Implementação 1

Lucas M. O. Xavier[1],
Nilson D. C. Filho[1], **Pedro M. Rodrigues**[1]

[1]Instituto de Ciências Exatas e Informática – Pontifícia Universidade Católica
de Minas Gerais (PUC Minas)
Caixa Postal 30.535-000 – Belo Horizonte – MG – Brasil

{998707, 1369636, 1373336}@sga.pucminas.br

Resumo. *Este trabalho aborda a implementação de duas representações fundamentais de grafos em C++, com foco na aplicação de algoritmos de busca em grafos direcionados e não-direcionados. O estudo inclui a implementação de um grafo direcionado utilizando lista de adjacência e um grafo não-direcionado utilizando matriz de adjacência. Os algoritmos aplicados, como a busca em profundidade (DFS) e a busca em largura (BFS), são utilizados para classificar arestas e calcular excentricidades, respectivamente. Os resultados demonstram a eficiência das abordagens em termos de análise e manipulação de grafos.*

1. Introdução

Grafos são estruturas matemáticas amplamente utilizadas para modelar relações entre objetos. Essas estruturas são aplicáveis em diversas áreas, como redes de comunicação, análise de tráfego, biologia computacional e muitos outros campos que necessitam da modelagem de relações e interações entre entidades. Neste trabalho, exploramos a implementação de grafos em C++, focando em dois tipos distintos: grafos direcionados e não-direcionados. Além disso, aplicamos algoritmos clássicos de busca para explorar as propriedades estruturais dos grafos.

2. Metodologia

Para o desenvolvimento desta tarefa, foram implementados dois tipos de grafos: um grafo direcionado utilizando lista de adjacência e um grafo não-direcionado utilizando matriz de adjacência. A metodologia consistiu na criação dos grafos, seguida da implementação de algoritmos de busca para exploração das propriedades específicas de cada estrutura.

2.1. Grafo Direcionado usando Lista de Adjacência

A lista de adjacência foi escolhida para representar o grafo direcionado devido à sua eficiência no armazenamento de grafos esparsos. Nesta representação, cada vértice do grafo possui uma lista associada que armazena seus vértices adjacentes, permitindo que as arestas sejam armazenadas de maneira compacta. Este método é particularmente vantajoso em situações onde o número de arestas é muito menor que o número de vértices ao quadrado, o que economiza espaço em comparação à matriz de adjacência.

A implementação do grafo direcionado também incluiu a aplicação de um algoritmo de busca em profundidade (DFS) para percorrer o grafo e classificar as arestas em quatro categorias distintas:

- Arestas de Árvore: Arestas que fazem parte da árvore de DFS.
- Arestas de Retorno: Arestas que apontam para um antecessor no caminho da DFS, indicando a presença de ciclos.
- Arestas de Avanço: Arestas que ligam um vértice a um descendente no caminho da DFS, mas que não fazem parte da árvore DFS.
- Arestas de Cruzamento: Arestas que conectam vértices de diferentes subárvores na árvore de DFS, mas que não são arestas de retorno.

2.2. Grafo Não-Direcionado usando Matriz de Adjacência

Para o grafo não-direcionado, a matriz de adjacência foi a representação escolhida. Neste caso, a matriz é uma estrutura bidimensional onde cada célula indica a presença ou ausência de uma aresta entre dois vértices. Essa representação é particularmente útil para grafos densos, onde a maioria dos pares de vértices está conectada por uma aresta.

A matriz de adjacência permite acesso imediato à informação de conectividade entre os vértices, o que simplifica a implementação de algoritmos que necessitam de múltiplos acessos rápidos a essa informação. Nesta implementação, utilizamos a busca em largura (BFS) para calcular a excentricidade de um vértice. A excentricidade é definida como a maior distância entre o vértice dado e qualquer outro vértice no grafo, sendo uma medida importante na análise da centralidade e da estrutura do grafo.

3. Implementação

3.1. Grafo Direcionado usando Lista de Adjacência

O grafo direcionado foi implementado utilizando uma lista de adjacência. Cada vértice no grafo é associado a uma lista que armazena todos os vértices adjacentes, ou seja, aqueles que são diretamente alcançáveis por uma aresta a partir do vértice em questão.

O algoritmo de DFS foi implementado para percorrer o grafo, marcando cada vértice à medida que é visitado. Durante essa travessia, as arestas são classificadas conforme são exploradas. O algoritmo foi projetado para lidar tanto com grafos criados manualmente pelo usuário quanto com grafos gerados aleatoriamente, garantindo flexibilidade na análise de diferentes topologias de grafos.

3.2. Grafo Não-Direcionado usando Matriz de Adjacência

Para o grafo não-direcionado, foi utilizada uma matriz de adjacência, onde cada célula [i][j] na matriz indica se há uma aresta entre os vértices i e j. A adição de arestas é feita de forma simétrica, garantindo que o grafo seja não-direcionado.

A BFS foi implementada para calcular a excentricidade de um vértice, que é a maior distância mínima entre o vértice em questão e qualquer outro vértice no grafo. O algoritmo utiliza uma fila para garantir que cada vértice seja processado na ordem correta, permitindo a determinação eficiente das distâncias mínimas.

4. Resultados

Os resultados das implementações foram satisfatórios e cumpriram os objetivos propostos. No caso do grafo direcionado, a classificação das arestas em categorias específicas forneceu insights valiosos sobre a estrutura do grafo e suas propriedades.

A execução do algoritmo DFS permitiu identificar corretamente ciclos, conexões hierárquicas e a interdependência entre os vértices. Ao executar o programa com a opção de gerar um grafo aleatório, foram obtidos os seguintes resultados para a classificação das arestas.

Já no grafo não-direcionado, a aplicação da BFS para calcular a excentricidade de um vértice demonstrou a eficácia da matriz de adjacência para operações de busca em grafos densos. A excentricidade calculada para cada vértice permitiu identificar quais vértices estavam mais centralizados ou periféricos em relação à estrutura global do grafo.

5. Conclusão

A implementação dos dois tipos de grafos em C++ mostrou-se adequada para os propósitos de análise e exploração das características estruturais dos grafos. A lista de adjacência provou ser uma representação eficiente para grafos direcionados, particularmente em termos de espaço, enquanto a matriz de adjacência demonstrou sua utilidade em grafos não-direcionados, especialmente quando é necessário acesso rápido às informações de conectividade.

No futuro, a implementação pode ser expandida para incluir outros tipos de grafos e algoritmos, como grafos ponderados e algoritmos de caminho mínimo. Além disso, a otimização dos algoritmos para lidar com grafos de grande escala pode ser explorada, aumentando a aplicabilidade das soluções desenvolvidas para cenários reais e complexos.