Grafos

Carlos Alberto Ynoguti

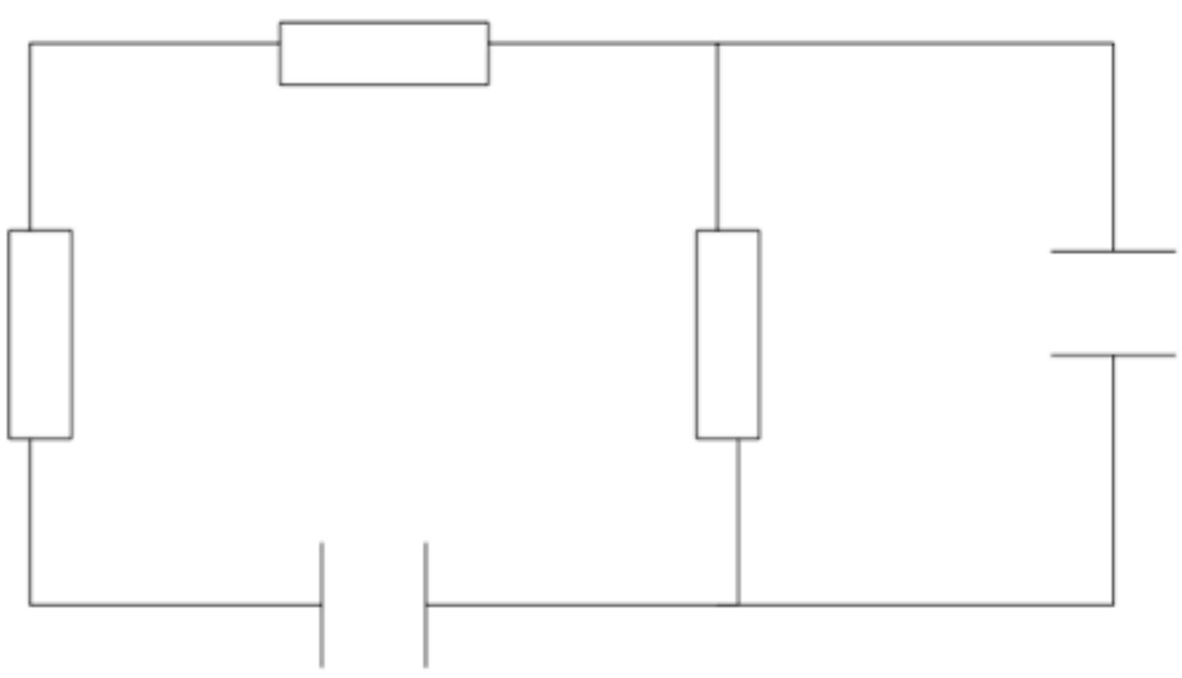
Objetivos desta aula

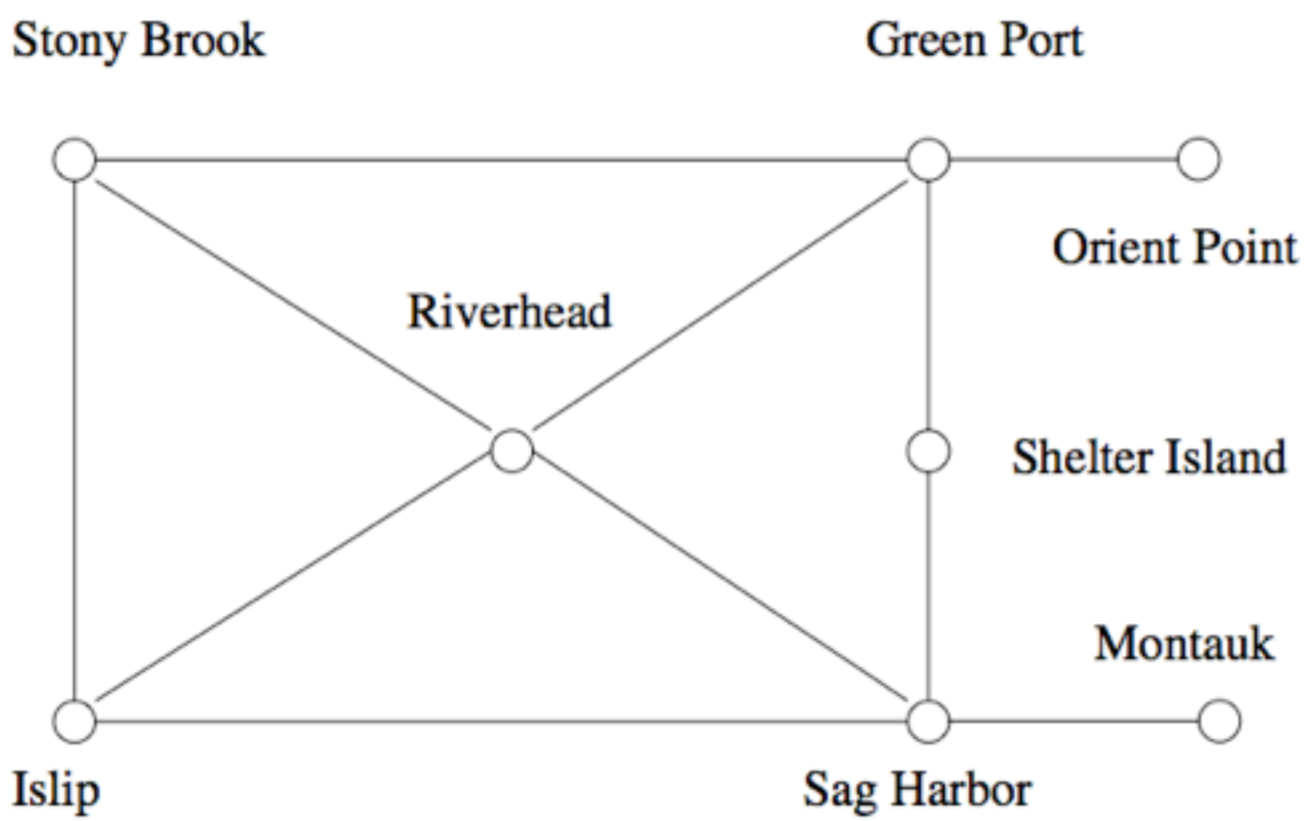
- Introduzir a teoria de grafos
  - Para que servem
  - Como são representados

O que são grafos?

- Um grafo G=(V,E) consiste de um conjunto de vértices V e um conjunto de arestas E. Cada aresta liga um par de vértices.

- Exemplos

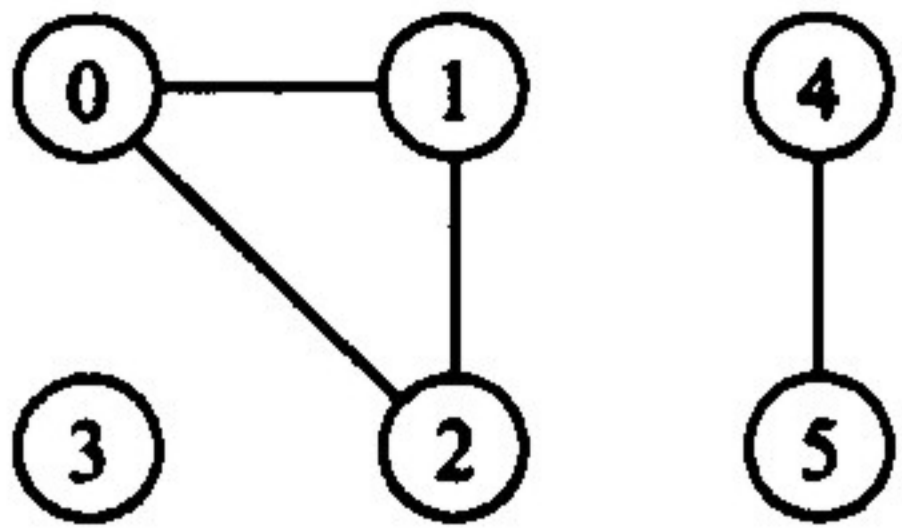




Exemplo

V = {0, 1, 2, 3, 4, 5}

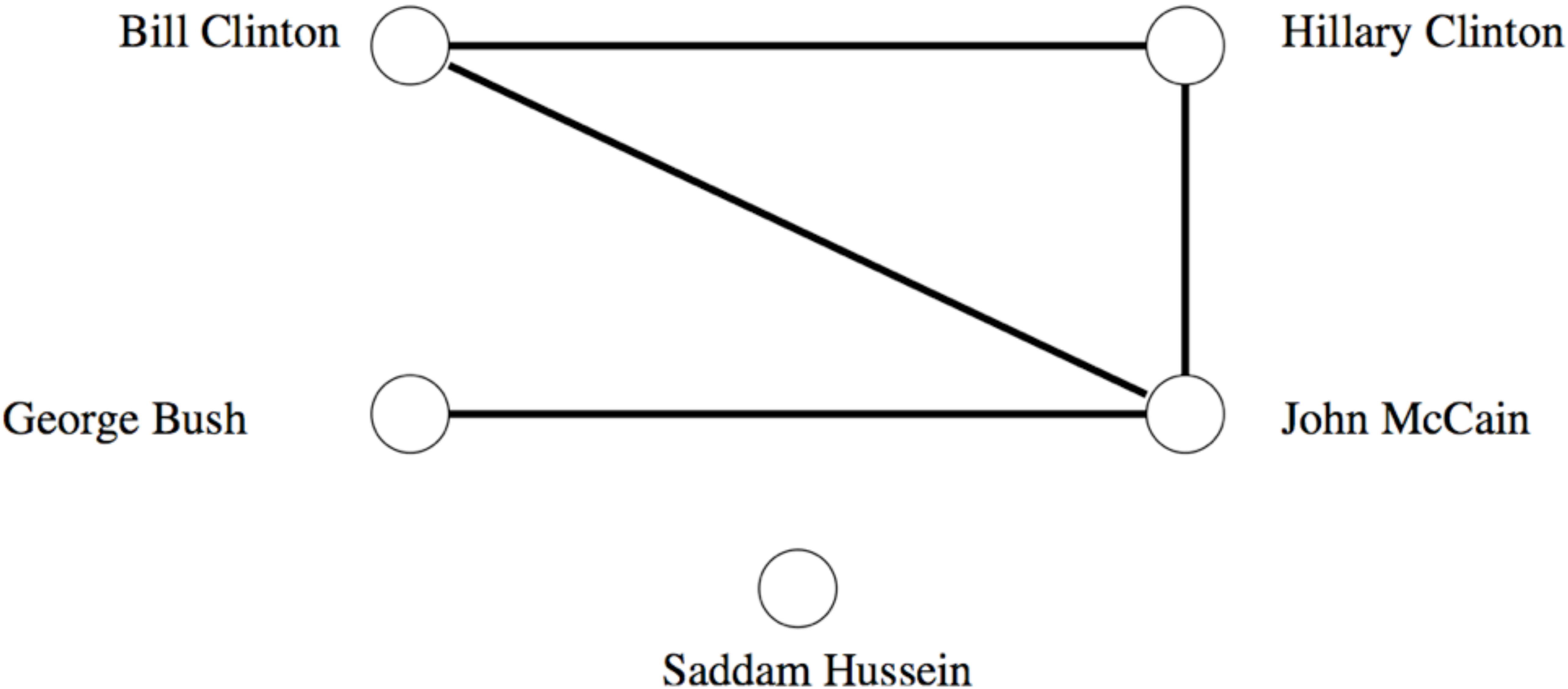
E = {(0,1), (1,2), (0,2), (4,5)} ou
E = {a1, a2, a3, a4}



Para que servem os grafos?

- Podem representar essencialmente todo tipo de relação entre coisas
- Exemplos:
  - Redes de estradas: vértices são as cidades e as estradas são as arestas.
  - Circuitos elétricos: vértices são as junções e os componentes são as arestas

Exemplo: redes sociais



Dicas e truques

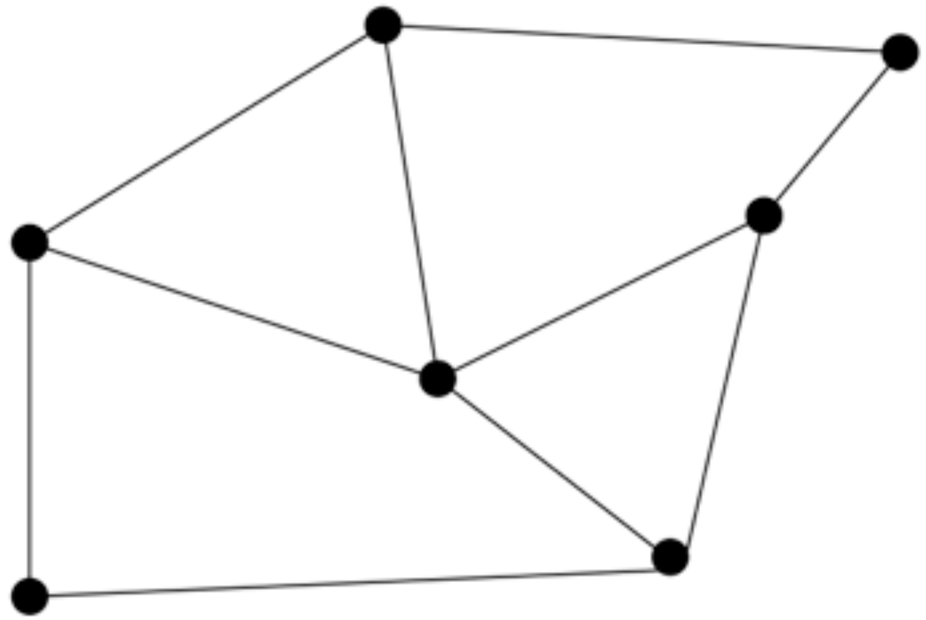
- Muitos problemas complicados podem ser facilmente resolvidos se forem modelados como grafos.
- Projetar algoritmos para lidar com grafos é bastante difícil: o melhor a fazer neste caso é modelar corretamente o problema e tirar vantagem dos algoritmos existentes.
- Conclusão: familiarizar-se com os tipos de problemas envolvendo grafos é mais importante que entender os detalhes dos algoritmos.

Orientado vs Não orientado

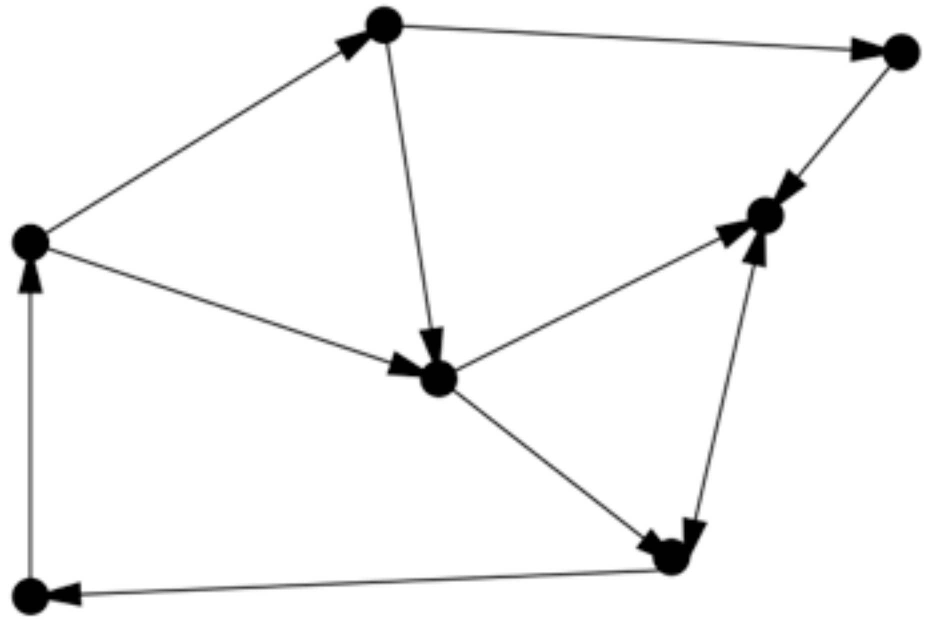
- **Não orientado:** se a aresta $(x,y) \in E \Rightarrow (y,x) \in E$

  Exemplo: estradas entre cidades

- **Orientado:** ruas dentro de uma cidade



não orientado



orientado

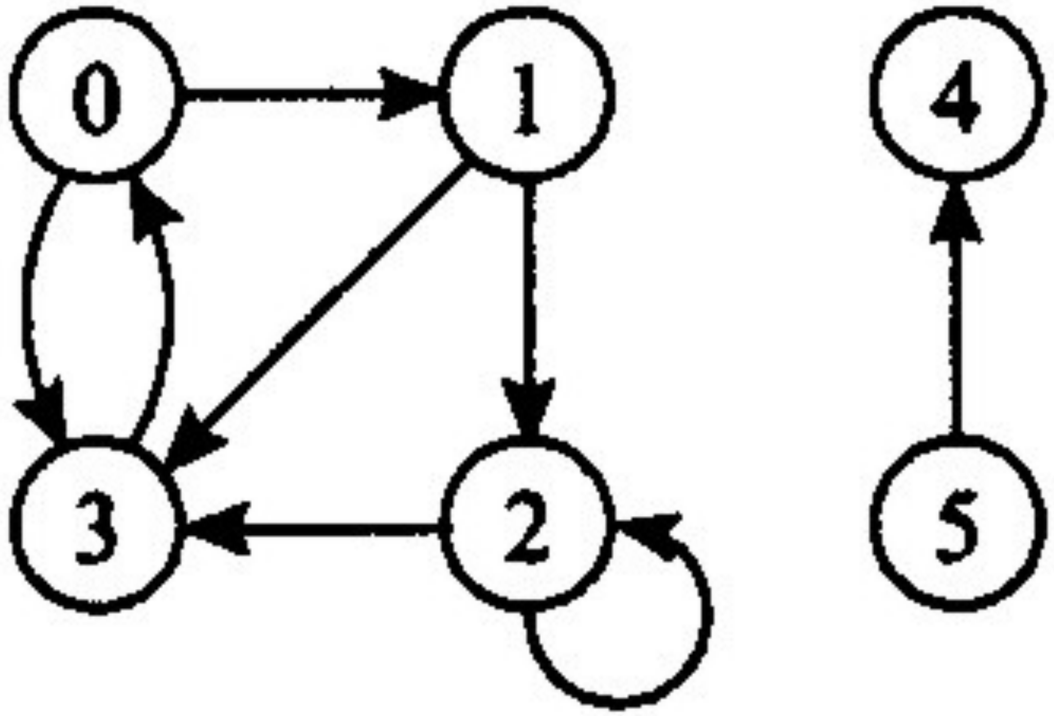
Se eu sou seu amigo, então você também é meu amigo?

- Define se um grafo é orientado ou não. Exemplos:
- Ouvir falar de: orientado, pois geralmente ouvimos falar de pessoas famosas, mas elas nunca ouviram falar de nós.
- Fazer sexo com: não orientado, pois a operação crítica sempre requer um parceiro.
- Ser amigo de: gostaria de acreditar que é um grafo não orientado.

Função aresta-vértice

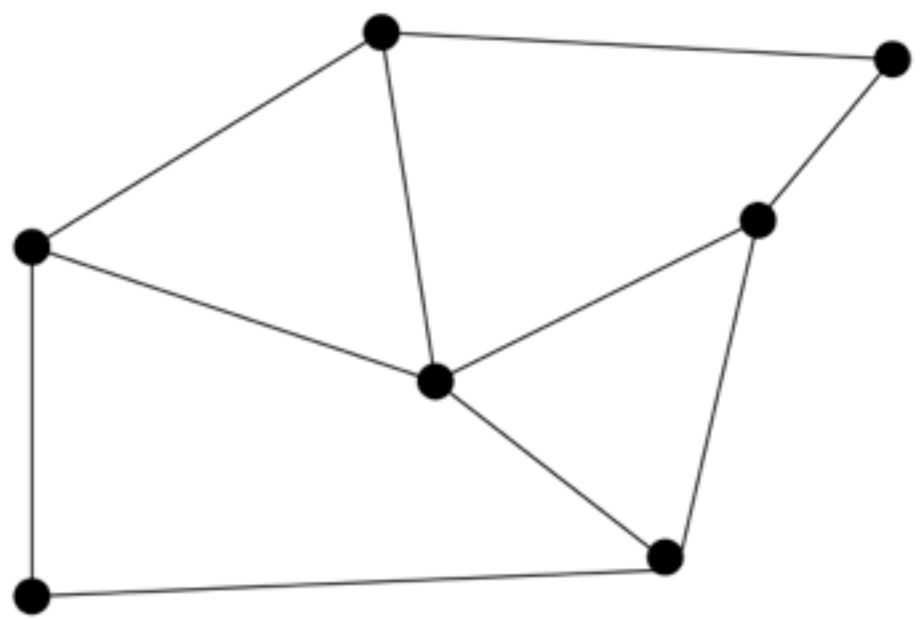
Associa aresta a vértices.

aresta/vértices
{0,1}
{0,3}
{1,2}
{2,2}
{2,3}
{1,3}
{3,0}
{5,4}

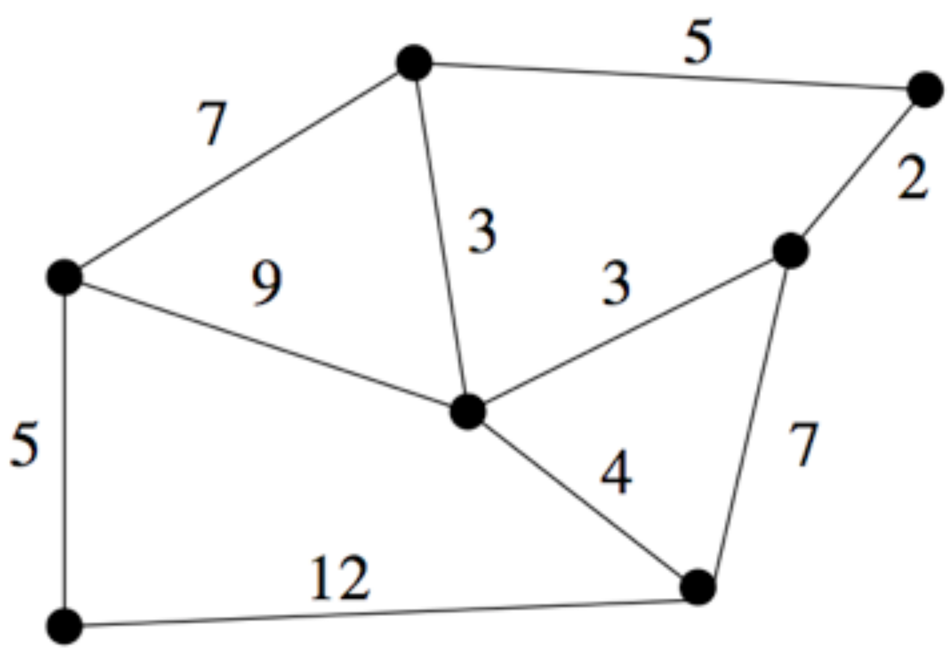


Ponderado vs Não ponderado

- Ponderado: arestas associadas a um valor numérico.
- Exemplo: distância entre cidades.
- Não ponderado: custo unitário para cada aresta.
- Exemplo: número de linhas executadas em um código.



não ponderado



ponderado

Ponderado vs Não ponderado

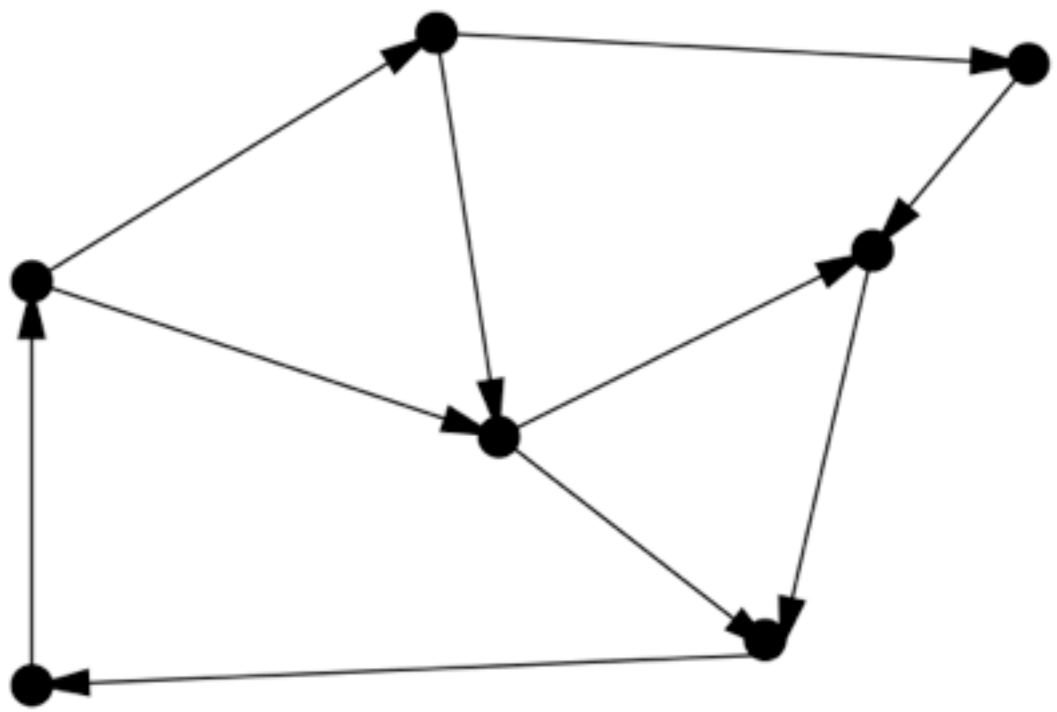
- Diferença: cálculo do menor caminho entre dois vértices
- Não ponderado: o menor caminho é aquele que possui o menor número de vértices ⇒ busca em largura.
- Ponderado: menor caminho depende dos pesos de cada aresta, e algoritmos mais sofisticados são necessários.

Você é um amigo próximo ou distante?

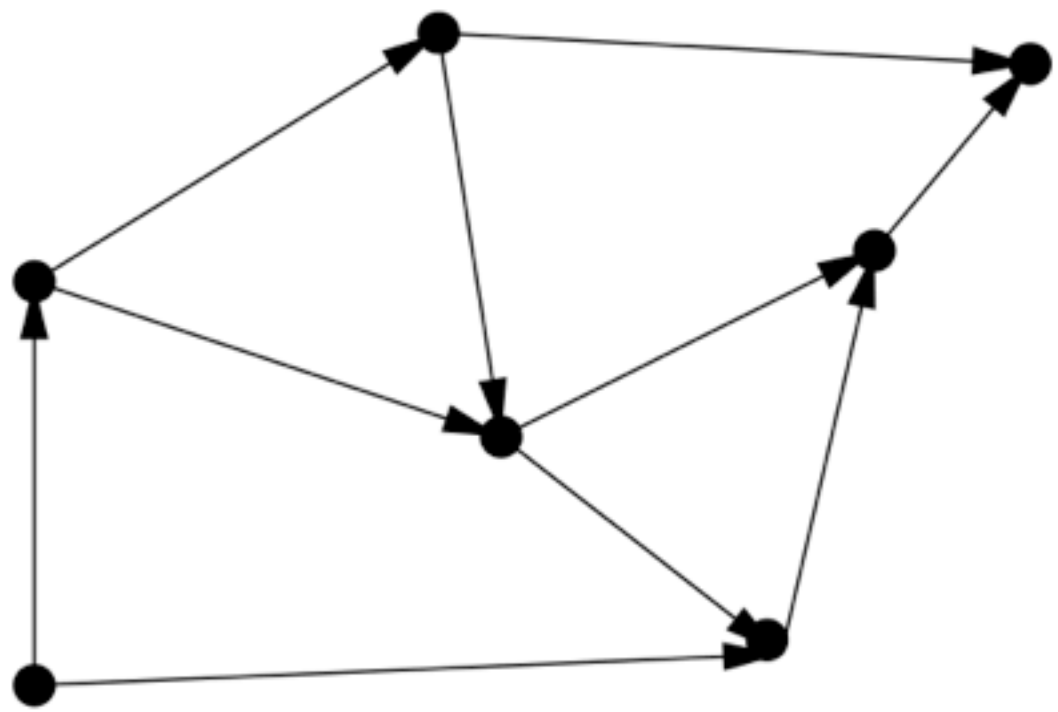
- **Gráficos ponderados:** cada aresta tem um valor numérico associado.
- **Força da amizade:** valores entre +10 (muito amigo) e -10 (inimigo mortal), por exemplo.
- **Redes de estradas:** pesos podem ser a distância em km, tempo de percurso, velocidade máxima, etc.
- **Grafos não ponderados:** arestas têm o mesmo peso.

Cíclico vs Acíclico

- Acíclico: não contém ciclos.
    - Exemplos: árvores e DAG (Directed Acyclic Graphs)
- Cíclicos: contém ciclos



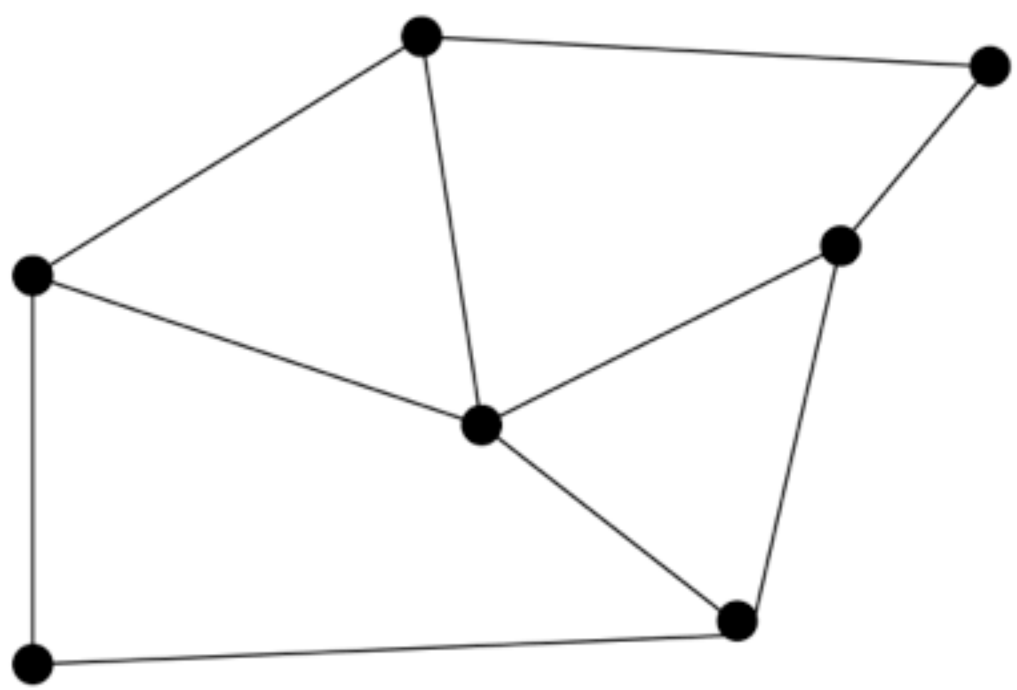
cíclico



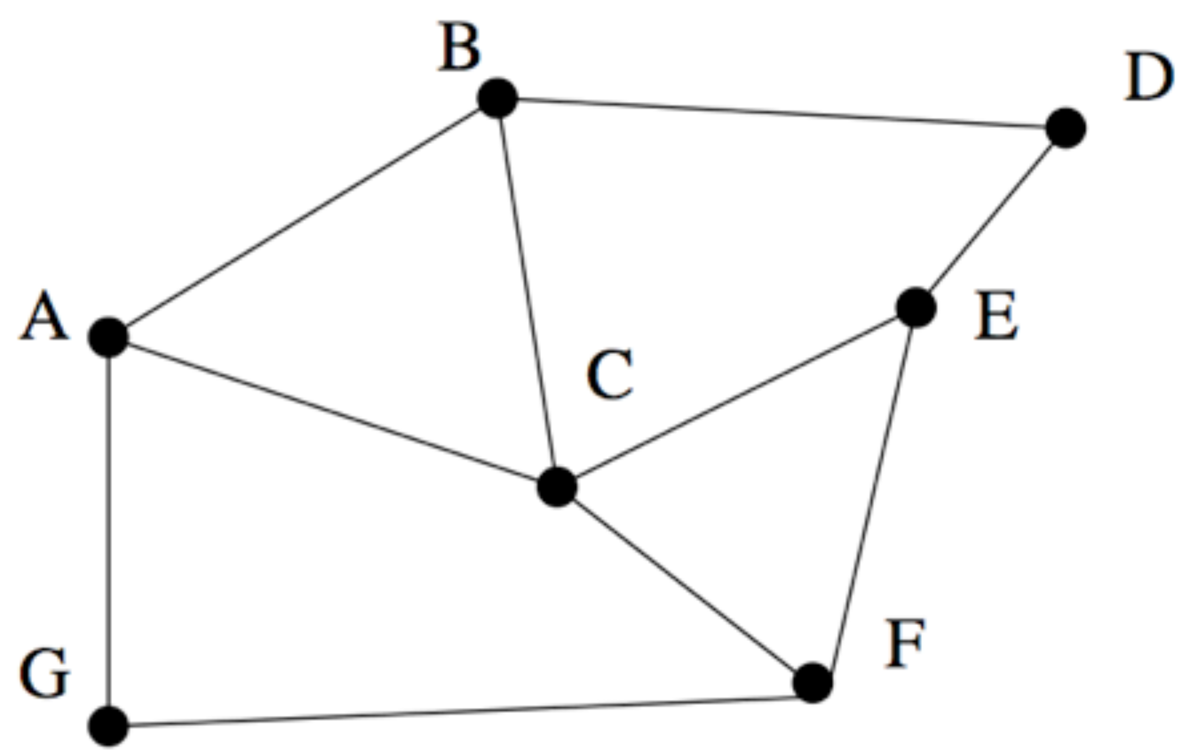
acíclico

Rotulado vs Não rotulado

- Rotulado: cada vértice tem um rótulo, que o identifica
- Não rotulado: vértices não identificados



não rotulado



rotulado

Quem tem mais amigos?

- Grau de um vértice: número de arestas associados a ele.
- Pessoas populares: grau alto
- Ermitões: grau 0
- Grafos densos: vértices de grau alto
- Grafos esparsos: vértices de grau baixo
- Grafo regular: todos os vértices têm o mesmo grau.

Conclusões

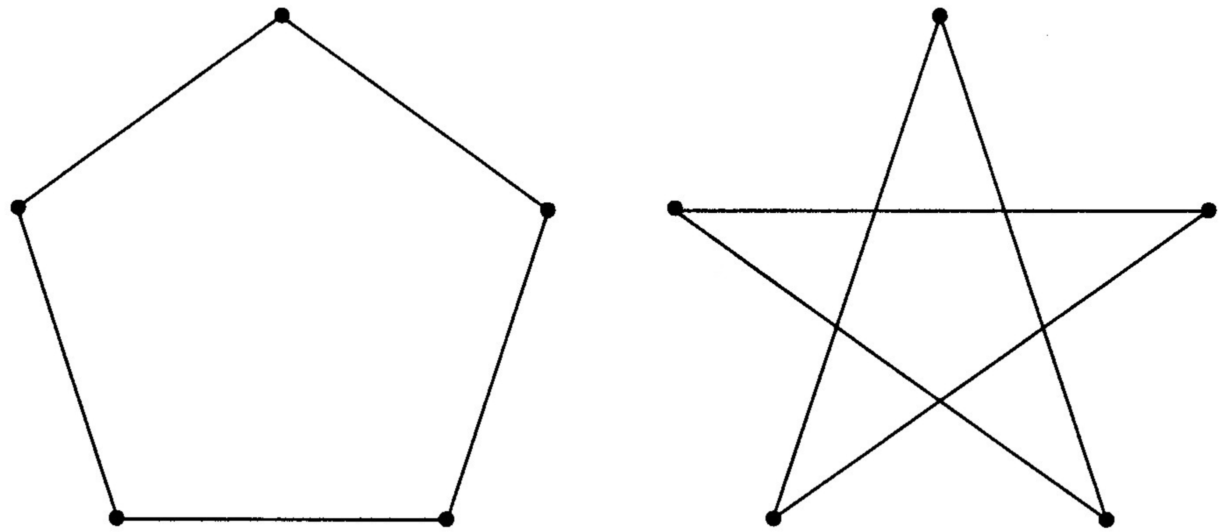
- Grafos podem ser usados para modelar uma grande variedade de estruturas e relações.
- A terminologia da teoria de grafos nos fornece uma linguagem para falarmos sobre elas.

Exercício 1

Considere os dois diagramas abaixo.

Rotule seus vértices e suas arestas de tal forma que os 2 diagramas representem o mesmo grafo.

Represente também a sua função aresta-vértice.



Exercício 2

Apresente duas possíveis representações do Grafo com as características abaixo:

Conjunto de vértices: V = {v1, v2, v3, v4}

Conjunto de arestas: *A* = {a1, a2, a3, a4}

Função aresta-vértice:

aresta	vértice
a1	{v1,v3}
a2	{v2,v4}
a3	{v2,v3}
a4	{v3,v1}

Exercício 3

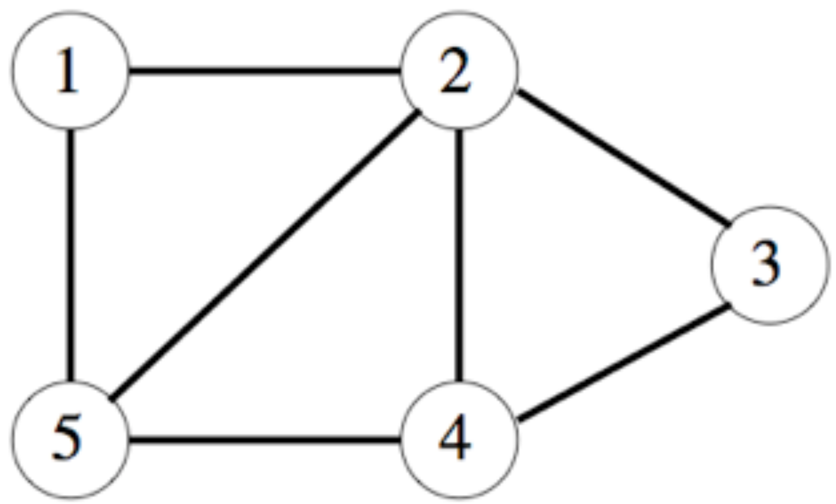
Cobra come sapo e pássaros; pássaros e aranhas comem insetos; sapos comem caracol, aranhas e insetos.

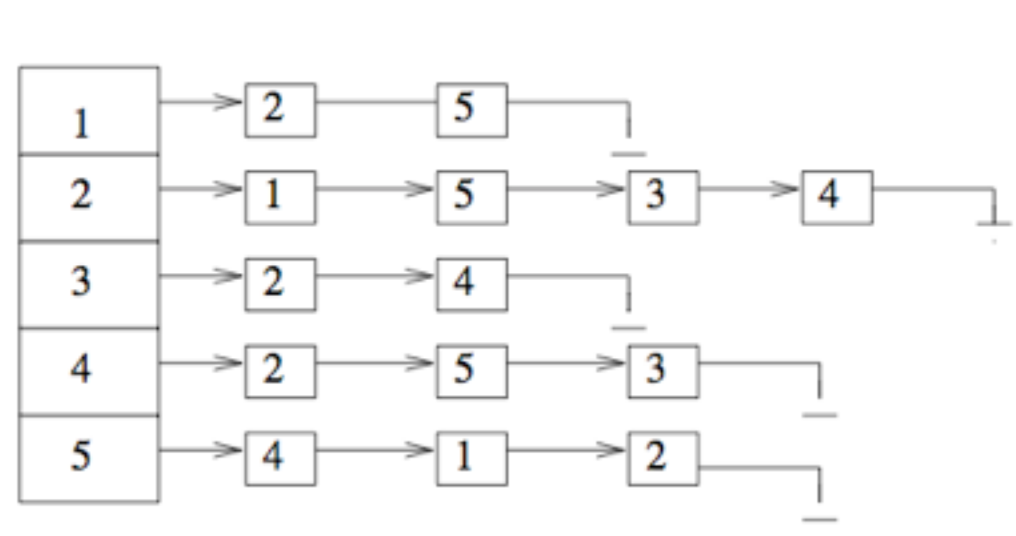
Desenhe um grafo ilustrando esse comportamento predatório.

Determine o conjunto de vértices, o conjunto de arestas e o grau de cada vértice

Estruturas de dados para os grafos

	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0





matriz de adjacência

lista de adjacência

Matriz de adjacência

- Para um grafo G=(V,E) com n vértices e m arestas, usa-se uma matriz M de dimensões $n \times n$.
- Elemento M[i,j] = 1 se (i,j) é uma aresta de G, e 0 se não for.
- Permite respostas rápidas à questão "(i, j) está em G?", e atualizações rápidas para inserção e remoção de arestas
- Pode usar muita memória para grafos com muitos vértices e poucas arestas.

Exemplo

- Manhattan: 15 avenidas que cortam 200 ruas
- Vértices: cruzamentos
- Arestas: ruas e avenidas
- Total: 3000 vértices e 6000 arestas
- Matriz de adjacências precisa de

  3000 x 3000 = 9000000 células, quase todas vazias!

Listas de adjacências

- Podemos representar grafos esparsos de maneira mais eficiente usando listas ligadas para armazenar os vizinhos adjacentes a cada vértice.
- Usam ponteiros, mas não são assustadoras uma vez que você tenha experiência com estruturas ligadas.
- Mais difícil verificar se uma aresta (i,j) está em G, pois precisamos varrer a lista correspondente para encontrar (ou não) a aresta.
- Surpreendentemente fácil projetar algoritmos que evitem estas consultas.

Matrizes de adjacências vs Listas de adjacências

Comparison	Winner
Faster to test if (x, y) is in graph?	adjacency matrices
Faster to find the degree of a vertex?	adjacency lists
Less memory on small graphs?	adjacency lists $(m+n)$ vs. (n^2)
Less memory on big graphs?	adjacency matrices (a small win)
Edge insertion or deletion?	adjacency matrices $O(1)$ vs. $O(d)$
Faster to traverse the graph?	adjacency lists $\Theta(m+n)$ vs. $\Theta(n^2)$
Better for most problems?	adjacency lists

Conclusão: use listas de adjacências!!!

Atravessando Grafos

- Objetivo: passar de forma sistemática por todas as arestas e vértices uma única vez.
- Pra que? Todas as operações básicas de manutenção de registros das informações contidas no grafo fazem uso disso.
- Exemplos: imprimir, fazer uma cópia, mudar a forma de representação, etc.

Idéia chave

- Marcar cada vértice já visitado.
- Manter registro daquilo que ainda não foi explorado.
- Cada vértice irá existir em um de 3 estados:
  - não descoberto: o vértice ainda não foi visitado
  - descoberto: o vértice foi descoberto, mas ainda não checamos as arestas que incidem sobre ele.
  - processado: vértice depois que visitamos todas as arestas que incidem sobre ele.

Na prática

- Início: somente vértice inicial marcado como descoberto.
- Para explorar completamente um vértice v, precisamos explorar todas as arestas que saem dele.
- Se uma aresta vai para um vértice desconhecido x, este vai para a lista dos vértices a serem processados.
- As arestas que dão num vértice processado são ignoradas.
- Podemos também ignorar arestas que dão em vértices descobertos pois eles já estão na lista de vértices a serem processados.

Observações

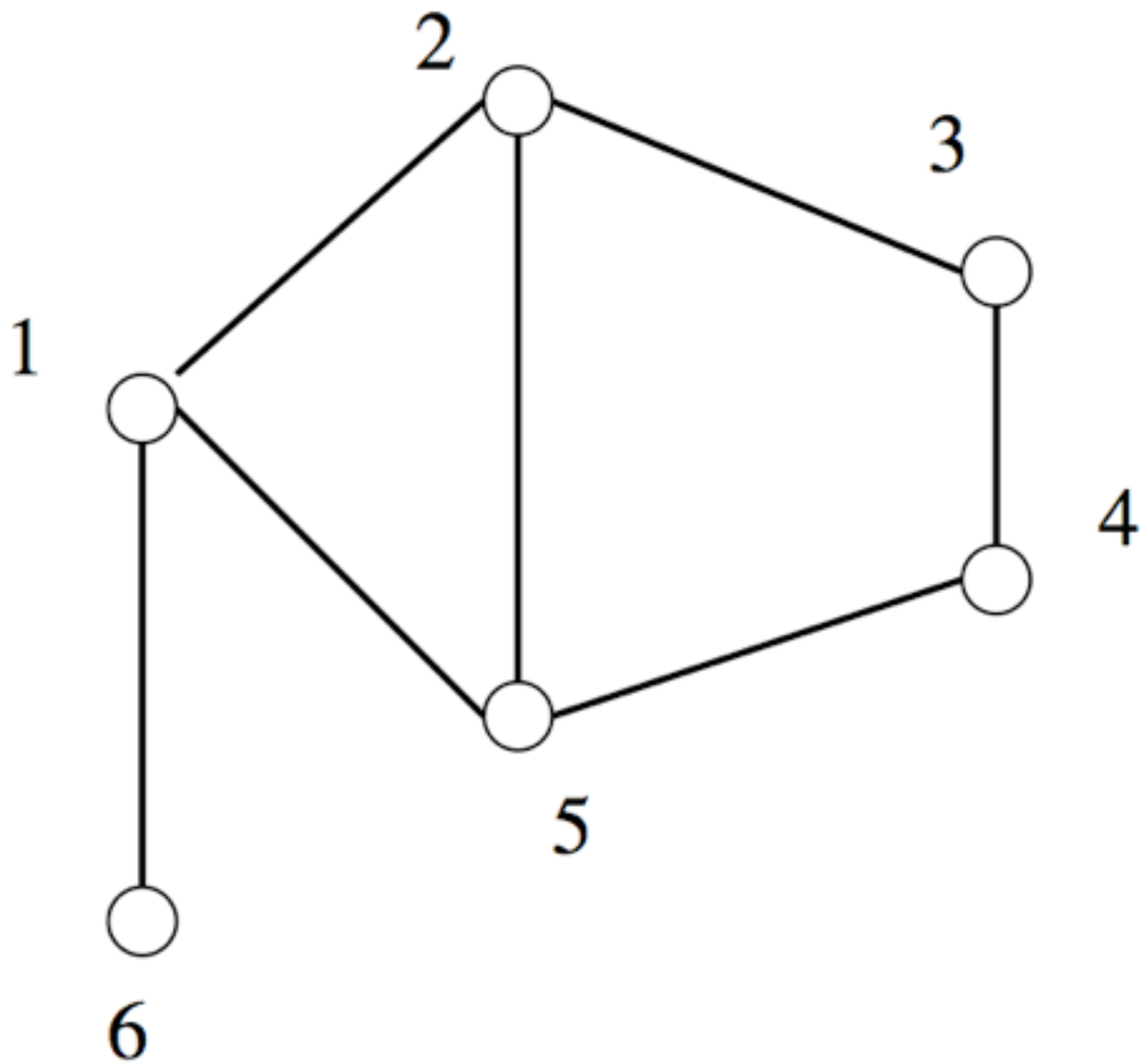
- **Grafo não orientado:** cada aresta será considerada duas vezes.

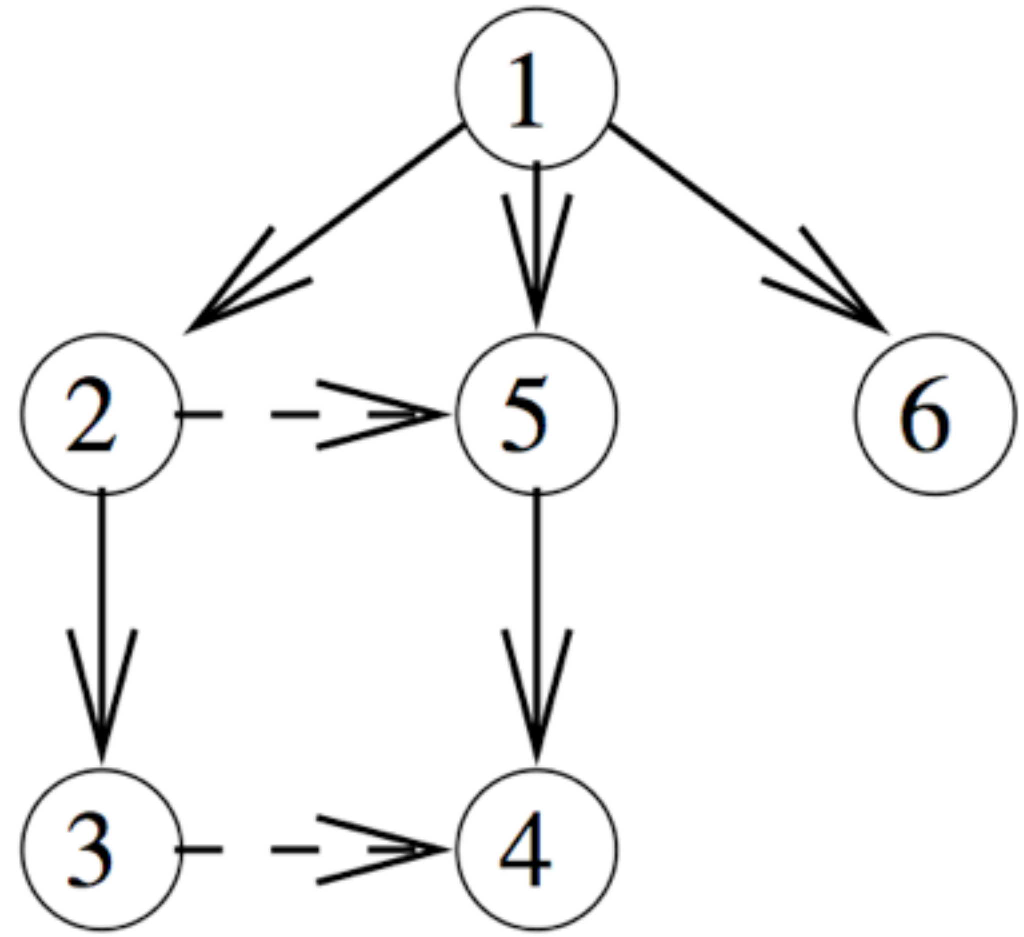
- **Grafo orientado:** cada aresta será considerada apenas uma vez, quando exploramos o vértice de origem.

Busca em largura

- Durante o percurso, cada nó do grafo muda seu estado de não descoberto para descoberto.
- Na busca em largura de grafos não orientados, atribui-se um sentido para cada aresta do vértice descoberto v para o vértice não descoberto v. Marcamos u como pai de v.
- Desde que cada nó tem exatamente um pai (exceto pela raiz), isto define uma árvore nos vértices do grafo.
- Esta árvore define o caminho mais curto da raiz para todos os outros nós da árvore.
- Muito útil em problemas de caminho mínimo

Geração da árvore



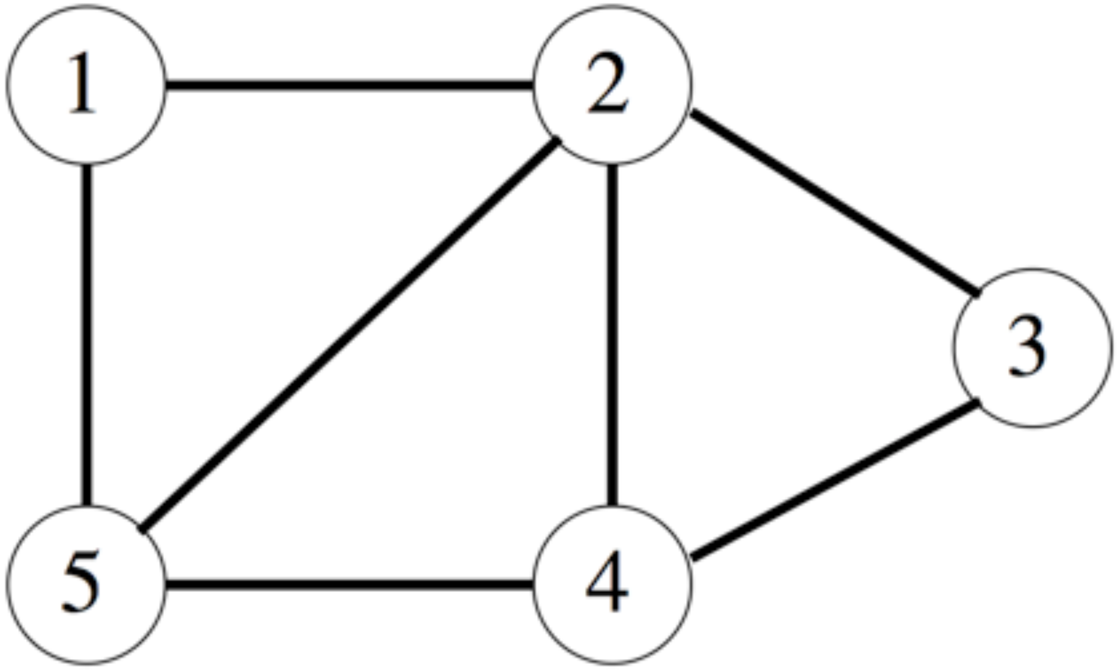


Algoritmo

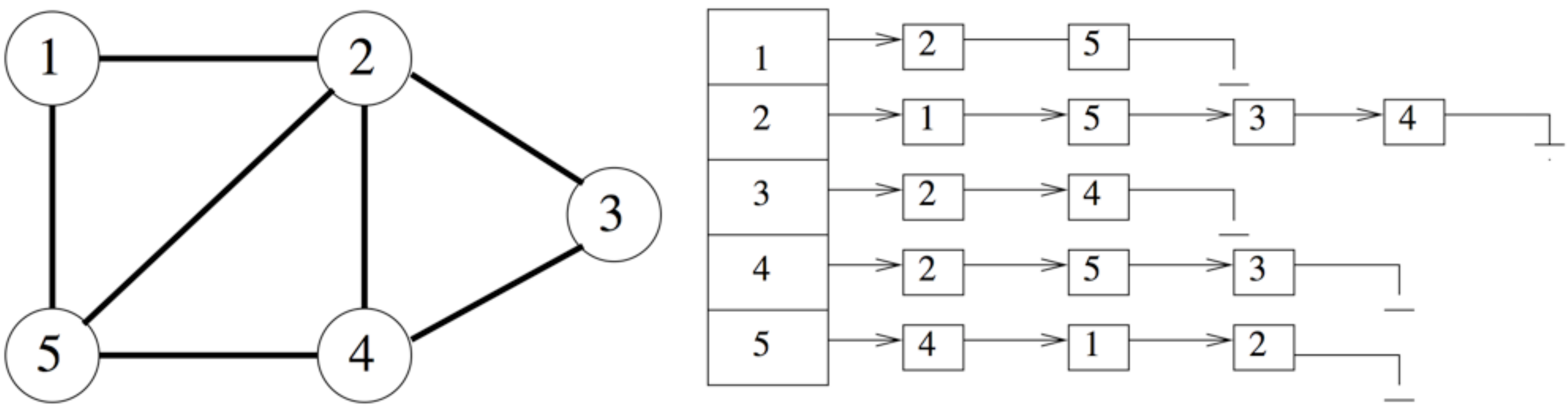
BFS(G, s)

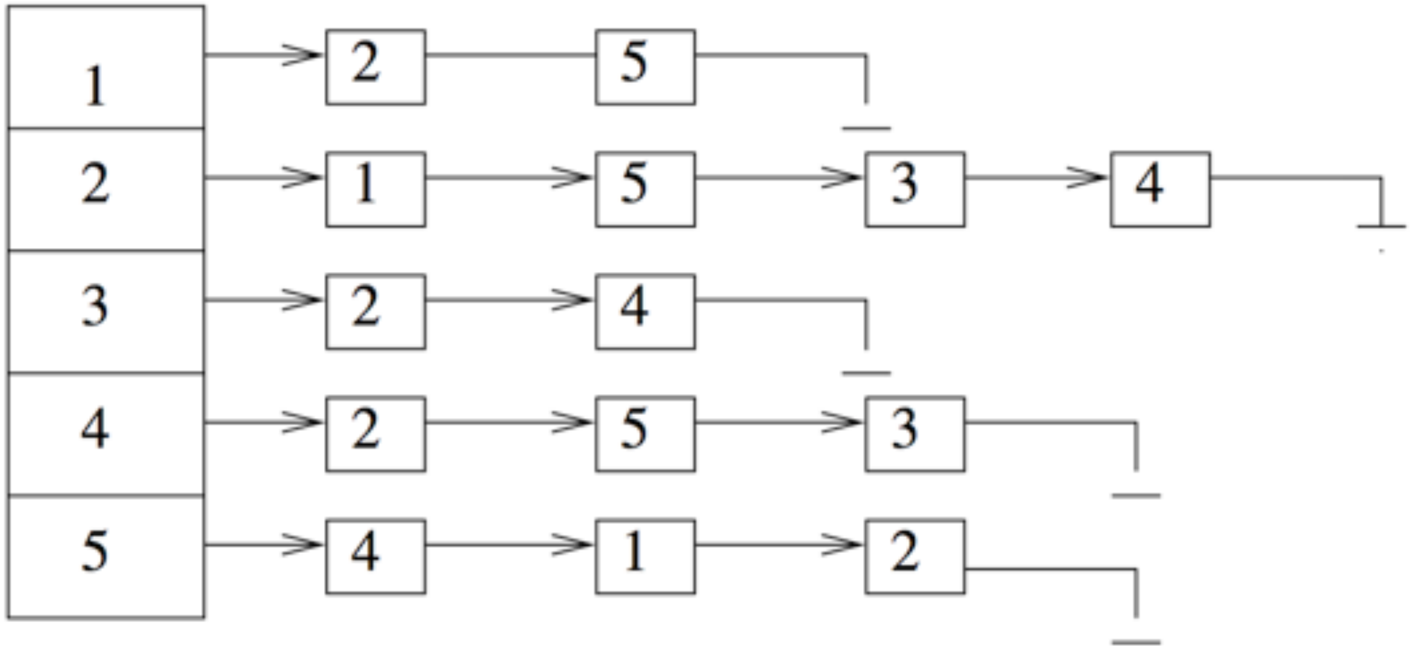
```
for each vertex u ∈ V[G] − {s} do
    state[u] = "undiscovered"
    p[u] = nil, i.e. no parent is in the BFS tree
state[s] = "discovered"
p[s] = nil
Q = {s}
while Q ≠ ∅ do
    u = dequeue[Q]
    process vertex u as desired
    for each v ∈ Adj[u] do
        process edge (u, v) as desired
        if state[v] = "undiscovered" then
            state[v] = "discovered"
            p[v] = u
            enqueue[Q, v]
    state[u] = "processed"
```

Primeiro passo: gerar lista de adjacências



Primeiro passo: gerar lista de adjacências






```
BFS(G, s)
for each vertex u ∈ V[G]−{s}
    state[u] = "undiscovered"
    p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
    u = dequeue[Q]
    process vertex u
    for each v ∈ Adj[u] do
        process edge (u,v)
        if state[v] = "undiscovered"
            state[v] = "discovered"
            p[v] = u
            enqueue[Q, v]
    state[u] = "processed"
```

Variáveis

	state	p
1		
2		
3		
4		
5		

Q =

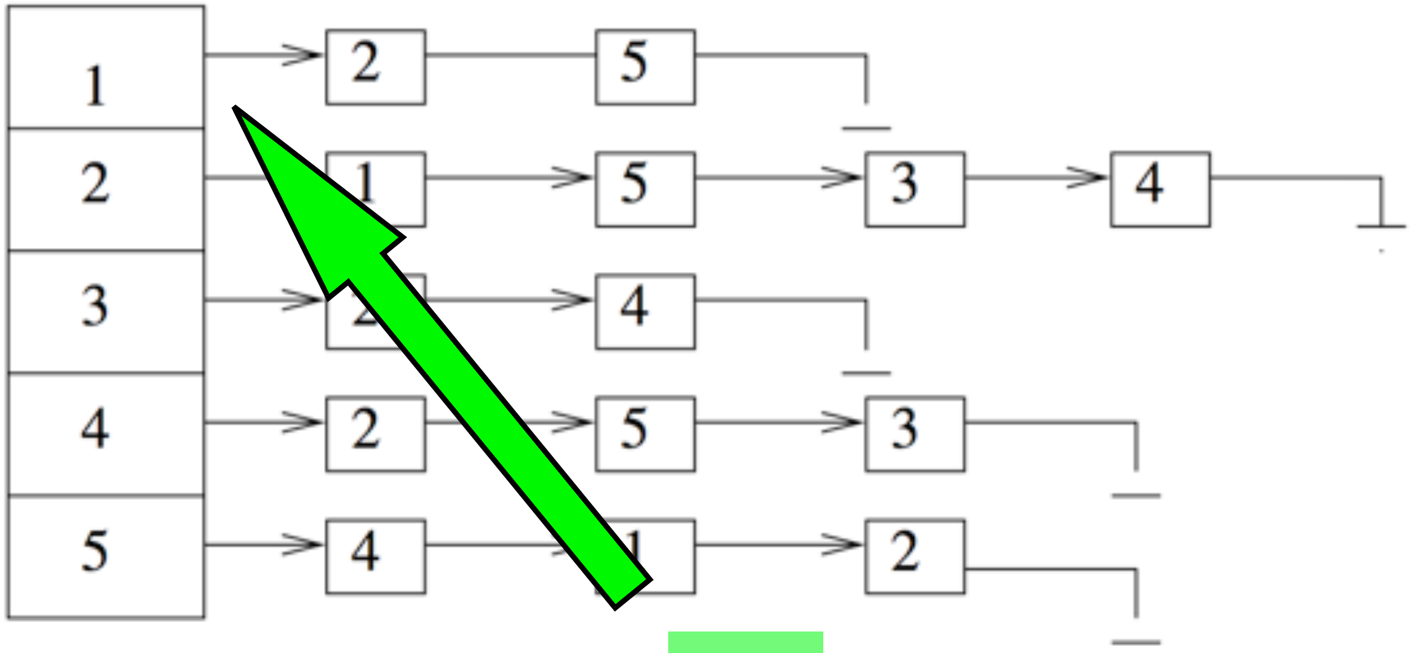
u =

v =

Saídas

Vértices:

Arestas:

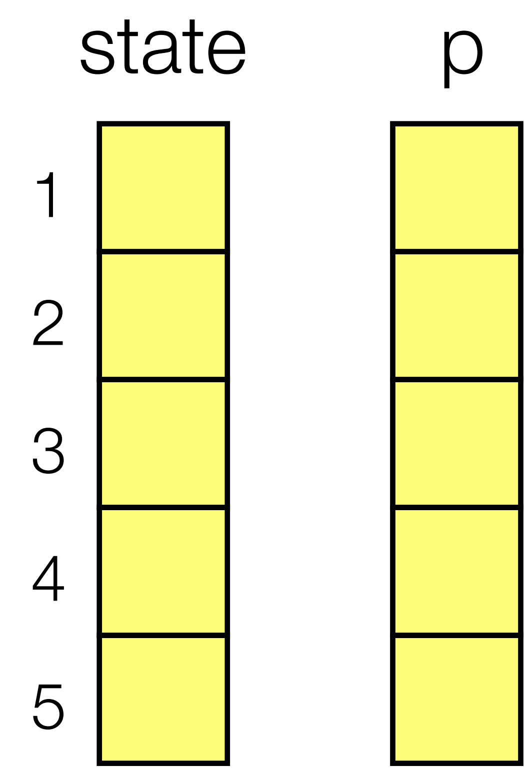


s=1

BFS(G, s)

```
for each vertex u ∈ V[G]−{s}
    state[u] = "undiscovered"
    p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
    u = dequeue[Q]
    process vertex u
    for each v ∈ Adj[u] do
        process edge (u,v)
        if state[v] = "undiscovered"
            state[v] = "discovered"
            p[v] = u
            enqueue[Q, v]
    state[u] = "processed"
```

Variáveis



Q =

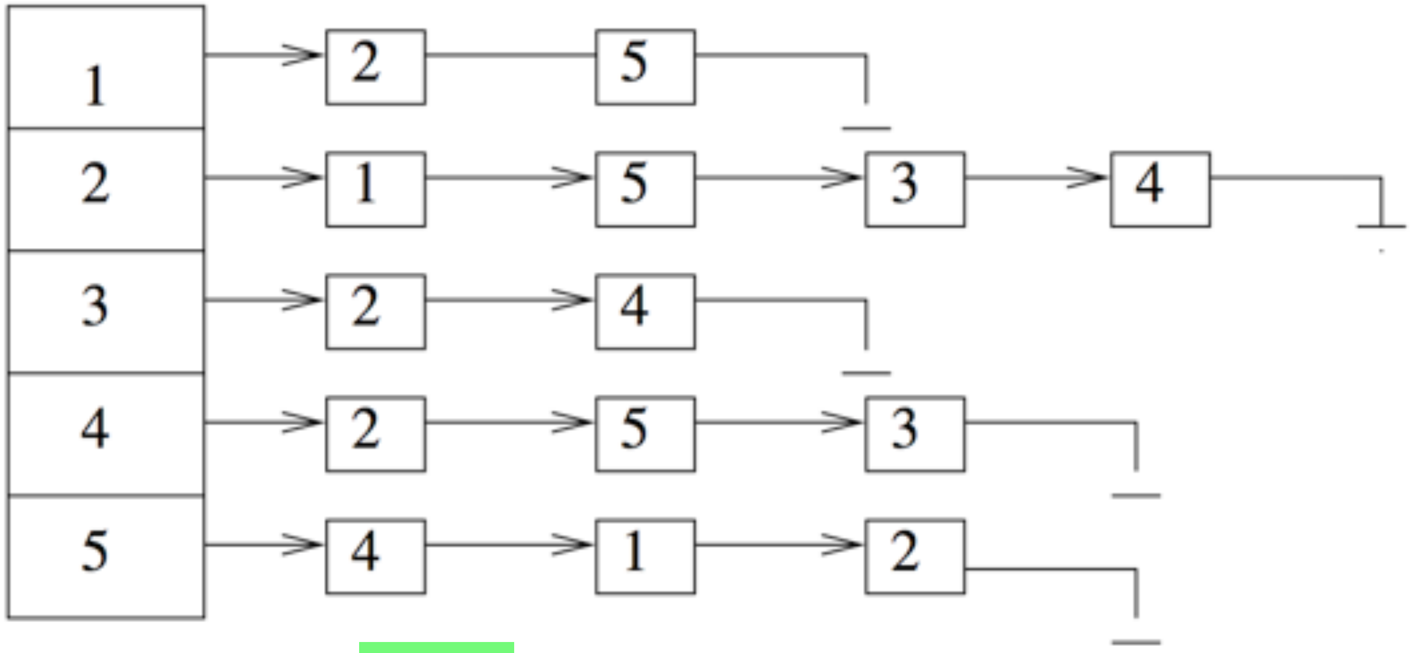
u =

v =

Saídas

Vértices:

Arestas:




```
BFS(G, s)   s=1
for each vertex u ∈ V[G]−{s}
     state[u] = “undiscovered”
     p[u] = null, // sem pais
state[s] = “discovered”
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∈ Adj[u] do
          process edge (u,v)
          if state[v] = “undiscovered”
               state[v] = “discovered”
               p[v] = u
               enqueue[Q, v]
     state[u] = “processed”
```

Variáveis

	state	p
1		
2	u	null
3	u	null
4	u	null
5	u	null

Q =

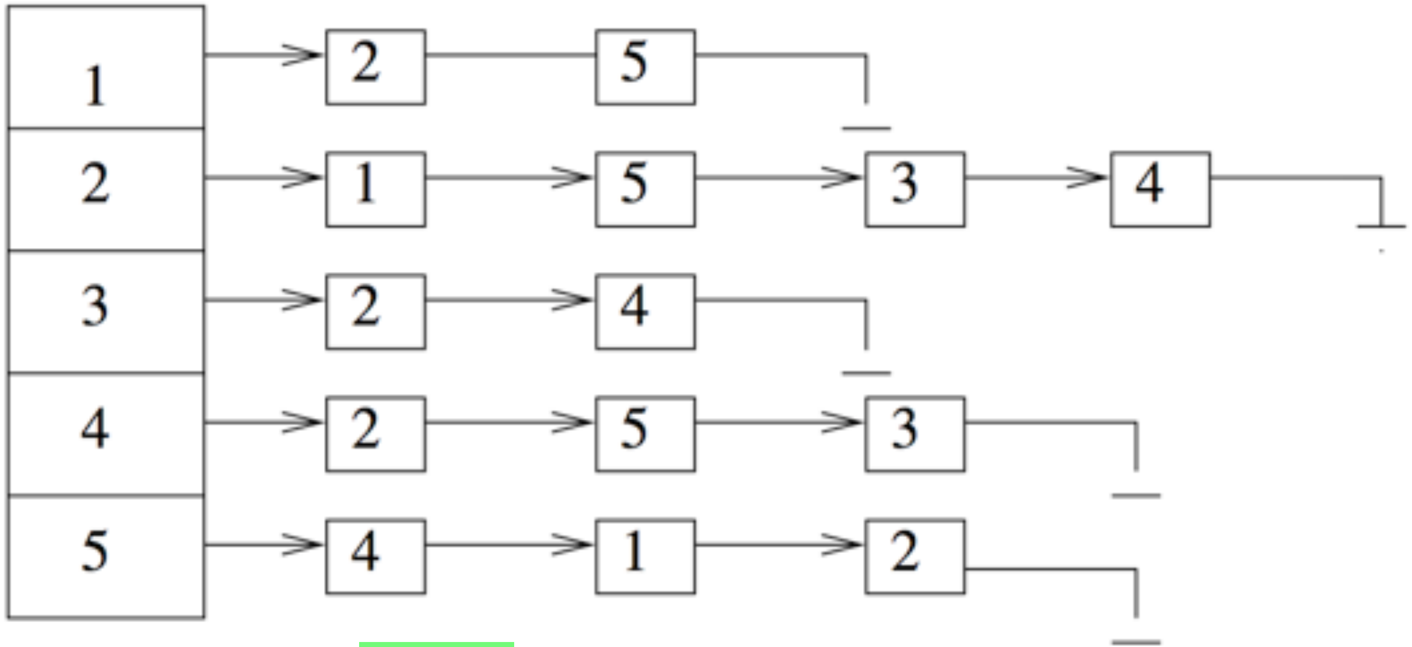
u =

v =

Saídas

Vértices:

Arestas:



BFS(G, s) s=1

```
for each vertex u ∈ V[G]−{s}
    state[u] = "undiscovered"
    p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
    u = dequeue[Q]
    process vertex u
    for each v ∈ Adj[u] do
        process edge (u,v)
        if state[v] = "undiscovered"
            state[v] = "discovered"
            p[v] = u
            enqueue[Q, v]
    state[u] = "processed"
```

Variáveis

	state	p
1	d	null
2	u	null
3	u	null
4	u	null
5	u	null

Q = {1}

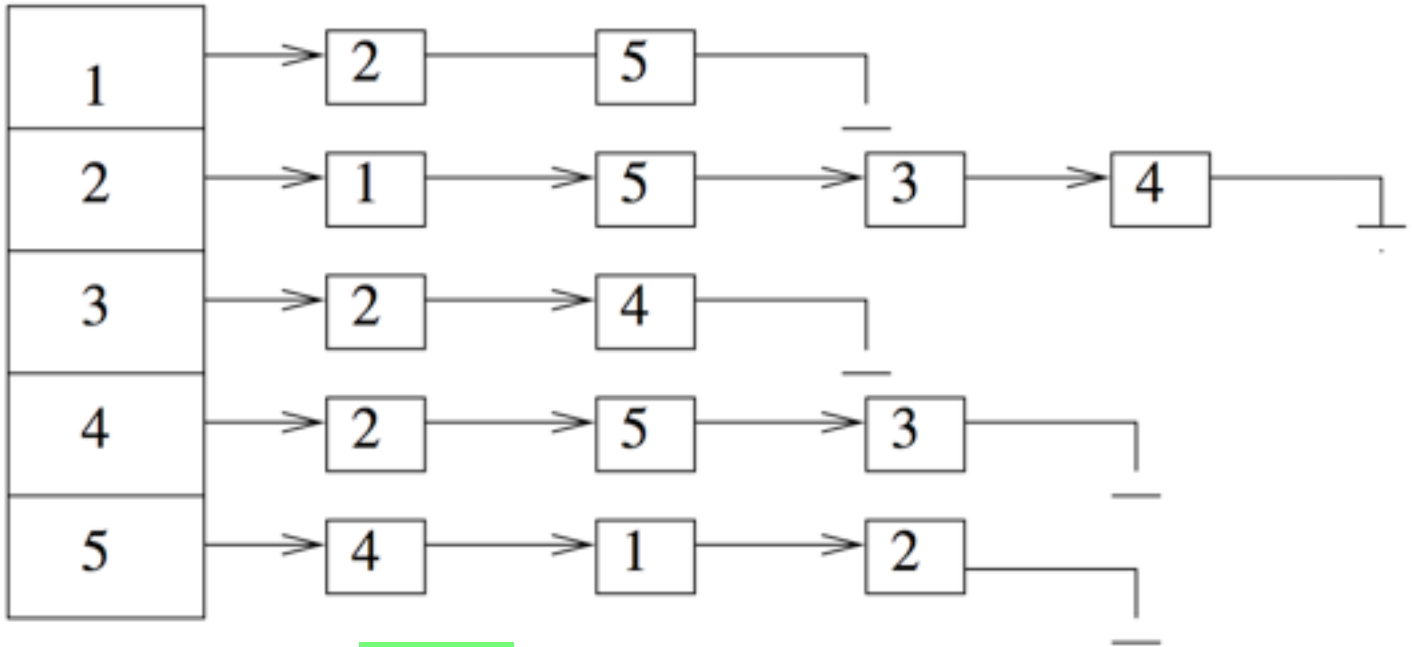
u =

v =

Saídas

Vértices:

Arestas:




```
BFS(G, s)   s=1
for each vertex u ∈ V[G]−{s}
     state[u] = "undiscovered"
     p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∈ Adj[u] do
          process edge (u,v)
          if state[v] = "undiscovered"
               state[v] = "discovered"
               p[v] = u
               enqueue[Q, v]
     state[u] = "processed"
```

Variáveis

	state	p
1	d	null
2	u	null
3	u	null
4	u	null
5	u	null

Q = {1}

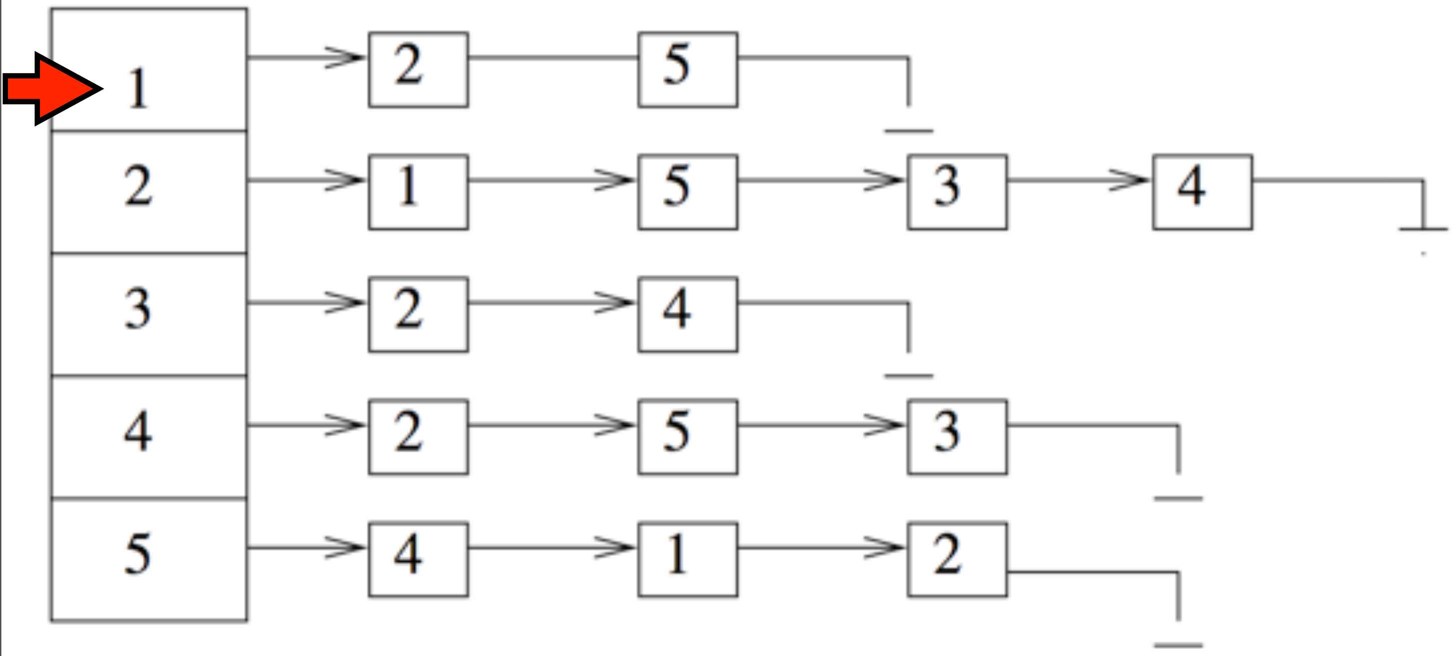
u =

v =

Saídas

Vértices:

Arestas:




```
BFS(G, s)
for each vertex u ∈ V[G]−{s}
     state[u] = "undiscovered"
     p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∈ Adj[u] do
          process edge (u,v)
          if state[v] = "undiscovered"
               state[v] = "discovered"
               p[v] = u
               enqueue[Q, v]
     state[u] = "processed"
```

Variáveis

	state	p
1	u	null
2	u	null
3	u	null
4	u	null
5	u	null

Q =

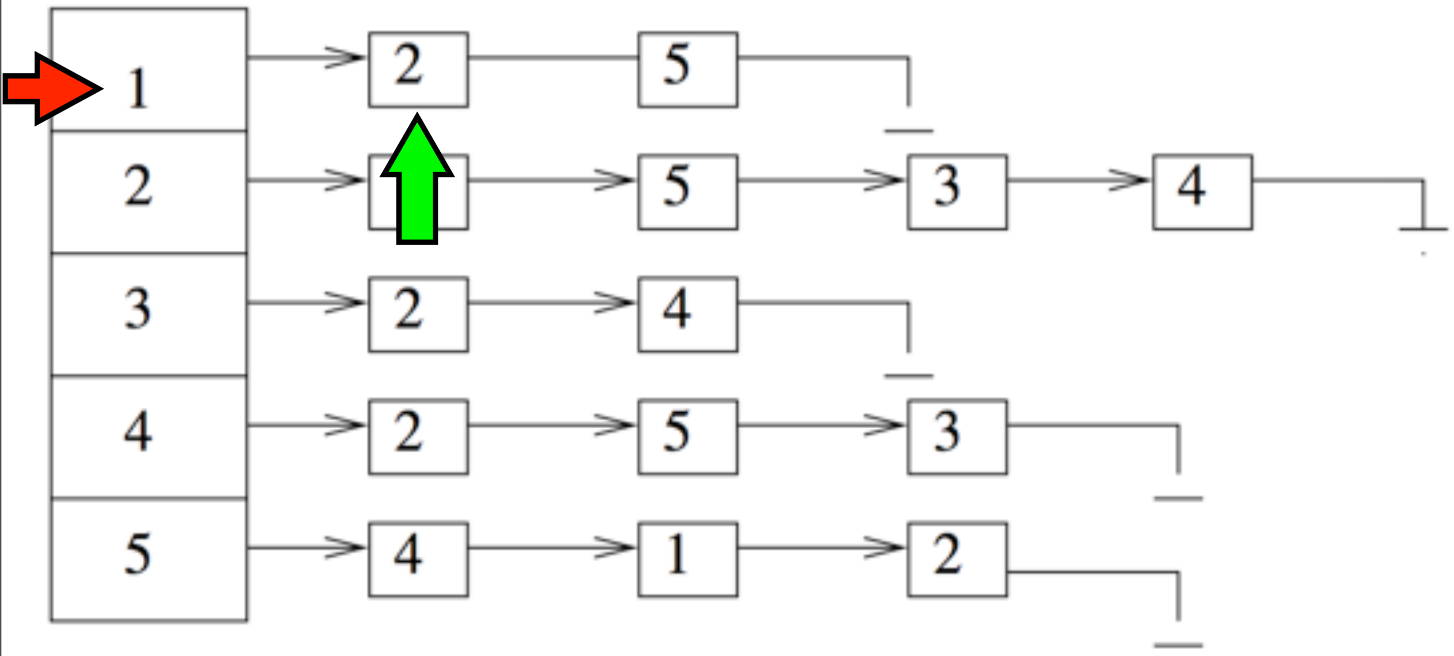
u = 1

v =

Saídas

Vértices: 1

Arestas:




```
BFS(G, s)
for each vertex u ∈ V[G]−{s}
     state[u] = "undiscovered"
     p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∈ Adj[u] do
          process edge (u,v)
          if state[v] = "undiscovered"
               state[v] = "discovered"
               p[v] = u
               enqueue[Q, v]
     state[u] = "processed"
```

Variáveis

	state	p
1	d	null
2	u	null
3	u	null
4	u	null
5	u	null

Q =

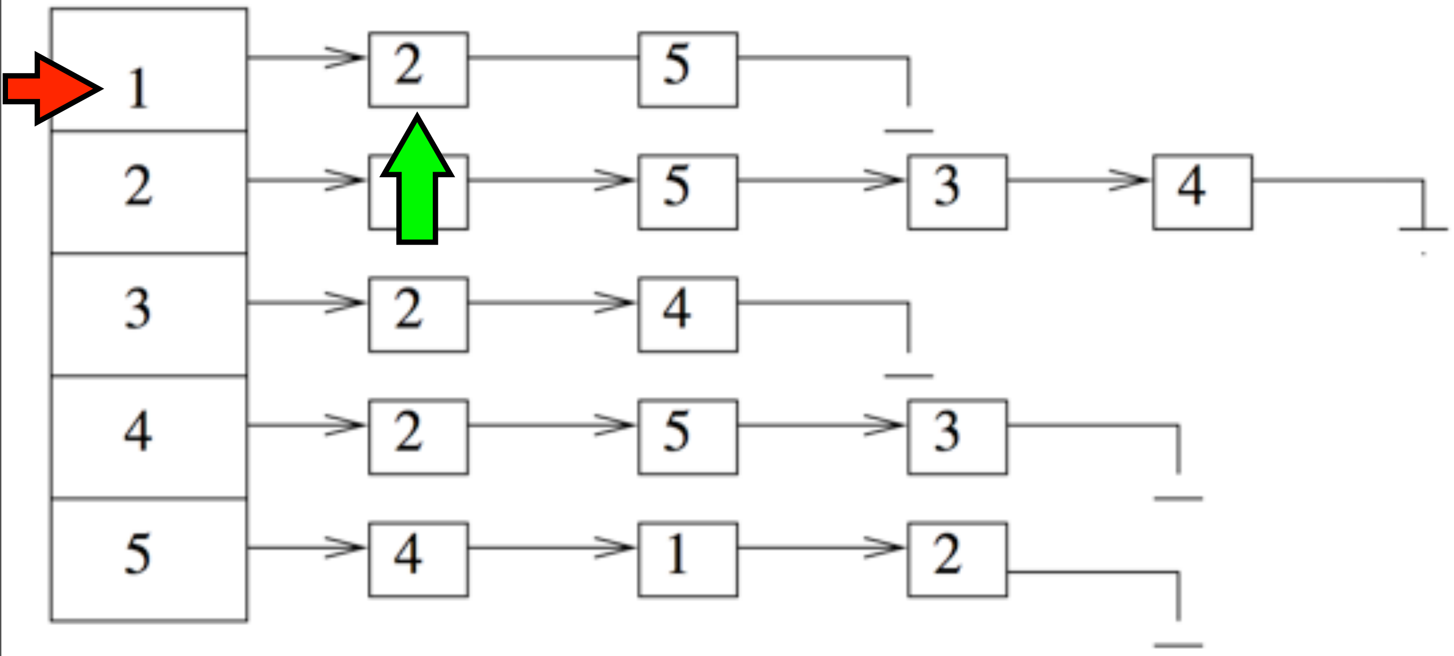
u = 1

v = 2

Saídas

Vértices: 1

Arestas: (1,2)




```
BFS(G, s)
for each vertex u ∈ V[G]−{s}
    state[u] = "undiscovered"
    p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
    u = dequeue[Q]
    process vertex u
    for each v ∈ Adj[u] do
        process edge (u,v)
        if state[v] = "undiscovered"
            state[v] = "discovered"
            p[v] = u
            enqueue[Q, v]
    state[u] = "processed"
```

Variáveis

	state	p
1	d	null
2	u	null
3	u	null
4	u	null
5	u	null

Q =

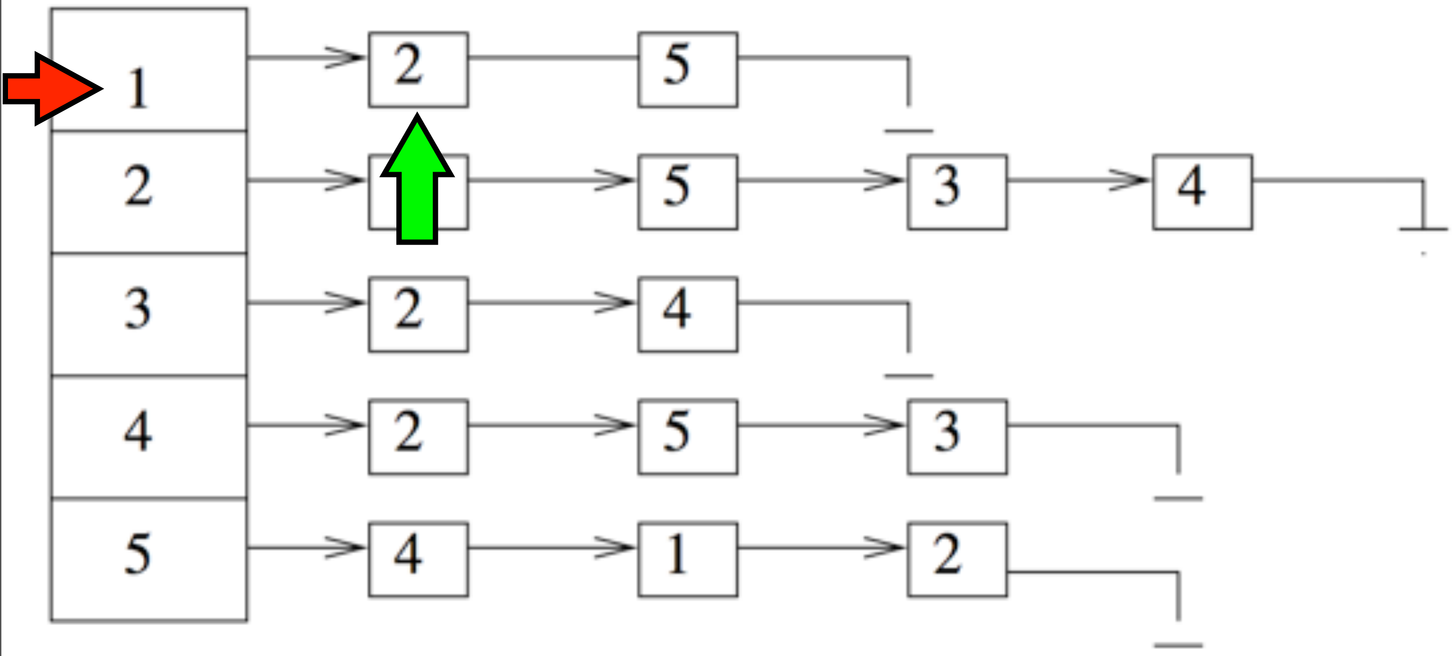
u = 1

v = 2

Saídas

Vértices: 1

Arestas: (1,2)




```
BFS(G, s)
for each vertex u ∈ V[G]−{s}
     state[u] = "undiscovered"
     p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∈ Adj[u] do
          process edge (u,v)
          if state[v] = "undiscovered"
               state[v] = "discovered"
               p[v] = u
               enqueue[Q, v]
     state[u] = "processed"
```

Variáveis

	state	p
1	d	null
2	d	1
3	u	null
4	u	null
5	u	null

Q = {2}

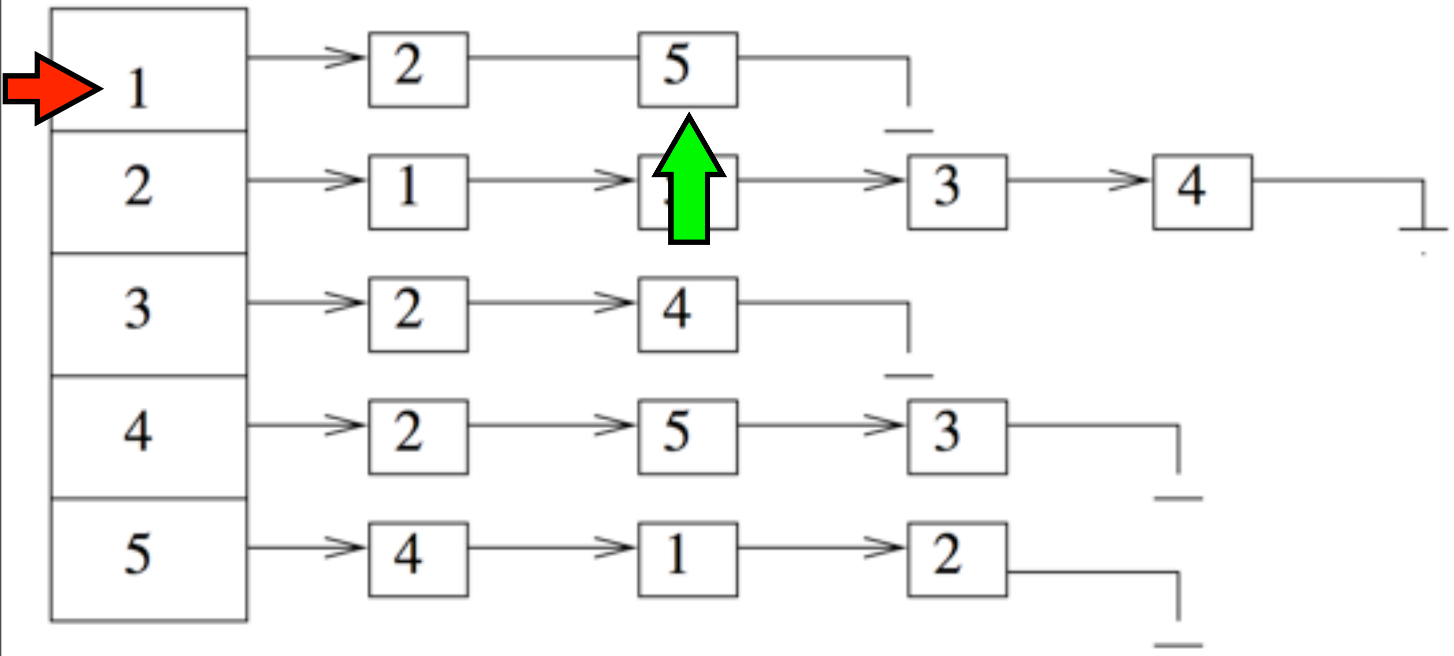
u = 1

v = 2

Saídas

Vértices: 1

Arestas: (1,2)




```
BFS(G, s)
for each vertex u ∊ V[G]−{s}
     state[u] = “undiscovered”
     p[u] = null, // sem pais
state[s] = “discovered”
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∊ Adj[u] do
          process edge (u,v)
          if state[v] = “undiscovered”
               state[v] = “discovered”
               p[v] = u
               enqueue[Q, v]
     state[u] = “processed”
```

Variáveis

	state	p
1	d	null
2	d	1
3	u	null
4	u	null
5	u	null

Q = {2}

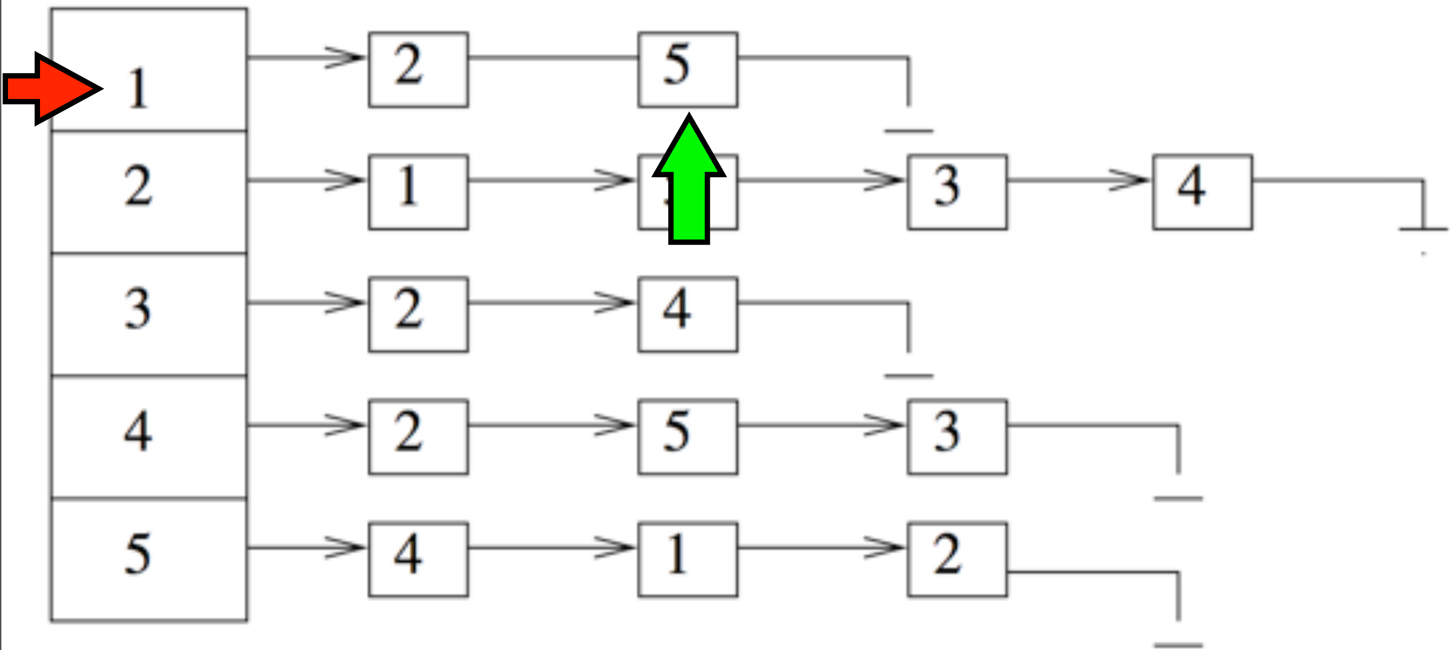
u = 1

v = 5

Saídas

Vértices: 1

Arestas: (1,2) (1,5)




```
BFS(G, s)
for each vertex u ∈ V[G]−{s}
     state[u] = “undiscovered”
     p[u] = null, // sem pais
state[s] = “discovered”
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∈ Adj[u] do
          process edge (u,v)
          if state[v] = “undiscovered”
               state[v] = “discovered”
               p[v] = u
               enqueue[Q, v]
     state[u] = “processed”
```

Variáveis

	state	p
1	d	null
2	d	1
3	u	null
4	u	null
5	u	null

Q = {2}

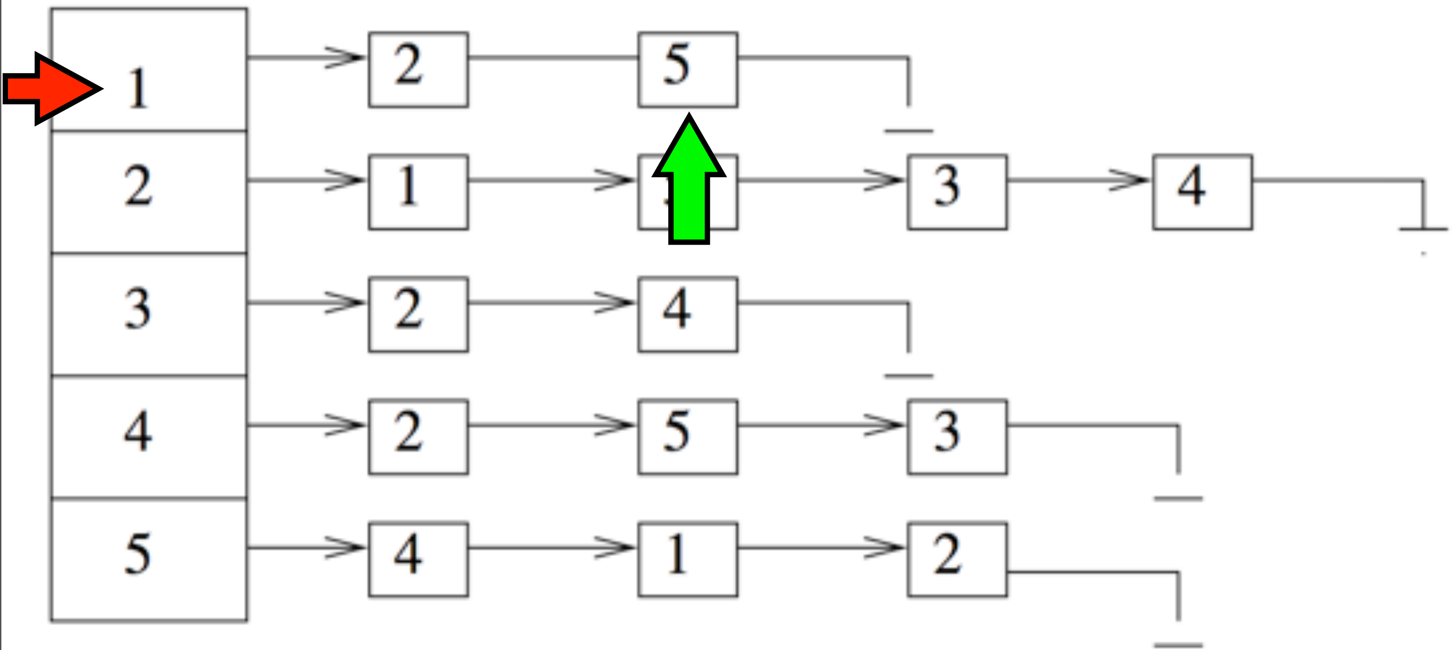
u = 1

v = 5

Saídas

Vértices: 1

Arestas: (1,2) (1,5)



```
BFS(G, s)
for each vertex u ∊ V[G]−{s}
     state[u] = “undiscovered”
     p[u] = null, // sem pais
state[s] = “discovered”
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∊ Adj[u] do
          process edge (u,v)
          if state[v] = “undiscovered”
               state[v] = “discovered”
               p[v] = u
               enqueue[Q, v]
     state[u] = “processed”
```

Variáveis

	state	p
1	d	null
2	d	1
3	u	null
4	u	null
5	d	1

Q = {2,5}

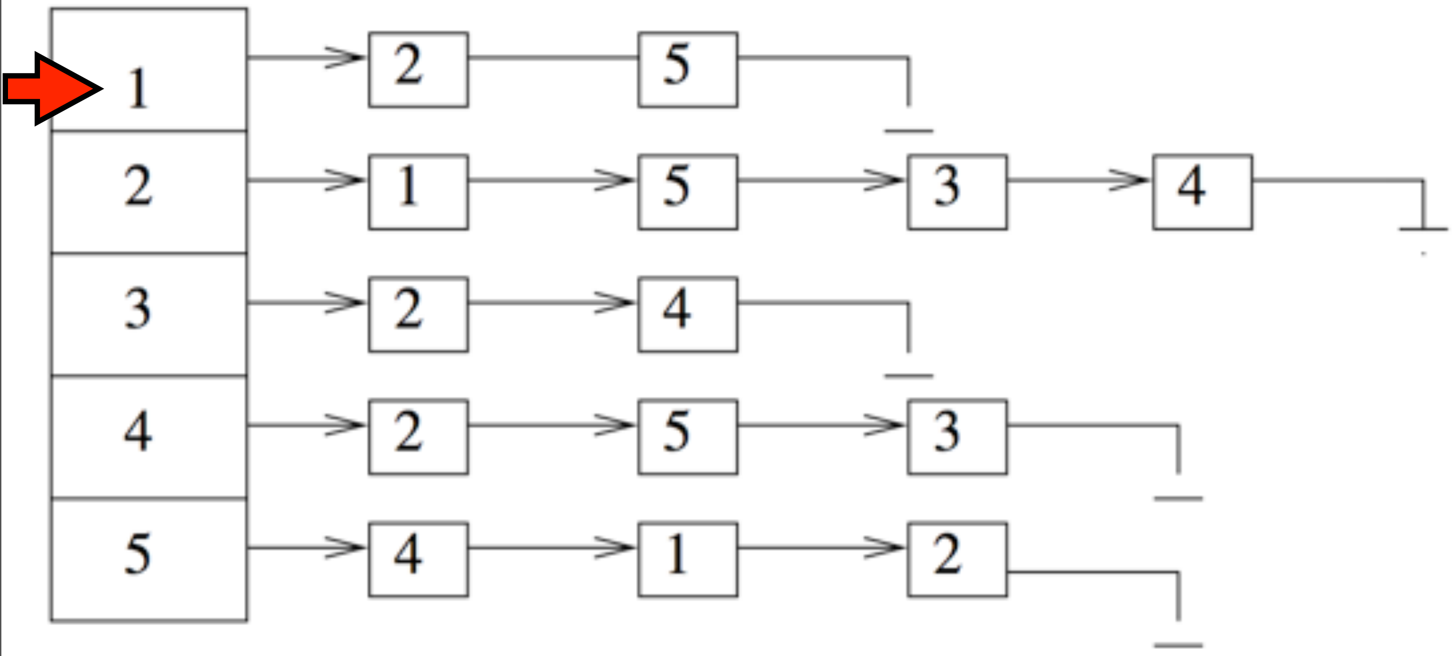
u = 1

v = 5

Saídas

Vértices: 1

Arestas: (1,2) (1,5)




```
BFS(G, s)
for each vertex u ∈ V[G]−{s}
    state[u] = "undiscovered"
    p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
    u = dequeue[Q]
    process vertex u
    for each v ∈ Adj[u] do
        process edge (u,v)
        if state[v] = "undiscovered"
            state[v] = "discovered"
            p[v] = u
            enqueue[Q, v]
    state[u] = "processed"
```

Variáveis

	state	p
1	p	null
2	d	1
3	u	null
4	u	null
5	d	1

Q = {2,5}

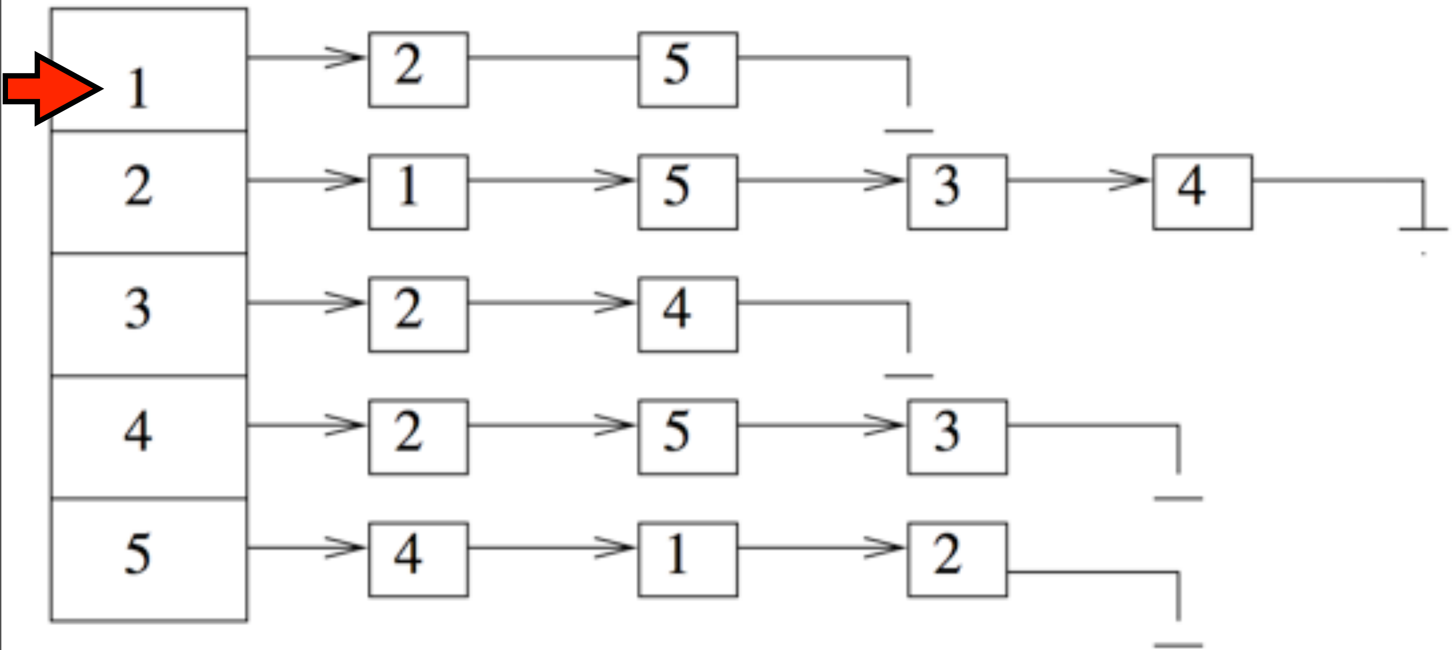
u = 1

v = 5

Saídas

Vértices: 1

Arestas: (1,2) (1,5)




```
BFS(G, s)
for each vertex u ∊ V[G]−{s}
     state[u] = "undiscovered"
     p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∊ Adj[u] do
          process edge (u,v)
          if state[v] = "undiscovered"
               state[v] = "discovered"
               p[v] = u
               enqueue[Q, v]
     state[u] = "processed"
```

Variáveis

	state	p
1	p	null
2	d	1
3	u	null
4	u	null
5	d	1

Q = {2,5}

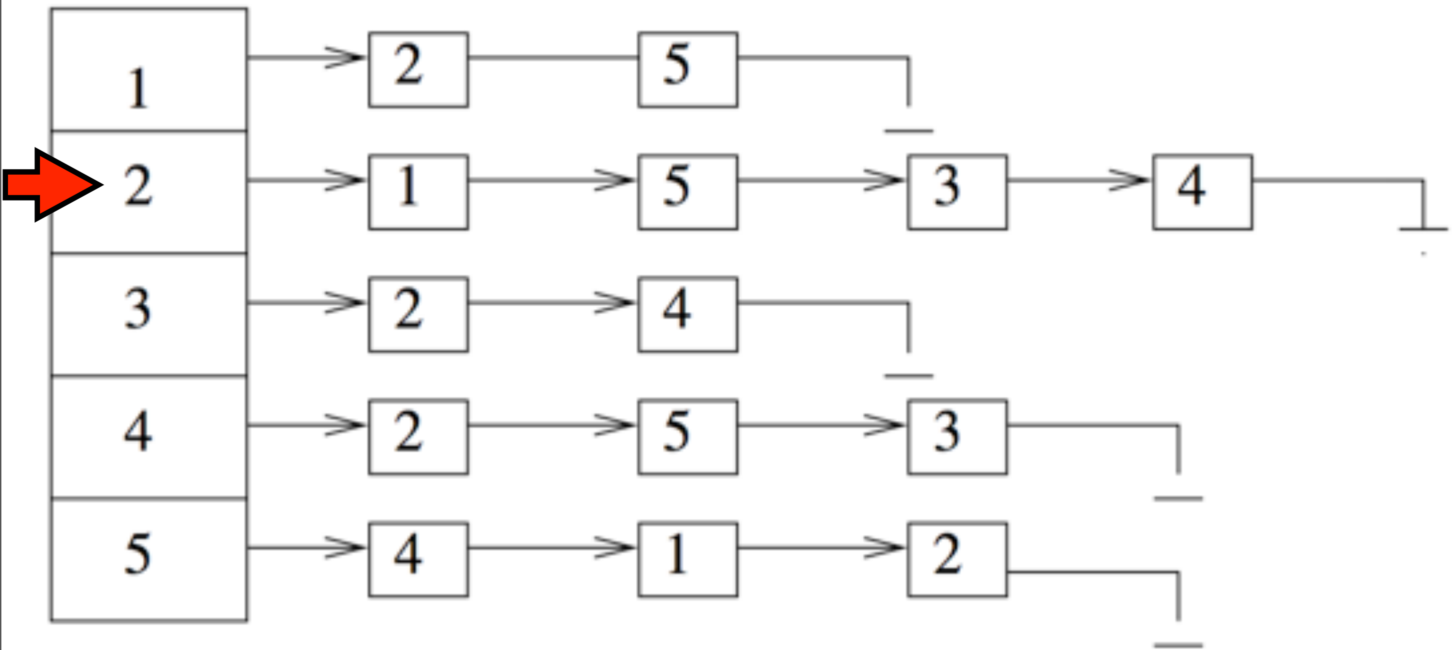
u = 1

v = 5

Saídas

Vértices: 1

Arestas: (1,2) (1,5)




```
BFS(G, s)
for each vertex u ∈ V[G]−{s}
     state[u] = "undiscovered"
     p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∈ Adj[u] do
          process edge (u,v)
          if state[v] = "undiscovered"
               state[v] = "discovered"
               p[v] = u
               enqueue[Q, v]
     state[u] = "processed"
```

Variáveis

	state	p
1	p	null
2	d	1
3	u	null
4	u	null
5	d	1

Q = {5}

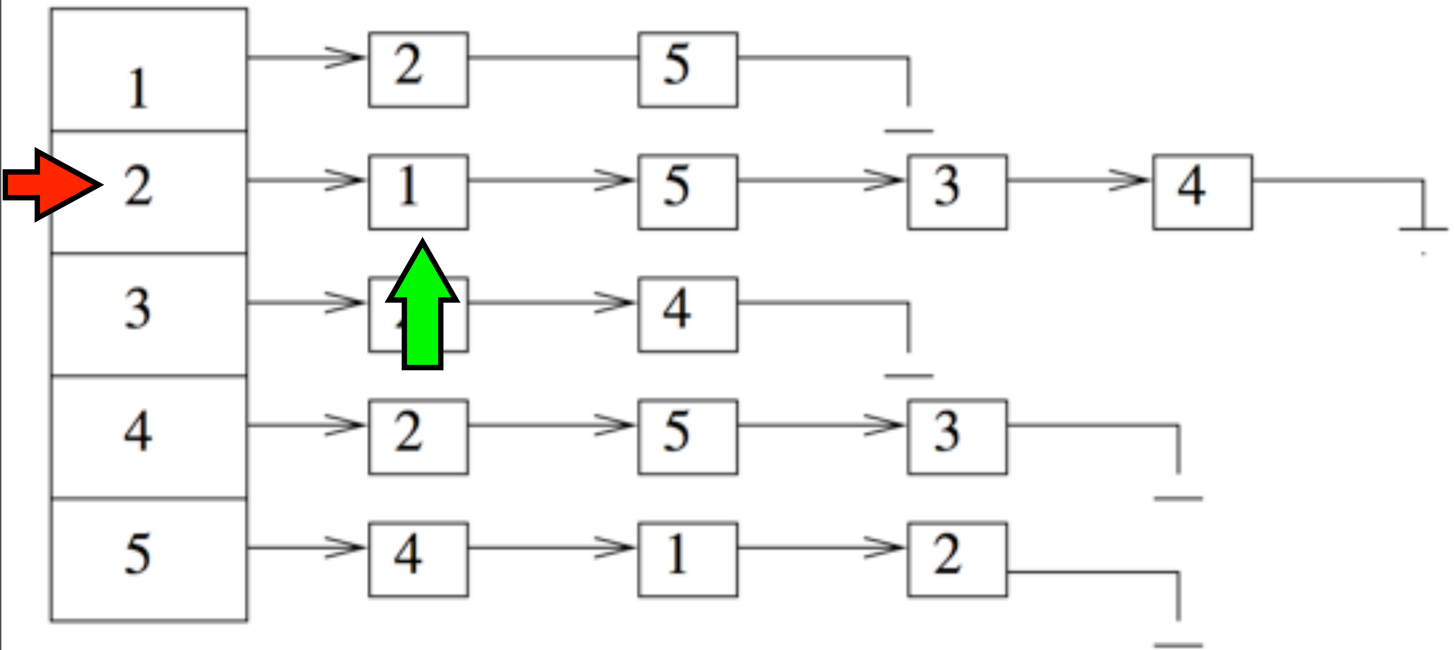
u = 2

v = 5

Saídas

Vértices: 1, 2

Arestas: (1,2) (1,5)




```
BFS(G, s)
for each vertex u ∈ V[G]−{s}
     state[u] = "undiscovered"
     p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∈ Adj[u] do
          process edge (u,v)
          if state[v] = "undiscovered"
               state[v] = "discovered"
               p[v] = u
               enqueue[Q, v]
     state[u] = "processed"
```

Variáveis

	state	p
1	p	null
2	d	1
3	u	null
4	u	null
5	d	1

Q = {5}

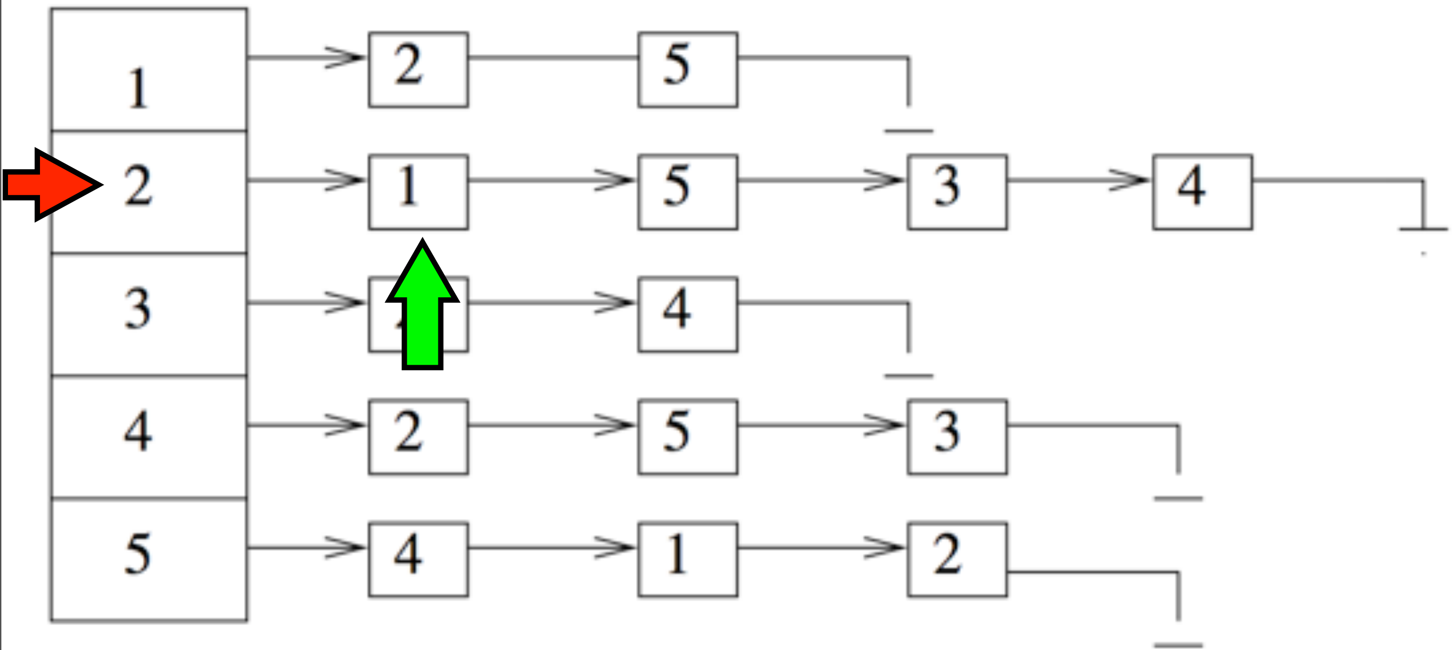
u = 2

v = 1

Saídas

Vértices: 1, 2

Arestas: (1,2) (1,5) (2,1)




```
BFS(G, s)
for each vertex u ∈ V[G]−{s}
     state[u] = "undiscovered"
     p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∈ Adj[u] do
          process edge (u,v)
          if state[v] = "undiscovered"
               state[v] = "discovered"
               p[v] = u
               enqueue[Q, v]
     state[u] = "processed"
```

Variáveis

	state	p
1	p	null
2	d	1
3	u	null
4	u	null
5	d	1

Q = {5}

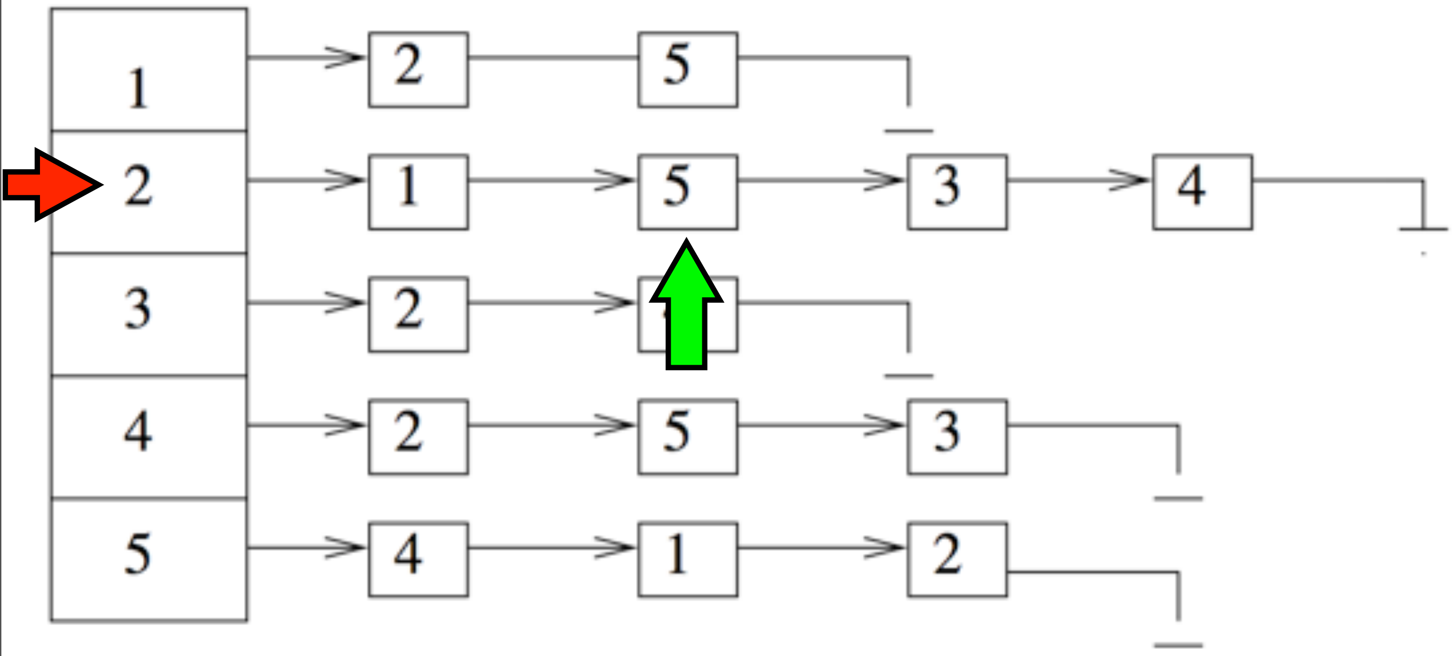
u = 2

v = 1

Saídas

Vértices: 1, 2

Arestas: (1,2) (1,5) (2,1)



BFS(G, s)
for each vertex u ∈ V[G]−{s}
    state[u] = "undiscovered"
    p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
    u = dequeue[Q]
    process vertex u
    for each v ∈ Adj[u] do
        process edge (u,v)
        if state[v] = "undiscovered"
            state[v] = "discovered"
            p[v] = u
            enqueue[Q, v]
    state[u] = "processed"

Variáveis

	state	p
1	p	null
2	d	1
3	u	null
4	u	null
5	d	1

Q = {5}

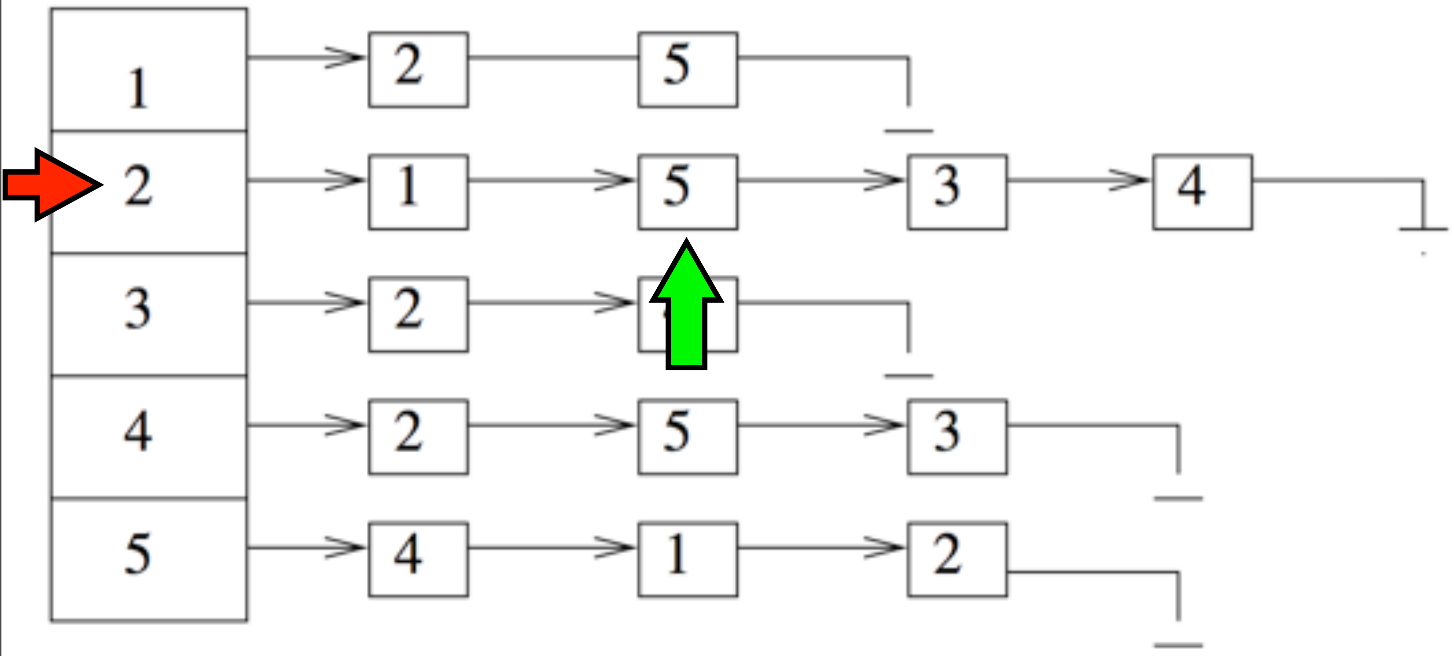
u = 2

v = 5

Saídas

Vértices: 1, 2

Arestas: (1,2) (1,5) (2,1) (2,5)



BFS(G, s)
for each vertex u ∊ V[G]−{s}
    state[u] = "undiscovered"
    p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
    u = dequeue[Q]
    process vertex u
    for each v ∊ Adj[u] do
        process edge (u,v)
        if state[v] = "undiscovered"
            state[v] = "discovered"
            p[v] = u
            enqueue[Q, v]
    state[u] = "processed"

Variáveis

	state	p
1	p	null
2	d	1
3	u	null
4	u	null
5	d	1

Q = {5}

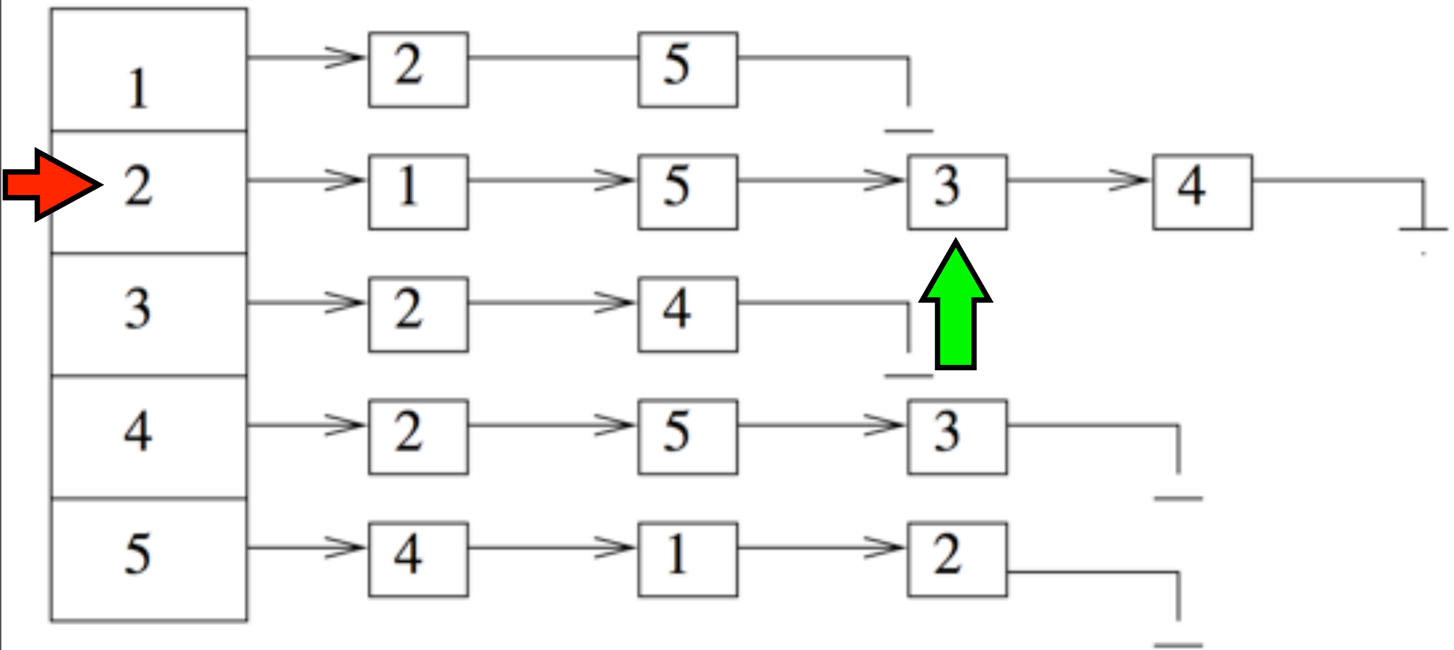
u = 2

v = 5

Saídas

Vértices: 1, 2

Arestas: (1,2) (1,5) (2,1) (2,5)




```
BFS(G, s)
for each vertex u ∈ V[G]−{s}
    state[u] = "undiscovered"
    p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
    u = dequeue[Q]
    process vertex u
    for each v ∈ Adj[u] do
        process edge (u,v)
        if state[v] = "undiscovered"
            state[v] = "discovered"
            p[v] = u
            enqueue[Q, v]
    state[u] = "processed"
```

Variáveis

	state	p
1	p	null
2	d	1
3	u	null
4	u	null
5	d	1

Q = {5}

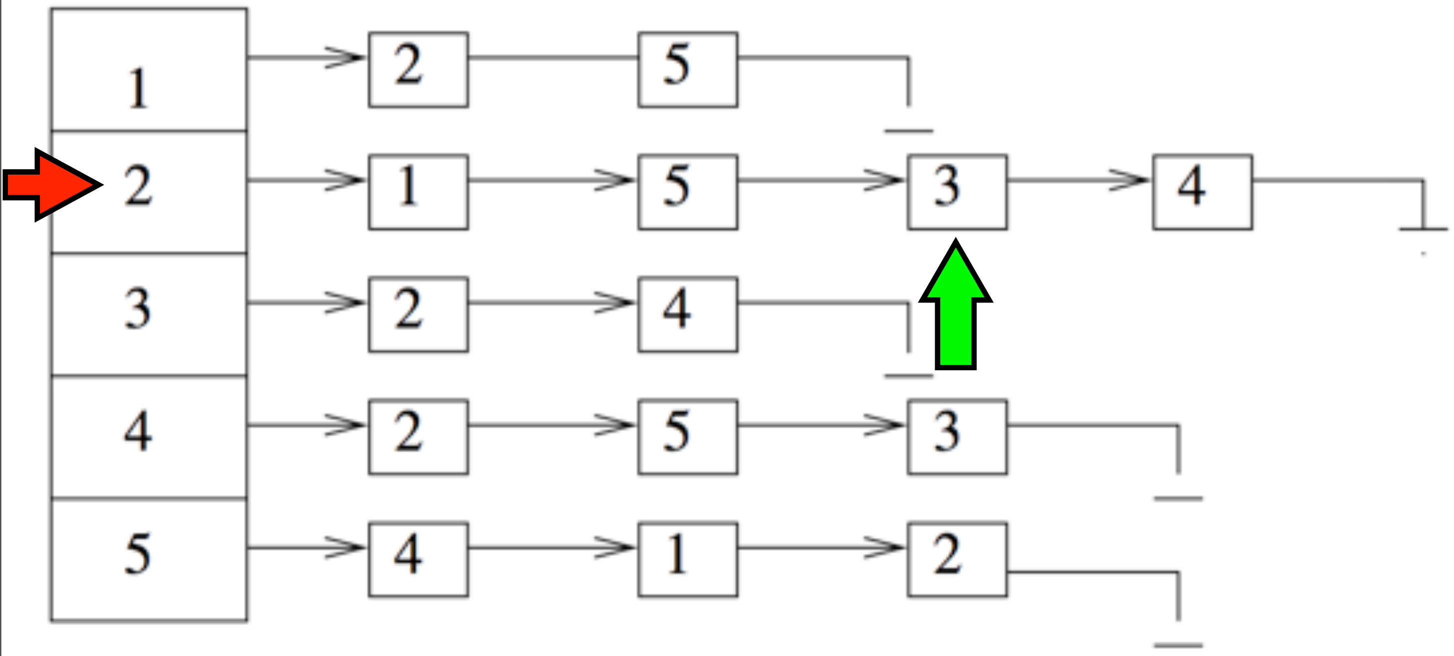
u = 2

v = 3

Saídas

Vértices: 1, 2

Arestas: (1,2) (1,5) (2,1) (2,5) (2,3)




```
BFS(G, s)
for each vertex u ∈ V[G]−{s}
     state[u] = "undiscovered"
     p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∈ Adj[u] do
          process edge (u,v)
          if state[v] = "undiscovered"
               state[v] = "discovered"
               p[v] = u
               enqueue[Q, v]
     state[u] = "processed"
```

Variáveis

	state	p
1	p	null
2	d	1
3	u	null
4	u	null
5	d	1

Q = {5}

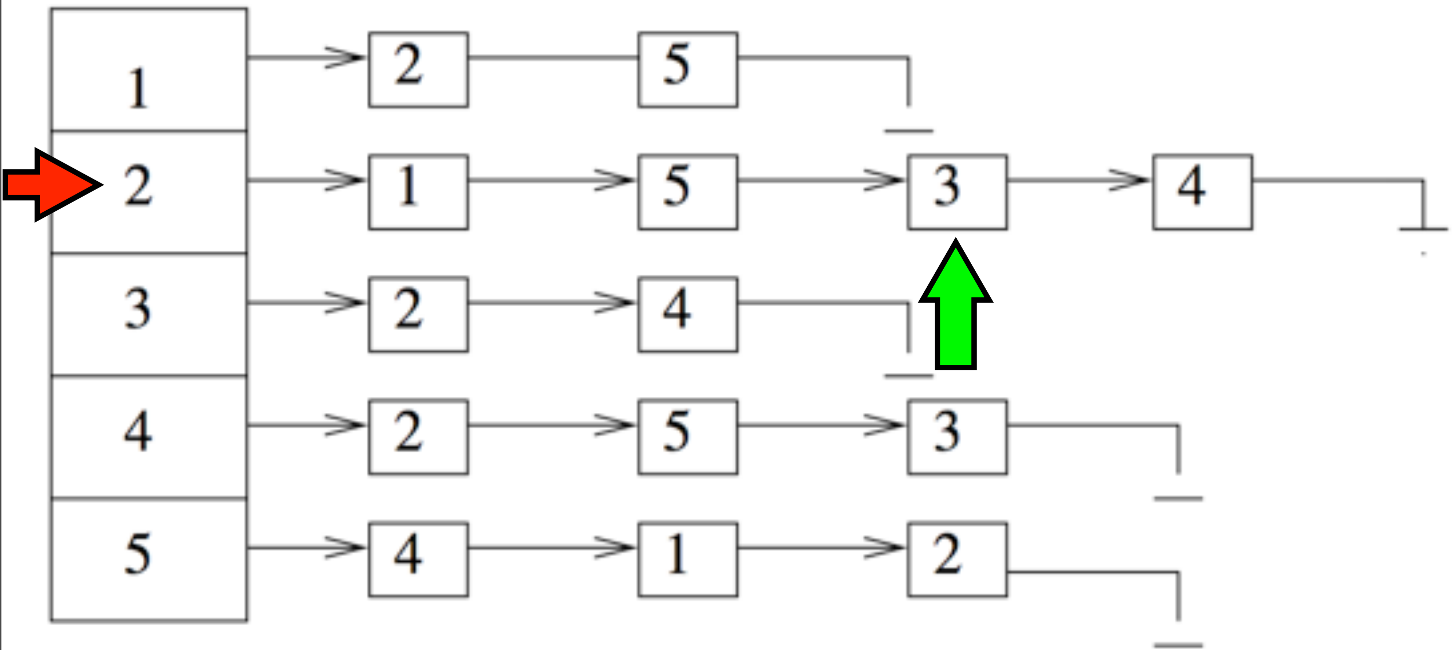
u = 2

v = 3

Saídas

Vértices: 1, 2

Arestas: (1,2) (1,5) (2,1) (2,5) (2,3)




```
BFS(G, s)
for each vertex u ∈ V[G]−{s}
    state[u] = "undiscovered"
    p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
    u = dequeue[Q]
    process vertex u
    for each v ∈ Adj[u] do
        process edge (u,v)
        if state[v] = "undiscovered"
            state[v] = "discovered"
            p[v] = u
            enqueue[Q, v]
    state[u] = "processed"
```

Variáveis

	state	p
1	p	null
2	d	1
3	d	2
4	u	null
5	d	1

Q = {5,3}

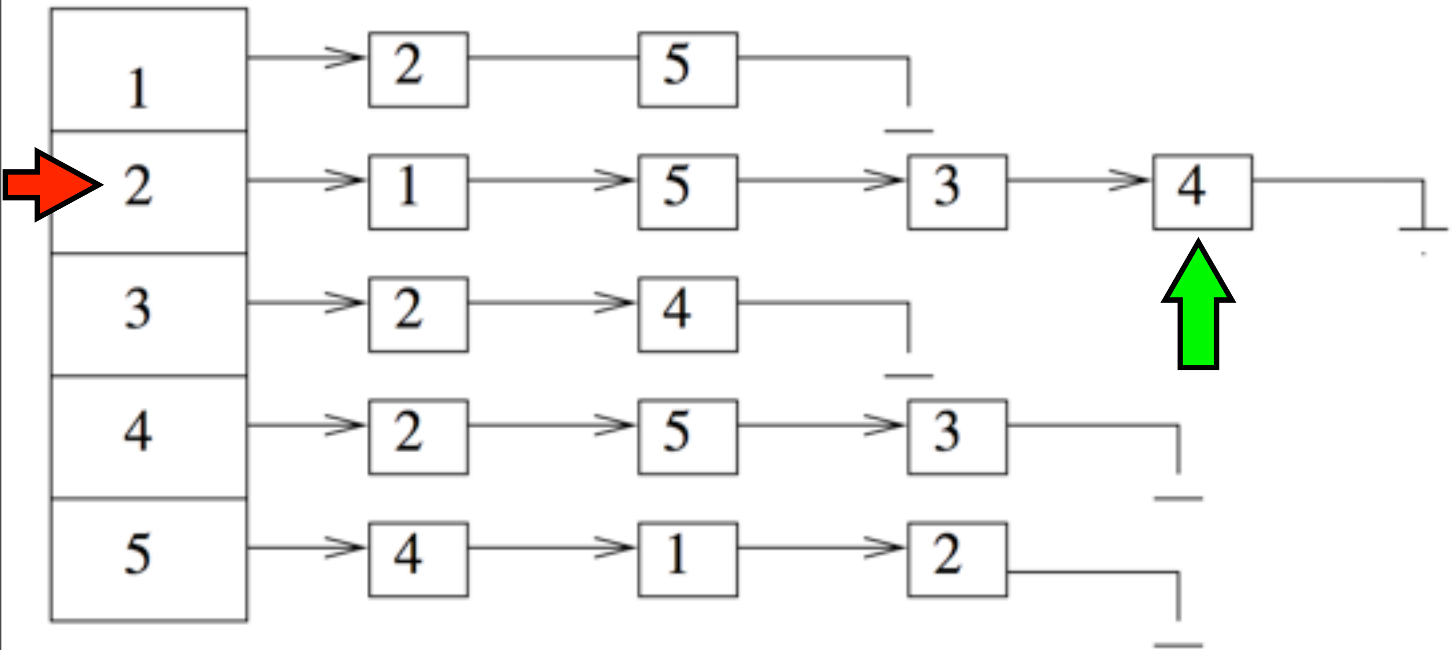
u = 2

v = 3

Saídas

Vértices: 1, 2

Arestas: (1,2) (1,5) (2,1) (2,5) (2,3)




```
BFS(G, s)
for each vertex u ∊ V[G]−{s}
     state[u] = “undiscovered”
     p[u] = null, // sem pais
state[s] = “discovered”
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∊ Adj[u] do
          process edge (u,v)
          if state[v] = “undiscovered”
               state[v] = “discovered”
               p[v] = u
               enqueue[Q, v]
     state[u] = “processed”
```

Variáveis

	state	p
1	p	null
2	d	1
3	d	2
4	u	null
5	d	1

Q = {5,3}

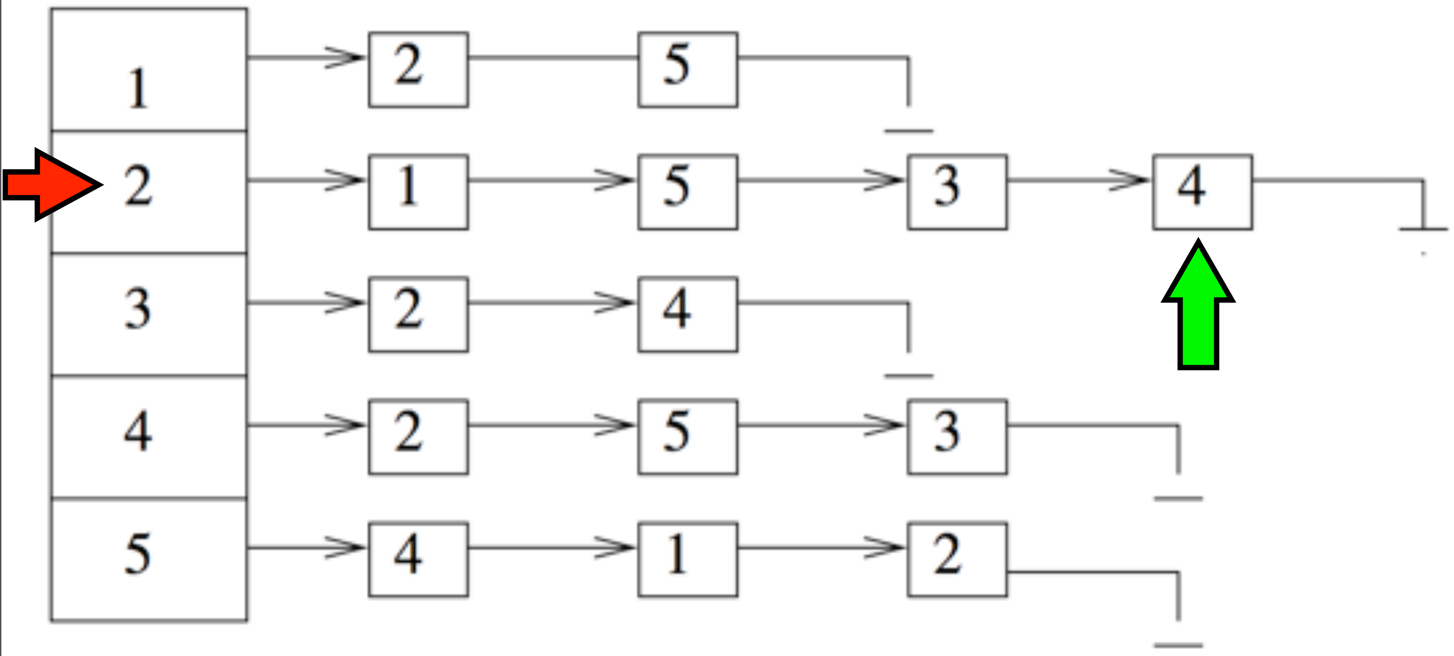
u = 2

v = 4

Saídas

Vértices: 1, 2

Arestas: (1,2) (1,5) (2,1) (2,5) (2,3) (2,4)




```
BFS(G, s)
for each vertex u ∈ V[G]−{s}
     state[u] = "undiscovered"
     p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∈ Adj[u] do
          process edge (u,v)
          if state[v] = "undiscovered"
               state[v] = "discovered"
               p[v] = u
               enqueue[Q, v]
     state[u] = "processed"
```

Variáveis

	state	p
1	p	null
2	d	1
3	d	2
4	u	null
5	d	1

Q = {5,3}

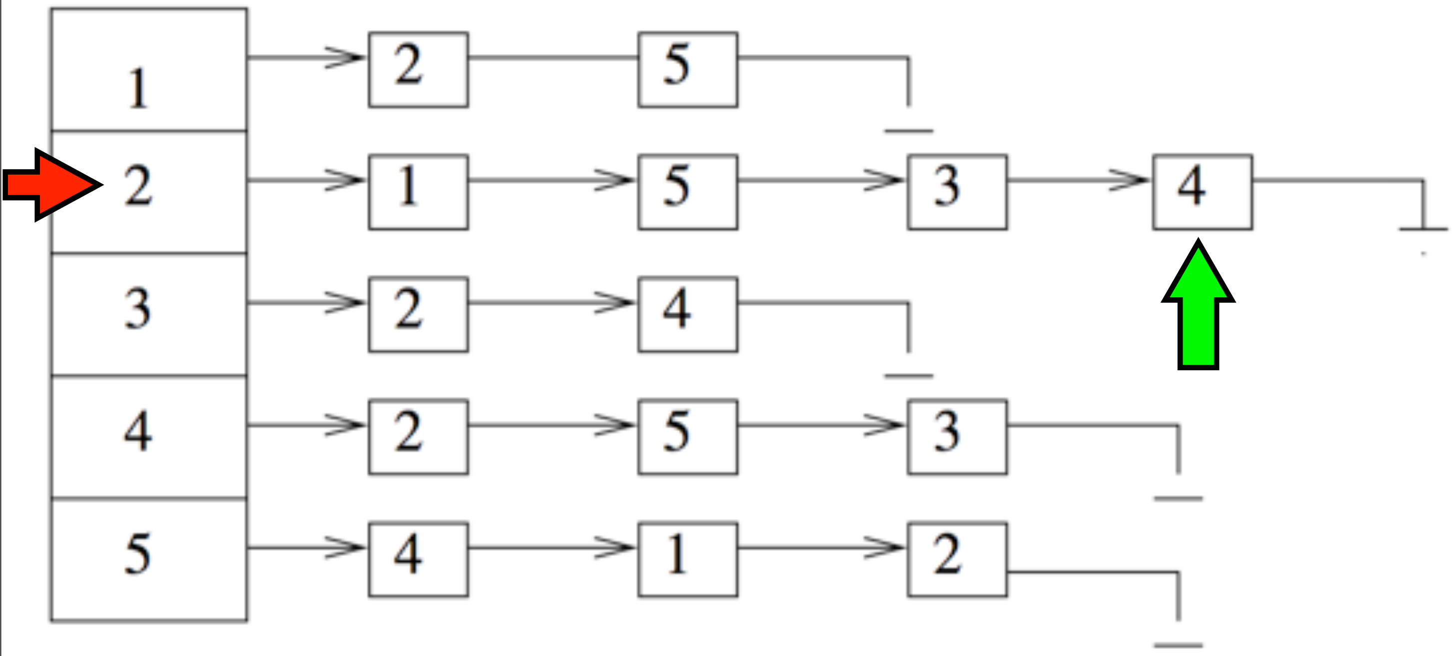
u = 2

v = 4

Saídas

Vértices: 1, 2

Arestas: (1,2) (1,5) (2,1) (2,5) (2,3) (2,4)



```
BFS(G, s)
for each vertex u ∈ V[G]−{s}
     state[u] = "undiscovered"
     p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∈ Adj[u] do
          process edge (u,v)
          if state[v] = "undiscovered"
               state[v] = "discovered"
               p[v] = u
               enqueue[Q, v]
     state[u] = "processed"
```

Variáveis

	state	p
1	p	null
2	d	1
3	d	2
4	d	2
5	d	1

Q = {5,3,4}

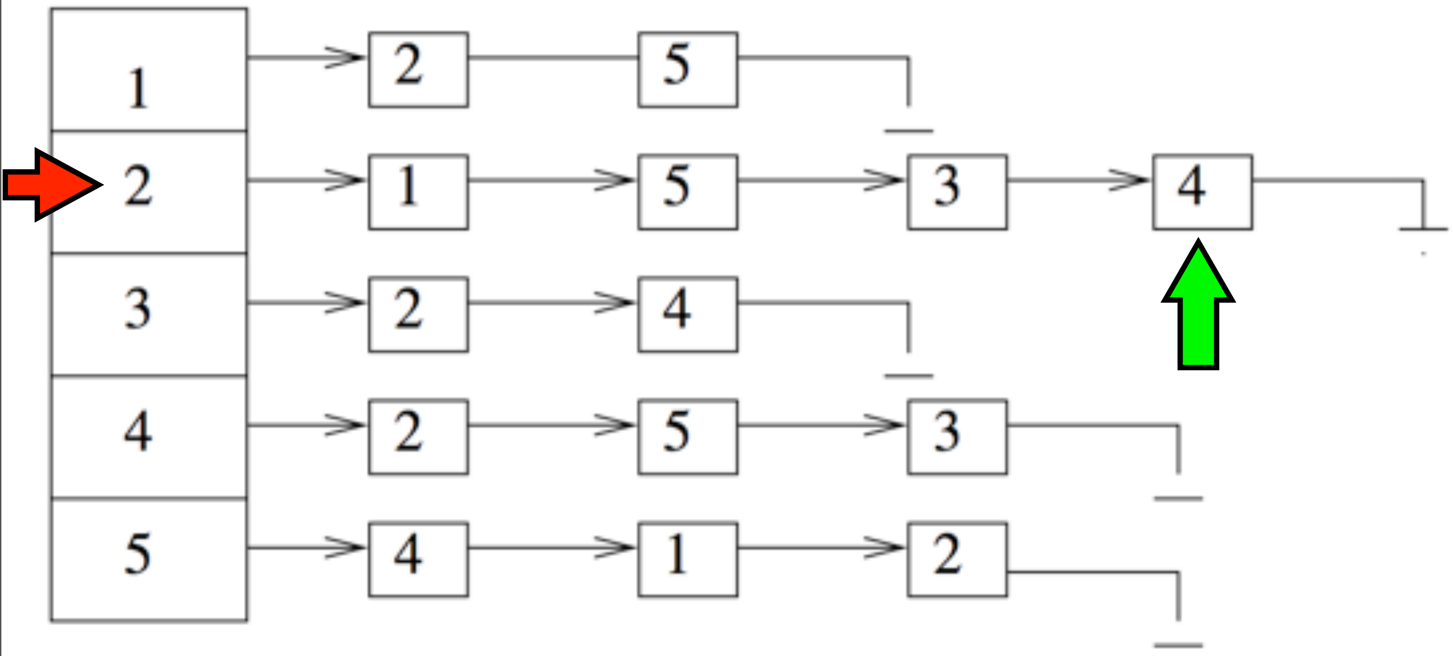
u = 2

v = 4

Saídas

Vértices: 1, 2

Arestas: (1,2) (1,5) (2,1) (2,5) (2,3) (2,4)




```
BFS(G, s)
for each vertex u ∈ V[G]−{s}
     state[u] = “undiscovered”
     p[u] = null, // sem pais
state[s] = “discovered”
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∈ Adj[u] do
          process edge (u,v)
          if state[v] = “undiscovered”
               state[v] = “discovered”
               p[v] = u
               enqueue[Q, v]
     state[u] = “processed”
```

Variáveis

	state	p
1	p	null
2	p	1
3	d	2
4	d	2
5	d	1

Q = {5,3,4}

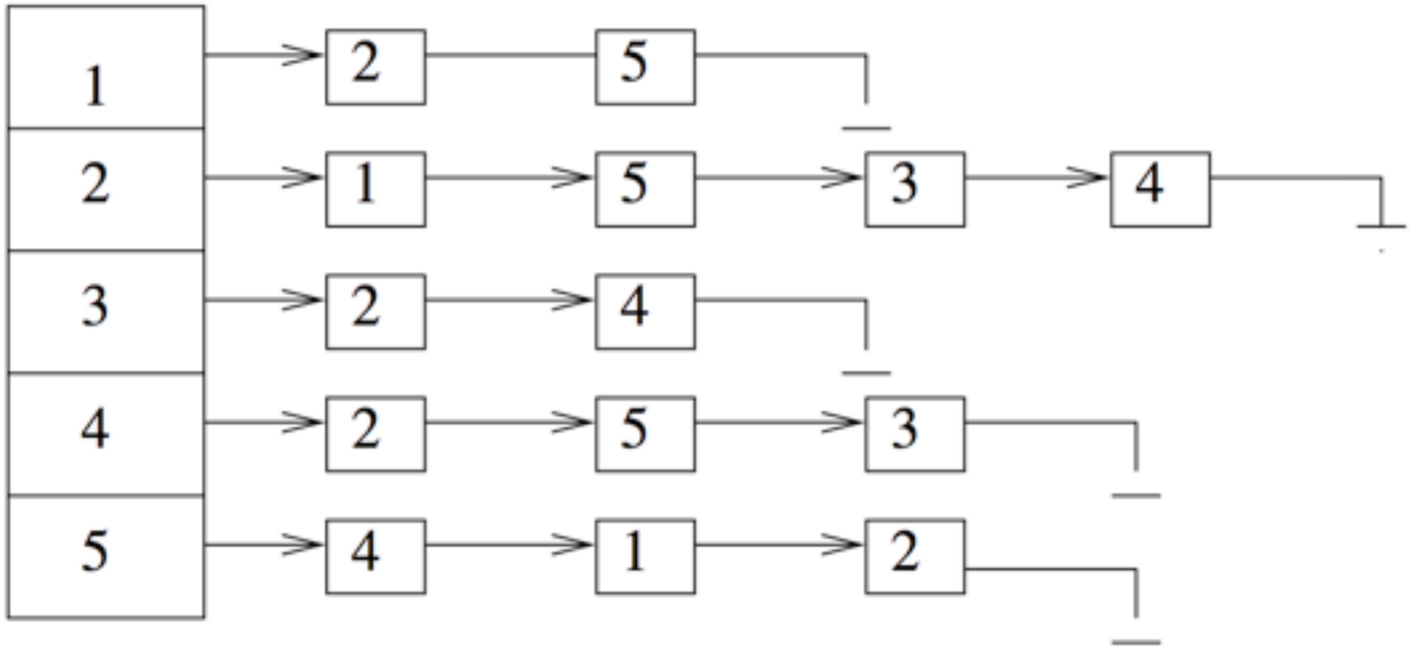
u = 2

v = 4

Saídas

Vértices: 1, 2

Arestas: (1,2) (1,5) (2,1) (2,5) (2,3) (2,4)




```
BFS(G, s)
for each vertex u ∊ V[G]−{s}
     state[u] = "undiscovered"
     p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∊ Adj[u] do
          process edge (u,v)
          if state[v] = "undiscovered"
               state[v] = "discovered"
               p[v] = u
               enqueue[Q, v]
     state[u] = "processed"
```

Variáveis

	state	p
1	p	null
2	p	1
3	d	2
4	d	2
5	d	1

Q = {5,3,4}

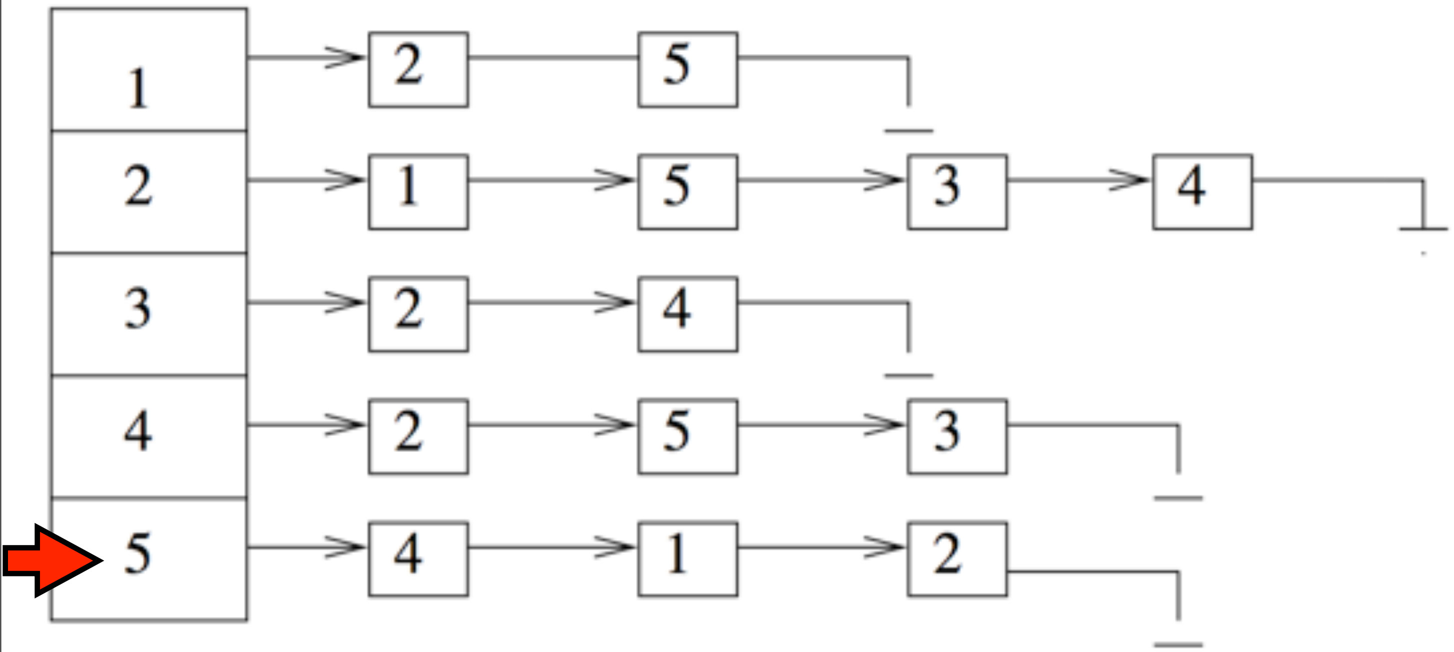
u = 2

v = 4

Saídas

Vértices: 1, 2

Arestas: (1,2) (1,5) (2,1) (2,5) (2,3) (2,4)




```
BFS(G, s)
for each vertex u ∈ V[G]−{s}
     state[u] = "undiscovered"
     p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∈ Adj[u] do
          process edge (u,v)
          if state[v] = "undiscovered"
               state[v] = "discovered"
               p[v] = u
               enqueue[Q, v]
     state[u] = "processed"
```

Variáveis

	state	p
1	p	null
2	p	1
3	d	2
4	d	2
5	d	1

Q = {3,4}

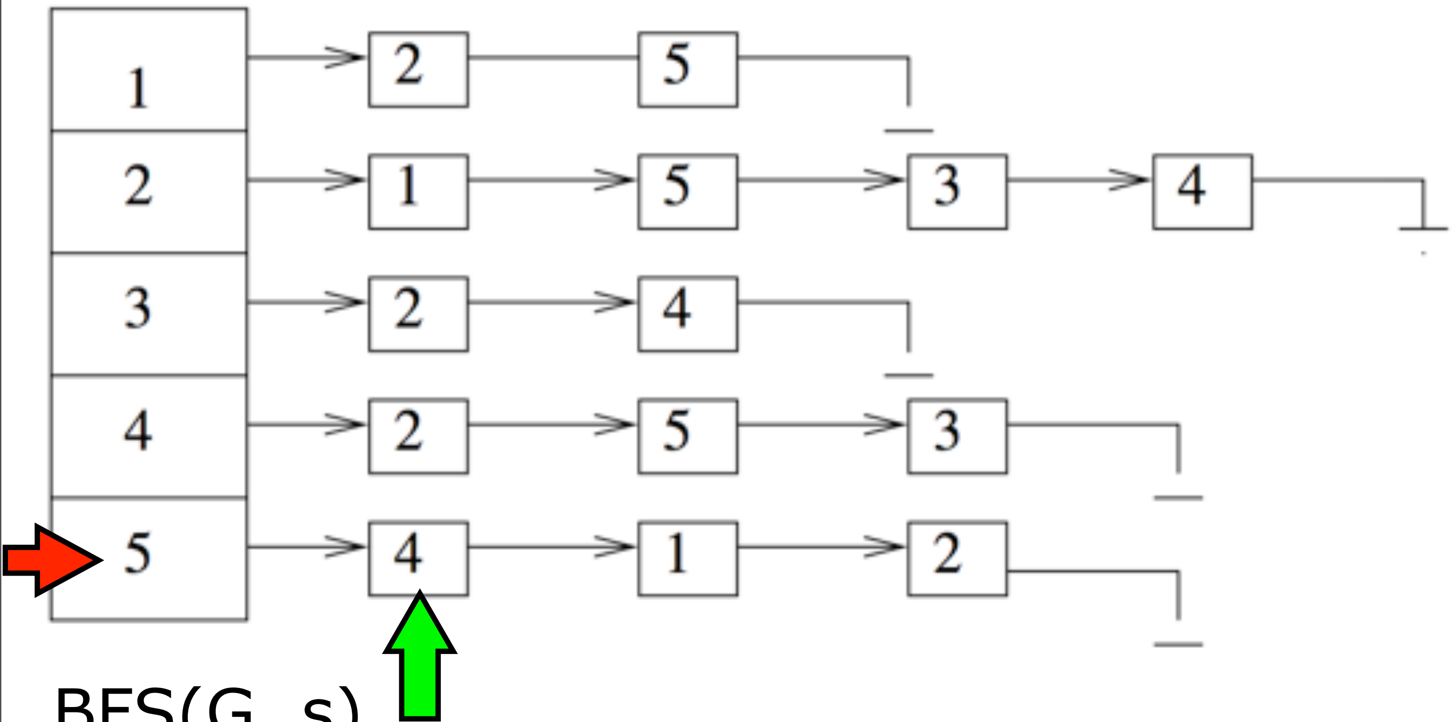
u = 5

v = 4

Saídas

Vértices: 1, 2, 5

Arestas: (1,2) (1,5) (2,1) (2,5) (2,3) (2,4)




```
BFS(G, s)
for each vertex u ∊ V[G]−{s}
     state[u] = "undiscovered"
     p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∊ Adj[u] do
          process edge (u,v)
          if state[v] = "undiscovered"
               state[v] = "discovered"
               p[v] = u
               enqueue[Q, v]
     state[u] = "processed"
```

Variáveis

	state	p
1	p	null
2	p	1
3	d	2
4	d	2
5	d	1

Q = {3,4}

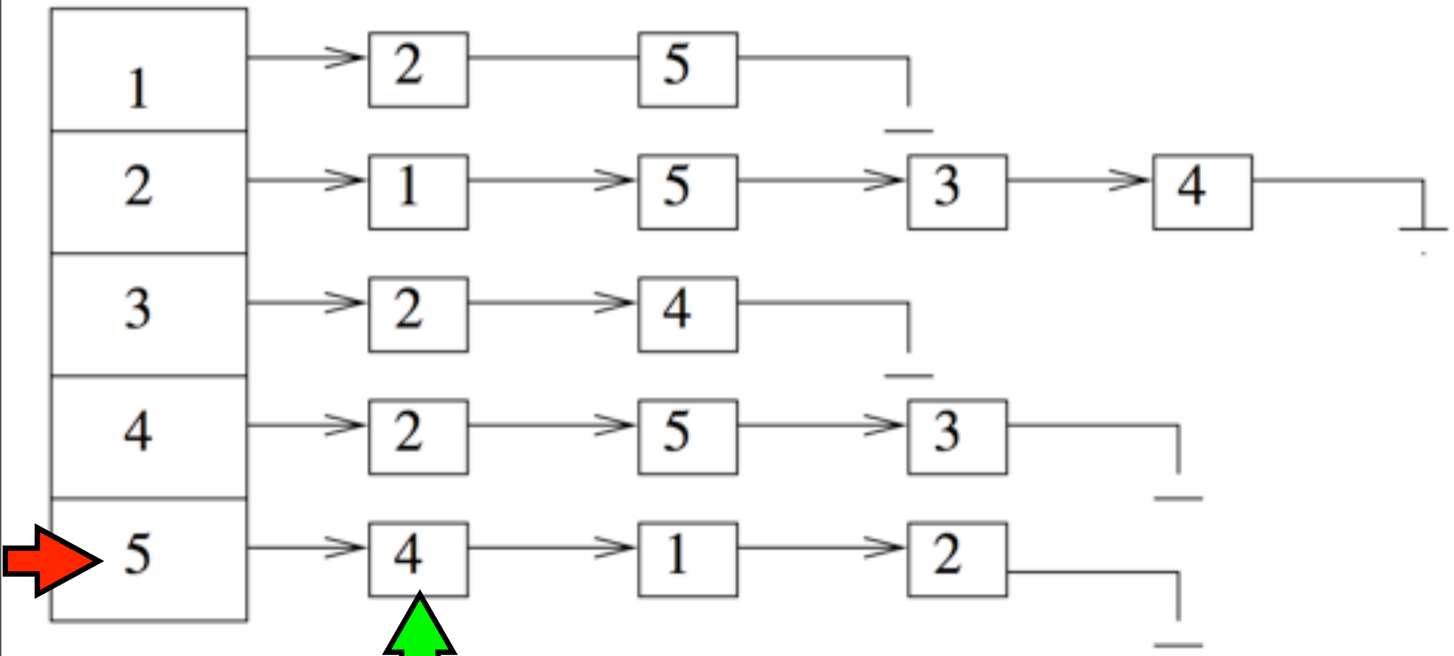
u = 5

v = 4

Saídas

Vértices: 1, 2, 5

Arestas: (1,2) (1,5) (2,1) (2,5) (2,3) (2,4)




```
BFS(G, s)
for each vertex u ∊ V[G]−{s}
     state[u] = "undiscovered"
     p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∊ Adj[u] do
          process edge (u,v)
          if state[v] = "undiscovered"
               state[v] = "discovered"
               p[v] = u
               enqueue[Q, v]
     state[u] = "processed"
```

Variáveis

	state	p
1	p	null
2	p	1
3	d	2
4	d	2
5	d	1

Q = {3,4}

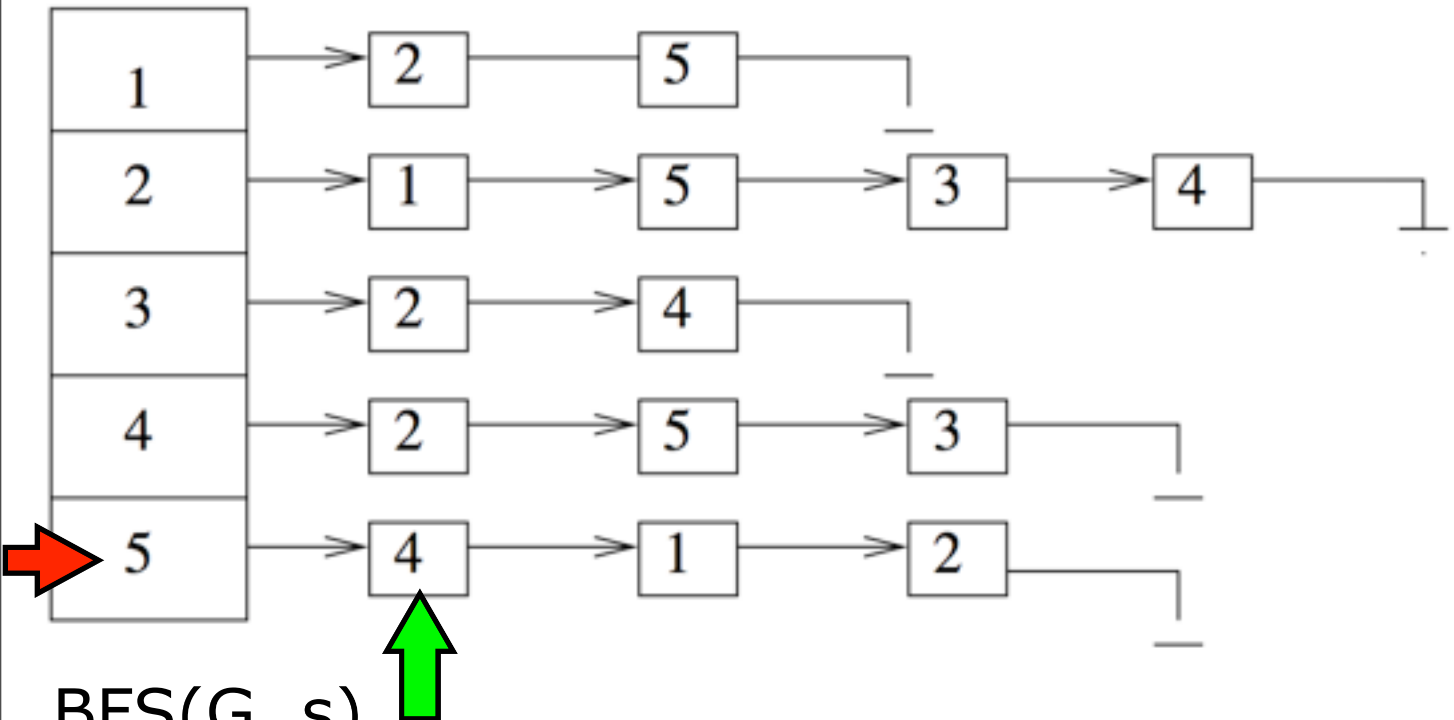
u = 5

v = 4

Saídas

Vértices: 1, 2, 5

Arestas: (1,2) (1,5) (2,1) (2,5) (2,3) (2,4) (5,4)




```
BFS(G, s)
for each vertex u ∊ V[G]−{s}
     state[u] = "undiscovered"
     p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∊ Adj[u] do
          process edge (u,v)
          if state[v] = "undiscovered"
               state[v] = "discovered"
               p[v] = u
               enqueue[Q, v]
     state[u] = "processed"
```

Variáveis

	state	p
1	p	null
2	p	1
3	d	2
4	d	2
5	d	1

Q = {3,4}

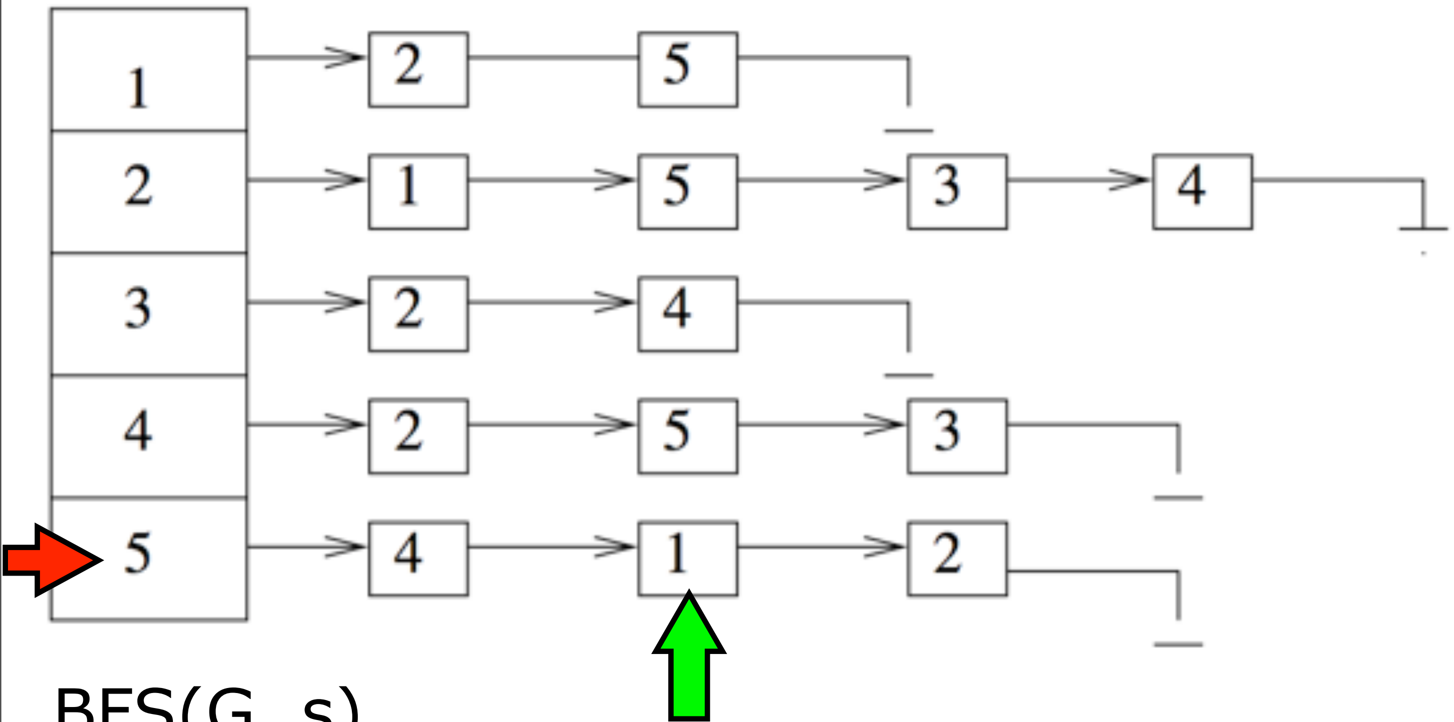
u = 5

v = 4

Saídas

Vértices: 1, 2, 5

Arestas: (1,2) (1,5) (2,1) (2,5) (2,3) (2,4) (5,4)




```
BFS(G, s)
for each vertex u ∊ V[G]−{s}
     state[u] = “undiscovered”
     p[u] = null, // sem pais
state[s] = “discovered”
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∊ Adj[u] do
          process edge (u,v)
          if state[v] = “undiscovered”
               state[v] = “discovered”
               p[v] = u
               enqueue[Q, v]
     state[u] = “processed”
```

Variáveis

	state	p
1	p	null
2	p	1
3	d	2
4	d	2
5	d	1

Q = {3,4}

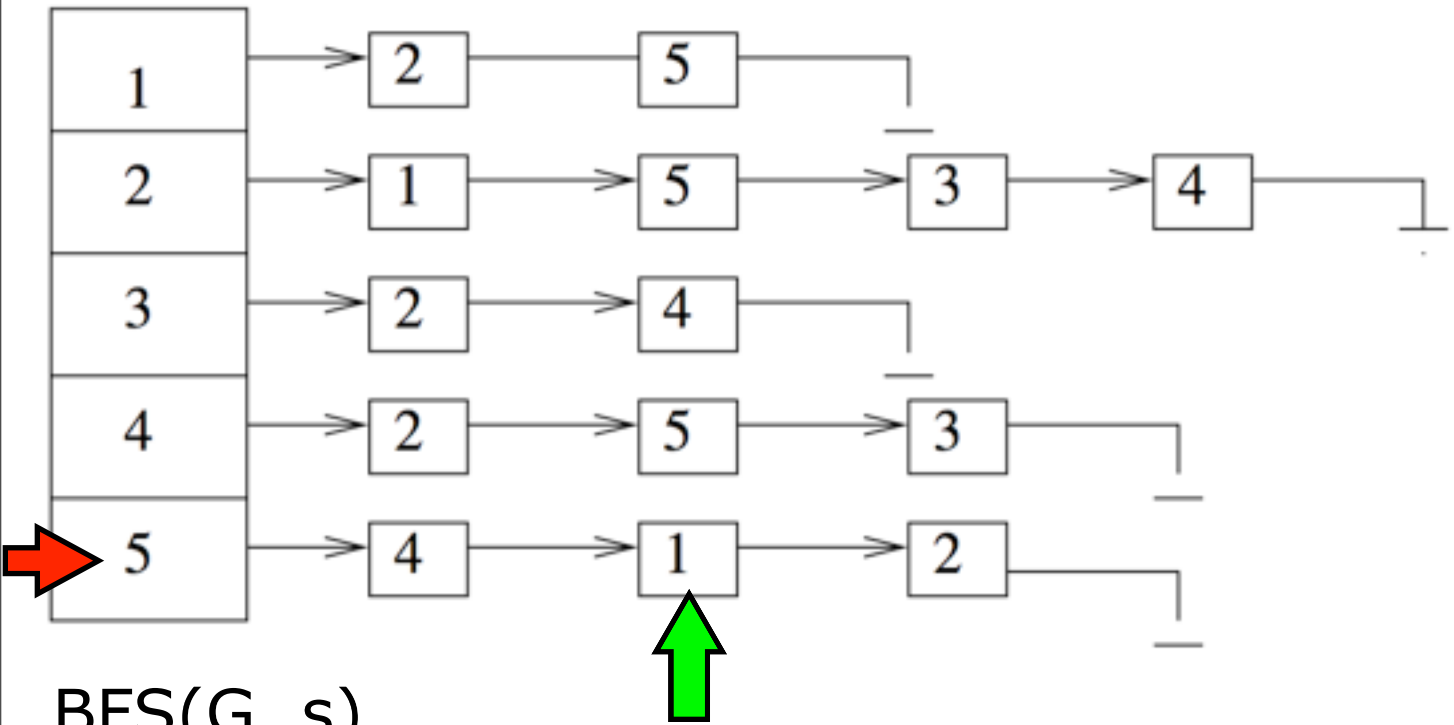
u = 5

v = 1

Saídas

Vértices: 1, 2, 5

Arestas: (1,2) (1,5) (2,1) (2,5) (2,3) (2,4) (5,4)




```
BFS(G, s)
for each vertex u ∊ V[G]−{s}
     state[u] = "undiscovered"
     p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∊ Adj[u] do
          process edge (u,v)
          if state[v] = "undiscovered"
               state[v] = "discovered"
               p[v] = u
               enqueue[Q, v]
     state[u] = "processed"
```

Variáveis

	state	p
1	p	null
2	p	1
3	d	2
4	d	2
5	d	1

Q = {3,4}

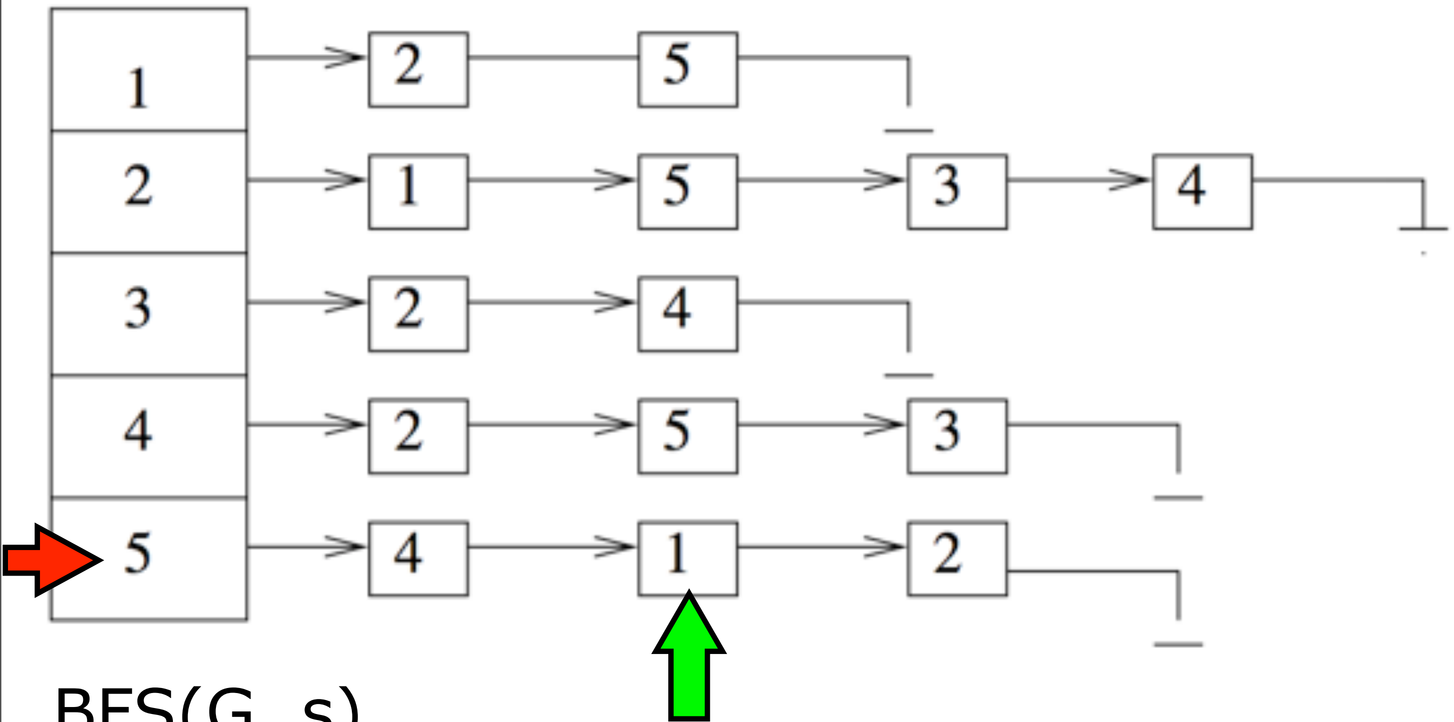
u = 5

v = 1

Saídas

Vértices: 1, 2, 5

Arestas: (1,2) (1,5) (2,1) (2,5) (2,3) (2,4) (5,4) (5,1)




```
BFS(G, s)
for each vertex u ∈ V[G]−{s}
    state[u] = "undiscovered"
    p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
    u = dequeue[Q]
    process vertex u
    for each v ∈ Adj[u] do
        process edge (u,v)
        if state[v] = "undiscovered"
            state[v] = "discovered"
            p[v] = u
            enqueue[Q, v]
    state[u] = "processed"
```

Variáveis

	state	p
1	p	null
2	p	1
3	d	2
4	d	2
5	d	1

Q = {3,4}

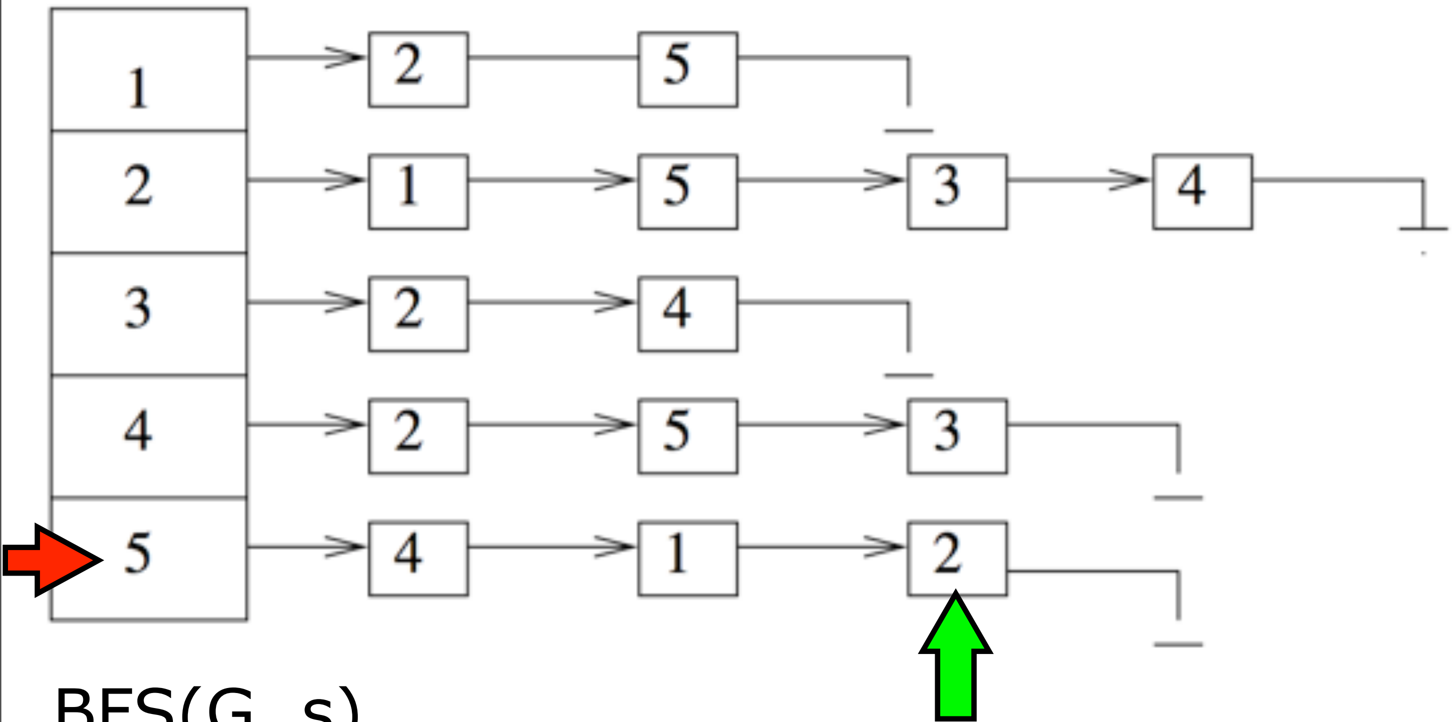
u = 5

v = 1

Saídas

Vértices: 1, 2, 5

Arestas: (1,2) (1,5) (2,1) (2,5) (2,3) (2,4) (5,4) (5,1)



BFS(G, s)
for each vertex u ∊ V[G]−{s}
    state[u] = "undiscovered"
    p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
    u = dequeue[Q]
    process vertex u
    for each v ∊ Adj[u] do
        process edge (u,v)
        if state[v] = "undiscovered"
            state[v] = "discovered"
            p[v] = u
            enqueue[Q, v]
    state[u] = "processed"

Variáveis

	state	p
1	p	null
2	p	1
3	d	2
4	d	2
5	d	1

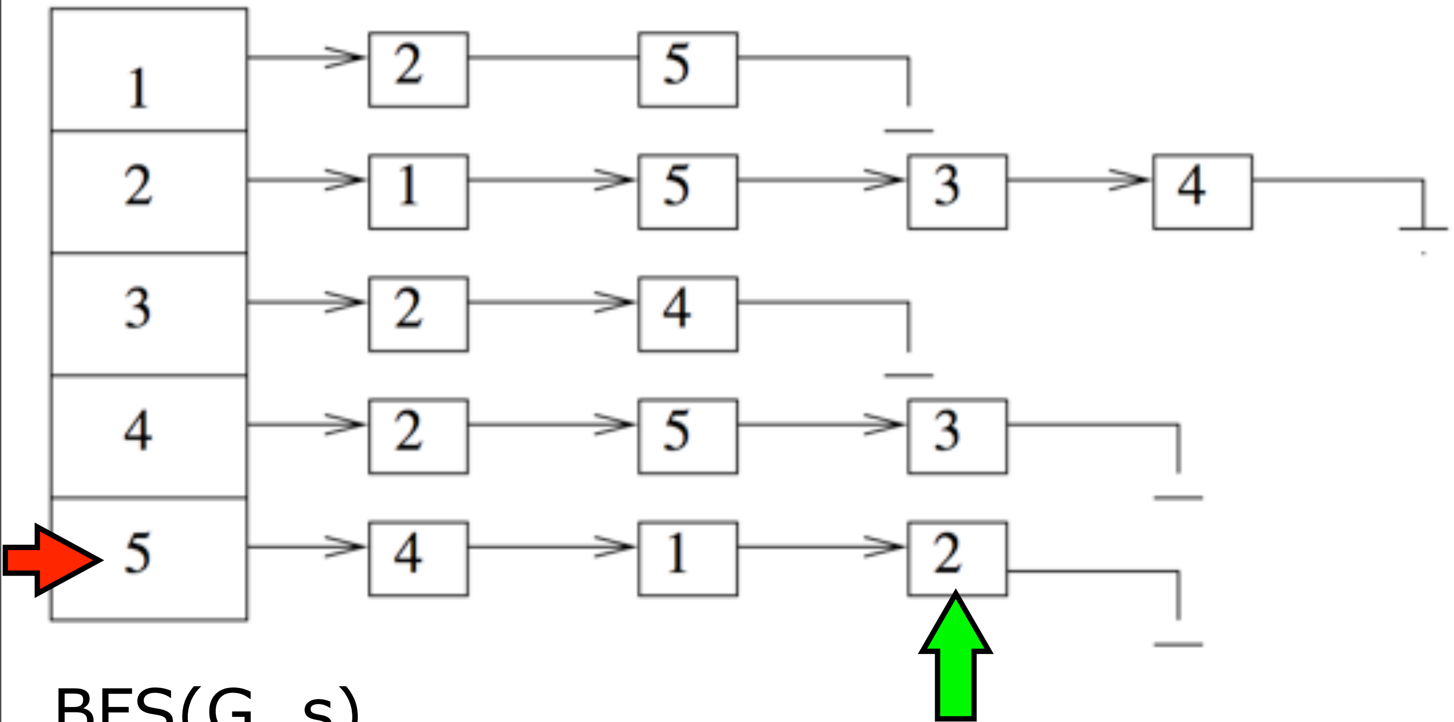
Q = {3,4}

u = 5

v = 2

Saídas

Vértices: 1, 2, 5
Arestas: (1,2) (1,5) (2,1) (2,5) (2,3) (2,4) (5,4) (5,1)




```
BFS(G, s)
for each vertex u ∈ V[G]−{s}
    state[u] = "undiscovered"
    p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
    u = dequeue[Q]
    process vertex u
    for each v ∈ Adj[u] do
        process edge (u,v)
        if state[v] = "undiscovered"
            state[v] = "discovered"
            p[v] = u
            enqueue[Q, v]
    state[u] = "processed"
```

Variáveis

	state	p
1	p	null
2	p	1
3	d	2
4	d	2
5	d	1

Q = {3,4}

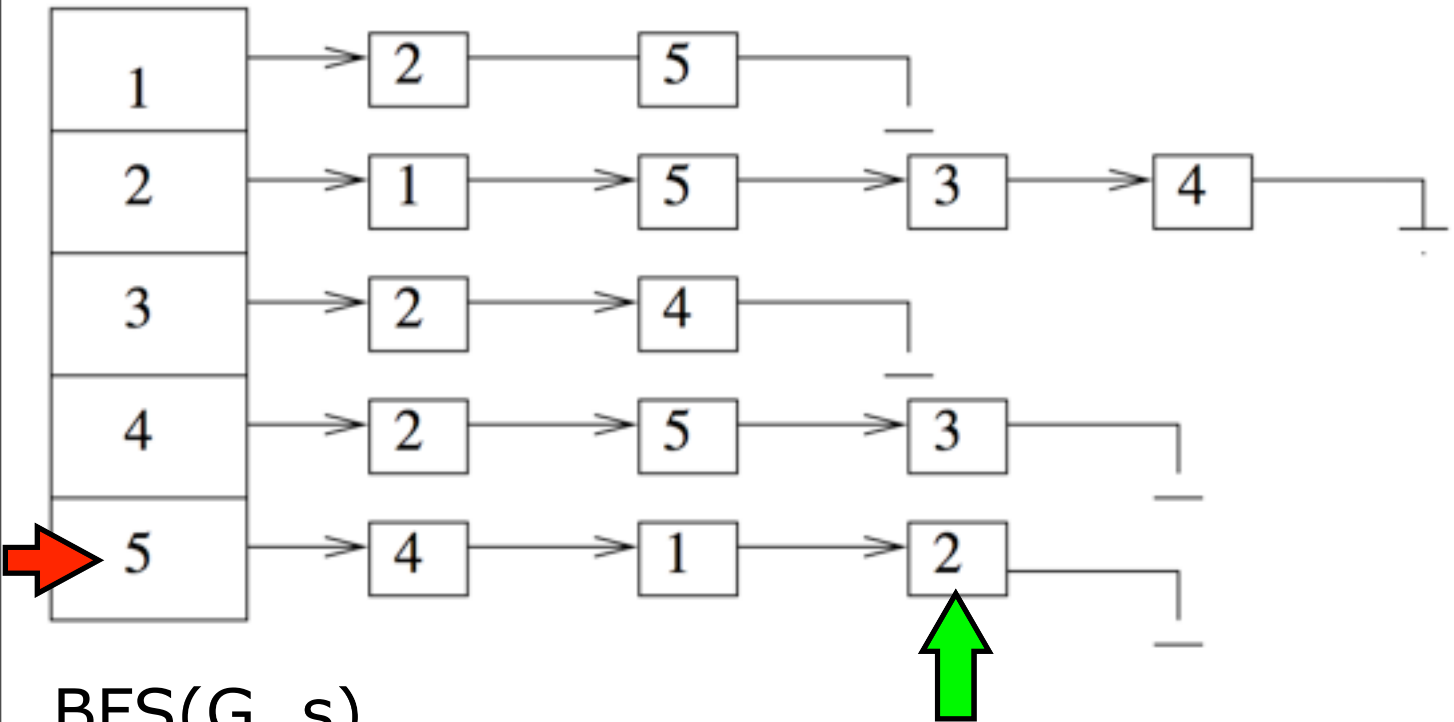
u = 5

v = 2

Saídas

Vértices: 1, 2, 5

Arestas: (1,2) (1,5) (2,1) (2,5) (2,3) (2,4) (5,4) (5,1) (5,2)




```
BFS(G, s)
for each vertex u ∈ V[G]−{s}
     state[u] = "undiscovered"
     p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∈ Adj[u] do
          process edge (u,v)
          if state[v] = "undiscovered"
               state[v] = "discovered"
               p[v] = u
               enqueue[Q, v]
     state[u] = "processed"
```

Variáveis

	state	p
1	p	null
2	p	1
3	d	2
4	d	2
5	d	1

Q = {3,4}

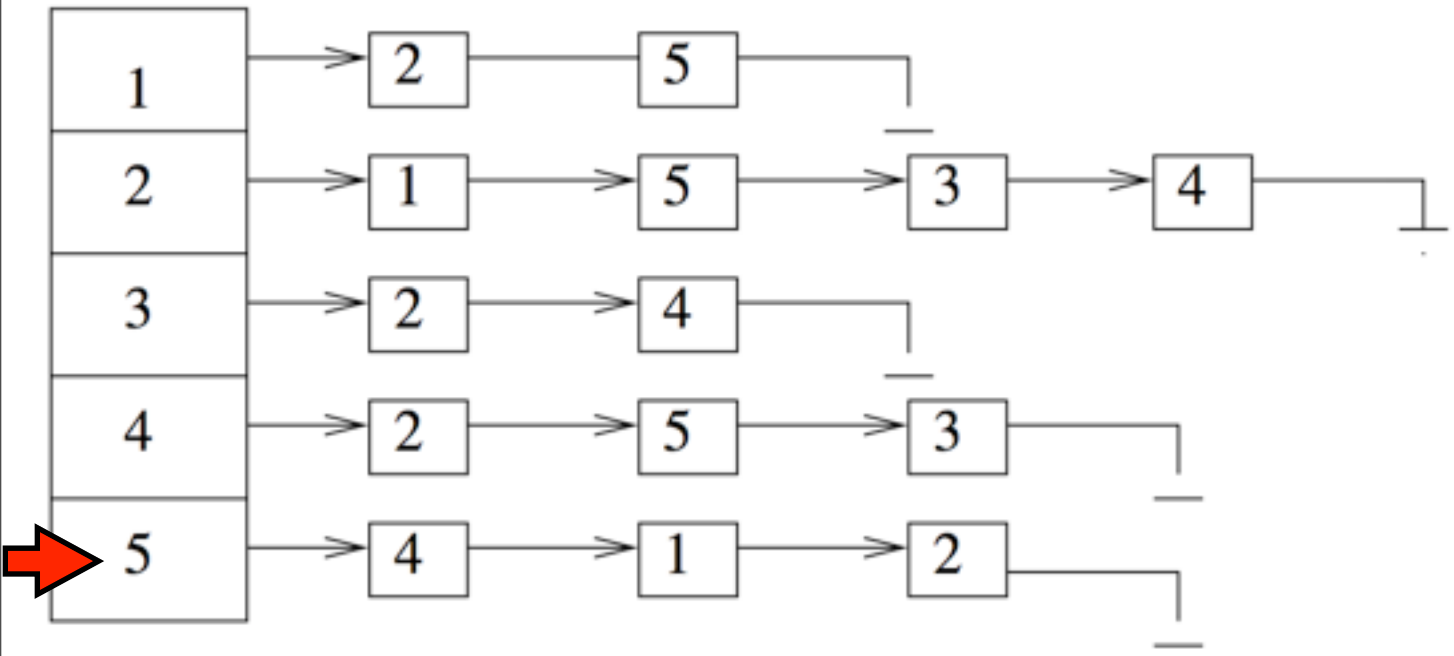
u = 5

v = 2

Saídas

Vértices: 1, 2, 5

Arestas: (1,2) (1,5) (2,1) (2,5) (2,3) (2,4) (5,4) (5,1) (5,2)




```
BFS(G, s)
for each vertex u ∈ V[G]−{s}
     state[u] = "undiscovered"
     p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∈ Adj[u] do
          process edge (u,v)
          if state[v] = "undiscovered"
               state[v] = "discovered"
               p[v] = u
               enqueue[Q, v]
     state[u] = "processed"
```

Variáveis

	state	p
1	p	null
2	p	1
3	d	2
4	d	2
5	p	1

Q = {3,4}

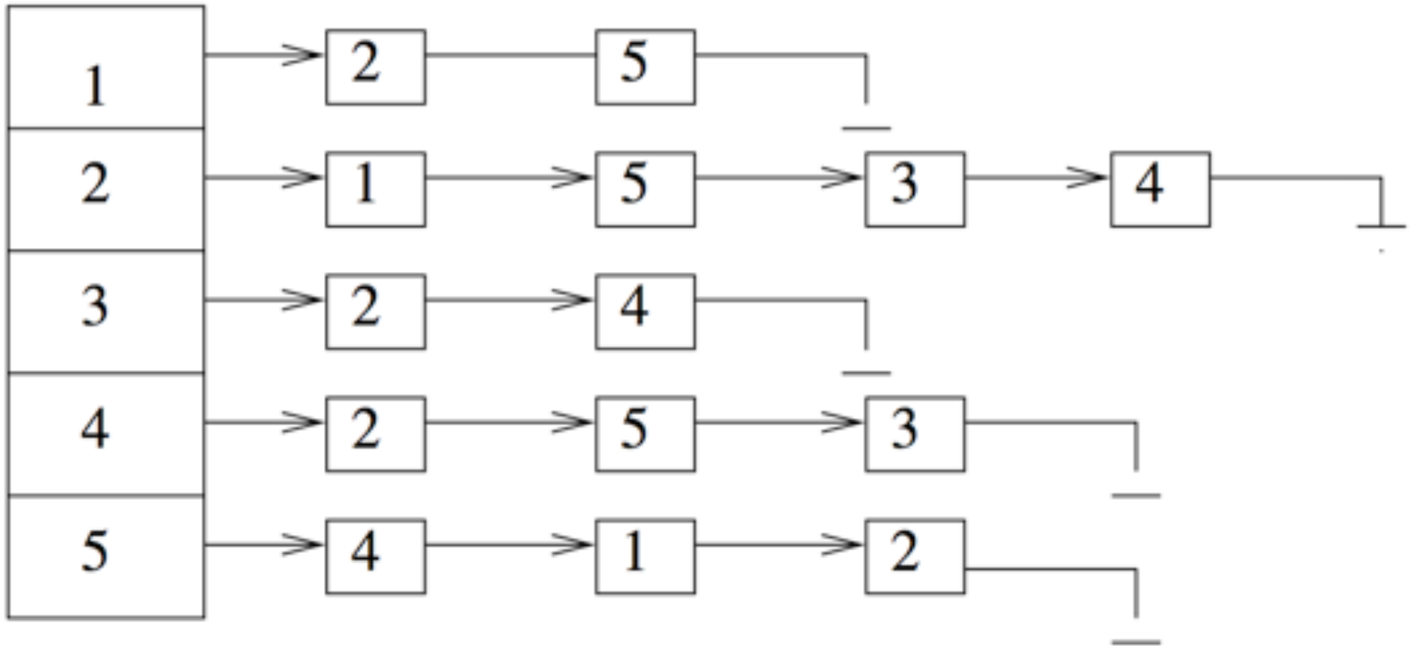
u = 5

v = 2

Saídas

Vértices: 1, 2, 5

Arestas: (1,2) (1,5) (2,1) (2,5) (2,3) (2,4) (5,4) (5,1) (5,2)




```
BFS(G, s)
for each vertex u ∊ V[G]−{s}
     state[u] = "undiscovered"
     p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∊ Adj[u] do
          process edge (u,v)
          if state[v] = "undiscovered"
               state[v] = "discovered"
               p[v] = u
               enqueue[Q, v]
     state[u] = "processed"
```

Variáveis

	state	p
1	p	null
2	p	1
3	d	2
4	d	2
5	p	1

Q = {3,4}

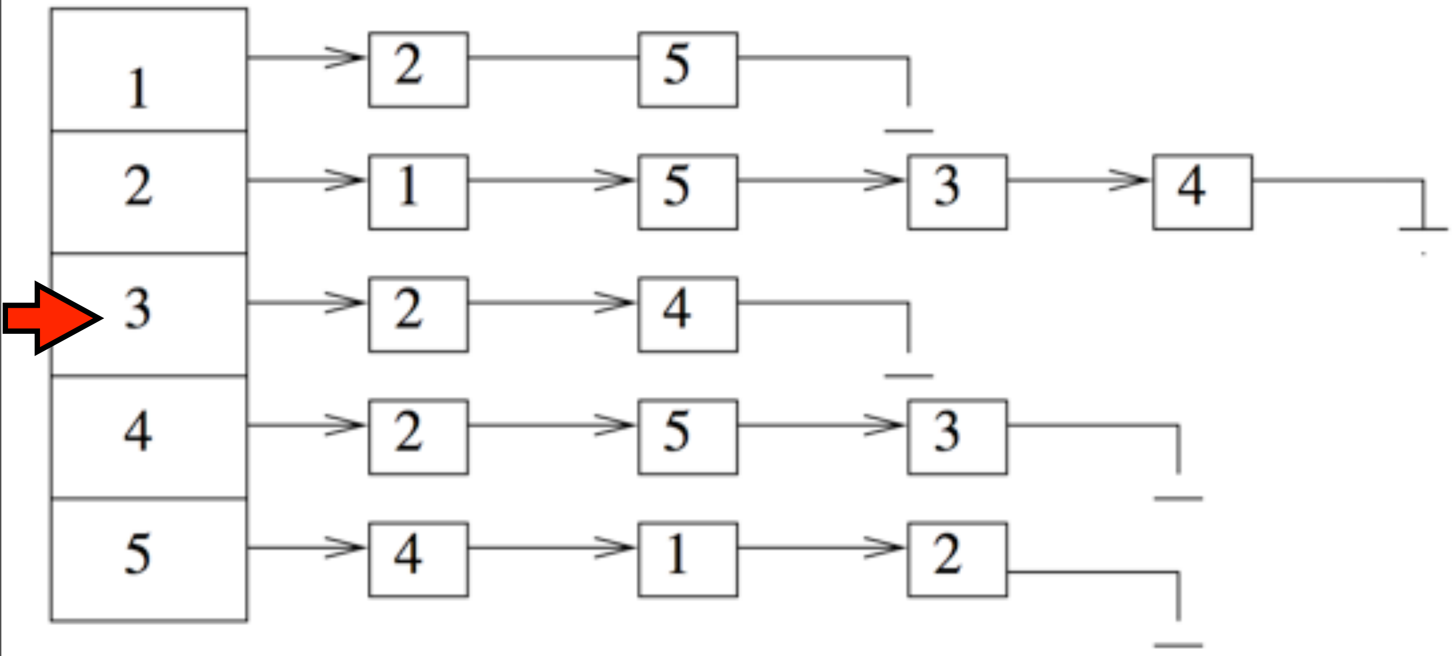
u = 5

v = 2

Saídas

Vértices: 1, 2, 5

Arestas: (1,2) (1,5) (2,1) (2,5) (2,3) (2,4) (5,4) (5,1) (5,2)




```
BFS(G, s)
for each vertex u ∊ V[G]−{s}
     state[u] = "undiscovered"
     p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∊ Adj[u] do
          process edge (u,v)
          if state[v] = "undiscovered"
               state[v] = "discovered"
               p[v] = u
               enqueue[Q, v]
     state[u] = "processed"
```

Variáveis

	state	p
1	p	null
2	p	1
3	d	2
4	d	2
5	p	1

Q = {4}

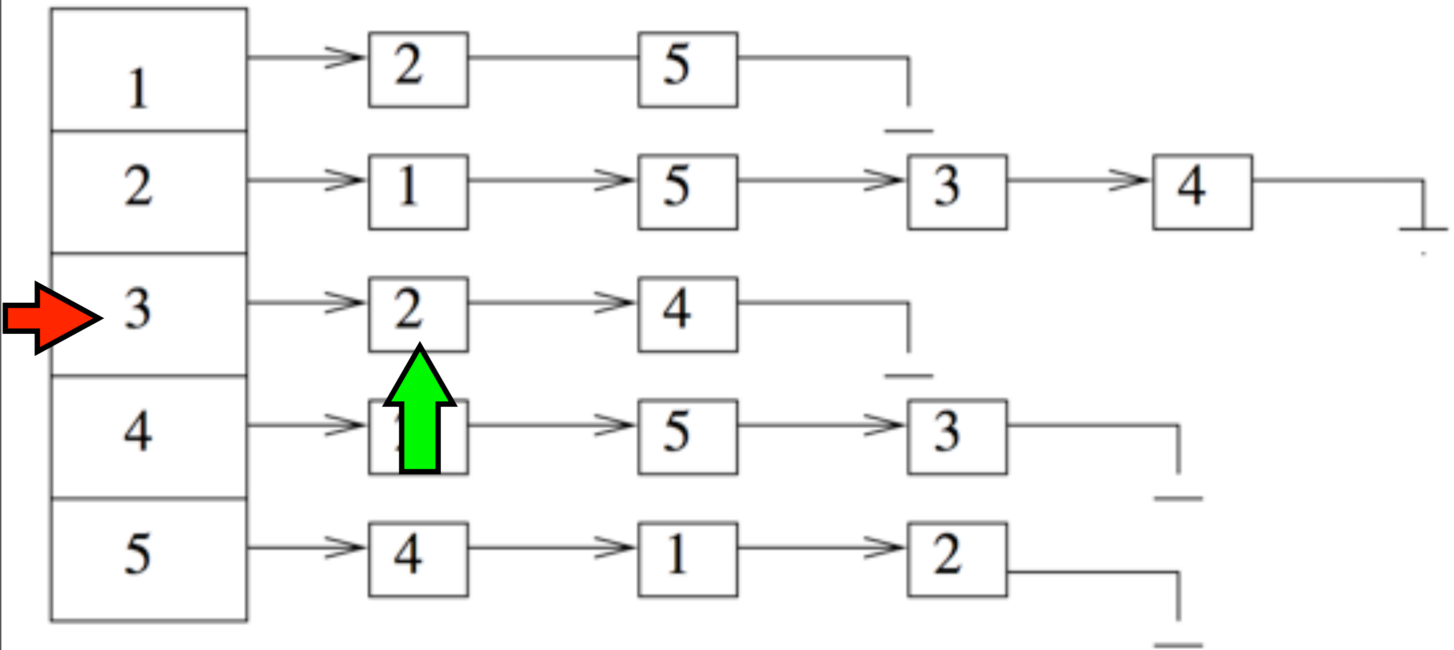
u = 3

v = 2

Saídas

Vértices: 1, 2, 5, 3

Arestas: (1,2) (1,5) (2,1) (2,5) (2,3) (2,4) (5,4) (5,1) (5,2)




```
BFS(G, s)
for each vertex u ∊ V[G]−{s}
     state[u] = "undiscovered"
     p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∊ Adj[u] do
          process edge (u,v)
          if state[v] = "undiscovered"
               state[v] = "discovered"
               p[v] = u
               enqueue[Q, v]
     state[u] = "processed"
```

Variáveis

	state	p
1	p	null
2	p	1
3	d	2
4	d	2
5	p	1

Q = {4}

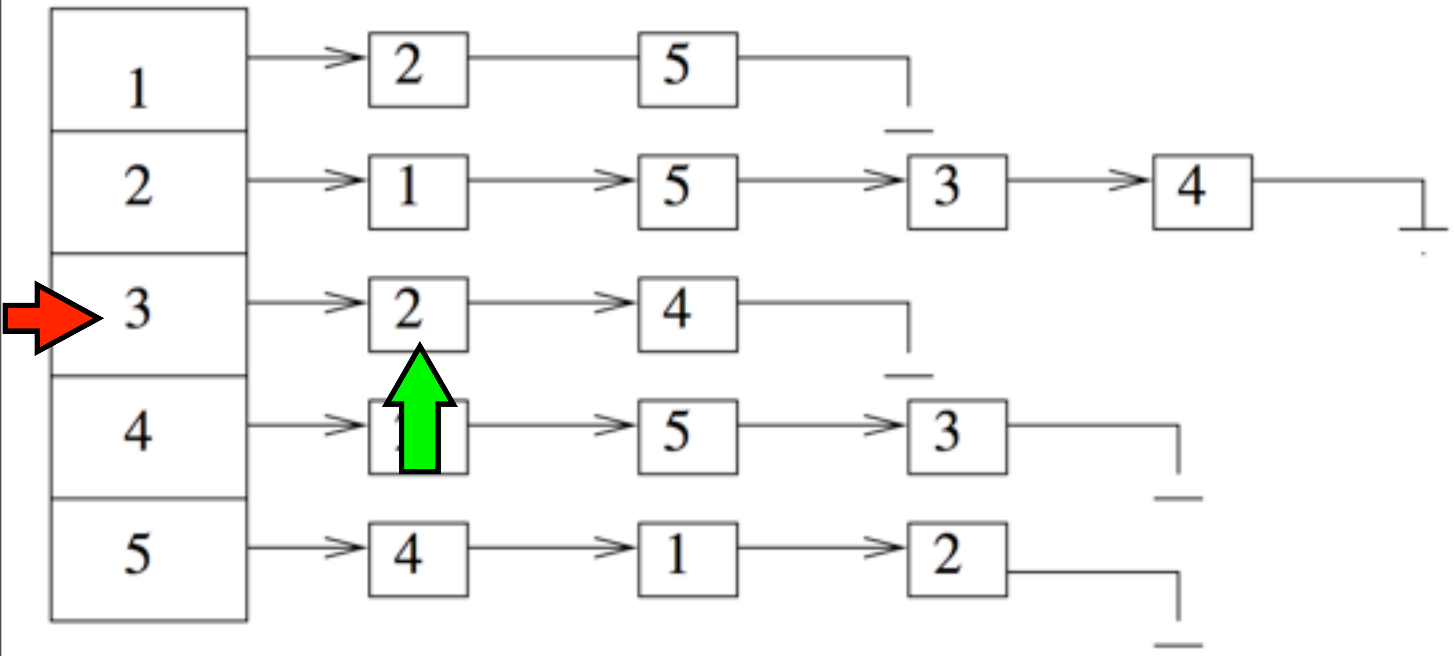
u = 3

v = 2

Saídas

Vértices: 1, 2, 5, 3

Arestas: (1,2) (1,5) (2,1) (2,5) (2,3) (2,4) (5,4) (5,1) (5,2)




```
BFS(G, s)
for each vertex u ∊ V[G]−{s}
     state[u] = “undiscovered”
     p[u] = null, // sem pais
state[s] = “discovered”
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∊ Adj[u] do
          process edge (u,v)
          if state[v] = “undiscovered”
               state[v] = “discovered”
               p[v] = u
               enqueue[Q, v]
     state[u] = “processed”
```

Variáveis

	state	p
1	p	null
2	p	1
3	d	2
4	d	2
5	p	1

Q = {4}

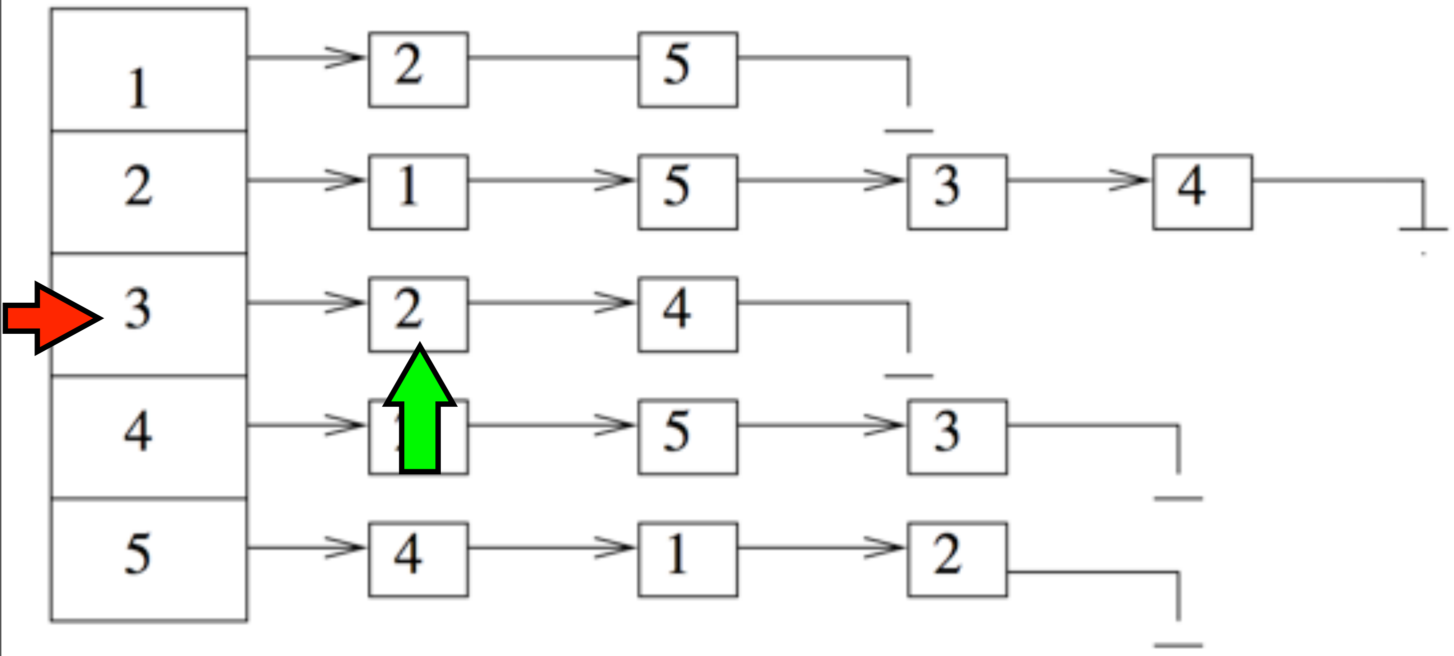
u = 3

v = 2

Saídas

Vértices: 1, 2, 5, 3

Arestas: (1,2) (1,5) (2,1) (2,5) (2,3) (2,4) (5,4) (5,1) (5,2) (3,2)




```
BFS(G, s)
for each vertex u ∈ V[G]−{s}
     state[u] = "undiscovered"
     p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∈ Adj[u] do
          process edge (u,v)
          if state[v] = "undiscovered"
               state[v] = "discovered"
               p[v] = u
               enqueue[Q, v]
     state[u] = "processed"
```

Variáveis

	state	p
1	p	null
2	p	1
3	d	2
4	d	2
5	p	1

Q = {4}

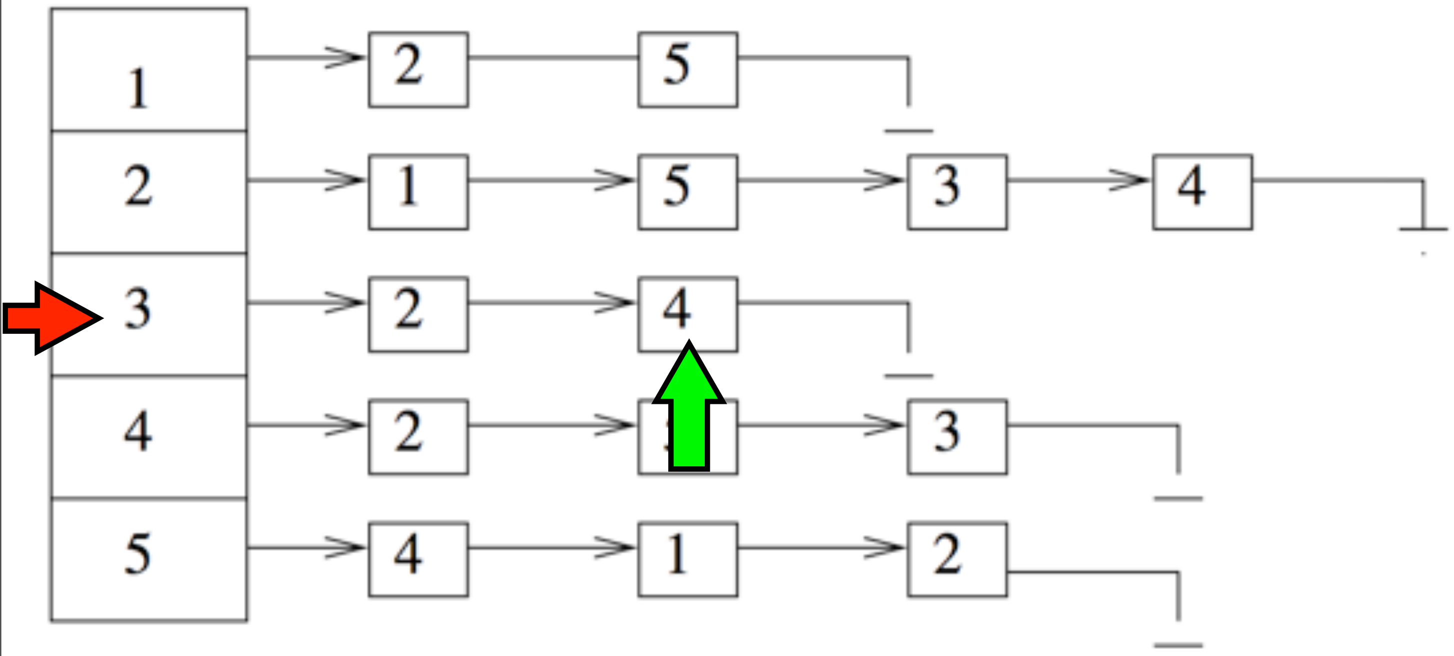
u = 3

v = 2

Saídas

Vértices: 1, 2, 5, 3

Arestas: (1,2) (1,5) (2,1) (2,5) (2,3) (2,4) (5,4) (5,1) (5,2) (3,2)




```
BFS(G, s)
for each vertex u ∊ V[G]−{s}
     state[u] = "undiscovered"
     p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∊ Adj[u] do
          process edge (u,v)
          if state[v] = "undiscovered"
               state[v] = "discovered"
               p[v] = u
               enqueue[Q, v]
     state[u] = "processed"
```

Variáveis

	state	p
1	p	null
2	p	1
3	d	2
4	d	2
5	p	1

Q = {4}

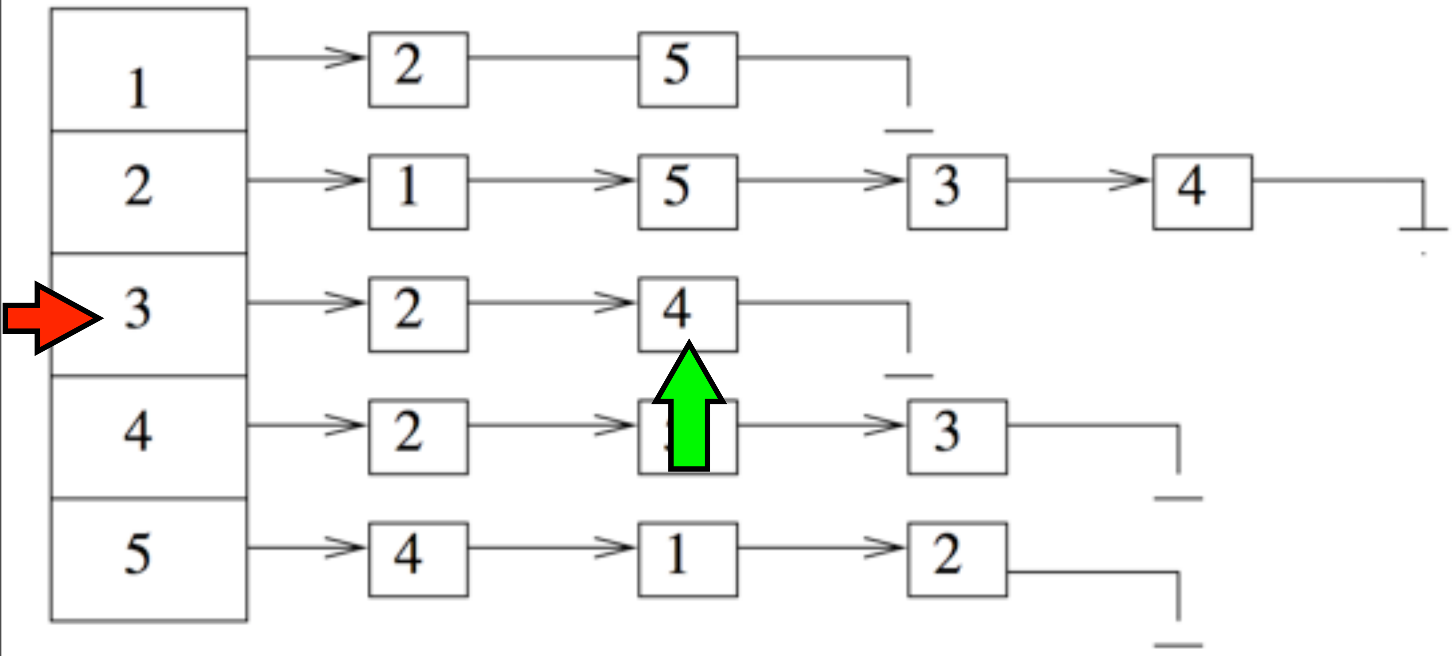
u = 3

v = 4

Saídas

Vértices: 1, 2, 5, 3

Arestas: (1,2) (1,5) (2,1) (2,5) (2,3) (2,4) (5,4) (5,1) (5,2) (3,2)




```
BFS(G, s)
for each vertex u ∈ V[G]−{s}
     state[u] = "undiscovered"
     p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∈ Adj[u] do
          process edge (u,v)
          if state[v] = "undiscovered"
               state[v] = "discovered"
               p[v] = u
               enqueue[Q, v]
     state[u] = "processed"
```

Variáveis

	state	p
1	p	null
2	p	1
3	d	2
4	d	2
5	p	1

Q = {4}

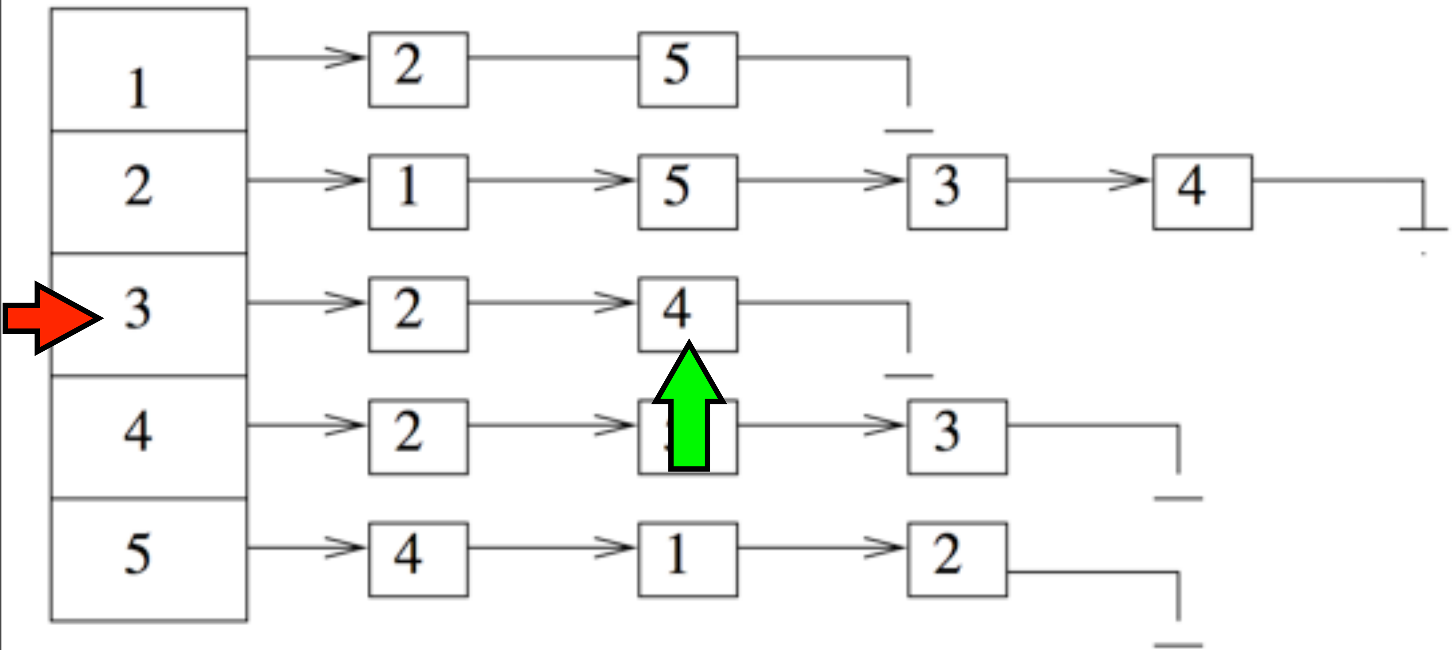
u = 3

v = 4

Saídas

Vértices: 1, 2, 5, 3

Arestas: (1,2) (1,5) (2,1) (2,5) (2,3) (2,4) (5,4) (5,1) (5,2) (3,2) (3,4)




```
BFS(G, s)
for each vertex u ∈ V[G]−{s}
     state[u] = "undiscovered"
     p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∈ Adj[u] do
          process edge (u,v)
          if state[v] = "undiscovered"
               state[v] = "discovered"
               p[v] = u
               enqueue[Q, v]
     state[u] = "processed"
```

Variáveis

	state	p
1	p	null
2	p	1
3	d	2
4	d	2
5	p	1

Q = {4}

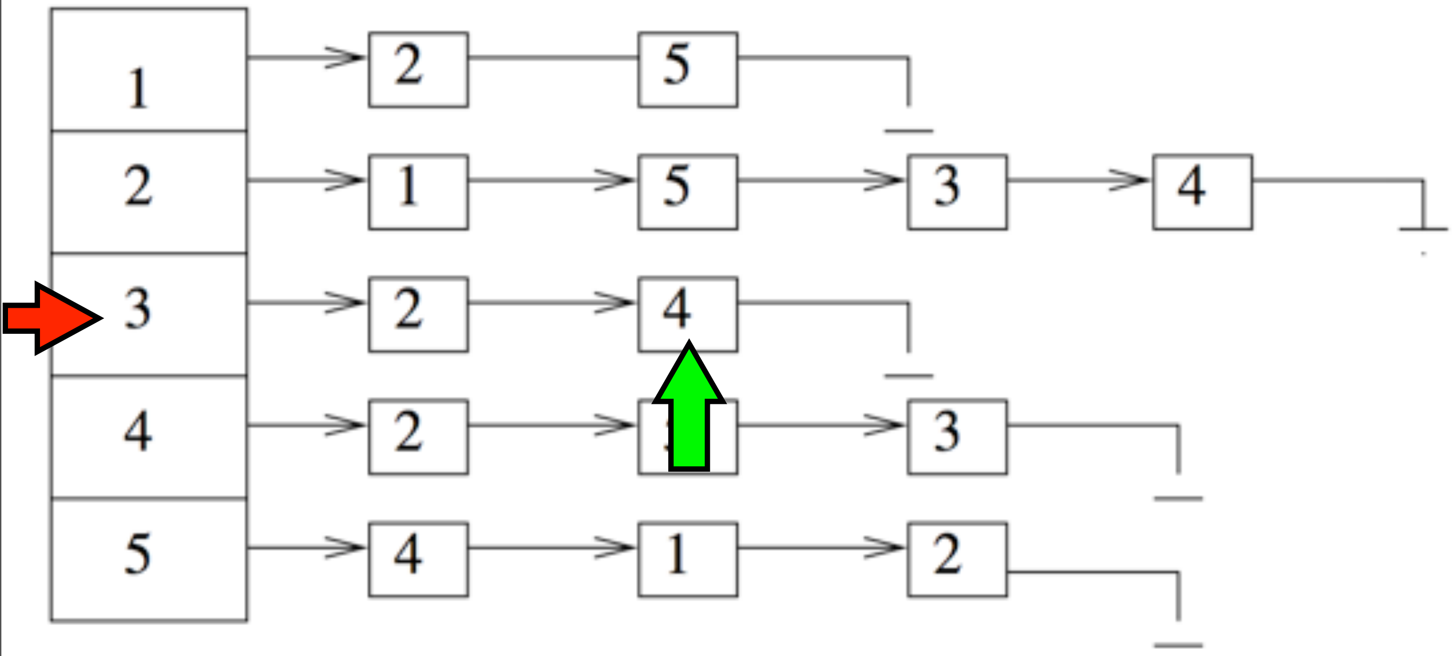
u = 3

v = 4

Saídas

Vértices: 1, 2, 5, 3

Arestas: (1,2) (1,5) (2,1) (2,5) (2,3) (2,4) (5,4) (5,1) (5,2) (3,2) (3,4)




```
BFS(G, s)
for each vertex u ∊ V[G]−{s}
     state[u] = “undiscovered”
     p[u] = null, // sem pais
state[s] = “discovered”
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∊ Adj[u] do
          process edge (u,v)
          if state[v] = “undiscovered”
               state[v] = “discovered”
               p[v] = u
               enqueue[Q, v]
     state[u] = “processed”
```

Variáveis

	state	p
1	p	null
2	p	1
3	p	2
4	d	2
5	p	1

Q = {4}

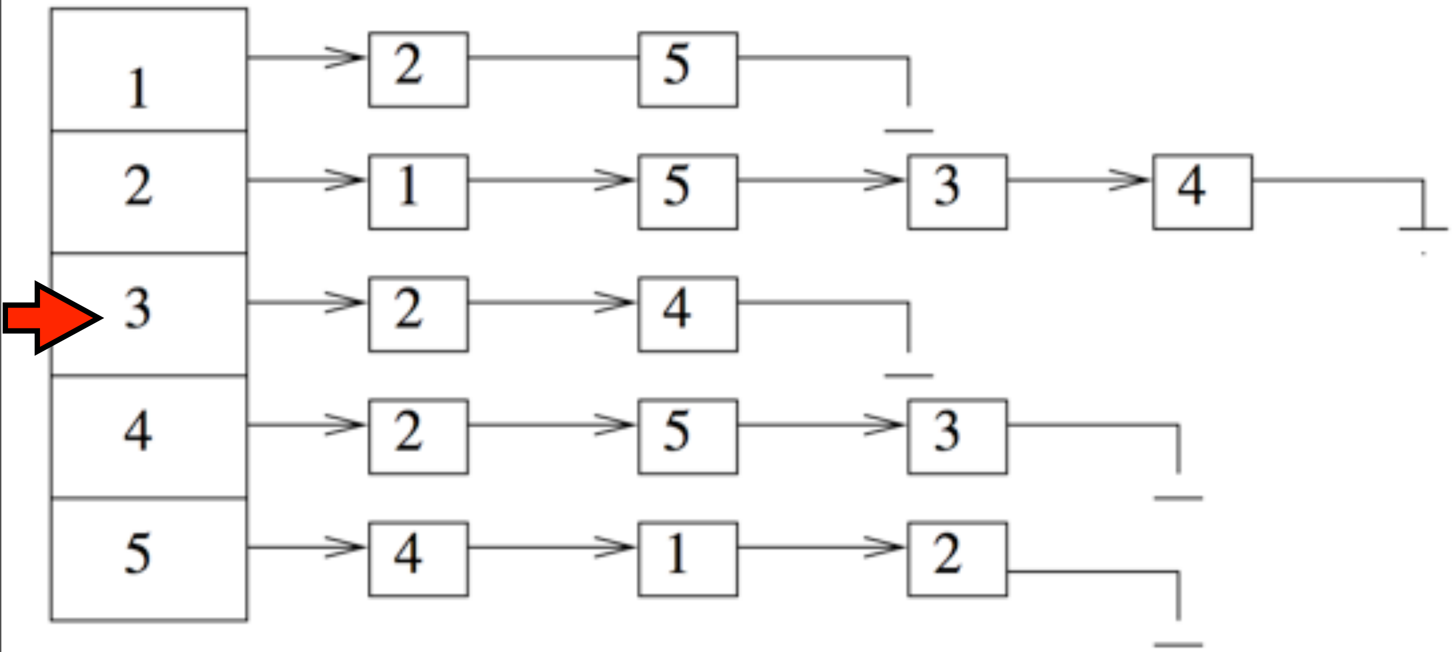
u = 3

v = 4

Saídas

Vértices: 1, 2, 5, 3

Arestas: (1,2) (1,5) (2,1) (2,5) (2,3) (2,4) (5,4) (5,1) (5,2) (3,2) (3,4)




```
BFS(G, s)
for each vertex u ∊ V[G]−{s}
     state[u] = "undiscovered"
     p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∊ Adj[u] do
          process edge (u,v)
          if state[v] = "undiscovered"
               state[v] = "discovered"
               p[v] = u
               enqueue[Q, v]
     state[u] = "processed"
```

Variáveis

	state	p
1	p	null
2	p	1
3	p	2
4	d	2
5	p	1

Q = {4}

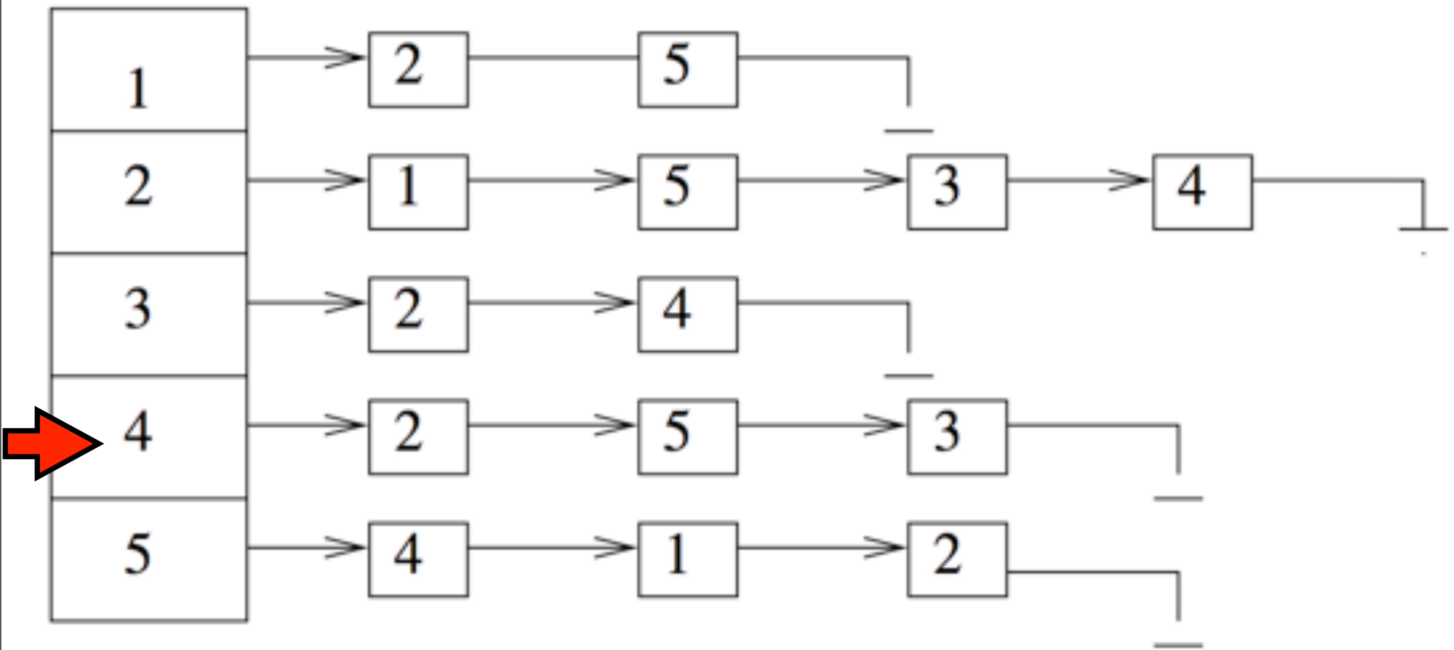
u = 3

v = 4

Saídas

Vértices: 1, 2, 5, 3

Arestas: (1,2) (1,5) (2,1) (2,5) (2,3) (2,4) (5,4) (5,1) (5,2) (3,2) (3,4)




```
BFS(G, s)
for each vertex u ∈ V[G]−{s}
    state[u] = "undiscovered"
    p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
    u = dequeue[Q]
    process vertex u
    for each v ∈ Adj[u] do
        process edge (u,v)
        if state[v] = "undiscovered"
            state[v] = "discovered"
            p[v] = u
            enqueue[Q, v]
    state[u] = "processed"
```

Variáveis

	state	p
1	p	null
2	p	1
3	p	2
4	d	2
5	p	1

Q = ∅

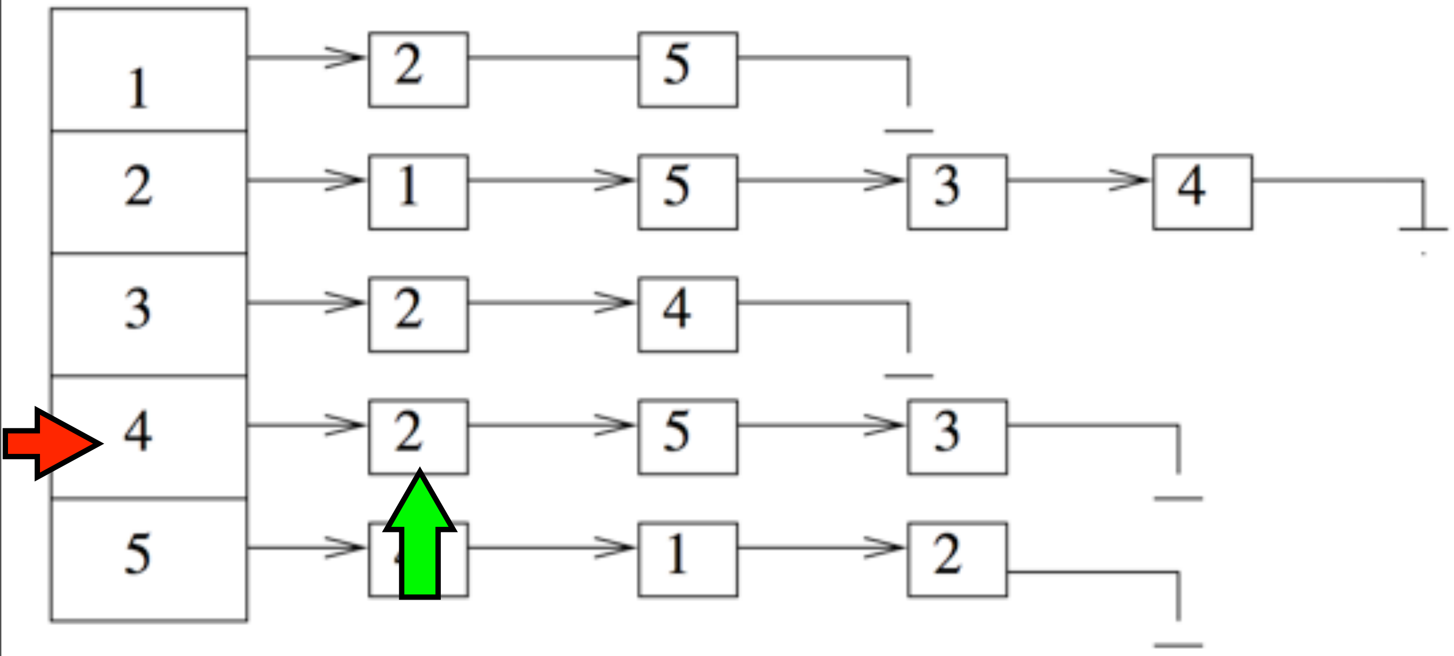
u = 4

v = 4

Saídas

Vértices: 1, 2, 5, 3, 4

Arestas: (1,2) (1,5) (2,1) (2,5) (2,3) (2,4) (5,4) (5,1) (5,2) (3,2) (3,4)



BFS(G, s)
for each vertex u ∈ V[G]−{s}
    state[u] = “undiscovered”
    p[u] = null, // sem pais
state[s] = “discovered”
p[s] = null
Q = {s}
while (Q != ∅)
    u = dequeue[Q]
    process vertex u
    for each v ∈ Adj[u] do
        process edge (u,v)
        if state[v] = “undiscovered”
            state[v] = “discovered”
            p[v] = u
            enqueue[Q, v]
    state[u] = “processed”

Variáveis

	state	p
1	p	null
2	p	1
3	p	2
4	d	2
5	p	1

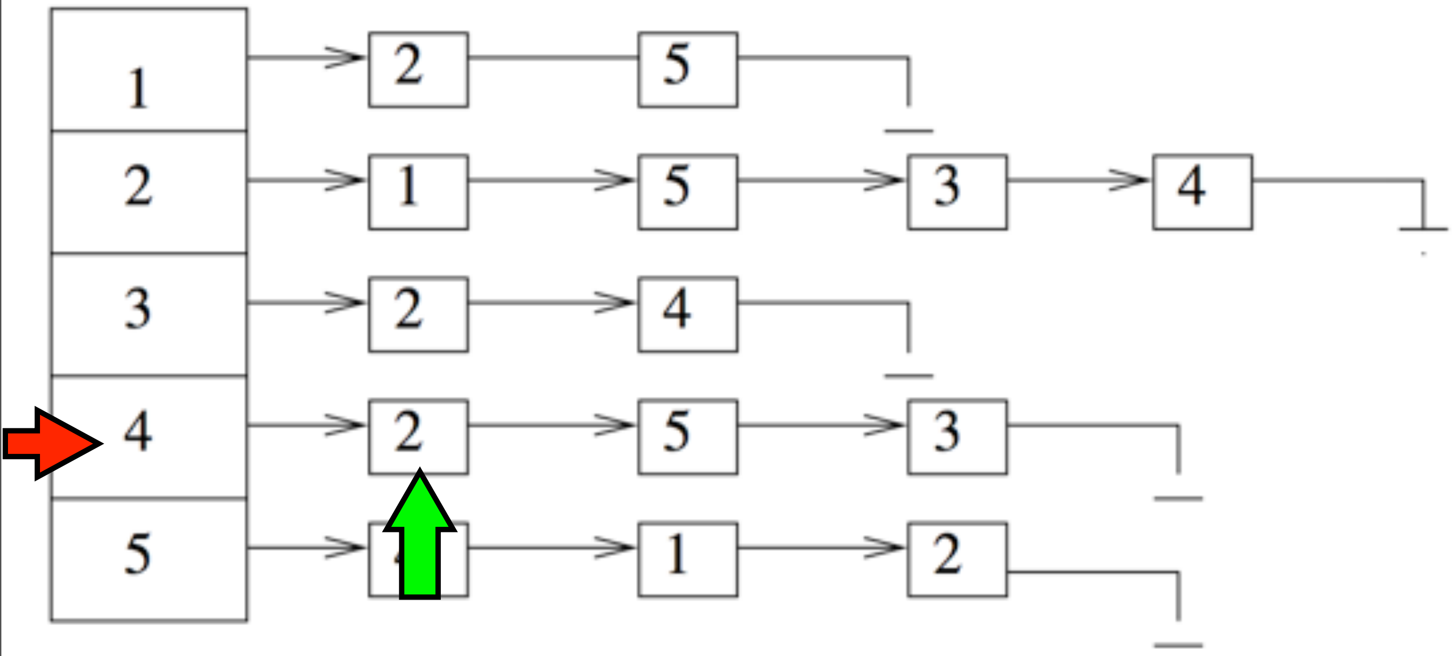
Q = ∅

u = 4

v = 2

Saídas

Vértices: 1, 2, 5, 3, 4
Arestas: (1,2) (1,5) (2,1) (2,5) (2,3) (2,4) (5,4) (5,1) (5,2) (3,2) (3,4)




```
BFS(G, s)
for each vertex u ∈ V[G]−{s}
     state[u] = "undiscovered"
     p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∈ Adj[u] do
          process edge (u,v)
          if state[v] = "undiscovered"
               state[v] = "discovered"
               p[v] = u
               enqueue[Q, v]
     state[u] = "processed"
```

Variáveis

	state	p
1	p	null
2	p	1
3	p	2
4	d	2
5	p	1

$Q = \varnothing$

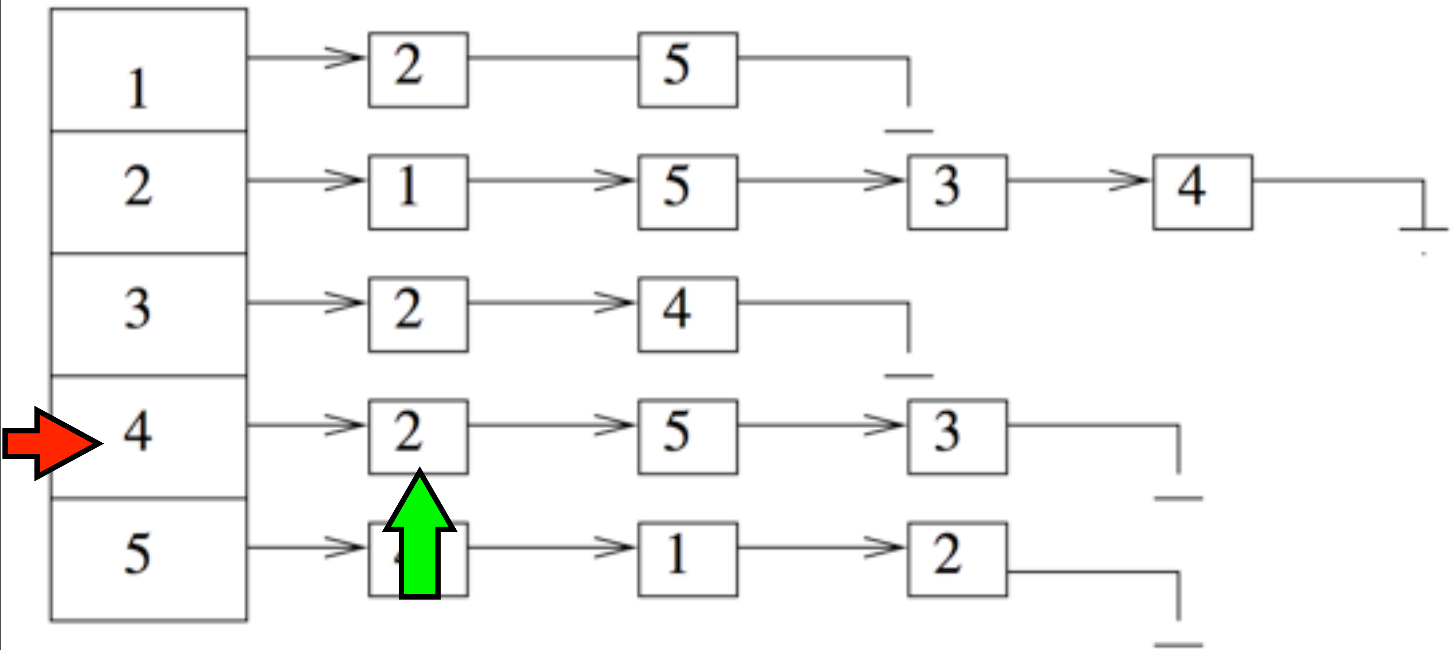
$u = 4$

$v = 2$

Saídas

Vértices: 1, 2, 5, 3, 4

Arestas: (1,2) (1,5) (2,1) (2,5) (2,3) (2,4) (5,4) (5,1) (5,2) (3,2) (3,4) (4,2)




```
BFS(G, s)
for each vertex u ∈ V[G]−{s}
     state[u] = "undiscovered"
     p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∈ Adj[u] do
          process edge (u,v)
          if state[v] = "undiscovered"
               state[v] = "discovered"
               p[v] = u
               enqueue[Q, v]
     state[u] = "processed"
```

Variáveis

	state	p
1	p	null
2	p	1
3	p	2
4	d	2
5	p	1

$Q = \varnothing$

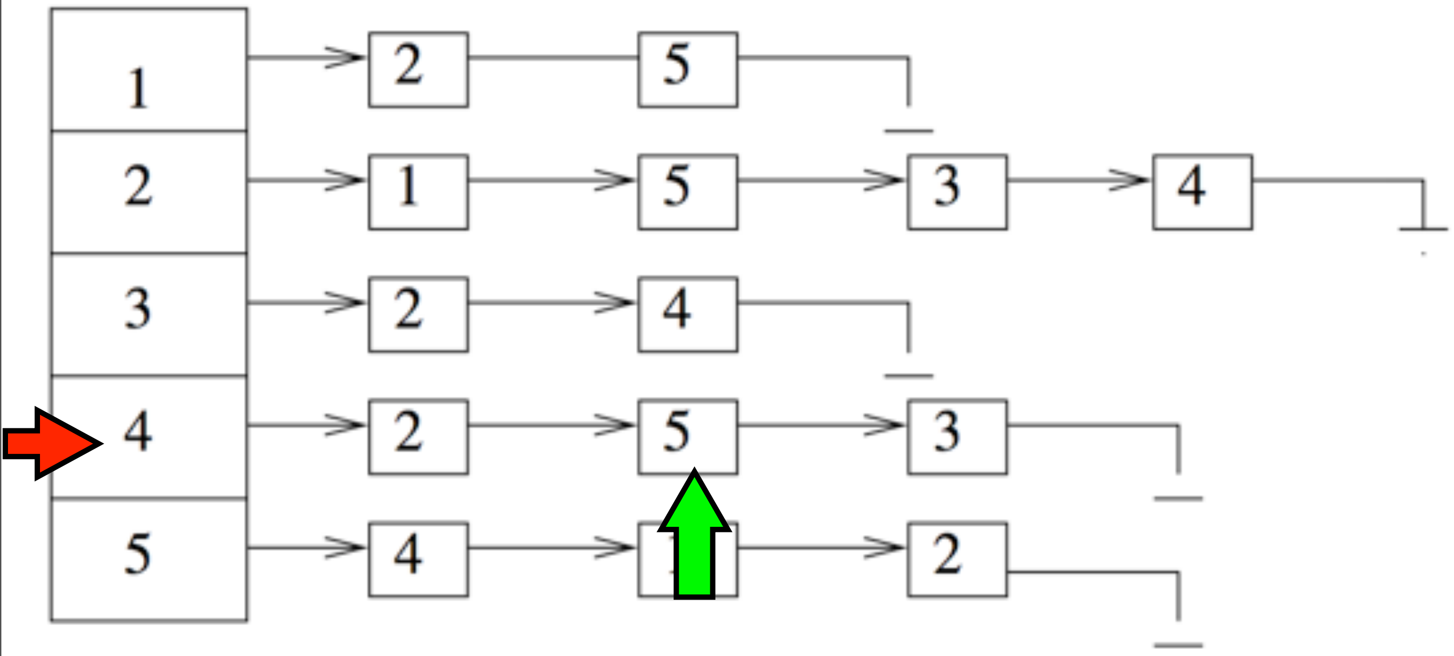
$u = 4$

$v = 2$

Saídas

Vértices: 1, 2, 5, 3, 4

Arestas: (1,2) (1,5) (2,1) (2,5) (2,3) (2,4) (5,4) (5,1) (5,2) (3,2) (3,4) (4,2)




```
BFS(G, s)
for each vertex u ∈ V[G]−{s}
     state[u] = "undiscovered"
     p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∈ Adj[u] do
          process edge (u,v)
          if state[v] = "undiscovered"
               state[v] = "discovered"
               p[v] = u
               enqueue[Q, v]
     state[u] = "processed"
```

Variáveis

	state	p
1	p	null
2	p	1
3	p	2
4	d	2
5	p	1

$Q = \varnothing$

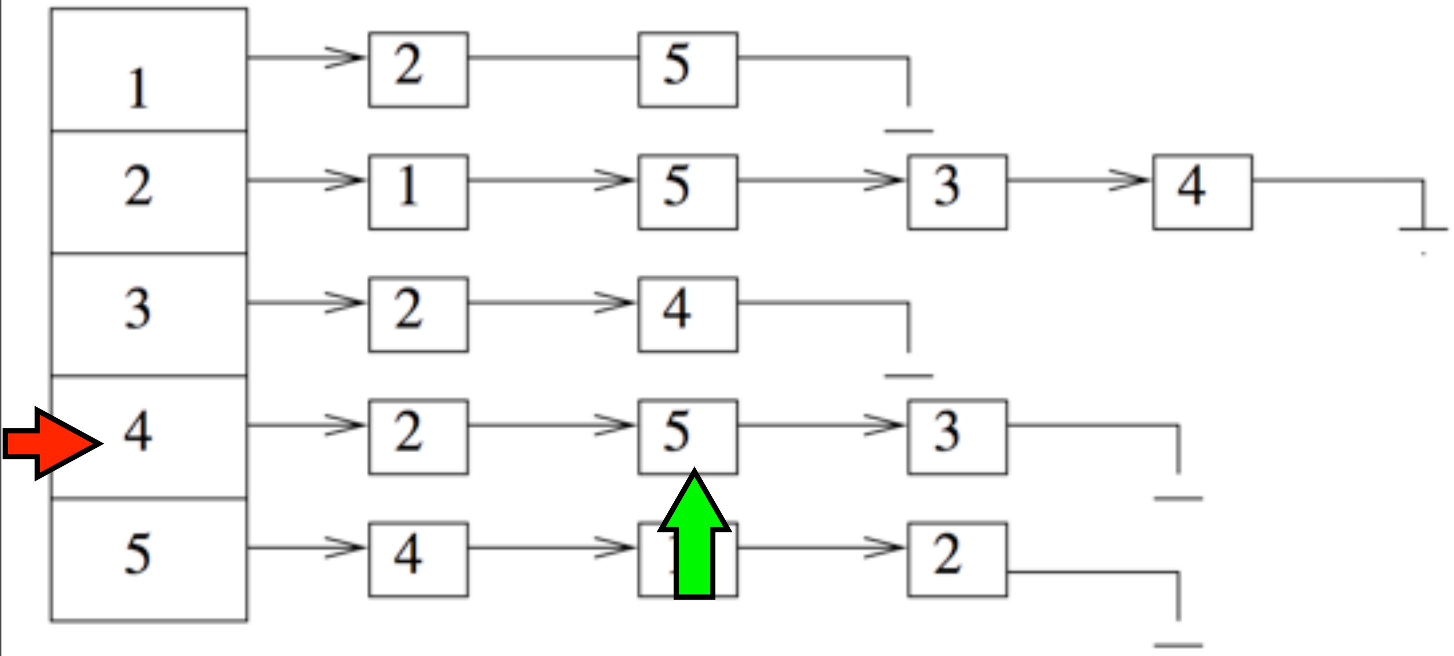
$u = 4$

$v = 5$

Saídas

Vértices: 1, 2, 5, 3, 4

Arestas: (1,2) (1,5) (2,1) (2,5) (2,3) (2,4) (5,4) (5,1) (5,2) (3,2) (3,4) (4,2)




```
BFS(G, s)
for each vertex u ∊ V[G]−{s}
    state[u] = "undiscovered"
    p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
    u = dequeue[Q]
    process vertex u
    for each v ∊ Adj[u] do
        process edge (u,v)
        if state[v] = "undiscovered"
            state[v] = "discovered"
            p[v] = u
            enqueue[Q, v]
    state[u] = "processed"
```

Variáveis

	state	p
1	p	null
2	p	1
3	p	2
4	d	2
5	p	1

Q = ∅

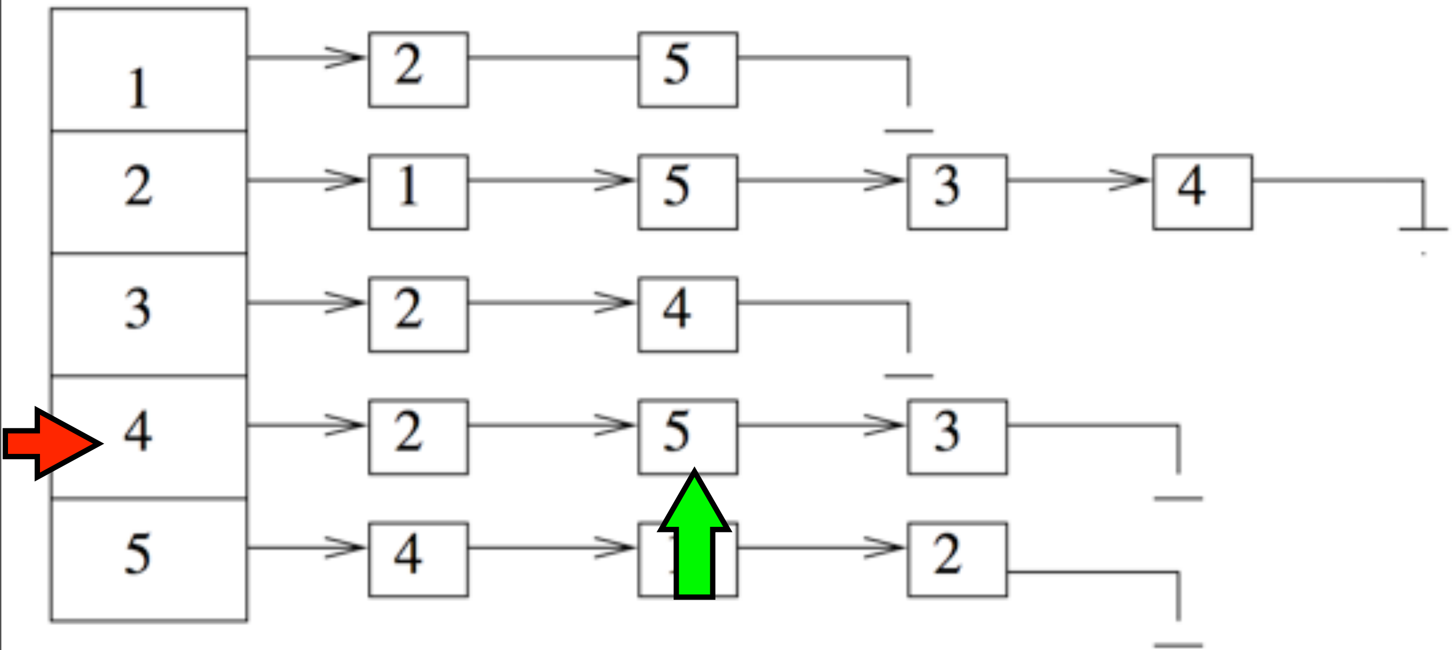
u = 4

v = 5

Saídas

Vértices: 1, 2, 5, 3, 4

Arestas: (1,2) (1,5) (2,1) (2,5) (2,3) (2,4) (5,4) (5,1) (5,2) (3,2) (3,4) (4,2) (4,5)




```
BFS(G, s)
for each vertex u ∊ V[G]−{s}
    state[u] = "undiscovered"
    p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
    u = dequeue[Q]
    process vertex u
    for each v ∊ Adj[u] do
        process edge (u,v)
        if state[v] = "undiscovered"
            state[v] = "discovered"
            p[v] = u
            enqueue[Q, v]
    state[u] = "processed"
```

Variáveis

	state	p
1	p	null
2	p	1
3	p	2
4	d	2
5	p	1

$Q = \varnothing$

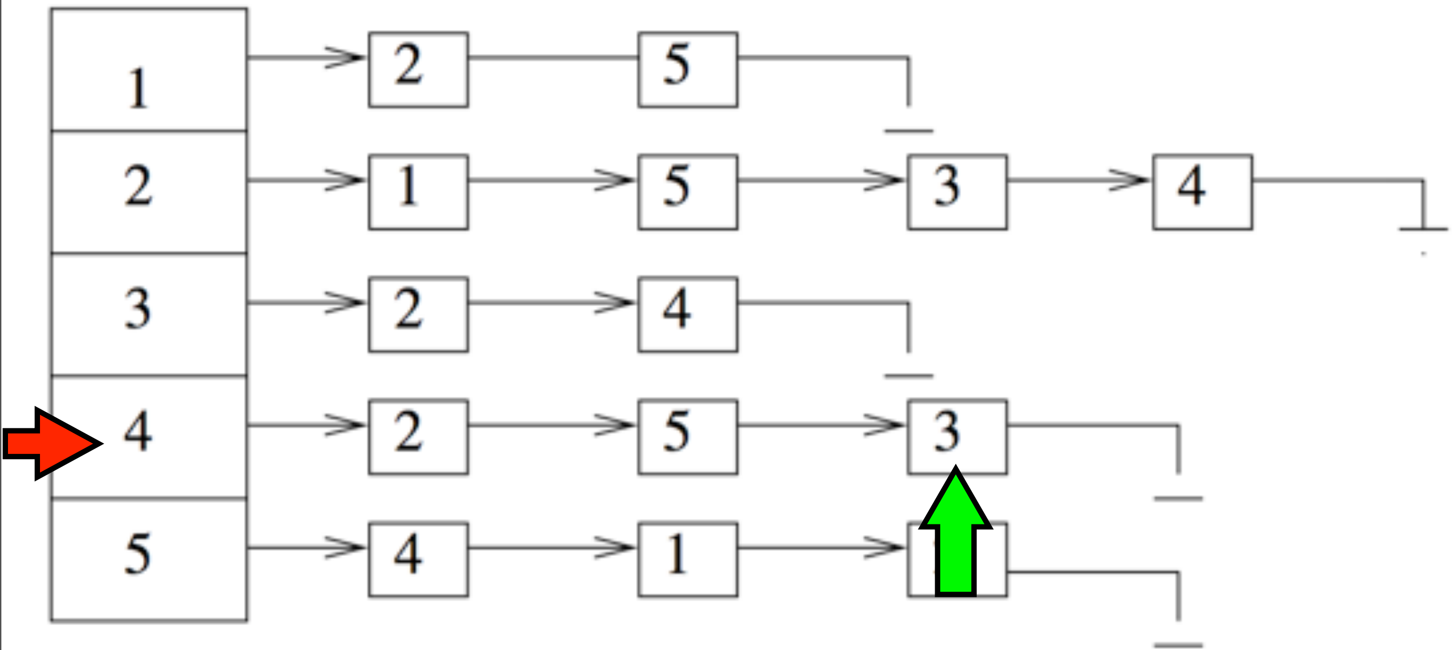
$u = 4$

$v = 5$

Saídas

Vértices: 1, 2, 5, 3, 4

Arestas: (1,2) (1,5) (2,1) (2,5) (2,3) (2,4) (5,4) (5,1) (5,2) (3,2) (3,4) (4,2) (4,5)




```
BFS(G, s)
for each vertex u ∊ V[G]−{s}
     state[u] = "undiscovered"
     p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∊ Adj[u] do
          process edge (u,v)
          if state[v] = "undiscovered"
               state[v] = "discovered"
               p[v] = u
               enqueue[Q, v]
     state[u] = "processed"
```

Variáveis

	state	p
1	p	null
2	p	1
3	p	2
4	d	2
5	p	1

$Q = \emptyset$

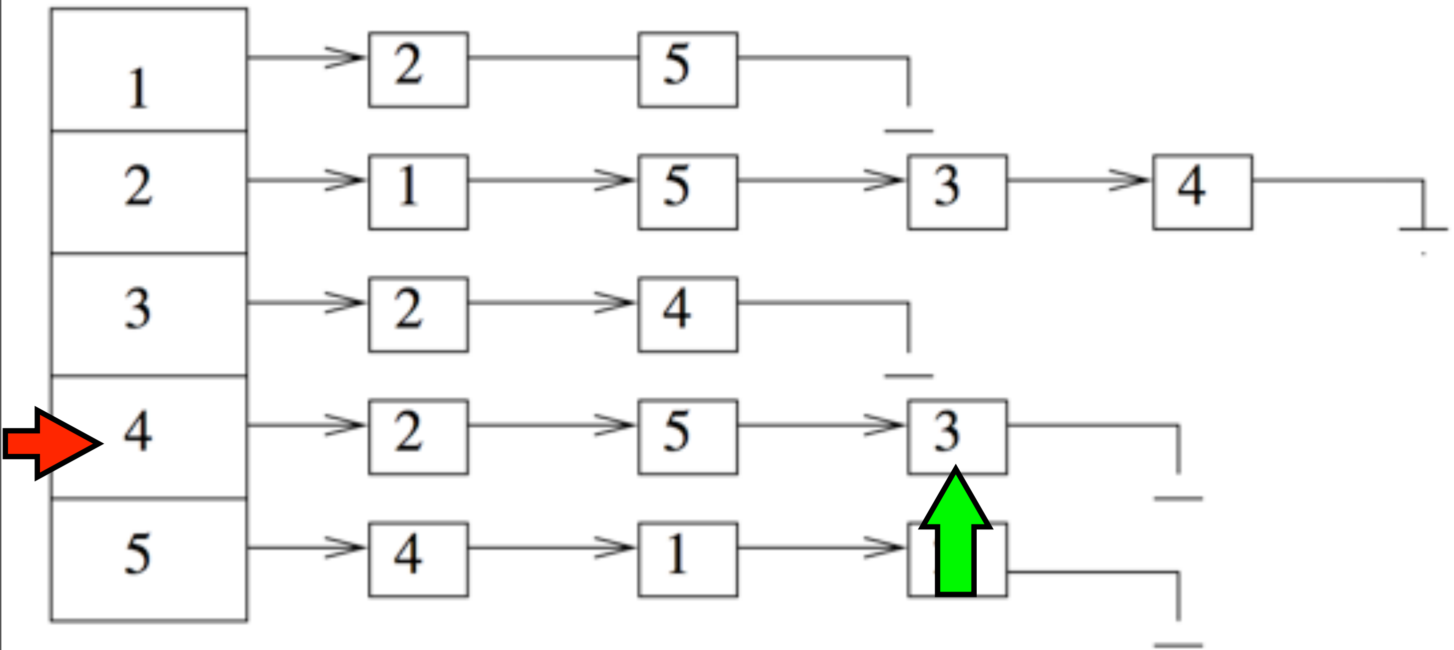
$u = 4$

$v = 3$

Saídas

Vértices: 1, 2, 5, 3, 4

Arestas: (1,2) (1,5) (2,1) (2,5) (2,3) (2,4) (5,4) (5,1) (5,2) (3,2) (3,4) (4,2) (4,5)




```
BFS(G, s)
for each vertex u ∈ V[G]−{s}
     state[u] = "undiscovered"
     p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∈ Adj[u] do
          process edge (u,v)
          if state[v] = "undiscovered"
               state[v] = "discovered"
               p[v] = u
               enqueue[Q, v]
     state[u] = "processed"
```

Variáveis

	state	p
1	p	null
2	p	1
3	p	2
4	d	2
5	p	1

Q = ∅

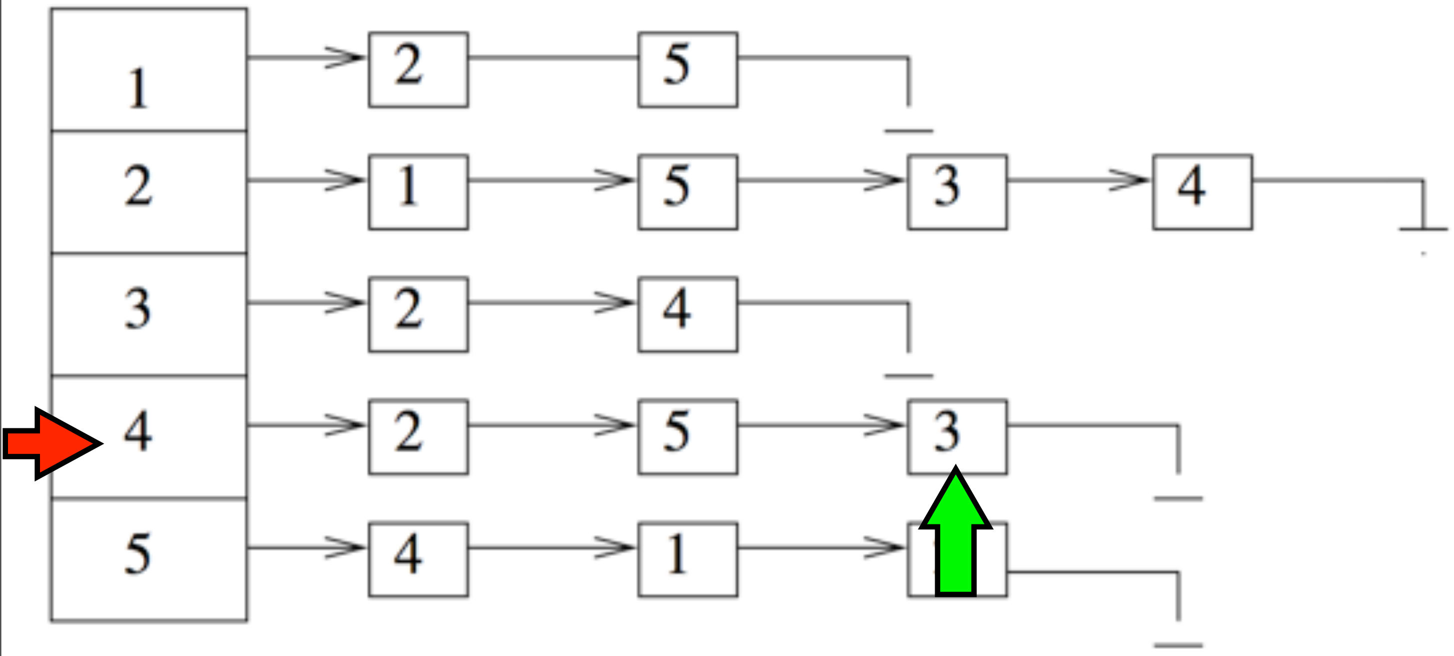
u = 4

v = 3

Saídas

Vértices: 1, 2, 5, 3, 4

Arestas: (1,2) (1,5) (2,1) (2,5) (2,3) (2,4) (5,4) (5,1) (5,2) (3,2) (3,4) (4,2) (4,5) (4,3)




```
BFS(G, s)
for each vertex u ∈ V[G]−{s}
    state[u] = "undiscovered"
    p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
    u = dequeue[Q]
    process vertex u
    for each v ∈ Adj[u] do
        process edge (u,v)
        if state[v] = "undiscovered"
            state[v] = "discovered"
            p[v] = u
            enqueue[Q, v]
    state[u] = "processed"
```

Variáveis

	state	p
1	p	null
2	p	1
3	p	2
4	p	2
5	p	1

Q = ∅

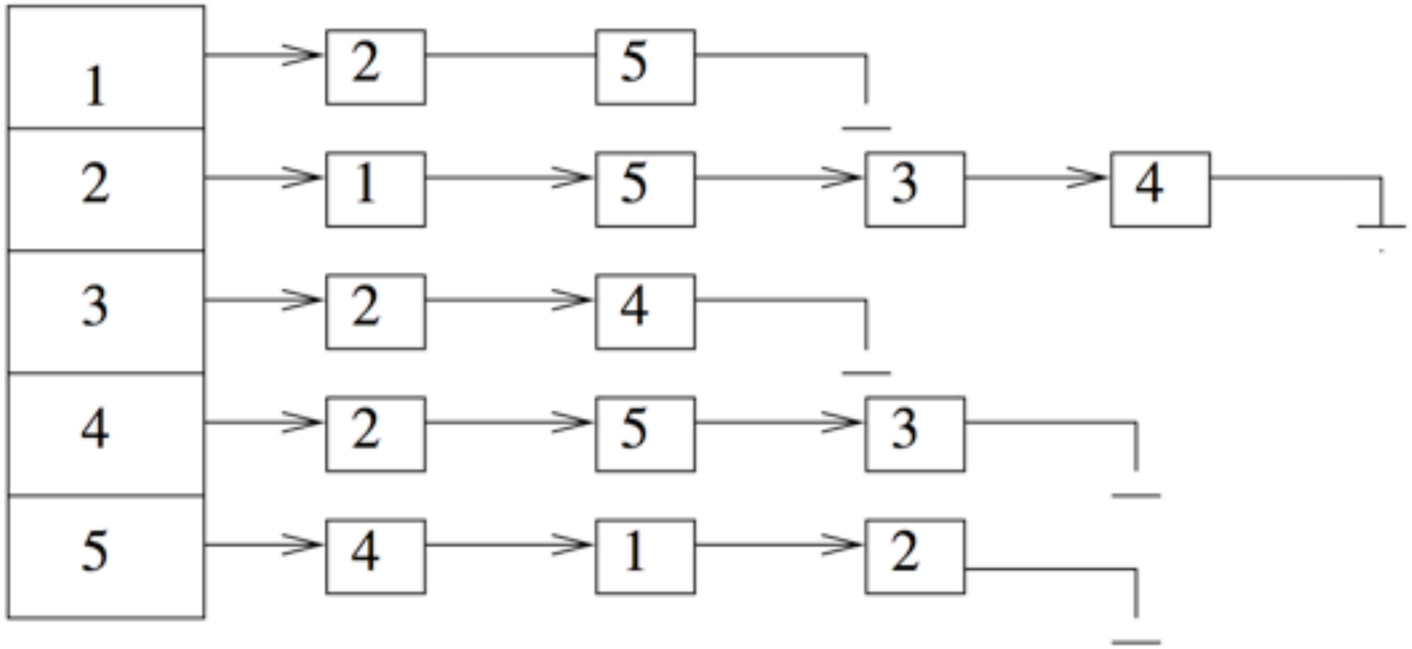
u = 4

v = 3

Saídas

Vértices: 1, 2, 5, 3, 4

Arestas: (1,2) (1,5) (2,1) (2,5) (2,3) (2,4) (5,4) (5,1) (5,2) (3,2) (3,4) (4,2) (4,5) (4,3)




```
BFS(G, s)
for each vertex u ∊ V[G]−{s}
     state[u] = "undiscovered"
     p[u] = null, // sem pais
state[s] = "discovered"
p[s] = null
Q = {s}
while (Q != ∅)
     u = dequeue[Q]
     process vertex u
     for each v ∊ Adj[u] do
          process edge (u,v)
          if state[v] = "undiscovered"
               state[v] = "discovered"
               p[v] = u
               enqueue[Q, v]
     state[u] = "processed"
```

Variáveis

	state	p
1	p	null
2	p	1
3	p	2
4	p	2
5	p	1

Q = ∅

u = 4

v = 3

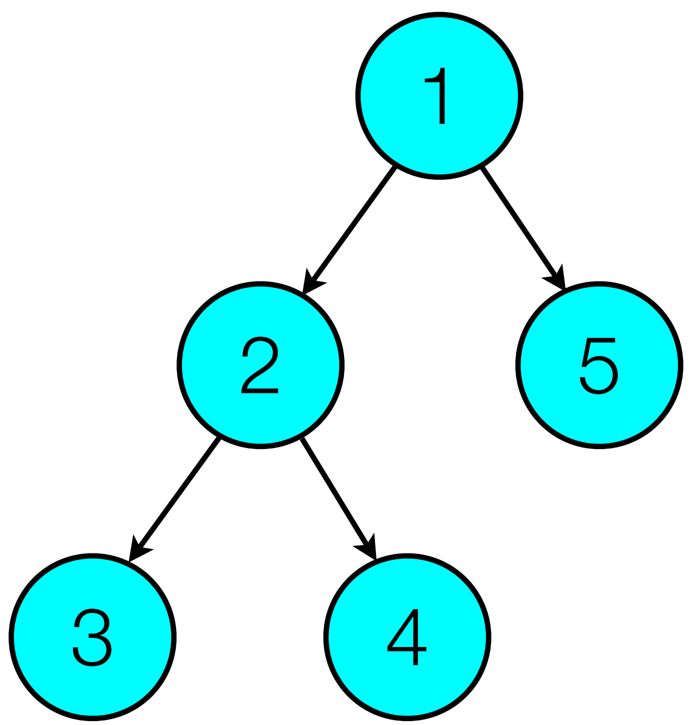
Saídas

Vértices: 1, 2, 5, 3, 4

Arestas: (1,2) (1,5) (2,1) (2,5) (2,3) (2,4) (5,4) (5,1) (5,2) (3,2) (3,4) (4,2) (4,5) (4,3)

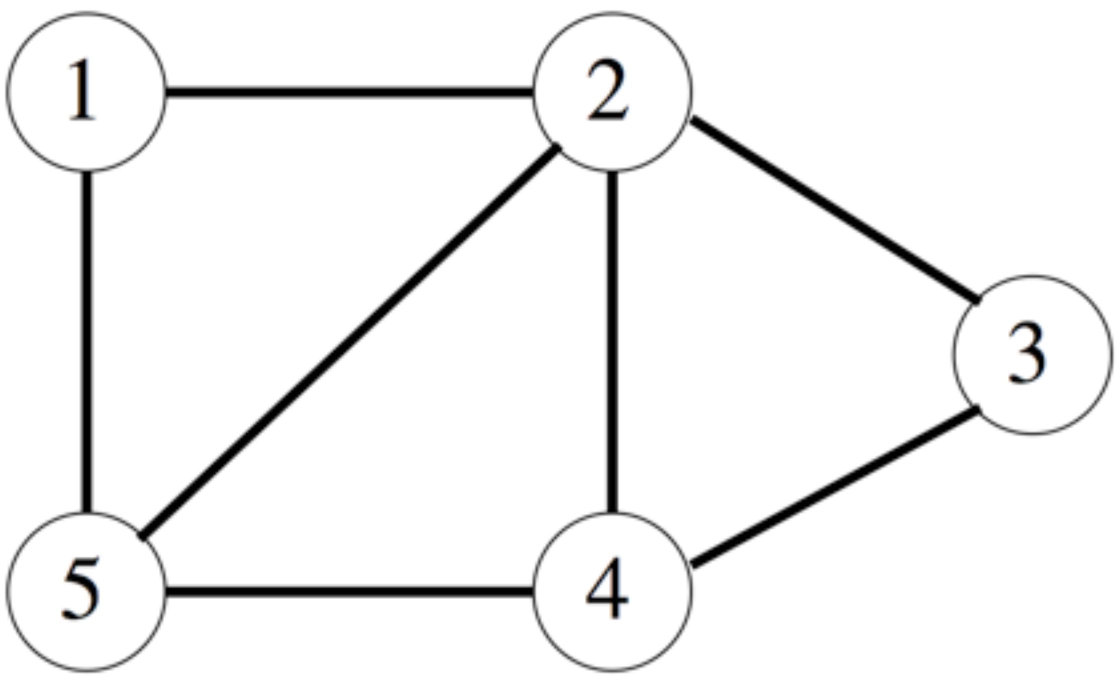
Árvore gerada e percurso em largura

	p
1	null
2	1
3	2
4	2
5	1



Vértices: 1, 2, 5, 3, 4

Arestas visitadas



Arestas: (1,2) (1,5) (2,1) (2,5) (2,3) (2,4) (5,4) (5,1) (5,2) (3,2) (3,4) (4,2) (4,5) (4,3)

Encontrando caminhos

- O vértice que descobriu o vértice i é denominado pai de i.
- Desta forma, cada vértice tem apenas um pai, e este é armazenado no vetor p.
- Como os vértices são descobertos em ordem crescente de distância da raiz, esta árvore tem uma propriedade importante: o caminho do nó x até a raiz através dela usa o menor número possível de arestas.
- Só funciona com grafos não ponderados. Para grafos ponderados tem-se outros algoritmos (mais adiante).

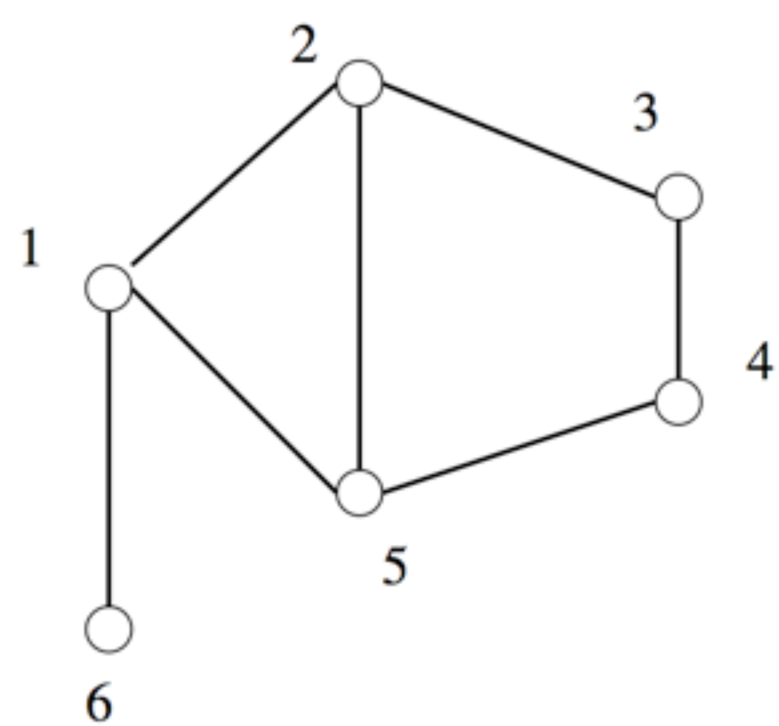
Problema

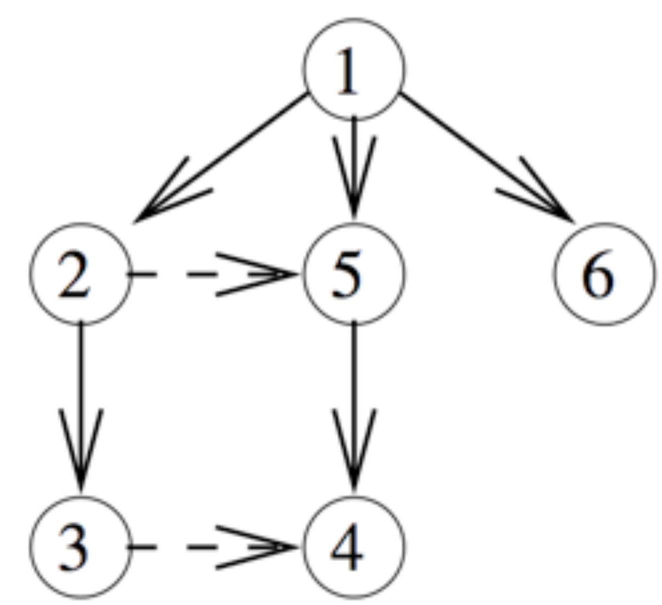
- Partindo de um vértice x, podemos chegar à raiz, mas isso em geral é o inverso do que queremos.
- Soluções:
    - encontrar o caminho e depois invertê-lo usando uma pilha
    - usar recursão para fazer isso

Usando recursão

```
find_path(int start, int end, int parents[])
{
        if ((start == end) || (end == -1))
                printf("\n%d",start);
        else {
                find_path(start,parents[end],parents);
                printf(" %d",end);
        }
}
```

vertex	1	2	3	4	5	6
parent	-1	1	2	5	1	1





Qual o menor caminho entre os vértices 1 e 4?

Aplicações da busca em largura

- Componentes conectados
- Problema de coloração de grafos

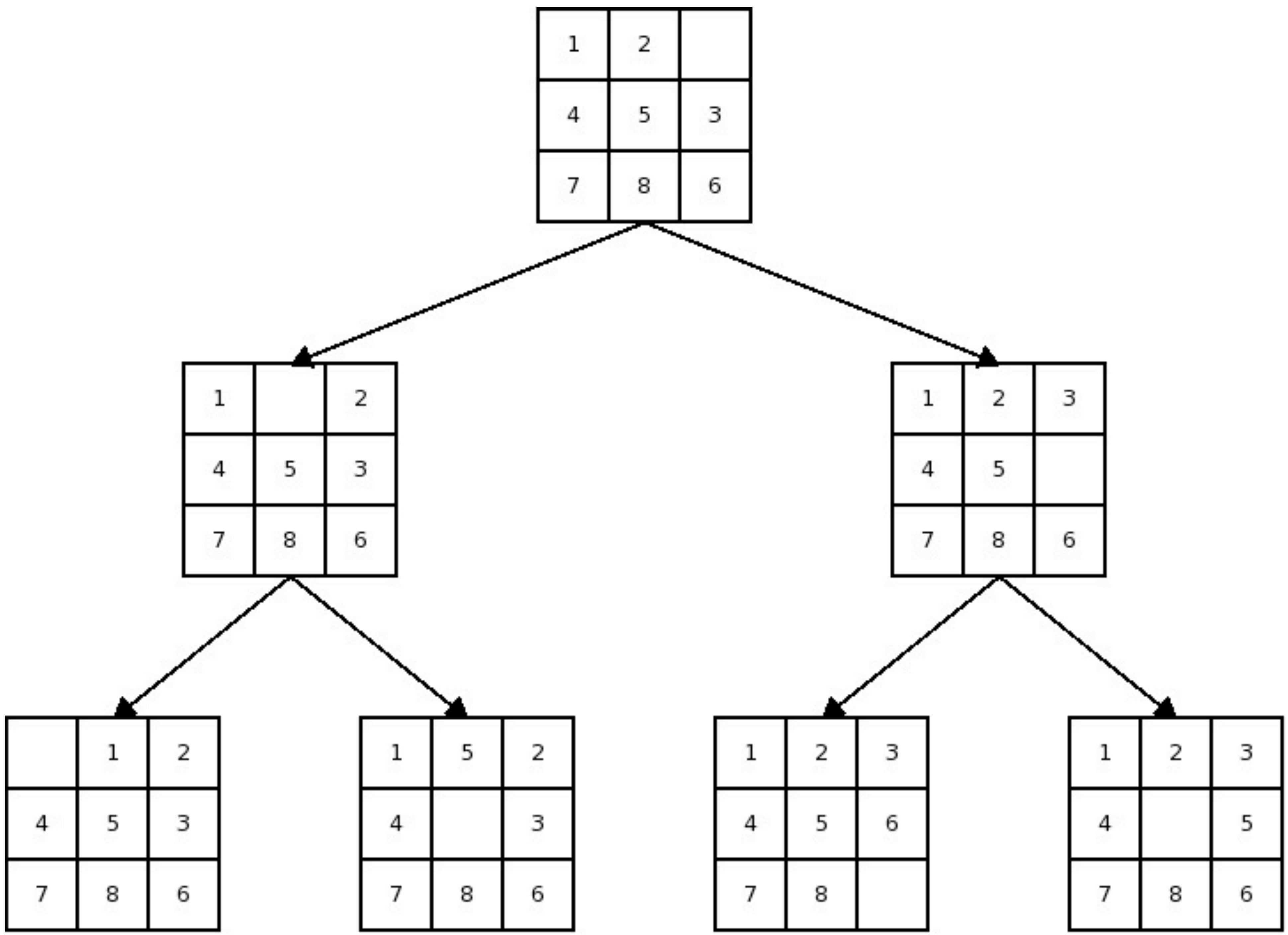
Componentes conectados

- Dizemos que um grafo é conectado se existe um caminho entre dois vértices quaisquer.
- Um componente conectado de um grafo não direcionado é o maior conjunto de vértices tal que existe um caminho entre cada par de vértices. Os componentes são pedaços do grafo sem conexão.
- Exemplo: um componente conectado poderia representar o grafo de amizades de uma tribo isolada do mundo.

Pra que serve isso?

Como?

Assim:


1 2
4 5 3
7 8 6
1 2
4 5 3
7 8 6
1 2 3
4 5
7 8 6
1 2
4 5 3
7 8 6
1 5 2
4 3
7 8 6
1 2 3
4 5 6
7 8
1 2 3
4 5
7 8 6

Como descobrir componentes conectados

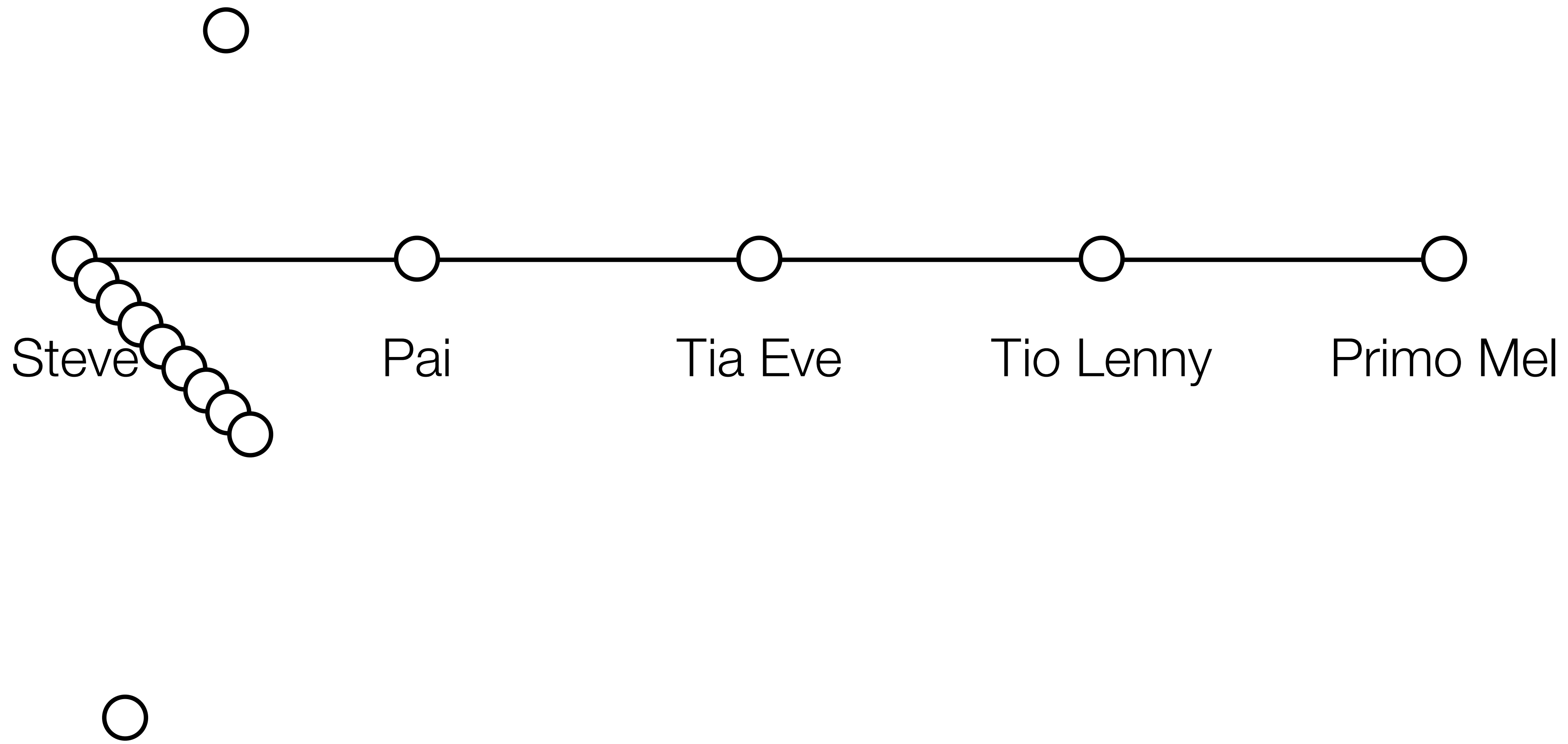
- Iniciamos com um vértice qualquer e realizamos uma busca em largura.
- Todos os vértices conectados a ele vão ser descobertos.
- Se houver vértices ainda não visitados, então temos um novo componente. Repetimos o procedimento para descobrir todos os vértices deste novo componente.
- Este procedimento é repetido até que todos os vértices do grafo tenham sido descobertos.

Problema de coloração de grafos

- Objetivo: atribuir um rótulo (ou cor) para cada vértice de um grafo de modo que nenhuma aresta ligue vértices de mesma cor.
- Problema trivial se usarmos uma cor para cada vértice, mas o objetivo é usar o menor número possível de cores.
- Aplicação: alocação de registradores em compiladores.

Grafo bipartido

- Um grafo é bipartido se puder ser colorido sem conflito usando somente duas cores.
- Podemos usar um percurso em largura para verificar se um grafo pode ser bipartido: atribui-se uma cor ao nó raiz e a outra cor aos seus filhos. Se conseguirmos repetir este processo até o final do percurso sem conflitos, então este grafo pode ser bipartido.


Steve
Pai
Tia Eve
Tio Lenny
Primo Mel