Primeiro Trabalho de Algoritmos Avançados - Corte mínimo de um grafo

João Reis, 98474

***Resumo* - Este documento tem como intuito documentar a análise realizada a dois algoritmos de pesquisa diferentes: Pesquisa Exaustiva e Pesquisa Gulosa. Esta análise advém da resolução de um pequeno problema envolvendo grafos: descobrir o corte mínimo de um determinado grafo. Ao longo do documento é explicado o raciocínio para cada um dos algoritmos. No final, é realizada uma análise dos resultados obtidos.**

***Abstract* - This paper is intended to document the analysis performed on two different search algorithms: Exhaustive Search and Greedy Search. This analysis comes from solving a small problem involving graphs: finding the minimum cut of a given graph. Throughout the paper the reasoning for each of the algorithms is explained. At the end, an analysis of the results obtained is performed.**

***Key words* - Python, Pesquisa Gulosa, Pesquisa Exaustiva, Vértice, Grafo, Tempo de Execução, Complexidade Computacional**

I. INTRODUÇÃO

Para o primeiro trabalho da unidade curricular Algoritmos Avançados, é proposto a resolução de um pequeno problema envolvendo grafos.

Para tal, primeiramente, foi necessário fazer uma geração de N grafos aleatórios não direcionados, com as seguintes características: os vértices são coordenadas 2D com os valores inteiros aleatórios compreendidos entre 1 e 20; utilizar as seguintes probabilidades, 12.5%, 25%, 50% e 75%, para calcular o número de arestas para um grafos; e, gerar grafos sucessivamente maiores, isto é, com cada vez mais vértices. De seguida, resolver o problema com os grafos gerados utilizando dois algoritmos diferentes: Algoritmo de pesquisa exaustivo e Algoritmo de pesquisa guloso. Por último, recolher dados e analisar a complexidade computacional de cada um dos algoritmos.

Para executar o código, basta executar um dos comandos indicados abaixo, dependendo do algoritmo que pretende analisar.

```
$ python3 exhaustive_search.py
$ python3 greedy_search.py
```

Este relatório é acompanhado com todos os ficheiros python produzidos para implementar os algoritmos.

II. PROBLEMA

Antes de explicar o problema em si, é necessário compreender a estrutura criada para representar os grafos não direcionados e sem pesos nas arestas associadas.

Um grafo será definido por um dicionário, onde a chave será um vértice e o seu valor será uma lista de todos os vértices à qual esse está ligado, isto é, uma lista de todas as arestas associadas a esse vértice. Um exemplo de um grafo poderia ser o seguinte:

```
{'a': ['d'], 'b': ['d', 'c'], 'c': ['b',
'd'], 'd': ['b', 'a', 'c']}
```

Os vértices são representados também utilizando um dicionário, onde a chave é uma representação alfabética e o seu valor será as coordenadas onde se situa esse vértice, tal como indica a seguinte representação:

```
{'a': (9, 2), 'b': (13, 2), 'c': (2, 14),
'd': (3, 17)}
```

Tendo estes conhecimentos, o problema que é proposto é encontrar um corte mínimo para um dado grafo não direcionado G(V, E), com n vértices e m arestas. O corte mínimo de G é uma divisão dos vértices do gráfico em dois conjuntos complementares, S e T, tais que o número de arestas entre o conjunto S e o conjunto T é o menor possível, ou seja, o número de arestas que separam esses conjuntos deve ser o menor.

Os próximos dois tópicos deste relatório explicarão como foi implementada a solução para este problema, utilizando os dois algoritmos pedidos.

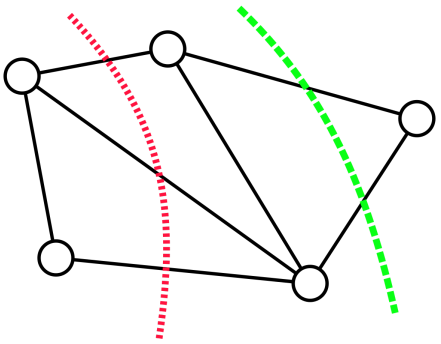

Fig. 1 - Representação gráfica do problema (o corte verde é uma solução)

III. ALGORITMO DE PESQUISA EXAUSTIVO

A primeira solução implementada foi a pesquisa exaustiva ou força bruta, que encontra todas as soluções possíveis ao problema.

Para cada grafo, a função *search* é chamada. Nesta função é onde ocorre a procura pelas soluções que satisfazem o problema. Para todos os subconjuntos possíveis dos vértices desse grafo, sendo cada subconjunto o conjunto S, descobre-se as arestas que serão cortadas devido à separação dos conjuntos S e T. Em seguida, o dicionário *min_cut_edges* é atualizado, sendo a chave o conjunto S e o seu valor as arestas "cortadas", caso o número de arestas for inferior ao número de arestas até aí então guardadas no dicionário. No final, é retornada a solução (*min_cut_edges*), o número de iterações e o tempo de execução.

Para otimizar o tempo de execução, os subconjuntos possíveis dos vértices é filtrado, isto é, o programa vai ignorar quando o conjunto S é o conjunto vazio ou é o conjunto de todos os vértices ou, ainda, se o complementar do conjunto S, conjunto T, já foi analisado para evitar que se repita duas vezes a mesma procura.

Por exemplo, um grafo com vértices = ['a', 'b', 'c'], e os conjuntos S = ['a', 'b'] e T = ['c'] já foram analisados, o programa irá ignorar quando for o caso onde S = ['c'] e T = ['a', 'b'], uma vez que a solução será a mesma.

Este algoritmo como analisa todos os casos possíveis, apesar de ser simples, é bastante dispendioso. Isto porque a sua complexidade computacional será de 2^n, sendo n o número de vértices, dado que esse número será o número total de subconjuntos possíveis dado um grafo com n vértices.

IV. ALGORITMO DE PESQUISA GULOSO

Enquanto que a pesquisa exaustiva procura por todas as soluções, a pesquisa gulosa apenas retorna uma solução, a primeira que encontrar. Este algoritmo utiliza heurísticas, e neste caso, a heurística utilizada é o número de arestas conectadas a um determinado vértice.

Tal como no outro algoritmo, para cada grafo, a função *search* é chamada. Nesta função, é criado um dicionário onde a chave é um vértice e o seu valor é o número de arestas associadas. Em seguida, este dicionário é ordenado pelo número de arestas, e em caso de empate, ordenado alfabeticamente pelo vértice.

A solução é bastante intuitiva, uma vez que o vértice com menor número de arestas será sempre uma de todas as soluções possíveis.

Assim sendo, o vértice com menor heurística será o conjunto S, e o corte mínimo será todas as arestas que estão ligadas a esse vértice.

A complexidade computacional deste algoritmo, em relação ao algoritmo anterior, sofre uma enorme redução, sendo a complexidade O(1). Contudo, como já foi referido, apenas é encontrada uma solução para o problema.

V. RESULTADOS

Neste tópico, serão analisados os resultados para ambos os algoritmos, pelos seguintes parâmetros:

- Tempo de Execução;
- Número de Operações Básicas;
- Número de Soluções Encontradas.

Para fazer esta análise, foram gerados 45 grafos com 2 a 17 vértices.

A. Tempo de Execução

Pela análise dos gráficos abaixo, verifica-se que os tempos de execução para ambos os algoritmos são extremamente diferentes.

O tempo de execução utilizando um algoritmo de pesquisa exaustiva cresce exponencialmente à medida que o número de vértices aumenta. O tempo vai de valores ligeiramente acima de 0 até valores superiores a dois minutos.

Fig. 2 - Tempo de execução utilizando pesquisa exaustiva

Enquanto que o tempo de execução utilizando um algoritmo de pesquisa gulosa cresce aproximadamente com uma forma linear à medida que o número de vértices aumenta. Neste caso, o tempo de execução varia entre 3 e 8 microssegundo, grandezas 10^{-6} vezes mais pequenas que as grandezas do gráfico anterior.

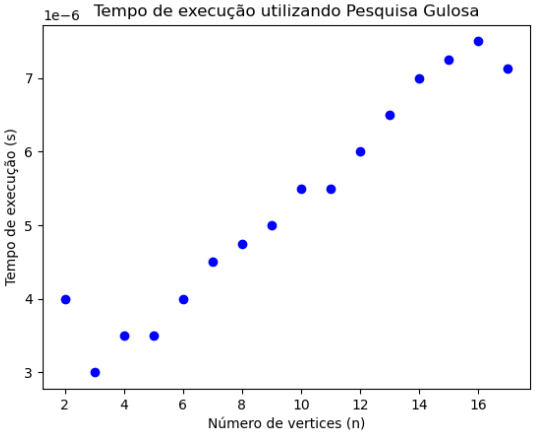

Fig. 3 - Tempo de execução utilizando pesquisa gulosa

O tempo de execução total, ou seja, o tempo que demorou a procurar a solução para todos os grafos, da pesquisa exaustiva foi 770.929 segundos, que corresponde à volta de 13 minutos. Já na pesquisa gulosa, o tempo de execução total foi 2.825 x 10^{-4} segundos.

Para problemas maiores, a estimativa de tempo de execução para a pesquisa exaustiva é muito superior a 15 minutos, dado o crescimento exponencial deste algoritmo. Já para a pesquisa gulosa, a estimativa é inferior a 1 segundo.

B. Número de Operações Básicas

Como se deve esperar, o número de operações básicas será significativamente maior na pesquisa exaustiva do que na pesquisa gulosa. Esta afirmação pode ser justificada pela complexidade computacional, uma vez que um algoritmo itera por vários subconjuntos, enquanto que o outro resolve o problema sem iterar por nenhum caso.

Os gráficos seguintes demonstram o número de operações básicas mediante o número de vértices.

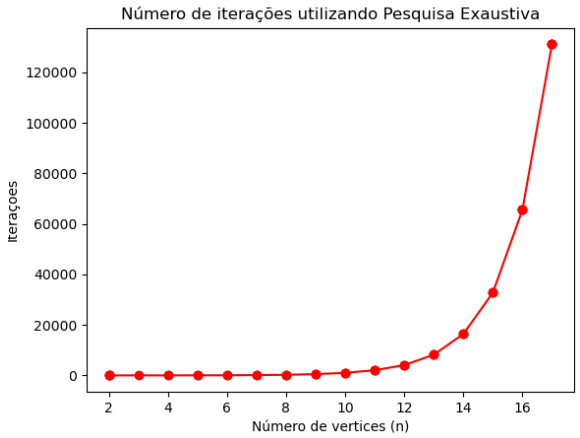

Fig. 4 - Número de operações básicas utilizando pesquisa gulosa

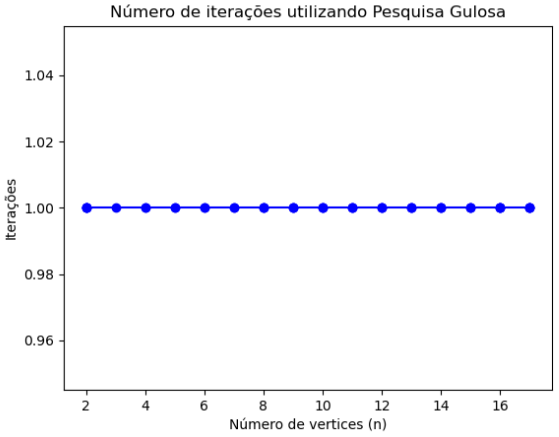

Fig. 5 - Número de operações básicas utilizando pesquisa gulosa

A forma do gráfico da Figura 4 é também é exponencial, o que indica que a partir de um grafo com 10 vértices o número de iterações começa a subir exponencialmente. Já o gráfico da Figura 5 corrobora a ideia de que o número de iterações para a pesquisa gulosa é sempre um (1).

C. Número de Soluções Encontradas

Em termos de soluções encontradas, como já foi dito antes, a pesquisa exaustiva encontra todas as soluções possíveis enquanto que a pesquisa gulosa apenas encontra uma solução. O gráfico seguinte representa graficamente o número de soluções encontradas para cada grafo.

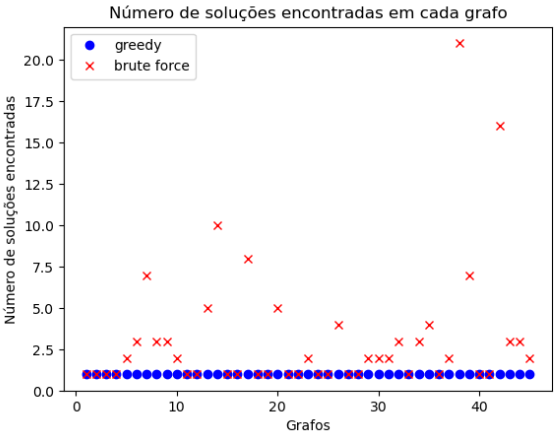

Fig. 6 - Número de soluções encontradas

VI. CONCLUSÃO

Em suma, após a análise dos dois algoritmos usados, conclui-se que, por uma lado, a pesquisa exaustiva determina todas as soluções possíveis para o problema, porém é bastante dispendiosa, uma vez que, como se verificou, o seu tempo de execução e o seu número de

operações básicas realizadas aumenta exponencialmente à medida que o número de vértices aumenta.

Por outro lado, a pesquisa gulosa é menos dispendiosa, dado que o número de operações básicas realizadas é constante e o tempo de execução é linear com o aumento do número de vértices, mas retorna apenas uma solução.

É, então, preferível utilizar outro algoritmo de pesquisa que obtenha melhores resultados do que os obtidos, por exemplo, utilizar Programação Dinâmica, estudada nas aulas teórico-práticas de Algoritmos Avançados.

VII. REFERÊNCIAS

[1] Minimum cut - Wikipedia https://en.wikipedia.org/wiki/Minimum_cut

[2] Python - Graphs https://www.tutorialspoint.com/python_data_structure/python_graphs.htm

[2] Min-cut Tutorials & Notes | Algorithms | HackerEarth https://www.hackerearth.com/practice/algorithms/graphs/min-cut/tutorial/