

Componentes Conexas

Algoritmos e Estruturas de Dados - 2025/02

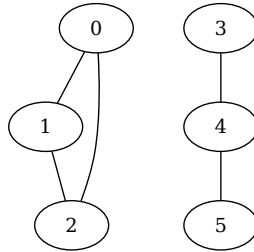
Última atualização: February 19, 2026

Seja $G = (V, A)$ um grafo não direcionado sem pesos. Um subconjunto $C \subseteq V$ de vértices é um *componente conexo* se e somente se

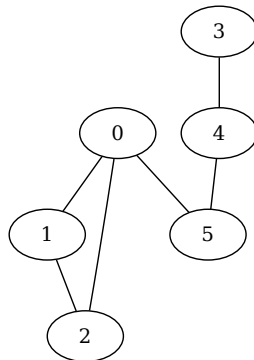
1. para todos par de vértices $v, w \in C$ existe um caminho que começa em v e termina em w
2. para todo $v \in C$ e $w \in V - C$, não existe caminho entre v e w .

Vamos praticar um pouco a identificação de componentes conexas?

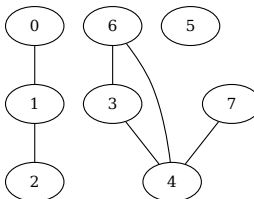
Exercício: Quantos componentes conexos tem o grafo abaixo? Anote na figura quais são.



Exercício: Quantos componentes conexos tem o grafo abaixo? Anote na figura quais são.



Exercício: Quantos componentes conexos tem o grafo abaixo? Anote na figura quais são.



Vamos agora exercitar nossa compreensão dessa definição com algumas questões teóricas.

Exercício: Dado um grafo $G = (V, A)$, $|V| > 0$, quais são os números mínimo e máximo de componentes conexos?

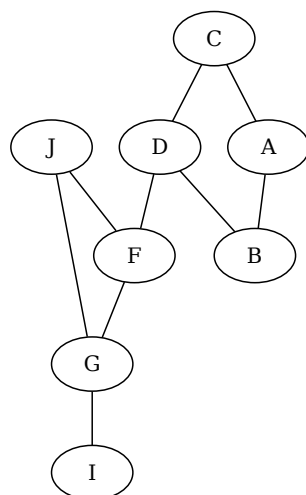
Exercício: O item 1 da definição só diz que o início e o fim do caminho tem que estar em C . É possível existir um caminho entre dois vértices em C que contenha ao menos um vértice que não está em C ?

Um algoritmo simples para componentes conexas

Em IA e TecProg já vimos um algoritmo que lembra muito a definição de Componente Conexo: Busca em Profundidade (DFS)! Como sempre, vamos começar simulando essa ideia para entender como o algoritmo pode ajudar.

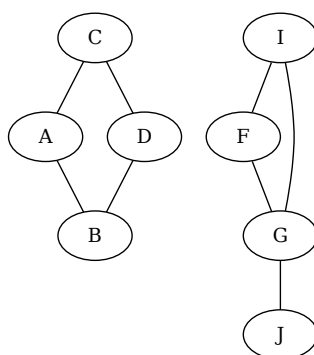
Exercício: Dado o grafo abaixo, comece uma DFS a partir do vértice D e numere os vértices de acordo com quando eles forem visitados.

A ordem de visitação dos vizinhos de um vértice é alfabética.



Exercício: Dado o grafo abaixo, comece uma DFS a partir do vértice D e numere os vértices de acordo com quando eles forem visitados.

A ordem de visitação dos vizinhos de um vértice é alfabética.



E o segundo componente?

Precisaríamos iniciar uma nova *DFS* para conseguir rotular os elementos do segundo componente. Em geral vamos precisar de **uma DFS para cada componente do grafo**.

Várias coisas relevantes acontecem nos exemplos acima:

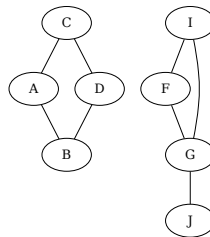
1. no primeiro todos os nós são visitados. Isso acontece pois há somente um componente conexo.
2. ainda no primeiro, quando temos um ciclo (como na sequência D-B-A-C) chega um momento em que encontramos um vértice que já foi visitado e que não é o predecessor.
3. no segundo exemplo iniciamos no vértice D, que não conecta com o componente de baixo. Por isso alguns nós ficam sem número quando acabamos a DFS

Importante

Entenda bem as propriedades acima antes de continuar! Elas são importantes no desenvolvimento desse e de outros algoritmos baseados em DFS

Vamos agora formalizar nosso algoritmo **IDENTIFICA-COMPONENTES(G, LABELS)**. A saída do algoritmo será um array em que cada vértice contém um número identificador de seu componente (começando em 1). Dois vértices estão no mesmo componente se eles possuem o mesmo identificador. Vejamos um exemplo.

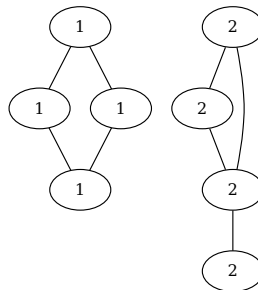
Entrada:



Saída: Duas saídas são possíveis:

- [1, 1, 1, 1, 2, 2, 2, 2] (se começamos em A, B, C ou D) **ou**
- [2, 2, 2, 2, 1, 1, 1, 1] (se começamos em F, G, I ou J).

Visualmente isso seria equivalente a:



Já sabemos que o IDENTIFICA-COMPONENTES será baseado em DFS e que esse algoritmo depende de algumas outras subrotinas. Você pode supor que os seguintes algoritmos estão disponíveis para usar como subrotinas.

1. NVERTICES(G) retorna o número de vértices do grafo
2. EH_VIZINHO(G, I, J) retorna VERDADEIRO se os vértices I e J são ligados por uma aresta
3. todos algoritmos do Array que já usamos em TecProg.

Com isso já temos o necessário para criar nosso algoritmo. Vamos tentar?

Algoritmo IDENTIFICA-COMPONENTES($G, LABELS$)

Dicas:

1. LABELS não vem inicializado. Como você pode inicializá-lo para que seja fácil diferenciar vértices não visitados de vértices que já tem rótulo?
2. Pode ser necessário criar mais de uma função.
3. Lembre-se que o número máximo de componentes conexas é $|V|$