Travessia de Grafos

Aplicações

Prof. Edson Alves

2019

Faculdade UnB Gama

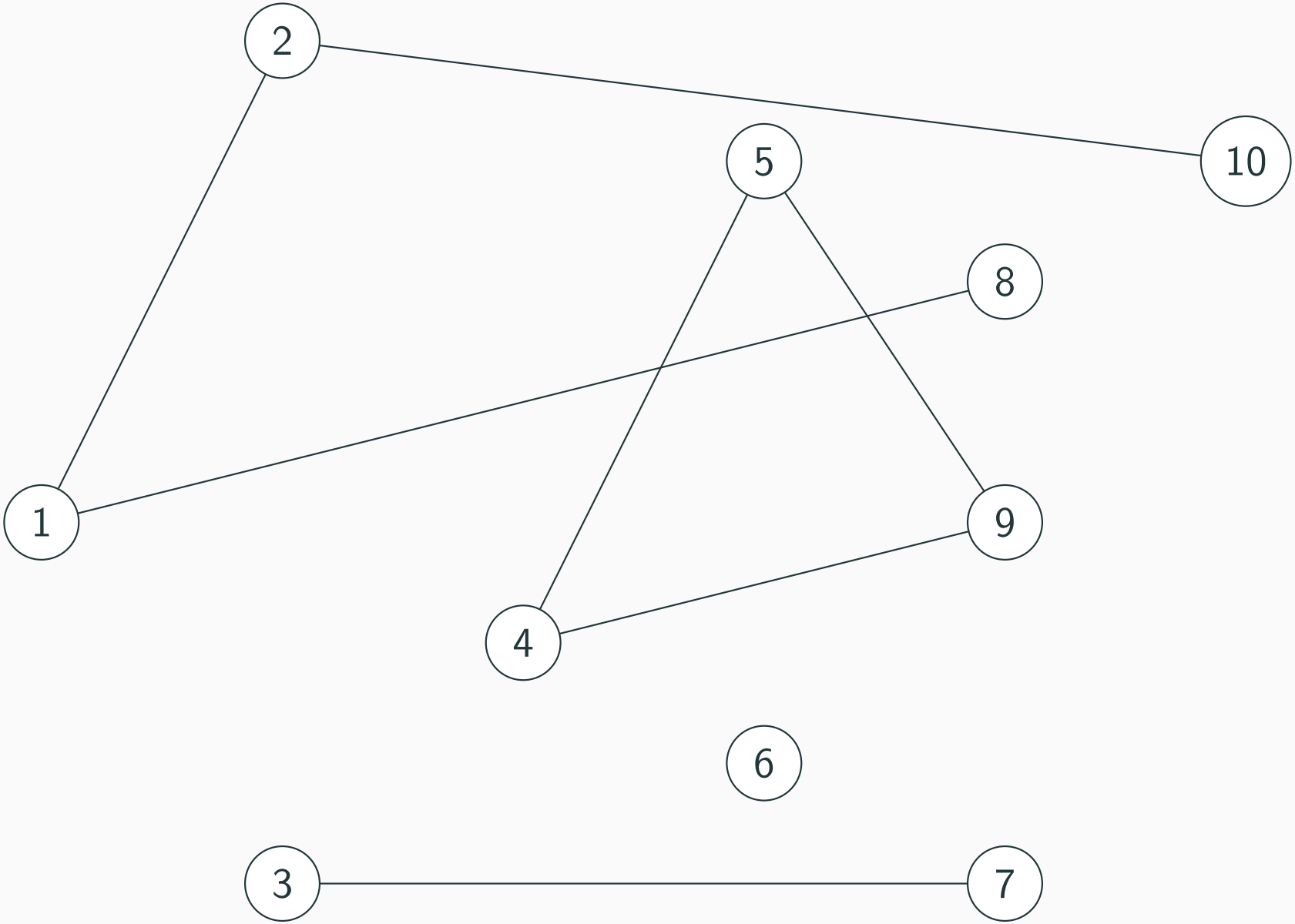
Sumário

Componentes Conectados

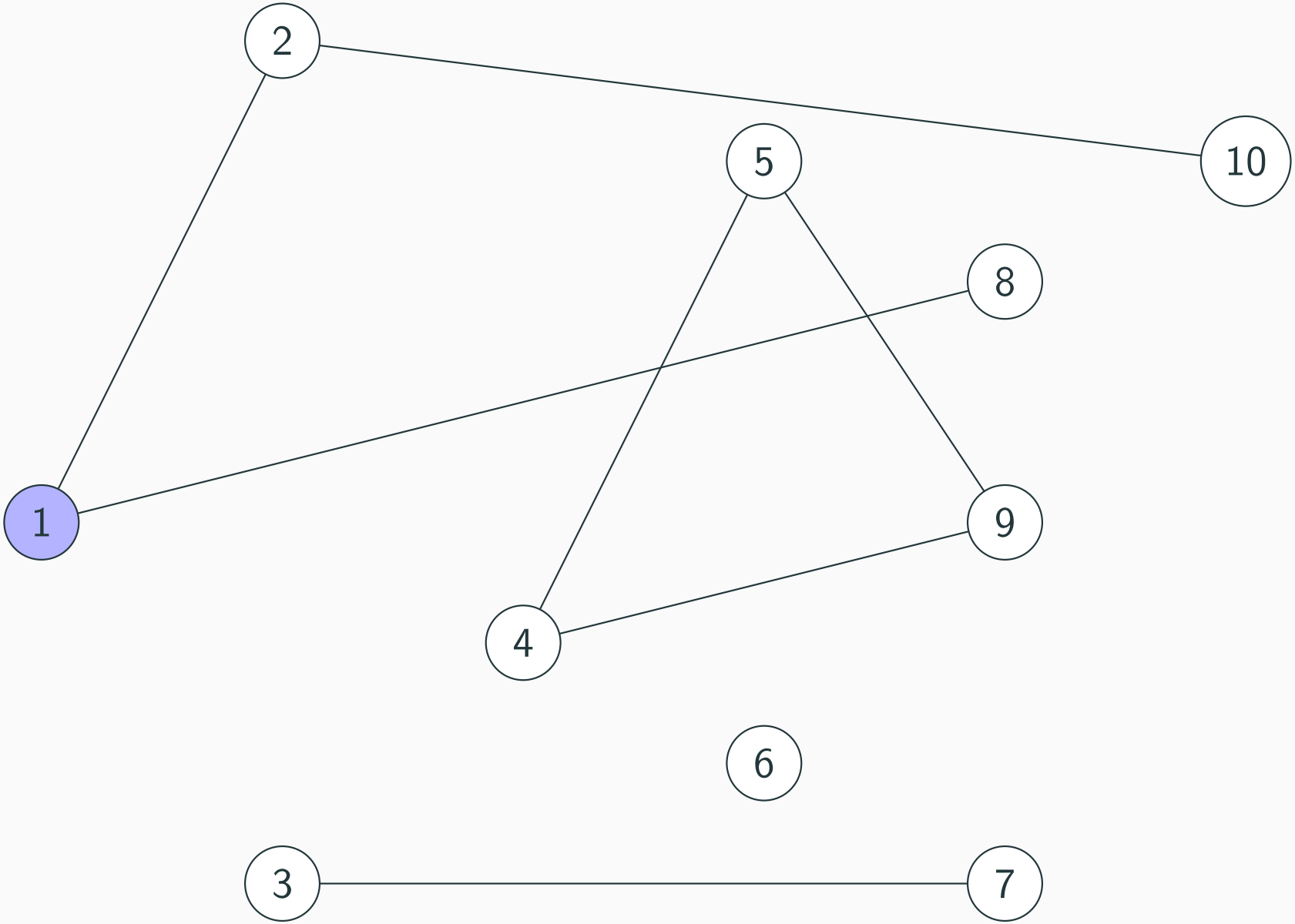
Conectividade de um grafo

- Um grafo simples não-direcionado 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 sejam visitados, 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

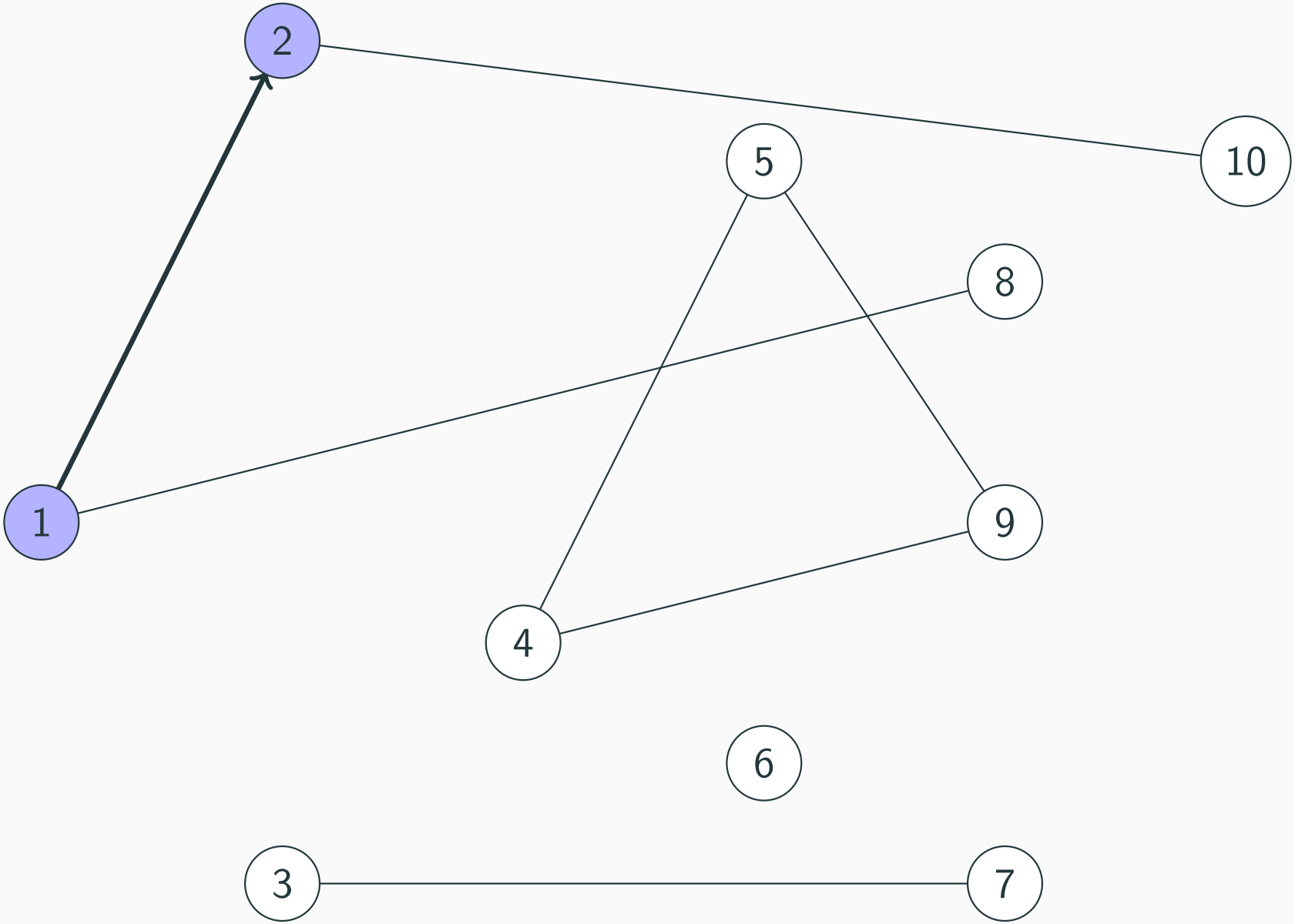
Visualização da identificação dos componentes conectados



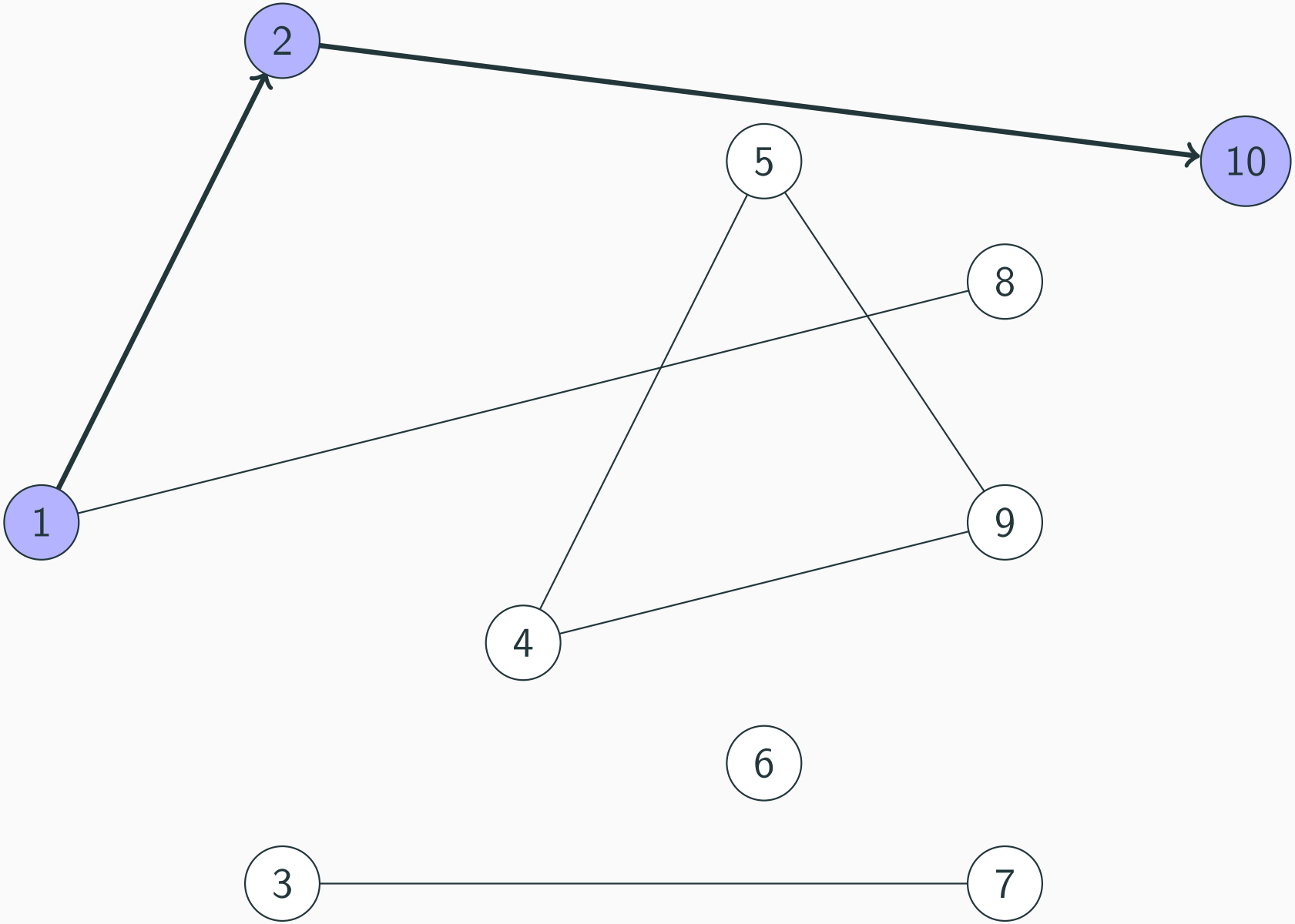
Visualização da identificação dos componentes conectados



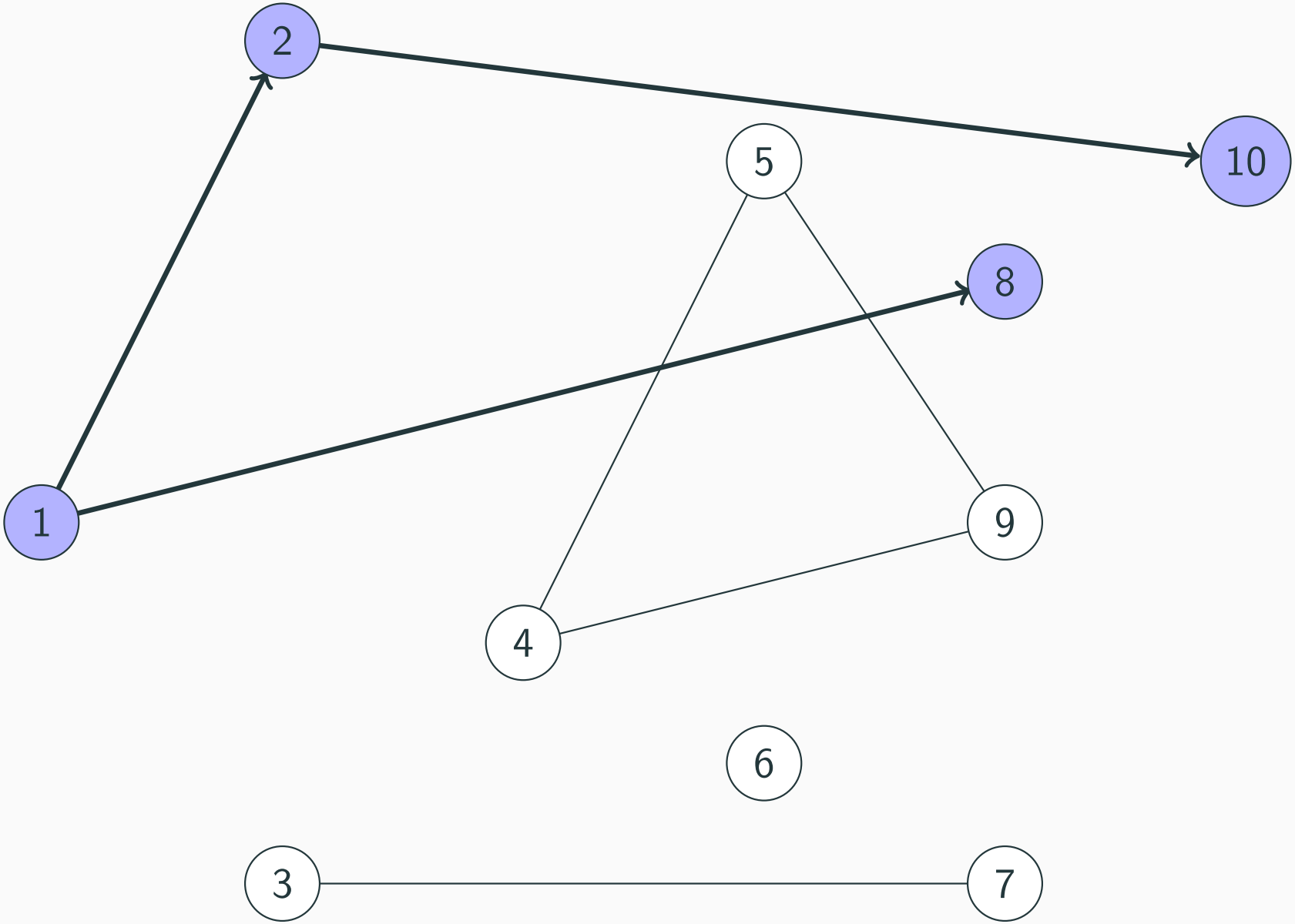
Visualização da identificação dos componentes conectados



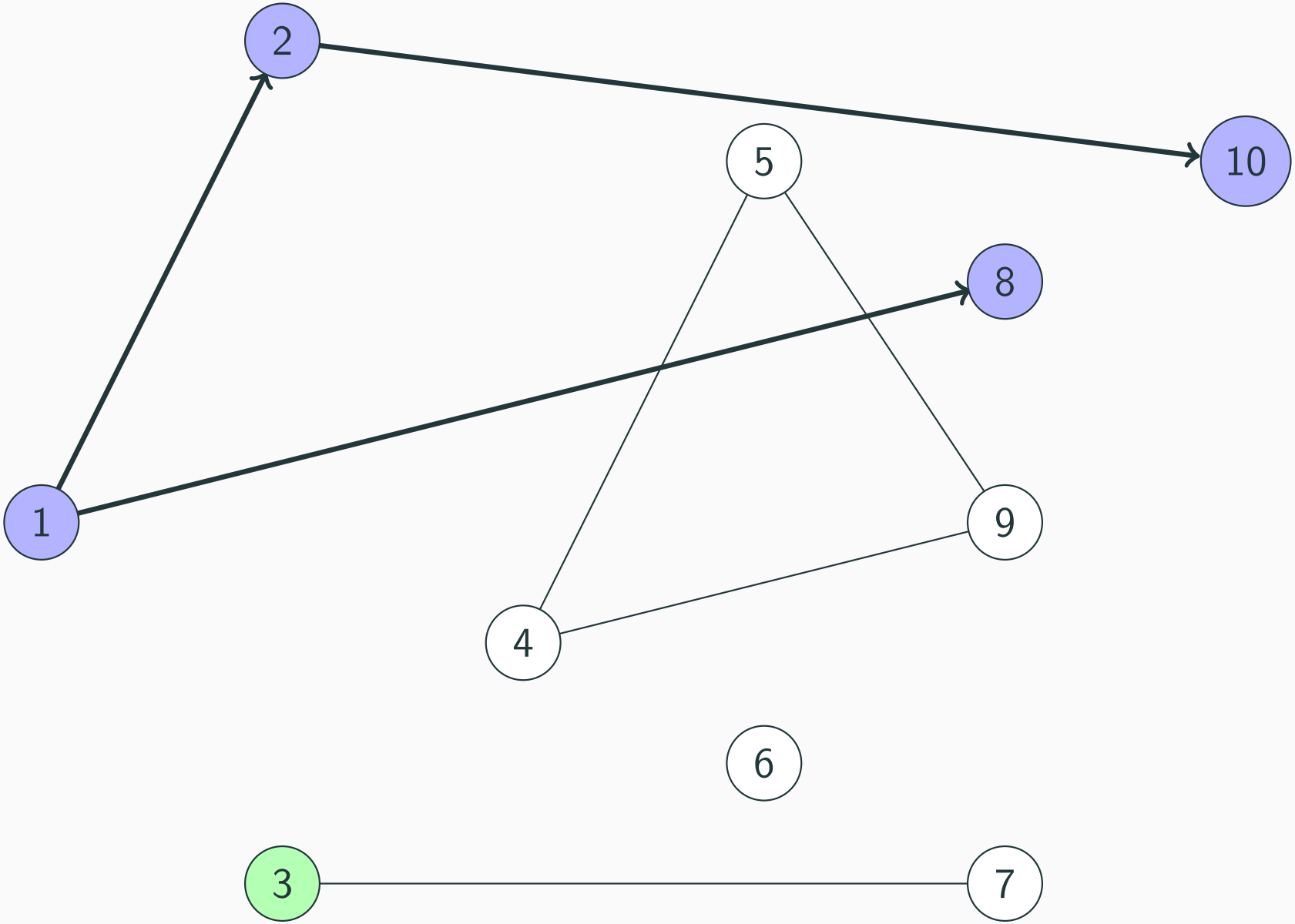
Visualização da identificação dos componentes conectados



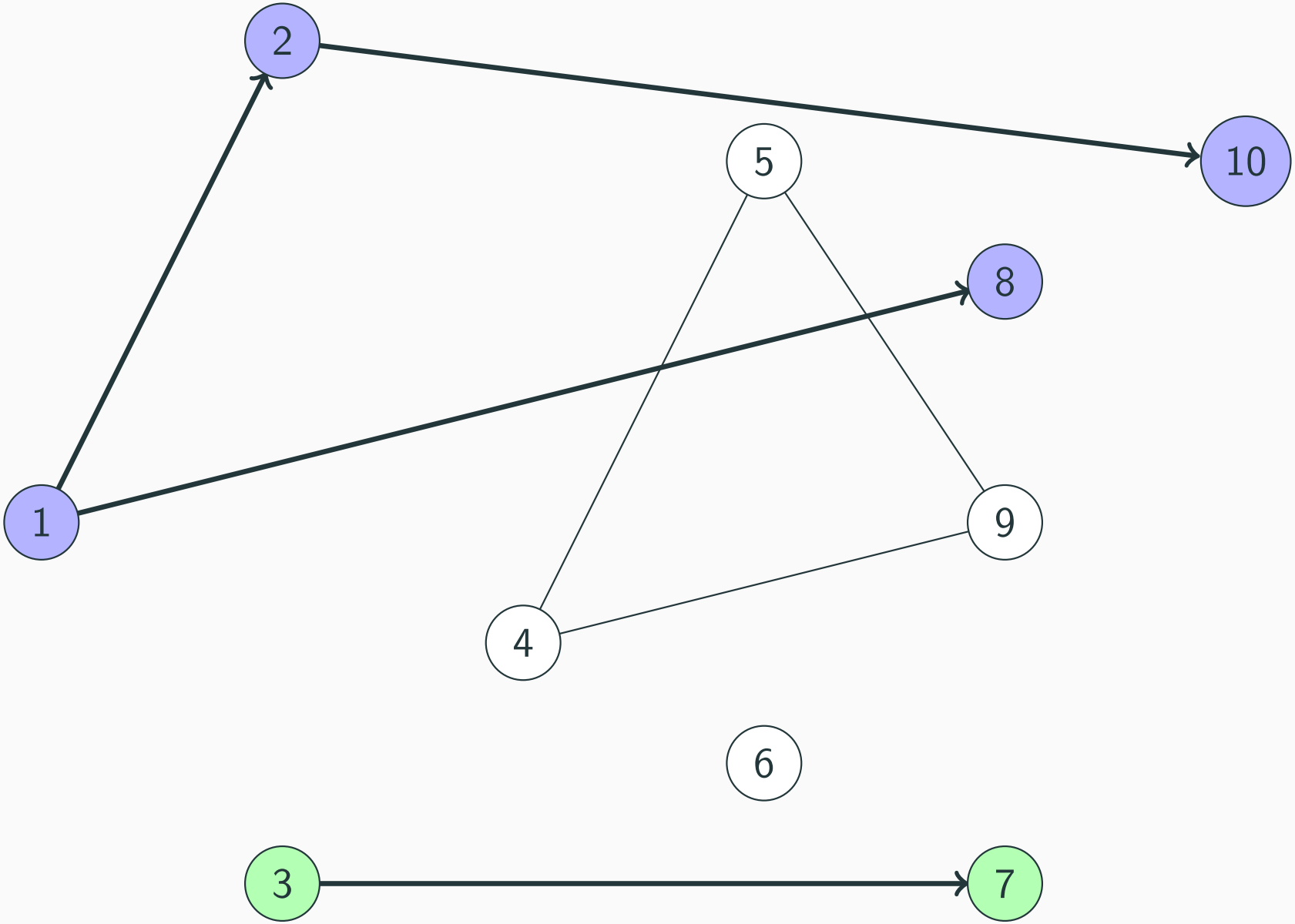
Visualização da identificação dos componentes conectados



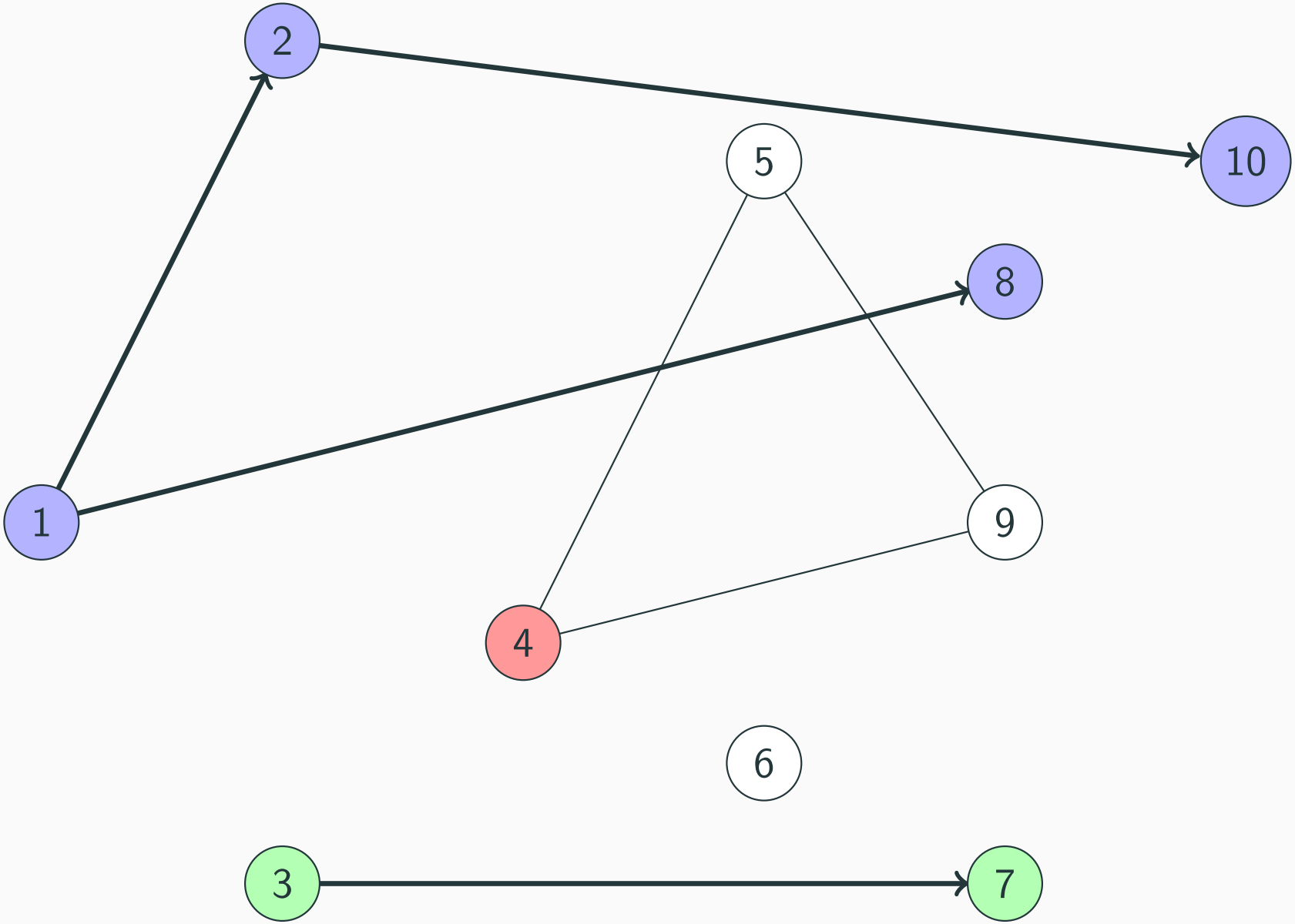
Visualização da identificação dos componentes conectados



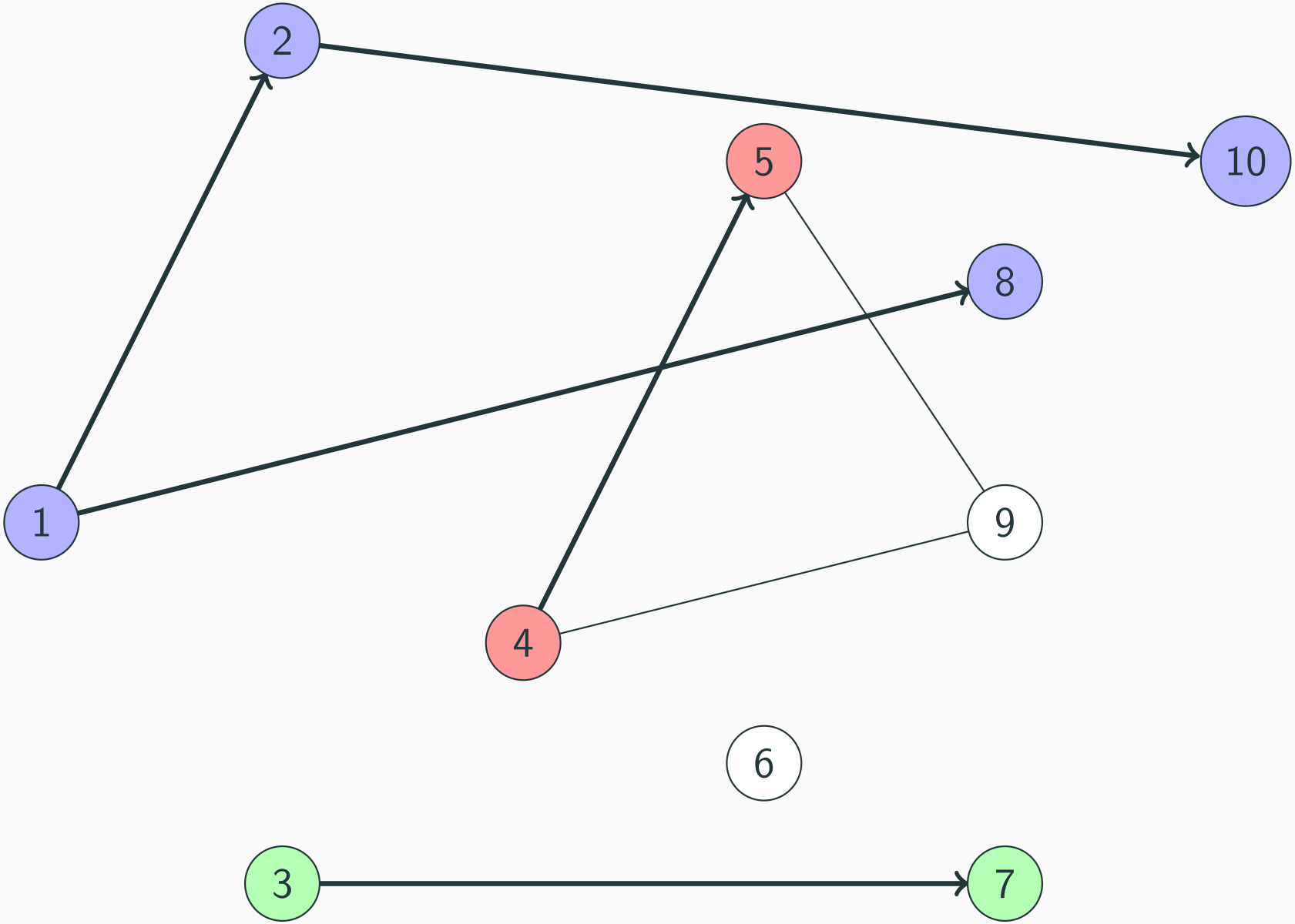
Visualização da identificação dos componentes conectados



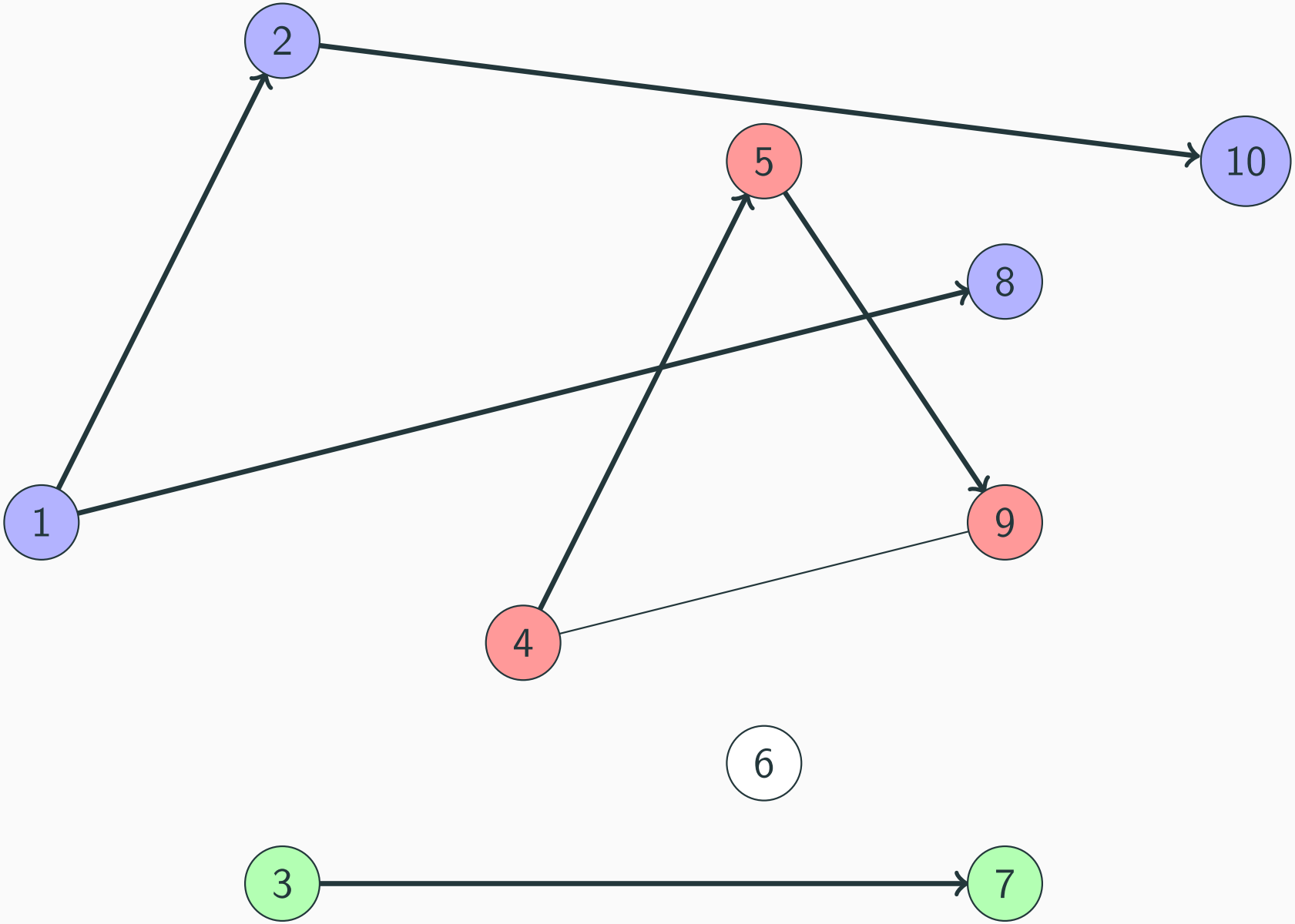
Visualização da identificação dos componentes conectados



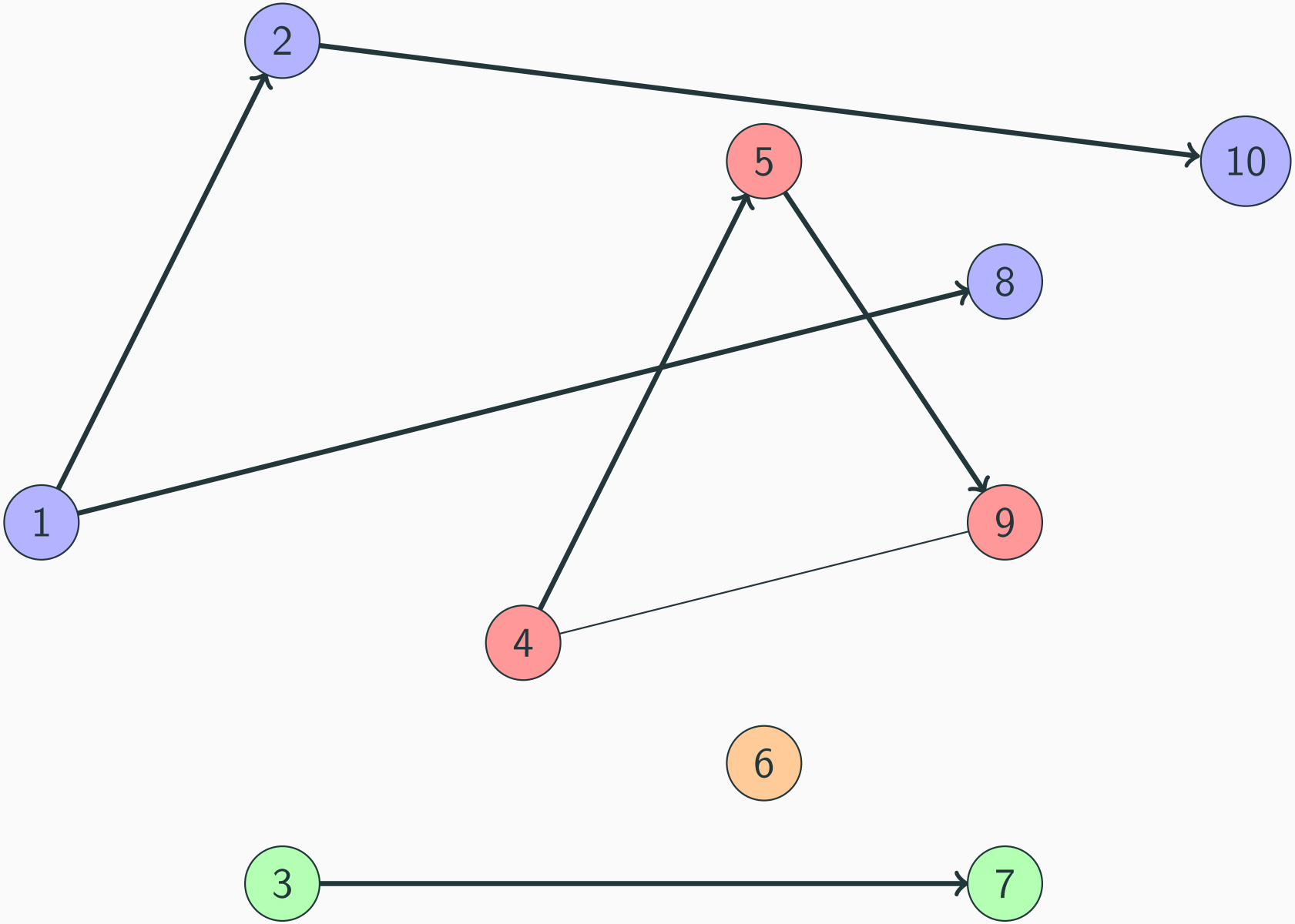
Visualização da identificação dos componentes conectados



Visualização da identificação dos componentes conectados



Visualização da identificação dos componentes conectados



Implementação da identificação dos componentes em C++

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

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

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

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

    process(u);

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

```

Implementação da identificação dos componentes em C++

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

Implementação da identificação dos componentes em C++

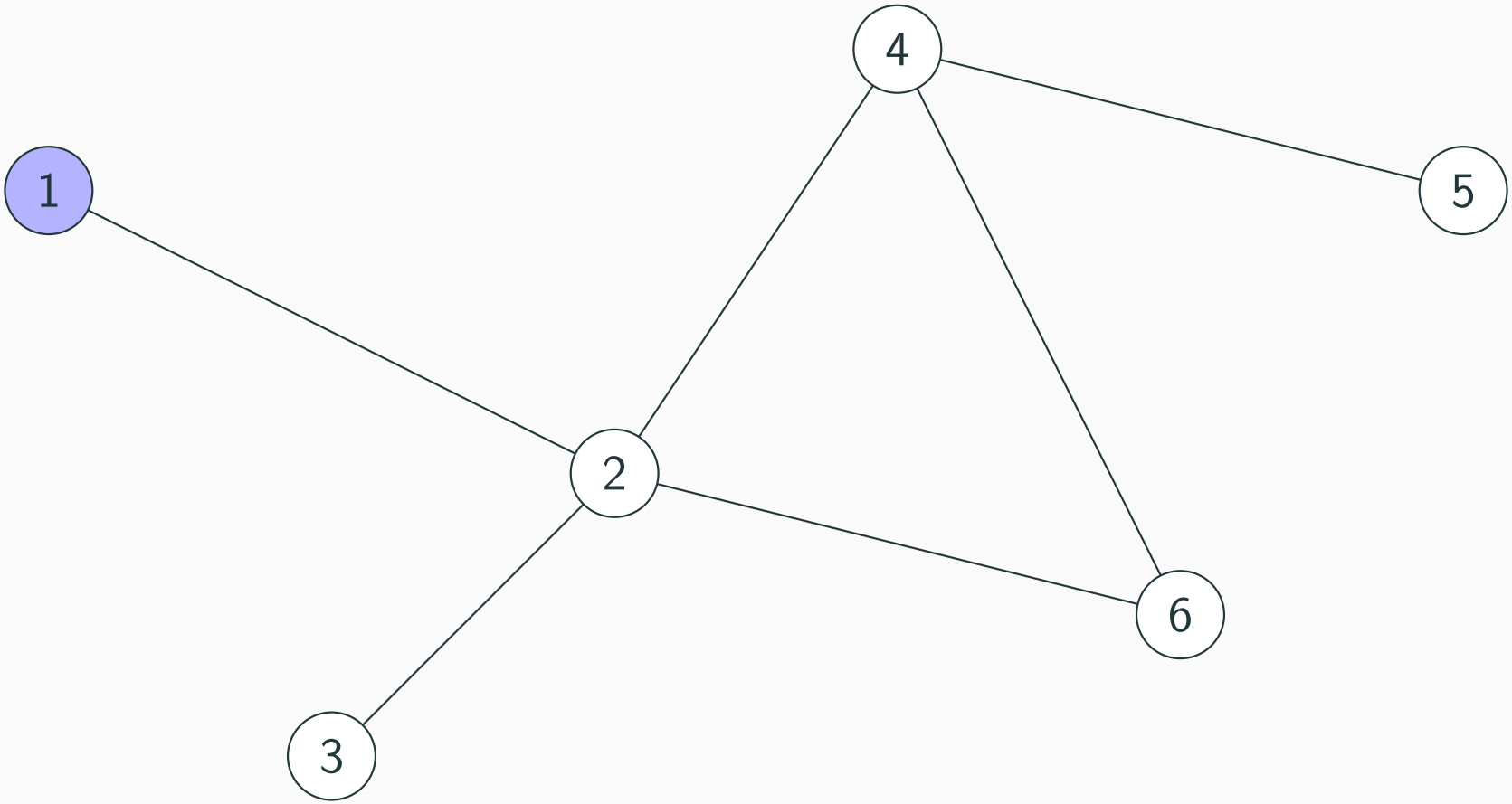
```
40 int main()
41 {
42     vector<ii> edges { ii(1, 2), ii(1, 8), ii(2, 10), ii(3, 7), ii(4, 5),
43         ii(4, 9), ii(5, 9) };
44
45     for (const auto& [u, v] : edges)
46     {
47         adj[u].push_back(v);
48         adj[v].push_back(u);
49     }
50
51     connected_components(10);
52
53     return 0;
54 }
```

Detecção de ciclos

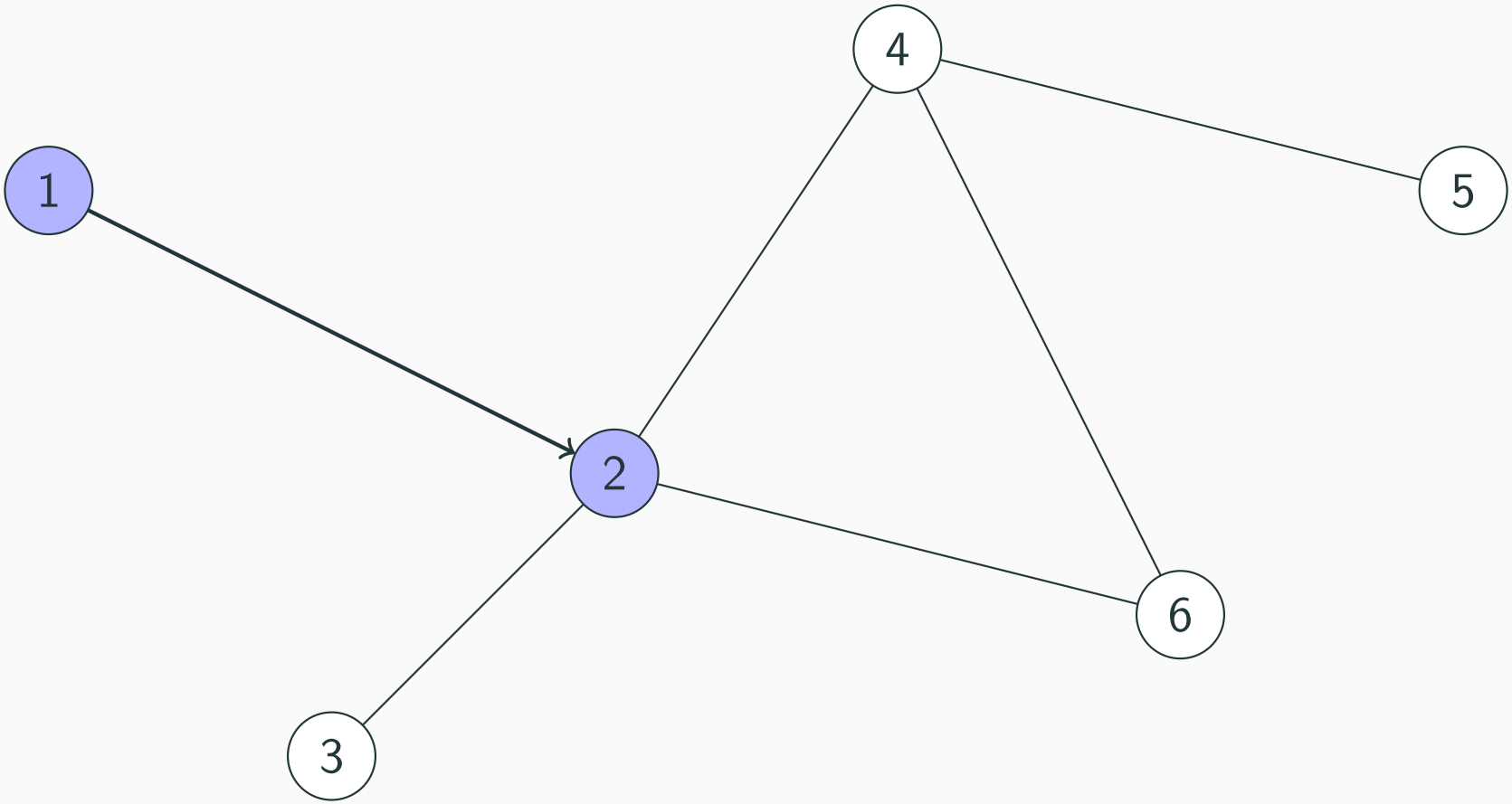
Detecção de ciclos

- 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

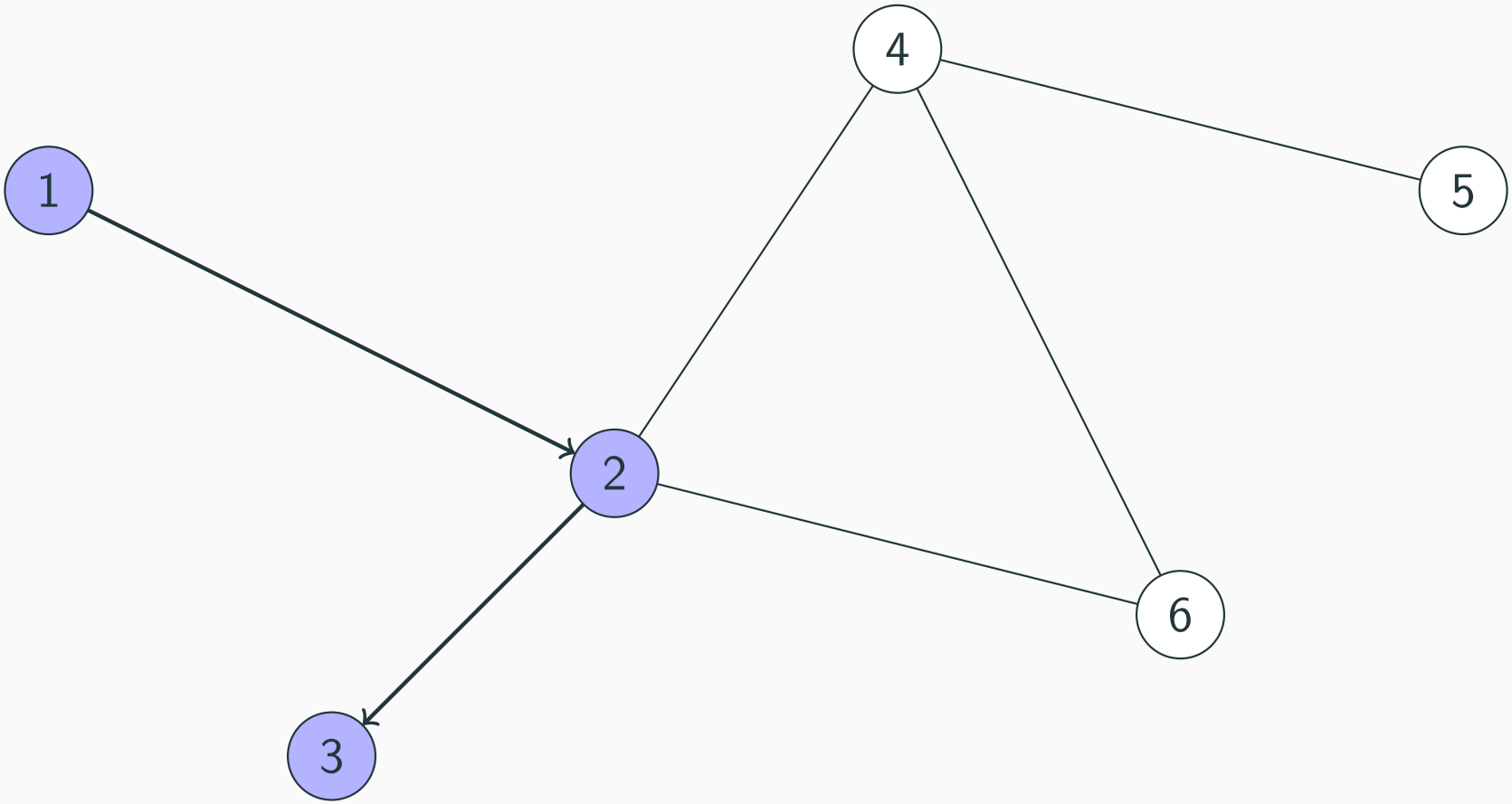
Visualização da identificação de ciclos



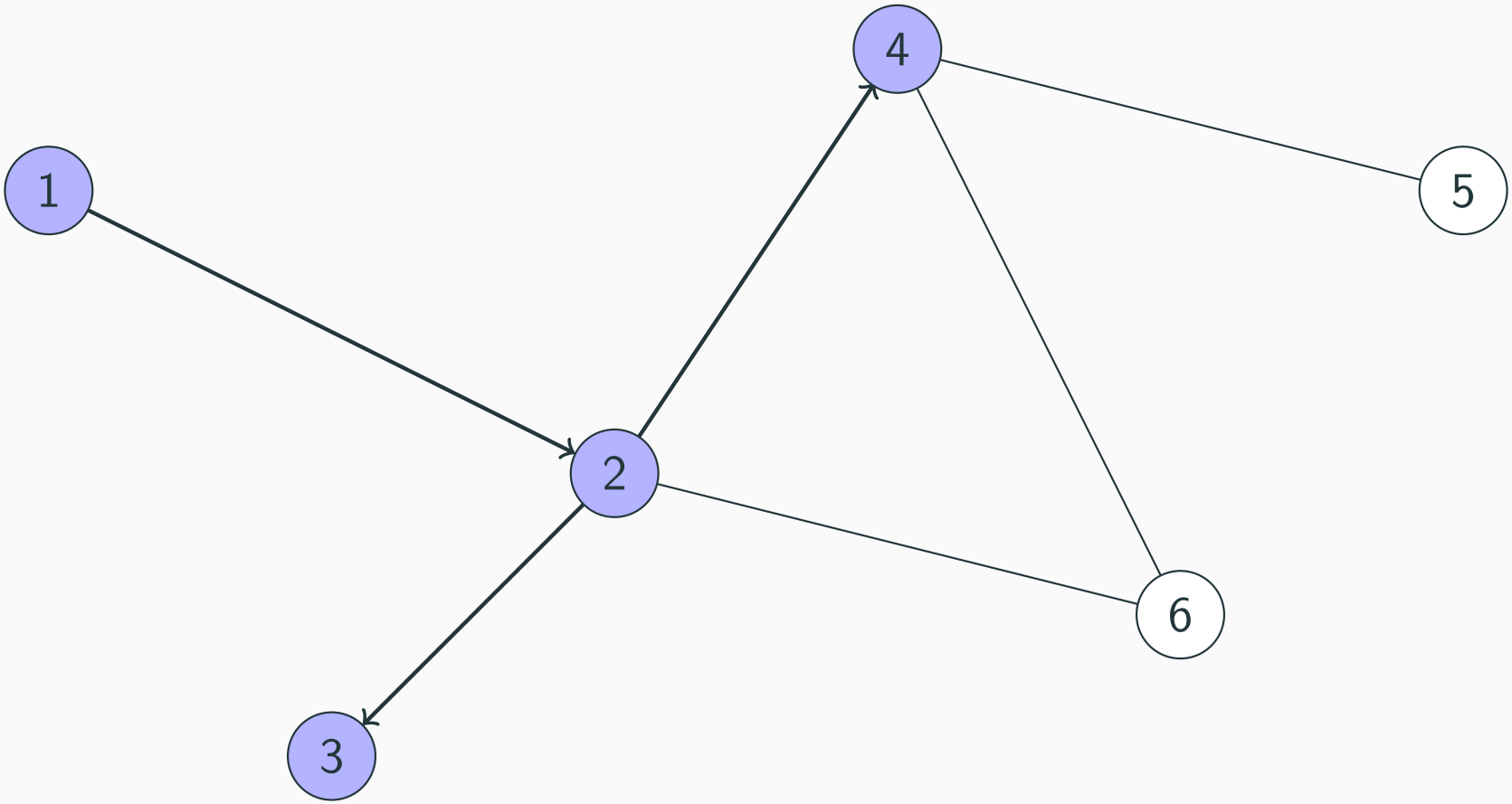
Visualização da identificação de ciclos



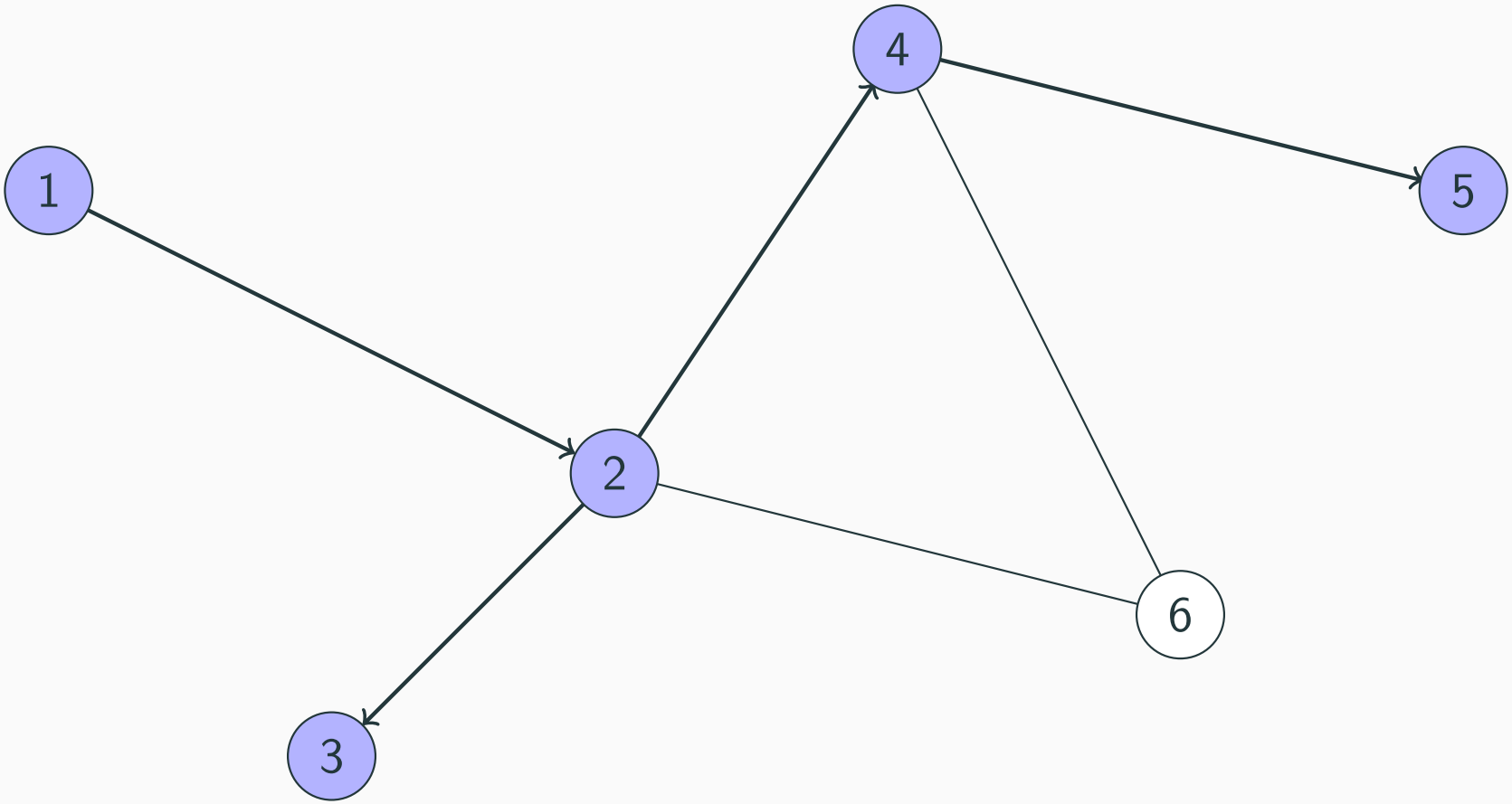
Visualização da identificação de ciclos



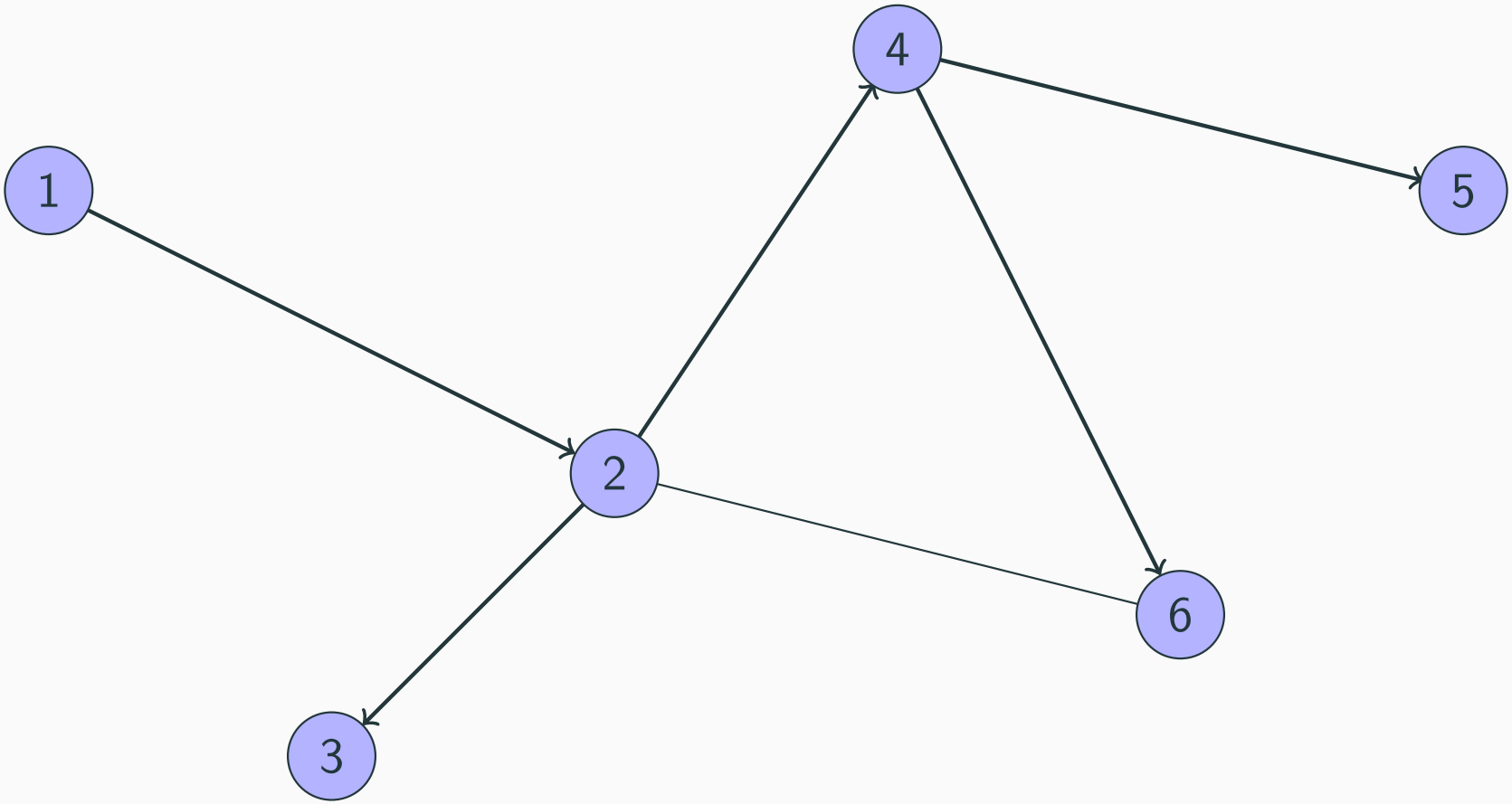
Visualização da identificação de ciclos



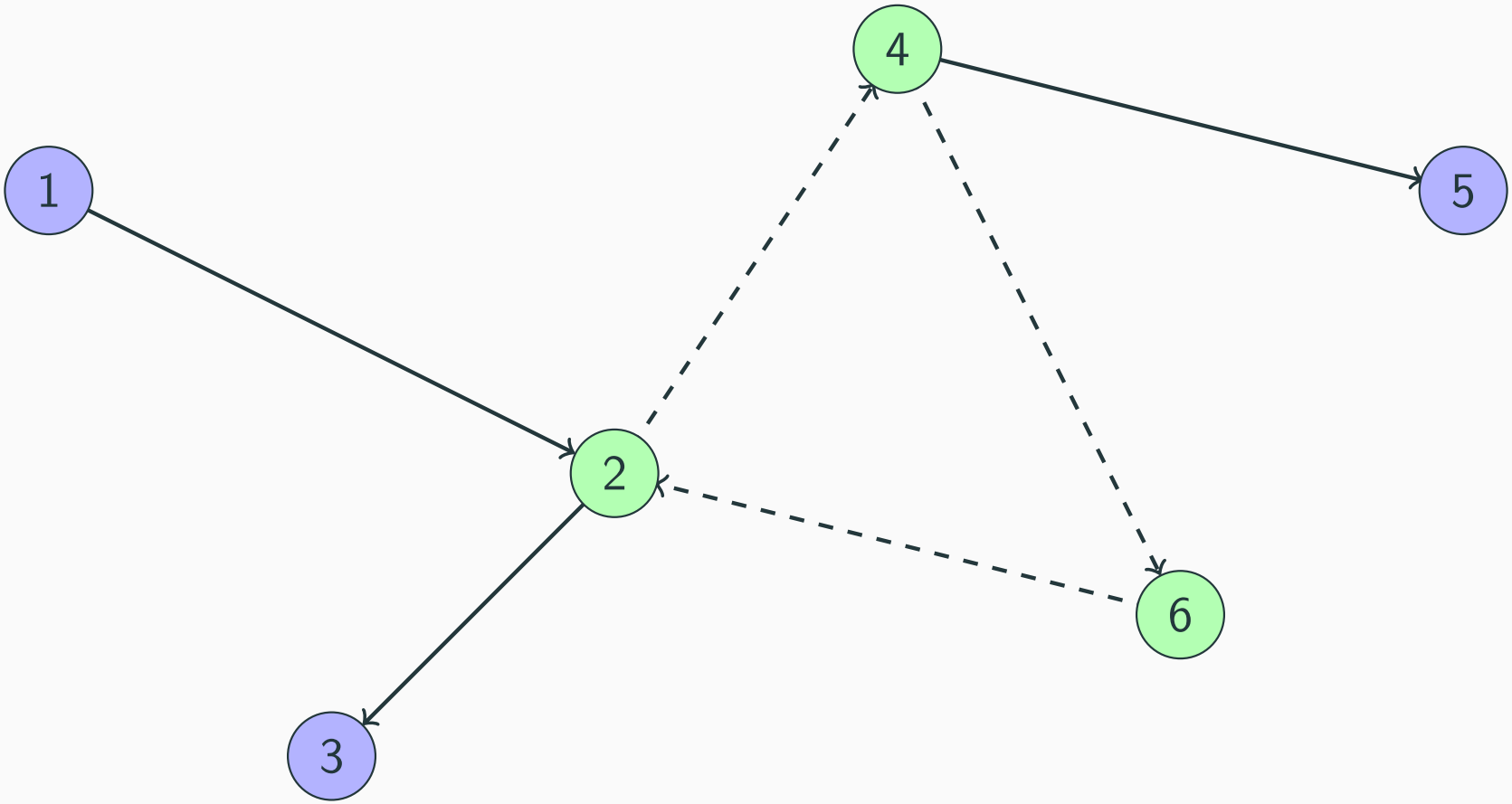
Visualização da identificação de ciclos



Visualização da identificação de ciclos



Visualização da identificação de ciclos



Exemplo de detecção de ciclo

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

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 v != parent[u])
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      vector<ii> edges { ii(1,2), ii(2,3), ii(2,4), ii(2,6),
47          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") << '\n';
56
57      return 0;
58  }
```

Outro exemplo de detecção de ciclo

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

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

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

Outro exemplo de detecção de ciclo

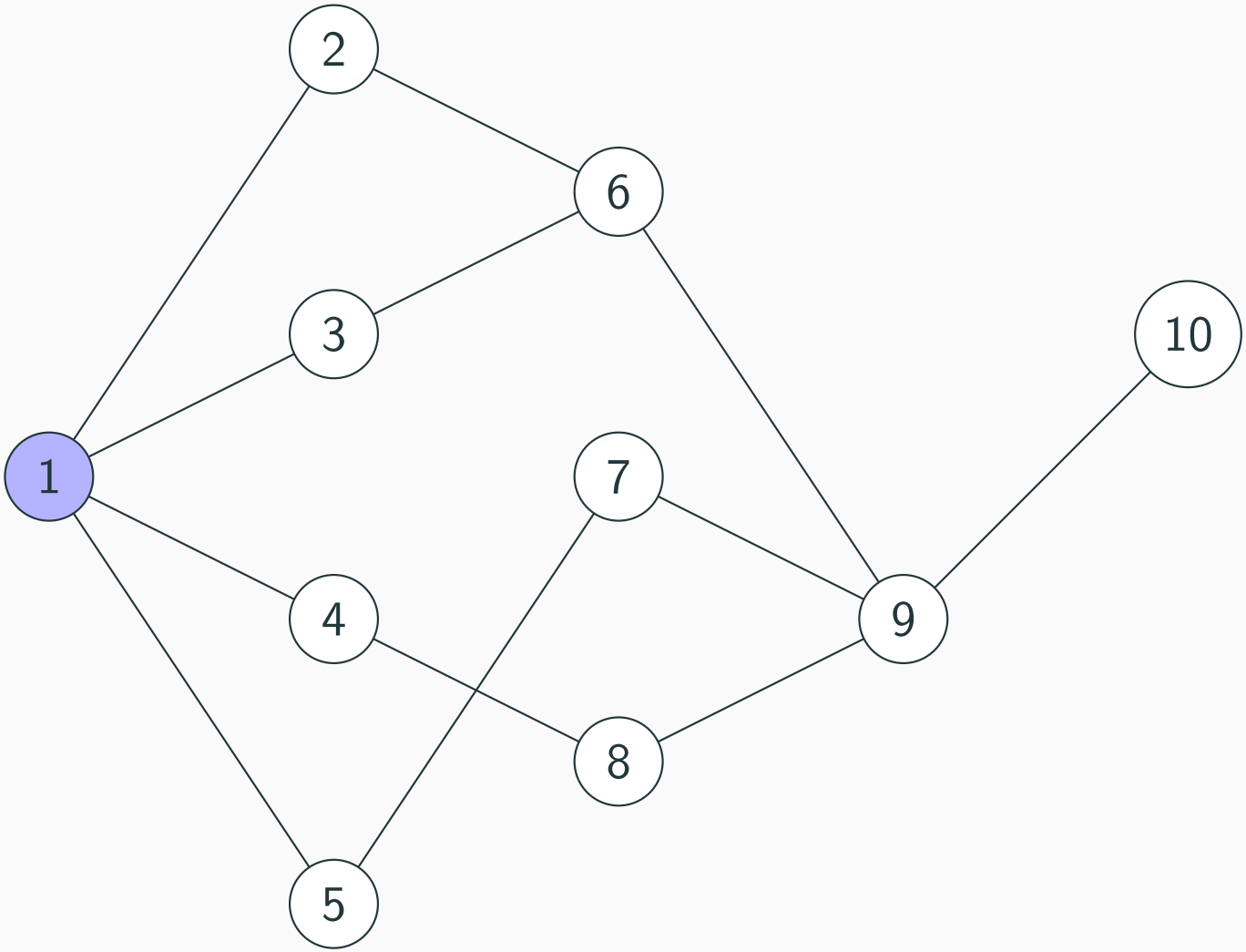
```
45      vector<ii> edges { ii(1,2), ii(2,3), ii(2,4), ii(2,6),
46          ii(4,5), ii(4,6) };
47
48      for (const auto& [u, v] : edges)
49      {
50          adj[u].push_back(v);
51          adj[v].push_back(u);
52      }
53
54      cout << "Tem ciclo? " << (has_cycle(6) ? "Sim" : "Nao") << endl;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

Grafos Bipartidos

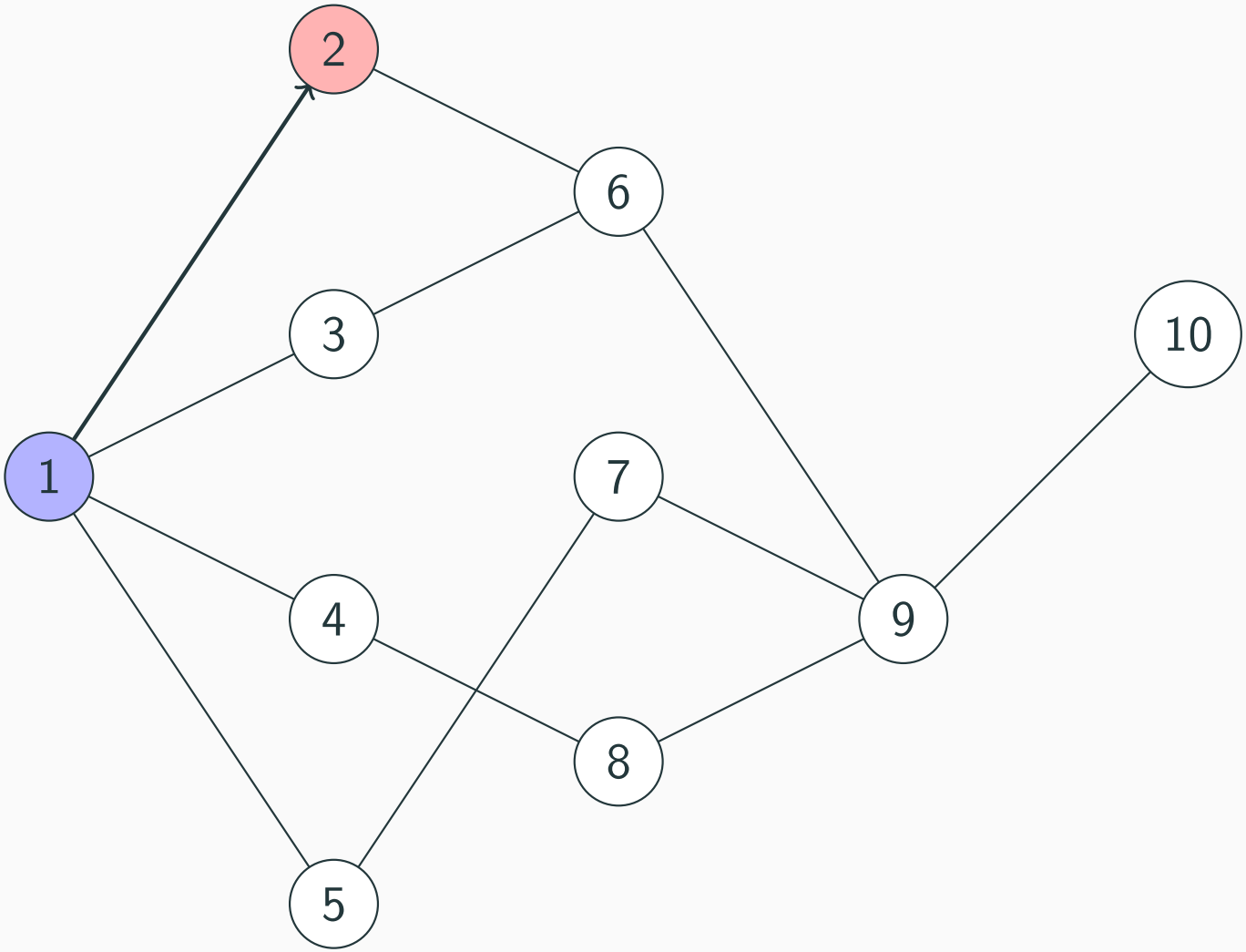
Definição

- Um grafo é dito bipartido se todos os seus vértices podem ser coloridos usando apenas duas cores, de modo que todos os pares de vértices vizinhos tenham cores distintas
- Tanto a DFS quanto a BFS podem ser utilizadas para verificar se um grafo é bipartido ou não
- Inicialmente todos os nós não tem cores atribuídas a eles, e o ponto de partida da travessia recebe uma cor (por exemplo, azul)
- A travessia continua nos seus vizinhos, que devem receber a cor oposta (por exemplo, vermelho)
- Se a travessia atingir um nó já colorido, e a cor dele for a mesma do nó que o antecedeu na travessia, o grafo não é bipartido
- Uma propriedade interessante dos grafos bipartidos é que eles não podem ter ciclos de tamanho ímpar

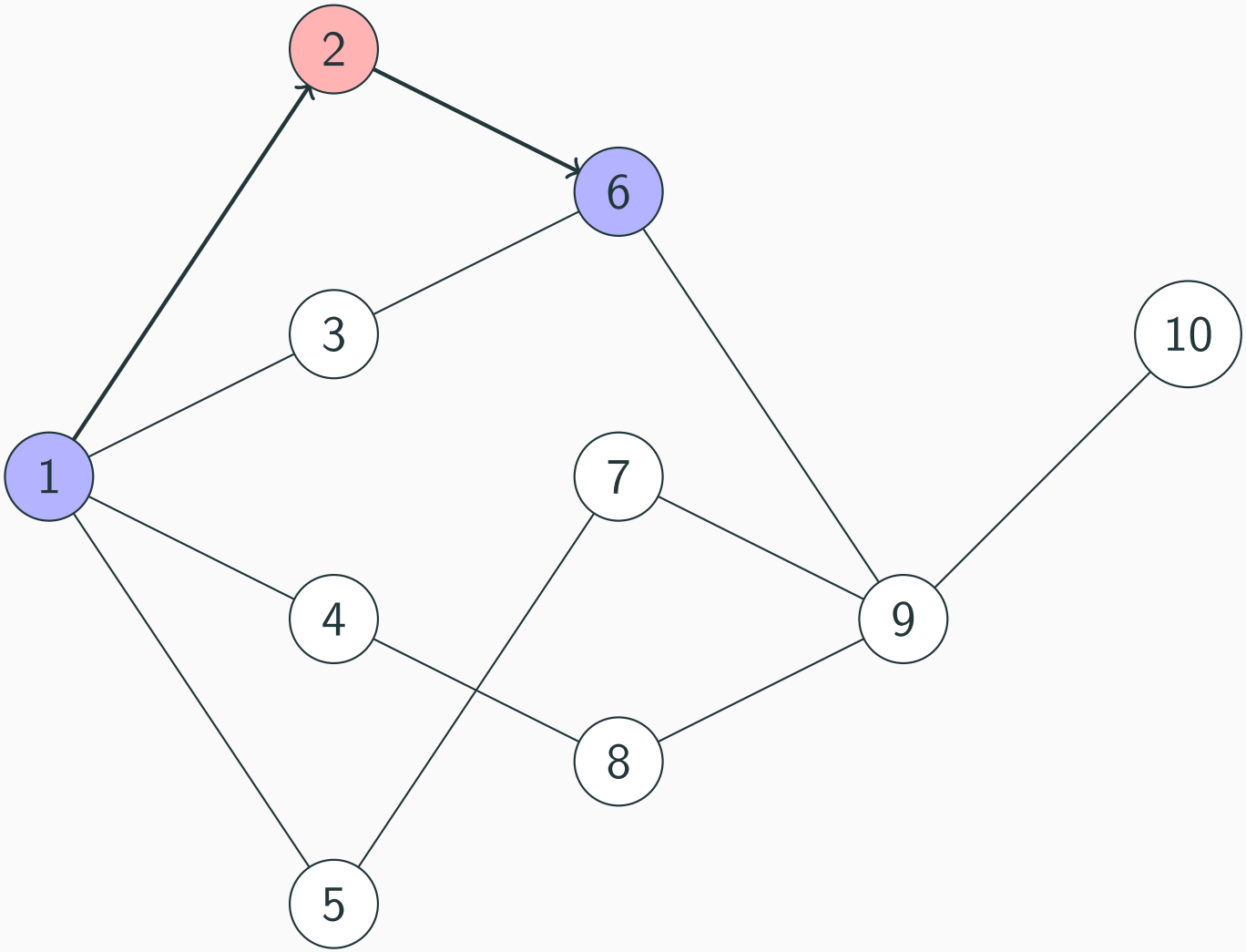
Visualização da identificação de grafos bipartidos



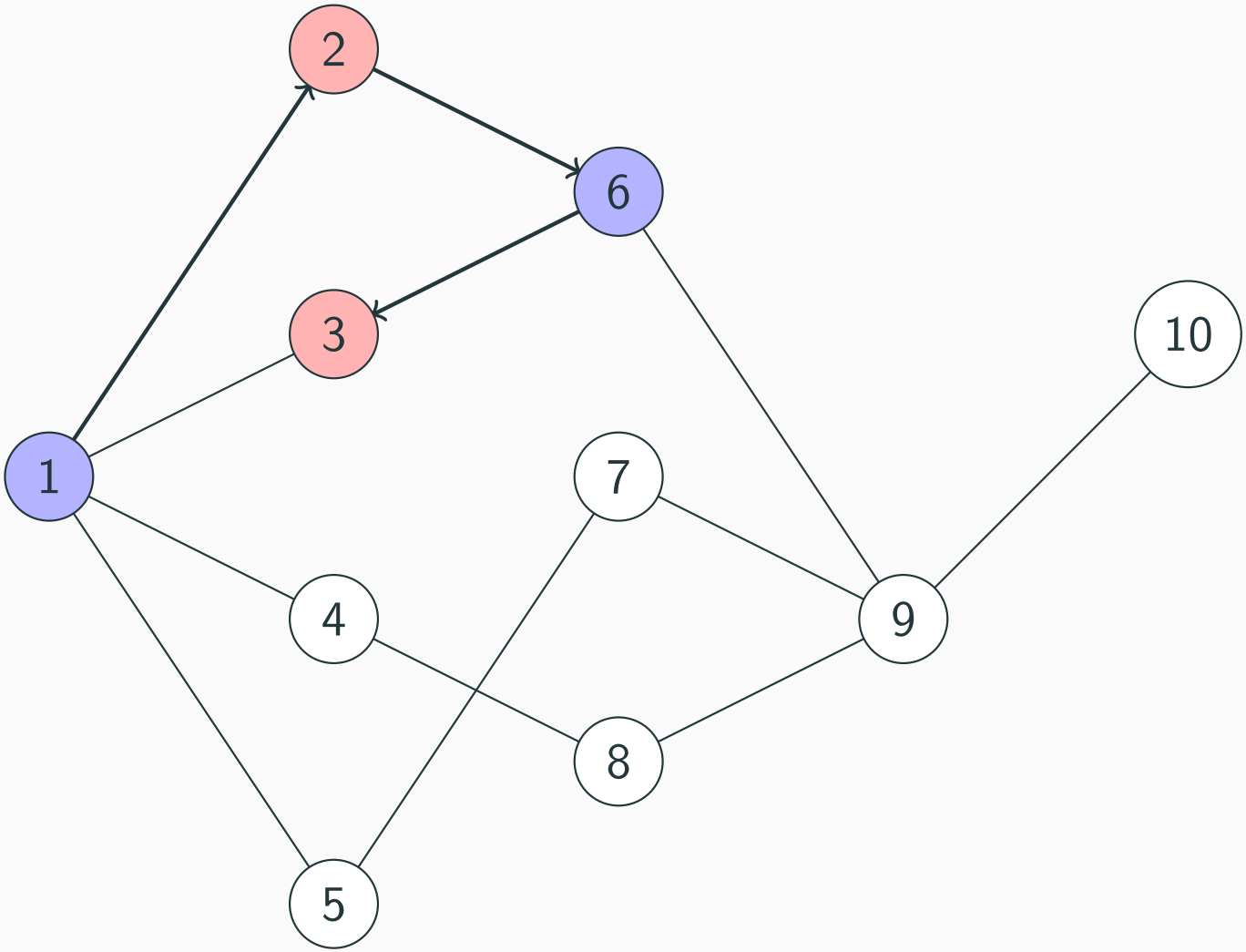
Visualização da identificação de grafos bipartidos



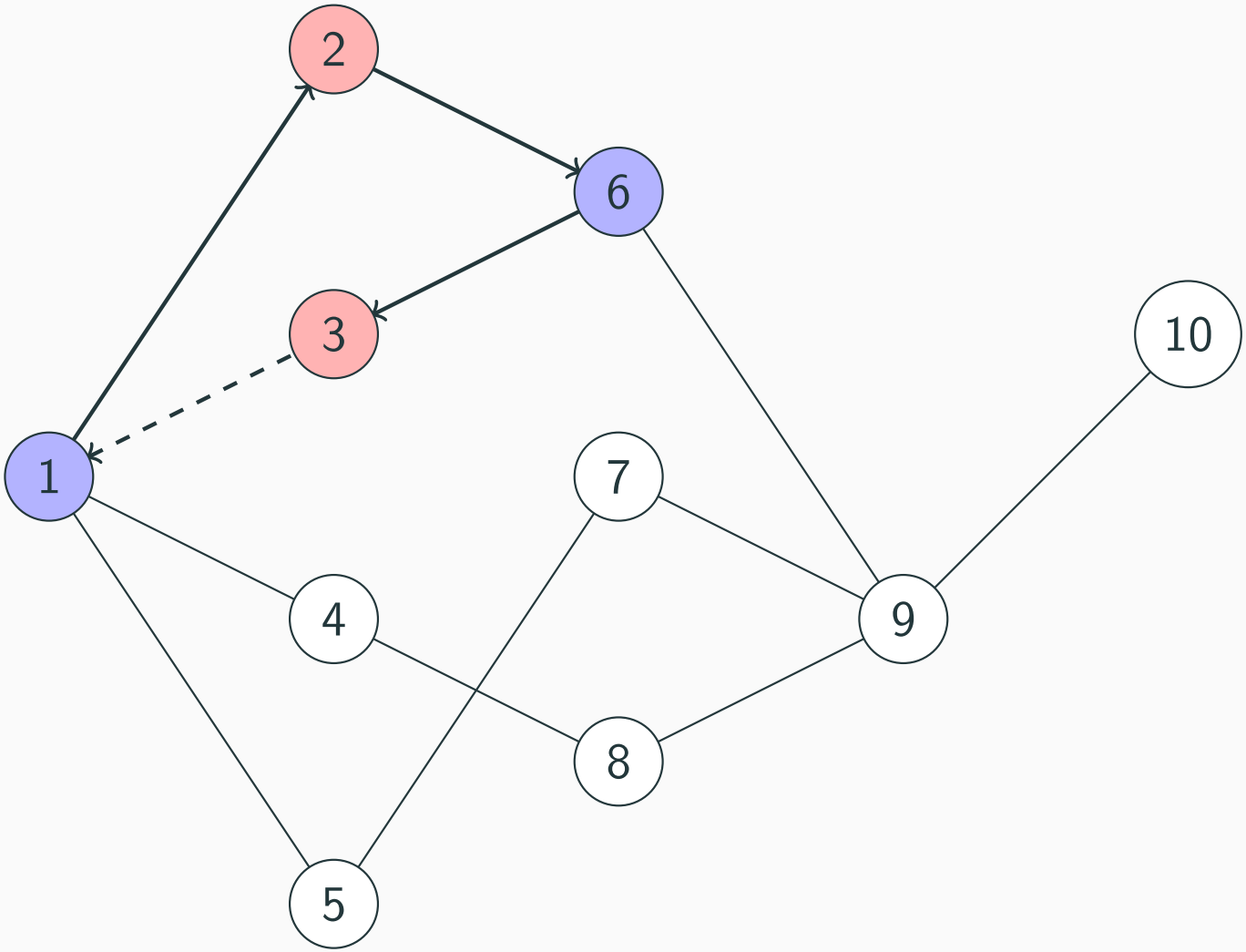
Visualização da identificação de grafos bipartidos



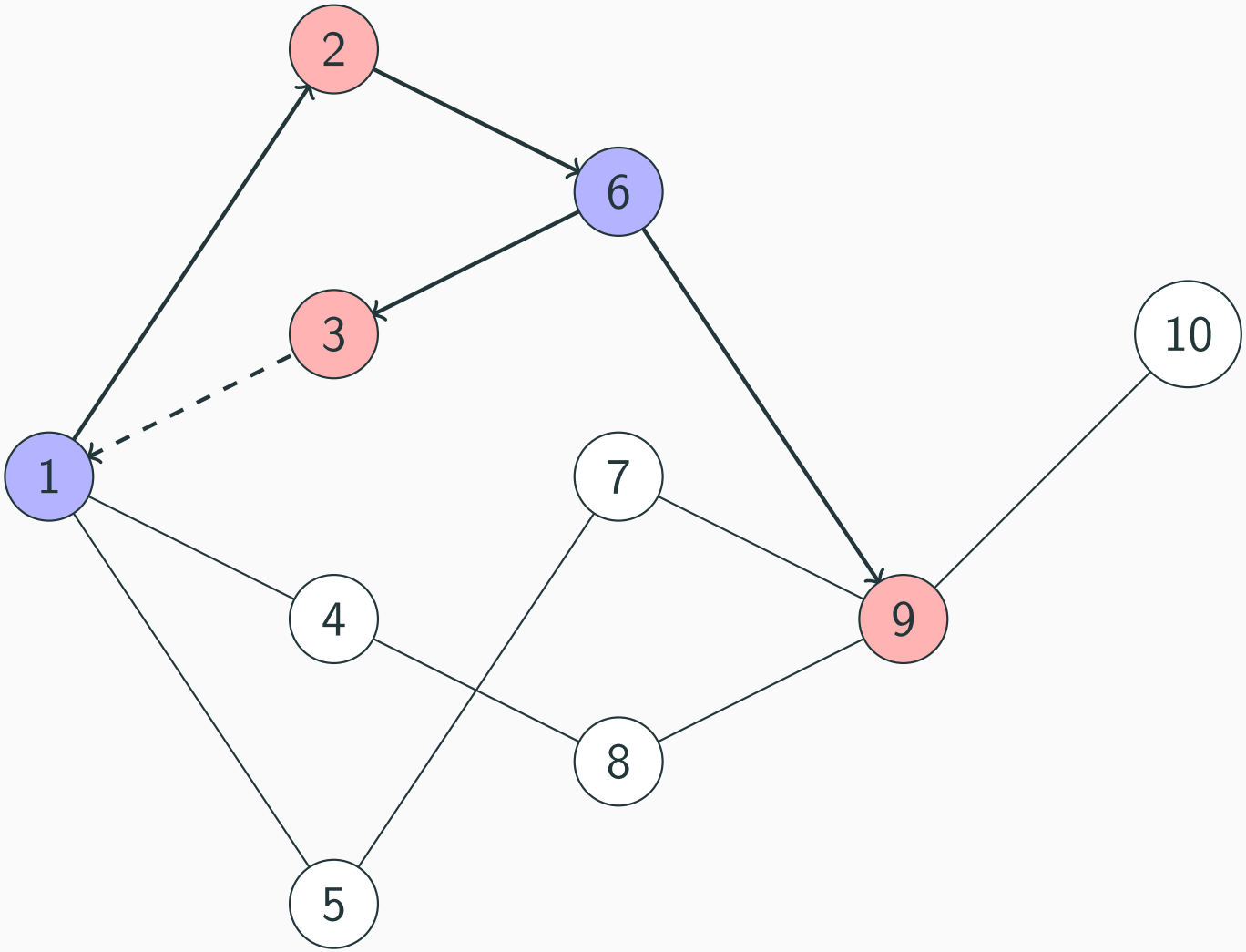
Visualização da identificação de grafos bipartidos



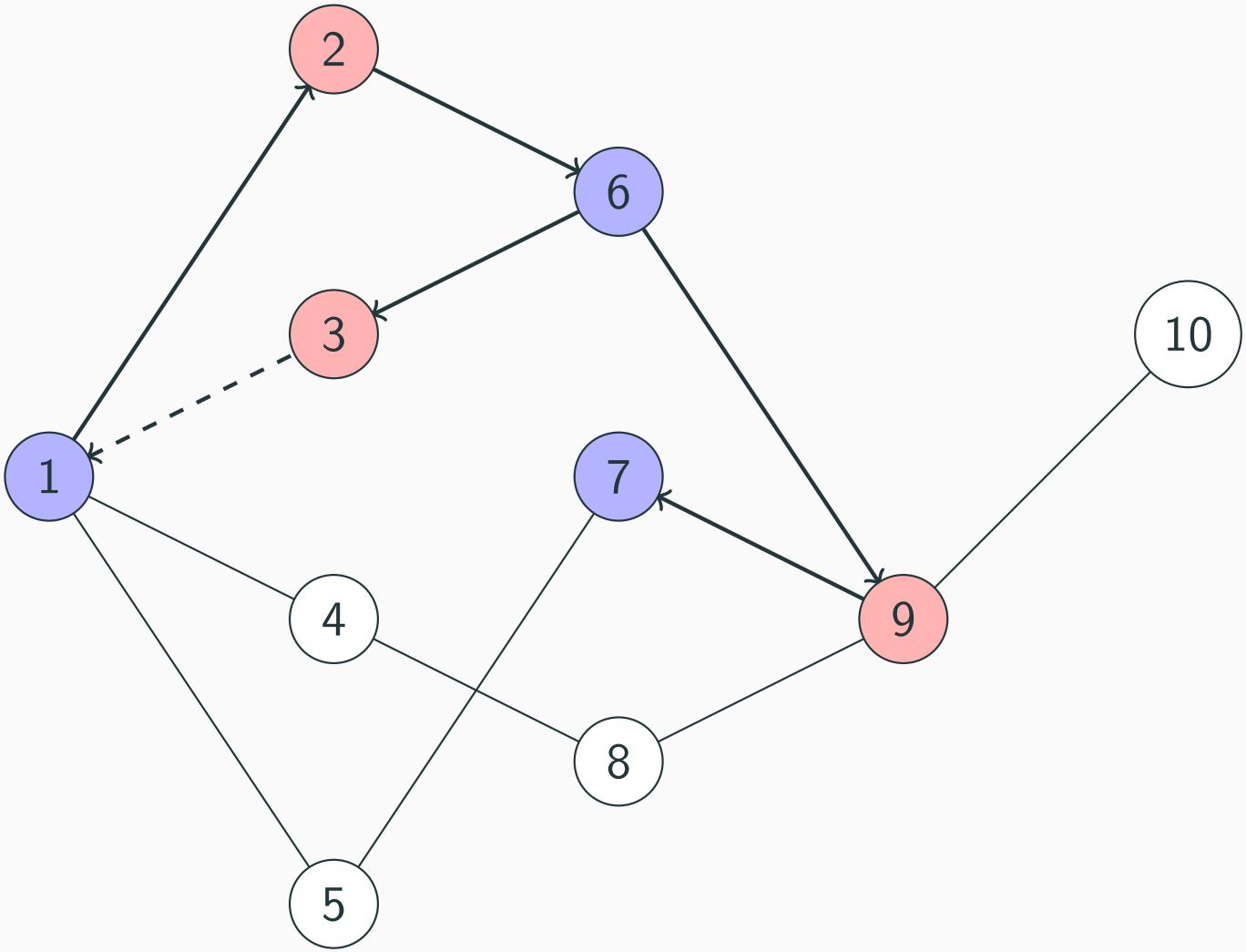
Visualização da identificação de grafos bipartidos



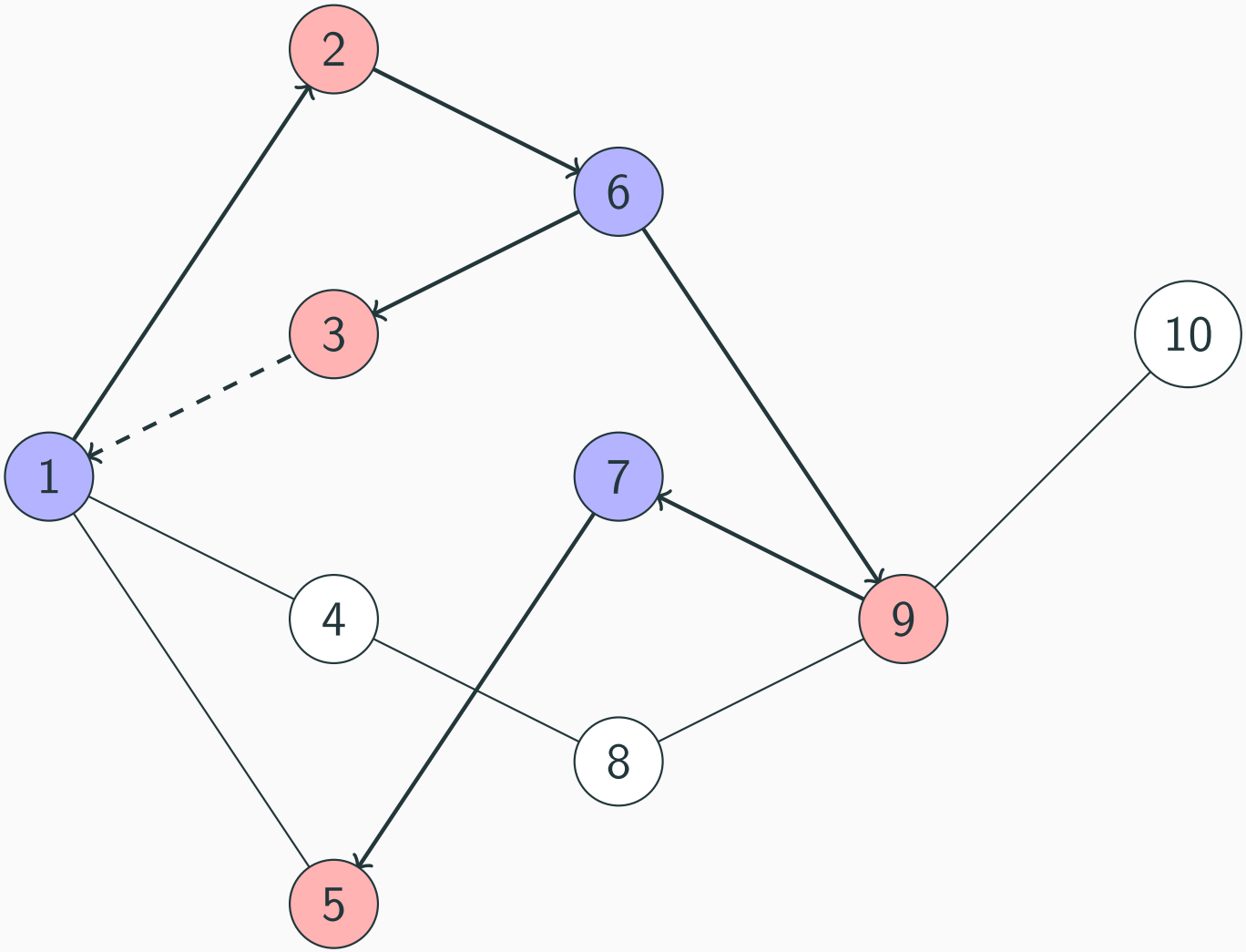
Visualização da identificação de grafos bipartidos



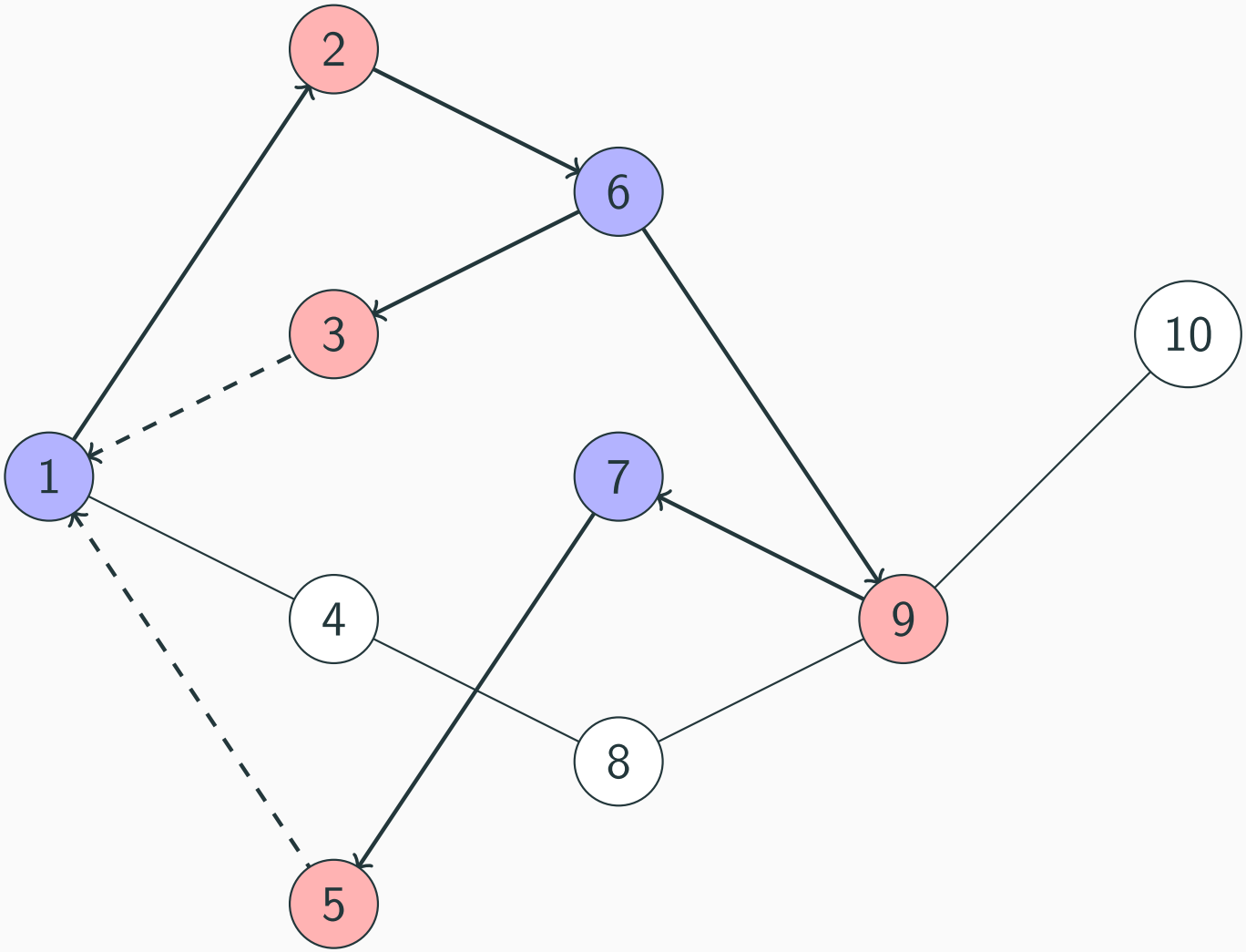
Visualização da identificação de grafos bipartidos



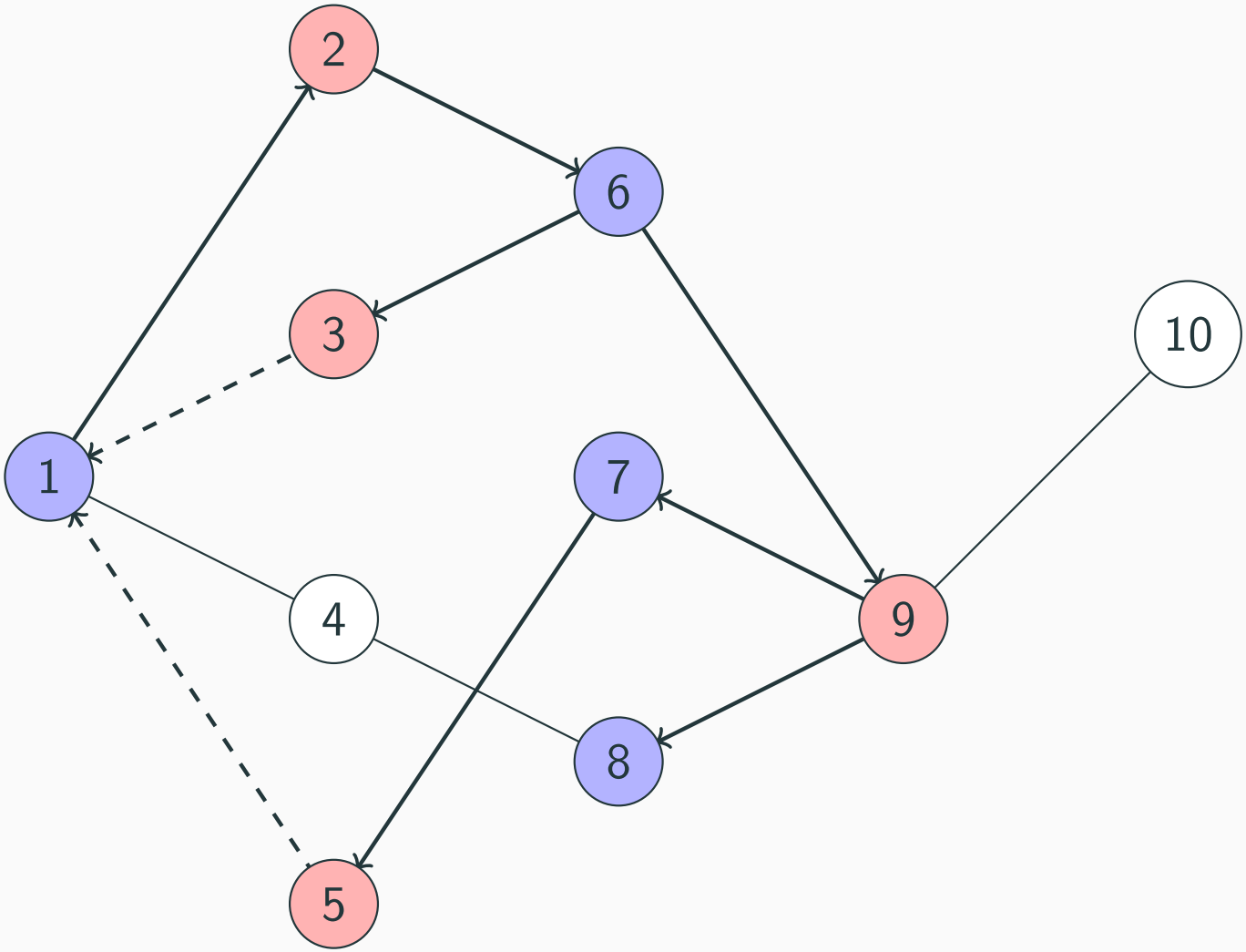
Visualização da identificação de grafos bipartidos



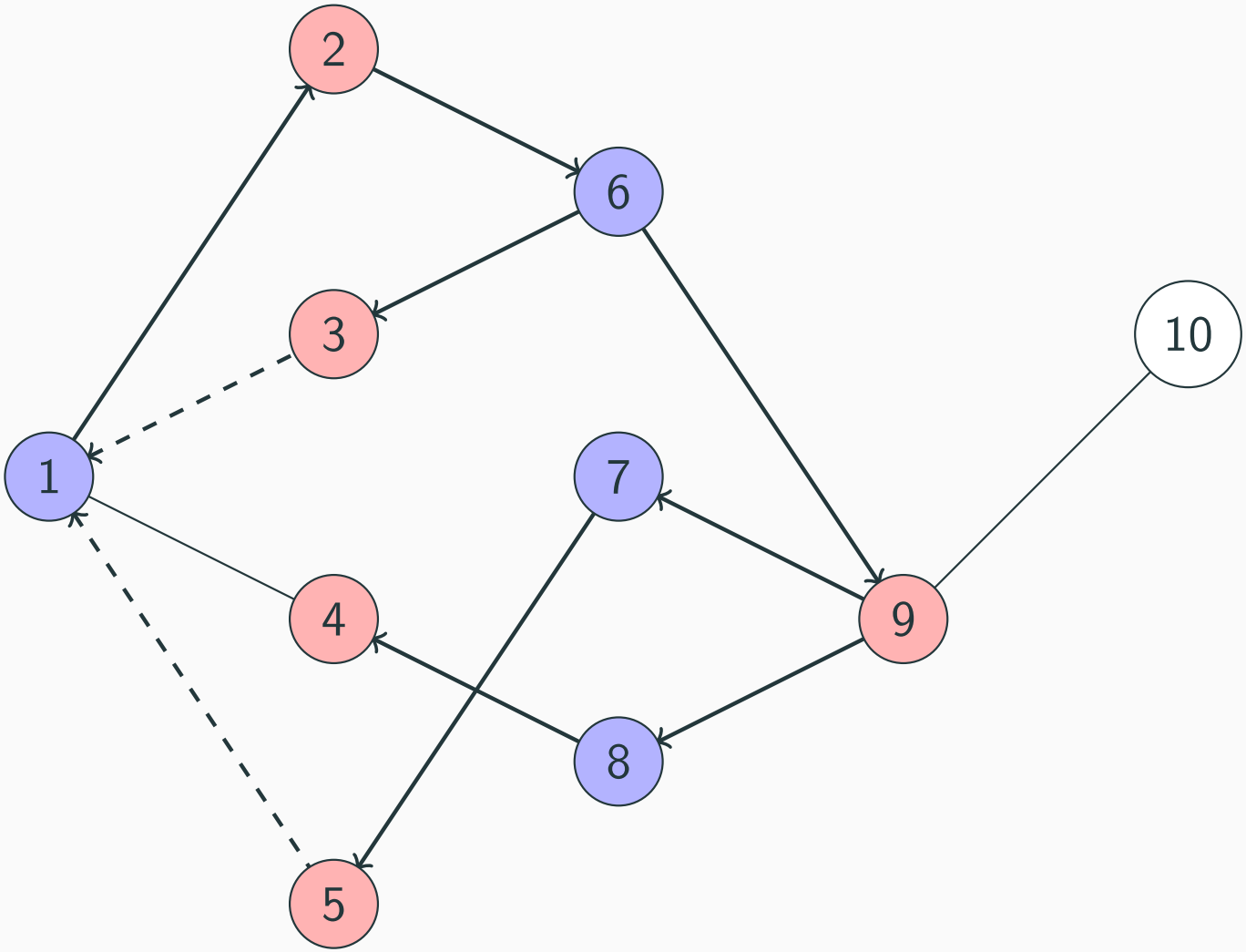
Visualização da identificação de grafos bipartidos



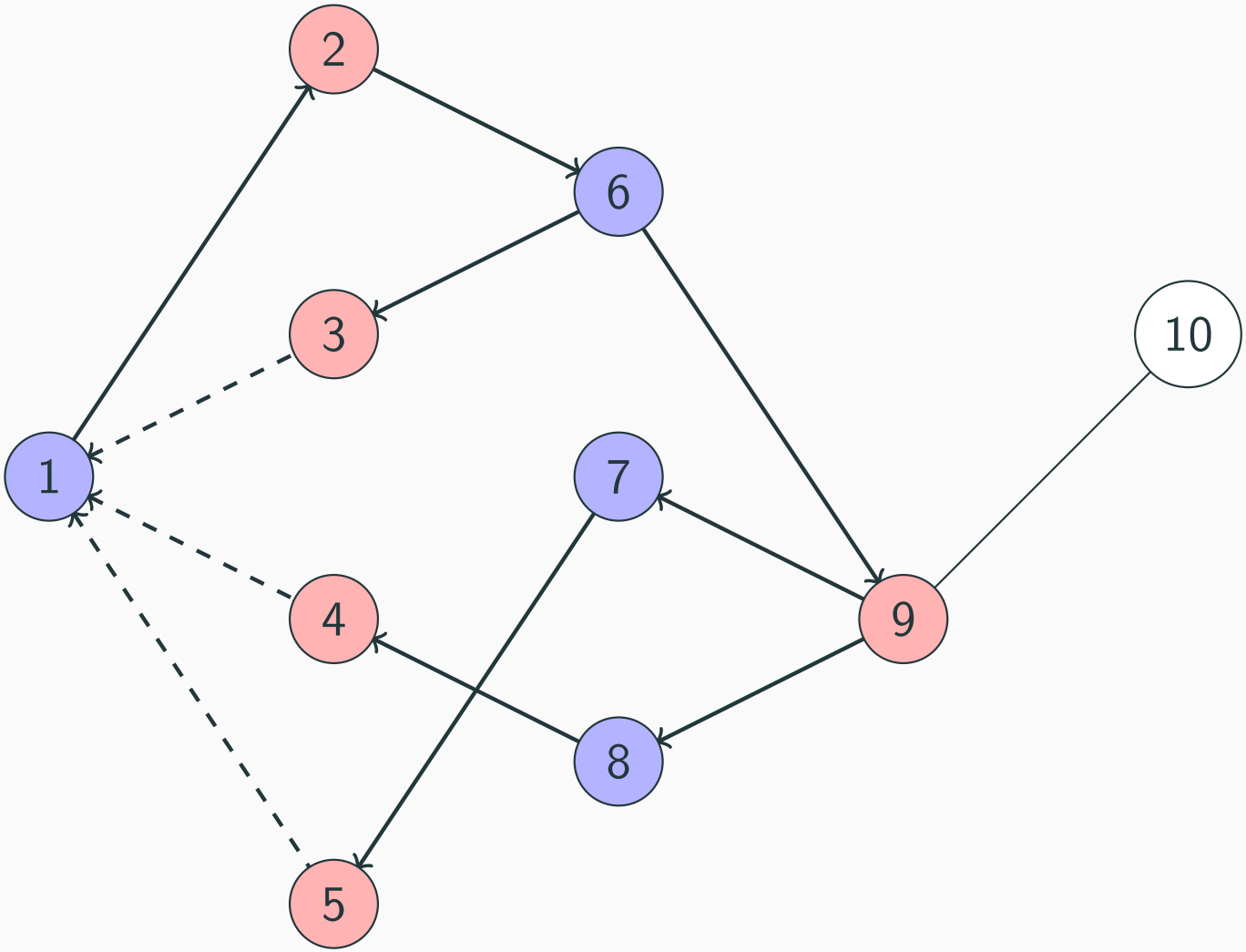
Visualização da identificação de grafos bipartidos



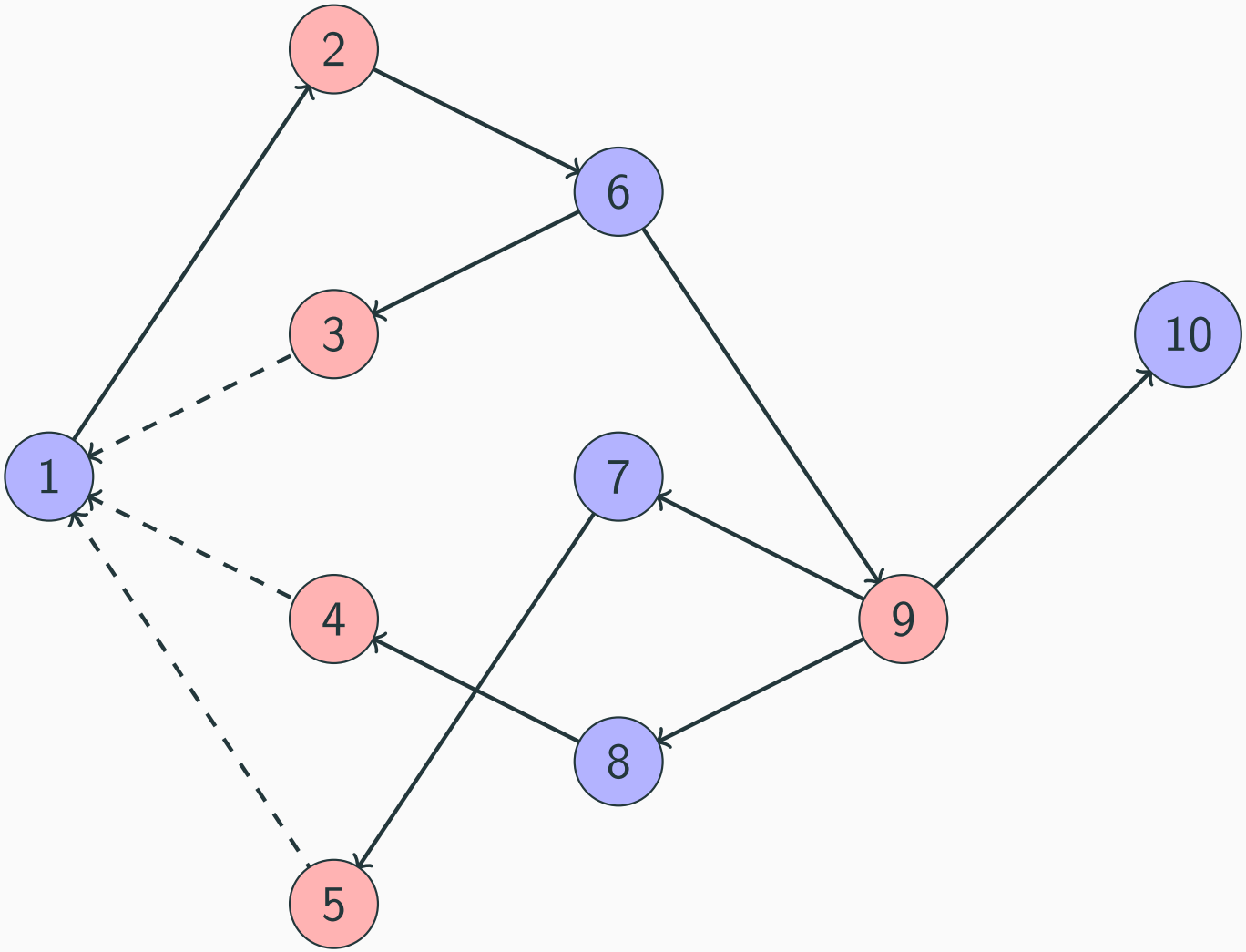
Visualização da identificação de grafos bipartidos



Visualização da identificação de grafos bipartidos



Visualização da identificação de grafos bipartidos



Implementação da identificação de grafos bipartidos em C++

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

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

const int MAX { 100010 }, NONE = 0, BLUE = 1, RED = 2;
int color[MAX];
vector<int> adj[MAX];

bool bfs(int s)
{
    queue<int> q;
    q.push(s);
    color[s] = BLUE;

    while (not q.empty())
    {
        auto u = q.front();
        q.pop();

```

Implementação da identificação de grafos bipartidos em C++

```
21          for (const auto& v : adj[u])
22              if (color[v] == NONE)
23              {
24                  color[v] = 3 - color[u];
25                  q.push(v);
26              } else if (color[v] == color[u])
27                  return false;
28      }
29
30      return true;
31  }
32
33  bool is_bipartite(int N)
34  {
35      for (int u = 1; u <= N; ++u)
36          if (color[u] == NONE and not bfs(u))
37              return false;
38
39      return true;
40  }
41
```

Implementação da identificação de grafos bipartidos em C++

```
42 int main()
43 {
44     vector<ii> edges { ii(1, 2), ii(1, 3), ii(1, 4), ii(1, 5), ii(2, 6),
45         ii(3, 6), ii(4, 8), ii(5, 7), ii(6, 9), ii(7, 9), ii(8, 9),
46         ii(9, 10) };
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 << (is_bipartite(edges.size()) ? "Sim" : "Nao") << '\n';
55
56     return 0;
57 }
```

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.