Code Jam 2017 - Round 2

Analysis: Roller Coaster Scheduling

Roller Coaster Scheduling: Analysis

Small dataset

There are various ways to solve the Small dataset; for instance, it can be reduced to a flow problem. It is also possible to find a greedy strategy or even a formula: the number of rides needed is max(number of tickets for customer 1, number of tickets for customer 2, number of tickets with seat 1), and the number of promotions needed is max(0, (maximum among counts of i-th seat among all tickets) - number of rides needed). However, these strategies are tricky to prove. The rest of our analysis will shed some light on why they work.

Large dataset

To solve the Large dataset, we can start with observations similar to those in the Small case: the number of tickets held by any one person is a lower bound on the number of total rides, and so is the number of tickets for position 1. This is because the set of tickets held by any one person, and the set of tickets for position 1, share the property that no two tickets in the set can be honored in the same ride, even with promotions. An extension of this is that for the set of all tickets for positions 1, 2, ..., K, at most K of them can be honored by a single ride (possibly using promotions). This means that, if S_K is the number of tickets for positions up to and including K, ceil(S_K / K) is also a lower bound on the final number of rides.

It is not difficult to see this when there is only one ticket per customer: if the maximum of all those lower bounds is R, for as long as there is some position P such that there are more than R tickets for position P, we can promote any ticket for position P to some previous position that has less than R tickets assigned, which is guaranteed to exist due to the R ≥ ceiling(S_P / P). After that, no position is assigned more than R tickets. Since there are no repeated customers, we can just grab one ticket for each position that has tickets remaining and assign them to a ride until there are no more left. This will yield an assignment with exactly R rides, and we proved above that there can't be less than R.

When there are customers holding more than one ticket, our greedy assignment in the last step above might fail. We can still prove that there is an assignment that works with a bit of mathematical modeling. Consider a fixed ride plan. Let us define the ride plan matrix as a square matrix of side S = max(**N**,**C**). The first **C** rows represent customers and the first **N** columns represent positions. The remaining rows or columns represent fake customers or positions, whose role will be clear in a moment. For each ride in the plan, we construct a one-to-one assignment of customers and positions. Customers that participate in the ride are assigned to their position on it. Customers that do not are assigned to empty positions or fake positions. If there are more positions than customers, each empty position is assigned a fake customer. Then, the value of a given cell of the ride plan matrix is the number of times the represented customer was assigned to the represented position. Notice that the value is an upper bound on the number of times a customer actually rode in the position, but not the exact number.

Notice that for any ride plan consisting of R rides, its ride plan matrix will have rows and columns that sum up to R. This is because for each ride, there is exactly one cell per row and one cell per column that gets a 1 added to it. The most interesting realization is that we can go the other way: for any matrix M such that all its rows and columns add up to R, there is a ride plan consisting of R rides such that M is a ride plan matrix for it. The proof is a simple variation

on the Birkhoffâ€“von Neumann theorem, which implies a matrix with that property can be expressed as a sum of permutation matrices, and each permutation matrix corresponds to a possible ride.

A given set of tickets can also be represented by a matrix of side S by having a cell contain the number of tickets a given cusotmer holds for a given position. Let us say that a matrix M is less than or equal to another matrix M', if and only if the cell at row i and column j in M is less than or equal to that the value of the cell at row i and column j in M'. After promotions, we need the matrix representing the tickets to be less than or equal to the ride plan matrix.

Notice now that the greedy promotion algorithm presented in the second paragraph of this section, actually yields a ticket matrix such that no row or column exceeds the established lower bound R, even if there is more than one ticket per customer. The proof is exactly the proof above. What was missing before was a way to know that set of tickets could be turned into an actual ride plan in the case where there is more than one ticket per customer. We now have such a way, which means the originally naive solution is actually a full solution for the problem. Moreover, since we need to report the minimum number of promotions and not the promotions themselves, we can just add S_P - R for each position P such that S_P > R, and we are done.