A Tourist Guide through Treewidth

Valia Mitsou

NTUA

Treewidth Notion

- Introduced by Robertson and Seymour in 1983.
- Intuitively represents how close is a graph G to a tree.
- The treewidth of a tree is 1, of a cycle is 2 and of a clique K_n is $n-1$.

Tree Decomposition

Definition: A tree decomposition of a graph $G(V, E)$ is a pair $(\mathcal{X}, T)$, where $T(I, F)$ is a tree and $\mathcal{X}$ is a family of "bags".
Every $i \in I$ is associated with a bag $X_i \in \mathcal{X}$.
Every bag contains some nodes of V s.t.:

- $\bigcup_{i \in I} X_i = V$
- for all edges $\{u, w\} \in E$ $\exists i \in I$ with $u \in X_i$ and $w \in X_i$
- $\forall i, j, k \in I$: if j is on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$

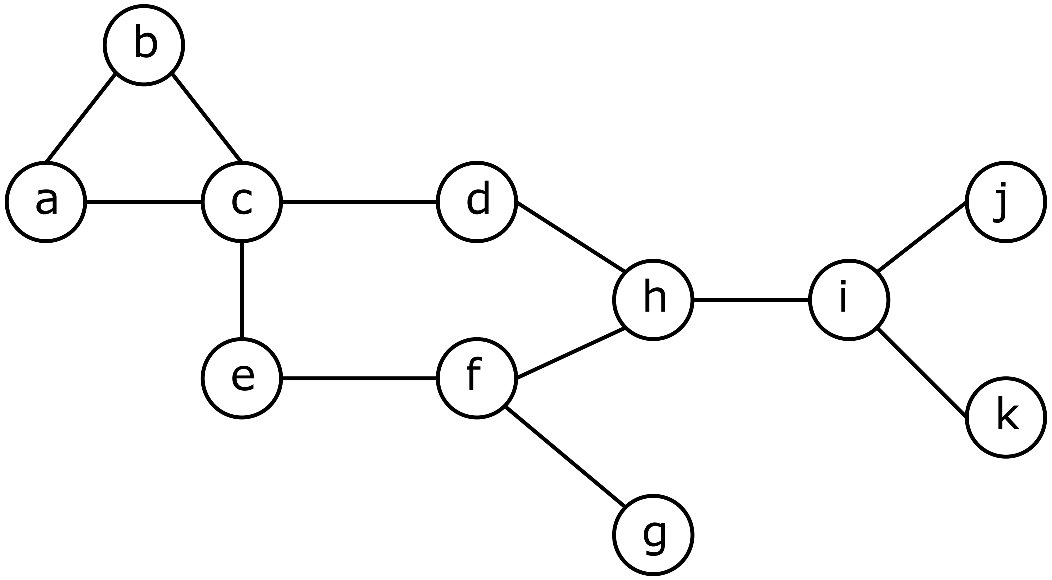
Treewidth

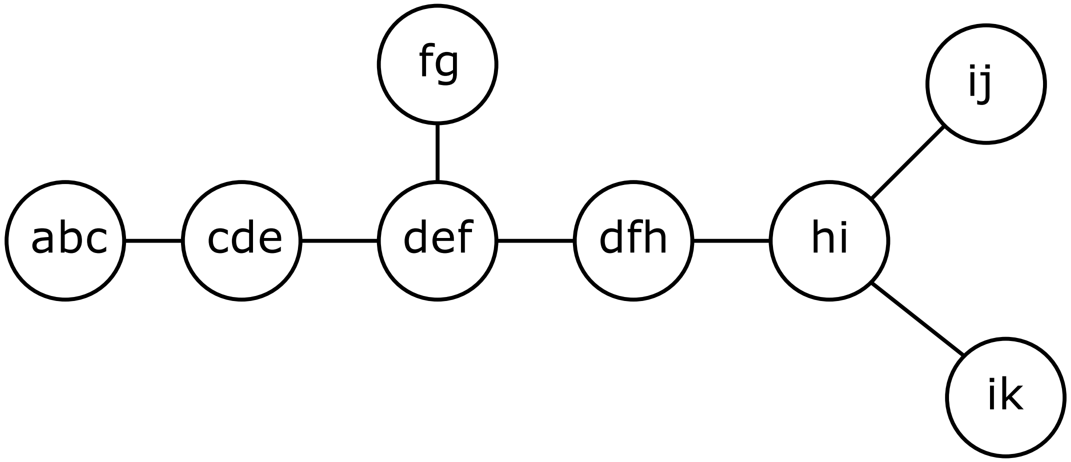
- The *width* of a tree decomposition $((I, F), \{X_i | i \in I\})$ is $\max_{i \in I} |X_i| - 1$.
- The *treewidth* of a graph G is the minimum width over all tree decompositions of G.

Path Decomposition

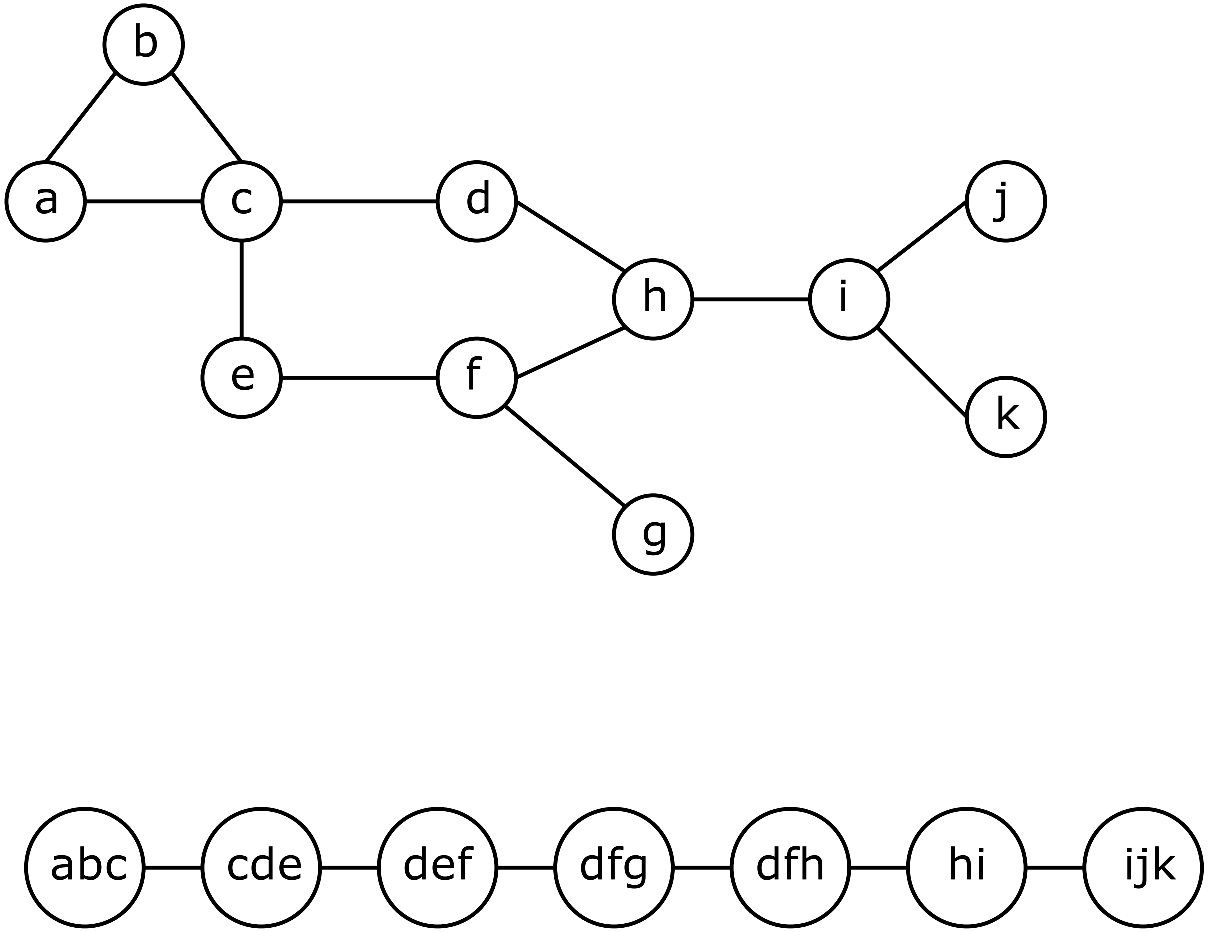
- *Path decomposition* of a graph G is a tree decomposition $(\mathcal{X}, T)$, where T is a path.
- We define *pathwidth* in similar way.
- treewidth of $G \leq$ pathwidth of G.
- Computing the treewidth or the pathwidth of an arbitrary graph is NP-Complete.

Example for tree decomposition





Example for path decomposition



What is a minor

A minor H of a graph G can be obtained from H by

- vertex deletions
- edge deletions
- edge contractions (replace two adjacent verices u, v by a new one that is adjacent to all neighbors of u or of v).

Wagner's Conjecture

- ***Theorem 1****: If $\mathcal{G}$ is a class of graphs that is closed under minors, then there exists a finite set of graphs $ob\mathcal{G}$, the obstruction set of $\mathcal{G}$, such for every graph $H : H \in \mathcal{G}$, iff no element of $ob\mathcal{G}$ is a minor of H.*
- ***Theorem 2****: If $\mathcal{G}$ is a class of graphs that is closed under minors, then there exists an algorithm that decides membership in $\mathcal{G}$ in $O(n^3)$ time* (check for each element in the obstruction set whether it is a minor of the input graph). Non-constructive.

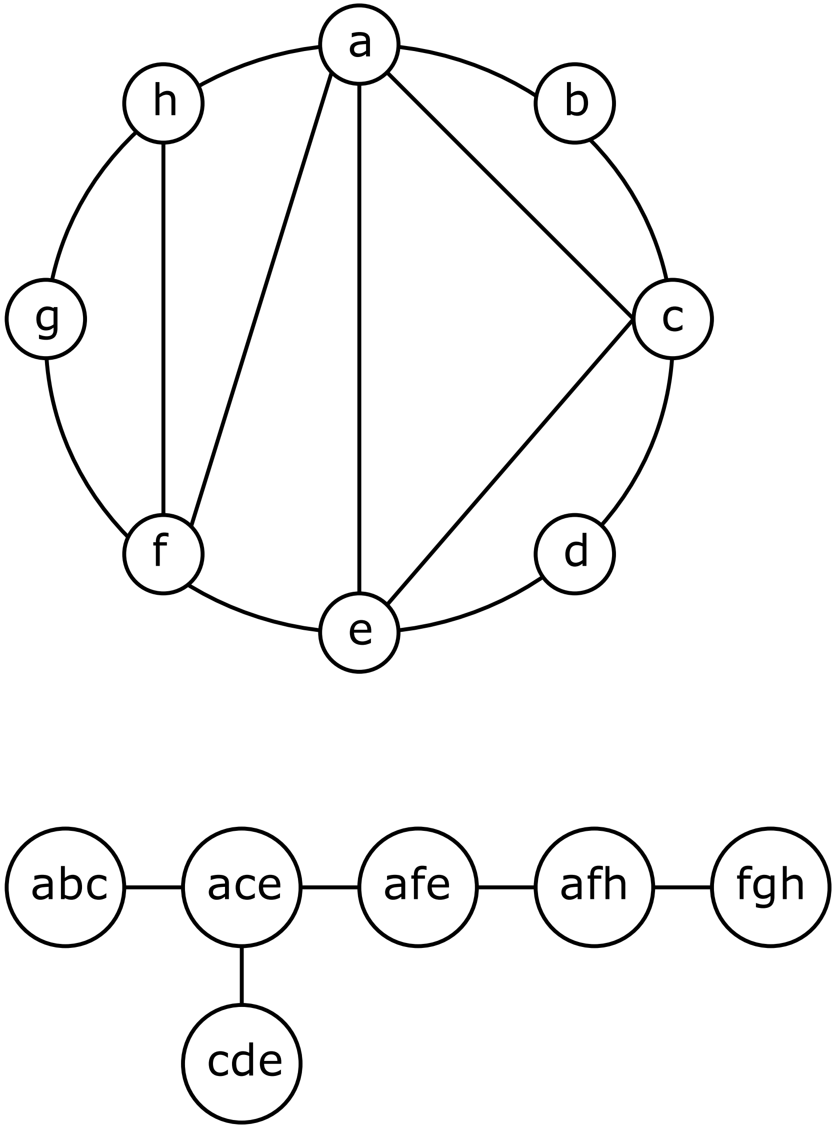
Examples

- The set $\{K_5, K_{3,3}\}$ is an obstruction set of the minor closed class of planar graphs.
- The set $\{K_3\}$ is the obstruction of the class of forests.
- The obstruction set of graphs of treewidth 2 is $\{K_4\}$

Complexity

Class	Treewidth	Pathwidth
Bounded degree	NP-C	NP-C
Bipartite	NP-C	NP-C
Trees / Forests	Linear	P
Outerplanar	Linear	P
Planar	Open	NP-C
Permutation	P	P
Interval	P	P
Circular arc	P	Open

Outerplanar Graphs



Approximations

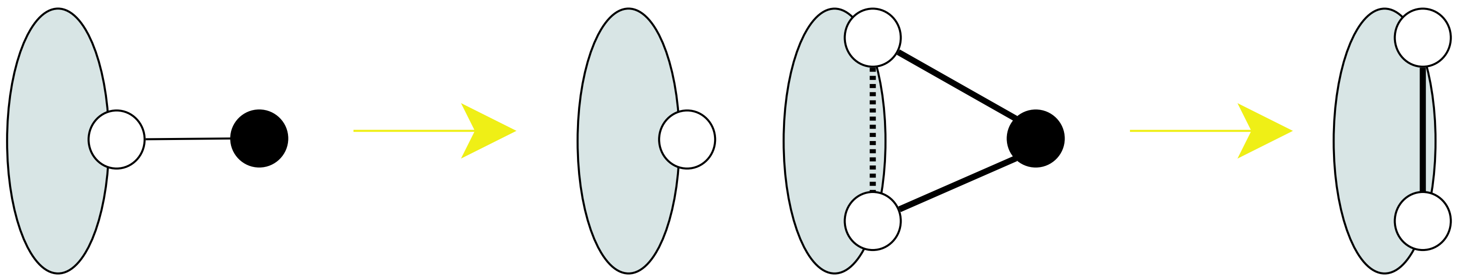
- Arbitrary Graphs: $O(\log n)$ ratio.
- It is an open problem if it is in APX.
- Planar Graphs: 1.5 ratio (exact solution of branchwidth).

Fixed Parameter

- The parameter k is fixed and the problem is to decide, given a graph G, if G has a treewidth $\leq k$ (decision) and to construct it if it exists (construction).
- Decision, Construction problems solvable in linear time:
  1. Either decide that treewidth $> k$ or find a tree decomposition of bounded but not optimal width. [Bodlaender and Kloks]
  2. Find the optimal tree decomposition. [Bodlaender]

Fixed Parameter

- There is a hidden constant factor that is at least exponential in k (impractical)
- For $k = 2, 3, 4$ linear time algorithms exist, based on graph reduction (practical).
- Example: For $k = 2$, by repeating the following operations in G if we conclude to a single vertex the treewidth is at most 2.



Why treewidth?

- When we have a tree decomposition of a graph G with width k bounded by a constant we are able to obtain polynomial-time (and often linear-time) algorithms for many NP-hard problems.
- Of course, this algorithm is usually exponential (or worse) on k, and therefore very often impractical.

Nice tree decompositions

In order to design algorithms for NP-hard problems it is useful to construct nice tree decompositions.

A nice tree decomposition is:

- a rooted binary tree
- it has four types of nodes

Nice tree decompositions

- Leaf nodes: leaves of the decomposition.
  $|X_i| = 1$.
- Introduce nodes: nodes with a single child j.
  $X_i = X_j \cup \{v\}$.
- Forget nodes: nodes with a single child j.
  $X_i = X_j - \{v\}$.
- Join nodes: nodes with two children j_1, j_2.
  $X_i = X_{j_1} = X_{j_2}$

Solving NP-Complete Problems

For graphs with bounded treewidth at most k

- Find a tree decomposition of G with width at most k.
- Transform it into a nice tree decomposition of width at most k.
- Compute for every node i a certain table. To compute a table we use the children tables and the type of i (leaf, forget, introduce, join).
- The answer of the problem can be found by inspecting the table of the root.

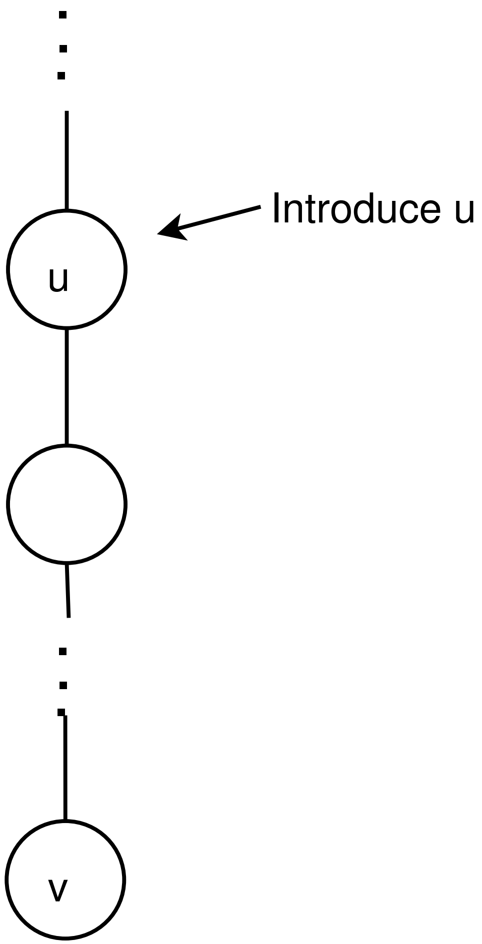
3-coloring

- For leaves: Fill their tables with all three possible colors of the corresponding vertex.
- For introduce nodes: Extend the child's table by coloring the new vertex u with a color that is not used by u's neighbors in the bag.
- For forget nodes: Take the child's table and "forget" the information regarding the forgotten vertex.
- For join nodes: Take the intersection of the children's tables, that is the entries that agree on the colors of the vertices of the bag.

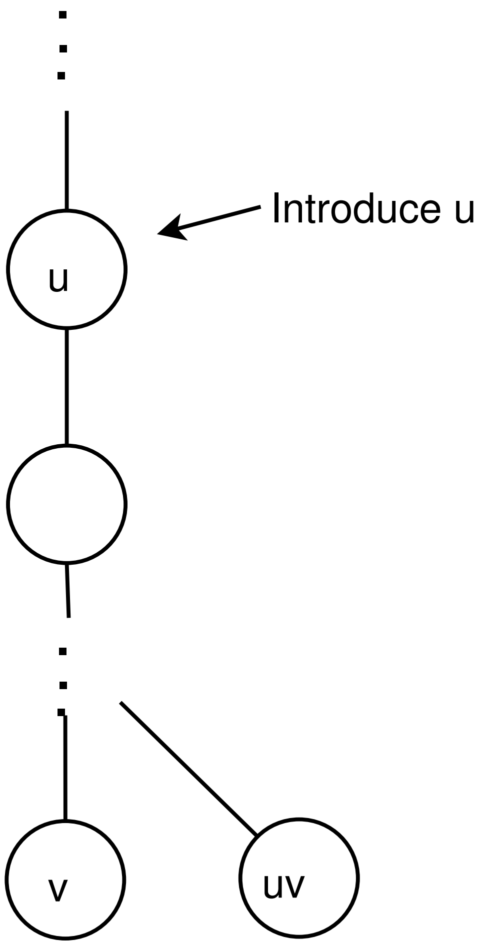
Why this works

- For leaf, forget and join nodes simple.
- For a vertex u that is added at an introduce node, we know that:
  1. It does not conflict with the other vertices in the same node.
  2. It has no edge with any vertex v further down the subtree. (because if it did, u, v would have to be in the same bag somewhere, violating connectedness for u or v).

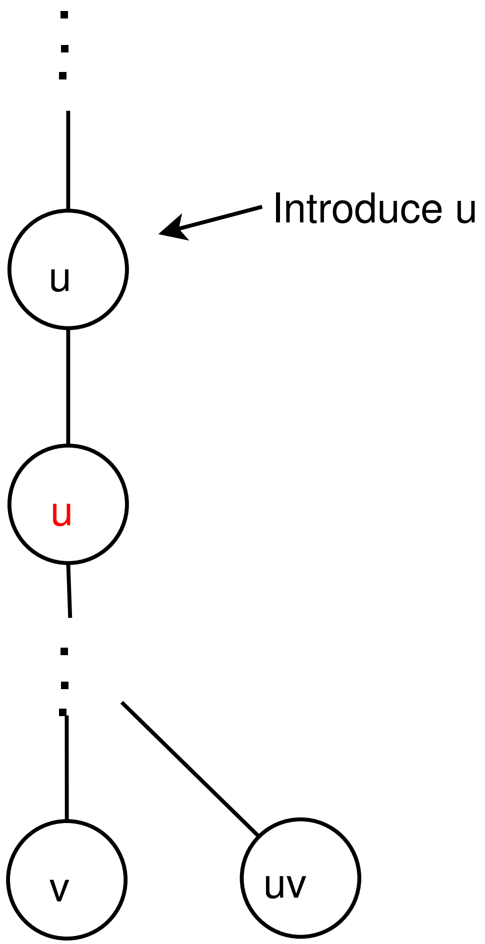
Why this works



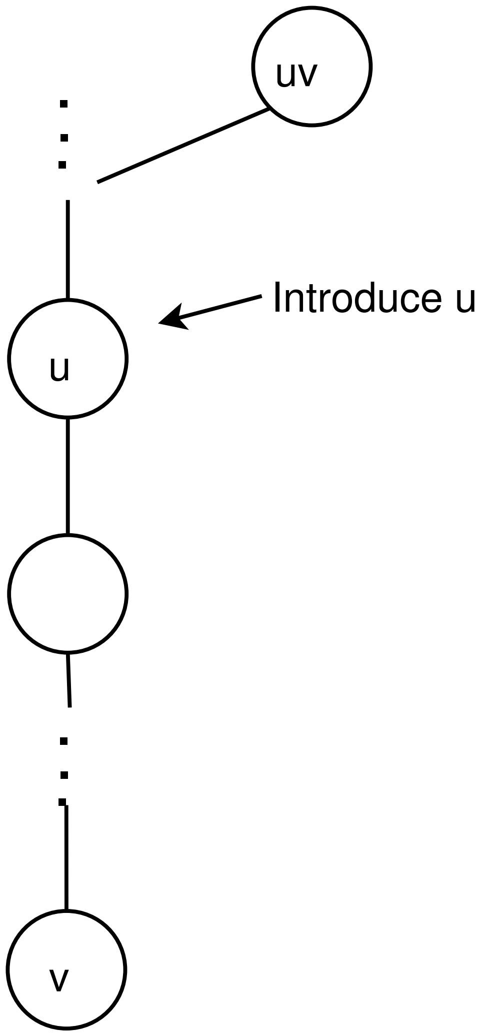
Why this works



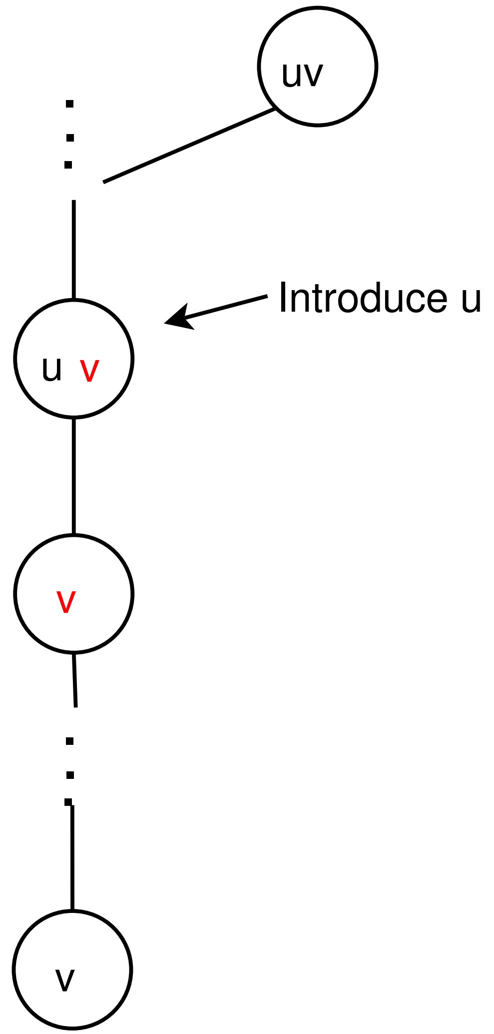
Why this works



Why this works



Why this works



- So, for completing the table of every node i it suffices to look inside the bag X_i and the bags of its children.
- The final answer is "yes" iff the table of the root is nonempty.
- Every table has constant size at most 3^{k+1} (because k is a constant).
- The nice tree decomposition has linear size.
- Thus we can solve 3-coloring in polynomial time!

Terminal Graphs

- A Terminal graph is a triple $H = (V, E, X)$, where (V, E) is a graph and $X \subseteq V$ ordered, called the *terminals*
- A terminal graph with l terminals is called *l-terminal*
- If H, H' are l-terminal graphs then $G = H \oplus H'$ is a graph obtained by taking the union of H, H' and identifying the i^{th} terminal of H to the i^{th} of H', $\forall i, 1 \leq i \leq l$.
- H is called a terminal subgraph of G.

General Method

To every node i of the nice tree decomposition we associate the terminal graph $G_i = (V_i, E_i, X_i)$, where

- V_i is the set of all the vertices of V that are in the bags in the subtree of i,
- (V_i, E_i) is the subgraph of G induced by the set V_i
- and X_i the bag of node i.
- G_i is a terminal subgraph of G.

General Method

Define notions of

- solution
- partial solution for terminal graphs
- extension of partial solution
- characteristic of a partial solution

Define an algorithm that computes the full set of characteristics for each node type.

General Method

In the 3-coloring example

- solution $\longrightarrow$ a 3-coloring of G
- partial solution $\longrightarrow$ a 3-coloring of G_i
- extension of partial solution $\longrightarrow$ (obvious)
- characteristic $\longrightarrow$ the colors of the terminal vertices

Monadic Second Order Logic

- ***Definition:*** Monadic Second Order Logic is a language to describe graph properties using the following constructions:
  - quantifications over vertices, edges, sets of vertices, sets of edges
  - membership tests
  - adjacency tests
  - logic operations
- *Each problem that can be stated in monadic second order logic can be solved in linear time on graph of bounded treewidth.*

Example

For example the problem of 3-coloring of a graph $G(V, E)$ can be stated as:

$$\exists W_1 \subseteq V : \exists W_2 \subseteq V : \exists W_3 \subseteq V : \forall v \in V :$$

$$(v \in W_1 \vee v \in W_2 \vee v \in W_3) \wedge \forall v \in V : \forall w \in V : (v, w) \in E$$

$$\Rightarrow (\neg(v \in W_1 \wedge w \in W_1) \wedge \neg(v \in W_2 \wedge w \in W_2)$$

$$\wedge \neg(v \in W_3 \wedge w \in W_3))$$

THE END!!!