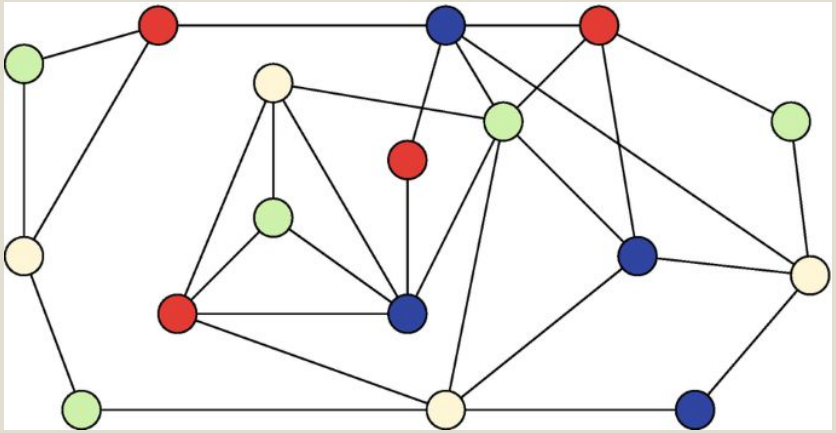
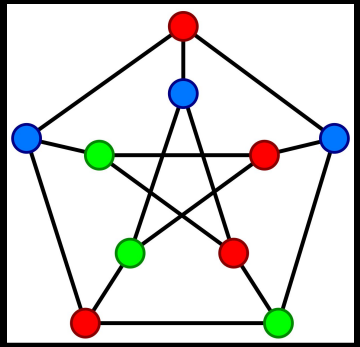
MIN GRAPH COLORING

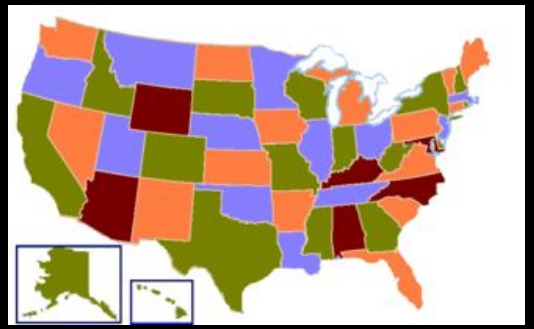


CS412 Colin Gregory, Austin Perdue, Madeline Burns

Min Graph Coloring Intro

- Problem
  - How can we color elements in a undirected graph so that **no two adjacent vertices have the same color?**
- Applications
  - Map Coloring
  - Register Allocation
  - Traffic Management
  - Event Planning





Decision vs. Optimization

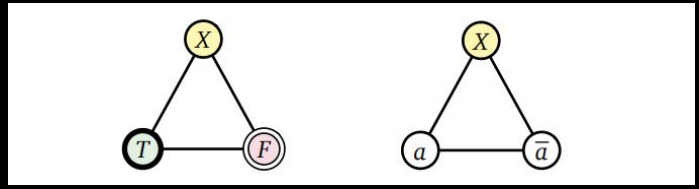
- Decision
  - Is it possible to color a graph with at most (*K*) colors?
  - “Yes” can be verified in polynomial time
  - **NP-Complete**
- Optimization
  - What is the minimum number of colors (*K*) needed to color a given graph
  - Must solve the problem in order to verify
  - **NP-Hard**

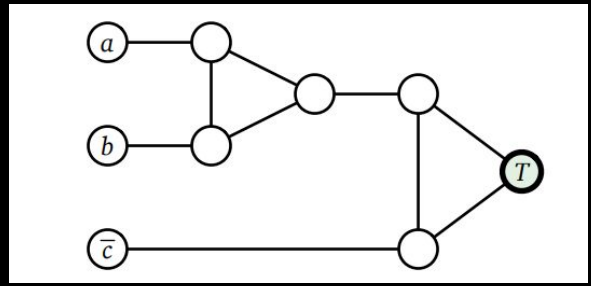
```
Is it possible to
color this graph
using two colors?

#Input:
2
a b
b c

#Output:
True
```

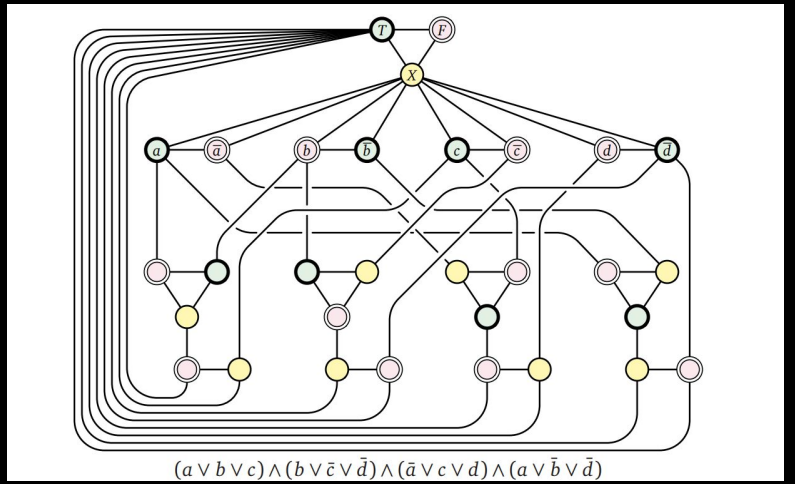
```
Minimum number
of colors?

#Input:
2
a b
b c

#Output:
2
a 1
b 0
c 1
```

3-Sat -> 3-Color

- Create **Truth gadget** - *O*(1)
  - T, F, and Other
  - Must have different colors
- Create **Variable gadgets** - *O*(n)
  - a, a′, and Other
- Create **Clause gadgets** - *O*(m)
  - Joins 3 literals in a clause to node T
- **Final graph** - *O*(n + m)
  - If graph is 3-colored then original 3-Sat formula is satisfiable







https://jeffe.cs.illinois.edu/teaching/algorithms/book/12-nphard.pdf

Exact Solution

Code

- Tries all possible assignments for the vertices **recursively**
- **Backtracking** occurs when it finds a conflict
- With *m* available colors and *V* vertices
  - **Exponential** Runtime: $O(m^V)$

Time & Input

- More edges = more runtime
- 60 % of max edges
  - 10 vertices = 45 max edges
  - Test with 27 edges

```python
def graph_color_util(self, color_assignment, colors, index) -> bool:
        # Base case
        if index == len(self.vertices):
            return True

        vertex = self.vertices[index]
        for color in colors: # Loop over the colors
            if self.is_safe(vertex, color_assignment, color): # Check if it is safe
                color_assignment[vertex] = color # If it is, then assign the color to the vertex
                if self.graph_color_util(color_assignment, colors, index + 1): # Recursive call
                    return True
                # Backtrack
                del color_assignment[vertex] # Delete color
        return False
```