Algumas convenções

Grafo G

Sage

Algumas convenções

Grafo G		Sage
$G.V$	cj. dos vértices	`G.vertices()`

Algumas convenções

Grafo G		Sage
$G.V$	cj. dos vértices	`G.vertices()`
$G.v.out$	vs. apontados por v	`G.neighbors_out(v)`

Algumas convenções

Grafo G		Sage
$G.V$	cj. dos vértices	`G.vertices()`
$G.v.out$	vs. apontados por v	`G.neighbors_out(v)`
$G.v.in$	vs. apontando v	`G.neighbors_in(v)`

Algumas convenções

Grafo G		Sage
$G.V$	cj. dos vértices	`G.vertices()`
$G.v.out$	vs. apontados por v	`G.neighbors_out(v)`
$G.v.in$	vs. apontando v	`G.neighbors_in(v)`
$G.v.adj$	vs. adjacentes a v	`G.neighbors(v)`

Algumas convenções

Grafo G		Sage
$G.V$	cj. dos vértices	`G.vertices()`
$G.v.out$	vs. apontados por v	`G.neighbors_out(v)`
$G.v.in$	vs. apontando v	`G.neighbors_in(v)`
$G.v.adj$	vs. adjacentes a v	`G.neighbors(v)`

Em fórmulas, sempre que G estiver claro:

n é o número de vértices

Algumas convenções

Grafo G		Sage
$G.V$	cj. dos vértices	`G.vertices()`
$G.v.out$	vs. apontados por v	`G.neighbors_out(v)`
$G.v.in$	vs. apontando v	`G.neighbors_in(v)`
$G.v.adj$	vs. adjacentes a v	`G.neighbors(v)`

Em fórmulas, sempre que G estiver claro:

n é o número de vértices

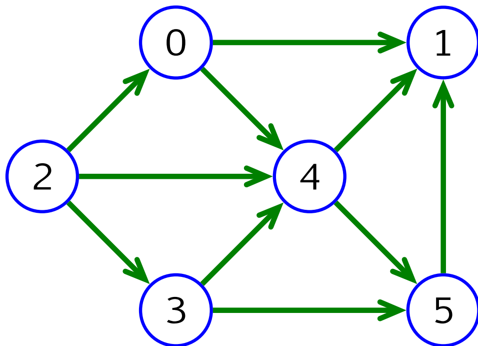
m é o número de arestas ou arcos

Acessibilidade

Acessibilidade

Num digrafo G dizemos que o vértice v é acessível a partir do vértice u se existe caminho dirigido de u para v. Equivalentemente, u alcança v. Notação: $u \hookrightarrow v$.

1 é acessível a partir de 2
$2 \hookrightarrow 1$
5 não alcança 2
$5 \not\hookrightarrow 2$



Acessibilidade

Acessibilidade

A relação de acessibilidade é:

1. *Reflexiva* (trivial)

Acessibilidade

A relação de acessibilidade é:

1. *Reflexiva* (trivial)
2. *Transitiva* (é só grudar caminhos)

Acessibilidade

A relação de acessibilidade é:

1. *Reflexiva* (trivial)
2. *Transitiva* (é só grudar caminhos)

Acessibilidade

A relação de acessibilidade é:

1. *Reflexiva* (trivial)
2. *Transitiva* (é só grudar caminhos)

Uma relação assim é chamada pré-ordem.

Acessibilidade

A relação de acessibilidade é:

1. *Reflexiva* (trivial)
2. *Transitiva* (é só grudar caminhos)

Uma relação assim é chamada pré-ordem.

Problema

Dados u, v, descobrir se v é acessível a partir de u.

Buscas

Buscas

Um esquema de busca (ou varredura) examina, sistematicamente, todos os vértices e todos os arcos de um digrafo.

Buscas

Um esquema de busca (ou varredura) examina, sistematicamente, todos os vértices e todos os arcos de um digrafo.

Cada arco é examinado **uma só vez**.
Depois de visitar sua ponta inicial o algoritmo percorre o arco e visita sua ponta final.

Buscas

Um esquema de busca (ou varredura) examina, sistematicamente, todos os vértices e todos os arcos de um digrafo.

Cada arco é examinado **uma só vez**. Depois de visitar sua ponta inicial o algoritmo percorre o arco e visita sua ponta final.

Esquema: a idéia é colher informação e executar ações durante a busca.

Busca em profundidade (DFS)

Busca em profundidade (DFS)

Nela, cada vértice tem três estados possíveis:

Busca em profundidade (DFS)

Nela, cada vértice tem três estados possíveis:

Inicial

Busca em profundidade (DFS)

Nela, cada vértice tem três estados possíveis:

Inicial

Descoberto

Busca em profundidade (DFS)

Nela, cada vértice tem três estados possíveis:

Inicial

Descoberto

Finalizado

Busca em profundidade (DFS)

Nela, cada vértice tem três estados possíveis:

Inicial

Descoberto

Finalizado

Busca em profundidade (DFS)

Nela, cada vértice tem três estados possíveis:

- Inicial
- Descoberto
- Finalizado

O esquema é naturalmente recursivo: cada vez que um vértice é descoberto ele se torna o centro das atenções, até ser finalizado.

DFS — driver

```
DFS(G)
    for each vertex u ∈ G.V
        u.state = inicial
        u.sob = NIL
    for each vertex u ∈ G.V
        if u.state == inicial
            DFS-VISIT(G, u)
```

DFS — o suco

```
DFS-VISIT(G, u)
    u.state = descoberto
    // começando a processar u
    for each vertex v ∈ u.out
        if v.state == inicial
            // achou v
            v.sob = u
            // processou o arco u → v
            DFS-VISIT(G, v)
        // else: está revendo v
    u.state = finalizado
    // acabou de processar u
```

A árvore de busca

Proposição

Ao fim da DFS, no subgrafo gerador com os arcos da forma $u.sob \to u$ cada componente é uma árvore orientada a partir da raíz.

A árvore de busca

Proposição

Ao fim da DFS, no subgrafo gerador com os arcos da forma $u.sob \to u$ *cada componente é uma árvore orientada a partir da raíz.*

A árvore de busca

Proposição

Ao fim da DFS, no subgrafo gerador com os arcos da forma $u.sob \to u$ *cada componente é uma árvore orientada a partir da raíz.*

Isso é um invariante do algoritmo!

A árvore de busca

Proposição

Ao fim da DFS, no subgrafo gerador com os arcos da forma $u.sob \rightarrow u$ cada componente é uma árvore orientada a partir da raíz.

Isso é um invariante do algoritmo!

Um grafo assim é normalmente chamado de arborescência.

A árvore de busca

Proposição

Ao fim da DFS, no subgrafo gerador com os arcos da forma $u.sob \to u$ cada componente é uma árvore orientada a partir da raíz.

Isso é um invariante do algoritmo!

Um grafo assim é normalmente chamado de arborescência.
Por preguiça, esta é chamada de

árvore de busca em profundidade.

Marcando tempos

Marcando tempos

Vamos marcar para cada vértice o tempo de descoberta e de fim do processamento.

Marcando tempos

Vamos marcar para cada vértice o tempo de descoberta e de fim do processamento.

O Sage só marca o tempo de descoberta

Marcando tempos

Vamos marcar para cada vértice o tempo de descoberta e de fim do processamento.

O Sage só marca o tempo de descoberta

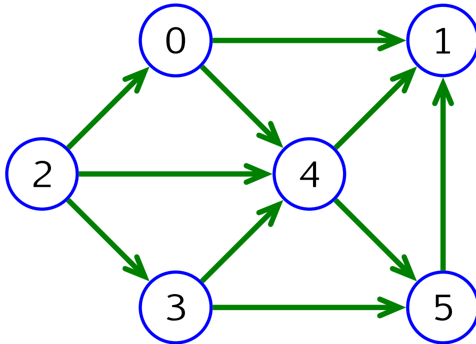
DFS(G)

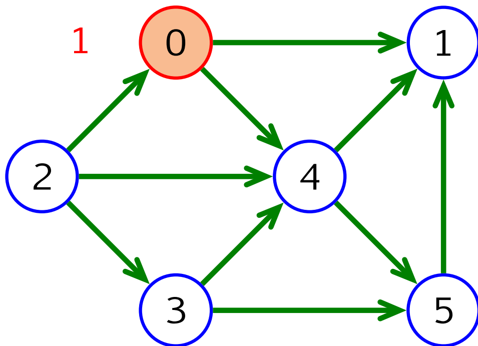
1 **for** each vertex $u \in G.V$
2 $\quad u.state = inicial$
3 $\quad u.sob =$ NIL
4 $time = 0$
5 **for** each vertex $u \in G.V$
6 $\quad$ **if** $u.state == inicial$
7 $\quad\quad$ DFS-VISIT(G, u)

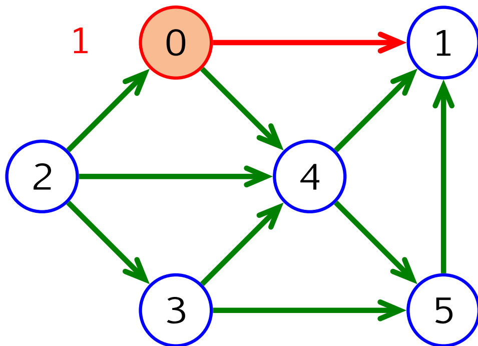
A parte recursiva

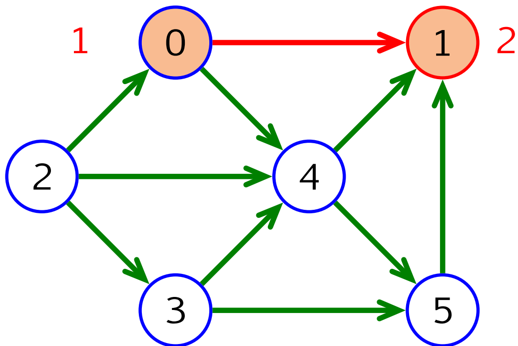
DFS-VISIT(G, u)

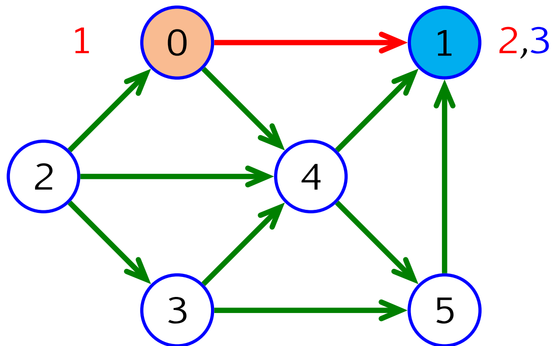
```
 1  u.state = descoberto
 2  u.d = ++time
 3  for each vertex v ∈ u.out
 4          // processando o arco u → v
 5          if v.state == inicial
 6                  // achou v
 7                  v.sob = u
 8                  DFS-VISIT(G, v)
 9          // else: está revendo v
10  u.state = finalizado
11  u.f = time += 1
```

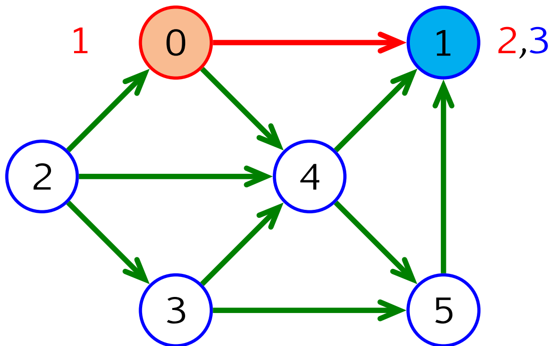

0
1
2
4
3
5

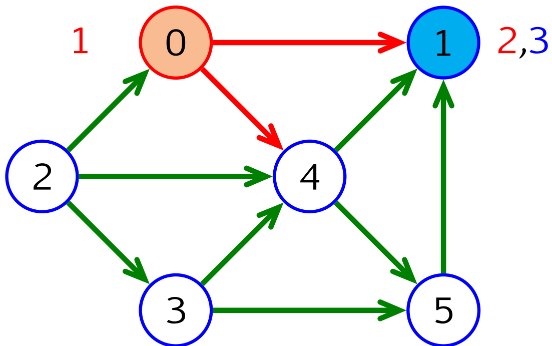

1
0
1
2
4
3
5

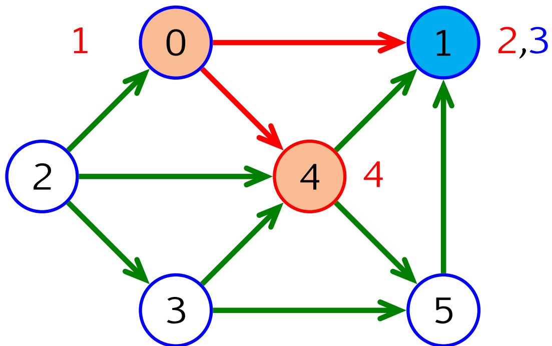

1
0
1
2
4
3
5


1
0
1
2
2
4
3
5

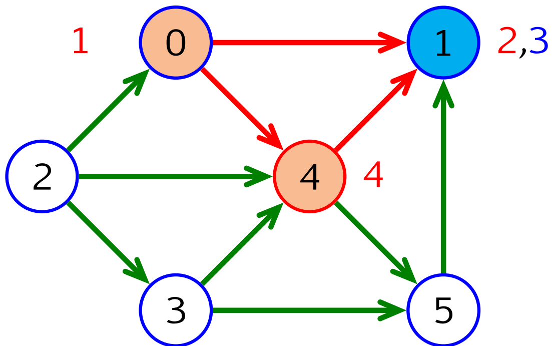

1
0
1
2,3
2
4
3
5

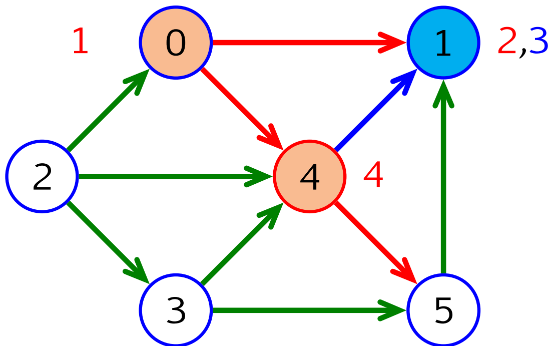

1
0
1
2,3
2
4
3
5


1
0
1
2,3
2
4
3
5

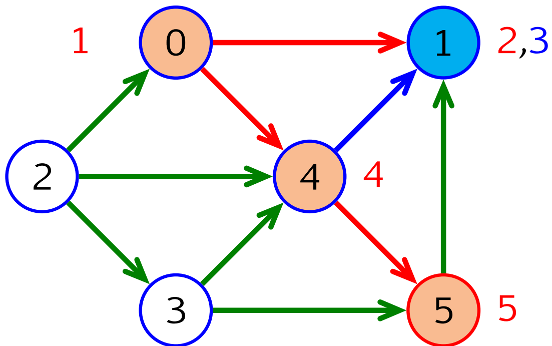


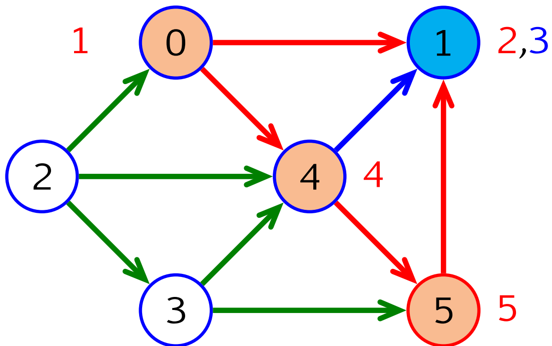
1
0
1
2,3
2
4
4
3
5

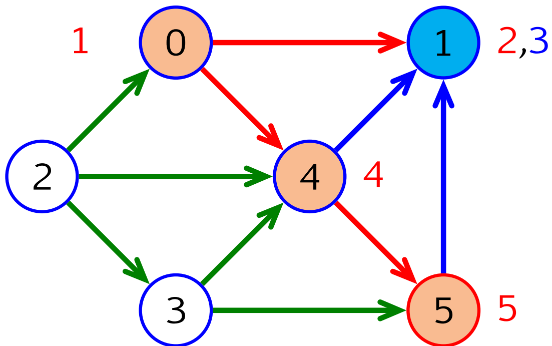

1
0
1
2,3
2
4
4
3
5



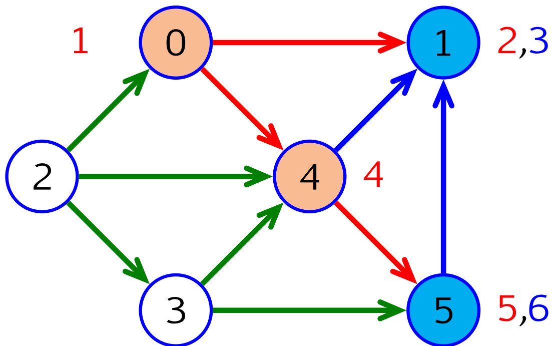
1
0
1
2,3
2
4
4
3
5

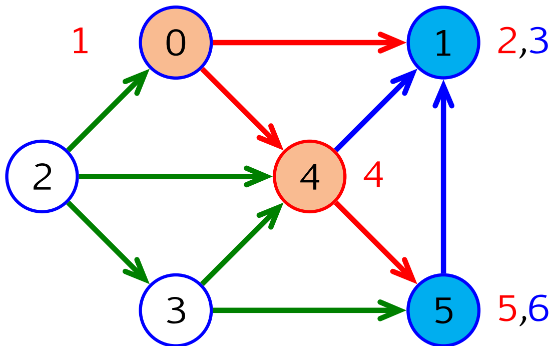

1
0
1
2,3
2
4
4
3
5
5


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

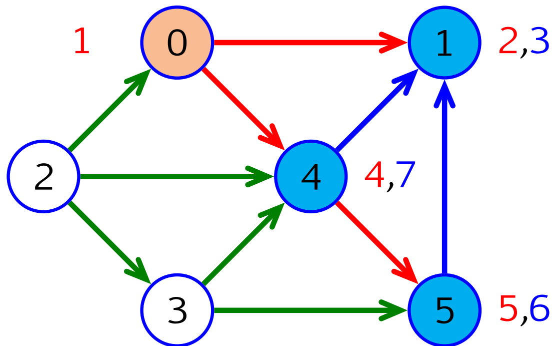


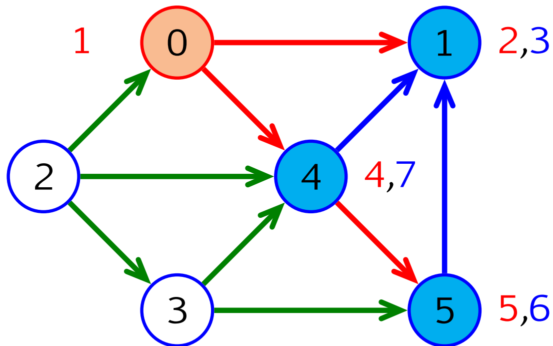
0
1
2
3
4
5
1
2,3
4
5

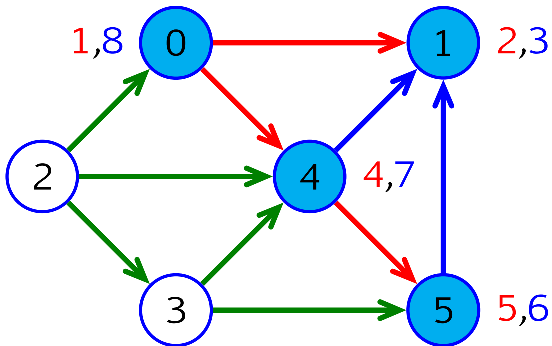

1
0
1
2,3
2
4
4
3
5
5,6

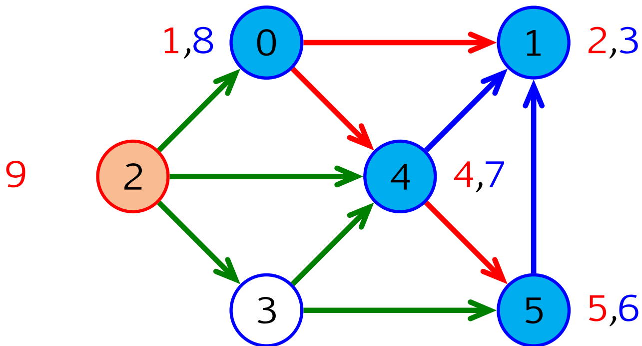


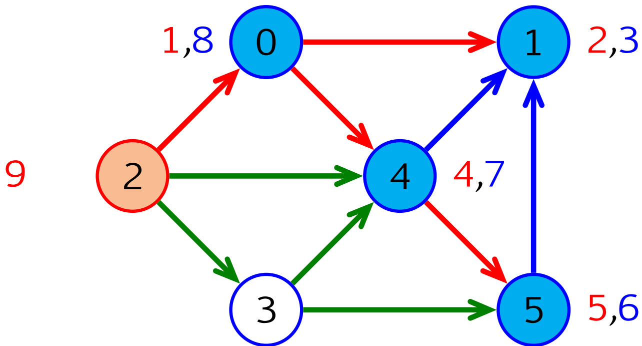
0
1
2
3
4
5
1
2,3
4
5,6

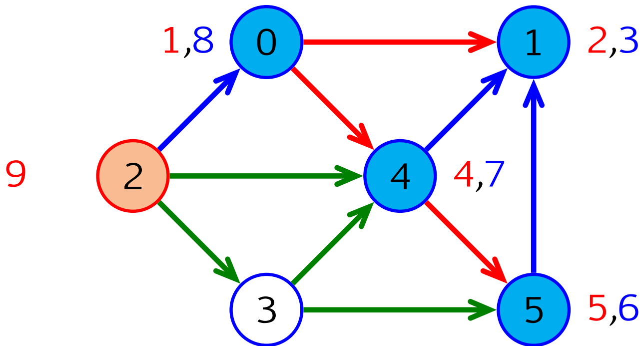

1
0
1
2,3
2
4
4,7
3
5
5,6

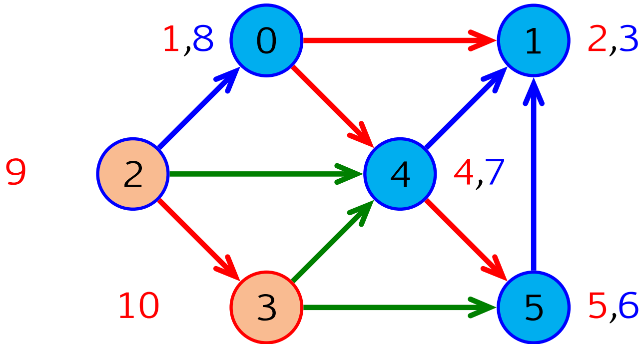

1
0
1
2,3
2
4
4,7
3
5
5,6

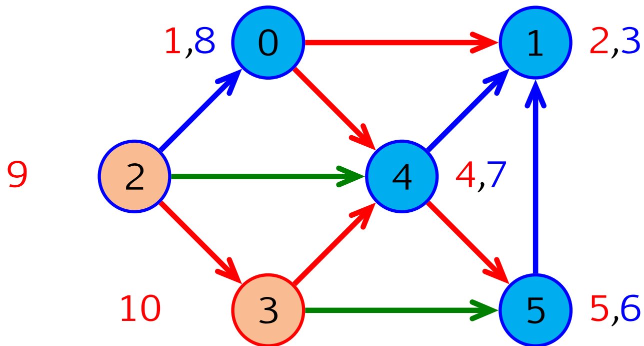

1,8
0
1
2,3
2
4
4,7
3
5
5,6

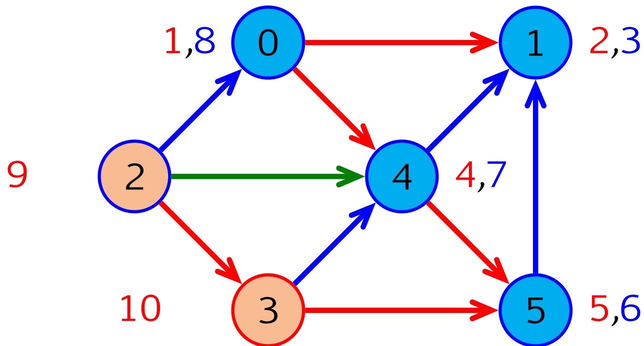

1,8
0
1
2,3
9
2
4
4,7
3
5
5,6


1,8
0
1
2,3
9
2
4
4,7
3
5
5,6

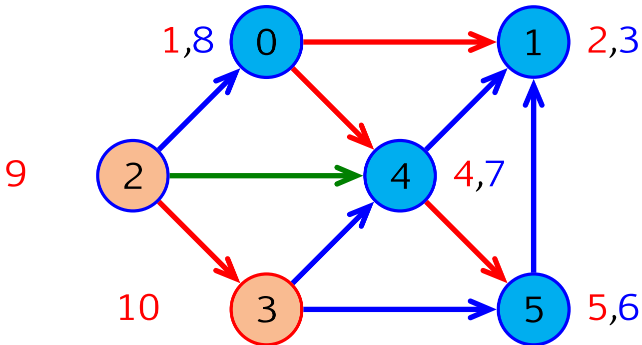

1,8
0
1
2,3
9
2
4
4,7
3
5
5,6

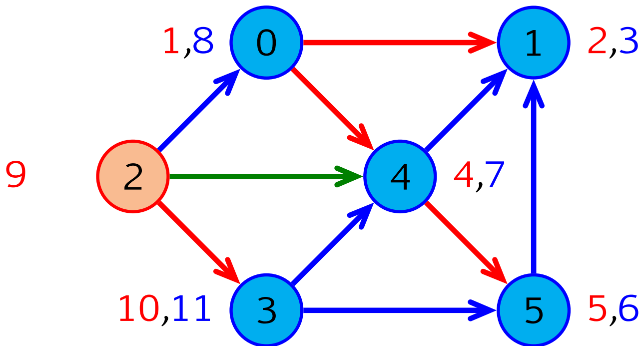

1,8
0
1
2,3
9
2
4
4,7
10
3
5
5,6

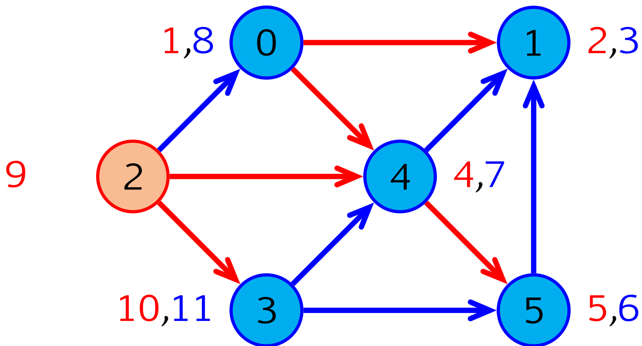

0
1
2
3
4
5
1,8
2,3
9
4,7
10
5,6

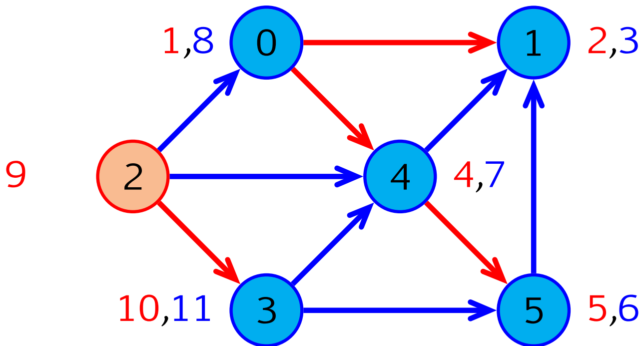


0
1
2
3
4
5
1,8
2,3
9
4,7
10
5,6

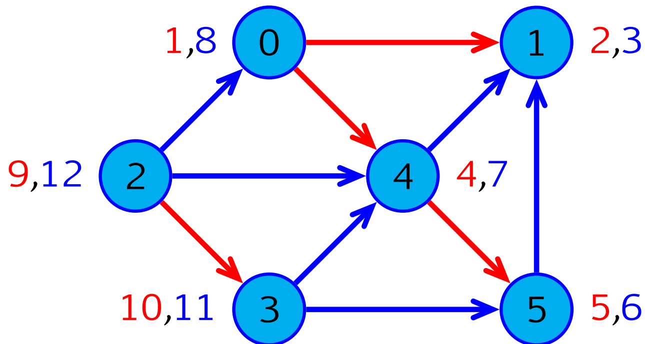

1,8
0
1
2,3
9
2
4
4,7
10
3
5
5,6

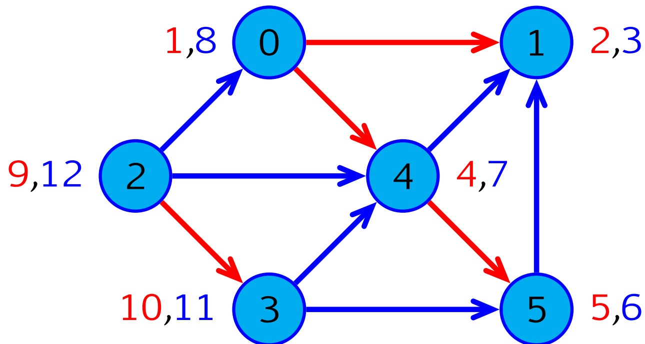

1,8
0
1
2,3
9
2
4
4,7
10,11
3
5
5,6


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



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


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



FIM!

Quem está abaixo de um vértice

Proposição

Se v é acessível a partir de u e u foi descoberto antes de v ($u.d < v.d$), então v é acessível a partir de u na árvore de busca.

Quem está abaixo de um vértice

Proposição

Se v é acessível a partir de u e u foi descoberto antes de v ($u.d < v.d$), então v é acessível a partir de u na árvore de busca.

Quem está abaixo de um vértice

Proposição

Se v é acessível a partir de u e u foi descoberto antes de v ($u.d < v.d$), então v é acessível a partir de u na árvore de busca.

Corolário

Se a busca em profundidade começa em u, quando este termina de ser processado os vértices na árvore de busca são exatamente aqueles acessíveis a partir de u.

Complexidade

Se a iteração pelos arcos saindo de um vértice tiver tempo linear no tamanho do conjunto, então o tempo da DFS é

$$\mathcal{O}(n + m)$$

Complexidade

Se a iteração pelos arcos saindo de um vértice tiver tempo linear no tamanho do conjunto, então o tempo da DFS é

$$\mathcal{O}(n + m)$$

PROVA: Cada arco (u, v) é examinado uma única vez.

Propriedades da numeração

Sejam u, v vértices, com $u.d < v.d$. Então:

Ou $u.d < u.f < v.d < v.f$

Propriedades da numeração

Sejam u, v vértices, com $u.d < v.d$. Então:

Ou $u.d < u.f < v.d < v.f$

ou $u.d < v.d < v.f < u.f$.

Propriedades da numeração

Sejam u, v vértices, com $u.d < v.d$. Então:

Ou $u.d < u.f < v.d < v.f$

ou $u.d < v.d < v.f < u.f$.

Propriedades da numeração

Sejam u, v vértices, com $u.d < v.d$. Então:

Ou $u.d < u.f < v.d < v.f$

ou $u.d < v.d < v.f < u.f$.

Intervalos da forma $[d, f]$ *são disjuntos ou encaixados.*

Uma classificação das arestas

Uma classificação das arestas

Um arco $u \to v$ é:

- da árvore se
  $u.d < v.d < v.f < u.f$ e $v.sob = u$

Uma classificação das arestas

Um arco $u \to v$ é:

- da árvore se
  $u.d < v.d < v.f < u.f$ e $v.sob = u$
- descendente se
  $u.d < v.d < v.f < u.f$ e $v.sob \neq u$

Uma classificação das arestas

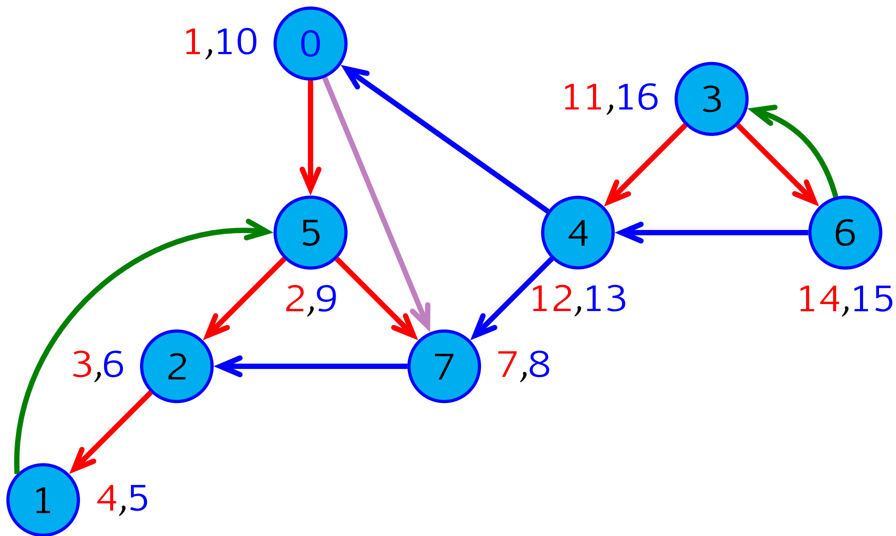
Um arco $u \to v$ é:

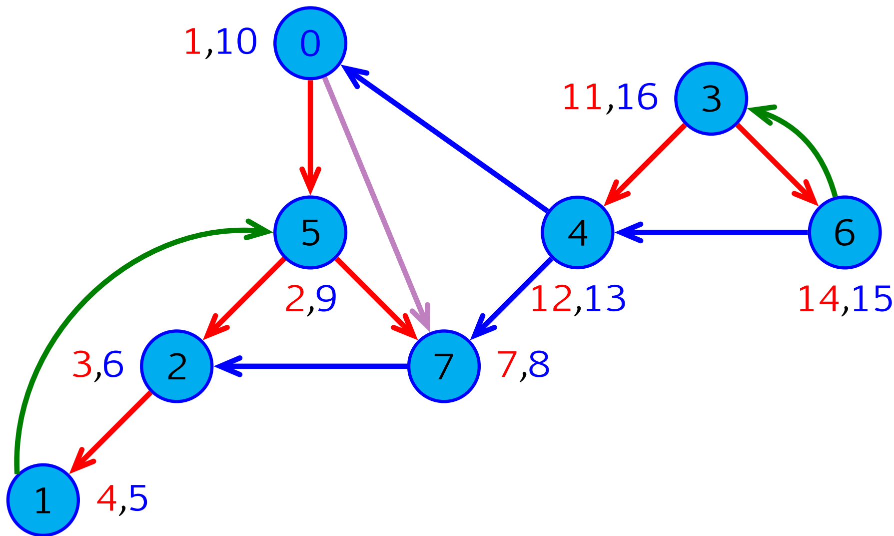
- da árvore se
  $u.d < v.d < v.f < u.f$ e $v.sob = u$
- descendente se
  $u.d < v.d < v.f < u.f$ e $v.sob \neq u$
- de retorno se
  $v.d < u.d < u.f < v.f$

Uma classificação das arestas

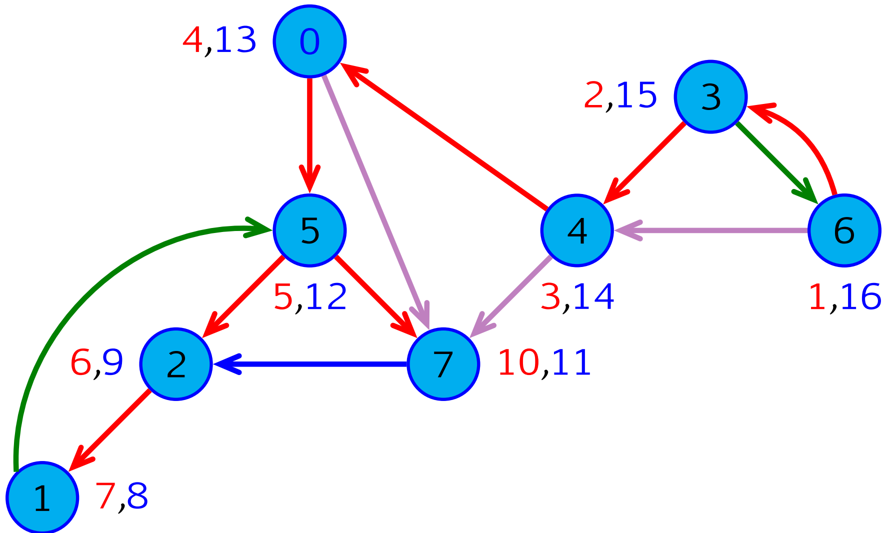
Um arco $u \to v$ é:

- da árvore se
  $u.d < v.d < v.f < u.f$ e $v.sob = u$
- descendente se
  $u.d < v.d < v.f < u.f$ e $v.sob \neq u$
- de retorno se
  $v.d < u.d < u.f < v.f$
- cruzado se
  $v.d < v.f < u.d < u.f$


1,10
0
11,16
3
5
4
6
2,9
12,13
14,15
3,6
2
7
7,8
1
4,5


(((( )) ( ))) (( ) ( ))
0
1,10
3
11,16
5
4
6
2,9
12,13
14,15
2
7
3,6
7,8
1
4,5

Processando a partir do 6



Como classificar?

Como classificar?

1: Percorrendo as arestas depois da DFS.

Como classificar?

1: Percorrendo as arestas depois da DFS.
2: Durante a DFS:

DFS-VISIT(G, u)

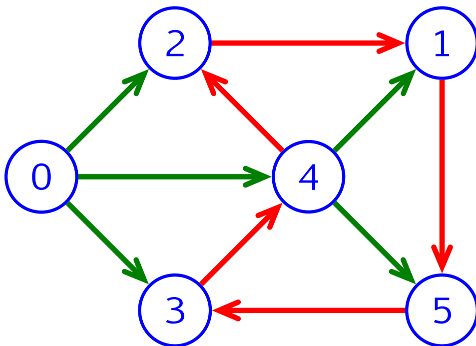
```
 1  u.state = descoberto
 2  u.d = ++time
 3  for each vertex v ∈ u.out
 4        // classifique   u → v
 5        if v.state == inicial
 6              // achou v
 7              v.sob = u
 8              DFS-VISIT(G, v)
 9        // else: está revendo v
10  v.state = finalizado
11  u.f = time += 1
```

Ciclos

Ciclos

Um ciclo num digrafo é uma seqüência da forma $v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_{k-1} \rightarrow v_p$, onde $v_{k-1} \rightarrow v_k$ é um arco $(k = 1, \ldots, p)$ e $v_0 = v_p$.

Exemplo: $2 \rightarrow 1 \rightarrow 5 \rightarrow 3 \rightarrow 4 \rightarrow 2$ é um ciclo



Quem não quer ciclos?

Quem não quer ciclos?

Um uso comum para grafos dirigidos é para descrever *relações de dependência*.

Quem não quer ciclos?

Um uso comum para grafos dirigidos é para descrever *relações de dependência*.

Um ciclo num grafo assim indica problema nas especificações.

Quem não quer ciclos?

Um uso comum para grafos dirigidos é para descrever *relações de dependência*.

Um ciclo num grafo assim indica problema nas especificações.

É bom poder detectar ciclos, se existirem.

Quem não quer ciclos?

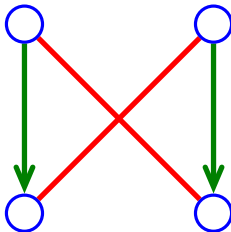
Quem não quer ciclos?

Árvores genealógicas nem sempre são árvores.

Quem não quer ciclos?

Árvores genealógicas nem sempre são árvores.

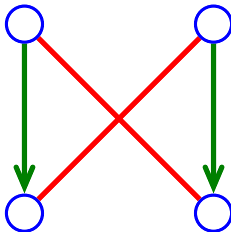
Antigo conto do rei e o filho que encontram uma rainha e a filha e fazem casamentos cruzados.



Quem não quer ciclos?

Árvores genealógicas nem sempre são árvores.

Antigo conto do rei e o filho que encontram uma rainha e a filha e fazem casamentos cruzados.

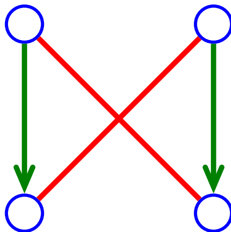


Hierarquias de classes com herança múltipla...

Quem não quer ciclos?

Árvores genealógicas nem sempre são árvores.

Antigo conto do rei e o filho que encontram uma rainha e a filha e fazem casamentos cruzados.



Hierarquias de classes com herança múltipla...

É bom poder detectar ciclos, se existirem.

Exercício

Conto All You Zombies, Robert E. Heinlein, 1958.

Filme: Predestination (2014)

Exercício: montar a árvore genealógica.

Procurando um ciclo

Procurando um ciclo

PROBLEMA: Dado um digrafo G, ele tem um ciclo?

Procurando um ciclo

PROBLEMA: Dado um digrafo G, ele tem um ciclo?

SUBPROBLEMAS:

- Se ele tem, dá para exibir?

Procurando um ciclo

PROBLEMA: Dado um digrafo G, ele tem um ciclo?

SUBPROBLEMAS:

- Se ele tem, dá para exibir?
- Se não tem, como convenço alguém disso?

Procurando um ciclo

PROBLEMA: Dado um digrafo G, ele tem um ciclo?

SUBPROBLEMAS:

- Se ele tem, dá para exibir?
- Se não tem, como convenço alguém disso?

Procurando um ciclo

Problema: Dado um digrafo G, ele tem um ciclo?

Subproblemas:

- Se ele tem, dá para exibir?
- Se não tem, como convenço alguém disso?

Subentendido:

Procurando um ciclo

PROBLEMA: Dado um digrafo G, ele tem um ciclo?

SUBPROBLEMAS:

- Se ele tem, dá para exibir?
- Se não tem, como convenço alguém disso?

Subentendido:

não aceitamos nada que não leve tempo que não seja não-polinomial!

Proposição

Para um digrafo G, são equivalentes:

1. *G tem um ciclo.*
2. *Alguma DFS em G tem aresta de retorno.*
3. *Toda DFS em G tem aresta de retorno.*

Proposição

Para um digrafo G, são equivalentes:

1. *G tem um ciclo.*
2. *Alguma DFS em G tem aresta de retorno.*
3. *Toda DFS em G tem aresta de retorno.*

Proposição

Para um digrafo G, são equivalentes:

1. *G tem um ciclo.*
2. *Alguma DFS em G tem aresta de retorno.*
3. *Toda DFS em G tem aresta de retorno.*

PROVA: $(3) \Rightarrow (2) \Rightarrow (1)$ é óbvio, usando *.sob* para capturar o ciclo.

Proposição

Para um digrafo G, são equivalentes:

1. *G tem um ciclo.*
2. *Alguma DFS em G tem aresta de retorno.*
3. *Toda DFS em G tem aresta de retorno.*

Prova: $(3) \Rightarrow (2) \Rightarrow (1)$ é óbvio, usando *.sob* para capturar o ciclo.
Para $(1) \Rightarrow (3)$, suponha que G tenha um ciclo C. Considere uma DFS; seja u o primeiro vértice descoberto de C e seja v o vértice de C que precede u no ciclo. Como v é acessível de u, ele é descoberto antes de finalizar u, assim $v \to u$ é de retorno. □