

题目描述

小明在学习二进制时，发现了一类不含 101 的数，也就是：
将数字用二进制表示，不能出现 101。
现在给定一个整数区间 $[l,r]$ ，请问这个区间包含了多少个不含 101 的数？

输入描述

输入的唯一一行包含两个正整数 l, r ($1 \leq l \leq r \leq 10^9$)。

输出描述

输出的唯一一行包含一个整数，表示在 $[l,r]$ 区间内一共有几个不含 101 的数。

用例

输入	1 10
输出	8
说明	区间 $[1,10]$ 内，5 的二进制表示为 101，10 的二进制表示为 1010，因此区间 $[1,10]$ 内有 $10-2=8$ 个不含 101 的数。
输入	10 20
输出	7
说明	区间 $[10,20]$ 内，满足条件的数字有 $[12,14,15,16,17,18,19]$ 因此答案为 7。

题目解析

本题如果用暴力法求解，很简单。

```
1  /* JavaScript Node ACM模式 控制台输入获取 */
2  const readline = require("readline");
3
4  const rl = readline.createInterface({
5    input: process.stdin,
6    output: process.stdout,
7  });
8
9  rl.on("line", (line) => {
10    const [l, r] = line.split(" ").map(Number);
11
12    console.log(getResult(l, r));
13  });
14
15  function getResult(l, r) {
16    let count = r - l + 1;
17    for (let i = l; i <= r; i++) {
18      if (Number(i).toString(2).indexOf("101") !== -1) {
19        count--;
20      }
21    }
22    return count;
23  }
```

但是本题的 $1 \leq l \leq r \leq 10^9$ ，也就是说区间范围最大是 $1 \sim 10^9$ ，那么上面 $O(n)$ 时间复杂度的算法会超时。

因此，我们需要找到一个性能更优的算法。

本题需要使用 [数位DP](#) 算法，具体逻辑原理请看

[数位DP - 带3的数_伏城之外的博客-CSDN博客](#)

[数位DP - 带49的数_伏城之外的博客-CSDN博客](#)

JavaScript算法源码

题目解析

本题如果用暴力法求解，很简单

```
1 /* JavaScript Node ACM模式 控制台输入获取 */
2 const readline = require("readline");
3
4 const rl = readline.createInterface({
5   input: process.stdin,
6   output: process.stdout,
7 });
8
9 rl.on("line", (line) => {
10   const [l, r] = line.split(" ").map(Number);
11
12   console.log(getResult(l, r));
13 });
14
15 function getResult(l, r) {
16   let count = r - l + 1;
17   for (let i = l; i <= r; i++) {
18     if (Number(i).toString(2).indexOf("101") !== -1) {
19       count--;
20     }
21   }
22   return count;
23 }
```

但是本题的 $1 \leq l \leq r \leq 10^9$ ，也就是说区间范围最大是 $1 \sim 10^9$ ，那么上面 $O(n)$ 时间复杂度的算法会超时。

因此，我们需要找到一个性能更优的算法。

本题需要使用 **数位DP** 算法，具体逻辑原理请看

[数位DP - 带3的数_伏城之外的博客-CSDN博客](#)


[数位DP - 带49的数_伏城之外的博客-CSDN博客](#)

JavaScript算法源码

```
1 /* JavaScript Node ACM模式 控制台输入获取 */
2 const readline = require("readline");
3
4 const rl = readline.createInterface({
5   input: process.stdin,
6   output: process.stdout,
7 });
8
9 rl.on("line", (line) => {
10   const [L, R] = line.split(" ").map(Number);
11   const count = digitSearch(R) - digitSearch(L - 1);
12
13   console.log(count);
14 });
15
16 function digitSearch(num) {
17   const arr = num.toString(2).split("").map(Number);
18   const f = new Array(arr.length)
19     .fill(0)
20     .map(() => new Array(2).fill(0).map(() => new Array(2)));
21
22   return dfs(0, true, f, arr, 0, 0);
23 }
24
25 function dfs(p, limit, f, arr, pre, prepre) {
26   if (p === arr.length) return 1;
27
28   if (!limit && f[p][pre][prepre]) return f[p][pre][prepre];
29
30   const max = limit ? arr[p] : 1;
31   let count = 0;
32
33   for (let i = 0; i <= max; i++) {
34     if (i === 1 && pre === 0 && prepre === 1) continue;
35     count += dfs(p + 1, limit && i === max, f, arr, i, pre);
36   }
37
38   if (!limit) f[p][pre][prepre] = count;
39
40   return count;
41 }
```

Java算法源码

```
1 import java.util.Arrays;
2 import java.util.Scanner;
3
4 public class Main {
5   public static void main(String[] args) {
6     Scanner sc = new Scanner(System.in);
7
8     int L = sc.nextInt();
```

 伏城之外 [已关注](#)

 1   4  3 [专栏目录](#) [已订阅](#)

Java算法源码

```
1 import java.util.Arrays;
2 import java.util.Scanner;
3
4 public class Main {
5     public static void main(String[] args) {
6         Scanner sc = new Scanner(System.in);
7
8         int L = sc.nextInt();
9         int R = sc.nextInt();
10        int count = digitSearch(R) - digitSearch(L - 1);
11        System.out.println(count);
12    }
13
14    public static int digitSearch(int num) {
15        Integer[] arr =
16            Arrays.stream(Integer.toBinaryString(num).split(""))
17                .map(Integer::parseInt)
18                .toArray(Integer[]::new);
19
20        int[][][] f = new int[arr.length][2][2];
21
22        return dfs(0, true, f, arr, 0, 0);
23    }
24
25    public static int dfs(int p, boolean limit, int[][][] f, Integer[] arr, int pre, int prepre) {
26        if (p == arr.length) return 1;
27
28        if (!limit && f[p][pre][prepre] != 0) return f[p][pre][prepre];
29
30        int max = limit ? arr[p] : 1;
31        int count = 0;
32
33        for (int i = 0; i <= max; i++) {
34            if (i == 1 && pre == 0 && prepre == 1) continue;
35            count += dfs(p + 1, limit && i == max, f, arr, i, pre);
36        }
37
38        if (!limit) f[p][pre][prepre] = count;
39
40        return count;
41    }
42 }
```

Python算法源码

```
1 # 算法实现
2 def dfs(p, limit, f, arr, pre, prepre):
3     if p == len(arr):
4         return 1
5
6     if not limit and f[p][pre][prepre] > 0:
7         return f[p][pre][prepre]
8
9     maxV = arr[p] if limit else 1
10    count = 0
11
12    for i in range(maxV + 1):
13        if i == 1 and pre == 0 and prepre == 1:
14            continue
15        count += dfs(p + 1, limit and i == maxV, f, arr, i, pre)
16
17    if not limit:
18        f[p][pre][prepre] = count
19
20    return count
21
22
23 def digitSearch(num):
24     arr = list(map(int, list(format(num, 'b'))))
25     f = [[[0 for k in range(2)] for j in range(2)] for i in range(len(arr))]
26     return dfs(0, True, f, arr, 0, 0)
27
28
29 # 输入获取
30 L, R = map(int, input().split())
31
32 # 算法调用
33 print(digitSearch(R) - digitSearch(L - 1))
```