Grafos e Algoritmos de Busca

Eduardo Camponogara

Departamento de Automação e Sistemas
Universidade Federal de Santa Catarina

DAS-9003: Introdução a Algoritmos

Sumário

Introdução

Grafos

- Aqui apresentaremos métodos para representar grafos e realizar buscas
- Busca em grafos significa seguir sistematicamente as arestas e visitar os vértices

Introdução

Grafos

Estudaremos três representações de grafos:

- listas de adjacência
- matrizes de adjacência
- matrizes de incidência

Estudaremos também:

- métodos de busca em largura
- métodos de busca em profundidade
- ordenação topológica

Sumário

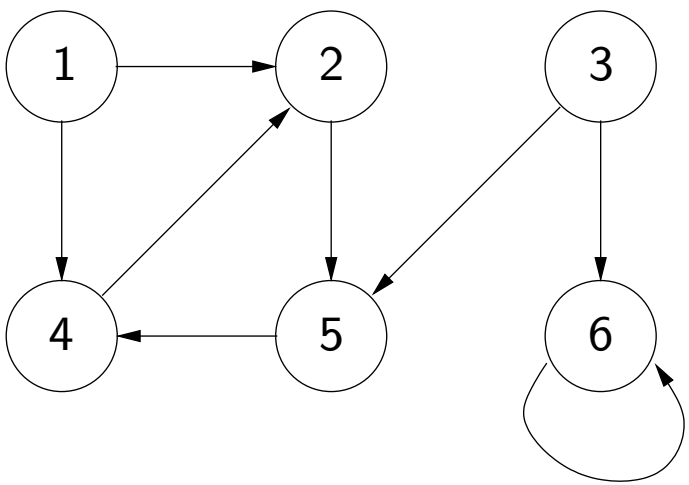
Representação de Grafos

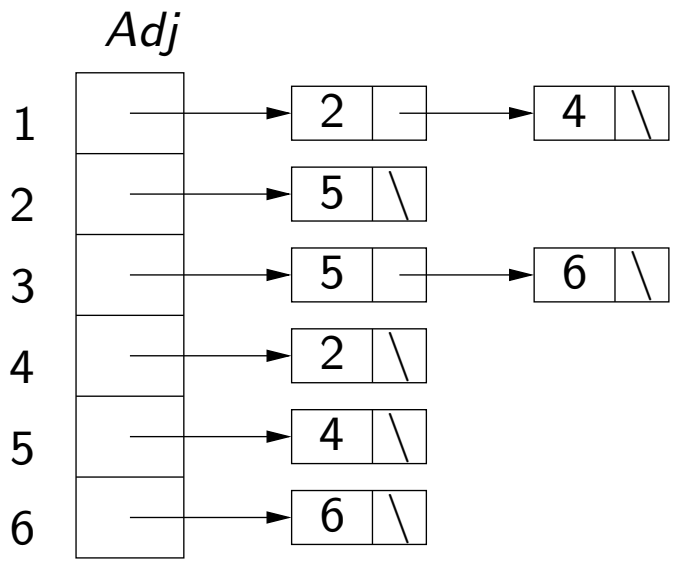
Lista de Adjacência

- Dado um grafo $G = (V, E)$, esta representação é tipicamente preferida pois é uma maneira compacta de representar grafos esparsos – aqueles onde $|E| \ll |V|^2$
- A representação por listas de adjacência consiste em um vetor Adj com $|V|$ listas de adjacência, uma para cada vértice $v \in V$.
- Para cada $u \in V$, $Adj[u]$ contém ponteiros para todos os vértices v tal que $(u, v) \in E$. Ou seja, $Adj[u]$ consiste de todos os vértices que são adjacentes a u

Representação de Grafos

Lista de Adjacência





Representação de Grafos

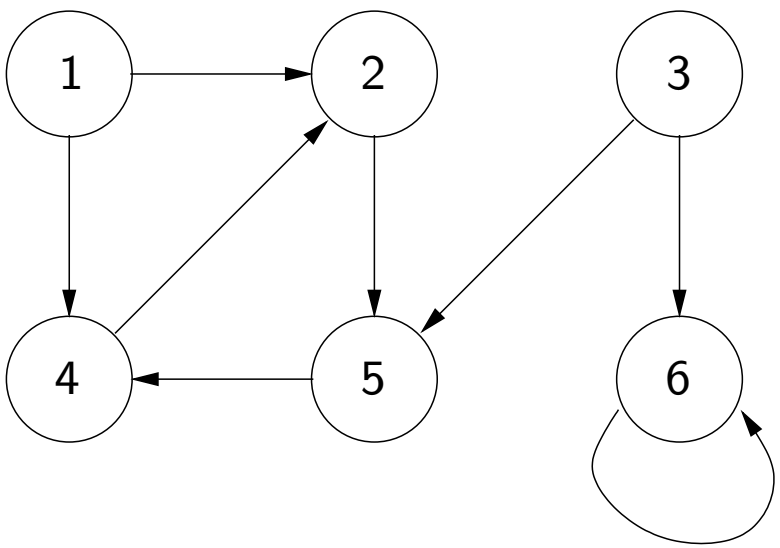
Matriz de Adjacência

- A representação por matriz de adjacência é preferida, entretanto, quando o grafo é denso, ou seja, quando $|E| \approx |V|^2$.
- Para um grafo $G = (V, E)$, assumimos que os vértices são rotulados com números $1, 2, \ldots, |V|$.
- A representação consiste de uma matriz $A = (a_{ij})$ de dimensões $|V| \times |V|$, onde

$$a_{ij} = \begin{cases} 1 & \text{se } (i,j) \in E \\ 0 & \text{caso contrário} \end{cases}$$

Representação de Grafos

Matriz de Adjacência



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

Representação de Grafos

Matriz de incidência

- O grafo direcionado $G = (V, E)$ é representado por uma matriz $A \in \mathbb{R}^{n \times m}$, onde $|V| = n$ e $|E| = m$.
- Cada linha de A corresponde a um vértice.
- Cada coluna de A corresponde a uma aresta.

$$\begin{bmatrix} (1,2) & (1,4) & (2,5) & (3,5) & (3,6) & (4,2) & (5,4) & (6,6) \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & -1 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \end{bmatrix}$$

Representação de Grafos

Matriz de incidência

- A matriz de incidência é utilizada com frequência em problemas de otimização
- Problema de fluxo em redes

$$\text{Minimize} \quad c^T x$$

Sujeito a :

$$Ax = b$$

$$l \leq x \leq u$$

$$x = [x_{ij} : (i,j) \in E]$$

Sumário

Algoritmos de Busca

Busca em Largura

- A busca em largura é um dos algoritmos mais simples para exploração de um grafo.
- Dados um grafo $G = (V, E)$ e um vértice s, chamado de fonte, a busca em largura sistematicamente explora as arestas de G de maneira a visitar todos os vértices alcançáveis a partir de s.

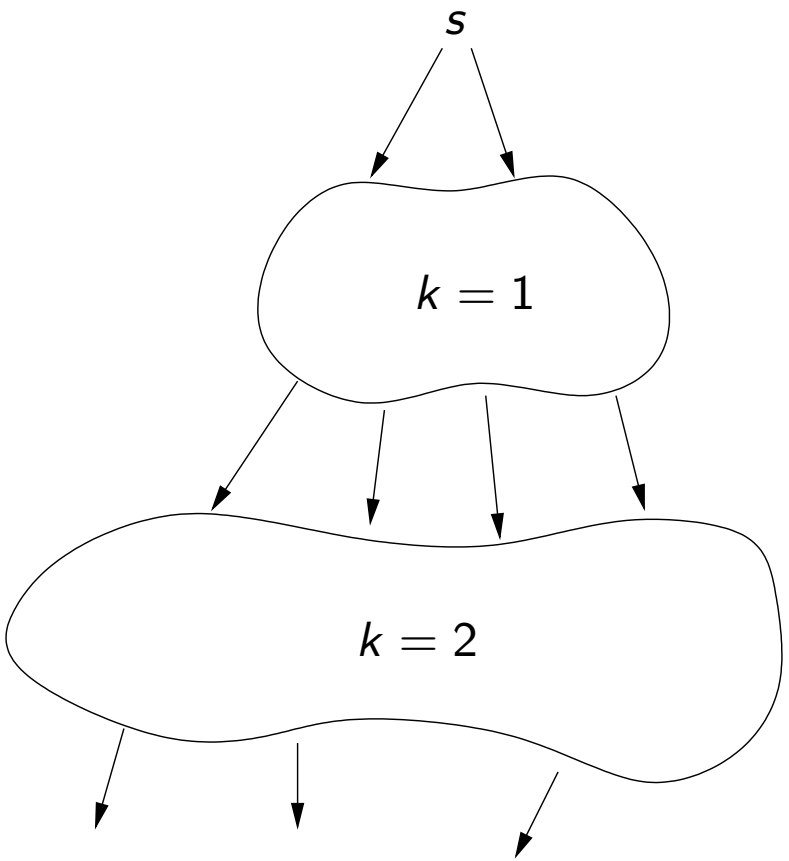
Algoritmos de Busca

Busca em Largura

- Esta busca é dita em largura porque ela expande a fronteira entre vértices conhecidos e desconhecidos de uma forma uniforme ao longo da fronteira.
- Ou seja, o algoritmo descobre todos os vértices com distância k de s antes de descobrir qualquer vértice de distância $k + 1$.

Algoritmos de Busca

Busca em Largura



Busca em Largura

Cores

- Para controlar a busca, a BL (Busca em Largura) pinta cada vértice na cor branca, cinza ou preta.
- Todos os vértices iniciam com a cor branca e podem, mais tarde, se tornar cinza e depois preta.
  - Branca: não visitado
  - Cinza: visitado
  - Preta: visitado e seus nós adjacentes visitados

Busca em Largura

Algoritmo

BFS(G, s)
    for each $u \in V[G] - \{s\}$
        $Color[u] \leftarrow white$
        $d[u] \leftarrow \infty$
        $\pi[u] \leftarrow$ NIL
    endfor
    $color[s] \leftarrow gray$
    $d[s] \leftarrow 0$
    $Q \leftarrow \{s\}$ * Queue *
⋮ ⋮

Busca em Largura

Algoritmo

```
while Q ≠ ∅
    u ← head[Q]
    for each v ∈ Adj[u]
        if color[v] = white
            color[v] ← gray
            d[v] ← d[u] + 1
            π[v] ← u
            Enqueue(Q,v)
        endif
    endfor
    Dequeue(Q)
    color[u] ← black
endwhile
```

Busca em Largura

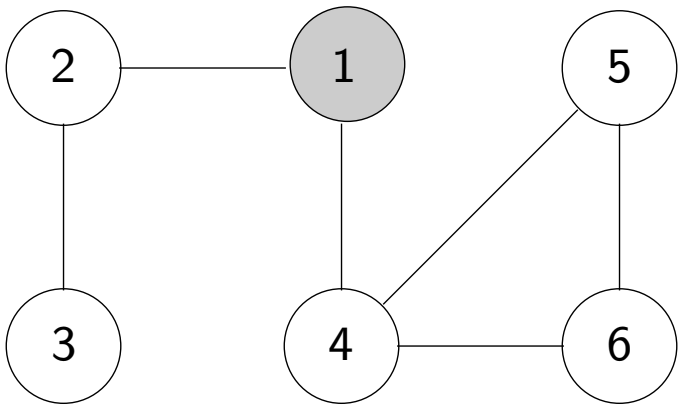
Algoritmo

- Quando um vértice é visitado pela primeira vez, sua cor é modificada de branco para cinza.
- Quando todos os vértices adjacentes a um vértice cinza são visitados, ele se torna preto.

Busca em Largura

Exemplo

- Início

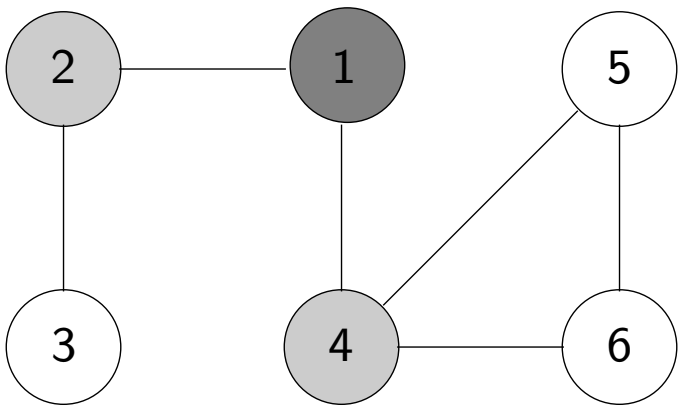


	1	2	3	4	5	6
d	0	∞	∞	∞	∞	∞
π	\	\	\	\	\	\
c	g	w	w	w	w	w
Q	1					

Busca em Largura

Exemplo

- Explorando vértice 1

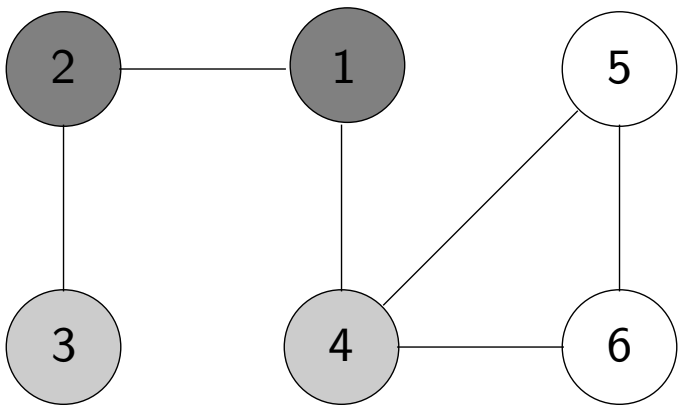


	1	2	3	4	5	6
d	0	1	∞	1	∞	∞
π	\	1	\	1	\	\
c	b	g	w	g	w	w
Q	2	4				

Busca em Largura

Exemplo

- Explorando vértice 2

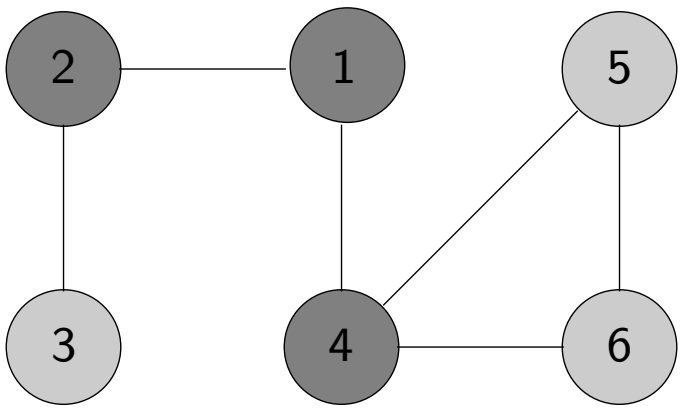


	1	2	3	4	5	6
d	0	1	2	1	∞	∞
π	0	1	2	1	\	\
c	b	b	g	g	w	w
Q	4	3				

Busca em Largura

Exemplo

- Explorando vértice 3

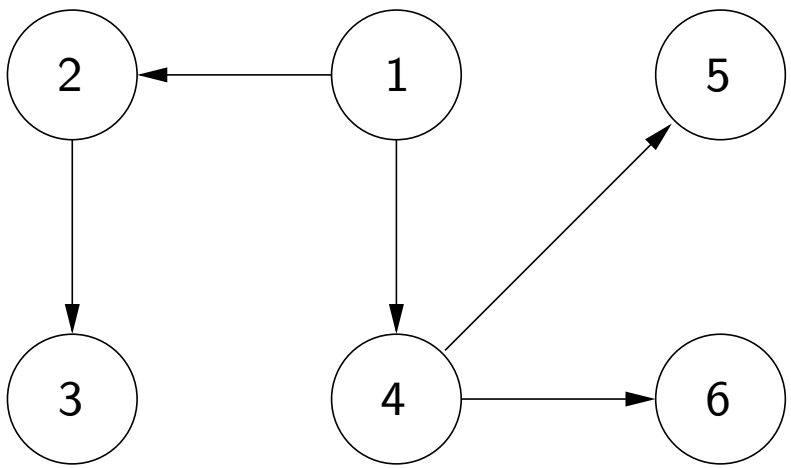


	1	2	3	4	5	6
d	0	1	2	1	2	2
π	0	1	2	1	4	4
c	b	b	g	b	g	g
Q	3	5	6			

Busca em Largura

Exemplo

- Árvore da busca em largura



Busca em Largura

Análise

- Cada vértice de V é colocado na fila Q no máximo uma vez.
- A lista de adjacência de um vértice qualquer de u é percorrida somente quando o vértice é removido da fila
- Daí concluímos que o algoritmo roda em tempo $O(|V| + |E|)$ pois as operações executadas levam $\Theta(1)$.

Busca em Largura

Caminho mais curto

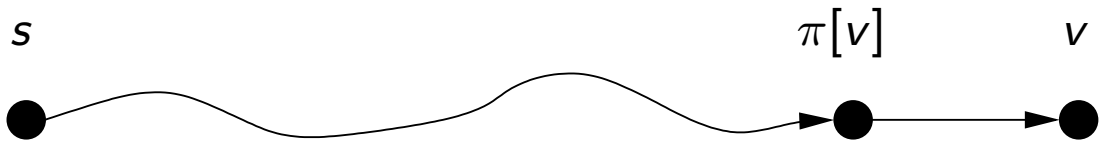
Seja $\delta(s, v)$ a distância do vértice v a partir do vértice s, sendo a distância o menor número de arestas em qualquer caminho em G com origem em s e destino para v.

Busca em Largura

Teorema

Seja $G = (V, E)$ um grafo direcionado ou não, e suponha que BFS é executada a partir de um vértice $s \in V$. Então:

- Durante a busca, BFS descobre cada vértice $v \in V$ que seja alcançável a partir de s
- Ao final, $d[v] = \delta(s, v)$ para todo $v \in V$.
- Além disso, para qualquer vértice $v \neq s$ que seja alcançável a partir de s, um caminho mais curto de s para v é o caminho de s para $\pi[v]$ seguido da aresta $(\pi[v], v)$.



Sumário

Algoritmos de Busca

Busca em Profundidade

- A estratégia aqui é explorar o grafo em profundidade.
- Na busca em profundidade, as arestas são exploradas a partir do vértice mais recentemente visitado.
- Da mesma forma que a busca em largura, sempre que um vértice v é descoberto durante a busca na lista de adjacência de um outro vértice já visitado u, a DFS memoriza este evento ao definir o predecessor de v, $\pi[v]$ como u.
- Diferentemente da BFS, cujo grafo predecessor forma uma árvore, o grafo predecessor de DFS pode ser composto de várias árvores.

Busca em Profundidade

Grafo predecessor

$$G_\pi = (V, E_\pi)$$
$$E_\pi = \{(\pi[v], v) : v \in V \text{ e } \pi[v] \neq \text{NIL } \}$$

Os vértices do grafo são coloridos durante a busca.

- Branco: antes da busca.
- Cinza: quando o vértice for visitado.
- Preto: quando os vértices adjacentes foram visitados.

Busca em Profundidade

timestamp

Além de construir uma floresta, DFS marca cada vértice com um *timestamp*. Cada vértice tem dois *timestamps*.

- $d[v] \rightarrow$ indica o instante em que v foi visitado (*pintado com cinza*).
- $f[v] \rightarrow$ indica o instante em que a busca pelos vértices na lista de adjacência de v foi completada (*pintado de preto*).

Usando *timestamp* $1, 2, \ldots$, verifica-se que

- $d[v],\ f[v] \in 1, \ldots, 2|V|, \quad \forall v \in V$
- $d[v] < f[v], \quad \forall v \in V$

Busca em Profundidade

Algoritmo

DFS(G)
    for each vertex $u \in V[G]$
        $color[u] \leftarrow white$
        $\pi[u] \leftarrow$ NIL
    $time \leftarrow 0$
    for each $u \in V[G]$
        if $color[u] = white$
            DFS_visit(u)

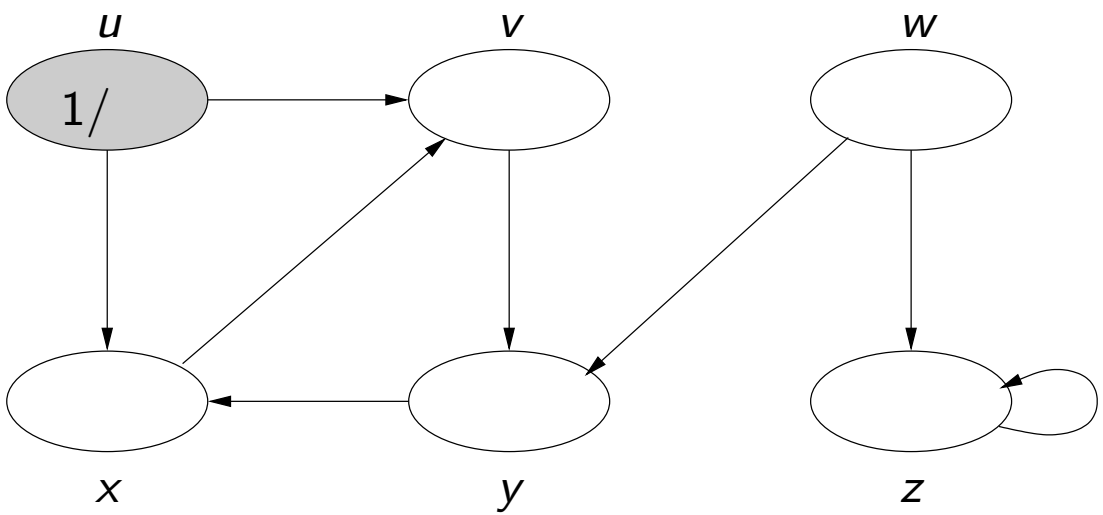
Busca em Profundidade

Algoritmo

DFS_visit(u)
    $color[u] \leftarrow gray$
    $d[u] \leftarrow time \leftarrow time + 1$
    for each $v \in Adj[u]$
        if $color[u] = white$
            $\pi[v] \leftarrow u$
            DFS_visit(v)
    $color[u] \leftarrow black$
    $f[u] \leftarrow time \leftarrow time + 1$

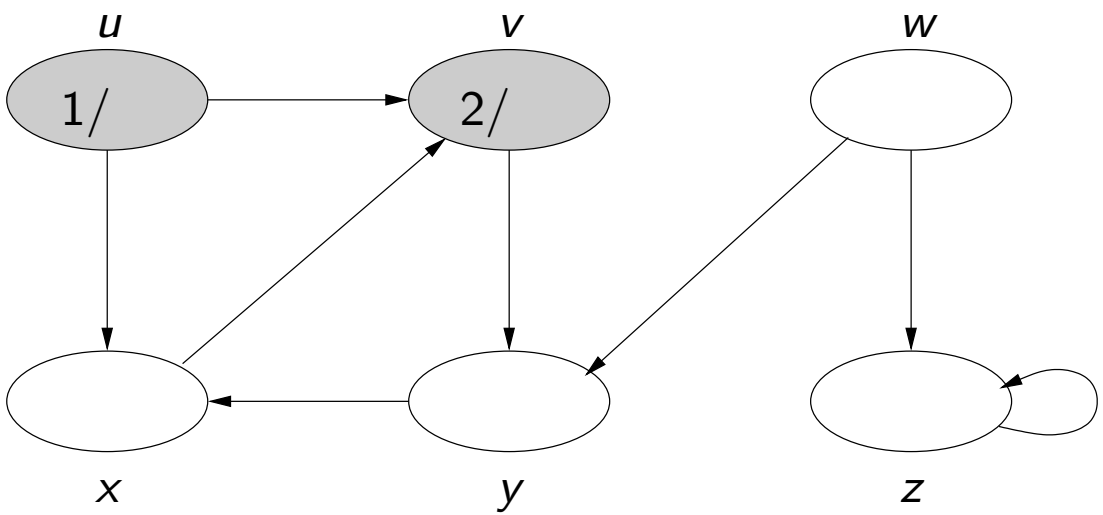
Busca em Profundidade

Exemplo



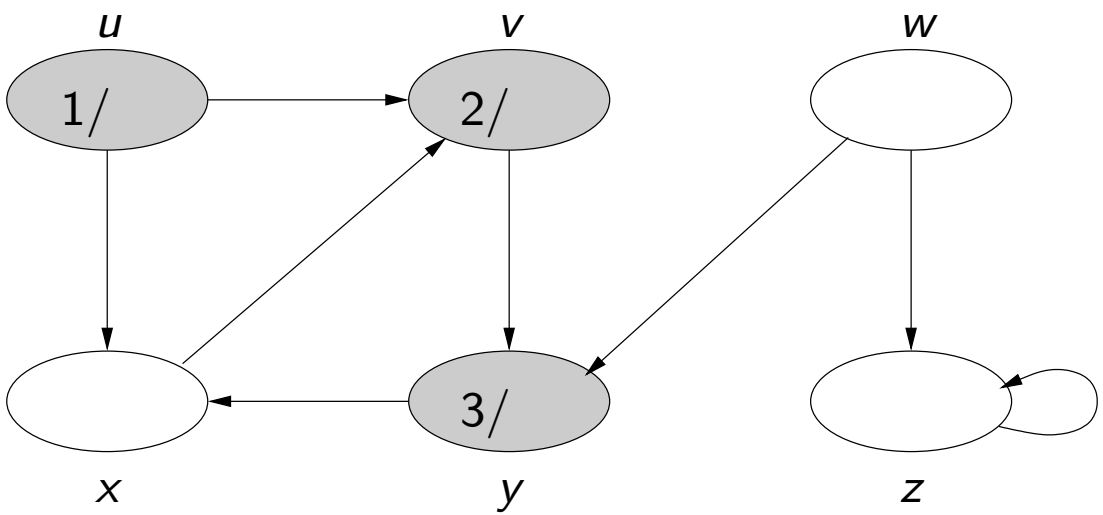
Busca em Profundidade

Exemplo



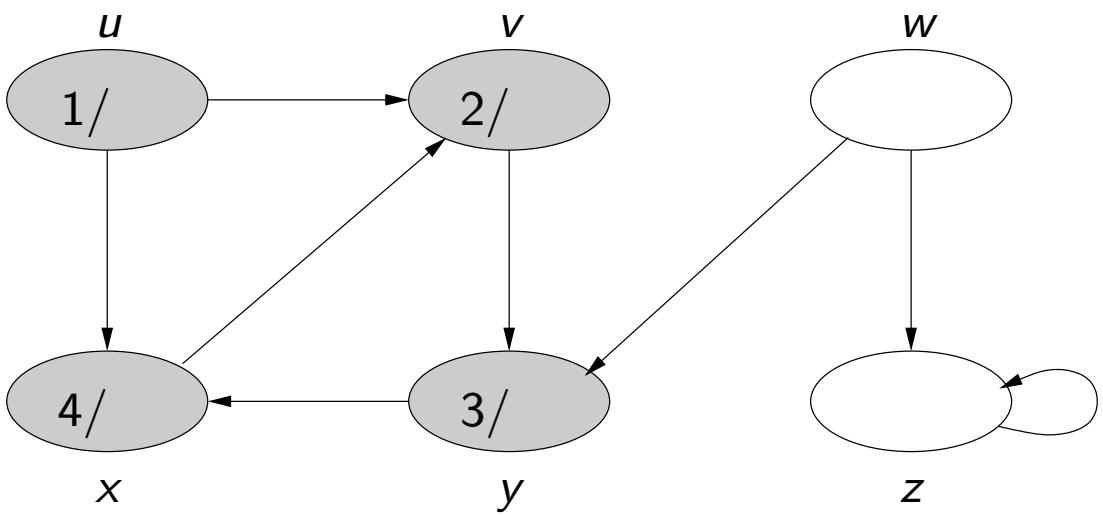
Busca em Profundidade

Exemplo



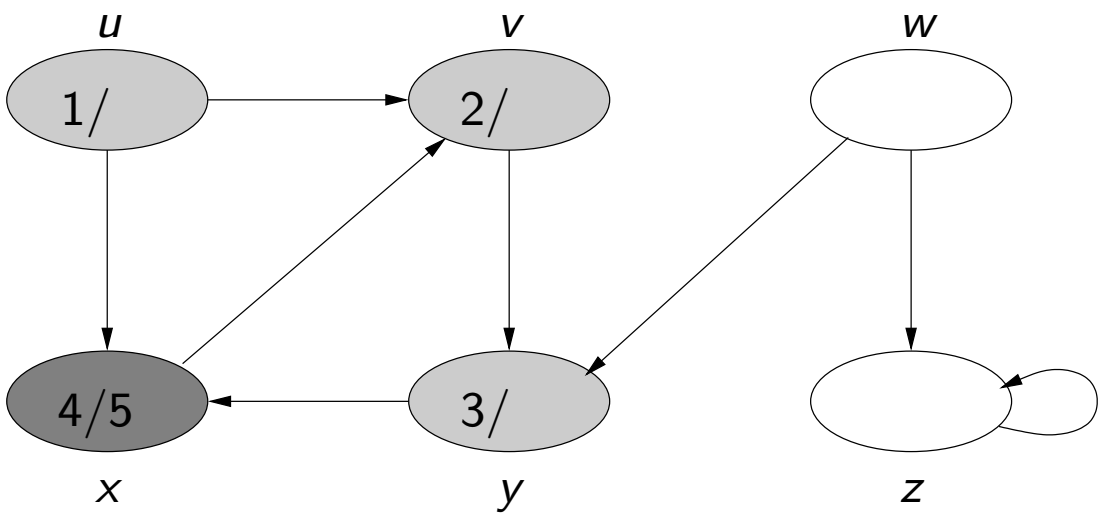
Busca em Profundidade

Exemplo



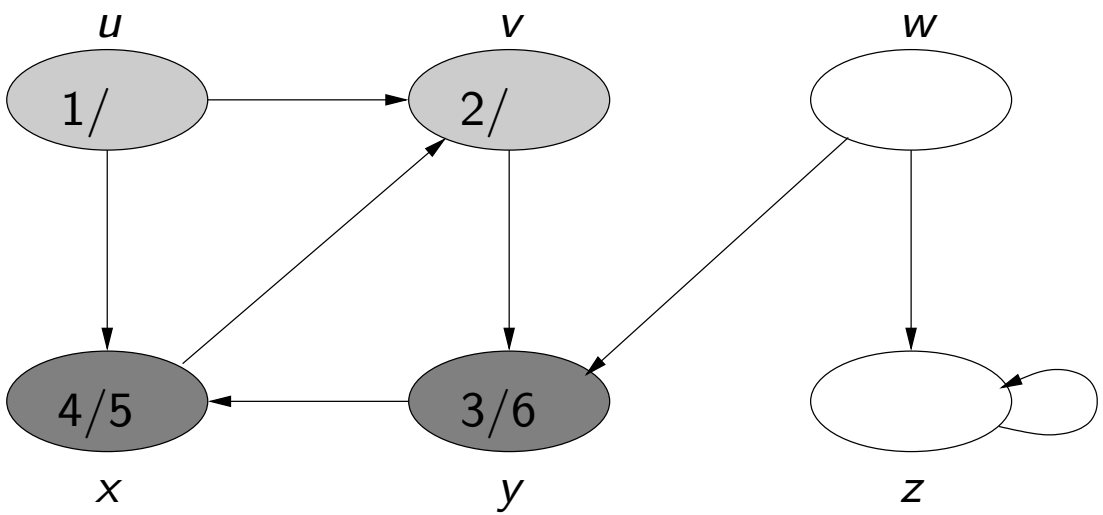
Busca em Profundidade

Exemplo



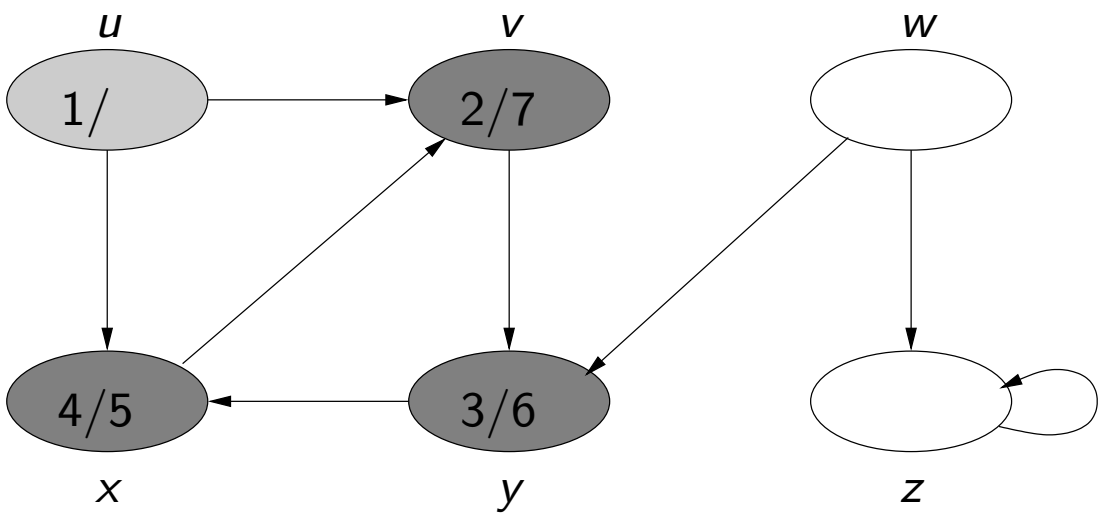
Busca em Profundidade

Exemplo



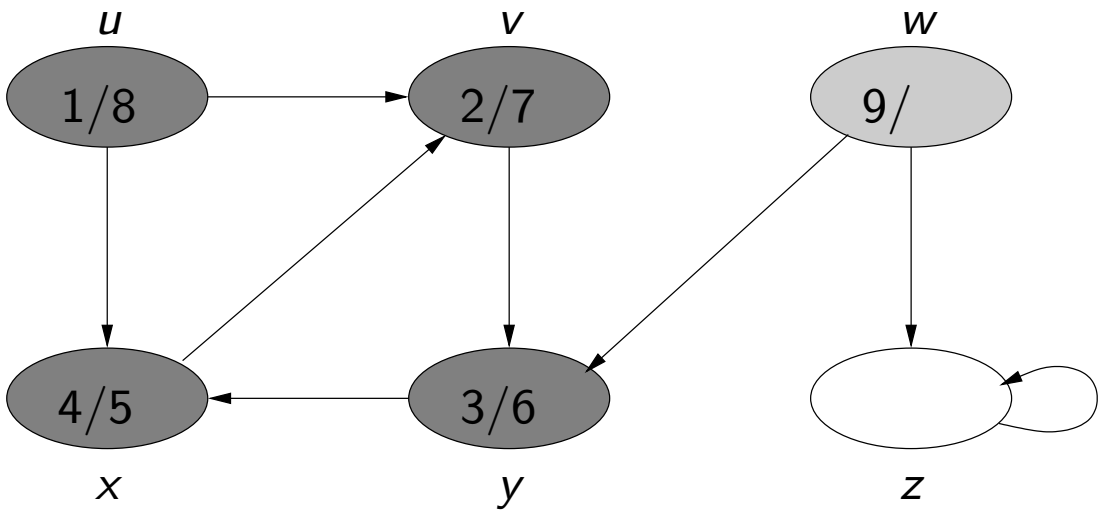
Busca em Profundidade

Exemplo



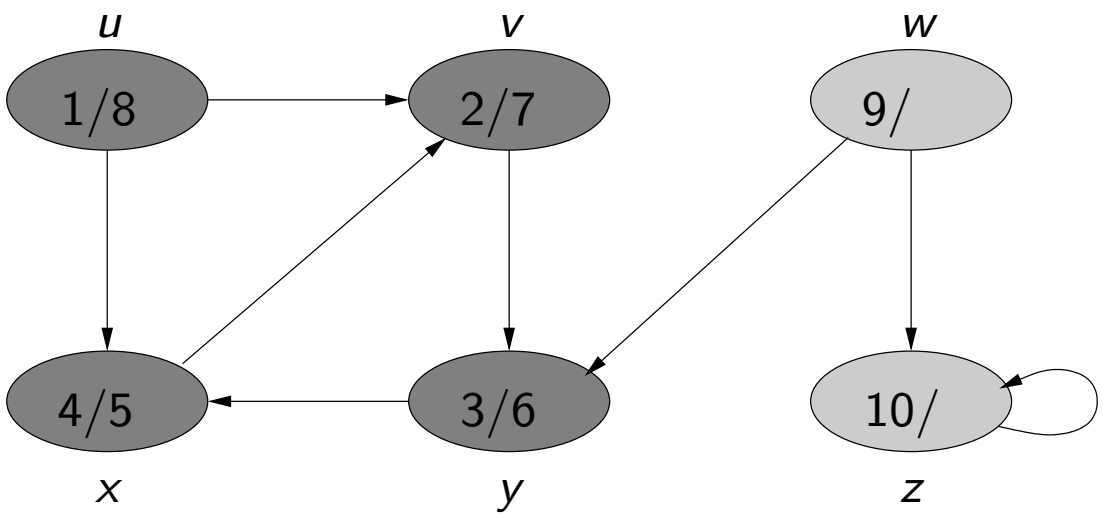
Busca em Profundidade

Exemplo



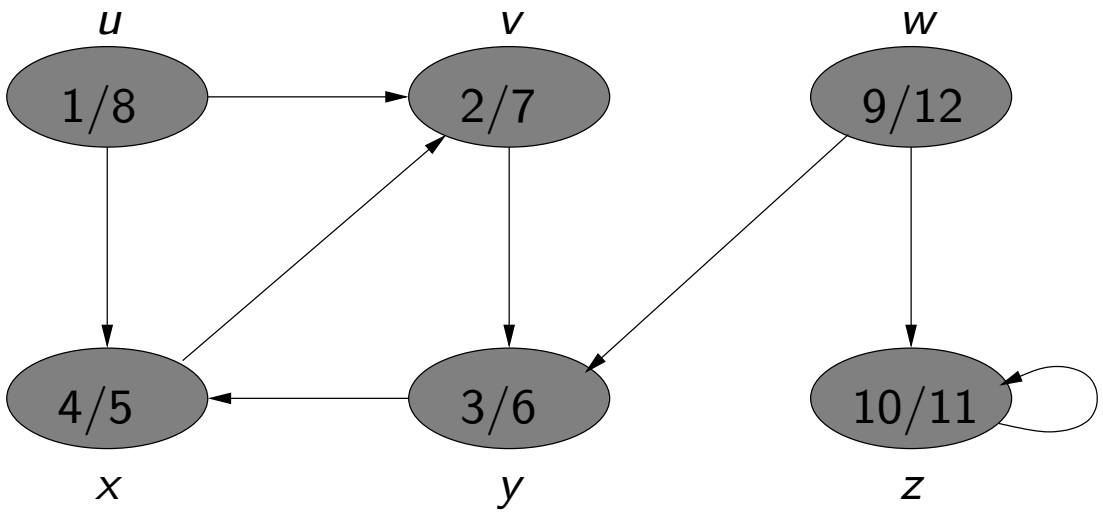
Busca em Profundidade

Exemplo



Busca em Profundidade

Exemplo



Busca em Profundidade

Análise

O procedimento DFS_visit(v) é chamado exatamente uma vez para cada vértice, pois é chamado apenas para vértices *white* e na primeira vez que isto acontece, ele é pintado de *gray*.

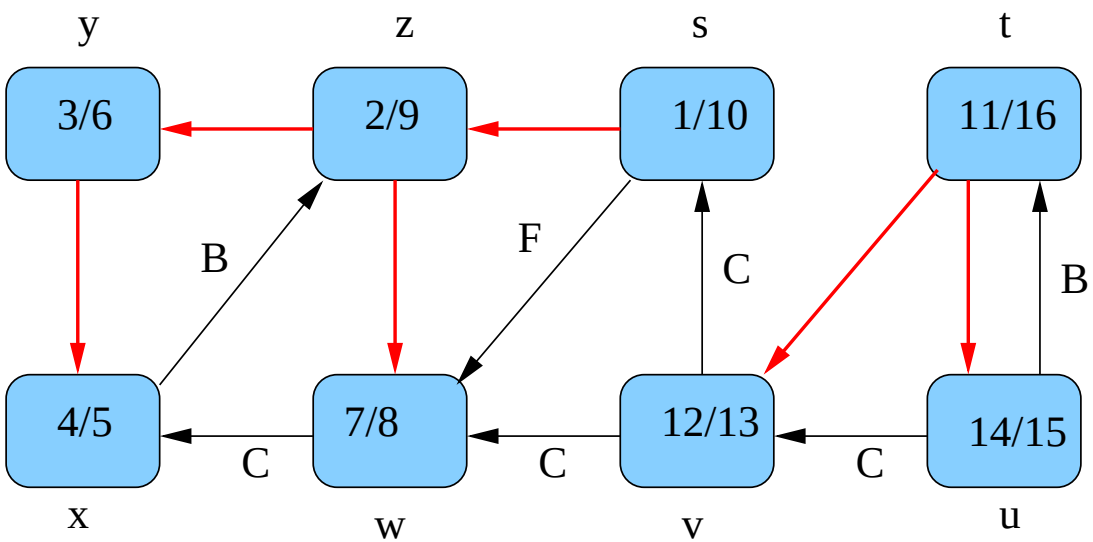
Propriedades da Busca em Profundidade

Teorema dos Parênteses

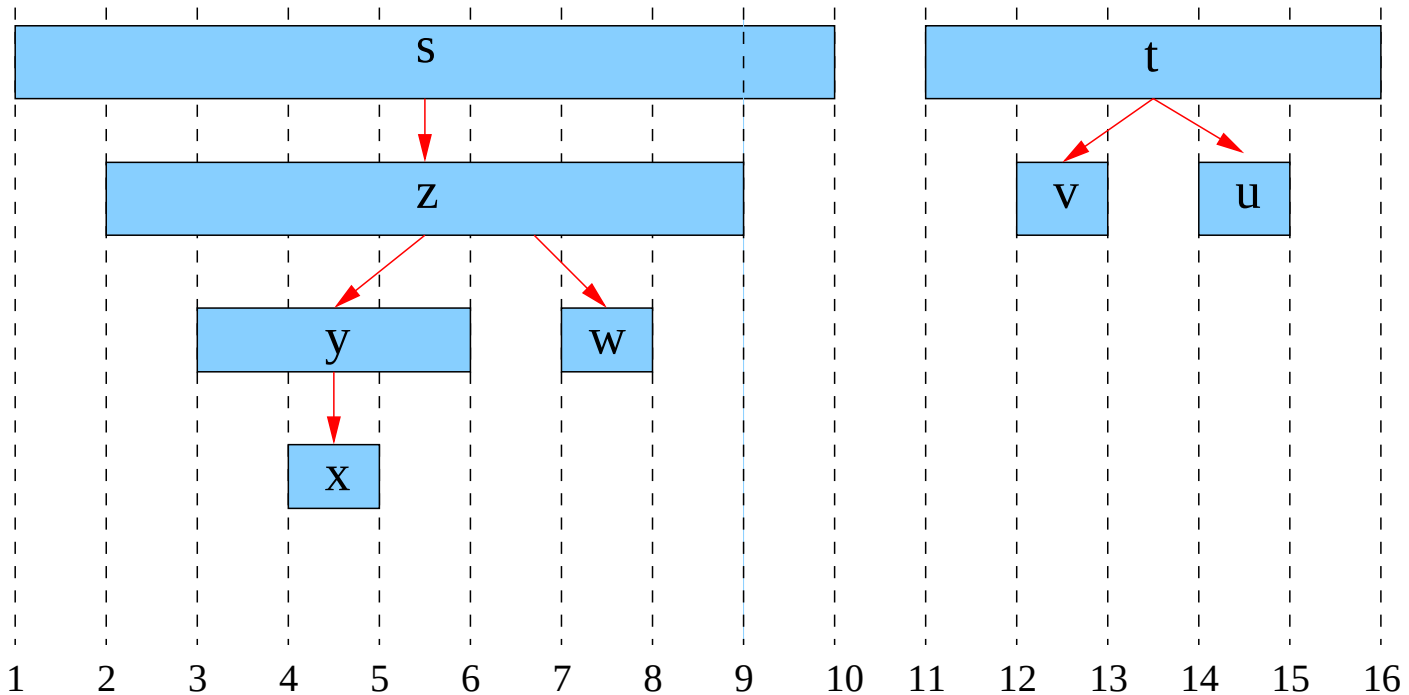
Na busca em profundidade de um grafo (direcionado ou não-direcionado) $G = (V, E)$, para quaisquer dois vértices u e v, exatamente uma de três condições vale:

1. os intervalos $[d[u], f[u]]$ e $[d[v], f[v]]$ são disjuntos, e u não é descendente de v, bem como v não é descendente de u na floresta da busca em profundidade
2. o intervalo $[d[u], f[u]]$ está contido em $[d[v], f[v]]$, e u é um descendente de v na floresta da busca em profundidade
3. o intervalo $[d[v], f[v]]$ está contido em $[d[u], f[u]]$, e v é um descendente de u na floresta da busca em profundidade

Teorema dos Parênteses



Teorema dos Parênteses



Propriedades da Busca em Profundidade

- A busca em profundidade pode ser usada para classificar as arestas de $G = (V, E)$.
- Tal classificação traz informações úteis sobre o grafo.
- Por exemplo, G é acíclico se não existem arestas "reversas"

Propriedades da Busca em Profundidade

Podemos classificar as arestas em quatro tipos de acordo com a floresta G_π produzida pela busca em profundidade;

Arestas Árvore: as arestas da floresta em profundidade G_π

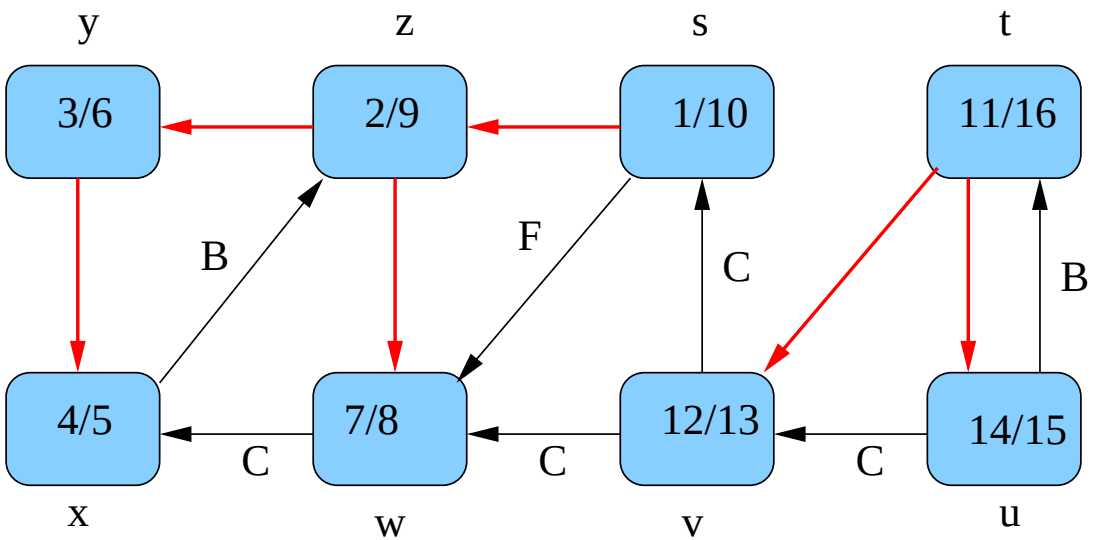
Arestas Reversas: as arestas (u, v) que conectam u a um ancestral v

Laços são considerados arestas reversas.

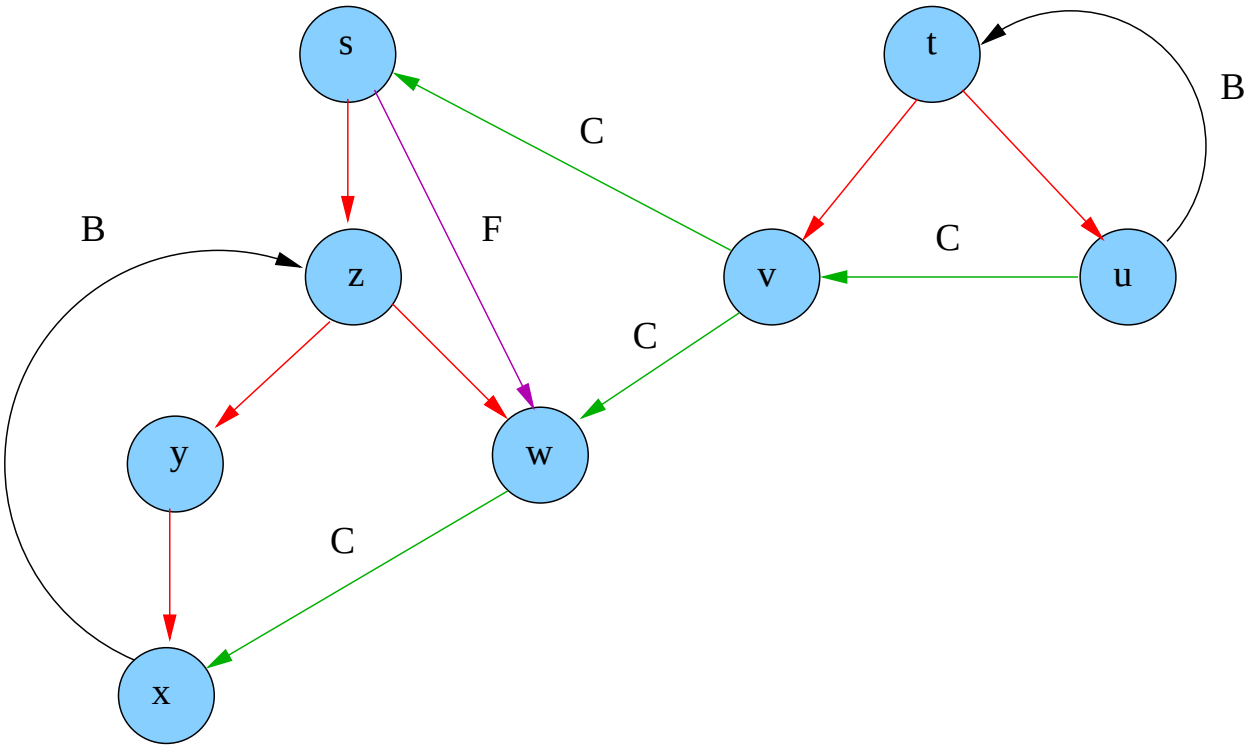
Arestas Diretas: as arestas (u, v) que conectam um vértice u a um descendente v

Arestas Cruzadas: todas as demais arestas

Busca em Profundidade



Classificação de Arestas



Sumário

Ordenação topológica

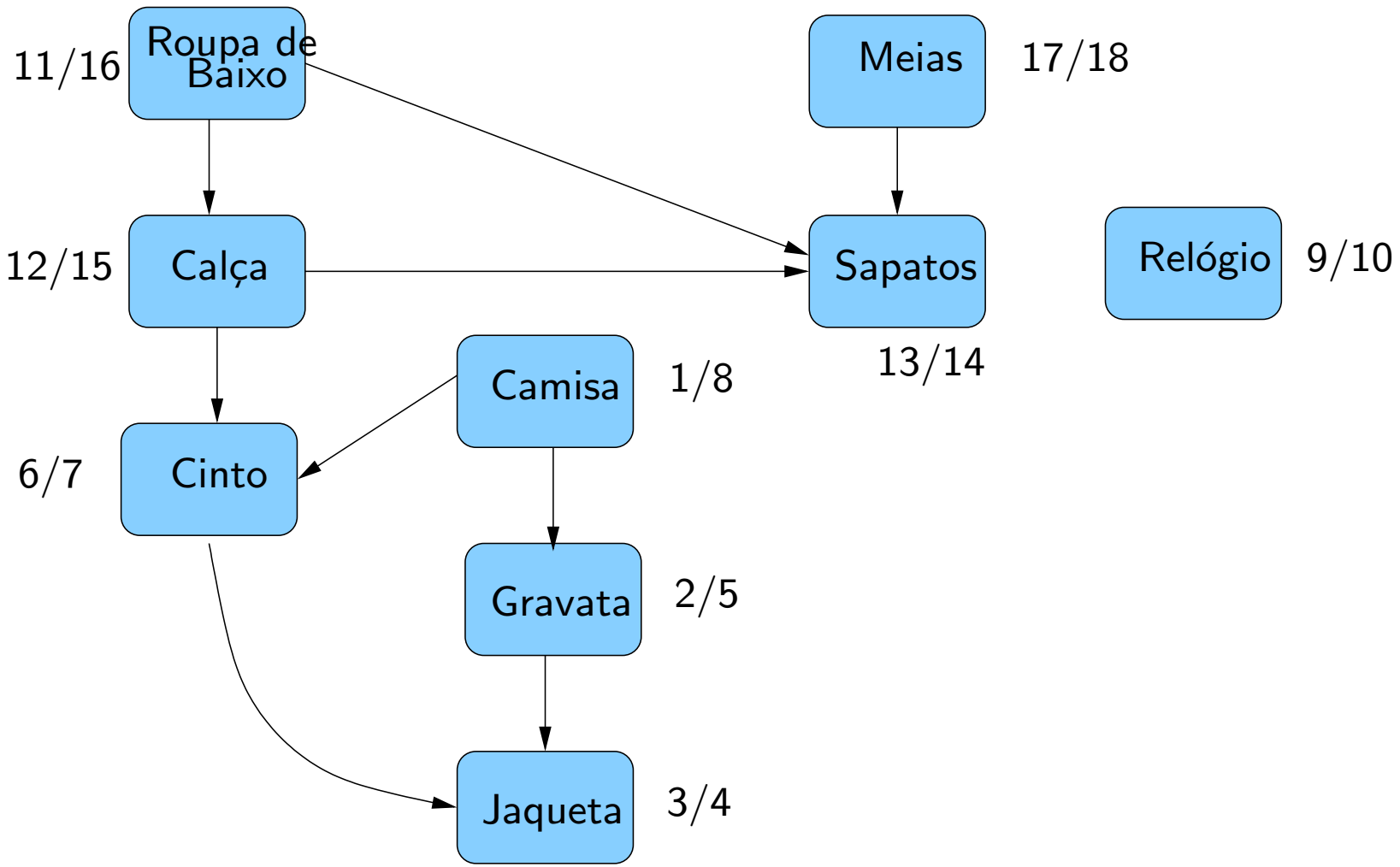
- Mostraremos como busca em profundidade pode ser empregada para encontrar uma ordenação topológica de um grafo direcionado acíclico $G = (V, E)$.
- Uma ordenação topológica $\langle u_1, \ldots, u_n \rangle$ dos vértices de G é uma ordenação linear tal que se $(u_i, u_j) \in E$, então u_i precede u_j na ordenação, ou seja, $i < j$.
- Ordenação topológica pode ser vista como um arranjo dos vértices na horizontal, tal que as arestas vão da esquerda para a direita.

Busca em Largura

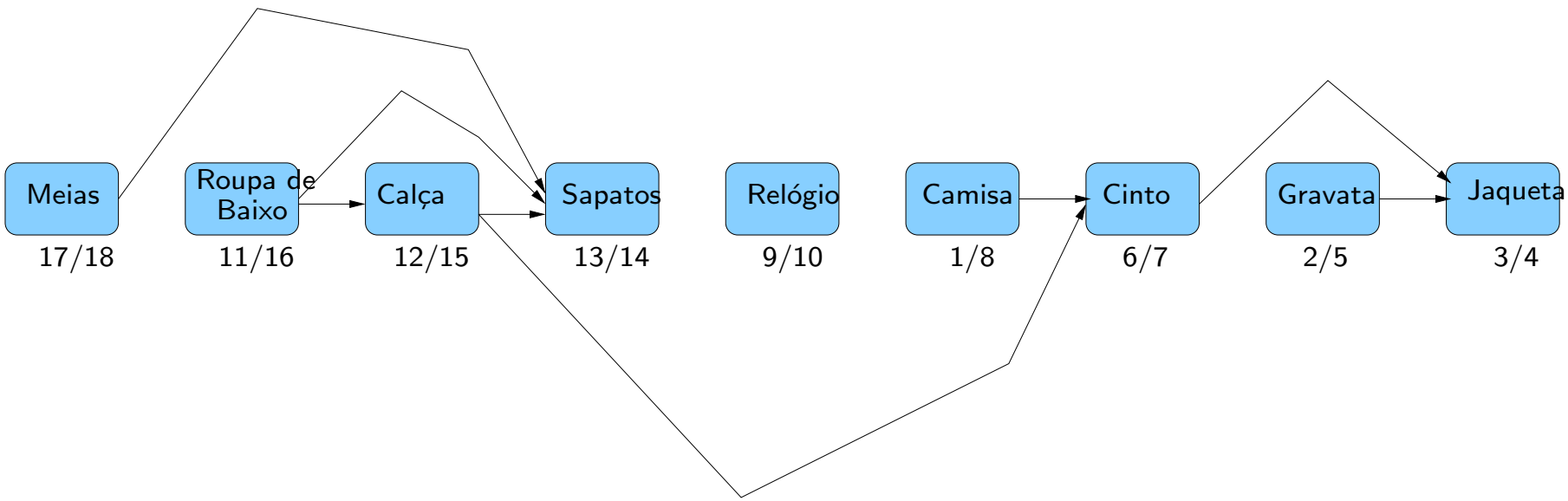
Topological-Sort(G)

call DFS(G) to compute finishing $f[v]$ for each vertex v,
    as each vertex is finished, insert it into the front of a linked list
return the linked list of vertices

Exemplo


11/16
Roupa de Baixo
Meias
17/18
12/15
Calça
Sapatos
Relógio
9/10
13/14
Camisa
1/8
6/7
Cinto
Gravata
2/5
Jaqueta
3/4

Exemplo


Meias
17/18
Roupa de Baixo
11/16
Calça
12/15
Sapatos
13/14
Relógio
9/10
Camisa
1/8
Cinto
6/7
Gravata
2/5
Jaqueta
3/4

Sumário

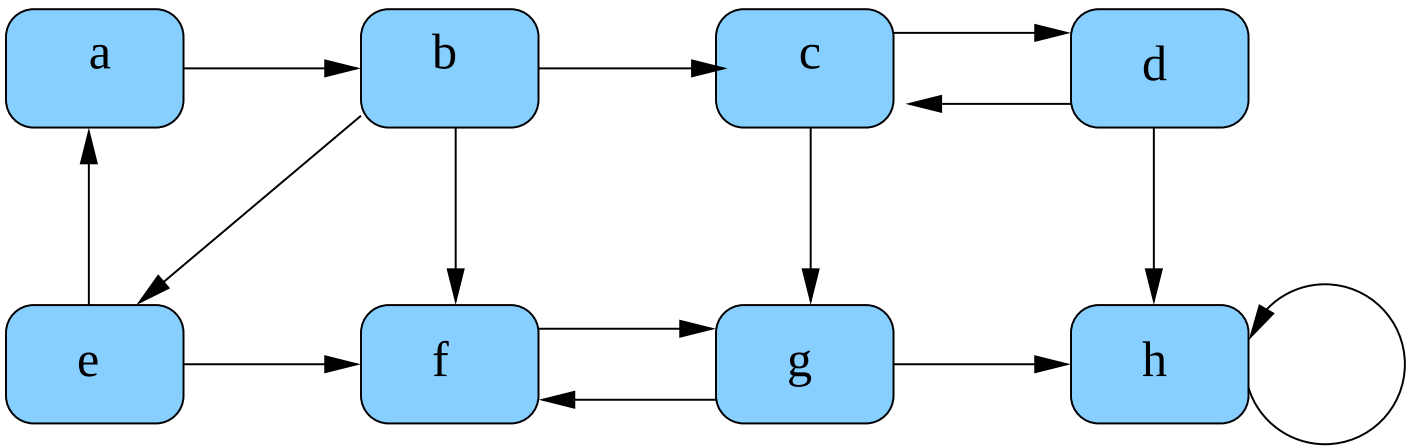
Componentes Fortemente Conexos

- Um componente fortemente conexo de um grafo direcionado $G = (V, E)$ é um conjunto de vértices $C \subseteq V$ máximo tal que para todo par de vértices u e v em C, existe um caminho $u \rightsquigarrow v$ e um caminho $v \rightsquigarrow u$.
- Estamos interessados em desenvolver um algoritmo que encontra todos os componentes fortemente conexos de G.
- Vamos utilizar o grafo transposto $G^T = (V, E^T)$ onde $E^T = \{(u, v) : (v, u) \in E\}$. G^T consiste de G com as arestas reversas.

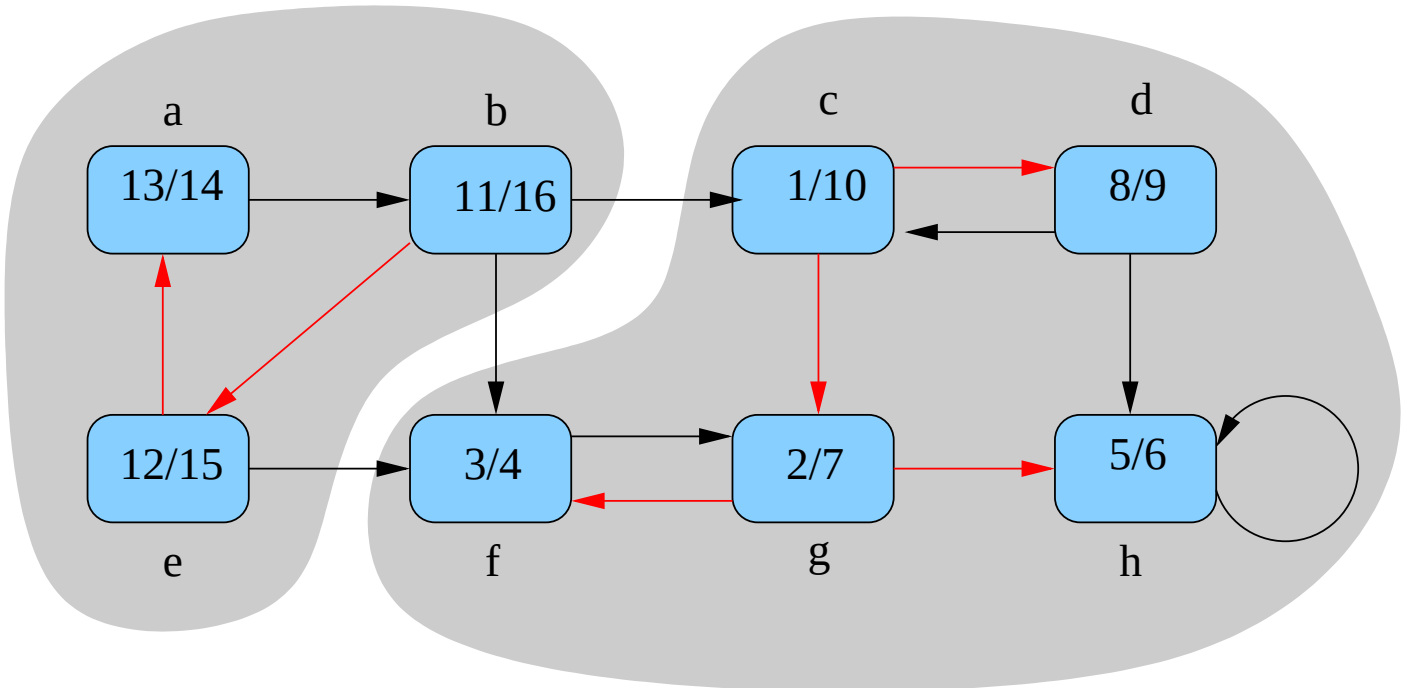
Strongly-Connected-Components(G)

1. call DFS(G) to compute finishing $f[u]$ for each vertex u
2. compute G^T
3. call $DFS(G^T)$, but in the main loop of DFS, consider the vertices in decreasing order of $f[u]$ (as computed in step 1)
4. output the vertices of each depth-first search tree built in step 3 as an independent connected component

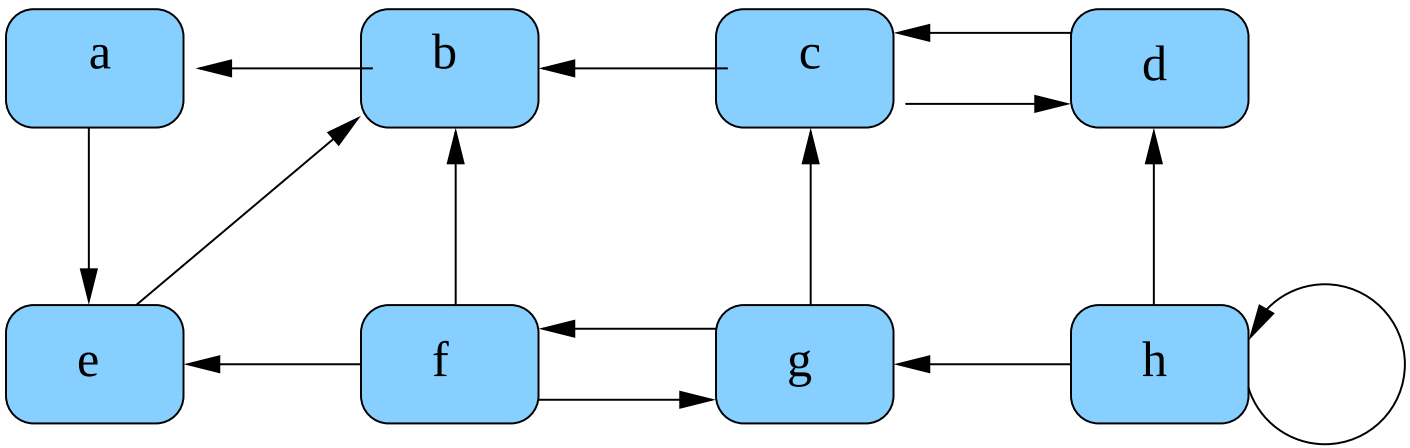
Exemplo: Grafo $G = (V, E)$



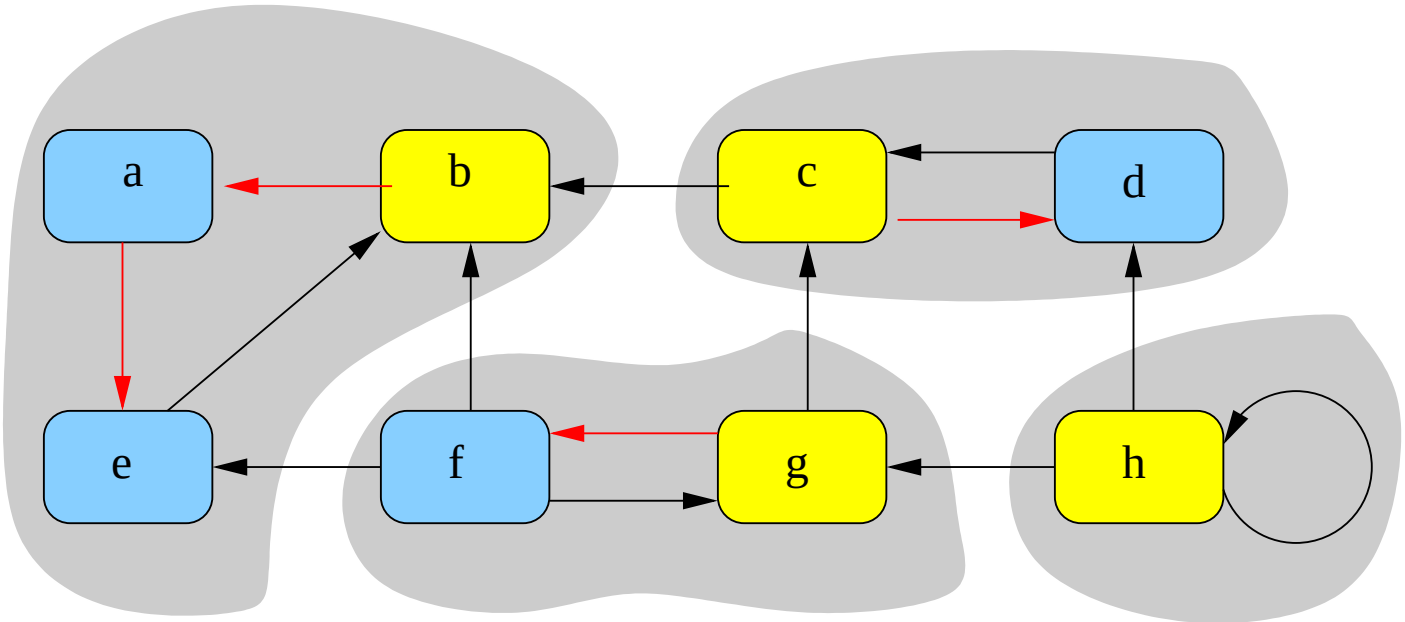
DFS(G)



G^T



DFS(G^T)



Nós em amarelo são raízes das árvores de profundidade.

Conclusões

- Fim!
- Obrigado pela presença