<https://leetcode.com/problems/remove-k-digits/description/>

Given string num representing a non-negative integer num, and an integer k, return *the smallest possible integer after removing* k *digits from* num.

**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 the smallest.

**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 contain leading zeroes.

**Example 3:**

Input: num = "10", k = 2

Output: "0"

Explanation: Remove all the digits from the number and it is left with nothing which is 0.

**Constraints:**

* 1 <= k <= num.length <= 105
* num consists of only digits.
* num does not have any leading zeros except for the zero itself.

**Attempt 1: 2023-03-24**

**Solution 1: Stack (30 min)**

class Solution {

public String removeKdigits(String num, int k) {

Stack<Character> stack = new Stack<Character>();

int len = num.length();

int i = 0;

while(i < len) {

while(k > 0 && !stack.isEmpty() && stack.peek() > num.charAt(i)) {

stack.pop();

k--;

}

stack.push(num.charAt(i));

i++;

}

// Why it requires additional handling of k > 0?

// Input num="1111", k=3

// Output "1111"

// Expected "1"

while(k > 0) {

stack.pop();

k--;

}

StringBuilder sb = new StringBuilder();

while(!stack.isEmpty()) {

sb.append(stack.pop());

}

sb.reverse();

// Why it requires remove heading 0s?

// Input num="10200", k=1

// Output "0200"

// Expected "200"

while(sb.length() > 1 && sb.charAt(0) == '0') {

sb.deleteCharAt(0);

}

// Why it requires make up empty string as 0?

// Input num="10", k=2

// Output ""

// Expected "0"

return sb.toString() == "" ? "0" : sb.toString();

}

}

**Refer to**

<https://leetcode.com/problems/remove-k-digits/discuss/88708/Straightforward-Java-Solution-Using-Stack>

class Solution {

public String removeKdigits(String num, int k) {

if(num == null || num.length() == 0) {

return "";

}

if(num.length() == k) {

return "0";

}

// Better than Stack<Integer> since no need calculate

// num.charAt(i) - '0'

Stack<Character> stack = new Stack<Character>();

// The given num does not contain any leading zero

// so we can directly put the 1st char on stack

stack.push(num.charAt(0));

int count = 0;

for(int i = 1; i < num.length(); i++) {

char curr = num.charAt(i);

// ??? This way not work ??? Instead remove the initial 0 after all

// Because second condition as '(stack.size() == 1 && stack.peek() == '0')'

// should not count into remove operation since 0200 need auto remove

// leading 0

// while((!stack.isEmpty() && curr < stack.peek() && count < k)

// || (stack.size() == 1 && stack.peek() == '0')) {

// stack.pop();

// count++;

// }

// Whenever meet a digit which is less than the previous digit,

// discard the previous one

while(!stack.isEmpty() && curr < stack.peek() && count < k) {

stack.pop();

count++;

}

stack.push(curr);

}

// Handle corner case as num = 112 and k = 1

while(count < k) {

stack.pop();

count++;

}

StringBuilder sb = new StringBuilder();

while(!stack.isEmpty()) {

sb.insert(0, stack.pop());

}

// Remove all the 0 at the head

while(sb.length() > 1 && sb.charAt(0) == '0') {

sb.deleteCharAt(0);

}

return sb.toString();

}

}

Just one suggestion though.

Instead of performing a reverse first and then removing the characters at head, you could remove the characters at the tail and then reverse the string.

deleteCharAt is a linear time operation. If you remove a character at the head, the characters will have to be moved forward by one position each time. On the other hand, removing the tail character can be achieved in constant time as there would be no shifting involved! :)

The change can be incorporated as follows:

while(sb.length() > 1 && sb.charAt(sb.length()-1) == '0')

sb.deleteCharAt(sb.length()-1);

return sb.reverse().toString();

**Why we can directly compare character with character (stack.peek()>num.charAt(i)), no need to transfer to integer ?**

**Because character '0' mapping to decimal value 48, '1' mapping to 49, '2' mapping to 50 .... '9' mapping to 57, monotonically increasing sequence**

**Refer to**

<https://www.cs.cmu.edu/~pattis/15-1XX/common/handouts/ascii.html>

ASCII Table

Dec = Decimal Value

Char = Character

'5' has the int value 53

if we write '5'-'0' it evaluates to 53-48, or the int 5

if we write char c = 'B'+32; then c stores 'b'

Dec Char Dec Char Dec Char Dec Char

--------- --------- --------- ----------

0 NUL (null) 32 SPACE 64 @ 96 `

1 SOH (start of heading) 33 ! 65 A 97 a

2 STX (start of text) 34 " 66 B 98 b

3 ETX (end of text) 35 # 67 C 99 c

4 EOT (end of transmission) 36 $ 68 D 100 d

5 ENQ (enquiry) 37 % 69 E 101 e

6 ACK (acknowledge) 38 & 70 F 102 f

7 BEL (bell) 39 ' 71 G 103 g

8 BS (backspace) 40 ( 72 H 104 h

9 TAB (horizontal tab) 41 ) 73 I 105 i

10 LF (NL line feed, new line) 42 \* 74 J 106 j

11 VT (vertical tab) 43 + 75 K 107 k

12 FF (NP form feed, new page) 44 , 76 L 108 l

13 CR (carriage return) 45 - 77 M 109 m

14 SO (shift out) 46 . 78 N 110 n

15 SI (shift in) 47 / 79 O 111 o

16 DLE (data link escape) 48 0 80 P 112 p

17 DC1 (device control 1) 49 1 81 Q 113 q

18 DC2 (device control 2) 50 2 82 R 114 r

19 DC3 (device control 3) 51 3 83 S 115 s

20 DC4 (device control 4) 52 4 84 T 116 t

21 NAK (negative acknowledge) 53 5 85 U 117 u

22 SYN (synchronous idle) 54 6 86 V 118 v

23 ETB (end of trans. block) 55 7 87 W 119 w

24 CAN (cancel) 56 8 88 X 120 x

25 EM (end of medium) 57 9 89 Y 121 y

26 SUB (substitute) 58 : 90 Z 122 z

27 ESC (escape) 59 ; 91 [ 123 {

28 FS (file separator) 60 < 92 \ 124 |

29 GS (group separator) 61 = 93 ] 125 }

30 RS (record separator) 62 > 94 ^ 126 ~

31 US (unit separator) 63 ? 95 \_ 127 DEL