Google Farewell Round 1st Training

CS Chapter IEEE NTUA SB

April 4, 2023

Alien Piano

An alien has just landed on Earth, and really likes our music. Lucky for us. The alien would like to bring home its favorite human songs, but it only has a very strange instrument to do it with: a piano with just 4 keys of different pitches. The alien converts a song by writing it down as a series of keys on the alien piano. Obviously, this piano will not be able to convert our songs completely, as our songs tend to have many more than 4 pitches. The alien will settle for converting our songs with the following rules instead:

- The first note in our song can be converted to any key on the alien piano.
- For every note after,
  - if its pitch is higher than the previous note, it should be converted into a higher-pitched key than the previous note's conversion;
  - if lower, it should be converted into a lower-pitched key than the previous note's conversion;
  - if exactly identical, it should be converted into the same key as the previous note's conversion.

Note: two notes with the same pitch do not need to be converted into the same key if they are not adjacent.

What the alien wants to know is: how often will it have to break its rules when converting a particular song? To elaborate, let us describe one of our songs as having K notes. The first note we describe as "note 1", the second note "note 2", and the last note "note K." So note 2 comes immediately after note 1. Now if note 2 is lower than note 1 in our version of the song, yet converted to an equally-pitched or lower-pitched key (relative to note 2's conversion) in the alien's version of the song, then we consider that a single rule break. For each test case, return the minimum amount of times the alien must necessarily break one of its rules in converting that song.

Input: The first line of the input gives the number of test cases, T. T test cases follow. Each test case consists of two lines. The first line consists of a single integer, K. The second line consists of K space-separated integers, $A_1, A_2, \ldots A_K$, where A_i refers to the pitch of the i-th note for this test case.

Output: For each test case, output one line containing Case #x: y, where x is the test case number (starting from 1) and y is the minimum number of times that particular test case will require the alien to break its own rules during the conversion process.

Limitations

Memory Limit: 1GB
$1 \leq T \leq 100$
$1 \leq A_i \leq 10^6$

Test Set 1:
Time limit: 20 seconds.
$1 \leq K \leq 10$

Test Set 2:
Time limit: 40 seconds.
$1 \leq K \leq 10^4$

Sample Input

```
2
5
1 5 100 500 1
8
2 3 4 5 6 7 8 9
```

Sample Output

```
Case #1: 0
Case #2: 1
```

We will use the notation A, B, C, D for the alien piano keys where A is the lowest note, and D is the highest. In sample case #1, the alien can simply map our song into the following sequence: A B C D C and this correctly reflects all the following:

- our first note with pitch 1 maps to A,
- our second note with pitch 5 maps to its key B. $5 > 1$, and B is a higher key than A,
- our third note with pitch 100 maps to its key C. $100 > 5$, and C is a higher key than B,
- our fourth note with pitch 500 maps to its key D. $500 > 100$, and D is a higher key than C,
- our fifth note with pitch 1 maps to its key C. $1 < 500$, and C is a lower key than D. So none of the rules are broken. Note: A B C D C is not the only way of conversion. A B C D A or A B C D B are also eligible conversions.

In sample case #2, the only conversion sequence that provides the minimal result of 1 rule broken is: A B C D A B C D. Notably, the rule break comes from the fact that our 4th note with pitch 4 is lower than our 5th note with pitch 5, but A is a lower key than D.

Test Set 1

For the small test set, one can generate all possible conversions recursively and select the one with the smallest number of rule violations as the answer. Since each note can be converted to four possible notes in the alien scale, this results into 4^K combinations to be tested and $O(4^K)$ time complexity.

Test Set 2

Dynamic Programming Solution

Let $dp[i][j]$ be the minimum number of rule violations required to convert the first i notes $A_1, A_2, ..., A_i$ such that the i-th note A_i is converted to note j of the alien piano ($1 \leq j \leq 4$). Then the answer is the minimum $dp[K][j]$ over all j, $1 \leq j \leq 4$. The table $dp[i][j]$ can be computed using dynamic programming as follows.

$$dp[i][j] = \begin{cases} 0 & \text{if } i = 1 \\ \min_{1 \leq j' \leq 4}\{dp[i-1][j'] + p(i, j', j)\} & \text{otherwise} \end{cases}$$

Here $p(i, j', j)$ is a binary penalty term accounting for a rule violation between the notes A_{i-1} and A_i. For example, if $A_{i-1} > A_i$, then $p(i, j', j) = 1$ whenever $j' \leq j$ and $p(i, j', j) = 0$ otherwise.

Since each entry of the table is calculated using a constant number of operations, the overall time complexity of the algorithm is $O(K)$, which is okay to pass the large test set.

The implementation of the above algorithm on c++ can be found in alienPianoDp.cpp file.

Greedy Solution

Let us say that a sequence of pitches is *playable* if it can be converted to the alien piano notes without violating any rules. Our goal here is to split the given sequence of pitches into as few playable fragments as possible. We can take the longest playable prefix of the sequence as the first fragment of the split. In this way, the remainder of the sequence is as short as possible, and therefore requires potentially fewer rule violations.

Now let us characterise the playable sequences. Note that repeated notes of the same pitch do not affect the playability of the sequence, therefore, without loss of generality, suppose that any two consecutive notes are at a different pitch. Let us say that two consecutive notes form an *upward step* if the second note has a higher pitch than the first. Otherwise, we call it a *downward step*.

Clearly, a sequence of notes is not playable if it contains more than three consecutive upward or downward steps, as we would run out of the alien scale. Otherwise, the sequence is playable and we can convert it using this simple strategy (assuming the note names A, B, C, and D of the alien piano from the lowest to the highest note):

1. If the first step is upward, convert the first note to A.
2. If the first step is downward, convert the first note to D.
3. If three consecutive notes form an upward step followed by a downward step, convert the second note to D.
4. If three consecutive notes form a downward step followed by an upward step, convert the second note to A.
5. In all other cases, convert a note one step higher or lower than the note before depending on whether they form an upward or downward step.

Since any maximal sequence of upward steps starts at A and has no more than three steps (and similarly for any maximal sequence of downward steps), we will never leave the alien scale. The following diagram illustrates the process of splitting the sequence $\{1, 8, 9, 7, 6, 5, 4, 3, 2, 1, 3, 2, 1, 3, 5, 7\}$ into two playable fragments. Since the subsequence $\{9, 7, 6, 5, 4\}$ consists of four downward steps, the sequence needs to be split between notes 5 and 4.

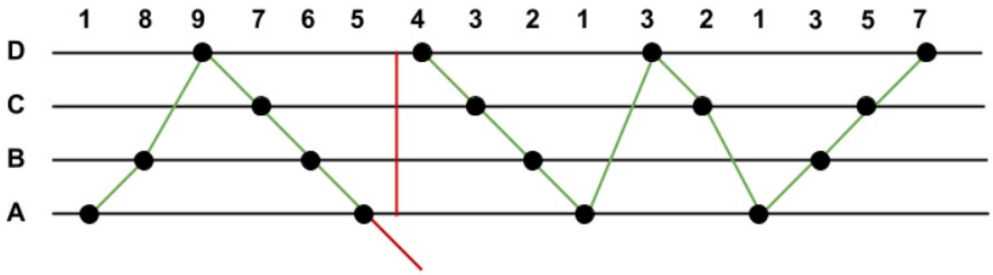

Figure 1: Splitting the sequence $\{1, 8, 9, 7, 6, 5, 4, 3, 2, 1, 3, 2, 1, 3, 5, 7\}$

According to the analysis above, a single pass through the given sequence of notes maintaining the current number of consecutive upward and downward steps results in an $O(K)$ solution, where K is the number of notes in the sequence. As soon as the number of upward or downward steps exceeds 3, we have to split the sequence by violating the rules and start a new fragment.

The implementation of the above algorithm on c++ can be found in alienPianoGreedy.cpp file.