

UNIVERSIDADE ESTADUAL PAULISTA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
CIÊNCIA DA COMPUTAÇÃO

LEANDRO HENRIQUE LIMA E SILVA

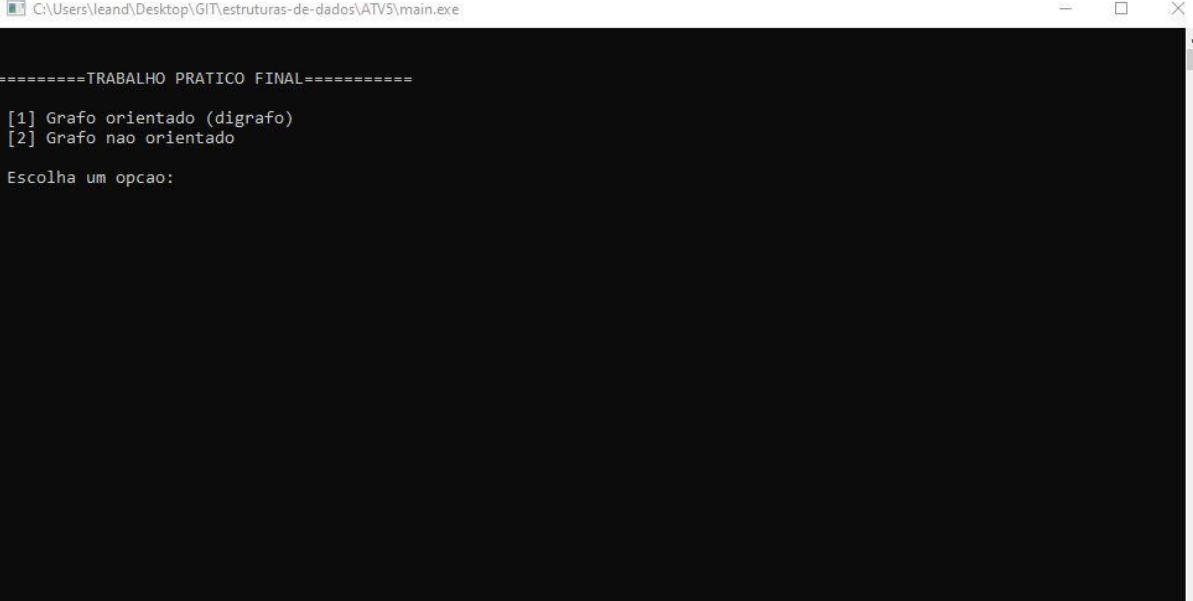
TRABALHO PRÁTICO FINAL

Documentação referente a implementação de programa para manipulação de grafos

1. Uso do programa

O programa desenvolvido foi criado com o objetivo de manipular grafos orientados e não orientados, sendo possível escolher numa implementação de matriz de adjacência ou lista de adjacência.

Para executar o programa, é necessário rodar o arquivo main.exe, que vai te mostrar a seguinte tela:



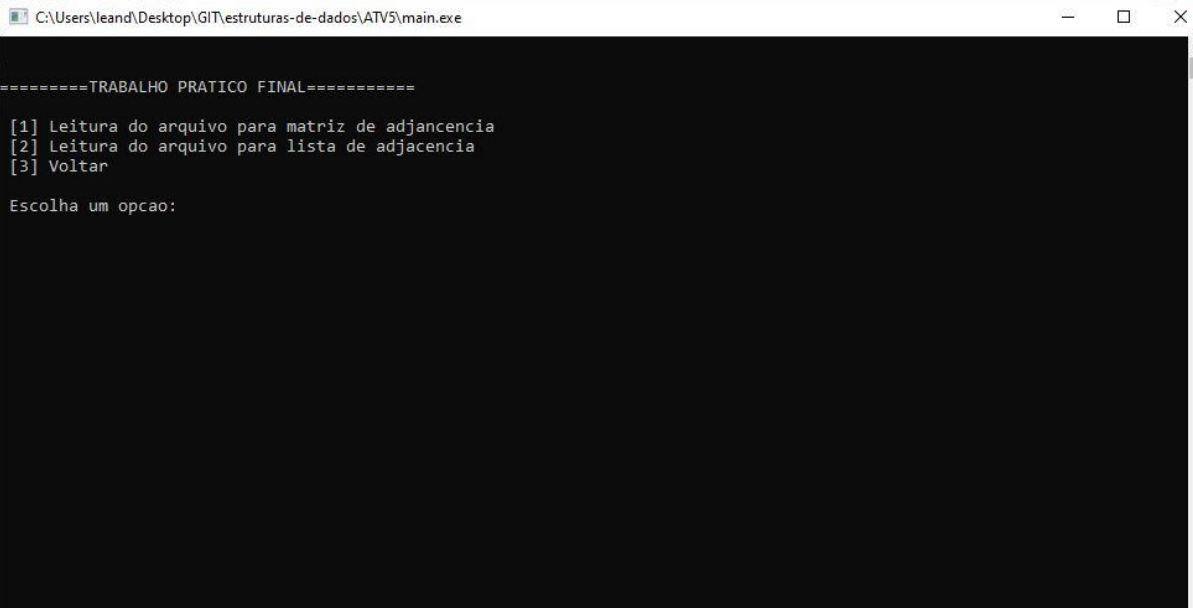
```
C:\Users\leand\Desktop\GIT\estruturas-de-dados\ATIV5\main.exe

=====TRABALHO PRATICO FINAL=====

[1] Grafo orientado (digrafo)
[2] Grafo nao orientado

Escolha um opcao:
```

Esta é a primeira página da aplicação, que contém o primeiro menu, criado para que o usuário escolha se ele deseja manipular um Grafo Orientado (digrafo) ou um Grafo Não Orientado.



```
C:\Users\leand\Desktop\GIT\estruturas-de-dados\ATIV5\main.exe

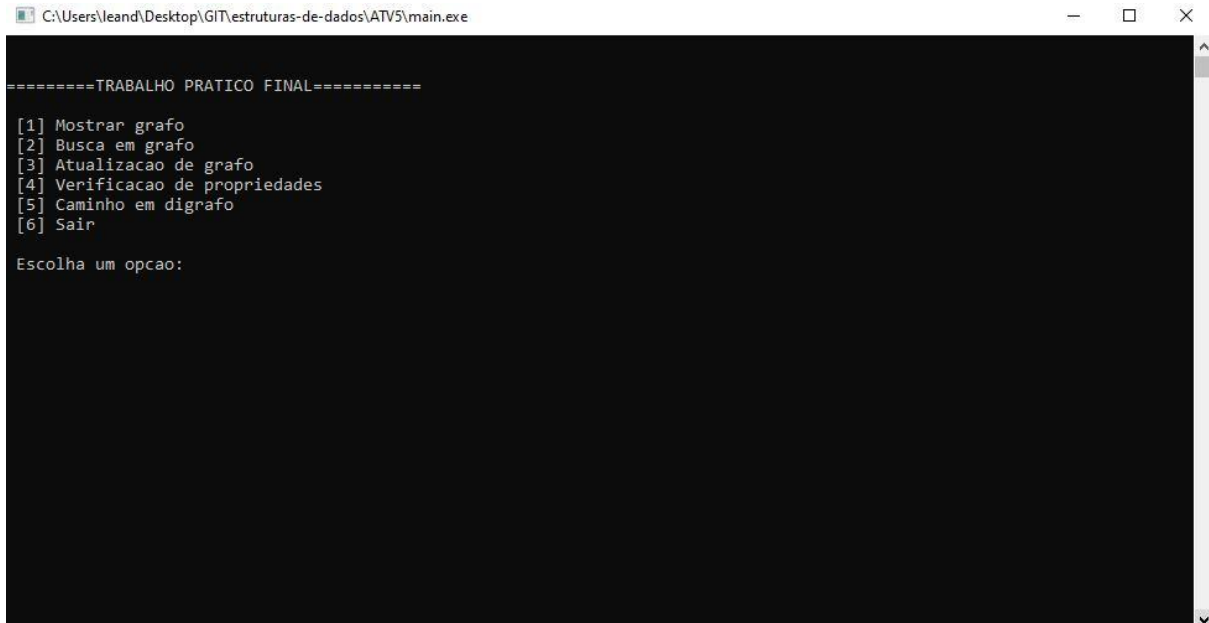
=====TRABALHO PRATICO FINAL=====

[1] Leitura do arquivo para matriz de adjacencia
[2] Leitura do arquivo para lista de adjacencia
[3] Voltar

Escolha um opcao:
```

Após escolher qual tipo de grafo ele deseja manipular, é necessário escolher em qual representação computacional será armazenado o grafo.

- Matriz de Adjacência;
- Lista de Adjacência.



```
C:\Users\leand\Desktop\GIT\estruturas-de-dados\ATIV5\main.exe

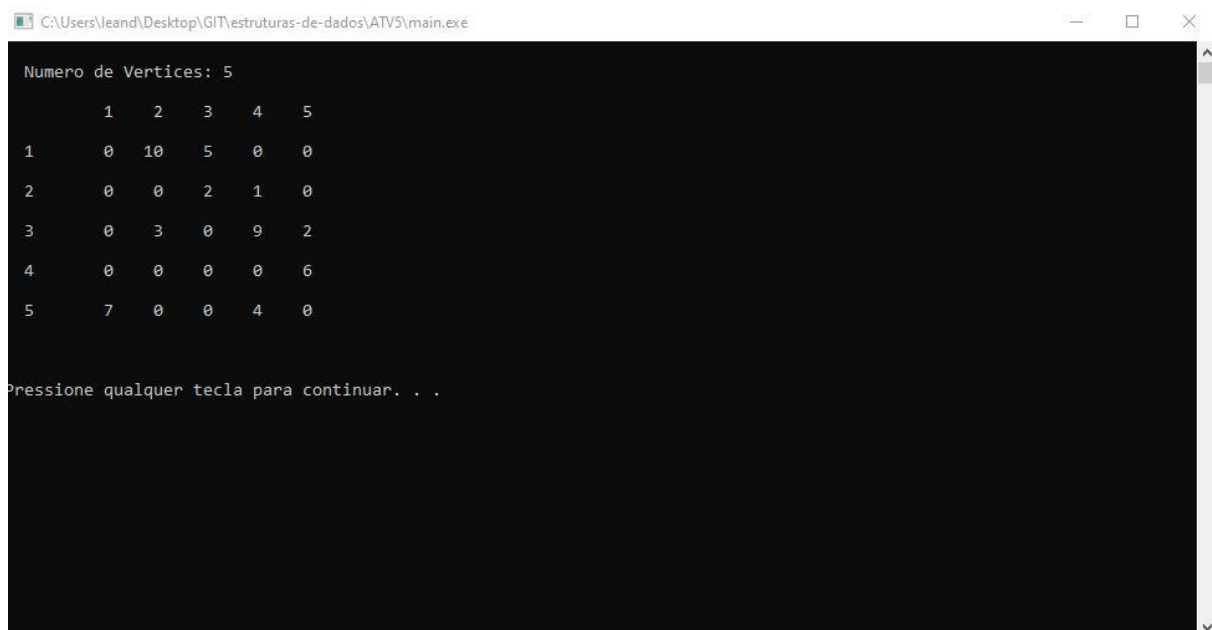
=====TRABALHO PRATICO FINAL=====

[1] Mostrar grafo
[2] Busca em grafo
[3] Atualizacao de grafo
[4] Verificacao de propriedades
[5] Caminho em digrafo
[6] Sair

Escolha um opcao:
```

Deste modo, tem-se agora o menu principal. Este menu contém todas as operações possíveis a serem realizadas no grafo.

1) Mostrar grafo



```
C:\Users\leand\Desktop\GIT\estruturas-de-dados\ATIV5\main.exe

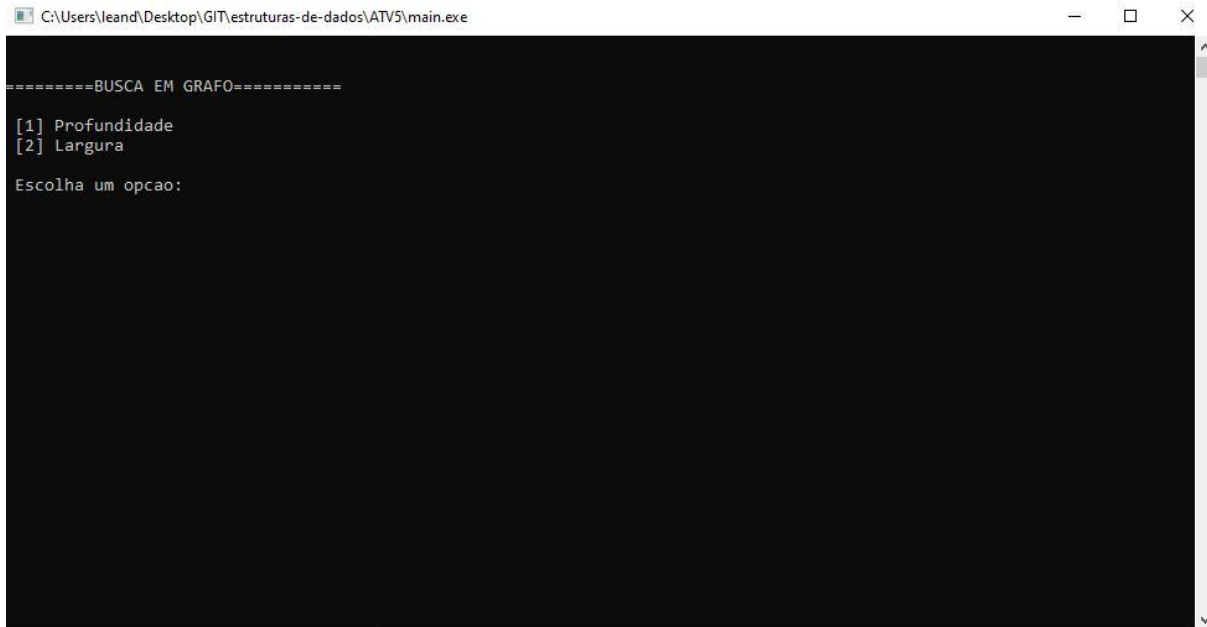
Numero de Vertices: 5

      1   2   3   4   5
1      0  10   5   0   0
2      0   0   2   1   0
3      0   3   0   9   2
4      0   0   0   0   6
5      7   0   0   4   0

Pressione qualquer tecla para continuar. . .
```

Mostra o grafo de acordo com a representação computacional definida. Neste caso, temos um um grafo orientado em uma matriz de adjacência.

2) Busca em grafo

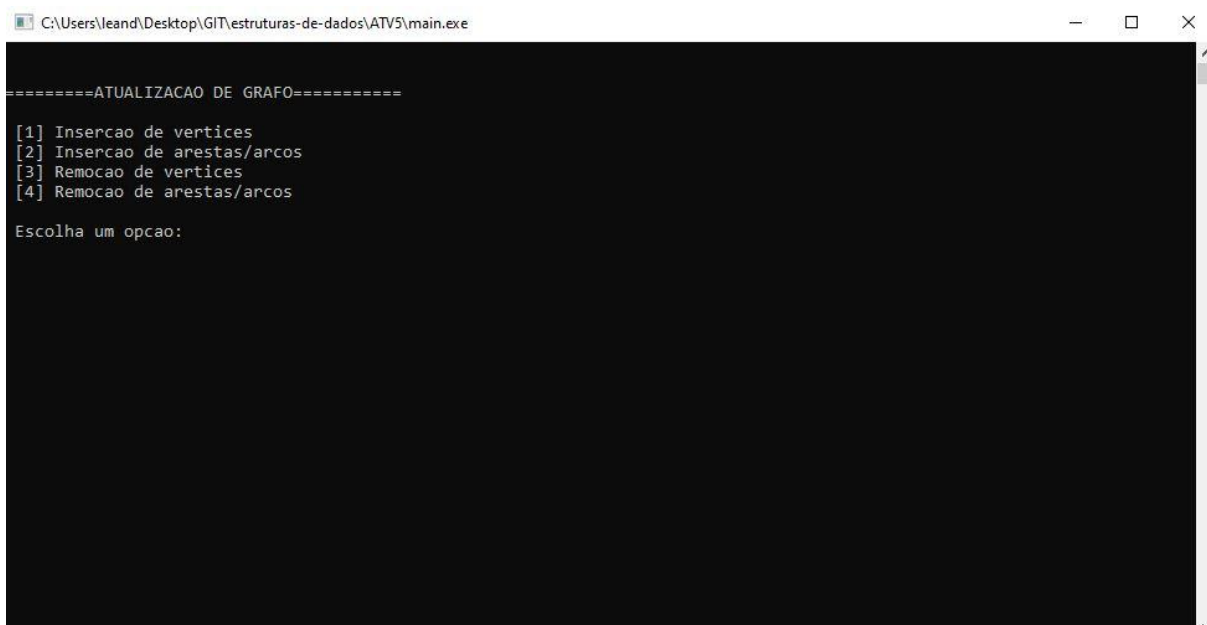


```
====BUSCA EM GRAFO====
[1] Profundidade
[2] Largura
Escolha um opcao:
```

Neste caso, é possível escolher qual busca o usuário deseja realizar.

- Profundidade;
- Largura.

3) Atualização de grafo

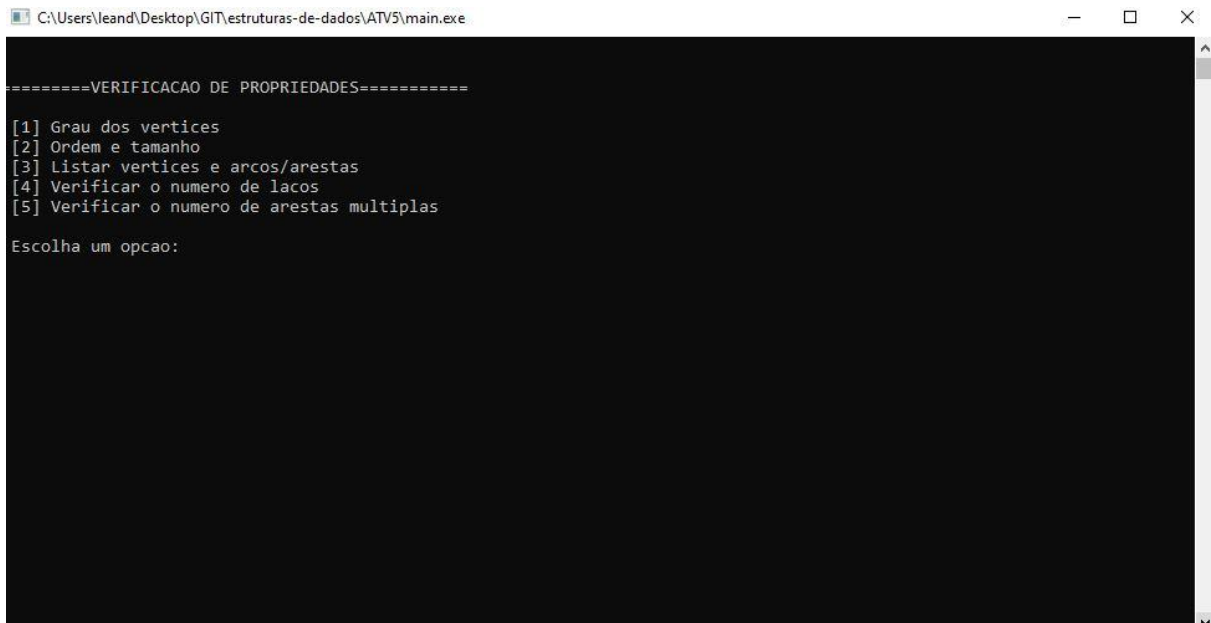


```
====ATUALIZACAO DE GRAFO====
[1] Insercao de vertices
[2] Insercao de arestas/arcos
[3] Remocao de vertices
[4] Remocao de arestas/arcos
Escolha um opcao:
```

Aqui é possível realizar as seguintes alterações/atualizações no grafo:

- Inserção de vértices;
- Inserção de arestas/arcos;
- Remoção de vértices;
- Remoção de arestas/arcos.

4) Verificação de propriedades



```
C:\Users\leand\Desktop\GIT\estruturas-de-dados\ATV5\main.exe

=====VERIFICACAO DE PROPRIEDADES=====

[1] Grau dos vertices
[2] Ordem e tamanho
[3] Listar vertices e arcos/arestas
[4] Verificar o numero de lacos
[5] Verificar o numero de arestas multiplas

Escolha um opcao:
```

É possível também realizar as seguintes verificações de propriedades do grafo.

- Grau dos vértices;
- Ordem e tamanho;
- Listar vértices e arcos/aresta;
- Verificar o número de laços;
- Verificar o número de arestas múltiplas.

5) Caminho em dígrafo

6) Sair



Nesta opção, a aplicação encerra a manipulação do grafo e pergunta para o usuário se ele deseja:

- Continuar no programa;
- Fechar o programa.

2. Implementação

2.1. Softwares utilizados

- Codeblocks: Desenvolver e compilar a aplicação;
- Google Docs: Para criação da documentação;
- Google Meet: Para gravar o vídeo.

2.2. Status dos processamentos

1. Log do processamento
 - a. Registro, em arquivo texto, do nome das funções à medida que são chamadas -> Funcionando integralmente.
2. Grafo armazenado em arquivo texto.
 - a. Formatação vista em aula -> Funcionando integralmente.
 - b. Leitura do arquivo para matriz de adjacências -> Funcionando integralmente.
 - c. Leitura do arquivo para lista de adjacências -> Funcionando integralmente.
 - d. Escrita a partir da matriz de adjacências para um arquivo -> Funcionando integralmente.
 - e. Escrita a partir da lista de adjacências para um arquivo -> Funcionando integralmente.
3. Implementação de busca em grafo (Profundidade e Largura).
 - a. Escrita, em arquivo texto, da tabela gerada na busca -> Funcionando integralmente.
 - b. Listar os caminhos da raiz/origem para os demais vértices -> Funcionando integralmente.
4. Atualização de grafo.
 - a. Inserção de vértices -> Funcionando integralmente.
 - b. Inserção de arestas/arcos -> Funcionando integralmente.
 - c. Remoção de vértices -> Não está funcionando, pois não consegui implementar.
 - d. Remoção de arestas/arcos -> Funcionando integralmente.

5. Verificação de propriedades.
 - a. Grau dos vértices em grafos orientados e não orientados -> Funcionando parcialmente, pois no caso de grafos não orientados em listas de adjacência, há um problema em detectar arestas múltiplas.
 - b. Ordem e tamanho -> Funcionando integralmente.
 - c. Listar vértices e arcos/arestas -> Funcionando parcialmente, pois no caso de grafos não orientados em listas de adjacência, há um problema em mostrar arestas múltiplas.
 - d. Verificar o número de laços -> Funcionando integralmente.
 - e. Verificar o número de arestas múltiplas -> Funcionando parcialmente, pois no caso de grafos não orientados em listas de adjacência, há um problema em detectar arestas múltiplas.
6. Caminho em dígrafo -> Não está funcionando, pois não consegui implementar.

2.3. Matriz de adjacências

Foi utilizado a estrutura *struct* para armazenar os dados referentes ao grafo. A escolha é devido a ser um tipo de dados composto que agrupa uma lista de variáveis. Também foi utilizado o comando *typedef*, que define um novo nome para um determinado tipo de dado.

```
typedef struct{
    int tipoGrafo; //0 para dígrafo e 1 se grafo não orientado
    int numVertices; //número de vértices do grafo
    int numArcos; //se o grafo for um dígrafo, esta variável será utilizada
    int numArestas; //se for um grafo não orientado, esta variável será utilizada
    int **matrix; //armazena a matriz de adjacência
} GraphMatrix;
```

Então, foi implementado a *struct* que foi definida com o nome “GraphMatrix” pelo comando *typedef*. Seus campos são:

- `int tipoGrafo`: um inteiro que armazena 0 se é um grafo orientado (dígrafo) ou 1 se é um grafo não orientado;
- `int numVertices`: um inteiro que armazena o número de vértices do grafo;
- `int numArcos`: um inteiro que armazena o número de arcos do grafo, caso ele seja um grafo orientado;
- `int numArestas`: um inteiro que armazena o número de arestas do grafo, caso ele seja um grafo não orientado;
- `int **matrix`: o campo é um ponteiro para a matriz de adjacência do grafo que será alocada dinamicamente, ou seja, enquanto o programa estiver sendo executado.

Funcionamento na prática: Grafo Orientado.

C:\Users\leand\Desktop\GIT\estruturas-de-dados\ATV5\main.exe

```
Numero de Vertices: 5

      1   2   3   4   5
1      0  10   5   0   0
2      0   0   2   1   0
3      0   3   0   9   2
4      0   0   0   0   6
5      7   0   0   4   0

Pressione qualquer tecla para continuar. . .
```

Funcionamento na prática: Grafo Não Orientado.

C:\Users\leand\Desktop\GIT\estruturas-de-dados\ATV5\main.exe

```
Numero de Vertices: 5

      1   2   3   4   5
1      0   1   1   0   1
2      1   0   2   1   0
3      1   2   1   1   1
4      0   1   1   0   2
5      1   0   1   2   0

Pressione qualquer tecla para continuar. . .
```

2.4. Lista de adjacências.

Foi utilizado a estrutura *struct* para armazenar os dados referentes ao grafo. A escolha é devido a ser um tipo de dados composto que agrupa uma lista de variáveis. Também foi utilizado o comando *typedef*, que define um novo nome para um determinado tipo de dado.

```
typedef struct no *No;

struct no{
    int vertice;
    int peso;
    No next;
};

typedef struct {
    int tipoGrafo; //0 para dígrafo e 1 se grafo não orientado
    int numVertices; //número de vértices do grafo
    int numArcos; //se o grafo for um dígrafo, esta variável será utilizada
    int numArestas; //se for um grafo não orientado, esta variável será utilizada
    No *lista; //armazena a lista de adjacência
} GraphList;
```

Primeiro foi utilizado o *typedef* para definir o nome “No” para o ponteiro que aponta para a *struct no*. Após isso, foi criada a *struct no* que armazena os dados referentes a cada nó da lista de adjacência.

- int vértice: armazena o vértice deste nó.
- int peso: armazena o peso que tem a aresta/arco que liga a este vértice.
- No next: o campo é um ponteiro que liga ao próximo nó.

Então, foi implementado a *struct* que foi definida com o nome “GraphList” pelo comando *typedef*. Seus campos são:

- int tipoGrafo: um inteiro que armazena 0 se é um grafo orientado (dígrafo) ou 1 se é um grafo não orientado;
- int numVertices: um inteiro que armazena o número de vértices do grafo;
- int numArcos: um inteiro que armazena o número de arcos do grafo, caso ele seja um grafo orientado;
- int numArestas: um inteiro que armazena o número de arestas do grafo, caso ele seja um grafo não orientado;

- No *lista: o campo é um ponteiro para a lista de adjacência do grafo que será alocada dinamicamente, ou seja, enquanto o programa estiver sendo executado.

Funcionamento na prática: Grafo Orientado.

```
C:\Users\leand\Desktop\GIT\estruturas-de-dados\ATV5\main.exe
Numero de Vertices: 5

LISTA
| 1 | -> 3,5  -> 2,10  ->
| 2 | -> 4,1  -> 3,2   ->
| 3 | -> 5,2  -> 4,9   -> 2,3  ->
| 4 | -> 5,6  ->
| 5 | -> 4,4  -> 1,7   ->

Pressione qualquer tecla para continuar. . .
```

Funcionamento na prática: Grafo Não Orientado.

```
C:\Users\leand\Desktop\GIT\estruturas-de-dados\ATV5\main.exe
Numero de Vertices: 5

LISTA
| 1 | -> 5,0  -> 3,0  -> 2,0  ->
| 2 | -> 3,0  -> 4,0  -> 1,0  ->
| 3 | -> 3,0  -> 2,0  -> 5,0  -> 4,0  -> 1,0  ->
| 4 | -> 5,0  -> 3,0  -> 2,0  ->
| 5 | -> 4,0  -> 3,0  -> 1,0  ->

Pressione qualquer tecla para continuar. . .
```

3. Funções

Funções utilizadas para manipulação da matriz de adjacência:

void graphMatrixView(FILE *log, GraphMatrix *graph);

//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//GraphMatrix *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a matriz de adjacência.

GraphMatrix* graphMatrixReadArq(FILE *log, FILE *arq);

//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//FILE *arq -> arquivo que contém os dados do grafo orientado ou não orientado.

void insertArqOrEdgesToGraphMatrix(FILE *log, FILE *arq, GraphMatrix *graph);

//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//FILE *arq -> arquivo que contém os dados do grafo orientado ou não orientado.

//GraphMatrix *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a matriz de adjacência.

void graphMatrixWriteArq(FILE *log, GraphMatrix *graph);

//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//GraphMatrix *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a matriz de adjacência.

void graphMatrixWriteArqDFS(FILE *log, GraphMatrix *graph, int *d, int *f);

//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//GraphMatrix *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a matriz de adjacência.

//int *d -> um ponteiro para uma lista que armazena o tempo de descoberta de cada vértice

//int *f -> um ponteiro para uma lista que armazena o tempo de finalização de cada vértice

void graphMatrixDFSVISIT(FILE *log, GraphMatrix *graph, int vertex, char *cor, int *d, int *f, int *tempo);

//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//GraphMatrix *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a matriz de adjacência.

//int vertex -> indica o vértice que está sendo visitado

//int *cor -> um ponteiro para uma lista que armazena a cor de cada vértice

//int *d -> um ponteiro para uma lista que armazena o tempo de descoberta de cada vértice

//int *f -> um ponteiro para uma lista que armazena o tempo de finalização de cada vértice

//int *tempo -> um ponteiro para uma variável inteira que armazena o tempo

void graphMatrixDFS(FILE *log, GraphMatrix *graph);

//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//GraphMatrix *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a matriz de adjacência.

int* newFila(FILE *log, int n);

//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//int n -> tamanho da fila

void enqueue(FILE *log, int *Q, int vertex, int n);

//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//int *Q -> um ponteiro para uma lista que armazena a fila

//int vertex -> indica o vértice que será adicionado à fila

//int n -> tamanho da fila

int dequeue(FILE *log, int *Q, int n);

//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//int *Q -> um ponteiro para uma lista que armazena a fila

//int n -> tamanho da fila

short vazia(FILE *log, int *Q, int n);

//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//int *Q -> um ponteiro para uma lista que armazena a fila

//int n -> tamanho da fila

void graphMatrixWriteArqBFS(FILE *log, GraphMatrix *graph, int raiz, int *d, int *f);

//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//GraphMatrix *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a matriz de adjacência.

//int raiz -> raiz da busca

//int *d -> um ponteiro para uma lista que armazena a distância de cada vértice a origem(raiz)

//int *f -> um ponteiro para uma lista que armazena o vértice predecessor de cada vértice

void graphMatrixBFS(FILE *log, GraphMatrix *graph, int raiz);

//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//GraphMatrix *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a matriz de adjacência.

//int raiz -> raiz da busca

GraphMatrix* graphMatrixInsertVertex(FILE *log, GraphMatrix *graph, int qtd);

//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//GraphMatrix *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a matriz de adjacência.

//int qtd -> indica a quantidade de vértices que serão inseridos no grafo

void graphMatrixInsertArc(FILE *log, GraphMatrix *graph, int ini, int fim, int peso);

//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//GraphMatrix *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a matriz de adjacência.

//int ini -> indica o vértice inicial do arco

//int fim -> indica o vértice final do arco

//int peso -> indica o peso do arco

void graphMatrixInsertEdges(FILE *log, GraphMatrix *graph, int ini, int fim, int peso);

//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//GraphMatrix *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a matriz de adjacência.

//int ini -> indica o vértice inicial da aresta

//int fim -> indica o vértice final do aresta

//int peso -> indica o peso do arco

void graphMatrixRemoveVertex(FILE *log, GraphMatrix *graph, int vertex);

//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//GraphMatrix *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a matriz de adjacência.

//int vertex -> indica o vértice que deve ser removido

void graphMatrixRemoveArc(FILE *log, GraphMatrix *graph, int ini, int fim);

//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//GraphMatrix *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a matriz de adjacência.

//int ini -> indica o vértice inicial da aresta

//int fim -> indica o vértice final do aresta

void graphMatrixRemoveEdges(FILE *log, GraphMatrix *graph, int ini, int fim);

//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//GraphMatrix *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a matriz de adjacência.

//int ini -> indica o vértice inicial da aresta

//int fim -> indica o vértice final do aresta

void grauVertexGraphMatrixOrd(FILE *log, GraphMatrix *graph);

//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//GraphMatrix *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a matriz de adjacência.

void grauVertexGraphMatrixNOOrd(FILE *log, GraphMatrix *graph);

//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//GraphMatrix *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a matriz de adjacência.

void orderAndSizeGraphMatrix(FILE *log, GraphMatrix *graph);

//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//GraphMatrix *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a matriz de adjacência.

void listVertexAndArcGraphMatrix(FILE *log, GraphMatrix *graph);

//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//GraphMatrix *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a matriz de adjacência.

void listVertexAndEdgesGraphMatrix(FILE *log, GraphMatrix *graph);

//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//GraphMatrix *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a matriz de adjacência.

void checkLacesGraphMatrix(FILE *log, GraphMatrix *graph);

//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//GraphMatrix *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a matriz de adjacência.

void checkMultipleEdgesGraphMatrix(FILE *log, GraphMatrix *graph);

//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//GraphMatrix *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a matriz de adjacência.

Funções utilizadas para manipulação da lista de adjacência:

void graphListView(FILE *log, GraphList *graph);

//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//GraphList *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a lista de adjacência.

GraphList* graphListReadArq(FILE *log, FILE *arq);

//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//FILE *arq -> arquivo que contém os dados do grafo orientado ou não orientado.

void insertArqOrEdgesToGraphList(FILE *log, FILE *arq, GraphList *graph);

//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//FILE *arq -> arquivo que contém os dados do grafo orientado ou não orientado.

//GraphList *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a lista de adjacência.

void graphListWriteArq(FILE *log, GraphList *graph);

//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//GraphList *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a lista de adjacência.

void graphListWriteArqDFS(FILE *log, GraphList *graph, int *d, int *f);

//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//GraphList *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a lista de adjacência.

//int *d -> um ponteiro para uma lista que armazena o tempo de descoberta de cada vértice

//int *f -> um ponteiro para uma lista que armazena o tempo de finalização de cada vértice

void graphListDFSVISIT(FILE *log, GraphList *graph, int vertex, char *cor, int *d, int *f, int *tempo);

//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//GraphList *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a lista de adjacência.

//int vertex -> indica o vértice que está sendo visitado

//int *cor -> um ponteiro para uma lista que armazena a cor de cada vértice

//int *d -> um ponteiro para uma lista que armazena o tempo de descoberta de cada vértice

//int *f -> um ponteiro para uma lista que armazena o tempo de finalização de cada vértice

//int *tempo -> um ponteiro para uma variável inteira que armazena o tempo

void graphListDFS(FILE *log, GraphList *graph);

//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//GraphList *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a lista de adjacência.

void graphListWriteArqBFS(FILE *log, GraphList *graph, int raiz, int *d, int *f);

//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//GraphList *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a lista de adjacência.

//int raiz -> raiz da busca

//int *d -> um ponteiro para uma lista que armazena a distância de cada vértice a origem(raiz)

//int *f -> um ponteiro para uma lista que armazena o vértice predecessor de cada vértice

void graphListBFS(FILE *log, GraphList *graph, int raiz);

//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//GraphList *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a lista de adjacência.

//int raiz -> raiz da busca

GraphList* graphListInsertVertex(FILE *log, GraphList *graph, int qtd);

//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//GraphList *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a lista de adjacência.

//int qtd -> indica a quantidade de vértices que serão inseridos no grafo

void insertArqOrEdgesGraphList(FILE *log, GraphList *graph, int ini, int fim, int peso);

//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//GraphList *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a lista de adjacência.

//int ini -> indica o vértice inicial do arco/aresta

//int fim -> indica o vértice final do arco/aresta

//int peso -> indica o peso do arco

void graphListRemoveVertex(FILE *log, GraphList *graph, int vertex);

//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//GraphList *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a lista de adjacência.

//int vertex -> indica o vértice que deve ser removido

void graphListRemoveArcOrEdges(FILE *log, GraphList *graph, int ini, int fim);
//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)
//GraphList *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a lista de adjacência.
//int ini -> indica o vértice inicial da arco/aresta
//int fim -> indica o vértice final do arco/aresta

void grauVertexGraphListOrd(FILE *log, GraphList *graph);
//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)
//GraphList *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a lista de adjacência.

void grauVertexGraphListNOrd(FILE *log, GraphList *graph);
//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)
//GraphList *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a lista de adjacência.

void orderAndSizeGraphList(FILE *log, GraphList *graph);
//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)
//GraphList *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a lista de adjacência.

void listVertexAndArcGraphList(FILE *log, GraphList *graph);
//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)
//GraphList *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a lista de adjacência.

void listVertexAndEdgesGraphList(FILE *log, GraphList *graph);
//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)
//GraphList *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a lista de adjacência.

void checkLacesGraphList(FILE *log, GraphList *graph);
//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)
//GraphList *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a lista de adjacência.

void checkMultipleEdgesGraphList(FILE *log, GraphList *graph);
//FILE *log -> ponteiro para um arquivo em que será escrito o nome desta função (log do processamento)

//GraphList *graph -> um ponteiro para a estrutura que armazena os dados do grafo e a lista de adjacência.