Graph coloring with branch &bound and backtracking

def is\_safe(node, color, graph, colors, n):

for k in range(n):

if graph[node][k] == 1 and colors[k] == color:

return False

return True

def solve(node, graph, m, colors, n, best\_colors, min\_colors\_used):

if node == n:

used\_colors = len(set(colors))

if used\_colors < min\_colors\_used[0]:

min\_colors\_used[0] = used\_colors

best\_colors[:] = colors[:]

return

for clr in range(1, m + 1): # Try colors from 1 to m

if is\_safe(node, clr, graph, colors, n):

colors[node] = clr

# Branch and Bound: Only continue if current colors used < best so far

if len(set(colors[:node + 1])) <= min\_colors\_used[0]:

solve(node + 1, graph, m, colors, n, best\_colors, min\_colors\_used)

colors[node] = 0 # Backtrack

def graph\_coloring(graph, m):

n = len(graph)

colors = [0] \* n

best\_colors = [0] \* n

min\_colors\_used = [m + 1] # Store minimum colors used as list for mutability

solve(0, graph, m, colors, n, best\_colors, min\_colors\_used)

if min\_colors\_used[0] <= m:

print(f"\n Solution with minimum colors used: {min\_colors\_used[0]}")

print("Colors assigned to vertices:")

for i in range(n):

print(f"Vertex {i+1} ---> Color {best\_colors[i]}")

else:

print(" No solution exists.")

# Example graph (4 vertices)

graph = [

[0, 1, 1, 1], # Edges from vertex 0 to 1,2,3

[1, 0, 1, 0], # Edges from vertex 1 to 0,2

[1, 1, 0, 1], # Edges from vertex 2 to 0,1,3

[1, 0, 1, 0] # Edges from vertex 3 to 0,2

]

m = 4 # Try with more colors to see the minimum required

graph\_coloring(graph, m)

theory

Graph Coloring Problem Overview:

The Graph Coloring Problem is a type of Constraint Satisfaction Problem (CSP) where:

You are given an undirected graph.

You must assign colors to each vertex such that no two adjacent vertices share the same color.

The goal is to use the minimum number of colors possible (i.e., find the chromatic number).

🔁 Backtracking Approach:

Try to assign a color to each vertex one by one.

For each vertex, check whether the color is safe (i.e., not used by any adjacent vertex).

If safe, assign it and move to the next vertex.

If not, try a different color.

If no color is valid, backtrack and change the color of the previous vertex.

⛔ Branch and Bound Optimization:

It improves over plain backtracking by pruning paths that are not promising.

Before continuing deeper in recursion, we check:

If the number of colors used so far already exceeds the best solution found, then we stop exploring that path.

This helps in reducing the search space and improving efficiency.