

Busca em Grafos

Paulo Vinícius Moreira Dutra¹

¹IF Sudeste MG – Campus Muriaé

Sistemas Inteligentes, 2022

Sumário

- **Parte I**
 - Introdução
 - Busca em largura
 - Busca em profundidade

Introdução

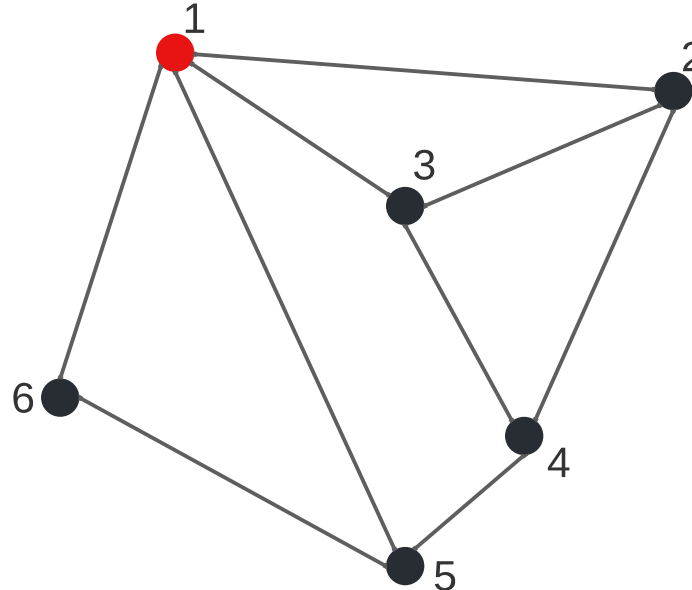
- A busca em grafo(**G**) consiste em explorar o grafo de forma sistemática, visitando as arestas e os vértices.
- O algoritmo em grafos deve ser o flexível suficiente para atender a diversidade de grafos.
- O algoritmo deve ser eficiente. Não deve haver repetições desnecessárias de visitas a um vértice e/ou aresta.
- Todos os vértices ou arestas devem ser visitados.

Introdução

- O algoritmo básico de busca em grafos:
 - Marcar os vértices
 - Não visitados;
 - Visitados;
 - Processados.

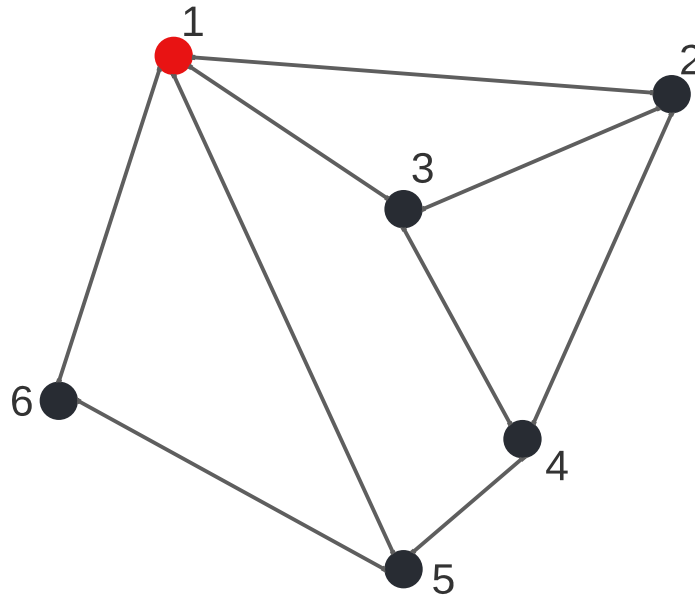
Introdução

- De modo geral, as operações de buscas dependem de um vértice inicial. Por exemplo, busca pelo menor caminho, temos que saber qual é o ponto de partida.



Introdução

- Durante a busca, o algoritmo pode precisar visitar todos ou apenas um conjunto dos vértices.



Introdução

- Há duas principais possibilidades de buscas:
 - Busca em profundidade (Depth-First Search - DFS)
 - Busca em largura (Breadth-First Search - BFS)

Busca em profundidade (DFS)

- A partir de um vértice inicial, a busca explora cada vizinho até o último vértice possível.

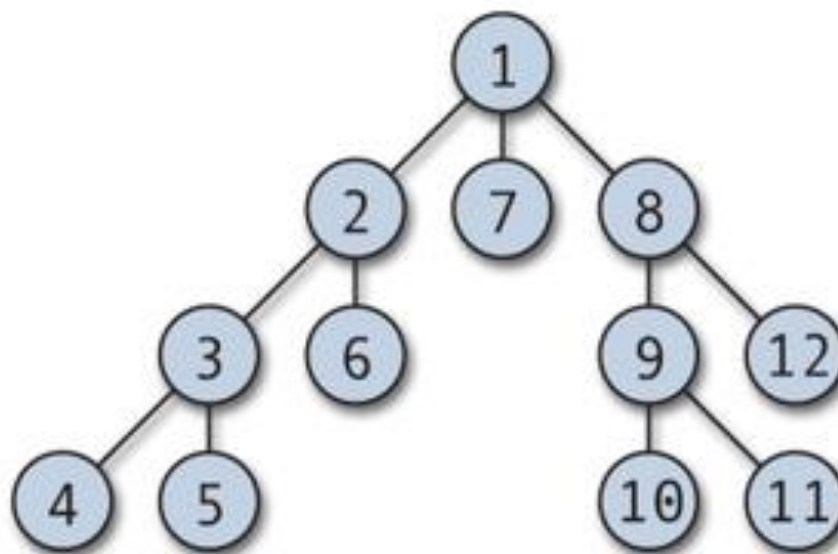


Figura: Representação gráfica da busca em profundidade

Busca em profundidade (DFS)

- Backtracking:
 - O grafo é percorrido até que a busca falhe, ou se encontrar um vértice sem vizinhos.
 - No backtracking a busca retorna pelo mesmo caminho para encontrar um caminho alternativo.

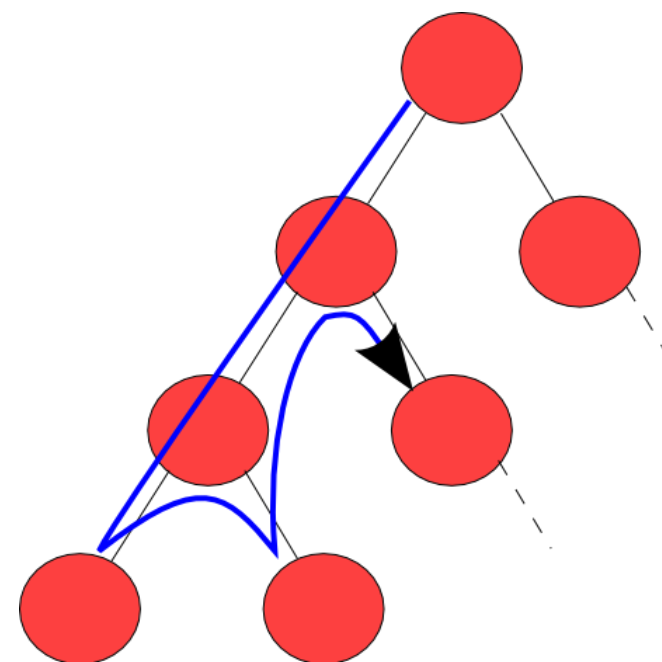


Figura: Representação gráfica da busca em profundidade. Backtracking

Busca em profundidade (DFS)

- Algoritmo – Utilizando Recursividade

Dados: $G(V,E)$, conexo

Procedimento $P(v,u)$

marcar v

para $w \in A(v)$ **efetuar**

se w é não marcado **então**

visitar aresta de árvore (v,w)

> arestas visitadas (I)

$P(w,v)$

caso contrário

se $w \neq u$ **então**

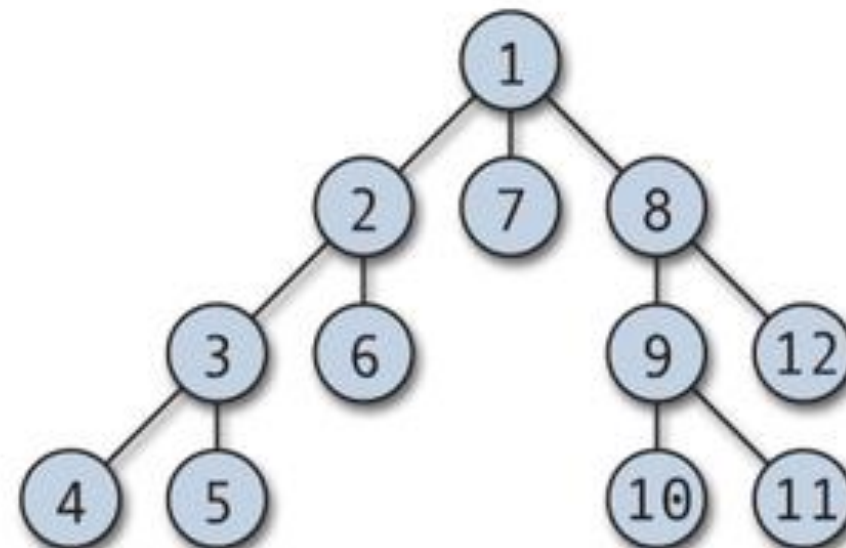
visitar aresta de retorno (v,w)

> arestas visitadas (II)

desmarcar todos os vértices

escolher uma raiz s

$P(s, \emptyset)$



Busca em profundidade (DFS)

- Algoritmo – Utilizando Pilha

Dados: $G(V,E)$, conexo

Procedimento $P(v)$

marcar v

colocar v na pilha Q

para $w \in A(v)$ **efetuar**

se w é não marcado **então**

visitar (v,w) > arestas visitadas (I)

$P(w)$

caso contrário

se $w \in Q$ e v,w não são consecutivos em Q **então**

visitar (v,w) > arestas visitadas (II)

retirar v de Q

desmarcar todos os vértices

definir uma pilha Q

escolher uma raiz s

$P(s)$

Busca em profundidade (DFS)

- Busca em largura em matriz de adjacência para um grafo não direcionado:

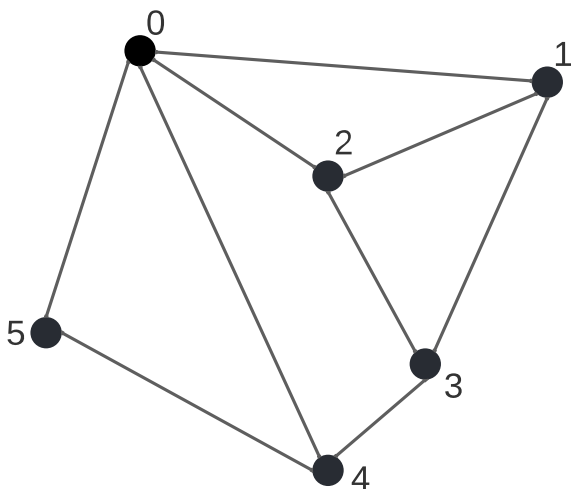


Figura: Grafo não-direcionado

	0	1	2	3	4	5
0	0	1	1	0	1	1
1	1	0	1	0	0	0
2	1	1	0	1	0	0
3	0	0	1	0	1	0
4	1	0	0	1	0	1
5	1	0	0	0	1	0

$$V = \{0, 1, 2, 3, 4, 5\}$$

$$E \{(0,1), (0,2), (0,4), (0,5), (1,0), (1,2), (2,0), (2,1), (2,3), (3,2), (3,4), (4,0), (4,3), (4,5), (5,0), (5,4)\}$$

Busca em profundidade (DFS) - Implementação

- Visitando todos os vértices do grafo:

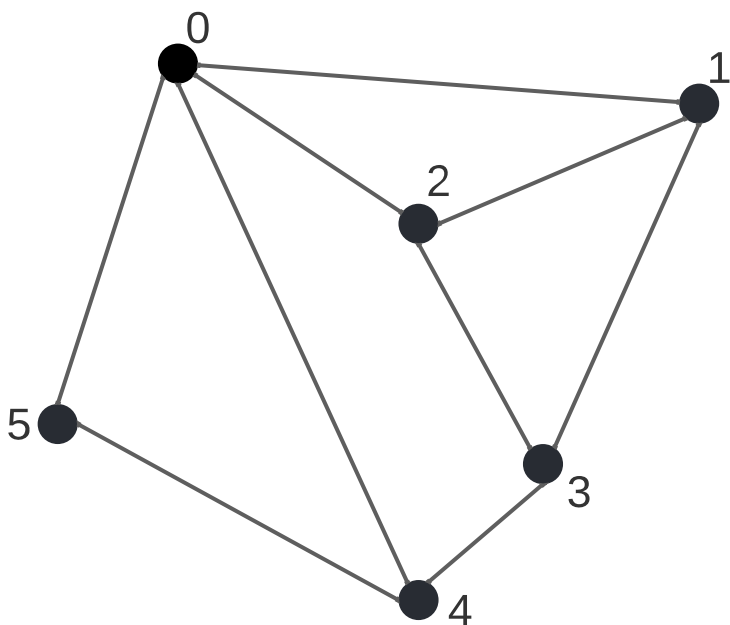


Figura: Grafo conexo e não-direcionado

```
#Grafo com 6 vértices
grafo = Grafo(6)
grafo.inserir_aresta(0, 1)
grafo.inserir_aresta(0, 2)
grafo.inserir_aresta(0, 4)
grafo.inserir_aresta(0, 5)
grafo.inserir_aresta(1, 2)
grafo.inserir_aresta(1, 3)
grafo.inserir_aresta(2, 3)
grafo.inserir_aresta(3, 4)
grafo.inserir_aresta(4, 5)
grafo.busca_profundidade_grafo(0)
```

Busca em profundidade (DFS) – Implementação em python

- Visitando todos os vértices do grafo:

```
def busca_profundidade_grafo(self, v_inicial):  
    visitados = np.zeros(self.n, dtype="int")  
    print(visitados)  
    self.busca_profundidade(v_inicial, visitados)  
    print(visitados)  
  
def busca_profundidade(self, v_inicial, visitados):  
    visitados[v_inicial] = -1 #Marca o vértice como visitado  
    for i in range(self.n):  
        v = self.matriz_adjacencia[v_inicial][i]  
        if visitados[i] == 0:  
            if v == 1: #Verifica se o vértice é vizinho  
                print("v({})->w({})".format(v_inicial, i))  
                self.busca_profundidade(i, visitados)
```

Busca em profundidade (DFS) – Implementação em python

- Primeiro devemos marcar os vértices como não visitados. Para isso, criamos um vetor com o mesmo tamanho de vértices do grafo.
- É iniciamos a busca a partir de um vértice inicial.

Marca os
vértices
como NÃO
visitados

```
def busca_profundidade_grafo(self, v_inicial):  
    visitados = np.zeros(self.n, dtype="int")  
    print(visitados)  
    self.busca_profundidade(v_inicial, visitados)  
    print(visitados)
```

Busca em profundidade (DFS) – Implementação em python

- Marca o vértice como visitado e realiza uma busca nos vizinhos ainda não visitados.

Marca o
vértice atual
como
visitados

Percorre
cada vértice
vizinho

```
def busca_profundidade(self, v_inicial, visitados):  
    visitados[v_inicial] = -1 #Marca o vértice como visitado  
    for i in range(self.n):  
        v = self.matriz_adjacencia[v_inicial][i]  
        if visitados[i] == 0:  
            if v == 1: #Verifica se o vértice é vizinho  
                print("v({})->w({})".format(v_inicial, i))  
                self.busca_profundidade(i, visitados)
```


Busca em profundidade (DFS) – Aplicações

- Ordenação topológica de um grafo
- Procurar a saída de um labirinto
- Verificar se um grafo é completamente conexo
 - Por exemplo, podemos verificar se um rede esta funcionamento corretamente
- Implementar uma ferramenta de preenchimento
 - Por exemplo, o balde de pintura do Paint ou Photoshop

Referências

Szwarciter, Jaime Luiz. **Teoria Computacional de Grafos**. Elsevier, 2018.
ISBN 978-85-352-8884-1.