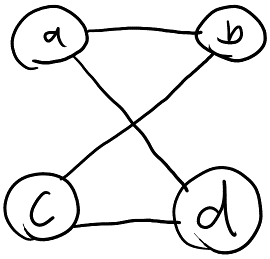
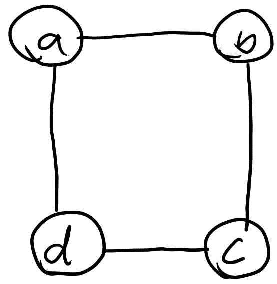
1.2.18 Planaridade

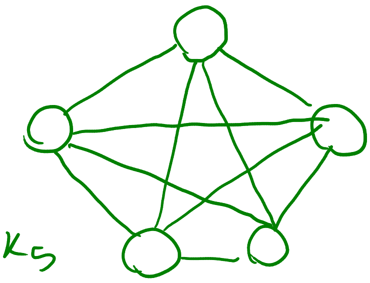


É planar?



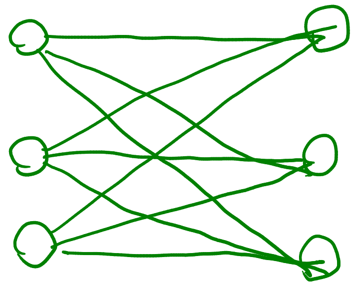
Sim!

K_5: grafo completo com 5 vértices



K_5

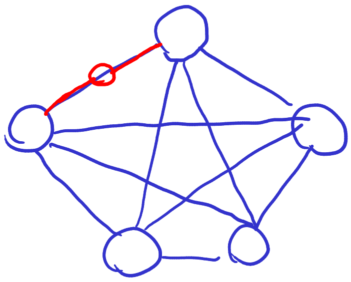
$K_{3,3}$



Grafo Homeomorfo:

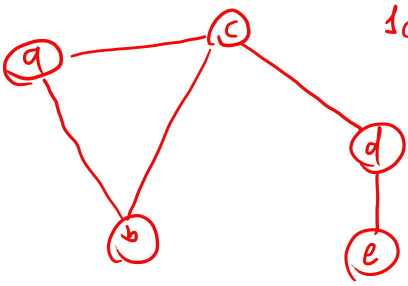
$G = K_5$

H:



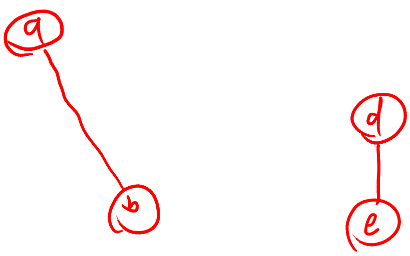
1.2.19 Vértices de Corte

1 componente conexa



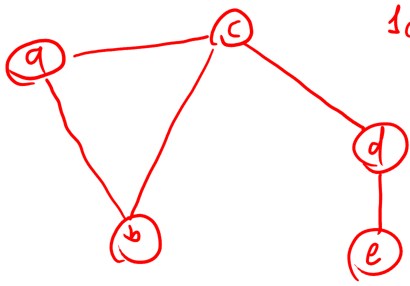
⇓ "c" é um vértice de corte

2 componentes conexas

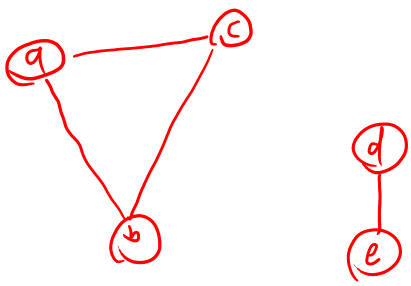


1.2.20 Ponte

1 componente conexa

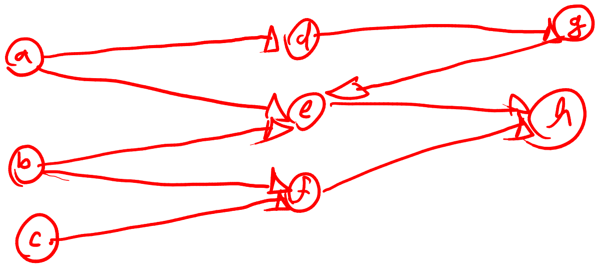


⇓ "{c, d}" é uma ponte



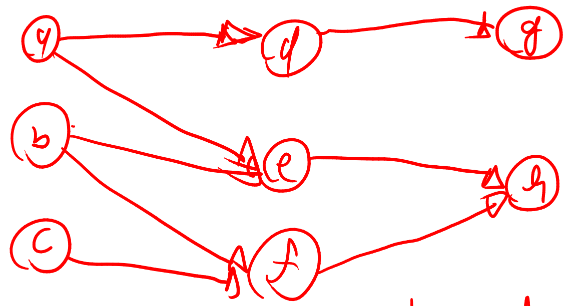
2 componentes

1.2.21 Base



$\{a,b,c\}$ é o conjunto base.

1.2.22 Anti-Base



$\{g,h\}$ é o conjunto anti-base.

Aquecimento para a Q2:

Q2.1) $L = \{(u, s_i, s_j)\}$, no qual o usuário "u" visita o site s_i antes de s_j.

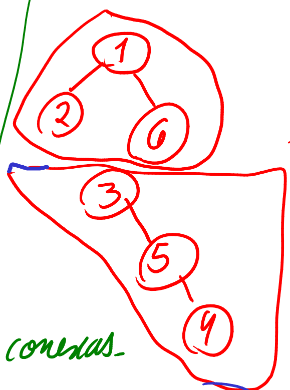
Algoritmo Q2.1

<u>Entrada</u>: um usuário "u", e um conjunto L

1. $V \leftarrow \{\}$
2. $A \leftarrow \{\}$
3. foreach $(p, s_i, s_j) \in L$ do
4.   if $p == u$ then
5. $V \leftarrow V \cup \{s_i, s_j\}$
6. $A \leftarrow A \cup \{(s_i, s_j)\}$
7. $G \leftarrow (V, A)$

Criando o grafo c/ os sites visitados pelo usuário "u".

8. $T \leftarrow$ Algoritmo 15 (G) // Comp. Fortemente conexas.
9. $S \leftarrow$ vetor com $|V|$ posições
10. foreach $v \in V$ do
11. $S_v \leftarrow \{v\}$
12. foreach $v \in V$ do
13.   if $T_v \neq NULL$ then
14. $X \leftarrow S_v \cup S_{T_v}$
15.   foreach $x \in X$ do
16. $S_x \leftarrow X$
17. $R \leftarrow \{\}$
18. foreach $v \in V$ do
19. $R \leftarrow R \cup \{S_v\}$
20. return R.



	T	
1	~	~~{1,2,6}~~
2	1	~~{2}1,6~~
3	~	{3}4,5
4	5	{4}5,3
5	3	{5}3,4
6	1	{6}1,2

$n = |V|$, $m = |E|$, $l = |L|$

$$t(n,m,l) \leq c_1 l + c_2 n + c_3 m + c_4 n + c_5 n \cdot (n^2 + n) + n \in O(l + n^3 + m)$$

Q2.2

Primeiramente, deve-se alterar o Algoritmo 19 das anotações da disciplina adicionando as linhas abaixo. O algoritmo alterado, deve ser utilizado nas chamadas do Algoritmo 18.

Entre as linhas 4 e 5:

if $T_u \neq \infty$ and $F_u = \infty$ then
  return false

No lugar da linha 6, utiliza-se

$R \leftarrow$ DFS-Visit-OT(G, u, C, T, A, F, tempo, O)
if R == false then
  return false

Depois da linha 9:
  return true

No Algoritmo 18, deve-se alterar o seguinte:

Na linha 8, alterar:

$r \leftarrow$ DFS-Visit-OT(G, u, C, T, A, F, tempo, O)
if r == false then
  return (false, ())

Na linha 9, alterar:
  return (true, O)

Depois das alterações, o seguinte algoritmo atende a questão:

Algoritmo Q2.2

Entrada: $R = \{(t_i, t_j)\}$ t_j depende da t_i.

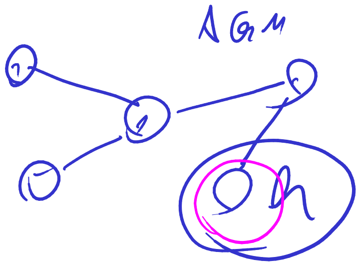
1. $V \leftarrow \{\}$
2. $A \leftarrow R$
3. foreach $(t_i, t_j) \in R$ do
4. $V \leftarrow V \cup \{t_i, t_j\}$
5. $G \leftarrow (V, A)$
6. $(r, O) \leftarrow$ Algoritmo18(G) com alterações acima descritas.
7. ...

$O(|V| + |R|)$

$\in O(|R|)$

```
if r == false then
    print("Há tarefa que não pode ser executada")
else
    foreach o ∈ O do
        print(o + ",")
```

Q2 3)



- AGM p/ cada cidade
- Se h tem 2 conexões na AGM
- Se não houver
  - gerar! (pode não ser possível

Algoritmo Q2 3

Entrada: o conj. H, e as funções h_i

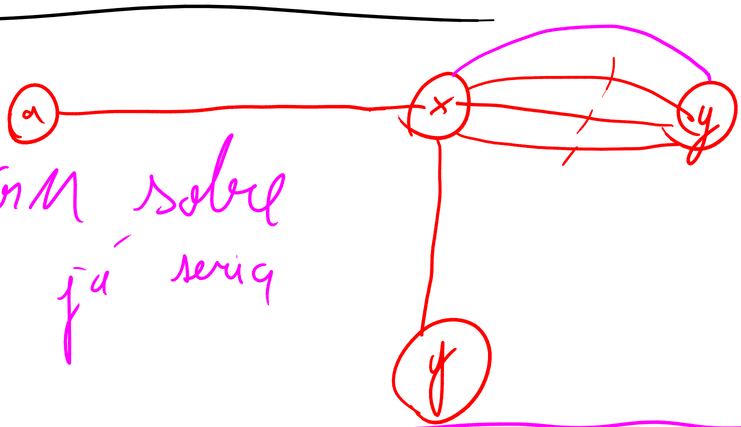
```
total ← 0
foreach G_i ∈ H do
    A ← Algoritmo 21 // Kruskal
    o ← h_i(G_i)
    qtd ← 0
    foreach {u,v} ∈ A do
        if u = o OR v = o then
            qtd ← qtd + 1
    if qtd < 2 then
        e ← argmin_{{a,b} ∈ E_i − A} { w_i({a,b}) | a = o ou b = o }
        if e = null then
            print("Para a cidade " + G_i + " não é possível"
                  + "conectar o hospital a 2 linhas")
        A ← A ∪ {e}
    ⋮
```

$$|H|.(|E^*| \log_2 |E^*| + |E^*| + |E^*| + |E^*|) \in O(|H|.(|E^*| \log_2 |E^*|))$$

assumindo E^* é o maior conj. de arestas.

```
foreach {a,b} ∈ A do
    total ← total + wi({a,b})
return total
```

Q2 4)



Uso da AGM sobre o grafo G já seria suficiente