题目描述

两个整数之间的汉明距离指的是这两个数字对应二进制位不同的位置的数目。

给出两个整数 `x` 和 `y`，计算它们之间的汉明距离。

示例：

```
输入: x = 1, y = 4

输出: 2

解释:
1   (0 0 0 1)
4   (0 1 0 0)
       ↑   ↑
```

题目解析

首先通过 异或 操作找出两个数字对应位不同的位置，然后统计这些位置的个数。

统计解法借鉴Java中Integer.bitCount()方法源码来进行讲解，通过固定步数得到异或后1的个数。

第一步：将奇数位与偶数位相加，可以得出每两位1的个数，并将个数记录在这两位空间中

> *i = i - ((i >>> 1) & 0x55555555)*

```
0x55555555 => 01 01 01 01 ... 01 01
i           & 0x55555555    取出奇数位的1
(i >>> 1)   & 0x55555555    取出偶数位的1
比如，两位的情况下总共就四种情况：00 11 01 10
假设 i = 00 11 01 10
i           & 0x55555555 = 00 11 01 10
                           01 01 01 01
                           -----------
                           00 01 01 00
(i >>> 1)   & 0x55555555 = 00 01 10 11
                                                    01 01 01 01
                                                    -----------
                                                    00 01 00 01
将奇数位1的个数与偶数位的1求和：
00 01 01 00
00 01 00 01
-----------
00 10 01 01
结合原数字可以看出，00（00：没有1） 11（10：两个1） 01（01：1个1） 10（01：1个1）

每两位在通过加法统计时，总共如下四种情况[i & 01 +  (i>>>1) & 01]：
11: 01 + 01 = 10 = 2, 10: 00 + 01 = 01 = 1, 01: 01 + 00 = 01 = 1, 00: 00 + 00 = 00
= 0
每两位在通过减法统计时，总共如下四种情况[i -  (i>>>1) & 01]：
11: 11 - 01 = 10 = 2, 10: 10 - 01 = 01 = 1, 01: 01 - 00 = 01 = 1, 00: 00 + 00 = 00
= 0
可以发现结果是一样的，但是少了一次位运算！
```

> *在将每两位1的个数统计完之后，就可以开始两位两位、四位四位...相加求出1的总数*

第二步：通过相邻两位1的个数相加，求出每四位包含1的个数，并将结果存储在所在的四位中

> *i = (i & 0x33333333) + ((i >>> 2) & 0x33333333)*

```
0x55555555 => 0011 0011 0011 ... 0011 0011
继续上一步的结果向下进行：00 10 01 01
i           & 0x33333333 = 0010 0101
                                                    0011 0011
                                                    ---------
                                                    0010 0001
(i >>> 2)   & 0x33333333 = 0000 1001
                                                    0011 0011
                                                    ---------
                                                    0000 0001

就和得出每四位所包含1的个数
0010 0001
0000 0001
---------
0010 0010
结合原数字可以看出，0011(0010:有两个1)  0110(0010:有两个1)
```

第三步：通过相邻四位1的个数相加，求出每八位包含1的个数，并将结果存储在所在的八位中

> *i = (i + (i >>> 4)) & 0x0f0f0f0f;*

```
0x0f0f0f0f => 00001111 ... 00001111
继续上一步的结果向下进行：0010 0010
i           & 0x0f0f0f0f = 00100010
                                                00001111
                                                --------
                                                00000010
(i >>> 4)   & 0x0f0f0f0f = 00000010
                                                00001111
                                                --------
                                                00000010
就和得出每八位所包含1的个数
00000010
00000010
--------
00000100
结合原数字可以看出，00110110(00000100:有四个1)

源码中直接先将相邻四位进行相加，然后做了一次无用位清除
```

第四步：通过相邻八位1的个数相加，求出每十六位包含1的个数，并将结果存储在所在的十六位中

> *i = i + (i >>> 8);*

> *可以理解为 (i & 0x0f0f0f0f) + ((i >>> 8) & 0x0f0f0f0f);*
>
> *0x0f0f0f0f => 00000000111111110000000011111111*

第五步：通过将int类型前十六位1的个数与后16位1的个数相加，求出int中所有1的个数

> *i = i + (i >>> 16);*
>
> *可以理解为 (i & 0x0000ffff) + ((i >>> 8) & 0x0000ffff);*
>
> *0x0000ffff => 00000000000000001111111111111111*

第六步：去除无用的位

> *return i & 0x3f;*
>
> *int类型32位，即最多0x100000个1，除此之外左边的位都是无用的。*
> *0x3f => 00111111*

动画理解

参考代码

```
class Solution {
    public int hammingDistance(int x, int y) {
        return Integer.bitCount(x ^ y);
    }
}
```

bitCount源码：

```
public static int bitCount(int i) {
    i = i - ((i >>> 1) & 0x55555555);
    i = (i & 0x33333333) + ((i >>> 2) & 0x33333333);
    i = (i + (i >>> 4)) & 0x0f0f0f0f;
    i = i + (i >>> 8);
    i = i + (i >>> 16);
    return i & 0x3f;
}
```

复杂度分析

时间复杂度：O(1)

空间复杂度：O(1)