Travessia de Grafos

Breadth-First Search

Prof. Edson Alves

2019

Faculdade UnB Gama

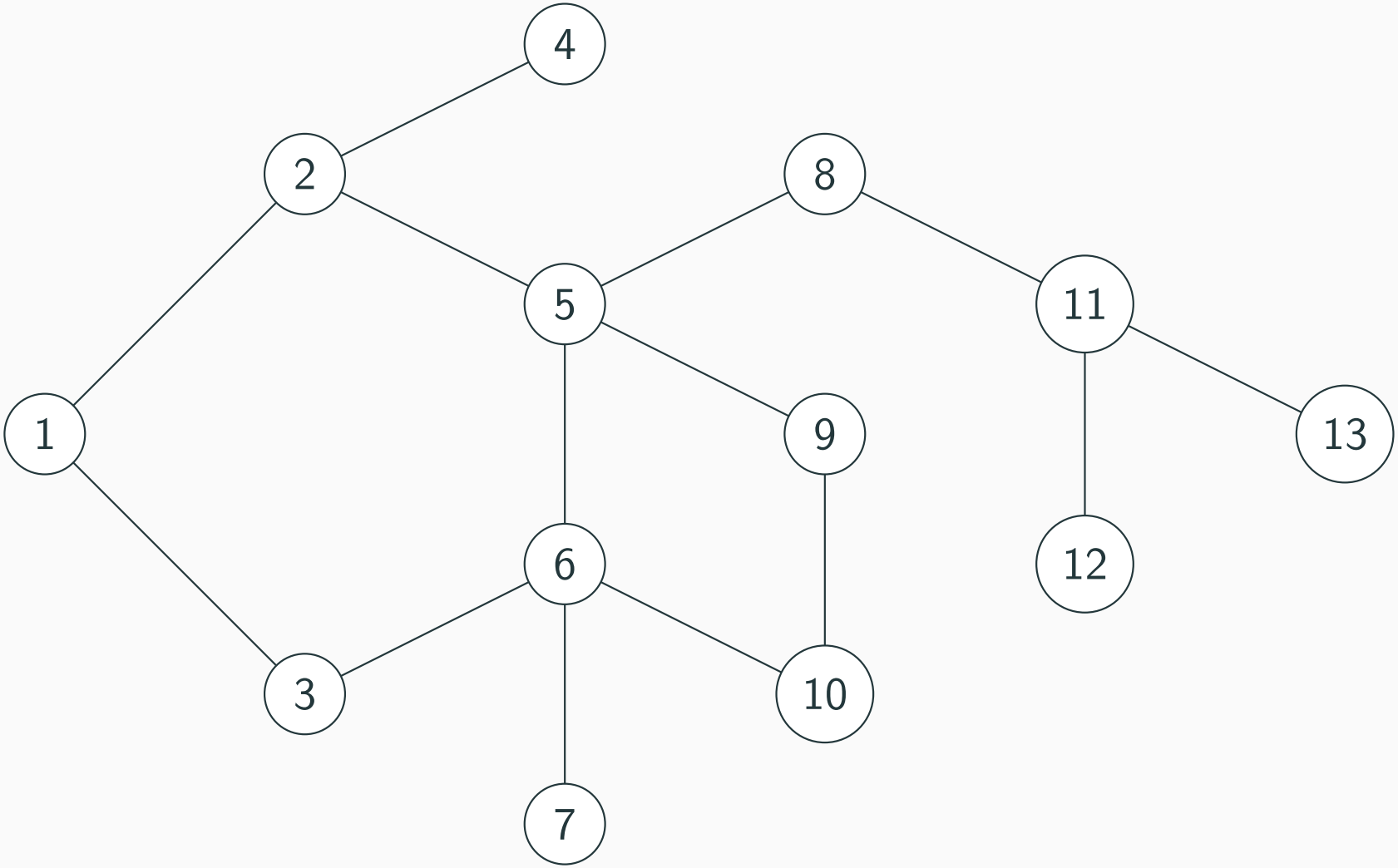
Sumário

Definição

Definição

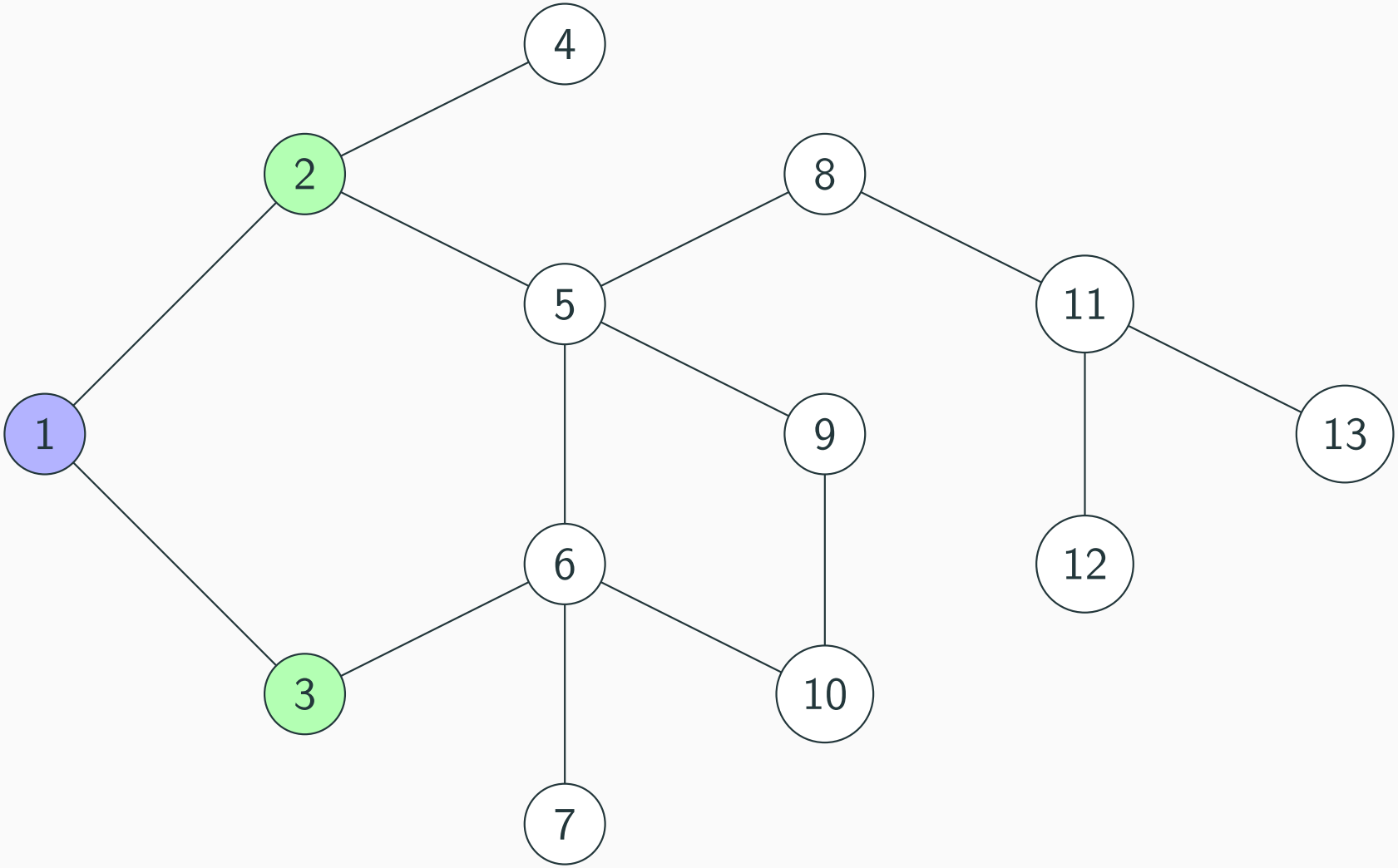
- A travessia por largura (*Breadth-First Search – BFS*) visita os nós em ordem crescente em relação à distância entre estes nós e o nó inicial s
- Desta forma, um subproduto da BFS é a distância de todos os nós que estão conectados até s
- A implementação da BFS é mais trabalhosa do que a da DFS, porque não se vale de recursão, sendo necessário o uso de uma fila explicitamente
- Ambas travessias visitam os mesmos nós, porém em ordens distintas
- A complexidade da BFS é $O(N + M)$, onde N é o número de vértices e M o número de arestas do grafo conectado
- No caso de uma representação por matrizes de adjacência, a complexidade é $O(N^2)$

Visualização da BFS



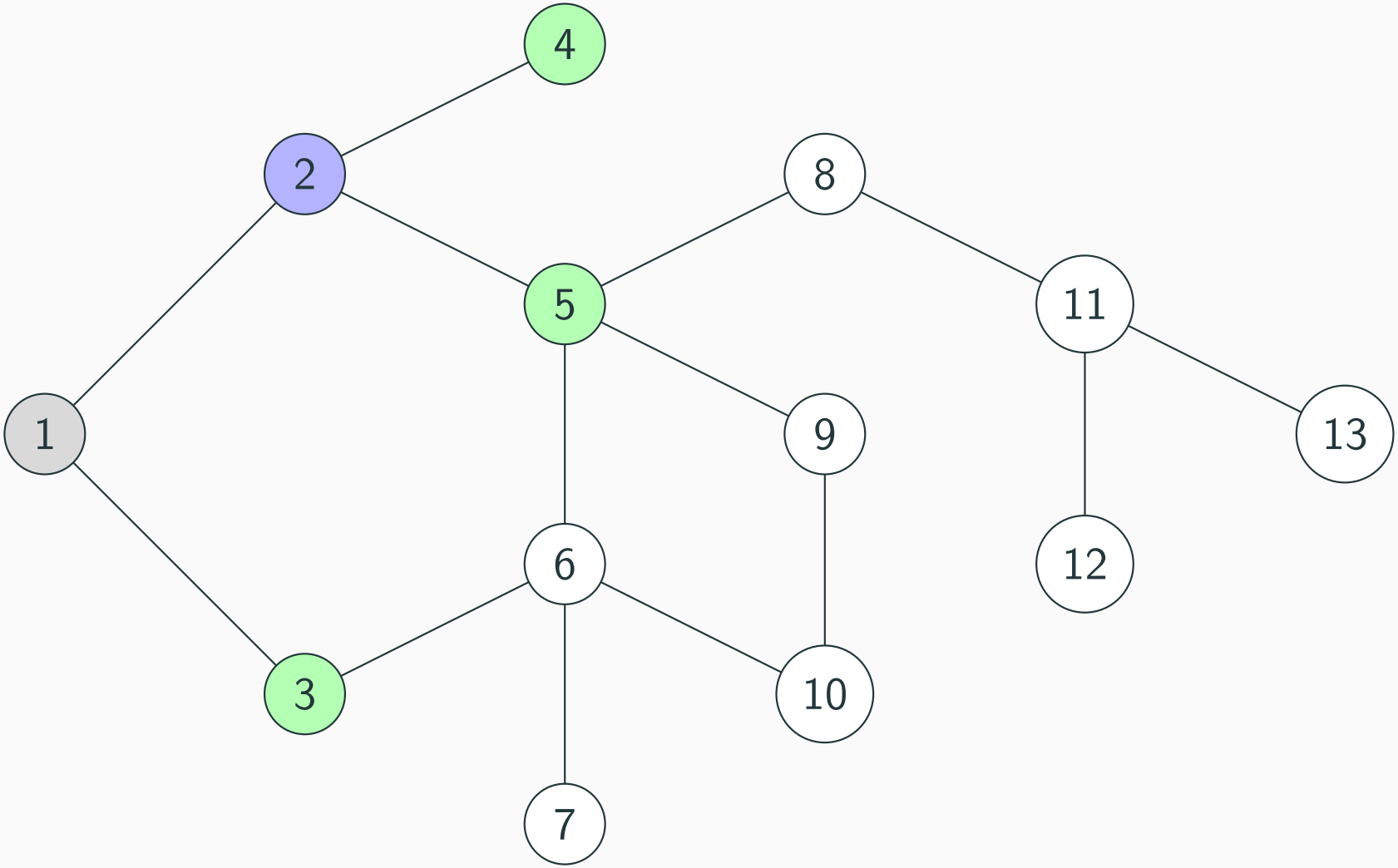
Fila: 1

Visualização da BFS



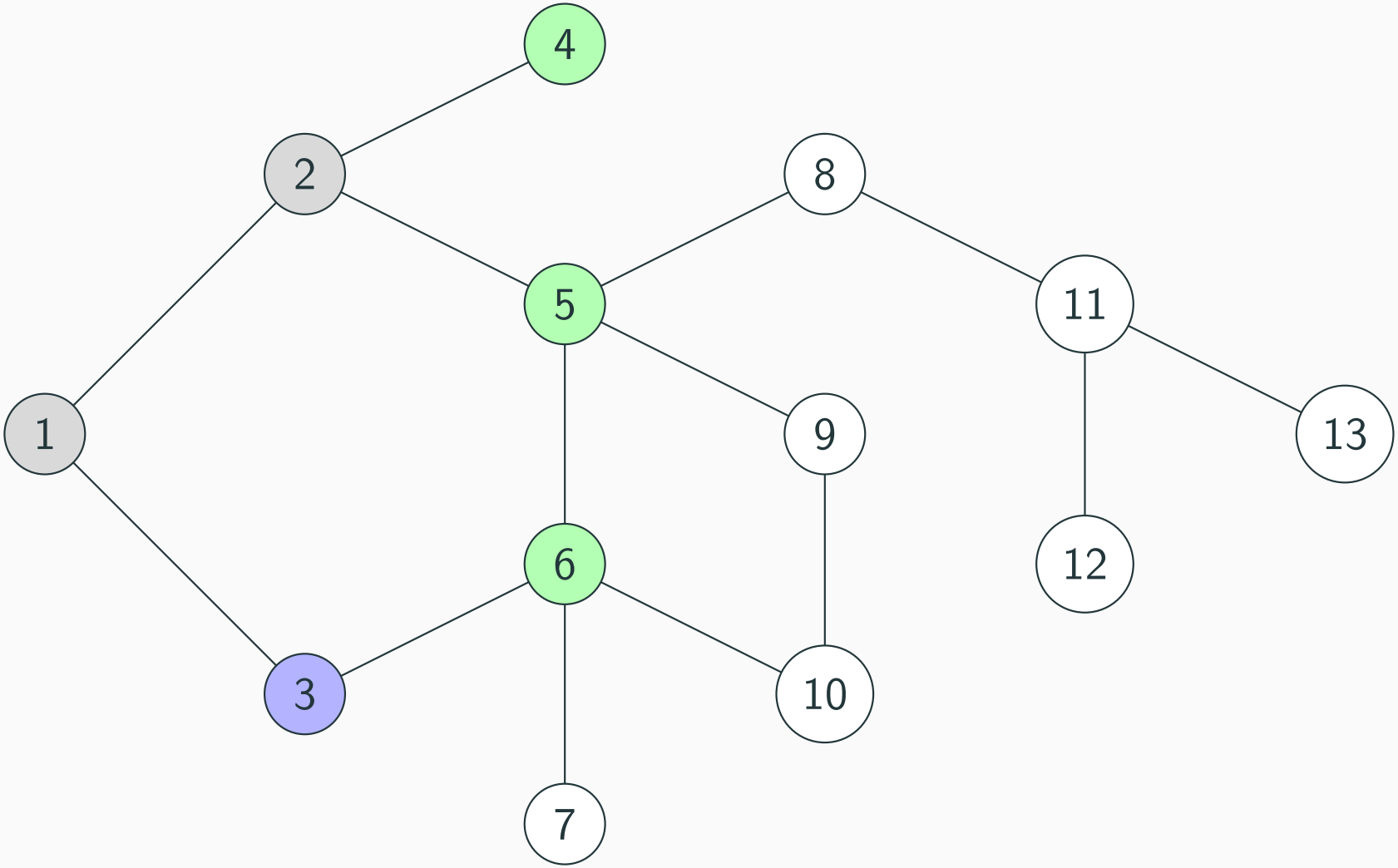
Fila: 2 3

Visualização da BFS



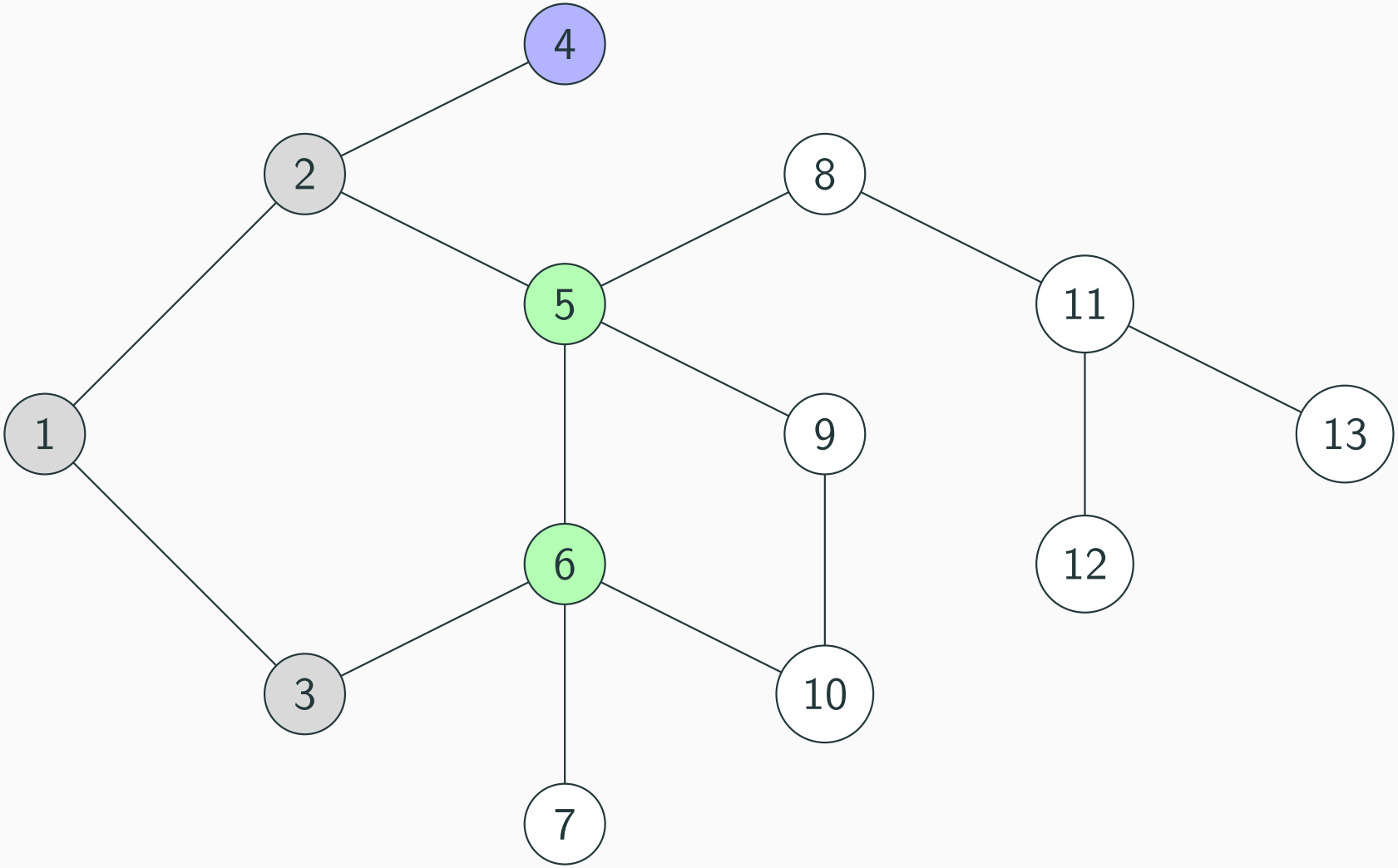
Fila: 3 4 5

Visualização da BFS



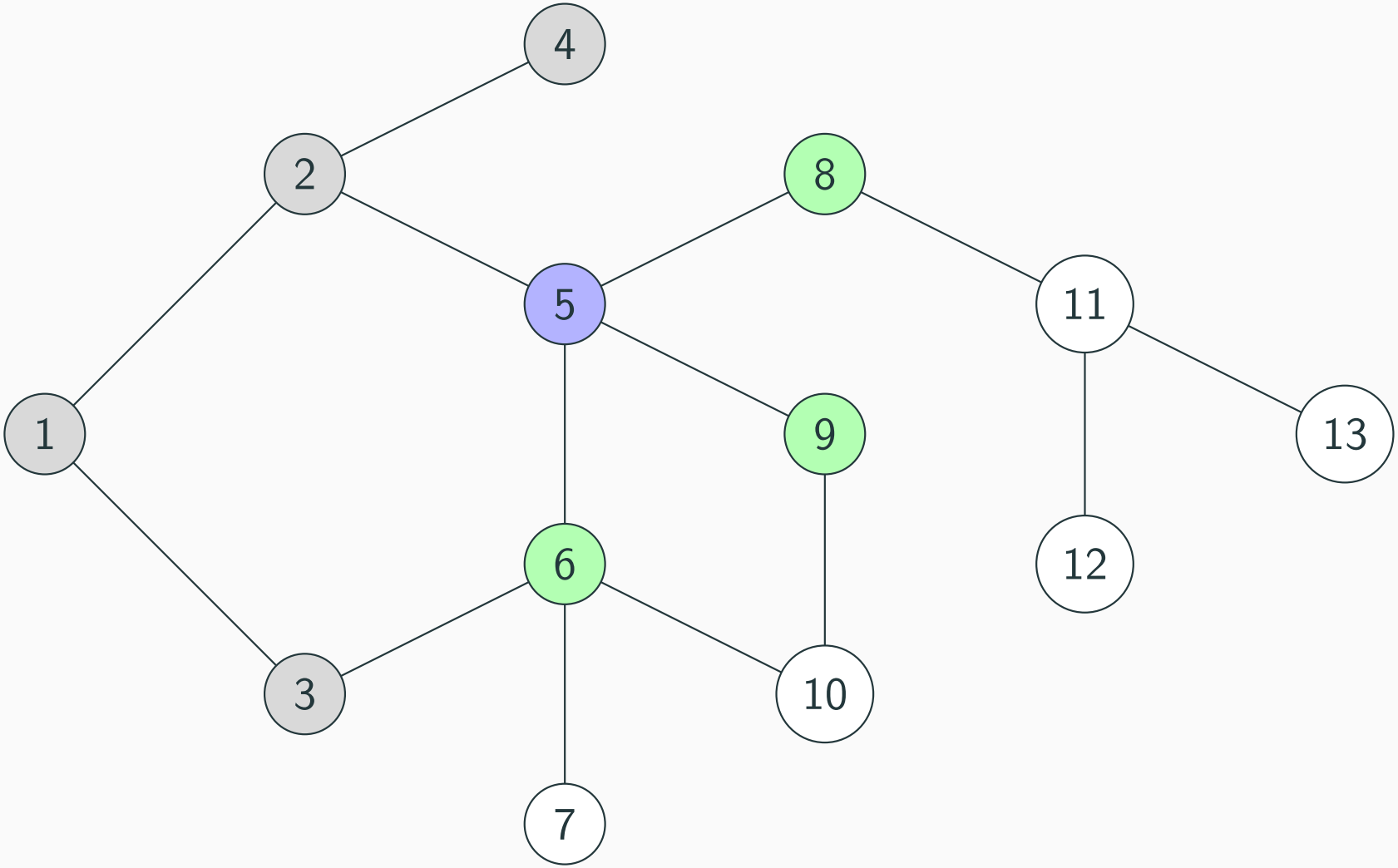
Fila: 4 5 6

Visualização da BFS



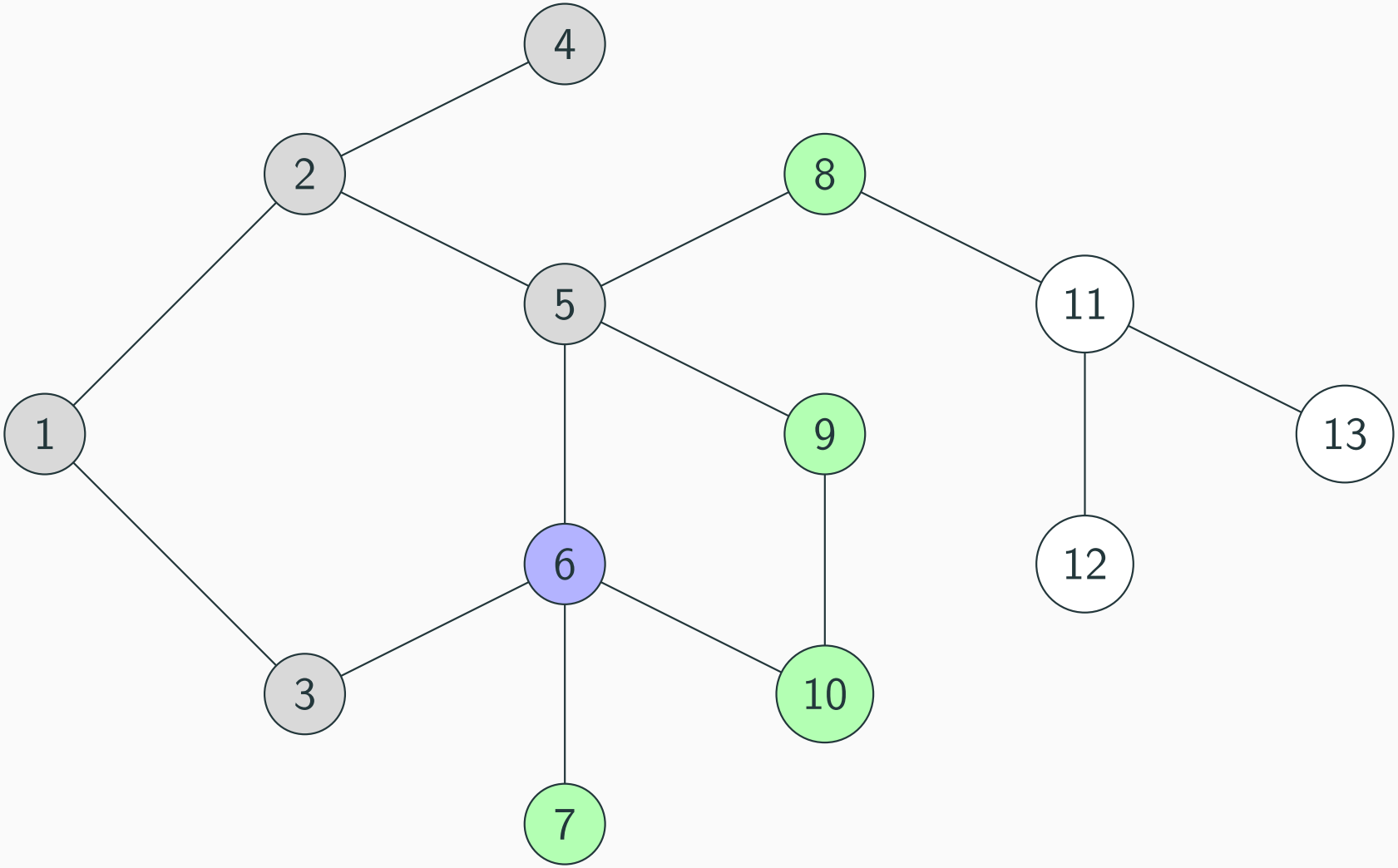
Fila: 5 6

Visualização da BFS



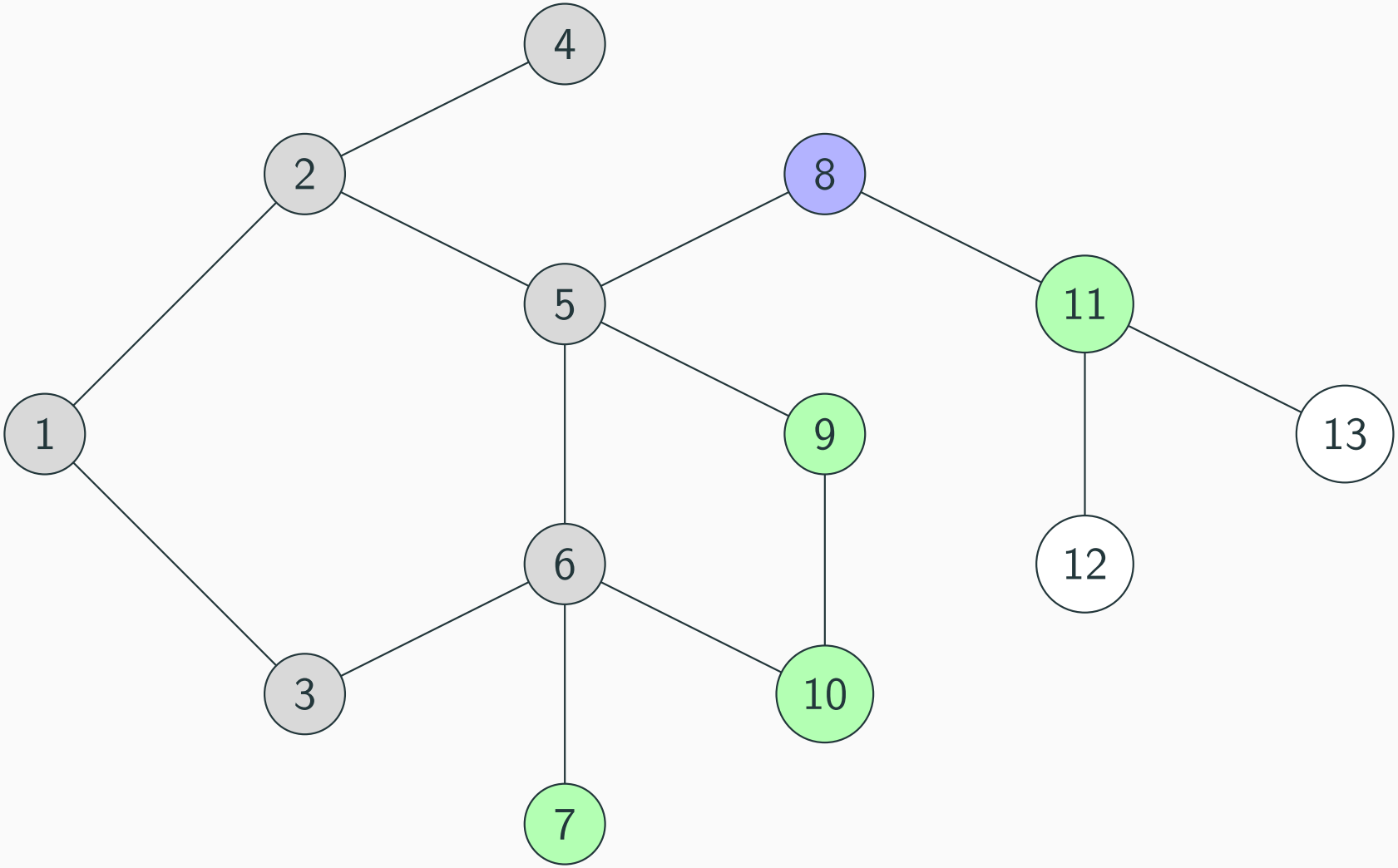
Fila: 6 8 9

Visualização da BFS



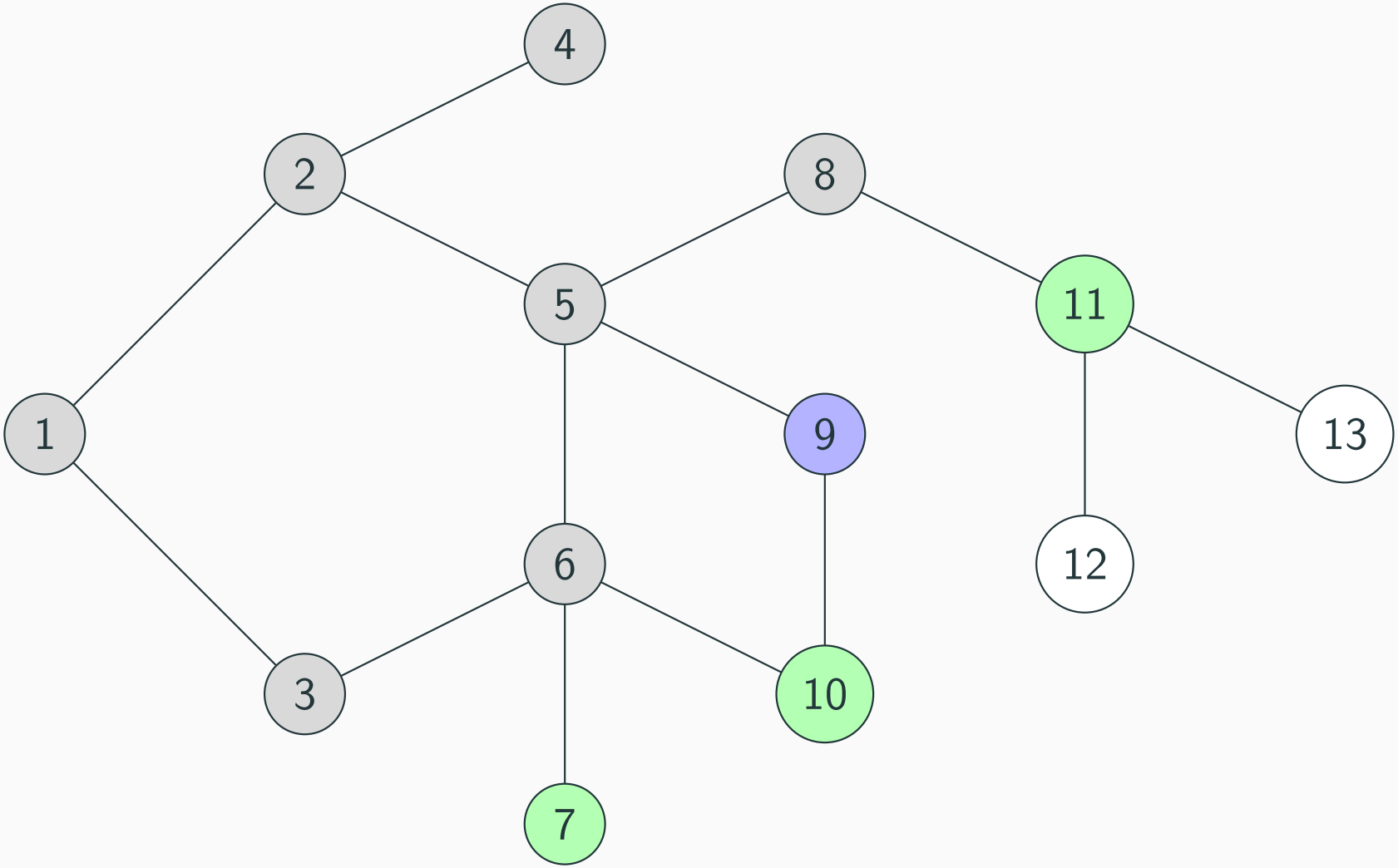
Fila: 8 9 7 10

Visualização da BFS



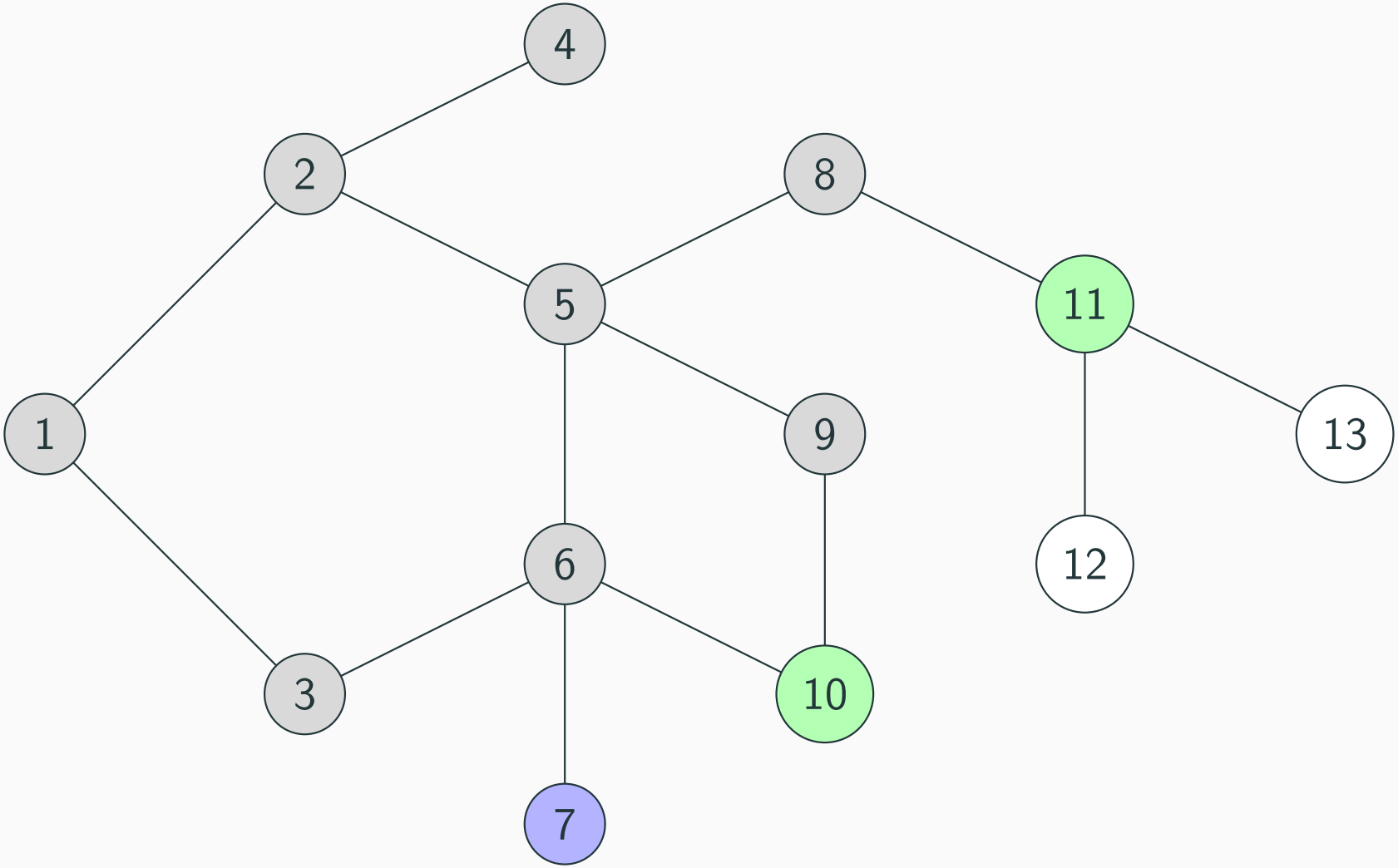
Fila: 9 7 10 11

Visualização da BFS



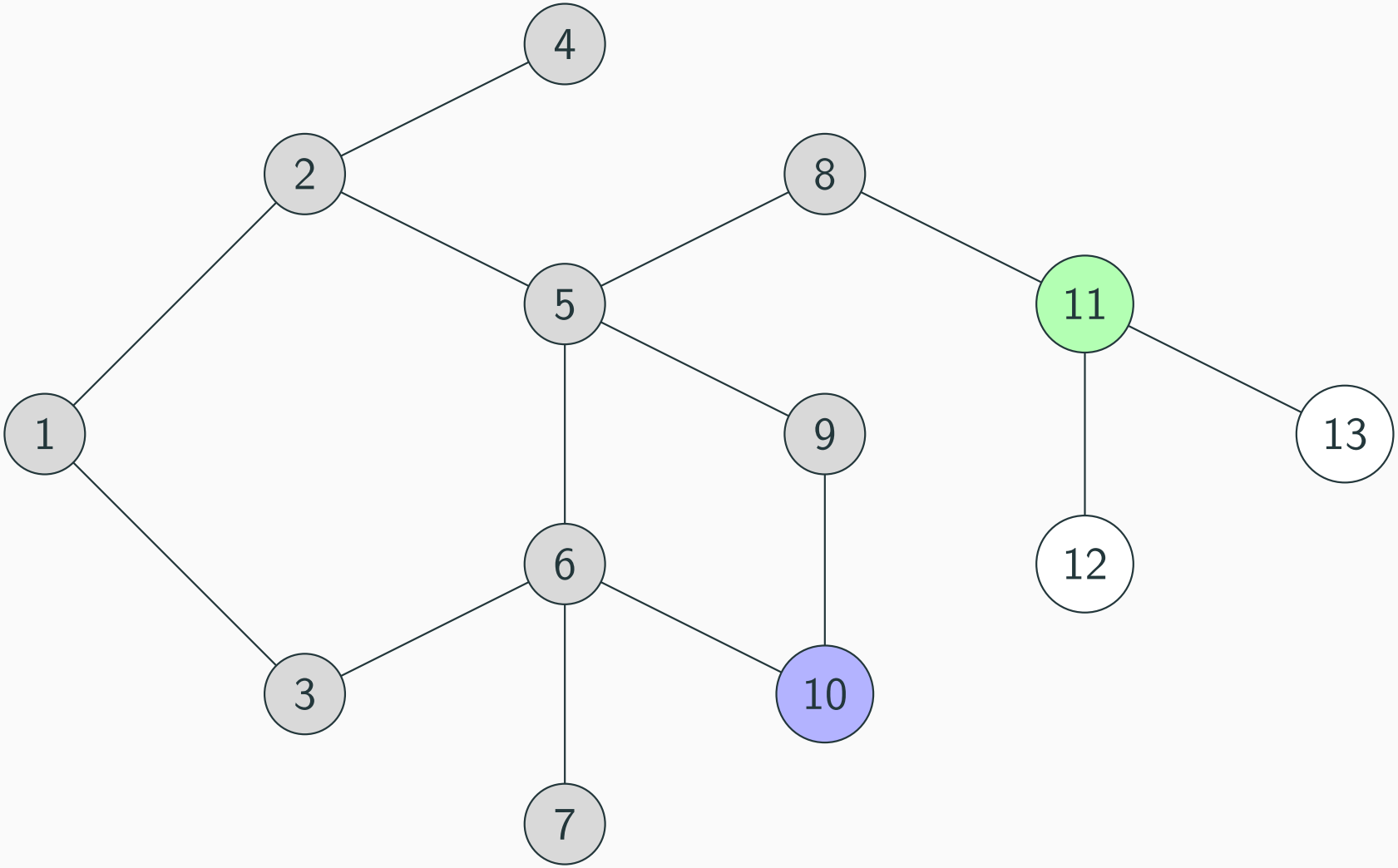
Fila: 7 10 11

Visualização da BFS



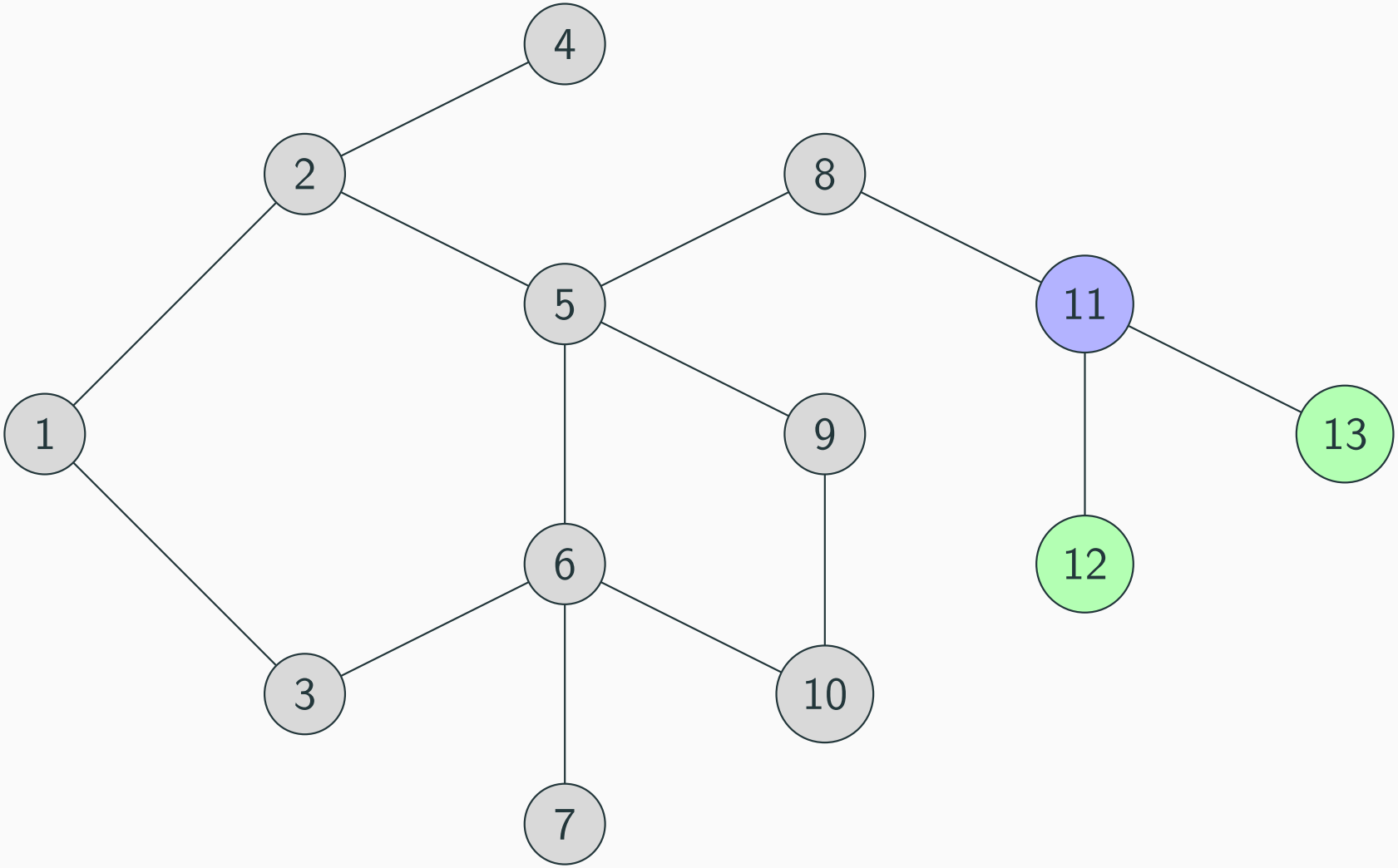
Fila: 10 11

Visualização da BFS



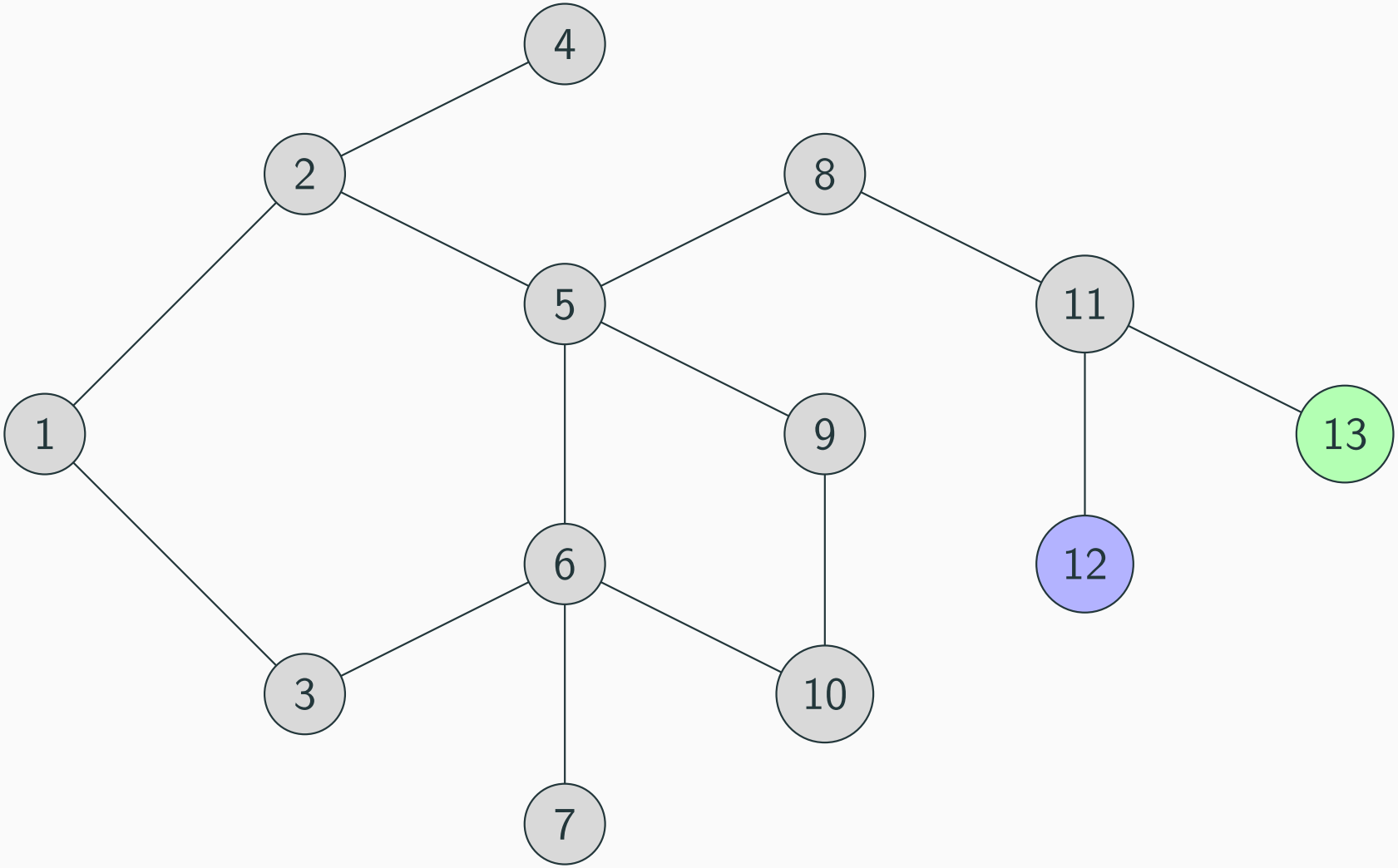
Fila: 11

Visualização da BFS



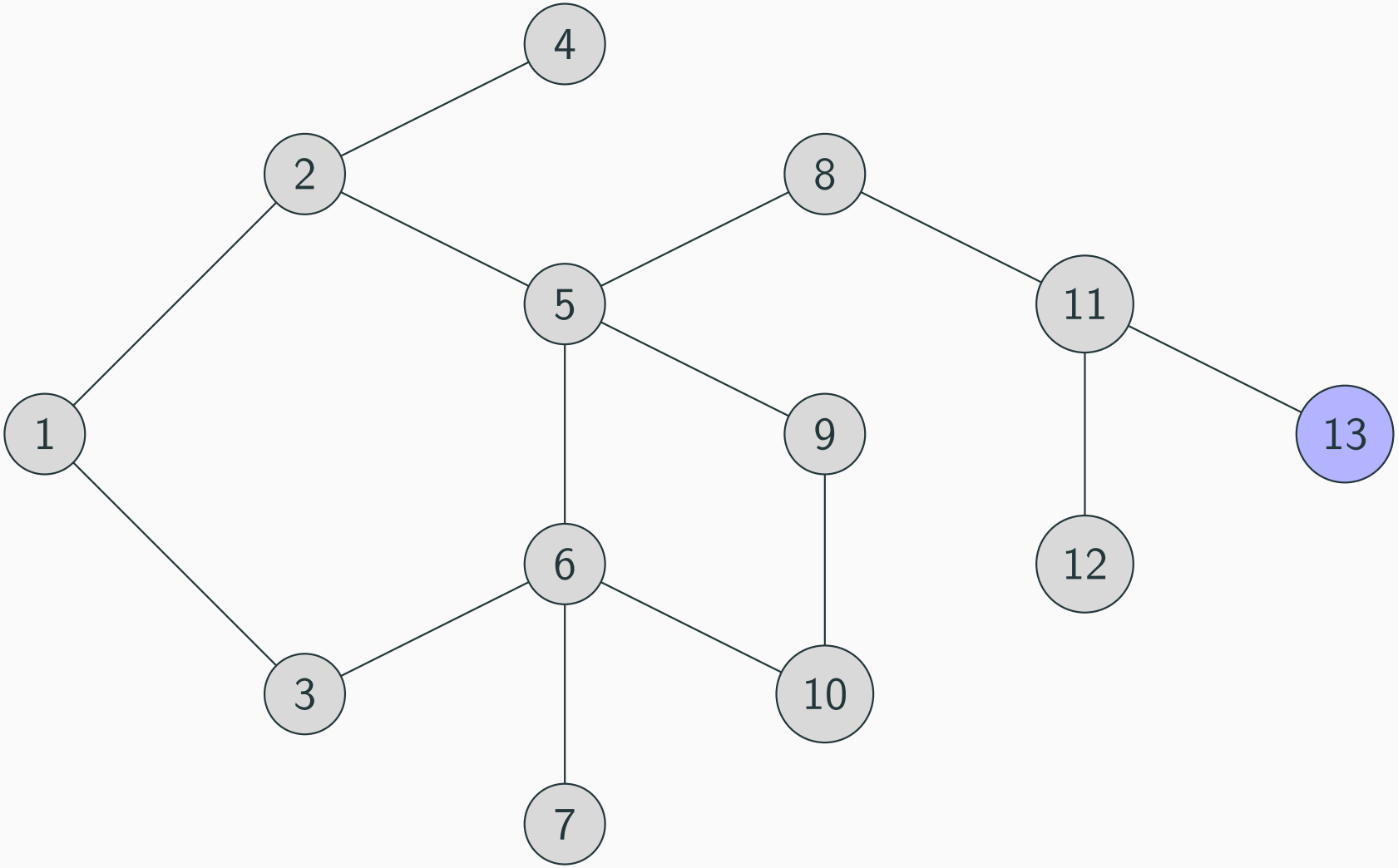
Fila: 12 13

Visualização da BFS



Fila: 13

Visualização da BFS



Fila: vazia

Implementação da BFS

Breadth-First Search

- Primeiramente são visitados todos os vizinhos do nó inicial s
- Em seguida são visitados os vizinhos de cada vizinho, e assim sucessivamente
- Uma fila é mantida para manter a ordem dos vértices a serem visitados
- A fila é processada retirando-se o primeiro da elemento da fila e enfileirando-se todos os vizinhos deste nó que ainda não foram visitados
- Antes de ser inserido na fila, o nó é marcado como visitado

Implementação da BFS em C++

```cpp
#include <bits/stdc++.h>

using namespace std;
using ii = pair<int, int>;

const int MAX { 100010 };

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

void bfs(int s, const function<void(int)>& process)
{
    visited.reset();

    queue<int> q;
    q.push(s);
    visited[s] = true;
    dist[s] = 0;

```

Implementação da BFS em C++

```
21      while (not q.empty())
22      {
23          auto u = q.front();
24          q.pop();
25
26          process(u);
27
28          for (const auto& v : adj[u])
29          {
30              if (visited[v])
31                  continue;
32
33              visited[v] = true;
34              dist[v] = dist[u] + 1;
35              q.push(v);
36          }
37      }
38  }
39
```

Implementação da BFS em C++

```cpp
int main()
{
    ii edges[] { ii(1, 2), ii(1, 3), ii(2, 4), ii(2, 5), ii(3, 6),
        ii(5, 6), ii(5, 8), ii(5, 9), ii(6, 7), ii(6, 10), ii(8, 11),
        ii(9, 10), ii(11, 12), ii(11, 13) };

    for (const auto& [u, v] : edges)
    {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    bfs(1, [](int u) { cout << u << ' '; });
    cout << '\n';

    return 0;
}
```

Referências

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**, Bartlomiej. *C++17 in Detail*, 2018[1].

[1] https://leanpub.com/cpp17indetail