Trees and the Total Coloring Game with Complete Defect

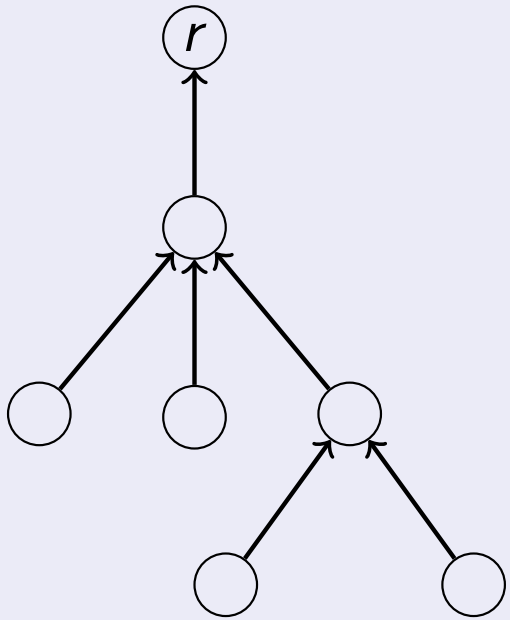
Prairie Wentworth-Nice

Swarthmore College

4 August 2017

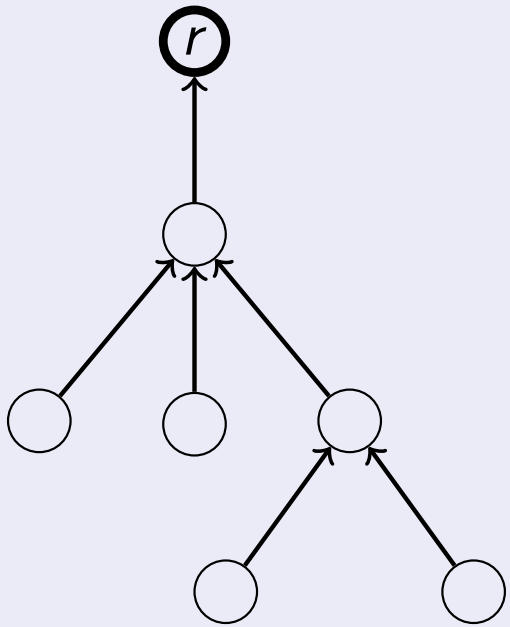
Trees

- A tree is a connected graph with no cycles.
- When a tree is rooted, a partial ordering of the edges and vertices is introduced.



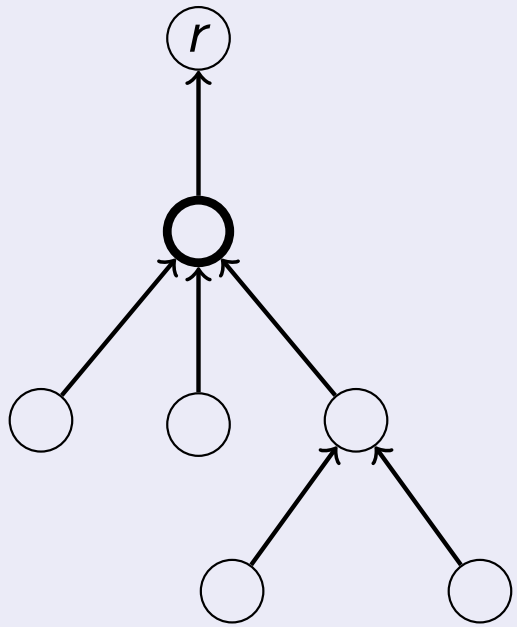
Trees

- A tree is a connected graph with no cycles.
- When a tree is rooted, a partial ordering of the edges and vertices is introduced.



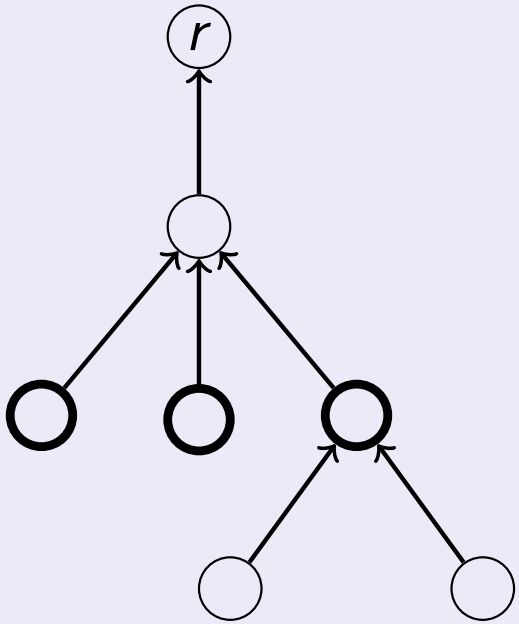
Trees

- A tree is a connected graph with no cycles.
- When a tree is rooted, a partial ordering of the edges and vertices is introduced.



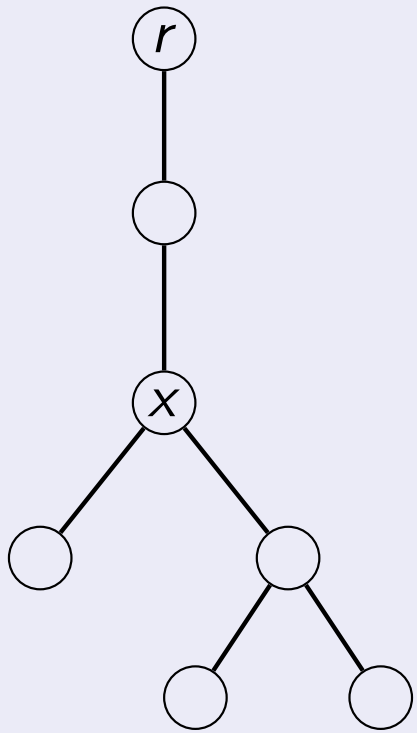
Trees

- A tree is a connected graph with no cycles.
- When a tree is rooted, a partial ordering of the edges and vertices is introduced.



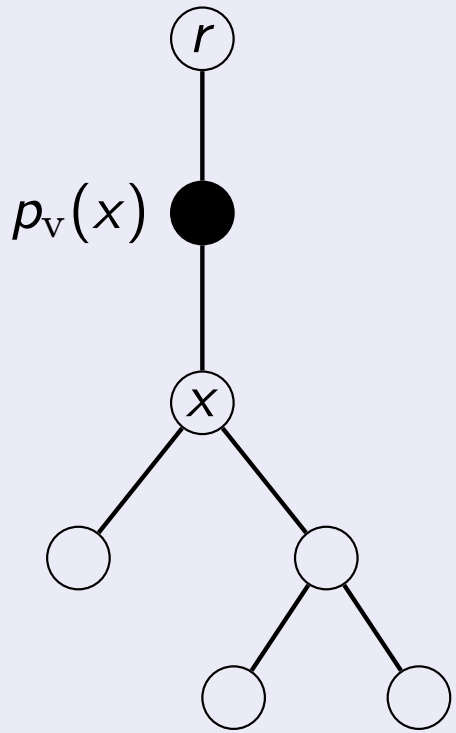
A Family Tree

- The parent vertex and parent edge of an element x, are the unique vertex and edge neighboring x that are closer to the root if they exist.
- The child vertices and child edges of an element x are the vertices and edges neighboring x that are further from the root if they exist.



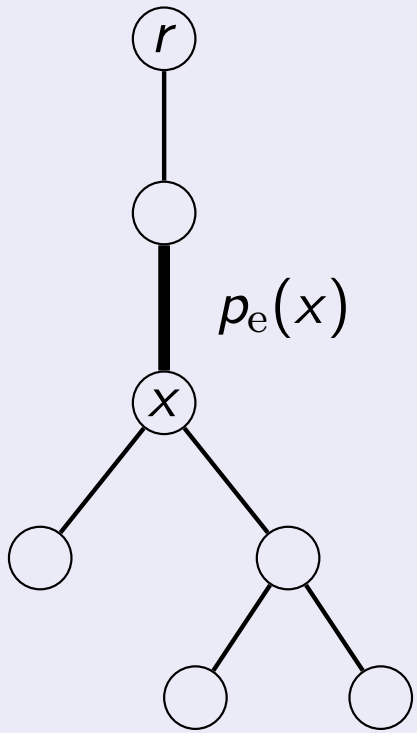
A Family Tree

- The parent vertex and parent edge of an element x, are the unique vertex and edge neighboring x that are closer to the root if they exist.
- The child vertices and child edges of an element x are the vertices and edges neighboring x that are further from the root if they exist.



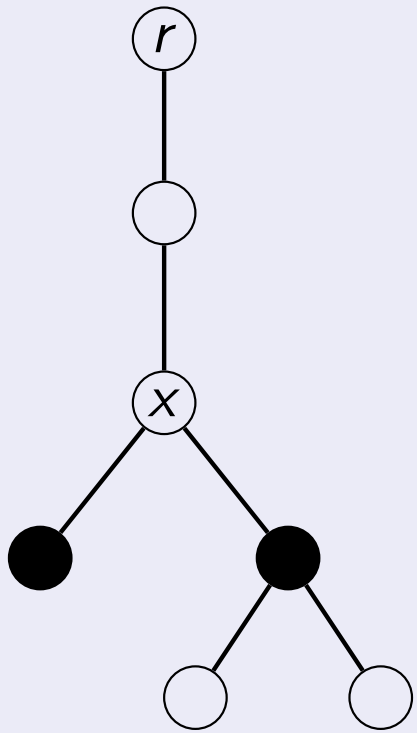
A Family Tree

- The parent vertex and parent edge of an element x, are the unique vertex and edge neighboring x that are closer to the root if they exist.
- The child vertices and child edges of an element x are the vertices and edges neighboring x that are further from the root if they exist.



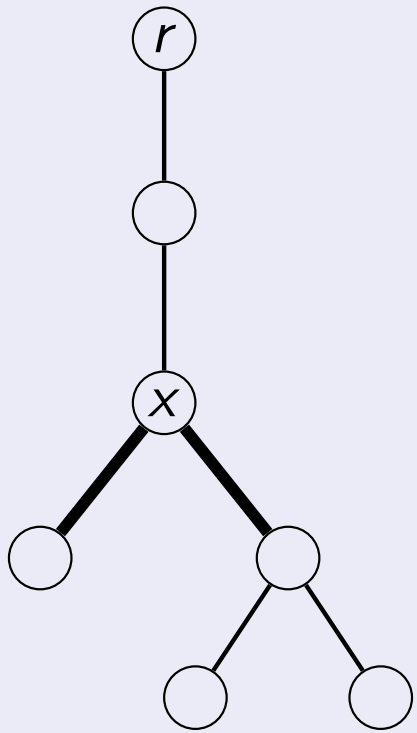
A Family Tree

- The parent vertex and parent edge of an element x, are the unique vertex and edge neighboring x that are closer to the root if they exist.
- The child vertices and child edges of an element x are the vertices and edges neighboring x that are further from the root if they exist.



A Family Tree

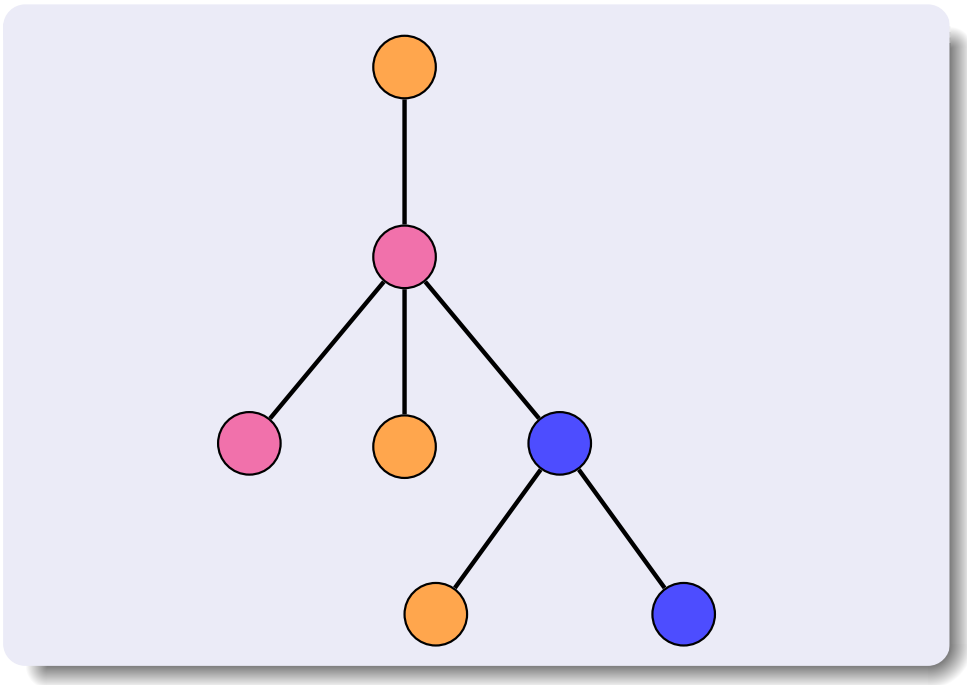
- The parent vertex and parent edge of an element x, are the unique vertex and edge neighboring x that are closer to the root if they exist.
- The child vertices and child edges of an element x are the vertices and edges neighboring x that are further from the root if they exist.



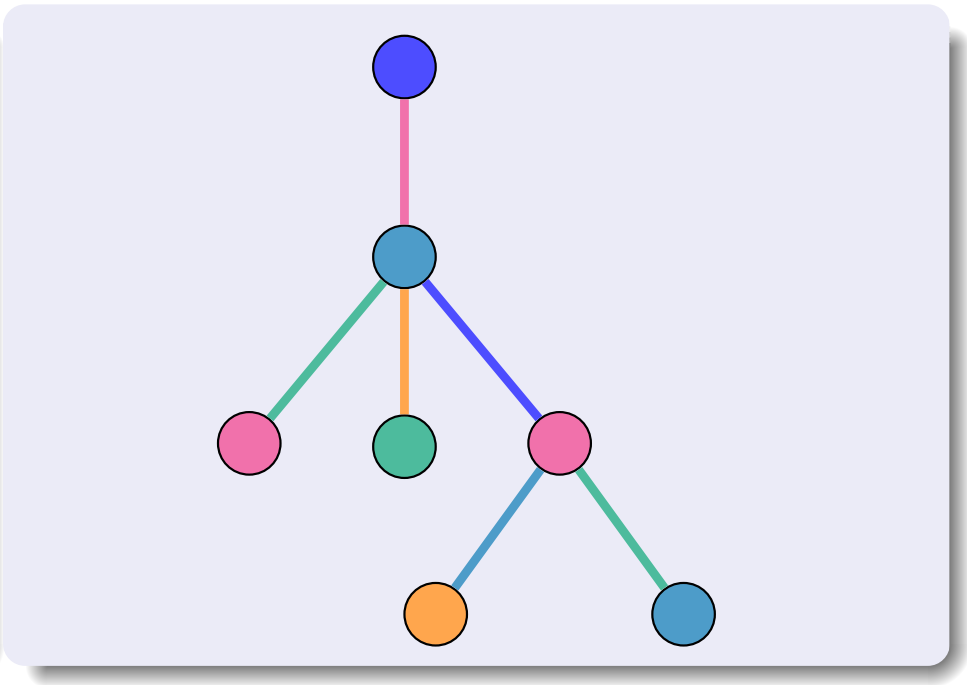
Coloring Games on Graphs

- Vertices
- Edges
- Variations

Defect Coloring



Total Coloring



Complete Defect

- The complete defect of an element x, denoted $\text{def}(x)$, is the total number of elements of the same color that are adjacent or incident to it.

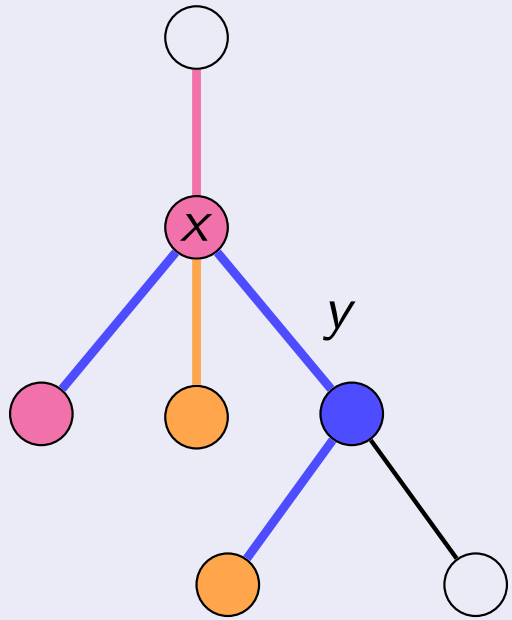


Figure: At this point, $\text{def}(x) = 2$, and $\text{def}(y) = 3$.

1-Relaxed Total Coloring Game

- Alice and Bob choose a graph G and a set of colors X.
- Alice takes the first turn and colors any element in G.
- Alice and Bob then take turns coloring the remaining uncolored elements of G.
- A color $\alpha \in X$ is legal for an element x if x has at most one neighbor, y, colored α, and $\text{def}(y) = 0$.

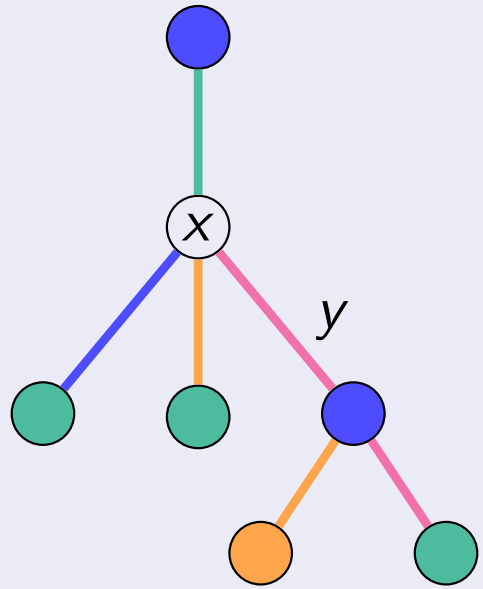


Figure: The only legal color for x from {orange, blue, pink, green}, is orange.

1-Relaxed Total Coloring Game

- If at some point there are no legal colors for an element, Bob wins.
- If G is completely colored at the end of the game, then Alice wins.
- The complete d-relaxed total game chromatic number, ${}^{d}\chi''_{g}(G)$, is the fewest number of colors for which Alice can always win on G with defect d.

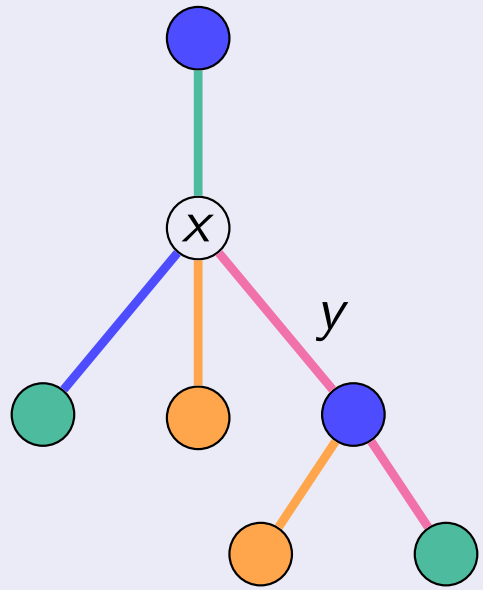
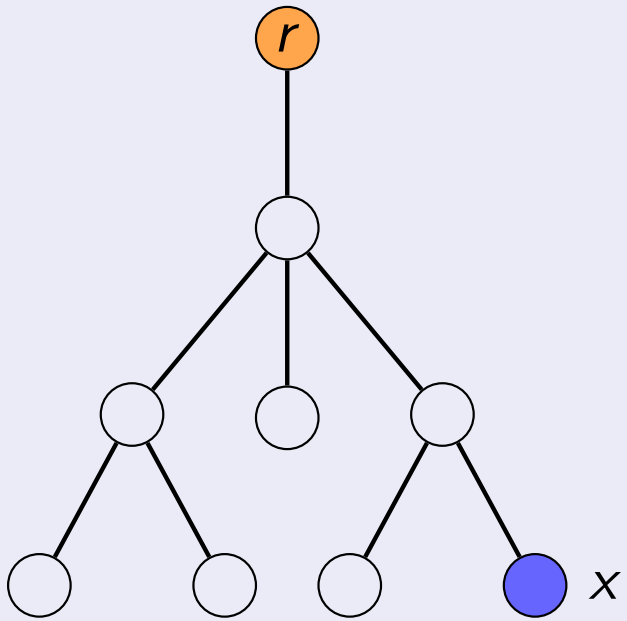


Figure: There is no legal color for x from the set {orange, blue, pink, green}.

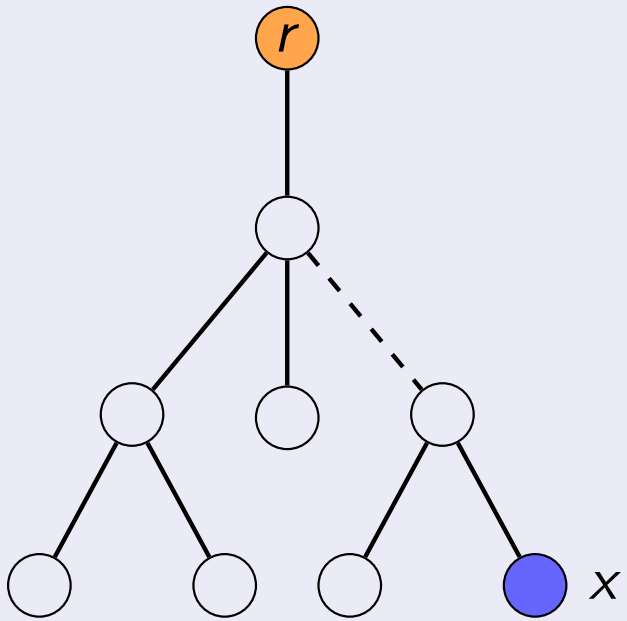
Alice's Strategy

- Turn 1: Alice roots the tree and colors the root.
- After Bob colors an element x: Alice uses an activation strategy to choose an element to color and picks a color for it.



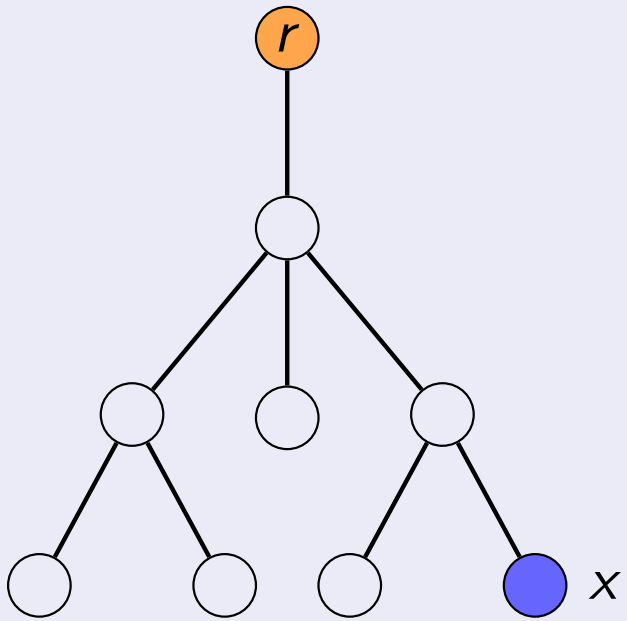
Alice's Strategy

- Turn 1: Alice roots the tree and colors the root.
- After Bob colors an element x: Alice uses an activation strategy to choose an element to color and picks a color for it.



Alice's Strategy

- Turn 1: Alice roots the tree and colors the root.
- After Bob colors an element x: Alice uses an activation strategy to choose an element to color and picks a color for it.



Search Stage

- Initial Step:
  - If $p_e(x)$ is inactive, set $g := p_e(x)$ and enter the recursive step.
  - If $p_e(x)$ is uncolored and activated, set $g := p_e(x)$ and enter the coloring stage.
  - If x is an edge, $p_e(x)$ is colored and $p_v(x)$ is uncolored, set $g := p_v(x)$ and enter the coloring stage.
  - If x is a vertex, $p_e(x)$ is colored, and $p_e(p_e(x))$ is colored, set $g := p_v(x)$.
  - Else if x is a vertex, $p_e(x)$ is colored, and $p_e(p_e(x))$ is uncolored, but there is a color neighboring $p_e(p_e(x))$ that is a legal color for $p_v(x)$, set $g := p_v(x)$.
  - Otherwise, let g be any uncolored element with no uncolored vertex or edge parents and enter the coloring stage.

Search Stage

- Initial Step:
  - **If $p_e(x)$ is inactive, set $g := p_e(x)$ and enter the recursive step.**
  - **If $p_e(x)$ is uncolored and activated, set $g := p_e(x)$ and enter the coloring stage.**
  - **If x is an edge, $p_e(x)$ is colored and $p_v(x)$ is uncolored, set $g := p_v(x)$ and enter the coloring stage.**
  - If x is a vertex, $p_e(x)$ is colored, and $p_e(p_e(x))$ is colored, set $g := p_v(x)$.
  - Else if x is a vertex, $p_e(x)$ is colored, and $p_e(p_e(x))$ is uncolored, but there is a color neighboring $p_e(p_e(x))$ that is a legal color for $p_v(x)$, set $g := p_v(x)$.
  - Otherwise, let g be any uncolored element with no uncolored vertex or edge parents and enter the coloring stage.

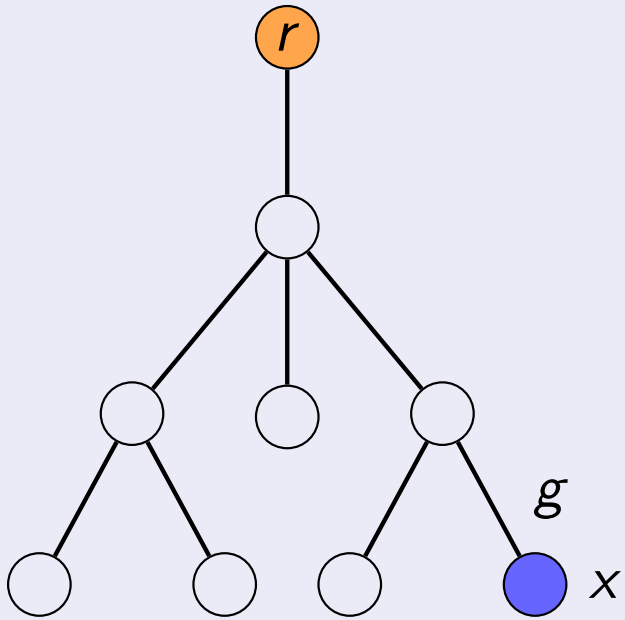
Search Stage

- Initial Step:
  - If $p_e(x)$ is inactive, set $g := p_e(x)$ and enter the recursive step.
  - If $p_e(x)$ is uncolored and activated, set $g := p_e(x)$ and enter the coloring stage.
  - If x is an edge, $p_e(x)$ is colored and $p_v(x)$ is uncolored, set $g := p_v(x)$ and enter the coloring stage.
  - **If x is a vertex, $p_e(x)$ is colored, and $p_e(p_e(x))$ is colored, set $g := p_v(x)$.**
  - **Else if x is a vertex, $p_e(x)$ is colored, and $p_e(p_e(x))$ is uncolored, but there is a color neighboring $p_e(p_e(x))$ that is a legal color for $p_v(x)$, set $g := p_v(x)$.**
  - Otherwise, let g be any uncolored element with no uncolored vertex or edge parents and enter the coloring stage.

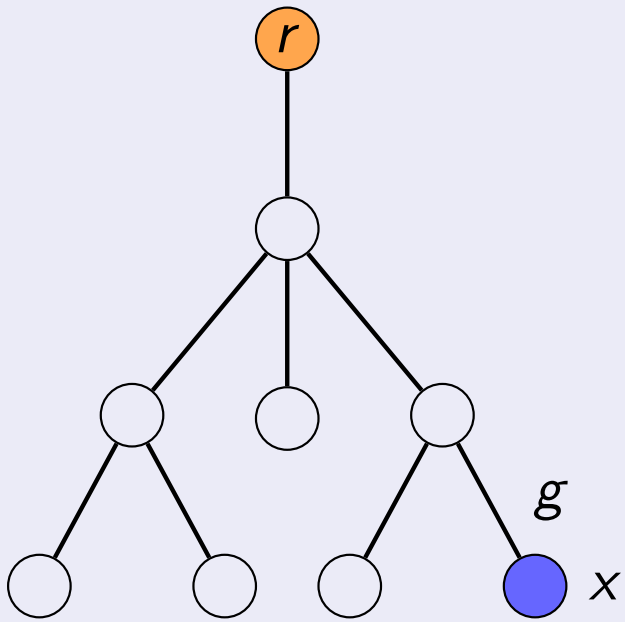
Search Stage

- Initial Step:
  - If $p_e(x)$ is inactive, set $g := p_e(x)$ and enter the recursive step.
  - If $p_e(x)$ is uncolored and activated, set $g := p_e(x)$ and enter the coloring stage.
  - If x is an edge, $p_e(x)$ is colored and $p_v(x)$ is uncolored, set $g := p_v(x)$ and enter the coloring stage.
  - If x is a vertex, $p_e(x)$ is colored, and $p_e(p_e(x))$ is colored, set $g := p_v(x)$.
  - Else if x is a vertex, $p_e(x)$ is colored, and $p_e(p_e(x))$ is uncolored, but there is a color neighboring $p_e(p_e(x))$ that is a legal color for $p_v(x)$, set $g := p_v(x)$.
  - **Otherwise, let g be any uncolored element with no uncolored vertex or edge parents and enter the coloring stage.**

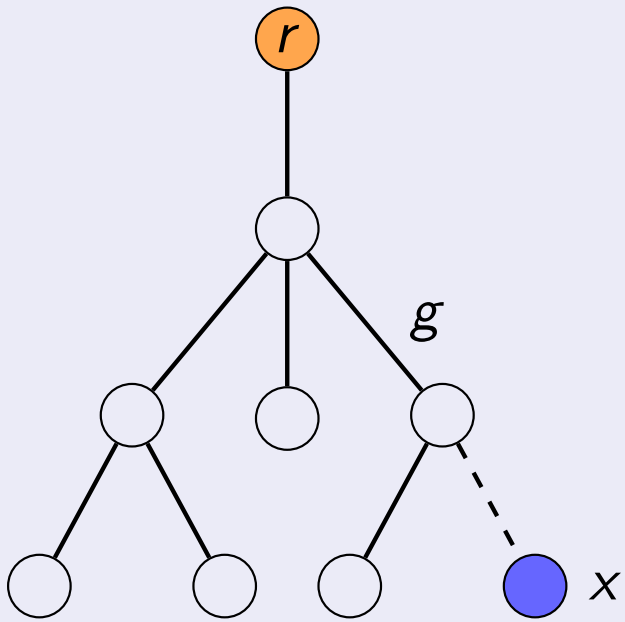
- Initial Step
  - If $p_e(x)$ is inactive, set $g := p_e(x)$ and enter the recursive step.



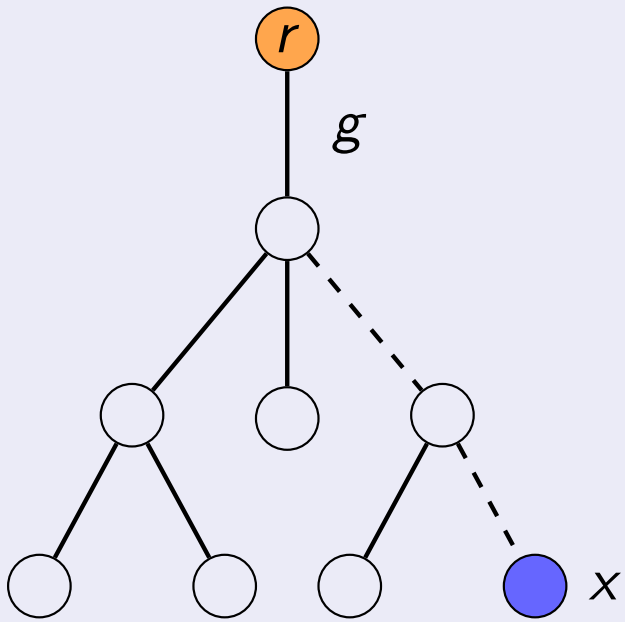
- Recursive Step:
  - **If $p_e(g)$ is inactive, activate g, set $g := p_e(g)$ and repeat the recursive step.**
  - If $p_e(g)$ is active and uncolored, activate g if it is not already active, set $g := p_e(g)$ and move to the coloring step
  - Else, if $p_v(g)$ uncolored, activate g, set $g := p_v(g)$, move to the coloring step.
  - Otherwise, enter the coloring stage.



- Recursive Step:
  - **If $p_e(g)$ is inactive, activate g, set $g := p_e(g)$ and repeat the recursive step.**
  - If $p_e(g)$ is active and uncolored, activate g if it is not already active, set $g := p_e(g)$ and move to the coloring step
  - Else, if $p_v(g)$ uncolored, activate g, set $g := p_v(g)$, move to the coloring step.
  - Otherwise, enter the coloring stage.



- Recursive Step:
  - If $p_e(g)$ is inactive, activate g, set $g := p_e(g)$ and repeat the recursive step.
  - If $p_e(g)$ is active and uncolored, activate g if it is not already active, set $g := p_e(g)$ and move to the coloring step
  - Else, if $p_v(g)$ uncolored, activate g, set $g := p_v(g)$, move to the coloring step.
  - **Otherwise, enter the coloring stage.**



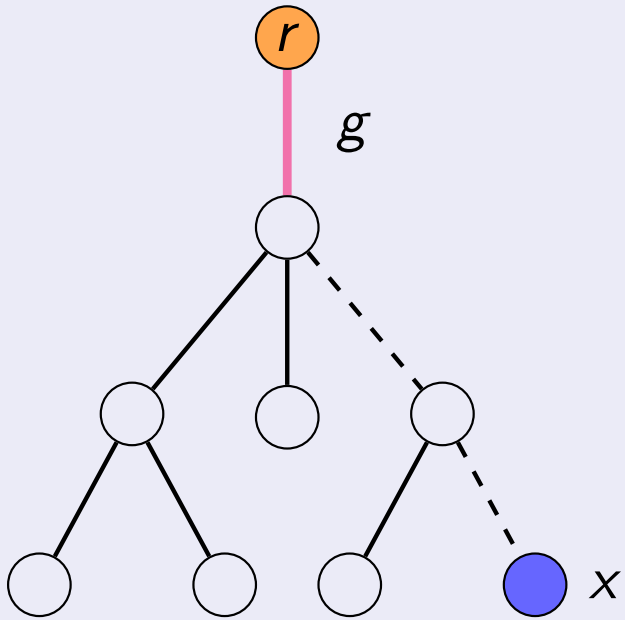
Coloring Stage

- Coloring Step:
  - If $p_e(g)$ is uncolored, then color g a color of one of the neighbors of $p_e(g)$.
  - If g is a vertex with one colored descendant, $p_v(g)$ is uncolored, and $p_e(p_v(g))$ is colored, color $p_v(g)$.
  - If g is an edge, $p_v(p_v(g))$ is uncolored with 3 of its children distinctly colored, $p_e(g)$ is colored and there is a legal color for $p_v(g)$ that is neighboring g and $p_v(p_v(g))$, then color $p_v(g)$ that color.
  - If g is a vertex, $p_v(p_v(g))$ is uncolored with three children distinctly colored, g has only one child colored and there is a legal color for $p_v(g)$ neighboring $p_v(p_v(g))$, then color $p_v(g)$ that color.
  - If $p_v(g)$ is uncolored, color g the same color as an already colored neighbor of $p_v(g)$ if possible.
  - Else, if $p_v(p_v(g))$ is uncolored, color g the same color as an already colored neighbor of $p_v(p_v(g))$ if possible.
  - Color g with any legal color.

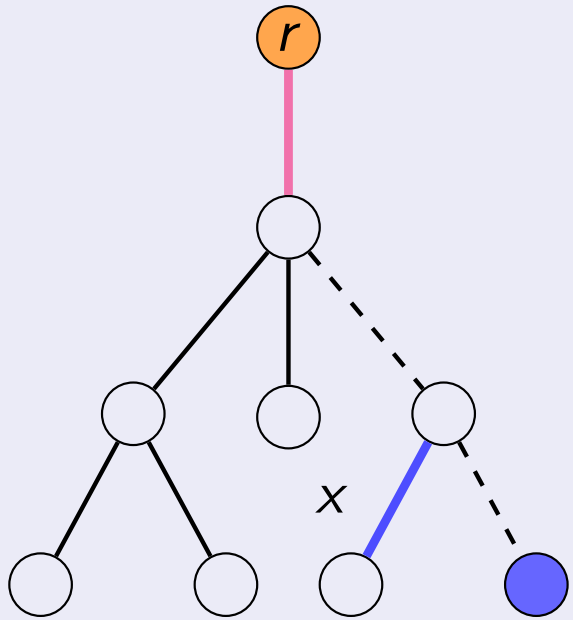
- Coloring Step:
  - **If $p_e(g)$ is uncolored, then color g a color of one of the neighbors of $p_e(g)$.**
  - **If g is a vertex with one colored descendant, $p_v(g)$ is uncolored, and $p_e(p_v(g))$ is colored, color $p_v(g)$.**
  - **If g is an edge, $p_v(p_v(g))$ is uncolored with 3 of its children distinctly colored, $p_e(g)$ is colored and there is a legal color for $p_v(g)$ that is neighboring g and $p_v(p_v(g))$, then color $p_v(g)$ that color.**
  - **If g is a vertex, $p_v(p_v(g))$ is uncolored with three children distinctly colored, g has only one child colored and there is a legal color for $p_v(g)$ neighboring $p_v(p_v(g))$, then color $p_v(g)$ that color.**
  - If $p_v(g)$ is uncolored, color g the same color as an already colored neighbor of $p_v(g)$ if possible.
  - **Else, if $p_v(p_v(g))$ is uncolored, color g the same color as an already colored neighbor of $p_v(p_v(g))$ if possible.**
  - Color g with any legal color.

- Coloring Step:
  - If $p_e(g)$ is uncolored, then color g a color of one of the neighbors of $p_e(g)$.
  - If g is a vertex with one colored descendant, $p_v(g)$ is uncolored, and $p_e(p_v(g))$ is colored, color $p_v(g)$.
  - If g is an edge, $p_v(p_v(g))$ is uncolored with 3 of its children distinctly colored, $p_e(g)$ is colored and there is a legal color for $p_v(g)$ that is neighboring g and $p_v(p_v(g))$, then color $p_v(g)$ that color.
  - If g is a vertex, $p_v(p_v(g))$ is uncolored with three children distinctly colored, g has only one child colored and there is a legal color for $p_v(g)$ neighboring $p_v(p_v(g))$, then color $p_v(g)$ that color.
  - **If $p_v(g)$ is uncolored, color g the same color as an already colored neighbor of $p_v(g)$ if possible.**
  - Else, if $p_v(p_v(g))$ is uncolored, color g the same color as an already colored neighbor of $p_v(p_v(g))$ if possible.
  - **Color g with any legal color.**

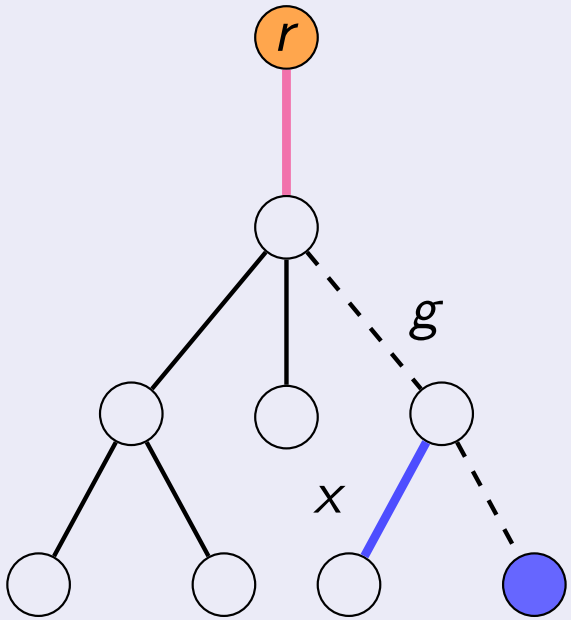
- Coloring Stage:
  - If $p_v(g)$ is uncolored, color g the same color as an already colored neighbor of $p_v(g)$ if possible.
  - **Color g with any legal color.**



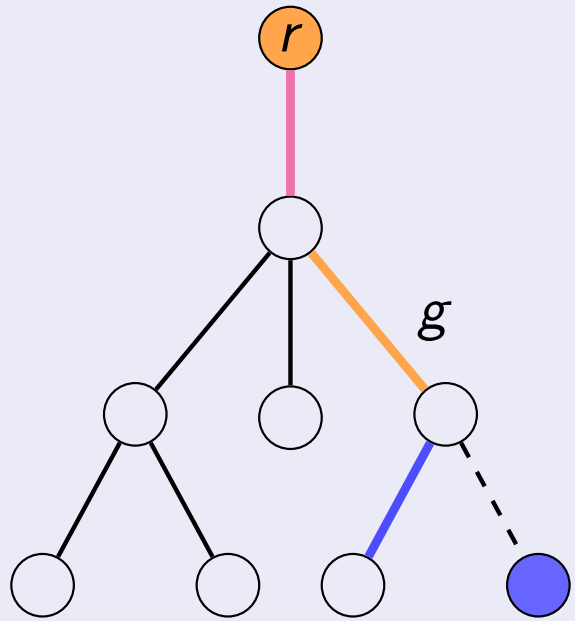
- Initial step:
  - If $p_e(x)$ is uncolored and activated, set $g := p_e(x)$ and enter the coloring stage.



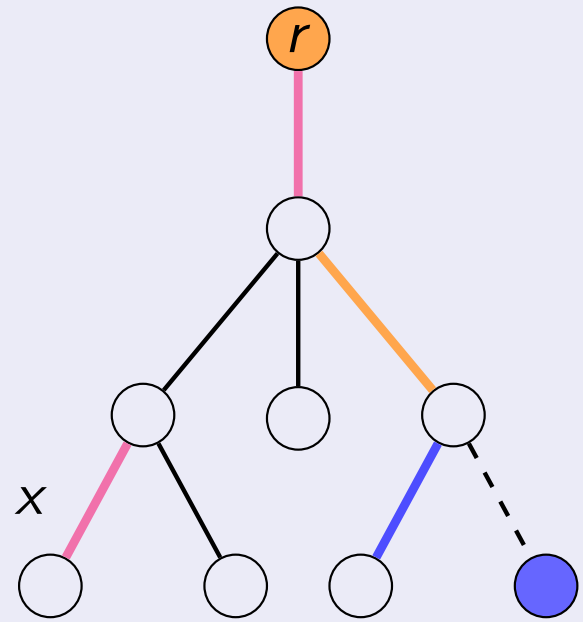
- Initial step:
  - If $p_e(x)$ is uncolored and activated, set $g := p_e(x)$ and enter the coloring stage.



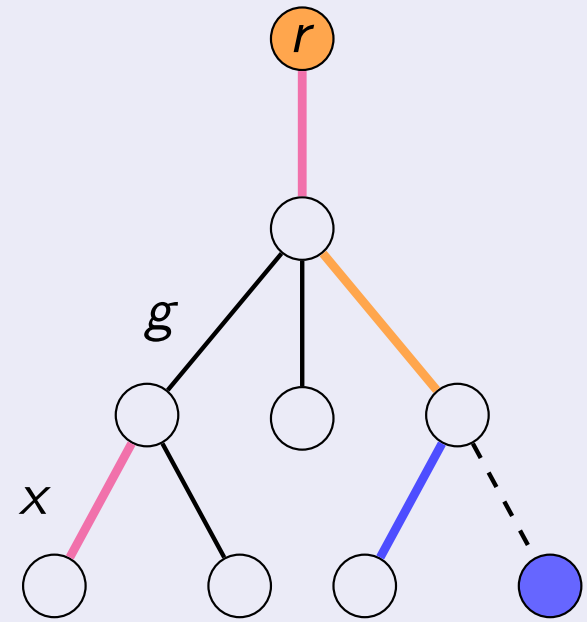
- Coloring stage:
  - **If $p_v(g)$ is uncolored, color g the same color as an already colored neighbor of $p_v(g)$ if possible.**
  - Color g with any legal color.



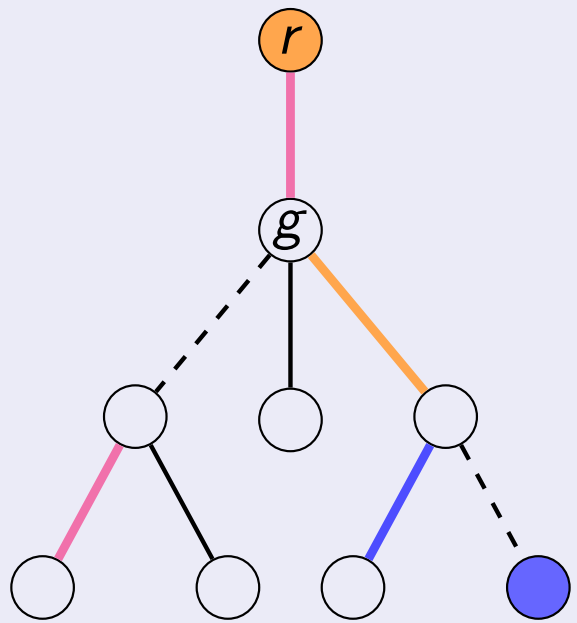
- Initial step:
  - If $p_e(x)$ is inactive, set $g := p_e(x)$ and enter the recursive step.



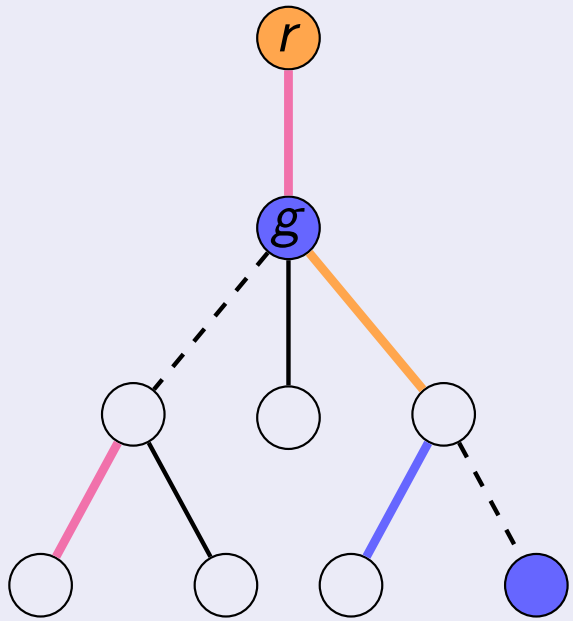
- Initial step:
  - If $p_e(x)$ is inactive, set $g := p_e(x)$ and enter the recursive step.



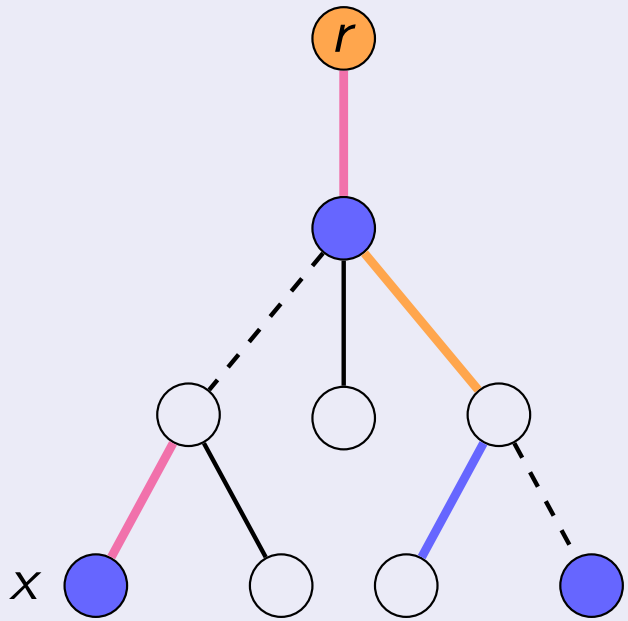
- Recursive Step:
  - If $p_e(g)$ is inactive, activate g, set $g := p_e(g)$ and repeat the recursive step.
  - If $p_e(g)$ is active and uncolored, activate g if it is not already active, set $g := p_e(g)$ and move to the coloring step
  - **Else, if $p_v(g)$ uncolored, activate g, set $g := p_v(g)$, move to the coloring step.**
  - Otherwise, enter the coloring stage.



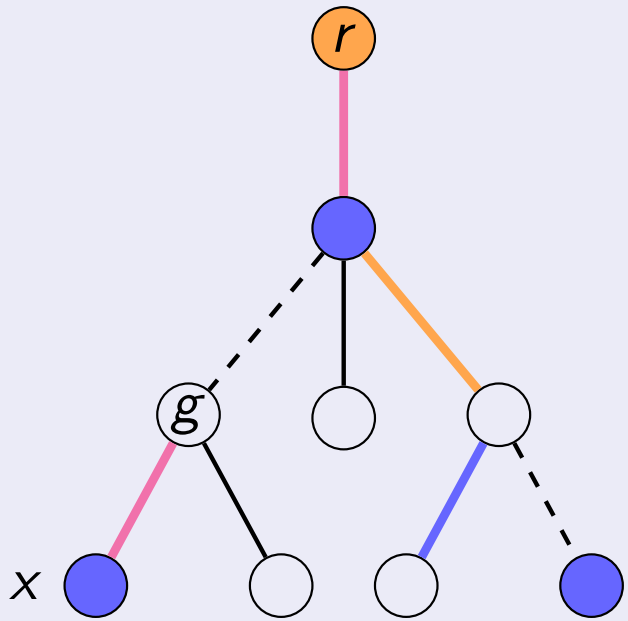
- Coloring Stage:
  - If $p_v(g)$ is uncolored, color g the same color as an already colored neighbor of $p_v(g)$ if possible.
  - **Color g with any legal color.**



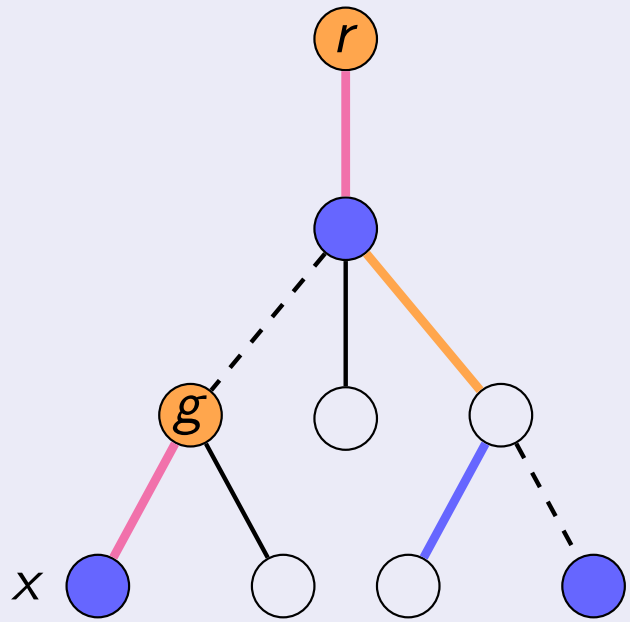
- Initial Step:
  - If x is a vertex, $p_e(x)$ is colored, and $p_e(p_e(x))$ is uncolored, but there is a color neighboring $p_e(p_e(x))$ that is a legal color for $p_v(x)$, set $g := p_v(x)$ and move to the coloring stage.



- Initial Step:
  - If x is a vertex, $p_e(x)$ is colored, and $p_e(p_e(x))$ is uncolored, but there is a color neighboring $p_e(p_e(x))$ that is a legal color for $p_v(x)$, set $g := p_v(x)$ and move to the coloring stage.



- Coloring Step:
  - If $p_e(g)$ is uncolored, then color g a color of one of the neighbors of $p_e(g)$.



Theorem

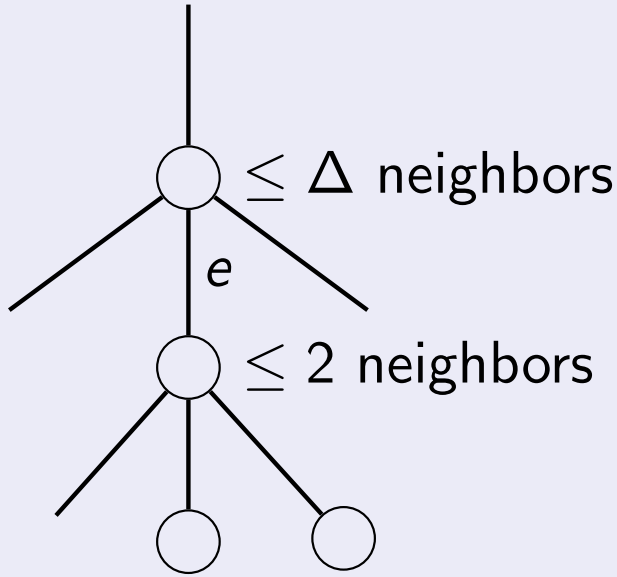
If T is a tree with $\Delta(T) = \Delta$, then ${}^{1}\chi_{g}''(T) \leq \max\{\Delta + 2, 6\}$.

Proof:

- Show that using Alice's strategy:
  - Any edge can be colored when playing with at least $\Delta + 2$ colors.
  - Any vertex can be colored when playing with at least 6 colors.

Edges

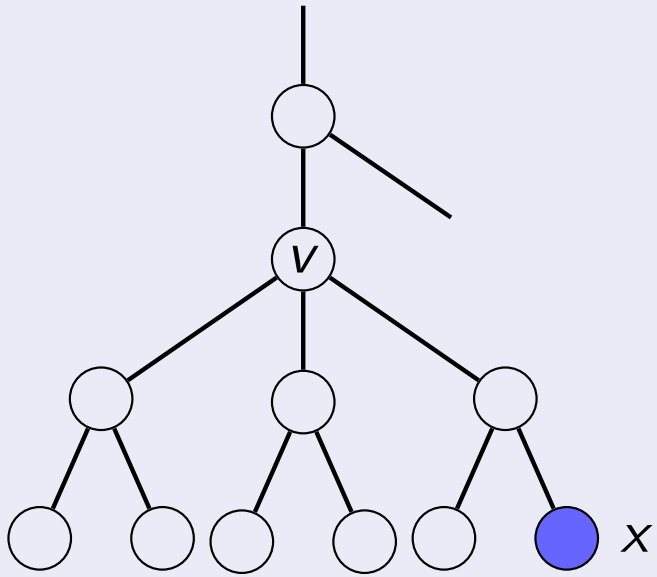
Consider the uncolored edge e:



- There are at most Δ distinctly colored neighbors of e that are not its children.
- If Bob colors two children, Alice will color e.
- If Alice colors a child of e, she will do so safely.

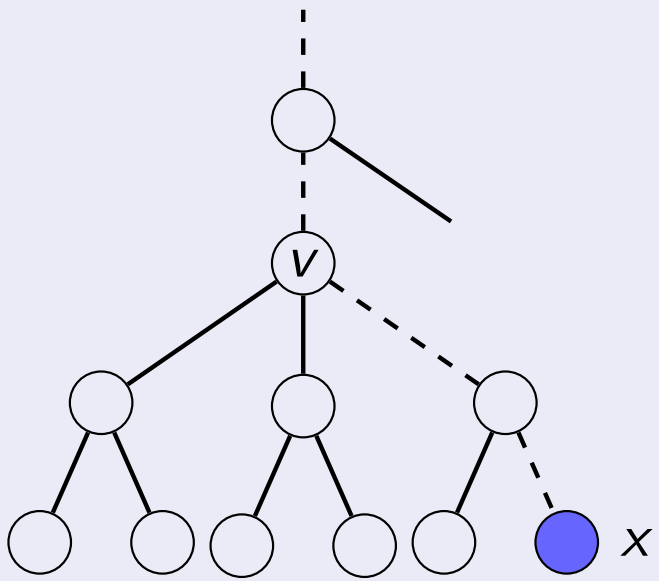
Vertices

Claim: At most 5 children of an uncolored vertex v can be colored before Alice will be prompted to color v.



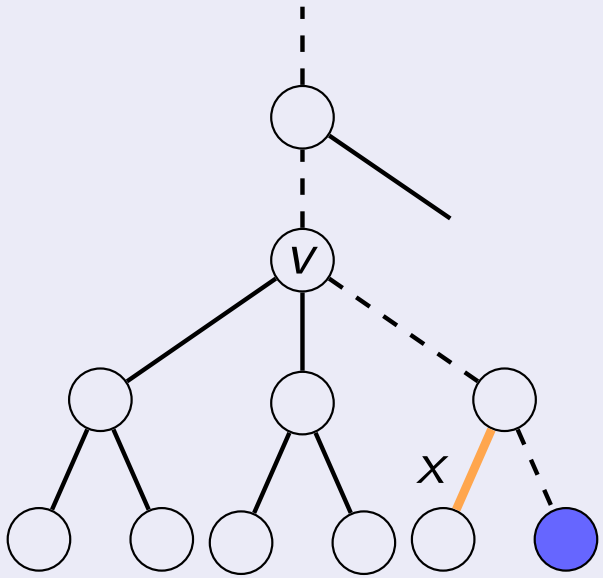
Vertices

Claim: At most 5 children of an uncolored vertex v can be colored before Alice will be prompted to color v.



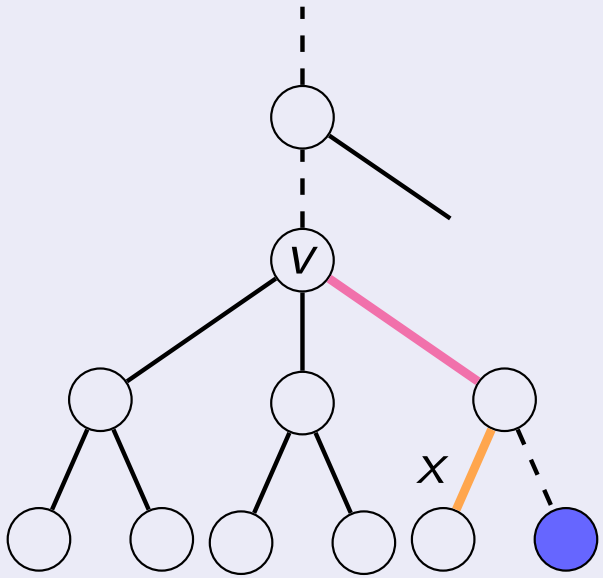
Vertices

Claim: At most 5 children of an uncolored vertex v can be colored before Alice will be prompted to color v.



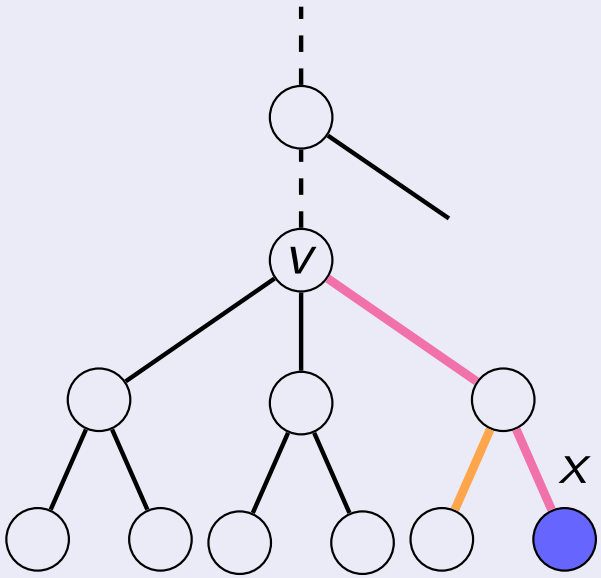
Vertices

Claim: At most 5 children of an uncolored vertex v can be colored before Alice will be prompted to color v.



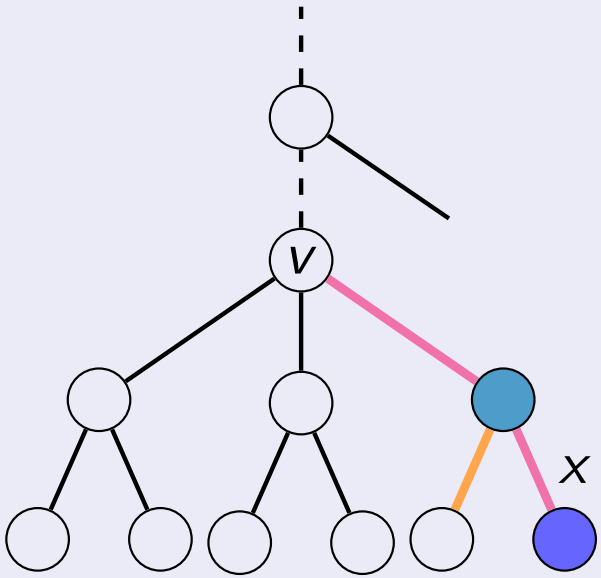
Vertices

Claim: At most 5 children of an uncolored vertex v can be colored before Alice will be prompted to color v.



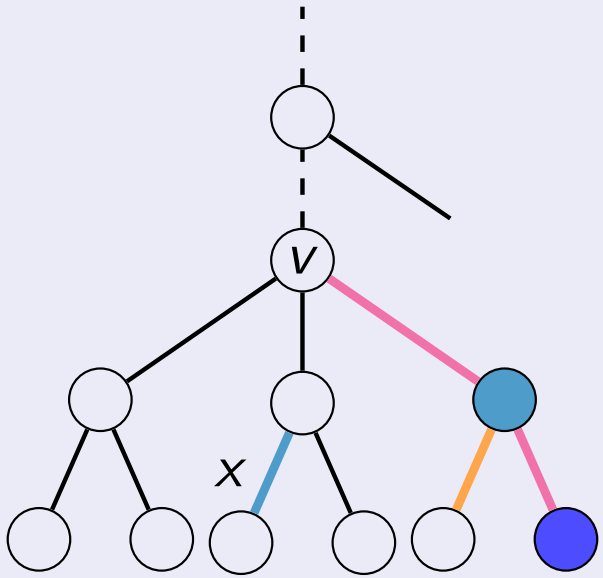
Vertices

Claim: At most 5 children of an uncolored vertex v can be colored before Alice will be prompted to color v.



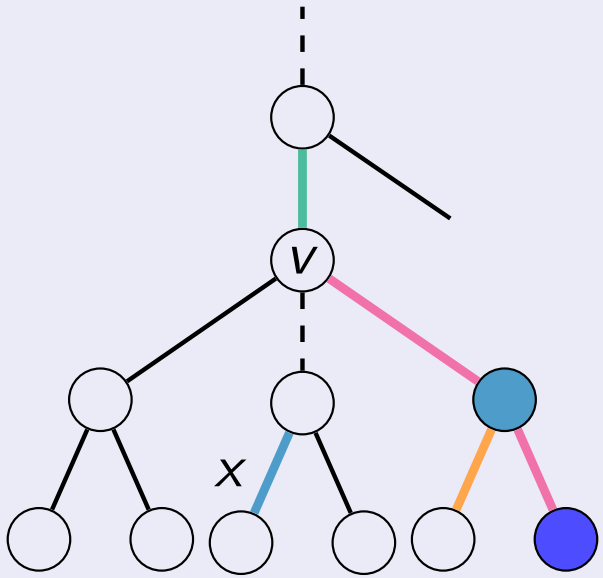
Vertices

Claim: At most 5 children of an uncolored vertex v can be colored before Alice will be prompted to color v.



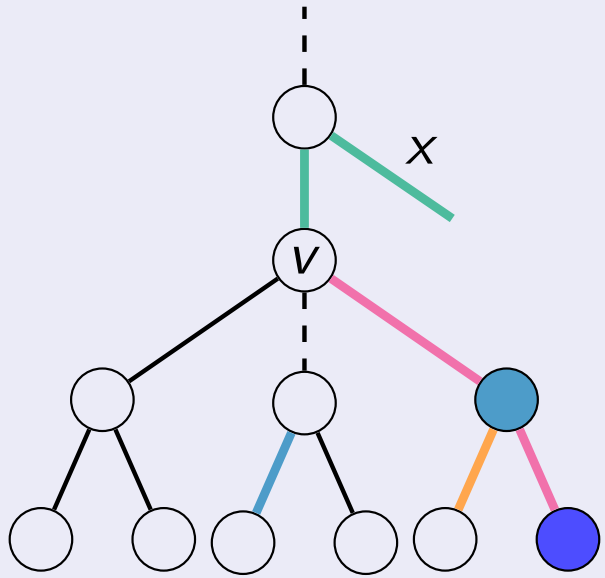
Vertices

Claim: At most 5 children of an uncolored vertex v can be colored before Alice will be prompted to color v.



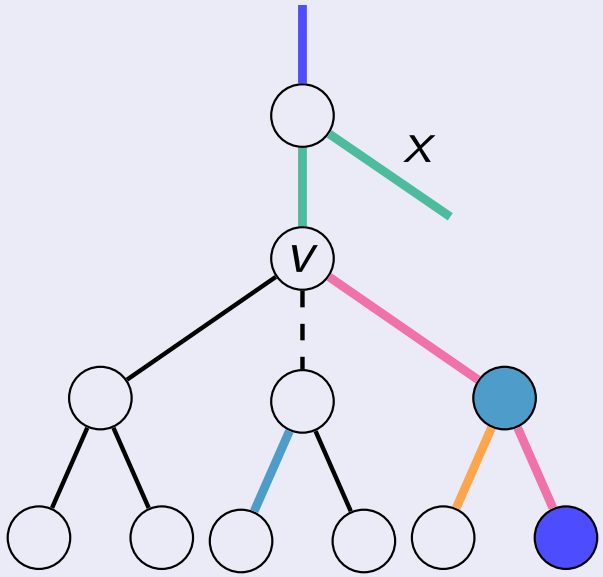
Vertices

Claim: At most 5 children of an uncolored vertex v can be colored before Alice will be prompted to color v.



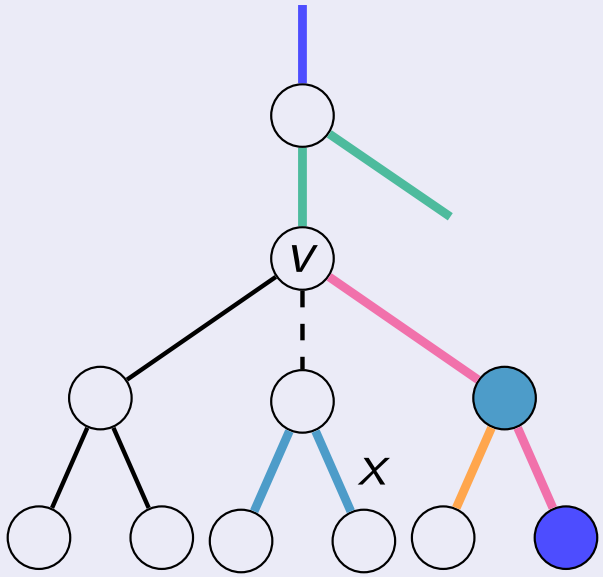
Vertices

Claim: At most 5 children of an uncolored vertex v can be colored before Alice will be prompted to color v.



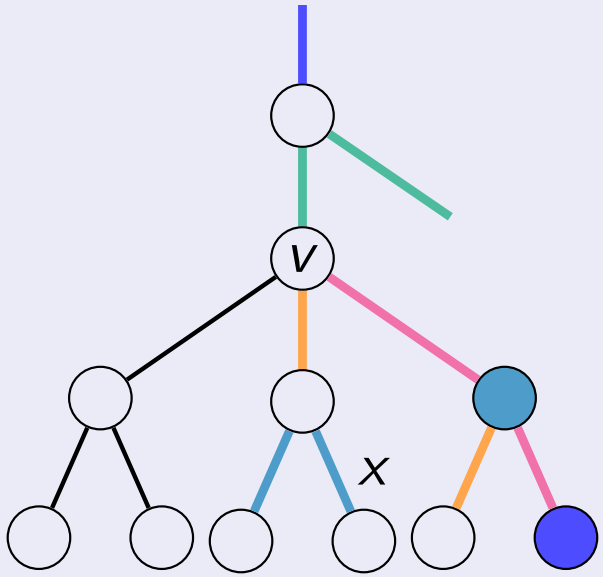
Vertices

Claim: At most 5 children of an uncolored vertex v can be colored before Alice will be prompted to color v.



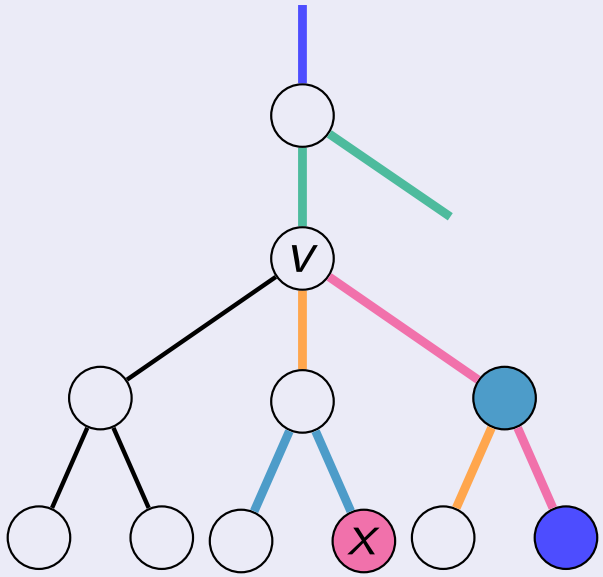
Vertices

Claim: At most 5 children of an uncolored vertex v can be colored before Alice will be prompted to color v.



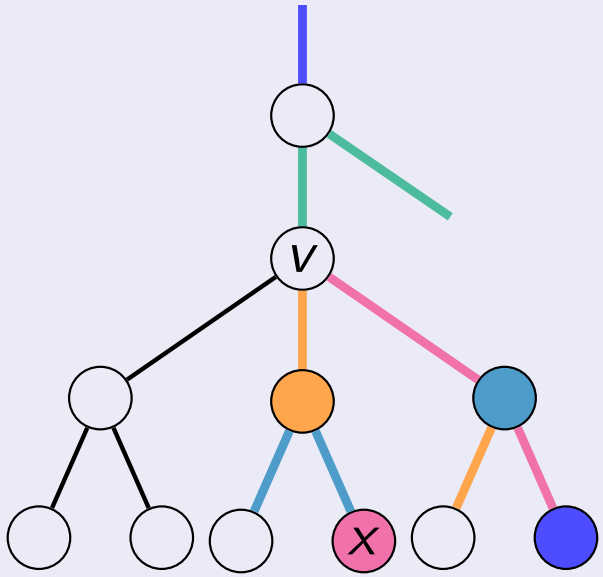
Vertices

Claim: At most 5 children of an uncolored vertex v can be colored before Alice will be prompted to color v.



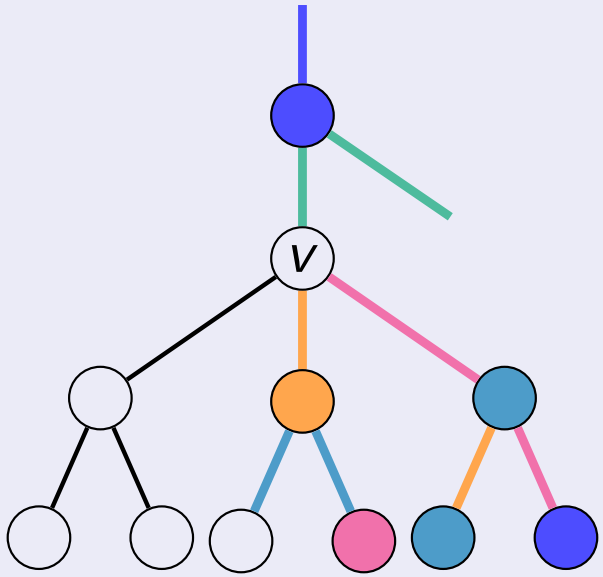
Vertices

Claim: At most 5 children of an uncolored vertex v can be colored before Alice will be prompted to color v.



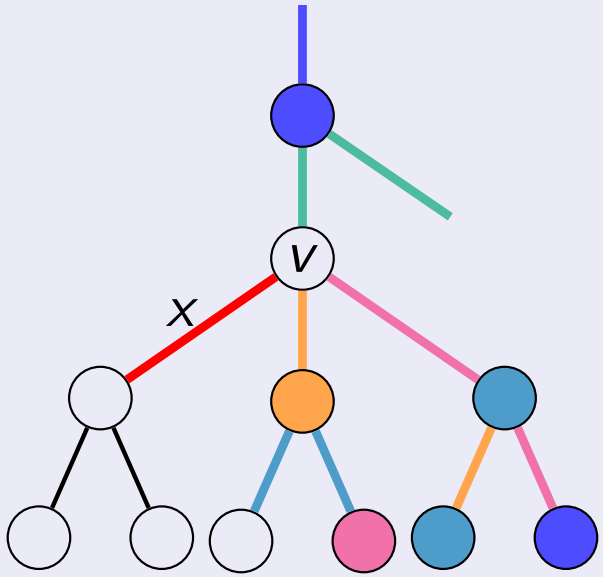
Vertices

Claim: At most 5 children of an uncolored vertex v can be colored before Alice will be prompted to color v.



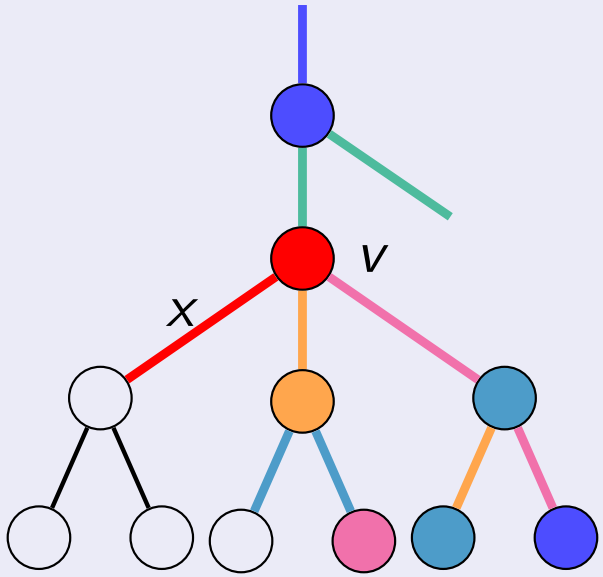
Vertices

Claim: At most 5 children of an uncolored vertex v can be colored before Alice will be prompted to color v.



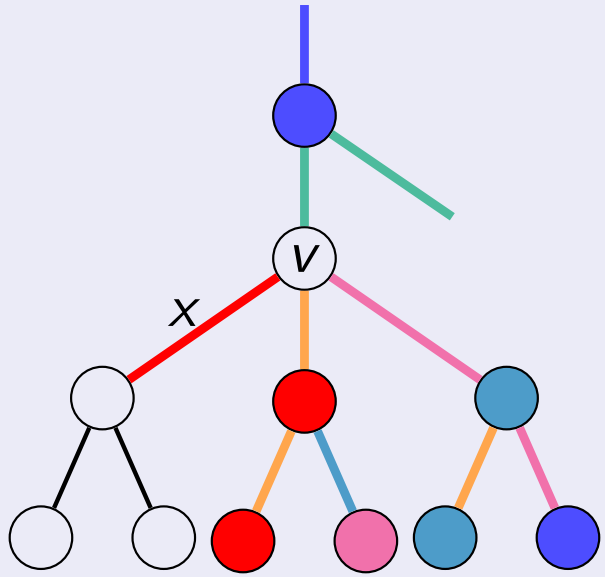
Vertices

Claim: At most 5 children of an uncolored vertex v can be colored before Alice will be prompted to color v.



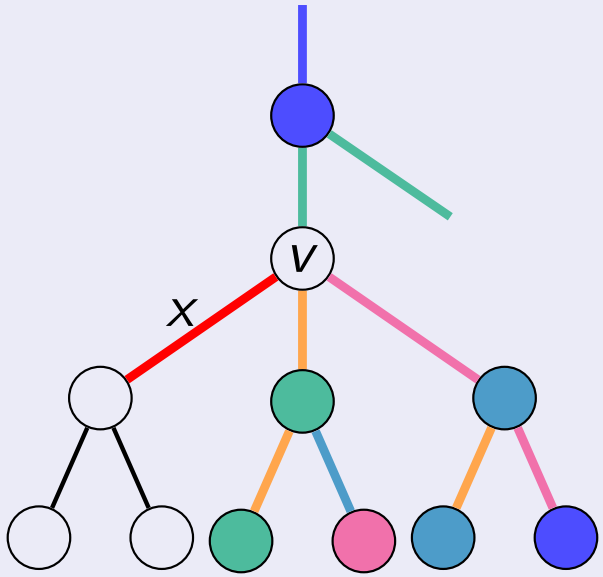
Vertices

These 7 neighbors must be colored at most 6 distinct colors before Alice will be prompted to color v. And the color of x must be distinct.



Vertices

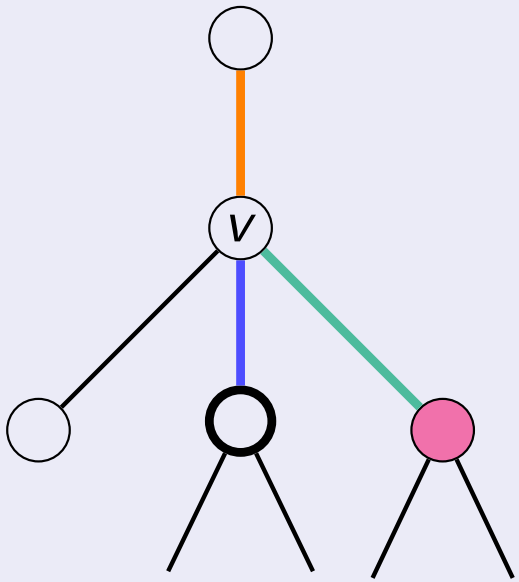
These 7 neighbors must be colored at most 6 distinct colors before Alice will be prompted to color v. And the color of x must be distinct.



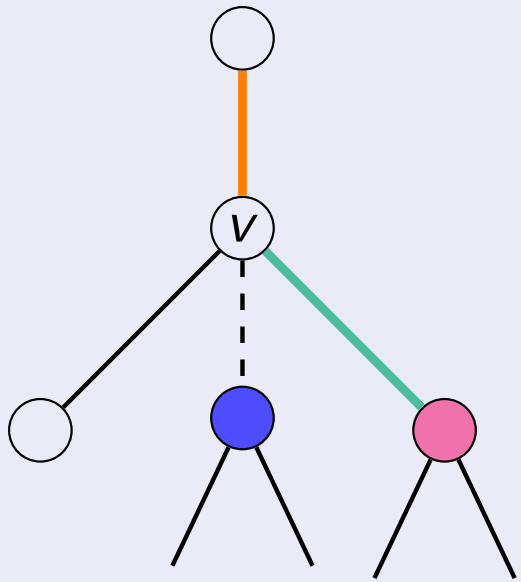
Vertices

3 children of v are colored in 2 branches

Case 1: A colored edge-vertex pair and a colored edge.



Case 2: A colored edge-vertex pair and a colored vertex.

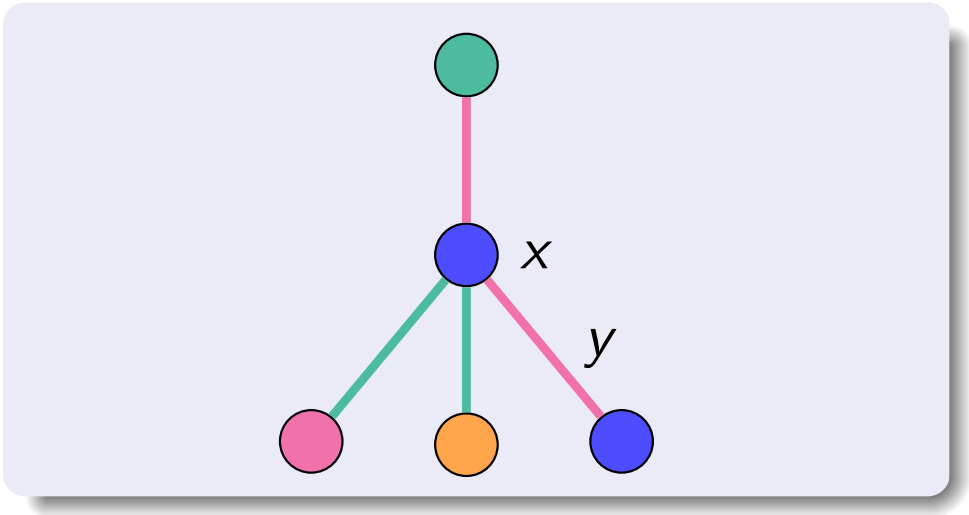


- If Alice and Bob play the 1-relaxed total coloring game on a tree T with $\Delta(T) = \Delta$ using $r = \max\{\Delta + 2, 6\}$ colors:
  - Any uncolored edge can be legally colored because $r \geq \Delta + 2$.
  - Any uncolored vertex can be legally colored because $r \geq 6$.
- Thus Alice will always win on T with r colors, and ${}^1\chi_g''(T) \leq \max\{\Delta + 2, 6\}$.

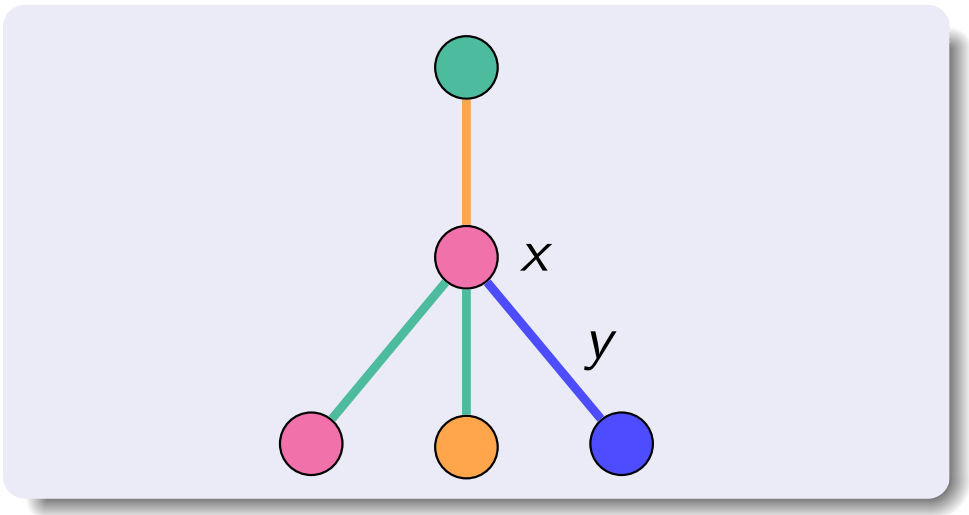
Future Work

- Other defect definitions
  - Separate $(\Delta + 3)$
  - Only edge-vertex
  - Only vertices
  - Only edges
- $\text{def}(x) > 1$
  - Defect $\longrightarrow$ color bound
  - Color bound $\longrightarrow$ defect

Separate Defect:



Complete Defect:



Acknowledgements

- Dr. Chuck Dunn
- Joshua Schroeder
- James Zak
- Dr. Josh Laison
- Dr. Erin McNicholas
- Willamette University
- NSF DMS 1460982

Thank You!

Questions?