1850. Minimum Adjacent Swaps to Reach the Kth Smallest Number

Leetcode Link

Problem Explanation

In this problem, we are given a string `num` that represents a large integer, and an integer `k`. The task is to find the minimum number of adjacent digit swaps that needs to be applied to `num` to reach the `k`th smallest wonderful integer. A wonderful integer is an integer that is a permutation of the digits in `num` and is greater in value than `num`.

Approach

The solution follows this approach:

1. Generate the `k`th smallest wonderful integer using a permutation algorithm such as next_permutation.
2. Count the minimum number of adjacent digit swaps needed to transform the original string into the generated permutation.

Let's walk through an example.

Example

```
num = "5489355142"
k = 4
```

1. We will first find the `k`th smallest wonderful integer using next_permutation.
    - After 1st permutation: "5489355214"
    - After 2nd permutation: "5489355241"
    - After 3rd permutation: "5489355412"
    - After 4th permutation: "5489355421"

    So, the 4th smallest wonderful integer is "5489355421".
2. Now, we will count the minimum number of adjacent digit swaps needed.
    1. Swap index 7 with index 8: "5489355142" -> "5489355412"
    2. Swap index 8 with index 9: "5489355412" -> "5489355421"

    The total number of swaps required is 2.

Now, let's implement the solution in different languages.

Solution in Python

```python
from itertools import permutations

class Solution:
    def getMinSwaps(self, num: str, k: int) -> int:
        perm = num

        for _ in range(k):
            perm = self.next_permutation(perm)

        return self.count_steps(num, perm)

    def count_steps(self, A: str, B: list) -> int:
        count = 0

        for i in range(len(A)):
            j = i
            while A[i] != B[j]:
                j += 1
            while i < j:
                B[j], B[j - 1] = B[j - 1], B[j]
                j -= 1
                count += 1

        return count

    def next_permutation(self, s: str) -> list:
        s = list(s)
        n = len(s)
        i = n - 1
        while i > 0 and s[i - 1] >= s[i]:
            i -= 1

        j = n - 1
        while i > 0 and s[i - 1] >= s[j]:
            j -= 1
        s[i - 1], s[j] = s[j], s[i - 1]
        s[i:] = reversed(s[i:])

        return s
```

Solution in Java

```java
import java.util.*;

class Solution {
    public int getMinSwaps(String num, int k) {
        List<Character> perm = new ArrayList<>();

        for (char ch : num.toCharArray())
            perm.add(ch);

        for (int i = 0; i < k; ++i)
            nextPermutation(perm);

        return countSteps(num, perm);
    }

    private int countSteps(String A, List<Character> B) {
        int count = 0;

        for (int i = 0, j = 0; i < A.length(); ++i) {
            j = i;
            while (A.charAt(i) != B.get(j))
                ++j;

            while (i < j) {
                Collections.swap(B, j, j - 1);
                --j;
                ++count;
            }
        }
        return count;
    }

    private void nextPermutation(List<Character> perm) {
        int i = perm.size() - 1;

        while (i > 0 && perm.get(i - 1) >= perm.get(i))
            --i;

        int j = perm.size() - 1;
        while (i > 0 && perm.get(i - 1) >= perm.get(j))
            --j;

        Collections.swap(perm, i - 1, j);
        Collections.reverse(perm.subList(i, perm.size()));
    }
}
```

Solution in C++

```cpp
#include <algorithm>
#include <string>
#include <vector>

class Solution {
 public:
  int getMinSwaps(std::string num, int k) {
    std::string perm = num;

    while (k--)
      std::next_permutation(begin(perm), end(perm));

    return countSteps(num, perm);
  }

 private:
  int countSteps(const std::string &A, std::string &B) {
    int count = 0;

    for (size_t i = 0, j = 0; i < A.length(); ++i) {
      j = i;
      while (A[i] != B[j])
        ++j;

      while (i < j) {
        std::swap(B[j], B[j - 1]);
        --j;
        ++count;
      }
    }
    return count;
  }
};
```

Solution in C#

```csharp
using System;
using System.Collections.Generic;

public class Solution {
    public int GetMinSwaps(string num, int k) {
        List<char> perm = new List<char>(num);

        for (int i = 0; i < k; ++i)
            NextPermutation(perm);

        return CountSteps(num, perm);
    }

    private int CountSteps(string A, IList<char> B) {
        int count = 0;

        for (int i = 0, j = 0; i < A.Length; ++i) {
            j = i;
            while (A[i] != B[j])
                ++j;

            while (i < j) {
                Swap(B, j, j - 1);
                --j;
                ++count;
            }
        }

        return count;
    }

    private void NextPermutation(IList<char> perm) {
        int i = perm.Count - 1;

        while (i > 0 && perm[i - 1] >= perm[i])
            --i;

        int j = perm.Count - 1;
        while (i > 0 && perm[i - 1] >= perm[j])
            --j;

        Swap(perm, i - 1, j);
        Reverse(perm, i, perm.Count);
    }

    private void Swap(IList<char> list, int i, int j) {
        char tmp = list[i];
        list[i] = list[j];
        list[j] = tmp;
    }

    private void Reverse(IList<char> list, int start, int end) {
        for (int i = start, j = end - 1; i < j; ++i, --j)
            Swap(list, i, j);
    }
}
```

```
## JavaScript Implementation

```javascript
class Solution {
    getMinSwaps(num, k) {
        let perm = num.split('');

        for (let i = 0; i < k; i++) {
            this.nextPermutation(perm);
        }

        return this.countSteps(num, perm);
    }

    countSteps(A, B) {
        let count = 0;

        for (let i = 0; i < A.length; i++) {
            let j = i;
            while (A[i] !== B[j]) {
                j++;
            }

            while (i < j) {
                [B[j], B[j - 1]] = [B[j - 1], B[j]];
                j--;
                count++;
            }
        }

        return count;
    }

    nextPermutation(perm) {
        let i = perm.length - 1;

        while (i > 0 && perm[i - 1] >= perm[i]) {
            i--;
        }

        let j = perm.length - 1;
        while (i > 0 && perm[i - 1] >= perm[j]) {
            j--;
        }

        [perm[i - 1], perm[j]] = [perm[j], perm[i - 1]];
        perm.splice(i, perm.length - i, ...perm.slice(i).reverse());
    }
}

let sol = new Solution();
let num = "5489355142";
let k = 4;
console.log(sol.getMinSwaps(num, k));  // Output: 2
```

The JavaScript implementation follows a similar approach to other languages. We first convert the input string `num` into an array of characters and then apply the `nextPermutation` function `k` times. After that, we count the number of swaps required using the `countSteps` function. Finally, we return the swap count.

Got a question? Ask the Teaching Assistant anything you don't understand.
```