Documentação - Trabalho Prático 2
Teoria dos Grafos e Computabilidade

Vinícius Henrique Giovanini

[1]ICEI – Pontifícia Universidade Católica de Minas Gerais (PUC)
Belo Horizonte – MG – Brazil

{vgiovanini}@sga.pucminas.br

Resumo. *Realização do desenvolvimento de um programa na linguagem* ***Python****, visando resolver o problema para determinar o número máximo de caminhos disjuntos em arestas, além da realização da análise do desempenho em grafos de diferentes topologias e diferentes tamanhos.*

1. Introdução

Em teoria de grafos e computabilidade, o problema de determinar o número máximo de caminhos disjuntos em arestas, consiste em, encontrar o máximo de caminhos entre um vértice origem (x) para um vértice destino (y), analisando todos os ramos possíveis do vértice inicial. A teoria na qual foi baseada para a realização da busca se trata na utilização do fluxo de arestas, considerando que cada aresta do grafo tenham fluxo com valor **um**, e quando o destino (y) for encontrado inverte todo o caminho percorrido, e após isso inicia-se outra busca com o próximo vértice da origem (x).

2. Estrutura dos Códigos

O trabalho foi codificado na linguagem Python utilizando quatro classes, a primeira chama **createKgraph.py** que visa criar grafos com topologia K, a segunda classe chamada **createRandomGraph.py** gera grafos com topologia aleatória, a terceira classe é chamada **createGraphSemiConnect** que tem a função de criar um grafo com todos os vértices conexos no primeiro vértice, e não podendo realizar conexões no anterior, e a classe **initMatriz.py** tem a função de criar a matriz de adjacência de um grafo qualquer, para a próxima classe **searchFluxoMax.py** se encarregar de encontrar a quantidade de fluxos presentes, e a classe **core.py** é a principal, utilizada para chamar as demais classes. Os grafos gerados são armazenados na pasta **db**. Os códigos gerados podem ser encontrados no GitHub de Grafos na parte de Atividade Avaliativas e TP02 [Giovanini 2022b].

3. Geração dos Grafos

3.1. K-Grafos

Uma das quatro topologias escolhidas para a criação foram Grafos K de tamanho quatro, eles são interligados por duas arestas direcionadas do primeiro conjunto de vértice K para o segundo, e o tamanho do grafo é passado como parâmetro da função, que determina a quantidade de conjuntos K quatro que serão gerados. A Figura 1 exemplifica um grafo de K quatro de tamanho dois.

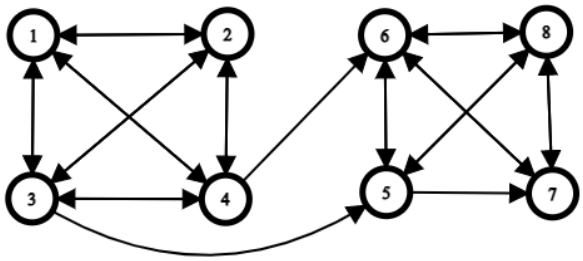
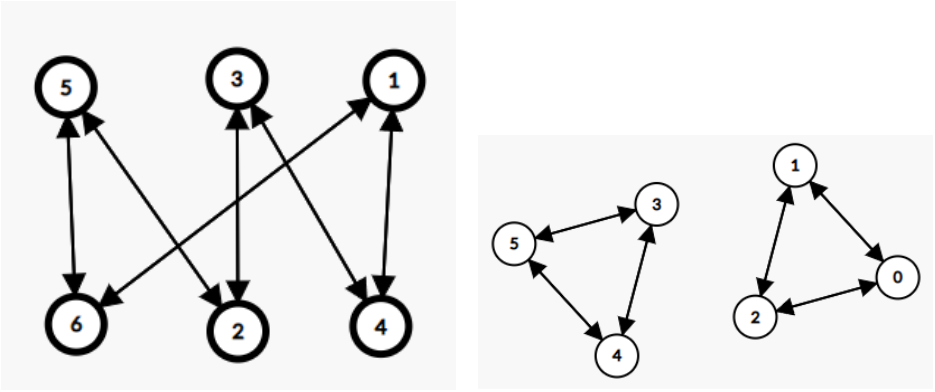


Figure 1. Representação de um Grafo K-Quatro de tamanho dois

3.2. Grafos Aleatórios

A segunda geração trata-se em grafos aleatórios, na qual a topologia é gerada randomicamente pela classe **createRandomGraph.py**, quando a mesma recebe como parâmetro a quantidade de vértices desejados, e quando realiza a geração dos pares origem destino, ela gera randomicamente em um intervalo de dois a dez a quantidade de vértices que irão sair da origem, dessa forma criando um grafo com os destinos aleatórios e a quantidade de destinos de cada vértice aleatórios também. Essa classe também pode gerar um grafo com mais de um componente conexo, já que o destino de cada vértice é gerado aleatoriamente, podendo ocorrer ciclos entre alguns vértices como pode ser visto na Figura 2(a) e 2(b).



(a) Grafo com um componente conexo.

(b) Possível geração aleatória do grafo com mais de um componente conexo.

Figure 2. Representação de algumas possibilidades de conexão

3.3. Grafos Circulares

A terceira geração consiste em um grafo Circular, na qual foi gerado de maneira sequencial, com cada vértice possuindo grau dois, e o primeiro vértice do grafo tem conexão com o último vértice, como pode ser visto em um exemplo de um grafo tamanho seis na Figura 3. Este grafo foi gerado pela classe **gerarGraph.py** presente na criação do primeiro trabalho prático [Giovanini 2022a].

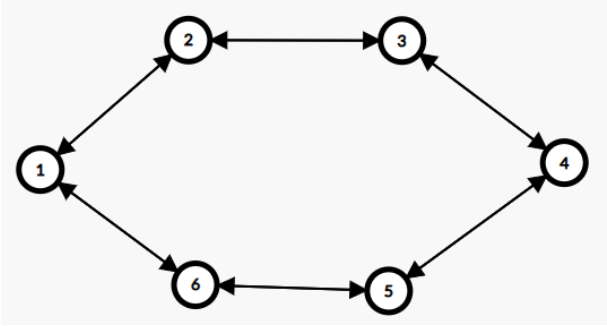


Figure 3. Representação de um Grafo Circular

3.4. Grafos Semi Conexos

A geração de grafos Semi Conexos trata-se de um grafo na qual o vértice inicial que será sempre o **um** tem arestas direcionadas para os demais, os vértices seguintes não poderão mais ter conexão com os já implementados, dessa forma o segundo vértice não terá conexão com o **um**, mas terá com os demais, assim gerando até chegar o último vértice, que receberá todas as arestas, mas não sairá nenhuma aresta do mesmo, como pode ser visualizado na Figura 4.

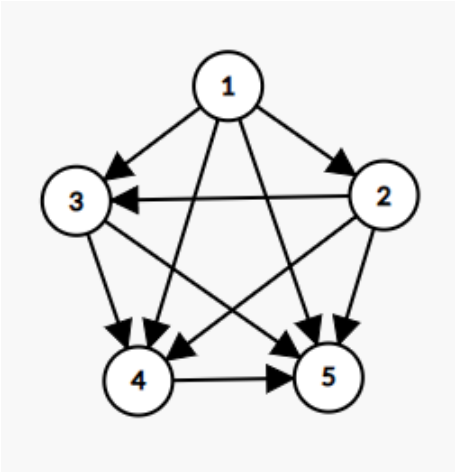


Figure 4. Representação de um Grafo Semi Conexo

4. Busca com Fluxo Máximo

4.1. Ideia Básica da Pesquisa

A pesquisa pelo número máximo de caminhos acontece com duas classes, a **initMatriz** e a **searchFluxoMax**, onde a primeira respectivamente é responsável por ler o arquivo e gerar uma matriz de adjacência contendo todas as arestas de cada vértice, a segunda é a principal da pesquisa, pois através dela recebendo essa matriz, e o vértice origem e destino, a mesma realiza uma busca por cada ramificação, sempre testando através do método **analisarVerticesAdj** se no nó atual tem alguma aresta direta para o destino, caso não tenha ela realiza dois testes em sequência, o primeiro através do método **selecionarVerticeNaoPercorrido** testa se o nó tem mais de um caminho a ser visitado, caso tenha somente um e seja caminho repetido, ele é percorrido novamente para testar se existem

arestas não trilhadas, caso contrário o método retorna o conjunto de aresta na qual não foi visitado ainda, e o segundo teste ocorre logo após, para selecionar a aresta a ser percorrida entre o conjunto retornado, utilizando o método **selecionandoMenorElemento**, na qual escolhe o vértice com menor valor numérico, realizando esse procedimento até encontrar o destino ou um vértice folha.

Na figura 5, podemos visualizar um diagrama com o fluxo da pesquisa, na qual as margens retangulares são referentes a **classe**, e as circulares refentes a **métodos e ações** realizadas. Pode observar que a classe **serachFluxMax** realiza uma repetição com os métodos **searchPrincipal** cujo o objetivo é analisar as arestas do vértice origem, e **buscaA** que percorre todo o ramo inicial até uma folha ou voltar ao vértice origem, caso ele ocorre dele percorrer todos os vértices e voltar para a origem, o primeiro método respectivamente irá verificar se existe alguma outra aresta disponível para a busca, caso não existe ela retorna um caminho vazio, caso tenha ela irá mandar o conjunto de vértices adjacentes dessa aresta para o buscaA realizar a procura.

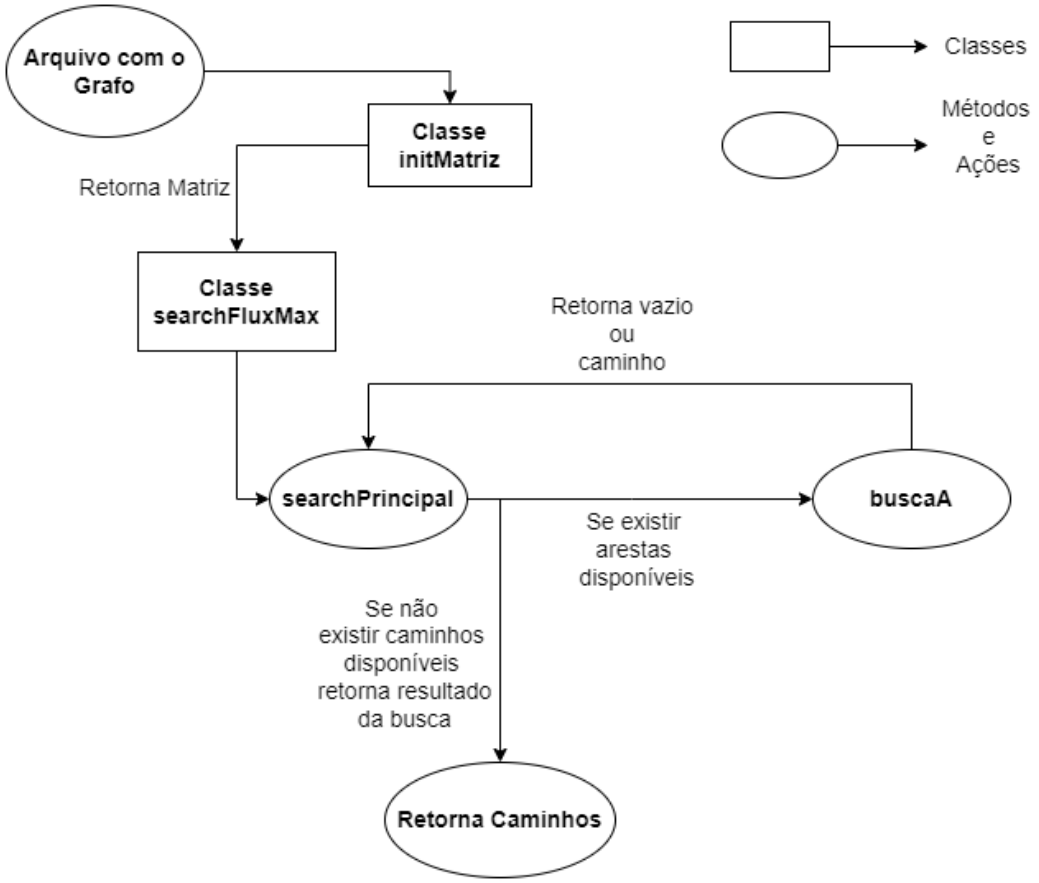
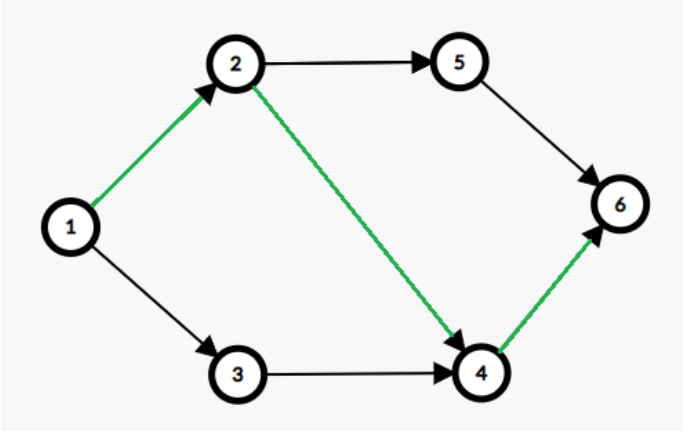


Figure 5. Diagrama do Loop de Repetição da Busca

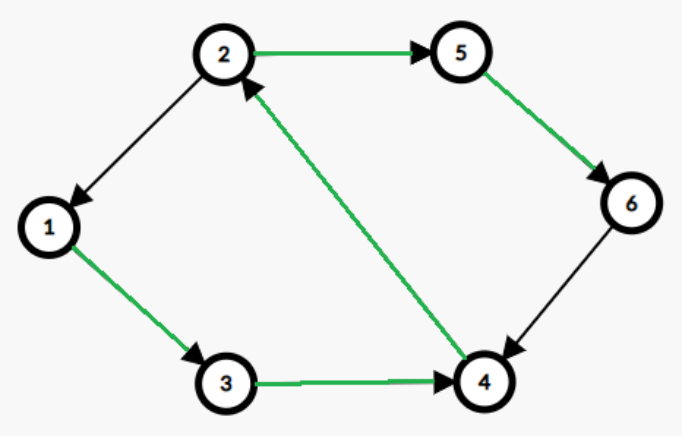
4.2. Ramificação com Sucesso

Quando o algoritmo sai de um dos ramos do vértice inicial, o mesmo salva seu caminho percorrido, e quando é encontrado o destino ele irá voltar pelo caminho percorrido

invertendo todas as arestas, pois o fluxo que é de **um** foi preenchido. No conjunto de Figuras 6, podemos observar na imagem 6(a) a descoberta do caminho de origem **um** para o destino **seis** na primeira iteração, e na imagem 6(b) podemos observar a inversão das arestas, e a busca para encontrar o vértice destino através de uma aresta invertida (4-2), assim chegando a dois caminhos percorridos.



(a) Primeira Iteração



(b) Segunda Iteração com Arestas Invertidas

Figure 6. Representação do Método de Busca em um Grafo de Seis Vértices

4.3. Ramificação sem Sucesso

Quando o algoritmo chega no vértice folha de algum ramo e não encontra o destino, ele retorna todos os nós, testando se nos mesmos existem alguma aresta não percorrida, e caso exista ele percorre em busca do destino, e caso encontre um nó folha novamente ele retorna para testar outros caminhos. Caso ele não encontre o destino nesse ramo e volte para o vértice origem, o algoritmo encerra a pesquisa nessa ramificação e procura no próximo ramo, caso não encontre o destino em nenhum ramo do vértice inicial ele retorna vazio, pois o caminho entre origem e destino é inexistente.

Na Figura 7, podemos analisar um grafo de oito vértices, com a origem sendo vértice um e o destino o oito, dessa maneira o método irá realizar os três testes em cada vértice

percorrido, o primeiro para verificar se alguma aresta adjacente leva para o vértice destino, o segundo para priorizar arestas que levam para nós não percorridos, caso não exista ele repete o nó, e selecionando sempre o vértice com menor valor numérico. Dessa maneira podemos perceber que o primeiro caminho é o **vermelho** que chega no nó folha quatro, logo em seguida ele realiza a volta pelo caminho **azul**, verificando que o nó três não tem mais aresta e voltando para o dois, analisando que possui duas arestas a serem percorridas, em seguida ele realiza os testes novamente, e detecta que não possui aresta direta para o vértice destino, e retorna o conjunto de aresta não percorridas nove e cinco, selecionando o menor elemento cinco representado pelo caminho **azul**, sendo um nó folha, voltando pelo caminho **amarelo** e testando o nove identificando o mesmo resultado, dessa maneira voltando através do caminho **verde** para o vértice inicial, e percorrendo o vértice sete pelo caminho **verde** quando é realizado o teste pelo método **analisarVerticesAdj**, que analisa os adjacentes detectando o vértice destino, encontrando um caminho, dessa forma voltando para o vértice inicial e detectando que não existe mais arestas a serem percorridas, encerrando a busca e retornando o caminho encontrado.

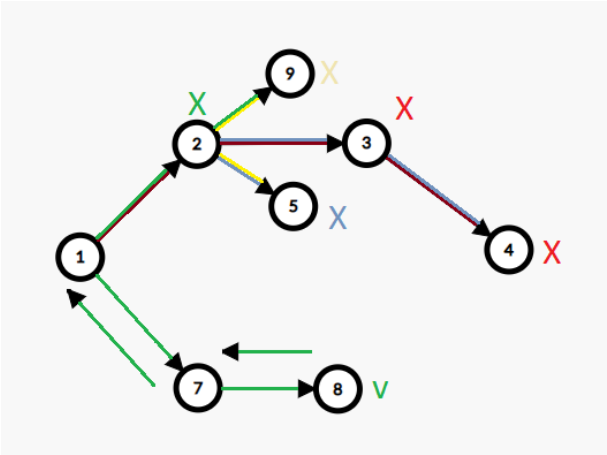


Figure 7. Representação de uma Busca em um grafo com oito vértices.

5. Testes e Resultados

5.1. Tempos Gerais

A imagem presente na Figura 8, consiste em um gráfico da relação do tempo de execução em segundos pelo tamanho do grafo, sem considerar o teste com tamanho de vinte mil. Visualizando a linha laranja percebe-se que o teste do grafo Semi Conexo é mais lento, já a linha verde referente ao K-Quatro contem tempos por quantidade de vértices mais constante, também percebe-se que o grafo circular retratado pela linha azul, por ser sequencial tem o melhor tempo de execução, quando comparado ao grafo aleatório visualizado pela linha vermelha, que possui um tempo ótimo, porém tempos inconsistentes, pelo fato do teste resultar em vários componentes conexos, gerando assim um teste rápido e sem um resultado plausível.

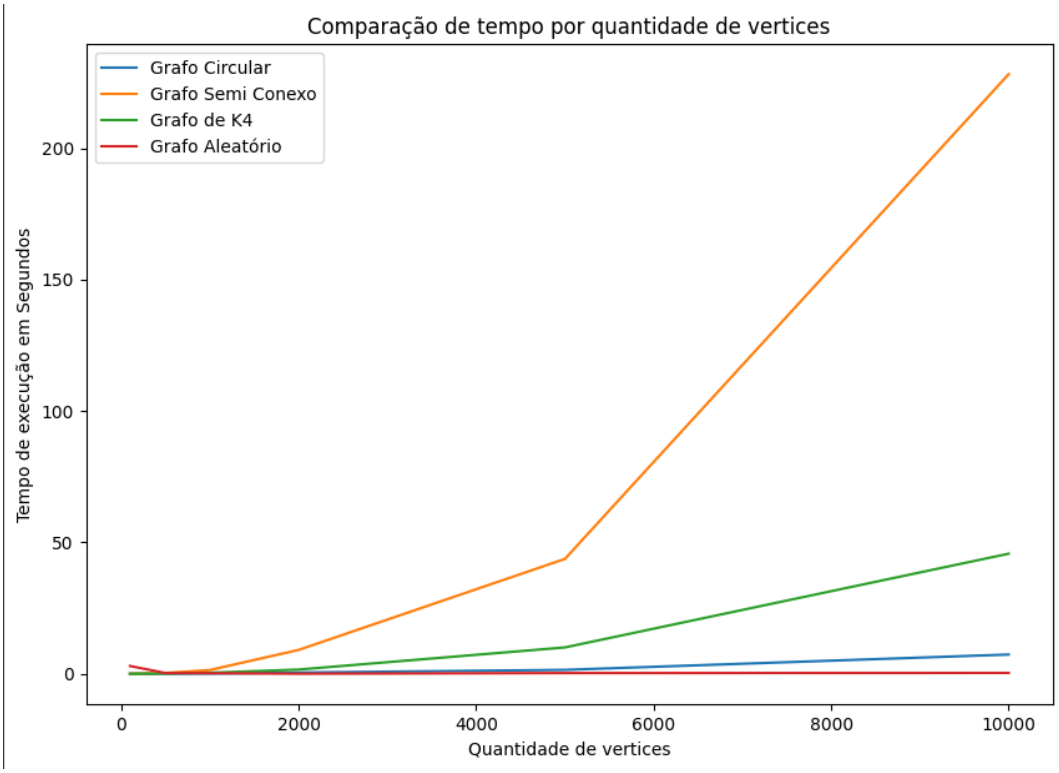


Figure 8. Gráfico Quantidade de Vértices x Tempo de Execução (Sem Teste 20.000)

5.2. Grafos K-Quatro

Nos testes dos grafos K-Quatro, foi empregado como parâmetro para a análise o número total de vértices presente, e não a quantidade de conjuntos K-Quatro. Foram utilizados as duas extremidades para teste, são elas o vértice inicial de todos os grafos, o **um**, e o **último vértice** que é correspondente ao tamanho do mesmo. A quantidade de caminho sempre será igual a dois, pois os conjuntos K-Quatro estão interligados por apenas duas arestas direcionadas, os resultados obtidos podem ser visualizados na Tabela 1. Quando observado a quantidade de vértice e o tempo de execução, podemos perceber um comportamento quase linear, visualizado no Gráfico 9.

Quantidade de Vértices	Vértice Origem	Vértice Destino	Tempo de Pesquisa (S)	Qtd de Caminhos Encontrados
100	1	100	0.011	2
500	1	500	0.089	2
1.000	1	1.000	0.396	2
2.000	1	2.000	1.605	2
5.000	1	5.000	10.032	2
10.000	1	10.000	45.699	2
20.000	1	20.000	191.323	2

Table 1. Tabela de Resultados Grafos K-Quatro

Figure 9. Comparação de Tempo x Quantidade de Vértices - K4

5.3. Grafos Circulares

Os grafos circulares são grafos criados de maneira sequencial, dessa forma os testes foram gerados usando a origem como o vértice **um** e o destino o **último vértice do grafo**, assim podemos perceber que o tempo aumenta a medida que o tamanho do grafo também aumenta, como pode ser visto na Tabela 2 e no Gráfico de comparação 10. Logo podemos perceber que a quantidade de caminhos será no máximo dois, caso seja pesquisado algum destino a partir do primeiro nó, no qual tem conexão para toda a sequência de vértices, e também para o último nó do grafo.

Quantidade de Vértices	Vértice Origem	Vértice Destino	Tempo de Pesquisa (S)	Qtd de Caminhos Encontrados
100	1	100	0.003	2
500	1	500	0.028	2
1.000	1	1.000	0.083	2
2.000	1	2.000	0.466	2
5.000	1	5.000	1.490	2
10.000	1	10.000	7.324	2
20.000	1	20.000	30.269	2

Table 2. Tabela de Resultados Grafos Circulares

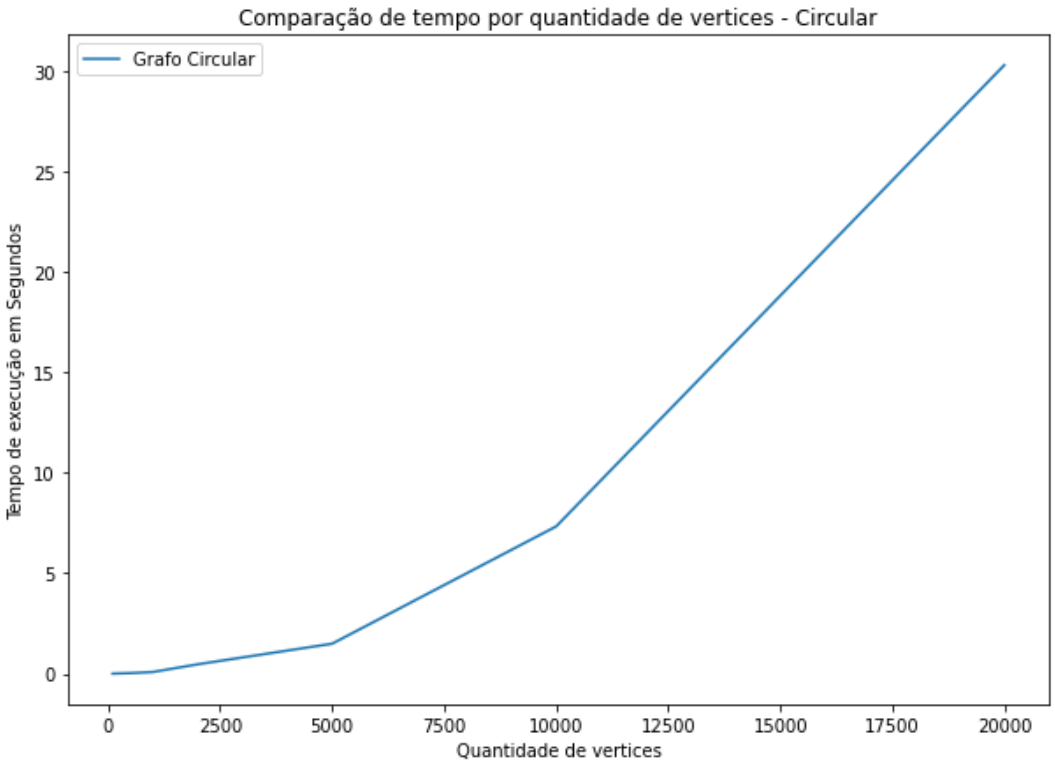


Figure 10. Comparação de Tempo x Quantidade de Vértices - Circular

5.4. Grafos Semi Conexos

Os grafos chamados Semi Conexos consiste em um grafo com o vértice inicial com aresta direcionada para todos os seus sucessores, recebendo uma aresta direcionada de todos os seus predecessores, dessa maneira podemos observar que o tempo de busca em cada nó vai diminuindo no momento que chega mais perto do final do grafo, como pode ser visualizado na Tabela 3, sendo constatado que no primeiro vértice é onde a busca terá maior tempo, pois o mesmo possui arestas direcionadas saindo para os demais nós do grafo, pode-se perceber também que a partir de um grafo de dez mil vértices o tempo de pesquisa aumenta consideravelmente, podendo ser consultada no Gráfico 11, isso ocorre devido à quantidade de arestas presentes em vértices iniciais, elevando a quantidade de pesquisas. Os vértices testados sempre foram o nó inicial **um** e o **último nó** do grafo, dessa maneira a quantidade de caminhos descobertos nesse teste será sempre o **tamanho total de vértices menos um**.

Quantidade de Vértices	Vértice Origem	Vértice Destino	Tempo de Pesquisa (S)	Qtd de Caminhos Encontrados
100	1	100	0.024	99
500	1	500	0.321	499
1.000	1	1.000	1.433	999
2.000	1	2.000	9.085	1999
5.000	1	5.000	43.710	4999
10.000	1	10.000	228.168	9999
20.000	1	20.000	4950.682	19999

Table 3. Tabela de Resultados Grafos Semi Conexos

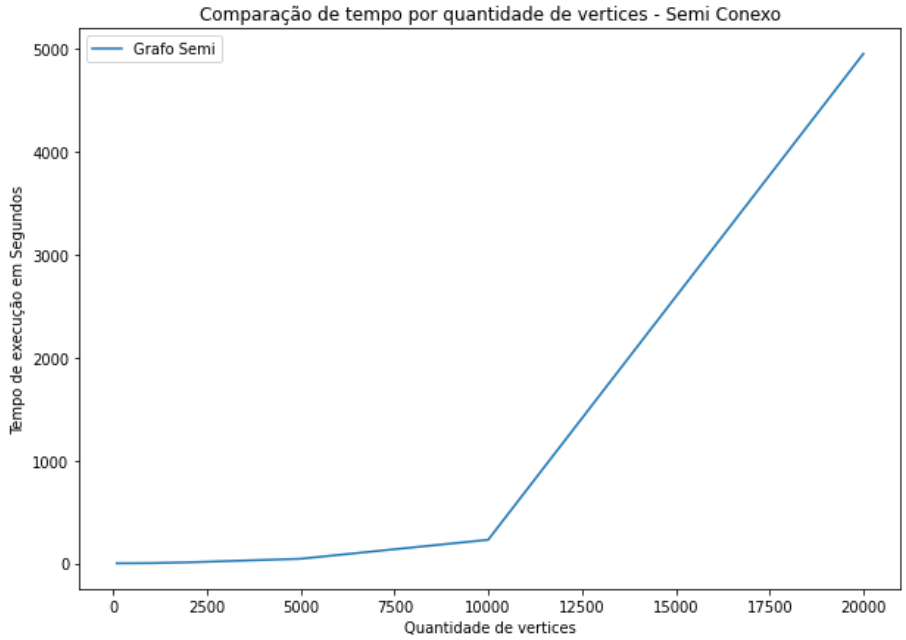


Figure 11. Comparação de Tempo x Quantidade de Vértices - Semi Conexos

5.5. Grafos Aleatórios

Os resultados para a geração dos grafos aleatórios não tiveram boa consistência na hora dos testes, podendo ser consultados na Tabela 4. Isso se deve ao fato que na geração da quantidade de arestas saindo de um determinado nó, seu destino é gerado de maneira randômica, podendo criar assim diversos componentes conexos, e com isso diminuindo o tamanho da busca do algoritmo. Foi testado diferentes valores origem e destino para tentar encontrar algum caminho, mas na maioria dos testes não foi obtido um tempo considerável de busca, nem o retorno de um caminho, isso ocorreu para grafos de todos os tamanhos como pode ser visto no Gráfico 12, no qual o tempo de execução da busca não mantém um padrão relacionado aos tamanhos dos grafos.

Quantidade de Vértices	Vértice Origem	Vértice Destino	Tempo de Pesquisa (S)	Qtd de Caminhos Encontrados
100	1	5	2.961	0
500	2	50	0.289	0
1.000	3	2	0.231	0
2.000	15	20	0.021	0
5.000	4	2	0.284	0
10.000	200	10.000	0.327	0
20.000	150	20.000	0.598	0

Table 4. Tabela de Resultados Grafos Aleatórios

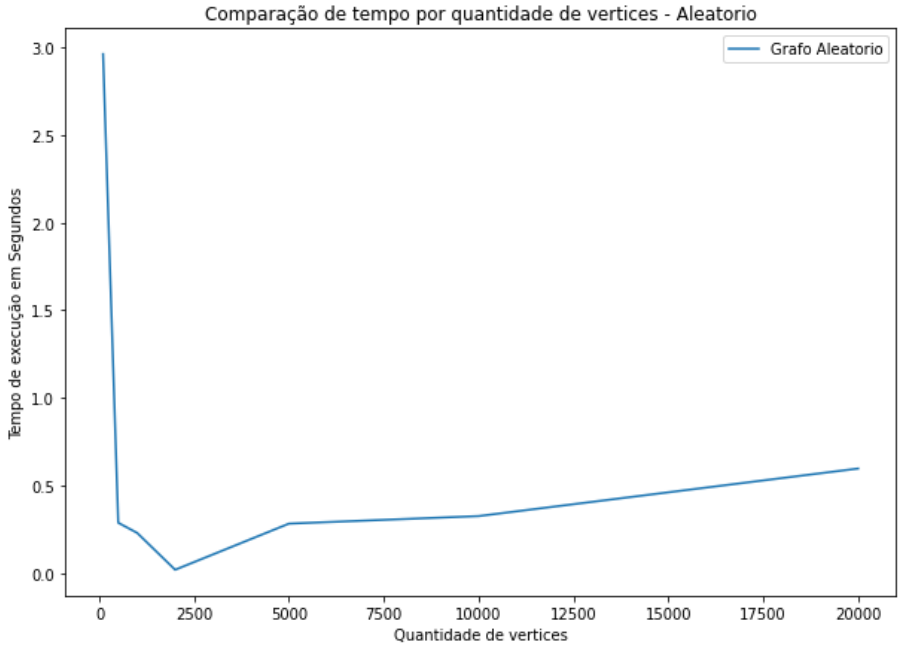


Figure 12. Comparação de Tempo x Quantidade de Vértices - Aleatórios

5.6. Conclusão

Durante o desenvolvimento do trabalho podemos perceber que, o método para realização de busca não é eficiente para grafos densos, podendo constatar nos teste dos grafos Semi Conexos, que com pequenos aumentos de vértices aumentam consideravelmente o número de arestas, o grafo K-Quatro teve uma eficiência mediana, visto que o método de busca encontrará no máximo dois caminhos entre origem e destino. O melhor grafo em questão de eficiência é o grafo circular, no qual ele é sequencial, e chegando no último vértice do grafo possui uma única conexão para o vértice inicial, provendo assim uma ótima eficiência para o algoritmo de busca. Os grafos aleatórios não tiveram uma boa qualidade de testes, visto que a geração do mesmo sorteia o número de aresta saindo de cada nó em um intervalo de um a dez, e considerando que um grafo poderia possuir diversos nós com somente uma ou duas arestas saindo, e ainda podendo ter mais de um componente conexo, limitou a qualidade das buscas e dos testes.

References

Giovanini, V. H. (2022a). Trabalho prático 1 - repositório do github de vinícius h. https://github.com/viniciushgiovanini/Grafos/tree/main/Atividade%20Avaliativas/TP01. Accessed: 2022-10-30.

Giovanini, V. H. (2022b). Trabalho prático 2 - repositório do github de vinícius h. https://github.com/viniciushgiovanini/Grafos/tree/main/Atividade20Avaliativas/TP02. Accessed: 2022-12-09.