LISTA 3

1. $$\sum_{v \in V(G)} d(v) = 2|E(G)|$$

→ $V(G)$ é o conjunto de vértices do grafo G
→ $E(G)$ é o conjunto de arestas do grafo G
→ $d(v)$ é o grau do vértice v. ou seja, o número de arestas incidentes a v.

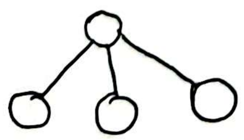
• Cada aresta em $E(G)$ conecta exatamente dois vértices, assim ao contar o grau dos vértices, cada aresta é contada duas vezes, uma para cada extremidade. Portanto, a soma dos graus dos vértices é igual ao dobro do número de arestas, como mostrado.

2. Caminho: conjunto de arestas que não passa duas vezes pelo mesmo vértice.

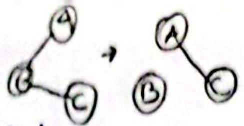
Por ser árvore, sabemos que é conexo, mas é um grafo acíclico. Assim podemos percorrer as arestas sem repetir a passagem pelos vértices. $|E| = |V| - 1$

3. $\Delta(T) \geqslant K$

Com pelo menos K vértices, sendo $K < V$



$\Delta(T) \geqslant 3$, sendo $K < V$

4. complementar (G) → 

$V' = G.V$ # recebe os vértices do grafo G

$E' = \cancel{\text{NULO}}\ \emptyset$ # inicializa o conjunto de ~~vértices~~ arestas como vazio

para cada v (vértice qualquer) em V(G) (conjunto de vértices)

para cada u (outro vértice) em V(G)

se v ≠ u and !existe_aresta(u, v) # assumindo que existe_aresta determina, desconsiderando a representação, se a aresta uv estão ou não no grafo

E'.adicionar(u, v)

$G' = (V', E')$

return G'

Para a matriz: → Na matriz o que é 0 vira 1 e o que é 1 vira 0, só temos dois laços

pertencimento ~~Calcular~~ matriz adjacência ∈ O(1)

os dois "para" ∈ $O(V^2)$

↓

$O(|V|^2)$

$O(n^2)$

Para a lista: $O(|V|^3)$ → Esse limite está muito frouxo?

▷ For v in [1 ... n]
for u in [1 ... n]
if (v ≠ u)
troque ?

$O(d(v) * |V|^2)$

$O(n^2 m)$

5. transposição (G) — O que é?

Complexidade
Matriz: $O(\|G^2\|)$
Lista: $O(d(v) * \|V\|)$
$O(\|V\|^2)$ $O(nm)$

$V' = G.V$

$E' = \emptyset$

para cada v em V(G)

para cada u em G.N(v) # vértices adjacentes a v

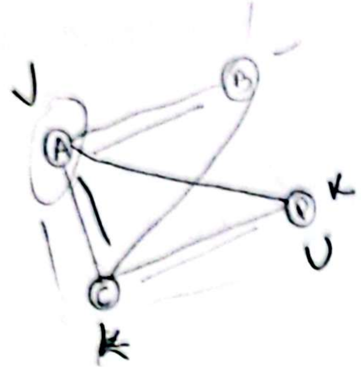
~~[illegible]~~

E'.adicionar(u, v)

$G' = (V', E')$

return G'

6.



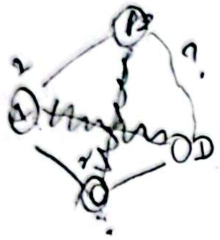
For $v \in V(G)$ — vizinhos → vizinhança

for $k \in N(v)$?

for $u \in N(k)$

If k vizinho de u e v

$O(\Delta m)$



7. ~~Vou fazer~~ Na lista de adjacência podemos adicionar os pesos do grafo ponderado aos vértices como tuplas.

Na matriz de adjacência podemos trocar as entradas 1 pelos valores dos pesos.

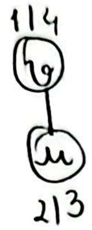
→Tudo isso assumindo que os pesos são valores positivos e maiores que 0.

Digitalizado com CamScanner

8. i) Se $i[u] < i[v] < f[v] < f[u]$, isso significa que v foi descoberto durante a exploração de u e completamente processado antes que a exploração de u fosse concluída.

Isso descreve exatamente o comportamento das arestas de árvore ou de avanço, pois v é descoberto depois de u e finalizado antes que a DFS conclua o processamento de u.

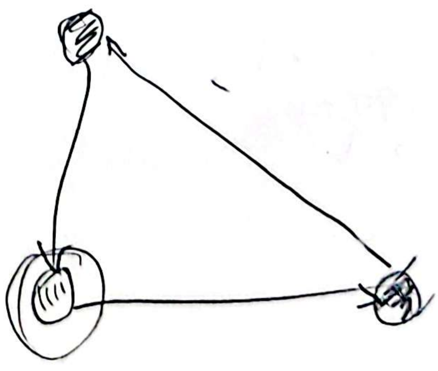
ii) Se $i[v] \leq i[u] < f[u] \leq f[v]$



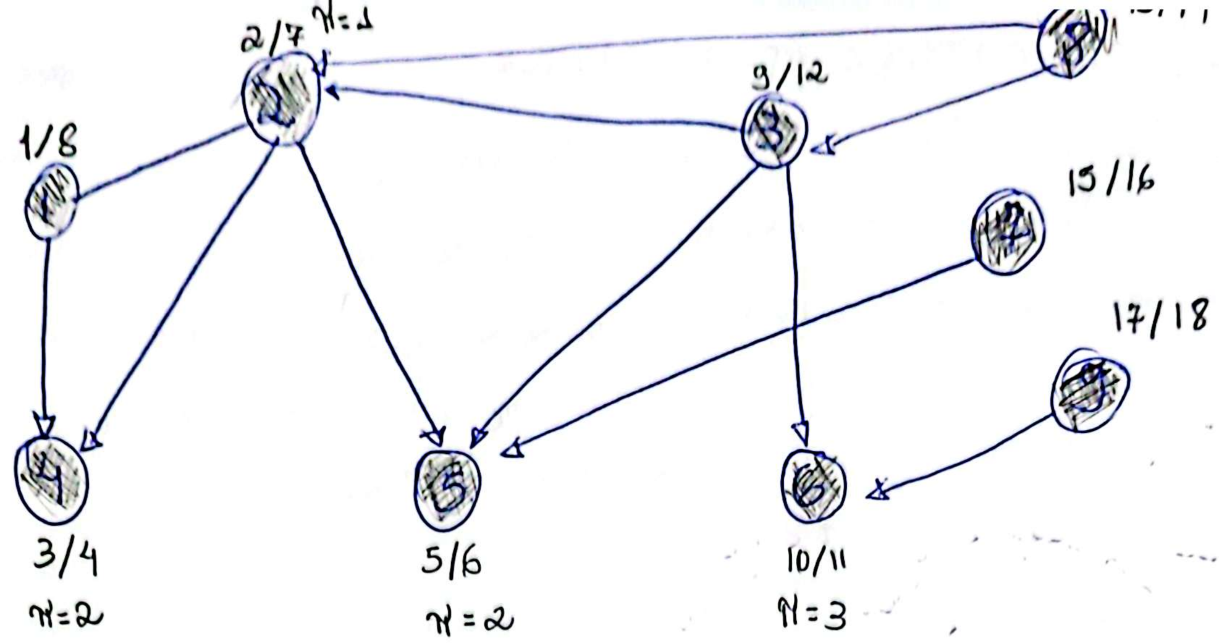
O vértice v é descoberto, finalizamos o u e retornamos para depois finalizar o v. Isso indica uma aresta de retorno.

iii) A condição $i[v] < f[v] < i[u] < f[u]$ caracteriza uma aresta de passagem na DFS. Ela indica que v foi finalizado antes que u fosse descoberto, confirmando que não há relação direta de ancestralidade ou descendência entre u e v na árvore de DFS e que a aresta uv é, portanto, uma aresta de passagem.

9. Durante a execução podemos ter arestas de avanço, de passagem e retorno.



10.



11. n= |G.V| # contagem do número de vértices
cria uma pilha vazia S com n elementos

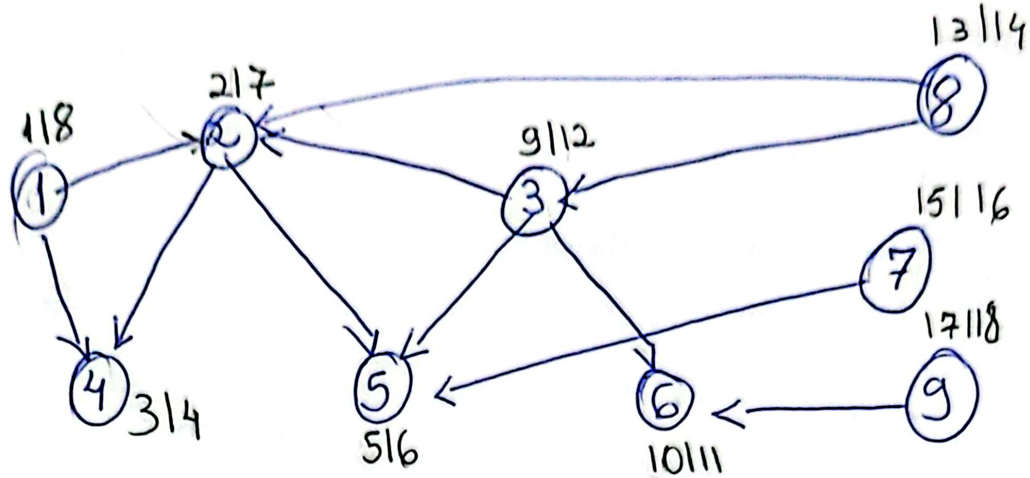
tempo = tempo + 1
i [u]
~~u.~~ d = tempo # vértice inicial
color [u] = cinza
S.PUSH (u, G. Adjacente [u] . head)
~~PUSH (S~~

enquanto ! S. vazio ()

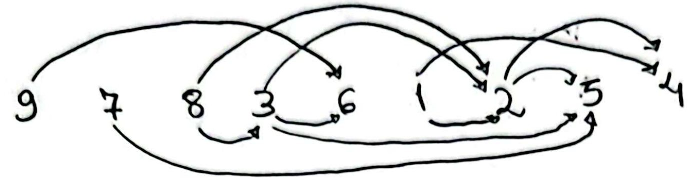
~~~~

u , adjx = POP (S)
~~for v em Adjacentes (u):~~
~~S. PUSH~~
enquanto d
~~~~

12.



?



13. Em um grafo orientado que contém ciclos NÃO é possível aplicar uma ordenação topológica. A ordenação topológica lineariza grafos orientados acíclicos de modo que para cada aresta (u, v) u apareça antes de v, mas esse processo considera que necessariamente não haja dependência cíclica.

DFS, por exemplo, faz uma ordenação topológica e detecta a presença de ciclos, já que isso impede que se complete a ordenação.

14. DFS ?

```
VERIFICAR_CICLO(G)
    para cada vértice u em G faça
        se u não foi visitado então
            se DFS-CICLO(G, u, -1) então
                return TRUE
    return FALSE

DFS-CICLO(G, u, pai)
    marque u como visitado
    para cada vizinho v de u faça
        se v não foi visitado então
            se DFS-CICLO(G, v, u) então
                return TRUE
        ~~if~~ else se v é diferente de pai então
            return TRUE
    return FALSE
```

Aplica o DFS normal, se tiver QUALQUER UMA aresta de retorno, há um ciclo no grafo

→ Se se deparar com uma aresta para um vértice CINZA a partir de um vértice CINZA, temos um ciclo

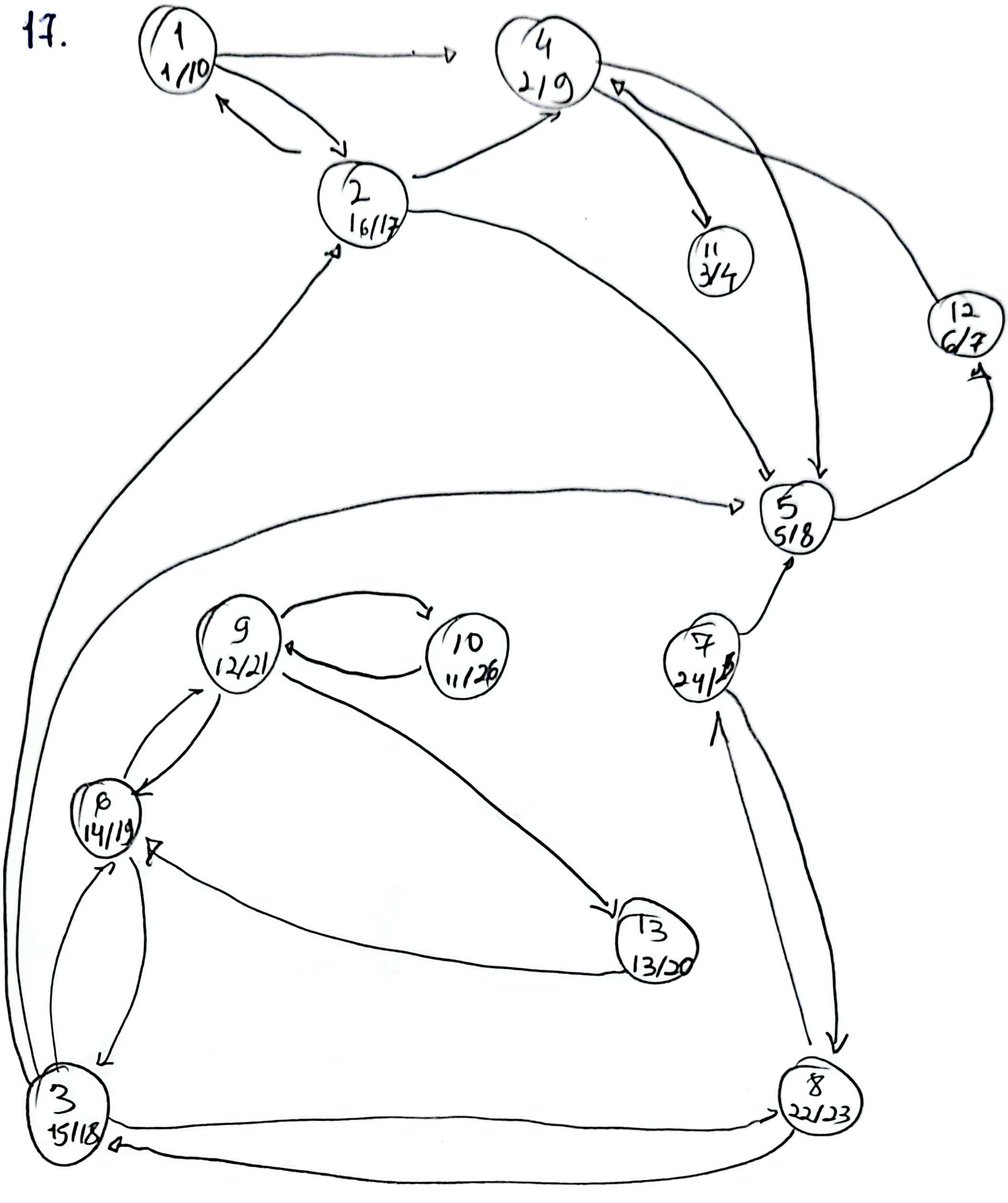
COMPLEXIDADE: O(V+E) visita todos os vértices e arestas

15. Não é possível executar o algoritmo em O(|V|), já que isso seria correspondente a executar percorrendo somente o número de vértices, no entanto, para detectar ciclos precisamos percorrer também as arestas do grafo.

16. - Adicionar um arco pode <u>diminuir</u> o número de componentes for-
temente conexas ao fundir componentes que antes estavam separados

- Caso a estrutura de conexões permaneça a mesma, não há altera-
ção de CFCs.

17.


1
1/10
4
2/9
2
16/17
11
3/4
12
6/7
5
5/8
9
12/21
10
11/26
7
24/25
8
14/19
13
13/20
3
15/18
8
22/23