Algoritmos e Estruturas de Dados

3ª Série

(Terceira série de problemas)

Contar triângulos numa rede

50461 João Freitas
50457 João Lopes

Licenciatura em Engenharia Informática e de Computadores

Semestre de Verão 2022/2023

10/6/2023

1. Introdução

Este trabalho tem como objetivo a aplicação de conhecimentos dados em aula, em particular grafos.

Como problema principal é necessário fazer um programa que, usando a estrutura do grafo criada no exercício 2 da série, conte os triângulos presentes num determinado grafo sendo ele orientado ou não, sendo esse grafo recebido como input. O programa deve apresentar quantos triângulos cada vértice têm, o número total de triângulos, assim como os k vértices com mais triângulos no grafo.

O relatório irá estar dividido da seguinte forma: Problema (análise, estrutura de dados e algoritmos), Avaliação Experimental do Problema (e Análise da Complexidade).

2. Contar triângulos numa rede

Iremos, nos tópicos a seguir, mostrar a nossa análise do problema com detalhe, as estruturas de dados utilizadas no nosso programa e os algoritmos e a sua complexidade, tanto temporal como espacial.

2.1 Análise do problema

Como referido na introdução, este problema tem como objetivo a programação de uma aplicação que permita, dado um ficheiro com um grafo, isto é, com as conexões entre vértices, escrever num ficheiro quantos triângulos passam em cada vértice e escrever na linha de comandos a quantidade total de triângulos, assim como perguntar ao user para introduzir um valor k que irá colocar num ficheiro de output os k vértices com a maior quantidade de triângulos. Caso o user coloque um valor igual ou inferior a 0, o programa acaba.

Assim sendo decidimos começar por fazer uma função para os grafos não orientados. Essa função irá vértice a vértice ver as suas conexões, e a cada vértice conectado ao vértice original irá comparar os vértices conectados a si com os conectados ao original, ou seja, os vértices que estão conectados ao vértice 1 e vértice 2 formam um triângulo.

Para os grafos não orientados teria de ser diferente, visto que para os grafos orientados seria apenas necessário ver o que está conectado tanto ao vértice 1 como ao vértice 2, enquanto que nos grafos não orientados seria necessário ver se o vértice 1 está conectado aos vértice 2 e se algum vértice das adjacências do vértice 2 tem uma adjacência que será o vértice 1 outra vez, ou seja, se existe algum vértice 3 que irá conectar-se de volta ao vértice 1, formando um triângulo.

Foi necessário também fazer uma função readFile, que consoante o grafo ser não orientado ou orientado, iria ler as arestas do ficheiro onde continha as arestas do grafo e adicionar a uma variável graph. Caso fosse não orientado colocaria uma aresta de vértice 1 para vértice 2 e vice-versa, caso contrário só colocaria uma aresta do vértice 1 para vértice 2.

Por fim, para escrever no ficheiro que iria conter os k vértices que tinham a maior quantidade de triângulos, foi necessário fazer 2 funções, getVerticesWithMostTriangles e writeVerticesWithMostTriangles, que iriam obter os vértices com maior quantidade de triângulos e iriam escrever no ficheiro kCounting, respetivamente.

2.2 Estruturas de Dados

Para este problema foi usado uma classe Graph criada no exercício 2, do tipo MutableMap<I,Graph.Vertex<I,D>>, onde I é o identificador de Vértice, e onde D é o tipo de dados associados ao vértice.

Dentro da class GraphStructure tem também as inner class VertexStructure e EdgeStructure. A classe VertexStructure é a que compõe o vértice em si, atribuindo-lhe também a sua respectiva adjacência, sendo essa um mutableSet do tipo edge. Enquanto que EdgeStructure constrói as arestas do tipo edge, possuindo a identificação dos vértices que compõem as arestas, e no caso de um grafo orientado, sendo id o vértice de origem da aresta e adjacent o vértice destino.

3. Avaliação Experimental

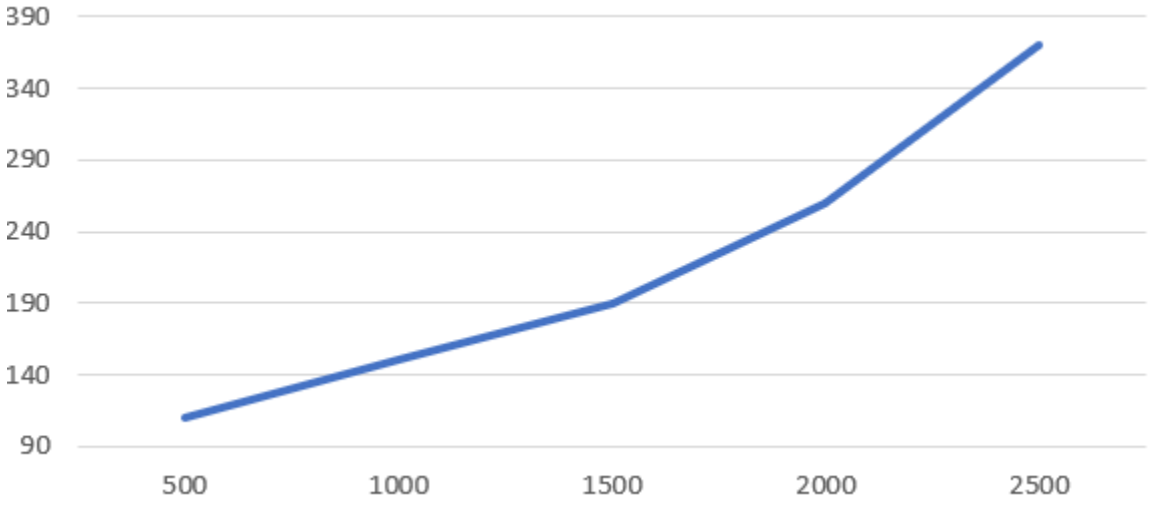
Após verificarmos a funcionalidade do programa pretendida, devemos testar as complexidades temporais e espaciais. Com este propósito, foram feitas tabelas com os resultados do tempo de execução do programa, onde se o grafo é ou não orientado e varia-se a quantidade de arestas que existem no grafo.

É de relevância notar que cada valor da tabela e gráfico na avaliação experimental foi feito através da média de tempo do teste, visto que cada teste foi realizado 10 vezes.

Ao analisar o problema é possível verificar que as complexidades irão depender dos vértices (v) e das arestas (e). A complexidade temporal e espacial dos algoritmos orientados e não orientados é a mesma, dado que apesar de serem algoritmos diferentes, com lógica também diferente, eles percorrem da mesma forma a lista de vértices, assim como as suas adjacências. Portanto a complexidade temporal é O(v*e^2), enquanto que a espacial é O(v+e), correspondendo nesse caso ao mapa de vértices e as suas respectivas adjacências.

Algoritmo não orientado:

edges	500	1000	1500	2000	2500
tempo(ms)	111,3	151,9	188,6	259	366,3



Algoritmo orientado:

edges	500	1000	1500	2000	2500
tempo(ms)	88,4	107,6	146,5	204,2	279,9