1625. Lexicographically Smallest String After Applying Operations

Medium | Topics | Companies | Hint

You are given a string `s` of **even length** consisting of digits from `0` to `9`, and two integers `a` and `b`.

You can apply either of the following two operations any number of times and in any order on `s`:

- Add `a` to all odd indices of `s` **(0-indexed)**. Digits post `9` are cycled back to `0`. For example, if `s = "3456"` and `a = 5`, `s` becomes `"3951"`.
- Rotate `s` to the right by `b` positions. For example, if `s = "3456"` and `b = 1`, `s` becomes `"6345"`.

Return *the **lexicographically smallest** string you can obtain by applying the above operations any number of times on* `s`.

A string `a` is lexicographically smaller than a string `b` (of the same length) if in the first position where `a` and `b` differ, string `a` has a letter that appears earlier in the alphabet than the corresponding letter in `b`. For example, `"0158"` is lexicographically smaller than `"0190"` because the first position they differ is at the third letter, and `'5'` comes before `'9'`.

Example 1:

```
Input: s = "5525", a = 9, b = 2
Output: "2050"
Explanation: We can apply the following operations:
Start:  "5525"
Rotate: "2555"
Add:    "2454"
Add:    "2353"
Rotate: "5323"
Add:    "5222"
Add:    "5121"
Rotate: "2151"
Add:    "2050"
There is no way to obtain a string that is lexicographically smaller than
"2050".
```

Example 2:

```
Input: s = "74", a = 5, b = 1
Output: "24"
Explanation: We can apply the following operations:
Start:  "74"
Rotate: "47"
Add:    "42"
Rotate: "24"
There is no way to obtain a string that is lexicographically smaller than
"24".
```

Example 3:

```
Input: s = "0011", a = 4, b = 2
Output: "0011"
Explanation: There are no sequence of operations that will give us a
lexicographically smaller string than "0011".
```

Constraints:

- `2 <= s.length <= 100`
- `s.length` is even.
- `s` consists of digits from `0` to `9` only.
- `1 <= a <= 9`
- `1 <= b <= s.length - 1`

Python:

```
class Solution:
    def findLexSmallestString(self, s: str, a: int, b: int) -> str:

        def dfs(s: str) -> str:

            if s in seen: return

            seen.add(s)
            res, odd =", True

            for ch in s:
                odd^= True
```

```
            res+= d[ch] if odd else ch

        dfs(res)
        dfs(s[b: ] + s[ :b])


    d, seen = {ch:str((int(ch) + a) % 10
            ) for ch in digits}, set()

    dfs(s)

    return min(seen)
```

JavaScript:

```
var findLexSmallestString = function(s, a, b) {
  let set = new Set(), smallest = s;

  let reverse = (str, start, end) => {
    while(start < end) {
      [str[start], str[end]] = [str[end], str[start]];
      start++;
      end--;
    }
    return str;
  };

  let rotate = (str) => {
    str = str.split('');
    str = reverse(str, 0, str.length-1);
    str = reverse(str, 0, b-1);
    str = reverse(str, b, str.length-1);
    return str.join('');
  };

  let add = (str) => {
    str = str.split('');
    for(let i=0; i<str.length; i++) {
      if(i%2 === 1) {
        str[i] = str[i]-'';
        str[i] += a;
        str[i] %= 10;
      }
    }
    return str.join('');
```

```
  };

  let recursive = (str) => {
    if(set.has(str)) {
      return;
    }
    if((str-'') < (smallest-'')) {
      smallest = str;
    }
    set.add(str);
    recursive(rotate(str));
    recursive(add(str));
  };

  recursive(s);
  return smallest;
};
```

Java:

```
class Solution {
    public String findLexSmallestString(String s, int a, int b) {
        int n = s.length();

        // Precompute minimal addition steps for each digit
        int[] bestAdd = new int[10];
        for (int d = 0; d < 10; d++) {
            int minVal = d, minStep = 0;
            for (int step = 1; step < 10; step++) {
                int newVal = (d + a * step) % 10;
                if (newVal < minVal) {
                    minVal = newVal;
                    minStep = step;
                }
            }
            bestAdd[d] = minStep;
        }

        // Determine reachable rotation starts
        boolean[] visited = new boolean[n];
        int idx = 0;
        while (!visited[idx]) {
            visited[idx] = true;
            idx = (idx + b) % n;
        }
```

```
        String answer = s;

        // Try each reachable rotation
        for (int start = 0; start < n; start++) {
            if (!visited[start]) continue;
            String rotated = s.substring(start) + s.substring(0, start);

            int evenAdd = 0, oddAdd = 0;
            if (n == 1) {
                evenAdd = bestAdd[rotated.charAt(0) - '0'];
            } else {
                evenAdd = (b % 2 == 1) ? bestAdd[rotated.charAt(0) - '0'] : 0;
                oddAdd  = bestAdd[rotated.charAt(1) - '0'];
            }

            StringBuilder sb = new StringBuilder(rotated);
            for (int j = 0; j < n; j++) {
                int d = sb.charAt(j) - '0';
                int times = (j % 2 == 0) ? evenAdd : oddAdd;
                d = (d + times * a) % 10;
                sb.setCharAt(j, (char)('0' + d));
            }

            String candidate = sb.toString();
            if (candidate.compareTo(answer) < 0) answer = candidate;
        }

        return answer;
    }
}
```