PARKING

The task can be solved by simulation. We need to keep track of how many trucks are in the rest area. For every minute between 0 and 100, we:

- Add the number of trucks arriving during that minute.
- Subtract the number of trucks departing during that minute.
- Charge for parking based on the number of trucks parked.

SEMAFORI

Simulation suffices for this task too. Two parameters are relevant; where we are and what time it is. Until we reach the end of the road, do:

- If there is no traffic light at the current position, increase the position and time by 1.
- If there is a traffic light that is green, also increase the position and time by 1.
- If there is a traffic light that is red, increase the time by 1.

What remains is to determine the light at a traffic light at a given time index T. If T starts from zero, then a traffic light is red when $T \bmod (R+G) < R$, and green otherwise.

GRANICA

For 60% of the score an easy brute force solution was available, trying every M between 0 and 10000 and checking whether all given integers give the same remainder when divided by M.

For the full score, we need the following deduction: if some two numbers A and B give the same remainder when divided by C, then C divides $A-B$. So for an integer M to be part of the output, it must divide the differences between each pair of input numbers. The largest such M is the greatest common divisor of the differences between input numbers. The remaining values of M are all divisors of the largest M (except for 1).

GEORGE

We model the city as an undirected weighted graph. When Mister George traverses an edge, we can consider the weight of that edge to change over time. For example, if Mister George enters a street during minute 10 and the street has weight 5, then the weight changes like this:

Time	9	10	11	12	13	14	15
Weight	5	10	9	8	7	6	5

After thinking about it, we conclude that the problem can be solved with Dijkstra's shortest path algorithm, with the weight of an edge depending on when we use it. For implementation details see the official source code.

PRINCEZA

Suppose we store in one linked list the points sorted by the sums of their coordinates (breaking ties by x-coordinate), and in another linked list we store the points sorted by the differences of their coordinates (again breaking ties by x-coordinate). Additionally, for each point we keep a pointer to its position in each list.

Imagine the frog is located in some point (x, y) and that the next direction to process is A. The next point in direction A is of the form (x+P, y+P) for a positive integer P. So, if it exists, the next point has the same difference of coordinates as the current point, and its x-coordinate is larger than the x-coordinate of the current point. Observe that the next point is exactly the successor of the current point in the second list. If the current point has no successor or the successor doesn't have the same difference of coordinates, the jump will not occur. If the jump occurs, delete the current point from both lists and move to the next point.

Similar reasoning can be applied to all four directions.

CESTARINE

Let us first solve the problem without the constraint that a driver may not use his ticket at the same exit where the ticket was issued. Because the tickets may be exchanged arbitrarily, any driver can obtain any ticket. It is easy to see that the optimal solution is to sort the tickets, sort the drivers by their desired exits, and give the drivers tickets in order.

What if a driver gets a ticket issued at the same exit he needs to use? The best action would be for him to swap his ticket with one of his colleagues next to him in the sorted sequence, which is always possible. But what if, as in the second example test case, two drivers want to exchange tickets with the same driver? Then we need to allow them to exchange tickets not only with immediately adjacent drivers, but also those 2 indices away in the sorted sequence. From this analysis we obtain a dynamic programming solution.

Let dp[n] be the cost of the optimal distribution of tickets for all drivers up to driver n (the drivers are sorted by their exits). Let distribution(n, k) be the cheapest distribution of tickets in positions n, n−1, ..., n−k to drivers in positions n, n−1, ... n−k. There are (k+1)! possible distributions, but because k will be at most 2, we can check every one and select the distribution which results in the least total cost. The DP relation is:

$$dp[n] = \min\{\ dp[n-k-1] + \text{distribution}(n, k) \text{ for } k \in [0, 2]\ \}$$

The solution is dp[N].