Trabalho Computacional de Matemática Discreta 2

Vitor Saraiva de Lima[a], Mateus Lima Pinho[a]

[a] *Curso de Engenharia de Computação,*
Instituto Politécnico(IPRJ),
Rio de Janeiro State University,
Rua Bonfim 25, Nova Friburgo, RJ 28625-570, Brazil

Abstract

In this work, a program was developed in the SciLab programming language that represents a graph given an adjacency matrix and the coordinates of the points of the same graph, in addition to demonstrate the dijkstra and search algorithms and the programming of a graphical interface in the last part.

Resumo

Neste trabalho, foi desenvolvido um programa na linguagem de programação SciLab que representa um grafo dada uma matriz adjacência e as coordenadas dos pontos do mesmo grafo, além de demonstrar os algorítmos de dijkstra e de busca e a programação de uma interface gráfica na última parte.
Keywords: graph theory, matrix adjacency, graph algorithms, scilab, graphical interface

Palavras-chave: teoria dos grafos, matriz adjacência, algorítmos de grafos, scilab, interface gráfica

Email addresses: vitorsaraivadelima@gmail.com (Vitor Saraiva de Lima), mateuslpinho@gmail.com (Mateus Lima Pinho)

1. Introdução

O trabalho pode ser dividido nas seguintes partes:

1. Construção de um código-fonte em Scilab implementando os algoritmos de dijkstra e de busca profundidade ou de nível;
2. Construção de um código-fonte em Scilab que dada uma matriz de um grafo produza a matriz adjacência do mesmo, o vetro de arestas e uma matriz B relacionada a representação dada como exemplo na Fig. 1. Também fazer o processo de volta da matriz B para a matriz adjacência A;
3. Fornecer a matriz de adjacência de um mapa do Brasil, bem como vetores x e y contendo posições de pontos representativos dos estados(não envolve programação);
4. Construção de um código-fonte em Scilab em que dada uma matriz de adjacência, A, e vetores posição de nós, representar na tela do computador o grafo resultante através de uma imagem;
5. Construção de um código-fonte em Scilab que da a opção escolha de mapas (Brasil, Inglaterra, Mercosul) em um menu e cria uma figura dividida em 4 zonas:
   (a) o mapa da região escolhida;
   (b) o grafo da região com os nós numerados;
   (c) o grafo rotulado com os nomes das regiões representadas pelos nós;
   (d) o grafo com os nós numerados sobreposto ao mapa da região.

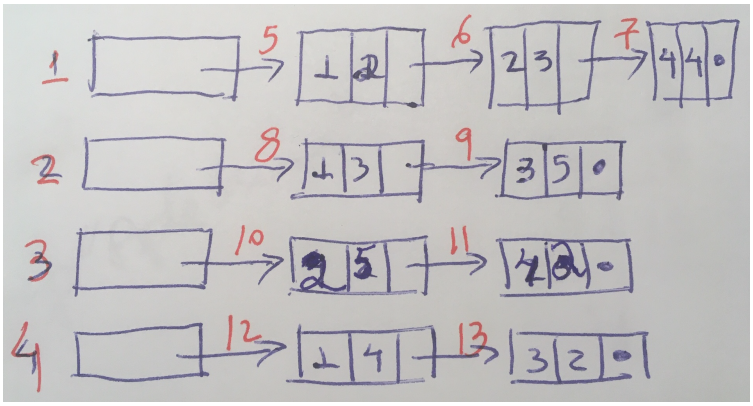

Figura 1: Exemplo de representação para a matriz B

2. Algorítmos de dijkstra e de busca

Nesta seção será abordado o assunto do item 1 da seção 1.

2.1. Algorítmo de dijkstra

O algorítmo de dijkstra soluciona o problema do caminho mais curto num grafo dirigido ou não dirigido com arestas de peso não negativo, ele considera um conjunto S de menores caminhos, iniciado com um vértice inicial I e a cada passo do algoritmo busca-se nas adjacências dos vértices pertencentes a S aquele vértice com menor distância relativa a I e adiciona-o a S e, então, repetindo os passos até que todos os vértices alcançáveis por I estejam em S (arestas que ligam vértices já pertencentes a S são desconsideradas).

Um exemplo prático do problema que pode ser resolvido pelo algoritmo de Dijkstra é: alguém precisa se deslocar de uma cidade para outra. Para isso, ela dispõe de várias estradas, que passam por diversas cidades. Qual delas oferece uma trajetória de menor caminho?

O código de scilab começa com o usuário definindo a matriz referente ao

grafo:

$$G = \begin{pmatrix} 2 & 3 & \%inf & 4 \\ 3 & 0 & 5 & \%inf \\ \%inf & 5 & 0 & 2 \\ 4 & \%inf & 2 & 0 \end{pmatrix}$$

```
x=size(G,1);
y=size(G,2);
vertices=max(size(G));
if x<>y then
    printf("Grafo Inválido!\n");
end

    //Algoritmo de Dijkstra
    function custo = dijkstra(G,inicio,fim)
        custo(1:vertices) = %inf;
        visitados(1:vertices) = 0;
        custo(inicio) = 0;
        while sum(visitados) < vertices;
            candidatos(1:vertices) = %inf;
            for index=1:vertices
                if visitados(index) == 0 then
                    candidatos(index) = custo(index);
                end
            end
            [custoatual, verticeatual] = min(candidatos);
            for index=1:vertices
                novocusto = custoatual + G(verticeatual, index);
                if novocusto < custo(index) then
                    custo(index)=novocusto;
                end
            end
            visitados(verticeatual) = 1;
        end
    endfunction
```

Figura 2: Algorítmo de dijkstra scilab

Onde %inf representam os nós que não estão conectados entre si. Com base nesse grafo temos a função em scilab demonstrada na Fig. 2

Este código varre os caminhos possíveis entre o nó início e o nó fim e marca como 1 os nós visitados e 0 os nós não visitados e enquanto percorre os caminhos faz uma comparação para saber qual o caminho de menor custo, sendo

o custo representado pelos números na matriz. Os nós início e fim utilizados na função sao gerados aleatoriamente utilizando os seguintes comandos:

```
random1 = grand(2, "prm", 1:vertices);//Números Aleatórios
    iniciod=random1(1,1);
    fimd=random1(2,1);
    custo=dijkstra(G,iniciod,fimd);
```

Figura 3: Números aleatórios

Onde random1 é uma matriz 2x4 em que 2 representa a quantidade de números aleatórios nescessários e 4 representa a quantidade de nós, os números a seres selecionados aleatoriamente.

2.2. Algorítmo de busca em profundidade

Na teoria dos grafos, busca em profundidade (ou busca em profundidade-primeiro, também conhecido em inglês por Depth-First Search - DFS) é um algoritmo usado para realizar uma busca ou travessia numa árvore, estrutura de árvore ou grafo. Intuitivamente, o algoritmo começa num nó raiz (selecionando algum nó como sendo o raiz, no caso de um grafo) e explora tanto quanto possível cada um dos seus ramos, antes de retroceder(backtracking).

O algoríitmo de busca escolhido foi o de busca em profundidade e ele segue o principio citado em 2.1 em que marca 1 os nós visitados e 0 como os não visitados e prossegue até que todos os nós tenham sido listados. O método de varredura segue o mesmo princípiode 2.1 em que tem-se um nó inicio e um nó fim que são gerados aleatoriamente com o mesmo método ultilizado na Fig. 3, demonstrado na Fig. 4 a seguir:

```
function buscap(vertice)
    global visitado
    global caminho
    if vertice==fimb then
        if caminho(1,1)==0 then
            caminho(1,1)=vertice
        else
            caminho=[caminho,vertice]
        end
    [nrRows,nrCols]=size(caminho)
    for counter=1:nrCols
        printf('%d ',caminho(1,counter))
    end
    printf('\n')
    else
        visitado(1,vertice)=1;
        if caminho(1,1)==0 then
            caminho(1,1)=vertice
        else
            caminho=[caminho,vertice]
        end
    for nr=1:vertices
        if At(vertice,nr)==1 then
            if visitado(1,nr)==0 then
                buscap(nr)
            end
        end
    end
end
visitado(1,vertice)=0
caminho(:,$)=[]
endfunction
```

Figura 4: Agorítmo de busca em profundidade

3. Matriz Adjacência

Nesta seção será abordado o assunto do item 2 da seção 1.

O código pega a matriz do grafo mencionada em 2.1 e a transforma em uma matriz adjacência, uma matriz de 1's e 0's que representa um grafo, a matriz foi montada simplesmente percorrendo a matriz G e analisando que se A(i,j) fosse zero ou %inf na matriz A seria 0 e se fosse um número diferente de zero e %inf seria 1. Após isso foi montado o vetor de arestas simplesmente utilizando o comando "a=sum(At,'r');"que faz a soma das linhas e salva em

uma nova matriz chamada "a".

```
function AparaB(A)
B=zeros(vertices+sum(a),3)
B(1,3)=vertices+1
for ind=1:vertices
    B(ind,1)=-1
    B(ind,2)=0
    if ind>1 then
        B(ind,3)=B(ind-1,3)+a(1,ind-1)
    end
end
ind=vertices+1
aux=zeros(1,vertices)
aux(1)=B(1,3)+a(2)
for i=2:vertices
    aux(i)=aux(i-1)+a(i)
end
for i=1:vertices
            for l=1:vertices
                if A(i,l)<>0 then
                    B(ind,1)=l
                    B(ind,2)=A(i,l)
                    B(ind,3)=ind+1
                else
                    continue;
                end
                ind=ind+1
            end
            B(aux(i),3)=0
end
printf("\n\nMatriz B convertida da matriz A:");
printf("\n  Nó  Peso  Aponta")
disp(B);
endfunction
```

Figura 5: Código de transformação da matriz adjacência A para a matriz B

Após isso a próxima parte do código preecheria a matriz B na coluna "nó"com -1 e com zero na coluna "peso"até a linha que tem mesmo número da quantidade de vértices do grafo. O código também analisa as adjacências dos vértices e completa a matriz B de acordo apontando sempre à linha

seguinte a não ser que o número do vértice tenha mudado, demonstrado no código da Fig. 5.

A função pega uma dada matriz A e a converte para a matriz B, gerando a matriz a seguir:

$$B = \begin{pmatrix} Nó & Peso & Aponta \\ -1 & 0 & 5 \\ -1 & 0 & 8 \\ -1 & 0 & 10 \\ -1 & 0 & 12 \\ 1 & 2 & 6 \\ 2 & 3 & 7 \\ 4 & 4 & 0 \\ 1 & 3 & 9 \\ 3 & 5 & 0 \\ 2 & 5 & 11 \\ 4 & 2 & 0 \\ 1 & 4 & 13 \\ 3 & 2 & 0 \end{pmatrix}$$

Foi feito também uma outra função com objetivo oposto à anterior, esta outra função recebe uma matriz B e a transforma em uma matriz adjacência A, código demonstrado na Fig. 6 a seguir:

```
function BparaA(B_)
v=1
while B_(v,1)==-1
    v=v+1
end
linhasB = size(B_,1)
nós=v-1
a_=zeros(1,nós)
for i=1:nós
    if i < nós then
        a_(1,i) = B_(i+1,3) - B_(i,3)
    else
        a_(1,i) = linhasB - (nós + sum(a_));
    end
end
A_=zeros(nós,nós)
cont = nós;
for k=1:nós
    for i=1:a_(k)
        A_(k,B_(cont+i,1))=B_(cont+i,2);
    end
    cont = cont+a_(k);
end
printf("\nMatriz A convertida da matriz B:")
disp(A_);
endfunction
```

Figura 6: Código de transformação da matriz B para a matriz adjacência A

4. Matriz adjacência e vetores posição de um mapa do Brasil

Nesta seção será abordado o assunto do item 3 da seção 1. Na Fig. 7 temos uma imagem com o mapa do Brasil dado para o trabalho juntamente com uma matriz adjacência e uma tabela das posições de x e y de seus pontos, que foram gerados utilizando o programa GeoGebra.

Matriz Adjacência

```
A = [0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
     1 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
     0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
     1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0;
     0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
     0 1 1 0 1 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0;
     0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 1 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 1 0 0 0 1 1 1 0 1 0 0 1 0 1 0 0 0 0 0 0;
     0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0;
     0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1;
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 1;
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 0 0 0;
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0;
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0;
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 1;
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1;
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0;
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 1 1 0 0 0;]
```

Vetor Posição		
Estado	X	Y
AC	2.63	13.5
AM	5.85	17.73
RR	5.5	21.11
RO	4.18	14.29
AP	9.74	19.6
PA	10.94	18.43
MA	12.77	17.54
PI	13.27	16.38
CE	14.89	16.94
RN	16.37	15.71
PB	16.59	15.11
PE	16.46	14.57
AL	16.15	13.72
SE	15.58	13
BA	14.95	12.02
TO	10.84	13.63
MT	7.68	10.44
DF	10.94	10.44
GO	10.4	9.84
MG	12.55	7.94
ES	13.81	7.66
RJ	12.74	6.27
SP	11.35	6.08
PR	10.4	4.88
SC	10.49	3.77
RS	9.61	2.29
MS	8	8

Figura 7: Matriz adjacência e tabela de posições de um mapa do Brasil

5. Montagem de grafo utilizando matriz adjacência e vetor posição

Nesta seção será abordado o assunto do item 4 da seção 1.

```
clear;
clc;

load('dados.txt');

s = size(A)
n = s(1)

for i=1:n
    for j=1:n
        if A(i,j)==1 then
            plot([x(i) x(j)],[y(i) y(j)],'.-r')
        end
    end
end

for i=1:n
    t=string(i);
    xstring(x(i),y(i),t)
end
```

Figura 8: Código-fonte da montagem de grafos

Nesta parte foi feito um código mostrado na Fig. 8 que tendo uma matriz adjacência de um grafo e dois vetores de posição, um de x e um de y, monta um grafo em forma de imagem.

6. Menu iterativo

Nesta seção será abordado o assunto do item 5 da seção 1. Nesse item do trabalho foi utilizada a ferramenta GuiBuilder do scilab, que permite criar uma interface garffica visualmente e depois gerar o código.

```
1  function brasil_callback(handles)
2  //Write your callback for brasil here
3  exec('Mapa Brasil.sce');
4  endfunction
26
27
1  function inglaterra_callback(handles)
2  //Write your callback for inglaterra here
3  exec('Mapa Inglaterra.sce');
4  endfunction
32
33
1  function mercosul_callback(handles)
2  //Write your callback for mercosul here
3  exec('Mapa Mercosul.sce');
4  endfunction
38
1  function clear_callback(handles)
2  //Write your callback for clear here
3  close()
4  exec('3.3.3.sce')
5  endfunction
```

Figura 9: Código-fonte da interface gráfica

Foi criada uma interface simples com quatro botões: um três para selecionar Brasil, Inglaterra ou Mercosul e um para limpar o que foi gerado. No código desse programa foi programado para que quando pressionado um dos botões das opções chama-se um outro código de scilab que geraria a imagem da opção selecionada como mostra a Fig. 9

O código que gera a imagem dos mapas com os seus grafos equivalentes citados em 5a,5b,5c e 5d da seção 1 tem o mesmo código mudando somente o nomes dos arquivos de imagem dos mapa e do de dados, na Fig. 10 a seguir está o código citado.

```
imagem = imread('Nome da imagem.jpg')
s = size(A)
n = s(1)
razao = 29.5
subplot(2,2,1)
imshow(imagem);
subplot(2,2,2)
for i=1:n
    for j=1:n
        if A(i,j)==1 then
            plot([x(i) x(j)],[y(i) y(j)],'.-b')
        end
    end
end

for i=1:n
    t=string(i);
    xstring(x(i),y(i),t)
end

subplot(2,2,3)
for i=1:n
    for j=1:n
        if A(i,j)==1 then
            plot([x(i) x(j)],[y(i) y(j)],'.-b')
        end
    end
end
```

```

for i=1:n
    t=r(i);
    xstring(x(i),y(i),t)
end

subplot(2,2,4)
imshow(imagem);
imagem1 = rgb2gray(imagem)
imshow(imagem1);
x2=x*razao
y2=y*razao
for i=1:n
    for j=1:n
        if A(i,j)==1 then
            plot([x2(i) x2(j)],[y2(i) y2(j)],'.-b')
        end
    end
end
for i=1:n
    t=string(i);
    xstring(x2(i),y2(i),t)
end
```

Figura 10: Código-fonte da montagem dos mapas

Nas Fig. 11, 12 e 13 a seguir estão os mapas gerados pelo programa.

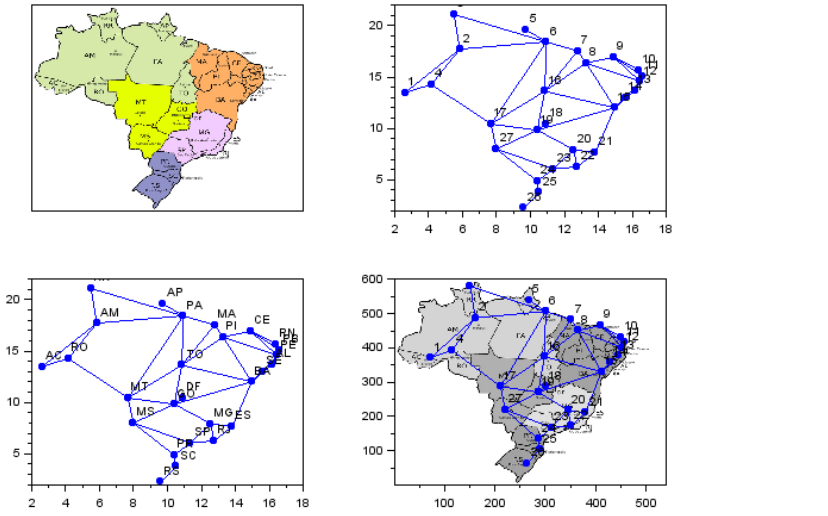

Figura 11: Mapa Brasil

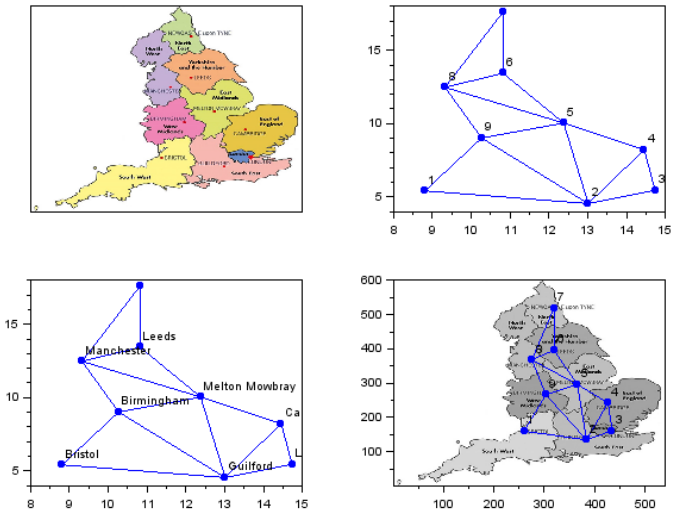

Figura 12: Mapa Inglaterra

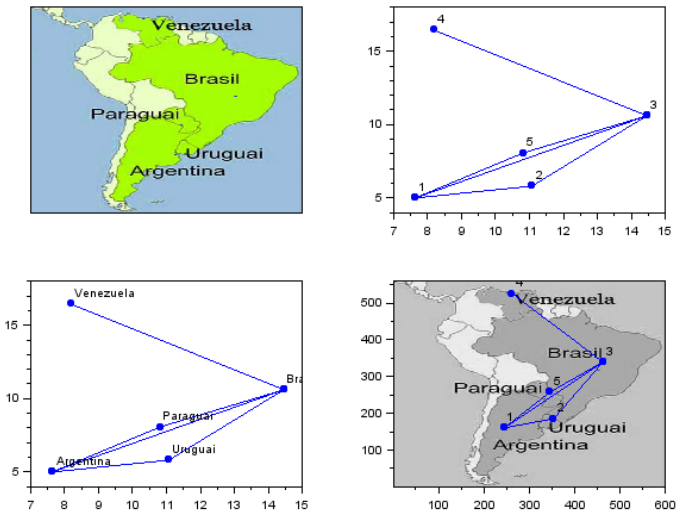

Figura 13: Mapa Mercosul

7. Conclusão

Foi concluído que este trabalho computacional foi algo bom a ser feito, pois ajudou a adcionar conhecimentos anteriormente não possuídos, como o conhecimento da linguagem de programação SciLab, que é muito boa para programas envolvendo cálculos matemáticos por sua grande variedade de funções trantando tal assunto e por ser próxima da linguagem de MatLab que é outra linguagem ótima para o mesmo, juntamente com o novo conhecimento adquirido de interfaces gráficas.

Uma das grandes dificuldades encontradas para o desenvolvimento do trabalho foi a falta de conhecimento da linguagem, por ser algo que nunca foi visto por nossa parte e por consequência disso uma dificuldade na implementação das tarefas solicitadas para o trabalho.

Referências

[1] Judith L. Gersting, (2007) *Mathematical structures for computer science*, Macmillan

[2] Polyanna Possani da Costa, (2011) *Teoria dos grafos e suas aplicações*, Universidade Estadual Paulista (UNESP)

[3] Thomas H. Cormen and Charles E. Leiserson and Ronald L. Rivest and Clifford Stein, (2002) *Algoritmos: teoria e prática*, Editora Campus, Vol. 2

[4] `https://help.scilab.org/docs/5.4.1/pt_BR/index.html`

[5] `https://www.scilab.org/tutorials/application-development-%E2%80%93-gui-building`

[6] `https://rosettacode.org/wiki/Dijkstra%27s_algorithm`

[7] `https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/dfs.html`