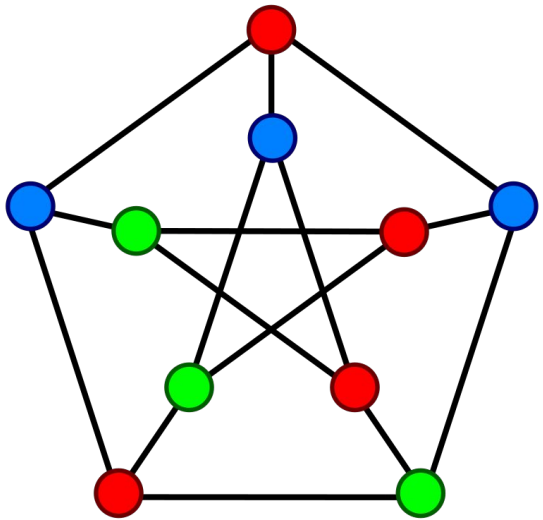
Minimum Graph Coloring

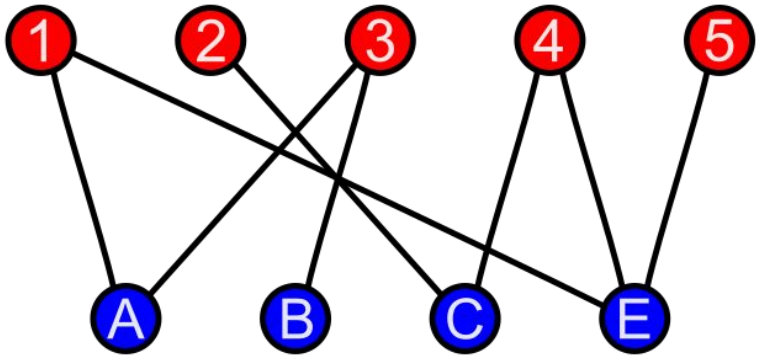
CS 412: NP-Complete Project

Graph Coloring

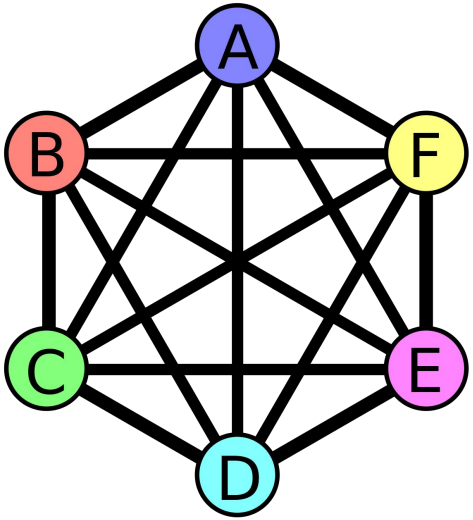
- Assign a color to each vertex in the graph
- No edges may connect two vertices of the same color
- Chromatic number χ(G) — minimum colorability for graph G
- Graphs are k-colorable



Unique Cases



Bipartite graphs



Complete graphs

Optimization & Decision Problems

- Optimization
  - Given a graph G, find the minimum number of colors required to color G (i.e., G's chromatic number).
- Decision
  - Is the graph G minimally k-colorable?

How to verify correctness?

- Check that the coloring is valid
- Loop through each vertex, confirming no adjacent vertices have same coloring
- Runtime complexity: O(E)

Exact Solution

- M-coloring algorithm
  - Uses backtracking
  - Ensure a *valid* color can be assigned (which is any color >= M)
  - If no safe option, backtrack and return false
  - If safe, add color to vertex and move to the next vertex
- Loop through M-coloring algorithm from M=1, increasing M until an M-coloring is found
- M-coloring time complexity: $O(M^v)$
- Exact solution time complexity: $O(M^{v+1})$

Dominant Term

```
def graphColoringAlgorithmUtil(m, colorArray, currentVertex, num_vertices):
    # base case
    if currentVertex == num_vertices:
        return True

    for i in range(1, m + 1):
        if isSafe(currentVertex, colorArray, i, num_vertices) == True:
            colorArray[currentVertex] = i
            if graphColoringAlgorithmUtil(m, colorArray, currentVertex + 1, num_vertices):
                return True

            # backtrack
            colorArray[currentVertex] = 0
```

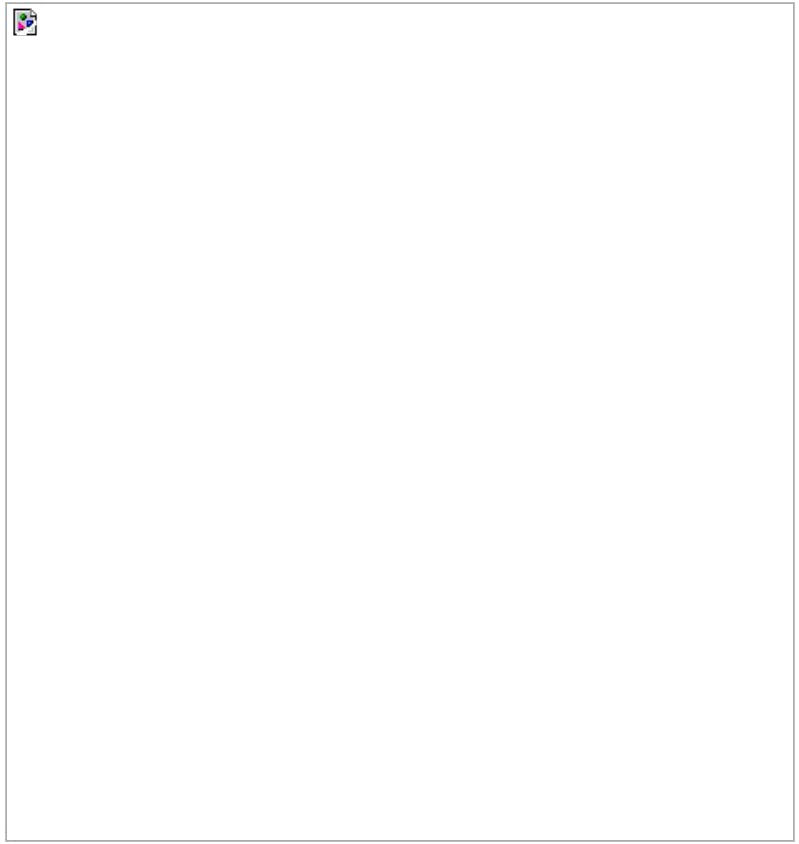
Approximation Algorithm

- Greedy algorithm
- Loop through vertices
    - Give each vertex the first valid color
    - A color is valid if it hasn’t been already been assigned to any of the vertex’s neighbors

Runtime Complexity

$O(V^2 + E)$

```
# Assign colors to remaining V-1 vertices
for u in range(1, num_vertices):

    # Process all adjacent vertices and
    # flag their colors as unavailable
    for i in adj[u]:
        if (result[i] != -1):
            available[result[i]] = True

    # Find the first available color
    cr = 0
    while cr < num_vertices:
        if (available[cr] == False):
            break

        cr += 1

    # Assign the found color
    result[u] = cr

    # Reset the values back to false
    # for the next iteration
    for i in adj[u]:
        if (result[i] != -1):
            available[result[i]] = False
```



Importance

- Graph theory
- Map colorings
- Compilers
    - Task scheduling
    - Register allocation
- Circuit design