Algoritmos e Estruturas de Dados

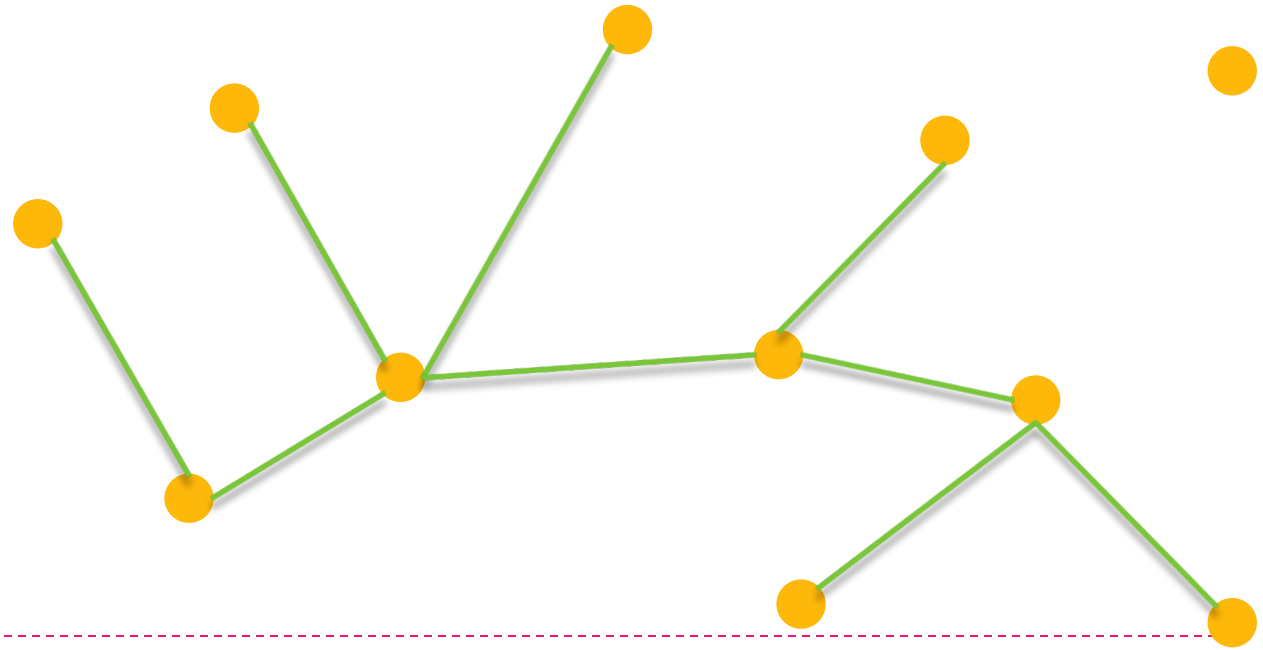
Grafos – 2021

Grafo - Definição

- Um grafo G=(V,E) é um par de dois conjuntos não vazios em que V é um conjunto finito e E é uma relação binária em V (isto é, $E \subseteq VxV$) tal que:
  - Um arco $e \in E$ liga dois vértices
  - Um vértice $v \in V$ pode estar ligado a qualquer número de outros vertices

vértice

arco



Rede

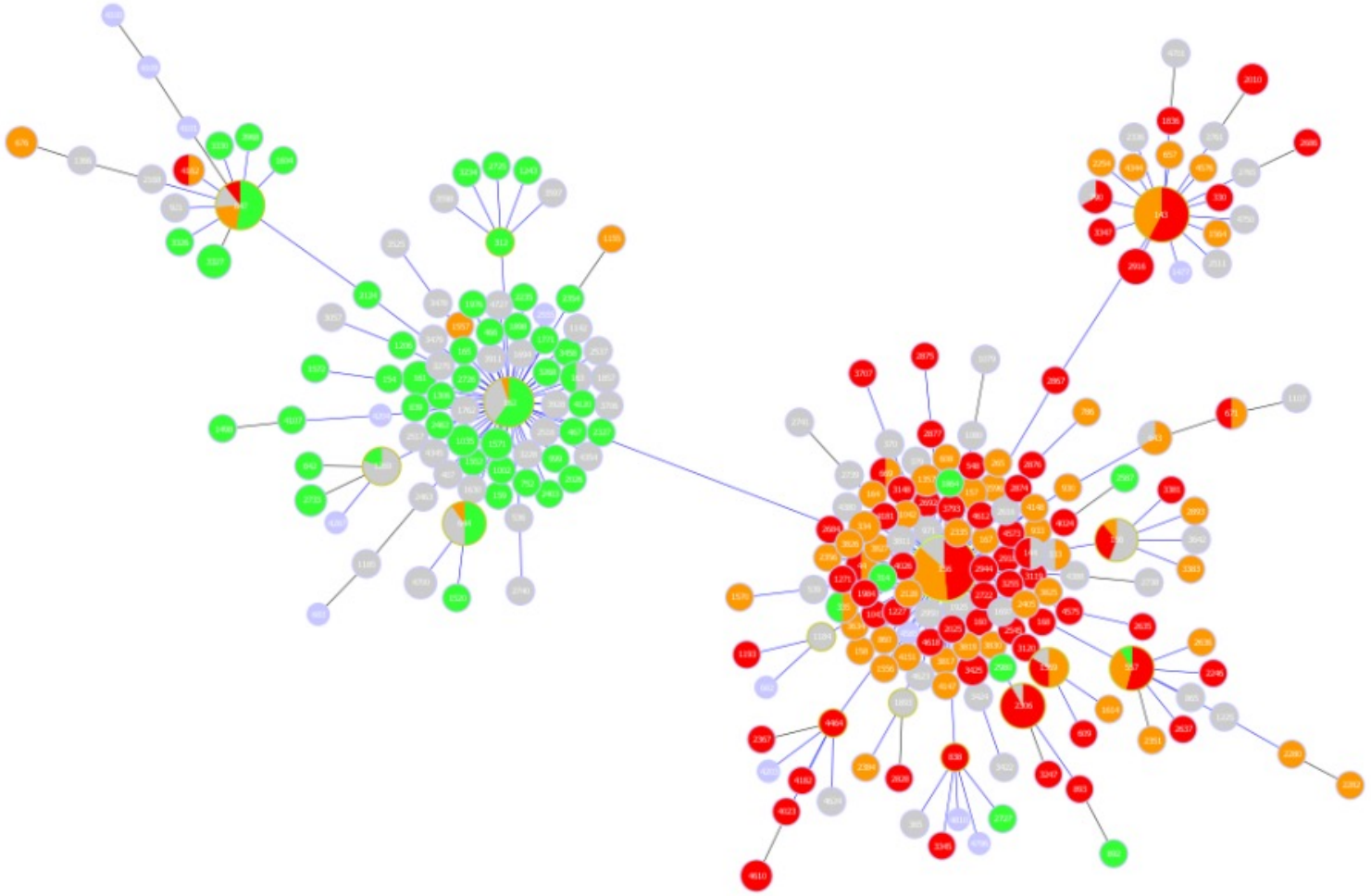
- Uma rede é
  - Um padrão de interconeções entre um conjunto de entidades.

Mais precisamente….

- Uma coleção de objetos na qual alguns pares de objetos estão conectados através de links
- As redes consistem de dois blocos de construção principais:
  - Vertices
  - Arcos
- As redes podem ser representadas por grafos.

Networks	Vertices	Edges
Twitter	Users	Follower/Following; Mentions; Replies
Facebook	Friends	Friendship Relations
Skype	Contacts	Messages/Conversations
Traffic	Cross	Roads

Exemplo de Aplicação de Grafos

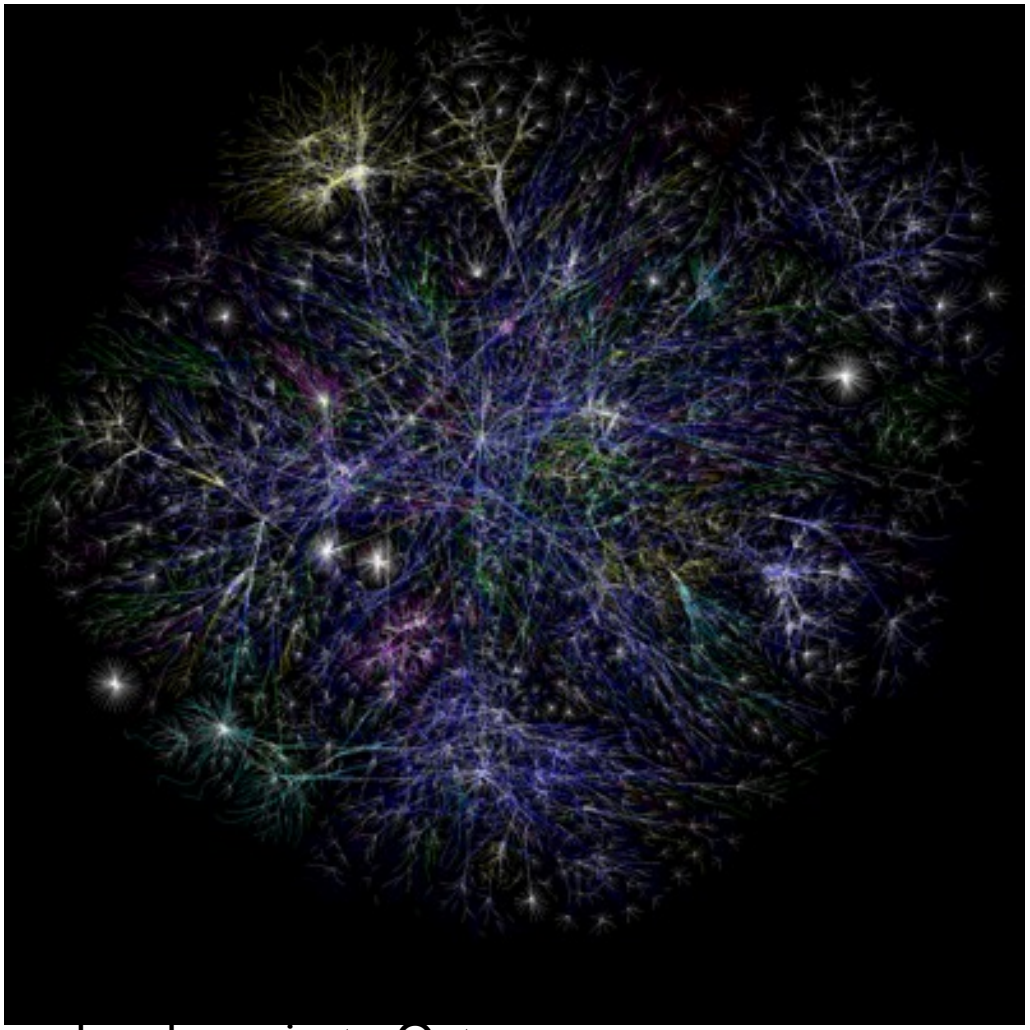


S. pneumoniae CC156 distinguida por resistência à penincilina:
http://www.phyloviz.net/wiki

Exemplo de Aplicação de Grafos

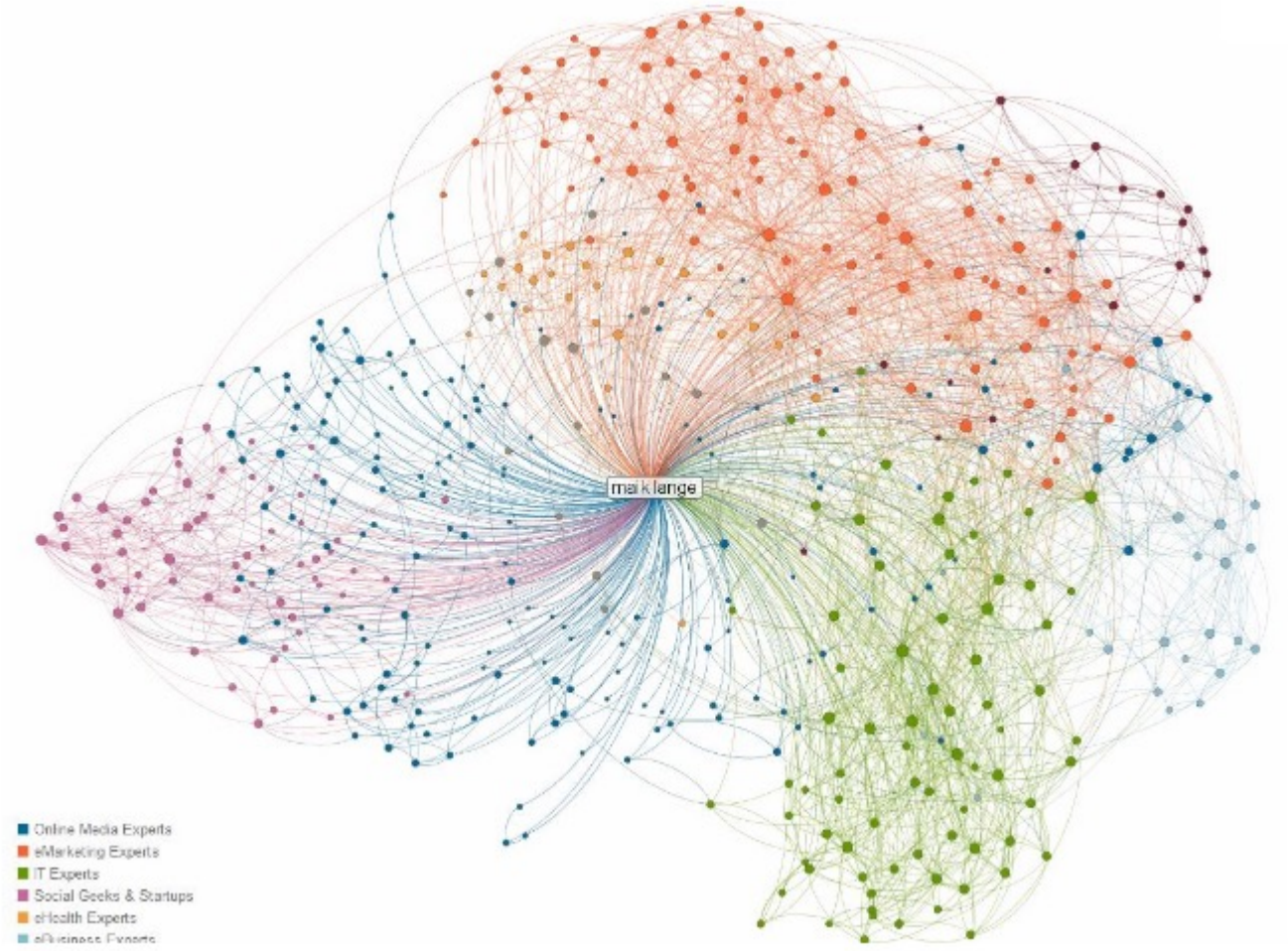
- Rede de metro de Lisboa

Exemplo de Aplicação de Grafos



A Internet mapeada pelo projecto Opte

Exemplo de aplicação de grafos

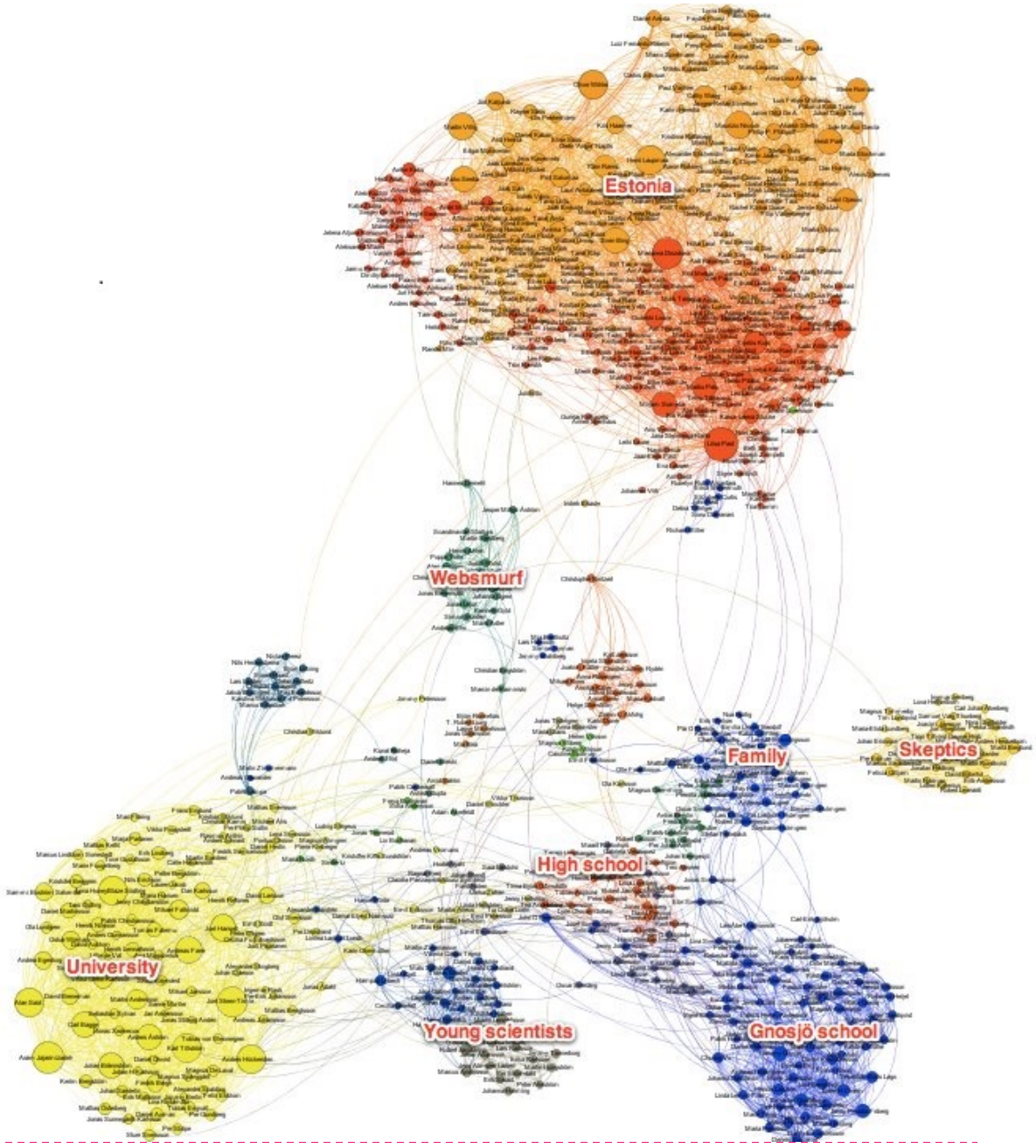


Comunity-finding: exemplo do LinkedIn

Structural Analysis Example:

unclecj.blogspot.com

Imagem gerada pela Aplicação Gephi



Conectividade de redes

- As redes estão conectadas a dois níveis
  - Ao nível estrutural
    - Quem está ligado a quem
  - Ao nível comportamental
    - Cada ação individual tem consequências implícitas para os resultados de todos no sistema
- Para analisar conectividade da rede e inferir propriedades, baseamo-nos em
  - Teoria de Grafos
  - Teoria de Jogos

Análise ao nível estrutural

- *Small-World Phenomenon*
- Detecção de ligações fortes/fracas
- Centralidade dos indivíduos
  - Determinar a importância relativa de um indivíduo na rede.
    - Betweenness centrality, Closeness centrality, degree centrality…
    - Capturam diferentes aspectos da relevância de um indivíduo.
  - Detecção de comunidades

Análise ao nível comportamental

- Como é que um grupo de indivíduos pode simultâneamente escolher em agir, sabendo que o resultado das suas acções depende nas decisões tomados por todos os indivíduos?

Traffic Network Example:

Alguns problemas de processamento de grafos

- Caminho: Existe algum caminho entre s e t?
- Caminho mais curto: Qual é o caminho mais curto entre s e t?
- Ciclo: Existe um ciclo no grafo?
- Circuito de Euler: Existe um ciclo no grafo que utiliza cada arco exactamente uma só vez?
- Circuito de Hamilton: Existe um ciclo que utiliza cada vértice exactamente uma só vez?
- Conectividade: Existe alguma forma de conectar todos os vértices?
- MST: Qual é a melhor forma de conectar todos os vértices, com o menor custo?
- Biconectividade: Existe algum vértice que, se for removido, desconecta o grafo?
- Planaridade: Podemos desenhar um grafo num plano sem cruzar arcos?
- Isomorfismo: Será que duas matrizes de adjacências representam o mesmo grafo?

Exemplos de aplicação

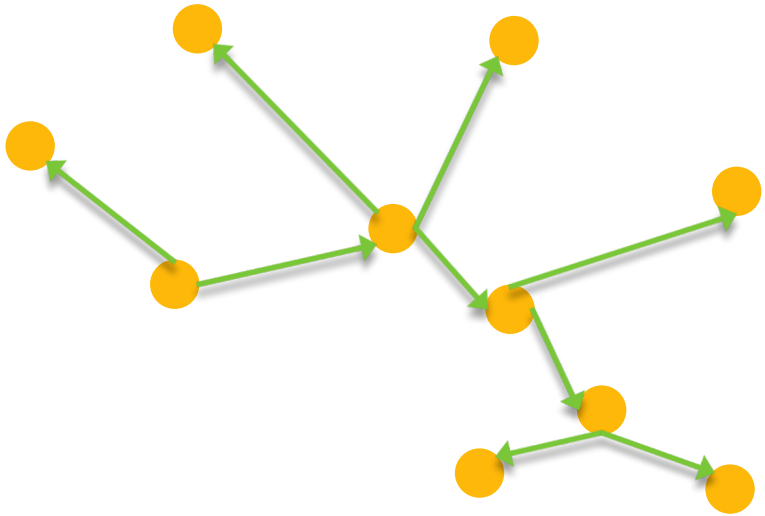
- Betweeness Centrality de um nó é:
  - O número de caminhos mais curtos entre todos os pares de vértices, que passem através daquele nó
    - Maior é o valor de betweeness centrality para um vértice

      =>

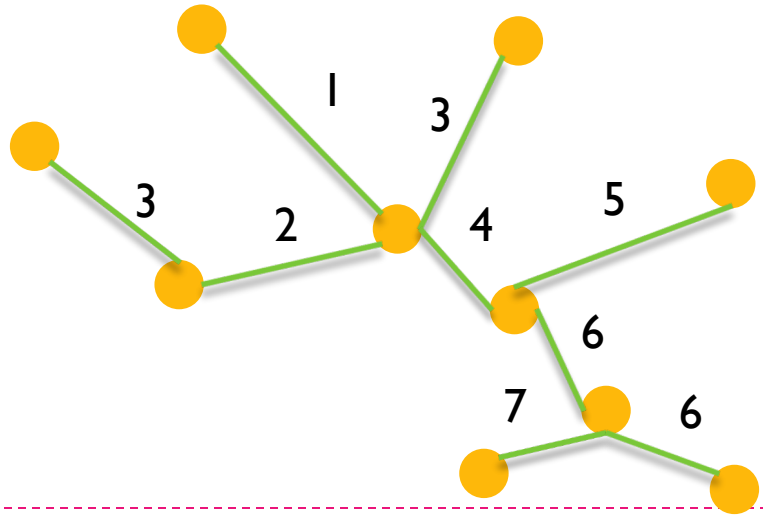
      Maior é a sua importância como um intermediário entre dois grupos que de outra forma poderiam estar "separados"

Algumas variações

- Grafo Orientado
  - Os grafos podem ter orientação



- Grafo Pesado
  - Os arcos podem ter peso (custo)



Exemplos

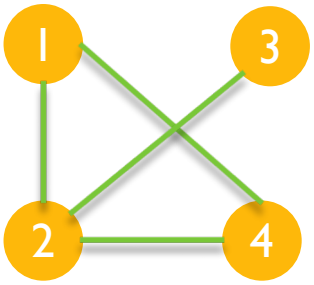
Redes	Vertices	Arcos
Twitter	Users	Follower (orientado)
Facebook	Friends	Friendship Relations (não orientado)
Skype	Contacts	Messages/Conversations (não orientado)
Traffic	Cruzamentos	Estradas (orientado)

Small-World Phenomenon: o quanto estamos perto uns dos outros?

	Milgram Exp.	MSN	Facebook	Twitter	Twitter
Vertex	pessoa	user	user	user	user
Link	Pessoa seleccionada	message exchange (conversation)	friendship	follower	mention
Symmetric	no	yes	yes	no	no
Avg. dist	6.2	6.6	4.74	4.12	6.63

Representação

Não orientado



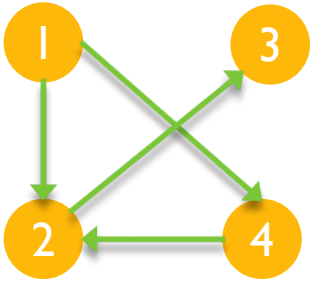
Listas

1	2	4	/	
2	1	3	4	/
3	2	/		
4	1	2	/	

Matriz de Adjacências

	1	2	3	4
1	0	1	0	1
2	1	0	1	1
3	0	1	0	0
4	1	1	0	0

Orientado



1	2	4	/
2	3	/	
3	/		
4	2	/	

	1	2	3	4
1	0	1	0	1
2	0	0	1	0
3	0	0	0	0
4	0	1	0	0

Grafos: definição e representação

- Grafo definido por um conjunto V de vértices e um conjunto E de arcos, $G = (V, E)$
  - Arcos representam ligações entre pares de vértices $E \subseteq V \times V$
    - Grafo esparso: $|E| << |V \times V|$
  - Representação dos arcos
    - **Matriz de adjacências:** arcos representados por matriz
      - Para grafos densos
    - **Listas de adjacências:** arcos representados por listas
      - Para grafos esparsos
- Grafos podem ser orientados ou não orientados
  - Existência (ou não) da noção de orientação nos arcos

Grafos: definição e representação

- Listas de adjacências
  - Grafos não orientados
    - Tamanho das listas é 2 $|E|$
  - Grafos orientados
    - Tamanho das listas é $|E|$
  - Tamanho total das listas de adjacências é $O(V+E)$
- Matriz de adjacências: para qualquer grafo $\Theta(V^2)$
- Grafos pesados
  - Função de pesos: w: $E \rightarrow \mathbf{R}$
    - Permite associar um peso a cada arco

Matriz de adjacências - Vantagens

- Representação mais adequada quando:
  - Há espaço disponível
  - Grafos são densos
  - Algoritmos requerem mais de V^2 operações
- Adição e remoção de arcos é feita de forma eficiente
- Fácil evitar existência de arcos paralelos (repetidos)
- Fácil determinar se dois vértices estão ou não ligados

Matriz de adjacência - Incovenientes

- Grafos esparsos de grande dimensão requerem espaço de memória proporcional a V^2
  - Neste caso, a simples inicialização do grafo (proporcional a V^2) pode dominar o tempo de execução global do algoritmo
- Para o caso de grafos muito esparsos, mas com um número muito elevado de vértices, pode nem sequer existir memória suficiente para armazenar a matriz

Lista de adjacências - Vantagens

- Inicialização é proporcional a *V*
- Utiliza sempre espaço proporcional a *V+E*
  - Adequado para grafos esparsos
  - Algoritmos que assentem na análise de arcos em grafos esparsos.
- Adição de arcos é feita de forma eficiente

Lista de adjacências - Incovenientes

- Arcos paralelos e adjacência entre vértices
  - Requer que se pesquise as listas de adjacências, o que pode levar um tempo proporcional a *V*
- Remoção de arcos
  - Pode levar um tempo proporcional a *V*
- Não aconselhável para
  - Grande utilização de remoção de arcos

Desempenho das representações

	Matriz de adjacências	Lista de Adjacências
Espaço	$O(V^2)$	$O(V + E)$
Inicialização	$O(V^2)$	O(V)
Inserir arco	O(1)	O(1)
Procurar arco	O(1)	O(V)
Remover arco	O(1)	O(V)

Procura

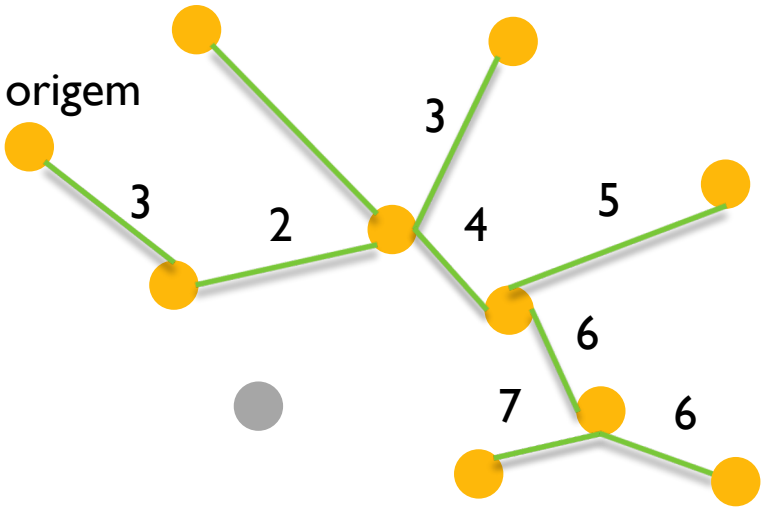
- Algumas propriedades simples em grafos são fáceis de determinar, independentemente da ordem pela qual se examinam os arcos
  - Ex: grau de todos os vértices.
- Outras propriedades estão associadas a caminhos, pelo que se torna necessário identificá-las através de pesquisa feita de vértice em vértice ao longo dos arcos
  - A maioria dos algoritmos em grafos que vamos estudar utilizam este modelo abstracto básico
- Torna-se então necessário analisar o essencial dos algoritmos de *procura em grafos* e suas propriedades estruturais

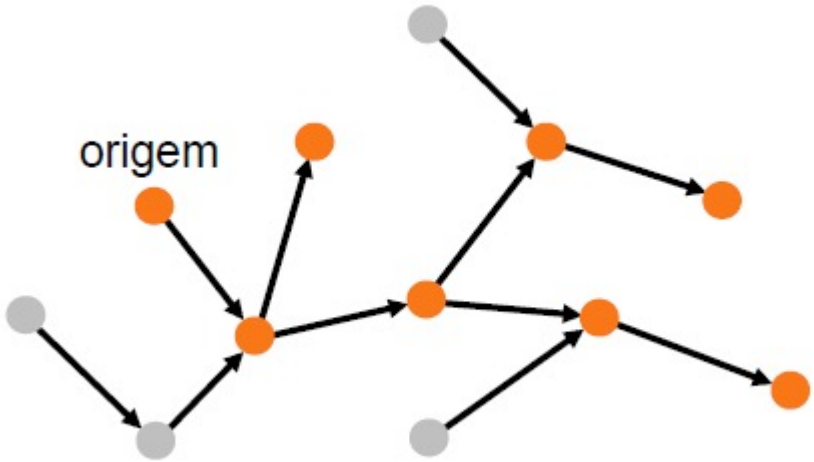
Procura

- Procurar em grafos é equivalente a percorrer labirintos
  - Necessário marcar pontos já visitado
  - Ser-se capaz de recuar, num caminho efectuado, até ao ponto de partida
- Os vários algoritmos de procura em grafos limitam-se a executar uma determinada estratégia de procura em labirintos
  - Procura em profundidade primeiro (*DFS – Depth-first-search*)
  - Admite duas implementações: recursiva e com uso de pilha explícita
  - Substituindo a pilha por uma fila FIFO, transforma-se em procura em largura primeiro (*BFS Breadth-first-search*)

Procura

- Dado um vértice origem/fonte
- Visitar todos os vértices atingíveis a partir da origem
  - Todos os vértices que estão em qualquer caminho do grafo que comece na origem





- A ordem pela qual os vértices são visitados depende do tipo de procura

Procura em Largura Primeiro (BFS)

- Visita os vértices por ordem da sua distância à origem
- Vértices mais próximos são visitados em primeiro lugar
  - Vértices não atingíveis a partir da origem, não são visitados

- Dados $G = (V, E)$ e vértice fonte s, BFS explora sistematicamente vértices de G para descobrir todos os vértices atingíveis a partir de s
  - Cálculo da distância: menor número de arcos de s para cada vértice atingível
  - Identificação de árvore BF: caminho mais curto de s para cada vértice atingível v
- Fronteira entre nós descobertos e não descobertos é expandida uniformemente
  - Vértices à distância k descobertos antes de qualquer vértice à distância $k+1$

Procura em Largura Primeiro (BFS)

- Implementação:
- Dado um vértice v
  - v.color - cor do vértice *v*: branco, cinzento e preto
    - branco: não visitado
    - cinzento: já visitado mas algum dos adjacentes não visitado ou procura em algum dos adjacentes não terminada
    - preto: já visitado e procura nos adjacentes já terminada
  - v.π: predecessor de *v* na árvore BF
  - *v*.d: tempo de descoberta de v

Procura em Largura Primeiro (BFS)

- Árvores BF: (sub-grafo de G)
  - Vértices atingíveis a partir de s
  - Arcos que definem a relação predecessor da BFS

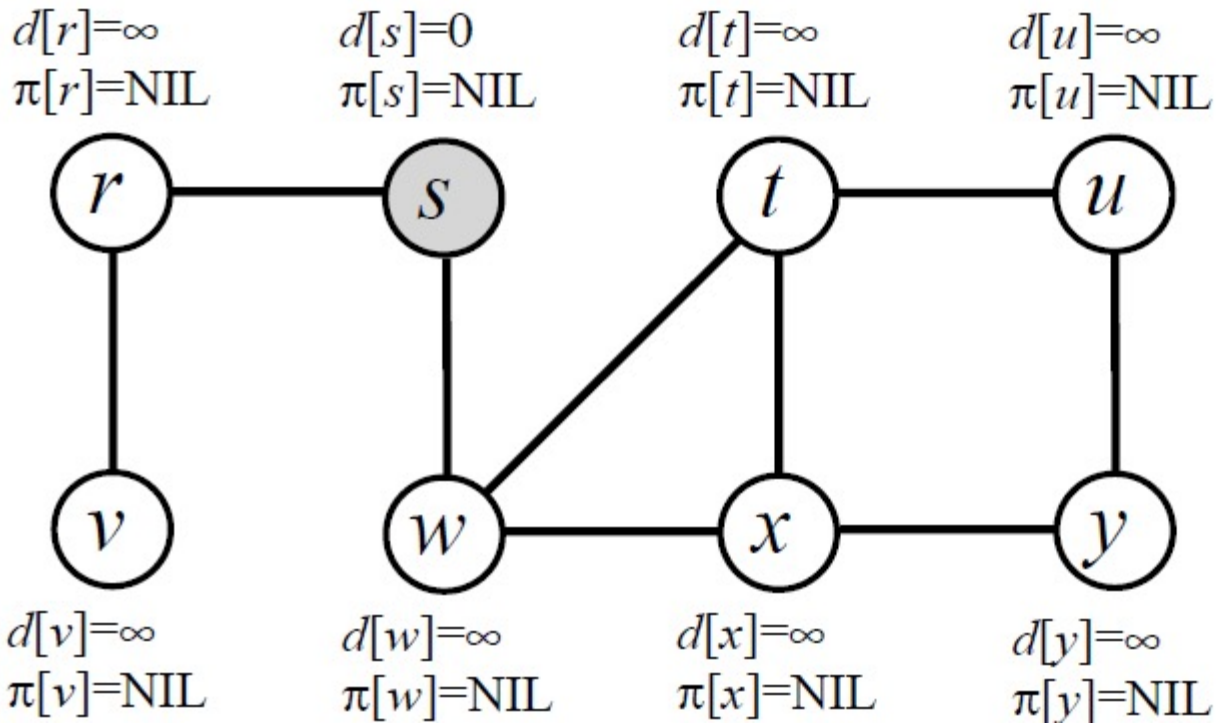
$$G_\pi (V_\pi, E_\pi)$$

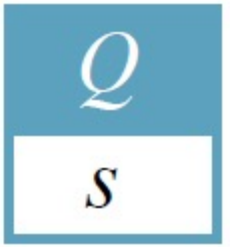
$$V_\pi = \{v \in V : \pi[v] \neq NIL\} \cup \{s\} \qquad E_\pi = \{(\pi[v], v) \in E : v \in V - \{s\}\}$$

BFS – Breath First Search

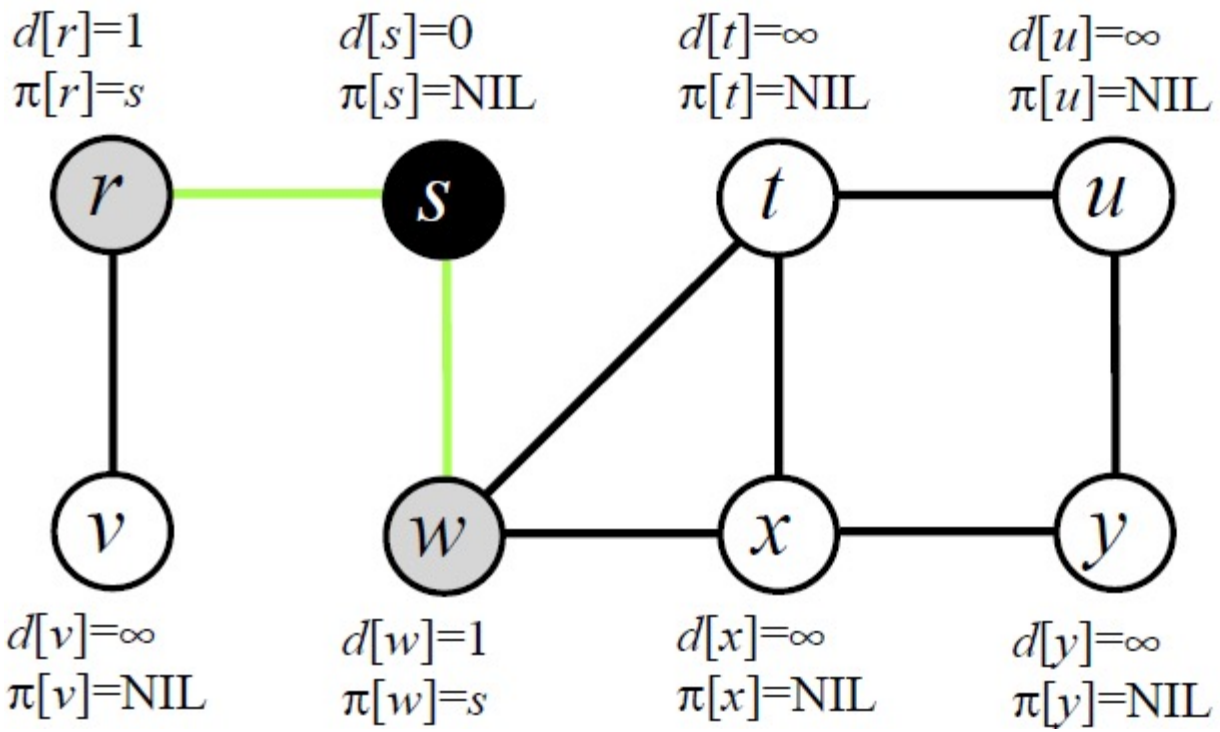
```
BFS(G, s)
 1  for each vertex u ∈ G.V − {s}
 2      u.color = WHITE
 3      u.d = ∞
 4      u.π = NIL
 5  s.color = GRAY
 6  s.d = 0
 7  s.π = NIL
 8  Q = ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11      u = DEQUEUE(Q)
12      for each v ∈ G.Adj[u]
13          if v.color == WHITE
14              v.color = GRAY
15              v.d = u.d + 1
16              v.π = u
17              ENQUEUE(Q, v)
18      u.color = BLACK
```

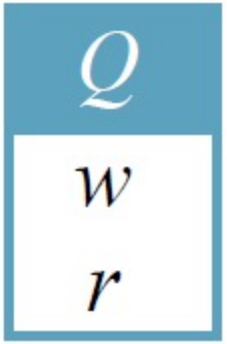
Exemplo BFS


d[r]=∞
π[r]=NIL
d[s]=0
π[s]=NIL
d[t]=∞
π[t]=NIL
d[u]=∞
π[u]=NIL
r
s
t
u
v
w
x
y
d[v]=∞
π[v]=NIL
d[w]=∞
π[w]=NIL
d[x]=∞
π[x]=NIL
d[y]=∞
π[y]=NIL

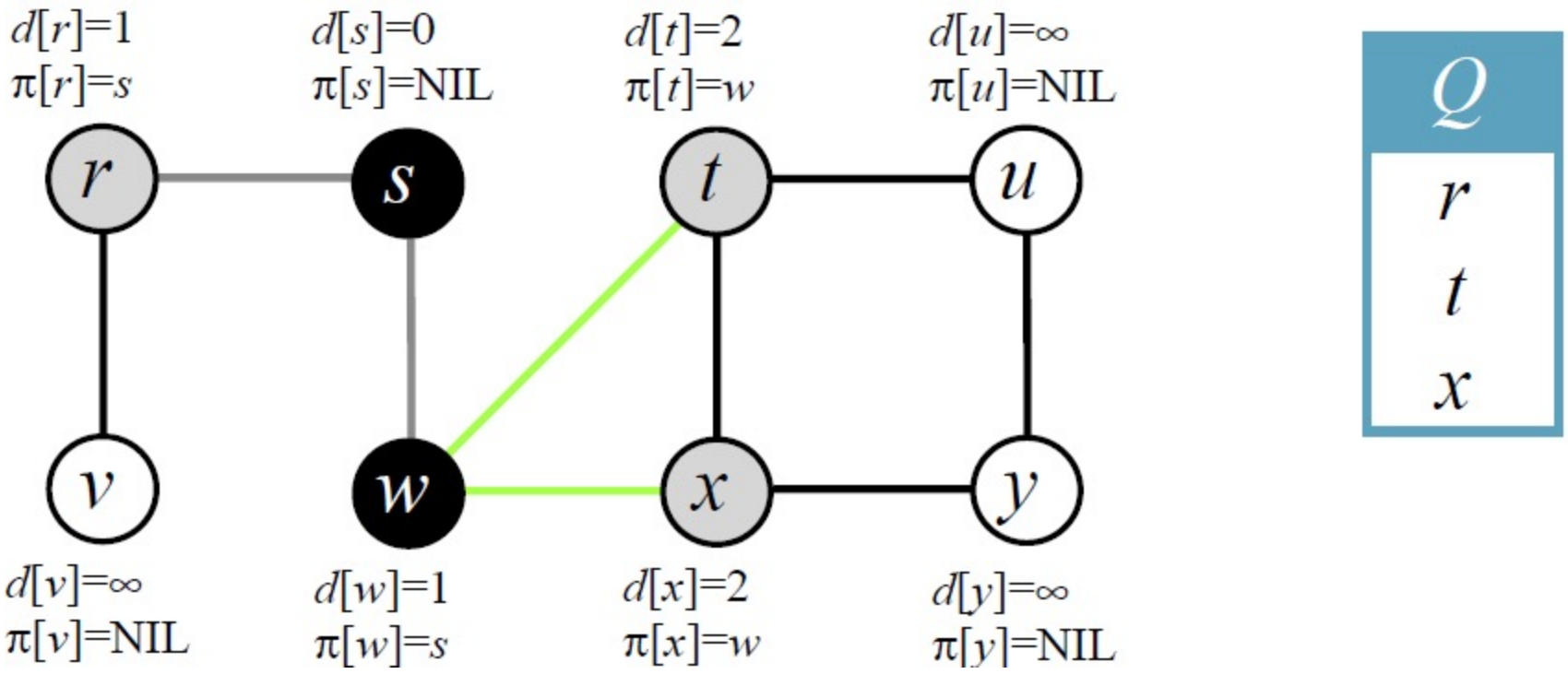

Q
s

Exemplo BFS

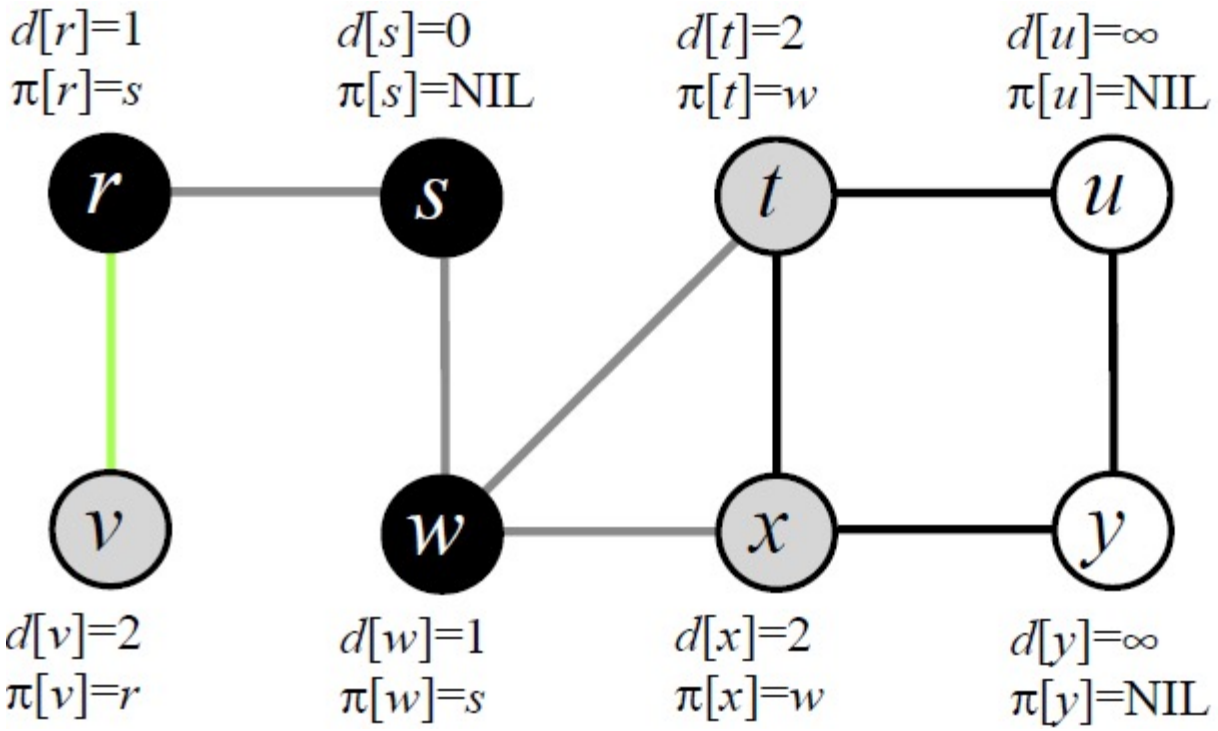

d[r]=1
π[r]=s
d[s]=0
π[s]=NIL
d[t]=∞
π[t]=NIL
d[u]=∞
π[u]=NIL
r
s
t
u
v
w
x
y
d[v]=∞
π[v]=NIL
d[w]=1
π[w]=s
d[x]=∞
π[x]=NIL
d[y]=∞
π[y]=NIL


Q
w
r

Exemplo BFS

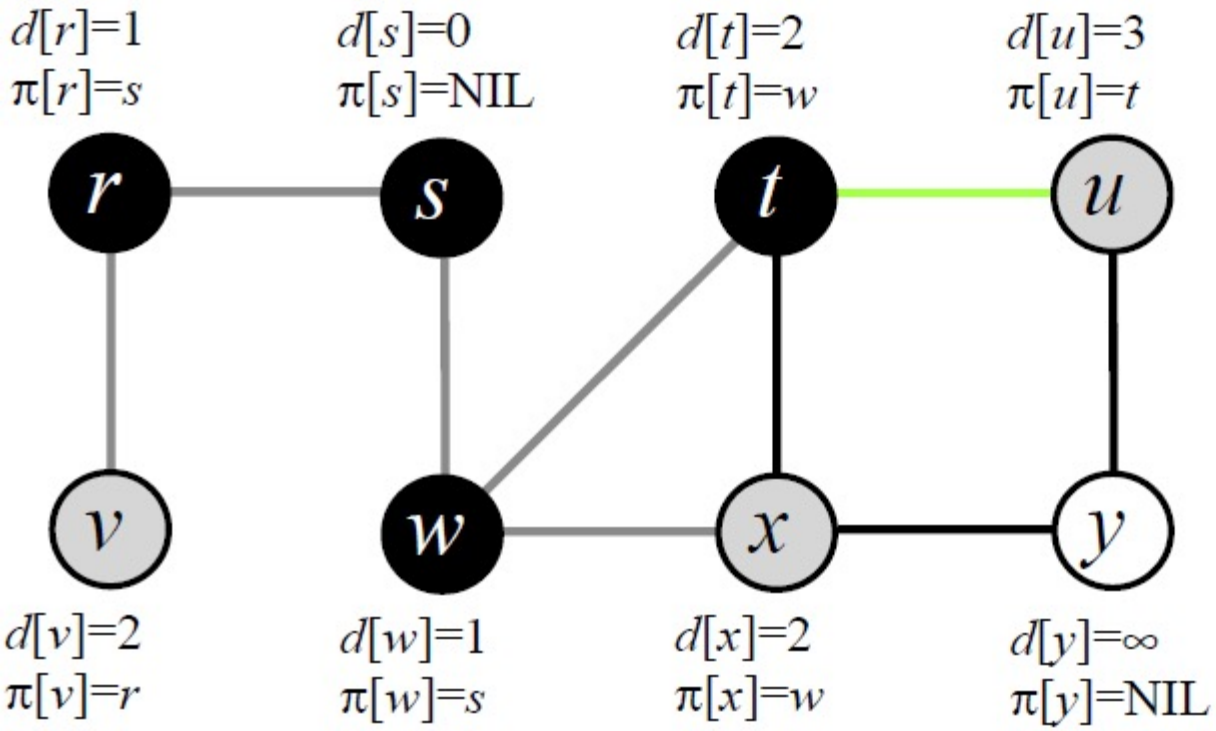

d[r]=1
π[r]=s
d[s]=0
π[s]=NIL
d[t]=2
π[t]=w
d[u]=∞
π[u]=NIL
r
s
t
u
v
w
x
y
d[v]=∞
π[v]=NIL
d[w]=1
π[w]=s
d[x]=2
π[x]=w
d[y]=∞
π[y]=NIL
Q
r
t
x

Exemplo BFS



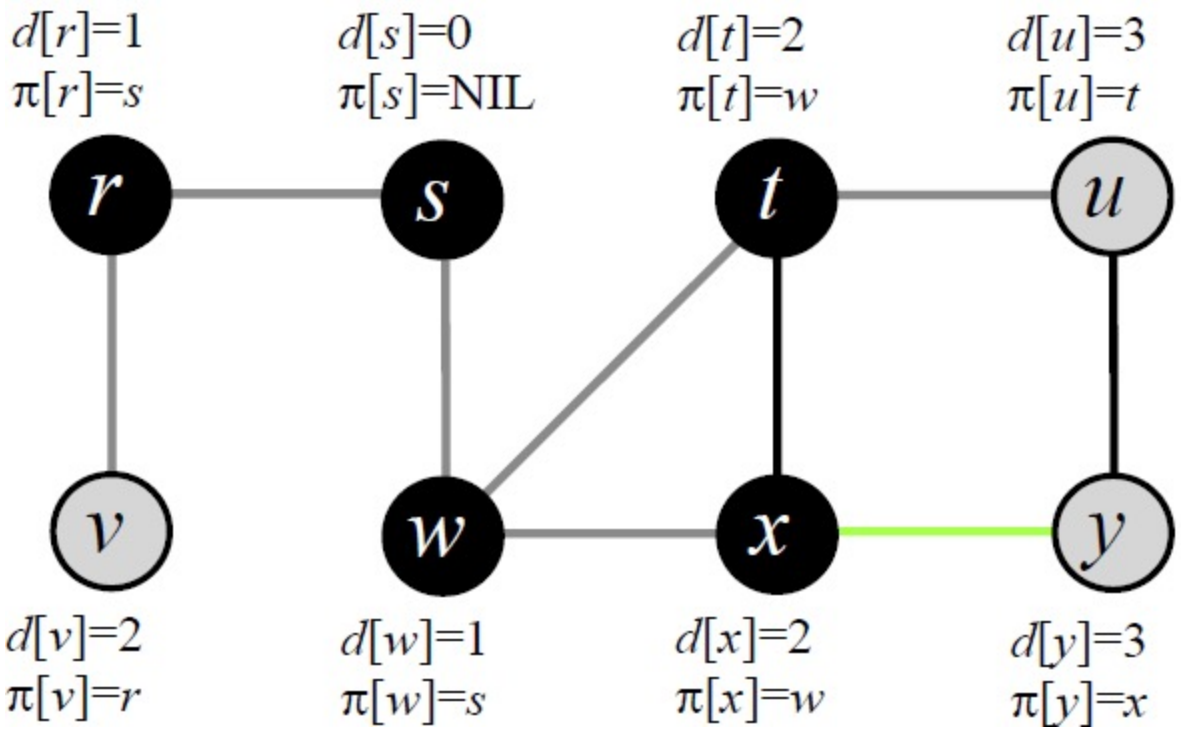
Q
t
x
v

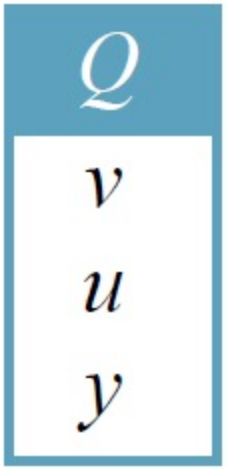
Exemplo BFS



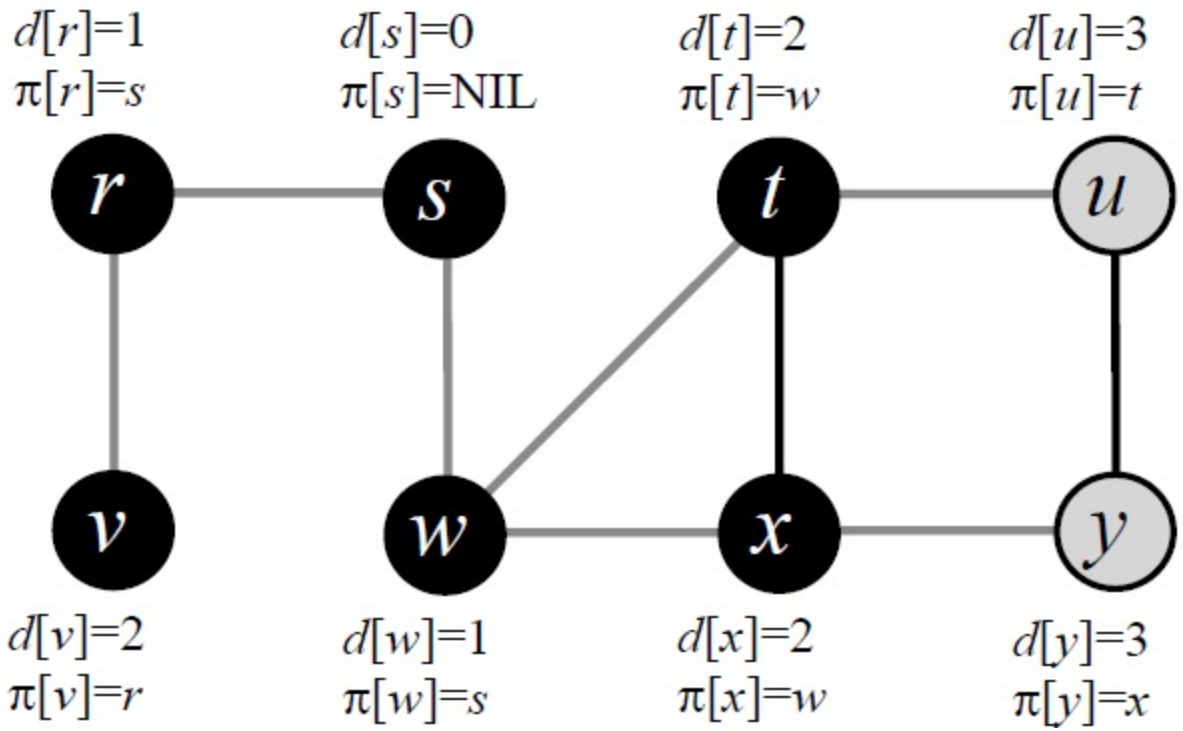
Q
x
v
u

Exemplo BFS


d[r]=1
π[r]=s
d[s]=0
π[s]=NIL
d[t]=2
π[t]=w
d[u]=3
π[u]=t
r
s
t
u
v
w
x
y
d[v]=2
π[v]=r
d[w]=1
π[w]=s
d[x]=2
π[x]=w
d[y]=3
π[y]=x

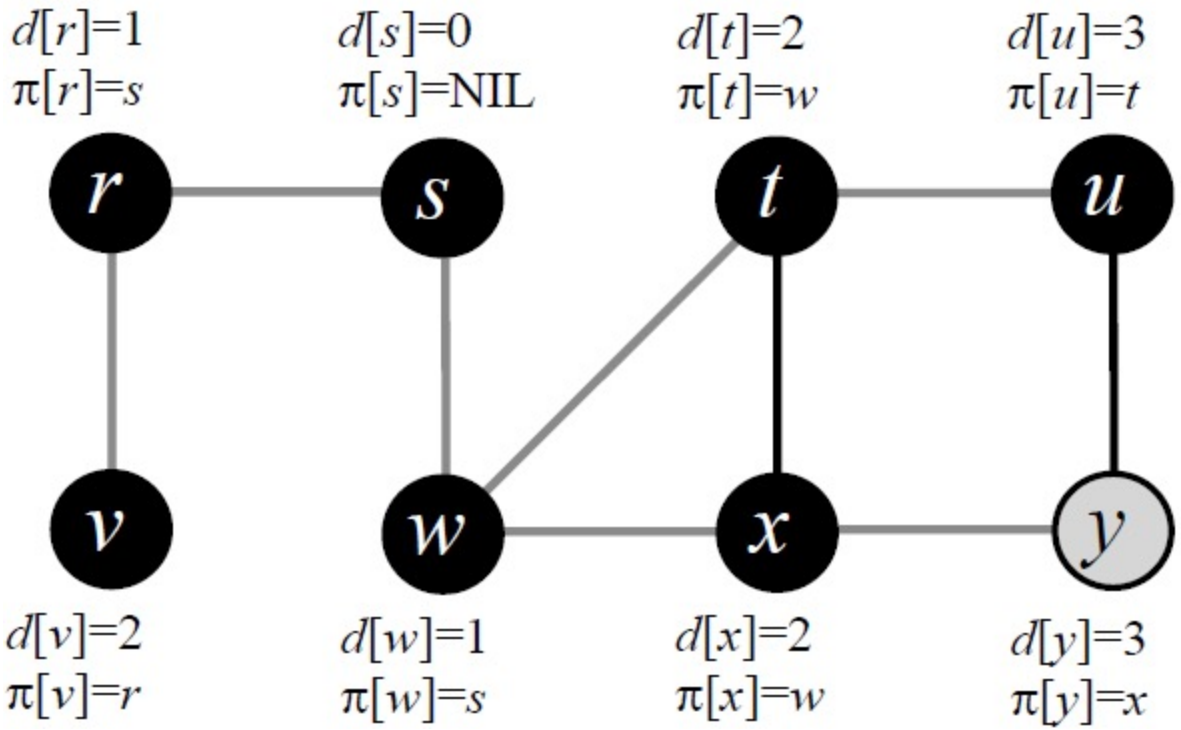

Q
v
u
y

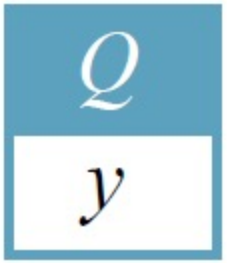
Exemplo BFS



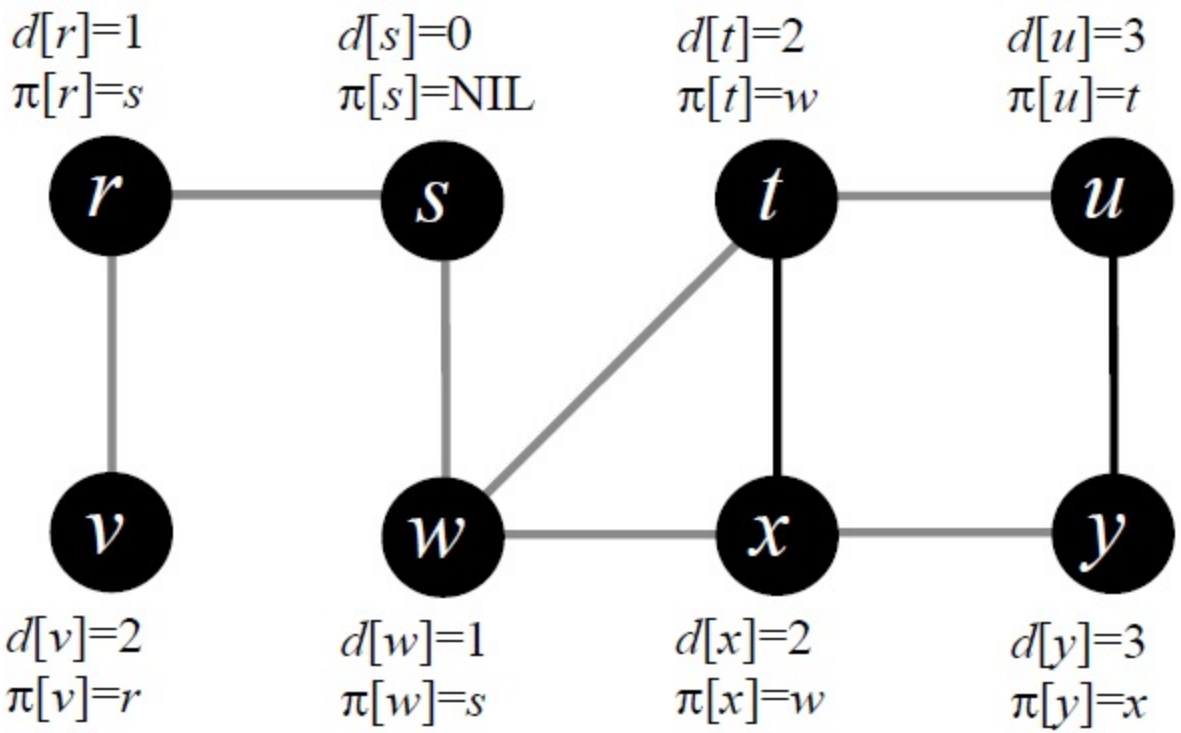
Q
u
y

Exemplo BFS


d[r]=1
π[r]=s
d[s]=0
π[s]=NIL
d[t]=2
π[t]=w
d[u]=3
π[u]=t
r
s
t
u
v
w
x
y
d[v]=2
π[v]=r
d[w]=1
π[w]=s
d[x]=2
π[x]=w
d[y]=3
π[y]=x


Q
y

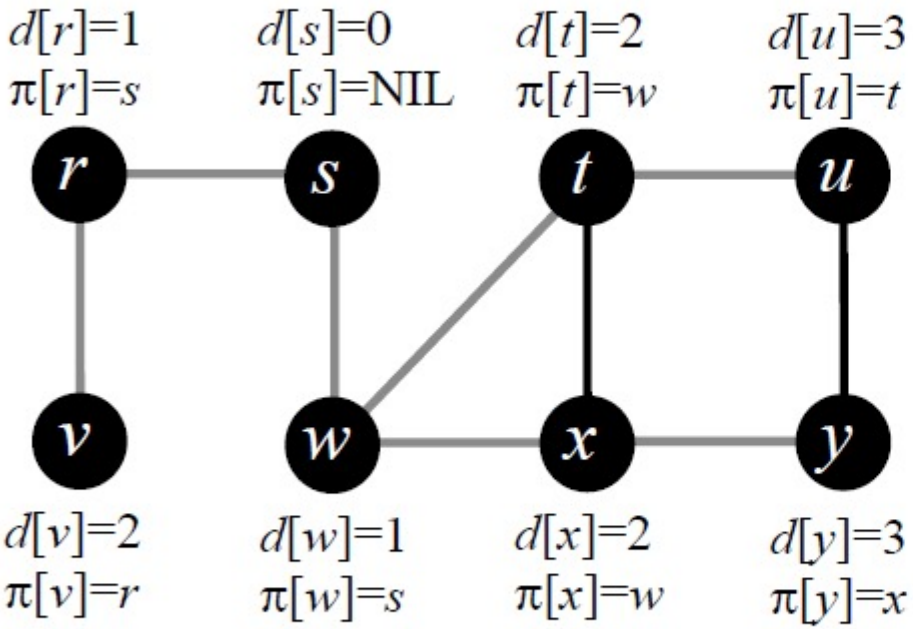
Exemplo BFS

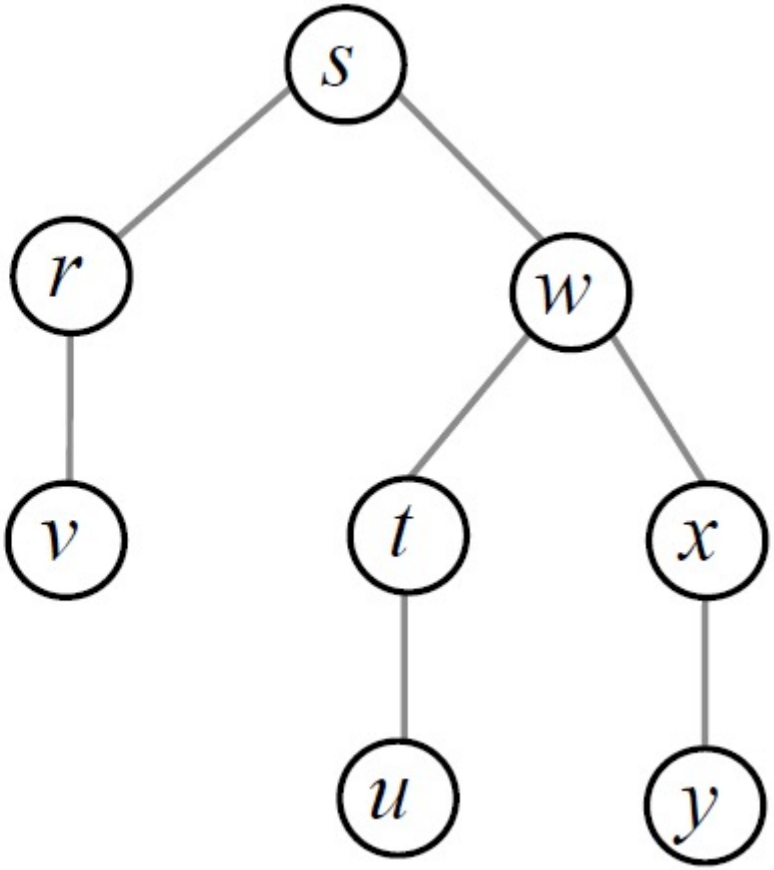

d[r]=1
π[r]=s
d[s]=0
π[s]=NIL
d[t]=2
π[t]=w
d[u]=3
π[u]=t
r
s
t
u
v
w
x
y
d[v]=2
π[v]=r
d[w]=1
π[w]=s
d[x]=2
π[x]=w
d[y]=3
π[y]=x


Q

Exemplo BFS

- Árvore BF: pai na árvore é o predecessor π





Procura em Largura Primeiro

- Tempo de execução
  - Matriz de adjacências: $O(V^2)$
    - Inicialização: $O(V)$
    - Cada vértice colocado na fila apenas 1 vez: $O(V)$
    - Linha da matriz visitada 1 vez por cada vértice: $O(V^2)$
  - Listas de adjacências: $O(V+E)$
    - Inicialização: $O(V)$
    - Cada vértice colocado na fila apenas 1 vez: $O(V)$
    - Lista de adjacências visitada 1 vez por cada vértice: $O(E)$

Procura em Largura Primeiro (BFS)

- Seja δ (s,v):
  - a menor distância de s a v - menor número de arcos em qualquer caminho de s para v
- Resultado
  - Para qualquer arco (*u*,*v*):
    - Se *u* atingível, então *v* atingível
      - Caminho mais curto para *v* não pode ser superior a caminho mais curto para *u* mais o arco (*u*,*v*)
    - Se *u* não atingível, então resultado é válido (independentemente de *v* ser atingível)
- No final da BFS
  - $d[u] = \delta(s,u)$, para todo o vértice *u* de *V*

Procura em Profundidade Primeiro (DFS)

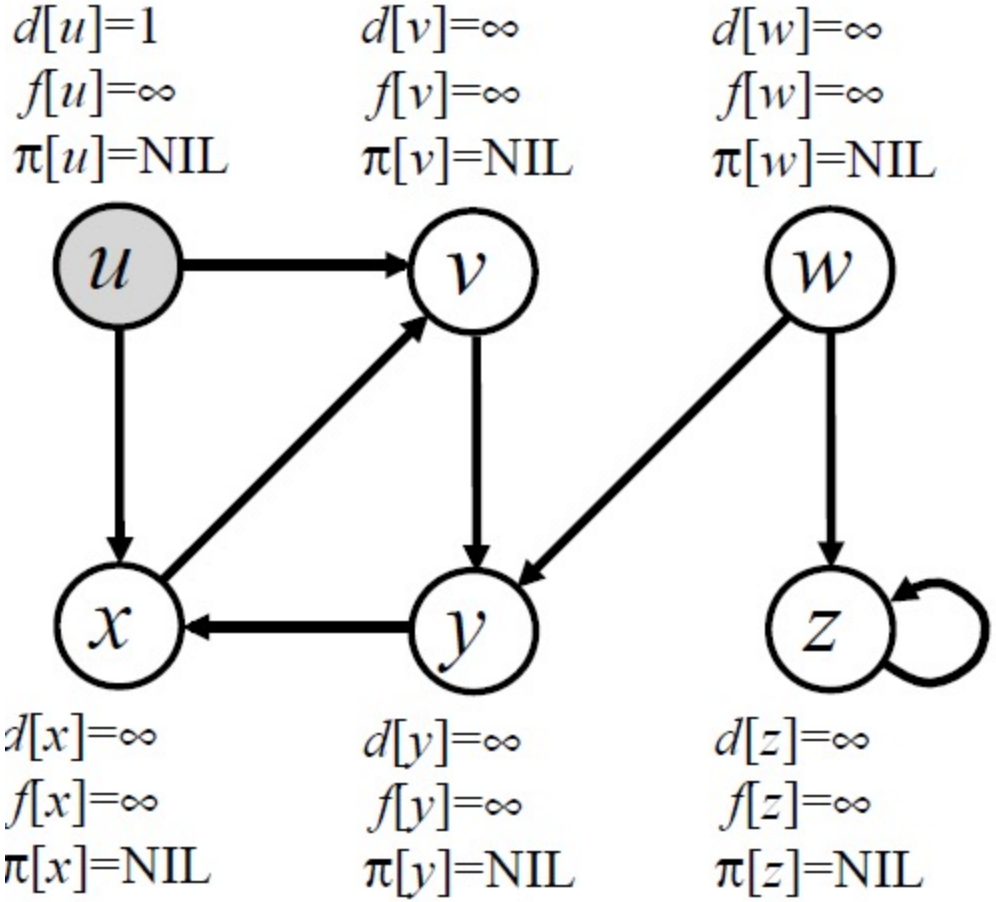
- Grafo pesquisado dando prioridade aos arcos dos vértices mais recentemente visitados
- Resultado da procura:
  - Floresta DF: $G_\pi (V,E_\pi)$ $E_\pi = \{(\pi[v],v) : v \in V \wedge \pi[v] \neq NIL\}$
  - Floresta DF composta por várias árvores DF
- Implementação:
  - *color*[*u*]: cor do vértice (branco, cinzento, preto)
  - *d*[*u*]: tempo de início (de visita do vértice)
  - *f*[*u*]: tempo de fim (de visita do vértice)
  - $\pi[u]$: predecessor

Procura em Profundidade Primeiro (DFS)

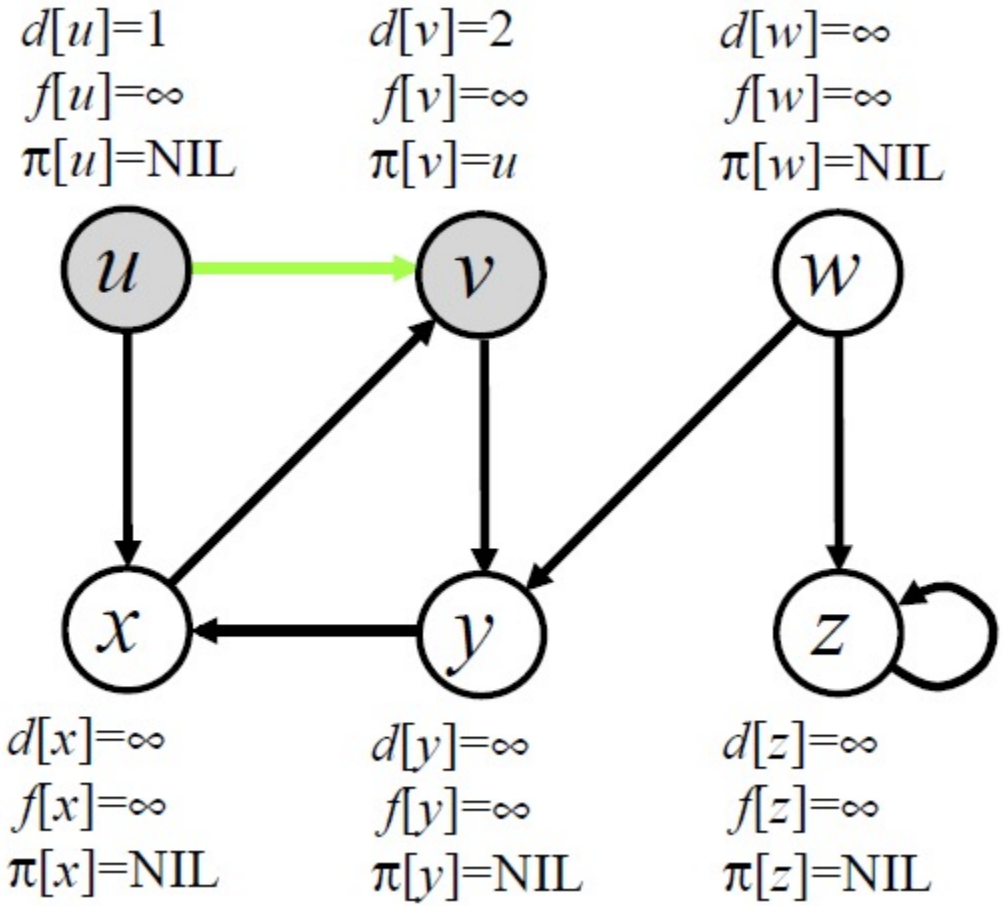
```
DFS(G)
1  for each vertex u ∈ G.V
2      u.color = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6      if u.color == WHITE
7          DFS-VISIT(G, u)

DFS-VISIT(G, u)
 1  time = time + 1                  // white vertex u has just been discovered
 2  u.d = time
 3  u.color = GRAY
 4  for each v ∈ G.Adj[u]            // explore edge (u, v)
 5      if v.color == WHITE
 6          v.π = u
 7          DFS-VISIT(G, v)
 8  u.color = BLACK                  // blacken u; it is finished
 9  time = time + 1
10  u.f = time
```

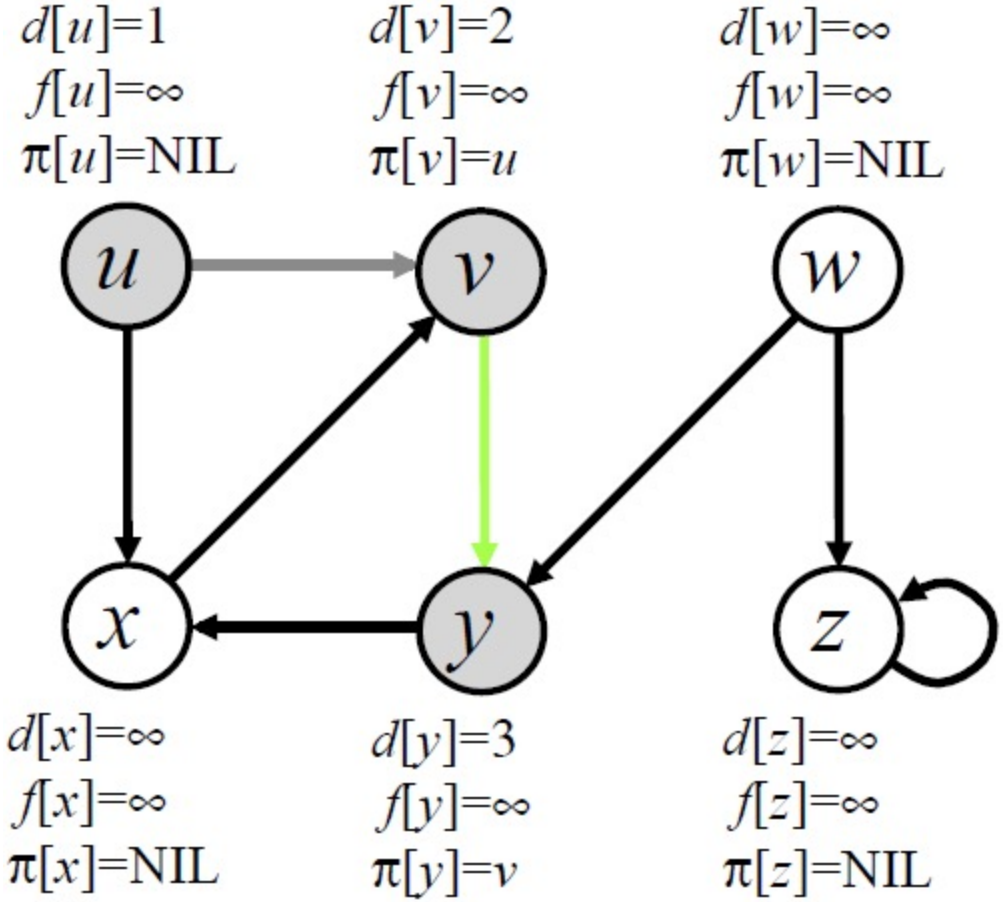
Exemplo DFS


d[u]=1
f[u]=∞
π[u]=NIL
d[v]=∞
f[v]=∞
π[v]=NIL
d[w]=∞
f[w]=∞
π[w]=NIL
u
v
w
x
y
z
d[x]=∞
f[x]=∞
π[x]=NIL
d[y]=∞
f[y]=∞
π[y]=NIL
d[z]=∞
f[z]=∞
π[z]=NIL

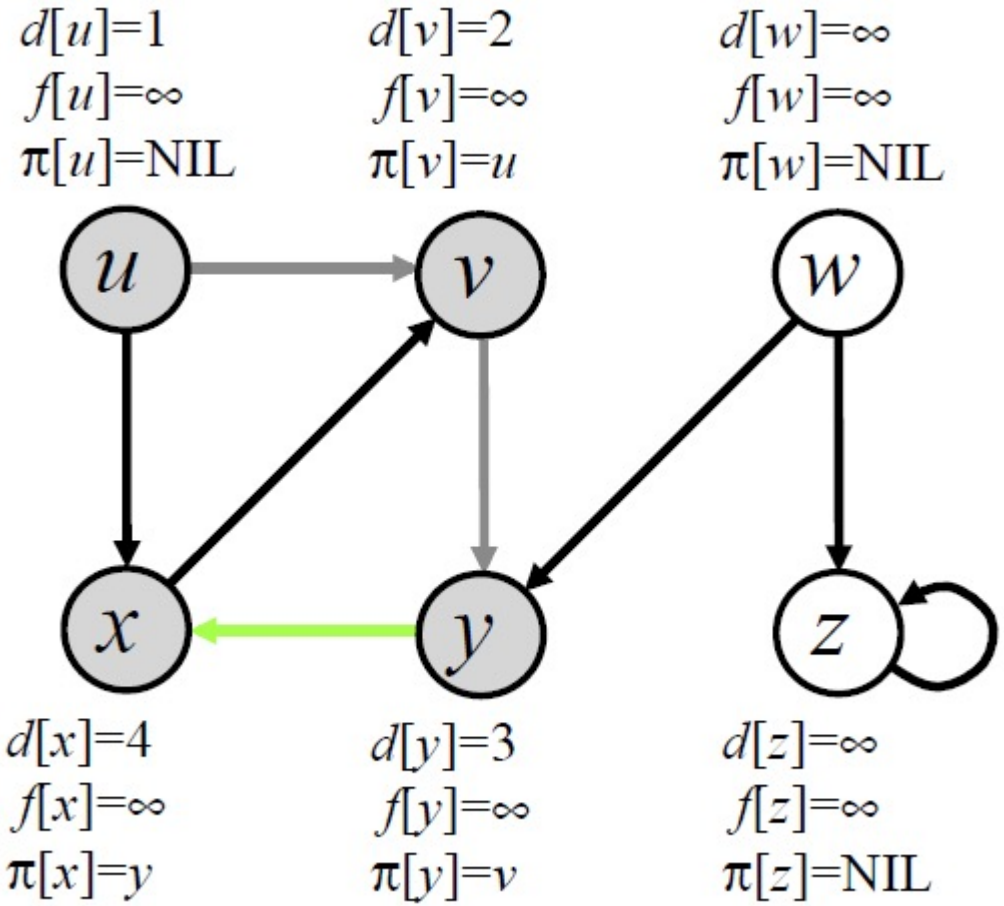
Exemplo DFS


d[u]=1
f[u]=∞
π[u]=NIL
d[v]=2
f[v]=∞
π[v]=u
d[w]=∞
f[w]=∞
π[w]=NIL
u
v
w
x
y
z
d[x]=∞
f[x]=∞
π[x]=NIL
d[y]=∞
f[y]=∞
π[y]=NIL
d[z]=∞
f[z]=∞
π[z]=NIL

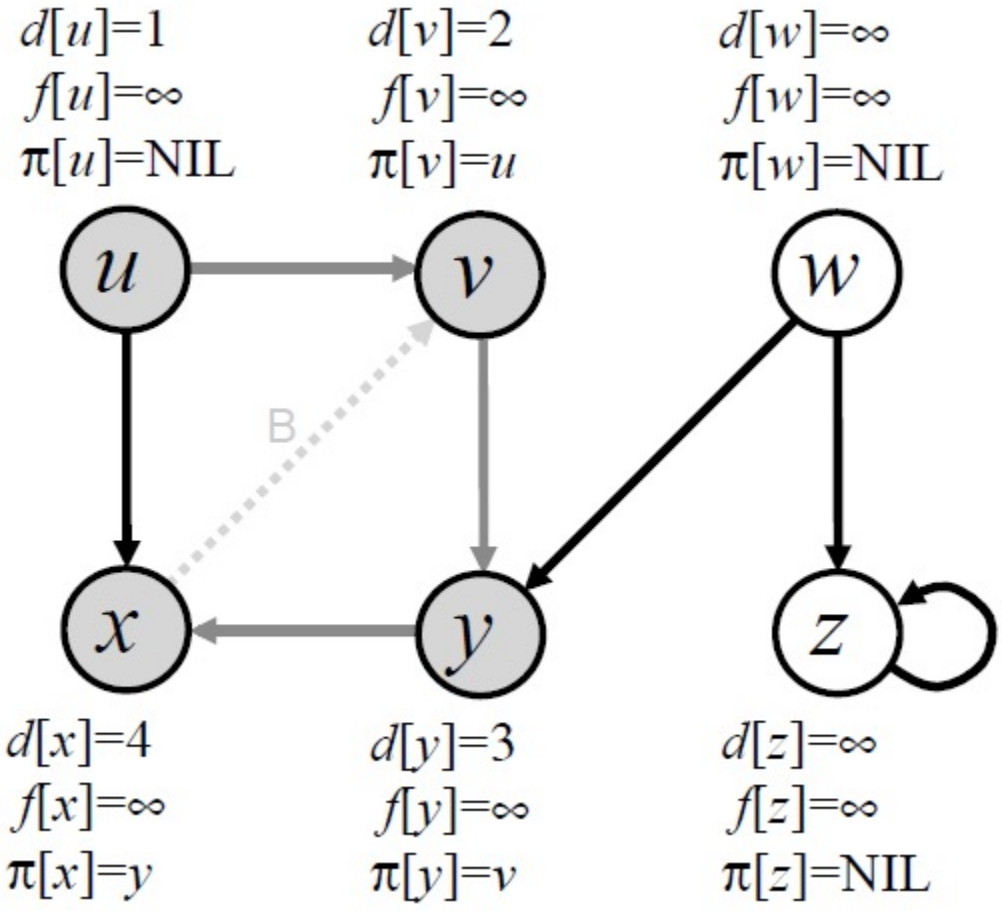
Exemplo DFS


d[u]=1
f[u]=∞
π[u]=NIL
d[v]=2
f[v]=∞
π[v]=u
d[w]=∞
f[w]=∞
π[w]=NIL
u
v
w
x
y
z
d[x]=∞
f[x]=∞
π[x]=NIL
d[y]=3
f[y]=∞
π[y]=v
d[z]=∞
f[z]=∞
π[z]=NIL

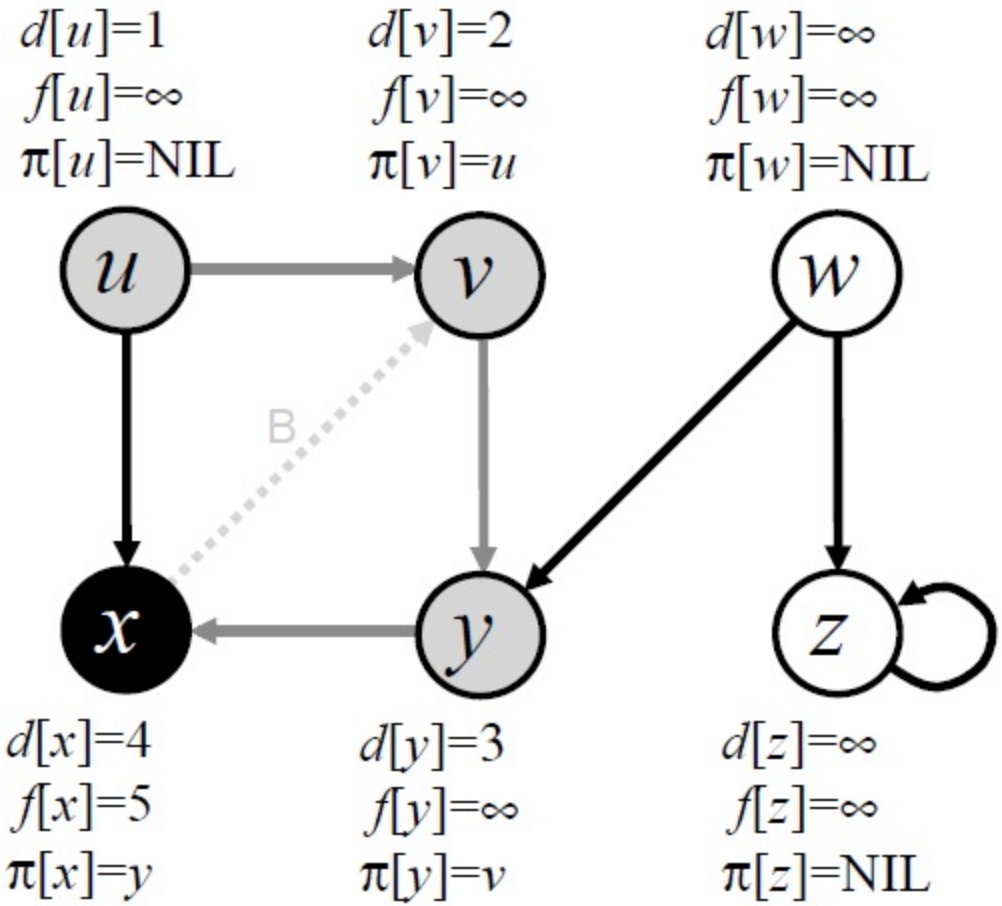
Exemplo DFS


d[u]=1
f[u]=∞
π[u]=NIL
d[v]=2
f[v]=∞
π[v]=u
d[w]=∞
f[w]=∞
π[w]=NIL
u
v
w
x
y
z
d[x]=4
f[x]=∞
π[x]=y
d[y]=3
f[y]=∞
π[y]=v
d[z]=∞
f[z]=∞
π[z]=NIL

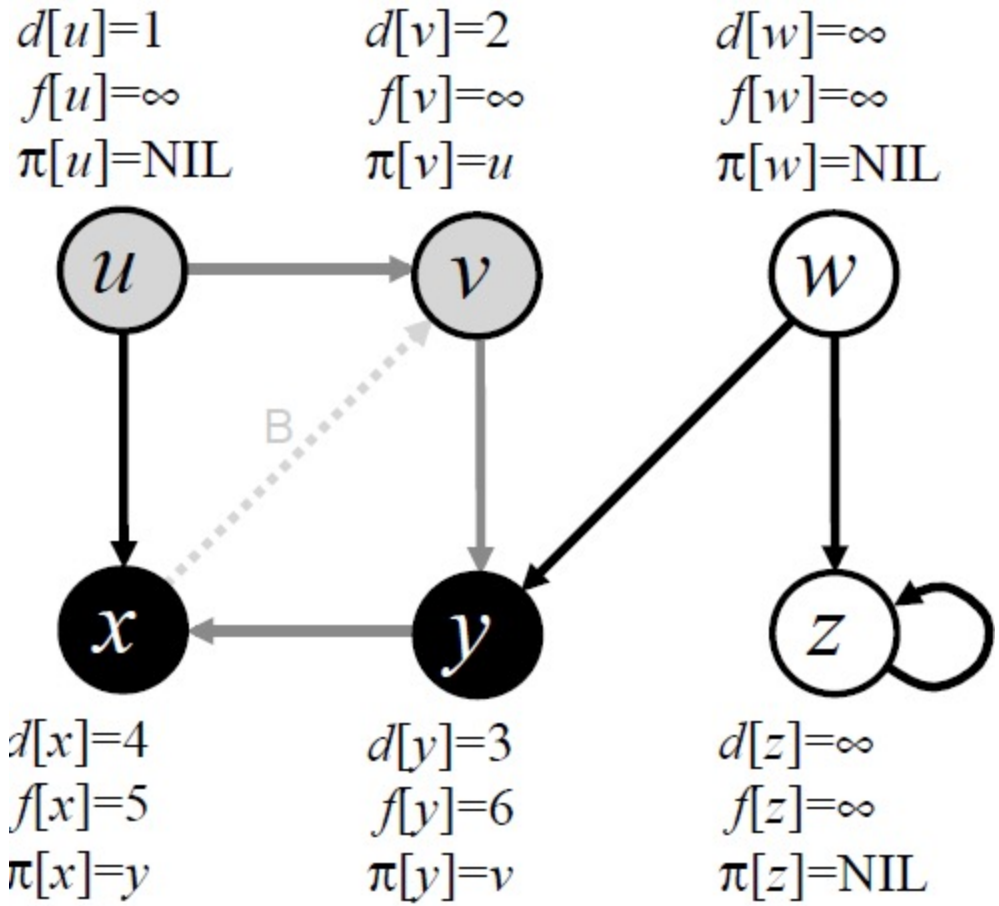
Exemplo DFS


d[u]=1
f[u]=∞
π[u]=NIL
d[v]=2
f[v]=∞
π[v]=u
d[w]=∞
f[w]=∞
π[w]=NIL
u
v
w
B
x
y
z
d[x]=4
f[x]=∞
π[x]=y
d[y]=3
f[y]=∞
π[y]=v
d[z]=∞
f[z]=∞
π[z]=NIL

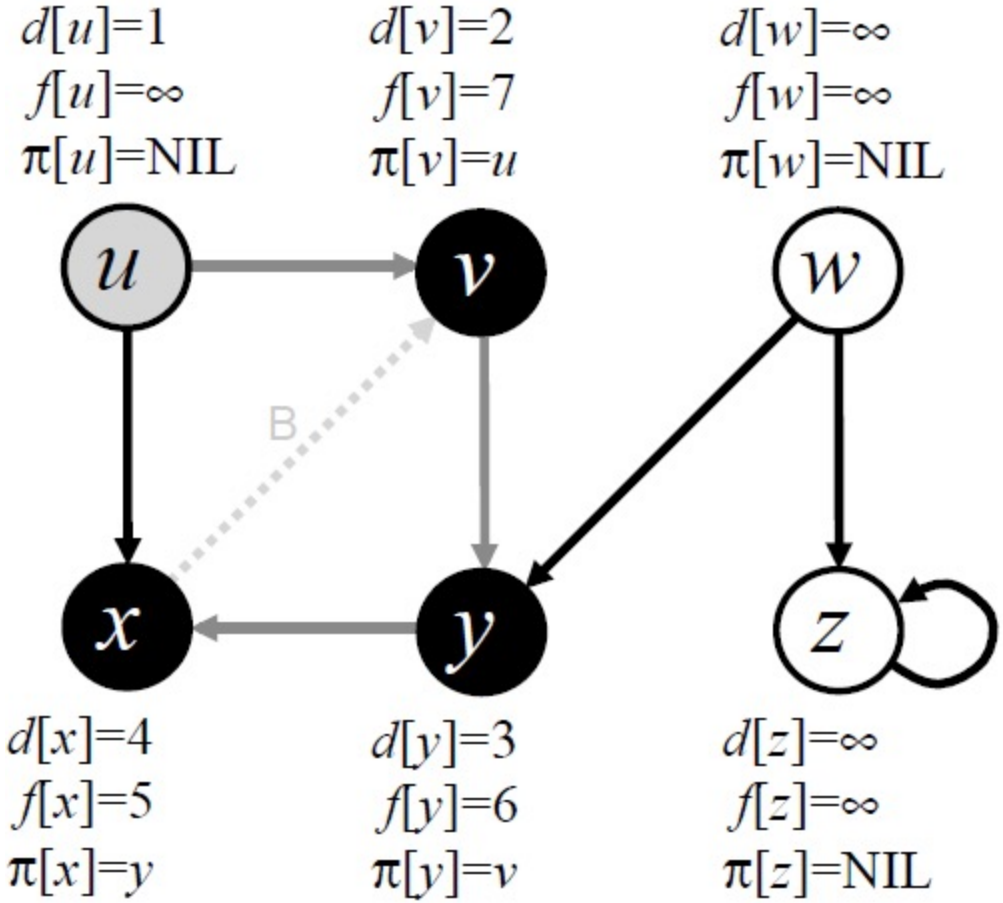
Exemplo DFS


d[u]=1
f[u]=∞
π[u]=NIL
d[v]=2
f[v]=∞
π[v]=u
d[w]=∞
f[w]=∞
π[w]=NIL
u
v
w
B
x
y
z
d[x]=4
f[x]=5
π[x]=y
d[y]=3
f[y]=∞
π[y]=v
d[z]=∞
f[z]=∞
π[z]=NIL

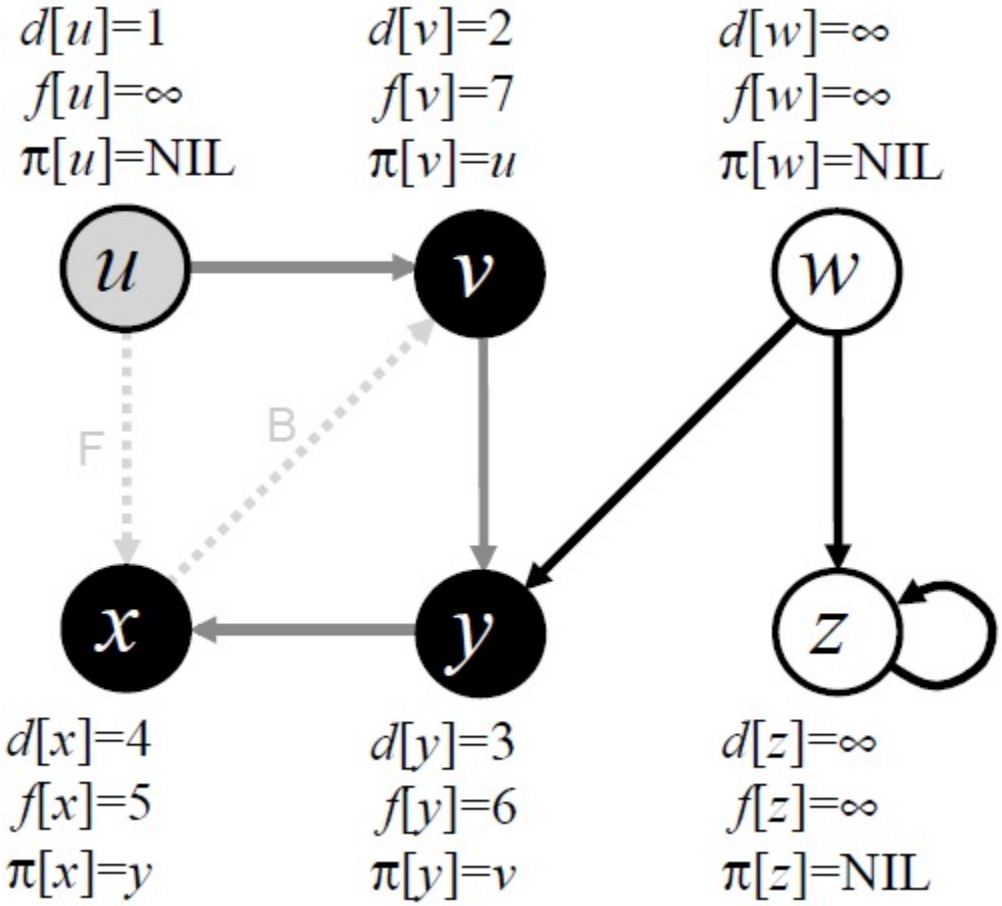
Exemplo DFS


d[u]=1
f[u]=∞
π[u]=NIL
d[v]=2
f[v]=∞
π[v]=u
d[w]=∞
f[w]=∞
π[w]=NIL
u
v
w
B
x
y
z
d[x]=4
f[x]=5
π[x]=y
d[y]=3
f[y]=6
π[y]=v
d[z]=∞
f[z]=∞
π[z]=NIL

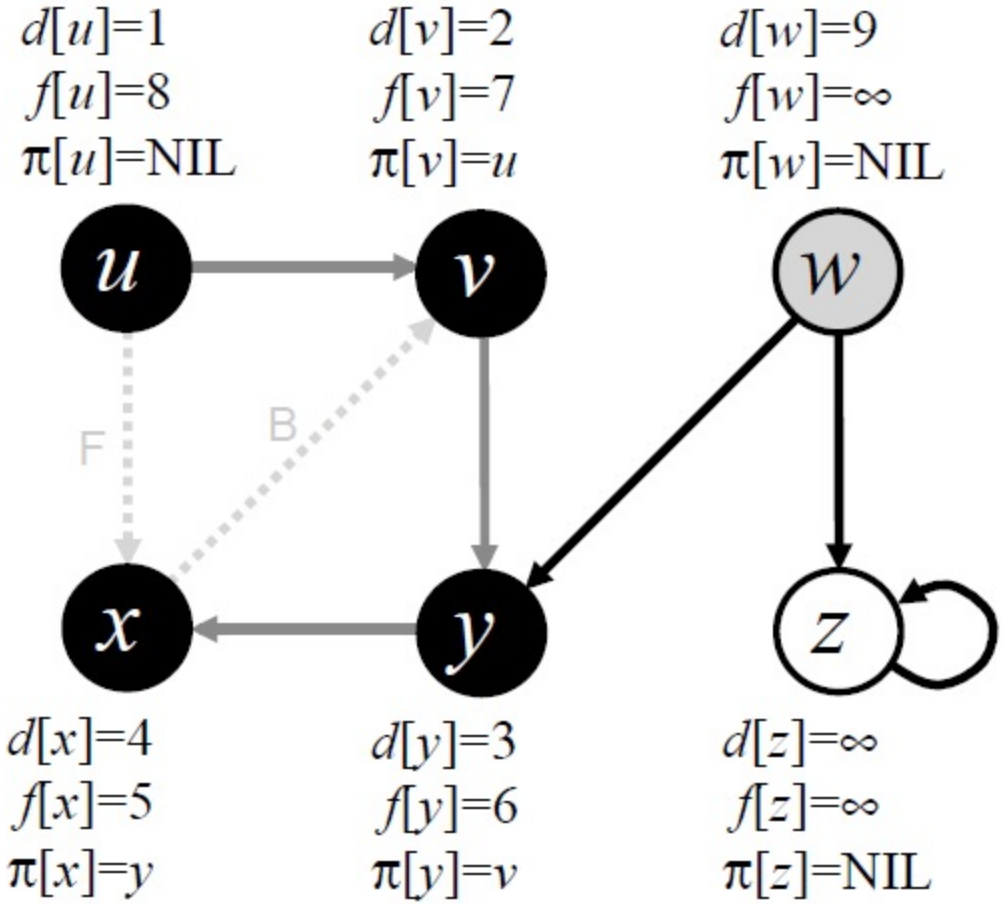
Exemplo DFS


d[u]=1
f[u]=∞
π[u]=NIL
d[v]=2
f[v]=7
π[v]=u
d[w]=∞
f[w]=∞
π[w]=NIL
u
v
w
B
x
y
z
d[x]=4
f[x]=5
π[x]=y
d[y]=3
f[y]=6
π[y]=v
d[z]=∞
f[z]=∞
π[z]=NIL

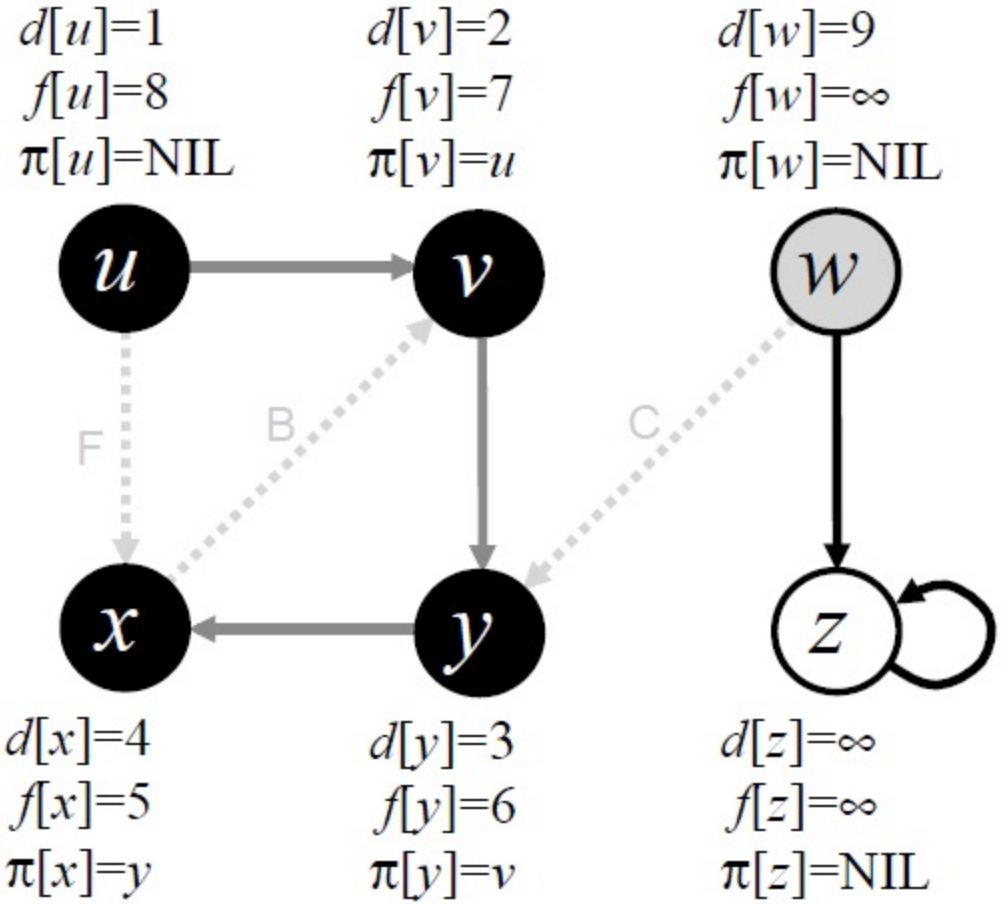
Exemplo DFS


d[u]=1
f[u]=∞
π[u]=NIL
d[v]=2
f[v]=7
π[v]=u
d[w]=∞
f[w]=∞
π[w]=NIL
u
v
w
F
B
x
y
z
d[x]=4
f[x]=5
π[x]=y
d[y]=3
f[y]=6
π[y]=v
d[z]=∞
f[z]=∞
π[z]=NIL

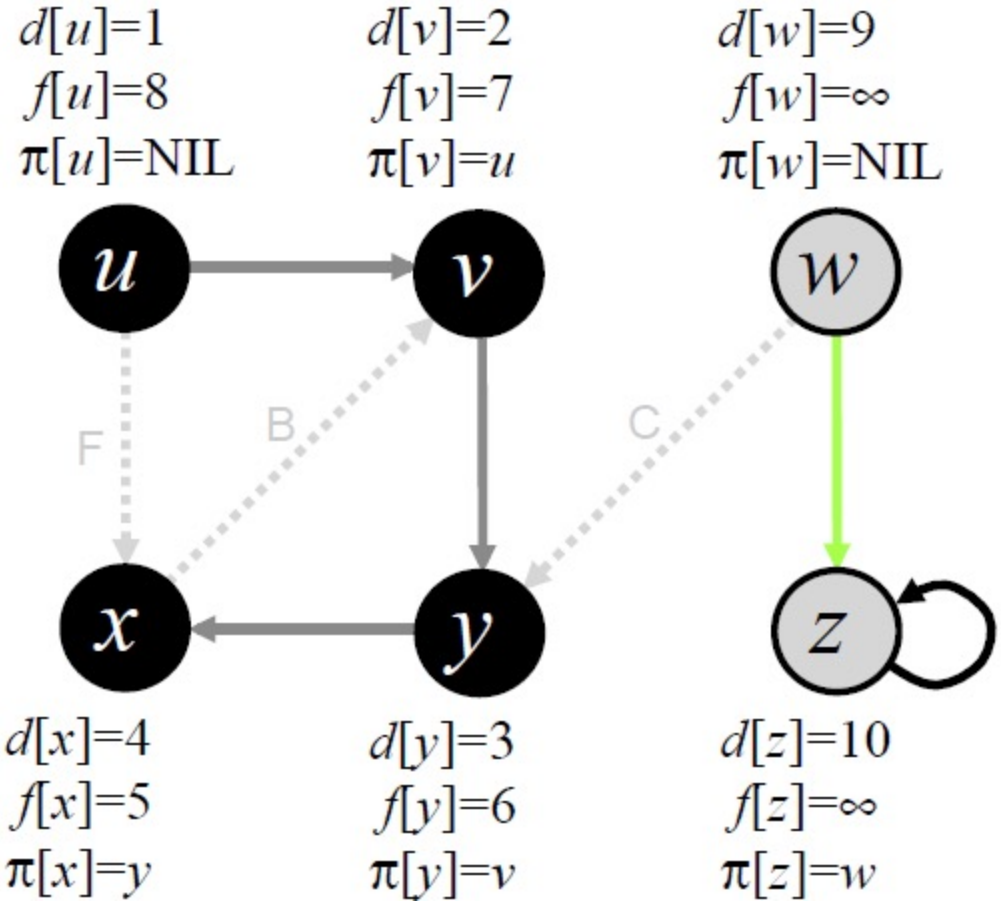
Exemplo DFS


d[u]=1
f[u]=8
π[u]=NIL
d[v]=2
f[v]=7
π[v]=u
d[w]=9
f[w]=∞
π[w]=NIL
u
v
w
F
B
x
y
z
d[x]=4
f[x]=5
π[x]=y
d[y]=3
f[y]=6
π[y]=v
d[z]=∞
f[z]=∞
π[z]=NIL

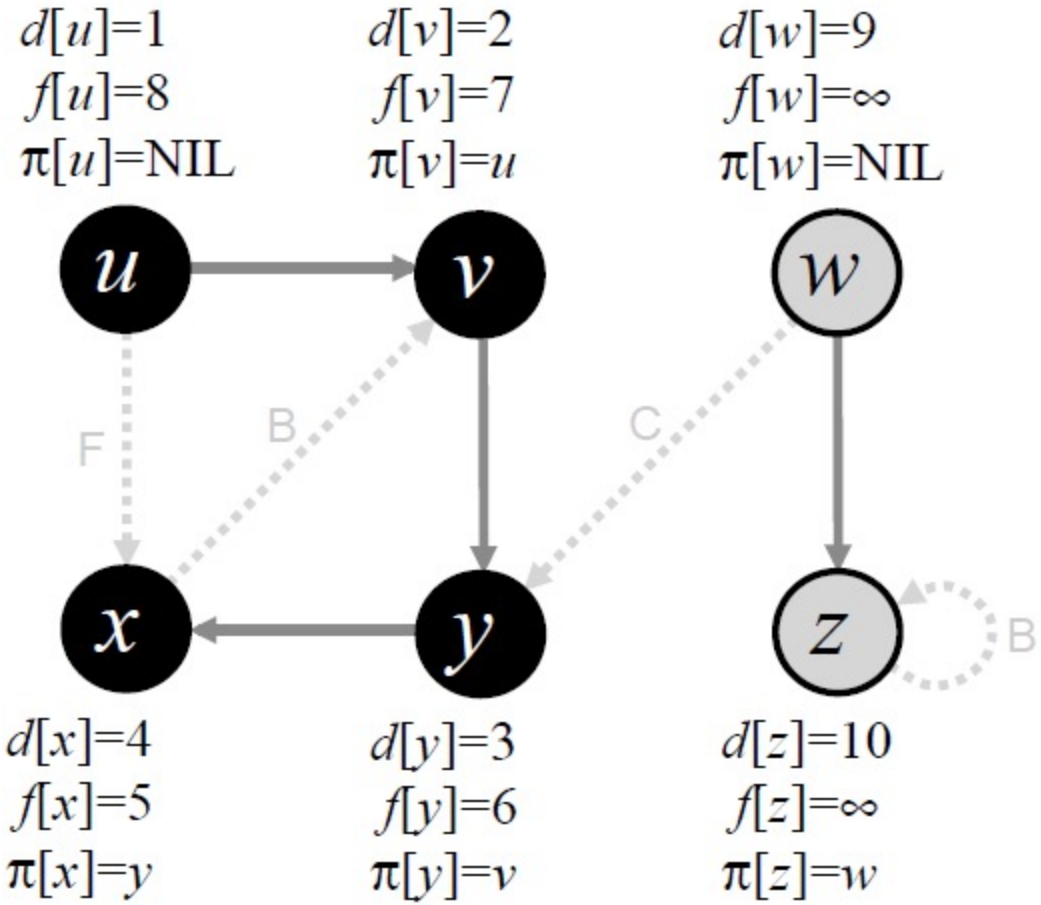
Exemplo DFS


d[u]=1
f[u]=8
π[u]=NIL
d[v]=2
f[v]=7
π[v]=u
d[w]=9
f[w]=∞
π[w]=NIL
u
v
w
F
B
C
x
y
z
d[x]=4
f[x]=5
π[x]=y
d[y]=3
f[y]=6
π[y]=v
d[z]=∞
f[z]=∞
π[z]=NIL

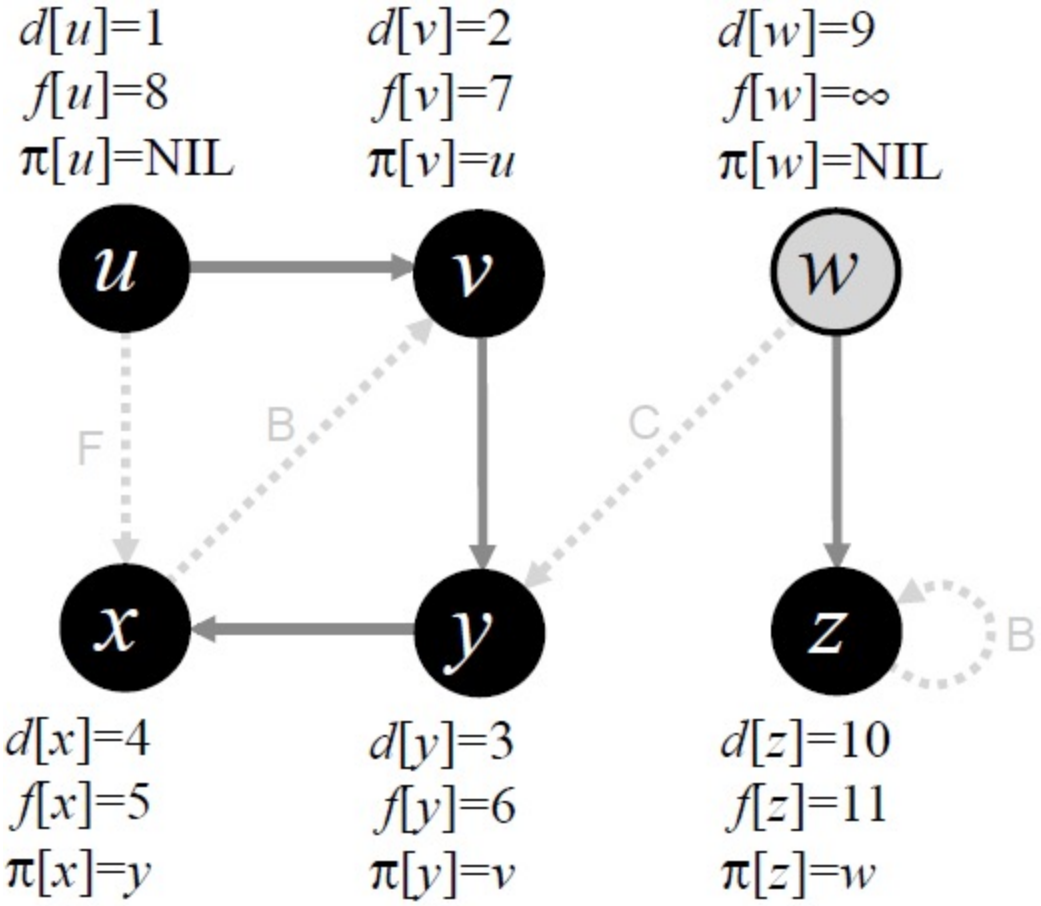
Exemplo DFS


d[u]=1
f[u]=8
π[u]=NIL
d[v]=2
f[v]=7
π[v]=u
d[w]=9
f[w]=∞
π[w]=NIL
u
v
w
F
B
C
x
y
z
d[x]=4
f[x]=5
π[x]=y
d[y]=3
f[y]=6
π[y]=v
d[z]=10
f[z]=∞
π[z]=w

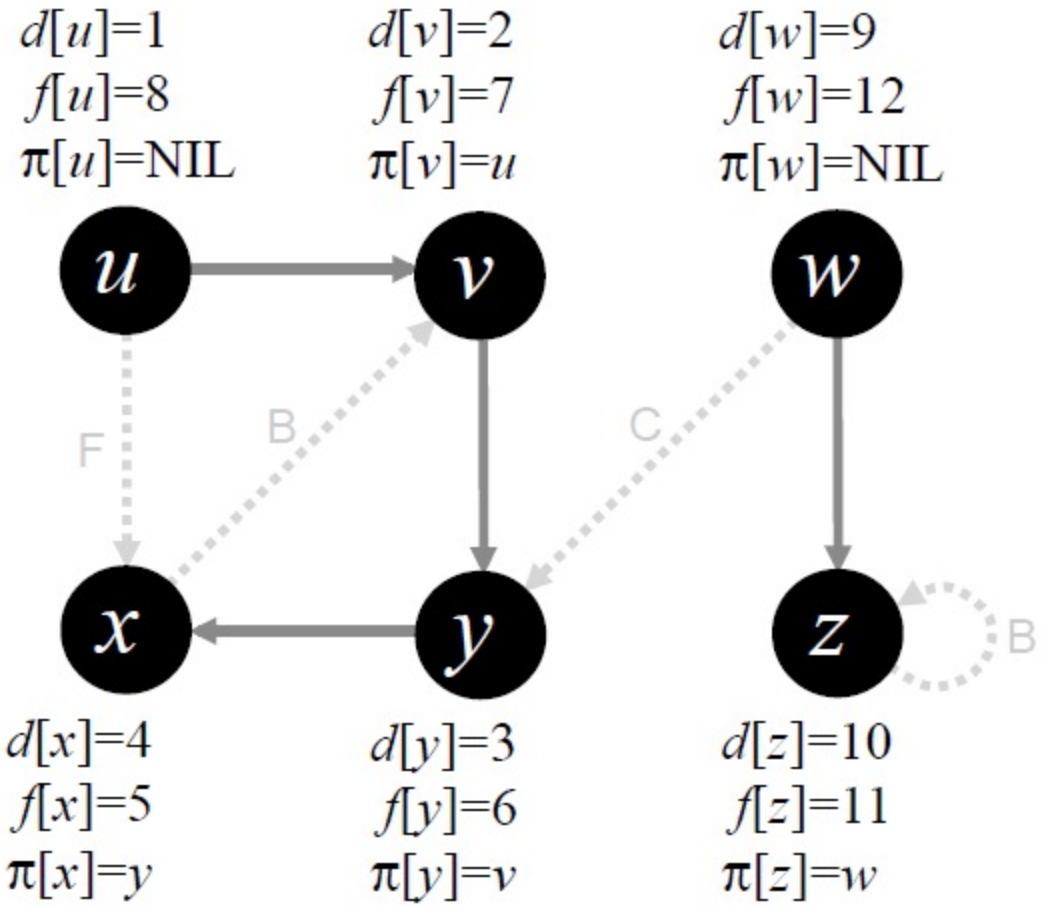
Exemplo DFS


d[u]=1
f[u]=8
π[u]=NIL
d[v]=2
f[v]=7
π[v]=u
d[w]=9
f[w]=∞
π[w]=NIL
u
v
w
F
B
C
x
y
z
B
d[x]=4
f[x]=5
π[x]=y
d[y]=3
f[y]=6
π[y]=v
d[z]=10
f[z]=∞
π[z]=w

Exemplo DFS

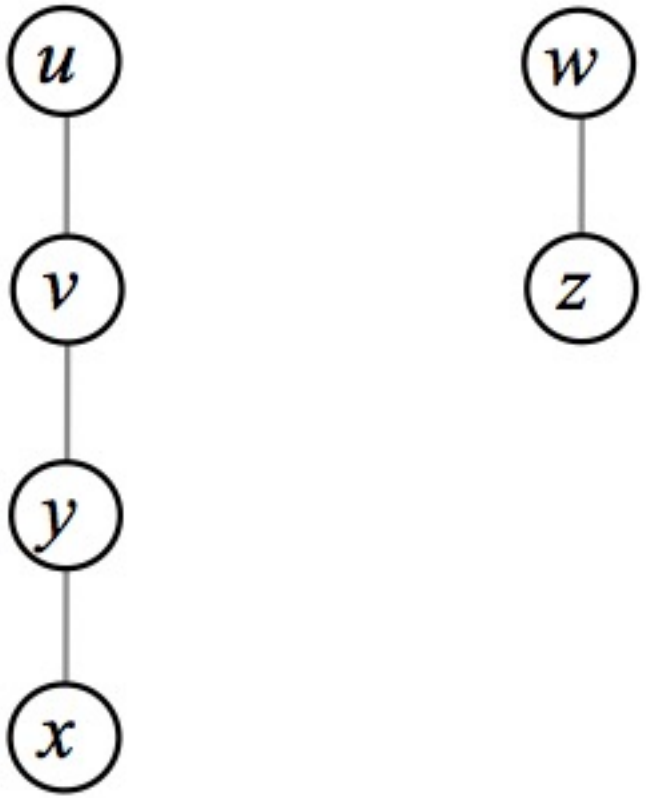

d[u]=1
f[u]=8
π[u]=NIL
d[v]=2
f[v]=7
π[v]=u
d[w]=9
f[w]=∞
π[w]=NIL
u
v
w
F
B
C
x
y
z
B
d[x]=4
f[x]=5
π[x]=y
d[y]=3
f[y]=6
π[y]=v
d[z]=10
f[z]=11
π[z]=w

Exemplo DFS


d[u]=1
f[u]=8
π[u]=NIL
d[v]=2
f[v]=7
π[v]=u
d[w]=9
f[w]=12
π[w]=NIL
u
v
w
F
B
C
x
y
z
B
d[x]=4
f[x]=5
π[x]=y
d[y]=3
f[y]=6
π[y]=v
d[z]=10
f[z]=11
π[z]=w

Exemplo

- FlorestaDF: pai numa árvore é o predecessor π
  - Floresta porque pode conter várias árvores



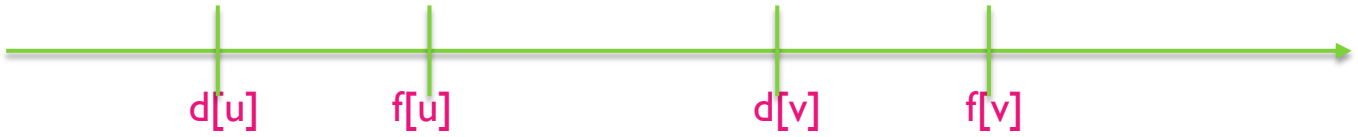
Procura em Profundidade Primeiro (DFS)

- Tempo de execução: O(V+E)
  - Inicialização: *O(V)*
  - Chamadas a DFS-Visit dentro de DFS: *O(V)*
  - Arcos analisados em DFS-Visit: Θ(E)
    - Chamadas a DFS-Visit dentro de DFS-Visit: *O(V)*
    - Mas

$$\sum_{v \in V} |\, Adj[v] \,| = \Theta(E)$$

Procura em Profundidade Primeiro (DFS)

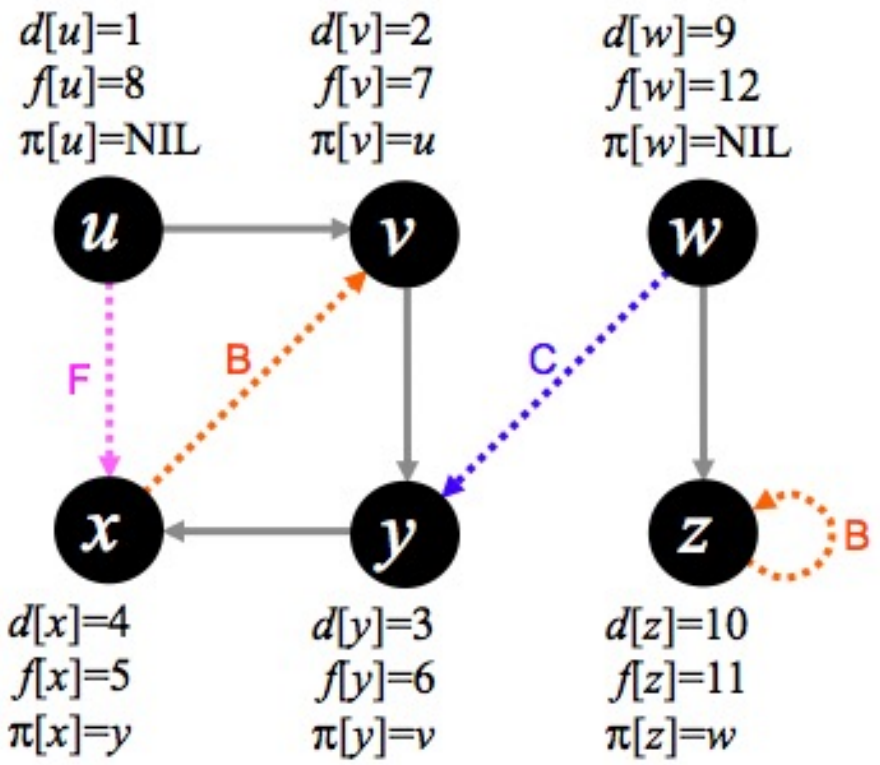
- Numa DFS de $G = (V, E)$, para cada par de vértices u e v apenas um dos 3 casos seguintes é verdade:
  - Os intervalos [d[u], f[u]] e [d[v], f[v]] são disjuntos



  - [d[u], f[u]] está contido em [d[v], f[v]] e u é descendente de v numa árvore DF

  - [d[v], f[v]] está contido em [d[u], f[u]] e v é descendente de u numa árvore DF

Exemplo DFS



Arcos de árvore

Arcos para trás (back edges)

Arcos para a frente (forward Edges)

Arcos de cruzamento (cross edges)

Procura em Profundidade Primeiro (DFS)

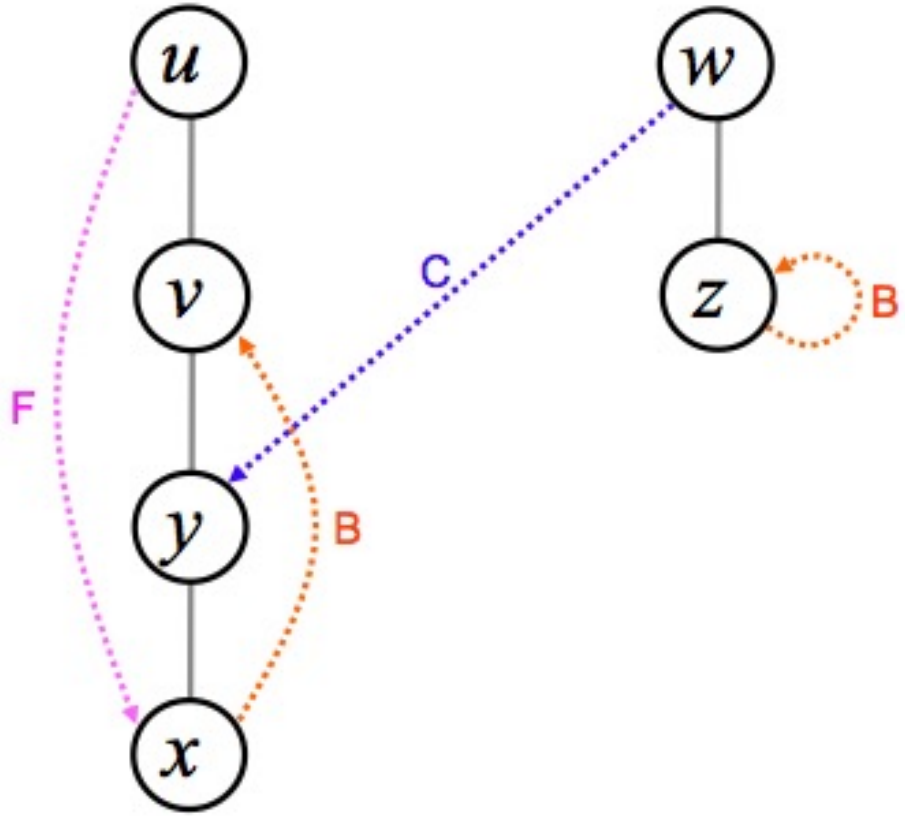
- Classificação de cada arco (u,v)
  - Arcos de árvore: (tree edges)
    - Arcos na floresta DF
    - (u,v) é arco de árvore se *v* foi visitado devido ao arco (u,v) ser visitado
  - Arcos para trás: (back edges)
    - Ligam vértice *u* a vértice *v* antecessor na mesma árvore DF
  - Arcos para a frente: (forward edges)
    - Ligam vértice *v* a vértice descendente na mesma árvore DF
  - Arcos de cruzamento: (cross edges)
    - Na mesma árvore DF, se *u* (ou *v*) não antecessor de *v* (ou *u*)
    - Entre árvores DF diferentes

Procura em Profundidade Primeiro

- Classificação de cada arco (u,v)
- Arco de árvore:
  - d[u] < d[v] < f[v] < f[u] ; color[v] = white;
  - visita *v* a partir de *u*
- Arco para trás:
  - *d[v] < d[u] < f[u] < f[v] ; color[v] = gray*
- Arco para a frente:
  - d[u] < d[v] < f[v] < f[u] ; color[v] = black
- Arco de cruzamento:
  - d[v] < f[v] < d[u] < f[u] ; color[v] = black

Exemplo DFS

Mais fácil de compreender a classificação dos arcos representando-os na floresta DF

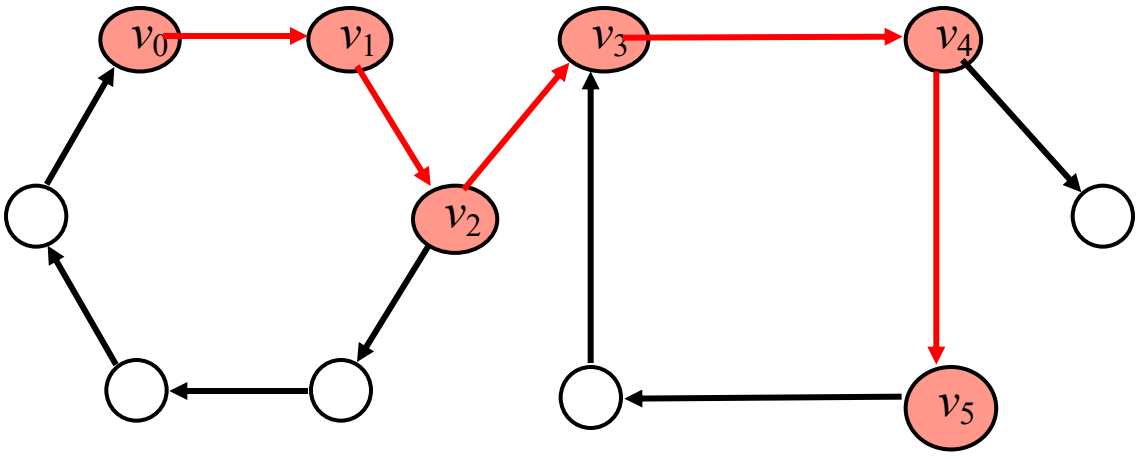


Procura em Profundidade Primeiro

- Dado $G = (V, E)$, não orientado, cada arco é arco de árvore ou arco para trás
  - i.e., não existem arcos para a frente e de cruzamento
- Numa floresta DFS, v é descendente de u se e só se quando *u* é descoberto existe um caminho de vértices brancos de *u* para *v*
  - *v* descendente de *u* ⇒ existe caminho de vértices brancos de *u* para *v*
- Qualquer vértice *w* descendente de *u* verifica [d[w], f[w]] ⊂ [d[u], f [u]], pelo que *w* é branco quando *u* é descoberto

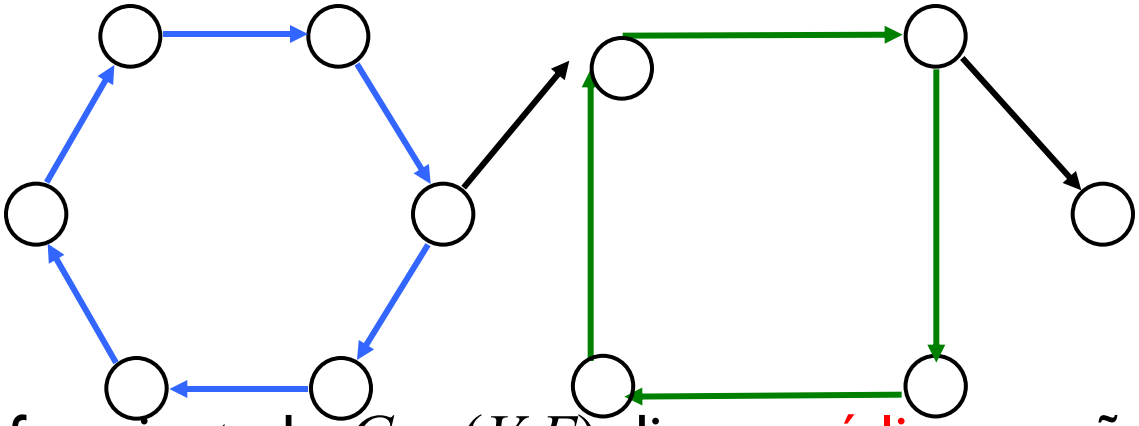
Caminho

- Dado grafo $G = (V, E)$, um caminho p é uma sequência $<v_0, v_1, \ldots, v_k>$ tal que para todo o i, $0 \leq i \leq k-1$, $(v_i, v_{i+1}) \in E$
- Se existe um caminho p de u para v, então v diz-se atingível a partir de u via p



DAG

- Um ciclo num grafo $G = (V,E)$ é um caminho $<v_0, v_1, \ldots, v_k>$, tal que $v_0 = v_k$
  - No grafo existem 2 ciclos (azul e verde)



- Um grafo orientado $G = (V,E)$ diz-se acíclico se não tem ciclos
  - Directed acyclic graph (DAG)

Ordenação Topologica

- Uma ordenação topológica de um DAG G=(V,E) é uma ordenação de todos os vértices tal que se (u,v) ∈E então *u* aparece antes de *v* na ordenação
  - **Utilizando informação de DFS**

TOPOLOGICAL-SORT(g)

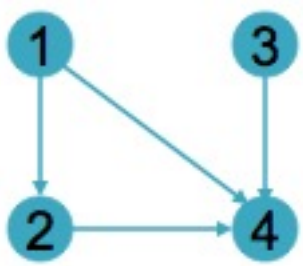
- invocar DFS(G) para computar os tempos de finalização para cada vértice

- sempre que cada vértice está terminado, inserir no início de uma lista ligada

- **retornar** a lista ligada de vértices

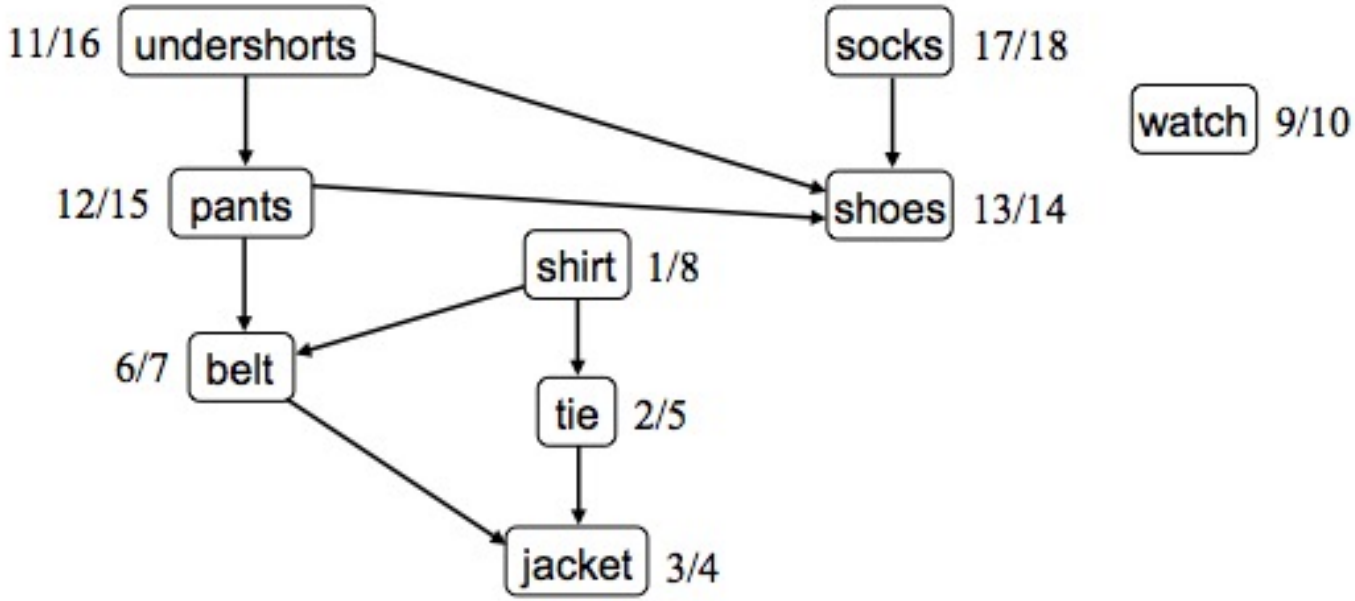
Ordenação Topologica

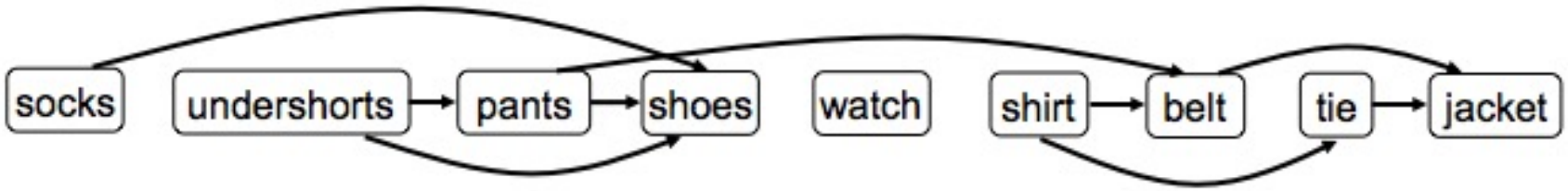
- Intuição:



- Na DFS:
  - Tempo de fim de 3 é sempre > Tempo de fim de 4
  - Tempo de fim de 2 é sempre > Tempo de fim de 4
  - Tempo de fim de 1 é sempre > Tempo de fim de 2,4
- Sem ciclos, se existe caminho de *u* para *v*, verifica-se sempre *f[u] > f[v]* **!**

Ordenação Topologica





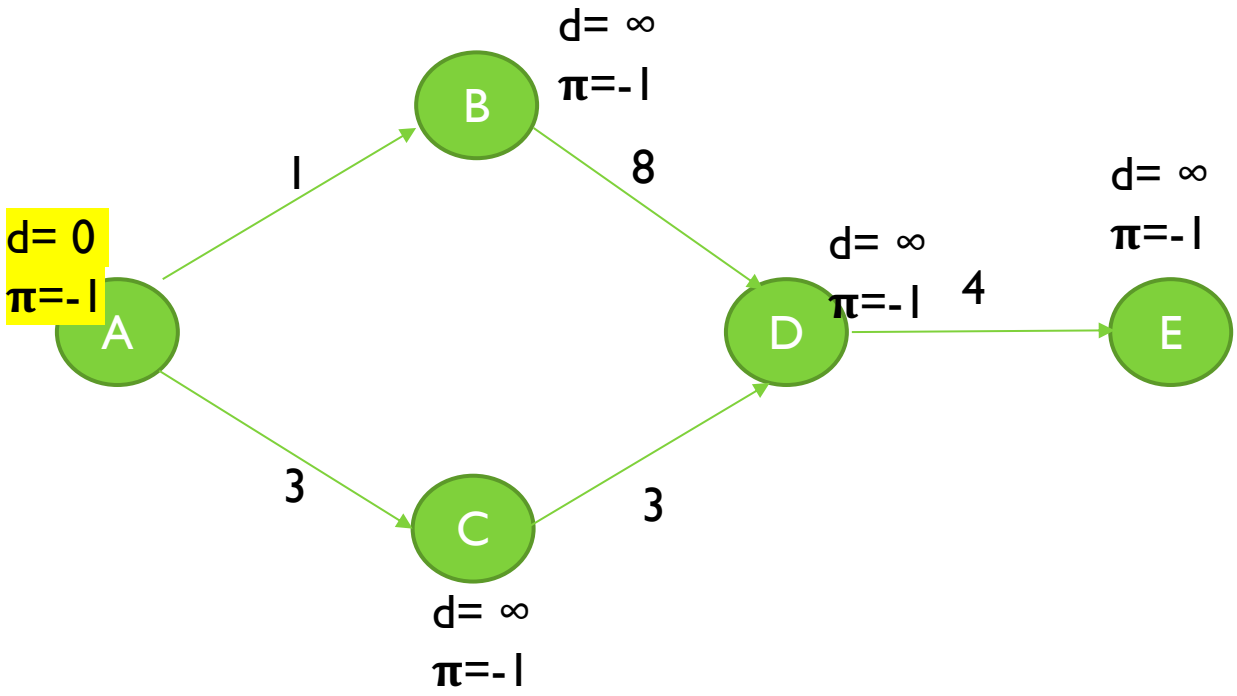
Dijkstra

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S = ∅
3  Q = G.V
4  while Q ≠ ∅
5      u = EXTRACT-MIN(Q)
6      S = S ∪ {u}
7      for each vertex v ∈ G.Adj[u]
8          RELAX(u, v, w)
```

```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for each vertex v ∈ G.V
2      v.d = ∞
3      v.π = NIL
4  s.d = 0
```

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```

Dijsktra



PQ

A – (0)

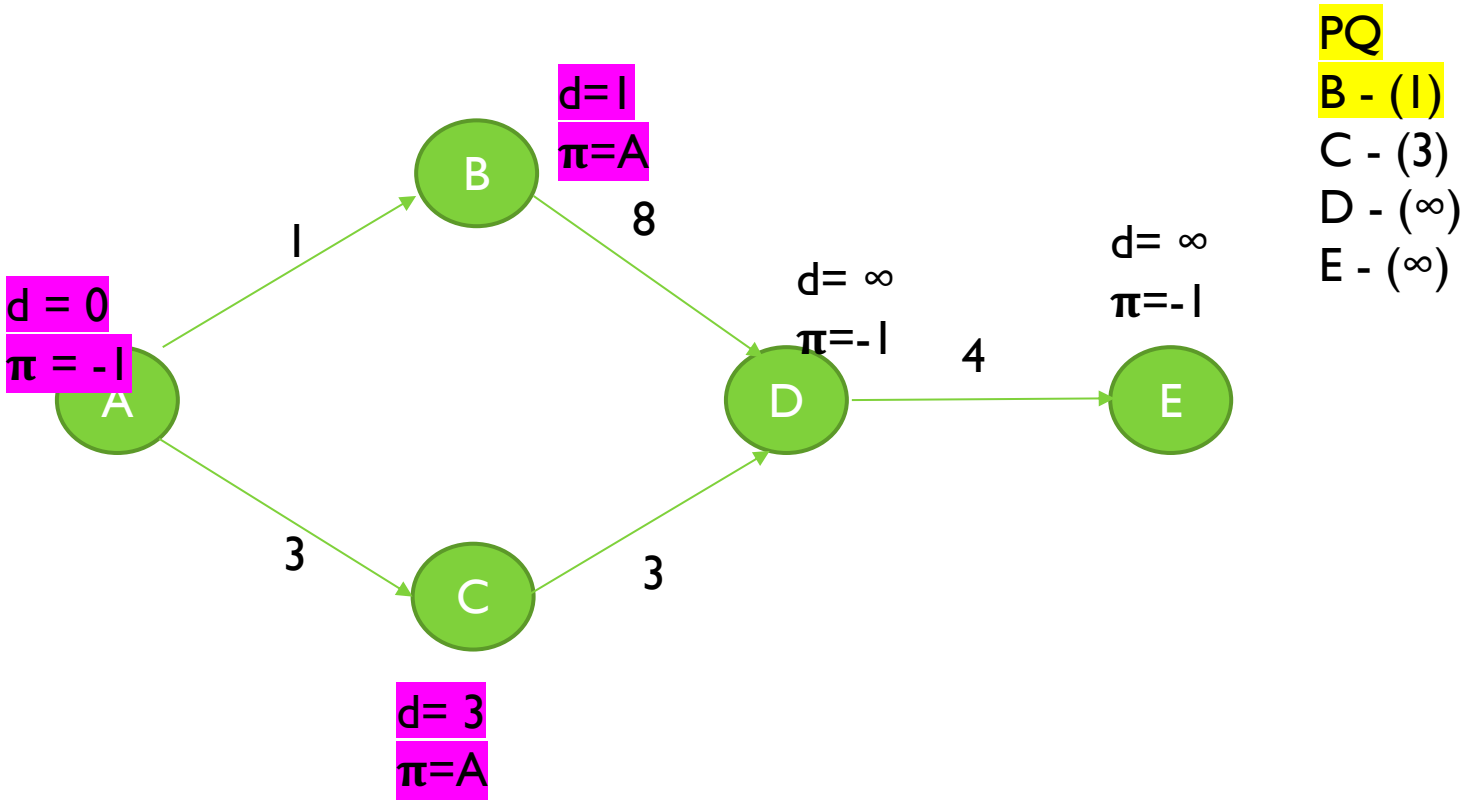
B – (∞)

D- (∞)

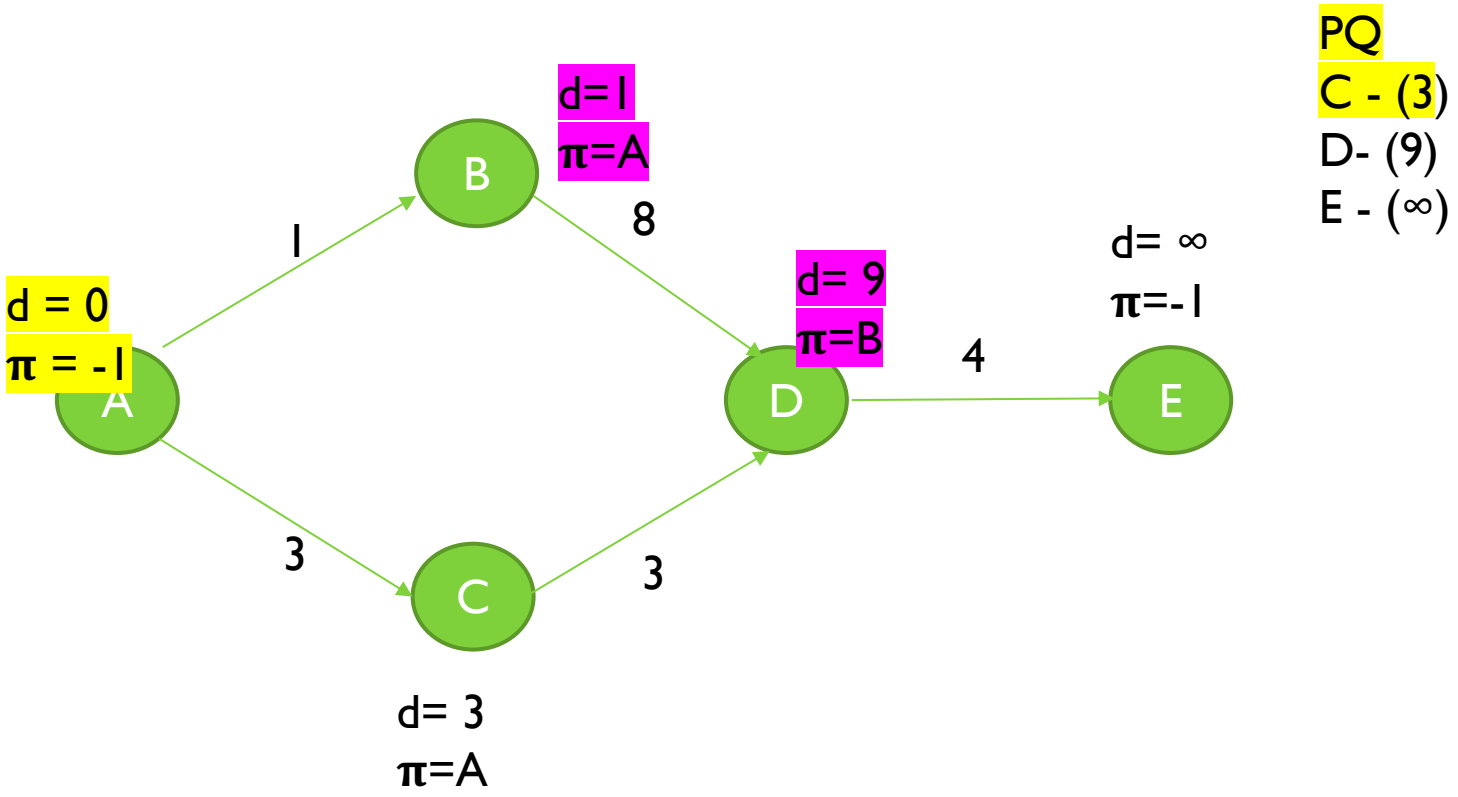
C - (∞)

E - (∞)

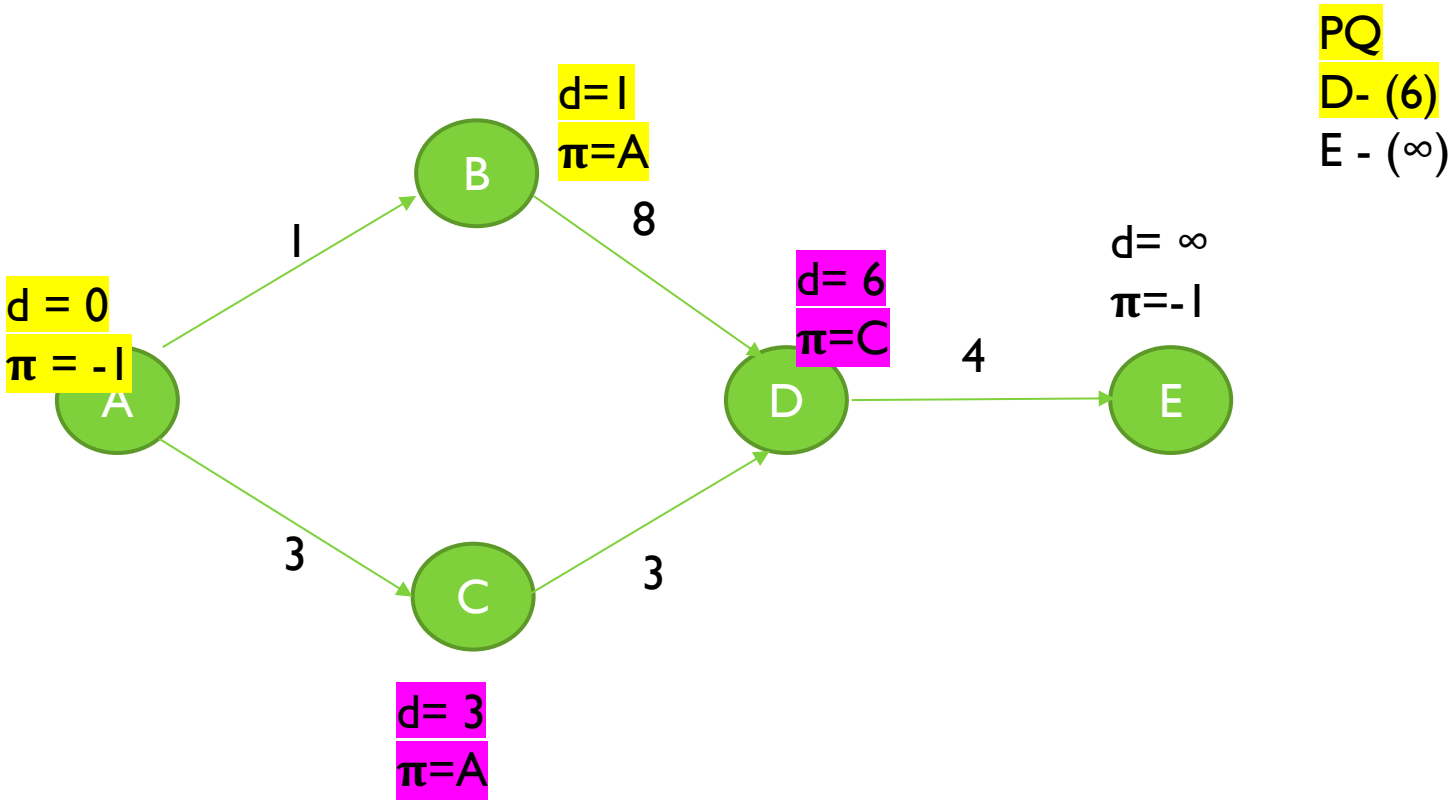
Dijsktra



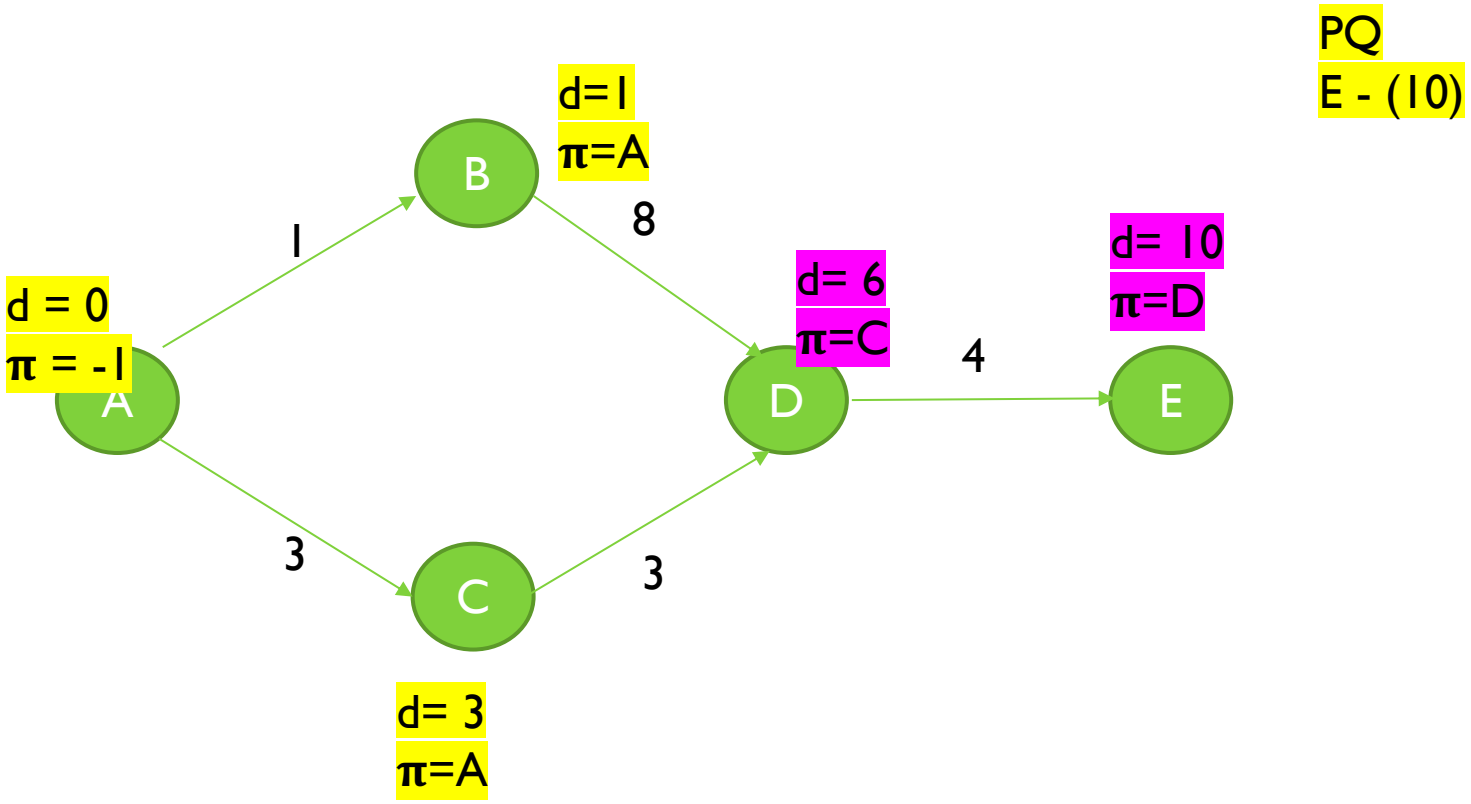
Dijsktra



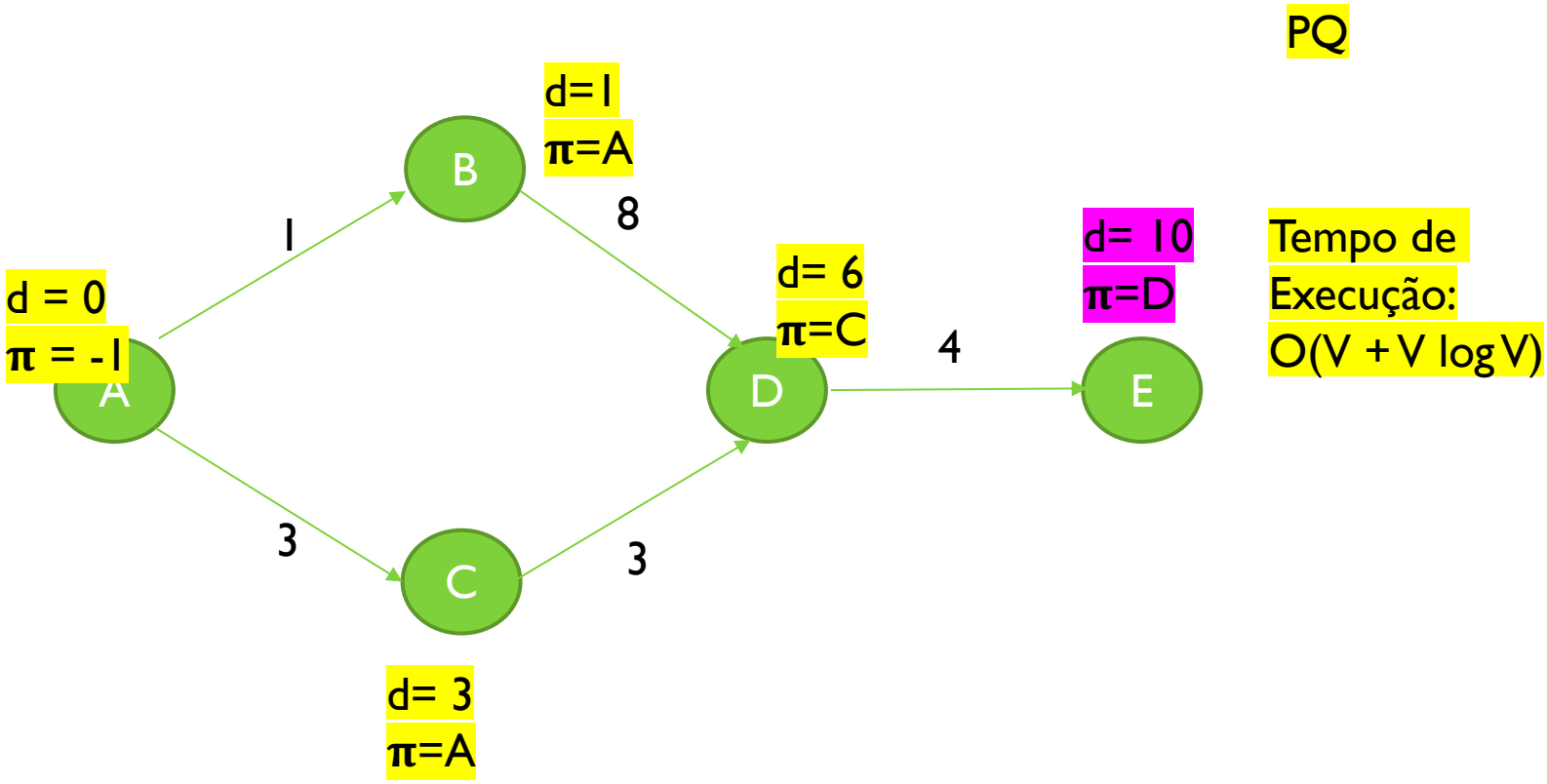
Dijsktra



Dijsktra



Dijsktra


PQ
d=1
π=A
B
1
8
d = 0
π = -1
A
d= 6
π=C
d= 10
π=D
4
D
E
Tempo de
Execução:
O(V + V log V)
3
3
C
d= 3
π=A

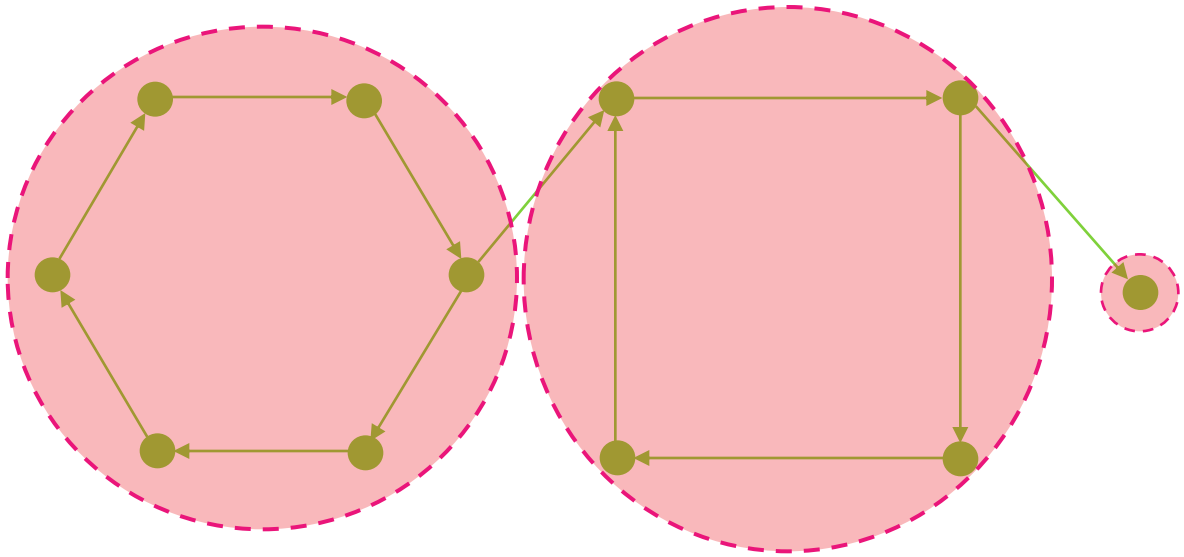
Componentes Fortemente Ligados

- Definição:
  - Dado grafo orientado $G = (V, E)$ um componente fortemente ligado (SCC) é um conjunto máximo de vértices $U \subseteq V$, tal que para $u,v \in U, u$ é atingível a partir de v, e v é atingível a partir de u
    - Obs: um vértice simples é um SCC

- Outras definições:
  - Grafo transposto de $G = (V, E)$
    - $G^T = (V, E^T)$ tal que: $E^T = \{(u,v) : (v,u) \in E\}$
  - OBS: G e G^T têm os mesmos SCCs

Componentes Fortemente Ligados

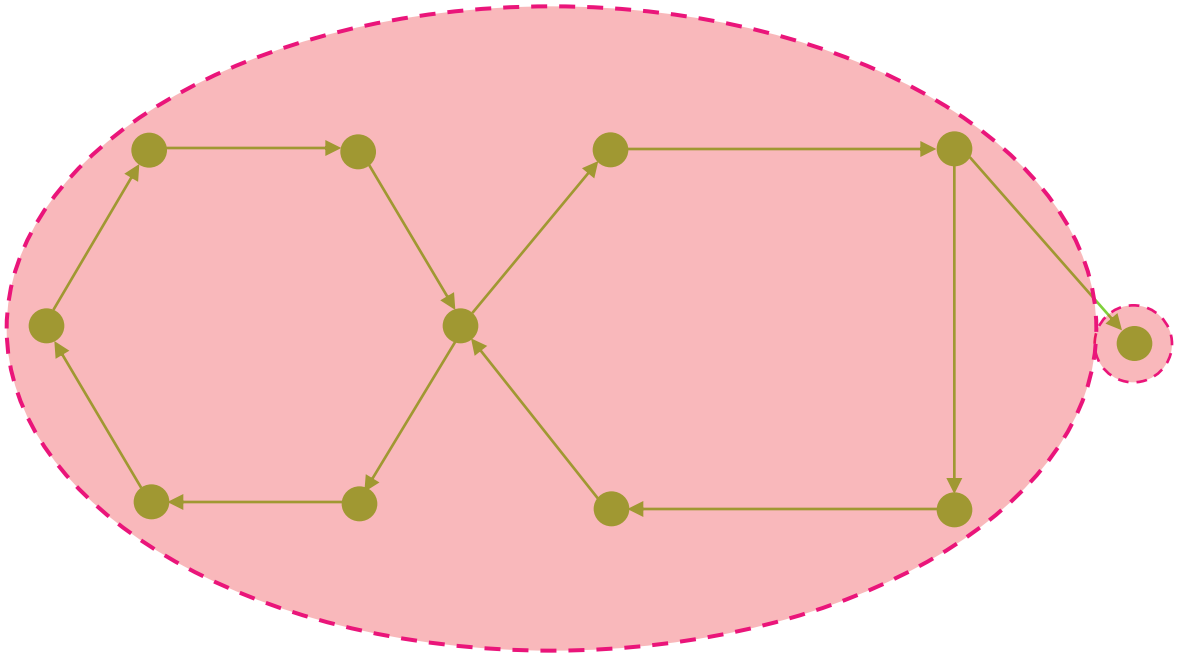
- Quantos SCCs ?



- Resposta: 3

Componentes Fortementes Ligados

- Quantos SCCs ?



- Resposta: 2

Componentes Fortementes Ligados

- Algoritmo

SCCs (G)

Executar DFS (G) para cálculo do tempo de fim $f[v]$ para cada v

Representar G^T

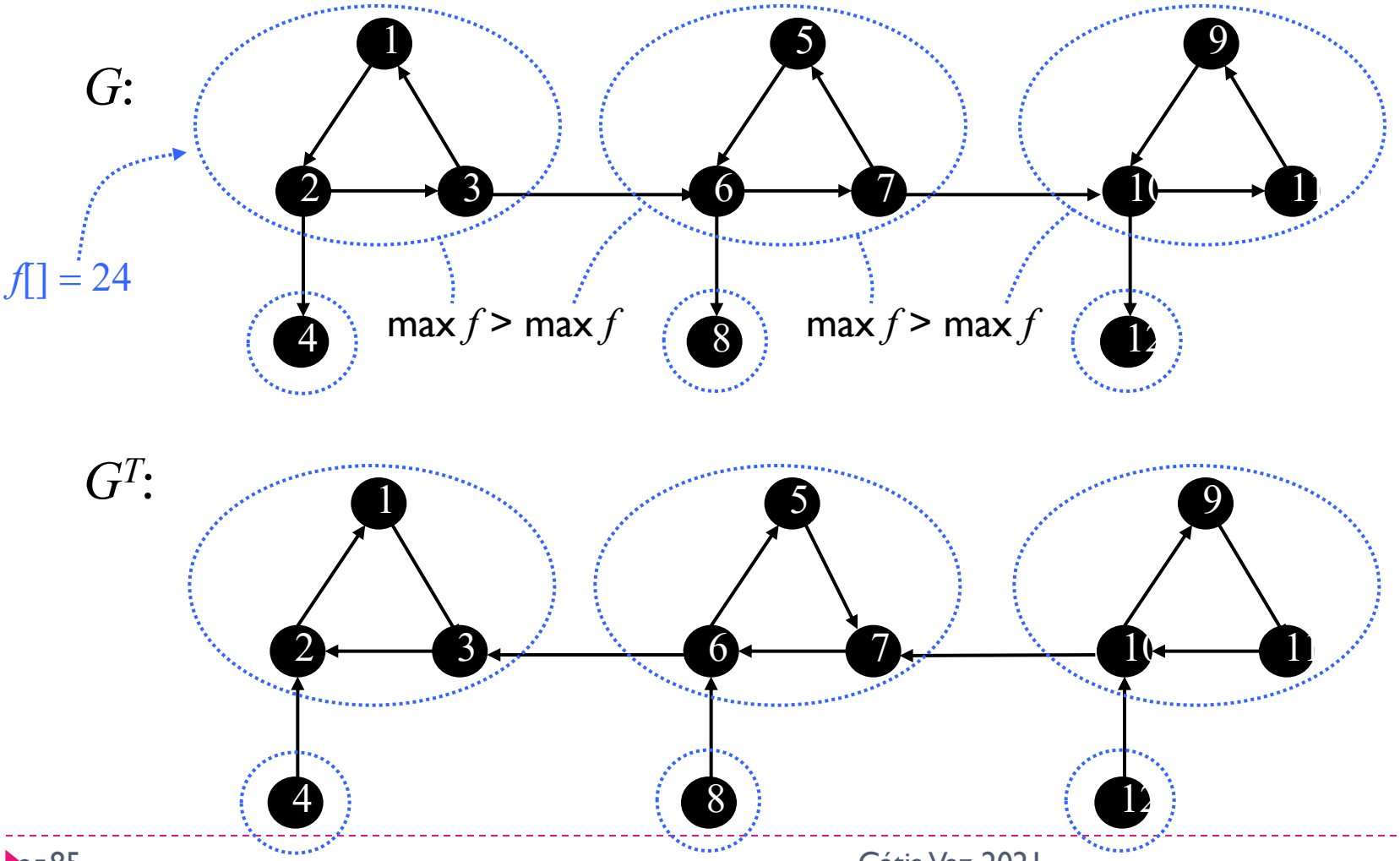
Executar DFS (G^T)

(no ciclo principal de DFS considerar os vértices por ordem decrescente de tempo de fim de DFS (G))

Cada árvore de DFS encontrada corresponde a novo SCC

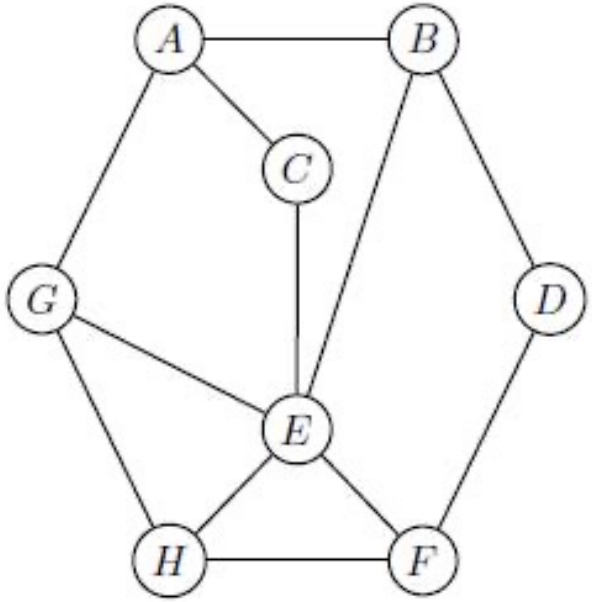
- Tempo de execução: $O(V+E)$

Componentes Fortementes Ligados



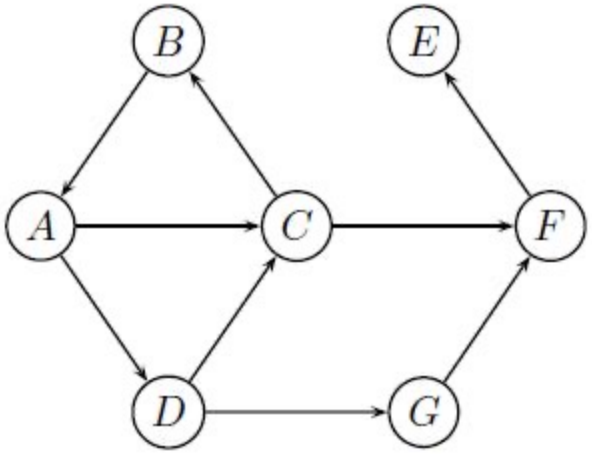
Exercício

- Qual a sequência de vértices visitados numa travessia em largura primeiro (BFS) sobre o grafo abaixo, com origem no vértice A ? Considere que os vértices são visitados por ordem alfabética e que os vértices adjacentes de um vértice também são visitados por ordem alfabética.



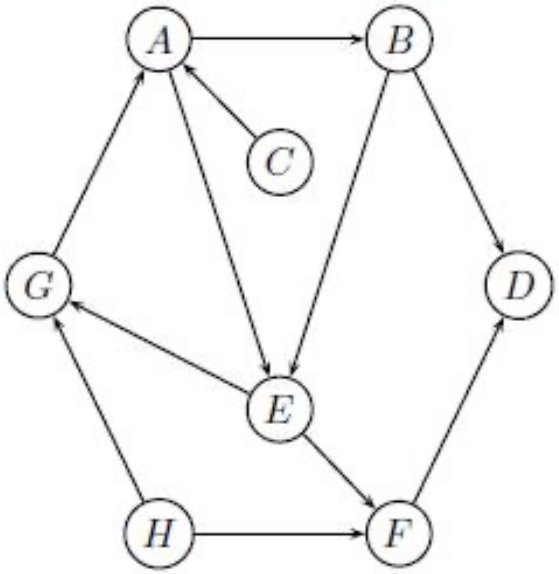
Exercício

- Qual a sequência de vértices visitados numa travessia em profundidade primeiro (DFS) sobre o grafo abaixo, com origem no vértice A ? Considere que os vértices são visitados por ordem alfabética e que os vértices adjacentes de um vértice também são visitados por ordem alfabética



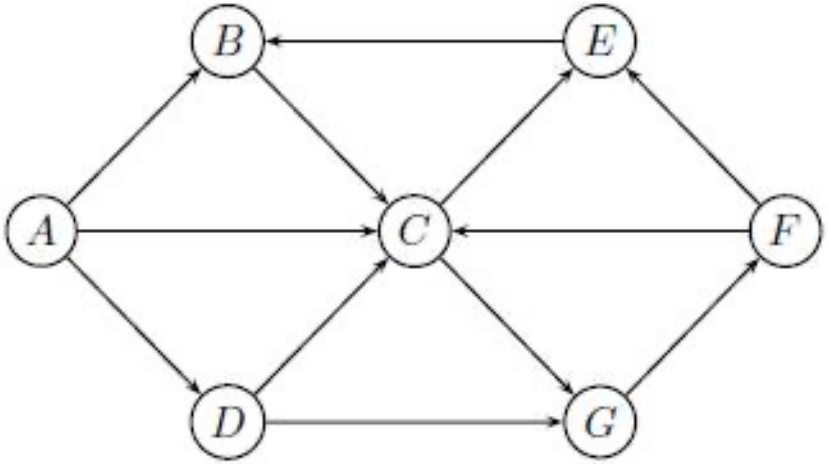
Exercício

- Considere uma travessia em profundidade primeiro (DFS) do grafo abaixo, com origem em A, e na qual os vértices são visitados por ordem alfabética e os vértices adjacentes de um vértice também são visitados por ordem alfabética. Indique os arcos de árvore, arcos para trás, arcos para a frente e arcos de cruzamento.



Exercício

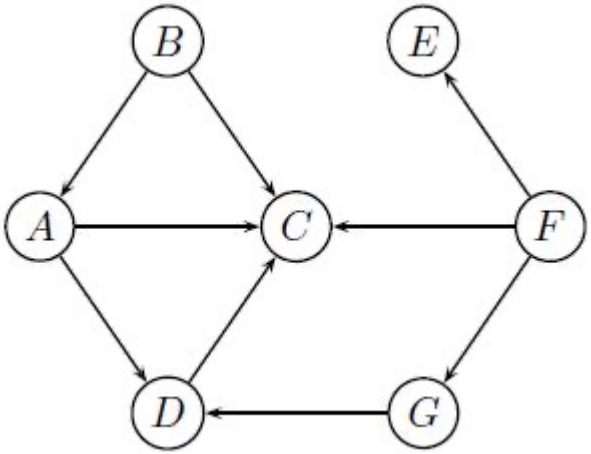
- Considere uma travessia em profundidade primeiro (DFS) sobre o grafo abaixo, com origem no vértice A. Considere ainda que os vértices são visitados por ordem alfabética e que os vértices adjacentes de um vértice são também visitados por ordem alfabética. Qual o número de arcos para trás e de cruzamento que resultam da execução desta travessia?



Ordenação Topológica

- Qual das sequências de vértices corresponde a uma ordenação topológica do grafo abaixo ?

a. <A,B,C,D,E,F,G>
b. <B,A,D,C,G,F,E>
c. <B,A,D,F,E,G,C>
d. <B,F,A,E,D,C,G>
e. <C,G,E,B,A,D,F>
f. <F,E,G,D,C,B,A>
g. <F,G,E,B,A,D,C>



Componentes fortemente ligados

- Qual o número de componentes fortemente ligados do grafo dirigido da figura abaixo?