Previous Next

201. Bitwise AND of Numbers Range

Feb. 8, 2020 | 18.4K views

Average Rating: 4.71 (28 votes)

Given a range [m, n] where 0 <= m <= n <= 2147483647, return the bitwise AND of all numbers in this range, inclusive.

Example 1:

```
Input: [5,7]
Output: 4
```

Example 2:

```
Input: [0,1]
Output: 0
```

Solution

Overview

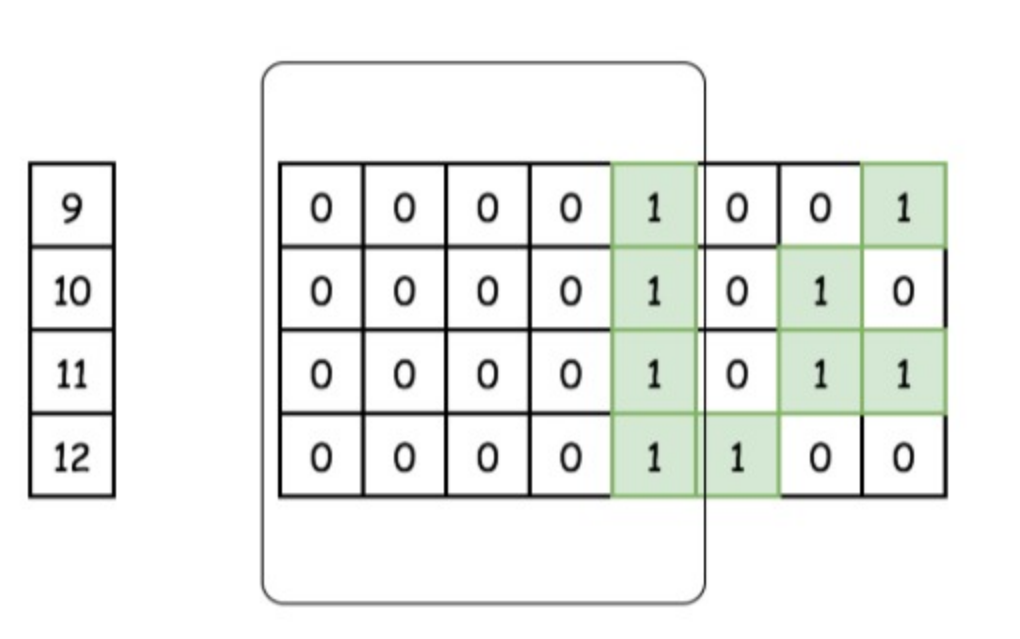
First of all, one of the most intuitive solutions that one might come up with might be to iterate all the numbers ***one by one*** in the range and do the the bit AND operation to obtain the result.

This could work for test cases with a small range. Unfortunately it would exceed the time limit set by the online judge for test cases with a relative large range. In this article, we would illustrate some other solutions that do not require the iteration of all numbers.

First of all, let us look into the characteristic of the AND operation.

> For a series of bits, *e.g.* `[1, 1, 0, 1, 1]`, as long as there is one bit of zero value, then the result of AND operation on this series of bits would be zero.

Back to our problem, first we could represent each number in the range in its binary form which we could view as a string of binary numbers (*e.g.* `9 = 00001001`). We then align the numbers according to the position of binary string.



In the above example, one might notice that after the AND operation on all the numbers, the remaining part of bit strings is the ***common prefix*** of all these bit strings.

The final result asked by the problem consists of this common prefix of bit string as its left part, with the rest of bits as zeros.

More specifically, the *common prefix* of all these bit string is also the common prefix between the ***starting*** and ***ending*** numbers of the specified range (*i.e.* 9 and 12 respectively in the above example).

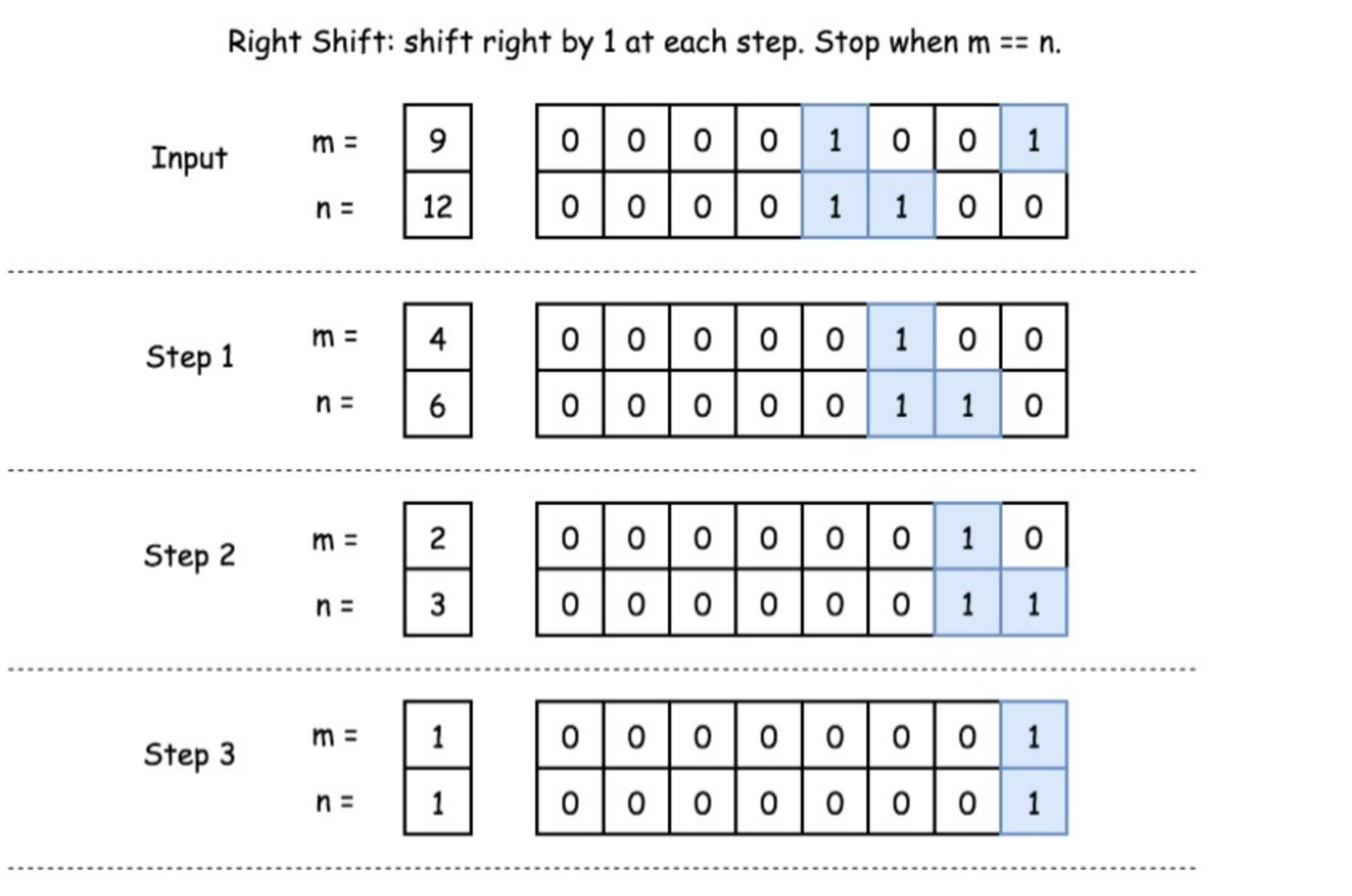
> As a result, we then can reformulate the problem as *"given two integer numbers, we are asked to find the* **common prefix** *of their binary strings."*

Approach 1: Bit Shift

Intuition

Given the above intuition about the problem, our task is to calculate the *common prefix* for the bit strings of the two given numbers. One of the solutions would be to resort to the **bit shift** operation.

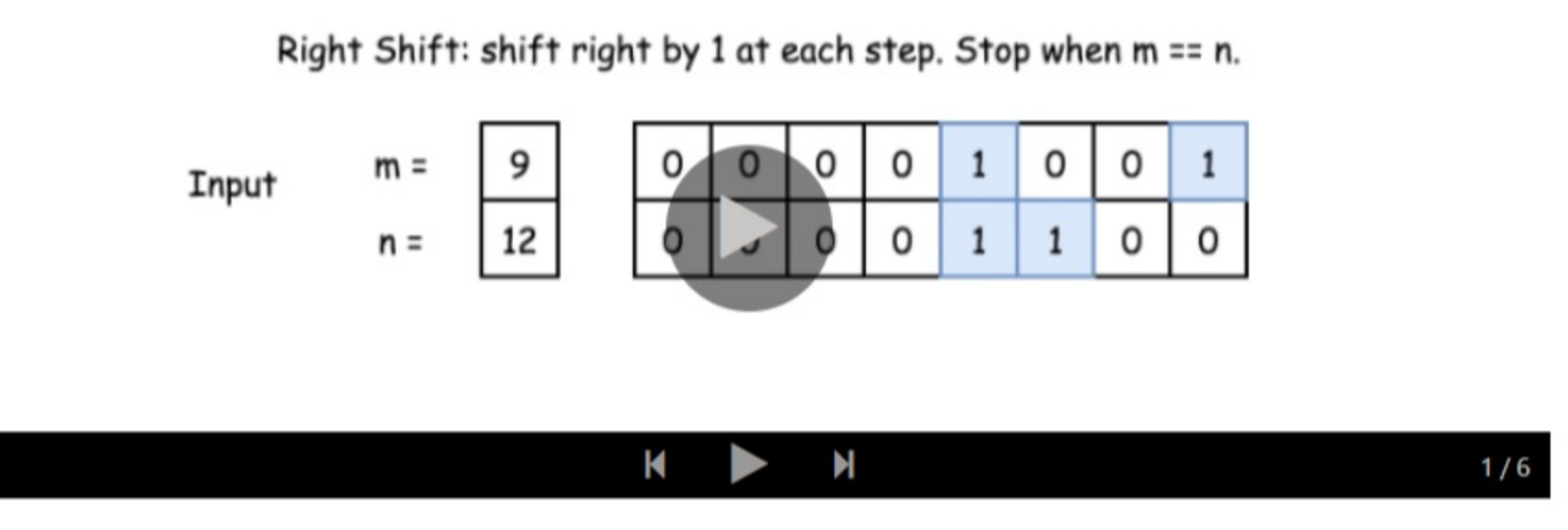
> The idea is that we shift both numbers to the right, until the numbers become equal, *i.e.* the numbers are reduced into their common prefix. Then we append zeros to the common prefix in order to obtain the desired result, by shifting the common prefix to the left.



Algorithm

As stated in the intuition section, the algorithm consists of two steps:

- We reduce both numbers into their common prefix, by doing right shift iteratively. During the iteration, we keep the count on the number of shift operations we perform.
- With the common prefix, we restore it to its previous position, by left shifting.



Java | Python | Copy

```java
class Solution {
  public int rangeBitwiseAnd(int m, int n) {
    int shift = 0;
    // find the common 1-bits
    while (m < n) {
      m >>= 1;
      n >>= 1;
      ++shift;
    }
    return m << shift;
  }
}
```

Complexity Analysis

- Time Complexity: $\mathcal{O}(1)$.
  - Although there is a loop in the algorithm, the number of iterations is bounded by the number of bits that an integer has, which is fixed.
  - Therefore, the time complexity of the algorithm is constant.
- Space Complexity: $\mathcal{O}(1)$. The consumption of the memory for our algorithm is constant, regardless the input.

Approach 2: Brian Kernighan's Algorithm

Intuition

Speaking of bit shifting, there is another related algorithm called Brian Kernighan's algorithm which is applied to turn off the rightmost bit of one in a number.

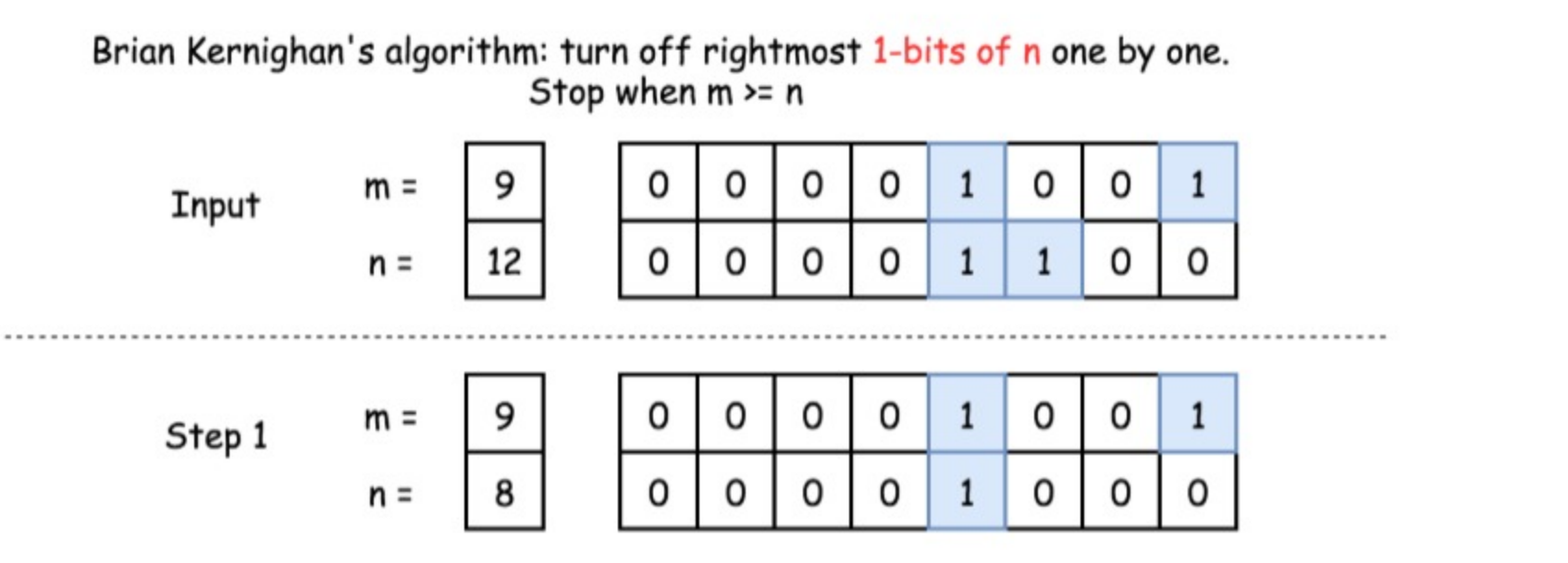
The secret sauce of the *Brian Kernighan's algorithm* can be summarized as follows:

> When we do AND bit operation between `number` and `number-1`, the rightmost bit of one in the original `number` would be turned off (from one to zero).

Based on the above trick, we could apply it to figure out the common prefix of two bit strings.

> The idea is that for a given range $[m, n]$ (*i.e.* $m < n$), we could iteratively apply the trick on the number n to *turn off* its rightmost bit of one until it becomes less or equal than the beginning of the range (m), which we denote as n'. Finally, we do AND operation between n' and m to obtain the final result.

By applying the Brian Kernighan's algorithm, we basically turn off the bits that lies on the right side of the *common prefix*, from the ending number n. With the rest of bits reset, we then can easily obtain the desired result.



In the example (`m=9, n=12`) shown in the above figure, the common prefix would be `00001`. After applying the Brian Kernighan's algorithm on the number `n`, its trailing 3 bits would all become zeros. Finally, we apply the AND operation between the reduced `n` and the `m` to obtain the common prefix.

Algorithm

Java | Python | Copy

```java
class Solution {
  public int rangeBitwiseAnd(int m, int n) {
    while (m < n) {
      // turn off rightmost 1-bit
      n = n & (n - 1);
    }
    return m & n;
  }
}
```

By the way, one could refer to the problem called Hamming distance as another exercise to apply the Brian Kernighan's algorithm.

Complexity Analysis

- Time Complexity: $\mathcal{O}(1)$.
  - Similar as the bit shift approach, the number of iteration in the algorithm is bounded by the number of bits in an integer number, which is constant.
  - Though having the same asymptotic complexity as the bit shift approach, the Brian Kernighan's algorithm requires less iterations, since it skips all the zero bits in between.
- Space Complexity: $\mathcal{O}(1)$, since no additional memory is consumed by the algorithm.

Rate this article:

Previous Next

Comments: 12 Sort By

Type comment here... (Markdown is supported)

Preview Post

eric_072 7 April 23, 2020 6:59 PM
beautiful solution
6 Share Reply

JT123 17 April 23, 2020 6:31 PM
for second solution, it doesn't look like you need to return m & n. You could just return n
4 Share Reply
SHOW 5 REPLIES

ukohank517 93 April 24, 2020 5:07 AM
It's amazing!!!
2 Share Reply

pmane4422 157 April 24, 2020 4:58 AM
premium-worth solution
2 Share Reply

joeldancastellon 2 April 24, 2020 1:55 AM
the complexity should really be log(max(m,n)) right? these could be arbitrarily big numbers
2 Share Reply
SHOW 3 REPLIES

sainvamshid 6 April 6, 2020 5:54 AM
Really solutions.
2 Share Reply

MercLabs 3 April 25, 2020 5:33 PM
Beautiful solution.. enjoyed it
1 Share Reply

aayushgarg 117 April 24, 2020 9:10 AM
In Approach 2, while returning, we can just return n

```
public int rangeBitwiseAnd(int m, int n) {
    while (m < n)
      n = n & (n - 1);
```

Read More
1 Share Reply

tarxvzf 20 May 10, 2020 1:56 AM
Simulation is also accepted with a simple optimization - return as soon as running ans is 0.

```
    public int rangeBitwiseAnd(int m, int n) {
        int ans = m;
        for (long i = ((long) m) + 1; i <= n && ans != 0; i++)
```

Read More
0 Share Reply
SHOW 1 REPLY

Sudhanshu1987 2 April 26, 2020 3:42 PM
loved the 2nd solution. I could come up with 1st one.. second one is a revelation
0 Share Reply

< 1 2 >