1625. Lexicographically Smallest String After Applying Operations Breadth-First Search String Medium Leetcode Link

Problem Description

of s and moving them to the front.

are also provided with two integers a and b. Our task is to transform s into the lexicographically smallest possible string by repeatedly applying two types of operations in any order: 1. Add Operation: We can add the integer a to each digit at an odd index (where indexing starts from 0). If adding a causes the

In this problem, we are given a string s representing a sequence of digits from 0 to 9. The length of 5 is guaranteed to be even. We

- digit to become greater than 9, it wraps around to 0 and continues from there (essentially we take the sum modulo 10). 2. Rotate Operation: We can rotate the string to the right by b positions. Rotating by b positions means taking the last b characters
- A lexicographically smaller string is defined as one that would come first in a dictionary. For example, '2034' is lexicographically smaller than '2035' because it has a smaller digit ('4' instead of '5') in the last position where they differ.

The aim, therefore, is to find the sequence of operations that, when applied to s, will result in the lexicographically smallest string.

Intuition The challenge of this problem lies in determining the correct order and amount of operations to apply since both rotate and add

Since s is of an even length, there are two distinct cases for the rotate operation:

1. If b is even, rotating the string does not change the positions of the odd and even indexed characters relative to each other. 2. If b is odd, each rotation swaps the positions of odd and even indexed characters.

Rotating the string by its length n or a multiple of it returns the string to its original configuration. Thus, we need to rotate the

operations can be done an unlimited number of times. The brute force approach of trying every possible combination of operations is

string at most n-1 times to explore all unique configurations.

Given these properties, we can infer a few strategies that help reduce the problem space:

not feasible due to the potentially massive number of combinations.

operation to odd indices when b is even.

- Since rotating by an even b doesn't change the relative positions of characters, we only need to focus on applying the add
- However, when b is odd, because rotating changes the parity of the indices, we will need to apply the add operation to all characters, both at odd and even positions.

The solution provided employs a breadth-first search (BFS) type of approach. Here are the steps:

3. If the rotation step is such that b is odd, extend the same add operation to even indexed characters.

- 1. Apply the rotation operation one step at a time, each time attempting to compute a lexicographically smaller result. 2. After each rotate, apply the add operation repeatedly (up to 10 times because after 10 additions we will have cycled through all possible digit values) to the odd indexed characters, checking each time if a smaller string is achieved.
- By repeatedly applying this sequence of operations, we ensure that we explore all possible combinations of rotations and additions without redundancy. We keep track of the smallest string seen so far and update this minimum whenever we find a smaller one.
- This process is guaranteed to terminate because there are a finite number of possible strings (since s is of finite length), and every step is designed to make progress towards a lexicographically smaller string.

The solution provided adopts a method similar to breadth-first search (BFS). In graph theory, BFS is an algorithm for traversing or

searching tree or graph data structures. It starts at an arbitrary node and explores all of the neighbor nodes at the present depth

prior to moving on to nodes at the next depth level. In our case, the nodes represent different versions of the string we obtain after

Solution Approach

We initialize ans with the original string s. This will hold the smallest string found during the search.

A Python list s is created from the original string to allow for mutable operations. There is an outer loop that permits us to try every possible rotation on the string. We can cycle through all the rotations by

b = 2.

each rotation and addition operation.

The variable n stores the length of the string.

rotating one step at a time, up to n times.

If t is lexicographically smaller than ans, we update ans.

Finally, after all possible transformations are tried, we return ans.

Here's how the solution works:

• The state of s is adjusted with s = s[-b:] + s[:-b]. This line effectively rotates the string to the right by b positions. An inner loop (ranging from 0 to 9) applies the addition operation to all odd indices of s. This is done 10 times because after 10 additions, the character would have been incremented by a ten times, which is equivalent to adding 0 considering the modulo operation. • If the shift b is odd (b & 1), we account for the switch between the odd and even indices due to the rotations. Yet another

After each application of the addition operation, we join the characters into a string t using ''.join(s) and compare it with ans.

BFS can be a quite wide exploration, but in this algorithm, we are keeping only one copy of the string at each level of depth, rather

nested loop runs for 10 iterations to apply the addition operation to what were originally even indices before rotation.

current smallest string and, by domain constraints, we know there's no need to revisit a string we have already considered. The Python list structure allows us to perform rotates and additions very efficiently, which is key in handling possibly numerous

than exploring multiple strings in parallel. This is an efficient way to handle the problem since we are always iterating from the

string possible. Example Walkthrough

Let's use a simple example to demonstrate the solution approach. Suppose we are given a string s = "4625", and integers a = 3, and

The string s is of length 4, which is even, and we see that the rotation parameter b is also even. Hence, rotating the string will not

change odd and even positions relative to each other. Let's follow the steps outlined in the solution approach:

iterations. The algorithm effectively traverses a space of n * 10 * 10 possible strings (where n is the number of rotations and 10

comes from the number of potential additions to both odd and even indices), ensuring that we find the lexicographically smallest

 Add a=3 to each digit at odd indices (indexing starts at 0): ■ Apply addition operation: s[1] = (5 + 3) % 10 = 8 and s[3] = (6 + 3) % 10 = 9. Now s = "2849". Update ans since "2849" is lexicographically smaller than "4625". As b is even, we don't need to perform addition on even indices this round. 2. Second Rotation: 5 = "4928" (rotated right again by b=2 from original, so it is "2546" -> "4928")

• Apply addition operation: s[1] = (9 + 3) % 10 = 2 and s[3] = (8 + 3) % 10 = 1. Now s = "4212".

Apply addition operation: s[1] = (2 + 3) % 10 = 5 and s[3] = (9 + 3) % 10 = 2. Now s = "1522".

4. Fourth Rotation: s = "2149" (rotated right again by b=2 from original, so it is "4625" -> "2149")

"4212".

Python Solution

class Solution:

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

35

36

6

8

9

10

17

18

19

20

21

22

23

24

25

26

27

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

40

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

32

33

34

35

36

37

38

40

41

42

43

47

48

Time Complexity

of rotation.

39 };

} else {

return smallestLexString;

1 // Define the string length variable globally.

for (let i = 0; i < stringLength; ++i) {</pre>

for (let j = 0; j < 10; ++j) {

if (b % 2 === 1) {

} else {

return smallestLexString;

Time and Space Complexity

// Rotate the string by 'b' positions.

Typescript Solution

2 let stringLength: number;

Start with ans = "4625", the original string s.

1. First Rotation: s = "2546" (rotated right by b=2)

Add a=3 to each digit at the odd indices:

Add a=3 to each digit at the odd indices:

As b is even, we skip addition to even indices again.

Update ans since "4212" is lexicographically smaller than "2849".

ans remains "4212" because "1522" is not lexicographically smaller.

to find the smallest string through efficient updating. The final answer is "4212".

def find_lex_smallest_string(self, s: str, a: int, b: int) -> str:

Initialize answer with the initial string.

Change the string n times, which ensures that

Add 'a' to every digit at odd indices.

for k in range(1, n, 2):

Rotate the string based on the value of b.

s[k] = str((int(s[k]) + a) % 10)

for k in range(0, n, 2):

current = ''.join(s)

if answer > current:

public String findLexSmallestString(String s, int a, int b) {

// Perform rotation by 'b' characters to the left

for (int k = 1; k < length0fS; k += 2) {

for (int p = 0; p < 10; ++p) {

// Rotating the string s for 'lengthOfS' times

for (int i = 0; i < length0fS; ++i) {</pre>

if ((b & 1) == 1) {

answer = current

all rotations are visited.

s = s[-b:] + s[:-b]

if b % 2 == 1:

// The length of the string s

int lengthOfS = s.length();

String answer = s;

for _ in range(n):

return answer

3. Third Rotation: s = "1249" (rotated right again by b=2 from original, so it is "4625" -> "1249")

We begin with the outer loop, attempting all possible rotations:

The length of the string n = 4.

 Add a=3 to each digit at the odd indices: • Apply addition operation: s[1] = (1 + 3) % 10 = 4 and s[3] = (9 + 3) % 10 = 2. Now s = "2422". ans remains "4212" because "2422" is not lexicographically smaller.

5. After the loop completes, we have tried all rotations and addition operations. Our ans holds the lexicographically smallest string:

Thus, following the BFS-like strategy, we have successfully explored all the possible rotations and have applied addition operations

answer = s# Get the length of the input string. n = len(s)# Convert the string to a list for easier manipulation. s = list(s)

for _ in range(10): # Repeat this operation up to 10 times, one for each digit.

since rotation places previously odd indexed numbers at even indices.

Join the list to form a string and compare with the answer.

If 'b' is odd, addition is also applicable at even indices,

// The variable 'answer' will hold the lexicographically smallest string found

// Only add to odd indexed characters of the array starting from 1

// This time add 'a' to even indexed characters

s[k] = ((s[k] - '0' + a) % 10) + '0';

smallestLexString = min(smallestLexString, s);

smallestLexString = min(smallestLexString, s);

// Return the lexically smallest string after trying all possibilities.

function findLexSmallestString(s: string, a: number, b: number): string {

stringLength = s.length; // Set the length of the string.

for (let k = 1; k < stringLength; k += 2) {

for (let p = 0; p < 10; ++p) {

function minLexString(str1: string, str2: string): string {

return str1.localeCompare(str2) < 0 ? str1 : str2;

let digit = (parseInt(s[k]) + a) % 10;

// Update smallestLexString if current string 's' is smaller.

// Function to find the lexicographically smallest string after applying rotation and addition operations.

// Perform rotation and increment operations to find the smallest lexicographical string.

// Increment each digit at odd indices by 'a', using modulo 10 for wrap around.

s = s.substring(stringLength - b) + s.substring(0, stringLength - b);

for (let k = 0; k < stringLength; k += 2) {

let digit = (parseInt(s[k]) + a) % 10;

// Try incrementing the digits at odd indices and find the smallest string.

s = s.substring(0, k) + digit.toString() + s.substring(k + 1);

// If 'b' is odd, also apply incrementation to digits at even indices.

let smallestLexString: string = s; // Initialize the smallest lexicographical string found with the original string.

// Increment each digit at even indices by 'a', using modulo 10 for wrap around.

s = s.substring(0, k) + digit.toString() + s.substring(k + 1);

// Update smallestLexString if current string 's' is smaller.

smallestLexString = minLexString(smallestLexString, s);

// Update smallestLexString if current string 's' is smaller.

smallestLexString = minLexString(smallestLexString, s);

// Return the lexically smallest string after trying all possibilities.

// Helper function to find the smaller of two lexicographically ordered strings.

under given operations. The time complexity can be broken down into multiple parts:

// Update smallestLexString if current string 's' is smaller.

for (int k = 0; k < length0fS; k += 2) {

charArray[k] = (char) (((charArray[k] - '0' + a) % 10) + '0');

for _ in range(10): # Repeat for even indices.

s[k] = str((int(s[k]) + a) % 10)

29 # If 'b' is even, only update based on odd indices' additions. 30 else: 31 current = ''.join(s) 32 if answer > current: 33 answer = current 34 # Return the lexicographically smallest string obtained.

Update the answer if the current string is lexicographically smaller.

11 s = s.substring(b) + s.substring(0, b);12 // Convert string s to a character array for manipulation char[] charArray = s.toCharArray(); 13 14 15 // Try adding 'a' to every digit (10 times since there are only 10 possibilities for a digit) 16 for (int j = 0; j < 10; ++j) {

Java Solution

1 class Solution {

```
28
 29
                             // Create new string from the character array
 30
                             s = String.valueOf(charArray);
 31
                             // Update the answer if a new lexicographically smaller string is found
 32
                             if (answer.compareTo(s) > 0) {
 33
                                 answer = s;
 34
 35
 36
                     } else {
 37
                         // If 'b' is even, create a new string and potentially update the answer
 38
                         s = String.valueOf(charArray);
                         if (answer.compareTo(s) > 0) {
 39
 40
                             answer = s;
 41
 42
 43
 44
 45
             // Return the lexicographically smallest string found
 46
             return answer;
 47
 48
 49
C++ Solution
  1 class Solution {
  2 public:
         string findLexSmallestString(string s, int a, int b) {
             int stringLength = s.size();
                                                      // Length of the string.
             string smallestLexString = s;
                                                      // The lexically smallest string found so far.
  5
  6
             // Perform rotation and replacement operations to find the smallest lexicographical string.
             for (int i = 0; i < stringLength; ++i) {</pre>
  8
                 // Rotate the string by 'b' positions.
  9
 10
                 s = s.substr(stringLength - b) + s.substr(0, stringLength - b);
 11
 12
                 // Try incrementing the digits at odd indices and finding the smallest string.
 13
                 for (int j = 0; j < 10; ++j) {
 14
                     // Increment each digit at odd index 'k' by 'a', and handle wrap around using modulo 10.
 15
                     for (int k = 1; k < stringLength; k += 2) {
 16
                         s[k] = ((s[k] - '0' + a) % 10) + '0';
 17
 18
 19
                     // Check if 'b' is odd, if yes, also increment the digits at even indices.
 20
                     if (b % 2 == 1) {
 21
                         for (int p = 0; p < 10; ++p) {
 22
                             // Increment each digit at even index 'k' by 'a', and handle wrap around using modulo 10.
 23
                             for (int k = 0; k < stringLength; k += 2) {
```

// If 'b' is odd, we need to handle even indexed characters as well because rotation affects them

charArray[k] = (char) (((charArray[k] - '0' + a) % 10) + '0');

```
26
27
28
29
30
31
```

check. 2. For every rotation, we have an inner loop running 10 times to increment the digits at odd indices.

- 4. Within the second and third bullet point loops, there is an iteration over the characters of the string in steps, which is O(n). Considering these nested loops and their iteration counts, the time complexity can be calculated as follows:
 - If b is odd, we must consider additional inner iterations for increasing even indices, which adds another factor of 10, leading to

The given code performs a series of rotations and digit increments to find the lexicographically smallest string that can be achieved

1. The outermost loop runs at most n times where n is the length of the string, since that's the number of possible rotations we can

3. When b is odd, an additional loop over 10 iterations is executed for each of the outer loop iterations to increase the even indices.

• If b is even, the complexity is $0(n * 10 * n) = 0(n^2)$ because the changes are applied to odd indices regardless of the number

As the b being even or odd determines whether the innermost loop is executed, the overall time complexity is $0(n^2)$.

- complexity is O(n).
- $0(n * 10 * 10 * n) = 0(100 * n^2) = 0(n^2)$, as constant factors are dropped in time complexity considerations.
- Space Complexity The space complexity of the code is primarily due to the storage of the string s as a list and the temporary storage of the modified string t.
 - There is no additional space used that grows with the input, as the loops use a constant amount of space. Therefore, the space
- The list converted string takes O(n) space. The temporary string t also takes O(n) space.