Trivial solution 0 p.

We can always just ask about each single alley in the park and this will take $4N$ queries ($\pm$ a constant), this solution is too trivial and, because of that, no points are given for it.

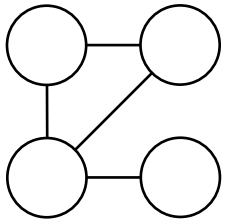
Subtask 1 for 15 p.

The first subtask is there, in case we can't make all observations needed for the second one, but we've made some which is enough to bring down the number of queries. That is why there is no particular intended solution for it.

Subtask 2 for 45 p.

The solution to the second subtask is based only on examining the possible directions of the alleys and then asking for all unknown directions. In fact, it turns out that all outside alleys (those between two places with a difference of two) always have a positive direction (the proof will be described below). After that we ask one query per inside alley – that is $2N$ queries.

The method we will use for finding that is the following. We will examine the alleys in groups of four. Four such sequential alleys are all alleys but one between four sequential places. The next 4-tuple of alleys is again all alleys but one between four sequential places, but moved two to the right. The structure to examined is shown in the figure:



Obviously, if we follow no restrictions, we can have $2^4 = 16$ combinations of the directions of the four alleys. After that we can eliminate those with cycles in them and we have 14 combinations left. After that we can see which ones are possible for the first 4-tuple: 1111, 1110, 1101, but not 1100 (the digits represent the directions of the alleys from lowest to highest). 1100 is not allowed, because the there will be no alleys coming out of place two. After that we can examine for each of those configurations, which ones are reachable from it, then from the new ones and so on. We can see that not all configurations are reachable at all. We can repeat the same process starting from the last 4-tuple, for which we have other possibilities. Then we can see which ones can reach it. That way we will eliminate other configurations. In the end, we will only be left with three: 1111, 1101 and 0111. This means that no allowed configuration has outside alleys with a negative direction. The whole working out of this result is not written in this analysis, since it is trivial once we know the method.

Subtask 3 for 70 p.

In the solution to subtask 2, we still don't take advantage of the fact that not all possibilities for the inside alleys are allowed. The already found orientations of the outside alleys fulfil the rule of at least one ingoing and at least one outgoing alley for each place (except the entrance and exit), but the no cycles rule can be broken. (From here on we will only examine the directions of the inside alleys.) In fact, we can notice that, if the directions of two sequential (inside) alleys are both negative, there will always be a cycle. This means that we got the problem to the following:

We have a string of $2N$ zeroes and ones without sequential zeroes. We must find the string with queries of the type described. We can see that there are exactly F_{2N+2} such strings, where F_k is the k-th Fibonacci number. This is caused by the fact that the number of strings of length k which end on a 1 is equal to the number of strings of length $k-1$, while the number of strings that end on 10 is equal to the number of strings with length $k-2$. Each query in the best case eliminates half of all possibilities. Therefore, the number of queries needed for a big N is approximately $\log_2 F_{2N} \approx \log_2 \phi^{2N} = 2N \log_2 \phi \approx 1.388N$.

The problem is that there is no quick way to find the best queries. That's is why вe can use a suboptimal method and split the string into equal length intervals, and find each interval with a minimum number of queries. This is not optimal, because it assumes that all combinations of sequential intervals are possible, while that is not the case, because they can meet with two zeroes in their ends. But, in fact the solutions to both the third and the fourth subtasks are based on that method.

We can first write a program that calculates the number of queries for a given length, by first finding the number of strings with that length, taking the $\log_2$ of that and rounding up. We can see that the first two lengths, strictly better than the ones smaller than them, are 4 and 11. For length 4 we need 3 queries and for 11 – 8. We have $2N$ elements in the string and that is how we get the constraints for the third and fourth subtask. We can also figure out these constants by looking at the grading of the task.

To solve the third subtask it is enough to write down the 8 options and figure out what queries to ask:

1111
1110
1101
1011
0111
1010
0101
0110

If we first ask for the XOR of all alleys and then the XOR of the first and last alleys, we get the following four groups of two configurations:

First XOR Second XOR	0	1
0	1111 0110	1101 1011
1	1010 0101	1110 0111

We can easily differentiate between the two configurations left with the third query. With that the solution to the third subtask is complete.

Subtask 4 for 100 p.

The fourth subtask doesn't require the figuring out of new mathematical ideas. The focus here is finding the needed three of queries for a string of length 11. While with length 4, we can easily do this by hand, with 11 that is far from the case. Several approaches are possible: precompute at the beginning of the solution (the time limit is as free as possible, to allow for this), precompute locally and hard code the answers and a mix between those two (e.g. hard coding the first few queries, which are the slowest to find). In the case when something is done locally, it can be only partially automated, e.g. finding the first few queries by hand and then our program checking whether we can fill the whole tree with queries.

The author solution performs a precompute at the beginning of its executions. A brute force with some optimizations is used:

1. When we are in a situation where $2^Q > T$, where Q is the number of queries left and T the number of possible configurations, we terminate this branch of the brute force.
2. When there are fewer queries left (we are lower in the tree) simpler queries are tried, e.g. with only two or three alleys.
3. In the higher levels of the three, when our choices strongly impact the difficulty of finding appropriate questions lower in the tree, we use a stronger version of 1. So, we don't just check whether it is theoretically possible to find appropriate following queries, but we choose such a question, so that the number of possibilities with XOR 0 and with XOR 1 are as close as possible. The constant the performs the best is 0.55 times the number of current possibilities. Any lower and it is too restrictive, any higher and it is too free.
4. When there is only on query and two possibilities left, we use a faster method (about twice as fast) to find the needed question.

Not all of these optimizations (especially not the last one) are needed to fit into the time limit and probably there are many other possible optimizations. In the attached author solution you can find an implementation of this brute force with these optimizations. It is important to note, that we can speed up the author solution non-negligibly, if we remove the separation of pieces of code into separate functions (especially in some critical places), because these calls take time. However, this is not done in the author solution, so it is easier to follow.