# 题目

给定范围 [m, n]，其中 0 <= m <= n <= 2147483647，返回此范围内所有数字的按位与（包含 m, n 两端点）。

**示例 1:**

输入: [5,7]

输出: 4

**示例 2:**

输入: [0,1]

输出: 0

# 分析

## 方法一：位运算（位移）

**思路：**

此题其实就是寻找[m,n]范围内二进制数高位（左边）没有变化的数，后面补上0即为所求的结果。

判断m、n是否相等，如果不相等，m+1会使m的二进制数末位进位，有进位说明m的末位肯定有0的情况，0与任何数相与皆得0，所以结果的末位肯定是0。同理，不断右移1位进行比较，直到最终 m=n 时，说明找到了[m,n]这个范围内高位没有变化的数，左移相同位数得到的结果就是所求的值。

**代码：**

class Solution {

public:

int rangeBitwiseAnd(int m, int n) {

int shift=0;

while(m < n)

{

m = m>>1;

n = n>>1;

shift++;

}

return m<<shift;

}

};

**复杂度分析**

时间复杂度：O(1)。虽然算法中有一个循环，但是迭代次数是由整数的位数限定的，所以迭代次数是固定的。因此，算法的时间复杂度是常数级别的。

空间复杂度：O(1)，不管输入是什么，我们的内存消耗是常数的。

或：

class Solution {

public:

int rangeBitwiseAnd(int m, int n) {

while(n>m)

n = n&(n-1);

return n;

}

};

或：

通过数学推导，可以得出题目其实等价于求m和n从高位开始，有多少位是相同的，这些相同的位组成的数就是答案

class Solution {

public:

int rangeBitwiseAnd(int m, int n) {

int res = 0;

int k = 1 << (sizeof(int) \* 8 - 2); // 非负整数的最大二次幂

while (k > 0 && (m & k) == (n & k)) {

res |= k & m;

k >>= 1;

}

return res;

}

};

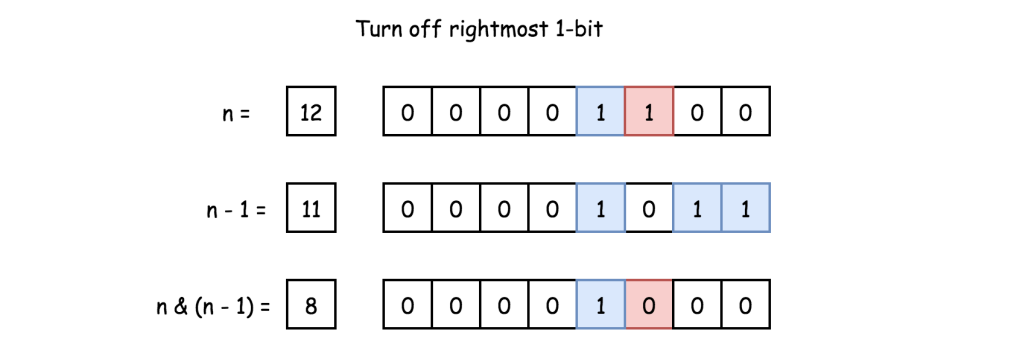
## 方法二：位运算（Brian Kernighan 算法）

**思路：**

说到位移，还有一个相关的算法叫做Brian Kernighan算法，它用于清除二进制串中最右边的1。

Brian Kernighan算法的秘诀总结如下：

当我们在number和number-1之间进行位运算时，原始number中最右边的 1将变为0。

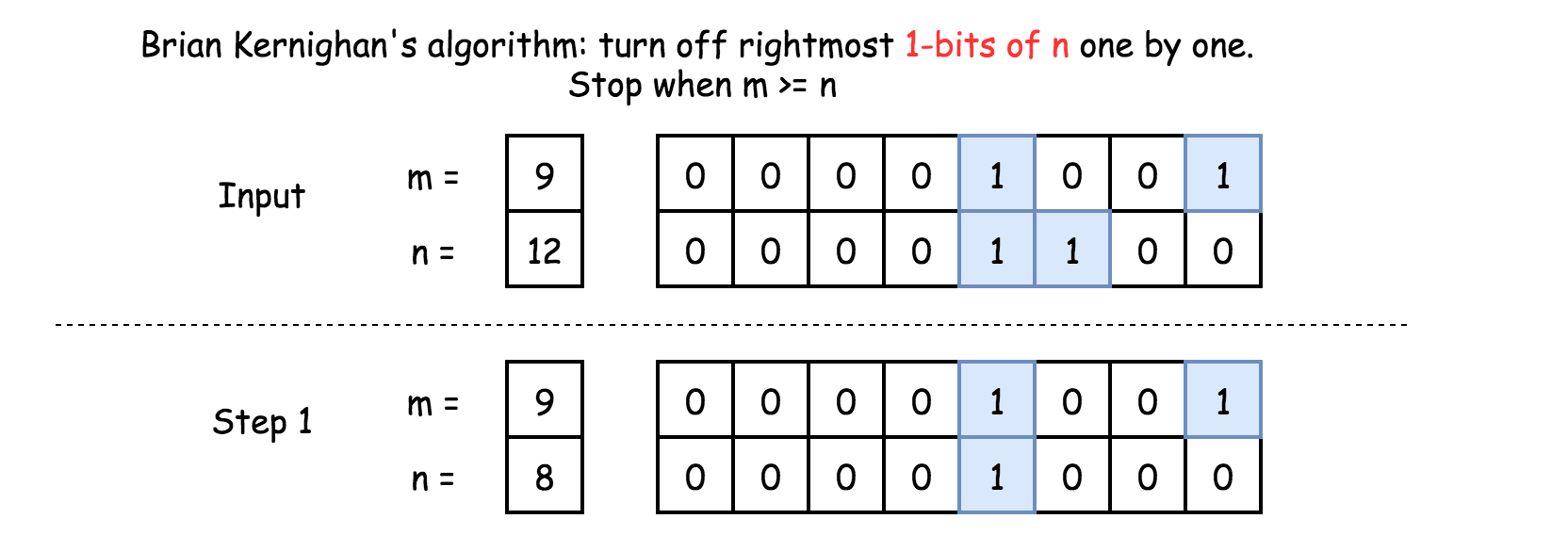


基于上述技巧，我们可以应用它来计算两个位字符串的公共前缀。

其思想是，对于给定的范围[m,n]（即m<nm<n），我们可以对数字n迭代的应用技巧，清除最右边的1，得到的数字我们表示为n。直到它小于或等于m，最后，我们在n和m之间进行操作以获得最终结果。

通过应用Brian Kernighan算法，我们基本上清除了公共前缀右侧的位。

将剩下的位复原后，我们就可以得到结果。



在上图所示的示例（m=9, n=12）中，公共前缀是00001。在对数字n应用Brian Kernighan算法后，后面三位都将变为零。最后，我们将n和m进行与操作获得公共前缀。

**代码：**

class Solution {

public int rangeBitwiseAnd(int m, int n) {

while (m < n) {

// turn off rightmost 1-bit

n = n & (n - 1);

}

return m & n;

}

}

汉明距离可以作为另个一练习应用 Brian Kernighan 算法的题目。

**复杂度分析：**

时间复杂度：O(1)。与位移方法类似，该算法中的迭代次数由整数中的位数限定。尽管与位移方法具有相同的渐近复杂度，但Brian Kernighan的算法需要的迭代次数较少，因为它跳过了这两个数字之间的所有零位。

空间复杂度：O(1)，没有使用额外的空间。