Code Jam 2022 - World Finals

Analysis: SchrÃ¶dinger and Pavlov

The difficulty of this problem resides on the fact that the state of a box can affect outcomes long after the dog passed them by blocking or not blocking a tunnel. We deal with that difficulty by remembering the state of boxes. Of course, there are way too many boxes to remember the entire state, so we compress it to a smaller amount of information that is manageable. How we do that is different for each test set.

Test Set 1

In Test Set 1, all tunnels go to nearby boxes. That means once the dog is at box number i, the state of boxes with numbers lower than $i-5$ cannot affect the outcome anymore. So, we can do a dynamic programming solution that computes the probability that there is a cat at box i given the state of the k closest boxes. That is only $O(\mathbf{N} \times 2^k)$ states, and because k is bounded by only 10 for Test Set 1, that is small enough to solve the problem.

Test Set 2

For Test Set 2, the number of close boxes that we may need to remember is not bounded by a small number, so we cannot use a solution exponential in it. Let us consider a directed graph with boxes as the nodes and tunnels as directed edges. Because it has the same number of nodes and edges, it is a functional graph. Functional graphs look like forests except they have cycles as the "roots". Notice that only the connected component of the underlying undirected graph that contains the last box matters for the final answer (boxes in other components do not affect whether or not there is a cat in the last box). So, we can discard all other components and assume moving forward the graph is connected, so it's actually a functional graph with a single cycle.

As a thought exercise, let us assume the tunnels graph is actually a directed tree instead (one tunnel is removed). In this case, we can solve the problem by simulating the dog run and remembering things in a smart way. We maintain a forest of the nodes corresponding to every box and the tunnels that may have been used so far, that is, tunnels coming out of boxes that the dog already passed. For each node, we compute the probability that a cat is there. The probabilities of a cat being in any two boxes are independent if they are on different trees of this forest, but otherwise they might not be.

We start with the forest with all nodes and no edges. Probabilities for each node start at 0, $1/2$c or 1 depending on whether the box is empty, unknown, or contains a cat, respectively. Then, when we simulate the dog passing through box i, that adds one edge to the forest, which merges two trees. Because the probability of the roots of those trees are independent up to this step, we can calculate the probability of the tunnel being used by multiplying the probability that there is a cat in box i and no cat in the destination box. Then, we update all probabilities as the weigthed average of the outcome of both cases (a cat running through the new tunnel or not).

To make the approach above work for an actual functional graph instead of a tree, we need to deal with the sole cycle. We can do this by branching out when we add the first edge of the cycle to the forest (that is, when the dog runs through the box with the smallest number from among those in the cycle). Instead of merging those two components, we consider all 4 cases for the state of the two boxes at the endpoints of the tunnel. For each one, we can compute its probability with a multiplication as before, because at this points the two endpoint probabilities are independent. Then, instead of merging the two components, we start 4 computations, one for each case, but in all of them the components are kept separated. At the end, we have 4

results for the last box. The final result is the weighted average of that box where the weights are the probabilities of each case that we calculated when forking the computation.