Form minimum number from given sequence

Given a pattern containing only I’s and D’s. **I** for increasing and **D** for decreasing. Devise an algorithm to print the minimum number following that pattern. Digits from 1-9 and digits can’t repeat.

Examples:

```
Input: D        Output: 21
Input: I        Output: 12
Input: DD       Output: 321
Input: II       Output: 123
Input: DIDI     Output: 21435
Input: IIDDD    Output: 126543
Input: DDIDDIID Output: 321654798
```

Source: Amazon Interview Question

We strongly recommend that you click here and practice it, before moving on to the solution.

Below are some important observations

Since digits can’t repeat, there can be at most 9 digits in output.

Also number of digits in output is one more than number of characters in input. Note that the first character of input corresponds to two digits in output.

Idea is to iterate over input array and keep track of last printed digit and maximum digit printed so far. Below is C++ implementation.

```
// C++ program to print minimum number that can be formed
// from a given sequence of Is and Ds
#include <iostream>
using namespace std;

// Prints the minimum number that can be formed from
// input sequence of I's and D's
void PrintMinNumberForPattern(string arr)
{
    // Initialize current_max (to make sure that
    // we don't use repeated character
    int curr_max = 0;

    // Initialize last_entry (Keeps track for
    // last printed digit)
    int last_entry = 0;

    int j;

    // Iterate over input array
    for (int i=0; i<arr.length(); i++)
    {
        // Initialize 'noOfNextD' to get count of
        // next D's available
        int noOfNextD = 0;

        switch(arr[i])
        {
        case 'I':
            // If letter is 'I'

            // Calculate number of next consecutive D's
            // available
            j = i+1;
            while (arr[j] == 'D' && j < arr.length())
            {
                noOfNextD++;
                j++;
            }

            if (i==0)
            {
                curr_max = noOfNextD + 2;

                // If 'I' is first letter, print incremented
                // sequence from 1
                cout << " " << ++last_entry;
                cout << " " << curr_max;

                // Set max digit reached
                last_entry = curr_max;
            }
            else
            {
                // If not first letter

                // Get next digit to print
                curr_max = curr_max + noOfNextD + 1;

                // Print digit for I
                last_entry = curr_max;
                cout << " " << last_entry;
            }

            // For all next consecutive 'D' print
            // decremented sequence
            for (int k=0; k<noOfNextD; k++)
            {
                cout << " " << --last_entry;
                i++;
            }
            break;

        // If letter is 'D'
        case 'D':
            if (i == 0)
            {
                // If 'D' is first letter in sequence
                // Find number of Next D's available
                j = i+1;
                while (arr[j] == 'D' && j < arr.length())
                {
                    noOfNextD++;
                    j++;
                }

                // Calculate first digit to print based on
                // number of consecutive D's
                curr_max = noOfNextD + 2;

                // Print twice for the first time
                cout << " " << curr_max << " " << curr_max - 1;

                // Store last entry
                last_entry = curr_max - 1;
            }
            else
            {
                // If current 'D' is not first letter

                // Decrement last_entry
                cout << " " << last_entry - 1;
                last_entry--;
            }
            break;
        }
    }
    cout << endl;
}

// Driver program to test above
int main()
{
    PrintMinNumberForPattern("IDID");
    PrintMinNumberForPattern("I");
    PrintMinNumberForPattern("DD");
    PrintMinNumberForPattern("II");
    PrintMinNumberForPattern("DIDI");
    PrintMinNumberForPattern("IIDDD");
    PrintMinNumberForPattern("DDIDDIID");
    return 0;
}
```

Output:

```
 1 3 2 5 4
 1 2
 3 2 1
 1 2 3
 2 1 4 3 5
 1 2 6 5 4 3
 3 2 1 6 5 4 7 9 8
```

This solution is suggested by Swapnil Trambake.

Alternate Solution:
Let's observe a few facts in case of minimum number:

- The digits can't repeat hence there can be 9 digits at most in output.
- To form a minimum number , at every index of the output, we are interested in the minimum number which can be placed at that index.

The idea is to iterate over the entire input array , keeping track of the minimum number (1-9) which can be placed at that position of the output.

The tricky part of course occurs when 'D' is encountered at index other than 0. In such a case we have to track the nearest 'I' to the left of 'D' and increment each number in the output vector by 1 in between 'I' and 'D'.

We cover the base case as follows:

- If the first character of input is 'I' then we append 1 and 2 in the output vector and the minimum available number is set to 3 .The index of most recent 'I' is set to 1.
- If the first character of input is 'D' then we append 2 and 1 in the output vector and the minimum available number is set to 3, and the index of most recent 'I' is set to 0.

Now we iterate the input string from index 1 till its end and:

- If the character scanned is 'I' ,minimum value which has not been used yet is appended to the output vector .We increment the value of minimum no. available and index of most recent 'I' is also updated.
- If the character scanned is 'D' at index i of input array, we append the ith element from output vector in the output and track the nearest 'I' to the left of 'D' and increment each number in the output vector by 1 in between 'I' and 'D'.

Following is the C++ program for the same:

```cpp
// C++ program to print minimum number that can be formed
// from a given sequence of Is and Ds
#include<bits/stdc++.h>
using namespace std;

void printLeast(string arr)
{
    // min_avail represents the minimum number which is
    // still available for inserting in the output vector.
    // pos_of_I keeps track of the most recent index
    // where 'I' was encountered w.r.t the output vector
    int min_avail = 1, pos_of_I = 0;

    //vector to store the output
    vector<int>v;

    // cover the base cases
    if (arr[0]=='I')
    {
        v.push_back(1);
        v.push_back(2);
        min_avail = 3;
        pos_of_I = 1;
    }
    else
    {
        v.push_back(2);
        v.push_back(1);
        min_avail = 3;
        pos_of_I = 0;
    }

    // Traverse rest of the input
    for (int i=1; i<arr.length(); i++)
    {
        if (arr[i]=='I')
        {
            v.push_back(min_avail);
            min_avail++;
            pos_of_I = i+1;
        }
        else
        {
            v.push_back(v[i]);
            for (int j=pos_of_I; j<=i; j++)
                v[j]++;

            min_avail++;
        }
    }

    // print the number
    for (int i=0; i<v.size(); i++)
        cout << v[i] << " ";
    cout << endl;
}

// Driver program to check the above function
int main()
{
    printLeast("IDID");
    printLeast("I");
    printLeast("DD");
    printLeast("II");
    printLeast("DIDI");
    printLeast("IIDDD");
    printLeast("DDIDDIID");
    return 0;
}
```

Output:

```
1 3 2 5 4
1 2
3 2 1
1 2 3
2 1 4 3 5
1 2 6 5 4 3
3 2 1 6 5 4 7 9 8
```