Daily Coding Problem Blog

Daily Coding Problem #56

Problem

This problem was asked by Google.

Given an undirected graph represented as an adjacency matrix and an integer k, write a function to determine whether each vertex in the graph can be colored such that no two adjacent vertices share the same color using at most k colors.

Solution

We can use backtracking to solve this problem. More specifically, we start at vertex 0, try out every color from 0 to `k - 1`, and then see if we can recursively paint the rest of the graph without any conflicting colors. We'll create a helper function `valid(graph, colors)` that looks at the last colored vertex and all its neighbours to see if it conflicts with any of its neighbours (i.e. has the same color). We can skip over all uncolored vertices here.

To represent the colors, we can just keep a separate colors list that maps 1-to-1 with the vertices. You can also convert the graph into nodes and add a color property as well.

```
def valid(graph, colors):
    last_vertex, last_color = len(colors) - 1, colors[-1]
    colored_neighbors = [i
            for i, has_edge
            in enumerate(graph[last_vertex])
            if has_edge and i < last_vertex]
    for neighbor in colored_neighbors:
        if colors[neighbor] == last_color:
            return False
    return True

def colorable(graph, k, colors=[]):
    if len(colors) == len(graph):
        return True

    for i in range(k):
        colors.append(i)
        if valid(graph, colors):
            if colorable(graph, k, colors):
                return True
        colors.pop()

    return False
```

This runs in O(k^N) time and O(k) space, where N is the number of vertices, since we're iterating over k colors and we are backtracking over N vertices.

© Daily Coding Problem 2019

Privacy Policy

Terms of Service

Press