Componentes Fortemente Conexos

Sumário

1. Revisão de conceitos
2. Definição
3. Algoritmo trivial
4. Kosaraju
5. Execução - Kosaraju
6. Tarjan
7. Execução - Tarjan
8. Contração de vértices
9. Questões

Definição

Definição

Assume-se que ***G*** é um grafo direcionado

- Uma componente fortemente conexa é um conjunto de vértices tal que, para qualquer par de vértices ***(u,v)*** (contidos nesse conjunto) existe um caminho de ***u*** para ***v*** e e vice-versa.

Algoritmo Trivial

Algoritmo trivial

- Para cada vértice ***v***
  - Rodar uma dfs e verificar vértices alcançáveis a partir de ***v***

Algoritmo trivial

- Para cada vértice ***v*** // O(V)
  - Rodar uma dfs e verificar vértices alcançáveis a partir de ***v*** // O(V + E)
- Complexidade final: $O(V^2 + VE)$

Algoritmo de Kosaraju

Algoritmo de Kosaraju

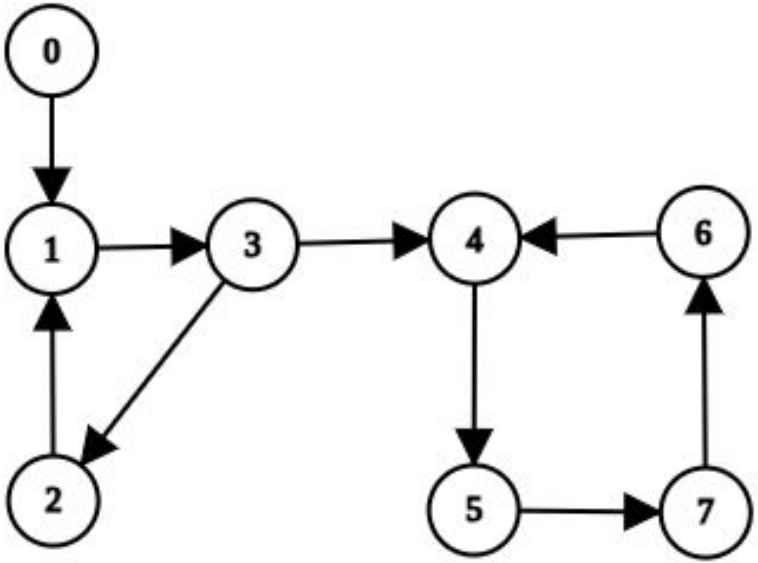
- Para cada vértice não visitado, rodar uma DFS:
  - DFS:
    - Marcar ***u*** como visitado;
    - Visitar todos os vértices ***v*** adjacentes de ***u*** ainda não visitados;
    - Guardar ***u*** em uma pilha;
- Para cada vértice da pilha, rodar uma DFS no grafo transposto:
  - DFS:
    - Marcar u como visitado;
    - Visitar todos os vértices ***v*** adjacentes de ***u*** ainda não visitados;
- Os vértices visitados por cada DFS no grafo transposto serão uma componente fortemente conexa;

Algoritmo de Kosaraju

- Para cada vértice não visitado, rodar uma DFS:
  - DFS:
    - Marcar ***u*** como visitado;
    - Visitar todos os vértices ***v*** adjacentes de ***u*** ainda não visitados;
    - Guardar ***u*** em uma pilha;
- Para cada vértice da pilha, rodar uma DFS no grafo transposto:
  - DFS:
    - Marcar u como visitado;
    - Visitar todos os vértices ***v*** adjacentes de ***u*** ainda não visitados;
- Os vértices visitados por cada DFS no grafo transposto serão uma componente fortemente conexa;
- Complexidade final: O(V + E)

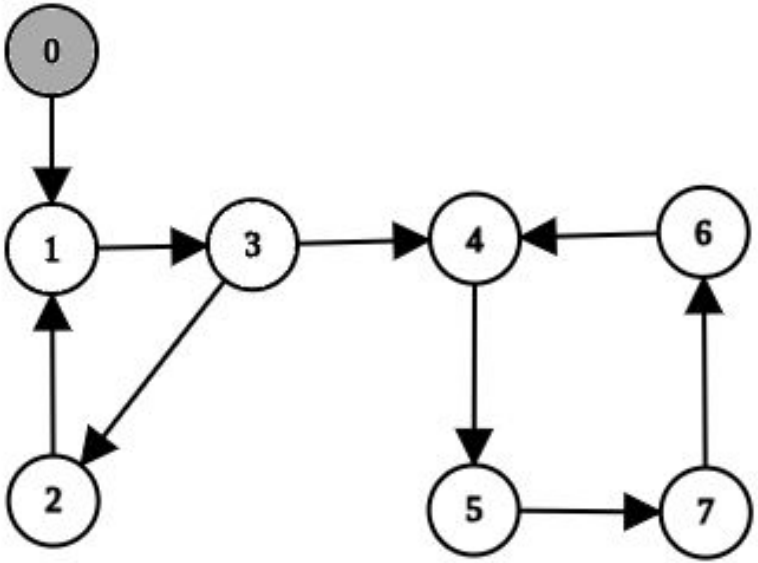
Execução - Algoritmo de Kosaraju

Execução Kosaraju - Primeira DFS



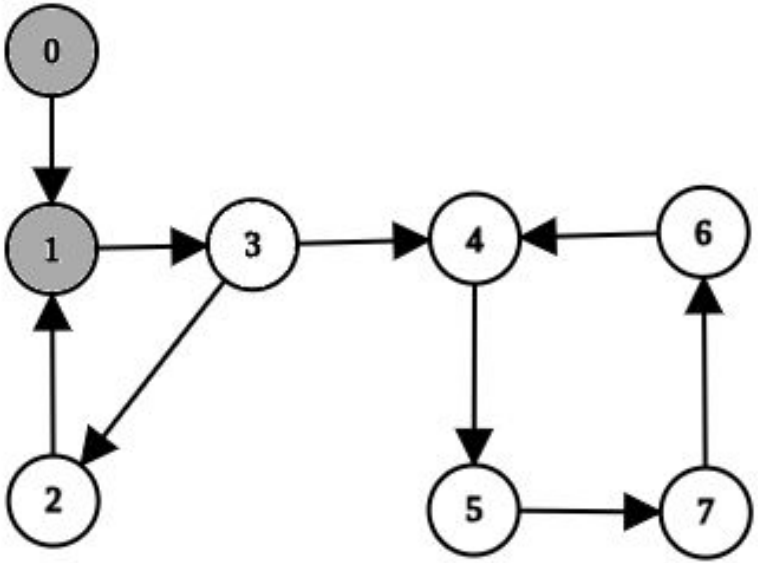
Execução Kosaraju - Primeira DFS

Stack:



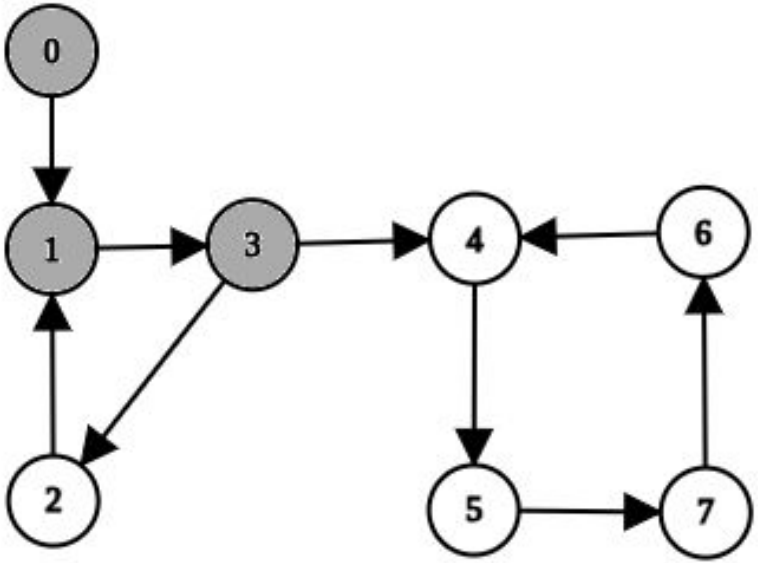
Execução Kosaraju - Primeira DFS

Stack:



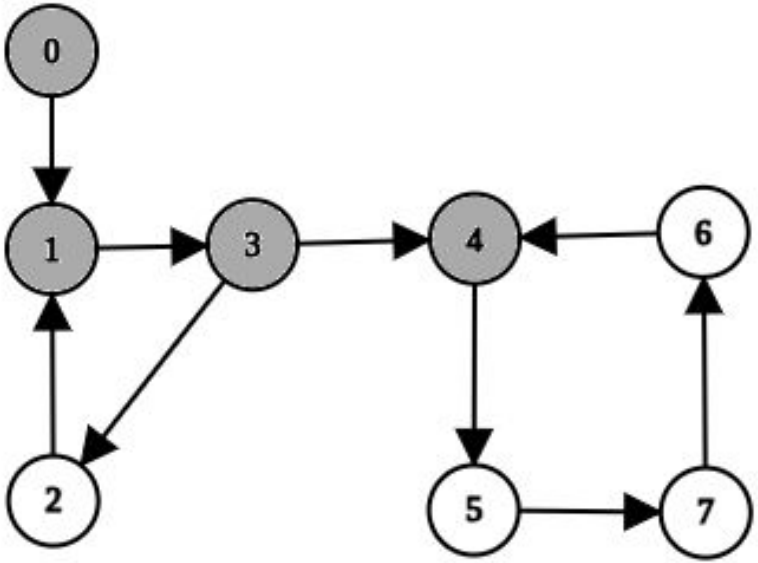
Execução Kosaraju - Primeira DFS

Stack:



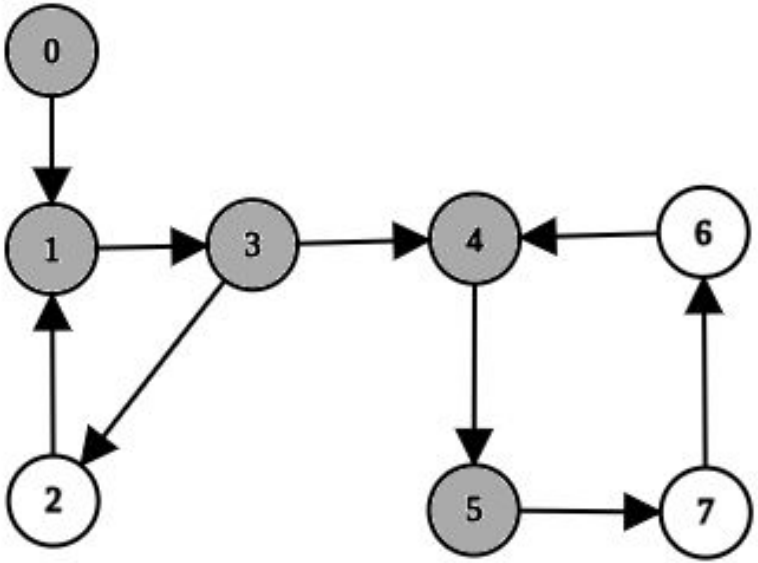
Execução Kosaraju - Primeira DFS

Stack:

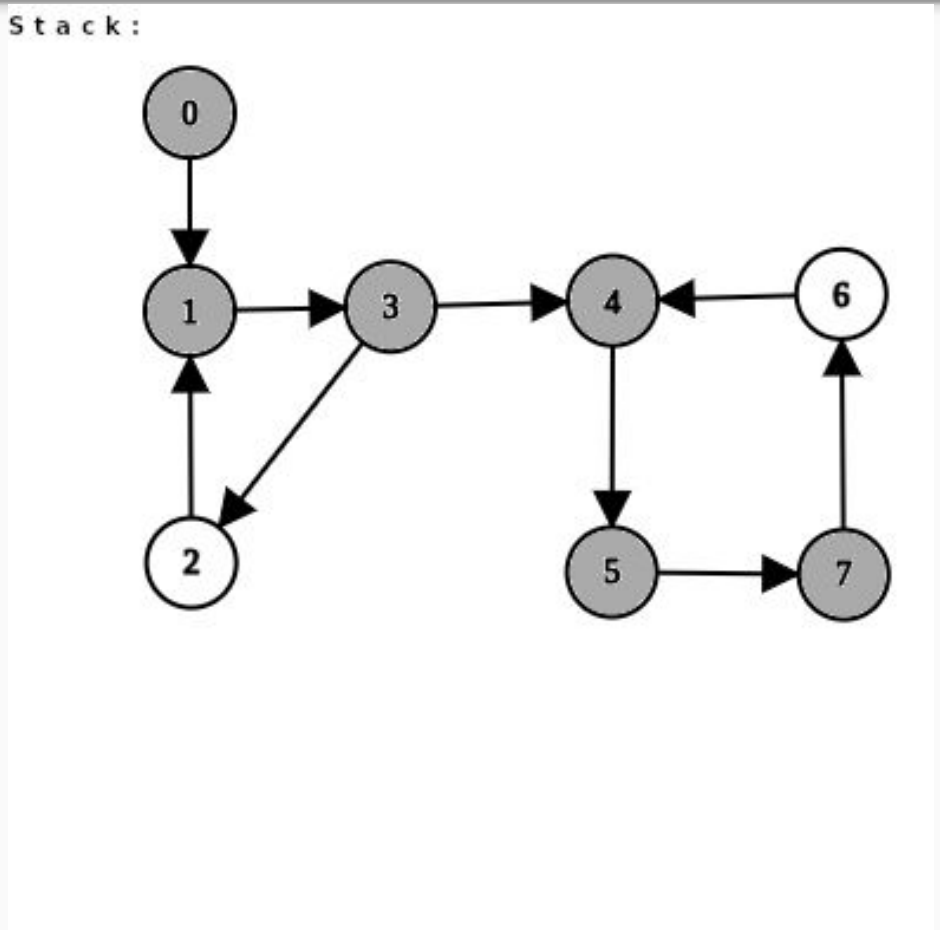


Execução Kosaraju - Primeira DFS

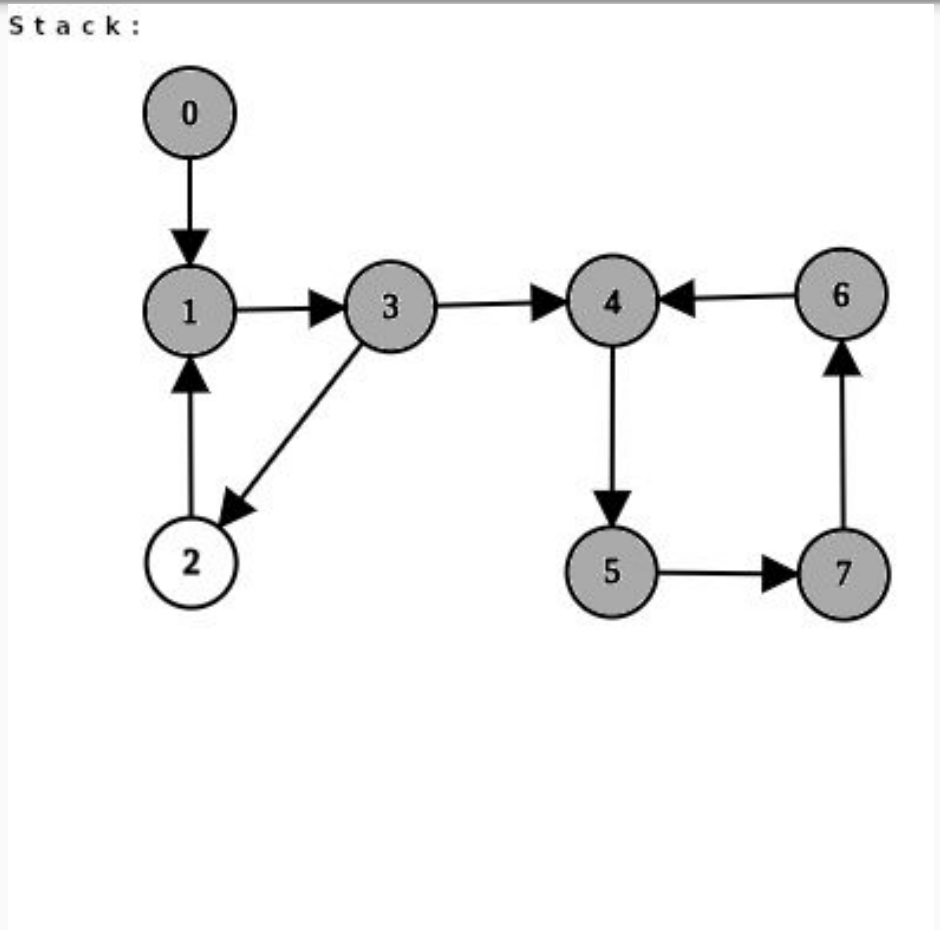
Stack:



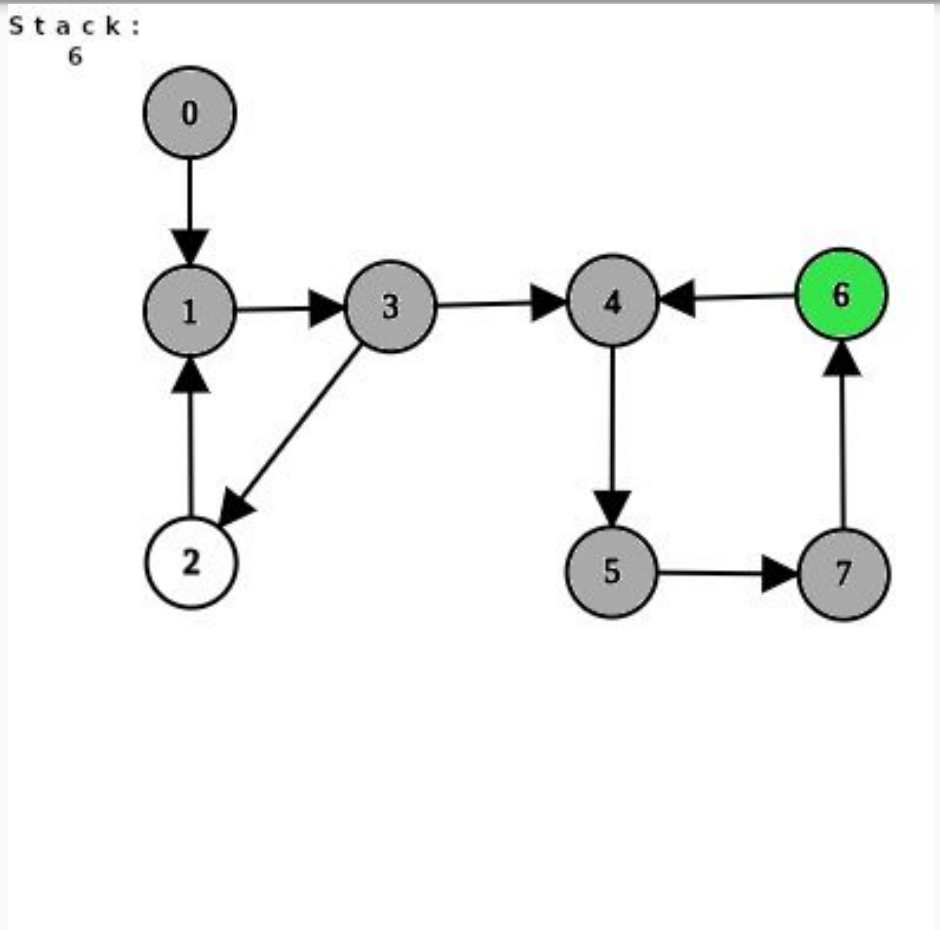
Execução Kosaraju - Primeira DFS



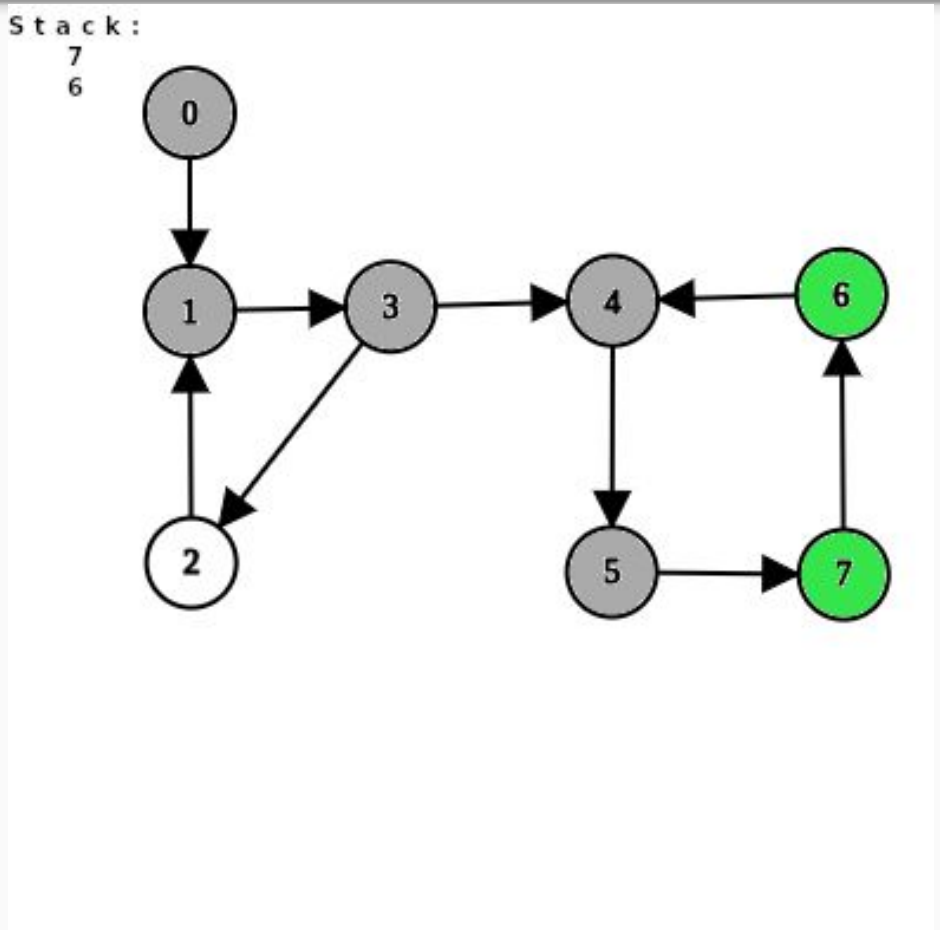
Execução Kosaraju - Primeira DFS



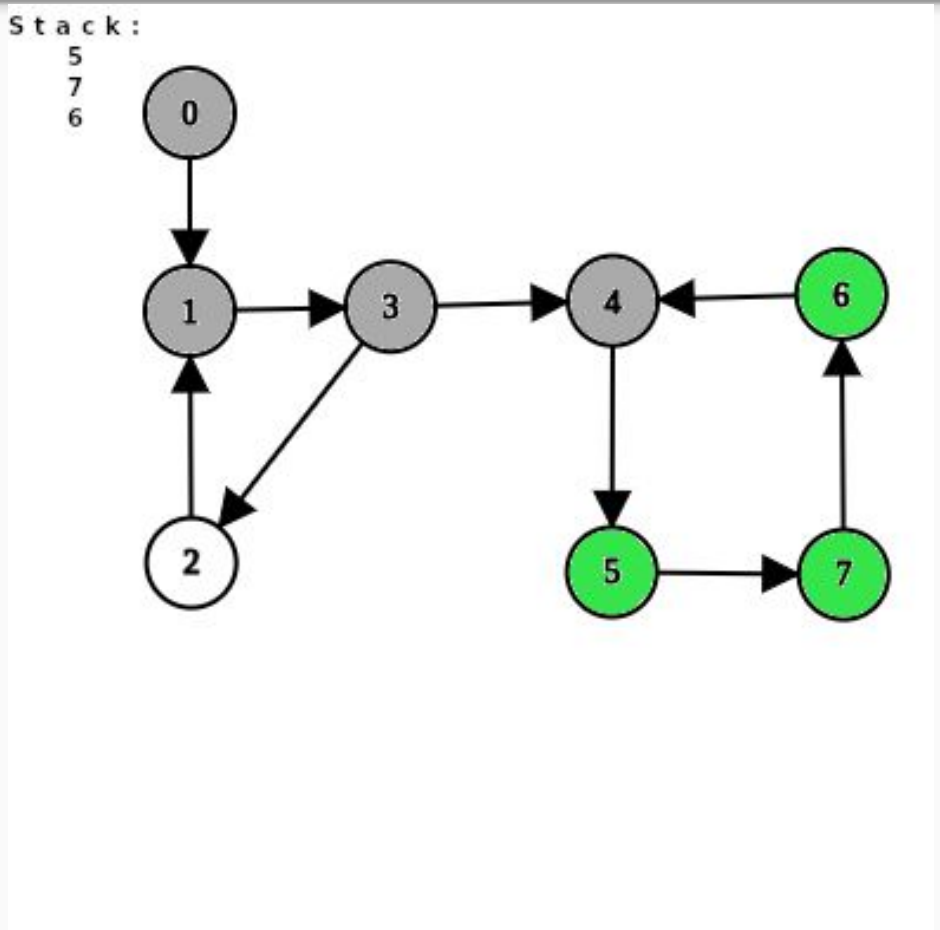
Execução Kosaraju - Primeira DFS



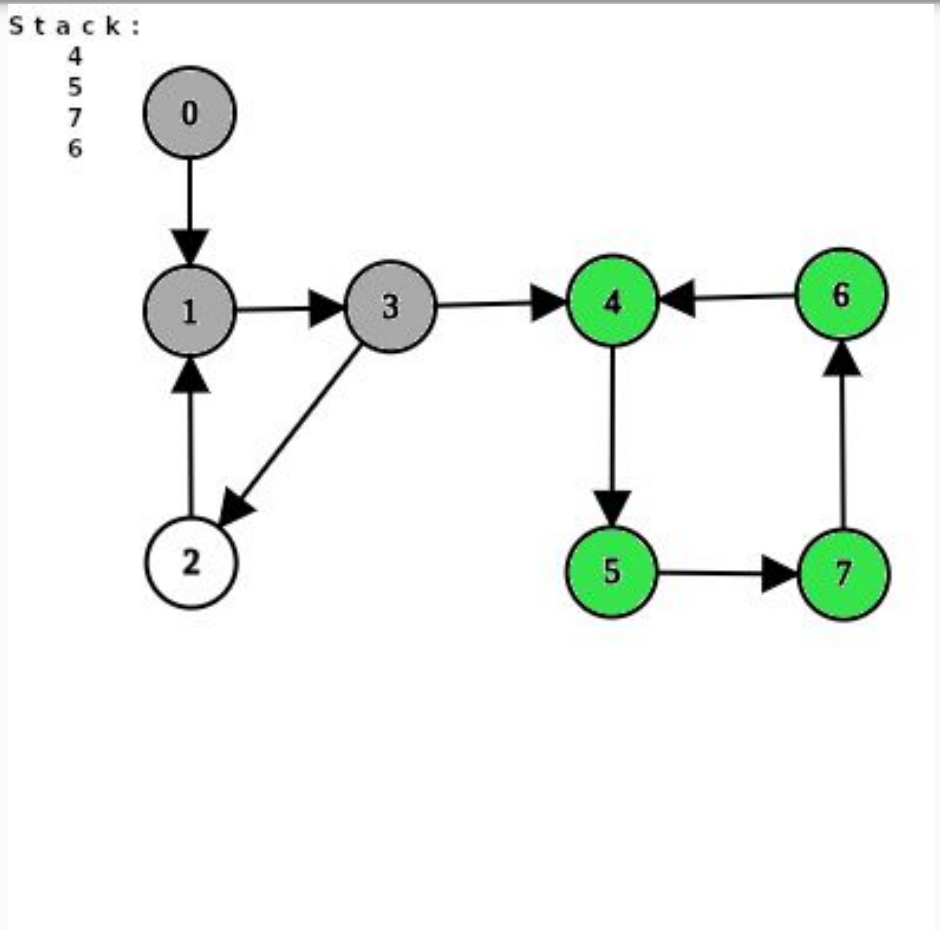
Execução Kosaraju - Primeira DFS



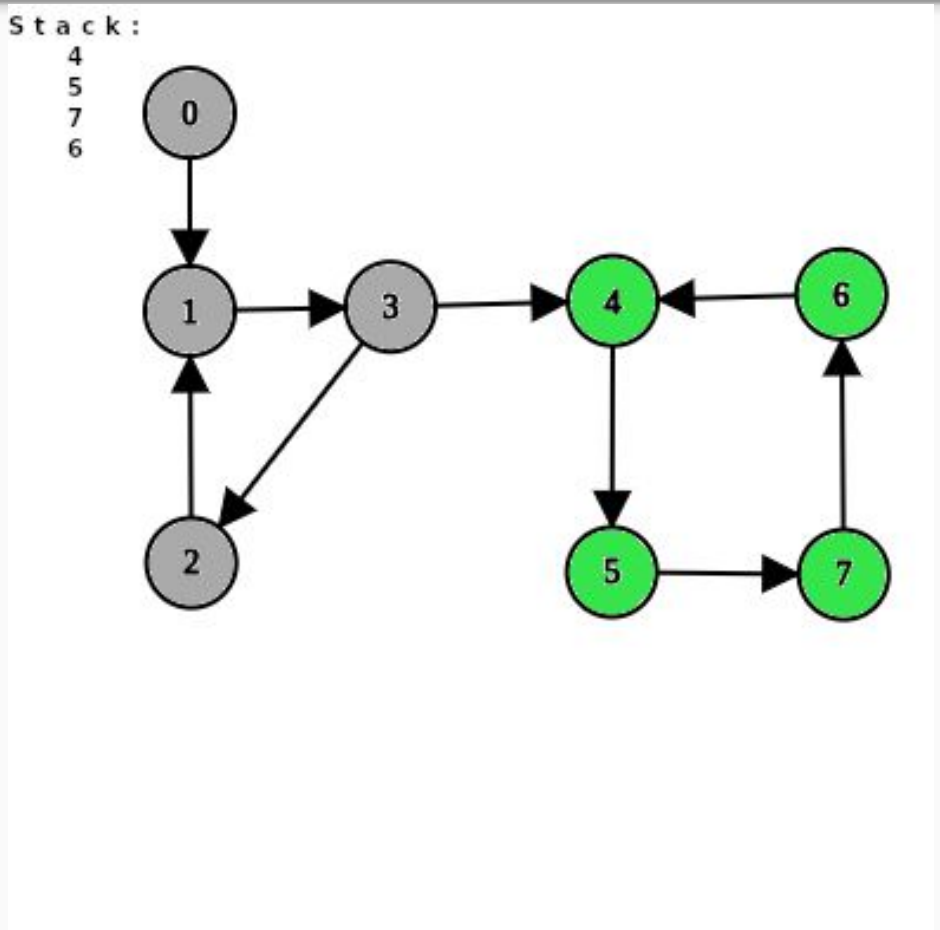
Execução Kosaraju - Primeira DFS



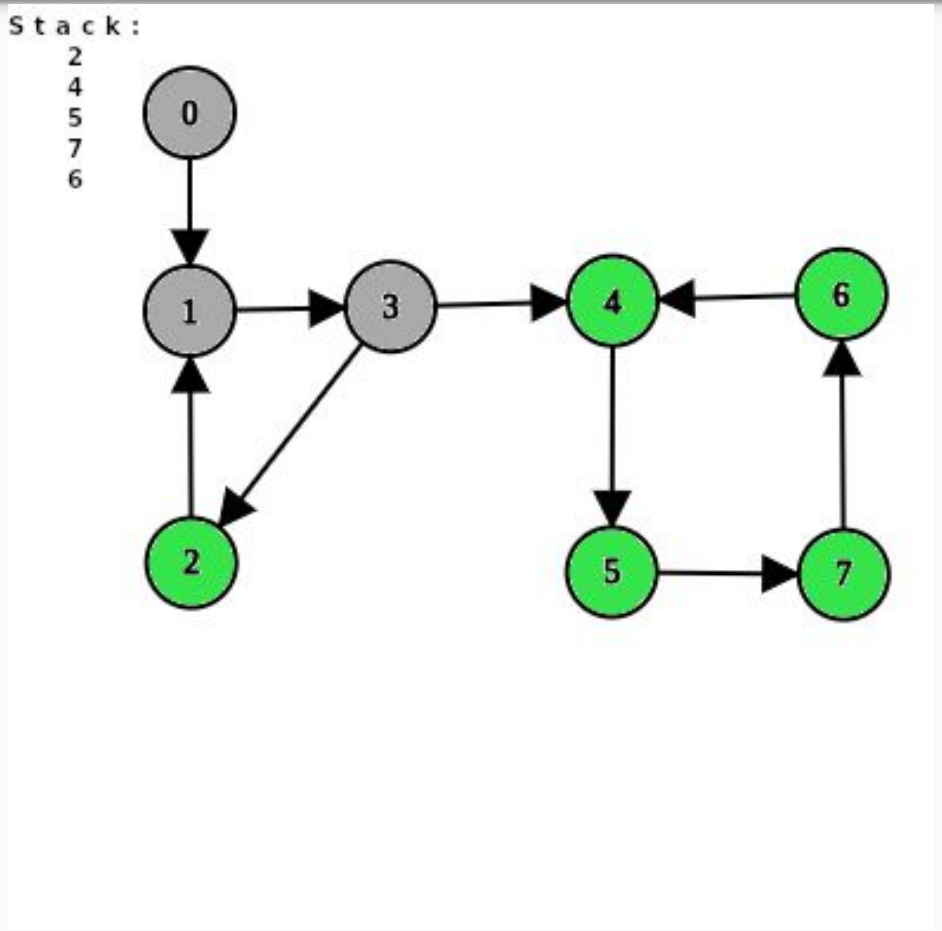
Execução Kosaraju - Primeira DFS



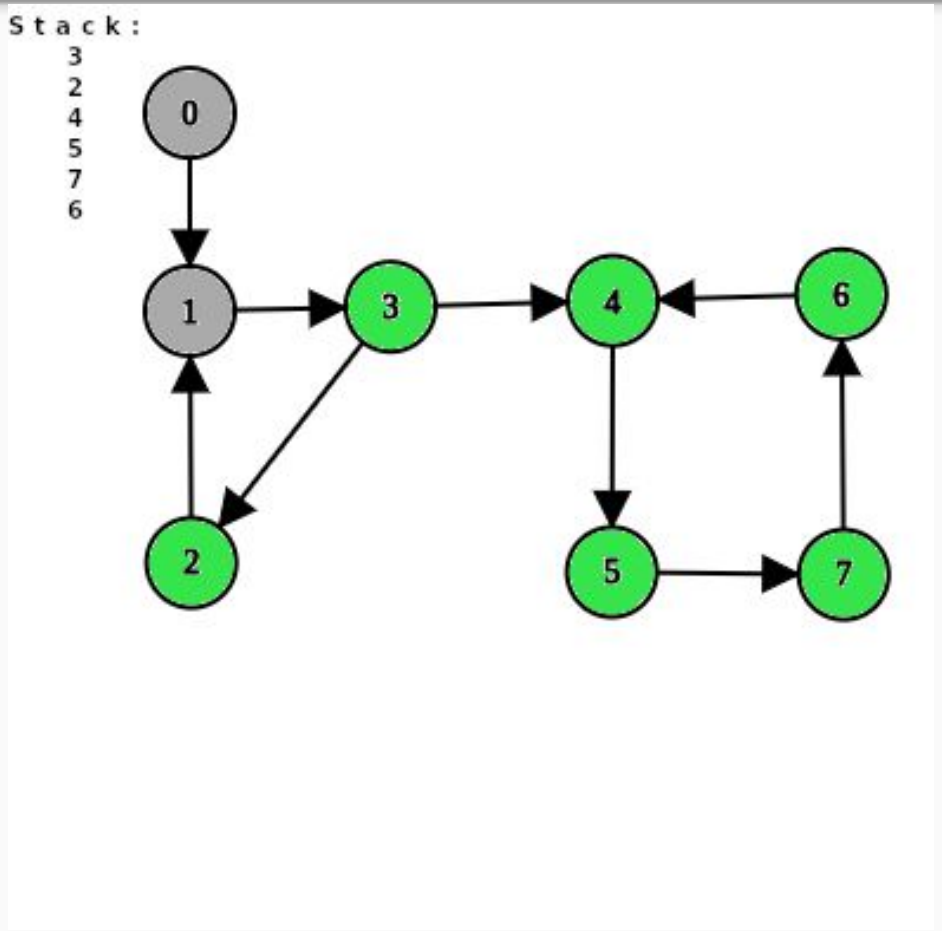
Execução Kosaraju - Primeira DFS



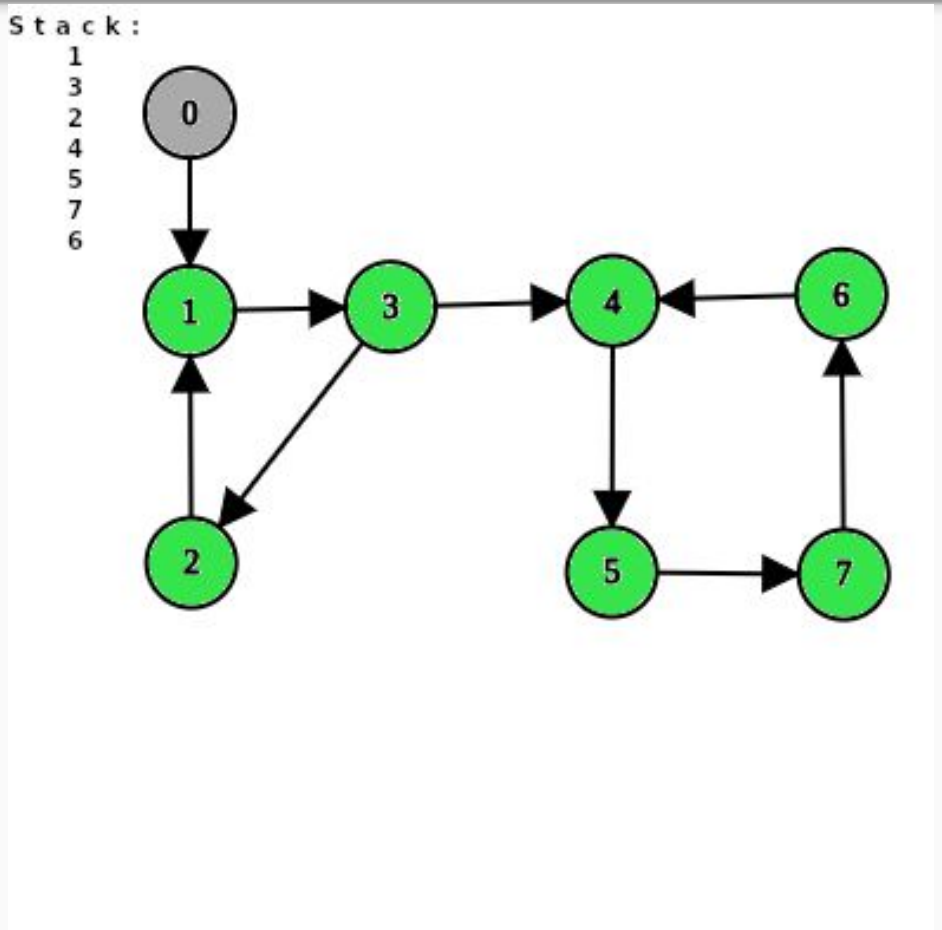
Execução Kosaraju - Primeira DFS



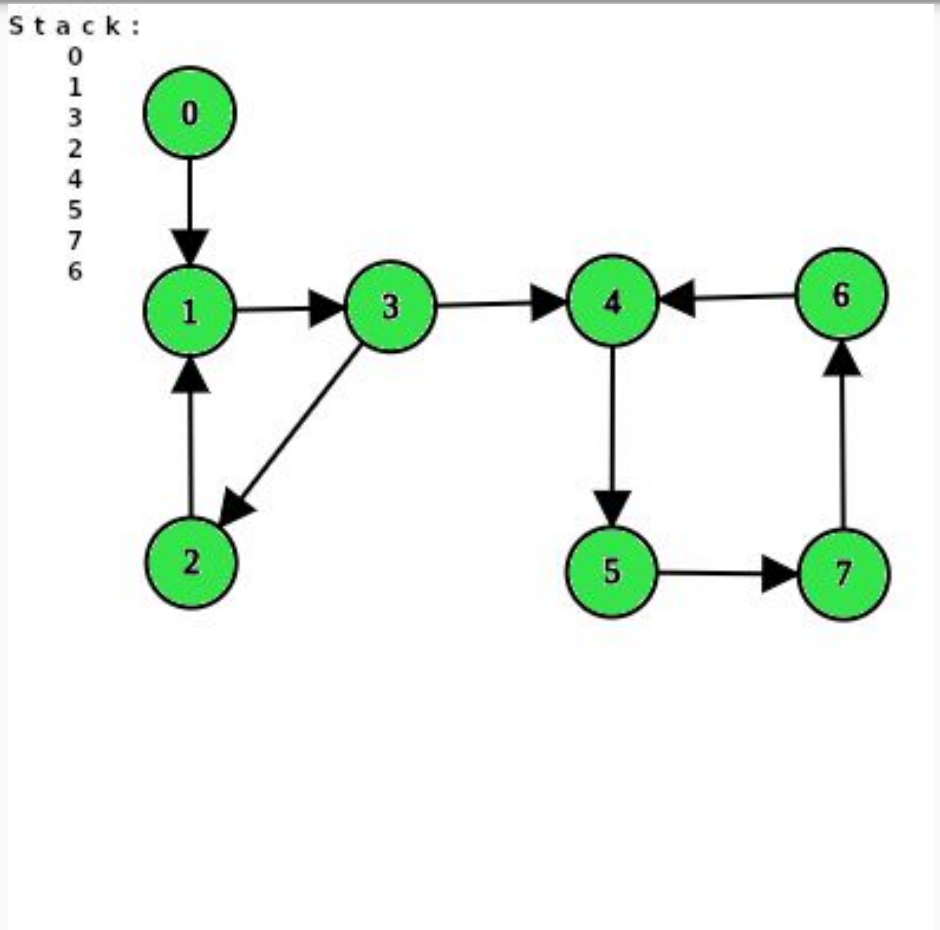
Execução Kosaraju - Primeira DFS



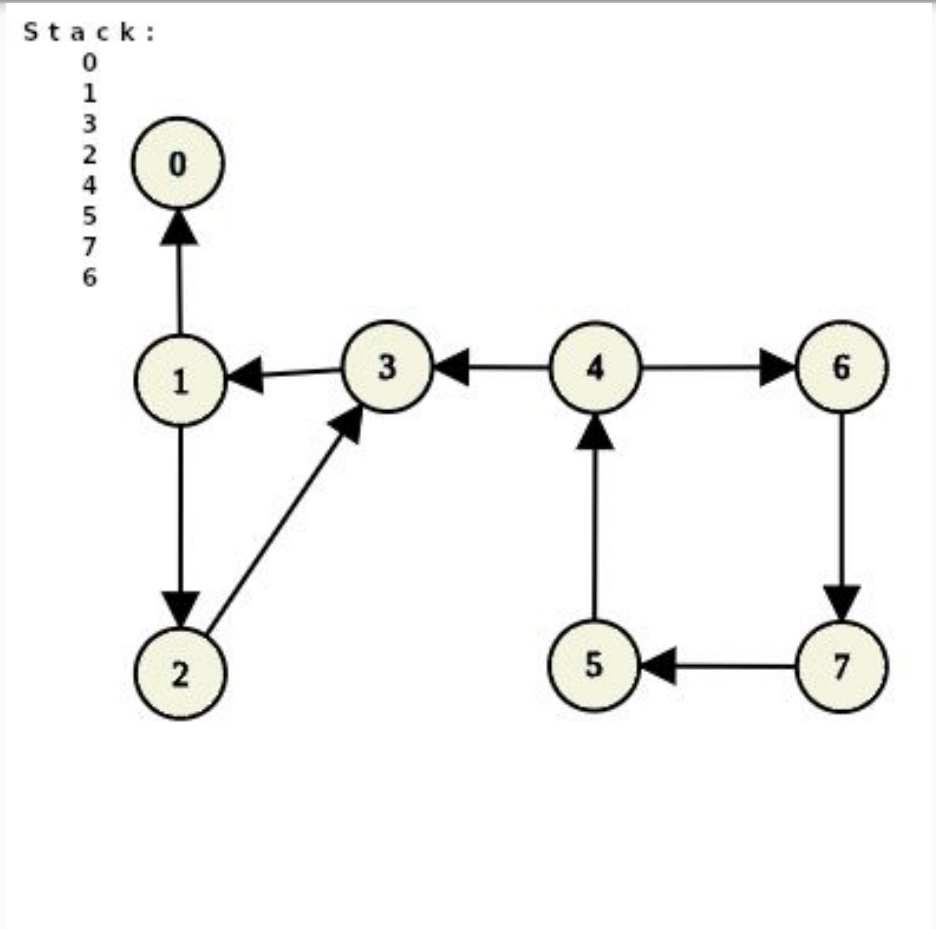
Execução Kosaraju - Primeira DFS



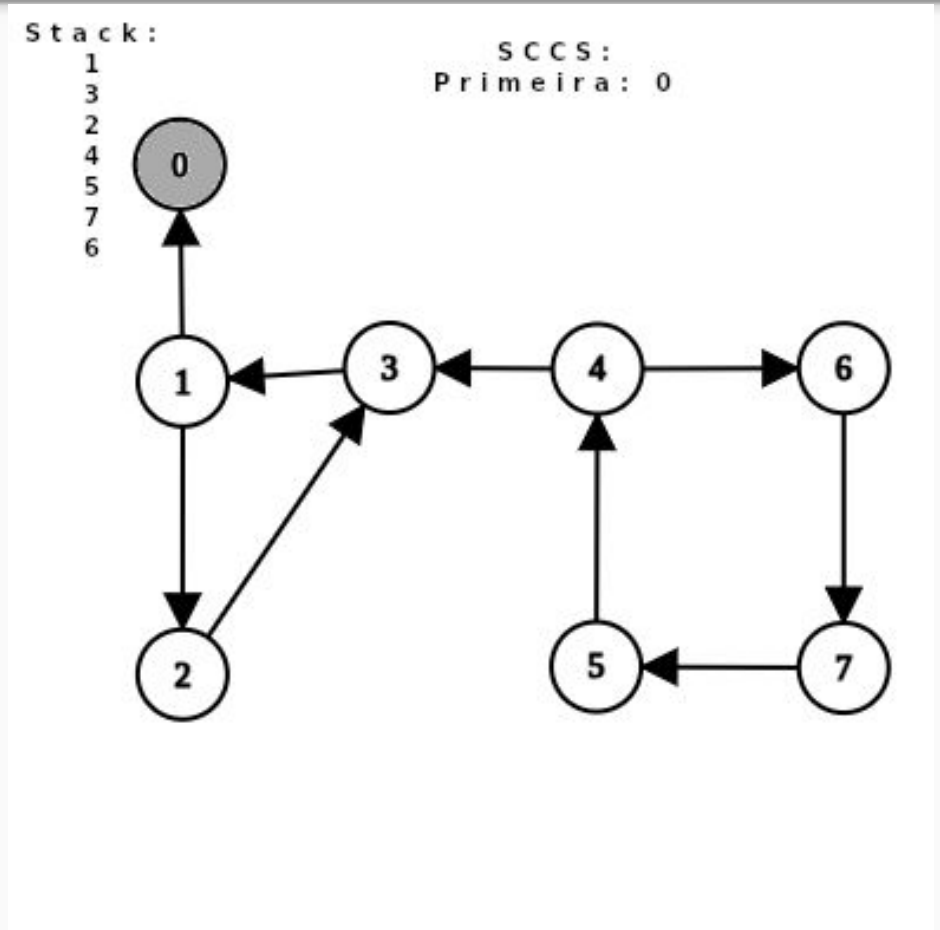
Execução Kosaraju - Primeira DFS



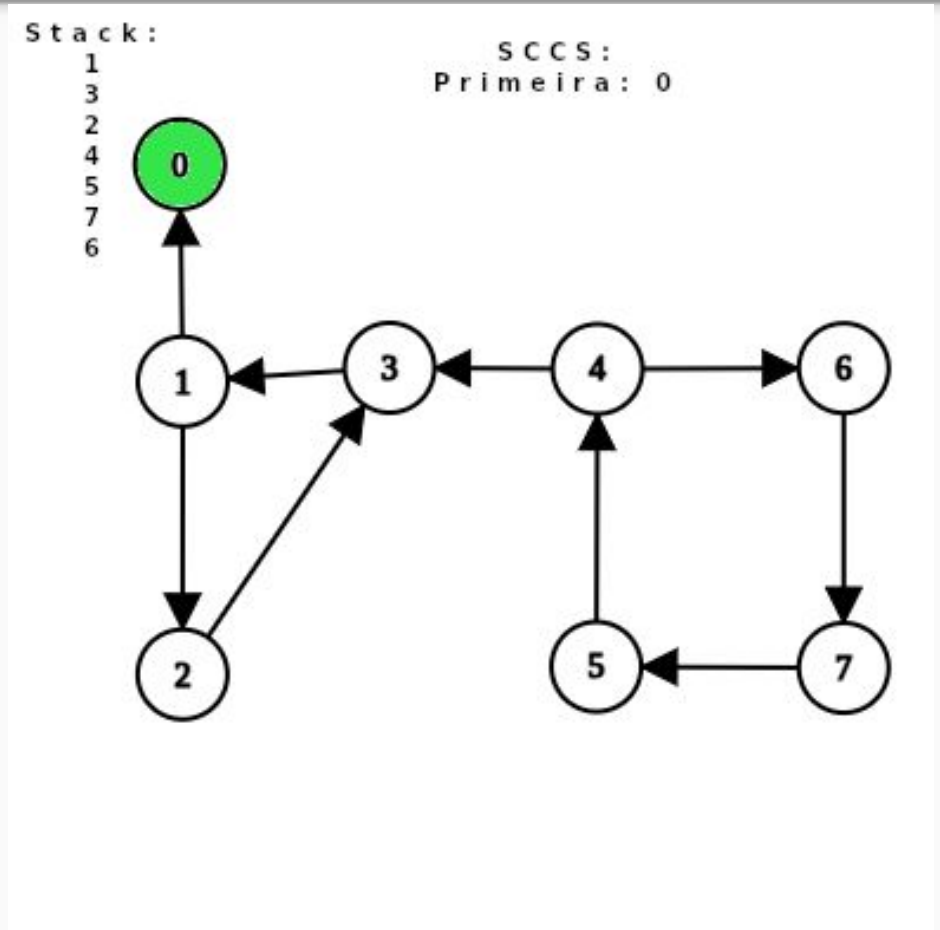
Execução Kosaraju - Segunda DFS (Grafo transposto)



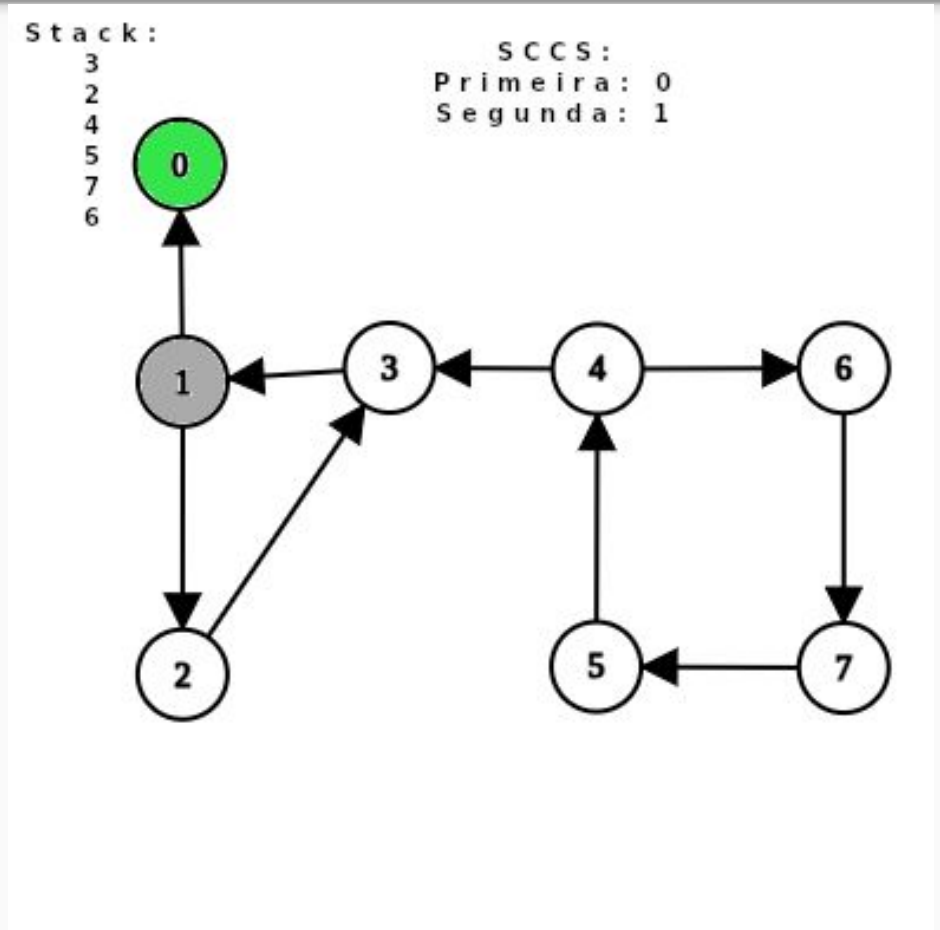
Execução Kosaraju - Segunda DFS (Grafo transposto)



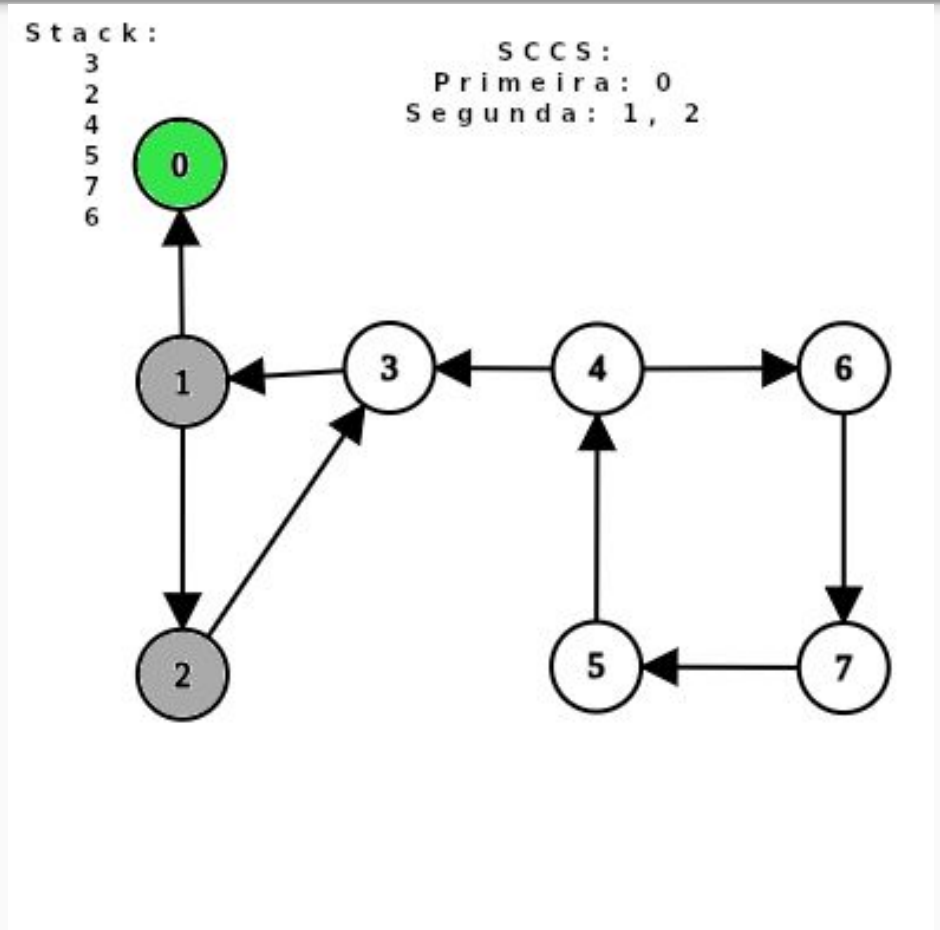
Execução Kosaraju - Segunda DFS (Grafo transposto)



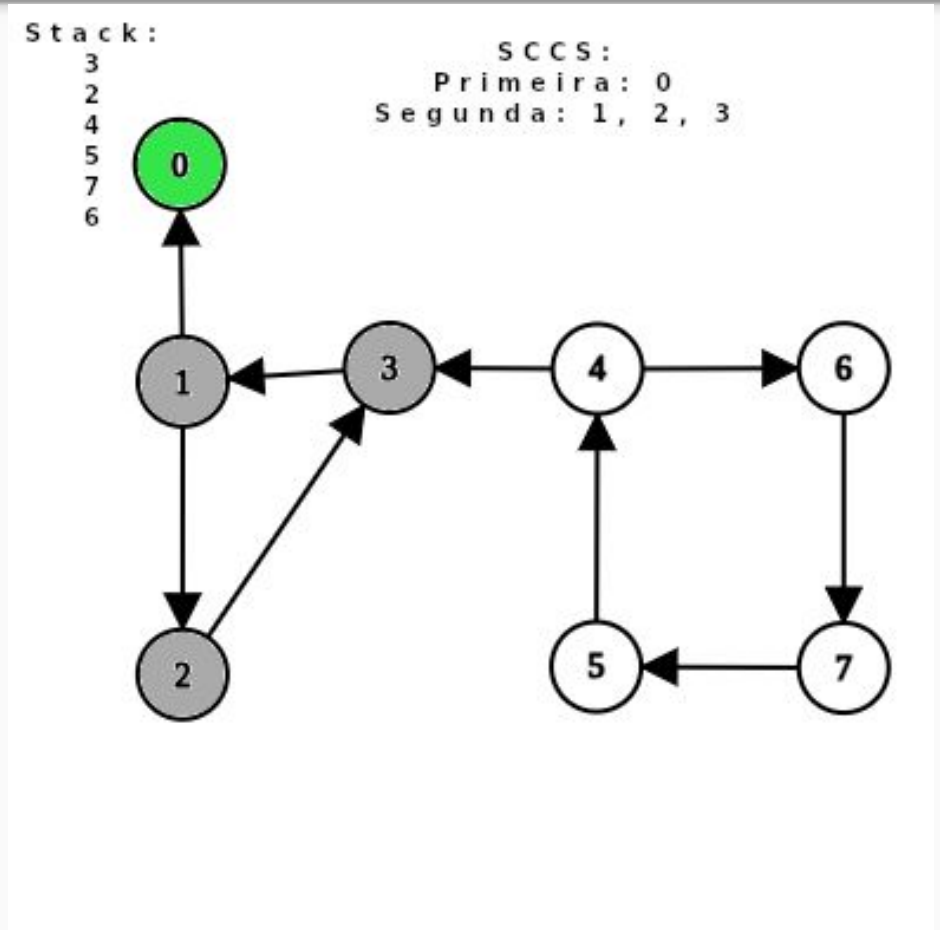
Execução Kosaraju - Segunda DFS (Grafo transposto)



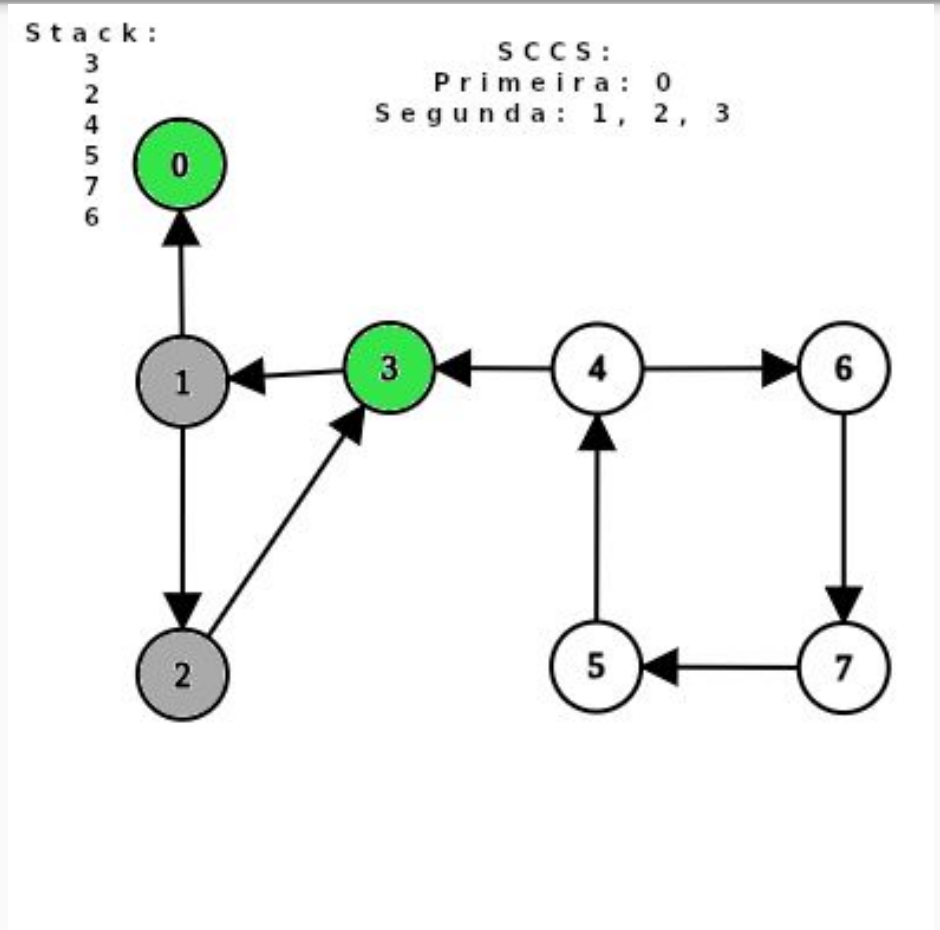
Execução Kosaraju - Segunda DFS (Grafo transposto)



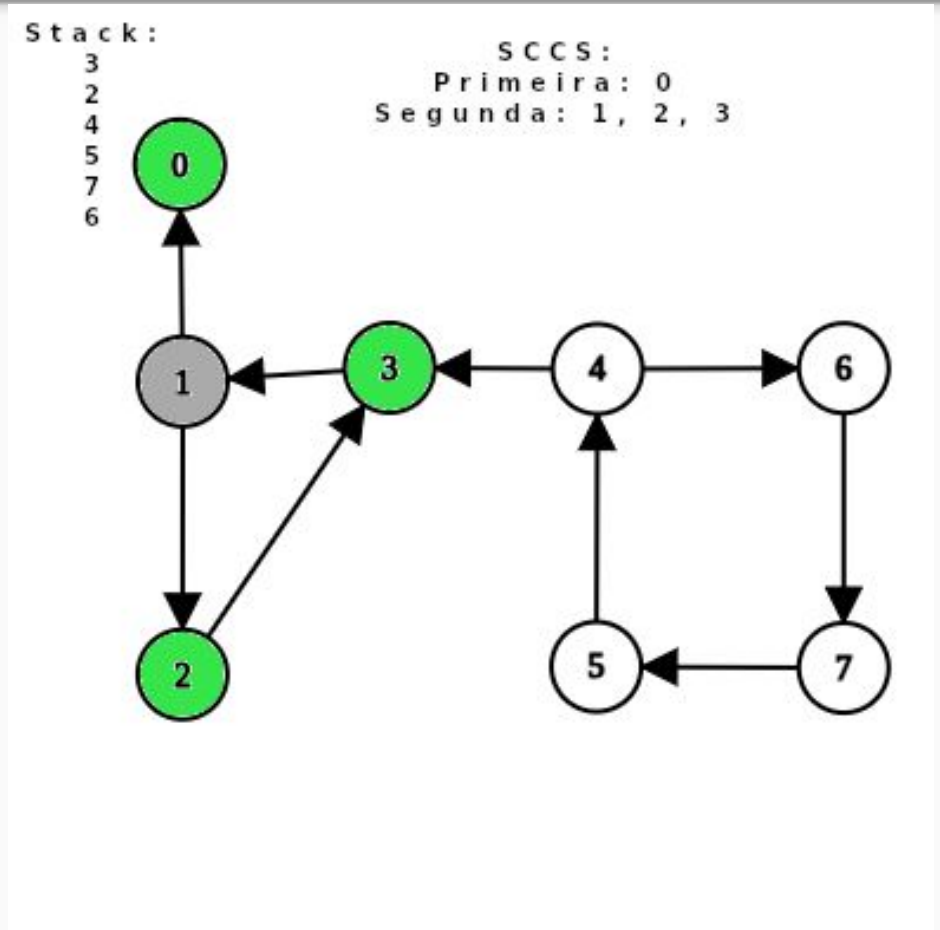
Execução Kosaraju - Segunda DFS (Grafo transposto)



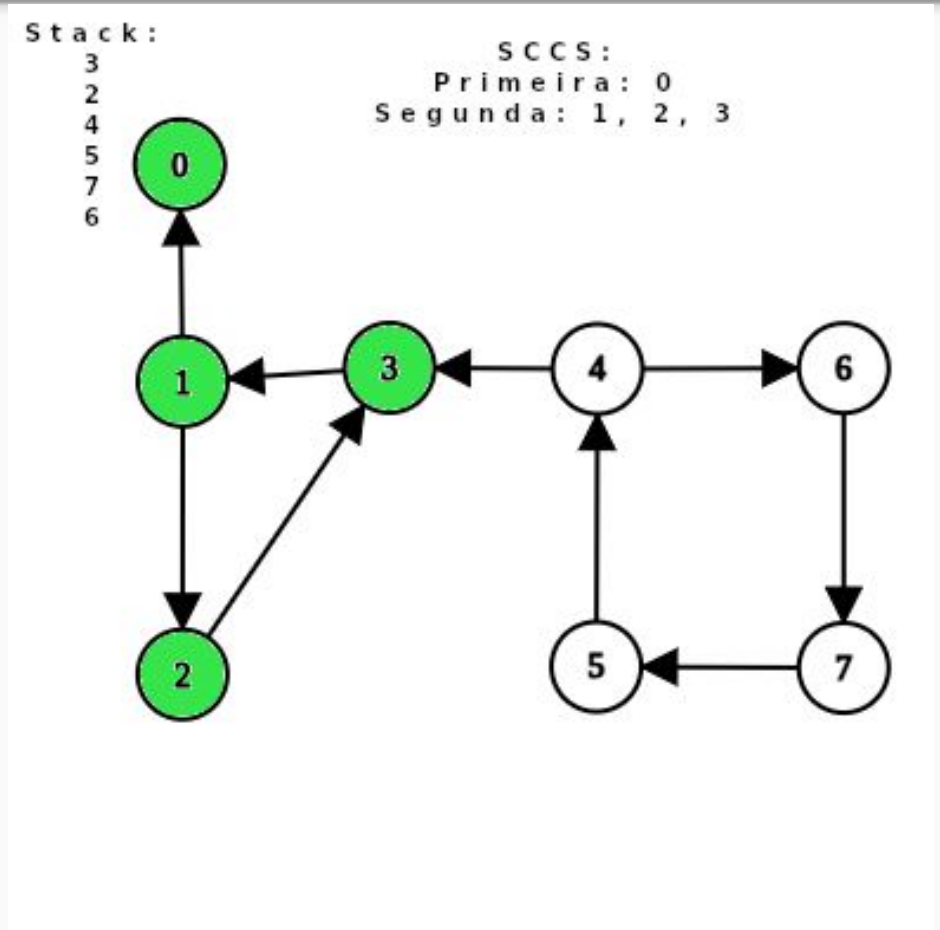
Execução Kosaraju - Segunda DFS (Grafo transposto)



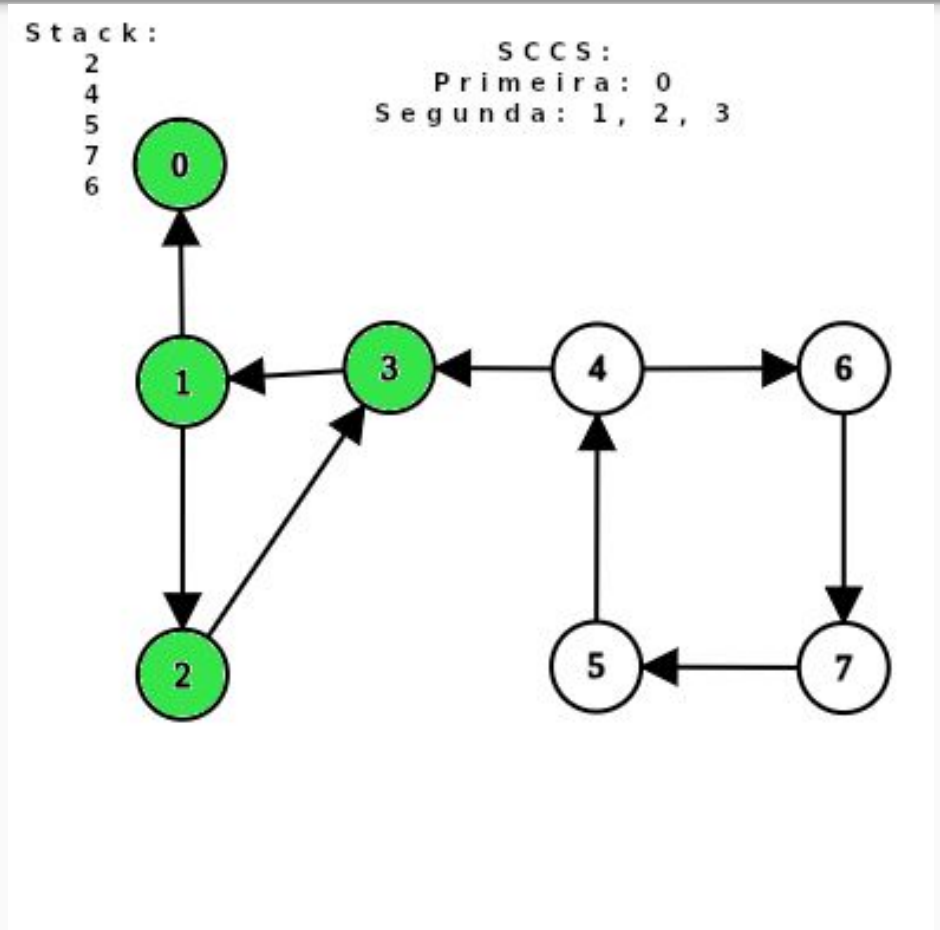
Execução Kosaraju - Segunda DFS (Grafo transposto)



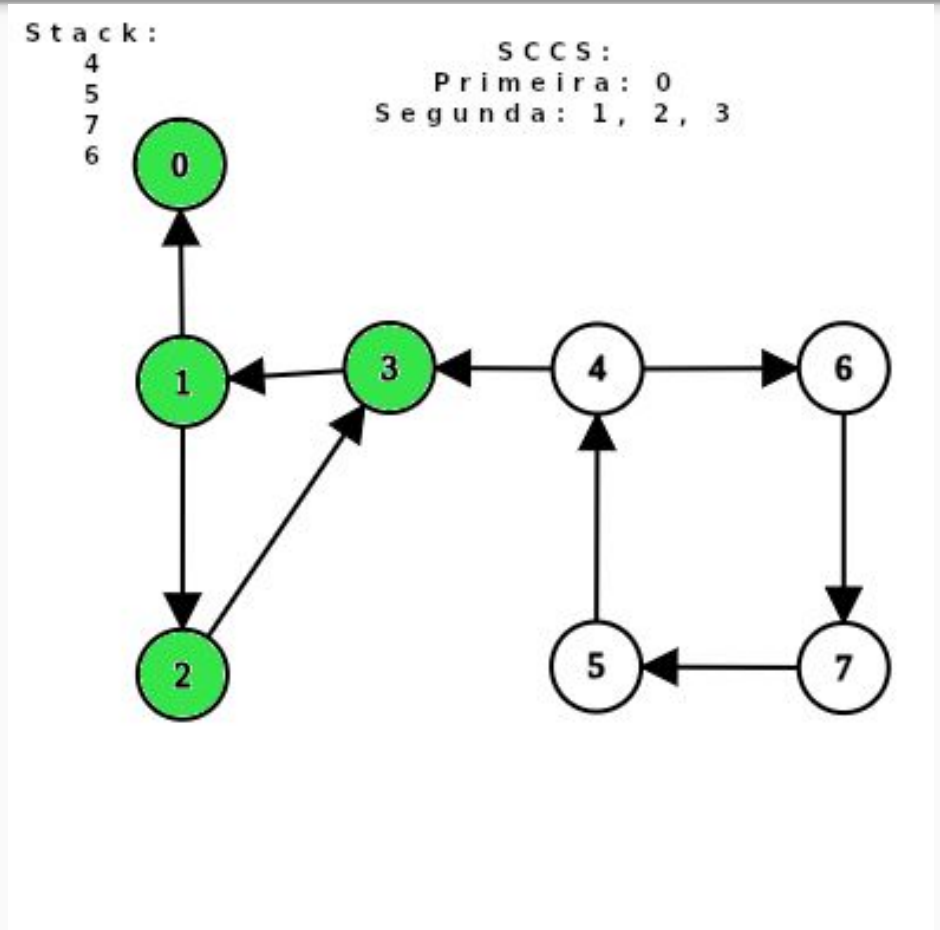
Execução Kosaraju - Segunda DFS (Grafo transposto)



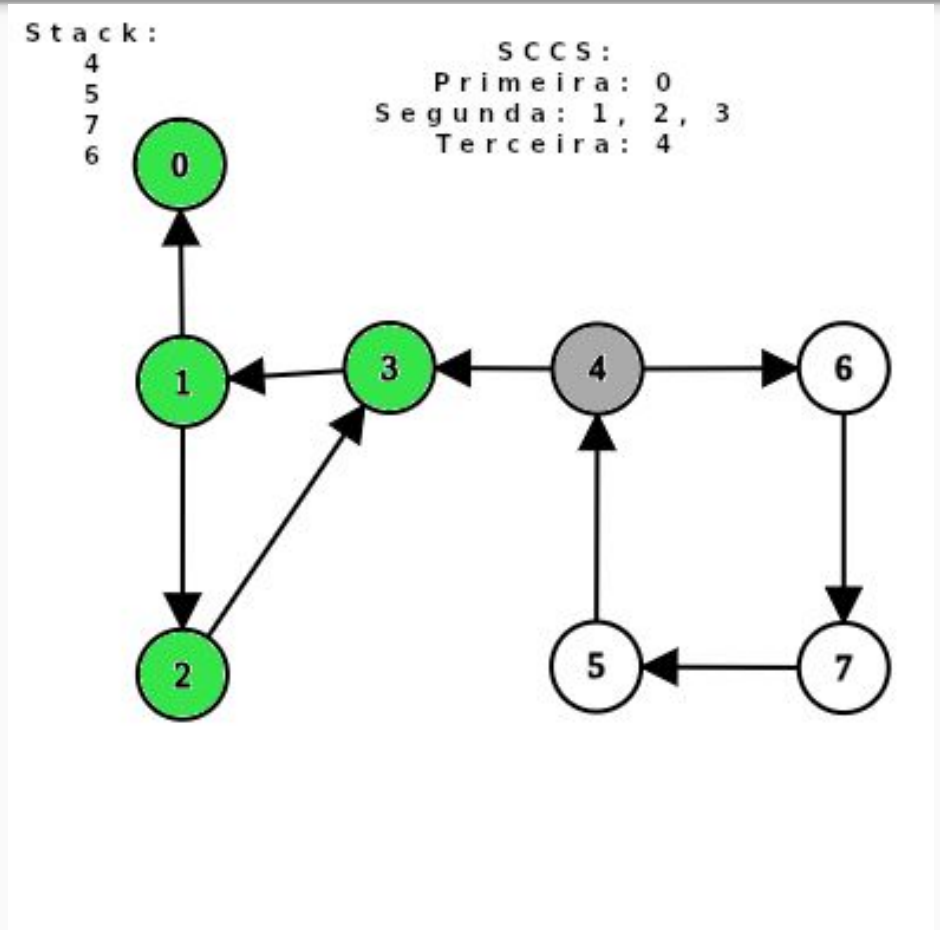
Execução Kosaraju - Segunda DFS (Grafo transposto)



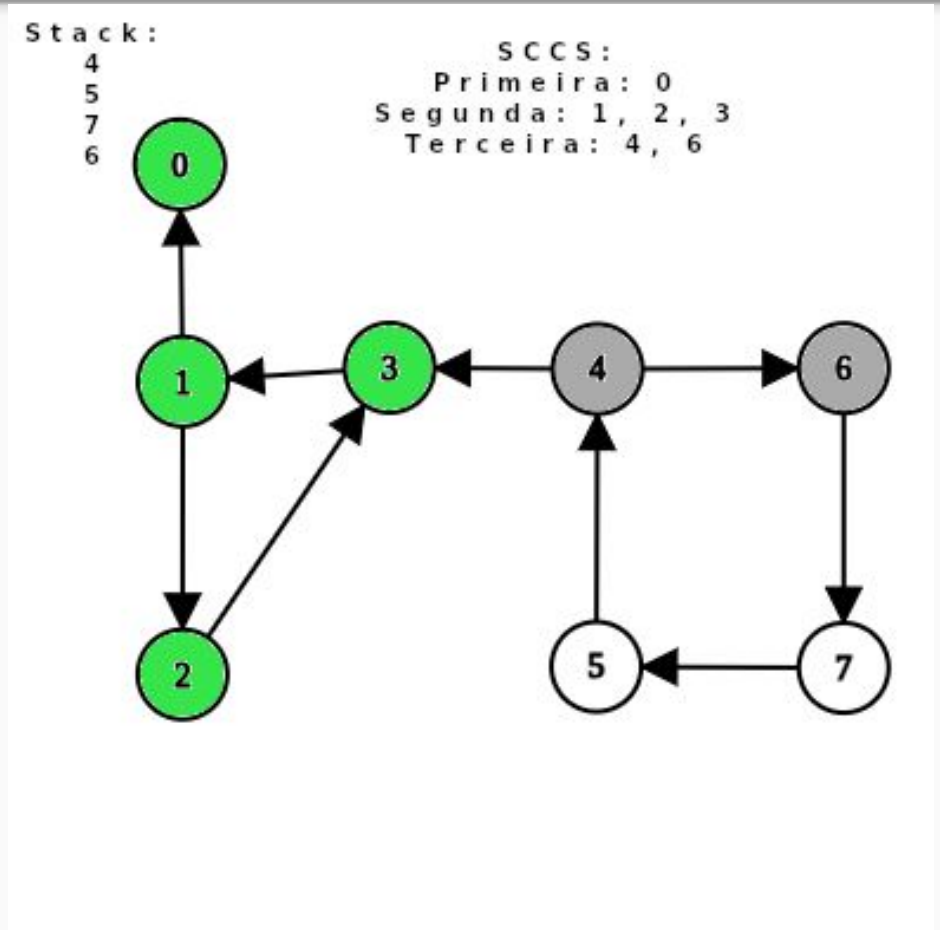
Execução Kosaraju - Segunda DFS (Grafo transposto)



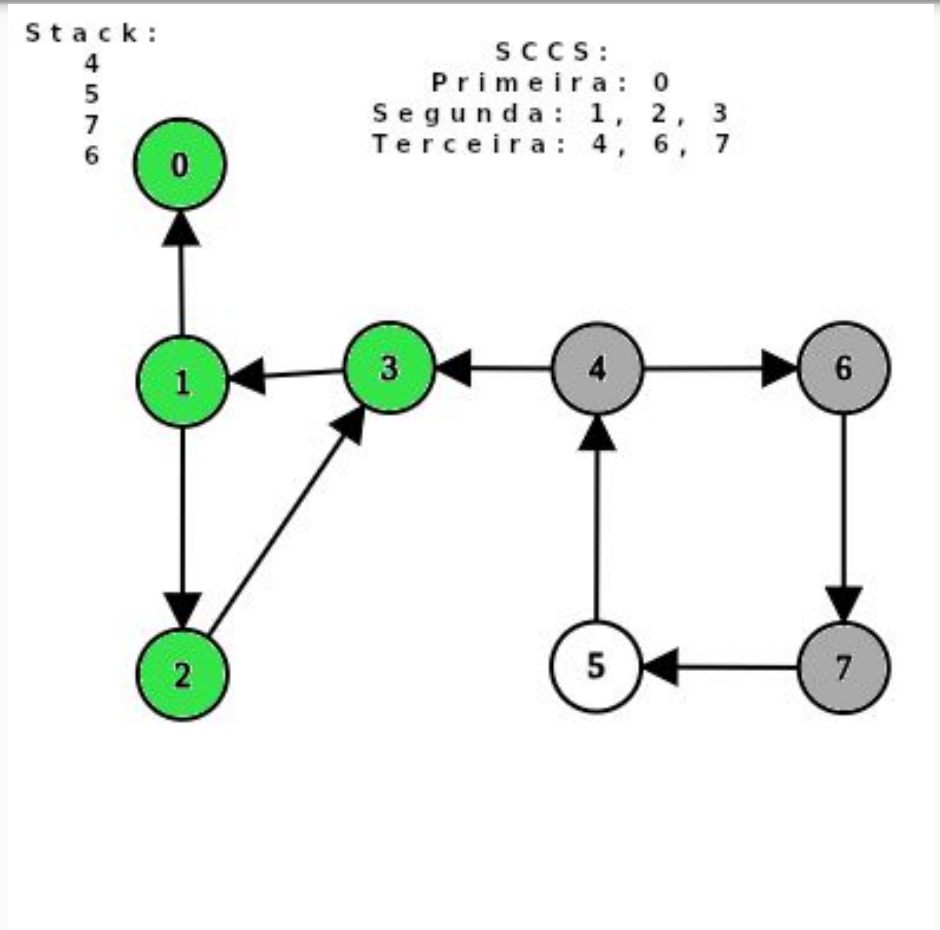
Execução Kosaraju - Segunda DFS (Grafo transposto)



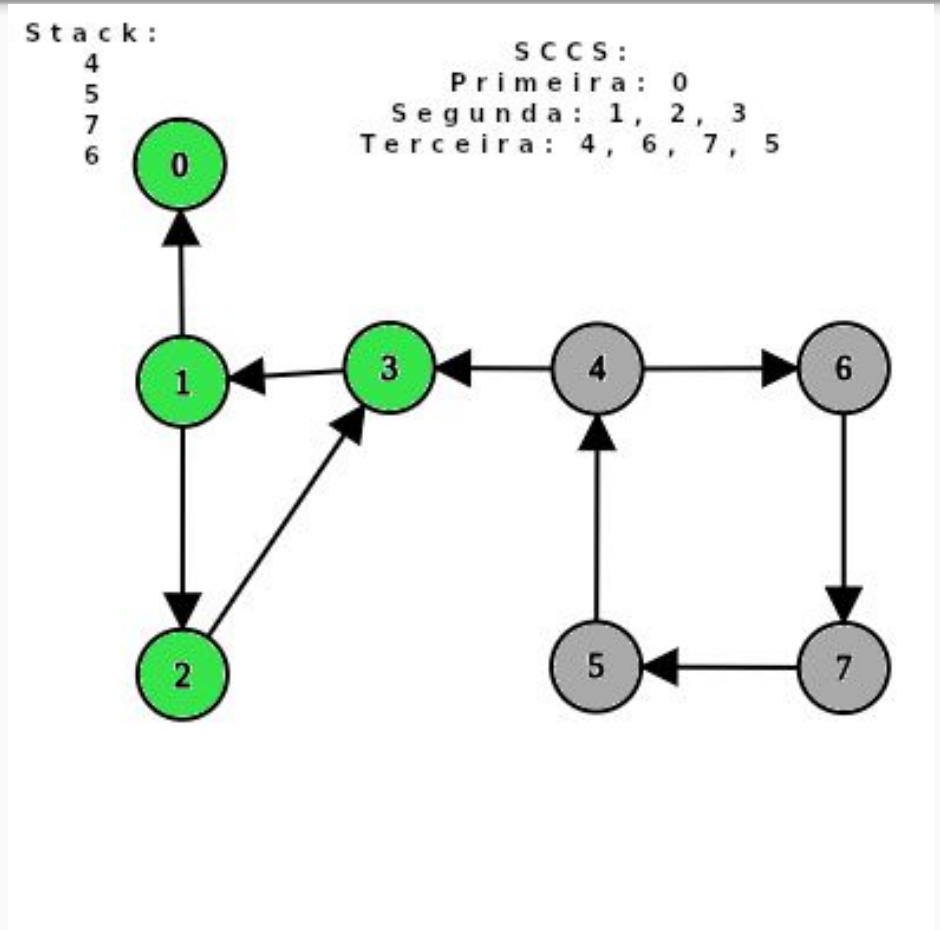
Execução Kosaraju - Segunda DFS (Grafo transposto)



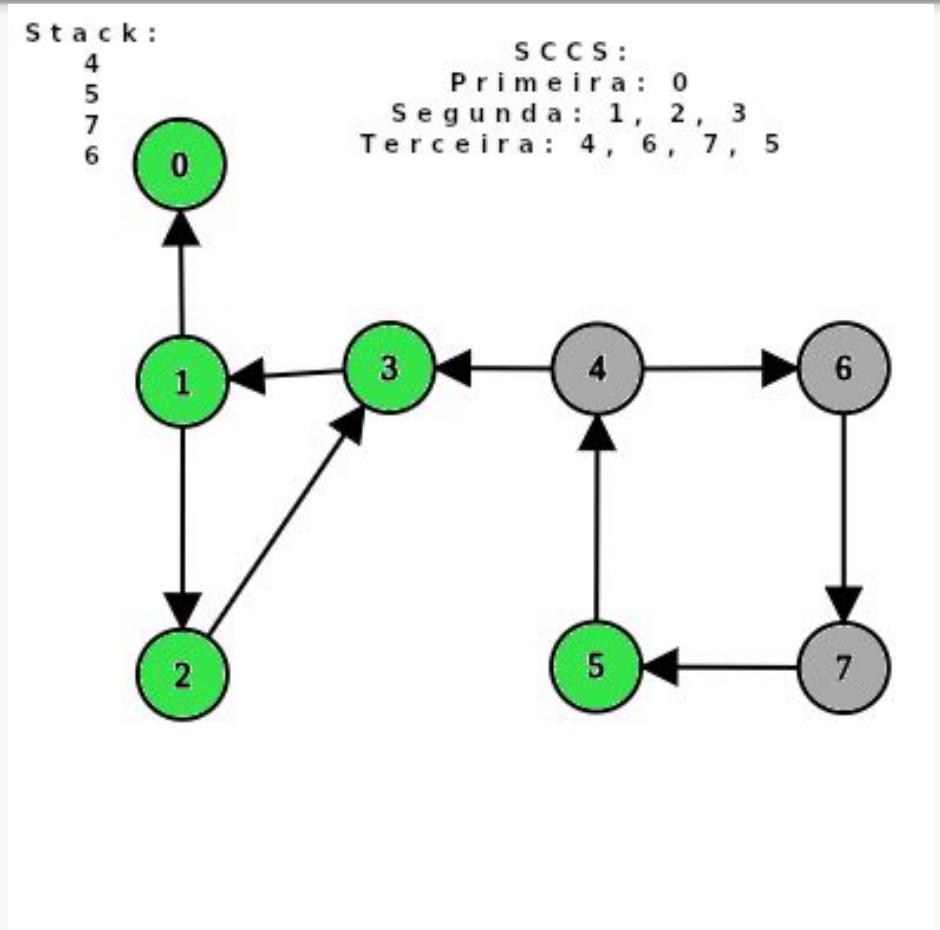
Execução Kosaraju - Segunda DFS (Grafo transposto)



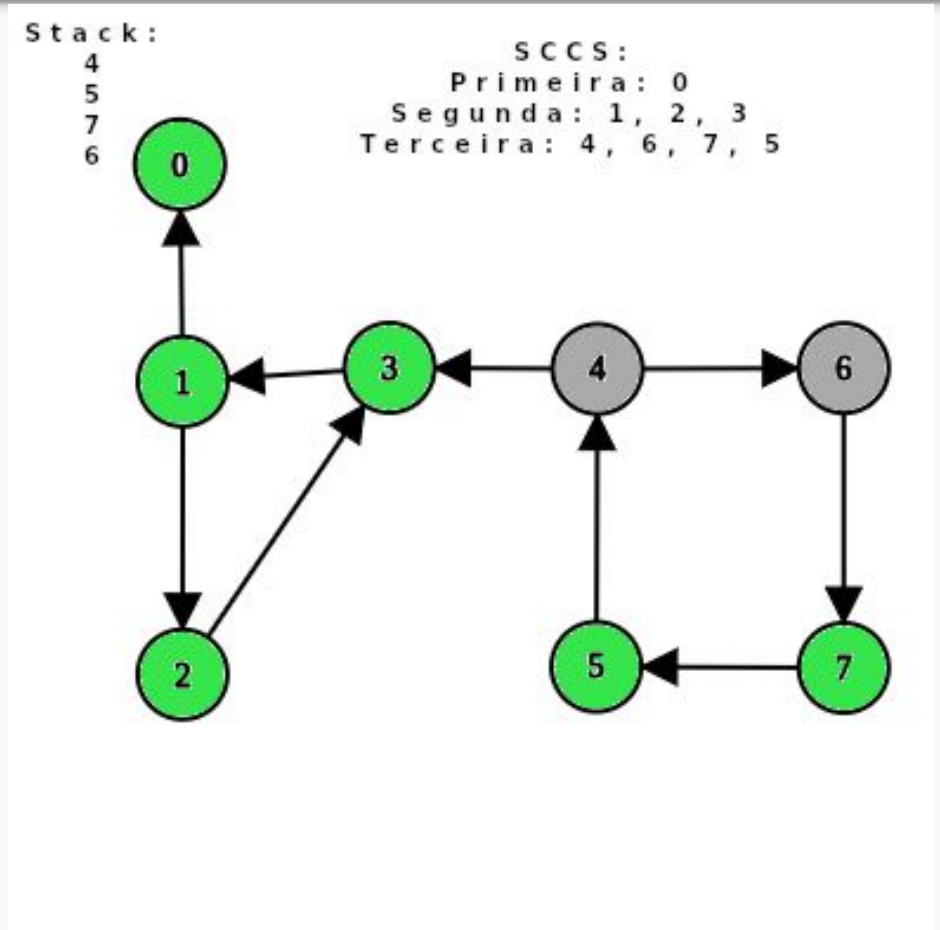
Execução Kosaraju - Segunda DFS (Grafo transposto)



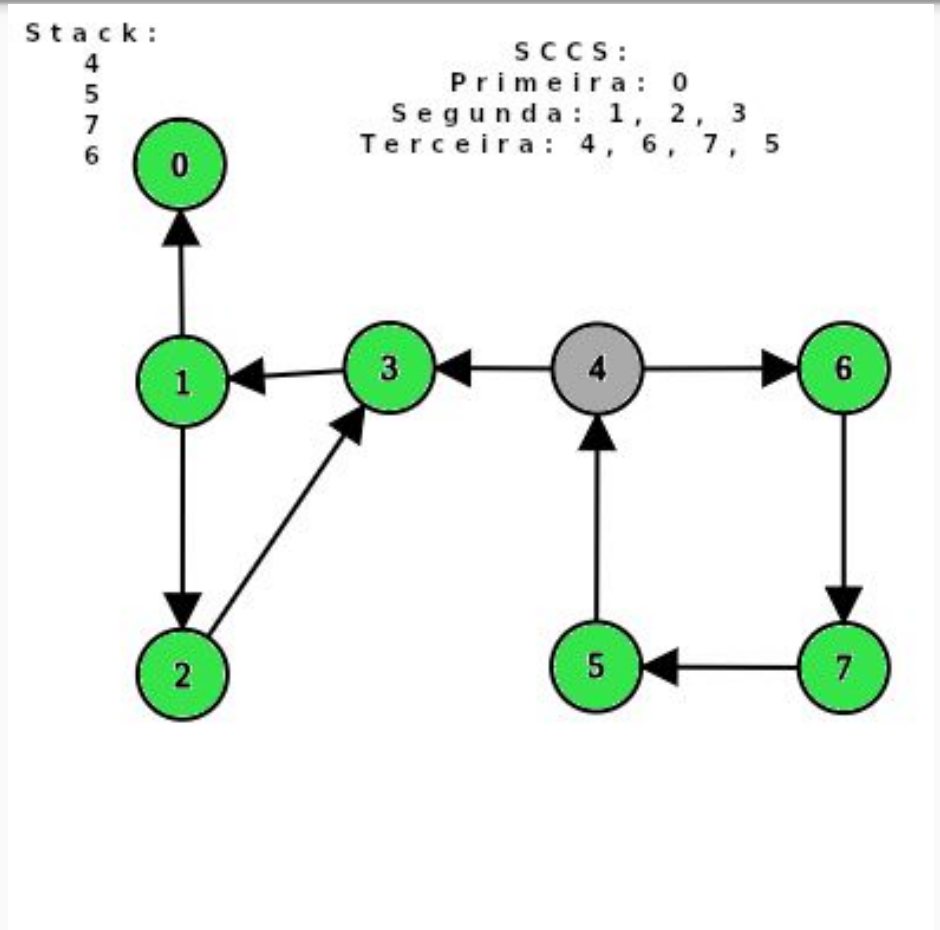
Execução Kosaraju - Segunda DFS (Grafo transposto)



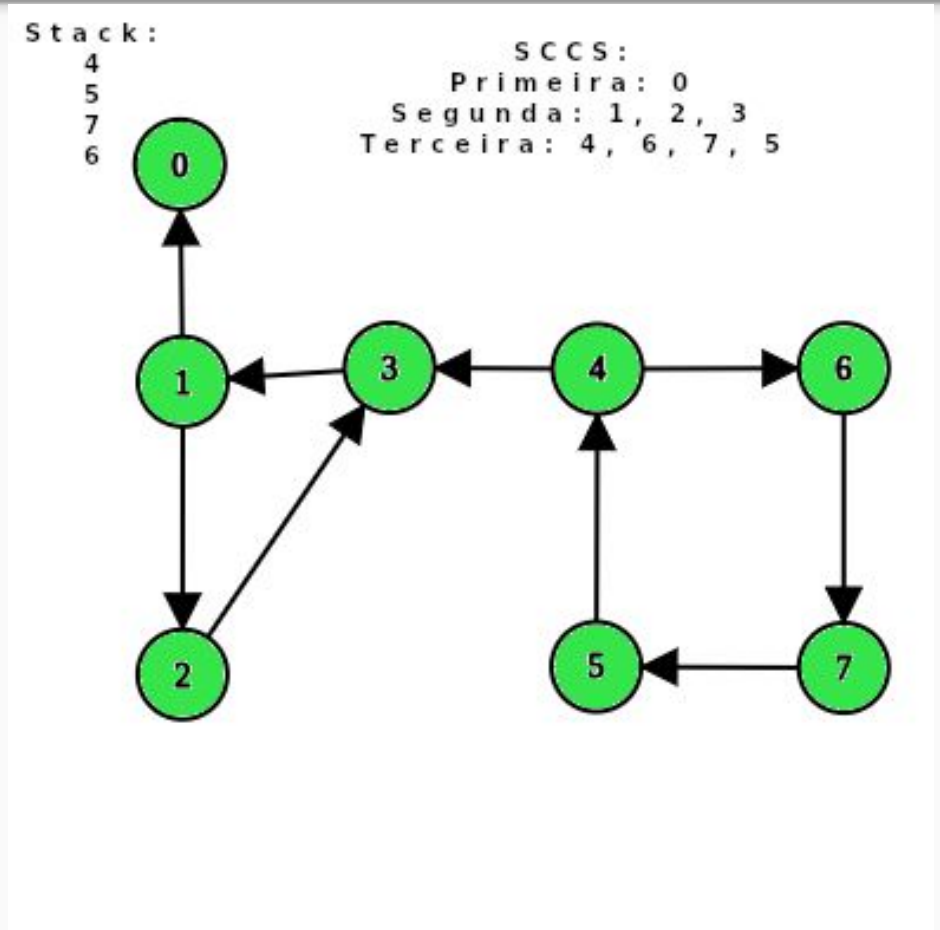
Execução Kosaraju - Segunda DFS (Grafo transposto)



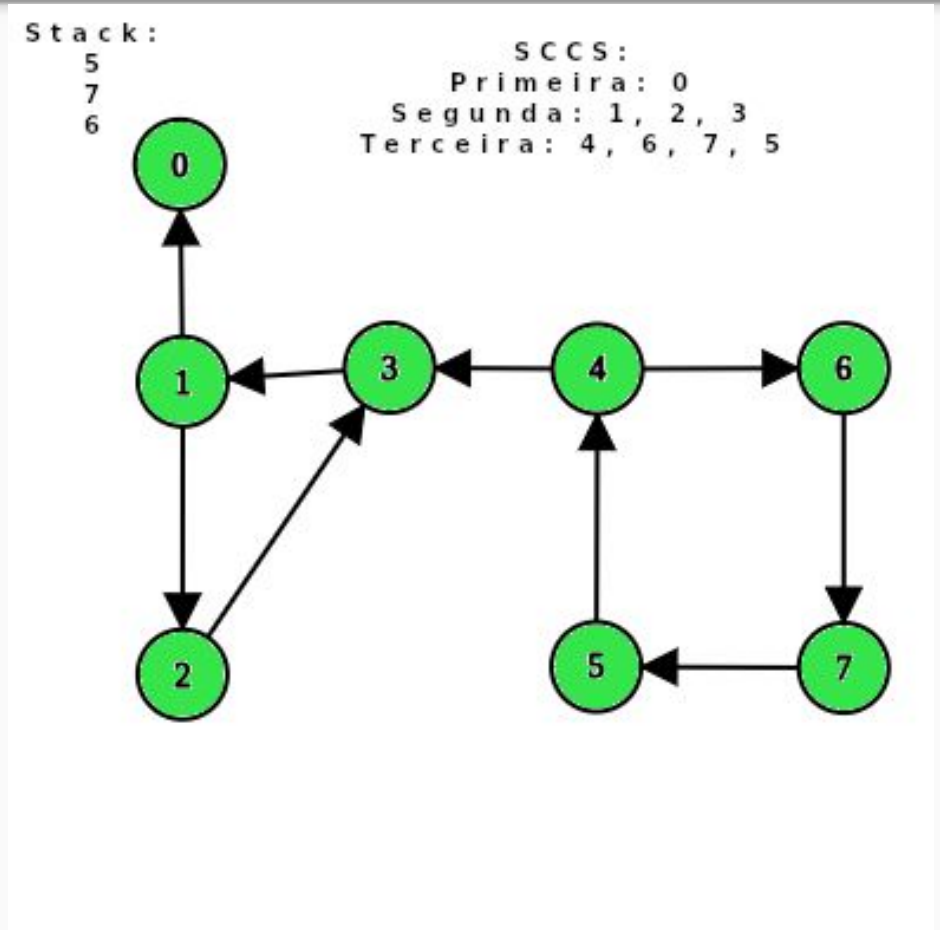
Execução Kosaraju - Segunda DFS (Grafo transposto)



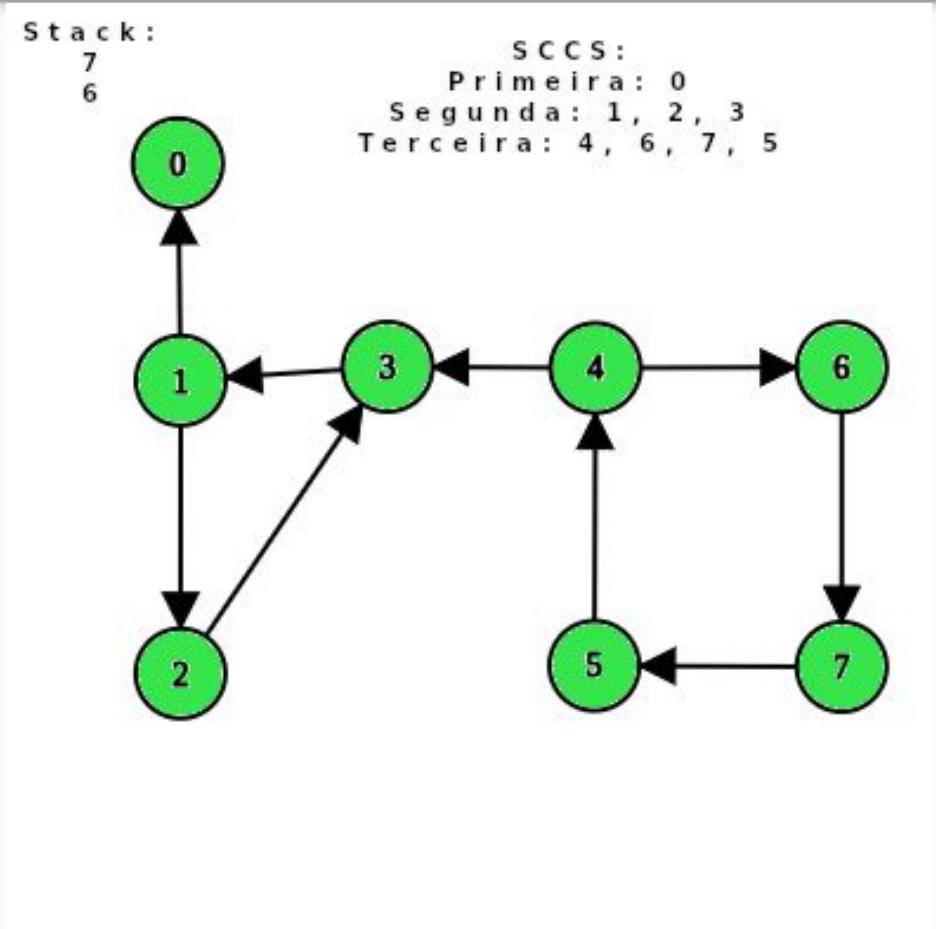
Execução Kosaraju - Segunda DFS (Grafo transposto)



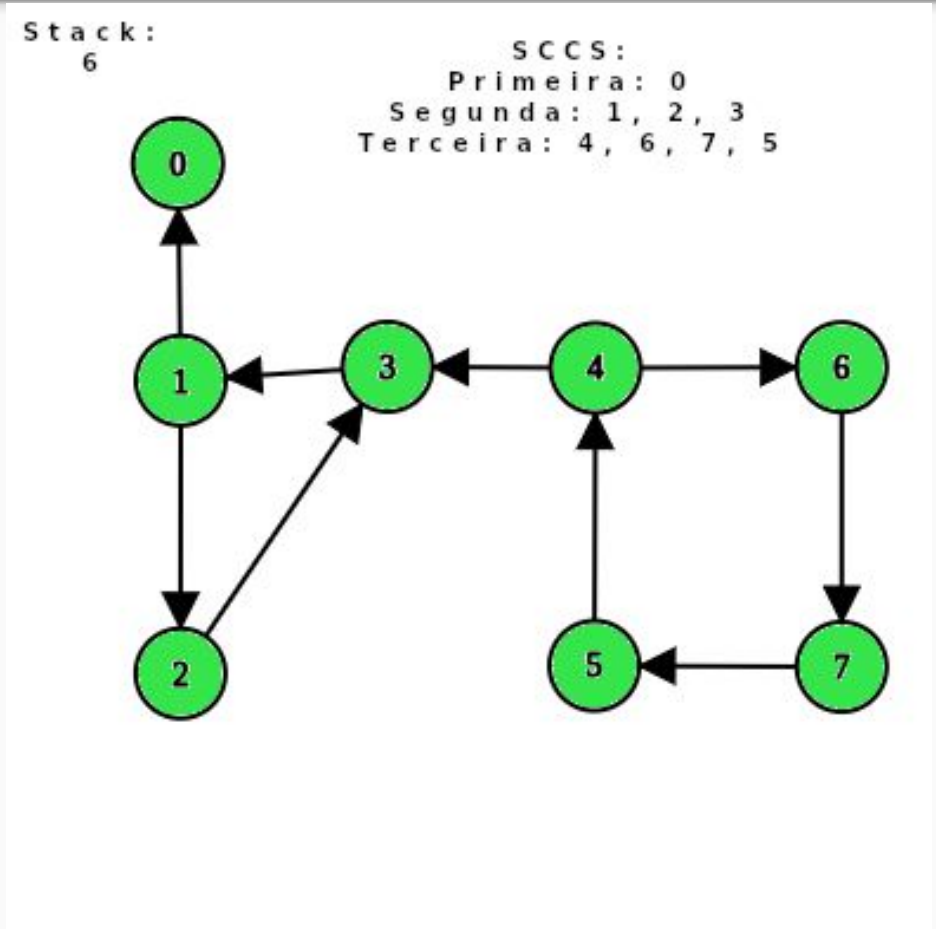
Execução Kosaraju - Segunda DFS (Grafo transposto)



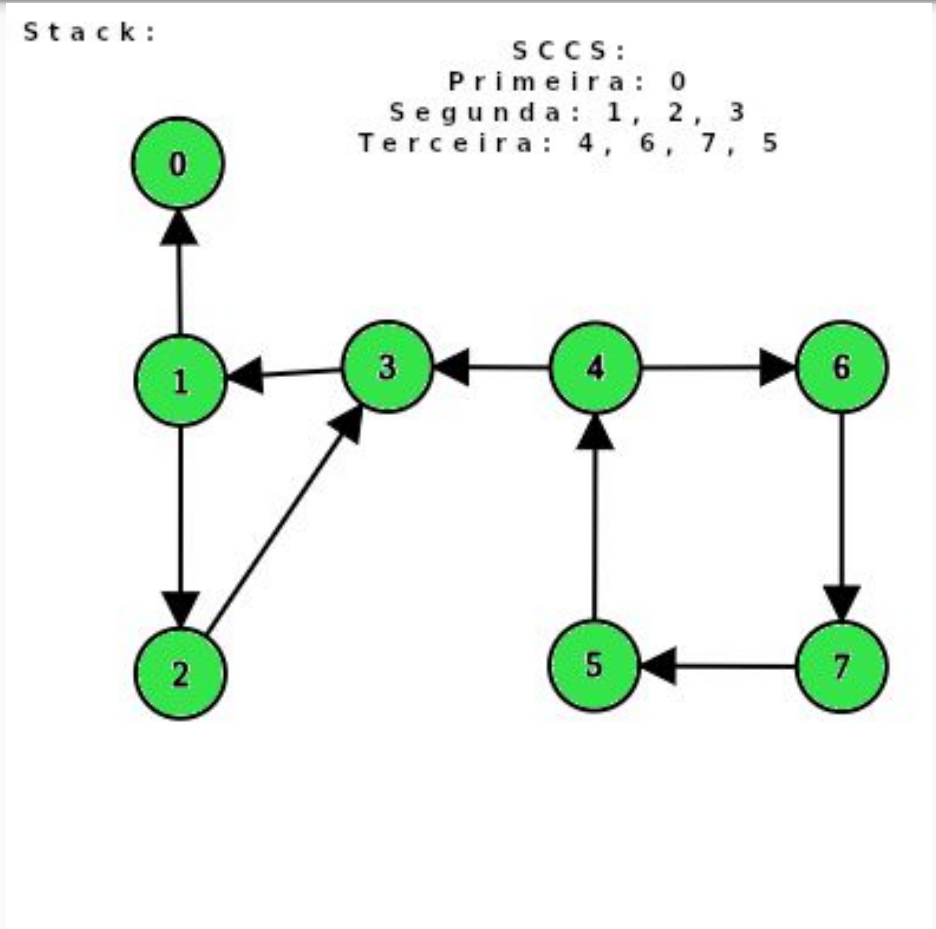
Execução Kosaraju - Segunda DFS (Grafo transposto)



Execução Kosaraju - Segunda DFS (Grafo transposto)



Execução Kosaraju - Segunda DFS (Grafo transposto)



Implementação - Algoritmo de Kosaraju

Implementação Kosaraju

DFS:

```
void dfs(int u, vector<vector<int>> & adj, vector<bool> & visited, vector<int> & visited_stack){
    visited[u] = true;
    for (int v: adj[u]){
        if (!visited[v]){
            dfs(v, adj, visited, visited_stack);
        }
    }
    visited_stack.push_back(u);
}
```

Implementação Kosaraju

Main:

```
vector<vector<int>> adj(MAXN, vector<int>()), adjT(MAXN, vector<int>());
vector<int> visited_stack, scc; vector<bool> visited(MAXN, false);

// Leitura de arestas

for (int i = 0; i < n; i++){
    if (!visited[i]) { dfs(i, adj, visited, visited_stack) };
}

visited.assign(n, false);

for (int i = n - 1; i >= 0; i--){
    int u = visited_stack[i];
    if (!visited[u]){
        scc.clear();
        dfs(u, adjT, visited, scc);
        for (int u : scc) { cout << u << endl; }
    }
}
```

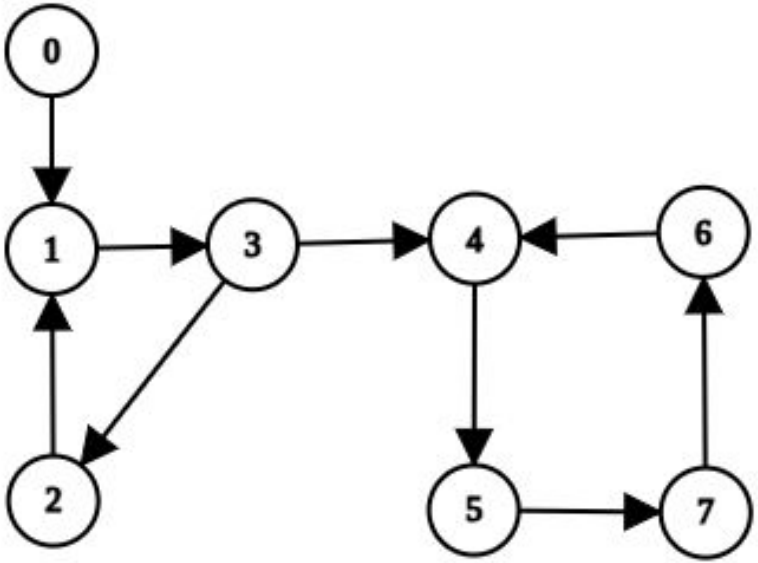
Algoritmo de Tarjan

Tarjan

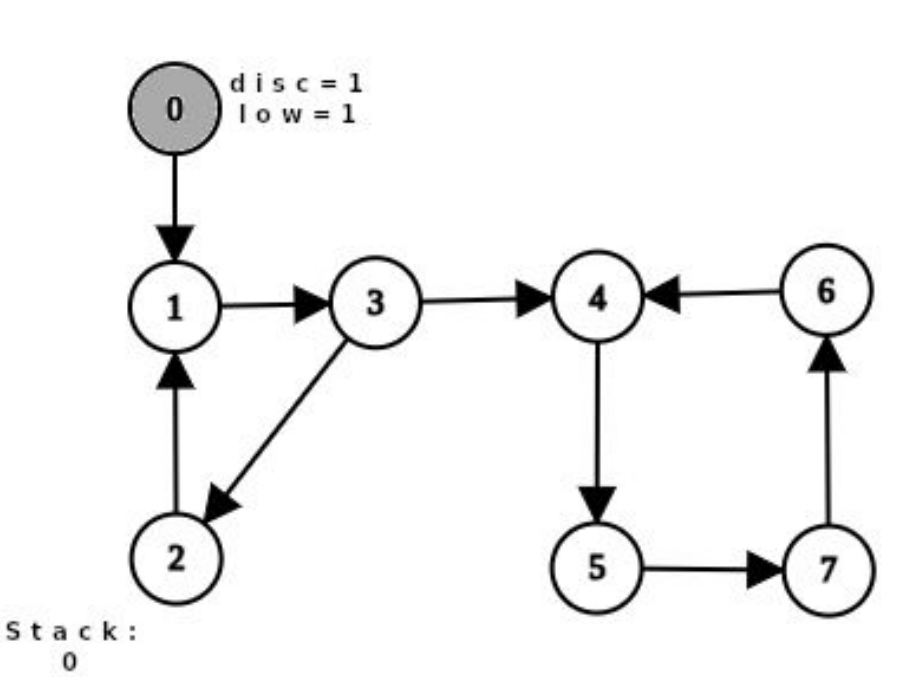
- SCCs formam subtrees em spanning tree da DFS
- Calcular para cada vértice durante a DFS:
  - *discovery(u)* - Valor da iteração da DFS da primeira visita ao vértice ***u***;
  - *low(u)* - Menor valor de *discovery* alcançável na *spanning subtree* da DFS (sem considerar back edges para o vértice pai de ***u***);
- Ao visitar um vértice ***u*** pela primeira vez, inicializaremos os valores de *discovery(u)* e *low(u)* com o valor da iteração atual, marcaremos o vértice como visitado e vamos inserir ***u*** em uma pilha ***P***;
- Após visitar cada vértice ***v*** adjacente não visitado, iremos atualizar o valor de *low(u)* com o menor valor entre *low(u)* e *low(v), se o vértice* ***v*** estiver marcado como visitado;
- Se após visitar todos os adjacentes de ***u***, *low(u)==discovery(u)*, então ***u*** é a raíz de uma subtree e começo de uma SCC
  - Vamos retirar os vértices da pilha ***P*** e **marcaremos o vértice como não vistado**, até encontrarmos o vértice ***u*** na pilha. Os vértices vistos fazem parte de uma SCC.

Execução - Algoritmo de Tarjan

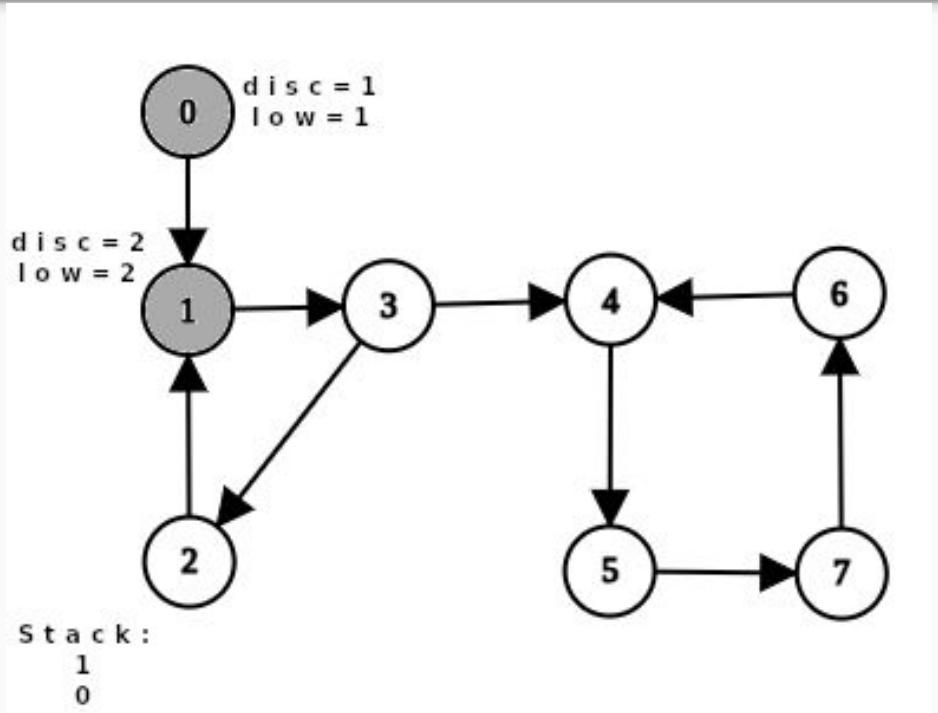
Execução Tarjan



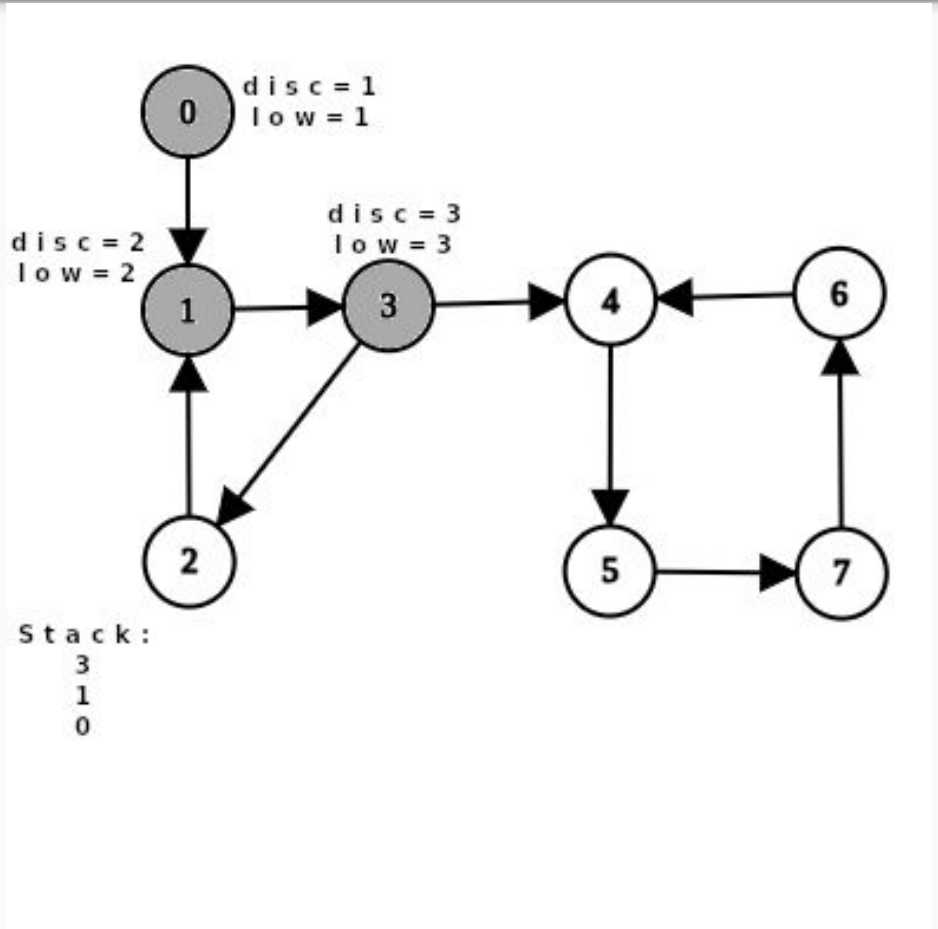
Execução Tarjan



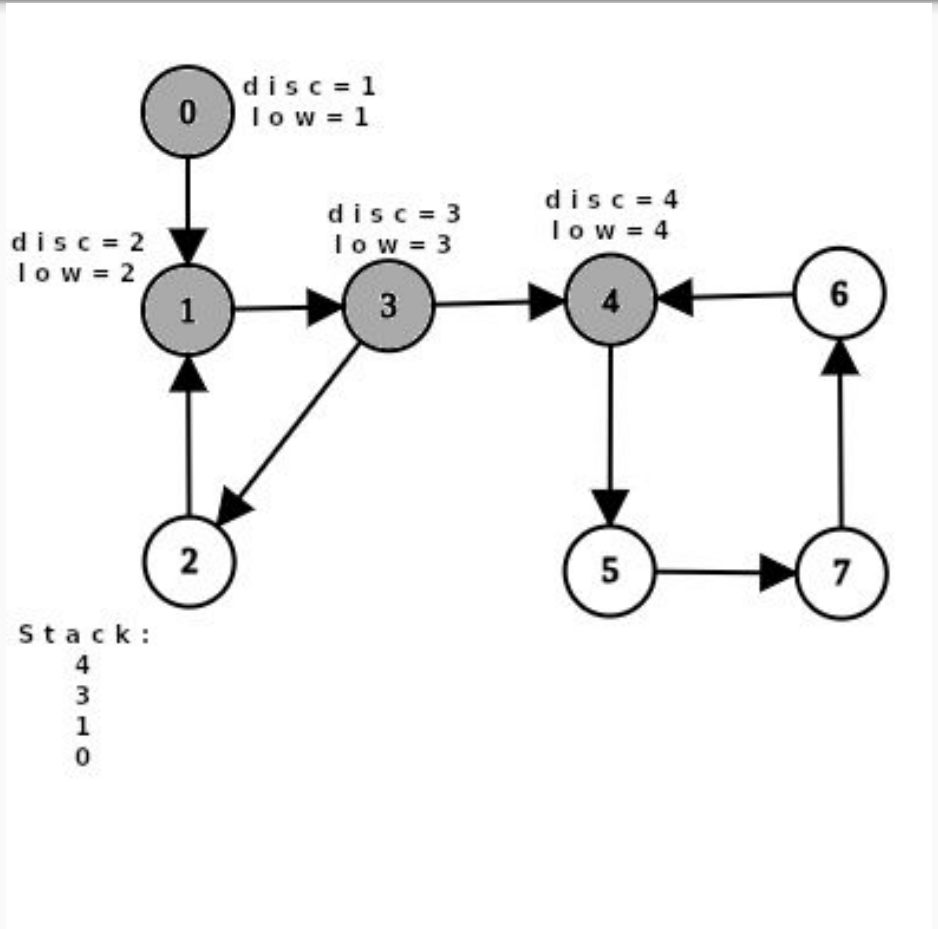
Execução Tarjan



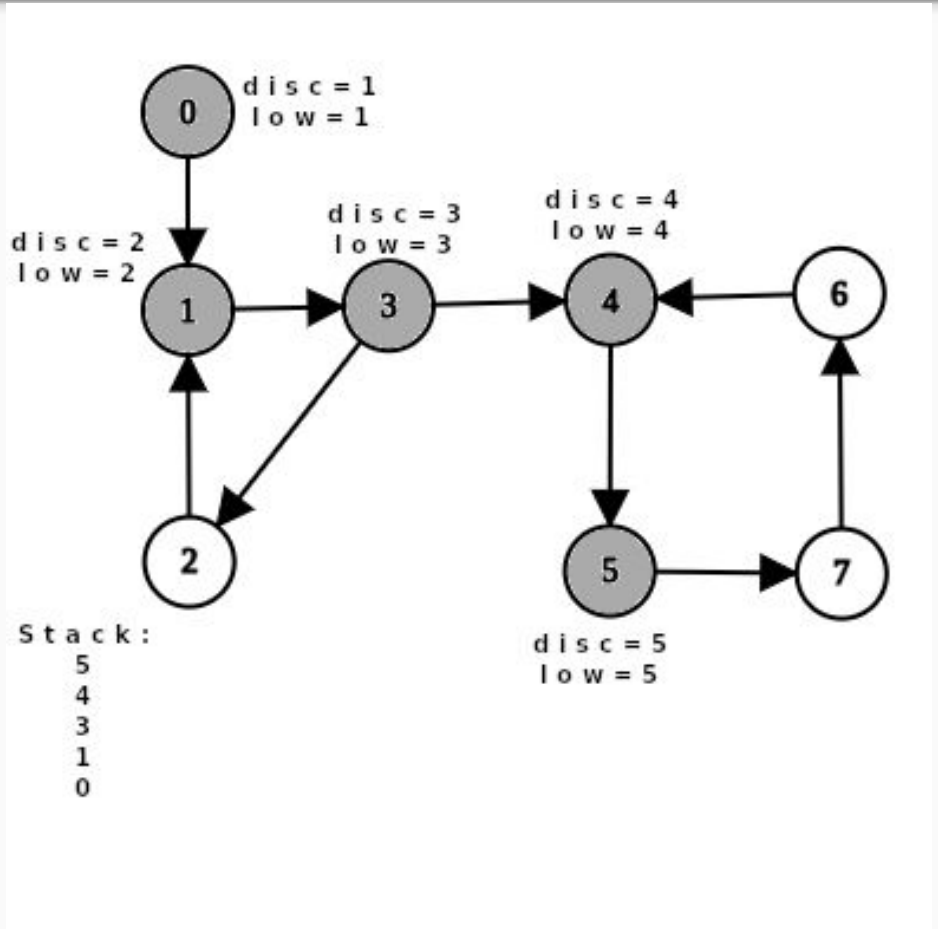
Execução Tarjan



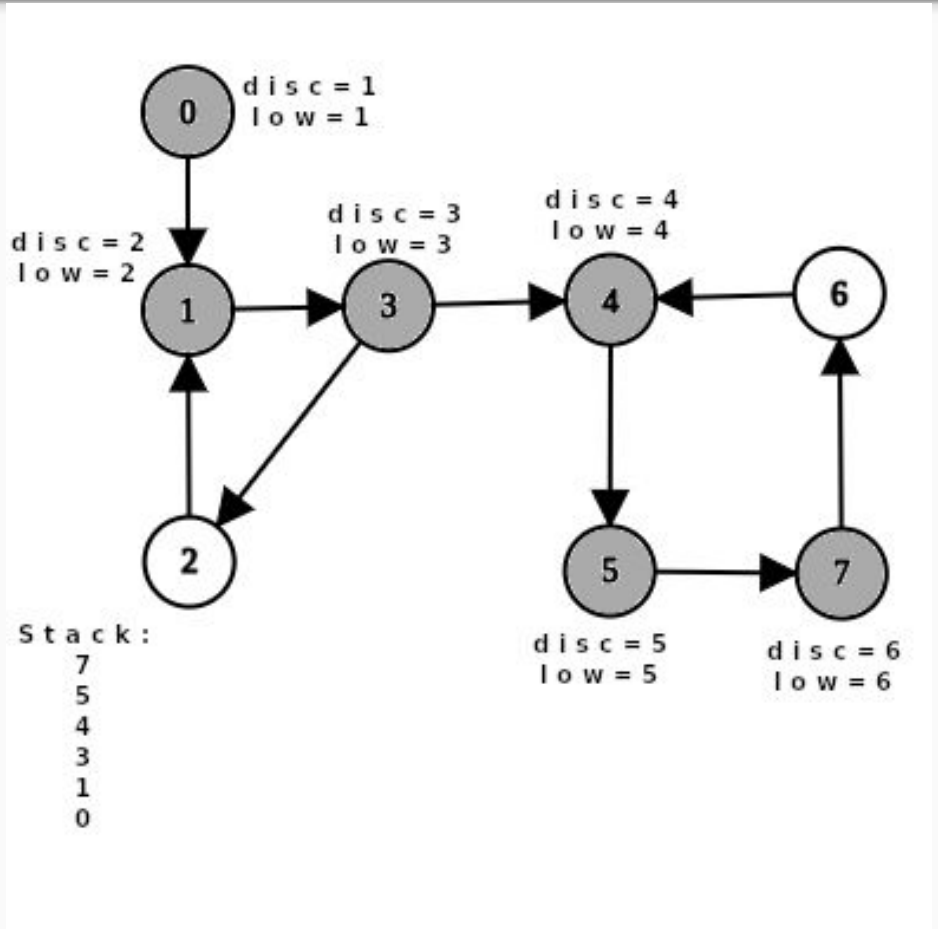
Execução Tarjan



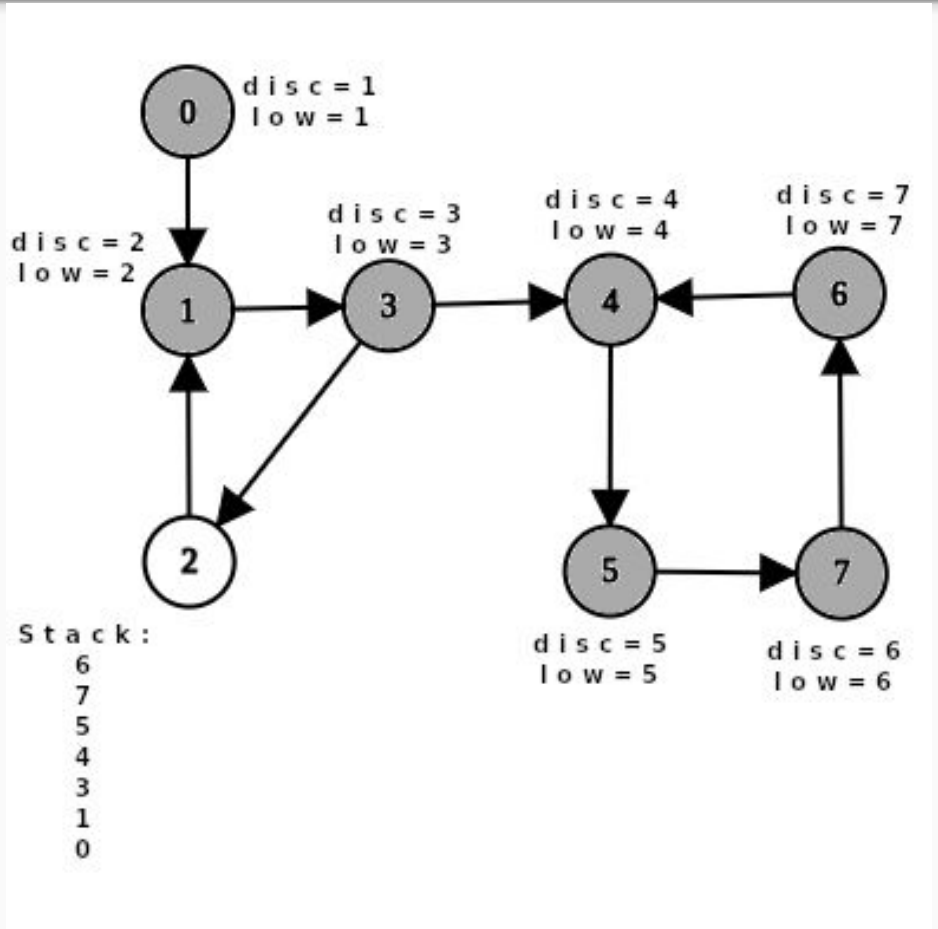
Execução Tarjan



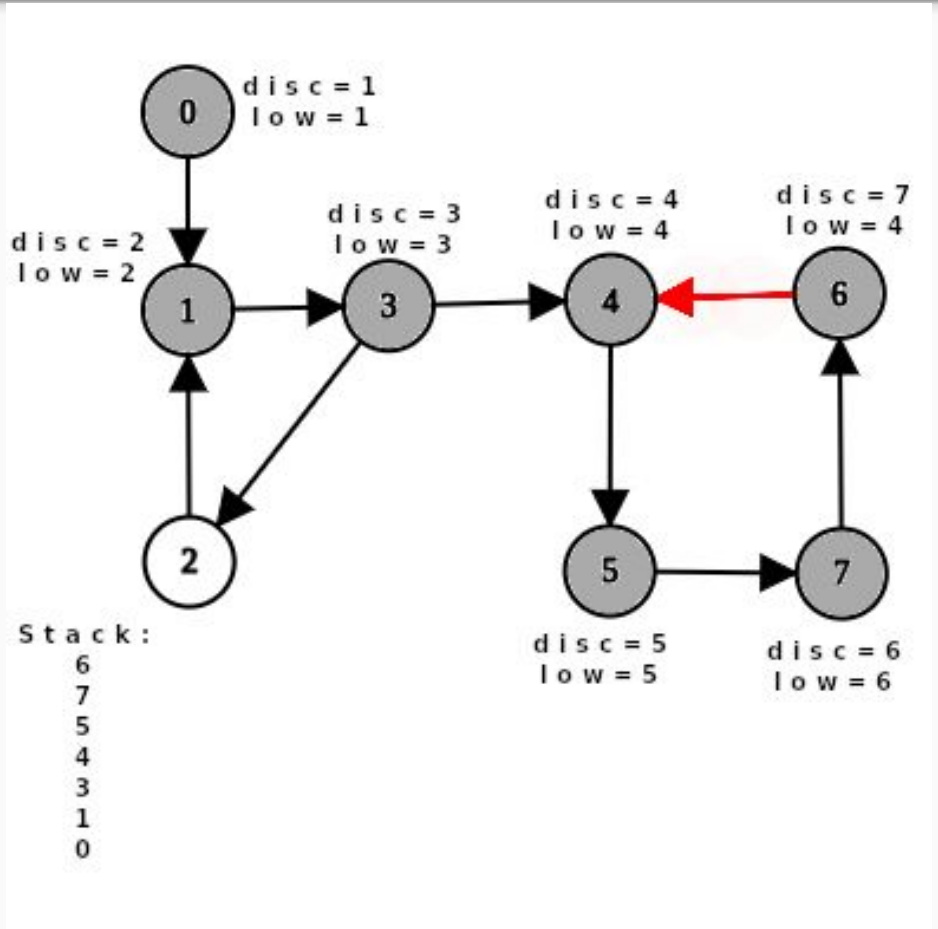
Execução Tarjan



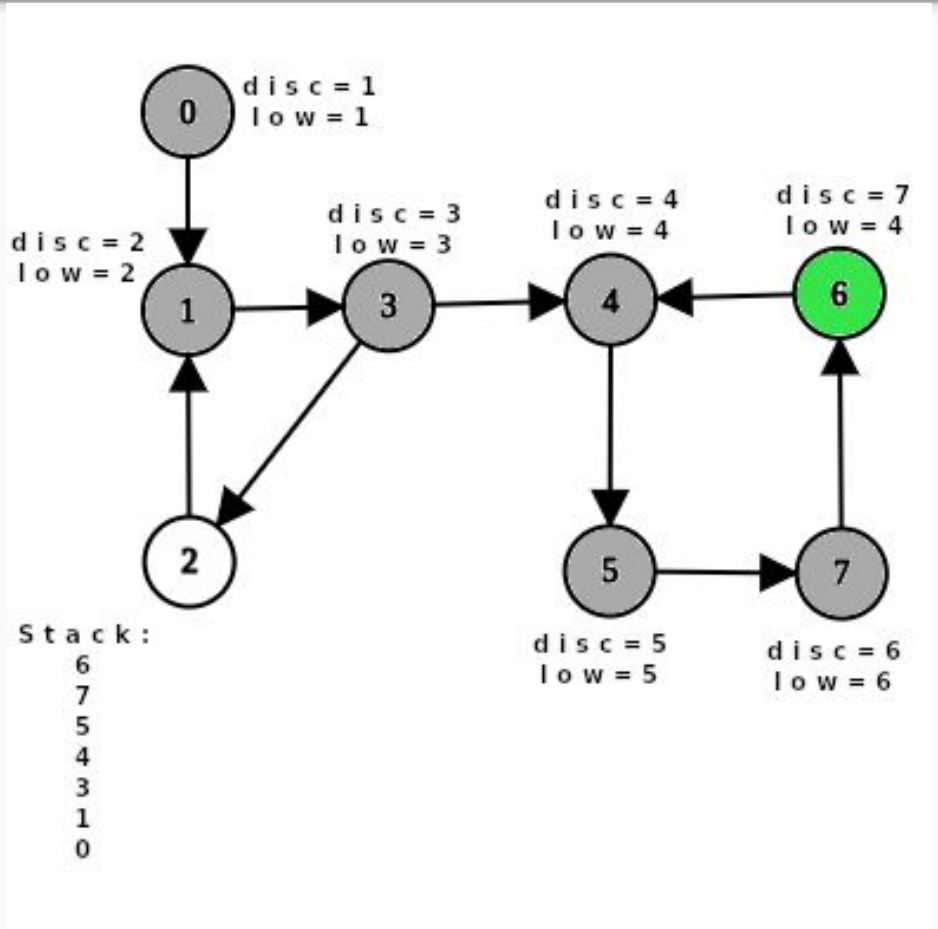
Execução Tarjan



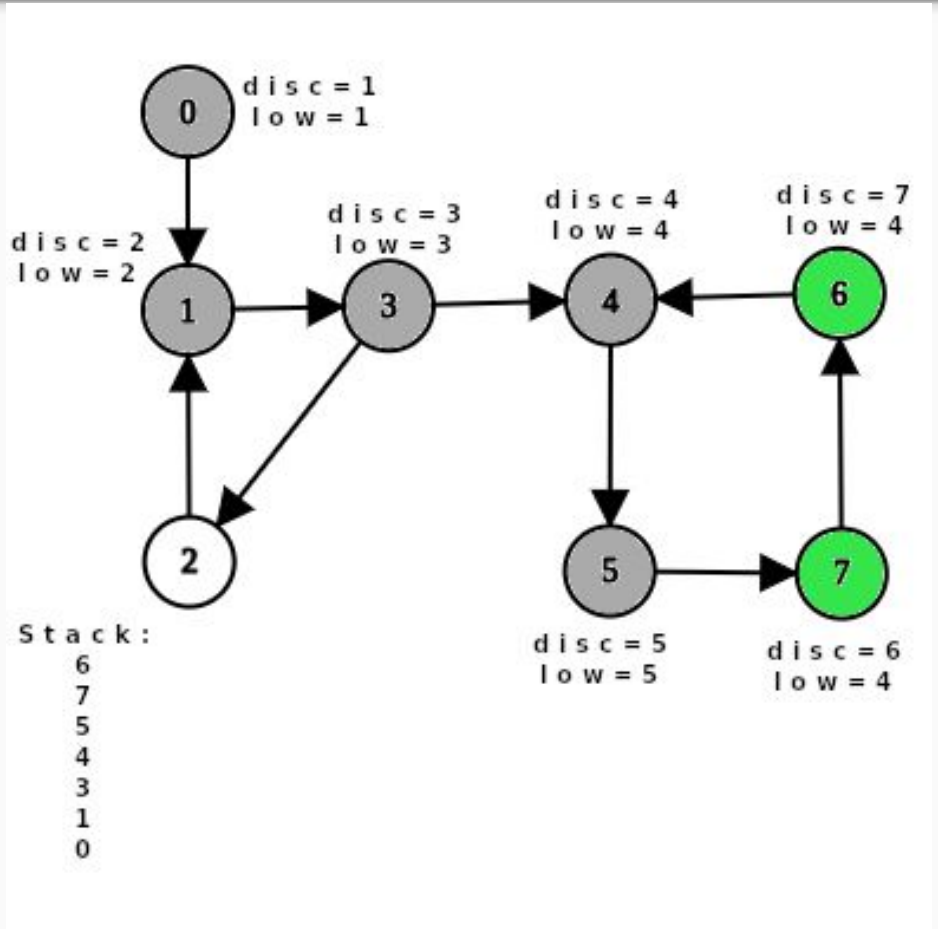
Execução Tarjan



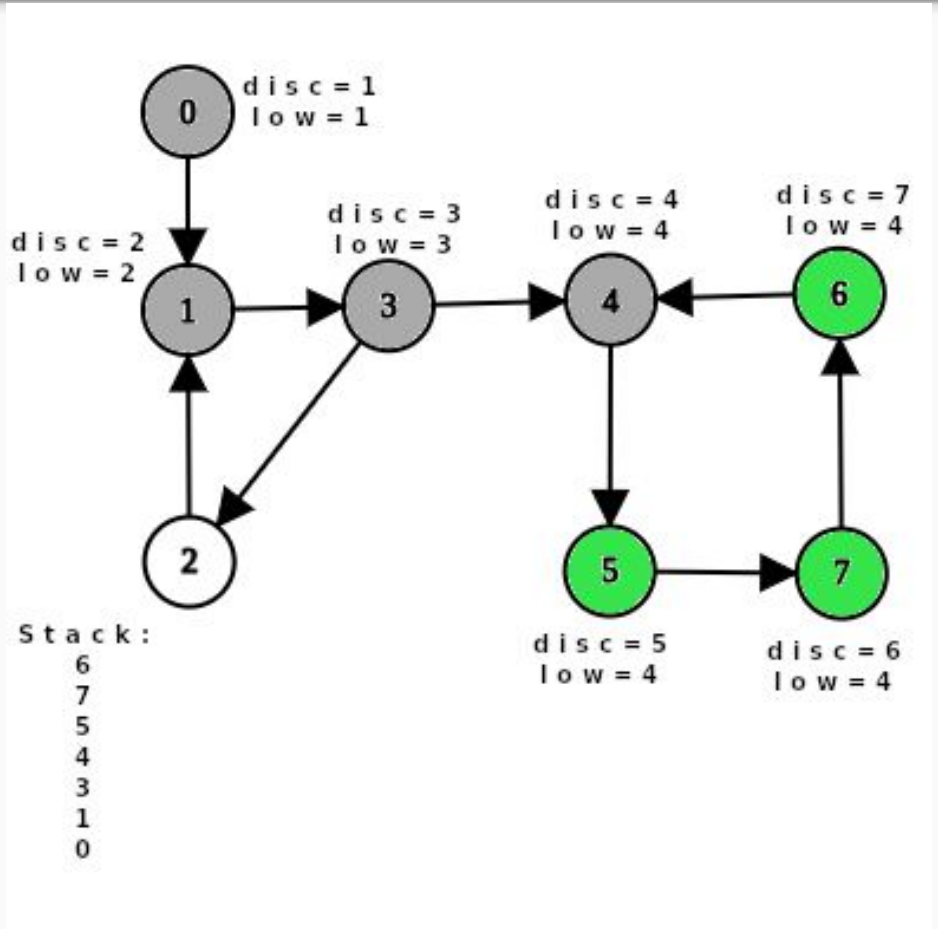
Execução Tarjan



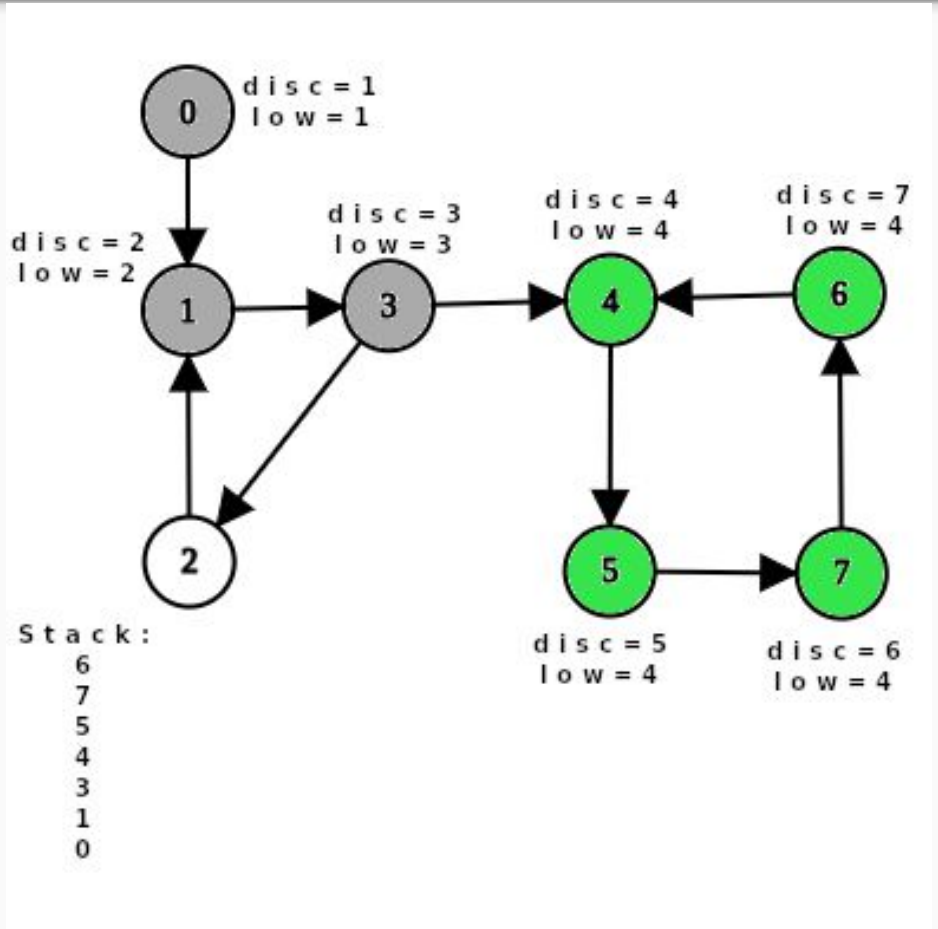
Execução Tarjan



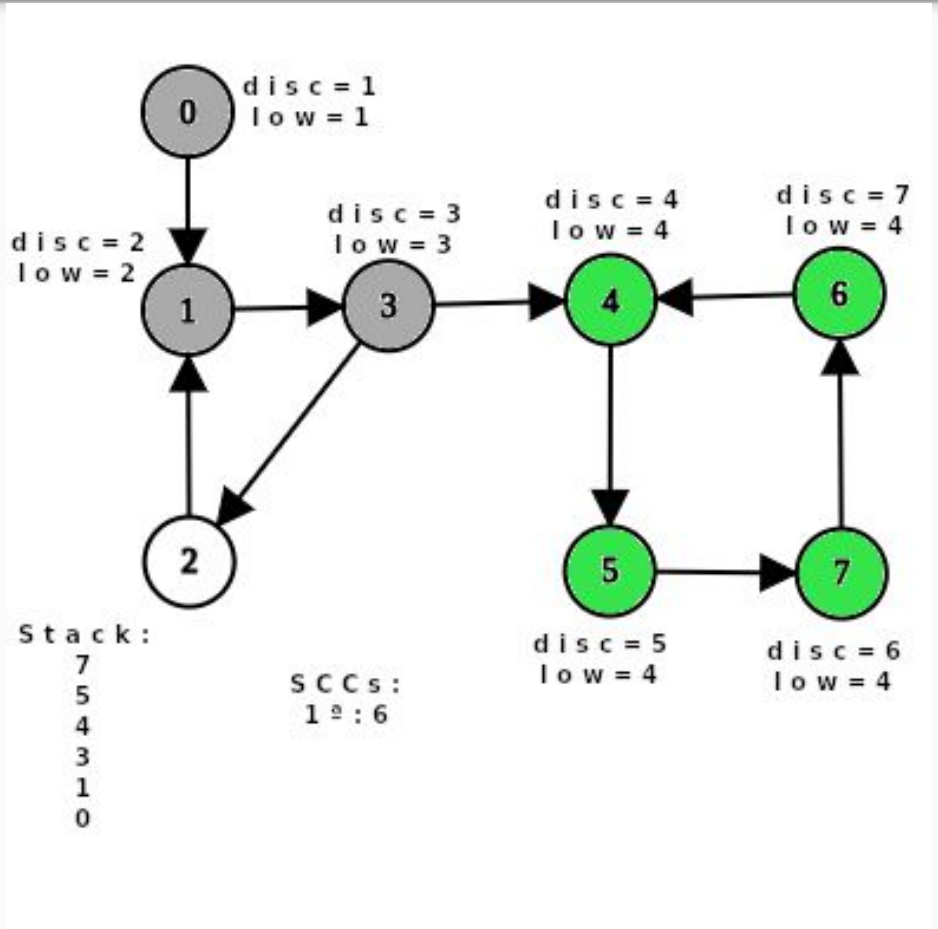
Execução Tarjan



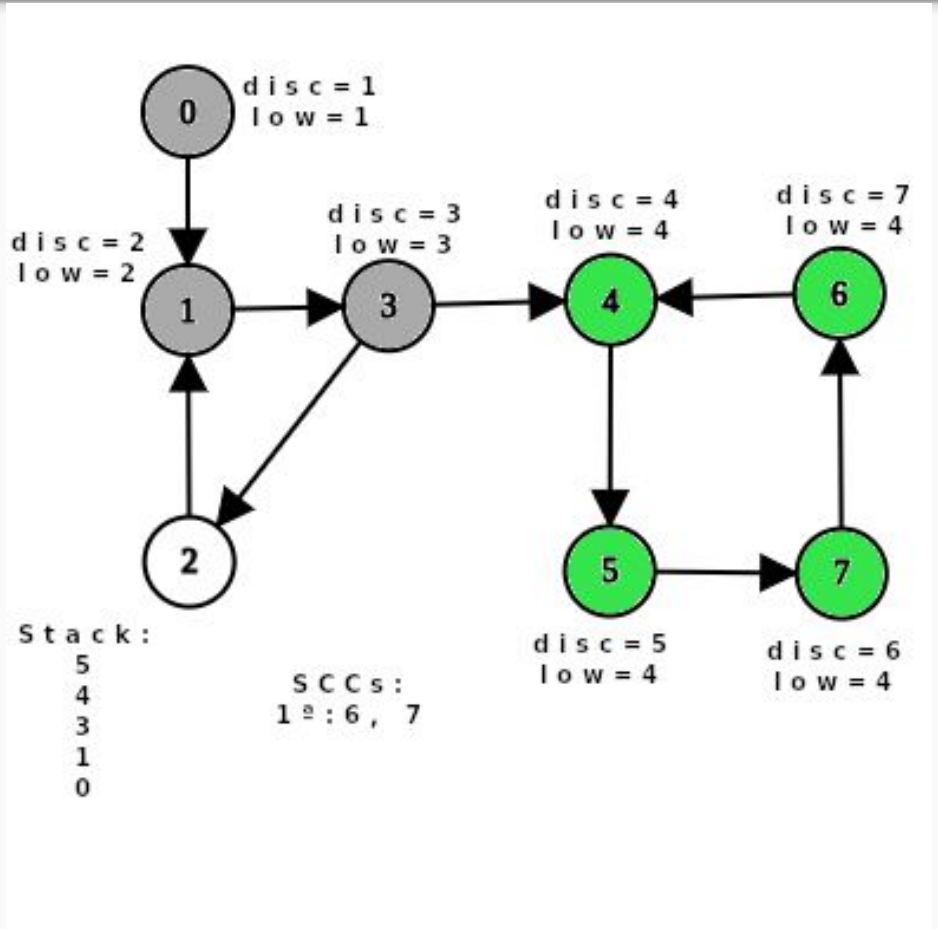
Execução Tarjan



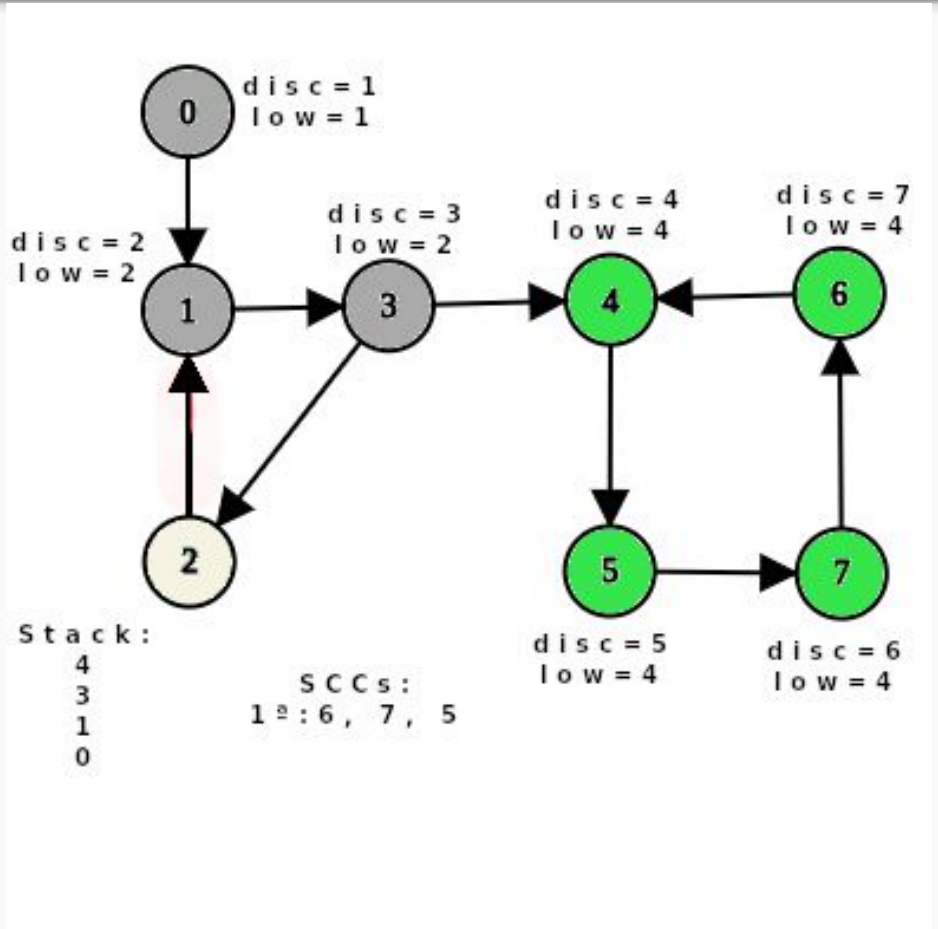
Execução Tarjan



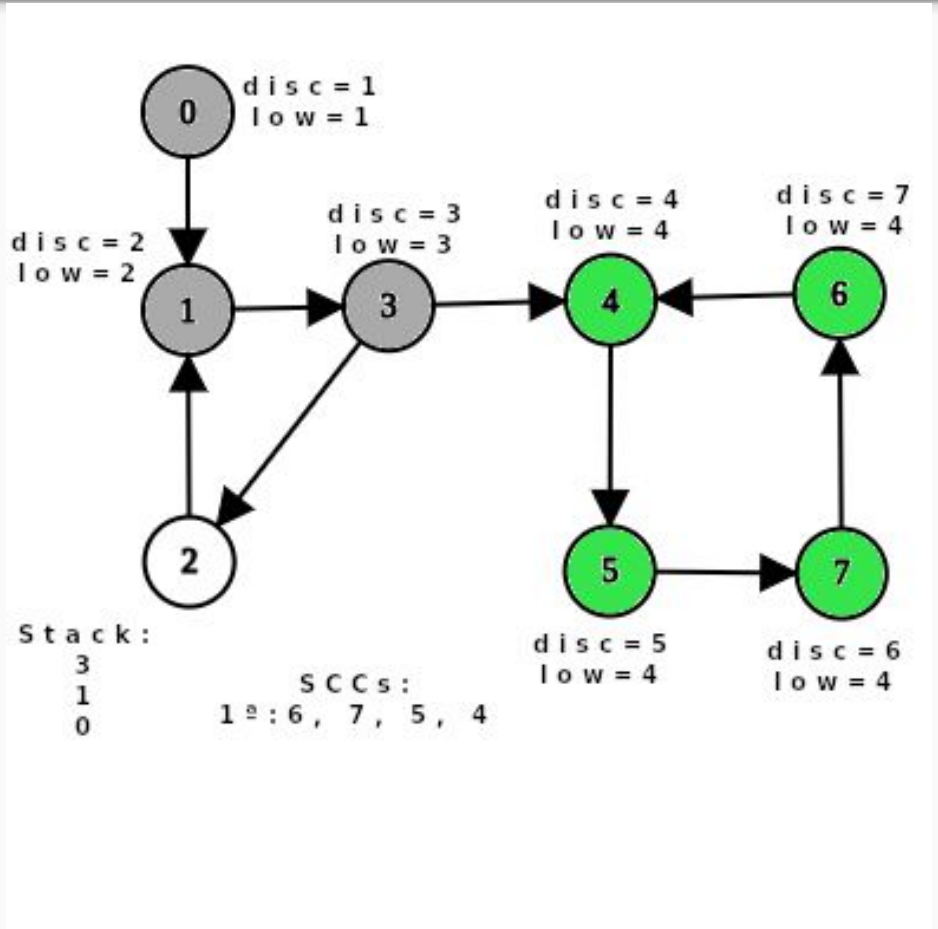
Execução Tarjan



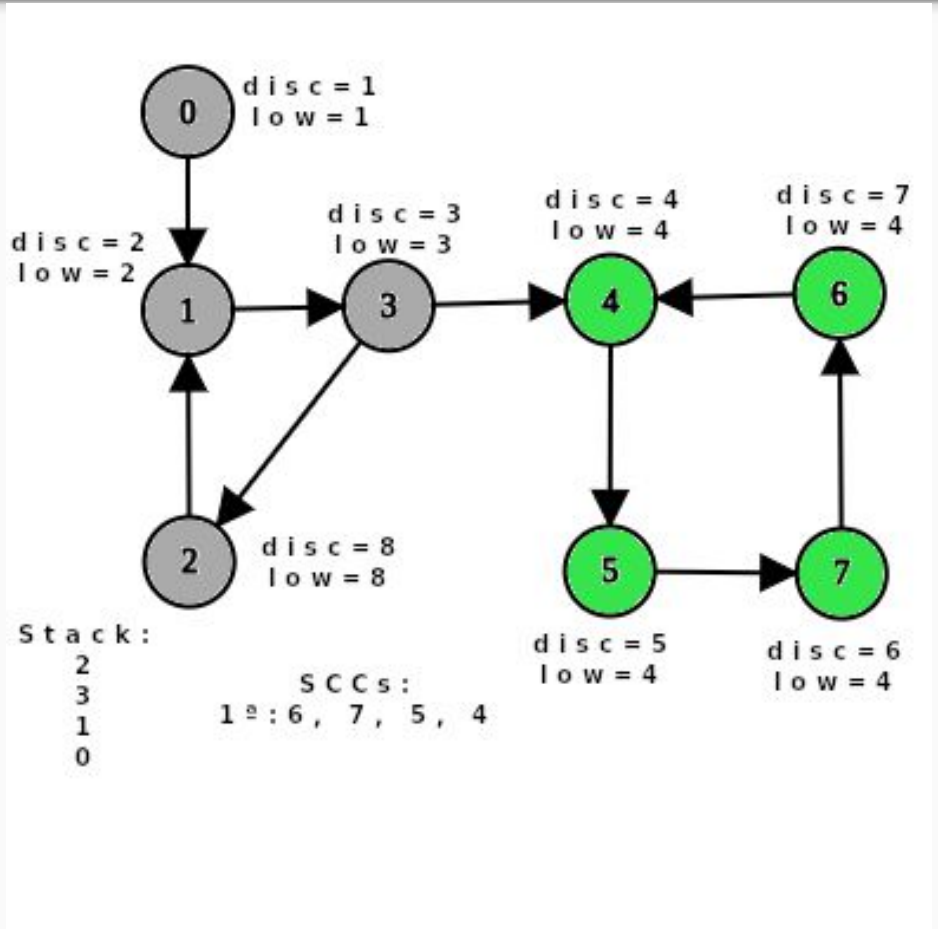
Execução Tarjan



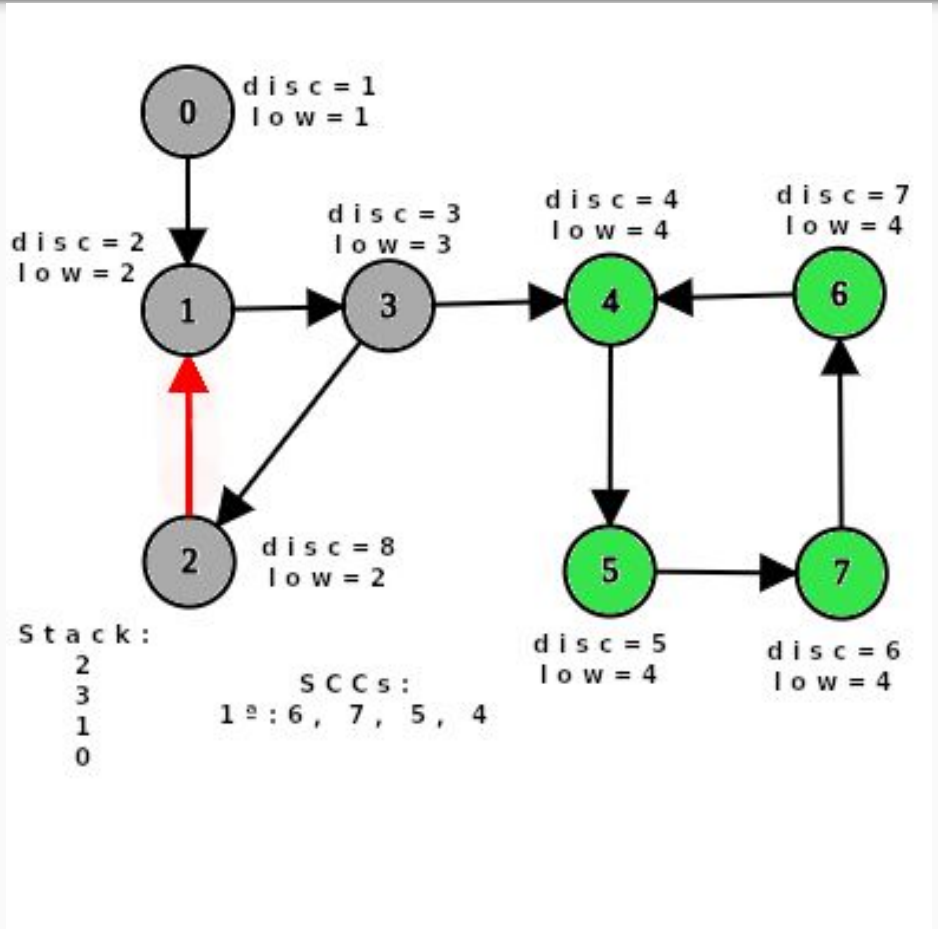
Execução Tarjan



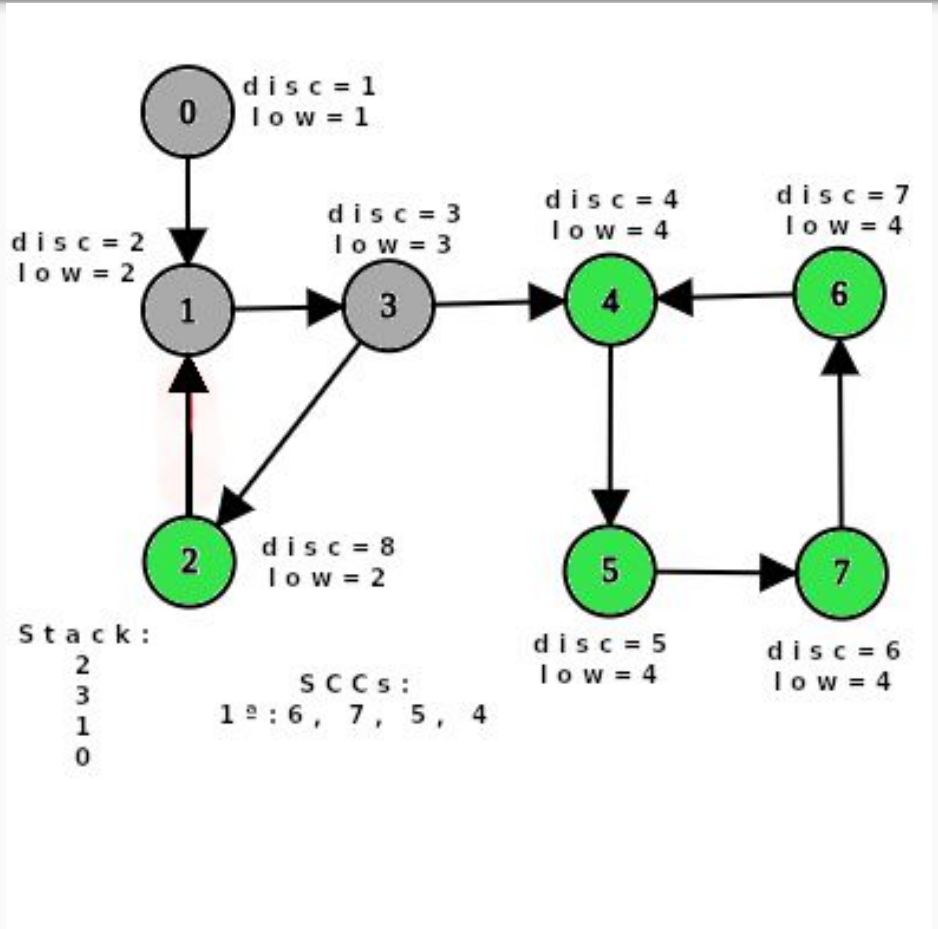
Execução Tarjan



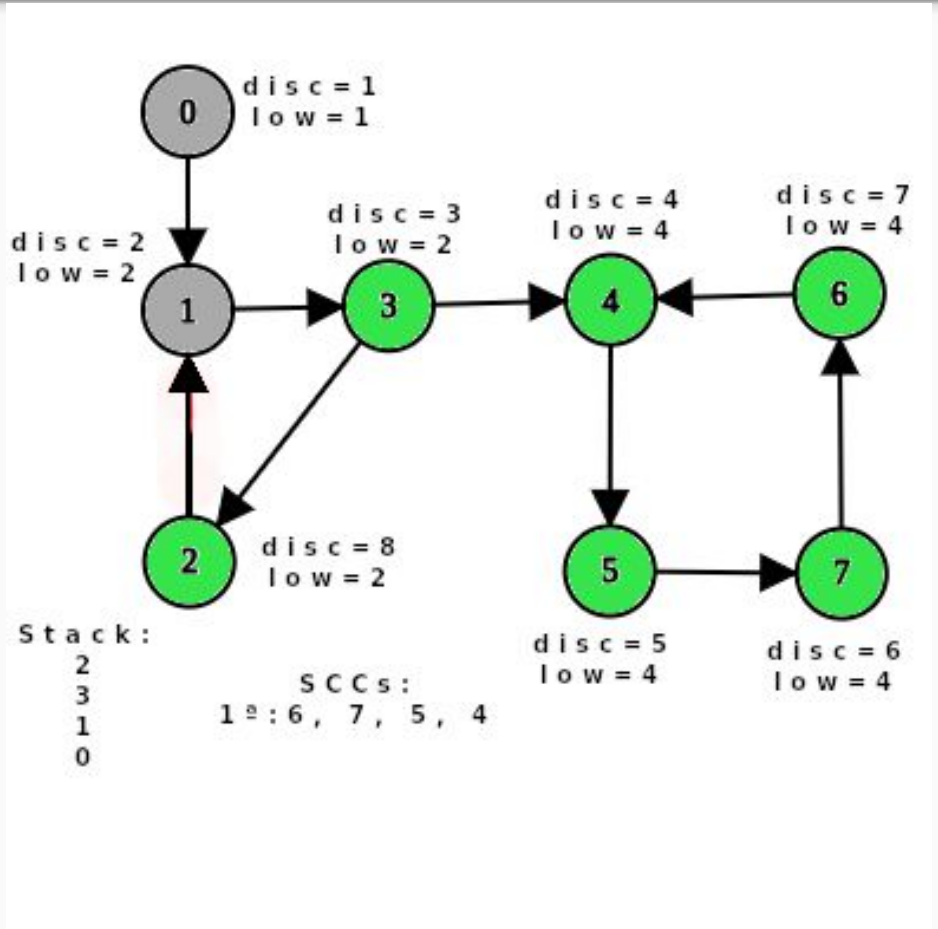
Execução Tarjan



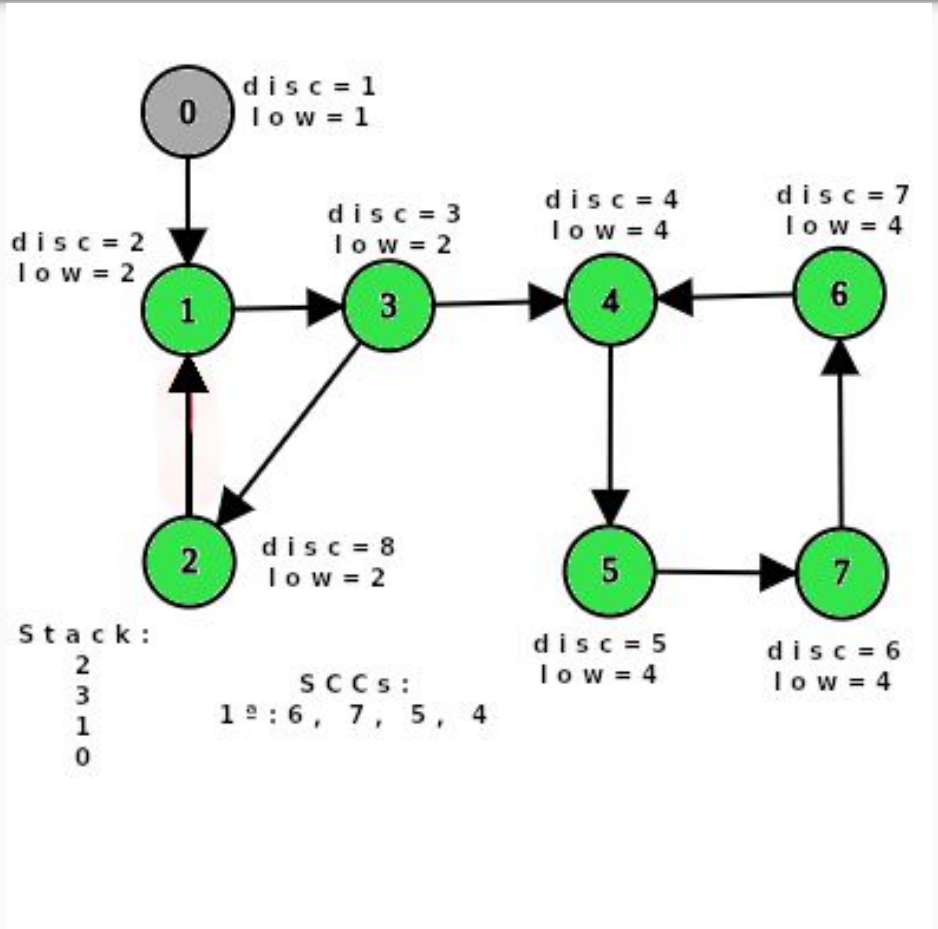
Execução Tarjan



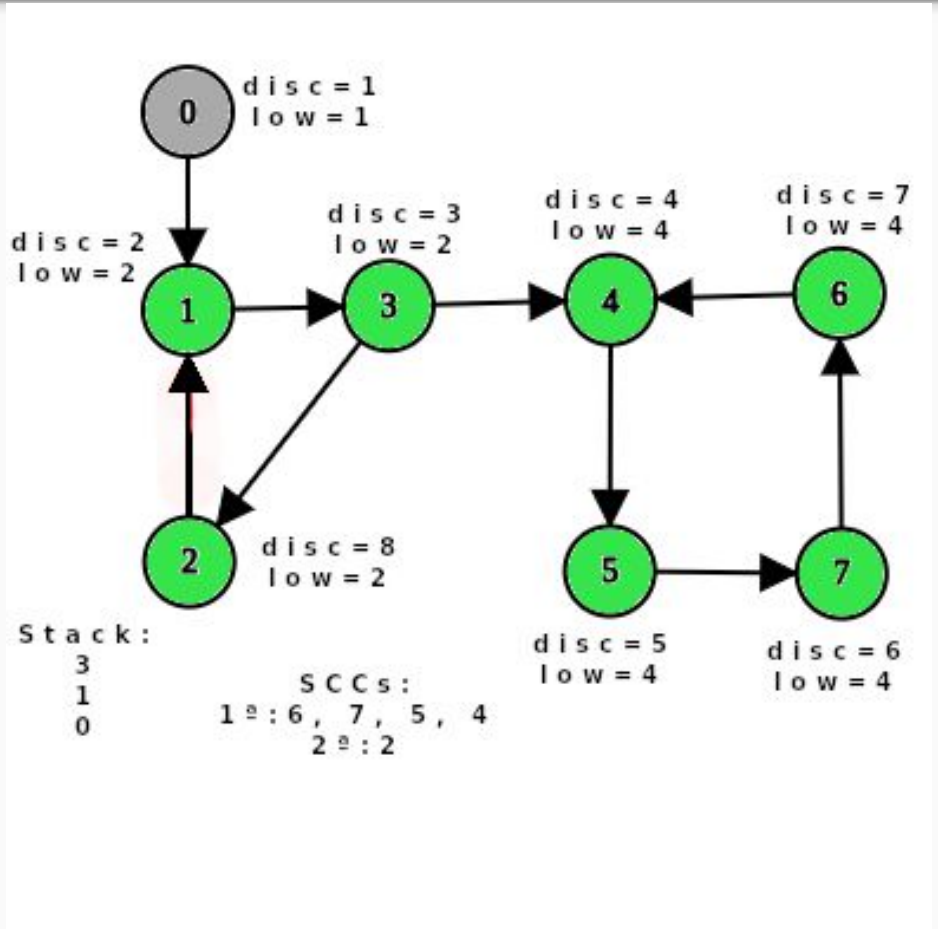
Execução Tarjan



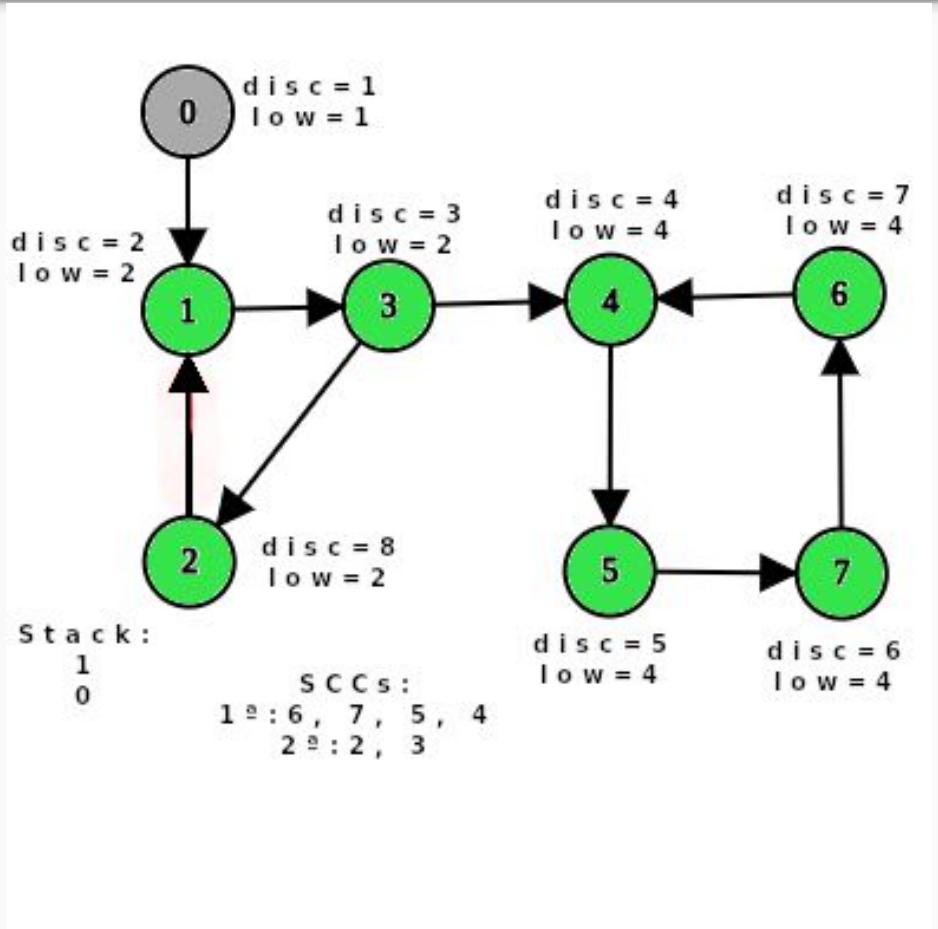
Execução Tarjan



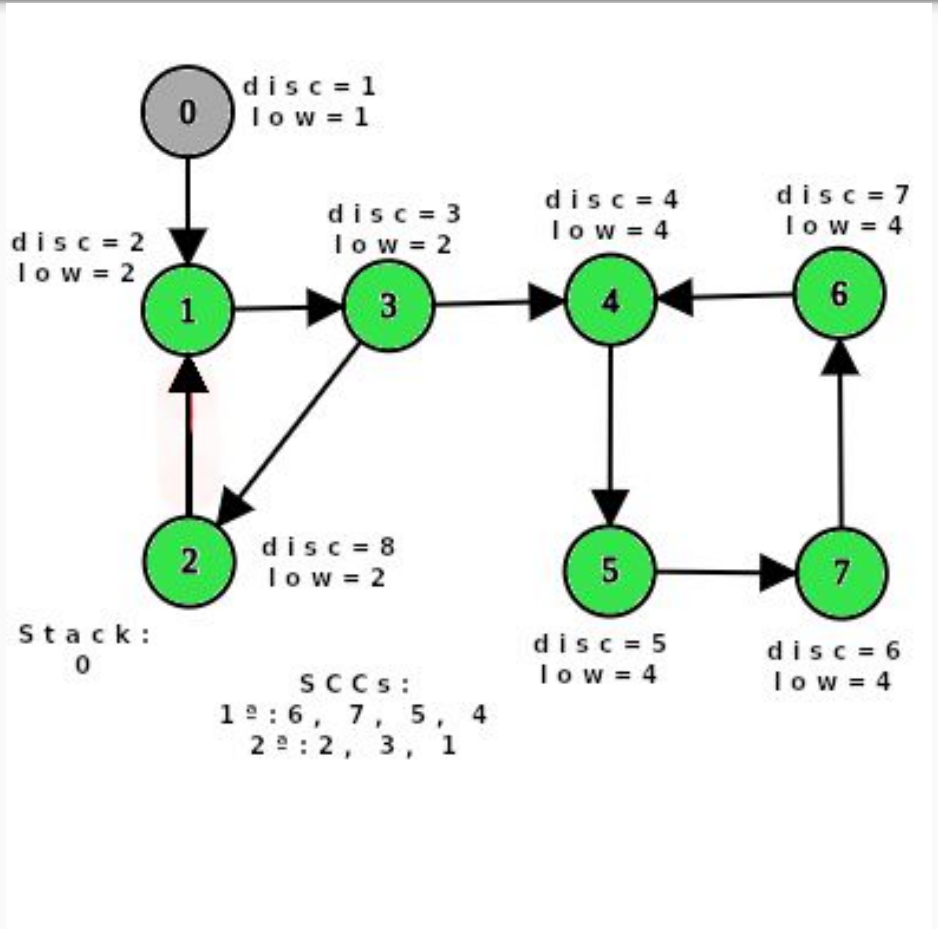
Execução Tarjan



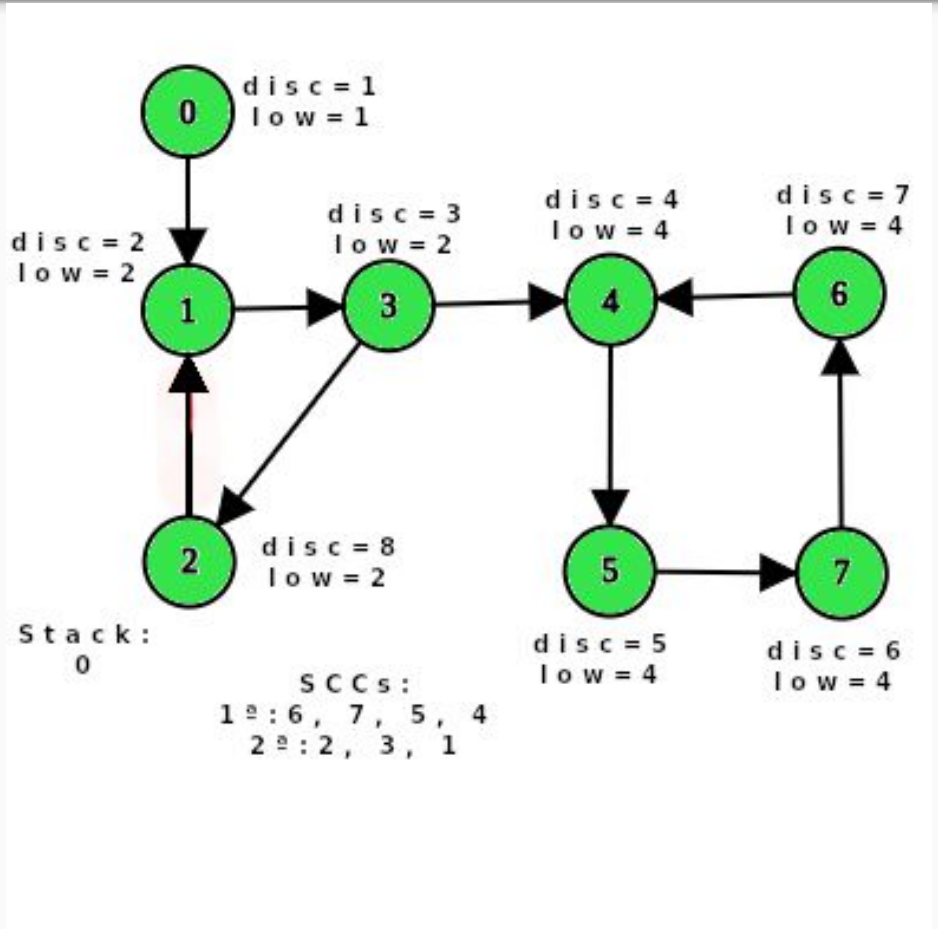
Execução Tarjan



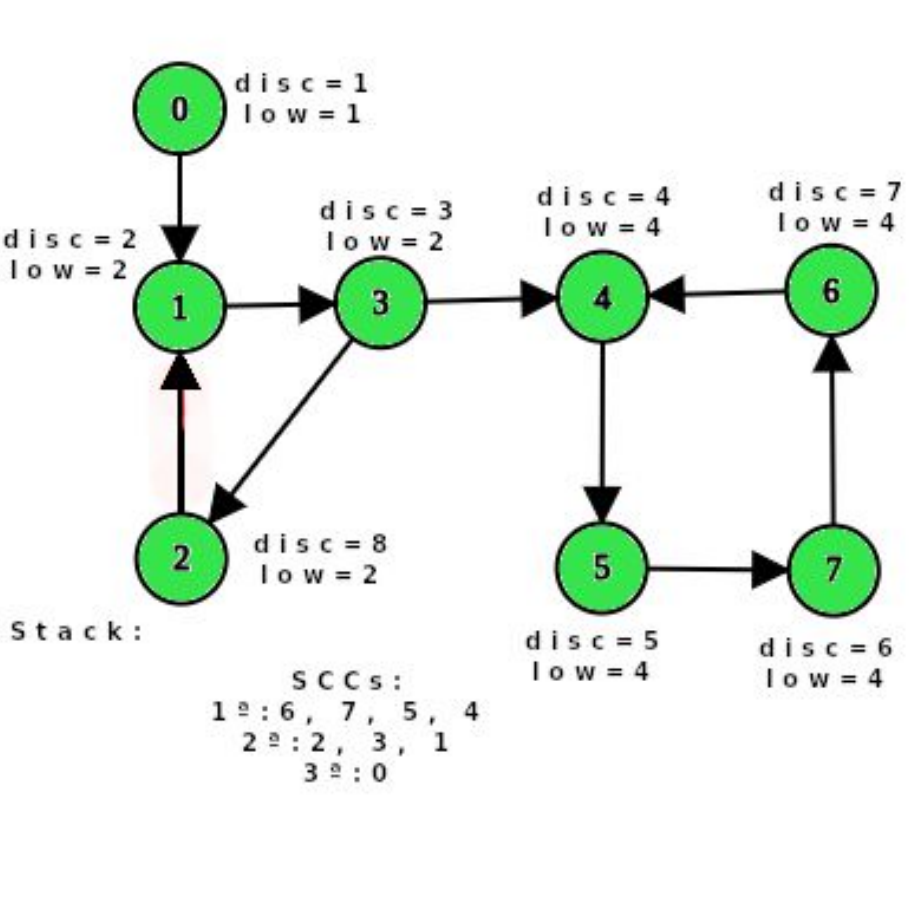
Execução Tarjan



Execução Tarjan



Execução Tarjan



Implementação - Algoritmo de Tarjan

Implementação Tarjan

```
void tarjanSCC(int u, vector<vector<int>> & adj, vector<int> & discovery, vector<int> & low, vector<bool> & visited){
    discovery[u] = low[u] = dfs_iteration++;
    visited_stack.push_back(u);
    visited[u] = true;
    for (int v: adj[u]){
        if (discovery[v] == -1){
            tarjanSCC(v, adj, discovery, low, visited);
        }
        if (visited[v]){
            low[u] = min(low[u], low[v]);
        }
    }

    if (low[u] == discovery[u]){
        counter_sccs++;
        cout << "SCC " << counter_sccs << ":" << endl;
        while(true){
            int v = visited_stack.back();
            visited_stack.pop_back();
            visited[v] = false;
            cout << v << endl;
            if (v == u){
                break;
            }
        }
    }
}
```

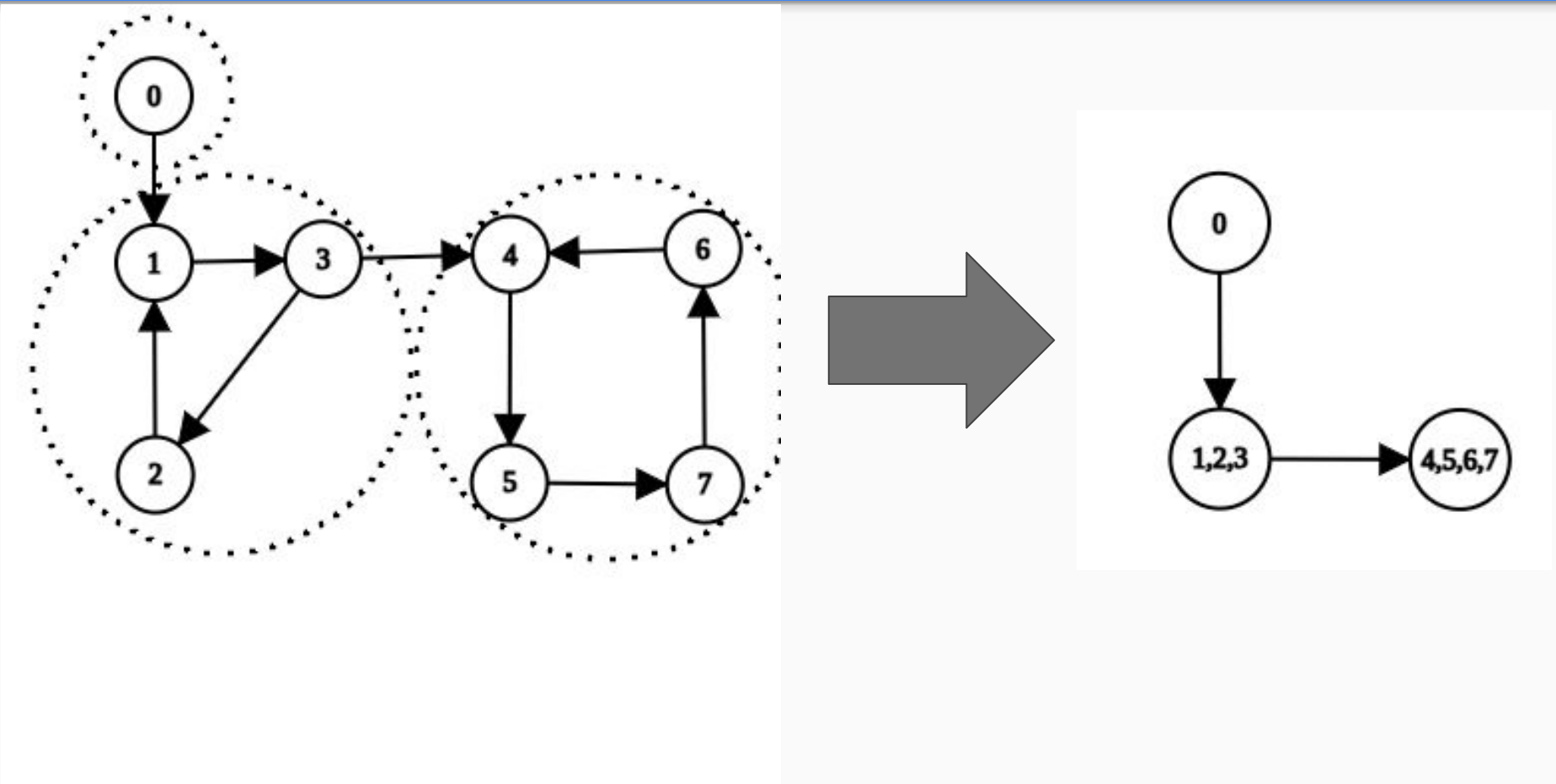
Implementação Tarjan

Main:

```
for (int i = 0; i < n; i++){
    if (discovery[i] == -1){
        tarjanSCC(i, adj, discovery, low, visited);
    }
}
```

Contração de Vértices

Contração de Vértices



Questões

Questões

1. https://www.spoj.com/problems/TFRIENDS/
2. https://codeforces.com/problemset/problem/427/C
3. https://br.spoj.com/problems/BURACOS/
4. https://www.spoj.com/problems/GOODA/
5. https://br.spoj.com/problems/CARDAPIO/