

Prof. Hélder Pereira Borges

helder@ifma.edu.br

Grafos Buscas



Busca em Grafos



- Processo de **explorar** um grafo de modo **sistemático**, seguindo pelas arestas de modo a visitar seus vértices
- Visitar **todos** os vértices ou apenas um **subconjunto**? Depende do contexto
- Uma busca finaliza quando se **encontra** o objeto procurado ou quando todos os vértices tiverem sido **visitados**
- Principais tipos de busca em grafos
 - Busca em Profundidade (DFS)
 - Busca em Largura (BFS)
 - Busca pelo Menor Caminho

~~Largura~~ X Profundidade

- BFS
 - Encontrar o caminho mais curto entre os vértices **V** e **W**.
 - Ou, dado um estado inicial, encontrar o menor número de passos para alcançar o estado final
- DFS
 - Verificar todas as possibilidades para identificar qual é a melhor
- BFS ou DFS
 - Verificar a conectividade entre dois vértices
 - Ou se, a partir de v, é possível chegar em w

Largura
~~BFS~~

X Produtividade
~~DFS~~

vantagens

- Encontra o caminho mais curto entre vértices
- Encontra soluções ótimas
- Encontra vértice próximo em menos tempo

desvantagens

- Consumo memória para armazenar todos os vértices

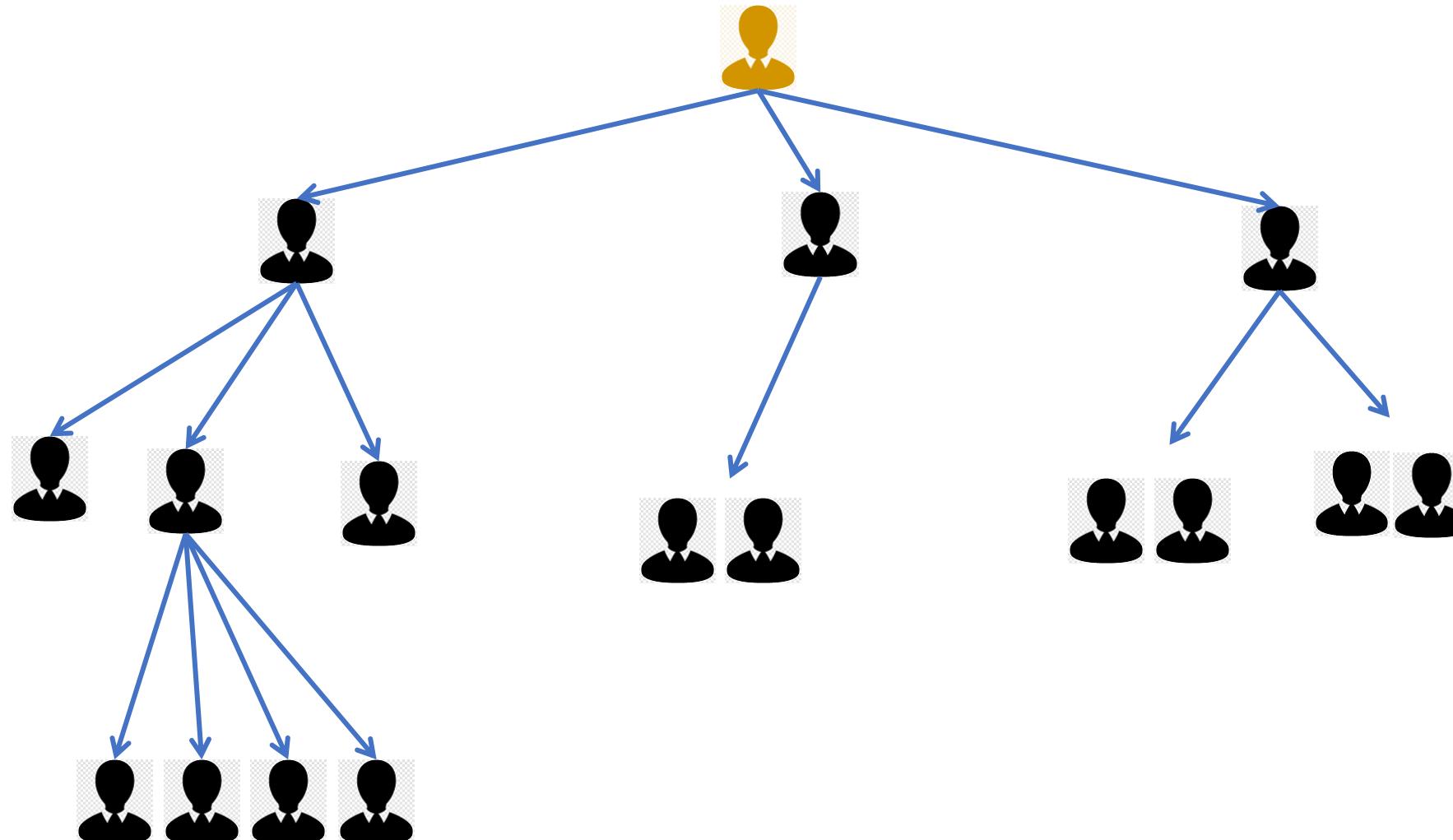
- Consome menos memória
- Encontra vértice distante em menos tempo

desvantagens

- Pode não encontrar a solução ideal para o problema
- Pode ficar preso em busca de caminhos inúteis

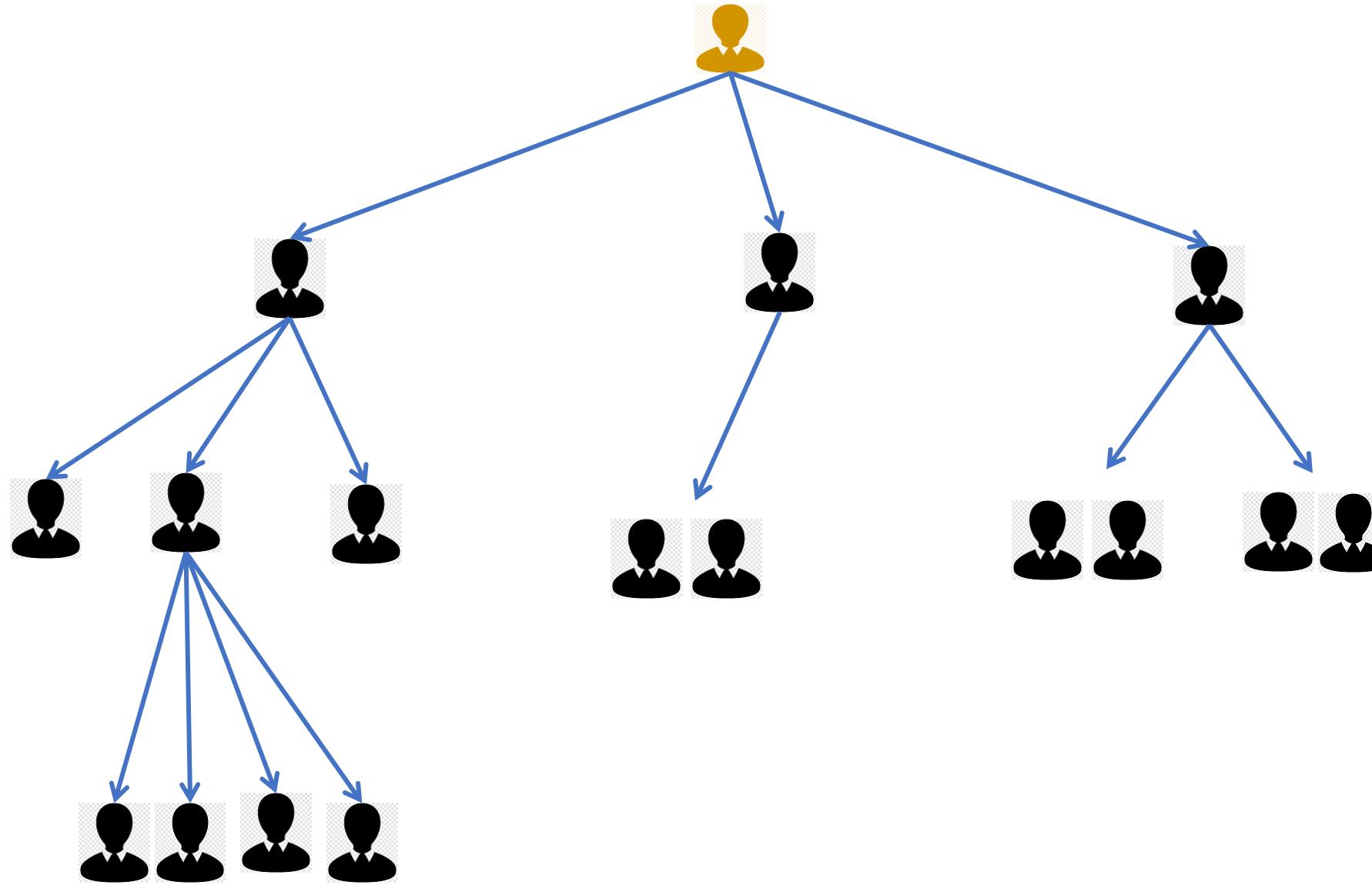
Exemplo

- Uma empresa com muitos departamentos e com uma hierarquia
 - CEO >> Diretores >> Gerentes >> ...



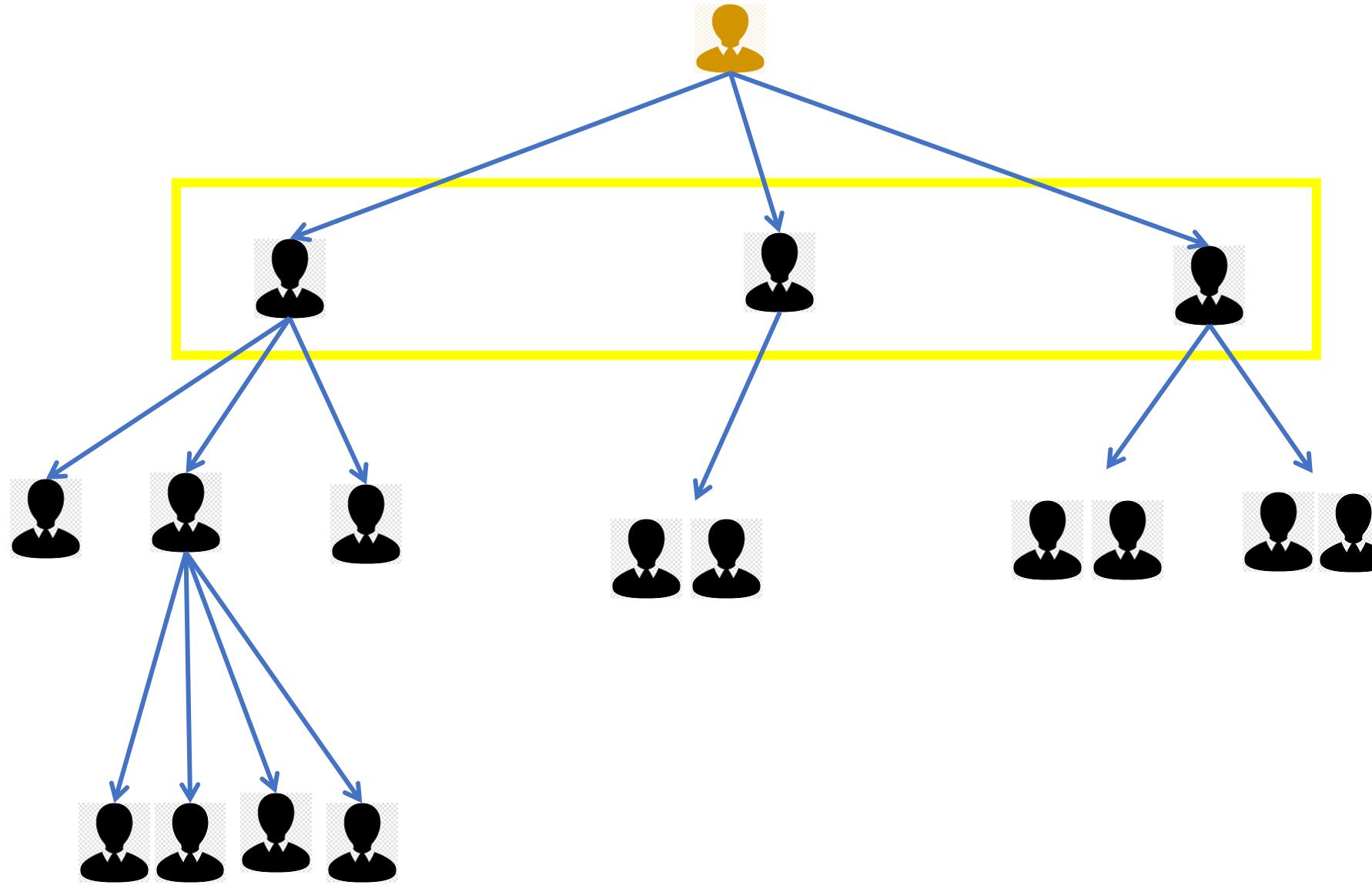
Situação 1: Metas obrigatórias da empresa

... estudar e praticar ... the best way for everything



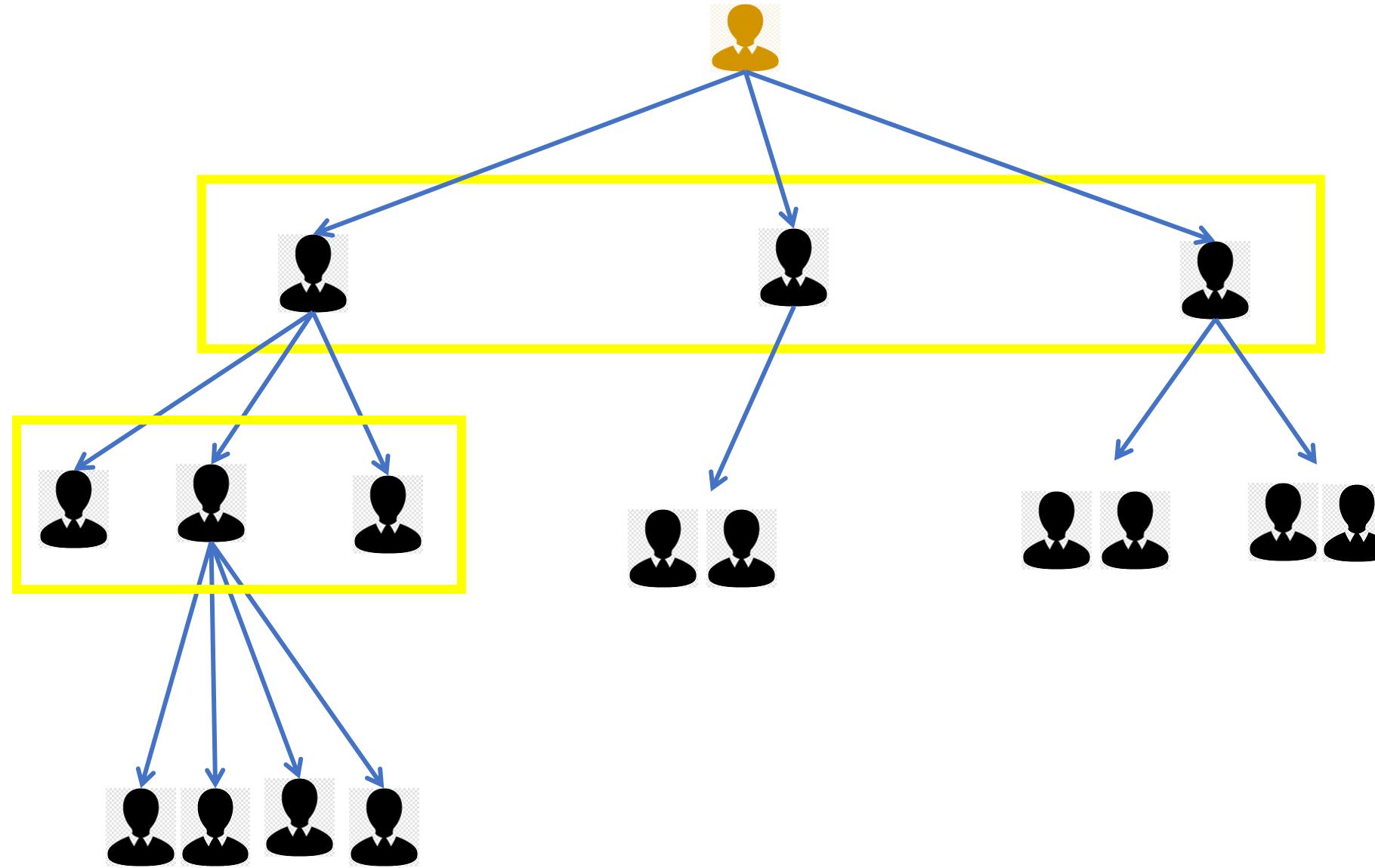
Situação 1: Metas obrigatórias da empresa

... estudar e praticar ... the best way for everything



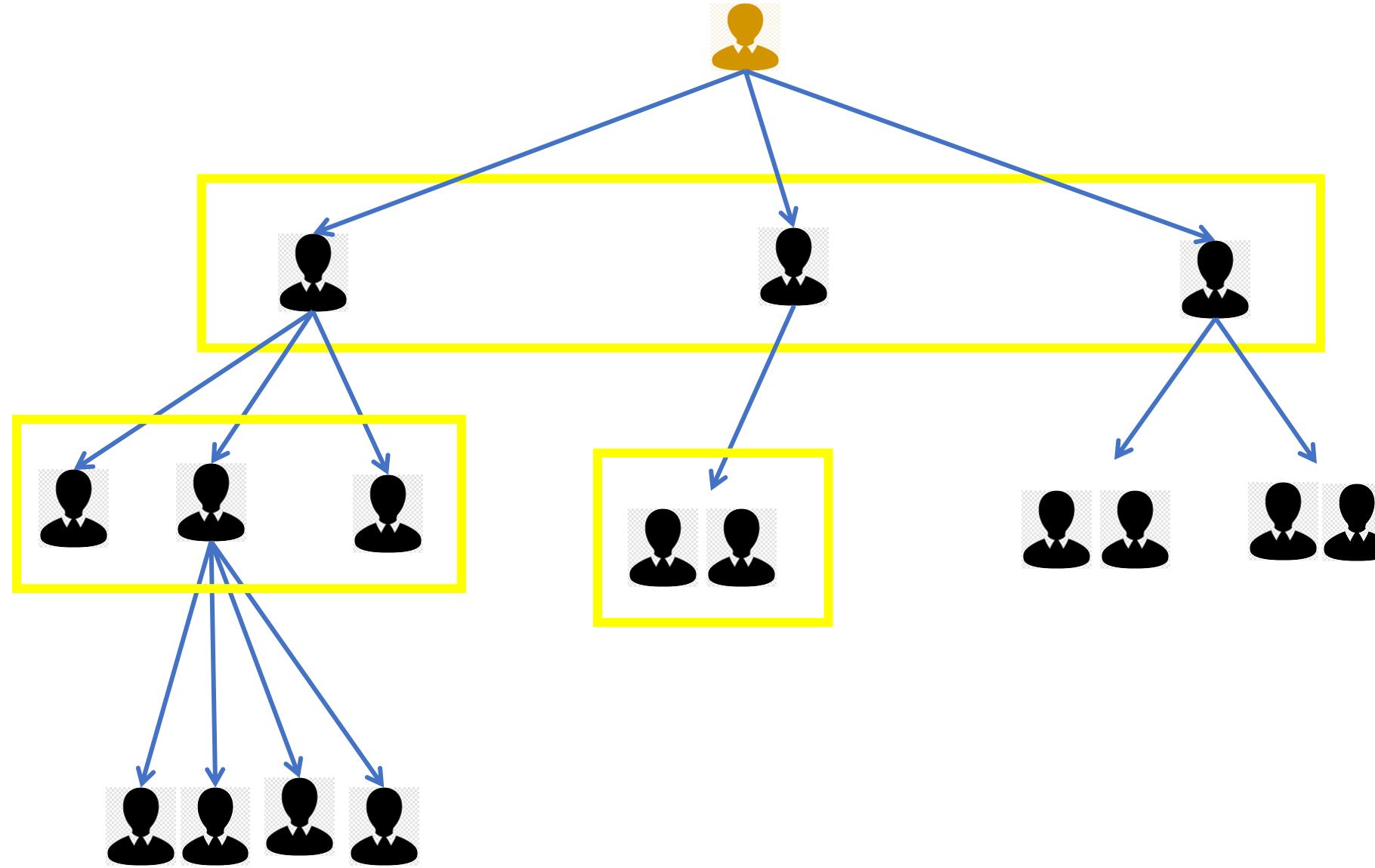
Situação 1: Metas obrigatórias da empresa

... estudar e praticar ... the best way for everything



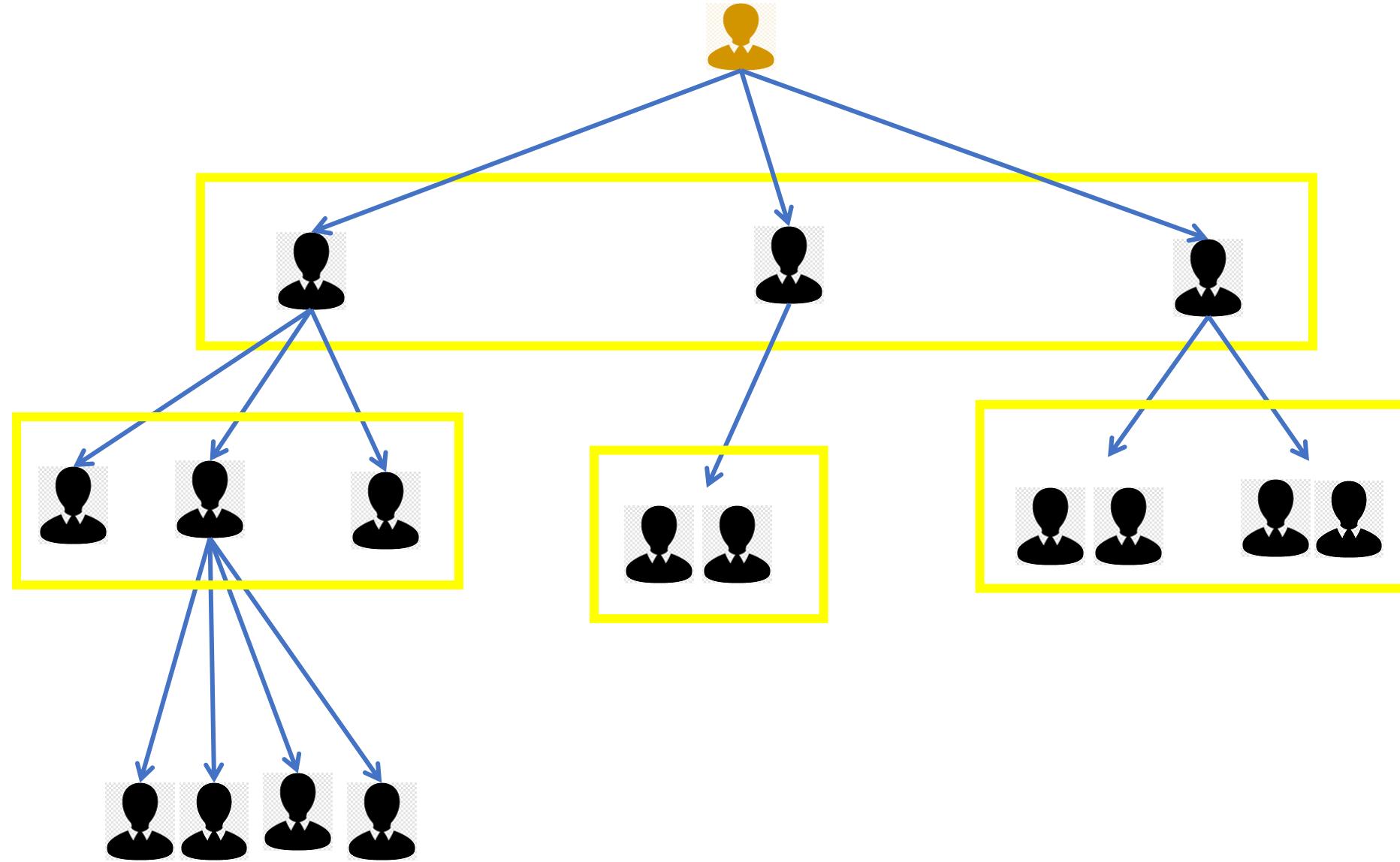
Situação 1: Metas obrigatórias da empresa

... estudar e praticar ... the best way for everything



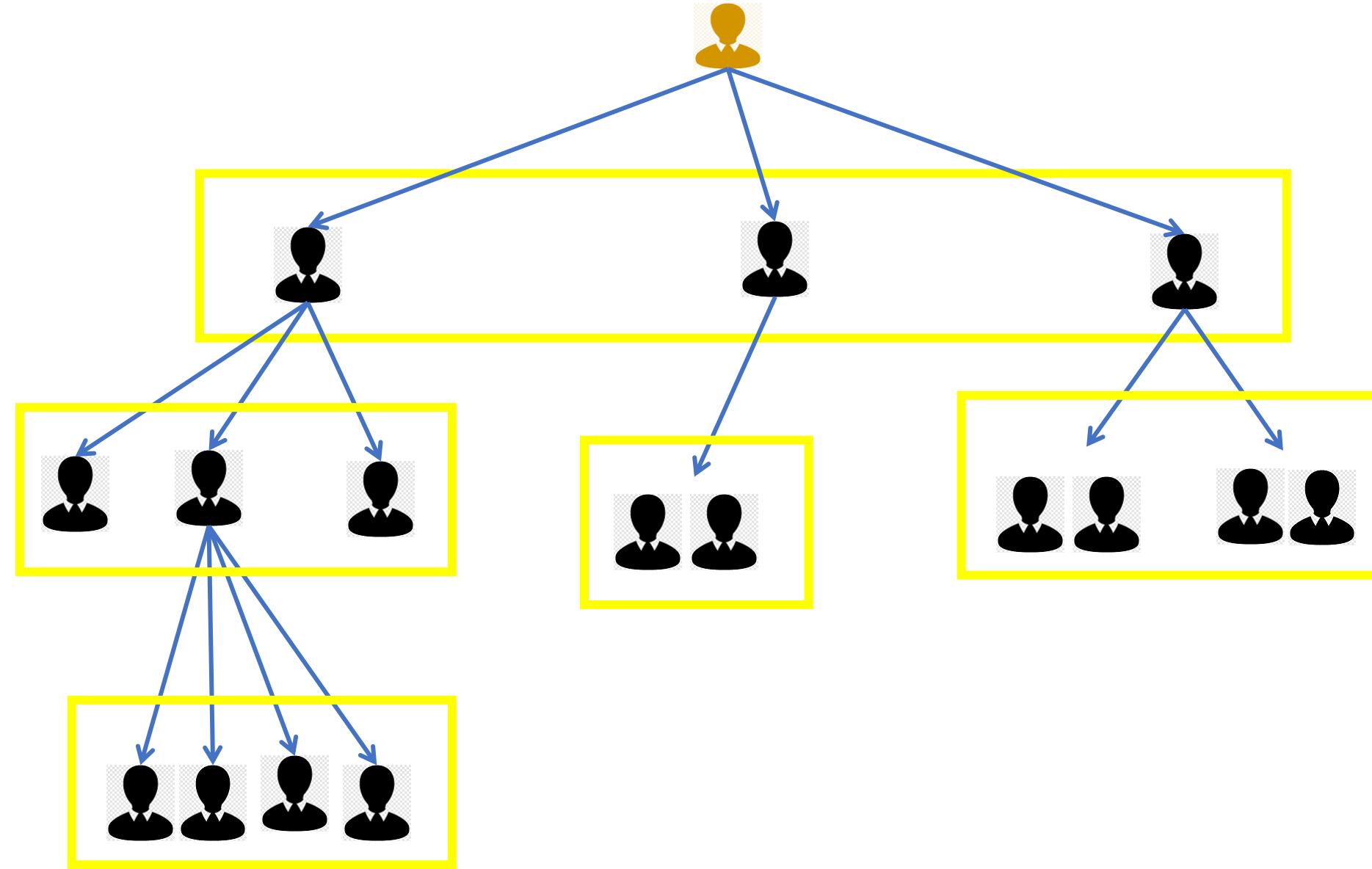
Situação 1: Metas obrigatórias da empresa

... estudar e praticar ... the best way for everything



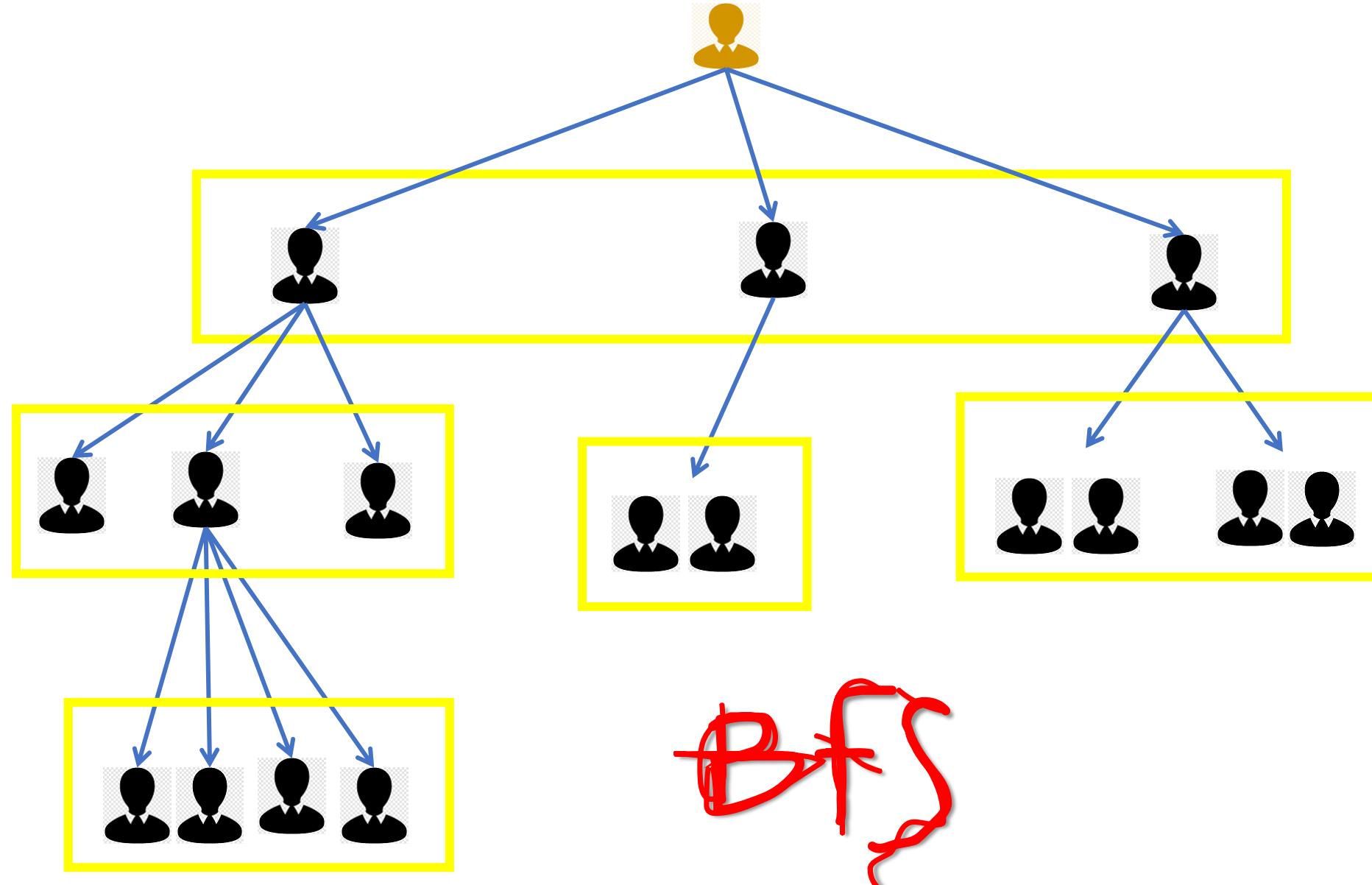
Situação 1: Metas obrigatórias da empresa

... estudar e praticar ... the best way for everything



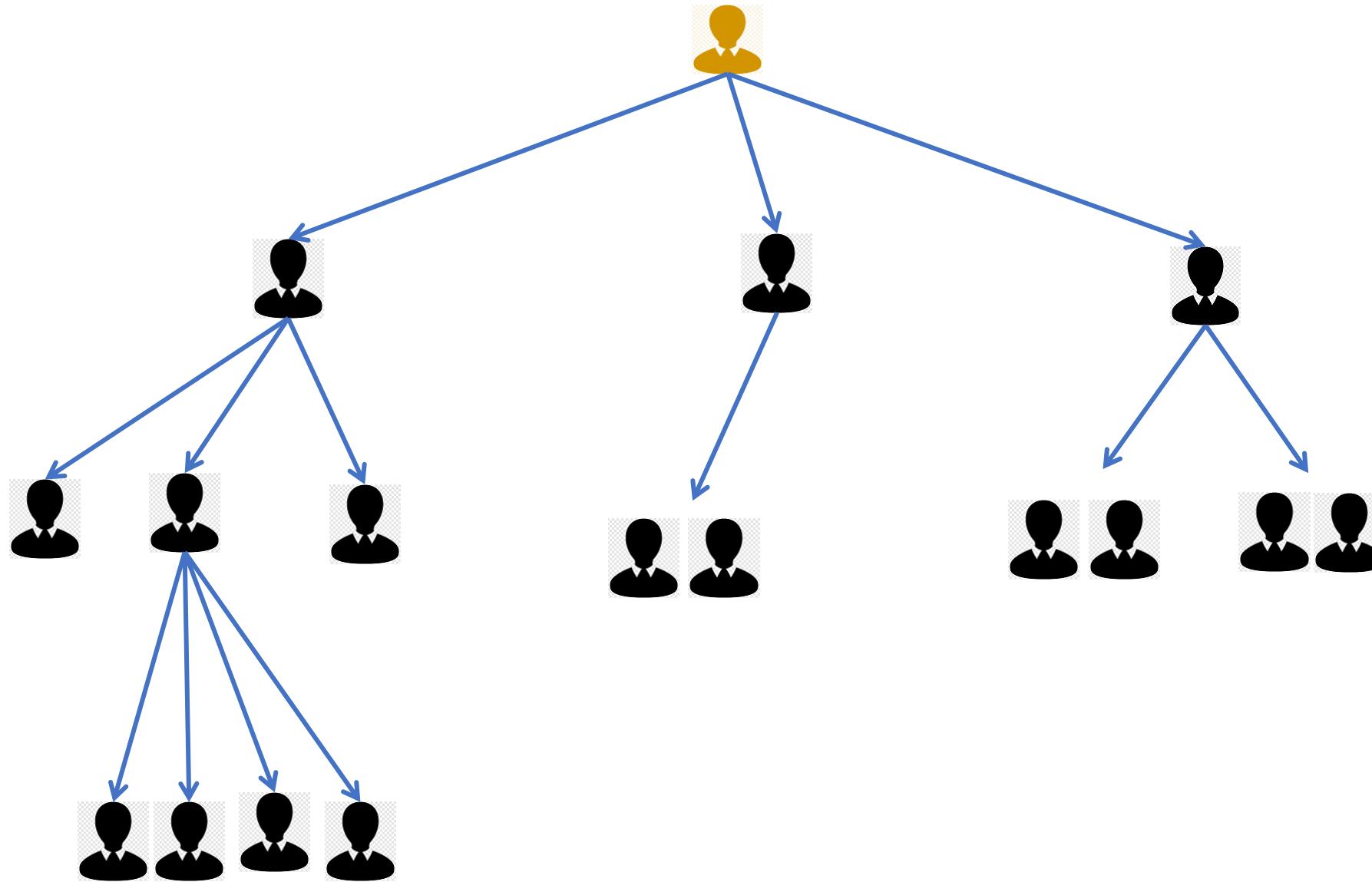
Situação 1: Metas obrigatórias da empresa

... estudar e praticar ... the best way for everything



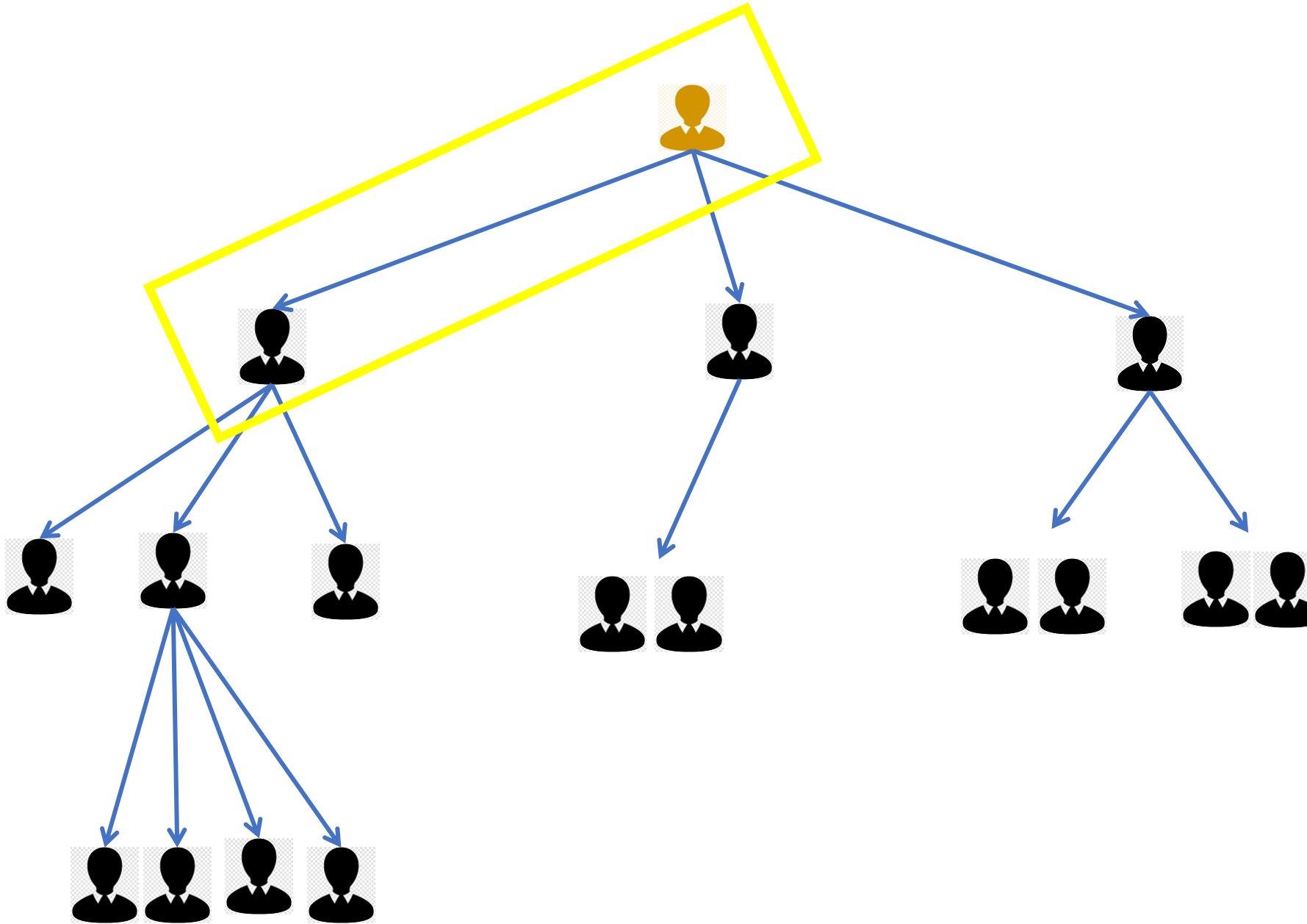
Situação 2: Implementar uma inovação

... estudar e praticar ... the best way for everything



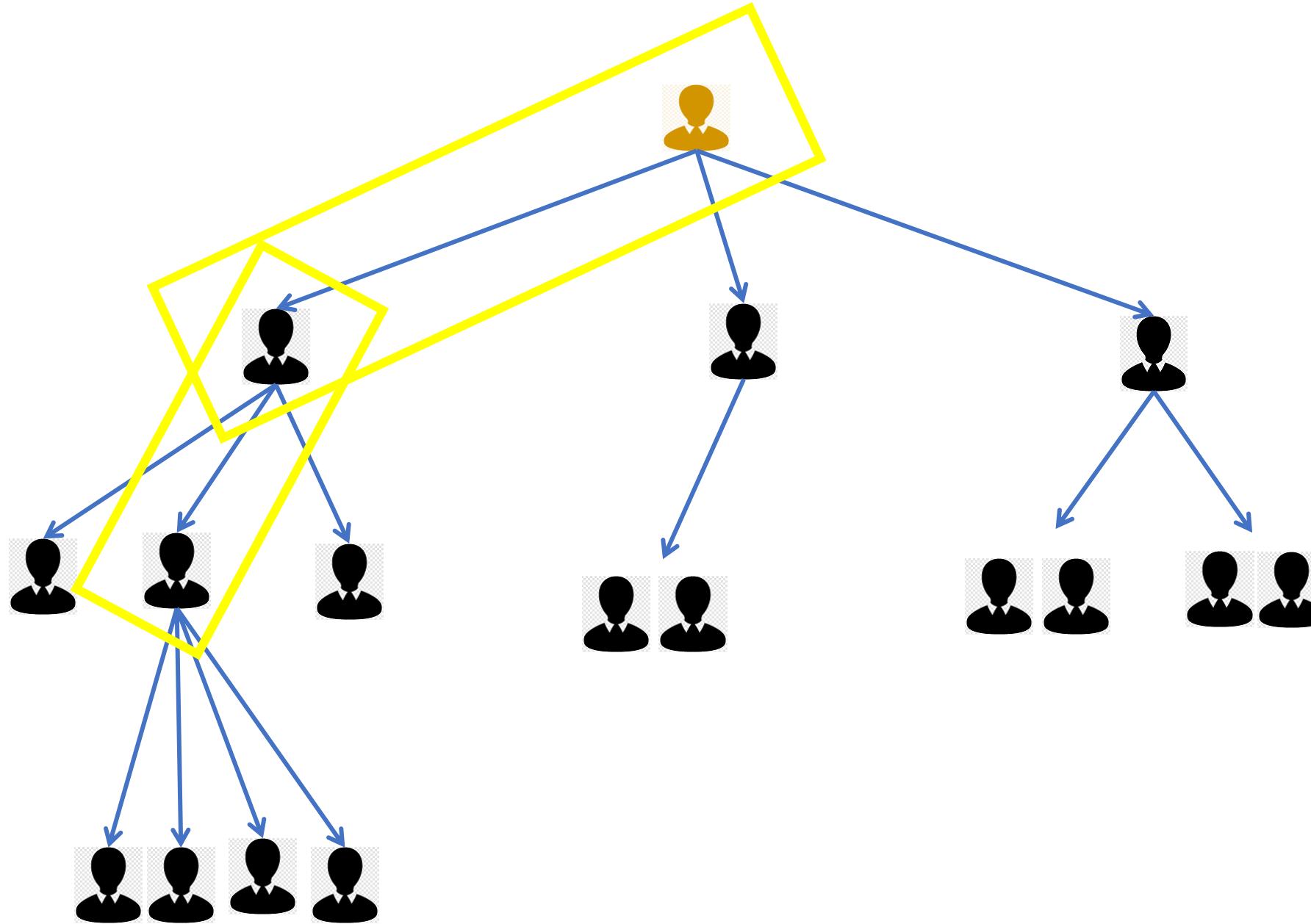
Situação 2: Implementar uma inovação

... estudar e praticar ... the best way for everything



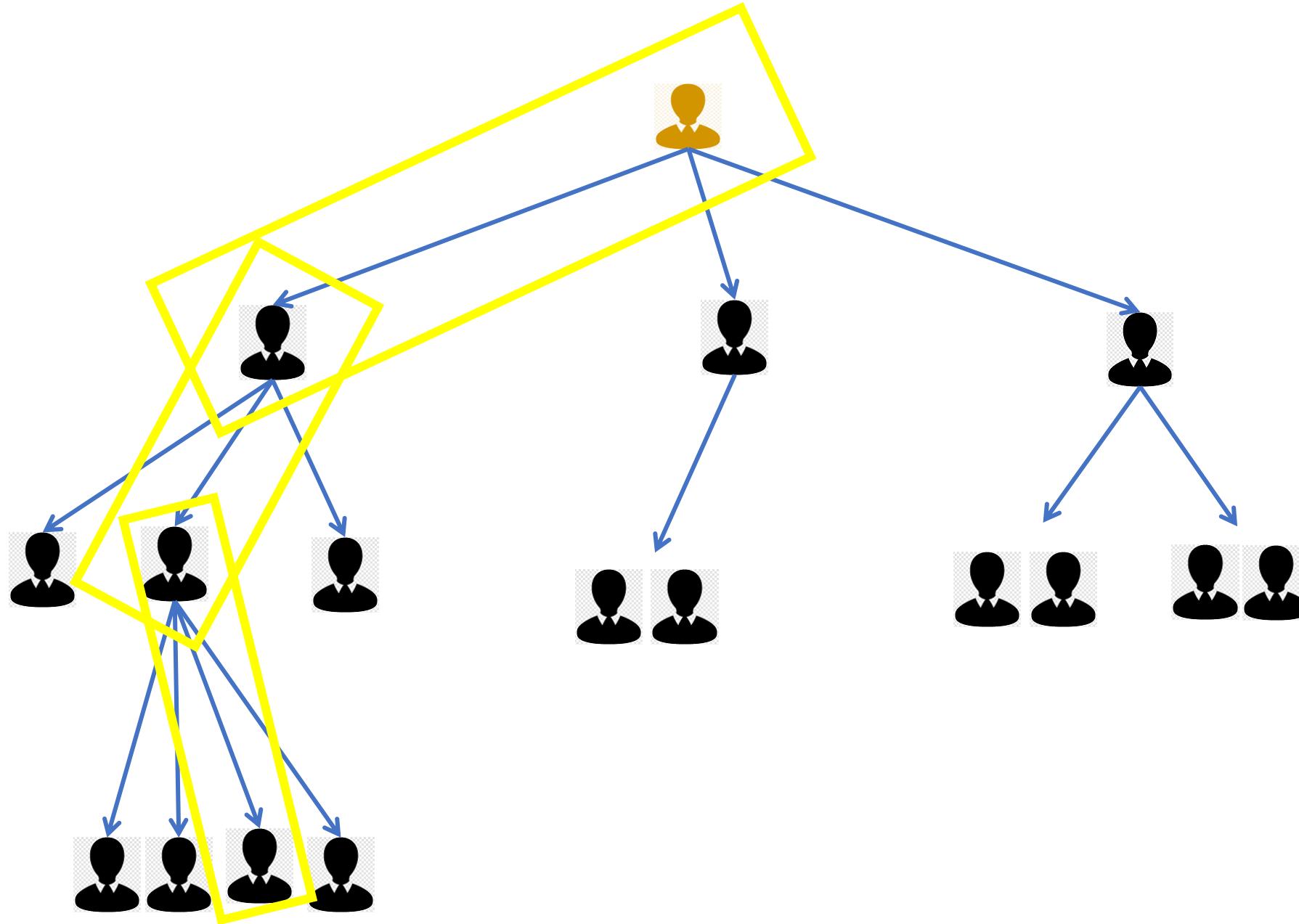
Situação 2: Implementar uma inovação

... estudar e praticar ... the best way for everything



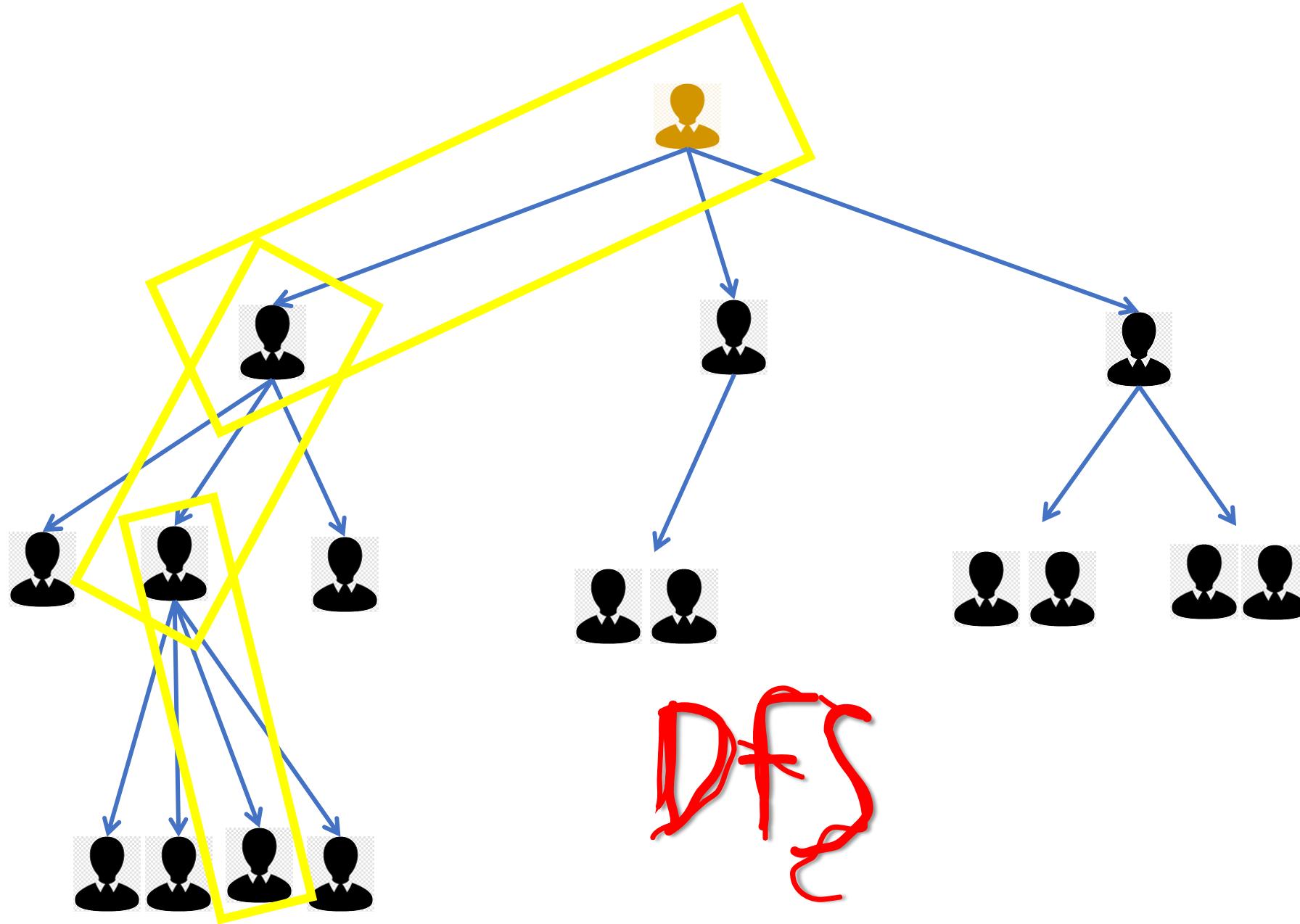
Situação 2: Implementar uma inovação

... estudar e praticar ... the best way for everything



Situação 2: Implementar uma inovação

... estudar e praticar ... the best way for everything



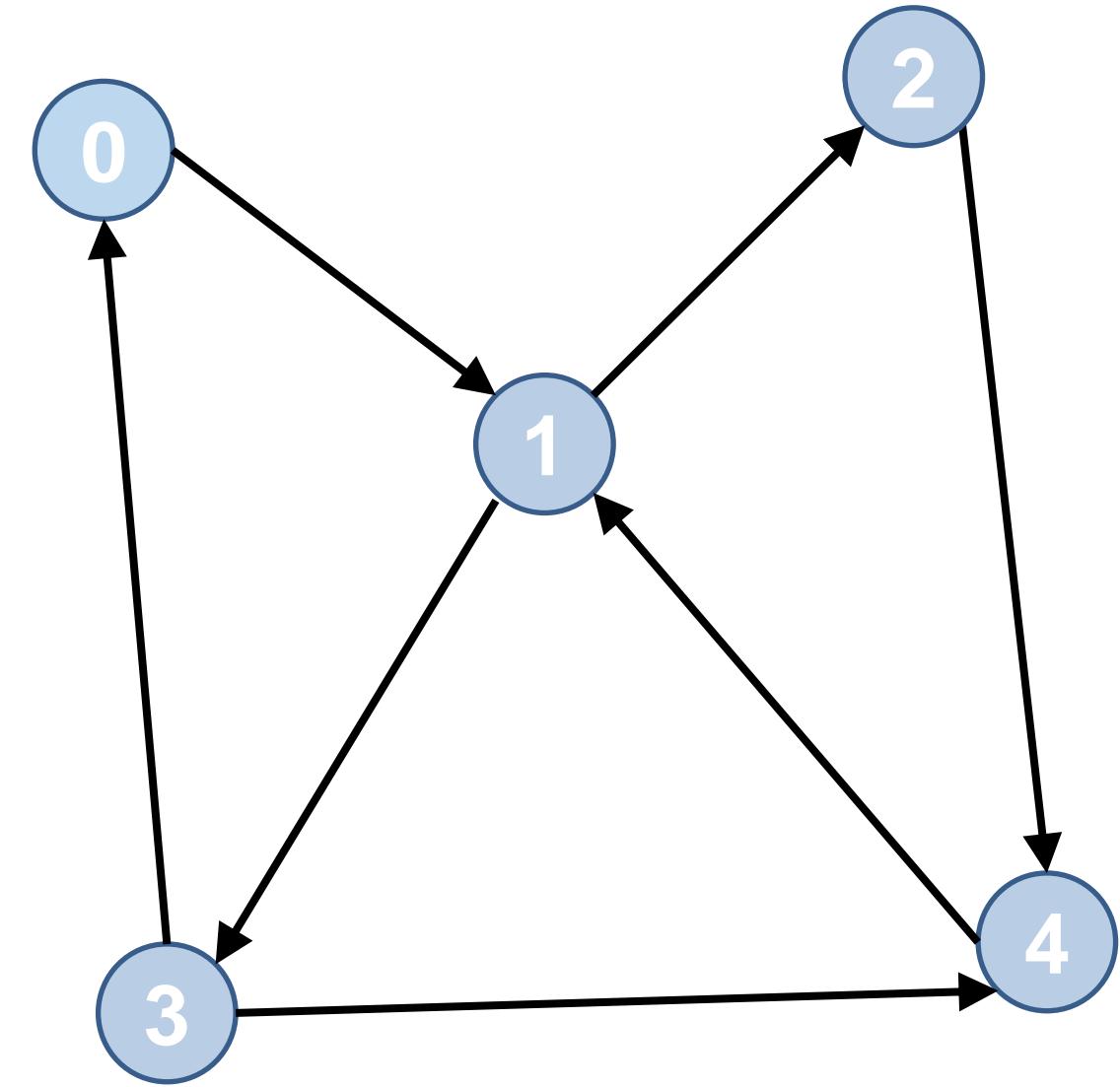
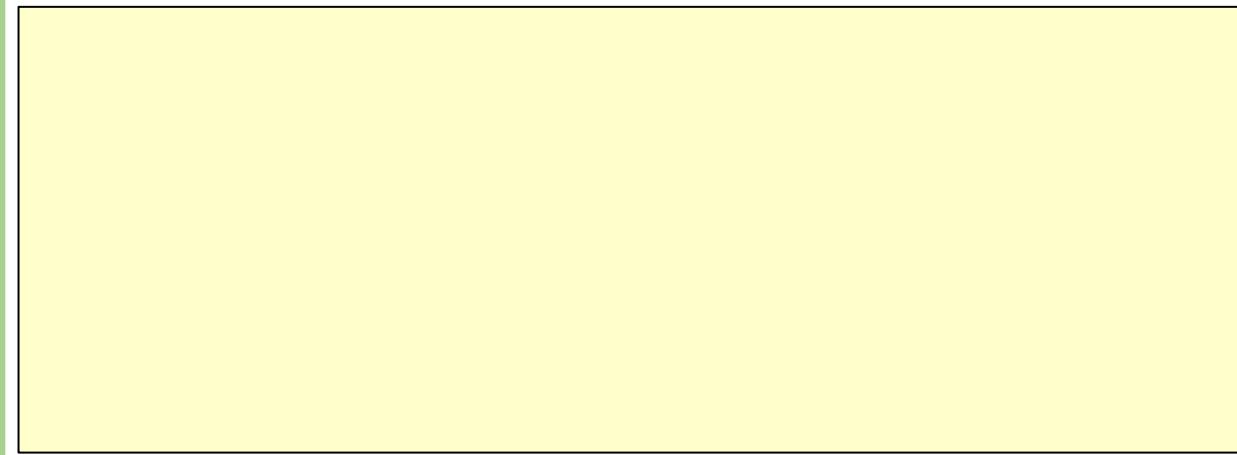
**Busca em
Profundidade**

Busca em Profundidade - Depth-First Search (DFS)

- Inicia a partir de um **vértice inicial** arbitrário
 - Explora todos os vértices que são vizinhos atingíveis do vértice inicial
 - Vizinho > vizinho do vizinho > vizinho do vizinho do vizinho
 - Quando chegar em um vértice sem vizinhos, retrocede ao pai (**backtracking**) e segue por outro vizinho
 - Caso sobrem vértices não visitados um novo vértice inicial deve ser definido
- Aprofunda nos vértices vizinhos até encontrar
 - O alvo da busca
 - Ou um vértice sem vizinhos que possam ser visitados

Busca em Profundidade

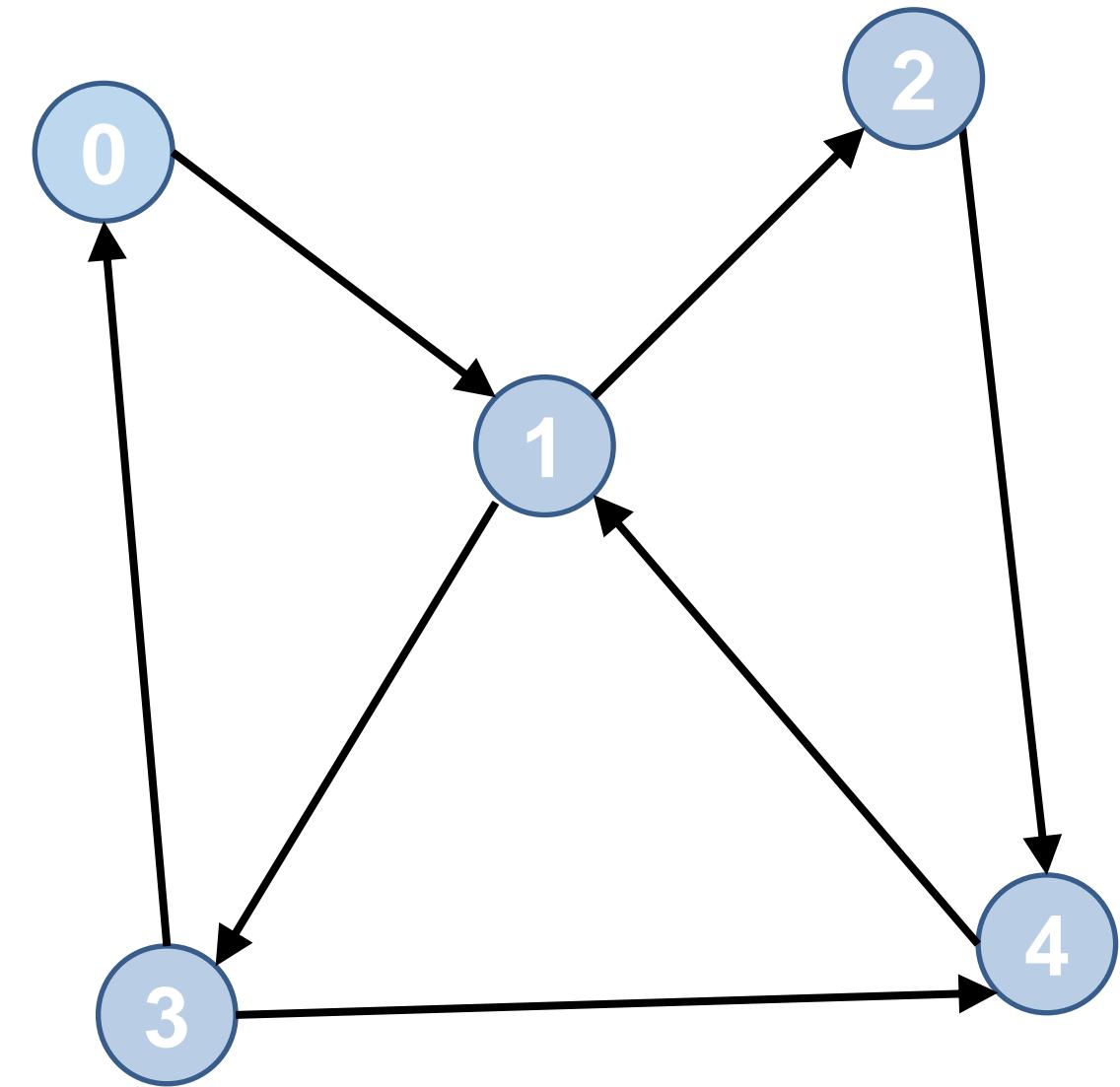
Procurar o 5



Busca em Profundidade

Procurar o 5

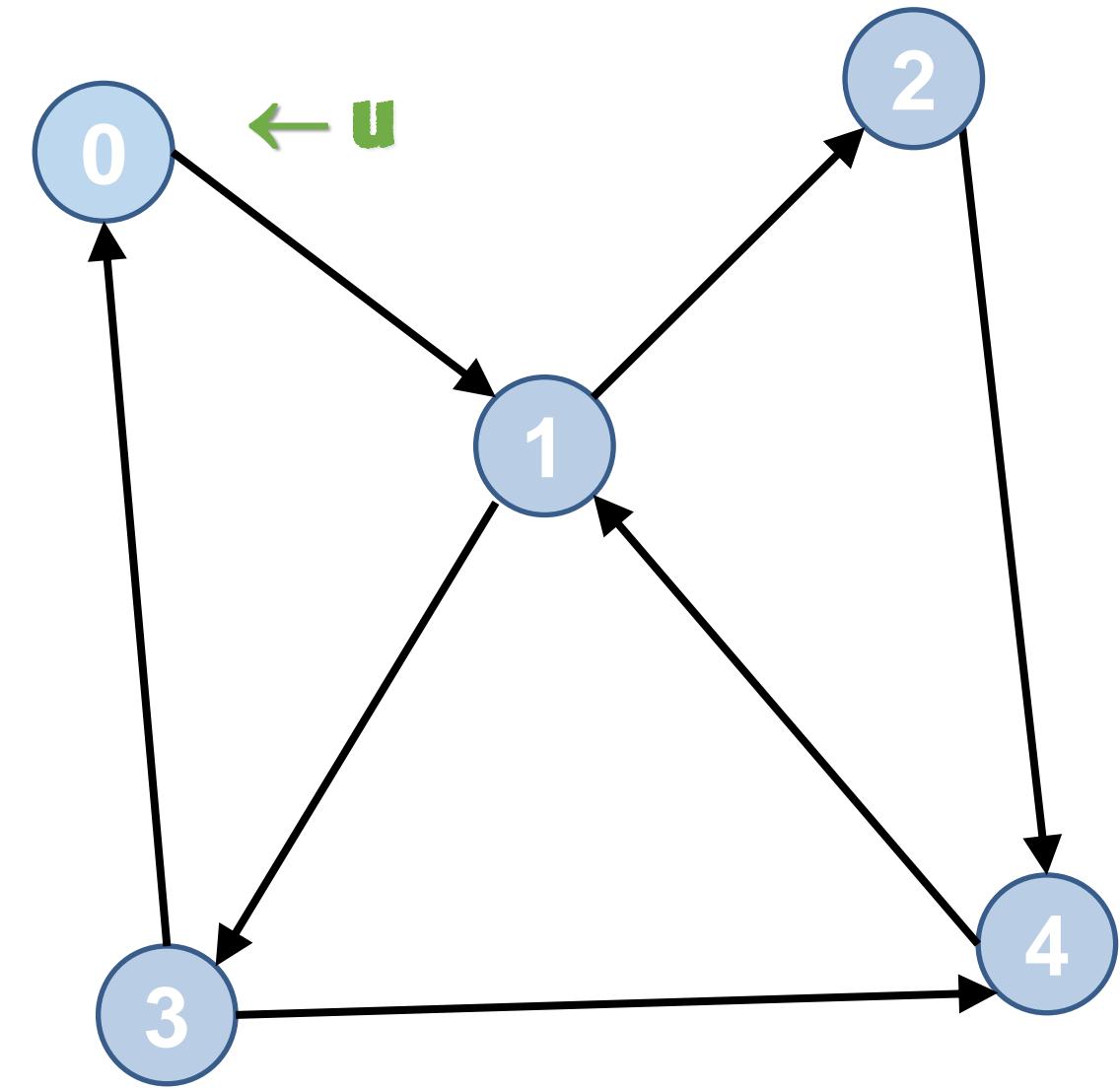
01 Escolher nó inicial (**u**)



Busca em Profundidade

Procurar o 5

01 Escolher nó inicial (**u**)

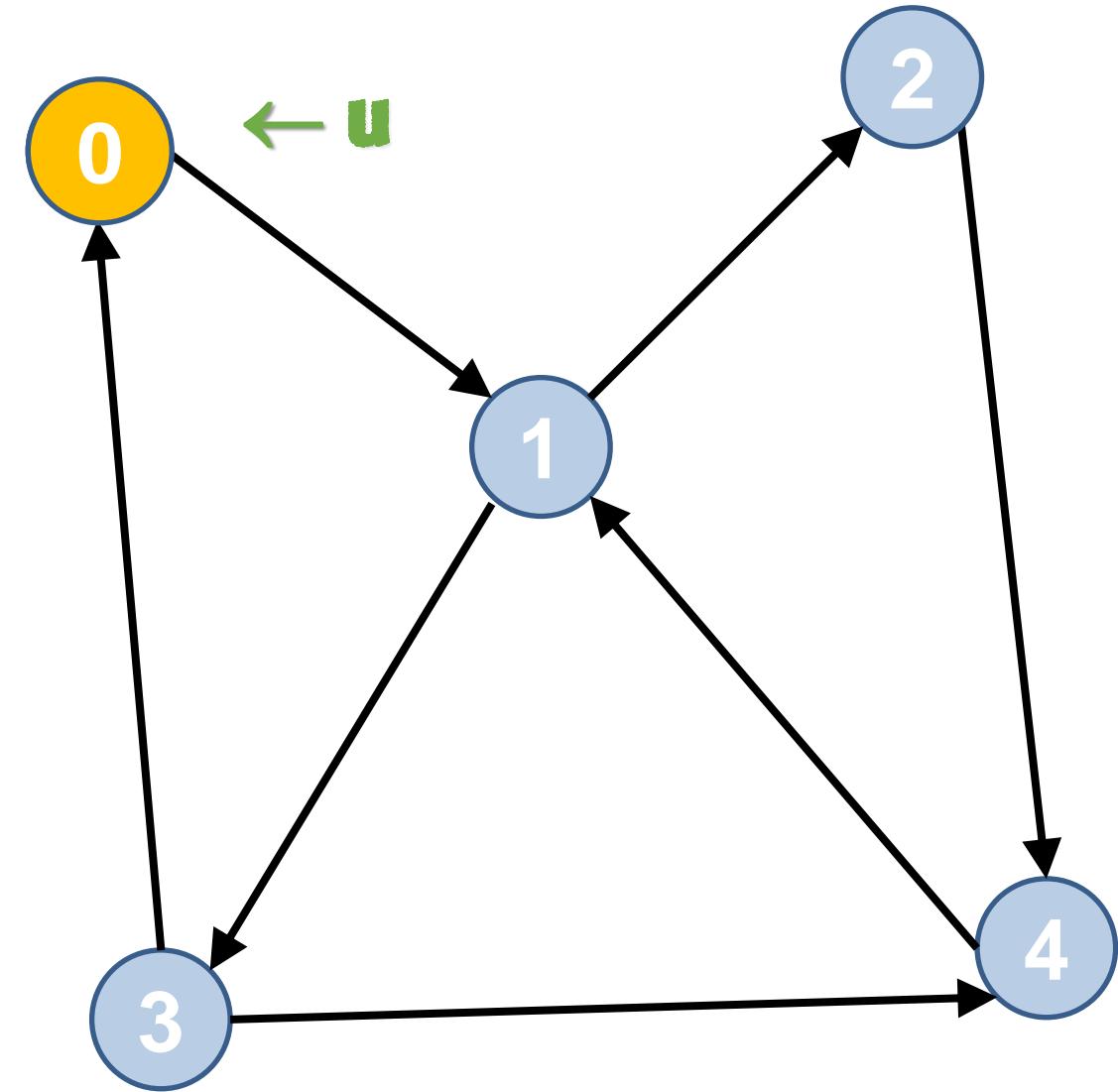


Busca em Profundidade

Procurar o 5

01 Escolher nó inicial (**u**)

02 Visitar **u** //testar



Busca em Profundidade

Procurar o 5

01 Escolher nó inicial (**u**)

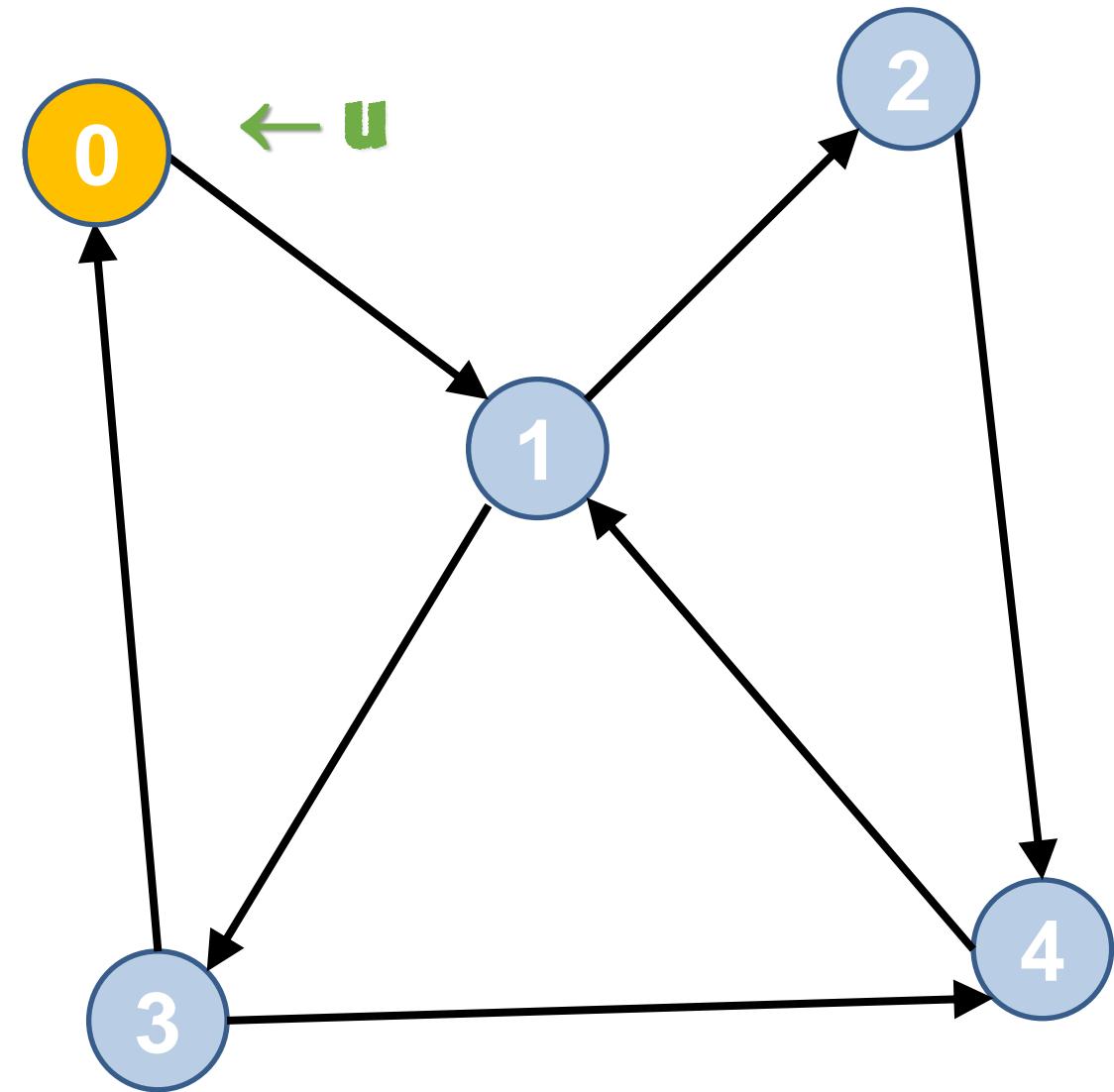
02 Visitar **u** //testar

03 Enquanto existir adjacente não visitado de **u**

04 **u** = adjacente(**u**) não visitado

05 Visitar **u** //testar

06 Fim Enquanto



Busca em Profundidade

Procurar o 5

01 Escolher nó inicial (**u**)

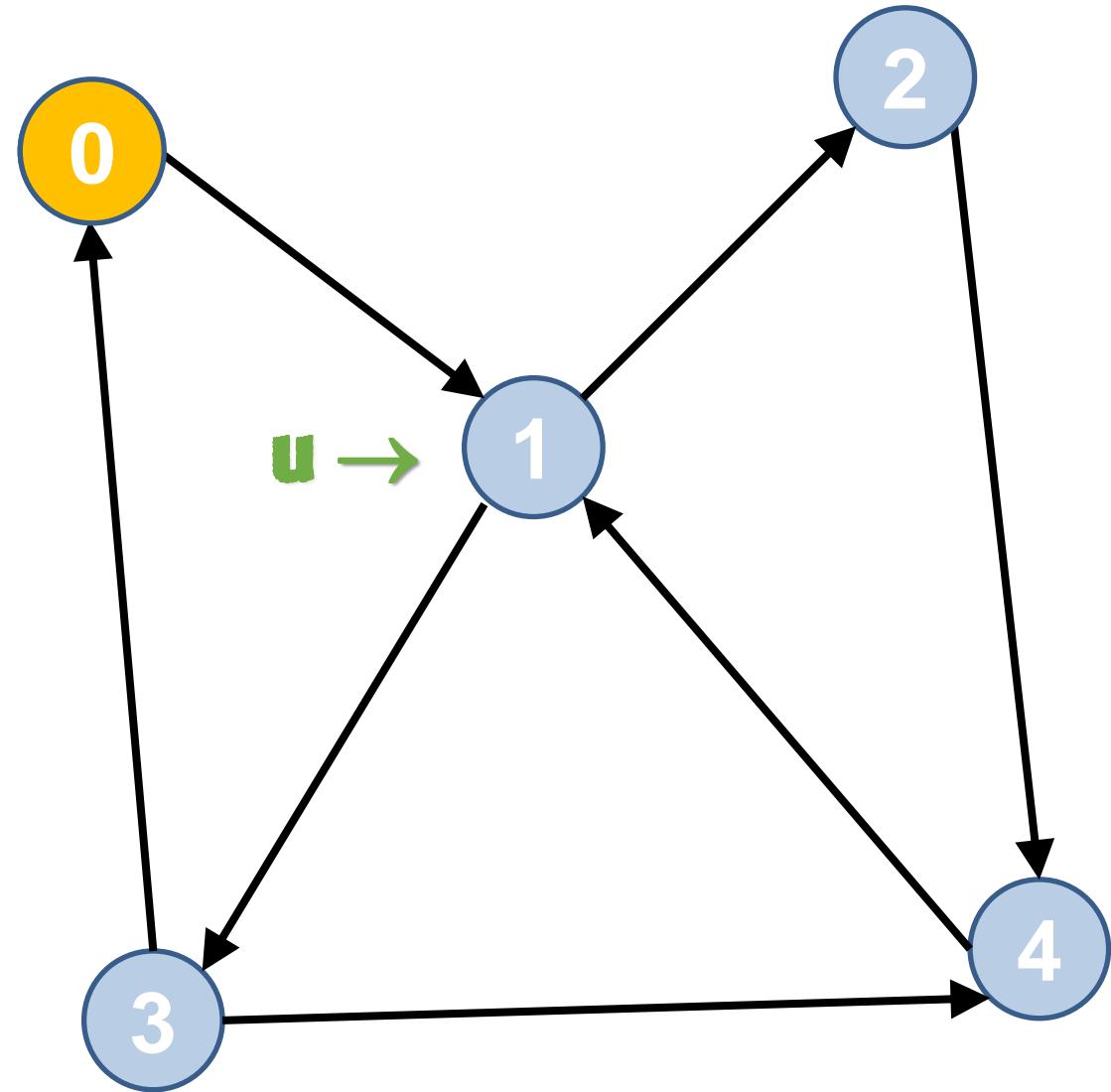
02 Visitar **u** //testar

03 Enquanto existir adjacente não visitado de **u**

04 **u** = adjacente(**u**) não visitado

05 Visitar **u** //testar

06 Fim Enquanto



Busca em Profundidade

Procurar o 5

01 Escolher nó inicial (**u**)

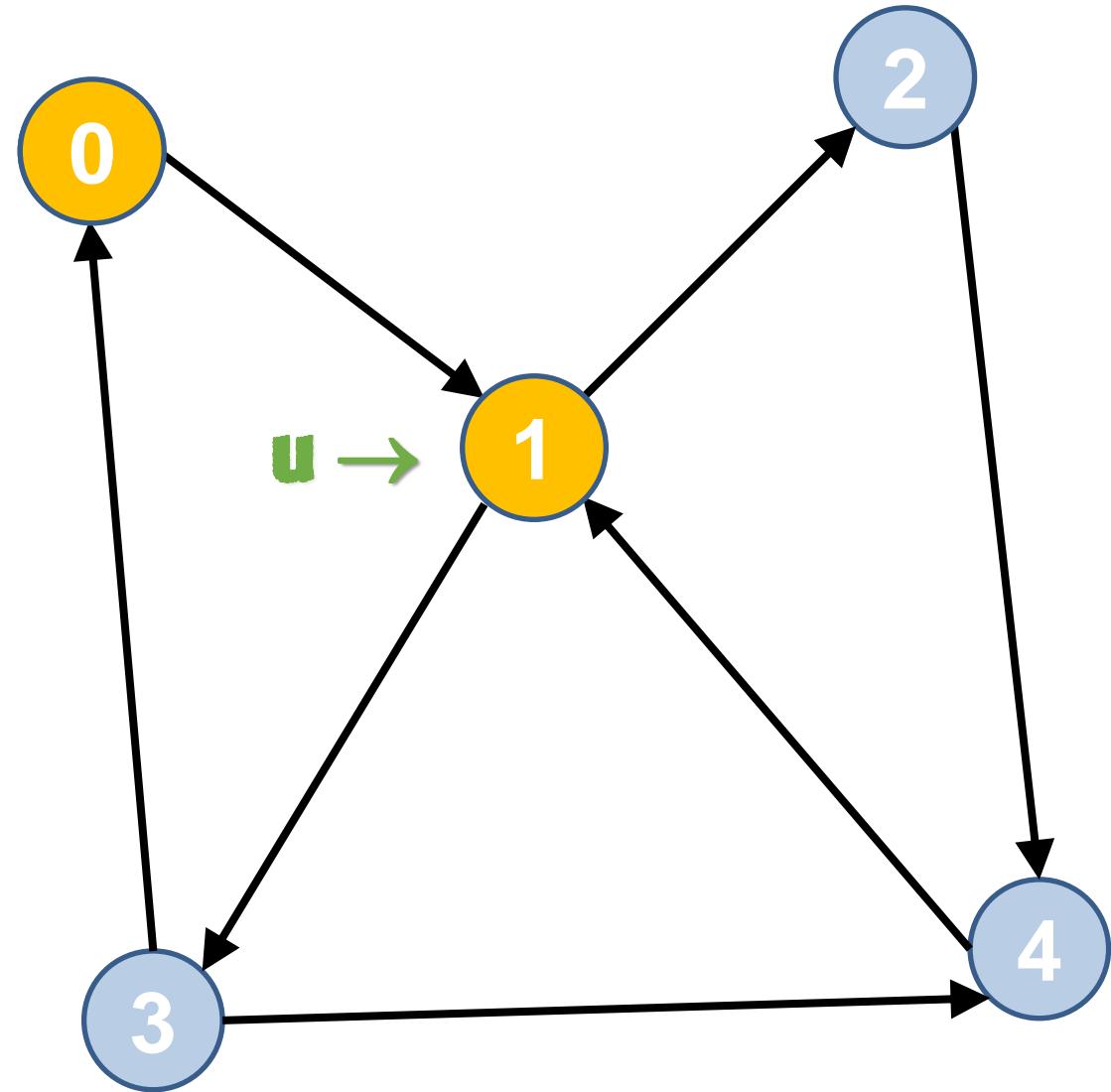
02 Visitar **u** //testar

03 Enquanto existir adjacente não visitado de **u**

04 **u** = adjacente(**u**) não visitado

05 Visitar **u** //testar

06 Fim Enquanto



Busca em Profundidade

Procurar o 5

01 Escolher nó inicial (**u**)

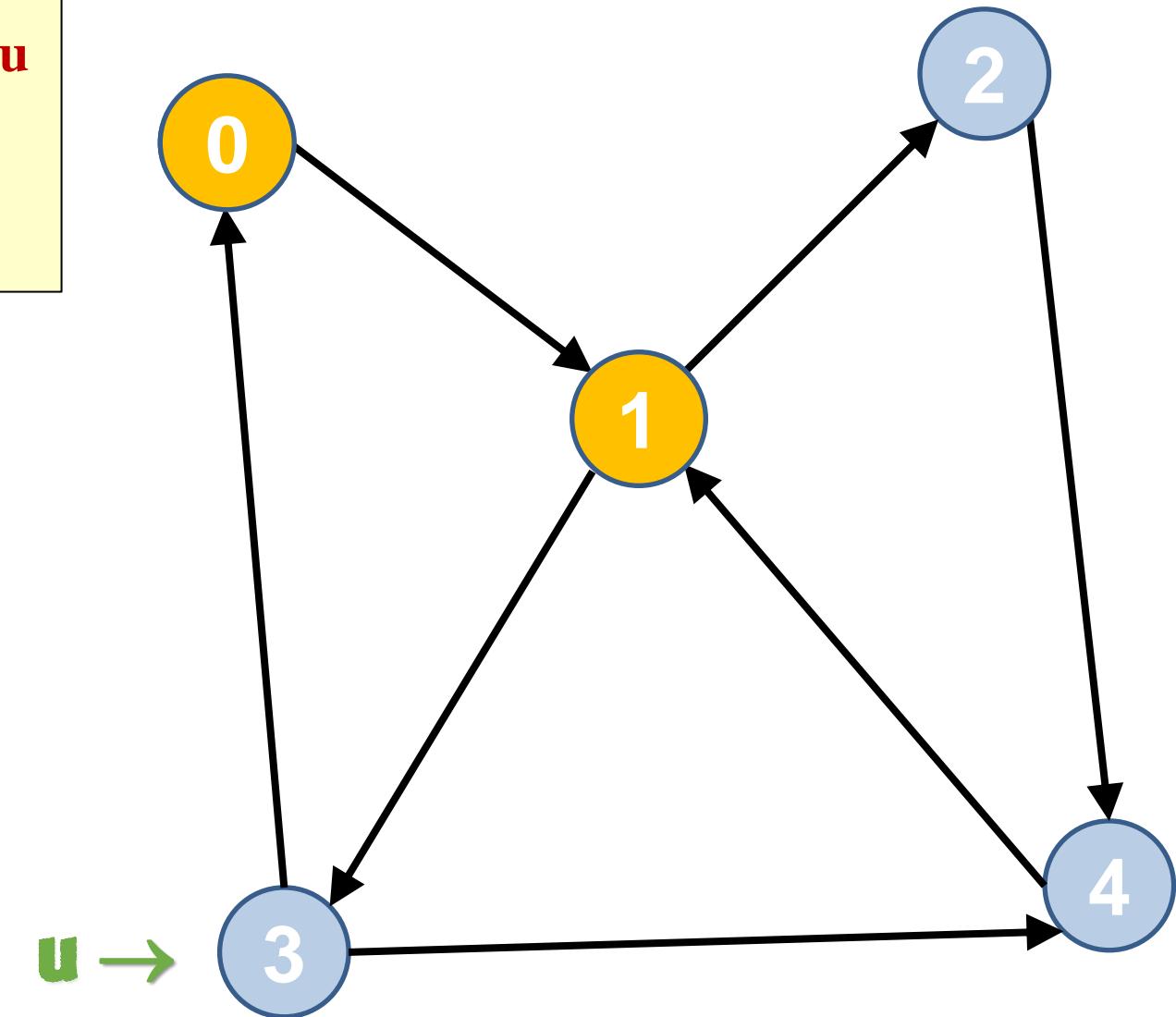
02 Visitar **u** //testar

03 Enquanto existir adjacente não visitado de **u**

04 **u** = adjacente(**u**) não visitado

05 Visitar **u** //testar

06 Fim Enquanto



Busca em Profundidade

Procurar o 5

01 Escolher nó inicial (**u**)

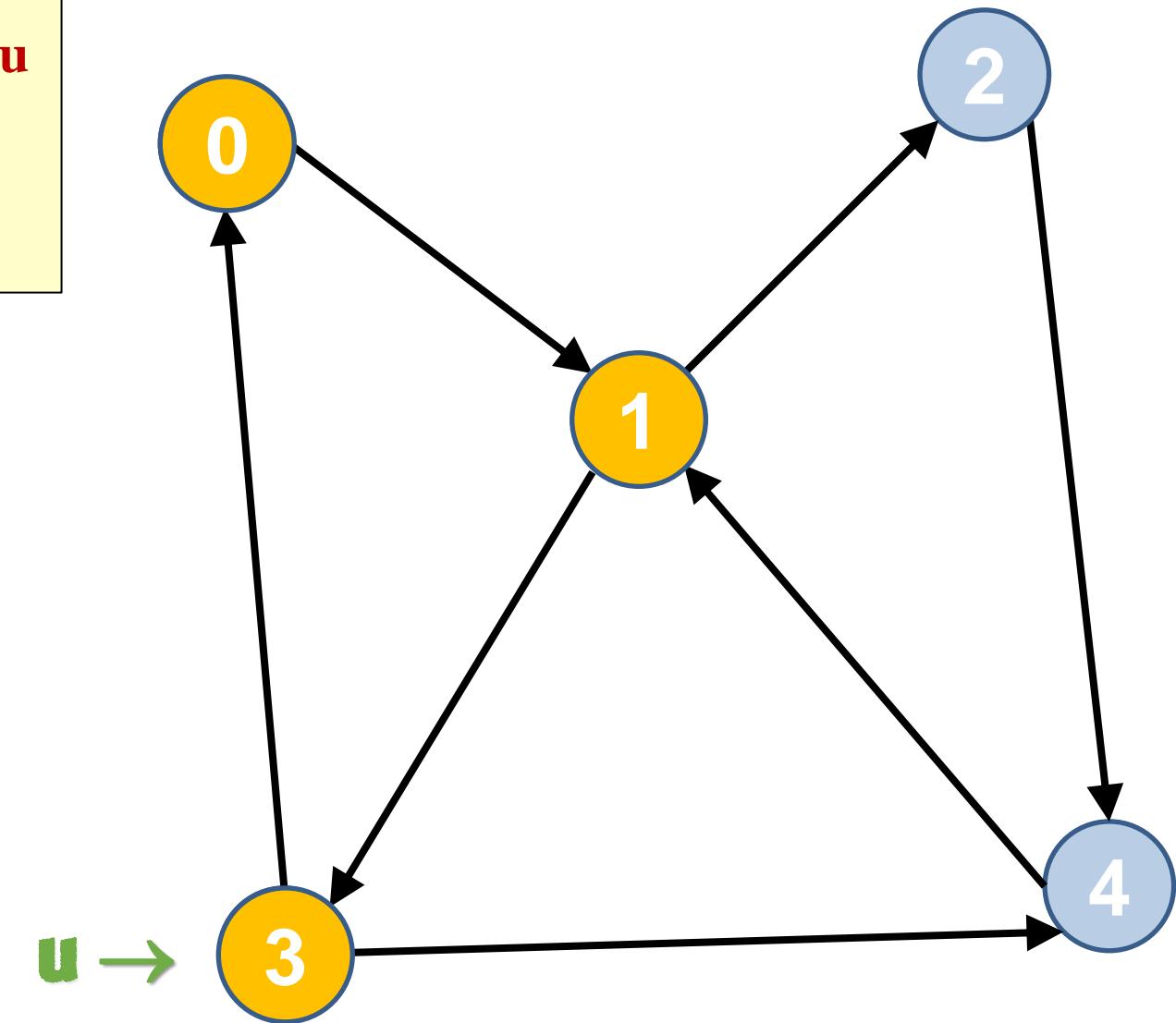
02 Visitar **u** //testar

03 Enquanto existir adjacente não visitado de **u**

04 **u** = adjacente(**u**) não visitado

05 Visitar **u** //testar

06 Fim Enquanto



Busca em Profundidade

Procurar o 5

01 Escolher nó inicial (**u**)

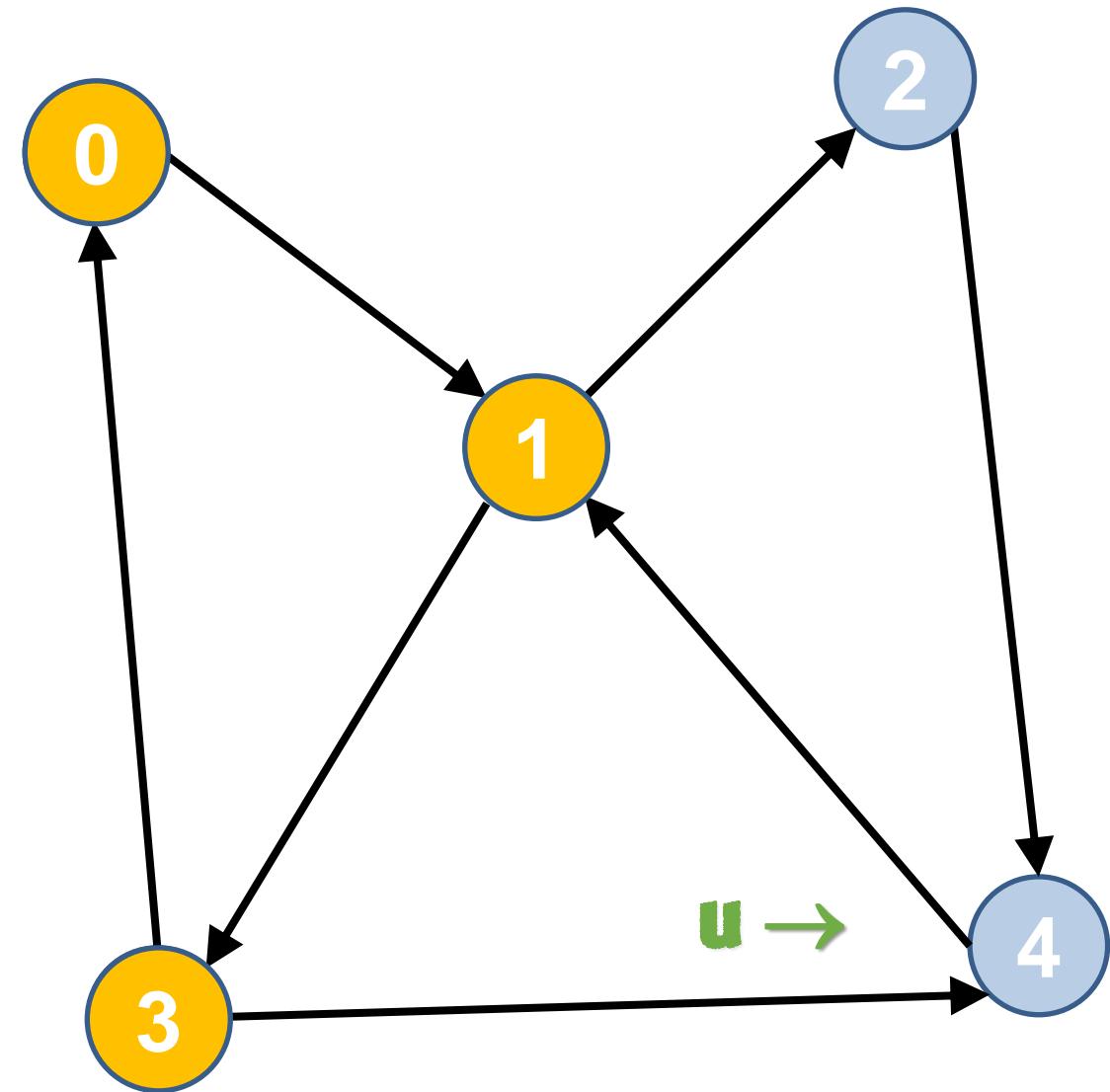
02 Visitar **u** //testar

03 Enquanto existir adjacente não visitado de **u**

04 **u** = adjacente(**u**) não visitado

05 Visitar **u** //testar

06 Fim Enquanto



Busca em Profundidade

Procurar o 5

01 Escolher nó inicial (**u**)

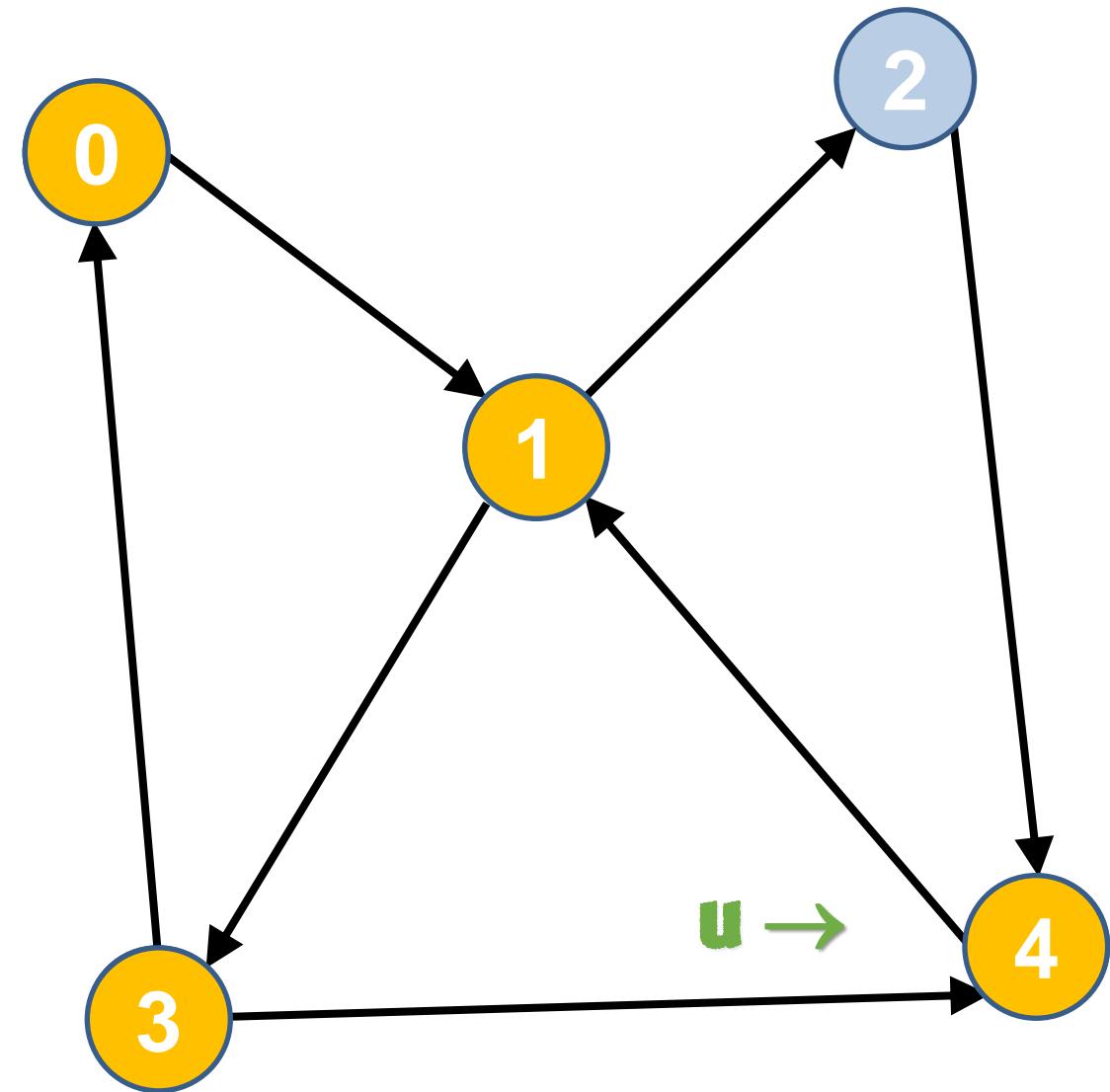
02 Visitar **u** //testar

03 Enquanto existir adjacente não visitado de **u**

04 **u** = adjacente(**u**) não visitado

05 Visitar **u** //testar

06 Fim Enquanto



Busca em Profundidade

Procurar o 5

01 Escolher nó inicial (**u**)

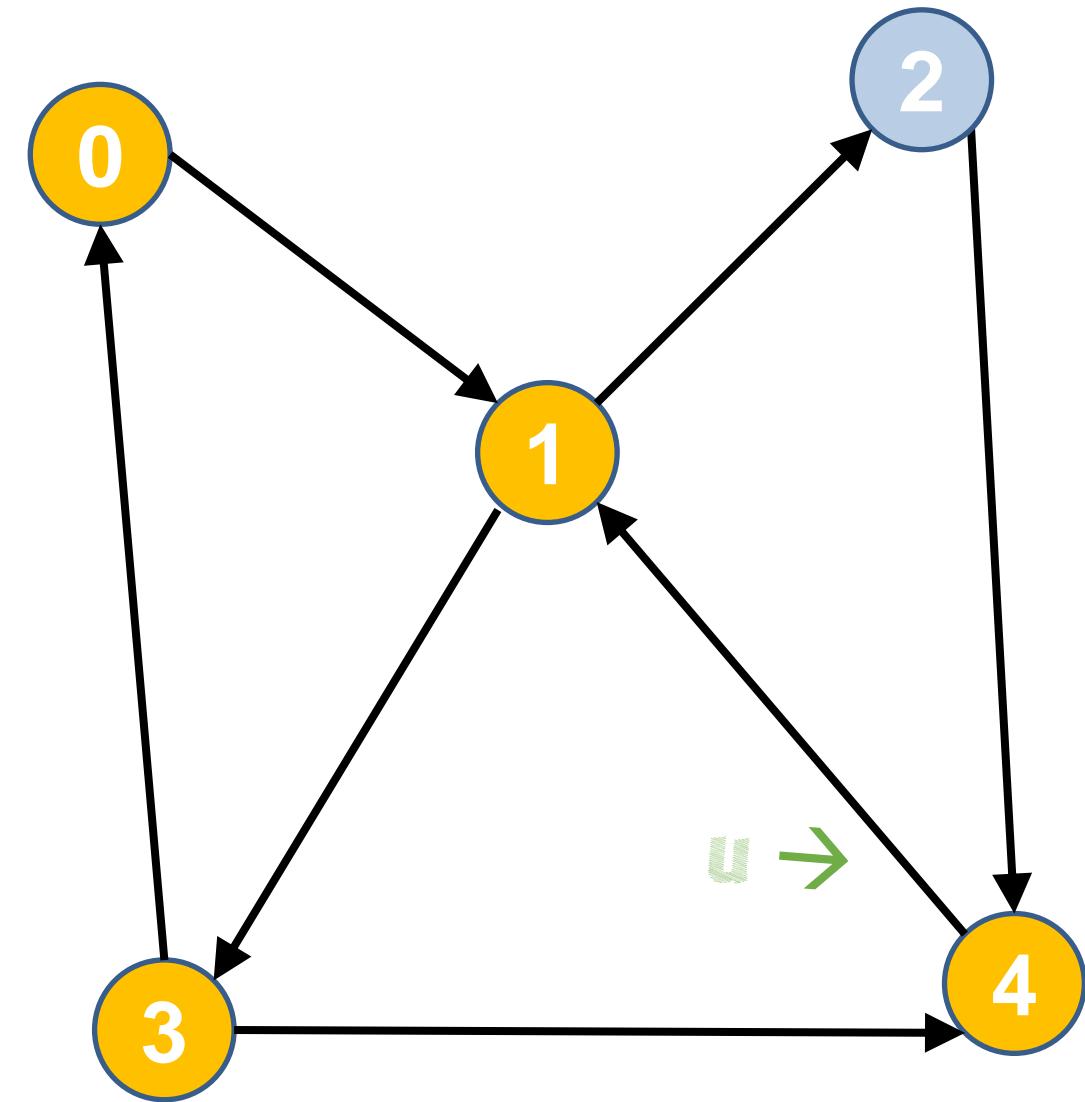
02 Visitar **u** //testar

03 Enquanto existir adjacente não visitado de **u**

04 **u** = adjacente(**u**) não visitado

05 Visitar **u** //testar

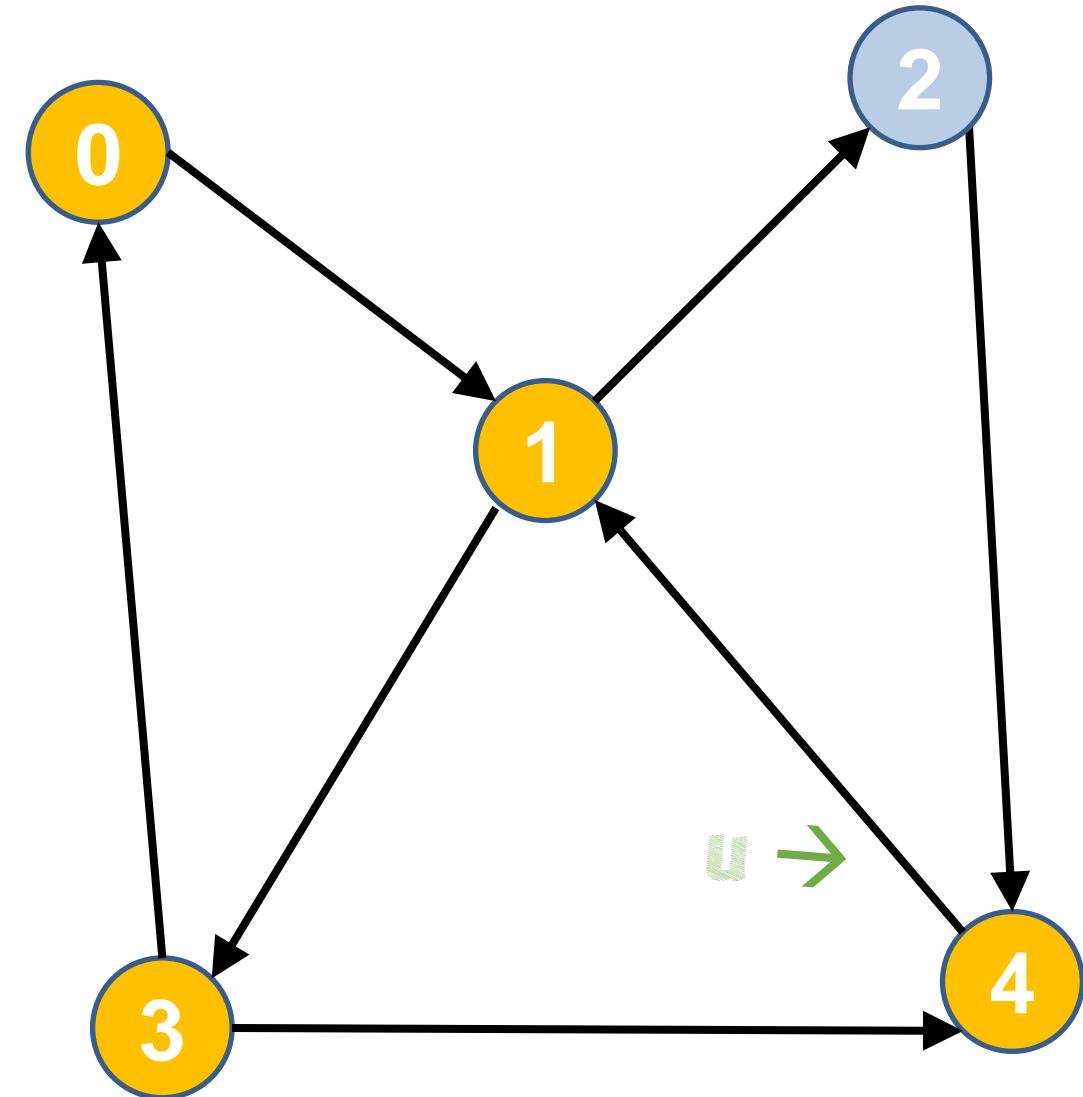
06 Fim Enquanto



Busca em Profundidade

Procurar o 5

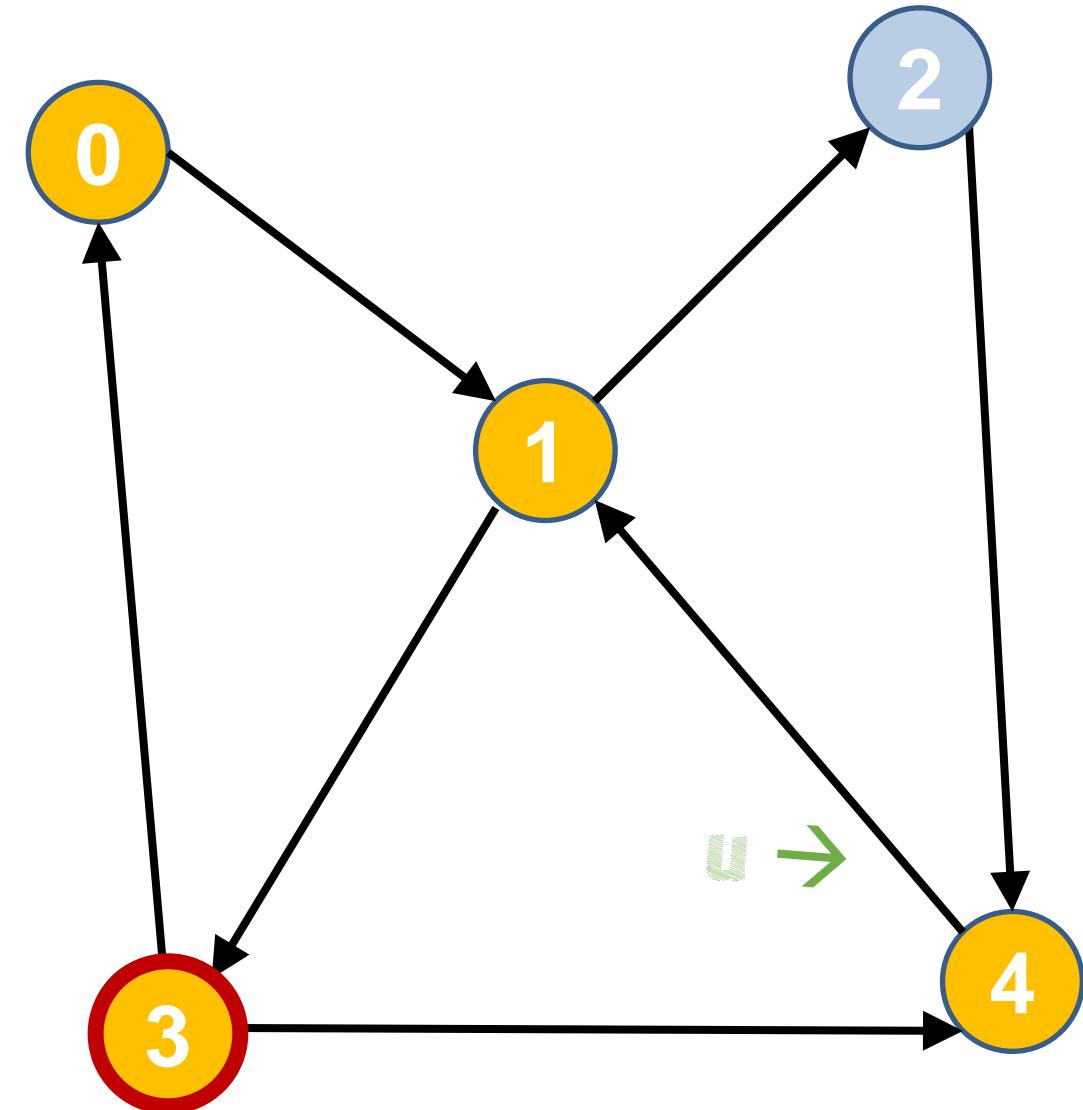
- 01 Escolher nó inicial (**u**)
- 02 Visitar **u** //testar
- 03 Enquanto existir adjacente não visitado de **u**
- 04 **u** = adjacente(**u**) não visitado
- 05 Visitar **u** //testar
- 06 Fim Enquanto
- 07 Se (**Backtracking (Pai(u))**)
- 08 GOTO Linha 03
- 09 Senão e ainda houver vértice não visitado
- 10 Definir novo **u**



Busca em Profundidade

Procurar o 5

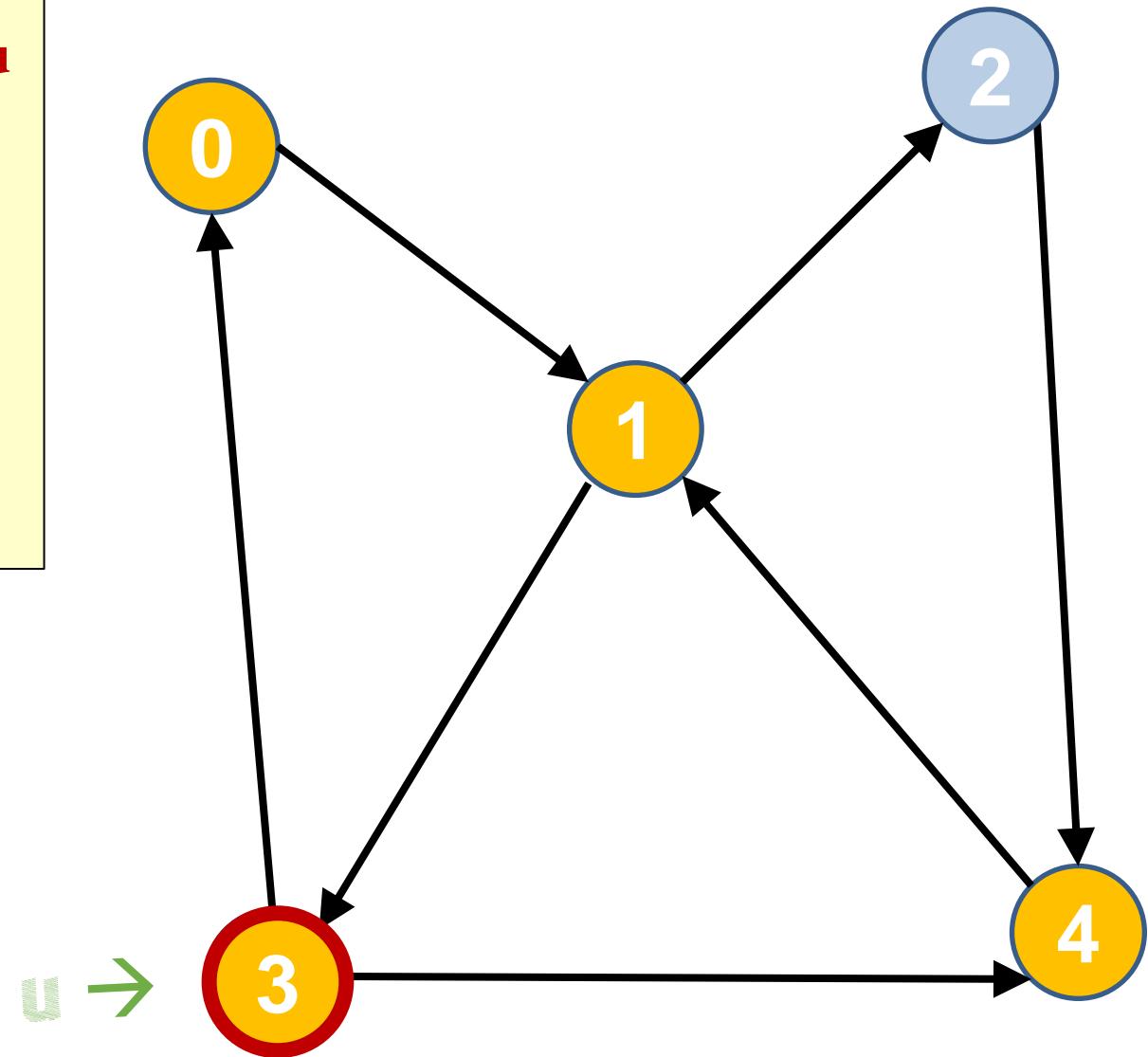
- 01 Escolher nó inicial (**u**)
- 02 Visitar **u** //testar
- 03 Enquanto existir adjacente não visitado de **u**
- 04 **u** = adjacente(**u**) não visitado
- 05 Visitar **u** //testar
- 06 Fim Enquanto
- 07 Se (**Backtracking (Pai(u))**)
- 08 GOTO Linha 03
- 09 Senão e ainda houver vértice não visitado
- 10 Definir novo **u**



Busca em Profundidade

Procurar o 5

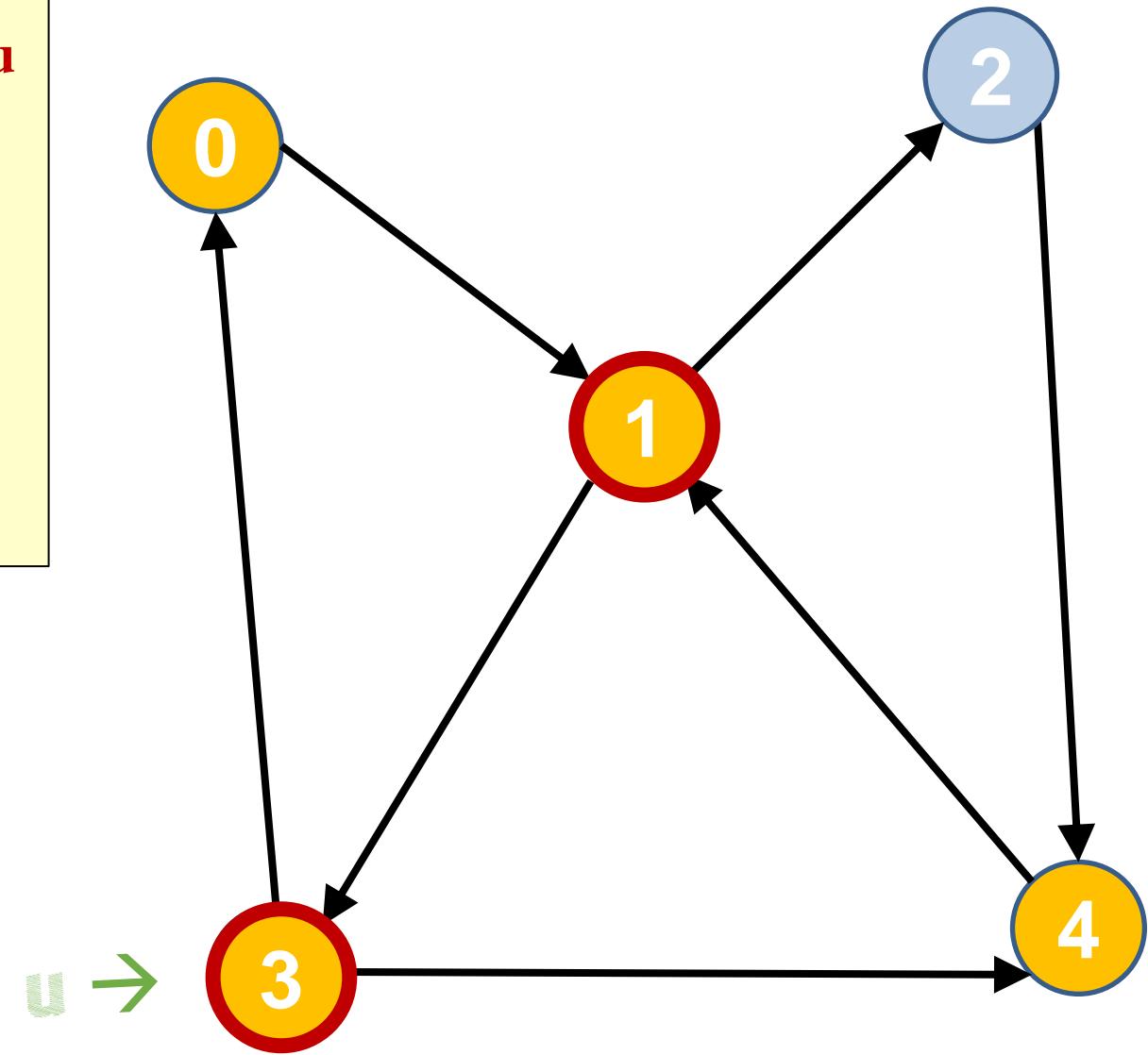
- 01 Escolher nó inicial (**u**)
- 02 Visitar **u** //testar
- 03 Enquanto existir adjacente não visitado de **u**
- 04 **u** = adjacente(**u**) não visitado
- 05 Visitar **u** //testar
- 06 Fim Enquanto
- 07 Se (**Backtracking (Pai(u))**)
- 08 **GOTO Linha 03**
- 09 Senão e ainda houver vértice não visitado
- 10 Definir novo **u**



Busca em Profundidade

Procurar o 5

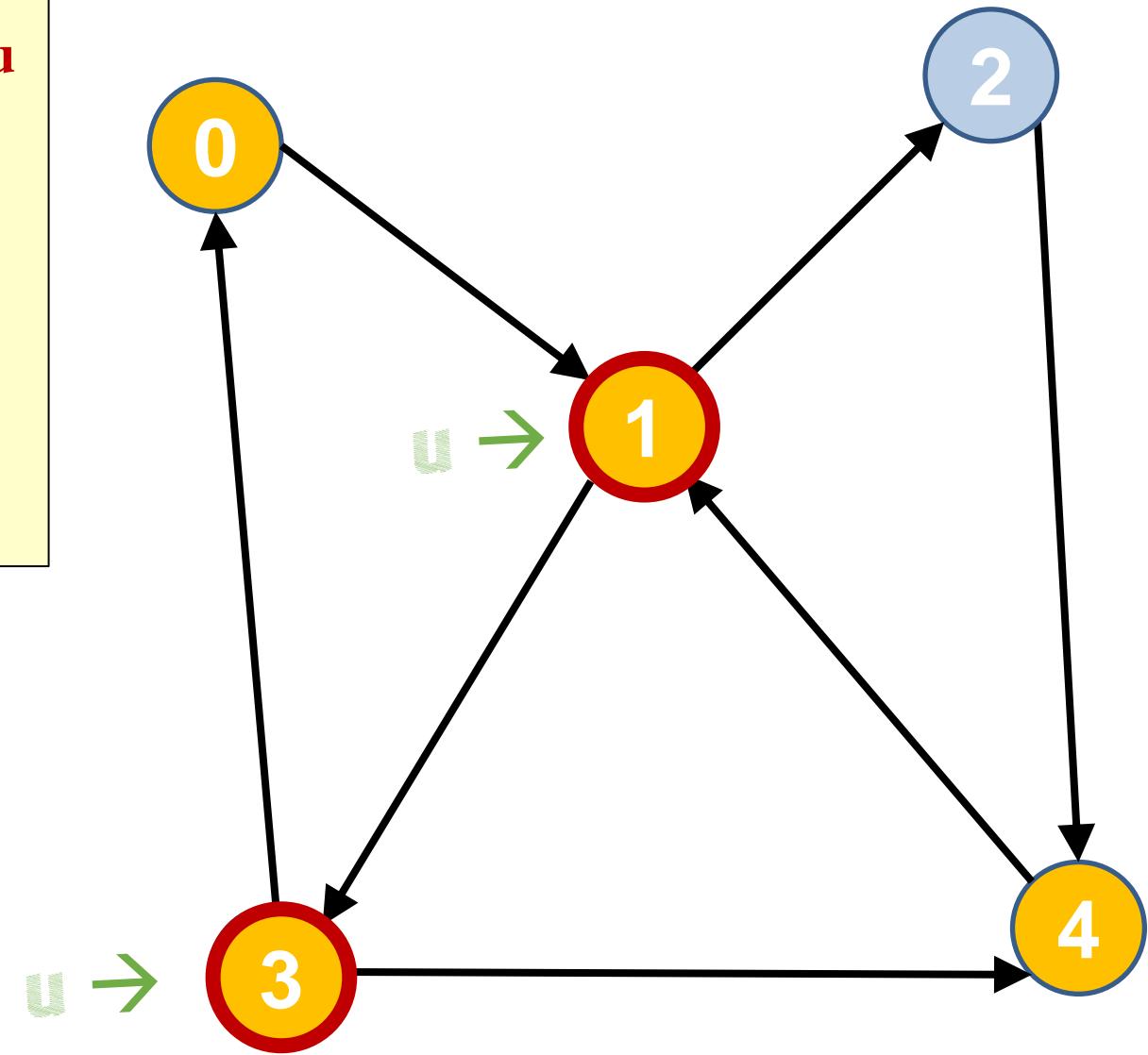
- 01 Escolher nó inicial (**u**)
- 02 Visitar **u** //testar
- 03 Enquanto existir adjacente não visitado de **u**
- 04 **u** = adjacente(**u**) não visitado
- 05 Visitar **u** //testar
- 06 Fim Enquanto
- 07 Se (**Backtracking (Pai(u))**)
- 08 **GOTO Linha 03**
- 09 Senão e ainda houver vértice não visitado
- 10 Definir novo **u**



Busca em Profundidade

Procurar o 5

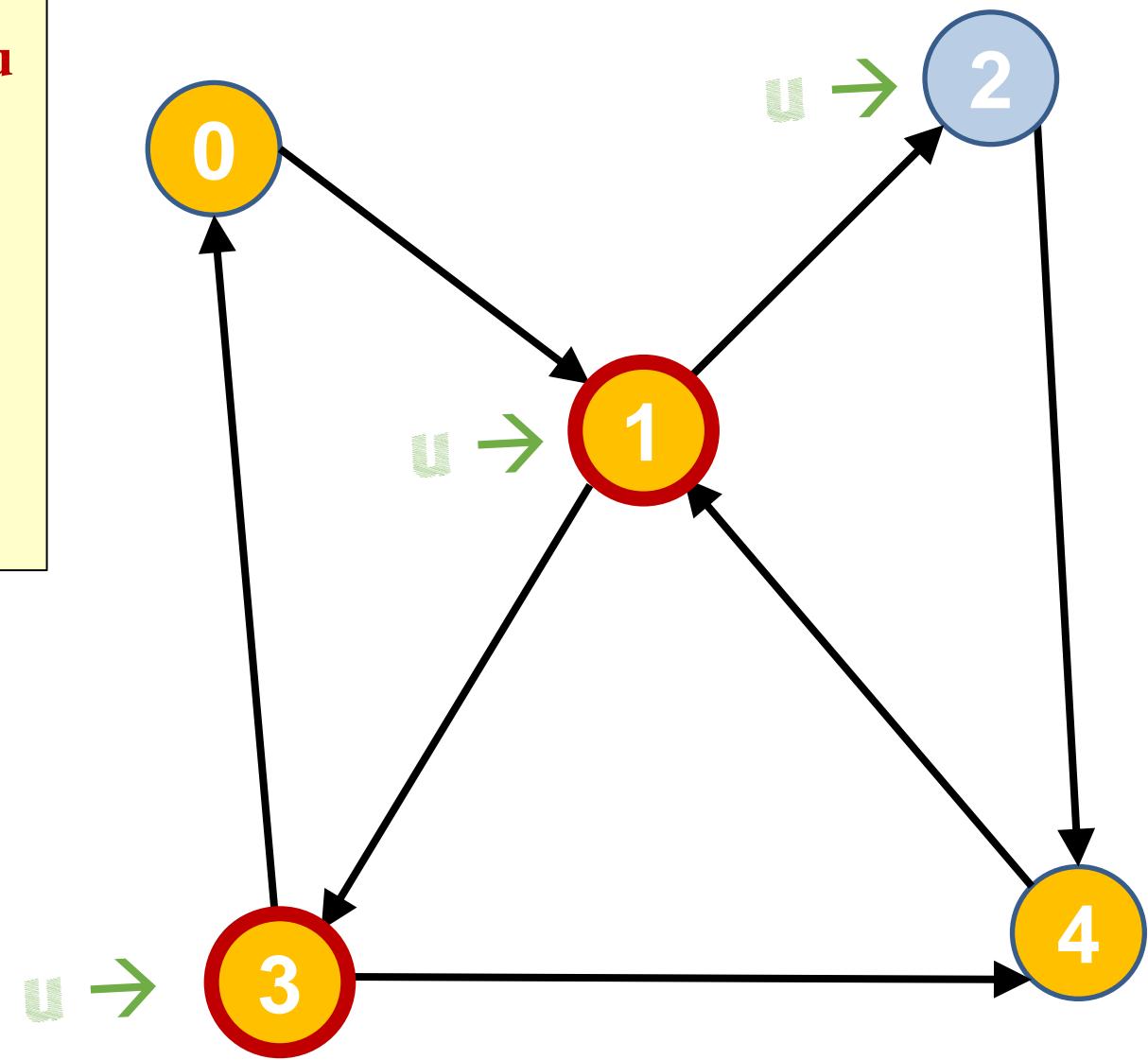
- 01 Escolher nó inicial (**u**)
- 02 Visitar **u** //testar
- 03 Enquanto existir adjacente não visitado de **u**
- 04 **u** = adjacente(**u**) não visitado
- 05 Visitar **u** //testar
- 06 Fim Enquanto
- 07 Se (**Backtracking (Pai(u))**)
- 08 **GOTO Linha 03**
- 09 Senão e ainda houver vértice não visitado
- 10 Definir novo **u**



Busca em Profundidade

Procurar o 5

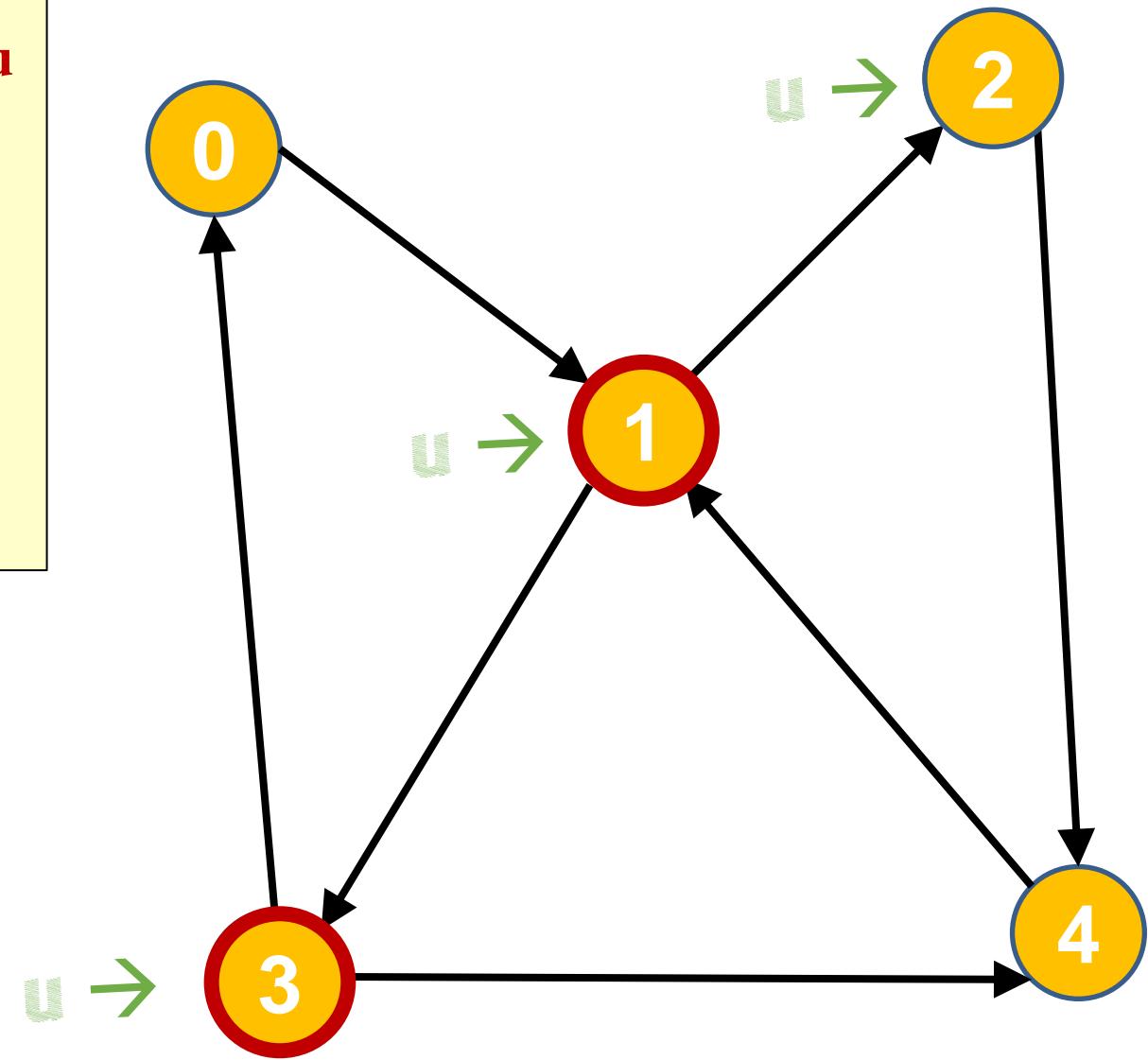
- 01 Escolher nó inicial (**u**)
- 02 Visitar **u** //testar
- 03 Enquanto existir adjacente não visitado de **u**
- 04 **u** = adjacente(**u**) não visitado
- 05 Visitar **u** //testar
- 06 Fim Enquanto
- 07 Se (**Backtracking (Pai(u))**)
- 08 **GOTO Linha 03**
- 09 Senão e ainda houver vértice não visitado
- 10 Definir novo **u**



Busca em Profundidade

Procurar o 5

- 01 Escolher nó inicial (**u**)
- 02 Visitar **u** //testar
- 03 Enquanto existir adjacente não visitado de **u**
- 04 **u** = adjacente(**u**) não visitado
- 05 Visitar **u** //testar
- 06 Fim Enquanto
- 07 Se (**Backtracking (Pai(u))**)
- 08 **GOTO Linha 03**
- 09 Senão e ainda houver vértice não visitado
- 10 Definir novo **u**

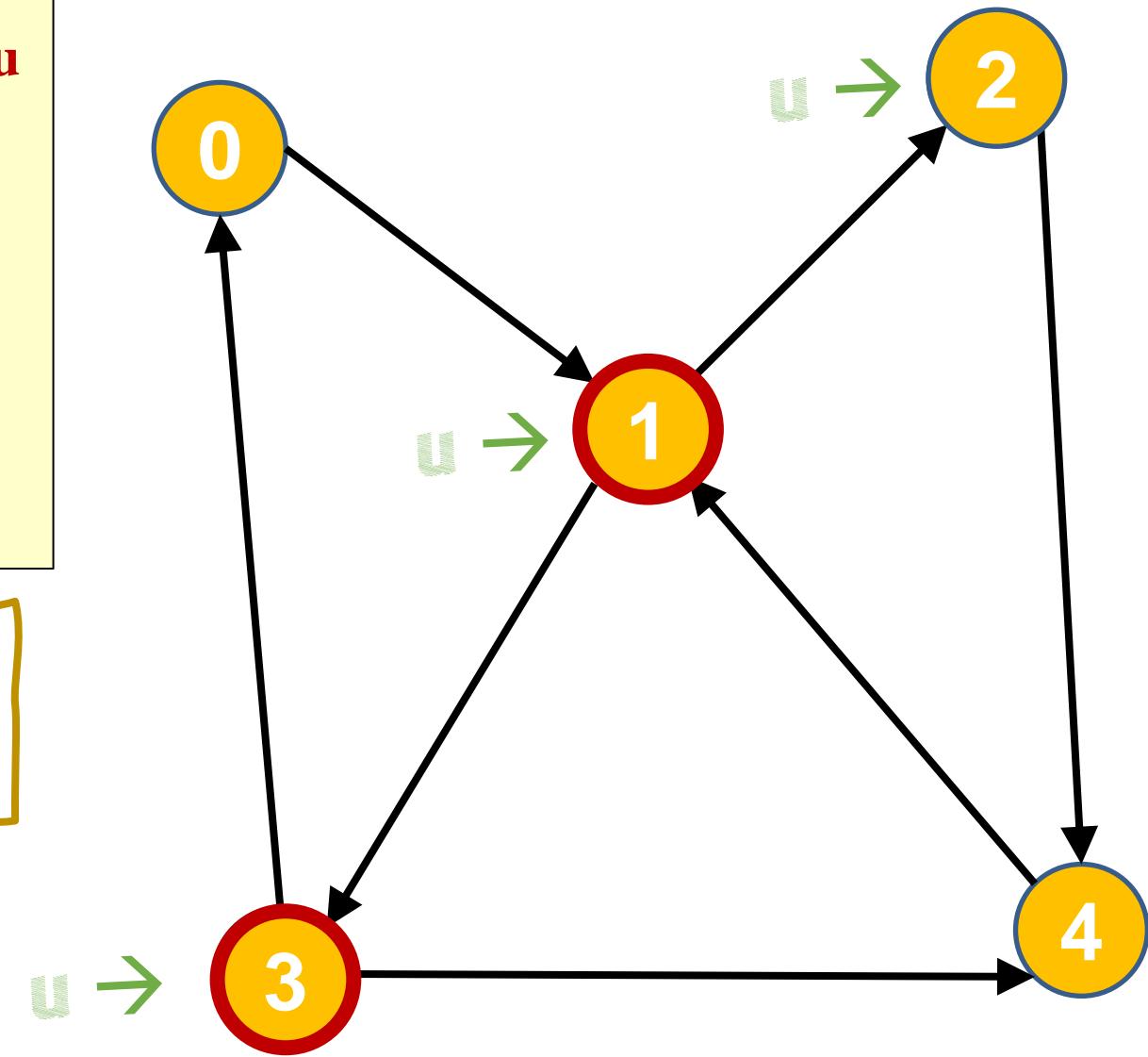


Busca em Profundidade

Procurar o 5

- 01 Escolher nó inicial (**u**)
- 02 Visitar **u** //testar
- 03 Enquanto existir adjacente não visitado de **u**
 - 04 **u** = adjacente(**u**) não visitado
 - 05 Visitar **u** //testar
- 06 Fim Enquanto
- 07 Se (**Backtracking (Pai(u))**)
- 08 **GOTO Linha 03**
- 09 Senão e ainda houver vértice não visitado
- 10 Definir novo **u**

0 > 1 > 3 > 4 > 2



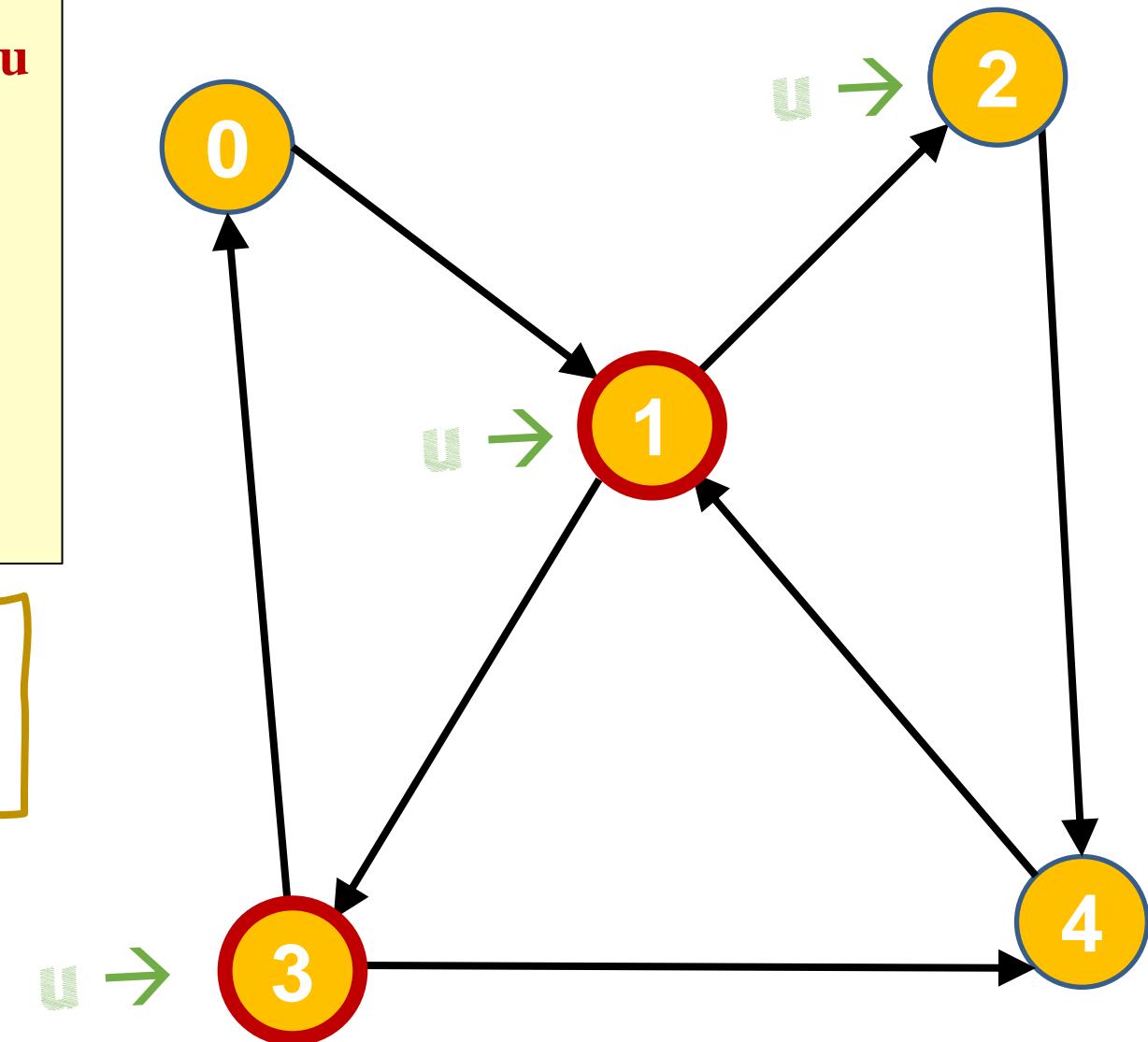
Busca em Profundidade

Procurar o 5

- 01 Escolher nó inicial (**u**)
- 02 Visitar **u** //testar
- 03 Enquanto existir adjacente não visitado de **u**
 - 04 **u** = adjacente(**u**) não visitado
 - 05 Visitar **u** //testar
- 06 Fim Enquanto
- 07 Se (**Backtracking (Pai(u))**)
- 08 GOTO Linha 03
- 09 Senão e ainda houver vértice não visitado
- 10 Definir novo **u**

0 > 1 > 3 > 4 > 2

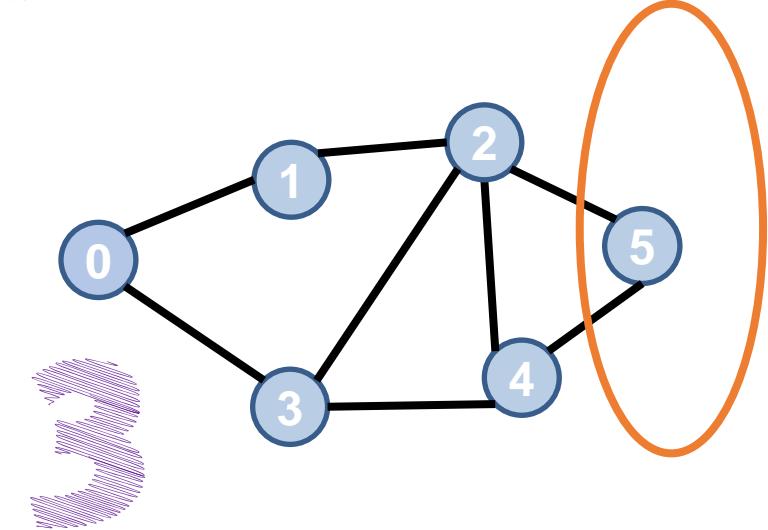
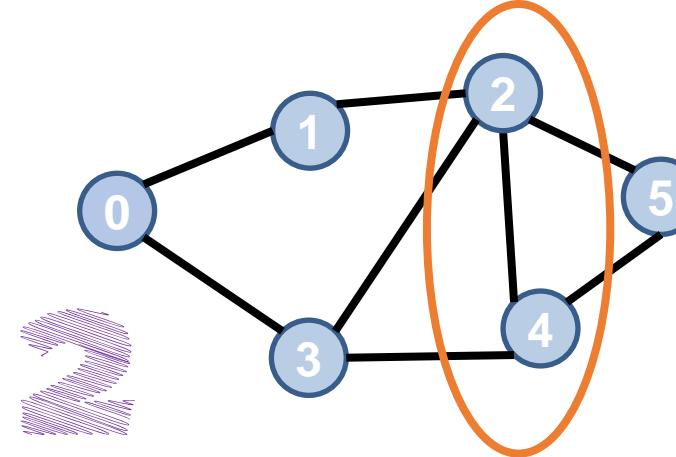
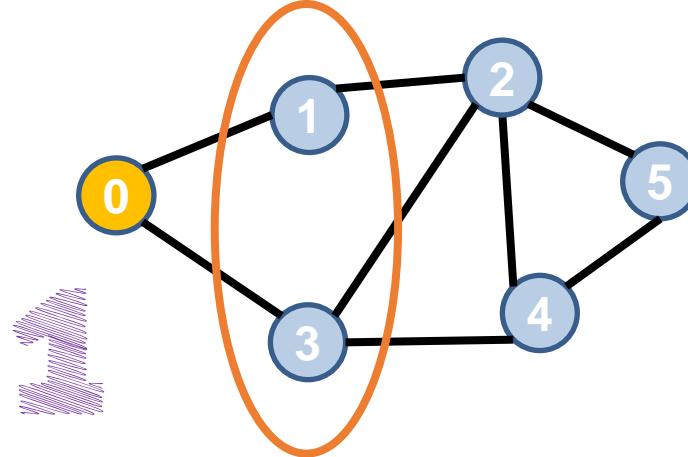
0 > 1 > 2 > 4 > 3



**Busca em
Largura**

Busca em Largura - Breadth-first search (BFS)

- Dado um vértice inicial, visita todos os seus nós adjacentes
 - Para cada vértice vizinho, são visitados todos os seus vizinhos ainda inexplorados
 - Esse processo continua até que
 - o alvo da busca seja encontrado
 - não existam mais vértices a serem visitados
- Pode ser visto como uma busca em camadas



Busca em Largura

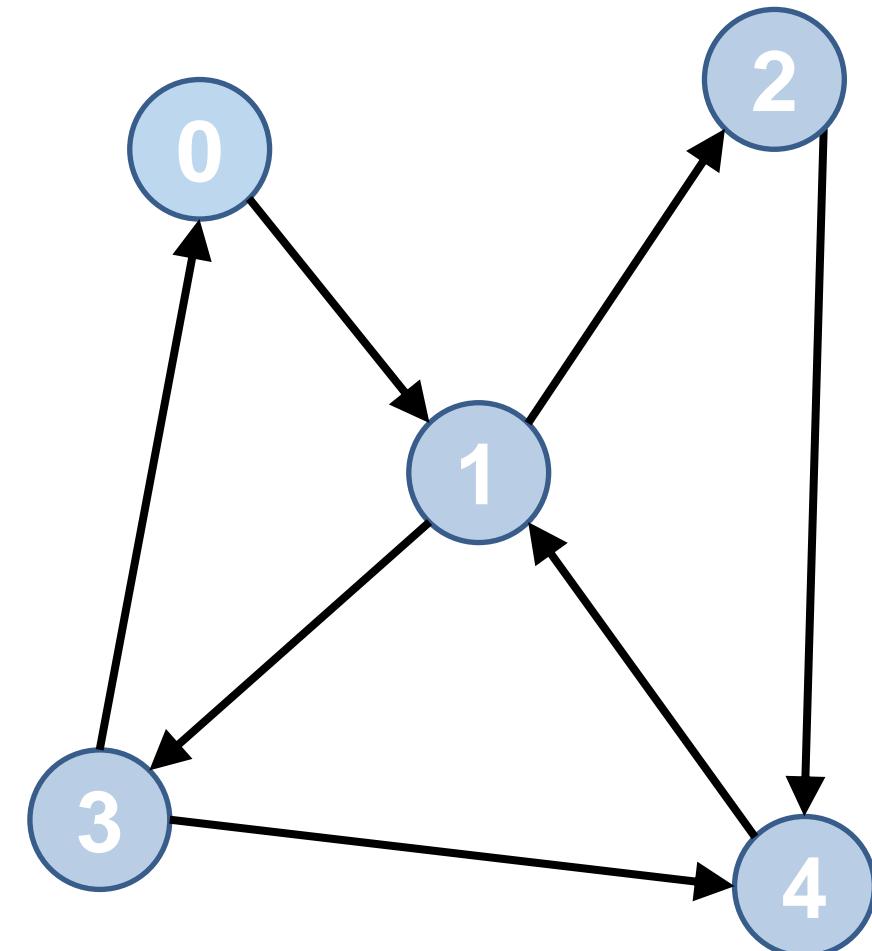
- Dado um nó inicial **H**
 - A busca em largura determina a **distância**, em quantidade de arestas, para cada vértice que pode ser alcançado a partir de **H**
 - **H** até **P** = **x** arestas
 - **H** até **V** = **y** arestas
 -
 - Vértices que se encontram a **d + 1** arestas de **H**, só serão visitados, após todos os vértices a uma distância de **d** arestas terem sido visitados

Busca em Largura

Procurar o 5

... estudar e praticar ... the best way for everything

01
02
03
04
05
06
07
08
09
10

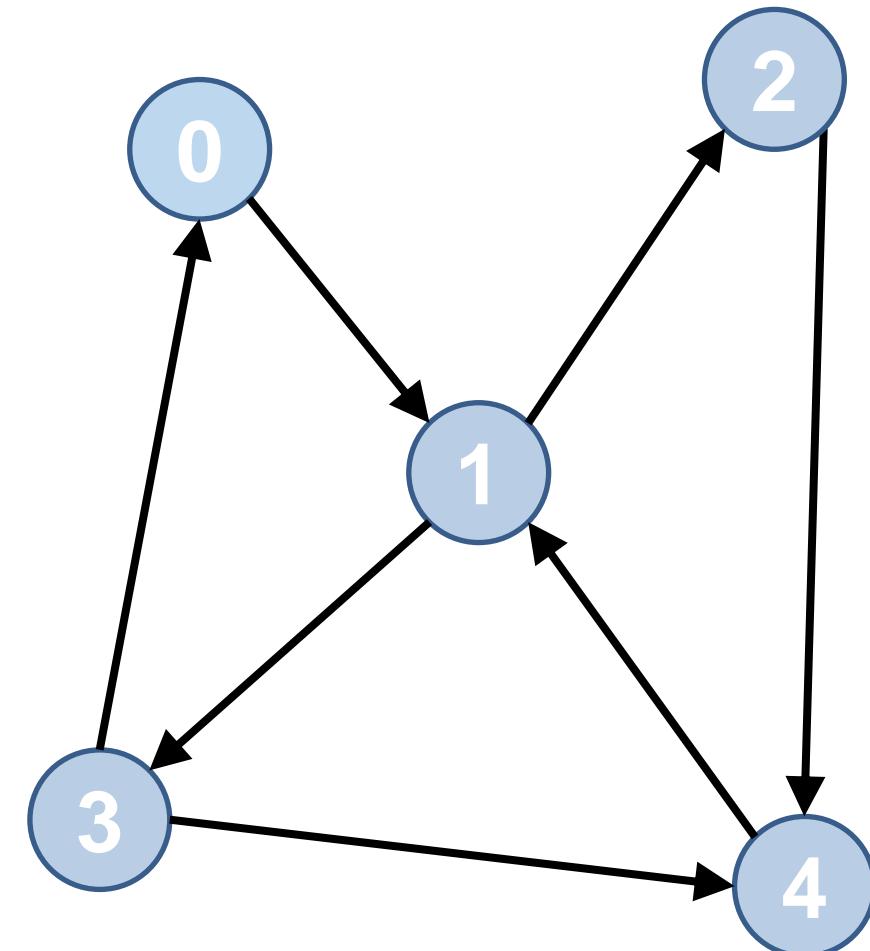


FILA

Busca em Largura

Procurar o 5

01 | Escolher nó inicial (**u**)
02
03
04
05
06
07
08
09
10

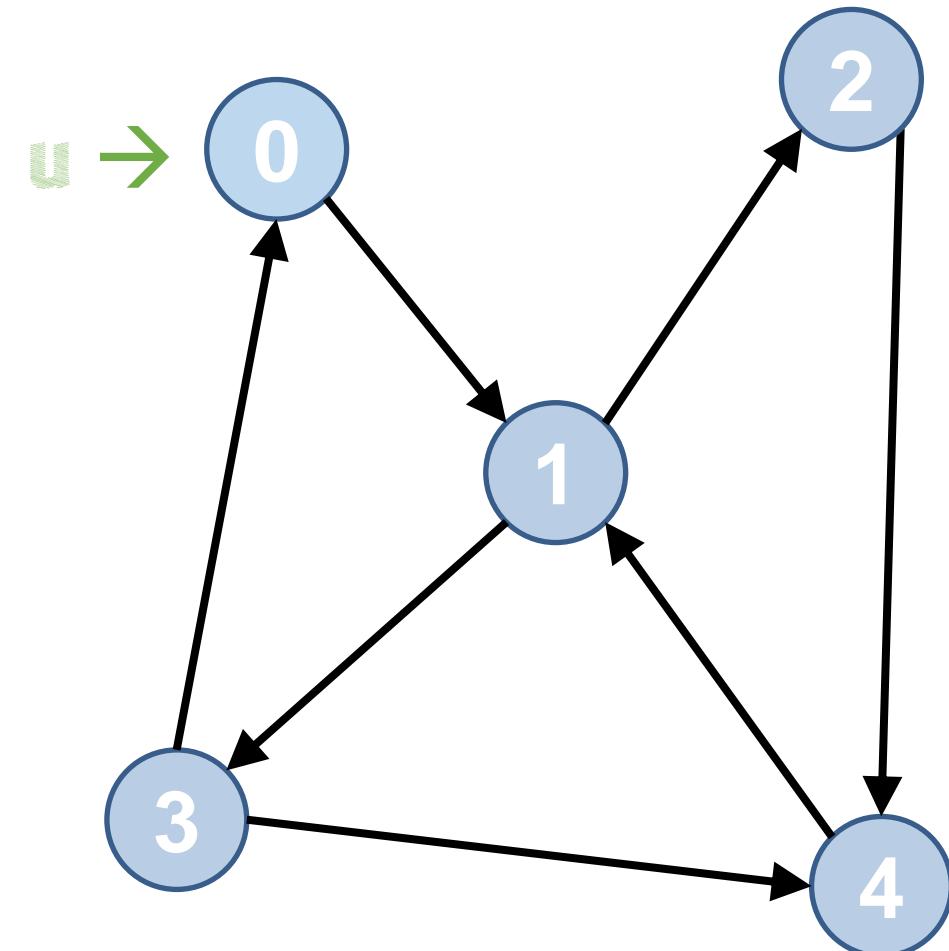


FILA

Busca em Largura

Procurar o 5

01 | Escolher nó inicial (**u**)
02
03
04
05
06
07
08
09
10

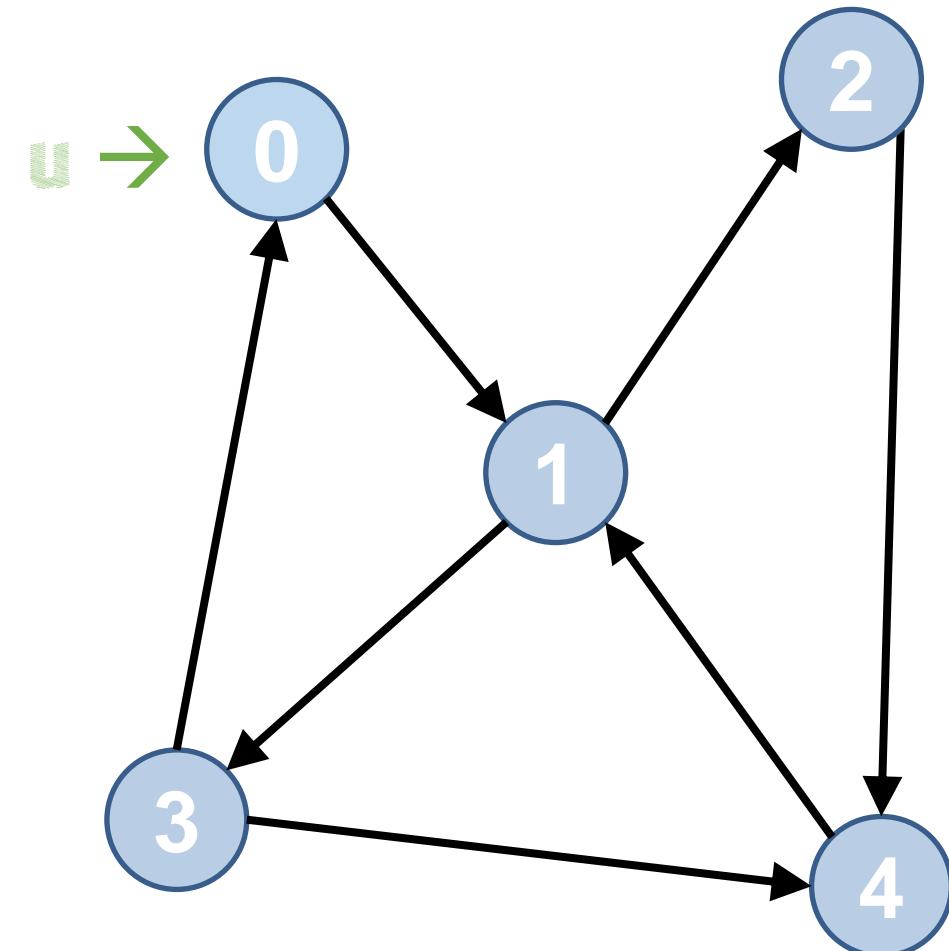


FILA

Busca em Largura

Procurar o 5

- 01 Escolher nó inicial (**u**)
- 02 Visitar **u** //testar
- 03
- 04
- 05
- 06
- 07
- 08
- 09
- 10

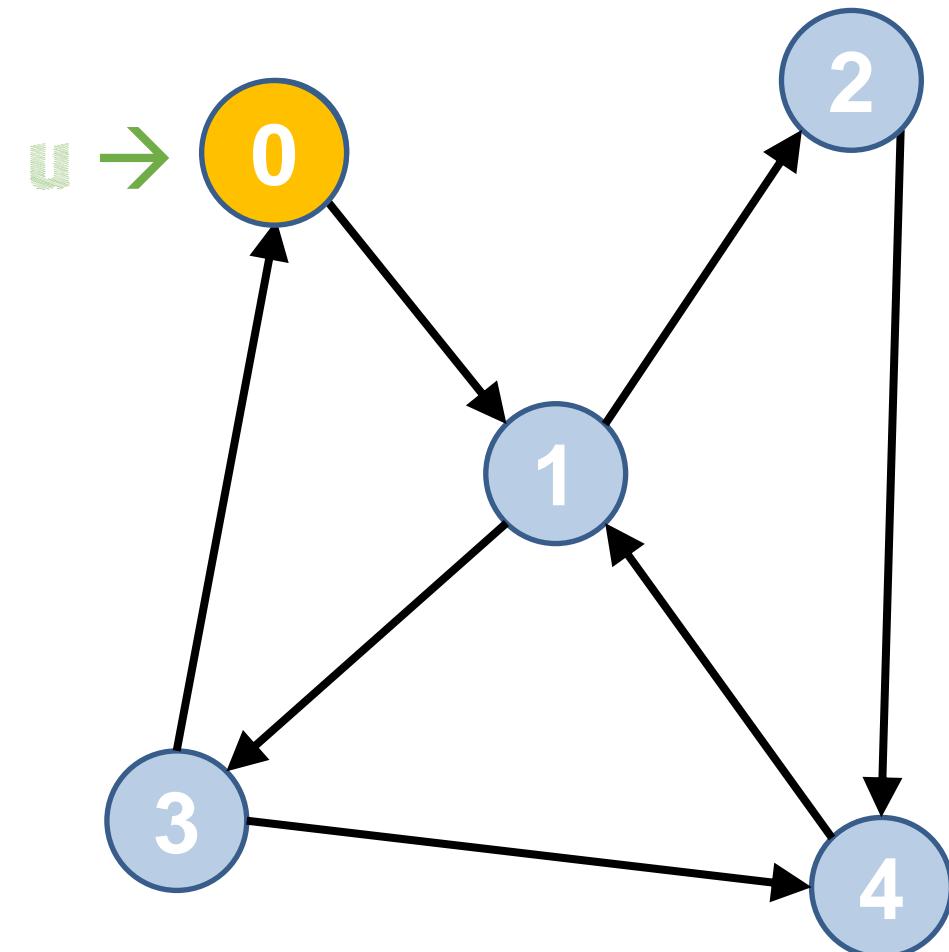


FILA

Busca em Largura

Procurar o 5

- 01 Escolher nó inicial (**u**)
- 02 Visitar **u** //testar
- 03
- 04
- 05
- 06
- 07
- 08
- 09
- 10

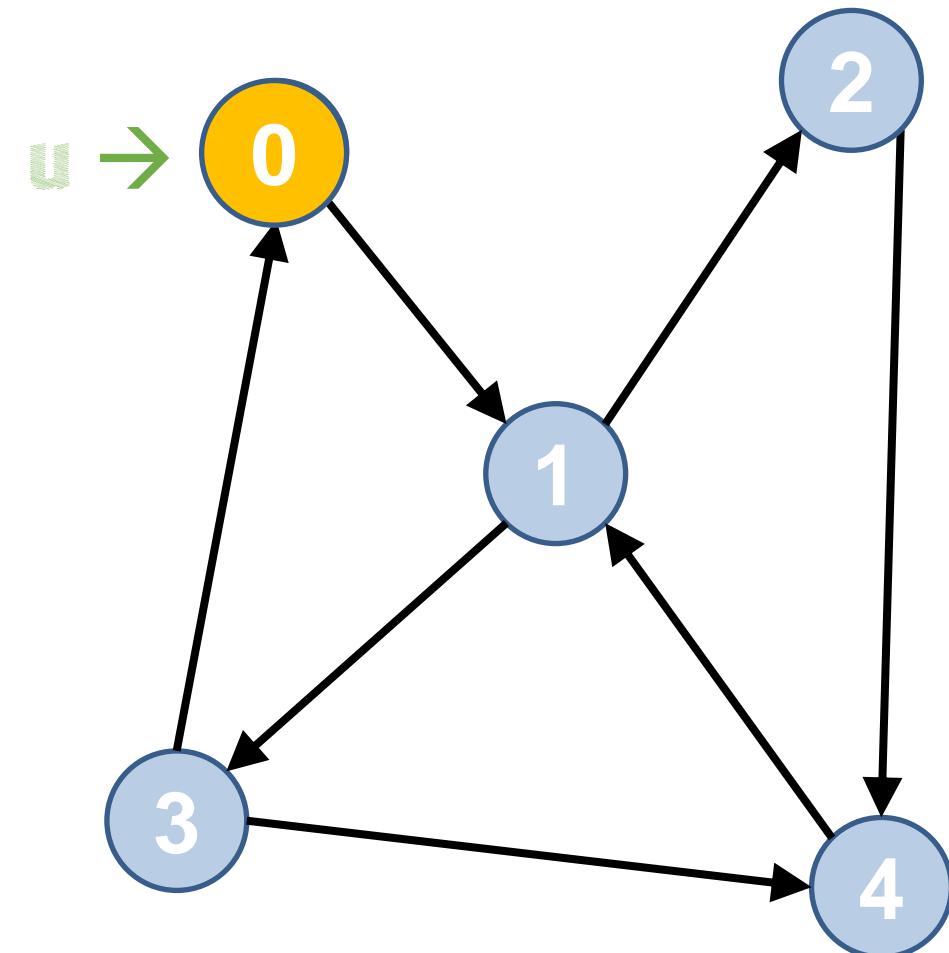


FILA

Busca em Largura

Procurar o 5

- 01 Escolher nó inicial (**u**)
- 02 Visitar **u** //testar
- 03 Adicione **u** na Fila
- 04
- 05
- 06
- 07
- 08
- 09
- 10

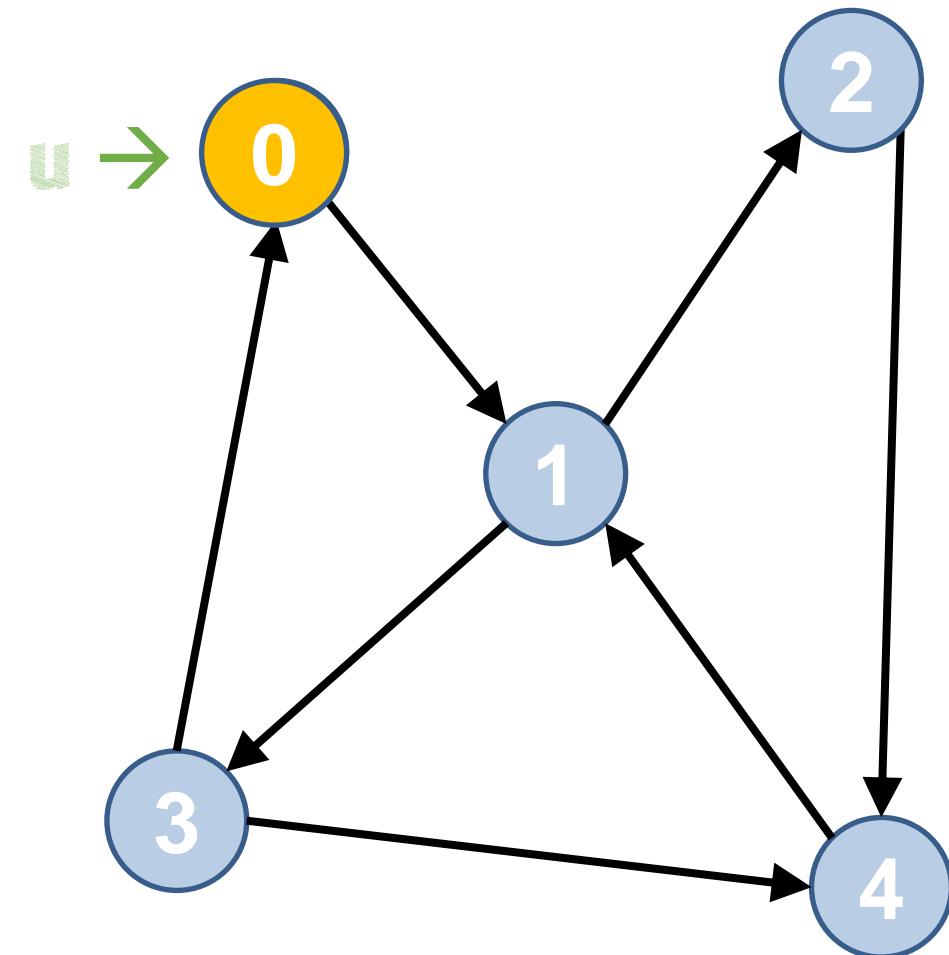


FILA

Busca em Largura

Procurar o 5

- 01 Escolher nó inicial (**u**)
- 02 Visitar **u** //testar
- 03 Adicione **u** na Fila
- 04
- 05
- 06
- 07
- 08
- 09
- 10



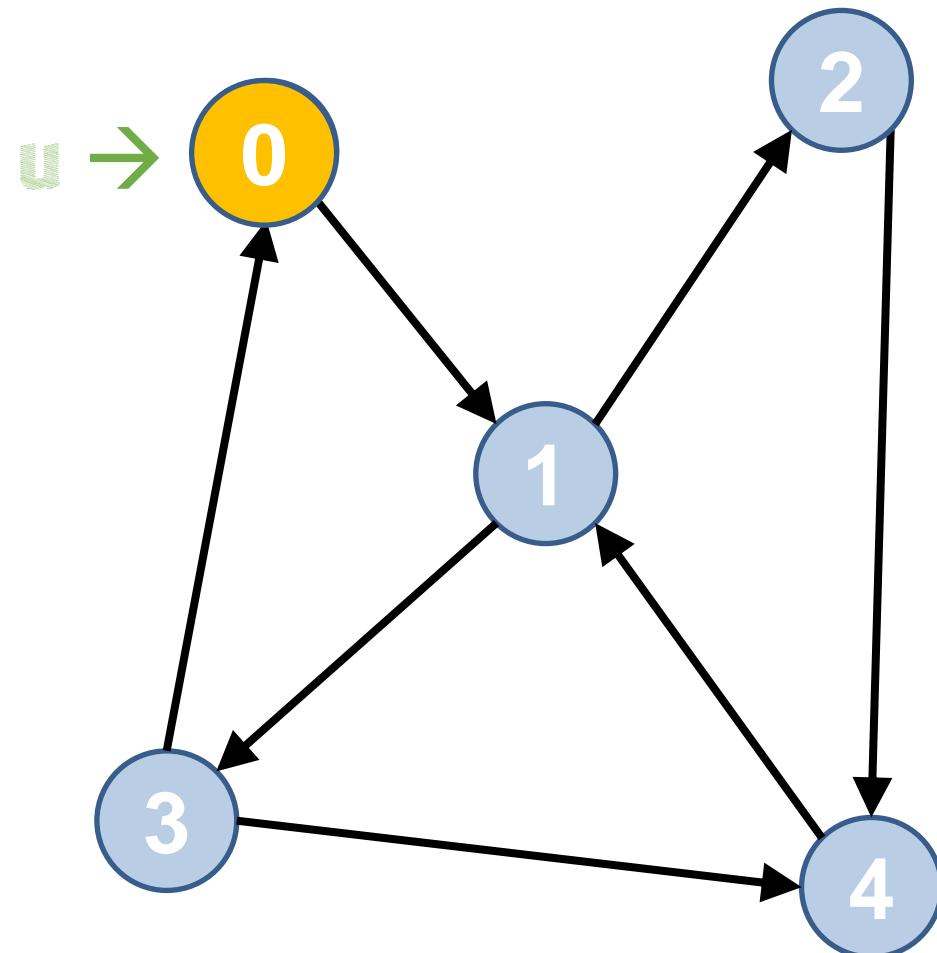
FILA

0

Busca em Largura

Procurar o 5

- 01 Escolher nó inicial (**u**)
- 02 Visitar **u** //testar
- 03 Adicione **u** na Fila
- 04 Enquanto a fila não estiver vazia
 Remova o primeiro nó da fila (**u**)
 Para cada **v** adjacente não visitado de **u**
 Marcar como visitado //testar
 Adicionar **v** a fila
- 09 Fim Enquanto
- 10 Se houver nó não visitado, repetir com novo **u**

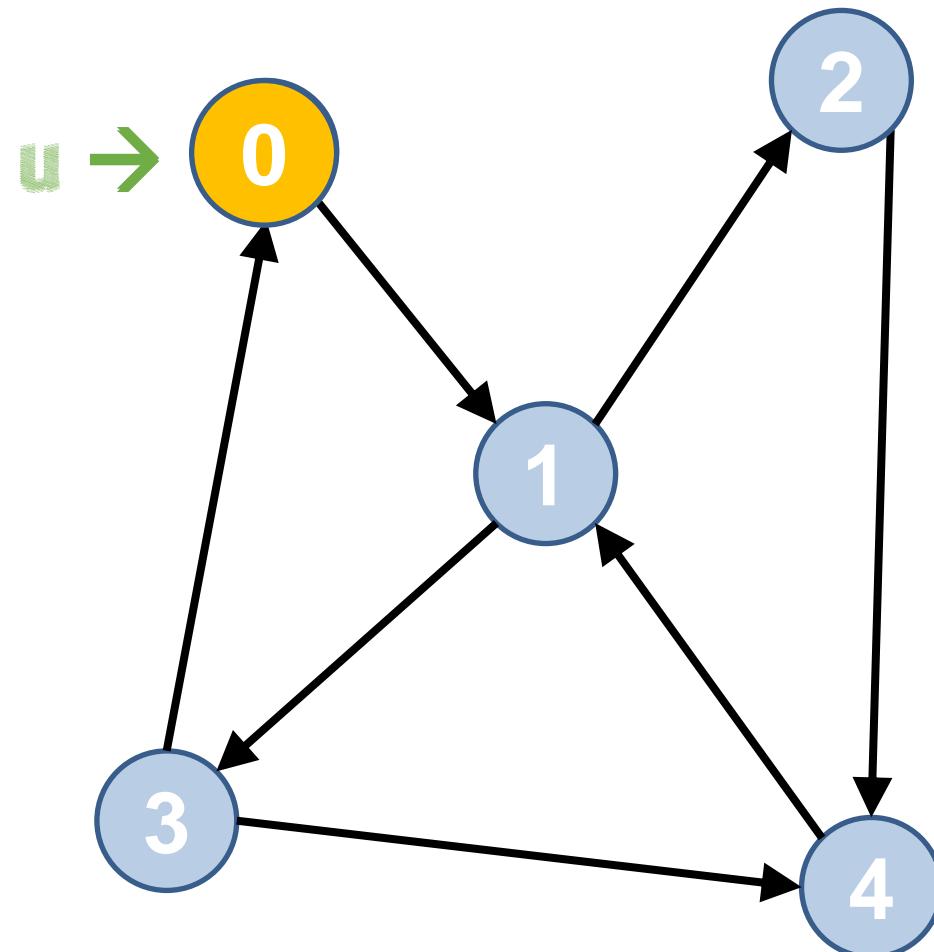


FILA | 0 |

Busca em Largura

Procurar o 5

- 01 Escolher nó inicial (**u**)
- 02 Visitar **u** //testar
- 03 Adicione **u** na Fila
- 04 Enquanto a fila não estiver vazia
 Remova o primeiro nó da fila (**u**)
 Para cada **v** adjacente não visitado de **u**
 Marcar como visitado //testar
 Adicionar **v** a fila
- 09 Fim Enquanto
- 10 Se houver nó não visitado, repetir com novo **u**

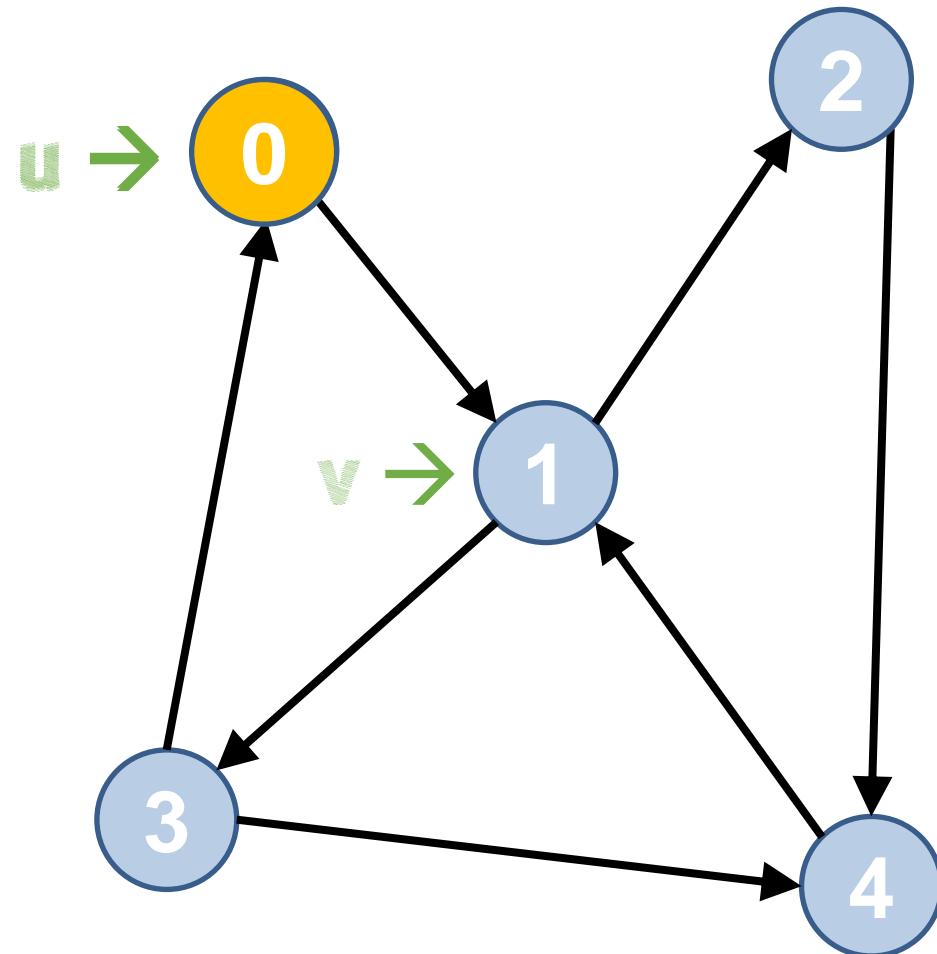


FILA

Busca em Largura

Procurar o 5

- 01 Escolher nó inicial (**u**)
- 02 Visitar **u** //testar
- 03 Adicione **u** na Fila
- 04 Enquanto a fila não estiver vazia
 Remova o primeiro nó da fila (**u**)
 Para cada **v** adjacente não visitado de **u**
 Marcar como visitado //testar
 Adicionar **v** a fila
- 09 Fim Enquanto
- 10 Se houver nó não visitado, repetir com novo **u**

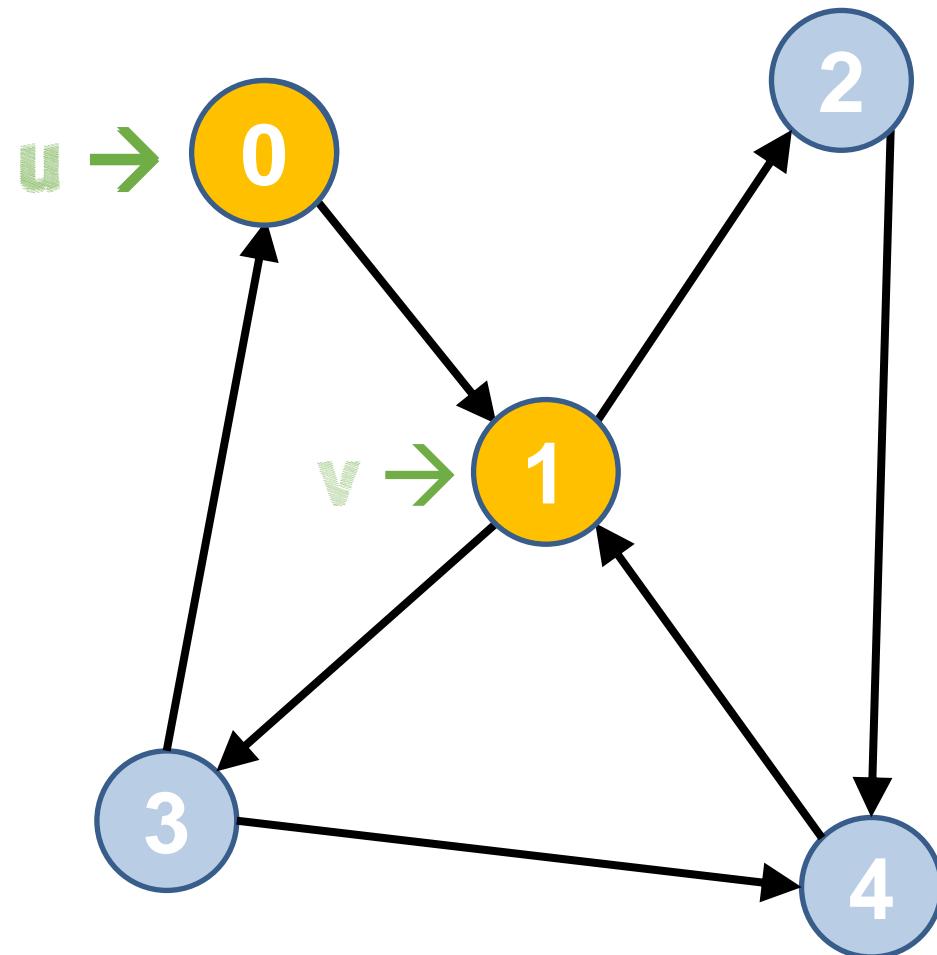


FILA

Busca em Largura

Procurar o 5

- 01 Escolher nó inicial (**u**)
- 02 Visitar **u** //testar
- 03 Adicione **u** na Fila
- 04 Enquanto a fila não estiver vazia
 Remova o primeiro nó da fila (**u**)
 Para cada **v** adjacente não visitado de **u**
 Marcar como visitado //testar
 Adicionar **v** a fila
- 09 Fim Enquanto
- 10 Se houver nó não visitado, repetir com novo **u**

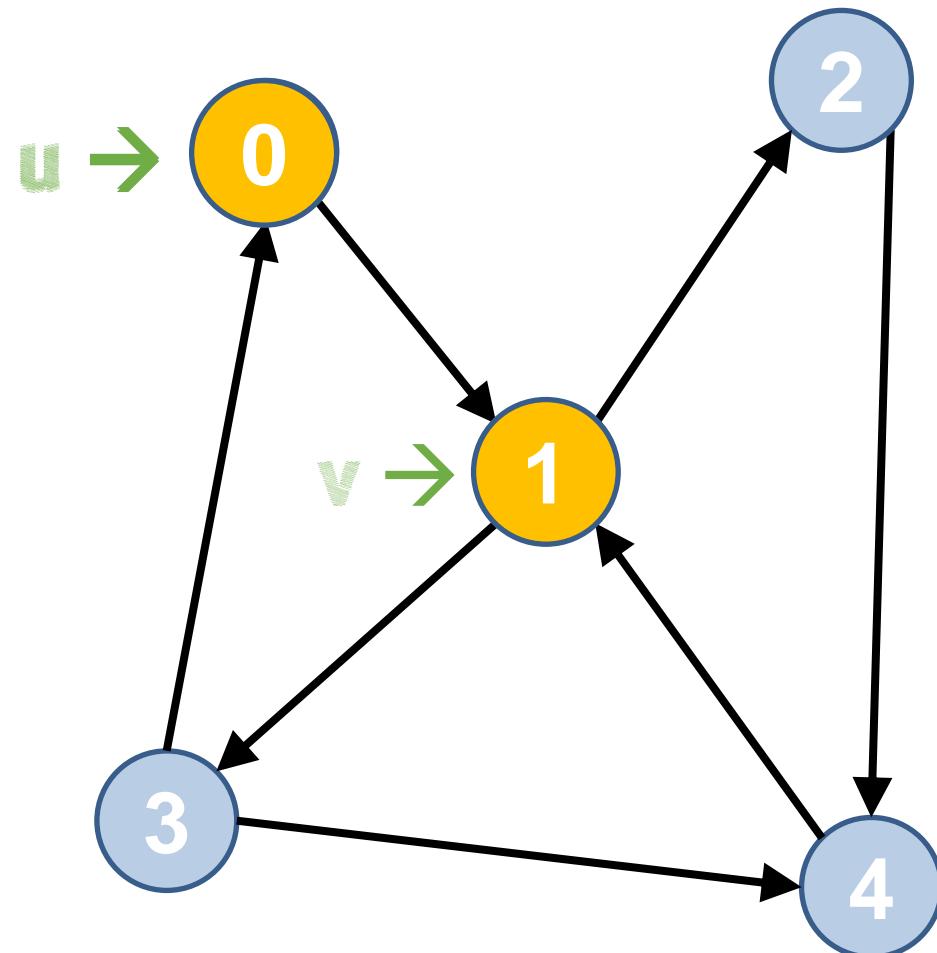


FILA

Busca em Largura

Procurar o 5

- 01 Escolher nó inicial (**u**)
- 02 Visitar **u** //testar
- 03 Adicione **u** na Fila
- 04 Enquanto a fila não estiver vazia
 Remova o primeiro nó da fila (**u**)
 Para cada **v** adjacente não visitado de **u**
 Marcar como visitado //testar
 Adicionar **v** a fila
- 09 Fim Enquanto
- 10 Se houver nó não visitado, repetir com novo **u**



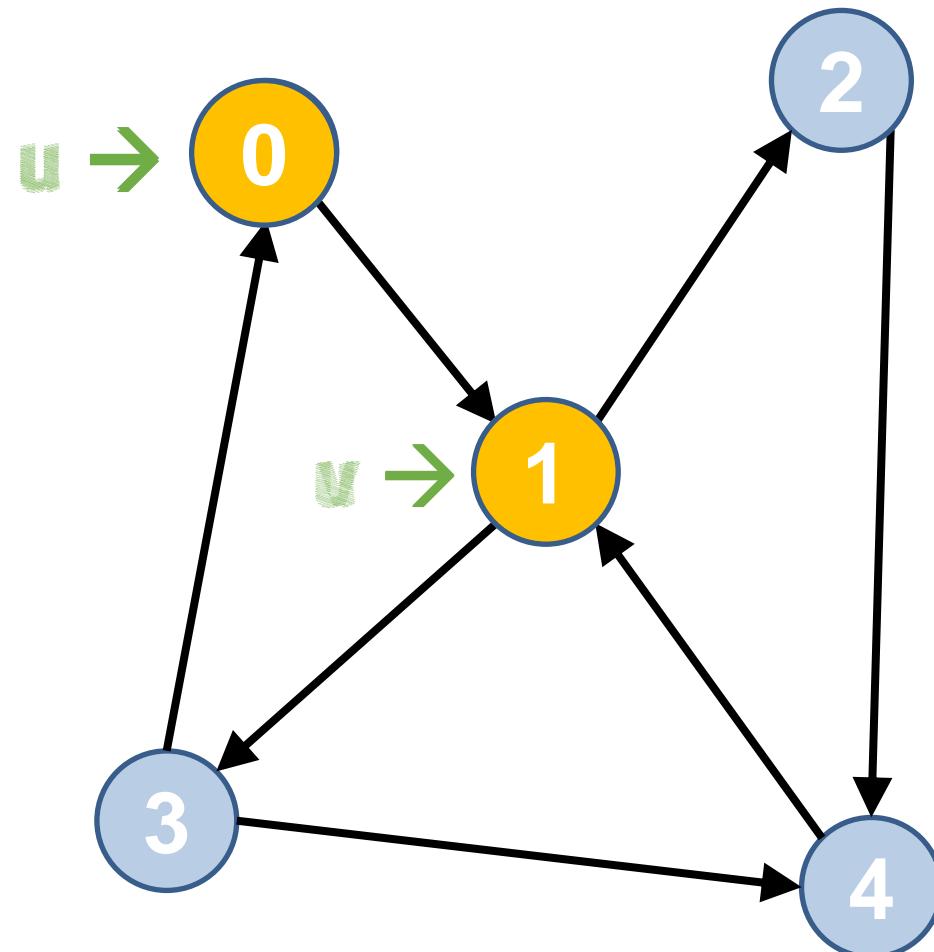
FILA

1

Busca em Largura

Procurar o 5

- 01 Escolher nó inicial (**u**)
- 02 Visitar **u** //testar
- 03 Adicione **u** na Fila
- 04 Enquanto a fila não estiver vazia
 Remova o primeiro nó da fila (**u**)
 Para cada **v** adjacente não visitado de **u**
 Marcar como visitado //testar
 Adicionar **v** a fila
- 09 Fim Enquanto
- 10 Se houver nó não visitado, repetir com novo **u**

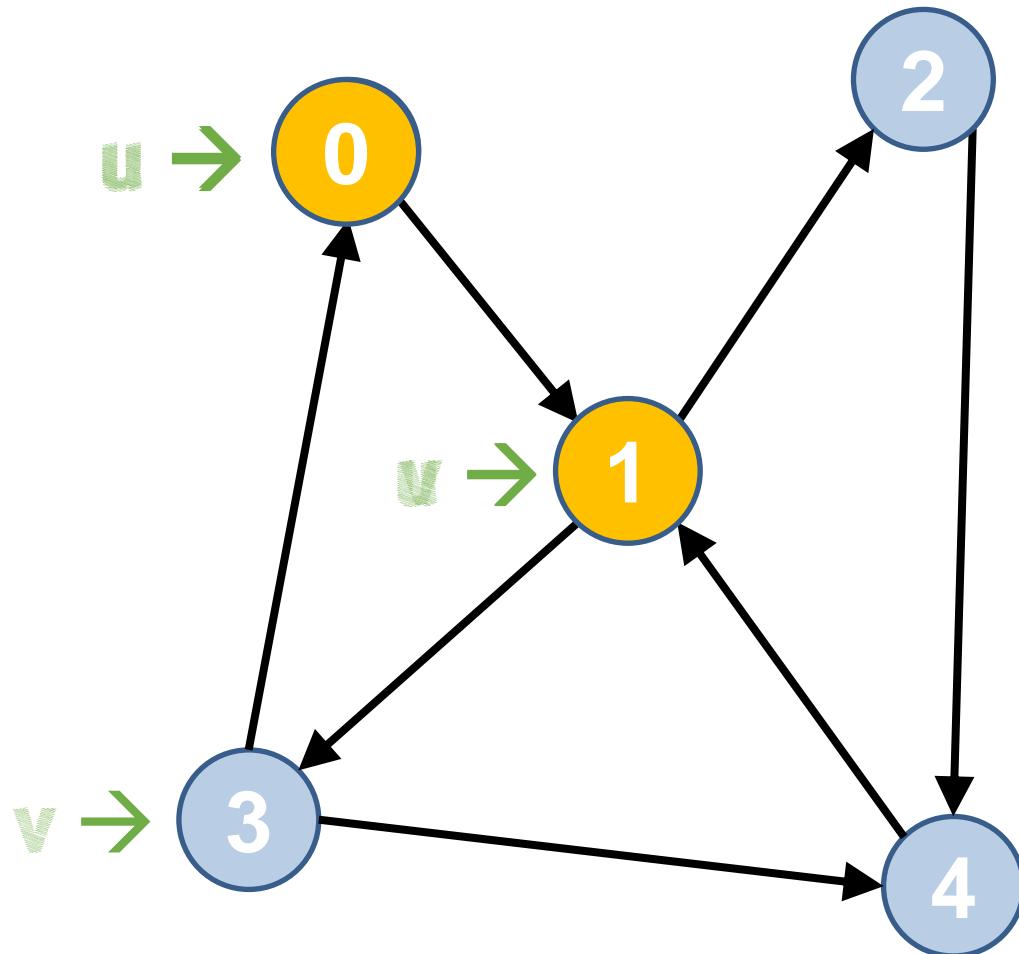


FILA

Busca em Largura

Procurar o 5

- 01 Escolher nó inicial (**u**)
- 02 Visitar **u** //testar
- 03 Adicione **u** na Fila
- 04 Enquanto a fila não estiver vazia
 Remova o primeiro nó da fila (**u**)
 Para cada **v** adjacente não visitado de **u**
 Marcar como visitado //testar
 Adicionar **v** a fila
- 09 Fim Enquanto
- 10 Se houver nó não visitado, repetir com novo **u**

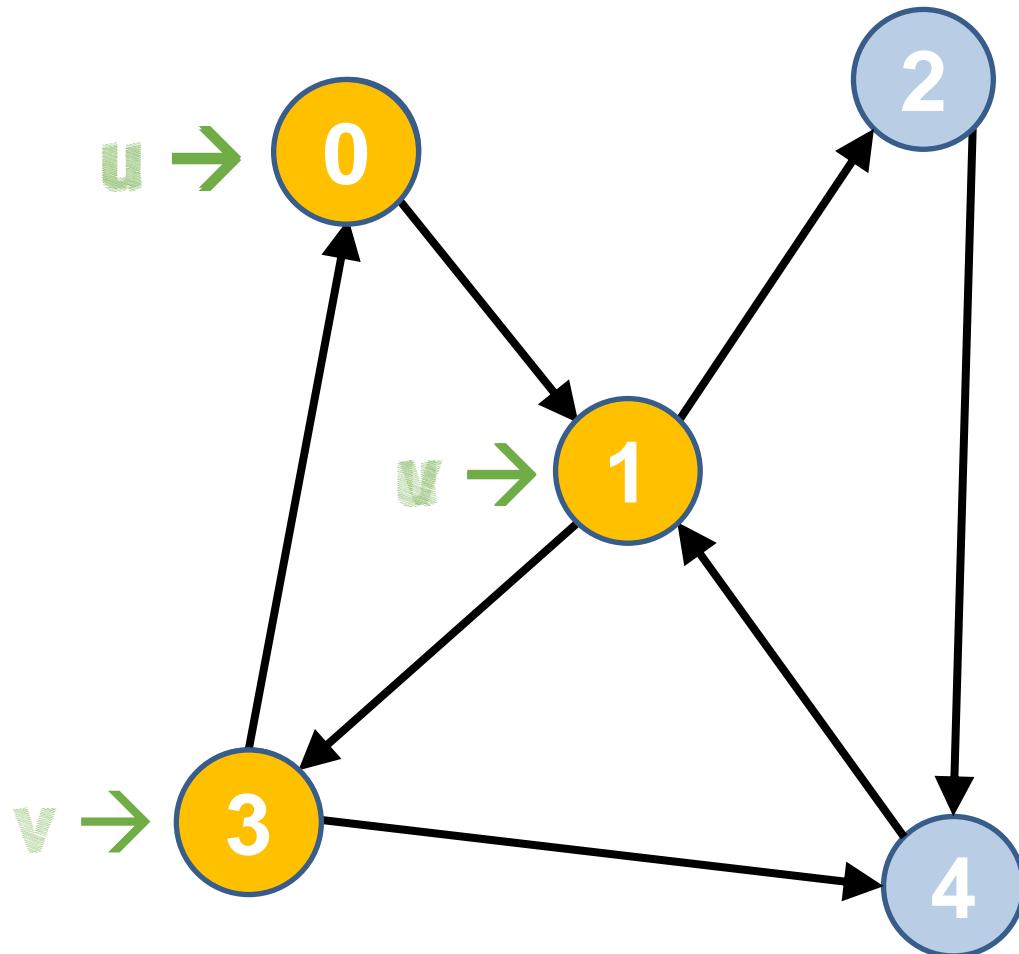


FILA

Busca em Largura

Procurar o 5

- 01 Escolher nó inicial (**u**)
- 02 Visitar **u** //testar
- 03 Adicione **u** na Fila
- 04 Enquanto a fila não estiver vazia
 Remova o primeiro nó da fila (**u**)
 Para cada **v** adjacente não visitado de **u**
 Marcar como visitado //testar
 Adicionar **v** a fila
- 09 Fim Enquanto
- 10 Se houver nó não visitado, repetir com novo **u**

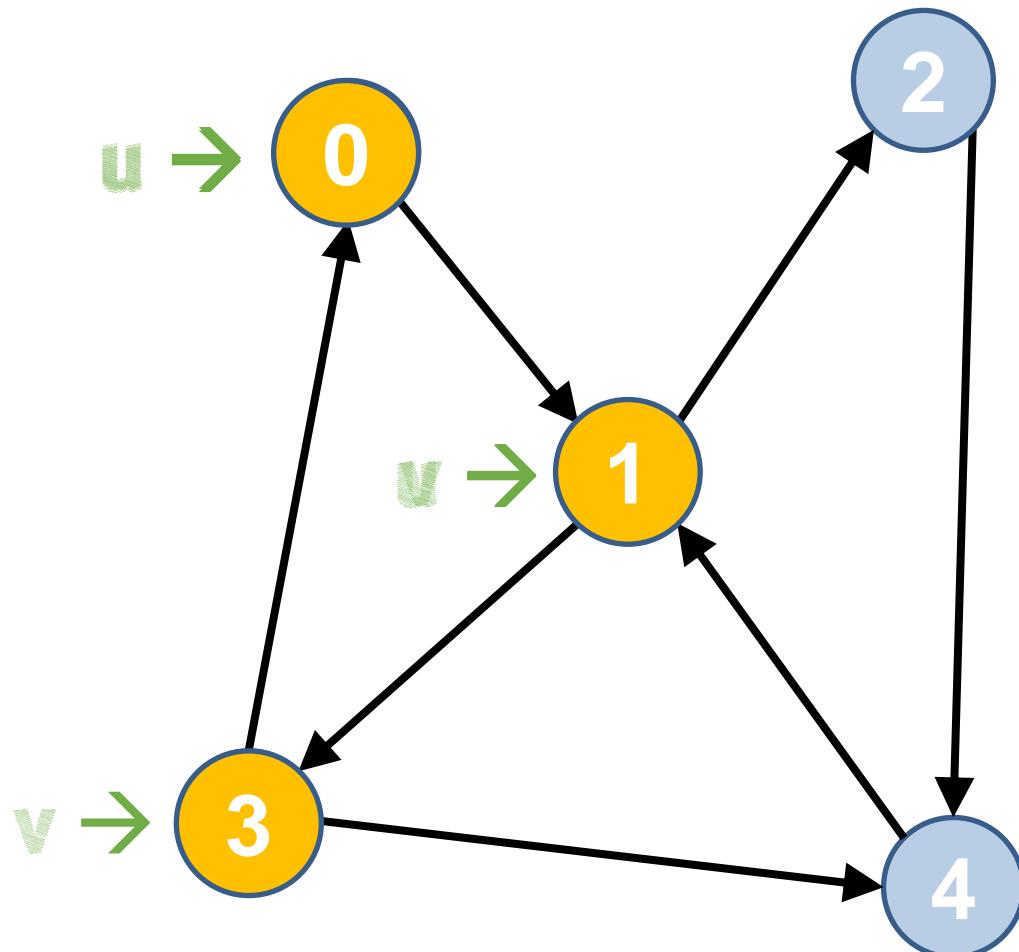


FILA

Busca em Largura

Procurar o 5

- 01 Escolher nó inicial (**u**)
- 02 Visitar **u** //testar
- 03 Adicione **u** na Fila
- 04 Enquanto a fila não estiver vazia
 Remova o primeiro nó da fila (**u**)
 Para cada **v** adjacente não visitado de **u**
 Marcar como visitado //testar
 Adicionar **v** a fila
- 09 Fim Enquanto
- 10 Se houver nó não visitado, repetir com novo **u**

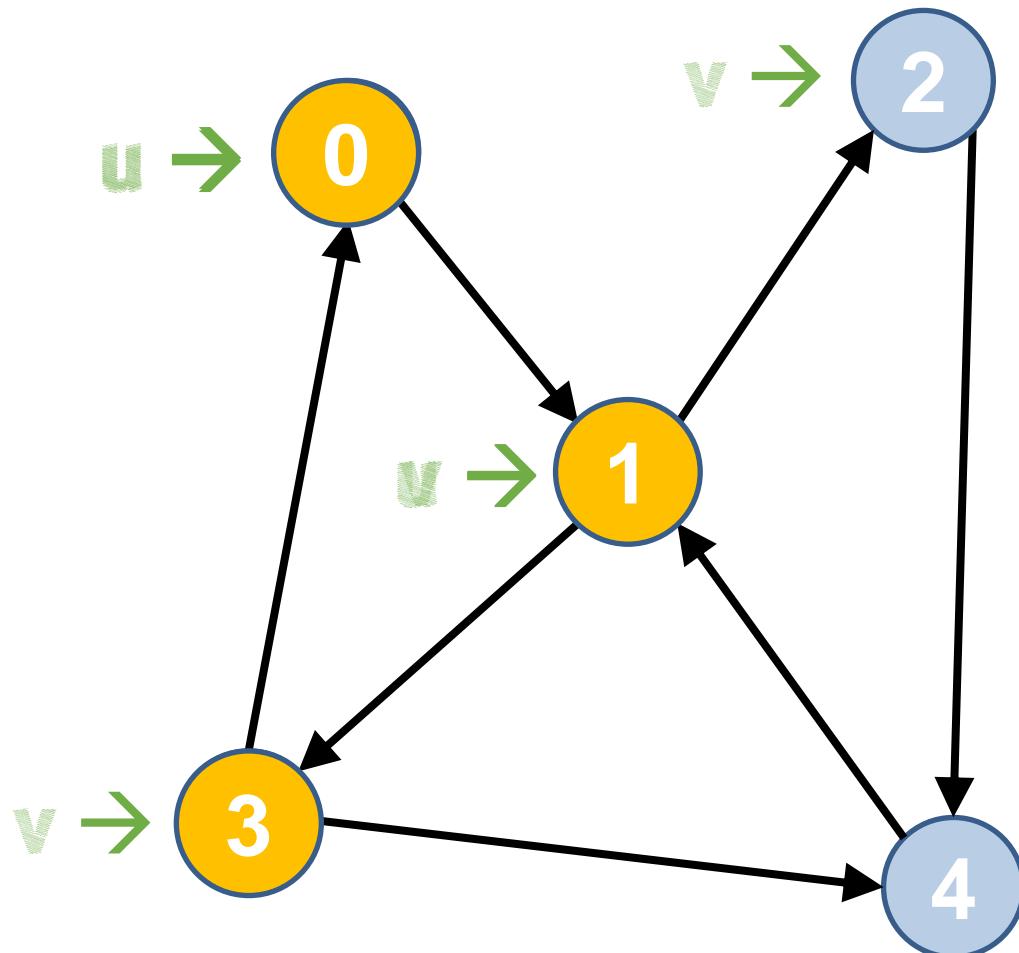


FILA | 3 |

Busca em Largura

Procurar o 5

- 01 Escolher nó inicial (**u**)
- 02 Visitar **u** //testar
- 03 Adicione **u** na Fila
- 04 Enquanto a fila não estiver vazia
 Remova o primeiro nó da fila (**u**)
 Para cada **v** adjacente não visitado de **u**
 Marcar como visitado //testar
 Adicionar **v** a fila
- 09 Fim Enquanto
- 10 Se houver nó não visitado, repetir com novo **u**

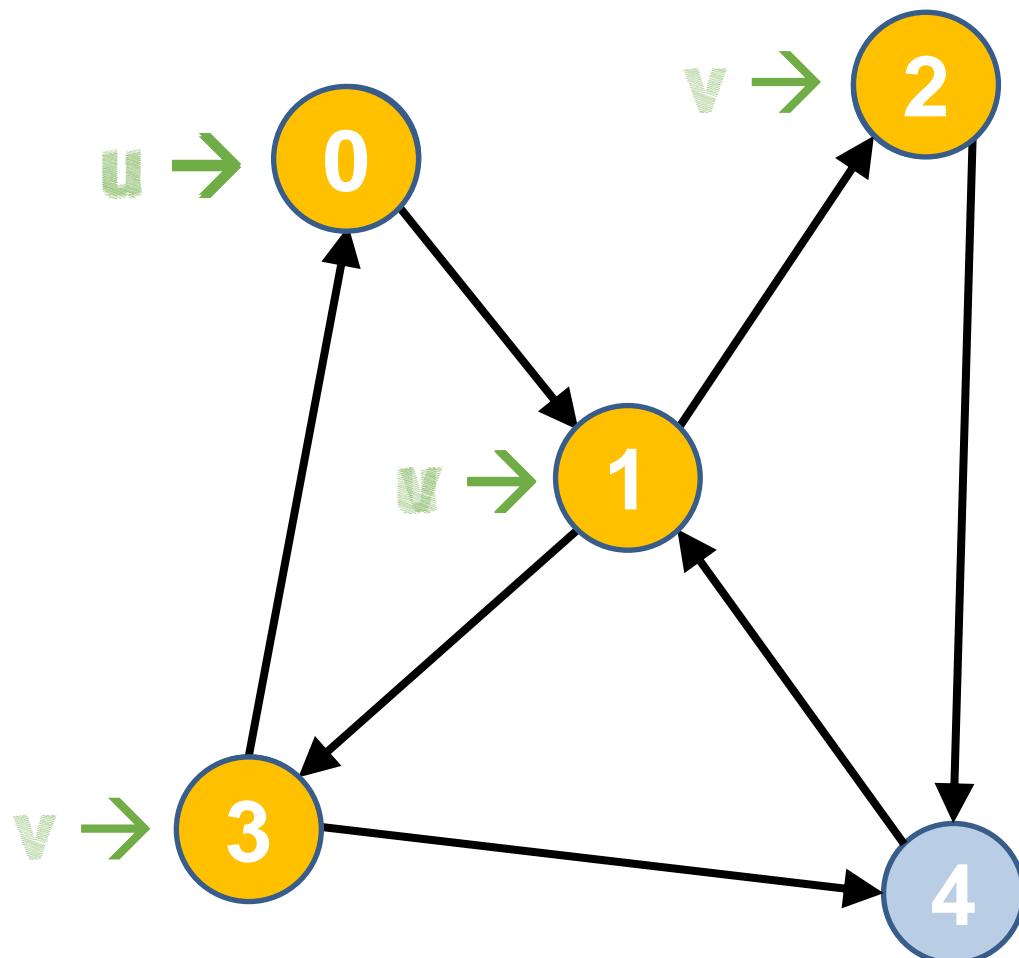


FILA | 3 |

Busca em Largura

Procurar o 5

- 01 Escolher nó inicial (**u**)
- 02 Visitar **u** //testar
- 03 Adicione **u** na Fila
- 04 Enquanto a fila não estiver vazia
 Remova o primeiro nó da fila (**u**)
 Para cada **v** adjacente não visitado de **u**
 Marcar como visitado //testar
 Adicionar **v** a fila
- 09 Fim Enquanto
- 10 Se houver nó não visitado, repetir com novo **u**

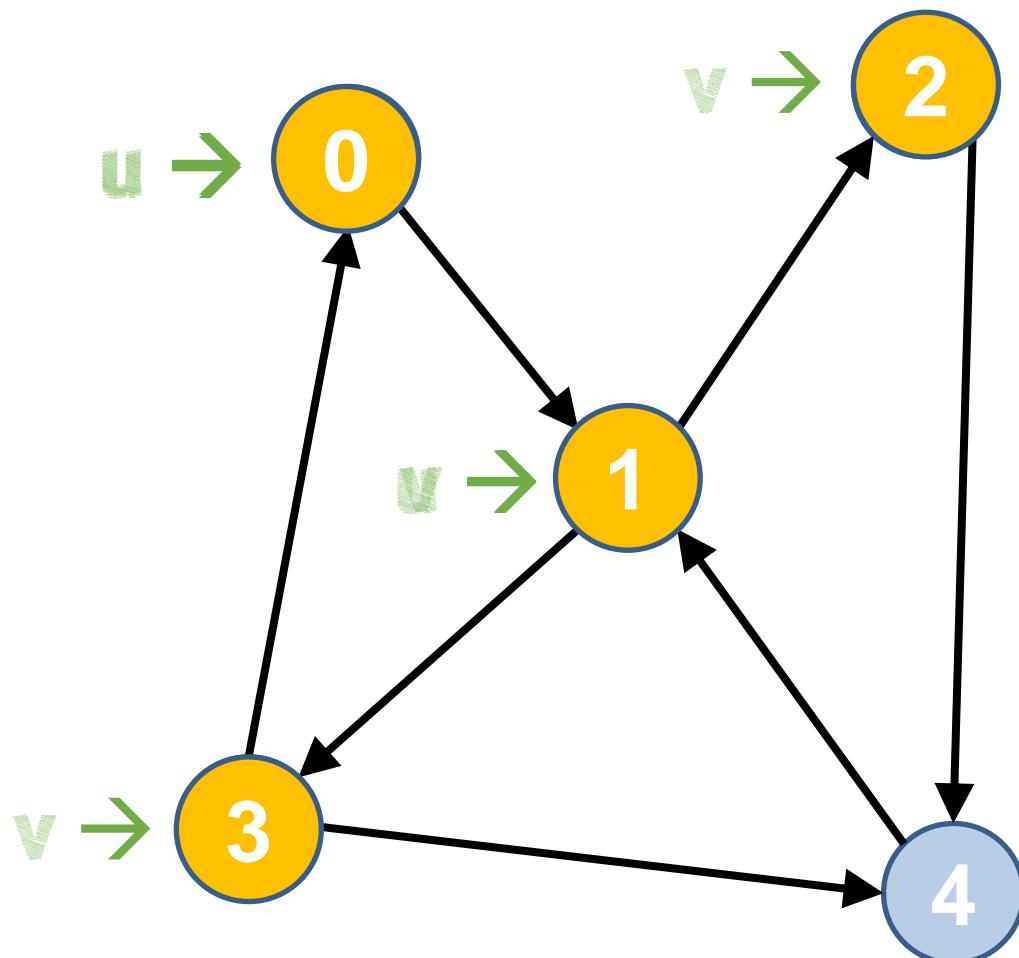


FILA | 3 |

Busca em Largura

Procurar o 5

- 01 Escolher nó inicial (**u**)
- 02 Visitar **u** //testar
- 03 Adicione **u** na Fila
- 04 Enquanto a fila não estiver vazia
 Remova o primeiro nó da fila (**u**)
 Para cada **v** adjacente não visitado de **u**
 Marcar como visitado //testar
 Adicionar **v** a fila
- 09 Fim Enquanto
- 10 Se houver nó não visitado, repetir com novo **u**

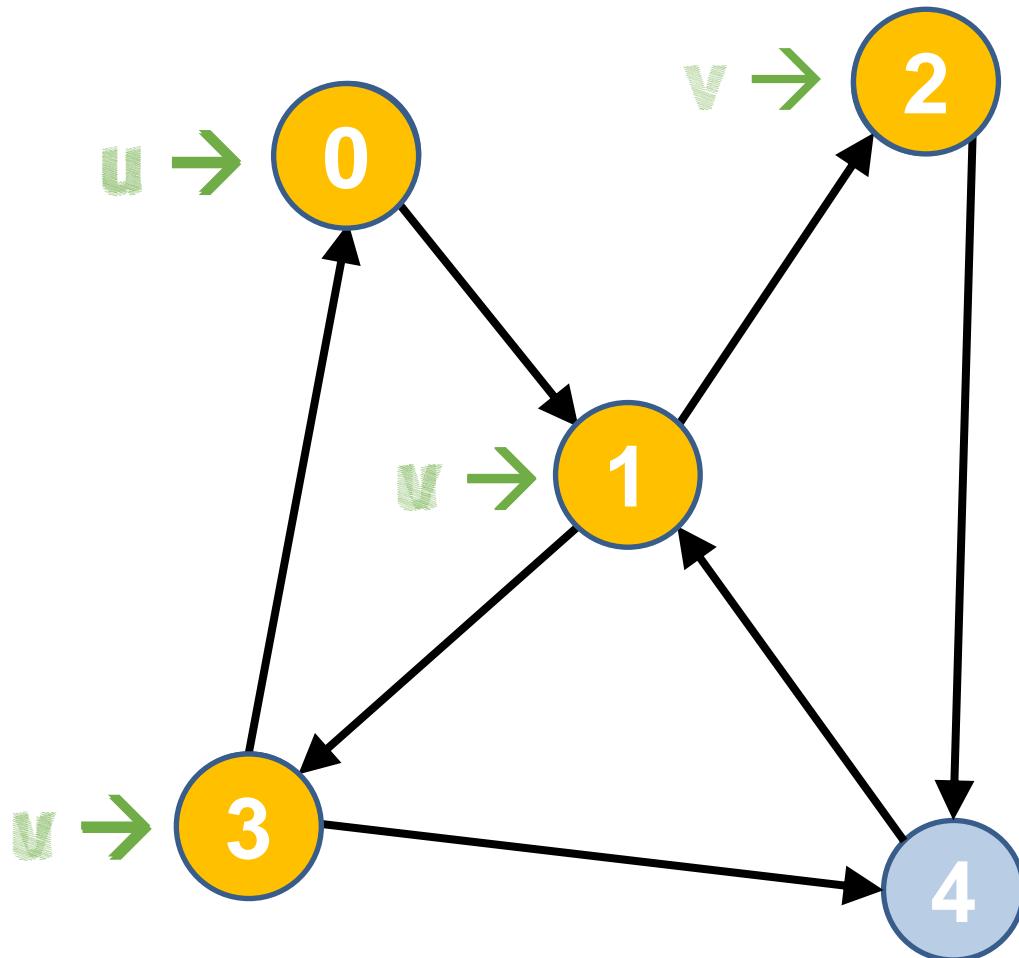


FILA | 3 | 2 |

Busca em Largura

Procurar o 5

- 01 Escolher nó inicial (**u**)
- 02 Visitar **u** //testar
- 03 Adicione **u** na Fila
- 04 Enquanto a fila não estiver vazia
 - 05 Remova o primeiro nó da fila (**u**)
 - 06 Para cada **v** adjacente não visitado de **u**
 - 07 Marcar como visitado //testar
 - 08 Adicionar **v** a fila
- 09 Fim Enquanto
- 10 Se houver nó não visitado, repetir com novo **u**



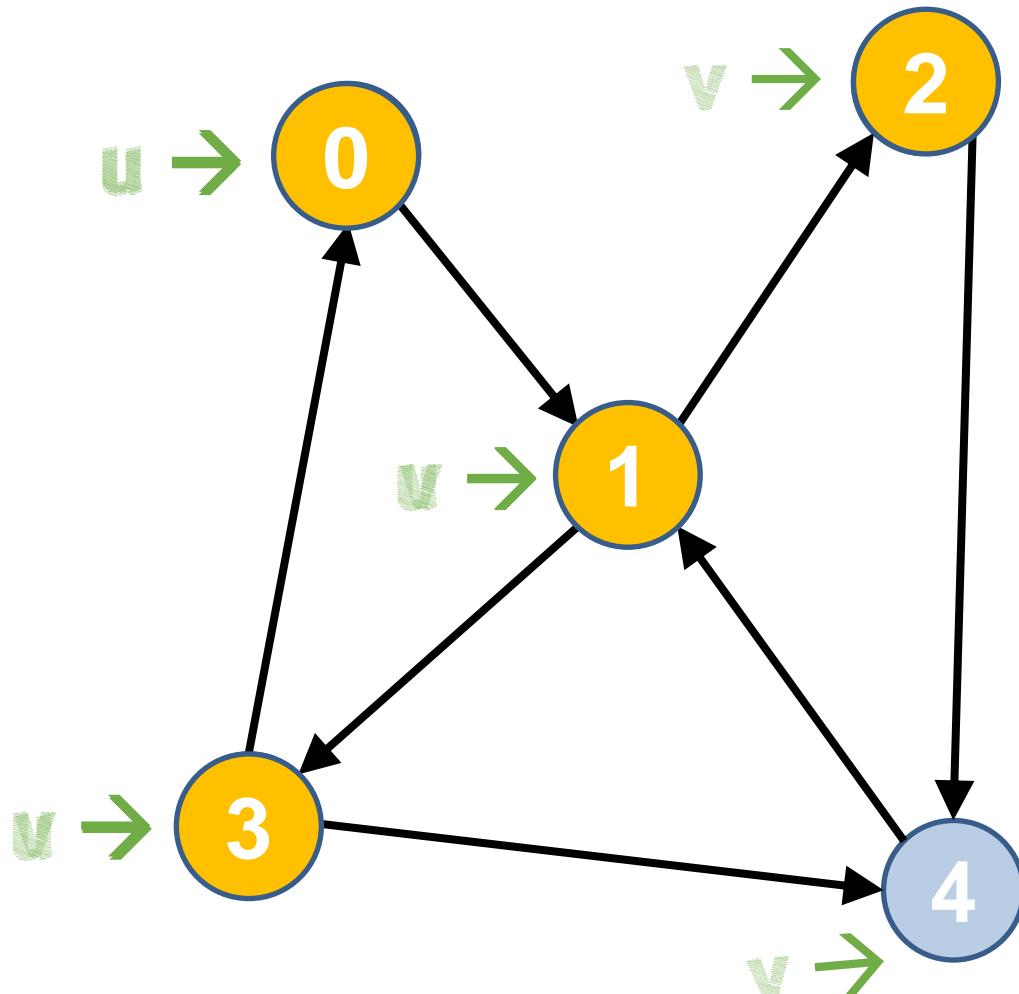
FILA

2

Busca em Largura

Procurar o 5

- 01 Escolher nó inicial (**u**)
- 02 Visitar **u** //testar
- 03 Adicione **u** na Fila
- 04 Enquanto a fila não estiver vazia
 Remova o primeiro nó da fila (**u**)
 Para cada **v** adjacente não visitado de **u**
 Marcar como visitado //testar
 Adicionar **v** a fila
- 09 Fim Enquanto
- 10 Se houver nó não visitado, repetir com novo **u**

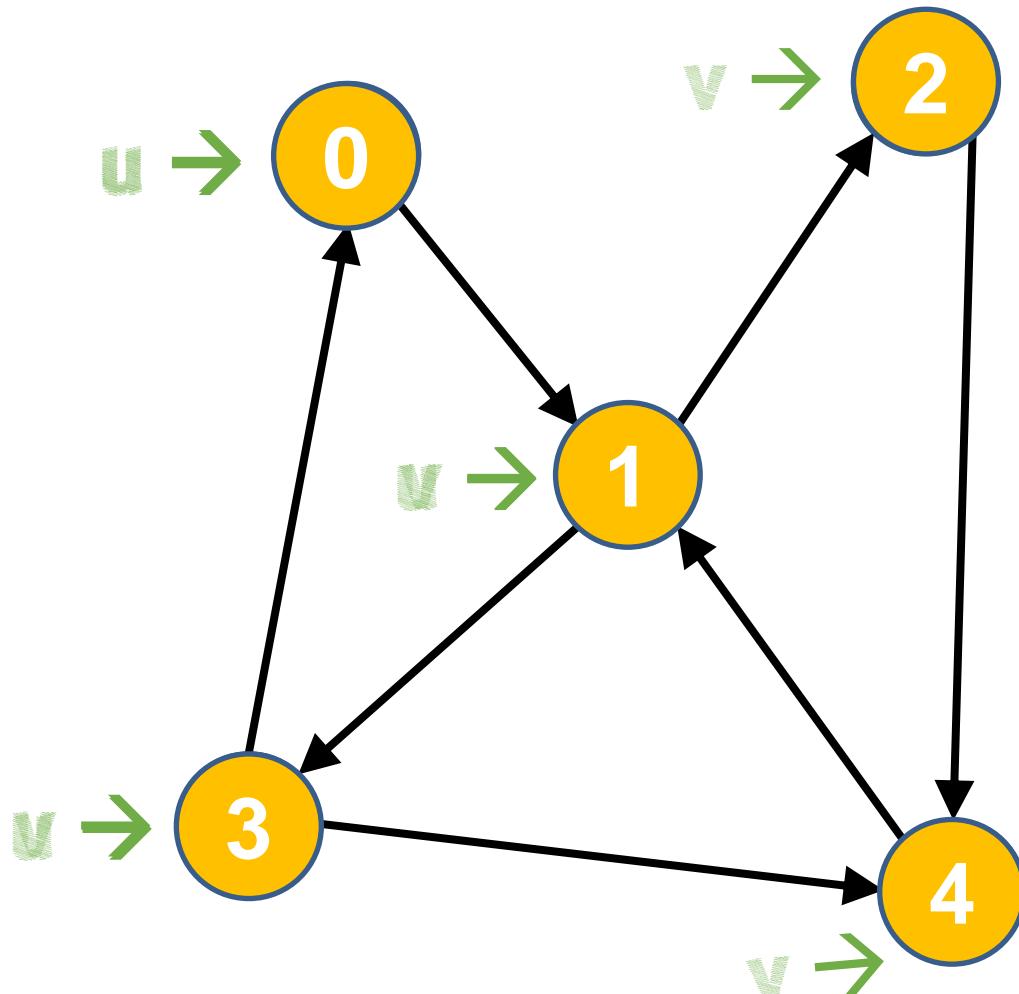


FILA | 2 |

Busca em Largura

Procurar o 5

- 01 Escolher nó inicial (**u**)
- 02 Visitar **u** //testar
- 03 Adicione **u** na Fila
- 04 Enquanto a fila não estiver vazia
 Remova o primeiro nó da fila (**u**)
 Para cada **v** adjacente não visitado de **u**
 Marcar como visitado //testar
 Adicionar **v** a fila
- 09 Fim Enquanto
- 10 Se houver nó não visitado, repetir com novo **u**

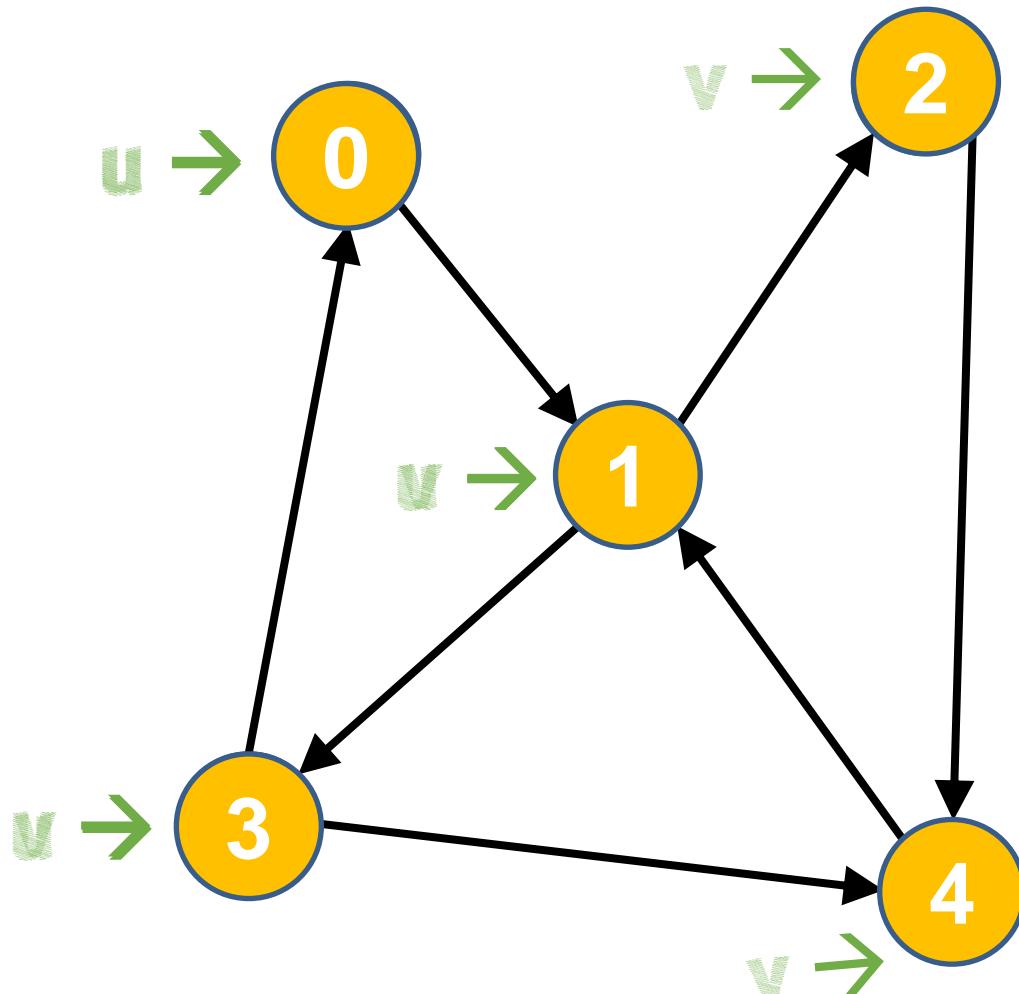


FILA | 2 |

Busca em Largura

Procurar o 5

- 01 Escolher nó inicial (**u**)
- 02 Visitar **u** //testar
- 03 Adicione **u** na Fila
- 04 Enquanto a fila não estiver vazia
 Remova o primeiro nó da fila (**u**)
 Para cada **v** adjacente não visitado de **u**
 Marcar como visitado //testar
 Adicionar **v** a fila
- 09 Fim Enquanto
- 10 Se houver nó não visitado, repetir com novo **u**

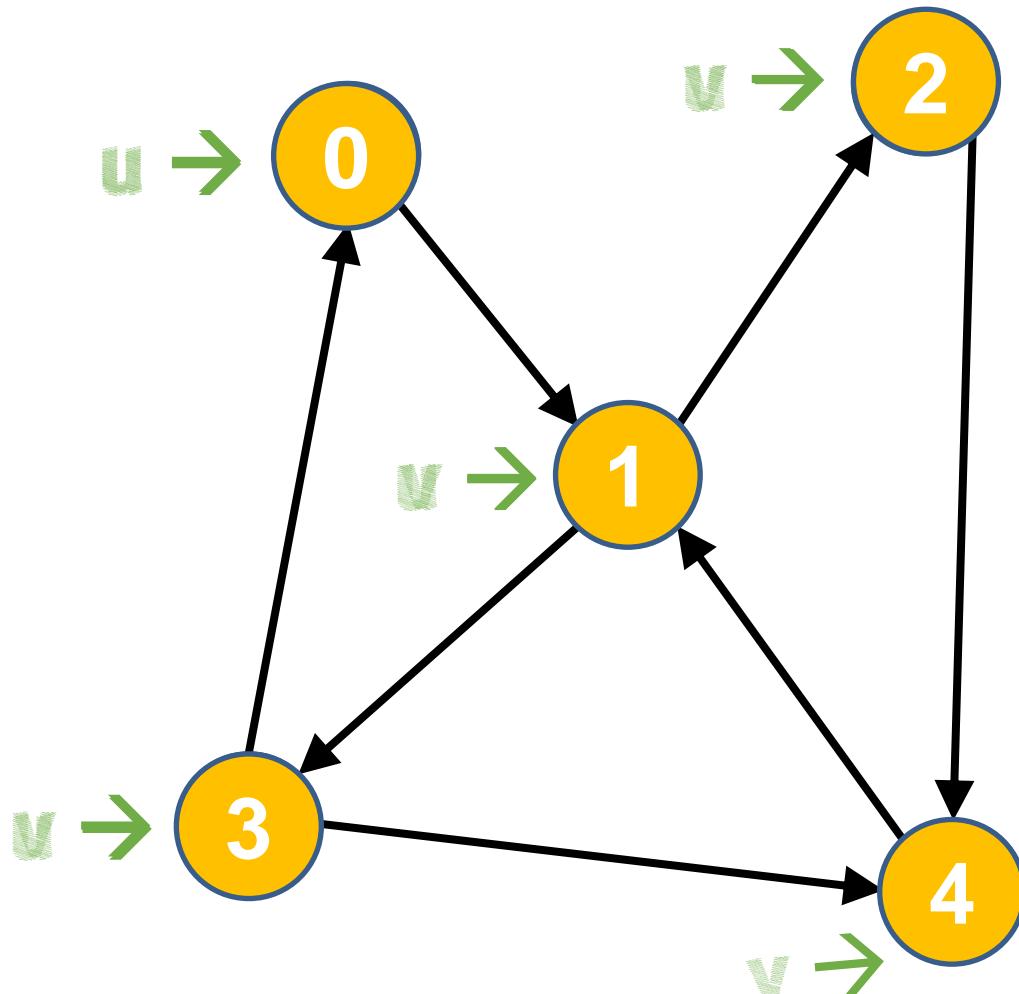


FILA | 2 | 4 |

Busca em Largura

Procurar o 5

- 01 Escolher nó inicial (**u**)
- 02 Visitar **u** //testar
- 03 Adicione **u** na Fila
- 04 Enquanto a fila não estiver vazia
 - 05 Remova o primeiro nó da fila (**u**)
 - 06 Para cada **v** adjacente não visitado de **u**
 - 07 Marcar como visitado //testar
 - 08 Adicionar **v** a fila
- 09 Fim Enquanto
- 10 Se houver nó não visitado, repetir com novo **u**

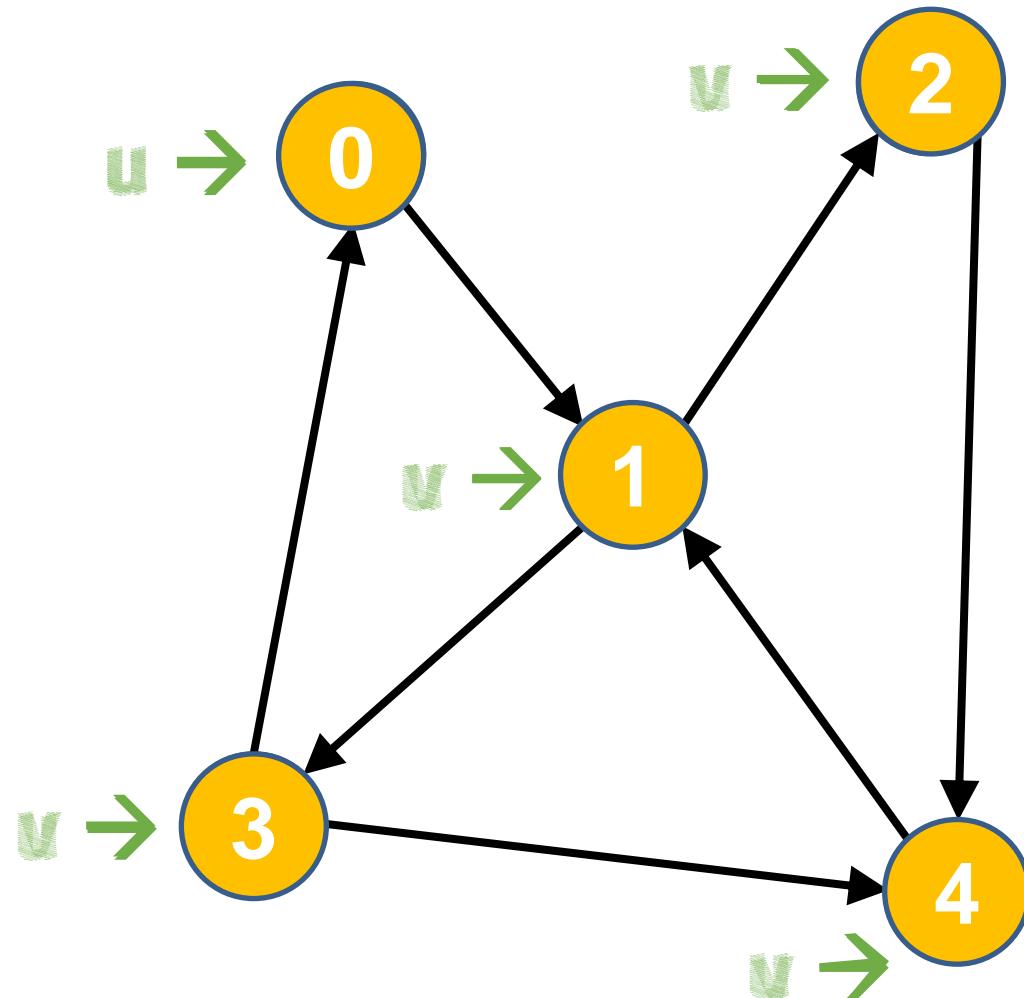


FILA | 4 |

Busca em Largura

Procurar o 5

- 01 Escolher nó inicial (**u**)
- 02 Visitar **u** //testar
- 03 Adicione **u** na Fila
- 04 Enquanto a fila não estiver vazia
 Remova o primeiro nó da fila (**u**)
 Para cada **v** adjacente não visitado de **u**
 Marcar como visitado //testar
 Adicionar **v** a fila
- 09 Fim Enquanto
- 10 Se houver nó não visitado, repetir com novo **u**



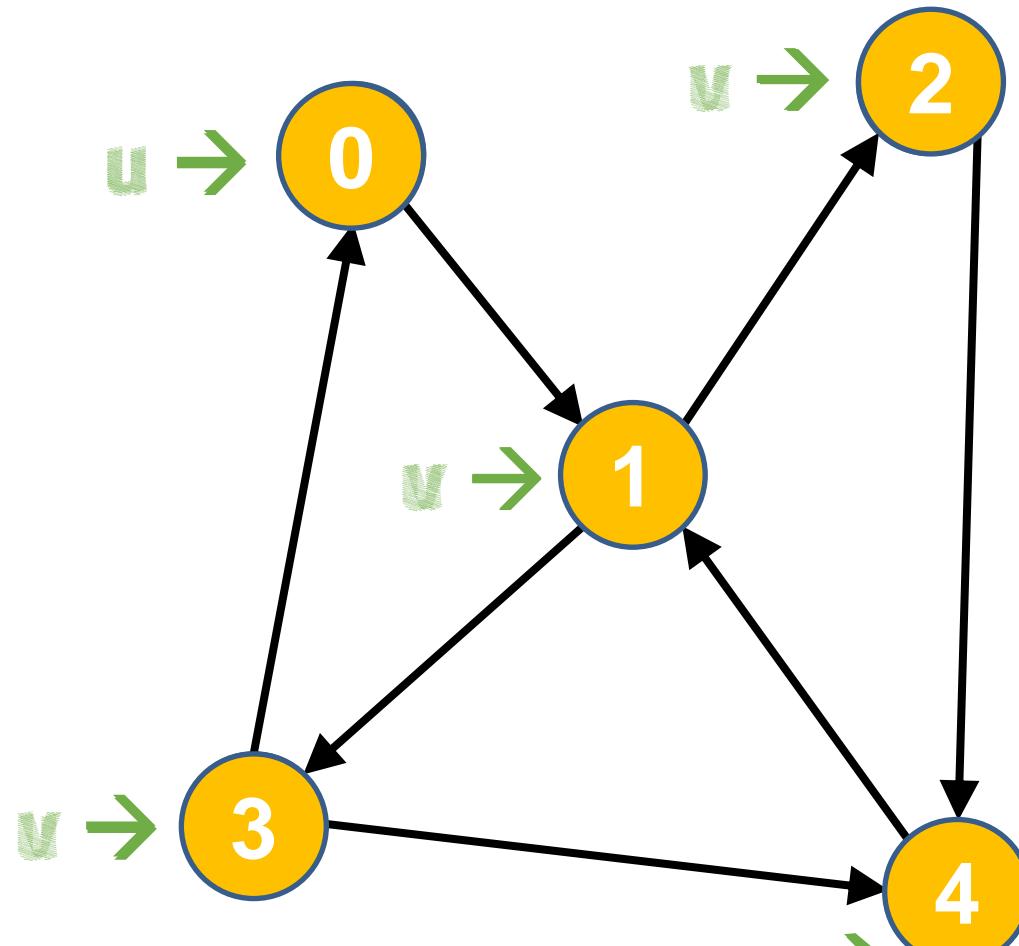
FILA

Busca em Largura

Procurar o 5

- 01 Escolher nó inicial (**u**)
- 02 Visitar **u** //testar
- 03 Adicione **u** na Fila
- 04 Enquanto a fila não estiver vazia
 Remova o primeiro nó da fila (**u**)
 Para cada **v** adjacente não visitado de **u**
 Marcar como visitado //testar
 Adicionar **v** a fila
- 09 Fim Enquanto
- 10 Se houver nó não visitado, repetir com novo **u**

0 → 1 → 3 → 2 → 4



FILA

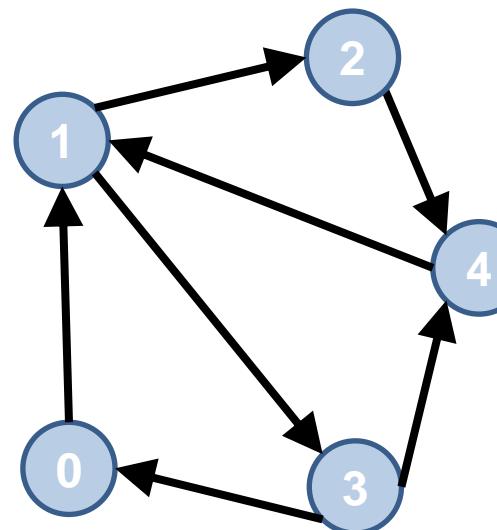
**Busca
Menor Caminho**

Problema do Menor Caminho (PMC)

- A busca em largura provê o menor caminho (**qtd de arestas**) entre um vértice e todos os demais, ignora os pesos das arestas
- O **PMC** consiste em determinar o caminho mais curto entre 2 vértices
 - O menor caminho entre os vértices **A** e **B** é a aresta que os conecta, porém, é comum que vértices não sejam adjacentes

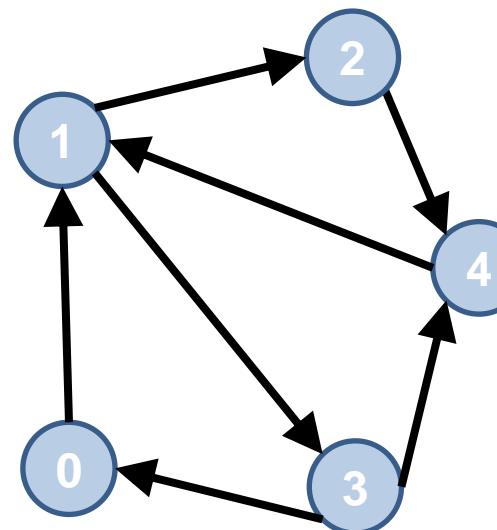
Problema do Menor Caminho (PMC)

- A busca em largura provê o menor caminho (**qtd de arestas**) entre um vértice e todos os demais, ignora os pesos das arestas
- O **PMC** consiste em determinar o caminho mais curto entre 2 vértices
 - O menor caminho entre os vértices **A** e **B** é a aresta que os conecta, porém, é comum que vértices não sejam adjacentes



Problema do Menor Caminho (PMC)

- A busca em largura provê o menor caminho (**qtd de arestas**) entre um vértice e todos os demais, ignora os pesos das arestas
- O **PMC** consiste em determinar o caminho mais curto entre 2 vértices
 - O menor caminho entre os vértices **A** e **B** é a aresta que os conecta, porém, é comum que vértices não sejam adjacentes

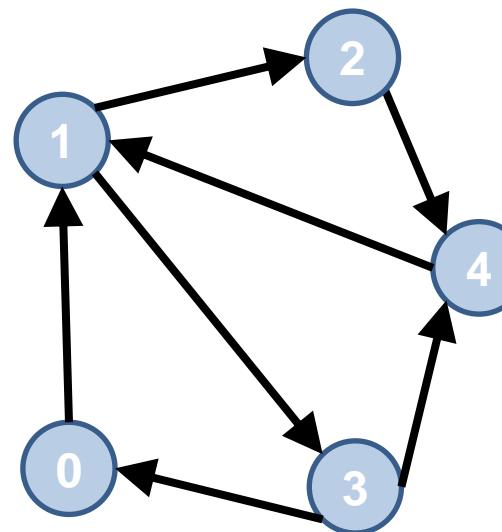


V₀ e V₄ não são adjacentes, mas estão conectados:
(0,1), (1,2), (2,4)

Problema do Menor Caminho (PMC)

- A busca em largura provê o menor caminho (**qtd de arestas**) entre um vértice e todos os demais, ignora os pesos das arestas
- O **PMC** consiste em determinar o caminho mais curto entre 2 vértices
 - O menor caminho entre os vértices **A** e **B** é a aresta que os conecta, porém, é comum que vértices não sejam adjacentes

O menor caminho é
a menor sequência
de arestas ligando 2
vértices

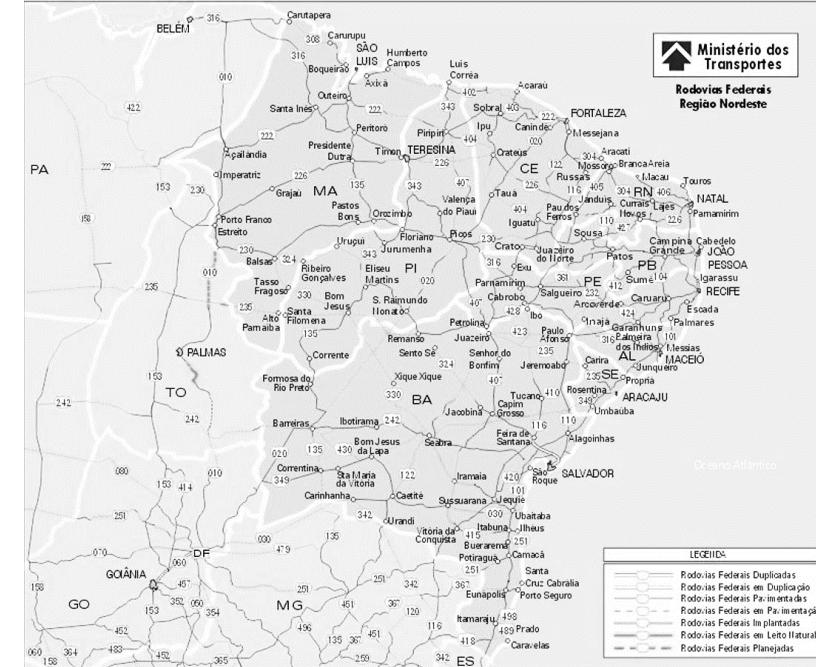


V₀ e V₄ não são
adjacentes,
mas estão
conectados:
(0,1), (1,2), (2,4)

Problema do Menor Caminho (PMC)

... estudar e praticar ... the best way for everything

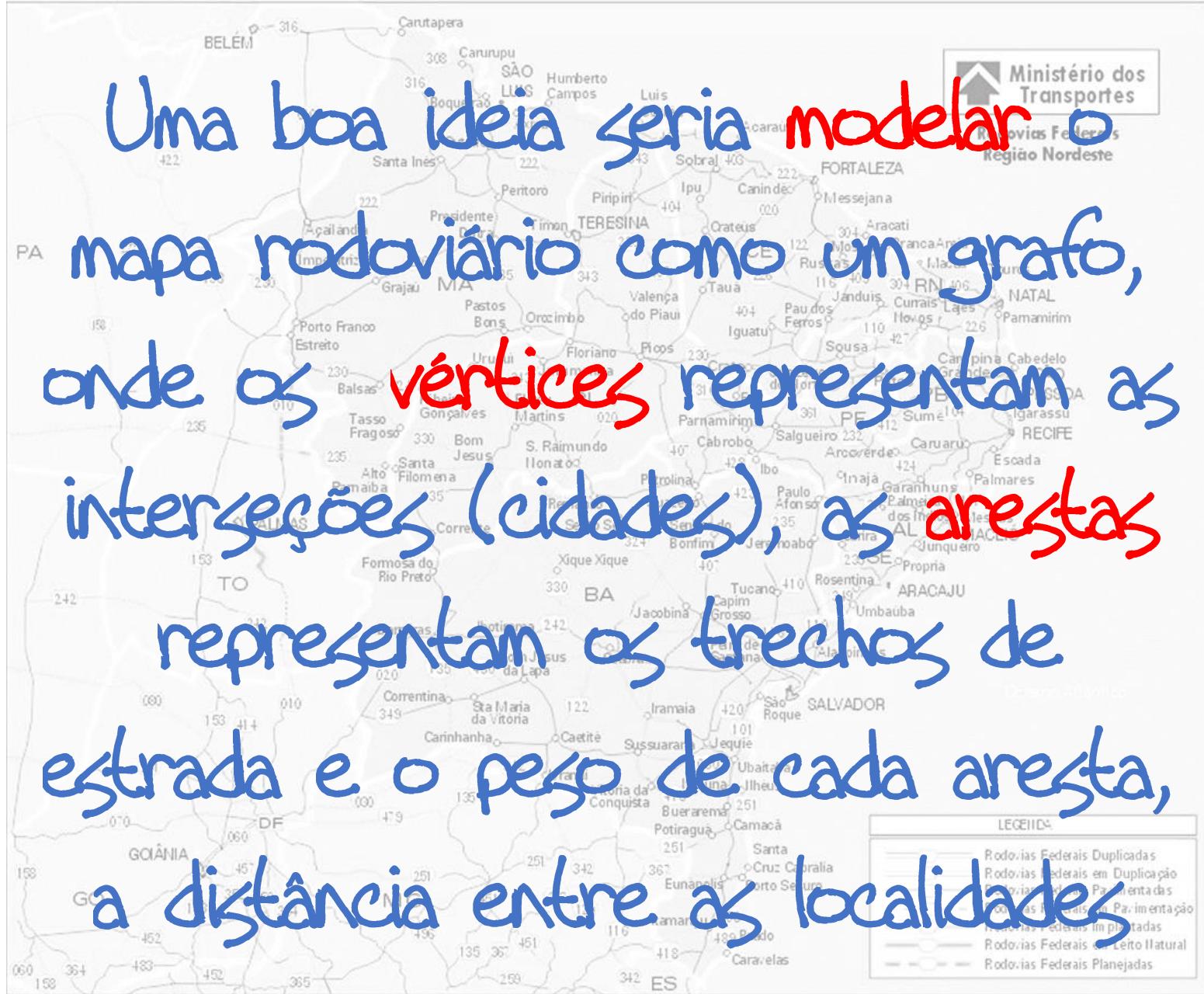
- Aplicando a busca em largura para encontrar a menor rota entre duas cidades, o resultado será a **quantidade de cidades** entre A e B e não o **total de quilômetros**, que provavelmente seria uma informação mais útil
 - Uma opção seria identificar  as rotas possíveis, somar as distâncias em cada rota e então selecionar a mais curta
 - Solução **muuuuuito custosa** e talvez inviável, precisará examinar uma exorbitante quantidade de possíveis rotas



Problema do Menor Caminho (PMC)

... estudar e praticar ... the best way for everything

Uma boa ideia seria modelar o mapa rodoviário como um grafo, onde os vértices representam as interseções (cidades), as arestas representam os trechos de estrada e o peso de cada aresta, a distância entre as localidades.



Algoritmo de Dijkstra

- Algoritmo guloso que resolve o PMC
 - Dado um grafo ponderado e direcionado $G = (V, A)$
 - Avalia a soma dos pesos das arestas, entre o nó inicial e os demais nós no grafo

peso $(u, v) \geq 0, \forall$ aresta $(u,v) \in A$

- O caminho mínimo em um grafo não pode conter ciclos
- Resumindo
 - Dado um vértice inicial, o algoritmo calcula a menor distância deste vértice a todos os demais, caso exista um caminho entre eles

Algoritmo de Dijkstra

- Excluindo o vértice inicial
 - Todos os vértices possuem um vértice antecessor contendo uma árvore de caminhos mais curtos
 - Esta árvore possui raiz em s , e é um subgrafo direcionado de G , $G'=(V', A')$, em que $V' \subseteq V$ e $A' \subseteq A$, tal que:
 - 1. V' é o conjunto de vértices alcançáveis a partir de $s \in G$
 - 2. G' forma uma árvore com raiz s
 - 3. \forall os vértices $v \in V'$, o caminho simples de s ate v é um caminho mais curto de s até v em G

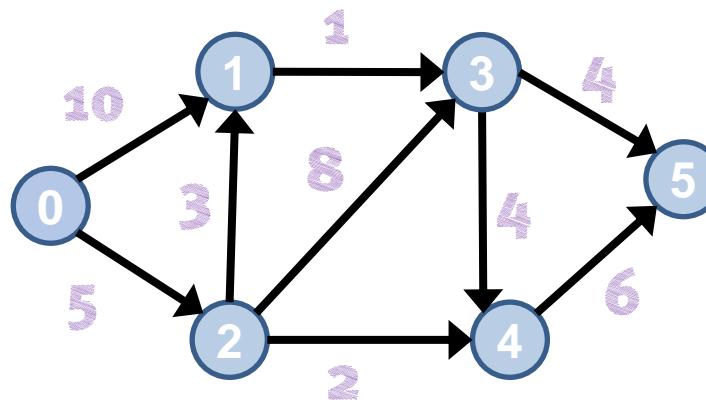
Algoritmo de Dijkstra

- Técnica de **Relaxamento**
 - Relaxar uma aresta (u, v) consiste em testar se é possível melhorar o caminho mínimo até v a partir de u
 - Caso positivo, atualizar $v.d$ e $v.pai$
 - d representa a estimativa de custo mínimo e pai é o vértice antecessor



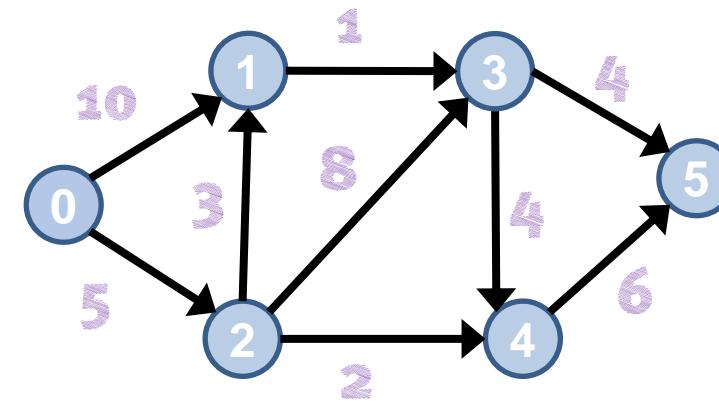
Algoritmo de Dijkstra

```
01 Inicializar algoritmo
02     vértice inicial = s
03     d[s] = 0 //distância conhecida
04     d[v] =  $\infty$ ,  $\forall v \neq s$  //super estimativa das distâncias
05     p[v] = -1,  $\forall v \in V$  //pai, antecessor do vértice
06     aberto[v] = true,  $\forall v \in V$  //inclusive o inicial
07
08
09
10
11
12
```



Algoritmo de Dijkstra

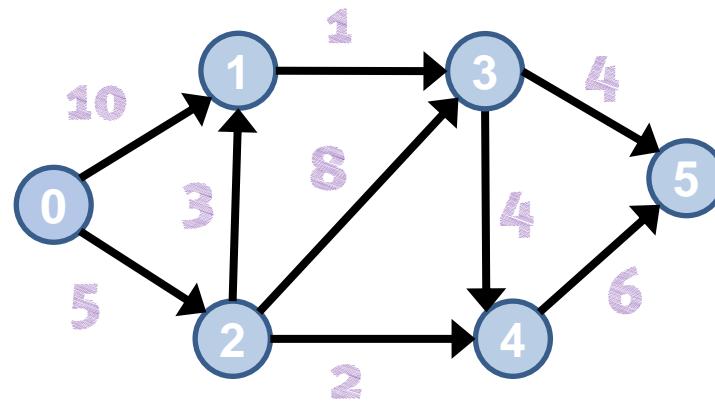
```
01 Inicializar algoritmo
02     vértice inicial = s
03     d[s] = 0 //distância conhecida
04     d[v] =  $\infty$ ,  $\forall v \neq s$  //super estimativa das distâncias
05     p[v] = -1,  $\forall v \in V$  //pai, antecessor do vértice
06     aberto[v] = true,  $\forall v \in V$  //inclusive o inicial
```



Inicialização >>

Algoritmo de Dijkstra

```
01 Inicializar algoritmo
02     vértice inicial = s
03     d[s] = 0 //distância conhecida
04     d[v] =  $\infty$ ,  $\forall v \neq s$  //super estimativa das distâncias
05     p[v] = -1,  $\forall v \in V$  //pai, antecessor do vértice
06     aberto[v] = true,  $\forall v \in V$  //inclusive o inicial
07
08
09
10
11
12
```

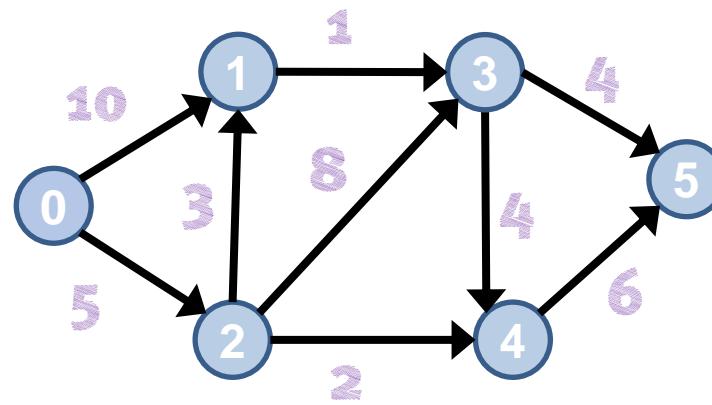


Inicialização >>>

V	Pai	Distância	Aberto
0	∅	0	T
1	∅	∞	T
2	∅	∞	T
3	∅	∞	T
4	∅	∞	T
5	∅	∞	T

Algoritmo de Dijkstra

```
01 Inicializar algoritmo
02     vértice inicial = s
03     d[s] = 0 //distância conhecida
04     d[v] =  $\infty$ ,  $\forall v \neq s$  //super estimativa das distâncias
05     p[v] = -1,  $\forall v \in V$  //pai, antecessor do vértice
06     aberto[v] = true,  $\forall v \in V$  //inclusive o inicial
07 Enquanto houver vértice aberto
08     Encontrar u com menor estimativa dentre os abertos (no grafo inteiro)
09         Fechar u //distância já identificada
10          $\forall v$ , adjacente a u AND Aberto
11             Relaxar aresta (u, v)
12 Fim enquanto
```



Inicialização >>>

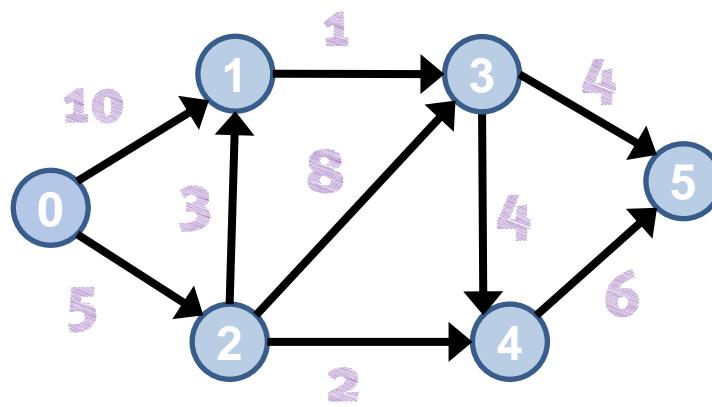
V	Pai	Distância	Aberto
0	∅	0	T
1	∅	∞	T
2	∅	∞	T
3	∅	∞	T
4	∅	∞	T
5	∅	∞	T

Algoritmo de Dijkstra

```

01 Inicializar algoritmo
02   vértice inicial = s
03   d[s] = 0 //distância conhecida
04   d[v] =  $\infty$ ,  $\forall v \neq s$  //super estimativa das distâncias
05   p[v] = -1,  $\forall v \in V$  //pai, antecessor do vértice
06   aberto[v] = true,  $\forall v \in V$  //inclusive o inicial
07 Enquanto houver vértice aberto
08   Encontrar u com menor estimativa dentre os abertos (no grafo inteiro)
09     Fechar u //distância já identificada
10      $\forall v$ , adjacente a u AND Aberto
11       Relaxar aresta (u, v)
12 Fim enquanto

```



Inicialização >>

```

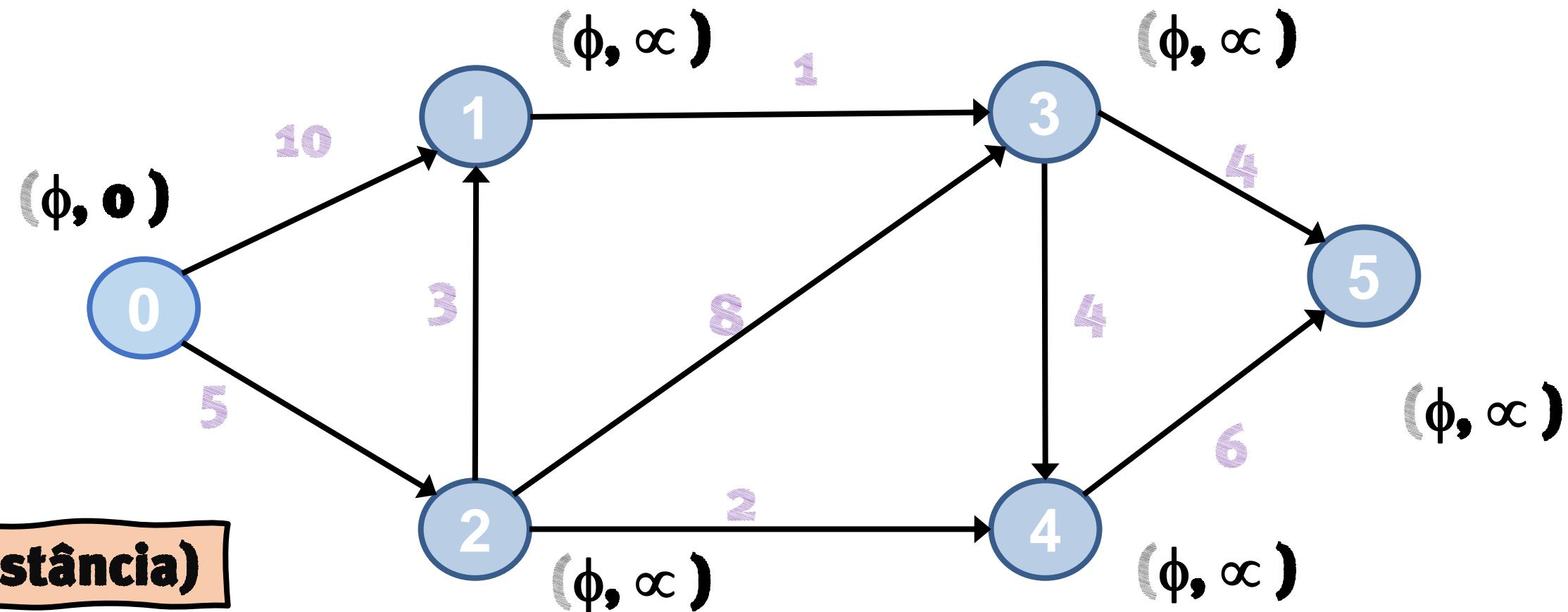
12 Relaxar (u, v){
13   se (v.d > u.d + peso(u,v)){
14     v.d = u.d + peso(u,v)
15     v.pai = u
16   }
  
```

V	Pai	Distância	Aberto
0	∅	0	T
1	∅	∞	T
2	∅	∞	T
3	∅	∞	T
4	∅	∞	T
5	∅	∞	T

Algoritmo de Dijkstra

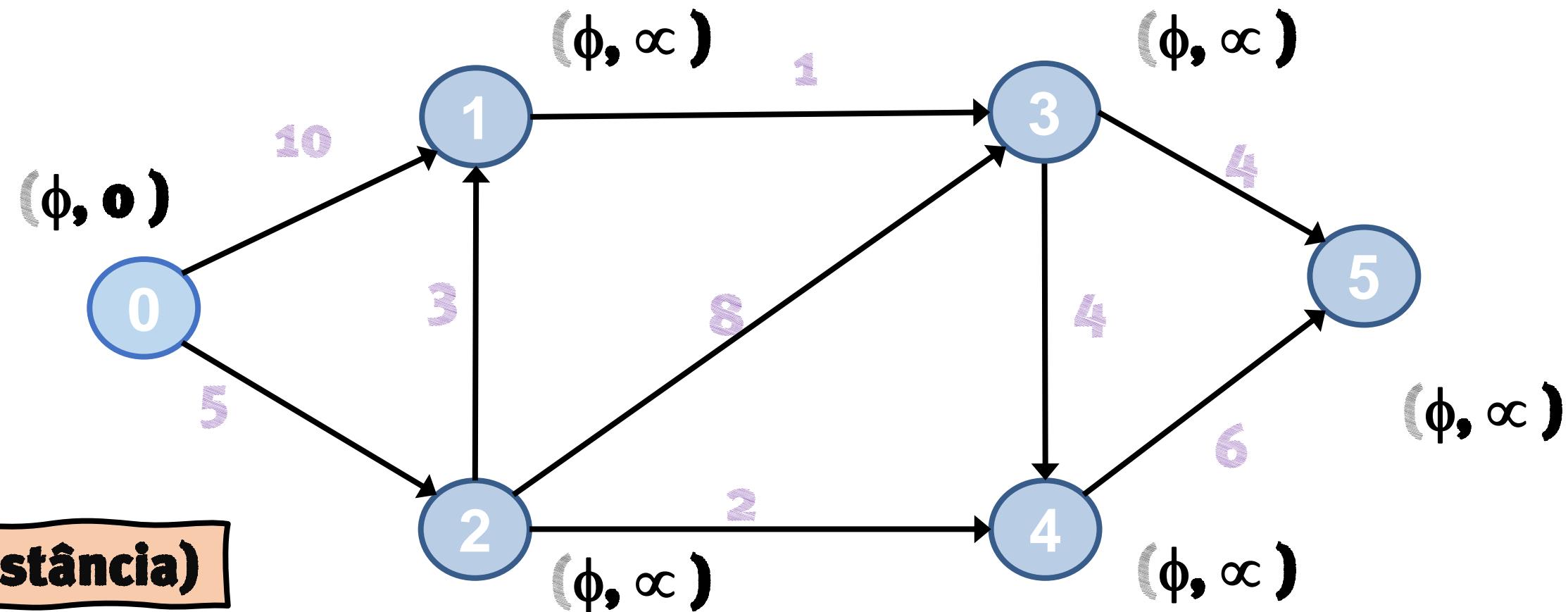
07
08
09
10
11

... estudar e praticar ... the best way for everything



Algoritmo de Dijkstra

07 | Enquanto houver vértice aberto
08
09
10
11



Algoritmo de Dijkstra

07 | Enquanto houver vértice aberto

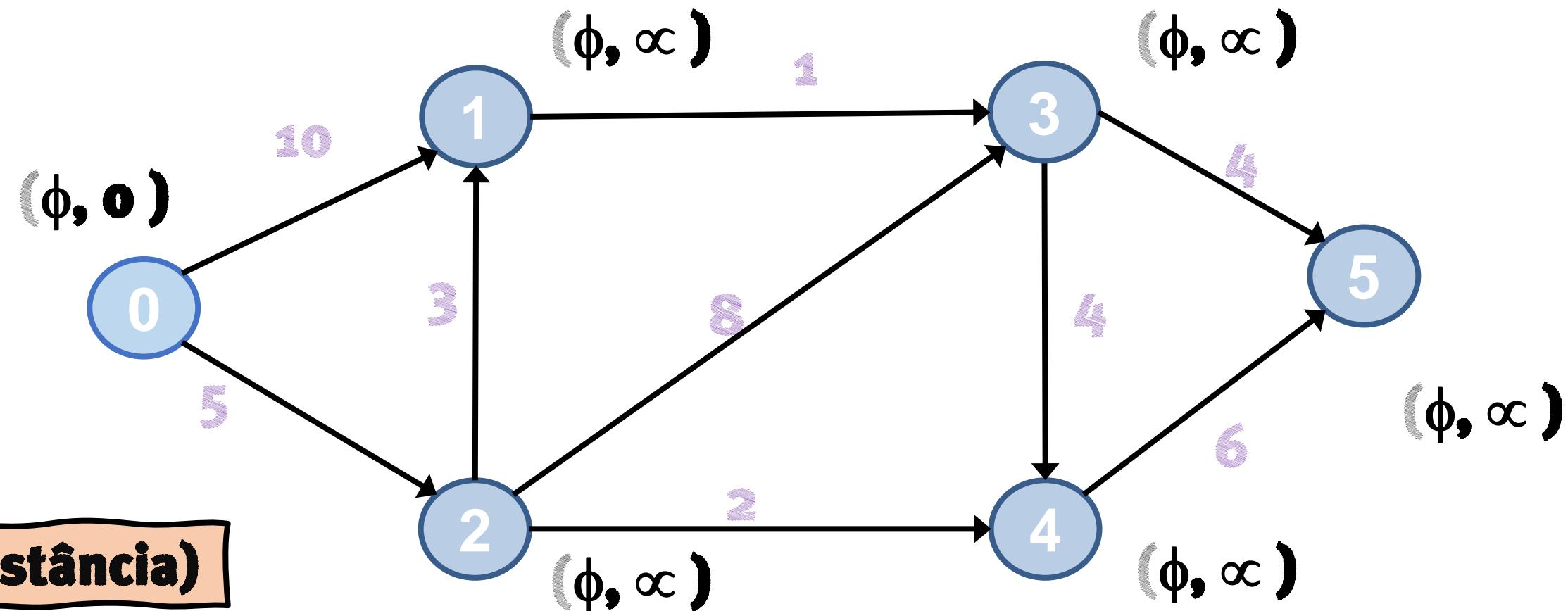
Encontrar **u** com menor estimativa dentre os abertos

08 |

09 |

10 |

11 |



Algoritmo de Dijkstra

07 | Enquanto houver vértice aberto

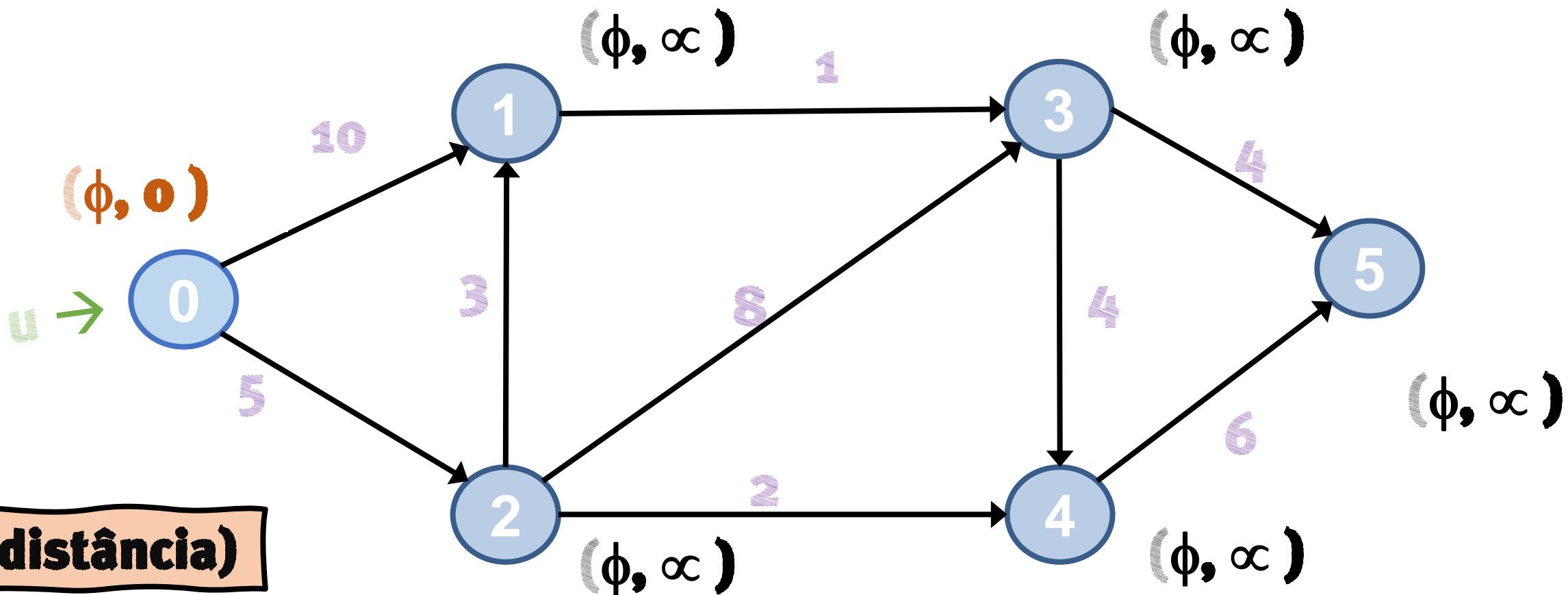
Encontrar **u** com menor estimativa dentre os abertos

08 |

09 |

10 |

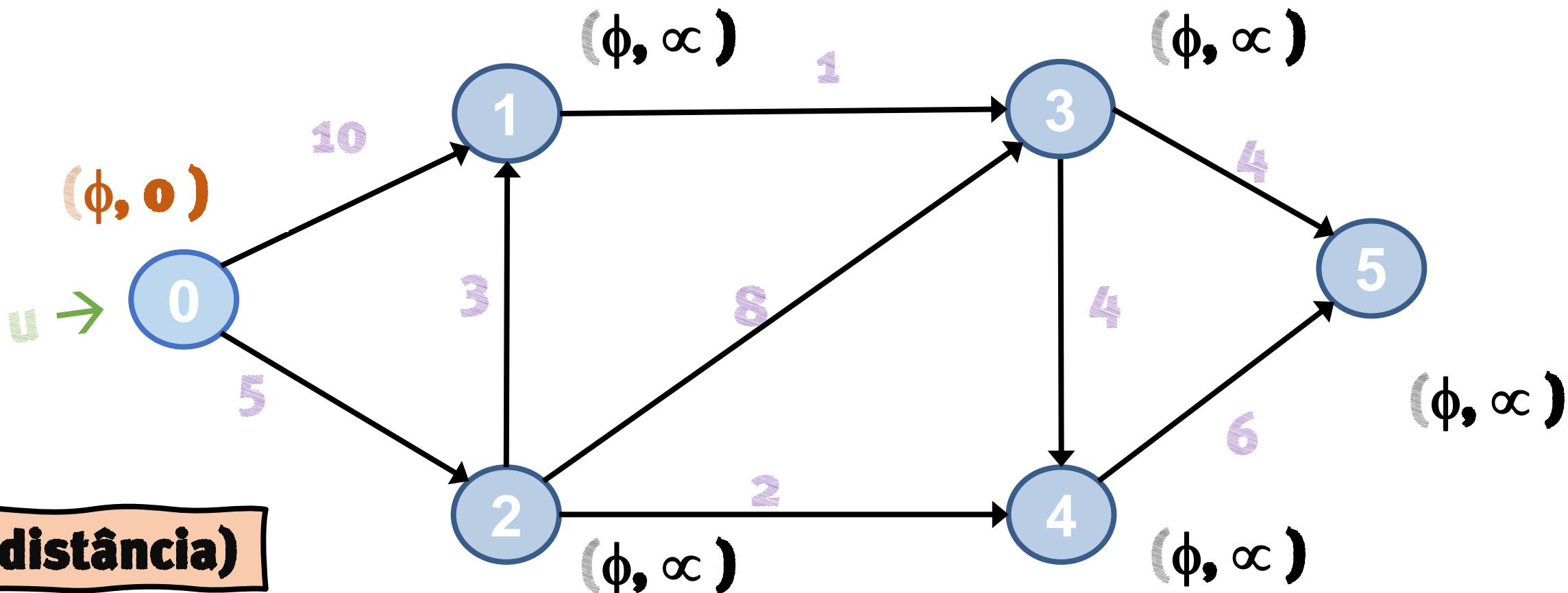
11 |



Algoritmo de Dijkstra

07 | Enquanto houver vértice aberto
08 | Encontrar **u** com menor estimativa dentre os abertos
09 | Fechar **u**

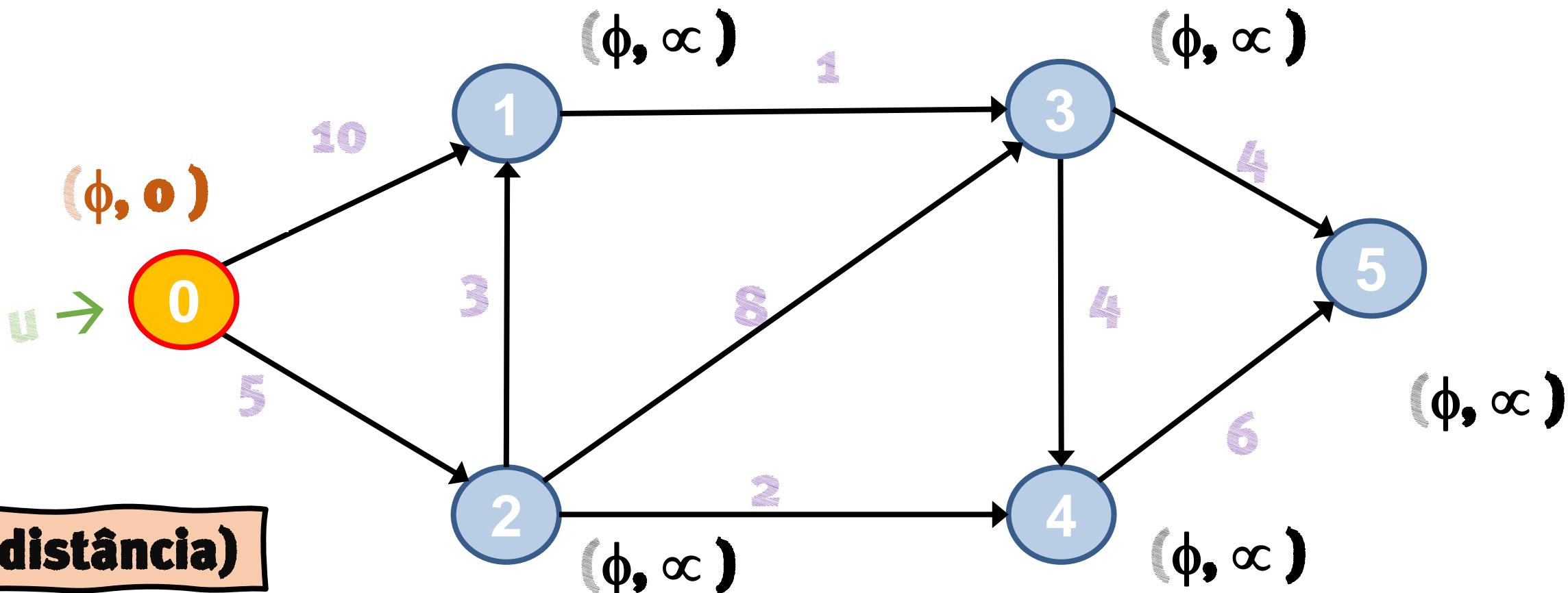
10
11



Algoritmo de Dijkstra

07 | Enquanto houver vértice aberto
08 | Encontrar **u** com menor estimativa dentre os abertos
09 | Fechar **u**

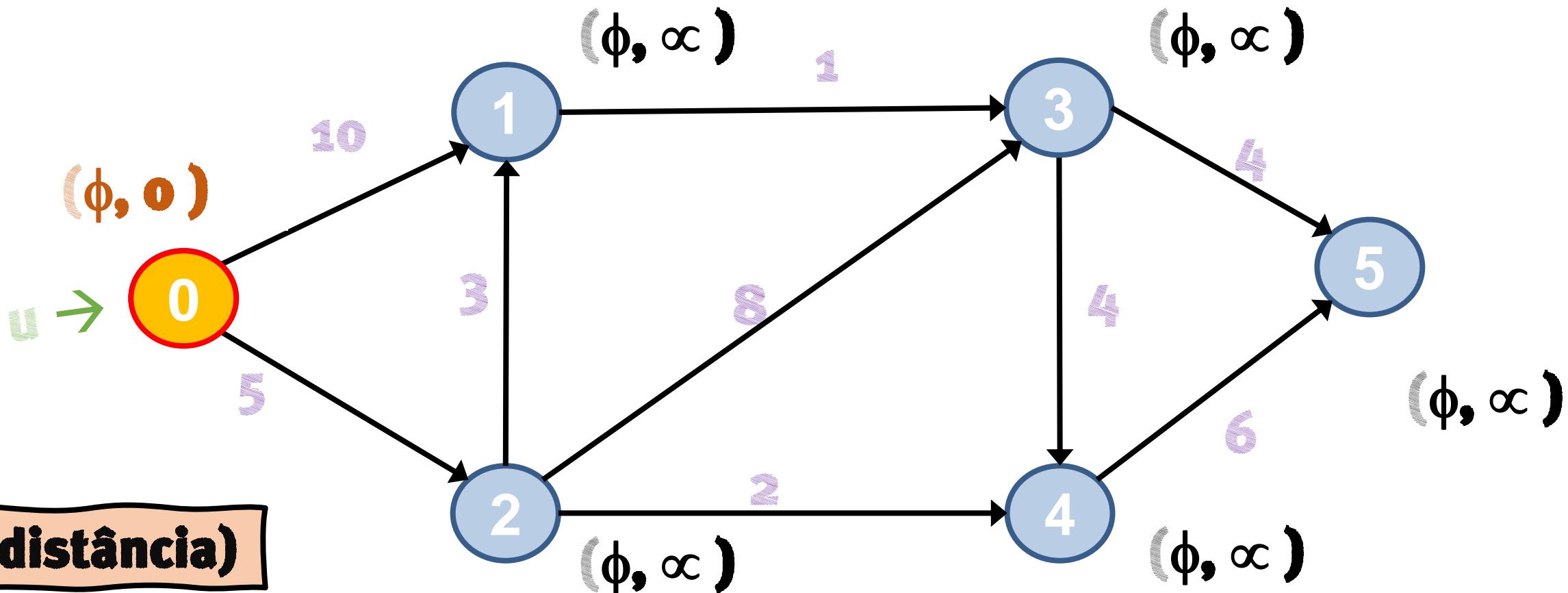
10
11



Algoritmo de Dijkstra

- 07 Enquanto houver vértice aberto
08 Encontrar **u** com menor estimativa dentre os abertos
09 Fechar **u**
10 $\forall v$, adjacente de **u** AND Aberto
11 Relaxar aresta (**u**, **v**)

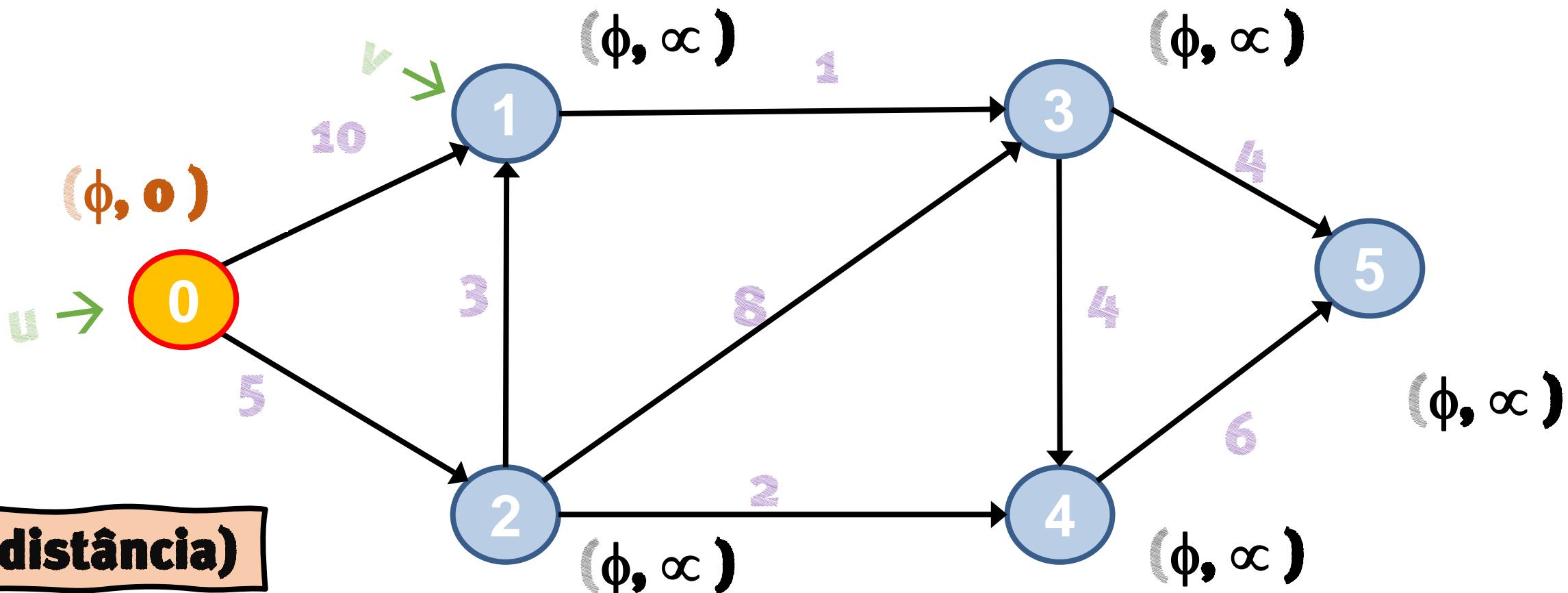
... estudar e praticar ... the best way for everything



Algoritmo de Dijkstra

07 Enquanto houver vértice aberto
08 Encontrar **u** com menor estimativa dentre os abertos
09 Fechar **u**
10 $\forall v$, adjacente de **u** AND Aberto
11 Relaxar aresta (**u**, **v**)

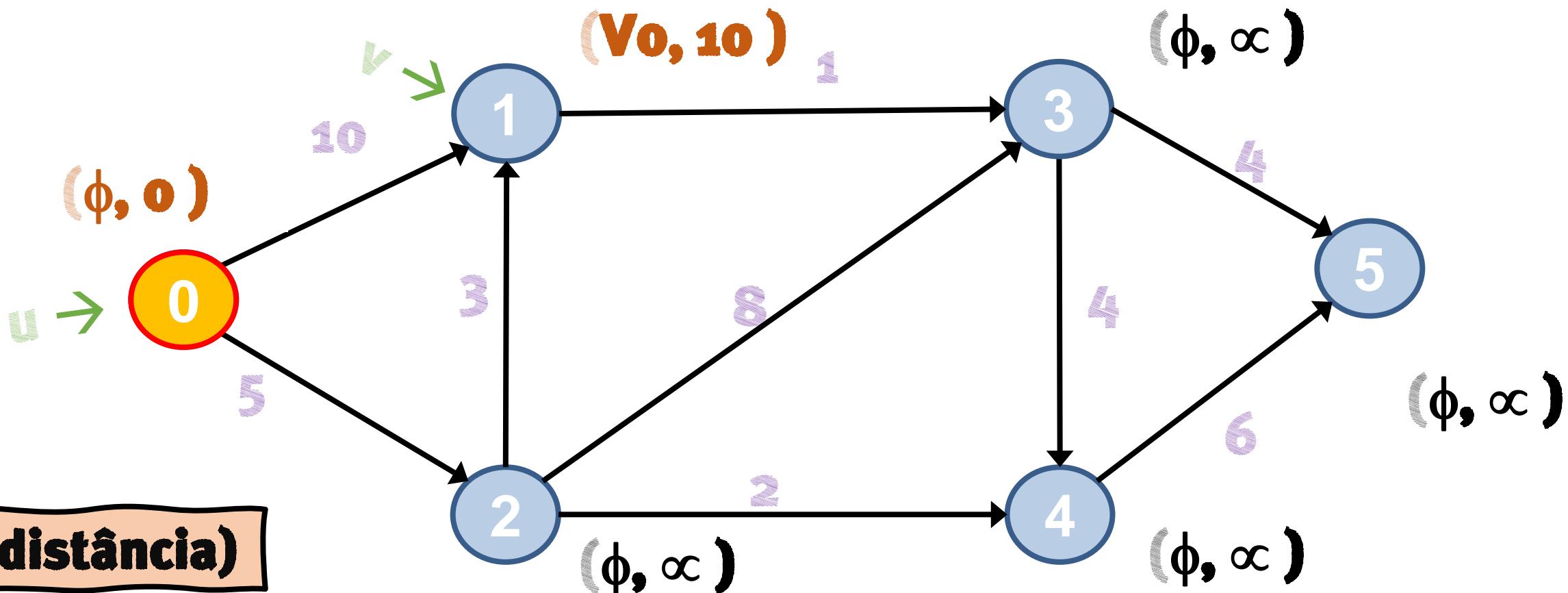
... estudar e praticar ... the best way for everything



Algoritmo de Dijkstra

07 | Enquanto houver vértice aberto
08 | Encontrar **u** com menor estimativa dentre os abertos
09 | Fechar **u**
10 | $\forall v$, adjacente de **u** AND Aberto
11 | Relaxar aresta (**u**, **v**)

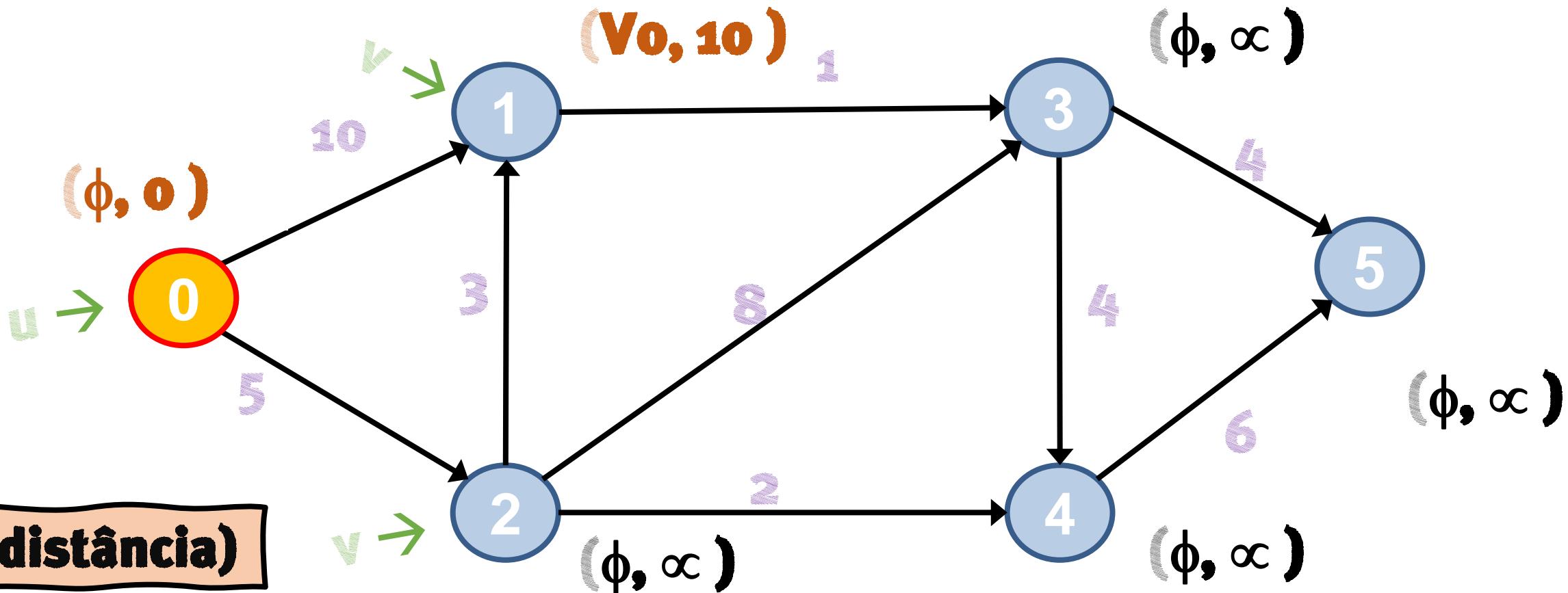
... estudar e praticar ... the best way for everything



Algoritmo de Dijkstra

07 Enquanto houver vértice aberto
08 Encontrar u com menor estimativa dentre os abertos
09 Fechar u
10 $\forall v$, adjacente de u AND Aberto
11 Relaxar aresta (u, v)

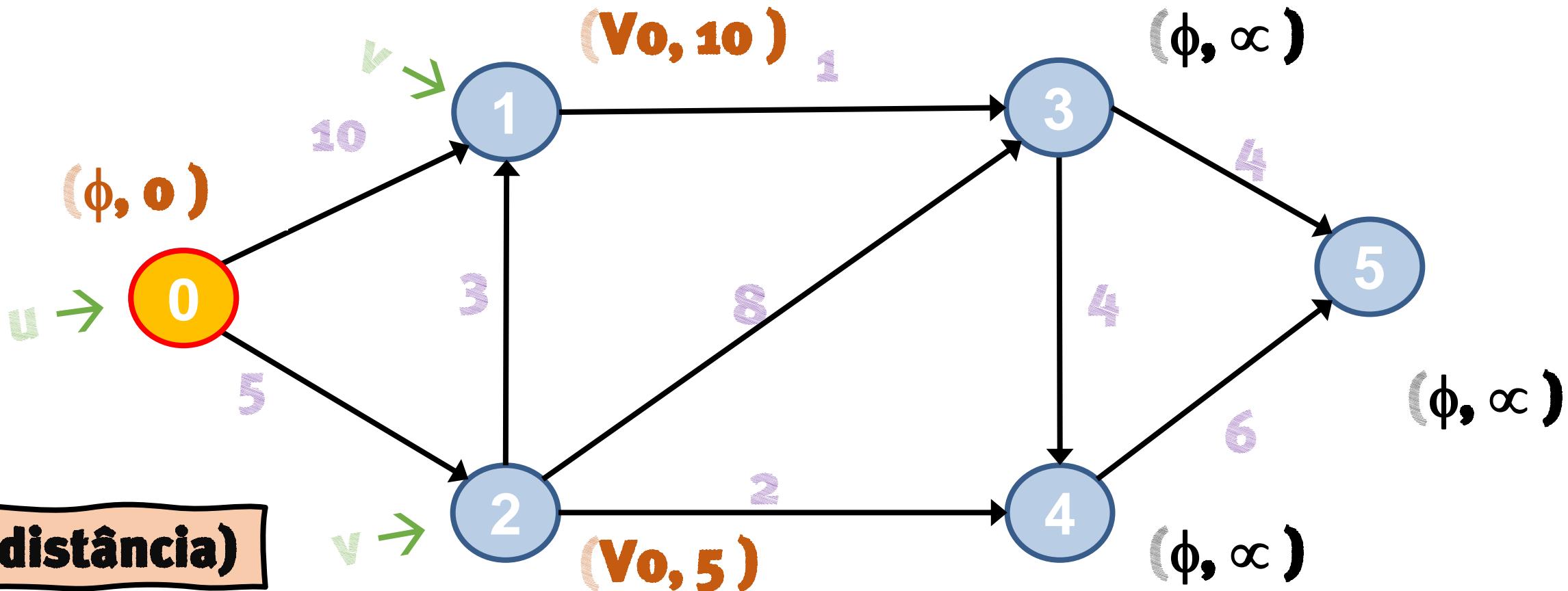
... estudar e praticar ... the best way for everything



Algoritmo de Dijkstra

07 Enquanto houver vértice aberto
08 Encontrar u com menor estimativa dentre os abertos
09 Fechar u
10 $\forall v$, adjacente de u AND Aberto
11 Relaxar aresta (u, v)

... estudar e praticar ... the best way for everything



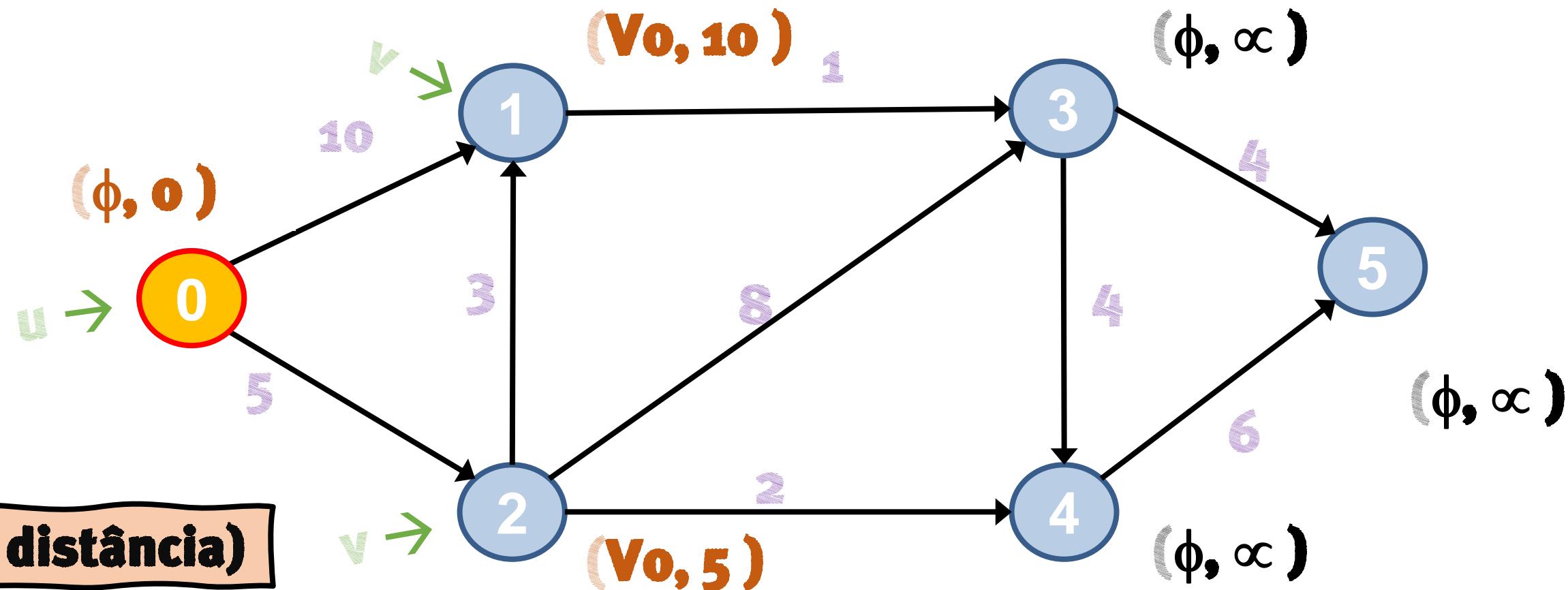
Algoritmo de Dijkstra

- 07 Enquanto houver vértice aberto
- 08 Encontrar **u** com menor estimativa dentre os abertos
- 09 Fchar **u**
- 10 $\forall v$, adjacente de **u** AND Aberto
- 11 Relaxar aresta **(u, v)**

```

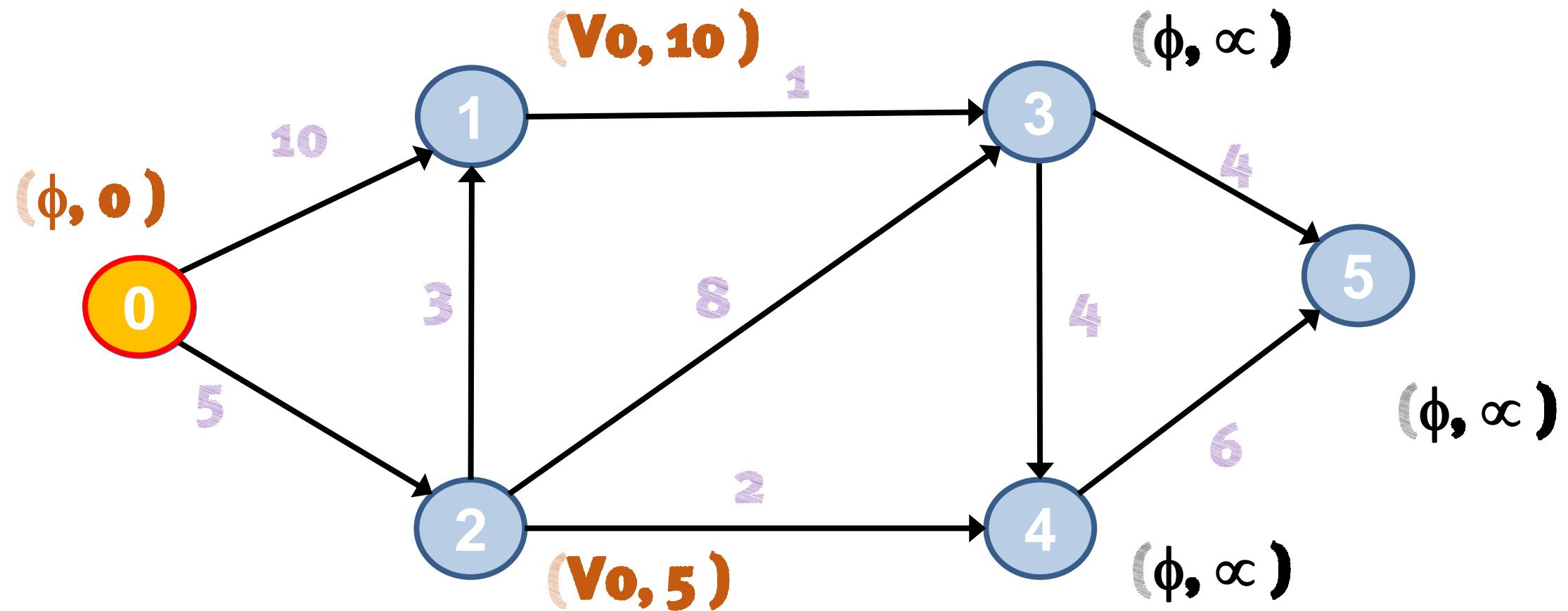
12 Relaxar (u, v){
13   se (v.d > u.d + peso(u,v)){
14     v.d = u.d + peso(u,v)
15     v.pai = u
16   }

```



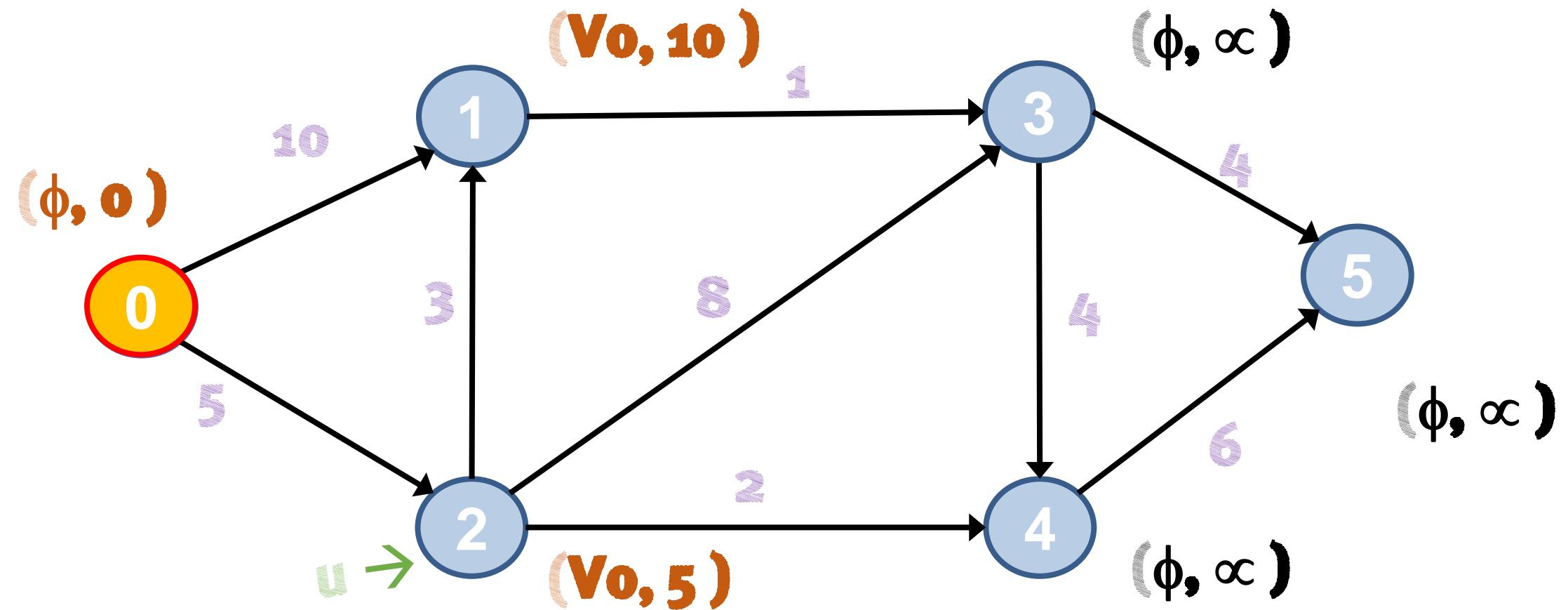
Algoritmo de Dijkstra

07 Enquanto houver vértice aberto
08 Encontrar **u** com menor estimativa dentre os abertos
09 Fechar **u**
10 $\forall v$, adjacente de **u** AND Aberto
11 Relaxar aresta (**u**, **v**)



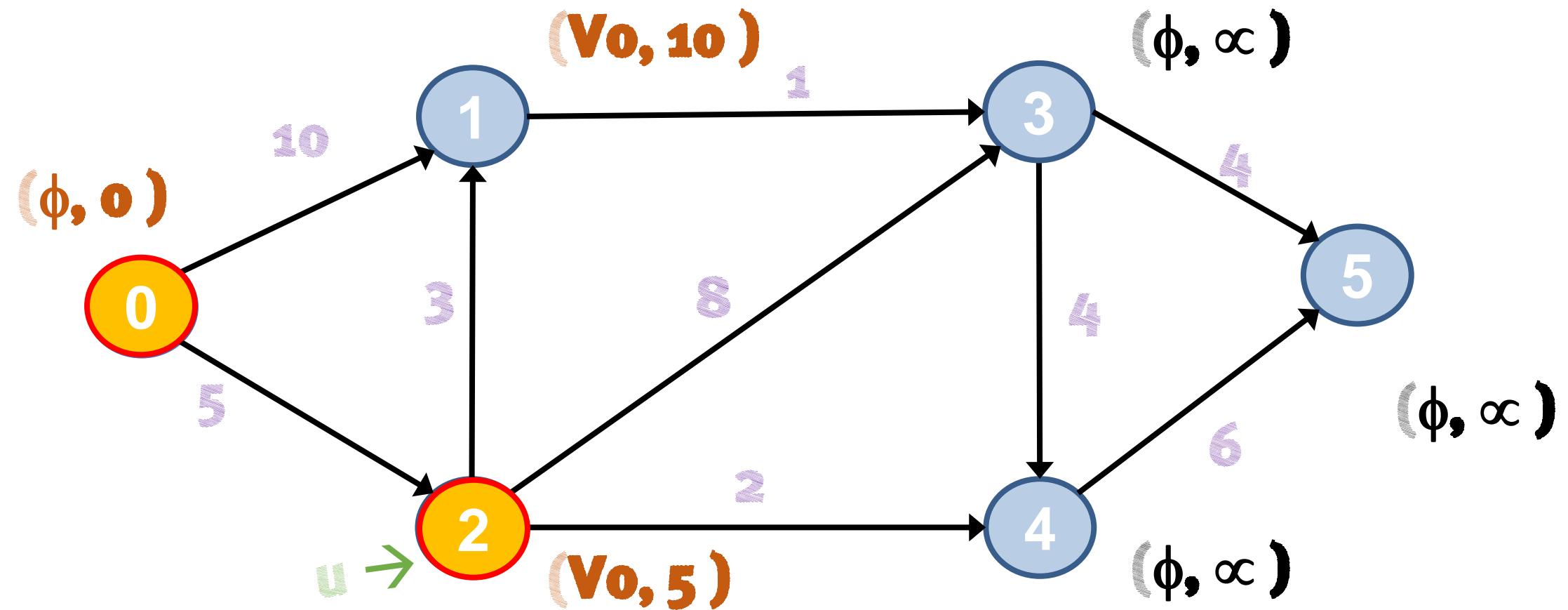
Algoritmo de Dijkstra

07 Enquanto houver vértice aberto
08 Encontrar **u** com menor estimativa dentre os abertos
09 Fechar **u**
10 $\forall v$, adjacente de **u** AND Aberto
11 Relaxar aresta (**u**, **v**)



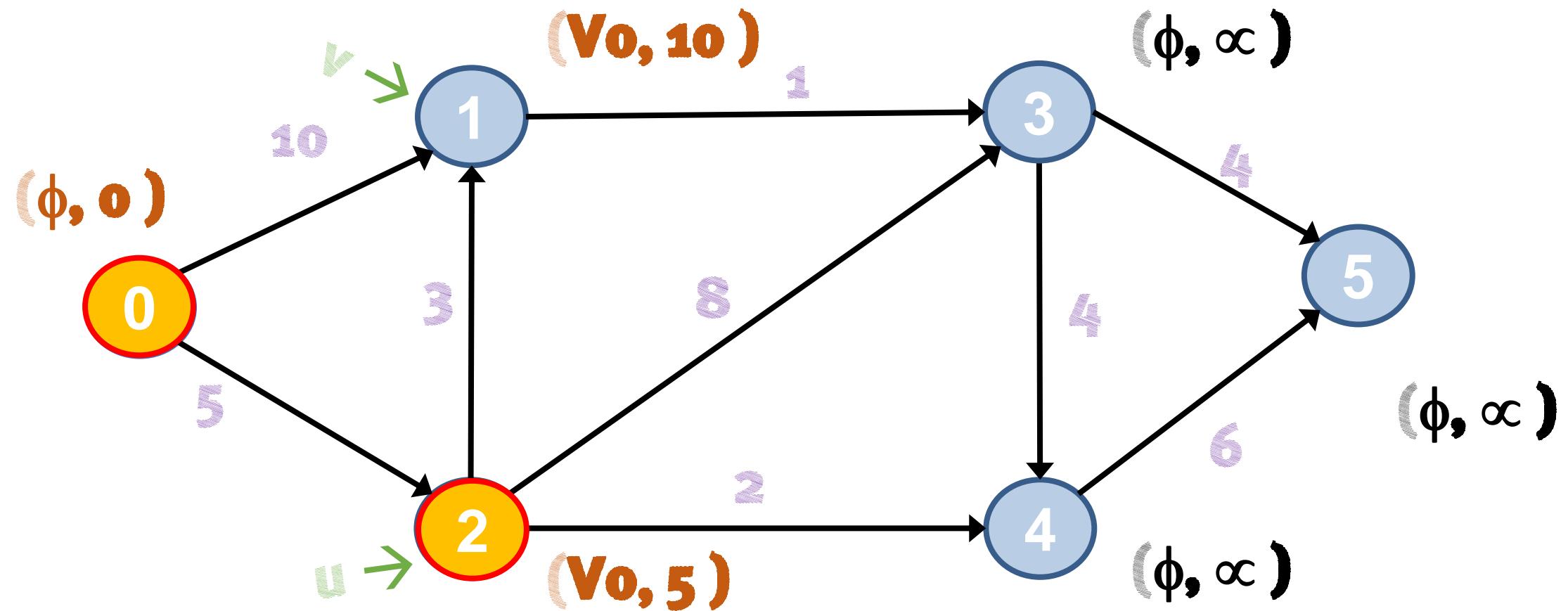
Algoritmo de Dijkstra

07 Enquanto houver vértice aberto
08 Encontrar **u** com menor estimativa dentre os abertos
09 Fechar **u**
10 $\forall v$, adjacente de **u** AND Aberto
11 Relaxar aresta (**u**, **v**)



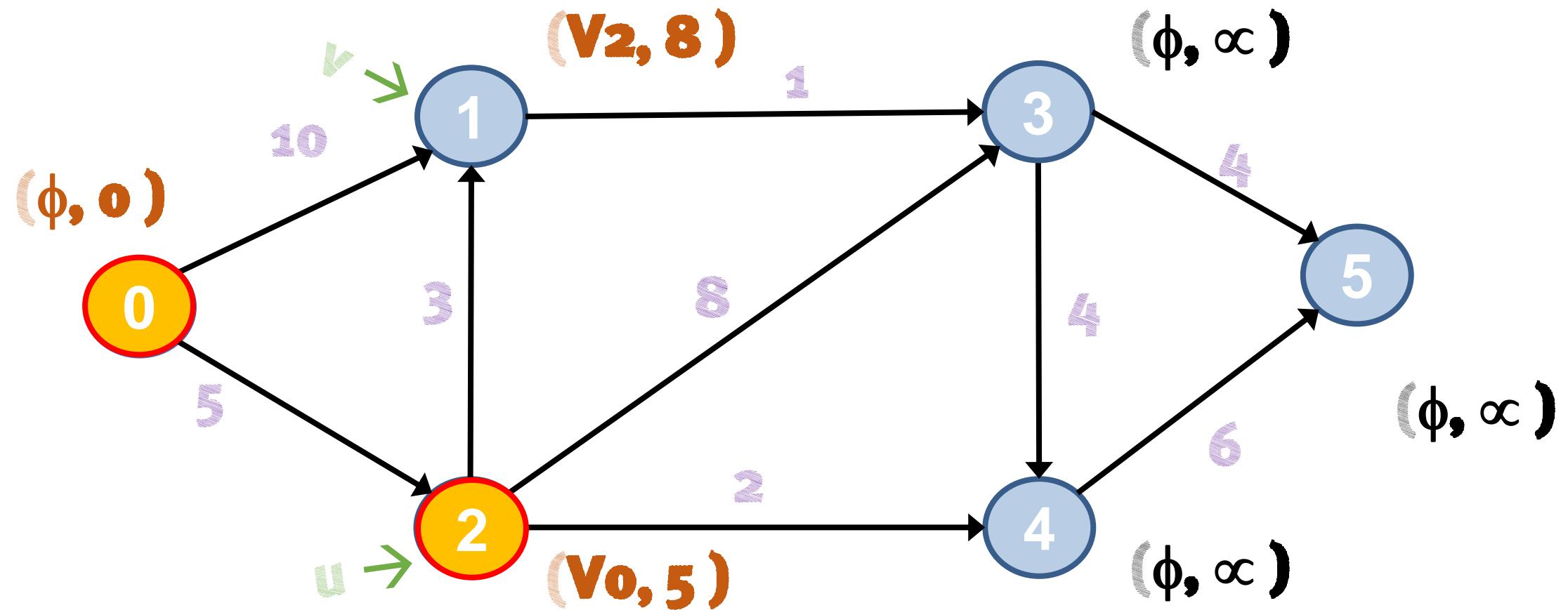
Algoritmo de Dijkstra

07 Enquanto houver vértice aberto
08 Encontrar **u** com menor estimativa dentre os abertos
09 Fechar **u**
10 $\forall v$, adjacente de **u** AND Aberto
11 Relaxar aresta (**u**, **v**)



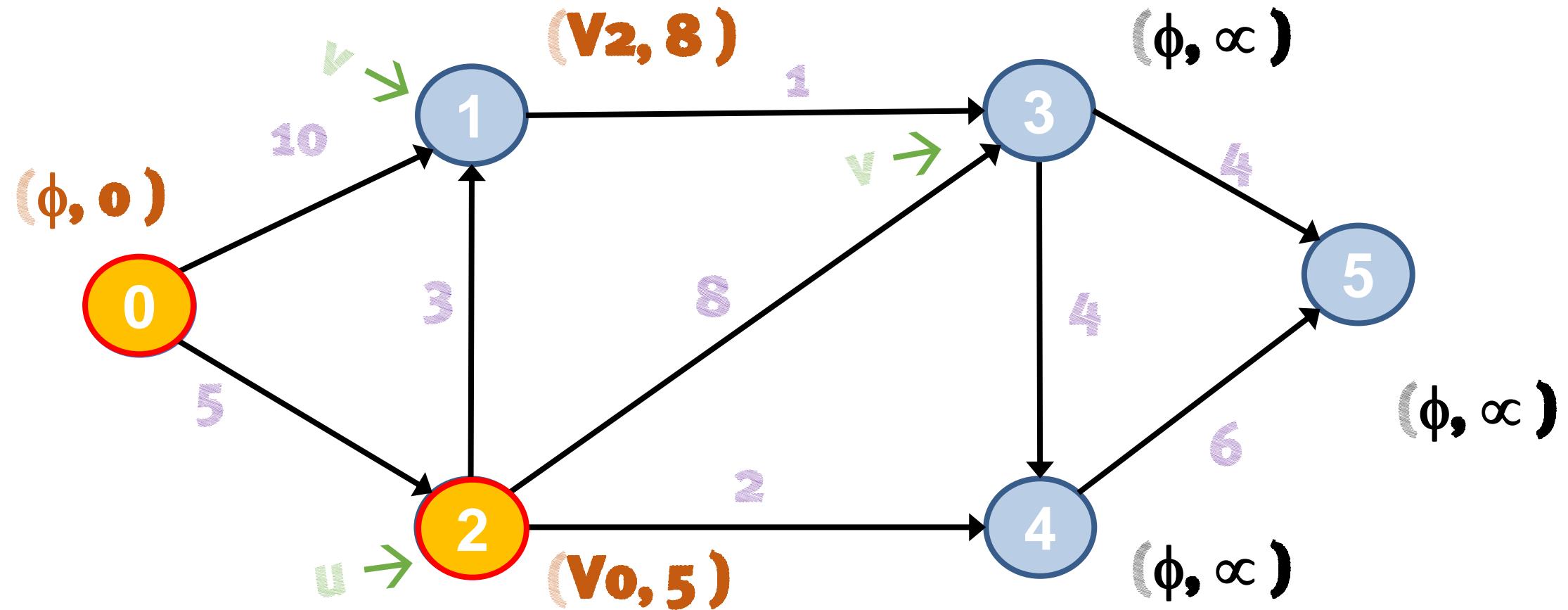
Algoritmo de Dijkstra

07 | Enquanto houver vértice aberto
08 | Encontrar **u** com menor estimativa dentre os abertos
09 | Fechar **u**
10 | $\forall v$, adjacente de **u** AND Aberto
11 | Relaxar aresta (**u**, **v**)



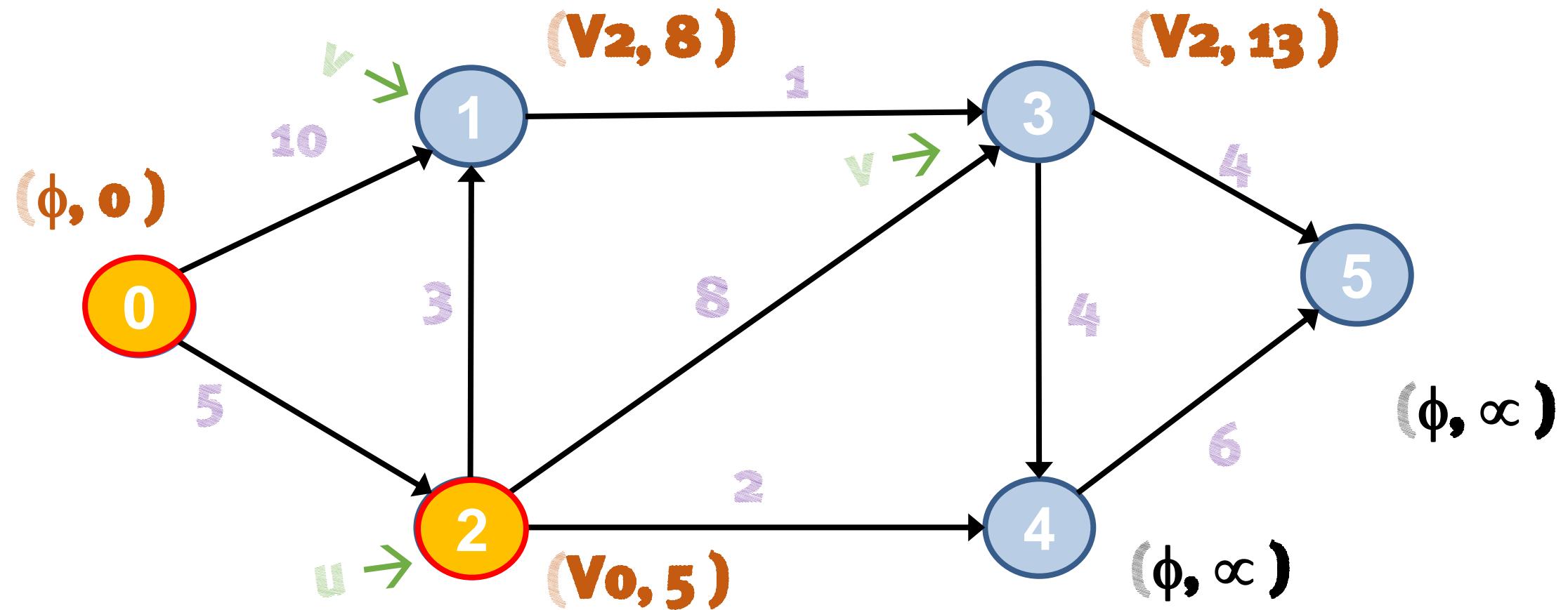
Algoritmo de Dijkstra

07 Enquanto houver vértice aberto
08 Encontrar **u** com menor estimativa dentre os abertos
09 Fechar **u**
10 $\forall v$, adjacente de **u** AND Aberto
11 Relaxar aresta (**u**, **v**)



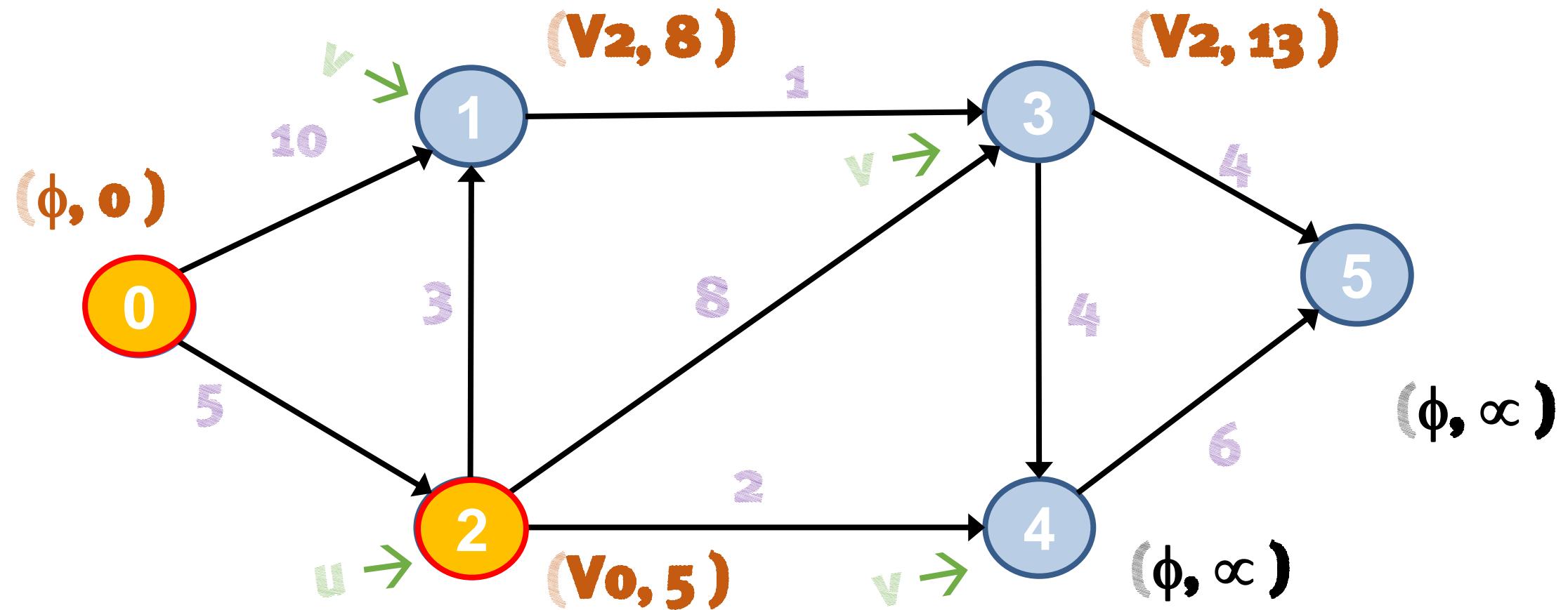
Algoritmo de Dijkstra

07 Enquanto houver vértice aberto
08 Encontrar **u** com menor estimativa dentre os abertos
09 Fechar **u**
10 $\forall v$, adjacente de **u** AND Aberto
11 Relaxar aresta (**u**, **v**)



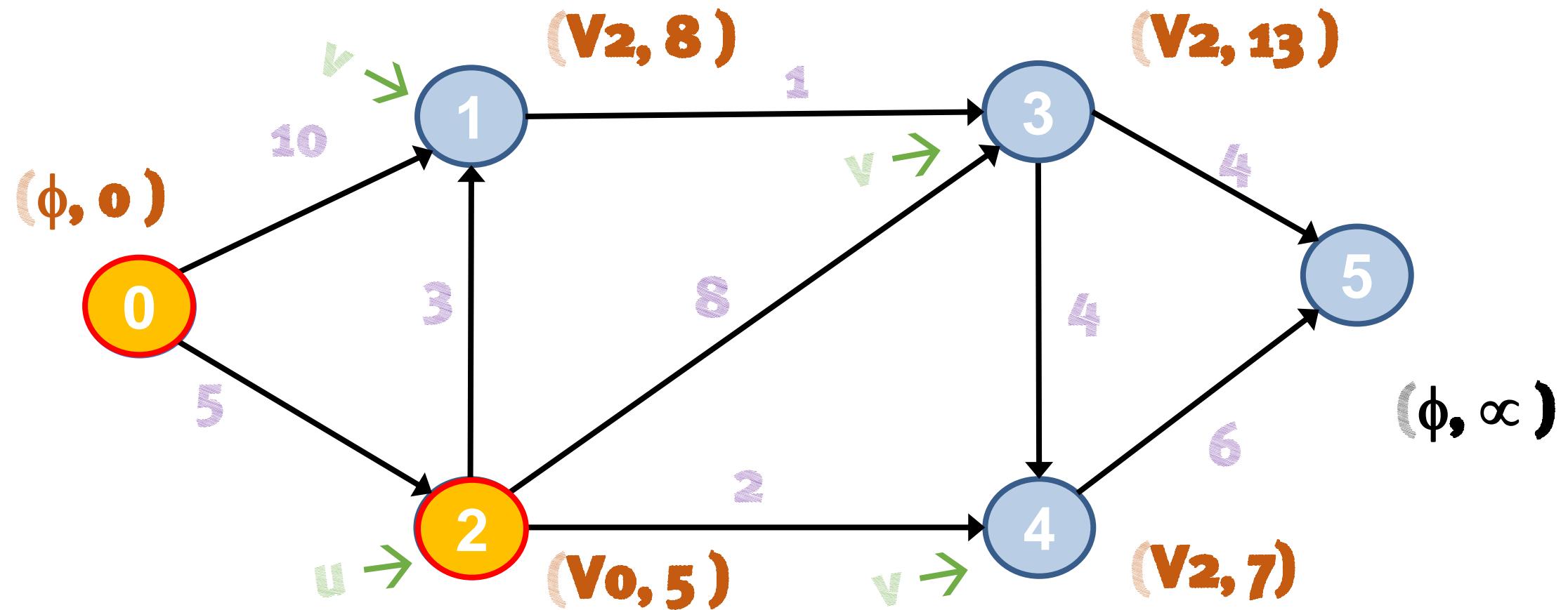
Algoritmo de Dijkstra

07 Enquanto houver vértice aberto
08 Encontrar **u** com menor estimativa dentre os abertos
09 Fechar **u**
10 $\forall v$, adjacente de **u** AND Aberto
11 Relaxar aresta (**u**, **v**)



Algoritmo de Dijkstra

07 Enquanto houver vértice aberto
08 Encontrar **u** com menor estimativa dentre os abertos
09 Fechar **u**
10 $\forall v$, adjacente de **u** AND Aberto
11 Relaxar aresta (**u**, **v**)

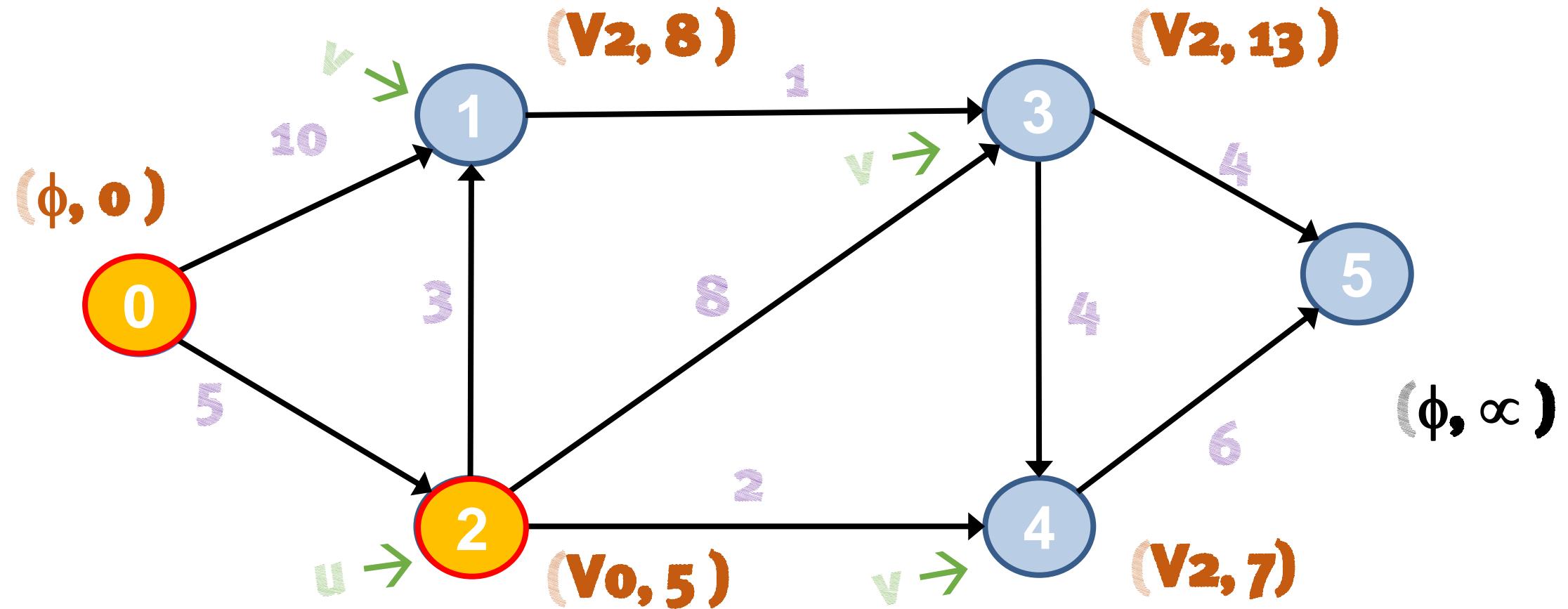


Algoritmo de Dijkstra

```

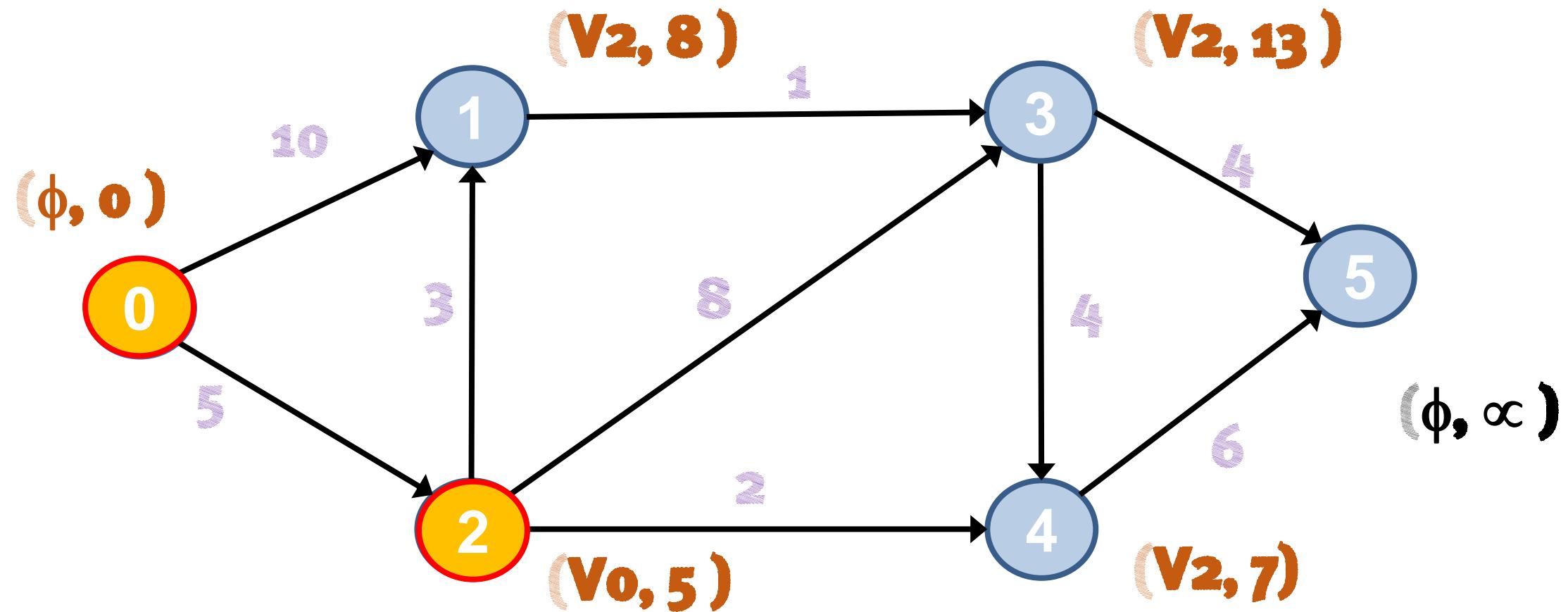
07 | Enquanto houver vértice aberto
08 |   Encontrar u com menor estimativa dentre os abertos
09 |   Fechar u
10 |    $\forall v$ , adjacente de u AND Aberto
11 |     Relaxar aresta (u, v)
  
```

12	Relaxar (u , v) {
13	se (v .d > u .d + peso(u , v)){
14	v .d = u .d + peso(u , v)
15	v .pai = u
16	}



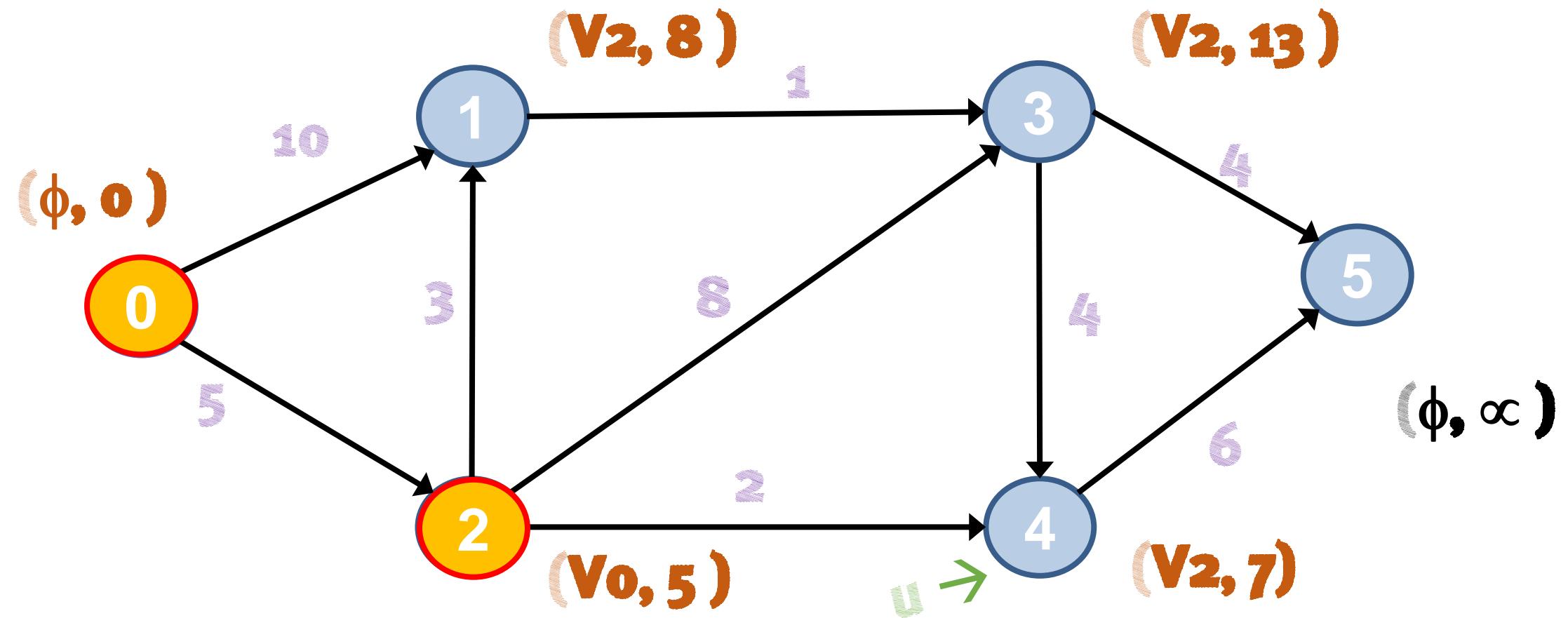
Algoritmo de Dijkstra

07 Enquanto houver vértice aberto
08 Encontrar **u** com menor estimativa dentre os abertos
09 Fechar **u**
10 $\forall v$, adjacente de **u** AND Aberto
11 Relaxar aresta (**u**, **v**)



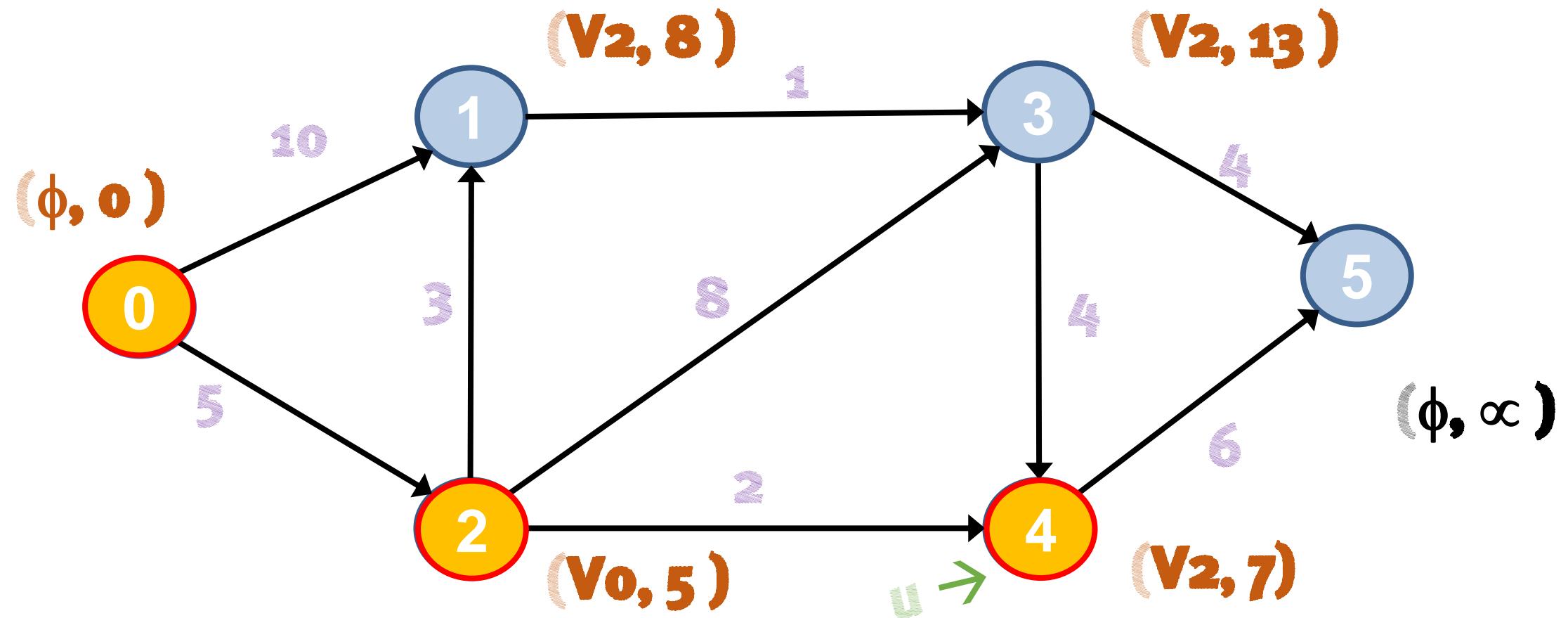
Algoritmo de Dijkstra

07 Enquanto houver vértice aberto
08 Encontrar **u** com menor estimativa dentre os abertos
09 Fechar **u**
10 $\forall v$, adjacente de **u** AND Aberto
11 Relaxar aresta (**u**, **v**)



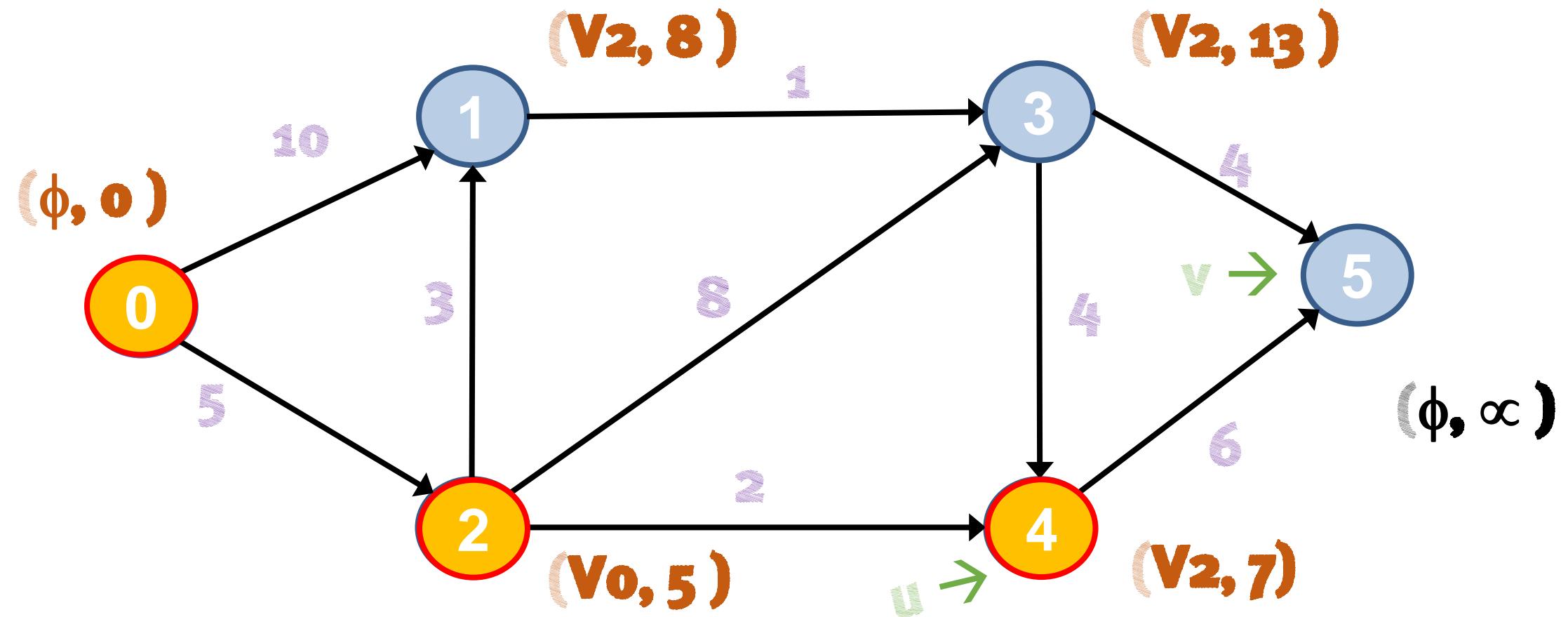
Algoritmo de Dijkstra

07 Enquanto houver vértice aberto
08 Encontrar **u** com menor estimativa dentre os abertos
09 Fechar **u**
10 $\forall v$, adjacente de **u** AND Aberto
11 Relaxar aresta (**u**, **v**)



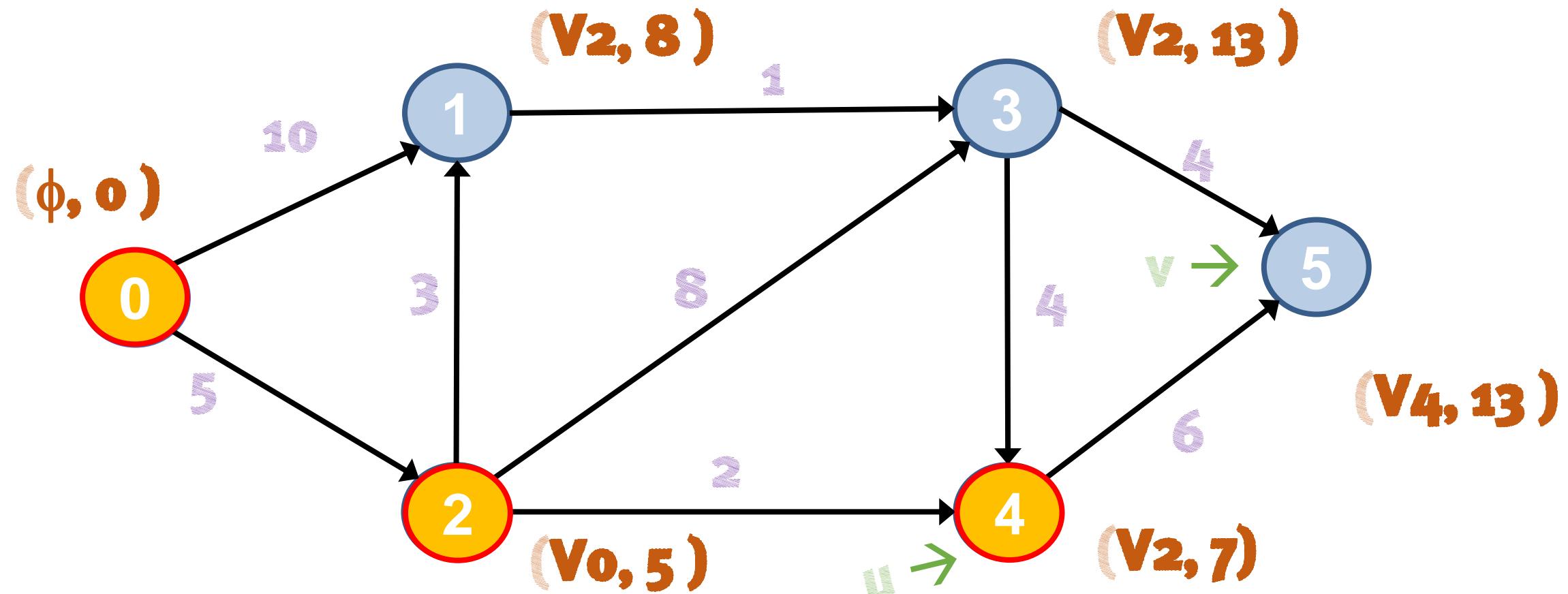
Algoritmo de Dijkstra

07 Enquanto houver vértice aberto
08 Encontrar **u** com menor estimativa dentre os abertos
09 Fechar **u**
10 $\forall v$, adjacente de **u** AND Aberto
11 Relaxar aresta (**u**, **v**)



Algoritmo de Dijkstra

07 | Enquanto houver vértice aberto
08 | Encontrar **u** com menor estimativa dentre os abertos
09 | Fechar **u**
10 | $\forall v$, adjacente de **u** AND Aberto
11 | Relaxar aresta (**u**, **v**)

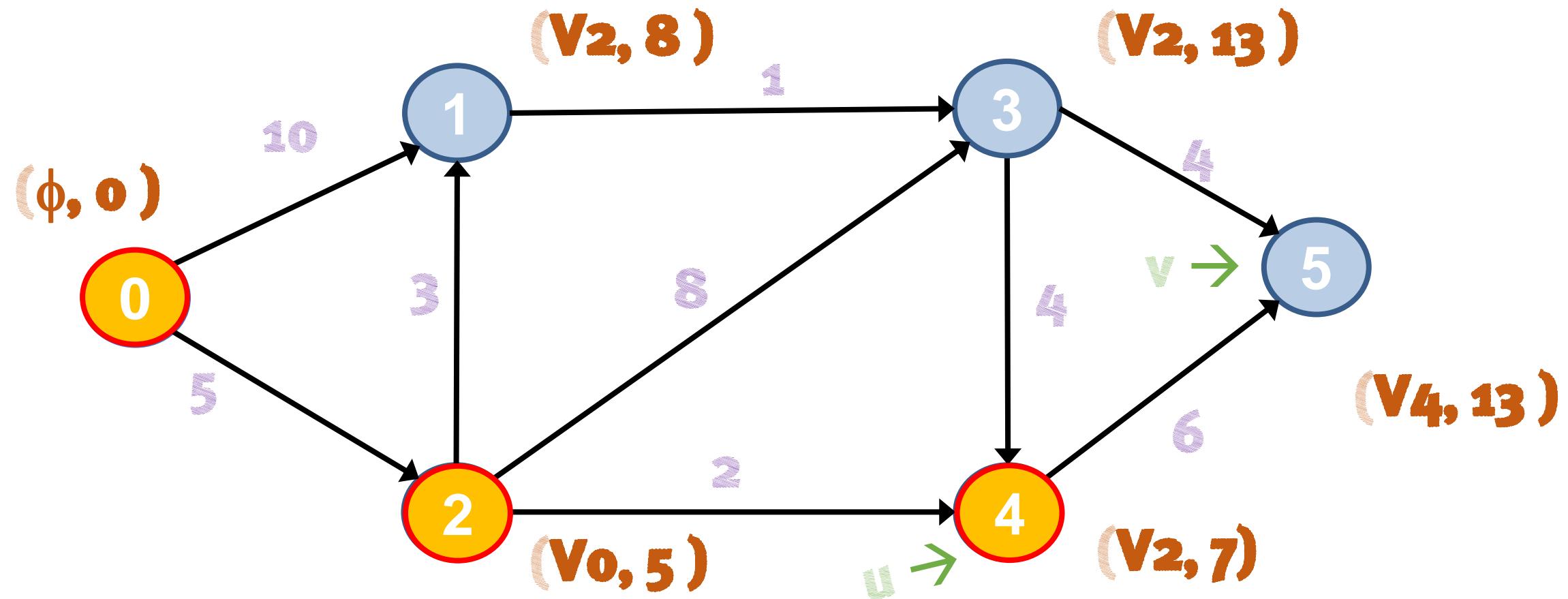


Algoritmo de Dijkstra

```

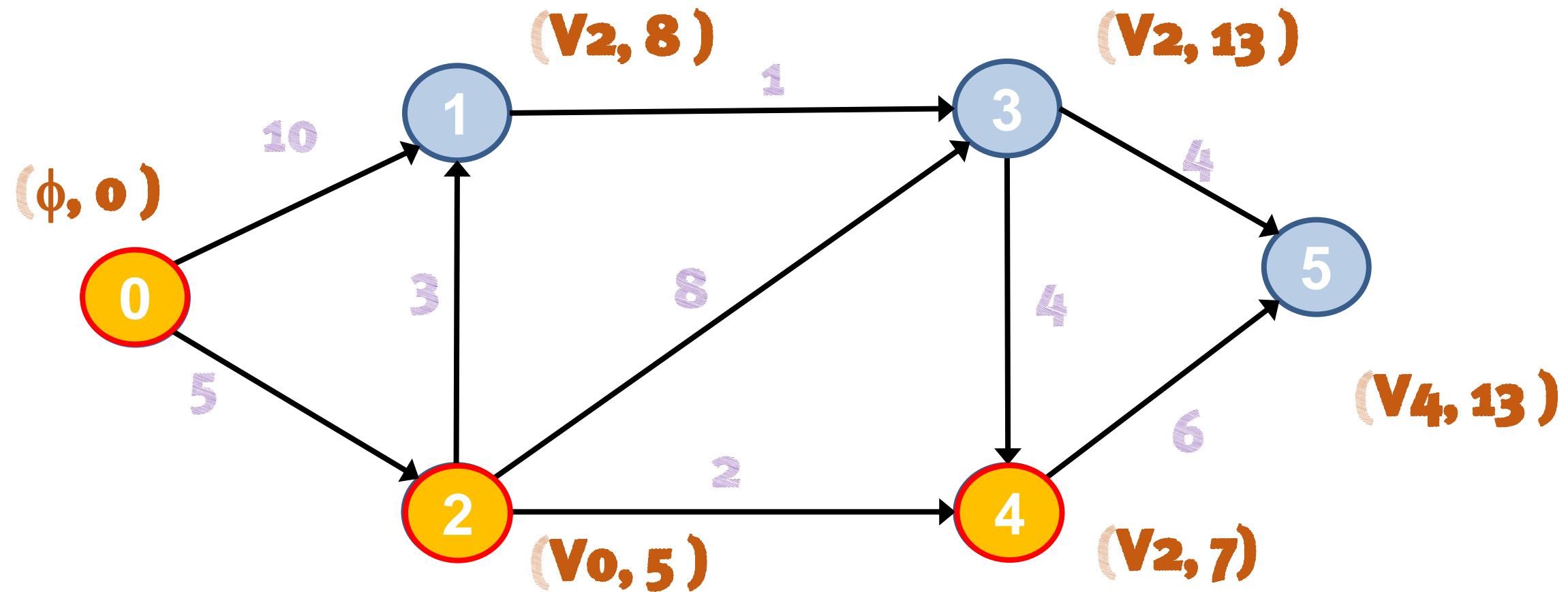
07 | Enquanto houver vértice aberto
08 |   Encontrar u com menor estimativa dentre os abertos
09 |   Fechar u
10 |    $\forall v$ , adjacente de u AND Aberto
11 |     Relaxar aresta (u, v)
  
```

12	Relaxar (u , v) {
13	se (v.d > u.d + peso(u , v)){
14	v.d = u.d + peso(u , v)
15	v.pai = u
16	}



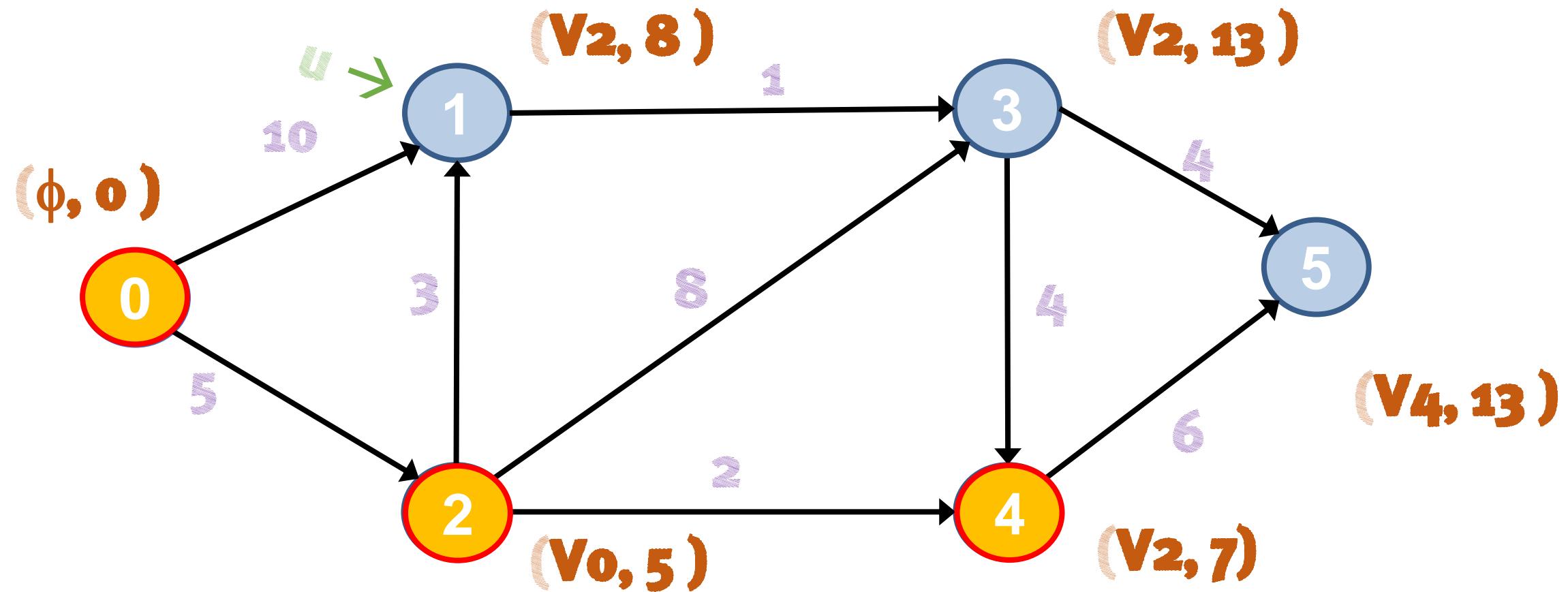
Algoritmo de Dijkstra

07 Enquanto houver vértice aberto
08 Encontrar **u** com menor estimativa dentre os abertos
09 Fechar **u**
10 $\forall v$, adjacente de **u** AND Aberto
11 Relaxar aresta (**u**, **v**)



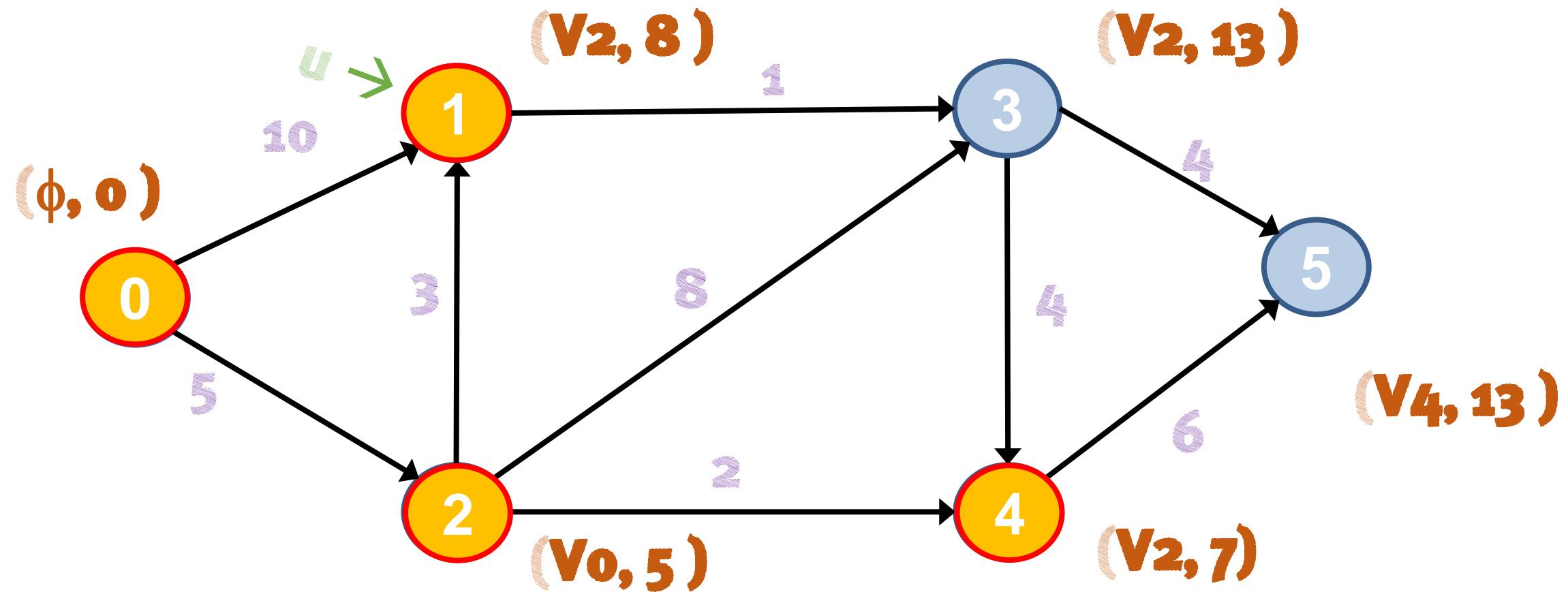
Algoritmo de Dijkstra

07 | Enquanto houver vértice aberto
08 | Encontrar **u** com menor estimativa dentre os abertos
09 | Fechar **u**
10 | $\forall v$, adjacente de **u** AND Aberto
11 | Relaxar aresta (**u**, **v**)



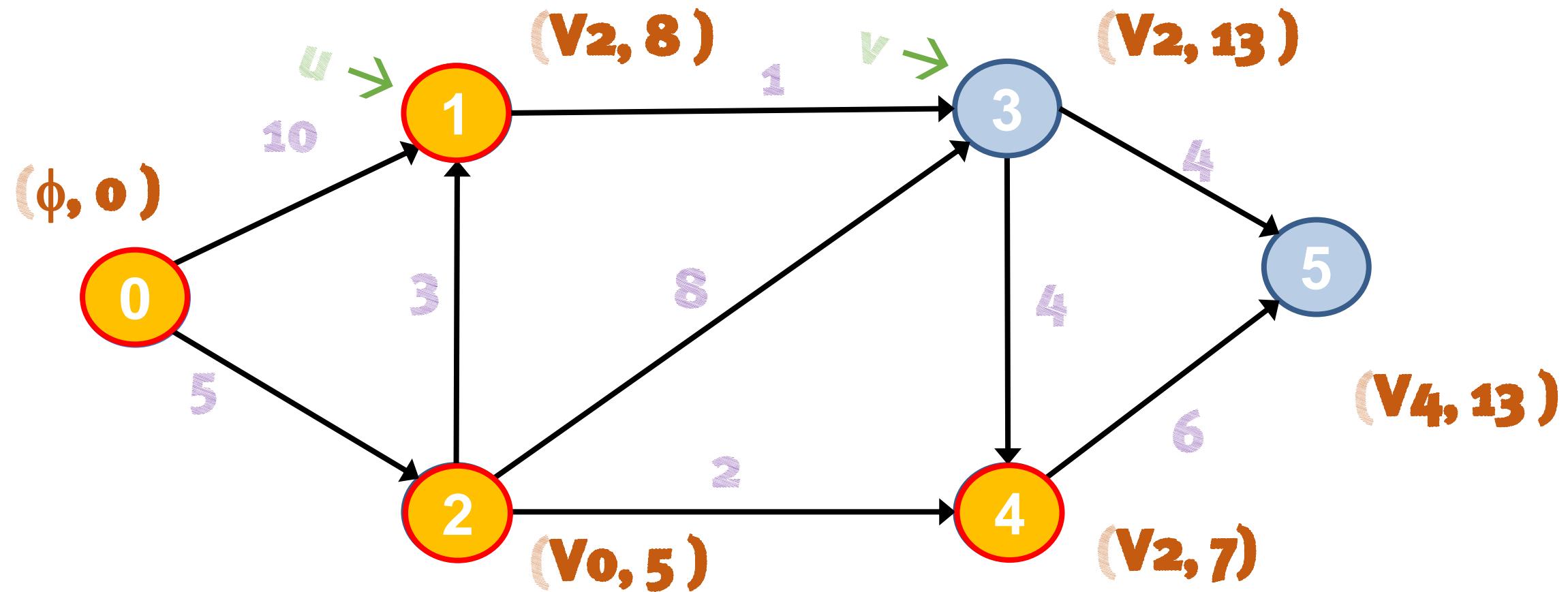
Algoritmo de Dijkstra

07 | Enquanto houver vértice aberto
08 | Encontrar **u** com menor estimativa dentre os abertos
09 | Fechar **u**
10 | $\forall v$, adjacente de **u** AND Aberto
11 | Relaxar aresta (**u**, **v**)



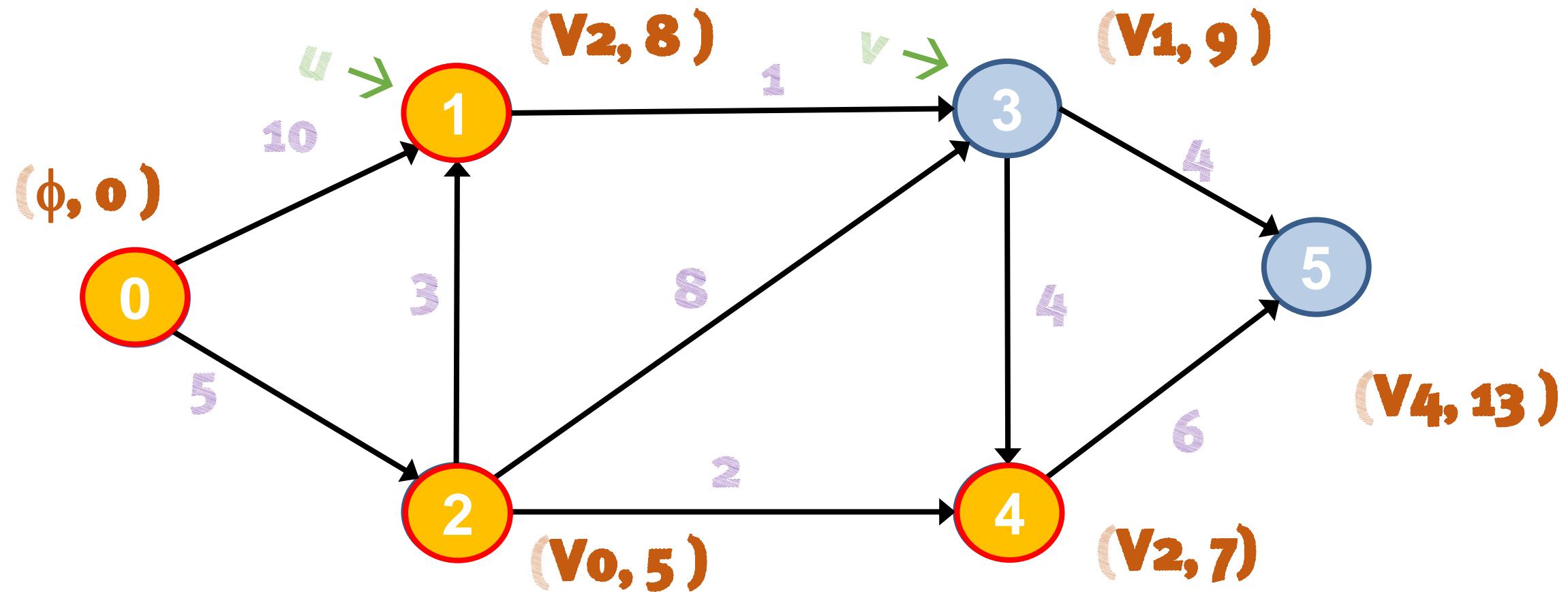
Algoritmo de Dijkstra

07 | Enquanto houver vértice aberto
08 | Encontrar **u** com menor estimativa dentre os abertos
09 | Fechar **u**
10 | $\forall v$, adjacente de **u** AND Aberto
11 | Relaxar aresta (**u**, **v**)



Algoritmo de Dijkstra

07 Enquanto houver vértice aberto
08 Encontrar **u** com menor estimativa dentre os abertos
09 Fechar **u**
10 $\forall v$, adjacente de **u** AND Aberto
11 Relaxar aresta (**u**, **v**)



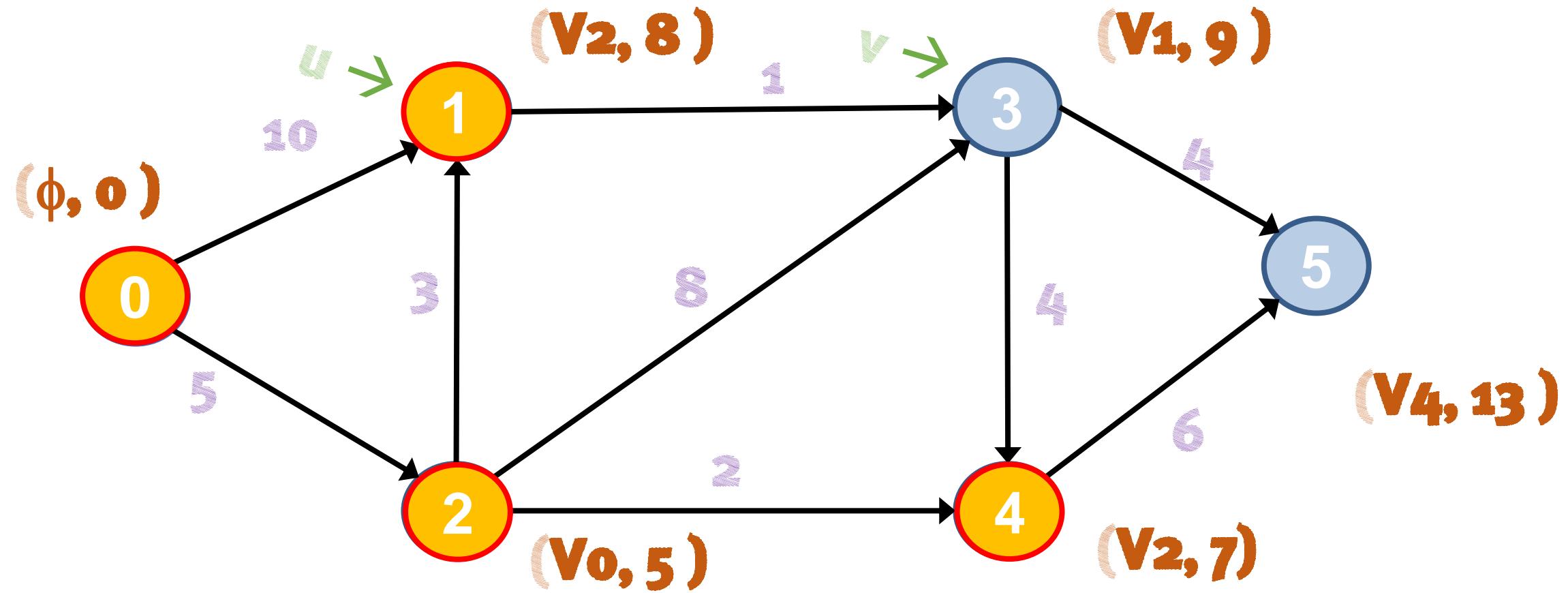
Algoritmo de Dijkstra

```

07 | Enquanto houver vértice aberto
08 |   Encontrar u com menor estimativa dentre os abertos
09 |   Fechar u
10 |    $\forall v$ , adjacente de u AND Aberto
11 |     Relaxar aresta (u, v)

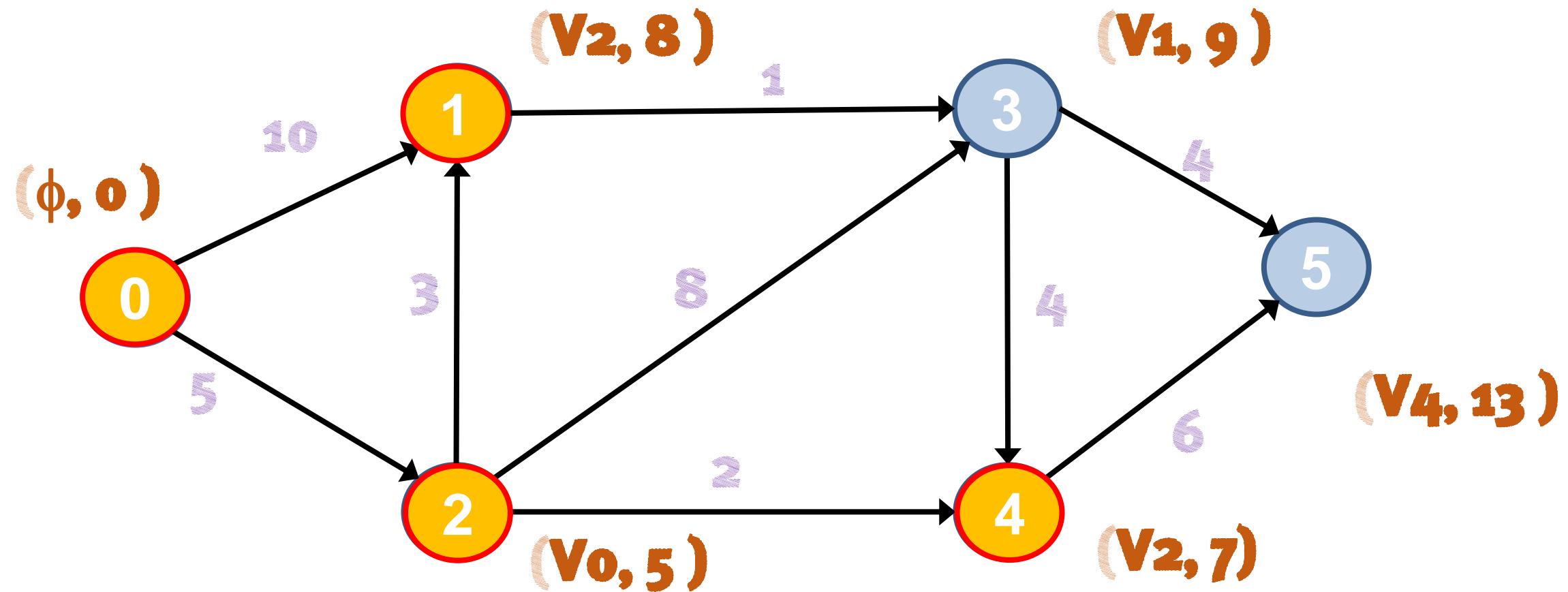
```

12	Relaxar (u , v){
13	se (v.d > u.d + peso(u , v)){
14	v.d = u.d + peso(u , v)
15	v.pai = u
16	}



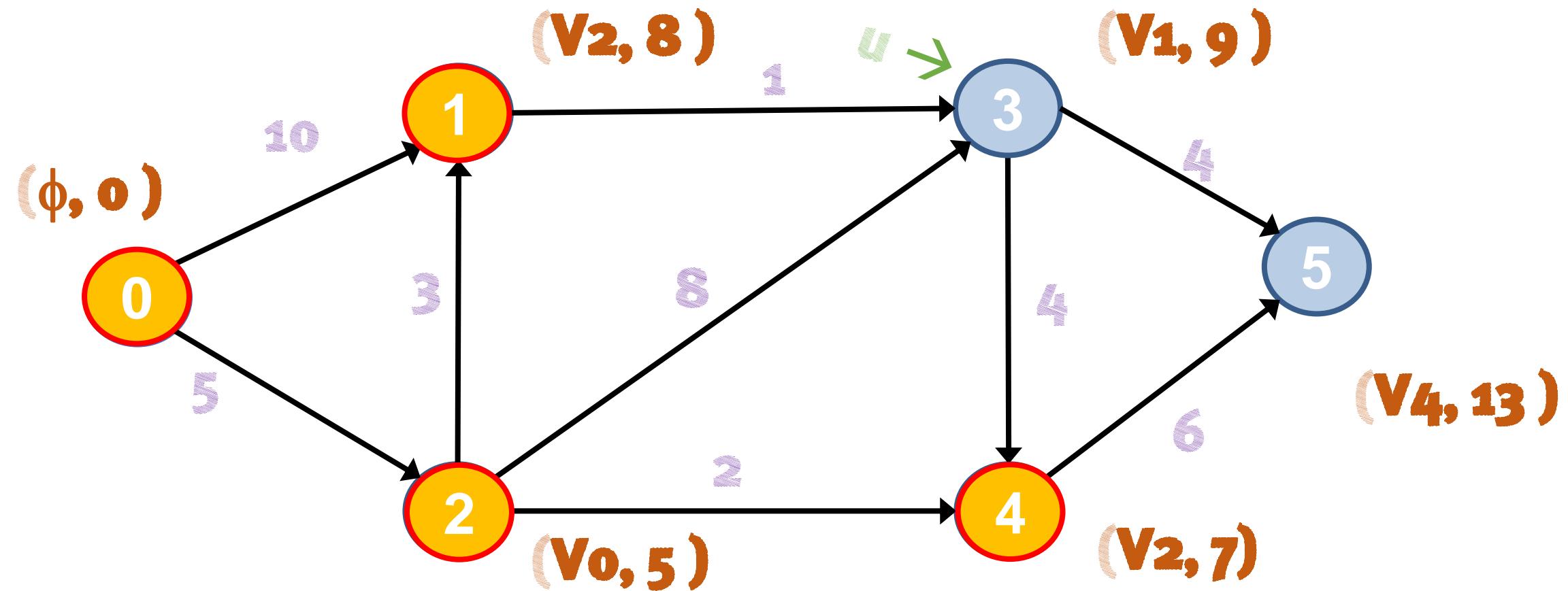
Algoritmo de Dijkstra

07 | Enquanto houver vértice aberto
08 | Encontrar **u** com menor estimativa dentre os abertos
09 | Fechar **u**
10 | $\forall v$, adjacente de **u** AND Aberto
11 | Relaxar aresta (**u**, **v**)



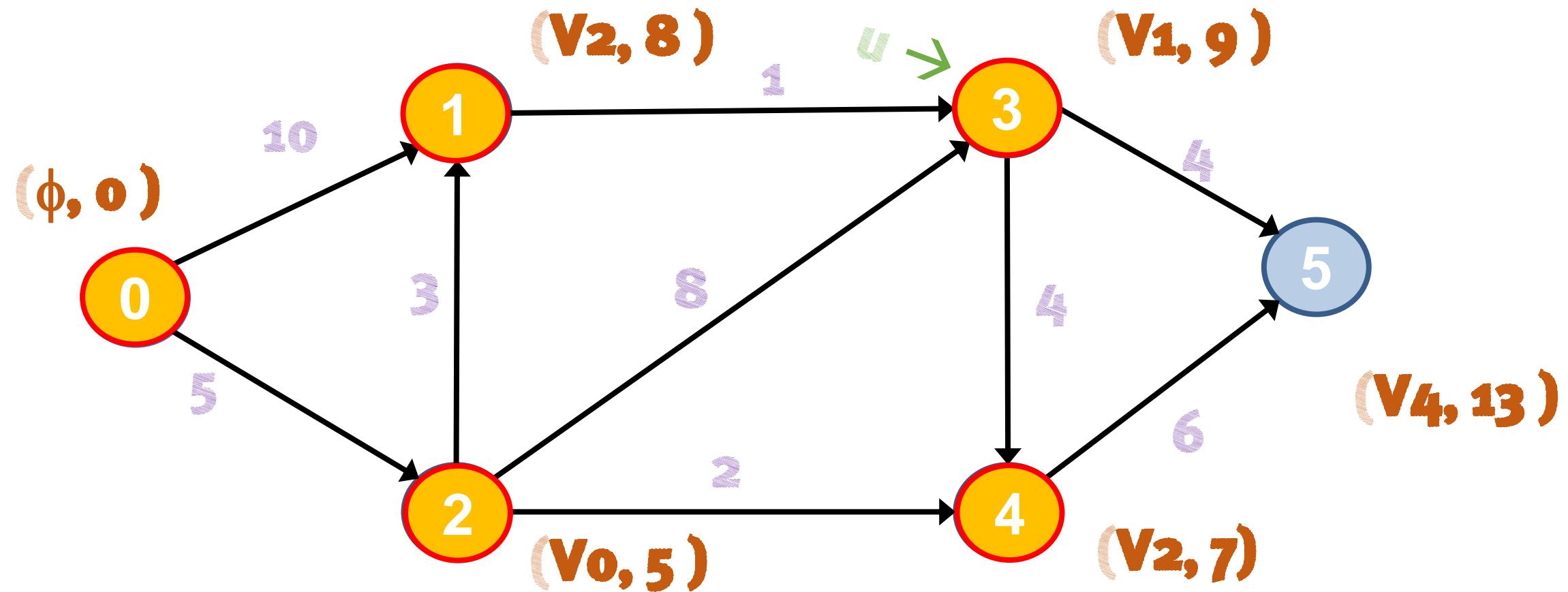
Algoritmo de Dijkstra

07 | Enquanto houver vértice aberto
08 | Encontrar **u** com menor estimativa dentre os abertos
09 | Fechar **u**
10 | $\forall v$, adjacente de **u** AND Aberto
11 | Relaxar aresta (**u**, **v**)



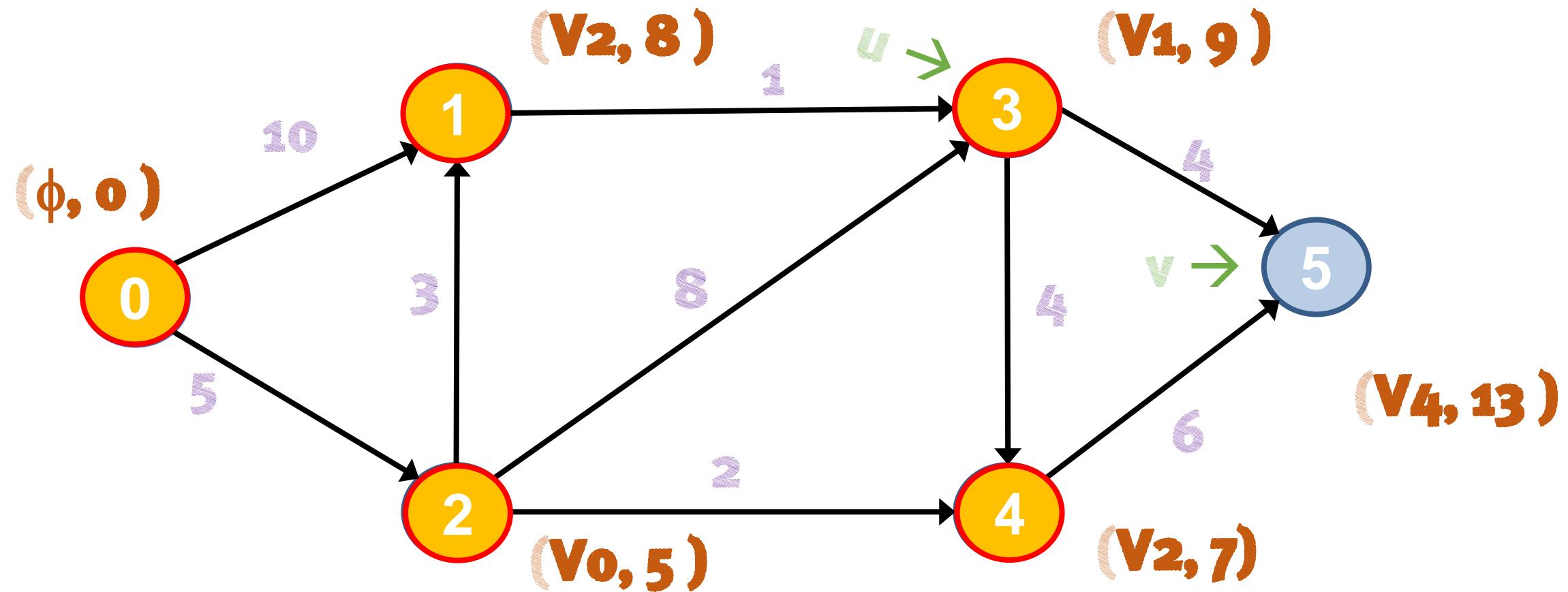
Algoritmo de Dijkstra

07 Enquanto houver vértice aberto
08 Encontrar **u** com menor estimativa dentre os abertos
09 Fechar **u**
10 $\forall v$, adjacente de **u** AND Aberto
11 Relaxar aresta (**u**, **v**)



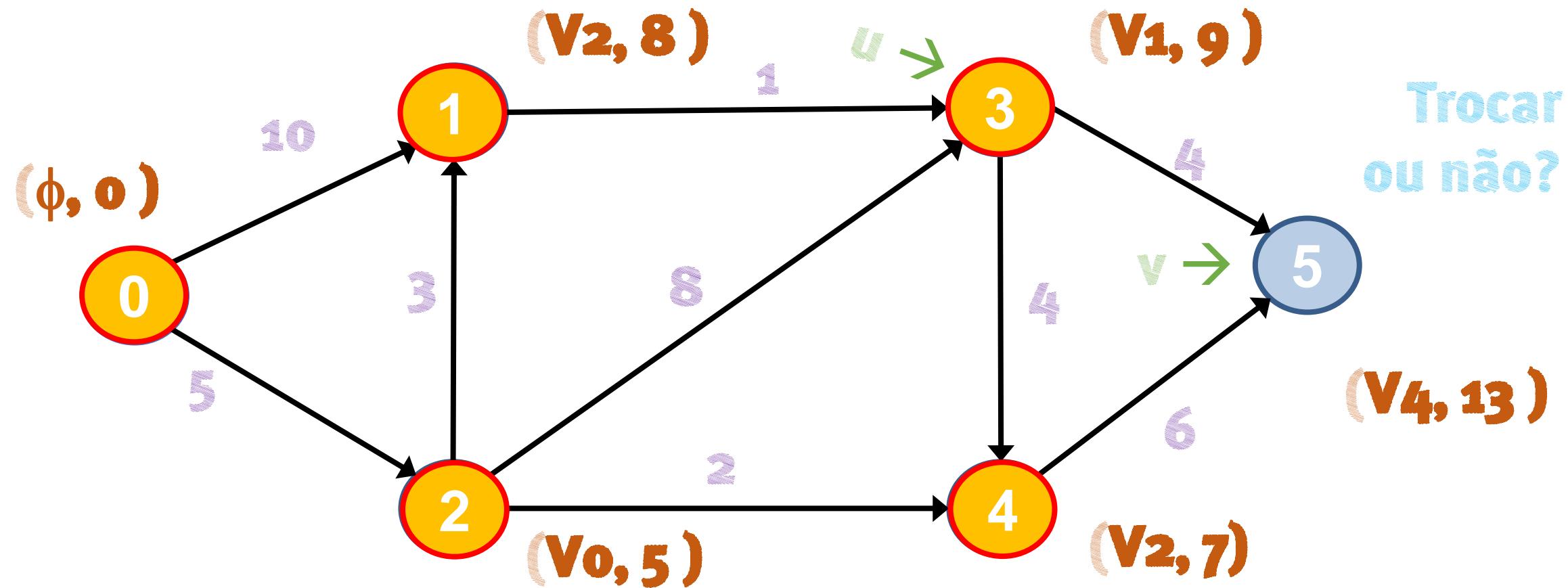
Algoritmo de Dijkstra

07 | Enquanto houver vértice aberto
08 | Encontrar **u** com menor estimativa dentre os abertos
09 | Fechar **u**
10 | $\forall v$, adjacente de **u** AND Aberto
11 | Relaxar aresta (**u**, **v**)



Algoritmo de Dijkstra

07 Enquanto houver vértice aberto
08 Encontrar **u** com menor estimativa dentre os abertos
09 Fechar **u**
10 $\forall v$, adjacente de **u** AND Aberto
11 Relaxar aresta (**u**, **v**)



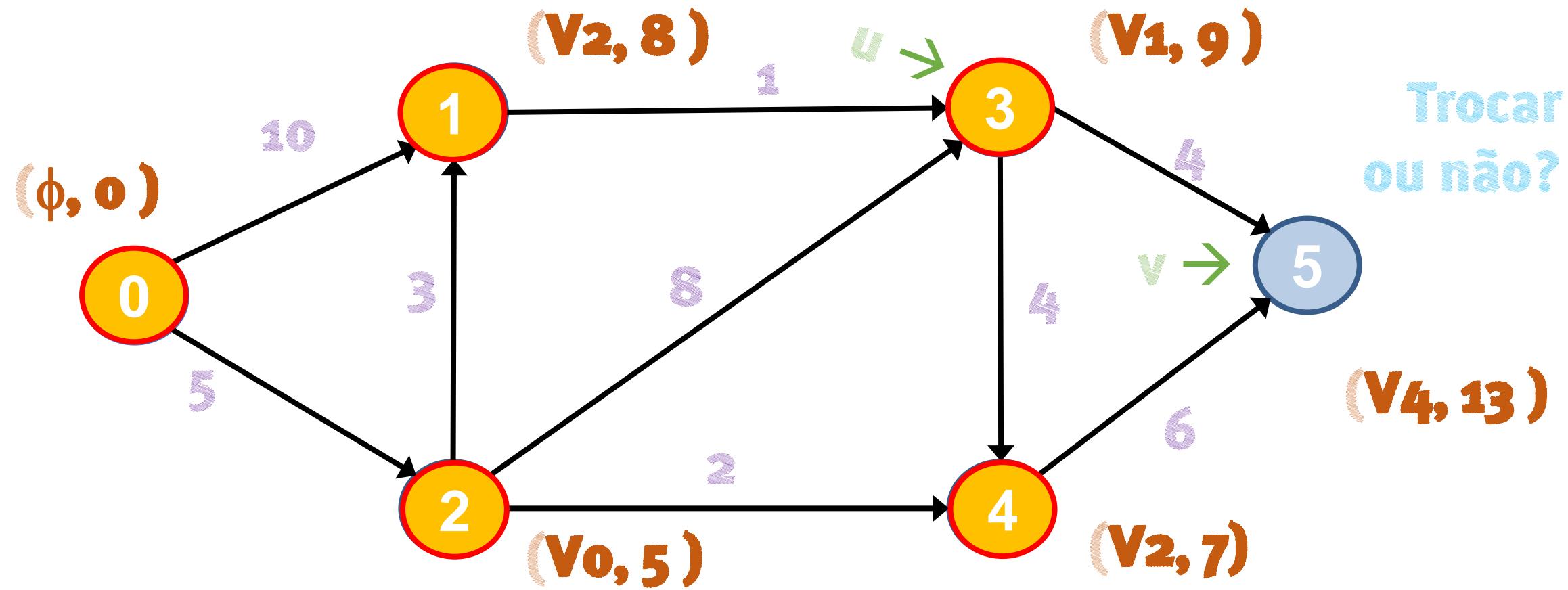
Algoritmo de Dijkstra

```

07 | Enquanto houver vértice aberto
08 |   Encontrar u com menor estimativa dentre os abertos
09 |   Fechar u
10 |    $\forall v$ , adjacente de u AND Aberto
11 |     Relaxar aresta (u, v)

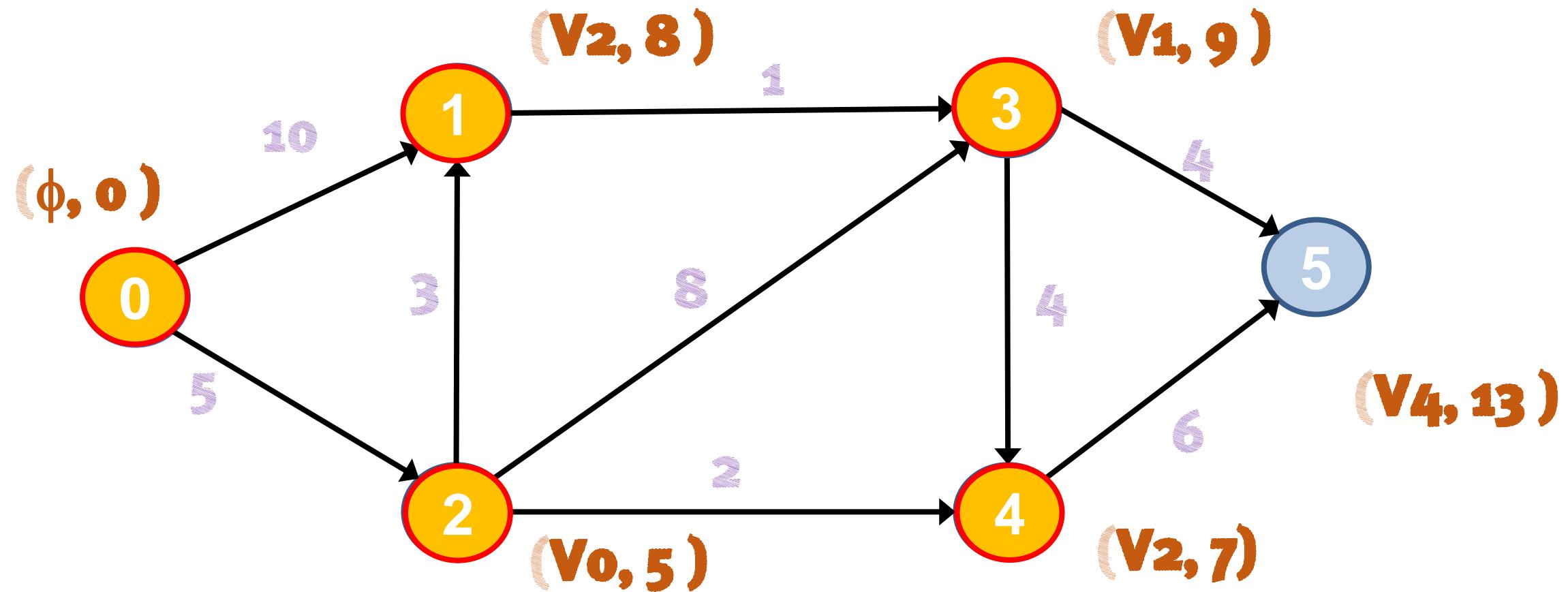
```

12	Relaxar (u , v) {
13	se (v.d > u.d + peso(u , v)){
14	v.d = u.d + peso(u , v)
15	v.pai = u
16	}



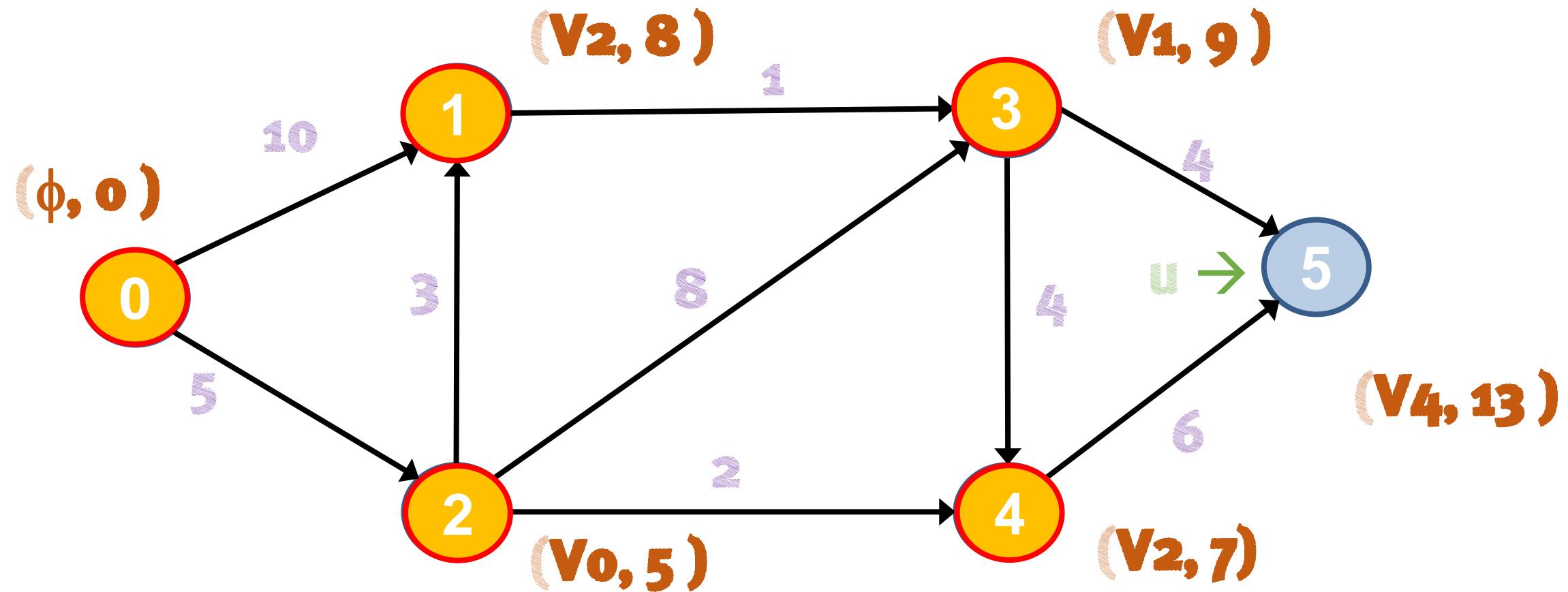
Algoritmo de Dijkstra

07 | Enquanto houver vértice aberto
08 | Encontrar **u** com menor estimativa dentre os abertos
09 | Fechar **u**
10 | $\forall v$, adjacente de **u** AND Aberto
11 | Relaxar aresta (**u**, **v**)



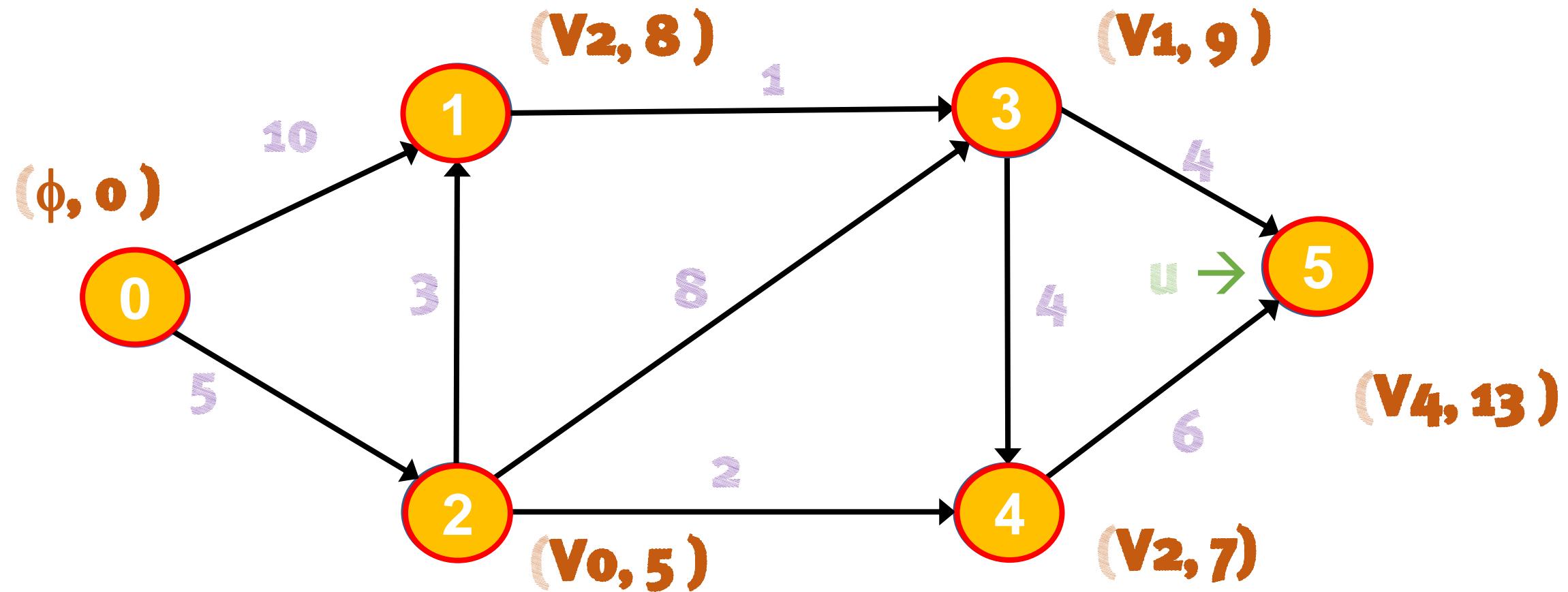
Algoritmo de Dijkstra

07 | Enquanto houver vértice aberto
08 | Encontrar **u** com menor estimativa dentre os abertos
09 | Fechar **u**
10 | $\forall v$, adjacente de **u** AND Aberto
11 | Relaxar aresta (**u**, **v**)



Algoritmo de Dijkstra

07 | Enquanto houver vértice aberto
08 | Encontrar **u** com menor estimativa dentre os abertos
09 | Fechar **u**
10 | $\forall v$, adjacente de **u** AND Aberto
11 | Relaxar aresta (**u**, **v**)



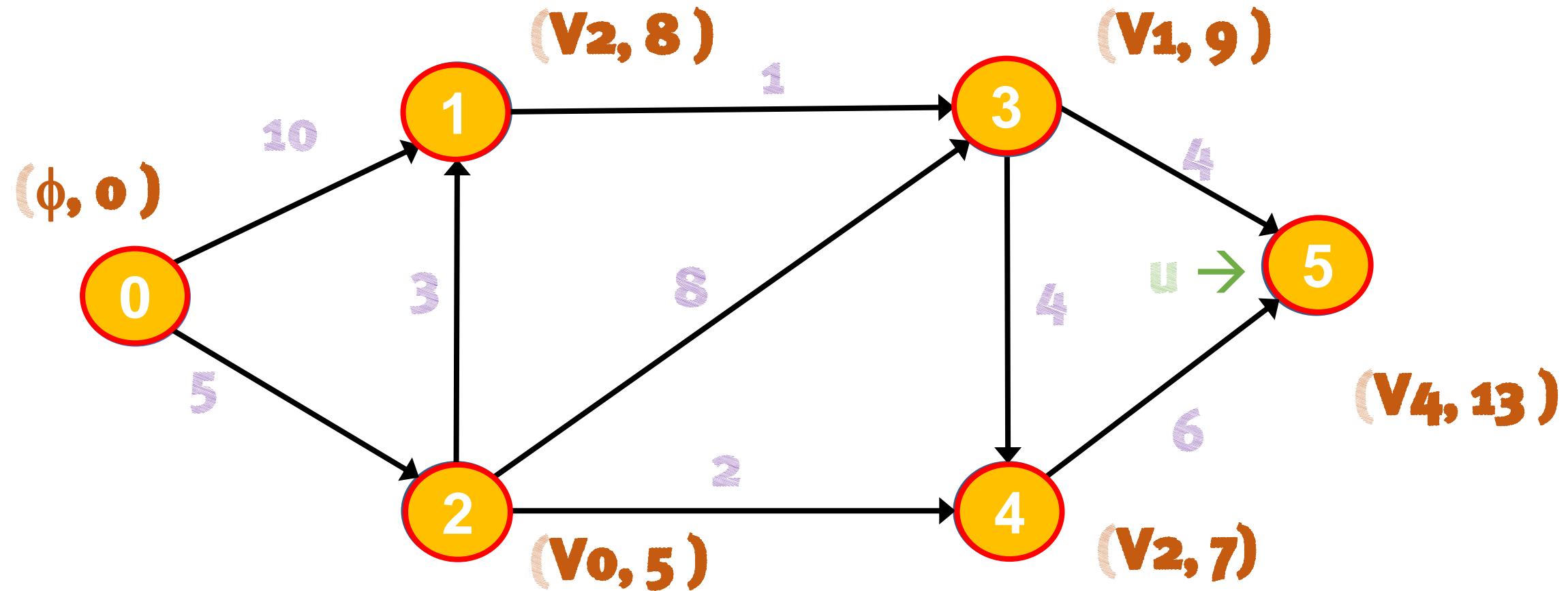
Algoritmo de Dijkstra

```

07 | Enquanto houver vértice aberto
08 |   Encontrar u com menor estimativa dentre os abertos
09 |   Fechar u
10 |    $\forall v$ , adjacente de u AND Aberto
11 |     Relaxar aresta (u, v)

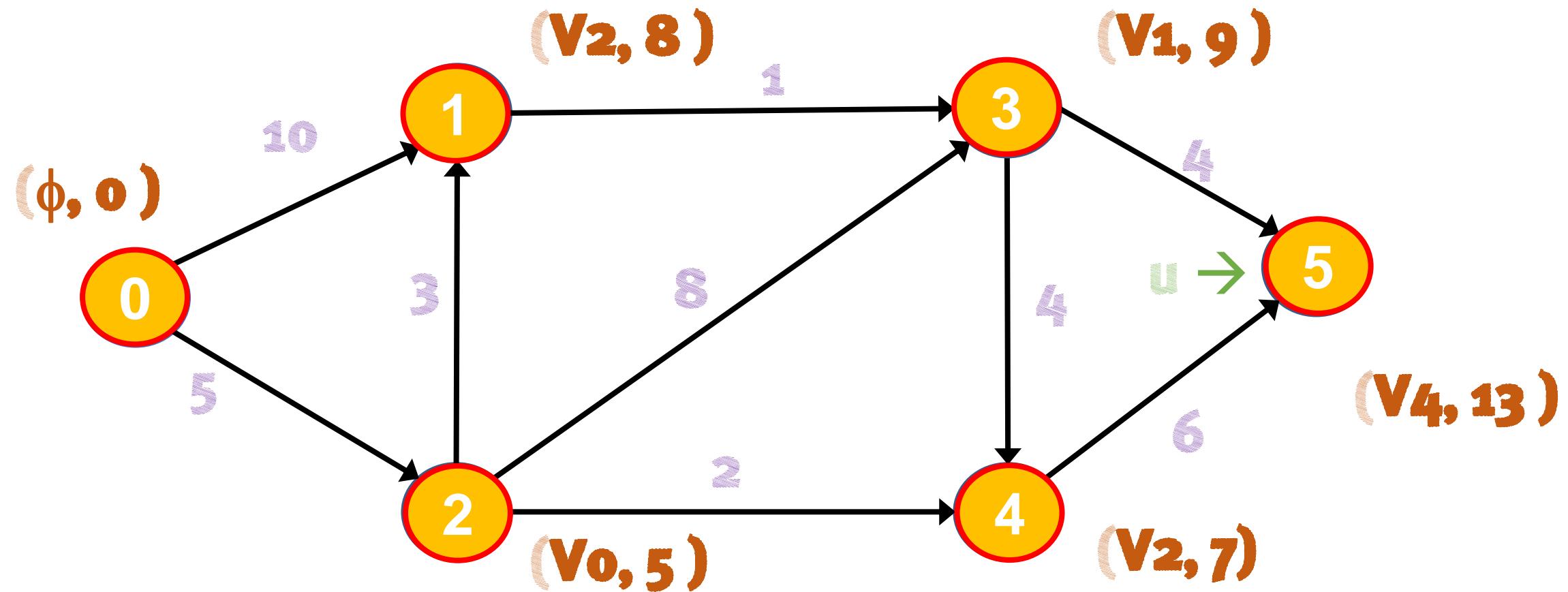
```

12	Relaxar (u , v) {
13	se (v.d > u.d + peso(u , v)){
14	v.d = u.d + peso(u , v)
15	v.pai = u
16	}



Algoritmo de Dijkstra

07 | Enquanto houver vértice aberto
08 | Encontrar **u** com menor estimativa dentre os abertos
09 | Fechar **u**
10 | $\forall v$, adjacente de **u** AND Aberto
11 | Relaxar aresta (**u**, **v**)



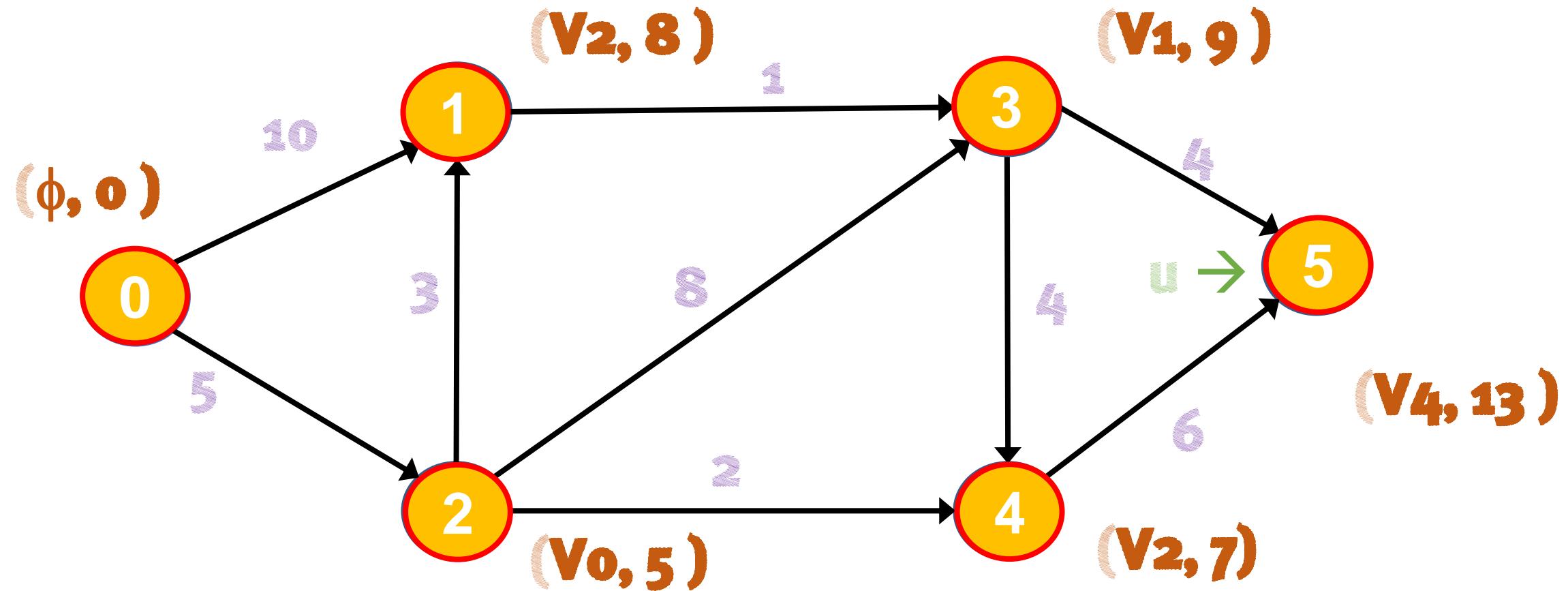
Algoritmo de Dijkstra

```

07 | Enquanto houver vértice aberto
08 |   Encontrar u com menor estimativa dentre os abertos
09 |   Fechar u
10 |    $\forall v$ , adjacente de u AND Aberto
11 |     Relaxar aresta (u, v)

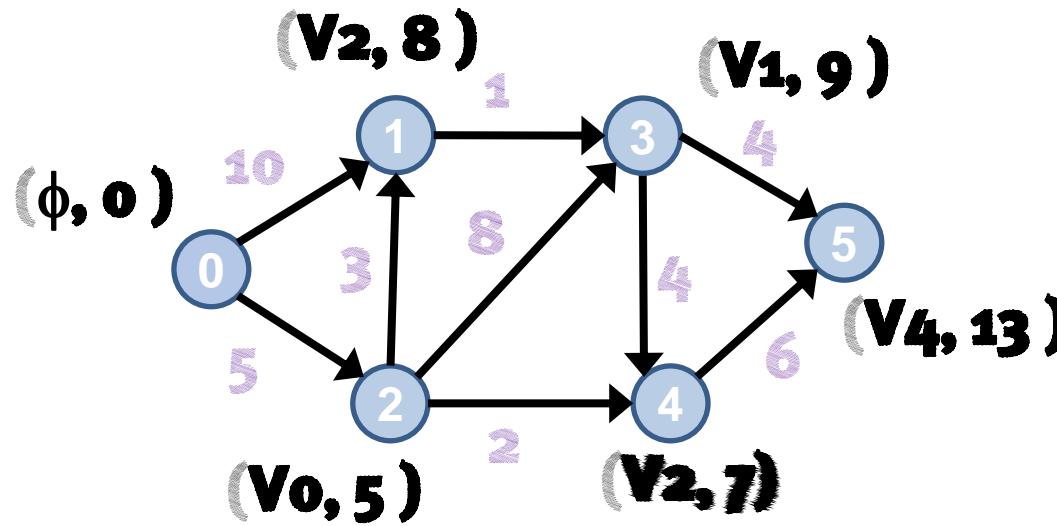
```

12	Relaxar (u , v){
13	se (v.d > u.d + peso(u , v)){
14	v.d = u.d + peso(u , v)
15	v.pai = u
16	}



Algoritmo de Dijkstra

- Define as menores distâncias de cada vértice até o **vértice inicial**
- Um nó **u** é fechado quando se conhece a menor distância dele ao **nó inicial**
- Além da distância, o caminho também pode ser obtido a partir dos antecessores



Menor caminho de V₀ a V₃?

$$V_3 \gg V_1 = 1$$

$$V_1 \gg V_2 = 3$$

$$V_2 \gg V_0 = 5$$

MC = (V₀, V₂, V₁, v₃) com custo = 9

Atividade

- Implementar cada um dos algoritmos apresentados. Utilizar, obrigatoriamente, o grafo desenvolvido por você
- Padrão dados de entrada para todas atividades de grafos

- Tipo Grafo (ND ou D)
- Vértice;Vértice;Peso
- Vértice;Vértice;Peso
-

ND
A;B;9
A;C;5
C;D;3
D;B;9
...

D
A;B;9
A;C;5
C;D;3
D;B;9
...

- Para o caso de grafos não ponderados, o terceiro parâmetro não deverá ser considerado

Prof. Hélder Pereira Borges

helder@ifma.edu.br

Grafos Buscas

