

# Travessia de Grafos

## *Depth-First Search*

---

Prof. Edson Alves

2019

Faculdade UnB Gama

1. Definição
2. Implementação

# Definição

---

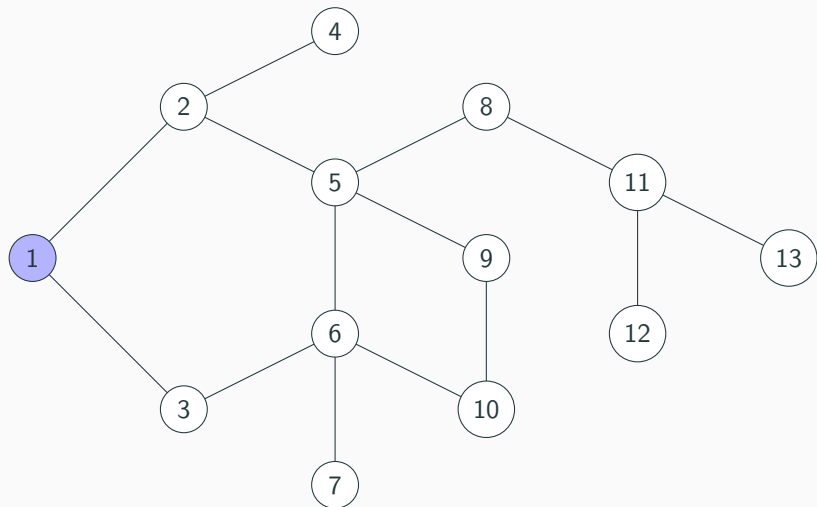
# Travessia de um grafo

- Uma travessia de um grafo consiste em visitar todos os nós alcançáveis a partir de um nó inicial  $s$
- Cada nó deve ser processado uma única vez, embora a travessia possa passar por um nó mais de uma vez
- Uma travessia  $T_1$  é diferente de uma travessia  $T_2$  se ambas diferem na ordem de visitação dos vértices
- Um grafo conectado com  $N$  nós tem  $N!$  travessias possíveis
- Dentre todas estas travessias, duas se destacam pela aplicabilidade em situações práticas: a travessia por profundidade e a travessia por largura

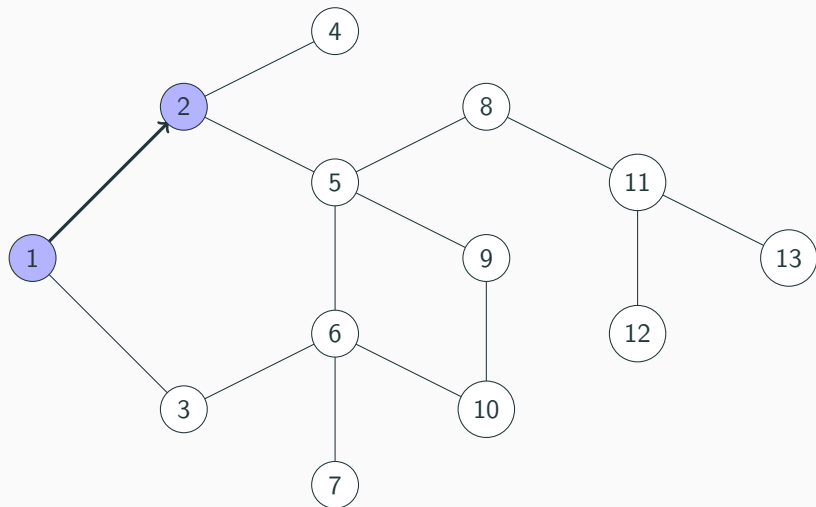
# Depth-First Search

- A travessia por profundidade (*Depth-First Search* – *DFS*) segue, a partir do nó inicial  $s$ , um caminho único, enquanto encontrar novos nós
- Quando não for possível encontrar novos nós, a DFS retorna ao nó anterior e retoma o caminho usando o próximo nó encontrado
- A DFS mantém um registro dos nós visitados, de forma que cada nó seja processado uma única vez
- Em um grafo conectado com  $N$  nós e  $M$  arestas, a complexidade da DFS é  $O(N + M)$ , pois cada nó e cada aresta são processados uma única vez
- Se o grafo for representado como uma matriz de adjacências, a complexidade é  $O(N^2)$

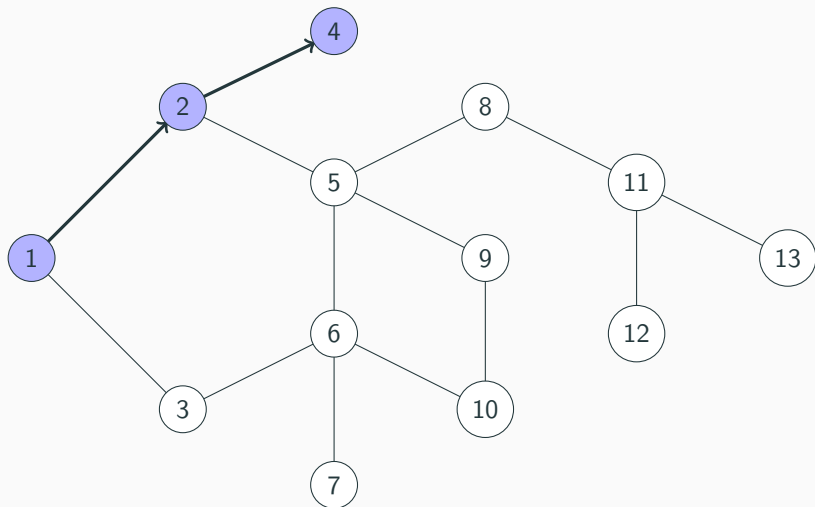
## Visualização da DFS



## Visualização da DFS

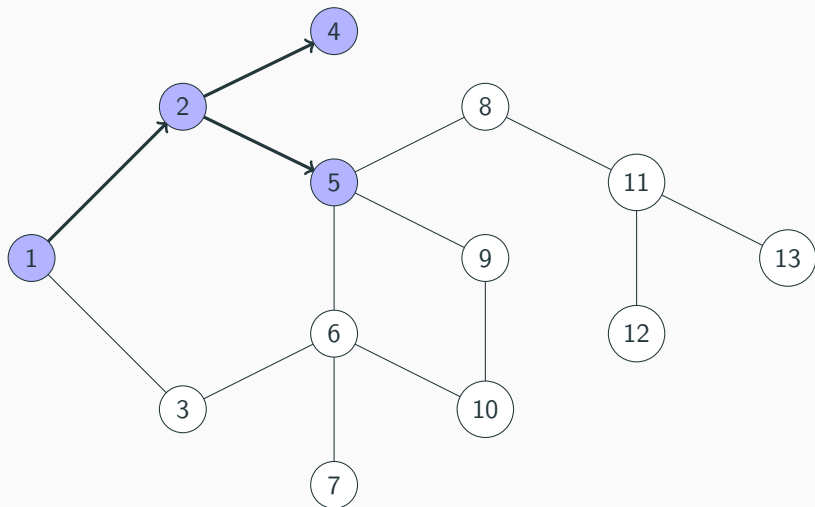


## Visualização da DFS

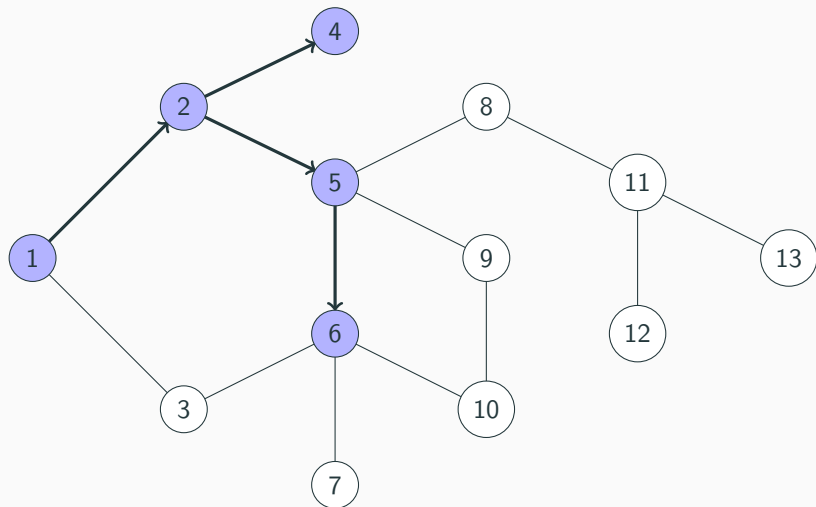




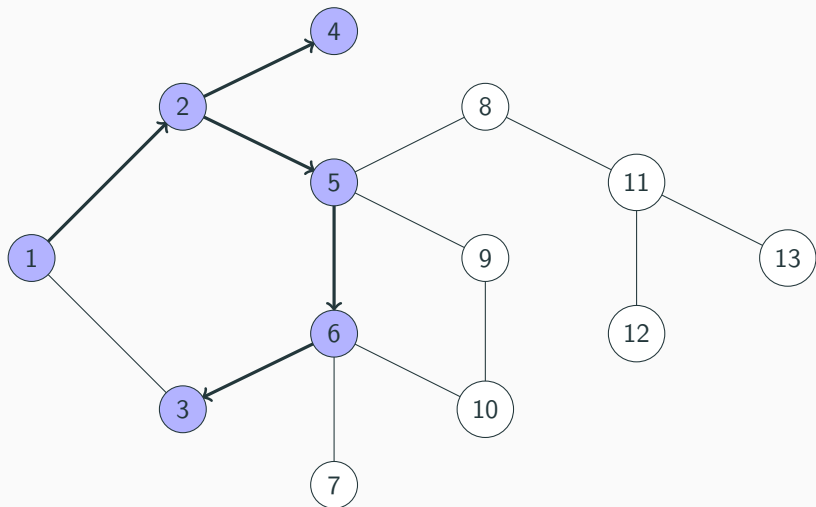
## Visualização da DFS



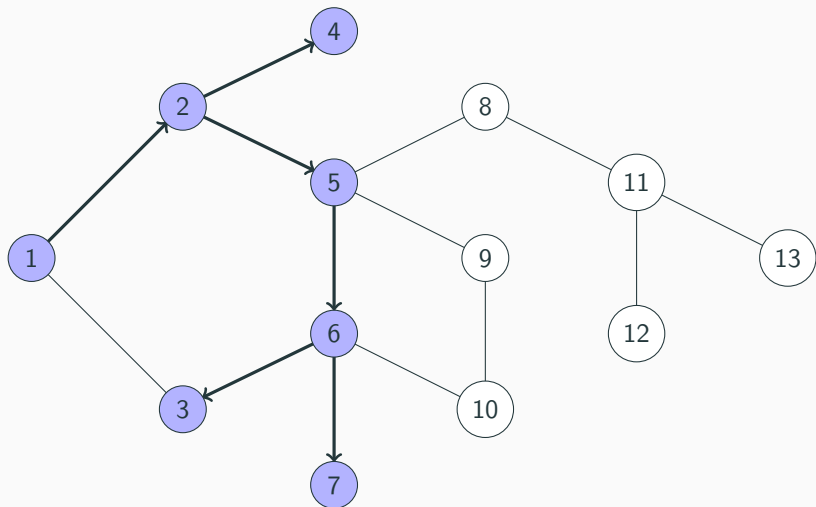
## Visualização da DFS



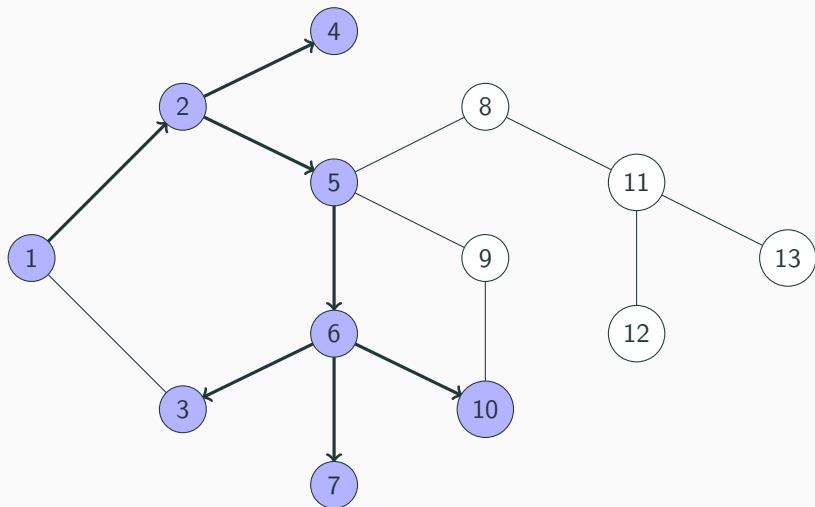
## Visualização da DFS



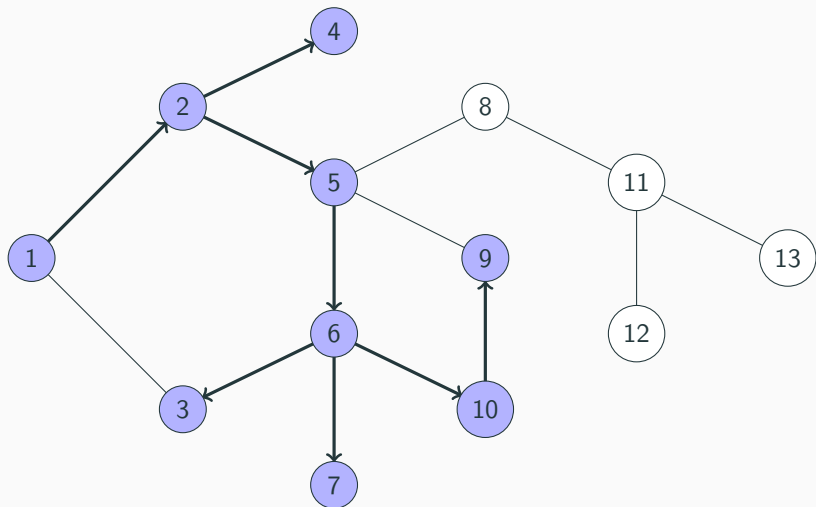
## Visualização da DFS



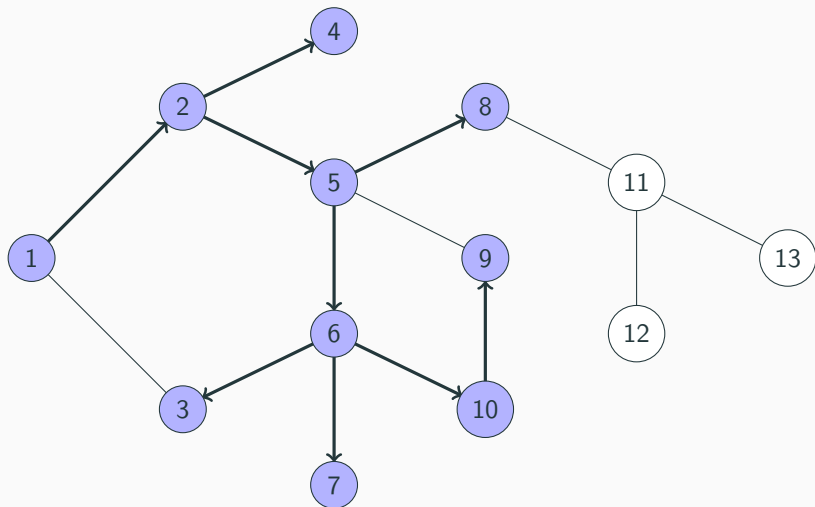
## Visualização da DFS



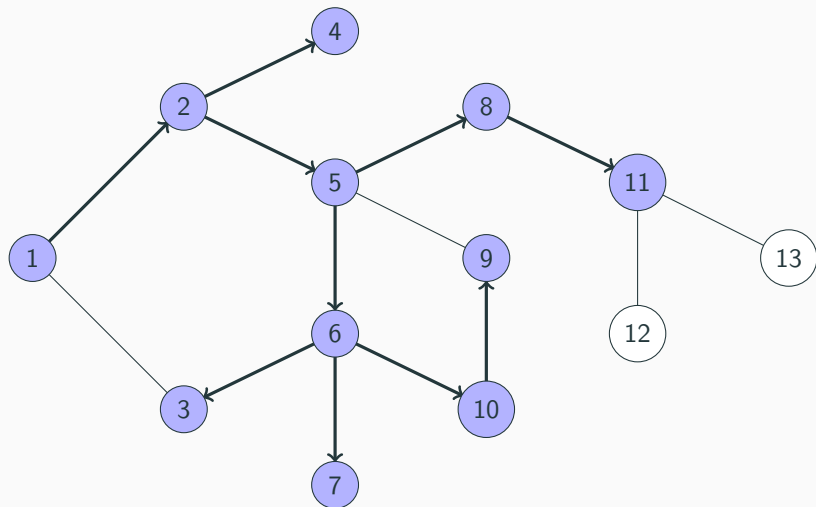
## Visualização da DFS



## Visualização da DFS

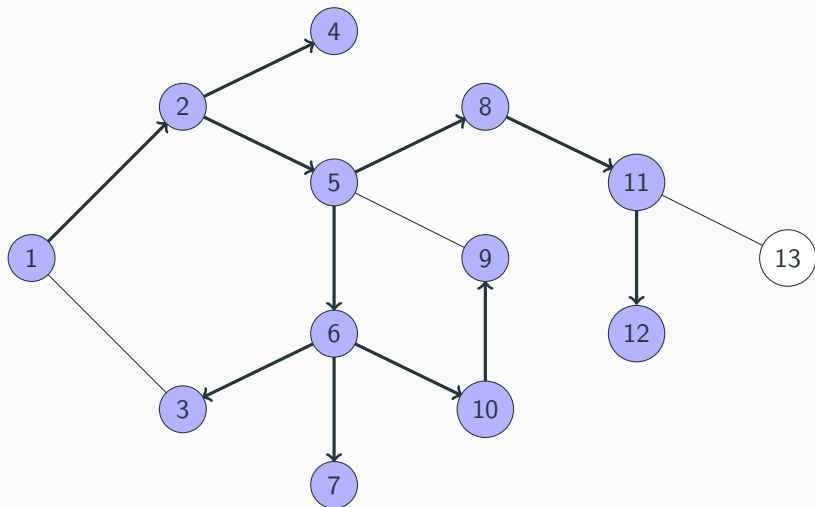


# Visualização da DFS

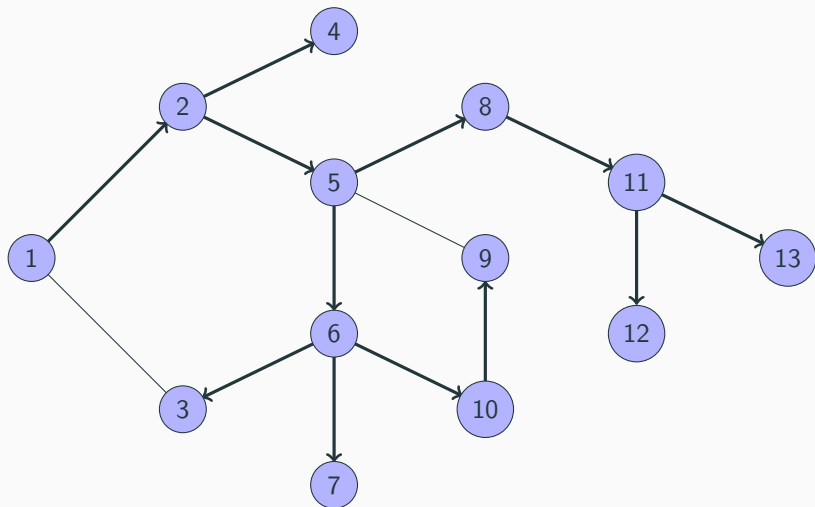




## Visualização da DFS



# Visualização da DFS



# Implementação

---

# Implementação da DFS

- A DFS é implementada naturalmente por meio de recursão
- Deve se manter um registro dos vértices já visitados
- O caso base da recursão acontece quando a busca atinge um vértice já visitado
- Caso o vértice ainda não tenha sido visitado, ele deve ser marcado, processado, e a busca deve prosseguir em todos os seus vizinhos ainda não visitados
- Se o registro dos vértices visitados não for passado como parâmetro da função, ele deve ser reiniciado a cada busca
- O processamento do vértice pode ser parametrizado, de modo que a rotina de travessia possa ser reutilizada com vários processamentos distintos

# Implementação da DFS em C++

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4 using edge = pair<int, int>;
5
6 const int MAX { 100010 };
7 bitset<MAX> visited;
8 vector<int> adj[MAX];
9
10 void dfs(int u, const function<void(int)>& process)
11 {
12     if (visited[u]) return;
13
14     visited[u] = true;
15     process(u);
16
17     for (const auto& v : adj[u])
18         dfs(v, process);
19 }
20
```

# Implementação da DFS em C++

```
21 int main()
22 {
23     vector<edge> edges { { 1, 2 }, { 1, 3 }, { 2, 4 }, { 2, 5 }, { 3, 6 },
24         { 5, 6 }, { 5, 8 }, { 5, 9 }, { 6, 7 }, { 6, 10 }, { 8, 11 },
25         { 9, 10 }, { 11, 12 }, { 11, 13 } };
26
27     for (const auto& [u, v] : edges)
28     {
29         adj[u].push_back(v);
30         adj[v].push_back(u);
31     }
32
33     visited.reset();
34
35     dfs(1, [] (int u) { cout << u << " "; });
36
37     cout << '\n';
38
39     return 0;
40 }
```

# Implementação iterativa da DFS em C++

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4 using edge = pair<int, int>;
5
6 const int MAX { 100010 };
7 bitset<MAX> visited;
8 vector<int> adj[MAX];
9
10 void dfs(int u, const function<void(int)>& process)
11 {
12     stack<int> s;
13     s.push(u);
14
15     while (not s.empty())
16     {
17         auto v = s.top();
18     }
```

# Implementação iterativa da DFS em C++

```
19     if (not visited[v])
20     {
21         visited[v] = true;
22         process(v);
23     }
24
25     for (const auto& w : adj[v])
26         if (not visited[w])
27         {
28             s.push(w);
29             break;
30         }
31
32     if (s.top() == v)
33         s.pop();
34 }
35 }
36
```



# Implementação iterativa da DFS em C++

```
37 int main()
38 {
39     vector<edge> edges { { 1, 2 }, { 1, 3 }, { 2, 4 }, { 2, 5 }, { 3, 6 },
40         { 5, 6 }, { 5, 8 }, { 5, 9 }, { 6, 7 }, { 6, 10 }, { 8, 11 },
41         { 9, 10 }, { 11, 12 }, { 11, 13 } };
42
43     for (const auto& [u, v] : edges)
44     {
45         adj[u].push_back(v);
46         adj[v].push_back(u);
47     }
48
49     visited.reset();
50
51     dfs(1, [] (int u) { cout << u << " "; cout.flush(); });
52
53     cout << '\n';
54
55     return 0;
56 }
```

1. **HALIM**, Felix; **HALIM**, Steve. *Competitive Programming 3*, 2010.
2. **LAAKSONEN**, Antti. *Competitive Programmer's Handbook*, 2018.
3. **SKIENA**, Steven S.; **REVILLA**, Miguel A. *Programming Challenges*, 2003.
4. **FILIPEK**, Bartłomiej. *C++17 in Detail*, 2018<sup>1</sup>.

---

<sup>1</sup><https://leanpub.com/cpp17indetail>