Kick Start 2022 - Round A

Analysis: Palindrome Free Strings

Test Set 1

Since $\mathbf{N}$ is small in this test set, we can enumerate all different ways to fill in the question marks recursively, which produces at most $2^{\mathbf{N}}$ candidate strings. Then, we check for each string if it contains a palindromic substring of 5 or more characters. If it does not, we have found a solution.

Checking for the presence of palindromic substrings can be done naively, by checking every possible substring, which takes $O(\mathbf{N}^3)$ time ($O(\mathbf{N}^2)$ substrings times $O(\mathbf{N})$ to check if a substring is palindromic). It is possible to achieve a lower complexity by searching for the longest palindromic substring in $O(\mathbf{N}^2)$ or $O(\mathbf{N})$ time.

The overall time complexity will be between $O(2^{\mathbf{N}} \times \mathbf{N})$ and $O(2^{\mathbf{N}} \times \mathbf{N}^3)$ but with the given constraints, even the slowest solution should pass.

Test Set 2

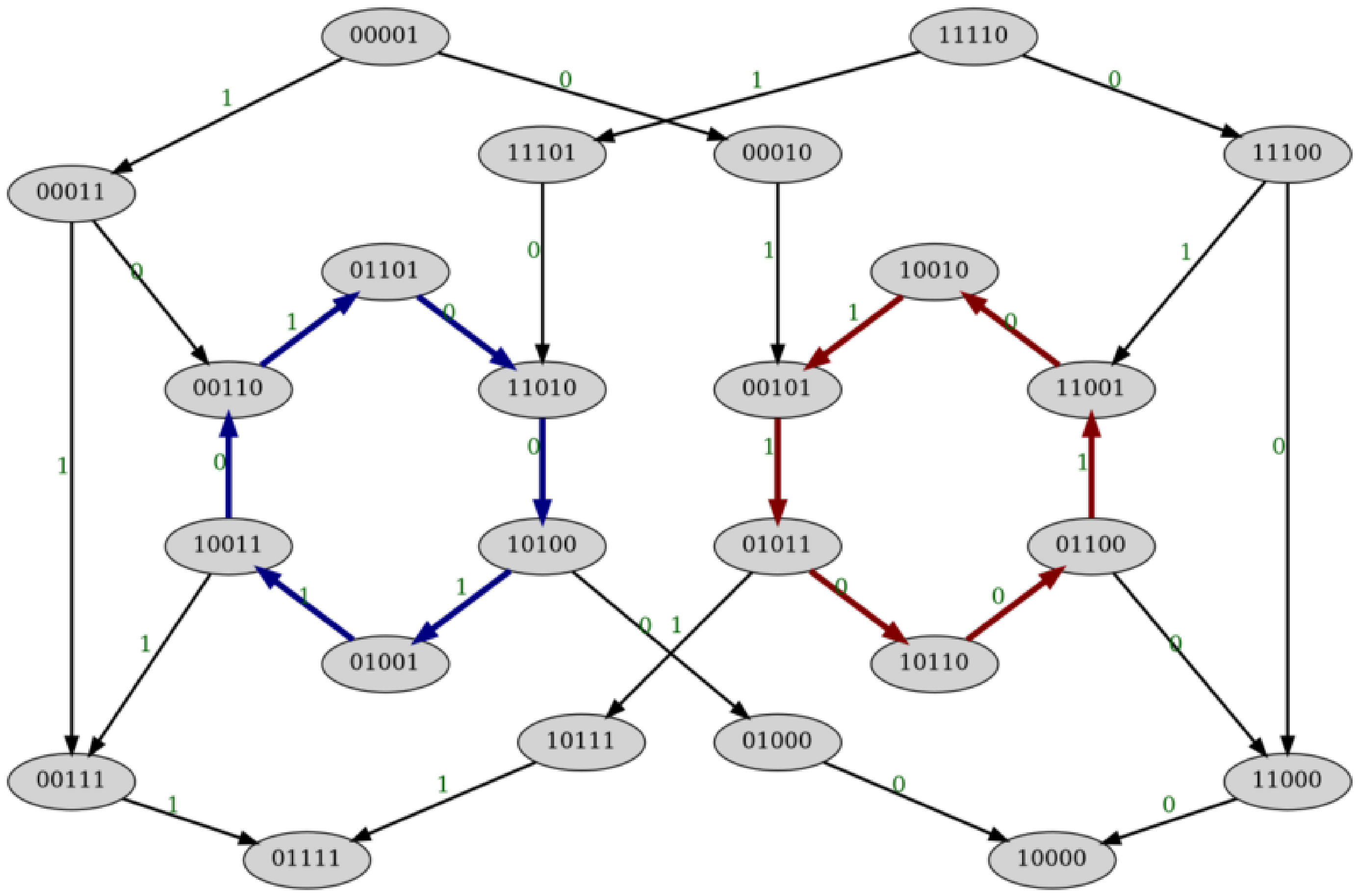
If $\mathbf{N} < 5$ the answer is trivially `POSSIBLE`, so without loss of generality, we will assume $\mathbf{N} \geq 5$ in the following discussion.

Observation 1: If a string contains a palindromic substring P of length $|P| > 6$, then it must also contain a palindromic substring of length 5 or 6. For example, we can take the middle 6 characters of P if $|P|$ is even, and the middle 5 characters otherwise. Consequently, as we fill in the question marks in $\mathbf{S}$, our only concern is to avoid introducing palindromes of length 5 or 6.

If at any point in the recursive solution described above, the last 5 or 6 characters of the prefix generated so far form a palindrome, we can backtrack immediately, thus pruning the search space. If we manage to generate the entire binary string of length $\mathbf{N}$ without forming a palindrome of length 5 or 6, the answer is `POSSIBLE`. Otherwise, we have exhausted the search space with no success, and the answer is `IMPOSSIBLE`.

This reduces the upper bound to $O(2^{\mathbf{N}})$, which is an improvement over the solution for Test Set 1, but it is still prohibitively slow for Test Set 2. It turns out that the complexity is much lower in practice.

Consider again that if we have a valid solution prefix, only its last five characters determine whether it is possible to append a `0` or `1` without introducing a palindrome. That means that all valid strings can be generated by a finite state machine with exactly 24 states (32 five-character strings, minus 8 palindromes). A transition from `abcde` to `bcdef` is possible if and only if neither `abcde` nor `bcdef` is a 5-character palindrome and `abcdef` is not a 6-character palindrome. All state transitions are shown in the directed graph below:



The graph shows, for example, that if the last five characters were `11101` then we can add `0` to obtain `11010`, but we cannot add `1`, which would create the palindrome `11011`. Any valid string can be formed by starting from some vertex (corresponding with the first 5 characters of the string) and then following edges of the graph, appending the labels on the edges to the string.

Observation 2: The graph contains only two cycles, which do not overlap and neither is reachable from the other. That means that the only way to form long strings is by repeating the pattern `001101` (going through the left cycle, colored blue) or `110010` (going through the right cycle, colored red). Since we can start and end at any vertex of the graph, there are a few variations possible at both ends of the string, but since the longest path before entering a cycle has length 2, and the longest path after exiting from a cycle has length 2, all characters in the string except for the first and last two must consist of repetitions of one of the two 6-character patterns mentioned before. This gives an upper bound of $4 \times 12 \times 4 = 192$ palindrome-free strings of any fixed length, though this is an overestimation and it can be shown that the real maximum is only 36.

The resursive solution will generate each possible prefix once. Therefore, it takes $O(M)$ time, where M is the number of palindrome-free prefixes of the solution. We have shown above that $M \leq 36\mathbf{N}$, which makes the overall runtime $O(\mathbf{N})$ and allows it to pass without any additional memory.

Dynamic programming solutions

It seems unlikely that contestants would take the time to draw the state graph before attempting to solve the problem, so they might have missed the second observation. Fortunately, it is also possible to solve the problem with only the first observation, by using dynamic programming.

The simplest way to reduce the runtime of the backtracking solution is to add memoization. If the solution is implemented as a function $f(p)$ where p is a partial solution string, we can replace this with a function $f'(p', i)$ where p' is the 5-character suffix of p and $i = |p|$ (the length of the prefix, or, equivalently, the position in the input). Observation 1 tells us that these representations are equivalent. We can simply cache the results of f' for each pair of arguments which requires $O(2^5 \times \mathbf{N})$ = $O(\mathbf{N})$ time and space.

A second approach is bottom-up dynamic programming. We process the characters in the input from left to right, while maintaining a set of possible prefixes truncated to the last five characters. To process an input character, we calculate a new set by extending the prefixes from the old set. For example, if the set contains `10100` and the next input character is `1` then we can add `01001` to the next set. If the input character were `?`

instead, then we could also add `01000`. If it becomes empty at any point, the answer is `IMPOSSIBLE`. If it is nonempty when we reach the end of the input, then the answer is `POSSIBLE`.

The second approach effectively executes a nondeterministic finite state automaton corresponding with the graph shown above. It is nondeterministic because whenever the input contains a question mark, the transitions labeled with `0` and `1` are both possible, hence the need to track a set of states rather than a single current state only. This approach takes $O(2^5 \times \mathbf{N})$ = $O(\mathbf{N})$ time too, but since we only maintain a single set of possible states, the space requirement is reduced to $O(2^5) = O(1)$.