# Relatório – Programação II

# Projeto II

---

#Ano 2023-2024
#Grupo 31
#62228 Oujie Wu
#59348 Dmytro Umanskyi

---

Neste semestre, após ler o enunciado e as especificações fornecidas pelo professor, investigamos a estrutura básica de solução do projeto e baixamos os arquivos **Python depthFirstSearchInGraphCap122.py** e **knapsackFillingGreedyANDBruteForceCap141.py** Com base nesses arquivos, realizamos e adicionamos uma série de desenvolvimentos e funções, resultando na criação de 8 arquivos Python e 4 tipos diferentes de testSets.

## Funções Utilizadas

Primeiramente, para garantir que tivéssemos algo para testar no início do projeto, criar um testSet (conjunto de testes) para guiar o desenvolvimento foi essencial. Isso nos permitiu considerar a forma real da solução e comparar com métodos de solução de problemas anteriores, ajustando assim nossos algoritmos. Nestes testSets, criamos um total de 4 tipos diferentes de testSets para se adaptar a situações específicas, como **'out of network'** e **'do not communicate'** entre os nós, além de problemas de borda ponderada e ordem de nomes.

Em seguida, para facilitar a gestão, separamos os diferentes métodos das classes contidas em **depthFirstSearchInGraphCap122.py** em arquivos individuais, resultando em classes isoladas como **Node(object), Edge(object), Digraph(object) e Graph(Digraph)**. Após a classificação, começamos a adicionar funcionalidades de inicialização a essas classes, como nós, bordas, grafos direcionados e não direcionados.

Além disso, criamos mais 3 novos arquivos: **infoFromFiles.py, Path.py e WeightedEdge.py.** Os arquivos Path.py e WeightedEdge.py utilizam e mantêm a lógica de alguns métodos do arquivo **knapsackFillingGreedyANDBruteForceCap141.py.**

O arquivo **infoFromFiles.py** tem a função de ler os arquivos **myLevadasNetwork** e myStations dos testSets, construir um grafo com nós e bordas ponderadas e retornar uma lista de pares de pontos.

A classe **Path.py** é uma ferramenta para gerenciar caminhos e encontrar o melhor caminho. Ela é usada para busca e otimização de caminhos no grafo, principalmente através da busca em profundidade (DFS) para encontrar todos os possíveis caminhos, determinando e armazenando os melhores caminhos com base no tempo gasto.

No **WeightedEdge.py**, reutilizamos a lógica de alguns métodos do arquivo **knapsackFillingGreedyANDBruteForceCap141.py.** e, com base nisso, adicionamos mais casos de comparação de pesos. Esta classe é usada para representar bordas ponderadas no grafo, conectando dois nós (origem e destino) e incluindo o peso da borda, que representa o tempo gasto para ir do nó de origem ao nó de destino.

Finalmente, através da função central **safeLevadas.py**, integramos todas as funcionalidades mencionadas. Esta função lê dados dos arquivos de entrada (dos testSets), constrói a estrutura do grafo e, com base nos pares de pontos predefinidos, realiza a busca de caminhos, escrevendo os resultados no arquivo de saída. As etapas específicas incluem: leitura dos dados de entrada, busca dos nós no grafo, execução da busca em profundidade (DFS) para encontrar o melhor caminho, processamento dos resultados da busca e registro das informações relevantes.

Através dessas operações, **safeLevadas.py** realiza a otimização eficiente dos caminhos das levadas, permitindo que o aplicativo levadas gerencie e selecione caminhos de maneira mais eficiente, atendendo a requisitos de negócios mais complexos.

## Contribuições

Neste projeto, Oujie Wu foi responsável por construir os testSets adaptados a diferentes tipos de situações e desenhar mapas de rotas dos diferentes **myLevadasNetworks**, facilitando a visualização de como os diferentes pontos estão conectados. Além disso, ele participou do teste e desenvolvimento da função **safeLevadas.py**, comparando os resultados dos diferentes testSets para garantir que os resultados do algoritmo fossem corretos.

Dmytro Umanskyi desempenhou um papel central no design, experimentação, integração e implementação de **Node.py, Digraph.py, Graph.py, Edge.py, Path.py, WeightedEdge.py**, **safeLevadas.py** pois essas classes recém-introduzidas estão diretamente relacionadas às funções principais do nosso projeto: encontrar nós no grafo, executar busca em profundidade (DFS) para encontrar o melhor caminho, processar os resultados da busca e registrar as informações relevantes. Essas classes facilitaram a integração e o uso de todos os dados no planejamento, resultando finalmente em um texto - o arquivo **myResult** - que calcula o caminho mais curto e o tempo necessário entre os dois pontos requisitados.