Coding Competitions Farewell Rounds - Round C

Analysis: Game Sort: Part 2

Let N be the length of $\mathbf{S}$.

Test Set 1

In Test Set 1, we have either $\mathbf{P} = 2$ or $\mathbf{P} = 3$. We can solve each of those cases separately:

1. If $\mathbf{P} = 2$, there are exactly $N - 1$ possible places to split $\mathbf{S}$ into two parts. We can try them all, and for each one, check whether Badari can win by using a solution to problem Game Sort: Part 1. Then we know that Amir wins if any of those $N - 1$ places results in a situation where Badari cannot win.
2. If $\mathbf{P} = 3$, there are exactly $\frac{(N-1)(N-2)}{2}$ possible ways to separate: as before, there are $N - 1$ possible places to split, and we must now choose two of those in order to separate $\mathbf{S}$ into three parts. Just like before, one option is to try all such pairs, and for each one, check whether Badari can win.

Such an algorithm has a time complexity $O(N^3 \lg N)$ or perhaps $O(N^3)$ depending on implementation details, so it is enough to pass Test Set 1.

Test Set 2

First of all, we should notice that, whenever $\mathbf{P} = N$, Amir has no choice and $\mathbf{S}$ has to be separated into its N individual letters. Furthermore, for each single-letter part Badari has no choice of rearrangement. Thus, Badari wins in this case if and only if the letters of $\mathbf{S}$ are already sorted in non-decreasing order.

This is a special case to check separately, as many of the properties that we will use later to solve Test Set 2 are false if $\mathbf{P} = N$. From now on, we assume $\mathbf{P} < N$.

We can solve the case $\mathbf{P} = 2$ as in Test Set 1, trying all $N - 1$ possible ways to split $\mathbf{S}$. To avoid doing so in quadratic time, we can keep two arrays with a count of each character per part. Then, when moving from position i to $i + 1$, we only need to increment one count and decrement the other, and use the check from the solution for Game Sort: Part 1. We thus get an $O(N)$ solution for $\mathbf{P} = 2$.

Finally, the only remaining case is $3 \leq \mathbf{P} < N$. Having at least three parts allows Amir to win lots of strings easily. For instance, for any string which does not start with its alphabetically smallest character (let us call it c), Amir can win by simply splitting the first occurrence of c as a single-letter part.

Similarly, if the alphabetically smallest character c appears anywhere on the string that is not the first or second character, it is possible to win by splitting that occurrence of c as a single-letter part, while leaving the first and second characters of the string unsplit (it is possible because $\mathbf{P} < N$).

In the remaining case for $\mathbf{P} \geq 3$, $\mathbf{S}$ has the following property:

- c appears in $\mathbf{S}$ either exactly once as the first character of $\mathbf{S}$, or exactly twice as the first two characters of $\mathbf{S}$.

Let us call a string having that property a *bad* string, and any string without the property *good*. The only remaining case to solve is for bad strings and $\mathbf{P} \geq 3$.

We can assume that Badari plays optimally, and that she follows the optimal strategy explained in the solution of Game Sort: Part 1. That strategy implies that, for a bad string, no matter how we separate $\mathbf{S}$ into parts, after Badari's rearrangement, the first part is lexicographically lower than or equal to all the other parts. This is because the first part will either contain a single c, or every occurrence of c.

Thus, to split a bad string into $\mathbf{P}$ parts, we must choose the length l of the first part, and then separate the remaining $N - l$ characters into $\mathbf{P} - 1$ parts. Note that the first part will be in sorted order, and by the property of bad strings, it will be the lexicographically smallest part and will not impose restrictions on the rearrangement of the second part. So Amir wins such a split if and only if he wins the split of the last $N - l$ characters into $\mathbf{P} - 1$ parts.

Using all these ideas, we can recursively compute, for each suffix of the initial string $\mathbf{S}$ and each k with $2 \leq k \leq \mathbf{P}$, who wins when the game is played for that suffix as initial string and k as the number of parts.

Note that, with this approach, computing who wins for each suffix when splitting into $k = 2$ parts must be done independently as a base case. It is possible to compute all such values efficiently, but it involves even more case analysis and careful implementation, so we will leave it as an exercise for the reader. Such an approach would lead to an overall $O(N \times \mathbf{P})$ time solution, which should pass if implemented carefully.

Although the main idea of this solution is relatively direct, the implementation has many special cases and details, and is tricky to code quickly and correctly. This approach can be implemented more efficiently and in a simpler way, by not storing all of the $N \times \mathbf{P}$ subproblems. Instead, we can store, for each k between 2 and $\mathbf{P}$, the index of the shortest suffix that Amir wins when splitting that suffix into k parts. In a sense, the shortest suffix that works is the only relevant working suffix, because when choosing the length l of a first part to cut, if any winning suffix is available, then the shortest one is. This key observation also avoids having to code the general problem of identifying *all* winning suffixes for $k = 2$, since we only need to find *the shortest* such suffix now, which is quite a simpler problem that we will also leave as an exercise. (Hint: focus only on which characters are lower, equal to, or higher than the very last character of the input string.) With this, we get an $O(N)$ solution.

Another approach

There is another approach which is quite simpler to code, but maybe harder to find and prove correct. We will sketch the algorithm and leave the proof of correctness as an exercise for the reader.

We solve cases $\mathbf{P} = N$ and $\mathbf{P} = 2$ exactly as before. Then we observe that, for $\mathbf{P} \geq 4$, Amir has even more favorable strings:

1. If $\mathbf{S}$ is not sorted, then Amir wins by taking the first consecutive pair of letters in $\mathbf{S}$ that are out of order, and separating them both into single-letter parts.
2. If $\mathbf{S}$ contains three consecutive identical letters, then Amir wins by splitting so that the first two occurrences form a part of length 2, and the third consecutive occurrence forms a single-letter part.
3. If neither (1) nor (2) hold, then it is easy to see that Amir cannot win.

Only the hardest case where $\mathbf{P} = 3$ remains. Suppose that Amir splits $\mathbf{S}$ into three parts, say A, B, C (so $\mathbf{S} = ABC$). It is clear that, if partitioning string AB into A, B is a win for Amir with $\mathbf{P} = 2$ and input string AB, then A, B, C works for $\mathbf{P} = 3$ and input string $\mathbf{S} = ABC$. Similarly,

if partitioning BC into B, C is a win for Amir for $\mathbf{P} = 2$ and input string BC, then also A, B, C for $\mathbf{P} = 3$ and input $\mathbf{S} = ABC$.

The surprising result is that, for any string $\mathbf{S}$, Amir wins if and only if one of these two cases occur. This is not obvious: There are examples of partitions A, B, C that work, but neither A, B nor B, C work; e.g. `XY,AZ,XY` for input string `XYAZXY`. However, in any such case there will be some other partition that works and has this property, e.g. `XY,A,ZXY` for input string `XYAZXY`, as `XY,A` works.

Finally, we only need to check if Amir wins *any* prefix or suffix of $\mathbf{S}$ with $\mathbf{P} = 2$. Note that this is actually simpler than computing for *every* suffix and prefix whether Amir wins with $\mathbf{P} = 2$, and can be done by a slight modification of the full solution to the $\mathbf{P} = 2$ case, which we leave as an exercise.