# Algoritmo de Otimização de Rota

Este projeto é um sistema simples que calcula alguns pontos específicos para transportar produtos de fornecedores para o Centro de Distribuição - CD.

## Parâmetros usados

A tabela abaixo contém os dados usados para executar as operações consideradas para o projeto proposto. Cada linha corresponde a um fornecedor, coordenadas x e y de sua localização que é usada para medir distâncias e Prod é a quantidade disponível para cada uma das posições:

| i | x | y | prod |
|---|---|---|---|
| 1 | 8 | 286 | 95 |
| 2 | 24 | 638 | 99 |
| 3 | 43 | 41 | 114 |
| 4 | 44 | 937 | 69 |
| 5 | 49 | 973 | 44 |
| 6 | 68 | 541 | 116 |
| 7 | 192 | 450 | 72 |
| 8 | 220 | 166 | 67 |
| 9 | 260 | 860 | 102 |
| 10 | 271 | 450 | 71 |
| 11 | 313 | 215 | 61 |
| 12 | 328 | 797 | 100 |
| 13 | 336 | 665 | 28 |
| 14 | 403 | 278 | 37 |
| 15 | 436 | 655 | 106 |
| 16 | 457 | 994 | 40 |
| 17 | 476 | 581 | 43 |
| 18 | 491 | 205 | 42 |
| 19 | 510 | 189 | 113 |
| 20 | 555 | 753 | 60 |

| i | x | y | prod |
|---|---|---|---|
| 21 | 574 | 786 | 74 |
| 22 | 595 | 604 | 21 |
| 23 | 631 | 205 | 99 |
| 24 | 635 | 455 | 30 |
| 25 | 643 | 139 | 90 |
| 26 | 673 | 803 | 90 |
| 27 | 706 | 838 | 116 |
| 28 | 749 | 634 | 56 |
| 29 | 750 | 265 | 46 |
| 30 | 759 | 147 | 90 |
| 31 | 767 | 750 | 107 |
| 32 | 821 | 422 | 105 |
| 33 | 876 | 514 | 87 |
| 34 | 880 | 198 | 41 |
| 35 | 887 | 670 | 43 |
| 36 | 919 | 336 | 84 |
| 37 | 928 | 310 | 41 |
| 38 | 972 | 423 | 112 |
| 39 | 976 | 373 | 91 |
| 40 | 987 | 667 | 26 |

Alguns parâmetros mais foram utilizados da como a seguir:

- Coordenadas do Centro de Distribuição: (500, 500)
- Custo de transporte por Km: 0,3 €
- Capacidade do caminhão: 200 unidades
- Custo diário do caminhão: 50 €

# Estrutura do Sistema:

É composto por 5 classes que visa dar uma organização e mais consistência ao gerenciar os dados. Abaixo segue descrição de cada uma delas e seus métodos mais importantes:

> SupplierProcessStarter.java: Esta é a classe principal, portanto, todo o processamento de dados é executado aqui, o que significa que é onde tudo começa. Abaixo estão listados os seus principais métodos:

- Main: É responsável por disparar todos os processos necessários para os outros métodos listados a diante.
- initVaribles: Inicia todas as variáveis globais que a classe precisa para funcionar apropriadamente.
- dataFeed: carrega todos os dados de pré-definidos que o sistema precisa processar, exceto a quantidade total a ser transportada que o usuário precisa inserir na primeira tela logo ao início do Sistema.
- orderingList: Este método é chamado pelo dataFeed para ordenar os registros distância do CD, permitindo que a leitura do array possa ser feita em sequência, da menor para a maior distância.
- calcTruckDemand: Gera uma lista de caminhões com base na razão da demanda dos produtos (informada pelo usuário) a serem coletados em caminhões com capacidade de 200 unidades.
- checkFreeTruck: Utilizado pelos métodos dos cenários para atribuir os caminhões que ainda não foram utilizados, checando a lista em busca do próximo disponível.
- cenario1NextDcDistance: Processa todos os cálculos relacionados à distância entre os fornecedores a partir do CD e monta todo o cenário com base nessa distância para efeitos de custo e distância.
- cenario2NextNeighbor: O mesmo que o método anterior, porém usa a distância menor do vizinho mais próximo para decidir para onde ir nos seguintes passos.
- calcDistance: Usado para fazer cálculos básicos entre 2 coordenadas, devolvendo a distância para quem chamou este método.
- nextDistance: Este método busca a distância mais curta relativo ao CD que foi calculada e armazenada em um array list, que é usado essencialmente pelo método

cenario1NextDcDistance.

- nextNeighbor: Similar ao método anterior, ele também obtém a distância mais curta, mas agora a partir da posição atual do caminhão, que é chamado de próximo vizinho.

- calcListOrders: executado no final dos cenários para obter todos os valores parciais (distância e valores) e os soma para fornecer os resultados finais/totais.

- melhorRota: verifica a melhor solução calculada pelo método acima, comparando os resultados de todos os cenários e o imprime.

> DataSupplierClass.java: criado para encapsular os dados principais pré-definidos, tornando organizados, consistentes e seguros para gerenciar em todas as operações.

Quando o construtor da segunda classe é instanciado, ele carrega dados nas respectivas variáveis e também executa o método para calcular x e y das coordenadas CD, definindo essa distância e grava-a em uma variável.

> TruckClass.java: formata um objeto para salvar as informações de caminhões que serão usadas em ambos os cenários para fazer parte de dados de vários arrays list.

## Cenários de teste

Foi considerado nessa primeira e simples versão de projeto 2 cenários, Next Neighbor e Shorter Distance em relação ao CD. Ambos são semelhantes quanto aos cálculos de custo e distância, sendo que a diferença está no detalhe de escolher a referência para ser tomada como base para selecionar o próximo ponto.

Em uma versão futura, pretende-se adicionar um terceiro cenário que considere a razão entre unidades (quantidade) disponível para serem coletadas e distância ser percorrida, tomando a melhor relação como parâmetro de escolha para próximo ponto, esperando ter assim o melhor custo x benefício.

# Instruções para executar o sistema de algoritmo de otimização de rotas:

Este simples Sistema foi desenvolvido usando o Java 11 e o IntelliJ IDE, sendo tudo o que é necessário para executá-lo. Mas se não for possível reproduzir este mesmo ambiente, existe uma opção para executá-lo em linha de comando, conforme descrito a seguir, atenção especial dada apenas para a versão do Java, porque seria necessário compilar tudo novamente, mas apenas no caso de utilizar outra versão.

- Caso exista a necessidade de nova compilação, vá para a pasta do projeto e digite o comando abaixo no Terminal (ou qualquer que seja no seu sistema que permita executá-lo pela linha de comando, considerando aqui que o Java já esteja instalado e rodando):

```
javac -d out/ src/pt/ipp/SupplierAlg/*
```

- Após a compilação, vá para a pasta out/ e execute este comando:

```
java pt/ipp/SupplierAlg.SupplierProcessStarter
```

# Requisitos

Para tê-lo funcionando, tudo o que você precisa é simplesmente o seguinte item (configurado e funcionando apropriadamente):

- Kit de Desenvolvimento Java SE 11