Algoritmo e Estrutura de Dados II

João Gabriel Cavalcante França

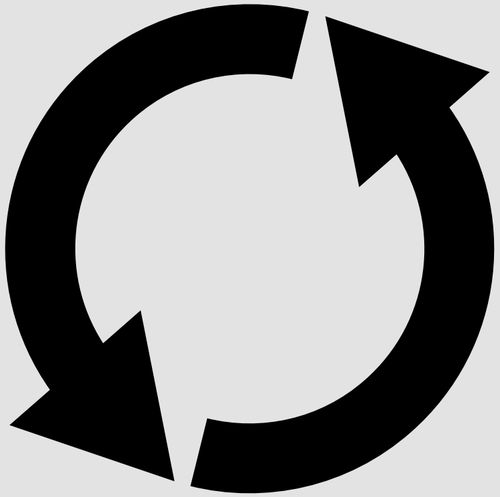
Matheus Franco Cascão Costa

Vitor Paulo Eterno Godoi

- **Detecção de Ciclos**
  - **Dirigidos**
  - **Não dirigidos**

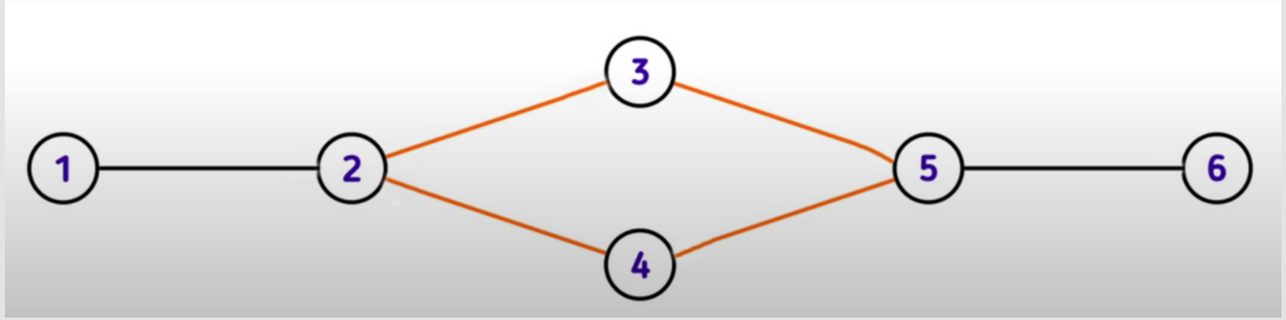
- **Algoritmo de Floyd-Warshall**

Detecção de Ciclos em Grafos Não Dirigidos



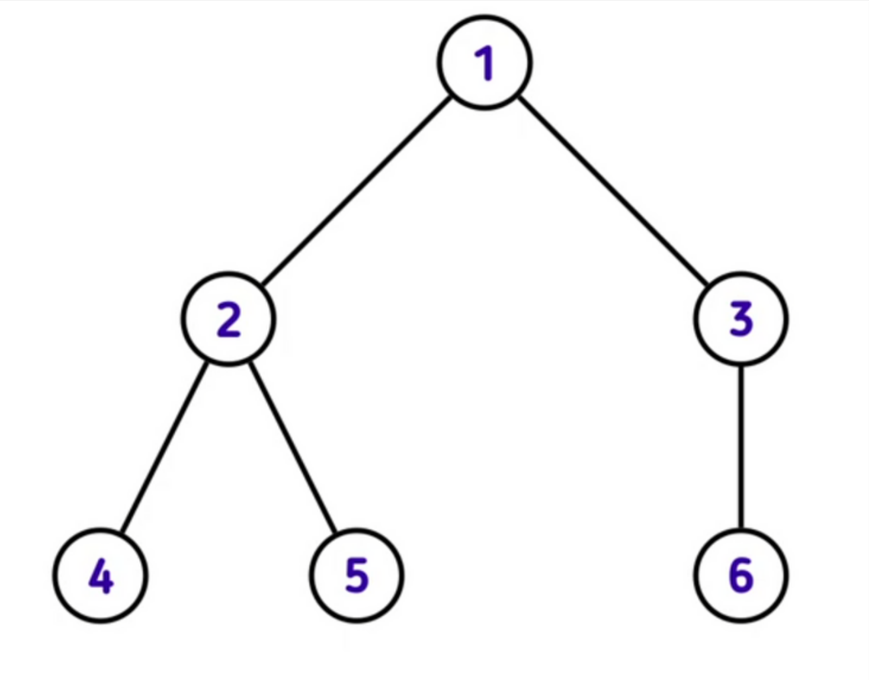
Definição de Ciclo

Um **ciclo** é um caminho, com três ou mais arestas distintas, cujos pontos de partida e de chegada são iguais.



Exemplo de grafo Acíclico

Para que essa distinção entre um grafo cíclico e acílico fique ainda mais clara, pode-se observar um exemplod e um grafo acíclico, nesse caso, uma **árvore:**



Um grafo é dito **acíclico** se não possui nenhum ciclo.

Não há como fazer um caminho dentro desse grafo em que um **vértice retorne para ele mesmo** com três ou mais arestas distintas.

Objetivo:

Dado um grafo não direcionado, cheque se o grafo contém um ciclo ou não.

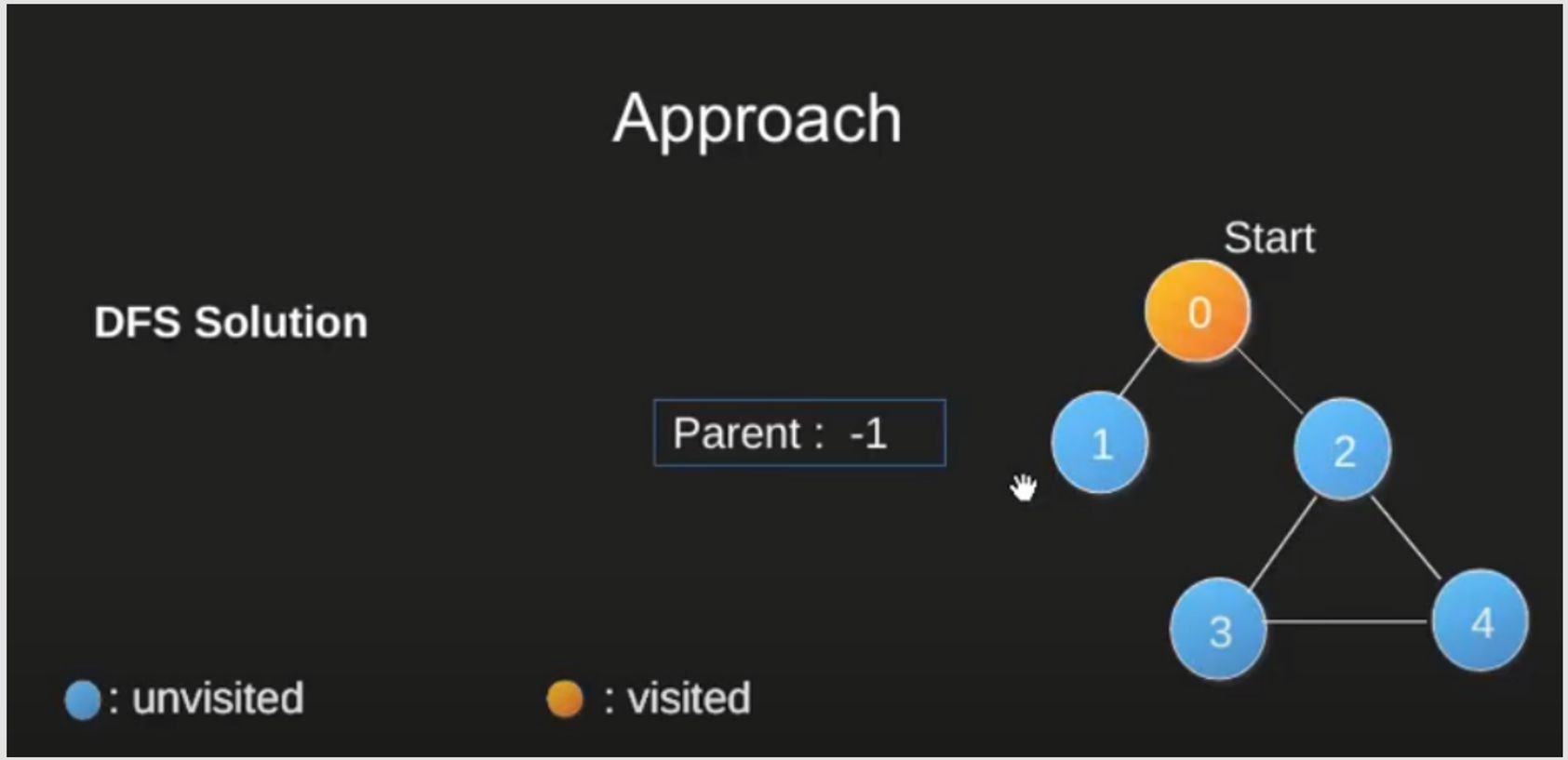
Possíveis algoritmos:

- **DFS**
- BFS
- Disjoint-set

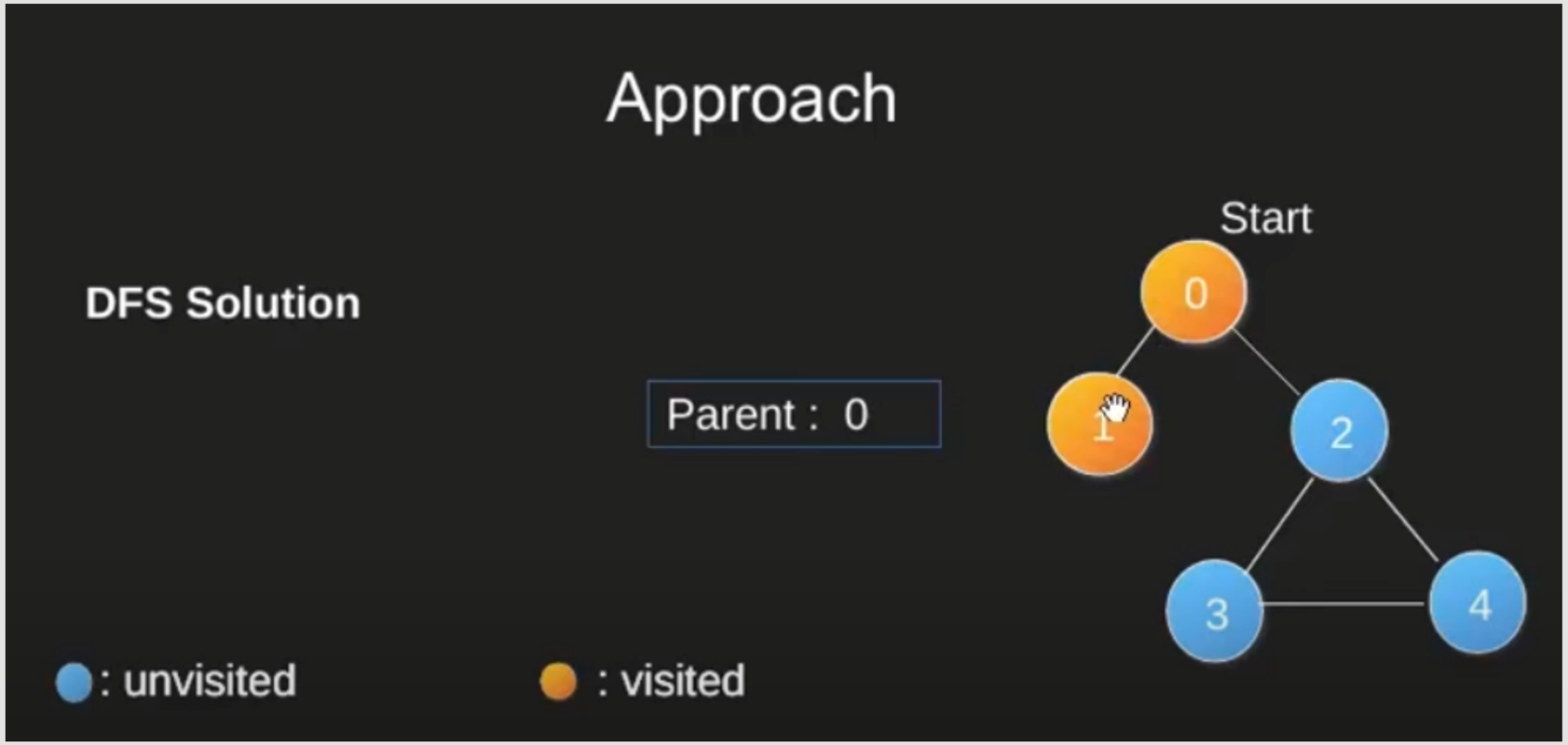
Ideia:

Para cada vértice visitado "v", se existir um vértice adjacente "u" tal que "u" já tenha sido visitado e "u" não seja pai de "v", então há um ciclo no grafo.

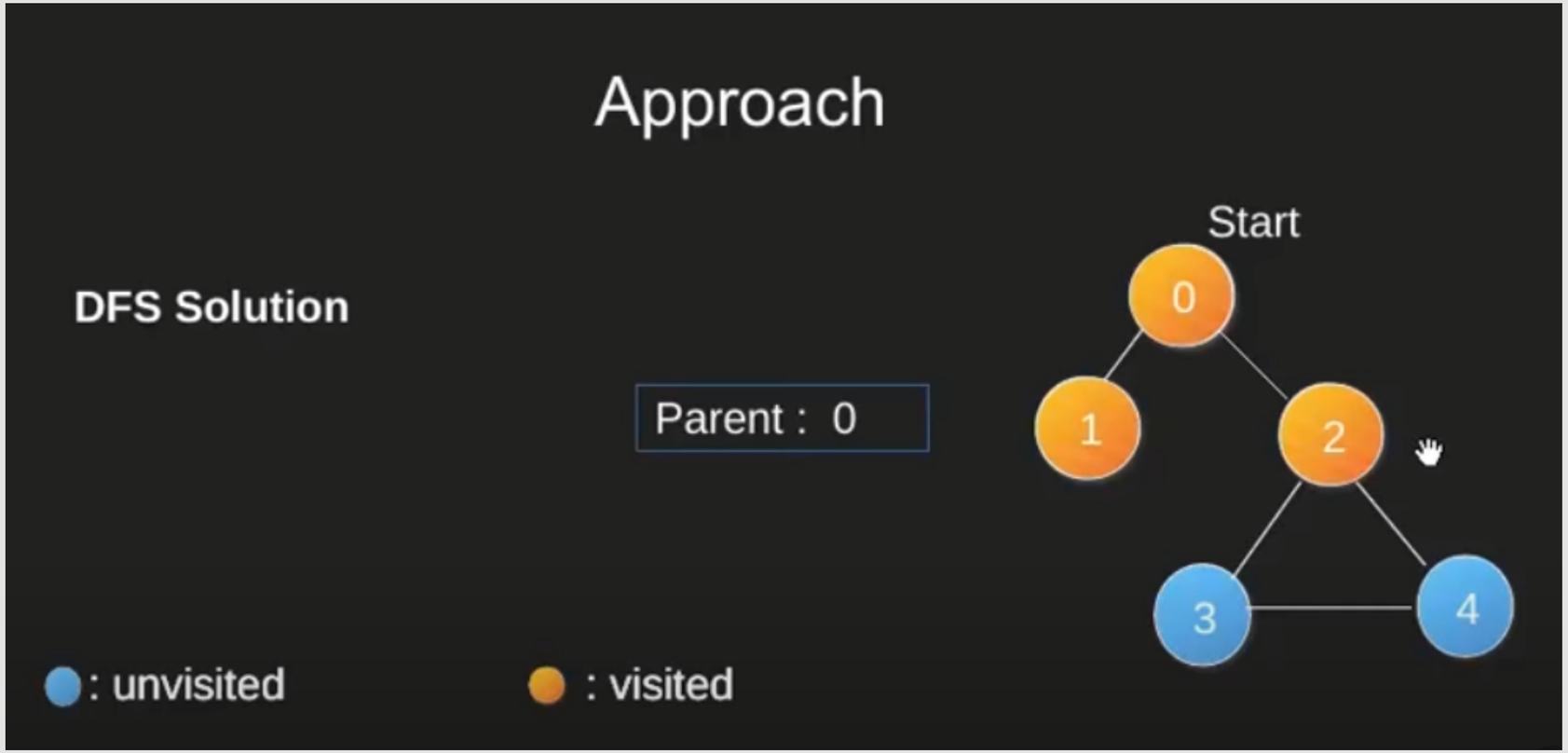
- **DFS**


Approach
DFS Solution
Parent : -1
Start
0
1
2
3
4
: unvisited
: visited

- DFS


Approach
DFS Solution
Parent : 0
Start
0
1
2
3
4
: unvisited
: visited

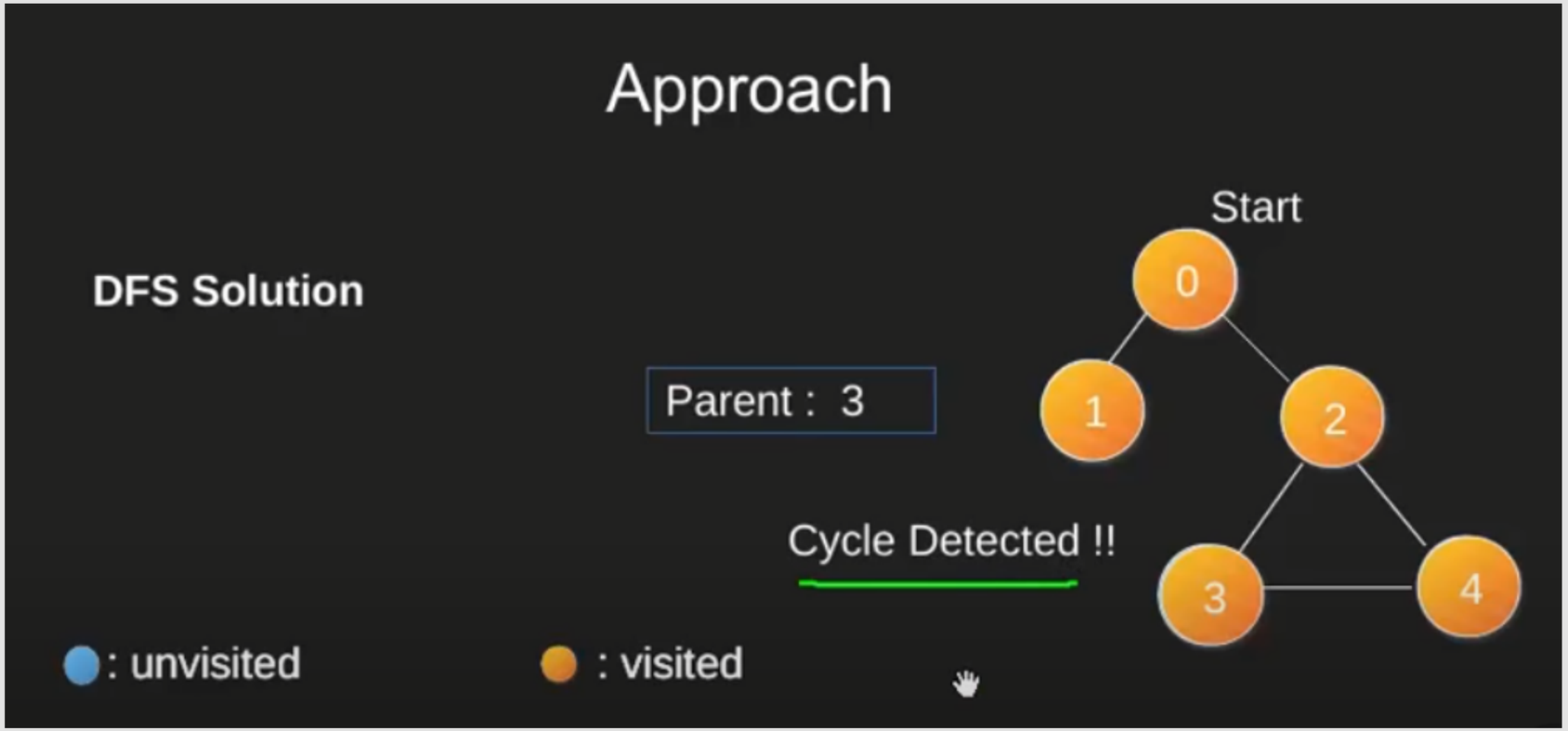
- **DFS**


Approach
DFS Solution
Parent : 0
Start
0
1
2
3
4
: unvisited
: visited

- **DFS**


Approach
DFS Solution
Parent : 2
Start
0
1
2
3
4
: unvisited
: visited

- DFS


Approach
DFS Solution
Parent : 3
Cycle Detected !!
Start
0
1
2
3
4
: unvisited
: visited

• DFS

```python
from collections import defaultdict

class Graph:
    def __init__(self, vertices):
        self.V = vertices  # No. of vertices
        self.graph = defaultdict(list)

    def addEdge(self, v, w):
        self.graph[v].append(w)
        self.graph[w].append(v)

    def isCyclicUtil(self, v, visited, parent):
        visited[v] = True
        for i in self.graph[v]:
            if visited[i] == False:
                if(self.isCyclicUtil(i, visited, v)):
                    return True
            elif parent != i:
                return True

        return False

    def isCyclic(self):
        visited = [False]*(self.V)

        for i in range(self.V):
            if visited[i] == False:
                if(self.isCyclicUtil
                   (i, visited, -1)) == True:
                    return True

        return False
```

- **Complexidade de tempo:**

O(V+E)

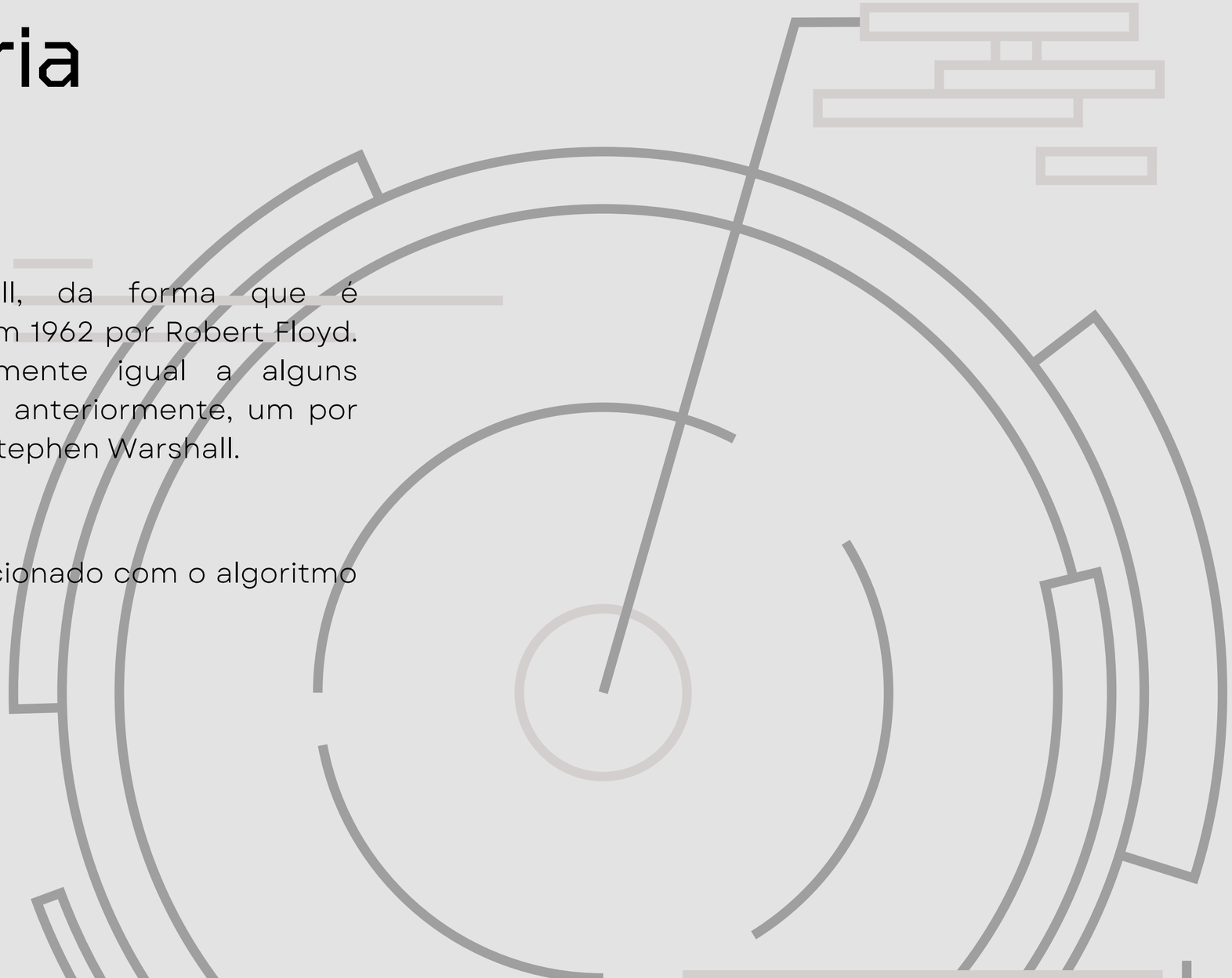
algoritmos em grafos

Algoritmo de Floyd-Warshall

História

O algoritmo de Floyd-Warshall, da forma que é reconhecido hoje, foi publicado em 1962 por Robert Floyd. Apesar disso, ele é essencialmente igual a alguns algoritmos que foram publicados anteriormente, um por Bernard Roy em 1959 e outro por Stephen Warshall.

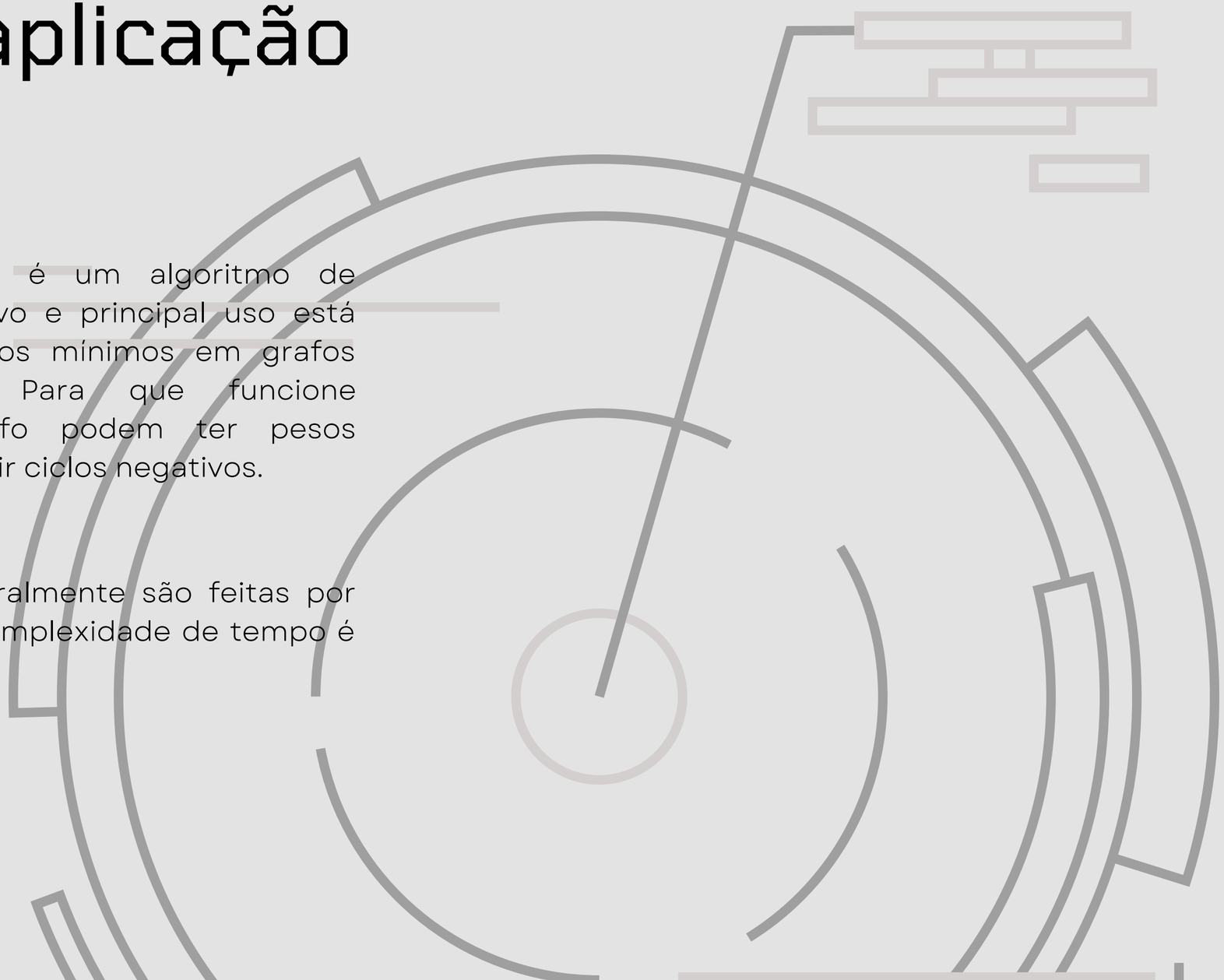
Além disso, ele também está relacionado com o algoritmo de Kleene, publicado em 1956.



Definição e aplicação

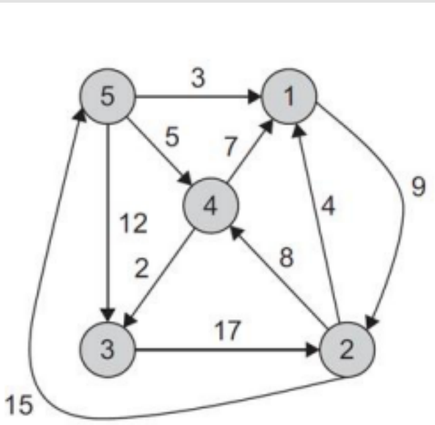
O algoritmo de Floyd-Warshall é um algoritmo de aplicação em grafos cujo objetivo e principal uso está relacionado à descobrir caminhos mínimos em grafos ponderados e direcionados. Para que funcione corretamente, arestas do grafo podem ter pesos negativos, porém não devem existir ciclos negativos.

Sua representação e retorno geralmente são feitas por matrizes de adjacências, e sua complexidade de tempo é da ordem de O(n^3).



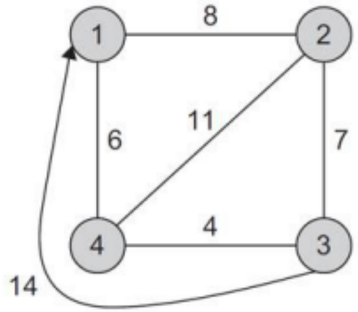
Matriz de Adjacências

Matriz de adjacências é uma das formas computacionais de representar os grafos, isto é, como o computador identifica uma estrutura de grafo. Existem alguns tipos de matrizes, as utilizadas pelo algoritmo de Floyd-Warshall é aquela que representa um grafo direcionado e ponderado.



	1	2	3	4	5
1	0	9	0	0	0
2	4	0	0	8	15
3	0	17	0	0	0
4	7	0	2	0	0
5	3	0	12	5	0

(a)



	1	2	3	4
1	0	8	14	6
2	8	0	7	11
3	14	7	0	4
4	6	11	4	0

(b)

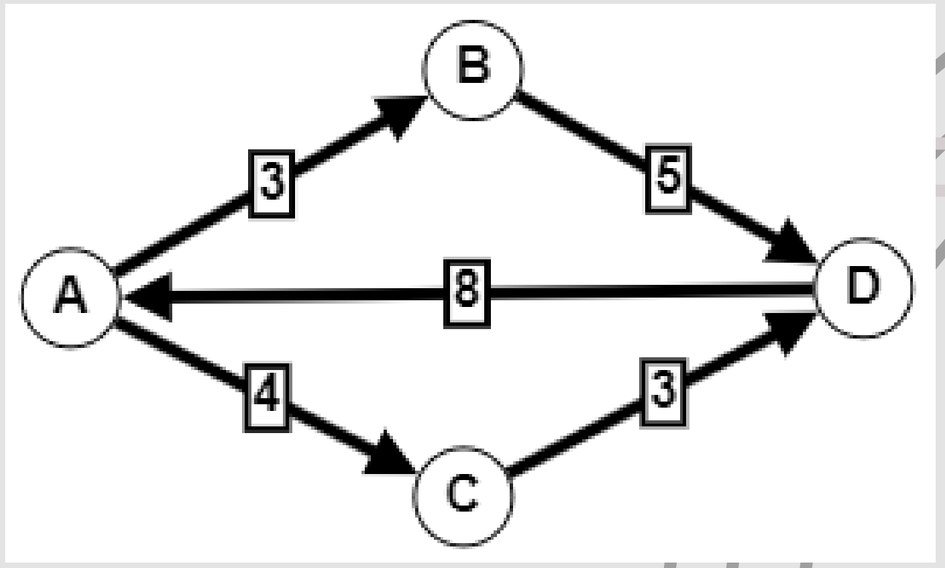
Pseudocódigo

```
ROTINA fw(Inteiro[1..n,1..n] grafo)
  # Inicialização
  VAR Inteiro[1..n,1..n] dist := grafo
  VAR Inteiro[1..n,1..n] pred
  PARA i DE 1 A n
    PARA j DE 1 A n
      SE dist[i,j] < Infinito ENTÃO
        pred[i,j] := i
  # Laço principal do algoritmo
  PARA k DE 1 A n
    PARA i DE 1 A n
      PARA j DE 1 A n
        SE dist[i,j] > dist[i,k] + dist[k,j] ENTÃO
          dist[i,j] = dist[i,k] + dist[k,j]
          pred[i,j] = pred[k,j]
  RETORNE dist
```

Código Fonte

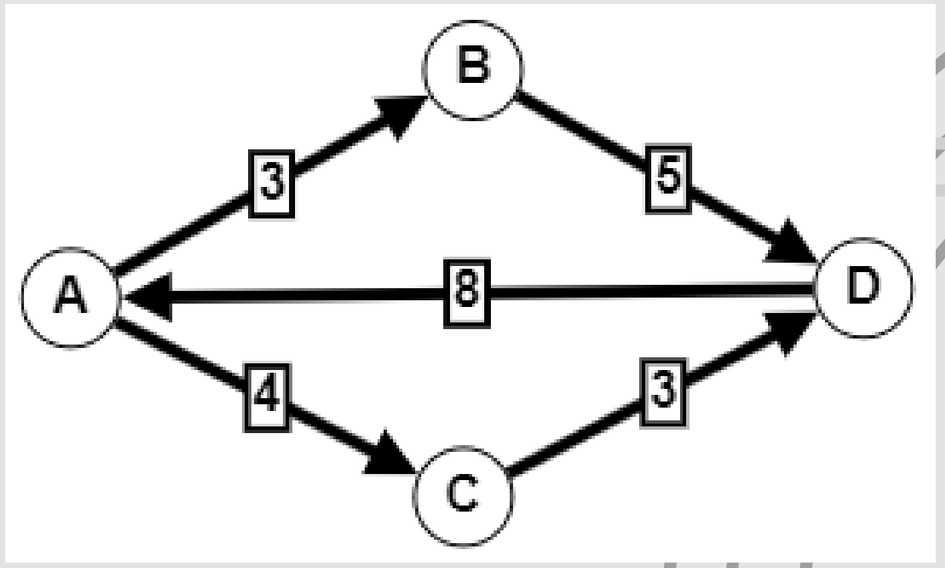
```
for (k = 0; k < V; k++) {
    for (i = 0; i< V; i++) {
        for (j = 0; j < V; j++) {
            if (dist[i][k] + dist[k][j] < dist[i][j]) {
                dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }
}
```

Exemplo da aplicação



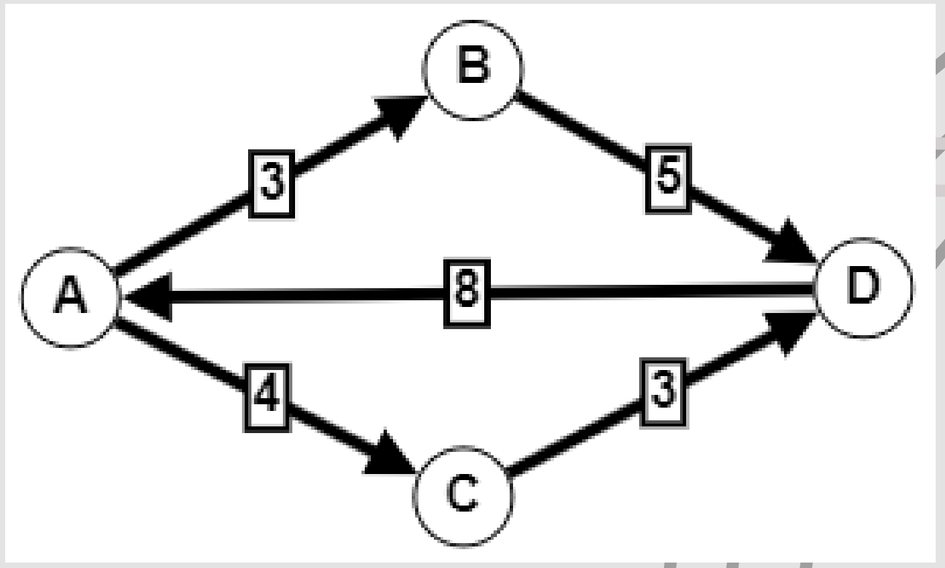
	A	*B*	*C*	*D*
A	—	3	4	INF
B	INF	—	INF	5
C	INF	INF	—	3
D	8	INF	INF	—

Exemplo da aplicação



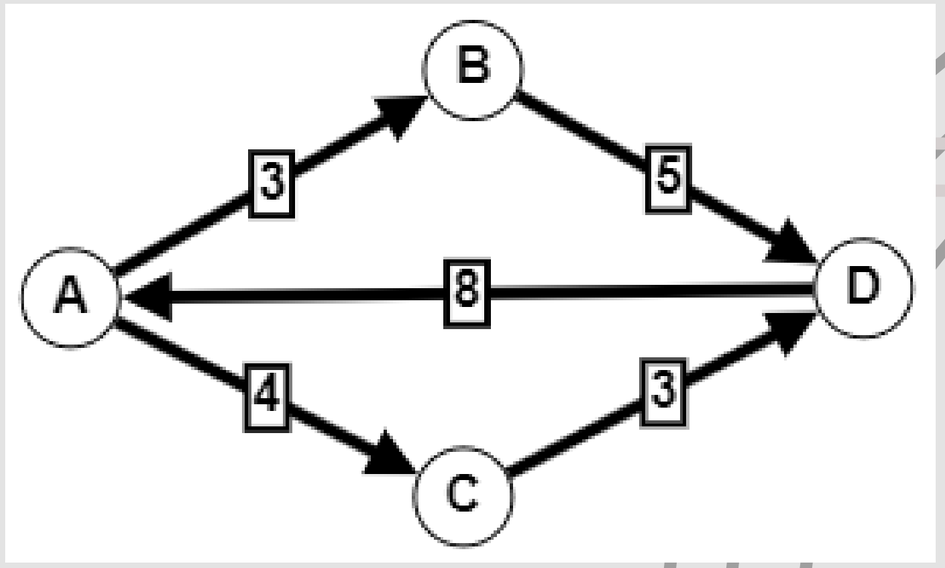
	A	B	C	D
A	—	3	4	INF
B	INF	—	INF	5
C	INF	INF	—	3
D	8	11	12	—

Exemplo da aplicação



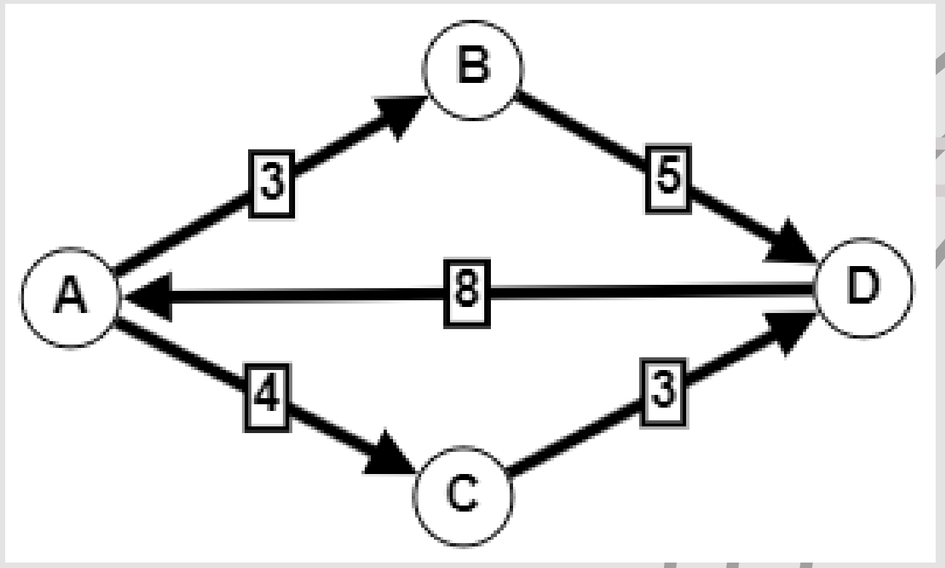
	A	*B*	*C*	*D*
A	—	3	4	8
B	INF	—	INF	5
C	INF	INF	—	3
D	8	11	12	—

Exemplo da aplicação



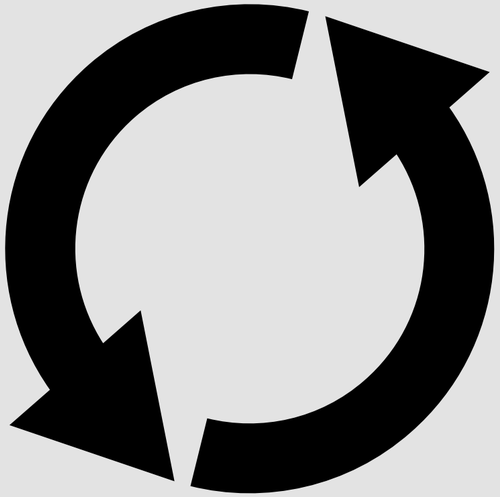
	A	*B*	*C*	*D*
A	—	3	4	7
B	INF	—	INF	5
C	INF	INF	—	3
D	8	11	12	—

Exemplo da aplicação



	A	B	C	D
A	—	3	4	7
B	13	—	17	5
C	11	14	—	3
D	8	11	12	—

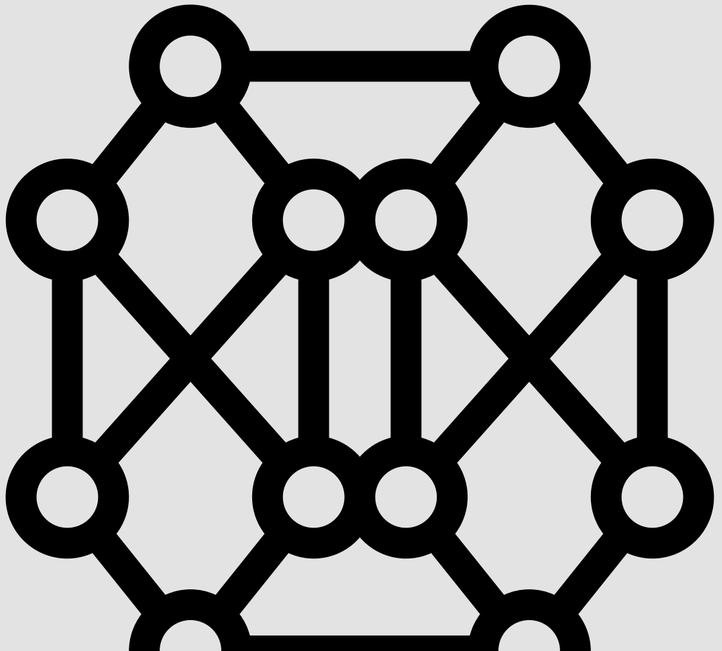
Detecção de Ciclos em Grafos Dirigidos



- Arestas positivas: DFS (Depth-First-Search)

Extra:

- Arestas negativas: algoritmo de Bellman-Ford

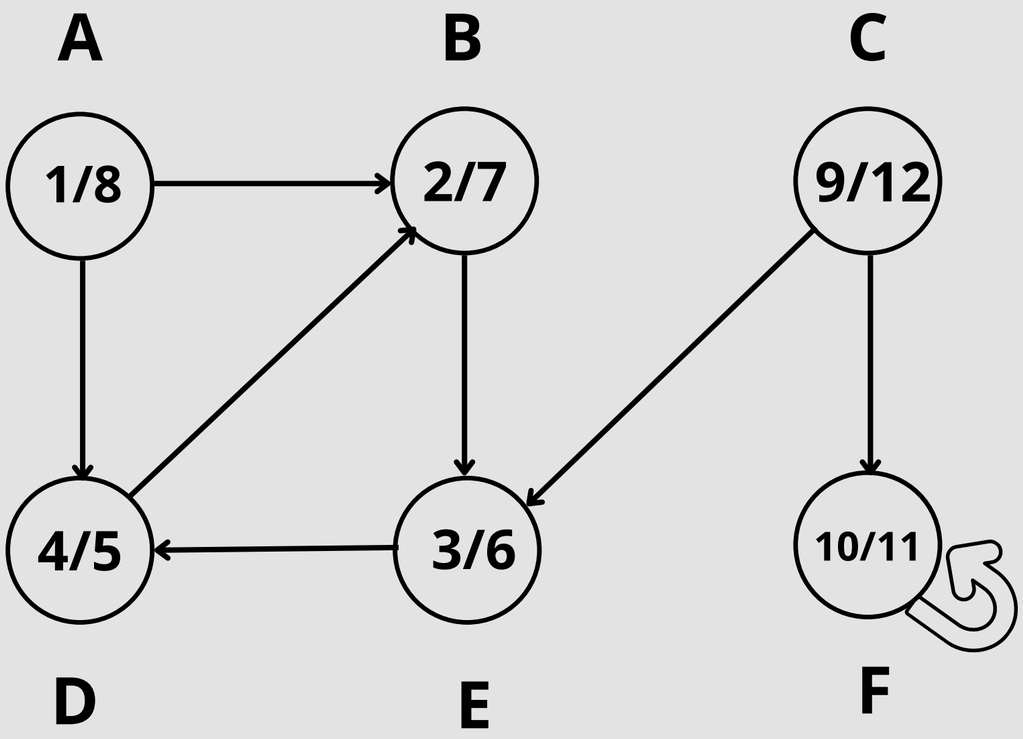


• DFS

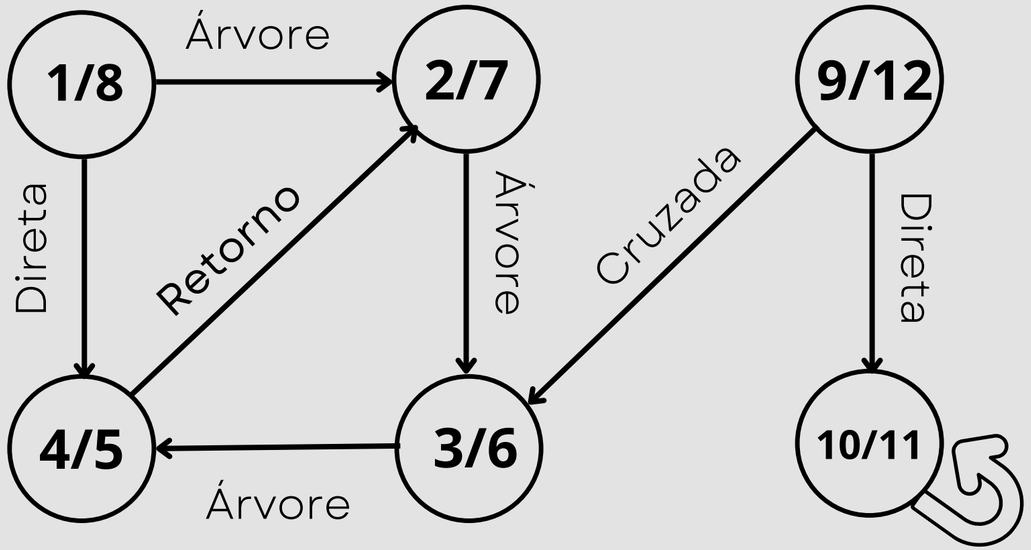
```
DFS(G)
1  for each vertex u ∈ G.V                          O(V)
2      u.color = WHITE
3      u.π = NIL
4  time = 0                                         O(1)
5  for each vertex u ∈ G.V                          O(V)
6      if u.color == WHITE
7          DFS-VISIT(G, u)

DFS-VISIT(G, u)
 1  time = time + 1                                  O(V)
 2  u.d = time                                       // white vertex u has just been discovered
 3  u.color = GRAY
 4  for each v ∈ G.Adj[u]                            // explore edge (u, v)
 5      if v.color == WHITE                          O(E)
 6          v.π = u
 7          DFS-VISIT(G, v)
 8  u.color = BLACK                                  // blacken u; it is finished
 9  time = time + 1
10  u.f = time
```

- **Complexidade de tempo: O(V+E)**

- Exemplo de detecção de Ciclos em uma DFS
    1. **Executar o DFS**



- Classificação de Arestas na prática:



na implementação do algoritmo, as arestas poderiam receber um atributo de tipo :

é uma aresta pra frente se u.d < v.d

é uma aresta cruzada se u.d > v.d

Teorema:

Se G tem um ciclo ⇔ G tem uma aresta para trás – ou seja, pode-se usar DFS para detectar ciclos!