UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"

Implementação para Manipulação de Grafos

Gabriel Cecon Carlsen

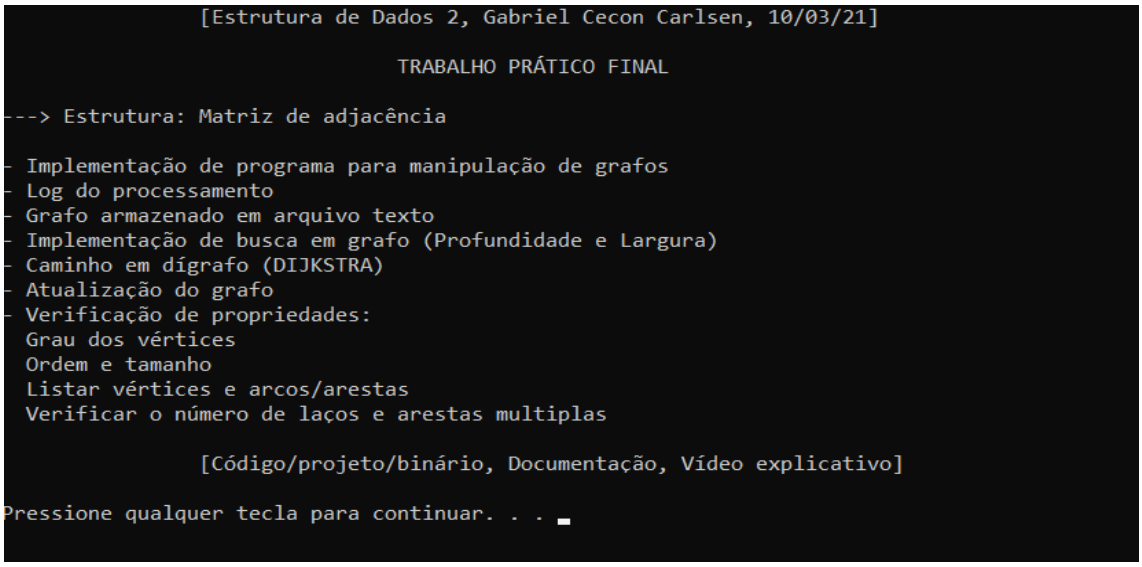
2021
Presidente Prudente

Introdução

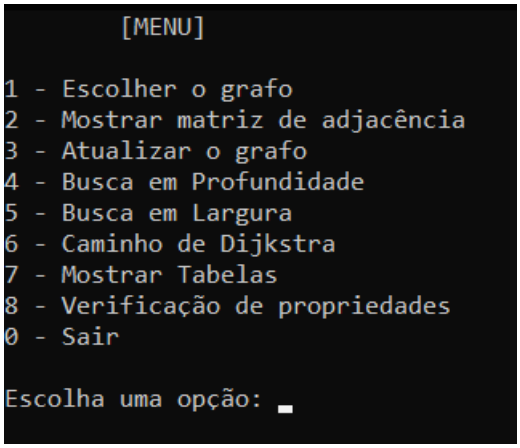
Relatório referente ao trabalho prático final de Estruturas de Dados 2, com o objetivo da implementação um programa (desenvolvido em C) para manipulação de Grafos. O seguinte relatório conterá explicações sobre: funcionamento do código, implementação, estruturas e funções utilizadas.

1. Uso do programa

Ao compilar o código, teremos uma breve apresentação contendo um resumo das funcionalidades presentes no trabalho prático.

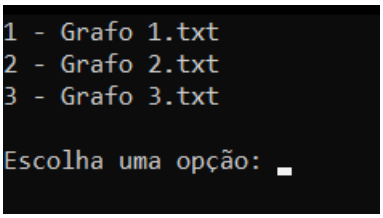


Ao pressionar qualquer tecla, o usuário será redirecionado para o menu principal.

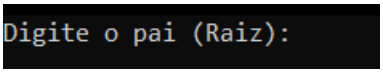


1. Escolher o grafo

O usuário poderá escolher entre três arquivos.txt (1.txt, 2.txt, 3.txt) contendo grafos padronizados no formato solicitado.



Após a escolha é solicitado o vértice pai, ou seja, a raiz para futuras manipulações.



Seguimos de volta ao menu principal, onde o usuário poderá manipular o grafo escolhido.

Obs: Todos os passos executados no programa estão sendo escritos em um arquivo externo **Logger.txt** como requerido, ou seja, ao chamar a opção 4: busca em profundidade, por exemplo, o programa escreverá no arquivo o estado atual usando a função ***fprintf()*** (linha de código: 176).

2. Mostrar matriz de adjacência

Esta opção imprimirá a matriz de adjacência (estrutura usada para representar o grafo), chamando a função ***print_matriz()*** (linha de código: 128 do arquivo ‘main.c’).

3. Atualizar o grafo

Aqui é possível atualizar o grafo, adicionando uma aresta, remover ou atualizar o peso. Será pedido o vértice de saída, vértice de chega e o peso entre os dois, ao fim a matriz antiga será imprimida, assim como, a nova pós alterações.

4. Busca em Profundidade

Está opção executará a busca em profundidade (DFS) do grafo escolhido anteriormente. Segue abaixo os passos:

- Inicializa a estrutura para a busca em profundidade, com a chamada da função ***inicializa()***;
- Chama a função ***DFS()*** para executar a busca em profundidade;
- Em conclusão, chama a função ***gravar_busca_profundidade()*** que gravará o resultado (tabela) da busca no log de processamento **Logger.txt**.

5. Busca em Largura

Está opção executará a busca em largura (BFS) do grafo escolhido anteriormente. Segue abaixo os passos:

- Inicializa a estrutura para a busca em largura, com a chama da função ***inicializa_largura()***;
- Chama a função ***BFS()*** para executar a busca em largura;
- Faz a verificação da existência de algum vértice em branco que não pode ser alcançado pela raiz (localmente sem a chamada de função auxiliar);
- E por fim, chama a função ***grava_busca_largura()*** para gravar a solução da busca no log de processamento **Logger.txt**.

6. Caminho de Dijkstra

Está opção executará o caminho mínimo segundo o algoritmo de Dijkstra do grafo escolhido anteriormente. Segue abaixo os passos:

- Inicializa a estrutura para o caminho de Dijkstra, com chamada da função ***inicializa_dijkstra()***;
- Verifica localmente a existência de elementos negativos, ou seja, não pode continuar com o caminho;

- Caso possa continuar, chama a função ***dijkstra()*** para a executar o caminho mínimo;
- Com a finalização do algoritmo de Dijkstra, o resultado é gravado no log de processamento **Logger.txt** ao Chamar a função ***gravar_dijkstra()***.

7. Mostrar tabelas

A opção ‘7’ faz um print das tabelas das buscas de profundidade e largura, assim como do caminho mínimo de Dijkstra. Cada impressão apresenta sua função especifica, sendo elas:

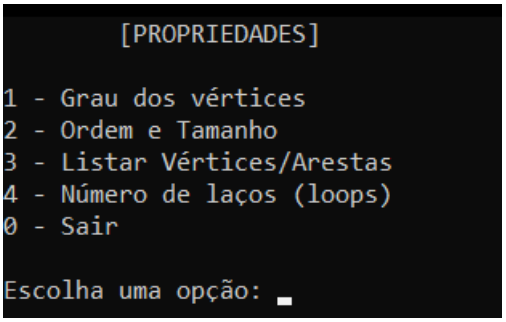
- ***print_tabela()***, busca em profundidade;
- ***print_tabela_largura()***, busca em largura;
- ***print_tabela_dijkstra()***, caminho mínimo de Dijkstra;

A tabela não será exibida, caso nenhuma busca o caminho tenha sido executado.

Vale ressaltar que o programa não possui verificações relativas a erros de execução, caso ocorra, sugiro que reinicie a aplicação ‘main.c’.

8. Verificação de propriedades

Abrirá um submenu contendo opções para a verificação de propriedades relativas ao grafo escolhido.



1. Grau dos vértices

Chama a função ***grauVertices()*** que calculará o grau de incidência dos vértices do grafo escolhido.

2. Ordem e Tamanho

Chama a função ***ordem()*** e ***tamanho()*** para calcular respectivamente a ordem (número de vértices) e o tamanho (número de vértices + arestas) do grafo previamente escolhido.

3. Listar Vértices/Arestas

Esta opção chama a função ***listarVerticesArestas()*** para listar os vértices e arestas do grafo escolhido.

4. Número de laços (loops)

A ultima propriedade implementada chama a função ***numLacos()*** que calcula o número de laços (loops) do grafo selecionado.

0. Sair

Volta para o menu principal.

OBS: Todas as assinaturas e implementações das funções usadas no menu propriedades podem ser encontradas na biblioteca ‘propriedades.h’.

0. Sair

Esta opção encerra a estrutura condicional switch-case, onde todos os arquivos abertos serão fechados pela função ***fclose()***, subsequentemente finalizando o programa principal ‘main.c’.

2. Implementação

2.1. Softwares utilizados

Para a implementação do programa de manipulação de grafos usei o ambiente de desenvolvimento Code:Blocks (versão 16.01) com compilador GNU GCC. Assim como o software visualgo (https://visualgo.net/pt) para melhor visualização dos grafos.

2.2. Status dos processamentos

- ☑ **Matriz de Adjacência**
- ☒ **Lista de Adjacência**

- ☑ **Busca em Profundidade**
- ☑ **Busca em Largura**
- ☑ **Caminho de Dijkstra**

- ☑ **Verificação de propriedades**
- ☑ **Log de processamento**
- ☑ **Atualização do grafo**
- ☑ **Grafo armazenado em texto**

2.3. Matriz de adjacências

Utilizei uma matriz de inteiro com tamanho pré-definido (#define tamanho 100).

```
int matriz_adj[tamanho][tamanho]
```

A biblioteca ‘matriz.h’ auxiliou na implementação e manipulação necessária, com as funções:

```c
//Implementações
void iniciliza_matriz(int matriz[tamanho][tamanho]){//Inicializa todos os elemento da matriz com 0
    for(int i = 0; i < tamanho; i++){
        for(int j = 0; j < tamanho; j++){
            matriz[i][j] = 0;
        }
    }
}

void print_matriz(int matriz[tamanho][tamanho], int vertices){//Print de todos os elementos da matriz
    for(int i = 0; i < vertices; i++){
        for(int j = 0; j < vertices; j++)
            printf(" %3d ",matriz[i][j]);
            printf("\n");
    }
    return;
}

void gravar_grafo(int matriz[tamanho][tamanho], int vertices, int tipo, FILE *f){//Gravando grafo no file
    fprintf(f, "%d\n", tipo);
    fprintf(f, "%d\n", vertices);

    for (int i = 0; i < vertices; i++){
        for (int j = 0; j < vertices; j++){
            if ((matriz[i][j] != 0) && (matriz[i][j] == 1))
                fprintf(f, "%d %d 0\n", i, j);
            else if ((matriz[i][j] != 0) && (matriz[i][j] != 1))
                fprintf(f, "%d %d %d\n", i, j, matriz[i][j]);
        }
    }
    return;
}
```

- ***inicializa_matriz()***: Inicializa todos os elementos da matriz com 0;

- ***Print_matriz()***: Imprime todos os elementos da matriz;

- ***gravar_grafo()***: Grava o grafo no arquivo FILE definido;

2.4 Lista de adjacências

OBS: **Tive problemas para a implementação dessa estrutura de dados relacionando-a as manipulações requeridas, logo não consegui completar esta parte do trabalho :(**

3. Funções

menu.h

```
//Assinaturas
void apresentacao();//Apresentação do trabalho prático
void menu();//Menu de opções
void grafo_selecao();//Seleção do grafo
void menu_propriedades();//Menu de propriedades
```

logger.h

```
//Assinatura
void logger(FILE *f, int grafo);
/*#Parâmetros: Recebe um ponteiro para um FILE (arquivo que será escrito)
  e um int (grafo escolhido).

  #Imprime no log de processamneto a data/hora do inicio da execução
  e o grafo escolhido para a manipulaçao.

*/
```

propriedades.h

```
//Assinaturas
void grauVertices(FILE *logger_file, int matriz[tamanho][tamanho], int vertices);
//#Parâmetros: *FILE para escrita em arquivo, matriz adjacênte, número de vértices
void ordem(FILE *logger_file,int vertices);
//#Parâmetros: *FILE para escrita em arquivo, número de vértices
void tamanhoTotal(FILE *logger_file,int matriz[tamanho][tamanho], int vertices);
//#Parâmetros: *FILE para escrita em arquivo, matriz adjacênte
void numLacos(FILE *logger_file, int matriz[tamanho][tamanho]);
//#Parâmetros: *FILE para escrita em arquivo, matriz adjacênte
void arestasMult(FILE *logger_file, int matriz[tamanho][tamanho]);
//#Parâmetros: *FILE para escrita em arquivo, matriz adjacênte
void listarVerticesArestas(FILE *logger_file, int matriz[tamanho][tamanho], int vertices);
//#Parâmetros: *FILE para escrita em arquivo, matriz adjacênte, número de vértices
```

caminho dijkstra.h

```
//Assinaturas
void gravar_dijkstra(struct dijkstra *x, int raiz, FILE *f, FILE *logger_file);
//Parâmetros: estrutura dijkstra, inteiro raiz, FILE f grafo lido, FILE do log de processamento
void inicializa_dijkstra(struct dijkstra *x, int vertices, int adj[tamanho][tamanho]);
//Parâmetros: estrutura dijkstra, número de vértices, matriz adjacênte
void relaxa(struct dijkstra *x, int vertice_saida, int vertice_chegada, int peso, FILE *logger_file);
//Parâmetros: estrutura dijkstra, vértice de saída, vértice de chegada, peso do vértice, FILE do log de processamento
short int isAllClosed(struct dijkstra *x);
//Parâmetro: estrutura dijkstra
int min(struct dijkstra *x, FILE *logger_file);
//Parâmetros: estrutura dijkstra, FILE do log de processamento
void dijkstra(struct dijkstra *x, int raiz, FILE *logger_file);
//Parâmetros:estrutura dijkstra, inteiro raiz, FILE do log de processamento
void print_tabela_dijkstra(struct dijkstra *x, FILE *logger_file);
//Parâmetros: estrutura dijkstra, FILE do log de processamento
```

busca em profundidade.h

```
//Assinaturas
void inicializa (struct busca_profundidade *x, int adj[tamanho][tamanho], int vertices, int tipo, FILE* logger_file);
//Parâmetros: estrutura busca_profundidade, matriz de adjacênciam número de vértices, tipo de grafo, FILE do log de processamento
void print_tabela(struct busca_profundidade *x, FILE *logger_file);
//Parâmetros: estrutura busca_profundidade, FILE do log de processamento
void DFS(struct busca_profundidade *x, int pai, FILE* logger_file);
//Parâmetros: estrutura busca_profundidade, inteiro raiz, FILE do log de processamento
void DFS_visit(struct busca_profundidade *x, int vertice, FILE* logger_file);
//Parâmetros: estrutura busca_profundidade, número de vértices, FILE do log de processamento
```

busca em largura.h

```
//Assinaturas
void inicializa_largura(struct busca_largura *x, int adj[tamanho][tamanho], int vertices, int tipo);
//Parâmetros: estrutura busca_largura, matriz de adjacência, número de vértices, tipo de grafo
void print_tabela_largura(struct busca_largura *x, FILE *logger_file);
//Parâmetros: estrutura busca_largura, FILE do log de processamento
void gravar_busca_largura(struct busca_largura *x, FILE *f, int raiz, FILE *logger_file);
//Parâmetros: estrutura busca_largura, FILE f grafo lido, inteiro raiz, FILE do log de processamento
```

queue.h

```
//Assinaturas
void inicializa_queue(struct queue* x);
//Parâmetros: estrutura queue
void queue_insert(struct queue* x, int element);
//Parâmetros: estrutura queue, elemento inserido na fila
struct node* queue_remove(struct queue* x);
//Parâmetros: estrutura queue
```

4. Exemplos de execução

BuscaLarguraGrafo.txt

```
Tipo: 1
Vertices: 5
Vertice Raiz: 0

Vertice    Cor    Distancia    Pai
   0        P         0        -1
   1        P         2         2
   2        P         1         0
   3        P         1         0
   4        P         2         3
```

BuscaProfundidadeGrafo.txt

```
Tipo: 1
Vertices: 5
Vertice Raiz: 0

Vertice    Cor    Descoberta    Finaliza
   0        P         1           10
   1        P         3           4
   2        P         2           5
   3        P         6           9
   4        P         7           8
```

Grafo.txt

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

Logger.txt

```
Data: 9/3/2021
Hora: 14:57:25
Grafo escolhido: Grafo 1

Escolhendo a raiz do grafo
Raiz do grafo escolhida com sucesso: 0

Gravando grafo a partir da matriz de Adjacência



Tipo do Grafo: 1
Quantidade de Vertices: 5


CALCULANDO ORDEM
Ordem: 5

CALCULANDO TAMANHO
Tamanho: 11

[Fechando os arquivos]
```
