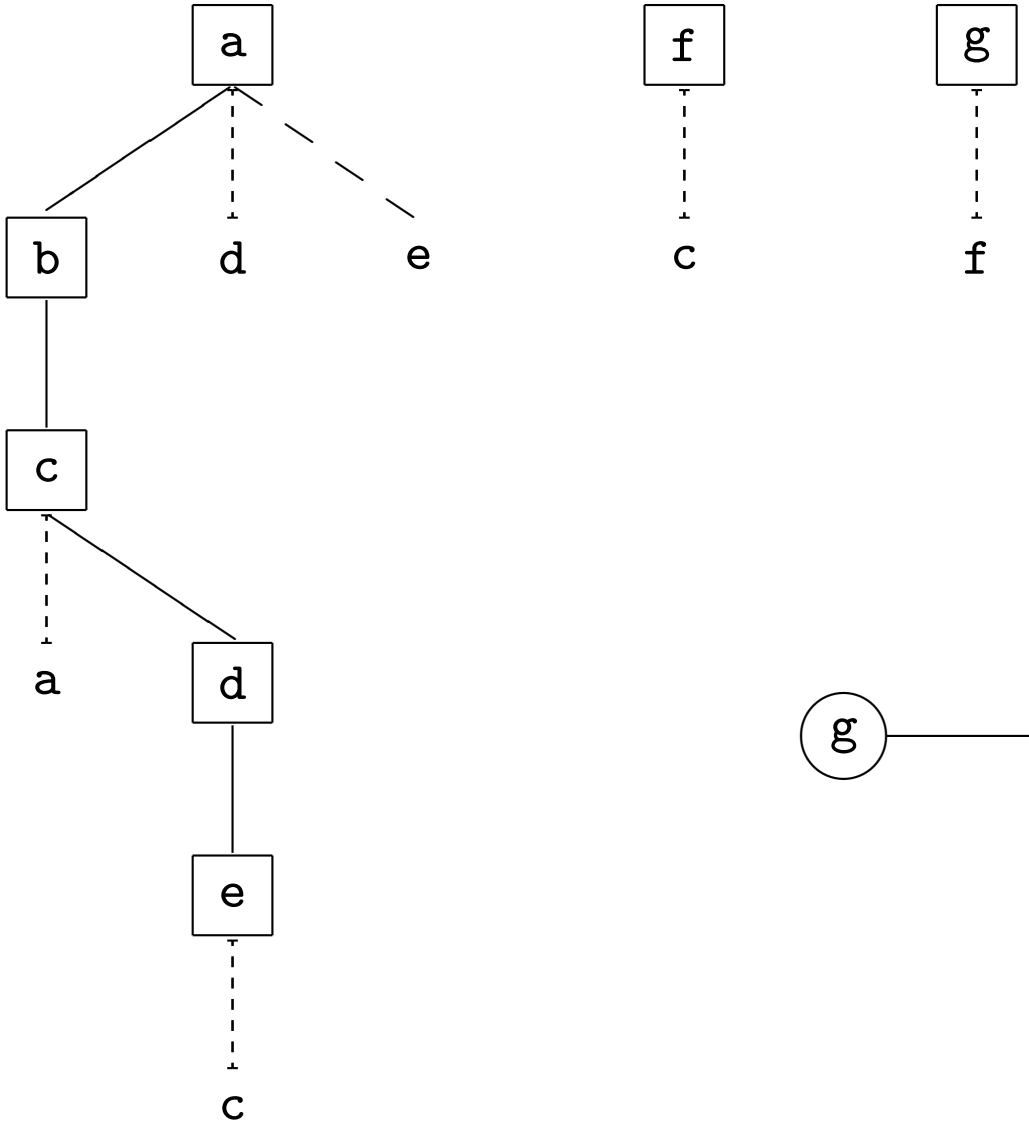
Capítulo III

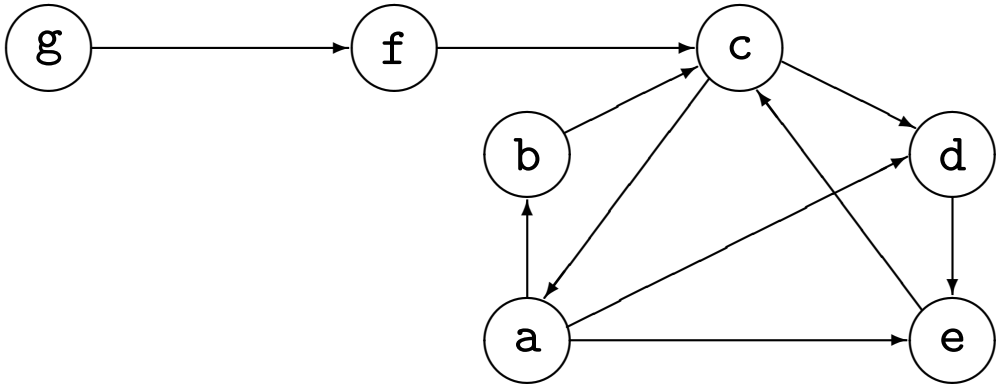
Percursos em Profundidade e em Largura (num grafo orientado ou não orientado)

Percurso em Profundidade



Ordem:

a b c d e f g



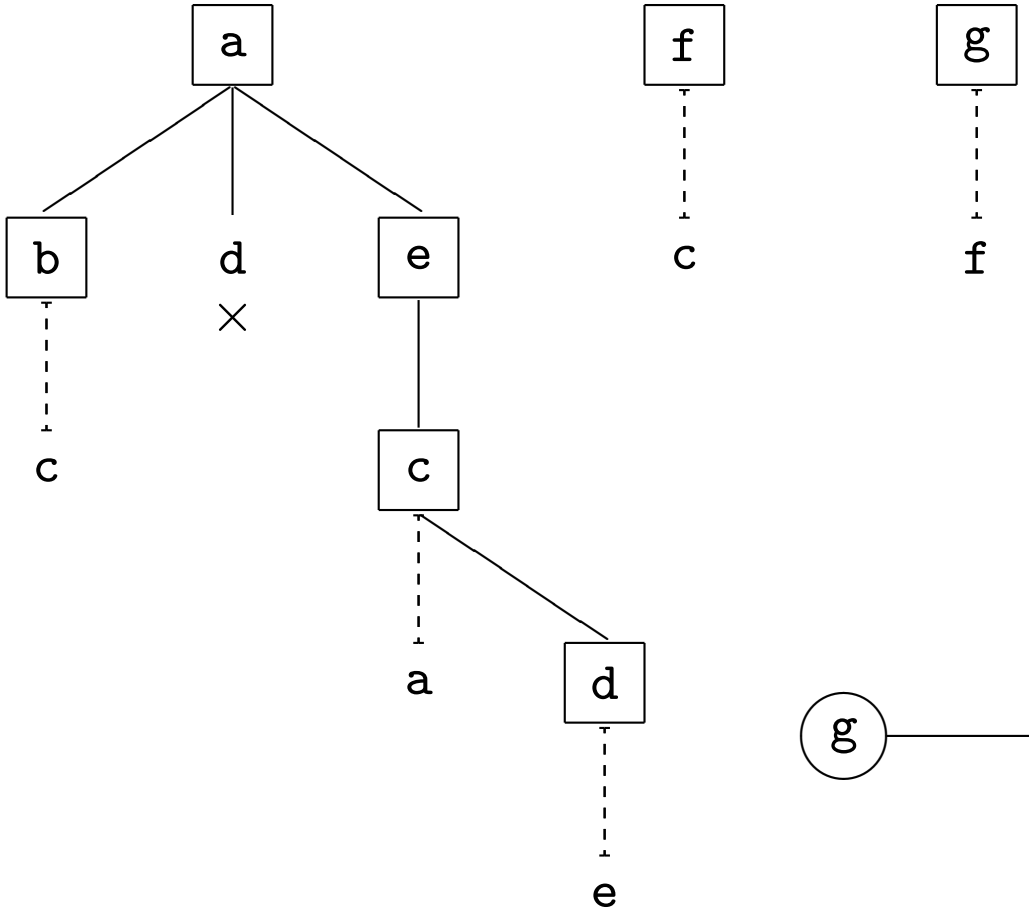
Percurso em Profundidade

```
void dfsTraversal( Graph graph )
{
    boolean[] explored = new boolean[ graph.numVertices() ];

    for every Vertex v in graph.vertices()
        explored[v] = false;

    for every Vertex v in graph.vertices()
        if ( !explored[v] )
            dfsExplore(graph, explored, v);
}
```

Árvore em Profundidade (recursivo)

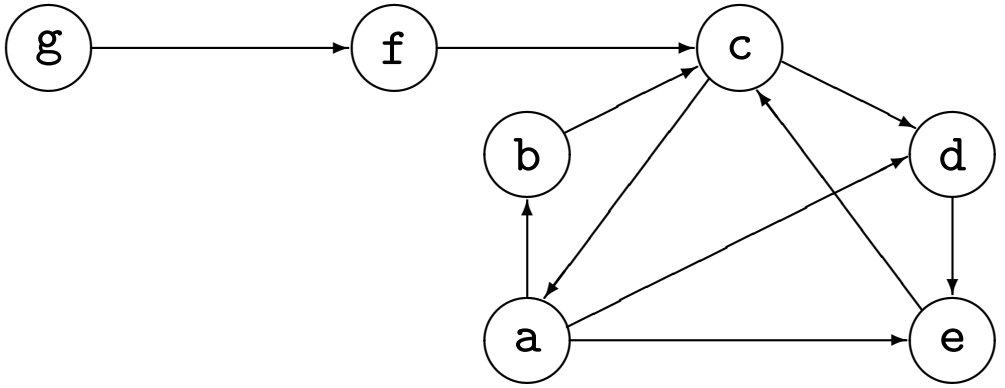
```
void dfsExplore( Graph graph,  boolean[] explored,  Vertex root )
{
    TREAT(root);
    explored[root] = true;
    for every Vertex v in graph.outAdjacentVertices(root)
        if ( !explored[v] )
            dfsExplore(graph, explored, v);
}
```

Percurso em Profundidade



Ordem:

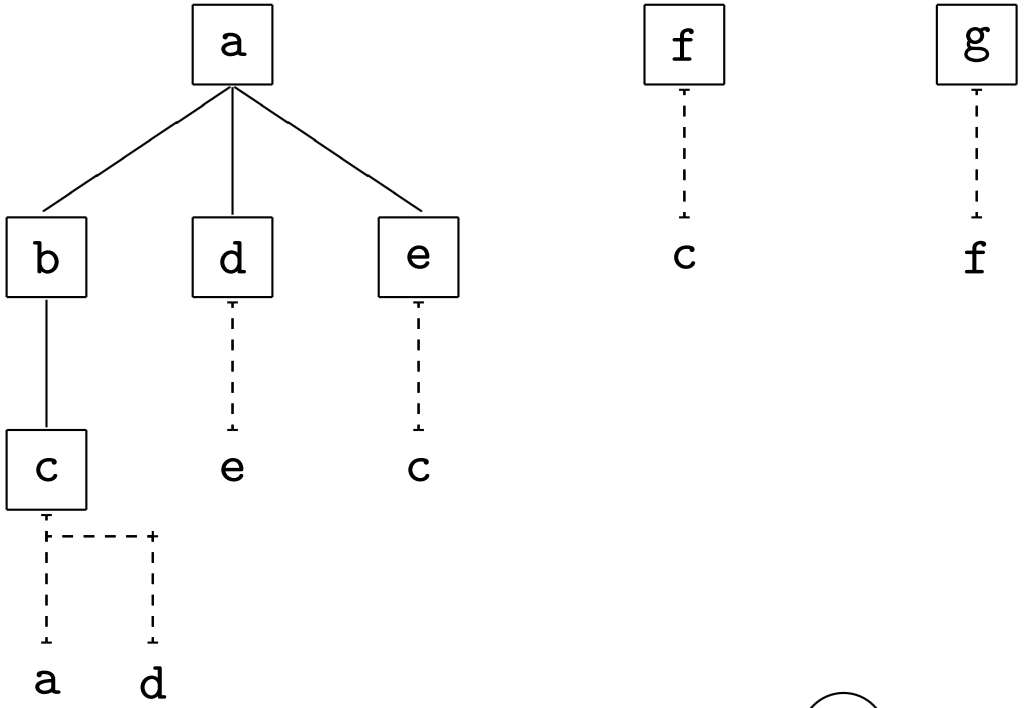
a e c d b f g



Árvore em Profundidade (iterativo)

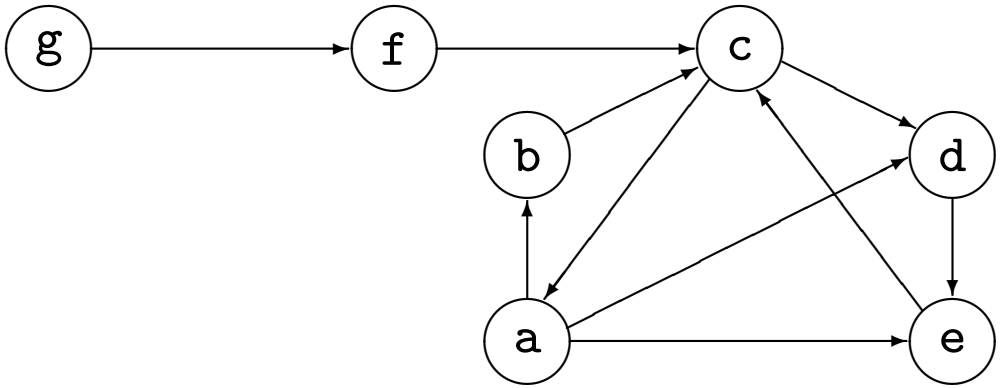
```
void dfsExplore( Graph graph,  boolean[] explored,  Vertex root )
{
    Stack<Vertex> foundUnexplored = new StackIn...<Vertex>(?);
    foundUnexplored.push(root);
    do {
            Vertex vertex = foundUnexplored.pop();
            if ( !explored[vertex] )
            {   TREAT(vertex);    explored[vertex] = true;
                for every Vertex w in graph.outAdjacentVertices(vertex)
                    if ( !explored[w] )
                        foundUnexplored.push(w);
            }
        }
    while ( !foundUnexplored.isEmpty() );
}
```

Percurso em Largura



Ordem:

a b d e c f g



Percurso em Largura

```
void bfsTraversal( Graph graph )
{
    boolean[] found = new boolean[ graph.numVertices() ];

    for every Vertex v in graph.vertices()
        found[v] = false;

    for every Vertex v in graph.vertices()
        if ( !found[v] )
            bfsExplore(graph, found, v);
}
```

Árvore em Largura (iterativo)

```
void bfsExplore( Graph graph,  boolean[] found,  Vertex root )
{
    Queue<Vertex> waiting = new QueueIn...<Vertex>(?);
    waiting.enqueue(root);
    found[root] = true;
    do {
          Vertex vertex = waiting.dequeue();    TREAT(vertex);
          for every Vertex w in graph.outAdjacentVertices(vertex)
              if ( !found[w] )
              {   waiting.enqueue(w);
                  found[w] = true;
              }
       }
    while ( !waiting.isEmpty() );
}
```

Complexidade dos Percursos

Implementação do Grafo (V, A)	**Profundidade** (recursivo) ou **Profundidade** (iterativo) ou **Largura** (iterativo)
Matriz	$\Theta(\|V\|^2 + \|V\| \times \text{custo(TREAT)})$
Vetor de Listas	$\Theta(\|V\| + \|A\| + \|V\| \times \text{custo(TREAT)})$

foundUnexplored.size() $\leq |A|$ **waiting**.size() $\leq |V| - 1$