UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS TRINDADE
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO CIÊNCIAS DA COMPUTAÇÃO

Rita Louro Barbosa – 22203157

NE5413 – Grafos

Atividade Prática A1

Introdução

Este relatório contém a apresentação do trabalho realizado na resolução de 5 problemas propostos. Para essa resolução, foi implementada uma estrutura de grafo e demais estruturas atreladas, função de leitura de arquivo de grafo e os seguintes algoritmos: busca em largura (breadth first search), Busca de ciclo Euleriano com a utilização do algoritmo de Hierholzer, Bellman Ford e Floyd Warshall.

Observação importante: Procurou-se seguir com fidelidade os algoritmos apresentados no livro. Foram adicionados comentários ao longo do código para facilitar a identificação da implementação de cada etapa do pseudocódigo presente no livro.

1) Questão 1

A) A questão 1 implementa as seguintes estruturas:

- **Struct WeightedGraph**

Para a representação do grafo, foi criada uma estrutura de grafo ponderado contendo uma lista de adjacências utilizando um **HashMap**, no qual cada chave (id) leva a uma lista de “Edge”s. A utilização de um HashMap para armazenar as listas de “Edges” foi feita por 2 motivos principais: Inicialmente, a ideia era permitir qualquer tipo de dado para indexação. Ou seja, o ID (chave) poderia ser de qualquer tipo com possibilidade de Hash (string, char, int, float). Em segundo lugar, o HashMap funciona com otimizações na complexidade de busca, tendo complexidade próxima a O(n) na maioria dos casos (n = número de chaves). Ao longo da implementação do trabalho, verificou-se que a possibilidade de Ids de tipo genérico não forneceria uma utilidade relevante para nenhum caso de grafo utilizado, uma vez que , na maioria dos casos, os grafos são indexados por valores inteiros. Logo, restringiu-se o tipo de chave para i32.

- **Struct Edge**

É a estrutura criada para armazenar os dados de uma aresta, contendo o vizinho (destino) e o peso (w).

- **Struct Node<T>**

Ao longo da realização do trabalho, ao verificar a exigência da possibilidade de consulta das informações dos nodos, surgiu a necessidade de se ter uma lista de nodos contendo a informação de cada nodo (id e rótulo).

B) Funções:

Foram implementadas todas as funções requisitadas na descrição da questão 1: *qtdVertices(), qtdArestas(), grau(v), rotulo(v), vizinhos(v), haAresta(u, v), peso(u, v), ler(arquivo).* Também foi implementado o método arestas(), que retorna um vetor de

triplas, contendo os vértices da aresta e o peso. Além disso, foram implementadas funções complementares que pudessem servir futuramente para a expansão das funcionalidades do grafo. As funções atreladas à estrutura Weighted Graph procuraram manter a menor complexidade possível . É possível vê-las detalhadamente no escopo do código.

2. Questão 2

Para essa questão, foi implementado o algoritmo de busca em largura descrito na página 24 do livro “Anotações para a disciplina de grafos".

Para ele, foram criadas 4 estruturas complementares para a busca:

- O vetor booleano ‘C’ para determinar o conhecimento dos vértices,
- O vetor D para armazenar a distância percorrida de ‘s’ até determinado vértice,
- O vetor A, que guarda o id do vértice antecessor ao vértice da posição
- Uma fila Q para determinar a ordem da busca

Para representar os valores “infinito “ e “vazio”, foram utilizados, respectivamente, o valor máximo da representação de inteiro de 32 bits e o valor -1 (já que Rust não permite a criação de vetores de tipo i32 com posições com valor vazio).
Quanto à complexidade, esta pode ser aproximada para O(|V|+|E|).

3. Questão 3

Esta questão implementa a busca pelo ciclo euleriano, descrito na página 34 do livro “Anotações para a disciplina de grafos". Para resolução da questão, foram necessárias 3 funções:

- cicloEuleriano: Verifica se todos os vértices têm grau par e, em caso positivo, chama o algoritmo de Hierholzer para encontrar o ciclo Euleriano. Retorna um par indicando se o ciclo foi encontrado e o ciclo em si.

- algoritmoHierholzer: Algoritmo principal que implementa o algoritmo de Hierholzer. para ele, foram utilizadas, principalmente, as estruturas:
    - Matriz C: Registra a visita de cada aresta. Escolheu-se utilizar uma matriz para explorar a eficiência de acesso de cada posição pelo índice {u,v} em cada visita. As arestas que não fazem parte do grafo são inicializadas com “true” (como se já tivessem sido visitadas) , para que não interfiram no andamento do algoritmo.
    - Vetor E: Contém todas as arestas do grafo, em duplas, para facilitar o andamento do algoritmo (já que o grafo original é estruturado como lista de adjacências)

- buscarSubcicloEuleriano: Esta função auxiliar encontra um sub ciclo Euleriano a partir de um vértice dado. Utiliza as seguintes estruturas:
    - O grafo “graph” não dirigido

    - Matriz C
    - Vetor ciclo: guarda o ciclo euleriano encontrado.

A complexidade do algoritmo é aproximadamente $O(|V|+|E|)O(|V|+|E|)$, onde V é o número de vértices e E é o número de arestas no grafo.

4. Questão 4

Foi implementado o algoritmo de Bellman-Ford, descrito na página 52 do livro “Anotações para a disciplina de grafos", para encontrar o caminho mais curto a partir de um vértice de origem em um grafo ponderado. Para isso, foram usadas as seguintes estruturas:

- Vetor D para armazenar o valor da distância do vértice ‘s’ para determinado vértice. Este vetor foi inicializado com o valor “f32::INFINITY” que representa o valor infinito para um float de 32 bits.
- Vetor A para armazenar o antecessor de cada vértice ‘v’ no caminho mínimo. Este vetor foi inicializado com o valor “-1” para representar o valor “vazio”.

5. Questão 5

Para essa questão, foi implementado o algoritmo de Floyd-Warshall , descrito na página 52 do livro “Anotações para a disciplina de grafos", que encontra o caminho mínimo para um grafo G(V, E , w) ponderado dirigido ou não para todos os pares de vértices . Para isso, utilizou a função Floyd-Warshall e função auxiliar W:

- **floyd_warshall:** Esta função implementa o algoritmo de Floyd-Warshall para encontrar os caminhos mais curtos entre todos os pares de vértices em um grafo ponderado. Para isso, utiliza a seguinte estrutura principal:
    - Matriz D: Matriz que representa as distâncias de todos os vértices para os demais vértices do grafo.
- **Função W**: Esta função auxiliar constroi a matriz de distâncias inicial, onde os valores das células representam os pesos das arestas entre os vértices. Se não houver uma aresta entre dois vértices, o valor na matriz é definido como infinito.

6. Rodando arquivos:

- **Necessário:** Ter Rust e Cargo instalados.

Para isso, siga as instruções, conforme seu sistema operacional:

- Linux: Utilize o comando:

curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh

- Windows , com Chocolatey instalado, utilize o comando:
choco install rust

Para demais casos, verificar o instalador no site oficial de Rust:
https://doc.rust-lang.org/cargo/getting-started/installation.html

- **Rodando arquivos de questões:**

Vá até a pasta “codigo” e abra o terminal nela.

Comando genérico:
cargo run --bin qx nomeArquivo.net arg1 arg2

onde: x = número da questão e args conforme necessário

OBS: Arquivos de grafos devem ser colocados na pasta ‘grafos_teste’, localizada na pasta ‘src’

Comandos específicos:

Questão 1:
cargo run --bin q1 nomeArquivo.net

Questão 2:

cargo run --bin q2 nomeArquivo.net s

* s = id do vértice de origem s

Questão 3:

cargo run --bin q3 nomeArquivo.net

Questão 4:

cargo run --bin q4 nomeArquivo.net s

* s = id do vértice de origem s

Questão 5:

cargo run --bin q5 nomeArquivo.net