Curso Estruturas de Dados e Algoritmos Expert

Prof. Nelio Alves

Grafos (parte 3)

DEVSUPERIOR

https://devsuperior.com.br

Percorrendo um grafo

- Percorrer um grafo é um **problema fundamental**
- Temos que ter uma forma **sistemática** de visitar arestas e vértices
- O algoritmo deve ser suficientemente **flexível** para se adequar à diversidade de grafos

Percorrendo um grafo

- Dois algoritmos comuns
  - Busca em largura (BFS)
  - Busca em profundidade (DFS)

- Abordagem comum
  - Marcar vértices
    - Visitados, não visitados, processados
  - Usar estrutura de dados de apoio para processar vértices ainda não visitados

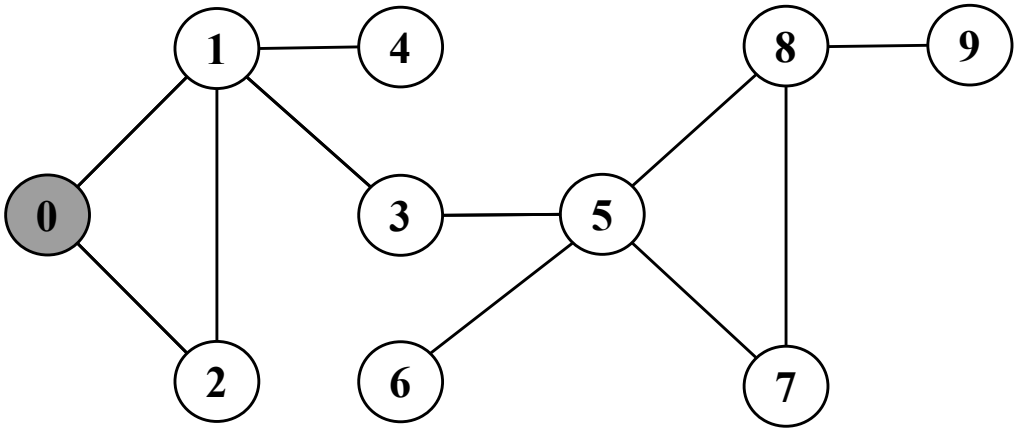
Busca em Largura (BFS)

https://devsuperior.com.br

Prof. Dr. Nelio Alves

Intuição da BFS

Percorrer os vértices mais próximos primeiro

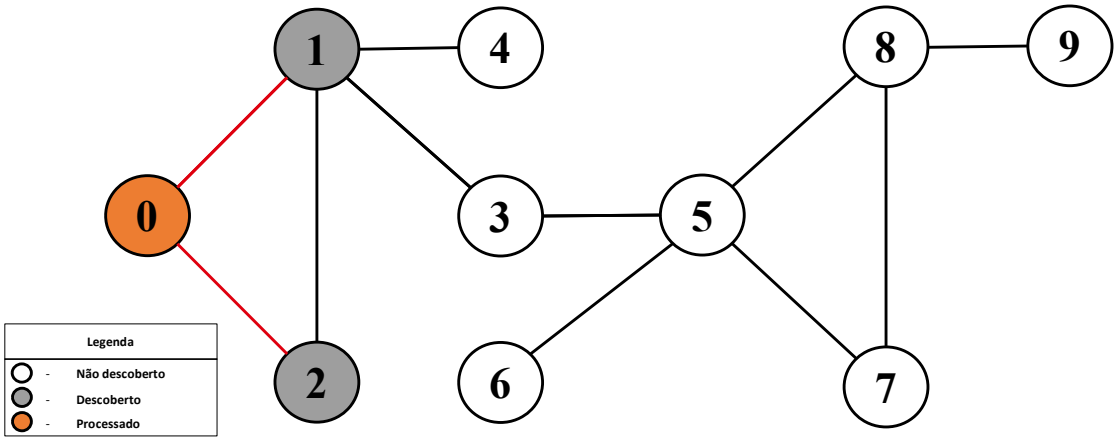


Nó inicial: 0

5

Intuição da BFS

Percorrer os vértices mais próximos primeiro

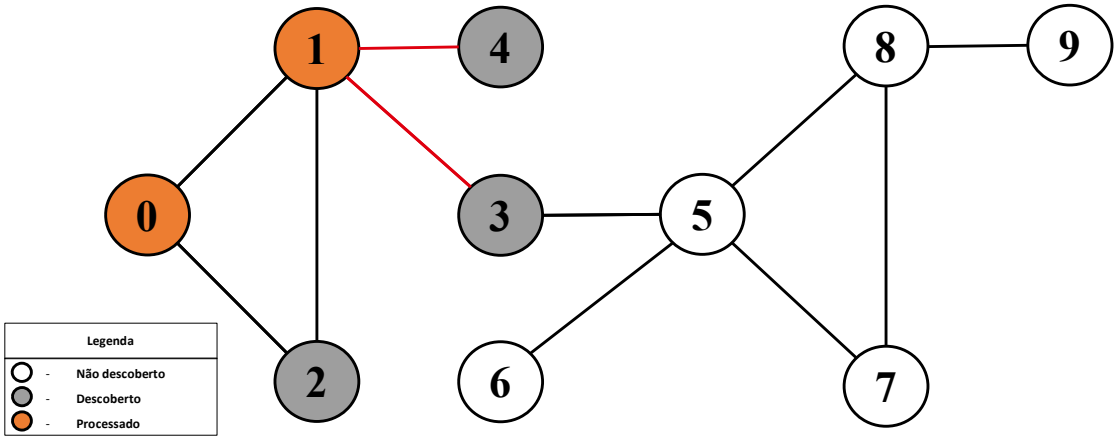


Marca todos os nós não visitados adjacentes a **0**: 1 e 2

6

Intuição da BFS

Percorrer os vértices mais próximos primeiro

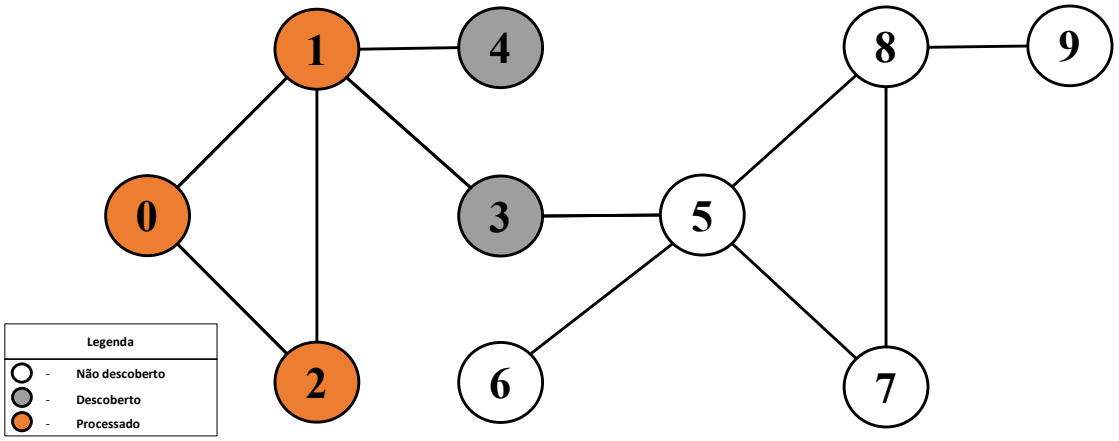


Marca todos os nós não visitados adjacentes a **1**: 3 e 4

7

Intuição da BFS

Percorrer os vértices mais próximos primeiro

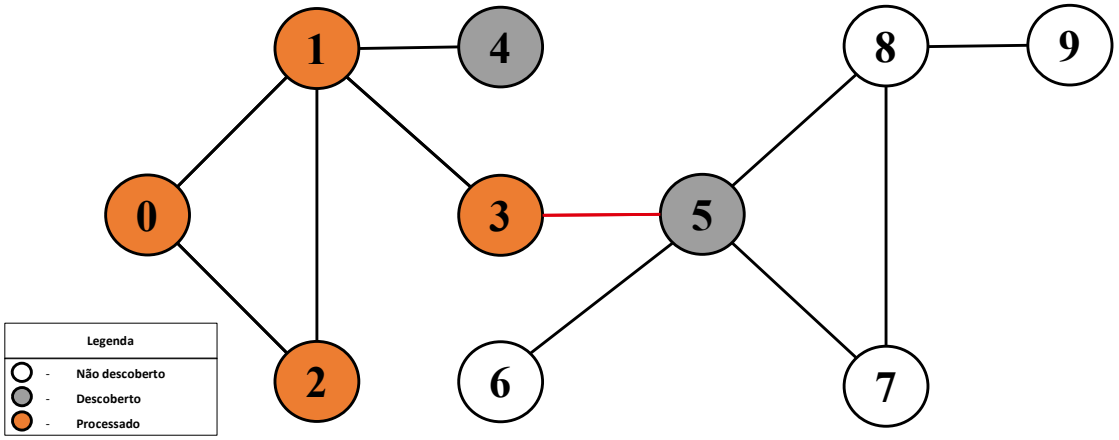


Marca todos os nós não visitados adjacentes a **2**: nenhum

8

Intuição da BFS

Percorrer os vértices mais próximos primeiro

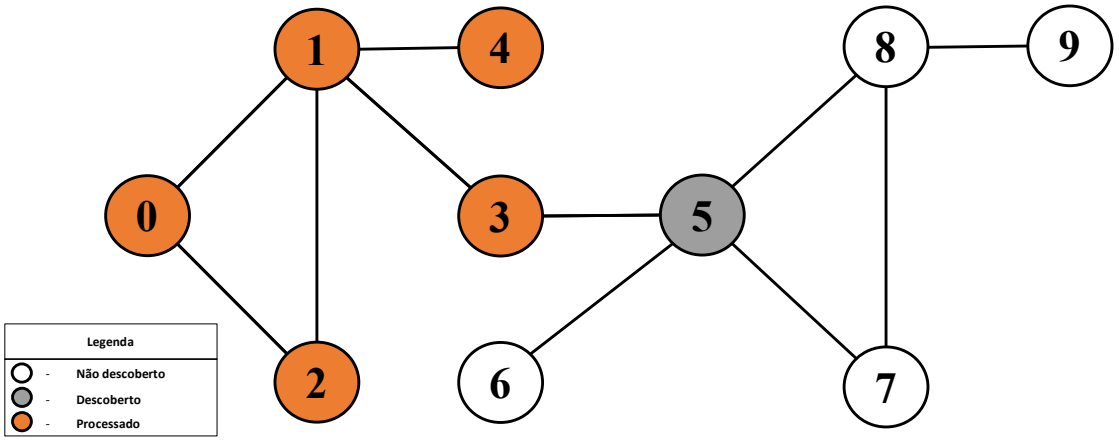


Marca todos os nós não visitados adjacentes a **3**: 5

9

Intuição da BFS

Percorrer os vértices mais próximos primeiro

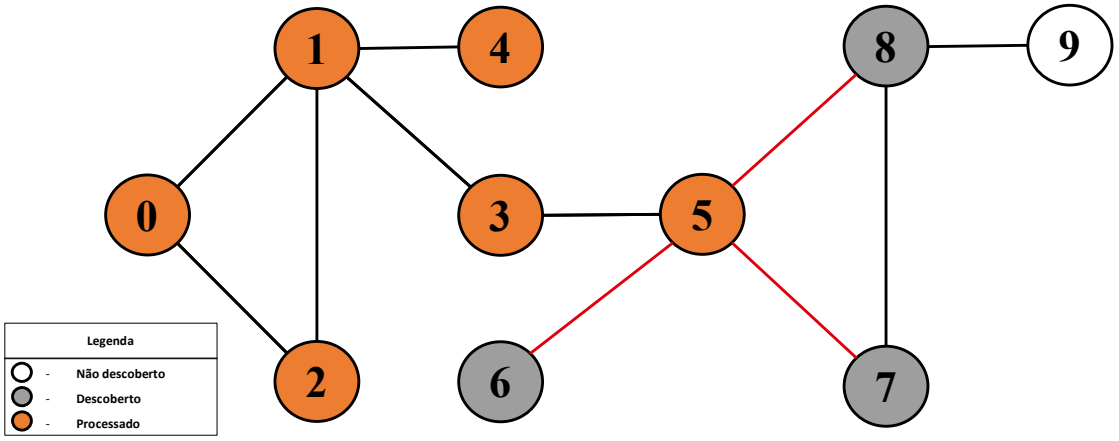


Marca todos os nós não visitados adjacentes a **4**: nenhum

10

Intuição da BFS

Percorrer os vértices mais próximos primeiro

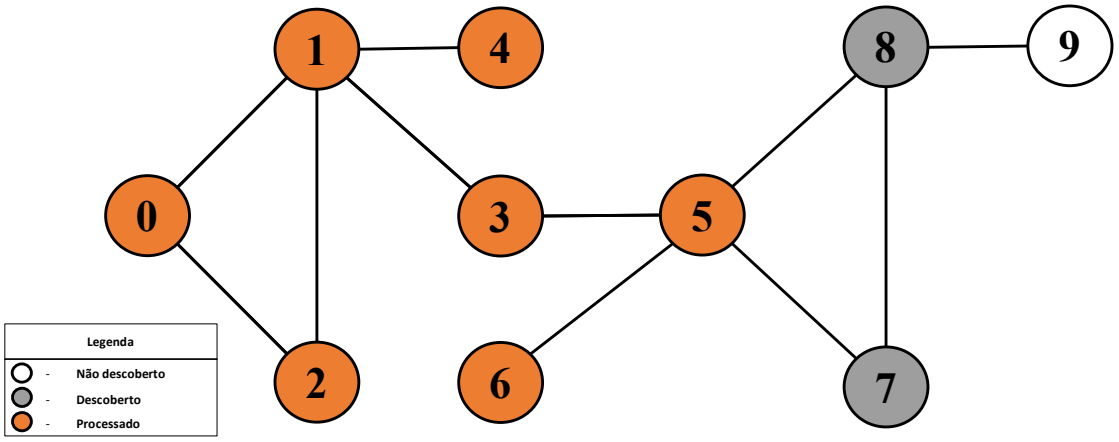


Marca todos os nós não visitados adjacentes a **5**: 6, 8 e 7

11

Intuição da BFS

Percorrer os vértices mais próximos primeiro

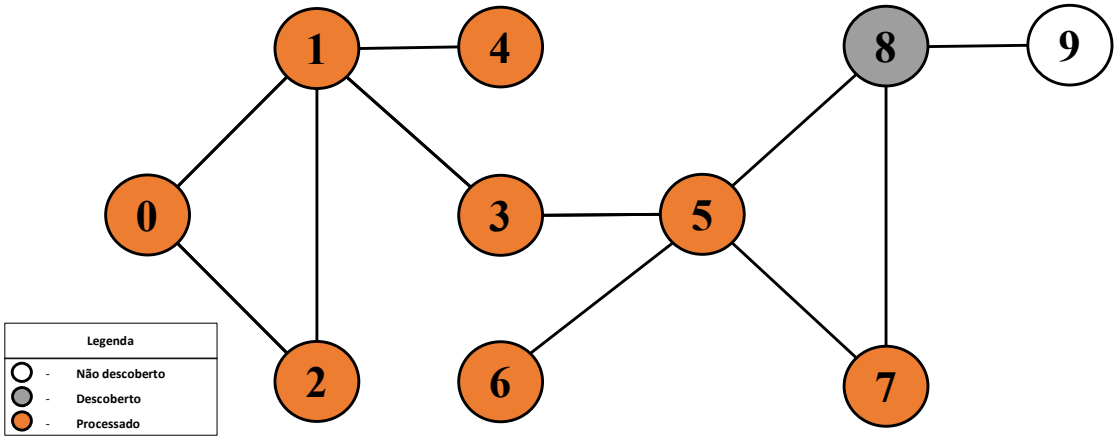


Marca todos os nós não visitados adjacentes a **6**: nenhum

12

Intuição da BFS

Percorrer os vértices mais próximos primeiro

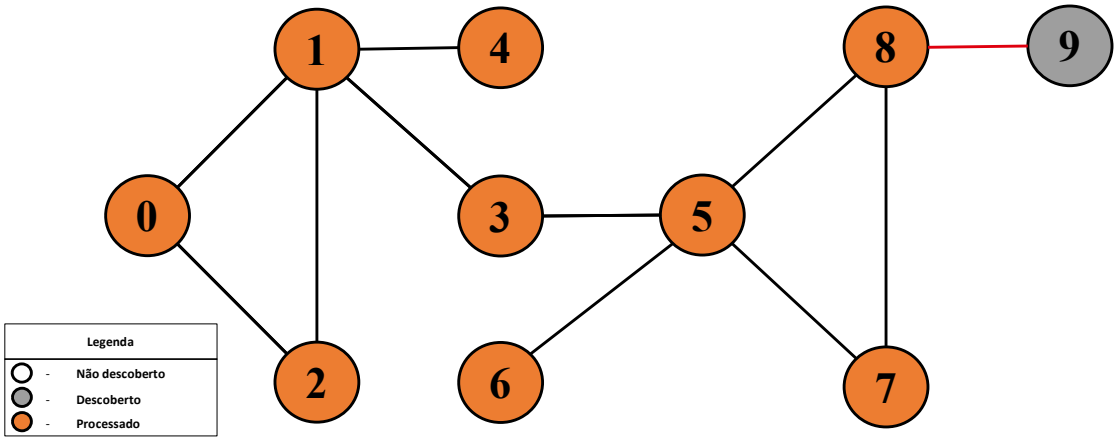


Marca todos os nós não visitados adjacentes a 7: nenhum

13

Intuição da BFS

Percorrer os vértices mais próximos primeiro

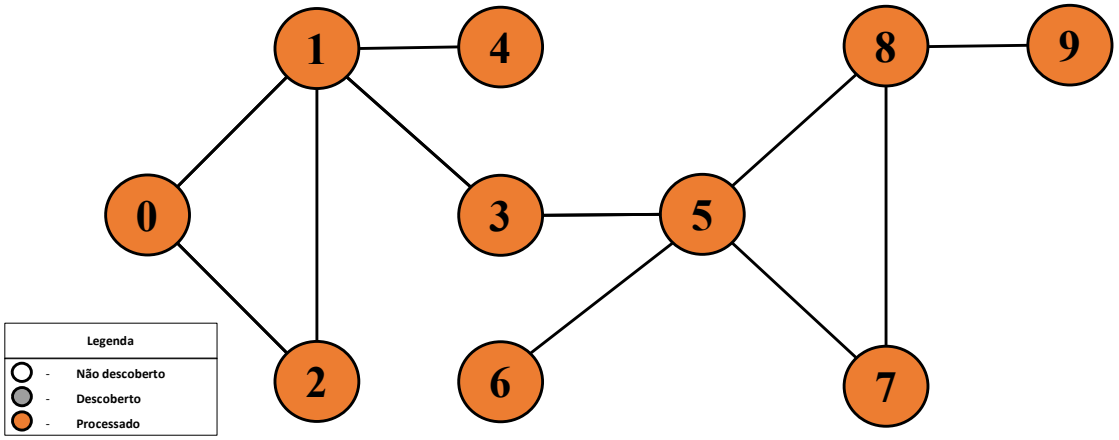


Marca todos os nós não visitados adjacentes a 8: 9

14

Intuição da BFS

Percorrer os vértices mais próximos primeiro

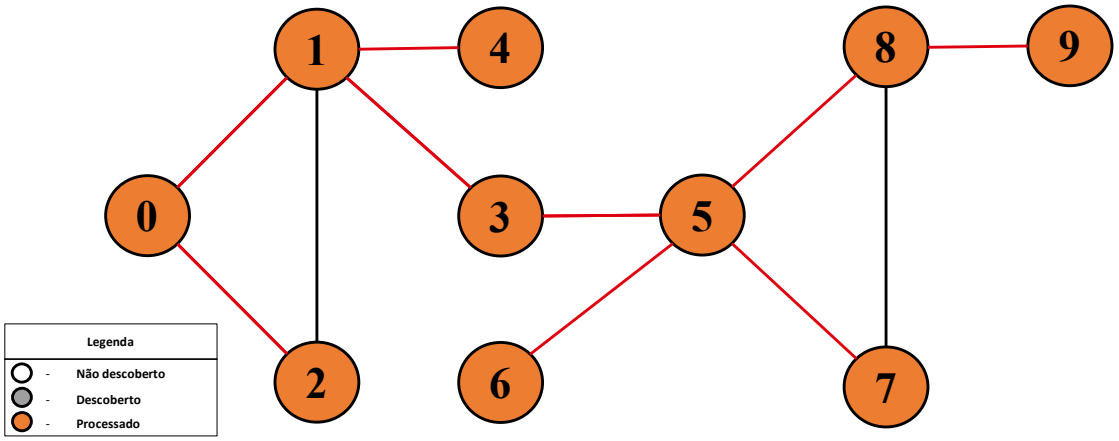


Marca todos os nós não visitados adjacentes a 9: nenhum

15

Intuição da BFS

Percorrer os vértices mais próximos primeiro

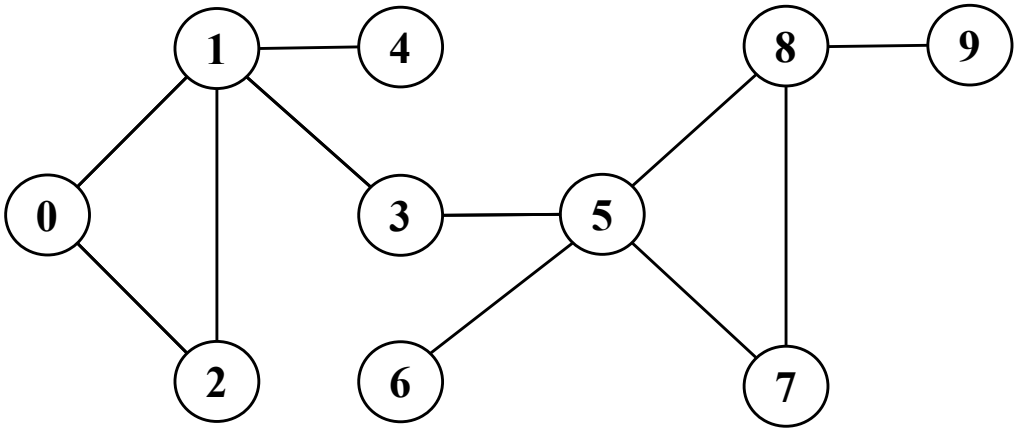


Ordem de visita: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

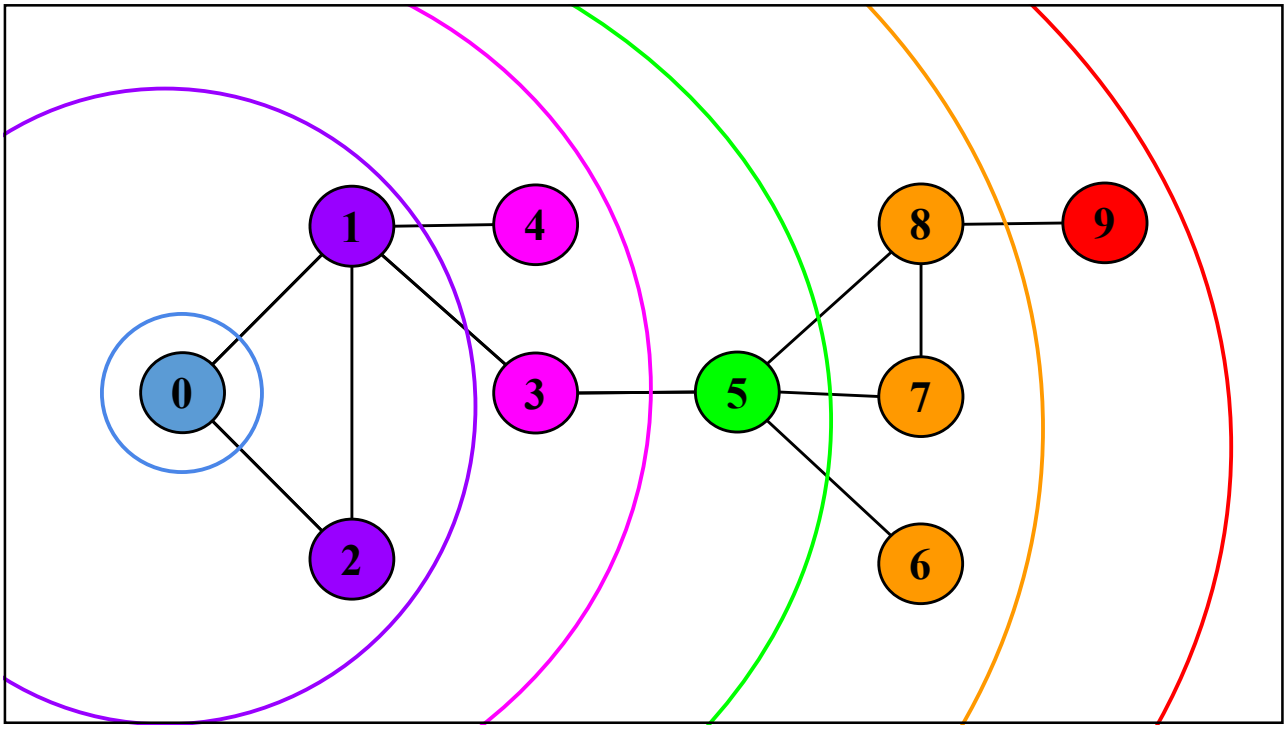
16

Intuição da BFS

Percorremos o grafo como uma onda se propaga na água



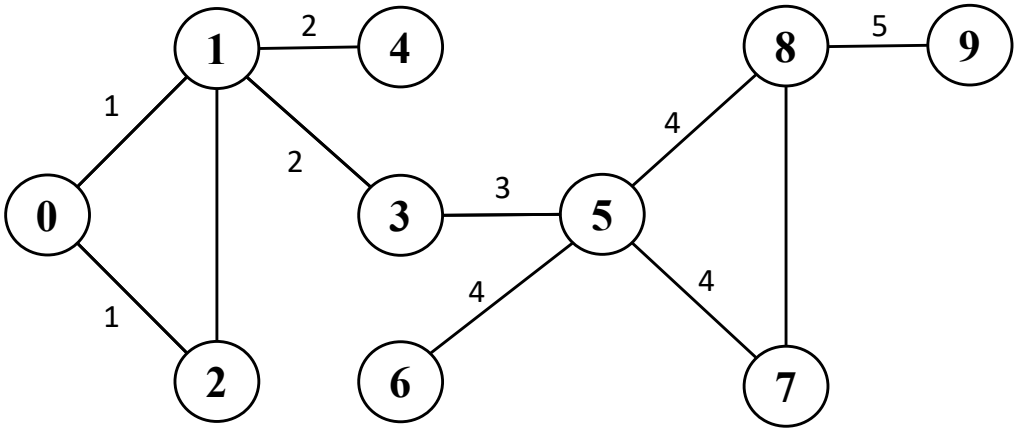
17



18

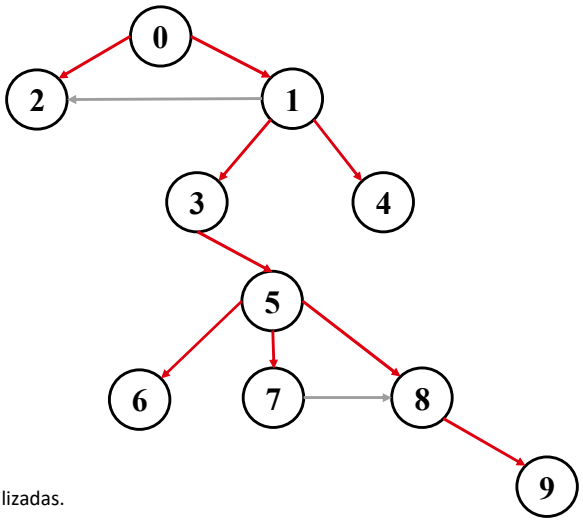
Busca em Largura

Consequência: se percorrem primeiro os vértices a uma distância k = 1 do vértice inicial, depois k+1, k+2, k+3 e assim por diante… Solução natural para **menor distância a partir da origem**!



Busca em Largura

Consequência: forma-se uma **árvore de busca em largura**



Arestas em cinza não são utilizadas.

Busca em Largura - Algoritmo

- Usaremos como estrutura de apoio uma **fila**

  1. Fila começa com o vértice inicial
  2. O primeiro vértice da fila é recuperado e processado, seus vizinhos não visitados são inseridos na fila
  3. Se a fila está vazia, encerra processo; se não, volta ao passo 2

Busca em Largura - Implementação

Pseudocódigo

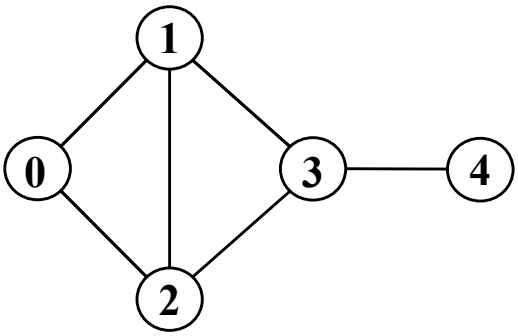
```
visited = [False] * n_vertices
bfs(v)
   queue.append(v)
   visited[v] = True
   while queue not empty
      v = queue.pop()
      for w in neighbors(v)
         if not visited[w]
            queue.append(w)
            visited[w] = True
```

Busca em Largura - Exemplo detalhado

Pseudocódigo

```
visited = [False] * n_vertices
bfs(v)
    queue.append(v)
    visited[v] = True
    while queue not empty
        v = queue.pop()
        for w in neighbors(v)
            if not visited[w]
                queue.append(w)
                visited[w] = True
```

bfs(v): ?



v = ?
queue = ?

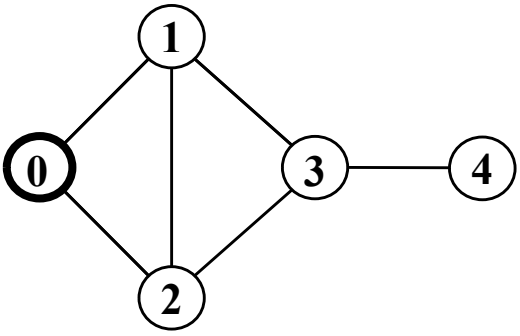
Busca em Largura - Exemplo detalhado

Pseudocódigo

```
visited = [False] * n_vertices
bfs(v)
    queue.append(v)
    visited[v] = True
    while queue not empty
        v = queue.pop()
        for w in neighbors(v)
            if not visited[w]
                queue.append(w)
                visited[w] = True
```

Coloca vértice inicial na fila

bfs(0): 0



v = 0
queue = {0}

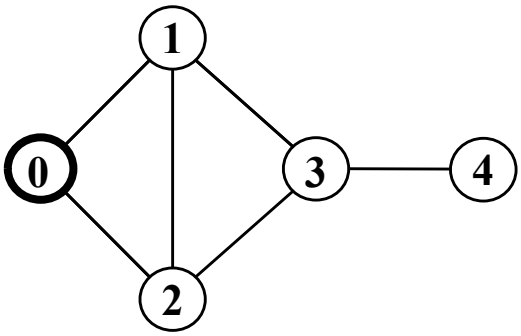
Busca em Largura - Exemplo detalhado

Pseudocódigo

```
visited = [False] * n_vertices
bfs(v)
   queue.append(v)
   visited[v] = True
   while queue not empty
      v = queue.pop()
      for w in neighbors(v)
         if not visited[w]
            queue.append(w)
            visited[w] = True
```

Tira vértice no topo da fila e atribui a v

bfs(0): 0



v = 0
queue = {}

25

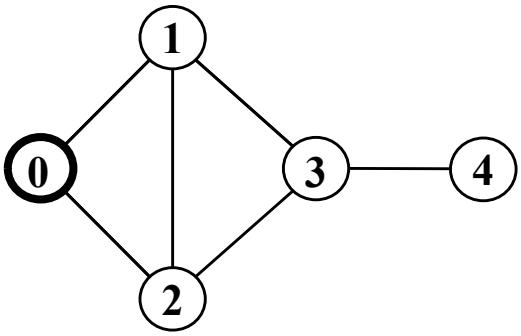
Busca em Largura - Exemplo detalhado

Pseudocódigo

```
visited = [False] * n_vertices
bfs(v)
   queue.append(v)
   visited[v] = True
   while queue not empty
      v = queue.pop()
      for w in neighbors(v)
         if not visited[w]
            queue.append(w)
            visited[w] = True
```

Olha vizinhos de 0, ambos não foram visitados

bfs(0): 0



v = 0
queue = {}

26

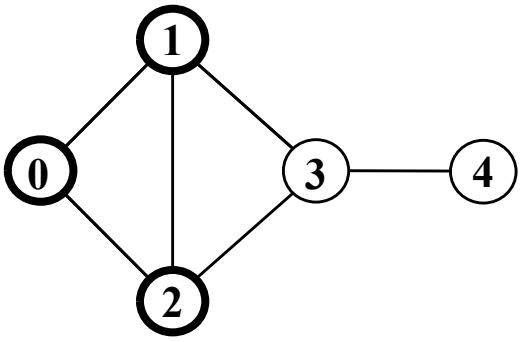
Busca em Largura - Exemplo detalhado

Pseudocódigo

```
visited = [False] * n_vertices
bfs(v)
   queue.append(v)
   visited[v] = True
   while queue not empty
      v = queue.pop()
      for w in neighbors(v)
         if not visited[w]
            queue.append(w)
            visited[w] = True
```

Adiciona ambos na fila e marca como visitado

bfs(0): 0, 1, 2



v = 0
queue = {2, 1}

27

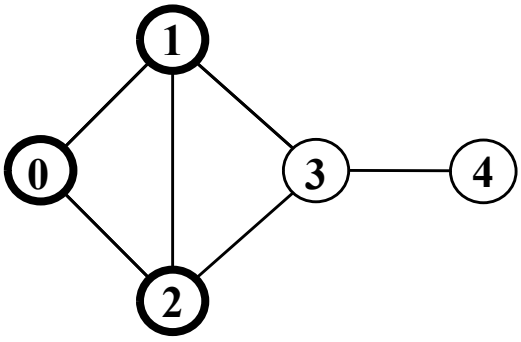
Busca em Largura - Exemplo detalhado

Pseudocódigo

```
visited = [False] * n_vertices
bfs(v)
   queue.append(v)
   visited[v] = True
   while queue not empty
      v = queue.pop()
      for w in neighbors(v)
         if not visited[w]
            queue.append(w)
            visited[w] = True
```

Tira 1 da fila e atribui a v

bfs(0): 0, 1, 2



v = 1
queue = {2}

28

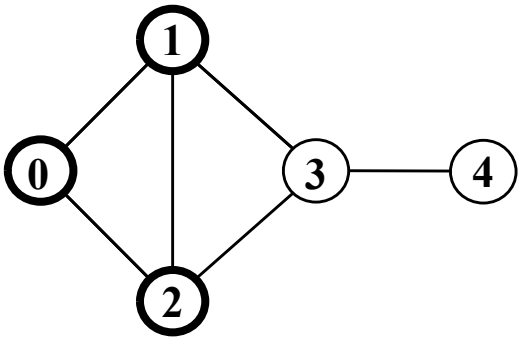
Busca em Largura - Exemplo detalhado

Pseudocódigo

```
visited = [False] * n_vertices
bfs(v)
   queue.append(v)
   visited[v] = True
   while queue not empty
      v = queue.pop()
      for w in neighbors(v)
         if not visited[w]
            queue.append(w)
            visited[w] = True
```

Olha vizinhos de 1, o 3 ainda não foi visitado

bfs(0): 0, 1, 2



v = 1
queue = {2}

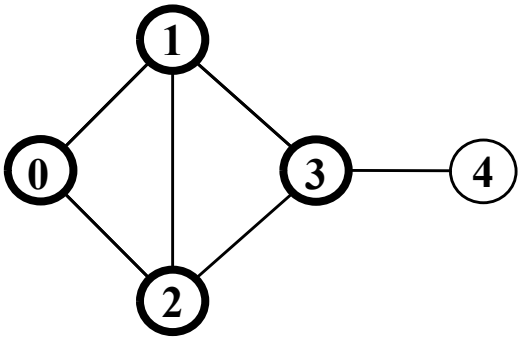
Busca em Largura - Exemplo detalhado

Pseudocódigo

```
visited = [False] * n_vertices
bfs(v)
   queue.append(v)
   visited[v] = True
   while queue not empty
      v = queue.pop()
      for w in neighbors(v)
         if not visited[w]
            queue.append(w)
            visited[w] = True
```

Adiciona 3 na fila e marca como visitado

bfs(0): 0, 1, 2, 3



v = 1
queue = {3, 2}

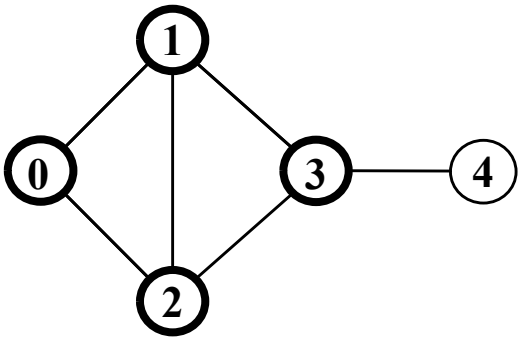
Busca em Largura - Exemplo detalhado

Pseudocódigo

```
visited = [False] * n_vertices
bfs(v)
   queue.append(v)
   visited[v] = True
   while queue not empty
      v = queue.pop()
      for w in neighbors(v)
         if not visited[w]
            queue.append(w)
            visited[w] = True
```

Tira 2 da fila e atribui a v

bfs(0): 0, 1, 2, 3



v = 2
queue = {3}

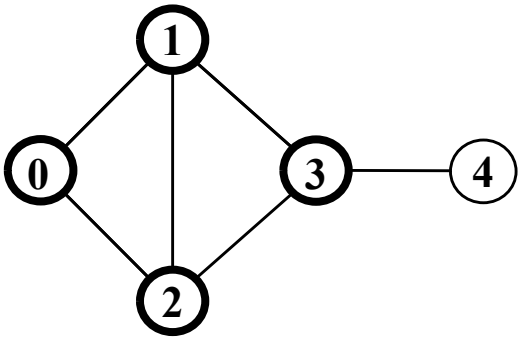
Busca em Largura - Exemplo detalhado

Pseudocódigo

```
visited = [False] * n_vertices
bfs(v)
   queue.append(v)
   visited[v] = True
   while queue not empty
      v = queue.pop()
      for w in neighbors(v)
         if not visited[w]
            queue.append(w)
            visited[w] = True
```

Todos os vizinhos de 2 já foram visitados

bfs(0): 0, 1, 2, 3



v = 2
queue = {3}

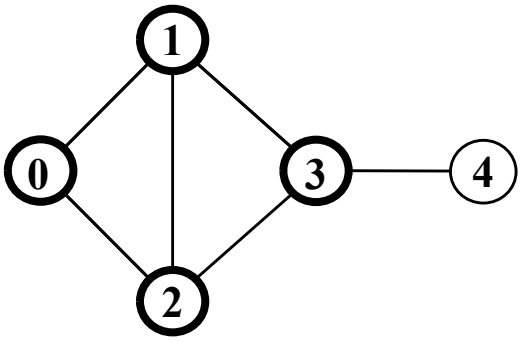
Busca em Largura - Exemplo detalhado

Pseudocódigo

```
visited = [False] * n_vertices
bfs(v)
   queue.append(v)
   visited[v] = True
   while queue not empty
      v = queue.pop()
      for w in neighbors(v)
         if not visited[w]
            queue.append(w)
            visited[w] = True
```

Tira 3 da fila e atribui a v

bfs(0): 0, 1, 2, 3



v = 3
queue = {}

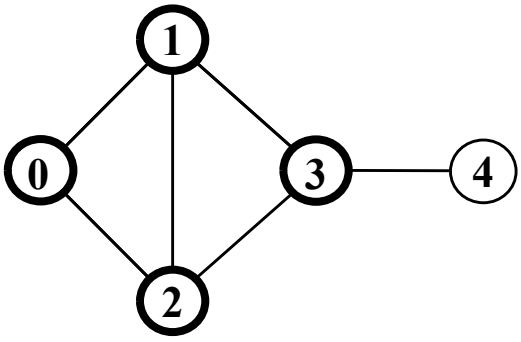
Busca em Largura - Exemplo detalhado

Pseudocódigo

```
visited = [False] * n_vertices
bfs(v)
   queue.append(v)
   visited[v] = True
   while queue not empty
      v = queue.pop()
      for w in neighbors(v)
         if not visited[w]
            queue.append(w)
            visited[w] = True
```

Olha vizinhos de 3, o 4 ainda não foi visitado

bfs(0): 0, 1, 2, 3



v = 3
queue = {}

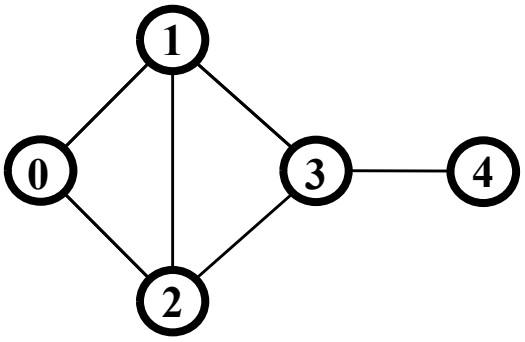
Busca em Largura - Exemplo detalhado

Pseudocódigo

```
visited = [False] * n_vertices
bfs(v)
   queue.append(v)
   visited[v] = True
   while queue not empty
      v = queue.pop()
      for w in neighbors(v)
         if not visited[w]
            queue.append(w)
            visited[w] = True
```

Adiciona 4 na fila e marca como visitado

bfs(0): 0, 1, 2, 3, 4



v = 3
queue = {4}

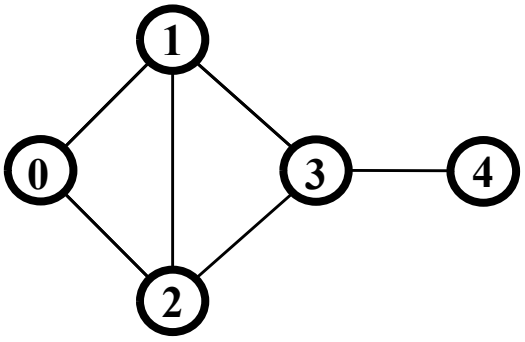
Busca em Largura - Exemplo detalhado

Pseudocódigo

```
visited = [False] * n_vertices
bfs(v)
   queue.append(v)
   visited[v] = True
   while queue not empty
      v = queue.pop()
      for w in neighbors(v)
         if not visited[w]
            queue.append(w)
            visited[w] = True
```

Tira 4 da fila e atribui a v

bfs(0): 0, 1, 2, 3, 4



v = 4
queue = {}

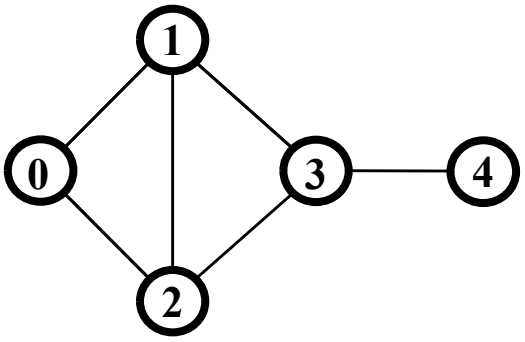
Busca em Largura - Exemplo detalhado

Pseudocódigo

```
visited = [False] * n_vertices
bfs(v)
   queue.append(v)
   visited[v] = True
   while queue not empty
      v = queue.pop()
      for w in neighbors(v)
         if not visited[w]
            queue.append(w)
            visited[w] = True
```

Todos os vizinhos de 4 já foram visitados

bfs(0): 0, 1, 2, 3, 4



v = 4
queue = {}

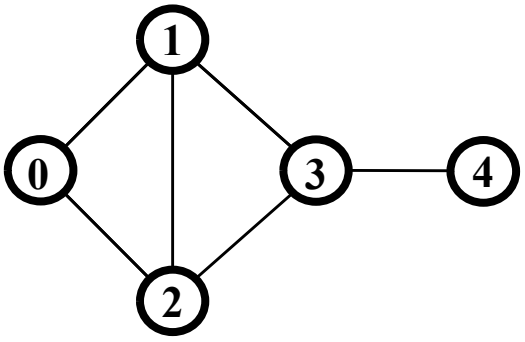
Busca em Largura - Exemplo detalhado

Pseudocódigo

```
visited = [False] * n_vertices
bfs(v)
   queue.append(v)
   visited[v] = True
   while queue not empty
      v = queue.pop()
      for w in neighbors(v)
         if not visited[w]
            queue.append(w)
            visited[w] = True
```

Fila vazia, finaliza execução

bfs(0): 0, 1, 2, 3, 4



v = 4
queue = {}

Busca em Largura - Complexidade

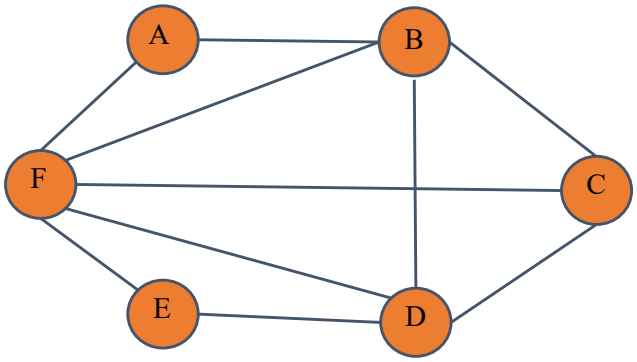
O(|V|+|A|), isto é, linear em relação ao tamanho da representação do grafo por listas de adjacências

- Todos os vértices são enfileirados/desenfileirados no máximo uma vez; o custo de cada uma dessas operações é O(1), e elas são executadas O(|V|) vezes
- A lista de adjacências de cada vértices é percorrida no máximo uma vez (quando o vértice sai da fila); o tempo total é O(|A|) (soma dos comprimentos de todas as listas, igual ao número de arestas)

Somando estas duas, a complexidade da BFS é **O(|V|+|A|)**

Exercícios de Fixação

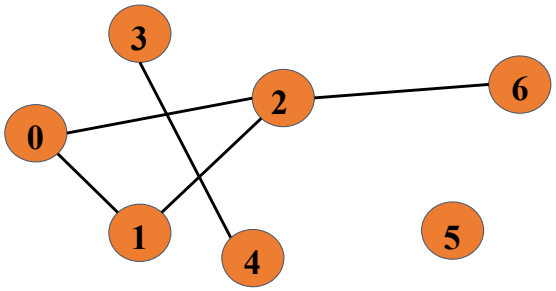
Execute a BFS no grafo abaixo começando pelo vértice A



Exercícios de Fixação

- Como a busca em largura pode ajudar a determinar o número de componentes conexas de um grafo?
  - Implemente uma função connected_components() que utiliza a BFS para descobrir quantas componentes conexas há em um grafo.

Exemplo:



connected_components() = 3

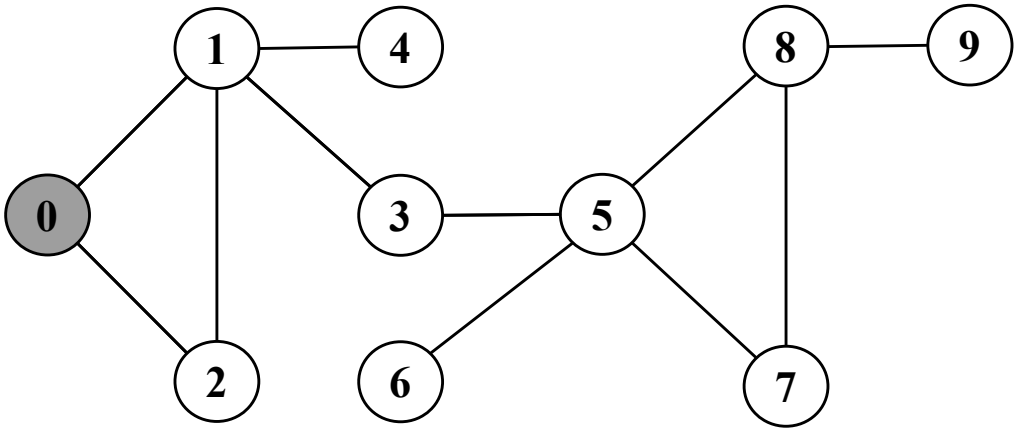
Busca em Profundidade (DFS)

https://devsuperior.com.br

Prof. Dr. Nelio Alves

Intuição da DFS

Avançar na busca sem olhar para vértices vizinhos no mesmo nível

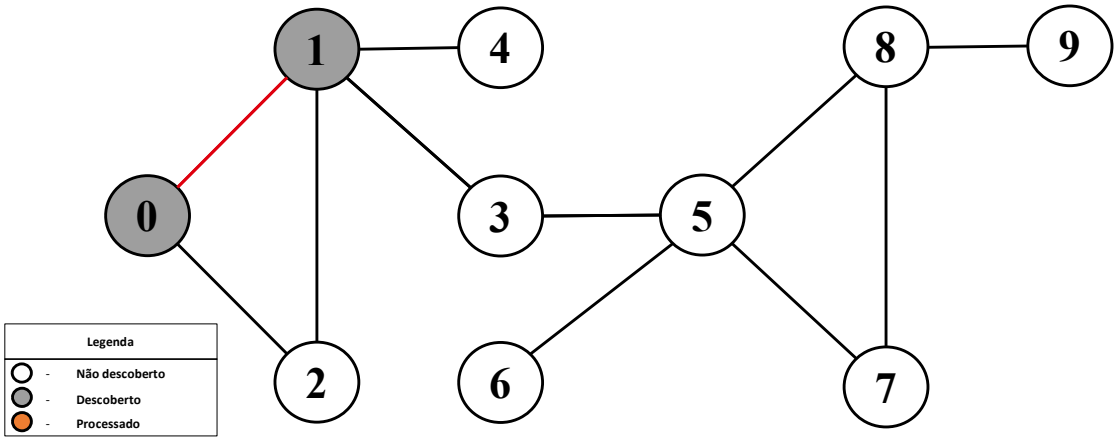


Nó inicial: 0

43

Intuição da DFS

Avançar na busca sem olhar para vértices vizinhos no mesmo nível

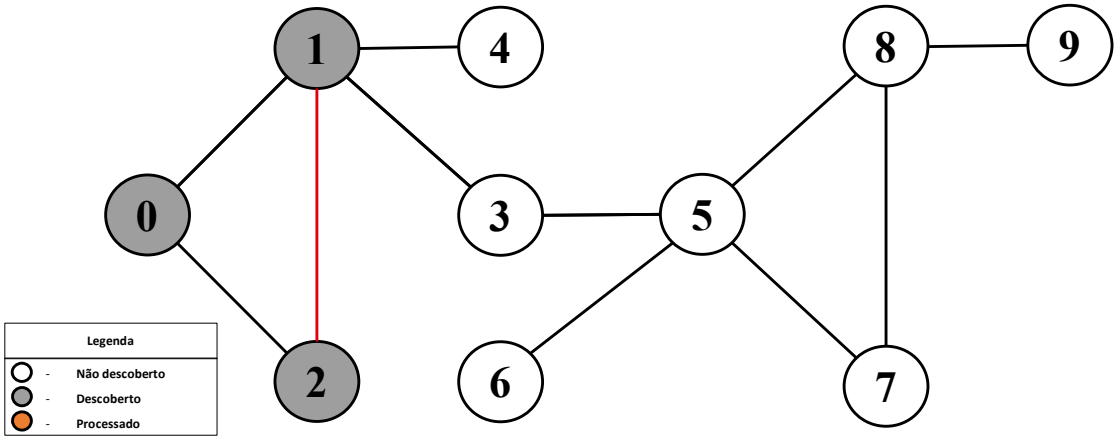


Busca pelo primeiro vértice adjacente a 0: 1

44

Intuição da DFS

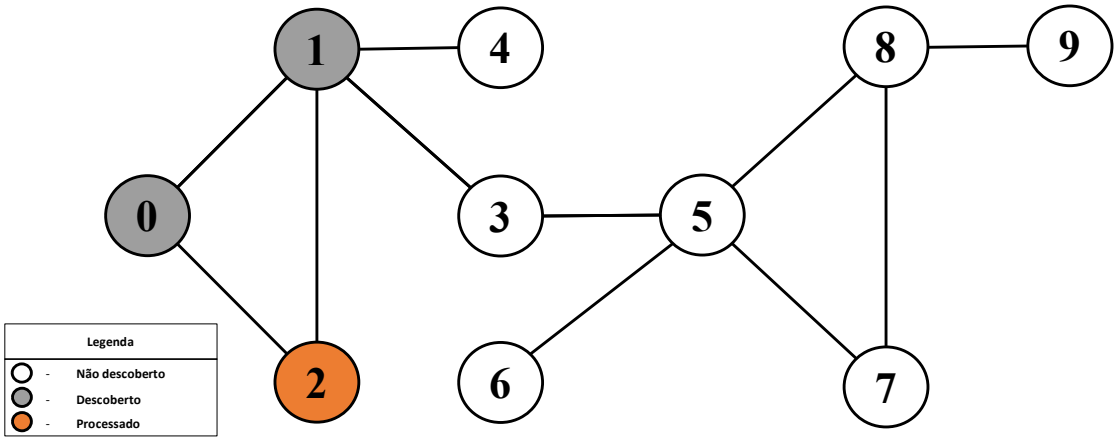
Avançar na busca sem olhar para vértices vizinhos no mesmo nível



Busca pelo primeiro vértice adjacente a 1: 2

Intuição da DFS

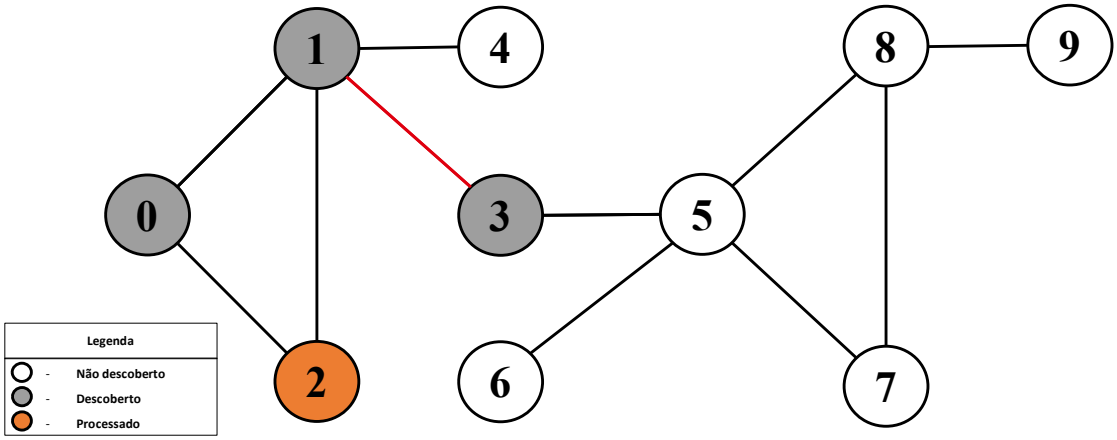
Avançar na busca sem olhar para vértices vizinhos no mesmo nível



Vértice 2 não tem mais vértices adjacentes! Marca como processado e volta ao anterior.

Intuição da DFS

Avançar na busca sem olhar para vértices vizinhos no mesmo nível

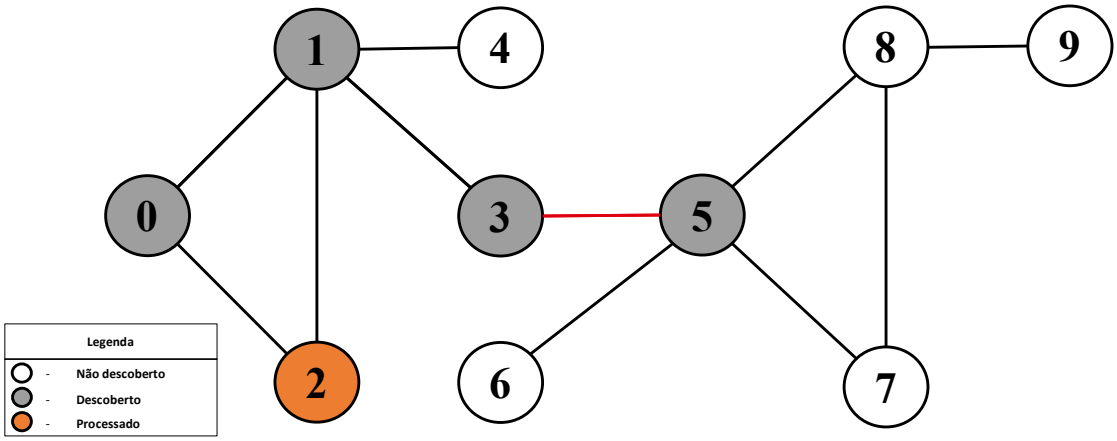


Busca por outro vértice adjacente a 1: 3

47

Intuição da DFS

Avançar na busca sem olhar para vértices vizinhos no mesmo nível

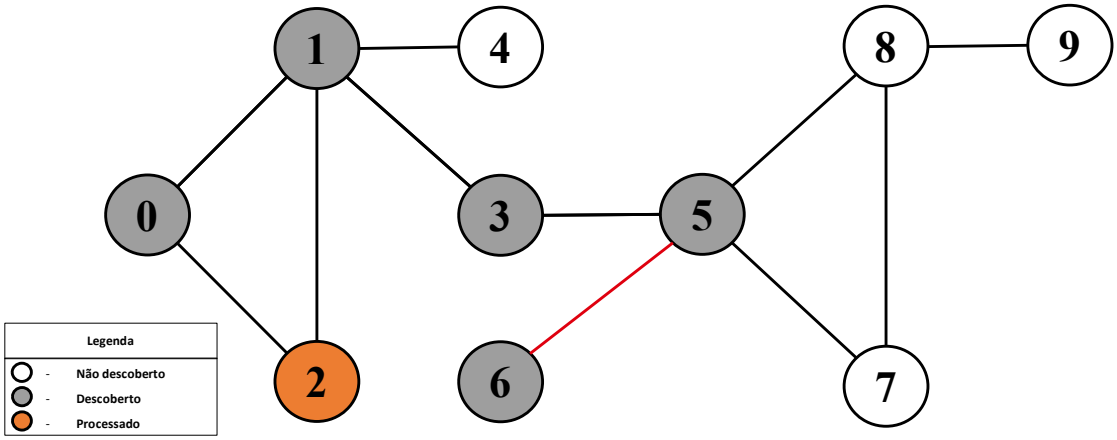


Busca pelo primeiro vértice adjacente a 3: 5

48

Intuição da DFS

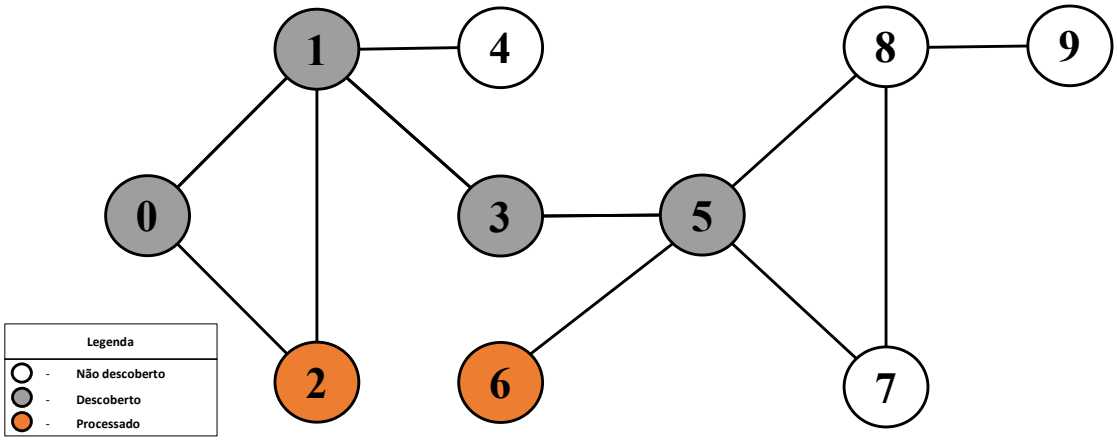
Avançar na busca sem olhar para vértices vizinhos no mesmo nível



Busca pelo primeiro vértice adjacente a 5: 6

Intuição da DFS

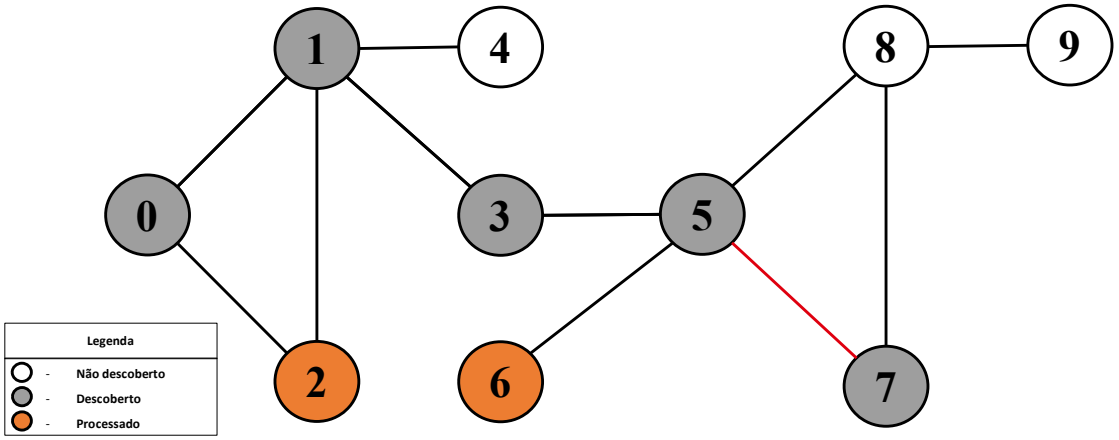
Avançar na busca sem olhar para vértices vizinhos no mesmo nível



Vértice 6 não tem mais vértices adjacentes! Marca como processado e volta ao anterior.

Intuição da DFS

Avançar na busca sem olhar para vértices vizinhos no mesmo nível

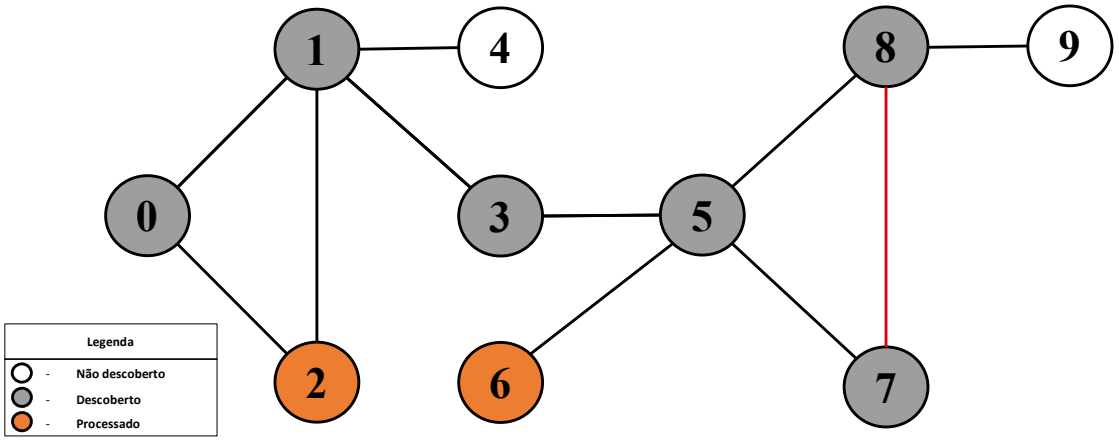


Busca por outro vértice adjacente a 5: 7

51

Intuição da DFS

Avançar na busca sem olhar para vértices vizinhos no mesmo nível

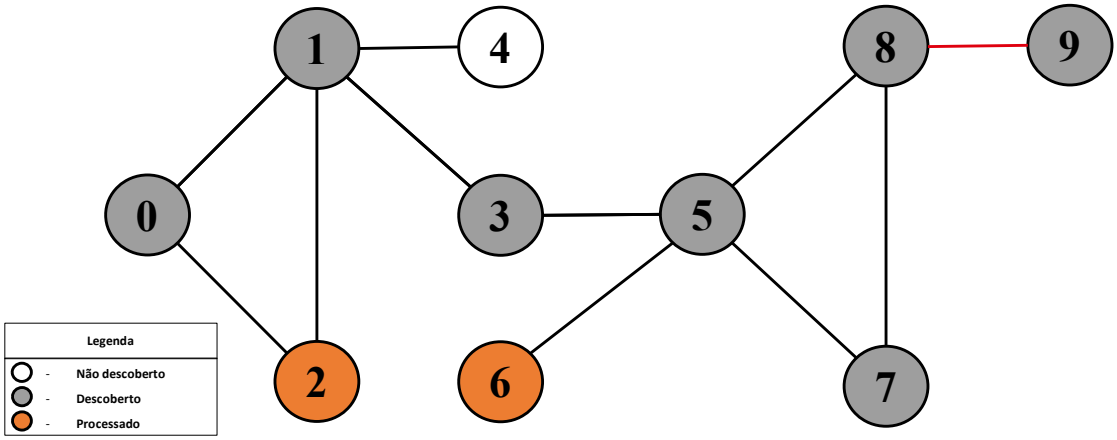


Busca por outro vértice adjacente a 7: 8

52

Intuição da DFS

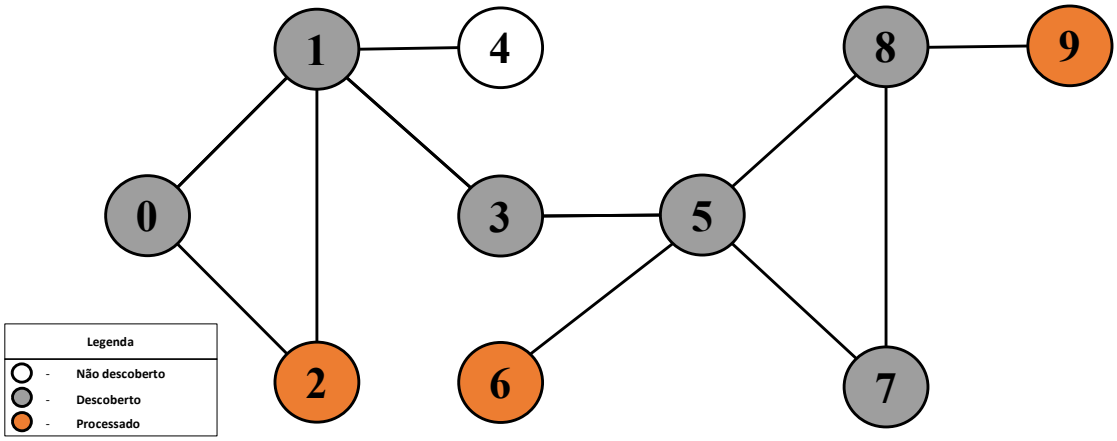
Avançar na busca sem olhar para vértices vizinhos no mesmo nível



Busca por outro vértice adjacente a 8: 9

Intuição da DFS

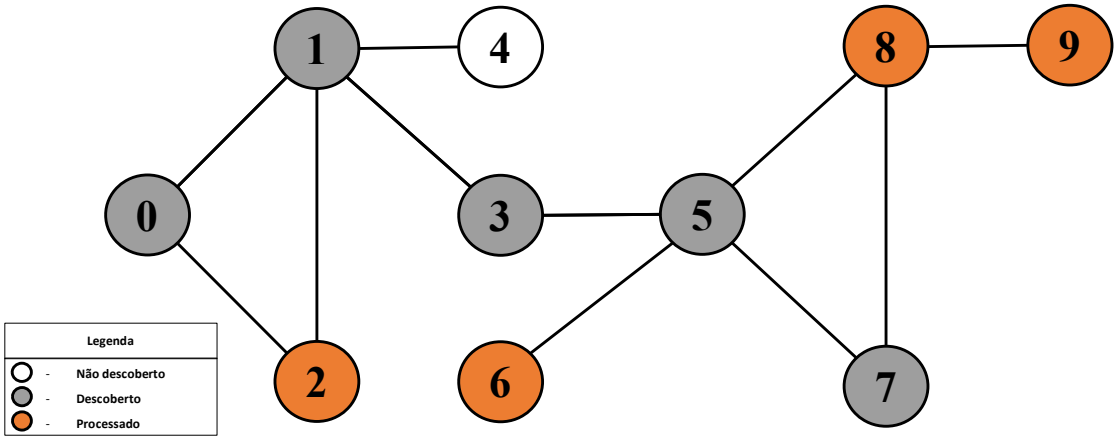
Avançar na busca sem olhar para vértices vizinhos no mesmo nível



Vértice 9 não tem mais vizinhos! Marca e volta ao anterior.

Intuição da DFS

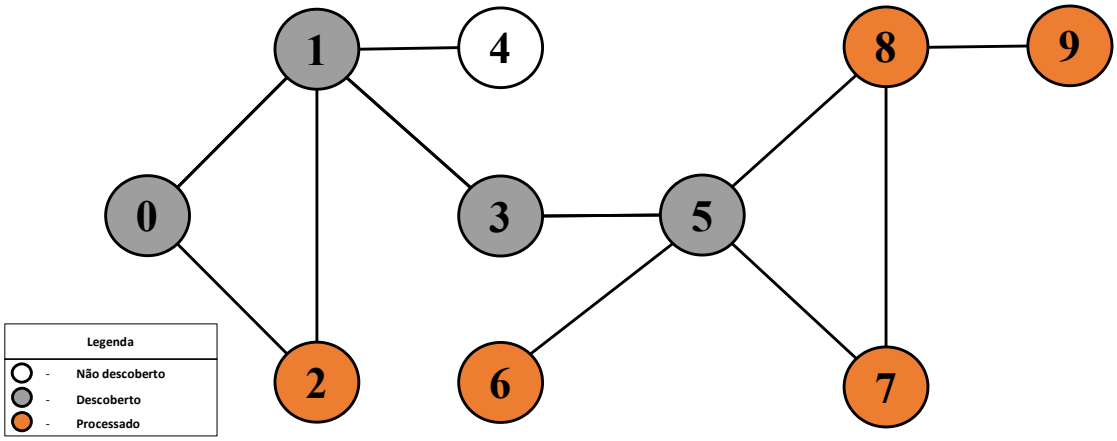
Avançar na busca sem olhar para vértices vizinhos no mesmo nível



Vértice 8 não tem mais vizinhos! Marca e volta ao anterior.

Intuição da DFS

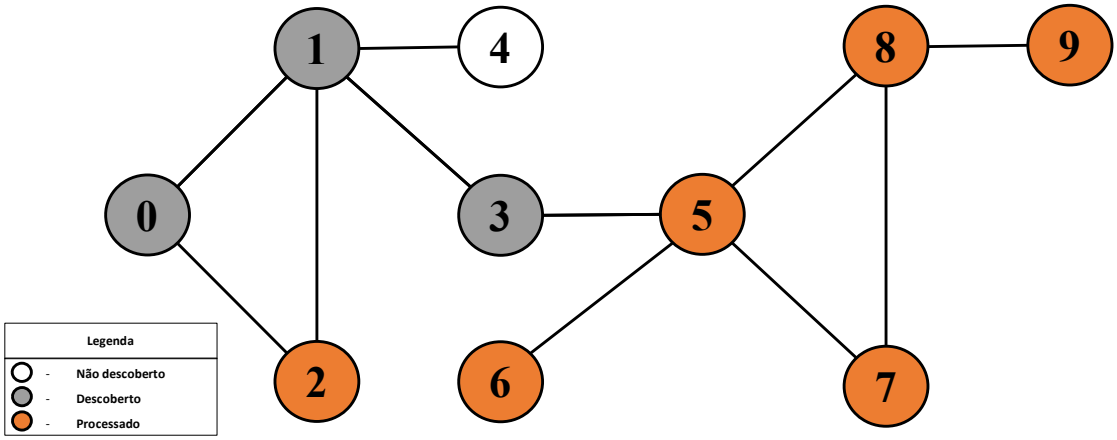
Avançar na busca sem olhar para vértices vizinhos no mesmo nível



Vértice 7 não tem mais vizinhos! Marca e volta ao anterior.

Intuição da DFS

Avançar na busca sem olhar para vértices vizinhos no mesmo nível

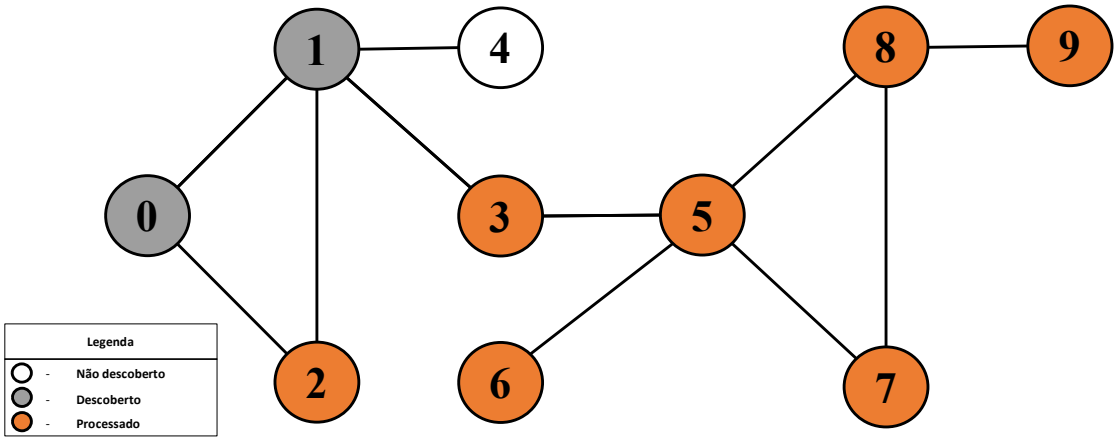


Vértice 5 não tem mais vizinhos! Marca e volta ao anterior.

57

Intuição da DFS

Avançar na busca sem olhar para vértices vizinhos no mesmo nível

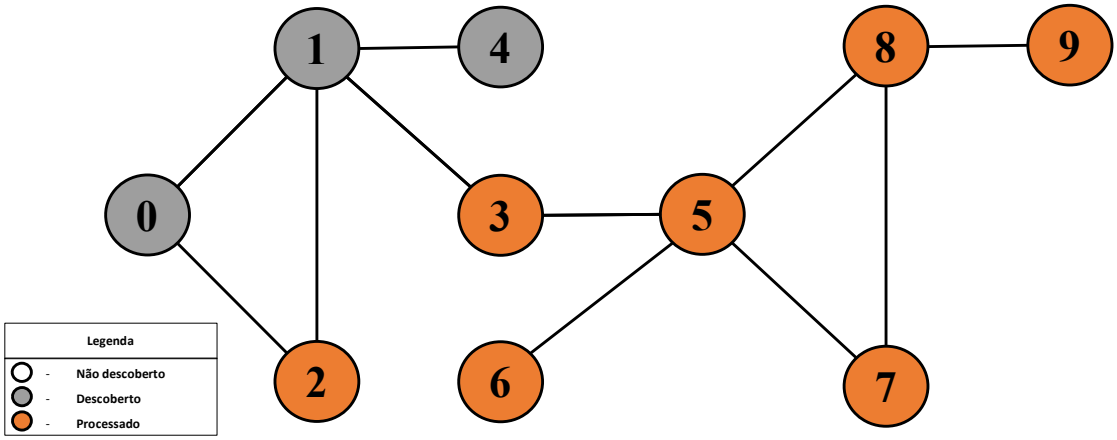


Vértice 3 não tem mais vizinhos! Marca e volta ao anterior.

58

Intuição da DFS

Avançar na busca sem olhar para vértices vizinhos no mesmo nível

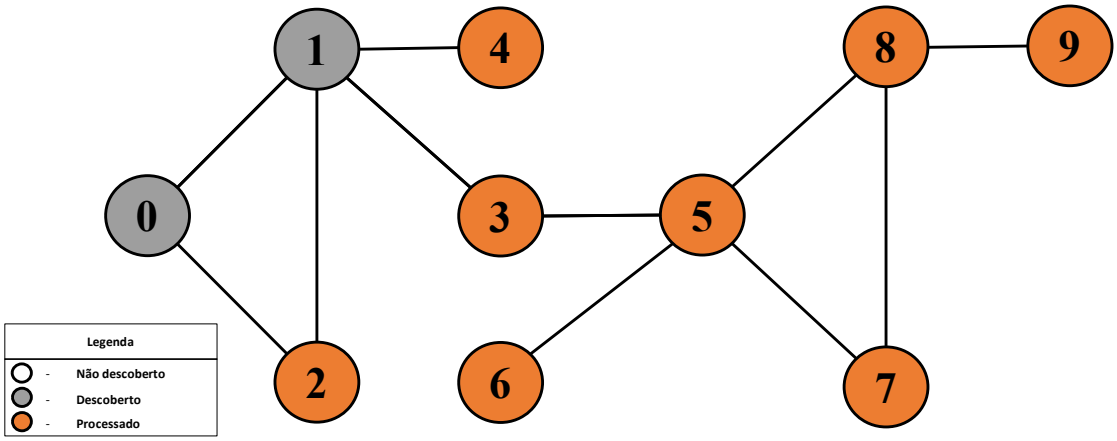


Busca por outro vértice adjacente a 1: 4

59

Intuição da DFS

Avançar na busca sem olhar para vértices vizinhos no mesmo nível

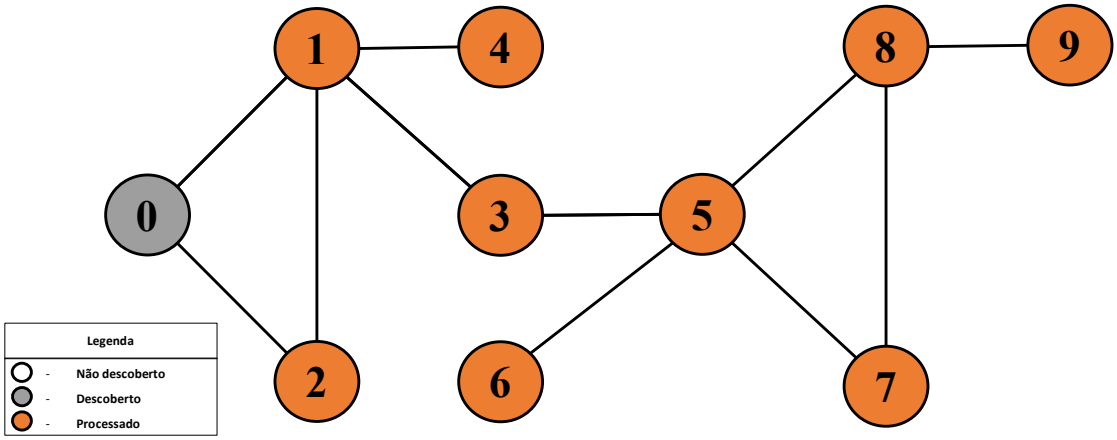


Vértice 4 não tem mais vizinhos! Marca e volta ao anterior.

60

Intuição da DFS

Avançar na busca sem olhar para vértices vizinhos no mesmo nível

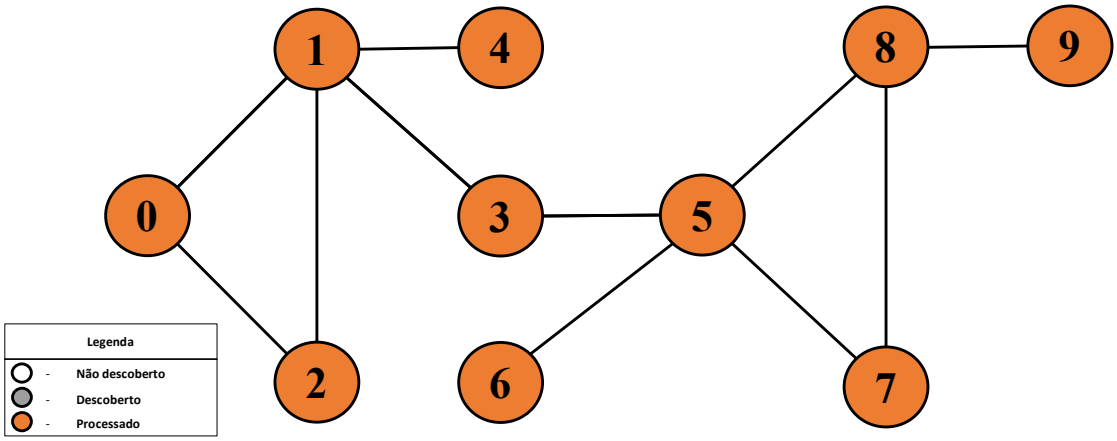


Vértice 1 não tem mais vizinhos! Marca e volta ao anterior.

61

Intuição da DFS

Avançar na busca sem olhar para vértices vizinhos no mesmo nível

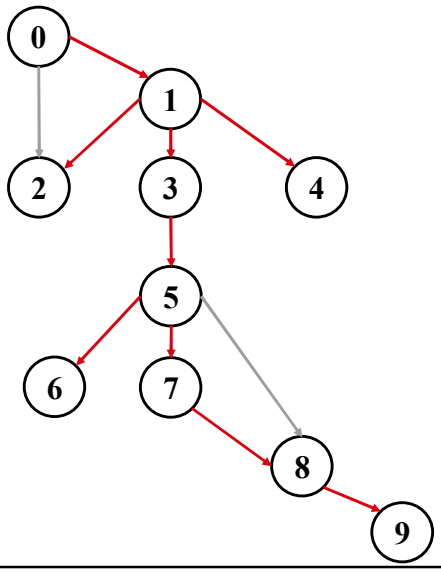


Vértice 0 não tem mais vizinhos! Marca e volta ao anterior. DFS finalizada!

62

Busca em Profundidade

Percorrendo o grafo: forma-se uma **árvore de busca em profundidade**



Arestas em cinza não são utilizadas.

Busca em Profundidade - Algoritmo

- Usaremos como estrutura de apoio uma **pilha**
  - A pilha pode ser implícita (abordagem recursiva) ou explícita (abordagem iterativa)

1. A cada escolha de caminho, empilhamos o vértice original e seguimos o caminho
2. Cada vez que o caminho acaba, voltamos ao vértice anterior empilhado

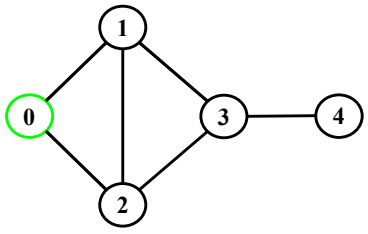
Busca em Profundidade - Implementação Recursiva

Pseudocódigo

```
visited = [False] * n_vertices
dfs(v)
   visited[v] = True
   for w in neighbors(v)
      if not visited[w]
         dfs(w)
```


Busca em Profundidade - Exemplo detalhado

dfs(0)

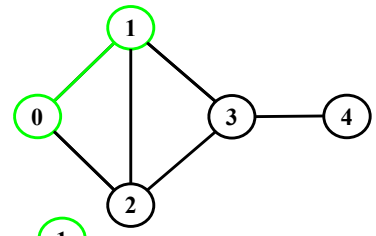



```
visited = [False] * n_vertices
dfs(v)
   visited[v] = True
   for w in neighbors(v)
      if not visited[w]
         dfs(w)
```

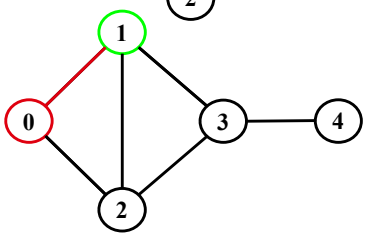
dfs(0): 0

Busca em Profundidade - Exemplo detalhado

dfs(0)



dfs(1)

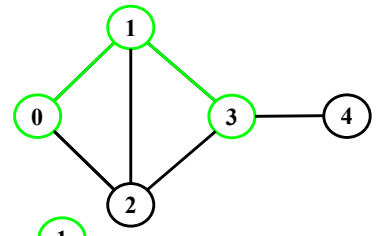



```
visited = [False] * n_vertices
dfs(v)
   visited[v] = True
   for w in neighbors(v)
      if not visited[w]
         dfs(w)
```

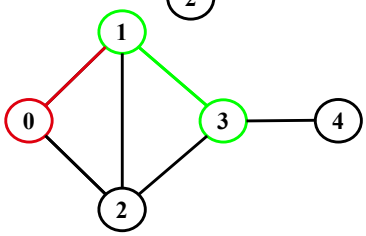
dfs(0): 0, 1

Busca em Profundidade - Exemplo detalhado

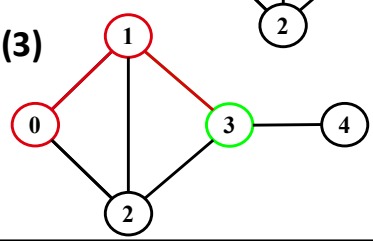
dfs(0)



dfs(1)



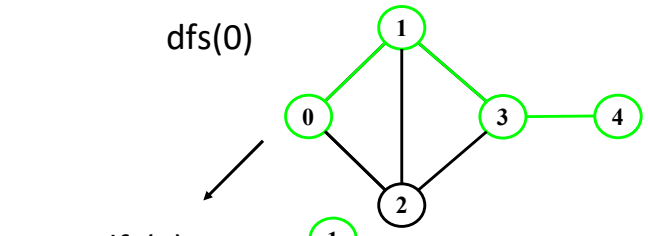
dfs(3)

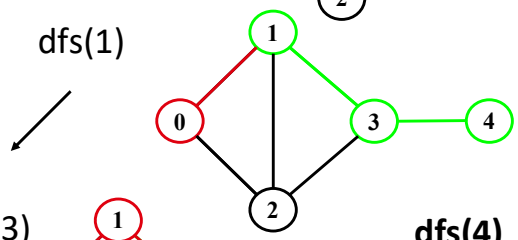


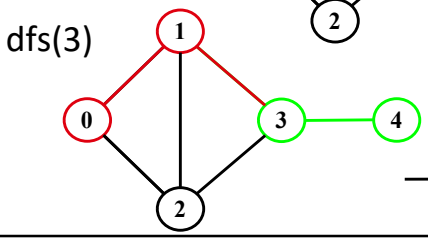

```
visited = [False] * n_vertices
dfs(v)
   visited[v] = True
   for w in neighbors(v)
      if not visited[w]
         dfs(w)
```

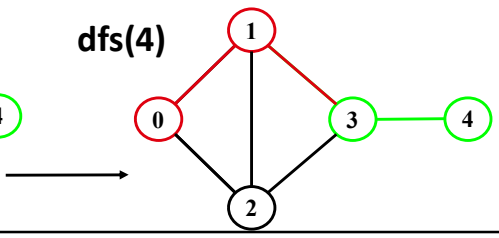
dfs(0): 0, 1, 3

Busca em Profundidade - Exemplo detalhado





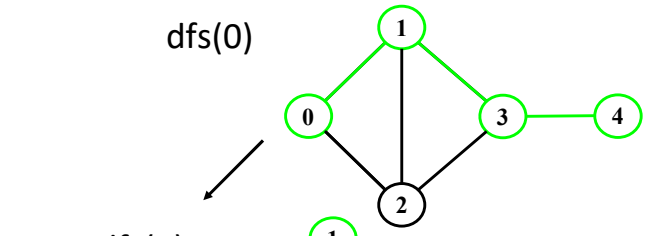


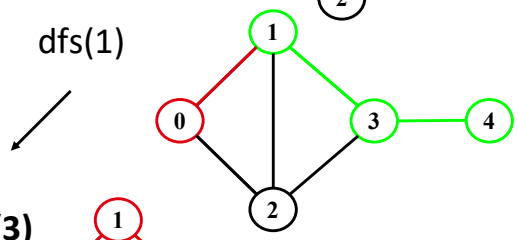


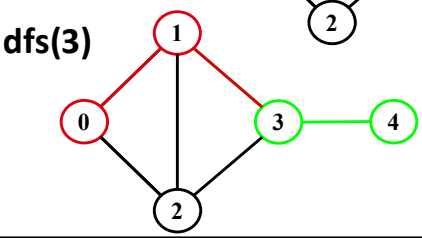

```
visited = [False] * n_vertices
dfs(v)
   visited[v] = True
   for w in neighbors(v)
      if not visited[w]
         dfs(w)
```

dfs(0): 0, 1, 3, 4

Busca em Profundidade - Exemplo detalhado



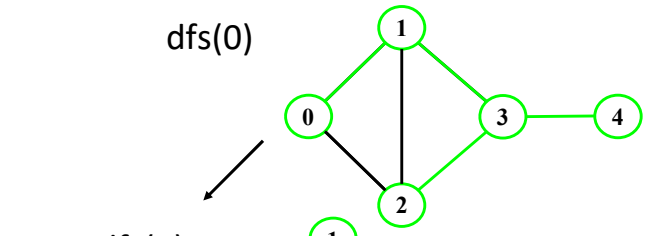


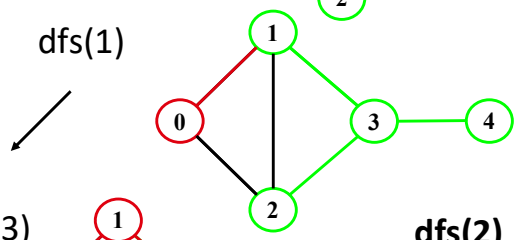


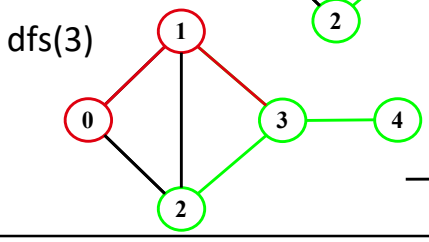

```
visited = [False] * n_vertices
dfs(v)
   visited[v] = True
   for w in neighbors(v)
      if not visited[w]
         dfs(w)
```

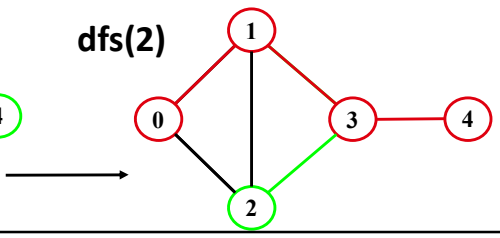
dfs(0): 0, 1, 3, 4

Busca em Profundidade - Exemplo detalhado





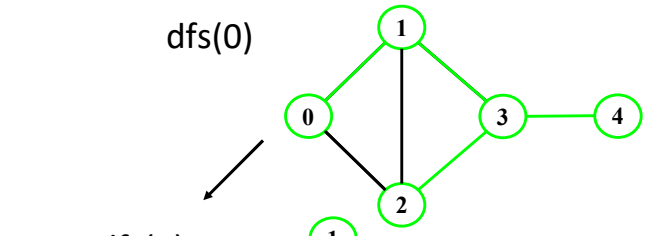


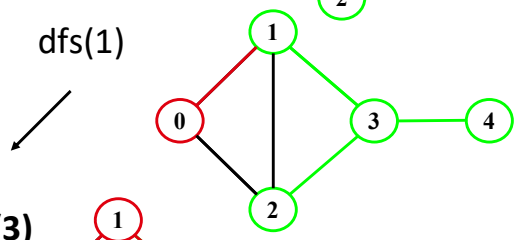


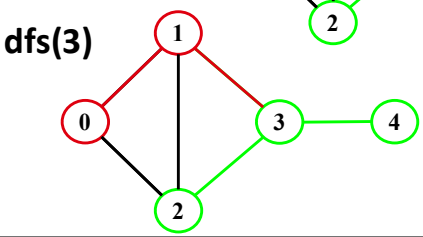

```
visited = [False] * n_vertices
dfs(v)
   visited[v] = True
   for w in neighbors(v)
      if not visited[w]
         dfs(w)
```

dfs(0): 0, 1, 3, 4, 2

Busca em Profundidade - Exemplo detalhado





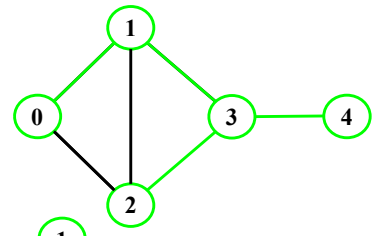



```
visited = [False] * n_vertices
dfs(v)
   visited[v] = True
   for w in neighbors(v)
      if not visited[w]
         dfs(w)
```

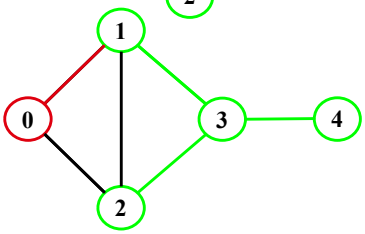
dfs(0): 0, 1, 3, 4, 2

Busca em Profundidade - Exemplo detalhado

dfs(0)



dfs(1)

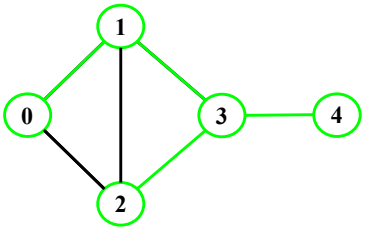



```
visited = [False] * n_vertices
dfs(v)
   visited[v] = True
   for w in neighbors(v)
      if not visited[w]
         dfs(w)
```

dfs(0): 0, 1, 3, 4, 2

Busca em Profundidade - Exemplo detalhado

dfs(0)




```
visited = [False] * n_vertices
dfs(v)
   visited[v] = True
   for w in neighbors(v)
      if not visited[w]
         dfs(w)
```

dfs(0): 0, 1, 3, 4, 2

Busca em Profundidade - Implementação Iterativa

Pseudocódigo

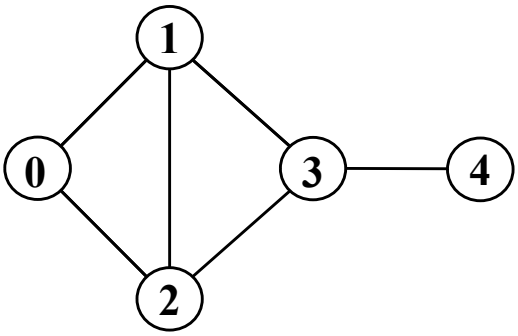
```
visited = [False] * n_vertices
dfs_iter(v)
   stack.append(v)
   visited[v] = True
   while stack not empty
      v = stack.pop()
      for w in neighbors(v)
         if not visited[w]
            stack.append(w)
            visited[w] = True
```


Busca em Profundidade - Exemplo detalhado

Pseudocódigo

```
visited = [False] * n_vertices
dfs_iter(v)
   stack.append(v)
   visited[v] = True
   while stack not empty
      v = stack.pop()
      for w in neighbors(v)
         if not visited[w]
            stack.append(w)
            visited[w] = True
```

dfs(v): ?



v = ?
stack = ?

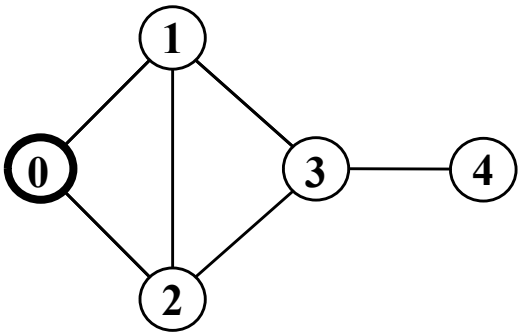
Busca em Profundidade - Exemplo detalhado

Pseudocódigo

```
visited = [False] * n_vertices
dfs_iter(v)
   stack.append(v)
   visited[v] = True
   while stack not empty
      v = stack.pop()
      for w in neighbors(v)
         if not visited[w]
            stack.append(w)
            visited[w] = True
```

Coloca vértice inicial na pilha e marca

dfs(0): 0



v = 0
stack = {0}

77

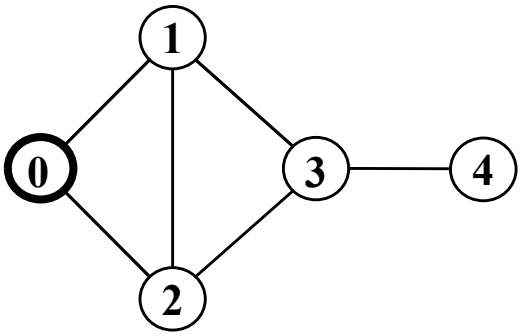
Busca em Profundidade - Exemplo detalhado

Pseudocódigo

```
visited = [False] * n_vertices
dfs_iter(v)
   stack.append(v)
   visited[v] = True
   while stack not empty
      v = stack.pop()
      for w in neighbors(v)
         if not visited[w]
            stack.append(w)
            visited[w] = True
```

Desempilha vértice no topo

dfs(0): 0



v = 0
stack = {}

78

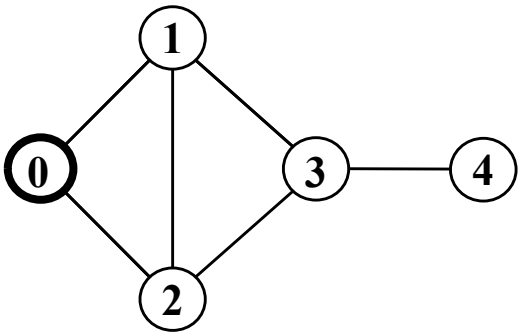
Busca em Profundidade - Exemplo detalhado

Pseudocódigo

```
visited = [False] * n_vertices
dfs_iter(v)
   stack.append(v)
   visited[v] = True
   while stack not empty
      v = stack.pop()
      for w in neighbors(v)
         if not visited[w]
            stack.append(w)
            visited[w] = True
```

Checa vizinhos de 0 por não visitados

dfs(0): 0



v = 0
stack = {}

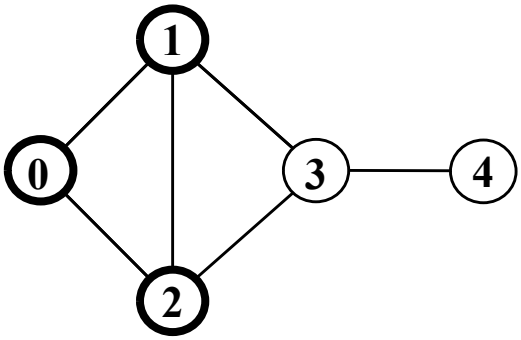
Busca em Profundidade - Exemplo detalhado

Pseudocódigo

```
visited = [False] * n_vertices
dfs_iter(v)
   stack.append(v)
   visited[v] = True
   while stack not empty
      v = stack.pop()
      for w in neighbors(v)
         if not visited[w]
            stack.append(w)
            visited[w] = True
```

Empilha vizinhos 1 e 2 e os marca

dfs(0): 0, 1, 2



v = 0
stack = {1, 2}

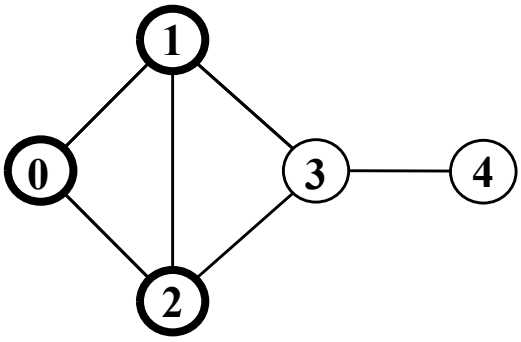
Busca em Profundidade - Exemplo detalhado

Pseudocódigo

```
visited = [False] * n_vertices
dfs_iter(v)
   stack.append(v)
   visited[v] = True
   while stack not empty
      v = stack.pop()
      for w in neighbors(v)
         if not visited[w]
            stack.append(w)
            visited[w] = True
```

Desempilha vértice no topo

dfs(0): 0, 1, 2



v = 2
stack = {1}

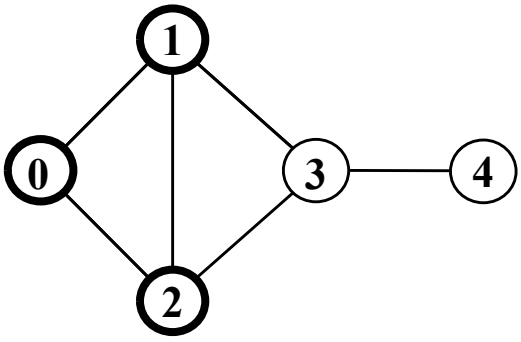
Busca em Profundidade - Exemplo detalhado

Pseudocódigo

```
visited = [False] * n_vertices
dfs_iter(v)
   stack.append(v)
   visited[v] = True
   while stack not empty
      v = stack.pop()
      for w in neighbors(v)
         if not visited[w]
            stack.append(w)
            visited[w] = True
```

Checa vizinhos de 2 por não visitados

dfs(0): 0, 1, 2



v = 2
stack = {1}

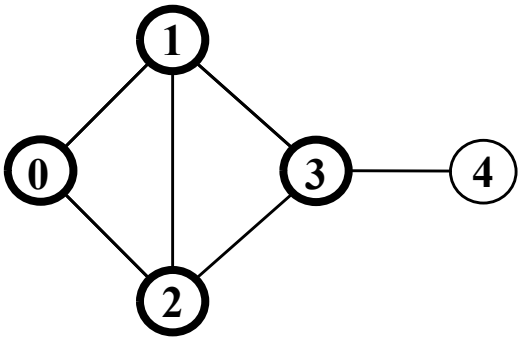
Busca em Profundidade - Exemplo detalhado

Pseudocódigo

```
visited = [False] * n_vertices
dfs_iter(v)
   stack.append(v)
   visited[v] = True
   while stack not empty
      v = stack.pop()
      for w in neighbors(v)
         if not visited[w]
            stack.append(w)
            visited[w] = True
```

Empilha 3 e o marca

dfs(0): 0, 1, 2, 3



v = 2
stack = {1, 3}

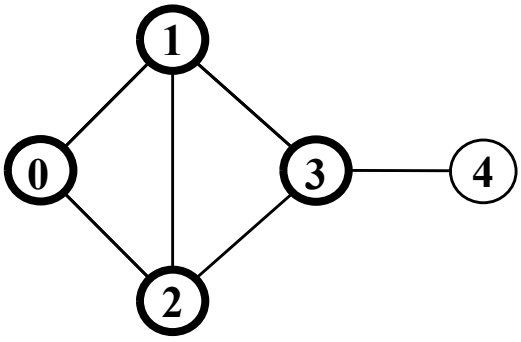
Busca em Profundidade - Exemplo detalhado

Pseudocódigo

```
visited = [False] * n_vertices
dfs_iter(v)
   stack.append(v)
   visited[v] = True
   while stack not empty
      v = stack.pop()
      for w in neighbors(v)
         if not visited[w]
            stack.append(w)
            visited[w] = True
```

Desempilha vértice no topo

dfs(0): 0, 1, 2, 3



v = 3
stack = {1}

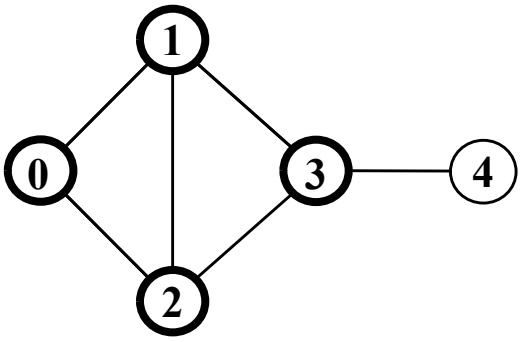
Busca em Profundidade - Exemplo detalhado

Pseudocódigo

```
visited = [False] * n_vertices
dfs_iter(v)
   stack.append(v)
   visited[v] = True
   while stack not empty
      v = stack.pop()
      for w in neighbors(v)
         if not visited[w]
            stack.append(w)
            visited[w] = True
```

Checa vizinhos de 3 por não visitados

dfs(0): 0, 1, 2, 3



v = 3
stack = {1}

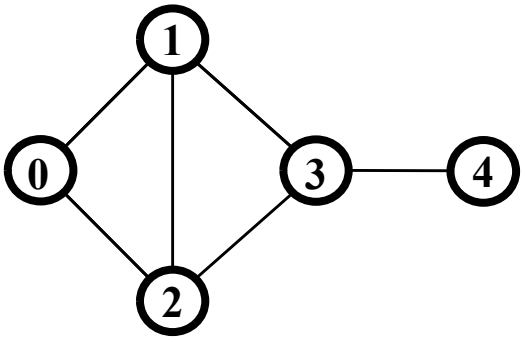
Busca em Profundidade - Exemplo detalhado

Pseudocódigo

```
visited = [False] * n_vertices
dfs_iter(v)
   stack.append(v)
   visited[v] = True
   while stack not empty
      v = stack.pop()
      for w in neighbors(v)
         if not visited[w]
            stack.append(w)
            visited[w] = True
```

Empilha 4 e o marca

dfs(0): 0, 1, 2, 3, 4



v = 3
stack = {1, 4}

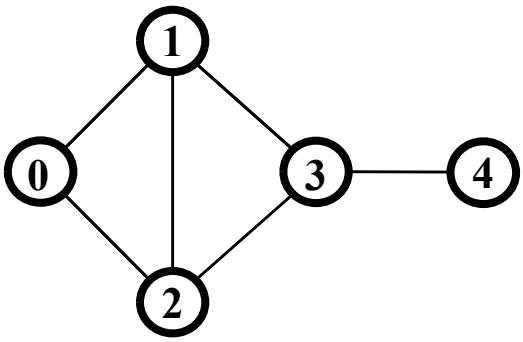
Busca em Profundidade - Exemplo detalhado

Pseudocódigo

```
visited = [False] * n_vertices
dfs_iter(v)
   stack.append(v)
   visited[v] = True
   while stack not empty
      v = stack.pop()
      for w in neighbors(v)
         if not visited[w]
            stack.append(w)
            visited[w] = True
```

Desempilha vértice no topo

dfs(0): 0, 1, 2, 3, 4



v = 4
stack = {1}

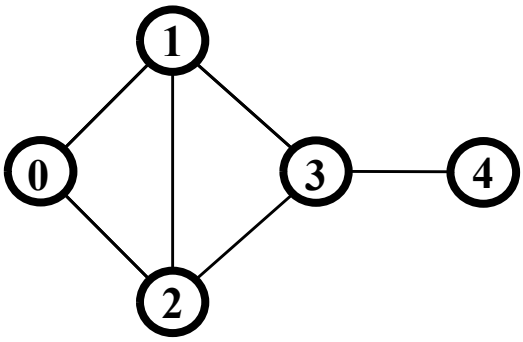
Busca em Profundidade - Exemplo detalhado

Pseudocódigo

```
visited = [False] * n_vertices
dfs_iter(v)
   stack.append(v)
   visited[v] = True
   while stack not empty
      v = stack.pop()
      for w in neighbors(v)
         if not visited[w]
            stack.append(w)
            visited[w] = True
```

Checa vizinhos de 4 por não visitados, todos já foram

dfs(0): 0, 1, 2, 3, 4



v = 4
stack = {1}

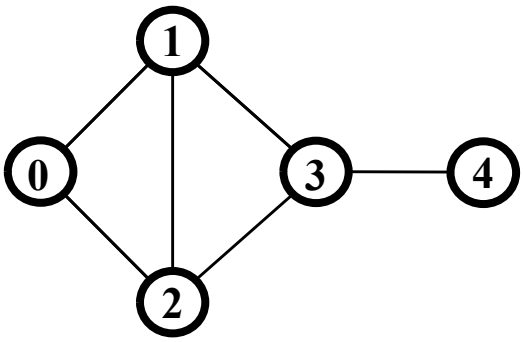
Busca em Profundidade - Exemplo detalhado

Pseudocódigo

```
visited = [False] * n_vertices
dfs_iter(v)
   stack.append(v)
   visited[v] = True
   while stack not empty
      v = stack.pop()
      for w in neighbors(v)
         if not visited[w]
            stack.append(w)
            visited[w] = True
```

Desempilha vértice no topo

dfs(0): 0, 1, 2, 3, 4



v = 1
stack = {}

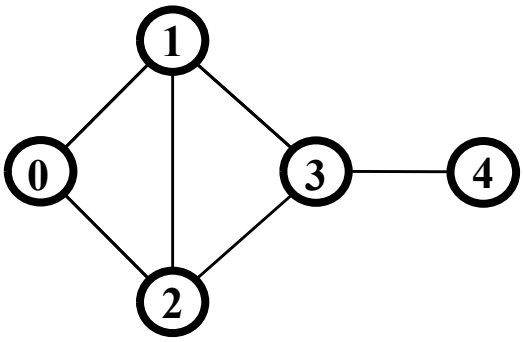
Busca em Profundidade - Exemplo detalhado

Pseudocódigo

```
visited = [False] * n_vertices
dfs_iter(v)
   stack.append(v)
   visited[v] = True
   while stack not empty
      v = stack.pop()
      for w in neighbors(v)
         if not visited[w]
            stack.append(w)
            visited[w] = True
```

Checa vizinhos de 1 por não visitados, todos já foram

dfs(0): 0, 1, 2, 3, 4



v = 1
stack = {}

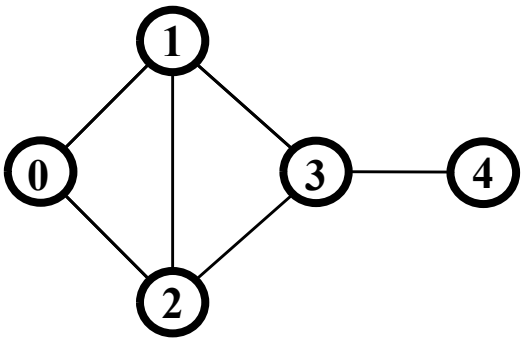
Busca em Profundidade - Exemplo detalhado

Pseudocódigo

```
visited = [False] * n_vertices
dfs_iter(v)
   stack.append(v)
   visited[v] = True
   while stack not empty
      v = stack.pop()
      for w in neighbors(v)
         if not visited[w]
            stack.append(w)
            visited[w] = True
```

Pilha vazia, encerra execução

dfs(0): 0, 1, 2, 3, 4



v = 1
stack = {}

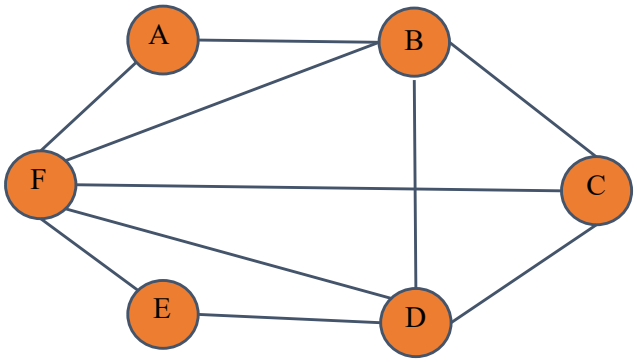
Busca em Profundidade - Complexidade

- A função DFS é chamada exatamente uma vez para cada vértice v, sendo que existem |V| vértices, então O(|V|) vezes no total
- A cada chamada de função, o laço é executado |adj[v]| vezes, logo será executado O(|A|) vezes no total

Somando estas duas, a complexidade da DFS é **O(|V|+|A|)**

Exercícios de Fixação

Execute a DFS no grafo abaixo começando pelo vértice A

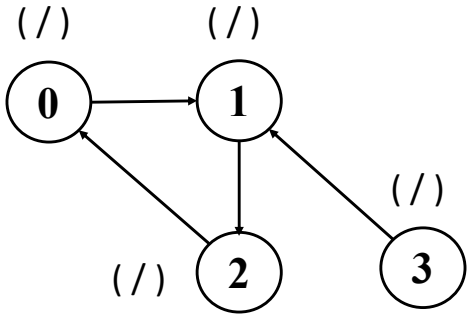


Busca em Profundidade - Tempos de entrada e saída

- Na DFS pode ser útil registrar o tempo de descoberta e o tempo de término de cada vértice.

Busca em Profundidade - Tempos de entrada e saída

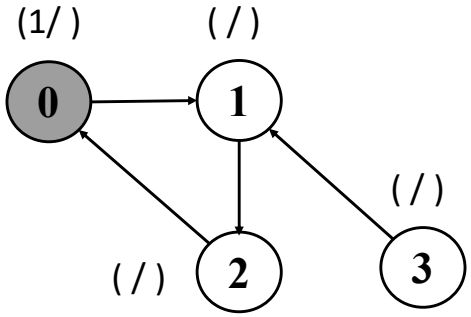
- Vamos percorrer o seguinte grafo registrando os tempos de descoberta (cinza) e o tempo de término (laranja).



(tempo de descoberta / tempo de término)

Busca em Profundidade - Tempos de entrada e saída

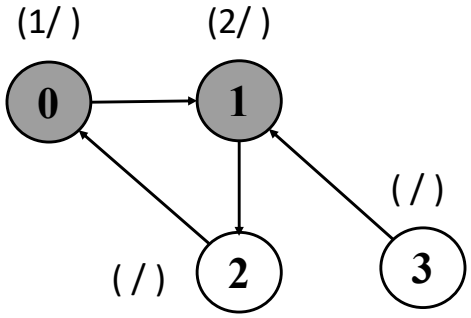
- Vamos percorrer o seguinte grafo registrando os tempos de descoberta (cinza) e o tempo de término (laranja).



(tempo de descoberta / tempo de término)

Busca em Profundidade - Tempos de entrada e saída

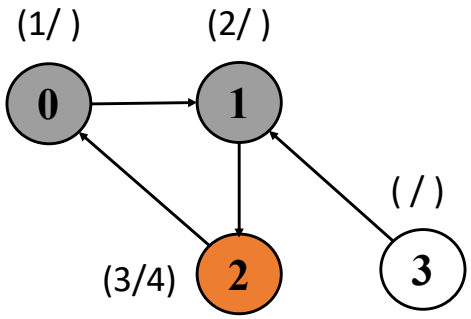
- Vamos percorrer o seguinte grafo registrando os tempos de descoberta (cinza) e o tempo de término (laranja).



(tempo de descoberta / tempo de término)

Busca em Profundidade - Tempos de entrada e saída

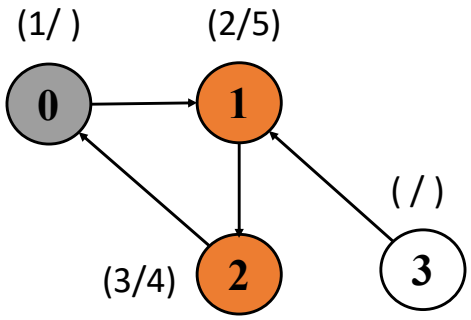
- Vamos percorrer o seguinte grafo registrando os tempos de descoberta (cinza) e o tempo de término (laranja).



(tempo de descoberta / tempo de término)

Busca em Profundidade - Tempos de entrada e saída

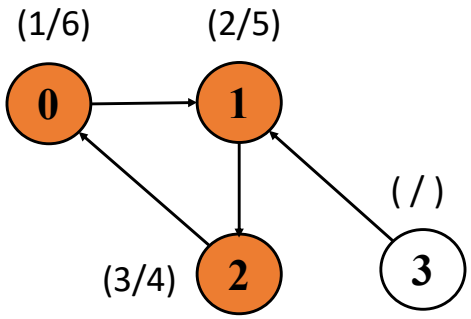
- Vamos percorrer o seguinte grafo registrando os tempos de descoberta (cinza) e o tempo de término (laranja).



(tempo de descoberta / tempo de término)

Busca em Profundidade - Tempos de entrada e saída

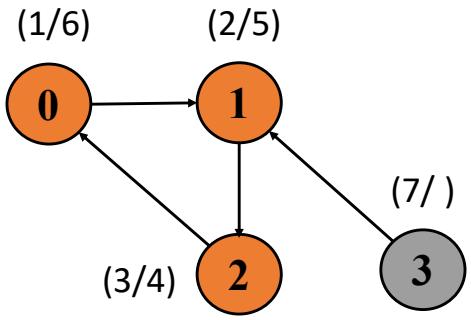
- Vamos percorrer o seguinte grafo registrando os tempos de descoberta (cinza) e o tempo de término (laranja).



(tempo de descoberta / tempo de término)

Busca em Profundidade - Tempos de entrada e saída

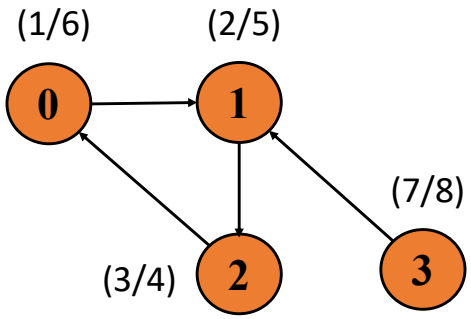
- Vamos percorrer o seguinte grafo registrando os tempos de descoberta (cinza) e o tempo de término (laranja).



(tempo de descoberta / tempo de término)

Busca em Profundidade - Tempos de entrada e saída

- Vamos percorrer o seguinte grafo registrando os tempos de descoberta (cinza) e o tempo de término (laranja).



(tempo de descoberta / tempo de término)

Ordenação Topológica

https://devsuperior.com.br

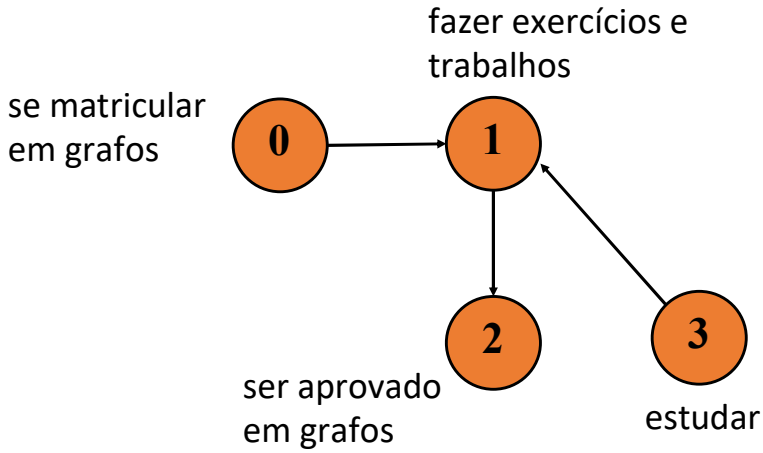
Prof. Dr. Nelio Alves

Definição

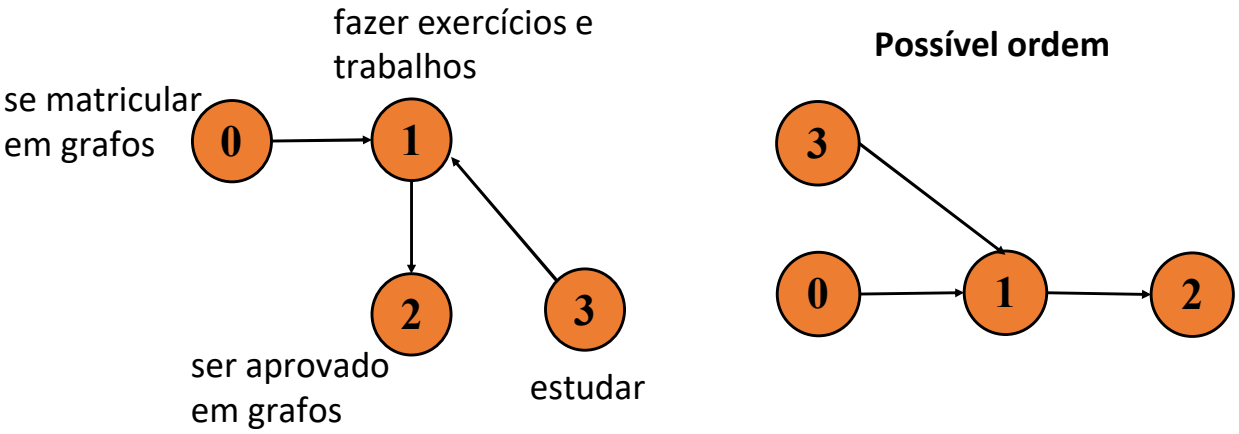
Ordenação Topológica: é uma ordenação linear dos vértices de um grafo direcionado acíclico tal que um vértice u precede um vértice v se existe uma aresta (u, v)

- Útil para programar a execução de uma sequência de tarefas que dependem de outras
  - para construir um edifício, é preciso solidificar a base, para depois colocar os pilares e depois subir outros andares
  - certas tarefas/disciplinas podem ser executadas/cursadas simultaneamente, outras não

Exemplo



Exemplo



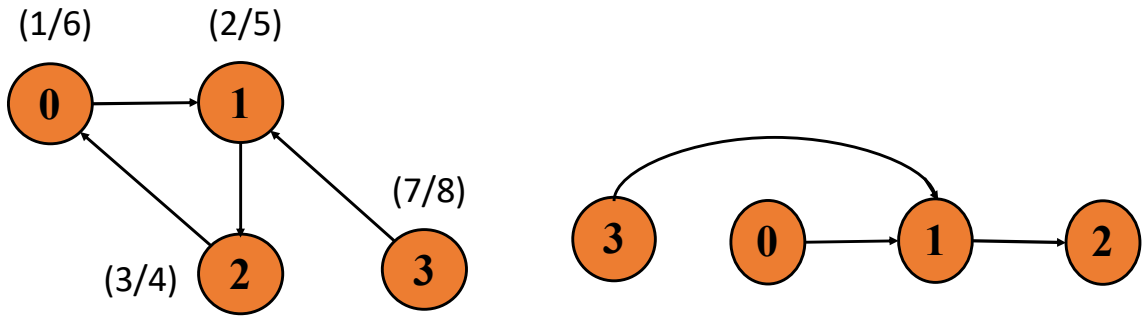
Algoritmo - Ordenação topológica com DFS

1. Faça a busca em profundidade no grafo
   a. Ao término do processamento de cada vértice, insira o vértice no início de uma lista linear

1. Ao percorrer a lista do começo ao final, tem-se a ordenação topológica do grafo!

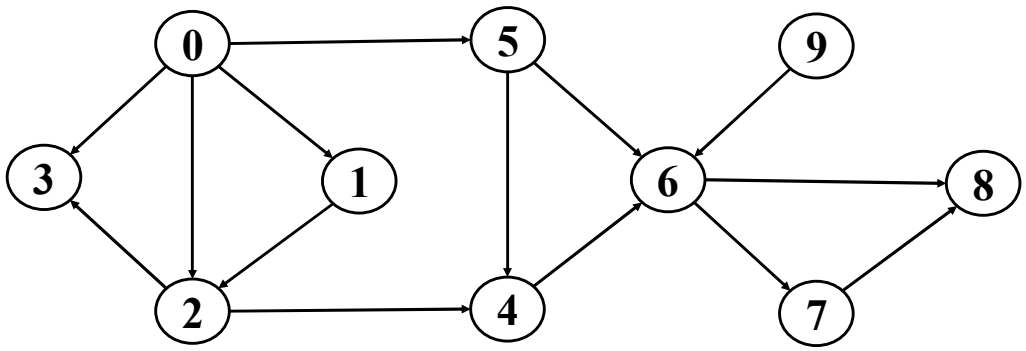
Algoritmo - Ordenação topológica com DFS

Exemplo



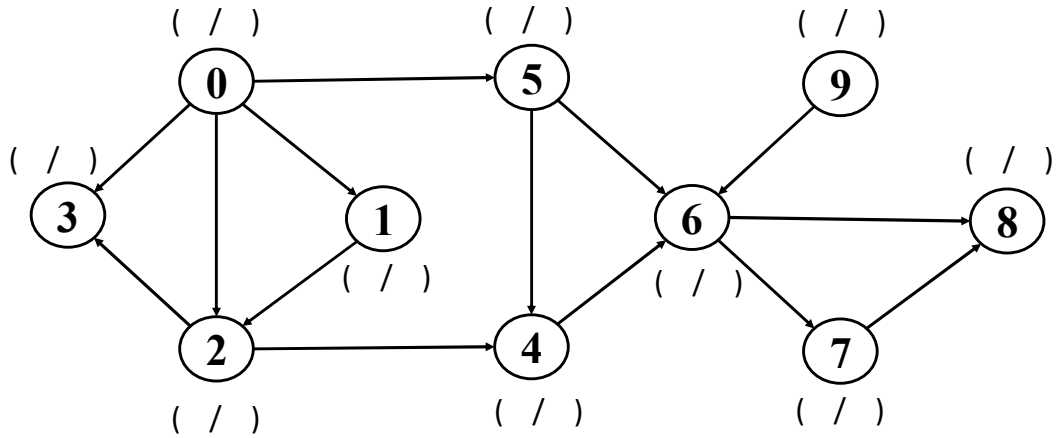
Ordenação topológica

Faça a ordenação topológica do grafo abaixo.



Ordenação topológica

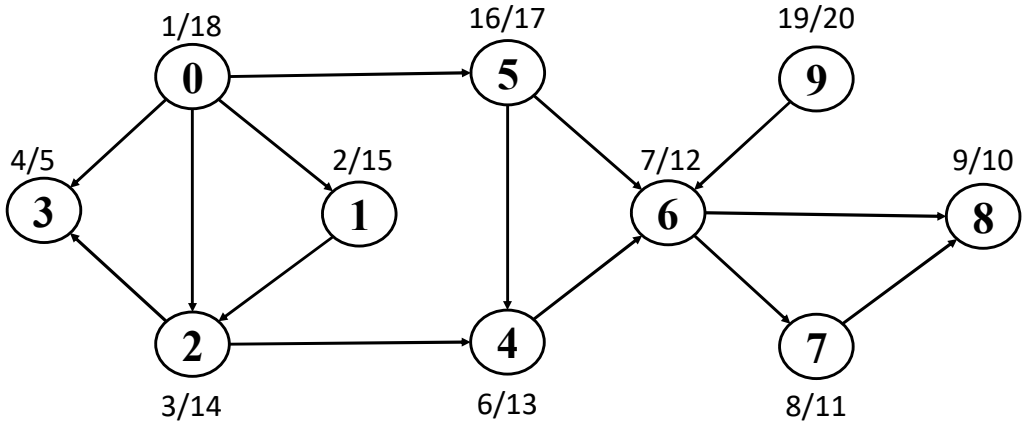
Faça a ordenação topológica do grafo abaixo.



Preencha com os tempos de visita, adicione cada vértice finalizado a uma lista. Ao fim teremos a ordenação topológica.

Ordenação topológica

Faça a ordenação topológica do grafo abaixo.

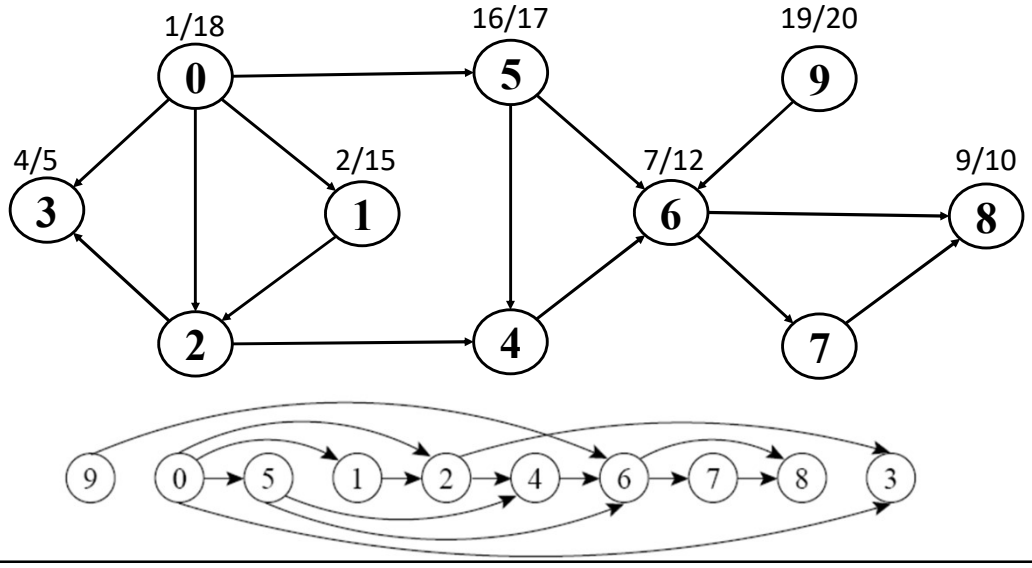


Lista de ordenação topológica*: 9 0 5 1 2 4 6 7 8 3

*Vértices com maior tempo de término aparecem primeiro

Ordenação topológica

Faça a ordenação topológica do grafo abaixo.



Ordenação topológica

- Como alterar o algoritmo de busca em profundidade para realizar a ordenação topológica?
  - Após finalizar visita a um vértice, inserir vértice no início de uma lista

- Qual a complexidade de tempo do algoritmo?
  - **O(|V|+|A|)**

Ordenação topológica

Observações:

- A ordenação topológica **pode não ser única**
- **Não é possível** gerar uma ordenação topológica em grafos com ciclos