Exercício de Teoria dos Grafos

Nome: Edson G. M. Jordão- RGM: 32198914
Curso: Ciência da Computação - Turma: 4B
edsongabrielmj@gmail.com

Universidade da Cidade de São Paulo (UNICID) - Rua Cesário Galeno, 448/475 São Paulo – SP – Brasil – CEP: 03071-000

Abstract. *This project aims to answer the list of exercises in the subject of Graph Theory in C programming language, through knowledge obtained during classes, and our creativity and imagination when solving problems. With the help of complementary material provided by Professor Juliano Ratusznei.*

Resumo. Este trabalho tem por objetivo responder à lista de exercícios da matéria de Teoria dos Grafos na linguagem de programação C através de conhecimentos obtidos durante as aulas, e nossa criatividade e imaginação na hora de resolver problemas. Com o auxílio do material complementar disponibilizado pelo professor Juliano Ratusznei.

São Paulo - 2024

Descrição da Problemática:

Implementar um algoritmo para percorrer um grafo:

1. Alocação de memória dinâmica para os vértices inseridos;
2. Uma função de busca em largura;
3. Representação do grafo;
4. Possibilitar a inserção dos vértices de origem e destino;
5. Imprimir quaisquer arestas em ordem utilizando os vértices de origem e destino como base.

1. **Descrição da Função:**

Implementação de duas estruturas de dados, para armazenar as informações fornecidas pelo usuário.

Codificação:

```
struct Aresta {
    int origem;
    int destino;
};

struct Aresta* aresta;
int V;
int numArestas;
```

Descrição da aprendizagem obtida através da problemática:

Com esse exercício, aprendi a forma correta de se declarar uma estrutura de dados onde tem-se uma origem e um destino. Além de esclarecer a maneira de declarar um ponteiro sendo uma variável global.

2. **Descrição da Função:**

Criar uma função para a impressão da lista de arestas.

Codificação:

```
void imprimirListaAresta() {
    printf("\nLista de arestas:\n");
    int i;
    for (i = 0; i < numArestas; i++) {
        printf("%d  <==>  %d\n", aresta[i].origem, aresta[i].destino);
    }
}
```

Descrição da aprendizagem obtida através da problemática:

Com esse exercício, aprendi a forma com que se dá a impressão de uma lista de Vértices.

3. **Descrição da Função:**

Implementação de uma função para inicializar uma lista de arestas.

Codificação:

```
void inicializarLista() {
    printf("Para sair (-99):\n");

    int contArestas = 0;
    int origem, destino;

    while (1) {
        if (contArestas == V * (V - 1) / 2) {
            printf("Limite máximo de arestas atingido.\n");
            break;
        }

        printf("\nInsira a origem da aresta %d: ", contArestas + 1);
        scanf("%d", &origem);

        if (origem == -99) {
            break;
        }

        printf("Insira o destino da aresta %d: ", contArestas + 1);
        scanf("%d", &destino);

        if (destino == -99) {
            destino = origem;
        }

        aresta[contArestas].origem = origem;
        aresta[contArestas].destino = destino;
        contArestas++;
    }
    numArestas = contArestas;
}
```

Descrição da aprendizagem obtida através da problemática:

Com esse exercício, aprendi a forma com que se insere um elemento no início da lista de arestas.

4. **Descrição da Função:**

Implementação da primeira parte de uma função fazer a busca em largura em uma lista de arestas.

Codificação:

```
void buscaEmLargura(int origem, int destino) {
    int visitado[V], i;
    for (i = 0; i < V; i++) {
        visitado[i] = 0;
    }

    int fila[V], frente = 0, traseira = 0;
    fila[traseira++] = origem;
    visitado[origem - 1] = 1;

    int pai[V];
    for (i = 0; i < V; i++) {
        pai[i] = -1;
    }
```

Descrição da aprendizagem obtida através da problemática:

Com essa parte do exercício, aprendi a forma com que se faz um for de forma eficaz.

4.1 Descrição da Função:

Implementação da segunda parte de uma função fazer a busca em largura em uma lista de arestas.

Codificação:

```
while (frente != traseira) {
        int u = fila[frente++];
        for (int i = 0; i < numArestas; i++) {
            if (aresta[i].origem == u || aresta[i].destino == u) {
                int v = (aresta[i].origem == u) ?
                                        aresta[i].destino : aresta[i].origem;
                if (!visitado[v - 1]) {
                    fila[traseira++] = v;
                    visitado[v - 1] = 1;
                    pai[v - 1] = u;
                    if (v == destino) {
                        int caminho[V], idx = 0;
                        caminho[idx++] = destino;
                        int p = pai[v - 1];
                        while (p != -1) {
                            caminho[idx++] = p;
                            p = pai[p - 1];
                        }
                        printf("Caminho de %d a %d: ", origem, destino);
                        for (int j = idx - 1; j >= 0; j--) {
                            printf("%d ", caminho[j]);
                        }
                        printf("\n");
                        return;
                    }
                }
            }
        }
    }

    printf("Não há caminho de %d a %d.\n", origem, destino);
}
```

Descrição da aprendizagem obtida através da problemática:

Com esse exercício, aprendi a forma com que se armazena uma estrutura de dados em *'aresta'* e se busca um caminho entre vértices de origem e destino utilizando uma fila para armazenar os vértices a serem visitados, onde o pai de cada vértice é atualizado para rastrear o caminho.

5. **Descrição da Função:**

Implementação de uma função principal para imprimir e solicitar todas as informações necessárias.

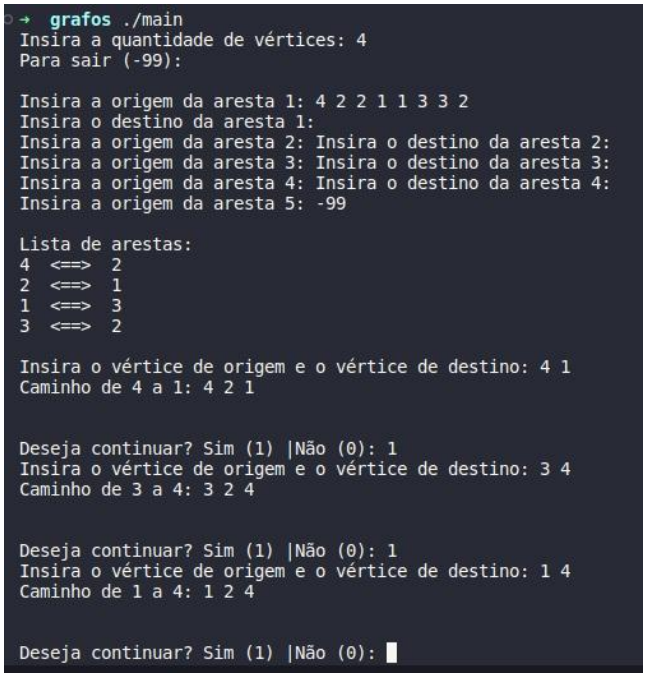
Codificação:

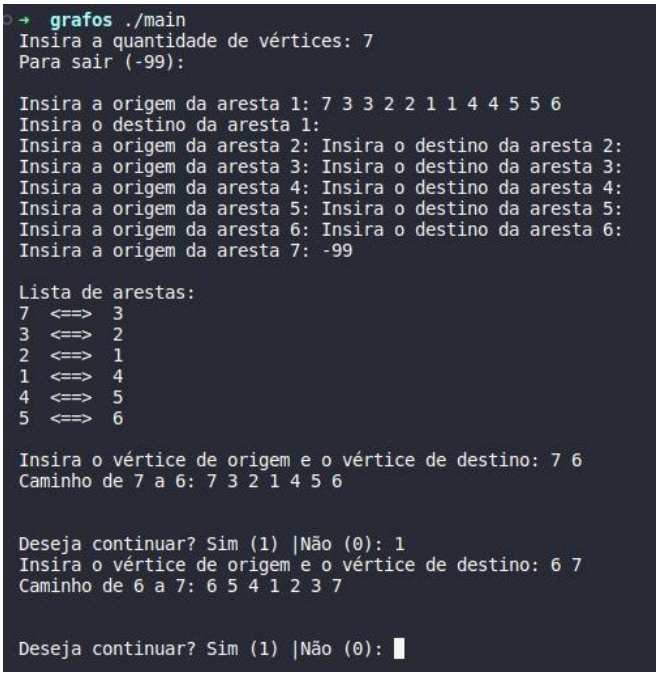
```c
int main(void) {
    setlocale(LC_ALL,"Portuguese");

    int origem, destino;
    int continuar = 1;

    printf("Insira a quantidade de vértices: ");
    scanf("%d", &V);

    // Calcula o número máximo de arestas permitido (n(n - 1)/2)
    int maxArestas = V * (V - 1) / 2;

    // Aloca espaço para o número máximo de arestas
    aresta = (struct Aresta*)malloc(maxArestas * sizeof(struct Aresta));

    // Inicializa a lista de arestas
    inicializarLista();
    imprimirListaAresta();
    printf("\n");

    do{
        printf("Insira o vértice de origem e o vértice de destino: ");
        scanf("%d %d", &origem, &destino);

        // Realiza a busca em largura
        buscaEmLargura(origem, destino);

        printf("\n\n");
        printf("Deseja continuar? Sim (1) |Não (0): ");
        scanf("%d",&continuar);

    }while(continuar == 1);

    free(aresta); // Libera a memória alocada para o array de arestas

    return 0;
}
```

Descrição da aprendizagem obtida através da problemática:

Com esse exercício, implementei uma função principal para organizar todas as funções e coletar todos os dados necessários.

6. Resultados no terminal:





Referências:

SOUZA, EDUARDO; Ciência da Computação na prática — Teoria dos Grafos: Busca em Largura. EDUARDO SOUZA, disponível em: https://inside.contabilizei.com.br/ci%C3%AAncia-da-computa%C3%A7%C3%A3o-na-pr%C3%A1tica-teoria-dos-grafos-busca-em-largura-15a1f1de1d1a. Acessado em abr/2024.

GABRIEL, EDSON; Segundo Semestre: Estrutura de Dados I. Algoritmos: Listas, disponível em: https://github.com/EGMJ/Faculdade/blob/master/2_SEM/Estrutura%20de%20dados%20I/Aulas/lista.c. Acessado em abr/2024.

FEOFILOFF, PAULO; Grafos: Busca em largura. IME USP, disponível em: https://www.ime.usp.br/~pf/algoritmos_em_grafos/aulas/bfs.html. Acessado em abr/2024.

FEOFILOFF, PAULO; Algoritmo de Busca em largura. IME USP, disponível em: https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/bfs.html. Acessado em abr/2024.