Busca em Profundidade
5189-32

Rodrigo Calvo
rcalvo@uem.br

Departamento de Informática – DIN
Universidade Estadual de Maringá – UEM

1° semestre de 2016

Introdução

- A busca em profundidade, (*depth-first search* - dfs), é um algoritmo para caminhar no grafo.
- A estratégia é buscar o mais profundo no grafo sempre que possível.
- O algoritmo é a base para muitos outros algoritmos importantes, tais como verificação de grafos acíclicos, ordenação topológica e componentes fortemente conectados.

Busca em Profundidade
Funcionamento

- As arestas são exploradas a partir do vértice *v* mais recentemente descoberto que ainda possui arestas não exploradas saindo dele.
- Quando todas as arestas adjacentes a *v* forem exploradas, a busca anda para trás (*backtrack*) para explorar vértices que saem do vértice do qual *v* foi descoberto.
- Este processo continua até que todos os vértices acessíveis a partir da origem tenham sidos descobertos
- Se restarem vértices não descobertos, a busca se repetirá para estes vértices

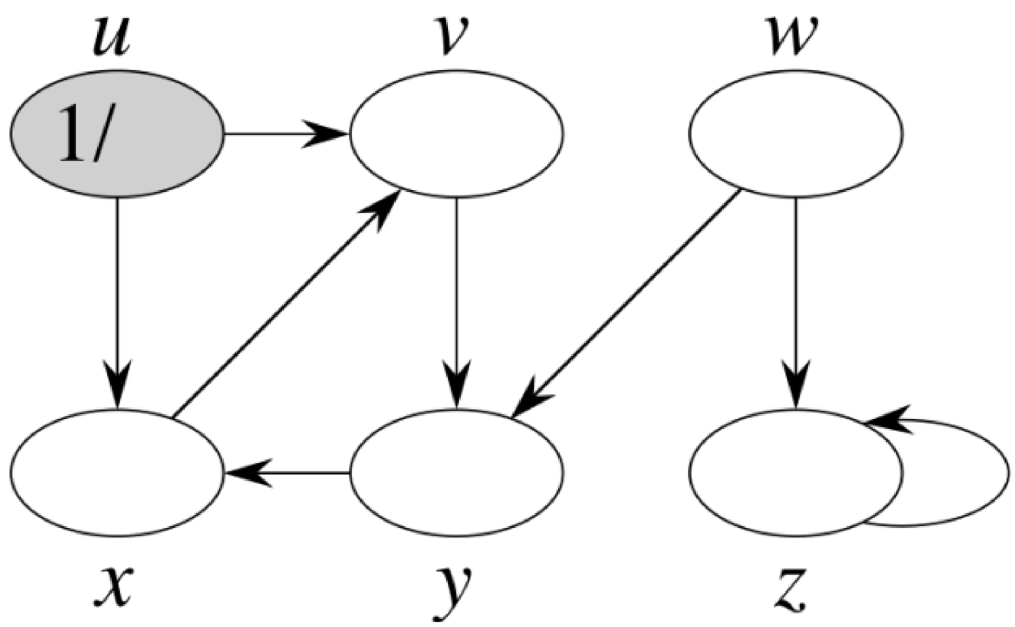
Busca em Profundidade
Atributos

- Durante a execução do algoritmo, diversos atributos são definidos para os vértices
- $v.\pi$ → antecessor do vértice v
  - Quando um vértice v é descoberto a partir de um vértice u, o campo predecessor $v.\pi = u$ é definido
- $v.cor$ → indica o estado do vértice
  - Cada vértice é inicialmente branco, o vértice é marcado de cinza quando é descoberto e marcado de preto quando é terminado (sua lista de adjacências é completamente examinada)

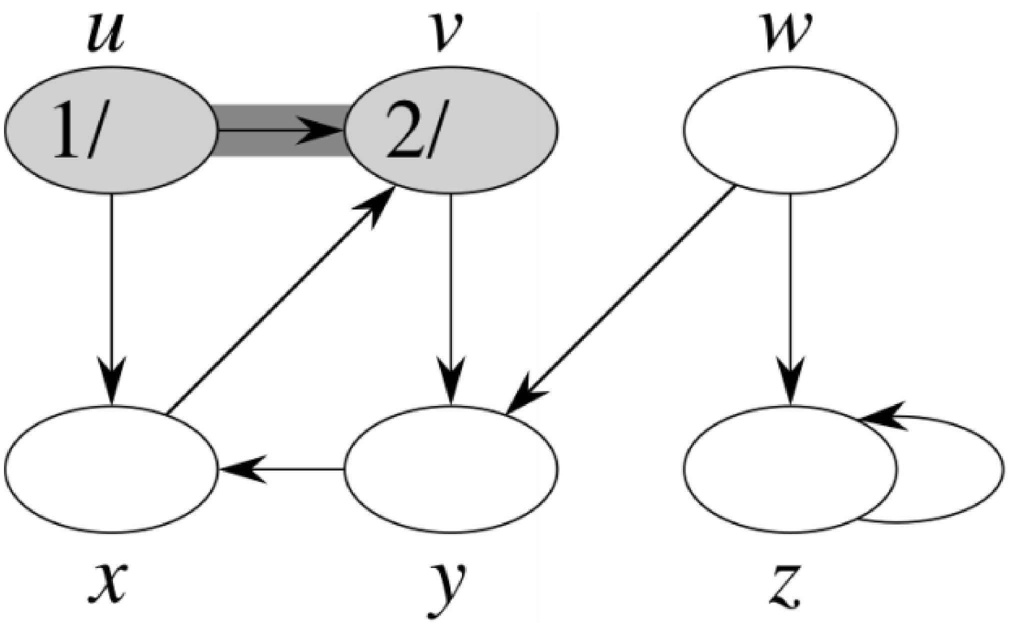
Busca em Profundidade
Atributos

- *v.adj* → lista de adjacências do vértice *v*
- *v.d* e *v.f* →marcadores de tempo
  - Indica o instante quando o vértice v é descoberto ou finalizado (sua lista de adjacências é finalizada), respectivamente

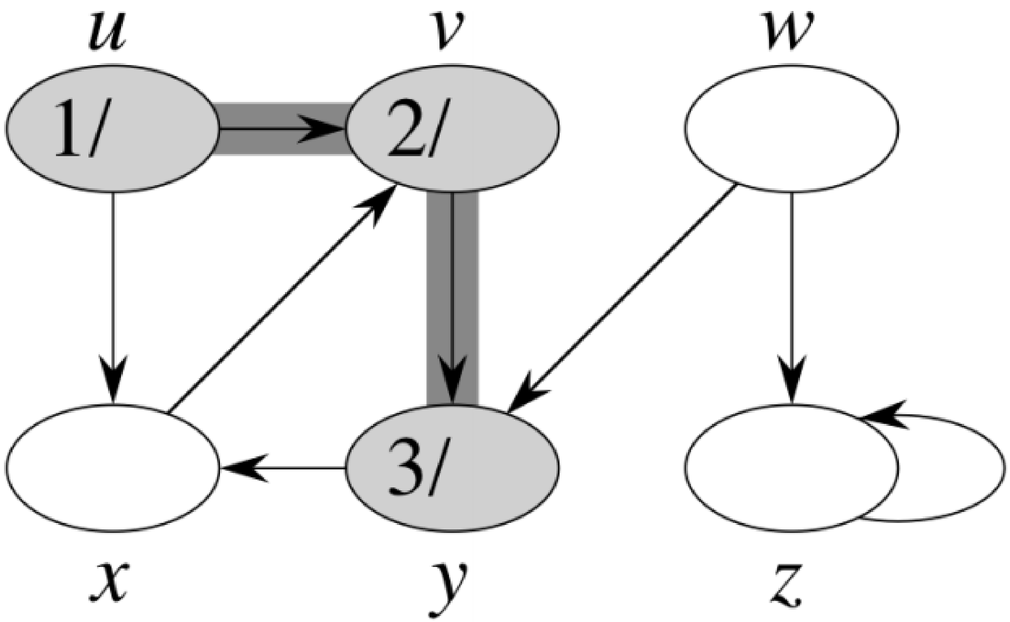
Busca em Profundidade: Exemplo



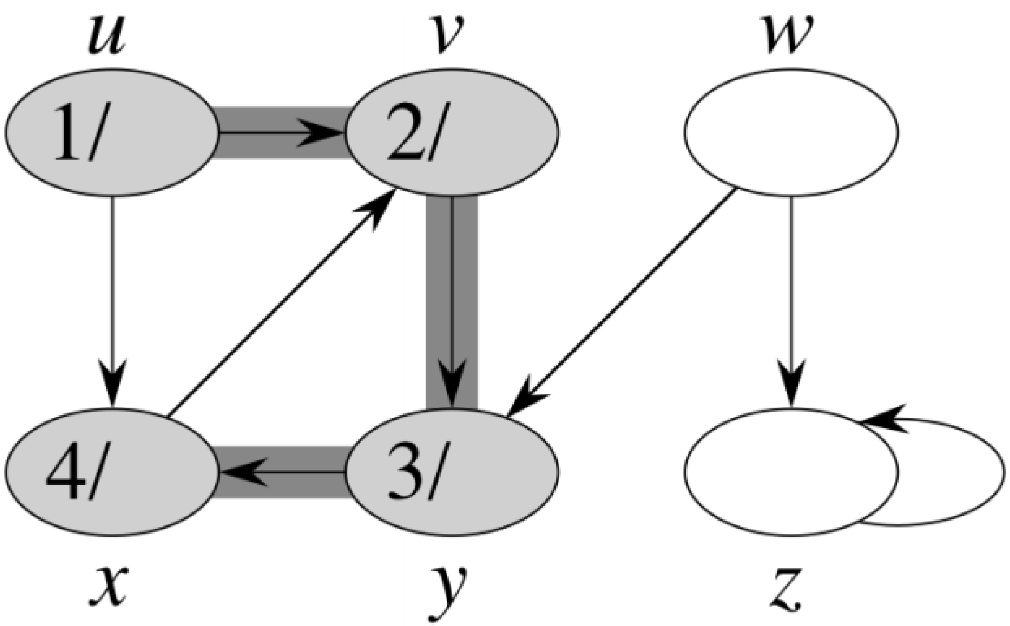
Busca em Profundidade: Exemplo



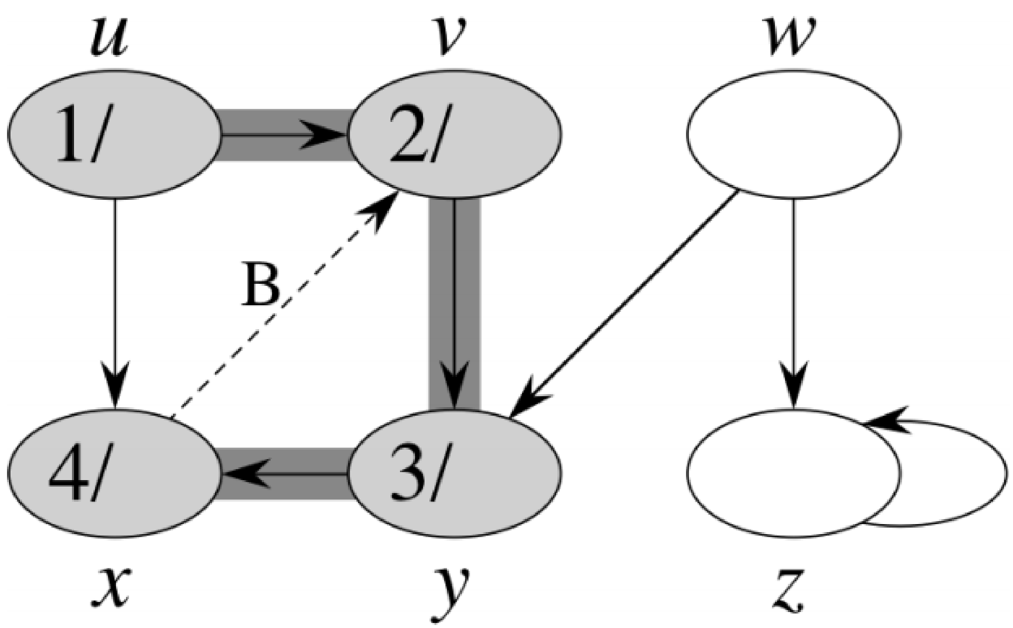
Busca em Profundidade: Exemplo



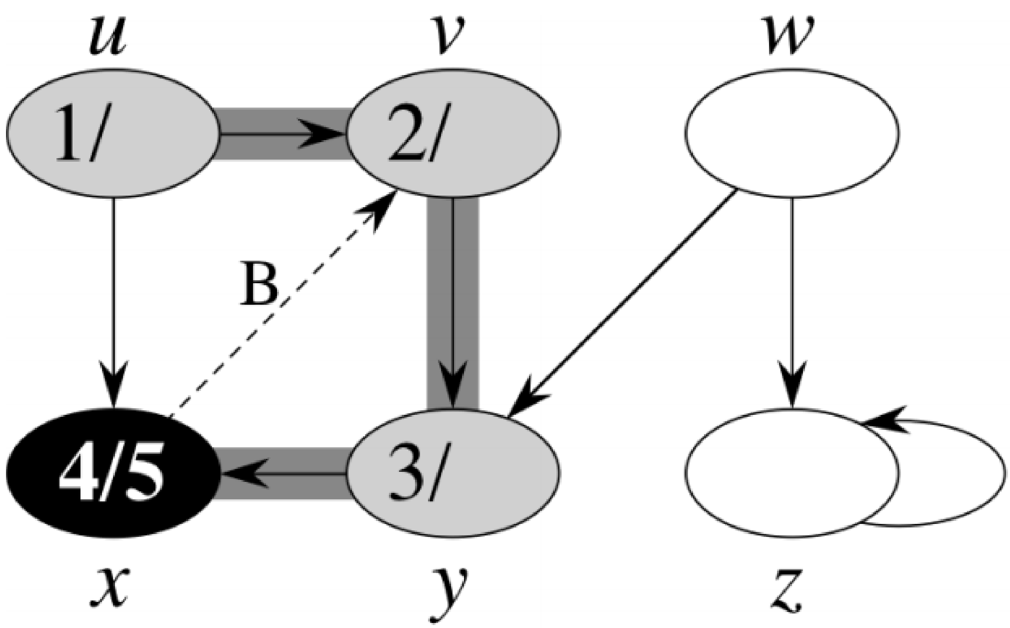
Busca em Profundidade: Exemplo



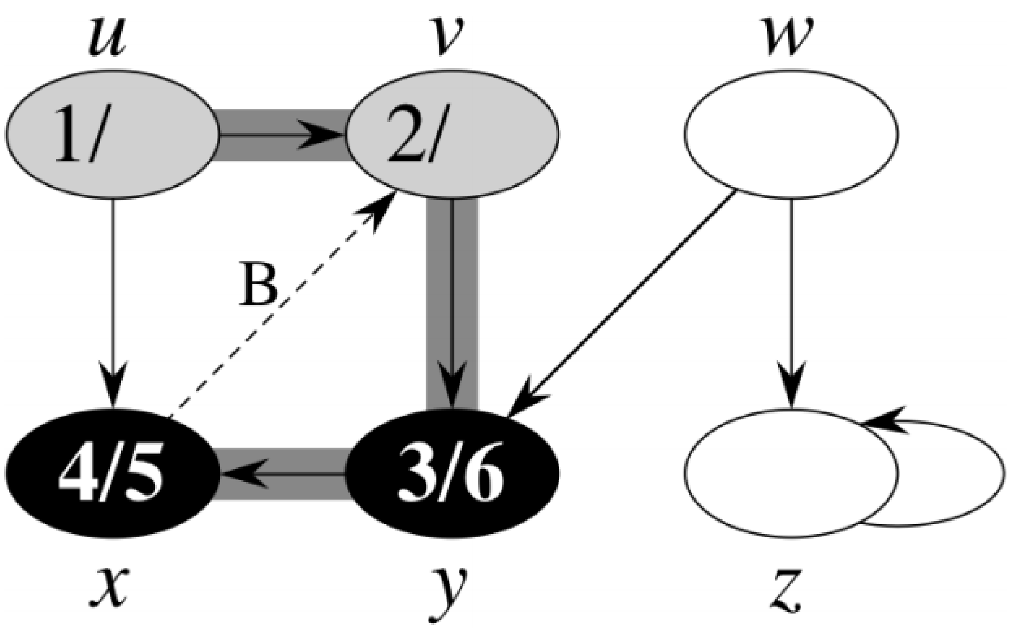
Busca em Profundidade: Exemplo



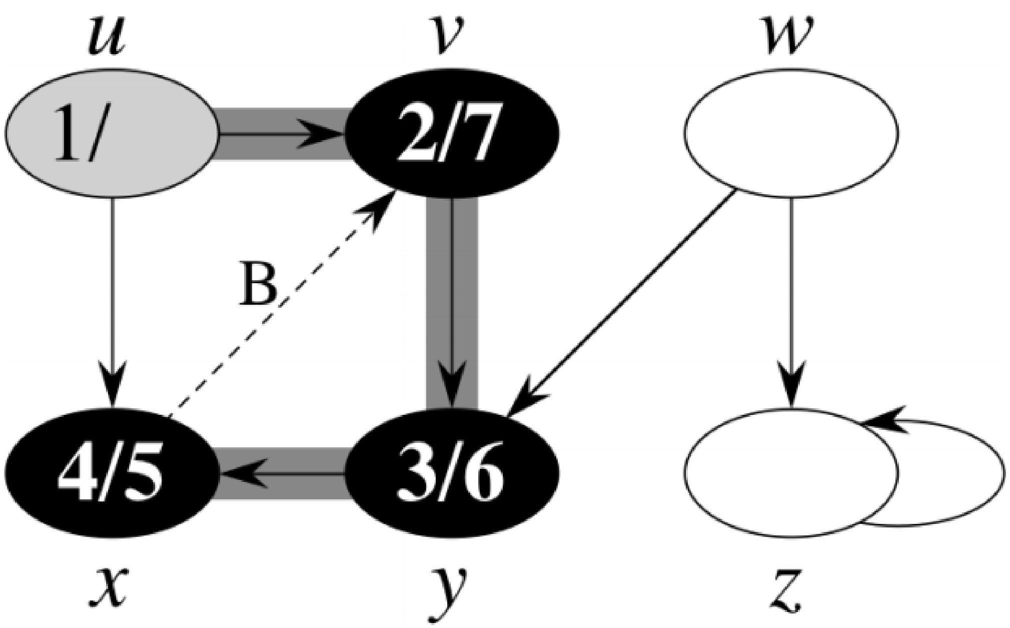
Busca em Profundidade: Exemplo



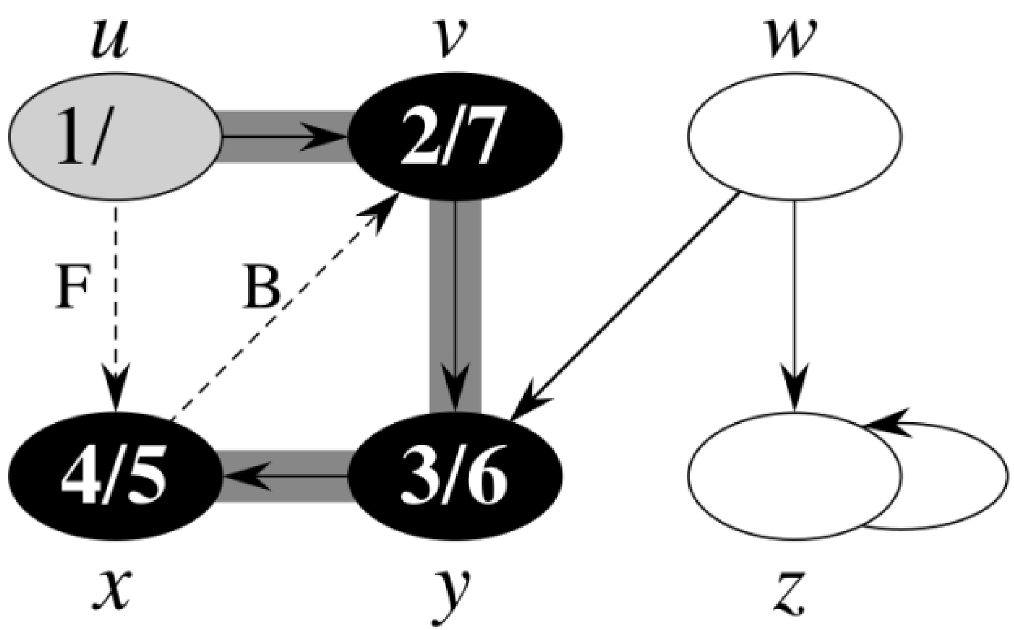
Busca em Profundidade: Exemplo



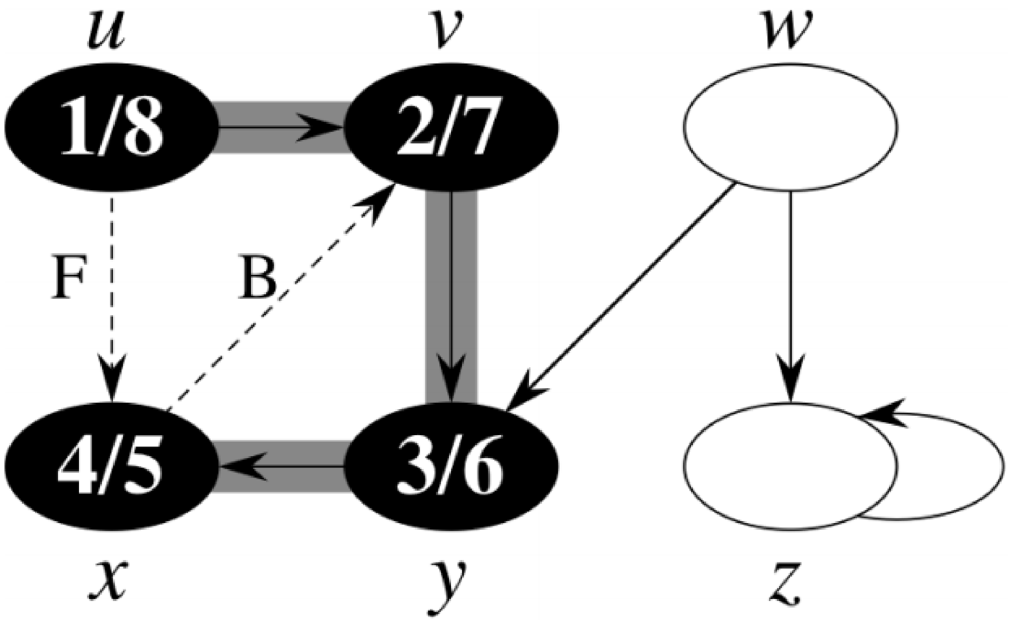
Busca em Profundidade: Exemplo



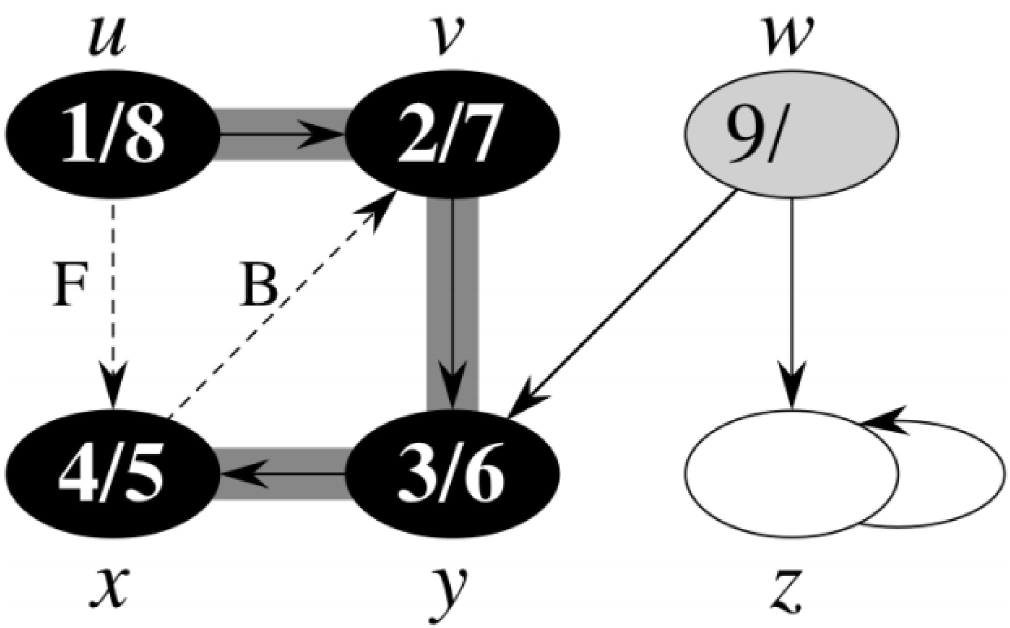
Busca em Profundidade: Exemplo



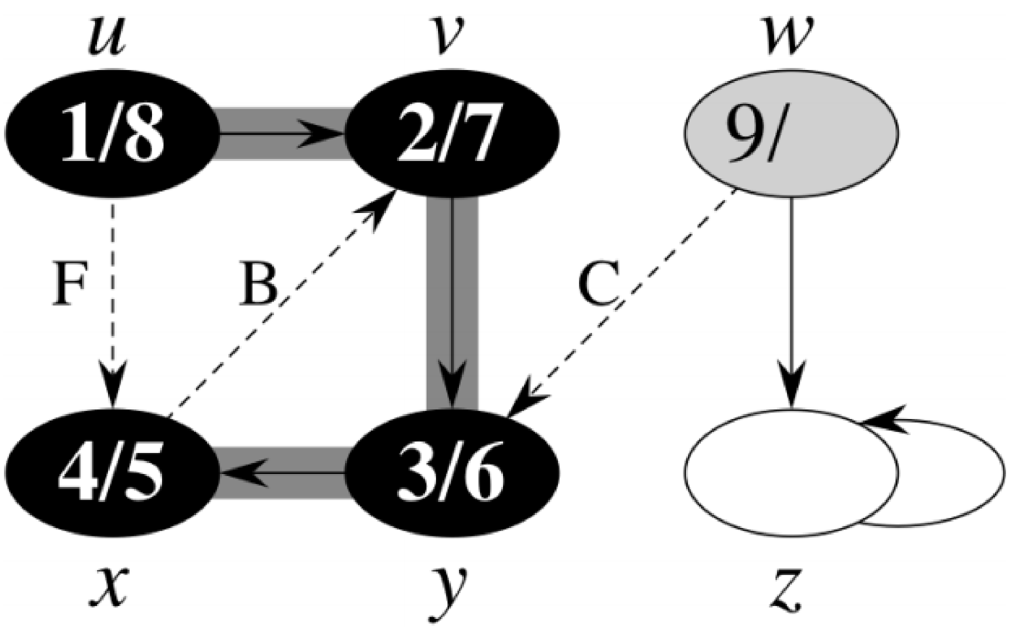
Busca em Profundidade: Exemplo



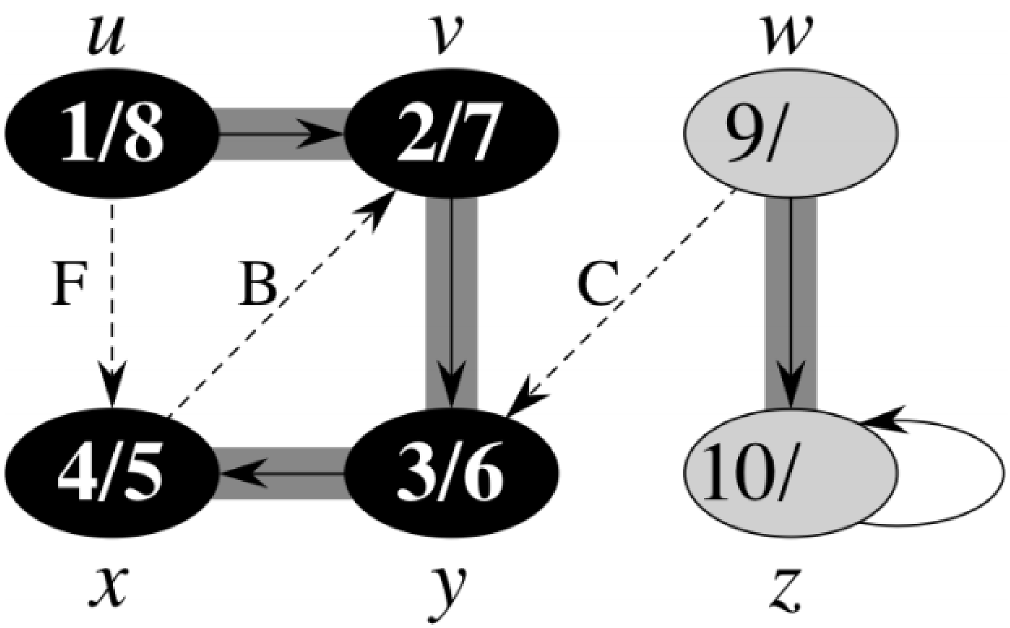
Busca em Profundidade: Exemplo



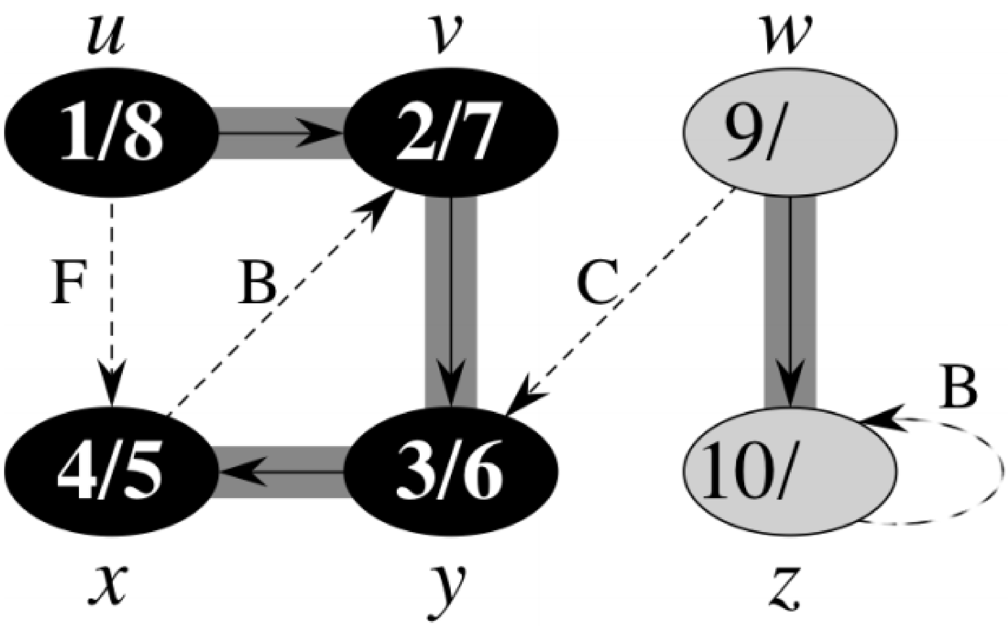
Busca em Profundidade: Exemplo



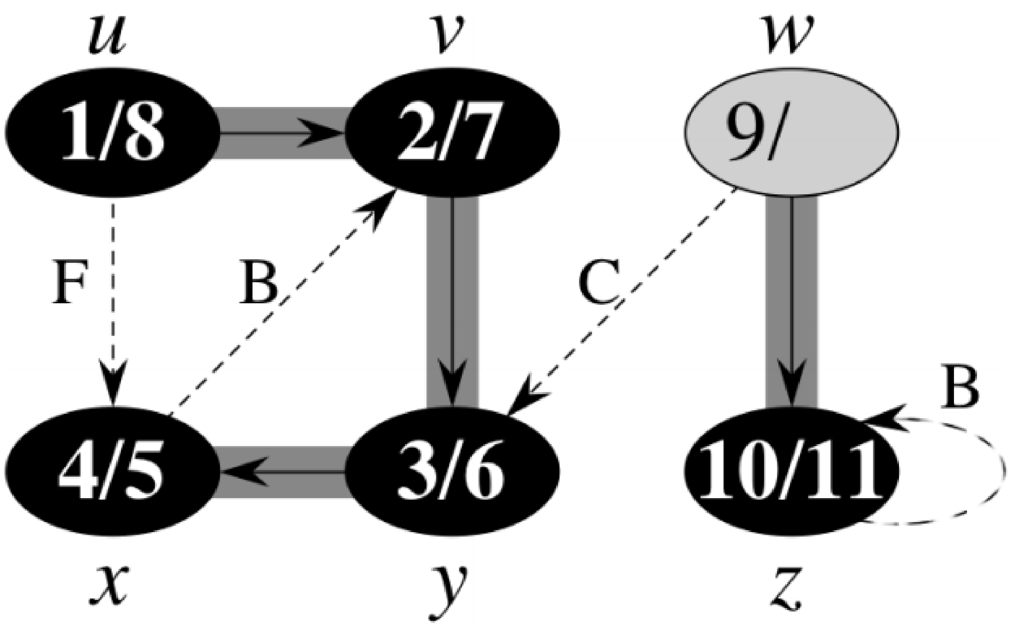
Busca em Profundidade: Exemplo



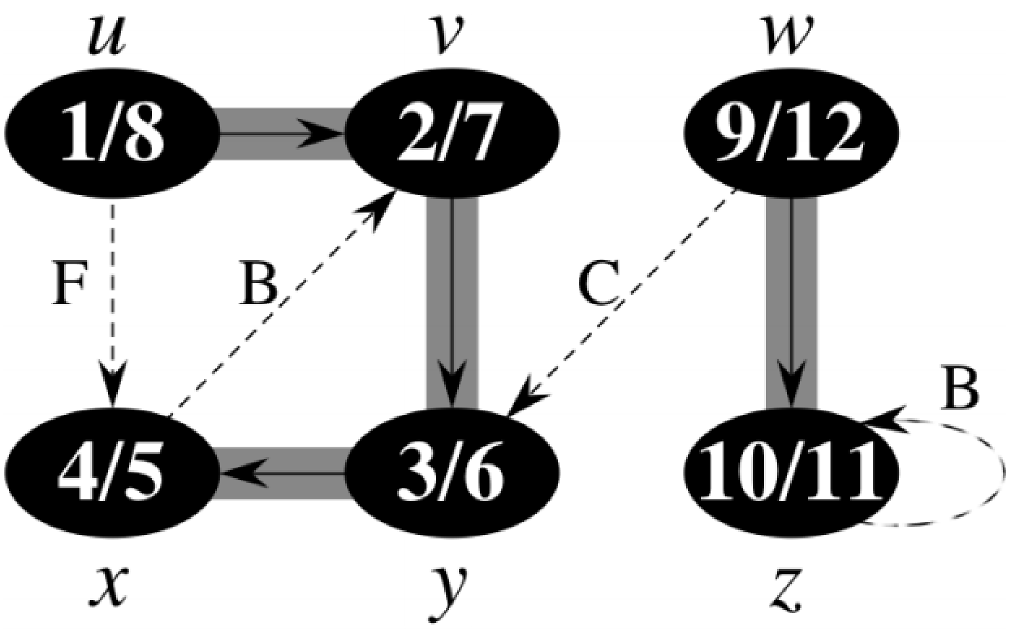
Busca em Profundidade: Exemplo



Busca em Profundidade: Exemplo



Busca em Profundidade: Exemplo



Algoritmo

```
dfs(G)
for cada vértice u em G.V
  u.cor = branco
  u.π = nil
tempo = 0
for cada vértice u em G.V
  if u.cor == branco
    dfs-visit(u)

dfs-visit(u)
tempo = tempo + 1
u.cor = cinza
u.d = tempo
for cada vértice v em u.adj
  if v.cor == branco
    v.π = u
    dfs-visit(v)
u.cor = preto
tempo = tempo + 1
u.f = tempo
```

Análise do tempo de execução

- Usamos a análise agregada
- Os loops nas linhas 1 a 3 e nas linhas 5 a 7 de dfs demoram tempo $\Theta(V)$, sem contar o tempo das chamadas a **dfs-visit**
- O procedimento **dfs-visit** é chamado exatamente uma vez para cada vértice, isto porque **dfs-visit** é chamado para os vértices brancos, e no início de **dfs-visit** o vértice é pintado de cinza
- Durante a execução de **dfs-visit(v)**, o laço nas linhas 4 a 7 é executado $|v.adj|$ vezes
- Desde que $\sum_{u \in V} |u.adj| = \Theta(A)$, então o custo total da execução das linhas 4 a 7 de **dfs-visit** é $\Theta(A)$
- Portanto, o tempo de execução do **dfs** é $\Theta(V + A)$

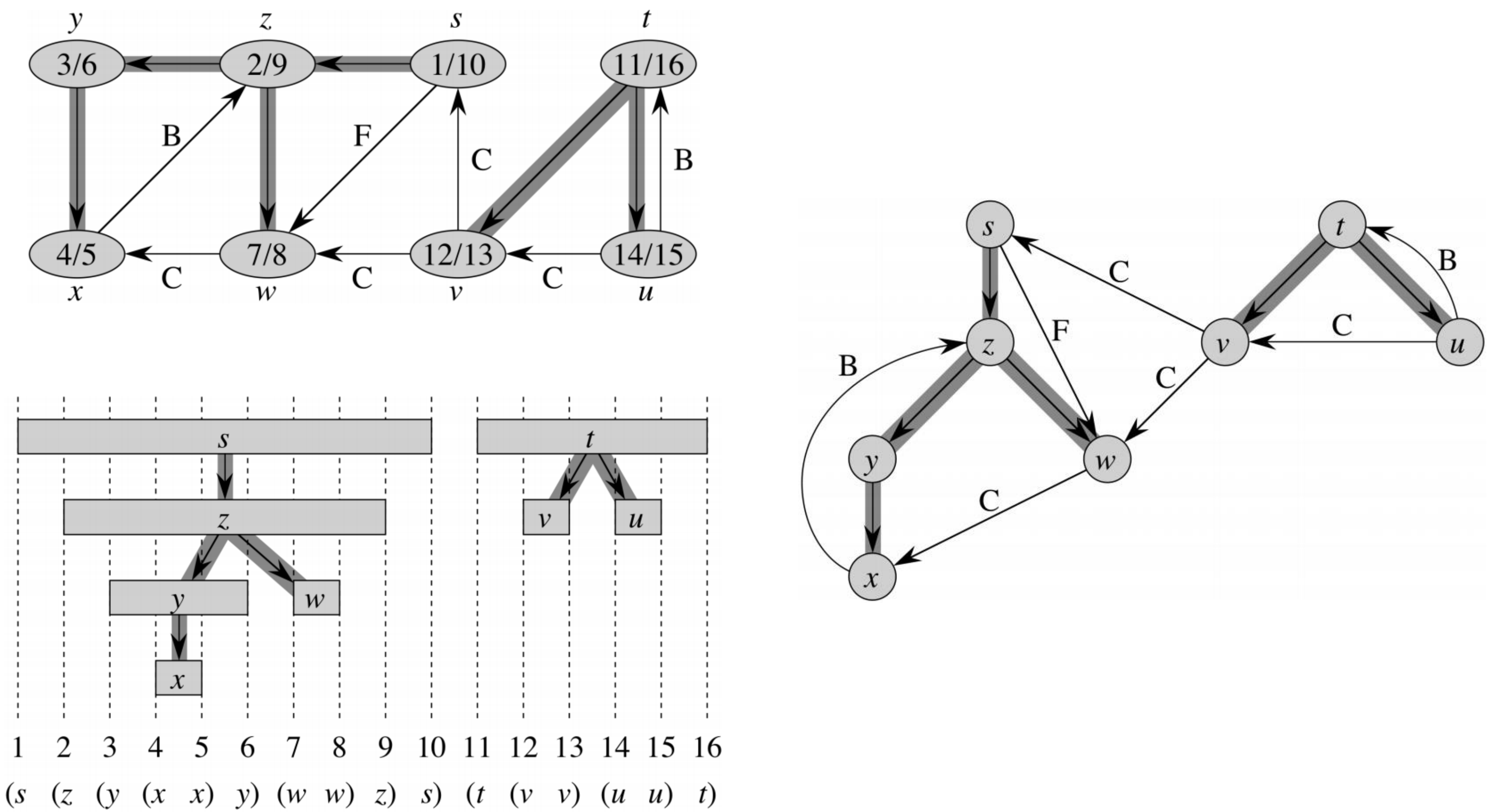
Floresta primeiro na profundidade

- O procedimento **dfs** constrói uma floresta primeiro na profundidade, contendo diversas árvores primeiro na profundidade
- Para um grafo G = (V, A), definimos o **subgrafo predecessor** de uma busca primeiro na profundidade de G como o grafo $G_\pi = (V, A_\pi)$ onde
  - $A_\pi = \{(v.\pi, v) \mid v \in V \text{ e } v.\pi \neq \text{NIL}\}$
- As arestas em A_π são **arestas da árvore**

Propriedades

- Teorema 22.7 (Teorema do parênteses)
  - Para dois vértices quaisquer *u* e *v*, exatamente uma das três condições a seguir é verdadeira
    - Os intervalos [*u.d, u.f*] e [*v.d, v.f*] são disjuntos e nem *u* e nem *v* são descendentes um do outro na floresta primeiro na profundidade
    - O intervalo [*u.d, u.f*] está contido inteiramente no intervalo [*v.d, v.f*] e u é descendente de *v* em uma árvore primeiro na profundidade
    - O intervalo [*v.d, v.f*] está contido inteiramente no intervalo [*u.d, u.f*] e v é descendente de u em uma árvore primeiro na profundidade
  - Veja a prova no livro

Propriedades



Classificação das arestas

- As arestas podem ser classificadas em termos da floresta primeiro na profundidade G_π
  - **Arestas da árvore** são as arestas na floresta primeiro na profundidade chamada G_π. Uma aresta (u, v) é uma **aresta da árvore** se *v* foi descoberto primeiro pela exploração da aresta (*u, v*);
  - **Arestas de retorno** são as arestas (*u, v*) que conectam um vértice *u* a um ancestral *v* na árvore primeiro na profundidade;
  - **Arestas para frente (ou de avanço)** são as arestas (*u, v*) que não são arestas da árvore e conectam o vértice *u* a um descendente *v* na árvore primeiro na profundidade.
  - **Arestas cruzadas** são todas as outras arestas.

Grafo acíclico

- A busca em profundidade pode ser usada para verificar se um grafo é acíclico ou contém um ou mais ciclos.
- Se uma aresta de retorno é encontrada durante a busca em profundidade em *G*, então o grafo tem ciclo.
- Um grafo direcionado *G* é acíclico se e somente se a busca em profundidade em *G* não apresentar arestas de retorno.

Exercícios

1) Mostre que em um grafo não direcionado G, toda aresta de G é uma aresta de árvores ou é uma aresta de retorno.

2) Mostre que a aresta (u,v) é

   a) Uma aresta de árvore se e somente se $u.d < v.d < v.f < u.f$.

   b) Uma aresta de retorno se e somente se $v.d < u.d < u.f < v.f$.

   c) Uma aresta cruzada se e somente se $v.d < v.f < u.d < u.f$

Bibliografia

- Thomas H. Cormen et al. Introduction to Algorithms. 3rd edition. Capítulo 22.3.

- Nivio Ziviani. Projeto de Algoritmos com Implementações em Pascal e C. 3a Edição Revista e Ampliada, Cengage Learning, 2010. Capítulo 7. Seção 7.3