

Teoria dos Grafos - COS242

Prof. Daniel Ratton e Fábio Botler

Monitora: Maria Luiza Wuillaume

Trabalho - Parte 3

Aluno: Patrick Carneiro Trindade

DRE: 119052764

1 Introdução

1.1 O programa criado

Para este trabalho foi criado um conjunto de funções e estruturas (struct) em Go a fim de explorar grafos ponderados e encontrar o fluxo máximo entre 2 arestas usando o algoritmo de Ford Fulkerson.

O código pode ser encontrado no github, no link abaixo:

<https://github.com/ptk-trindade/graph-project>

1.1.1 Estrutura do código

Abaixo estão listados os principais arquivos do código e o que eles contêm.

Arquivo	Descrição
ioFiles.go	Funções que leem e escrevem no disco
graphDirected.go	Funções que trabalham com o grafo direcionado com peso (capacidade) nas arestas
main.go	A função main deste arquivo permite que o usuário rode e interaja com o programa

1.2 A máquina utilizada

Para rodar este programa foi utilizada uma máquina Windows, com um processador Intel i5 11ª geração e 16GB de memória RAM.

Device name	C11-MZN7T7HEK5U
Full device name	C11-MZN7T7HEK5U.dir.svc.accenture.com
Processor	11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 1.38 GHz
Installed RAM	16.0 GB (15.4 GB usable)
Device ID	BB13073A-4DE0-4B12-81E0-24F9434244A6
Product ID	00330-80000-00000-AA815
System type	64-bit operating system, x64-based processor
Pen and touch	Touch support with 10 touch points

imagem 1: especificações da máquina utilizada

1.3 Os grafos analisados

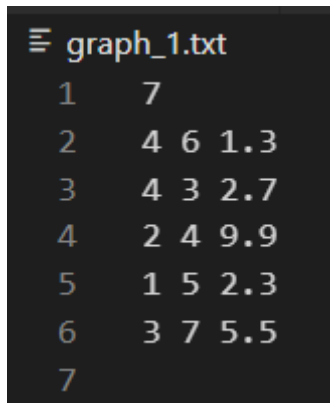
Para este relatório, foram analisados 8 grafos, que possuem as características descritas abaixo:

nome	quantidade de vértices	quantidade de arestas	capacidade das arestas			
			máximo	mínimo	médio	mediana
grafo 1	1 000	6 009	10	1	5.54	6.0
grafo 2	1 000	6 026	10 000	1	4 988.52	5 018.0
grafo 3	10 000	60 017	10	1	5.53	6.0
grafo 4	10 000	60 002	10 000	1	5 005.08	5 011.5
grafo 5	100 000	1 099 802	10	1	5.50	5.0
grafo 6	100 000	1 099 793	10 000	1	5 002.91	5 003.0
grafo 7	1 000 000	10 999 698	10	1	5.50	5.0
grafo 8	1 000 000	10 999 731	100 000	1	50 011.00	50 004.0

Essas e algumas buscas pelo grafo podem ser encontradas na pasta “output_log” do repositório. Mais informações sobre os arquivos de saída na seção a seguir.

2 Input e Output (I/O)

Para input do grafo, o usuário deve criar um arquivo txt. O nome do arquivo será perguntado assim que o programa começa sua execução, e o arquivo deverá ter o seguinte formato:



```
graph_1.txt
1 7
2 4 6 1.3
3 4 3 2.7
4 2 4 9.9
5 1 5 2.3
6 3 7 5.5
7
```

imagem 2: arquivo de input “grafo.txt”

Neste, a primeira linha representa o número de vértices do grafo, e cada linha seguinte representa uma aresta com os dois vértices (inteiros) e o peso da aresta (float ou inteiro).

O programa manipula então os seguintes arquivos:

I/O	nome	descrição
input	grafo.txt*	Número de vértices e arestas do grafo
output	output.txt	Informações básicas sobre o grafo
output	fordFulkersonV1.txt	Todas as arestas do grafo mostrando suas capacidades e qual o fluxo em cada uma delas quando se alcança o fluxo máximo. Utilizando as 2 versões do código (as quais serão descritas mais à frente)
output	fordFulkersonV2.txt	

*Pode ser qualquer nome, porém não é recomendado utilizar o mesmo nome de um dos arquivos de output.

3 Versões criadas

O Algoritmo de Ford Fulkerson foi implementado de 2 formas diferentes. O que muda entre cada uma delas é a forma de se encontrar o caminho entre os vértices source (S) e sink (T).

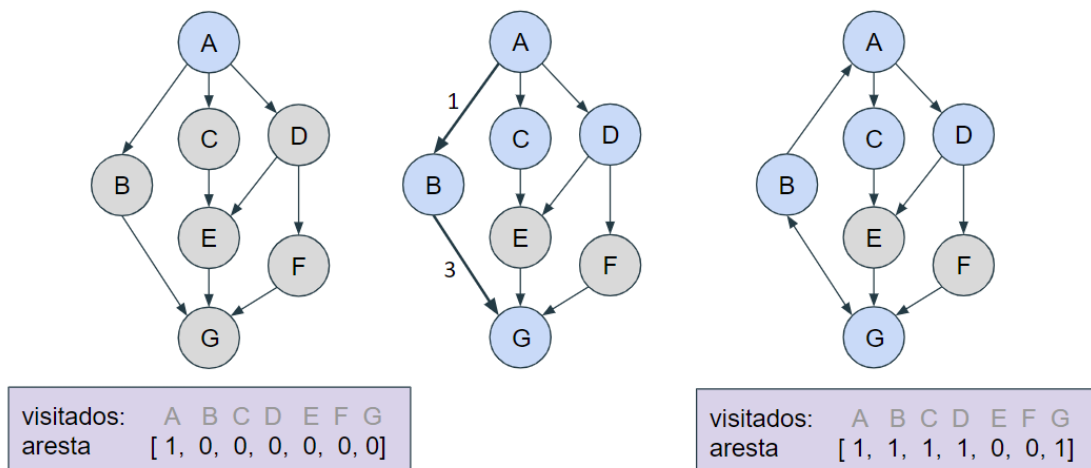
3.1 Primeira versão (1 BFS)

Na primeira, roda-se uma BFS em S até descobrir-se o ponto T. Passa por todas as arestas do caminho encontrado para encontrar a de menor capacidade e depois atualiza as capacidades dessas arestas e de suas complementares.

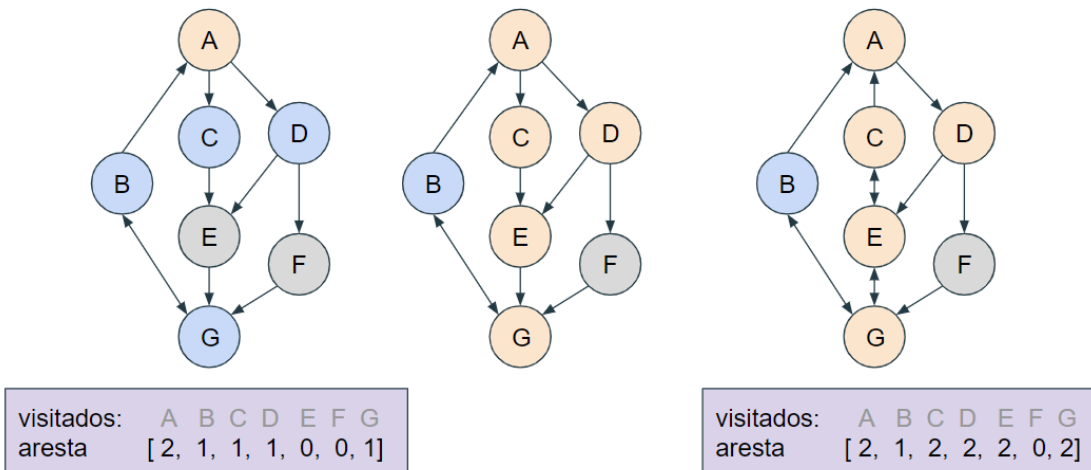
Com as arestas atualizadas, há uma nova chamada para função de BFS até que não haja mais caminho entre S e T.

3.2 Segunda versão (2 BFS)

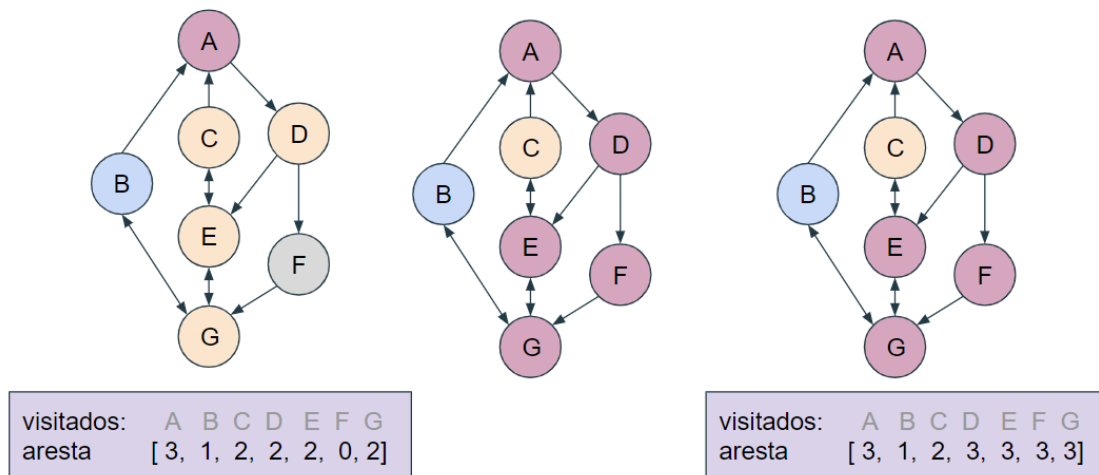
Já na segunda versão, reutilizamos a lista de visitados para as BFS futuras. Então criamos a lista de visitados e vamos preenchendo-a com o id 1 (azul).



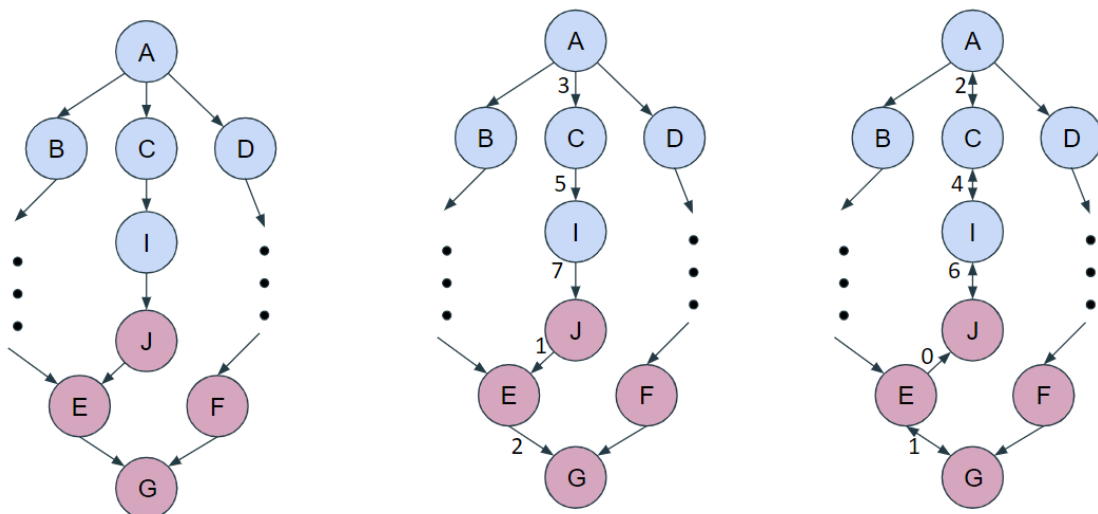
Na próxima BFS, usamos a mesma lista de visitados, porém usando o id 2 (laranja).



Depois o id 3 (roxo) e assim sucessivamente.

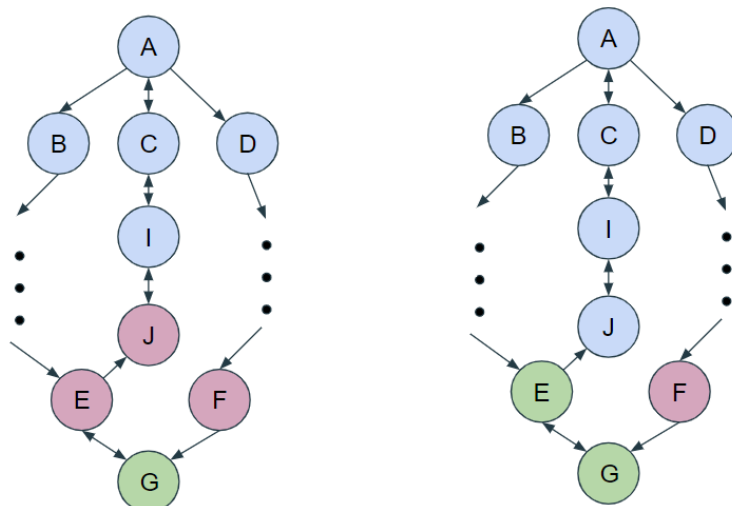


Além disso, rodam-se 2 BFS, uma partindo de A (origem) e outra de G (destino). Quando essas 2 BFS se encontram, temos então um caminho. Neste, iremos encontrar a aresta de menor capacidade e atualizar as capacidades das arestas.



Em nossas 2 BFS, apenas aquela que tiver a(s) aresta(s) de menor capacidade será refeita*. Assim “salvamos” parte do processamento. No exemplo a BFS roxa tinha menor capacidade e será descartada. Para a próxima BFS “inferior” se utilizará a cor verde, veja na imagem abaixo:

*Em alguns casos as menores arestas estão nas duas árvores e as duas precisam ser refeitas.



4 Estudos de caso

Foram feitos vários testes de performance com cada uma das versões do programa rodando para cada um dos 8 grafos, eles podem ser vistos abaixo:

4.1 Fluxos máximos e tempo de execução

Para cada grafo e versão de código, foi calculado o fluxo máximo 10 vezes e cronometrado o tempo de execução. Os resultados obtidos foram os seguintes:

Grafo	Versão 1		Versão 2	
	Fluxo máximo (entre 1 e 2)	Tempo médio de execução (em segundos)	Fluxo máximo (entre 1 e 2)	Tempo médio de execução (em segundos)
grafo 1	284	0.00442	284	0.00173
grafo 2	276820	0.00779	276820	0.00158
grafo 3	291	0.07947	291	0.02282
grafo 4	253278	0.13651	253278	0.02384
grafo 5	618	4.39314	618	0.47168
grafo 6	548517	8.00897	548517	0.63860
grafo 7	611	44.97805	611	8.95774
grafo 8	5382665	114.20848	5382665	9.42628

Mais informações sobre os fluxos podem ser encontradas em:

https://github.com/ptk-trindade/graph-project/tree/main/output_log