INSTITUTO DE CIÊNCIA E TECNOLOGIA
Bacharelado em Ciência e Tecnologia
Disciplina: Algoritmos e Estruturas de Dados II
Profa. Lilian Berton

Relatório

Trabalho III: **Medidas de centralidade em grafos**

Renata Sendreti Broder, 112347
dez./2017

INTRODUÇÃO

A proposta do trabalho referente ao terceiro módulo da Unidade Curricular foi a de escolher um grafo na base de dados Snap e calcular, a partir dele, as medidas de centralidade de grafos: a quantidade de graus, closeness, betweenness e clustering. Também foi proposto que se plotasse o grafo, para que fosse possível visualizar melhor o significado das medidas.

BASE DE DADOS

A base escolhida para este trabalho foi a “Wiki-Vote”, a qual contém as relações quem votou em quem - grafo direcionado - para fazer parte da administração da Wikipédia na história das eleições para administração - até Janeiro de 2008. No total foram computadas 2794 eleições com 103663 votos e 7066 usuários (votando ou sendo votado). Como a grande quantidade de vértices acaba por limitar - devido a restrições do processamento da máquina - os cálculos, foram excluídos vértices, de modo que a quantidade total analisada foi de 5001 vértices.

MEDIDAS DE CENTRALIDADE

Closeness

Esta medida tem como objetivo mostrar a proximidade de um vértice a outros vértices, assim, ela é obtida a partir da soma de todos os caminhos mínimos dividido por todos os vértices do grafo aos quais ela consegue chegar. Para calcular, inicialmente foi utilizada a busca em largura, por esta ser menos custosa computacionalmente - o algoritmo encontra-se no arquivo .c anexado - entretanto, como havia a necessidade de se calcular a medida Betweenness e de o algoritmo utilizado para tanto precisar calcular o caminho mínimo, optou-se por realizar tal cálculo apenas uma vez.

Betweenness

Este coeficiente mostra quantas vezes é necessário passar por um determinado vértice em todos os caminhos mínimos do grafo, ou seja, ele é a representação numérica da importância do vértice - é importante notar que o coeficiente não mostra literalmente o número de vezes que se passa por um vértice, mas sua importância. Por exemplo, se para sair de A e chegar em um determinado vértice E e tem-de se, obrigatoriamente, passar por D, o coeficiente de D será maior do que de B e C se, por exemplo, para se chegar em D seja possível chegar por B ou C com o mesmo custo. O algoritmo utilizado para calcular esta medida foi o de Brandes e o print da implementação se encontra no Anexo.

Graus

Esta medida é literalmente a quantidade de arestas que partem de um determinado vértice, e ela pretende expressar. Ela foi obtida através da incrementação da variável criada para isso a cada vez que uma aresta fosse inserida para um vértice.

Clustering

O clustering serve para medir a tendência de os vértices se agruparem e, para isso, foi verificado se existia triângulo de vértices e, quando existia, na verificação, o valor foi incrementado para cada vértice.

CONSIDERAÇÕES FINAIS

Por impossibilidades técnicas, não foi possível plotar o grafo para fazer a comparação com as medidas retornadas a tempo do envio do relatório, no entanto verificarei mesmo após a entrega do mesmo, uma vez que considero importante para o aprendizado e, mais do que isso, curioso. Mais uma vez, o trabalho desta disciplina supera as expectativas, uma vez que é interessante desenvolvê-lo, pois trabalha-se com dados que parecem mais palpáveis do que apenas a teoria e

implementação de algoritmos sem aplicações em dados que fazem sentido no nosso universo.

REFERÊNCIAS

BORBA, Elizandro Max. Medidas de Centralidade em grafos e aplicações em redes de dados. Disponível em <https://www.lume.ufrgs.br/bitstream/handle/10183/86094/000909891.pdf?sequence=1>. Acesso em 7 dez. 2017.

BRANDES, Ulrik. A faster algorithm for betweenness centrality. Disponível em <http://www.algo.uni-konstanz.de/publications/b-fabc-01.pdf>. Acesso em 7 dez. 2017.

DADOS

SNAP: NETWORK DATASET: Wikipedia vote network. Disponível em <https://snap.stanford.edu/data/wiki-Vote.html>. Acesso em 4 dez. 2017.

ANEXO

Implementação do algoritmo de Brandes:

```
void betweenness(tipoGrafo *grafo, int inicio){
    if(grafo->adj[inicio].prox == NULL) return;
    int i, j;

    for(i=0; i<grafo->tam; i++){
        tipoLista parents;
        inicializaLista(&parents, grafo);

        tipoPilha pilha;
        inicializaPilha(&pilha);

        float *qtdCamMin = malloc(grafo->tam*sizeof(float));
        int *dist = malloc(grafo->tam*sizeof(int));
        for(j=0; j<grafo->tam; j++){
            qtdCamMin[j] = 0.0;
            dist[j] = -1;
        }
        qtdCamMin[inicio] = 1;
        dist[inicio] = 0;

        tipoFila fila;
        inicializaFila(&fila);

        tipoVertice removido;

        insereNaFila(&fila, inicio);
        while(fila.inicio != NULL){
            removido = removeDaFila(&fila);
            insereNaPilha(&pilha, removido.vertID);
            tipoVertice *percorre = grafo->adj[removido.vertID].prox;
            while(percorre != NULL){
                if(dist[percorre->vertID] == -1){
                    insereNaFila(&fila, percorre->vertID);
                    dist[percorre->vertID] = dist[removido.vertID]+1;
                }
                if(dist[percorre->vertID] == dist[removido.vertID]+1){
                    qtdCamMin[percorre->vertID] += qtdCamMin[removido.vertID];
                    insereNaLista(&parents, percorre->vertID, removido.vertID);
                }
                percorre = percorre->prox;
            }
        }

        float total=0;
        float qtdV = 0; //vertices nos quais ele consegue chegar
        for(i=0; i<grafo->tam; i++){
            if(dist[i] != INF){
                total += dist[i];
                qtdV++;
            }
        }
        grafo->adj[inicio].somaCamMin = total;
        grafo->adj[inicio].vOndeChega = qtdV;

        float closeness = total/qtdV;
        grafo->adj[inicio].closeness = closeness;


        float *sigma = malloc(grafo->tam*sizeof(float));
        for(j=0; j<grafo->tam; j++) sigma[j] = 0.0;

        while(pilha.topo != NULL){
            int w = desempilha(&pilha);
            for(j=0; j<parents.lista[w].qtd; j++){
                int pai = retiraLista(&parents, w);
                sigma[pai] = sigma[pai] + (qtdCamMin[pai]/qtdCamMin[w])*(1.0+sigma[w]);
            }
            if(w != inicio){
                grafo->adj[w].betweenness += sigma[w];
            }
        }
        //for(j=0; j<grafo->tam; j++) printf("%.2f ", sigma[j]);
    }
}
```