EGOI 2025 Editorial - Dark Ride

Task Author: Ivan Gaspardy

July 16, 2025

Introduction

In this interactive problem, there are N rooms and N light switches that each turn on the light in a distinct room (but we do not know the matching). The goal is to find out which switches belong to the first and last room. In each query, you can turn on any number of switches. The answer to a query is the number of changes between a dark room and a lit room.

Subtask 1 ($N = 3$)

In this subtask, there is a starting room, an ending room, and a middle room between them. Suppose the light is only on in the middle room. The passengers will scream once when entering the room, and once when exiting it. What happens if the light is only on in the last room? What happens when the light is only on in the first room? In these cases, the passengers will only scream once, because the rollercoaster won't enter the first room, and it won't exit the last room. Thus we can check in a single query, whether or not a particular switch belongs to the middle room by setting only that switch on. If that switch belongs to the middle room, the query will return 1. Otherwise it will return 2. Thus we make three queries (100, 010, and 001) to find the rooms at both ends.

Subtask 2 ($N \leq 30$)

We can extend the idea from the previous subtask to solve this. In one query, we can check if a particular switch controls one of the end rooms, by making 30 queries, each of which only has one switch on. The two queries that return 1 must have had one of the end switches on.

Subtask 3 ($p_0 = 0$)

Let's call room 0 the first room. In this subtask, we never turn on switch 0, and thus, the first room stays dark. We further extend our observation from subtask 1. How many times will the passengers scream? First, suppose the last room is dark as well. We know that they'll scream the first time they enter a lit room. We also know that every time they enter a lit room, they'll eventually enter a dark room, since the last room is dark. Hence they must scream an even number of times. If the last room is lit, then they will scream an odd number of times since in the end, they do not go from a lit room to a dark room again. We can use this knowledge to binary search which switch controls the light in the last room. This will take $\log_2 N$ queries, which is no more than 15. In general, if we know where one output switch is, then we can binary search for the other one in at most 15 queries.

Subtask 4 (N is even, and one output switch in each half)

Without loss of generality, let's say the output switch in the first half of the switches controls the first room, and the output switch in the second half controls the last room. We can extend our binary search approach from subtask 2. Previously we knew exactly where the switch for the first room was, but now we only have a range for where it could be. This actually doesn't stop us from applying the same strategy as Subtask 2 to locate the switch that controls the last room, since we can set all of the switches in the first half to dark, and be assured that the first room is dark, so we're free to perform our binary search on only the switches in the second half. Once we've located one of the output switches, finding the second one is a matter of repeating the algorithm, but in reverse. That is, set the entire second half to dark, and run the binary search in the first half. Each binary search will take no more than 14 queries, so we will be able to locate both switches in 28 queries.

Subtask 5 ($N \leq 1,000$)

Here we are not guaranteed anything about the locations of the significant switches. What would be ideal is if, like Subtask 4, we somehow managed to split all of the switches into two groups such that each group has only one significant switch in it. We discussed earlier that the query results will be odd if and only if the end rooms ended up with different light settings, so we would like to create a circumstance where an odd query is guaranteed since once we have that, we can apply our solution from Subtask 4. What happens if we split the array down the middle into a first half and a second half, and we make a query where we only set the first half on? If the result is odd, great! Otherwise, we know that either both switches are in the first half, or both are in the second half. We can actually repeat this strategy on each half of the array. We split each half into halves, making quarters, and we set the first and third quarters to be on (that is the first half of the first half, and the first half of the second half). Below is a diagram where $N = 8$.

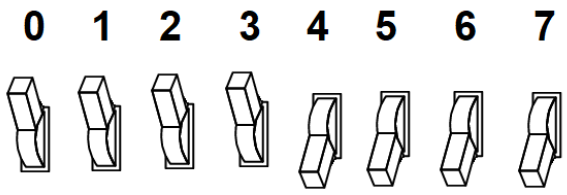

fig. a

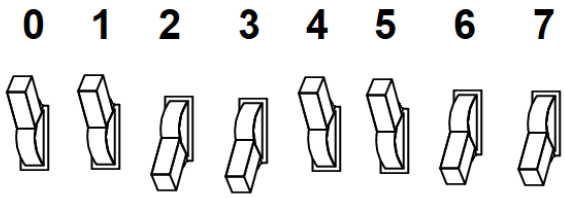

fig. b

fig. a is our first query, and fig. b is our second query. Now if we end up with an odd result, great! Otherwise, we know that the two significant switches must be in the same quarter of the array, so we can cut each of the quarters in half, making eighths, and repeat! This strategy will guarantee that we end up with an odd result in no more than $\log_2 N$ queries, which for this subtask, is not more than 10. Once we've got

an odd query, we've successfully split the array into two groups such that each group contains one significant switch, so we can apply our solution to Subtask 4. Note that because $N \leq 1,000$, our Subtask 4 algorithm will take only 20 queries, so we fit in under 30 total queries.

Full Solution

For the full solution, we find that we only actually need to run half of the Subtask 4 algorithm. That is, we only need to use binary search to locate one of the significant switches. In Subtask 5, we stopped subdividing the switches into smaller and smaller groups as soon as we hit an odd query. If we don't do this, and we instead let it run through 15 iterations, and then use binary search to find one of the significant switches, we will actually have all the necessary information to locate the second significant switch. We can do this by going over our initial 15 queries. Since we now know where one of the significant switches is (let's call this switch A), we can tell based on the query result, whether or not the two significant switches had the same light setting or not. It happens that there will always be exactly one switch that is consistently set the same as switch A when the result is even, and set different from switch A when the result is odd. Thus it must be switch B, and we've located both the end switches in no more than 30 queries.