Trabalho Prático 02

Camila Lacerda Grandini
camila.grandini@sga.pucminas.br
Dept. of Computer Science, Pontifícia Universidade Católica de Minas Gerais (PUC Minas), Brazil

Joana Moreira Woldaynsky
jmrwoldaynsky@sga.pucminas.br
Dept. of Computer Science, Pontifícia Universidade Católica de Minas Gerais (PUC Minas), Brazil

José Fernando Rossi
jfrjunior@sga.pucminas.br
Dept. of Computer Science, Pontifícia Universidade Católica de Minas Gerais (PUC Minas), Brazil

ACM Reference Format:
Camila Lacerda Grandini, Joana Moreira Woldaynsky, and José Fernando Rossi. 2022. Trabalho Prático 02. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUÇÃO

No estudo das Teorias dos Grafos os algoritmos baseados em grafos são usados em diversas áreas para auxiliar nas resoluções de inúmeros problemas. Um exemplo são grafos com caminho simples, determinados por um grafo direcionado, no qual não há repetição de vértices no caminho, ou seja, só pode existir uma única aresta entre cada par de vértices.

Essa aplicação de caminhos, levando em conta o estudo sobre caminhos disjuntos - no qual dois ou mais caminhos de um mesmo grafo não possuem nenhuma aresta em comum entre si - chegamos ao problema de encontrar o número máximo de caminhos disjuntos de um grafo, o qual apresenta várias aplicações. Esse trabalho apresentará a implemententação do método de Ford-Fulkerson como resolução, inicializando um fluxo unitário para todas as arestas.

2 DESENVOLVIMENTO

Para desenvolvimento do trabalho foi utilizada a linguagem de programção Java, a qual permite uma melhor manipulação de dados na análise de orientação a objetos. O algoritmo de Ford-Fulkerson também auxilia na busca por caminhos disjuntos, uma vez que sua implemetação é utilizada quando se deseja encontrar um fluxo de valor máximo que faça o melhor uso possível das capacidades disponíveis no caminhos de um grafo.

Foram criadas classes distintas que, em conjuto, compõe a implmentação da busca por caminhos disjuntos em grafos, esses serão explicados abaixo:

2.1 Classe GraphRepresentation e Classe Edge

As duas classes são responsáveis por definir e representar o grafo. Na classe Edge, são definidas as variáveis de destino e origem. Já na claase GraphRepresentation definimos as variáveis da matriz gerada pelo grafo depois de realizar a busca de caminhos disjuntos. Além disso, a classe faz a leitura do arquivo, através do FileReader, lendo as informações inicias do grafo e enviando para a classe GFG.

2.2 Classe GFG

A classe GFG é a principal responsável para buscar o número máximo de arestas disjuntas em um grafo, pela implementação de Ford-Fulkerson. Primeiramente, ela valida se o grafo vai de um ponto *s -> t* no grafo residual e salva o predecessor para armazenar o caminho. Ele marca os vértices não visitados e cria um array para todos aqueles que já foram visitados. A partir desse ponto, é feito um busca em largura equanto o array não estiver vazio.

Para encontrar os caminhos disjuntos, ele cria uma rede residual com as capacidades encontradas na busca, ou seja, o fluxo máximo no caminho de cada vértice, e atualiza a capacidade residual das arestas inversas ao longo do caminho, tendo como resultado os caminhos disjuntos do grafo e gerando uma matriz com os resultados obtidos.

2.3 Classe GraphGenerator

O GraphGenerator é responsável por gerar os grafos de cada tipo de acordo com suas especificidades. Ele gera o arquivo do grafo, no formato .txt, que será interpretado pelas outras classes, a fim de gerar o resultado final do número de caminhos disjuntos.

2.4 Classe TrabP02

A classe TrabP02 é responsável por representar todos os caminhos encontrados na classe acima, tendo como basse a representação dos grafos criada no código anterior, partindo da origem até o destino do grafo, lido pelo Scanner(System.in).

2.5 Classe TesteCompleteGraph

Essa classe é responsável gerar os grafos que serão lidos pela classe TrabP02 e testados nos exemplos abaixo. Geramos 4 tamanhos diferentes de grafos - grafos com 10, 100, 1000 e 10000 vértices - para cada um dos três tipos de grafos: circular, nulo e completo.

3 RESULTADOS

Os testes foram realizados com três tipos diferentes de grafos e com instâncias de tamanhos diferentes, demostrando a eficácia e eficiência obtidos na implementação do algoritmo. Todos os resultados apresentam uma amostra do que foi obtido durante a execução de cada tipo de grafo e suas respectivas quantidades de vértices.

3.1 Grafos Completos

Um grafo onde todos os seus vértices tem o grau máximo. Ou seja, existe aresta presente entre todos os pares de vértices. Nesse caso, o grafo de tamanho 10.000 não conseguiu ser executado por

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ser um grafo completo com um número muito alto de vértices, ocasionando em um estouro de memória. Mas, pelas análises dos resultados obtidos, podemos observar que os grafos completos, em sua maioria, possuem como quantidade de caminhos disjuntos o número de vértices - 1. Isso se dá por causa dos ciclos presente em grafos completos, a partir dos quais todas as arestas se conectam.

Figura 1 - Resultados dos Grafos Completos de tamanho 10 e 100

```
Maximum disjunt paths from 1 to 10: 9        With 0.001 s
1 -> 10
1 -> 2 -> 10
1 -> 3 -> 10
1 -> 4 -> 10
1 -> 5 -> 10
1 -> 6 -> 10
1 -> 7 -> 10
1 -> 8 -> 10
1 -> 9 -> 10



Maximum disjunt paths from 1 to 100: 99      With 0.02 s
1 -> 100
1 -> 2 -> 100
1 -> 3 -> 100
1 -> 4 -> 100
1 -> 5 -> 100
1 -> 6 -> 100
1 -> 7 -> 100
1 -> 8 -> 100
1 -> 9 -> 100
1 -> 10 -> 100
1 -> 11 -> 100
1 -> 12 -> 100
```

Fonte: Dados da Pesquisa

Figura 2 - Resultados dos Grafos Completos de tamanho 1000

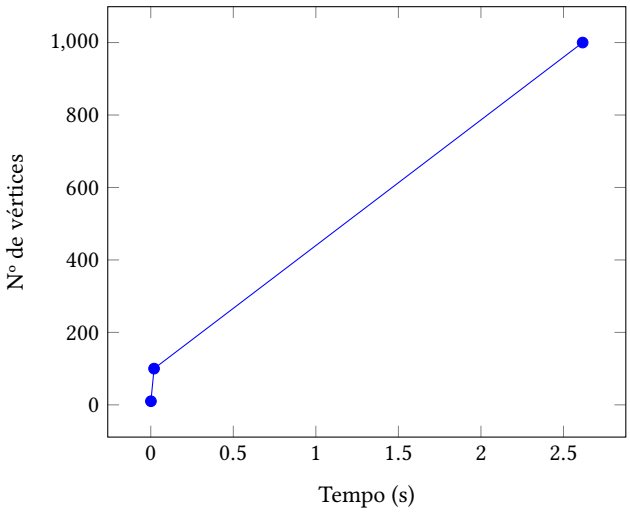
```
Maximum disjunt paths from 1 to 1000: 999        With 2.616 s
1 -> 1000
1 -> 2 -> 1000
1 -> 3 -> 1000
1 -> 4 -> 1000
1 -> 5 -> 1000
```

Fonte: Dados da Pesquisa

O tempo de execução de cada grafo é proporcional ao seu tamanho, uma vez que quando maior o grafo mais custosa se torna a busca feita pelo algoritmo. Estes resultados podem ser analisados abaixo:

Table 1: Tempo de execução dos Grafos Completos

Número de vértices	Tempo de execução (s)
10	0,001
100	0,02
1.000	2,616
10.000	NULL



3.2 Grafos Nulos

O grafo nulo é o grafo sem arestas, ou seja, não há ligação entre os vértices. Tento em vista que grafos nulos não possuem arestas, os valores de caminhos disjuntos serão igual a 0.

Figura 3 - Resultados dos Grafos Nulos tamanho 10, 100, 1.000 e 10.000

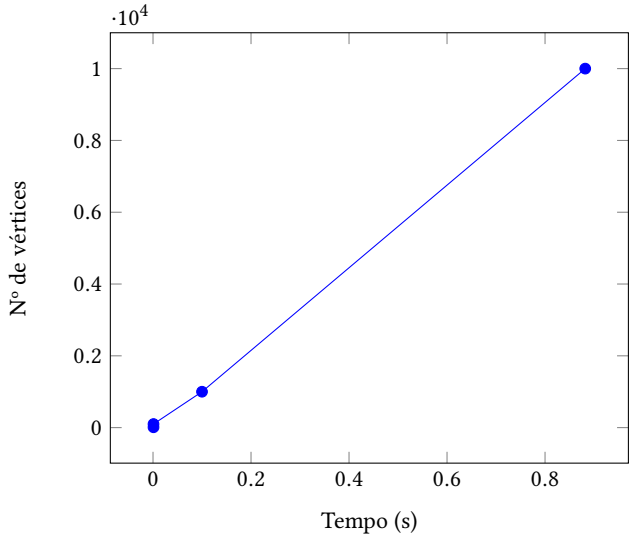
```
Maximum disjunt paths from 1 to 10: 0        With 0.001 s


Maximum disjunt paths from 1 to 100: 0       With 0.001 s


Maximum disjunt paths from 1 to 1000: 0      With 0.01 s


Maximum disjunt paths from 1 to 10000: 0        With 0.882 s
```

Fonte: Dados da Pesquisa

Assim como no grafo completo, o tempo de execução depende do tamanho de vértices, mas como neste caso não há arestas, o tempo de execução se torna muito mais rápido do que o exemplo anterior, pois o custo da busca no gráfo se torna irrisório.

Table 2: Tempo de execução dos Grafos Nulos

Número de vértices	Tempo de execução (s)
10	0,001
100	0,001
1.000	0,1
10.000	0,882



3.3 Grafos Cirulares

Um grafo circular, é um grafo em que é formado por um único ciclo em que cada vértice tenha exatamente um sucessor e um predecessor, ou dois vizinhos, ou seja, o número de vértices é igual ao número de arestas, e cada vértice tem grau 2, exatamente duas arestas incidentes a ele. Analisando os resultados de grafos circulares, podemos perceber que, por ser um único ciclo, somente haverá um caminho disjunto, independente do tamanho do grafo, já que uma aresta retira o ciclo do grafo.

Figura 4 - Resultados dos Grafos Circulares tamanho 10, 100, 1.000 e 10.000

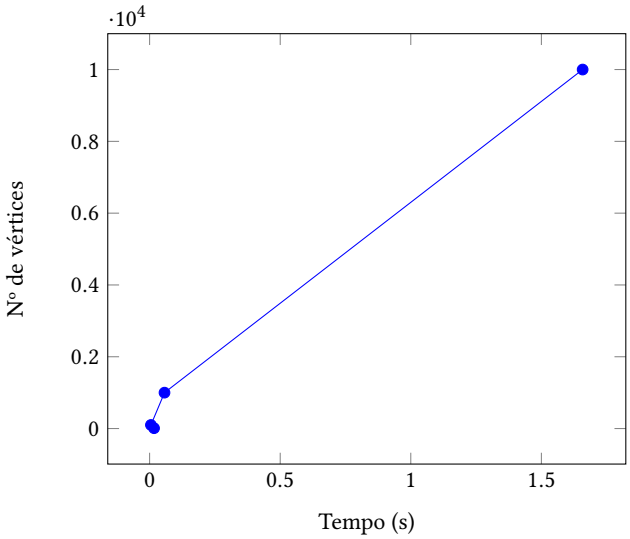
```
Maximum disjunt paths from 1 to 10: 1        With 0.017 s
1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10


Maximum disjunt paths from 1 to 100: 1       With 0.005 s
1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 ->


Maximum disjunt paths from 1 to 1000: 1      With 0.057 s
1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 ->


Maximum disjunt paths from 1 to 10000: 1         With 1.658 s
1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 ->
```

Fonte: Dados da Pesquisa

Analisando o tempo de execução vemos que o tempo é melhor do que o de grafos completos, mas pior do que o de grafos nulos, isso se dá pelo fato de ele fazer uma busca muito menor do que a feita em grafos completos. Porém, na busca, ainda existe uma aresta para cada vertíce a ser explorada, o que o torna menos eficiênte do que o nulo.

Table 3: Tempo de execução dos Grafos Circulares

Número de vértices	Tempo de execução (s)
10	0,017
100	0,005
1.000	0,057
10.000	1,658



4 CONCLUSÃO

A partir do desenvolvimento e resultados obtidos durante o experimento acima podemos concluir que, de acordo com a nossa implementação, o valor do fluxo máximo é igual ao número máximo de caminhos disjuntos de arestas. A complexidade de tempo do algoritmo acima é O(maxflow * E) e é executado um loop enquanto há um caminho aumentante. A eficiência de cada grafo depende da complexidade de busca e tamanho do grafo; Foi possível perceber isso quando analisamos os tempos de execução de cada um dos tipos com tamanhos diferentes.