**Instituto Federal Catarinense-Campus Videira**

**Curso:** Ciência da Computação

**Professora:** Diego Ricardo Krohl

**Disciplina:** Estrutura de Dados II

**Aluno:** Cristiano Rodrigues Pirolli

**Turma:** 2024

# ALGORITMO PARA SISTEMA DE LOGÍSTICA

Este relatório detalha os módulos Python desenvolvidos e adaptados, explicando a função de cada algoritmo implementado e a sua aplicabilidade na resolução de desafios comuns em logística, como planeamento de rotas, gestão de redes e otimização de recursos.

## 1. Gestão da Base de Dados (`database_manager.py`)

- **Implementação**: Este módulo é responsável pela interação com uma base de dados MySQL. Ele define a estrutura das tabelas (Pontos, Conexões, Caminhões, Motoristas, Cargas) e fornece funções CRUD (Criar, Ler, Atualizar, Apagar) para manipular esses dados. Utiliza o `python-dotenv` para gerir dados sensíveis (credenciais da base de dados) de forma segura através de um ficheiro `.env`.
- **Resolução de Problemas**: É a espinha dorsal para a persistência de dados. Permite que o sistema armazene e recupere informações essenciais como localizações geográficas, as rotas disponíveis entre elas com as suas distâncias, detalhes da frota de veículos, informações dos motoristas e o estado das cargas. Sem uma gestão de base de dados eficaz, qualquer sistema de logística complexo seria inviável.

## 2. Estrutura de Grafo (`graph.py`)

- **Implementação**: Define uma classe `Grafo` que representa uma rede usando listas de adjacência. Permite adicionar vértices (que podem ser cidades, depósitos, clientes) e arestas ponderadas (que podem representar distância, tempo de viagem ou custo entre os vértices). As arestas podem ser bidirecionais (padrão) ou unidirecionais.

- **Resolução de Problemas**: Modela a rede logística do mundo real. Cidades, armazéns e clientes tornam-se nós (vértices), e as estradas ou rotas entre eles tornam-se ligações (arestas) com pesos que quantificam, por exemplo, a distância. Esta estrutura é fundamental para que algoritmos de otimização de rotas possam operar.

## 3. Caminho Mais Curto (`caminho_mais_curto.py` - Algoritmo de Dijkstra)

- **Implementação**: Implementa o algoritmo de Dijkstra para encontrar o caminho de menor custo (peso) entre um nó inicial e um nó final num grafo. Utiliza uma fila de prioridade (`heapq`) para explorar eficientemente os nós.
- **Resolução de Problemas**: Soluciona um dos problemas mais clássicos da logística: determinar a rota mais rápida, mais curta ou mais barata entre dois pontos. Por exemplo, calcular a melhor rota para um camião ir de um depósito até um cliente específico.

## 4. Árvore Geradora Mínima (`arvore_geradora_minima.py` - Algoritmo de Kruskal)

- **Implementação**: Contém o algoritmo de Kruskal para encontrar uma Árvore Geradora Mínima (AGM) num grafo. Uma AGM conecta todos os vértices do grafo com o menor custo total de arestas, sem formar ciclos. Utiliza a estrutura de dados União-Disjunção (DSU) para detetar ciclos eficientemente.
- **Resolução de Problemas**: Embora menos usado para o roteamento diário de veículos individuais, é útil no planeamento estratégico de redes. Por exemplo, se uma empresa precisa de instalar uma rede de fibra ótica (ou construir estradas de serviço mínimas) para conectar todos os seus depósitos e filiais com o menor custo total de cablagem/construção, Kruskal pode encontrar essa rede ótima.

## 5. Análise Euleriana (`analise_euleriana.py`)

- **Implementação**: Fornece funções para verificar se um grafo possui um Caminho Euleriano (percorre cada aresta exatamente uma vez) ou um Circuito Euleriano (um caminho Euleriano que começa e termina no mesmo vértice). A verificação baseia-se na análise dos graus dos vértices.
- **Resolução de Problemas**: Aplicável a problemas de logística onde o objetivo é cobrir todas as "ruas" ou "conexões" de uma área. Exemplos incluem:
    - Planeamento de rotas para varrimento de ruas.
    - Inspeção de todas as secções de uma pipeline ou rede elétrica.
    - Entrega de correio onde cada rua precisa de ser percorrida. A implementação atual verifica a *existência*, mas não constrói o caminho em si.

## 6. Seleção de Rotas Baseada em Múltiplos Critérios (`selecao_de_rota.py`)

- **Implementação**: Apresenta uma função que seleciona a "melhor" rota de uma lista de rotas candidatas com base numa lista hierárquica de critérios (ex: primeiro menor tempo, depois menor custo como desempate).
- **Resolução de Problemas**: Nem sempre a rota "mais curta" ou "mais rápida" é a "melhor" globalmente. Este módulo permite uma tomada de decisão mais sofisticada, considerando múltiplos atributos de uma rota (distância, tempo, custo, etc.) e as prioridades definidas pelo utilizador/sistema. Por exemplo, escolher uma rota que, embora não seja a mais curta, evite portagens (custo) ou tenha um tempo de viagem aceitável.

## 7. Codificação de Huffman (`huffman_coding.py`)

- **Implementação**: Implementa o algoritmo de Codificação de Huffman, que é um método de compressão de dados sem perdas. Ele constrói uma árvore baseada na frequência dos caracteres para gerar códigos de comprimento variável (caracteres mais frequentes recebem códigos mais curtos).
- **Resolução de Problemas**: A sua aplicação direta em problemas de otimização de rotas logísticas é menos óbvia que os outros algoritmos. No entanto, pode ser útil para:
    - Comprimir dados armazenados ou transmitidos, como longas sequências de instruções de rota, dados de telemetria de veículos ou descrições de cargas, economizando espaço em disco ou largura de banda.

## Conclusão

Os módulos desenvolvidos fornecem um conjunto de ferramentas robusto para abordar diversos aspetos de um sistema de gestão logística. A capacidade de modelar a rede como um grafo (`graph.py`) e armazenar dados de forma persistente (`database_manager.py`) serve de base. Algoritmos como Dijkstra (`caminho_mais_curto.py`) são cruciais para o cálculo de rotas ótimas individuais. Kruskal (`arvore_geradora_minima.py`) auxilia no design eficiente de redes. A análise Euleriana (`analise_euleriana.py`) resolve problemas de cobertura total de arestas, enquanto a `selecao_de_rota.py` permite decisões mais complexas e multicritério. A `huffman_coding.py` pode otimizar o manuseamento de dados.

Integrados, estes componentes permitem construir um sistema capaz de não só calcular rotas, mas também de gerir a infraestrutura e tomar decisões logísticas mais informadas e eficientes.