402. Remove K Digits

Sept. 8, 2019 | 29.8K views

Given a non-negative integer *num* represented as a string, remove *k* digits from the number so that the new number is the smallest possible.

Note:

- The length of *num* is less than 10002 and will be ≥ *k*.
- The given *num* does not contain any leading zero.

Example 1:

```
Input: num = "1432219", k = 3
Output: "1219"
Explanation: Remove the three digits 4, 3, and 2 to form the new number 1219 which is
```

Example 2:

```
Input: num = "10200", k = 1
Output: "200"
Explanation: Remove the leading 1 and the number is 200. Note that the output must not
```

Example 3:

```
Input: num = "10", k = 2
Output: "0"
Explanation: Remove all the digits from the number and it is left with nothing which i
```

Solution

Approach 1: Brute-force [Time Limit Exceeded]

Intuition

At the first glance, one of the first intuitions that might come to one's mind is to enumerate all the possible combinations and find the minimal number among them, *i.e. brute-force*.

Though after a small moment of reflection, we would easily rule it out. We could name a few reasons. The major caveat is that the algorithm would have an **exponential time complexity**, since we need to enumerate the combinations of selecting k numbers out of a list of n, *i.e.* C_n^k. Even for a trial example, the algorithm could run out of the time limit.

Apart from the complexity issue, another technical issue that one needs to solve in the above brute-force approach, is *to compare the values of two digit strings*. As naive as it sounds, we could convert the digit string to the numerical value. Soon one would realize that this method does not scale. For an unsigned 32 bit-integer, the maximum value it can hold is a number with 10 digits (*i.e. 4,294,967,295*). We can expect there are plenty of test cases with strings of hundreds of digits.

One would argue that for comparison, we don't need to convert the digit string to its numeric value, but simply compare the sequence of digits one by one from left to right. Indeed, it would work.

But then, if we look at the overall problem again, it seems that there should be some ***deterministic way*** *to construct the solution, without the need of exhausting all possible solutions.*

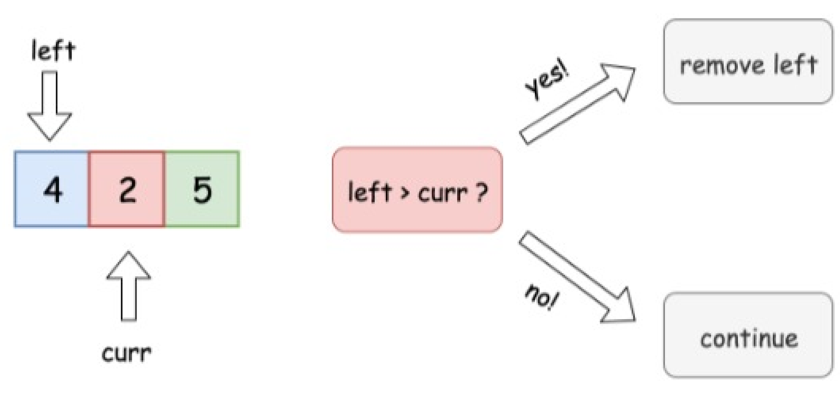
Approach 2: Greedy with Stack

Intuition

We've got a hint while entertaining the idea of brute-force, that given two sequences of digit of the same length, it is the **leftmost distinct** digits that determine the superior of the two numbers, *e.g.* for `A = 1axxx`, `B = 1bxxx`, if the digits `a > b`, then `A > B`.

With this insight, the first intuition we got for our problem is that *we should iterate from the left to right*, when removing the digits. The more a digit to the left-hand side, the more weight it carries.

Now that we fix on the order of the iteration, it is critical to come up some **criteria** on how we eliminate digits, in order to obtain the minimum value.



Let us start from a simple example. Given a sequence of digits, *e.g.* `425`, if we are asked to remove just one digit, then from left to right, we have the candidates as `4`, `2` and `5`. And we compare each digit with its left neighbor. Starting from `2`, which is less than its left neighbor `4`. *At this very moment, we are sure that we should remove the digit* `4`. Because the consequence of not doing so is that we won't obtain the minimum number, no matter what we do subsequently.

Imagine if we keep the digit `4`, then all the possible solutions are lead with the digit `4` (*i.e.* 42, 45). While in one of the opposite cases, *e.g.* removing `4` and keeping `2`, we have solutions lead with `2` (*i.e.* 25), which is obviously less than any of the solutions of keeping the digit `4`.

We could summarize the above scenario of removing a digit, as a rule below:

> Given a sequence of digits $[D_1 D_2 D_3 \ldots D_n]$, if the digit D_2 is less than its left neighbor D_1, then we should remove the left neighbor (D_1) in order to obtain the minimum result.

Algorithm

Believe it or not, the above rule is the only key needed to solve the problem. It clearly defines a condition on which we can remove a digit without a doubt. By removing the digits one by one, we are steadily approaching the optimal solution step by step. Now, it might ring a bell, to one of the popular algorithmic paradigms — **Greedy**.

> Indeed, the problem could be solved with the greedy algorithm. The above rule clarifies the essential logic on how we can approach the final solution. Once we remove a digit from the sequence, the remaining digits forms a new problem where we can continue to apply the rule.

One might notice that, there could be some cases where the condition to apply the rule does not hold for any of the digits. To put it in another word, in those cases, we would have a ***monotonic increasing sequence***, *i.e.* each digit is bigger than its previous digit. In this scenario, we simply remove the pending large digits, again, ***greedily***. We skip the rigorous proof here, and claim that the solution obtained by the above measure is indeed the optimal one.

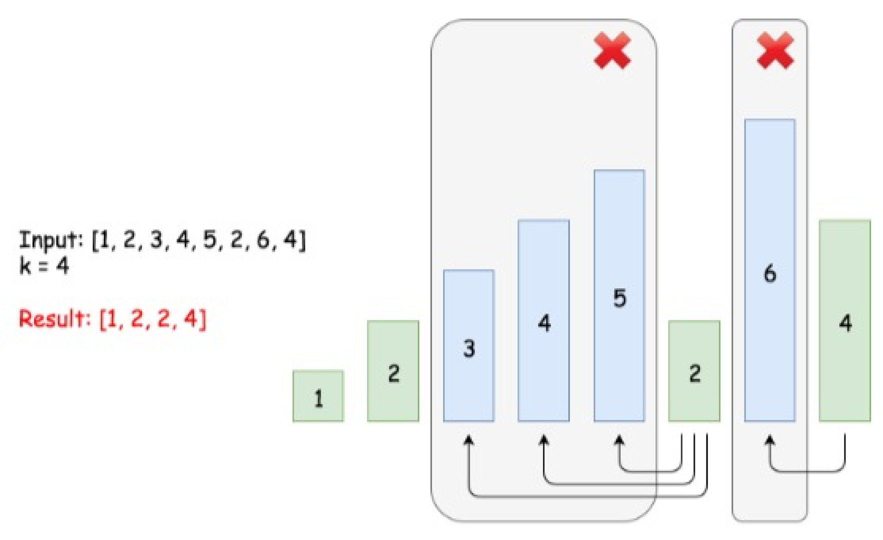
On the other hand, we did provide a ***proof by contradiction***, with the simple example of `425` in the Intuition section, that by repeatedly applying the rule we would obtain the optimal solution.

Implementation

One could implement the above algorithm with the help of the **stack** data structure. We use a stack to hold the digits that we would keep at the end.

Iterating the sequence of digits from left to right, the main loop can be broken down as follows:

- 1). for each digit, if the digit is less than the top of the stack, *i.e.* the left neighbor of the digit, then we pop the stack, *i.e.* removing the left neighbor. At the end, we push the digit to the stack.
- 2). we repeat the above step (1) until any of the conditions does not hold any more, *e.g.* the stack is empty (no more digits left). or in another case, we have already removed `k` digits, therefore mission accomplished.



We demonstrate how the algorithm works in the above graph. Given the input sequence `[1, 2, 3, 4, 5, 2, 6, 4]` and the number of digits to remove `k=4`, the rule is triggered for the first time at the digit of `5`. Once we remove the digit `5`, the rule is triggered again at the digit `4` till the digit `3`. And then later, at the digit `6`, the rule is triggered as well.

Out of the above main loop, we need to handle some corner cases to make the solution more complete.

- case 1). when we get out of the main loop, we removed `m` digits, which is *less than asked*, *i.e.*(`m < k`). In the extreme case, we would not remove any digit for the monotonic increasing sequence in the loop, *i.e.* `m==0`. In this case, we just need to remove the additional `k-m` digits from the tail of the sequence.
- case 2). once we remove all the `k` digits from the sequence, there could be some leading zeros left. To format the final number, we need to strip off those leading zeros.
- case 3). we might end up removing all numbers from the sequence. In this case, we should return zero, instead of empty string.

Here are some sample implementations.

Java | Python | Copy

```
1  class Solution:
2      def removeKdigits(self, num: str, k: int) -> str:
3          numStack = []
4
5          # Construct a monotone increasing sequence of digits
6          for digit in num:
7              while k and numStack and numStack[-1] > digit:
8                  numStack.pop()
9                  k -= 1
10
11             numStack.append(digit)
12
13         # - Trunk the remaining K digits at the end
14         # - in the case k==0: return the entire list
15         finalStack = numStack[:-k] if k else numStack
16
17         # trip the leading zeros
18         return "".join(finalStack).lstrip('0') or "0"
```

Complexity Analysis

- Time complexity : $\mathcal{O}(N)$. Although there are nested loops, the inner loop is bounded to be run at most k times globally. Together with the outer loop, we have the exact $(N+k)$ number of operations. Since $0 < k \leq N$, the time complexity of the main loop is bounded within $2N$. For the logic outside the main loop, it is clear to see that their time complexity is $\mathcal{O}(N)$. As a result, the overall time complexity of the algorithm is $\mathcal{O}(N)$.
- Space complexity : $\mathcal{O}(N)$. We have a stack which would hold all the input digits in the worst case.

Rate this article:

Previous Next

Comments: 14 Sort By

Type comment here... (Markdown is supported)

Preview Post

zzznotsomuch 56 May 14, 2020 6:23 AM
Nice explanation. Who is with me when I say I almost got the greedy part but did not get the stack part and had to see the solution?
16 Share Reply
SHOW 1 REPLY

master_oogway13 13 September 26, 2019 6:53 AM
Nicely explained, Thanks! I am not sure if I read this but after k removal, we simply add all the elements to the stack. That's because since the top k conflicting digits are already gone and any further removal will not affect min state.
7 Share Reply

kevin2702 141 May 14, 2020 10:48 AM
Great solution. I implemented this problem using DP and always get Time Limit Exceeded.
3 Share Reply

ntkw 58 May 14, 2020 3:20 PM
Very good job. I thought that this would be a two pointer solution. but I failed miserably. Well done with the explanation.

Greedy approaches are usually tricky to get :)
2 Share Reply

alexander1089 12 May 14, 2020 11:03 AM
How do I come up with a solution to a problem like this in an interview ?
2 Share Reply
SHOW 2 REPLIES

t4nm0y 1 June 24, 2020 3:34 PM
this is the most beautiful explanation I have read today. thanks for the article.
1 Share Reply

codersingh99 2 June 8, 2020 12:10 AM
can anyone explain this :
# - Trunk the remaining K digits at the end
# - in the case k==0: return the entire list
finalStack = numStack[:-k] if k else numStack
0 Share Reply
SHOW 1 REPLY

pranav95 0 May 17, 2020 1:09 AM
Came up with my own approach and it was accepted but can someone tell me what is the run time for this algo? I think it is linear in best case and polynomial in worst case.
public class removeKDigits {
int start=0;
int end=0;
Read More
0 Share Reply

hankijooit 14 May 13, 2020 8:08 PM
Great article, for this line, is this in case where no elements were popped? Say "1111", 2? where k would be still 2 at line 14?

```
/* remove the remaining digits from the tail. */
```

Read More
0 Share Reply
SHOW 2 REPLIES

mercurywin 0 May 13, 2020 7:00 PM
It's amazing. I can think of the first greedy part but stack is really make the difference here instead of looping over the 1-digit way.
0 Share Reply