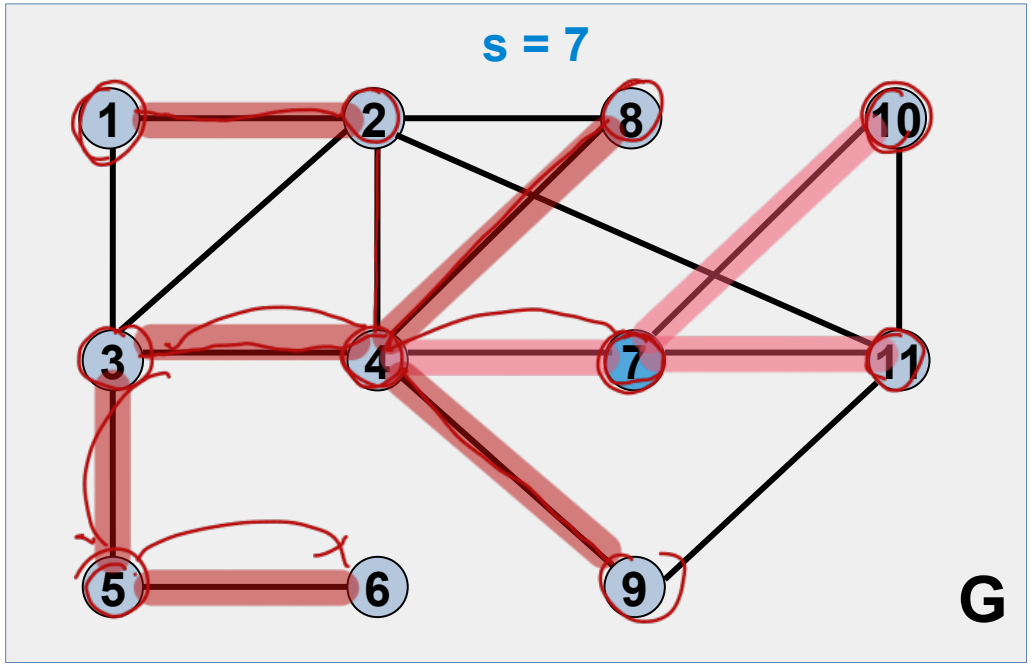
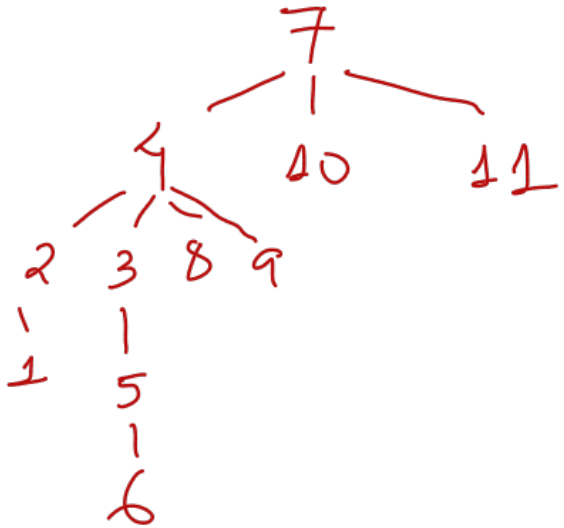
Busca em Largura

GRAFOS

Busca em Largura



F



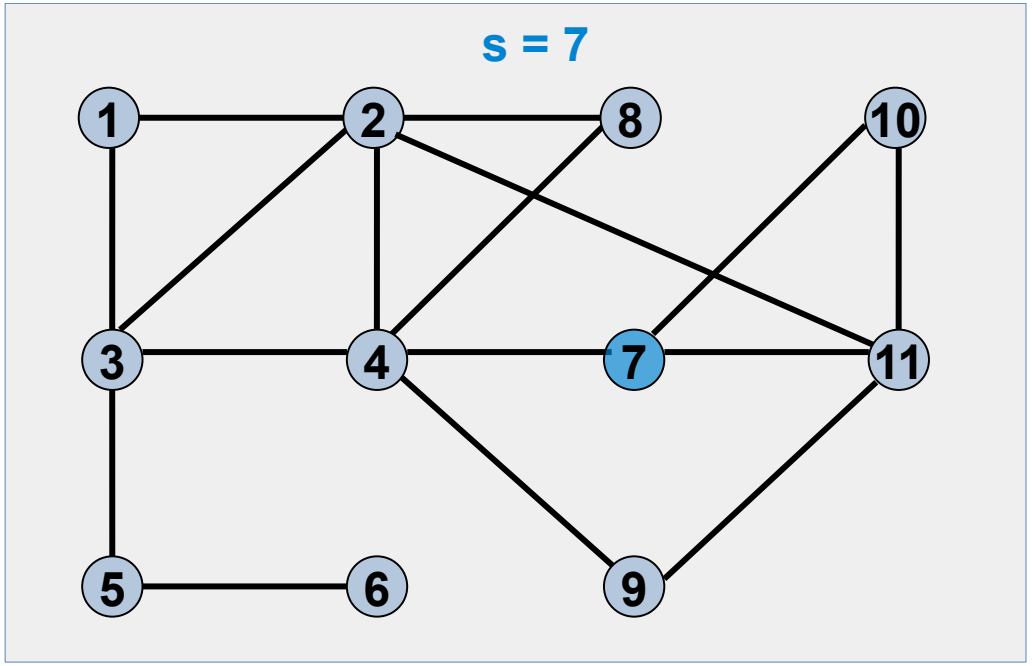
Vértices	1	2	3	4	5	6	7	8	9	10	11
Visitado	1	1	1	1	1	1	1	1	1	1	1
Predecessor	2	4	4	7	3	5	–	4	4	7	7

Algoritmo

```
BuscaLargura(Grafo G, vértice s)
  s.visitado = 1;
  Cria fila vazia F;
  ENFILEIRA(F,s);
  Enquanto F.tamanho >0 faça
    u = DESENFILEIRA(F);
    Para todo vértice v ∈ N(u) faça
      Se v.visitado==0 então
        v.visitado = 1;
        v.predecessor = u;
        ENFILEIRA(F,s);
```

BuscaLargura(Grafo *G*, vértice *s*)

```
s.visitado = 1;
Cria fila vazia F;
ENFILEIRA(F,s);
Enquanto F.tamanho >0 faça
  u = DESENFILEIRA(F);
  Para todo vértice v ∈ N(u) faça
    Se v.visitado==0 então
      v.visitado = 1;
      v.predecessor = u;
      ENFILEIRA(F,s);
```



	1	2	3	4	5	6	7	8	9	10	11
1	---	1	1								
2	1	---	1	1				1			1
3	1	1	---	1	1						
4		1	1	---			1	1	1		
5			1		---	1					
6					1	---					
7				1			---			1	1
8		1		1				---			
9				1					---		1
10							1			---	1
11		1					1		1	1	---

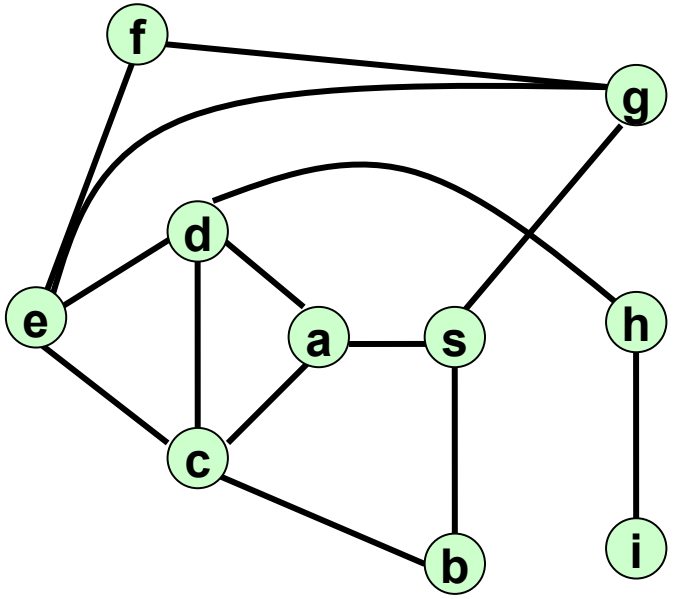
F

Vértices	1	2	3	4	5	6	7	8	9	10	11
Visitado	1	1	1	1	1	1	1	1	1	1	1
Predecessor	2	4	4	7	3	5	—	4	4	7	7

Algoritmo

```
Procedimento largura(G=(V,E))
  // A(v) é a lista de vértices adjacentes a v
  Definir uma fila Q
  Definir uma raiz s ∈ V
  Insere_fila(Q,s)
  marcar(s)
  enquanto Q é não vazia
     se w ∈ A(v) não é marcado então
          insere_fila(Q,w); marcar(w); visitar(v,w)
     senão
        se (v,w) não visitada
           visitar(v,w)
```

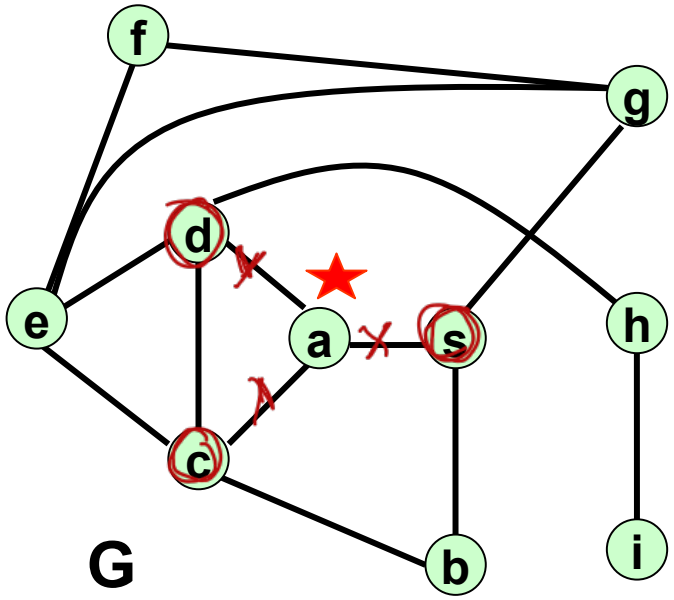
Algoritmo



G

```
Procedimento largura(G=(V,E))
  // A(v) é a lista de vértices adjacentes a v
  Definir uma fila Q
  Definir uma raiz s ∈ V
  Insere_fila(Q,s); marcar(s)
  enquanto Q é não vazia
     v ← remove_fila(Q)
     para w ∈ A(v) efetuar
     se w não é marcado então
          insere_fila(Q,w); marcar(w); visitar(v,w)
     senão
        se (v,w) não visitada
            visitar(v,w)
```

Algoritmo

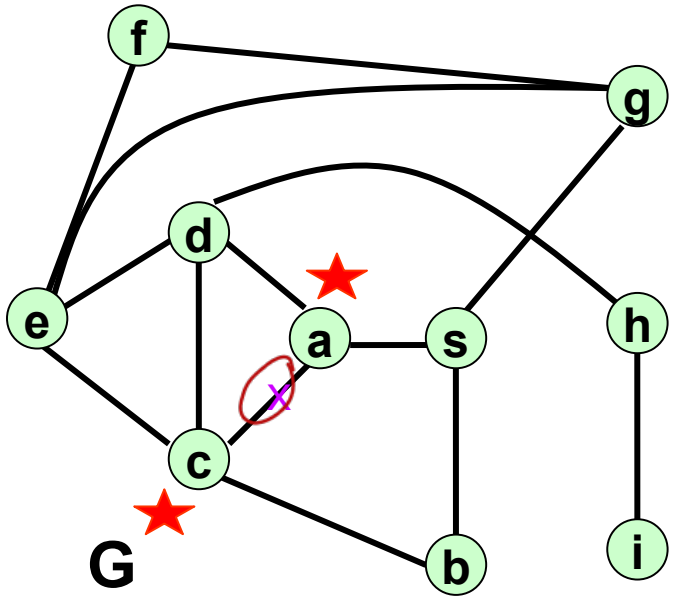


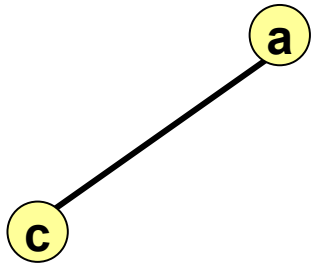
Q

~~a~~	c	d	s			

a

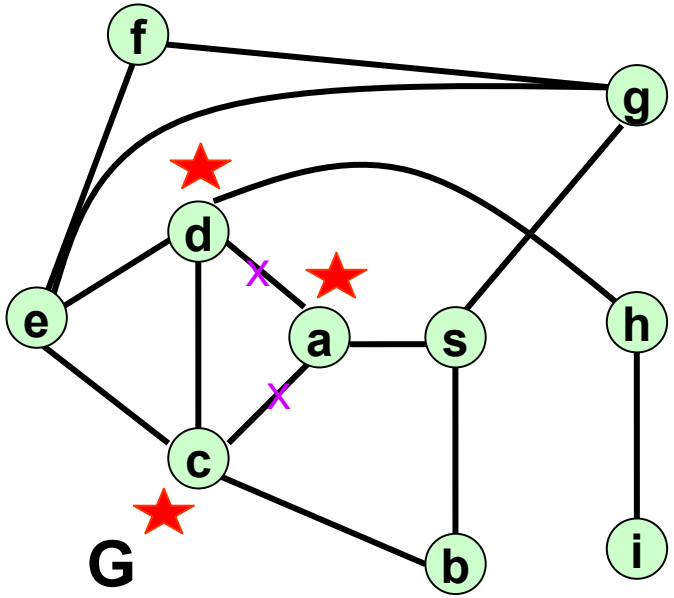
Algoritmo

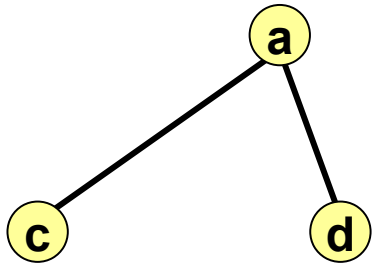





```
Procedimento largura(G=(V,E))
  // A(v) é a lista de vértices adjacentes a v
  Definir uma fila Q
  Definir uma raiz s ∈ V
  Insere_fila(Q,s); marcar(s)
  enquanto Q é não vazia
     v ← remove_fila(Q)
     para w ∈ A(v) efetuar
     se w não é marcado então
          insere_fila(Q,w); marcar(w); visitar(v,w)
     senão
        se (v,w) não visitada
            visitar(v,w)
```

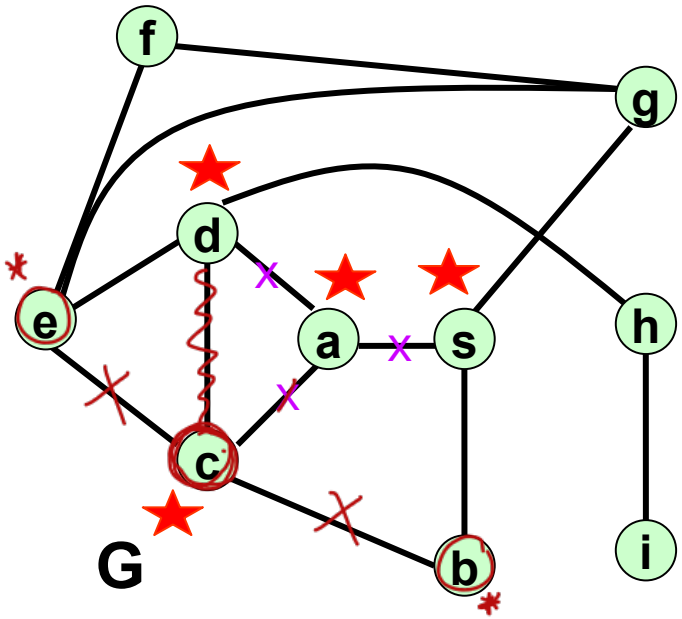
Algoritmo

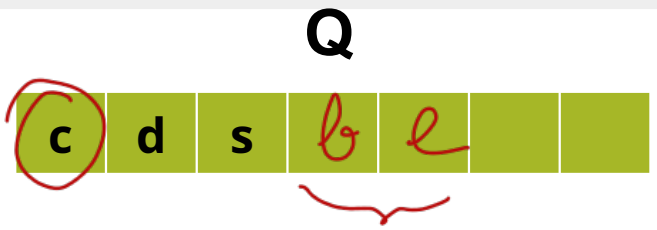


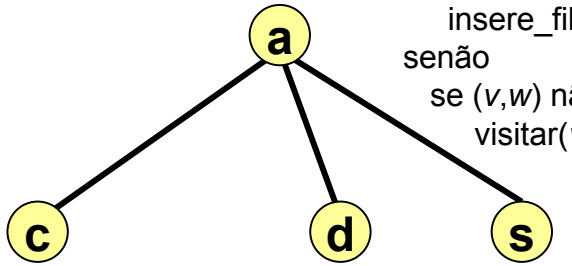



```
Procedimento largura(G=(V,E))
  // A(v) é a lista de vértices adjacentes a v
  Definir uma fila Q
  Definir uma raiz s ∈ V
  Insere_fila(Q,s); marcar(s)
  enquanto Q é não vazia
     v ← remove_fila(Q)
     para w ∈ A(v) efetuar
     se w não é marcado então
          insere_fila(Q,w); marcar(w); visitar(v,w)
     senão
        se (v,w) não visitada
            visitar(v,w)
```

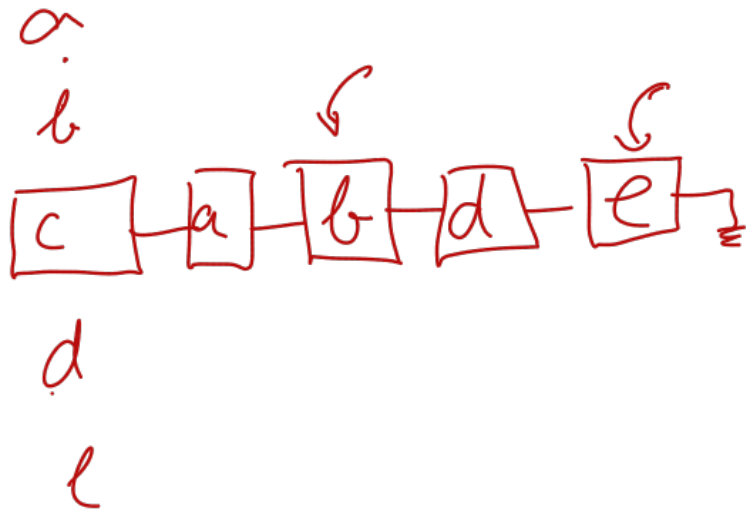
Algoritmo



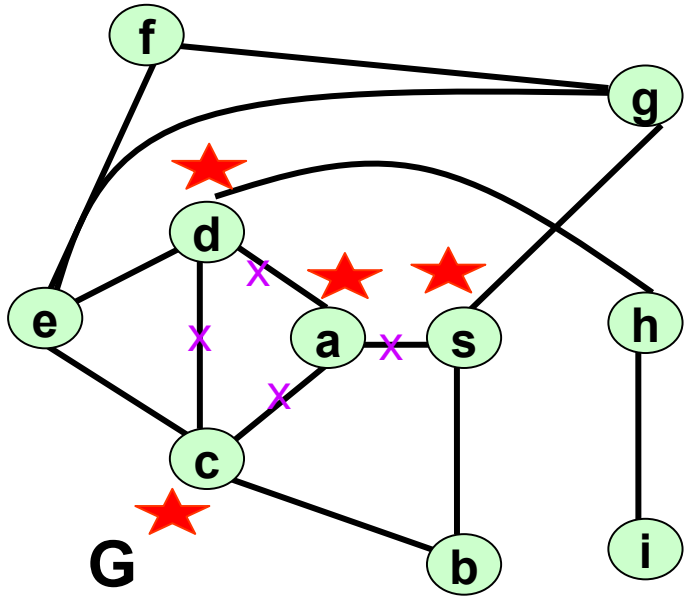




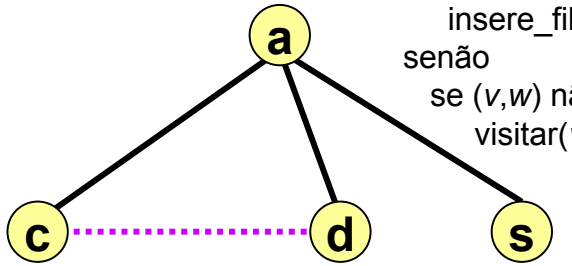

```
Procedimento largura(G=(V,E))
  // A(v) é a lista de vértices adjacentes a v
  Definir uma fila Q
  Definir uma raiz s ∈ V
  Insere_fila(Q,s); marcar(s)
  enquanto Q é não vazia
     v ← remove_fila(Q)
     para w ∈ A(v) efetuar
     se w não é marcado então
          insere_fila(Q,w); marcar(w); visitar(v,w)
     senão
       se (v,w) não visitada
          visitar(v,w)
```



Algoritmo

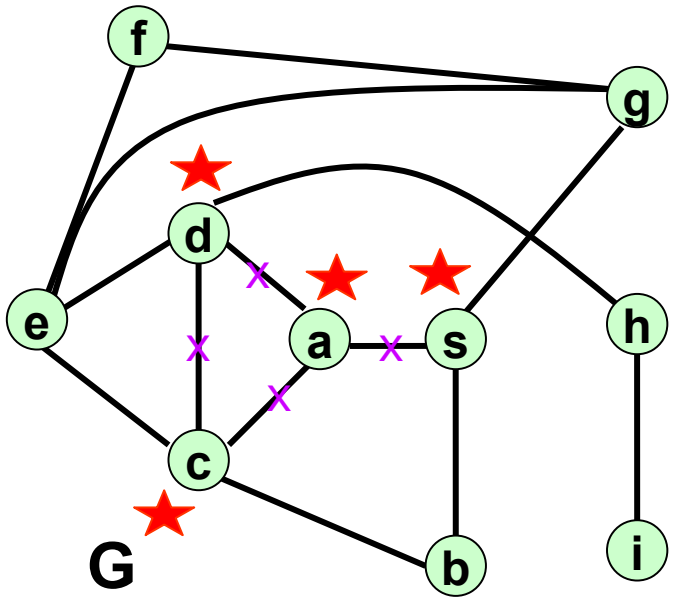


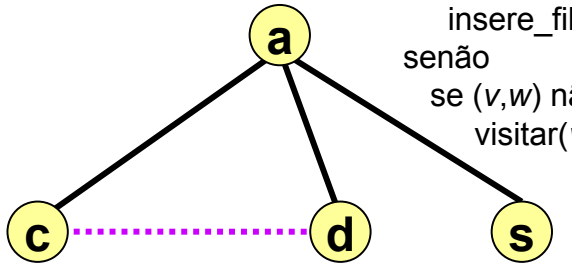
Q




```
Procedimento largura(G=(V,E))
  // A(v) é a lista de vértices adjacentes a v
  Definir uma fila Q
  Definir uma raiz s ∈ V
  Insere_fila(Q,s); marcar(s)
  enquanto Q é não vazia
     v ← remove_fila(Q)
     para w ∈ A(v) efetuar
     se w não é marcado então
          insere_fila(Q,w); marcar(w); visitar(v,w)
     senão
       se (v,w) não visitada
          visitar(v,w)
```

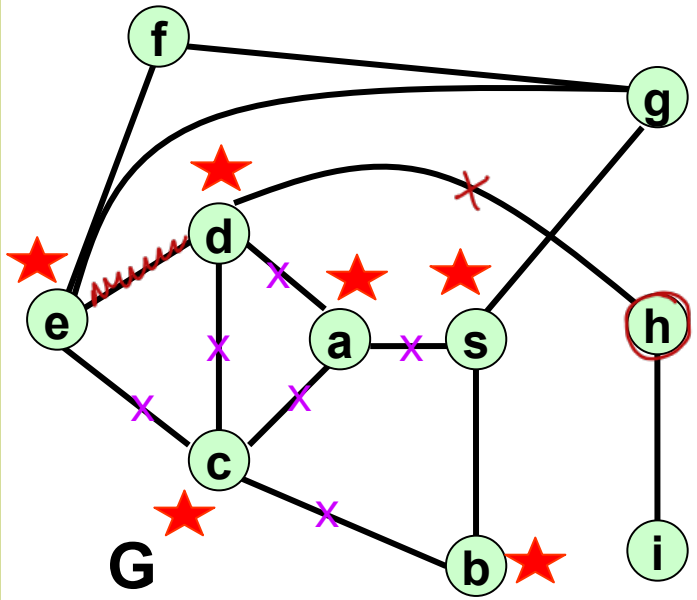
Algoritmo



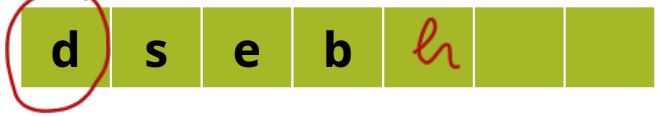


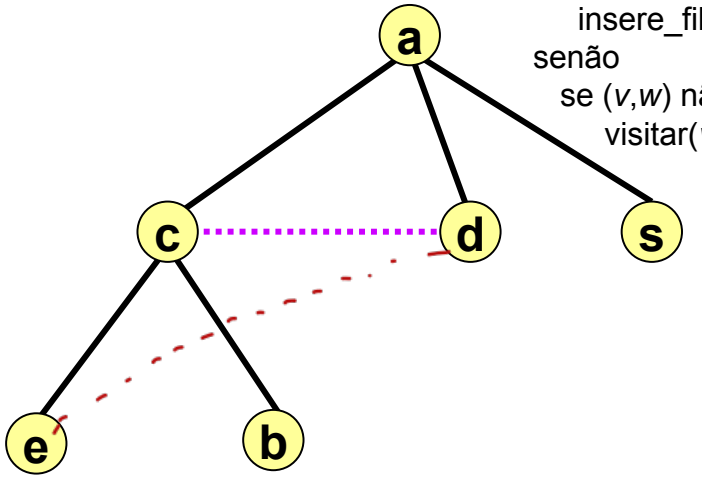

```
Procedimento largura(G=(V,E))
  // A(v) é a lista de vértices adjacentes a v
  Definir uma fila Q
  Definir uma raiz s ∈ V
  Insere_fila(Q,s); marcar(s)
  enquanto Q é não vazia
     v ← remove_fila(Q)
     para w ∈ A(v) efetuar
     se w não é marcado então
          insere_fila(Q,w); marcar(w); visitar(v,w)
     senão
       se (v,w) não visitada
          visitar(v,w)
```

Algoritmo



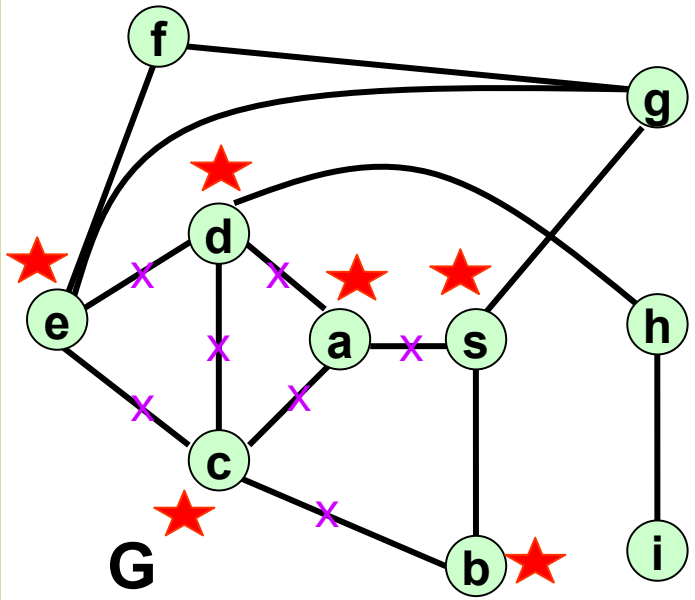
Q





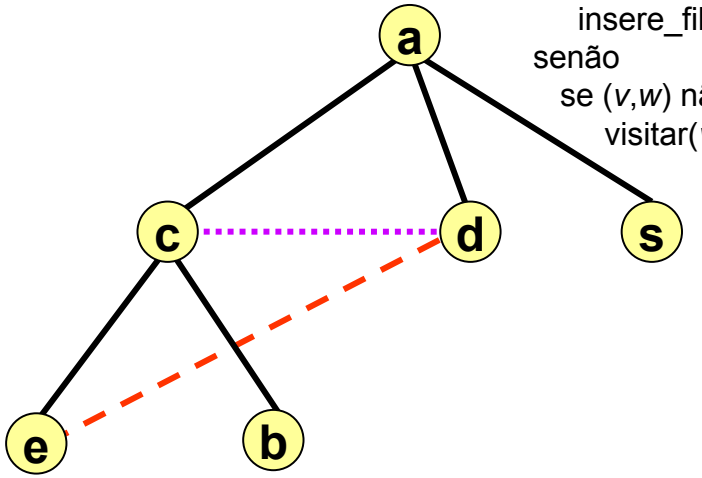

```
Procedimento largura(G=(V,E))
  // A(v) é a lista de vértices adjacentes a v
  Definir uma fila Q
  Definir uma raiz s ∈ V
  Insere_fila(Q,s); marcar(s)
  enquanto Q é não vazia
     v ← remove_fila(Q)
     para w ∈ A(v) efetuar
     se w não é marcado então
          insere_fila(Q,w); marcar(w); visitar(v,w)
     senão
       se (v,w) não visitada
          visitar(v,w)
```

Algoritmo



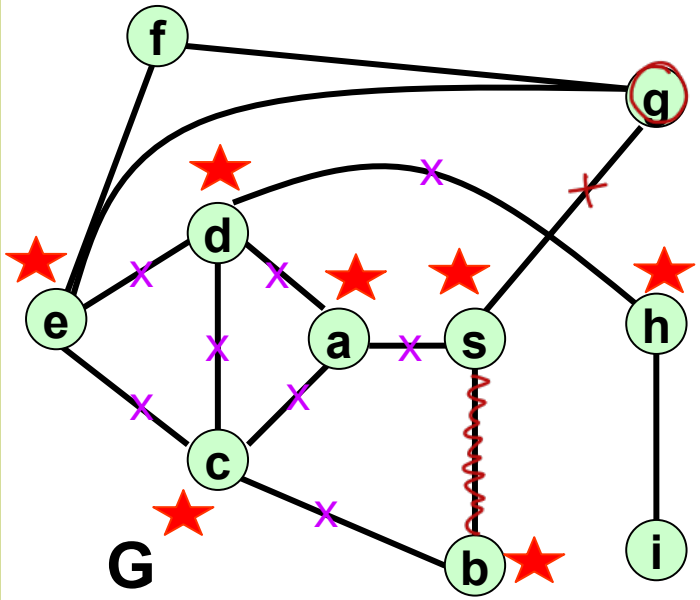
Q

s	e	b				

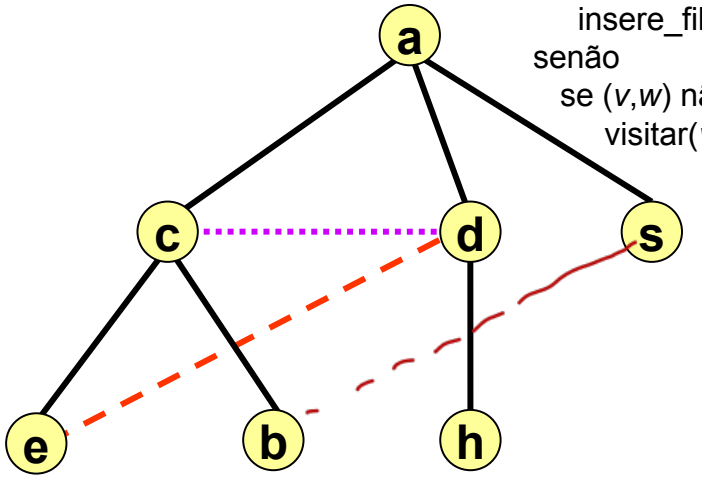



```
Procedimento largura(G=(V,E))
  // A(v) é a lista de vértices adjacentes a v
  Definir uma fila Q
  Definir uma raiz s ∈ V
  Insere_fila(Q,s); marcar(s)
  enquanto Q é não vazia
     v ← remove_fila(Q)
     para w ∈ A(v) efetuar
     se w não é marcado então
          insere_fila(Q,w); marcar(w); visitar(v,w)
     senão
       se (v,w) não visitada
          visitar(v,w)
```

Algoritmo

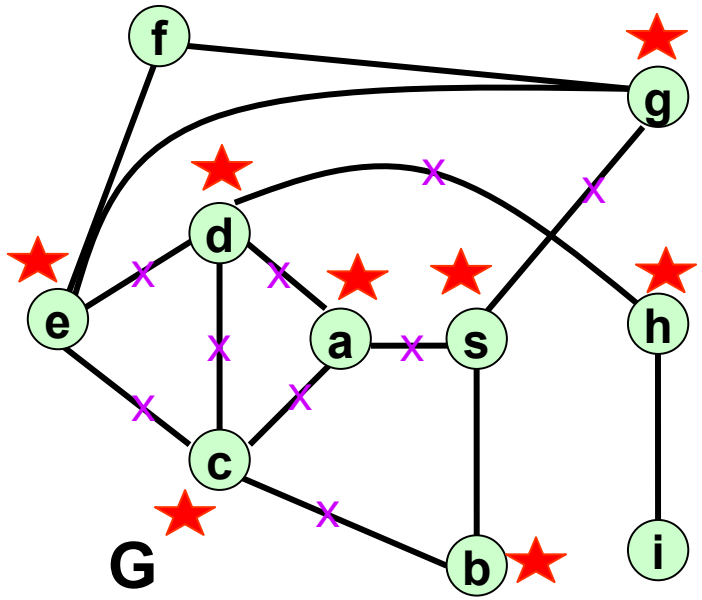


Q



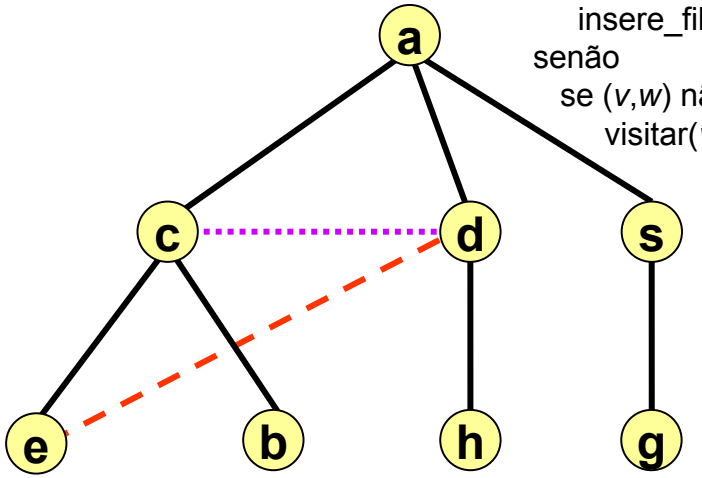

```
Procedimento largura(G=(V,E))
  // A(v) é a lista de vértices adjacentes a v
  Definir uma fila Q
  Definir uma raiz s ∈ V
  Insere_fila(Q,s); marcar(s)
  enquanto Q é não vazia
     v ← remove_fila(Q)
     para w ∈ A(v) efetuar
     se w não é marcado então
          insere_fila(Q,w); marcar(w); visitar(v,w)
     senão
       se (v,w) não visitada
          visitar(v,w)
```

Algoritmo



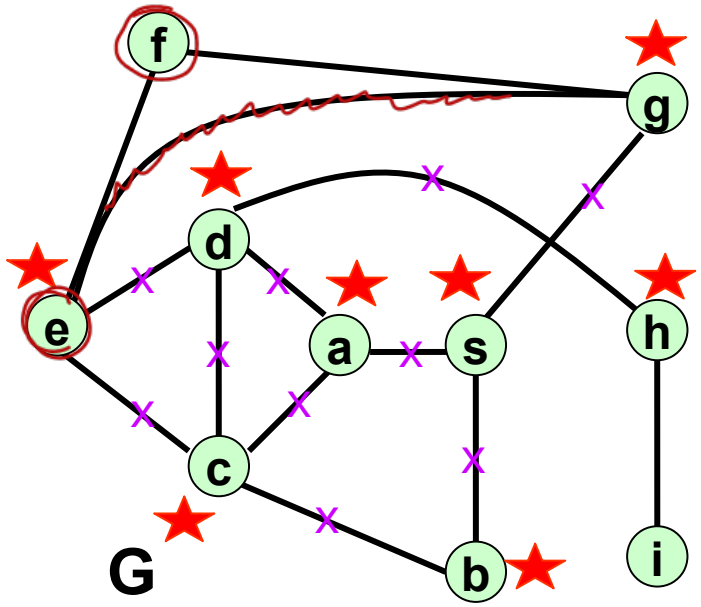
Q

e	b	h	g			

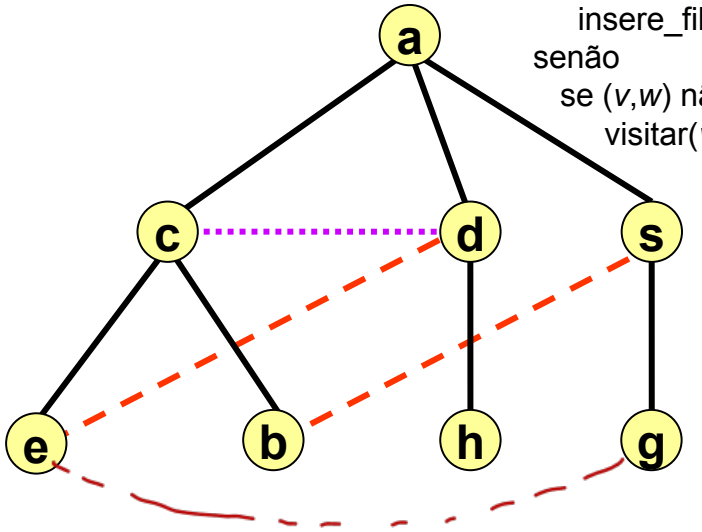


Procedimento largura(G=(V,E))
// $A(v)$ é a lista de vértices adjacentes a v
Definir uma fila Q
Definir uma raiz $s \in V$
Insere_fila(Q,s); marcar(s)
enquanto Q é não vazia
  v ← remove_fila(Q)
  para $w \in A(v)$ efetuar
  se w não é marcado então
    insere_fila(Q,w); marcar(w); visitar(v,w)
  senão
    se (v,w) não visitada
      visitar(v,w)

Algoritmo

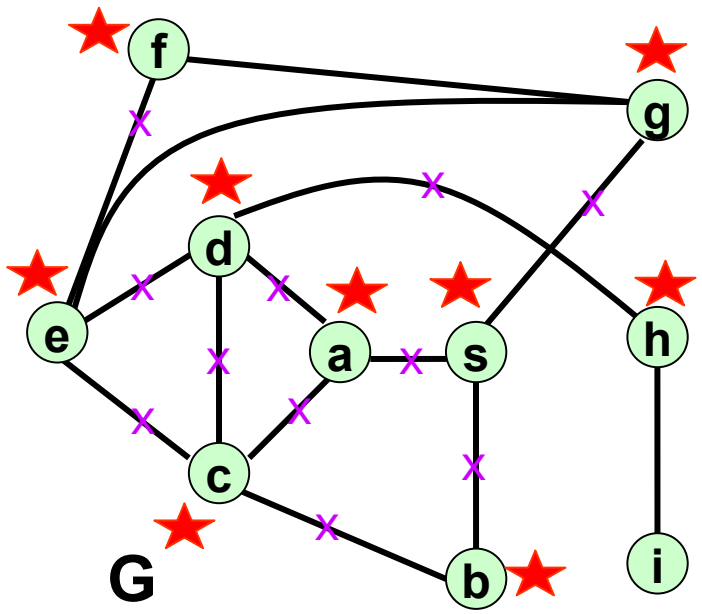


Q



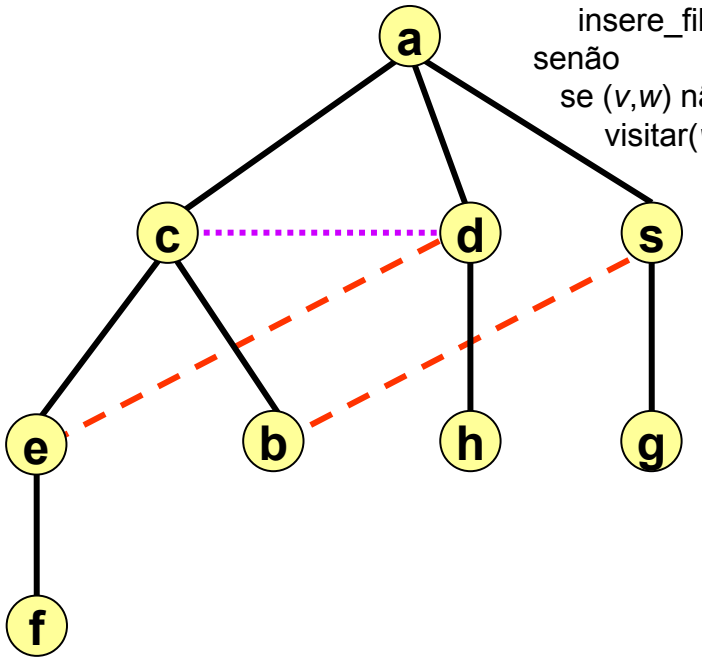

```
Procedimento largura(G=(V,E))
  // A(v) é a lista de vértices adjacentes a v
  Definir uma fila Q
  Definir uma raiz s ∈ V
  Insere_fila(Q,s); marcar(s)
  enquanto Q é não vazia
     v ← remove_fila(Q)
     para w ∈ A(v) efetuar
     se w não é marcado então
          insere_fila(Q,w); marcar(w); visitar(v,w)
     senão
       se (v,w) não visitada
          visitar(v,w)
```

Algoritmo



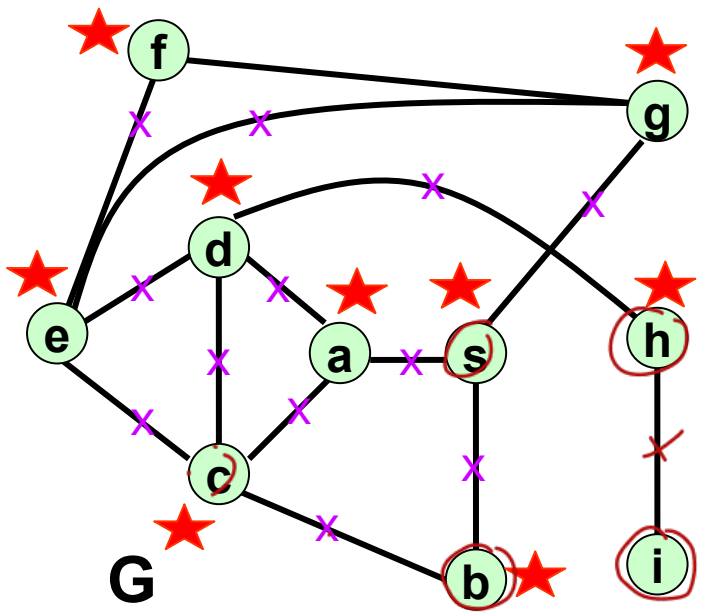
Q

b	h	g	f			

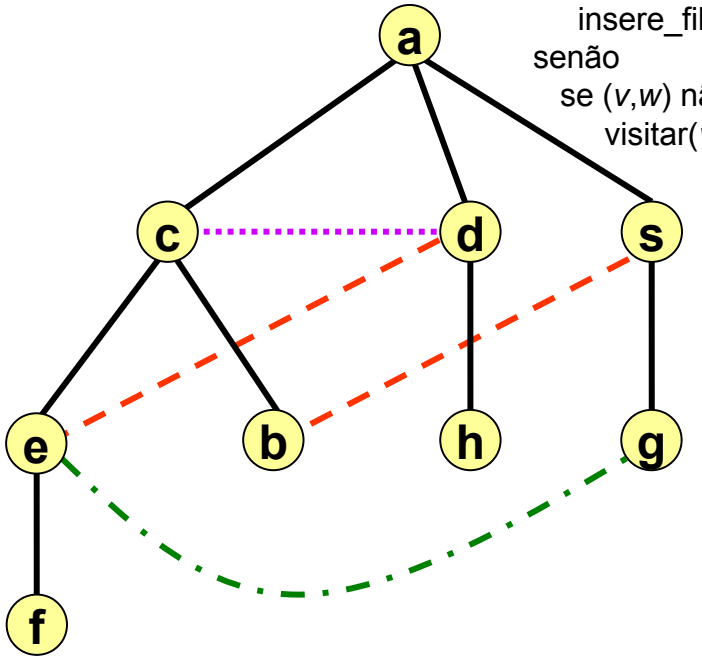



```
Procedimento largura(G=(V,E))
  // A(v) é a lista de vértices adjacentes a v
  Definir uma fila Q
  Definir uma raiz s ∈ V
  Insere_fila(Q,s); marcar(s)
  enquanto Q é não vazia
     v ← remove_fila(Q)
     para w ∈ A(v) efetuar
     se w não é marcado então
          insere_fila(Q,w); marcar(w); visitar(v,w)
     senão
       se (v,w) não visitada
          visitar(v,w)
```

Algoritmo

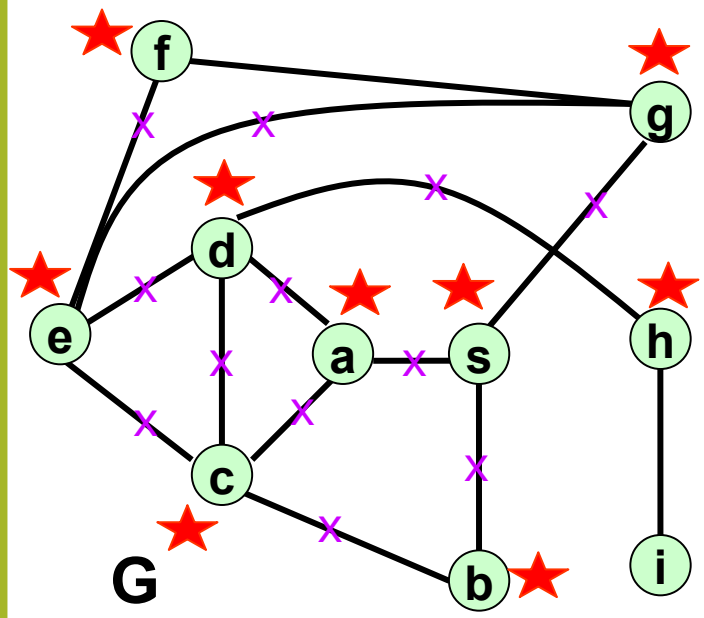


Q



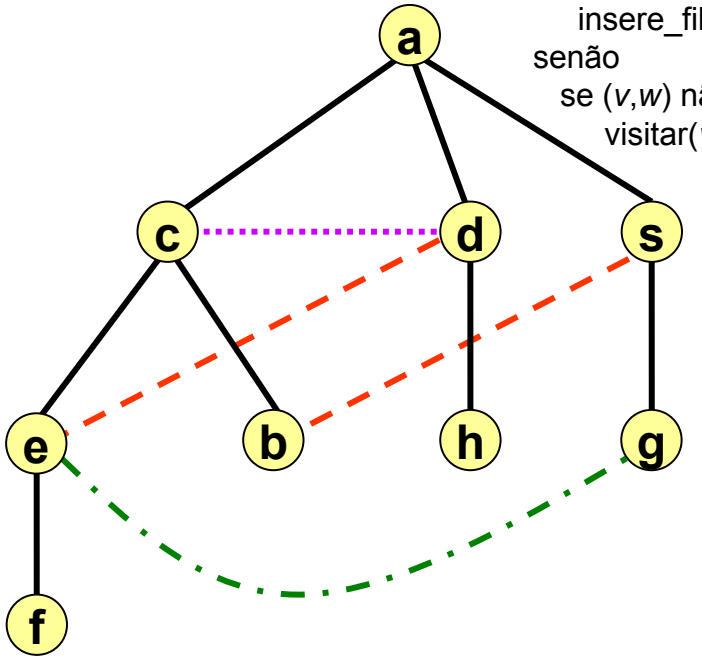

```
Procedimento largura(G=(V,E))
  // A(v) é a lista de vértices adjacentes a v
  Definir uma fila Q
  Definir uma raiz s ∈ V
  Insere_fila(Q,s); marcar(s)
  enquanto Q é não vazia
     v ← remove_fila(Q)
     para w ∈ A(v) efetuar
     se w não é marcado então
          insere_fila(Q,w); marcar(w); visitar(v,w)
     senão
       se (v,w) não visitada
          visitar(v,w)
```

Algoritmo



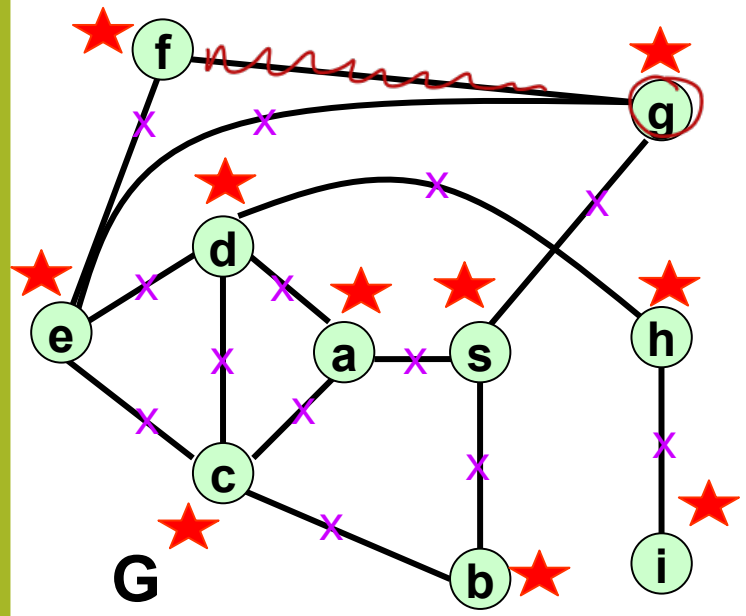
Q

h	g	f				

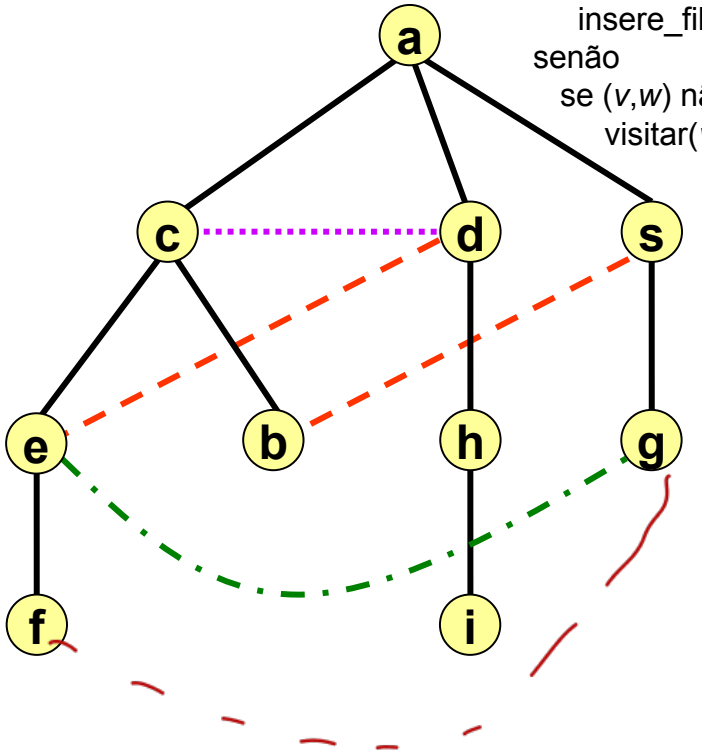



```
Procedimento largura(G=(V,E))
  // A(v) é a lista de vértices adjacentes a v
  Definir uma fila Q
  Definir uma raiz s ∈ V
  Insere_fila(Q,s); marcar(s)
  enquanto Q é não vazia
     v ← remove_fila(Q)
     para w ∈ A(v) efetuar
     se w não é marcado então
          insere_fila(Q,w); marcar(w); visitar(v,w)
     senão
       se (v,w) não visitada
          visitar(v,w)
```

Algoritmo

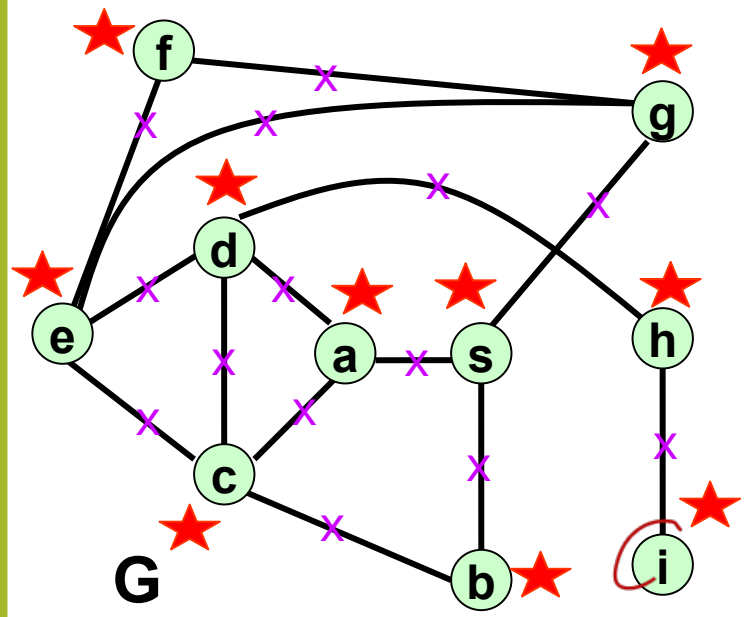


Q

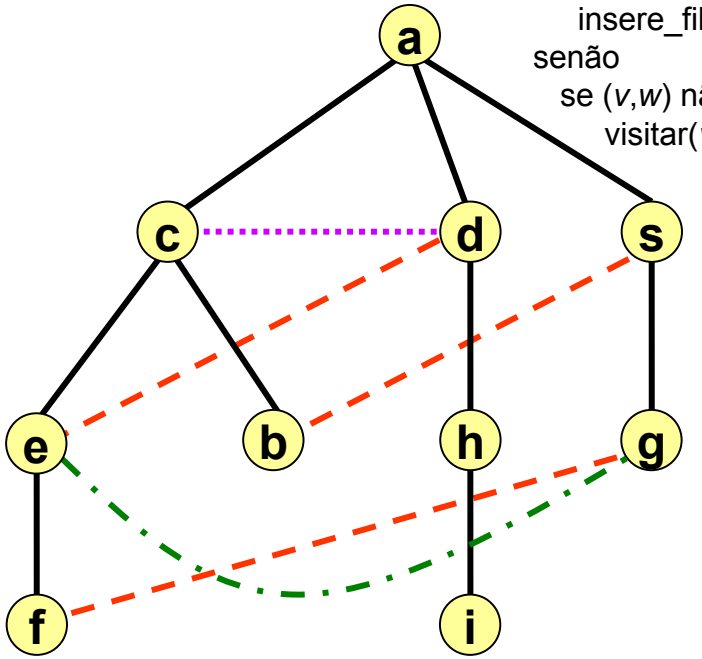



```
Procedimento largura(G=(V,E))
  // A(v) é a lista de vértices adjacentes a v
  Definir uma fila Q
  Definir uma raiz s ∈ V
  Insere_fila(Q,s); marcar(s)
  enquanto Q é não vazia
     v ← remove_fila(Q)
     para w ∈ A(v) efetuar
     se w não é marcado então
          insere_fila(Q,w); marcar(w); visitar(v,w)
     senão
       se (v,w) não visitada
          visitar(v,w)
```

Algoritmo

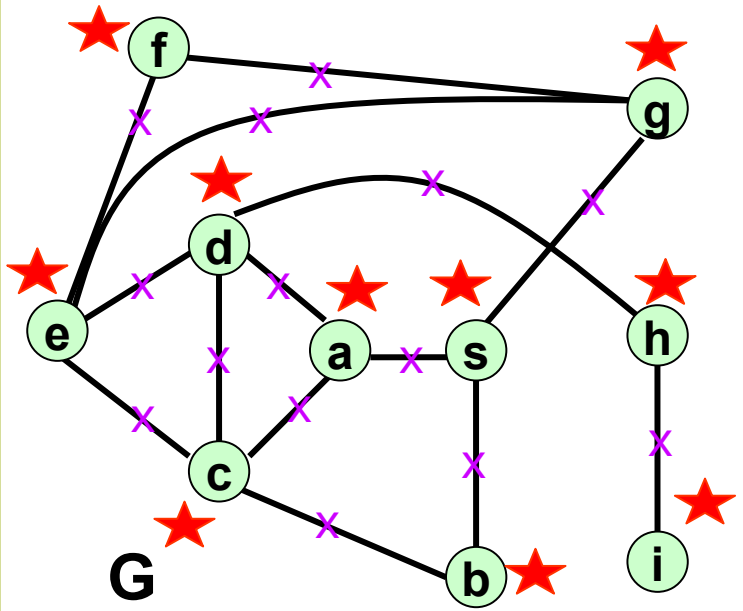


Q

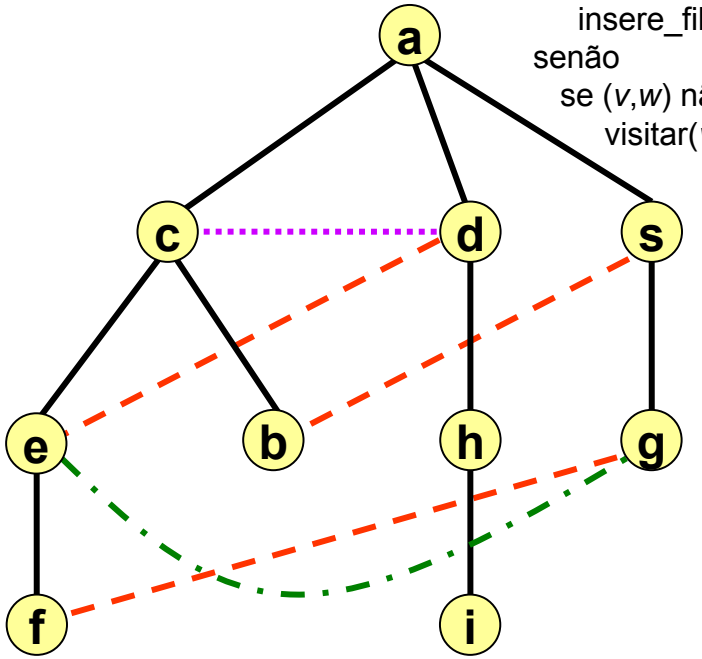



```
Procedimento largura(G=(V,E))
  // A(v) é a lista de vértices adjacentes a v
  Definir uma fila Q
  Definir uma raiz s ∈ V
  Insere_fila(Q,s); marcar(s)
  enquanto Q é não vazia
     v ← remove_fila(Q)
     para w ∈ A(v) efetuar
     se w não é marcado então
          insere_fila(Q,w); marcar(w); visitar(v,w)
     senão
       se (v,w) não visitada
          visitar(v,w)
```

Algoritmo

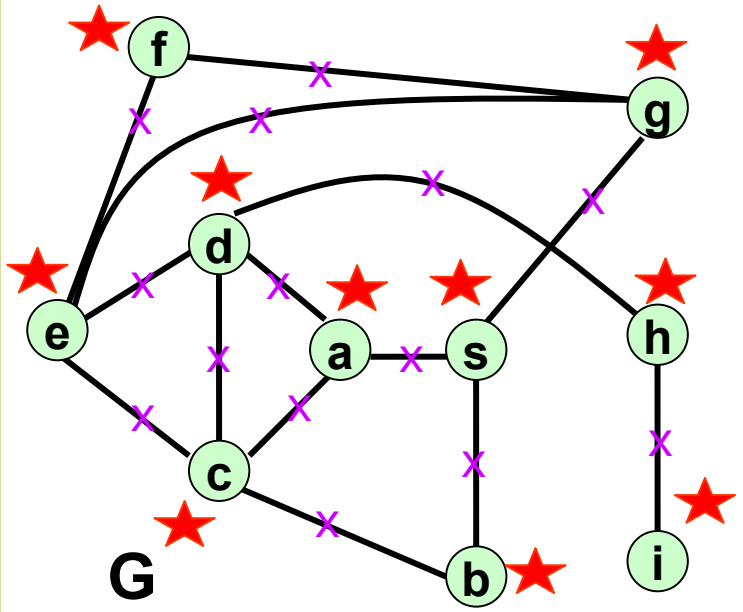


Q

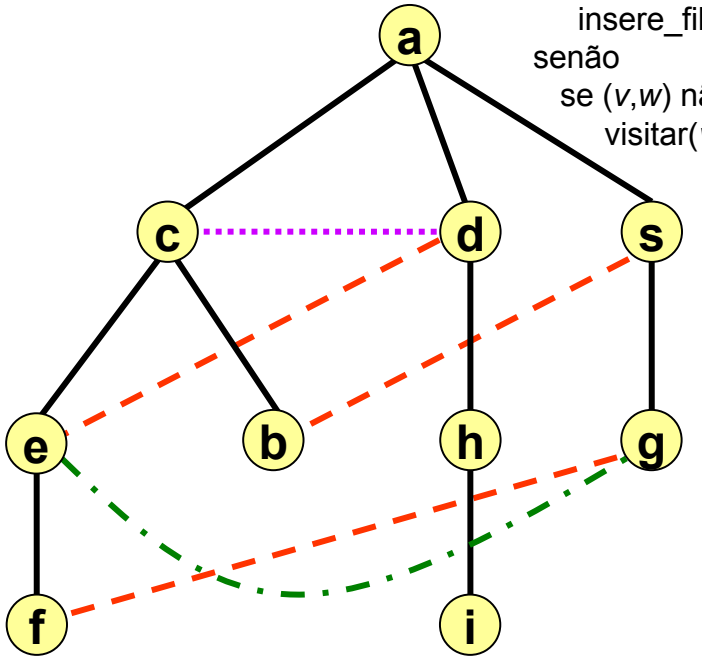



```
Procedimento largura(G=(V,E))
  // A(v) é a lista de vértices adjacentes a v
  Definir uma fila Q
  Definir uma raiz s ∈ V
  Insere_fila(Q,s); marcar(s)
  enquanto Q é não vazia
     v ← remove_fila(Q)
     para w ∈ A(v) efetuar
     se w não é marcado então
          insere_fila(Q,w); marcar(w); visitar(v,w)
     senão
       se (v,w) não visitada
          visitar(v,w)
```

Algoritmo

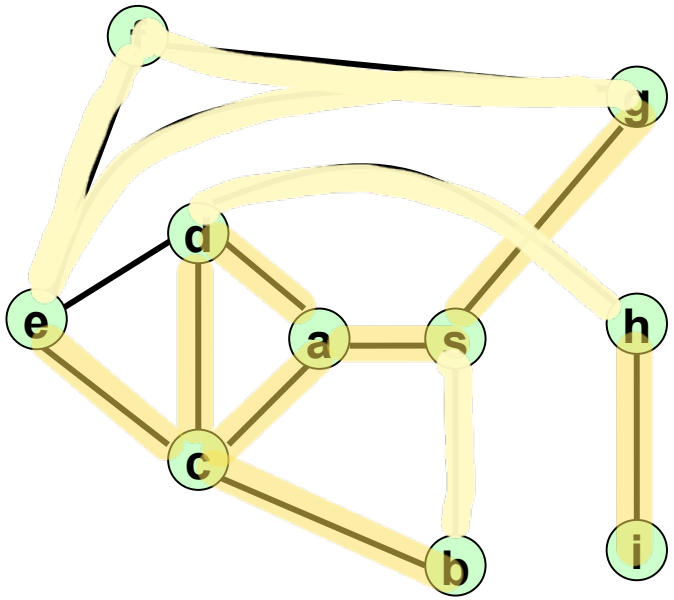


Q

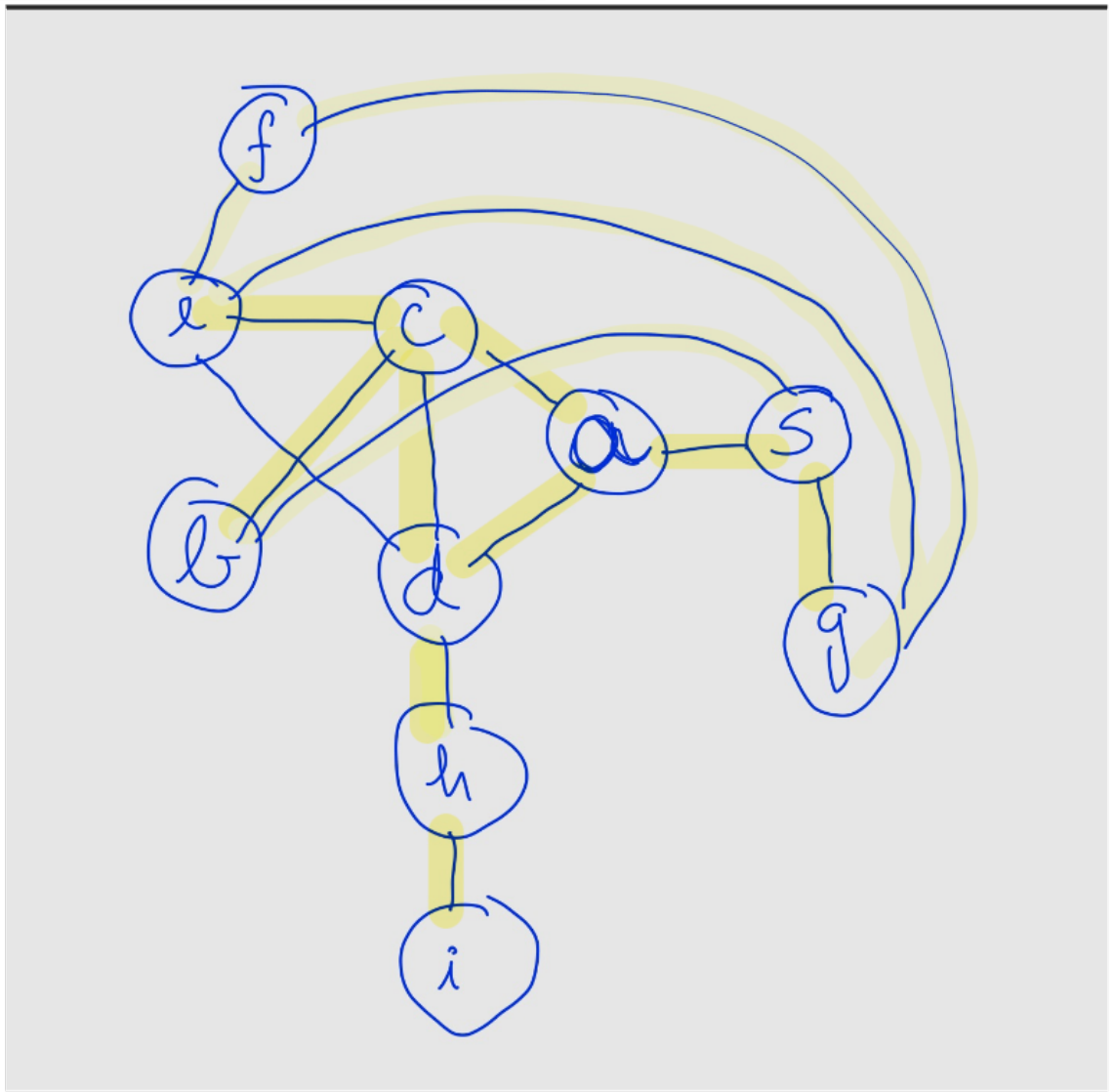



```
Procedimento largura(G=(V,E))
  // A(v) é a lista de vértices adjacentes a v
  Definir uma fila Q
  Definir uma raiz s ∈ V
  Insere_fila(Q,s); marcar(s)
  enquanto Q é não vazia
     v ← remove_fila(Q)
     para w ∈ A(v) efetuar
     se w não é marcado então
          insere_fila(Q,w); marcar(w); visitar(v,w)
     senão
       se (v,w) não visitada
          visitar(v,w)
```

Busca em Largura



Grafo



Busca em Largura

Seja E_T o conjunto das arestas visitadas em (I)

Teorema. Seja G(V,E) um grafo conexo, (v,w) ∈ E e $T(V,E_T)$ uma árvore de largura de G. Então nível(v) e nível(w) diferem, no máximo, de uma unidade.

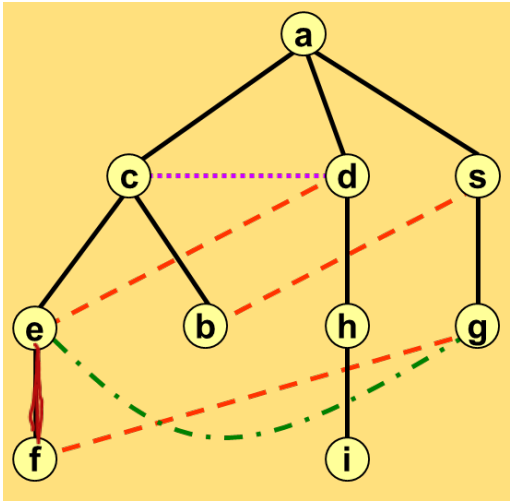
Prova. **EXERCÍCIO**

Busca em Largura

Observe que o teorema anterior divide as arestas (v,w) em duas classes:

1. |nível(v) – nível(w)| = 1

2. |nível(v) – nível(w)| = 0



Cada uma dessas classes pode ser subdividida em outras duas.

Busca em Largura

Considere a aresta (v,w) com nível(v) $\leq$ nível(w)

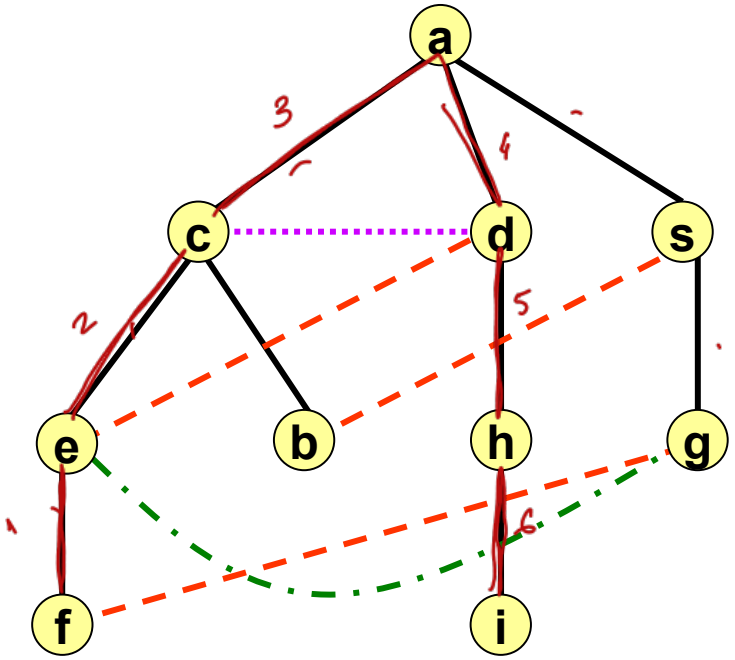
1. (v,w) é aresta ***pai*** (aresta da árvore), quando *v* é pai de *w* em T.

2. (*v*,*w*) é aresta ***tio*** quando nível(w) = nível(v) + 1 e (v,w)$\notin$ T

3. (v,w) é aresta ***irmão*** quando v e w possuem o mesmo pai em T

4. (v,w) é aresta ***primo*** quando nível(v) = nível(w) e v, w não possuem o mesmo pai em T

Busca em Largura



Aresta pai

Aresta irmão

Aresta tio

Aresta primo

Busca em Largura

Para algumas aplicações pode ser importante a ordem em que os vértices foram removidos da fila.

Define-se largura(v) como sendo o número de ordem em que v foi retirado da fila Q.

Busca em Largura

O diâmetro de um grafo é a maior distância entre qualquer par de vértices.

Como usar Busca em Largura para determinar o diâmetro de um grafo?
Qual a complexidade?