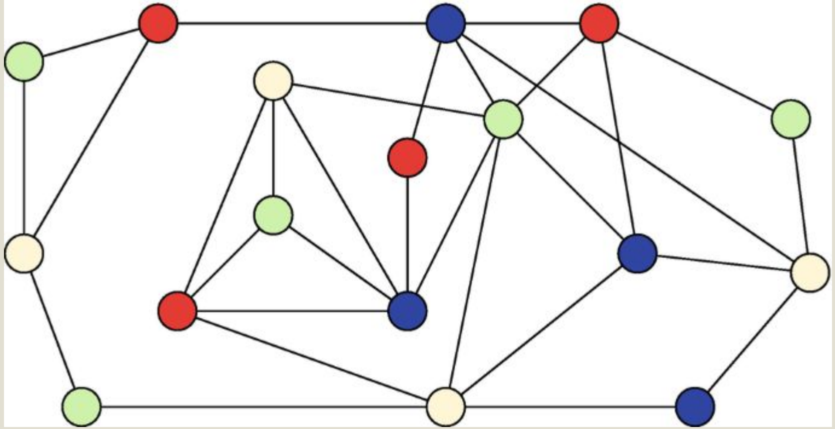
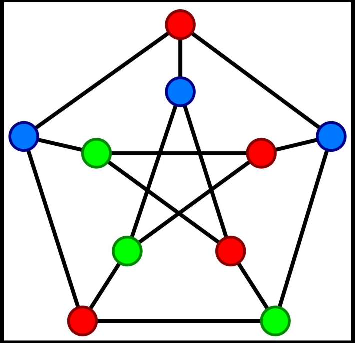
# MIN GRAPH COLORING

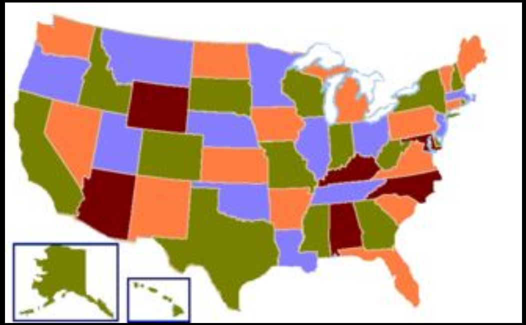


CS412 Colin Gregory, Austin Perdue, Madeline Burns

# Min Graph Coloring Intro

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

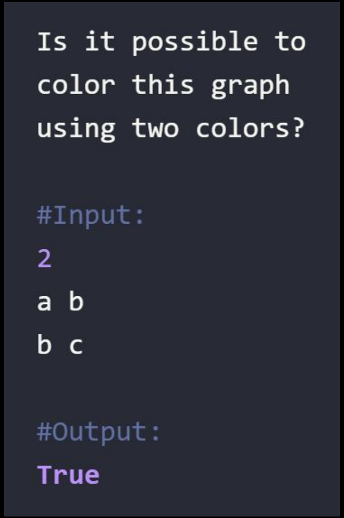




https://en.wikipedia.org/wiki/Graph_coloring

# Decision vs. Optimization

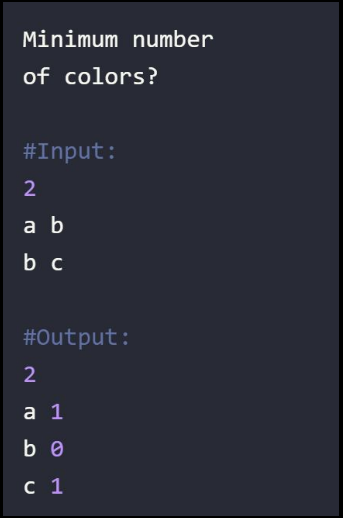
- Decision
    - Is it possible to color a graph with at most (*K*) colors?
    - **NP-Complete**
- Optimization
    - What is the minimum number of colors (*K*) needed to color a given graph
    - **NP-Hard**



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

#Input:
2
a b
b c

#Output:
True
```



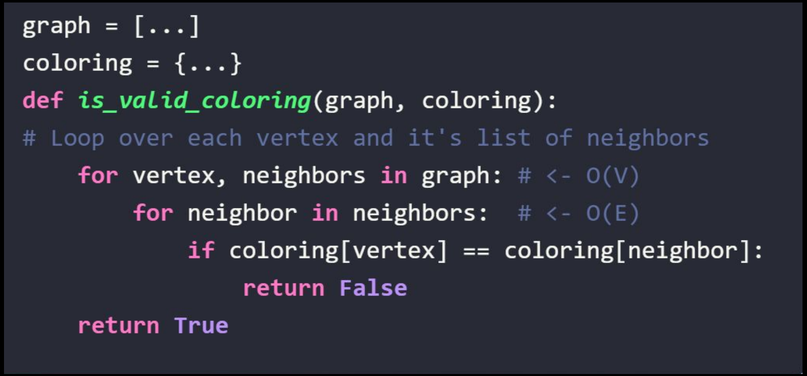
```
Minimum number
of colors?

#Input:
2
a b
b c

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

# Certifier process

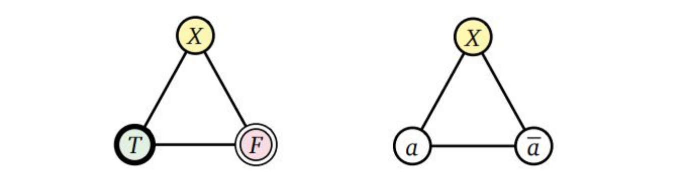
- Checks each vertex’s neighbor once
- Resulting in a time complexity of *O(V + E)*
- **Polynomial time** complexity
  - Scales with the number of vertices and edges

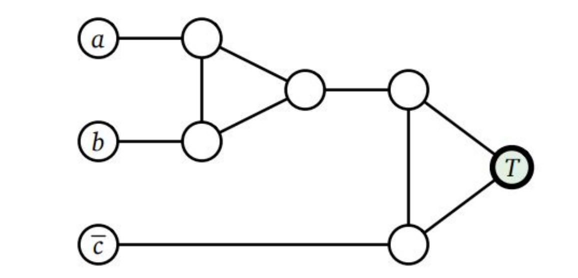


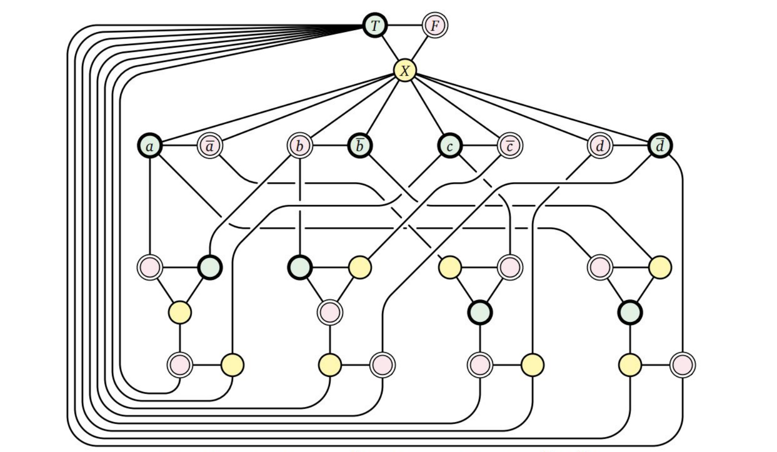
```
graph = [...]
coloring = {...}
def is_valid_coloring(graph, coloring):
# Loop over each vertex and it's list of neighbors
    for vertex, neighbors in graph: # <- O(V)
        for neighbor in neighbors:  # <- O(E)
            if coloring[vertex] == coloring[neighbor]:
                return False
    return True
```

# 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







$(a \vee b \vee c) \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d) \wedge (a \vee \bar{b} \vee \bar{d})$

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

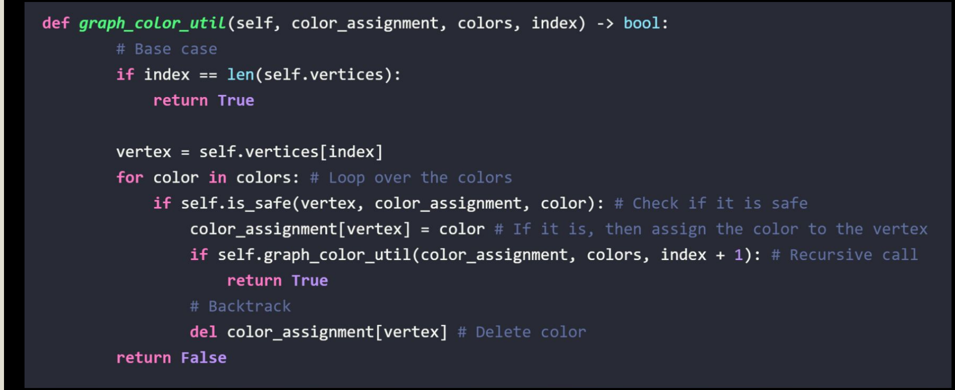
# Exact Solution

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: $O(m^V)$

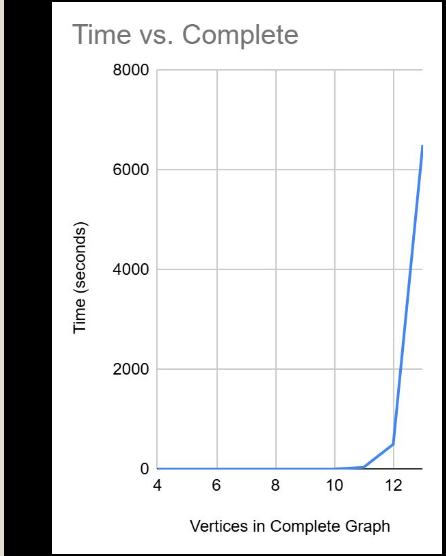
Time & Input

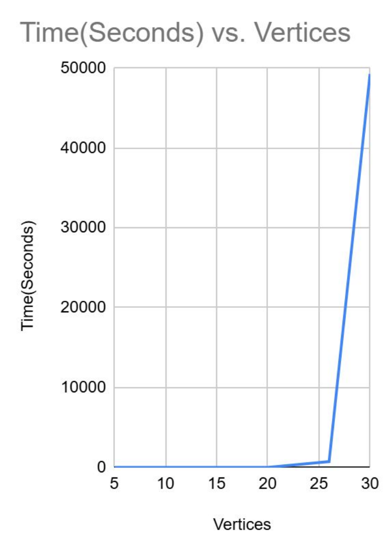
- More edges = more runtime
- Tested with 60% of max edges



```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
```


Time vs. Complete
Time (seconds)
Vertices in Complete Graph


Time(Seconds) vs. Vertices
Time(Seconds)
Vertices