

# MAC 5711 - Análise de Algoritmos

Rodrigo Augusto Dias Faria  
Departamento de Ciência da Computação - IME/USP

7 de novembro de 2015

## Lista 7

**1. (CRLS 22.2-1)** Simule o funcionamento da BFS no grafo da Figura 22.2(a) do CLRS (segunda edição) a partir do vértice 3, determinando os valores de  $d$  e  $\pi$  para cada vértice.

**Resposta:** Inicializamos os atributos *color*,  $d$  e  $\pi$  de cada vértice  $v \in V$  do grafo com *WHITE*,  $\infty$  e *NIL*, respectivamente, conforme a primeira iteração do algoritmo BFS, exceto para o nó 3 que é o parâmetro  $s$  neste caso, cujo  $d = 0$  e sua cor é *GRAY*, conforme pode ser visto em **a**).

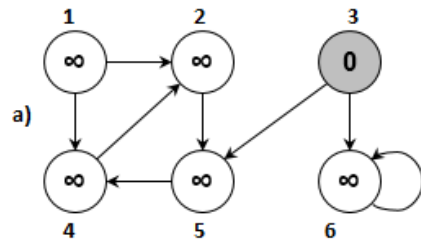
Colocamos  $s$  na fila e, então, visitamos a lista de adjacências de cada vértice a partir de  $s$ , sendo que  $u$  é o vértice retirado da pilha - aquele cuja cor é *BLACK* ao final do *for* da linha 12.

Note que o valor de  $d$  está dentro de cada vértice do grafo e, a cada novo vértice que descobrimos a partir de  $u$ , ou seja, aquele que ainda tem a cor *WHITE*, marcamos em seu atributo  $\pi$  o seu antecessor - o próprio  $u$  - que destacamos em laranja.

Note, também, que há casos em que nenhum novo vértice é descoberto, como em **d**), por exemplo.

Quando a pilha estiver vazia, concluímos a execução do algoritmo. Note que, neste caso, o vértice 1 não pode ser atingido a partir de 3 e, portanto, seu antecessor  $\pi$  fica marcado como *NIL*. O vértice  $s$  também tem seu antecessor *NIL*, já que é a entrada da busca no grafo e isso ocorre sempre.

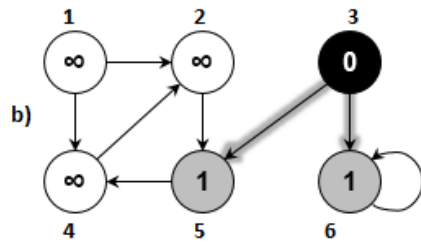
As arestas que estão destacadas formam a *breadth-first tree* e o antecessor de cada nó da árvore é dado pelo atributo  $\pi$  de cada vértice.



$Q$ 

3
---

$V$	1	2	3	4	5	6
$\pi$	nil	nil	nil	nil	nil	nil

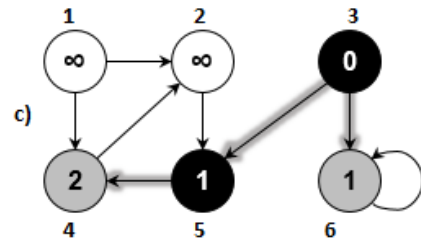


$Q$ 

5	6
---	---

$V$	1	2	3	4	5	6
$\pi$	nil	nil	nil	nil	3	3

$u = 3$



$Q$ 

6	4
---	---

$V$	1	2	3	4	5	6
$\pi$	nil	nil	nil	5	3	3

$u = 5$

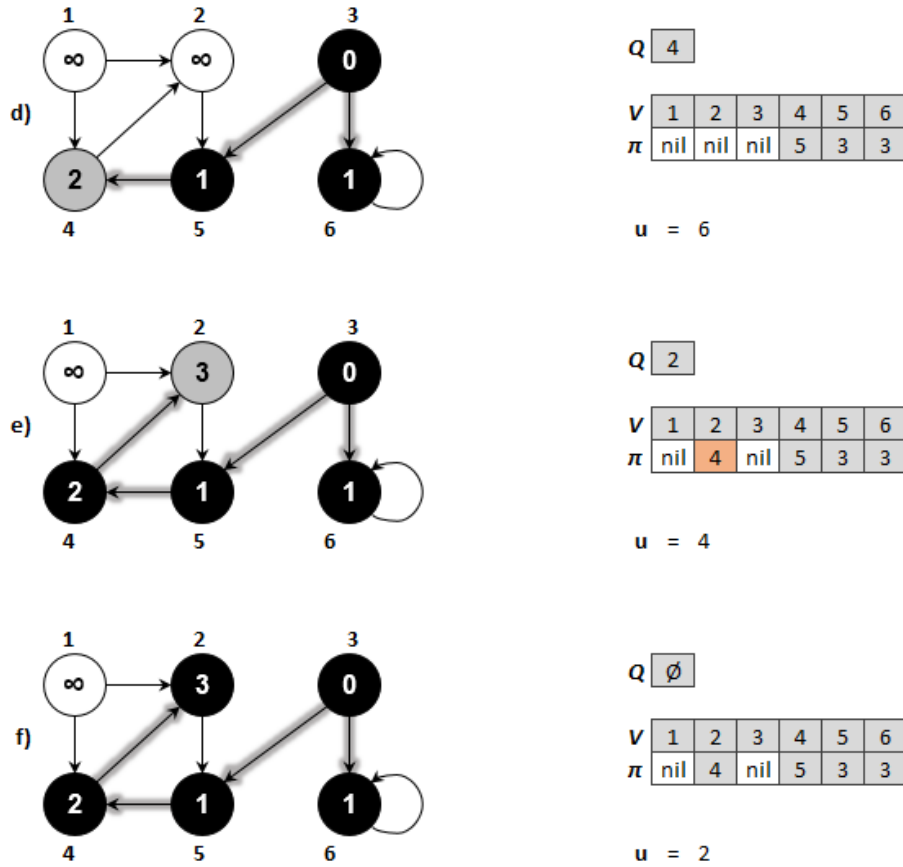
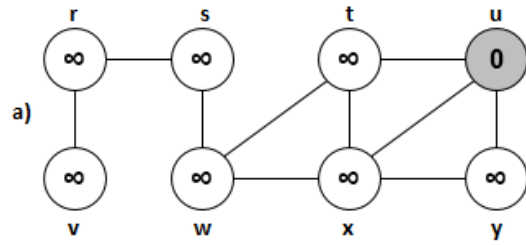


Figura 1: Sequência das operações do algoritmo BFS, sendo  $s = 3$ .

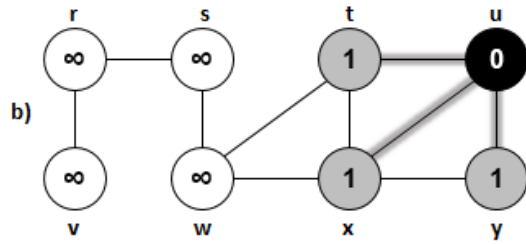
**2. (CRLS Ex. 22.2-2)** Simule o funcionamento da BFS no grafo da Figura 22.3 do CLRS (segunda edição) a partir do vértice  $u$ , determinando os valores de  $d$  e  $\pi$  para cada vértice.

**Resposta:** A mesma analogia aplicada na questão anterior pode ser utilizada aqui, mesmo tratando-se de um grafo não orientado, o algoritmo BFS funciona em ambos os casos, conforme vimos em sala/CLRS.



Q [ u ]

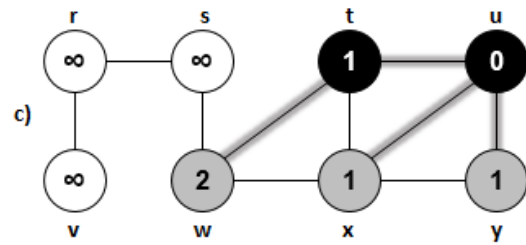
V	r	s	t	u	v	w	x	y
$\pi$	nil	nil	nil	nil	nil	nil	nil	nil



Q [ t x y ]

V	r	s	t	u	v	w	x	y
$\pi$	nil	nil	u	nil	nil	nil	u	u

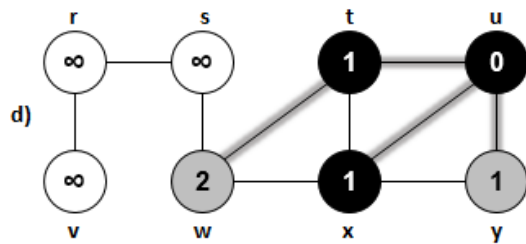
u = u



Q [ x y w ]

V	r	s	t	u	v	w	x	y
$\pi$	nil	nil	u	nil	nil	t	u	u

u = t



Q [ y w ]

V	r	s	t	u	v	w	x	y
$\pi$	nil	nil	u	nil	nil	t	u	u

u = x



Figura 2: Sequência das operações do algoritmo BFS, sendo  $s = u$ .

3. (CRLS 22.2-4) Argumente que o valor de  $d[u]$  atribuído ao vértice  $u$  na BFS é independente da ordem em que os vértices das listas de adjacências são dados. Por outro lado, mostre, usando o exemplo da Figura 22.3 do CLRS, que a árvore BFS depende da ordem dos vértices nas listas de adjacências.

Se tomarmos, por exemplo, o vértice  $u$  do exercício anterior, de qualquer forma que organizarmos a sua lista de adjacências ( $\{t, x, y\}$ ,  $\{x, y, t\}$ ,  $\{y, t, x\}$ ), o atributo  $d$  de cada um destes vértices sempre será 1, o que nos mostra que eles estão em um nível imediatamente abaixo de  $u$  na árvore.

i)

Graph structure:

- Nodes:  $r$  (4),  $s$  (3),  $t$  (1),  $u$  (0) in the top row;  $v$  (5),  $w$  (2),  $x$  (1),  $v$  (1) in the bottom row.
- Edges:  $(r,s)$ ,  $(s,t)$ ,  $(t,u)$ ,  $(r,v)$ ,  $(v,w)$ ,  $(w,x)$ ,  $(x,v)$ ,  $(s,w)$ ,  $(t,x)$ ,  $(u,v)$ .

Q  $\emptyset$

V	r	s	t	u	v	w	x	y
$\pi$	s	w	u	nil	r	x	u	u

u = v

**4. (CRLS 22.2-5)** Considere um grafo orientado  $D = (N, A)$ . Dê um exemplo de um conjunto  $A_\pi \subseteq A$  de arcos em  $D$  que formam uma árvore tal que, entre quaisquer dois nós  $u$  e  $v$  em  $D$ , o único caminho entre  $u$  e  $v$  em  $A_\pi$  é um caminho mínimo em  $D$  entre  $u$  e  $v$ , porém,  $A_\pi$  jamais seria produzida por uma execução da BFS em  $D$ , independente da ordem dos nós nas listas de adjacências de  $D$  e do nó inicial  $s$ .

```

graph LR
    s((s)) --> t((t))
    s((s)) --> u((u))
    t((t)) --> v((v))
    t((t)) --> x((x))
    u((u)) --> v((v))
    u((u)) --> x((x))

```

Note que, independente da ordem dos elementos adjacentes a  $s$ ,  $t$  e  $u$ , essa representação jamais poderá ser obtida pela BFS. A ordem das listas de adjacências dos elementos  $t$  e  $u$  também não influenciam na forma com que a árvore será gerada. A figura 5 mostra o resultado da execução da BFS, bem como a lista de adjacências em ordens diferentes em cada caso.

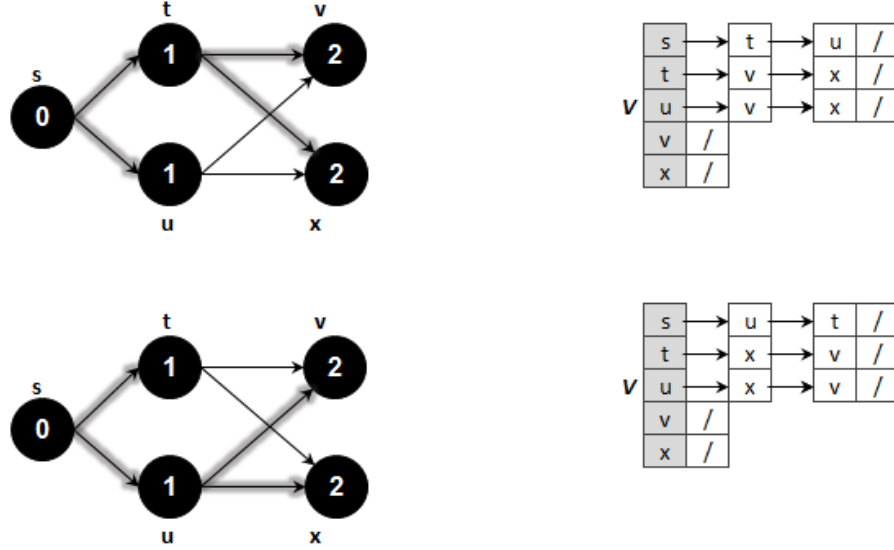


Figura 5: Execução da BFS no grafo orientado  $D = (N, A)$  com a lista de adjacências em duas ordens distintas.

5. Escreva uma versão não recursiva da busca em profundidade.

**Resposta:** Podemos utilizar uma pilha como apoio para aprofundar na lista de adjacências de cada nó da árvore, substituindo a recursão.

Além disso, nós usamos uma fila FIFO para a lista de adjacências de cada vértice  $u$  e, assim que um novo vértice  $v$  adjacente à  $u$  é encontrado e ele ainda não foi visitado, nós o visitamos e empilhamos  $v$  em  $S$ .

Note que nós somente retiramos  $u$  da pilha (linha 11 da DFS-VISIT) quando todos os vértices da lista de adjacências dele foram visitados, ou seja, o momento em que devemos marcar  $u$  como finalizado (sub-rotina BLACKEN).

Outro ponto importante é que, como visto na linha 12 da ITERATIVE-DFS, o vértice  $s$  que dá origem à busca tem seu ancestral como  $nil$ , o que garante o funcionamento da mesma forma que a DFS original. Isso também garante que a busca funciona nos casos em que tivermos mais de uma componente conexa no grafo.

**Consumo de tempo:** O *loop* das linhas 2-8 da ITERATIVE-DFS tomam  $\Theta(V)$  para inicializar cada vértice  $u \in V[G] + \Theta(E)$  para montar a fila de adjacências de cada vértice  $u$ . As sub-rotinas BLACKEN, GRAYEN, bem como as operações na fila/pilha tomam  $\Theta(1)$ . O *loop* das linhas 10-13 consome tempo  $\Theta(V)$ , ou seja, executa no máximo uma vez para cada vértice  $v \in G$ , já que DFS-VISIT é executado apenas em vértices que ainda não foram descobertos - àqueles que ainda são brancos. Como a fila de adjacências é visitada no máximo uma vez na sub-rotina DFS-VISIT e a soma do comprimento da fila de adjacências de cada vértice  $v$  é  $\Theta(E)$ , o tempo gasto na DFS-VISIT é  $O(E)$ .

Portanto, o consumo de tempo total será  $O(V + E)$ , o que mantém o comportamento assintótico original da DFS.

ITERATIVE-DFS( $G$ )

```
1  // let  $S$  be an empty stack
2  for each vertex  $u \in V[G]$ 
3       $color[u] = \text{WHITE}$ 
4       $\pi[u] = \text{NIL}$ 
5       $n = Adj[u].length$ 
6      for  $i = n$  to 1
7           $v = Adj[u][i]$ 
8          ENQUEUE( $Qadj[u], v$ )
9   $time = 0$ 
10 for each vertex  $u \in V[G]$ 
11     if  $color[u] == \text{WHITE}$ 
12         GRAYEN( $u, \text{NIL}$ )
13         DFS-VISIT( $u$ )
```

DFS-VISIT( $s$ )

```
1  PUSH( $S, s$ )
2  while  $S \neq \emptyset$ 
3       $u = \text{TOP}(S)$ 
4      if  $Qadj[u] \neq \emptyset$ 
5           $v = \text{DEQUEUE}(Qadj[u])$ 
6          if  $color[v] == \text{WHITE}$ 
7              GRAYEN( $v, u$ )
8              PUSH( $S, v$ )
9      else
10         BLACKEN( $u$ )
11         POP( $S$ )
```

GRAYEN( $v, u$ )

```
1   $color[v] = \text{GRAY}$ 
2   $time = time + 1$ 
3   $d[v] = time$ 
4   $\pi[v] = u$ 
```

BLACKEN( $u$ )

```
1   $color[u] = \text{BLACK}$ 
2   $time = time + 1$ 
3   $f[u] = time$ 
```

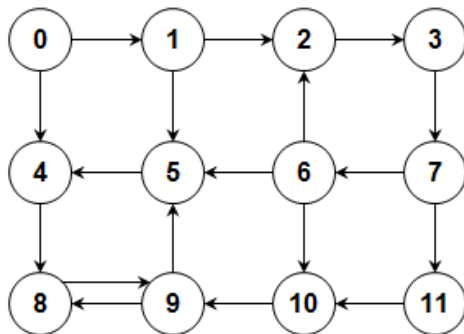
6. Execute uma busca em profundidade a partir do vértice 0 no grafo orientado dado pelas listas de adjacência a seguir. Exiba o rastreamento da busca.

```
0: 1 4
1: 2 5
2: 3
```

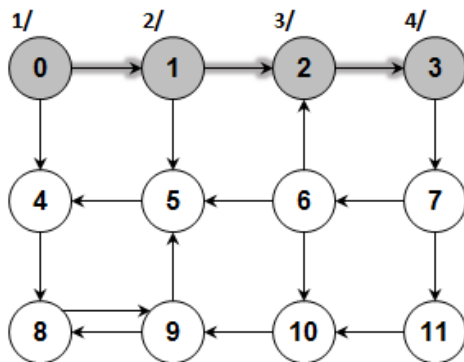


3: 7  
 4: 8  
 5: 4  
 6: 5 10 2  
 7: 11 6  
 8: 9  
 9: 5 8  
 10: 9  
 11: 10

**Resposta:** A figura 6 mostra o grafo formado pela lista de adjacências dada, bem como o rastreamento no momento em que o vértice 3, 6 e 8 são descobertos, e o resultado final da DFS ao final de todas as chamadas recursivas, respectivamente. Os tempos  $d$  e  $f$  estão no vetor à direita e, também, acima de cada vértice.



$v$	0	1	2	3	4	5	6	7	8	9	10	11
$\pi$	nil	nil	nil	nil	nil	nil	nil	nil	nil	nil	nil	nil
$d$												
$f$												



$v$	0	1	2	3	4	5	6	7	8	9	10	11
$\pi$	nil	0	1	2	nil	nil	nil	nil	nil	nil	nil	nil
$d$	1	2	3	4								
$f$												

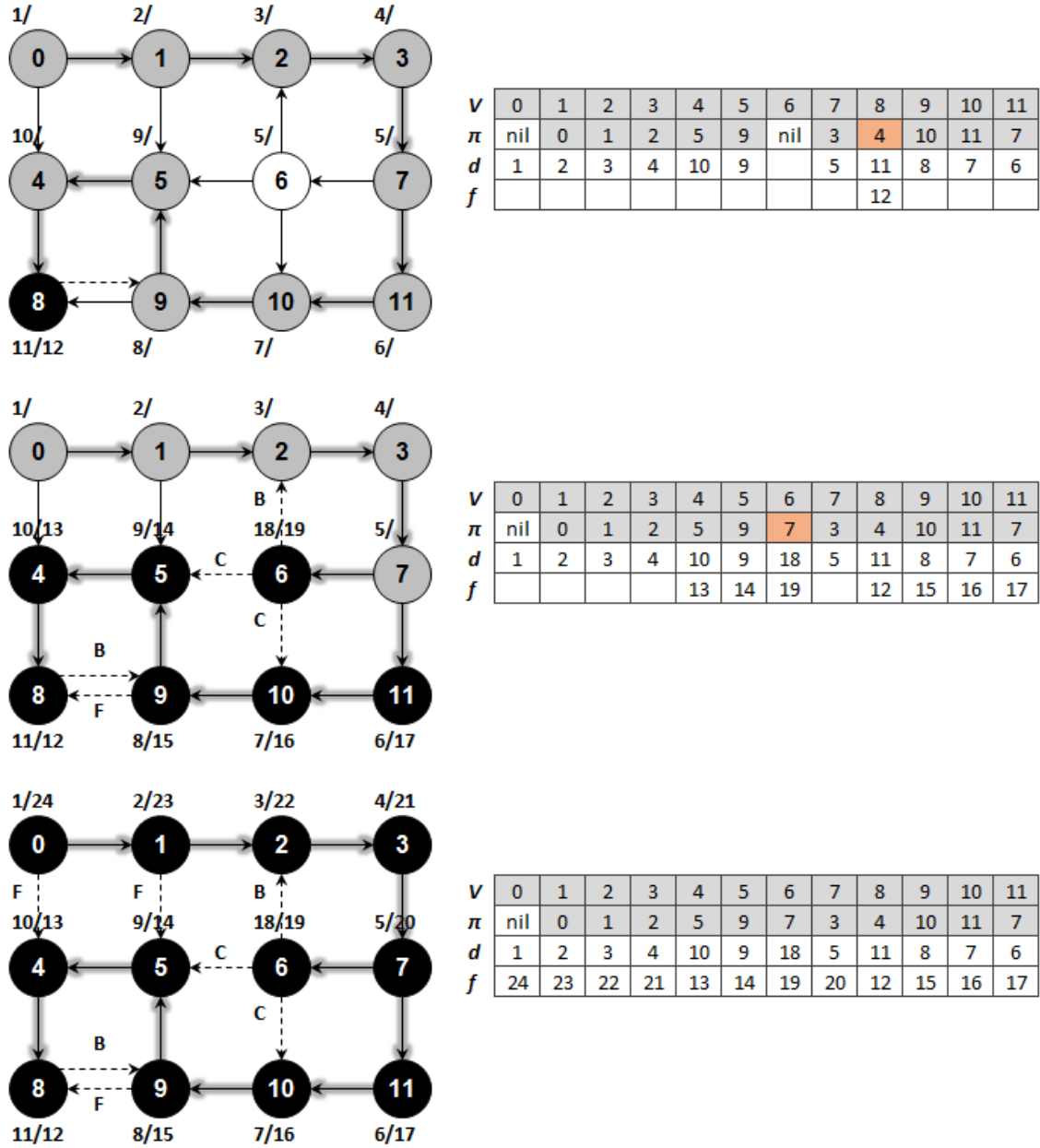


Figura 6: Rastreamento da DFS na lista de adjacências dada.

**7. (CRLS 22.3-1)** Desenhe uma tabela  $3 \times 3$ , com as linhas e colunas indexadas pelas cores branco, cinza e preto. Em cada entrada  $(i, j)$ , indique se, em qualquer ponto durante uma DFS de um grafo orientado, pode existir um arco de um nó de cor  $i$  para um nó de cor  $j$ . Para cada arco possível, indique as classificações que ele pode ter (de árvore, de retorno, para frente, cruzado). Faça um segundo quadro considerando um grafo não orientado.

As tabelas 1 e 2 mostram as classificações dos arcos para o grafo orientado e não orientado, respectivamente.

As siglas significam *Tree*, *Back*, *Cross* e *Forward edge*.

	White	Gray	Black
White	x	x	x
Gray	T	B	F/C
Black	x	x	B

Tabela 1: Classificação dos arcos no grafo orientado para a DFS.

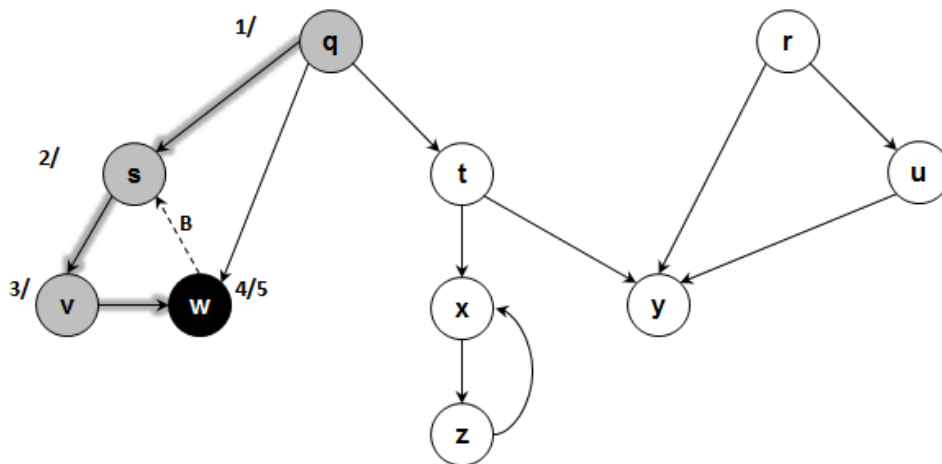
	White	Gray	Black
White	x	x	x
Gray	T	B	x
Black	x	B	B

Tabela 2: Classificação dos arcos no grafo não orientado para a DFS.

**8. (CRLS 22.3-2)** Mostre como a DFS funciona no grafo da Figura 22.6 do CLRS (segunda edição). Assuma que o laço das linhas 5-7 da DFS visitam os vértices em ordem alfabética, e que os vértices se encontram em ordem alfabética nas listas de adjacências. Mostre os valores de  $d$  e  $f$  para cada vértice ao final da DFS.

**Resposta:** A figura 7 mostra o rastreamento da DFS em 3 momentos distintos: quando todos os vértices adjacentes à  $w$  são visitados, todos os vértices adjacentes à  $t$  são visitados e a árvore com todas as chamadas recursivas concluídas, respectivamente.

Os valores  $d$  e  $f$ , bem como o antecessor de cada vértice dado por  $\pi$  estão na tabela de rastreamento abaixo de cada imagem. Os tipos de arestas também estão devidamente destacados.



$v$	q	r	s	t	u	v	w	x	y	z
$\pi$	nil	nil	q	nil	nil	s	v	nil	nil	nil
$d$	1		2			3	4			
$f$							5			

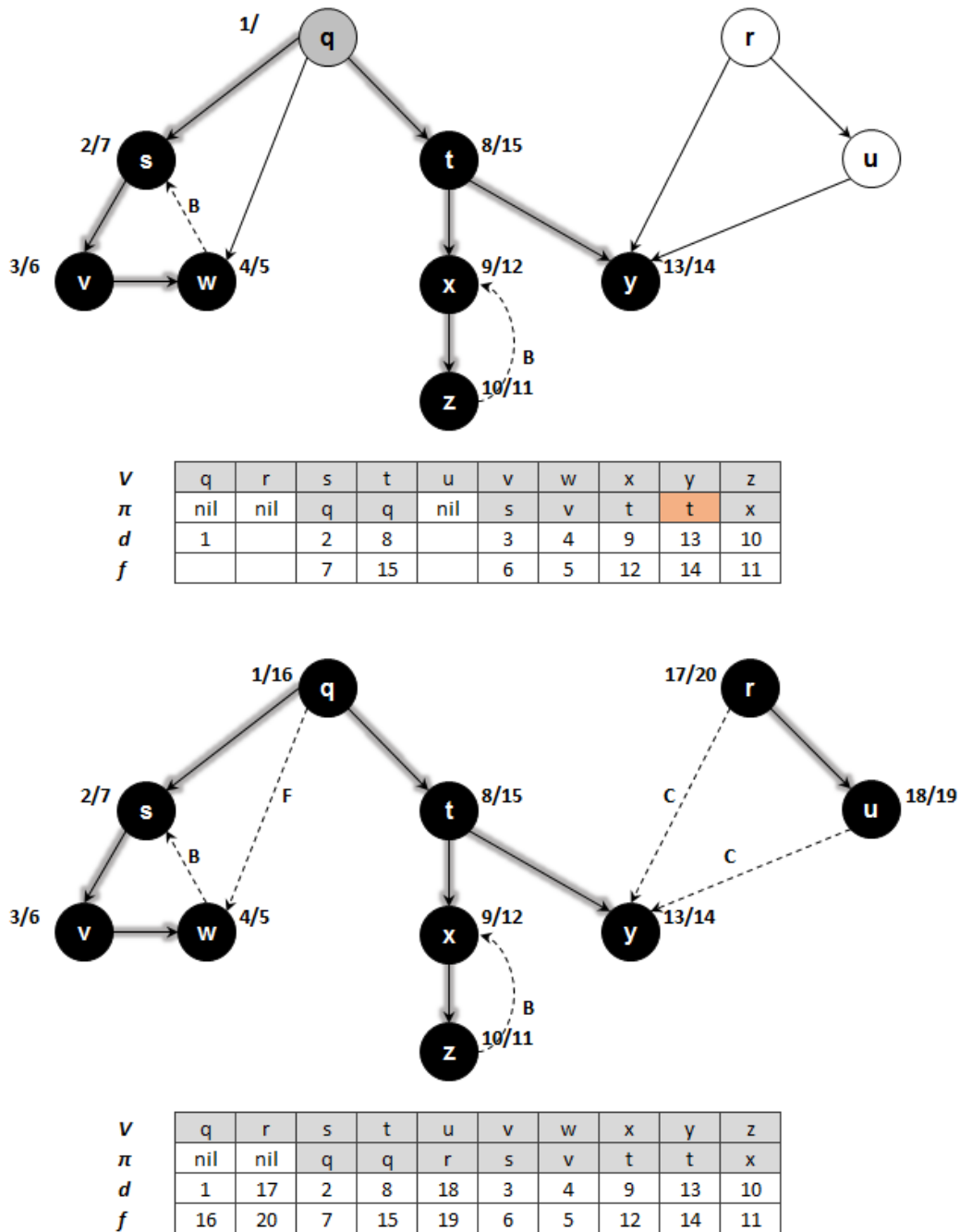


Figura 7: Rastreamento da DFS na figura 22.6 do CLRS.

**9. (CRLS 22.3-7)** Mostre um contraexemplo para a conjectura que se existe um caminho de  $u$  a  $v$  em um grafo orientado  $G$ , e se  $d[u] < d[v]$  numa DFS de  $G$ , então  $v$  é descendente de  $u$  na floresta DFS produzida.

Podemos observar a própria árvore à esquerda da figura 22.5 (c) do CLRS como um contraexemplo. Seja a DFS produzida a partir de  $s$ , se tomarmos os vértices  $x$  e  $w$ ,  $d[x] < d[w]$ , existe um caminho de  $x$  a  $w$ , mas  $w$  não é descendente de  $x$ .

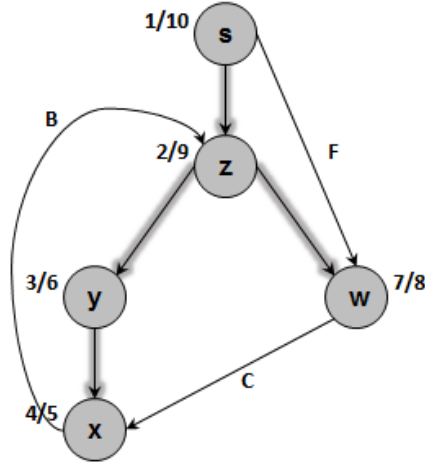


Figura 8: Contraexemplo utilizando parte do grafo da figura 22.5 (c) do CLRS.

**10. (CRLS 22.3-8)** Mostre um contraexemplo para a conjectura que se existe um caminho de  $u$  a  $v$  em um grafo orientado  $G$ , então qualquer DFS deve resultar em  $d[v] \leq f[u]$ .

A figura 9 mostra um contraexemplo da conjectura. Temos um caminho de  $u$  a  $v$  no grafo, porém, aplicando a DFS a partir de  $s$ , temos que  $d[v] > f[u]$ .

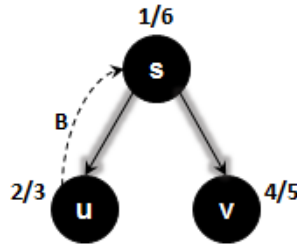


Figura 9: Contraexemplo da conjectura dada.

**11. (CRLS 22.3-10)** Mostre como um vértice  $u$  num grafo orientado pode terminar sozinho numa árvore de uma floresta DFS mesmo tendo arcos saindo e entrando dele em  $G$ .

A figura 10 mostra um exemplo onde o vértice  $u$  pode ficar sozinho numa árvore de uma floresta DFS gerada a partir do vértice  $s$ .

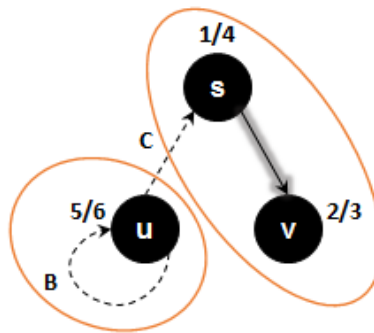


Figura 10: Exemplo onde o vértice  $u$  pode ficar isolado.

Um outro exemplo pode ser visto na figura 11, sendo que cada árvore é formada por um único vértice, já que a busca começa em  $v$ .

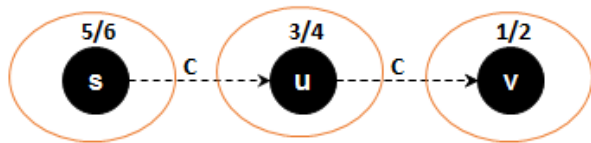


Figura 11: Segundo exemplo onde o vértice  $u$  pode ficar isolado.

## CLRS (Outros)

A.1-7 Avalie o produtório  $\prod_{k=1}^n 2(4^k)$ .

$$\prod_{k=1}^n 2(4^k) = \prod_{k=1}^n 2((2^2)^k) = \prod_{k=1}^n 2(2^{2k}) = \prod_{k=1}^n 2^{2k+1}$$

Se avaliarmos o produtório para  $n = 3$ , por exemplo:

$$\prod_{k=1}^3 2^{2k+1} = 2^{2+1} \times 2^{4+1} \times 2^{6+1}$$

Percebemos que o expoente de 2 cresce em uma série aritmética:

$$\sum_{k=1}^n 2k + 1 = \sum_{k=1}^n 2k + \sum_{k=1}^n 1 = 2 \sum_{k=1}^n k + n = 2\left(\frac{n(n+1)}{2}\right) + n = n(n+2)$$

Portanto:

$$\prod_{k=1}^n 2(4^k) = 2^{n(n+2)}$$