In all methods the generation of a valid tune can be achieved by backtracking, so we will focus here only on finding the minimum number of mistakes.

40 points

Let's define $A_{i,j}$ as the maximal amount of notes Linas could have played correctly after playing the first i notes such that the i-th note was note j (j-th in the list of notes). Then you can easily compute all $A_{i,j}$ using the following relation:

$A_{i,j} = \max_{k \to j}(A_{i-1,k}) + \begin{cases} 0, j \neq L_i \\ 1, j = L_i \end{cases}$, where $k \to j$ denotes that it is possible to play note j after note k.

In the worst case (you can play note j after any other note) you will need to check N notes to calculate each $A_{i,j}$, and you need S operations to check if you can play one note after another as well. This leads to a dynamic programming solution with time complexity $O(LN^2S)$. The answer is $L - \max_j(A_{N,j})$.

65 points

Instead of comparing two notes every time you need to whether one note can be played after another, this information can be pre-computed. Make a graph with N vertices corresponding to different notes, draw edges between pairs of notes if they can be played successively (i.e. the strings that to these notes are different in at most G positions), and save the graph as an adjacency matrix to enable checking if there is an edge between a particular pair of notes in O(1) time. The time complexity of this is $O(N^2S)$. Combining this with the earlier algorithm we obtain a solution that runs in $O(N^2S + LN^2)$ time.

65 points – alternative solution

To get to a 100 point solution you need to think differently. Let's take the graph described earlier and use Floyd-Warshall algorithm to compute distances between all pairs of vertices in the graph (distance here corresponds to the minimal time we need to play one note after another (i.e., if the distance between notes A and B is t, we can play B after A only after playing t – 1 or more notes between them). Alternatively, you can run BFS algorithm N times from each vertex; as all the edges in the graph have length 1 you will get the same result. Complexity of this part is $O(N^3)$.

Now let's define A_i as the maximal amount of notes Linas could have played correctly after playing the first i notes such that the i-th note was correct. You can now compute all A_i using the following relation:

$A_i = \max(0, \max_{(k<i) \wedge (|L_k \to L_i| \leq i-k)}(A_k)) + 1$, where $|L_k \to L_i|$ denotes the distance between two notes: L_k and L_i.

The answer is $L - \max_i(A_i)$. The overall complexity of this algorithm is $O(N^2S + N^3 + L^2)$.

100 points

There are two approaches to get to a full solution, but the main idea is the same – to compute A_i you don't need to look back at all previous values $A_{j, j<i}$.

Imagine the graph is connected. It is then sufficient to look only at the previous 2N notes (or, in fact, 2D notes, where D ≤ N is the diameter of the graph). At first note that all shortest paths between notes are at most N. Now, if we pick an arbitrary note from the interval [0, i-2N) it is guaranteed that we can transition to some note in the interval [i-2N, i-N) and play it correctly. Furthermore, we are guaranteed to be able to transition to the note at position I. All this means that if we check all notes in the interval [i-2N, i), it is useless to check earlier notes, i.e., from the interval [0, i-2N).

But by doing this we are assuming that the graph is connected. Otherwise it wouldn't be sufficient to look only at the previous 2N notes, and we would have to consider each connected component of the graph separately, looking at 2n (where n is the size of the particular component) previous notes in that component (which would not change the time complexity of the solution but would make it slightly more difficult to implement). This gives a solution with overall complexity $O(N^2S + N^3 + LN)$.

Another approach is the idea that you need to check only one note of each type. The suggestion is that for each type of note you should select the last occurrence of this note after which you can still manage to make a transition to the current note. You can do so using a binary search, which leads to a solution with overall complexity $O(N^2S + N^3 + LN \log L)$. This is fast enough to score 100 points, however, it is easier to compute the array C, where $C_{i,j}$ is the location of the last note of type j before the position i in the given tune. We get:

$$A_i = \max\left(0, \max_{1 \le j \le N}\left(A_{C_{i-|j \to L_i|,j}}\right)\right) + 1$$

This also gives us a solution with overall complexity $O(N^2S + N^3 + LN)$.