4ª Aula Prática - Algoritmo de Dijkstra

Disciplina: Redes de Comunicação de Dados
Alunos: Moisés Goulart de Oliveira e Victor Dallagnol Bento

O algoritmo de Dijkstra, soluciona o problema mais curto entre os vértices de um grafo. O objetivo deste é gerar uma matriz de posições com seus vizinhos e então encontra o caminho mais curto entre dois pontos (nós), e isso se aplica a todos os nós.

Na implementação feita no matlab, o caminho mais curto entre dois vértices com o maior custo no grafo obtido foi entre os pontos 16 e 41, sendo que o custo foi de 3,675 (figura 1).

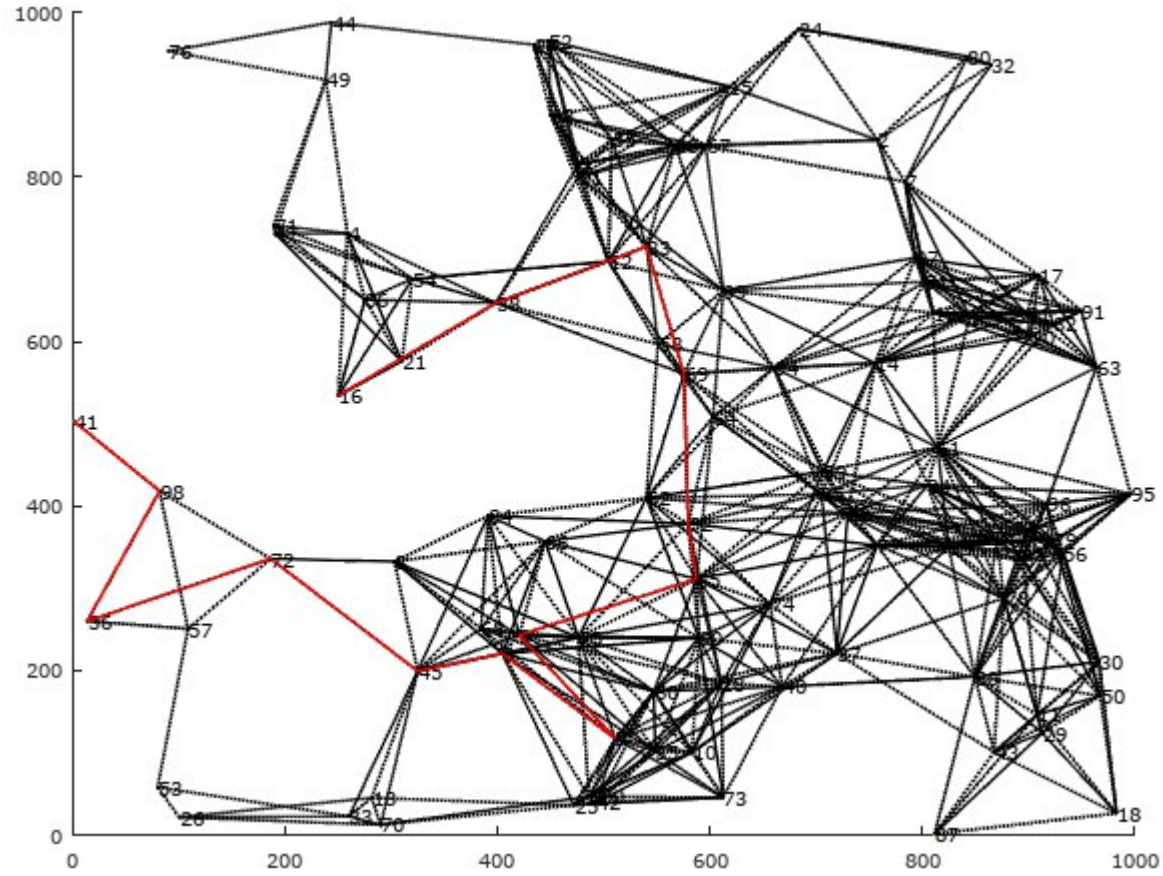

Figura 1 - Grafo obtido em simulação.

Nessa primeira etapa do algoritmo é feito o mapeamento de todos os custos para descobrir a maior distância entre os nodos. O algoritmo percorre todos nós alterando todas origens.

```
n=num_nos; % número de nós na rede
% todos os nós ainda não foram visitados;


distancia_maior(1:n) = inf;
distancia_maior_index(1:n) = inf;



for s = 1:n        % testando o algoritmo com uma origem apenas
    visitado(1:n) = 0;
    distancia(1:n) = inf;  % armazena a distância mais curta entre a origem e o nó
    precedente(1:n) = 0;   % armazena o nó precedente no caminho mais curto entre
                           % a origem e outro nó

    distancia(s) = 0;
    for k = 1:n
        temp = [];
        for h = 1:n
            if visitado(h) == 0   % ainda n?o est? no caminho mais curto;
                temp=[temp distancia(h)];
            else
                temp=[temp inf];
            end
        end
         [t, u] = min(temp);    % inicia no n? com a menor distancia at? a origem
         visitado(u) = 1;       % marca como visitado --> faz parte do caminho mais curto;
         for v = 1:n            % para cada vizinho de u;
            if ((custo(u,v) + distancia(u)) < distancia(v))
                distancia(v) = distancia(u) + custo(u,v);   % atualiza a distancia para um valor menor;
                precedente(v) = u;                          % atualiza o precedente de v;
            end
         end
    end

    % quem est? mais longe ?
    [L,d]=max(distancia);
    distancia_maior(s) = L;
    distancia_maior_index(s) = d;
end
```

Nesta segunda parte do código, a partir do nó mais distante, é feito o mapeamento para descobrir o caminho mais curto entre a origem e ele.

```
[dist index] = max(distancia_maior);
distancia_maior_index(index);


s = index        % testando o algoritmo com uma origem apenas
visitado(1:n) = 0;
distancia(1:n) = inf;  % armazena a distância mais curta entre a origem e o nó
precedente(1:n) = 0;   % armazena o nó precedente no caminho mais curto entre
                       % a origem e outro nó

distancia(s) = 0;
for k = 1:n
    temp = [];
    for h = 1:n
        if visitado(h) == 0   % ainda n?o est? no caminho mais curto;
            temp=[temp distancia(h)];
        else
            temp=[temp inf];
        end
    end
     [t, u] = min(temp);    % inicia no n? com a menor distancia at? a origem
     visitado(u) = 1;       % marca como visitado --> faz parte do caminho mais curto;
     for v = 1:n            % para cada vizinho de u;
        if ((custo(u,v) + distancia(u)) < distancia(v))
            distancia(v) = distancia(u) + custo(u,v);   % atualiza a distancia para um valor menor;
            precedente(v) = u;                          % atualiza o precedente de v;
        end
     end
end
```

Para printar o melhor caminho utilizamos o código:

```
t = d;
caminho = [d];

whilhe t ~= s
    p = precedente(t);
    caminho = [p caminho];
    t = p;
end

for k = 1:(lenght(caminho) - 1)
    line ([posX(caminho(k)) posX(caminho(k+1))], [posY(caminho(k)) posY(cami
nho(k+1))], 'Color', 'r', 'LineWidth', 2);
end
```