ESCOLA POLITÉCNICA

Programa IT Academy – Processo Seletivo – Edição #18

Nome Completo: Ana Carolina de Oliveira Xavier E-mail: anacarolina.oxavier@gmail.com

Etapa 1 – Questões de lógica

Esta seleção possui 15 questões de lógica de caráter eliminatório. As questões são apresentadas no formulário de Exercício Técnico e devem ser respondidas no próprio formulário online, que deverá ser acessado através do link a seguir: https://forms.gle/yZtVcv1b5fCgScLBA

Etapa 2

RESUMO DA SOLUÇÃO

Para realizar o desenvolvimento do teste técnico deste processo seletivo, utilizei a linguagem de programação Java e para a interface do programa optei por textual/console. Como ferramenta de desenvolvimento (IDE) utilizei o IntelliJ.

O desenvolvimento do código foi realizado pensando em seis (6) classes, sendo elas:

Classe "**Main()**": Esta classe é responsável pelo contato direto com o usuário e a classe Agencia(). Na Main() está o menu principal, esse que controla o funcionamento principal do programa, a partir dele é possível acessar as funcionalidades: "Consulta de trechos e modalidades", "Cadastrar transporte", "Dados estatístico" e "Sair do programa", quando se está dentro deste menu é possível sair do programa a qualquer momento assim como escolher qualquer uma das outras funcionalidades. Na classe Main() também há três (3) métodos extras: um método que sempre que é chamado imprime as opções do menu principal e permite que o usuário informe qual é sua opção (através do número indicativo), e dois métodos apenas para a impressão dos nomes das cidades e dos exemplos de itens, para que assim o código não fique desnecessariamente maior.

Classe "**Agencia()**": Esta classe é responsável pelo funcionamento dos métodos que satisfazem as funcionalidades do programa. O principal objetivo desta classe é evitar possíveis repetições no código, criando métodos para cada necessidade do programa. A classe se comunica diretamente com as demais classes, e não permitindo o contato direto do usuário com todos os métodos. Nesta classe é realizada a leitura do arquivo .csv, a partir da leitura do arquivo são criadas cidades (objetos da classe Cidade()), onde o primeiro índice é o nome da cidade e os demais elementos são as distâncias entre as cidades, essas cidades são adicionadas em uma lista de cidades que pode ser posteriormente acessadas. Além do método de leitura do arquivo, há os principais métodos:

calcularTrecho(int indexSaida, int indexDestino, int modalidade): Este método satisfaz a primeira funcionalidade do programa, onde é informado por parâmetro o

número referente a cidade de origem e saída, junto com o número referente a modalidade de caminhão desejada pelo usuário para assim poder ser realizada a simulação da distância percorrida no trecho e o valor do transporte.

calculaDistanciaTotal(List<String> cidades): Este método é responsável por calcular a distancia total do trajeto do transporte a ser cadastrado, satisfazendo uma parte da segunda funcionalidade do programa. O método recebe por parâmetro uma lista de cidades informada pelo usuário, assim ele realiza o calculo do trecho a cada duas cidades, somando as distancia em uma variável que armazena a distância total do percurso. Este método utiliza o método "calculaTrecho()" para a realização do cálculo da distância total.

calculaDistancia(String cSaida, String cDestino): Seu principal objetivo é auxiliar o método para calcular a distancia total do percurso do usuário. Este método recebe por parâmetro duas cidades (de origem e de destino), a com o auxilio da lista de cidades (onde armazena os objetos Cidade) e o *array* de nomes de cidades – no array de *nomes de cidades,* além de informar os nomes das cidades existentes, os índices do vetor representam a posição da cidade na lista de cidades, por exemplo: se o nome da cidade BELEM estiver no índice [1], a distancia dessa cidade será encontrada no índice get(1) na lista de distancias das cidades – é realizado o cálculo da distância do trecho entre as duas cidades, além de armazenar esse trecho em uma lista de trechos para poder ser acessado posteriormente.

listaItens(): Este método é responsável por realizar a lista de itens que o usuário deseja transportar. O usuário pode continuar informando o nome e o peso do item desejado até informar a palavra "sair" para poder encerrar a lista. A cada item informado, este é adicionado ao mapa de itens do objeto itens. Ao final do método, este imprime os itens informados e informa o peso total da lista.

caminhaoAdequado(List<String> cidades): Este método é responsável por informar o caminhão adequado para que seja possível transportar os itens desejados pelo usuário. Neste método existe uma lista com os caminhões disponíveis para transporte e uma lista que armazenará os caminhões necessários para o transporte. A partir do conceito de repetição, é percorrido a lista de caminhões disponíveis e a cada caminhão é verificado se sua capacidade é maior ou igual ao peso da lista de itens. Caso este caminhão supra o peso dos itens, este é adicionado na lista de caminhões necessários e esse peso suprido é subtraído do peso total, para que assim seja realizado a verificação novamente com outro caminhão – essa verificação acontecerá até que todo o peso seja suprido. Após a implementação da lista de caminhões necessários, esta lista é percorrida para calcular o custo por quilometro de cada caminhão necessário pela distancia total. Ao final do método, é criado um novo objeto Transporte com todas as informações sobre o transporte cadastrados, e esse novo transporte é adicionado a uma lista de transporte que pode ser posteriormente acessada. Não foi possível satisfazer essa funcionalidade complemente devido as dificuldades encontradas em desenvolver um algoritmo que analisasse o melhor custo benefício de transporte.

dadosEstatisticos(): Este método imprime alguns dados estatísticos disponíveis no sistema. Não foi possível satisfazer totalmente essa funcionalidade devido a falta de acesso e existência de algumas variáveis.

Classe "**Cidade()**": Esta classe é responsável pela criação das cidades. As cidades são criadas a partir da leitura do arquivo .csv disponibilizado para o teste. A cada cidade criada é informado seu nome e uma lista contendo todas as distancias entre as demais cidades.

Esta classe além dos métodos getters, setters e toString, possui um método que retorna a distancia entre uma cidade informando o índice referente a cidade desejada.

Classe "**Modalidade()**": Esta é uma classe "enum", ou seja, uma classe onde declaramos valores constantes já definidos. Com ela é possível acessar as características das três modalidades de caminhões disponíveis. Esta classe possui métodos getters, setters e toString.

Classe "**Transporte()**": Esta classe é responsável pela criação dos cadastros de transportes realizado pelo usuário. Para cada transporte é informado o peso total dos itens, a lista de caminhões necessários, o valor total do percurso e o valor do percurso unitário dos itens. Esta classe possui métodos getters, setters e toString.

Classe "**Itens()**": Esta classe cria os itens informados pelo usuário em mapas, assim armazenando-os como pares de chave-valor, então para cada nome de item é relacionado um peso. Esta classe além dos métodos getters, setters e toString, também possui o método "pesoTotal()" onde é calculado o peso total dos itens criados, e o método "addItens" onde é adicionado itens ao mapa.

Todo o desenvolvimento é acompanhado de comentários com explicações sobre código.

TESTES (aqui você deverá colar capturas de tela de todas as funcionalidades desenvolvidas e realizar comentários, use o espaço que julgar necessário)

Menu principal:

```
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
         Menu de opções!
   1 --> Consultar trechos x modalidade
   2 --> Cadastrar transporte
   3 --> Dados estatísticos
   0 --> Sair do programa
Opção desejada (informe o número referente a opção): 0
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
Fim do programa :)
```

Este menu foi implementado de forma que o usuário possa informar o número corresponde a funcionalidade desejada. O programa inicia com a apresentação deste menu, e sempre ao fim de uma funcionalidade, ele somente deixará de executar caso o usuário informe que deseja sair do programa (informando o número zero).

FUNCIONALIDADE 1 – Consultar trechos x modalidade

```
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
Consulta de trechos e modalidades
  --> Modalidades de transporte disponíveis:
 1 - Modalidade: Pequeno - Preço por km: R$4.87
 2 - Modalidade: Medio - Preço por km: R$11.92
 3 - Modalidade: Grande - Preço por km: R$27.44
Cidades com trechos de destino e saída:
 1 - ARACAJU,         2 - BELEM,         3 - BELO HORIZONTE
 4 - BRASILIA,        5 - CAMPO GRANDE,  6 - CUIABA
 7 - CURITIBA,        8 - FLORIANOPOLIS, 9 - FORTALEZA,
10 - GOIANA,         11 - JOAO PESSOA,  12 - MACEIO,
13 - MANAUS,         14 - NATAL,        15 - PORTO ALEGRE
16 - PORTO VELHO,    17 - RECIFE,       18 - RIO BRANCO
19 - RIO DE JANEIRO, 20 - SALVADOR,     21 - SAO LUIS
22 - SAO PAULO,      23 - TERESINA,     24 - VITORIA
    > Digite os dados do transporte:
    * Indique o número referente a cidade e a modalidade:
      Cidade de saída: 3
      Cidade de destino: 8
      Modalidade: 2

O transporte de BELO HORIZONTE para FLORIANOPOLIS
 utilizando um caminhão de modalidade Medio, a distancia é de 1301 km e o custo será de R$15507.92
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
```

Se o usuário escolher a funcionalidade 1 do menu principal, será apresentado a funcionalidade de consultar trechos e modalidade, realizando uma simulação do transporte sobre aquele trecho. Textualmente será apresentado as modalidades possíveis e as cidades possíveis, para que o usuário possa ter conhecimento sobre os dados e informar seus desejos de maneira mais simplificada. É informado ao usuário que ele deve informar ao sistemas suas

opções através dos números que as representam, caso a informação passada pelo usuário seja inválida, isso será informado ao usuário.

```java
public void calcularTrecho(int indexSaida, int indexDestino, int modalidade){
    // --> Para armazenar o valor final calculado.
    double valorTotal = 0;

    // --> "cidadeSaida" armazena a cidade de saída que o usuário informar.
    Cidade cidadeSaida = listaCidades.get(indexSaida-1);
    // --> "cidadeDestino" armazena o valor (km) referente a cidade de destino informada.
    //      --> Para isso, é necessario utilizar a "cidadeSaida" que armazena a cidade de saída.
    //      --> "Estando" dentro do array da cidade de saída, buscamos a quilometragem da cidade de destino.
    String cidadeDestino = cidadeSaida.getDistanciaX( index: indexDestino-1);

    String mod = getNomeByIndex(modalidade);
    double valorMod = getPrecoKmByIndex(modalidade);

    // --> Calcula o valor total.
    valorTotal = Double.parseDouble(cidadeDestino) * valorMod;

    // --> Imprime as informações sobre a consulta.
    Cidade cidadeDestinoNome = listaCidades.get(indexDestino-1);
    System.out.println("O transporte de " + cidadeSaida.getNome() + " para " + cidadeDestinoNome.getNome()
            + "\n utilizando um caminhão de modalidade " + mod + ", a distancia é de " + cidadeDestino
            + " km e o custo será de R$" + valorTotal);
}
```

Método para calcular o trecho em Java

Para o calculo da consulta, é chamado o método “calcularTrecho()” da classe Agencia. Para realizar o calculo é informado por parâmetro o número referente a cidade origem e o número referente a cidade de destino, além do número referente a modalidade escolhida. Com o auxilio da lista de cidades criadas ao iniciar o programa, o método consegue identificar qual é a cidade de origem através do índice da lista, e com o conhecimento da cidade de origem é possível descobrirmos qual a distancia até a cidade de destino a partir de um método da própria classe Cidade que retorna a distancia a partir do índice da cidade. Para as modalidades, existem dois métodos auxiliares (implementados com switch case) que retornam o preço por quilometro e o nome da modalidade de caminhão a partir do número informado pelo usuário. Tendo conhecimento sobre a distancia total do percurso e o preço por quilometro do caminhão, então é realizado o calculo do trecho.

Ao final será apresentado ao usuário qual é o trecho escolhido, a modalidade de caminhão escolhida, a distancia total do trecho e o custo do trecho.

FUNCIONALIDADE 2 – Cadastrar transporte

```
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
Cadastrar transporte
Cidades disponíveis:
Cidades com trechos de destino e saída:
 1 - ARACAJU,         2 - BELEM,         3 - BELO HORIZONTE
 4 - BRASILIA,        5 - CAMPO GRANDE,  6 - CUIABA
 7 - CURITIBA,        8 - FLORIANOPOLIS, 9 - FORTALEZA,
10 - GOIANA,         11 - JOAO PESSOA,  12 - MACEIO,
13 - MANAUS,         14 - NATAL,        15 - PORTO ALEGRE
16 - PORTO VELHO,    17 - RECIFE,       18 - RIO BRANCO
19 - RIO DE JANEIRO, 20 - SALVADOR,     21 - SAO LUIS
22 - SAO PAULO,      23 - TERESINA,     24 - VITORIA

Informe as cidades para o transporte:
* A sequencia de cidades deve ser separadas por vírgula, SEM espaço entre as vírgulas e escritas como foi apresentado (usar letra maiúscula).
* Necessário informar ao menos DUAS cidades
SAO LUIS,MANAUS,CUIABA,JOAO PESSOA
```

Caso o usuário escolha a funcionalidade 2, inicialmente será apresentado as cidades disponíveis para que o usuário tenha a informação de todas as cidades e como são escritas, assim, é solicitado ao usuário que ele informe o nome de todos as cidades que deseja para o percurso – juntamente com as observações de como deve ser escrito as informações e que deve-se informar ao menos duas cidades.

```java
public double calcularDistancia(String cSaida, String cDestino){
    // --> Percorre a lista de cidades.
    for(Cidade cidade: listaCidades){
        // --> Se a cidade atual possuir o nome igual ao informado.
        if(cidade.getNome().equals(cSaida)){
            // --> Então percorre o array de nomes de cidades.
            for(int i = 0; i < nomesCidades.length; i++){
                // --> Se o nome da cidade for igual ao nome informado da cidade de destino.
                if(nomesCidades[i].equals(cDestino)){
                    // --> Então retorna a distancia desta cidade da lista de cidades.
                    //     --> Lembrete: A lista de cidades armazena o nome da cidade no primeiro indice, e nos demais as distancias entre as outras cidades.
                    //                   O array de nomesCidades armazena somente os nomes das cidades. E com esse array, conseguimos localizar onde cada cidade
                    //                   se encontra na lista de cidades.
                    //                      exem: Se o nome da cidade BELEM se encontra no indice [1] do vetor, ele estará no indice get(1) da lista de cidades.
                    trechos.add("De " + cSaida + " para " + cDestino + " a distancia do trecho é: " + cidade.getDistanciaX(i) + "\n");
                    return Double.parseDouble(cidade.getDistanciaX(i));
                }
            }
        }
    }
    // --> Caso não seja possivel calcular a distancia.
    System.out.println("Não foi possivel calcular a distância.");
    return -1;
}
```

Método para calcular a distância apenas um trecho do percurso em Java

```java
public int calculaDistanciaTotal(List<String> cidades){
    // --> Variável que armazena a distancia total.
    int distanciaTotal = 0;
    // --> Varável para verificar se há antecedente.
    String antecedente = null;
    // --> Percorrre o arrar de cidades que foi informado.
    for(String nomeCidades: cidades){
        // --> Se houver uma cidade.
        if(antecedente != null){
            // --> Então soma a distancia total ao calculo do trecho entre as duas cidades.
            distanciaTotal += calcularDistancia(antecedente, nomeCidades);
        }
        // --> Recebe a próxima cidade da lista.
        antecedente = nomeCidades;
    }
    // --> Retorna a distancia total.
    return distanciaTotal;
}
```

Método para calcular a distância total em Java

O método calcularDistanciaTotal() funciona em conjunto com o método calcularDistancia(). O método a ser chamado será o método para calcular a distancia total do percurso, este método recebe a lista de cidades que o usuário informará e percorrendo a lista de cidades, ele sempre calculará a distância entre duas cidades e soma-se as distancias em uma variável que retornará o valor total ao fim do método – a variável "antecedente" serve como auxilio para verificar se ainda há cidades parta realizar o cálculo. O método calcularDistancia() calcula a distância somente entre duas cidades, ele funciona de maneira semelhante ao método calcularTrecho(), entretanto este método recebe o nome de duas cidades e com o auxilio da lista de cidades e o *array* que armazena os nomes das cidades (que também são suas posições), ele retorna a distancia entre as duas cidades apenas acessando a lista de distancias do objeto Cidade, o qual será verificado para que seja a cidade de origem - e esse trecho é adicionado a lista de trechos para poder ser acessado posteriormente.

```
Exemplo de itens para transporte:
CELULAR -> peso: 0,5
GELADEIRA -> peso: 60,0
FREEZER -> peso: 100,0
CADEIRA -> peso: 5,0
LUMINARIA -> peso: 0,8
LAVADORA DE ROUPA -> peso: 120,0
*Informe o nome do item e peso como os exemplos acima ^

Informe os itens de transporte:
   --> Digite o nome do item:
     *Caso deseje terminar a lista, digite 'sair'
Cimento
   --> Peso do item:
5000,0
   --> Digite o nome do item:
     *Caso deseje terminar a lista, digite 'sair'
Tijolos
   --> Peso do item:
3000,0
   --> Digite o nome do item:
     *Caso deseje terminar a lista, digite 'sair'
sair
```

```
TIJOLOS: 3000.0
CIMENTO: 5000.0
Peso total dos itens: 8000.0
```

Após informar as cidades do percurso, o usuário então será apresentado a alguns exemplos de como informar os itens que deseja transportar. Primeiro o usuário deve informar o nome do item e depois o seu peso (em quilos), até que digite “sair” para que o sistema reconheça que terminou sua lista de itens. Caso ele termine sua lista, será informado quais são os itens da lista e o peso total da lista.

```java
public void listaDeItens(){
    Scanner entrada = new Scanner(System.in);
    // --> Auxiliar para saber se a lista continua ou não.
    boolean continua = true;
    // --> Enquanto o usuário quiser continuar.
    while(continua){
        System.out.println("   --> Digite o nome do item: ");
        System.out.println("     *Caso deseje terminar a lista, digite 'sair'");
        // --> Recebe o item.
        String nomeItem = entrada.nextLine();
        // --> Verifica se o item não é, na verdade, o usuário querendo terminar a lista.
        if(nomeItem.equalsIgnoreCase( anotherString: "sair")){
            // --> Se ele quiser terminar a lista, continua recebe false.
            continua = false;
        }else{
            // --> Se não, tenta:
            try{
                // --> Coloca o nome do item para letra maiúscula (nome ser padrão de formatação).
                String item = nomeItem.toUpperCase();
                System.out.println("   --> Peso do item: ");
                // --> Recebe o peso do item.
                double peso = entrada.nextDouble();
                // --> Limpa a entrada.
                entrada.nextLine();
                // --> Adicona o item.
                itens.addItem(item, peso);
            // --> Retorna uma exceção, se necessário.
            }catch (IllegalArgumentException e){
                System.out.println("Item inválido. Tente novamente :)");
            }
        }
    }
```

Método para realizar a lista de itens em Java

Para a inserção dos itens, considerei que seria melhor que o Scanner de entrada no sistema fosse no próprio método da lista de itens. Este método apresenta ao usuário a possibilidade de ficar informando itens até que deseje terminar a lista – para isso deve informar a palavra “sair” -, sempre que um item for criado é chamado o método do próprio objeto Item, onde é possível adicionar o item ao conjunto de itens. Ao fim do método é apresentado os itens adicionados e o peso total dos itens (calculado através do método “pesoTotal” da classe Item).

```
Valores do transporte:
 N° de transportes 1
 --> Caminhão (s) necessários: [Modalidade: Grande - Preço por km: R$27.44]
  --> Valor total do transporte: 303431.52
  --> Valor unitário: 151715.76
```

Após os itens, é apresentado os valores do transporte. Nesse momento é apresentado o número de transportes já cadastrados, os caminhões necessários para realizar a carga, o valor

total do transporte e o valor do transporte unitário (Para esse exemplo foi realizado o procedimento acima descrito, portando, são dois itens (cimento e tijolos) com o peso total de 8000,0 quilos).

```java
public void caminhaoAdequado(List<String> cidades){
    // --> Criado uma lista simples com as modalidades de caminhao.
    List<Modalidade> caminhoes = new ArrayList<>();
    caminhoes.add(Modalidade.PEQUENO);
    caminhoes.add(Modalidade.MEDIO);
    caminhoes.add(Modalidade.GRANDE);

    // --> Cria uma lista com os caminhões necessários para o transporte.
    List<Modalidade> necessarios = new ArrayList<>();
    // --> Guarda o peso total dos itens.
    double pesoTotal = itens.pesoTotal();
    // --> Auxiliar para caso encontre um caminhão.
    boolean encontrouCaminhao = false;
    // --> Percorre o array de caminhões possíveis.
    for(Modalidade caminhao: caminhoes){
        // --> Se a capacidade do caminhao for maior ou igual ao peso dos itens.
        double capacidadeCaminhao = caminhao.getCapacidade();
        if(capacidadeCaminhao >= pesoTotal){
            // --> Então armazena que encontrou um caminhão possível.
            encontrouCaminhao = true;
            // --> Se o caminhão ainda não está na lista de caminhões necessários.
            if(!necessarios.contains(caminhao))
                // --> Então adiciona o caminhão encontrado.
                necessarios.add(caminhao);
        }
        // --> Diminui, então, o peso que já foi liberado.
        pesoTotal -= capacidadeCaminhao;
        // --> Caso o peso já tiver sido suprido, não continua no for.
        if(pesoTotal < 0){
            break;
        }
    }
    // --> No caso de não encontrar nenhum caminhão.
    if(!encontrouCaminhao){
        System.out.println("Tente novamente :)");
    }
    // --> Variável para armazenar o custo total do transporte.
    double custoTotal = 0;
    // --> Variável para armazenar o custo por unidade.
    double valorUnitario = 0;
    // --> Variável para armazenar a distancia total do transporte.
    int distanciaTotal = calculaDistanciaTotal(cidades);
    // --> Percorre o array de caminhões necessários.
    for(Modalidade caminhao: necessarios){
        // --> Armazena o custo total da distancia a partir do custo por km do caminhão.
        double custoTotalDistancia = caminhao.getPrecoKm() * distanciaTotal;
        // --> Soma a variável de custo total.
        custoTotal += custoTotalDistancia;
    }
    valorUnitario = custoTotal/itens.getItens().size();
    // --> Cria um novo transporte com as informações que adquirimos.
    Transporte transporte = new Transporte(pesoTotal, necessarios, custoTotal, valorUnitario);
    // --> Adiciona na lista de transportes.
    transportes.add(transporte);
    // --> Imprime o transporte criado.
    System.out.println();
    System.out.println(transporte.toString());
    itens.clear();
}
// ================================================================================================================
```

Método para verificar os caminhões necessários em Java

Para o método de verificação de caminhões necessário são utilizadas duas listas: uma lista contendo os caminhões disponíveis para o transporte e uma lista onde será armazenado os caminhões necessários para o transporte, há também a variável "pesoTotal" que receberá o peso total da lista anteriormente informada pelo usuário como também há a variável "encontrouCaminhão" que servirá de auxiliar para as verificações posteriores. Utilizando o conceito do laço de repetição, a lista de caminhões disponíveis é percorrida, e a cada caminhão é realizada a verificação de que a capacidade desse caminhão é maior ou igual ao peso total da lista. Se for encontrado um caminhão, então é verificado se esse caminhão já se encontra na lista de caminhões necessários e caso não se encontre, ele é adicionado. Após a inserção, o peso total é subtraído pela capacidade do caminhão e o laço de repetição faz tudo novamente até que o peso total não seja mais um valor positivo (ou seja, não existam mais itens). Assim, a lista de caminhões necessários é percorrida, e é somado a uma variável "custoTotal" a multiplicação da distancia total com o preço por quilometro dos caminhões. Neste momento também é calculado o valor unitário dos itens do transporte (dividindo o custo total pela quantidade de itens). Ao fim do método, é realizada a criação de um novo transporte (classe Transporte) com as informações adquiridas, e esse transporte é adicionado a lista de transportes para poder ser acessado posteriormente. Esse método não satisfaz completamente a funcionalidade devido a dificuldade de implementar um algoritmo para retornar o melhor custo benefício dos caminhões.

FUNCIONALIDADE 3 – Dados estatísticos

```
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
Dados estatísticos
Trechos realizados:
[De SAO LUIS para MANAUS a distancia do trecho é: 5335
, De MANAUS para CUIABA a distancia do trecho é: 2357
, De CUIABA para JOAO PESSOA a distancia do trecho é: 3366
, De SAO LUIS para MANAUS a distancia do trecho é: 5335
, De MANAUS para CUIABA a distancia do trecho é: 2357
, De CUIABA para JOAO PESSOA a distancia do trecho é: 3366
]
Transportes realizados
[Valores do transporte:
 N° de transportes 1
 --> Caminhão (s) necessários: [Modalidade: Grande - Preço por km: R$27.44]
  --> Valor total do transporte: 303431.52
  --> Valor unitário: 151715.76]
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
```

Ao selecionar a funcionalidade 3, é apresentado ao usuário algumas estatísticas do sistema. Para isso, é chamado o método "dadosEstatisticos()" que imprime a lista de trechos e a lista de transportes cadastrados. Esse método não satisfaz completamente a funcionalidade devido a falta de acesso ou existência de algumas variáveis.

AUTOAVALIAÇÃO

Você concluiu a implementação de 100% das funcionalidades solicitadas?

( ) Sim (X) Não

Para as 3 principais funcionalidades solicitadas, como você avalia a sua solução?

Marque um 'X'.

	Inexistente/ Insuficiente	Pouco satisfeito(a)	Satisfeito(a)	Muito satisfeito(a)
Funcionalidade 1				X
Funcionalidade 2			X	
Funcionalidade 3		X		

Principais dificuldades

Para o desenvolvimento do teste técnico, inicialmente tive dificuldades em ler o arquivo .csv oferecido de maneira útil para o programa. Para conseguir realizar o que apresentei tive que fazer muitos testes antes de chegar na solução atual.

Para mim, a parte mais complicada do teste técnico foi a implementação do sistema dos caminhões da funcionalidade 2. Eu tive diversos raciocínios diferentes para aplicar o sistema de melhor custo beneficio entretanto não consegui implementa-lo, fazendo com que minha entrega final apenas diga o caminhão necessário para o transporte, mas não o melhor custo benefício para o usuário.

Desempenho Geral

No fim do exercício, por mais que não tenha entregado as funcionalidades 100% como foi solicitado, me surpreendi com meu rendimento. Acredito que consegui satisfazer bem uma parte do que foi solicitado, e isso me deixou animada a continuar tentando. Ao longo do desenvolvimento do teste houve muitos momentos de frustação onde eu não conseguia fazer tal método ou raciocínio funcionar como queria, e ao fim do teste não pude entrega-lo como eu desejava, mas acredito que meu resultado seja fruto de todo esforço que depositei nos últimos dias para entregar o melhor que poderia. Para toda a realização do exercício utilizei conhecimentos que adquiri ao longo dos 2 semestres passados na faculdade, tenho o costume de sempre guardar meus códigos de trabalhos avaliativos e de treino em repositórios em meu GitHub (https://github.com/ana-xavier), e quando havia algum detalhe que gostaria de implementar mais não me recordava de como corretamente fazer, eu acessava meus códigos antigos para consulta.

Obrigado por participar deste processo seletivo.
Salve o documento em PDF com o seu nome completo.