Implementação de Grafo com Carga Primária

Trabalho I da Disciplina de Teoria dos Grafos

Lucas Oliveira Macedo, Matheus Azevedo de Sá e Rian Carlos Valcanaia

Professor: Gilmário Barbosa dos Santos

Maio de 2025

1 Identificação da Tarefa

Esta tarefa consiste na implementação, em linguagem C, de um sistema capaz de construir e manipular um grafo cujos vértices representam observações presentes em uma base de dados no formato CSV. O grafo deve ser carregado em memória utilizando uma das estruturas discutidas em aula (matriz de adjacências, matriz de incidência ou lista de adjacências).

1.1 Especificação Técnica

- Cada linha do arquivo CSV representa um vértice do grafo, totalizando 204 vértices.
- As arestas são estabelecidas entre pares de vértices cuja distância euclidiana normalizada (DEN) seja menor ou igual a 0,3.
- A distância euclidiana normalizada é obtida a partir da normalização das distâncias euclidianas entre todos os pares de vértices.
- O sistema deve permitir:
  - Carregamento do grafo a partir do arquivo CSV original.
  - Persistência do grafo em um novo arquivo CSV, contendo no cabeçalho informações como: total de vértices, maior e menor distância euclidiana (DE) e normalizada (DEN) com seus respectivos pares de vértices, e quantidade/tamanho dos componentes conexos.
  - Não é necessário implementar integração entre C e Python, mas é obrigatória a utilização de um script Python para visualização tridimensional do grafo, permitindo manipulação do ponto de vista via mouse.

1.2 Resumo dos Passos Implementados

- Leitura da base de dados CSV.
- Cálculo das distâncias euclidianas para todos os pares de vértices.
- Normalização das distâncias.
- Aplicação do limiar de 0,3 para definição das arestas.
- Construção da matriz de adjacências.
- Geração do arquivo CSV de persistência com as informações solicitadas no cabeçalho.
- Visualização do grafo via script Python conforme orientado.

2 Ferramentas Utilizadas

Para este trabalho foram usados arquivos cabeçalhos da biblioteca padrão de C, além da IDE *Visual Studio Code* para desenvolvimento de código e *Git/Github* para versionamento de código. Também foi utilizado o script de Python disponibilizado pelo professor no Moodle para plotagem do grafo em 3D.

2.1 stdio.h

Neste cabeçalho da biblioteca padrão estão localizadas as funções referentes às operações nas quais os mecanismos operam em função da entrada e da saída padrão, bem como em arquivos também.

2.2 stdlib.h

Neste arquivo cabeçalho da biblioteca padrão estão localizadas as funções responsáveis pela manipulação da alocação de memória, e da desalocação também, bem como funções para converter números que estão representados em strings para algum tipo de dado responsável por representar números.

2.3 string.h

Neste arquivo cabeçalho da biblioteca padrão estão contidos protótipos utilizados para a manipulação de strings. Estas funções são capazes desde a contagem, cópia e concatenação, comparação e diversas outras funções úteis para o atual projeto.

2.4 math.h

Neste arquivo cabeçalho da biblioteca padrão estão contidas funções e macros para cálculos matemáticos. Ela fornece protótipos para funções que implementam operações matemáticas como funções trigonométricas, exponenciais, logaritmos e outros cálculos.

2.5 float.h

Neste arquivo cabeçalho da biblioteca padrão estão contidos macros e definições para limites e precisão dos números de ponto flutuante na linguagem.

3 Estruturas de Dados Utilizadas

No programa, são usadas duas estruturas principais para representar os vértices do grafo e as informações calculadas durante a construção do grafo: `No` e `Info_grafo`.

- `No:` representa um vértice no grafo, que é um ponto no espaço tridimensional. Cada vértice possui um identificador único e suas coordenadas espaciais. No código:

```
typedef struct no {
    int indice;  // Identificador unico do vertice.
    float x, y, z;  // Coordenadas do vertice no espaco 3D.
} No;

```

- `Info_Grafo:` A estrutura `Info_grafo` armazena informaçõe relacionadas ao grafo como um todo. Ela guarda métricas sobre as distâncias entre os vértices, tanto as distâncias Euclidianas quanto as distâncias Euclidianas normalizadas, e também os pares de vértices que determinam esses valores extremos. No código:

```
typedef struct info_grafo {
    int totalVertices;  // Numero total de vertices no grafo.
    float maxDE, minDE, maxDEN, minDEN;  // Maior e menor
    distancia Euclidiana e Normalizada.
    struct no maxDE_1, maxDE_2;  // Vertices que determinam a
    maior distancia Euclidiana.
    struct no minDE_1, minDE_2;  // Vertices que determinam a
    menor distancia Euclidiana.
    struct no maxDEN_1, maxDEN_2;  // Vertices que determinam
    a maior distancia Normalizada.
    struct no minDEN_1, minDEN_2;  // Vertices que determinam
    a menor distancia Normalizada.
} Info_grafo;

```

4 Funcionalidades

4.1 Funções Principais

Estas funções representam os **processos centrais** do programa:

4.1.1 Carregamento de Arquivo

Lê um arquivo.csv contendo vértices no formato x,y,z e armazena em um vetor dinâmico de estruturas no.

```
int carregaArquivo(struct no **vertices, int *qtAlocada, int *
    totalVertices, char nomeArquivo[100])
```

4.1.2 Criação do Grafo

Cria um grafo não-direcionado a partir de um conjunto de vértices, com base na distância euclidiana entre eles.

```
int criaGrafo(No *vertices, int qtVertices, struct info_grafo *
    infos, int ***matrizADJ)
```

4.1.3 Gravação do Grafo em Arquivo

Salva as informações do grafo no arquivo Grafo.csv, que possibilita a leitura pelo sistema, bem como possui uma estrutura bem clara para a leitura do usuário.

```
int salvaGrafo(struct info_grafo infos, int qtVertices, int **
    matrizADJ, struct no *vertices)
```

4.1.4 Leitura de Grafo Salvo em Arquivo

Lê um grafo salvo em arquivo .csv, reconstruindo as informações estruturais e a matriz de adjacência.
Obs: esta função só carrega um grafo se o grafo possuir a saída gerada por este sistema, use aqui o arquivo gerado Grafo.csv.

```
int carregaGrafo(struct info_grafo *infos, int ***matrizADJ)
```

4.1.5 Plotagem de Grafo em 3D

Recebe um arquivo.txt com uma lista de arestas e chama o script de Python para plotar o grafo em 3D a partir da linha de comando da máquina utilizando a função `system()`. Vale ressaltar que só funcionará se a sua máquina for Linux com Python instalado.

```
void plotar_grafo_3d(const char *arquivo_txt)
```

4.2 Funções Secundárias

Estas funções oferecem usporte essencial à lógica principal do sistema:

4.2.1 Limpeza de Tela

Limpa a tela do terminal com comandos compatíveis com o sistema operacional (Windows ou Unix).

```
void limpaTela()
```

4.2.2 Limpeza de Buffer

Remove caracteres residuais do buffer de entrada (stdin) para evitar leituras incorretas.

```
void limpaBuffer()
```

4.2.3 Tratamento de Erros

Garante que o valor inserido esteja dentro de um intervalo válido [ini, fim].

```
void entrada(int ini, int fim, int *num)
```

4.2.4 Alocação de Memória

Realoca memória para um vetor de vértices tridimensionais de forma segura.

```
int alocaVertices(struct no **vertices, int alocar)
```

4.2.5 Cálculo de Distância Euclidiana

Calcula a distância euclidiana entre dois pontos tridimensionais.

```
double calculaDE(struct no A, struct no B)
```

4.2.6 Normalização de Distância Euclidiana

Normaliza uma distância com base nos valores mínimos e máximos observados.

```
float normalizaDE(float x, float min, float max)
```

4.2.7 Alocação de Matriz

Aloca dinamicamente uma matriz quadrada de inteiros.

```
int **alocaMatriz(int tam)
```

4.2.8 Liberação de Memória de uma Matriz

Libera a memória de uma matriz quadrada alocada dinamicamente.

```
void liberaMatriz(int **matriz, int tam)
```

4.2.9 Conversão de Arquivo .csv para .txt

Abre um arquivo .csv e converte-o para um arquivo .txt para ser utilizado posteriormente para o plot do grafo em 3D.

```
int converteCSVparaTXT()
```

4.2.10 Implementação de DFS (Depth-First Search)

Realiza uma busca em profundidade a partir de um vértice inicial, marcando vértices visitados e calculando o tamanho do componente conexo.

```
void dfs(int v, int **matrizADJ, int *visitados, int totalVertices,
     int *tamanhoComponente)
```

4.2.11 Contagem de Componentes Conexos

Identifica e conta todos os componentes conexos do grafo, registrando a frequência de componentes por tamanho.

```
void contaComponentesConexos(int **matrizADJ, int totalVertices,
    Info_grafo *infos)
```

5 Compilação

A compilação do código é feita a partir do *GNU C Compiler (GCC)*. Para isso é preciso que o GCC esteja instalado em sua máquina.

5.1 Verificar se o GCC Está Instalado na Máquina

Utilize o comando:

```
gcc --version
```

A mensagem que o terminal devolver será o suficiente para saber se o GCC está instalado ou não. Caso não esteja instalado, utilize o gerenciador de pacotes da sua distribuição Linux e procure pelo pacote do GCC:

5.1.1 Para Distribuições Basedas em Debian

```
sudo apt install build-essential
```

5.1.2 Para Distribuições Basedas em Fedora

```
sudo dnf install make gcc gcc-c++
```

5.1.3 Para Distribuições Basedas em Arch

```
sudo pacman -S gcc
```

5.2 Instalação de Bibliotecas Utilizadas para Plotar o Grafo

Dentro da pasta do trabalho há um arquivo nomeado `requirements.txt`. Este arquivo possui o nome de todas as bibliotecas que são necessárias para o script de Python funcionar corretamente. Para instalar as bibliotecas você deve, pelo terminal, estando na pasta do trabalho, utilizar o seguinte comando:

```
pip install -r requirements.txt
```

Caso as bibliotecas já estejam instaladas, o pip irá mostrar uma mensagem informando que os requisitos já estão satisfeitos.

5.3 Compilação do Código

Uma vez instalados todos os requisitos, estando na pasta do arquivo fonte, basta utilizar o seguinte comando no terminal:

```
gcc trabalho.c -lm && ./a.out
```

Para que o código funcione corretamente, é essencial que:

1. Os arquivos `displayGrafo.py`, `grafoteste.csv` e `requirements.txt` estejam todos na **mesma** pasta, caso contrário o código não funcionará por erros de caminho.
2. Você tenha instalado o compilador GCC e a linguagem Python, incluindo as dependências citadas anteriormente.

5.4 O que Esperar

Logo após compilar, você irá se deparar com uma janela do terminal com 4 opções:

1. Criar grafo a partir de dataset de dados.
2. Carregar grafo gerado.
3. Plotar grafo 3D.
4. Sair.

Na opção 1 - Criar grafo a partir de dataset dados, você deverá informar o nome do arquivo .csv com os dados que serão utilizados como dataset para o programa. Para o dataset dado pelo professor basta colocar o nome `grafoteste.csv` e deverá aparecer uma mensagem de grafo carregado com sucesso. Logo após criar o grafo, na pasta do trabalho será criado um novo arquivo chamado `Grafo.csv` que é uma lista de arestas que indica quais vértices têm arestas entre si. No cabeçalho deste arquivo você irá encontrar:

- O total de vértices lidos;
- A maior Distâcia Euclidiana (DE) e o par de vértices (Vi, Vj) que a determinaram;
- A menor DE e o par de vértices (Vi, Vj) que a determinaram;
- A maior Distância Euclidiana Normalizada (DEN) e o par de vértices (Vi, Vj) que a determinaram;
- A menor DEN e o par de vértices (Vi, Vj) que a determinaram;
- Quantos componentes conexos e respectivos tamanhos (quantidade de vértices).

Na opção 3 - Plotar grafo 3D, o terminal irá carregar por alguns segundos e logo em seguida será aberta uma janela no navegador padrão da sua máquina com a visualização do grafo em 3D, podendo mover livremente o grafo para análise. Todas as opções, exceto a opção de sair, irão solicitar que você aperte enter para voltar ao menu principal do programa.