Aplicações de BFS e DFS

CLRS 22 Elementary Graph Algorithms

CLRS 22.2 e 22.3

Busca em largura

```
BFS (G, s)
 1  para cada u ∈ G.V \ {s} faça
 2       u.cor ← branco    u.d ← ∞    u.π ← nil
 3  Q ← ∅          ▷ fila dos vértices descobertos
 4  s.cor ← cinzento    s.d ← 0    s.π ← nil
 5  Q.INSIRA(s)

 6  enquanto Q ≠ ∅ faça
 7       u ← Q.REMOVA()
 8       para cada v ∈ u.Estrela faça
 9            se v.cor = branco
10                 então v.cor ← cinzento
11                       v.d ← u.d + 1
12                       v.π ← u
13                       Q.INSIRA(v)
14       u.cor ← preto
```

Descrição

Vértice branco: ainda não descoberto

Vértice cinzento: descoberto mas não processado
(são os vértices em Q)

Vértice preto: processado

Descrição

Vértice branco: ainda não descoberto

Vértice cinzento: descoberto mas não processado
(são os vértices em Q)

Vértice preto: processado

BFS devolve em π uma árvore BF enraizada em s.

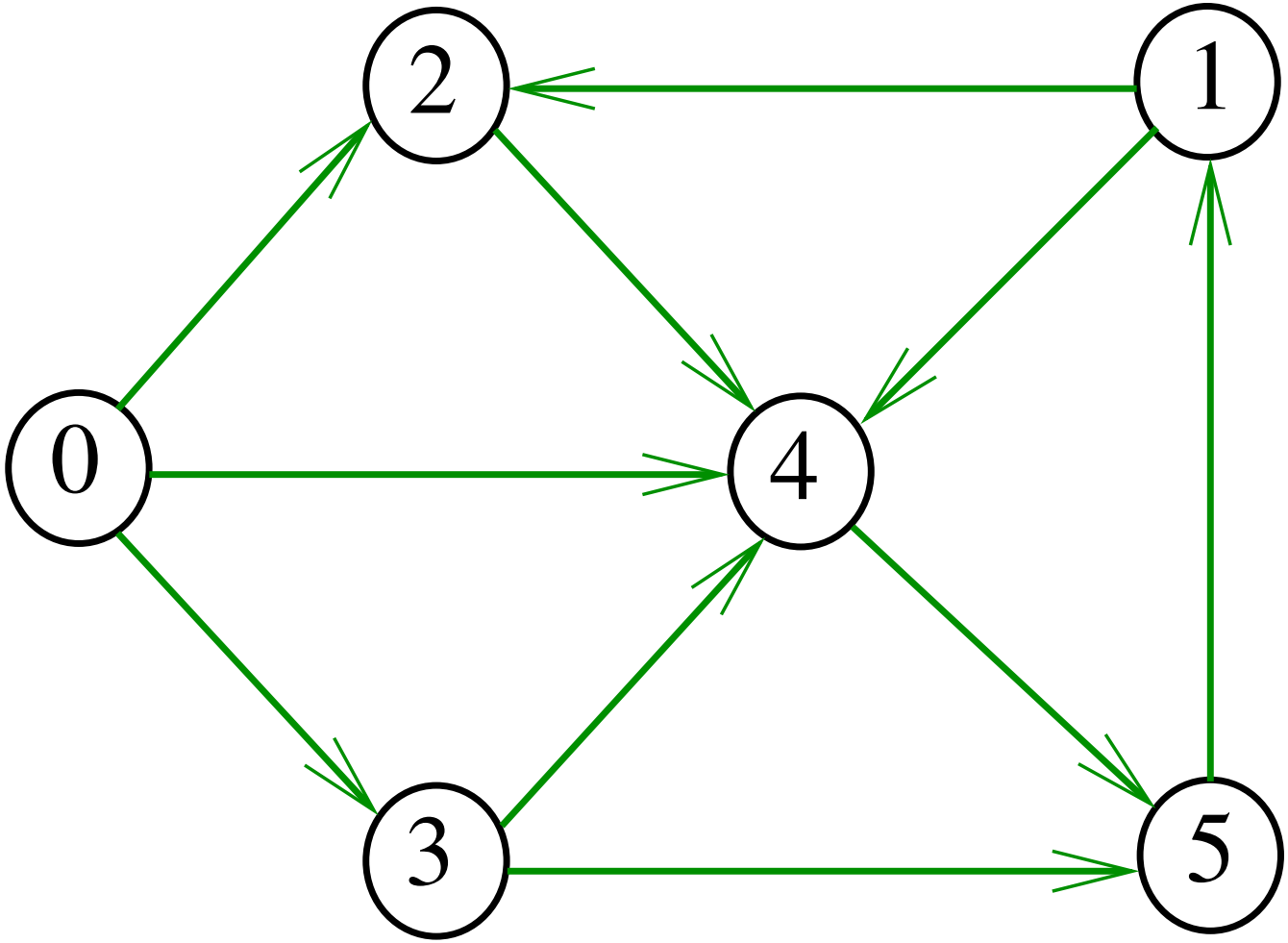
$u.\pi$: predecessor ou pai de u na árvore BF

$u.d$: distância de s a u em G

BFS descobre todos os vértices à distância k
antes de descobrir qualquer um à distância $k+1$

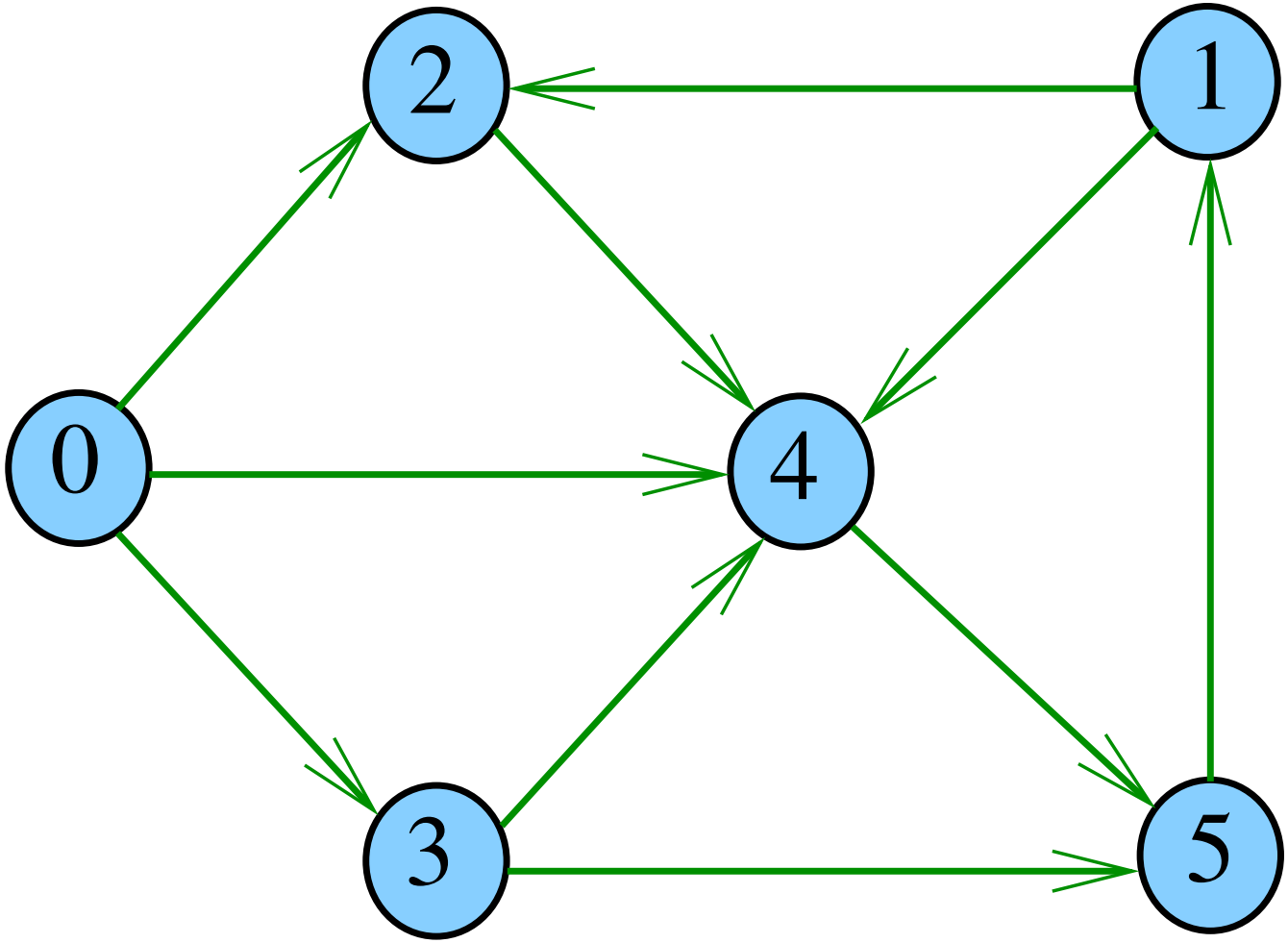
BFS(G,0)

i	0	1	2	3	4	5
q[i]						



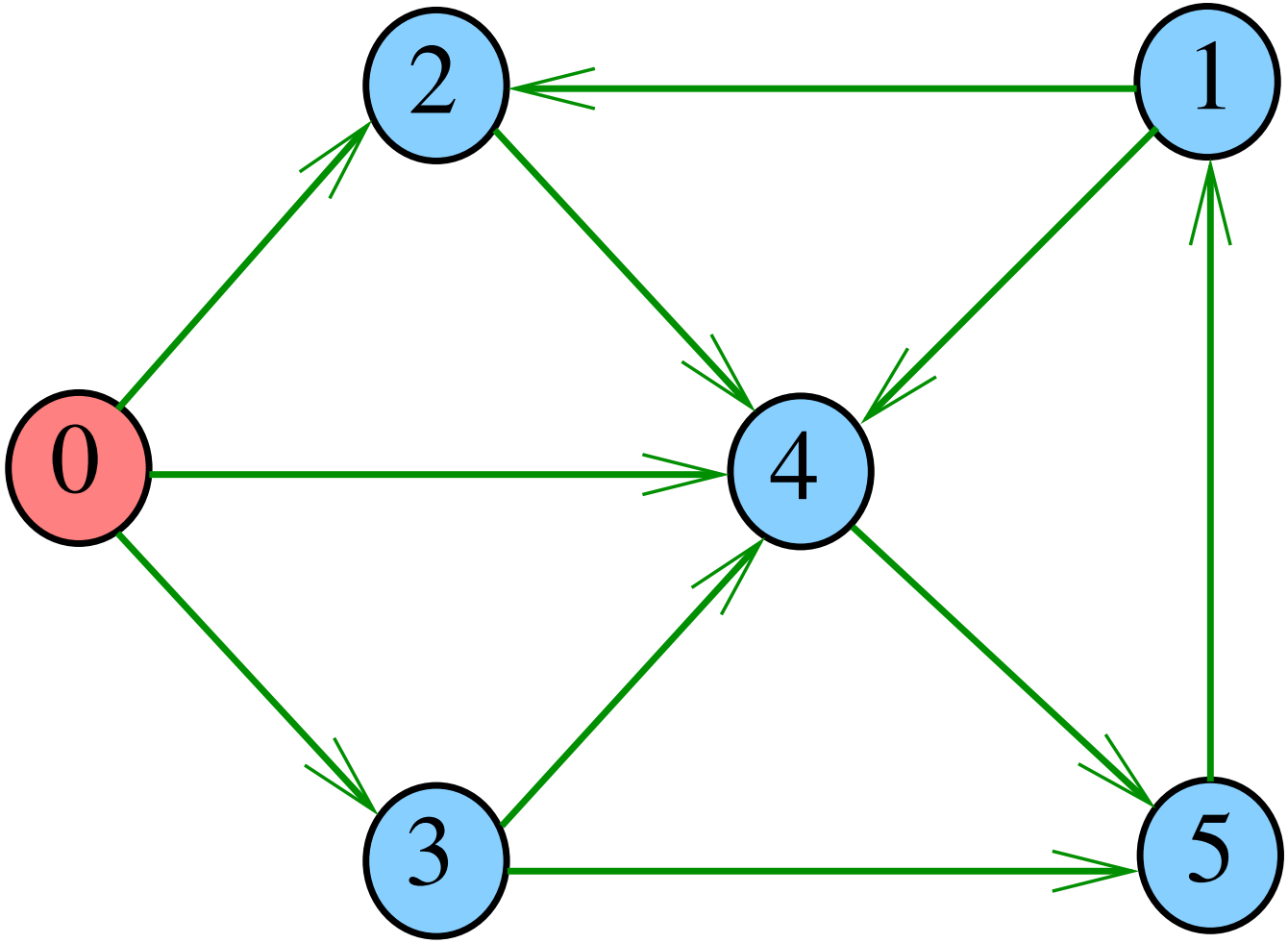
BFS(G,0)

i	0	1	2	3	4	5
q[i]						



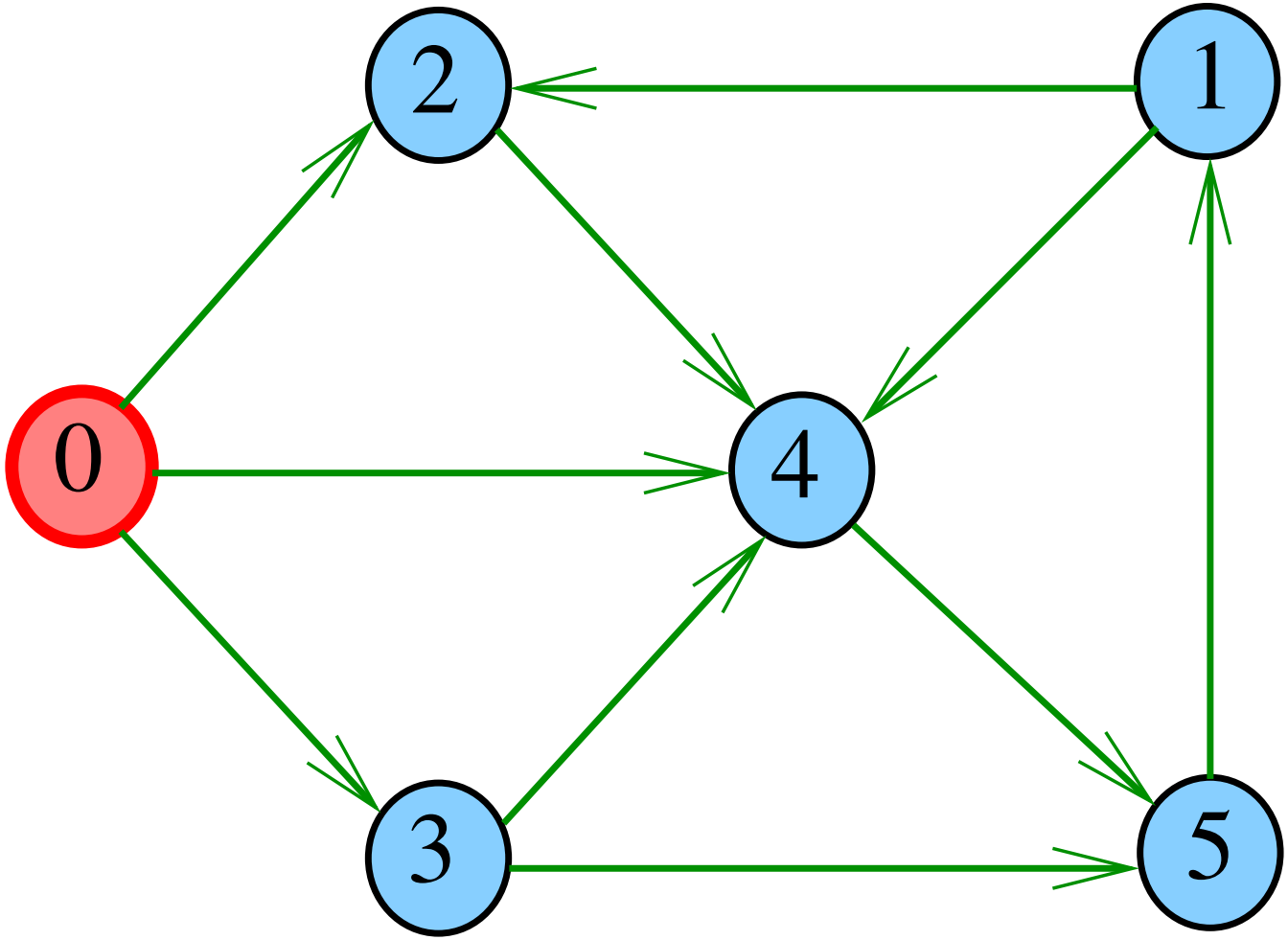
BFS(G,0)

i	0	1	2	3	4	5
q[i]	**0**					



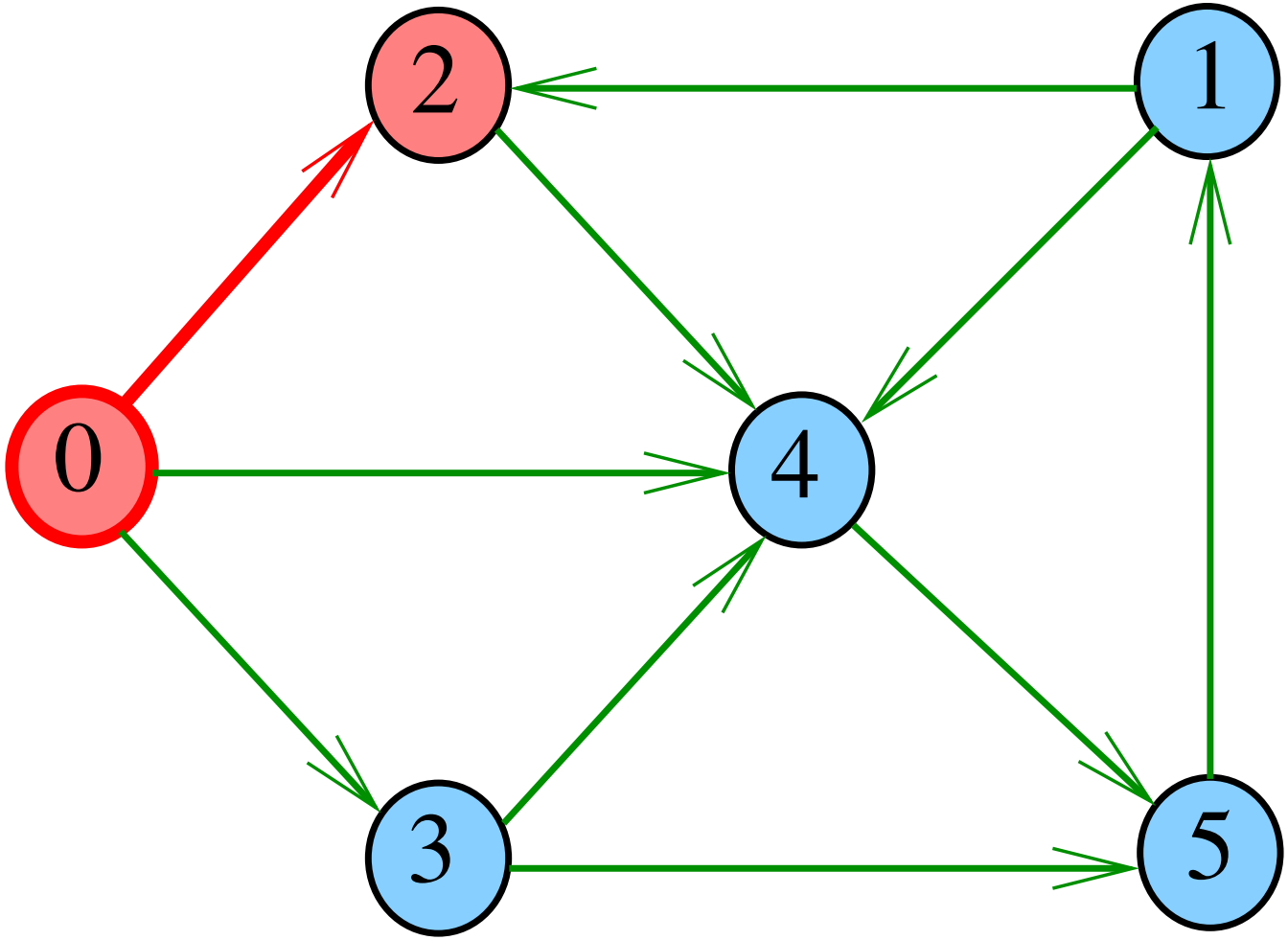
BFS(G,0)

i	0	1	2	3	4	5
q[i]	**0**					



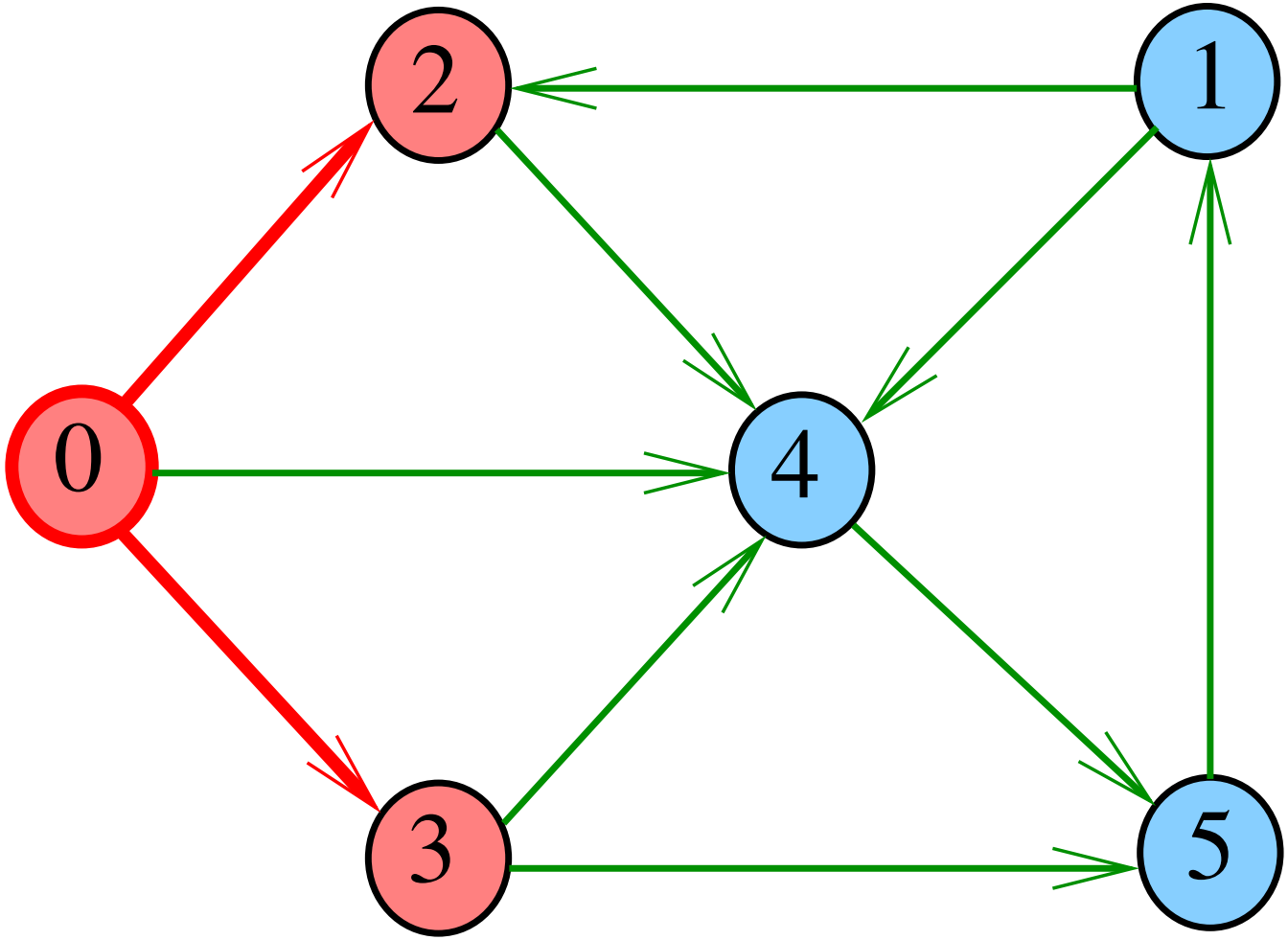
BFS(G,0)

i	0	1	2	3	4	5
q[i]	**0**	**2**				



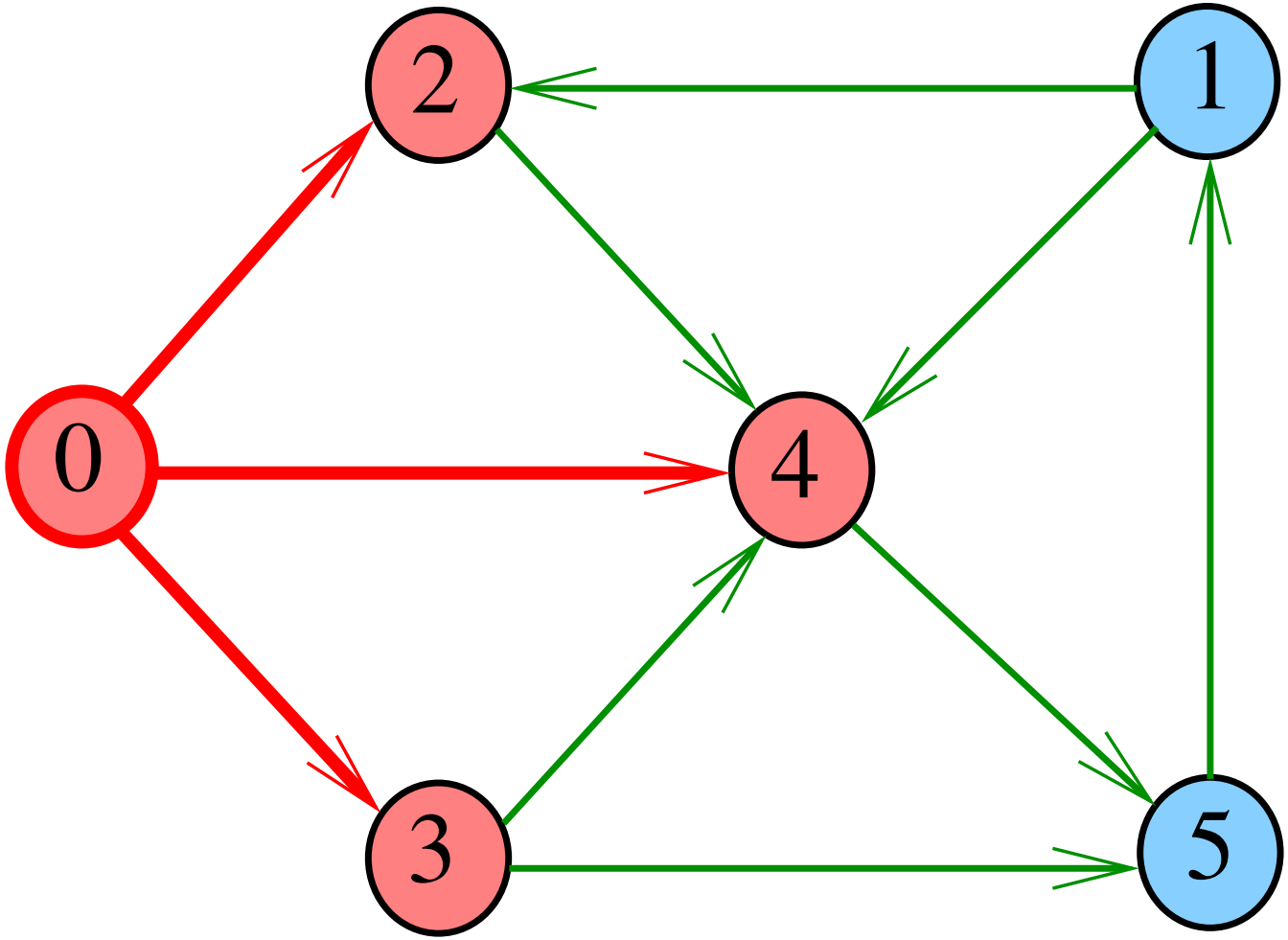
BFS(G,0)

i	0	1	2	3	4	5
q[i]	**0**	**2**	**3**			



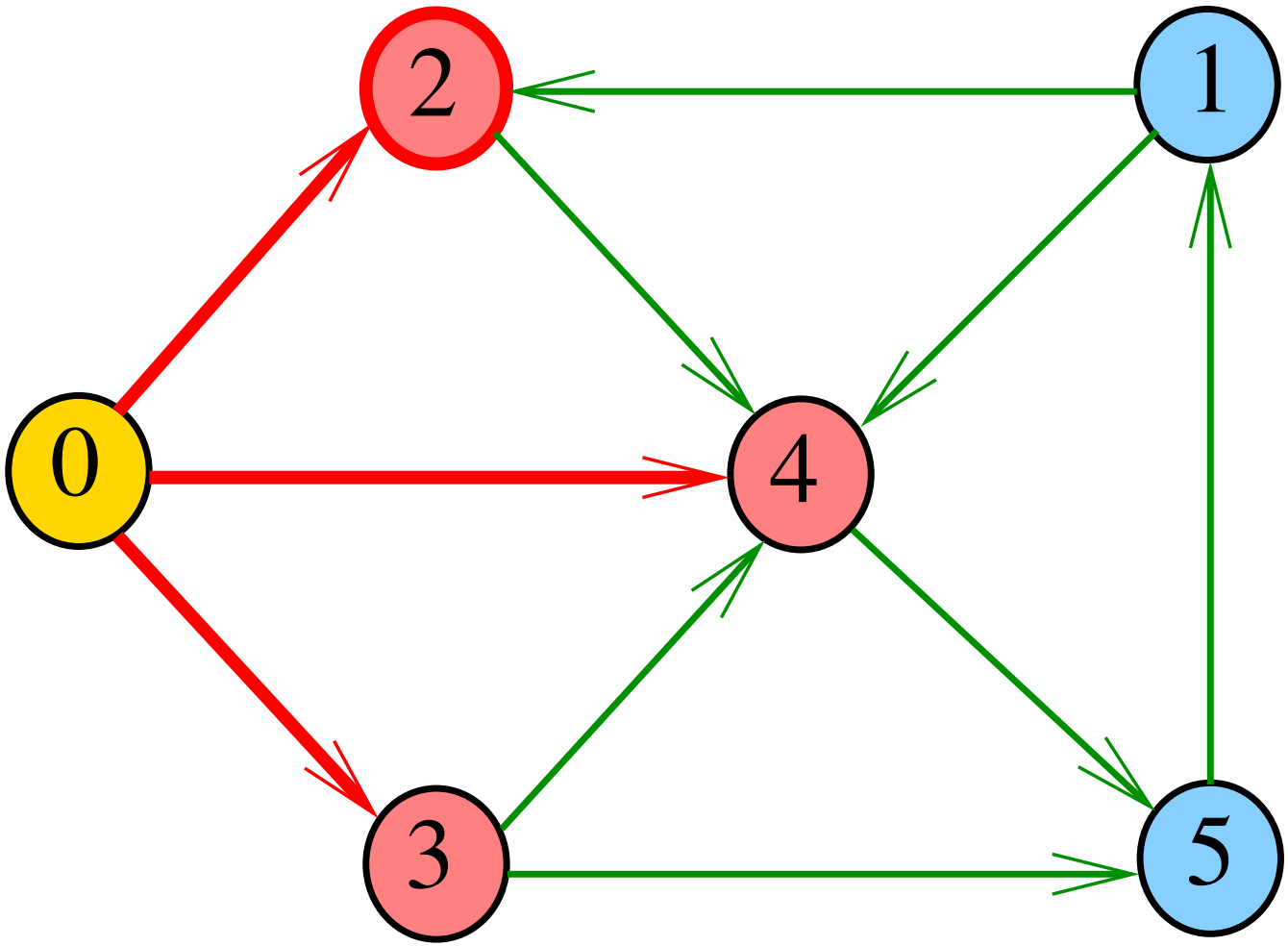
BFS(G,0)

i	0	1	2	3	4	5
q[i]	**0**	**2**	**3**	**4**		



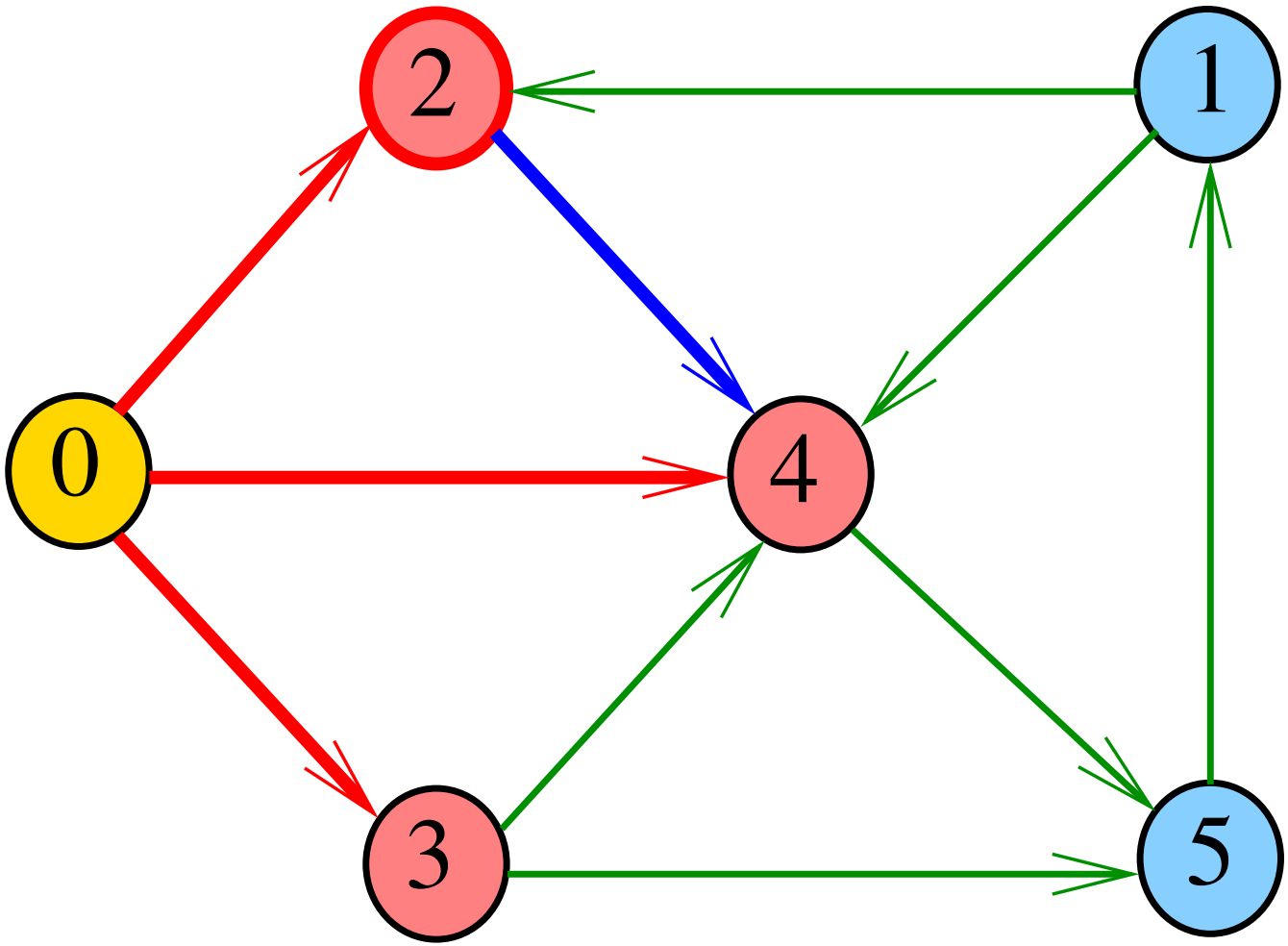
BFS(G,0)

i	0	1	2	3	4	5
q[i]	0	2	3	4		



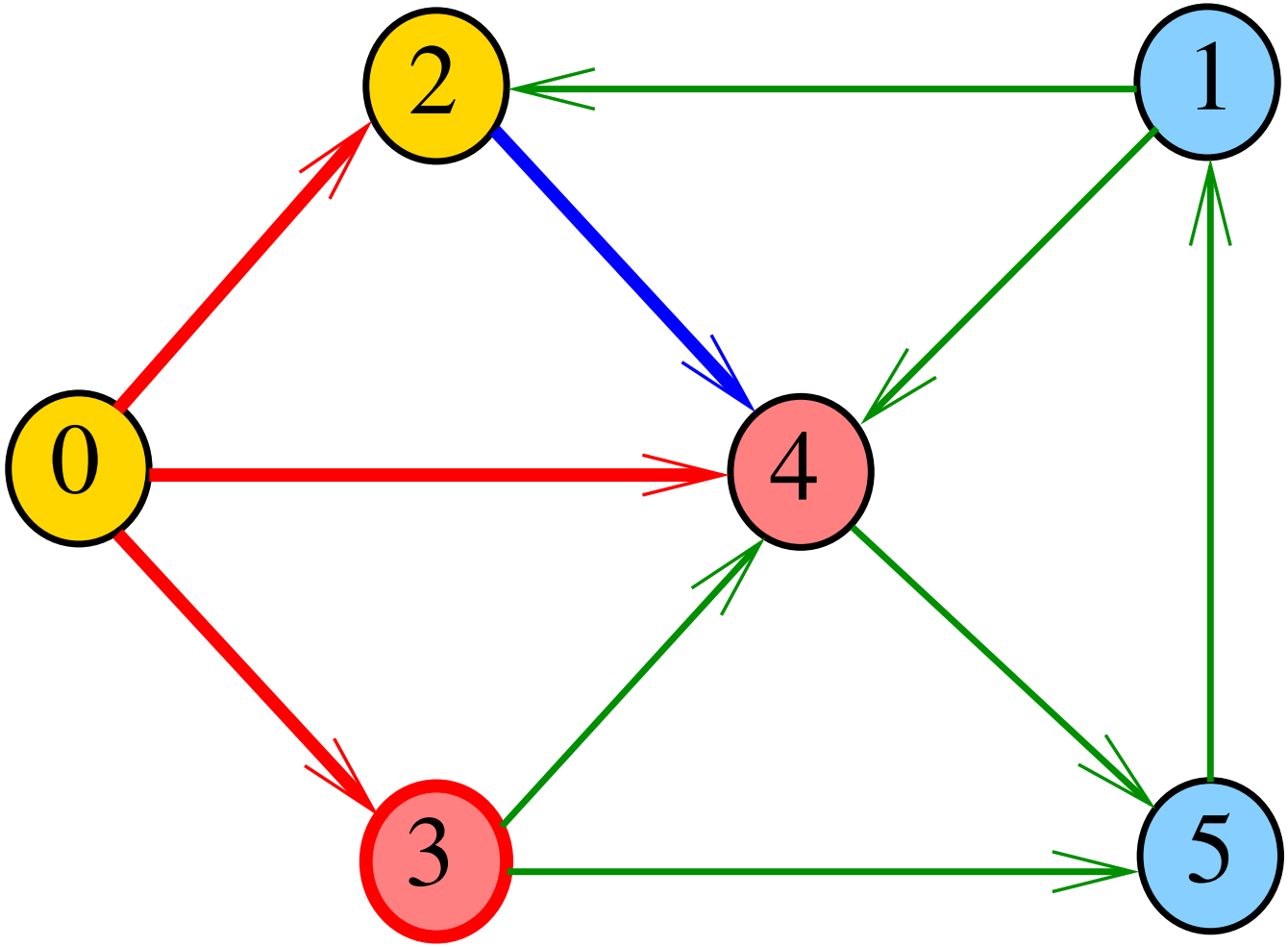
BFS(G,0)

i	0	1	2	3	4	5
q[i]	0	2	3	4		



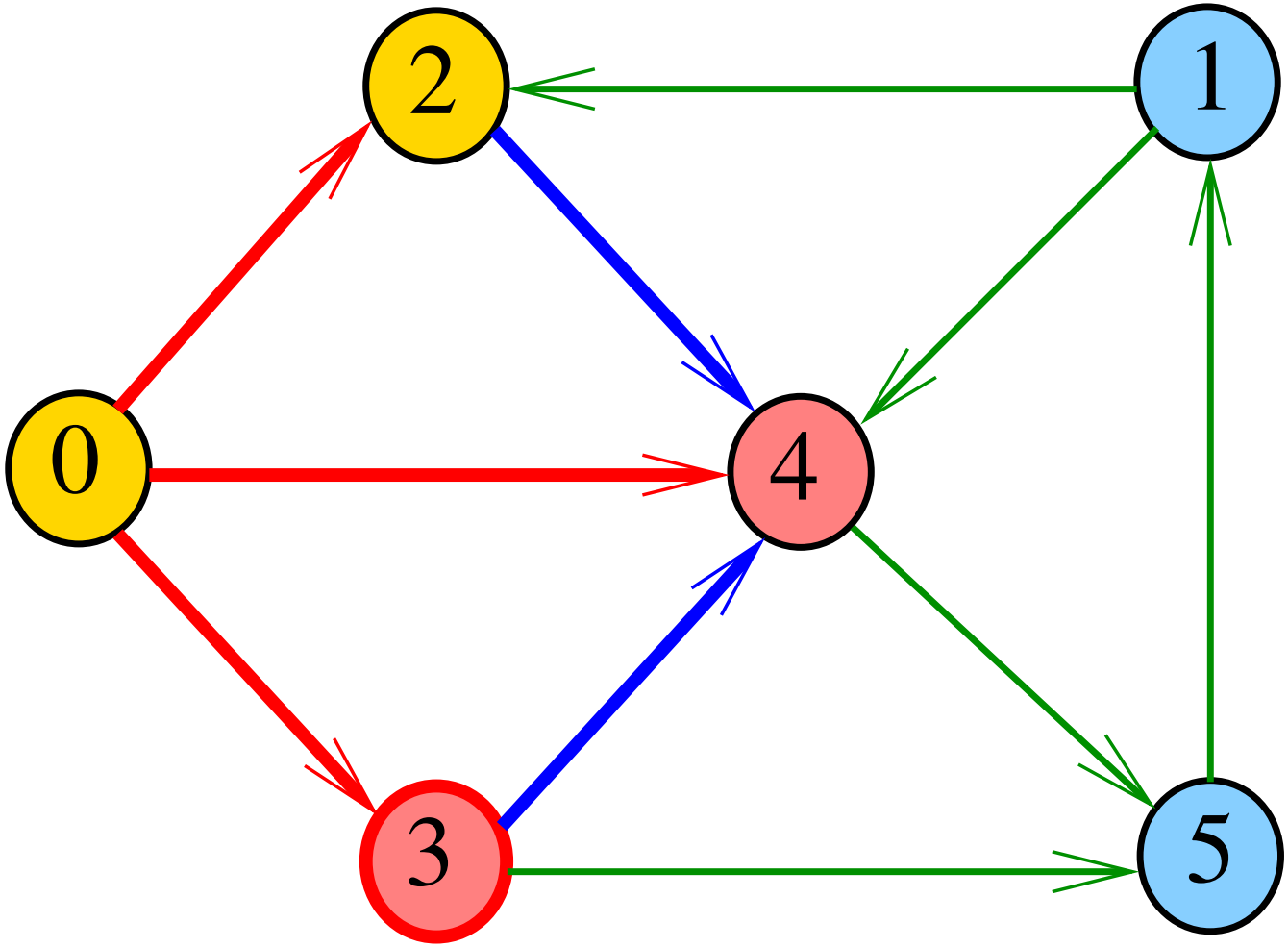
BFS(G,0)

i	0	1	2	3	4	5
q[i]	0	2	3	4		



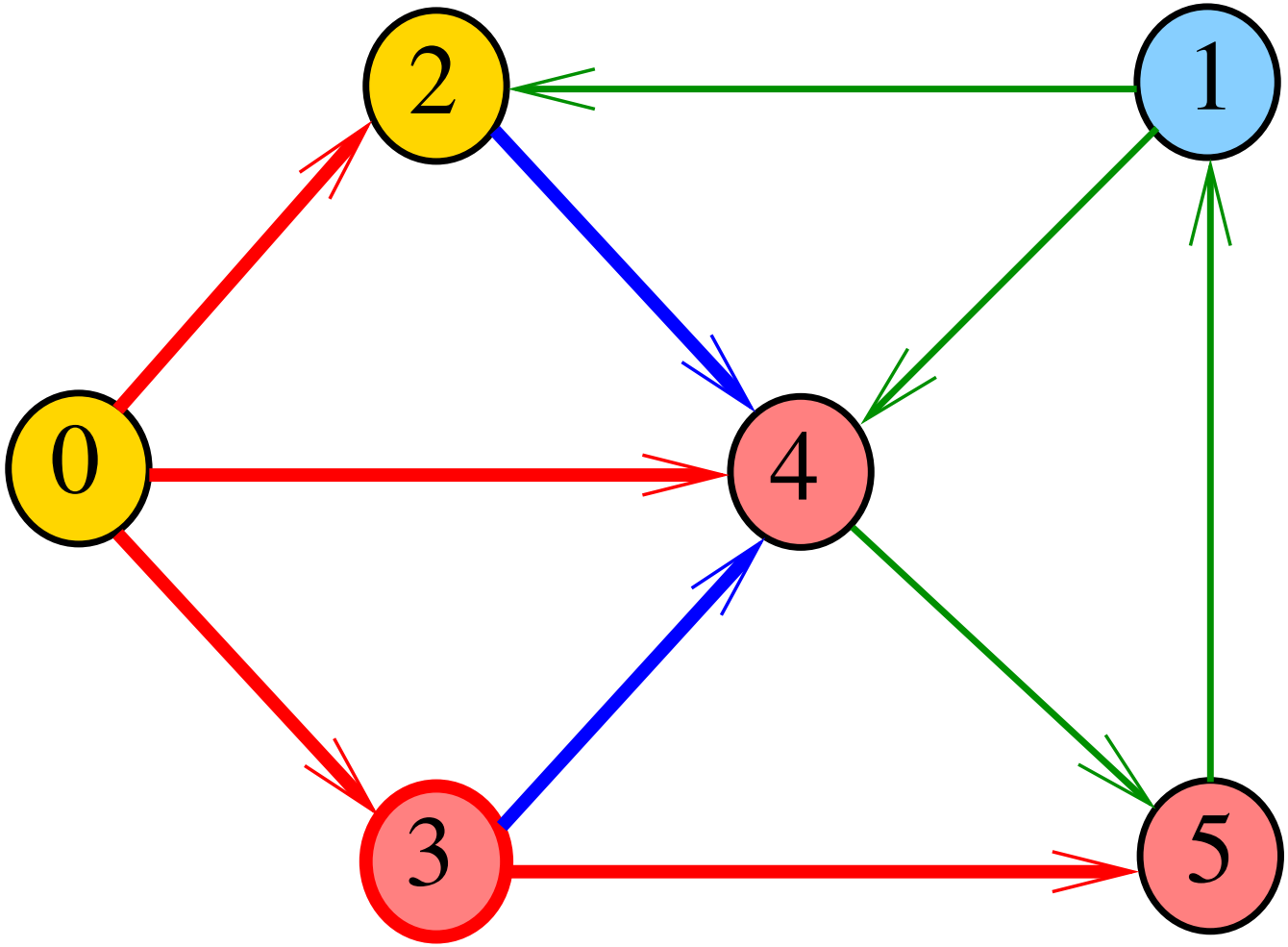
BFS(G,0)

i	0	1	2	3	4	5
q[i]	0	2	3	4		



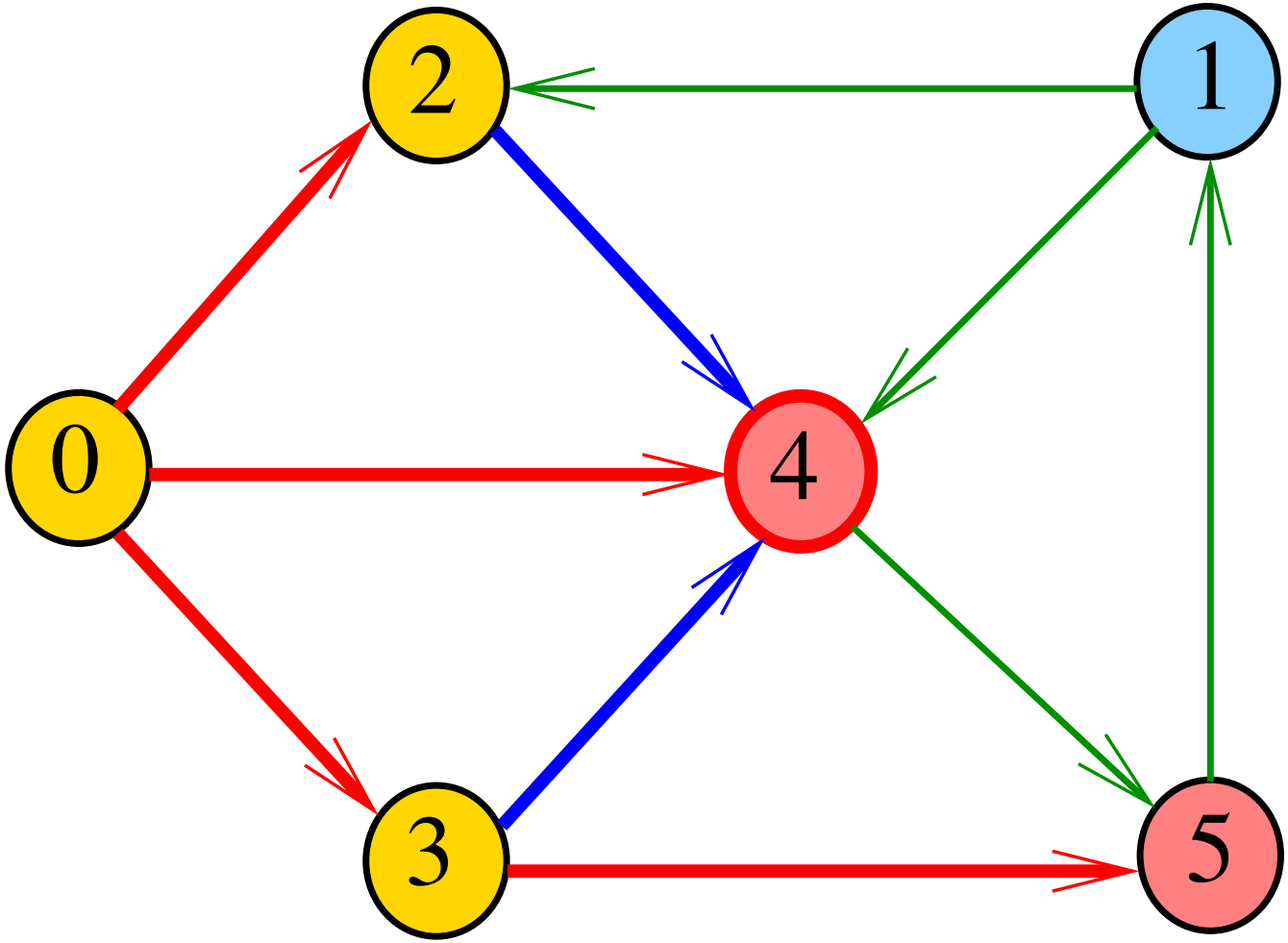
BFS(G,0)

i	0	1	2	3	4	5
q[i]	0	2	3	4	5	



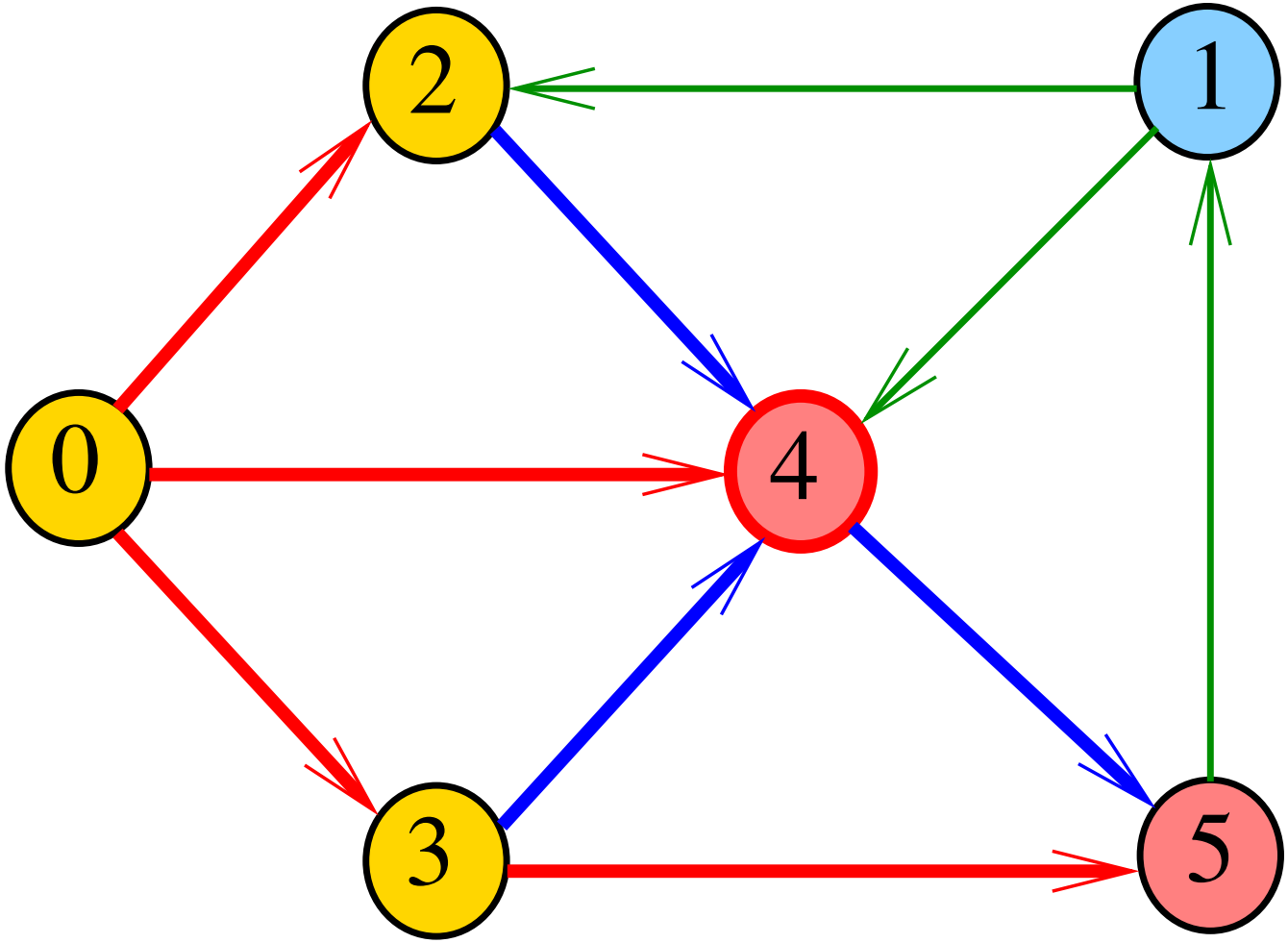
BFS(G,0)

i	0	1	2	3	4	5
q[i]	0	2	3	4	5	



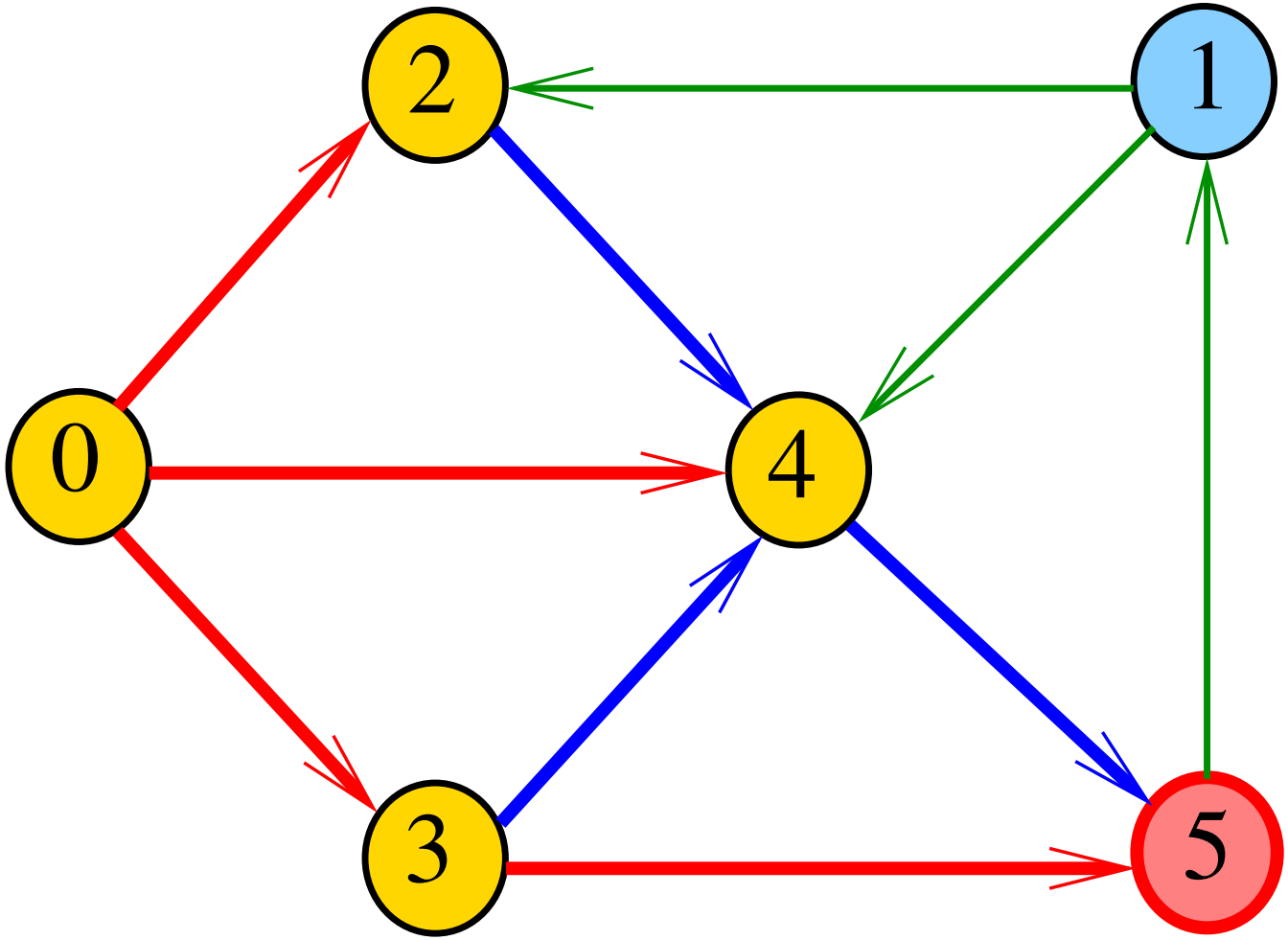
BFS(G,0)

i	0	1	2	3	4	5
q[i]	0	2	3	4	5	



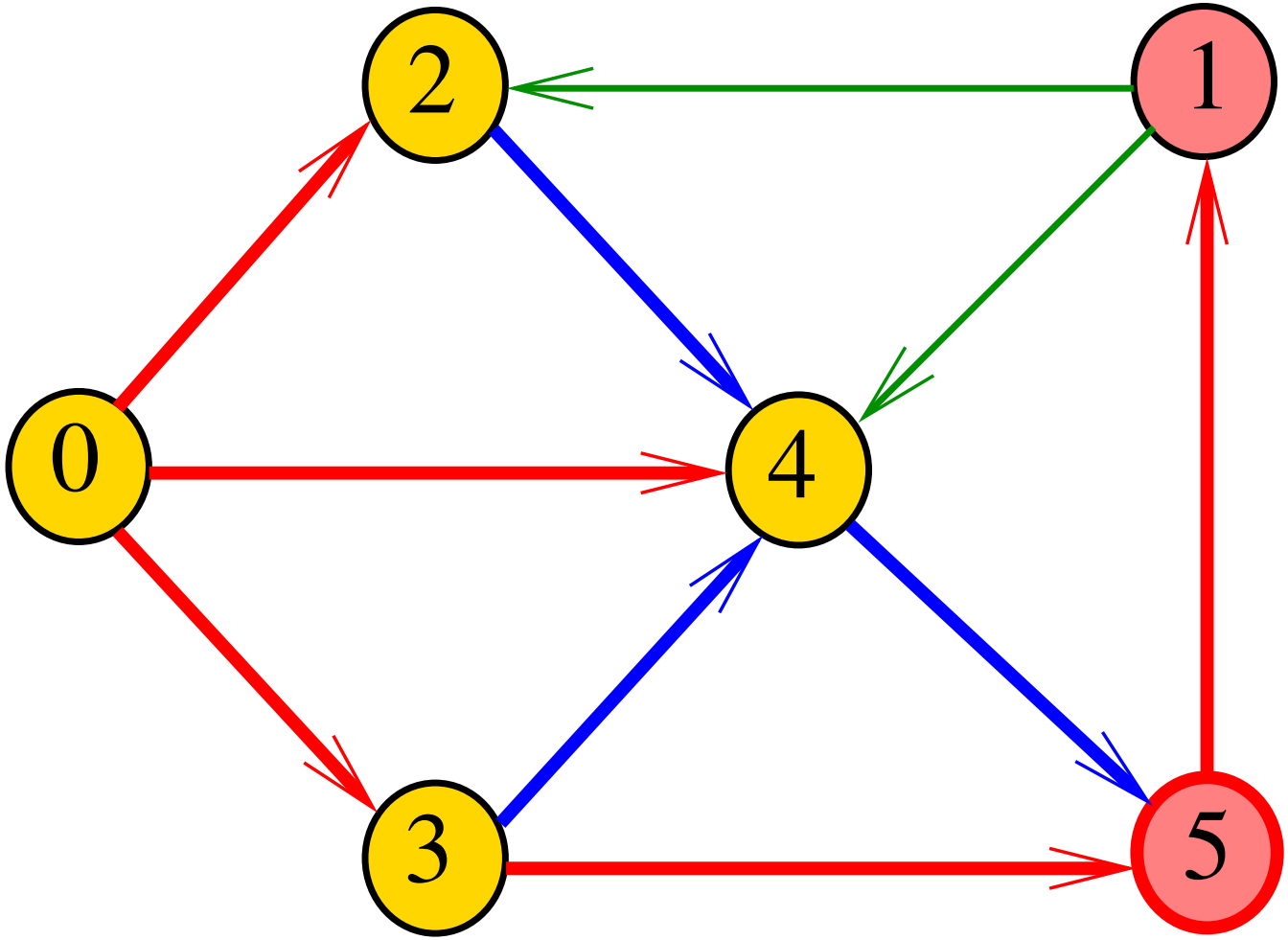
BFS(G,0)

i	0	1	2	3	4	5
q[i]	0	2	3	4	5	



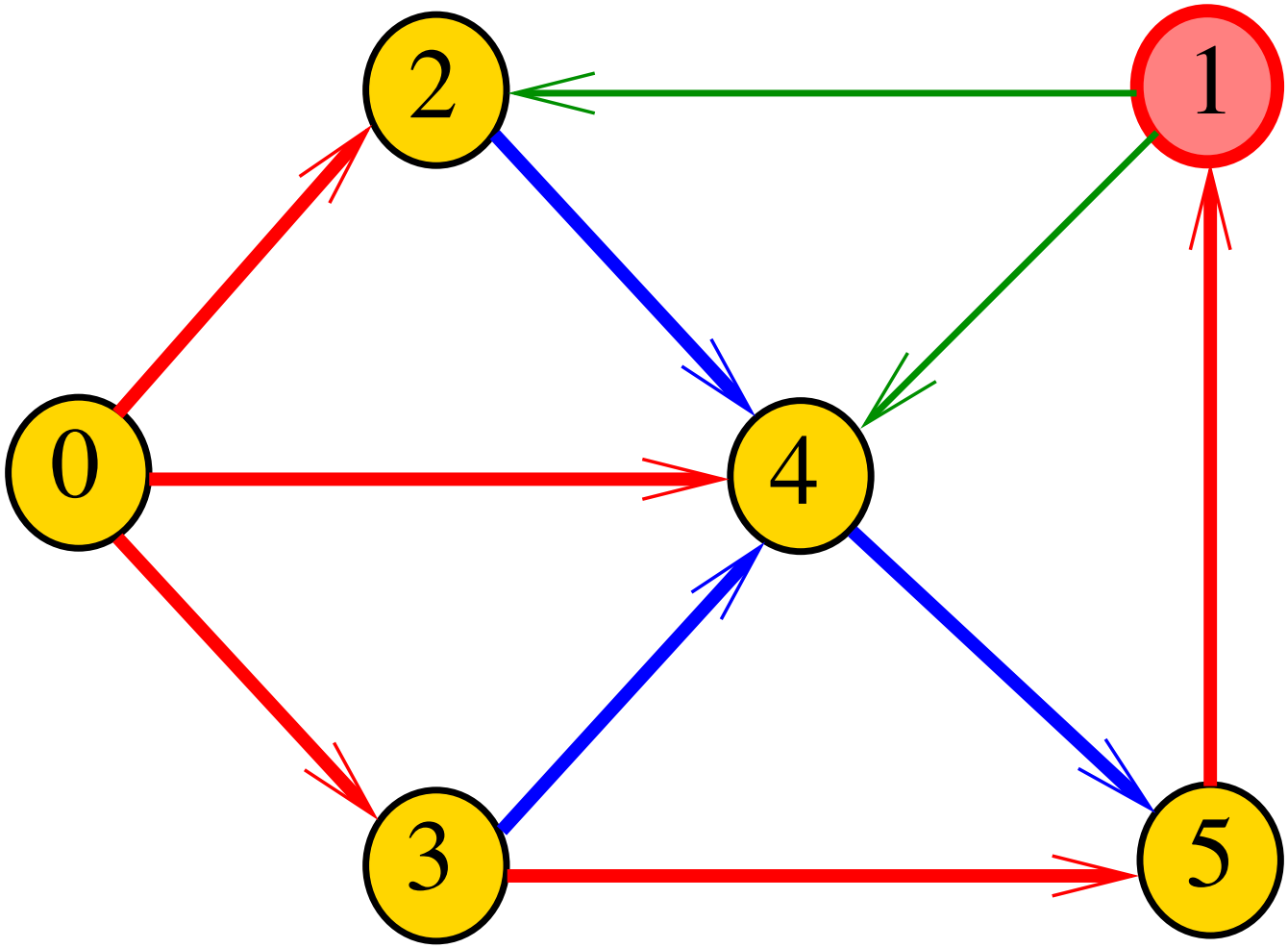
BFS(G,0)

i	0	1	2	3	4	5
q[i]	0	2	3	4	5	1



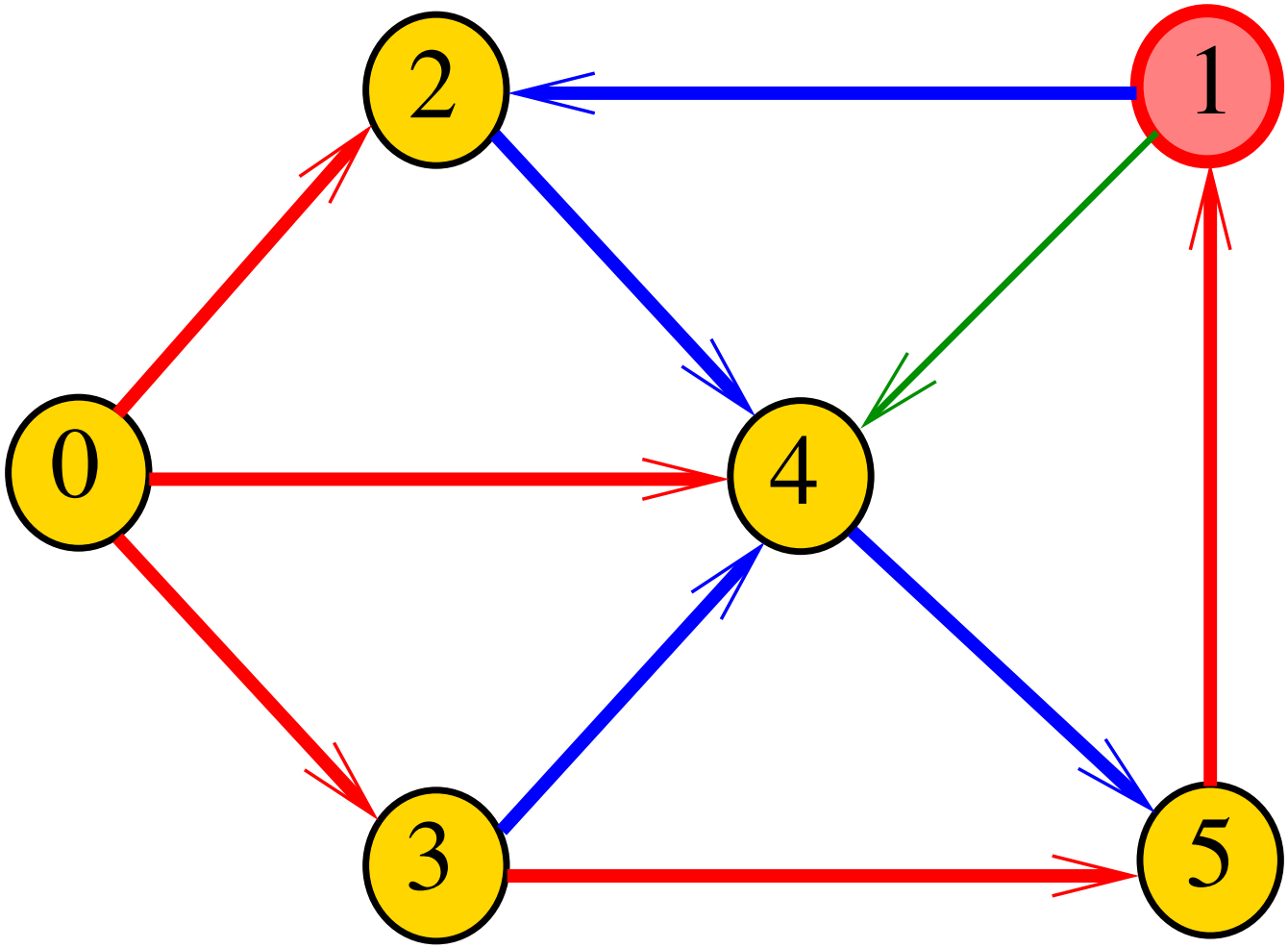
BFS(G,0)

i	0	1	2	3	4	5
q[i]	**0**	**2**	**3**	**4**	**5**	**1**



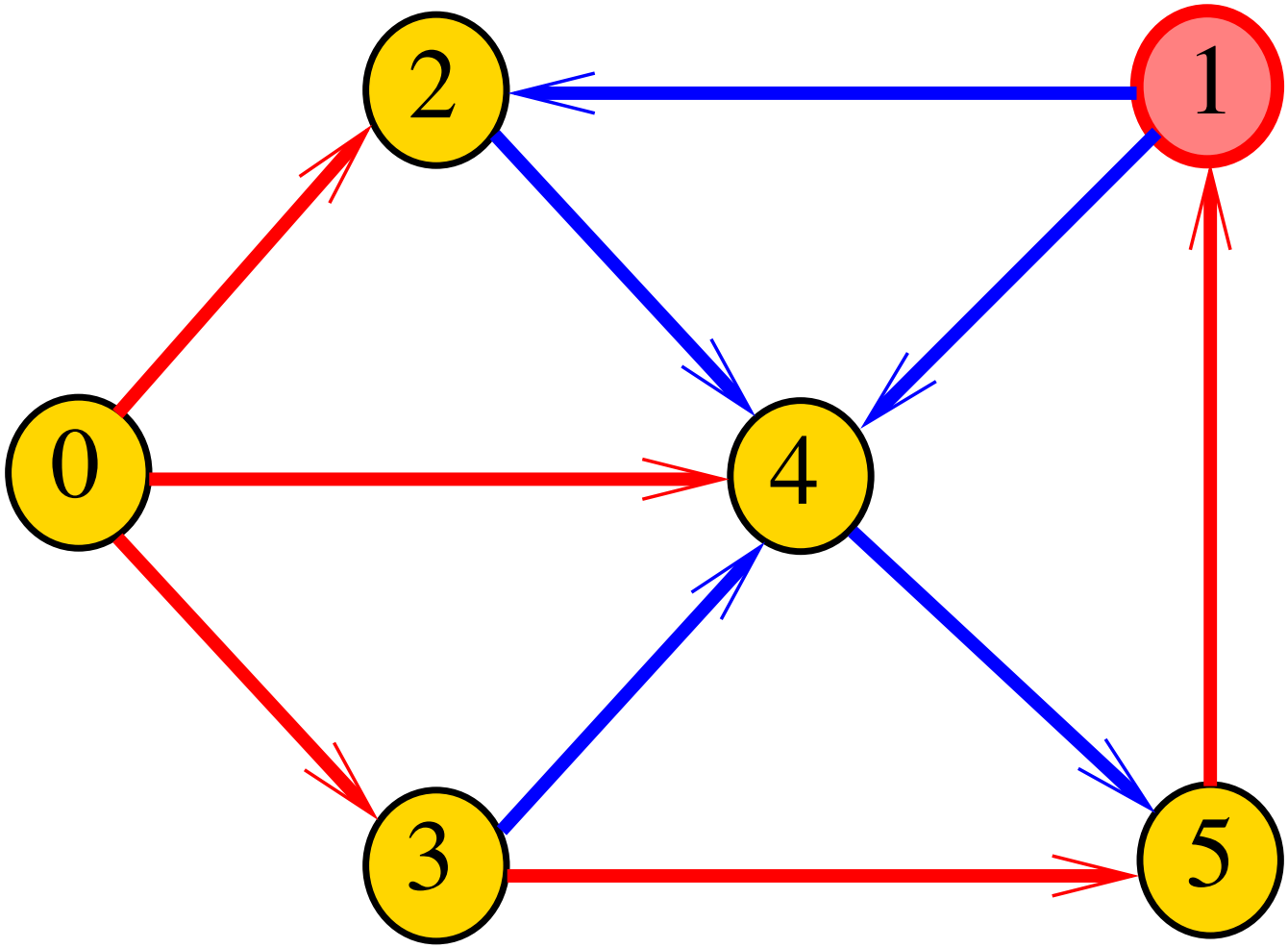
BFS(G,0)

i	0	1	2	3	4	5
q[i]	**0**	**2**	**3**	**4**	**5**	**1**



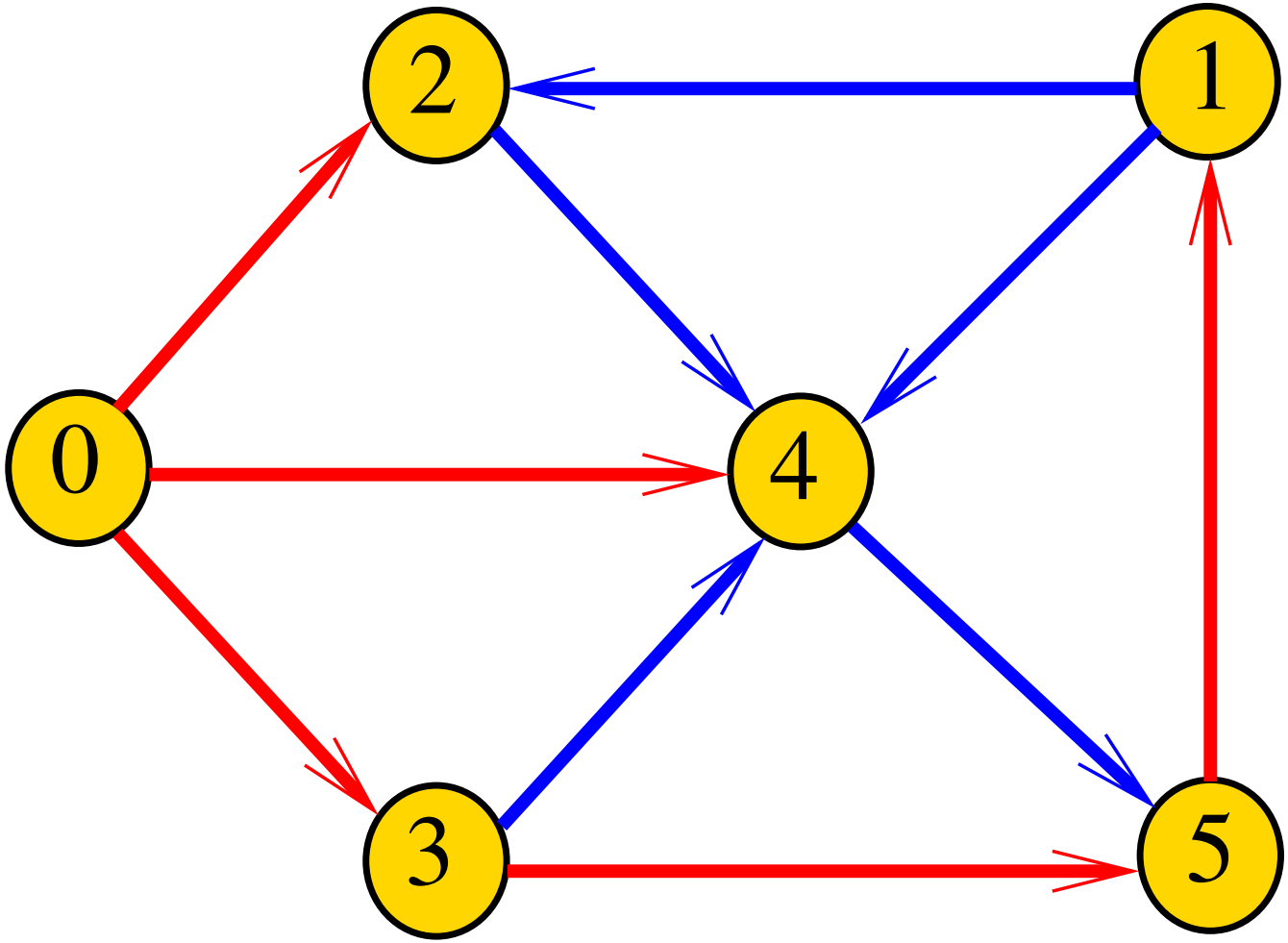
BFS(G,0)

i	0	1	2	3	4	5
q[i]	**0**	**2**	**3**	**4**	**5**	**1**



BFS(G,0)

i	0	1	2	3	4	5
q[i]	**0**	**2**	**3**	**4**	**5**	**1**



Consumo de tempo

Cada vértice é descoberto uma única vez, pois é branco
e, ao ser descoberto, passa a ser cinzento, e depois preto.

A estrela de cada vértice descoberto é
percorrida uma única vez, quando o vértice sai de Q.

Logo, com listas de adjacência, o consumo de tempo é $O(n+m)$,
pois a inicialização custa $\Theta(n)$ e
a soma do tamanho das listas de adjacências percorridas é $O(m)$.

O consumo de tempo de uma BFS é
linear no tamanho do grafo.

Algumas aplicações

- Determinar componentes conexas de um grafo

Componentes conexas

Num grafo, dizemos que um vértice u é alcançável a partir de um vértice v se existe caminho de v a u.

Algumas aplicações

- Determinar componentes conexas de um grafo

Componentes conexas

Num grafo, dizemos que um vértice u é alcançável a partir de um vértice v se existe caminho de v a u.

- Num grafo dirigido, essa relação é transitiva e reflexiva.

Algumas aplicações

- Determinar componentes conexas de um grafo

Componentes conexas

Num grafo, dizemos que um vértice u é alcançável a partir de um vértice v se existe caminho de v a u.

- Num grafo dirigido, essa relação é transitiva e reflexiva.
- Num grafo não dirigido, ela é de equivalência. Os subgrafos induzidos pelas classes são as componentes conexas do grafo.

Algumas aplicações

- Determinar componentes conexas de um grafo

Componentes conexas

Num grafo, dizemos que um vértice u é alcançável
a partir de um vértice v se existe caminho de v a u.

- Num grafo dirigido, essa relação é transitiva e reflexiva.
- Num grafo não dirigido, ela é de equivalência. Os subgrafos induzidos pelas classes são as componentes conexas do grafo.

Como achar as componentes?

Algumas aplicações

- Determinar componentes conexas de um grafo

Componentes conexas

Num grafo, dizemos que um vértice u é alcançável
a partir de um vértice v se existe caminho de v a u.

- Num grafo dirigido, essa relação é transitiva e reflexiva.
- Num grafo não dirigido, ela é de equivalência. Os subgrafos induzidos pelas classes são as componentes conexas do grafo.

Como achar as componentes?
Basta achar as classes.

Algumas aplicações

- Determinar componentes conexas de um grafo

Componentes conexas

Num grafo, dizemos que um vértice u é alcançável
a partir de um vértice v se existe caminho de v a u.

- Num grafo dirigido, essa relação é transitiva e reflexiva.
- Num grafo não dirigido, ela é de equivalência. Os subgrafos induzidos pelas classes são as componentes conexas do grafo.

Como achar as componentes?
Basta achar as classes.

BFS a partir de um vértice determina sua classe.

Algumas aplicações

- Determinar componentes conexas de um grafo

Componentes conexas

Num grafo, dizemos que um vértice u é alcançável
a partir de um vértice v se existe caminho de v a u.

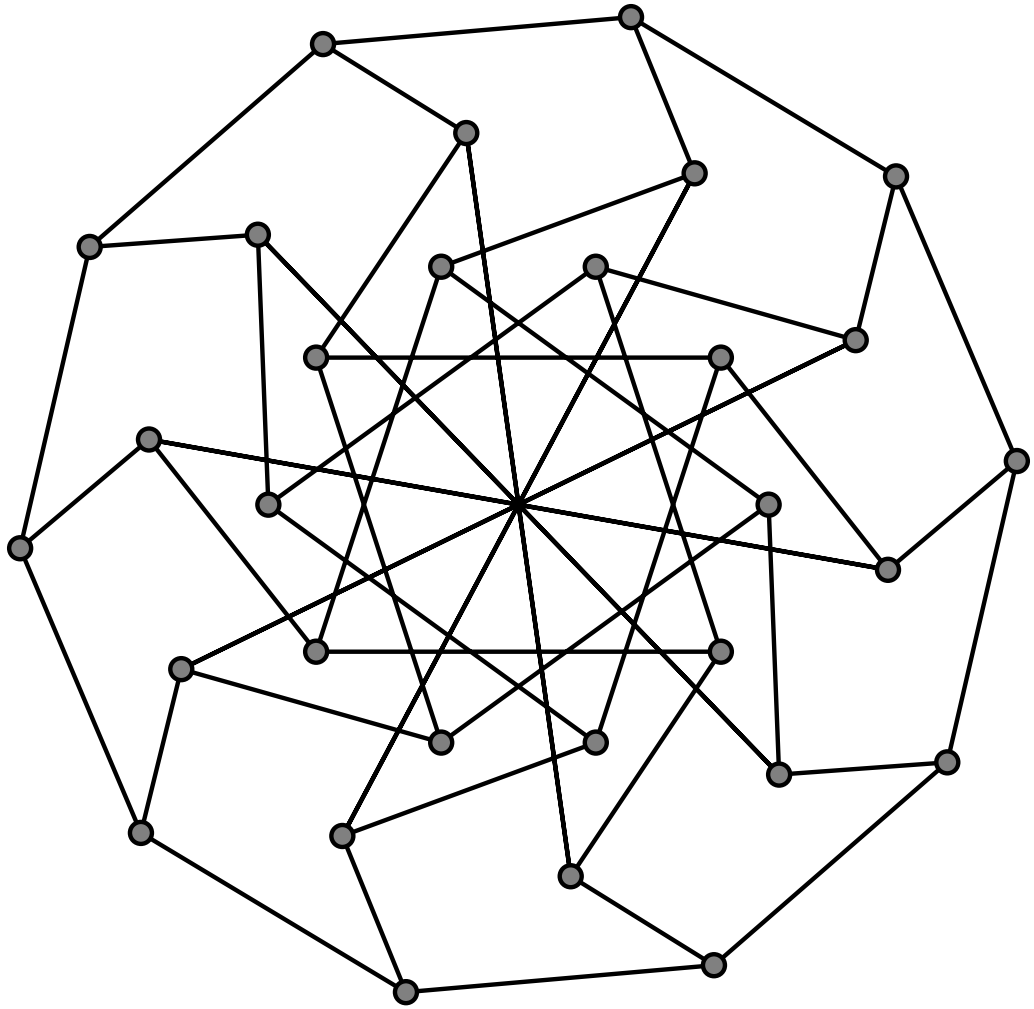
- Num grafo dirigido, essa relação é transitiva e reflexiva.
- Num grafo não dirigido, ela é de equivalência. Os subgrafos induzidos pelas classes são as componentes conexas do grafo.

Como achar as componentes?
Basta achar as classes.

BFS a partir de um vértice determina sua classe.

- Determinar se um grafo é bipartido

Grafo conexo



Grafos bipartidos

Um grafo é bipartido se é possível dividir seus vértices em duas classes de tal modo que cada aresta tem uma ponta em cada classe.

Grafos bipartidos

Um grafo é bipartido se é possível dividir seus vértices em duas classes de tal modo que cada aresta tem uma ponta em cada classe.

ALGORITMO - Decide se G (conexo) é bipartido

Grafos bipartidos

Um grafo é bipartido se é possível dividir seus vértices em duas classes de tal modo que cada aresta tem uma ponta em cada classe.

ALGORITMO - Decide se G (conexo) é bipartido

Faça uma BFS. Pequena alteração: Se $v.d == u.d$, devolva NÃO.
Se terminou, a bipartição é dada pela paridade do d.

Grafos bipartidos

Um grafo é bipartido se é possível dividir seus vértices em duas classes de tal modo que cada aresta tem uma ponta em cada classe.

ALGORITMO - Decide se G (conexo) é bipartido

Faça uma BFS. Pequena alteração: Se $v.d == u.d$, devolva NÃO.
Se terminou, a bipartição é dada pela paridade do d.

O algoritmo tem a mesma complexidade da BFS.
A corretude vem junto com o

Grafos bipartidos

Um grafo é bipartido se é possível dividir seus vértices em duas classes de tal modo que cada aresta tem uma ponta em cada classe.

ALGORITMO - Decide se G (conexo) é bipartido

Faça uma BFS. Pequena alteração: Se $v.d == u.d$, devolva NÃO.
Se terminou, a bipartição é dada pela paridade do d.

O algoritmo tem a mesma complexidade da BFS.
A corretude vem junto com o

Teorema
Um grafo é bipartido se e só se não tem um circuito ímpar.

Grafos bipartidos

Um grafo é bipartido se é possível dividir seus vértices em duas classes de tal modo que cada aresta tem uma ponta em cada classe.

ALGORITMO - Decide se G (conexo) é bipartido

Faça uma BFS. Pequena alteração: Se $v.d == u.d$, devolva NÃO.
Se terminou, a bipartição é dada pela paridade do d.

O algoritmo tem a mesma complexidade da BFS.
A corretude vem junto com o

Teorema
Um grafo é bipartido se e só se não tem um circuito ímpar.

Prova: "$\Leftarrow$" é fácil.

Grafos bipartidos

Um grafo é bipartido se é possível dividir seus vértices em duas classes de tal modo que cada aresta tem uma ponta em cada classe.

ALGORITMO - Decide se G (conexo) é bipartido

Faça uma BFS. Pequena alteração: Se $v.d == u.d$, devolva NÃO.
Se terminou, a bipartição é dada pela paridade do d.

O algoritmo tem a mesma complexidade da BFS.
A corretude vem junto com o

Teorema
Um grafo é bipartido se e só se não tem um circuito ímpar.

Prova: "$\Leftarrow$" é fácil.
"$\Rightarrow$": modificar o algoritmo para
devolver um circuito ímpar no caso NÃO.

Grafos tripartidos

Um grafo é tripartido se é possível dividir seus vértices em três classes de tal modo que cada aresta tenha as pontas em classes diferentes.

Grafos tripartidos

Um grafo é tripartido se é possível dividir seus vértices em três classes de tal modo que cada aresta tenha as pontas em classes diferentes.

Exercício

Encontrar um algoritmo polinomial
para decidir se um grafo é tripartido.

Grafos tripartidos

Um grafo é tripartido se é possível dividir seus vértices em três classes de tal modo que cada aresta tenha as pontas em classes diferentes.

Exercício

Encontrar um algoritmo polinomial
para decidir se um grafo é tripartido.

PRÊMIOS

Grafos tripartidos

Um grafo é tripartido se é possível dividir seus vértices em três classes de tal modo que cada aresta tenha as pontas em classes diferentes.

Exercício

Encontrar um algoritmo polinomial
para decidir se um grafo é tripartido.

PRÊMIOS

Nota A na disciplina

Grafos tripartidos

Um grafo é tripartido se é possível dividir seus vértices em três classes de tal modo que cada aresta tenha as pontas em classes diferentes.

Exercício

Encontrar um algoritmo polinomial
para decidir se um grafo é tripartido.

PRÊMIOS

Nota A na disciplina

+

Grafos tripartidos

Um grafo é tripartido se é possível dividir seus vértices em três classes de tal modo que cada aresta tenha as pontas em classes diferentes.

Exercício

Encontrar um algoritmo polinomial
para decidir se um grafo é tripartido.

PRÊMIOS

Nota A na disciplina

+

Título de Doutor

Grafos tripartidos

Um grafo é tripartido se é possível dividir seus vértices em três classes de tal modo que cada aresta tenha as pontas em classes diferentes.

Exercício

Encontrar um algoritmo polinomial
para decidir se um grafo é tripartido.

PRÊMIOS

Nota A na disciplina

+

Título de Doutor

+

Grafos tripartidos

Um grafo é tripartido se é possível dividir seus vértices em três classes de tal modo que cada aresta tenha as pontas em classes diferentes.

Exercício

Encontrar um algoritmo polinomial
para decidir se um grafo é tripartido.

PRÊMIOS

Nota A na disciplina

+

Título de Doutor

+

US$$10^6$$

Busca em profundidade

DFS

Outra moldura de algoritmos
que vão fazendo seu serviço enquanto percorrem o grafo.

Busca em profundidade

DFS

Outra moldura de algoritmos
que vão fazendo seu serviço enquanto percorrem o grafo.

Produz uma estrutura bem mais sofisticada que a BFS.

Busca em profundidade - elementos

Vértice branco: ainda não descoberto

Vértice cinzento: descoberto, em processamento

Vértice preto: processado

Busca em profundidade - elementos

Vértice branco: ainda não descoberto

Vértice cinzento: descoberto, em processamento

Vértice preto: processado

DFS termina descrevendo via π uma floresta DF (Ou BP).

$u.\pi$: predecessor ou pai de u na floresta DF

Busca em profundidade - elementos

Vértice branco: ainda não descoberto

Vértice cinzento: descoberto, em processamento

Vértice preto: processado

DFS termina descrevendo via π uma floresta DF (Ou BP).

$u.\pi$: predecessor ou pai de u na floresta DF

Faz duas marcas de tempo:

$u.d$: momento da descoberta de u
$u.f$: momento da finalização de u

Busca em profundidade - elementos

Vértice branco: ainda não descoberto

Vértice cinzento: descoberto, em processamento

Vértice preto: processado

DFS termina descrevendo via π uma floresta DF (Ou BP).

$u.\pi$: predecessor ou pai de u na floresta DF

Faz duas marcas de tempo:

$u.d$: momento da descoberta de u
$u.f$: momento da finalização de u

u é branco antes de $u.d$,
cinzento entre $u.d$ e $u.f$,
preto depois de $u.f$.

Busca em profundidade

```
DFS(G)
 1  para cada u ∈ V(G) faça
 2       u.cor ← branco        u.π ← nil
 3  tempo ← 0
 4  para cada u ∈ V(G) faça
 5       se u.cor = branco
 6            então DFS-Visit(G, u)
```

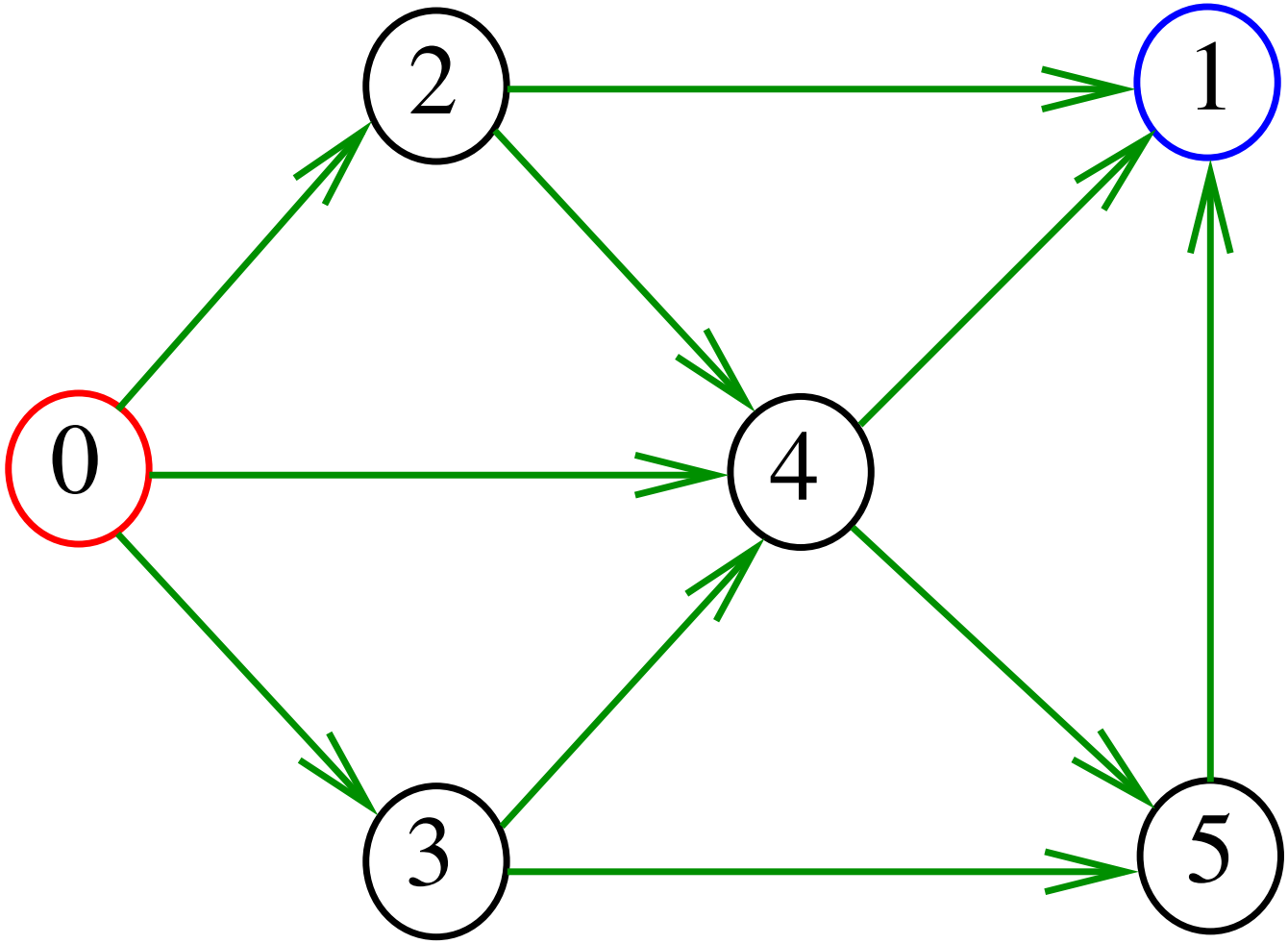
Busca em profundidade

```
DFS(G)
 1  para cada u ∈ V(G) faça
 2       u.cor ← branco        u.π ← nil
 3  tempo ← 0
 4  para cada u ∈ V(G) faça
 5       se u.cor = branco
 6            então DFS-Visit(G, u)
```

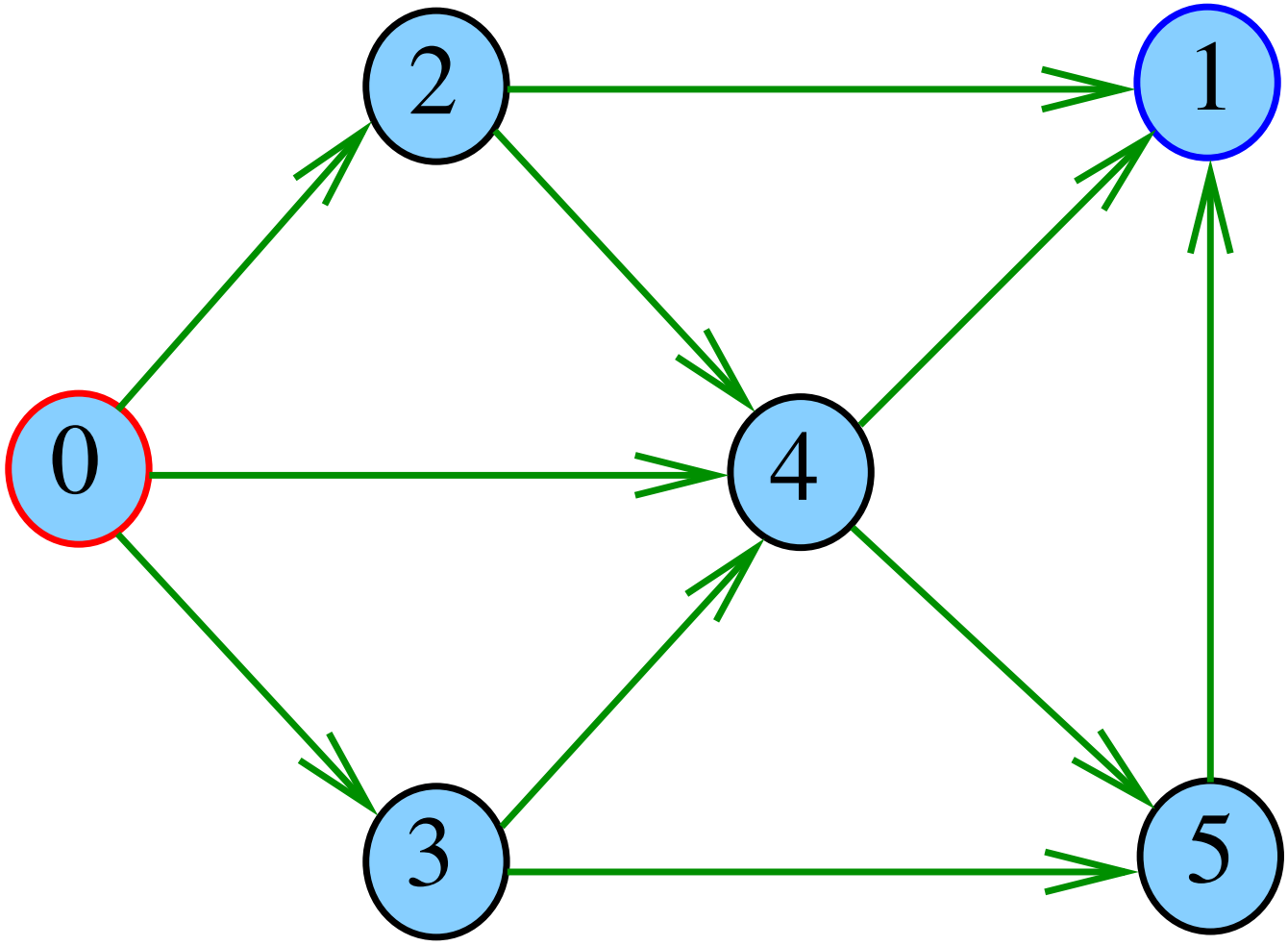
```
DFS-Visit(G, u)
 1  u.cor ← cinzento    u.d ← tempo    tempo ← tempo + 1
 3  para cada v ∈ u.Estrela faça
 4       se v.cor = branco
 5            então v.π ← u
 6                  DFS-Visit(G, v)
 7  u.cor ← preto    u.f ← tempo    tempo ← tempo + 1
```

`dfs`$(G, 0)$

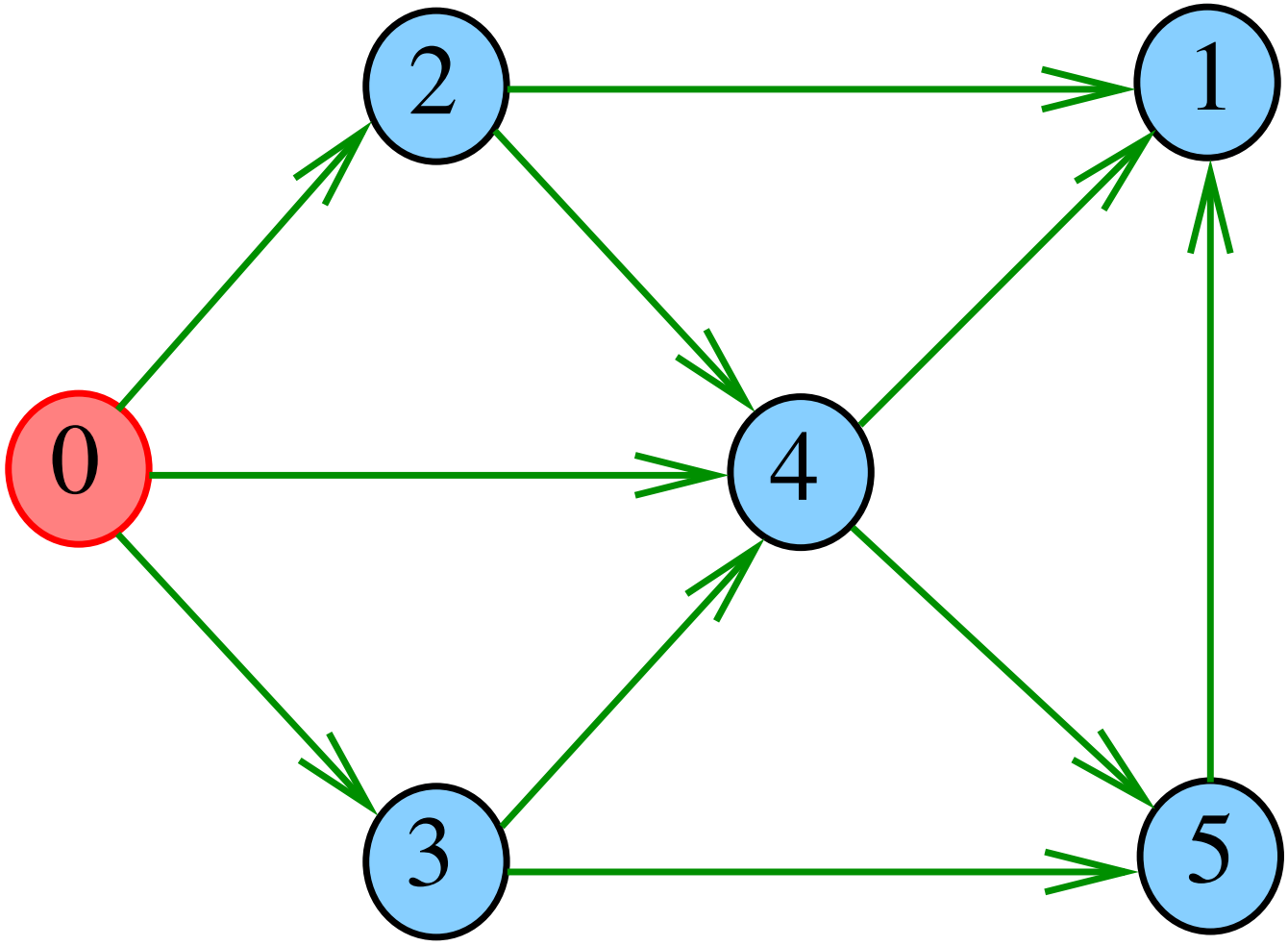
Existe caminho de 0 a 1?



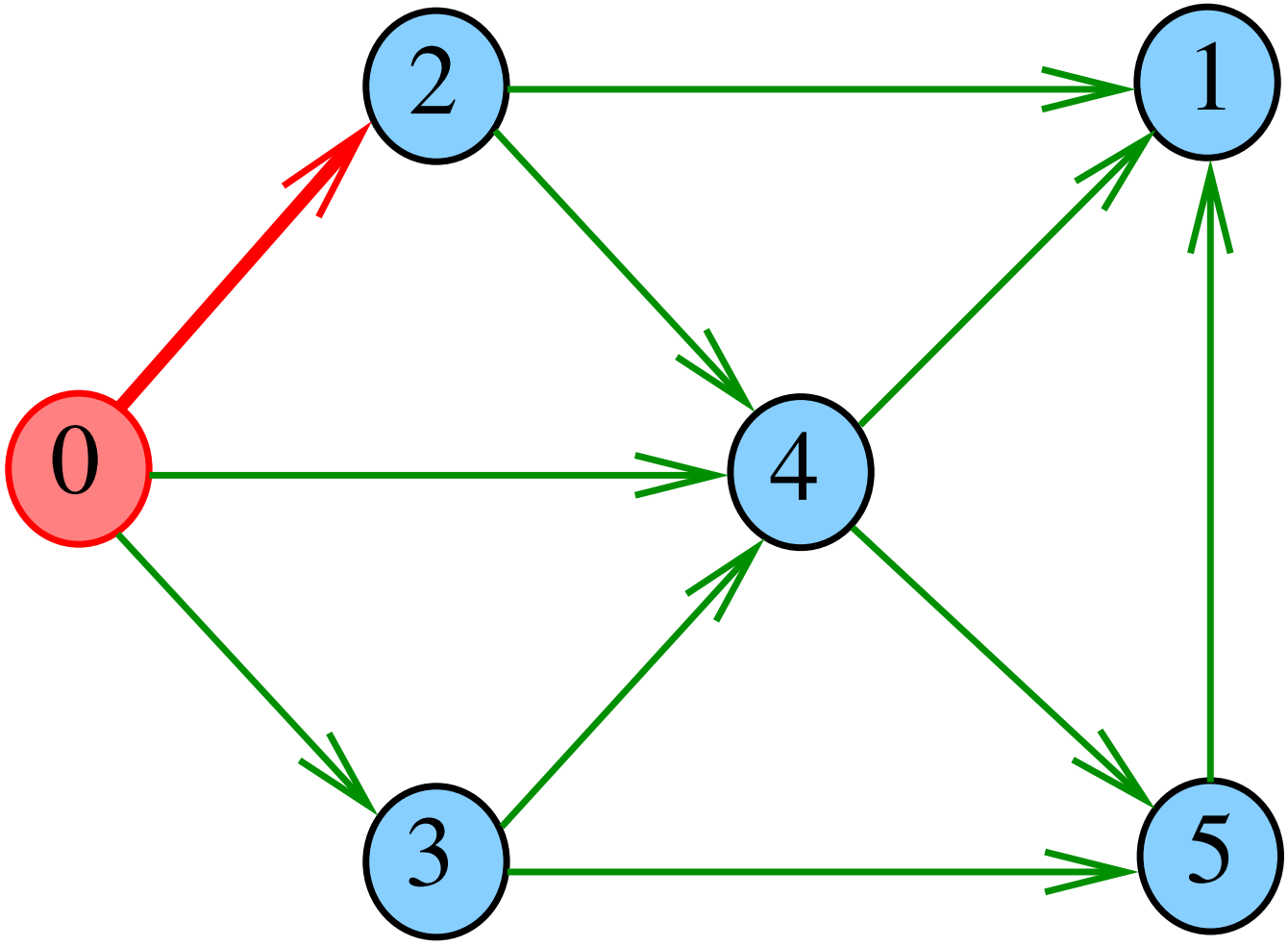
dfs(G, 0)



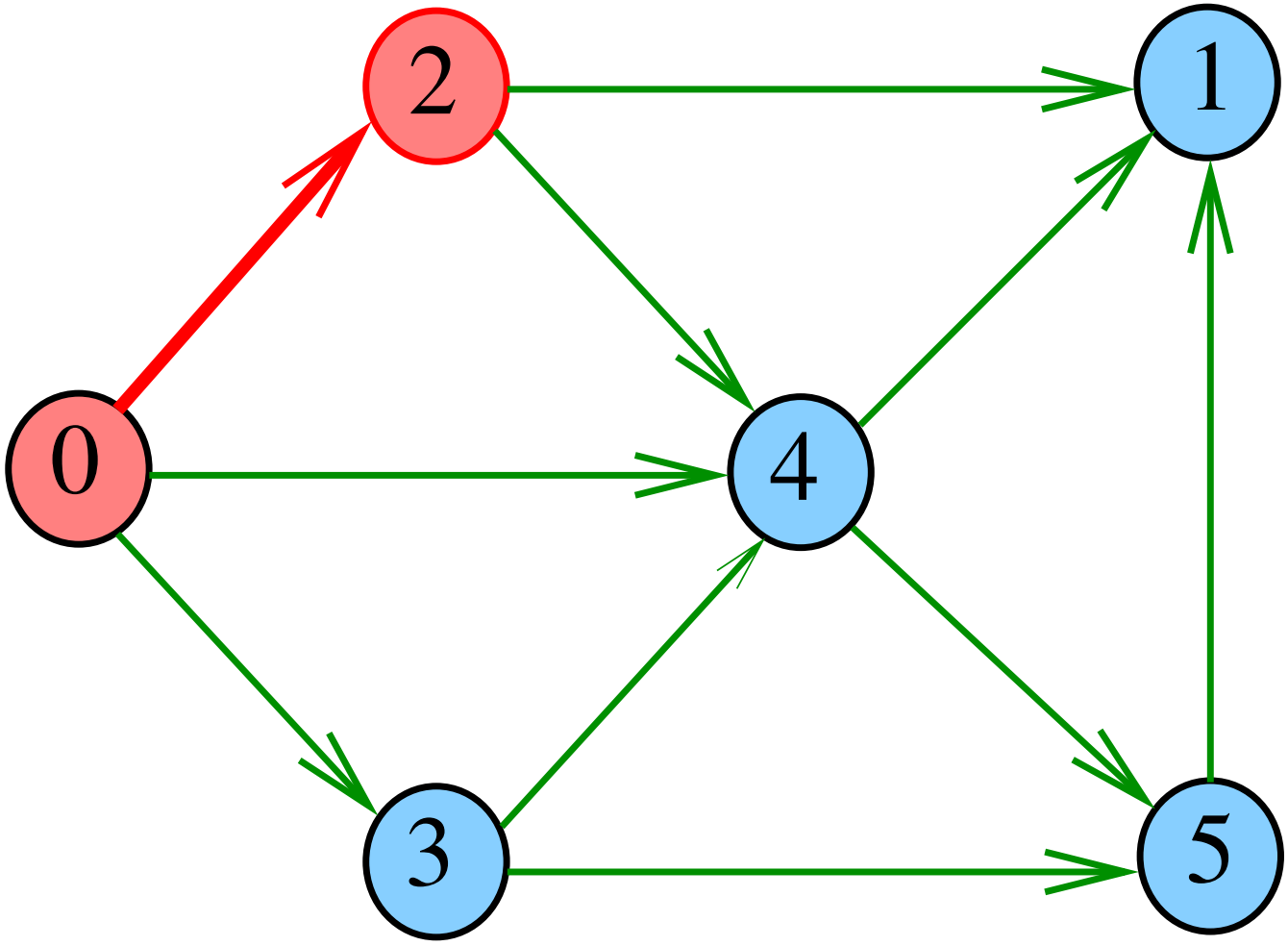
dfs(G, 0)



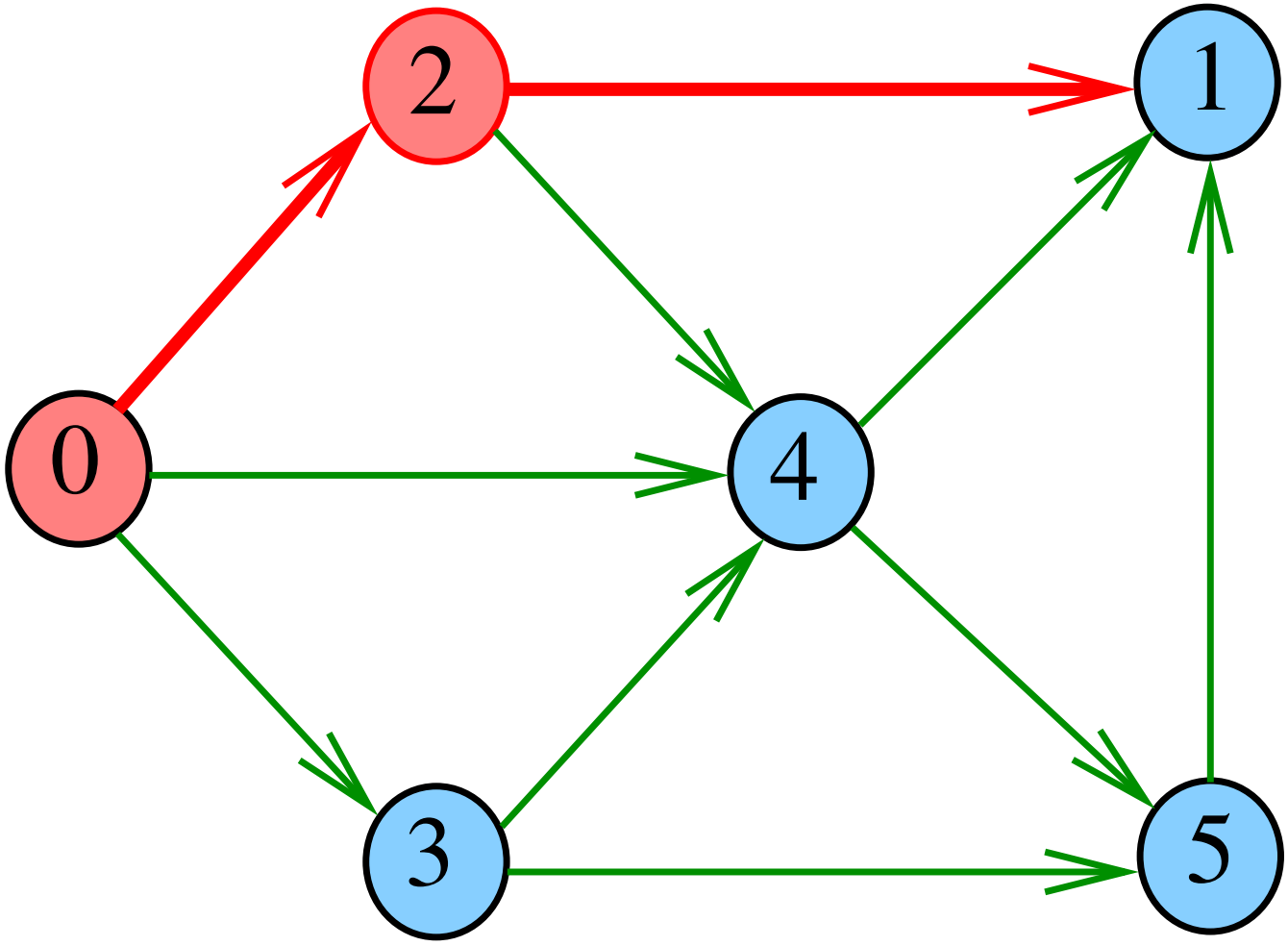
dfs(G, 0)



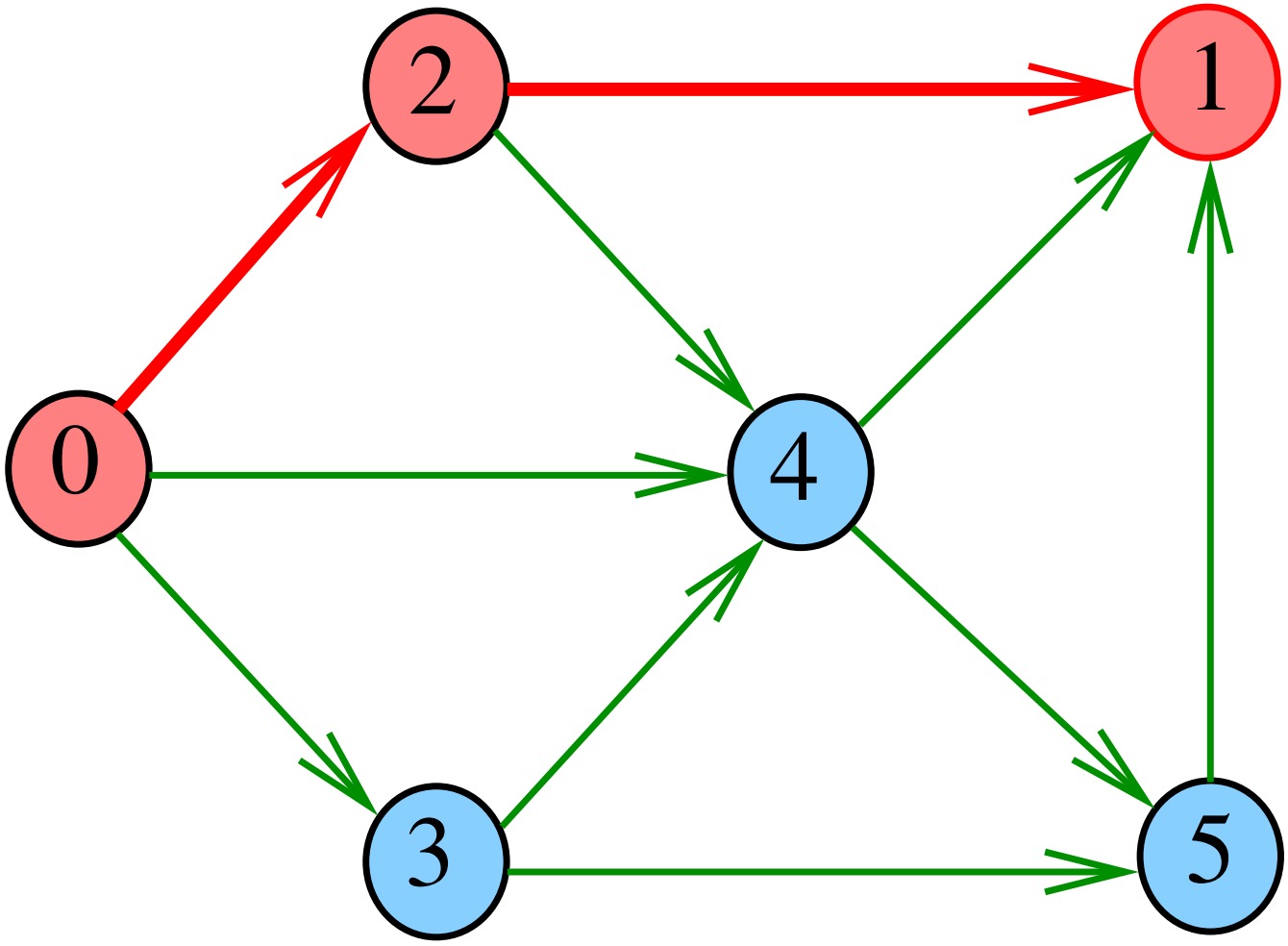
dfs(G, 2)



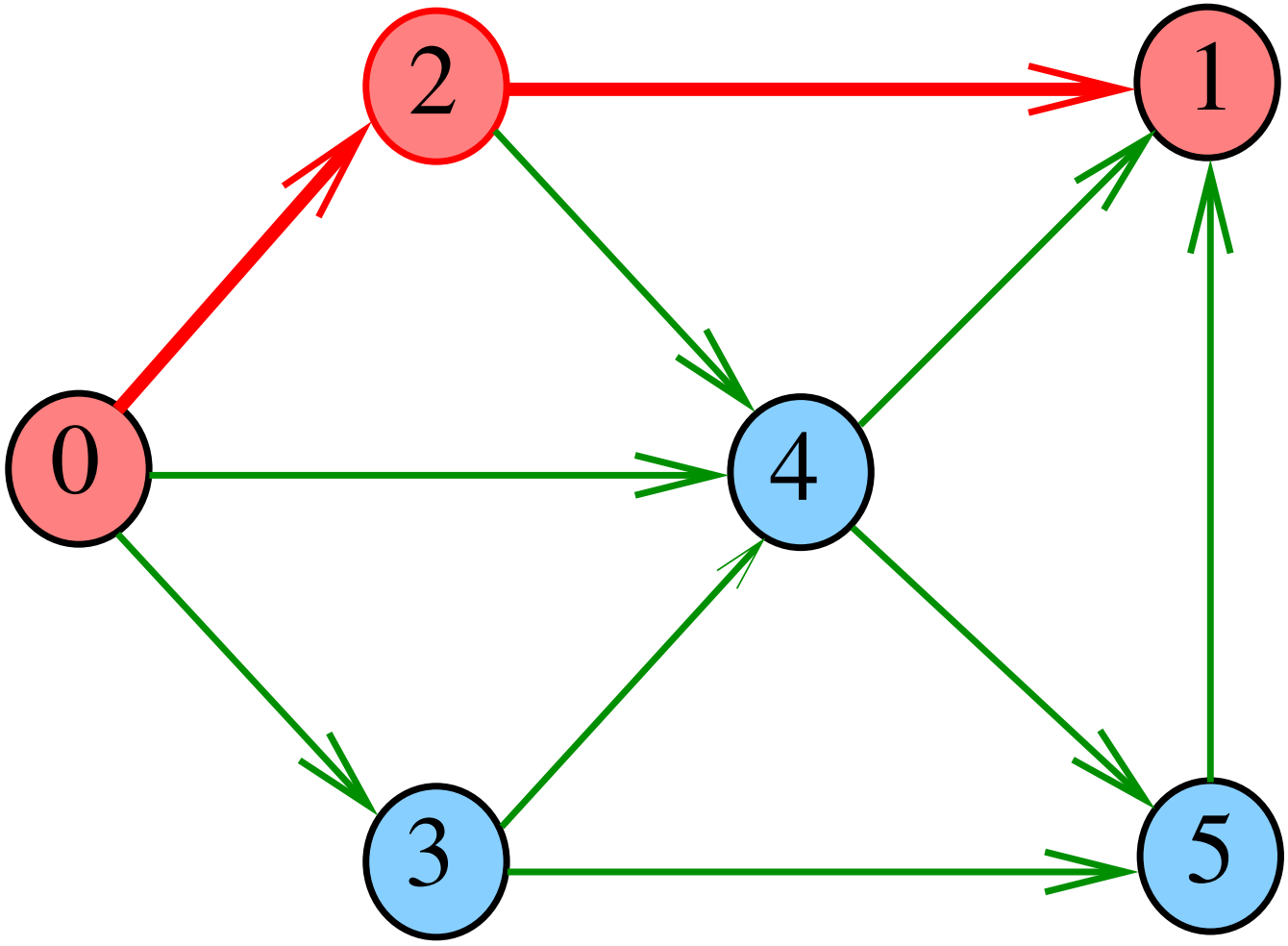
dfs(G, 2)



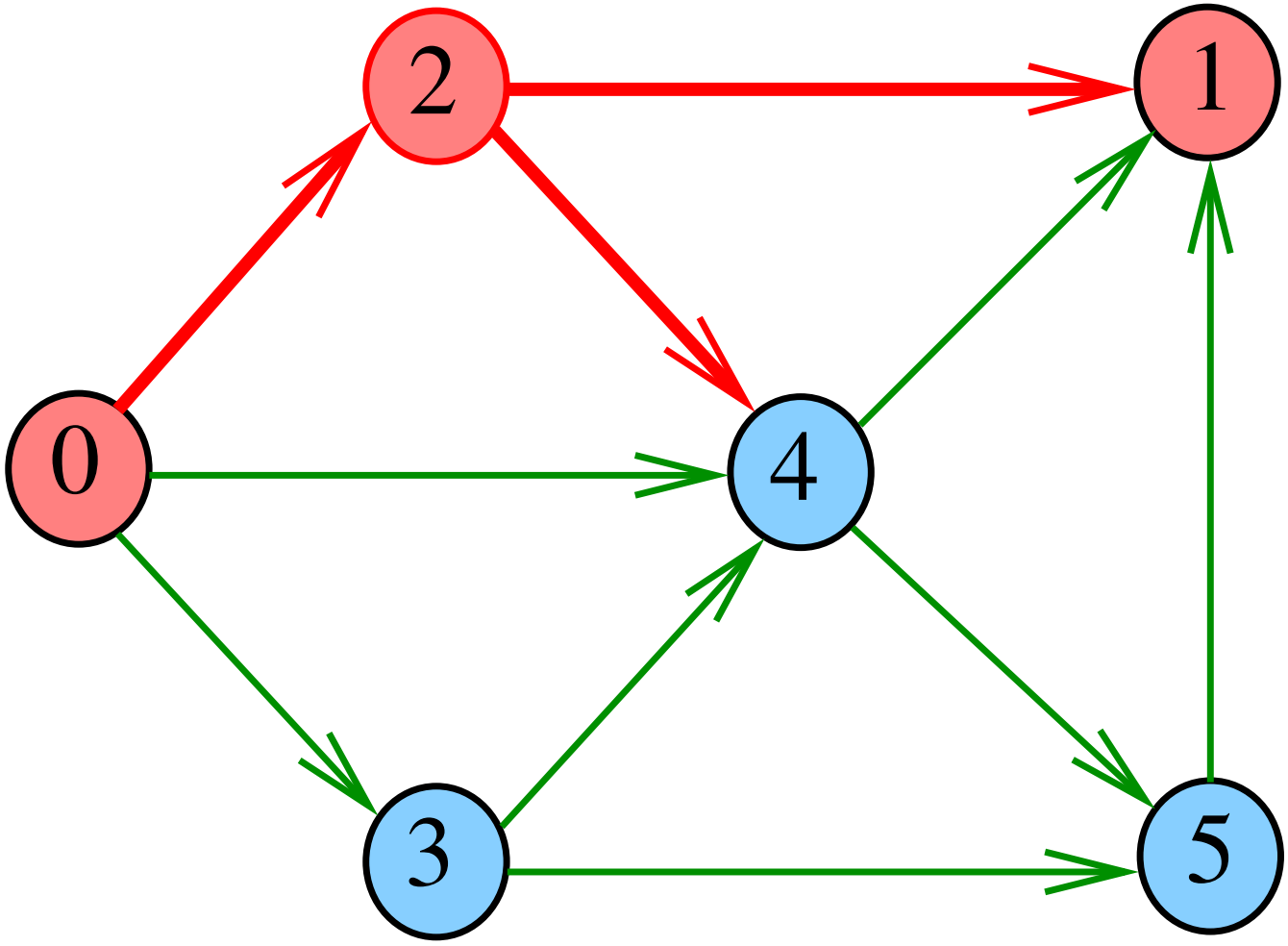
dfs(G, 1)



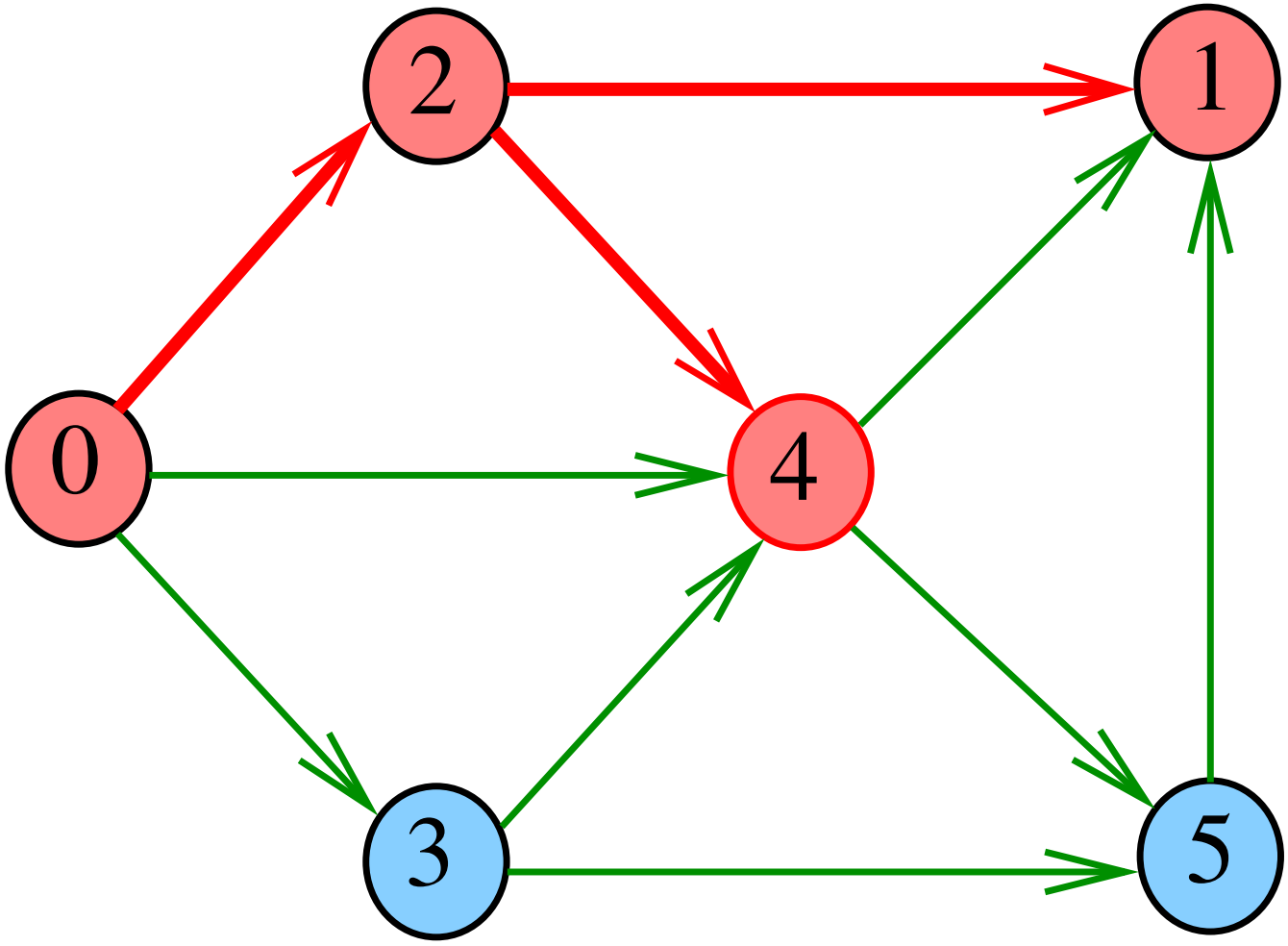
dfs(G, 2)



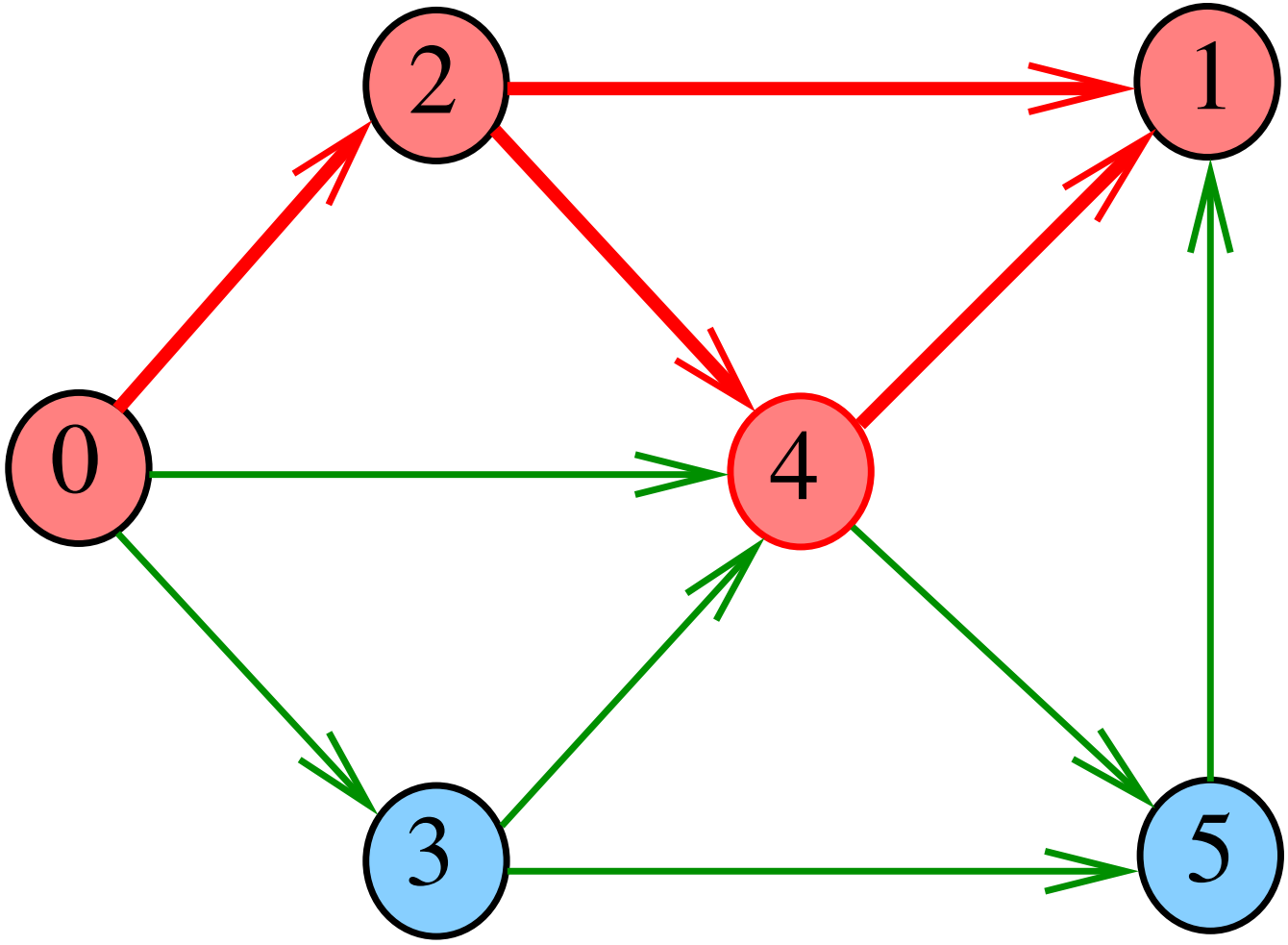
dfs(G, 2)



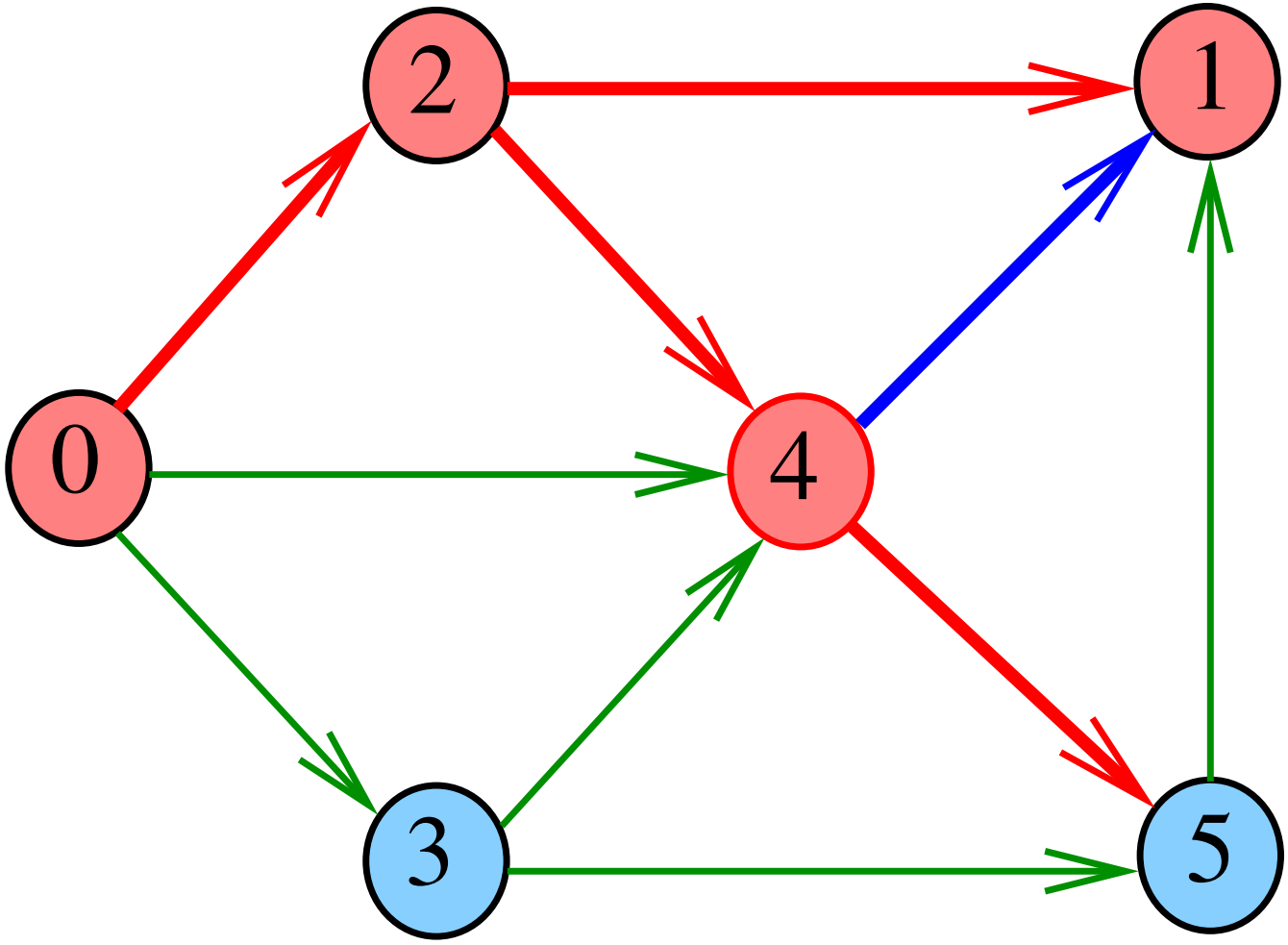
dfs(G, 4)



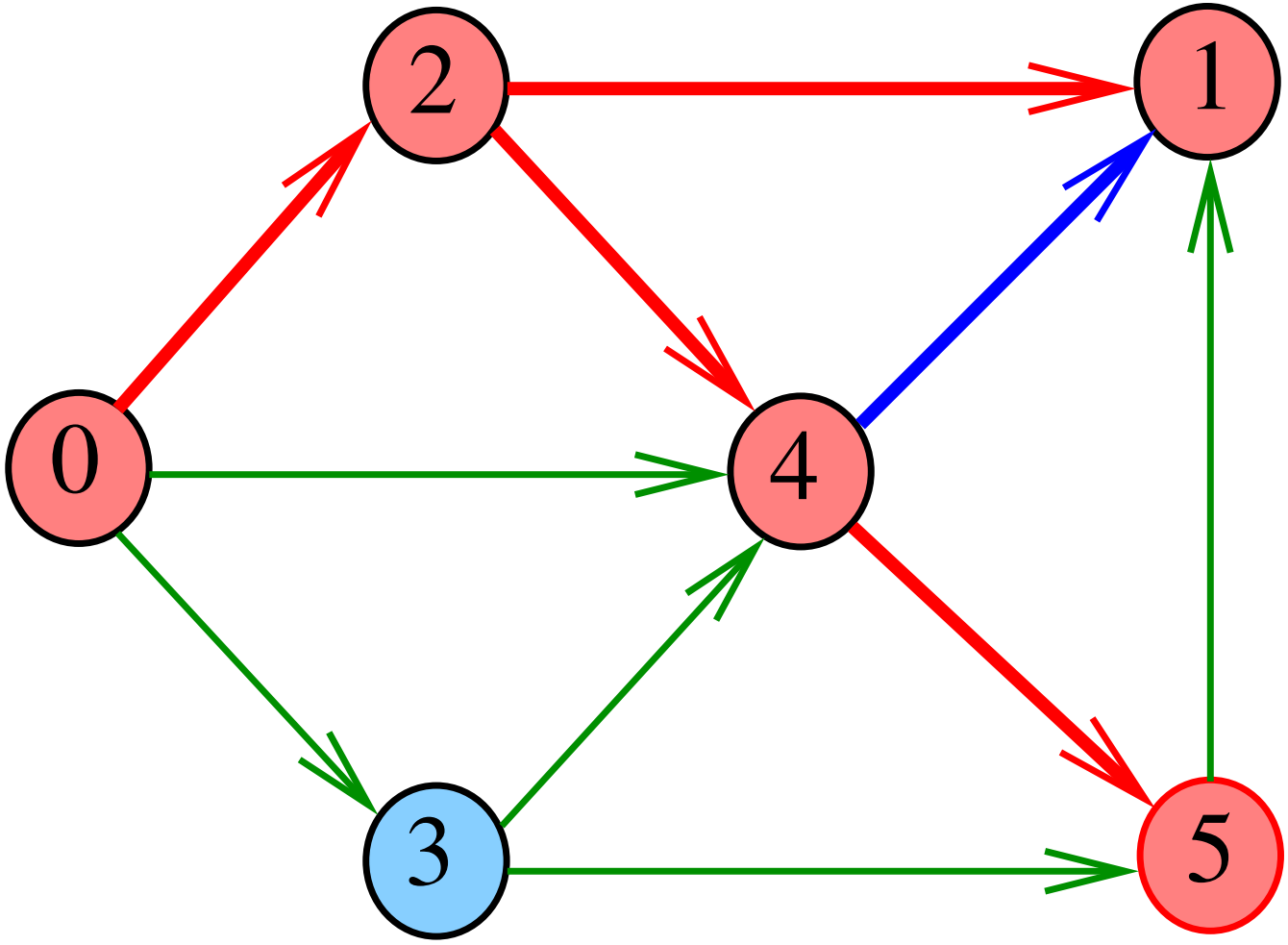
dfs(G, 4)



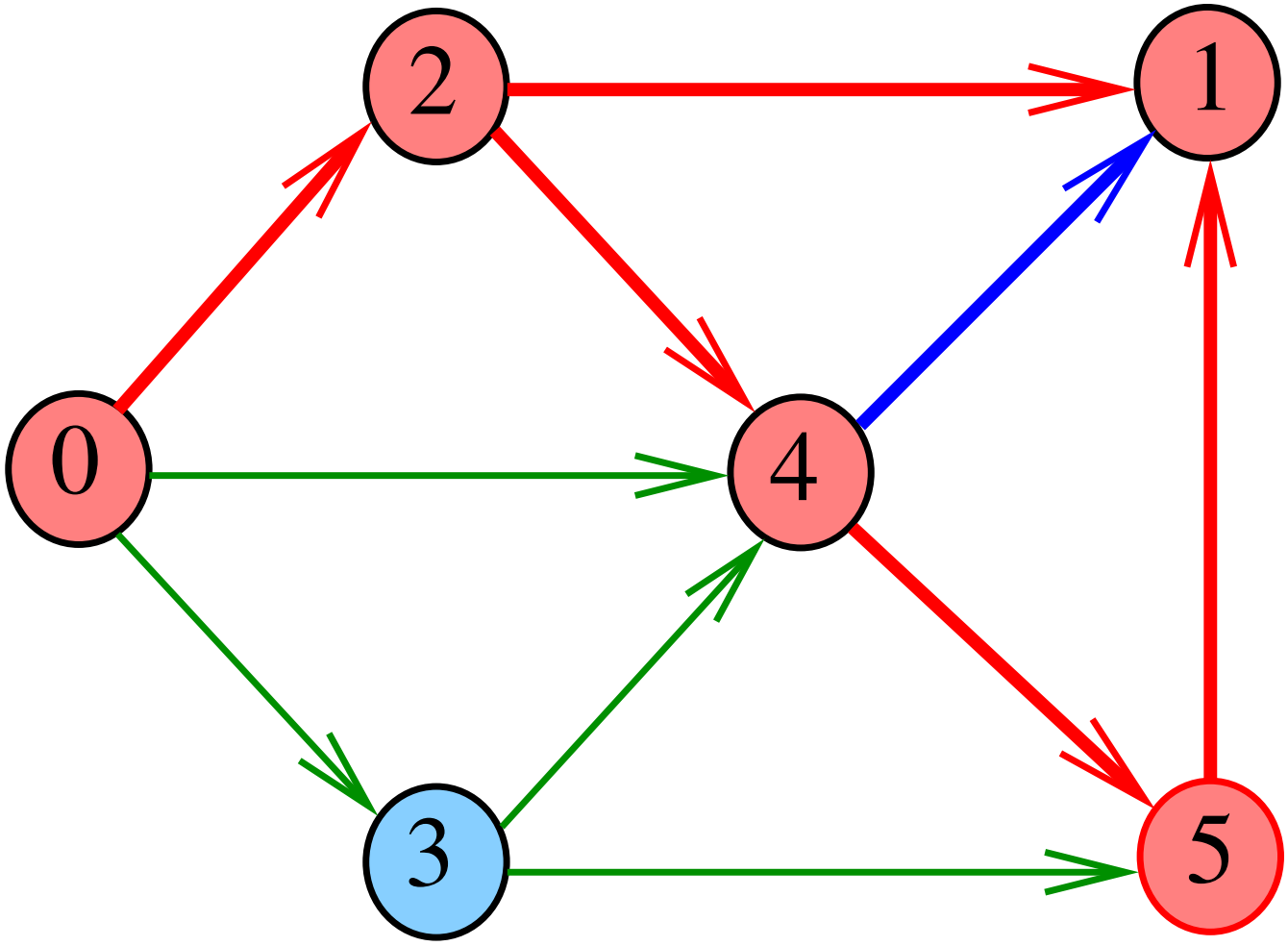
dfs(G, 4)



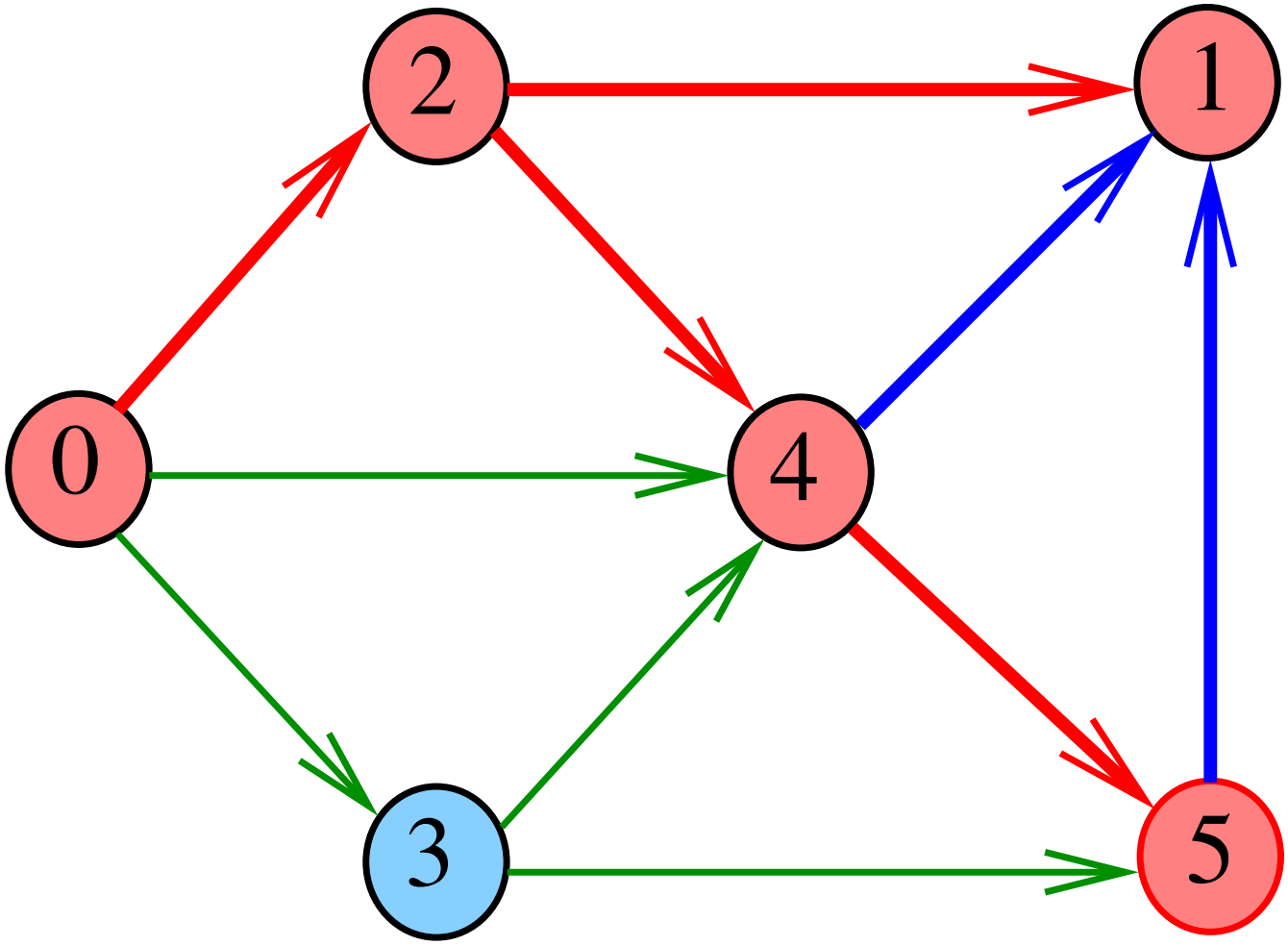
dfs(G, 5)



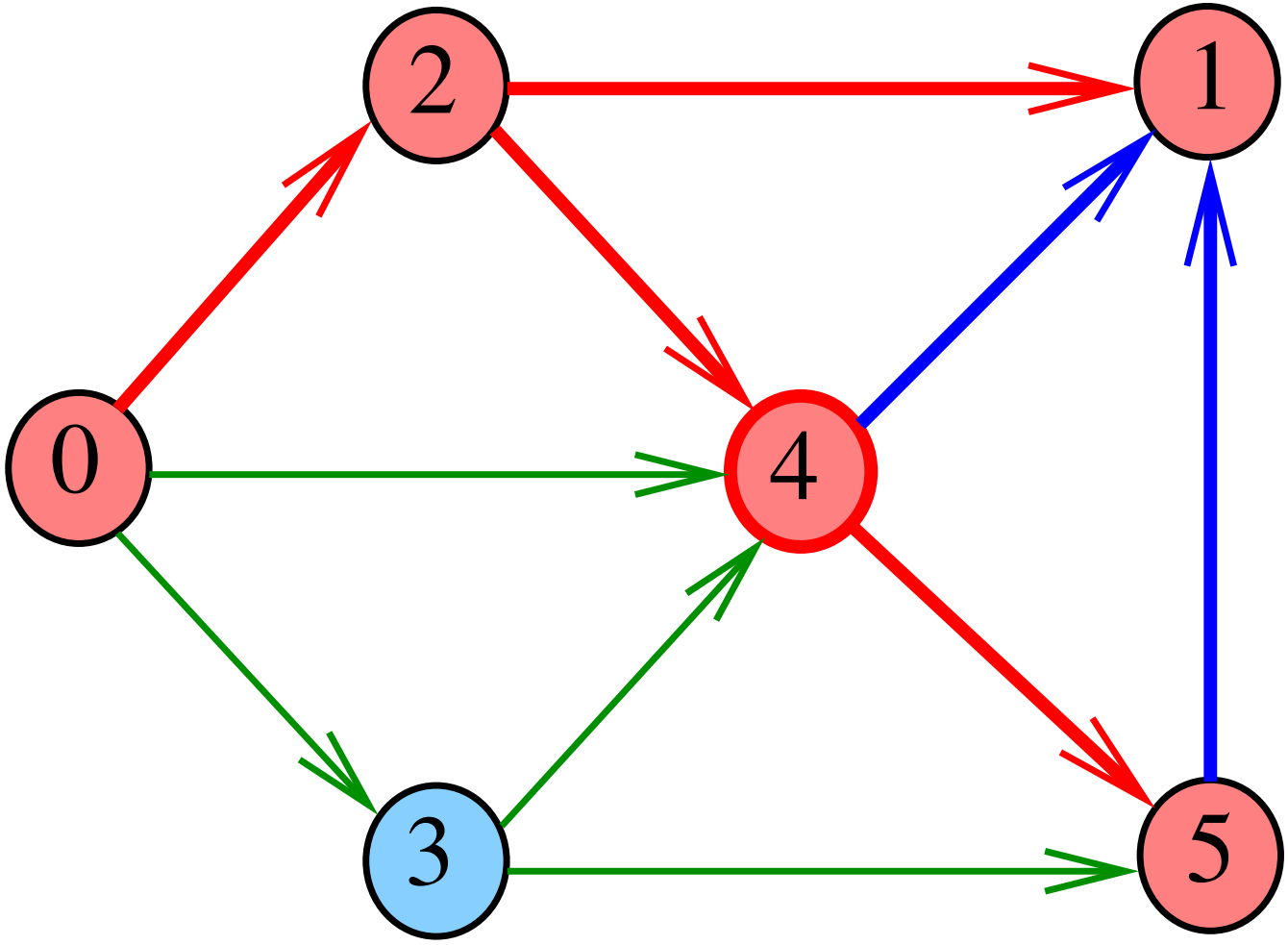
dfs(G, 5)



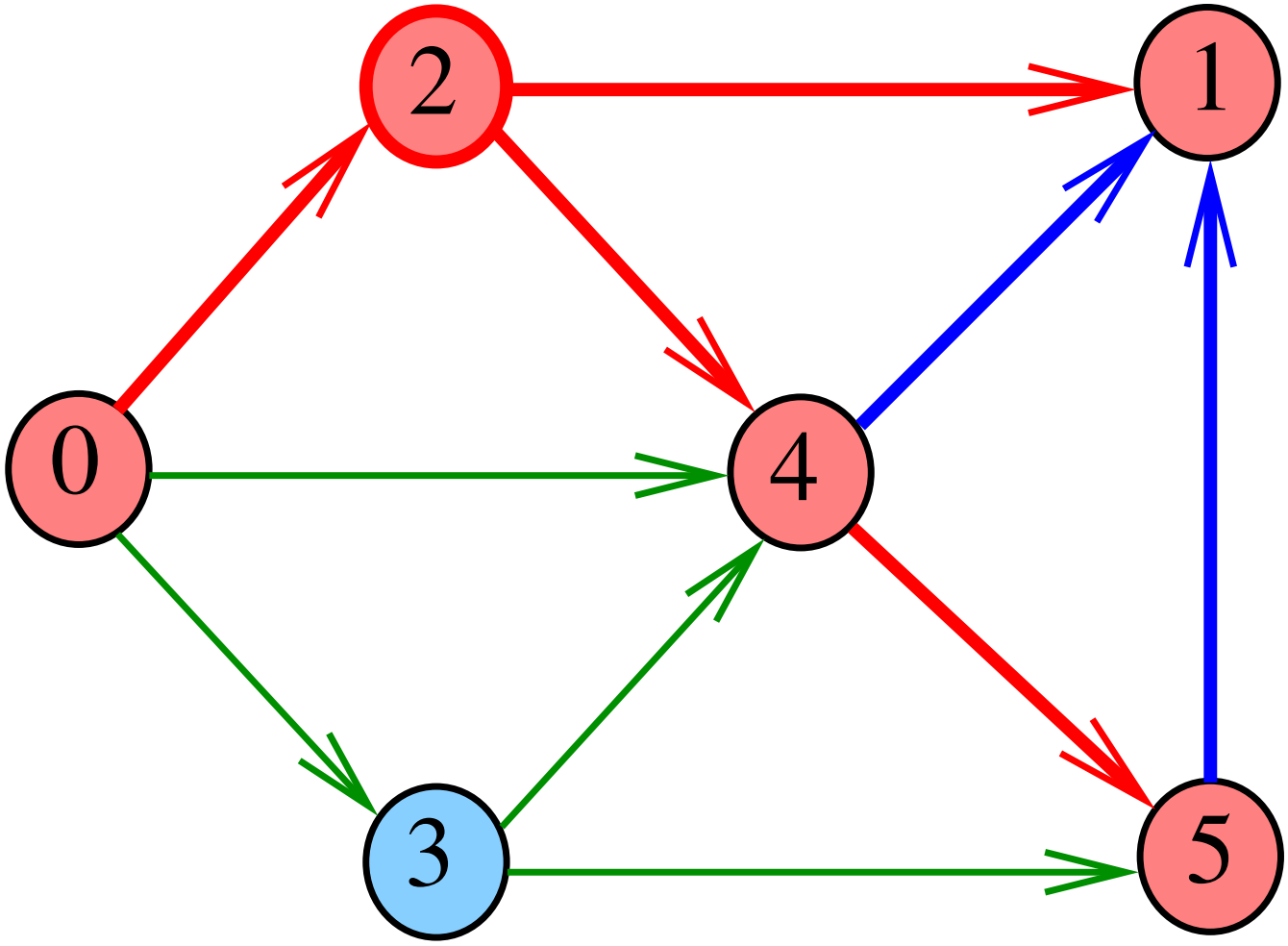
dfs(G, 5)



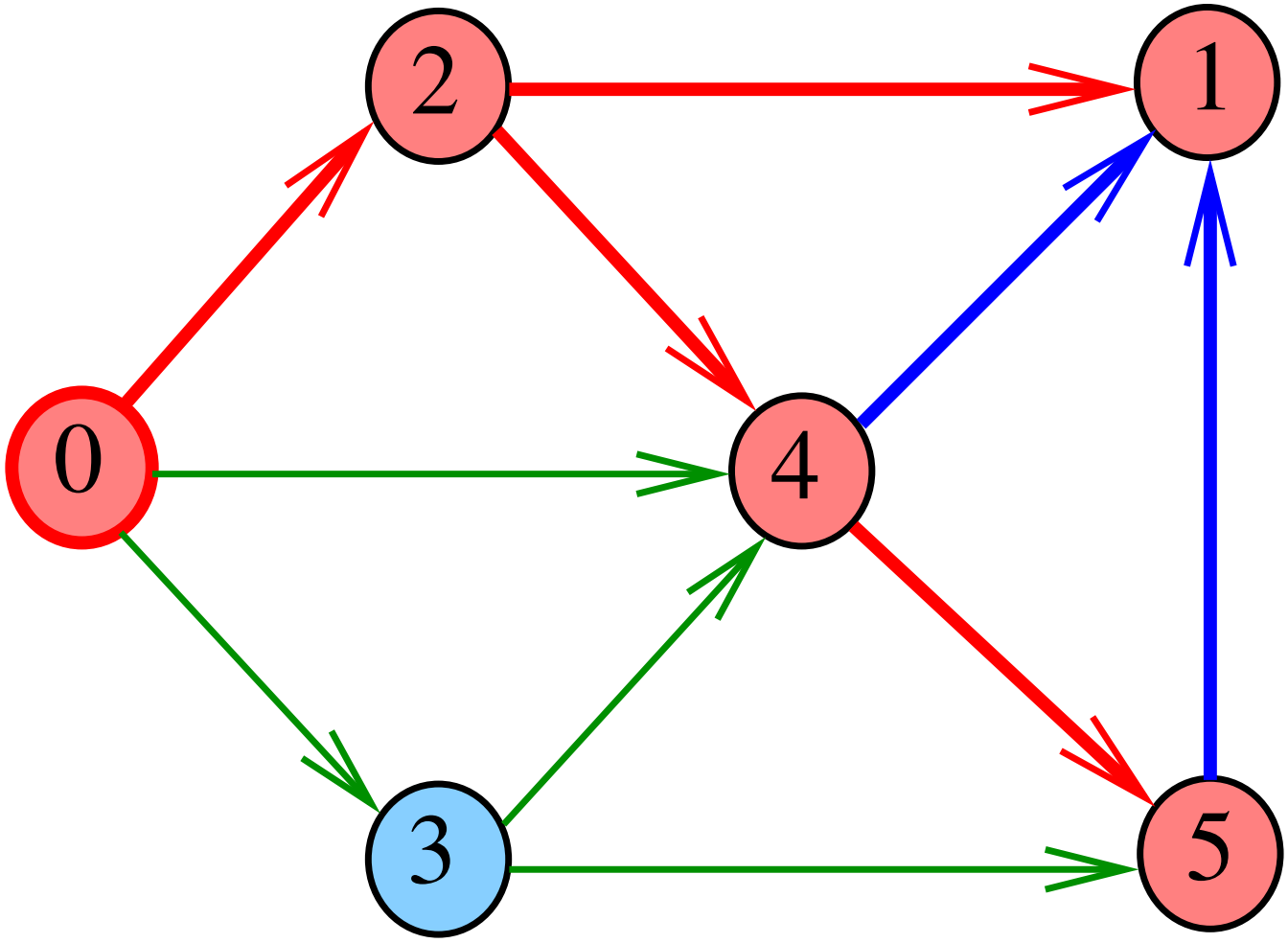
dfs(G, 4)



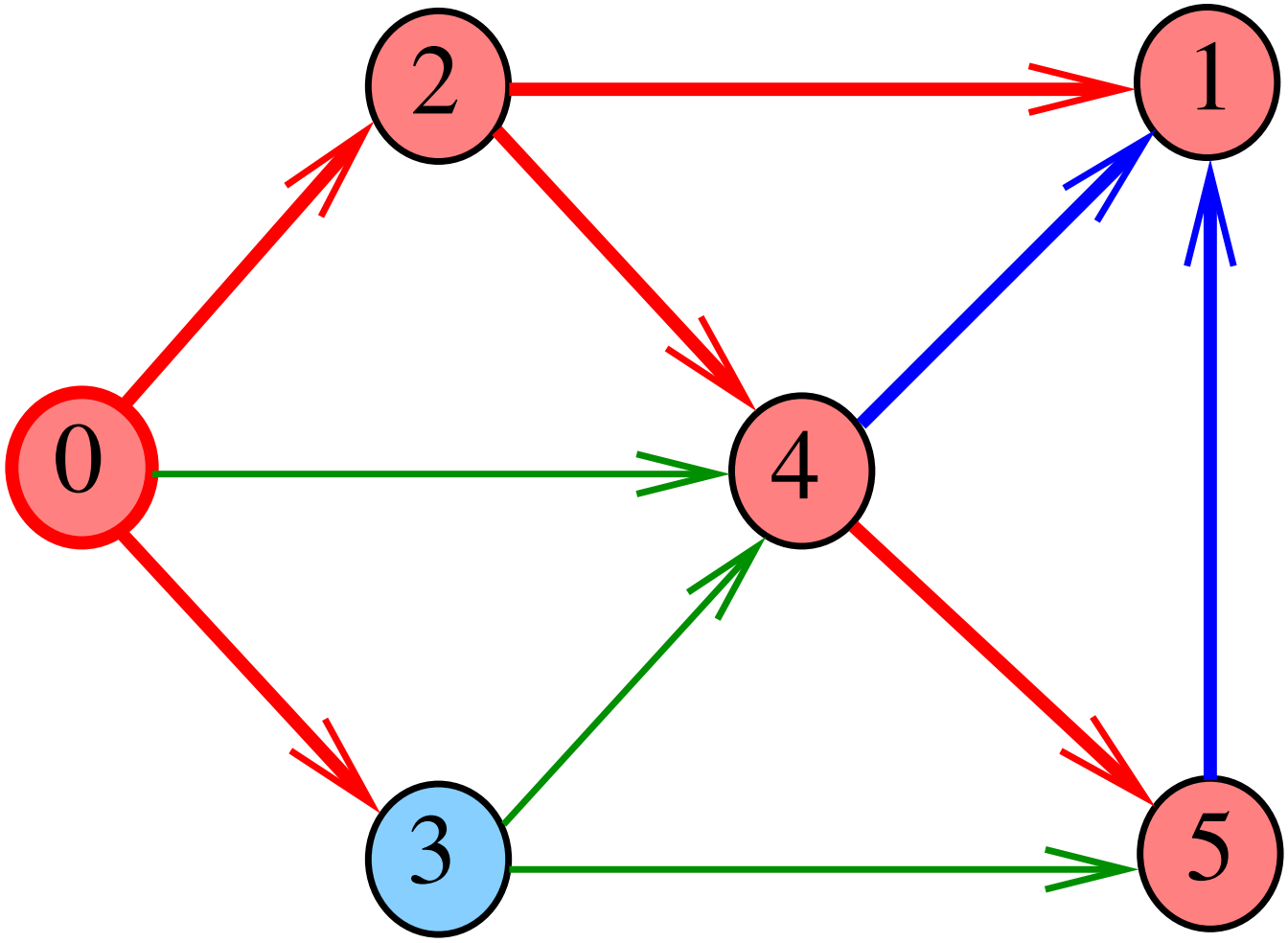
dfs(G, 2)



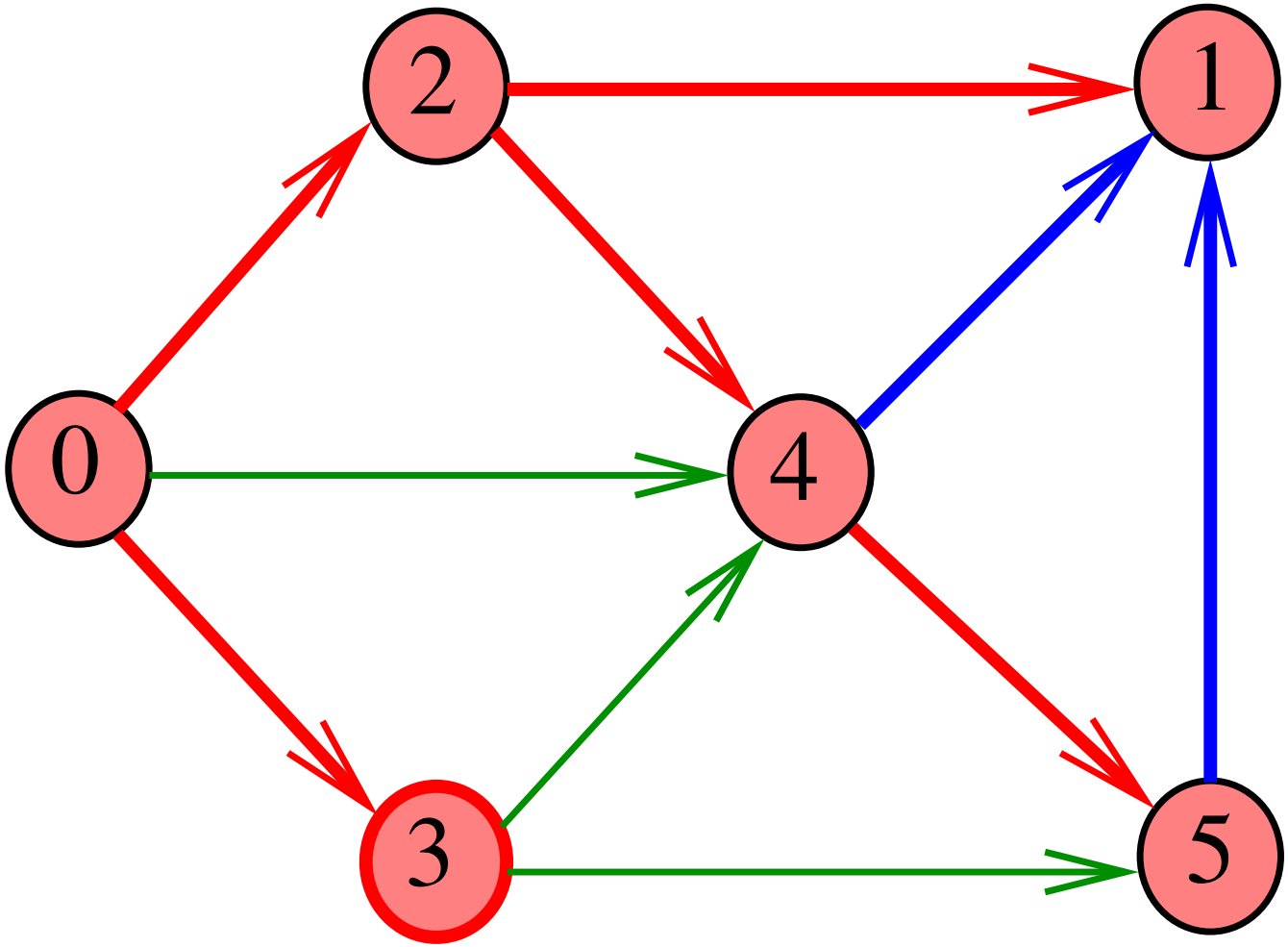
dfs(G, 0)



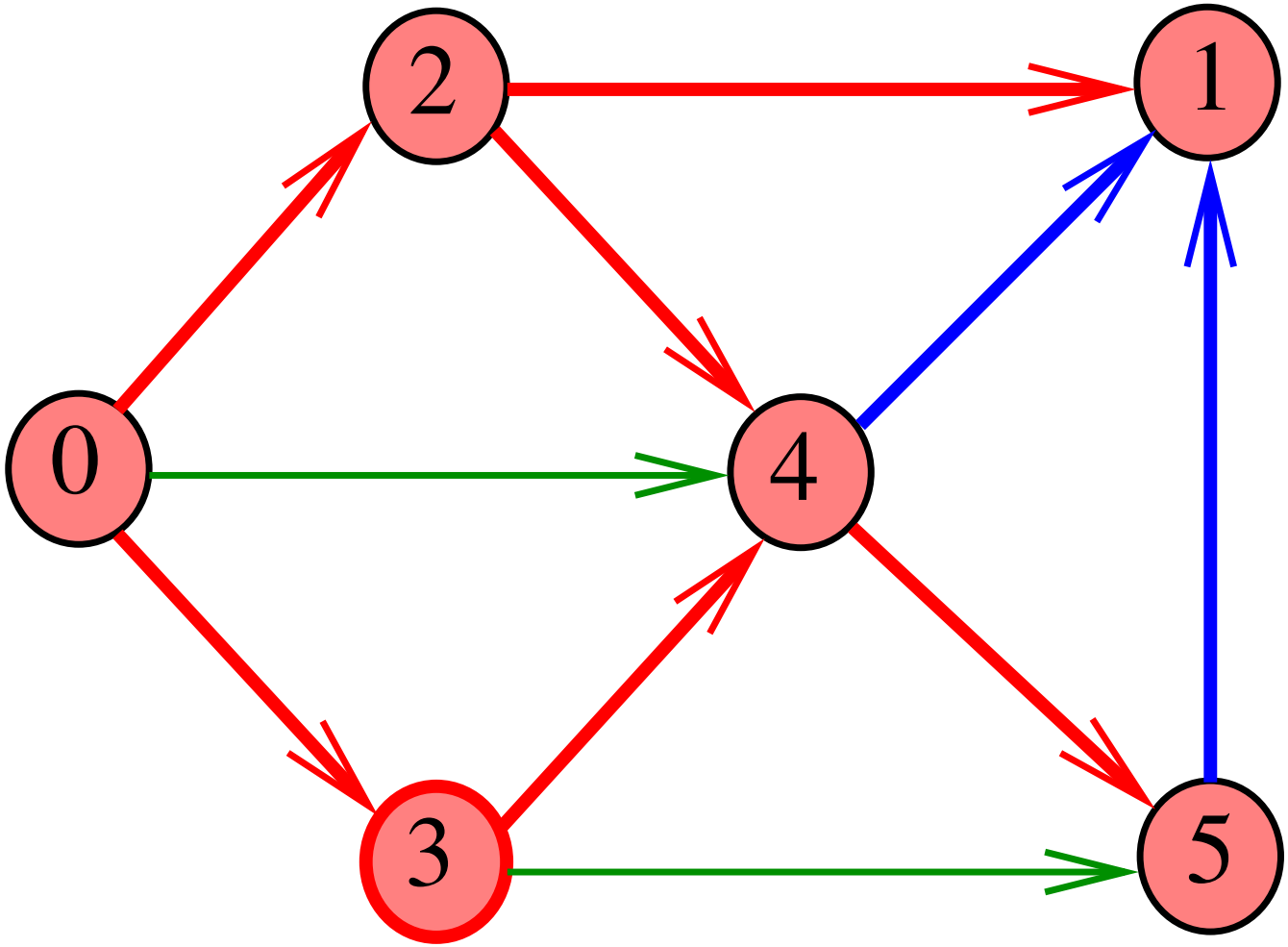
dfs(G, 0)



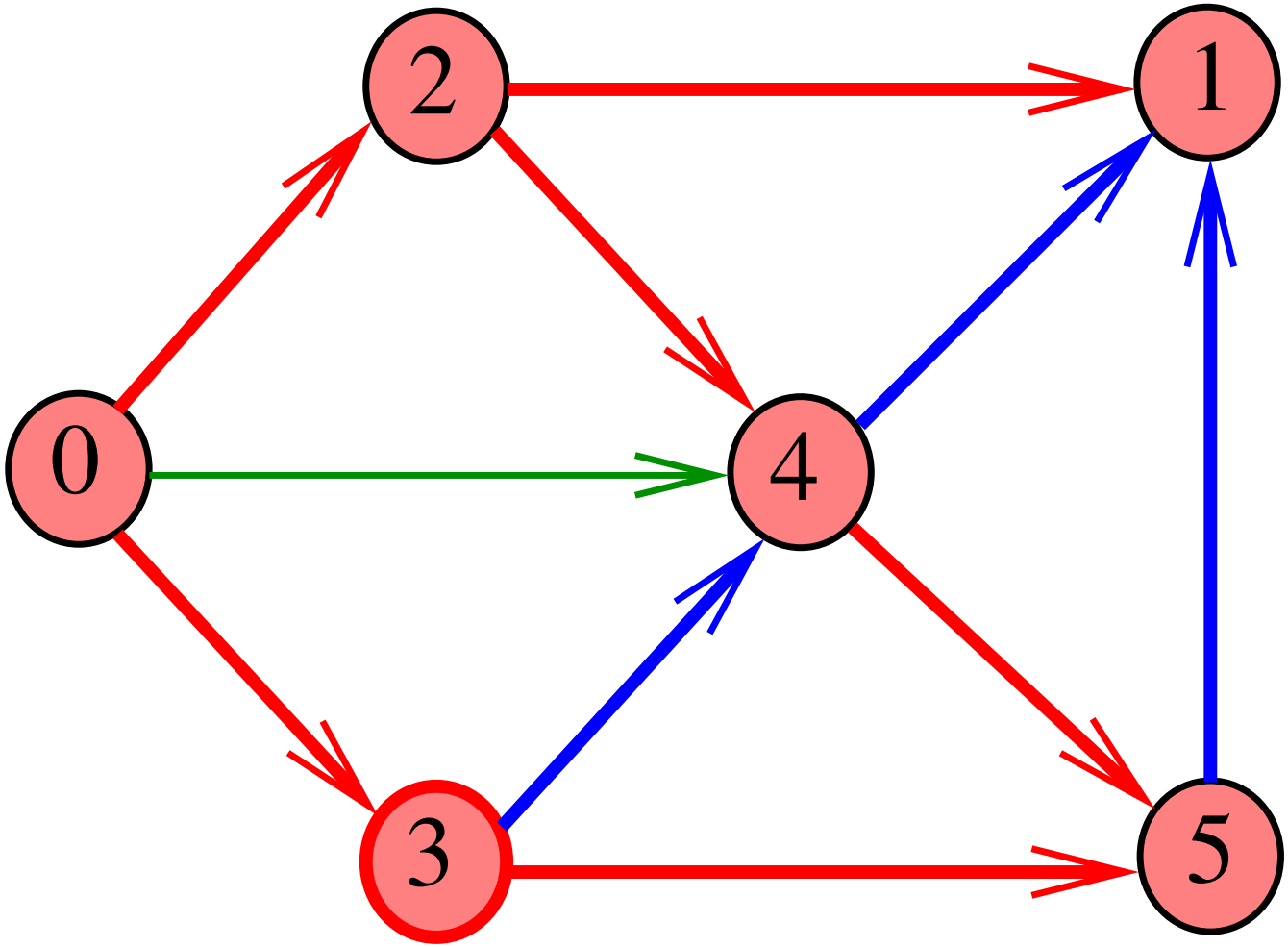
dfs(G, 3)



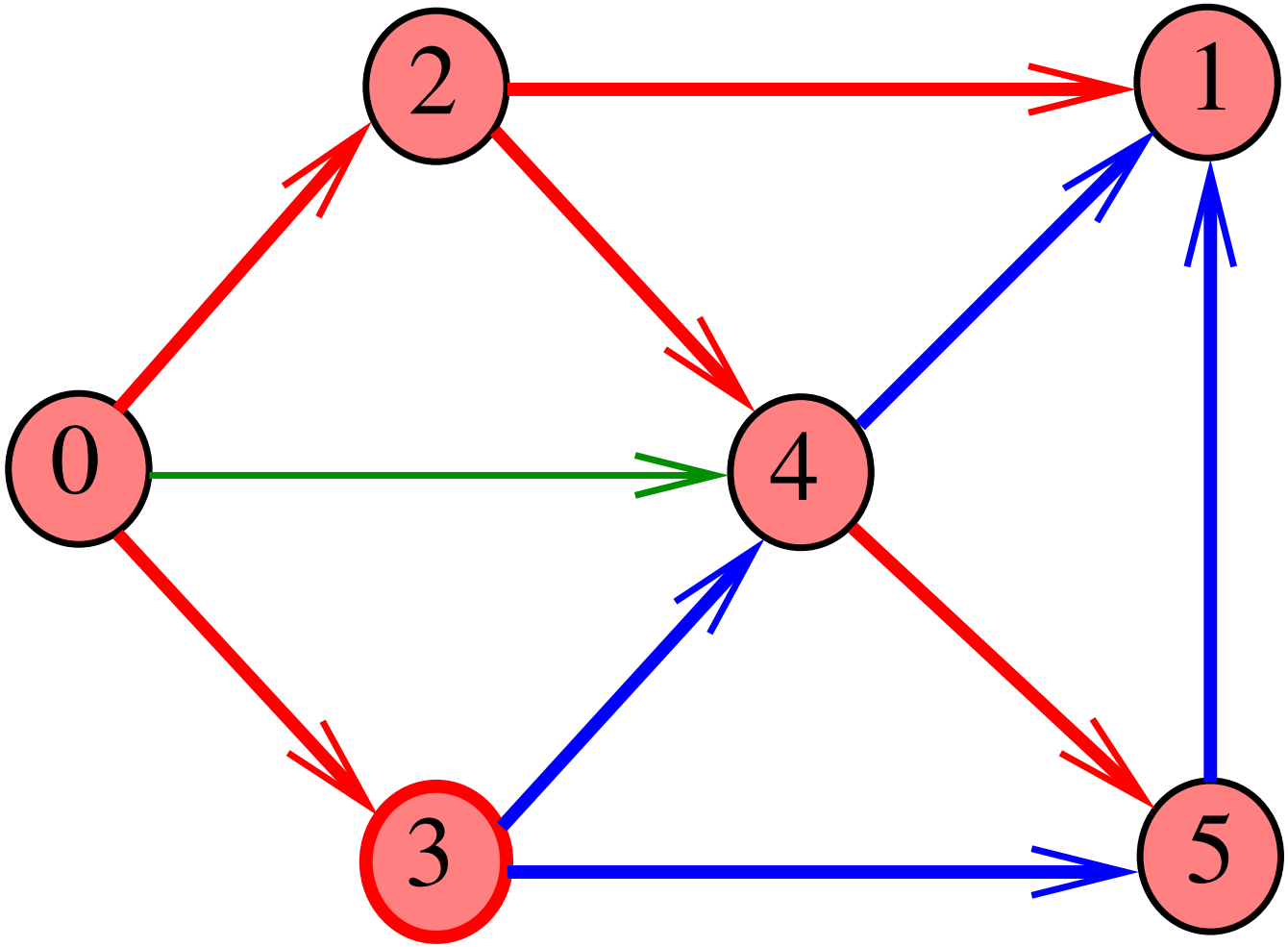
dfs(G, 3)



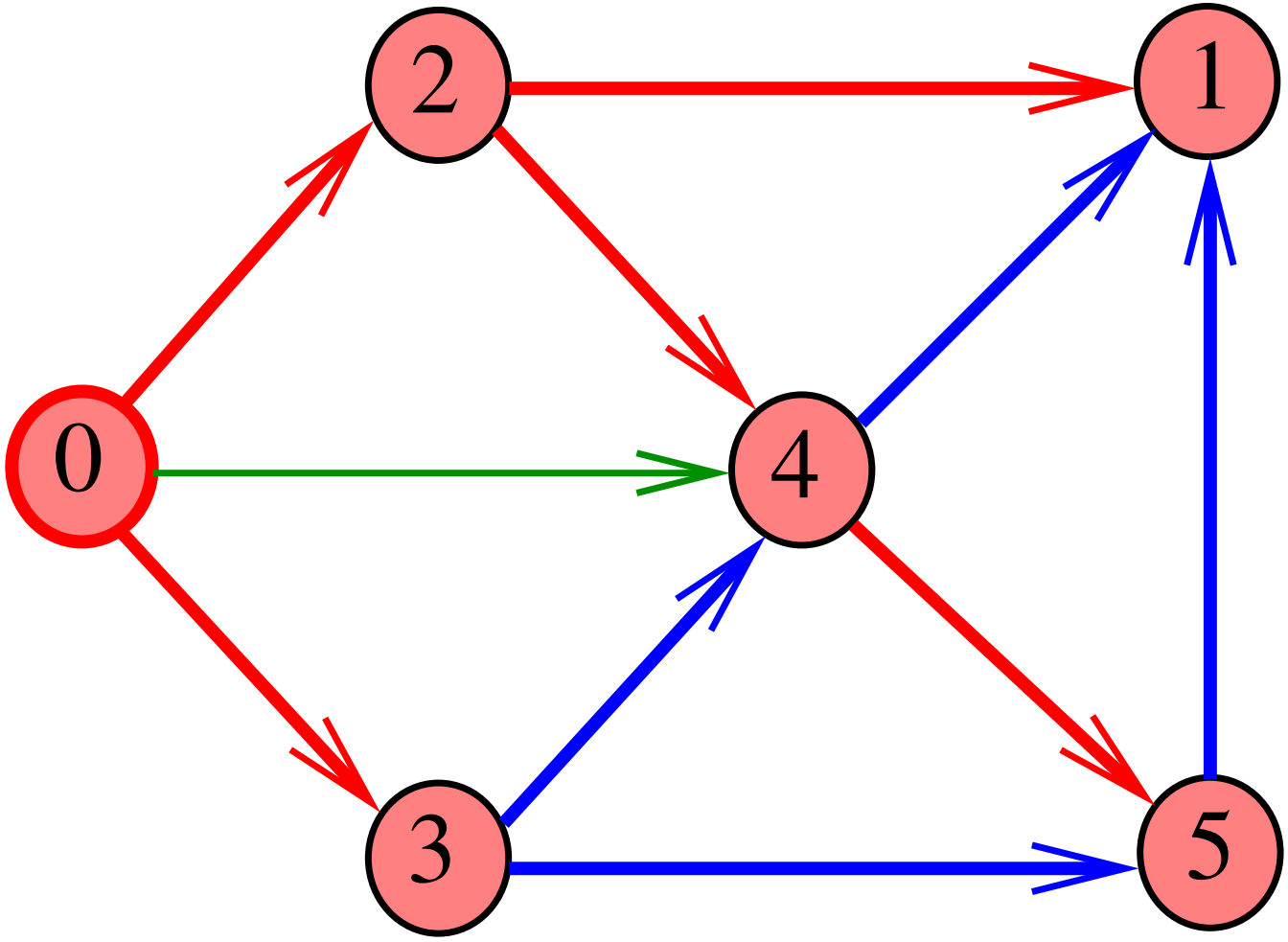
dfs(G, 3)



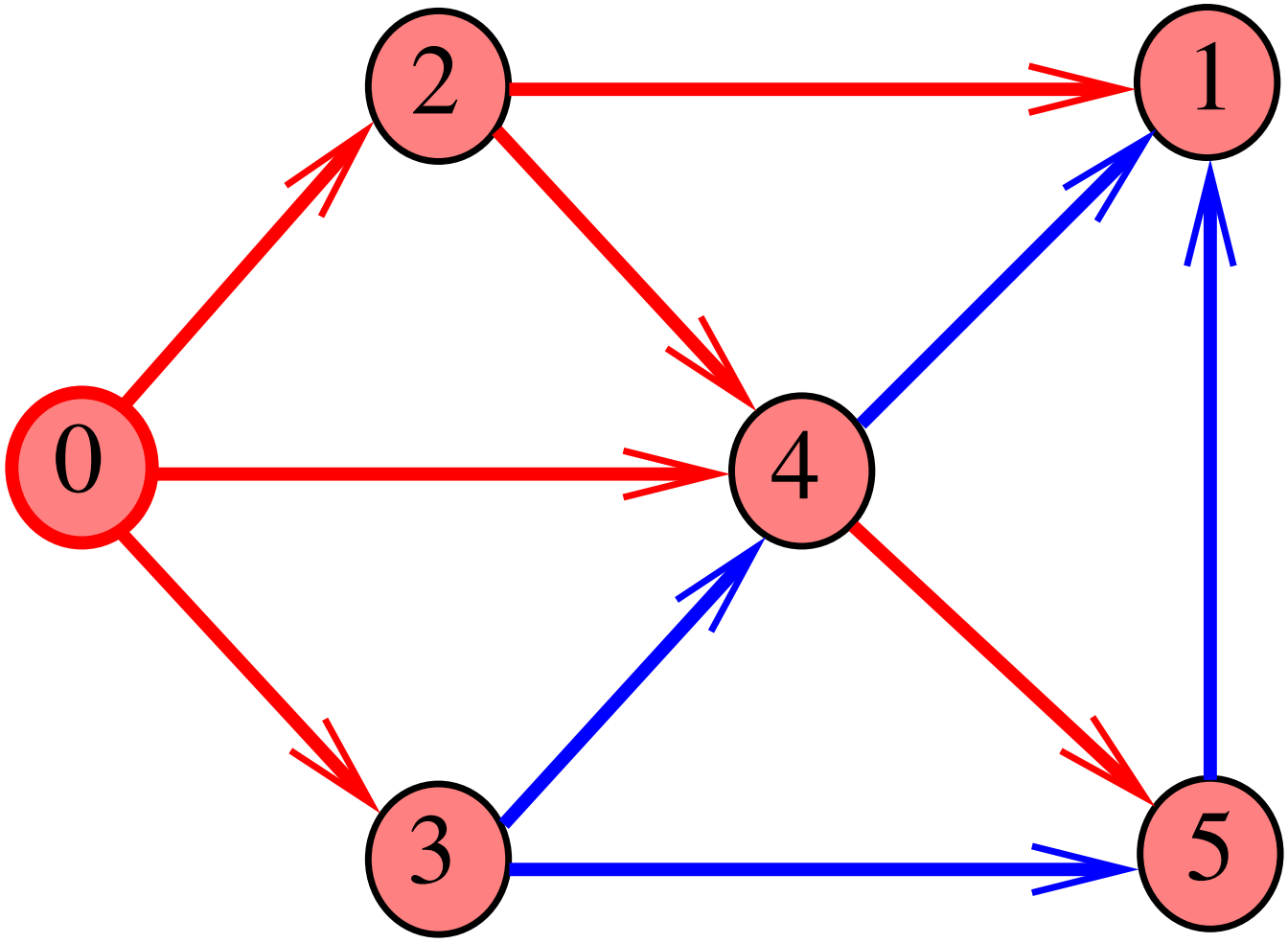
dfs(G, 3)



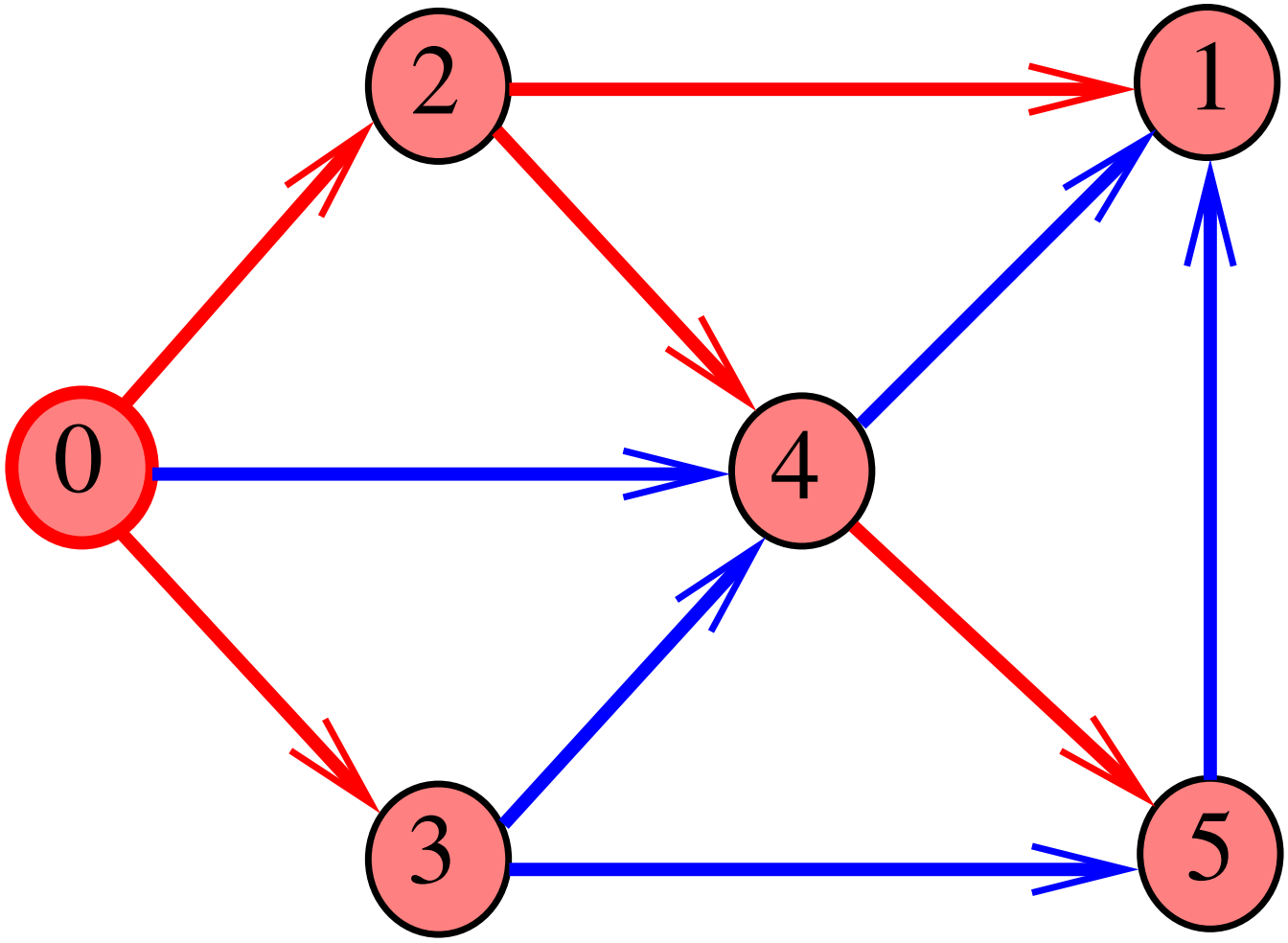
dfs(G, 0)



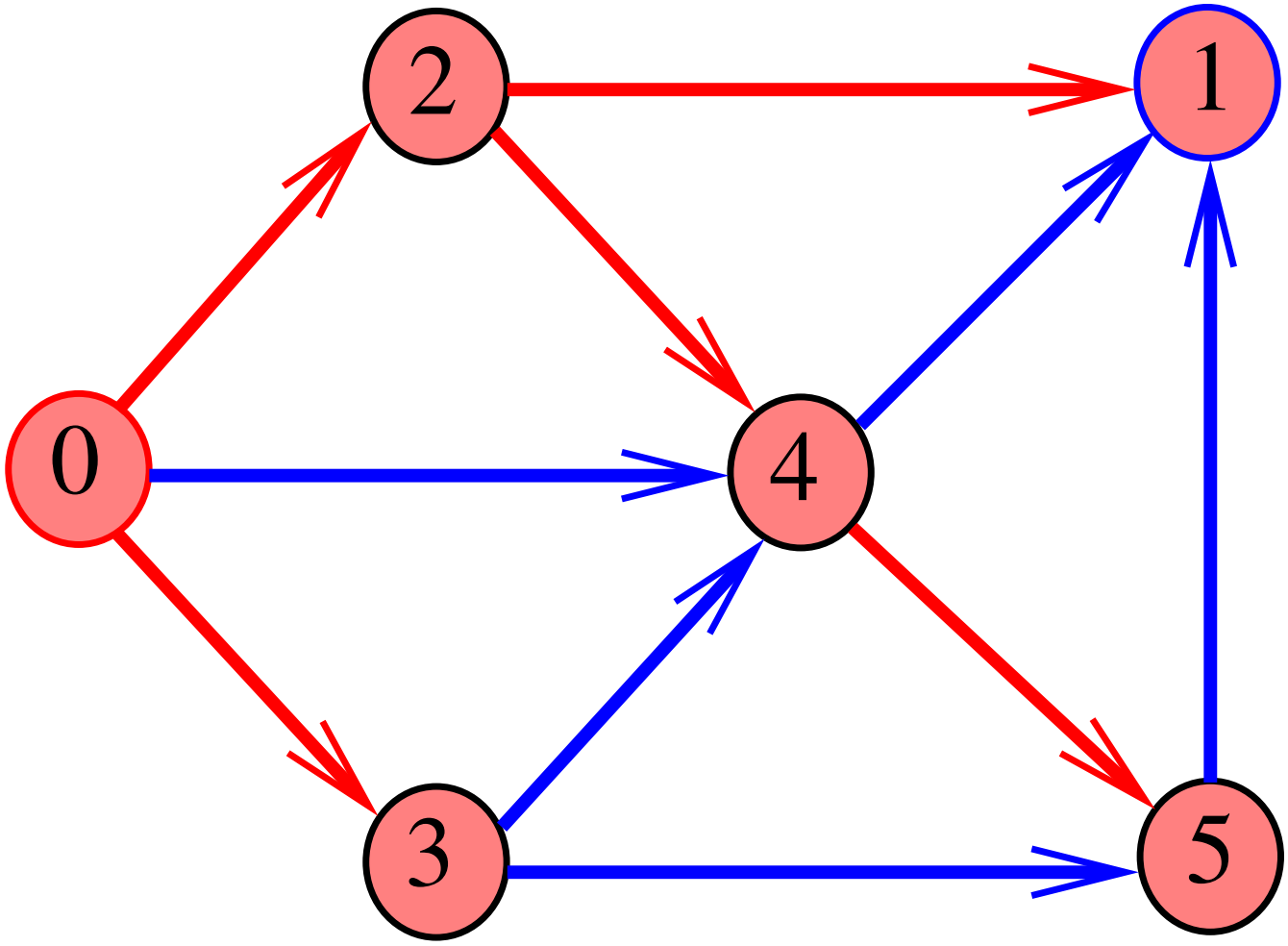
dfs(G, 0)



dfs(G, 0)



dfs(G, 0)



Consumo de tempo

Cada vértice é descoberto uma única vez, pois é branco
e, ao ser descoberto, passa a ser cinzento, e depois preto.

Consumo de tempo

Cada vértice é descoberto uma única vez, pois é branco
e, ao ser descoberto, passa a ser cinzento, e depois preto.

A lista de adjacência de cada vértice descoberto
é percorrida **uma única vez**.

Consumo de tempo

Cada vértice é descoberto uma única vez, pois é branco
e, ao ser descoberto, passa a ser cinzento, e depois preto.

A lista de adjacência de cada vértice descoberto
é percorrida uma única vez.

Logo, o consumo de tempo é $O(n+m)$,
onde $n = |V|$ e $m = |E|$,
pois a inicialização custa $\Theta(n)$ e
a soma do tamanho das listas de adjacências percorridas é $O(m)$.

Consumo de tempo

Cada vértice é descoberto uma única vez, pois é branco
e, ao ser descoberto, passa a ser cinzento, e depois preto.

A lista de adjacência de cada vértice descoberto
é percorrida uma única vez.

Logo, o consumo de tempo é $O(n+m)$,
onde $n = |V|$ e $m = |E|$,
pois a inicialização custa $\Theta(n)$ e
a soma do tamanho das listas de adjacências percorridas é $O(m)$.

O consumo de tempo de uma DFS é
linear no tamanho do grafo.

Propriedades da DFS de um grafo

Mesma estrutura que sequência válida de parênteses.

Propriedades da DFS de um grafo

Mesma estrutura que sequência válida de parênteses.

Teorema: *Para vértices u e v com $u.d < v.d$, exatamente uma das duas condições abaixo valem na floresta resultante:*

Propriedades da DFS de um grafo

Mesma estrutura que sequência válida de parênteses.

Teorema: *Para vértices u e v com $u.d < v.d$, exatamente uma das duas condições abaixo valem na floresta resultante:*

a) *Os intervalos $[u.d, u.f]$ e $[v.d, v.f]$ são disjuntos, e nem u é descendente de v, nem v é descendente de u.*

Propriedades da DFS de um grafo

Mesma estrutura que sequência válida de parênteses.

Teorema: *Para vértices u e v com $u.d < v.d$, exatamente uma das duas condições abaixo valem na floresta resultante:*

a) *Os intervalos $[u.d, u.f]$ e $[v.d, v.f]$ são disjuntos, e nem u é descendente de v, nem v é descendente de u.*

b) *O intervalo $[v.d, v.f]$ está contido no intervalo $[u.d, u.f]$, e v é descendente de u.*

Propriedades da DFS de um grafo

Mesma estrutura que sequência válida de parênteses.

Teorema: *Para vértices u e v com $u.d < v.d$, exatamente uma das duas condições abaixo valem na floresta resultante:*

a) *Os intervalos $[u.d, u.f]$ e $[v.d, v.f]$ são disjuntos, e nem u é descendente de v, nem v é descendente de u.*

b) *O intervalo $[v.d, v.f]$ está contido no intervalo $[u.d, u.f]$, e v é descendente de u.*

Corolário: *O vértice v é um descendente próprio do vértice u na floresta resultante da DFS sse $u.d < v.d < v.f < u.f$.*

Propriedades da DFS de um grafo

Mesma estrutura que sequência válida de parênteses.

Teorema: *Para vértices u e v com $u.d < v.d$, exatamente uma das duas condições abaixo valem na floresta resultante:*

a) *Os intervalos $[u.d, u.f]$ e $[v.d, v.f]$ são disjuntos, e nem u é descendente de v, nem v é descendente de u.*

b) *O intervalo $[v.d, v.f]$ está contido no intervalo $[u.d, u.f]$, e v é descendente de u.*

Corolário: *O vértice v é um descendente próprio do vértice u na floresta resultante da DFS sse $u.d < v.d < v.f < u.f$.*

Teorema (do caminho branco): *v é descendente de u sse, no momento $u.d$ em que u é descoberto, v pode ser alcançado de u por um caminho com apenas vértices brancos.*

Classificação das arestas

Quatro tipos de arestas derivadas de uma DFS:

Classificação das arestas

Quatro tipos de arestas derivadas de uma DFS:

a) Arestas da árvore: **arestas da floresta DF.**

Classificação das arestas

Quatro tipos de arestas derivadas de uma DFS:

a) Arestas da árvore: arestas da floresta DF.

b) Arestas de retorno: **de um vértice para um ascendente na floresta DF.**

Classificação das arestas

Quatro tipos de arestas derivadas de uma DFS:

a) Arestas da árvore: arestas da floresta DF.

b) Arestas de retorno: de um vértice para um ascendente na floresta DF.

c) Arestas de avanço: **de um vértice para um descendente na floresta DF.**

Classificação das arestas

Quatro tipos de arestas derivadas de uma DFS:

a) Arestas da árvore: arestas da floresta DF.

b) Arestas de retorno: de um vértice para um ascendente na floresta DF.

c) Arestas de avanço: de um vértice para um descendente na floresta DF.

d) Arestas cruzadas: **todas as outras arestas.**

Classificação das arestas

Quatro tipos de arestas derivadas de uma DFS:

a) Arestas da árvore: arestas da floresta DF.

b) Arestas de retorno: de um vértice para um ascendente na floresta DF.

c) Arestas de avanço: de um vértice para um descendente na floresta DF.

d) Arestas cruzadas: todas as outras arestas.

Teorema: *Se o grafo não é dirigido, então só há arestas da árvore e arestas de retorno.*

Classificação das arestas

Quatro tipos de arestas derivadas de uma DFS:

a) Arestas da árvore: arestas da floresta DF.

b) Arestas de retorno: de um vértice para um ascendente na floresta DF.

c) Arestas de avanço: de um vértice para um descendente na floresta DF.

d) Arestas cruzadas: todas as outras arestas.

Teorema: *Se o grafo não é dirigido, então só há arestas da árvore e arestas de retorno.*

Aplicação: ordenação topológica.