Travessia de Grafos

Aplicações

Prof. Edson Alves

2018

Faculdade UnB Gama

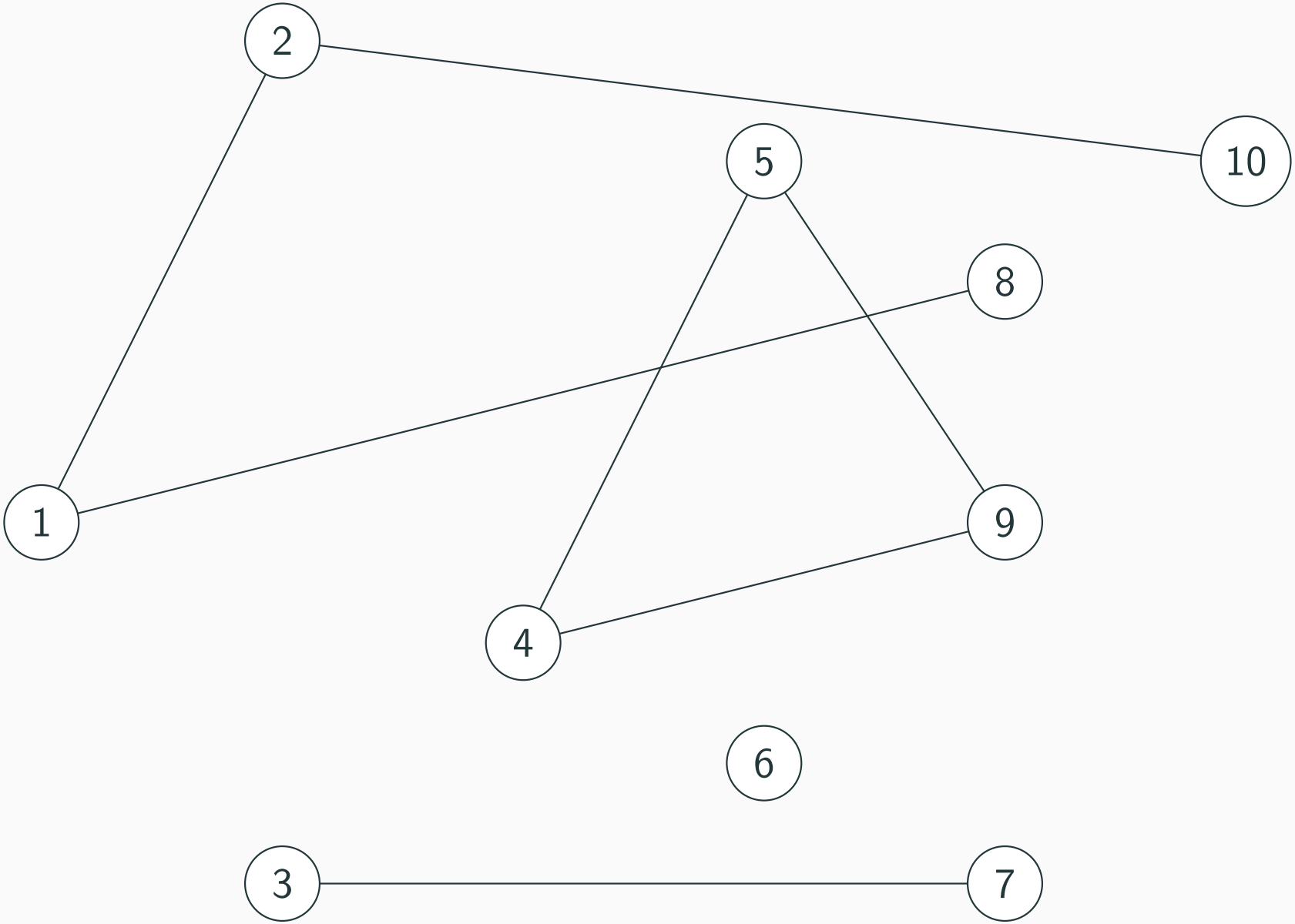
Sumário

Componentes Conectados

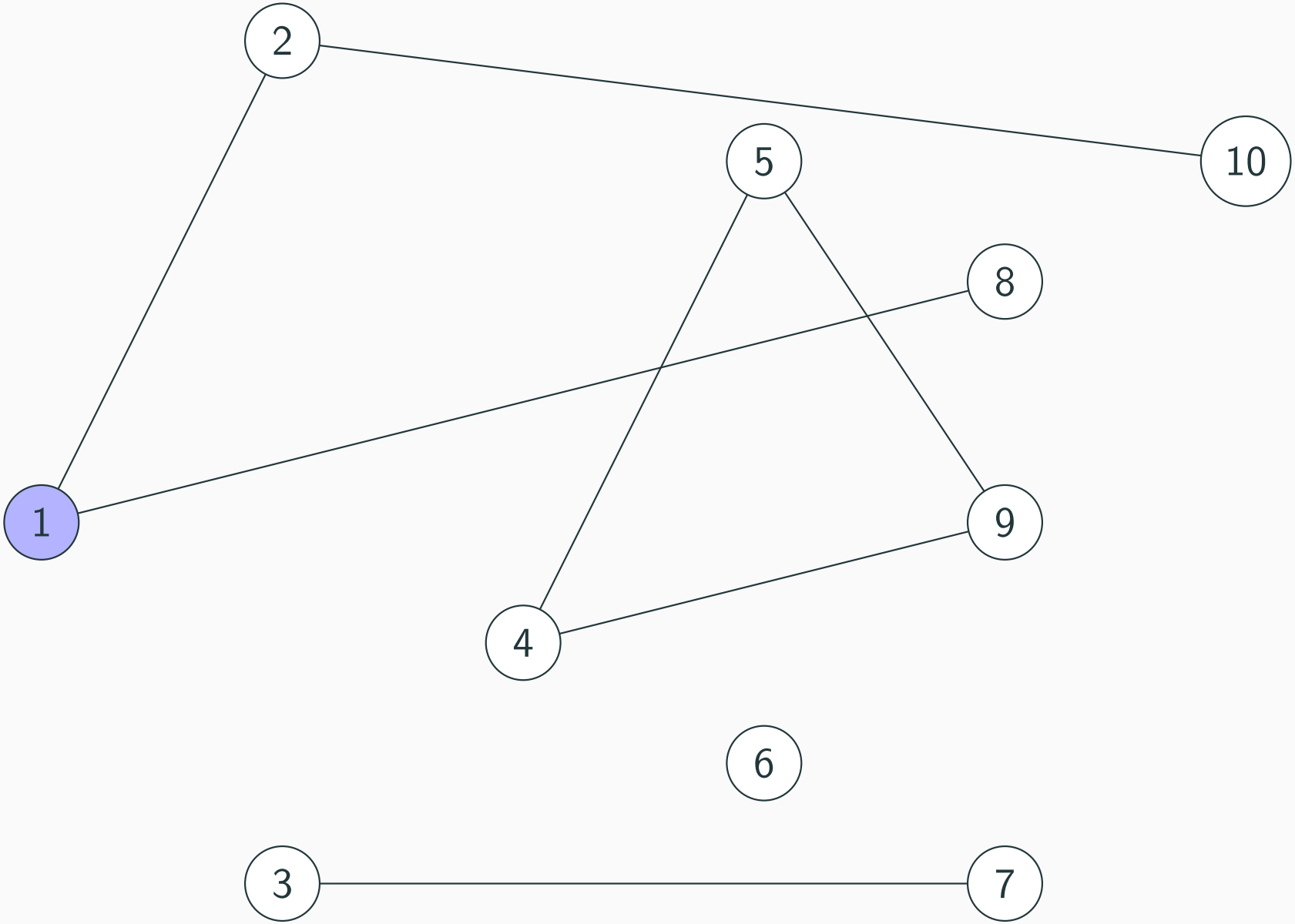
Conectividade de um grafo

- Um grafo G é dito conectado se, para qualquer par de vértices $u, v \in G$, com $u \neq v$, existe ao menos um caminho de u até v
- Uma maneira de se verificar se um grafo é conectado ou não é iniciar uma travessia em um vértice s qualquer
- Se a travessia visitar todos os N nós de G o grafo é conectado
- Caso um ou mais vértices não seja visitado, os nós visitados formam um componente conectado de G
- Para identificar todos os componentes conectados do grafo basta iniciar uma nova travessia em um dos vértices não visitados, enquanto houverem vértices não visitados

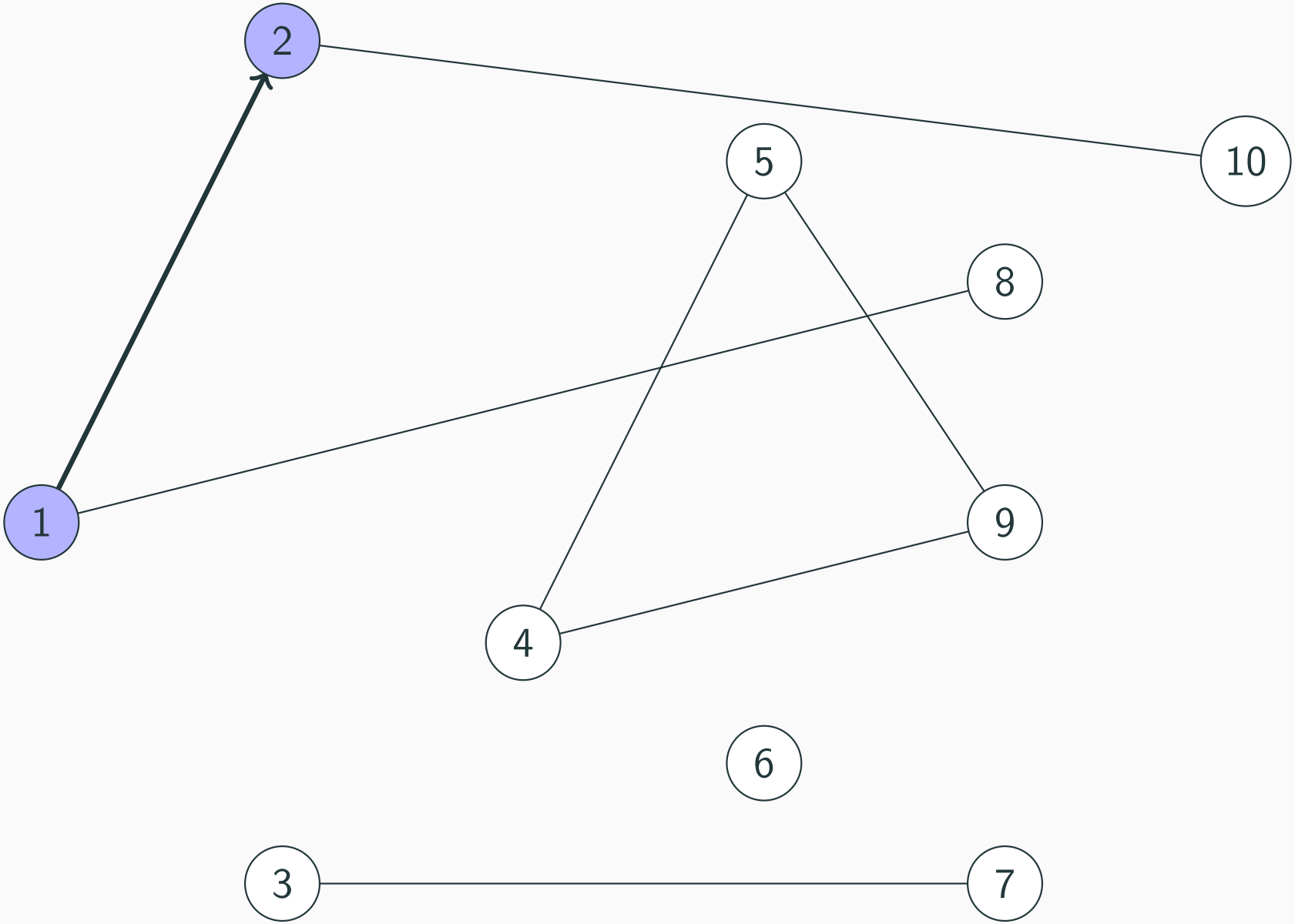
Visualização da identificação dos componentes conectados



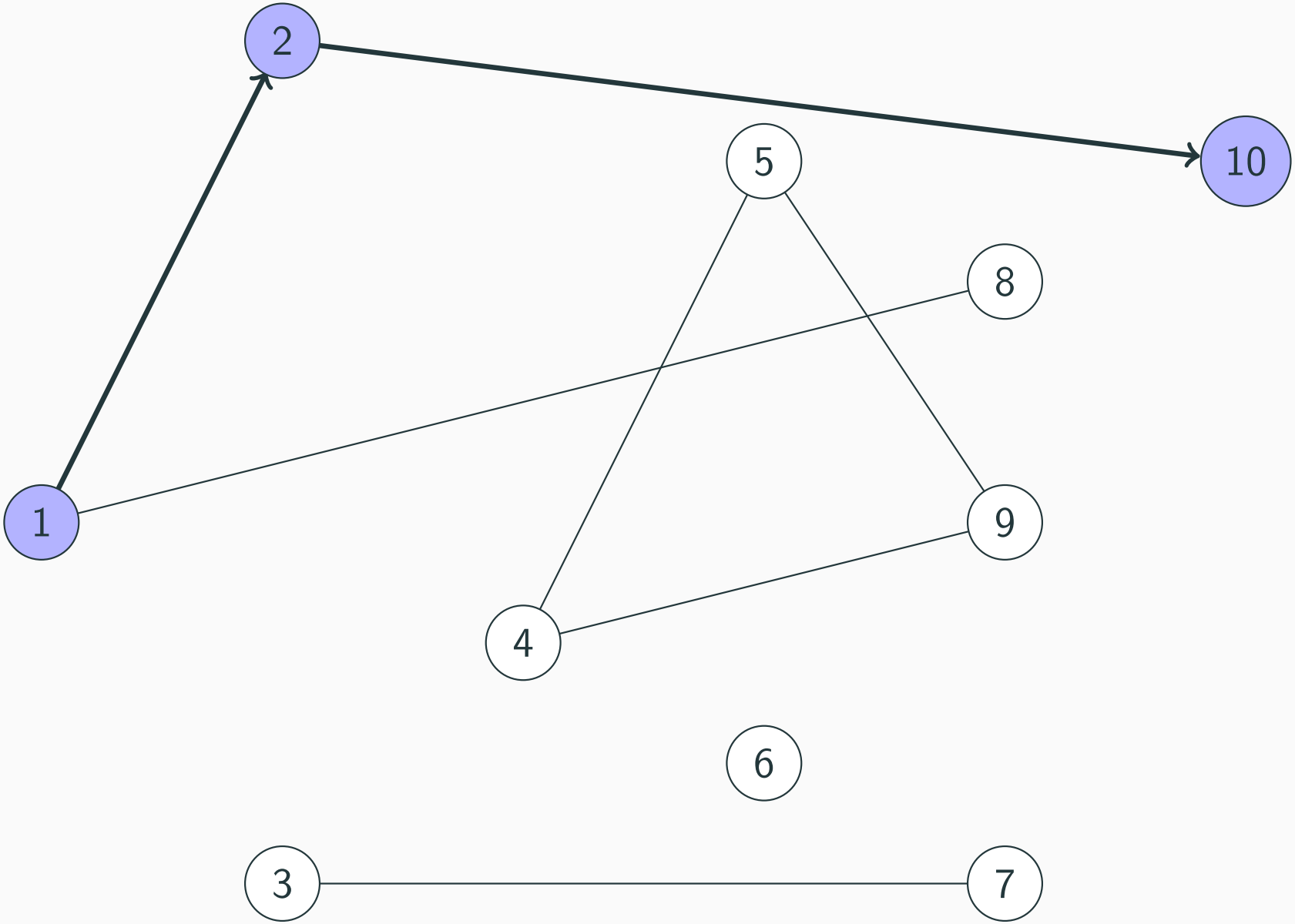
Visualização da identificação dos componentes conectados



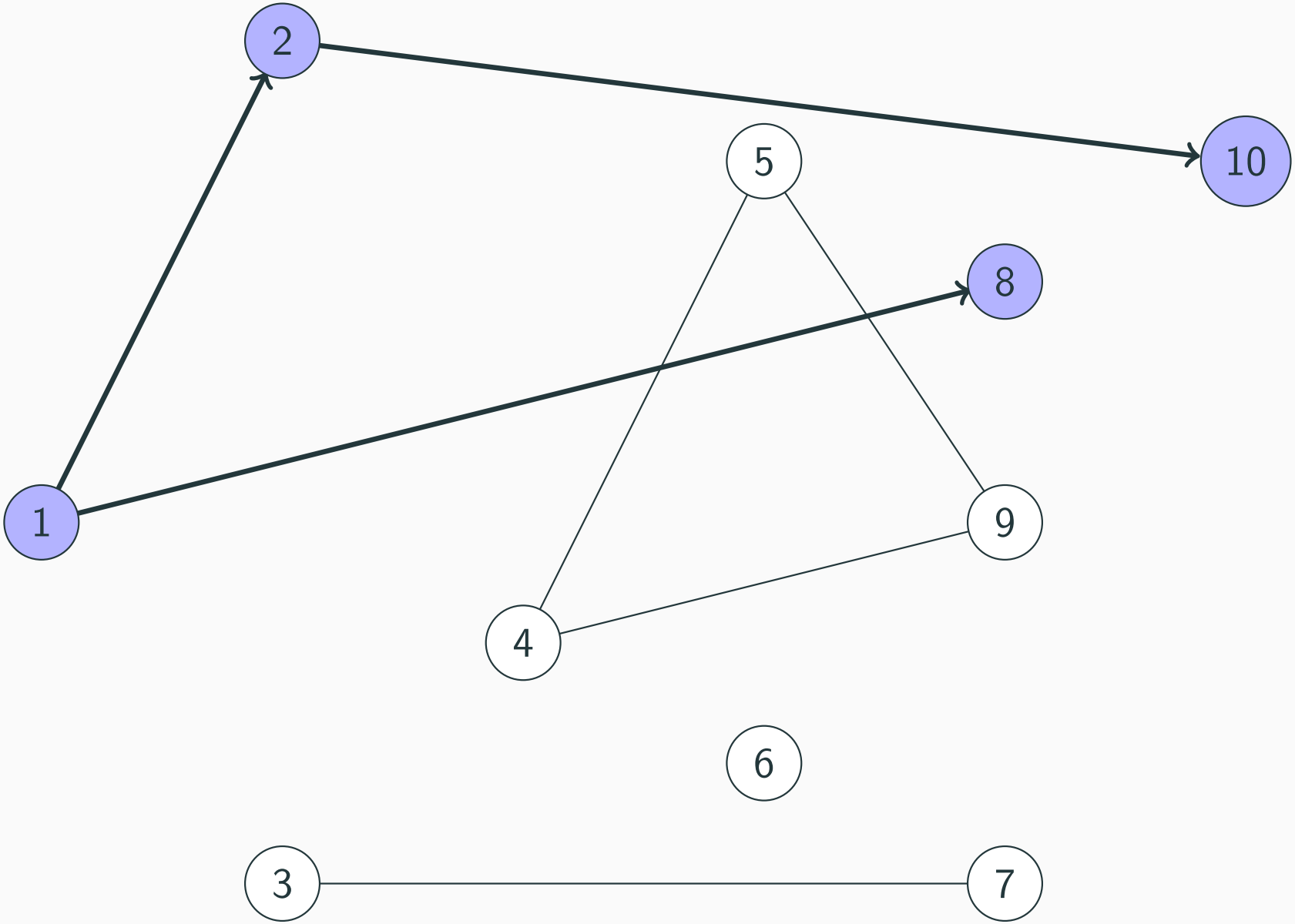
Visualização da identificação dos componentes conectados



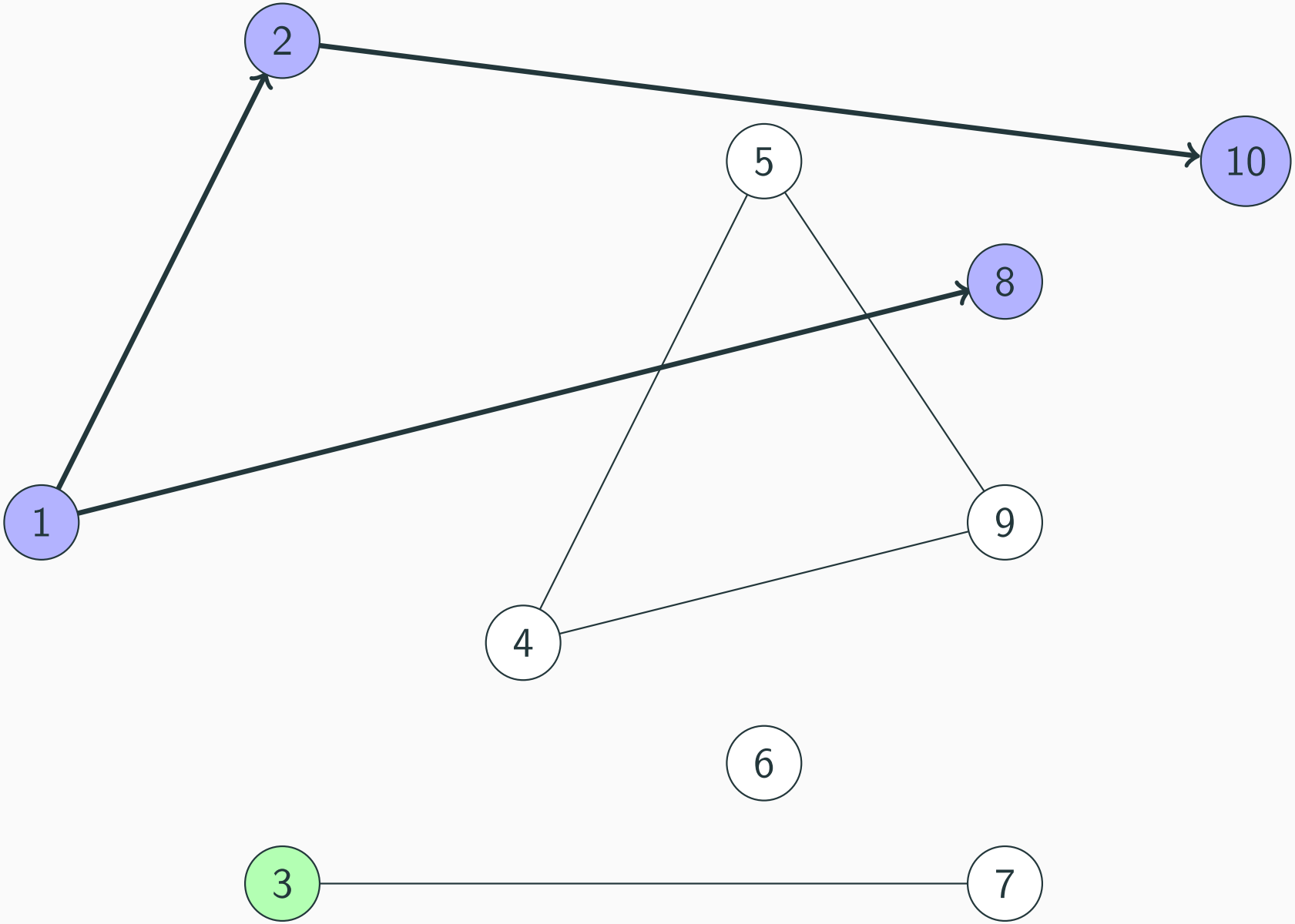
Visualização da identificação dos componentes conectados



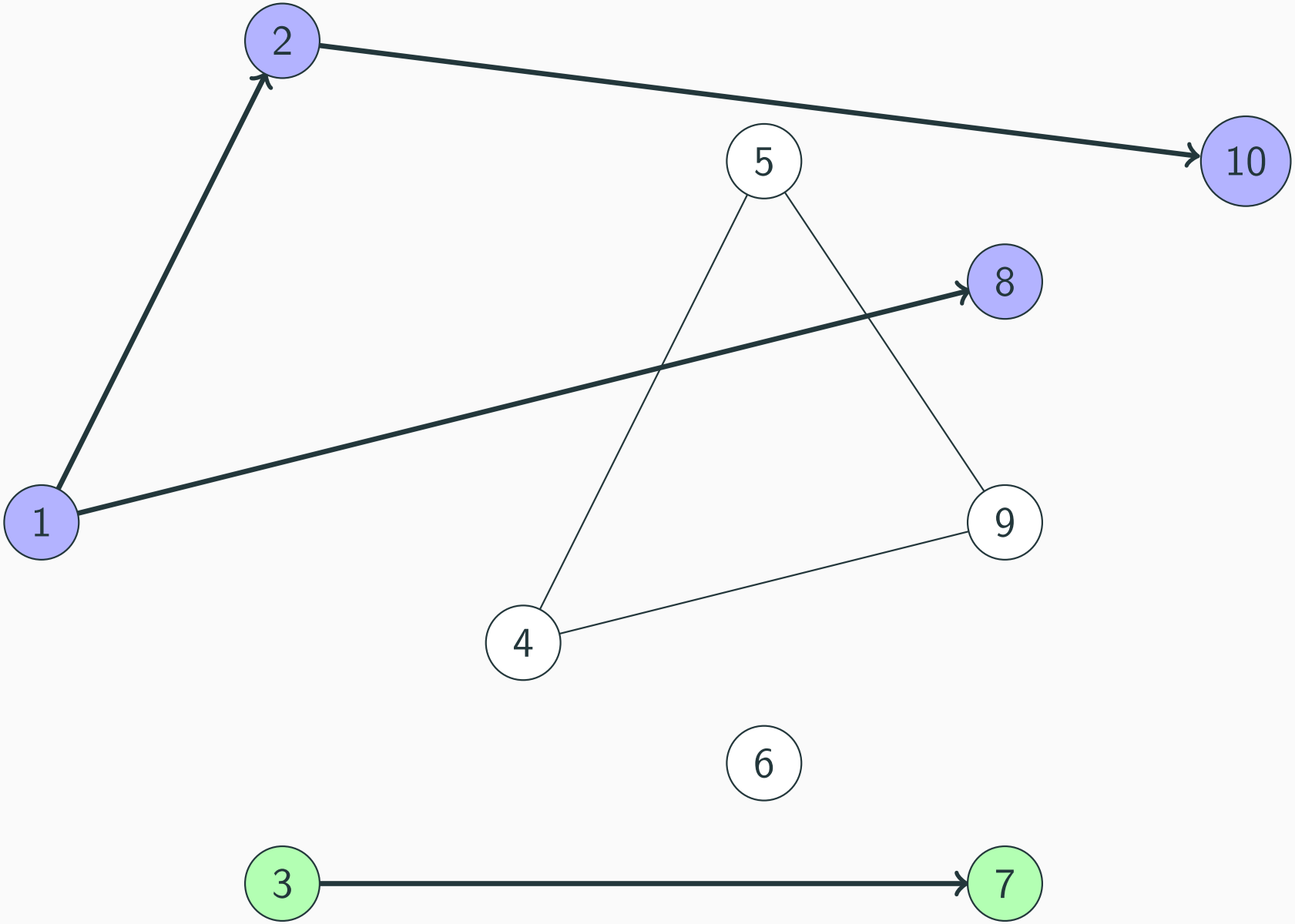
Visualização da identificação dos componentes conectados



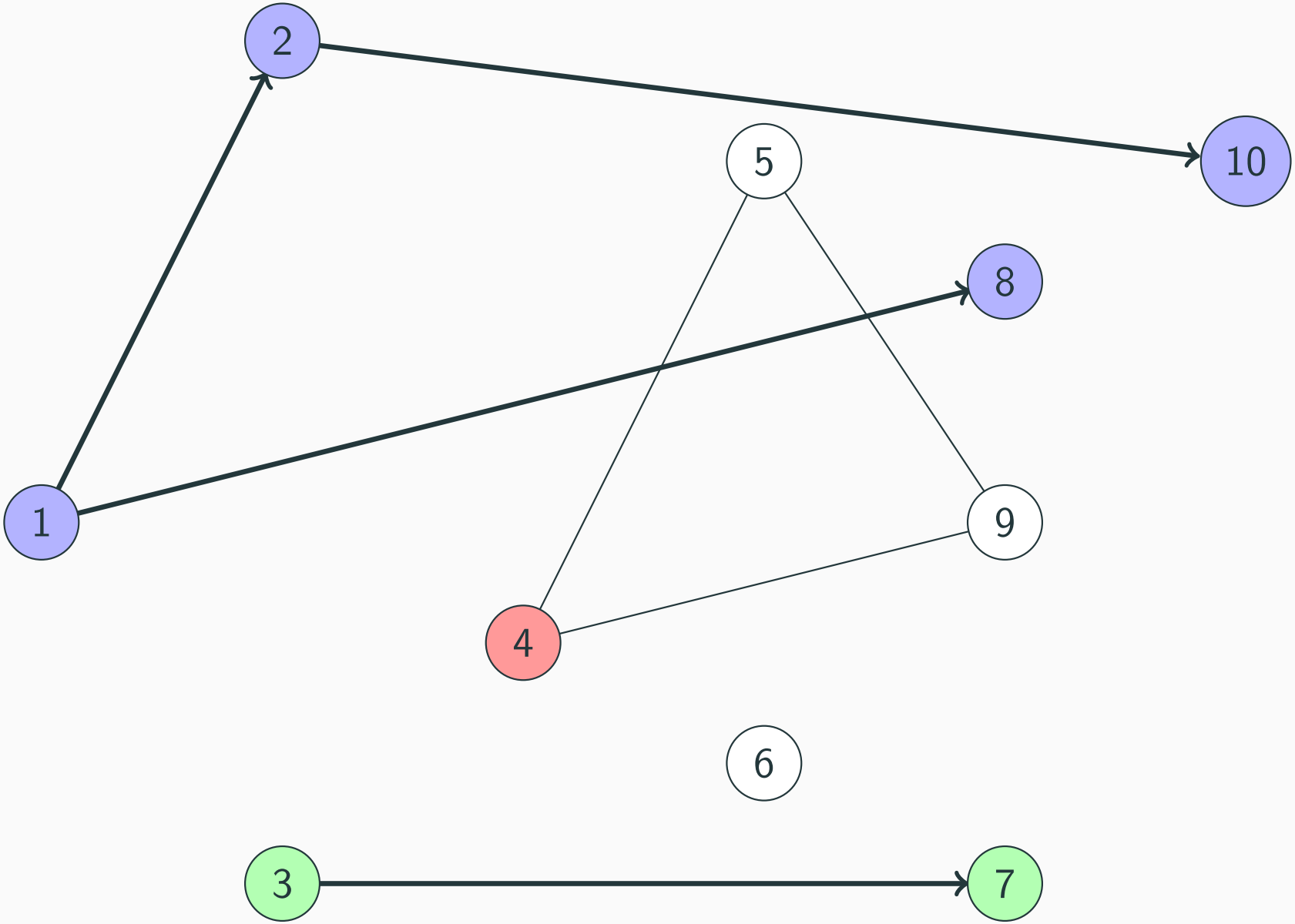
Visualização da identificação dos componentes conectados



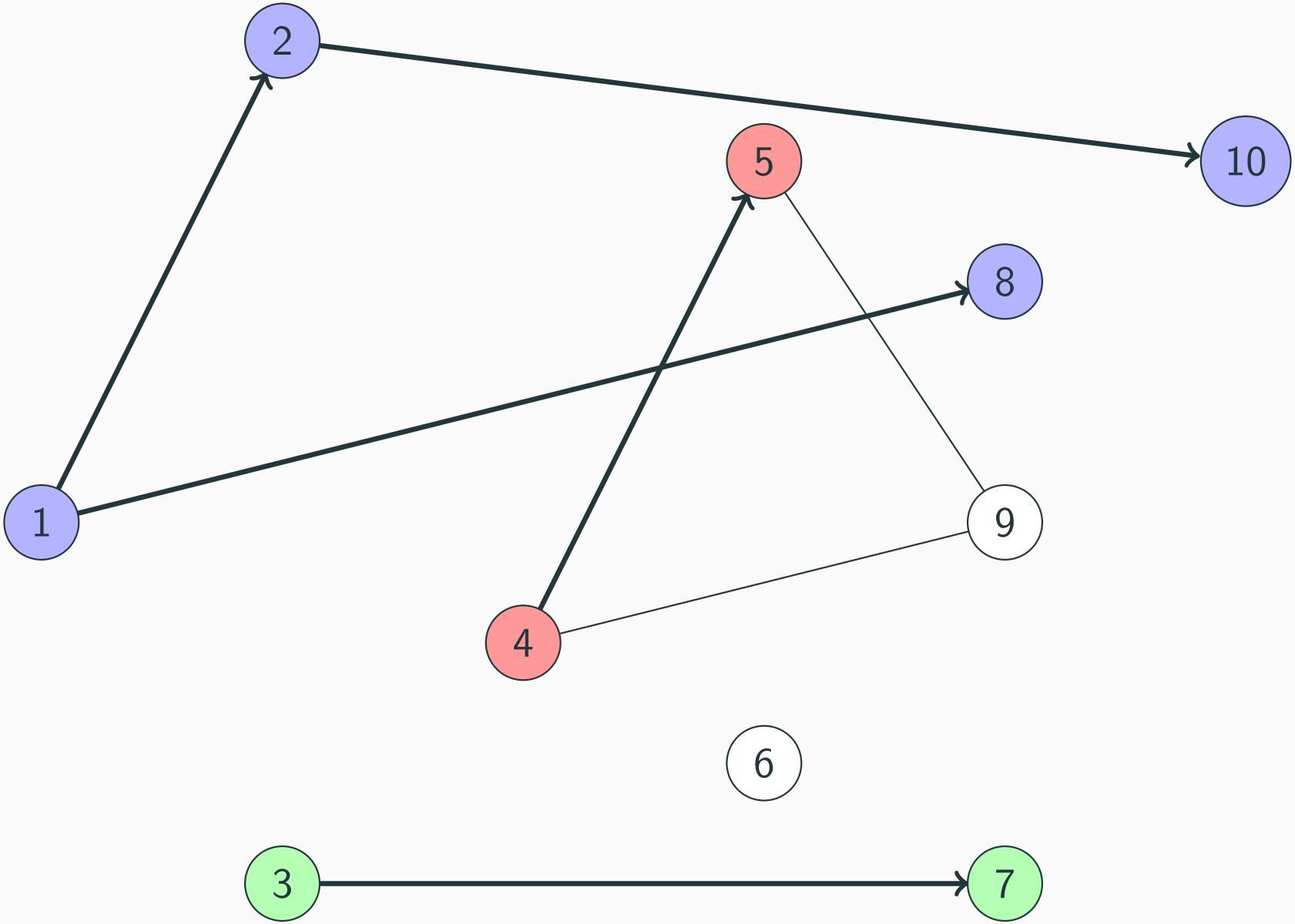
Visualização da identificação dos componentes conectados



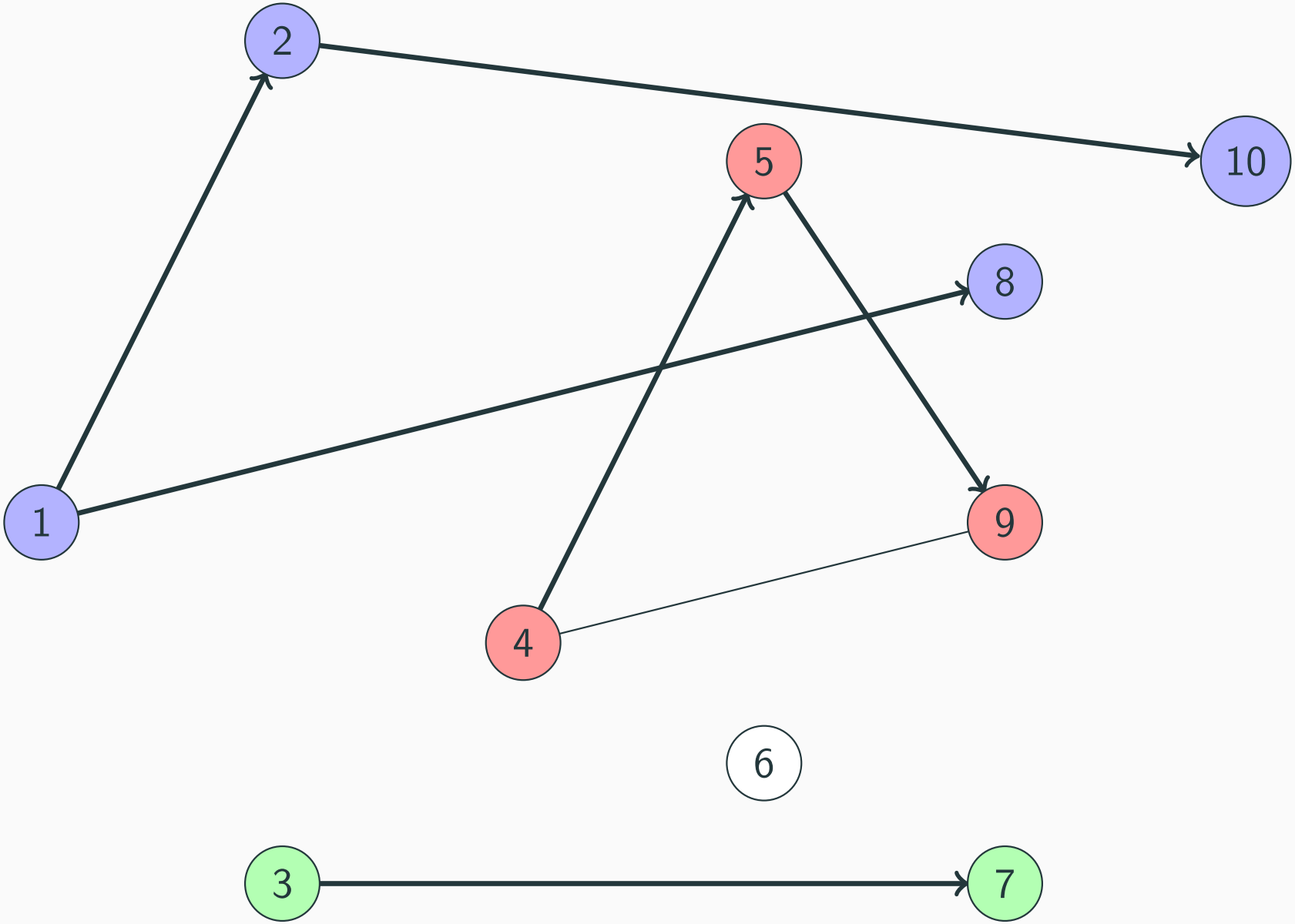
Visualização da identificação dos componentes conectados



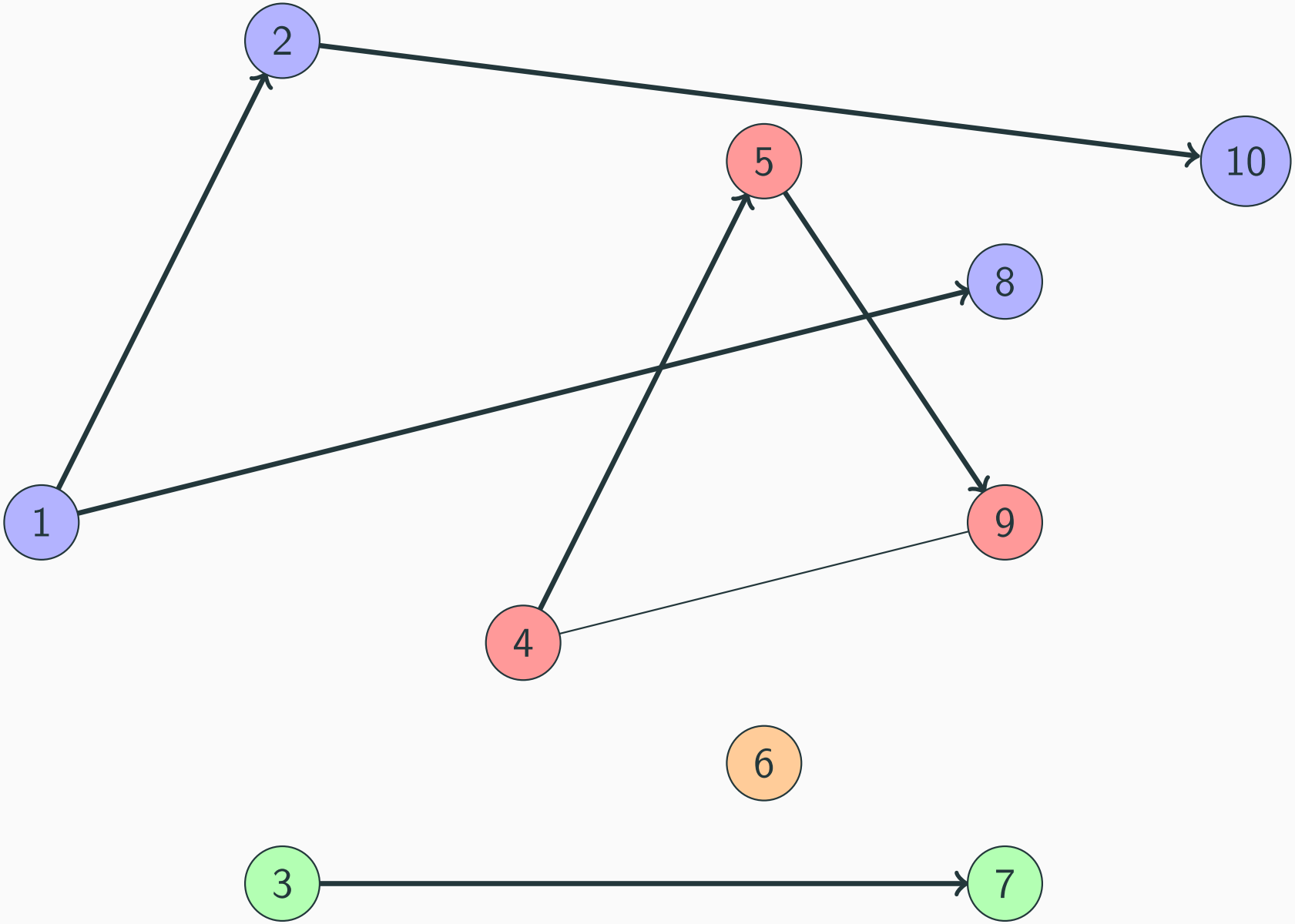
Visualização da identificação dos componentes conectados



Visualização da identificação dos componentes conectados



Visualização da identificação dos componentes conectados



Implementação da identificação dos componentes em C++

```cpp
#include <iostream>
#include <vector>
#include <bitset>

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

const int MAX { 100010 };
bitset<MAX> visited;
vector<int> adj[MAX];

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

    cout << " " << u;

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

Implementação da identificação dos componentes em C++

```
22
23 int connected_components(int N)
24 {
25     visited.reset();
26
27     int ans = 0;
28
29     for (int u = 1; u <= N; ++u)
30     {
31         if (not visited[u])
32         {
33             cout << "Component " << ++ans << ":";
34             dfs(u);
35             cout << endl;
36         }
37     }
38
39     return ans;
40 }
41
```

Implementação da identificação dos componentes em C++

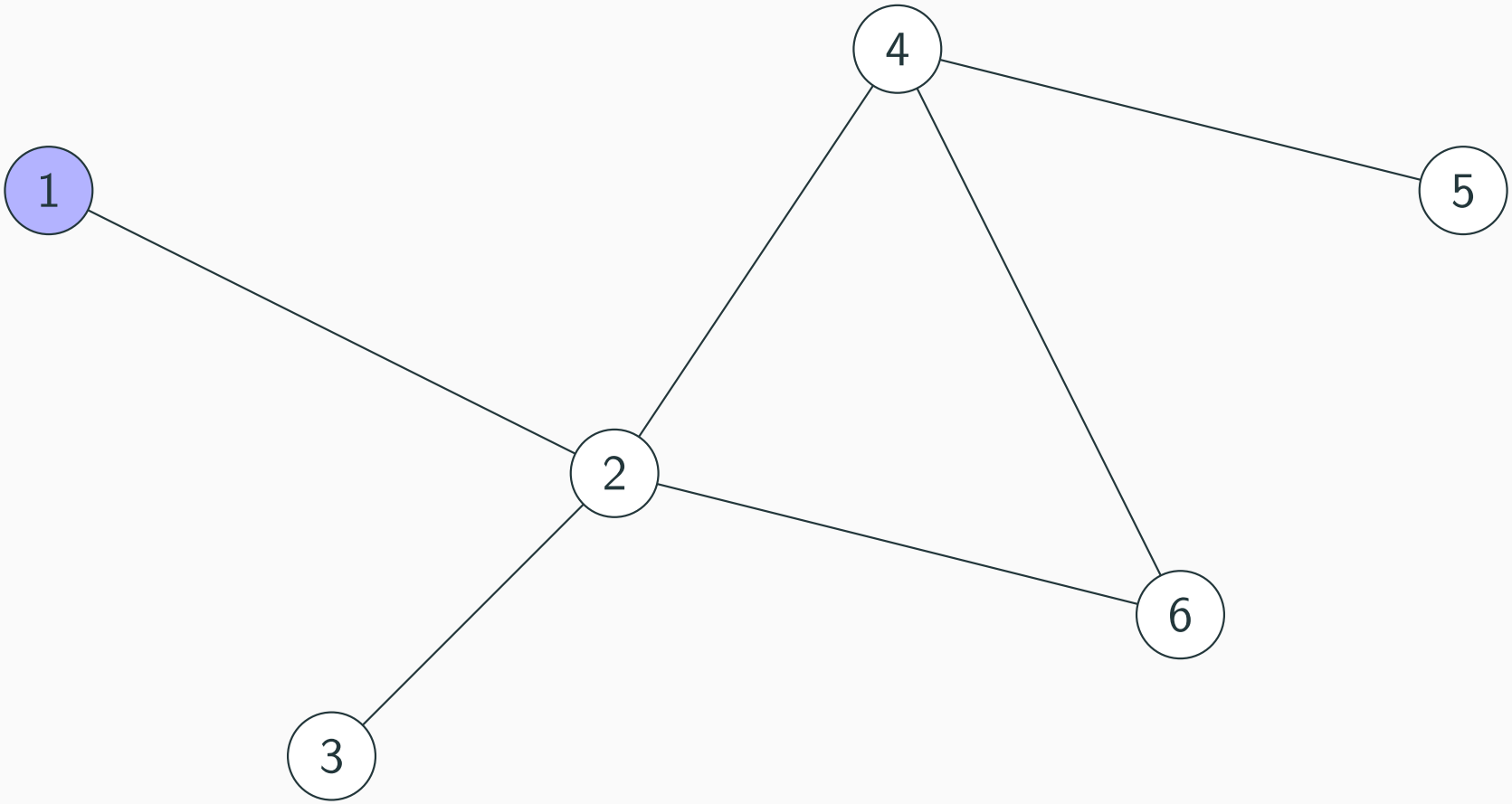
```
42 int main()
43 {
44     ii edges[] { ii(1, 2), ii(1, 8), ii(2, 10), ii(3, 7), ii(4, 5),
45         ii(4, 9), ii(5, 9) };
46 
47     for (const auto& [u, v] : edges)
48     {
49         adj[u].push_back(v);
50         adj[v].push_back(u);
51     }
52 
53     connected_components(10);
54 
55     return 0;
56 }
```

Detecção de ciclos

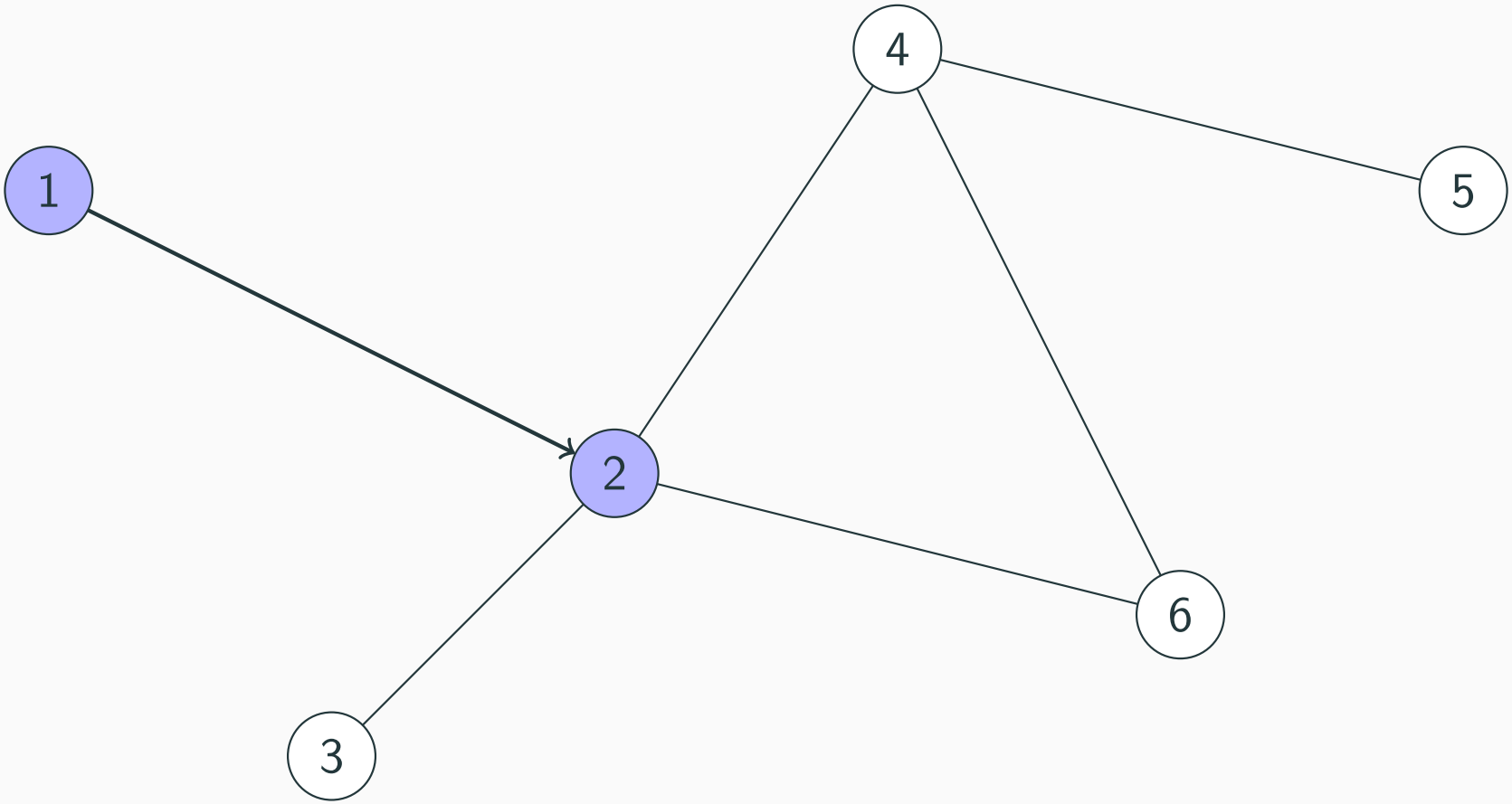
Detecção de ciclos

- Um ciclo é um caminho de tamanho maior ou igual a três cujos pontos de partida e chegada são iguais
- Um grafo que não contém nenhum ciclo é dito acíclico
- Uma travessia por profundidade pode ser utilizada para se determinar se um componente de um grafo é ou não acíclico
- Se, durante a travessia, um dos vizinhos do nó já foi visitado, e este vizinho não é o nó que o antecedeu na busca, então existe um ciclo começando e terminando no nó atual, e que passa por este vizinho
- Outra maneira de se detectar ciclos é contar o número de arestas E e vértices V do componente: se $E > V - 1$ então o componente tem um ciclo
- A complexidade da detecção de ciclos é a mesma da travessia: $O(N + M)$, onde N é o número de nós e M o número de arestas do grafo

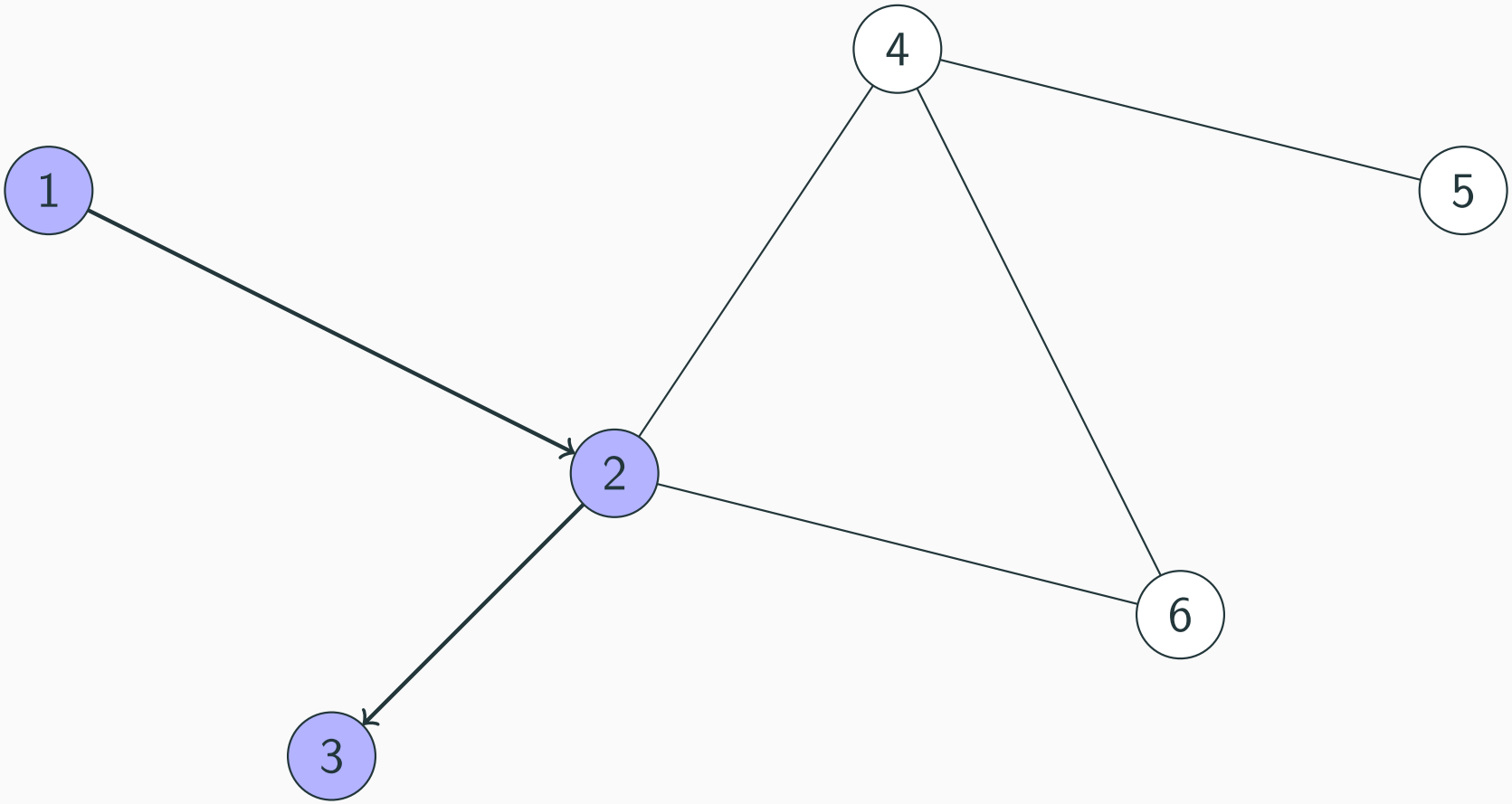
Visualização da identificação de ciclos



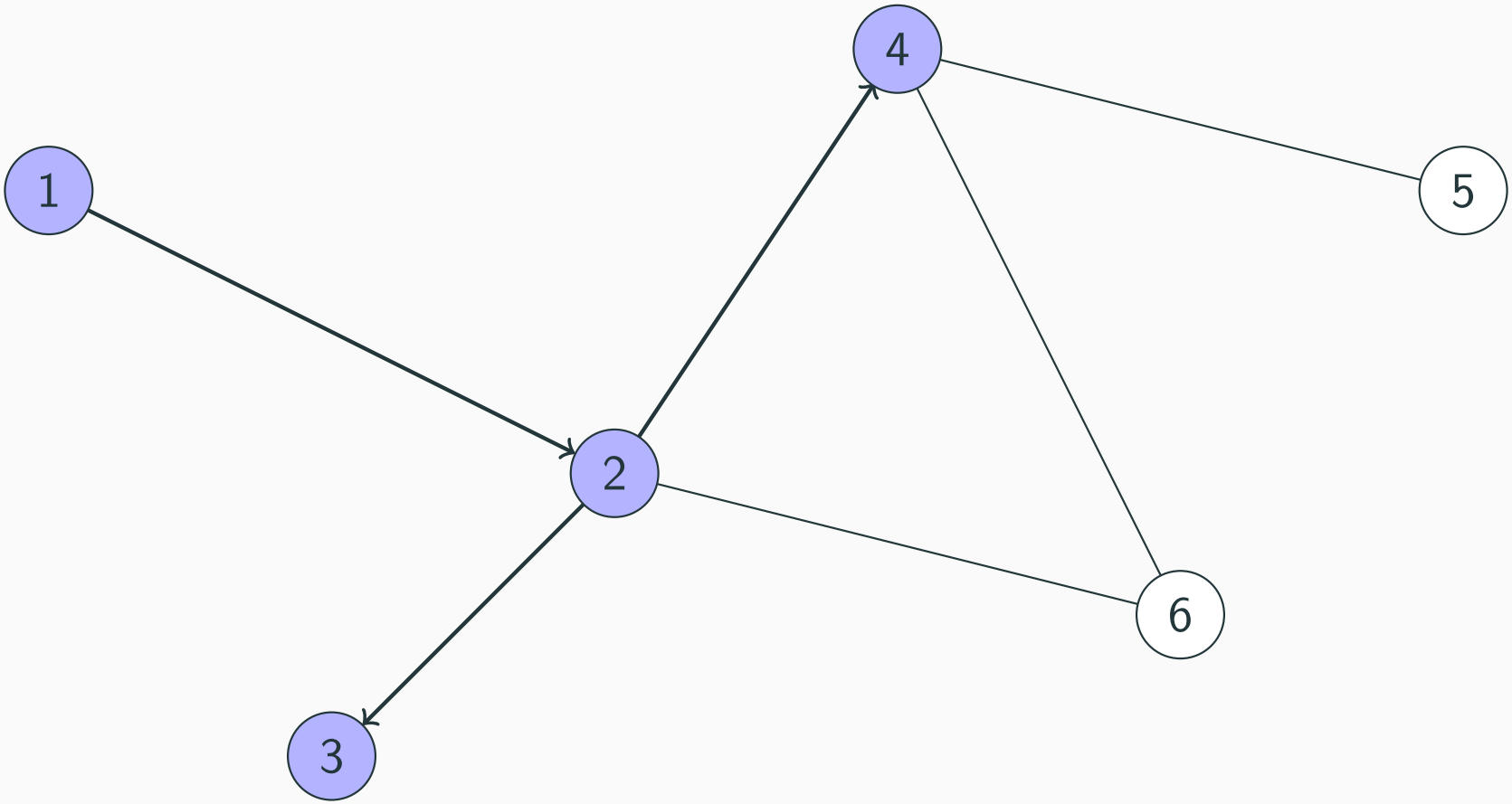
Visualização da identificação de ciclos



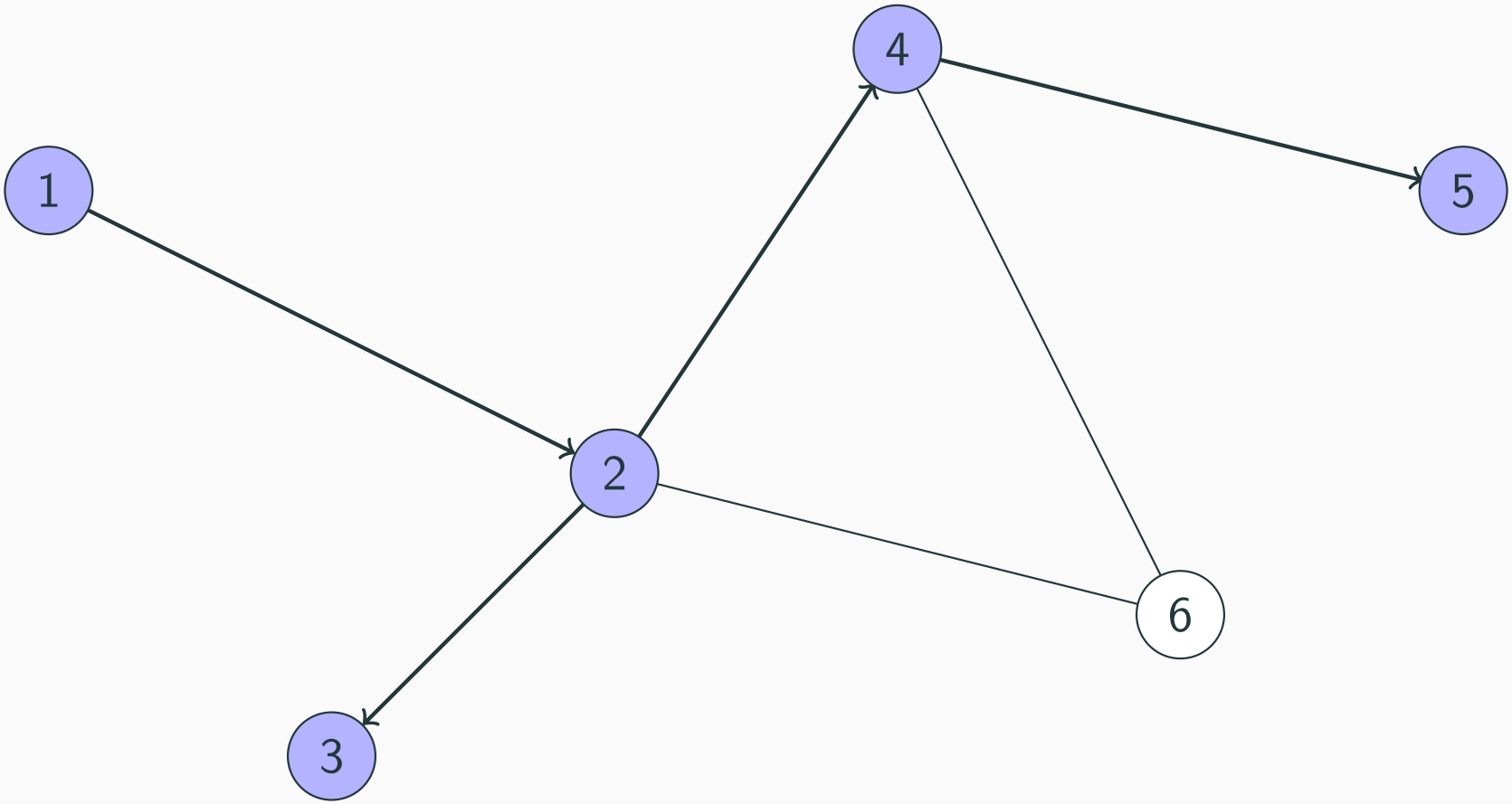
Visualização da identificação de ciclos



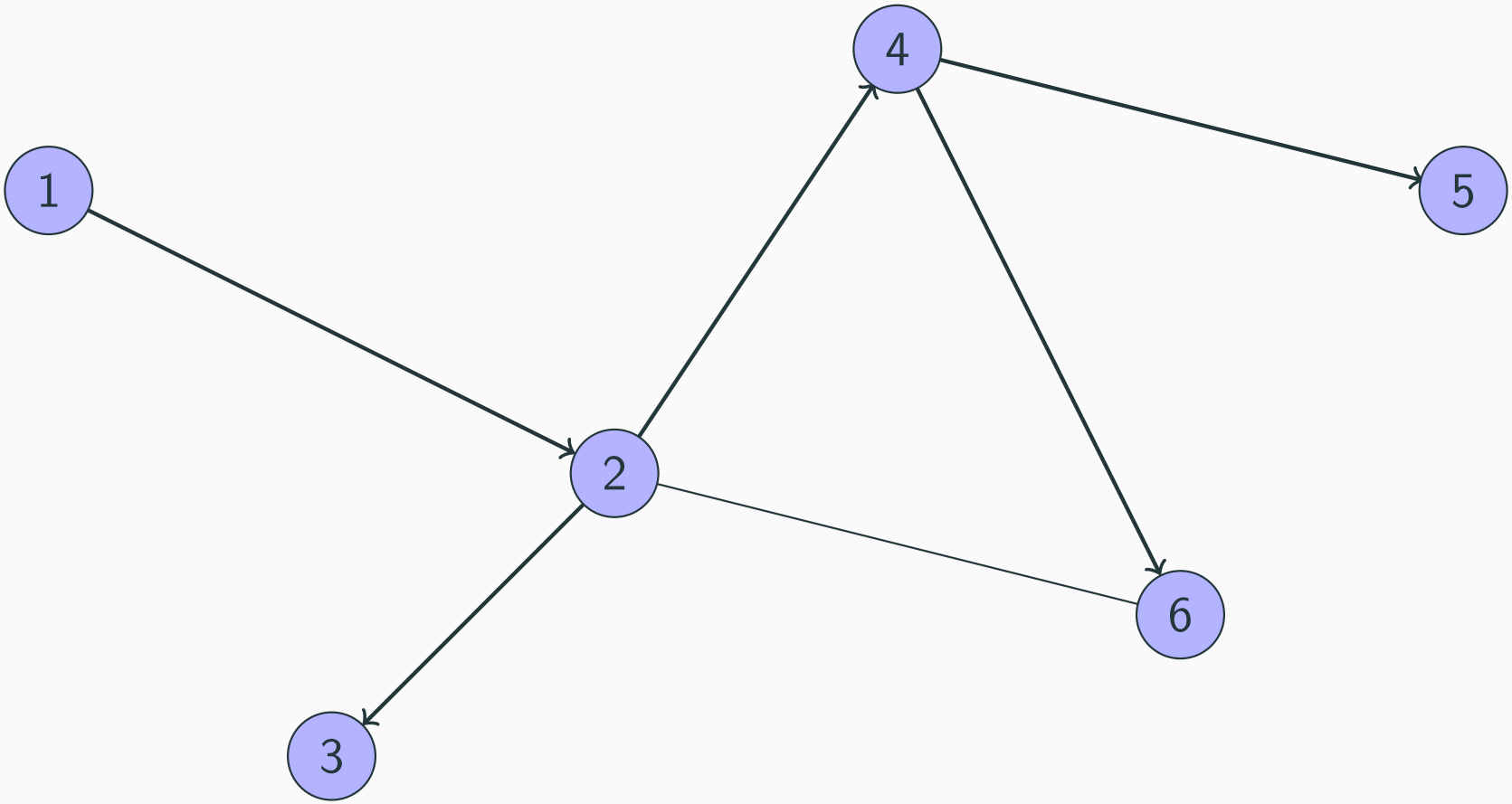
Visualização da identificação de ciclos



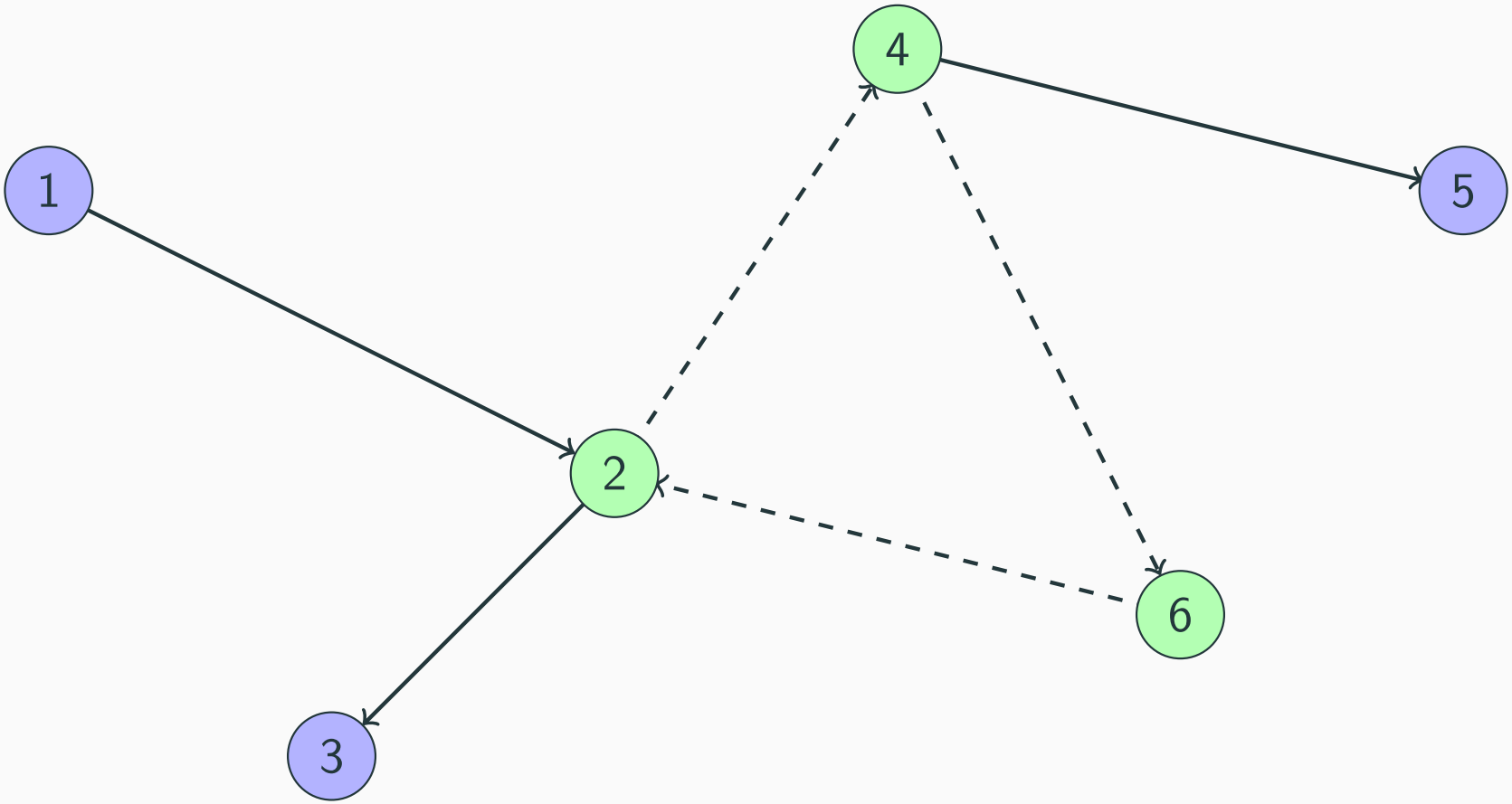
Visualização da identificação de ciclos



Visualização da identificação de ciclos



Visualização da identificação de ciclos



Exemplo de detecção de ciclo

```cpp
#include <iostream>
#include <cstring>
#include <bitset>
#include <vector>

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

const int MAX { 100010 };
bitset<MAX> visited;
vector<int> adj[MAX];
int parent[MAX];

bool dfs(int u)
{
    if (visited[u]) return false;
    visited[u] = true;

```

Exemplo de detecção de ciclo

```
19      for (const auto& v : adj[u])
20      {
21          parent[v] = parent[v] ? parent[v] : u;
22
23          if (visited[v] and parent[u] != v)
24              return true;
25          else
26              if (dfs(v)) return true;
27      }
28
29      return false;
30  }
31
32  bool has_cycle(int N)
33  {
34      visited.reset();
35      memset(parent, 0, sizeof parent);
36
37      for (int u = 1; u <= N; ++u)
38          if (not visited[u] and dfs(u))
39              return true;
```

Exemplo de detecção de ciclo

```
40
41      return false;
42  }
43
44  int main()
45  {
46      ii edges[] { ii(1,2), ii(2,3), ii(2,4), ii(2,6), ii(4,5), ii(4,6) };
47
48      for (const auto& [u, v] : edges)
49      {
50          adj[u].push_back(v);
51          adj[v].push_back(u);
52      }
53
54      cout << "Tem ciclo? " << (has_cycle(6) ? "Sim" : "Nao") << endl;
55
56      return 0;
57  }
```

Outro exemplo de detecção de ciclo

```cpp
#include <iostream>
#include <functional>
#include <vector>
#include <bitset>

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

const int MAX { 100010 };
bitset<MAX> visited;
vector<int> adj[MAX];

void dfs(int u, function<void(int)> process) {
    if (visited[u]) return;
    visited[u] = true;

    process(u);

    for (const auto& v : adj[u])
        dfs(v, process);
}
```

Outro exemplo de detecção de ciclo

```
23 bool has_cycle(int N) {
24     visited.reset();
25 
26     for (int u = 1; u <= N; ++u)
27         if (not visited[u])
28         {
29             vector<int> cs;
30             size_t edges = 0;
31 
32             dfs(u, [&](int u) {
33                 cs.push_back(u);
34 
35                 for (const auto& v : adj[u])
36                     edges += (visited[v] ? 0 : 1);
37             });
38 
39             if (edges >= cs.size()) return true;
40         }
41 
42     return false;
43 }
```

Outro exemplo de detecção de ciclo

```
45 int main()
46 {
47     ii edges[] { ii(1,2), ii(2,3), ii(2,4), ii(2,6), ii(4,5), ii(4,6) };
48 
49     for (const auto& [u, v] : edges)
50     {
51         adj[u].push_back(v);
52         adj[v].push_back(u);
53     }
54 
55     cout << "Tem ciclo? " << (has_cycle(6) ? "Sim" : "Nao") << endl;
56 
57     return 0;
58 }
```

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.