FUNDAÇÃO HERMÍNIO OMETTO

Engenharia da Computação – 7º Período

Guilherme Cavenaghi – RA 109317

Rafael Pereira de Souza - RA 109680

Vinicius Rossi – RA 110273

RELATÓRIO SISTEMA GRAFO METROVIARIO

Araras – SP

2023

1. Objetivo

O objetivo é implementar um código que controla as linhas de um metrô, para isso como é pedido no trabalho representamos a mesma através de um Grafo, basicamente é um modelo que representa relações entre objetos e permite ao programador solucionar problemas e desenvolver códigos mais eficientes.

2. Bibliotecas

iostream: Permite a interação com o usuário por meio do console, tratando a entrada e saída padrão.

vector: Permite armazenar e manipular elementos em uma sequência redimensionável.

queue: Implementa fila, seguindo o conceito FIFO (First In, First Out). É usada, por exemplo, para a busca em largura (BFS) no sistema metroviário.

stack: Oferece uma implementação de pilha, seguindo o conceito LIFO (Last In, First Out). É usada, por exemplo, para a busca em profundidade (DFS) no sistema metroviário.

climits: define constantes com os limites das variáveis numéricas, como INT_MAX e INT_MIN. Esses limites são úteis para inicializar pesos de conexões no sistema metroviário.

algorithm: Disponibiliza funções genéricas para operações em intervalos de elementos. Por exemplo, a função find_if pode ser usada para localizar uma estação específica a ser removida.

limits: Define constantes com os limites das variáveis numéricas, como numeric_limits<streamsize>::max(). Isso pode ser

empregado para limpar o buffer de entrada quando se detecta uma entrada inválida.

3. O código

Definimos uma classe chamada MetroSystem que representa um sistema metroviário. Essa classe encapsula todas as funcionalidades relacionadas ao sistema, como a criação do grafo, adição e remoção de estações, busca em largura, busca em profundidade, busca de caminho, busca do menor caminho, busca da árvore geradora mínima e impressão do sistema.

As funcionalidades definidas nessa classe foram:

createGraph(): É responsável por criar o grafo do sistema metroviário com base nas informações fornecidas pelo usuário.

addStation(): Permite adicionar uma nova estação ao sistema. O usuário informa o nome da estação e suas conexões com as estações existentes.

removeStation(): Permite remover uma estação do sistema. O usuário informa o nome da estação a ser removida.

breadthFirstSearch(): BFS -> Realiza uma busca em largura a partir de uma estação inicial, exibindo as estações visitadas.

depthFirstSearch(): DFS-> Realiza uma busca em profundidade a partir de uma estação inicial, exibindo as estações visitadas.

findPath(): É responsável por encontrar um caminho entre duas estações no sistema metroviário. Ela implementa uma busca em largura (BFS) modificada para rastrear o caminho percorrido.

findShortestPath(): É responsável por encontrar o caminho mais curto entre duas estações no sistema metroviário.Utilizando o algoritmo de Dijkstra para calcular os caminhos mais curtos a partir de uma estação de partida.

findMinimumSpanningTree(): Essa função implementa o algoritmo de Prim para encontrar a Árvore Geradora Mínima (Minimum Spanning Tree) no sistema metroviário. Encontra a árvore geradora mínima do sistema metroviário.

printMetroSystem(): Permite escolher a forma de impressão do sistema metroviário, podendo ser em lista de adjacência ou matriz de adjacência.

printAdjacencyList(): Imprime o sistema metroviário em forma de lista.

printAdjacencyMatrix(): Imprime o sistema metroviário em forma de matriz.

Seguindo para a struct Station onde representa uma estação, contendo o nome da estação e uma lista de conexões com outras estações, representadas por pares (índice da estação, tempo necessário para ir de uma estação para outra).

stations é um vetor de Station, que armazena todas as estações do sistema.

adjacencyList é um vetor de vetores de pares, que representa a lista de adjacência do grafo do sistema metroviário. Cada índice do vetor representa uma estação, e cada elemento do vetor interno representa uma conexão com outra estação.

adjacencyMatrix é uma matriz de inteiros, que representa a matriz de adjacência do grafo do sistema metroviário. O valor -1 indica que não há uma conexão direta entre as estações.

Foi criado o grafo das linhas metroviárias usando a função createGraph(), solicitando ao usuário o número de estações, seus nomes e as conexões entre elas, e preenchendo os membros de

dados stations, adjacencyList e adjacencyMatrix com as informações fornecidas.

4. Manipulação de dados

O código solicita ao usuário que digite o número de estações do sistema metroviário. Essa abordagem permite criar um grafo com o tamanho adequado para acomodar todas as estações e suas conexões.

O uso do loop while com a função cin -> numStations garante que apenas um número inteiro válido seja aceito como entrada. Caso o usuário insira um valor inválido e solicita uma nova entrada.

O uso de cin.clear() e cin.ignore() garante que quaisquer caracteres inválidos sejam limpos.

Inicialização das estruturas de dados:

Após obter o número de estações, o código redimensiona os vetores stations, adjacencyList e adjacencyMatrix para terem o tamanho correto.

O código solicita ao usuário que digite o nome de cada estação individualmente e armazena os nomes no vetor stations.

A iteração ocorre de 0 a numStations - 1, onde i representa o índice da estação sendo lida.

Para se adicionar as conexões é iterado sobre cada estação e solicita ao usuário o tempo necessário para a viagem entre a estação atual e todas as outras estações do sistema.

A verificação (if (i != j)) e garante que não haja uma conexão entre uma estação e ela mesma.

O uso de um loop while com a função cin -> weight garante que apenas um número inteiro válido seja aceito como entrada para representar o tempo de viagem.

As informações das conexões são armazenadas tanto na lista de adjacência, quanto na matriz.

Ao finalizar a criação do grafo, o código exibe uma mensagem informando que o grafo foi criado com sucesso.

5. Escolhas

O código foi projetado para simular um sistema metroviário e resolver problemas relacionados a esse tipo de sistema. Ao criar um sistema metroviário, é fundamental garantir que as conexões entre as estações sejam eficientes e que os passageiros possam navegar facilmente de uma estação para outra. Para isso, foram implementados algoritmos específicos que ajudam a otimizar o sistema e fornecer informações úteis aos usuários.

Um desses algoritmos é o algoritmo de Prim, que é usado para encontrar a Árvore Geradora Mínima do sistema metroviário. Essa árvore representa um subconjunto das conexões originais que é essencial para garantir que todas as estações sejam acessíveis e que o tempo de percurso total seja minimizado. Essa abordagem ajuda a criar um sistema metroviário eficiente, no qual as estações são conectadas de maneira inteligente.

Outro algoritmo importante implementado é o algoritmo de Dijkstra. Ele é usado para encontrar o caminho mais curto entre duas estações no sistema metroviário, levando em consideração o tempo de percurso. Isso permite que os usuários planejem suas viagens de forma mais eficiente, obtendo informações precisas sobre a duração do trajeto entre diferentes estações. Essa funcionalidade é especialmente útil para os passageiros que desejam economizar tempo e chegar rapidamente ao seu destino.

Além desses algoritmos principais, a implementação do código também inclui outras estruturas de dados, como listas de adjacência e matriz de adjacência. Essas estruturas são utilizadas para representar as conexões entre as estações e fornecer acesso rápido e eficiente às informações relevantes. Dessa forma, é possível aplicar os algoritmos de forma eficaz e obter os resultados desejados.

6. O Sucesso

Em resumo, o código foi desenvolvido dessa forma para criar um sistema metroviário simulado e resolver problemas relacionados, como otimização das conexões entre as estações e determinação de caminhos mais curtos. Os algoritmos de Prim e Dijkstra desempenham papéis fundamentais nesse processo, garantindo que o sistema seja eficiente e forneça informações úteis aos usuários. As estruturas de dados complementares ajudam a organizar as informações e permitir operações rápidas nos dados do sistema.