

PROJETO ESINF : PARTE II METRO DE PARIS



Trabalho realizado por:

João Flores (1171409);

José Mota (1161263);

Este projeto insere-se no âmbito na cadeira estruturas de informação com o objetivo de manutenção e gestão da linha de metro de Paris.

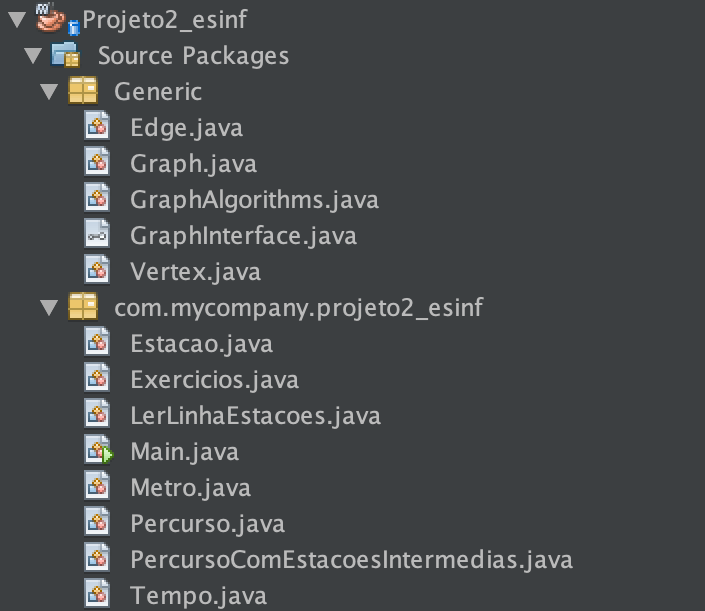
De modo a alcançar os objetivos propostos foi realizada uma analise metódica para aplicar uma estrutura correta e ponderar como realizar os objetivos.

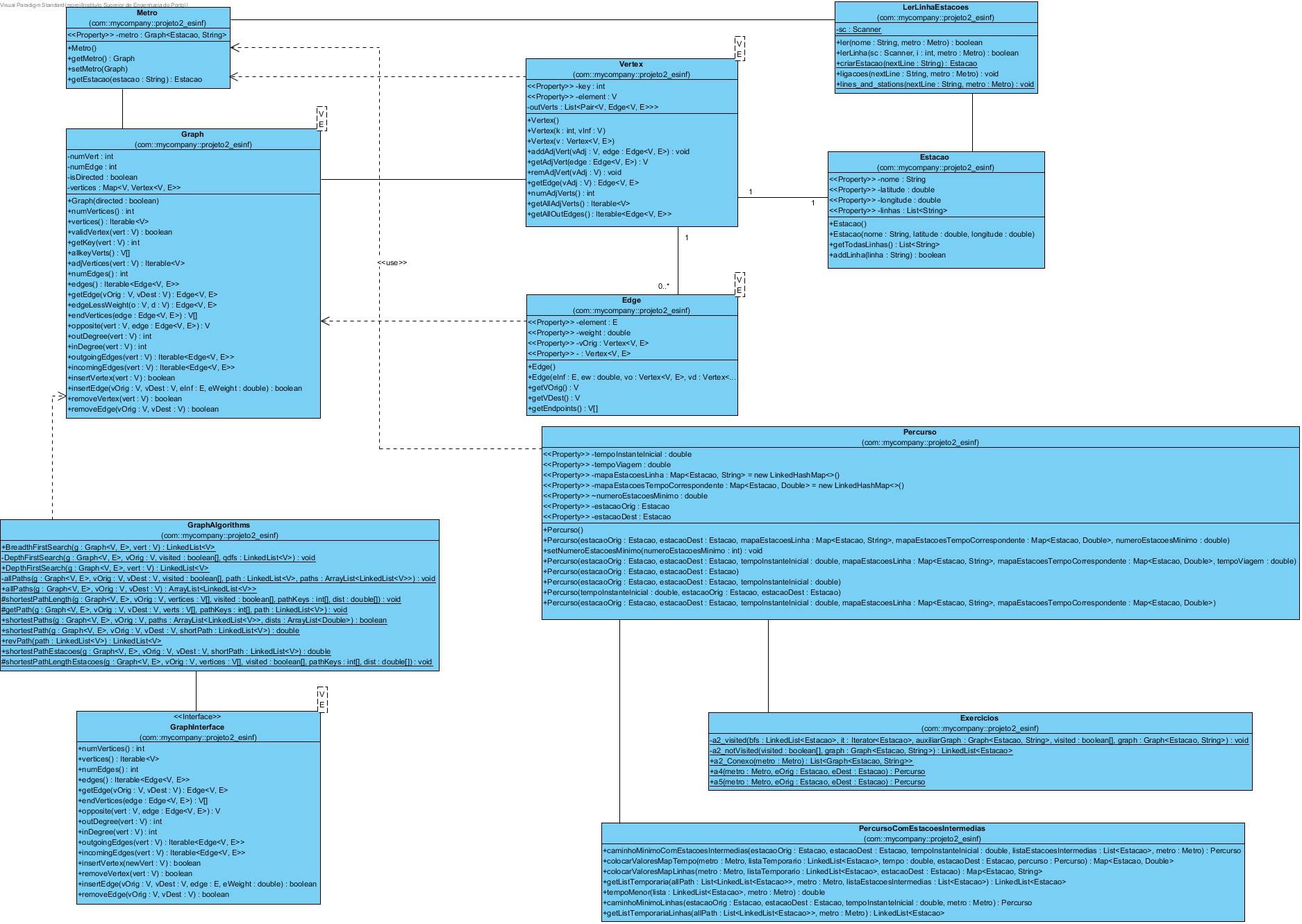
Os atributos mais pertinentes é a sugestão de trajetos para os utentes conforme as necessidades destes.

Dada a complexidade da estrutura a utilização de grafos com vértices e ligações é uma estrutura adequada para se puder implementar as necessidades dos utentes.

Para a realização deste projeto, tivemos a necessidade de implementar as seguintes classes :

* Metro
* Estacao
* Connections,LerLinhaEstacoes
* Classes onde foram implementados os requisitos pedidos no enunciado( Exercicios e PercursoComEstacoesIntermedias
* Classes genéricas : Edge,Vertex Graph, GraphAlgorithms
* Tempo



Primeiramente é necessário importar as estacoes mo metro de paris.

***NOTA: O DIAGRAMA DE CLASSES SEGUE DENTRO DA PASTA SUBMETIDA UMA VEZ QUE NÃO SE VÊ NITIDAMENTE!!***

Criar o grafo da rede de metro de Paris

A importação de dados é uma necessidade para qualquer projeto deste tipo, logo o ponto inicial do nosso projeto foi a criação de métodos para ler, instanciar e adicionar à estrutura.

Foi fornecido três ficheiros com extensão .csv, tipo de ficheiro texto separado por “;” foi criado um objeto do tipo scanner para ler linha a linha até ao fim do ficheiro.

A existência de um método que faz a distinção da instanciação, que vai ser redirecionado conforme o nome do ficheiro, a utilização deste método serve para evitar código repetido.

O método criarEstacao tem como objetivo separar a linha e colocar num vetor de String, a criação da estação é feita ao colocar os elementos do vetor por ordem do construtor. Quando criar a estação retorna-a e no método lerLinha coloca no grafo usando o insertVertex.

Quando as estações já estiverem todas no grafo é necessário colocar todas as linhas que a estação tem. Obtemos a estação pelo segundo parâmetro da linha e adicionamos a uma lista que contem todas as linhas da estação.

Com a utilização de grafos é necessário estabelecer as entre as estações, a ligação do parâmetro um com o parâmetro dois da linha representam respetivamente a estação de origem e destino, o peso entre as duas estações e também a respetiva linha. A utilização do insertEdge para adicionar a ligação entre as duas estações.

Verificar se o grafo é conexo, se não for devolver os seus componentes.

Para este requisito, tivemos de compreender o conceito de conexo. Para ser conexo, um temos de conseguir, através, de um ponto incial, chegar a outro (ponto final), passando por todos os pontos.

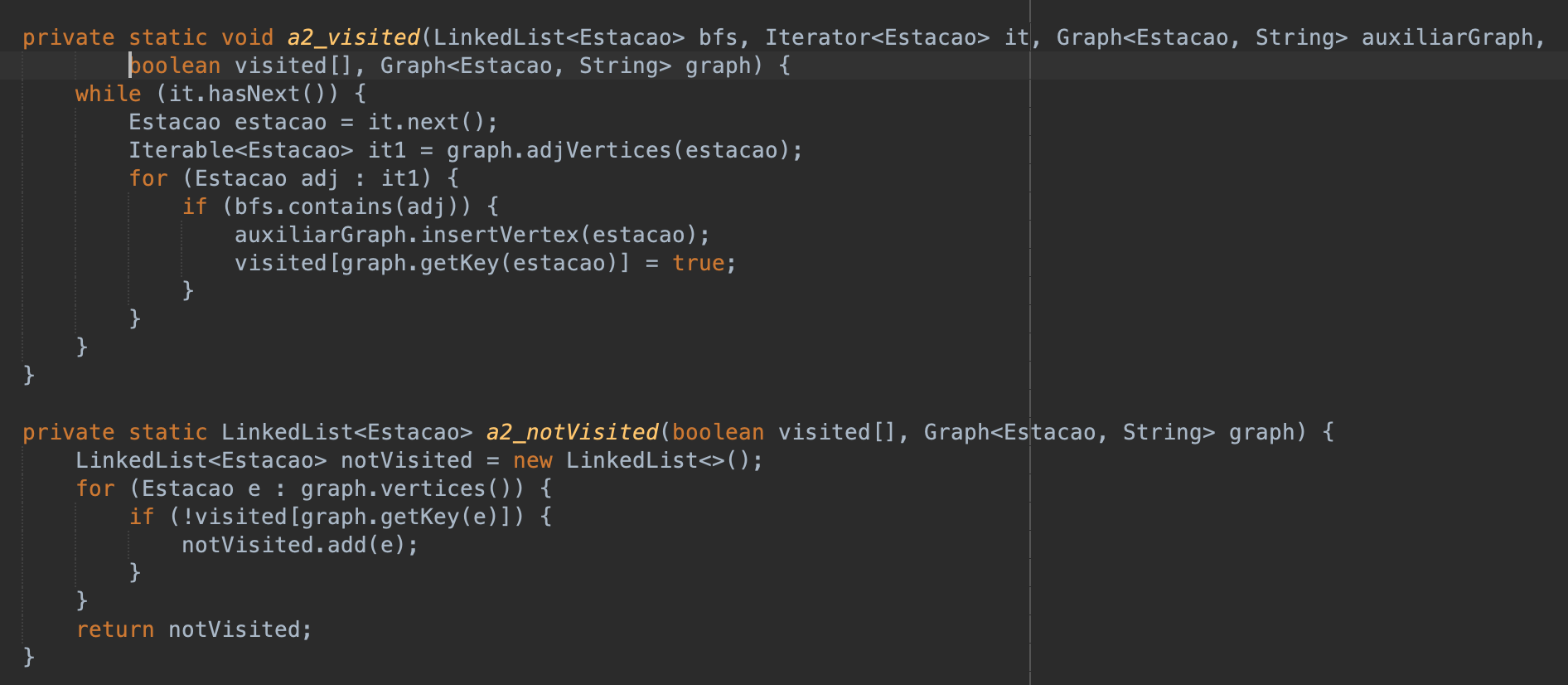
No método a2\_Conexo(método principal), obtemos a lista de estacoes (LinkedList) através do algoritmo BreadthFirstSearch, as quais consigamos chegar partindo do ponto inicial (obtido através do grafo criado, ( vamos buscar todos as keys,ou seja, todas as estações onde a primeira estação se encontra na posição zero do vetor Estacao-- Estacao[] keysEstacao = metro.getMetro().allkeyVerts() Estacao e = keysEstacao[0];).

Seguidamente colocamos num vetor de booleanos (boolean[] visited) se os vértices(Estacoes) já tinham sido ou não visitadas(a2\_visited), caso tenha sido visitada inserimos num grafo auxiliar que posteriormente será adicionado a uma list contendo os grafos criados, ou seja, as varias compontes dos grafo.

Posteriormente, vamos buscar as estacoes não visitadas(a2\_notVisited) e repetimos o processo, a primeira estacao na linkedList notVisited e utilizamos o BreadthFirstSearch para obter as ligações a essa estacao.e vemos os visitados e acrescescentamos essa estacao ao grafo e no final esse grafo sera a dicionado a list compontees

Caso o tamanho da list compontes seja 1 (apenas contem um grafo) ou se o a linkedList resultante do BreadthFirstSearch contiver todos os vértices, o grafo que contem as estacoes e fortemente conexo, caso contrário lua lista de grafos é retornada.



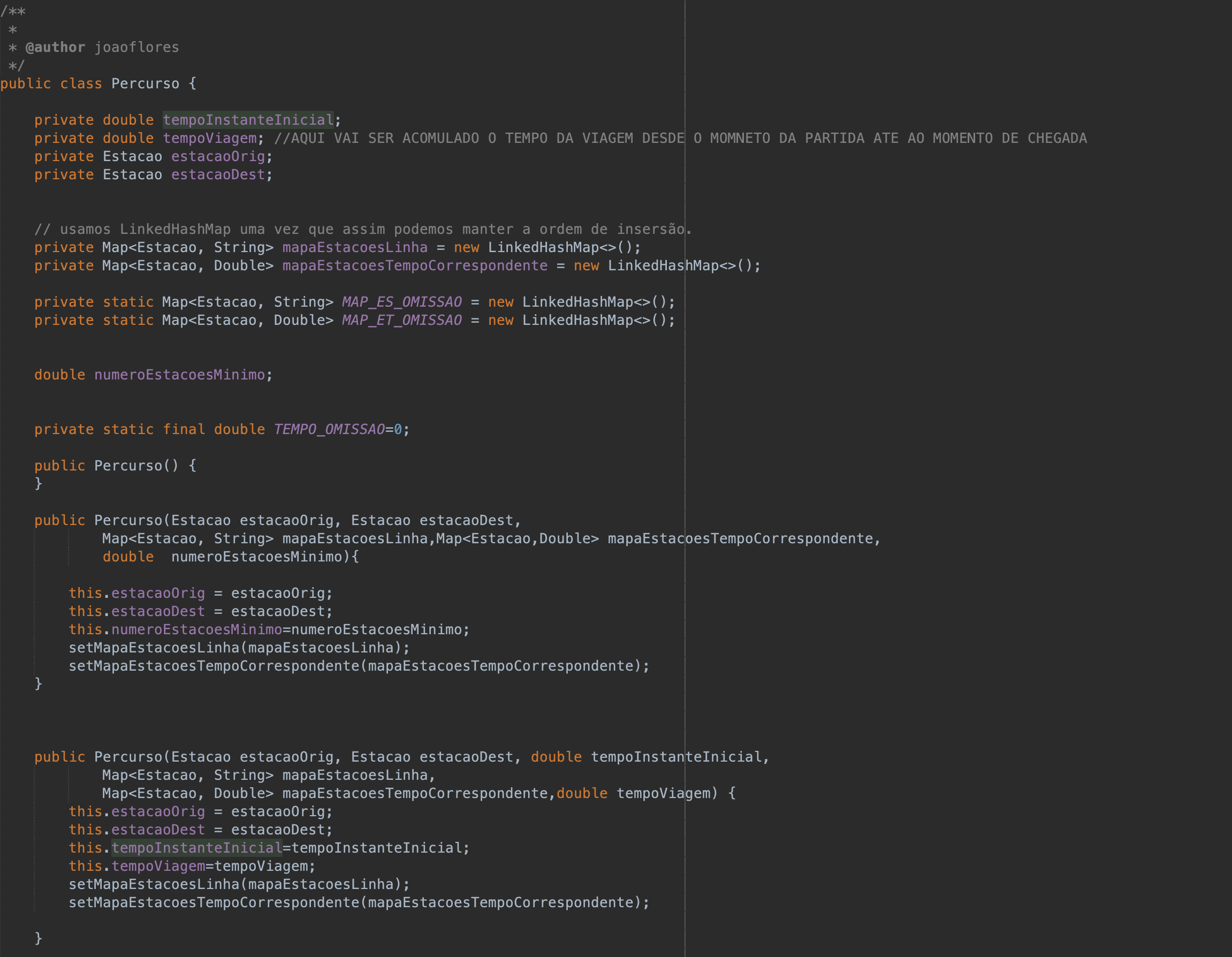


Dada uma estação origem e uma estação destino encontrar o caminho mínimo entre estas quanto ao número de estações, devolvendo uma instância da classe Percurso.

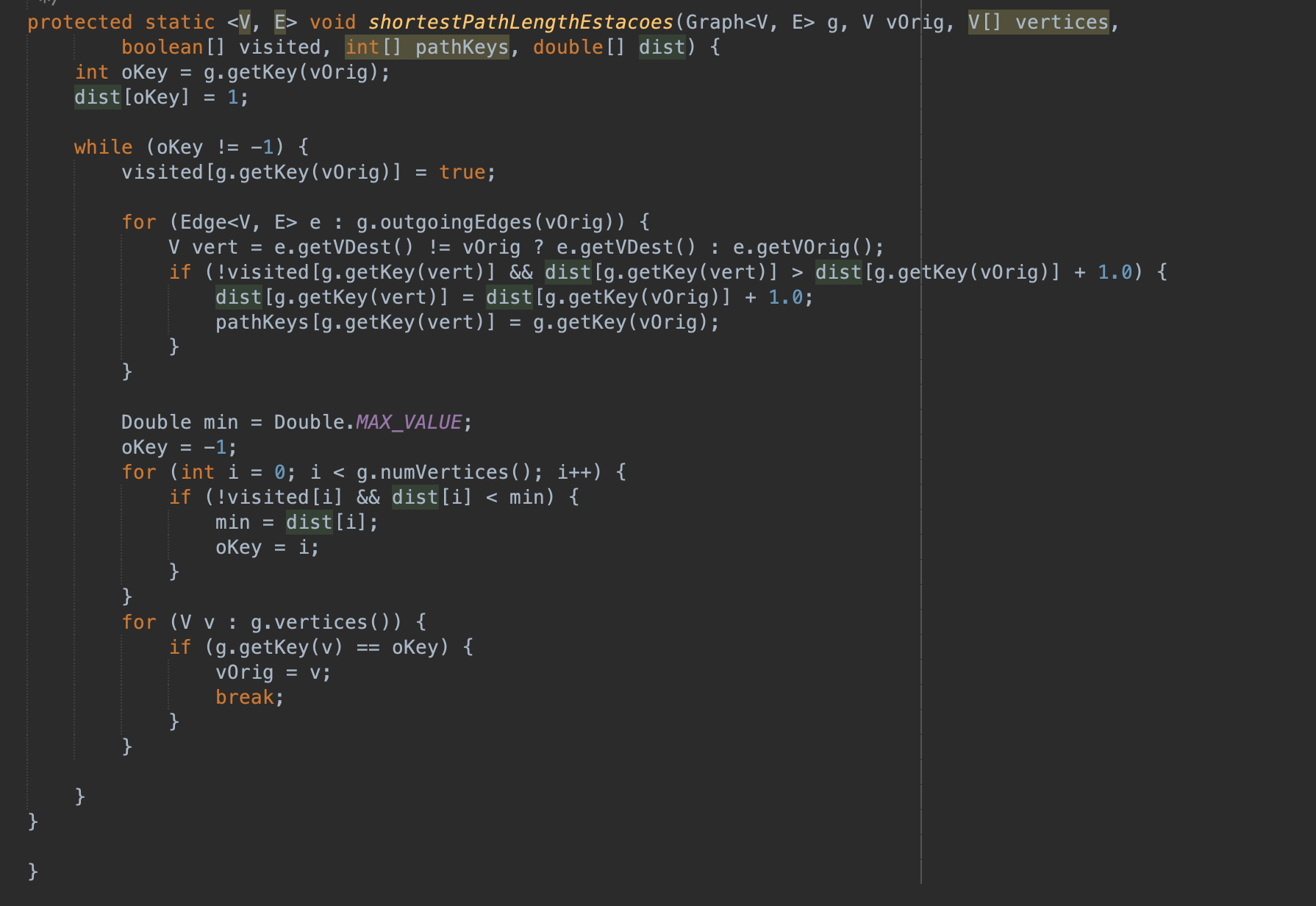
Para o desenvolvimento deste requisito, necessitamos de uma instância Percurso. Esta classe contém toda a informação relativa a um percurso incluindo as linhas, as estações

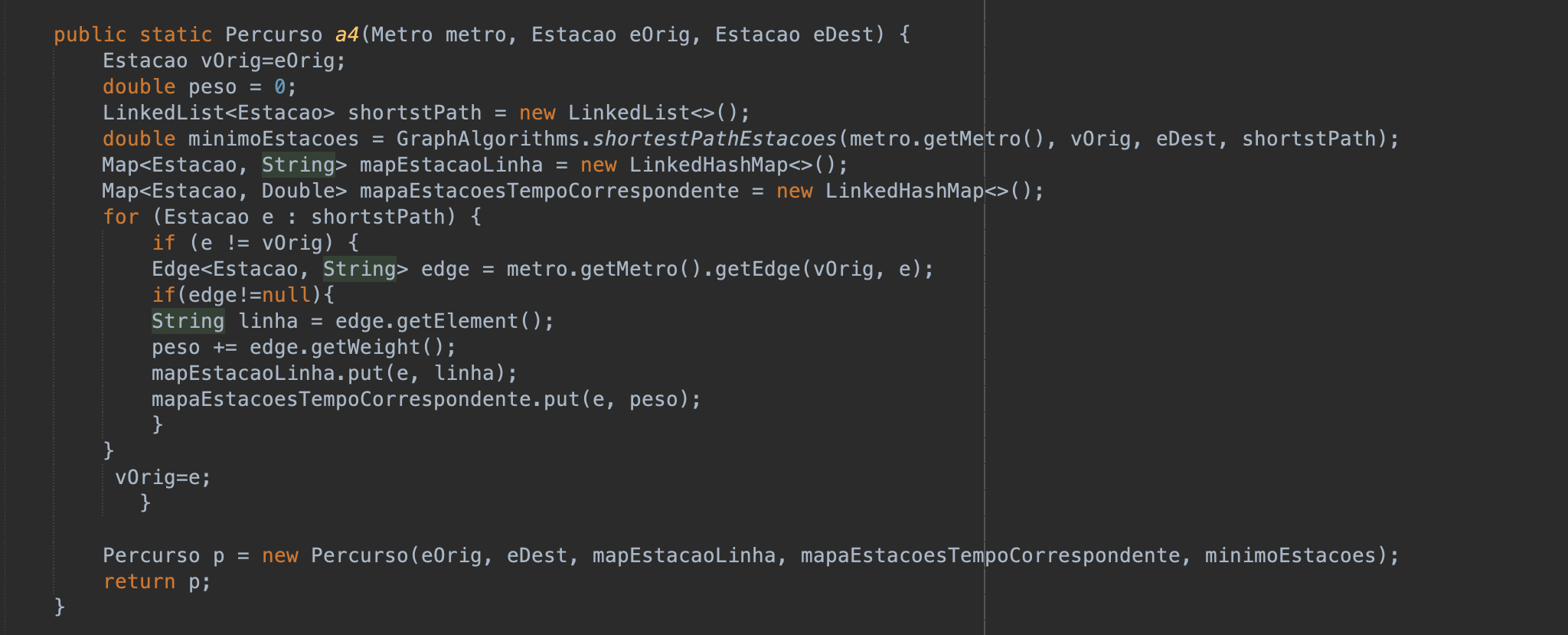
percorridas em cada linha e o instante em que passa em cada estação, dado um instante inicial do percurso.

Para tal a classe percurso tem um private Map<Estacao, String> mapaEstacoesLinha que contem as estacoes e as respetivas linhas, private Map<Estacao, Double> mapaEstacoesTempoCorrespondente que contem as estacoes e o tempo que demora a chegar a cada estacao que tem ligação, private double tempoInstanteInicial, private double tempoViagem; a estação origem, estacao de origem e o numero mínimo de estacoes.



stock\_count. Tendo a classe Percurso desenvolvida e testada, implementados o requisito solicitado na alínea 4).

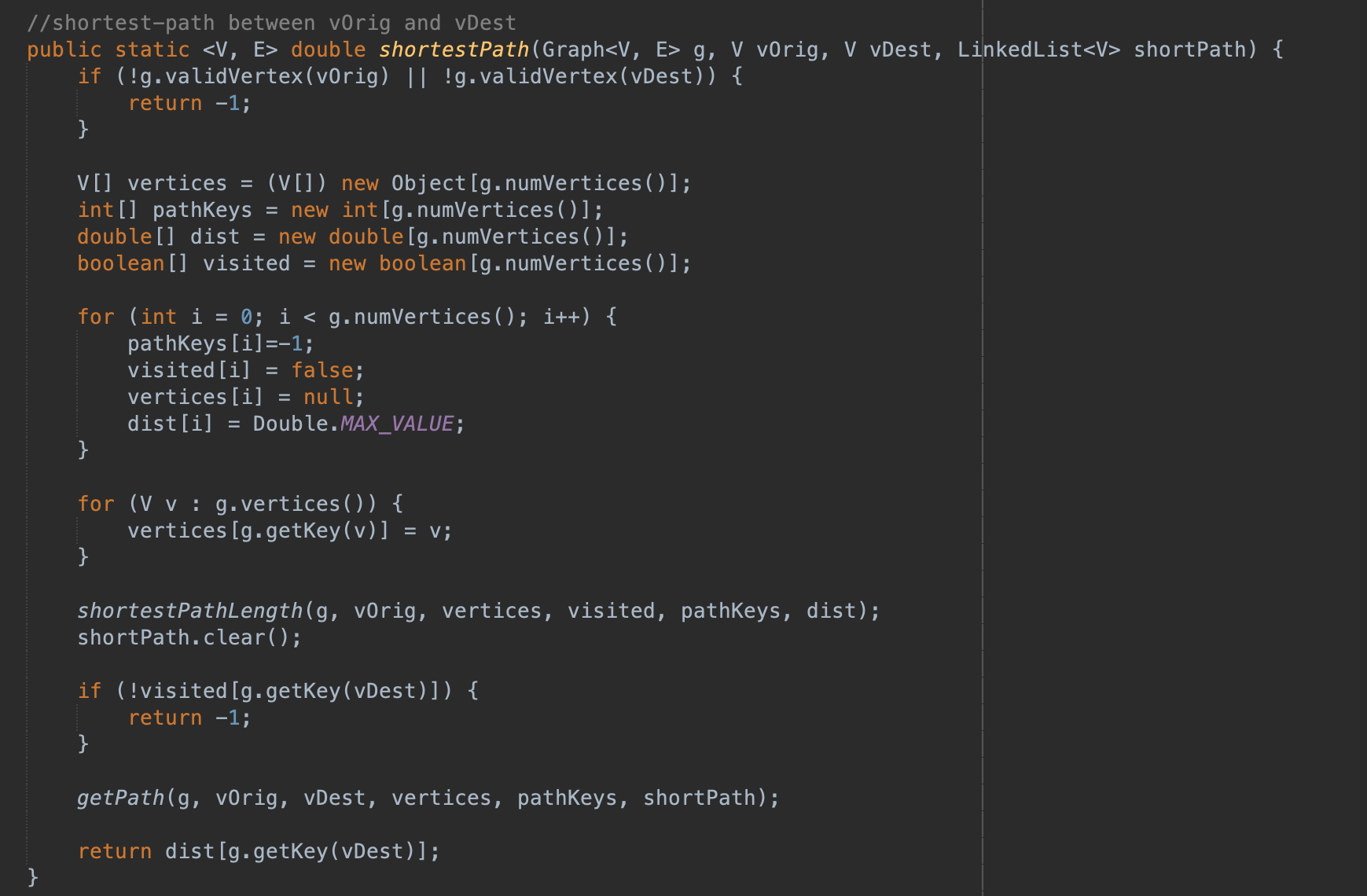
Foi recreado o método da classe GraphAlgorithms shortestPath a fim de retornar o número de estações mínima. Assim para esta alínea através do método shortestPathEstacoes obtemos o numero mínimo de estacoes a percorrer desde a estacao de origem até a estacao de destino.

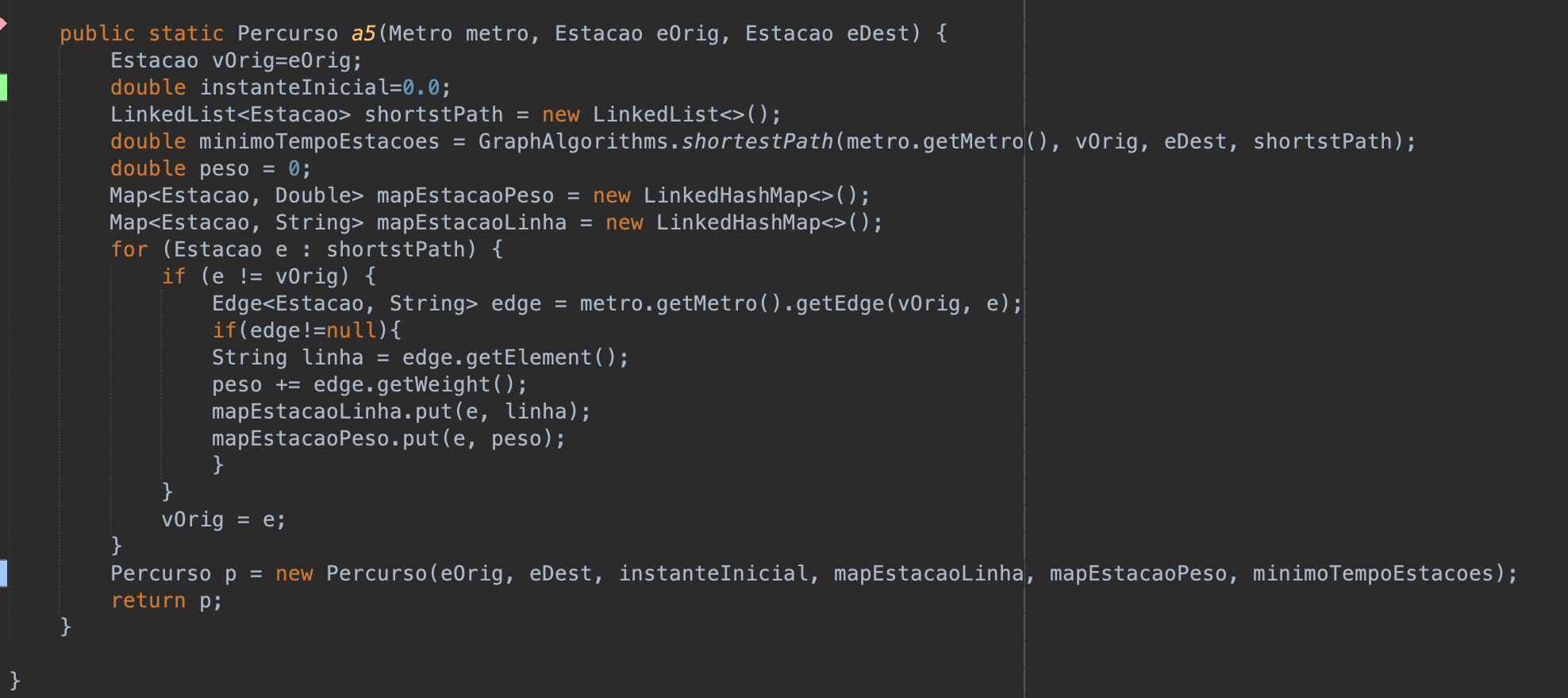


Dada uma estação origem, uma estação destino e um instante inicial encontrar o caminho mínimo entre

estas quanto ao tempo a percorrer, devolvendo uma instância da classe Percurso.

Este requisito é muito semelhante ao anterior. Apenas difere no facto de que o shortestPath retorna o custo mínimo,neste contexto retorna o tempo mínimo desde a estacao inicial ao destino.





Para finalizar preenchemos os mapas com a informação relativa as estacoes e a respetiva linha, bem como as estacoes e o tempo que a separa da próxima estacao. A linha conseguimos obter através do edge.getElement() e o peso(tempo que demora a chegar a próxima estacao) através do edge.getWeigth().

Por fim criamos a instância do percurso que contem a estação origem e destino, ,o instante inicial(0.0 segundos ), o mapa com as estações e a linha, o mapa com as estações e o tempo até a próxima estacao e o tempo mínimo a percorrer desde a estacai origem a destino .

Dada uma estação origem, uma estação destino, um instante inicial e um conjunto de outras estações intermédias, encontrar o caminho mínimo, quanto ao tempo a percorrer, entre a estação inicial e a final que passe obrigatoriamente por todas as estações intermédias, devolvendo uma instância da classe Percurso.

CaminhoMinimoComEstacoesIntermedias

Neste método é necessário uma estação de origem e destino um tempo inicial, uma lista com as estações que o utente necessita de passar e o grafo. A obtenção de todos os caminhos com as estações de início e fim posteriormente é necessário obter qual dos caminhos anteriores tem o peso menor, para tal é utilizado o método tempoMenor e getListaTemporaria que em conjunto vão retornar a lista que tem menor peso e também todas as estações necessárias, se não existir nenhuma com as estações intermedias o que o método retorna é uma lista vazia.

Dado que o retorno é um percurso é criada uma instância do percurso com o tempo inicial, estação origem, estação destino, o tempo de viagem total e os mapas com os tempos e linhas que utiliza os métodos colocarValoresMapTempo e colocarValoresMapLinhas mais abaixo explicados que vão ordenar ambos os mapas conforme os valores da lista.

ColocarValoresMapTempo

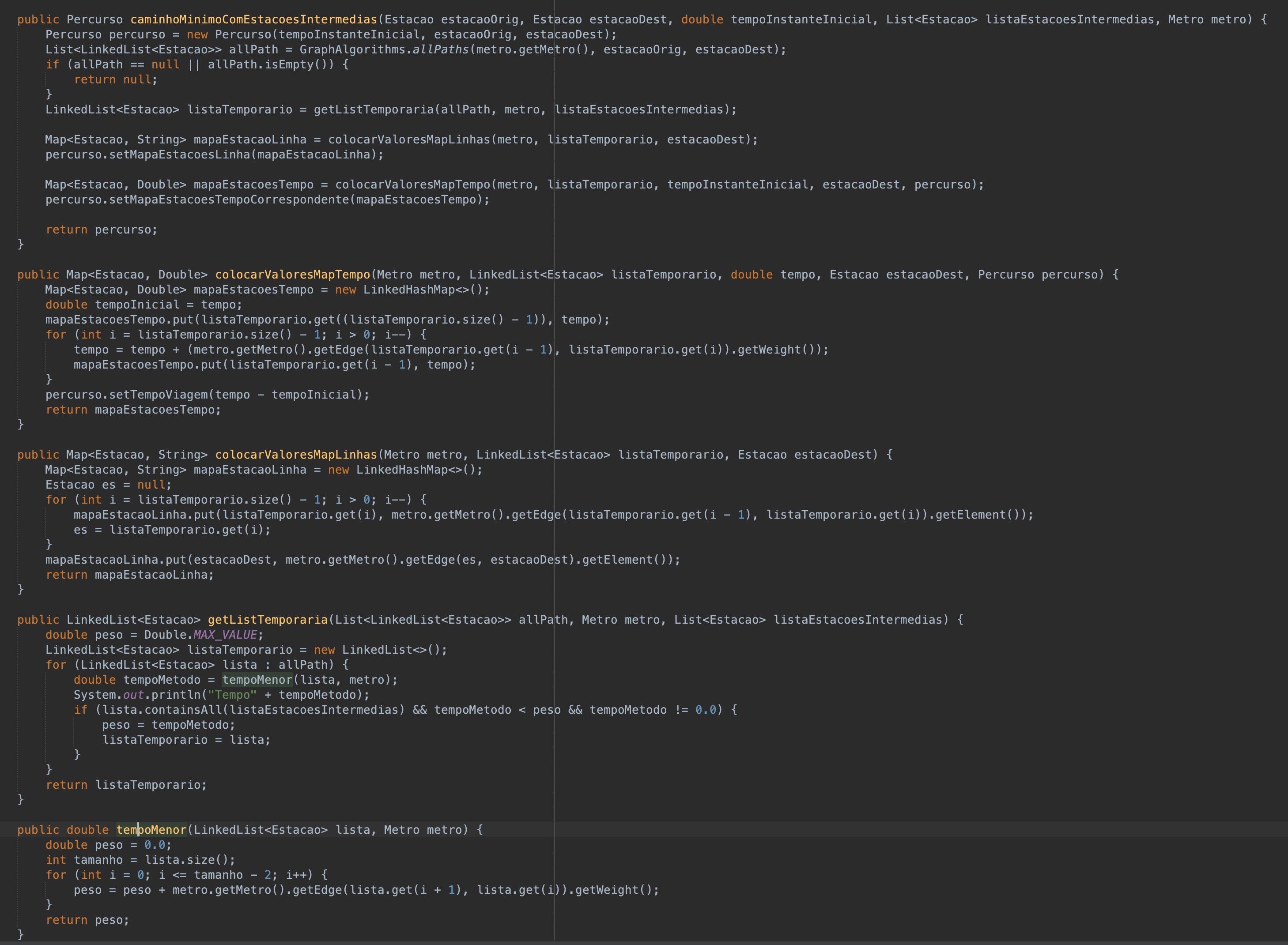
A uma lista colocar as estações e o tempo em que se encontrava nessa respetiva estação.

Um ciclo for para obter os edges para poder obter o peso de uma estação até outra e adicionar ao peso que já vinha das estações anteriores.

Finalmente fazer a diferença entre o início e o fim e fazer colocar esse valor no atributo tempo Viagem.

ColocarValoresMapLinha

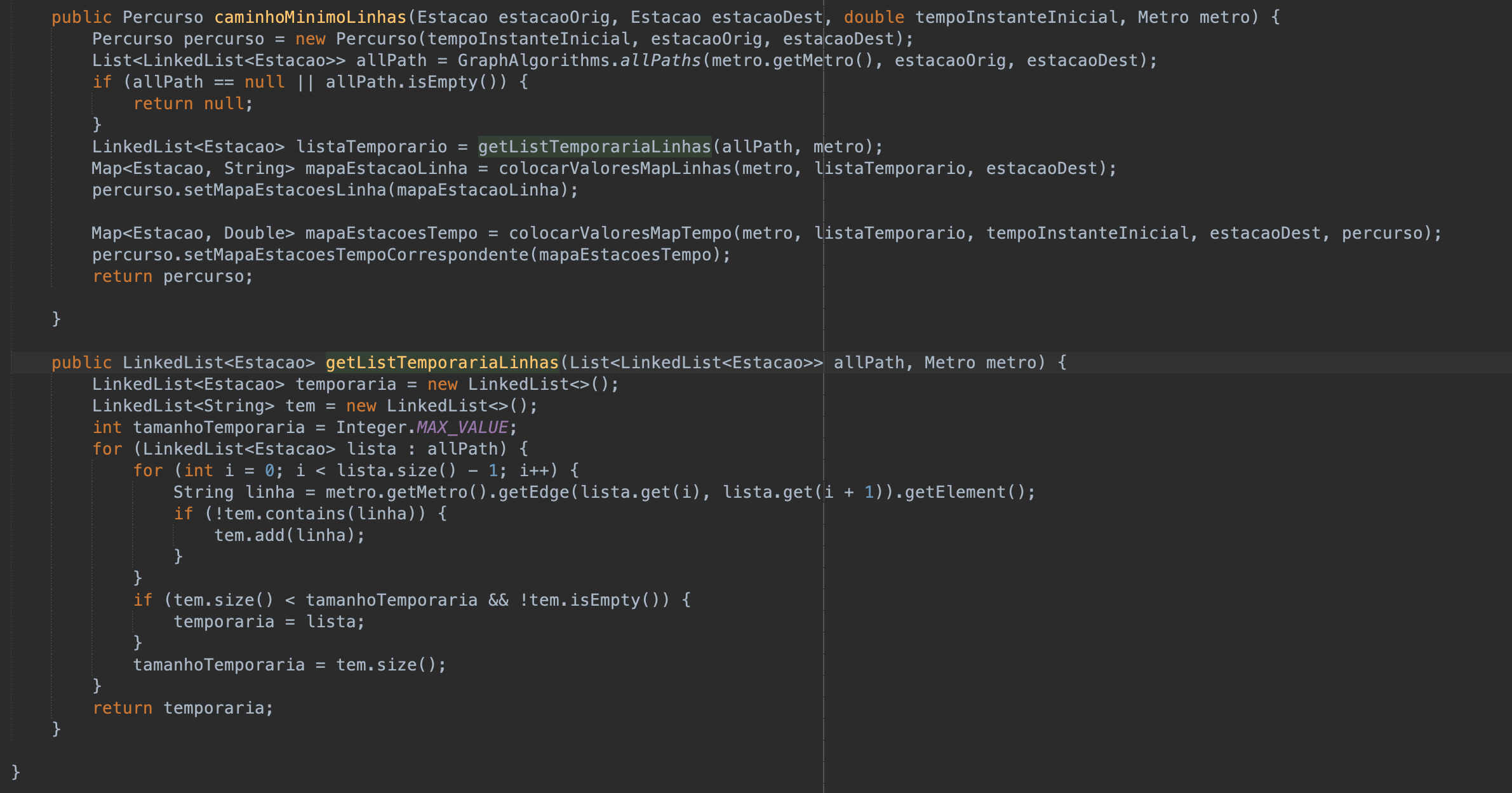
Mesma logica que o método acima explicado com só que a colocação no map em vez de ser tempo é uma String para indicar a linha.



Dada uma estação origem e uma estação destino encontrar o caminho mínimo entre estas quanto ao número de mudanças de linha, devolvendo uma instância da classe Percurso.

CaminhoMinimosLinhas

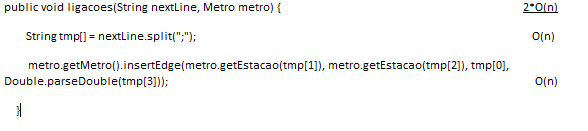
Mesmo raciocínio que o método anterior a variante é o getListTemporariaLinhas que em vez de retornar as com peso menor retorna a lista que passa por menos linhas.

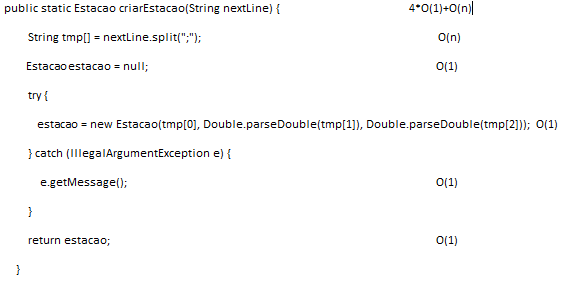


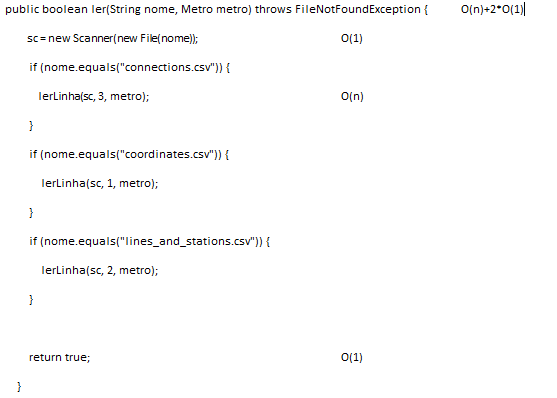
Analise da Complexidade

Alinea 1)

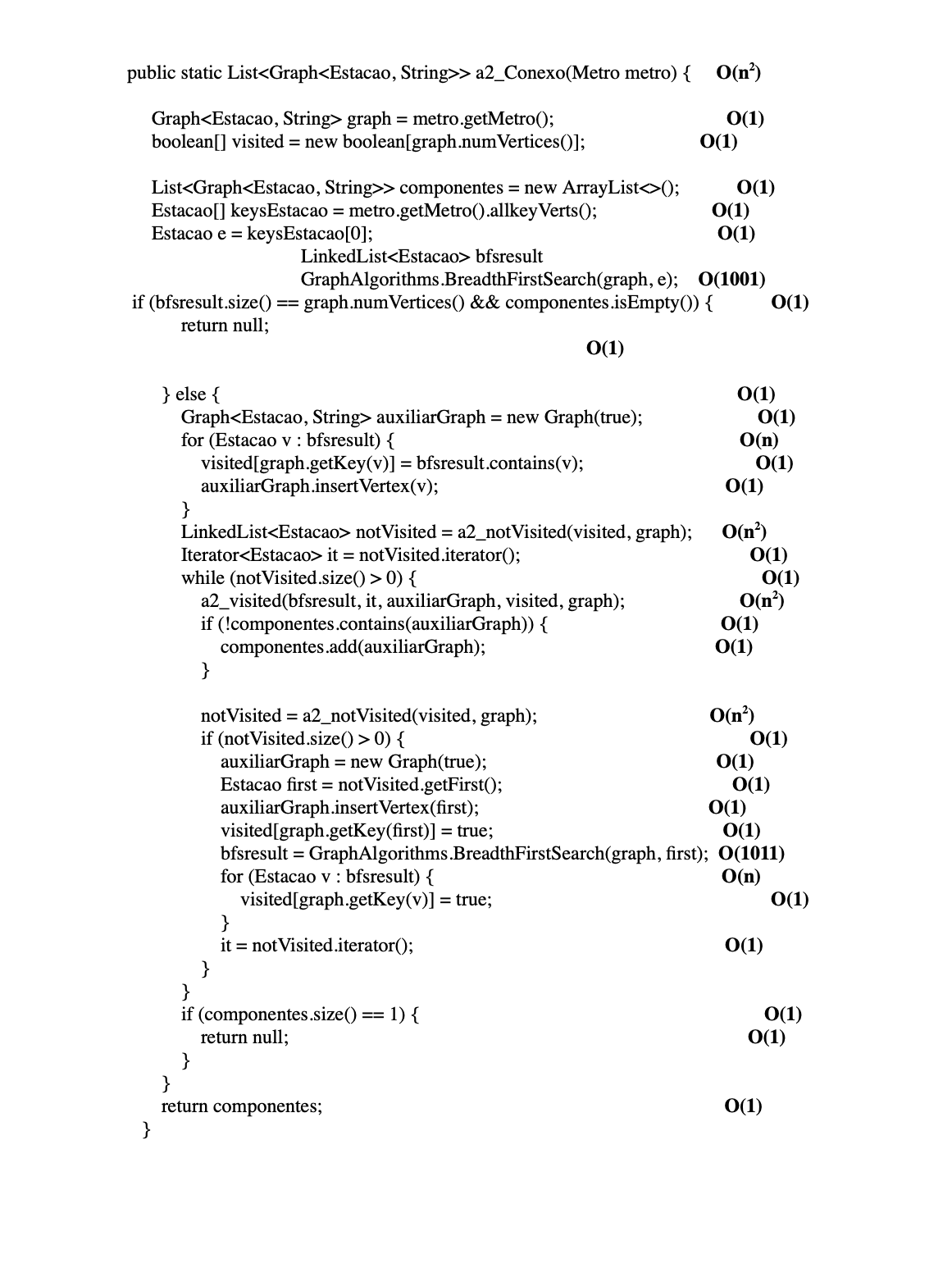


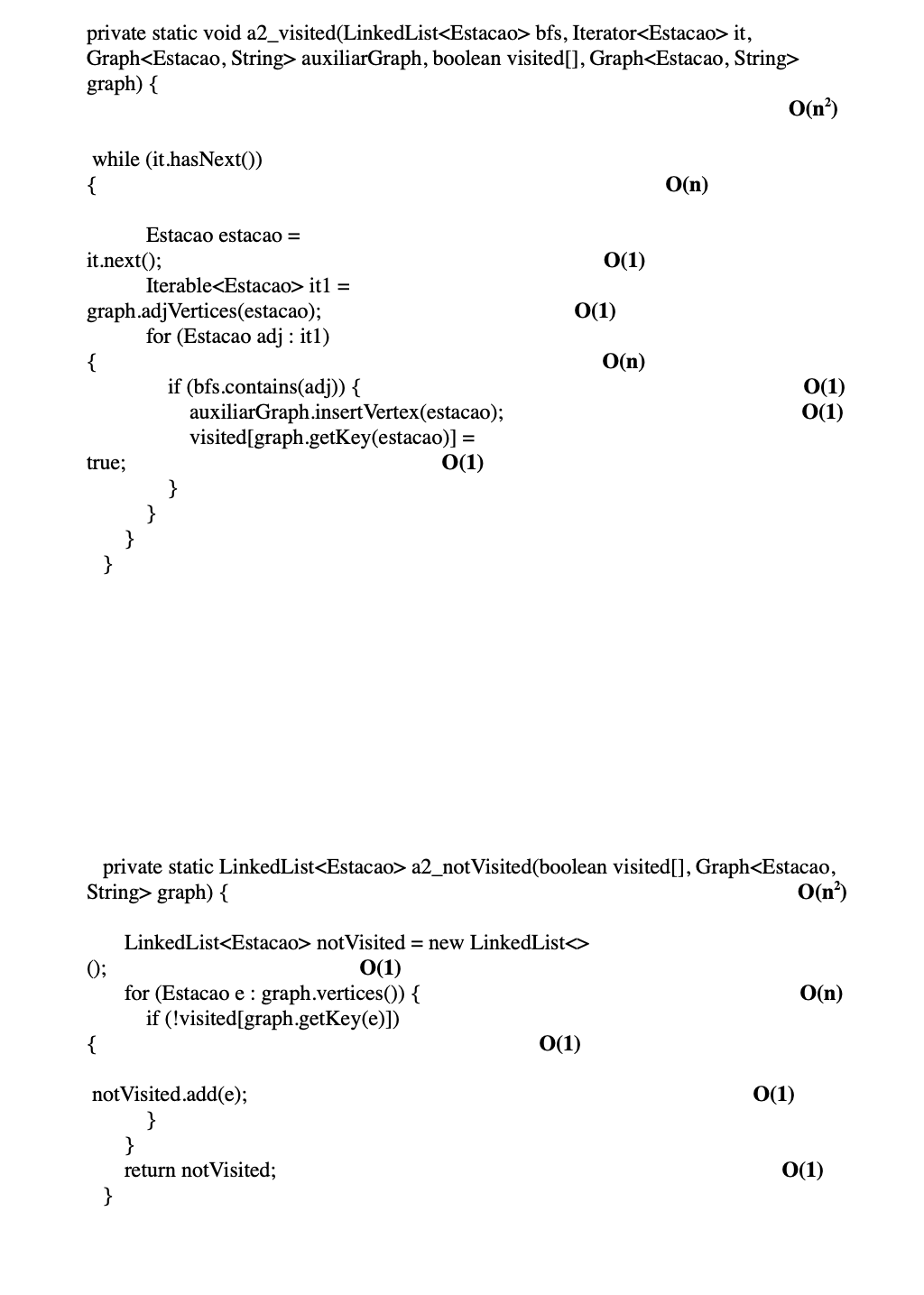




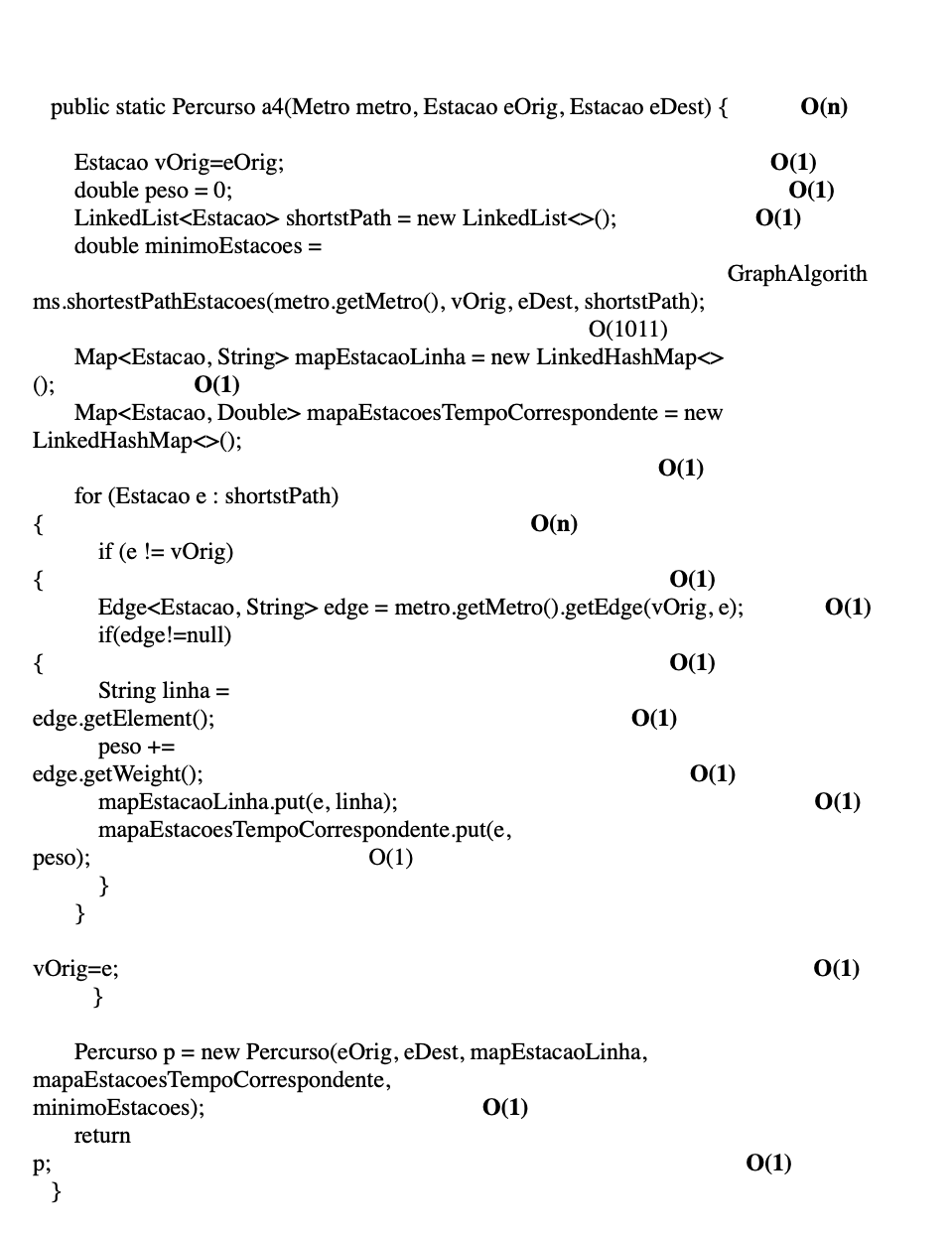


Alinea 2)

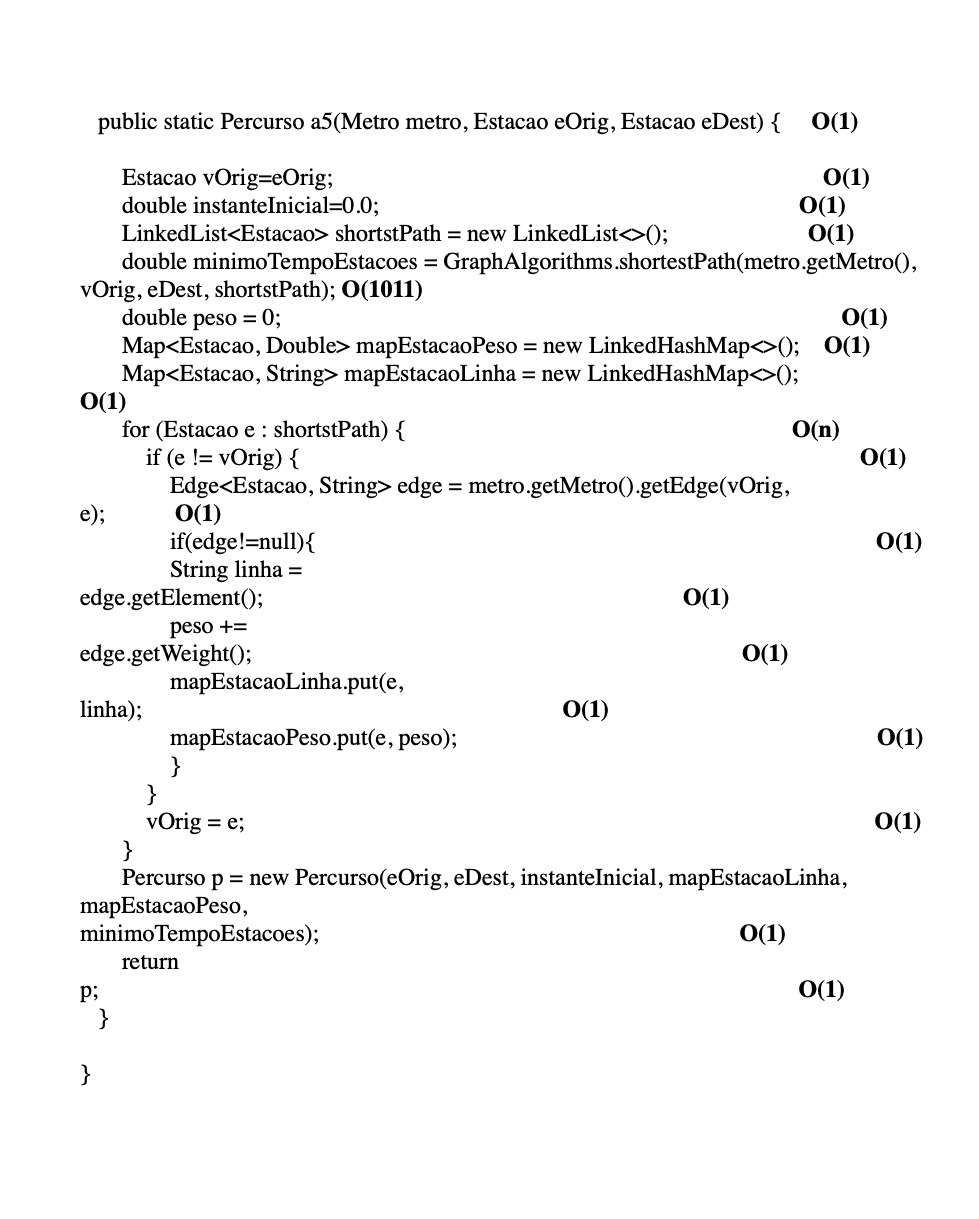


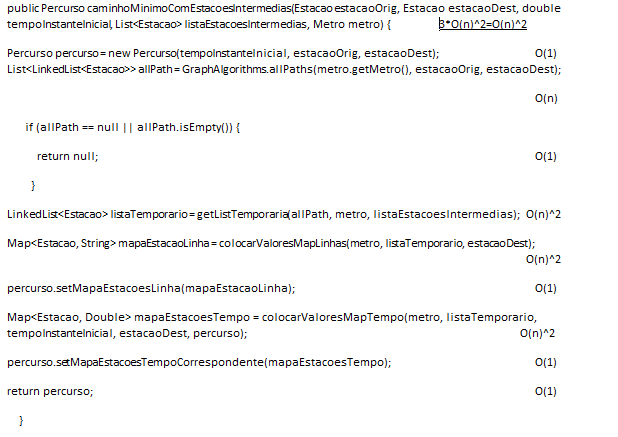


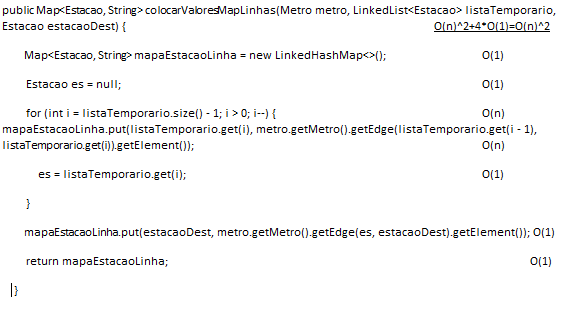
Alinea 4)

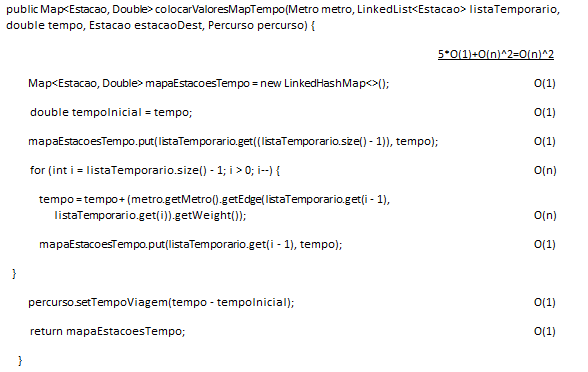


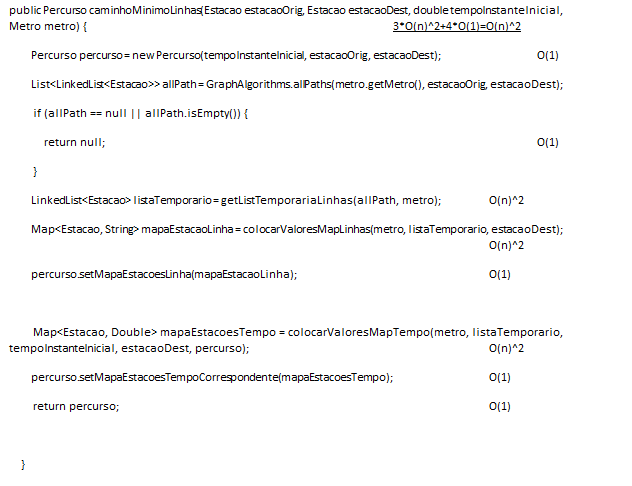
Alinea 5)

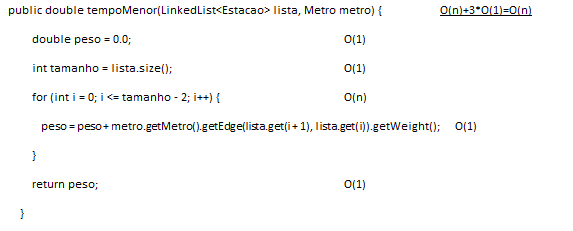


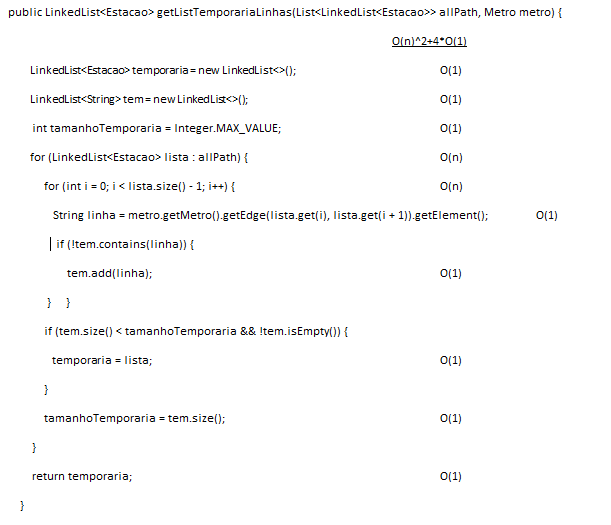
Alinea 6) e 7)

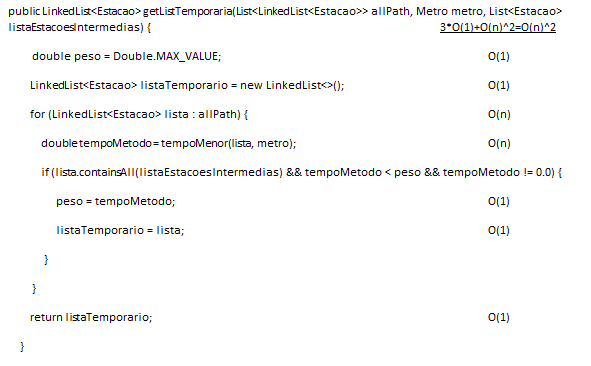












Possiveis Melhoramentos:

Um aspeto a melhorar em futuros trabalhos é a implementação de estruturas mais otimizadas e eficientes.