Grafos

Travessia por profundidade

Prof. Edson Alves

Faculdade UnB Gama

Definição de travessia

Definição de travessia

Uma travessia consiste em visitar todos os nós atingíveis a partir de um vértice de partida s, em alguma ordem. Cada vértice deve ser visitado exatamente uma vez.

Características da travessias

Características da travessias

- **Duas travessias são distintas se as ordens de visitação são diferentes**

Características da travessias

- ⋆ **Duas travessias são distintas se as ordens de visitação são diferentes**
- ⋆ **Um grafo conectado com N vértices tem $N!$ travessias distintas**

Características da travessias

- ⋆ **Duas travessias são distintas se as ordens de visitação são diferentes**
- ⋆ **Um grafo conectado com N vértices tem $N!$ travessias distintas**
- ⋆ **Travessias notáveis: por profundidade e por extensão**

Travessia por profundidade *(Depth-first search)*

Travessia por profundidade *(Depth-first search)*

Seja s o vértice de partida e u o vértice observado no momento. As regras abaixo definem a DFS:

Travessia por profundidade *(Depth-first search)*

Seja s o vértice de partida e u o vértice observado no momento. As regras abaixo definem a DFS:

1. **Faça** $u = s$

Travessia por profundidade *(Depth-first search)*

Seja s o vértice de partida e u o vértice observado no momento. As regras abaixo definem a DFS:

1. **Faça** $u = s$

2. **Visite** u

Travessia por profundidade *(Depth-first search)*

Seja s o vértice de partida e u o vértice observado no momento. As regras abaixo definem a DFS:

1. **Faça** $u = s$

2. **Visite** u

3.1 **Se** u **tive ao menos um vizinho** v **ainda não visitado, faça** $u = v$

Travessia por profundidade *(Depth-first search)*

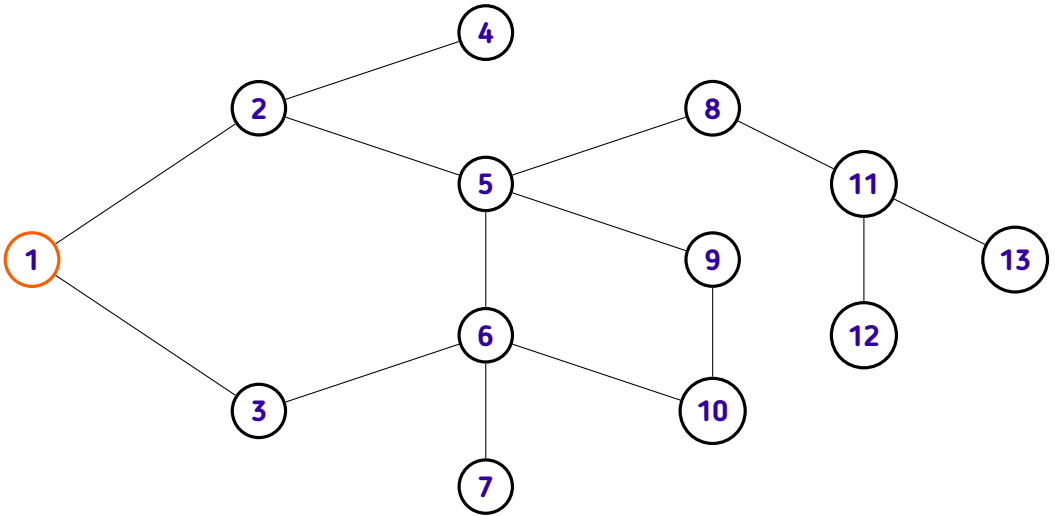
Seja s o vértice de partida e u o vértice observado no momento. As regras abaixo definem a DFS:

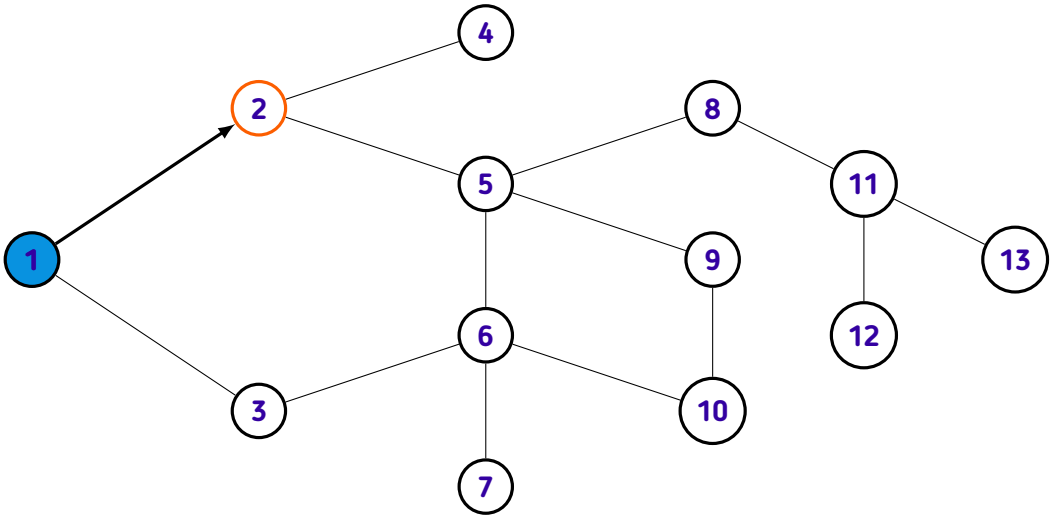
1. **Faça $u = s$**

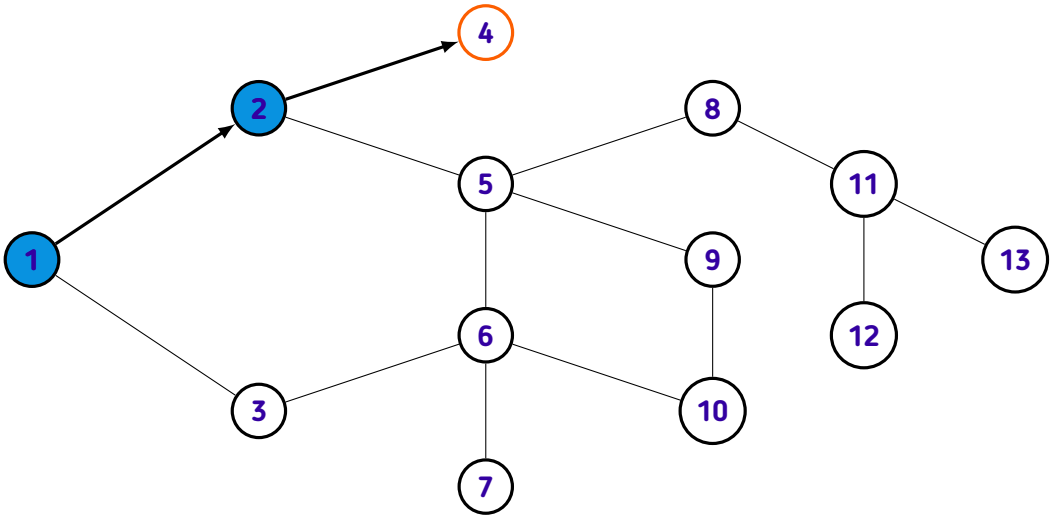
2. **Visite u**

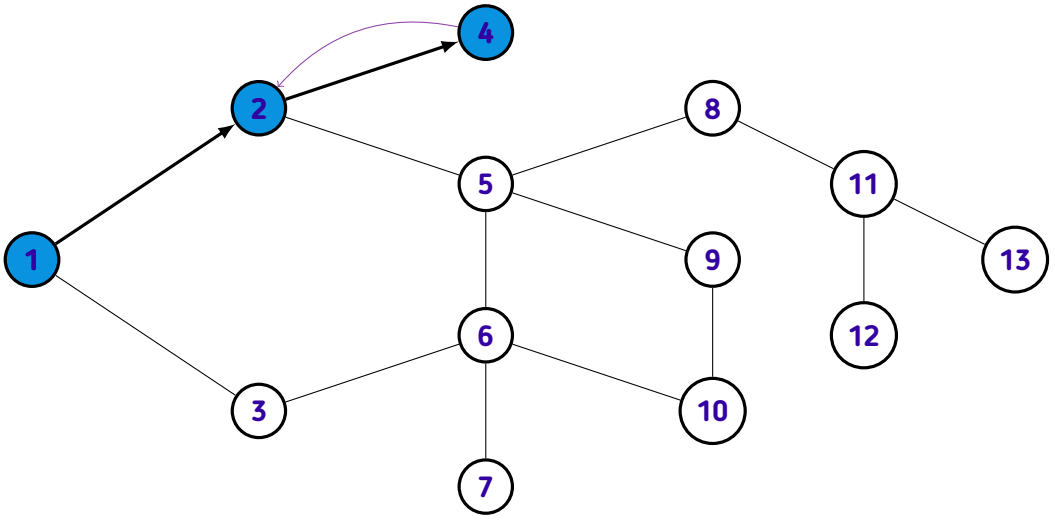
3.1 **Se u tive ao menos um vizinho v ainda não visitado, faça $u = v$**

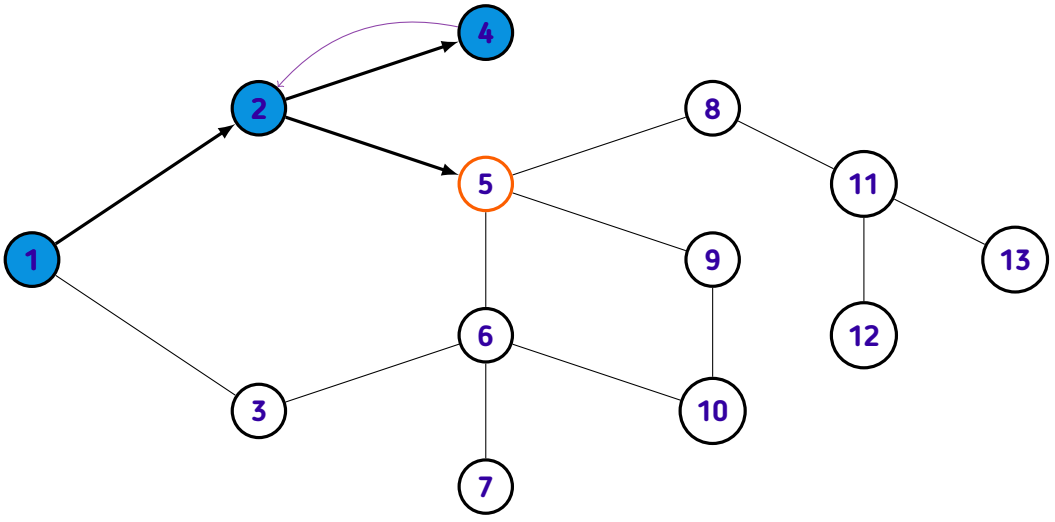
3.2 **Caso contrário, volte para o vértice que descobriu u**

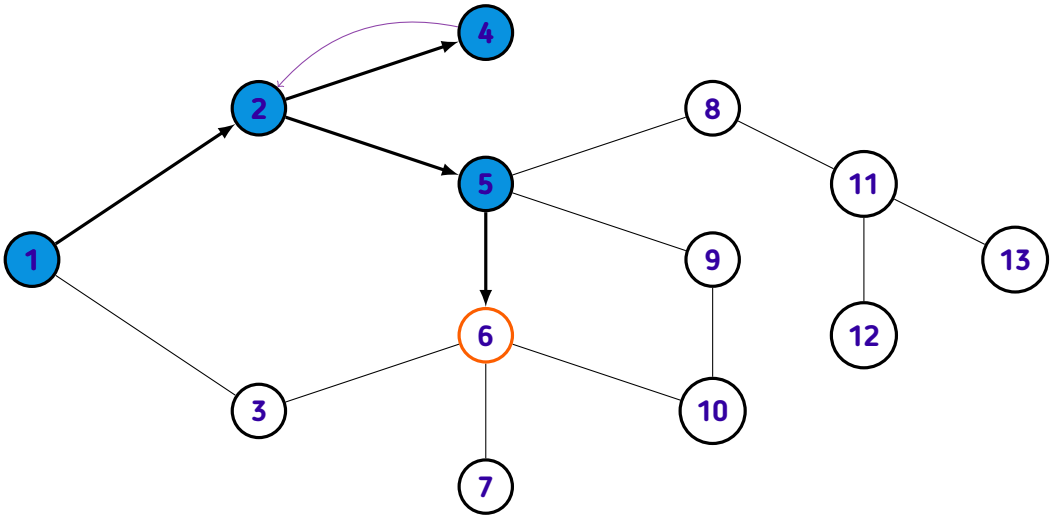

4
2
8
5
11
1
9
13
6
12
3
10
7

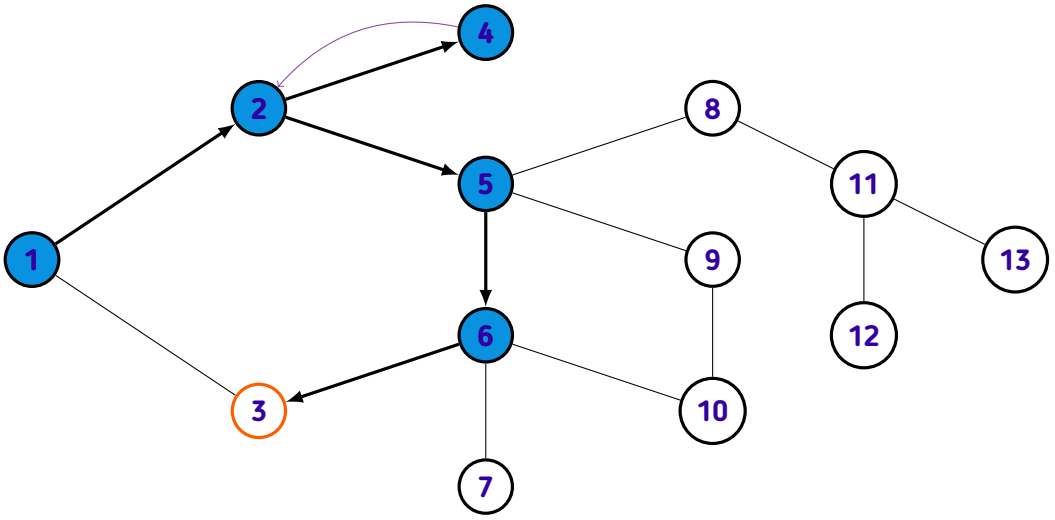

4
2
8
5
11
1
9
13
6
12
3
10
7

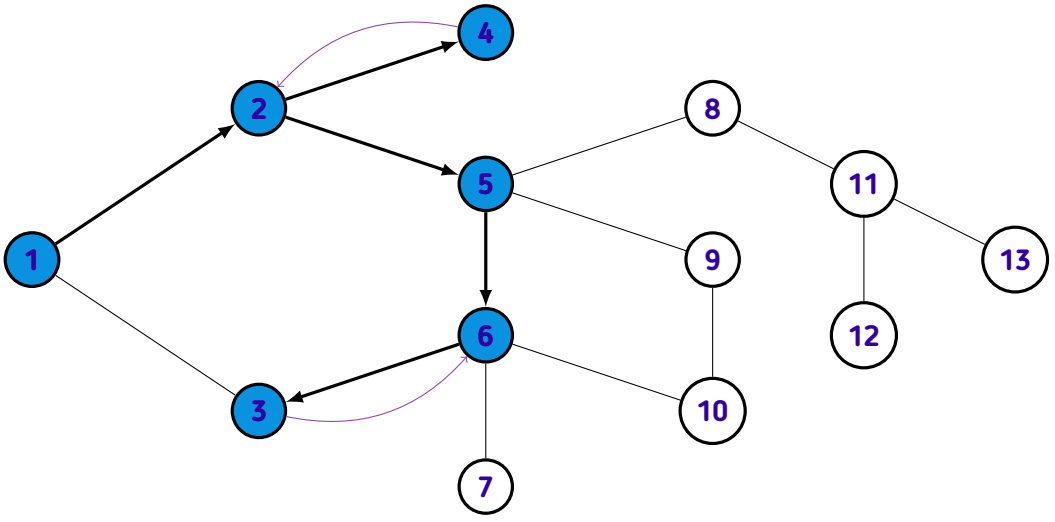

4
2
8
5
11
1
9
13
6
12
3
10
7

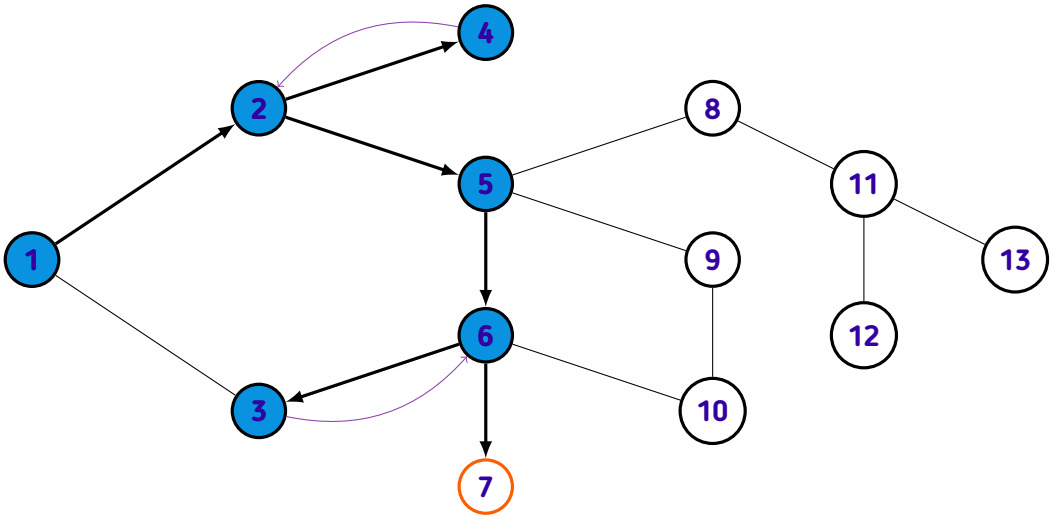

4
2
8
5
11
1
9
13
6
12
3
10
7

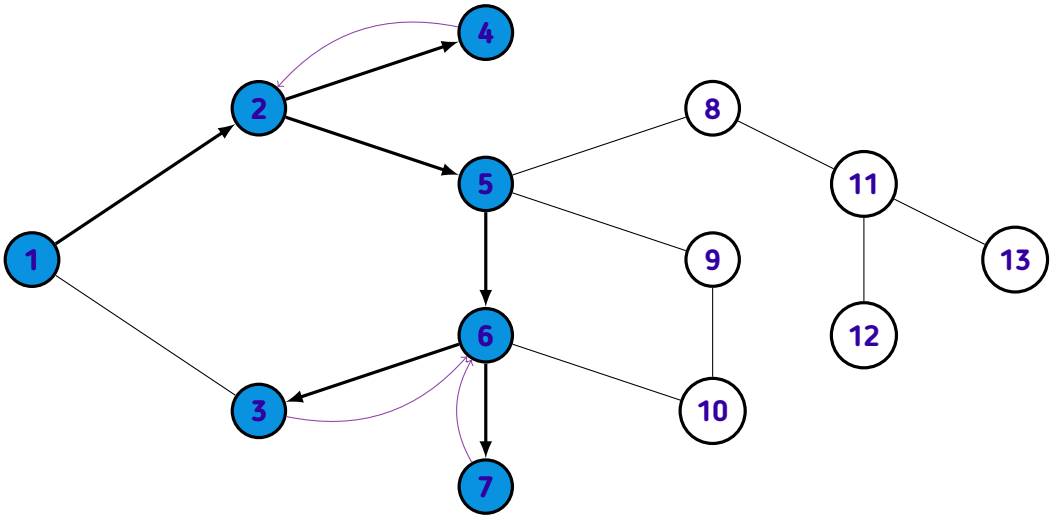

4
2
8
5
11
1
9
13
6
12
3
10
7

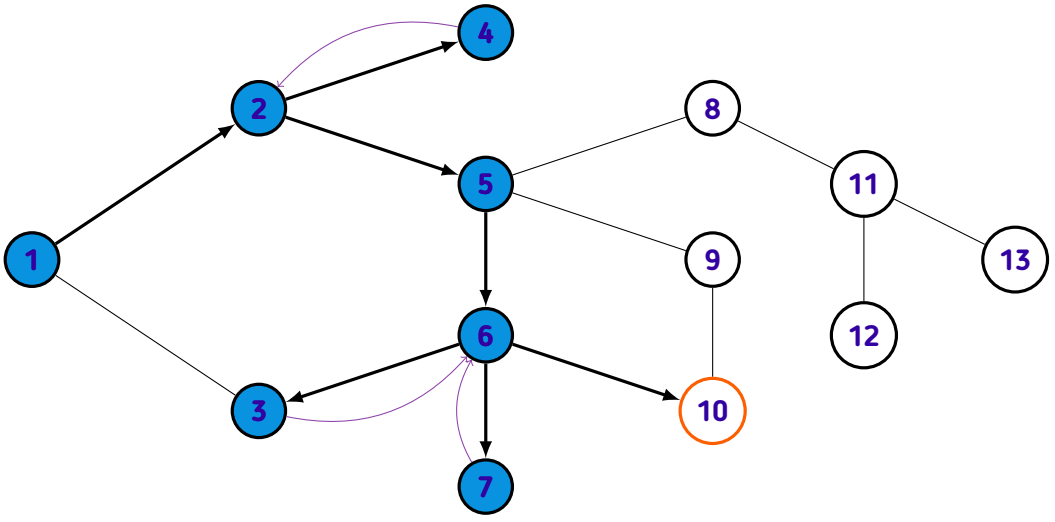

4
2
8
5
11
1
9
13
6
12
3
10
7

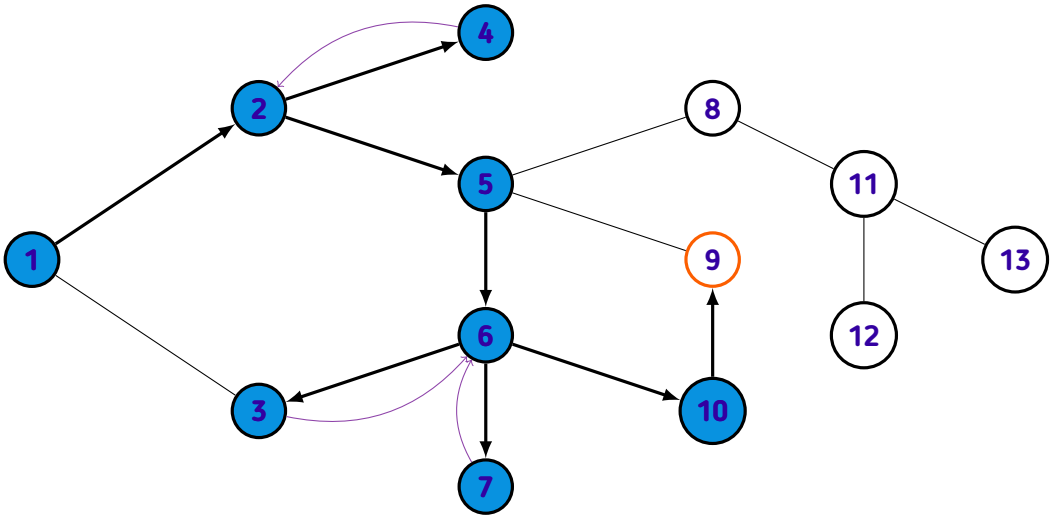

1
2
3
4
5
6
7
8
9
10
11
12
13

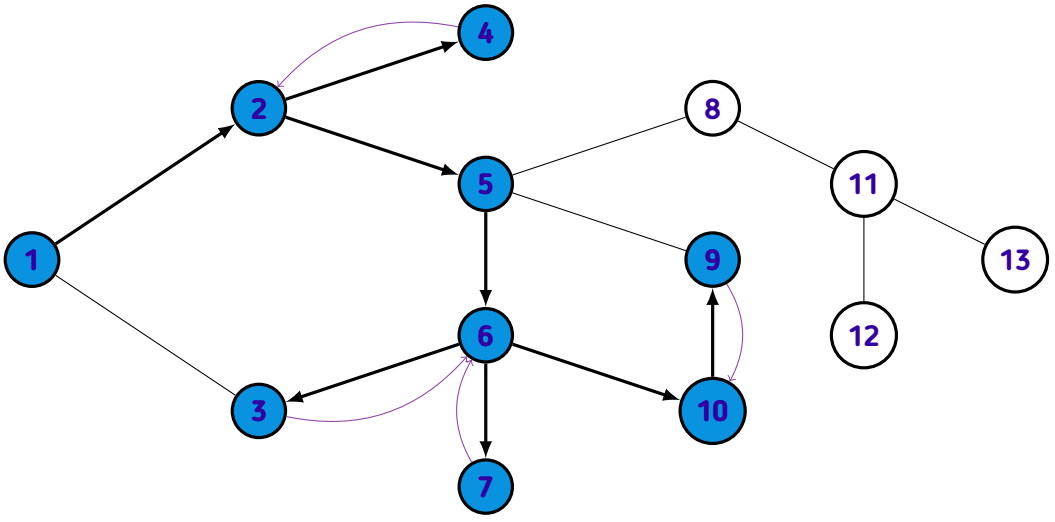

1
2
3
4
5
6
7
8
9
10
11
12
13

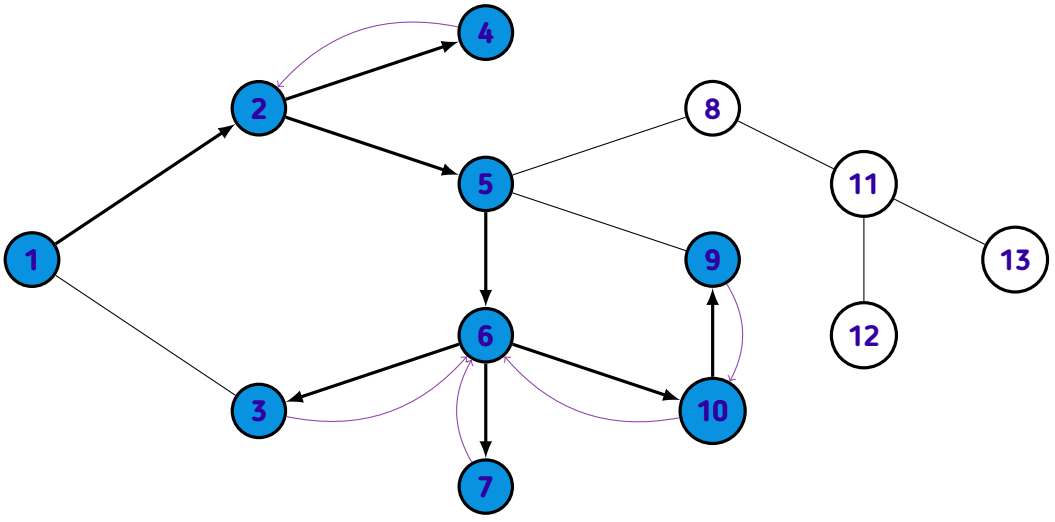

1
2
3
4
5
6
7
8
9
10
11
12
13

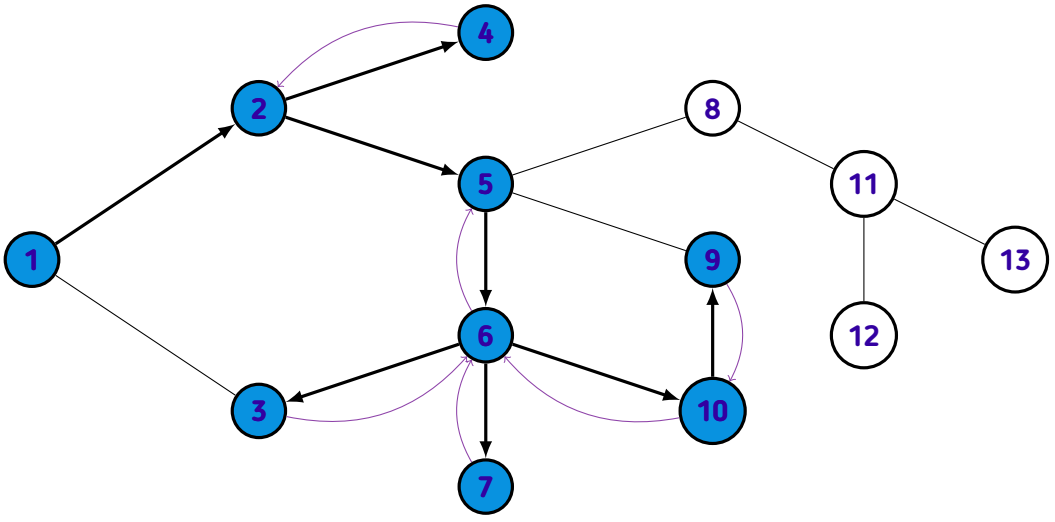

1
2
3
4
5
6
7
8
9
10
11
12
13

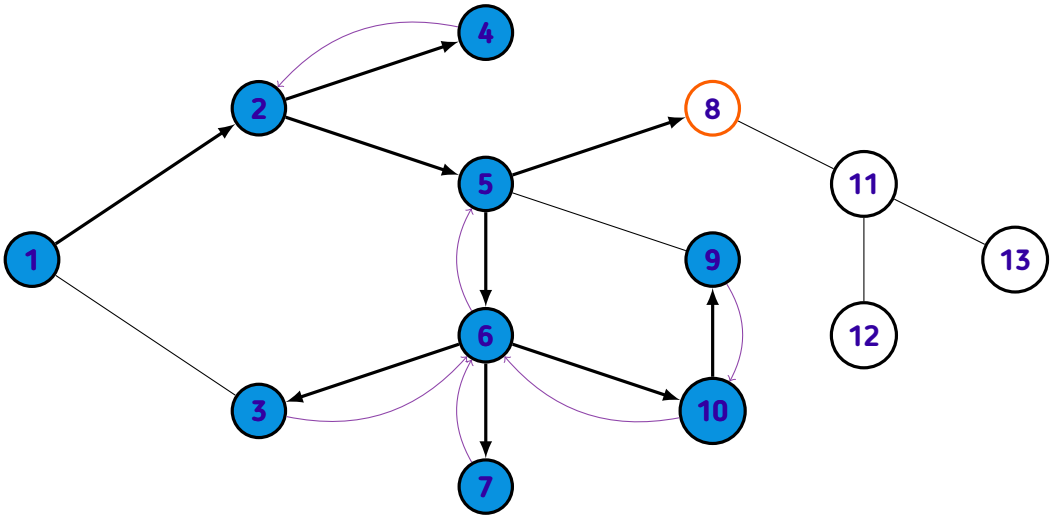

1
2
3
4
5
6
7
8
9
10
11
12
13

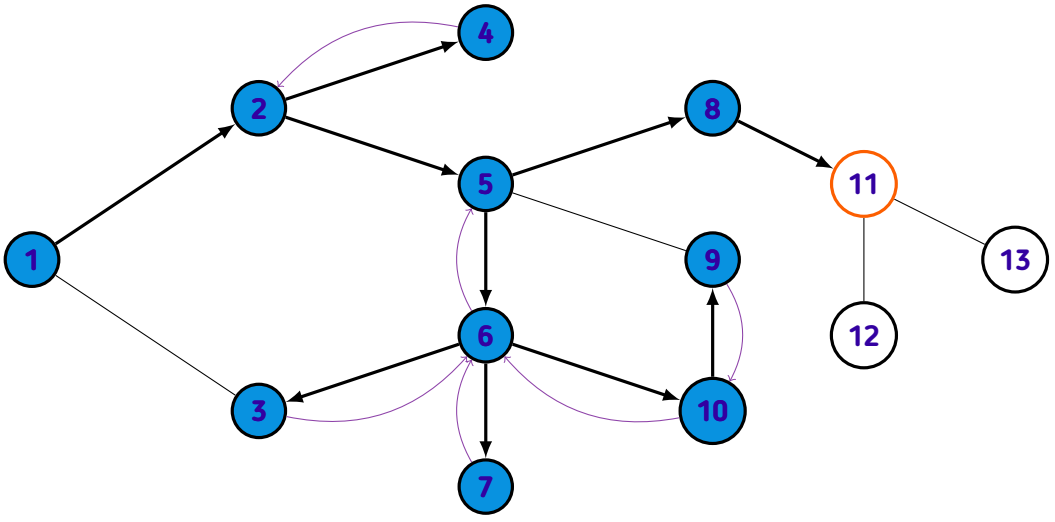

1
2
3
4
5
6
7
8
9
10
11
12
13

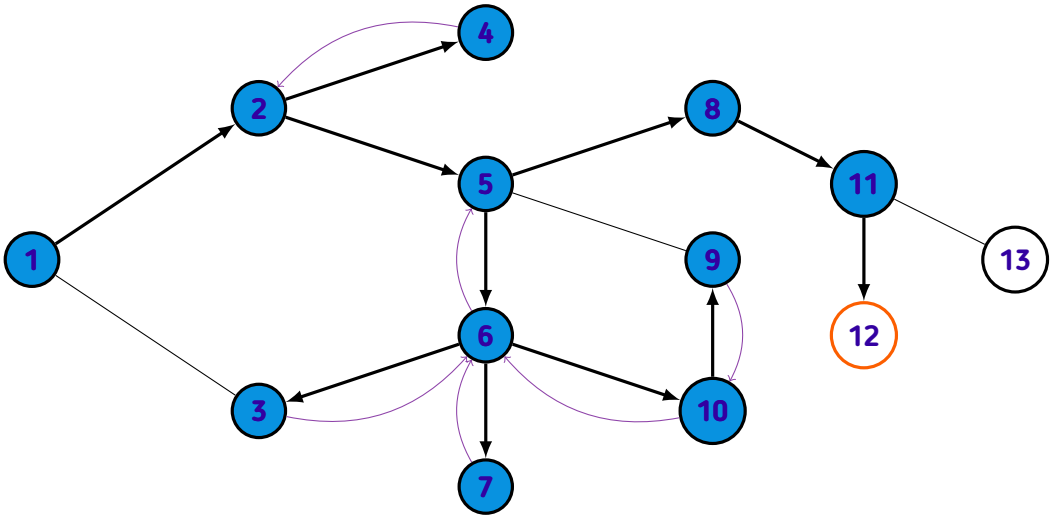

1
2
3
4
5
6
7
8
9
10
11
12
13

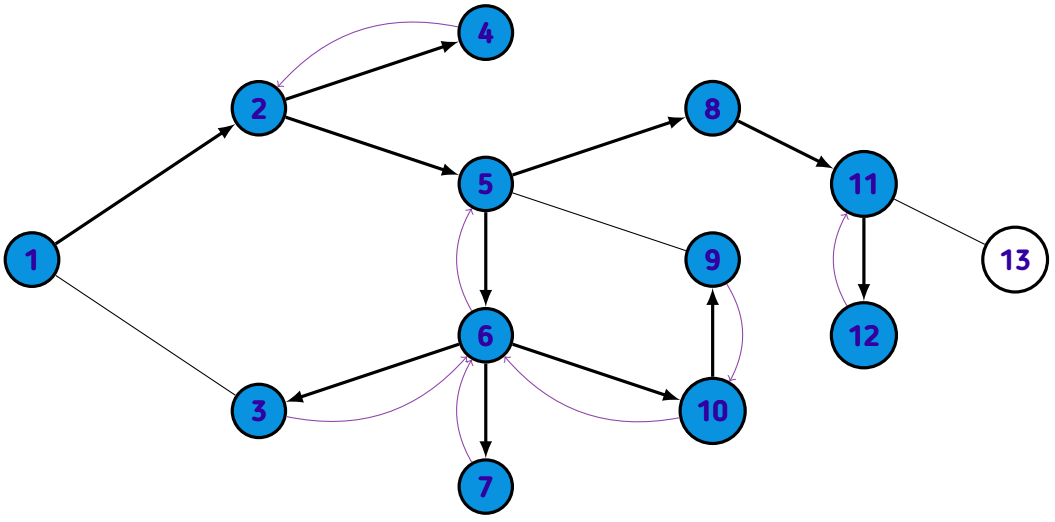

1
2
3
4
5
6
7
8
9
10
11
12
13

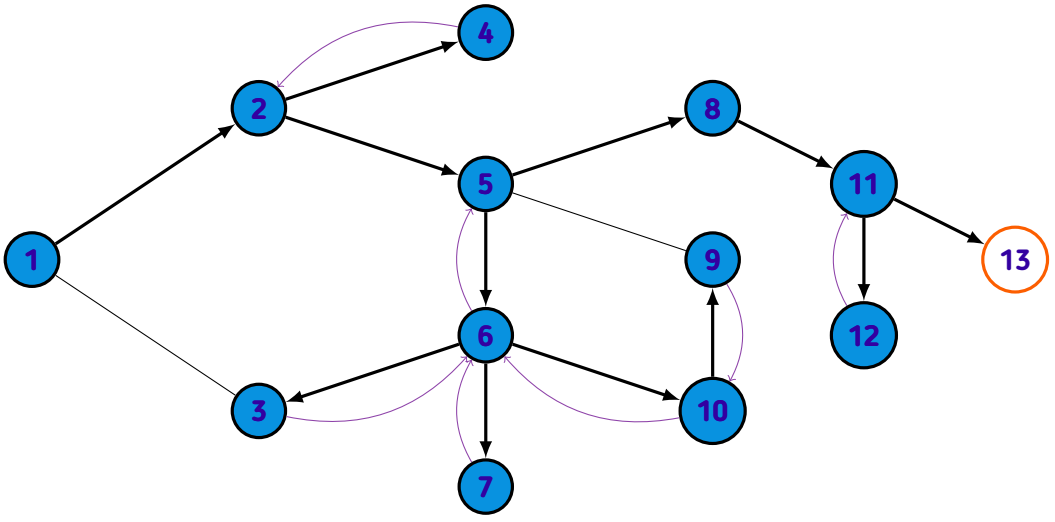

1
2
3
4
5
6
7
8
9
10
11
12
13

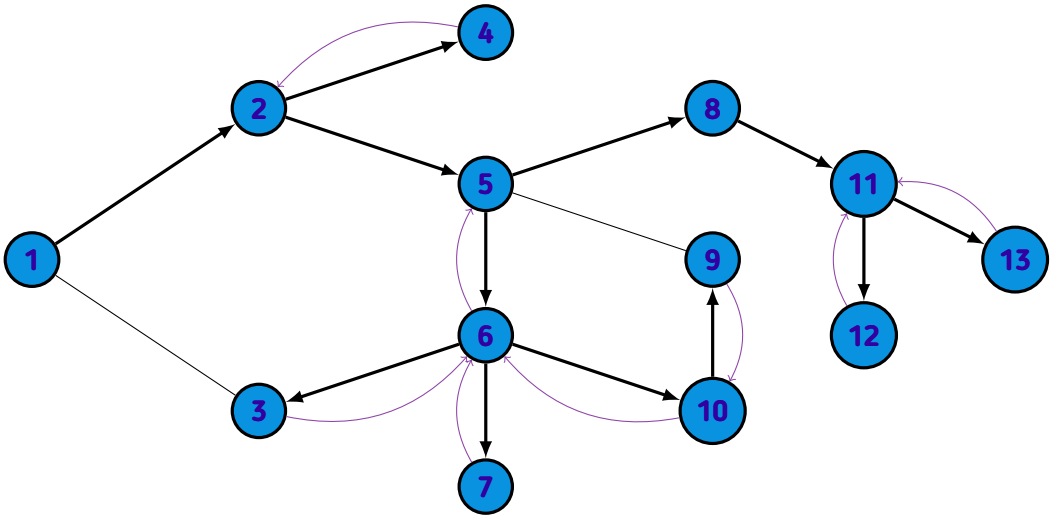

1
2
3
4
5
6
7
8
9
10
11
12
13

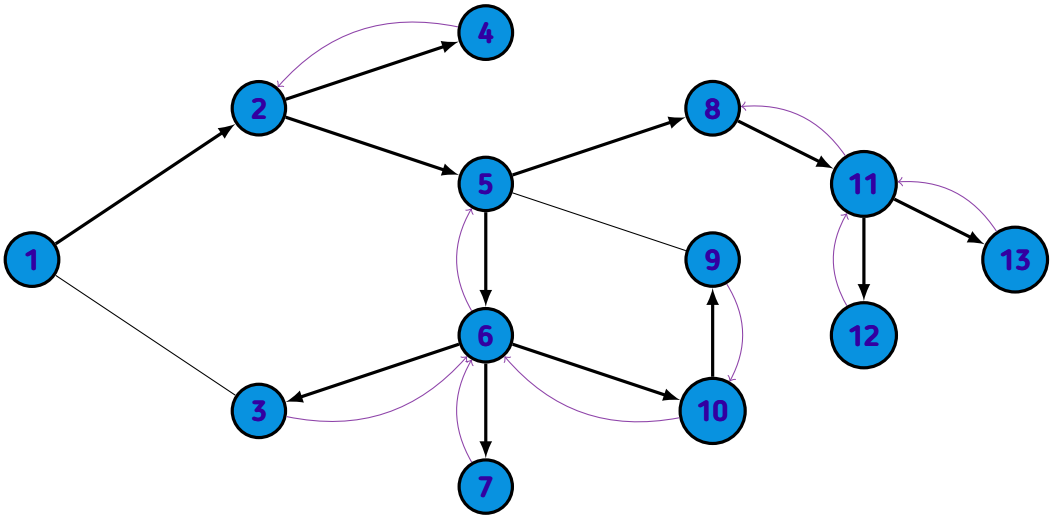

1
2
3
4
5
6
7
8
9
10
11
12
13

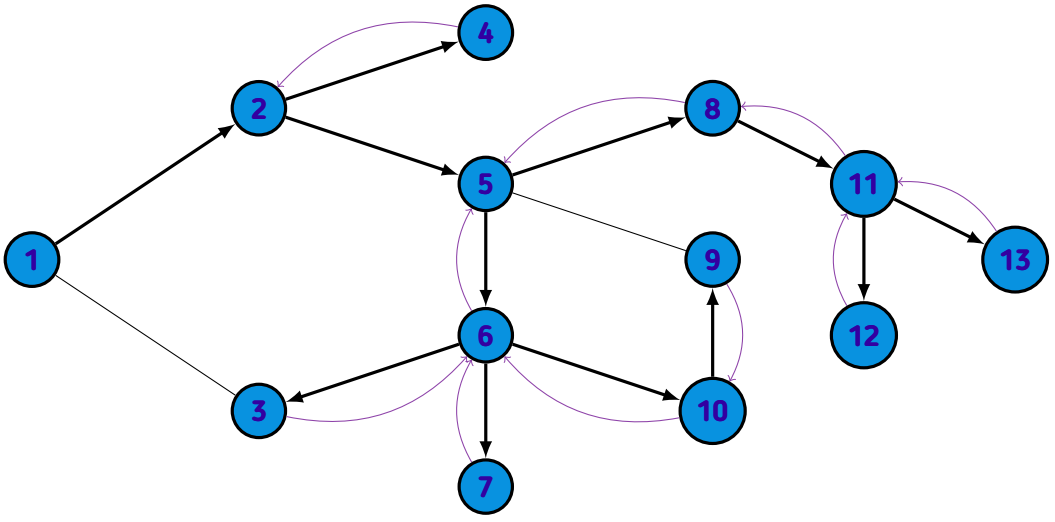

1
2
3
4
5
6
7
8
9
10
11
12
13

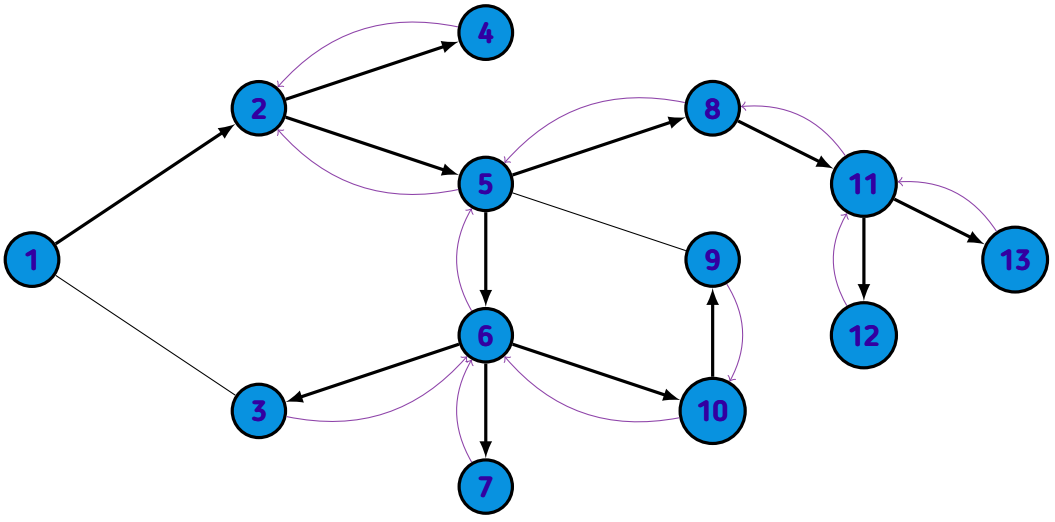

1
2
3
4
5
6
7
8
9
10
11
12
13

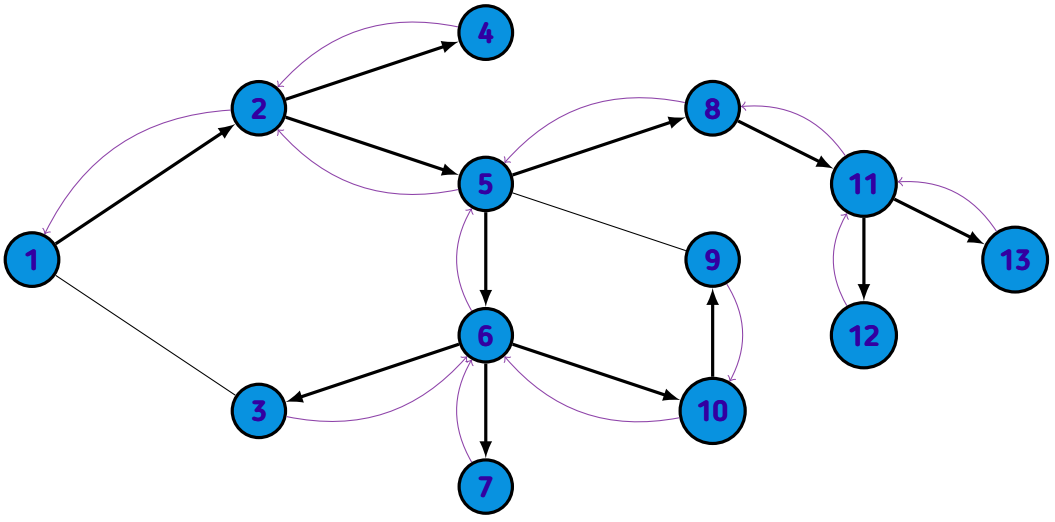

1
2
3
4
5
6
7
8
9
10
11
12
13


1
2
3
4
5
6
7
8
9
10
11
12
13


1
2
3
4
5
6
7
8
9
10
11
12
13


1
2
3
4
5
6
7
8
9
10
11
12
13


1
2
3
4
5
6
7
8
9
10
11
12
13


4
2
8
5
11
1
9
13
6
12
3
10
7

Características da DFS

Características da DFS

- **Cada nó é visitado uma única vez**

Características da DFS

- ⋆ **Cada nó é visitado uma única vez**
- ⋆ **Complexidade:** $O(N + M)$ **em listas de adjacências**

Características da DFS

- ⋆ **Cada nó é visitado uma única vez**
- ⋆ **Complexidade:** $O(N + M)$ **em listas de adjacências**
- ⋆ **Em matrizes de adjacência a complexidade é** $O(N^2)$

Implementação da DFS

Implementação da DFS

- A DFS pode ser implementada por recursão

Implementação da DFS

- A DFS pode ser implementada por recursão
- Caso-base: vértice já visitado

Implementação da DFS

- ⋆ **A DFS pode ser implementada por recursão**
- ⋆ **Caso-base: vértice já visitado**
- ⋆ **Chamada recursiva: vizinhos de u ainda não visitados**

```cpp
const int MAX { 200010 };

bitset<MAX> visited;
vector<int> adj[MAX];

void dfs(int u)
{
    if (visited[u])
        return;

    // processa/visita u

    visited[u] = true;

    for (auto v : adj[u])
        dfs(v);
}
```

Problemas sugeridos

1. **Codeforces Round #453 (Div. 2) – Problem B: Coloring a Tree**
2. **Codeforces Round #503 (by SIS, Div. 2) – Problem B: Badge**
3. **OJ 10113 – Exchange Rates**
4. **OJ 12442 – Forwarding Emails**

Referências

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