Análise de Caminhos Disjuntos em Arestas em Grafos Direcionados

Henrique Oliveira da Cunha Franco

1 Introdução

Este documento apresenta uma análise experimental de uma implementação que calcula caminhos disjuntos em arestas em grafos direcionados utilizando o método de fluxo máximo. O objetivo é avaliar a eficiência da implementação, medida pelo tempo de execução para instâncias de diferentes tamanhos de grafos.

2 Descrição da Implementação

O código foi desenvolvido em Java e consiste de duas classes principais: `MainGraph` e `MaxFlowPaths`. A classe `MainGraph` é responsável pela leitura do grafo a partir de um arquivo e pelo preenchimento de sua estrutura de dados em listas de adjacência. A classe `MaxFlowPaths` implementa o cálculo dos caminhos disjuntos em arestas entre dois vértices (definidos como origem e destino), utilizando o algoritmo de fluxo máximo baseado no método de Edmonds-Karp com um grafo residual. Os grafos em que esse método testado possuem duas categorias: esparsos e densos.

A função `Implementacao3` na classe principal coordena a execução do cálculo, medindo o tempo de execução para a análise de eficiência.

3 Metodologia

Os experimentos foram realizados com dois tipos de grafos direcionados: grafos esparsos e grafos densos. Para cada tipo, foram geradas instâncias aleatórias de diferentes tamanhos, variando o número de vértices e arestas. Para cada instância, o tempo de execução foi registrado, fornecendo uma métrica de eficiência. Por fim, o vértice sumidouro escolhido foi sempre 1 e o terminal escolhido foi o vértice de maior valor do grafo, de modo a garantir que o pior caso sempre seja executado. A seguir está uma explicação breve explicão que caracteriza os tipos de grafos aplicados na implementação.

3.1 Grafos Esparsos

Os grafos esparsos são caracterizados por terem um número relativamente baixo de arestas em comparação ao número de vértices. Nesse estudo, a geração dos grafos esparsos foi feita de forma que, para cada instância, o número de arestas fosse proporcionalmente menor que o número de vértices. Por exemplo, o primeiro grafo gerado possui 100 vértices e 100 arestas, enquanto o quarto possui 1000 vértices e 1000 arestas. O aumento linear no número de arestas pode levar a um aumento proporcional no tempo de execução, mas a complexidade do algoritmo se mantém controlada devido à menor densidade de arestas.

3.2 Grafos Densos

Em contraste, os grafos densos apresentam um número elevado de arestas em relação ao número de vértices. No estudo, os grafos densos foram gerados com uma alta densidade de arestas, de modo que o número de arestas se aproxima do limite máximo, que é dado por $V(V-1)$, onde V é o número de vértices. Por exemplo, o segundo grafo denso possui 1000 vértices e 50.000 arestas. Esse aumento no número de arestas leva a um crescimento exponencial na complexidade do algoritmo, resultando em tempos de execução significativamente maiores, como observado nos resultados.

4 Gerações de Grafos Aleatórios

Os grafos aleatórios foram gerados utilizando dois métodos: `sparseGraphGenerator` e `denseGraphGenerator`. O código de geração de grafos esparsos e densos utiliza valores base para o número de vértices e arestas, que são posteriormente elevados para criar instâncias de maior complexidade. Vale notar também que todos os grafos gerados são conexos, de modo que o vértice sumidouro sempre consiga encontrar o vértice terminal.

Para os grafos esparsos, o número de arestas é multiplicado por fatores que aumentam significativamente a densidade de arestas, conforme exemplificado:

- Grafo Esparso 1: 100 vértices, 100 arestas.
- Grafo Esparso 2: 200 vértices, 200 arestas (multiplicação por 2).
- Grafo Esparso 3: 500 vértices, 500 arestas (multiplicação por 5).
- Grafo Esparso 4: 1000 vértices, 1000 arestas (multiplicação por 10).

Para os grafos densos, a abordagem é semelhante, mas com um aumento ainda mais significativo no número de arestas:

- Grafo Denso 1: 100 vértices, 500 arestas.
- Grafo Denso 2: 1000 vértices, 50.000 arestas (multiplicação por 10).
- Grafo Denso 3: 10.000 vértices, 500.000 arestas (multiplicação por 100).
- Grafo Denso 4: 100.000 vértices, 5.000.000 arestas (multiplicação por 1000).

Essa estratégia de elevação dos valores permite testar o comportamento do algoritmo em diferentes cenários, avaliando sua escalabilidade e eficiência.

5 Resultados

Os resultados estão apresentados nas Tabelas 1 e 2, que mostram o tempo de execução em milissegundos para cada instância de grafo esparso e denso, respectivamente.

Table 1: Tempo de Execução para Grafos Esparsos

Instância	Número de Vértices	Número de Arestas	Tempo de Execução (ms)
Grafo Esparso 1	100	100	1
Grafo Esparso 2	200	200	1
Grafo Esparso 3	500	500	2
Grafo Esparso 4	1000	1000	3

Table 2: Tempo de Execução para Grafos Densos

Instância	Número de Vértices	Número de Arestas	Tempo de Execução (ms)
Grafo Denso 1	100	500	1
Grafo Denso 2	1000	50.000	805
Grafo Denso 3	10.000	500.000	1.629
Grafo Denso 4	100.000	5.000.000	27.410

6 Conclusão

Observa-se que o tempo de execução aumenta com o número de vértices em ambas as categorias de grafos. Grafos densos apresentaram um crescimento mais acentuado no tempo de execução, como esperado devido ao maior número de arestas e, consequentemente, à complexidade computacional aumentada no cálculo de caminhos disjuntos em arestas. A análise dos dados sugere que a implementação se comporta de forma eficiente para grafos esparsos (pela complexidade menor dos dados), mas a performance se degrada rapidamente em grafos densos, onde as interações entre as arestas tornam o processamento mais custoso, naturalmente.