Round Robin Match Scheduling
Robin Zhang, 2018

As a larger part of the Soccer Management Game project

From https://en.wikipedia.org/wiki/Round-robin_tournament#Scheduling_algorithm, the basic design of a round robin algorithm is as follows:

First, we have every team face off against another team. This is the first matchup. For the next matchups, we hold one team static (typically the first), and we rotate every other team around in a "circle". To visualize this, the "circle" will consist of the teams:

Team 1	Team 3	Team 5
Team 2	Team 4	Team 6

In this above case, Team 1 plays Team 2, Team 3 plays Team 4, and Team 5 plays Team 6. If we make Team 1 static, the circle consists of:

Team 1	Team 3	Team 5
Team 2	Team 4	Team 6

Note: the format above can also be pivoted to create a more vertical format of matchups, but this doesn't change the essence of the algorithm.

When we rotate the "circle" once, clockwise, we get:

Team 1	Team 2	Team 3
Team 4	Team 6	Team 5

To generate the entire schedule, we create these matchups, we rotate the "circle" until we reach the matchups that we started with.

There are some things to take note of:

1. To deal with an odd number of teams, we create an additional "fake" team. Any team that is playing this "fake" team will be on the "bye".
2. The structure of the table in the examples above makes it very easy to generate "home" and "away" and other formats.

Some math

Let's assume we have n teams.

The first thing we need to do is turn this into an algorithm that can be used by a computer program. One (naïve) solution is to form an array with our teams and simply rotate the teams in indices 1 through n-1 by pushing the second team (in index 1) to the end and moving the rest forward one index. However, this doesn't produce the matchups we need, as every other round, some of the matchups will be the same (the teams next to each other will be playing each other multiple times).

So, we could still use an array, but we need to structure it such that it successfully generates the algorithm we designed above. My solution is to store the structure above in an array. Because this algorithm usually runs given a list of teams (in an array), we don't want to modify the original list, so we need to create a

new array. We can then manipulate the order of the teams within this new array to create the matchups. We need a second array holding these new matchups, and we go through each matchup column in the team array to generate another matchup.

Pseudocode for this algorithm looks like the following:

```
Void generateSchedule(Team[] teams) {
      Create matchups array
      Make shallow copy of teams array

      While not at first matchup {
            For each pair of Teams in teams {
                  Add pair to matchups array
            }
            Rotate teams
      }
      Add more features if necessary (randomize, repeat matches, etc)
}
```

I believe the inner for loop is trivial. It fetches the Teams from indices x and x+1, so we need to use a x += 2 increment in the for loop.

Rotating the Teams is harder than it looks. Let's assume zero indexing, clockwise rotation, and fixing the zero indexed Team. For n Teams, with n being even (if there are an odd number of Teams, remember that we add a dummy Team, so there will always be an even number), the next spot for the odd number Teams is the next higher odd number Team, unless that index is n-1. In that case, we just decrement the index by 1. For even number indices, we decrease the index to the next lower even index, unless we are at index 2 (index 0 is fixed), in which case we move back to 1.

For example for 8 Teams, we have indices 0 through 7:
0 is fixed
1 -> 3 -> 5 -> 7 -> 6 -> 4 -> 2 -> 1.

The order is always the same for a given number of Teams. We simply rotate the Teams in our array in this order, while matching each pair of Teams to generate a matchup.

Other Settings

Some settings we want to be able to incorporate are:

1. Have n-repetitions of a match (a Team plays each other Team n times)
2. Have random matchups per week, instead of in order
3. Have weeks (matchups) where certain top Teams play (like a highlight of the season matchup)

To design the first setting, we create the final array with the matches with a size of

```
teams * (teams - 1) / 2 * number of times each Team plays each
specific other Team
```

We first generate the matchups, then we repeat those matchups for `n-1` more times.

Adding random matchups is done by using the pseudorandom number generator when generating the final schedule. There are two ways to do this. The first is to place the random matchups in place, meaning the empty matches array will be filled, randomly, with matchups. The second is to generate the in-order array first, then randomly transfer over to a new array.

The “randomness” that comes from this is the *week* that is taken into the matchup schedule. For instance, the first in-order week is when the first Team plays the second Team, the third plays the fourth, and so on. In a random scenario, this “first” week may be “randomly placed” into the fourth week, with a different set of matchups in the first week. So this means we have to take the entire week’s worth of matchups when we randomly generate the schedule.

The last setting is to place certain matchups in certain times of the schedule. The user will be able to specify which week this highlight matchup will occur, and subsequently less important matchups will occur around that week.

This setting assumes that each Team has a certain strength that can be measured. This may be done with overall attributes, previous season results, or an ELO rating.