

# Graphs' decompositions and resolutions of combinatorial problems

Stéphane Secouard  
Supervised by : Florent Madeleine

Caen University - Computer science

25 october 2016



- 1 Objectives
- 2 Tree decomposition of a graph
  - Tree decomposition
  - Treewidth
  - Nice tree
  - Example
- 3 Application :  $k$ -color
  - The problem
  - Illustration
- 4 Bibliography

## 1 Objectives

## 2 Tree decomposition of a graph

- Tree decomposition
- Treewidth
- Nice tree
- Example

## 3 Application : $k$ -color

- The problem
- Illustration

## 4 Bibliography

## Objectives (of my project)

- *The first goal of this project is, starting from a graph whose tree representation is known, to solve corresponding combinatorial problems.  
The  $k$ -coloring problem, the max clique problem or the Hamilton path problem can be explored.*
- *The second purpose is to implement a graph decomposition calculator.*
- *Finally, it can be considered extensions by working on the efficiency of implementations on large-size structures, or improving the shape of displayed results.*

- 1 Objectives
- 2 **Tree decomposition of a graph**
  - Tree decomposition
  - Treewidth
  - Nice tree
  - Example
- 3 Application : k-color
  - The problem
  - Illustration
- 4 Bibliography

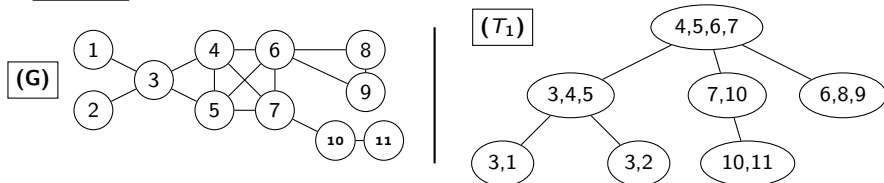
# Tree decomposition of a graph

## Definition (Tree decomposition of a graph)

A tree  $T$  is a tree decomposition of a graph  $G$  where its nodes are arranged satisfying the following properties :

- 1 If  $u$  and  $v$  are neighbors in  $G$ , then there is a bag of  $T$  containing both of them (a bag is a node of the tree).
- 2 For every vertex  $v$  of  $G$ , the bags of  $T$  containing  $v$  form a connected subtree

example :



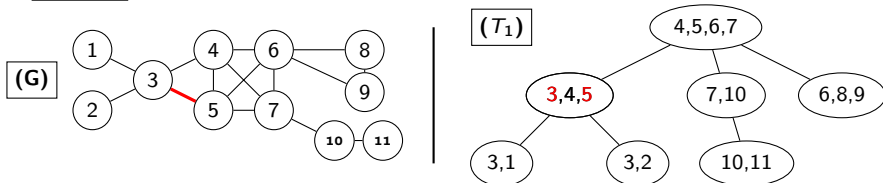
# Tree decomposition of a graph

## Definition (Tree decomposition of a graph)

A tree  $T$  is a tree decomposition of a graph  $G$  where its nodes are arranged satisfying the following properties :

- 1 If  $u$  and  $v$  are neighbors in  $G$ , then there is a bag of  $T$  containing both of them (a bag is a node of the tree).
- 2 For every vertex  $v$  of  $G$ , the bags of  $T$  containing  $v$  form a connected subtree

example :



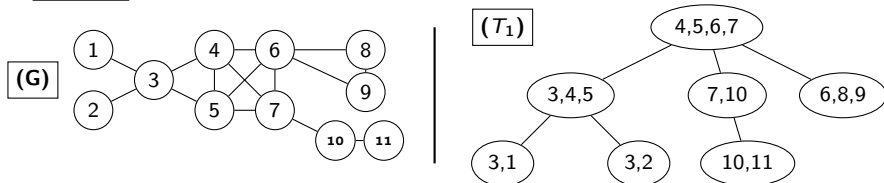
# Tree decomposition of a graph

## Definition (Tree decomposition of a graph)

A tree  $T$  is a tree decomposition of a graph  $G$  where its nodes are arranged satisfying the following properties :

- 1 If  $u$  and  $v$  are neighbors in  $G$ , then there is a bag of  $T$  containing both of them (a bag is a node of the tree).
- 2 For every vertex  $v$  of  $G$ , the bags of  $T$  containing  $v$  form a connected subtree

example :





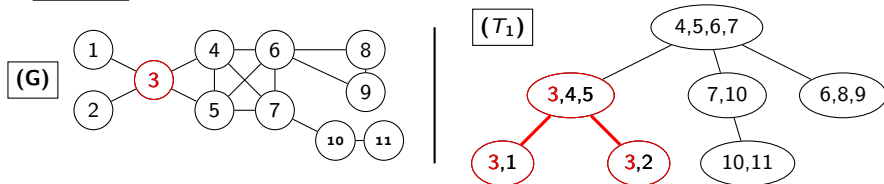
# Tree decomposition of a graph

## Definition (Tree decomposition of a graph)

A tree  $T$  is a tree decomposition of a graph  $G$  where its nodes are arranged satisfying the following properties :

- 1 If  $u$  and  $v$  are neighbors in  $G$ , then there is a bag of  $T$  containing both of them (a bag is a node of the tree).
- 2 For every vertex  $v$  of  $G$ , the bags of  $T$  containing  $v$  form a connected subtree

example :



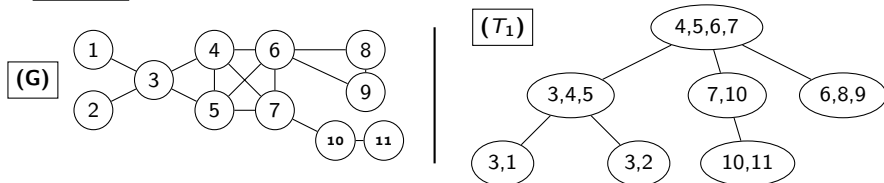
# Tree decomposition of a graph

## Definition (Tree decomposition of a graph)

A tree  $T$  is a tree decomposition of a graph  $G$  where its nodes are arranged satisfying the following properties :

- 1 If  $u$  and  $v$  are neighbors in  $G$ , then there is a bag of  $T$  containing both of them (a bag is a node of the tree).
- 2 For every vertex  $v$  of  $G$ , the bags of  $T$  containing  $v$  form a connected subtree

example :



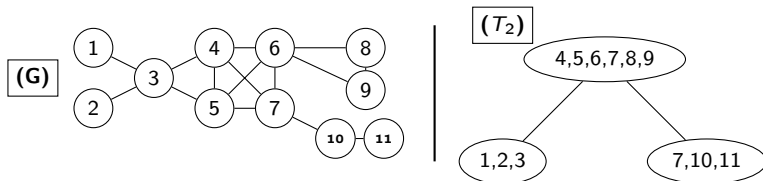
# Tree decomposition of a graph

## Definition (Tree decomposition of a graph)

A tree  $T$  is a tree decomposition of a graph  $G$  where its nodes are arranged satisfying the following properties :

- 1 If  $u$  and  $v$  are neighbors in  $G$ , then there is a bag of  $T$  containing both of them (a bag is a node of the tree).
- 2 For every vertex  $v$  of  $G$ , the bags of  $T$  containing  $v$  form a connected subtree

example :



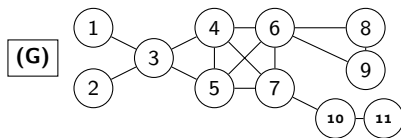
# Tree decomposition of a graph

## Definition (Tree decomposition of a graph)

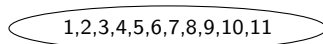
A tree  $T$  is a tree decomposition of a graph  $G$  where its nodes are arranged satisfying the following properties :

- 1 If  $u$  and  $v$  are neighbors in  $G$ , then there is a bag of  $T$  containing both of them (a bag is a node of the tree).
- 2 For every vertex  $v$  of  $G$ , the bags of  $T$  containing  $v$  form a connected subtree

example :



( $T_3$ )



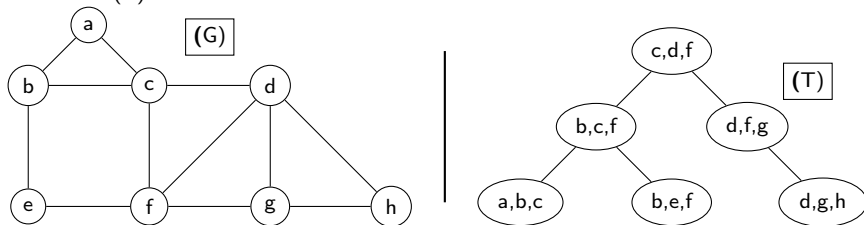
## Definition (treewidth)

- The *width* of a decomposition is (largest bag size - 1).
- The *treewidth* of a graph is the lowest width of all decompositions of this graph.

## Definition (treewidth)

- The *width* of a decomposition is (largest bag size - 1).
- The *treewidth* of a graph is the lowest width of all decompositions of this graph.

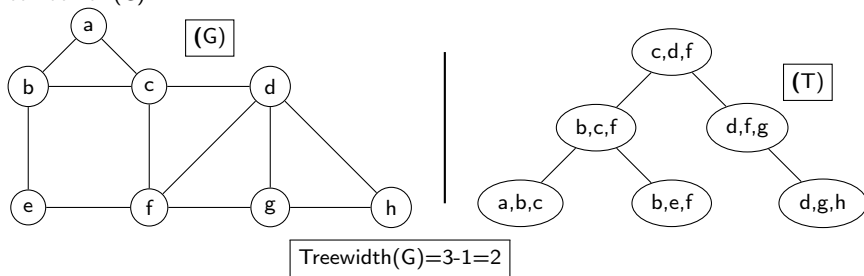
**Example :** Assume (T) is one of the representations of (G) which are giving the treewidth of (G) :



## Definition (treewidth)

- The *width* of a decomposition is (largest bag size - 1).
- The *treewidth* of a graph is the lowest width of all decompositions of this graph.

**Example :** Assume (T) is one of the representations of (G) which are giving the treewidth of (G) :



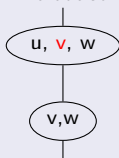
## Definition (nice tree)

A tree decomposition is *nice* if every node is one of the following 4 types :

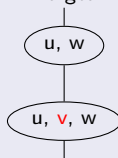
Leaf



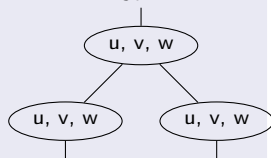
Introduce



Forget



Join



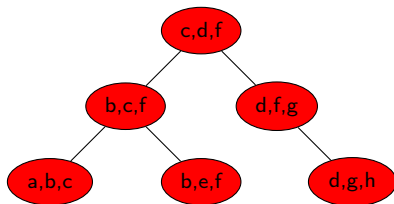
## Remark

- A tree decomposition can be turned into a nice tree decomposition
- A nice tree can be used to simplify a proof, or to find an easy program to solve a problem (as we will see later on).



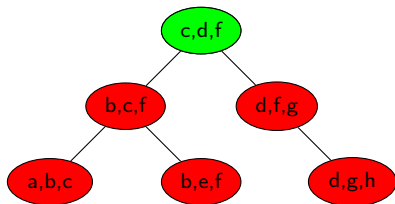
## Example

This is a tree decomposition of the graph seen previously. How can we get a nice tree from it?



## Example

This is a tree decomposition of the graph seen previously. How can we get a nice tree from it?

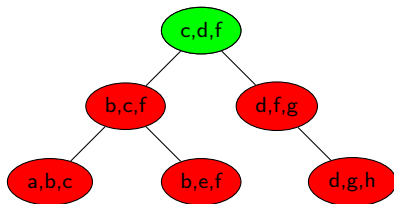


## Example

This is a tree decomposition of the graph seen previously. How can we get a nice tree from it?



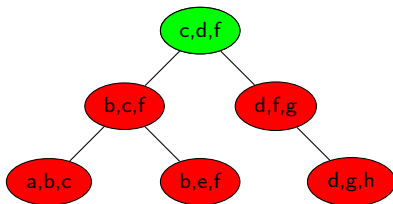
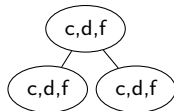
JOIN



## Example

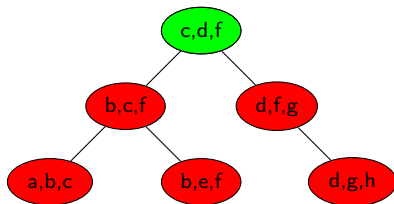
This is a tree decomposition of the graph seen previously. How can we get a nice tree from it?

JOIN



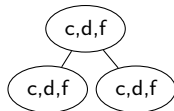
## Example

This is a tree decomposition of the graph seen previously. How can we get a nice tree from it?



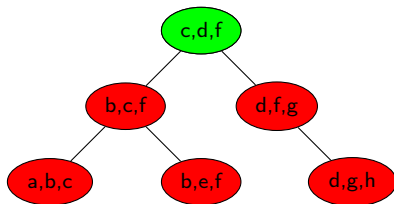
JOIN

FORGET



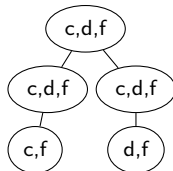
## Example

This is a tree decomposition of the graph seen previously. How can we get a nice tree from it?



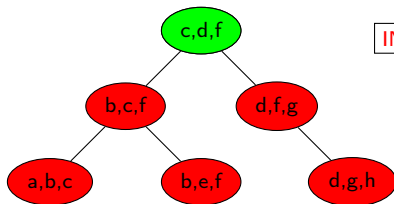
JOIN

FORGET



## Example

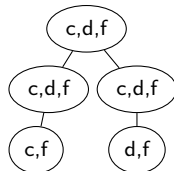
This is a tree decomposition of the graph seen previously. How can we get a nice tree from it?



JOIN

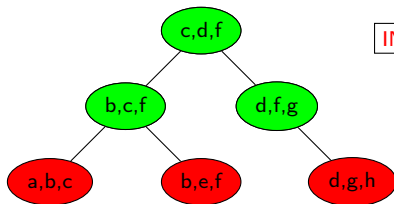
FORGET

INTRODUCE



## Example

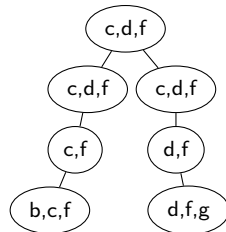
This is a tree decomposition of the graph seen previously. How can we get a nice tree from it?



JOIN

FORGET

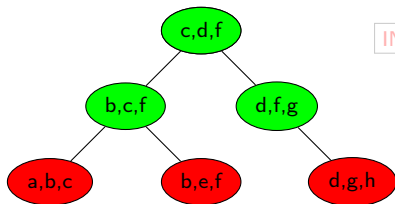
INTRODUCE





## Example

This is a tree decomposition of the graph seen previously. How can we get a nice tree from it?

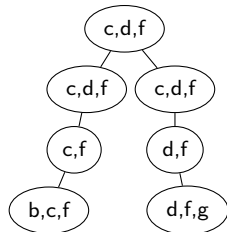


JOIN

FORGET

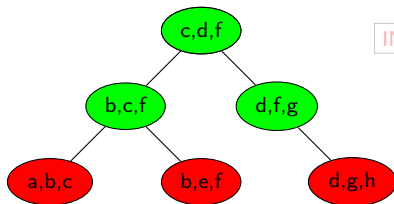
INTRODUCE

JOIN



## Example

This is a tree decomposition of the graph seen previously. How can we get a nice tree from it?

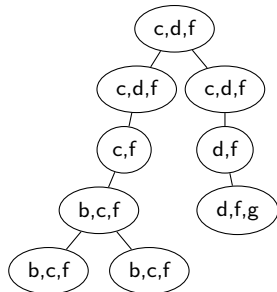


JOIN

FORGET

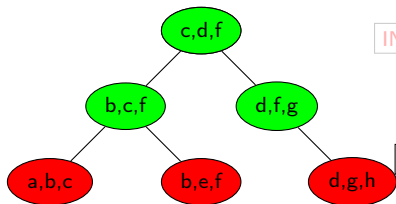
INTRODUCE

JOIN



## Example

This is a tree decomposition of the graph seen previously. How can we get a nice tree from it?



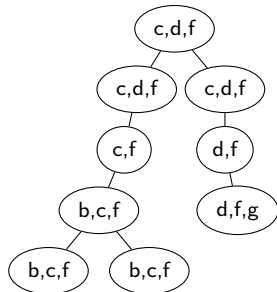
JOIN

FORGET

INTRODUCE

JOIN

FORGET



## Example

This is a tree decomposition of the graph seen previously. How can we get a nice tree from it?

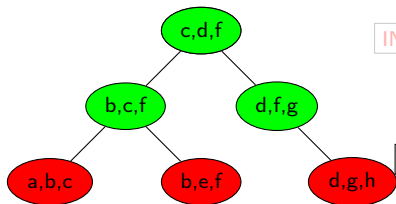
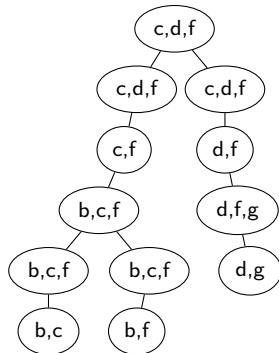


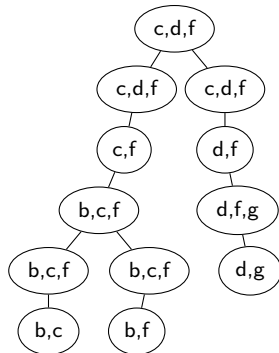
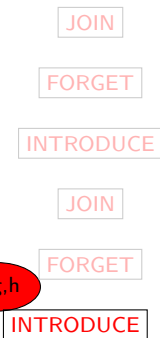
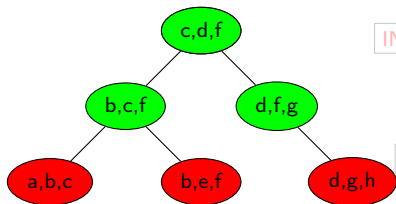
Diagram illustrating the operations used to transform the tree decomposition:

- JOIN
- FORGET
- INTRODUCE
- JOIN
- FORGET



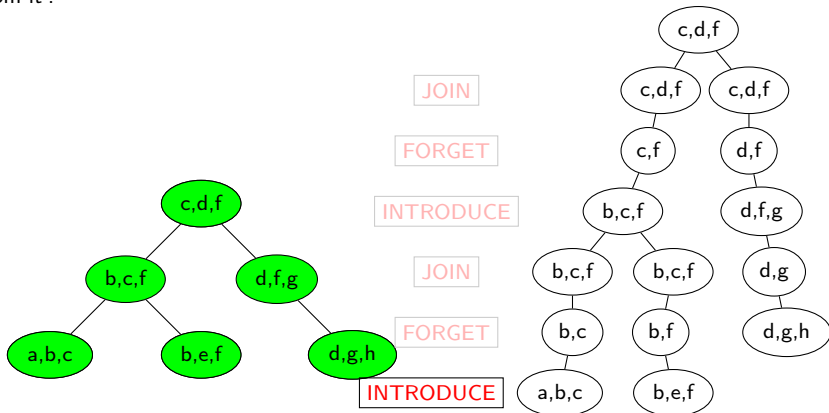
## Example

This is a tree decomposition of the graph seen previously. How can we get a nice tree from it?



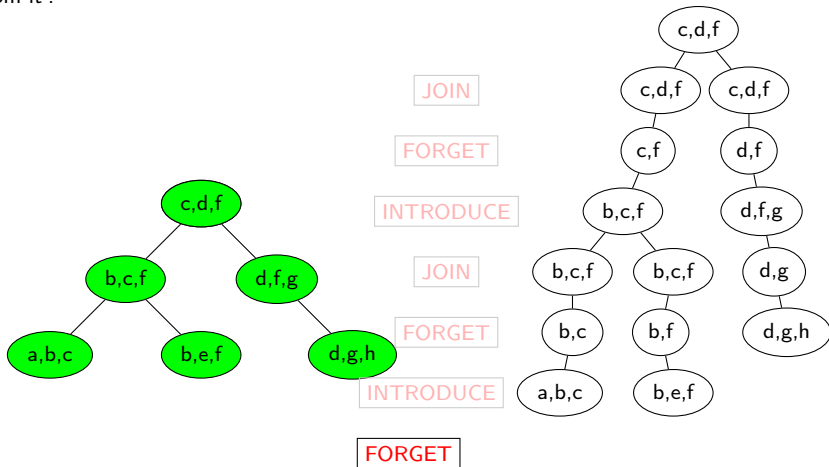
## Example

This is a tree decomposition of the graph seen previously. How can we get a nice tree from it?



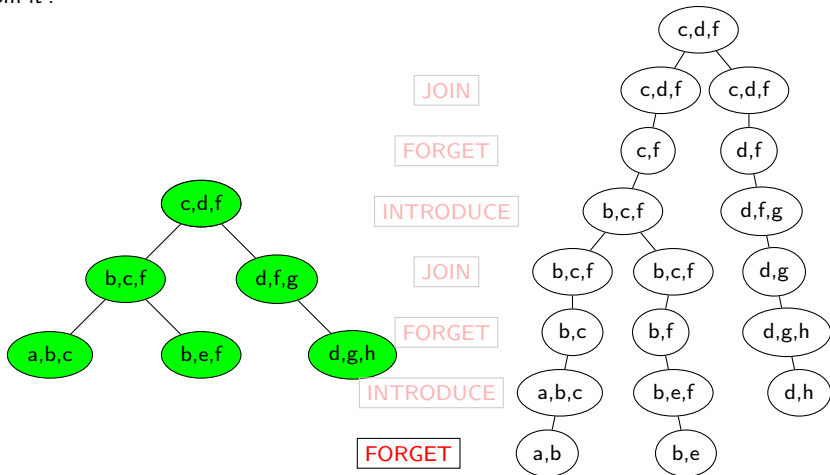
## Example

This is a tree decomposition of the graph seen previously. How can we get a nice tree from it?



## Example

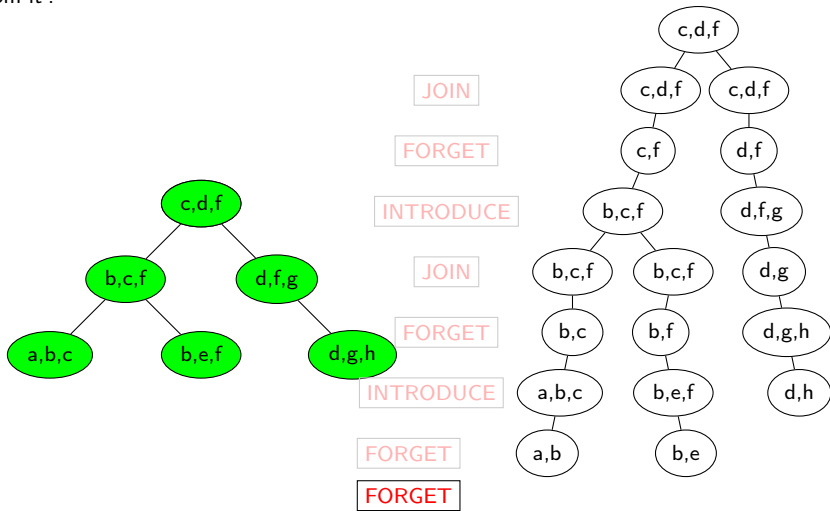
This is a tree decomposition of the graph seen previously. How can we get a nice tree from it?





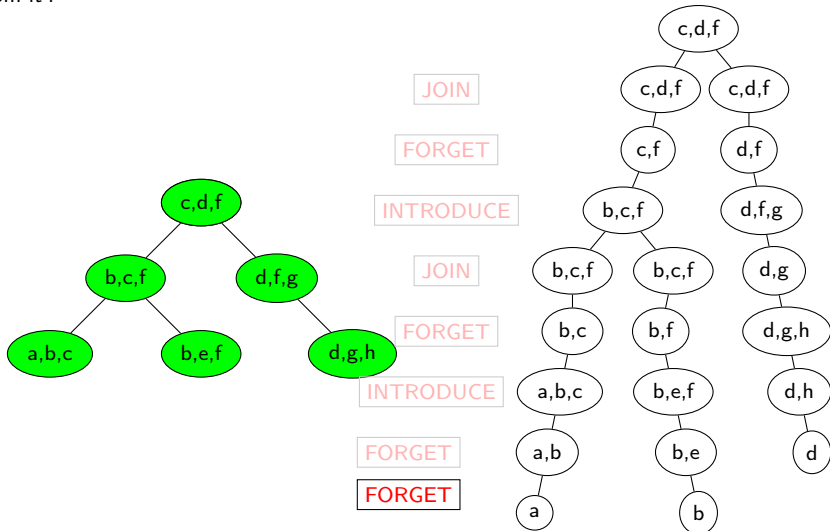
## Example

This is a tree decomposition of the graph seen previously. How can we get a nice tree from it?



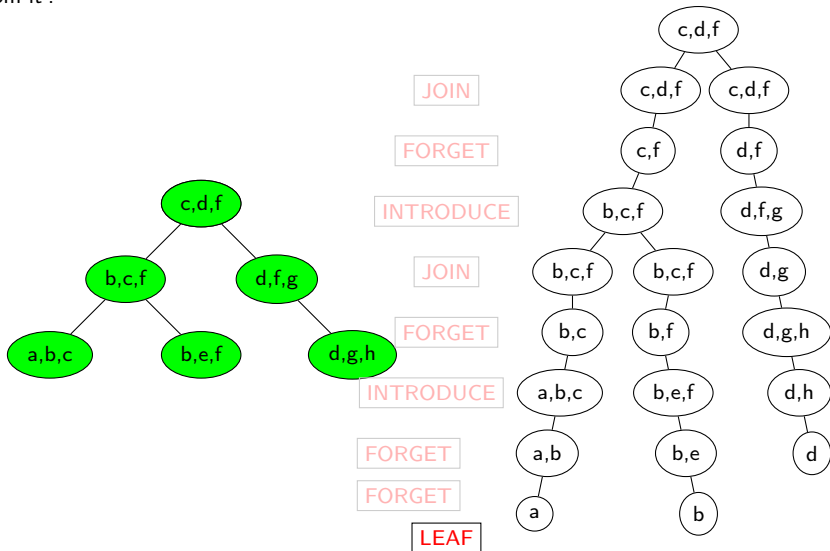
## Example

This is a tree decomposition of the graph seen previously. How can we get a nice tree from it?



## Example

This is a tree decomposition of the graph seen previously. How can we get a nice tree from it?



- 1 Objectives
- 2 Tree decomposition of a graph
  - Tree decomposition
  - Treewidth
  - Nice tree
  - Example
- 3 **Application : k-color**
  - **The problem**
  - **Illustration**
- 4 Bibliography

## Problem (k-color)

*Let  $(G)$  be a graph and  $k$  an integer. We want to know if it is possible to draw each vertex of the graph so that two neighbors have never the same color and with only  $k$  colors.*

*This problem is a problem of decisions problem which is NP-complet.*

## Solution

**tree-width :**  *$k$ -color is possible for a graph  $(G)$  if and only if  $k > \text{treewidth}(G)$ .*

**nice tree :** *a nice tree of  $(G)$  gives a way to find a  $k$ -coloration of  $(G)$  (if  $k > \text{treewidth}(G)$ ).*

# the problem

## Problem ( $k$ -color)

Let  $(G)$  be a graph and  $k$  an integer. We want to know if it is possible to draw each vertex of the graph so that two neighbors have never the same color and with only  $k$  colors.

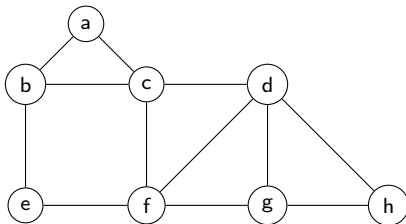
*This problem is a problem of decisions problem which is NP-complet.*

## Solution

**tree-width :**  $k$ -color is possible for a graph  $(G)$  if and only if  $k \geq \text{treewidth}(G)$ .

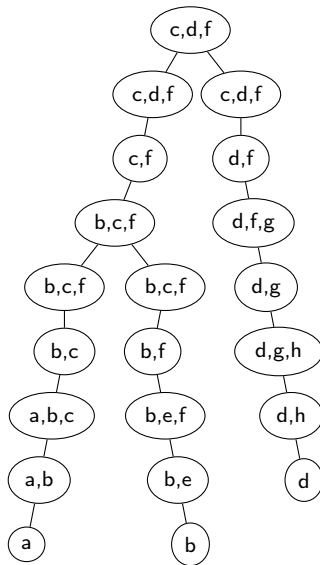
**nice tree :** a nice tree of  $(G)$  gives a way to find a  $k$ -coloration of  $(G)$  (if  $k \geq \text{treewidth}(G)$ ).

**Illustration :** we will use the previously trees to solve the problem with this graph :



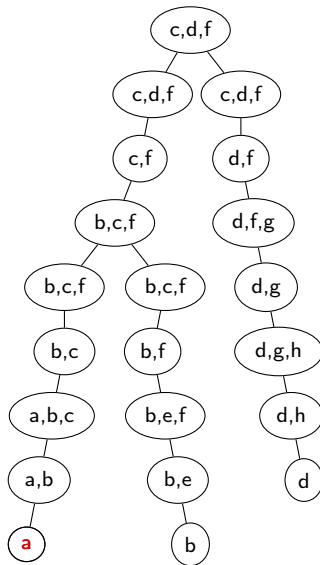
## nice tree of the graph

- $\text{treewith}(G)=2$  : we can solve the problem with 3 colors
- we can fix a color for a “Leaf”
- when we meet an “Introduce” we add a color
- when we meet a “Forget” we can state that the vertex which has disappeared won’t come back (property of the decomposition) and so we can reuse its color.



## nice tree of the graph

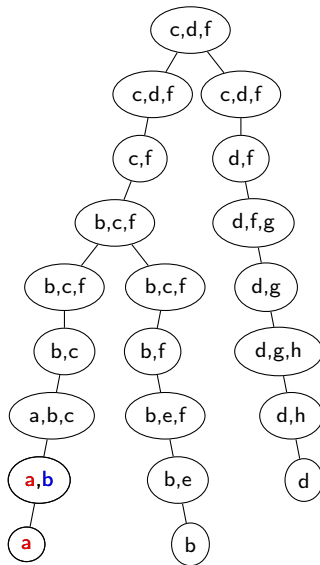
- $\text{treewith}(G)=2$  : we can solve the problem with 3 colors
- we can fix a color for a “Leaf”
- when we meet an “Introduce” we add a color
- when we meet a “Forget” we can state that the vertex which has disappeared won’t come back (property of the decomposition) and so we can reuse its color.





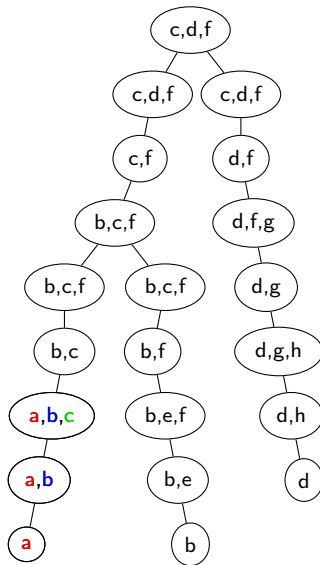
## nice tree of the graph

- $\text{treewidth}(G)=2$  : we can solve the problem with 3 colors
- we can fix a color for a “Leaf”
- when we meet an “Introduce” we add a color
- when we meet a “Forget” we can state that the vertex which has disappeared won’t come back (property of the decomposition) and so we can reuse its color.



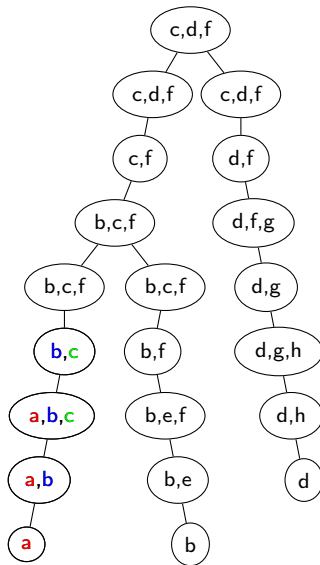
## nice tree of the graph

- $\text{treewidth}(G)=2$  : we can solve the problem with 3 colors
- we can fix a color for a “Leaf”
- when we meet an “Introduce” we add a color
- when we meet a “Forget” we can state that the vertex which has disappeared won’t come back (property of the decomposition) and so we can reuse its color.



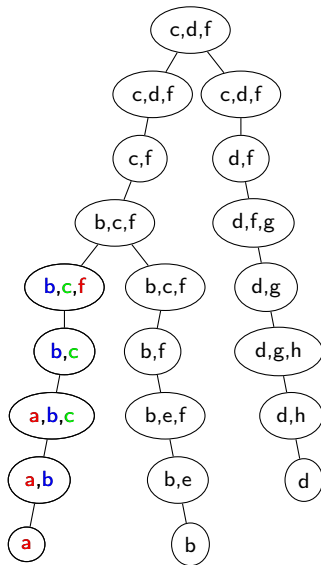
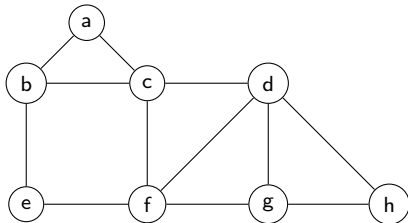
## nice tree of the graph

- $\text{treewidth}(G)=2$  : we can solve the problem with 3 colors
- we can fix a color for a "Leaf"
- when we meet an "Introduce" we add a color
- when we meet a "Forget" we can state that the vertex which has disappeared won't come back (property of the decomposition) and so we can reuse its color.



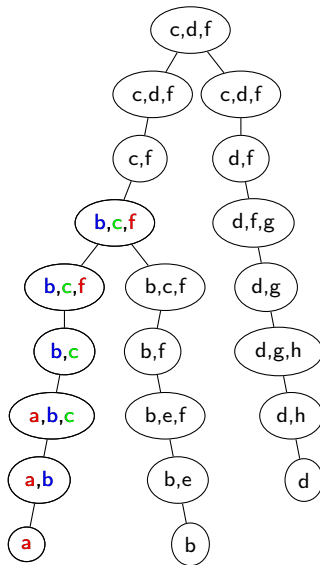
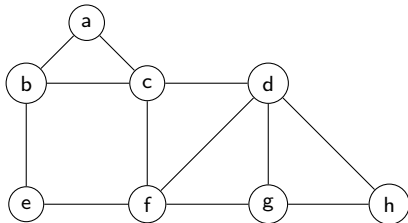
## nice tree of the graph

- $\text{treewidth}(G)=2$  : we can solve the problem with 3 colors
- we can fix a color for a “Leaf”
- when we meet an “Introduce” we add a color
- when we meet a “Forget” we can state that the vertex which has disappeared won’t come back (property of the decomposition) and so we can reuse its color.



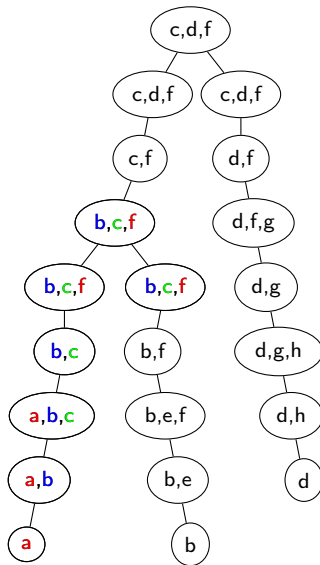
## nice tree of the graph

- $\text{treewidth}(G)=2$  : we can solve the problem with 3 colors
- we can fix a color for a “Leaf”
- when we meet an “Introduce” we add a color
- when we meet a “Forget” we can state that the vertex which has disappeared won’t come back (property of the decomposition) and so we can reuse its color.



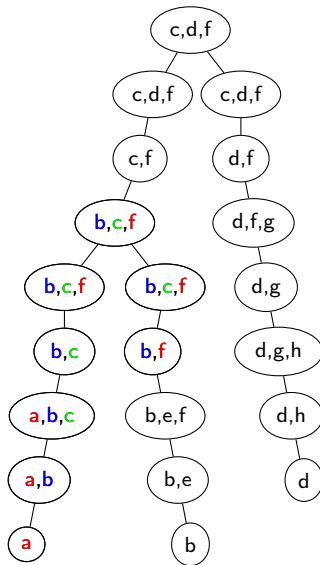
## nice tree of the graph

- $\text{treewidth}(G)=2$  : we can solve the problem with 3 colors
- we can fix a color for a “Leaf”
- when we meet an “Introduce” we add a color
- when we meet a “Forget” we can state that the vertex which has disappeared won’t come back (property of the decomposition) and so we can reuse its color.



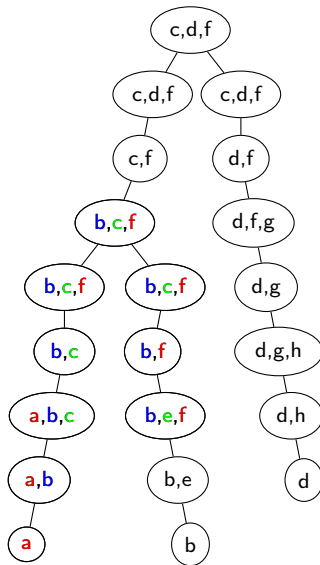
## nice tree of the graph

- $\text{treewidth}(G)=2$  : we can solve the problem with 3 colors
- we can fix a color for a “Leaf”
- when we meet an “Introduce” we add a color
- when we meet a “Forget” we can state that the vertex which has disappeared won't come back (property of the decomposition) and so we can reuse its color.



## nice tree of the graph

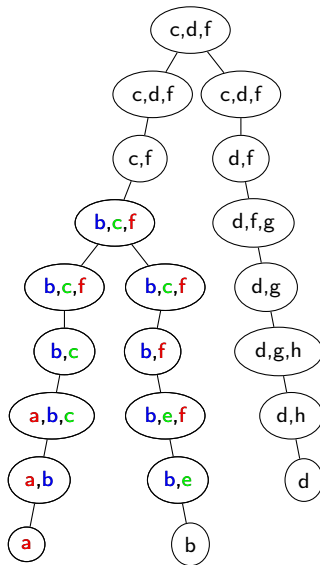
- $\text{treewidth}(G)=2$  : we can solve the problem with 3 colors
- we can fix a color for a “Leaf”
- when we meet an “Introduce” we add a color
- when we meet a “Forget” we can state that the vertex which has disappeared won’t come back (property of the decomposition) and so we can reuse its color.





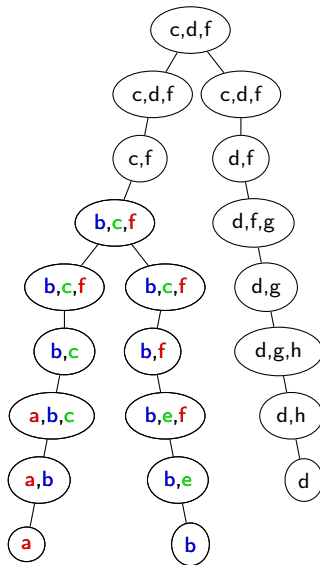
## nice tree of the graph

- $\text{treewidth}(G)=2$  : we can solve the problem with 3 colors
- we can fix a color for a "Leaf"
- when we meet an "Introduce" we add a color
- when we meet a "Forget" we can state that the vertex which has disappeared won't come back (property of the decomposition) and so we can reuse its color.



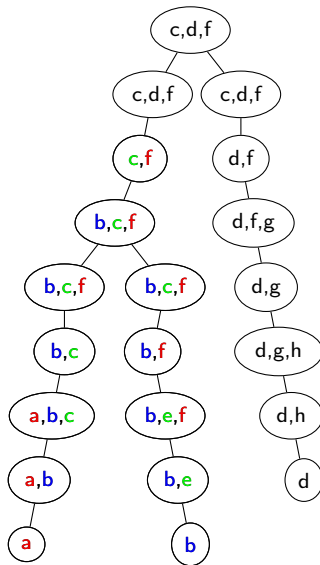
## nice tree of the graph

- $\text{treewidth}(G)=2$  : we can solve the problem with 3 colors
- we can fix a color for a "Leaf"
- when we meet an "Introduce" we add a color
- when we meet a "Forget" we can state that the vertex which has disappeared won't come back (property of the decomposition) and so we can reuse its color.



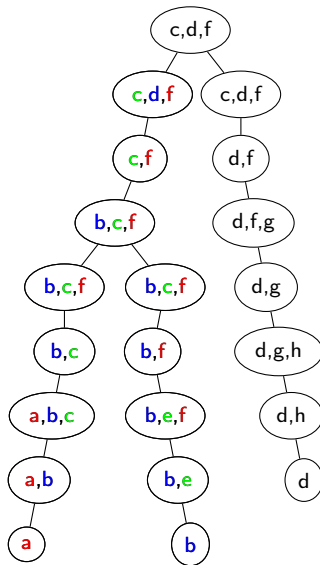
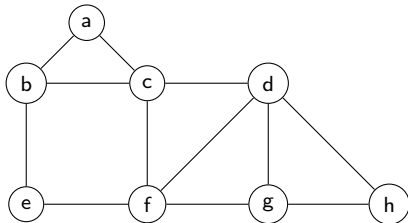
## nice tree of the graph

- $\text{treewidth}(G)=2$  : we can solve the problem with 3 colors
- we can fix a color for a "Leaf"
- when we meet an "Introduce" we add a color
- when we meet a "Forget" we can state that the vertex which has disappeared won't come back (property of the decomposition) and so we can reuse its color.



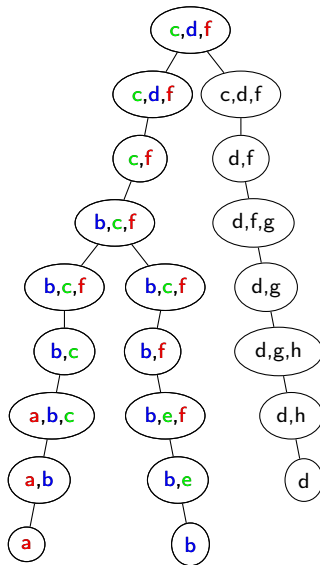
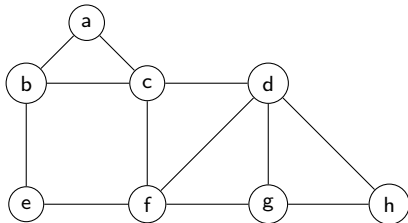
## nice tree of the graph

- $\text{treewidth}(G)=2$  : we can solve the problem with 3 colors
- we can fix a color for a “Leaf”
- when we meet an “Introduce” we add a color
- when we meet a “Forget” we can state that the vertex which has disappeared won’t come back (property of the decomposition) and so we can reuse its color.



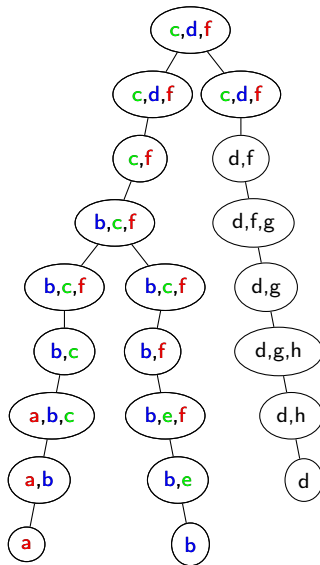
## nice tree of the graph

- $\text{treewidth}(G)=2$  : we can solve the problem with 3 colors
- we can fix a color for a “Leaf”
- when we meet an “Introduce” we add a color
- when we meet a “Forget” we can state that the vertex which has disappeared won’t come back (property of the decomposition) and so we can reuse its color.



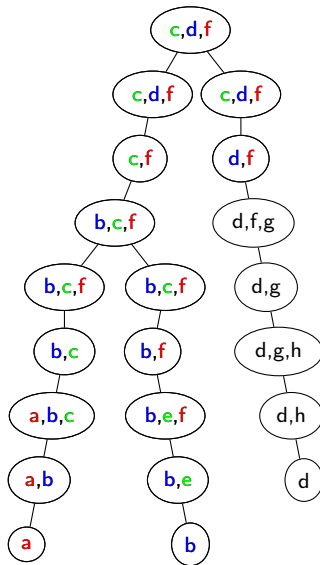
## nice tree of the graph

- $\text{treewidth}(G)=2$  : we can solve the problem with 3 colors
- we can fix a color for a “Leaf”
- when we meet an “Introduce” we add a color
- when we meet a “Forget” we can state that the vertex which has disappeared won’t come back (property of the decomposition) and so we can reuse its color.



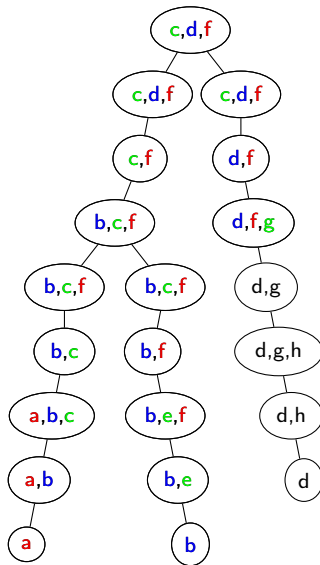
## nice tree of the graph

- $\text{treewidth}(G)=2$  : we can solve the problem with 3 colors
- we can fix a color for a "Leaf"
- when we meet an "Introduce" we add a color
- when we meet a "Forget" we can state that the vertex which has disappeared won't come back (property of the decomposition) and so we can reuse its color.



## nice tree of the graph

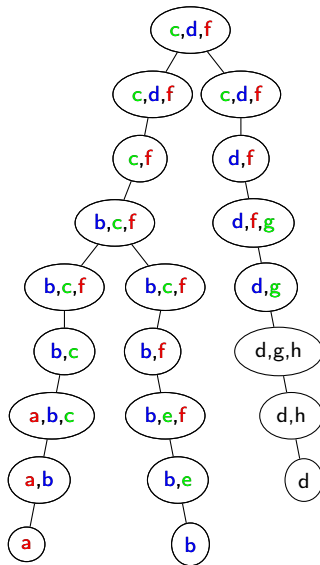
- $\text{treewidth}(G)=2$  : we can solve the problem with 3 colors
- we can fix a color for a “Leaf”
- when we meet an “Introduce” we add a color
- when we meet a “Forget” we can state that the vertex which has disappeared won’t come back (property of the decomposition) and so we can reuse its color.





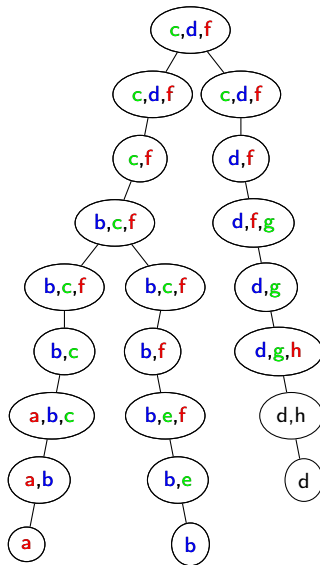
## nice tree of the graph

- $\text{treewidth}(G)=2$  : we can solve the problem with 3 colors
- we can fix a color for a "Leaf"
- when we meet an "Introduce" we add a color
- when we meet a "Forget" we can state that the vertex which has disappeared won't come back (property of the decomposition) and so we can reuse its color.



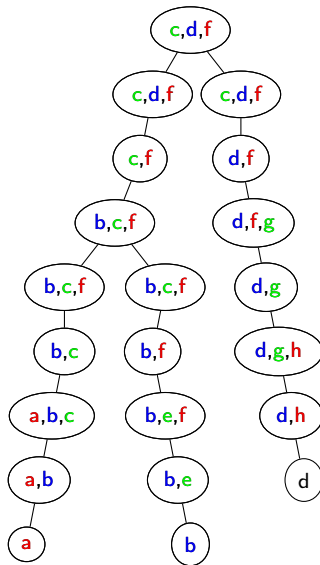
## nice tree of the graph

- $\text{treewidth}(G)=2$  : we can solve the problem with 3 colors
- we can fix a color for a “Leaf”
- when we meet an “Introduce” we add a color
- when we meet a “Forget” we can state that the vertex which has disappeared won’t come back (property of the decomposition) and so we can reuse its color.



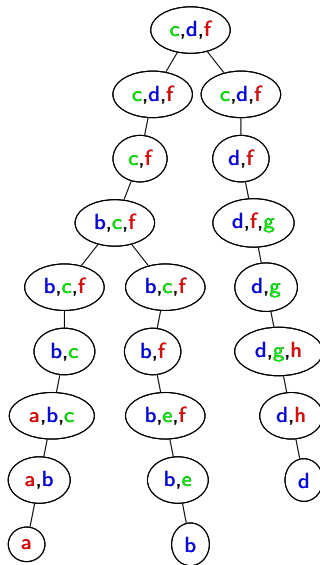
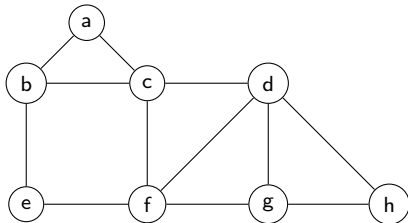
## nice tree of the graph

- $\text{treewidth}(G)=2$  : we can solve the problem with 3 colors
- we can fix a color for a “Leaf”
- when we meet an “Introduce” we add a color
- when we meet a “Forget” we can state that the vertex which has disappeared won't come back (property of the decomposition) and so we can reuse its color.



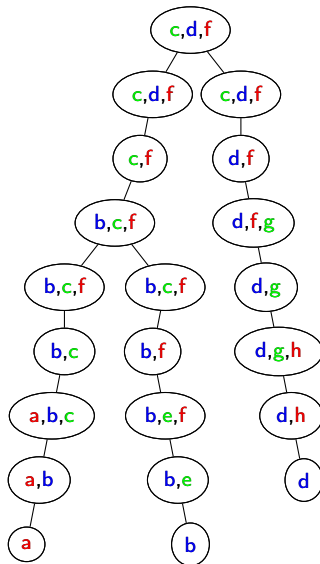
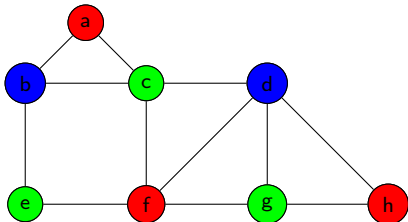
## nice tree of the graph

- $\text{treewidth}(G)=2$  : we can solve the problem with 3 colors
- we can fix a color for a “Leaf”
- when we meet an “Introduce” we add a color
- when we meet a “Forget” we can state that the vertex which has disappeared won't come back (property of the decomposition) and so we can reuse its color.



## nice tree of the graph

- $\text{treewidth}(G)=2$  : we can solve the problem with 3 colors
- we can fix a color for a “Leaf”
- when we meet an “Introduce” we add a color
- when we meet a “Forget” we can state that the vertex which has disappeared won’t come back (property of the decomposition) and so we can reuse its color.



- 1 Objectives
- 2 Tree decomposition of a graph
  - Tree decomposition
  - Treewidth
  - Nice tree
  - Example
- 3 Application :  $k$ -color
  - The problem
  - Illustration
- 4 Bibliography

- (1) Florent Madeleine : lesson for M2 Decim "Complexité des CSP et des requêtes"
- (2) Dániel Marx : Fixed Parameter Algorithms
- (3) wikipedia : articles of the graph section