pelo array interior não nos afeta. Devido ao elevado número de entradas, os custos da função de dispersão e do espaço para as listas interiores são amortecidos relativamente às possíveis alternativas.

### 2.1.2 `Line`

Uma Line contém as suas estações (Station) e os seus horários (Schedule).

As estações:

- são de dimensão fixa e conhecidas na altura de construção;
- durante a inserção de horário são usadas para validar:
  - que a estação de partida é uma das terminais
  - o sentido do horário;
  - a ordem de estações está correta;
- durante a procura do melhor horário são usadas para validar:
  - o sentido do horário;
- logo, necessitam ser iteráveis em qualquer sentido.
- devem ser listáveis por ordem de inserção;

Escolhemos utilizar uma ListInArray<Station>. Inserções e remoções não são relevantes para este caso, e permite-nos a iteração a partir do inicio ou do fim. Esta iteração permite-nos garantir a validade de um horário e o sentido pretendido na sua realização e na procura de um melhor horário.

Os horários:

- haverá inserções e remoções;
- podem ser realizados em dois sentidos, no sentido normal, partindo da primeira estação, e no sentido inverso, partindo da ultima estação;
- serão utilizados para listagem por ordem de tempo de partida, a partir de uma das terminais;
- contêm a informação completa do melhor comboio encontrado durante a procura pelo melhor horário.

Escolhemos implementar como duas AVLTree<Time, Schedule>. AVLTrees permitem manter a ordem dos seus elementos, e beneficia de operações de inserção, remoção e procura em O(log(n)), e iteração em O(n). Isto é de particular importância para o comboio escolhido durante a procura do melhor horário, cuja hora de partida é utilizada como chave de procura nestas coleções para devolver o melhor horário.

### 2.1.3 `StationRegistry`

Uma StationRegistry contém todas as linhas (Line) e todos os comboios (TrainTime) que passam na estação.

As linhas:

- haverá inserções e remoções;
- têm de ser listáveis por ordem alfanumérica;

Os comboios:

- haverá inserções e remoções;
- têm de ser listáveis por ordem de passagem

Para ambas as coleções escolhemos utilizar AVLTree<String, String> e <TimeTrain, Time> pelas mesmas razões mencionadas anteriormente, com ênfase no número de comboios, pois serão todos os comboios de todas a linhas que passam na estação.

### 2.1.4 `Station`

uma Station contém as paragens realizadas nessa mesma estação pelos comboios (Train) da linha a que pertencem.

As paragens:

- haverá inserções e remoções;
- serão nos dois sentidos já mencionados;
- são utilizadas durante a inserção de horários para impedir situações de ultrapassagem;
- são utilizadas durante a procura pelo melhor horário para descobrir os potenciais horários;
- devem ser listáveis por ordem de hora de passagem na estação.

Mais uma vez, duas AVLTree<Time, Train>, sendo estruturas adequadas para lidar com o potencial elevado número de inserções e remoções, enquanto mantendo a ordem que nos é mais util para resolver e validar horários.

### 2.1.5 `Schedule`

Uma Schedule contém uma coleção de paragens (Stop), que relacionam uma estação com um tempo.

As paragens:

- são de dimensão fixa e conhecidas na altura de construção;
- devem ser listáveis por ordem de inserção.

Como na coleção de estações na Line, escolhemos ListInArray<Stop> para guardar as paragens.

### 2.1.6 `Train`

Um Train contém uma coleção das estações (Station) que indica se para ou não nela.

As estações:

- são de dimensão fixa e conhecidas na altura de construção;
- no entanto, haverá atualizações, devido ao mecanismo implementado durante a inserção de um horário;

- utilizadas durante a procura do melhor horário para validar se o comboio para na estação desejada ou não;
- logo, a ordem em si não é relevante.

Escolhemos utilizar uma SepChainHashTables<String, Boolean>, pois durante a procura do melhor horário é importante ser o mais rápido possível. Como já sabemos durante a procura a estação de partida que desejamos, podemos perguntar ao Train se para ou não na estação que desejamos, permitindo uma resposta mais rápida.

### 2.1.7 `Misc.`

As restantes TADs Stop, Time e TrainTime não implementam estruturas de dados complexas.

## 2.2 Operações

Esta secção oferece uma descrição do comportamento do sistema a cada uma das operações descritas no enunciado, em termos das operações efetuadas sobre as estruturas de dados.

### 2.2.1 IL - Inserção de linha

| | |
|---|---|
| Entrada | IL nome-linha<br>nome-estacao-1<br>...<br>nome-estacao-n |

Tabela 2.1. Input do comando IL.

**Classes usadas**

- Network
- Line
- StationRegistry
- Station

**Estruturas usadas**

- ListInArray<String>
- Dictionary SepChainHashTable<String, Line> 'lines'
- Dictionary SepChainHashTable<String, StationRegistry> 'stations'
- ListInArray<Station> 'stations'
- AVLTree<String, String> 'lines'

**Comportamento do Sistema**

O sistema guarda o 'nome-linha' numa String

O sistema preenche um ListInArray<String> com os 'nome-estacao'

A network tenta aceder a uma Line no SepChainHashTable 'lines' com o 'nome-linha'; Se conseguir, dá erro. Se falhar, continua:

Cria uma nova Line, usando o 'nome-linha'.

A network itera a ListInArray<String> com os 'nome-estacao', criando um novo objeto Station por cada e procurando se já existe algum StationRegistry no SepChainHashTable 'stations' com cada nome. Se não existir, cria um novo StationRegistry com esse nome. Depois adiciona à AVLTree 'lines' desse StationRegistry o 'nome-linha'. E adiciona à nova Line a Station criada, no ListInArray<Station> 'stations'.

De seguida, a nova Line, já preenchida com Stations, é adicionada à SepChainHashTable 'lines' da Network.

### 2.2.2 `RL - Remoção de linha`

| Entrada | RL nome-linha |
|---|---|

Tabela 2.2. Input do comando RL.

**Classes usadas**

- Network
- Line
- StationRegistry
- Station
- TrainTime
- Time

**Estruturas usadas**

- Dictionary SepChainHashTable<String, Line> 'lines'
- Dictionary SepChainHashTable<String, StationRegistry> 'stations'
- ListInArray<Station> 'stations'
- OrderedDictionary AVLTree<String, String> 'lines'
- OrderedDictionary< AVLTree<Time,Train> 'stopsNormal'
- OrderedDictionary AVLTree<Time,Train> 'stopsReverse'
- OrderedDictionary AVLTree<TrainTime, Time> 'trainTimes'

**Comportamento do Sistema**

O sistema guarda o 'nome-linha' numa String

A network tenta remover a Line com 'nome-linha' do SepChainHashTable 'lines'; Se a Line devolvida for null, dá erro. Se for diferente de null, continua:

É feita uma iteração sobre todas as Stations da Line removida, acedendo a cada StationRegistry correspondente às Stations existentes. É removido do AVLTree 'lines' o 'nome-linha' referido. Caso a AVLTree 'lines' fique vazia, o StationRegistry é removido do SepChainHashTable 'stations' da Network; Caso a AVLTree 'lines' não fique vazia, é feito o passo adicional de remoção de TrainTimes, descrito de seguida:

Itera-se sobre as AVLTrees da Station em causa (sentido normal e sentido inverso) e para cada par <Time, Train> registado na Station, é removido do StationRegistry correspondente o TrainTime referente a esse par. Desta forma o registo da passagem dos comboios por esta linha é apagado dos StationRegistries, mantendo a integridade do registo.

### 2.2.3 `CL - Consulta de estações de uma linha`

| Entrada | CL nome-linha |
|---|---|

Tabela 2.3. Input do comando CL.

**Classes usadas**

- Network
- Line
- Station

**Estruturas usadas**

- Dictionary<String, Line> SepChainHashTable<String, Line> 'lines'
- ListInArray<Station> 'stations'

**Comportamento do Sistema**

O sistema guarda o 'nome-linha' numa String

A Network procura a Line com 'nome-linha' no SepChainHashTable 'lines'; Se a Line devolvida for null, dá erro. Se for diferente de null, a Line devolve um Iterator de Stations, que é depois iterado pelo sistema para devolver o nome de cada Station no output.

### 2.2.4 `CE - Consulta das linhas de uma estação`

| Entrada | CE nome-estação |
|---|---|

Tabela 2.4. Input do comando CE.

**Classes usadas**

- Network
- Network
- StationRegistry

**Estruturas usadas**

- Dictionary<String, StationRegistry> 'stations'
- OrderedDictionary<String, String> AVLTree<String, String> 'lines'

**Comportamento do Sistema**

O sistema guarda o 'nome-estação' numa String

A Network procura a StationRegistry com 'nome-estação' no SepChainHashTable 'stations'; Se a StationRegistry devolvida for null, dá erro. Se for diferente de null, o StationRegistry devolve um Iterator de Strings (com line names), que é depois iterado pelo sistema para devolver o nome de cada Line no output.

### 2.2.5 IH - Inserção de horário

| | |
|---|---|
| Entrada | IH nome-linha<br>número-comboio<br>nome-estacao-1 hora-1<br>...<br>nome-estacao-n hora-n |

Tabela 2.5. Input do comando IH.

**Classes usadas**

- Network
- Line
- Station
- Time
- Train
- Stop
- Schedule
- StationRegistry
- TrainTime

**Estruturas usadas**

- Dictionary SepChainHashTable<String, Line> 'lines'
- ListInArray<Station> 'stations'
- OrderedDictionary AVLTree<Time,Train> 'stopsNormal'
- OrderedDictionary AVLTree<Time,Train> 'stopsReverse'
- ListInArray<Stop>
- OrderedDictionary AVLTree <Time, Schedule> 'schedulesNormal'
- OrderedDictionary AVLTree <Time, Schedule> 'schedulesInverted'
- Dictionary SepChainHashTable<String, StationRegistry> 'stations'
- OrderedDictionary AVLTree<TrainTime, Time> 'trainTimes'

**Comportamento do Sistema**

O sistema guarda o 'nome-linha' numa String

O sistema guarda o 'número-comboio' noutra String

O sistema preenche um ListInArray<String> com os 'nome-estacao' e 'hora'

A network tenta aceder a uma Line no SepChainHashTable 'lines' com o 'nome-linha'; Se a Line devolvida for null, dá erro. Se for diferente de null, continua:

Dentro da Line, é verificada a integridade dos dados fornecidos pelo input:

Em primeiro lugar, verifica-se se a String da primeira 'nome-estacao' fornecida corresponde ao nome da primeira ou última Station da ListInArray<Station> 'stations' da Line. Se não corresponder dá erro e é estabelecida a direção deste futuro Schedule.

Em segundo lugar itera sobre 'nome-estacao' e 'hora', verificando se são estritamente crescentes e se seguem a sequência de Stations em ListInArray<Station> 'stations' da Line - iterando sobre estas stations, comparando com os outros tempos em OrderedDictionary<Time, Train> AVLTree<Time,Train> 'stopsReverse/Normal' dentro de cada Station.

Se algum destes falhar, dá erro. Se não, continua:

É criado um novo ListInArray<Stop> e Train. Para cada par de input 'nome-estacao' e 'hora', procura-se a Station correspondente em ListInArray<Station> 'stations' na Line, adiciona-se uma nova entrada em OrderedDictionary<Time, Train> AVLTree<Time,Train> stops(Reverse/Normal); Adiciona-se uma nova entrada a Dictionary SepChainHashTable<String, Boolean> 'stops' num novo Train;

E adiciona-se uma nova Stop à ListInArray<Stop>- esta ListInArray<Stop> é depois usada para inicializar um novo Schedule, que é colocado na respectiva OrderedDictionary AVLTree <Time, Schedule> 'schedulesNormal/Inverted' nesta Line.

Finalmente, na Network, acede-se ao StationRegistry com o nome de cada Stop guardada ainda em ListInArray<Stop>, adicionando um TrainTime na OrderedDictionary AVLTree<TrainTime, Time> 'trainTimes' a cada StationRegistry chamado.

### 2.2.6 `RH - Remoção de horário`

| | |
|---|---|
| Entrada | RH nome-linha<br>nome-estacao-partida hora-partida |

Tabela 2.6. Input do comando RH.

**Classes usadas**

- Network
- Line
- Station
- Time
- Train
- Stop
- Schedule
- StationRegistry
- TrainTime

**Estruturas usadas**

- Dictionary SepChainHashTable<String, Line> 'lines'
- OrderedDictionary AVLTree <Time, Schedule> 'schedulesNormal'
- OrderedDictionary AVLTree <Time, Schedule> 'schedulesInverted'
- ListInArray<Stop> 'stops'
- ListInArray<Station> 'stations'
- OrderedDictionary AVLTree<Time,Train> 'stopsNormal'
- OrderedDictionary AVLTree<Time,Train> 'stopsReverse'
- Dictionary SepChainHashTable<String, StationRegistry> 'stations'
- OrderedDictionary AVLTree<TrainTime, Time> 'trainTimes'

**Comportamento do Sistema**

O sistema guarda o 'nome-linha' numa String.

O sistema guarda o 'nome-estacao-partida' noutra String.

O sistema guarda o 'hora-partida' noutra String.

A network tenta aceder a uma Line no SepChainHashTable 'lines' com o 'nome-linha'; Se a Line devolvida for null, dá erro. Se for diferente de null, continua:

Dentro da Line, é verificada a integridade dos dados fornecidos pelo input:

Em primeiro lugar, verifica-se se a String da 'nome-estacao-partida' fornecida corresponde ao nome da primeira ou última Station da ListInArray<Station> 'stations' da Line, determinando em qual dos OrderedDictionary AVLTree <Time, Schedule> 'schedules-Normal/Inverted' o Schedule se pode encontrar. Acede-se à AVLTree e remove-se o dito Schedule. Se nada tiver sido removido, devolve erro.

Em segundo lugar, itera sobre ListInArray<Stop> stops do Schedule, iterando também ListInArray<Station> 'stations' de Line, removendo dentro dos OrderedDictionary AVLTree<Time,Train> 'stopsReverse/Normal' em cada Station os pares <Time,Train> correspondentes.

Finalmente, na Network, acede-se ao StationRegistry com o nome de cada Stop guardada dentro de ListInArray<Stop> no Schedule removido, removendo um TrainTime na OrderedDictionary AVLTree<TrainTime, Time> 'trainTimes' a cada StationRegistry chamado.

### 2.2.7 `CH - Consulta dos horários de uma linha`

| | |
|---|---|
| Entrada | CH nome-linha<br>nome-estacao-partida |

Tabela 2.7. Input do comando CH.

**Classes usadas**

- Network
- Line
- Time
- Schedule

**Estruturas usadas**

- Dictionary SepChainHashTable<String, Line> 'lines'
- OrderedDictionary AVLTree <Time, Schedule> 'schedulesNormal'
- OrderedDictionary AVLTree <Time, Schedule> 'schedulesInverted'

**Comportamento do Sistema**

O sistema guarda o 'nome-linha' numa String

O sistema guarda o 'nome-estacao-partida' noutra String

A network tenta aceder a uma Line no SepChainHashTable 'lines' com o 'nome-linha'; Se a Line devolvida for null, dá erro. Se for diferente de null, continua:

Dentro da Line, é verificado se o 'nome-estacao-partida' corresponde à primeira Station de OrderedDictionary AVLTree <Time, Schedule> 'schedulesNormal/Inverted'. Se não, dá erro.

Se sim, devolve o iterador da estrutura, devolvendo uma sequência de Stops dentro de Schedules no output, já ordenados pela ordem correta desde inserção.

### 2.2.8 `LC - Comboios por estação`

| | |
|---|---|
| Entrada | LC nome-estação |

Tabela 2.8. Input do comando LC.

**Classes usadas**

- Network
- StationRegistry
- TrainTime

**Estruturas usadas**

- Dictionary SepChainHashTable<String, StationRegistry> 'stations'
- AVLTree OrderedDictionary<TrainTime, Time> trainTimes

**Comportamento do Sistema**

O sistema guarda o 'nome-estação' numa String

A Network procura a StationRegistry com 'nome-estação' no SepChainHashTable 'stations'; Se a StationRegistry devolvida for null, dá erro. Se for diferente de null, o StationRegistry devolve um Iterator de AVLTree OrderedDictionary<TrainTime, Time> 'trainTimes', passando para o output uma sequência de valores de TrainTimes registados naquela StationRegistry

### 2.2.9 `MH - Melhor horário`

| | |
|---|---|
| Entrada | MH nome-linha<br>nome-estacao-partida<br>nome-estacao-destino<br>hora-chegada-esperada |

Tabela 2.9. Input do comando MH.

**Classes usadas**

- Network
- Line
- Station
- Time
- Train
- Stop
- Schedule

**Estruturas usadas**

- Dictionary SepChainHashTable<String, Line> 'lines'
- ListInArray<Station> 'stations'
- OrderedDictionary AVLTree<Time,Train> 'stopsNormal'
- OrderedDictionary AVLTree<Time,Train> 'stopsReverse'
- Stack<Train>
- OrderedDictionary AVLTree <Time, Schedule> 'schedulesNormal'
- OrderedDictionary AVLTree <Time, Schedule> 'schedulesInverted'

**Comportamento do Sistema**

O sistema guarda o 'nome-linha' numa String

O sistema guarda o 'nome-estacao-partida' noutra String

O sistema guarda o 'nome-estacao-destino' noutra String

O sistema guarda o 'hora-partida' noutra String

A network tenta aceder a uma Line no SepChainHashTable 'lines' com o 'nome-linha'; Se a Line devolvida for null, dá erro. Se for diferente de null, continua:

Dentro da Line, é verificada a integridade dos dados fornecidos pelo input: Em primeiro lugar, verifica-se se a String da 'nome-estacao-partida' fornecida corresponde ao nome da primeira ou última Station da ListInArray<Station> 'stations' da Line, determinando em qual dos OrderedDictionary AVLTree <Time, Schedule> 'schedulesNormal/Inverted' o Schedule se pode encontrar. Em simultâneo é feita a verificação de se a 'nome-estacao-destino' existe no mesmo Schedule, seguindo a sequência. Se não se encontrar a 'nome-estacao-partida' é levantado um erro, e se não se encontrar 'nome-estacao-destino', um outro erro diferente é levantado.

Confirmada a validade do input, acede-se à OrderedDictionary AVLTree<Time,Train> 'stopsNormal/Reverse' da Station correspondente a 'nome-estacao-destino', iterando e guardando num Stack<Train> os vários Train até encontra o que passe a 'hora-partida' do input.

Para cada um destes Train no Stack, verifica-se se ele efetivamente parte da 'nome-estacao-partida'. Se sim, usando o departureTime deste Train, devolve-se o Schedule de OrderedDictionary AVLTree <Time, Schedule> 'schedulesNormal/Inverted'

Finalmente, o sistema devolve os vários valores do iterador deste Schedule.

### 2.2.10 `TA - Terminar aplicação`

Serializa tudo o que for serializável e encerra o programa.

| Entrada | TA |
|---|---|

Tabela 2.10. Input do comando TA.

## 2.3 Complexidade Temporal

Esta secção apresenta o estudo das complexidades temporais de cada uma das operações descritas no enunciado, no melhor caso, pior caso e caso esperado.

### 2.3.1 IL - Inserção de linha

**Aceder a Line em SepChainHashTable<String, Line> 'lines'**

- Melhor caso: O(1)
- Pior caso: O(n)
- Caso esperado: O(1)

**Iterar sobre ListInArray<String> de nomes de estações**

- Melhor caso: O(m)
- Pior caso: O(m)
- Caso esperado: O(m)

**Aceder a StationRegistry em SepChainHashTable<String, StationRegistry> 'stations'**

- Melhor caso: O(1)
- Pior caso: O(q)
- Caso esperado: O(1)

**Inserir nova String (line name) em AVLTree<String, String> 'lines'**

- Melhor caso: O(log k)
- Pior caso: O(log k)
- Caso esperado: O(log k)

**Inserir nova Line em SepChainHashTable<String, Line> 'lines'**

- Melhor caso: O(1)
- Pior caso: O(1)
- Caso esperado: O(1)

**Complexidade total**

- Melhor caso: O(m * log k)
- Pior caso: O(n + m * q * log k)
- Caso esperado: O(m * log k)

*onde m é o número de estações inseridas no input, k é tamanho da AVLTree<String, String> 'lines' em cada StationRegistry, n e q são o número de colisões nas SepChainHashTable

### 2.3.2 `RL - Remoção de linha`

**Remover a Line em SepChainHashTable<String, Line> 'lines'**

- Melhor caso: O(1)
- Pior caso: O(n)
- Caso esperado: O(1)

**Iterar sobre ListInArray<Station> 'stations' de nomes de estações**

- Melhor caso: O(m)
- Pior caso: O(m)
- Caso esperado: O(m)

**Aceder a StationRegistry em SepChainHashTable<String, StationRegistry> 'stations'**

- Melhor caso: O(1)
- Pior caso: O(q)
- Caso esperado: O(1)

**Remover a String (line name) em AVLTree<String, String> 'lines'**

- Melhor caso: O(log k)
- Pior caso: O(log k)
- Caso esperado: O(log k)

**Iterar sobre OrderedDictionary AVLTree<Time,Train> 'stopsNormal/Reverse'**

- Melhor caso: O(s)
- Pior caso: O(s)
- Caso esperado: O(s)

**Remover o TrainTime em AVLTree<TrainTime, Time> 'trainTimes'**

- Melhor caso: O(log r)
- Pior caso: O(log r)
- Caso esperado: O(log r)

**Complexidade total**

- Melhor caso: O(m * log k + m * q * s * log r)
- Pior caso: O(n + m * q * log k + m * q * s * log r)
- Caso esperado: O(m * log k + m * q * s * log r)

*onde m é o número de estações da linha removida, k é tamanho da AVLTree<String, String> 'lines' em cada StationRegistry, s é o tamanho da AVLTree<Time,Train> 'stops-Normal/Reverse' em cada Station, n e q são o número de colisões nas SepChainHashTable, r é o tamanho de AVLTree<TrainTime, Time> 'trainTimes' em cada StationRegistry

### 2.3.3 `CL - Consulta de estações de uma linha`

**Aceder a Line em SepChainHashTable<String, Line> 'lines'**

- Melhor caso: O(1)
- Pior caso: O(n)
- Caso esperado: O(1)

**Iterar sobre Iterator das Stations**

- Melhor caso: O(m)
- Pior caso: O(m)
- Caso esperado: O(m)

**Complexidade total**

- Melhor caso: O(m)
- Pior caso: O(n + m)
- Caso esperado: O(m)

*onde m é o número de estações da linha, n é o número de colisões na SepChainHashTable

### 2.3.4 `CE - Consulta das linhas de uma estação`

**Aceder a StationRegistry em SepChainHashTable<String, StationRegistry> 'stations'**

- Melhor caso: O(1)
- Pior caso: O(n)
- Caso esperado: O(1)

**Iterar sobre Iterator das Strings (line names)**

- Melhor caso: O(m)
- Pior caso: O(m)
- Caso esperado: O(m)

Complexidade total

**Complexidade total**

- Melhor caso: O(m)
- Pior caso: O(n + m)

- Caso esperado: O(m)

*onde m é o número de linhas no StationRegistry, n é o número de colisões na SepChainHashTable

### 2.3.5 `IH - Inserção de horário`

**Aceder a Line em SepChainHashTable<String, Line> 'lines'**

- Melhor caso: O(1)
- Pior caso: O(n)
- Caso esperado: O(1)

**Verificar se a primeira estação dada no input corresponde a uma das estações terminais de ListInArray<Station> 'stations'**

- Melhor caso: O(1)
- Pior caso: O(1)
- Caso esperado: O(1)

**Iterar o input: Verificar se os Time seguem ordem crescente e comparar se a sequência de Stations da linha corresponde à dada (a complexidade no pior caso será igual ao pior caso entre estas coleções**

- Melhor caso: O(m)
- Pior caso: O(max(m,r))
- Caso esperado: O(max(m,r))

**Ainda na iteração anterior, verificar para cada Station do input se existe validade relativamente a outras Train que passem nessa estação, iterando sobre OrderedDictionary AVLTree<Time, Train> 'stopsNormal/Reverse' na Station**

- Melhor caso: O(s)
- Pior caso: O(s)
- Caso esperado: O(s)

**Itera-se de novo sobre as Stations de input, procurando cada uma delas em ListInArray<Station> 'stations' na Line**

- Melhor caso: O(m)
- Pior caso: O(m*r)
- Caso esperado: O(m*r)

**Insere-se um Train em OrderedDictionary AVLTree<Time, Train> 'stopsNormal/Reverse' de cada Station**

- Melhor caso: O(log k)
- Pior caso: O(log k)

- Caso esperado: O(log k)

**Insere-se uma stop em Train, sendo que Train contém Dictionary SepChainHashTable<String, Boolean> stops**

- Melhor caso: O(1)
- Pior caso: O(1)
- Caso esperado: O(1)

**É criado um Schedule, e inserido em OrderedDictionary AVLTree <Time, Schedule> 'schedulesNormal/Reverse' na Line**

- Melhor caso: O(log z)
- Pior caso: O(log z)
- Caso esperado: O(log z)

**Itera-se mais uma vez as Stations de input**

- Melhor caso: O(m)
- Pior caso: O(m)
- Caso esperado: O(m)

**Acede-se ainda a StationRegistry em SepChainHashTable<String, StationRegistry> 'stations'**

- Melhor caso: O(1)
- Pior caso: O(p)
- Caso esperado: O(1)

**Insere-se um elemento em OrderedDictionary AVLTree<TrainTime, Time> 'trainTimes'**

- Melhor caso: O(log x)
- Pior caso: O(log x)
- Caso esperado: O(log x)

**Complexidade total**

- Melhor caso: O(m * s + m * log k + log z + m * log x)
- Pior caso: O(max(m,r) *s + m * r * log k + log z + m*p * log x)
- Caso esperado: O(max(m,r) * s +m * r * log k+ log z + m * log x)

*onde m é o número de linhas no input, r é o número de Stations existentes na Line, s é o número de stops em cada Station, k é o número de Trains em cada Station, z é o número de Schedules na Line, x o número de trainTimes em cada StationRegistry, n , p são o número de colisões nas SepChainHashTables

### 2.3.6 `RH - Remoção de horário`

**Aceder a Line em SepChainHashTable<String, Line> 'lines'**

- Melhor caso: O(1)
- Pior caso: O(n)
- Caso esperado: O(1)

**Verificar se a primeira estação dada no input corresponde a uma das estações terminais de ListInArray<Station> 'stations'**

- Melhor caso: O(1)
- Pior caso: O(1)
- Caso esperado: O(1)

**É removido um Schedule, de OrderedDictionary AVLTree <Time, Schedule> 'schedulesNormal/Reverse' na Line**

- Melhor caso: O(log z)
- Pior caso: O(log z)
- Caso esperado: O(log z)

**Itera-se sobre as ListInArray<Stop> stops deste Schedule, iterando lá dentro as stations ListInArray<Station> stations para comparar**

- Melhor caso: O(m * r)
- Pior caso: O(m * r)
- Caso esperado: O(m * r)

**Em cada Station que encontra, remove elementos de OrderedDictionary AVL-Tree<Time,Train> 'stopsReverse/Normal'**

- Melhor caso: O(log x)
- Pior caso: O(log x)
- Caso esperado: O(log x)

**Iterar sobre ListInArray<Stop> proveniente do Schedule removido**

- Melhor caso: O(m)
- Pior caso: O(m)
- Caso esperado: O(m)

**Aceder a StationRegistry em SepChainHashTable<String, StationRegistry> 'stations'**

- Melhor caso: O(1)
- Pior caso: O(p)
- Caso esperado: O(1)

**Remover o TrainTime em AVLTree<TrainTime, Time> 'trainTimes'**

- Melhor caso: O(log t)
- Pior caso: O(log t)
- Caso esperado: O(log t)

**Complexidade total**

- Melhor caso: O(log z + m*r * log x + m * log t)
- Pior caso: O(n + log z + m*r * log x + m * p * log t)
- Caso esperado: O(log z + m*r * log x + m * log t)

*onde z é o número de Schedules na Line, m é o número de Stops em cada Schedule, r é o número de Stations existentes na Line, x é o número de pares <Time,Train> em cada Station, t é o número de TrainTime em cada StationRegistry, n , p são o número de colisões nas SepChainHashTables

### 2.3.7 `CH - Consulta dos horários de uma linha`

**Aceder a Line em SepChainHashTable<String, Line> 'lines'**

- Melhor caso: O(1)
- Pior caso: O(n)
- Caso esperado: O(1)

**Iterar sobre Iterator dos Schedules e Stops**

- Melhor caso: O(m * p)
- Pior caso: O(m * p)
- Caso esperado: O(m * p)

**Complexidade total**

- Melhor caso: O(m * p)
- Pior caso: O(n + m * p)
- Caso esperado: O(m * p)

*onde m é o número de Schedules na Line, p é o número de Stops em cada Schedule, n é o número de colisões na SepChainHashTable

### 2.3.8 `LC - Comboios por estação`

**Aceder a StationRegistry em SepChainHashTable<String, StationRegistry> 'stations'**

- Melhor caso: O(1)
- Pior caso: O(n)
- Caso esperado: O(1)

**Iterar sobre Iterator dos TrainTimes, AVLTree OrderedDictionary<TrainTime, Time> trainTimes**

- Melhor caso: O(m)
- Pior caso: O(m)
- Caso esperado: O(m)

**Complexidade total**

- Melhor caso: O(m)
- Pior caso: O(n + m)
- Caso esperado: O(m)

*onde m é o número de TrainTimes na StationRegistry, n é o número de colisões na SepChainHashTable

### 2.3.9 `MH - Melhor horário`

**Aceder a Line em SepChainHashTable<String, Line> 'lines'**

- Melhor caso: O(1)
- Pior caso: O(n)
- Caso esperado: O(1)

**Verificar se a estação de origem dada no input corresponde a uma das estações de ListInArray<Station> na Line + Verificar se a estação de destino dada no input corresponde a uma das estações de ListInArray<Station> na Line**

- Melhor caso: O(1)
- Pior caso: O(2m) = O(m)
- Caso esperado: O(m)

**Guardar num Stack todas os Trains de OrderedDictionary AVLTree<Time,Train> 'stopsNormal/Reverse' na Station de destino menores que a hora do input**

- Melhor caso: O(1)
- Pior caso: O(s)
- Caso esperado: O(s)

**Verificar qual o primeiro dos Trains do Stack que passa na estação de Partida, Dictionary SepChainHashTable<String, Boolean> stops**

- Melhor caso: O(1)
- Pior caso: O(s)
- Caso esperado: O(s)

**Usar esse Train para aceder ao Schedule em OrderedDictionary AVLTree <Time, Schedule> 'schedulesNormal/Reverse'**

- Melhor caso: O(log t)
- Pior caso: O(log t)
- Caso esperado: O(log t)

**Iterar sobre Iterator de Stops dentro do Schedule, ListInArray<Stop> stops, produzindo output**

- Melhor caso: O(k)
- Pior caso: O(k)
- Caso esperado: O(k)

**Complexidade total**

- Melhor caso: O( log t + k)
- Pior caso: O(n + m + s + k)
- Caso esperado: O(m + s + log t + k)

*onde m é o número de Schedules na Line, s é o número de Trains na Station de destino, t é o número de Schedules na Line , k é o número Stops num Schedule, n são o número de colisões nas SepChainHashTables

### 2.3.10 `TA - Terminar aplicação`

**Complexidade total**

- O(1)

## 2.4 Complexidade Espacial

Esta secção apresenta o estudo da complexidade espacial do sistema proposto como solução:

Complexidade Espacial: O(L + L * S + L * S * C + L * S * C * p + L * H + L * H * P + SR + SR * LN + SR * TT)

L: Número de Lines

S: Número médio de Stations por Line

C: Número médio de Trains por Station

p: Número médio de stops por Train

H: Número médio de Schedules por Line

P: Número médio de Stops por Schedule

SR: Número de StationRegistry

LN: Número médio de line names por StationRegistry

TT: Número médio de TrainTime por StationRegistry

# Capítulo 3

# Notas Finais

Os autores gostariam de deixar como notas finais que há muito que poderia ser melhorado neste projeto:

Em primeiro lugar, um refactor das StationRegistry e Station (merge destas duas estruturas) simplificaria imenso a complexidade espacial da solução (se bem que a custo de modularidade) - Foi algo que considerámos mas acabámos por dedicar o nosso tempo a outros ajustes;

Nas Stations, os pares <Time, Train> podiam perfeitamente ser da class TrainTime - bem como a coleção <TrainTime,Time> poderia ser simplicada para não ser um par (o valor de Time está repetido). Isto deve-se a um refactor tardio que foi feito por erro de interpretação do enunciado relativamente ao comando LC, que julgámos ser para listar os Comboios por hora de partida da estação inicial, ao invés de ser por hora de passagem na estação em questão. Apercebemo-nos do erro a tempo, mas ficaram no código alguns artefactos da implementação anterior, como este.

Gostaríamos de ter reduzido a presença das flags de isInverted no código. A sua presença em tantos sítios pode complicar a legibilidade do código.

Idealmente teríamos ainda feito uma revisão profunda de cada estrutura usada, repensando alguns edge cases com que nos deparámos na fase final e que podiam ser melhorados alterando algumas das estruturas usadas para serem mais simples.