Trabalho Teoria dos Grafos

Breno Corrêa, Lucas Justino, Salomão Alves, Tarcisio Junio

Faculdade de Computação - Universidade Federal de Uberlândia (UFU)

breno.csc.2013@hotmail.com.br, {*lucasjrt*99, *salomao.alves*222}@gmail.com, tarcisiojuniocizinho@hotmail.com

Abstract: *This work aims to analyze different types of graphs, using different metrics. The graphs were taken from the Stanford Large Netowrk Dataset Collection, consisting of 3 categories: Collaboration, Highways and E-mail. The graphs were saved in .txt files, so the program should read this file and implement it. All requested metrics have been implemented.*

Resumo: *Este trabalho tem como objetivo analisar diferentes tipos de grafos, utilizando diferentes métricas. O grafos foram tirados da Stanford Large Netowrk Dataset Collection, sendo compostos por 3 diferentes categorias: Colaboração, Rodovias e E-mail. Os grafos foram salvos em arquivos .txt, por isso, o programa deve ler este arquivo e implementá-lo. Todas as métricas pedidas foram implementadas.*

1 Implementação

Utilizamos como estrutura para implementar os grafos a Lista de Adjacência, pois devido ao tamanho dos grafos, fizemos alguns cálculos e percebemos que o melhor método para usar seria esta.

Para sua implementação, fizemos 3 estruturas (structs):

1. Struct Grafo
   Esta estrutura representá o grafo como um todo. Ela possui duas variáveis do tipo inteiro, as quais irão possuir o número de vértices e de arestas. Além de possuir um ponteiro para ponteiro, que aponta para a lista principal.

2. Struct Vertices
   Esta estruta representa cada vértice do Grafo, ela que formará a lista principal da representação da Lista de Adjacência. Ela contém três variáveis do tipo inteiro, uma para identificar qual vértices ela é (seu nome), a outra é para dizer qual o seu grau e outra para dizer qual sua excentricidade. Além disso, ela possui um ponteiro que aponta para um Struct Aresta ou para o NULL.

3. Struct Aresta
   Esta estrutura irá representar as arestas do grafo. Numa Lista de Adjacência, cada vértice da lista principal (a qual contém todos os vértices) irá apontar para todos os seus vértices adjacêntes, ou seja, esse 'ato' de apontar seria as arestas. Assim, esta estrutura possui uma variável do tipo inteiro que indica qual vértice ela é e um ponteiro que 'aponta' para outra Struct Aresta ou para o NULL.

Ao lermos os grafo de um arquivo .txt, fazemos o seguinte passo a passo:

1. Criamos um grafo vazio.

2. Adicionamos todos vértices, criando assim a lista principal da Lista de Adjacência. Logo, cada vértice aponta para o NULL, uma vez que nenhuma aresta foi inserida.

3. Depois, adicionamos todas as arestas. Ao fazer isto, incrementamos todas as variáveis que dependem desta ação, como por exemplo, o grau.

Para melhor trabalharmos, nós criamos um função chamada *indice()*, na qual recebe o nome do vértice e retorna o seu índice. Logo, para evitar problemas, sempre que queremos analisar o vértice v, utilizamos seu nome, chamando esta função para retornar o índice do mesmo.

Como os grafos são muito extenso, não tivemos suporte computacional suficiente para analisá-los, utilizamos um computador Samsung, com processador Intel® Core™ i5-5200U CPU @ 2.20GHz4, com memória de 7,7 GiB. Por isso, criamos a função *porcentagem_grafo()*, a qual reduzia os grafos para a porcentagem que o usuário quisesse. Nós utilizamos 0,05% de cada grafo.

Utilizamos os seguintes grafos:

1. ca-GrQc, de tamanho de 351,5 Kb
2. ca-HepTh, de tamanho de 658,6 Kb
3. email-EuAll, com o tamanho de 5 Mb

2 Números de Vértices e Arestas

O objetivo desta métrica é simplesmente retornar o número de vértices e o número de Arestas do grafo. Para isso, fizemos duas funções, *get_numvertices()* e *get_numarestas()* que quando chamadas retornam o número de vértice e de aresta, respectivamente. Elas fazem isso simplesmente retornando as variáveis que estão no Struct Grafo.

3 Componentes Conexos

Um grafo é dito conexo se todos os pares de vértices estão ligados por um caminho. Logo, um componente conexo é um subgrafo maximal conexo de G.

Utilizamos duas funções para implementar tal métrica. A primeira, chamada *componentesConexos()*, cria um vetor de visitados do tamanho do número de vértices. Depois, há um loop, no qual chamamos a função *busca_largura()*, a qual anda por todo os vértices do grafo, marcando-os com 1 no vetor visitado. Este loop se encerra quando todas as posições do vetor visitador forem 1. E a cada interação do loop, uma variável chamada *conexo* é incrementada. No final, retornamos o valor desta variável.

4 Número Cromático

Usamos n cores para colorir o grafo de tal maneira que não tenha nenhum par de vértices adjacentes com a mesma cor. O objetivo, claro, é de minimizar esse valor n. O valor mínimo de n é chamado número cromático.

Antes de tudo, é bom falar que para este código, nós utilizamos o Algoritmo Guloso com Heurística, e as cores são representadas por números.

Para achá-lo, implementamos duas funções. A primeira, chamada *cromatico()*, cria dois vetores do tamanho do número de vértices, um para representar as cores de cada vértice, este vetor foi todo inicializado por -1, ou seja, nenhuma cor, e o outro para o vértices. Ela chama a função *bubble_sort()* para o vetor dos vértices e retornando ele ordenado, na ordem decrescente em relação ao grau de cada. Depois ela passa os dois vetores para a função *num_cromatico()*.

Esta função, por sua vez, retorna o número cromático do grafo. Para isso, ela cria dois loop. O primeiro loop só termina quando todos as posições do vetor da cores não tiverem o -1, ou seja, já foi atribuída uma cor para todos os vértices. O segundo loop é responsável por testar uma certa cor para todos as posições, ou seja, ela tenta uma mesma cor para cada vértice, ao chegar no fim dos vértices, ele tenta a próxima cor.

5 Grau Médio

O grau médio de um grafo seria a soma do grau de todos os vértices divido pelo número de vértices. Assim, nós criamos uma função que retorna o grau de um certo vétices v. Logo, criamos um loop, o qual chama a esta função para todos os vértices e vai somando-os. Ao final do loop, ela divide a somatória pelo número de vértices e retorna a média.

6 Densidade

Mede a quantidade de arestas em relação ao número de vértices. Formalmente, podemos representá-la assim: $\text{den(Grafo)} = \frac{2m}{n(n-1)}$, sendo n o número de vértices e m o número de arestas. Assim, quanto maior o número de arestas e menor o de vértice, mais denso será o grafo. Para fazer a função, simplesmente escrevemos tal fórmula em forma de código, retornando o resultado.

7 Coeficiente de Agrupamento Médio

Ele calcula a média do Coeficiente de Agrumento de todos os vértices. Que por sua vez, mede o quão complexa é a vizinhança de um certo vértice, por meio da fórmula: $\frac{2e}{k(k-1)}$, onde e representa o número de pares conectados dos vizinhos e k representa o número de vizinhos.

Para implementá-la, fizemos duas funções. A primeira chama *agrupamento()*. Para achar o k, apenas buscamos o grau do vértice. Agora para o e, criamos um vetor e nele guardamos todos os vértices adjacentes. Depois, verificamos se existe aresta entre os vértices adjacentes. Com os valores de k e de e, calculamos o Coeficiente de Agrupamento e retornando-o.

Já a outra, que chama *agrupamentoMedio()*, continha um loop, o qual pecorria todos os vértices do Grafo e ao pecorré-los, chamamos a função *agrupamento()* para cada um, somamos o resultado retornado e depois dividimos pelo número de vértices. Retornando assim o Coeficiente de Agrupamento Médio.

8 Excentricidade Efetiva Média

Excentricidade Efetiva representa o maior caminho entre o vértice v e todos os outros vértices do grafo. Para calcular a média, basta somar o maior caminho de todos os vértices e dividir pelo número de vértices.

Para implementá-la, fizemos quatro funções. A primeira, chamada *caminho()*, retorna o maior caminho entre dois vértices. Para isso, ela cria um vetor do tamanho do número de vértices e atribui o valor de -1 para cada posição, menos para posição 0, que recebce o vértice inical, e passava tanto este vetor, quanto o vértice final, para um função auxiliar chamada *caminhoRecursao()*, a qual faz uma busca em profundidade. Quando uma busca era concluída, ela guardava a maior busca numa varíavel e testava outra busca, até que todos os caminhos possíveis sejam visitados.

A terceira, chamada *excentricidade()*, primeiro, verifica-se se a excentricidade já foi calculada,com o auxílio da função *get_excentricidade()*, se ela foi, retornamos o valor, se não, ela contêm um loop, que passa por todos os vértices e chamava a função *caminho()* para cada um deles e para o vértice v, e neste loop, ela vai comparando os caminhos para descobrir o maior caminho deste vértice em relação aos outros vértices do grafo. Podemos formalizá-la assim: $e(\mathrm{v}) = \max\{d(v, w) : v \in V\}$.

Já a quarta, chamada de *excentricidadeMedia()*, também continha um loop, o qual pecorria todos os vértices do Grafo, e ao pecorré-los, chamava a função *excentricidade()* para cada um dos vértices, somamos o resultado e depois dividimo-o pelo número de vértices. Retornando o resultado.

9 Diâmetro Efetivo

Representa a maior Excentricidade de todos os vértices do Grafo. Para implementá-la, criamos a função *diametroEfetivo()*, a qual possui um loop, que passa por todos os vértices e calcula a Excentricidade de cada um, chamando a função *excentricidade()*, e comparando os resultado, retornando a maior. Podemos representá-la assim: $diam(\text{Grafo}) = \max\{e(v)|v \in V\}$.

10 Raio Efetivo

Representa a menor Excentricidade de todos os vértices do Grafo. Para implementá-la, criamos a função *raioEfetivo()*, a qual possui um loop, que passa por todos os vértices e calcula a Excentricidade de cada um, chamando a função *excentricidade()*, e comparando os resultado, retornando a menor. Podemos representá-la assim: $rad(\text{Grafo}) = \min\{e(v)|v \in V\}$.

11 Centralidade Média

A Centralidade de um vértice mede a velocidade em que uma informação chega de um vértice até outro vértice, desde que ambos sejam conectados. E a Centralidade Média é a média dessa velocidade de cada vértice.

Para implementá-la, criamos três funções. A primeira chama *caminhoMenor()*, a qual faz a mesma coisa que a função *caminho()*. Ela tabém possuia uma função auxiliar, chamada *caminhoRecursaoMenor()*, que equivale à função *caminhoRecursao()*. Ou seja, apesar de utilizar a mesma lógica, a única diferença, que em vez de retorna o maior caminho possível, ela retorna o menor.

A segunda é chamada de *centralidade()*. Ela calcula a Centralidade de um vértice, a qual se obtém através da seguinte fórmula: $C_c(v) = \frac{n-1}{\sum d(v,w):w\in\{G-v\}}$. Para obtê-la, ela possui um loop que chama a função *caminhoMenor()*, soma-os e joga na fórmula.

A terceira é a *centralidadeMedia()*, ela simplesmente chama a função *centralidade()* para cada vértice, soma tudo e divide pelo número de vértice, retornando a média.

12 Porcentagem de Vértices Centrais

Um vértice central é considerado central, quando sua excentricidade for igual ao raio efetivo do grafo, ou seja, $e(\mathrm{v}) = rad(\mathrm{Grafo})$. Assim, sua porcentagem seria a taxa de vértices centrais em relação ao número de vértices.

Para implementá-la, criamos duas funções. A primeira chama *vertCentral()*, a qual verifica se um dado vértice é ou não central, retornando 1 para TRUE e 0 para FALSE. Ela verifica chamando a função *excentricidade()* e comparando o seu resultado com o retorno da função *raioEfetivo*.

A segunda, chamada *porceVertCentral()*, contém um loop, o qual verifica se cada vértice é central, por meio da função *vertCentral()*, caso seja, soma 1, se não for, contínua o loop. Ao terminar de somar, dividimos a somatória pelo número de vértices e retornamos a porcentagem.

13 Porcentagem de Extremos

Um vértice é extremo quando seu grau é 1, $d(\mathrm{v}) = 1$. Assim, sua porcentagem seria a taxa de vértices extremos em relação ao número de vértices.

Para implementá-la, criamos a função *porcentagemExtremos()*, a qual tem um loop, que verifica o grau de cada vértice. Se o grau for igual a 1, soma 1 no somatório, caso contrário, contínua o loop. Ao terminar de somar, dividimos a somatória pelo número de vértices e retornamos a porcentagem.

14 Comparação

As três imagens a seguir mostra o resultado da análise de cada grafo.

email-euall.png

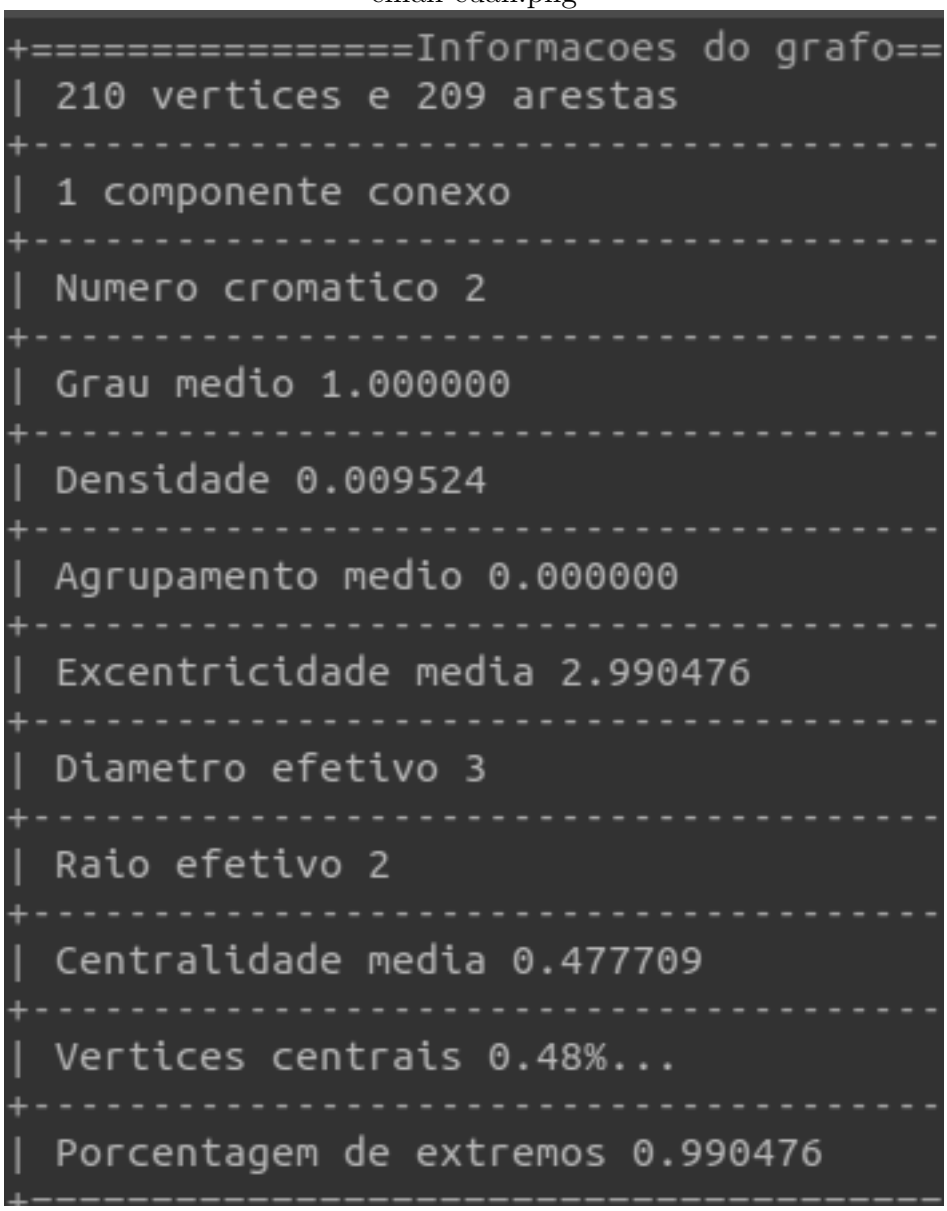

Figura 1: Grafo email-EuAll

ca-HepTh.png

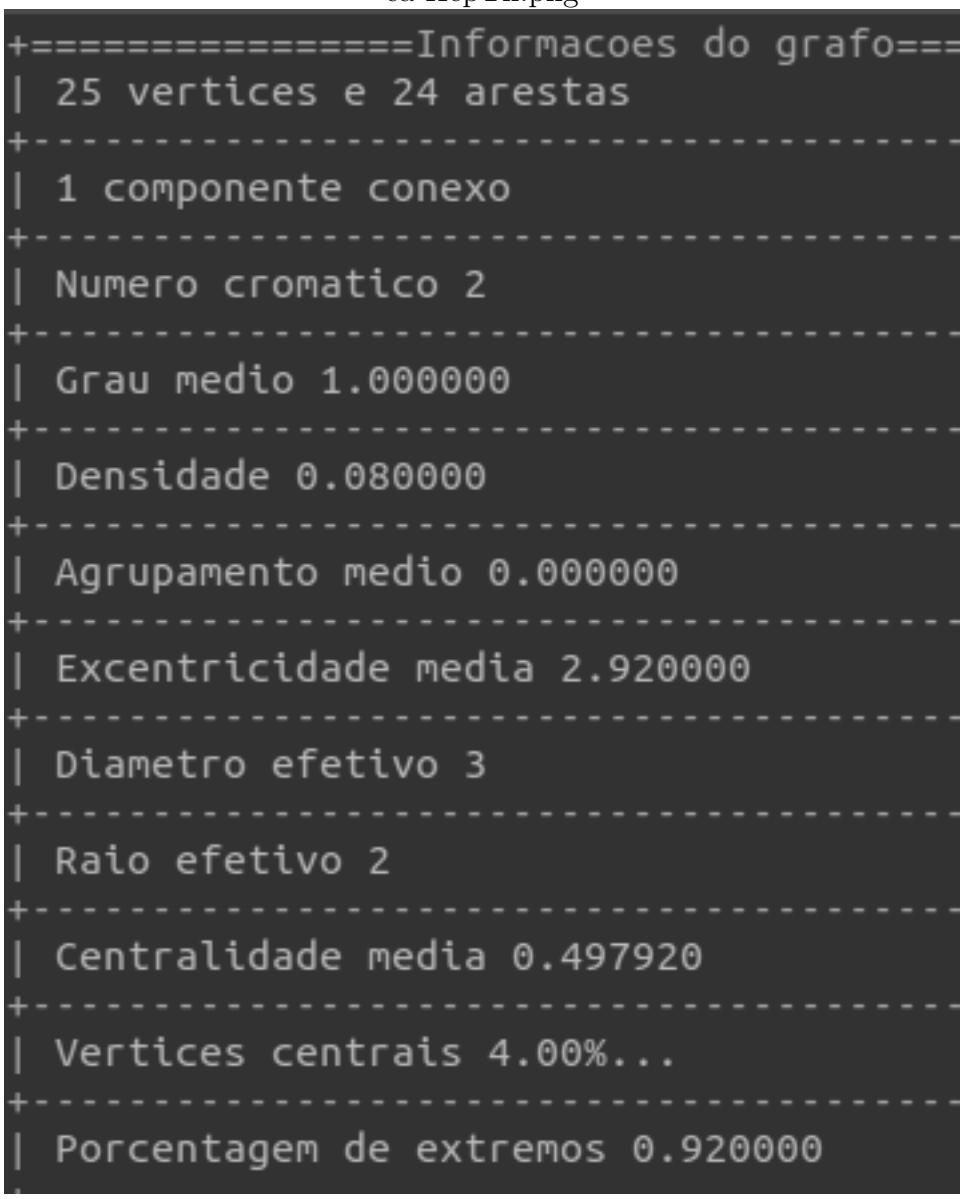

Figura 2: Grafo ca-HepTh

ca-GrQc.png

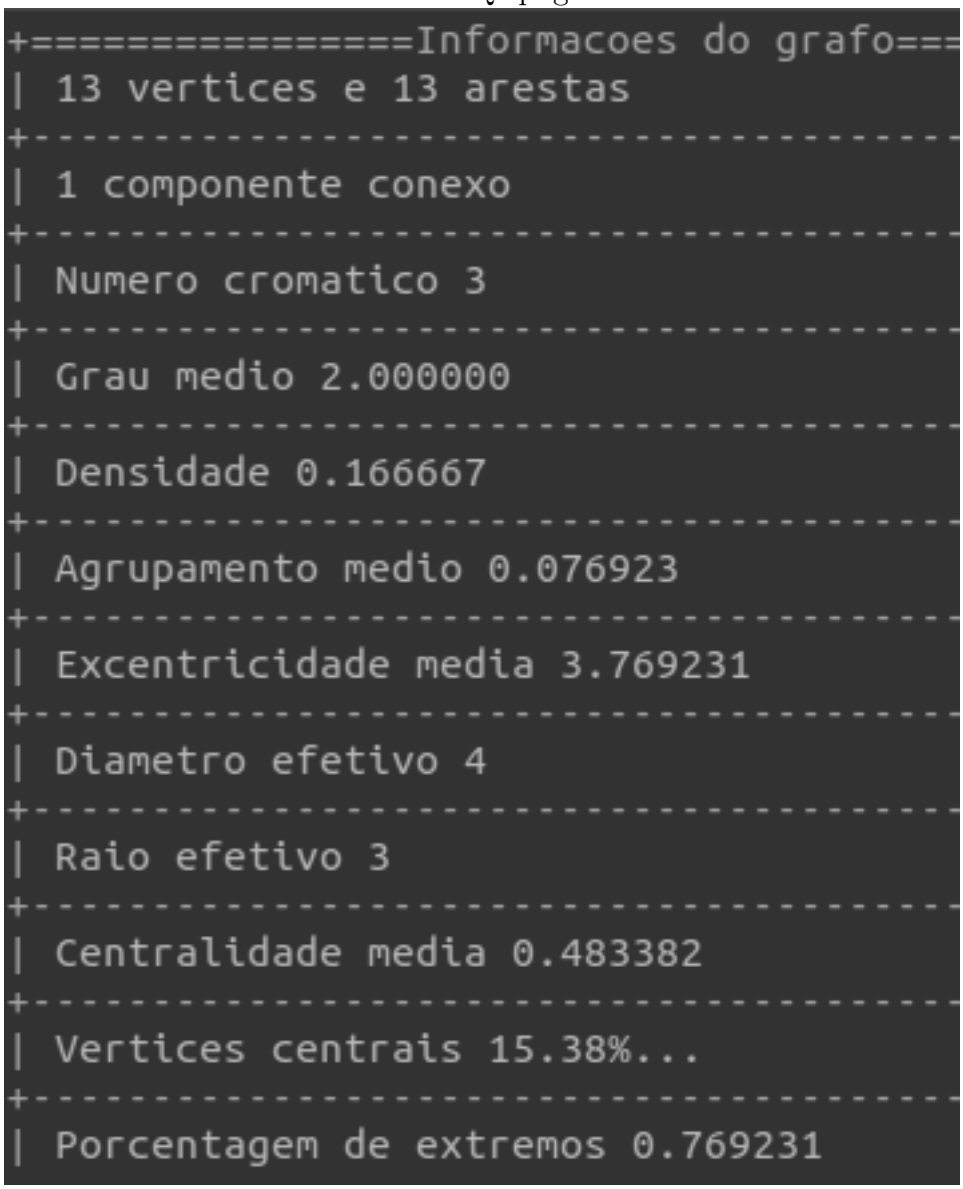

Figura 3: Grafo ca-GrQc.png