UNIVERSIDADE FEDERAL DE SERGIPE

Iuri Rodrigo Ferreira Alves da silva
Gregory Medeiros Melgaço Pereira
Raul Rodrigo Silva de Andrade
Rafael Castro Nunes
Ruan Robert Bispo dos Santos
Vítor do Bomfim Almeida Carvalho

Aplicação dos Algoritmos de Dijkstra e de Floyd-Warshall no software MATLAB

São Cristóvão,SE
23 de março de 2017

Iuri Rodrigo Ferreira Alves da silva
Gregory Medeiros Melgaço Pereira
Raul Rodrigo Silva de Andrade
Rafael Castro Nunes
Ruan Robert Bispo dos Santos
Vítor do Bomfim Almeida Carvalho

Aplicação dos Algoritmos de Dijkstra e de Floyd-Warshall no software MATLAB

Relatório em conformidade com as normas ABNT

Universidade Federal De Sergipe
Faculdade de Engenharia Eletrônica
Redes e Comunicações

São Cristóvão,SE
23 de março de 2017

Resumo

A busca por uma maior economia e menores custos nas diversas áreas das ações humanas foi um grande ponto de partida para o surgimento da Teoria dos Grafos com o consequente Problema dos Menores Caminhos. Para a resolução desse Problema dos Menores Caminhos surgiram vários tipos de algoritmos, como por exemplo o Algoritmo de Dijkstra e Algoritmo de Floyd-Warshall.

Nesse trabalho será feito uma explicação sobre o que são os grafos e sobre os algoritmos utilizados nesse projeto. Após isso, esses algoritmos serão implementados no software computacional MATLAB para ser possível a comparação entre as características obtidas dos dois algoritmos. Com essa comparação será possível dizer qual dos algoritmos é mais eficiente com relação ao número de iterações e tempo de processamento e sob quais condições isso acontece.

Palavras-chaves: Grafos, Problema dos Menores Caminhos, Floyd-Warshall, Dijkstra, Iterações

Lista de ilustrações

Lista de tabelas

Sumário

Introdução

No mundo globalizado e capitalista que vivemos é cada vez mais necessário caminhos ou métodos para se decidir qual a melhor alternativa ou caminho a ser usado visando uma maior economia, , seja ela na produção do meio industrial, nas conversões de moedas ou até melhores caminhos a serem tomados numa viagem.

Foi pensando nesses tipos de problemas que surgiu a Teoria dos Grafos em 1736 com Leonhard Euler (1707 1783). Um grafo é um conjunto de vértices e um conjunto de arestas que ligam pares de vértices distintos (com nunca mais que uma aresta a ligar qualquer par de vértices).

Leonhard resolveu um quebra-cabeça local da cidade Königsberg onde vivia, hoje conhecido como O Problema das Pontes de Königsberg. Esse problema consistia em partindo de um ponto inicial, voltar a esse mesmo ponto passando exatamente uma vez por cada ponte da cidade. Euler mostrou que isso só seria possível se cada porção de terra estivesse ligada a um número par de pontes, e , como isso não acontecia na sua cidade, não era possível.

Outro exemplo nesse mesmo sentido é o mostrado em (COSTALONGA, 2012), em que um carteiro sempre tentará caminhar a menor distancia possível, assim, para isso acontecer, ele tem de agir de tal forma a passar um número mínimo de vezes em determinada rua, otimizando assim seu trabalho.

Ao longo da história outros problemas similares ao de Euler surgiram, como por exemplo O Problema das Quatro Cores . Esse problema surgiu em 1852 e somente em 1976 utilizando métodos computacionais chegou-se a sua solução.

A Teoria dos Grafos ganhou muita visibilidade já no século passado, pois como dito em (COSTA, 2011) no desenvolvimento matemático assim como nas suas aplicações, foi dada grande importância a essa teoria. Com o passar do tempo grafos foram utilizados em várias áreas do conhecimento, desde a Economia até a Biologia.

Com esse grande destaque, se tornou necessária a criação de algoritmos que agilizassem suas soluções. Entre os mais conhecidos estão o de Dijkstra, de Bellman-Ford e o de Floyd-Warshall.

1 Revisão Bibliográfica

1.1 Dijkstra

Em (CARVALHO, 2008) é apresentado o algoritmo de dijkstra (E.W. Dijkstra), sendo este utilizado para obter o caminho de custo mínimo entre os vértices de um grafo. Escolhido um vértice como origem, este algoritmo calcula a distância mínima para todos os demais vértices do grafo.

Ainda em (CARVALHO, 2008) é resolvido um pequeno exemplo com o algoritmo de dijkistra que envolve distância entre cidades, mostrando o menor caminho entre elas. Esse exemplo foi feito passo a passo, explicando minunciosamente como funciona esse algoritmo. Também ressalta que este só pode ser utilizado em grafos ponderados e unicamente com pesos positivos, calculando a distância entre uma cidade e todas as outras, diferentemente do Algoritmo de Floyd que calcula a distância entre todas as cidades.

Em (BARROS; PAMBOUKIAN; ZAMBONI, 2007) há a apresentação do Algoritmo de dijkistra e a explicação do algoritmo passo a passo, feita de forma diferente do (CARVALHO, 2008) pois este é feito de forma mais mecânica, com um exemplo mecânico. Apenas com uma tabela e como o algoritmo funciona e seus passos.

Em (BARROS; PAMBOUKIAN; ZAMBONI, 2007) também é dita algumas aplicações, indo de uma cadeia de produção, até o clássico problema do carteiro que não pode passar duas vezes na mesma rua. Qualquer grafo simples que possua a matriz de pesos definida pode ser submetida à proposta de dijkistra.

1.2 Floyd-Warshall

Em (HOUGARDY, 2010) é apresentado o algorítimo de Floyd-Warshall, sendo este utilizado para obter o menor caminho de um vértice de origem até cada um dos outros vértices de G, onde G é um grafo direcionado com arestas ponderadas. O artigo ressalta que apesar de possuir um tempo de processamento elevado, superior a muitos outros algoritmos, o algoritmo de Floyd-Warshall possui uma estrutura de complexidade muito menor.

Ainda em (**??**), é apontado que o algoritmo funciona normalmente se não existir ciclos negativos no grafo analisado, entretando será detectada a existência destes ciclos.

Algumas diferenças são vistas entre Dijkstra e Floyd-Warshall em (BARROS; PAMBOUKIAN; ZAMBONI, 2007) e (**??**), enquanto o primeiro pode ser usado apenas

para grafos com pesos positivos, o segundo pode ser usado em grafos com pesos negativos. Pode-se observar também que o algoritmo de dijkstra leva um tempo de processamento muito menor por não computar todos os caminhos possíveis, diferentemente do de Floyd.

2 Objetivos

O objetivo do trabalho foi o estudo da teoria dos grafos na resolução do problema de menor caminho. Para tal, foram implemantados os algoritmos de Dijkstra e de Floyd-Warshall com objetivo de comparar o desempenho dos mesmos na resolução do problema.

3 Fundamentação Teórica

3.1 Grafos

O conceito de grafo é um conceito simples, porém muito amplo, que trata da relação de conexão entre elementos nos mais diversos problemas matemáticos. Podemos então definir grafo como a união de um conjunto de vértices e um conjunto de arestas. Geralmente os grafos tem a seguinte notação:

$$G = (V, A) \tag{3.1}$$

Onde V representa o conjunto de vértices e A representa o conjunto de arestas.

3.1.1 Vértices e Arestas

Os vértices, também chamados de nós, são os elementos de um conjunto. O número de elementos deste conjunto representa a ordem da estrutura. O conjunto de arestas representa as ligações entre os elementos do conjunto de vértices.

A depender do problema a ser estudado, estes elementos assumem significados diferentes. Os vértices podem significar pessoas, localizações geográficas, hosts, entre outros. Já as arestas representam então as relações entre esses elementos, desde distâncias físicas entre cidades, até relações de amizade ou afetividade entre pessoas.

3.1.2 Antecessor, Sucessor e Vizinho

Vértices vizinhos são aqueles que são ligados por uma aresta ou arco. Quando este vértice está na extremidade inicial da aresta ele é chamado de antecessor, quando está na extremidade final, é chamado de vértice sucessor.

3.1.3 Grafos Orientados ou Não Orientados

Em um grafo não orientado, a ligação entre dois vértices é feita através de uma linha, chamada de aresta. Já em um grafo orientado, a ligação é feita através de uma seta, chamada de arco, e esta representa o sentido correspondente desta ligação.

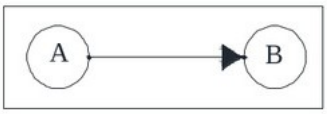

Figura 1 – Grafo Orientado

Figura 2 – Grafo Não Orientado

3.2 Representação de um grafo

Para melhor visualização de um grafo é usada a representação como nas figuras 1 e 2, com a identificação dos vértices entre círculos e as arestas ou arcos como linhas, porém para fins de cálculo os grafos são associados a matrizes. As matriz mais comumente utilizada é a matriz de adjacência, apesar de não ser a mais econômica computacionalmente.

3.2.1 Matriz de Adjacência

A matriz de adjacência é uma matriz booleana, quando utilizada em grafos, de tamanho n x n onde n é o número de vértices do grafo. Chamando de $A = a_{ij}$ a matriz adjacência, onde i e j representam o número de identificação do vértice, por exemplo, para a_{21} teremos nesta posição o valor correspondente a aresta que liga o vértice dois ao vértice um. A matriz é preenchida pela seguinte condição:

$a_{ij} = 1$, se existe ligação entre os vértices i e j

$a_{ij} = 0$, se não existe ligação entre os vértices i e j

Exemplo:

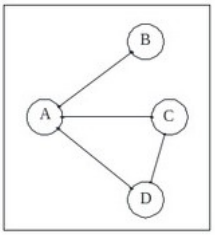

Figura 3 – Grafo Sem Peso

	A	B	C	D
A	0	1	1	1
B	1	0	0	0
C	1	0	0	1
D	1	0	1	0

Tabela 1 – Matriz Adjacência sem Peso

Caso as arestas contenham peso, a matriz de adjacência deve ser preenchida com o peso correspondente e onde não houver conexão entre os vértices, preencher com um valor que não possa ser considerado peso, como zero ou infinito.

Exemplo:

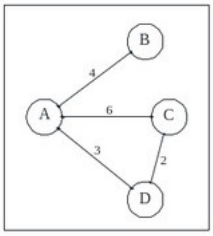

Figura 4 – Grafo Com Peso

	A	B	C	D
A	0	4	6	3
B	4	0	0	0
C	6	0	0	2
D	3	0	2	0

Tabela 2 – Matriz Adjacência com Peso

Uma observação importante é que, em grafos não direcionados, a matriz de adjacência é simétrica em relação a diagonal principal.

3.3 Aplicação para o Problema do Menor Caminho

Uma das aplicações mais comuns de grafos é para a resolução do problema de menor caminho, definindo um vértice de partida e um de chegada em um grafo, determinar qual caminho apresenta o menor custo para percorrer este caminho, considerando os pesos contidos nas arestas. Devido a importância da resolução deste tipo de problema, foram desenvolvidos algoritmos capazes de retornar como resultado o menor caminho dentro de um grafo.

3.3.1 Algoritmo de Dijkstra

Para utilizarmos o algoritmo de Dijkstra o primeiro passo é determinar o vértice de origem e o vértice de destino. Determinados a origem e o destino é feita uma análise através de interações vértice a vértice, buscando o menor, ou menores caminhos possíveis.

Partindo da origem, compara-se o custo de cada conexão ligada ao vértice do momento, para a primeira interação este vértice será a origem. Segue-se então para o próximo vértice, que apresentar o menor custo e o vértice antecessor é marcado. No novo vértice do momento é somado os custos para a chegada até ele e busca-se as conexões do vértice do momento com vértices não marcados, seguindo para o que apresente menor custo. Essa interação se repete até chegar no vértice de destino.

É importante salientar que este algoritmo não aceita custos (pesos) negativos.

3.3.2 Algoritmo de Floyd- Warshall

Este algoritmo é matricial e aceita custos negativos, mas a presença de ciclos negativos, ou seja, presença de mais de um valor negativo em uma malha de arestas, exige cuidado e conferência, já que o resultado pode não condizer com o real.

O algoritmo calcula todos os menores custos de todas as origens para todos os destinos, através de um número de interações finito e igual ao número de vértices do grafo. O algoritmo é inicializado com uma matriz onde os elementos são apenas as distâncias diretas entre os vértices, a diagonal principal, ou seja a distância entre os vértices e eles mesmos é zero e os vértices que não tem ligação direta, o elemento da matriz será infinito.

Após a montagem da matriz inicial, cada interação busca as menores distâncias entre os pontos, passando pelo vértice de número igual ao número da interação. Por exemplo, a terceira interação busca as menores distâncias entre os vértices, passando pelo vértice de número 3. Em cada interação, caso a distância seja menor do que a que já obtemos na matriz, o valor menor substitui o atual. Ao final das interações temos todas as menores distâncias entre os vértices.

4 Formulação do problema

Os algoritmos foram implementados no software MATLAB. Seguem abaixo os códigos comentados :

Grafos:

```
clc
clear
hold off

% Define ligações
s = [1 1 1 2 2 3 3 4 5 5 6 7];
t = [2 4 8 3 7 4 6 5 6 8 7 8];

% q = [1 1 3 3 5 5 7 7 9 9 11 11 13
 13 15 15 17 17 19 19 2 6 10
14 18 4 8 12 16 20];

% e = [2 3 4 5 6 7 8 9 10 11 12 13
 14 15 16 17 18 19 20 1 6 10 14
18 2 8 12 16 20 4];

% Pesos
weights = [10 10 1 10 1 10 1 1 12 12 12 12];
% weights = [10 10 1 10 1 10 1 1
12 12 12 12 9 2 5 7 12 14 9 1 3 2
24 12 14 10 12 1 2 3];

% weights = [10 10 1 10 10 10 1 10 1 10 1 1
 12 12 12 12 9 2 5 7 12 14 9 1 3 2 24 12 14
  10 12 1 2 3 1 10 1 1 10 10 1 10 1 10 1 1
  12 12 12 12 9 2 5 7 12 14 9 1 3 2 24 12
  14 10 12 1 2 3 12 12 12 12 9 2 5 7 12 14
  9 1 3 2 24 12 14 10 12 1 2 3];

% Nomes
names = {'A' 'B' 'C' 'D' 'E' 'F' 'G' 'H'};

% Cria Grafo
G = graph(s,t,weights,names);
% G = graph(q,e,weights);
% G = graph(bucky);

% Plots
p = plot(G,'EdgeLabel',G.Edges.Weight);

% Menor distancia
tic
[caminho,distancia] = dijkstra(G,2,5);
Td = toc;
tic
[caminho2,distancia2] = floyd(G,2,5);
Tf = toc;
```

```
% Plot fresco
pause(1);
duo = caminho(1);
for i=1:(size(caminho,2)-1)
duo = [duo caminho(i+1)];
highlight(p, duo,'EdgeColor','r')
pause(1);
end

figure(2)
c = categorical({'Floyd','Dijkstra'});
v = [Tf Td];
bar(c,v)
```

Dijkstra:

```
function [Caminho, Distancia] = dijkstra(G, origem, destino)

% Entrada: Grafo G
%          Origem da busca
%          Destino da busca
% Saída  : Um menor Caminho
%          A Distancia mínima

Caminho = [destino]; % inicializa o vetor caminho
Distancia = []; % inicializa o vetor distancia

% criando a matriz dados
dist = (inf*(1:size(G.Nodes,1))'); %distancias infinito
preced = (0*(1:size(G.Nodes,1))'); % coluna que indica de onde veio
dados = [(1:size(G.Nodes,1))' dist preced]; %nó, distancia,origem

dados(origem,2) = 0; % distancia da origem a origem é zero
P_no = origem; % proximo no recebe origem (inicializacao)
memo = []; % memoria de Nós percorridos

while (1)

atual = P_no; % Nó atual
memo = [memo; dados(atual,:)]; % por onde ja passei

vizinhos = neighbors(G,atual); % acha Nós vizinhos

for i=1:size(vizinhos,1) % atribuir distancias
if (dados(vizinhos(i),2) > dados(atual,2)+distances(G,atual,vizinhos(i)));
dados(vizinhos(i),2) = dados(atual,2)+distances(G,atual,vizinhos(i));
dados(vizinhos(i),3) = dados(atual,1);
end
end

% Define o proximo Nó da busca
vazio = dados; % cria uma matriz que pode ser alterada
for l=1:size(memo,1)
ind = find(vazio == memo(l,1)); % encontra elementos ja visitados
vazio(ind(1),:) = []; % elimina elementos passados
end
[x,y] = min(vazio,[],1);
%-------------------------%
```

```
% criterio de parada
if (isempty(vazio))
break
else
P_no = vazio(y(2),1); % escolhe proximo nó
end
end

% Define menor caminho
g = destino;
while (g ~= origem)
o = find(memo == g);
g = memo(o(1),3);
Caminho = [Caminho g];
end
Caminho = flip(Caminho);

% Define a menor Distancia
o = find(memo == destino);
Distancia = memo(o(1),2);


end
```

Floyd-Warshall:

```
function [Caminho, Distancia] = floyd(G, origem, destino)

% Entrada: Grafo G
%          Origem da busca
%          Destino da busca
% Saída  : Um menor Caminho
%          A Distancia mínima

Distancia = []; % inicializa o vetor distancia
Caminho = destino; % inicializa o vetor caminho

m = [0   10   Inf   10   Inf   Inf   Inf    1
10    0   10   Inf   Inf   Inf    1   Inf
Inf   10    0   10   Inf    1   Inf   Inf
10   Inf   10    0    1   Inf   Inf   Inf
Inf   Inf   Inf    1    0    12   Inf    12
Inf   Inf    1   Inf   12    0    12   Inf
Inf    1   Inf   Inf   Inf   12    0    12
1   Inf   Inf   Inf   12   Inf   12    0];
dis = m;

cam = [ 1 2 3 4 5 6 7 8;
1 2 3 4 5 6 7 8;
1 2 3 4 5 6 7 8;
1 2 3 4 5 6 7 8;
1 2 3 4 5 6 7 8;
1 2 3 4 5 6 7 8;
1 2 3 4 5 6 7 8];

for k=1:size(m,1)
for i=1:size(m,1)
for j=1:size(m,1)
```

```
        if (dis(i,k)+ dis(k,j) < dis(i,j))
        cam(i,j)=k;
        dis(i,j) = dis(i,k)+ dis(k,j);
        end
        end
        end
        end

        Distancia = dis(origem,destino);

        while(Caminho(size(Caminho,1)) ~= origem)
        Caminho = [Caminho; cam(destino,origem)];
        if Caminho(size(Caminho,1)) ~= origem
        destino = Caminho(size(Caminho,1));
        end
        end

        Caminho = flip(Caminho');

        end
```

5 Resultados Obtidos

Como resultados, os dois códigos feitos no MATLAB funcionaram perfeitamente. Eles foram capazes de dizer a melhor rota como também o custo dessa rota. Já na comparação entre os dois desepenhos o algoritmo de Floyd-Warshall possuiu um tempo de processamento muito menor do que o tempo de processamento do Dijkstra. Essa diferença nos tempos pode ser visualizada no gráfico de barras abaixo, onde a coluna da esquerda(verde) apresenta o tempo do Dijkstra e o da direita(azul) é o tempo do Floyd-Warshall :

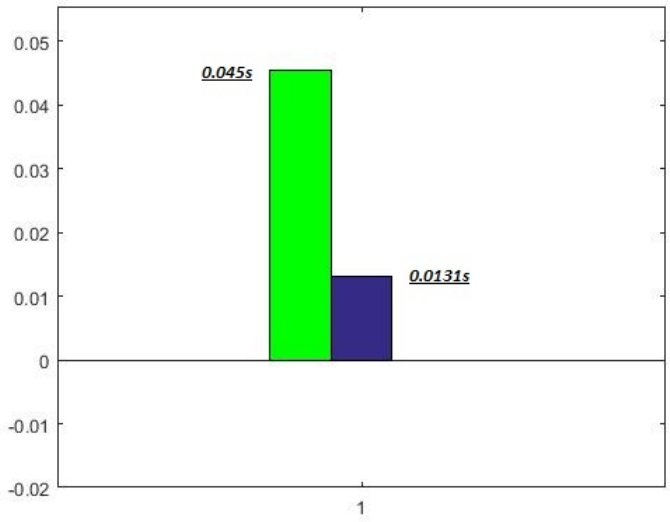

Figura 5 – Gráfico de barras Dijkstra e Floyd

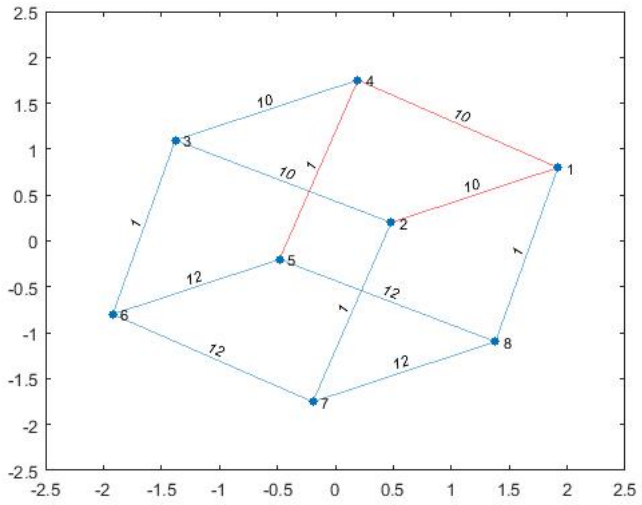

Figura 6 – Grafo com caminho obtido

Isso já era esperado pois no algoritmo de Dijkstra o número de iterações é muito superior ao do Floyd-Warhsall já que a cada vértice ele precisa analisar o caminho a ser tomado. No algoritmo de Floyd-Warshall é menor, onde número de iterações é igual ao número de vértices do grafo.

A figura 7 apresenta novamente uma comparação de desempenho para o grafo da figura 8, mostrando o mesmo resultado superior do floyd sobre o Dijkstra.

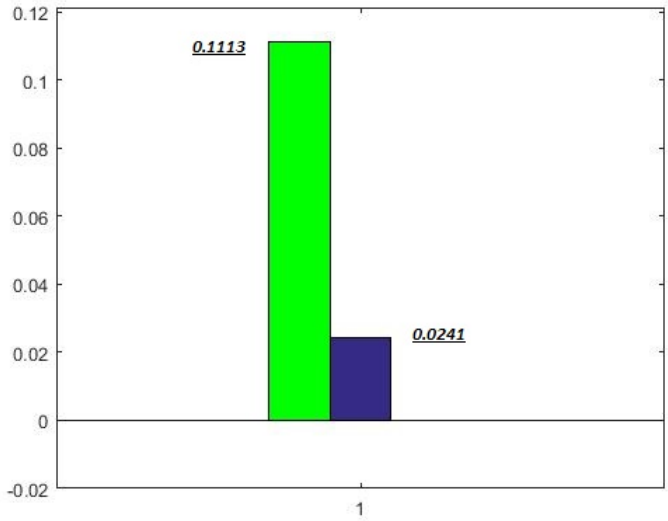

Figura 7 – Gráfico de barras Dijkstra e Floyd

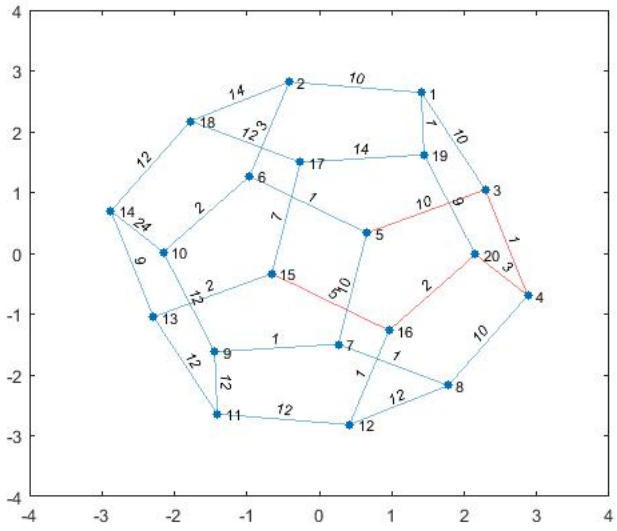

Figura 8 – Grafo com caminho obtido

Foi notado que a partir de 700 vértices, o algorítimo de Dijkstra apresentou um desempenho superior ao de Floyd-Warshall.

6 Conclusão

A partir dos resultados obtidos, é possível concluir a maior eficiência do algorítimo de Floyd-Warshall apenas para um menor numero de iterações. No momento em que um número de vértices mais elevados são calculados, o algorítimo de Dijkstra possui desempenho mais elevado.

Referências

BARROS, E. A.; PAMBOUKIAN, S. V.; ZAMBONI, L. C. Algoritmo de dijkstra: apoio didático e multidisciplinar na implementação, simulação e utilização computacional. In: *INTERNATIONAL CONFERENCE ON ENGINEERING AND COMPUTER EDUCATION, São Paulo*. [S.l.: s.n.], 2007. Citado na página 7.

CARVALHO, B. M. P. S. de. Algoritmo de dijkstra. *Universidade de Coimbra, Coimbra, Portugal*, 2008. Citado na página 7.

COSTA, P. P. da. *Teoria de Grafos e suas Aplicações*. Tese (Doutorado) — Instituto de Geociências, 2011. Citado na página 6.

COSTALONGA, J. P. Grafos e aplicações. *Notas do Minicurso da 23ª Semana da Matemática do DMA-UEM, Maringá*, 2012. Citado na página 6.

HOUGARDY, S. The floyd–warshall algorithm on graphs with negative cycles. *Information Processing Letters*, Elsevier, v. 110, n. 8-9, p. 279–281, 2010. Citado na página 7.