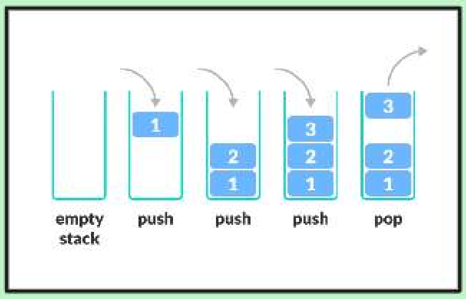
Remove K Digits (Leetcode-402)

@manojofficialmj



Leetcode - 402
Remove K digits

Ex1 Num = "1432219" K = 3
output = "1219"
} case-1

Ex2 Num = "123456" K = 3
output = "123"
} case-2

Ex3 Num = "10200" K = 1
output = "200"

Ex4 Num = "0000" K = 1
output = "0"
} case-3

Ex5 Num = "10" K = 2
output = "0"
} case-4

Note sub-string length = N-K

Intuition:

① Remove K digits from Left for No increasing order like

EX Num1 = 785
Num2 = 685

which one is smallest number?

→ Remove left most digit when (7>6) to get the smallest number is $\boxed{\text{NUM2} = 685}$

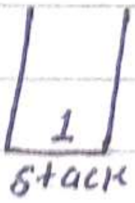
Ex 1. Num = "1432219" and K = 3

Initial state

K = 3

Iteration1

digit = 1 K = 3

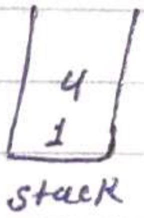


Iteration2

digit = 4 K = 3 digit > stack.Top()

4 > 1

↳ push(4)

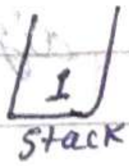


Iteration3

digit = 3 K = 2 digit < stack.Top()

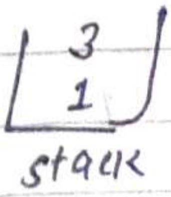
3 < 4

↳ stack.pop()



3 < 1 X

↳ push(3)

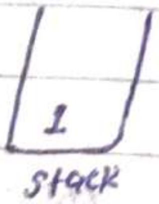


Iteration4

digit = 2   K = 1   digit < stack.Top()

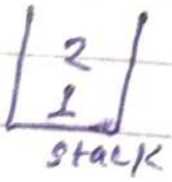
2 < 3

↳ stack.pop()



2 < 1 ✗

↳ stack.push(2)

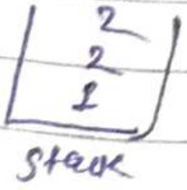


Iteration5

digit = 2   K = 1   digit < stack.Top()

2 < 2 ✗

↳ push(2)

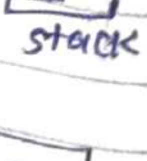


Iteration6

digit = 1   K = 0   digit < stack.Top()

1 < 2

↳ pop()

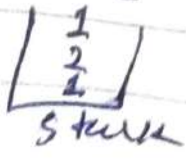


1 < 2

↳ STOP $\boxed{K == 0}$

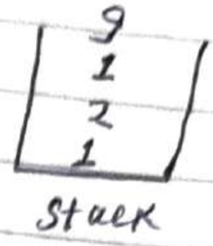
↳ K digits Remove kar chuke hai

push(1)



iteration 7

digit = 9   k=0



Final output

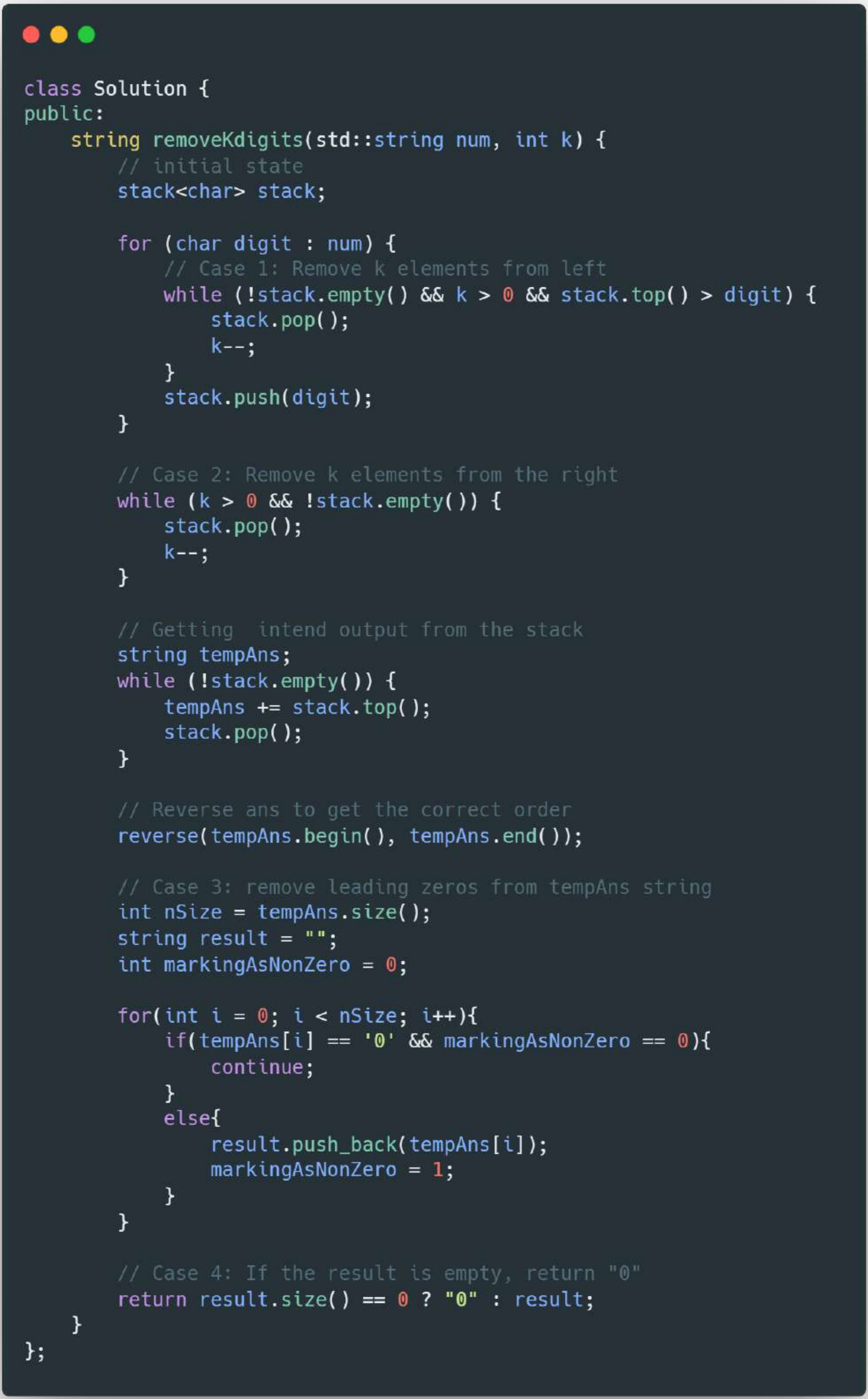
Reverse Stack ⇒ [1 2 1 9]

↗

This is smallest number after removing 3 digits from num.

Time Complexity = $O(N)$

Space Complexity = $O(N)$



```cpp
class Solution {
public:
    string removeKdigits(std::string num, int k) {
        // initial state
        stack<char> stack;

        for (char digit : num) {
            // Case 1: Remove k elements from left
            while (!stack.empty() && k > 0 && stack.top() > digit) {
                stack.pop();
                k--;
            }
            stack.push(digit);
        }

        // Case 2: Remove k elements from the right
        while (k > 0 && !stack.empty()) {
            stack.pop();
            k--;
        }

        // Getting  intend output from the stack
        string tempAns;
        while (!stack.empty()) {
            tempAns += stack.top();
            stack.pop();
        }

        // Reverse ans to get the correct order
        reverse(tempAns.begin(), tempAns.end());

        // Case 3: remove leading zeros from tempAns string
        int nSize = tempAns.size();
        string result = "";
        int markingAsNonZero = 0;

        for(int i = 0; i < nSize; i++){
            if(tempAns[i] == '0' && markingAsNonZero == 0){
                continue;
            }
            else{
                result.push_back(tempAns[i]);
                markingAsNonZero = 1;
            }
        }

        // Case 4: If the result is empty, return "0"
        return result.size() == 0 ? "0" : result;
    }
};
```