Daily Coding Problem #292

Problem

This problem was asked by Twitter.

A teacher must divide a class of students into two teams to play dodgeball. Unfortunately, not all the kids get along, and several refuse to be put on the same team as that of their enemies.

Given an adjacency list of students and their enemies, write an algorithm that finds a satisfactory pair of teams, or returns `False` if none exists.

For example, given the following enemy graph you should return the teams `{0, 1, 4, 5}` and `{2, 3}`.

```
students = {
    0: [3],
    1: [2],
    2: [1, 4],
    3: [0, 4, 5],
    4: [2, 3],
    5: [3]
}
```

On the other hand, given the input below, you should return `False`.

```
students = {
    0: [3],
    1: [2],
    2: [1, 3, 4],
    3: [0, 2, 4, 5],
    4: [2, 3],
    5: [3]
}
```

Solution

We can look at the first case above to gain some intuition.

Suppose we start with student `0`, and arbitrarily place him on team `0`. Then we know that his enemy, `3`, must be on team `1`. It follow that the enemies of `3`, namely `0`, `2`, `4`, and `5` must all be on team `0`. We can go back and forth in this way, until we either reach a contradiction or create a working set of teams.

In effect, we must perform a modified breadth-first search, where new students and their teams are added to the queue of elements to process in the order that they are found in our enemy graph.

Finally, it may be the case that a single pass of breadth-first search does not process every student. To handle this situation we can continue calling our `assign` function until we are sure to have visited each student.

```
from collections import deque

def make_teams(kids):
    teams = {0: [], 1: []}
    visited = set()

    while kids:
        start = next(iter(kids))
        if not assign(kids, teams, start, visited):
            return False

    return teams
```

```
def assign(kids, teams, start, visited):
    queue = deque([(start, 0)])

    while queue:
        kid, team = queue.popleft()
        teams[team].append(kid)

        enemies = kids.pop(kid)
        for enemy in enemies:
            if enemy in teams[team]:
                return False
            elif enemy not in visited:
                queue.append((enemy, 1 - team))

        visited.add(kid)

    return True
```

Each call to `assign` will process independent students, so altogether this algorithm will take as much time as traversing the vertices and edges of our enemy graph, or `O(V + E)`.