UNIVERSIDADE FEDERAL DE ALFENAS

CAIO FERNANDO DIAS
FELIPE DE GODOI CORREA
MATHEUS REIS DE LIMA

TÍTULO: UTILIZAÇÃO DE ALGORITMOS EM GRAFOS PARA RESOLVER UM PROBLEMA DE CENTRALIDADE EM REDES COMPLEXAS

ALFENAS/MG
2024

CAIO FERNANDO DIAS
FELIPE DE GODOI CORREA
MATHEUS REIS DE LIMA

TÍTULO: UTILIZAÇÃO DE ALGORITMOS EM GRAFOS PARA RESOLVER UM PROBLEMA DE CENTRALIDADE EM REDES COMPLEXAS

Trabalho apresentado à disciplina Algoritmos e Estruturas de Dados 3, do curso de Ciência da Computação, da Universidade Federal de Alfenas. Área de concentração: Ciências exatas.

ALFENAS/MG
2024

1. Introdução

Este trabalho prático possui como objetivo utilizar algoritmos em grafos para resolver um problema de centralidade em redes complexas. O termo redes complexas pode ser definido como um grande grafo com estrutura topográfica não trivial, cujas conexões não podem ser definidas de uma forma simples. Os vértices destas redes são categorizados de acordo com a importância de sua posição. É possível afirmar que quanto mais central um vértice é, maior sua relevância.

O índice closeness, utilizado neste trabalho, tem como objetivo quantificar os vértices mais influentes de uma rede complexa, que podem ser empregados para propagar informações mais rapidamente, de forma mais eficiente ou barata para o restante da rede.

A distância média entre um vértice e os outros vértices de um grafo é utilizada para computar seu *closeness*. Um vértice s ∈ V, matematicamente, possui valor C_s definido como:

$$C_s = \frac{\sum_{t \in V'} \delta_{st}}{n-1}$$

Figura 1: cálculo do closeness.

em que $V' = V - \{s\}$, δ_{st} é o custo do caminho mínimo entre os vértices *s* e *t* em um grafo *G*, e *n* é o número de vértices do grafo.

2. Algoritmos

Para implementar o algoritmo foi utilizada uma lista de adjacência, que associa uma lista encadeada com cada vértice do grafo, contendo todos os vizinhos de um vértice v, portanto, representa o conjunto de arcos que derivam de v.

A primeira função calcula, a partir do algoritmo de Dijkstra, os menores caminhos a partir de um vértice de origem src, retornando um vetor que contém as distâncias mínimas de src para todos os outros vértices.

A segunda função encontra a centralidade de proximidade de um vértice específico, utilizando o algoritmo de Dijkstra para calcular a soma das menores distâncias de um vértice para todos os outros, em seguida, calcula a centralidade de proximidade como o inverso dessa soma, normalizado pelo número total de vértices menos um.

```cpp
vector<int> Grafo::dijkstra(int src) {
    vector<int> dist(MAX_VERTICES, numeric_limits<int>::max());
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;
    pq.push({0, src});
    dist[src] = 0;

    while (!pq.empty()) {
        int u = pq.top().second;
        int d_u = pq.top().first;
        pq.pop();

        if (d_u > dist[u]) continue;

        for (const auto& edge : grafo[u]) {
            int v = edge.first;
            int weight = edge.second;
            if (dist[u] + weight < dist[v]) {
                dist[v] = dist[u] + weight;
                pq.push({dist[v], v});
            }
        }
    }

    return dist;
}

double Grafo::closeness_centrality(int vertex) {
    vector<int> shortest_paths = dijkstra(vertex);
    int total_distance = 0;

    for (int dist : shortest_paths) {
        total_distance += dist;
    }

    double closeness = (MAX_VERTICES - 1) / static_cast<double>(total_distance);
    return closeness;
}
```

Figura 2: algoritmos de dijkstra e closeness.

Foi proposto também que 10 grafos conexos, ponderados, não-direcionados e de 25~75 arestas fossem utilizados para a validação dos algoritmos. Para a criação desses grafos, foi utilizado um algoritmo de geração randômica, onde a cada execução do algoritmo, novos grafos são gerados, de acordo com o código em C++ abaixo:

```
#include "grafo.hpp"
#include "grafo.cpp"

// Matriz de adjacência para armazenar as arestas existentes
bool arestasAdicionadas[MAX_VERTICES][MAX_VERTICES] = {false};

int main() {

// Fixa a seed com um valor constante
srand(12345);

// Inicializa o gerador de números aleatórios
    srand((unsigned int)time(NULL));

    Grafo grafo[10];

    for (int x = 0; x < 10; x++)
    {
        for (int i = 0; i < MAX_VERTICES; i++) {
        int qntd_arestas = (rand() % 5) + 1;
        for (int j = 0; j < qntd_arestas; j++) {
            int destino = rand() % MAX_VERTICES;
            int peso = (rand() % 5) + 1;

            // Verifica se a aresta já foi adicionada
            if (!arestasAdicionadas[i][destino] && !arestasAdicionadas[destino][i] && i != destino) {
                grafo[x].adicionarAresta(i, destino, peso);
                // Marca a aresta como adicionada na matriz de adjacência
                arestasAdicionadas[i][destino] = true;
                arestasAdicionadas[destino][i] = true; // Grafo não orientado
                }
            }
        }

        int qntd_arestas = 0;
        int peso = 0;
        int destino = 0;
        memset(arestasAdicionadas, false, sizeof(arestasAdicionadas));
    // Testando o algoritmo de Dijkstra
    vector<int> distancias = grafo[x].dijkstra(0);

    cout << "\nDistancias minimas a partir do vertice 0:\n";
    for (int i = 0; i < MAX_VERTICES; i++) {
        if (distancias[i] != numeric_limits<int>::max()) {
            cout << "Vertice " << i << ": " << distancias[i] << "\n";
        } else {
            cout << "Vertice " << i << ": Infinito\n";
        }
    }

    // Calculando e imprimindo a centralidade de closeness para cada vértice
    cout << '\n';
    for (int v = 0; v < MAX_VERTICES; ++v) {
        double closeness = grafo[x].closeness_centrality(v);
        cout << "Medida de centralidade de proximidade do vertice " << v << ": " << closeness << endl;
    }
    grafo[x].imprimirGrafoNovo(x);
    }
    return 0;
}
```

Figura 3: geração e visualização normalizada dos 10 grafos.

Foram comumente utilizados alguns algoritmos para o desenho dos grafos, por meio do Unity e a linguagem de programação C# para, por exemplo, arrastar arestas, utilizar as entradas dos grafos e fazer cálculos dos vértices.

Como não é este o objetivo do trabalho, não serão aprofundadas as extensas linhas de código para a criação do visualizador interativo de grafos. Contudo, vale ressaltar, que há em anexo um vídeo de sua utilização.

Figura 4: início do algoritmo de desenho de grafos

3. Resultados

Os resultados são compostos pela exibição da centralidade de cada um dos vértices do grafo, qual é o vértice mais central, além de uma figura do grafo mostrando os vértices, sua topologia, centralidade e os pares ordenados.

É possível verificar nos grafos, por exemplo, que o vértice mais central é colorido de azul escuro, enquanto os mais distantes possuem coloração vermelha.

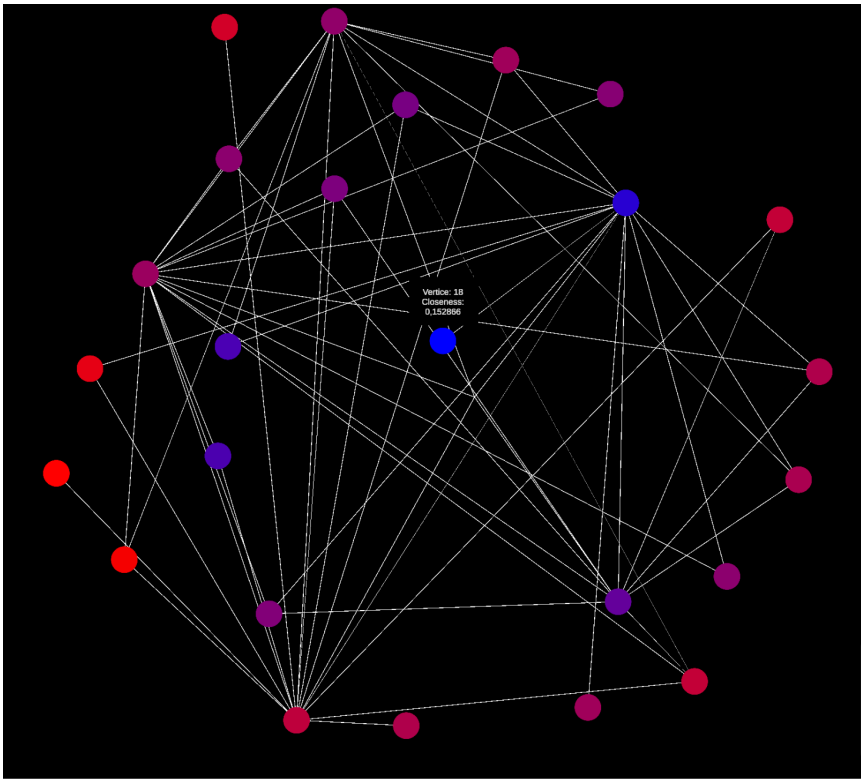

Figura 5: primeiro grafo

```
Imprimindo o grafo normalizado 8
(24, 3) (9, 1) (21, 2) (7, 3) (5, 4) (20, 4) - 0.255319
(15, 1) (11, 1) (7, 2) (13, 3) (21, 3) (12, 5) - 0.266667
(13, 3) (21, 4) (20, 1) (7, 3) (9, 5) - 0.24
(14, 5) (9, 5) (10, 3) (13, 3) (23, 5) - 0.177778
(19, 5) (20, 2) (8, 3) (7, 3) (11, 4) - 0.212389
(20, 3) (0, 4) (22, 1) (17, 1) (15, 5) (10, 3) (14, 3) - 0.244898
(15, 1) (10, 2) (17, 1) (20, 2) - 0.27907
(0, 3) (1, 2) (2, 3) (9, 5) (23, 3) (14, 1) (4, 3) (17, 4) - 0.25
(4, 3) (23, 1) (19, 1) (9, 2) (21, 5) - 0.224299
(0, 1) (2, 5) (3, 5) (7, 5) (23, 4) (13, 1) (8, 2) (18, 3) - 0.269663
(23, 5) (3, 3) (6, 2) (5, 3) (11, 1) (13, 1) (15, 1) - 0.3
(1, 1) (4, 4) (10, 1) - 0.258065
(1, 5) (23, 3) (21, 4) (19, 4) (14, 3) (15, 1) - 0.258065
(1, 3) (2, 3) (9, 1) (3, 3) (10, 1) (23, 2) (19, 3) - 0.289157
(3, 5) (7, 1) (12, 3) (24, 5) (5, 3) (20, 4) (19, 3) (23, 4) - 0.237624
(1, 1) (5, 5) (6, 1) (12, 1) (10, 1) - 0.303797
(24, 2) (20, 1) (22, 5) (17, 1) - 0.235294
(5, 1) (7, 4) (16, 1) (6, 1) (21, 4) - 0.269663
(9, 3) (23, 5) - 0.152866
(4, 5) (8, 1) (12, 4) (13, 3) (21, 1) (14, 3) - 0.228571
(2, 1) (4, 2) (5, 3) (14, 4) (16, 1) (0, 4) (6, 2) - 0.25
(0, 2) (1, 3) (2, 4) (12, 4) (17, 4) (19, 1) (8, 5) (23, 3) - 0.237624
(5, 1) (16, 5) (24, 5) - 0.198347
(7, 3) (8, 1) (9, 4) (10, 5) (12, 3) (13, 2) (21, 3) (14, 4) (18, 5) (3, 5) - 0.230769
(0, 3) (14, 5) (16, 2) (22, 5) - 0.196721
```

Figura 6: primeiro grafo normalizado

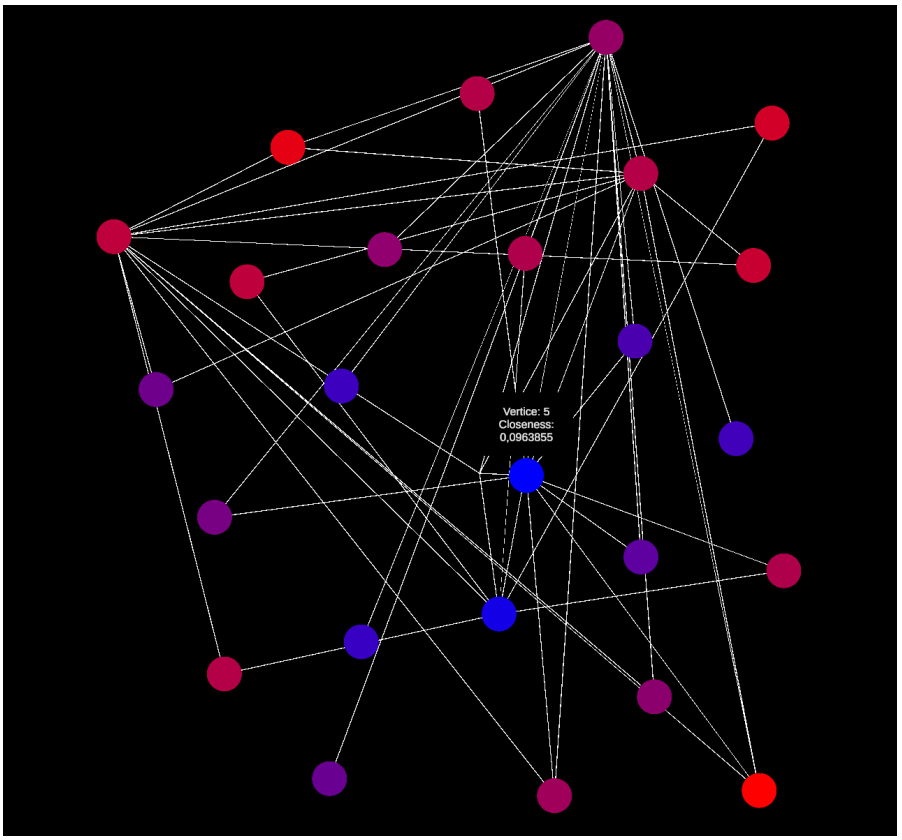

Figura 7: segundo grafo

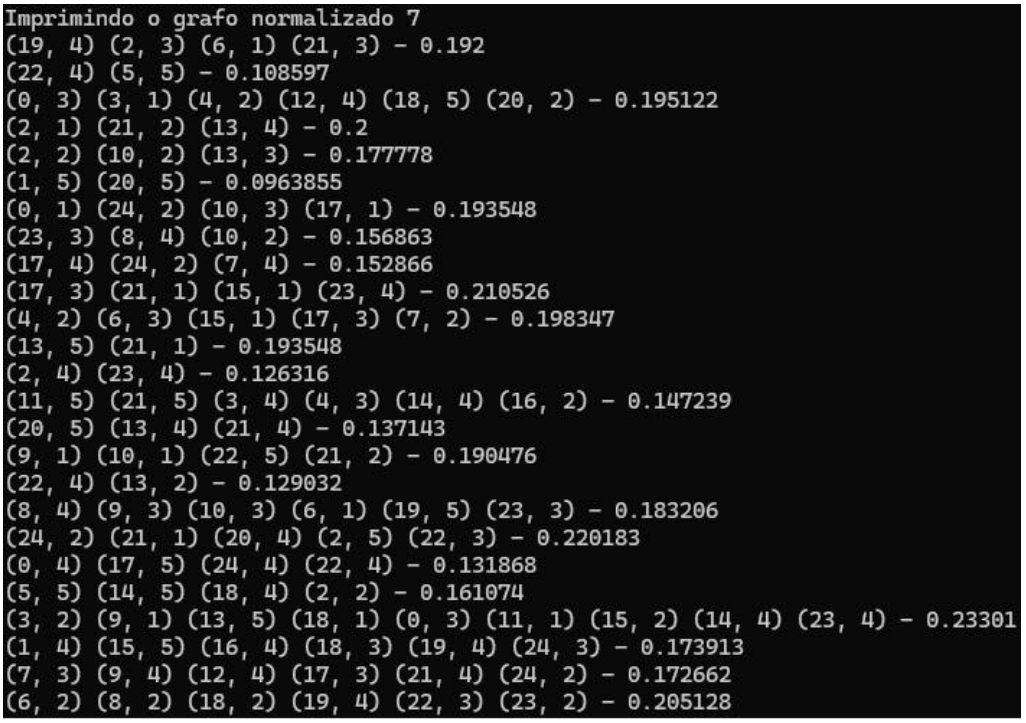

Figura 8: segundo grafo normalizado

Referências

https://repositorio.usp.br/directbitstream/30f00c12-d53f-4c46-911f-a84b360575a3/Relat%C3%B3rio%20T%C3%A9cnico_290_2007.pdf

https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/graphdatastructs.html