# Title: Graph 3-coloring

## Objective

The objective of this challenge is to develop an efficient 3-coloring algorithm to reduce the sum of weighted monochromatic edges (edges where both vertices have the same color).

Given the undirected graph $G = (V, E)$, define:

- $w$: $E \to \mathbb{R}^+$ as edge weights.
- $c$: $V \to \{1,2,3\}$ as the coloring function.
- A violation occurs if an edge $(u, v) \in E$ has $c(u) = c(v)$.

Objective:

$$\min \sum_{e=(u,v)\in E} w(e) \cdot I[c(u) = c(v)]$$

Where $I[\cdot]$ is the indicator function (1 if true, 0 otherwise).

## Background

Generally, most of approximate coloring algorithms may be classified to:

- **Heuristics**: work fast, provide a local optimum without estimating the lower bound.
- **Math programming** (ILP, IQP, SDP): work slow, can estimate the lower bound, converge to the global optimum

Since there's no one silver bullet, we will focus on a following class of graphs:

- **20000 nodes** are uniformly generated within a square.
- Each node is randomly connected to some **subset of its' 60 closest neighbors** (by Euclidean distance).
- Edge weights are distributed **lognormally**.
- Note: in the benchmark, we sum the weights for $(u, v)$ and $(v, u)$ before transforming to an undirected graph.
- Example graph generator: https://github.com/sedefe/3-coloring

## Challenge Tasks

1. Minimize the sum of weighted monochromatic edges.
2. Estimate the objective's lower bound.
3. Algorithm shall work in <30 min on a single-threaded CPU environment.

## Deliverables

**Report**

A comprehensive report covering the following:

- A clear description of the algorithm(s) implemented, both for tasks 1 and 2.
- Benchmarking report.

**Code**

- Python/C++ code.
- Only publicly available third-party code must be used (e.g. no use of commercial MILP solvers).