

# ESCOLA DE PRIMAVERA DA MARATONA SBC DE PROGRAMAÇÃO



PROMOÇÃO:



**UNIFEI**



APOIO:



Sociedade Brasileira  
de Computação



Grupo de Computação Competitiva

# DEPTH FIRST SEARCH

>  
BUSCA EM PROFUNDIDADE

Por: Oziel da Silva (UNIFEI - Itabira)

# CONTEÚDOS

- 01 - Problema motivador
- 02 - Definição do algoritmo
- 03 - Funcionamento do algoritmo
- 04 - Algoritmo
- 05 - Resolução do problema
- 06 - Outras aplicações
- 07 - Problemas

# MOTIVADOR

# BUSCA EM PROFUNDIDADE (DFS)

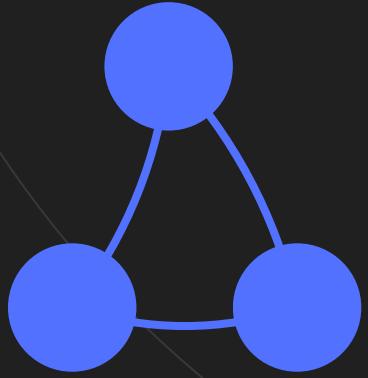
A **DFS** surgiu para resolver problemas fundamentais relacionados à exploração e análise de grafos e árvores.

- **Exploração Completa de Grafos:** Garantir que todos os vértices e arestas de um grafo sejam visitados de maneira eficiente, essencial para análises completas.
- **Detecção de Ciclos:** Identificar a existência de ciclos em um grafo, o que é crucial em muitos contextos, como verificação de dependências em sistemas operacionais ou compiladores.
- **Encontrar Componentes Conexos:** Determinar subgrafos em que todos os vértices estão interconectados, importante para entender a estrutura de redes.
- **Ordenação Topológica:** Em grafos direcionados acíclicos, a **DFS** permite ordenar os vértices de modo a respeitar as direções das arestas, útil em planejamento de tarefas e análise de dependências.



# BUSCA EM PROFUNDIDADE (DFS)

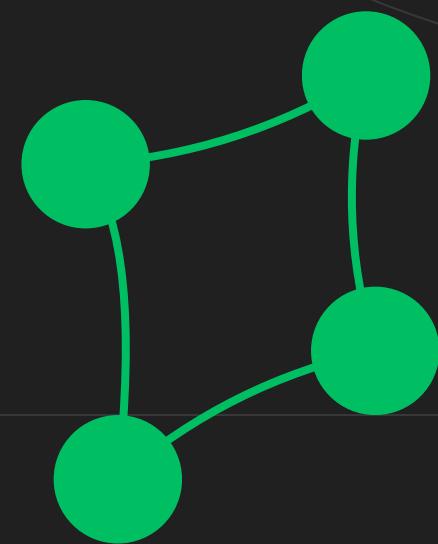
O problema clássico que motivou o desenvolvimento da **DFS** foi a necessidade de percorrer todas as rotas possíveis em um labirinto ou grafo sem se perder ou entrar em um *loop* infinito. Em termos computacionais, isso se traduz na necessidade de percorrer estruturas de dados recursivas, resolver labirintos e quebra-cabeças e análise de rede.



# DEFINIÇÃO

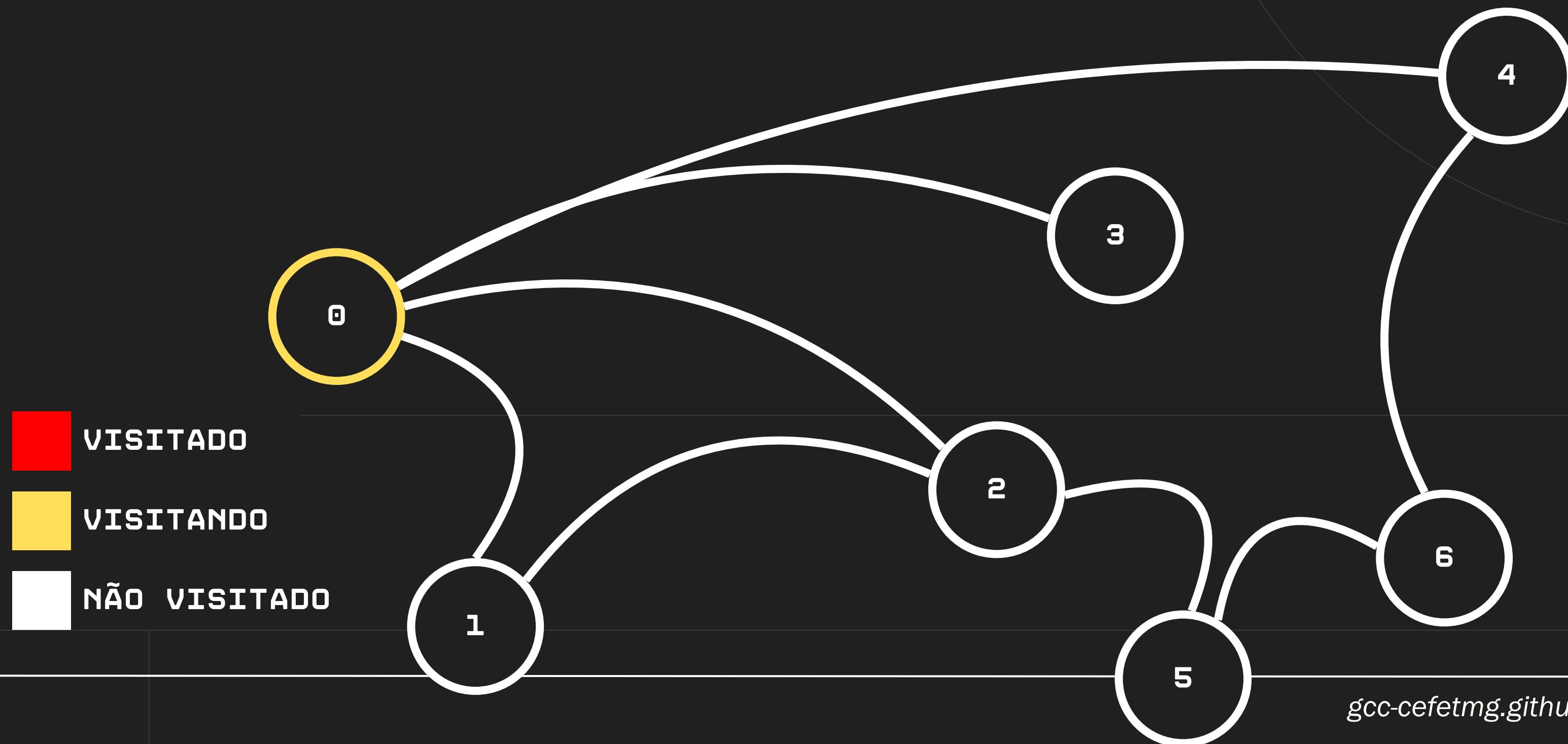
# BUSCA EM PROFUNDIDADE (DFS)

A Busca em Profundidade é um algoritmo de travessia ou busca em grafos que explora sistematicamente todos os vértices e arestas de uma estrutura de grafo, iniciando em um vértice raiz e seguindo cada caminho até seu limite antes de retroceder. O **DFS** utiliza uma abordagem recursiva ou uma estrutura de pilha para manter o controle dos vértices a serem visitados, marcando cada vértice como descoberto assim que é alcançado para evitar visitas repetidas.

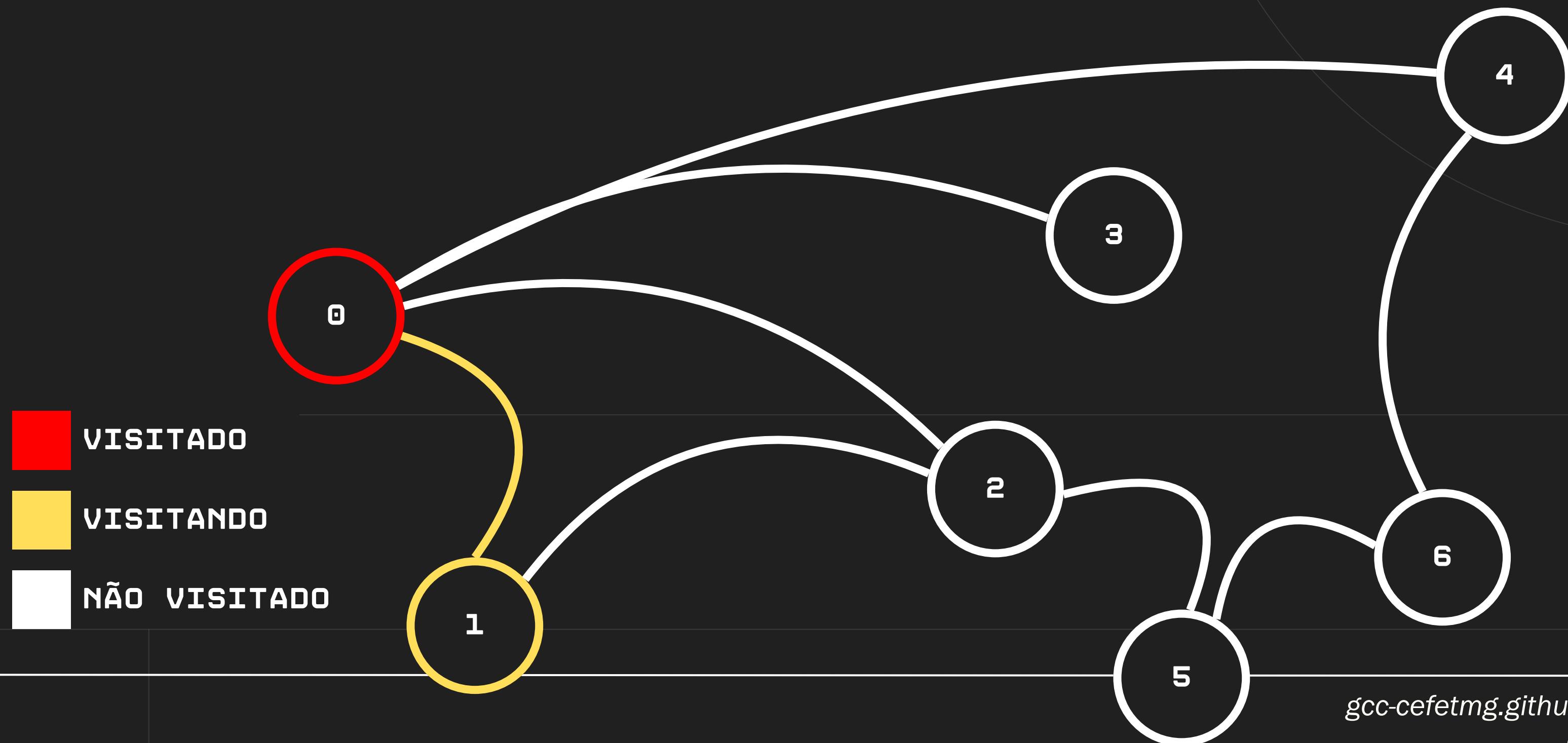


# FUNCIONAMENTO

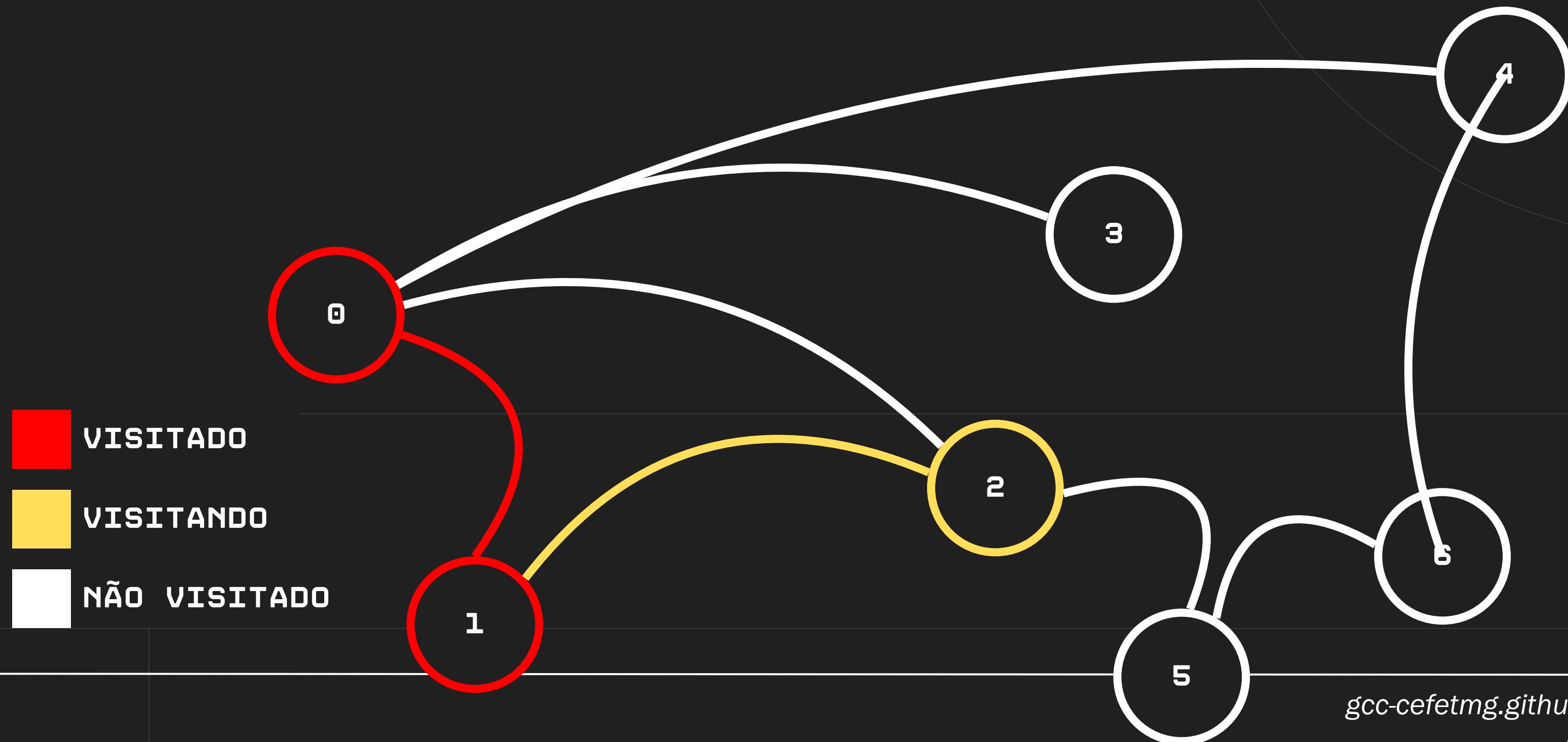
# FUNCIONAMENTO



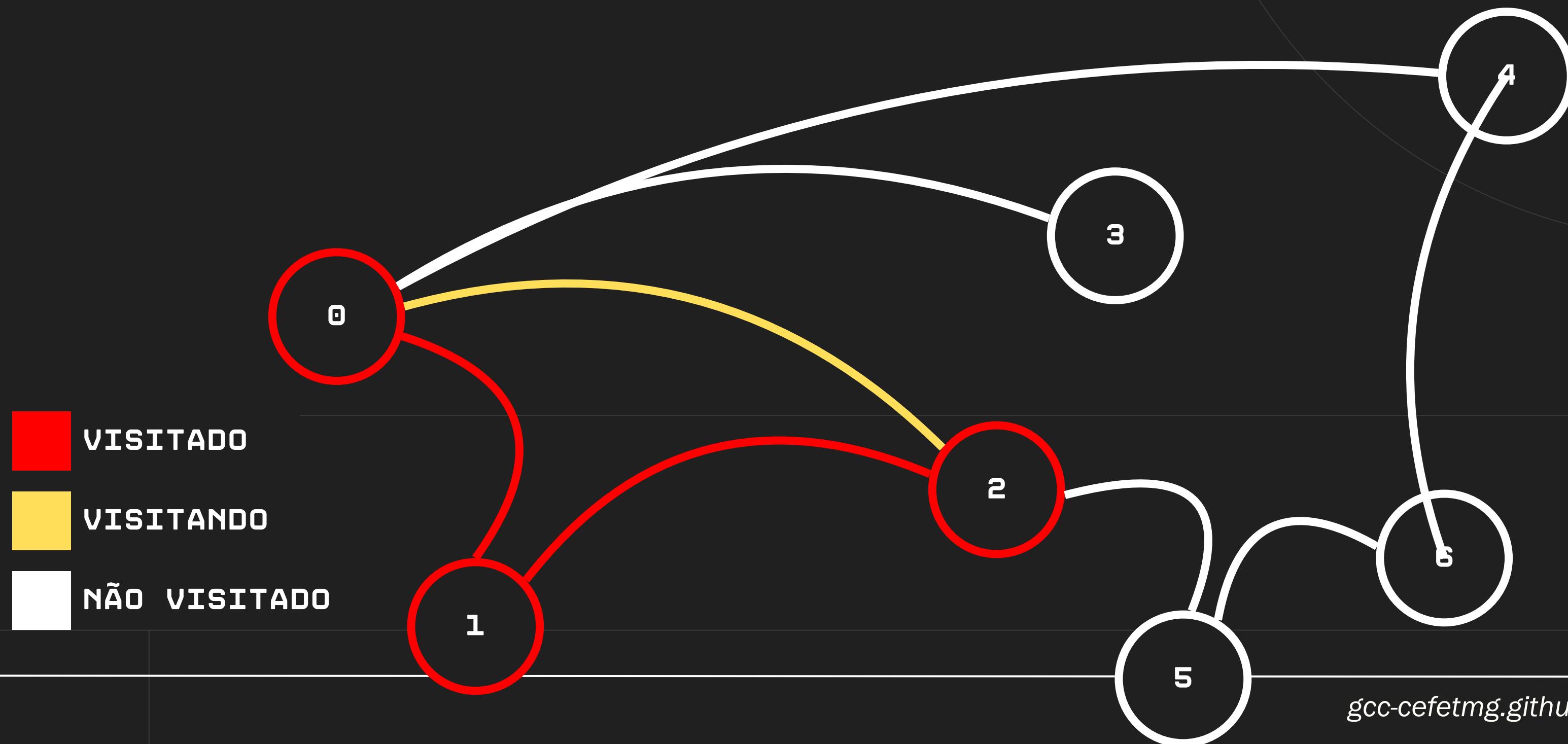
# FUNCIONAMENTO



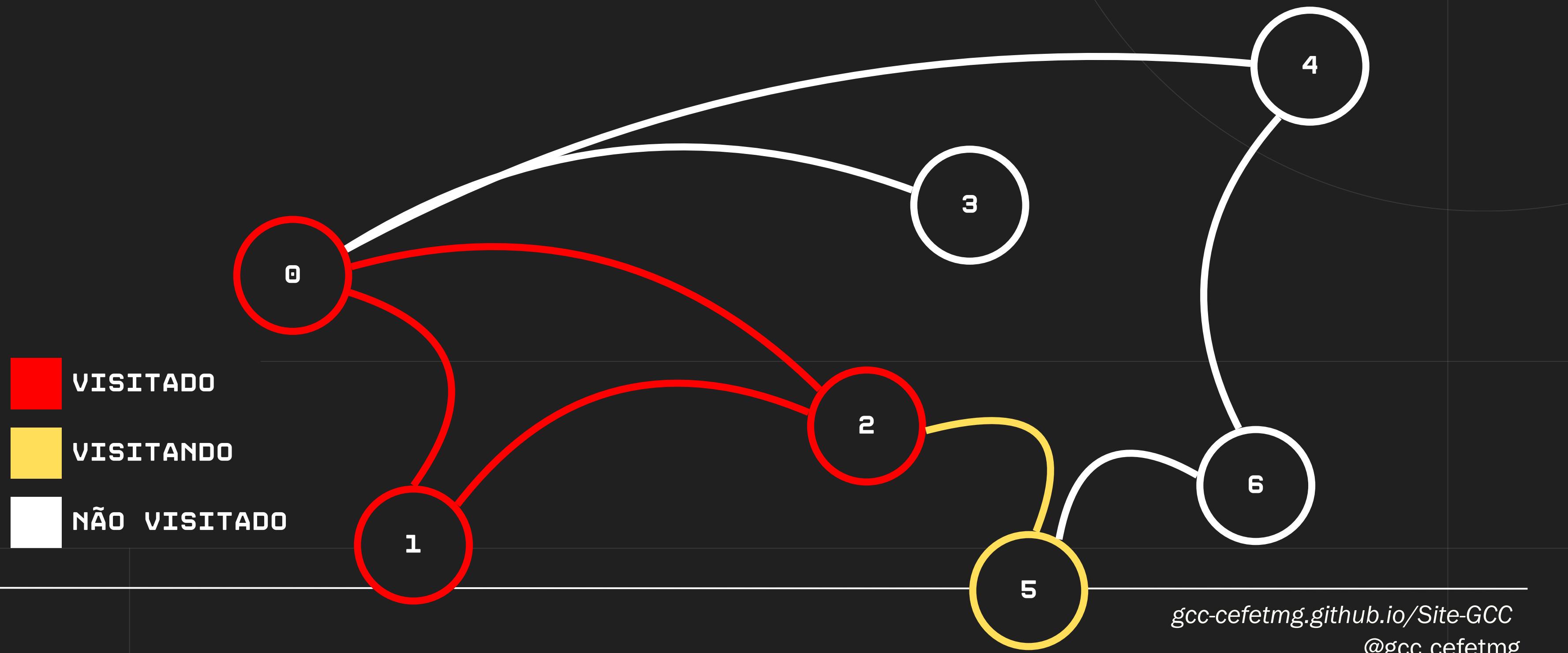
# FUNCIONAMENTO



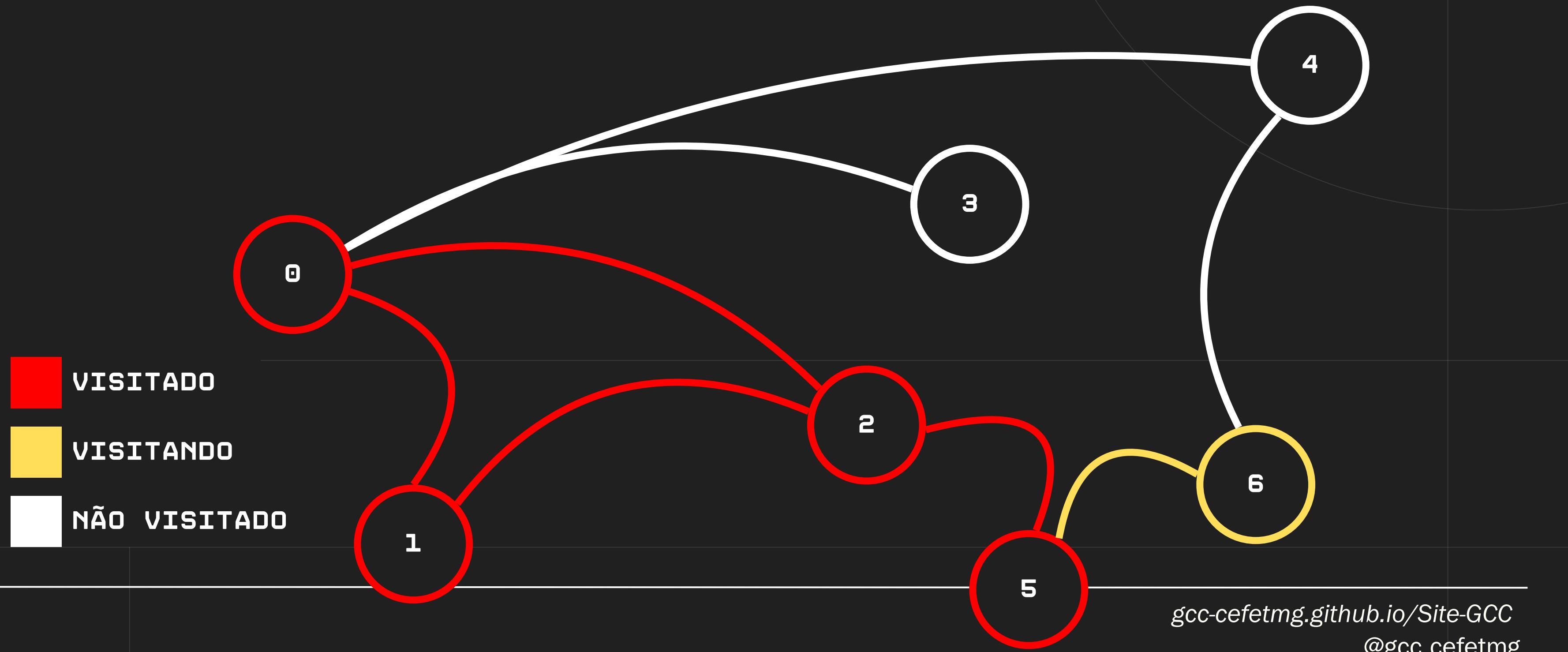
# FUNCIONAMENTO



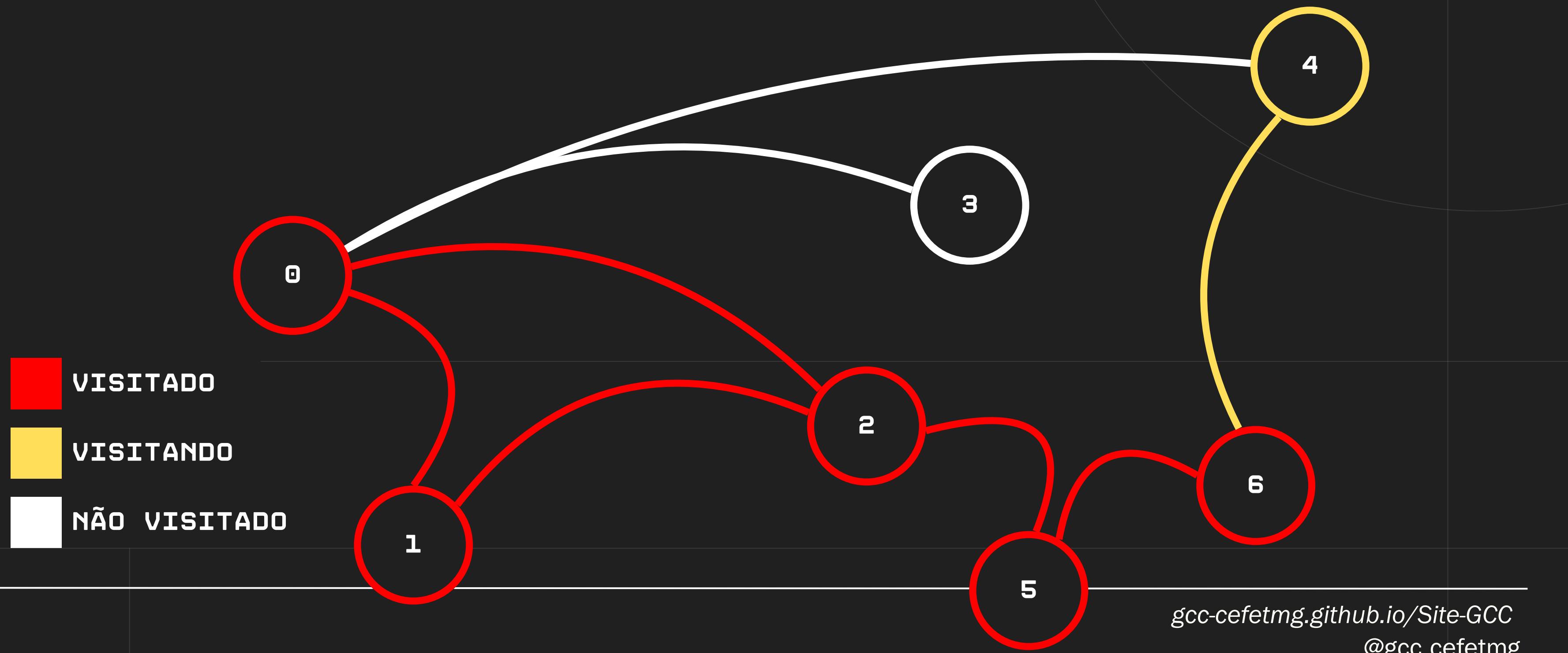
# FUNCIONAMENTO



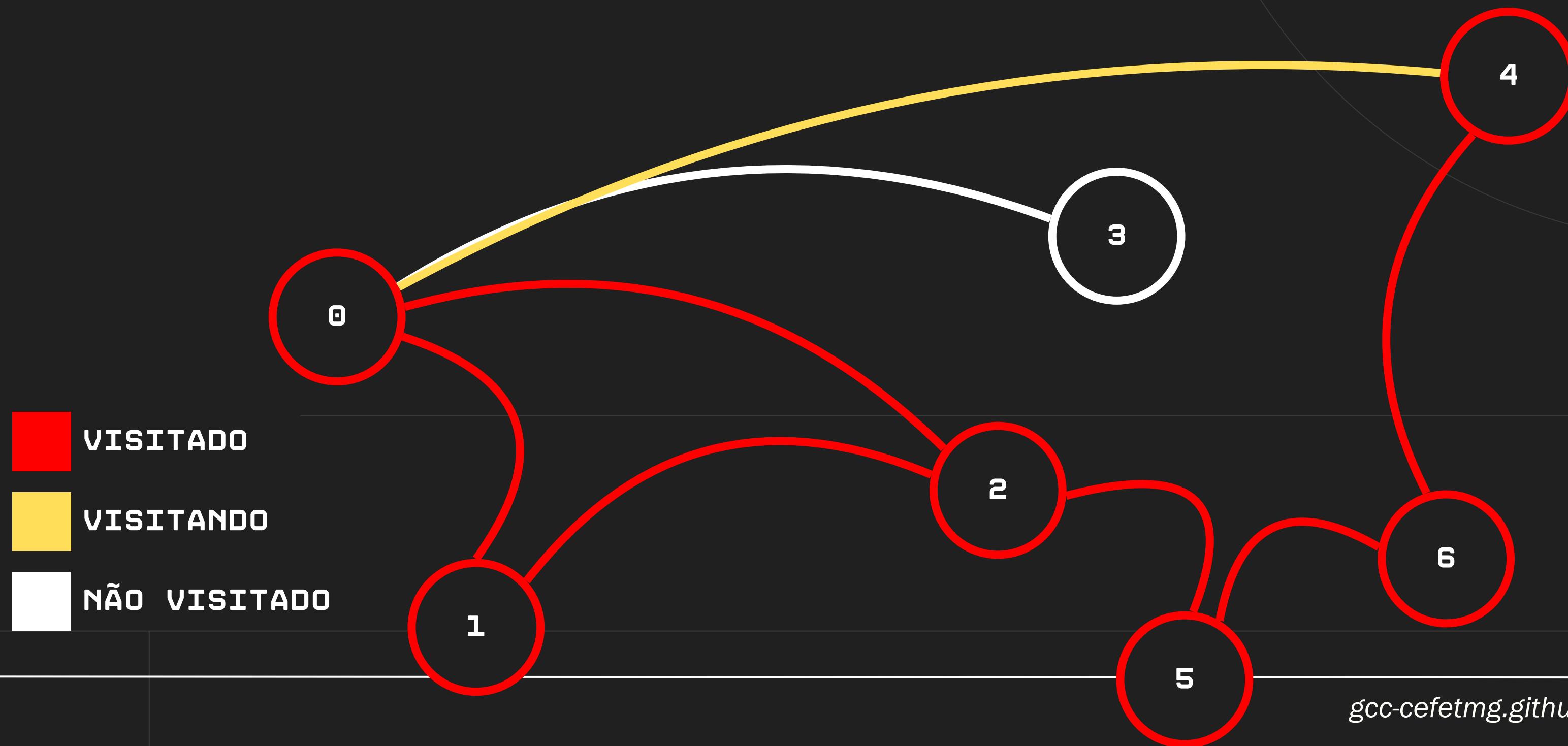
# FUNCIONAMENTO



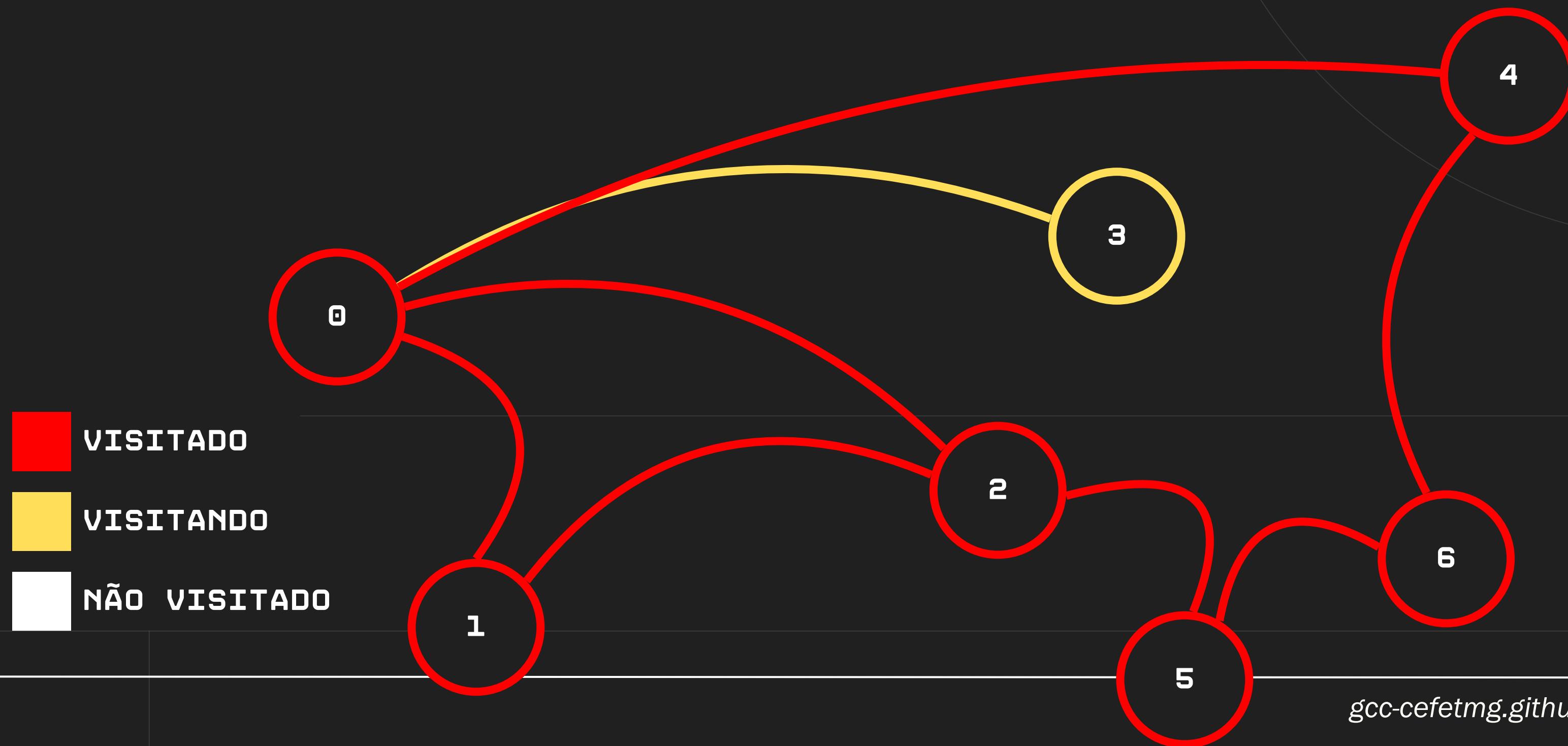
# FUNCIONAMENTO



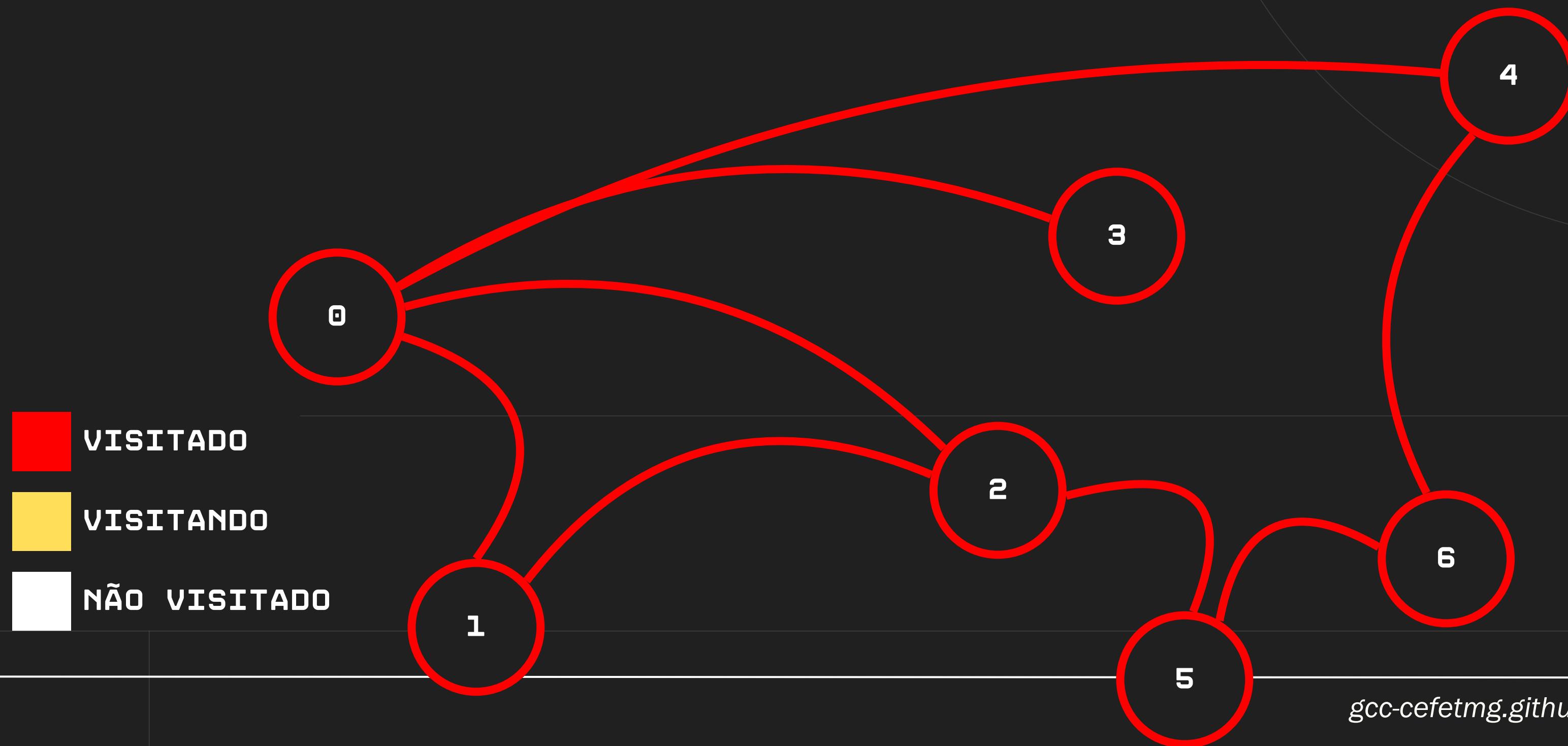
# FUNCIONAMENTO



# FUNCIONAMENTO



# FUNCIONAMENTO



# VISUALIZAÇÃO DO ALGORITMO



UNIVERSITY OF  
SAN FRANCISCO

<https://www.cs.usfca.edu/~galles/visualization/DFS.html>

# ALGORITMO



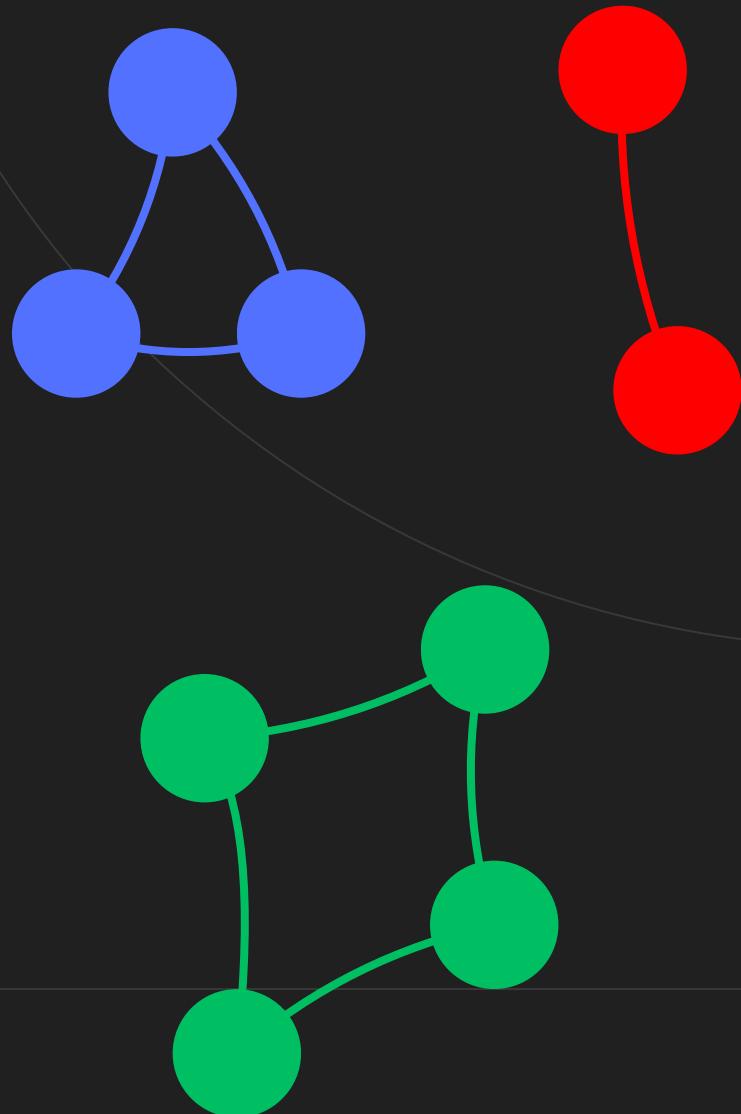
# RESOLUÇÃO DE UM EXEMPLO

# CASAS DE WESTEROS

Daenerys: “*Lannister, Targaryen, Baratheon, Stark, Tyrell.* Estão todos em uma mesma roda”.

As casas nobres de Westeros estão lutando constantemente pelo Trono de Ferro. Para vencer a Guerra dos Tronos, deve-se sempre saber quantas casas existem no continente. Também é importante saber o tamanho de cada casa, uma vez que casas com muitas pessoas são, normalmente, mais fortes que casas com poucos membros.

Existem  $N$  pessoas em Westeros. Para cada par de pessoas, um espião lhe informou se elas pertencem à mesma casa ou não. Se a informação obtida pelo espião for consistente, sua tarefa é determinar quantas casas existem em Westeros, e quantas pessoas pertencem a cada casa.

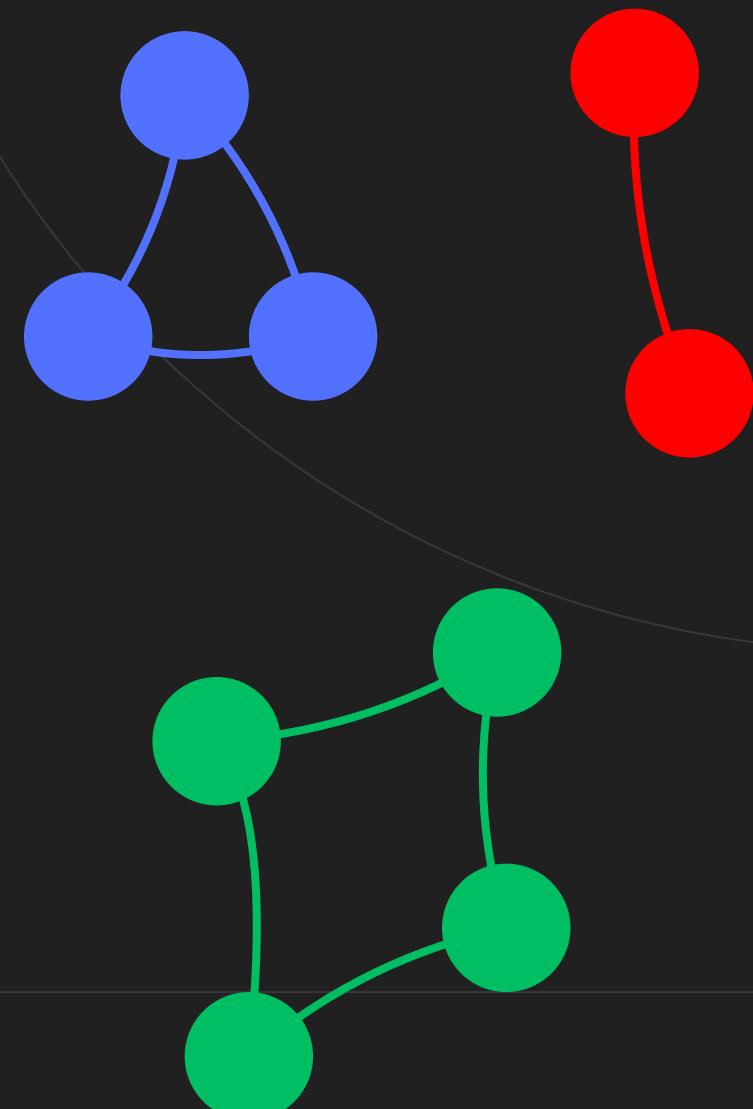
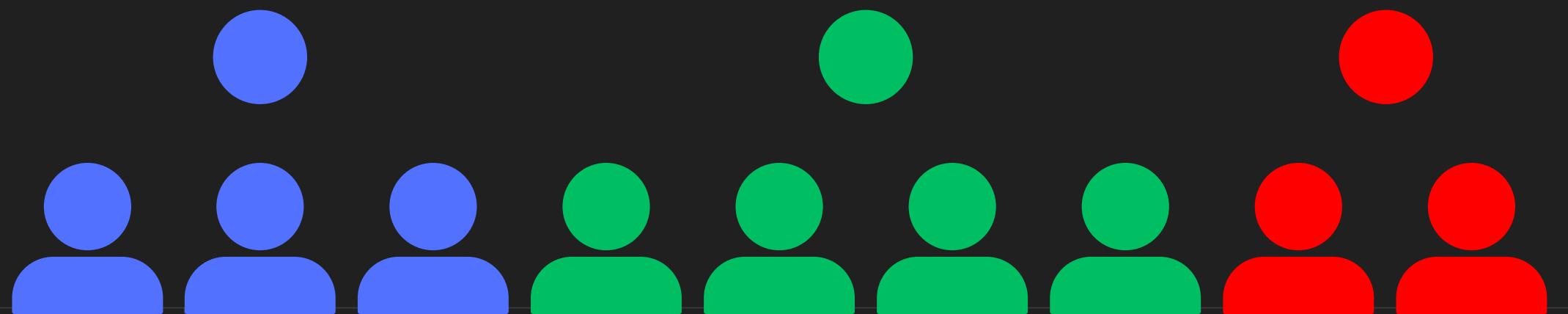


# FUNCIONAMENTO

Você tem ***N*** pessoas em Westeros, e para cada par de pessoas, sabe-se se elas pertencem à mesma casa (**S**) ou a casas diferentes (**D**). Sua tarefa é:

- Verificar se a informação fornecida é consistente.
- Se for consistente, determinar:
  - Quantas casas existem.
  - Quantas pessoas pertencem a cada casa, em ordem não crescente.

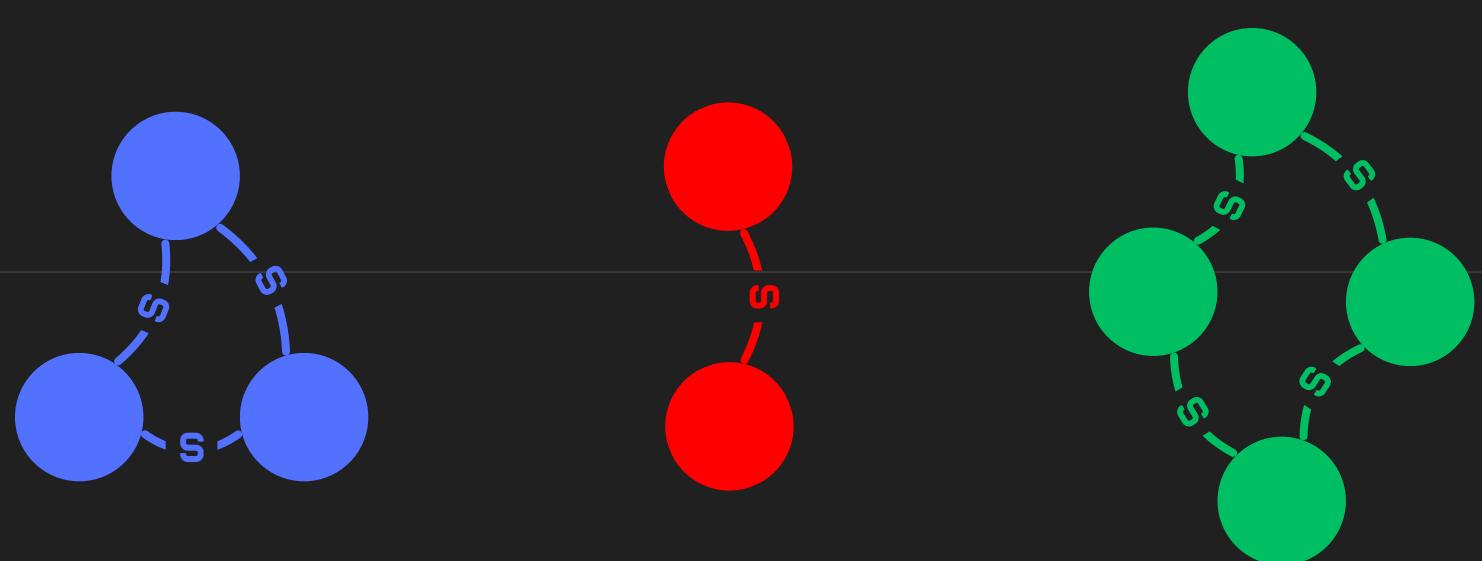
Se a informação for ***inconsistente***, você deve imprimir -1.



# MODELAGEM DO PROBLEMA

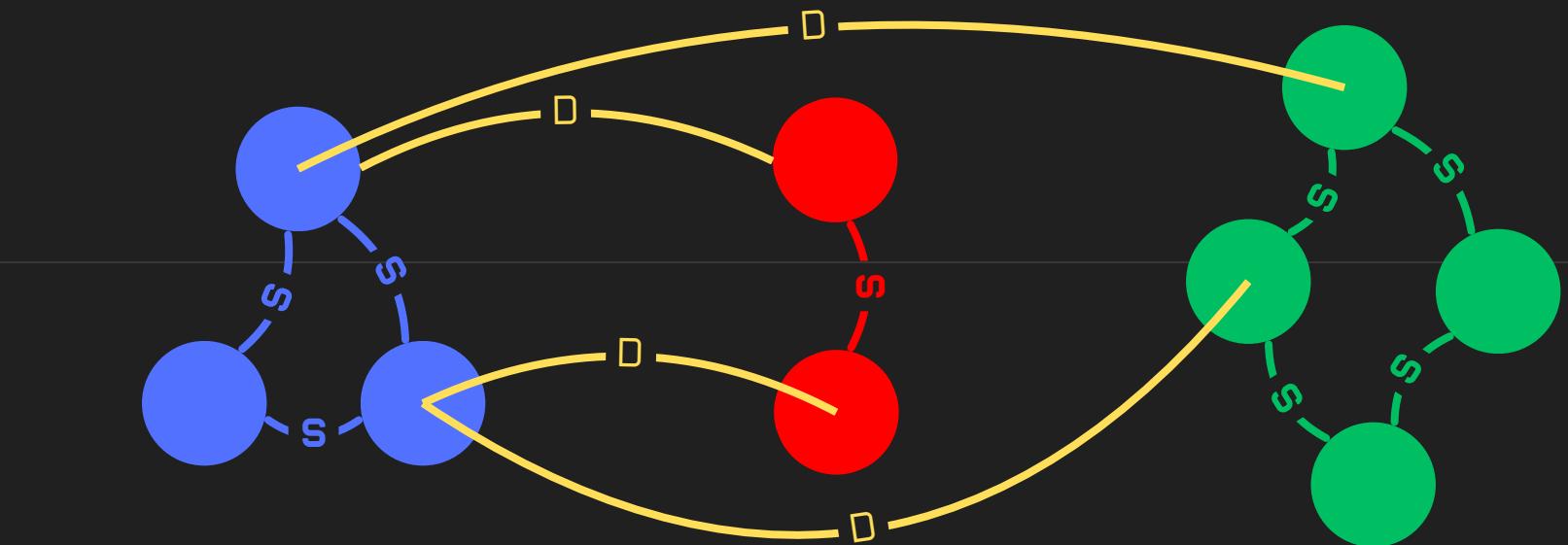
Para resolver este problema, podemos modelá-lo como um grafo não direcionado, onde:

- **Nós (vértices):** Representam cada uma das  $N$  pessoas.
- **Arestas:** Representam relações de mesma casa ( $S$ ) entre pessoas.



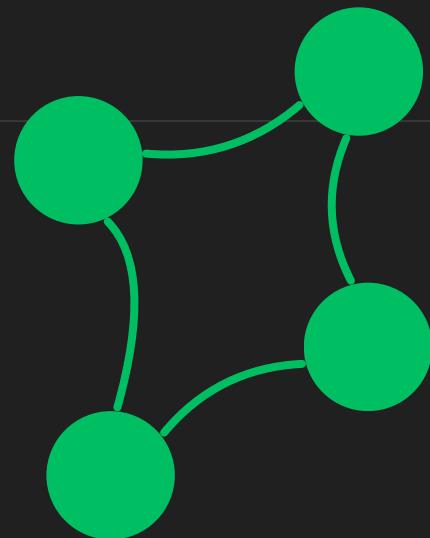
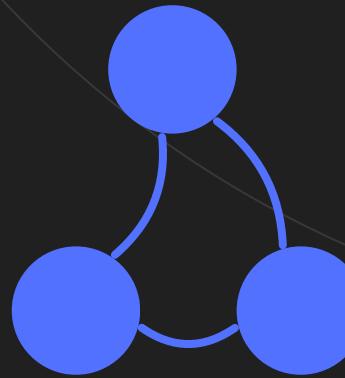
# CONSTRUÇÃO DO GRAFO

- Construir o grafo com base nas relações ‘S’:
  - Para cada par de pessoas  $(i, j)$  onde  $S[i][j] == 'S'$  e  $i \neq j$ , adicionamos uma aresta entre  $i$  e  $j$ .
- Armazenar as relações ‘D’:
  - Para cada par de pessoas  $(i, j)$  onde  $S[i][j] == 'D'$  e  $i < j$ , armazenamos o par  $(i, j)$  para verificação posterior.



# ENCONTRAR COMPONENTES CONEXAS COM DFS

- Inicializar um vetor de visitação:
  - Criação um vetor de visitas.
- Percorrer todos os nós:
  - Para cada pessoa  $i$  de 1 até  $N$ :
    - Se não houver visitado, significa que ainda não foi atribuída a uma **casa**.
    - Inicia-se uma **DFS** a partir de  $i$  para identificar todas as pessoas que pertencem à mesma **casa**.
    - Atribui-se uma **casa** a cada nó visitado durante a **DFS**.
- Durante a **DFS**:
  - Marca o nó como visitado.
  - Para cada vizinho  $j$  de  $i$  no grafo:
    - Se não tiver sido visitado, continue a **DFS** a partir de  $j$ .

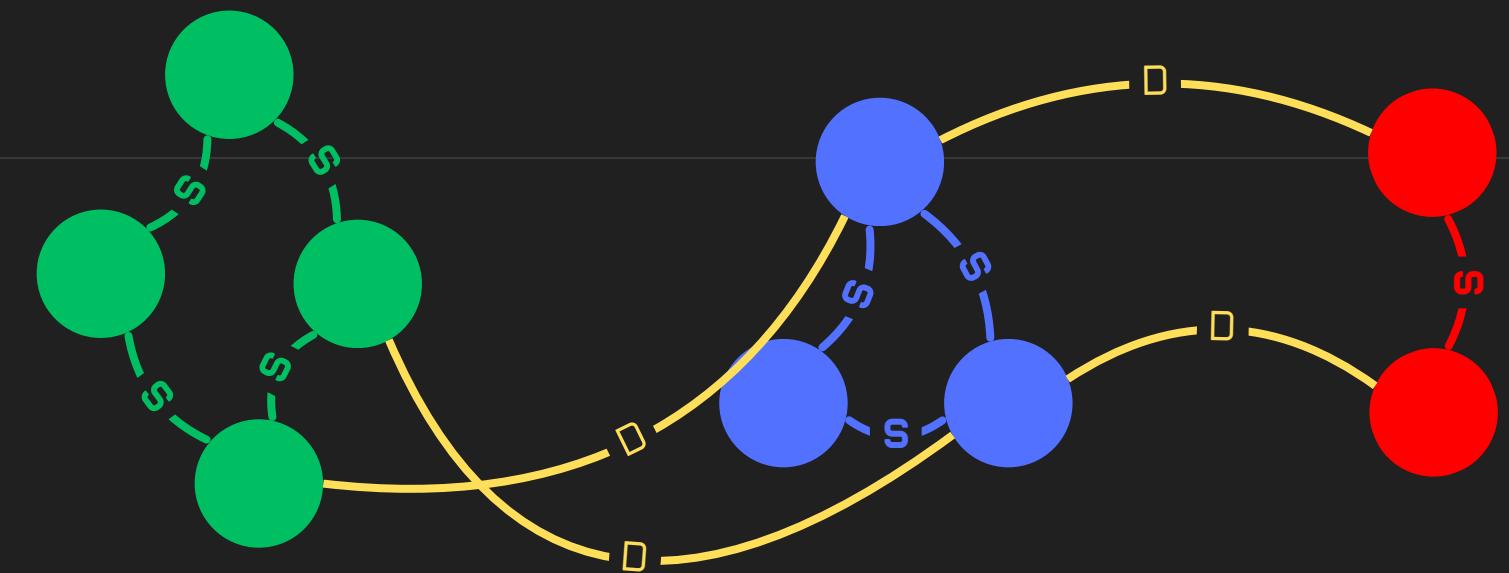


# VERIFICAÇÃO DE CONSISTÊNCIA

- Para cada relação ‘D’ armazenada:
  - É feita uma verificação para definir se as pessoas  $i$  e  $j$  pertencem à mesma casa.
    - Se pertencerem, há uma ***inconsistência***, pois pessoas que devem estar em casas diferentes estão na mesma casa.
    - Nesse caso, se imprime “-1” e termine o programa.
- Se todas as relações ‘D’ forem consistentes:
  - A informação é consistente.

# DETERMINAÇÃO DO NÚMERO DE CASAS E TAMANHOS

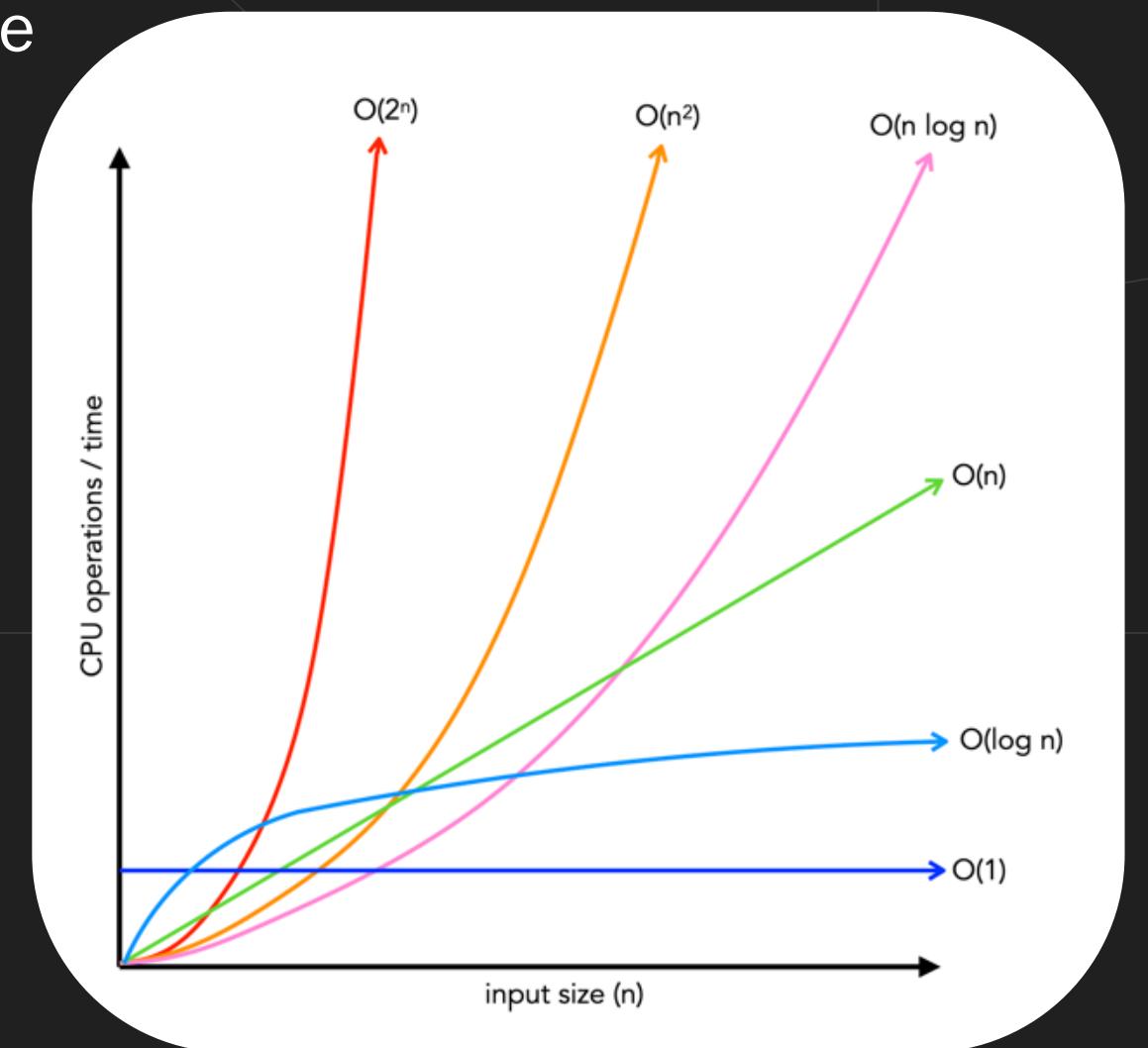
- Número de Casas (**K**):
  - É igual ao número de componentes conexas encontradas no grafo.
- Número de Pessoas em Cada Casa:
  - Para cada componente conexa, conte o número de **nós** (pessoas) que a compõem.
  - Ordene os tamanhos em **ordem não crescente**.



# 03 - FUNCIONAMENTO DO ALGORITMO

O algoritmo utiliza **DFS** para identificar componentes conexas em um grafo onde cada componente representa uma casa nobre em Westeros. Em seguida, verifica se as relações de “diferentes casas” (**D**) são consistentes, ou seja, se as pessoas que devem estar em casas diferentes realmente não estão na mesma componente. Se todas as relações forem consistentes, o algoritmo calcula e imprime o número de casas e o número de pessoas em cada uma, ordenados em **ordem decrescente**. Caso contrário, imprime **-1** para indicar inconsistências.

Essa abordagem garante eficiência mesmo para o maior valor de **N** permitido (1000), pois todas as operações principais têm complexidade  **$O(N^2)$** .  
(E  $10^{3^2} = 10^6$ , que é computacionalmente possível)

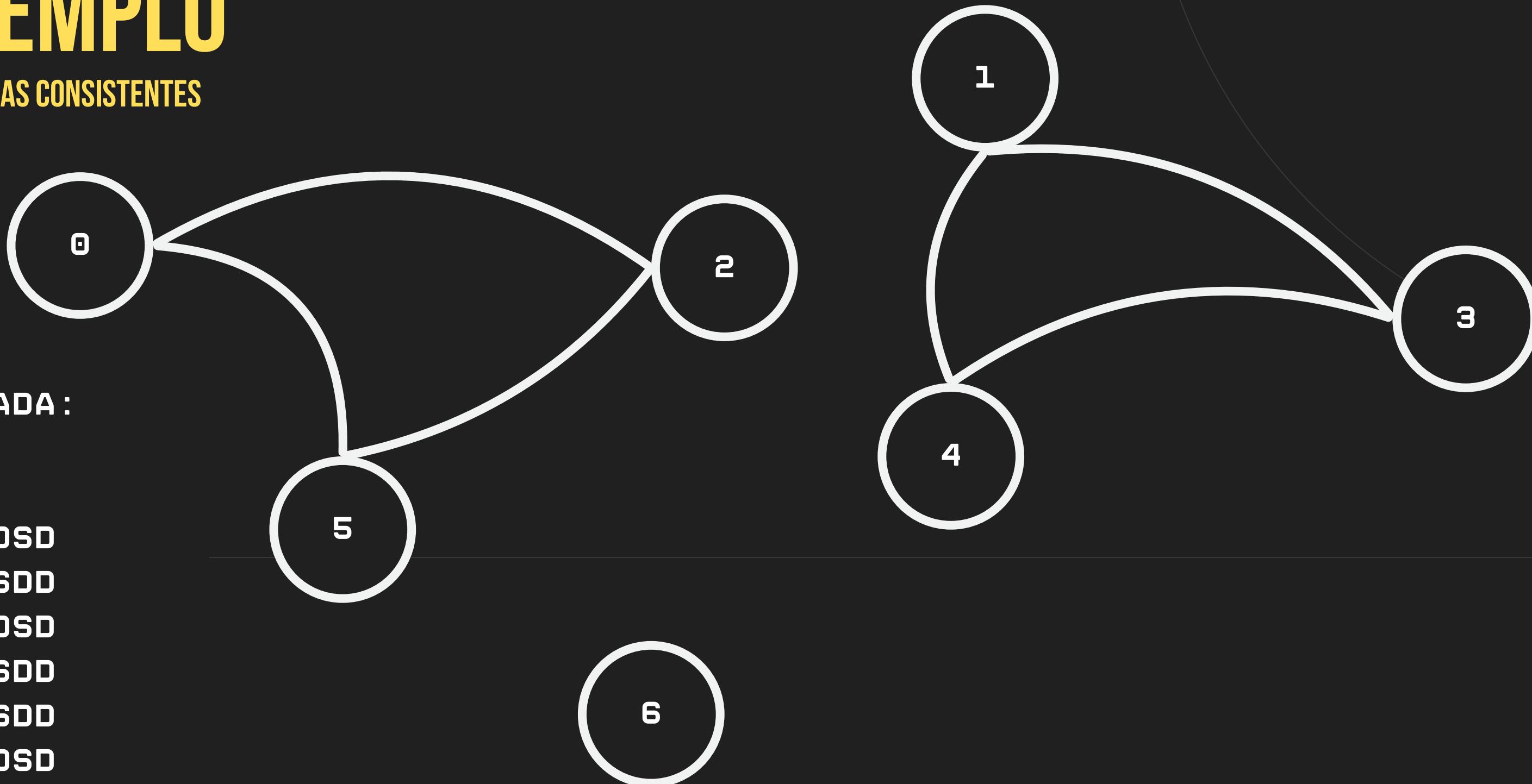


# EXEMPLO

COM CASAS CONSISTENTES

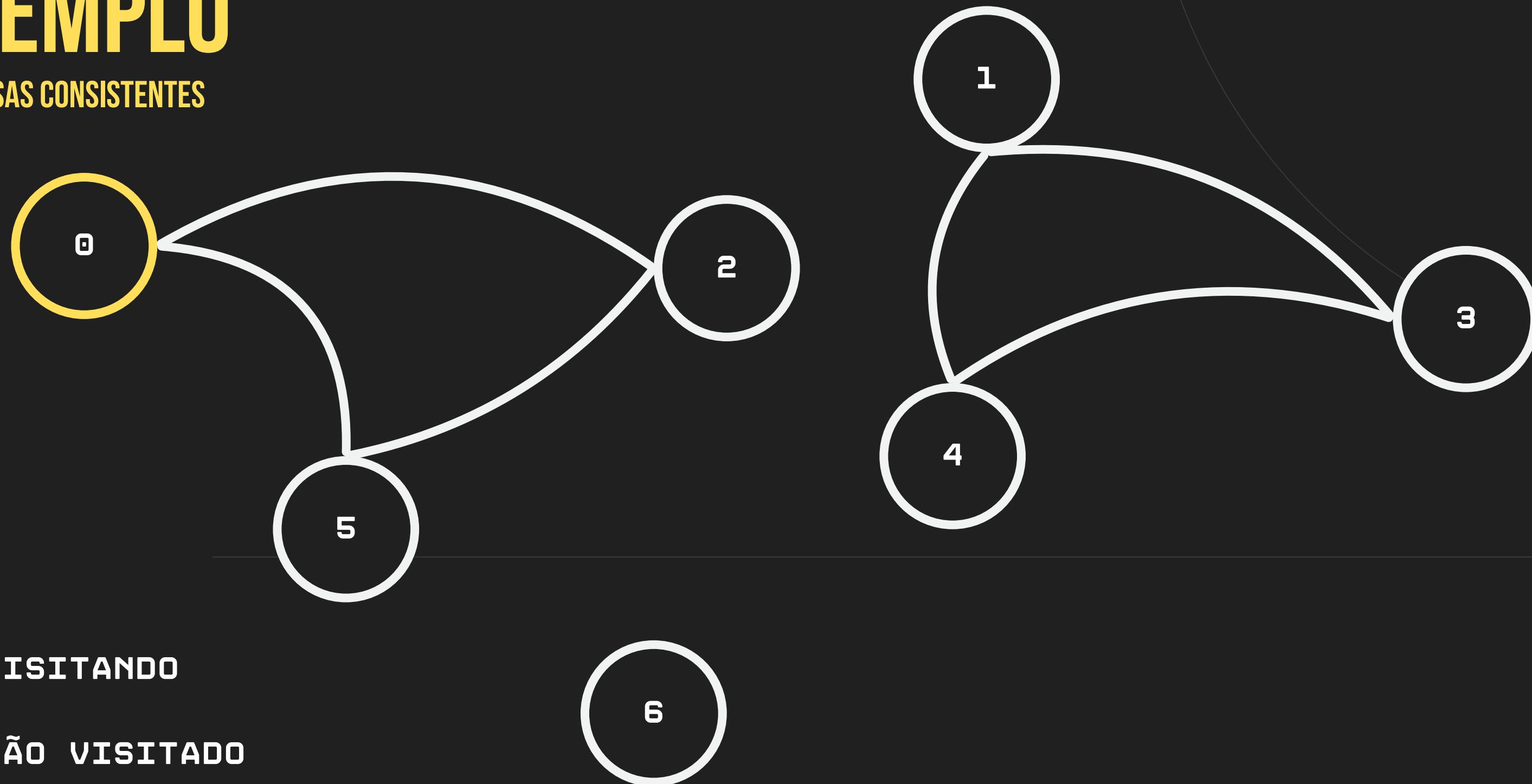
ENTRADA:

7  
SDSDDSD  
DSDSSDD  
SDSDDSD  
DSDSSDD  
DSDSSDD  
SDSDDSD  
DDDDDDDS



# EXEMPLO

COM CASAS CONSISTENTES

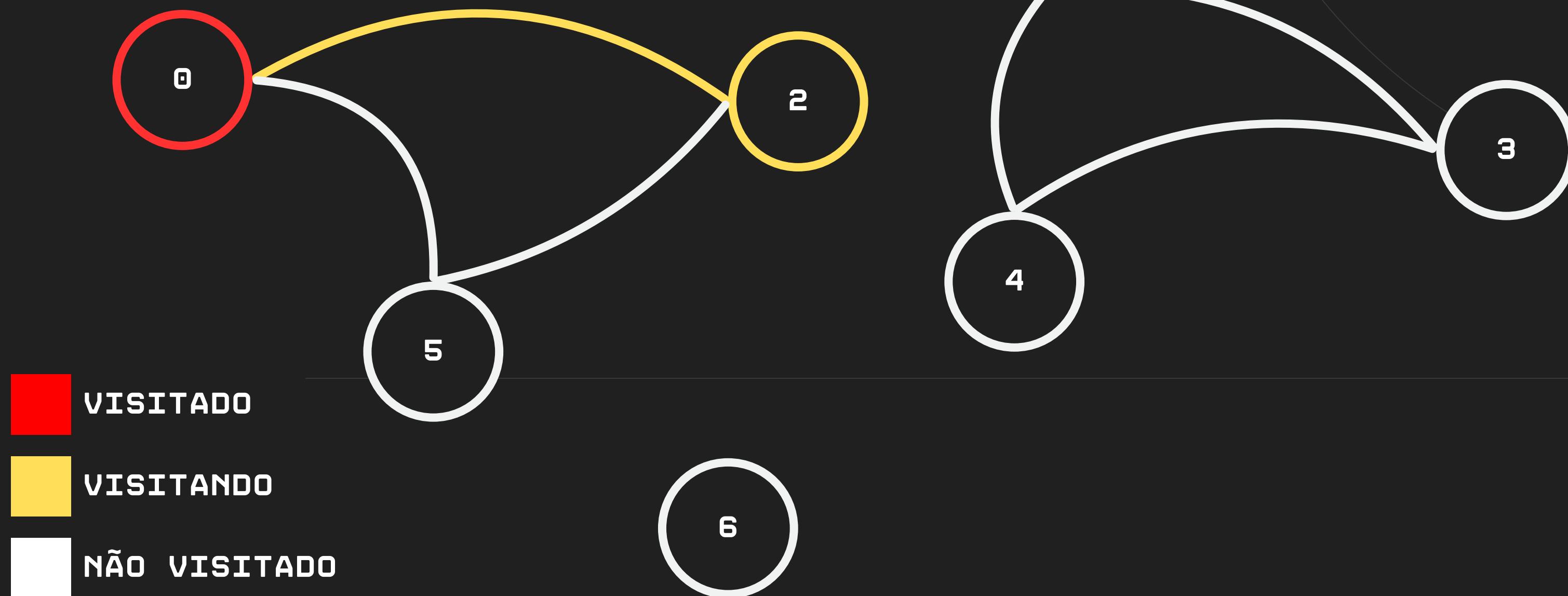


  VISITANDO

  NÃO VISITADO

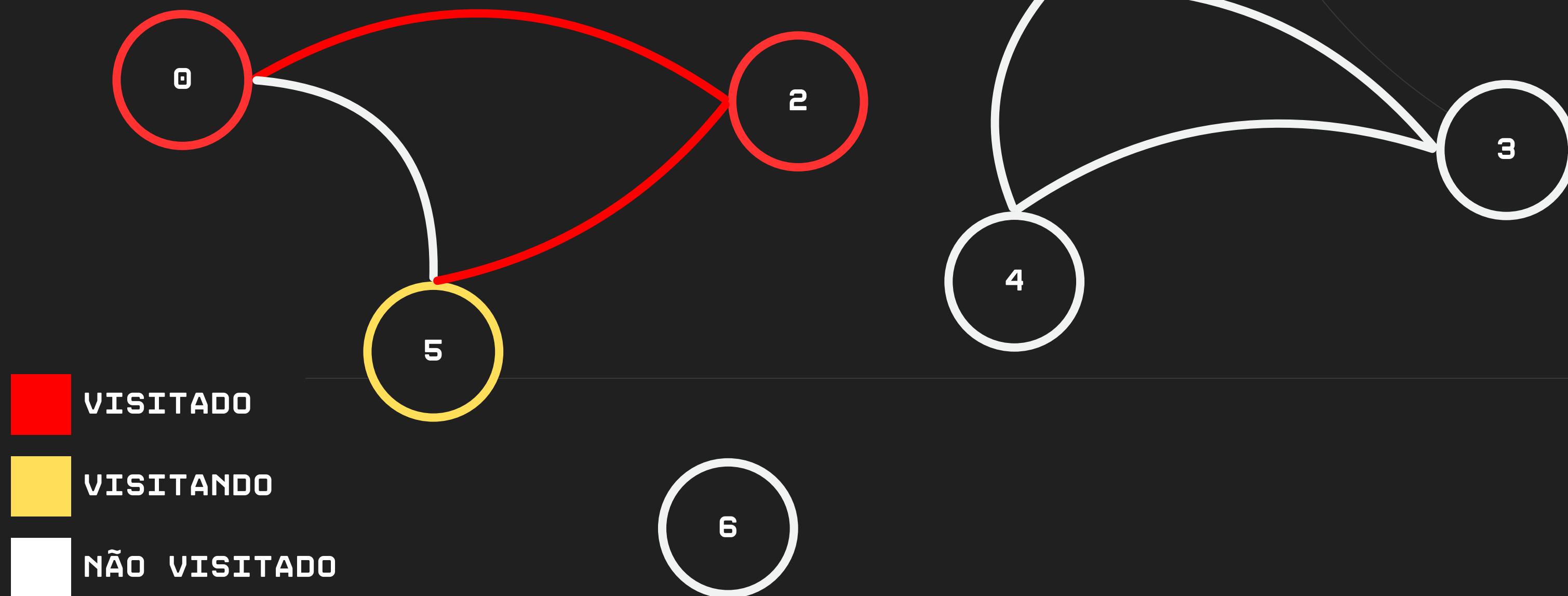
# EXEMPLO

COM CASAS CONSISTENTES



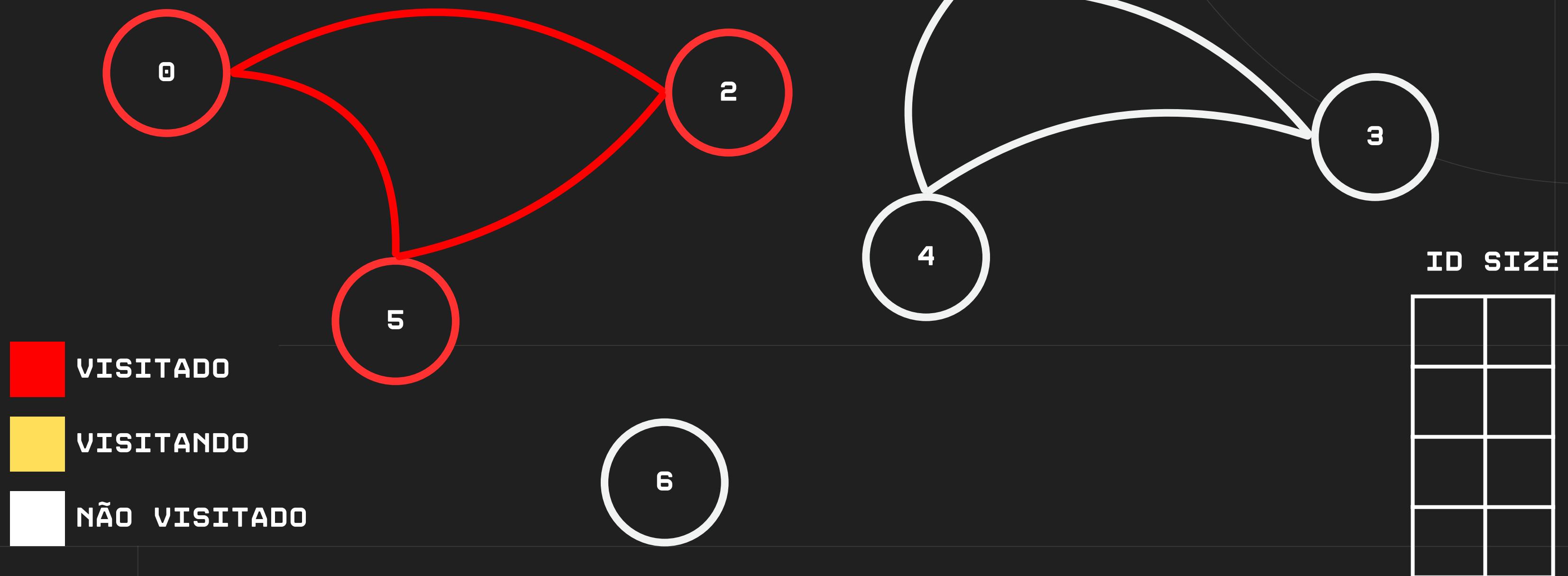
# EXEMPLO

COM CASAS CONSISTENTES



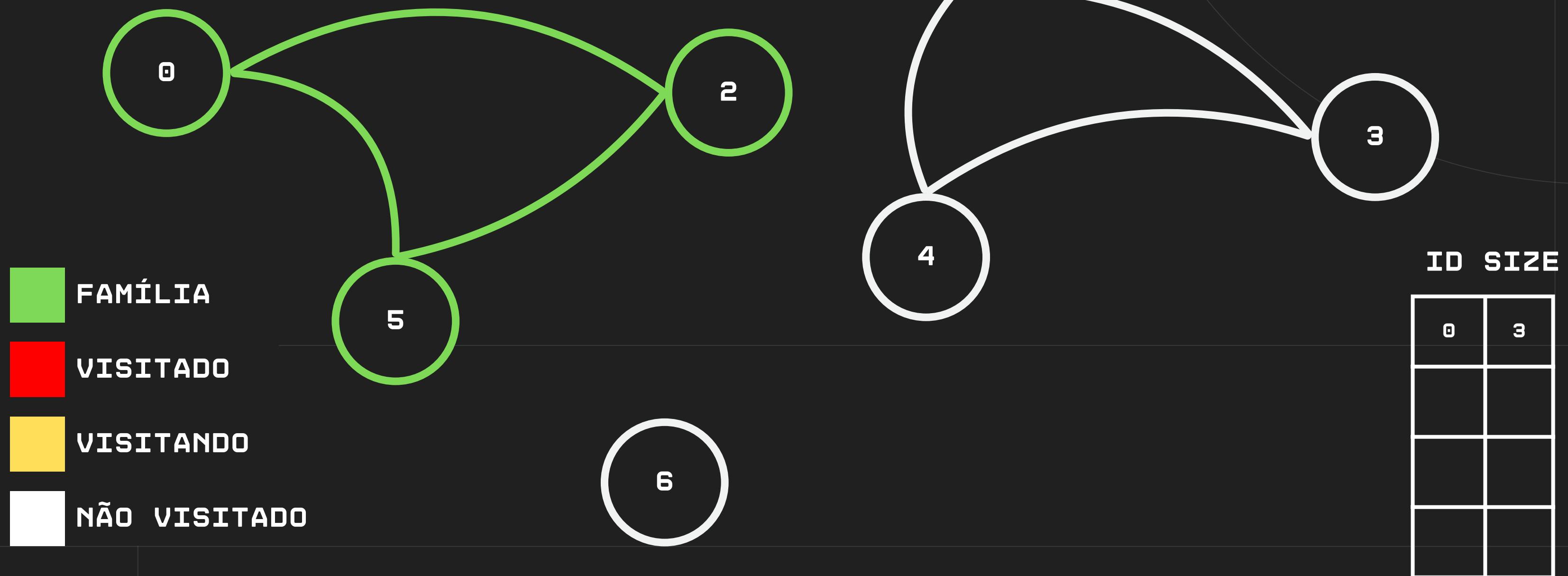
# EXEMPLO

COM CASAS CONSISTENTES



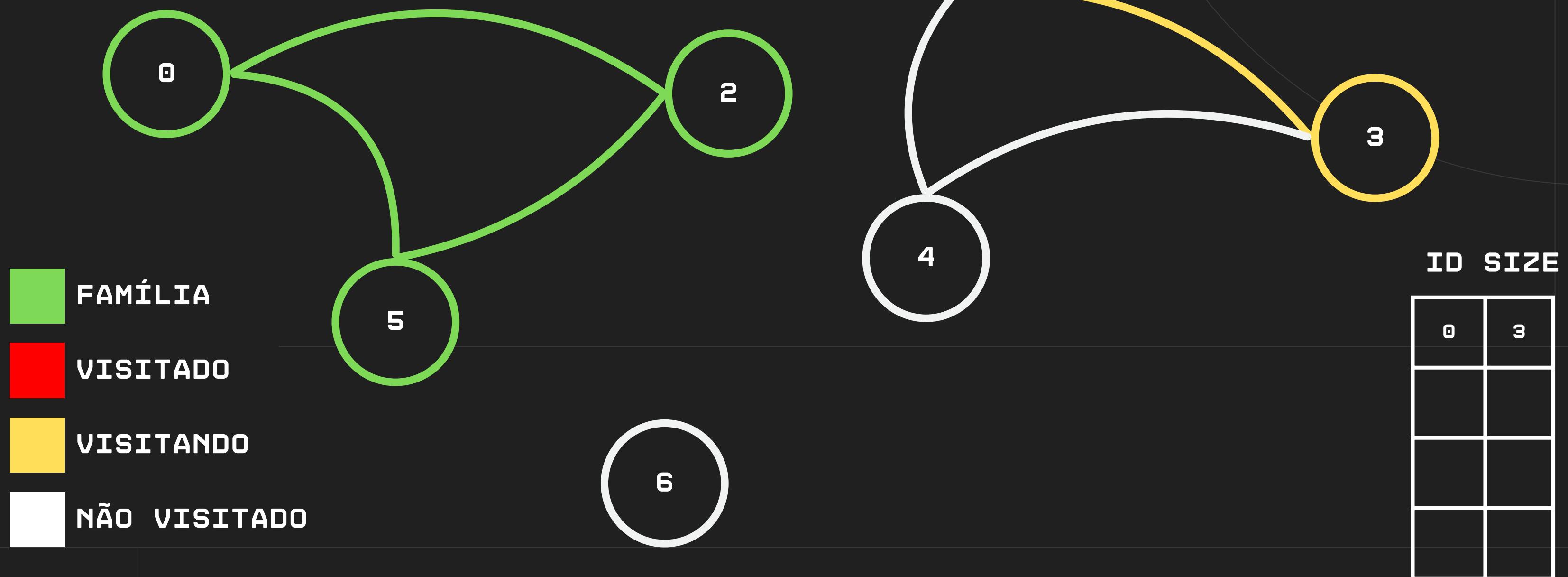
# EXEMPLO

COM CASAS CONSISTENTES



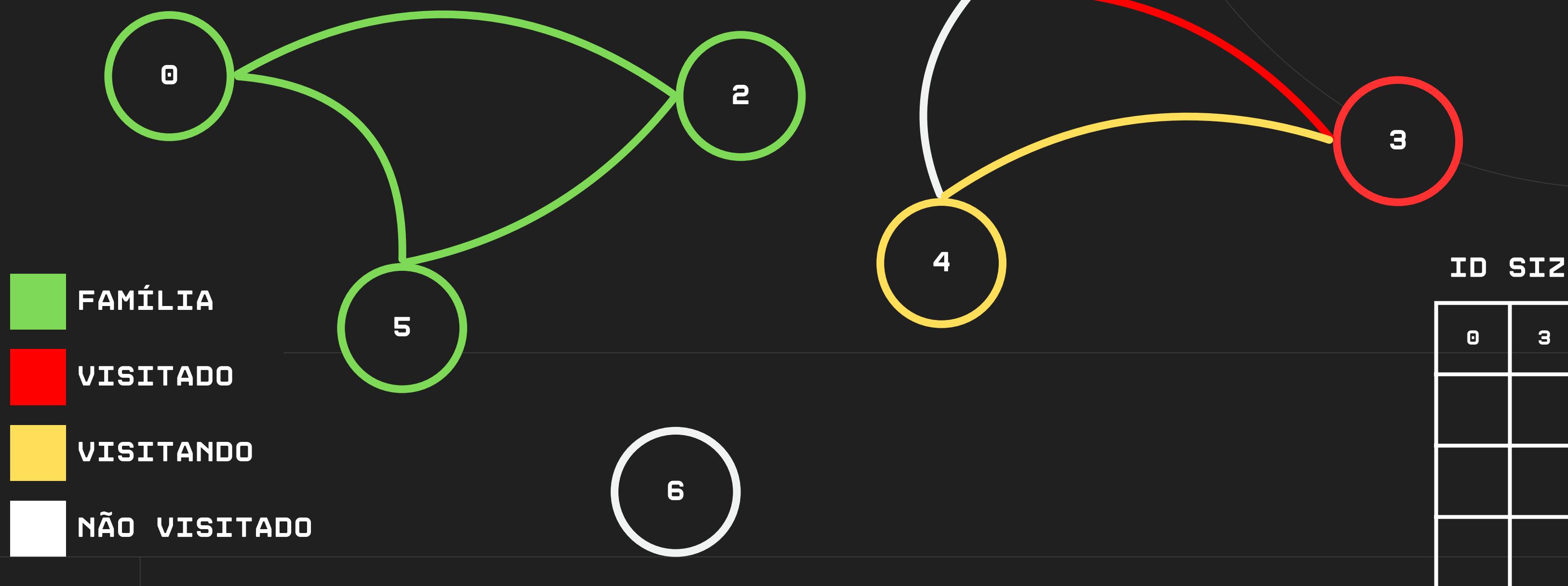
# EXEMPLO

COM CASAS CONSISTENTES



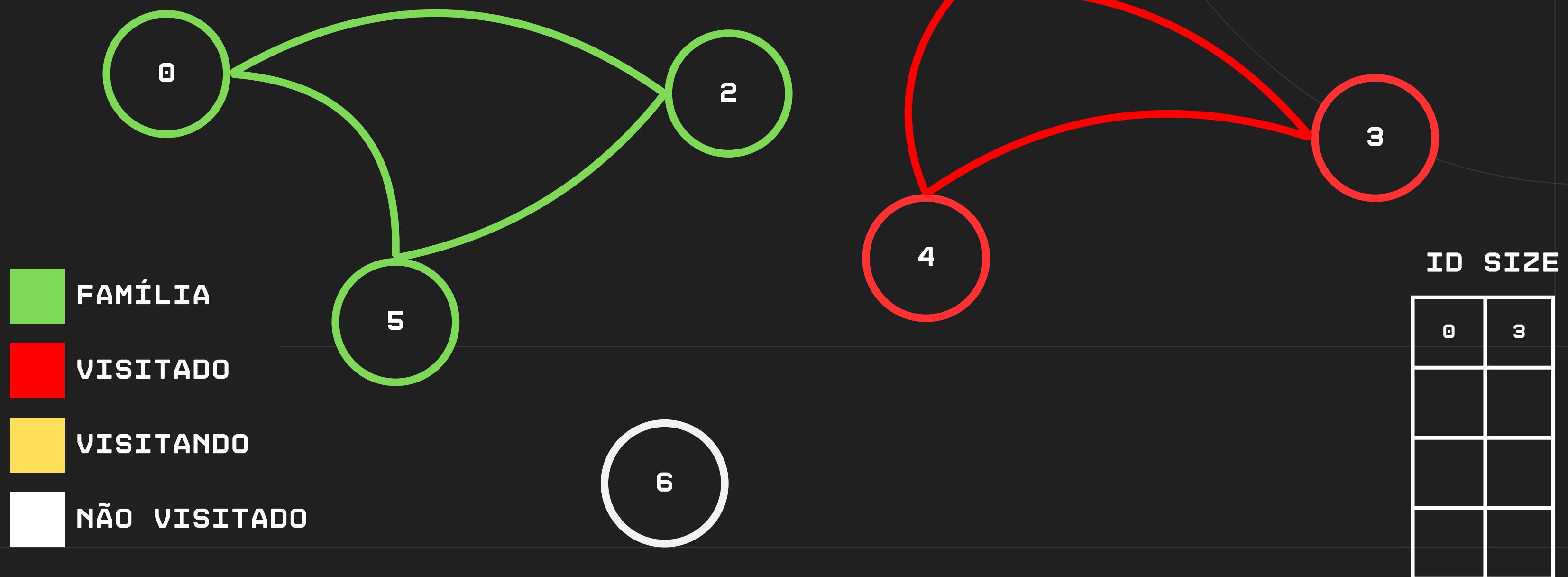
# EXEMPLO

COM CASAS CONSISTENTES



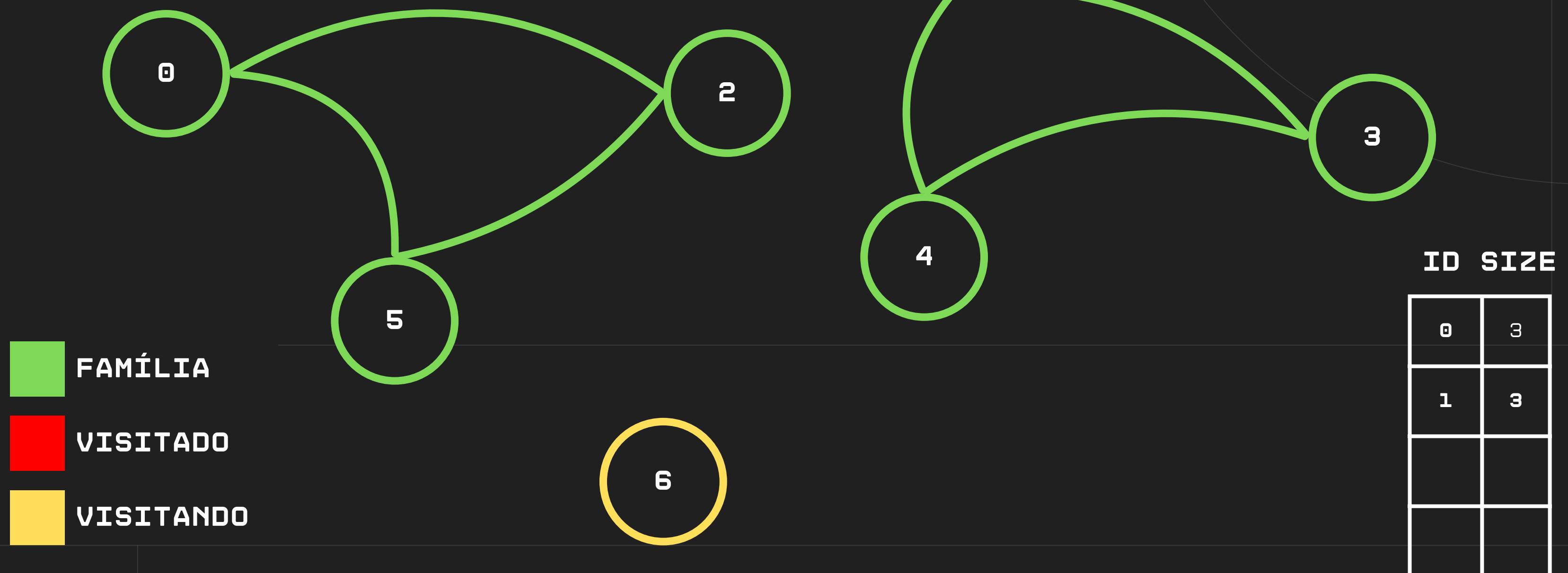
# EXEMPLO

COM CASAS CONSISTENTES



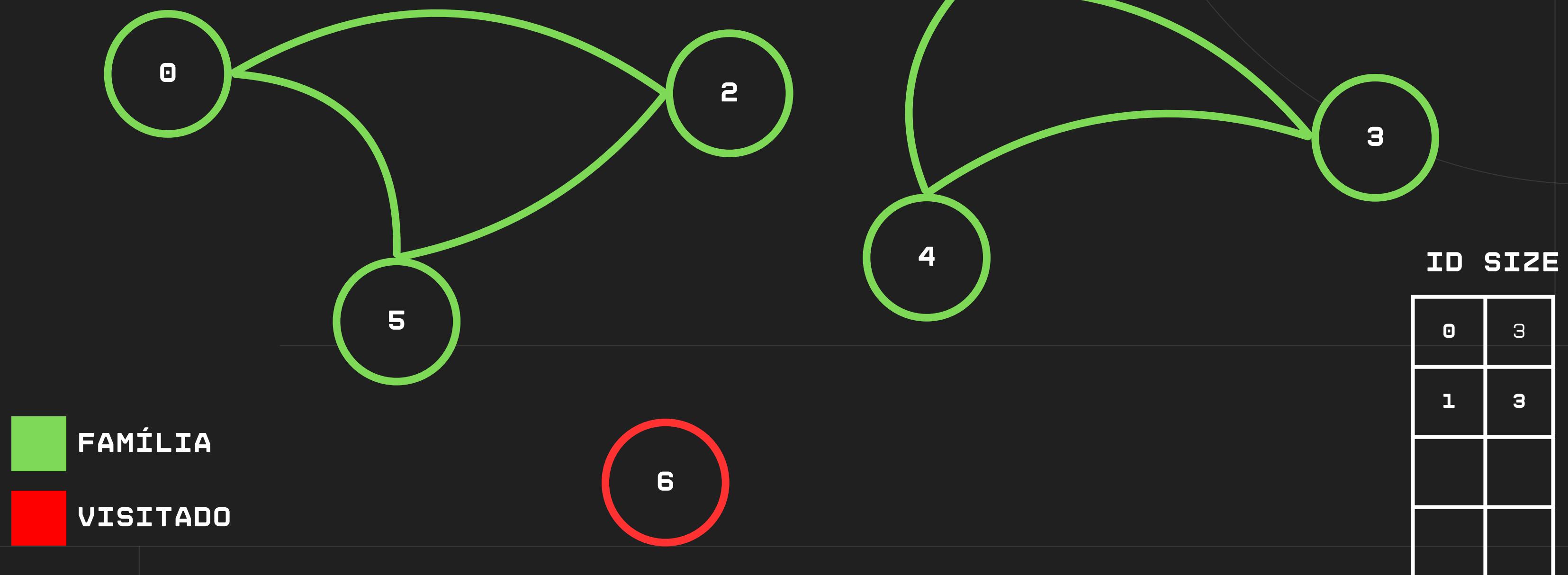
# EXEMPLO

COM CASAS CONSISTENTES



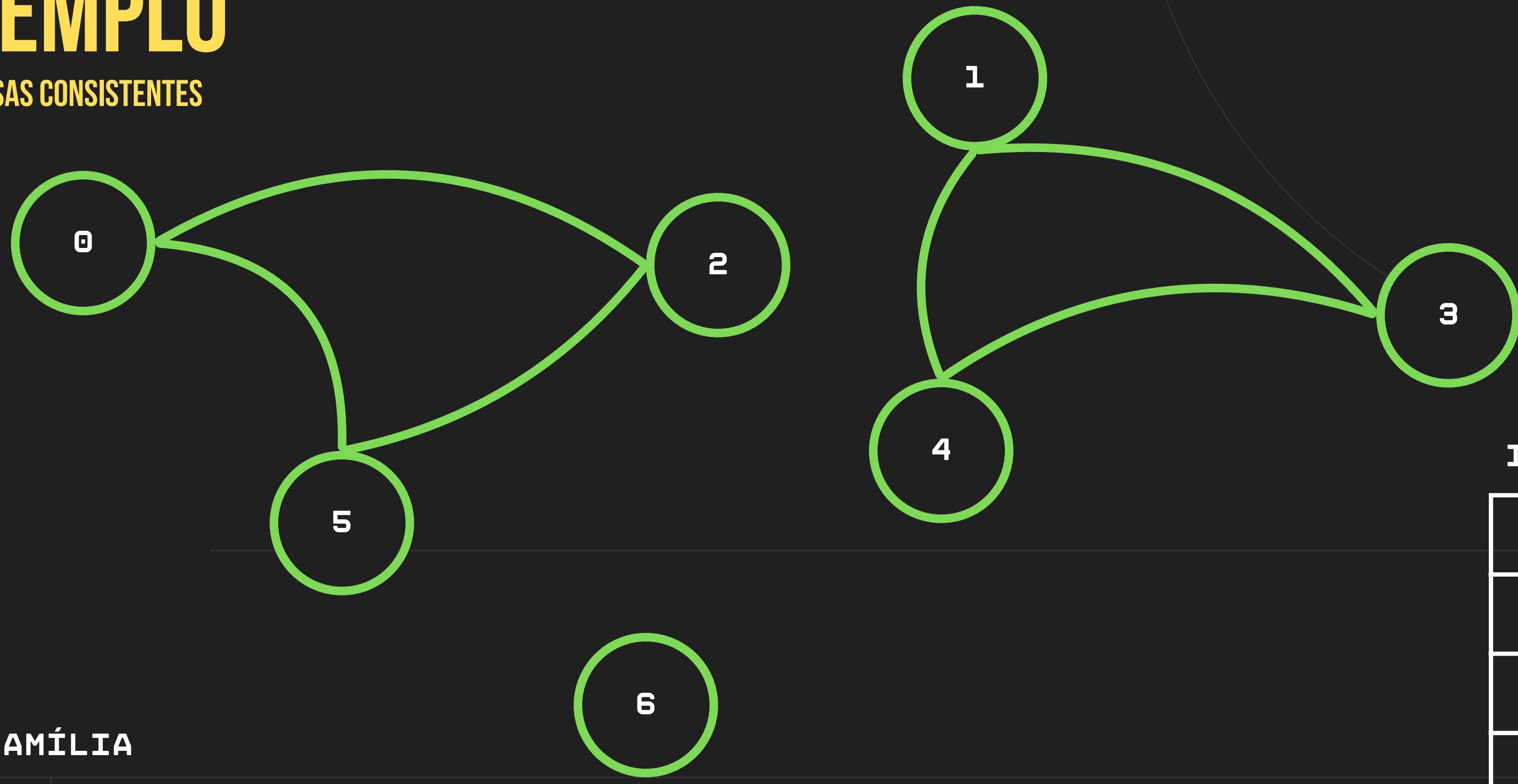
# EXEMPLO

COM CASAS CONSISTENTES

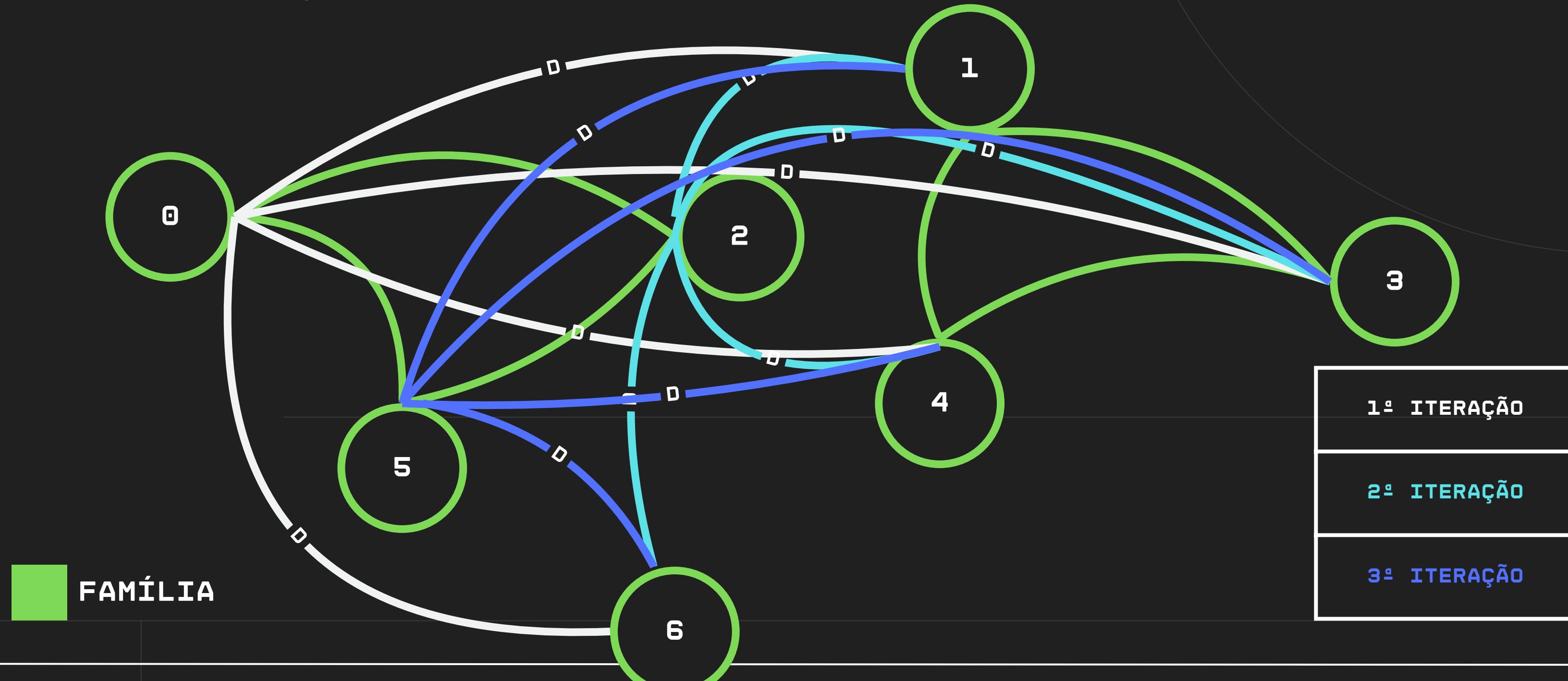


# EXEMPLO

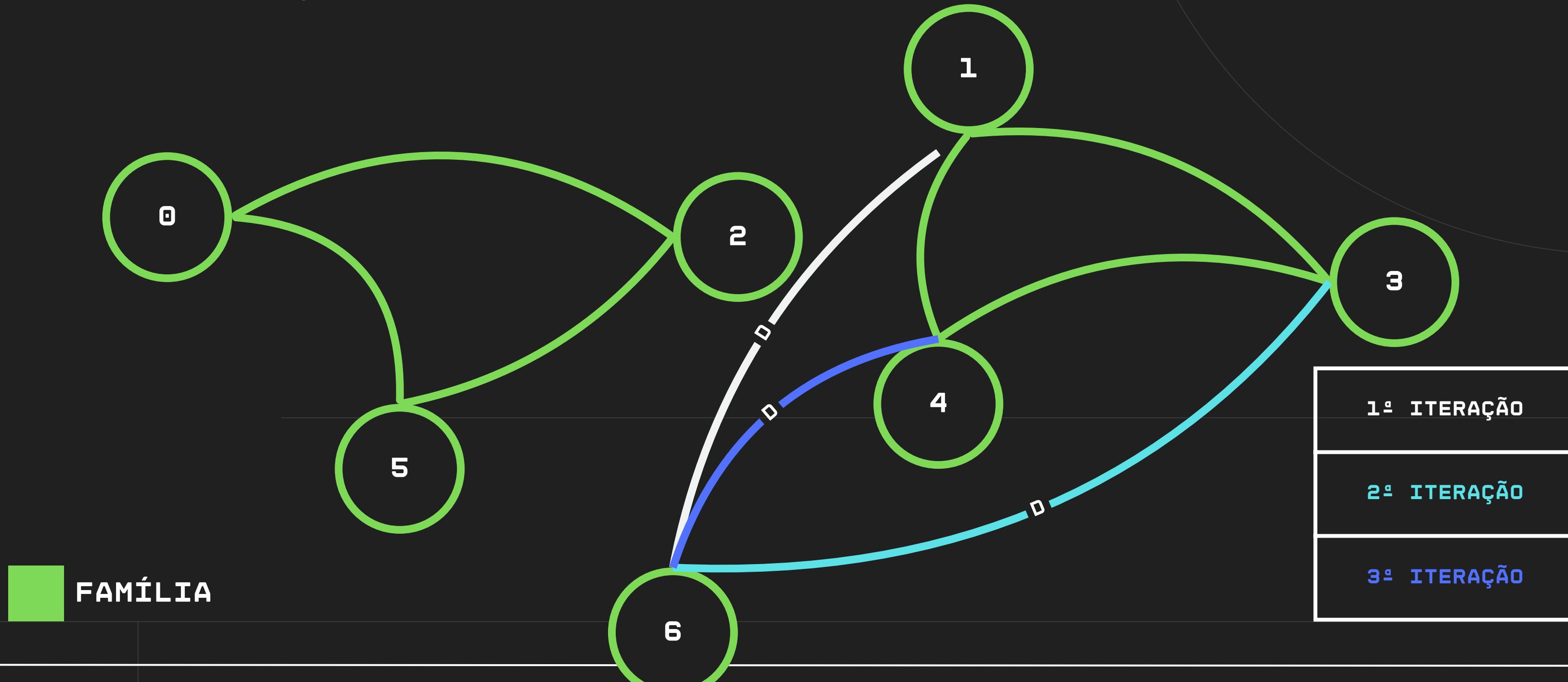
COM CASAS CONSISTENTES



# VERIFICAÇÃO DE INCONSISTÊNCIAS



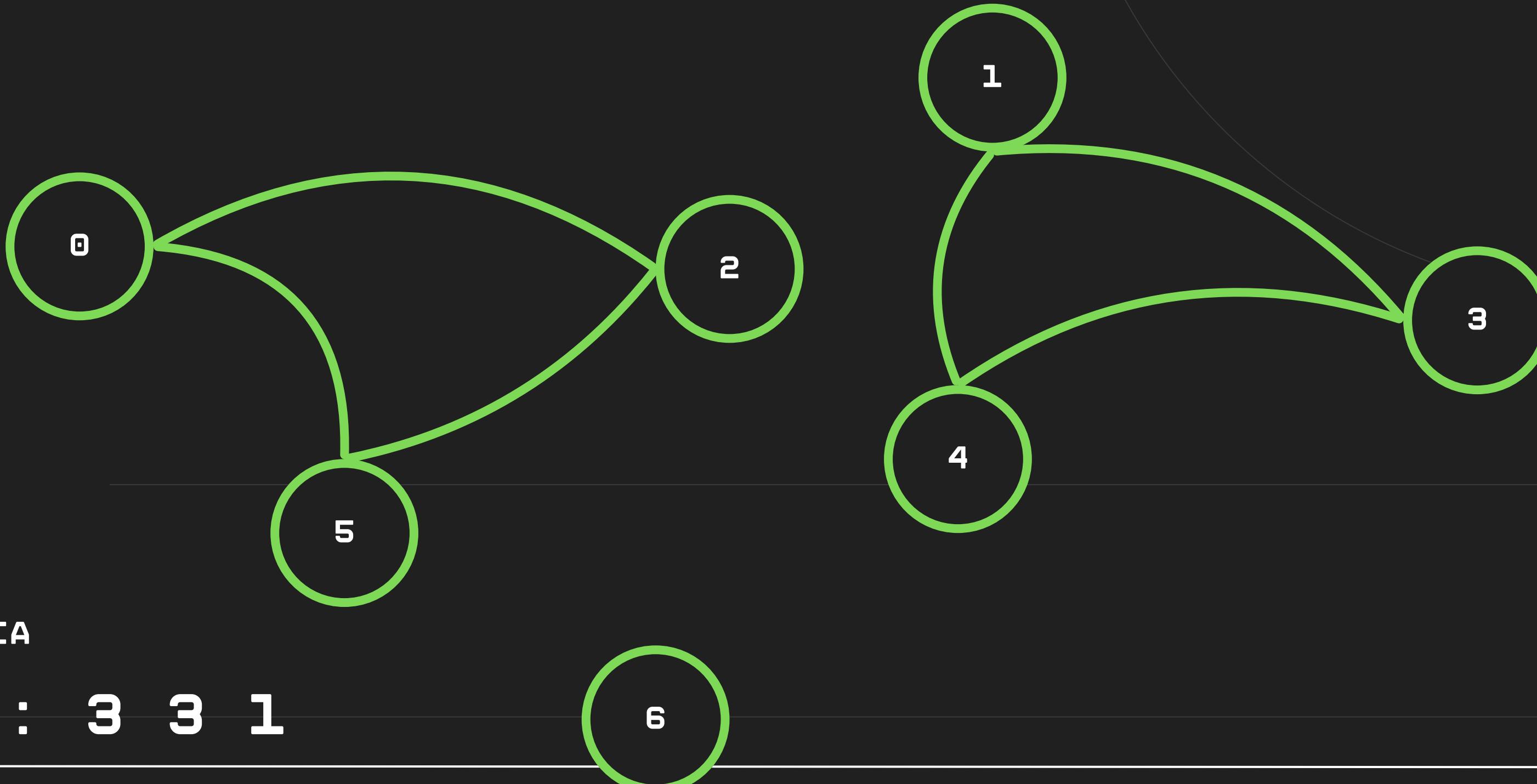
# VERIFICAÇÃO DE INCONSISTÊNCIAS



# SEM INCONSISTÊNCIAS

ID SIZE

ID	SIZE
0	3
1	3
2	1



FAMÍLIA

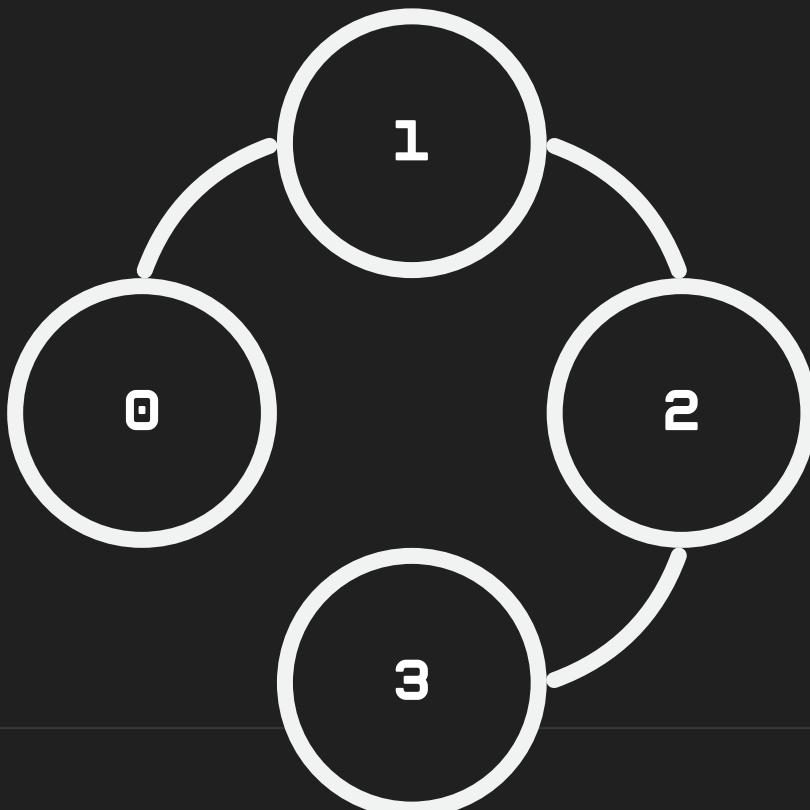
SAÍDA : 3 3 1

# EXEMPLO

COM CASAS INCONSISTENTES

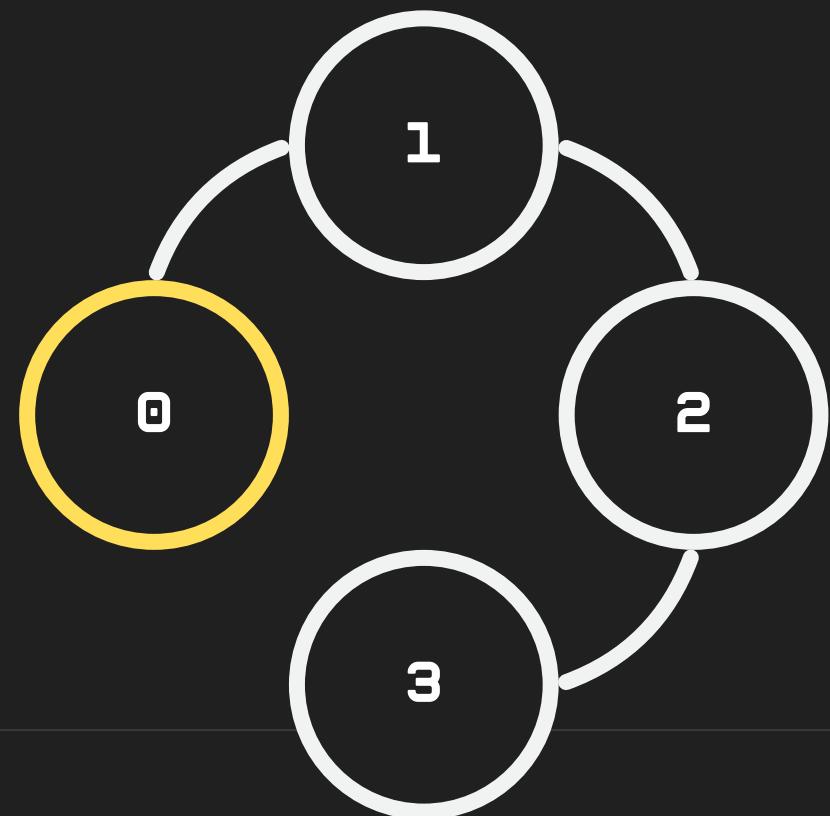
ENTRADA:

4  
SSDD  
SSSD  
DSSS  
DDSS



# EXEMPLO

COM CASAS INCONSISTENTES

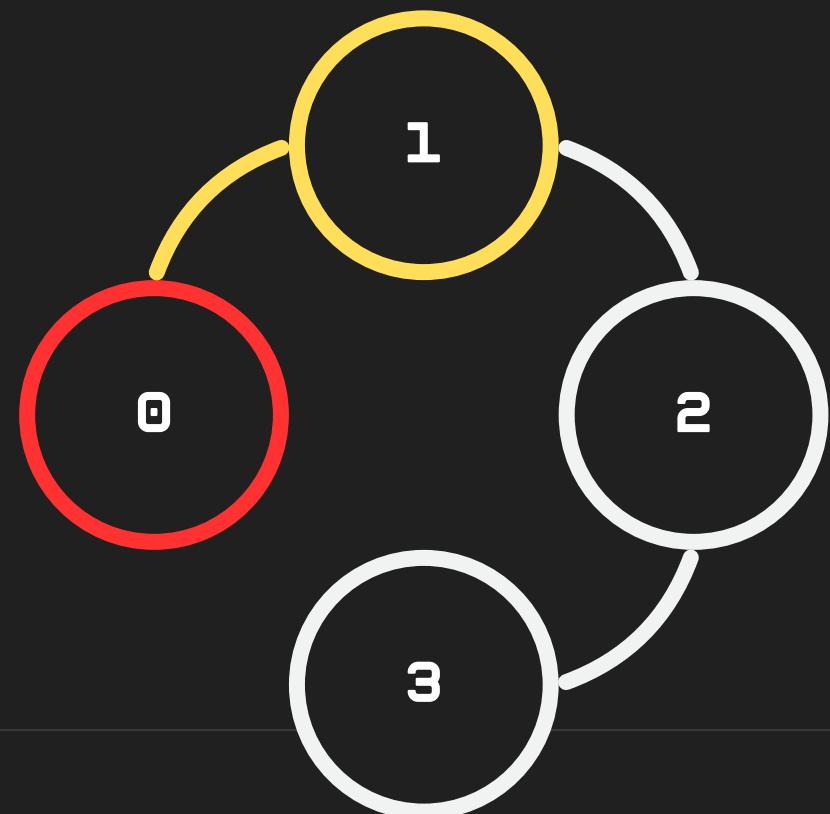


VISITANDO

NÃO VISITADO

# EXEMPLO

COM CASAS INCONSISTENTES



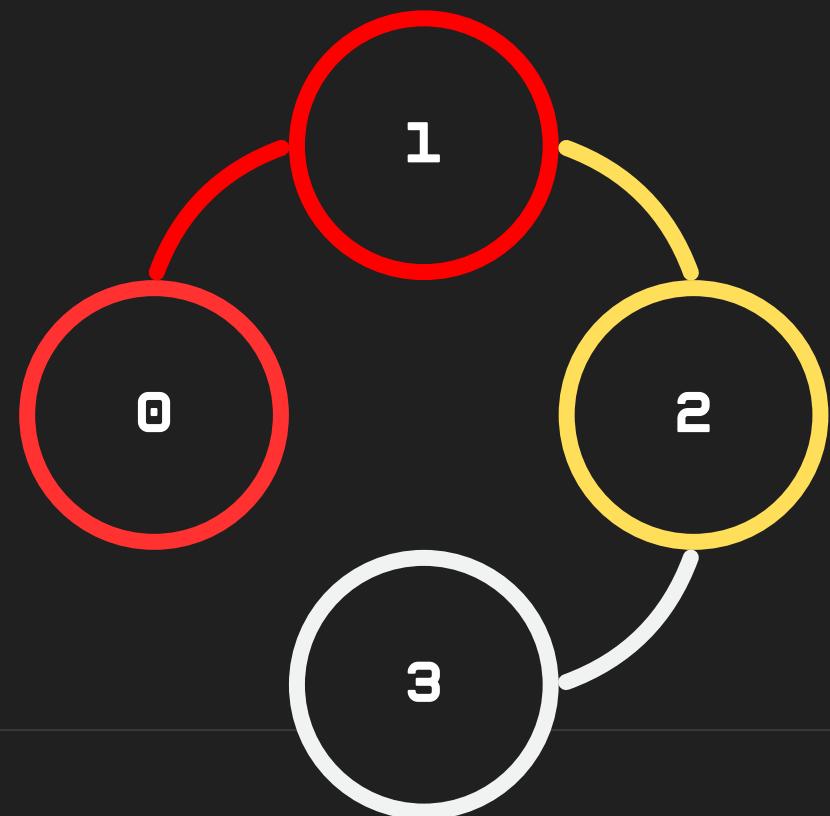
VISITADO

VISITANDO

NÃO VISITADO

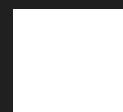
# EXEMPLO

COM CASAS INCONSISTENTES



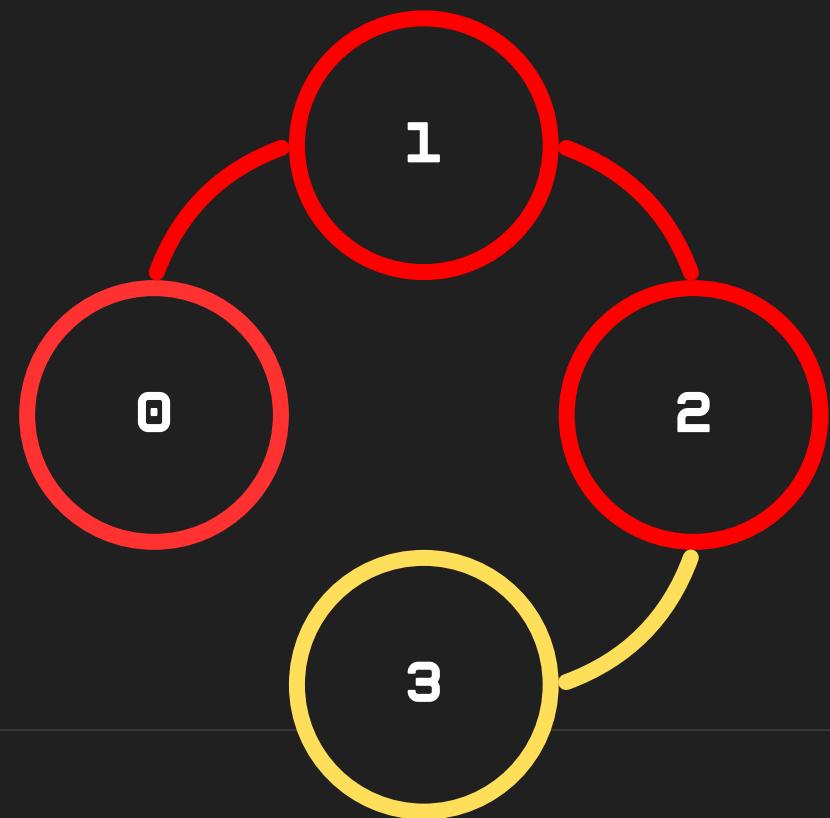
 VISITADO

 VISITANDO

 NÃO VISITADO

# EXEMPLO

COM CASAS INCONSISTENTES

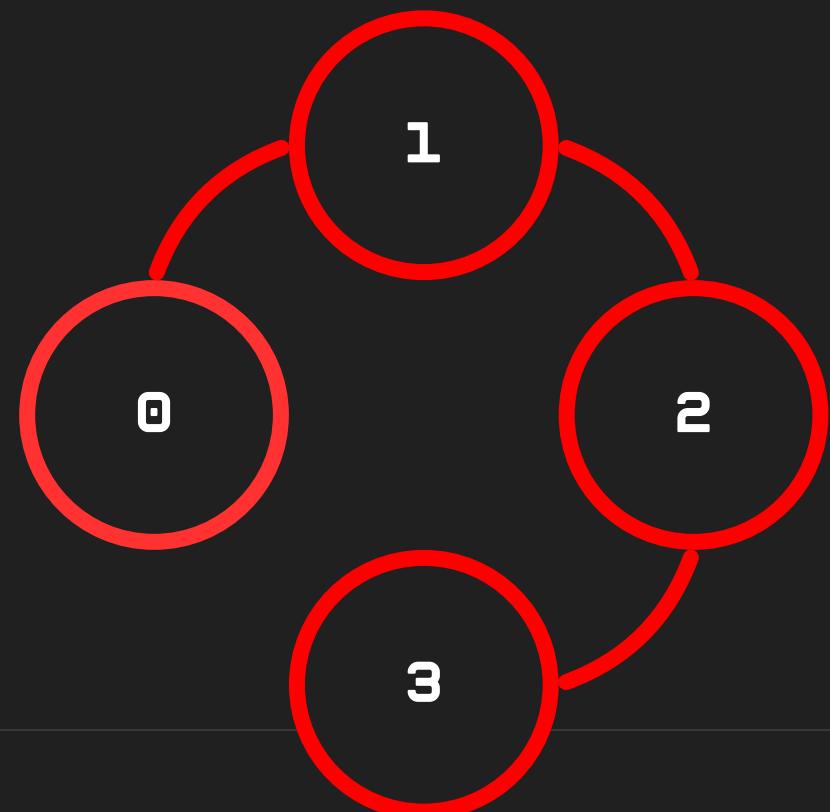


 VISITADO

 VISITANDO

# EXEMPLO

COM CASAS INCONSISTENTES

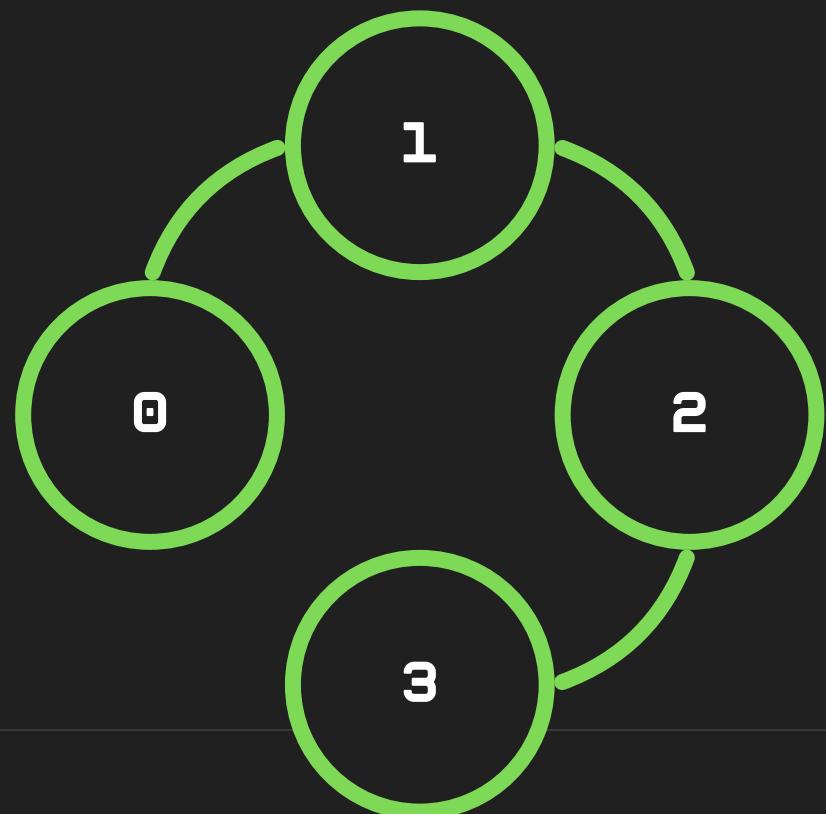


 VISITADO

ID	SIZE

# EXEMPLO

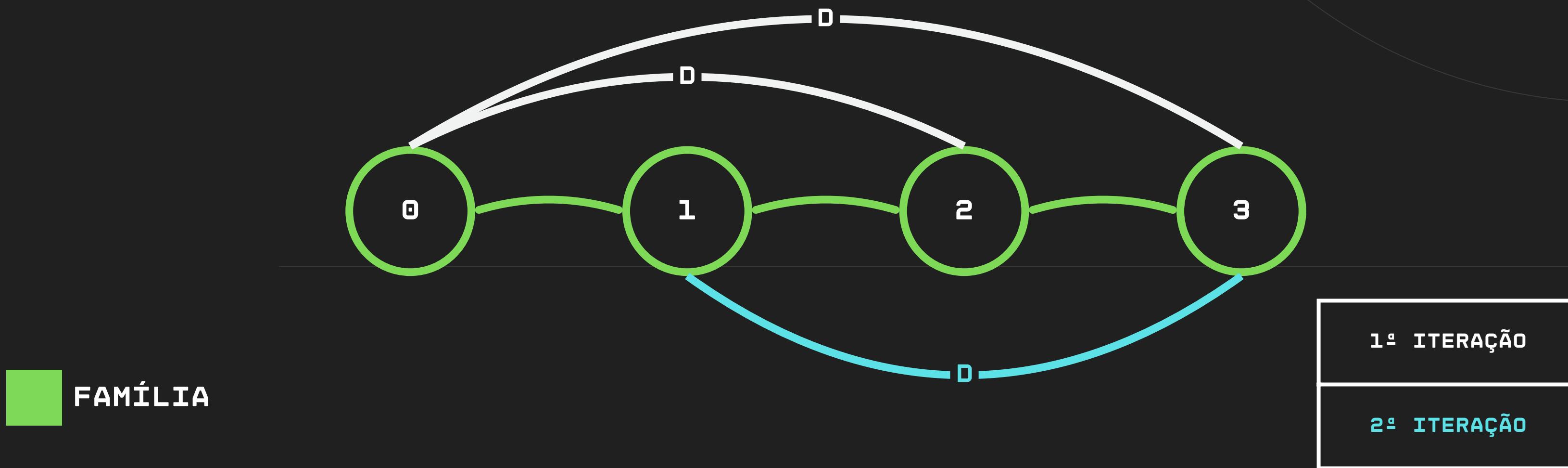
COM CASAS INCONSISTENTES



 FAMÍLIA

ID	SIZE
0	4

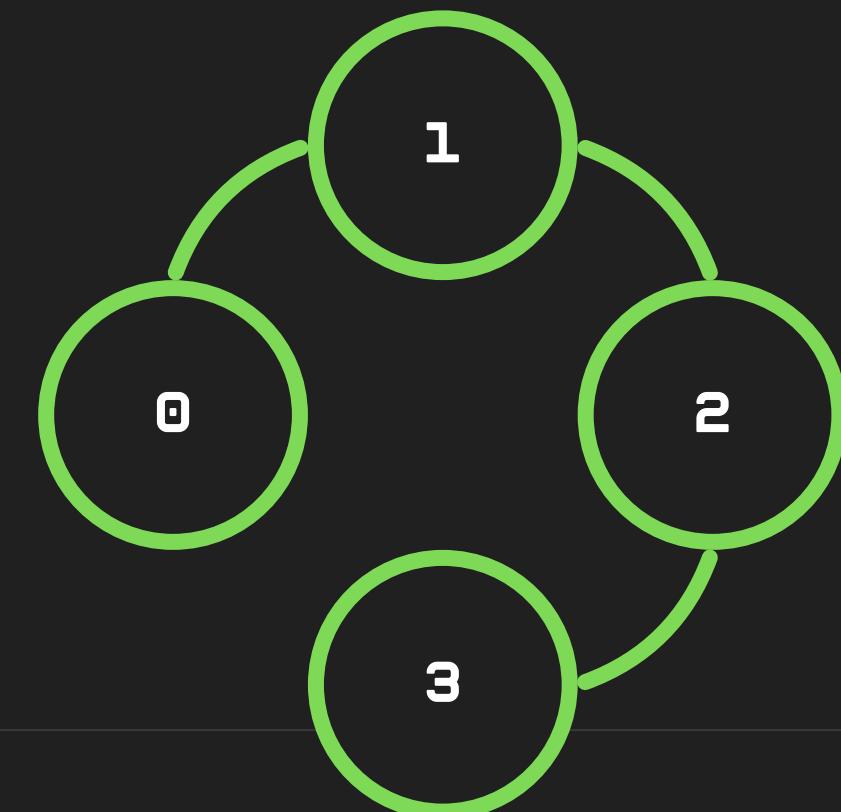
# VERIFICAÇÃO DE INCONSISTÊNCIAS



# INCONSISTENTE

ID SIZE

0	4



 FAMÍLIA

SAÍDA : -1

# ALGORITMO



# OUTRAS APLICAÇÕES

Além das utilizações comuns, como detecção de ciclos e ordenação topológica, a **DFS** é fundamental para identificar componentes fortemente conectados em grafos direcionados, encontrar pontes e articulações que, se removidas, aumentam a desconectividade do grafo, e verificar se um grafo é bipartido. Ela também é amplamente empregada em problemas de exploração de labirintos, construção de *spanning trees* e em técnicas de *backtracking* para resolver quebra-cabeças e enumerar todas as possibilidades em determinadas situações. A capacidade da **DFS** de explorar profundamente cada ramo antes de retroceder a torna uma ferramenta poderosa para análise de estruturas e das propriedades de grafos.

# PROBLEMAS

Desenhando Labirintos - <https://judge.beecrowd.com/pt/problems/view/1076>

Componentes Conexos - <https://judge.beecrowd.com/pt/problems/view/1082>

# OBRIGADO PELA ATENÇÃO

Grupo de Computação Competitiva

