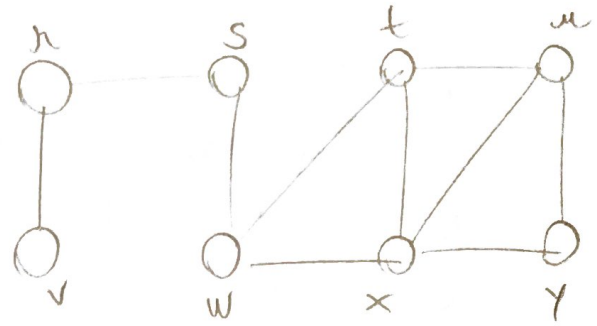
* Busca em Largura (BfS)
  - breadth-first search

```
BFS(G,S)        // G → grafo, S → vértice inicial
1.  para cada vértice u ∈ V[G]-{s}
2.      u.cor = Branco   // inicializa os vértices
3.      u.d = ∞
4.      u.pai = NULL
5.  s.cor = cinza
6.  s.d = 0
7.  s.pai = NULL
8.  Q = ∅
9.  EnFileira(Q,s)
10. Enquanto Q ≠ ∅
11.     u = DesenFileira(Q)
12.     para cada v = Adj[u]
13.         se v.cor == branco   // descobre novo vértice
14.             v.cor = cinza
15.             v.d = u.d + 1
16.             v.pai = u
17.             EnFileira(Q,v)
18.     u.cor = Preto
```

Exemplo:

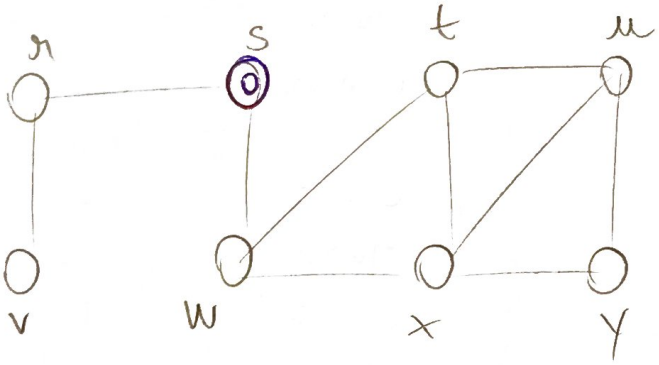
G:



vértice	cor	d	Pai
r	branco	∞	NULL
s	cinza	0	NULL
t	branco	∞	NULL
u	branco	∞	NULL
v	branco	∞	NULL
w	branco	∞	NULL
x	branco	∞	NULL
y	branco	∞	NULL

→ Vértice Inicial = S

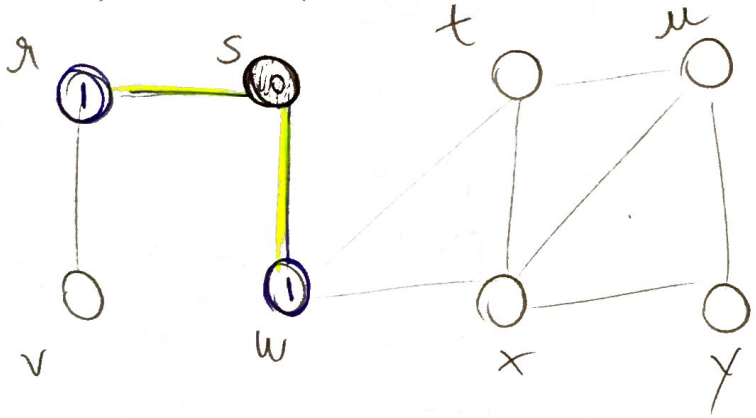
passo 01:



Q [s]
0

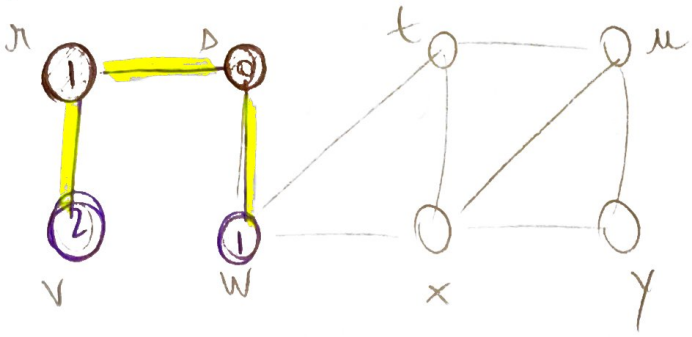
Vértice	cor	d	pai
r	cinza	1	S
s	preto	0	NULL
t	branco	∞	NULL
u	branco	∞	NULL
v	branco	∞	NULL
w	cinza	1	S
x	branco	∞	NULL
y	branco	∞	NULL

passo 02:
- remove S
- enfileira r, w



Q [r w]
1 1

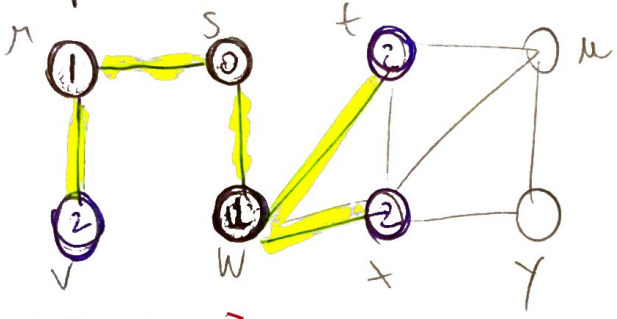
passo 03: remove r, add v



Q[w, v]
1 2

r	preto	1	s
s	preto	0	NIL
t	branco	∞	NIL
u	branco	∞	NIL
v	cinza	2	r
w	cinza	1	s
x	branco	∞	NIL
y	branco	∞	NIL

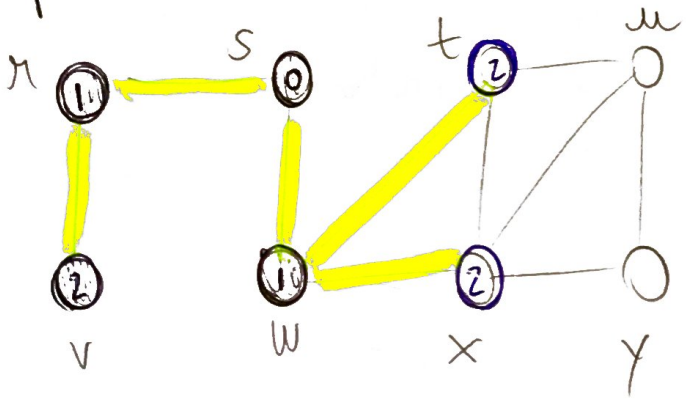
• passo 04: remove w, add t, x



Q[v, t, x]
2 2 2

r	preto	1	s
s	preto	0	NIL
t	cinza	2	w
u	branco	∞	NIL
v	cinza	2	r
w	preto	1	s
x	cinza	2	w
y	branco	∞	NIL

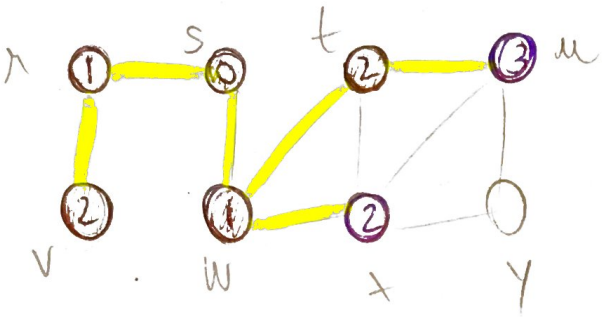
• passo 05: remove v



Q[t, x]
2 2

r	preto	1	s
s	preto	0	NIL
t	cinza	2	w
u	branco	∞	NIL
v	preto	2	r
w	preto	1	s
x	cinza	2	w
y	branco	∞	NIL

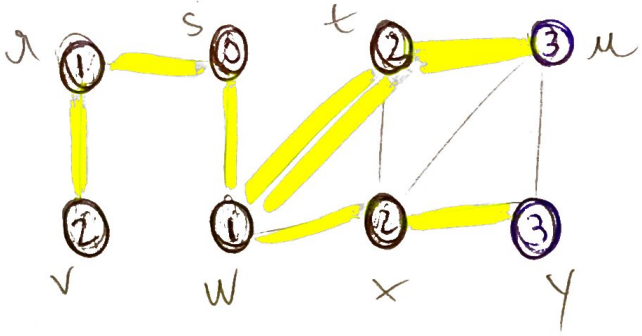
• passo 06: remove t, add u



Q [x, u]
2 3

r	preto	1	s
s	preto	0	NIL
t	preto	2	w
u	cinza	3	t
v	preto	2	r
w	preto	1	s
x	cinza	2	w
y	branco	∞	NIL

• passo 07: remove x, add y

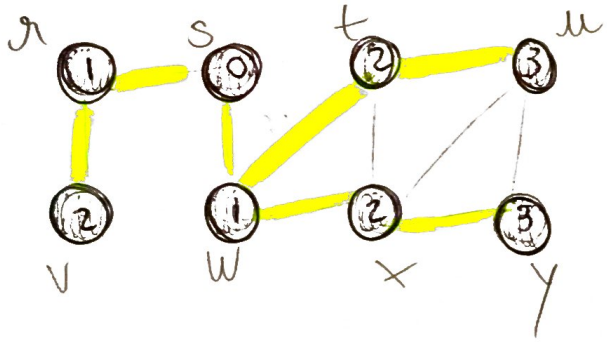


Q [u, y]
3 3

r	preto	1	s
s	preto	0	NIL
t	preto	2	w
u	cinza	3	t
v	preto	2	r
w	preto	1	s
x	preto	2	w
y	cinza	3	x

• passo 08: remove u

• passo 09: remove y



r	preto	1	s
s	preto	0	NIL
t	preto	2	w
u	preto	3	t
v	preto	2	r
w	preto	1	s
x	preto	2	w
y	preto	3	x

- análise de complexidade

→ $G_1 = (V, E) \Rightarrow O(V+E)$

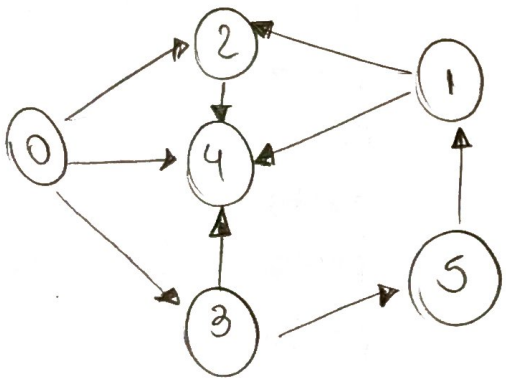
ou seja, a busca em largura é executada em tempo linear em relação ao tamanho da representação por lista de adjacências de G_1.

*Mostrar Percurso

// Bfs já deve ter sido calculada

```
Print-path(G, s, v)
1. Se v == s
2.     Imprimir s
3. Senão se v.pai == NIL
4.     Imprimir "não existe caminho"
5. Senão Print-Path(G, s, v.pai)
6.     Imprimir v;
```

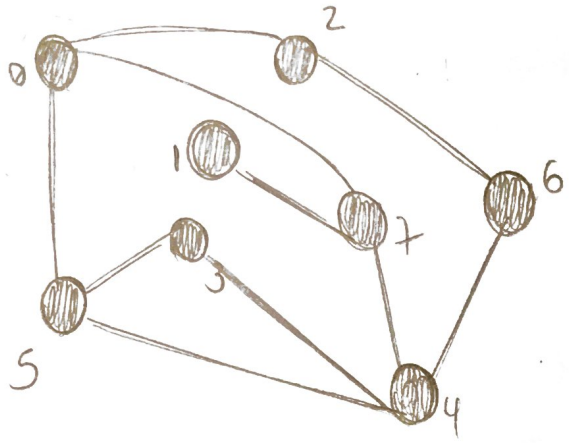
*Exemplo (Grafo dirigido)



Vértice inicial = 0

• exercício 2:

Vértice inicial = 0

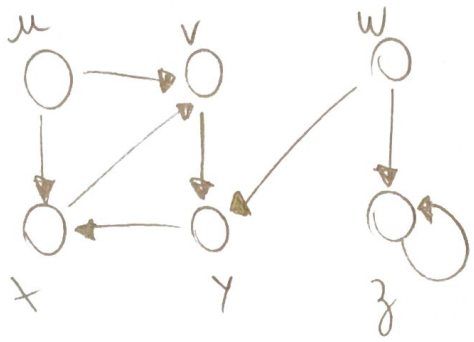


* Busca Em Profundidade (Dfs)

```
DFS(G)                                  // função principal
1.  para cada vértice u ∈ V[G]
2.      u.cor = branco
3.      u.pai = NULL
4.  tempo = 0
5.  para cada vértice u ∈ V[G]
6.      se u.cor == branco
7.          Dfs-visit(G,u)
```

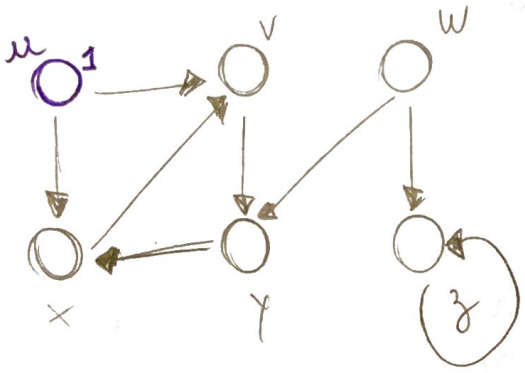
```
DFS-visit(G,u)                          // função auxiliar
1.  tempo = tempo + 1                   // vértice branco u acabou de ser descoberto
2.  u.d = tempo
3.  u.cor = cinza
4.  para cada v ∈ Adj[u]                // explorar aresta (u,v)
5.      se v.cor == branco
6.          v.pai = u
7.          Dfs-visit(G,v)
8.  u.cor = Preto                       // pintar u de preto, está finalizado
9.  tempo = tempo + 1
10. u.f = tempo
```

- Exemplo:



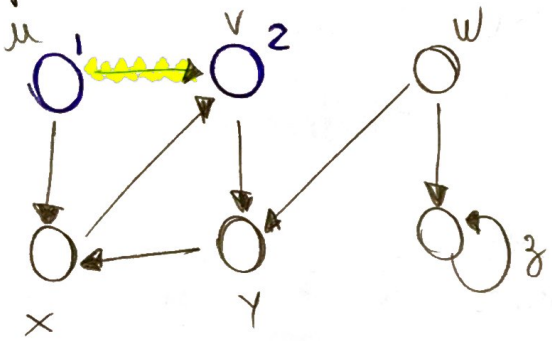
vértice	cor	d	f	pai
u	branco	∞	∞	NIL
v	branco	∞	∞	NIL
w	branco	∞	∞	NIL
x	branco	∞	∞	NIL
y	branco	∞	∞	NIL
z	branco	∞	∞	NIL

- passo 01:



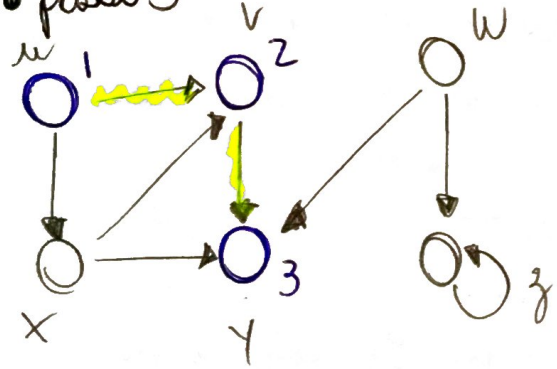
u	cinza	1	∞	NIL
v	branco	∞	∞	NIL
w	branco	∞	∞	NIL
x	branco	∞	∞	NIL
y	branco	∞	∞	NIL
z	branco	∞	∞	NIL

- passo 2:

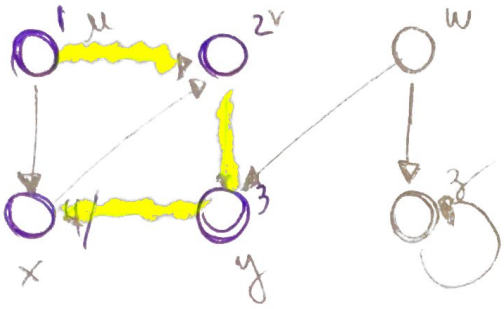


u	cinza	1	∞	NIL
v	cinza	2	∞	u
w	branco	∞	∞	NIL
x	branco	∞	∞	NIL
y	branco	∞	∞	NIL
z	branco	∞	∞	NIL

- passo 3

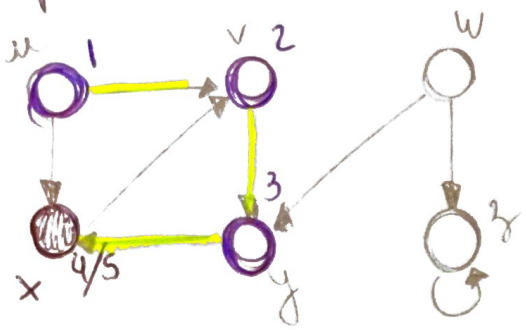


u	cinza	1	NIL
v	cinza	2	u
w	branco	∞	NIL
x	branco	∞	NIL
y	cinza	3	v
z	branco	∞	NIL



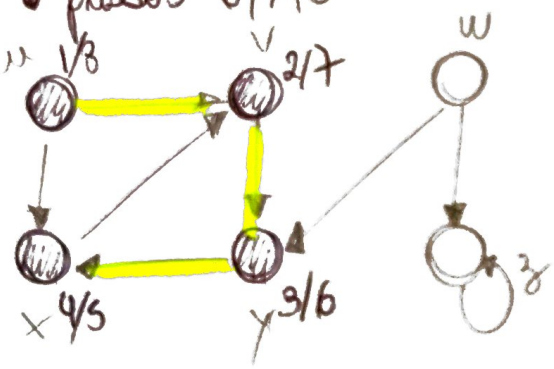
u	cinza	1	∞	NIL
v	cinza	2	∞	u
w	branco	∞	∞	NIL
x	cinza	4	∞	y
y	cinza	3	∞	v
z	branco	∞	∞	NIL

- passo 5



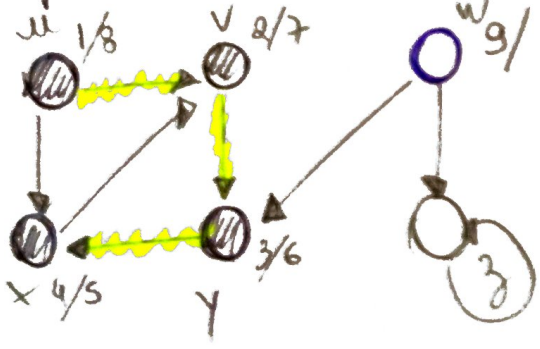
u	cinza	1	∞	NIL
v	cinza	2	∞	u
w	branco	∞	∞	NIL
x	preto	4	5	y
y	cinza	3	∞	v
z	branco	∞	∞	NIL

- passos 6,7,8



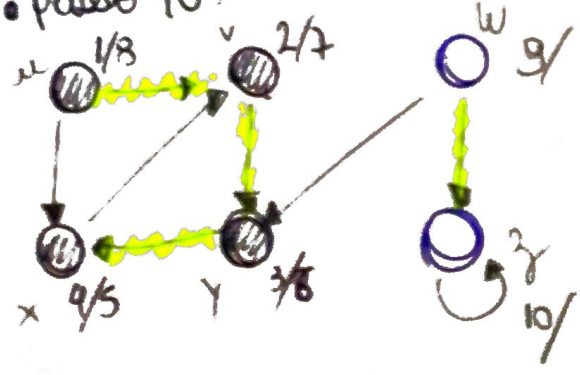
u	preto	1	8	NIL
v	preto	2	7	u
w	branco	∞	∞	NIL
x	preto	4	5	y
y	preto	3	6	v
z	branco	∞	∞	NIL

- passo 9:



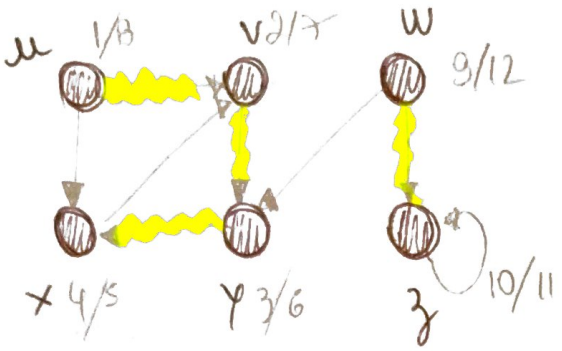
u	preto	1	8	NIL
v	preto	2	7	u
w	cinza	9	∞	NIL
x	preto	4	5	y
y	preto	3	6	v
z	branco	∞	∞	NIL

- passo 10:



u	preto	1	8	NIL
v	preto	2	7	u
w	cinza	9	∞	NIL
x	preto	4	5	y
y	preto	3	6	v
z	cinza	10	∞	w

- passos 11 e 12



u	preto	1	8	NIL
v	preto	2	7	u
w	preto	9	12	NIL
x	preto	4	5	y
y	preto	3	6	v
z	preto	10	11	w

- análise de complexidade

$\Theta(V+E)$

- tipos de arestas
  - aresta de árvore: são arestas na floresta em profundidade
  - aresta de retorno: (u,v) conectam um vértice u a um ancestral em uma árvore de profundidade. Laços contam.
  - aresta direta: são arestas (u,v) não de árvore que conectam um vértice u a um descendente v em árvore de profundidade.
  - aresta cruzada: as demais.

* Dfs consegue classificar arestas:

↳ ao explorar (u,v) pela primeira vez, a cor do vértice define o tipo de aresta:

1. Branco ⇒ aresta de árvore
2. Cinza ⇒ aresta de retorno
3. Preto ⇒ aresta cruzada

- Bfs × Dfs

→ bfs usa "fila", dfs usa "pilha" ou recursão

→ bfs: usuário escolhe o vértice inicial
dfs: algoritmo escolhe o vértice inicial

→ dfs visita todos os vértices, bfs apenas os alcançáveis a partir de s

→ bfs iterativa × dfs recursiva.

- Exercício: rodar Dfs em