IPS Instituto Politécnico de Setúbal
Escola Superior de Tecnologia de Setúbal

Programação Avançada

Implementação do TAD –Graph
Programação Avançada – 2020-21

Bruno Silva, Patrícia Macedo

Sumário 

- Revisão dos algoritmos para percorrer Arvores
- Algoritmos para percorrer Grafos
  - Percorrer Grafos em Largura
  - Percorrer Grafos em Profundidade

Percorrer Grafos

Qual a finalidade de Percorrer um grafo?

- Procurar se um vértice, ou uma aresta existem.
- Cópia de um grafo ou conversão entre representações diferentes.
- Contagem de números de vértices e/ou arestas.
- Determinação de caminho entre dois vértices ou ciclos, caso existam.

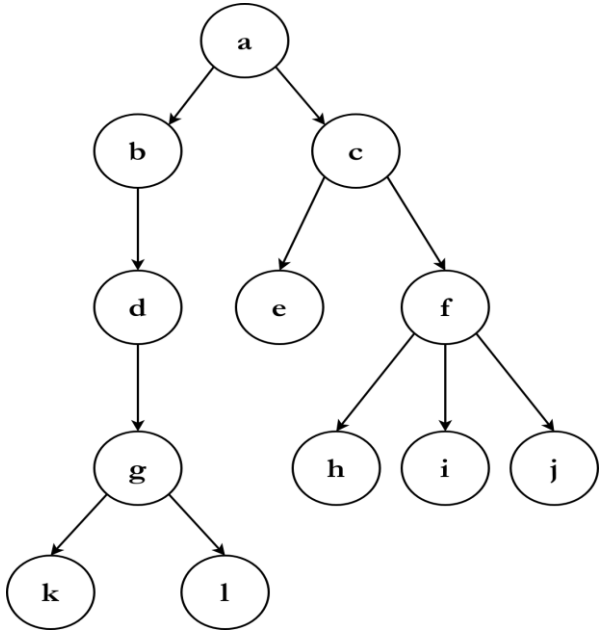
Percorrer Grafos: Algoritmos

- As estruturas lineares são percorridas normalmente sequencialmente de uma extremidade para a outra.
- As estruturas não lineares existem podem ser percorridas de diversas formas.
- As árvores pode ser percorridas em
  - largura
  - profundidade (pos-order e pre-order)
- Os grafos também podem ser percorridos em
  - largura
  - profundidade

Percorrer árvores algoritmos (revisão)

As árvores podem ser percorridas usando duas estratégias base

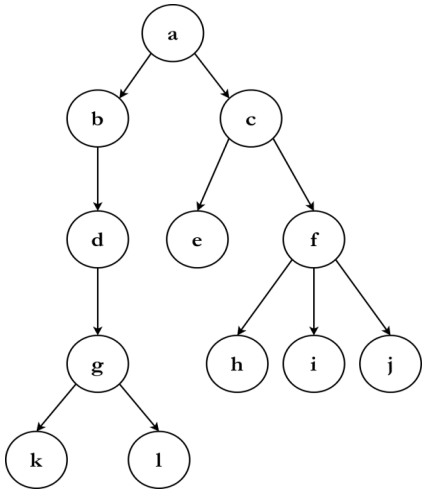
- Em Largura (breadth-first)
  - [esq/dir] **a b c d e f g h i j k l**
  - [dir/esq] **a c b f e d j i h g l k**
- Em Profundidade (depth-first)
  - Pre-Ordem - **a b d g k l c e f h i j**
  - Pós-Order - **k l g d b e h i j f c a**



Percorrer árvores algoritmos

Algoritmo breadth-first

```
BFS(arvore)
   Coloque a raiz da árvore na fila
   Enquanto a fila não está vazia faça:
      seja n o nó que retira da fila
      processe n
      para todo o f nó filho de n
                coloque f na fila
```

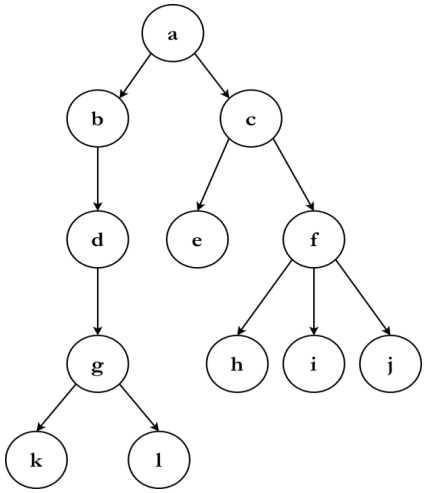


elementos	Fila
	a
a	b c
a b	c d
a b c	d e f
a b c d	e f g
a b c d e	f g
a b c d e f	g h i j
a b c d e f g	h i j k l
a b c d e f g h	i j k l
a b c d e f g h i	j k l
a b c d e f g h i j	k l
a b c d e f g h i j k	l
a b c d e f g h i j k l	

Percorrer árvores algoritmos

Algoritmo depth-first

```
BFS(arvore)
   Coloque a raiz da árvore na pilha
   Enquanto a pilha não está vazia faça:
      seja n o nó que retira da pilha
      processe n
      para todo o f nó filho de n
                coloque f na pilha
```



elementos	Pilha
	a
a	c b
a b	c d
a b d	c g
a b d g	c l k
a b d g k	c l
a b d g k l	c
a b d g k l c	f e
a b d g k l c e	f
a b d g k l c e f	j i h
a b d g k l c e f h	j i
a b d g k l c e f h i	j
a b d g k l c e f h i j	

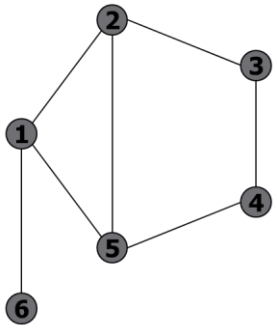
Percorrer Grafos :Algoritmos

Nos algoritmos de percorrer um grafo, devemos ter em conta:

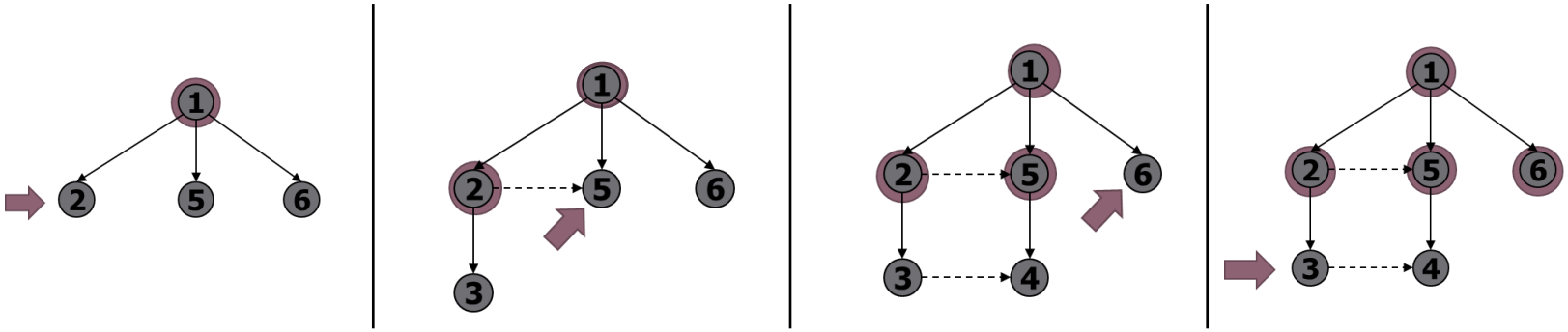
- Eficiência ⇒ um mesmo local não deve ser visitado repetidamente.
  - Marcam-se os vertices já visitados
- Corretude ⇒ o percurso deve ser feito de modo que não se perca nada.
  - Mantem-se uma coleção (pilha ou fila) com todos os vertices ainda por visitar.
    - Fila [FIFO] – Percorrer em Largura
    - Pilha [LIFO] – Percorrer em Profundidade

Pesquisa em Largura
Breadth-First Search - BFS

Inicia-se por um vértice (denominado vértice raiz) e exploram-se todos os vértices adjacentes a esse. Então, para cada um dos vértices adjacente, explora-se os seus vértices vizinhos ainda não visitados e assim por diante, até já não existirem mais vértices por visitar.



Grafo a percorrer em Largura a partir do vértice 1 – [1,2,5,6,3,4]



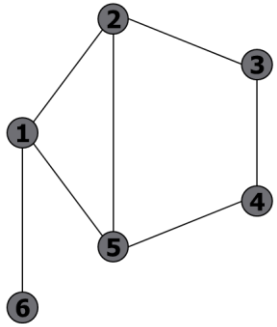
Algoritmo de Pesquisa em Largura

Algoritmo

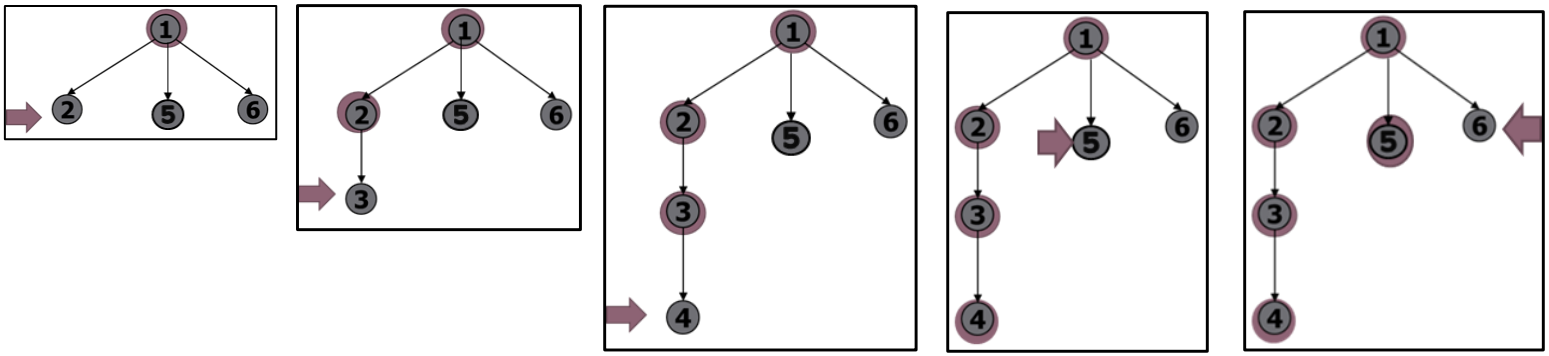
```
BFS(Graph,vértice_raiz)
    Marque vértice_raiz como visitado
    Coloque o vértice na fila
    Enquanto a fila não está vazia faça
    • seja  v  o vértice que retira da fila
    • para cada vértice  w  adjacente a v faça
        • se  w  não está marcado como visitado
        • então
            • marque  w como visitado
            • insira  w  na fila
```

Pesquisa em Profundidade
Depth-First Search - DFS

Inicia-se por um vértice (vértice_raiz) e explora-se tanto quanto possível cada um dos seus ramos, antes de retroceder.



Grafo a percorrer em Profundidade a partir do vértice 1 – [1,2,3,4,5,6]



Algoritmo de Pesquisa em Largura

Algoritmo genérico

```
DFS(Graph,vértice_raiz)
    Marque vértice como visitado
    Coloque o vértice na pilha
    Enquanto a pilha não está vazia faça
    • seja  v  o vértice que retira da pilha
      para cada vértice  w  adjacente a v faça
       • se  w  não está marcado como visitado
       • então
           • marque  w como visitado
           • insira  w  na pilha
```

Implementação dos Algoritmos

- Existem 2 abordagens para implementação:
  - Interna à classe – o método é implementado como método da classe que implementa a interface Graph.
  - Externa a classe – o método é implementado numa classe externa à classe que implementa Graph.

Interna
- Utiliza-se um conjunto para guardar os vértices já visitados, ou coloca-se um atributo visitado no nó do vértice.
- Implementa-se o método como método da classe Graph.

Externa
- Utiliza-se um conjunto para guardar os vértices já visitados.
- Implementa-se o método como método externo à classe.

ADT Graph | Exercícios de implementação

Continuando com o código iniciado na aula anterior sobre implementação do ADT Grafo.

1.

a) Acrescente na interface Graph o método

```
Iterable<Vertex<V>> DFS(<Vertex<V>>  v) throws InvalidVertexException;
```

b) Faça a Implementação do método na classe GraphEdgeList.

c) Construa um teste JUnit para testar o método DFS.
Sugestão: use os dados do exemplo do slide 11.

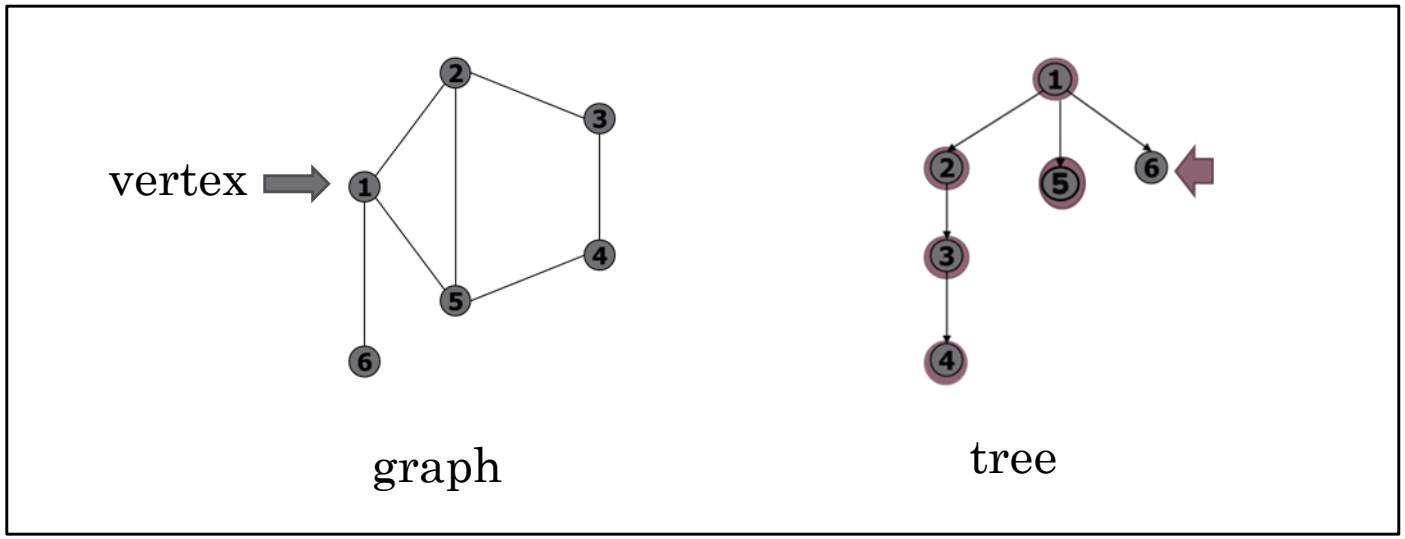
2. Repita o exercício 1 para o algoritmo BFS.

ADT Graph | Exercícios de implementação

3. Utilize o ADT Tree disponibilizado, nas aulas integrando-o no projeto em que está a trabalhar.
   Crie a classe GraphUtils<V,E>, e implemente o seguinte método estático.

   `Tree<V> treeDFS(Graph<V,E> graph , Vertex<V> vertex)`

   que devolve a árvore resultante de percorrer o grafo em profundidade

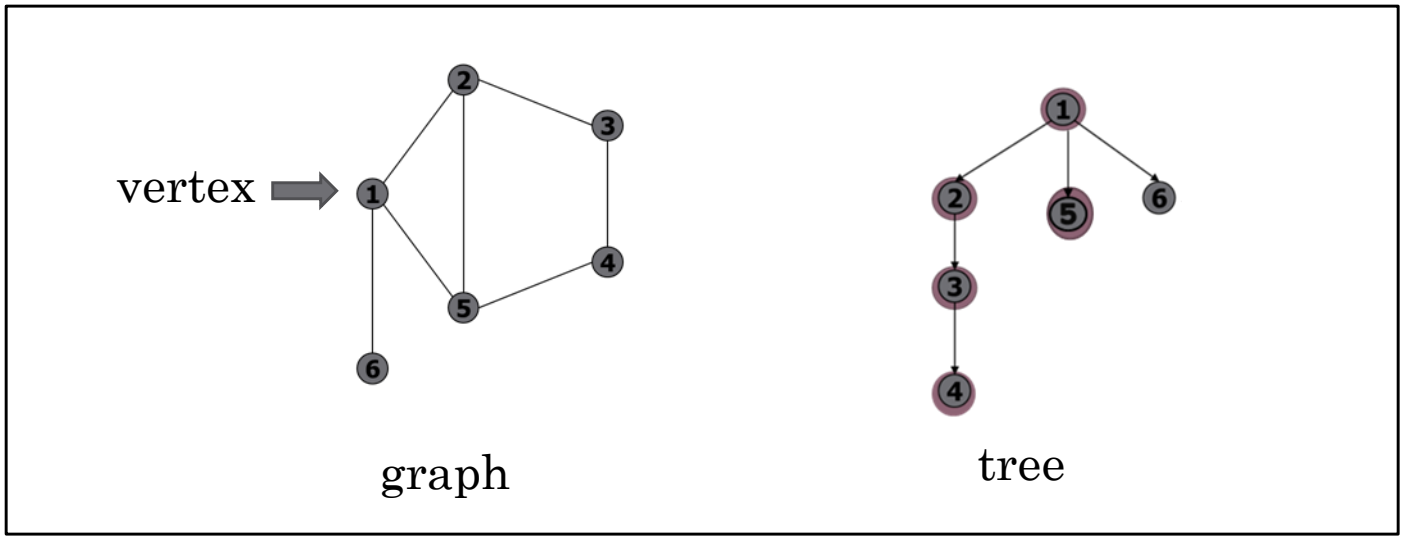


ADT Graph | Exercícios de implementação

3. Utilize o ADT Tree disponibilizado, nas aulas integrando-o no projeto em que está a trabalhar.
   Crie a classe GraphUtils, e implemente o seguinte método estático.

```
public static <V> Tree<V> treeDFS(Graph<V,?> graph , Vertex<V> vertex)
```

que devolve a árvore resultante de percorrer o grafo em profundidade



ADT Graph | Exercício Solução

```
public class GraphUtils {

    public static <V> Tree<V>  treeDFS(Graph<V,?> graph , Vertex<V> vertex)throws InvalidVertexException{
        Vertex<V> v,w;

        TreeLinked<V> tree = new TreeLinked(vertex.element());
        Stack<Vertex<V>> stack = new Stack();
        HashMap<V,Position<V>> treeMap= new HashMap<>();
        Set<Vertex<V>> visited= new HashSet();

        stack.push(vertex);
        visited.add(vertex);
        treeMap.put(vertex.element(), tree.root());

        while(!stack.empty()){
            v=stack.pop();
            Position<V> parent = treeMap.get(v.element());

            for(Edge edge: graph.incidentEdges(v)){
                w=graph.opposite(v,edge);
                if(!visited.contains(w))
                {
                    visited.add(w);
                    stack.push(w);
                    Position<V> pos = tree.insert(parent, w.element());
                    treeMap.put(w.element(),pos);
                }
            }
        }
        return tree;
    }
}
```

- Instanciar estruturas de suporte.
- O Map servirá para fazer uma ligação entre os elementos do vértices do grafo e os nós colocados na arvore.

- Atualizar vértices visitados Atualizar pilha
- Construir árvore: inserir nó.
- Atualizar map

Estudar e Explorar

- Percorrer Grafos: Em profundidade, em largura
- Visualizar:
  https://www.cs.usfca.edu/~galles/visualization/DFS.html

  https://www.cs.usfca.edu/~galles/visualization/BFS.html

- Ler mais em:

P 87)

P)