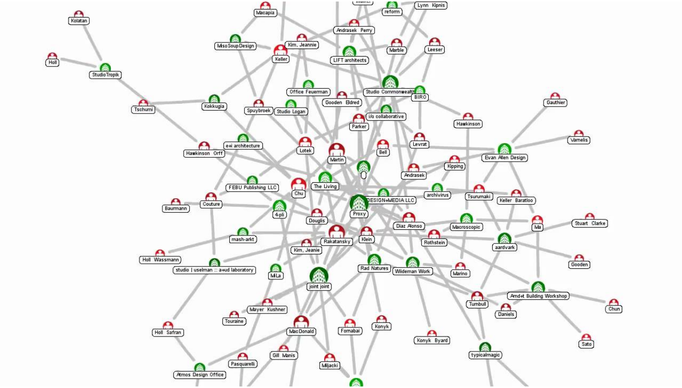
AULA 20

Digrafos



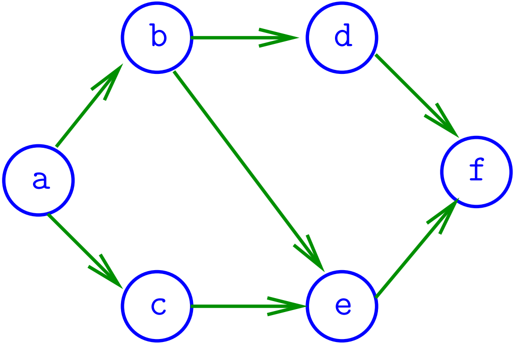
Fonte: Force Directed Graph

Referências: Directed graphs (SW): slides, vídeo.

Digrafos

Um **digrafo** (*directed graph*) consiste de um conjunto de vértices (bolas) e um conjunto de arcos (flechas)

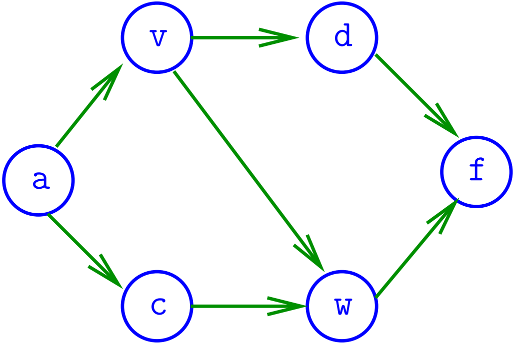
Exemplo: representação de um digrafo



Arcos

Um **arco** é um par ordenado de vértices

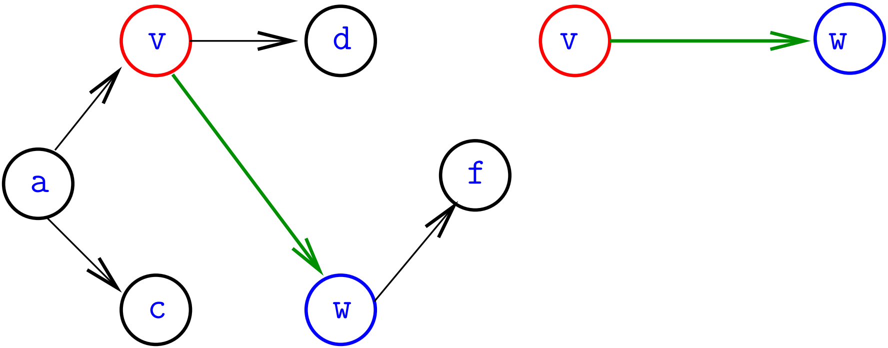
Exemplo: `v` e `w` são vértices e `v-w` é um arco



Ponta inicial e final

Para cada arco `v-w`, o vértice `v` é a **ponta inicial** e `w` é a **ponta final**

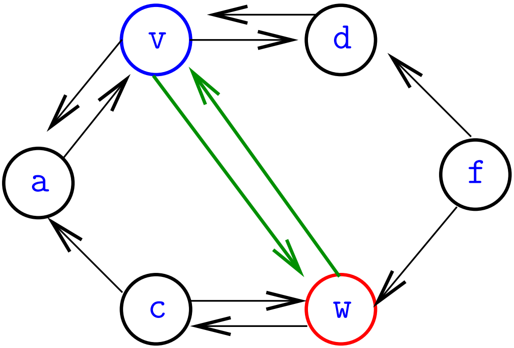
Exemplo: `v` é ponta inicial e `w` é ponta final de `v-w`



Arcos anti-paralelos

Dois arcos são **anti-paralelos** se a ponta inicial de um é ponta final do outro

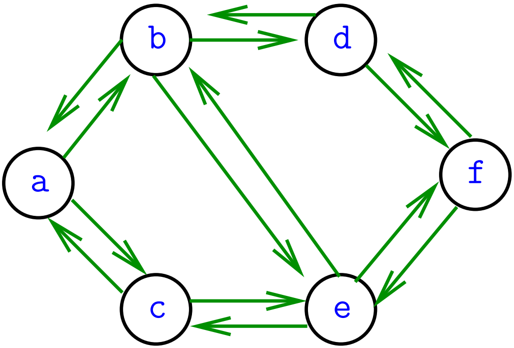
Exemplo: `v-w` e `w-v` são anti-paralelos



Digrafos simétricos

Um digrafo é **simétrico** se cada um de seus arcos é anti-paralelo a outro

Exemplo: digrafo simétrico

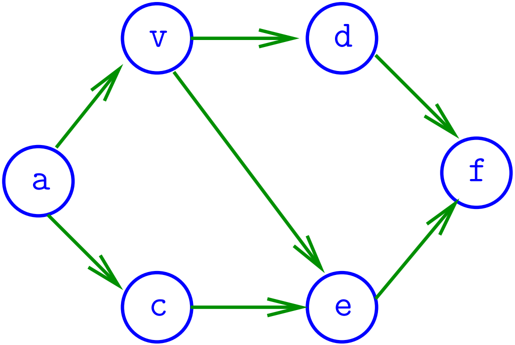


Graus de entrada e saída

grau de entrada de $\mathtt{v}$= no. arcos com ponta final $\mathtt{v}$

grau de saída de $\mathtt{v}$ = no. arcos com ponta inicial $\mathtt{v}$

Exemplo: $\mathtt{v}$ tem grau de entrada 1 e de saída 2



Número de arcos

Quantos arcos, no máximo, tem um digrafo com `V` vértices?

Número de arcos

Quantos arcos, no máximo, tem um digrafo com $\mathtt{V}$ vértices?

A resposta é $\mathtt{V} \times (\mathtt{V} - 1) = \Theta(\mathtt{V}^2)$

digrafo **completo** = todo par ordenado de vértices distintos é arco

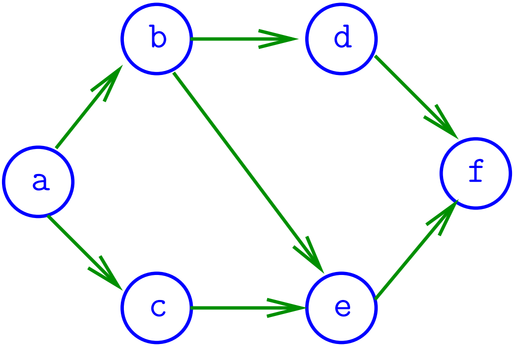
digrafo **denso** = tem "muitos" muitos arcos

digrafo **esparso** = tem "poucos" arcos

Especificação

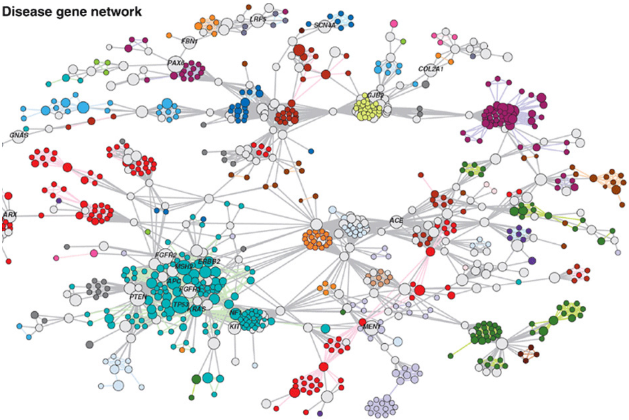
Digrafos podem ser especificados através de sua lista de arcos

Exemplo:




```
d-f
b-d
a-c
b-e
e-f
a-b
```

Grafos



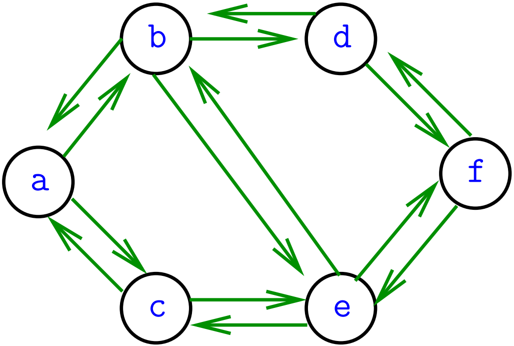
Fonte: Scaling Computation of Graph Structured Data with NScale

Referências: Undirected graphs (SW): slides, vídeo.

Grafos

Um **grafo** é um digrafo **simétrico**

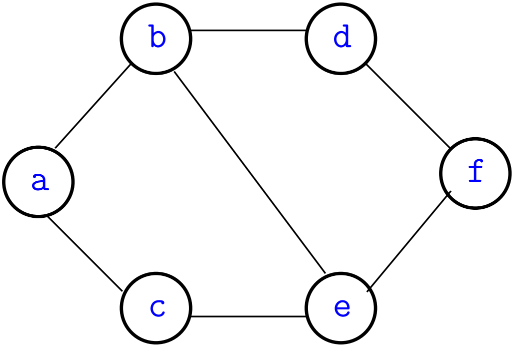
Exemplo: um grafo



Grafos

Um **grafo** é um digrafo **simétrico**

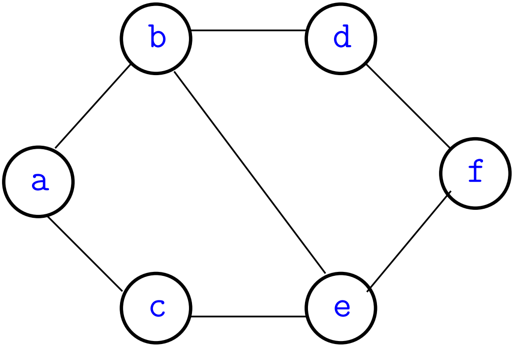
Exemplo: representação usual



Arestas

Uma **aresta** é um par de arcos anti-paralelos.

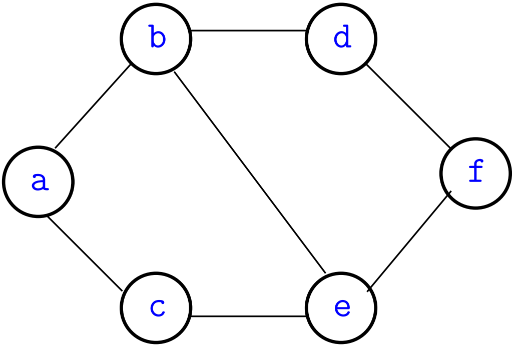
Exemplo: `b-a` e `a-b` são a **mesma** aresta



Especificação

Grafos podem ser especificados através de sua lista de arestas

Exemplo:



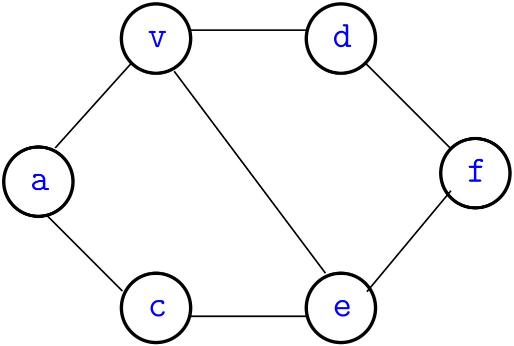

```
f-d
b-d
c-a
e-b
e-f
a-b
```

Graus de vértices

Em um grafo

grau de v = número de arestas com ponta em v

Exemplo: v tem grau 3



Número de arestas

Quantas arestas, no máximo, tem um grafo com `V` vértices?

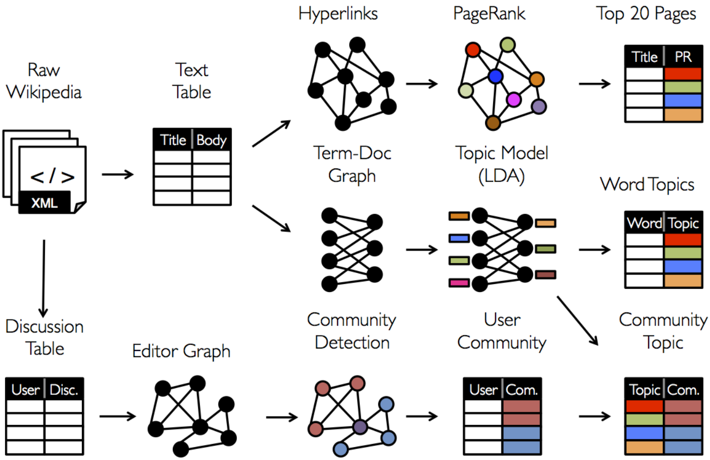
Número de arestas

Quantas arestas, no máximo, tem um grafo com $\mathtt{V}$ vértices?

A resposta é $\mathtt{V} \times (\mathtt{V} - 1)/2 = \Theta(\mathtt{V}^2)$

grafo **completo** = todo par **não**-ordenado de vértices distintos é aresta

Digrafos no computador



Fonte: GraphX Programming Guide

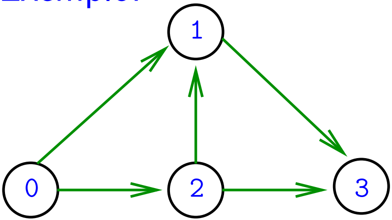
Matriz de adjacência de digrafos

Matriz de adjacência de um digrafo tem linhas e colunas indexadas por vértices:

`adj[v][w]` = 1 se v-w é um arco

`adj[v][w]` = 0 em caso contrário

Exemplo:



	0	1	2	3
0	0	1	1	0
1	0	0	0	1
2	0	1	0	1
3	0	0	0	0

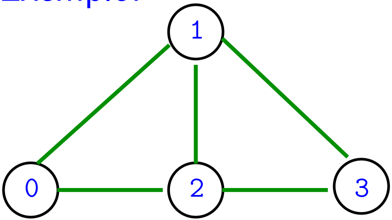
Consumo de espaço: $\Theta(V^2)$ fácil de implementar

Matriz de adjacência de grafos

Matriz de adjacência de um grafo tem linhas e colunas indexadas por vértices:

adj[v][w] = 1 se v-w é um aresta
adj[v][w] = 0 em caso contrário

Exemplo:

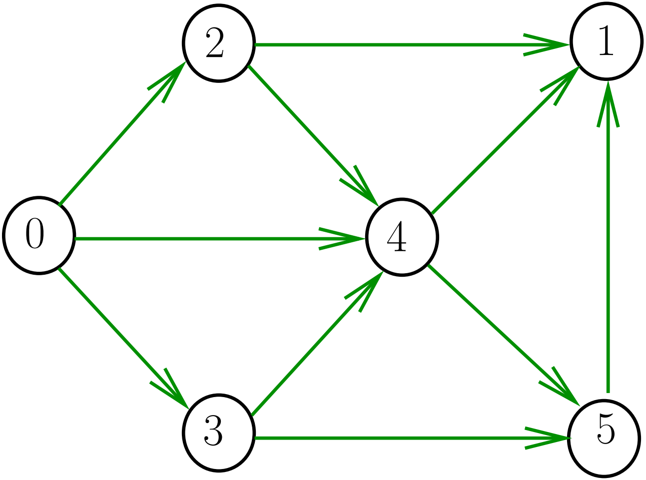


	0	1	2	3
0	0	1	1	0
1	1	0	1	1
2	1	1	0	1
3	0	1	1	0

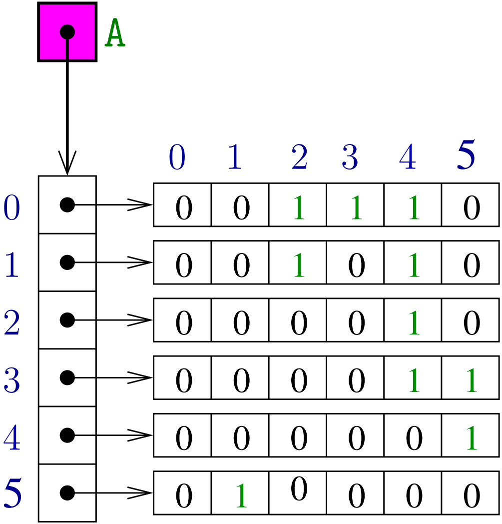
Consumo de espaço: $\Theta(V^2)$ fácil de implementar

Digrafo

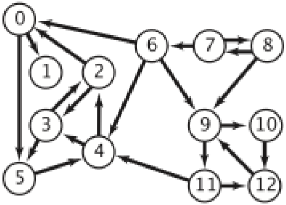
`Digraph G`



Estruturas de dados



Digrafos no algs4



	0	1	2	3	4	5	6	7	8	9	10	11	12
0	T	T	T	T	T	T							
1		T											
2	T	T	T	T	T	T							
3	T	T	T	T	T	T							
4	T	T	T	T	T	T							
5	T	T	T	T	T	T							
6	T	T	T	T	T	T	T			T	T	T	T
7	T	T	T	T	T	T	T	T	T	T	T	T	T
8	T	T	T	T	T	T	T	T	T	T	T	T	T
9	T	T	T	T	T	T				T	T	T	T
10	T	T	T	T	T	T				T	T	T	T
11	T	T	T	T	T	T				T	T	T	T
12	T	T	T	T	T	T				T	T	T	T

original edge (red)

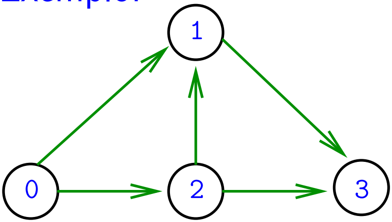
self-loop (gray)

12 is reachable from 6

Vetor de listas de adjacência de digrafos

Na representação de um digrafo através de **listas de adjacência** tem-se, para cada vértice `v`, uma lista dos vértices que são vizinhos `v`.

Exemplo:




```
0:  1,  2
1:   3
2:  1,  3
3:
```

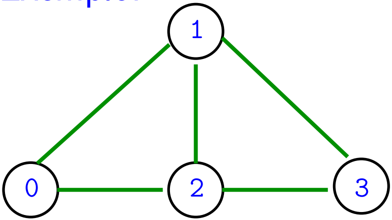
Consumo de espaço: $\Theta(V + A)$ (linear)

Manipulação eficiente

Vetor de lista de adjacência de grafos

Na representação de um grafo através de **listas de adjacência** tem-se, para cada vértice `v`, uma lista dos vértices que são pontas de arestas incidentes a `v`

Exemplo:



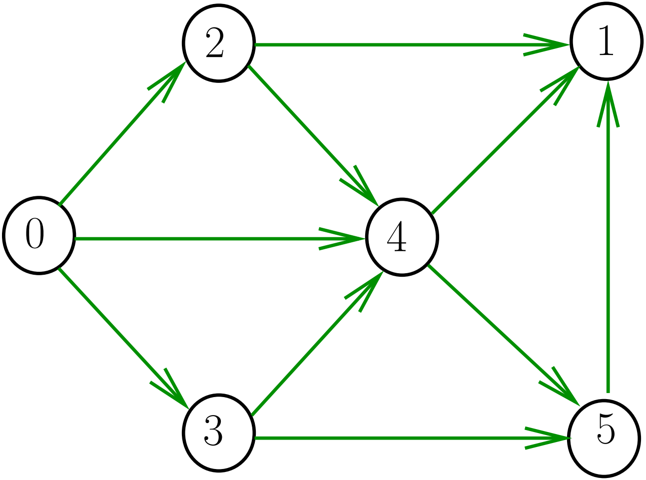

```
0:  1,  2
1:  3,  0,  2
2:  1,  3,  0
3:  1,  2
```

Consumo de espaço: $\Theta(\mathtt{V} + \mathtt{A})$ (linear)

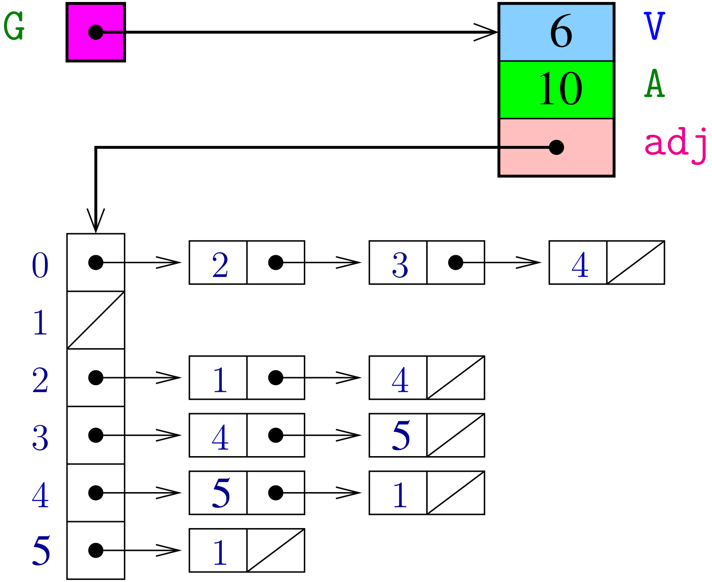
Manipulação eficiente

Digrafo

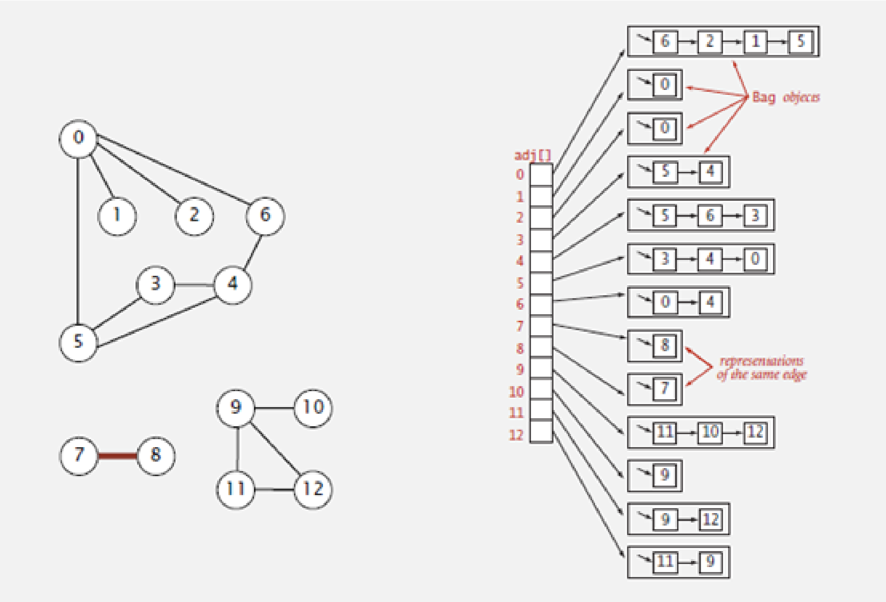
Digraph G



Estruturas de dados



Grafos no algs4



Esqueleto da classe Digraph

```
public class Digraph {
   private int V; // no.  vértices
   private int E; // no.  arcos
   private Bag<Integer>[] adj;
   private int[] indegree;
   public Digraph(int V) {...}
   public int V() { return V; }
   public int E() { return E; }
   public void addEdge(int v, int w) { }
   public Iterable<Integer> adj(intv) { }
   public int outdegree(int v) {...}
   public int indegree(int v) {...}
   public Digraph reverse() { ...}
}
```

Digraph

```
public Digraph(int V) {
   this.V = V;
   this.E = 0;
   indegree = new int[V];
   adj = (Bag<Integer>[]) new Bag[V];
   for (int v = 0; v < V; v++) {
      adj[v] = new Bag<Integer>();
   }
}
```

Digraph

```
// insere um arco
public void addEdge(int v, int w) {
   adj[v].add(w);
   indegree[w]++;
   E++;
}

// retorna a lista de adjacência de v
public Iterable<Integer> adj(int v) {
   return adj[v];
}
```

Digraph

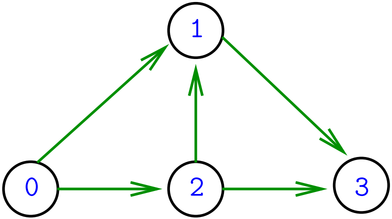
```
// retorna o grau de saída de v
public int outdegree(int v) {
   return adj[v].size();
}
// retorna o grau de entrada de v
public int indegree(int v) {
   return indegree[v];
}
```

Digraph

```
// retorna o sigrafo reverso
public Digraph reverse() {
   Digraph reverse = new Digraph(V);
   for (int v = 0; v < V; v++) {
       for (int w :  adj(v)) {
           reverse.addEdge(w, v);
       }
   }
   return reverse;
}
```

Matriz de incidência de digrafos

Uma **matriz de incidências** de um digrafo tem linhas indexadas por vértices e colunas por arcos e cada entrada $[k][vw]$ é -1 se $k = v$, $+1$ se $k = w$, e 0 em caso contrário.



	0-1	0-2	2-1	2-3	1-3
0	-1	-1	0	0	0
1	$+1$	0	$+1$	0	-1
2	0	$+1$	-1	-1	0
3	0	0	0	$+1$	$+1$

Consumo de espaço: $\Theta(nm)$

Interessante do ponto de vista de otimização linear.

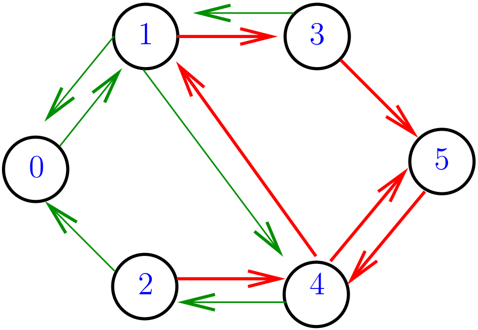
Caminhos em digrafos

Fonte: Finding Your Way & Making You A Priority

Caminhos

Um **caminho** num digrafo é qualquer seqüência da forma v_0–v_1–v_2–...–v_{k-1}–v_p, onde v_{k-1}-v_k é um arco para $k = 1, \ldots, p$.

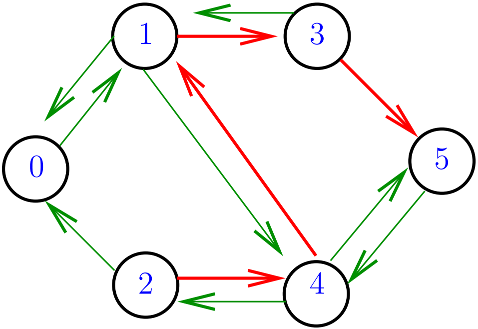
Exemplo: 2-4-1-3-5-4-5 é um caminho com **origem** 2 é **término** 5



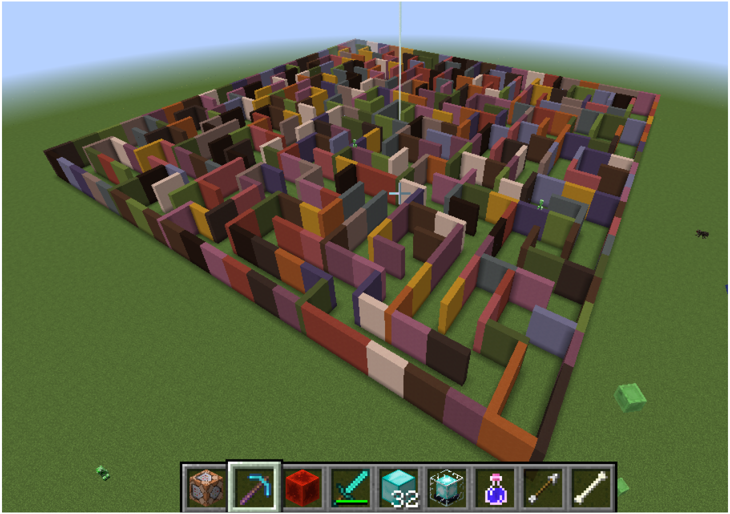
Caminhos simples

Um caminho é **simples** se não tem vértices repetidos

Exemplo: 2-4-1-3-5 é um caminho simples de 2 a 5



Procurando caminhos

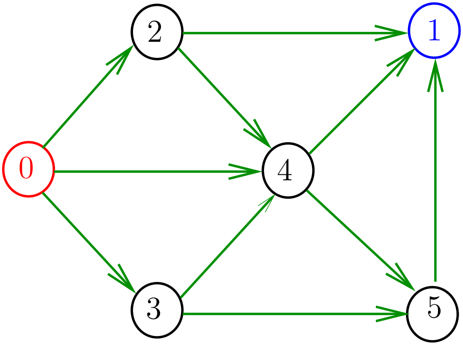


Fonte: Mincecraft maze created by Carl Eklof (algs4)

Procurando um caminho

Problema: dados um digrafo `G` e dois vértices `s` e `t` decidir se existe um caminho de `s` a `t`

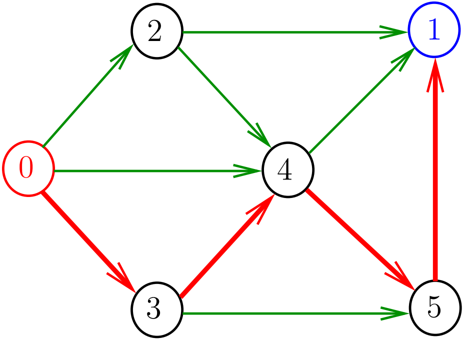
Exemplo: para `s` $= 0$ e `t` $= 1$ a resposta é SIM



Procurando um caminho

Problema: dados um digrafo G e dois vértices s e t decidir se existe um caminho de s a t

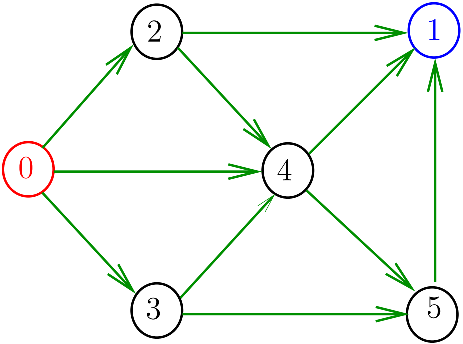
Exemplo: para $s = 0$ e $t = 1$ a resposta é SIM



Procurando um caminho

Problema: dados um digrafo G e dois vértices s e t decidir se existe um caminho de s a t

Exemplo: para $s = 5$ e $t = 4$ a resposta é NÃO

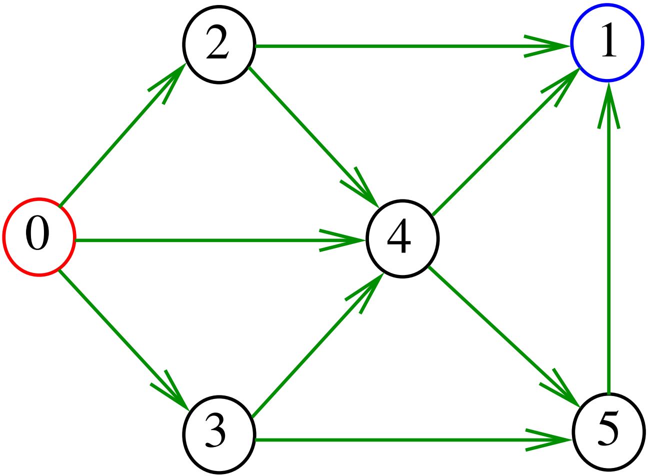


DFSpaths

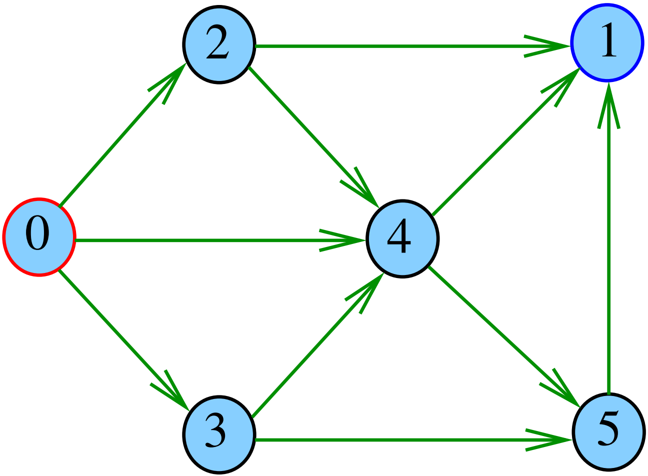
A classe DFSpaths recebe um digrafo G e um vértice s e determina todos os vértices alcançáveis a partir de s.
A classe implementa a técnica chamada busca em profundidade (*Depth First Search*).

```
public class DFSpaths{
   public void DFSpaths(Digraph G,int s){}

   // retorna true se há caminho de s a v
   public boolean hasPath(int v){ ...}

   private void dfs(Digraph G, int v) { }
}
```

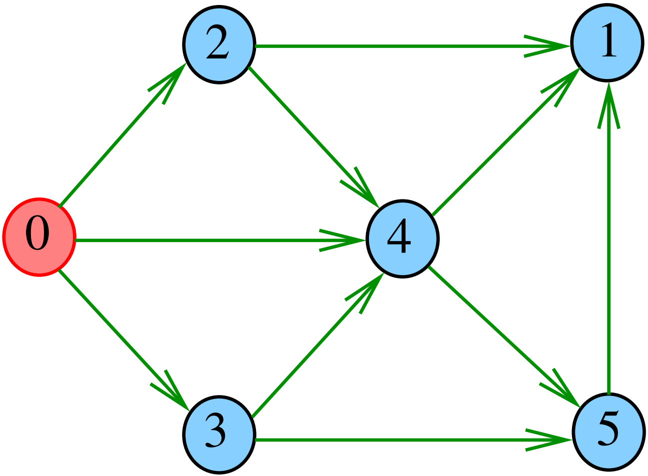
DFSpaths(G, 0)



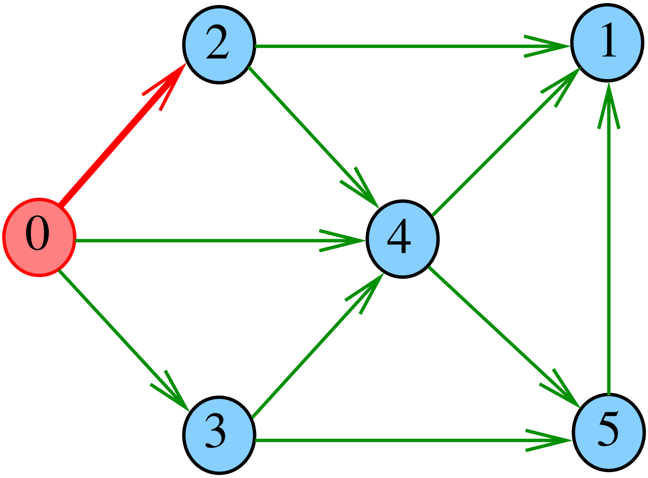
DFSpaths(G, 0)



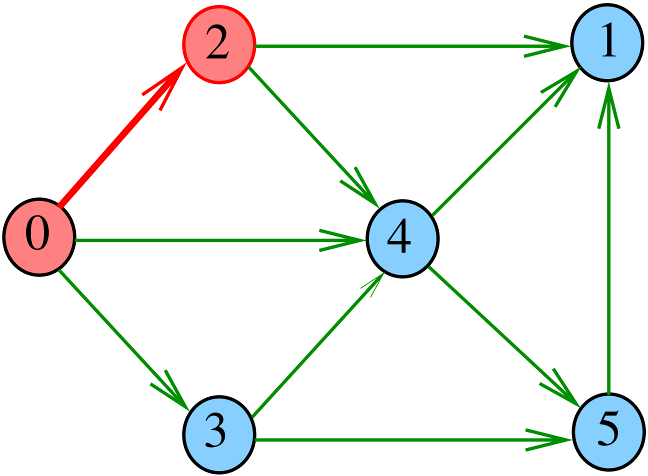
dfs(G, 0)



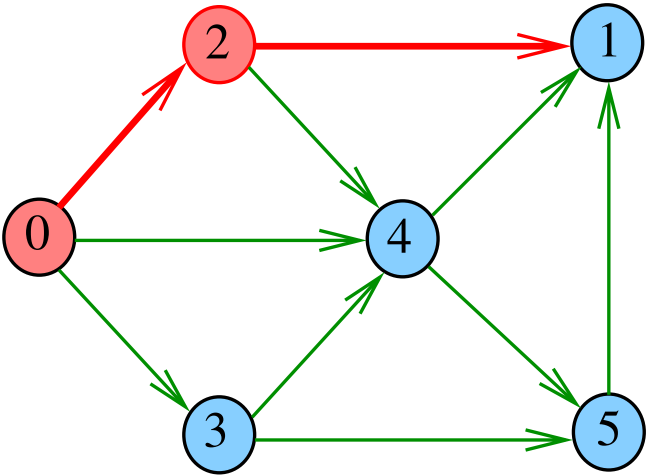
dfs(G, 0)



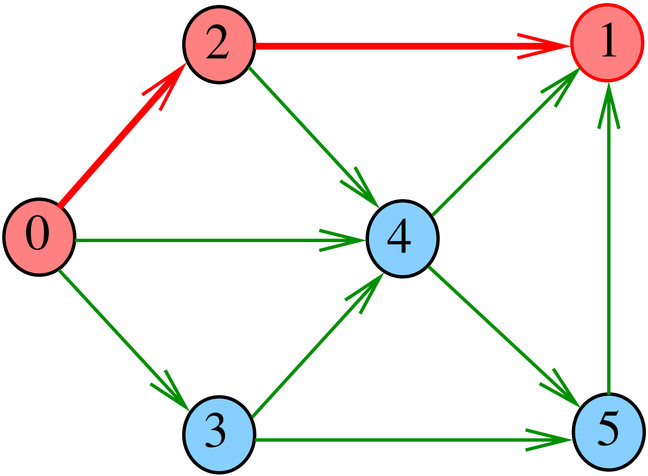
dfs(G, 2)



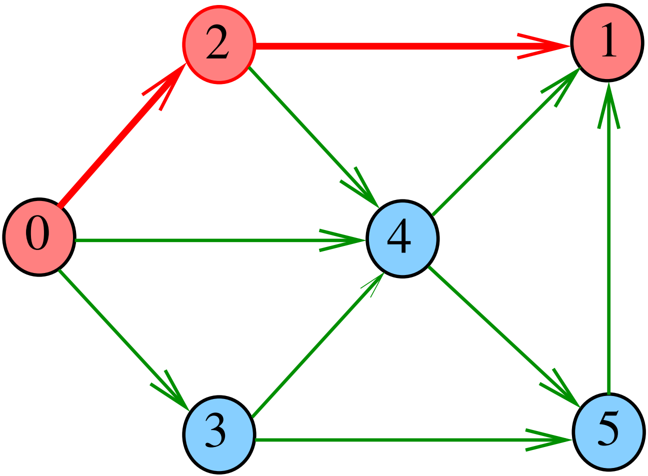
`dfs(G, 2)`



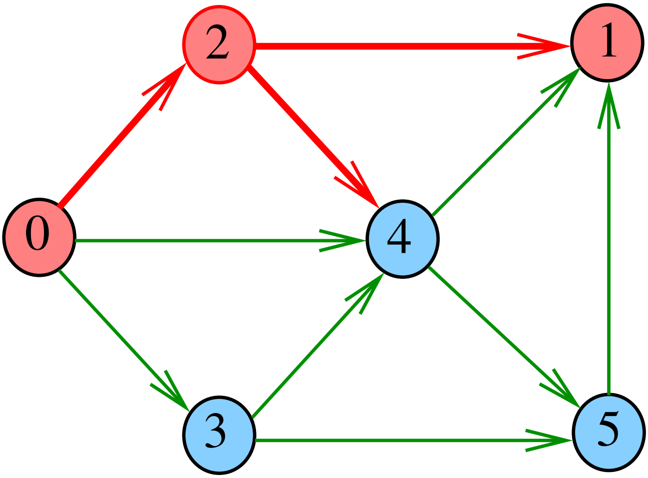
dfs(G, 1)



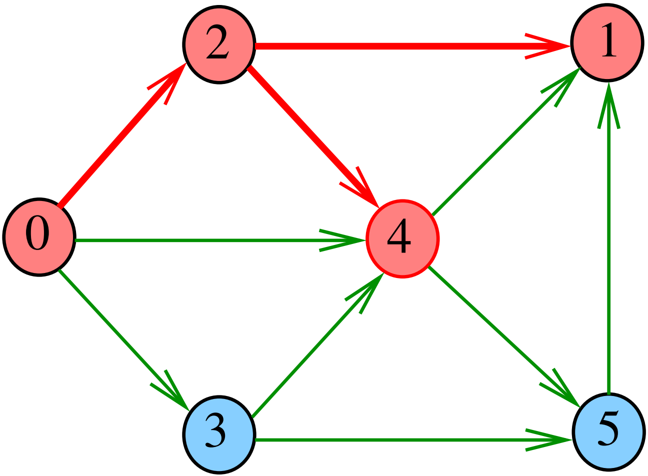
dfs(G, 2)



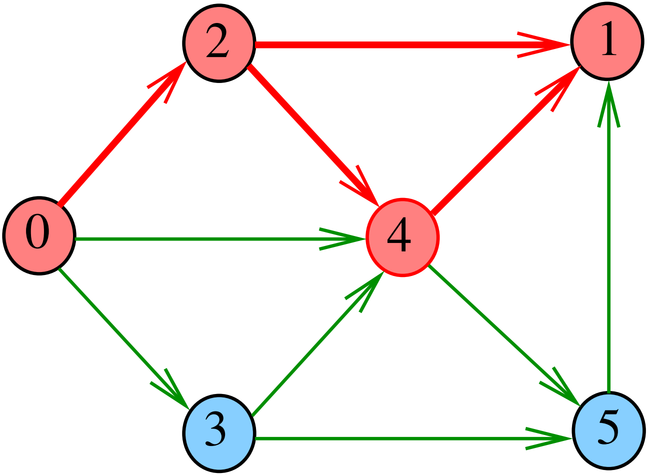
dfs(G, 2)



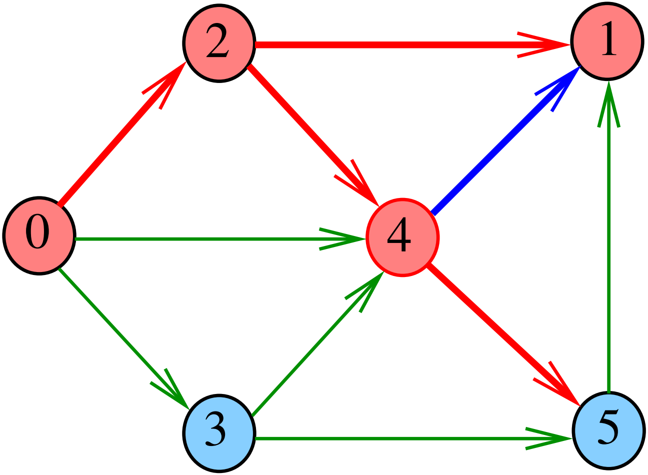
dfs(G, 4)



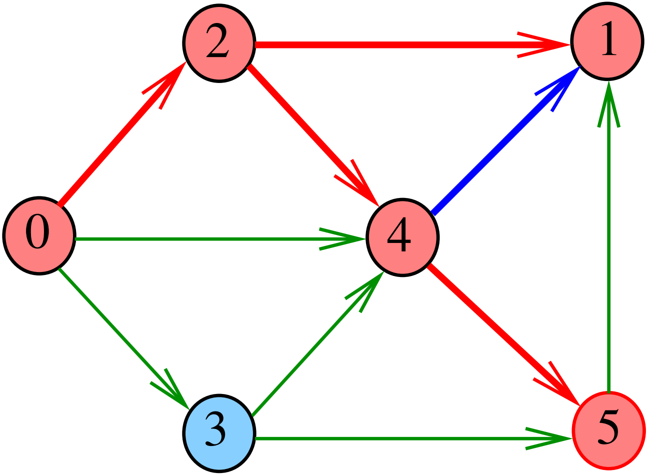
`dfs(G, 4)`



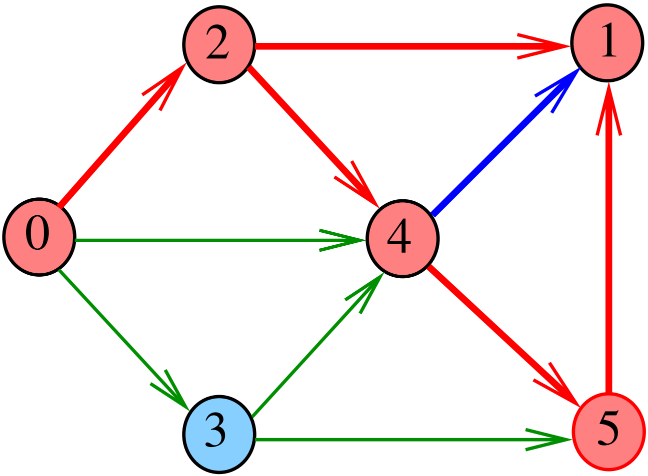
dfs(G, 4)



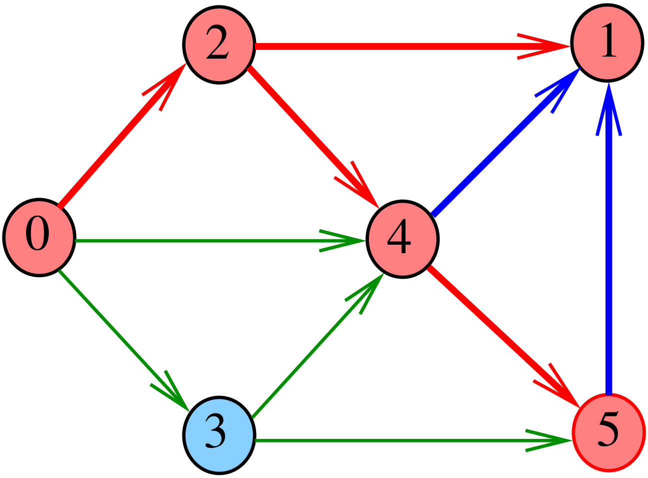
dfs(G, 5)



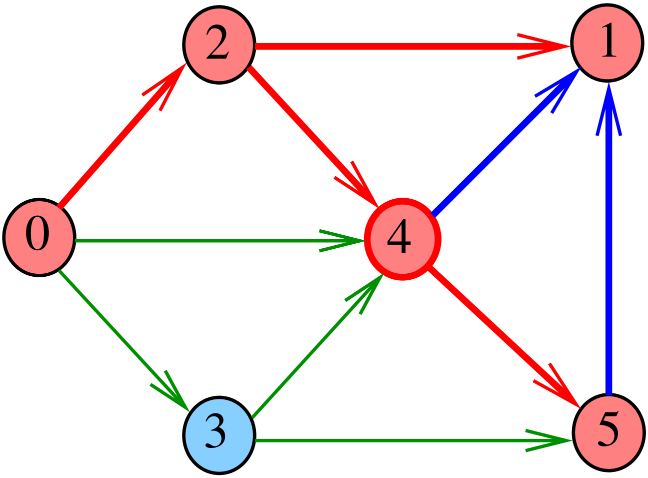
dfs(G, 5)



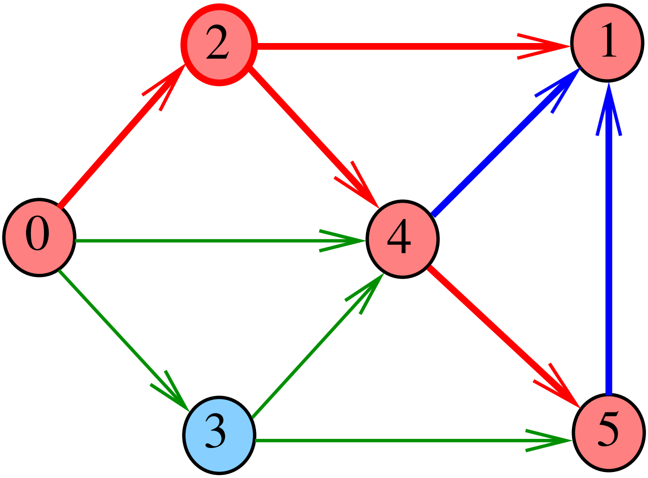
`dfs(G, 5)`



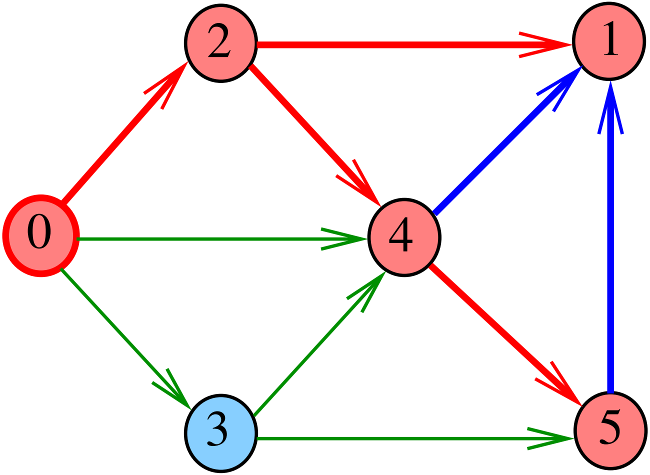
dfs(G, 4)



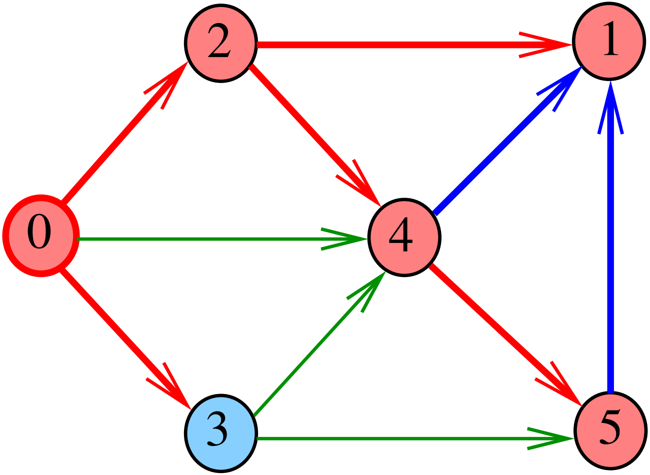
dfs(G, 2)



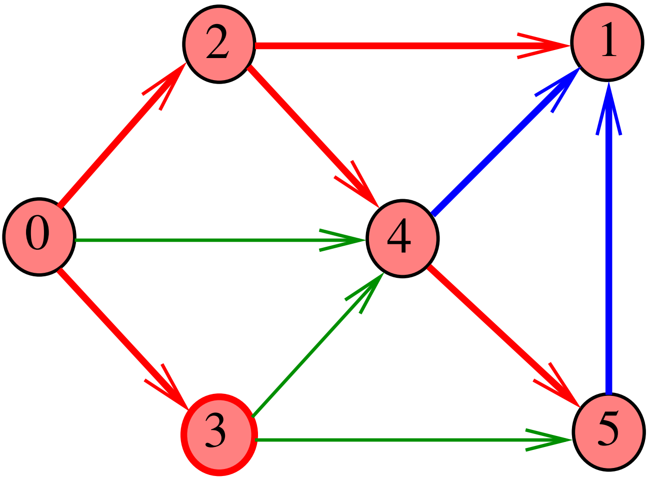
dfs(G, 0)



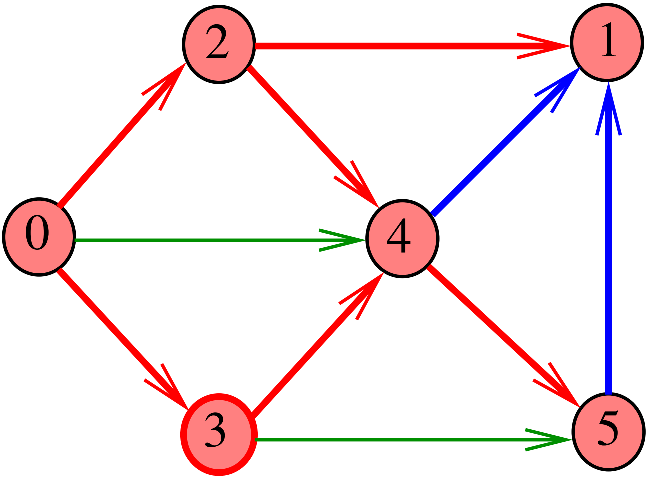
dfs(G, 0)



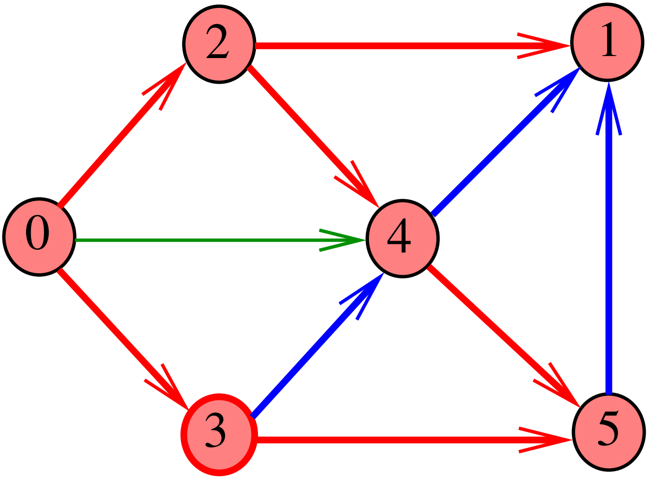
dfs(G, 3)



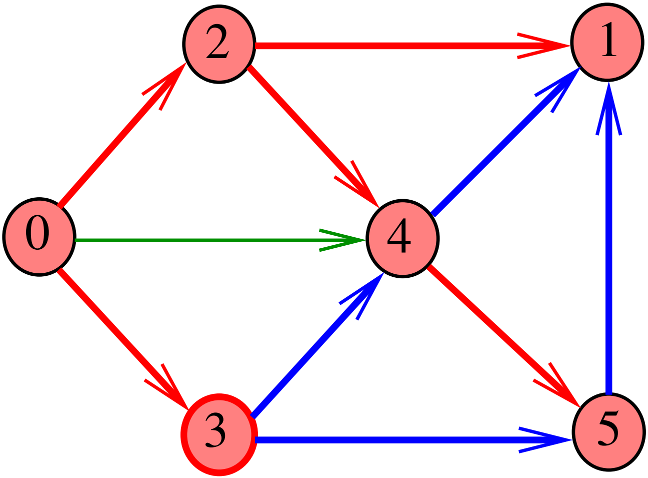
dfs(G, 3)



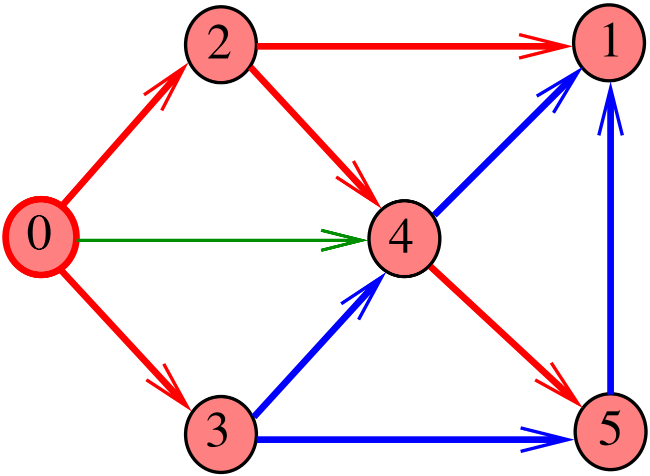
dfs(G, 3)



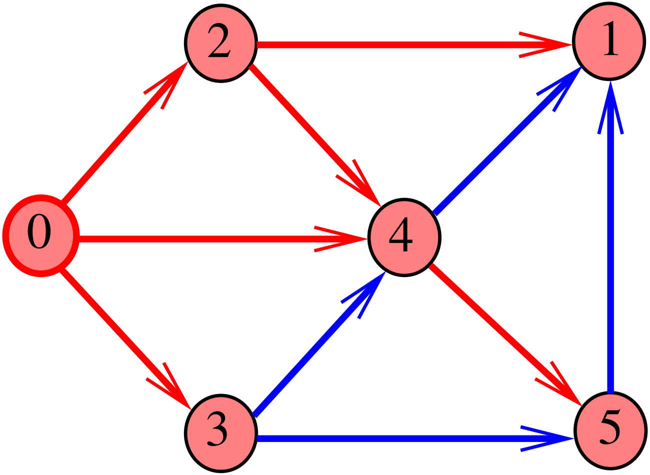
dfs(G, 3)



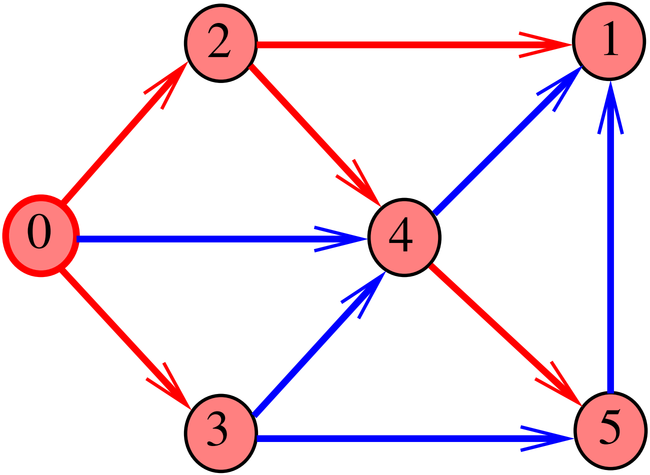
`dfs(G, 0)`



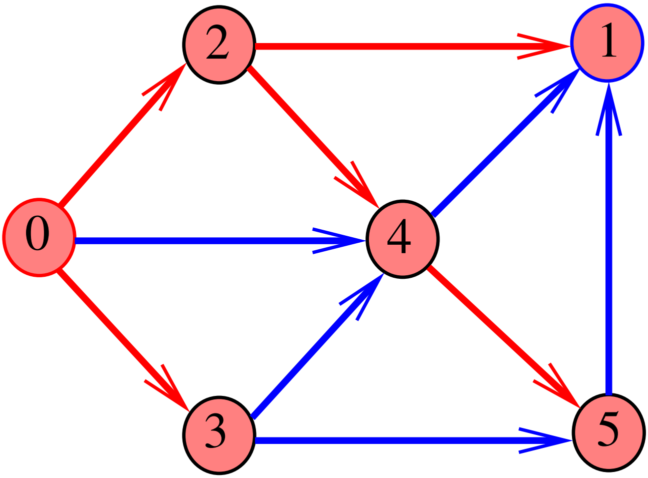
dfs(G, 0)



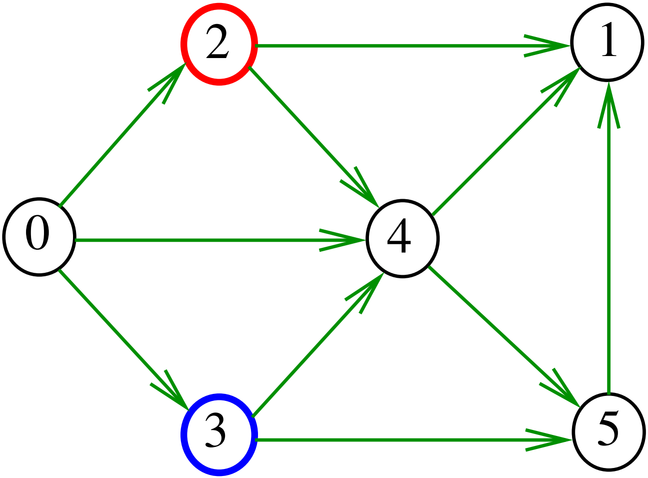
dfs(G, 0)



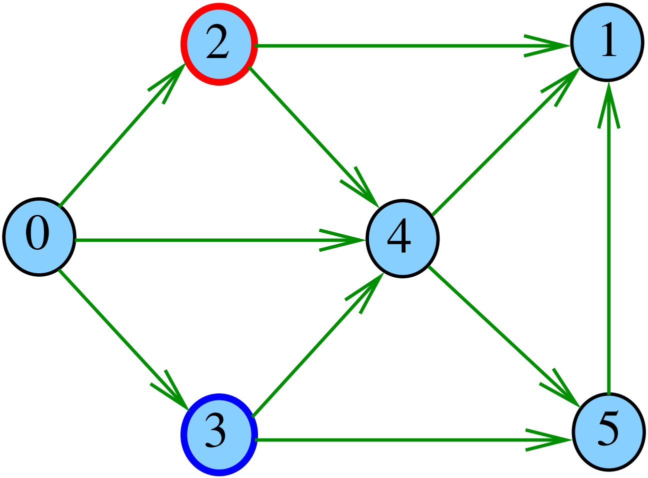
DFSpaths(G, 0)



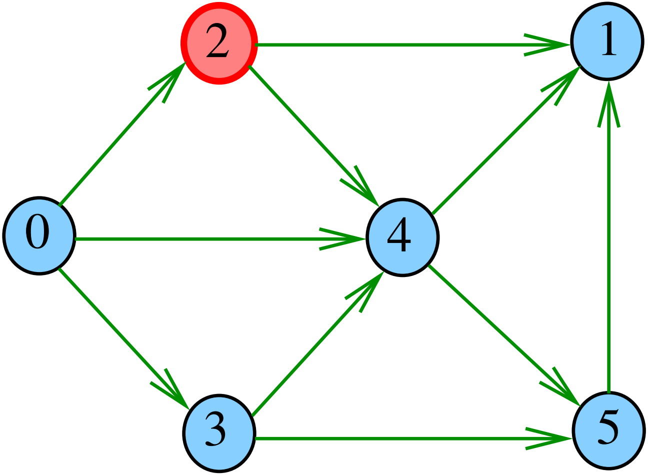
DFSpaths(G, 2)



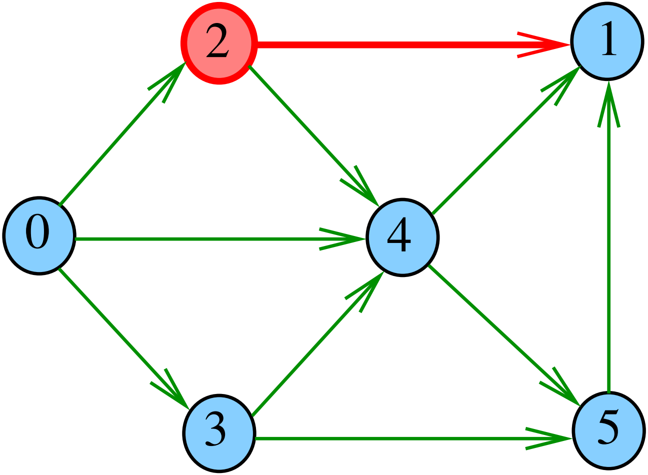
DFSpaths(G, 2)



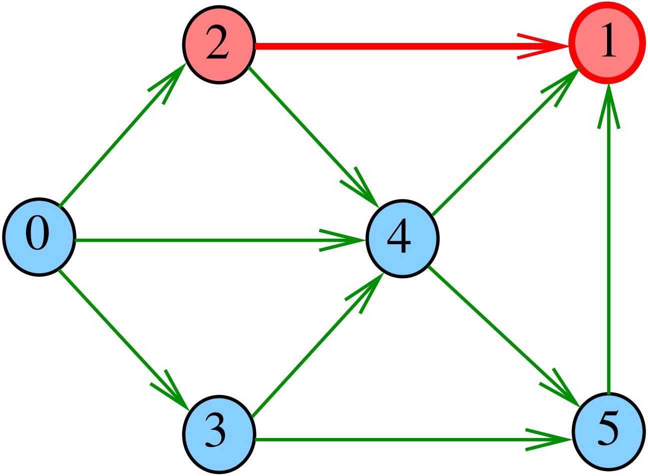
dfs(G, 2)



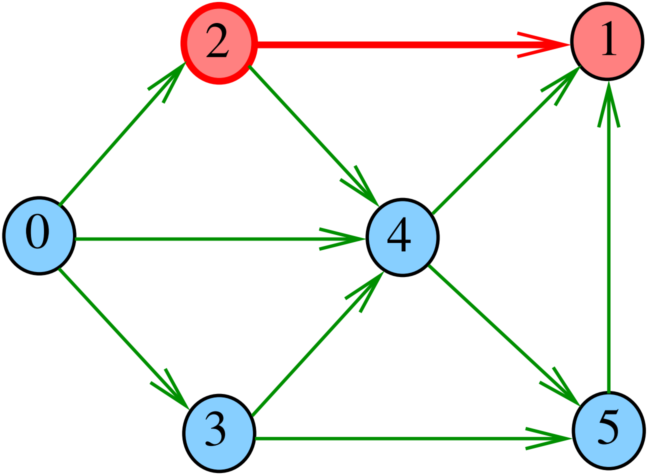
dfs(G, 2)



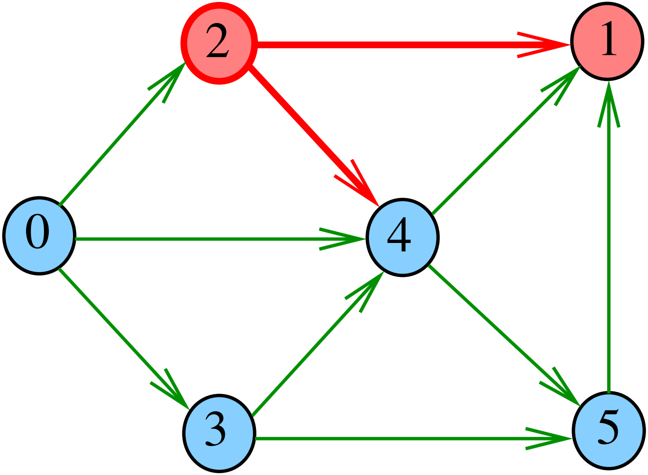
dfs(G, 1)



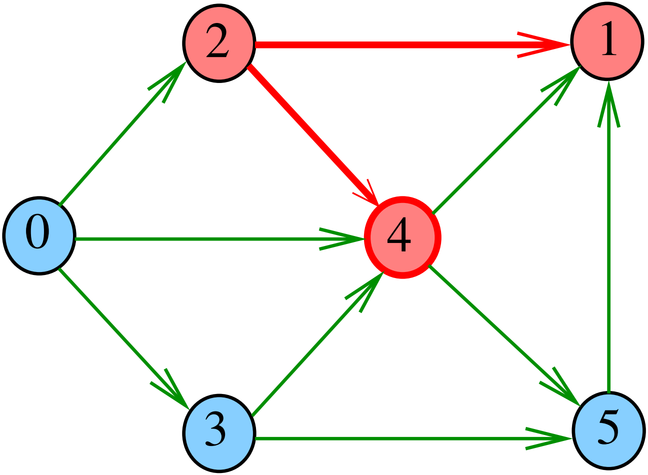
dfs(G, 2)



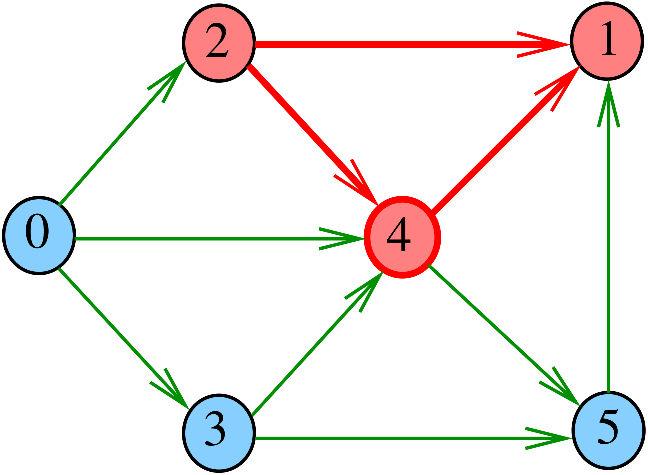
dfs(G, 2)



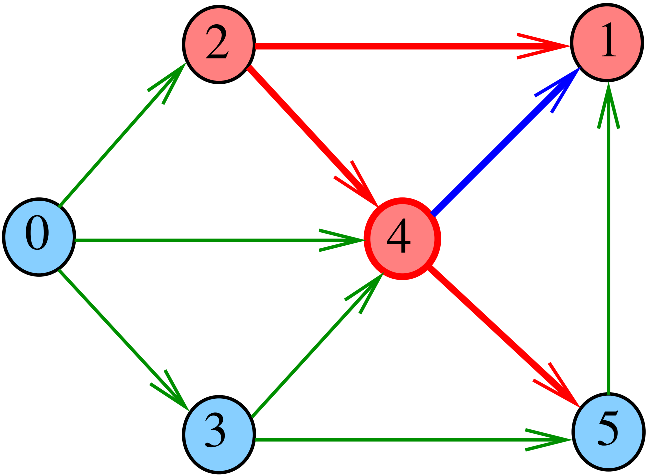
dfs(G, 4)



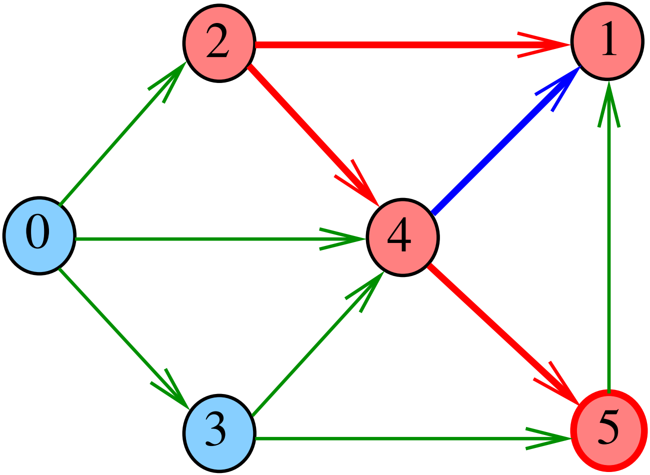
dfs(G, 4)



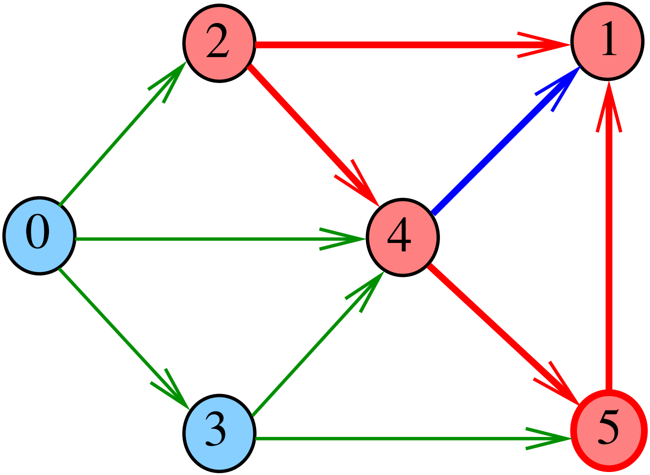
dfs(G, 4)



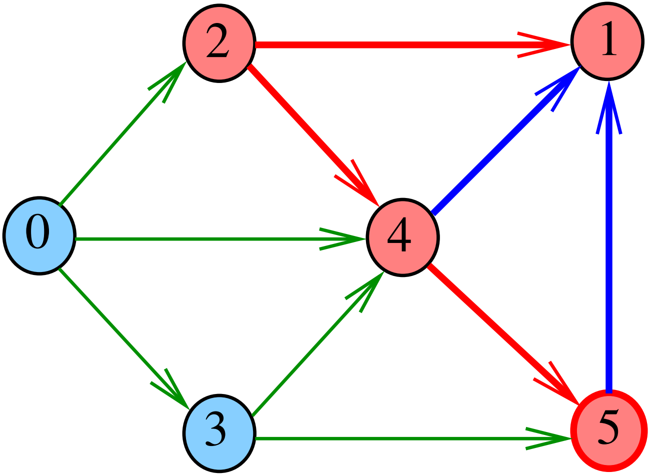
dfs(G, 5)



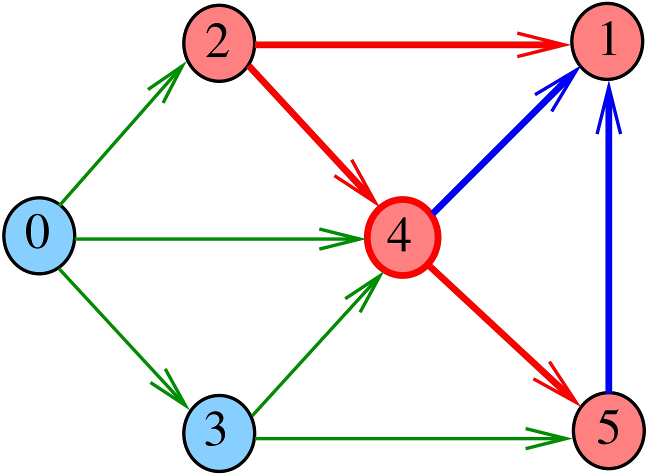
dfs(G, 5)



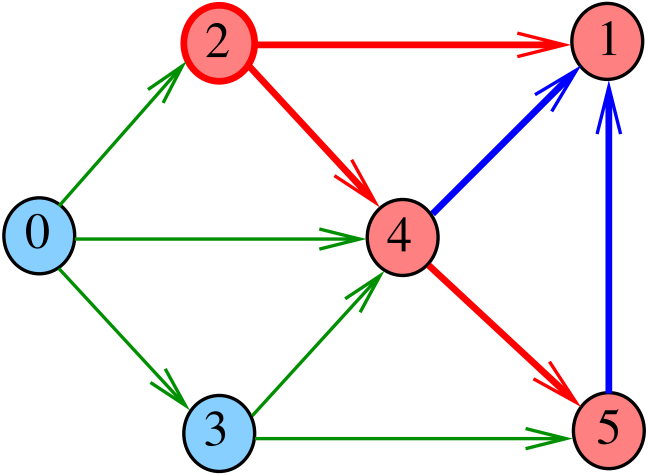
dfs(G, 5)



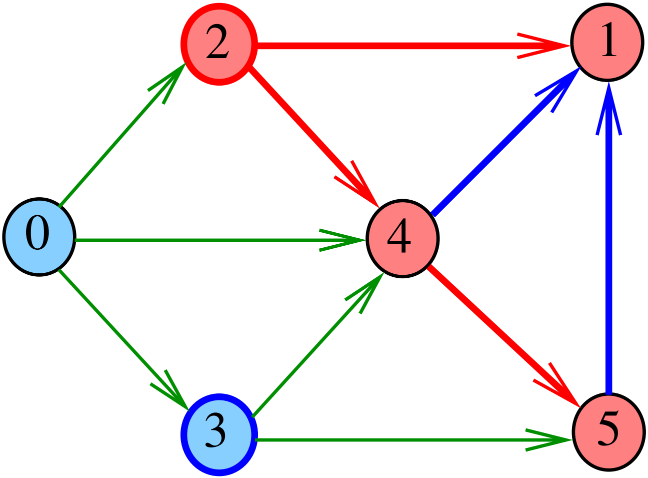
dfs(G, 4)



dfs(G, 2)



DFSpaths(G, 2)



DFSpaths

```
public class DFSpaths {
    private final int s;
    private boolean[] marked;
    public DFSpaths(Digraph G, int s) {}
    private void dfs(Digraph G, int v) {}
    public boolean hasPath(int v) {}
}
```

DFSpaths

Encontra um caminho de s a todo vértice alcançável a partir de s.

```
public DFSpaths(Digraph G, int s) {
   marked = new boolean[G.V()];
   this.s = s;
   dfs(G, s);
}
```

DFSpaths

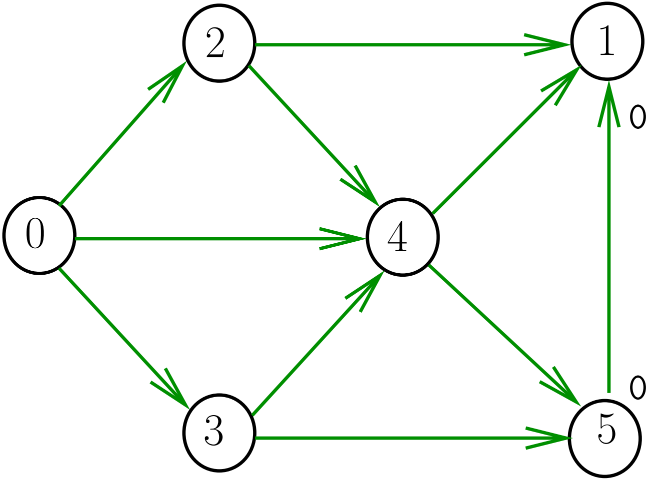
```
private void dfs(Digraph G, int v) {
   marked[v] = true;
   for (int w :  G.adj(v)) {
       if (!marked[w]) {
           dfs(G, w);
       }
   }
}
```

DFSpaths

Há um caminho de s a v?

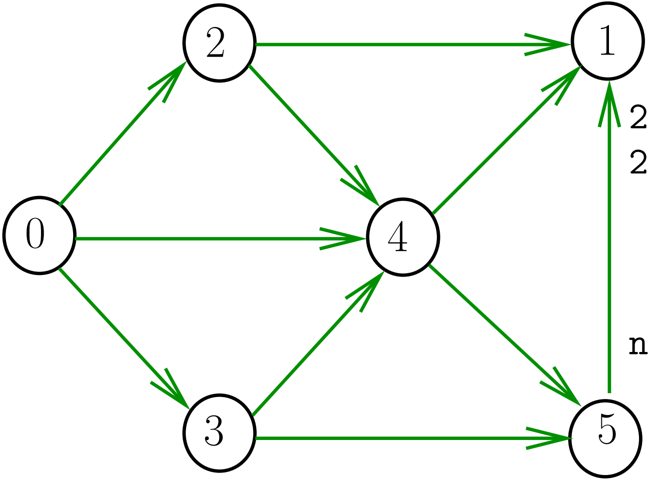
```
public boolean hasPath(int v) {
   return marked[v];
}
```

DFSpaths(G, 0)




```
0-2 dfs(G,2)
  2-1 dfs(G,1)
  2-4 dfs(G,4)
    4-1
    4-5 dfs(G,5)
      5-1
0-3 dfs(G,3)
  3-4
  3-5
0-4
existe caminho
```

DFSpaths(G, 2)




```
2-1 dfs(G,1)
2-4 dfs(G,4)
  4-1
  4-5 dfs(G,5)
    5-1
nao existe caminh
```

Consumo de tempo

Qual é o consumo de tempo de `DFSpaths`?

Consumo de tempo

Qual é o consumo de tempo de `DFSpaths`?

Qual é o consumo de tempo da função `dfs`?

Conclusão

O consumo de tempo de `DFSpaths` é $\Theta(V)$ mais o consumo de tempo da função `dfs()`.

Conclusão

O consumo de tempo da função `dfs()` para vetor de listas de adjacência é $O(V + E)$.

O consumo de tempo de `DFSpaths` para vetor de listas de adjacência é $\sim O(V + E)$.

Conclusão

O consumo de tempo da função `dfs()` para matriz de adjacências é $O(V^2)$.

O consumo de tempo de `DFSpaths` para matriz de adjacências é $O(V^2)$.