Build Lowest Number by Removing n digits from a given number

Difficulty Level : Hard • Last Updated : 27 Jun, 2022

Given a string 'str' of digits and an integer 'n', build the lowest possible number by removing 'n' digits from the string and not changing the order of input digits.

Examples:

```
Input: str = "4325043", n = 3
Output: "2043"

Input: str = "765028321", n = 5
Output: "0221"

Input: str = "121198", n = 2
Output: "1118"
```

Recommended: Please solve it on "***PRACTICE***" first, before moving on to the solution.

of first (n+1) digits and put it in result, and recur for the remaining characters. Below is complete algorithm.

```
Initialize result as empty string
     res = ""
buildLowestNumber(str, n, res)
1) If n == 0, then there is nothing to remove.
   Append the whole 'str' to 'res' and return

2) Let 'len' be length of 'str'. If 'len' is smaller or equal
   to n, then everything can be removed
   Append nothing to 'res' and return

3) Find the smallest character among first (n+1) characters
   of 'str'.  Let the index of smallest character be minIndex.
   Append 'str[minIndex]' to 'res' and recur for substring after
   minIndex and for n = n-minIndex

     buildLowestNumber(str[minIndex+1..len-1], n-minIndex).
```

Below is the implementation of the above algorithm:

C++

```
// C++ program for the above approach

#include <bits/stdc++.h>
using namespace std;

string removeKdigits(string num, int k)
{
    int n = num.size();
    stack<char> mystack;
    // Store the final string in stack
```

```
            k -= 1;
        }

        if (!mystack.empty() || c != '0'){
            mystack.push(c);
        }
    }

    // Now remove the largest values from the top of the
    // stack
    while (!mystack.empty() && k--)
        mystack.pop();
    if (mystack.empty())
        return "0";

    // Now retrieve the number from stack into a string
    // (reusing num)
    while (!mystack.empty()) {
        num[n - 1] = mystack.top();
        mystack.pop();
        n -= 1;
    }
    return num.substr(n);
}

int main()
{
    string str = "765028321";
    int k = 5;
    cout << removeKdigits(str, k);
    return 0;
}
```

Java

```
// Java program for the above approach
import java.io.*;
import java.util.*;

class GFG {
```

```
        // We have to delete all digits
        if (k >= num.length()) {
            return "0";
        }
        // Nothing to delete
        if (k == 0) {
            return num;
        }
        Stack<Character> s = new Stack<Character>();

        for (int i = 0; i < num.length(); i++) {
            char c = num.charAt(i);

            // Removing all digits in stack that are greater
            // than this digit(since they have higher
            // weightage)
            while (!s.isEmpty() && k > 0 && s.peek() > c) {
                s.pop();
                k--;
            }
            // ignore pushing 0
            if (!s.isEmpty() || c != '0')
                s.push(c);
        }

        // If our k isnt 0 yet then we keep popping out the
        // stack until k becomes 0
        while (!s.isEmpty() && k > 0) {
            k--;
            s.pop();
        }
        if (s.isEmpty())
            return "0";
        while (!s.isEmpty()) {
            result.append(s.pop());
        }
        String str = result.reverse().toString();

        return str;
    }
```

```
        String s = "765028321";
        int k = 5;
        System.out.println(removeKdigits(s, 5));
    }
}
// this code is contributed by gireeshgudaparthi
```

Python3

```
# Python program for the above approach
def removeKdigits(num, k):

    n = len(num)
    mystack = []

    # Store the final string in stack
    for c in num:
        while (len(mystack) > 0 and k > 0 and ord(mystack[len(mystack)-1
            mystack.pop()
            k -= 1

        if len(mystack) > 0 or c != '0' :
            mystack.append(c)

    # Now remove the largest values from the top of the
    # stack
    while len(mystack) > 0 and k:
        mystack.pop()
        k -= 1
    if len(mystack) == 0:
        return "0"

    # Now retrieve the number from stack into a string
    # (reusing num)
    while(len(mystack) > 0):
        num = num.replace(num[n - 1],mystack[len(mystack) - 1])
        mystack.pop()
        n -= 1
    return num[n:]
```

```
# This code is contributed by shinjanpatra
```

C#

```
// C# program for the above approach
using System;
using System.Collections.Generic;
using System.Collections;
class HelloWorld {

    static string removeKdigits(string Num, int k)
    {
        char[] num = Num.ToCharArray();
        int n = num.Length;
        Stack<char> mystack = new Stack<char>();

        // Store the final string in stack
        for (int i = 0; i < num.Length; i++) {
            while (mystack.Count > 0 && k > 0
                   && mystack.Peek() > num[i]) {
                mystack.Pop();
                k = k - 1;
            }

            if (mystack.Count > 0 || num[i] != '0') {
                mystack.Push(num[i]);
            }
        }

        // Now remove the largest values from the top of the
        // stack
        while (mystack.Count > 0 && k > 0) {
            mystack.Pop();
            k = k - 1;
        }

        if (mystack.Count == 0)
            return "0";
```

```
            num[n - 1] = temp;
            mystack.Pop();
            n = n - 1;
        }

        return new string(num).Substring(n);
    }

    static void Main()
    {
        string str = "765028321";
        int k = 5;
        Console.WriteLine(removeKdigits(str, k));
    }
}

// The code is contributed by Nidhi goel
```

Javascript

```
<script>

// JavaScript program for the above approach
function removeKdigits(num,k)
{
    let n = num.length;
    let mystack = [];

    // Store the final string in stack
    for (let c of num)
    {
        while (mystack.length>0 && k > 0 &&
        mystack[mystack.length-1].charCodeAt(0) > c.charCodeAt(0))
        {
            mystack.pop();
            k -= 1;
        }

        if(mystack.length > 0 || c !== '0')
```

```
    // Now remove the largest values from the top of the
    // stack
    while(mystack.length > 0 && k--)
        mystack.pop();
    if (mystack.length == 0)
        return "0";

    // Now retrieve the number from stack into a string
    // (reusing num)
    while(mystack.length > 0)
    {
        num = num.split('');
        num[n - 1] = mystack[mystack.length - 1];
        num = num.join('');
        mystack.pop();
        n -= 1;
    }
    return num.substr(n);
}

// Driver code
let str = "765028321"
let k = 5
document.write(removeKdigits(str, k))

// This code is contributed by shinjanpatra

</script>
```

Output

```
 221
```

Below is an optimized code in C++ contributed by Gaurav Mamgain

---

C++14

```
void insertInNonDecOrder(deque<char>& dq, char ch)
{

    // If container is empty , insert the current digit
    if (dq.empty())
        dq.push_back(ch);

    else {
        char temp = dq.back();

        // Keep removing digits larger than current digit
        // from the back side of deque
        while (temp > ch && !dq.empty()) {
            dq.pop_back();
            if (!dq.empty())
                temp = dq.back();
        }

        // Insert the current digit
        dq.push_back(ch);
    }
    return;
}

string buildLowestNumber(string str, int n)
{
    int len = str.length();

    // Deleting n digits means we need to print k digits
    int k = len - n;

    deque<char> dq;
    string res = "";

    // Leaving rightmost k-1 digits we need to choose
    // minimum digit from rest of the string and print it
    int i;
    for (i = 0; i <= len - k; i++)

        // Insert new digit from the back side in
        // appropriate position and/ keep removing
```

```
    while (i < len) {

        // keep the minimum digit in output string
        res += dq.front();

        // remove minimum digit
        dq.pop_front();

        // Again insert new digit from the back
        // side in appropriate position and keep
        // removing digits larger than current digit
        insertInNonDecOrder(dq, str[i]);
        i++;
    }

    // Now only one element will be there in the deque
    res += dq.front();
    dq.pop_front();
    return res;
}

string lowestNumber(string str, int n)
{
    string res = buildLowestNumber(str, n);

    // Remove all the leading zeroes
    string ans = "";
    int flag = 0;
    for (int i = 0; i < res.length(); i++) {
        if (res[i] != '0' || flag == 1) {
            flag = 1;
            ans += res[i];
        }
    }

    if (ans.length() == 0)
        return "0";
    else
        return ans;
}
```

```
    int n = 5;
    cout <<lowestNumber(str, n) << endl;
    return 0;
}
// This code is contributed by Gaurav Mamgain
```

Output

```
221
```

Time Complexity: *O(N)*

Space Complexity: *O(N)*

Approach-2:

Let's suppose the length of the given string num be n.so the result string will contain the length of n-k.

As we proceed to solve this problem we should make sure that the output string contains minimum values at their high weightage positions. so we ensure that by using a stack.

1. *Return 0 if k >=n. and return num if k=0.*
2. *Create a stack and iterate through num string and push the value at that position if it is greater than the top element of the stack.*
3. *Iterate through the num string and if the integer value at that position is less than the top of the stack we will pop the stack and decrement k until we reach the condition where the top of the stack is less than the value we are looking at(while k>0) (by this we are making sure that most significant positions of the result are filled with minimum values).*
4. *If the k is still greater than 0 we will pop stack until k becomes 0.*
5. *Append the elements in the stack to the result string.*
6. *Delete leading zeroes from the result string.*

Below is the implementation of the above approach:

C++

```
// C++ program for the above approach
```

```
string removeKdigits(string num, int k)
{
    int n = num.size();
    stack<char> mystack;
    // Store the final string in stack
    for (char c : num) {
        while (!mystack.empty() && k > 0
               && mystack.top() > c) {
            mystack.pop();
            k -= 1;
        }

        if (!mystack.empty() || c != '0')
            mystack.push(c);
    }

    // Now remove the largest values from the top of the
    // stack
    while (!mystack.empty() && k--)
        mystack.pop();
    if (mystack.empty())
        return "0";

    // Now retrieve the number from stack into a string
    // (reusing num)
    while (!mystack.empty()) {
        num[n - 1] = mystack.top();
        mystack.pop();
        n -= 1;
    }
    return num.substr(n);
}

int main()
{
    string str = "765028321";
    int k = 5;
    cout << removeKdigits(str, k);
    return 0;
}
```

```java
import java.io.*;
import java.util.*;

class GFG {

    public static String removeKdigits(String num, int k)
    {
        StringBuilder result = new StringBuilder();

        // We have to delete all digits
        if (k >= num.length()) {
            return "0";
        }
        // Nothing to delete
        if (k == 0) {
            return num;
        }
        Stack<Character> s = new Stack<Character>();

        for (int i = 0; i < num.length(); i++) {
            char c = num.charAt(i);

            // Removing all digits in stack that are greater
            // than this digit(since they have higher
            // weightage)
            while (!s.isEmpty() && k > 0 && s.peek() > c) {
                s.pop();
                k--;
            }
            // ignore pushing 0
            if (!s.isEmpty() || c != '0')
                s.push(c);
        }

        // If our k isnt 0 yet then we keep popping out the
        // stack until k becomes 0
        while (!s.isEmpty() && k > 0) {
            k--;
            s.pop();
        }
        if (s.isEmpty())
```

```
        String str = result.reverse().toString();

        return str;
    }

    // Driver Code
    public static void main(String[] args)
    {
        String s = "765028321";
        int k = 5;
        System.out.println(removeKdigits(s, 5));
    }
}
// this code is contributed by gireeshgudaparthi
```

Python3

```
# Python program for the above approach
def removeKdigits(num, k):

    n = len(num)
    mystack = []

    # Store the final string in stack
    for c in num:
        while (len(mystack) > 0 and k > 0
               and mystack[-1] > c):
            mystack.pop()
            k -= 1

        if (len(mystack) > 0 or c != '0'):
            mystack.append(c)

    # Now remove the largest values from the top of the
    # stack
    while (len(mystack) > 0 and k):
        k -= 1
        mystack.pop()
    if (len(mystack) == 0):
        return "0"
```

```
            num = num.replace(num[n - 1] , mystack[-1])
            mystack.pop()
            n -= 1

        return num[n:]

# driver code
Str = "765028321"
k = 5
print(removeKdigits(Str, k))

# This code is contributed by shinjanpatra
```

Javascript

```
<script>

// JavaScript program for the above approach


function removeKdigits(num,k){

    let n = num.length
    let mystack = []
    // Store the final string in stack
    for(let c of num){
        while (mystack.length>0 && k > 0 && mystack[mystack.length - 1] >
            mystack.pop()
            k -= 1
        }

        if (mystack.length>0 || c != '0')
            mystack.push(c)
    }

    // Now remove the largest values from the top of the
    // stack
    while (mystack.length>0 && k){
        k -= 1
```

```
    // Now retrieve the number from stack into a string
    // (reusing num)
    while (mystack.length>0){
        num = num.replace(num[n - 1] , mystack[mystack.length-1])
        mystack.pop()
        n -= 1
    }

    return num.substring(n,)
}

// driver code

let Str = "765028321"
let k = 5
document.write(removeKdigits(Str, k))

// code is contributed by shinjanpatra

</script>
```

Output

```
221
```

Time complexity: *O(N)*

Space complexity: *O(N)*

This article is contributed by **Pallav Gurha**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Next

Calculate the angle between hour hand and minute hand

RECOMMENDED ARTICLES

Page : 1 2 3

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Hard

Easy | Normal | Medium | Hard | Expert

Improved By : sanjeev2552, Rajnis09, gireeshgudaparthi, UtkarshPandey6, vaishaligoyal878, sooda367, arpitsrivastava467, prasanna1995, abhishek0719kadiyan, shinjanpatra, classroompxico, hardikkoriintern

Article Tags : Amazon, FactSet, number-digits, Mathematical, Strings

Practice Tags : Amazon, FactSet, Strings, Mathematical

Improve Article | Report Issue

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

GeeksforGeeks
A-143, 9th Floor, Sovereign Corporate Tower,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

NodeJS