华为OD机试 – 不含101的数（Java & JS & Python）

原创 伏城之外 已于 2023-02-15 13:50:16 修改 6586 收藏 23 版权
分类专栏： 华为OD机试（Java & JS & Python） # 华为OD机试2023A 文章标签： 华为机试 算法 JavaScript Java Python

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 ≤ l ≤ r ≤ 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

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

rl.on("line", (line) => {
  const [l, r] = line.split(" ").map(Number);

  console.log(getResult(l, r));
});

function getResult(l, r) {
  let count = r - l + 1;
  for (let i = l; i <= r; i++) {
    if (Number(i).toString(2).indexOf("101") !== -1) {
      count--;
    }
  }
  return count;
}
```

但是本题的1 ≤ l ≤ r ≤ 10^9，也就是说区间范围最大是 1 ~ 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

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

rl.on("line", (line) => {
  const [L, R] = line.split(" ").map(Number);
  const count = digitSearch(R) - digitSearch(L - 1);

  console.log(count);
});

function digitSearch(num) {
  const arr = num.toString(2).split("").map(Number);
  const f = new Array(arr.length)
    .fill(0)
    .map(() => new Array(2).fill(0).map(() => new Array(2)));

  return dfs(0, true, f, arr, 0, 0);
}

function dfs(p, limit, f, arr, pre, prepre) {
  if (p === arr.length) return 1;

  if (!limit && f[p][pre][prepre]) return f[p][pre][prepre];

  const max = limit ? arr[p] : 1;
  let count = 0;

  for (let i = 0; i <= max; i++) {
    if (i === 1 && pre === 0 && prepre === 1) continue;
    count += dfs(p + 1, limit && i === max, f, arr, i, pre);
  }

  if (!limit) f[p][pre][prepre] = count;

  return count;
}
```

Java算法源码

```java
import java.util.Arrays;
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int L = sc.nextInt();
    int R = sc.nextInt();
    int count = digitSearch(R) - digitSearch(L - 1);
    System.out.println(count);
  }

  public static int digitSearch(int num) {
    Integer[] arr =
        Arrays.stream(Integer.toBinaryString(num).split(""))
            .map(Integer::parseInt)
            .toArray(Integer[]::new);

    int[][][] f = new int[arr.length][2][2];

    return dfs(0, true, f, arr, 0, 0);
  }

  public static int dfs(int p, boolean limit, int[][][] f, Integer[] arr, int
    if (p == arr.length) return 1;

    if (!limit && f[p][pre][prepre] != 0) return f[p][pre][prepre];

    int max = limit ? arr[p] : 1;
    int count = 0;

    for (int i = 0; i <= max; i++) {
      if (i == 1 && pre == 0 && prepre == 1) continue;
      count += dfs(p + 1, limit && i == max, f, arr, i, pre);
    }

    if (!limit) f[p][pre][prepre] = count;

    return count;
  }
}
```

Python算法源码

```python
# 算法实现
def dfs(p, limit, f, arr, pre, prepre):
    if p == len(arr):
        return 1

    if not limit and f[p][pre][prepre] > 0:
        return f[p][pre][prepre]

    maxV = arr[p] if limit else 1
    count = 0

    for i in range(maxV + 1):
        if i == 1 and pre == 0 and prepre == 1:
            continue
        count += dfs(p + 1, limit and i == maxV, f, arr, i, pre)

    if not limit:
        f[p][pre][prepre] = count

    return count


def digitSearch(num):
    arr = list(map(int, list(format(num, 'b'))))
    f = [[[0 for k in range(2)] for j in range(2)] for i in range(len(arr))]
    return dfs(0, True, f, arr, 0, 0)


# 输入获取
L, R = map(int, input().split())

# 算法调用
print(digitSearch(R) - digitSearch(L - 1))
```