A black background with a black square

Description automatically generated with medium confidence**Relatório**

**Estruturas de**

**Informação**

**Sprint 3**

Trabalho Realizado por:

* 1221194 – Diana Neves
* 1220913 – Marco Ferreira
* 1220917 – Tomás Gonçalves
* 1221330 – Francisco Monteiro

**Diagrama de Classes**

A screenshot of a computer screen

Description automatically generated

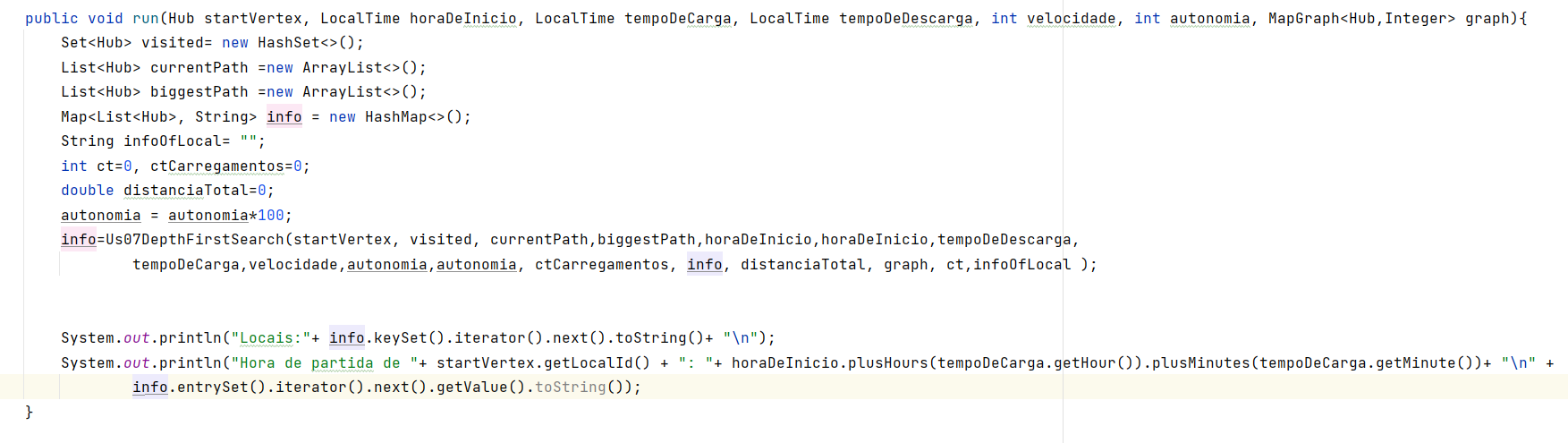
**Análise dos Métodos Implementados nas**

**User Stories :**

1. USEI07

1.1 run()

O método começa por inicializar estruturas de dados (Set, Map e duas Lists). Essas estruturas serão usadas para rastrear informações sobre os locais visitados, o caminho percorrido, o maior caminho encontrado e detalhes dos locais. A autonomia é ajustada multiplicando o valor original por 100. O método invoca um algoritmo de busca em profundidade (depth first search) para explorar o grafo, passando vários parâmetros relevantes para a busca. Essa função busca informações sobre rotas possíveis, considerando restrições de tempo, autonomia do veículo e distâncias. Após a execução do algoritmo de busca, o método imprime na saída informações relevantes, como os locais visitados e a hora de partida do ponto inicial, juntamente com informações detalhadas necessárias sobre o caminho percorrido.



1.2 *Us07DepthFirstSearch()*

Este método recebe vários parâmetros como o vértice atual, o vértice visitado os caminhos (o maior e o atual), informações de tempo como a hora atual, total, o tempo de descarga e de carga, a velocidade e a autonomia (atual e total).

* Marca o vértice atual como visitado e o adiciona ao caminho atual.
* Verifica se há arestas viáveis para percorrer sem recarregar o veículo, considerando a autonomia atual.
* Se não houver arestas viáveis sem recarregar, recarrega o veículo, atualiza a autonomia e o contador de carregamentos.
* Itera sobre os vizinhos do vértice atual, e calcula o tempo necessário para chegar a esses vizinhos com base na distância e velocidade do veículo.
* Verifica se é possível visitar o vizinho respeitando as restrições de tempo e autonomia.
* Se não existirem mais opções para explorar a partir desse vértice atual, remove-o do caminho atual.

No fim, é removido o vértice atual dos vértices visitados e retorna a informação, num mapa que contém a informação necessária a dar return (informações acerca dos caminhos).

Em termos de complexidade, este método depende do número de vértices (V) e arestas (E) no grafo dado como entrada. Aqui estão alguns pontos relevantes:

* O primeiro loop **for** que verifica a existência de arestas viáveis tem uma complexidade de O(E), pois percorre todas as arestas do grafo.
* O segundo loop **for**, iterando sobre os vizinhos do vértice atual, tem uma complexidade de O(V), pois percorre os vértices adjacentes ao vértice atual.

Portanto, a complexidade total do algoritmo, considerando ambos os loops, pode ser aproximadamente O(V + E), dependendo da estrutura e densidade do grafo.

No relatório, é crucial destacar a utilidade deste método na consideração de restrições de tempo, autonomia e distância entre locais. Além disso, é importante mencionar que a complexidade pode variar dependendo da estrutura do grafo e das restrições específicas consideradas durante a busca.







2.USEI08

*2.1 printCircuit()*

Método que pede ao usuários os dados necessários à execução do caminho do circuito, nomeadamente o número de *Hubs* (*nHubs*) que deseja visitar e o *Hub* de origem. Verifica se o *hub* de origem é válido e imprime as informações detalhadas sobre o mesmo (através do método *printHub(origin)).* Por fim apresenta o caminho do circuito (ida e volta), a distância total do percurso, o tempo total do mesmo e o número de carregamentos necessários a fazer para concluir o circuito apresentado.

A screenshot of a computer program

Description automatically generated

* 1. *obterHubOrigin()*

Primeiramente exibe uma lista de *hubs* disponíveis e solicita ao usuário que selecione um *hub* de origem. Conforme o input do utilizador, encontra o *hub* correspondente ao id inserido, se possível. Por fim obtém informações adicionais como a autonomia do veículo, necessária mais tarde para o cálculo do número de carregamentos.

A screenshot of a computer program

Description automatically generated

* 1. *caminhoIda(Hub origin)*

Calcula o caminho de ida, considerando o número de *hubs* desejados. Para tal utiliza o método *edgeMaisColabs*() para selecionais quais os *hubs* subsequentes conforme o número de colaboradores. Atualiza a distância percorrida somando os pesos de cada aresta atravessada à distância total (*totalDist*).

A screenshot of a computer code

Description automatically generated

* 1. *caminhoVolta(Hub previousLocation)*

Calcula o caminho de volta, usando o algoritmo “*shortestPath*” que encontra o menor caminho possível entre dois *hubs* e atualiza a distância percorrida.

A screenshot of a computer program

Description automatically generated

* 1. *edgeMaisColabs(Hub startLocation, Map<Hub, Boolean> isHub)*

Encontra o *hub* adjacente com o maior número de colaboradores, evitando *hubs* já visitados.

A screenshot of a computer code

Description automatically generated

* 1. *calcDistancia(Hub hub1, Hub hub2)*

Calcula a distância entre dois *hubs* através do peso da aresta entre os mesmos

A screenshot of a computer program

Description automatically generated

* 1. *calcTempoCircuito (double distance)*

Converte a distância total em tempo estimado, expresso em horas.

A close up of words

Description automatically generated

* 1. *checkHub(Hub hub)*

Verifica se um *hub* é válido ou não e, caso seja, apresenta as suas informações detalhadas. Caso não seja, informa que esse *hub* não é válido

A computer screen shot of a code

Description automatically generated

* 1. *calcNumeroCarregadores (int autonomy)*

Calcula o número de carregamentos necessários para percorrer o circuito com base na autonomia do veículo.

A screenshot of a computer code

Description automatically generated

* 1. *printHub(Hub hub)*

Exibe as informações detalhadas de um *hub* específico (passado por parâmetro), nomeadamente ID, coordenadas e horários de funcionamento.

A computer screen shot of a program code

Description automatically generated

1. USEI09
   1. hubForLocation()

O método hubForLocation é responsável por organizar e mostrar as localidades do grafo em N clusters que garantam apenas 1 hub por cluster de localidades. Realiza através do agrupamento de hubs(nós) em clusters através do método clusterize() e posteriormente realizar o print de clusters , caso não seja null ,através do método displayClusters, indicando o hub e as localidades de cada cluster.

Uma imagem com texto, Tipo de letra, file, captura de ecrã

Descrição gerada automaticamente

* 1. clusterize(MapGraph<Hub , Integer> graph)

O método clusterize é responsável pelo processo de agrupamento de hubs(nós) em clusters, com apenas um hub por cluster.Inicialmente, identifica os hubs potenciais, promovidos na USEI02, e inicia os clusters correspondentes. De seguida, calcula os caminhos mais curtos entre os hubs e as localidades no grafo. Valida os clusters criados, ajustando-os para garantir que cada cluster tenha pelo menos 1 hub, através da remoção temporária das arestas do grafo para redistribuir os hubs.

A sua complexidade é de O(n^2) devido as iterações e operações de validação realizadas sobre o grafo e os clusters.

Uma imagem com texto, captura de ecrã, documento

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

* 1. determineShortestPaths(MapGraph<Hub , Integer> graph)

O método determineShortestPaths calcula os caminhos mais curtos a partir de cada hub no grafo para todas as outras localidades, através do algoritmo de Djiskstra, armazenando esses caminhos num mapa.

A sua complexidade é O(V\*(V+E)\*log(V)), onde V é o número de vértices no grafo e ‘E’ é p número de arestas.

Uma imagem com texto, Tipo de letra, file, captura de ecrã

Descrição gerada automaticamente

* 1. getEdgeUsageMap(MapGraph<Hub, Integer>graph, Map<Hub, List<LinkedList<Hub>>> allPaths)

O método getEdgeUsageMap cria um mapa que conta o uso de cada aresta no grafo, com base nos caminhos fornecidos de cada hub.

A sua complexidade é de O(n\*m), onde n é o número de caminhos e m é o número total de arestas nos caminhos.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

* 1. selectEdgeToRemove(Map<Edge<Hub, Integer>,Integer> edgeUsageCounter)

O método selectEdgeToRemove é responsável por identificar e selecionar uma aresta para remoção com base no seu uso, conforme registrado no mapa edgeUsageCounter.

A sua complexidade é de O(n) onde n é o número de arestas no mapa edgeUsageCounter.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

* 1. validateClusters( Map< Hub , List<LinkedList>>> allPaths)

O método validateClusters realiza a validação dos clusters gerados com base nos caminhos mais curtos obtidos a partir de hubs.

A sua complexidade é de O(n^2), onde n é o número de hubs/clusters.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

* 1. displayClusters(Map<Hub,Set<Hub>> clusters)

O método displayClusters tem como objetivo mostrar os clusters e os seus respetivos hubs e localidades.

A sua complexidade é O(n) onde n é o número de clusters.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

1. USEI06
   1. execute()

O método execute realiza operações para encontrar caminhos entre uma localização e um hub, levendo em consideração a autonomia de um veículo.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

* 1. writePath(LinkedList<Hub> path, int distance, int time)

O método writePath tem finalidade de exibir informações sobre um caminho entre locais/hubs, incluindo a origem, o caminho percorrido, a distancia entre os locais, e o tempo total.

A sua complexidade é O(n), onde n é o número de elementos na lista path.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

* 1. calculateDistance (LinkedList<Hub> path)

O método calculateDistance recebe uma lista encadeada de locais/hubs e calcula a distância total percorrida ao longo desse caminho, somando os pesos das arestas entre os locais adjacentes e retorna também a distância total dividida por 100.

A sua complexidade é O(n), onde n é o número de elementos na lista path.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

* 1. seperateLocalsAndHubs (List<Hub> locals , List<Hub> hubs)

O método seperateLocalsAndHubs percorre todos os vertices presents no graph locations e separa esses vertices em duas listas diferentes: um para os hubs e outra para os locais, baseando-se na propriedade isPromoted() de cada hub.

A sua complexidade é O(n) , onde n é o número de vertices , já que o loop percorre todos os vertices uma vez e adiciona cada vértice a uma das listas locals ou hubs.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

1. USEI11
   1. importFile (String filename)

O método importFile faz a leitura de um arquivo para importer dados de locais/hubs a partir das linhas do arquivo.

A sua complexidade é de O(m\*n), onde m é o número de linhas no arquivo e n é o número de vertices no grafo.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente