Code Jam 2022 - Round 3

Analysis: Mascot Maze

This is a variation of the classic graph coloring problem, where the rooms, exits, and mascots of the problem form the nodes, edges, and colors of a graph.

Cases where there is a cycle of length two (two rooms x and y where there is an exit from x to y and an exit from y to x) are obviously impossible. We will show later that these are the only impossible cases, by giving an algorithm that always produces a coloring for graphs with no cycles of length two.

Test Set 1

Given the low limits, it is possible to implement a backtracking solution: recursively try coloring each vertex in each color. There are techniques which will help speed up the solution. For example, you can keep track of the colors that shouldn't be used for a vertex (since one of its neighbors was previously colored with that color).

However, plain backtracking algorithms are unlikely to work for Test Set 2, since early color assignments might make it impossible to color later nodes, and it can take a long time until those early assignments are revisited by the algorithm.

Test Set 2

Consider the graph G' which has an edge from one node to another if there is a path of length 1 or 2 linking them in G. That is, G' contains the edge (v,w) if (v,w) is in G, or there is a pair of edges (v,x) and (x,w) in G. It is sufficient to color G' so that no adjacent pair of nodes has the same color.

Since each node in G' has outdegree at most 6 (2 nodes that are one edge away in G and 4 nodes that are two edges away in G), the average indegree must be at most 6, and so at least some individual node V must have degree at most 6 + 6 = 12. No matter what colors were chosen for this node's neighbors, there is always at least one different color for this node because it has at most 12 neighbors.

The only case when the coloring is impossible for G' is when it contains a self-loop. In such a case it is also impossible for G.

Using these observations the following algorithm can be used to color G':

1. Find some V with the degree not greater than 12.
2. Temporarily remove V (and the edges connected to it) from the graph.
3. Recursively color the rest of the graph (which still maintains all the described properties).
4. Reinsert V and color it.

Here is one way to implement this solution efficiently:

- Build an adjacency list and an array of degrees for G'. Make sure to track both incoming and outgoing edges.
- Run a breadth-first search to remove the vertices of degree not greater than 12 one by one. Start by adding all such vertices to the queue and use adjacency lists for efficient removal. If the degree of any neighbor of the vertex currently being removed becomes 12, add this vertex to the queue. Note that you don't need to remove the vertex from the

adjacency lists (as you will need those further), just update the degree. While running BFS keep track of the order in which the vertices are traversed.
- Go through the vertices in the reverse order and color each one greedly: try each color and use it if none of the neighbours still have it.

Every step of the algorithm can be done in linear time, so the overall time complexity of the solution is linear.

Some heuristic approaches, like local search, can also be made to work if implemented efficiently.