SIN110 Algoritmos e Grafos

aula 11

Grafos

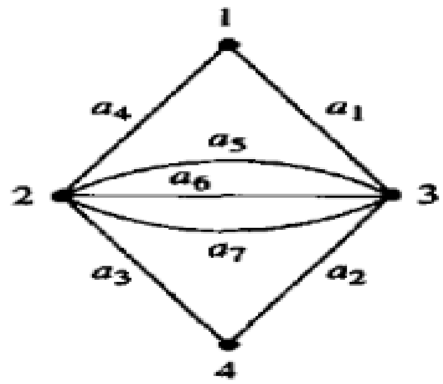
- *modelagem*, representação computacional (E07)
- Busca em Profundidade
- Busca em Largura

Grafos:

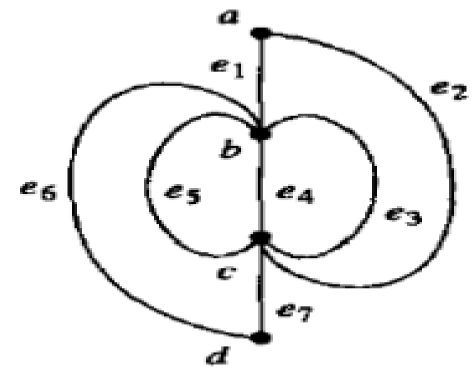
- conceitos,
- modelagem,
- representação computacional

E07 solução

1) Verificando os grafos abaixo diga se eles são isomorfos. Justifique *o sim* ou *o não*.



(a)



(b)

Observando, temos uma função f(Va) → f(Vb) tal que:

1 → a; 4 → d; 2 → b; 3 → c.

E as arestas (x,y)_a → (f(x),f(y))_b:

$(1,2) = a_4 \rightarrow e_1 = (a,b)$

$(1,3) = a_1 \rightarrow e_2 = (a,c)$

$(2,4) = a_3 \rightarrow e_6 = (b,d)$

$(3,4) = a_2 \rightarrow e_7 = (c,d)$

$(2,3) = a_5 \rightarrow e_3 = (b,c)$

$(2,3) = a_6 \rightarrow e_4 = (b,c)$

$(2,3) = a_7 \rightarrow e_5 = (b,c)$

Concluindo os grafos (a) e (b) são isomorfos.

2) nome "ABEL SÁ" dígrafo $G = (V,E)$ onde,

$V = \{A, B, E, L, S\}$ e

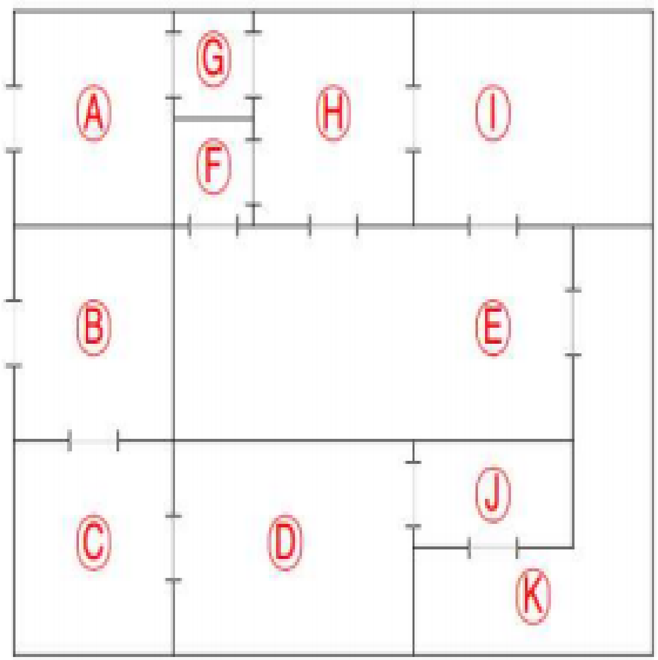
$E = \{(A,B), (B,E), (E,B), (E,L), (L,S), (S,A), (A,S)\}$

- Listas Adj:
  $A \rightarrow B \rightarrow S$
  $B \rightarrow E$
  $E \rightarrow B \rightarrow L$
  $L \rightarrow S$
  $S \rightarrow A$

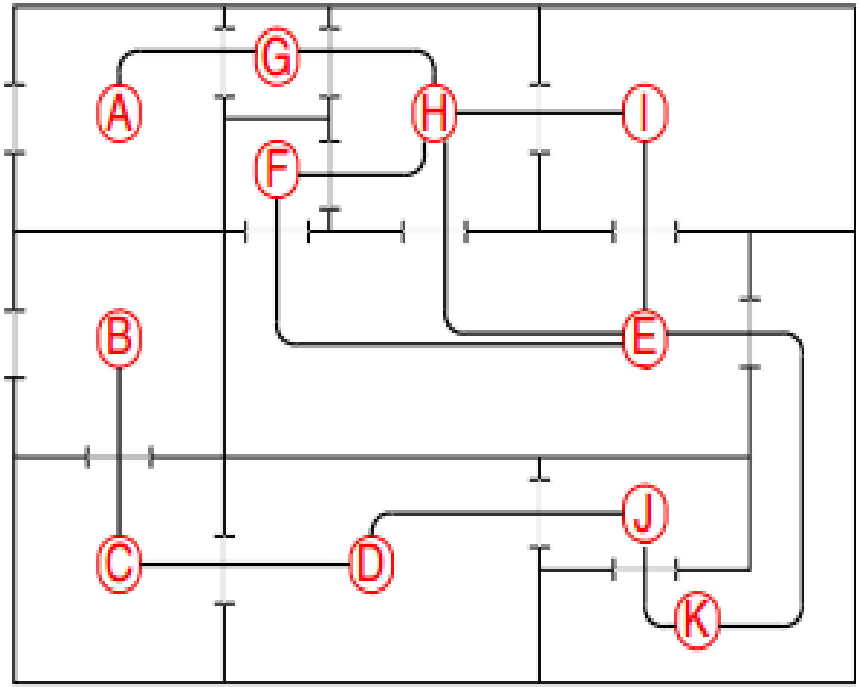
$$\begin{array}{ccc} A \rightarrow B & \leftrightarrow & E \\ \uparrow\downarrow & & \downarrow \\ S & \longleftarrow\text{-------} & L \end{array}$$

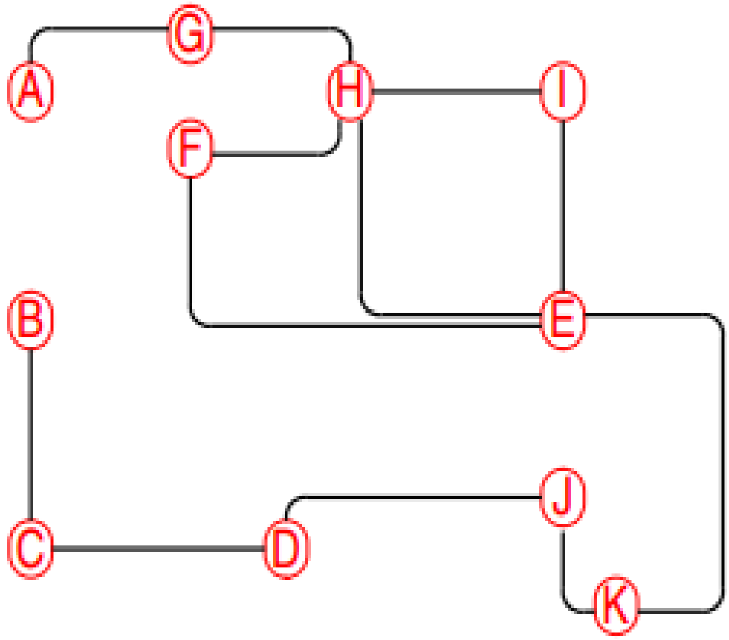
- Dígrafo de ordem 5;
- Fonte e sumidouro: não tem
- Ciclo: A – B – E – L – S – A
- Ponte e vert artic: não tem
- Dígrafo conexo, todos vértices alcançáveis a partir de todos demais.

3) Uma casa possui uma divisão representada pela planta abaixo. É possível uma pessoa sair do cômodo A, terminar no cômodo B e passar por todas as portas da casa exatamente uma única vez? Se sim, apresente um possível trajeto. Se encontrou um trajeto, poderia dizer que é Euleriano ou Hamiltoniano? Justifique.



A planta da casa pode ser representada pelo grafo abaixo:





Cada vértice deste grafo tem um grau par, exceto os vértices A e B que têm grau 1. Assim, existe um trajeto Euleriano de A para B.

➜ AGHFEIHEKJDCB

4) Um “grafo de palavras” é definido assim: cada vértice é uma palavra da língua portuguesa e duas palavras são adjacentes se diferem em exatamente uma posição. Por exemplo, rato e ralo são adjacentes, enquanto ralo e rota não são. Faça uma figura da parte do grafo definida pelas palavras abaixo:

caiado	cavado	cavalo	girafa	girava	ralo	ramo	rata	rato
remo	reta	reto	rota	vaiado	varado	virada	virado	virava

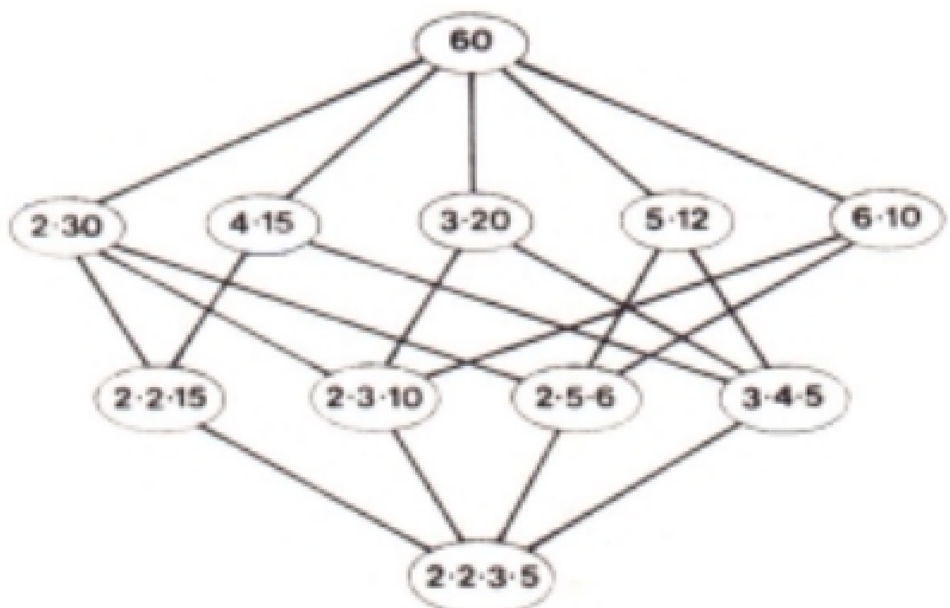
É possível sair de "girafa" e chegar em "cavalo" andando pelas arestas do grafo? Se for possível mostre o caminho encontrado.

Observando o grafo:

cavalo → cavado → caiado → vaiado → varado → virado → virada → virava → girava → girafa

É possível!

5) Elabore um grafo que representa as fatorações no número 60. Descreva o grafo G mostrando sua representação em Listas de Adjacências.



As fatorações de 60

6) Os amigos João, Pedro, Antônio, Marcelo e Francisco sempre se encontram para botar conversa fora e às vezes jogar dama, xadrez e dominó. As preferências de cada um são as seguintes: João só joga xadrez; Pedro não joga dominó; Antônio joga tudo; Marcelo não joga xadrez e dominó e Francisco não joga nada.

a)Represente através de um grafo bipartido G=(V,E) todas as possibilidades de um amigo jogar com os demais. Defina V e E.

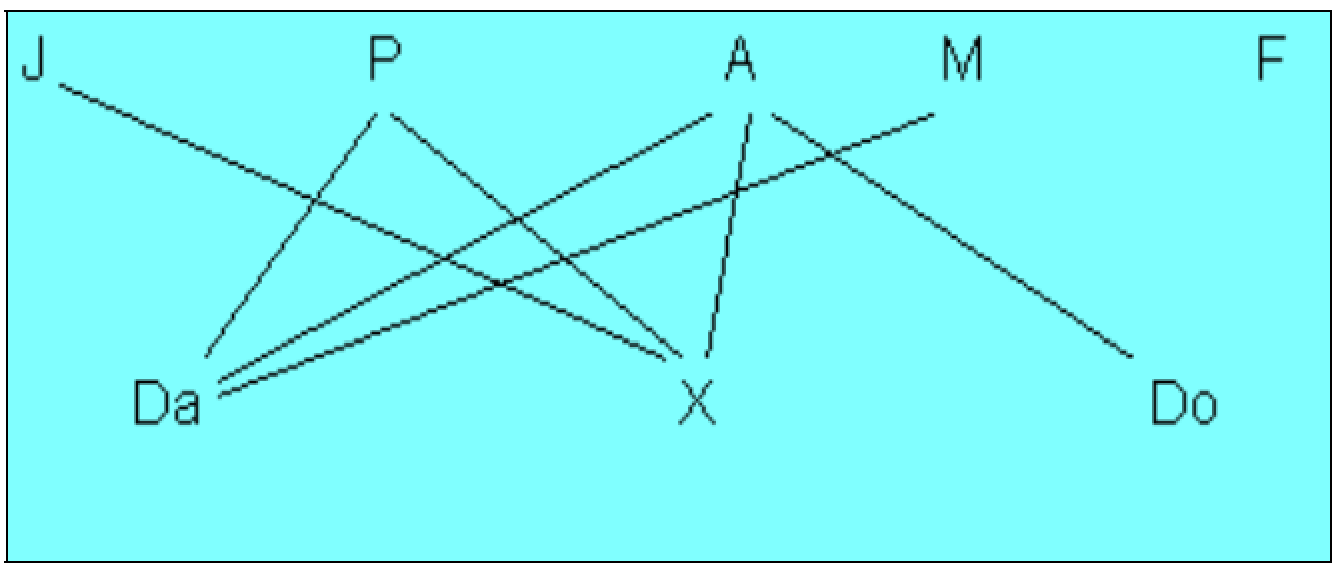
b) Defina um subgrafo em que todos, menos Francisco, joguem ao mesmo tempo.

c) A partir do grafo bipartido do item (a), construa um grafo rotulado que mostra "o quê", quem pode jogar com quem.

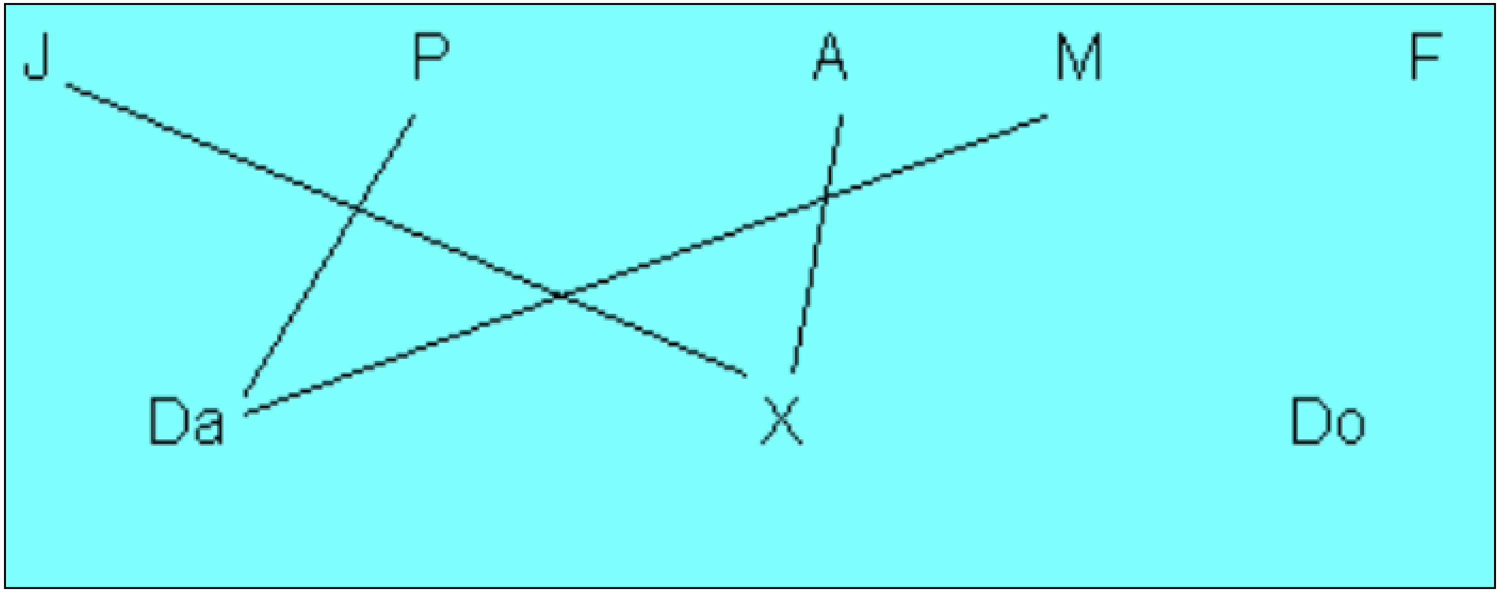
a) Represente através de um grafo bipartido $G=(V,E)$ todas as possibilidades de um amigo jogar com os demais. Defina V e E.

V{(J(oão), P(edro), A(ntônio), M(arcelo), F(rancisco), Da(ma), X(adrez), Do(minó)}

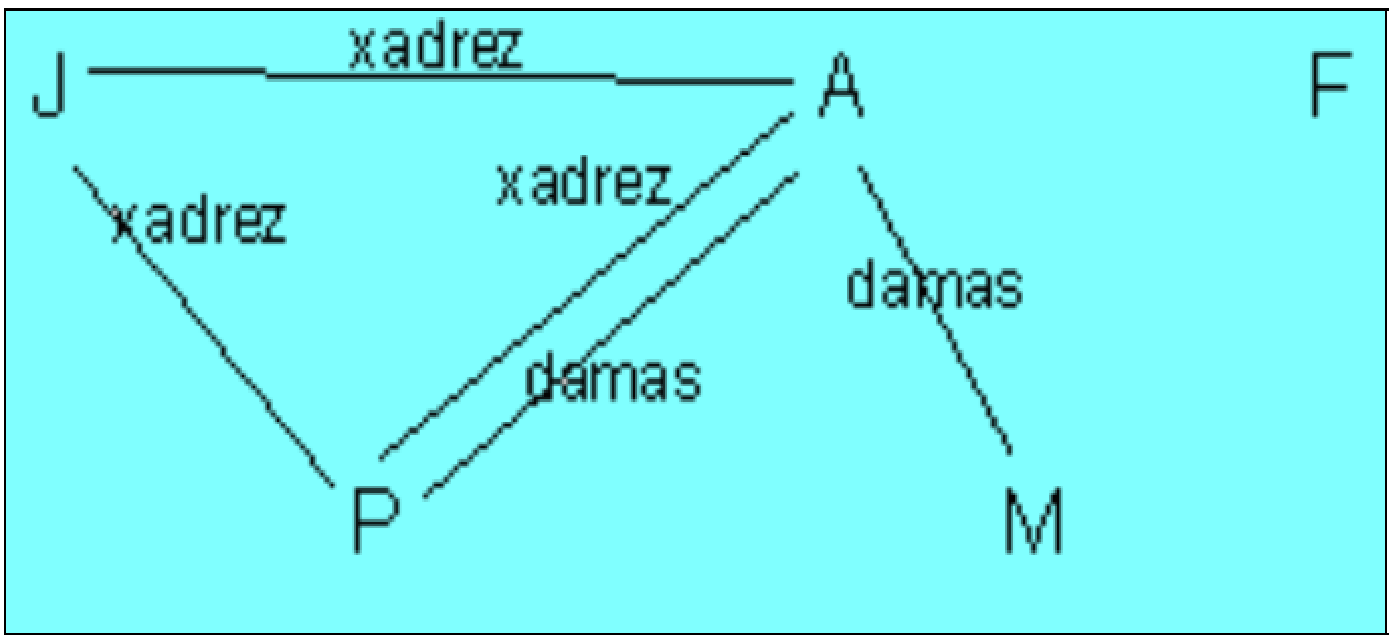
E={(J,X), (P,Da), (P, X), (A,X), (A,DA), (A,Do), (M,Da)},



b) Defina um subgrafo em que todos, menos Francisco, joguem ao mesmo tempo.



c) A partir do grafo bipartido do item (a), construa um grafo rotulado que mostra "o quê", quem pode jogar com quem.



7. Exercício com modelagem em grafos

Considere 3 jarros com capacidades de 8, 5 e 3 litros.

O maior, 8 litros, está cheio de vinho.

Deseja-se dividir esse vinho em duas porções de 4 litros para transporte.

Desenvolva uma sequencia de operações utilizando apenas os 3 jarros na execução para alcançar a divisão.

Exercício: modelagem em grafos

Considere 3 jarros com capacidades de 8, 5 e 3 litros. O maior, 8 litros, está cheio de vinho. Deseja-se dividir e duas porções de 4 litros para transporte. Desenvolva uma sequencia de operações utilizando apenas os 3 jarros na execução para alcançar a divisão.

→ *Desenvolvemos a solução modelando em um dígrafo*

Exercício: modelagem em grafos

→ Considere 3 jarros com capacidades de 8, 5 e 3 litros. O maior, 8 litros, está cheio de vinho. Deseja-se dividir e duas porções de 4 litros para transporte. Desenvolva uma sequencia de operações utilizando apenas os 3 jarros na execução para alcançar a divisão.

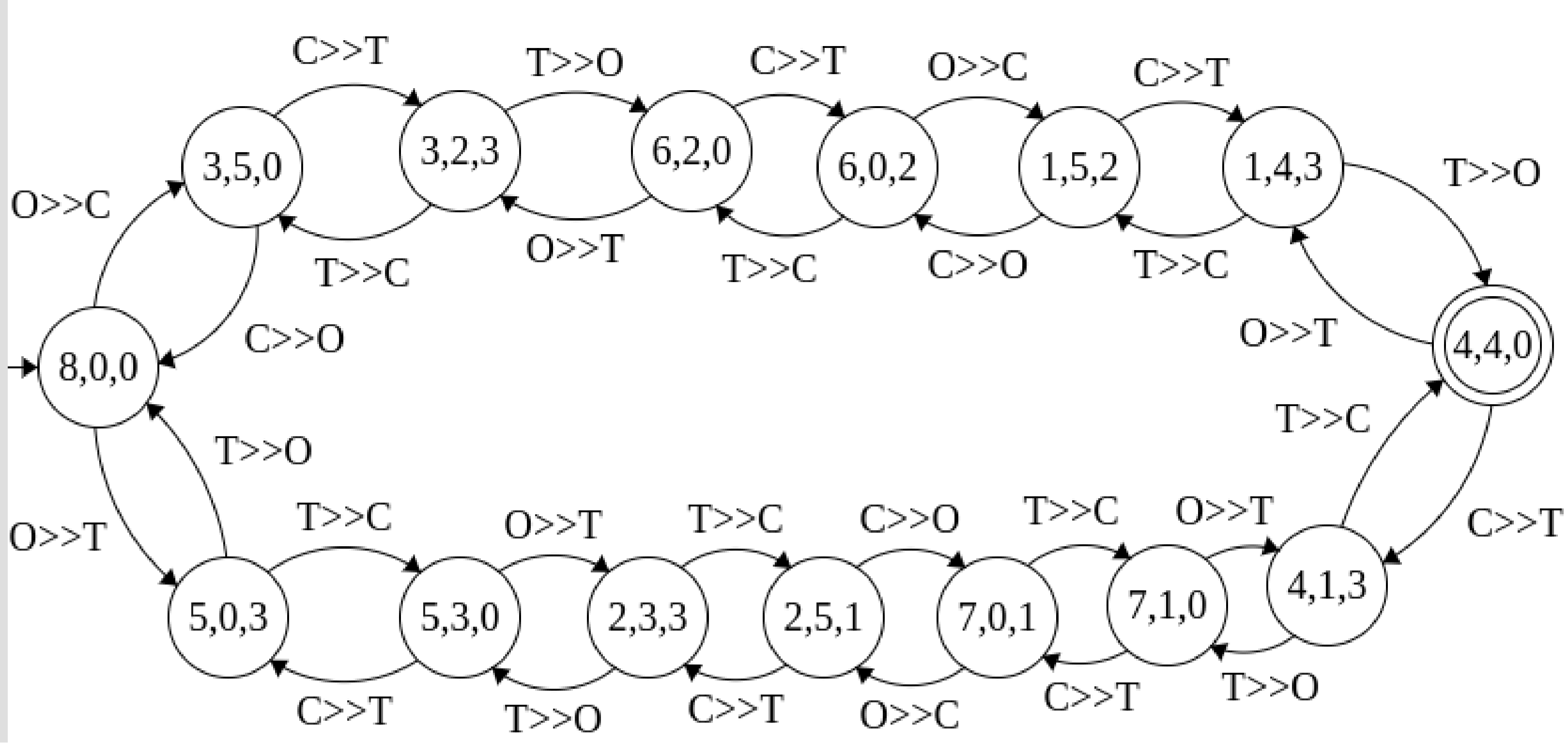
Modelando:

- *Vértices são os 3 jarros em triplas: (O,C,T) indicando o volume de vinho em cada um (exemplos: [8,0,0] de início, [4,4,0] no final e intermediários como [2,3,3] ou [5,3,0]) .*
- *Arcos com rótulos que indicam o movimento entre dois jarros, exemplos: "O>>T" – do jarro de oito litros para o de três litros (um movimento em que enche o de menor volume) ou, "T>>C" (esvazia o conteúdo do jarro de 3 litros no jarro de 5 litros) e assim por diante.*

Exercício - modelagem em grafos

→ Considere 3 jarros com capacidades de 8, 5 e 3 litros. O maior, 8 litros, está cheio de vinho. Deseja-se dividir e duas porções de 4 litros para transporte. Desenvolva uma sequencia de operações utilizando apenas os 3 jarros na execução para alcançar a divisão.

Dígrafo - solução com as duas sequencias possíveis:



Busca em Profundidade (DFS)

Busca em Profundidade (DFS)

- Explorar vértices descobertos mais recentes primeiro
- **Interpretação**
  - procurar uma saída de um labirinto
  - vai fundo atrás da saída (tomando decisões a cada encruzilhada)
  - volta a última encruzilhada quando encontrar um beco sem saída (ou lugar já visitado)

Pesquisa primeiro na profundidade

A busca em profundidade – do inglês depth-first search: DFS

Na busca em profundidade, as arestas são exploradas a partir do vértice visitado mais recentemente.

Sempre que um vértice v é descoberto durante a busca na lista de adjacências de um outro vértice já visitado u, busca em profundidade (DFS) memoriza este evento ao definir o predecessor de v - pred(v) - como u.

O vetor de predecessores forma uma árvore, o vetor predecessor da busca em profundidade pode ser composto de várias árvores.

O algoritmo DFS recebe um grafo cujos vértices são coloridos durante a busca:

- BRANCO: antes da busca;
- CINZA: quando o vértice for visitado e,
- PRETO: quando os vértices adjacentes já foram visitados.

DFS emprega uma variável global “tempo”, que marca as visitas em cada vértice em dois instantes:

- $d(v) \leftarrow$ tempo, no instante em que v foi visitado (pintado de CINZA) ,
- $f(v) \leftarrow$ tempo, no instante em que a busca pelos vértices na lista de adjacências de v foi completada sendo então pintado de PRETO.

Algoritmo:

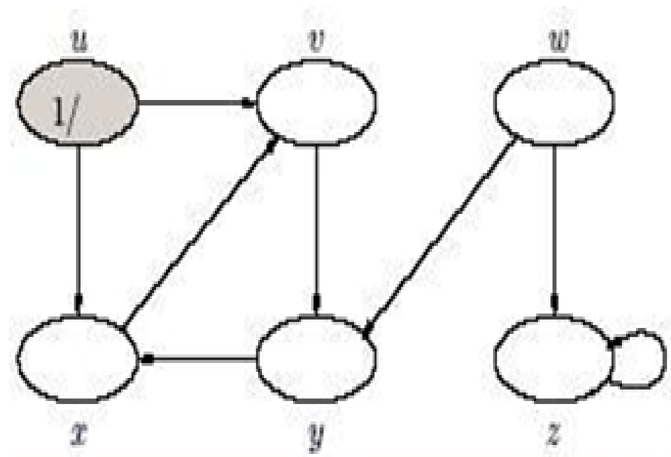
DFS(G)

1. para cada vértice u em G faça
2. cor[u] = BRANCO
3. pred(u) $\leftarrow$ NIL
4. tempo $\leftarrow$ 0
5. para cada vértice u em G faça
6. se cor(u) = branco
7. então Visita_DFS(u)

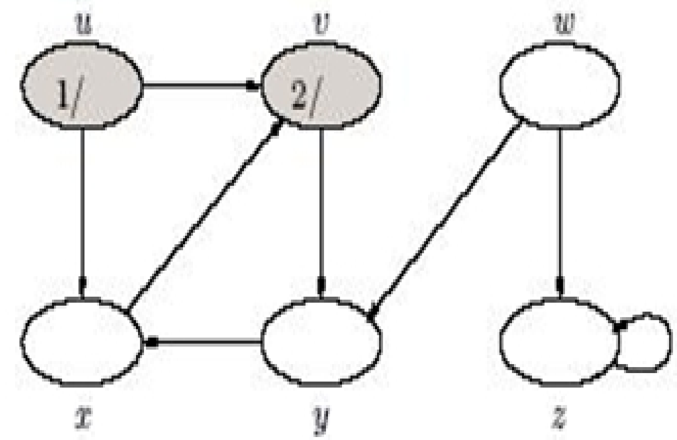
Visita_DFS(u)

```
cor[u] ← CINZA
tempo ← tempo + 1
d(u)  ← tempo
para cada v em Adj(u) faça
        se cor[v] = BRANCO
            então pred(v) ← u
                    Visita_DFS(v)
cor[u] ← PRETO
tempo ← tempo + 1
f(u)  ← tempo
```

Exemplo:

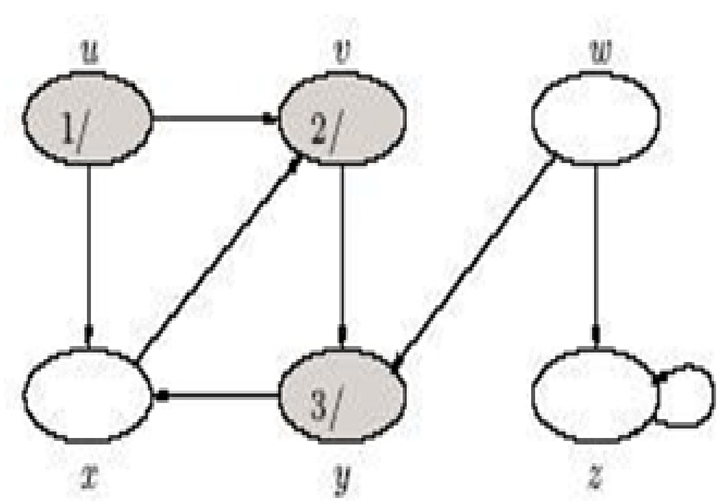


(i)

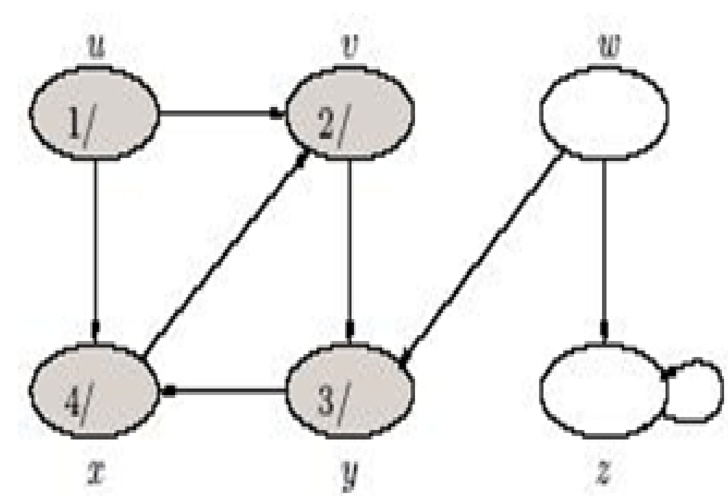


(ii)

Exemplo:

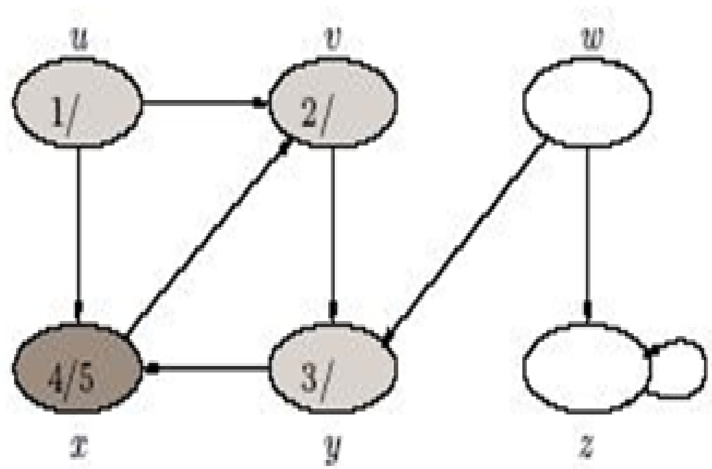


(iii)

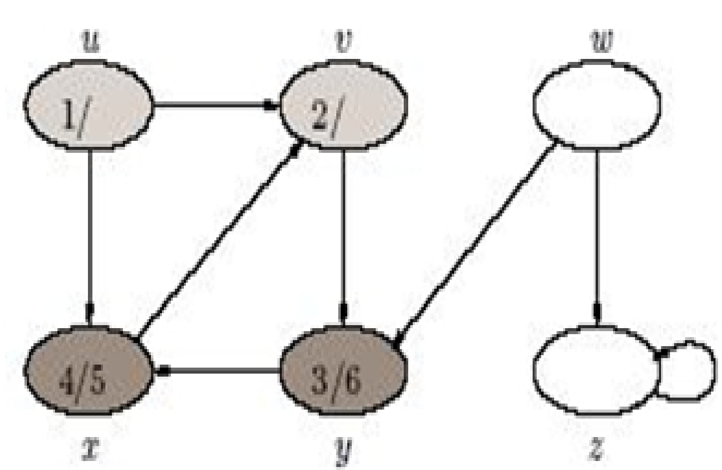


(iv)

Exemplo:

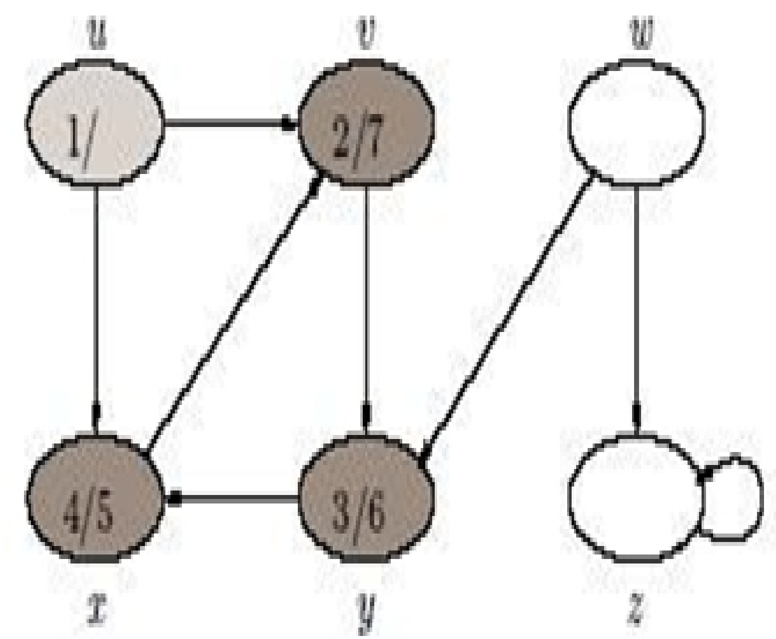


(v)

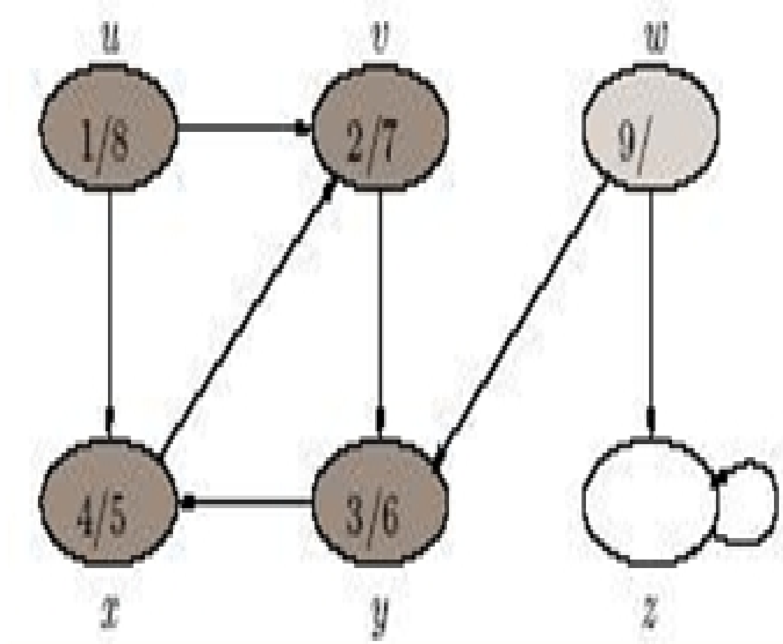


(vi)

Exemplo:

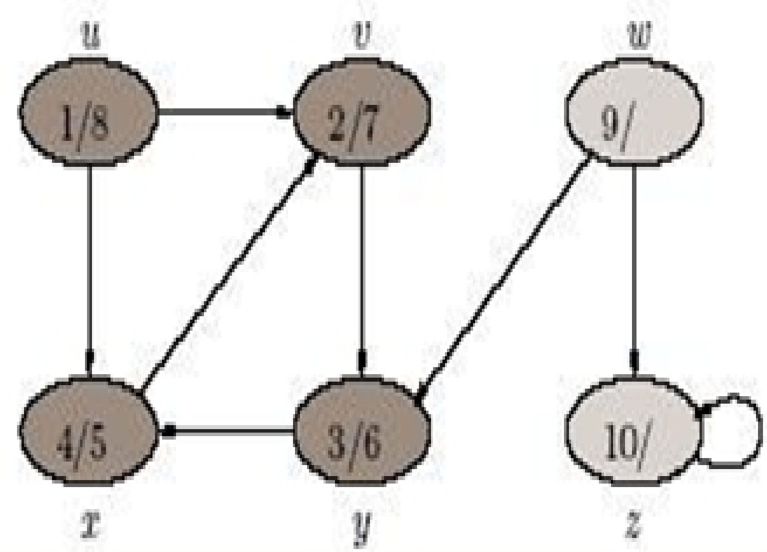


(vii)

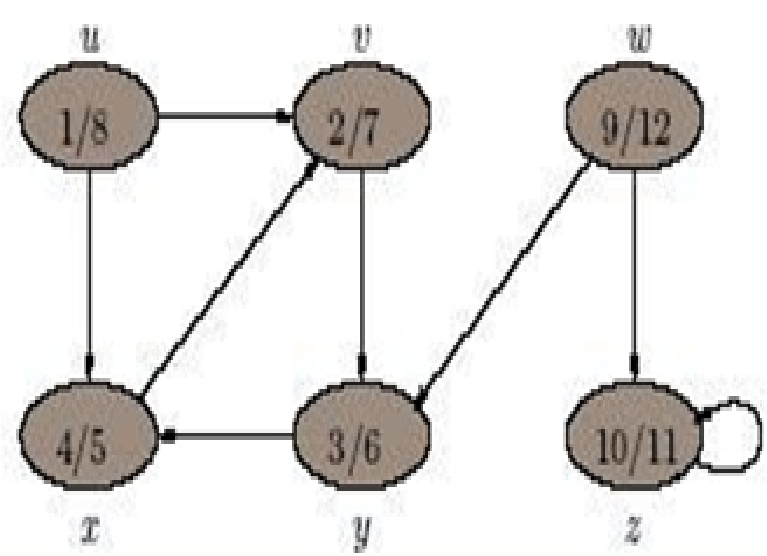


(viii)

Exemplo:



(ix)



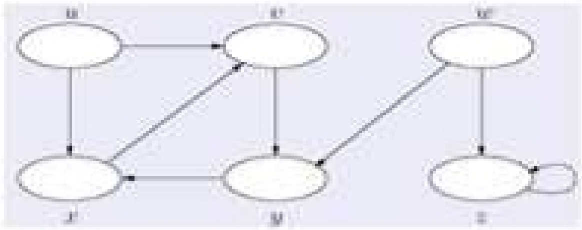
(x)

DFS(G)

```
para cada vértice u em G faça
      cor[u] = BRANCO
      pred[u] ← NIL
tempo ← 0
para cada vértice u em G faça
          se cor(u) = BRANCO
               então Visita_DFS(u)
devolve pred[1..n]
```

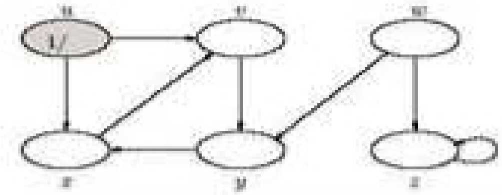
Visita_DFS(u)

```
 cor[u] ← CINZA
tempo ← tempo + 1
d(u)  ← tempo
para cada v em Adj(u) faça
          se cor[v] = BRANCO
                    então pred[v] ← u
                              Visita_DFS(v)
cor[u] ← PRETO
tempo ← tempo + 1
f(u)  ← tempo
```

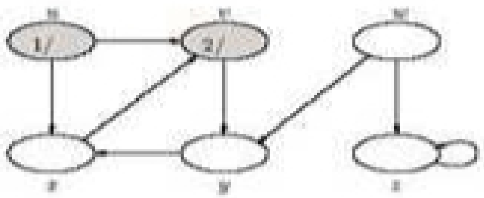


Dígrafo e suas listas adj:

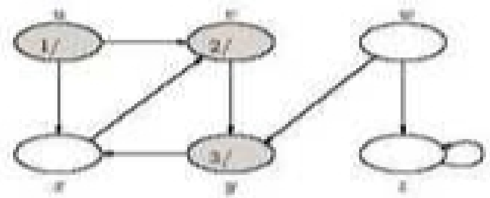
u	→	v	→	x
v	→	y		
w	→	z		
x	→	v		
y	→	x		
z	→	x		



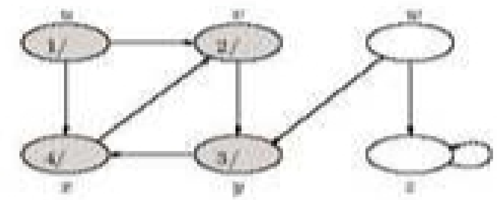
(i)



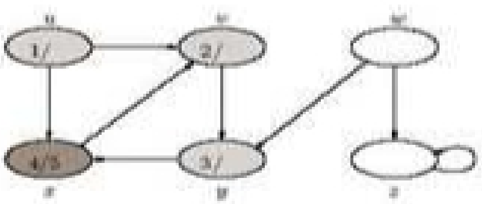
(ii)



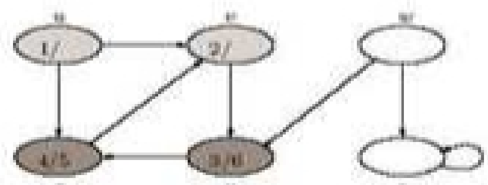
(iii)



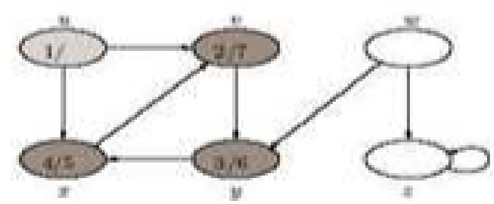
(iv)



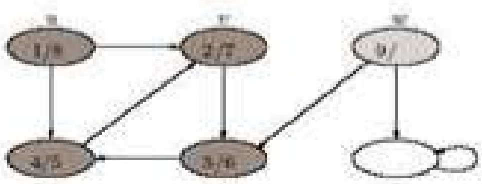
(v)



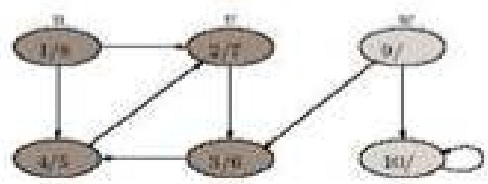
(vi)



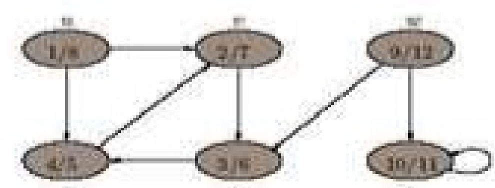
(vii)



(viii)

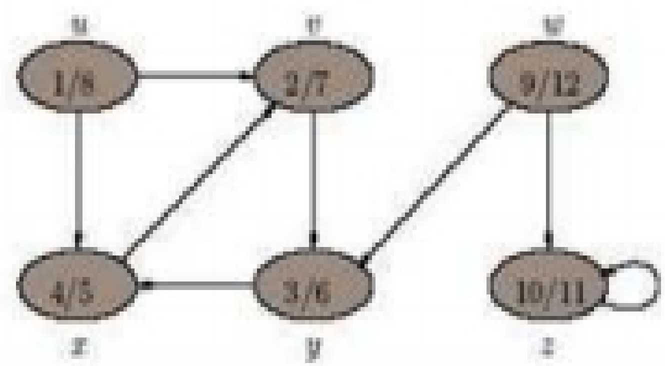


(ix)



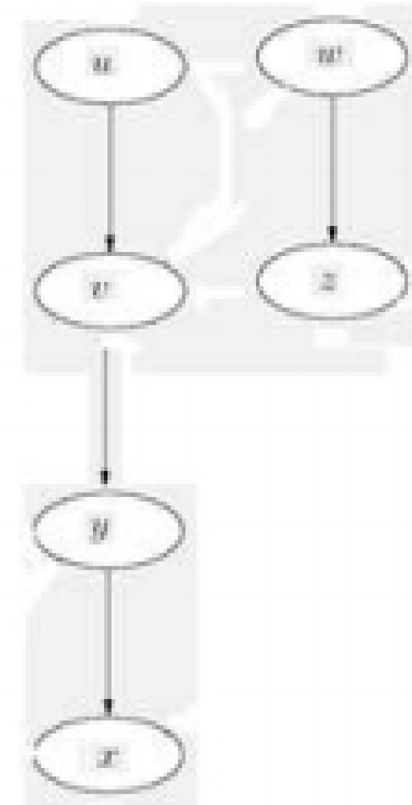
(x)

DFS



pred:	-	u	-	y	v	w
	[u]	[v]	[w]	[x]	[y]	[z]

Arborescências:



Busca em profundidade

- A ordem em que os nós e arestas são visitados depende:
  - Do vértice inicial
  - Da ordem em que os vértices e arestas aparecem na lista de adjacências

+ exemplo DFS

Aplique a **busca em profundidade,** ao *dígrafo* definido pelos arcos:

3-7 1-4 7-8 0-5 5-2 3-8 2-9 0-6 4-9 2-6 6-4

Considerando essa ordem de inclusão na construção do dígrafo em representação com listas de adjacências.

Arcos: 3-7 1-4 7-8 0-5 5-2 3-8 2-9 0-6 4-9 2-6 6-4, listas de adjacência:

Listas de Adjacência

0 → 5 → 6

1 → 4

2 → 9 → 6

3 → 7 → 8

4 → 9

5 → 2

6 → 4

7 → 8

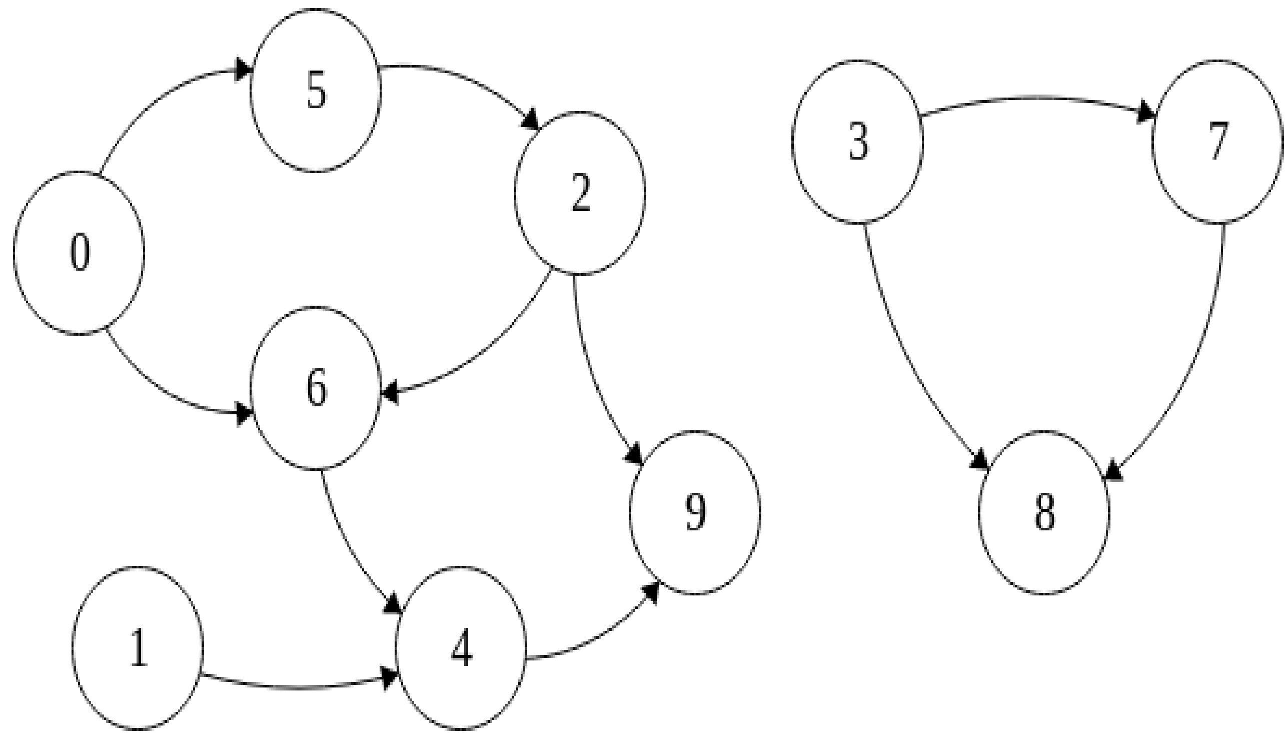
8

9

Listas de adjacência e, dígrafo correspondente:

Listas de Adjacência

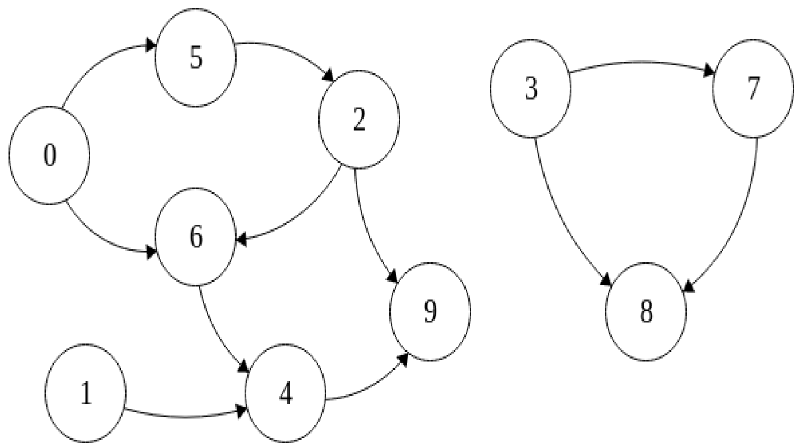
0 → 5 → 6
1 → 4
2 → 9 → 6
3 → 7 → 8
4 → 9
5 → 2
6 → 4
7 → 8
8
9



DFS:

Listas de Adjacência

0 → 5 → 6
1 → 4
2 → 9 → 6
3 → 7 → 8
4 → 9
5 → 2
6 → 4
7 → 8
8
9



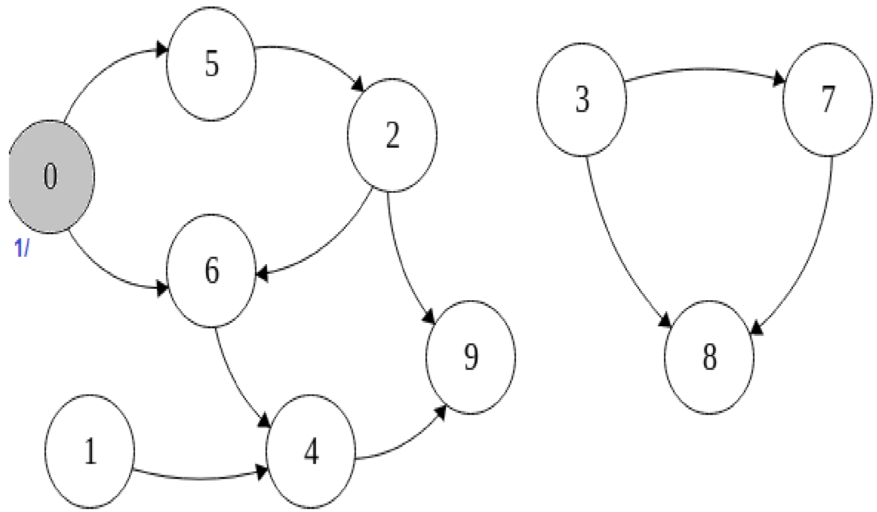
u	cor(u)	pred(u)	d(u)	f(u)
0	BR	-		
1	BR	-		
2	BR	-		
3	BR	-		
4	BR	-		
5	BR	-		
6	BR	-		
7	BR	-		
8	BR	-		
9	BR	-		

```
DFS(G)
1.  para cada vértice u em G faça
2.      cor[u] = BRANCO
3.      pred[u] ← NIL
4.  tempo ← 0
5.  para cada vértice u em G faça
6.          se cor(u) = BRANCO
7.              então Visita_DFS(u)
8.  devolve pred[1..n]

Visita_DFS(u)
1.  cor[u] ← CINZA
2.  tempo ← tempo + 1
3.  d(u) ← tempo
4.  para cada v em Adj(u) faça
5.          se cor[v] = BRANCO
6.                  então pred[v] ← u
7.                          Visita_DFS(v)
7.  cor[u] ← PRETO
8.  tempo ← tempo + 1
9.  f(u) ← tempo
```

DFS:

Listas de Adjacência

0 → 5 → 6
1 → 4
2 → 9 → 6
3 → 7 → 8
4 → 9
5 → 2
6 → 4
7 → 8
8
9




```
DFS(G)
1.  para cada vértice u em G faça
2.      cor[u] = BRANCO
3.      pred[u] ← NIL
4.  tempo ← 0
5.  para cada vértice u em G faça            u=0
6.      se cor(u) = BRANCO
7.          então Visita_DFS(u)
8.  devolve pred[1..n]

Visita_DFS(u)                                Vis(0)
1.  cor[u] ← CINZA
2.  tempo ← tempo + 1
3.  d(u) ← tempo
4.  para cada v em Adj(u) faça
5.      se cor[v] = BRANCO                   v=5
6.          então pred[v] ← u
7.              Visita_DFS(v)                Vis(5)
7.  cor[u] ← PRETO
8.  tempo ← tempo + 1
9.  f(u) ← tempo
```

u	cor(u)	pred(u)	d(u)	f(u)
0	CZ	-	1	
1	BR	-		
2	BR	-		
3	BR	-		
4	BR	-		
5	BR	-		
6	BR	-		
7	BR	-		
8	BR	-		
9	BR	-		

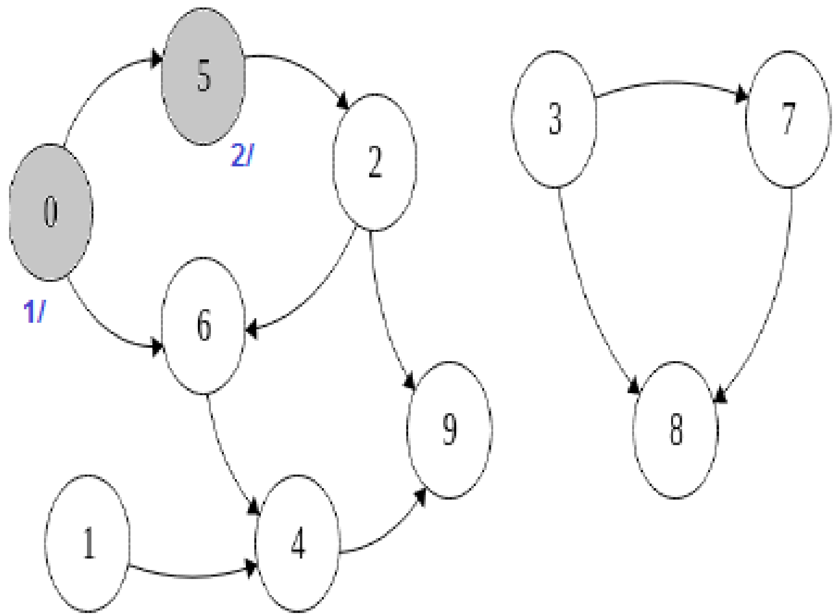
DFG

Vis(0)

DFS:

Listas de Adjacência

0 → 5 → 6
1 → 4
2 → 9 → 6
3 → 7 → 8
4 → 9
5 → 2
6 → 4
7 → 8
8
9




```
DFS(G)
1.  para cada vértice u em G faça
2.      cor[u] = BRANCO
3.      pred[u] ← NIL
4.  tempo ← 0
5.  para cada vértice u em G faça        u=0
6.      se cor(u) = BRANCO
7.          então Visita_DFS(u)
8.  devolve pred[1..n]

Visita_DFS(u)                            Vis(5)
1.  cor[u] ← CINZA
2.  tempo ← tempo + 1
3.  d(u)  ← tempo
4.  para cada v em Adj(u) faça           v=2
5.      se cor[v] = BRANCO
6.          então pred[v] ← u
7.              Visita_DFS(v)            Vis(2)
7.  cor[u] ← PRETO
8.  tempo ← tempo + 1
9.  f(u)  ← tempo
```

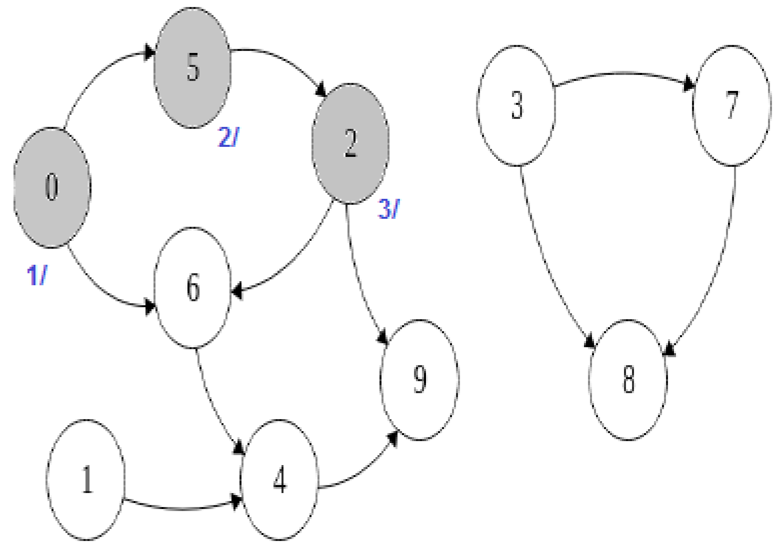
u	cor(u)	pred(u)	d(u)	f(u)
0	CZ	-	1	
1	BR	-		
2	BR	-		
3	BR	-		
4	BR	-		
5	CZ	0	2	
6	BR	-		
7	BR	-		
8	BR	-		
9	BR	-		

DFG
    Vis(0)
        Vis(5)

DFS:

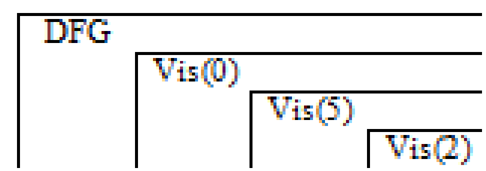
Listas de Adjacência

0 → 5 → 6
1 → 4
2 → 9 → 6
3 → 7 → 8
4 → 9
5 → 2
6 → 4
7 → 8
8
9



u	cor(u)	pred(u)	d(u)	f(u)
0	CZ	-	1	
1	BR	-		
2	CZ	5	3	
3	BR	-		
4	BR	-		
5	CZ	0	2	
6	BR	-		
7	BR	-		
8	BR	-		
9	BR	-		

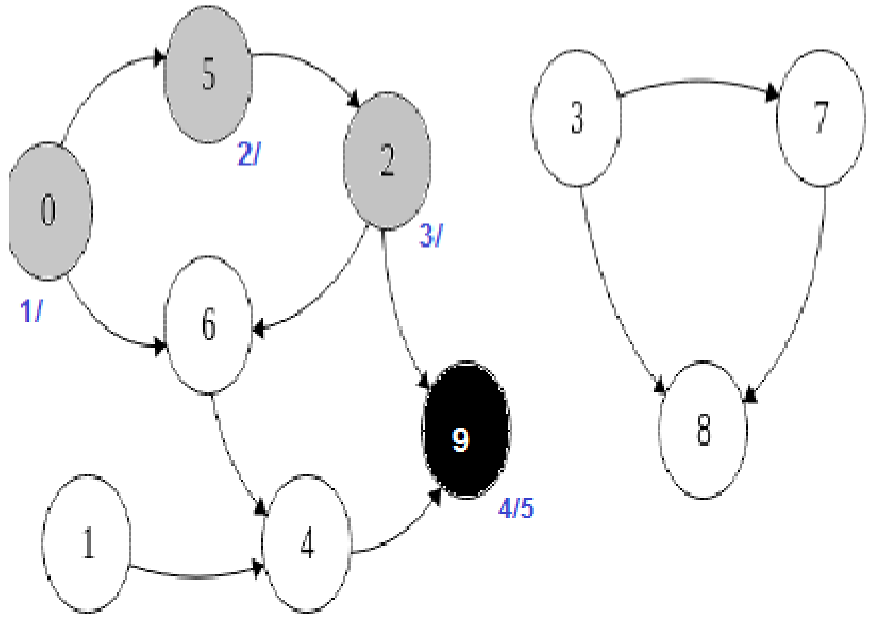
```
DFS(G)
para cada vértice u em G faça
    cor[u] = BRANCO
    pred[u] ← NIL
tempo ← 0
para cada vértice u em G faça          u=0
       se cor(u) = BRANCO
           então Visita_DFS(u)
devolve pred[1..n]

Visita_DFS(u)                          Vis(2)
 cor[u] ← CINZA
tempo ← tempo + 1
d(u)  ← tempo
para cada v em Adj(u) faça             v=9
       se cor[v] = BRANCO
              então pred[v] ← u
                     Visita_DFS(v)     Vis(9)
cor[u] ← PRETO
tempo ← tempo + 1
f(u)  ← tempo
```



DFS:

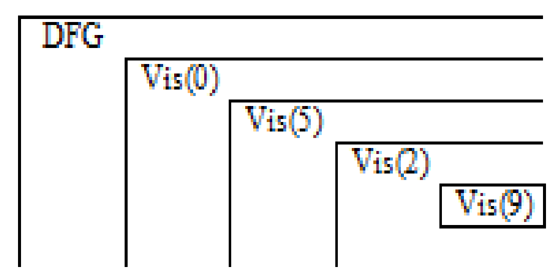
Listas de Adjacência

0 → 5 → 6		
1 → 4		
2 → 9 → 6		
3 → 7 → 8		
4 → 9		
5 → 2		
6 → 4		
7 → 8		
8		
9		




```
DFS(G)
para cada vértice u em G faça
     cor[u] = BRANCO
     pred[u] ← NIL
tempo ← 0
para cada vértice u em G faça                    u=0
       se cor(u) = BRANCO
            então Visita_DFS(u)
devolve pred[1..n]

Visita_DFS(u)                                    Vis(9)
 cor[u] ← CINZA
tempo ← tempo + 1
d(u)  ← tempo
para cada v em Adj(u) faça                       ---
       se cor[v] = BRANCO
            então pred[v] ← u
                      Visita_DFS(v)
cor[u] ← PRETO
tempo ← tempo + 1
f(u)  ← tempo
```

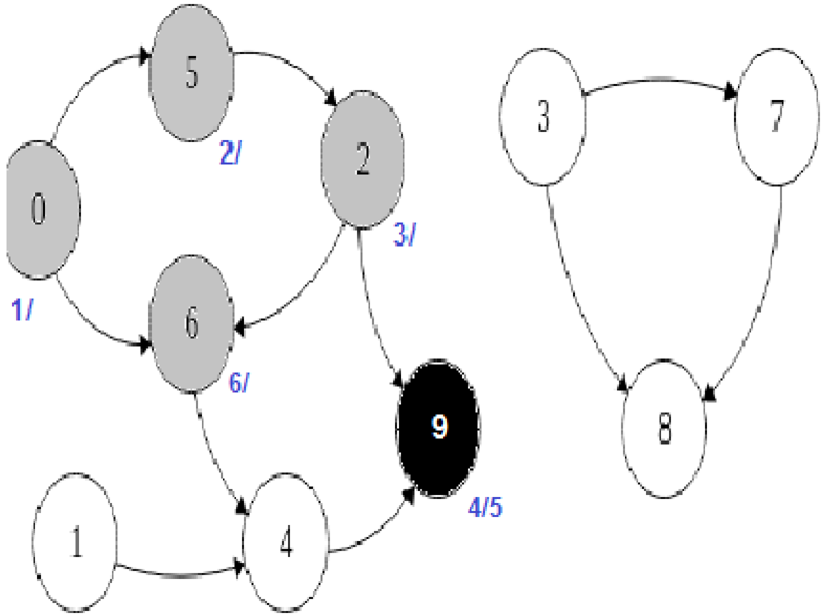
u	cor(u)	pred(u)	d(u)	f(u)
0	CZ	-	1	
1	BR	-		
2	CZ	5	3	
3	BR	-		
4	BR	-		
5	CZ	0	2	
6	BR	-		
7	BR	-		
8	BR	-		
9	PR	2	4	5



DFS:

Listas de Adjacência

- 0 → 5 → 6
- 1 → 4
- 2 → 9 → 6
- 3 → 7 → 8
- 4 → 9
- 5 → 2
- 6 → 4
- 7 → 8
- 8
- 9



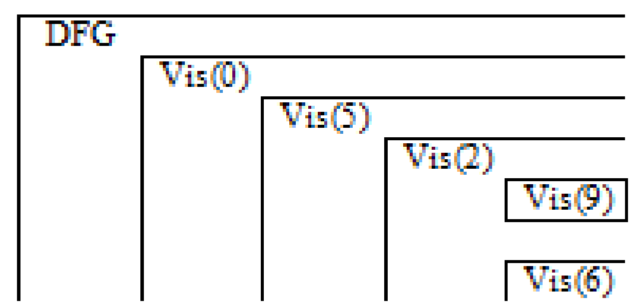
u	cor(u)	pred(u)	d(u)	f(u)
0	CZ	-	1	
1	BR	-		
2	CZ	5	3	
3	BR	-		
4	BR	-		
5	CZ	0	2	
6	CZ	2	6	
7	BR	-		
8	BR	-		
9	PR	2	4	5

DFS(G)

```
1.  para cada vértice u em G faça
2.       cor[u] = BRANCO
3.       pred[u] ← NIL
4.  tempo ← 0
5.  para cada vértice u em G faça            u=0
6.           se cor(u) = BRANCO
7.                então Visita_DFS(u)
8.  devolve pred[1..n]
```

Visita_DFS(u) Vis(6)

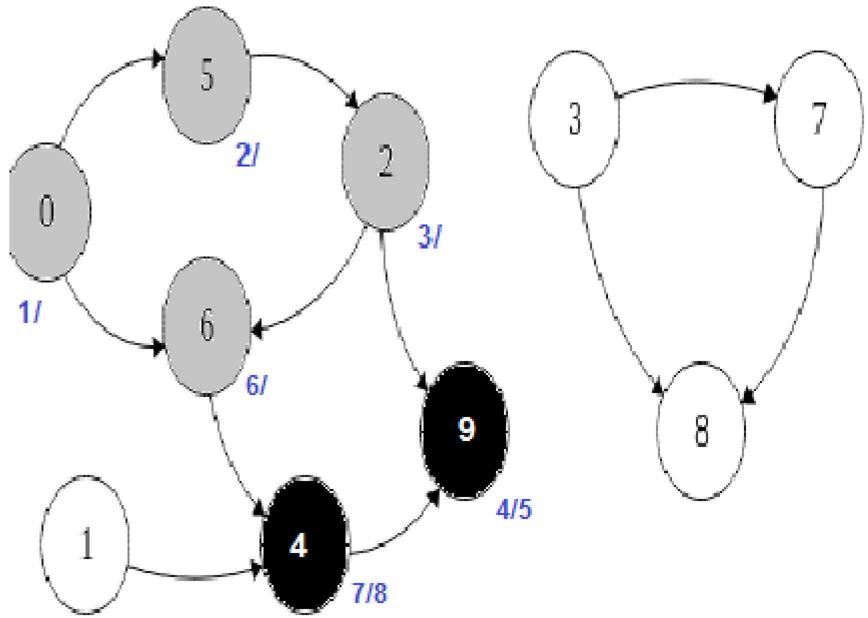
```
1.   cor[u] ← CINZA
2.  tempo ← tempo + 1
3.  d(u)  ← tempo
4.  para cada v em Adj(u) faça               v=4
5.           se cor[v] = BRANCO
6.                    então pred[v] ← u
7.                             Visita_DFS(v)  Vis(4)
7.  cor[u] ← PRETO
8.  tempo ← tempo + 1
9.  f(u)  ← tempo
```



DFS:

Listas de Adjacência

0 → 5 → 6
1 → 4
2 → 9 → 6
3 → 7 → 8
4 → 9
5 → 2
6 → 4
7 → 8
8
9



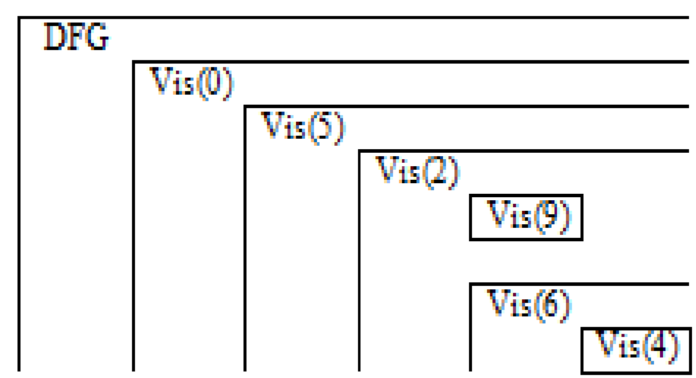
u	cor(u)	pred(u)	d(u)	f(u)
0	CZ	-	1	
1	BR	-		
2	CZ	5	3	
3	BR	-		
4	PR	6	7	8
5	CZ	0	2	
6	CZ	2	6	
7	BR	-		
8	BR	-		
9	PR	2	4	5

DFS(G)

```
para cada vértice u em G faça
     cor[u] = BRANCO
     pred[u] ← NIL
tempo ← 0
para cada vértice u em G faça                u=0
        se cor(u) = BRANCO
             então Visita_DFS(u)
devolve pred[1..n]
```

Visita_DFS(u) Vis(4)

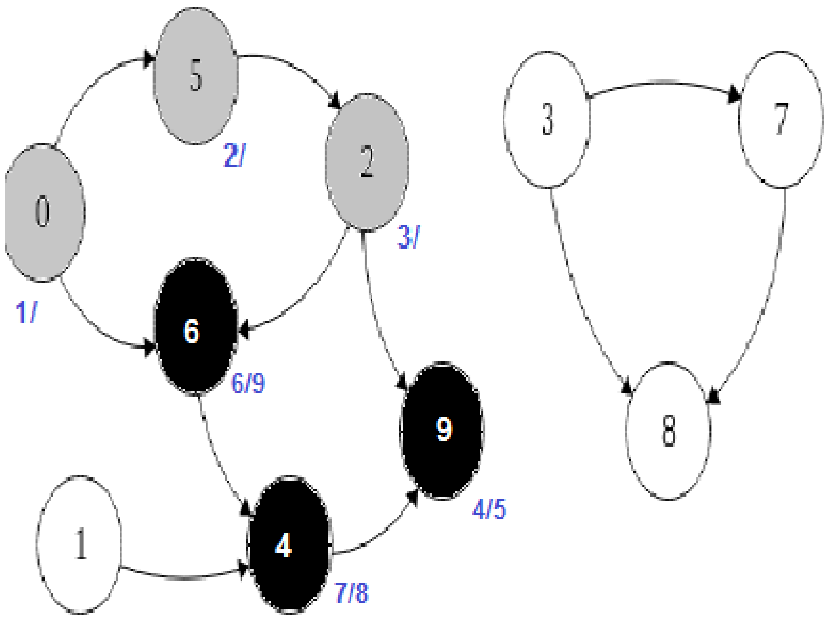
```
 cor[u] ← CINZA
tempo ← tempo + 1
d(u) ← tempo
para cada v em Adj(u) faça                 v=9
        se cor[v] = BRANCO
              então pred[v] ← u
                    Visita_DFS(v)
cor[u] ← PRETO
tempo ← tempo + 1
f(u) ← tempo
```



DFS:

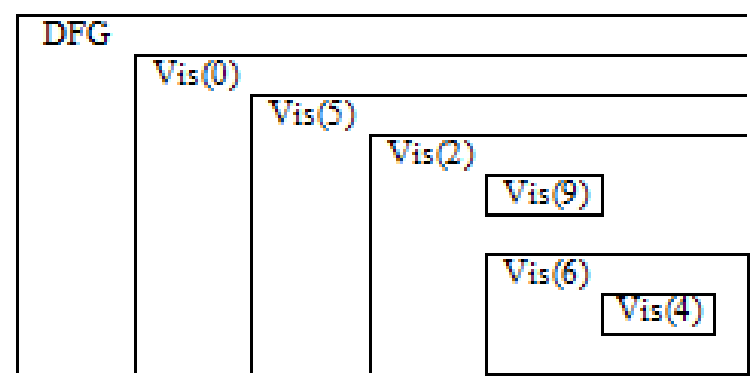
Listas de Adjacência

0 → 5 → 6
1 → 4
2 → 9 → 6
3 → 7 → 8
4 → 9
5 → 2
6 → 4
7 → 8
8
9




```
DFS(G)
 1.  para cada vértice u em G faça
 2.       cor[u] = BRANCO
 3.       pred[u] ← NIL
 4.  tempo ← 0
 5.  para cada vértice u em G faça          u=0
 6.           se cor(u) = BRANCO
 7.                então Visita_DFS(u)
 8.  devolve pred[1..n]

Visita_DFS(u)                               Vis(6)
 1.   cor[u] ← CINZA
 2.  tempo ← tempo + 1
 3.  d(u)  ← tempo
 4.  para cada v em Adj(u) faça              v=4
 5.           se cor[v] = BRANCO
 6.                então pred[v] ← u
 7.                          Visita_DFS(v)
 7.  cor[u] ← PRETO
 8.  tempo ← tempo + 1
 9.  f(u)  ← tempo
```

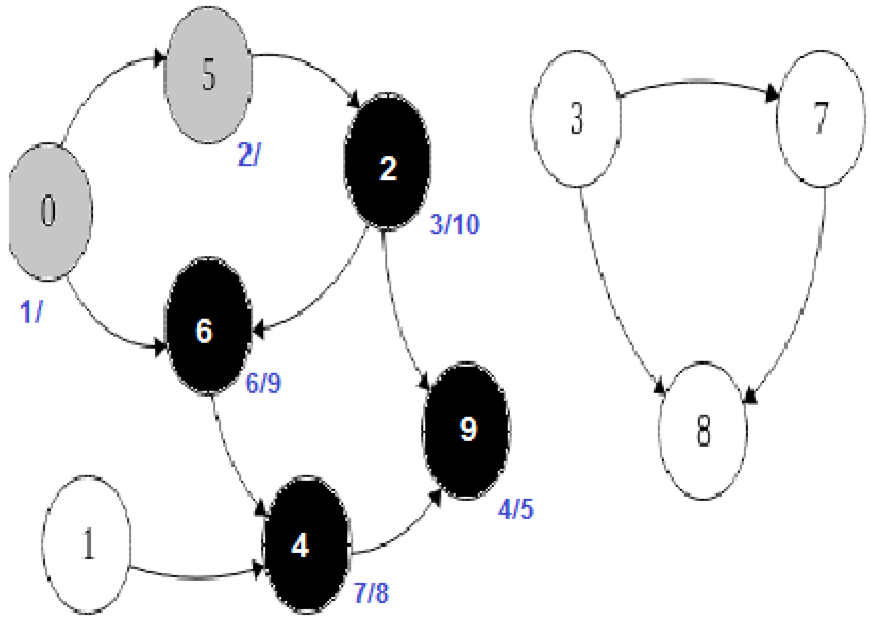
u	cor(u)	pred(u)	d(u)	f(u)
0	CZ	-	1	
1	BR	-		
2	CZ	5	3	
3	BR	-		
4	PR	6	7	8
5	CZ	0	2	
6	PR	2	6	9
7	BR	-		
8	BR	-		
9	PR	2	4	5



DFS:

Listas de Adjacência

0 → 5 → 6
1 → 4
2 → 9 → 6
3 → 7 → 8
4 → 9
5 → 2
6 → 4
7 → 8
8
9



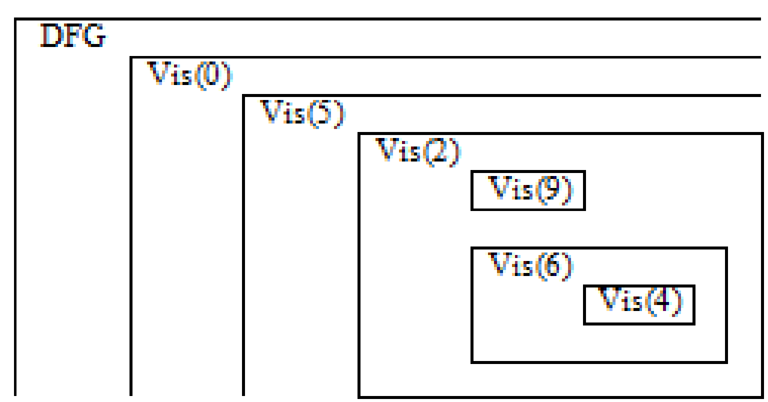
u	cor(u)	pred(u)	d(u)	f(u)
0	CZ	-	1	
1	BR	-		
2	PR	5	3	10
3	BR	-		
4	PR	6	7	8
5	CZ	0	2	
6	PR	2	6	9
7	BR	-		
8	BR	-		
9	PR	2	4	5

DFS(G)

```
para cada vértice u em G faça
    cor[u] = BRANCO
    pred[u] ← NIL
tempo ← 0
para cada vértice u em G faça                u=0
        se cor(u) = BRANCO
            então Visita_DFS(u)
devolve pred[1..n]
```

Visita_DFS(u) Vis(2)

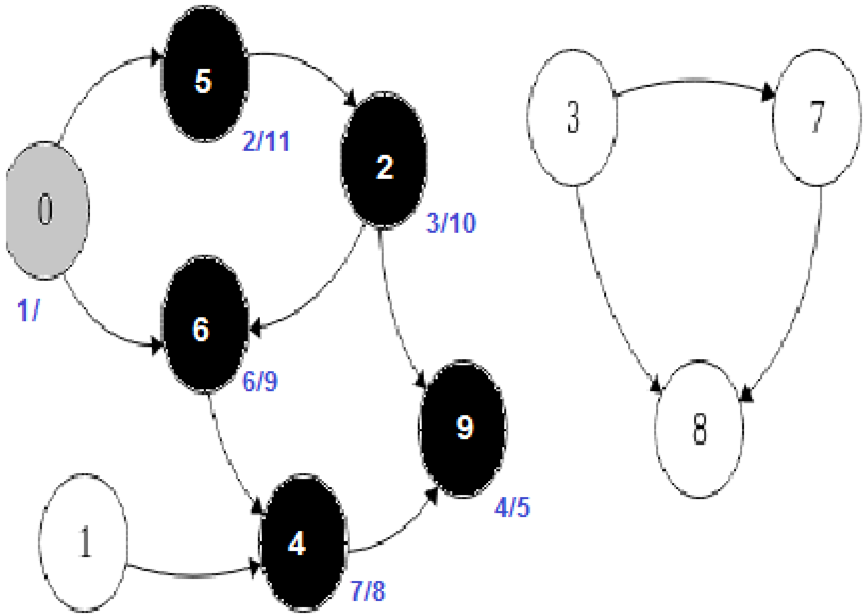
```
 cor[u] ← CINZA
tempo ← tempo + 1
d(u)  ← tempo
para cada v em Adj(u) faça                   v=6
        se cor[v] = BRANCO                   v=9
            então pred[v] ← u
                Visita_DFS(v)
cor[u] ← PRETO
tempo ← tempo + 1
f(u)  ← tempo
```



DFS:

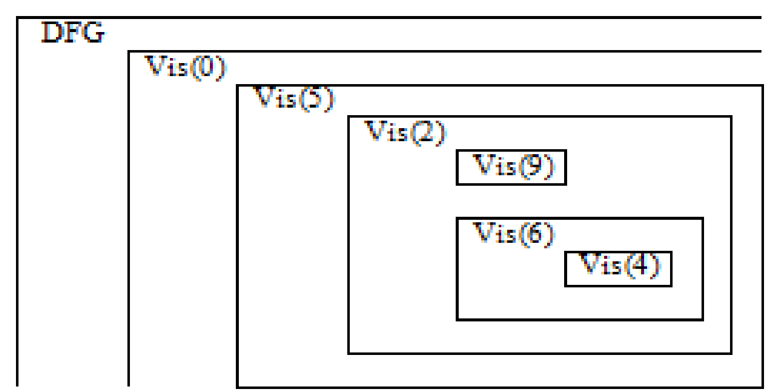
Listas de Adjacência

0 → 5 → 6
1 → 4
2 → 9 → 6
3 → 7 → 8
4 → 9
5 → 2
6 → 4
7 → 8
8
9




```
DFS(G)
para cada vértice u em G faça
     cor[u] = BRANCO
     pred[u] ← NIL
tempo ← 0
para cada vértice u em G faça          u=0
          se cor(u) = BRANCO
                 então Visita_DFS(u)
devolve pred[1..n]

Visita_DFS(u)                          Vis(5)
 cor[u] ← CINZA
tempo ← tempo + 1
d(u)  ← tempo
para cada v em Adj(u) faça             v=2
          se cor[v] = BRANCO
                então pred[v] ← u
                          Visita_DFS(v)
cor[u] ← PRETO
tempo ← tempo + 1
f(u)  ← tempo
```

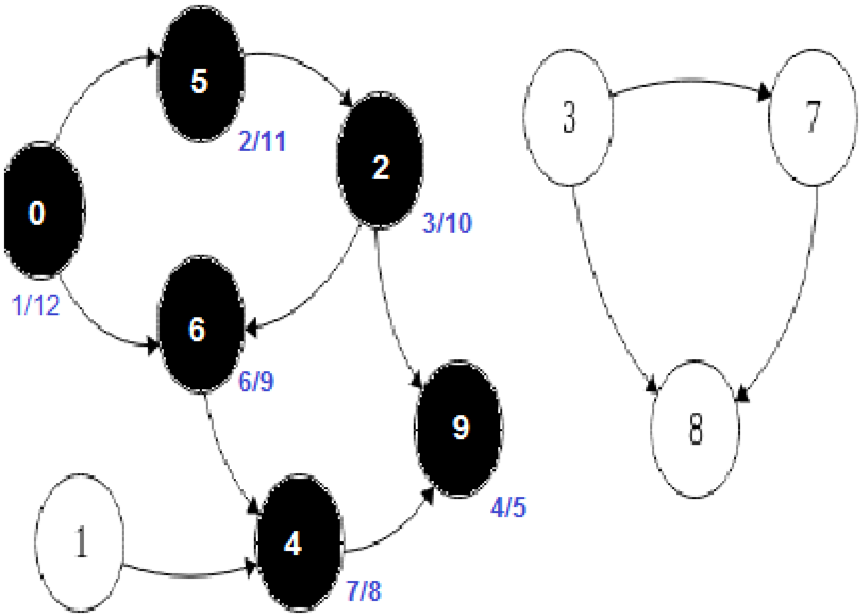
u	cor(u)	pred(u)	d(u)	f(u)
0	CZ	-	1	
1	BR	-		
2	PR	5	3	10
3	BR	-		
4	PR	6	7	8
5	PR	0	2	11
6	PR	2	6	9
7	BR	-		
8	BR	-		
9	PR	2	4	5



DFS:

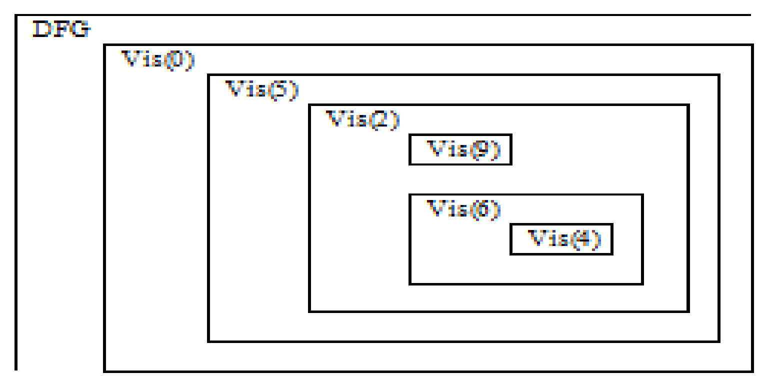
Listas de Adjacência

0 → 5 → 6
1 → 4
2 → 9 → 6
3 → 7 → 8
4 → 9
5 → 2
6 → 4
7 → 8
8
9



u	cor(u)	pred(u)	d(u)	f(u)
0	PR	-	1	12
1	BR	-		
2	PR	5	3	10
3	BR	-		
4	PR	6	7	8
5	PR	0	2	11
6	PR	2	6	9
7	BR	-		
8	BR	-		
9	PR	2	4	5

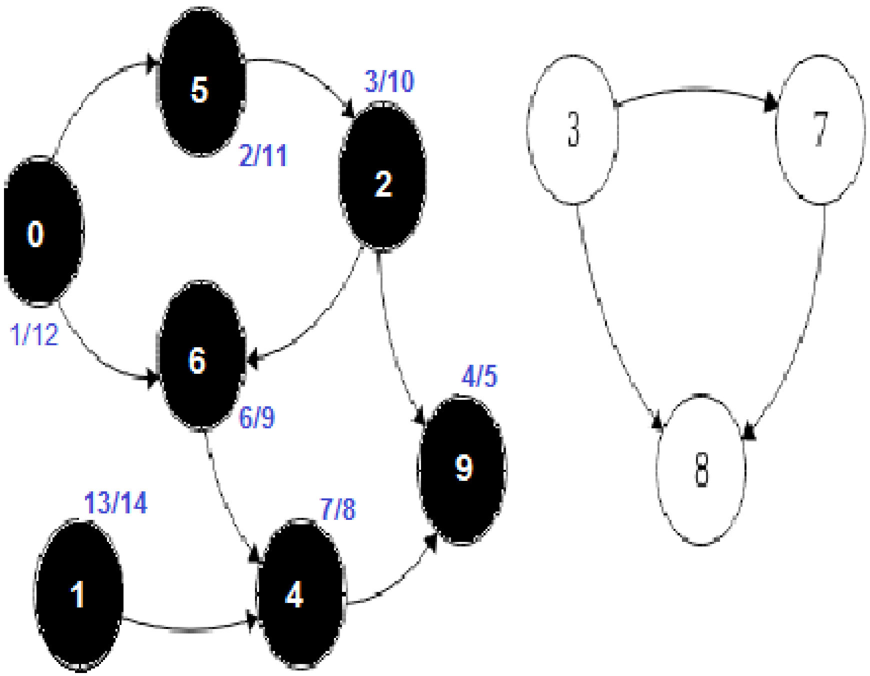
```
DFS(G)
   para cada vértice u em G faça
       cor[u] = BRANCO
       pred[u] ← NIL
   tempo ← 0
   para cada vértice u em G faça          u=0
           se cor(u) = BRANCO
               então Visita_DFS(u)
   devolve pred[1..n]

Visita_DFS(u)                              Vis(0)
   cor[u] ← CINZA
   tempo ← tempo + 1
   d(u) ← tempo
   para cada v em Adj(u) faça             v=5
           se cor[v] = BRANCO
               então pred[v] ← u
                     Visita_DFS(v)
   cor[u] ← PRETO
   tempo ← tempo + 1
   f(u) ← tempo
```



DFS:

Listas de Adjacência

0 → 5 → 6
1 → 4
2 → 9 → 6
3 → 7 → 8
4 → 9
5 → 2
6 → 4
7 → 8
8
9



DFS(G)

```
para cada vértice u em G faça
    cor[u] = BRANCO
    pred[u] ← NIL
tempo ← 0
para cada vértice u em G faça
    se cor(u) = BRANCO
        então Visita_DFS(u)
devolve pred[1..n]
```

u = 1

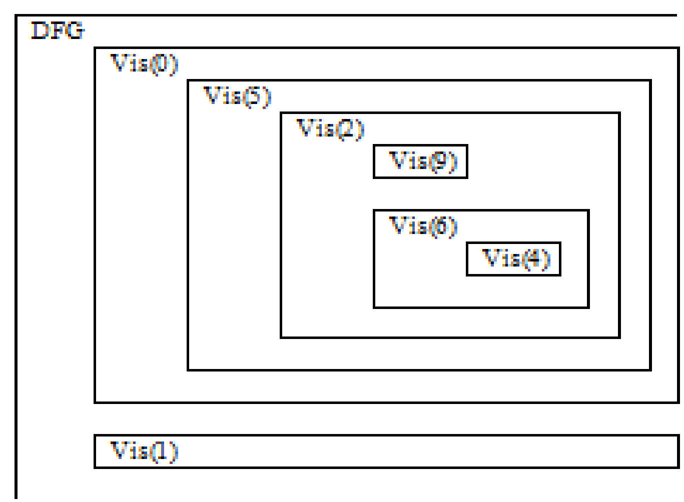
Visita_DFS(u)

Vis(1)

```
cor[u] ← CINZA
tempo ← tempo + 1
d(u) ← tempo
para cada v em Adj(u) faça
    se cor[v] = BRANCO
        então pred[v] ← u
            Visita_DFS(v)
cor[u] ← PRETO
tempo ← tempo + 1
f(u) ← tempo
```

v=4

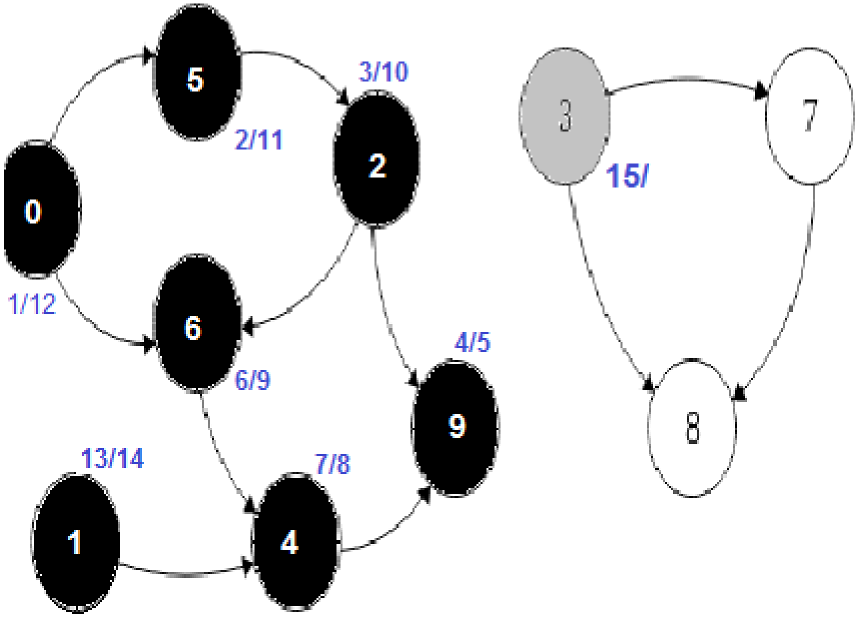
u	cor(u)	pred(u)	d(u)	f(u)
0	PR	-	1	12
1	PR	-	13	14
2	PR	5	3	10
3	BR	-		
4	PR	6	7	8
5	PR	0	2	11
6	PR	2	6	9
7	BR	-		
8	BR	-		
9	PR	2	4	5



DFS:

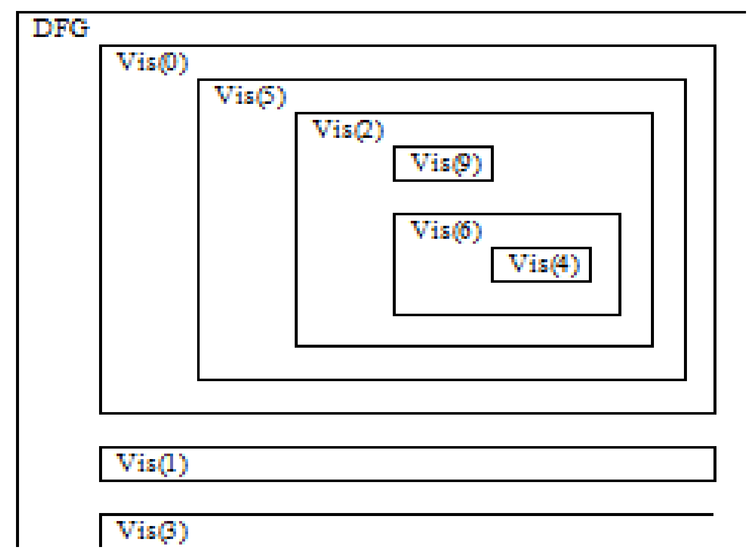
Listas de Adjacência

0 → 5 → 6
1 → 4
2 → 9 → 6
3 → 7 → 8
4 → 9
5 → 2
6 → 4
7 → 8
8
9




```
DFS(G)
 1.  para cada vértice u em G faça
 2.      cor[u] = BRANCO
 3.      pred[u] ← NIL
 4.  tempo ← 0
 5.  para cada vértice u em G faça        u=3
 6.          se cor(u) = BRANCO
 7.              então Visita_DFS(u)
 8.  devolve pred[1..n]

Visita_DFS(u)                             Vis(3)
 1.   cor[u] ← CINZA
 2.  tempo ← tempo + 1
 3.  d(u)  ← tempo
 4.  para cada v em Adj(u) faça           v=7
 5.          se cor[v] = BRANCO
 6.                 então pred[v] ← u
 7.                       Visita_DFS(v)   Vis(7)
 7.  cor[u] ← PRETO
 8.  tempo ← tempo + 1
 9.  f(u)  ← tempo
```

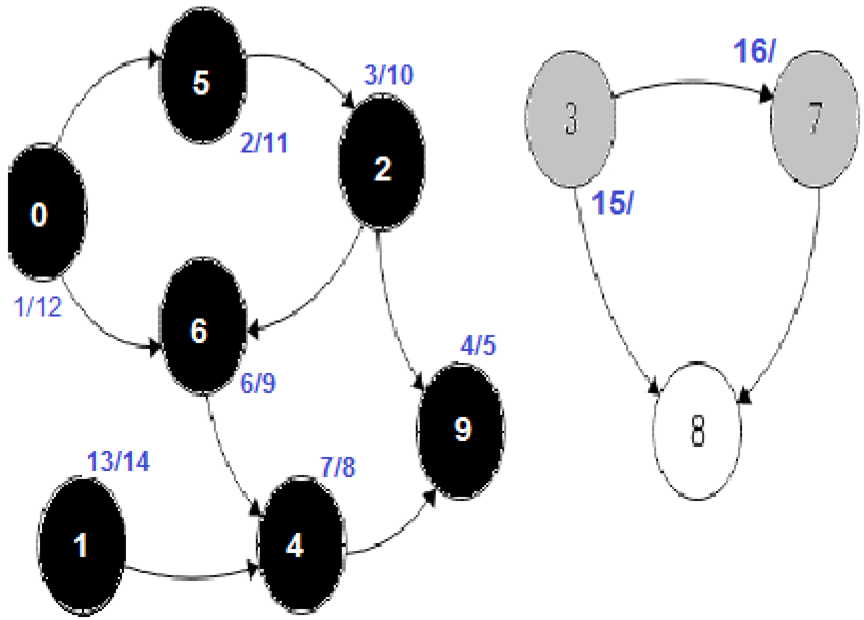
u	cor(u)	pred(u)	d(u)	f(u)
0	PR	-	1	12
1	PR	-	13	14
2	PR	5	3	10
3	CZ	-	15	
4	PR	6	7	8
5	PR	0	2	11
6	PR	2	6	9
7	BR	-		
8	BR	-		
9	PR	2	4	5



DFS:

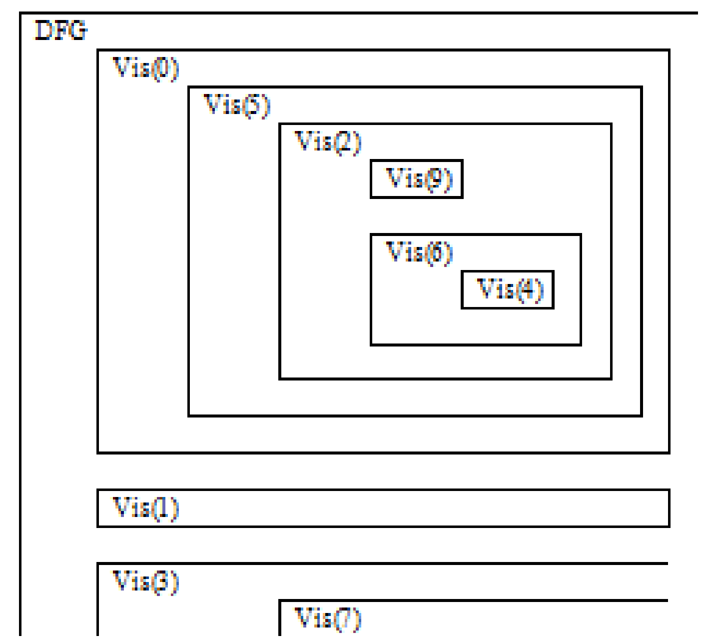
Listas de Adjacência

0 → 5 → 6
1 → 4
2 → 9 → 6
3 → 7 → 8
4 → 9
5 → 2
6 → 4
7 → 8
8
9




```
DFS(G)
   para cada vértice u em G faça
      cor[u] = BRANCO
      pred[u] ← NIL
   tempo ← 0
   para cada vértice u em G faça                u=3
         se cor(u) = BRANCO
            então Visita_DFS(u)
   devolve pred[1..n]

Visita_DFS(u)                                   Vis(7)
   cor[u] ← CINZA
   tempo ← tempo + 1
   d(u) ← tempo
   para cada v em Adj(u) faça
         se cor[v] = BRANCO                     v=8
               então pred[v] ← u
                     Visita_DFS(v)
   cor[u] ← PRETO
   tempo ← tempo + 1
   f(u) ← tempo
```

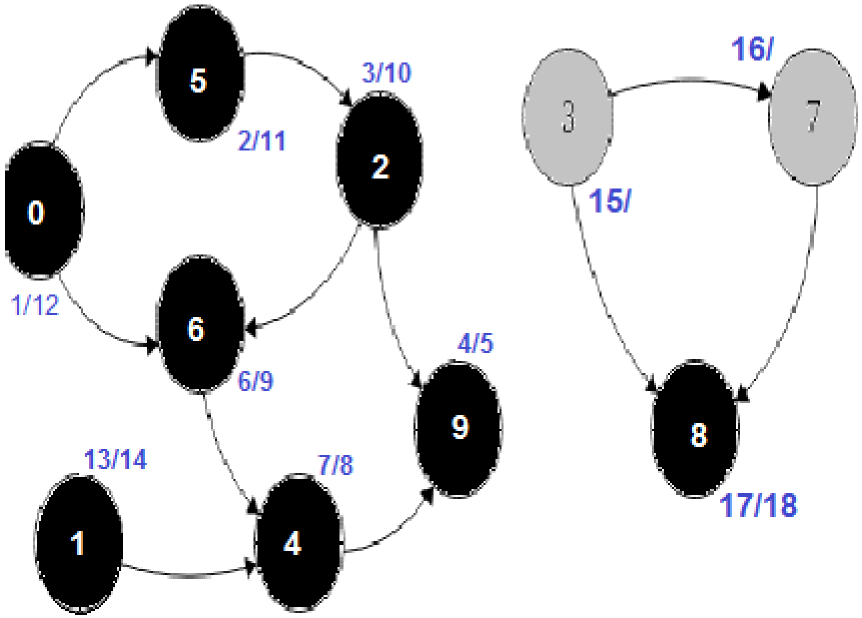
u	cor(u)	pred(u)	d(u)	f(u)
0	PR	-	1	12
1	PR	-	13	14
2	PR	5	3	10
3	CZ	-	15	
4	PR	6	7	8
5	PR	0	2	11
6	PR	2	6	9
7	CZ	3	16	
8	BR	-		
9	PR	2	4	5



DFS:

Listas de Adjacência

0 → 5 → 6
1 → 4
2 → 9 → 6
3 → 7 → 8
4 → 9
5 → 2
6 → 4
7 → 8
8
9



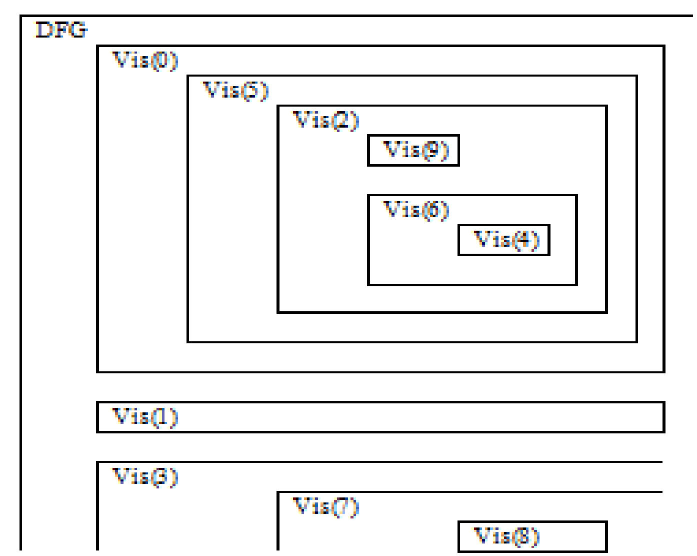

```
DFS(G)
  para cada vértice u em G faça
        cor[u] = BRANCO
        pred[u] ← NIL
  tempo ← 0
  para cada vértice u em G faça
          se cor(u) = BRANCO
                então Visita_DFS(u)
  devolve pred[1..n]
```

u=3

```
Visita_DFS(u)
   cor[u] ← CINZA
  tempo ← tempo + 1
  d(u)  ← tempo
  para cada v em Adj(u) faça
          se cor[v] = BRANCO
                então pred[v] ← u
                        Visita_DFS(v)
  cor[u] ← PRETO
  tempo ← tempo + 1
  f(u)  ← tempo
```

Vis(8)

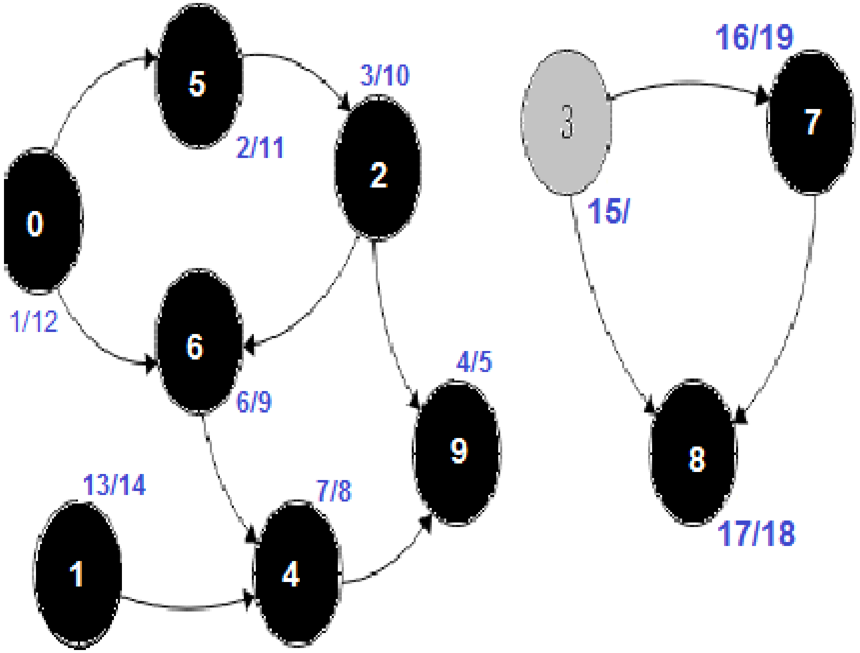
u	cor(u)	pred(u)	d(u)	f(u)
0	PR	-	1	12
1	PR	-	13	14
2	PR	5	3	10
3	CZ	-	15	
4	PR	6	7	8
5	PR	0	2	11
6	PR	2	6	9
7	CZ	3	16	
8	PR	7	17	18
9	PR	2	4	5



DFS:

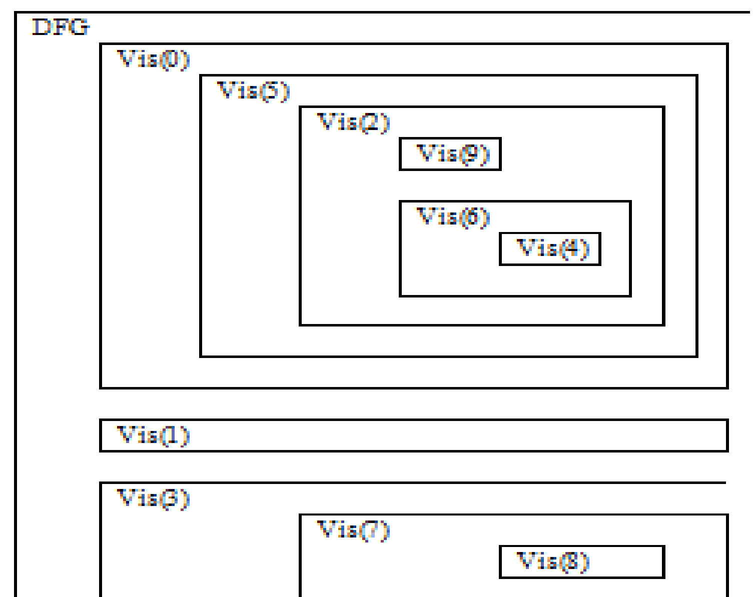
Listas de Adjacência

0 → 5 → 6
1 → 4
2 → 9 → 6
3 → 7 → 8
4 → 9
5 → 2
6 → 4
7 → 8
8
9




```
DFS(G)
   para cada vértice u em G faça
        cor[u] = BRANCO
        pred[u] ← NIL
   tempo ← 0
   para cada vértice u em G faça              u=3
        se cor(u) = BRANCO
             então Visita_DFS(u)
   devolve pred[1..n]

Visita_DFS(u)                                 Vis(7)
   cor[u] ← CINZA
   tempo ← tempo + 1
   d(u) ← tempo
   para cada v em Adj(u) faça
        se cor[v] = BRANCO                    v=8
             então pred[v] ← u
                   Visita_DFS(v)
   cor[u] ← PRETO
   tempo ← tempo + 1
   f(u) ← tempo
```

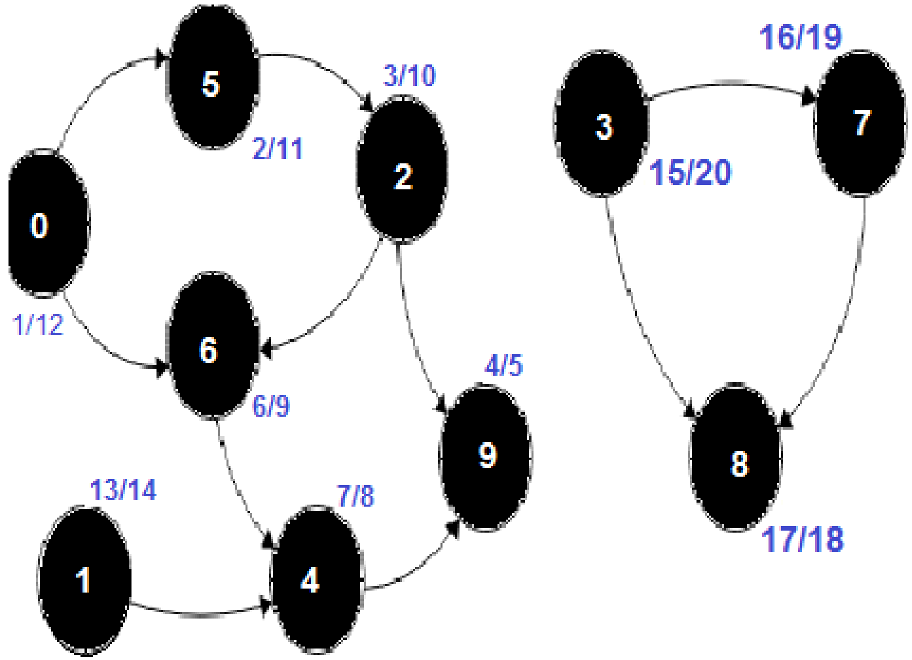
u	cor(u)	pred(u)	d(u)	f(u)
0	PR	-	1	12
1	PR	-	13	14
2	PR	5	3	10
3	CZ	-	15	
4	PR	6	7	8
5	PR	0	2	11
6	PR	2	6	9
7	PR	3	16	19
8	PR	7	17	18
9	PR	2	4	5



DFS:

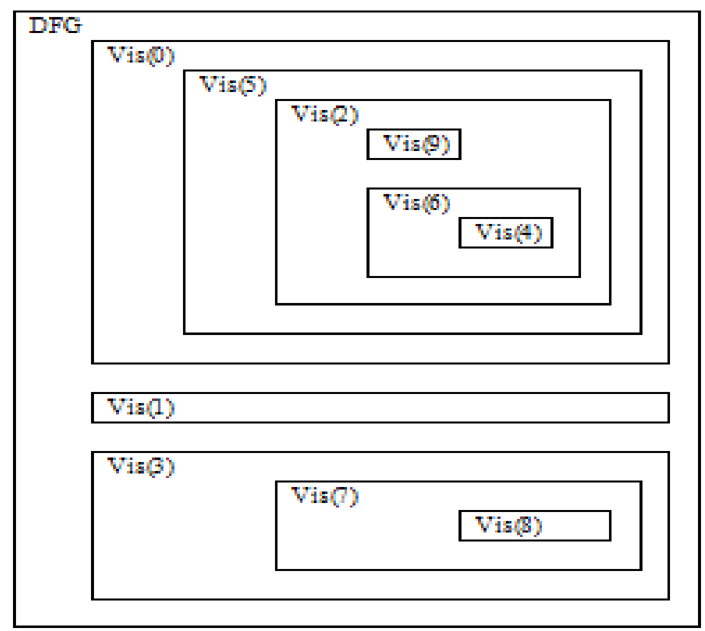
Listas de Adjacência

0 → 5 → 6
1 → 4
2 → 9 → 6
3 → 7 → 8
4 → 9
5 → 2
6 → 4
7 → 8
8
9



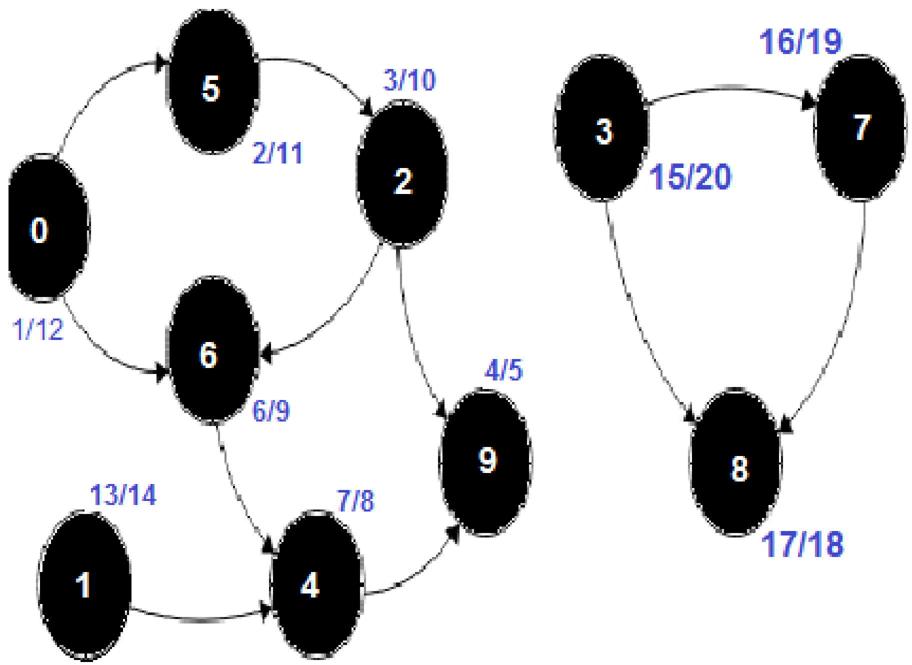

```
DFS(G)
1.  para cada vértice u em G faça
2.      cor[u] = BRANCO
3.      pred[u] ← NIL
4.  tempo ← 0
5.  para cada vértice u em G faça          u=3
6.      se cor(u) = BRANCO
7.          então Visita_DFS(u)
8.  devolve pred[1..n]

Visita_DFS(u)                              Vis(3)
1.  cor[u] ← CINZA
2.  tempo ← tempo + 1
3.  d(u) ← tempo
4.  para cada v em Adj(u) faça
5.      se cor[v] = BRANCO                 v=8
6.          então pred[v] ← u              v=7
7.              Visita_DFS(v)
7.  cor[u] ← PRETO
8.  tempo ← tempo + 1
9.  f(u) ← tempo
```

u	cor(u)	pred(u)	d(u)	f(u)
0	PR	-	1	12
1	PR	-	13	14
2	PR	5	3	10
3	PR	-	15	20
4	PR	6	7	8
5	PR	0	2	11
6	PR	2	6	9
7	PR	3	16	19
8	PR	7	17	18
9	PR	2	4	5



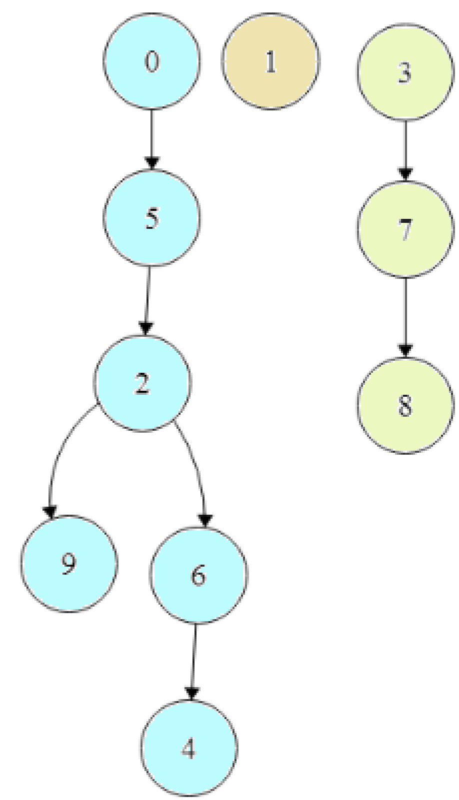
Listas de Adjacência

0 → 5 → 6
1 → 4
2 → 9 → 6
3 → 7 → 8
4 → 9
5 → 2
6 → 4
7 → 8
8
9



u	cor(u)	pred(u)	d(u)	f(u)
0	PR	-	1	12
1	PR	-	13	14
2	PR	5	3	10
3	PR	-	15	20
4	PR	6	7	8
5	PR	0	2	11
6	PR	2	6	9
7	PR	3	16	19
8	PR	7	17	18
9	PR	2	4	5

Arborescências



(dígrafo tem 3 componentes)

+ + exemplo DFS

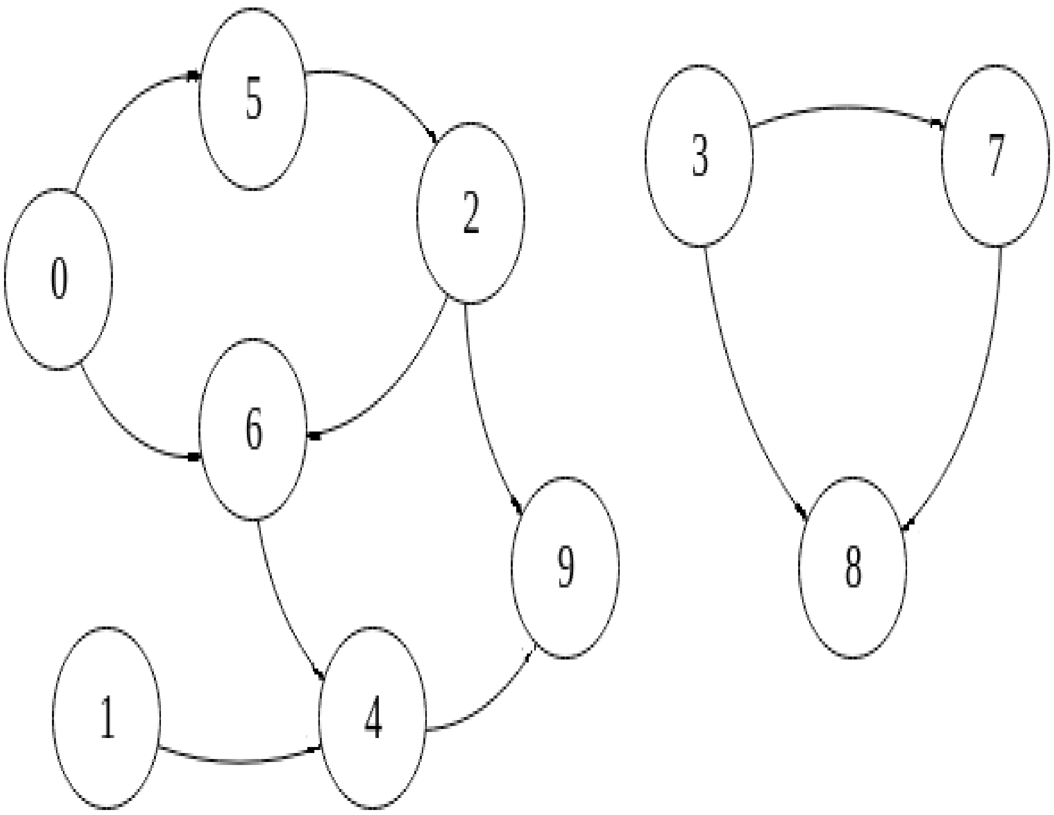
Aplique a **busca em profundidade,** ao *grafo* definido pelas arestas (considerando essa ordem de inclusão na construção do grafo):

3-7 1-4 7-8 0-5 5-2 3-8 2-9 0-6 4-9 2-6 6-4,

Considerando sua representação em matriz de adjacências.

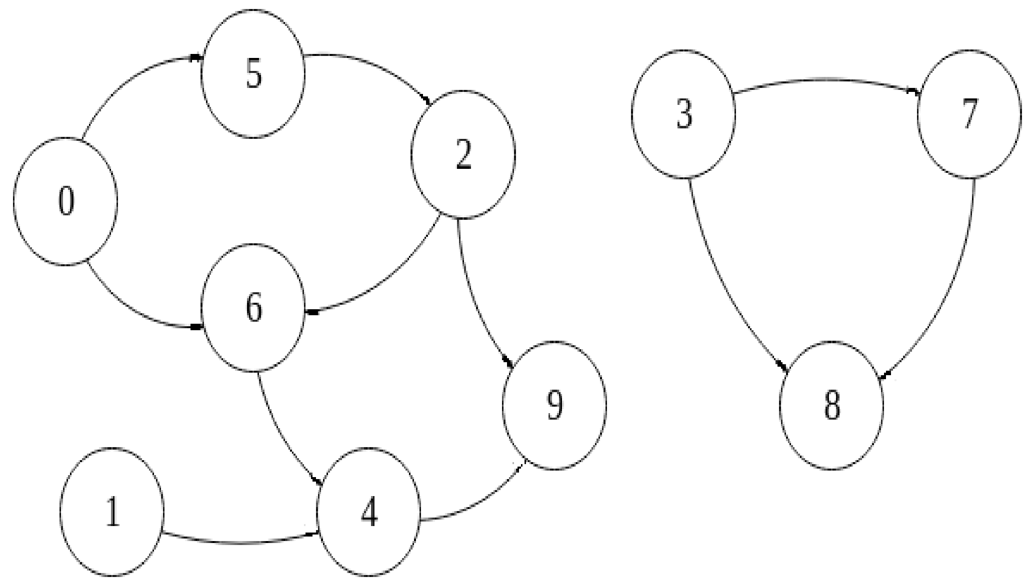
Matriz de adjacências   e, grafo correspondente

Adj	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	1	1	0	0	0
1	0	0	0	0	1	0	0	0	0	0
2	0	0	0	0	0	1	1	0	0	1
3	0	0	0	0	0	0	0	1	1	0
4	0	1	0	0	0	0	0	0	0	1
5	1	0	1	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0
7	0	0	0	1	0	0	0	0	1	0
8	0	0	0	1	0	0	0	1	0	0
9	0	0	1	0	1	0	0	0	0	0



DFS

Adj	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	1	1	0	0	0
1	0	0	0	0	1	0	0	0	0	0
2	0	0	0	0	0	1	1	0	0	1
3	0	0	0	0	0	0	0	1	1	0
4	0	1	0	0	0	0	0	0	0	1
5	1	0	1	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0
7	0	0	0	1	0	0	0	0	1	0
8	0	0	0	1	0	0	0	1	0	0
9	0	0	1	0	1	0	0	0	0	0



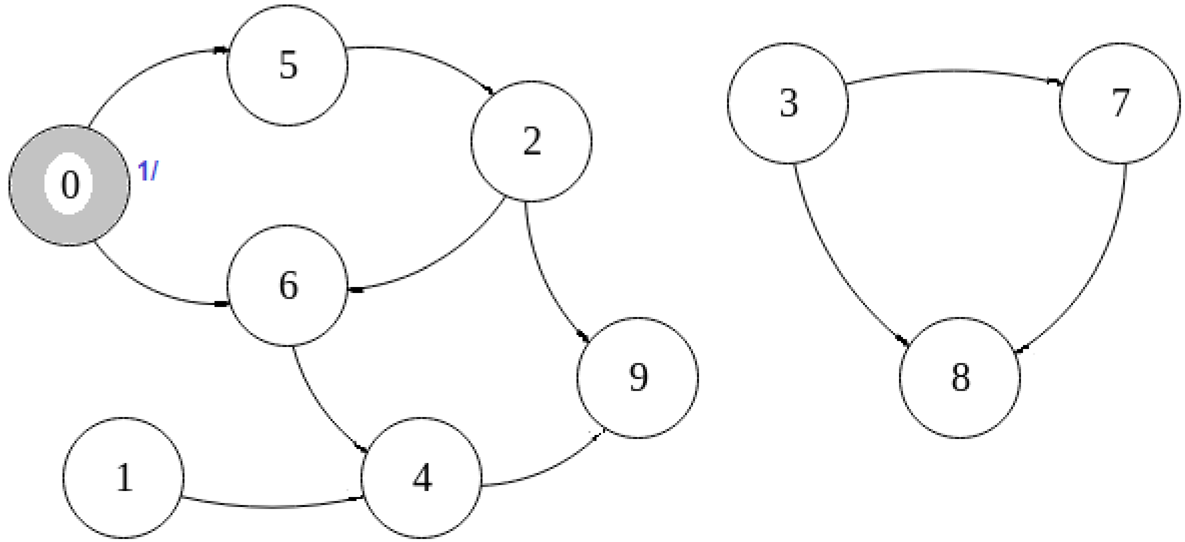

```
DFS(G)
para cada vértice u em G faça
    cor[u] = BRANCO
    pred[u] ← NIL
tempo ← 0
para cada vértice u em G faça
        se cor(u) = BRANCO
            então Visita_DFS(u)
devolve pred[1..n]

Visita_DFS(u)
 cor[u] ← CINZA
tempo ← tempo + 1
d(u)  ← tempo
para cada v em Adj(u) faça
        se cor[v] = BRANCO
                então pred[v] ← u
                        Visita_DFS(v)
cor[u] ← PRETO
tempo ← tempo + 1
f(u)  ← tempo
```

u	cor(u)	pred(u)	d(u)	f(u)
0	BR	-		
1	BR	-		
2	BR	-		
3	BR	-		
4	BR	-		
5	BR	-		
6	BR	-		
7	BR	-		
8	BR	-		
9	BR	-		

DFG

DFS

Adj	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	1	1	0	0	0
1	0	0	0	0	1	0	0	0	0	0
2	0	0	0	0	0	1	1	0	0	1
3	0	0	0	0	0	0	0	1	1	0
4	0	1	0	0	0	0	0	0	0	1
5	1	0	1	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0
7	0	0	0	1	0	0	0	0	1	0
8	0	0	0	1	0	0	0	1	0	0
9	0	0	1	0	1	0	0	0	0	0



DFS(G)

```
1.  para cada vértice u em G faça
2.      cor[u] = BRANCO
3.      pred[u] ← NIL
4.  tempo ← 0
5.  para cada vértice u em G faça
6.          se cor(u) = BRANCO
7.              então Visita_DFS(u)
8.  devolve pred[1..n]
```

Visita_DFS(u)

```
1.   cor[u] ← CINZA
2.  tempo ← tempo + 1
3.  d(u)  ← tempo
4.  para cada v em Adj(u) faça
5.          se cor[v] = BRANCO
6.              então pred[v] ← u
7.                  Visita_DFS(v)
7.  cor[u] ← PRETO
8.  tempo ← tempo + 1
9.  f(u)  ← tempo
```

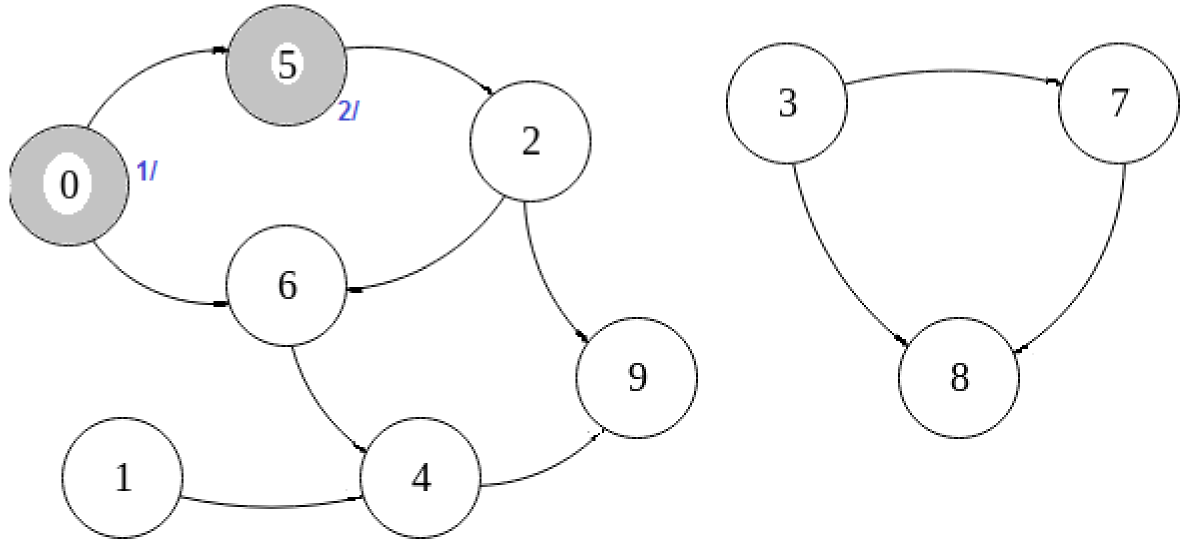
u	cor(u)	pred(u)	d(u)	f(u)
0	CZ	-	1	
1	BR	-		
2	BR	-		
3	BR	-		
4	BR	-		
5	BR	-		
6	BR	-		
7	BR	-		
8	BR	-		
9	BR	-		

DFG

Vis(0)

DFS

Adj	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	1	1	0	0	0
1	0	0	0	0	1	0	0	0	0	0
2	0	0	0	0	0	1	1	0	0	1
3	0	0	0	0	0	0	0	1	1	0
4	0	1	0	0	0	0	0	0	0	1
5	1	0	1	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0
7	0	0	0	1	0	0	0	0	1	0
8	0	0	0	1	0	0	0	1	0	0
9	0	0	1	0	1	0	0	0	0	0

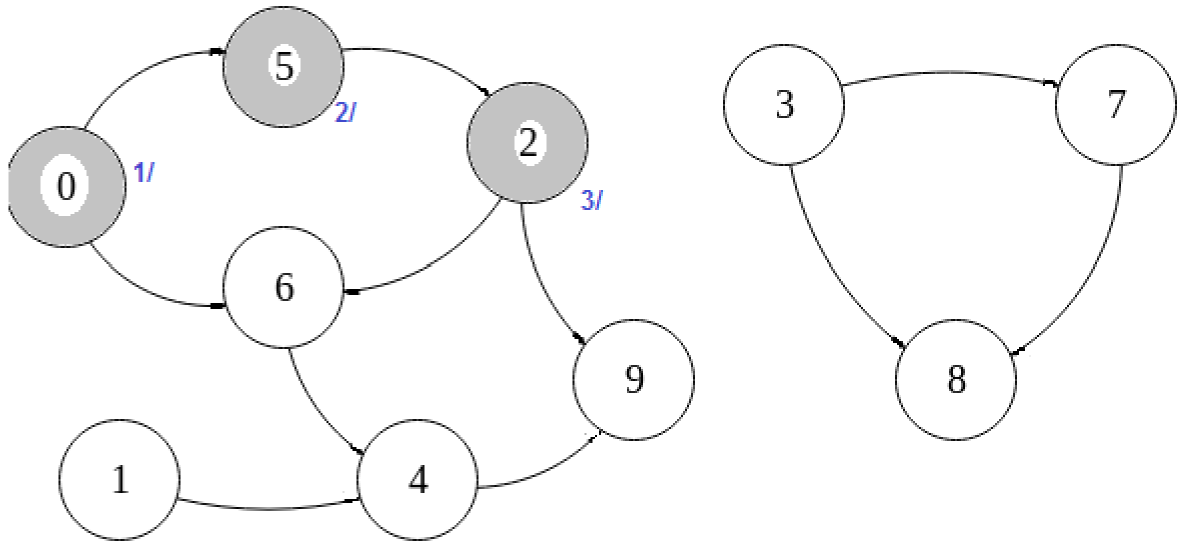



```
DFS(G)
1.  para cada vértice u em G faça
2.      cor[u] = BRANCO
3.      pred[u] ← NIL
4.  tempo ← 0
5.  para cada vértice u em G faça
6.          se cor(u) = BRANCO
7.              então Visita_DFS(u)
8.  devolve pred[1..n]

Visita_DFS(u)
1.   cor[u] ← CINZA
2.  tempo ← tempo + 1
3.  d(u)  ← tempo
4.  para cada v em Adj(u) faça
5.          se cor[v] = BRANCO
6.                  então pred[v] ← u
7.                          Visita_DFS(v)
7.  cor[u] ← PRETO
8.  tempo ← tempo + 1
9.  f(u)  ← tempo
```

u	cor(u)	pred(u)	d(u)	f(u)
0	CZ	-	1	
1	BR	-		
2	BR	-		
3	BR	-		
4	BR	-		
5	CZ	0	2	
6	BR	-		
7	BR	-		
8	BR	-		
9	BR	-		

DFS

Adj	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	1	1	0	0	0
1	0	0	0	0	1	0	0	0	0	0
2	0	0	0	0	0	1	1	0	0	1
3	0	0	0	0	0	0	0	1	1	0
4	0	1	0	0	0	0	0	0	0	1
5	1	0	1	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0
7	0	0	0	1	0	0	0	0	1	0
8	0	0	0	1	0	0	0	1	0	0
9	0	0	1	0	1	0	0	0	0	0

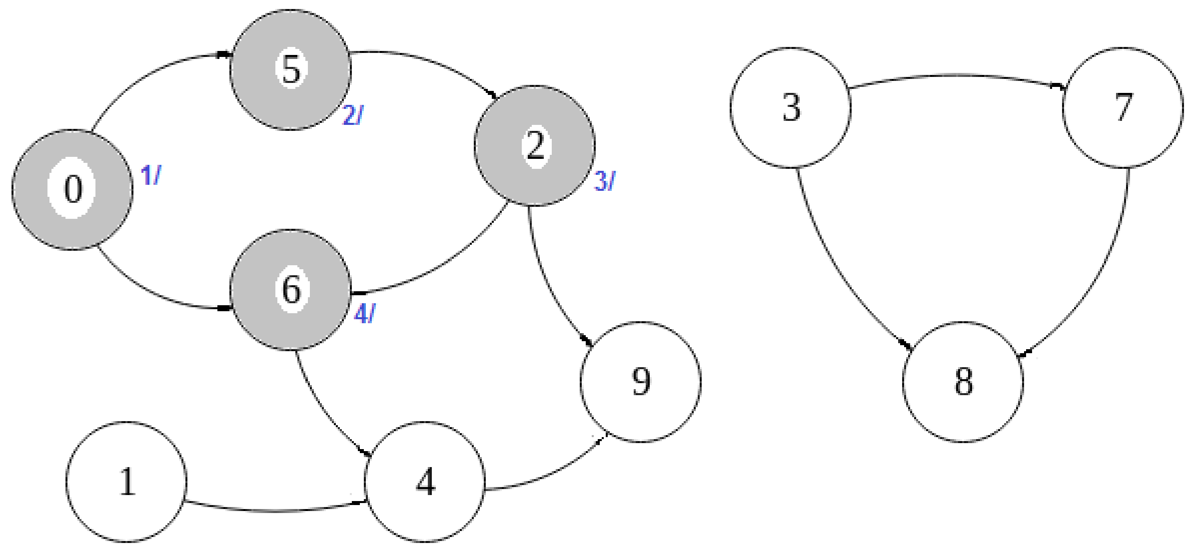



```
DFS(G)
para cada vértice u em G faça
     cor[u] = BRANCO
     pred[u] ← NIL
tempo ← 0
para cada vértice u em G faça
        se cor(u) = BRANCO
             então Visita_DFS(u)
devolve pred[1..n]

Visita_DFS(u)
 cor[u] ← CINZA
tempo ← tempo + 1
d(u)  ← tempo
para cada v em Adj(u) faça
        se cor[v] = BRANCO
                então pred[v] ← u
                        Visita_DFS(v)
cor[u] ← PRETO
tempo ← tempo + 1
f(u)  ← tempo
```

u	cor(u)	pred(u)	d(u)	f(u)
0	CZ	-	1	
1	BR	-		
2	CZ	5	3	
3	BR	-		
4	BR	-		
5	CZ	0	2	
6	BR	-		
7	BR	-		
8	BR	-		
9	BR	-		

DFS

Adj	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	1	1	0	0	0
1	0	0	0	0	1	0	0	0	0	0
2	0	0	0	0	0	1	1	0	0	1
3	0	0	0	0	0	0	0	1	1	0
4	0	1	0	0	0	0	0	0	0	1
5	1	0	1	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0
7	0	0	0	1	0	0	0	0	1	0
8	0	0	0	1	0	0	0	1	0	0
9	0	0	1	0	1	0	0	0	0	0



DFS(G)

```
para cada vértice u em G faça
    cor[u] = BRANCO
    pred[u] ← NIL
tempo ← 0
para cada vértice u em G faça
        se cor(u) = BRANCO
            então Visita_DFS(u)
devolve pred[1..n]
```

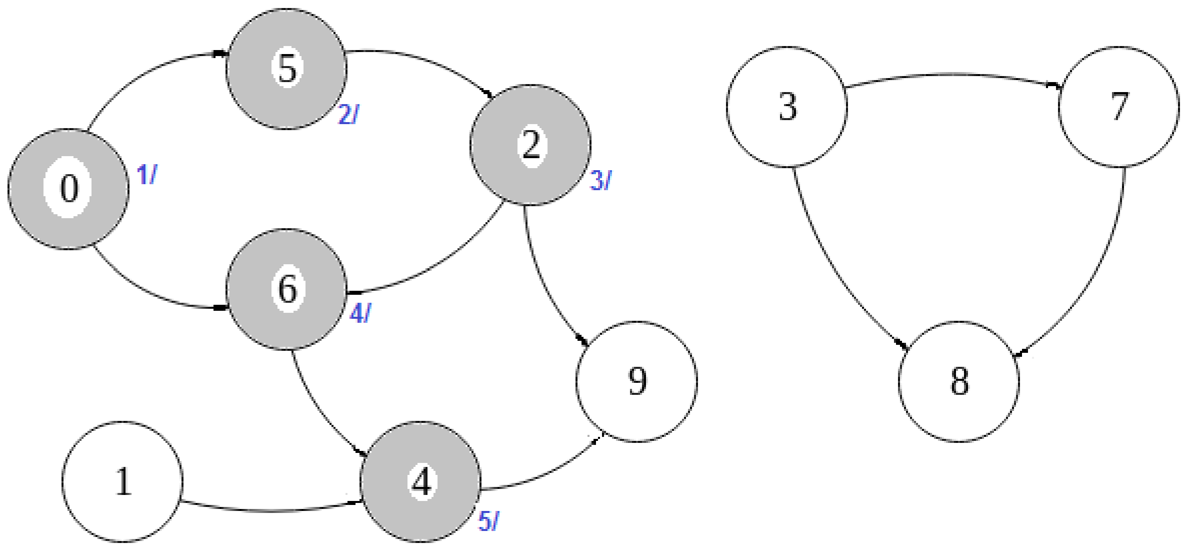
Visita_DFS(u)

```
 cor[u] ← CINZA
tempo ← tempo + 1
d(u)  ← tempo
para cada v em Adj(u) faça
        se cor[v] = BRANCO
                então pred[v] ← u
                        Visita_DFS(v)
cor[u] ← PRETO
tempo ← tempo + 1
f(u)  ← tempo
```

u	cor(u)	pred(u)	d(u)	f(u)
0	CZ	-	1	
1	BR	-		
2	CZ	5	3	
3	BR	-		
4	BR	-		
5	CZ	0	2	
6	CZ	2	4	
7	BR	-		
8	BR	-		
9	BR	-		

DFS

Adj	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	0	0	0	0	0	1	1	0	0	0
1	0	0	0	0	1	0	0	0	0	0
2	0	0	0	0	0	1	1	0	0	1
3	0	0	0	0	0	0	0	1	1	0
4	0	1	0	0	0	0	0	0	0	1
5	1	0	1	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0
7	0	0	0	1	0	0	0	0	1	0
8	0	0	0	1	0	0	0	1	0	0
9	0	0	1	0	1	0	0	0	0	0



DFS(G)

```
para cada vértice u em G faça
      cor[u] = BRANCO
      pred[u] ← NIL
tempo ← 0
para cada vértice u em G faça
          se cor(u) = BRANCO
              então Visita_DFS(u)
devolve pred[1..n]
```

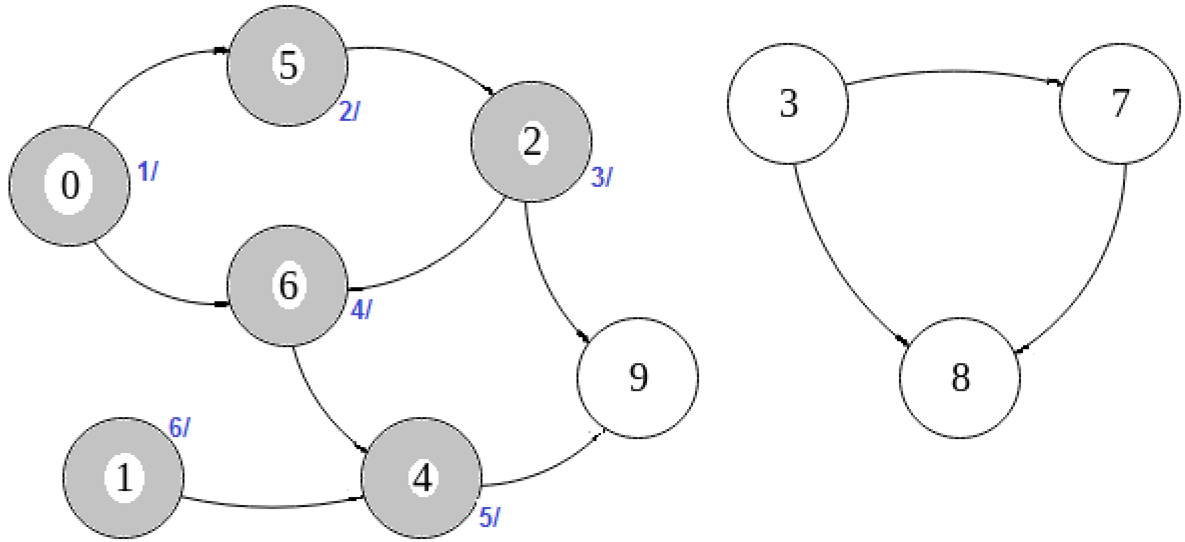
Visita_DFS(u)

```
 cor[u] ← CINZA
tempo ← tempo + 1
d(u)  ← tempo
para cada v em Adj(u) faça
          se cor[v] = BRANCO
                   então pred[v] ← u
                             Visita_DFS(v)
cor[u] ← PRETO
tempo ← tempo + 1
f(u)  ← tempo
```

u	cor(u)	pred(u)	d(u)	f(u)
0	CZ	-	1	
1	BR	-		
2	CZ	5	3	
3	BR	-		
4	CZ	6	5	
5	CZ	0	2	
6	CZ	2	4	
7	BR	-		
8	BR	-		
9	BR	-		

DFS

Adj	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	1	1	0	0	0
1	0	0	0	0	1	0	0	0	0	0
2	0	0	0	0	0	1	1	0	0	1
3	0	0	0	0	0	0	0	1	1	0
4	0	1	0	0	0	0	0	0	0	1
5	1	0	1	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0
7	0	0	0	1	0	0	0	0	1	0
8	0	0	0	1	0	0	0	1	0	0
9	0	0	1	0	1	0	0	0	0	0



DFS(G)

1. para cada vértice u em G faça
2. cor[u] = BRANCO
3. pred[u] ← NIL
4. tempo ← 0
5. para cada vértice u em G faça
6. se cor(u) = BRANCO
7. então Visita_DFS(u)
8. devolve pred[1..n]

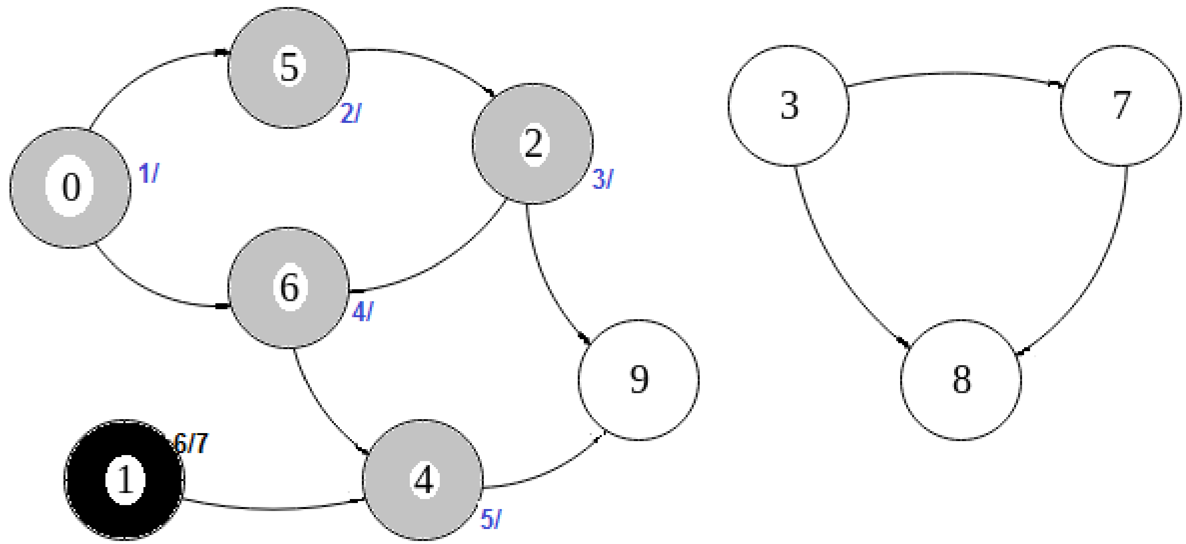
Visita_DFS(u)

1. cor[u] ← CINZA
2. tempo ← tempo + 1
3. d(u) ← tempo
4. para cada v em Adj(u) faça
5. se cor[v] = BRANCO
6. então pred[v] ← u
7. Visita_DFS(v)
7. cor[u] ← PRETO
8. tempo ← tempo + 1
9. f(u) ← tempo

u	cor(u)	pred(u)	d(u)	f(u)
0	CZ	-	1	
1	CZ	4	6	
2	CZ	5	3	
3	BR	-		
4	CZ	6	5	
5	CZ	0	2	
6	CZ	2	4	
7	BR	-		
8	BR	-		
9	BR	-		

DFS

Adj	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	1	1	0	0	0
1	0	0	0	0	1	0	0	0	0	0
2	0	0	0	0	0	1	1	0	0	1
3	0	0	0	0	0	0	0	1	1	0
4	0	1	0	0	0	0	0	0	0	1
5	1	0	1	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0
7	0	0	0	1	0	0	0	0	1	0
8	0	0	0	1	0	0	0	1	0	0
9	0	0	1	0	1	0	0	0	0	0

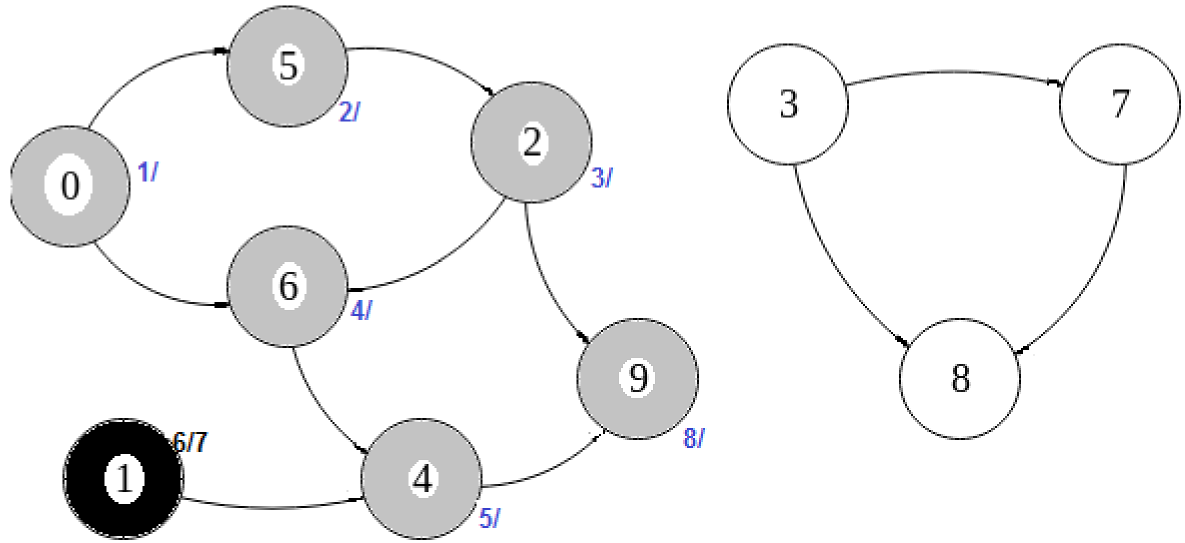



```
DFS(G)
para cada vértice u em G faça
     cor[u] = BRANCO
     pred[u] ← NIL
tempo ← 0
para cada vértice u em G faça
        se cor(u) = BRANCO
             então Visita_DFS(u)
devolve pred[1..n]

Visita_DFS(u)
 cor[u] ← CINZA
tempo ← tempo + 1
d(u) ← tempo
para cada v em Adj(u) faça
        se cor[v] = BRANCO
               então pred[v] ← u
                      Visita_DFS(v)
cor[u] ← PRETO
tempo ← tempo + 1
f(u) ← tempo
```

u	cor(u)	pred(u)	d(u)	f(u)
0	CZ	-	1	
1	PR	4	6	7
2	CZ	5	3	
3	BR	-		
4	CZ	6	5	
5	CZ	0	2	
6	CZ	2	4	
7	BR	-		
8	BR	-		
9	BR	-		

DFS

Adj	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	1	1	0	0	0
1	0	0	0	0	1	0	0	0	0	0
2	0	0	0	0	0	1	1	0	0	1
3	0	0	0	0	0	0	0	1	1	0
4	0	1	0	0	0	0	0	0	0	1
5	1	0	1	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0
7	0	0	0	1	0	0	0	0	1	0
8	0	0	0	1	0	0	0	1	0	0
9	0	0	1	0	1	0	0	0	0	0

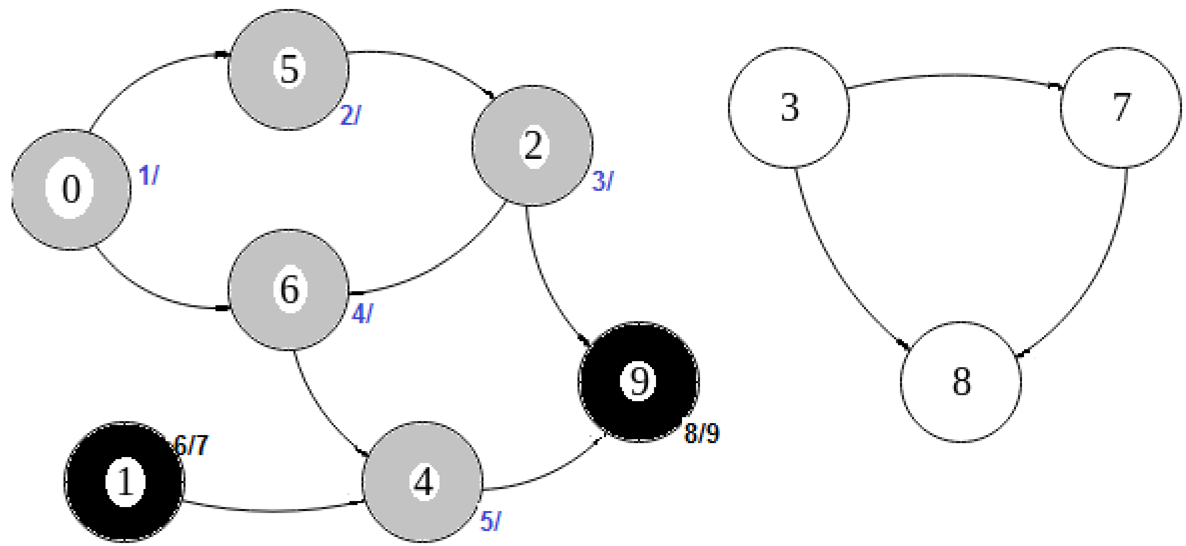



```
DFS(G)
para cada vértice u em G faça
    cor[u] = BRANCO
    pred[u] ← NIL
tempo ← 0
para cada vértice u em G faça
        se cor(u) = BRANCO
            então Visita_DFS(u)
devolve pred[1..n]

Visita_DFS(u)
 cor[u] ← CINZA
tempo ← tempo + 1
d(u)  ← tempo
para cada v em Adj(u) faça
        se cor[v] = BRANCO
                então pred[v] ← u
                        Visita_DFS(v)
cor[u] ← PRETO
tempo ← tempo + 1
f(u)  ← tempo
```

u	cor(u)	pred(u)	d(u)	f(u)
0	CZ	-	1	
1	PR	4	6	7
2	CZ	5	3	
3	BR	-		
4	CZ	6	5	
5	CZ	0	2	
6	CZ	2	4	
7	BR	-		
8	BR	-		
9	CZ	4	8	

DFS

Adj	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	1	1	0	0	0
1	0	0	0	0	1	0	0	0	0	0
2	0	0	0	0	0	1	1	0	0	1
3	0	0	0	0	0	0	0	1	1	0
4	0	1	0	0	0	0	0	0	0	1
5	1	0	1	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0
7	0	0	0	1	0	0	0	0	1	0
8	0	0	0	1	0	0	0	1	0	0
9	0	0	1	0	1	0	0	0	0	0

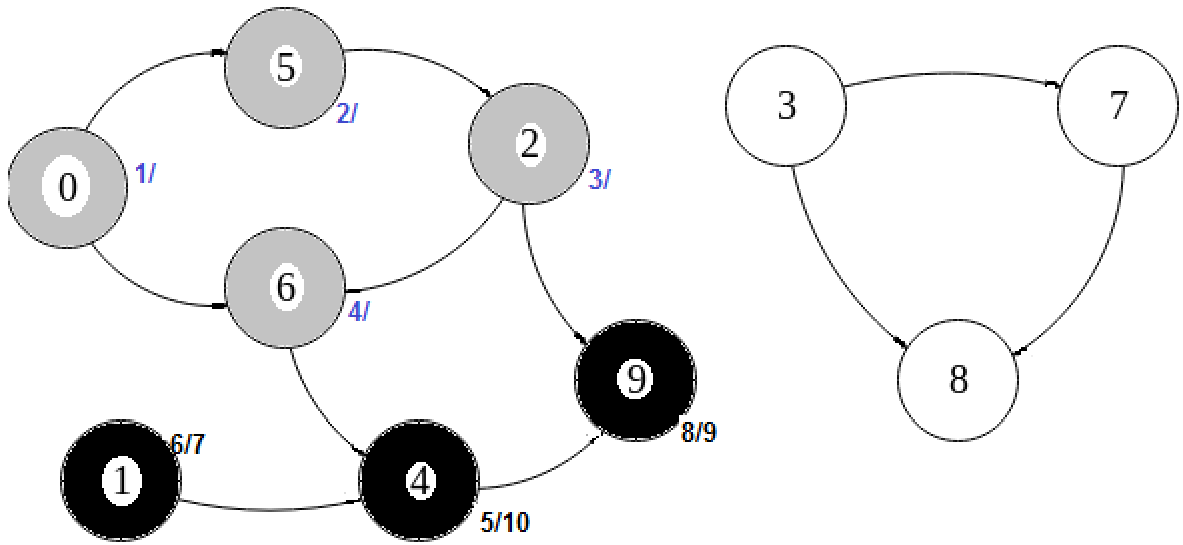



```
DFS(G)
1.  para cada vértice u em G faça
2.      cor[u] = BRANCO
3.      pred[u] ← NIL
4.  tempo ← 0
5.  para cada vértice u em G faça
6.          se cor(u) = BRANCO
7.              então Visita_DFS(u)
8.  devolve pred[1..n]

Visita_DFS(u)
1.   cor[u] ← CINZA
2.  tempo ← tempo + 1
3.  d(u)  ← tempo
4.  para cada v em Adj(u) faça
5.          se cor[v] = BRANCO
6.                  então pred[v] ← u
7.                          Visita_DFS(v)
7.  cor[u] ← PRETO
8.  tempo ← tempo + 1
9.  f(u)  ← tempo
```

u	cor(u)	pred(u)	d(u)	f(u)
0	CZ	-	1	
1	PR	4	6	7
2	CZ	5	3	
3	BR	-		
4	CZ	6	5	
5	CZ	0	2	
6	CZ	2	4	
7	BR	-		
8	BR	-		
9	PR	4	8	9

DFS

Adj	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	1	1	0	0	0
1	0	0	0	0	1	0	0	0	0	0
2	0	0	0	0	0	1	1	0	0	1
3	0	0	0	0	0	0	0	1	1	0
4	0	1	0	0	0	0	0	0	0	1
5	1	0	1	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0
7	0	0	0	1	0	0	0	0	1	0
8	0	0	0	1	0	0	0	1	0	0
9	0	0	1	0	1	0	0	0	0	0



DFS(G)

```
1.  para cada vértice u em G faça
2.      cor[u] = BRANCO
3.      pred[u] ← NIL
4.  tempo ← 0
5.  para cada vértice u em G faça
6.          se cor(u) = BRANCO
7.              então Visita_DFS(u)
8.  devolve pred[1..n]
```

Visita_DFS(u)

```
1.   cor[u] ← CINZA
2.  tempo ← tempo + 1
3.  d(u)  ← tempo
4.  para cada v em Adj(u) faça
5.          se cor[v] = BRANCO
6.                  então pred[v] ← u
7.                          Visita_DFS(v)
7.  cor[u] ← PRETO
8.  tempo ← tempo + 1
9.  f(u)  ← tempo
```

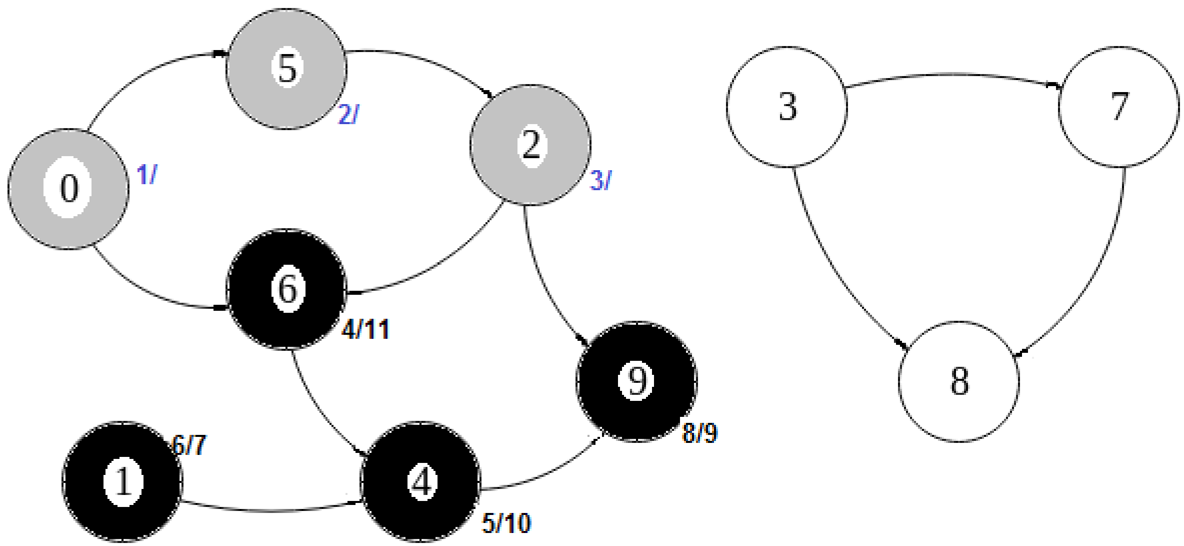
u	cor(u)	pred(u)	d(u)	f(u)
0	CZ	-	1	
1	PR	4	6	7
2	CZ	5	3	
3	BR	-		
4	PR	6	5	10
5	CZ	0	2	
6	CZ	2	4	
7	BR	-		
8	BR	-		
9	PR	4	8	9

DFS

Adj	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	1	1	0	0	0
1	0	0	0	0	1	0	0	0	0	0
2	0	0	0	0	0	1	1	0	0	1
3	0	0	0	0	0	0	0	1	1	0
4	0	1	0	0	0	0	0	0	0	1
5	1	0	1	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0
7	0	0	0	1	0	0	0	0	1	0
8	0	0	0	1	0	0	0	1	0	0
9	0	0	1	0	1	0	0	0	0	0

DFS(G)
1. para cada vértice u em G faça
2. cor[u] = BRANCO
3. pred[u] ← NIL
4. tempo ← 0
5. para cada vértice u em G faça
6. se cor(u) = BRANCO
7. então Visita_DFS(u)
8. devolve pred[1..n]

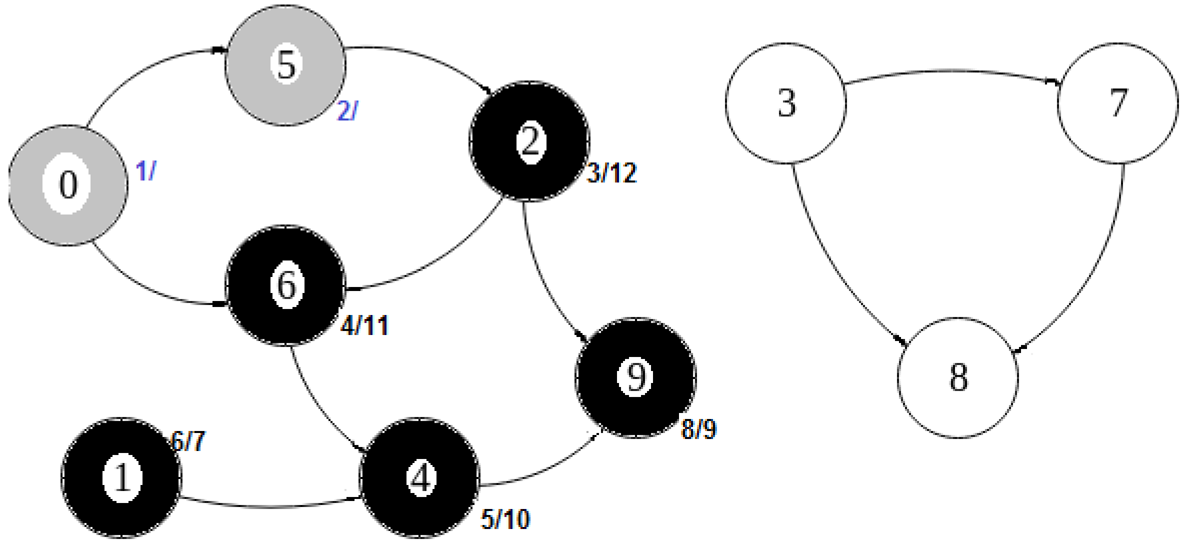
Visita_DFS(u)
1. cor[u] ← CINZA
2. tempo ← tempo + 1
3. d(u) ← tempo
4. para cada v em Adj(u) faça
5. se cor[v] = BRANCO
6. então pred[v] ← u
7. Visita_DFS(v)
7. cor[u] ← PRETO
8. tempo ← tempo + 1
9. f(u) ← tempo



u	cor(u)	pred(u)	d(u)	f(u)
0	CZ	-	1	
1	PR	4	6	7
2	CZ	5	3	
3	BR	-		
4	PR	6	5	10
5	CZ	0	2	
6	PR	2	4	11
7	BR	-		
8	BR	-		
9	PR	4	8	9

DFS

Adj	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	1	1	0	0	0
1	0	0	0	0	1	0	0	0	0	0
2	0	0	0	0	0	1	1	0	0	1
3	0	0	0	0	0	0	0	1	1	0
4	0	1	0	0	0	0	0	0	0	1
5	1	0	1	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0
7	0	0	0	1	0	0	0	0	1	0
8	0	0	0	1	0	0	0	1	0	0
9	0	0	1	0	1	0	0	0	0	0



DFS(G)

```
para cada vértice u em G faça
      cor[u] = BRANCO
      pred[u] ← NIL
tempo ← 0
para cada vértice u em G faça
          se cor(u) = BRANCO
                então Visita_DFS(u)
devolve pred[1..n]
```

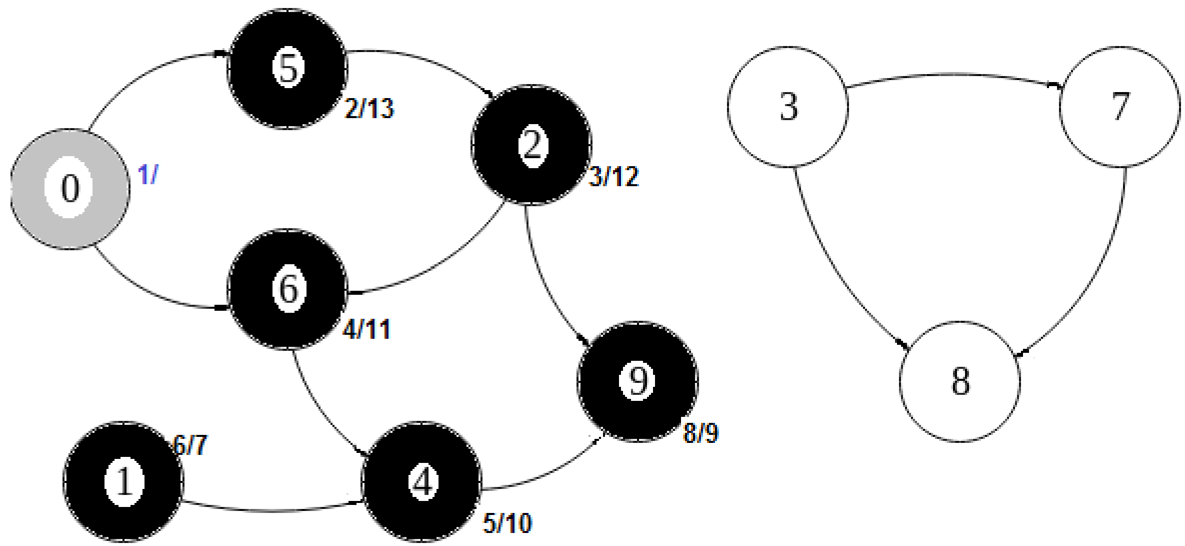
Visita_DFS(u)

```
 cor[u] ← CINZA
tempo ← tempo + 1
d(u)  ← tempo
para cada v em Adj(u) faça
          se cor[v] = BRANCO
                  então pred[v] ← u
                         Visita_DFS(v)
cor[u] ← PRETO
tempo ← tempo + 1
f(u)  ← tempo
```

u	cor(u)	pred(u)	d(u)	f(u)
0	CZ	-	1	
1	PR	4	6	7
2	PR	5	3	12
3	BR	-		
4	PR	6	5	10
5	CZ	0	2	
6	PR	2	4	11
7	BR	-		
8	BR	-		
9	PR	4	8	9

DFS

Adj	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	1	1	0	0	0
1	0	0	0	0	1	0	0	0	0	0
2	0	0	0	0	0	1	1	0	0	1
3	0	0	0	0	0	0	0	1	1	0
4	0	1	0	0	0	0	0	0	0	1
5	1	0	1	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0
7	0	0	0	1	0	0	0	0	1	0
8	0	0	0	1	0	0	0	1	0	0
9	0	0	1	0	1	0	0	0	0	0



DFS(G)

```
1. para cada vértice u em G faça
2.      cor[u] = BRANCO
3.      pred[u] ← NIL
4. tempo ← 0
5. para cada vértice u em G faça
6.          se cor(u) = BRANCO
7.               então Visita_DFS(u)
8. devolve pred[1..n]
```

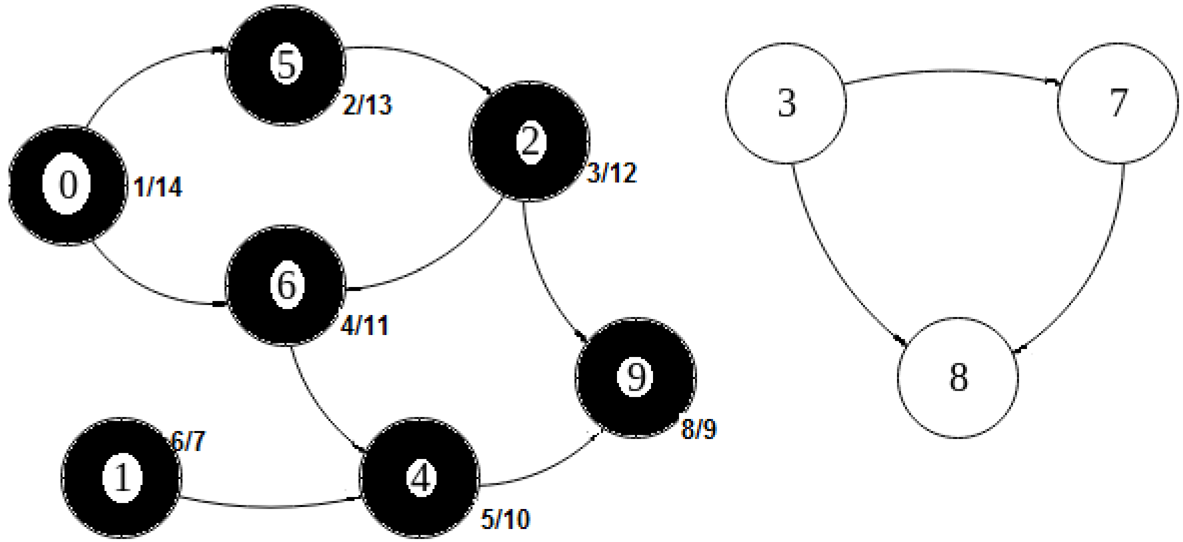
Visita_DFS(u)

```
1.  cor[u] ← CINZA
2. tempo ← tempo + 1
3. d(u)  ← tempo
4. para cada v em Adj(u) faça
5.          se cor[v] = BRANCO
6.                 então pred[v] ← u
7.                          Visita_DFS(v)
7. cor[u] ← PRETO
8. tempo ← tempo + 1
9. f(u)  ← tempo
```

u	cor(u)	pred(u)	d(u)	f(u)
0	CZ	-	1	
1	PR	4	6	7
2	PR	5	3	12
3	BR	-		
4	PR	6	5	10
5	PR	0	2	13
6	PR	2	4	11
7	BR	-		
8	BR	-		
9	PR	4	8	9

DFS

Adj	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	1	1	0	0	0
1	0	0	0	0	1	0	0	0	0	0
2	0	0	0	0	0	1	1	0	0	1
3	0	0	0	0	0	0	0	1	1	0
4	0	1	0	0	0	0	0	0	0	1
5	1	0	1	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0
7	0	0	0	1	0	0	0	0	1	0
8	0	0	0	1	0	0	0	1	0	0
9	0	0	1	0	1	0	0	0	0	0



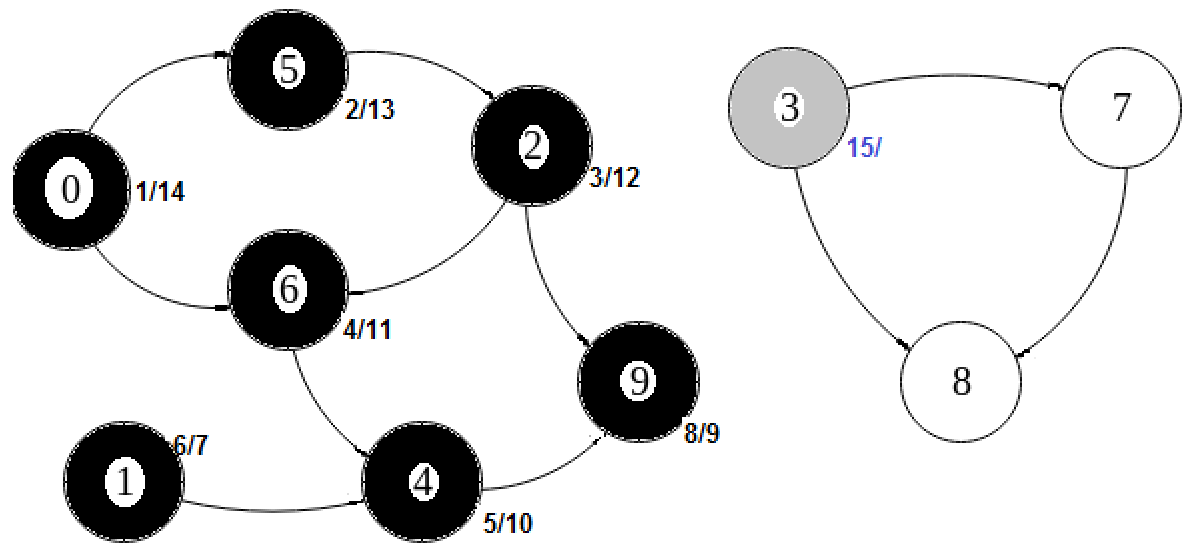
DFS(G)
```
1. para cada vértice u em G faça
2.     cor[u] = BRANCO
3.     pred[u] ← NIL
4. tempo ← 0
5. para cada vértice u em G faça
6.         se cor(u) = BRANCO
7.             então Visita_DFS(u)
8. devolve pred[1..n]
```

Visita_DFS(u)
```
1.  cor[u] ← CINZA
2. tempo ← tempo + 1
3. d(u)  ← tempo
4. para cada v em Adj(u) faça
5.         se cor[v] = BRANCO
6.                 então pred[v] ← u
7.                         Visita_DFS(v)
7. cor[u] ← PRETO
8. tempo ← tempo + 1
9. f(u)  ← tempo
```

u	cor(u)	pred(u)	d(u)	f(u)
0	PR	-	1	14
1	PR	4	6	7
2	PR	5	3	12
3	BR	-		
4	PR	6	5	10
5	PR	0	2	13
6	PR	2	4	11
7	BR	-		
8	BR	-		
9	PR	4	8	9

DFS

Adj	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	1	1	0	0	0
1	0	0	0	0	1	0	0	0	0	0
2	0	0	0	0	0	1	1	0	0	1
3	0	0	0	0	0	0	0	1	1	0
4	0	1	0	0	0	0	0	0	0	1
5	1	0	1	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0
7	0	0	0	1	0	0	0	0	1	0
8	0	0	0	1	0	0	0	1	0	0
9	0	0	1	0	1	0	0	0	0	0



DFS(G)

```
para cada vértice u em G faça
    cor[u] = BRANCO
    pred[u] ← NIL
tempo ← 0
para cada vértice u em G faça
        se cor(u) = BRANCO
            então Visita_DFS(u)
devolve pred[1..n]
```

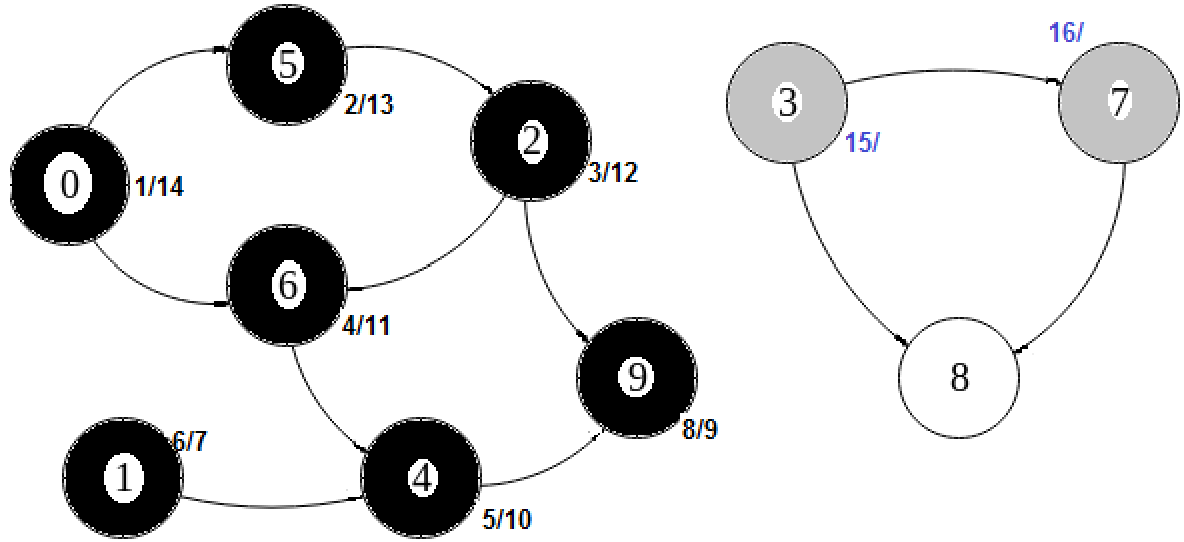
Visita_DFS(u)

```
 cor[u] ← CINZA
tempo ← tempo + 1
d(u)  ← tempo
para cada v em Adj(u) faça
        se cor[v] = BRANCO
                então pred[v] ← u
                        Visita_DFS(v)
cor[u] ← PRETO
tempo ← tempo + 1
f(u)  ← tempo
```

u	cor(u)	pred(u)	d(u)	f(u)
0	PR	-	1	14
1	PR	4	6	7
2	PR	5	3	12
3	CZ	-	15	
4	PR	6	5	10
5	PR	0	2	13
6	PR	2	4	11
7	BR	-		
8	BR	-		
9	PR	4	8	9

DFS

Adj	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	1	1	0	0	0
1	0	0	0	0	1	0	0	0	0	0
2	0	0	0	0	0	1	1	0	0	1
3	0	0	0	0	0	0	0	1	1	0
4	0	1	0	0	0	0	0	0	0	1
5	1	0	1	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0
7	0	0	0	1	0	0	0	0	1	0
8	0	0	0	1	0	0	0	1	0	0
9	0	0	1	0	1	0	0	0	0	0

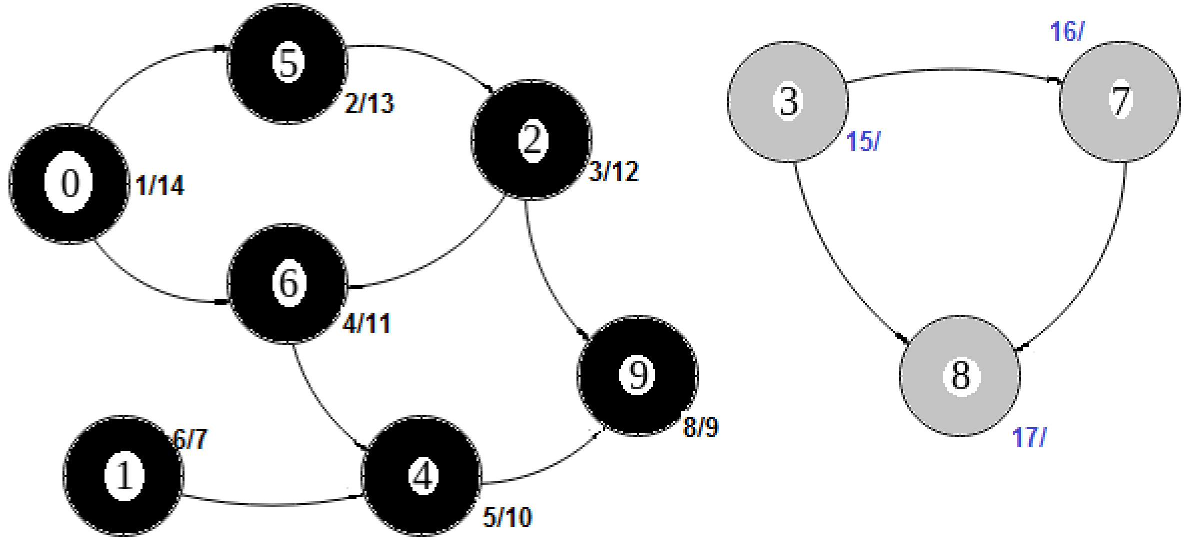



```
DFS(G)
para cada vértice u em G faça
     cor[u] = BRANCO
     pred[u] ← NIL
tempo ← 0
para cada vértice u em G faça
        se cor(u) = BRANCO
             então Visita_DFS(u)
devolve pred[1..n]

Visita_DFS(u)
 cor[u] ← CINZA
tempo ← tempo + 1
d(u)  ← tempo
para cada v em Adj(u) faça
        se cor[v] = BRANCO
              então pred[v] ← u
                     Visita_DFS(v)
cor[u] ← PRETO
tempo ← tempo + 1
f(u)  ← tempo
```

u	cor(u)	pred(u)	d(u)	f(u)
0	PR	-	1	14
1	PR	4	6	7
2	PR	5	3	12
3	CZ	-	15	
4	PR	6	5	10
5	PR	0	2	13
6	PR	2	4	11
7	CZ	3	16	
8	BR	-		
9	PR	4	8	9

DFS

Adj	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	1	1	0	0	0
1	0	0	0	0	1	0	0	0	0	0
2	0	0	0	0	0	1	1	0	0	1
3	0	0	0	0	0	0	0	1	1	0
4	0	1	0	0	0	0	0	0	0	1
5	1	0	1	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0
7	0	0	0	1	0	0	0	0	1	0
8	0	0	0	1	0	0	0	1	0	0
9	0	0	1	0	1	0	0	0	0	0

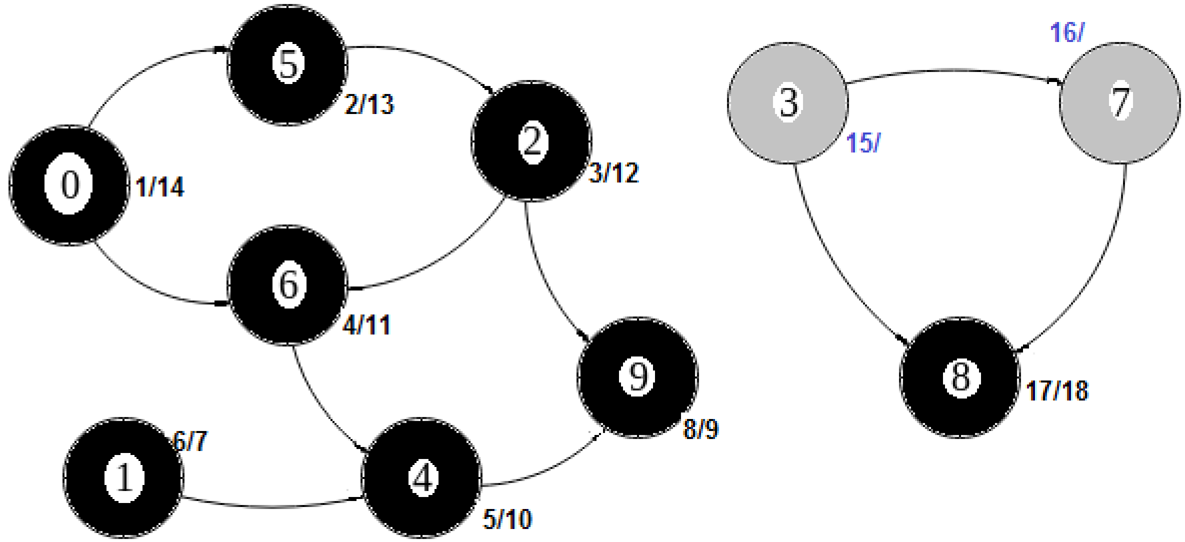



```
DFS(G)
para cada vértice u em G faça
    cor[u] = BRANCO
    pred[u] ← NIL
tempo ← 0
para cada vértice u em G faça
        se cor(u) = BRANCO
            então Visita_DFS(u)
devolve pred[1..n]

Visita_DFS(u)
 cor[u] ← CINZA
tempo ← tempo + 1
d(u)  ← tempo
para cada v em Adj(u) faça
        se cor[v] = BRANCO
                então pred[v] ← u
                        Visita_DFS(v)
cor[u] ← PRETO
tempo ← tempo + 1
f(u)  ← tempo
```

u	cor(u)	pred(u)	d(u)	f(u)
0	PR	-	1	14
1	PR	4	6	7
2	PR	5	3	12
3	CZ	-	15	
4	PR	6	5	10
5	PR	0	2	13
6	PR	2	4	11
7	CZ	3	16	
8	CZ	7	17	
9	PR	4	8	9

DFS

Adj	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	1	1	0	0	0
1	0	0	0	0	1	0	0	0	0	0
2	0	0	0	0	0	1	1	0	0	1
3	0	0	0	0	0	0	0	1	1	0
4	0	1	0	0	0	0	0	0	0	1
5	1	0	1	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0
7	0	0	0	1	0	0	0	0	1	0
8	0	0	0	1	0	0	0	1	0	0
9	0	0	1	0	1	0	0	0	0	0



DFS(G)

```
para cada vértice u em G faça
     cor[u] = BRANCO
     pred[u] ← NIL
tempo ← 0
para cada vértice u em G faça
        se cor(u) = BRANCO
             então Visita_DFS(u)
devolve pred[1..n]
```

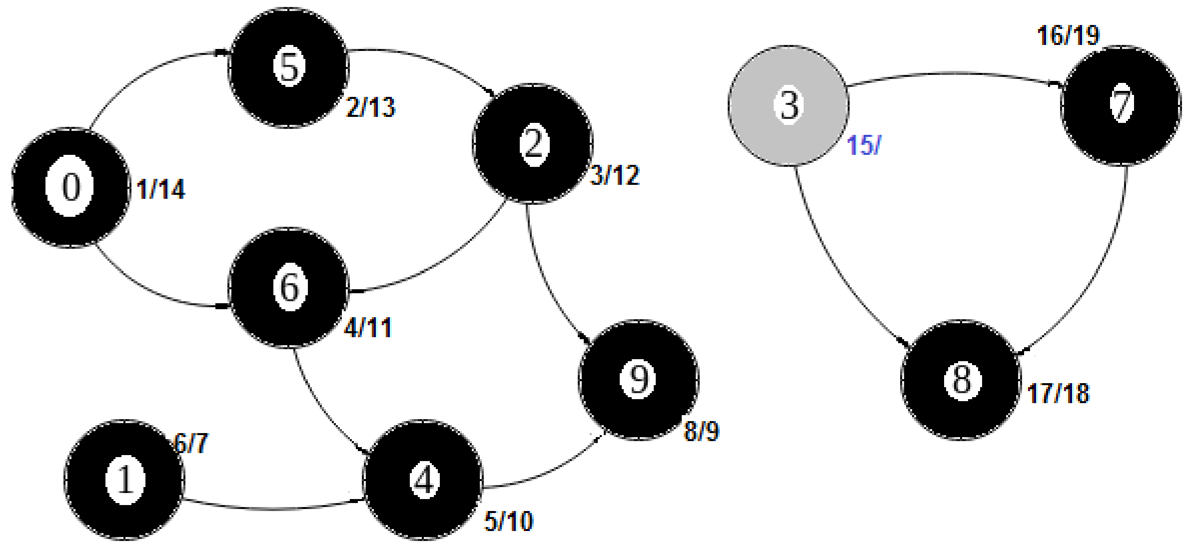
Visita_DFS(u)

```
 cor[u] ← CINZA
tempo ← tempo + 1
d(u)  ← tempo
para cada v em Adj(u) faça
        se cor[v] = BRANCO
                então pred[v] ← u
                       Visita_DFS(v)
cor[u] ← PRETO
tempo ← tempo + 1
f(u)  ← tempo
```

u	cor(u)	pred(u)	d(u)	f(u)
0	PR	-	1	14
1	PR	4	6	7
2	PR	5	3	12
3	CZ	-	15	
4	PR	6	5	10
5	PR	0	2	13
6	PR	2	4	11
7	CZ	3	16	
8	PR	7	17	18
9	PR	4	8	9

DFS

Adj	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	1	1	0	0	0
1	0	0	0	0	1	0	0	0	0	0
2	0	0	0	0	0	1	1	0	0	1
3	0	0	0	0	0	0	0	1	1	0
4	0	1	0	0	0	0	0	0	0	1
5	1	0	1	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0
7	0	0	0	1	0	0	0	0	1	0
8	0	0	0	1	0	0	0	1	0	0
9	0	0	1	0	1	0	0	0	0	0



DFS(G)

1. para cada vértice u em G faça
2. cor[u] = BRANCO
3. pred[u] ← NIL
4. tempo ← 0
5. para cada vértice u em G faça
6. se cor(u) = BRANCO
7. então Visita_DFS(u)
8. devolve pred[1..n]

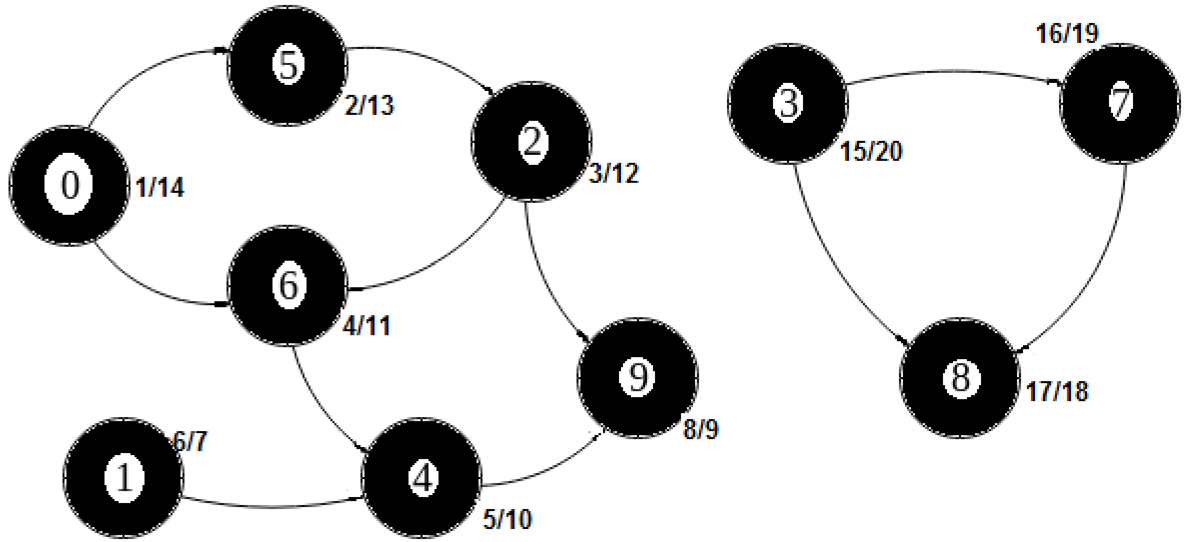
Visita_DFS(u)

1. cor[u] ← CINZA
2. tempo ← tempo + 1
3. d(u) ← tempo
4. para cada v em Adj(u) faça
5. se cor[v] = BRANCO
6. então pred[v] ← u
7. Visita_DFS(v)
7. cor[u] ← PRETO
8. tempo ← tempo + 1
9. f(u) ← tempo

u	cor(u)	pred(u)	d(u)	f(u)
0	PR	-	1	14
1	PR	4	6	7
2	PR	5	3	12
3	CZ	-	15	
4	PR	6	5	10
5	PR	0	2	13
6	PR	2	4	11
7	PR	3	16	19
8	PR	7	17	18
9	PR	4	8	9

DFS

Adj	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	1	1	0	0	0
1	0	0	0	0	1	0	0	0	0	0
2	0	0	0	0	0	1	1	0	0	1
3	0	0	0	0	0	0	0	1	1	0
4	0	1	0	0	0	0	0	0	0	1
5	1	0	1	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0
7	0	0	0	1	0	0	0	0	1	0
8	0	0	0	1	0	0	0	1	0	0
9	0	0	1	0	1	0	0	0	0	0



DFS(G)

1. para cada vértice u em G faça
2. cor[u] = BRANCO
3. pred[u] ← NIL
4. tempo ← 0
5. para cada vértice u em G faça
6. se cor(u) = BRANCO
7. então Visita_DFS(u)
8. devolve pred[1..n]

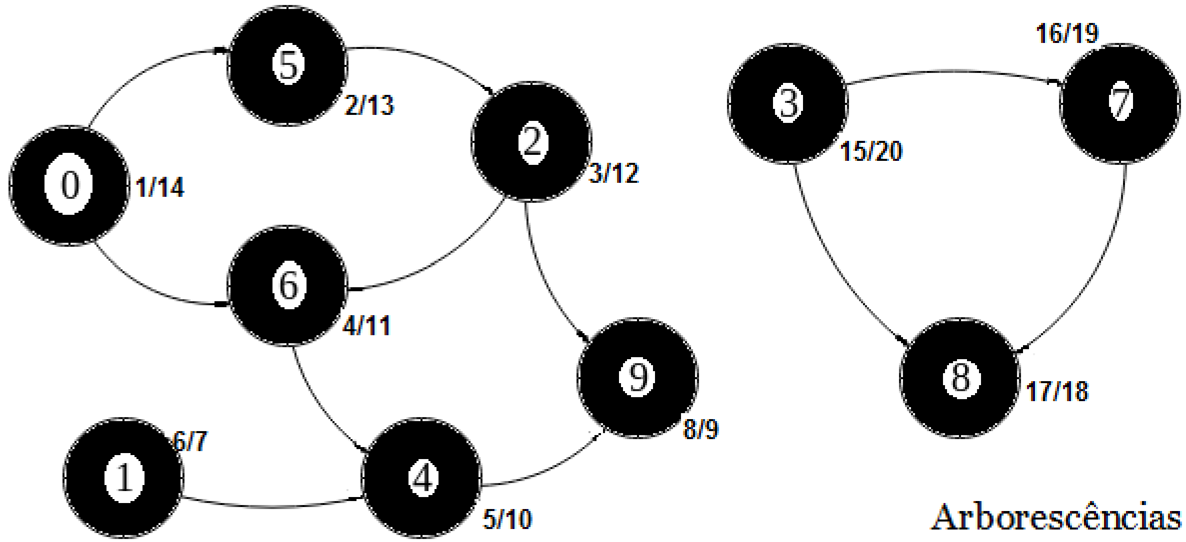
Visita_DFS(u)

1. cor[u] ← CINZA
2. tempo ← tempo + 1
3. d(u) ← tempo
4. para cada v em Adj(u) faça
5. se cor[v] = BRANCO
6. então pred[v] ← u
7. Visita_DFS(v)
7. cor[u] ← PRETO
8. tempo ← tempo + 1
9. f(u) ← tempo

u	cor(u)	pred(u)	d(u)	f(u)
0	PR	-	1	14
1	PR	4	6	7
2	PR	5	3	12
3	PR	-	15	20
4	PR	6	5	10
5	PR	0	2	13
6	PR	2	4	11
7	PR	3	16	19
8	PR	7	17	18
9	PR	4	8	9

DFS

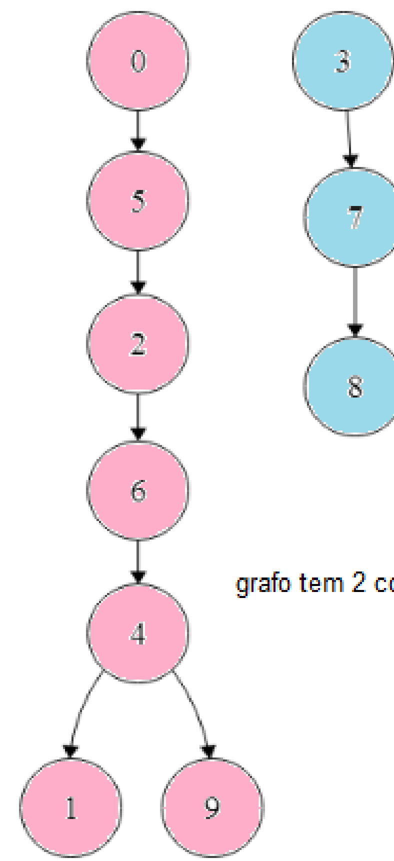
Adj	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	1	1	0	0	0
1	0	0	0	0	1	0	0	0	0	0
2	0	0	0	0	0	1	1	0	0	1
3	0	0	0	0	0	0	0	1	1	0
4	0	1	0	0	0	0	0	0	0	1
5	1	0	1	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0
7	0	0	0	1	0	0	0	0	1	0
8	0	0	0	1	0	0	0	1	0	0
9	0	0	1	0	1	0	0	0	0	0




```
DFS(G)
para cada vértice u em G faça
     cor[u] = BRANCO
     pred[u] ← NIL
tempo ← 0
para cada vértice u em G faça
          se cor(u) = BRANCO
               então Visita_DFS(u)
devolve pred[1..n]

Visita_DFS(u)
 cor[u] ← CINZA
tempo ← tempo + 1
d(u)  ← tempo
para cada v em Adj(u) faça
          se cor[v] = BRANCO
                    então pred[v] ← u
                              Visita_DFS(v)
cor[u] ← PRETO
tempo ← tempo + 1
f(u)  ← tempo
```

u	cor(u)	pred(u)	d(u)	f(u)
0	PR	-	1	14
1	PR	4	6	7
2	PR	5	3	12
3	PR	-	15	20
4	PR	6	5	10
5	PR	0	2	13
6	PR	2	4	11
7	PR	3	16	19
8	PR	7	17	18
9	PR	4	8	9

Arborescências

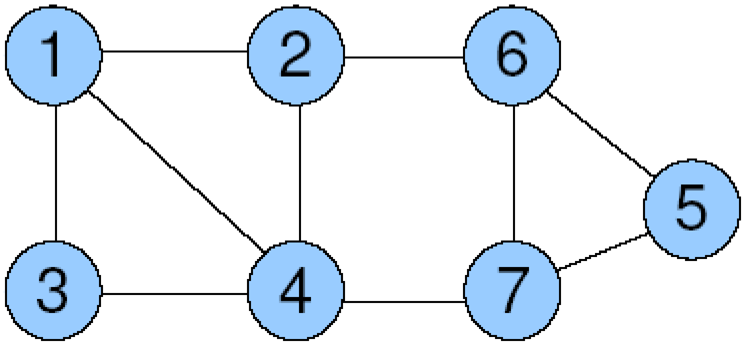


grafo tem 2 componentes

Busca em Largura (BFS)

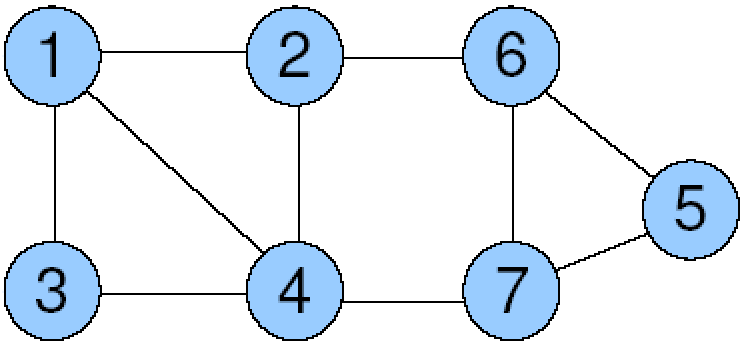
Busca em Largura (BFS)

- Explorar vértices descobertos mais antigos primeiro



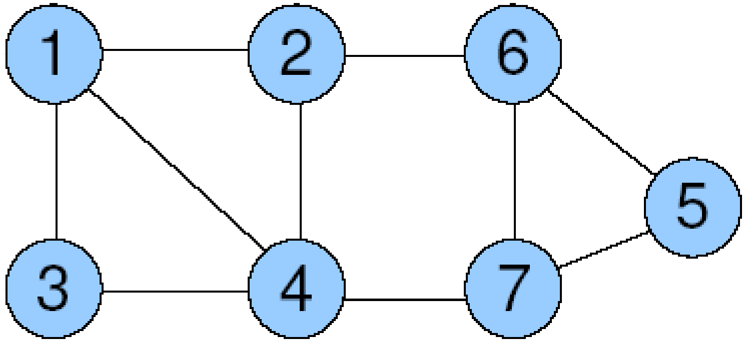
- Origem: vértice 1

Busca em Largura (BFS)



- Origem: vértice 1
- Em que ordem os vértices são *descobertos*?

Busca em Largura (BFS)

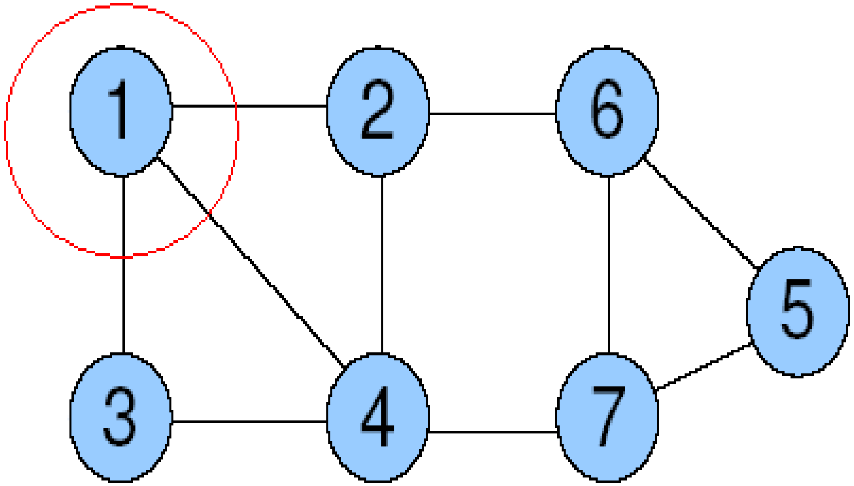


- Em que ordem os vértices são *descobertos*?

Assumir arestas são exploradas em ordem crescente dos vértices adjacentes (matriz ou lista de adjacência)

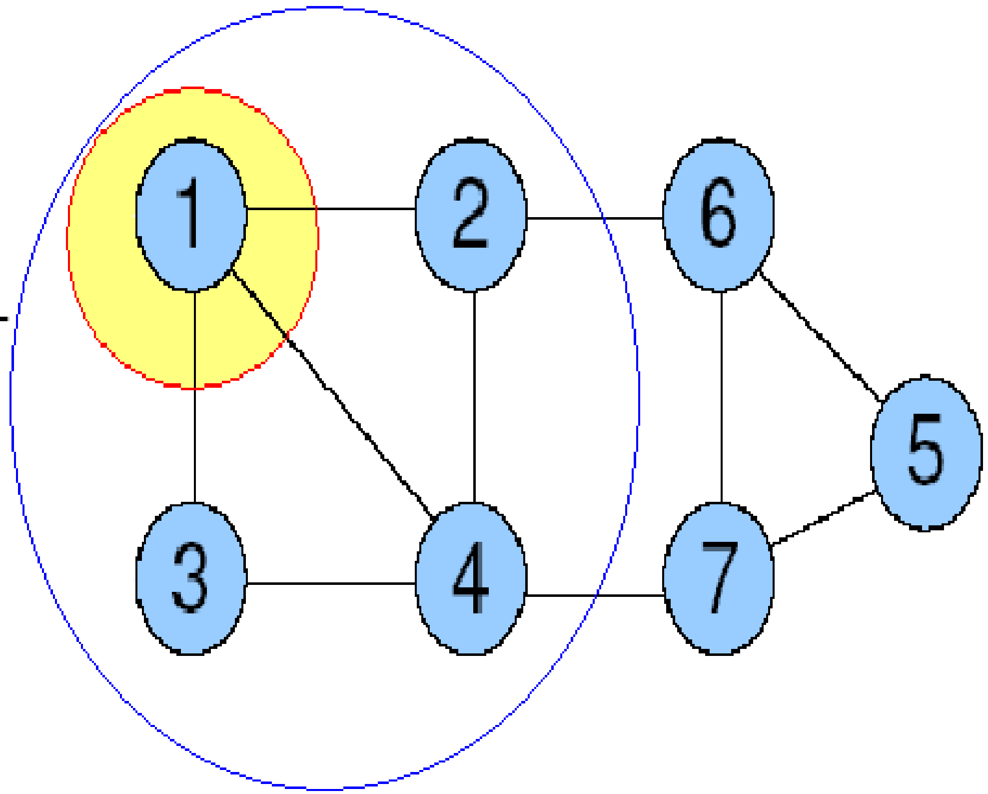
Busca em Largura: Interpretação

- Onda é propagada à partir da raiz
  - Origem: vértice 1



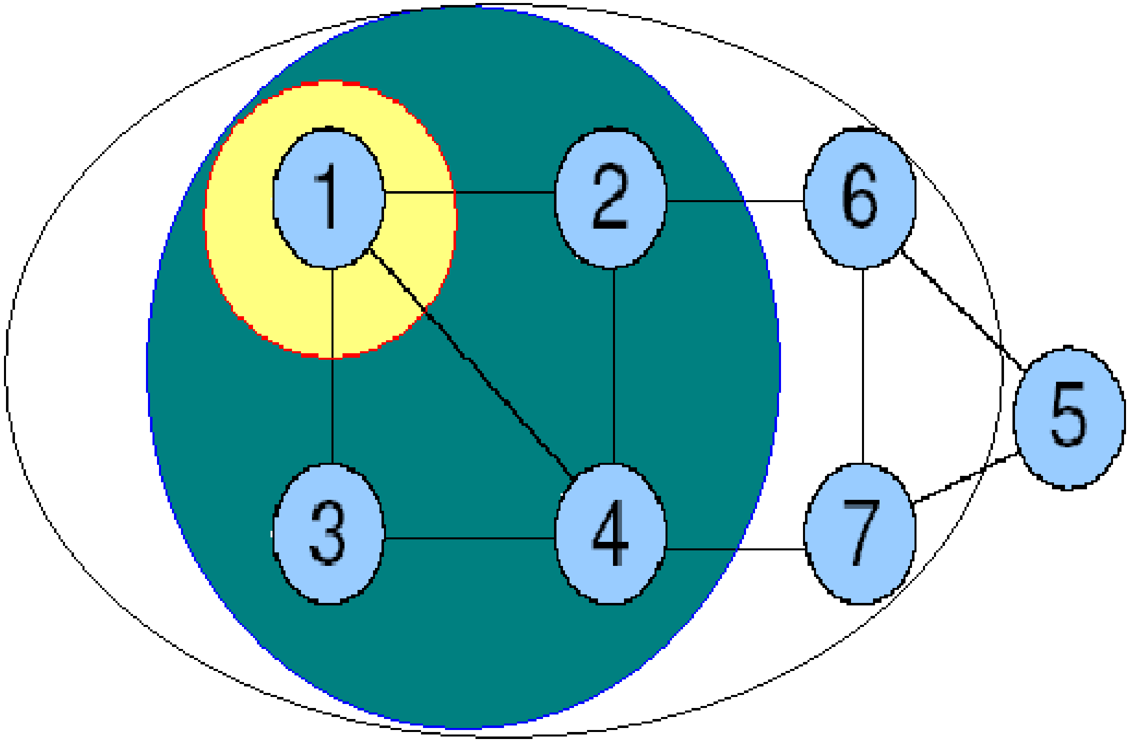
Busca em Largura: Interpretação

- Onda é propagada à partir da raiz
  - Origem: vértice 1



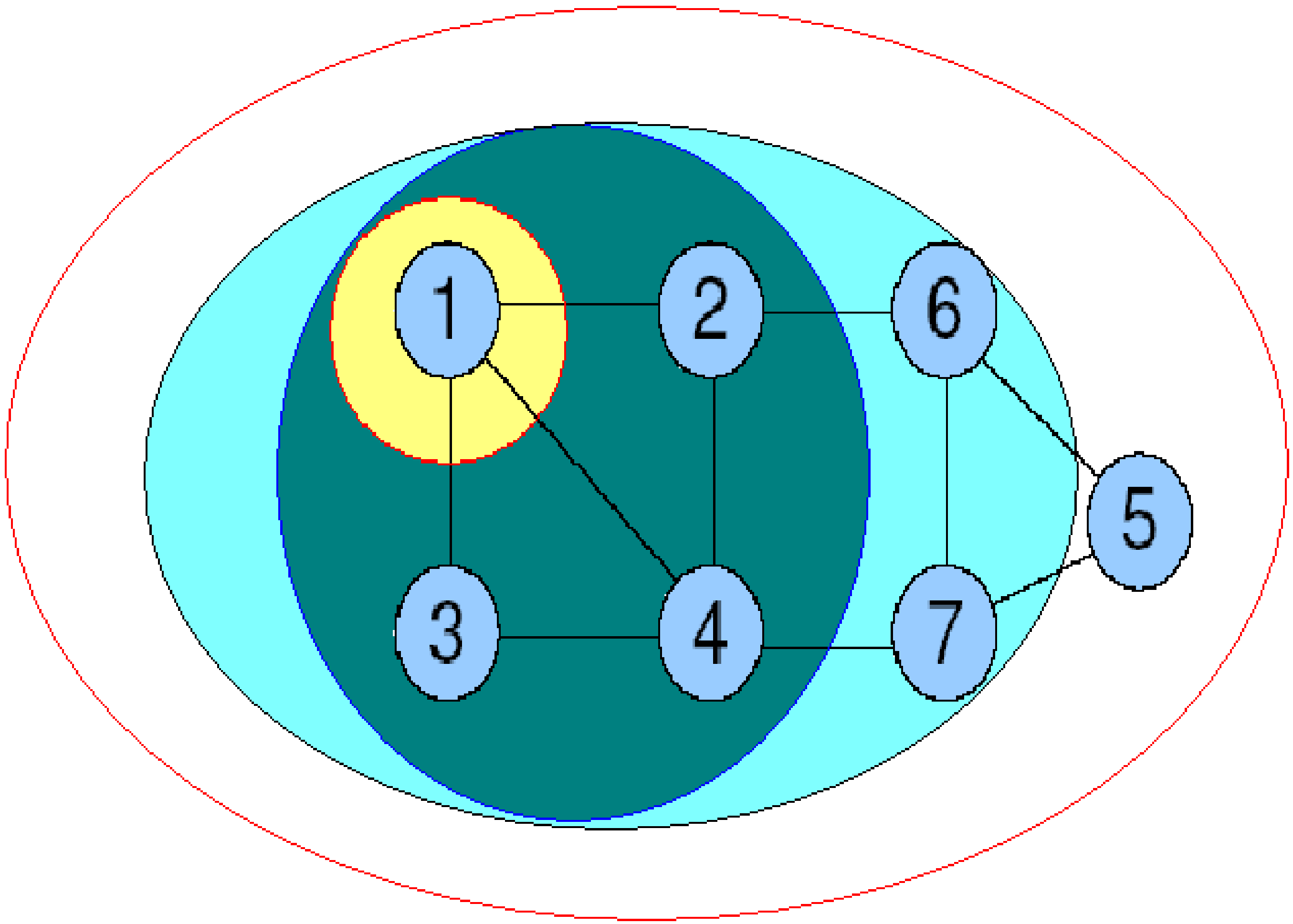
Busca em Largura: Interpretação

- Onda é propagada à partir da raiz
  - Origem: vértice 1



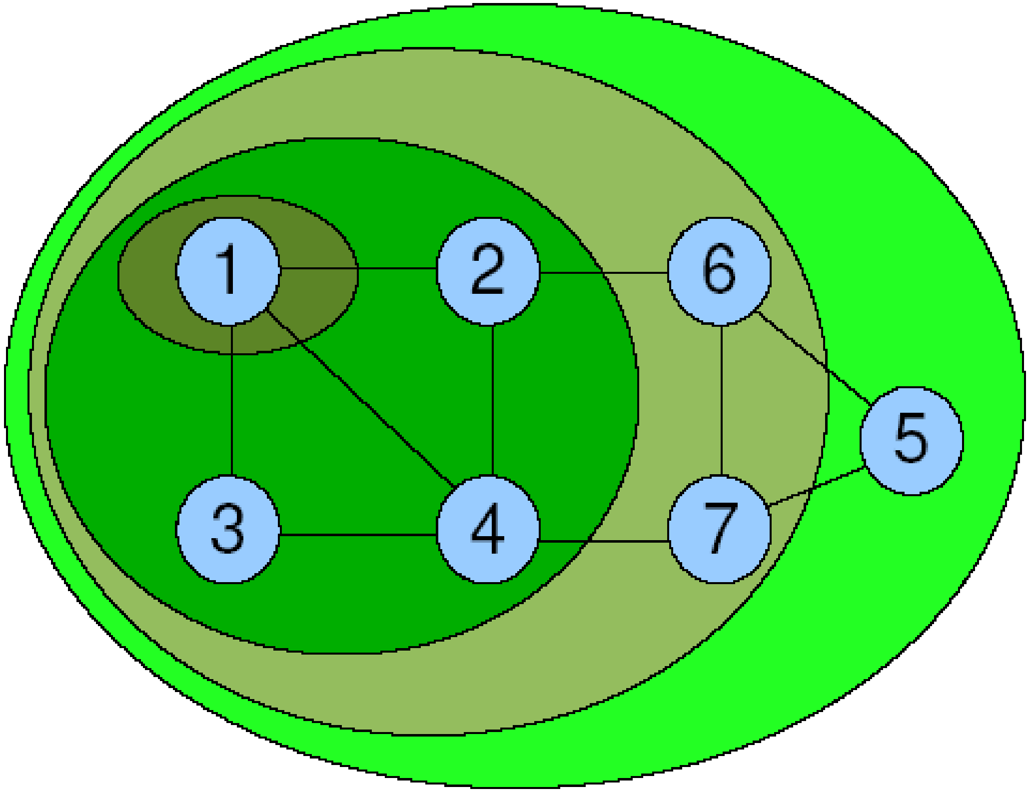
Busca em Largura: Interpretação

- Onda é propagada à partir da raiz

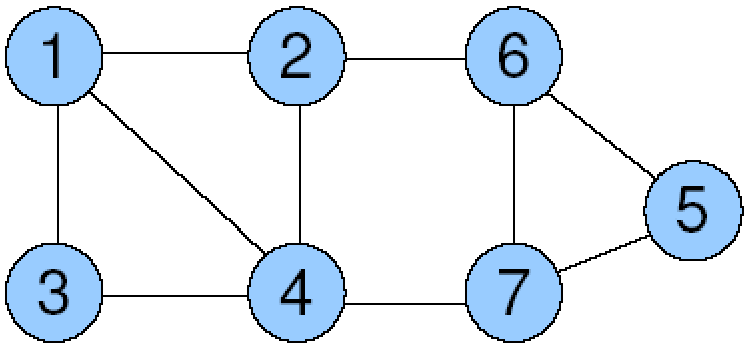


Interpretação

- Onda expande em círculos, descobrindo vértices alcançáveis!



Camadas: Exemplo



- L_0 : vértice 2
- L_i : ?

4. Busca em Largura

Pesquisa primeiro na extensão: a busca em largura

A busca em largura ou pesquisa primeiro na extensão - do inglês breadth-first search: BFS - é um dos algoritmos mais simples para se pesquisar um (dí)grafo possibilitando descobrir distâncias entre vértices.

O comprimento de um caminho é o número de arestas (arcos) do caminho:

- A distância de um vértice x a um vértice y é o comprimento de um caminho de comprimento mínimo que começa em x e termina em y.
- Naturalmente a distância de x a y pode não ser igual à distância de y a x.

4. Busca em Largura

Pesquisa primeiro na extensão: a busca em largura

A distância de x a y é *infinita* se não existe caminho algum. É importante ressaltar que a afirmação "a distância de x a y é k" equivale a duas afirmações:

(i) existe um caminho de x a y cujo comprimento é k,

(ii) não existe caminho de x a y cujo comprimento seja menor que k.

4. Busca em Largura

Dado um vértice x de um grafo, encontrar a distância de x a cada um dos demais vértices.

No algoritmo:

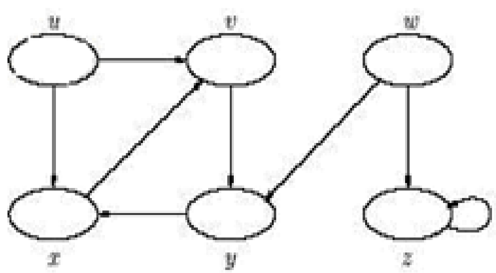
- Vértices são inicialmente identificados com BRANCO.
- Depois tornar-se CINZA e finalmente PRETO.
- Um vértice se torna CINZA quando é atingido pela primeira vez e permanece CINZA enquanto seus vizinhos não tiverem sido todos explorados. Porém é preciso que os vértices de cor CINZA sejam examinados *na mesma ordem em que foram encontrados.*
- Para administrar essa ordem, vamos guardar os vértices de cor CINZA em uma **fila**.

4. Busca em Largura

```
BFS(G,x)
para u ← 1 até n faça
        cor[u] ← BRANCO
        d[u] ← ∞
cor[x] ← CINZA
d[x] = 0
Q ← Inicialiaza-Fila(Q,x)
enquanto Q ≠ ∅ faça
        u ← Primeiro-da-Fila(Q)
        para cada v em Adj[u] faça
                se cor[v] = BRANCO
                        então cor[v] ← CINZA
                                dist[v] ← dist[u]+1
                                Insira-na-Fila(Q,v)
        Remova-da-Fila(Q)
        cor[u] ← PRETO
devolve dist[1..n]
```

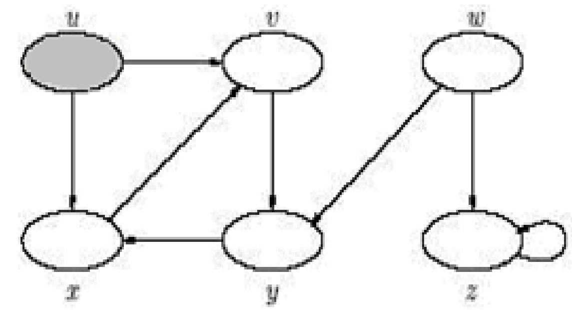
4. Busca em Largura

Exemplo: andamento de BFS em um dígrafo começando por "u"



Dígrafo e listas:

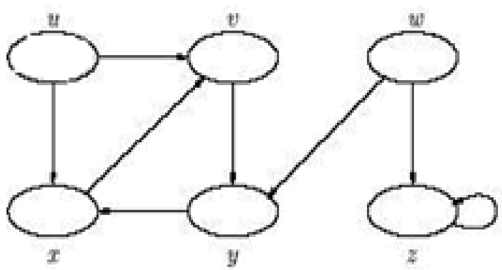
u	→	v	→	x
v	→	y		
w	→	z		
x	→	v		
y	→	x		
z	→	x		



(i) Fila: u

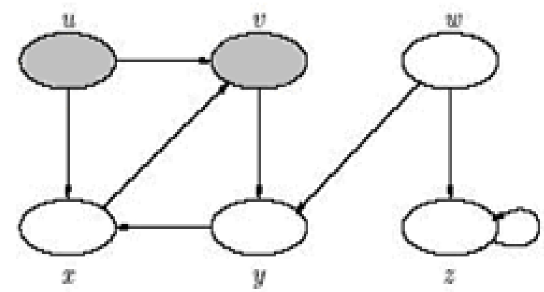
4. Busca em Largura

Exemplo: andamento de BFS em um dígrafo começando por "u"



Dígrafo e listas:

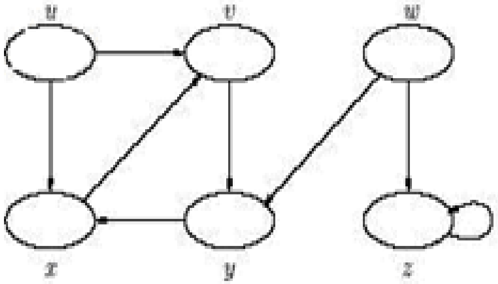
u	→	v	→	x
v	→	y		
w	→	z		
x	→	v		
y	→	x		
z	→	x		



(ii) Fila: u,v

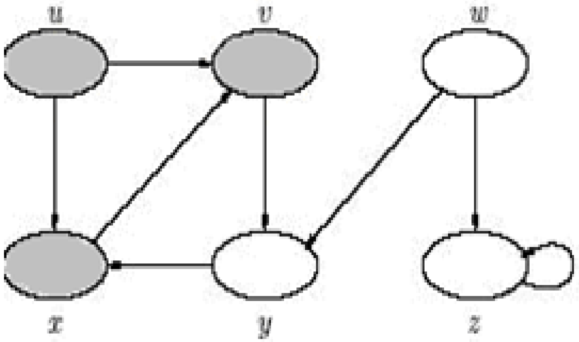
4. Busca em Largura

Exemplo: andamento de BFS em um dígrafo começando por "u"



Dígrafo e listas:

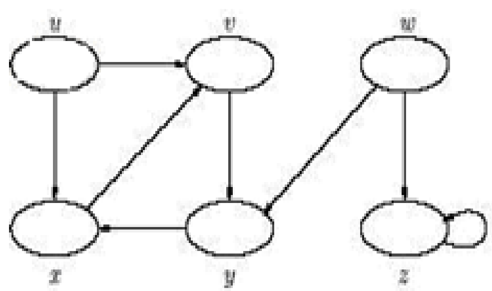
u	→	v	→	x
v	→	y		
w	→	z		
x	→	v		
y	→	x		
z	→	x		



(iii) Fila: u, v, x

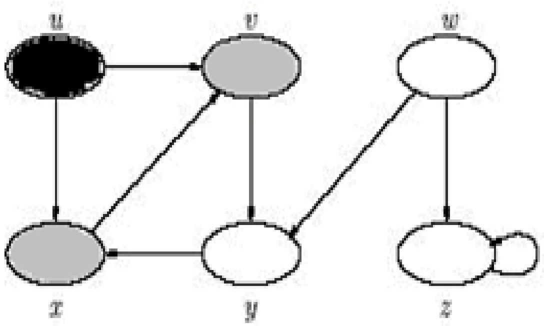
4. Busca em Largura

Exemplo: andamento de BFS em um dígrafo começando por "u"



Dígrafo e listas:

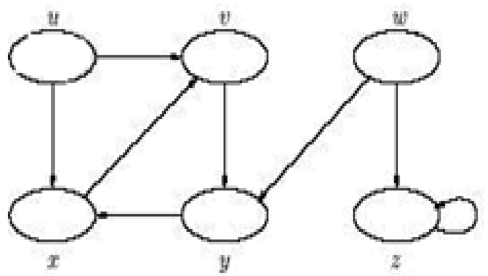
u	→	v	→	x
v	→	y		
w	→	z		
x	→	v		
y	→	x		
z	→	x		



(iv) Fila: v, x

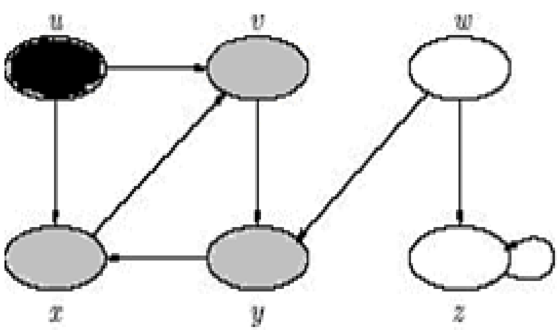
4. Busca em Largura

Exemplo: andamento de BFS em um dígrafo começando por "u"



Dígrafo e listas:

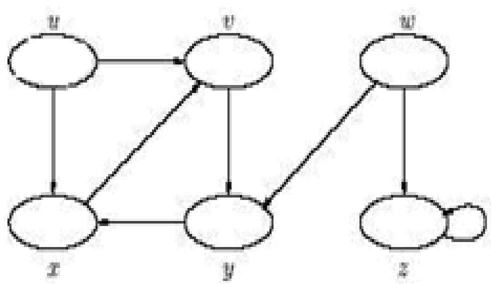
u	→	v	→	x
v	→	y		
w	→	z		
x	→	v		
y	→	x		
z	→	x		



(v) Fila: v, x, y

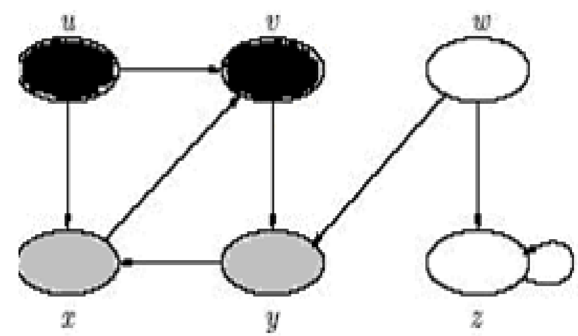
4. Busca em Largura

Exemplo: andamento de BFS em um dígrafo começando por "u"



Dígrafo e listas:

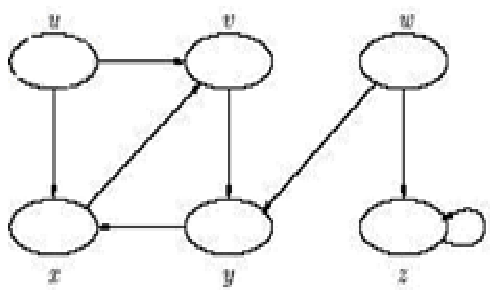
u	→	v	→	x
v	→	y		
w	→	z		
x	→	v		
y	→	x		
z	→	x		



(vi) Fila: x, y

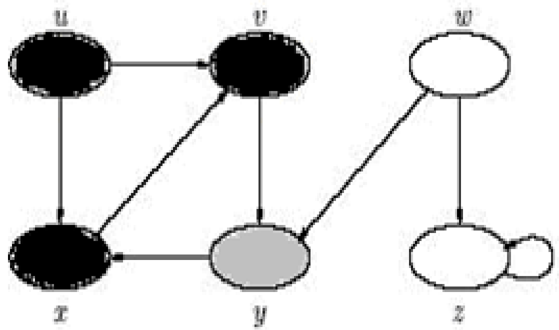
4. Busca em Largura

Exemplo: andamento de BFS em um dígrafo começando por "u"



Dígrafo e listas:

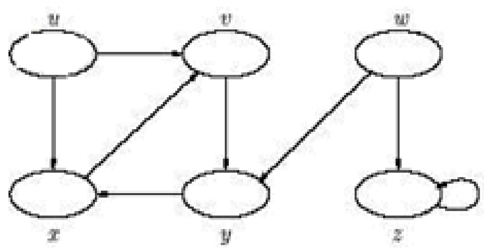
u	→	v	→	x
v	→	y		
w	→	z		
x	→	v		
y	→	x		
z	→	x		



(vii) Fila: y

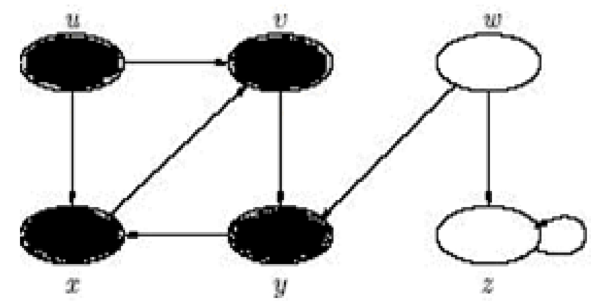
4. Busca em Largura

Exemplo: andamento de BFS em um dígrafo começando por "u"



Dígrafo e listas:

u	→	v	→	x
v	→	y		
w	→	z		
x	→	v		
y	→	x		
z	→	x		



(viii) Fila: ∅

Distâncias de u a cada vértice:

	[u]	[v]	[w]	[x]	[y]	[z]
dist:	0	1	∞	1	2	∞

4. Busca em Largura

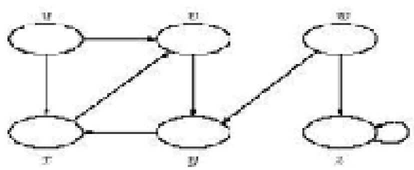
Arborescência da busca em largura

BFS(G,x)

```
para u ← 1 até n faça
        cor[u] ← BRANCO
        d[u] ← ∞
        pred(u) ← NIL
cor[x] ← CINZA
d[x] = 0
Q ← Inicialiaza-Fila(Q,x)
 enquanto Q ≠ Ø faça
          u ← Primeiro-da-Fila(Q)
          para cada v em Adj[u] faça
                   se cor[v] = BRANCO
                         então cor[v] ← CINZA
                                   dist[v] ← dist[u]+1
                                   pred(v) ← u
                                   Insira-na-Fila(Q,v)
          Remova-da-Fila(Q)
        cor[u] ← PRETO
  devolve dist[1..n], pred[1..n]
```

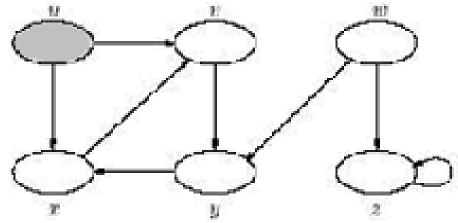
4. Busca em Largura

Exemplo:



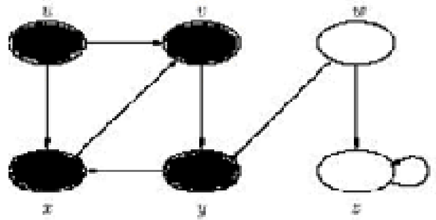
Dígrafo

⇒



(i) Fila: u

⇒

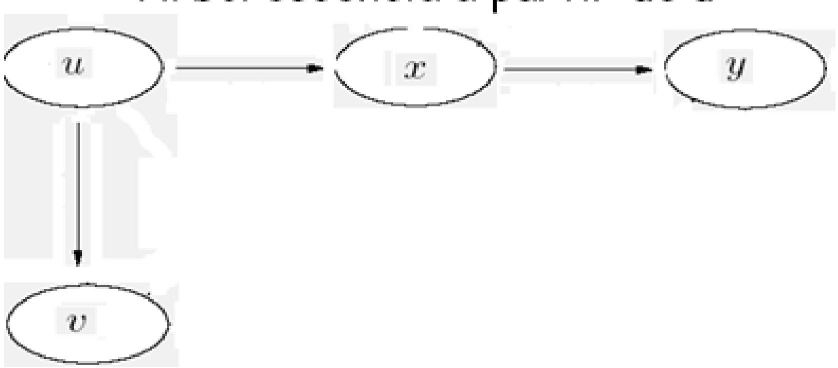


(viii) Fila: ∅

Configuração final dos vetores:

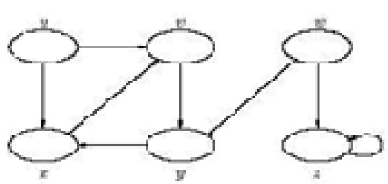
	[u]	[v]	[w]	[x]	[y]	[z]
pred:	-	u	-	u	x	-
dist:	0	1	∞	1	2	∞

Arborescência a partir de u



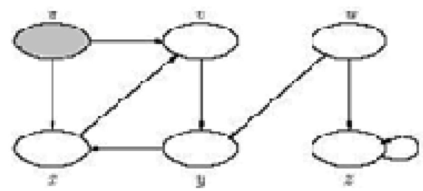
4. Busca em Largura

Exemplo: revendo o andamento do algoritmo BFS em um dígrafo começando a busca por "u"



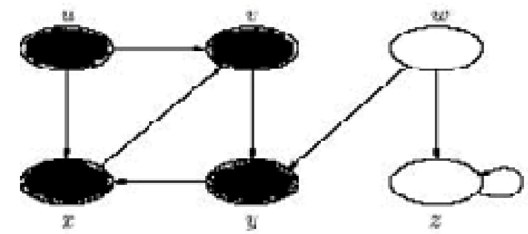
Dígrafo

⇒



(i) Fila: u

⇒

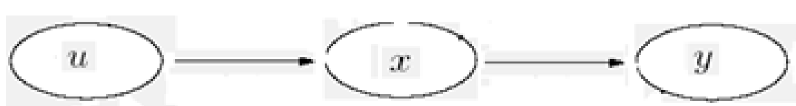


(viii) Fila: ∅

Configuração final dos vetores:

	[u]	[v]	[w]	[x]	[y]	[z]
pred:	-	u	-	u	x	-
dist:	0	1	∞	1	2	∞

Arborescência para o caminho u - y:



Que é impressa: u → x → y

+ exemplo ...

grafo