Task: Garden

Proposed by: Normunds Vilcins

In this task, we would like to compute the number of possible paths that could have led each group to the specified intersection P, using the given number of steps K.

Notice that each path is completely determined by its initial intersection. Thus, to compute the number of possible paths, we only need to check whether each intersection, if used as an initial intersection, would bring the group to intersection P after exactly K steps. As we need to check all N initial intersections for each of the Q groups, an efficient algorithm for checking whether the group will be at intersection P after exactly K steps is needed, which will be discussed in following sections.

1 Graph construction

This problem can be treated as a graph problem. A natural approach is to construct a graph G containing the following information: for each intersection, where the group would move to. Since they may take only one of the two most beautiful trails, we will use two vertices to represent an intersection. Namely, for the i-th intersection, let v_i represent this intersection where the next chosen trail must be the most beautiful trail incident to it, and v'_i represent this same intersection but where the next chosen trail must be the *second* most beautiful trail incident to it (or the most beautiful trail if no alternative is available). In other words, v_i represents the i-th intersection when the last taken trail is *not* the most beautiful trail incident to this intersection, and v'_i represents this intersection when the last taken trial is the most beautiful one incident to this intersection.

Now for each vertex, we add an outgoing edge representing the most beautiful or second most beautiful trail, according to the conditions mentioned above. With this, G will contain $2N$ vertices, and exactly one outgoing edge from each vertex.

The construction of the graph G takes $O(M+N)$ running time by first creating $2N$ vertices, then scanning through the array R representing trails, and conditionally adding these edges to G under the described conditions.

2 An $O(M + NKQ)$ solution

A simple way to check where the couple would arrive after K steps is to simulate their path, for each intersection as an initial vertex. Since they always choose the most beautiful trail in the first steps, the corresponding starting vertices in G are $v_0, \ldots, v_{N-1}$.

To simulate their walk, we simply start at some vertex v_s, then follow the unique outgoing edge for that vertex, and repeat this process for K steps. Since the vertices corresponding to

intersection P are v_P and v'_P, then this path ends at this intersection if and only if after K steps, we stop at one of these vertices. That is, to find the number of possible paths, we simulate their walk for all possible initial vertices v_i, and count the number of starting vertices that end at v_P or v'_P after K steps.

Clearly, this process takes $O(K)$ total running time for each starting vertex. Since there are N possible starting vertices and Q questions, this algorithm takes $O(M + NKQ)$ running time, including graph construction. This running time is sufficient to fully solve subtask 1.

3 An $O(M + NQ \log K)$ solution

As K becomes large in subtask 2, we need a better way to simulate the algorithm mentioned in the previous section. Notice that the edges in G represents 1-step traveling. To simulate faster, we will use the permutation-composition approach.

We first precompute the result of 2^k-step traveling from each vertex in G, where $k = 0, 1, 2, \ldots$, using a technique similar to successive squaring. Let $T_{v,2^k}$ represents the vertex we arrive at after traveling from v for 2^k steps. Then for $k = 0, 1, 2, \ldots$, we can compute $T_{v,2^k}$ easily: If $k = 0$, then the destination is specified in G; otherwise, we compose the two paths of length 2^{k-1} using the formula $T_{v,2^k} = T_{T_{v,2^{k-1}},2^{k-1}}$. In other words, traveling 2^k steps from v is the same as traveling 2^{k-1} steps from v, then from that vertex, continue for 2^{k-1} more steps.

Then, notice that for each value of K, we can decompose this number into sum of distinct, non-negative powers of two. Suppose that $K = 2^{k_1} + 2^{k_2} + \cdots + 2^{k_l}$ where $k_1 < k_2 < \cdots < k_l$ for some positive integer l. Then the result of traveling k steps from v can be found by simply composing travelings of $2^{k_1}, 2^{k_2}, \cdots, 2^{k_l}$ that we have precomputed. Using this technique, therefore, we can compute the destination for each starting intersection in $O(\log K)$ running time. Note that since $K < 2^{30}$, we only need to compute $T_{v,2^k}$ for $k = 0, 1, 2, \ldots, 29$.

This algorithm takes $O(N \log K)$ extra running time to compute the values of $T_{v,2^k}$, as each of them can be computed in constant time. Then, we can find the destination of each path in $O(\log K)$. Thus, the total running time is $O(M + NQ \log K)$, which is sufficient to fully solve subtask 2.

4 An $O(M + NQ)$ solution

Let us consider a more general question of determining whether a path starting at vertex s with length K ends at vertex t. Recall that each vertex in G has exactly one outgoing edge. So, from any initial vertex, by simply following these edges, we will eventually enter a cycle. Thus, if we start at s, exactly one of the following conditions are met:

- We never reach t.
- We reach t just once exactly after some number of steps F. In this case, t is reachable, but is not on any cycle.
- We reach t for the first time after some number of steps F, and enter it every C steps. In this case, t is reachable, and is on a cycle of size C.

For our purpose of solving the problem, s can vary depending on our initial vertex; namely, it can be any of the vertices v_i for $i = 0, 1, \ldots, N-1$. However, t can only be vertices v_P and v'_P. Since t does not vary very much, it is easier to check whether t is on a cycle, and whether it is possible to reach t from s.

To solve this problem, we create the graph G^T, which is the same as graph G with its edges reversed. Then, we perform a depth-first search on this graph starting at t. During this search, we keep track of the distance of each reachable vertex from t. This number is the distance from s to t in G; that is, the number of steps F that brings us from s to t for the first time. At the same time, if we reach some vertex with a departing edge to t, then we obtain the size of the cycle C, which is the distance from t to that vertex plus 1.

Thus, whether the path in G starting at s with length K ends at t can be determined as following:

- If we cannot reach s in G^T, then this path in G cannot end at t.
- If we reach s in G^T after F steps, but t is not on a cycle, then this path in G ends at t if and only if $K = F$.
- If we reach s in G^T after F steps, and t is in a cycle of size C, then this path in G ends at t if and only if $K = F + nC$ for some non-negative integer n.

For our task, the path starting at the i-th intersection with length K will end at intersection P if and only if the path in G starting at v_i with length K ends at v_P or v'_P. Note that during the implementation, we do not need to create graph G, but we can create G^T directly. It is also convenient to first create an array for storing F_s for each vertex s. We then initialize F_s and C to ∞, and update them during the depth-first search. We perform the search twice, starting at v_P and v'_P, respectively.

Since the depth-first search takes $O(M + N)$ running time and each query can be checked in constant running time, this algorithm takes $O(M + NQ)$ running time in total, which is sufficient to obtain full marks for this task.