Implementação de Conceitos em Teoria de Grafos.

Bruno Henrique da Silva Brum[1], **Lucas Zegrine Duarte**[1]

[1]ICEI – Pontifícia Universidade Católica de Minas Gerais (PUCMG)
R. Dom José Gaspar, 500 - Coração Eucarístico, Belo Horizonte - MG, 30535-901

lucas.zegrine@gmail.com

lucas.duarte.1327493@sga.pucminas.br

Abstract. *This work addresses the implementation of graph theory concepts, both in the form of undirected and directed graphs, using data structures such as adjacency matrices and adjacency lists, respectively. It discusses the breadth-first search (BFS) and depth-first search (DFS) methods for traversing and classifying the edges of graphs. It also covers methods for determining properties of vertices and edges in a graph, such as the eccentricity of a vertex and types of edges (tree, back, forward, and cross edges).*

Resumo. *Este trabalho aborda a implementação de conceitos de teoria de grafos, tanto na forma de grafos não-direcionados quanto direcionados, utilizando estruturas de dados como matriz de adjacência e lista de adjacência, respectivamente. São discutidos os métodos de busca em largura (BFS) e profundidade (DFS) para percorrer e classificar as arestas dos grafos. Também são abordados métodos para determinar propriedades dos vértices e arestas de um grafo, como a excentricidade de um vértice e tipos de arestas (árvore, retorno, avanço e cruzamento).*

1. Introdução.

Serão apresentados conceitos fundamentais da teoria de grafos em que é são discutidas possíveis implementações de cada conceito assim como pequenas explicações de como estes foram implementados. Juntamente a este trabalho foram realizadas as implementações em C++ das respectivas secções.

2. Implementação Grafo Não-Direcionado

Para representar um grafo não-direcionado, foi utilizada uma matriz de adjacência em que cada posição (i, j) indica se existe ou não uma aresta conectando os dois vértices i e j. Por ser um grafo não-direcionado, as células (i, j) e (j, i), possuem o mesmo valor, já que um grafo não direcionado não possui vetor de direção entre os vértices.

3. Implementação Grafo Direcionado

Na implementação de um grafo direcionado, cada aresta (u, v), possui um direcionamento, onde u é o vértice de origem da aresta, e v é o vértice de destino. Portando o vector de direcionamento de dada aresta vai de u para v. Para representar este tipo de grafo, foi utilizado uma lista de adjacência, na qual se cria um vetor de vetores de inteiros (vetor[i][j]) em que i corresponde aos vértices e j aos vértices vizinhos a i.

4. Implementação Busca em Largura (BFS)

A busca em largura é utilizada para percorrer grafos por níveis, ou seja, são visitados todos os vértices vizinhos do vértice atual antes de avançar para os vértices mais distantes. Para isso criamos uma fila contendo os vértices a serem visitados, e um vetor contendo os vértices já visitados.

4.1. Percorrendo o Grafo

O algoritmo itera por todos os vértices da fila e em cada iteração analisamos os vizinhos do primeiro vértice i da fila e quando todos os seus vizinhos foram visitados, i é removido da fila. Cada vizinho visitado é adicionado ao vetor de vértices visitados.

5. Implementação Busca em Profundidade (DFS)

5.1. Coloração

Na implementação do método de busca em profundidade (DFS), deve ser criado um vetor de inteiros com objetivo de armazenar as cores de cada vértice do grafo. Todos os vértices serão inicializados com cor branca, significando que o vértice nunca foi visitado anteriormente. Quando o vértice é descoberto pela primeira vez, este será marcado com aresta de cor azul e caso todos os vizinhos de um vértice já tenham sido visitados, então o vértice receberá o vermelho.

5.2. Tempos de Descoberta e Término

Durante a execução do DFS, cada vértice u recebe um tempo de descoberta, registrado no momento em que u é visitado pela primeira vez. Após a exploração completa de todos os vértices adjacentes a u, o tempo de término é registrado.

5.3. Percorrendo o Grafo

Para caminhar pelo grafo utilizando a lógica DFS, é necessário chamar recursivamente uma função auxiliar iterando por todos os vértices do grafo. Para que isso seja feito é necessário armazenar o índice do próximo vértice vizinho do vértice atual, e caso o vértice vizinho seja de cor branca, a função deve ser chamada recursivamente utilizado o índice do vértice vizinho como parâmetro, caso contrário, se não houver nenhum vértice vizinho a ser descoberto, a cor do vértice atual deve ser vermelha.

6. Implementação do Método de Classificação das Arestas

Para ser possível categorizar as arestas quando às suas características, sendo, ”aresta de árvore”, ”aresta de retorno”, ”aresta de avanço”, ”aresta de cruzamento”. Foi preciso implementar um registro armazenando informações da aresta, como o vértice de origem u, o vértice de destino v e o tipo de aresta. Para classificar a aresta é preciso que a função de Busca em Profundidade atualize os tempos de descoberta e termino e a coloração da aresta a medida que a busca seja realizada.

6.1. Aresta de Árvore

Dada um aresta(u, v), esta será uma aresta de árvore se o vértice v for branco, pois a aresta ainda não foi visitada.

6.2. Aresta de Retorno

Dada uma aresta(u, v), esta será uma aresta de retorno se o vértice v for azul, ou seja, a aresta v já foi descoberta anteriormente e o vértice u está retornando ao vértice v.

6.3. Aresta de Avanço

Para arestas de avanço, o vértice u, tem de ser descoberto antes do vértice v, assim como o vértice v tem de possuir todos os seus vértices vizinhos descobertos. Para isso checamos se o tempo de descoberta de ambos os vértices e o tempo do vértice u deve ser menos que o tempo do vértice v e o vértice v deve ter cor vermelha.

6.4. Aresta de Cruzamento

Para arestas de cruzamento, o vértice u, deve ter sido descoberto depois do descobrimento do vértice v, assim como o vértice v dever possuir todos os seus vértices vizinhos descobertos.

6.5. Algoritmo

A seguir, é apresentado um pseudo-código para a classificação das arestas em um grafo utilizando o algoritmo de Busca em Profundidade (DFS). Este algoritmo categoriza as arestas em quatro tipos: árvore, retorno, avanço e cruzamento, com base na cor dos vértices e nos tempos de descoberta e término registrados durante a execução do DFS.

Algorithm 1: Classificação de Arestas em DFS

```
Data: Grafo G = (V, E)
Result: Classificação das arestas em G
foreach vértice u ∈ V do
    cor[u] ← branco;
    descoberta[u] ← NULL;
    término[u] ← NULL;
tempo ← 0;
foreach vértice u ∈ V do
    if cor[u] = branco then
        DFS_Visita(u);
Função DFS_Visita(u):
    tempo ← tempo + 1;
    descoberta[u] ← tempo;
    cor[u] ← azul;
    foreach vértice v adjacente a u do
        if cor[v] = branco then
            classificar(u, v) como aresta de árvore;
            DFS_Visita(v);
        else if cor[v] = azul then
            classificar(u, v) como aresta de retorno;
        else if cor[v] = vermelho then
            if descoberta[u] ¡ descoberta[v] then
                classificar(u, v) como aresta de avanço;
            else
                classificar(u, v) como aresta de
                 cruzamento;
    cor[u] ← vermelho;
    tempo ← tempo + 1;
    término[u] ← tempo;
```

7. Implementação do Método de Encontrar Excentricidade de Vértices

Para encontrar a excentricidade de um vértice v em um grafo é necessário determinar o maior das menores distâncias entre o vértice v e todos os outros vértices do grafo. Ou seja, a excentricidade de u é o valor máximo entre as distâncias mínimas de u para qualquer outro vértice v.

7.1. Distância

Dado uma aresta (u, v), para calcularmos a distância entre os vértices u e v, precisamos que ao realizarmos a busca em largura, seja criado um vetor que armazene os valores de distância de cada vértice em relação com o vértice inicial. Portanto, o vértice inicial u terá distância 0 em relação a si mesmo e o vértice adjacente v conectado ao vértice inicial u terá distância 1 em relação ao vértice inicial u. Os vértices subsequentes conectados à v terão a distância em relação u igual à distância do vértice anterior (v) + 1.

8. Menção

Os algoritmos descritos acima foram retiradas dos slides disponibilizados durante aula do professor Silvio Jamil Guimarães, na disciplina de Projeto e Análise de Algoritmos da PUC Minas de Minas Gerais.