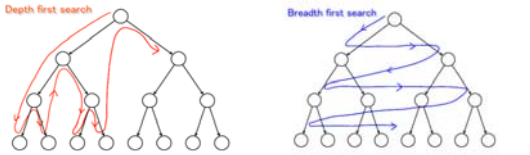
Teoria dos Grafos

Busca em Grafos

Busca em Grafos

- Objetivo: sistematicamente explorar todos os vértices e arestas de um grafo.
- Dois tipos de busca:
  - *–Busca em amplitude (Breadth First Search – BFS).*
  - *–Busca em profundidade (Depth First Search – DFS).*



Busca em Amplitude (BFS)

- Representa um dos mais simples algoritmos de busca em grafos.
- É usado como modelo para alguns algoritmos importantes:
  - Menor caminho.
  - Árvore de cobertura mínima.
- Similar ao caminhamento por níveis em árvores.
- Idéia: processa os vértices por níveis, começando por aqueles vértices mais próximos do vértice inicial *s* e deixando os vértices mais distantes para depois.

Busca em Amplitude (BFS)

- O algoritmo BFS pode ser resumido nos seguintes passos:
  1. Distinguir o vértice inicial *s*.
  2. Sistematicamente explorar as arestas do grafo para descobrir todos os vértices alcançáveis a partir de *s*.
  3. Computar a distância (menor número de arestas) de *s* para todos os vértices alcançáveis.
  4. Produzir uma árvore de amplitude cuja raiz é *s* e contém todos os vértices alcançáveis.
  5. Para todo vértice *v* alcançável a partir de *s*, o caminho na árvore de amplitude corresponde ao menor caminho de *s* para *v* no grafo.

Busca em Amplitude (BFS)

- O algoritmo descobre todos os vértices com distância *k* a partir de *s*, antes de descobrir os vértices com distância *k+1*.
- Para manter controle do processamento dos vértices, o algoritmo utiliza um esquema de 3 cores: branco, cinza e preto:
  - Todos os vértices começam com cor branca e depois podem mudar para cinza e, posteriormente, para preto.
  - Quando um vértice é visitado a primeira vez ele deixa de ser branco.
  - É necessário usar duas cores diferente do branco para garantir a sistemática da amplitude.

Algoritmo BFS

- Assumir que o grafo $G = (V, E)$ é representado com lista de adjacências.
- Para cada vértice no grafo, o algoritmo mantém estruturas auxiliares:
  - A variável *cor[u]* mantém a informação sobre a cor de cada vértice.
  - A variável $\pi[u]$ mantém a informação do predecessor de cada vértice. Quando não existe predecessor $\pi[u] = NIL$.
  - *d[u]* mantém o valor da distância entre o vértice inicial e *u*.
  - Uma fila *Q* com política FIFO para gerenciar a lista de vértices de cor cinza.

Algoritmo BFS

```
BFS(G, s)
for ∀u ∈ V[G] – {s} do
        cor[u] ← BRANCO
        d[u] ← ∞
        π[u] ← NIL
cor[s] ← CINZA
d[s] ← 0
π[s] ← NIL
Q ← {s}
while Q ≠ Ø do
        u ← Desenfileira[Q]
        for ∀v ∈ Adj[u] do
                if cor[v] = BRANCO then
                        cor[v] ← CINZA
                        d[v] ← d[u] + 1
                        π[v] ← u
                        Enfileira(Q, v)
        cor[u] ← PRETO
```


Algoritmo BFS

```
BFS(G, s)
for ∀u ∈ V[G] – {s} do
        cor[u] ← BRANCO
        d[u] ← ∞
        π[u] ← NIL
cor[s] ← CINZA
d[s] ← 0
π[s] ← NIL
Q ← {s}
while Q ≠ Ø do
        u ← Enfileira[Q]
        for ∀v ∈ Adj[u] do
                if cor[v] = BRANCO then
                        cor[v] ← CINZA
                        d[v] ← d[u] + 1
                        π[v] ← u
                        Enfileira(Q, v)
        cor[u] ← PRETO
```

Inicia variáveis auxiliares
Para cada um dos vértices,
Com exceção da origem

Algoritmo BFS

```
BFS(G, s)
for ∀u ∈ V[G] – {s} do
        cor[u] ← BRANCO
        d[u] ← ∞
        π[u] ← NIL
cor[s] ← CINZA
d[s] ← 0
π[s] ← NIL
Q ← {s}
while Q ≠ Ø do
        u ← Desenfileira[Q]
        for ∀v ∈ Adj[u] do
                if cor[v] = BRANCO then
                        cor[v] ← CINZA
                        d[v] ← d[u] + 1
                        π[v] ← u
                        Enfileira(Q, v)
        cor[u] ← PRETO
```

Inicia variáveis auxiliares da origem s e a fila Q

Algoritmo BFS

```
BFS(G, s)
for ∀u ∈ V[G] – {s} do
        cor[u] ← BRANCO
        d[u] ← ∞
        π[u] ← NIL
cor[s] ← CINZA
d[s] ← 0
π[s] ← NIL
Q ← {s}
while Q ≠ Ø do
        u ← Desenfileira[Q]
        for ∀v ∈ Adj[u] do
                if cor[v] = BRANCO then
                        cor[v] ← CINZA
                        d[v] ← d[u] + 1
                        π[v] ← u
                        Enfileira(Q, v)
        cor[u] ← PRETO
```

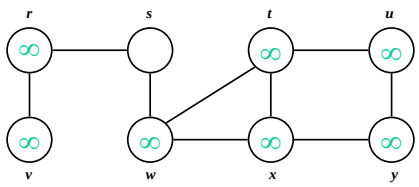
Se o vértice adjacente é branco, significa que ele nunca foi visitado. Deve ser pintado de CINZA e enfileirado para posterior processamento.

Algoritmo BFS

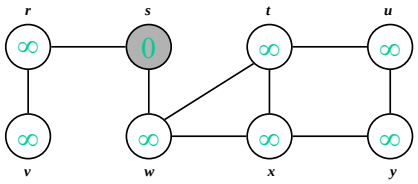
```
BFS(G, s)
for ∀u ∈ V[G] – {s} do
        cor[u] ← BRANCO
        d[u] ← ∞
        π[u] ← NIL
cor[s] ← CINZA
d[s] ← 0
π[s] ← NIL
Q ← {s}
while Q ≠ Ø do
        u ← Desenfileira[Q]
        for ∀v ∈ Adj[u] do
                if cor[v] = BRANCO then
                        cor[v] ← CINZA
                        d[v] ← d[u] + 1
                        π[v] ← u
                        Enfileira(Q, v)
        cor[u] ← PRETO
```

Quando todos os adjacentes de u forem processados, ele passa a ser PRETO

Exemplo BFS

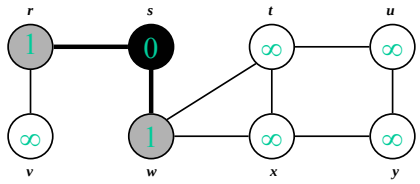


Exemplo BFS



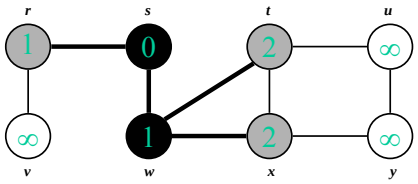
Q: | *s* |

Exemplo BFS



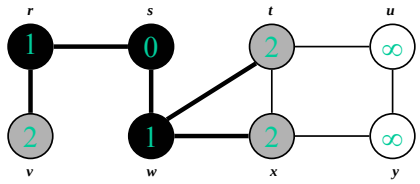
Q: | *w* | *r* |

Exemplo BFS



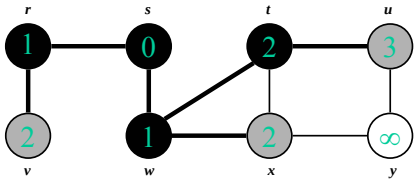
Q: | *r* | *t* | *x* |

Exemplo BFS



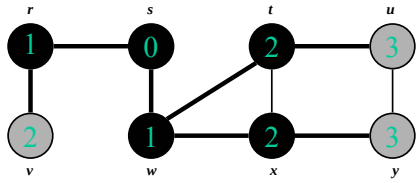
Q: | *t* | *x* | *v* |

Exemplo BFS



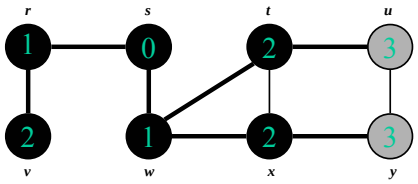
Q: | *x* | *v* | *u* |

Exemplo BFS



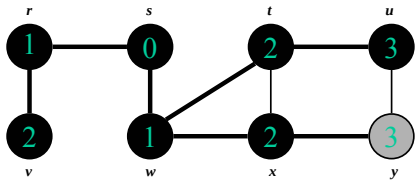
Q: | *v* | *u* | *y* |

Exemplo BFS



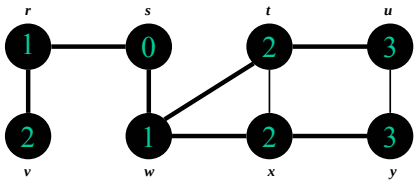
Q: | *u* | *y* |

Exemplo BFS



Q: | *y* |

Exemplo BFS



Q: Ø

Busca em Profundidade (DFS)

- DFS é uma generalização do caminhamento em pré-ordem em árvores.
- A idéia principal é buscar verticalmente, sempre que possível.
- Sempre que um novo vértice *v* é descoberto, ele deve ser explorado por completo.
- Quando *v* é totalmente explorado, fazer backtracking para o seu predecessor.

Busca em Profundidade (DFS)

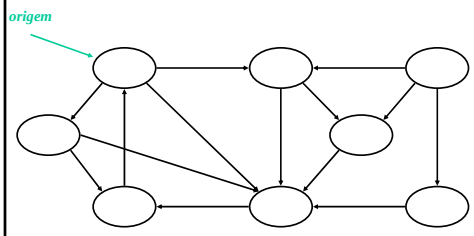
- Alguns detalhes sobre o algoritmo de DFS:
  - Sempre que um vértice *v* é descoberto, o valo de π (predecessor) é atualizado.
  - Pode ser formada uma floresta de árvores.
  - Cada vértice *v* tem duas marcas de tempo *d[v]* e f[v] que indicam, respectivamente, quando *v* foi primeiramente visitado (cor CINZA) e quando *v* foi totalmente explorado (cor PRETO).

Algoritmo DFS

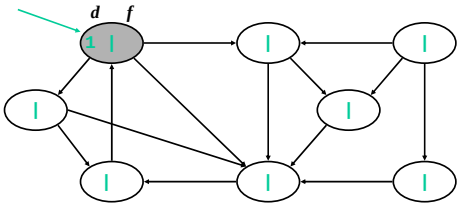
```
DFS(G)
for ∀u ∈ V[G]  do
        cor[u] ← BRANCO
        π[u] ← NIL
tempo ← 0
for ∀u ∈ V[G] do
        if cor[u] = BRANCO then
                VisitaDFS(u)
```

```
VisitaDFS(u)
cor[u] ← CINZA
d[u] ← tempo ← tempo + 1
for ∀v ∈ Adj[u] do
        if cor[v] = BRANCO then
                π[v] ← u
                VisitaDFS(v)
cor[u] ← PRETO
F[u] ← tempo ← tempo + 1
```

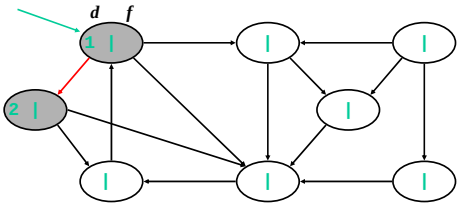
Exemplo DFS



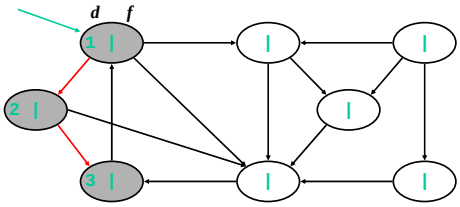
Exemplo DFS



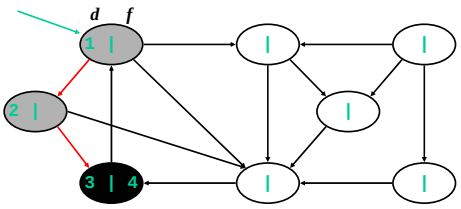
Exemplo DFS



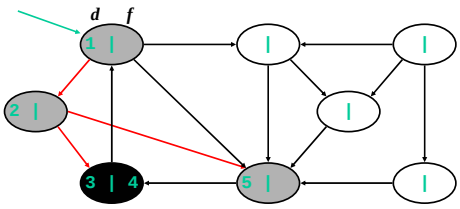
Exemplo DFS



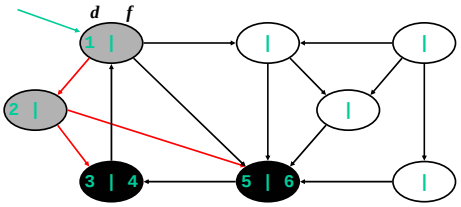
Exemplo DFS



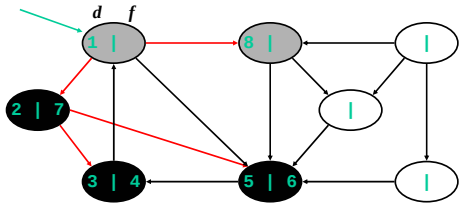
Exemplo DFS



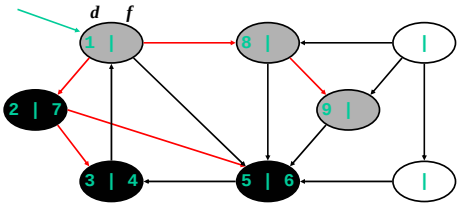
Exemplo DFS



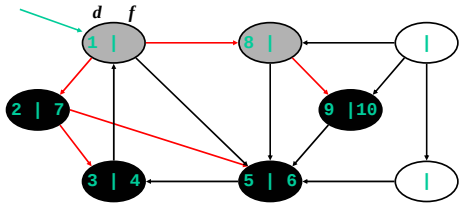
Exemplo DFS



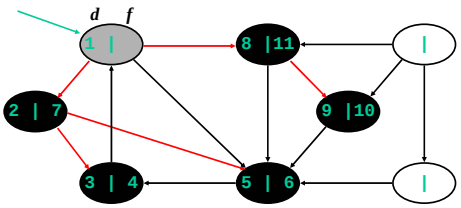
Exemplo DFS



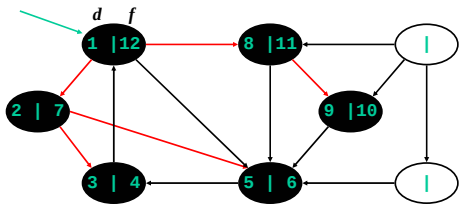
Exemplo DFS



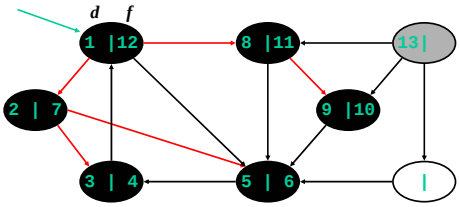
Exemplo DFS



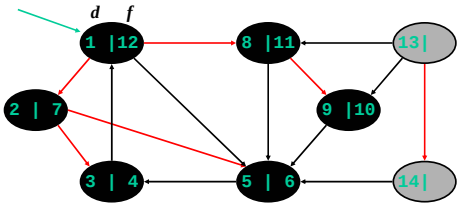
Exemplo DFS



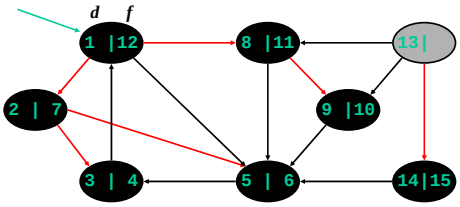
Exemplo DFS



Exemplo DFS



Exemplo DFS



Exemplo DFS