FACULDADE DE CIÊNCIAS E TECNOLOGIA

DEPARTAMENTO DE MATEMÁTICA E COMPUTAÇÃO

CIÊNCIA DA COMPUTAÇÃO

Juan Cardoso da Silva

Pedro Takahashi

**Trabalho de LFA – Parte 2**

Relatório e Manual do usuário.



PRESIDENTE PRUDENTE

2019

Sumário

[Sumário 2](#_Toc26030879)

[Como usar a aplicação 3](#_Toc26030880)

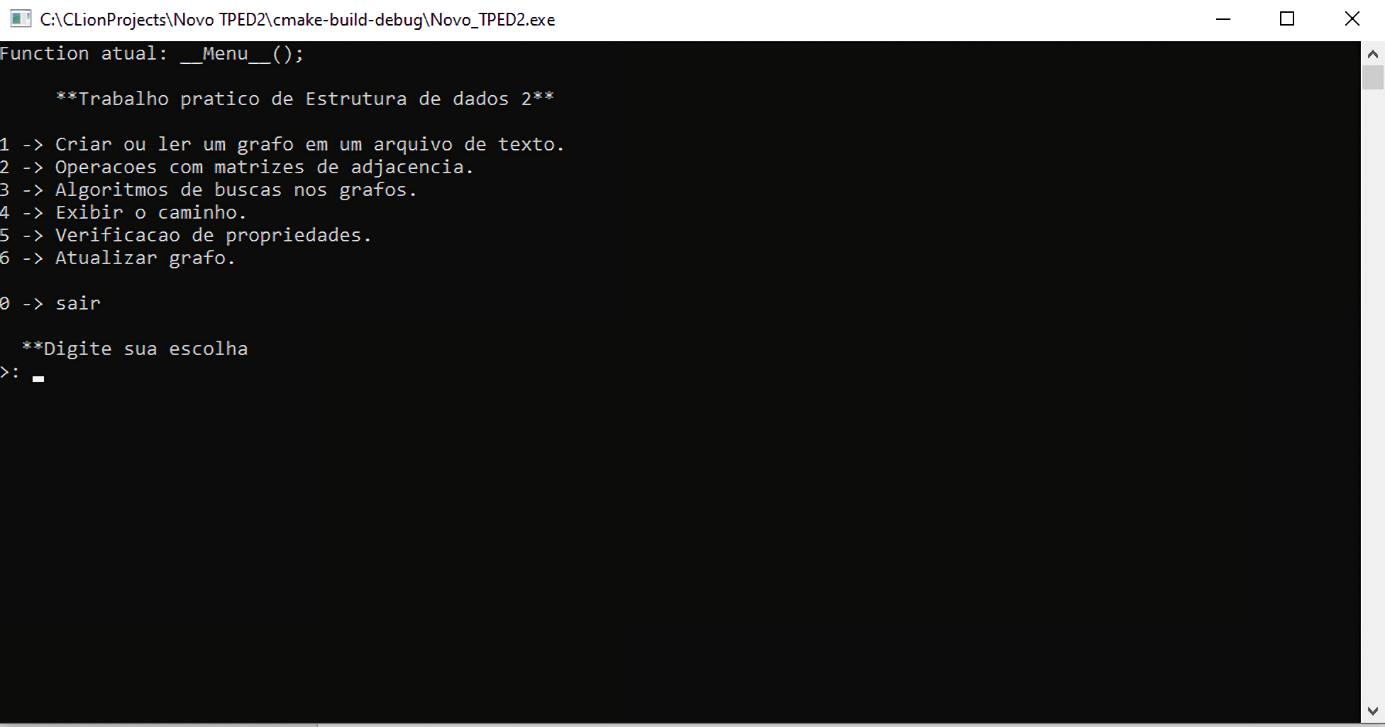
[Estruturação de arquivos 6](#_Toc26030881)

[Estrutura de dados utilizadas 7](#_Toc26030882)

[Explicação das buscas e dos caminhos 9](#_Toc26030883)

# **Como usar a aplicação**

A navegação pelo aplicativo do trabalho é realizada com números de zero a seis, cada número representa uma das opções que estarão disponíveis no menu de cada parte da aplicação, como na imagem abaixo.



Após digitar o número que representa sua escolha aperte enter parar que o dado seja lido, entrando na função que será responsável por realizar a operação da escolha.

Para realizar as manipulações de matrizes, caminhos, propriedades, buscas e atualização de grafo, recomenda-se ler um grafo para gerar a matriz de adjacência para realizar as manipulações ou ler uma matriz já escrita ( é possível ler e mostrar uma tabela para as buscas, mas o caminho depende que crie-se uma busca nova, sendo assim, a necessidade de uma matriz de adjacência).

***Atenção:***

**🡪Digitar dados para navegar no menu que não seja um dos números validos para a navegação vai resultar na aplicação parar de responder ou simplesmente entrar em loop infinito.**

🡪 **Ao realizar a leitura de um determinado tipo de arquivo de texto, leia o mesmo tipo e não tipos diferentes para evitar erros ou bugs durante a aplicação, exemplo: ler um arquivo do tipo “DFS-grafo.txt”, invés de um “grafo.txt”, o prefixo DFS no arquivo indica que ele é uma tabela da busca de profundidade, mais exemplos a respeito dessa estruturação estará disponível na parte de estruturação de arquivos.**

**Exemplo de navegação na aplicação:**

Vamos supor que queremos realizar uma busca por profundidade, primeiro precisamos criar um grafo ou ler uma matriz de adjacência pronta, digita-se 1 para acessar o sub menu de criação e leitura de grafos e 3 para entrar na função que realiza as leituras, escreva o nome o arquivo a ser lido corretamente ( exemplo *“digrafo.txt”*), o grafo será lido e a matriz de adjacência gerada logo em seguida, agora é possível realizar manipulações com matrizes, atualizar o grafo, verificar as propriedades e realizar as buscas, porem queremos apenas realizar uma busca por profundidade, digitaremos 3 para entrar no sub menu e 1 para entrar na função da busca por profundidade, aparecerá 4 opções, digitares 3 para realizara busca, o aplicativo pedira a origem, digita-se a origem e aperta enter para computar o input e para continuar, depois digita-se 4 para mostrar a tabela gerada (é possível salvar essa tabela também).

**Estruturação dos menus**

\_\_Menu\_\_()

|  
 |=> Leitura e escrita de grafos

|=> Leitura, escrita e visualização de matrizes de adjacência

|=> Algoritmos de buscas

| |=> Busca em profundidade (DFS)

| | |=> Leitura, escrita e exibição de tabelas.

| | |=> Criar a tabela depois da busca.

| |=> Busca em Largura (BFS)

| | |=> Leitura, escrita e exibição de tabelas.

| | |=> Criar a tabela depois da busca.

|

|=> Exibir o caminho

| |=> Caminho DFS

| |=> Caminho BFS

|

|=> Verificação de propriedades do grafo.

|=> Atualizar o grafo.

|

|=> Sair da aplicação

***O fluxo das funções e a estruturação das mesmas podem ser vistas no log em arquivo de texto e o log só é criado quando sai da aplicação pelo menu principal.***

# **Estruturação de arquivos**

**🡪Leitura de grafos (dígrafo ou grafo) podem são feitas apenas pelo primeiro menu (“*1 -> Criar ou Ler um grafo em um arquivo de texto”*).**

**🡪Leitura de matrizes de adjacência são apelas pela opção de manipulação de matrizes de adjacência e seus arquivos tem um tipo próprio com sufixo .mt , ou seja para ler uma matriz deve digitar o nome do arquivo “*nome\_arquivo.mt*”.**

**🡪A leitura ou escrita das buscas em profundidade e largura possuem o prefixo DFS e BFS respectivamente, para diferenciar dos outros arquivos de texto.**

**🡪Para mostrar uma tabela, deve-se ler ela ou criar uma busca a partir de uma matriz de adjacência.**

# **Estrutura de dados utilizadas**

**typedef struct** info\_\_matriz{  
 **int** grafo\_saida; // Aonde o vértice vai.  
 **int** grafo\_chegada; // De onde o vértice sai.  
 **int** peso; // Peso da aresta.  
 **int** tipo; // tipo do grafo.  
}tm;

// A info\_matriz é utilizada para leitura dos grafos no

//formato de arquivo //de texto para auxiliar na criação das //matrizes de adjacência.

**typedef struct** matriz {  
 **int** indice[30]; // Aonde o nome do vértice fica  
 **int** m[30][30]; // Matriz de adjacência  
 **int** peso[30][30]; // Matriz dos pesos.  
 **int** tamanho; // Tamanho do grafo  
 **int** visitado[30]; // Variável não utilizada  
 **int** contem\_peso; // Variável o não utilizada  
 **int** origem; // Determina a origem de uma busca  
 **int** numero\_vertices; // Variável não utilizada   
 **int** numero\_arestas; // Variável não utilizada  
 **int** tipo; // Tipo da matriz de adjacência  
 **char** classificacao\_grafo[50]; // classificação do grafo  
  
}ma;  
  
// A Matriz é a estrutura principal aonde a maioria das

//manipulações vão utilizar para gerar os resultados e

//verificações.

**typedef struct** auxiliar\_matriz {  
 **int** indice[30];  
 **int** m[30][30];  
 **int** peso[30][30];  
 **int** tamanho;  
 **int** visitado[30];  
 **int** contem\_peso;  
 **int** origem;  
 **int** numero\_vertices;  
 **int** numero\_arestas;  
 **int** tipo;  
 **char** classificacao\_grafo[50];  
  
}aux\_ma;

// Não muito o que explicar da auxiliar\_matriz (uma cópia da //estrutura matriz) a não ser que ela é uma estrutura de dados //que realiza as operações destrutivas dos algoritmos,

//deixando a estrutura matriz intacta.

**typedef struct** DFS\_data{  
 **int** origem; // Variável para a origem da busca  
 **int** tamanho; // Variável para o tamanho do índice.  
 **int** tempo; // Variável que armazena o tempo  
 **int** indice[30]; // Variável para os índices(vértices)  
 **char** indice\_cor[30]; // Variável para lembrar as cores.  
 **int** descoberta[30]; // Armazena o tempo de descoberta  
 **int** finalizacao[30]; // Armazena o tempo de finalização  
 **int** caminho[30]; // Guarda os vértices visitados.  
  
 **int** dfs\_esta\_criada; // Variável não utilizada.  
}dfs\_struct;  
  
**typedef struct** BFS\_data{  
 **int** origem; // Variável para a origem da busca  
 **int** tamanho; // Variável para o tamanho do índice.  
 **int** tempo; // Variável que armazena o tempo  
 **int** vertice[30]; // Variável para os índices(vértices)  
 **int** pai[30]; // Variável que armazena o pai.  
 **char** indice\_cor[30]; // Variável para lembrar as cores.  
 **int** distancia[30]; // Guarda a distância entre 2 vértice  
 **int** caminho[30]; // Guarda os vértices visitados.  
  
 **int** bfs\_esta\_criada; // Variável não utilizada.  
}bfs\_struct;

// Ambas as tabelas são utilizadas para realização da busca

//(tanto por profundidade quanto por largura), leitura de uma

//tabela de texto e visualizar uma tabela lida ou gerada.

# **Explicação das buscas e dos caminhos**

A busca por profundidade pega o ponteiro para a matriz auxiliar e o ponteiro para a tabela dfs e realiza recursivamente, a busca por profundidade, primeiramente visitando os vértices brancos e depois colocando-os como cinzas adiciona o tempo no vetor de tempo e incrementa para a próxima iteração, se o ponteiro da matriz( que aponta para uma matriz **m[n][n]**) na posição **u** e **v**, for igual a **1** (existe uma aresta ligando **u** e **v**), entra na função que realiza a busca novamente, até percorrer todos só vértices que tem arestas, depois recursivamente pinta os vértices de preto e continua a somar o tempo, e coloca-o na finalização, como no código abaixo:

**void** visit\_dfs(**struct** auxiliar\_matriz \*aux2\_matriz, **int** origem, dfs\_struct \*dfs\_tabela){  
 dfs\_tabela->indice\_cor[origem] = **'c'**;  
 dfs\_tabela->tempo = dfs\_tabela->tempo + 1;  
 dfs\_tabela->descoberta[origem] = dfs\_tabela->tempo;  
  
 dfs\_caminho[contador\_caminho] = origem;  
 dfs\_tabela->caminho[contador\_caminho] = origem;  
  
 **int** v = 0;  
 **for**(v = 0; v < aux2\_matriz->tamanho; v++){  
 **if**(aux2\_matriz->m[origem][v] == 1 && dfs\_tabela->indice\_cor[v] == **'b'**){  
  
 **if**(debug == **true**)  
 printf(**"-> origem:%d v:%d cor:%c\n"**,origem,v,dfs\_tabela->indice\_cor[v]);  
  
 contador\_caminho++;  
 visit\_dfs(aux2\_matriz, v, dfs\_tabela, debug);  
 }  
 }  
  
 dfs\_tabela->indice\_cor[origem] = **'p'**;  
 dfs\_tabela->tempo = dfs\_tabela->tempo + 1;  
 dfs\_tabela->finalizacao[origem] = dfs\_tabela->tempo;  
}

O caminho é adicionado conforme os vertices são visitados, a origem é adicionada ao vetor de caminho e é incrementado ao entrar na verificação da recursão.

A busca em profundidade pega o ponteiro para a matriz auxiliar e o ponteiro para uma tabela do tipo bfs, e realiza iterativamente a busca em largura. Primeiro inicializa as variáveis da tabela antes e entrar no **while** responsável pela iteração, logo depois no **while** ( que só para se a fila não estivar vazia ), uma variável **v** recebe o vértice de retorno da função de **deletar\_fila()** ( do qual retorna uma variável enfileirada), pinta de preto o vertice na posição **v**, adiciona **v** para o vetor de caminho e incremente o vetor para próxima iteração, depois entra em um **for** de verificação na posição da matriz **m[n][n]** na posição **v** e **i**( o **i** é da iteração do **for**), se na posição **v** e **i** for igual a 1 e o vértice for branco, enfileira o valor de **i**, printa o vértice de cinza e incrementa o vetor de distancia e adiciona no vetor de pai na posição **i**, o valor de **v**.

**void** BFS(**int** v, **struct** auxiliar\_matriz \*aux2\_matriz, **struct** BFS\_data \*bfs\_tabela, **bool** debug) {  
 **int** i;  
 **int** contador = 0;  
 **int** distancia = 0;  
  
 inserir\_fila(v);  
 bfs\_tabela->indice\_cor[v] = **'c'**;  
 bfs\_tabela->distancia[v] = distancia;  
 bfs\_tabela->pai[v] = -1;**while**(!fila\_esta\_vazia()) {  
 v = deletar\_fila();bfs\_tabela->indice\_cor[v] = **'p'**;bfs\_tabela->caminho[contador] = v;  
 contador++;**for**(i=0; i< aux2\_matriz->tamanho; i++) {  
 **if**(aux2\_matriz->m[v][i] == 1 &&

bfs\_tabela->indice\_cor[i] == **'b'**) {

inserir\_fila(i);  
  
 bfs\_tabela->indice\_cor[i] = **'c'**;  
 bfs\_tabela->distancia[i] = bfs\_tabela->distancia[v] + 1;  
 bfs\_tabela->pai[i] = v;}  
 }  
 }  
}