PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Instituto de Ciências Exatas e de Informática

Relatório Implementação N.04 - Fluxo Máximo

Resumo

Este relatório descreve a implementação de um método para determinar todos os caminhos disjuntos em arestas em um grafo direcionado. O método implementado recebe um arquivo contendo a descrição do grafo direcionado e um par de vértices (origem e destino) e exibe a quantidade de caminhos disjuntos em arestas entre os vértices, listando cada caminho encontrado.

1 INTRODUÇÃO

Este relatório descreve a implementação de um método para determinar todos os caminhos disjuntos em arestas em um grafo direcionado. O método implementado recebe um arquivo contendo a descrição do grafo direcionado e um par de vértices (origem e destino) e exibe a quantidade de caminhos disjuntos em arestas entre os vértices, listando cada caminho encontrado.

2 DESENVOLVIMENTO

2.1 Representação do Grafo

```
Map<Integer, Map<Integer, Integer>> grafo = new HashMap<>();
```

O grafo direcionado foi representado por meio de uma estrutura de dados baseada em um mapa, em que cada chave corresponde a um vértice do grafo e o valor associado a ele é uma lista de pares representando as arestas de saída e a capacidade da aresta, que, inicialmente, foi definida como 1 para todas as conexões.

2.2 Algoritmo Implementado

O algoritmo de Ford-Fulkerson foi implementado para encontrar o fluxo máximo no grafo. No caso de grafos com arestas com capacidades iguais a um, o fluxo máximo corresponde ao número máximo de caminhos disjuntos em arestas e cada caminho aumentante encontrado corresponde a um caminho disjunto.

```
public List<List<Integer>> FF(Grafo g) {
        List<List<Integer>> caminhos = new ArrayList<>();
        Grafo redeResidual = criarRedeResidual();
        List<Integer> caminhoAumentante;
        while ((caminhoAumentante = encontrarCaminho(redeResidual))
            != null) {
            caminhos.add(caminhoAumentante);
            redeResidual = atualizarResidual(redeResidual,
                caminhoAumentante);
        }
        return caminhos;
    }
```

Criação da Rede Residual: Para cada grafo de entrada, uma rede residual é criada.

Busca de Caminho Aumentante: Foi utilizada a busca em largura (BFS) para encontrar caminhos aumentantes na rede residual.

Atualização do Grafo Residual: Após encontrar um caminho aumentante, o grafo residual é atualizado, decrementando a capacidade das arestas no caminho direto e incrementando a capacidade das arestas reversas.

2.2.1 Grafo Bipartido

Um grafo bipartido é aquele em que os vértices podem ser divididos em dois conjuntos disjuntos, de modo que as arestas sempre conectam um vértice de um conjunto a outro, nunca dentro do mesmo conjunto. Este tipo de grafo foi escolhido para realizar testes de eficiência do algoritmo para analisar como ele responde a grafos esparsos, com poucas conexões. Realizando testes com quatro tamanhos diferentes de grafos bipartidos, considerando S = vértice 1 e T = maior vértice do grafo, os resultados foram estes:

Vértices	Tempo de execução	Caminhos Encontrados
10	0.8389 ms	1
20	1.0444 ms	0
40	1.2642 ms	0
60	2.1129 ms	1

Tabela 1 – Resultados dos testes com grafos bipartidos

2.2.2 Grafo Denso

Um grafo denso é caracterizado por possuir um número de arestas próximo ao máximo possível para a quantidade de vértices, ou seja, grande parte dos vértices está conectada entre si. Este tipo de grafo foi escolhido para testar a eficiência do algoritmo, pois sua alta conectividade gera muitos caminhos possíveis, exigindo mais operações do algoritmo tanto na busca por caminhos quanto nas atualizações das capacidades residuais. Isso permite avaliar como o algoritmo se comporta em cenários onde há muitas alternativas de fluxo e maior complexidade computacional. Foram realizados testes com quatro tamanhos diferentes de grafos densos, considerando S = vértice 1 e T = o maior vértice do grafo, e os resultados foram os seguintes:

Vértices	Arestas	Tempo de execução	Caminhos Encontrados
10	50	1.1063 ms	1
15	100	1.6614 ms	1
23	250	3.1252 ms	22
25	200	4.9894 ms	15

Tabela 2 – Resultados dos testes com grafos densos

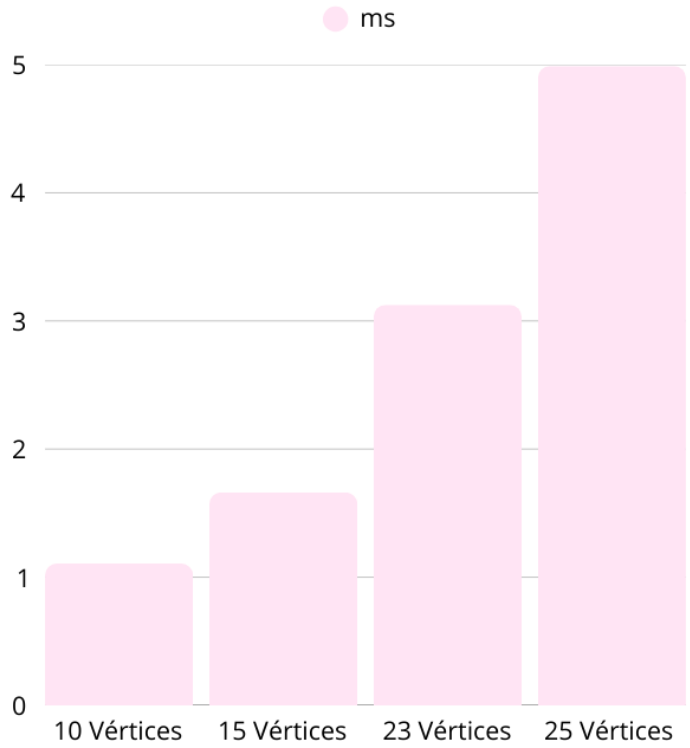

Figura 1 – Gráfico de desempenho para grafos densos

3 CONCLUSÃO

A partir dos testes realizados, foi possível analisar o comportamento do algoritmo de busca de caminhos disjuntos em arestas em diferentes tipos de grafos. Os resultados mostraram que o tipo de grafo influencia diretamente tanto no desempenho quanto na quantidade de caminhos encontrados. Nos testes com grafos bipartidos, observou-se que, na maioria dos casos, o algoritmo não encontrou caminhos disjuntos, especialmente à medida que o número de vértices aumentava. Isso ocorre devido à estrutura naturalmente esparsa desse tipo de grafo, que oferece poucas possibilidades de conexões entre os vértices dos dois conjuntos. Por exemplo, nos grafos com 20 e 40 vértices, nenhum caminho foi encontrado, o que evidencia a limitação estrutural para esse tipo de problema.

Em contrapartida, os testes com grafos densos apresentaram resultados bem diferentes. À medida que a densidade do grafo aumentou, o número de caminhos disjuntos encontrados também cresceu consideravelmente, assim como o tempo de execução. No grafo com 23 vértices e 250 arestas, o algoritmo encontrou 22 caminhos, e no grafo com 25 vértices e 200 arestas, foram 15 caminhos, demonstrando que a alta conectividade favorece múltiplos caminhos possíveis entre a origem e o destino. Apesar do aumento no tempo de execução, que chegou a 4.9894 ms no maior grafo denso, o algoritmo manteve um desempenho aceitável e escalável.

Diante disso, conclui-se que grafos densos são cenários ideais para avaliar tanto a eficiência

quanto a robustez do algoritmo, já que exigem mais processamento devido à quantidade de caminhos possíveis e às atualizações sucessivas das capacidades residuais. Por outro lado, os grafos bipartidos, por sua baixa conectividade, são úteis para testar o comportamento do algoritmo em situações onde os caminhos possíveis são limitados ou inexistentes.