Google interview: Convert Arabic to Roman numerals

-Charles 10/18/2017

Method 1

Algorithm

My idea is to write the number from left to right, using the largest available token. I repeatedly remove the largest value represented by a token, and write that token in our answer.

However, for an integer like N = 9, the answer is 'IX', not 'VIIII' We can repair our initial idea by using not just tokens like 'M', 'D', 'C', 'L', 'X', 'V', 'I', but also tokens like 'CM', 'CD, 'XC', 'XL', 'IX', 'IV'.

Complexity Analysis

Time Complexity: In our specific case of N < 9999, the most longest Roman numerals is MDCCCLXXXVIII, we will loop through our tokens array up to 13 times, so it's complexity is O(1).

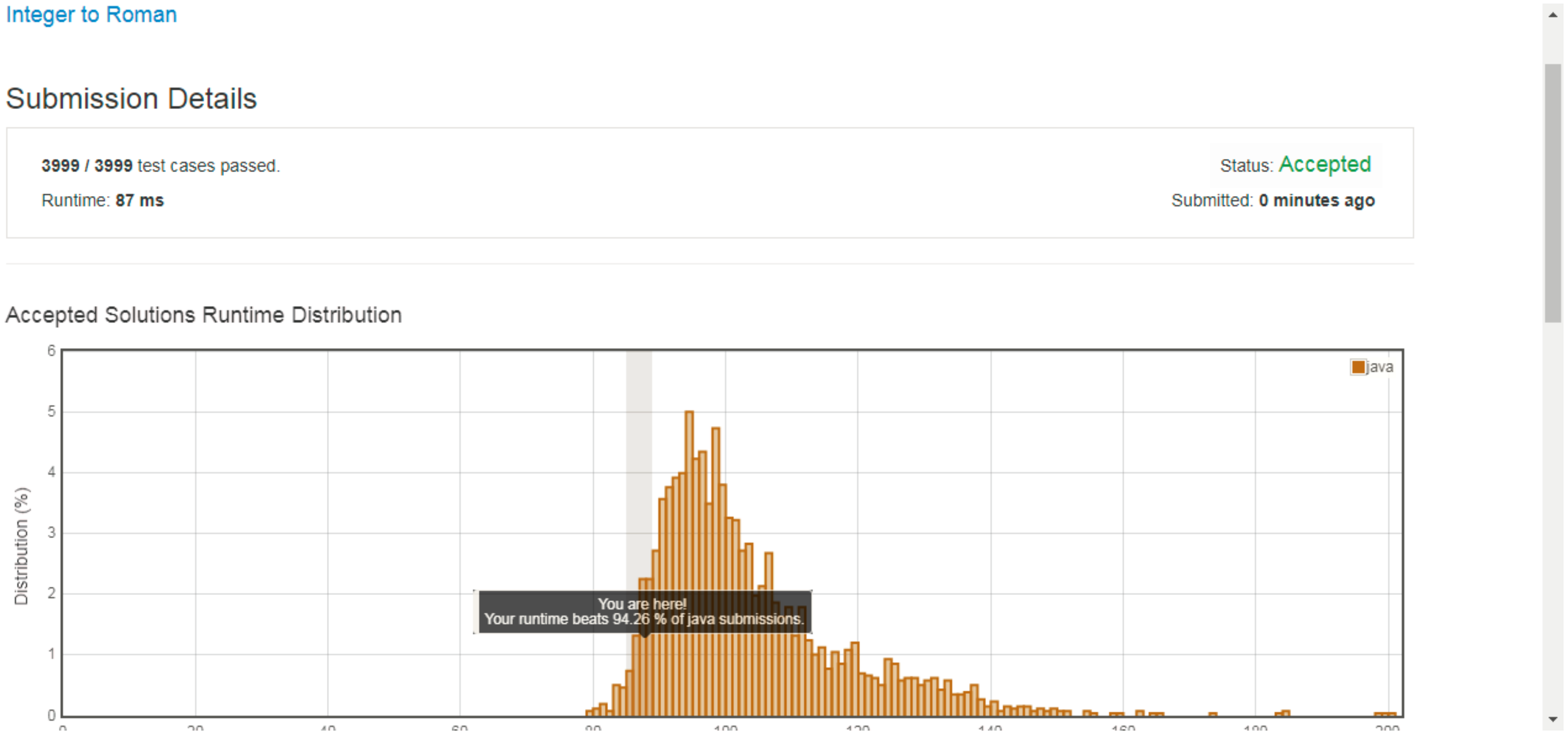
Space Complexity: As described above, the most longest Roman numerals is only 13 characters, it's complexity is O(1) too.

Code in Java:

```
package convertAtoR;

import java.util.*;

//import java.math.*;
//import java.security.*;
```

```java
//import java.util.Map.Entry;

public class AtoR1 {
    public String convert(int anum) {
        int[] a = { 9000, 5000, 4000, 1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1 };
        String[] c = { "MX̅", "V̅", "MV̅", "M", "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX", "V", "IV", "I" };
        String roman = "";
        int remainder = anum;

        if (anum > 9999 || anum < 1) {
            return "Expect number from 1 to 9,999";
        }
        for (int i = 0; i < 30; i++) {
            while (remainder >= a[i]) {
                roman = roman + c[i];
                remainder = remainder - a[i];
            }
            if (remainder <= 0)
                break;
        }
        return roman;
    }

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        AtoR1 a = new AtoR1();
        System.out.println("Enter a Arabic number between 1 to 9999");
        int Anum = in.nextInt();
        String Rnum = a.convert(Anum);

        System.out.println("The Arabic number : " + Anum);
        System.out.println("equals");
        System.out.println("The Roman numerals : " + Rnum);
    }
}
```

Result:



Integer to Roman

Submission Details

3999 / 3999 test cases passed.

Runtime: **87 ms**

Status: Accepted

Submitted: **0 minutes ago**

Accepted Solutions Runtime Distribution

You are here!
Your runtime beats 94.26 % of java submissions.

Method 1

Algorithm

The answers for the thousands, hundreds, tens, and ones place can be independently added together.

Complexity Analysis

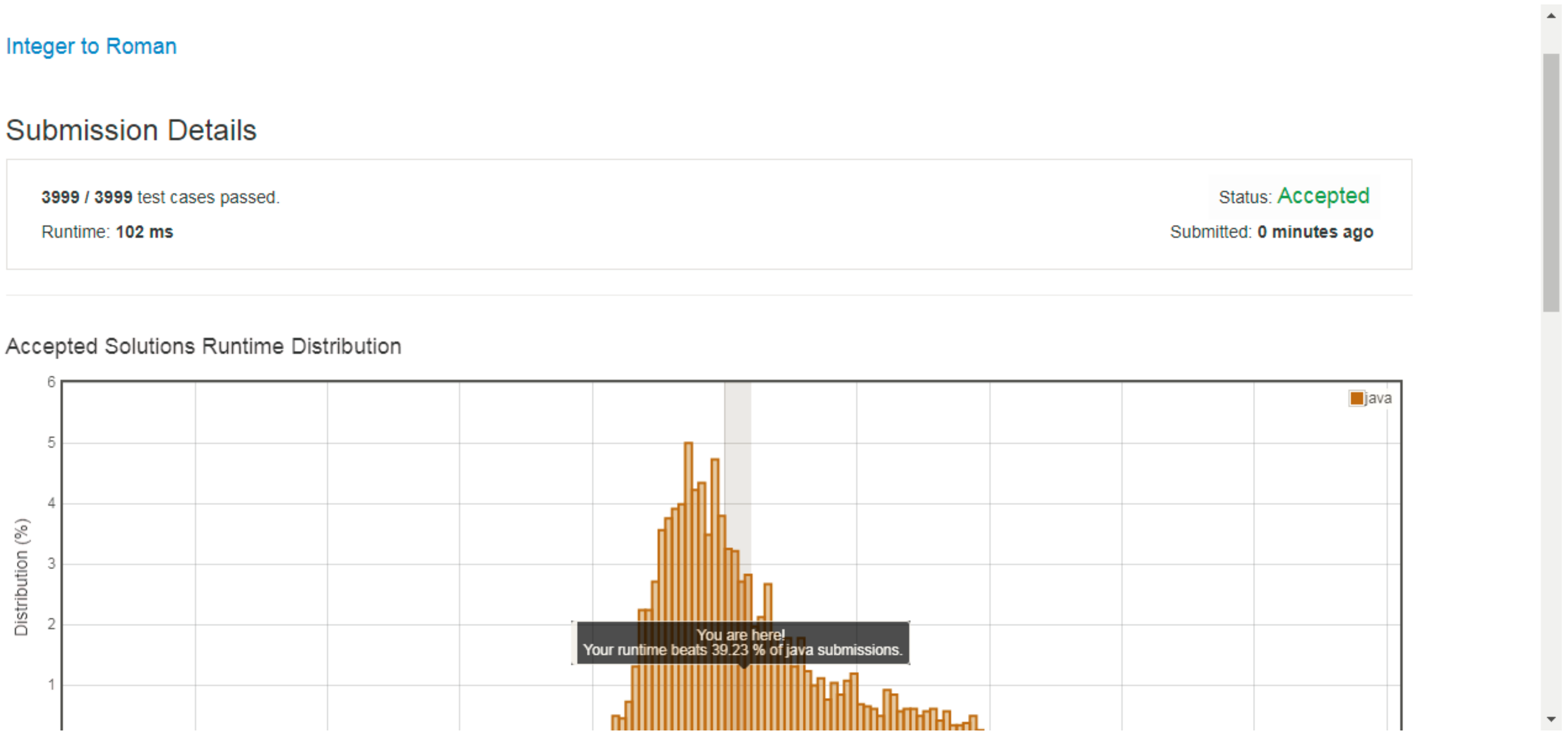
Time Complexity: We seek four strings and add them together in O(1) time.

Space Complexity: We use a constant amount of space O(1) to hold the correct partial answers directly in our code.

Code in Java:

```java
package convertAtoR;

import java.util.Scanner;

public class AtoR2 {

    public String convert(int a) {
        String[][] r = { { "", "I", "II", "III", "IV", "V", "VI", "VII", "VIII", "IX" },
                { "", "X", "XX", "XXX", "XL", "L", "LX", "LXX", "LXXX", "XC" },
                { "", "C", "CC", "CCC", "CD", "D", "DC", "DCC", "DCCC", "CM" },
                { "", "M", "MM", "MMM", "MV̅", "V̅", "V̅M", "V̅MM", "V̅MMM", "MX̅" } };
        return r[3][a / 1000 % 10] + r[2][a / 100 % 10] + r[1][a / 10 % 10] + r[0][a % 10];
    }

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        AtoR2 a = new AtoR2();
        System.out.println("Enter a Arabic number between 1 to 9999");
        int Anum = in.nextInt();
        String Rnum = a.convert(Anum);

        System.out.println("The Arabic number : " + Anum);
        System.out.println("equals");
        System.out.println("The Roman numerals : " + Rnum);
    }

}
```

Result:



Integer to Roman

Submission Details

3999 / 3999 test cases passed.

Runtime: **102 ms**

Status: Accepted

Submitted: **0 minutes ago**

Accepted Solutions Runtime Distribution