Graphs' decompositions
and
resolutions of combinatorial problems

Stéphane Secouard
Supervised by : Florent Madeleine

Caen University - Computer science

25 october 2016

université de Caen
Basse-Normandie

Table of contents

Tree decomposition

Definition (Graph decomposition)

A tree T is a decomposition of a graph G when its vertices are arranged satisfying the following properties :

1. If u and v are netighbors in G, then there is a bag of T containing both of them.
2. For every vertex v of G, the bags of T containing v form a connected subtree

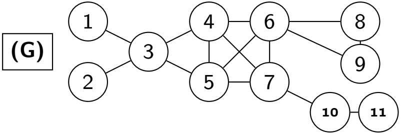
Tree decomposition

Definition (Graph decomposition)

A tree T is a decomposition of a graph G when its vertices are arranged satisfying the following properties :

1. If u and v are netighbors in G, then there is a bag of T containing both of them.
2. For every vertex v of G, the bags of T containing v form a connected subtree

Example :



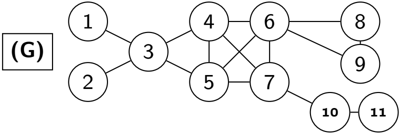
Tree decomposition

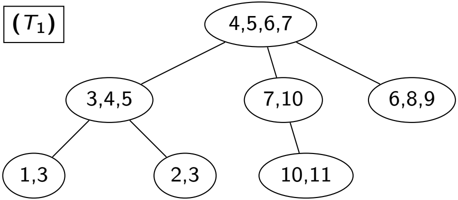
Definition (Graph decomposition)

A tree T is a decomposition of a graph G when its vertices are arranged satisfying the following properties :

1. If u and v are netighbors in G, then there is a bag of T containing both of them.
2. For every vertex v of G, the bags of T containing v form a connected subtree

Example :





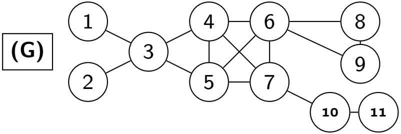
Tree decomposition

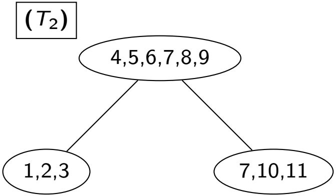
Definition (Graph decomposition)

A tree T is a decomposition of a graph G when its vertices are arranged satisfying the following properties :

1. If u and v are netighbors in G, then there is a bag of T containing both of them.
2. For every vertex v of G, the bags of T containing v form a connected subtree

Example :





Tree decomposition

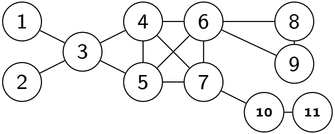
Definition (Graph decomposition)

A tree T is a decomposition of a graph G when its vertices are arranged satisfying the following properties :

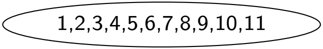
1. If u and v are netighbors in G, then there is a bag of T containing both of them.
2. For every vertex v of G, the bags of T containing v form a connected subtree

Example :

(G)



(T_3)



Treewidth

Definition (treewidth)

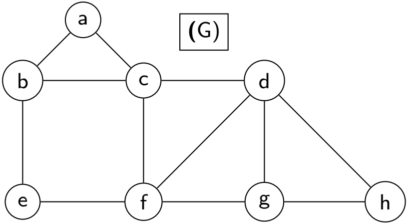
- The *width* of a decomposition is largest bag size - 1.
- The *treewidth* of a graph is the width of the best decomposition of this graph.

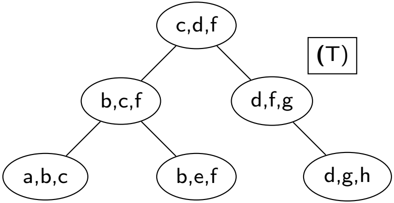
Treewidth

Definition (treewidth)

- The *width* of a decomposition is largest bag size - 1.
- The *treewidth* of a graph is the width of the best decomposition of this graph.

Example : Assume (T) is one of the best representation og (G) :



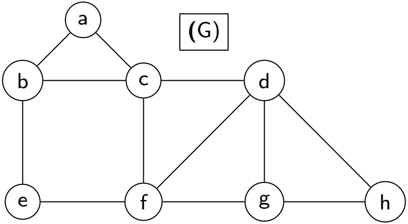


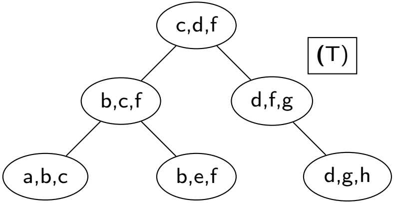
Treewidth

Definition (treewidth)

- The *width* of a decomposition is largest bag size - 1.
- The *treewidth* of a graph is the width of the best decomposition of this graph.

Example : Assume (T) is one of the best representation og (G) :





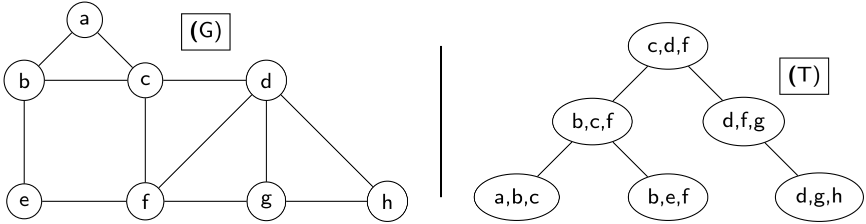
Treewidth(G)=3-1=2

Treewidth

Definition (treewidth)

- The *width* of a decomposition is largest bag size - 1.
- The *treewidth* of a graph is the width of the best decomposition of this graph.

Example : Assume (T) is one of the best representation og (G) :



Treewidth(G)=3-1=2

Remark

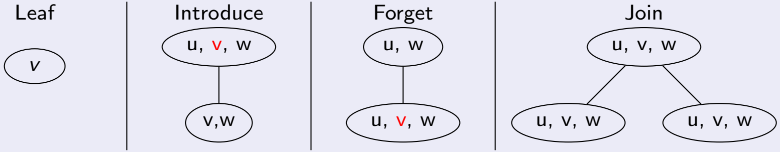
The treewidth of a tree is 1 and if a graph have a treewidth of 1 we can claim that this graph is a forest (i.e. a collection of trees).

Nice tree

Definition (nice tree)

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

- Leaf : no children, $|B_x| = 1$
- Introduce : 1 child y, $B_x = B_y \cup \{v\}$ for some vertex v
- Forget : 1 child y, $B_x = B_y \backslash \{v\}$ for some vertex v.
- Join : 2 children y_1, y_2 with $B_x = B_{y_1} = B_{y_2}$

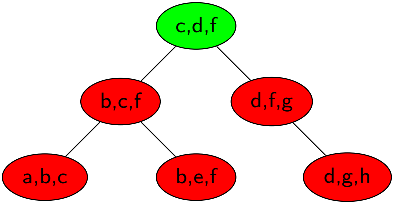


Remark

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

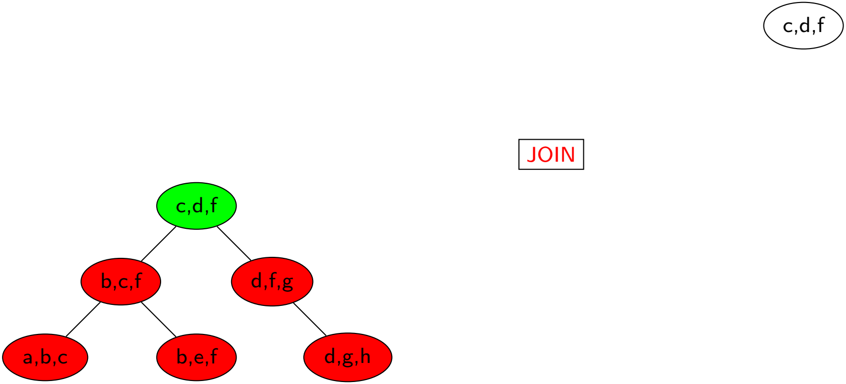
example

This is a decomposition tree saw previously. How could we obtain a nice tree from it ?



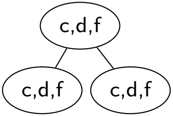
example

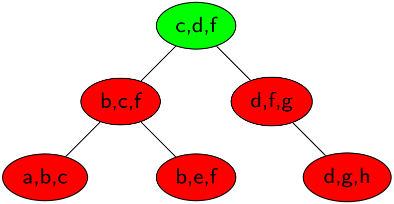
This is a decomposition tree saw previously. How could we obtain a nice tree from it ?



example

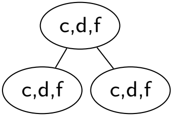
This is a decomposition tree saw previously. How could we obtain a nice tree from it ?



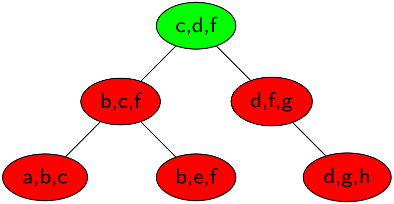


example

This is a decomposition tree saw previously. How could we obtain a nice tree from it ?

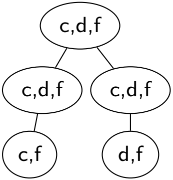


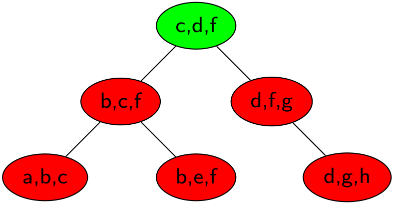
FORGET



example

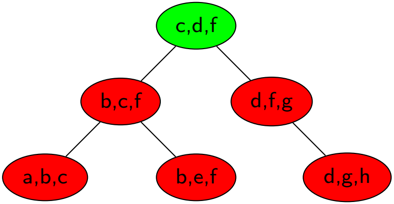
This is a decomposition tree saw previously. How could we obtain a nice tree from it ?



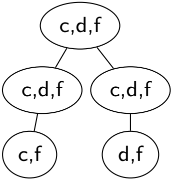


example

This is a decomposition tree saw previously. How could we obtain a nice tree from it ?

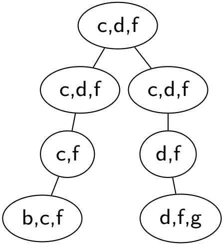


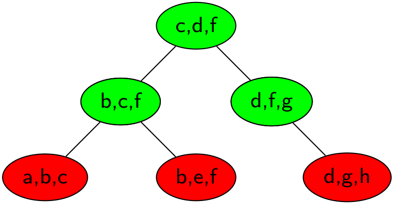
INTRODUCE



example

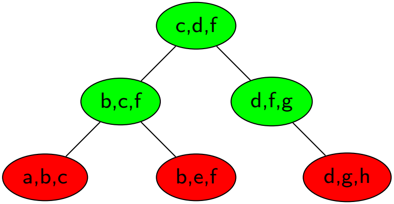
This is a decomposition tree saw previously. How could we obtain a nice tree from it ?



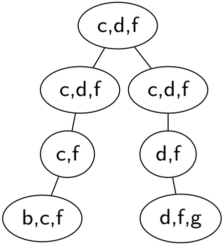


example

This is a decomposition tree saw previously. How could we obtain a nice tree from it ?

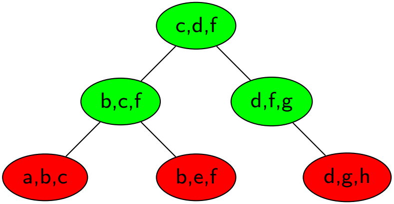


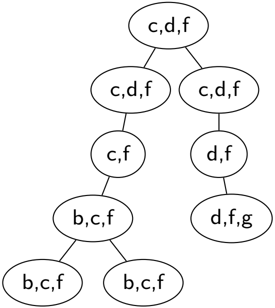
JOIN



example

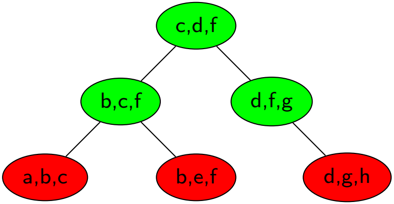
This is a decomposition tree saw previously. How could we obtain a nice tree from it ?



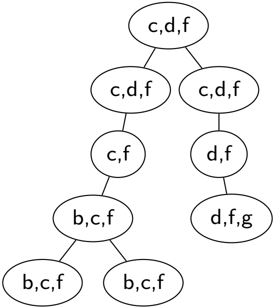


example

This is a decomposition tree saw previously. How could we obtain a nice tree from it ?

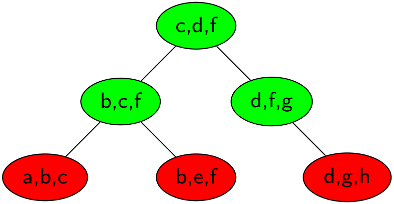


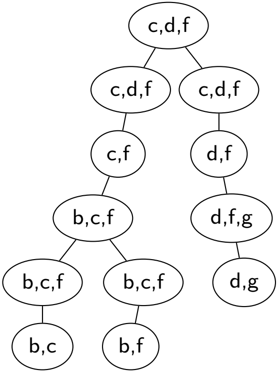
FORGET



example

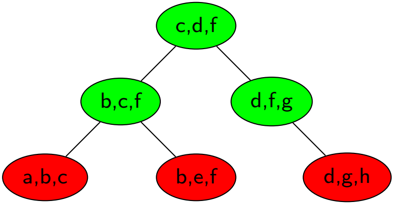
This is a decomposition tree saw previously. How could we obtain a nice tree from it ?



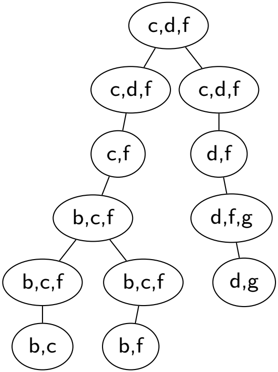


example

This is a decomposition tree saw previously. How could we obtain a nice tree from it ?

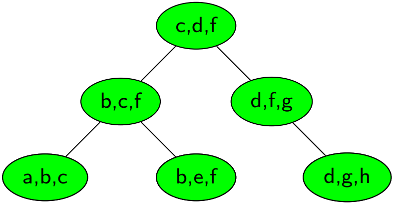


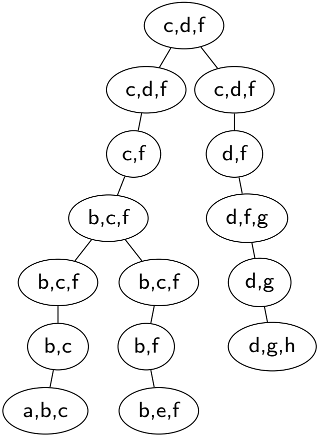
INTRODUCE



example

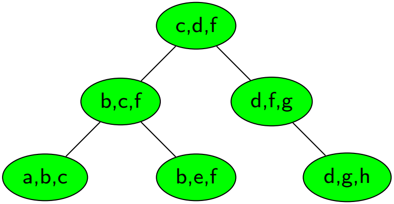
This is a decomposition tree saw previously. How could we obtain a nice tree from it ?



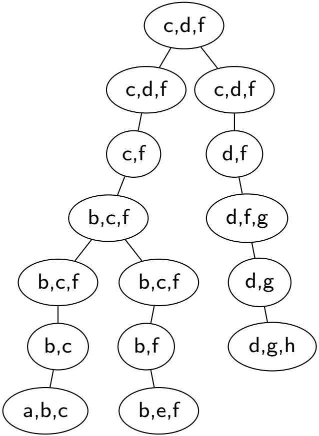


example

This is a decomposition tree saw previously. How could we obtain a nice tree from it ?

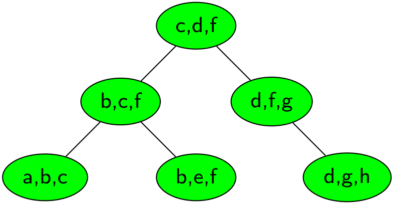


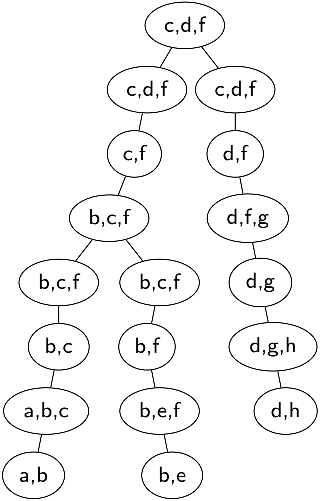
FORGET



example

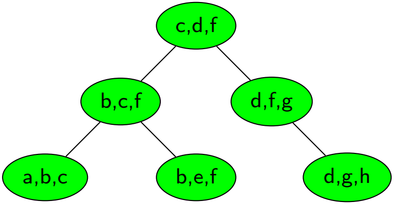
This is a decomposition tree saw previously. How could we obtain a nice tree from it ?



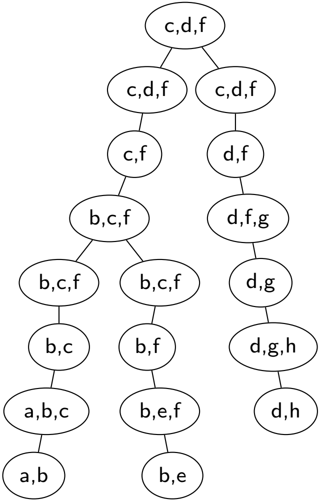


example

This is a decomposition tree saw previously. How could we obtain a nice tree from it ?

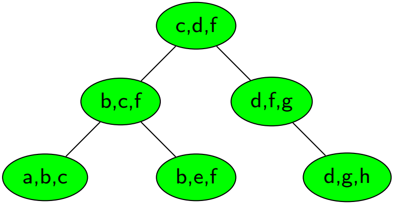


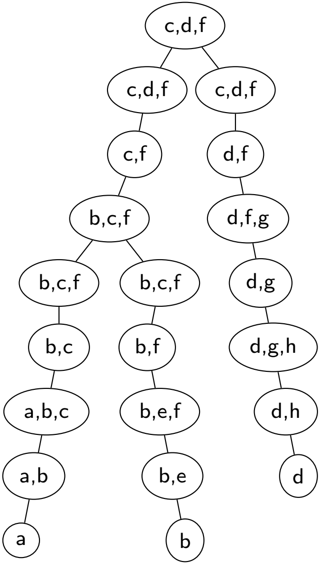
FORGET



example

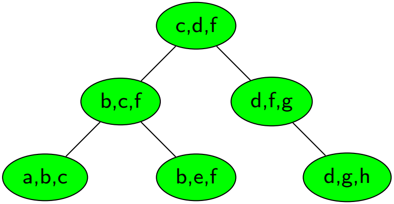
This is a decomposition tree saw previously. How could we obtain a nice tree from it ?



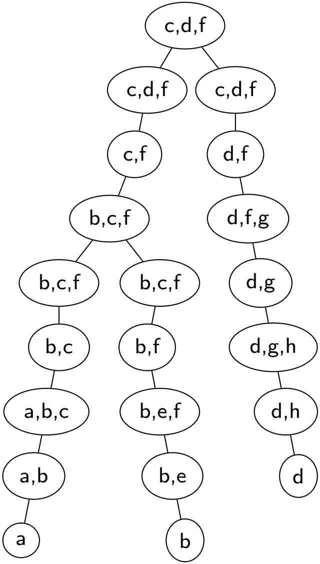


example

This is a decomposition tree saw previously. How could we obtain a nice tree from it ?



LEAF



the problem

Problem (k-color)

problem : *Let (G) be a graph and k an integer. We want to know if it is possible to draw each vertice of the graph such that two neighbors have never the same color and with only k color.*
This problem is a problem of decision which is NP.

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

nice tree : *a nice tree of (G) give a way to find a k-coloration of (G) (if* $k \geqslant treewidth(G)$*).*

the problem

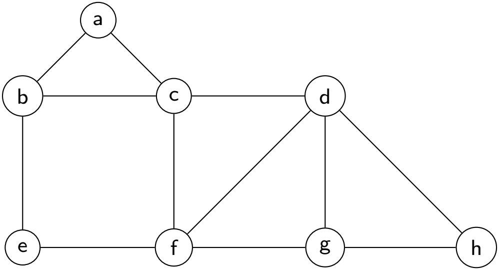
Problem (k-color)

problem : *Let (G) be a graph and k an integer. We want to know if it is possible to draw each vertice of the graph such that two neighbors have never the same color and with only k color.*
This problem is a problem of decision which is NP.

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

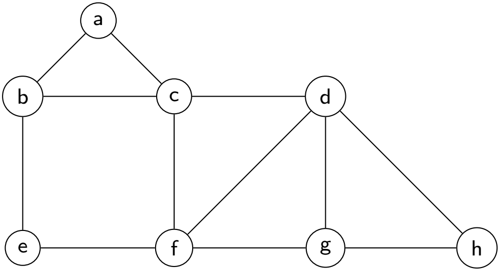
nice tree : *a nice tree of (G) give a way to find a k-coloration of (G) (if $k \geqslant treewidth(G)$).*

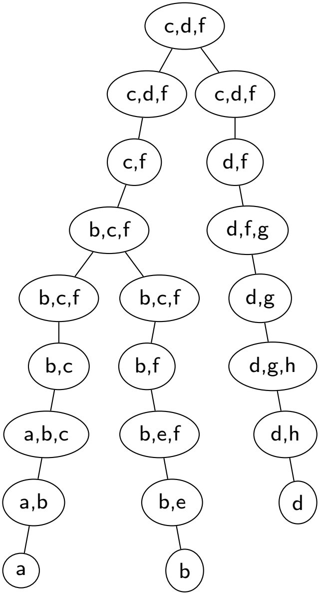
Illustration : we will use the previously trees to solve the problem with this graph :



nice tree of the graph

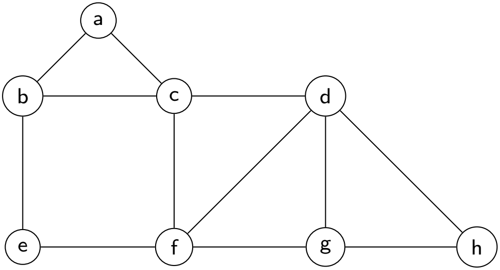
- treewith(G)=3 : we can solve the problem with 3 colors
- we can fixe a color for a "Leaf"
- when we meet an "Introduce" we add a color
- when we meet a "Forget" we can claim that the vertex which has disappeared won't come back (propertie of the decomposition) and so we can reuse the color.

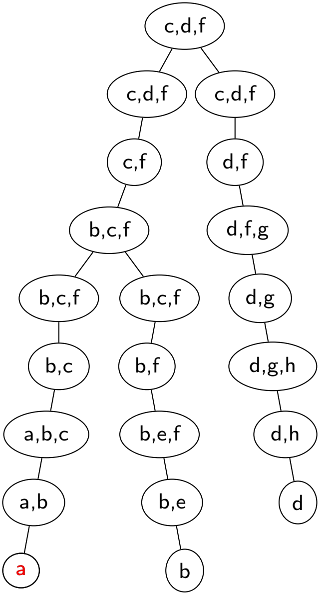




nice tree of the graph

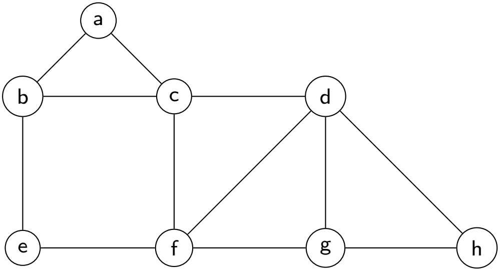
- treewith(G)=3 : we can solve the problem with 3 colors
- we can fixe a color for a "Leaf"
- when we meet an "Introduce" we add a color
- when we meet a "Forget" we can claim that the vertex which has disappeared won't come back (propertie of the decomposition) and so we can reuse the color.

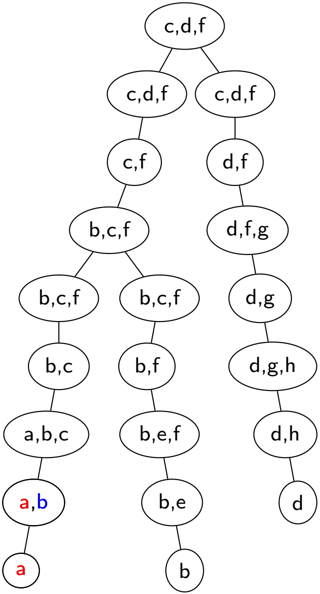




nice tree of the graph

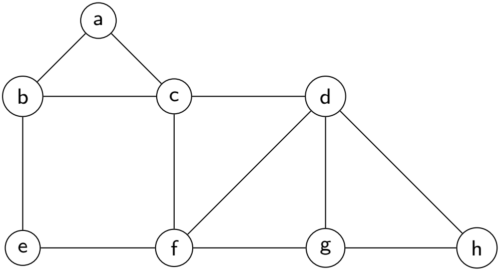
- treewith(G)=3 : we can solve the problem with 3 colors
- we can fixe a color for a “Leaf”
- when we meet an “Introduce” we add a color
- when we meet a “Forget” we can claim that the vertex which has disappeared won’t come back (propertie of the decomposition) and so we can reuse the color.

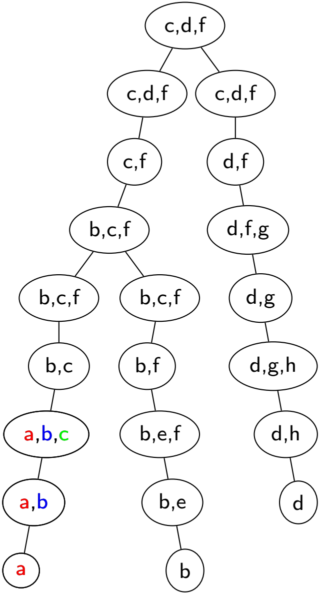




nice tree of the graph

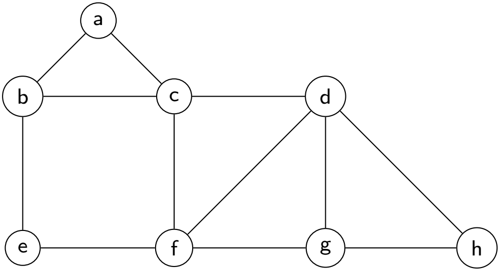
- treewith(G)=3 : we can solve the problem with 3 colors
- we can fixe a color for a “Leaf”
- when we meet an “Introduce” we add a color
- when we meet a “Forget” we can claim that the vertex which has disappeared won’t come back (propertie of the decomposition) and so we can reuse the color.

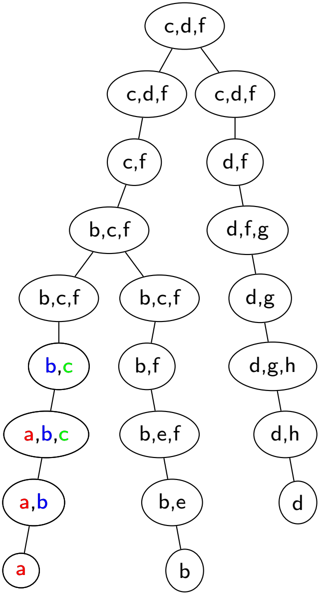




nice tree of the graph

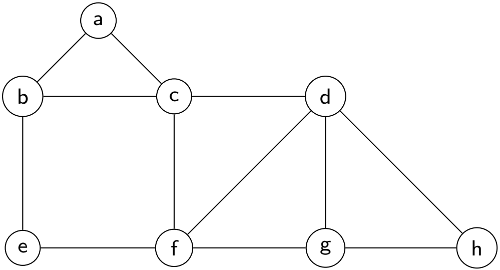
- treewith(G)=3 : we can solve the problem with 3 colors
- we can fixe a color for a "Leaf"
- when we meet an "Introduce" we add a color
- when we meet a "Forget" we can claim that the vertex which has disappeared won't come back (propertie of the decomposition) and so we can reuse the color.

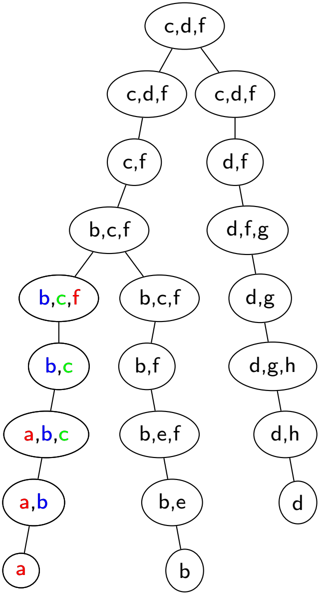




nice tree of the graph

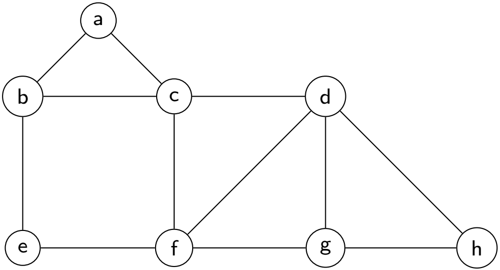
- treewith(G)=3 : we can solve the problem with 3 colors
- we can fixe a color for a "Leaf"
- when we meet an "Introduce" we add a color
- when we meet a "Forget" we can claim that the vertex which has disappeared won't come back (propertie of the decomposition) and so we can reuse the color.

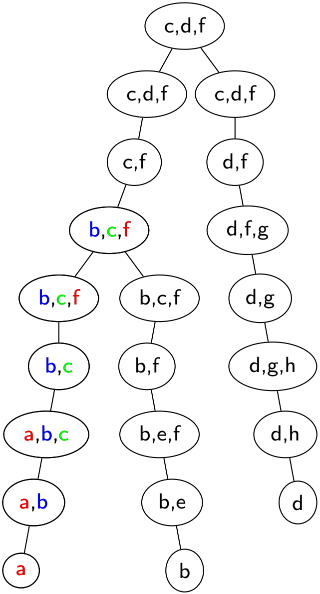




nice tree of the graph

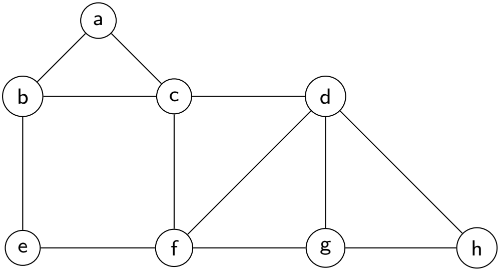
- treewith(G)=3 : we can solve the problem with 3 colors
- we can fixe a color for a “Leaf”
- when we meet an “Introduce” we add a color
- when we meet a “Forget” we can claim that the vertex which has disappeared won’t come back (propertie of the decomposition) and so we can reuse the color.

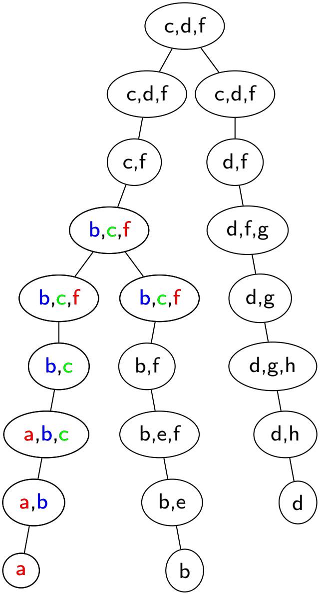




nice tree of the graph

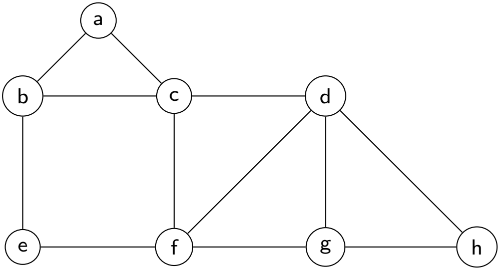
- treewith(G)=3 : we can solve the problem with 3 colors
- we can fixe a color for a "Leaf"
- when we meet an "Introduce" we add a color
- when we meet a "Forget" we can claim that the vertex which has disappeared won't come back (propertie of the decomposition) and so we can reuse the color.

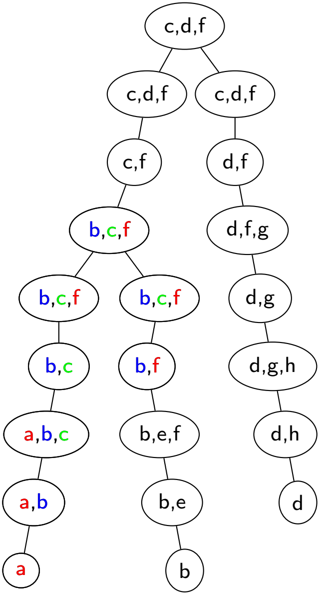




nice tree of the graph

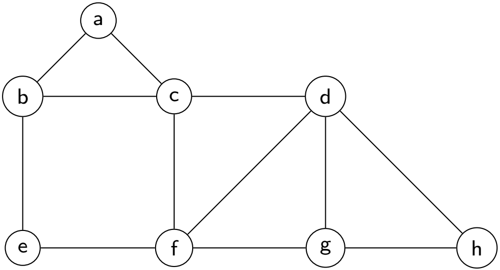
- treewith(G)=3 : we can solve the problem with 3 colors
- we can fixe a color for a "Leaf"
- when we meet an "Introduce" we add a color
- when we meet a "Forget" we can claim that the vertex which has disappeared won't come back (propertie of the decomposition) and so we can reuse the color.

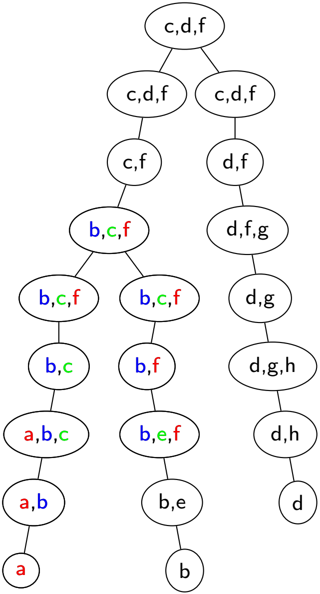




nice tree of the graph

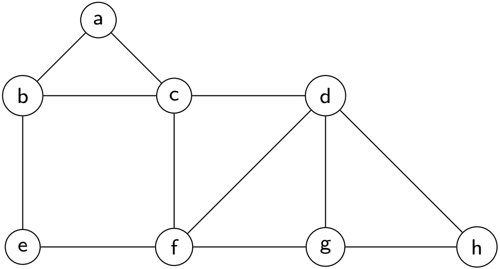
- treewith(G)=3 : we can solve the problem with 3 colors
- we can fixe a color for a "Leaf"
- when we meet an "Introduce" we add a color
- when we meet a "Forget" we can claim that the vertex which has disappeared won't come back (propertie of the decomposition) and so we can reuse the color.

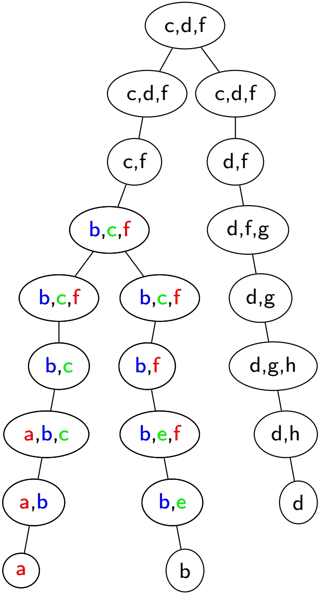




nice tree of the graph

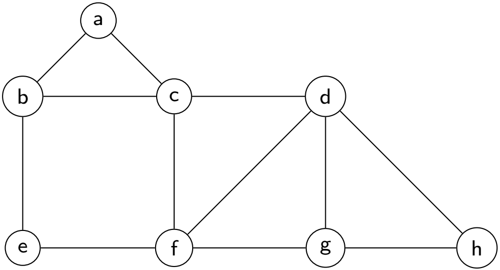
- treewith(G)=3 : we can solve the problem with 3 colors
- we can fixe a color for a “Leaf”
- when we meet an “Introduce” we add a color
- when we meet a “Forget” we can claim that the vertex which has disappeared won’t come back (propertie of the decomposition) and so we can reuse the color.

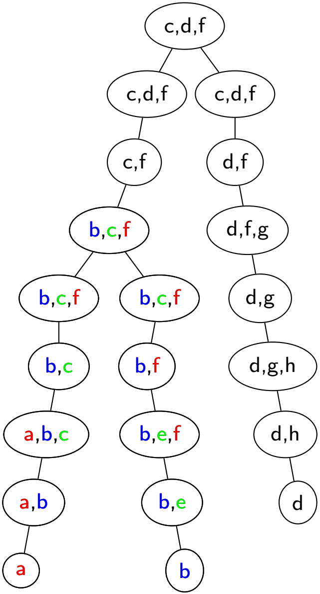




nice tree of the graph

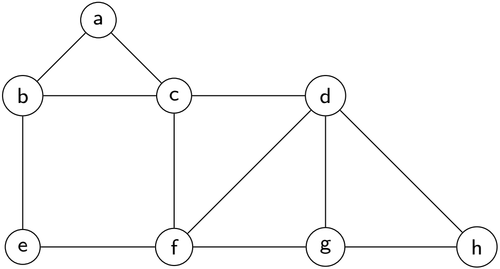
- treewith(G)=3 : we can solve the problem with 3 colors
- we can fixe a color for a "Leaf"
- when we meet an "Introduce" we add a color
- when we meet a "Forget" we can claim that the vertex which has disappeared won't come back (propertie of the decomposition) and so we can reuse the color.

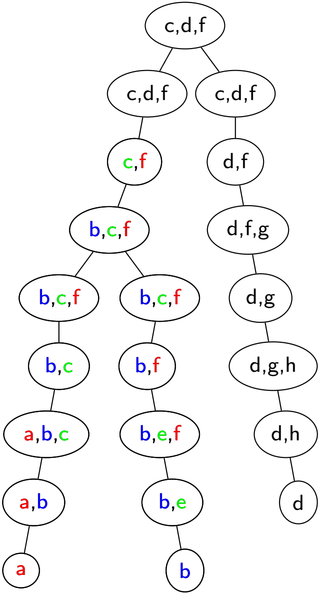




nice tree of the graph

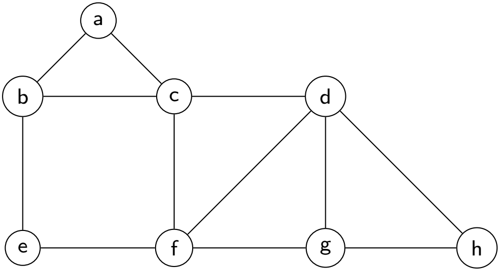
- treewith(G)=3 : we can solve the problem with 3 colors
- we can fixe a color for a "Leaf"
- when we meet an "Introduce" we add a color
- when we meet a "Forget" we can claim that the vertex which has disappeared won't come back (propertie of the decomposition) and so we can reuse the color.

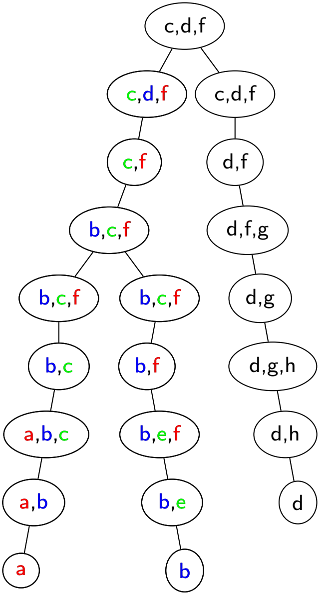




nice tree of the graph

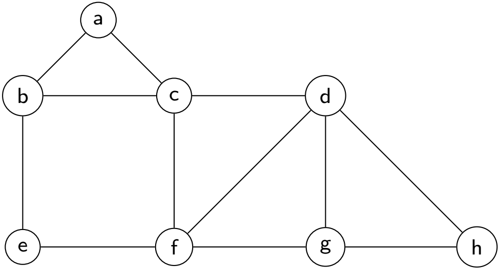
- treewith(G)=3 : we can solve the problem with 3 colors
- we can fixe a color for a “Leaf”
- when we meet an “Introduce” we add a color
- when we meet a “Forget” we can claim that the vertex which has disappeared won’t come back (propertie of the decomposition) and so we can reuse the color.

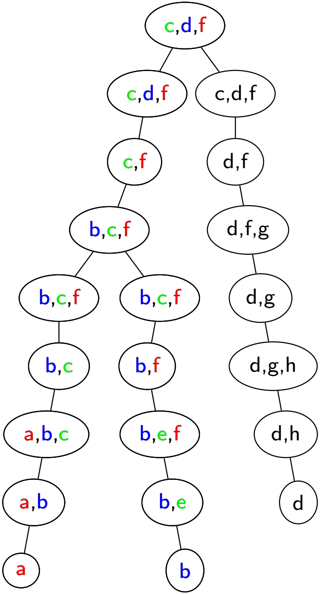




nice tree of the graph

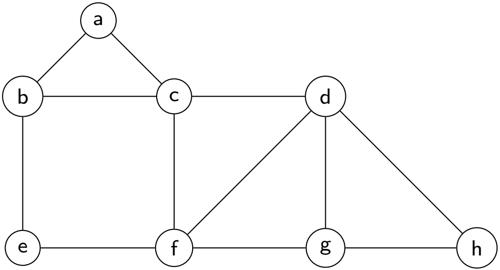
- treewith(G)=3 : we can solve the problem with 3 colors
- we can fixe a color for a "Leaf"
- when we meet an "Introduce" we add a color
- when we meet a "Forget" we can claim that the vertex which has disappeared won't come back (propertie of the decomposition) and so we can reuse the color.

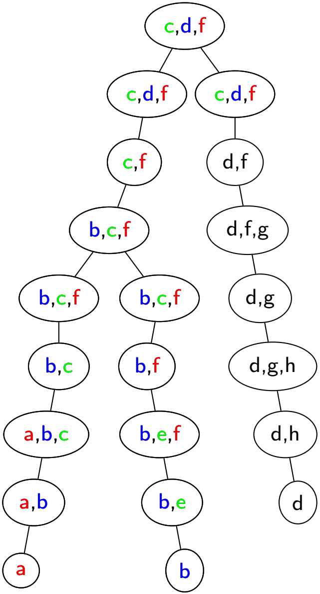




nice tree of the graph

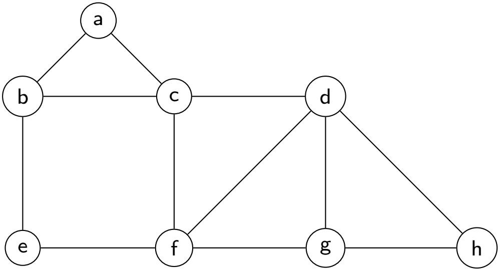
- treewith(G)=3 : we can solve the problem with 3 colors
- we can fixe a color for a "Leaf"
- when we meet an "Introduce" we add a color
- when we meet a "Forget" we can claim that the vertex which has disappeared won't come back (propertie of the decomposition) and so we can reuse the color.

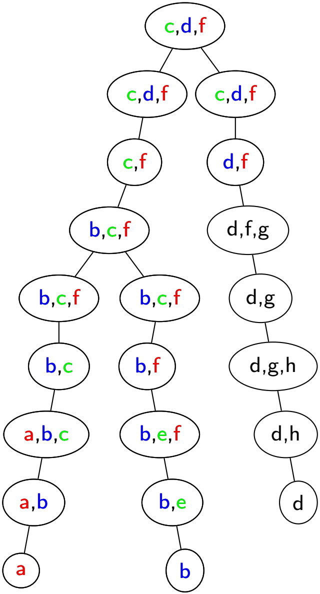




nice tree of the graph

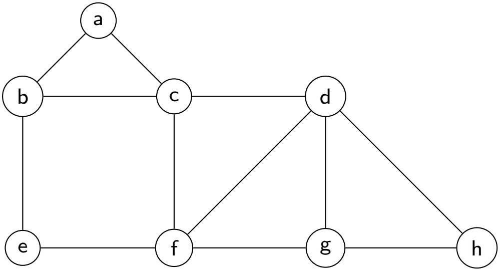
- treewith(G)=3 : we can solve the problem with 3 colors
- we can fixe a color for a "Leaf"
- when we meet an "Introduce" we add a color
- when we meet a "Forget" we can claim that the vertex which has disappeared won't come back (propertie of the decomposition) and so we can reuse the color.

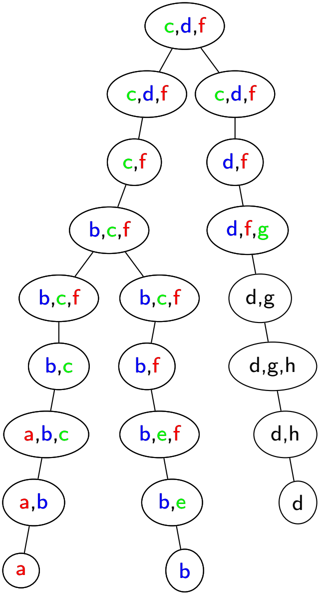




nice tree of the graph

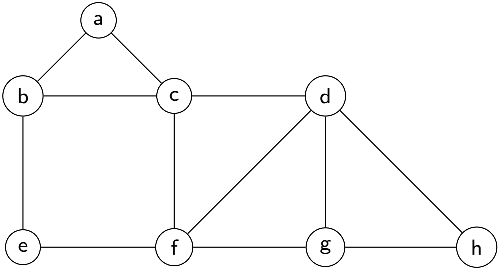
- treewith(G)=3 : we can solve the problem with 3 colors
- we can fixe a color for a “Leaf”
- when we meet an “Introduce” we add a color
- when we meet a “Forget” we can claim that the vertex which has disappeared won’t come back (propertie of the decomposition) and so we can reuse the color.

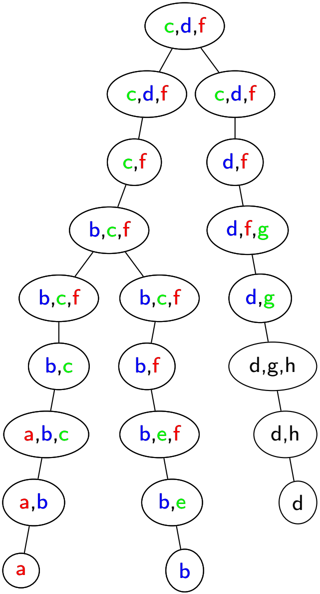




nice tree of the graph

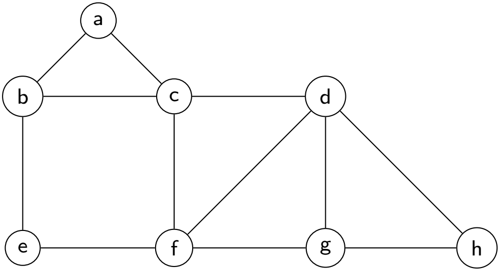
- treewith(G)=3 : we can solve the problem with 3 colors
- we can fixe a color for a “Leaf”
- when we meet an “Introduce” we add a color
- when we meet a “Forget” we can claim that the vertex which has disappeared won’t come back (propertie of the decomposition) and so we can reuse the color.

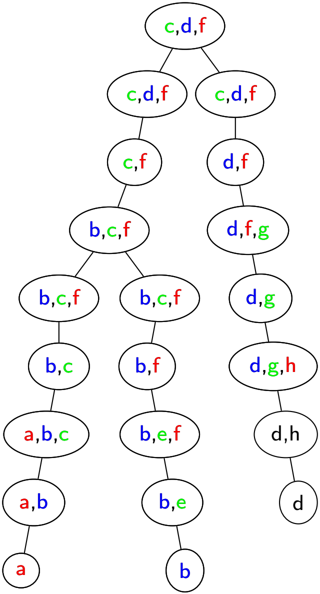




nice tree of the graph

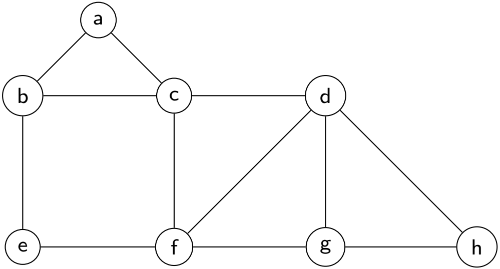
- treewith(G)=3 : we can solve the problem with 3 colors
- we can fixe a color for a "Leaf"
- when we meet an "Introduce" we add a color
- when we meet a "Forget" we can claim that the vertex which has disappeared won't come back (propertie of the decomposition) and so we can reuse the color.

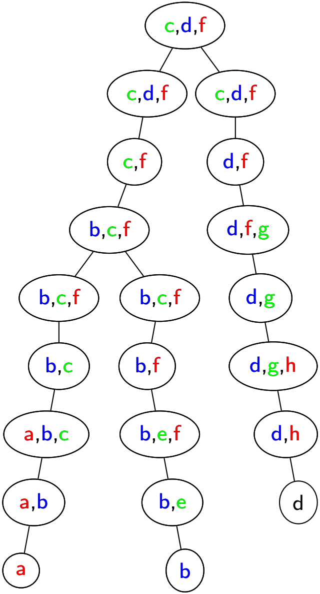




nice tree of the graph

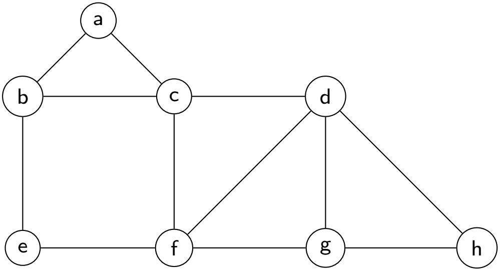
- treewith(G)=3 : we can solve the problem with 3 colors
- we can fixe a color for a "Leaf"
- when we meet an "Introduce" we add a color
- when we meet a "Forget" we can claim that the vertex which has disappeared won't come back (propertie of the decomposition) and so we can reuse the color.

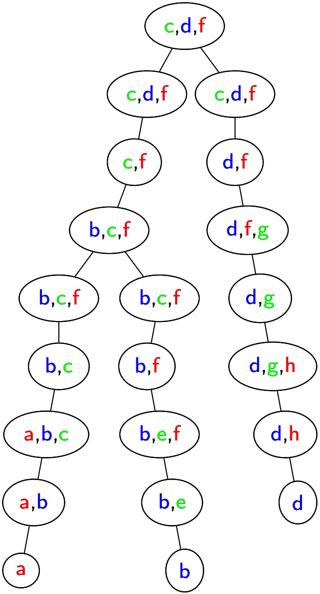




nice tree of the graph

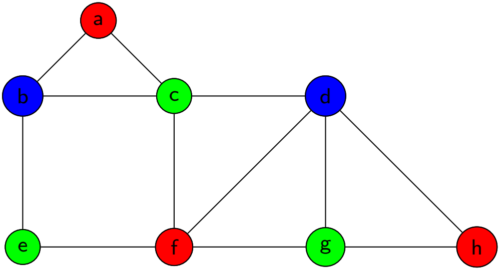
- treewith(G)=3 : we can solve the problem with 3 colors
- we can fixe a color for a “Leaf”
- when we meet an “Introduce” we add a color
- when we meet a “Forget” we can claim that the vertex which has disappeared won’t come back (propertie of the decomposition) and so we can reuse the color.

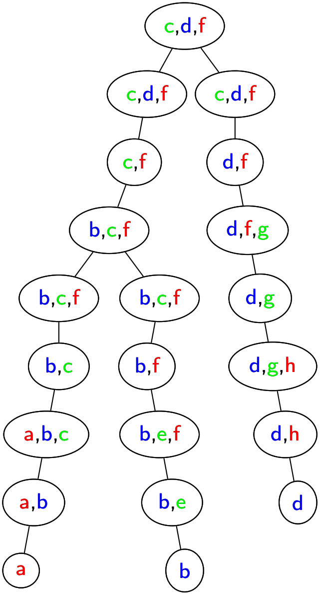




nice tree of the graph

- treewith(G)=3 : we can solve the problem with 3 colors
- we can fixe a color for a "Leaf"
- when we meet an "Introduce" we add a color
- when we meet a "Forget" we can claim that the vertex which has disappeared won't come back (propertie of the decomposition) and so we can reuse the color.





Other applications

Mission

Bibliography