Reverse Bits

Reverse bits of a given 32 bits unsigned integer.

For example, given input 43261596 (represented in binary as **00000010100101000001111010011100**), return 964176192 (represented in binary as **00111001011110000010100101000000**).

Follow up:
If this function is called many times, how would you optimize it?

Related problem: Reverse Integer

Credits:
Special thanks to @ts for adding this problem and creating all test cases.

Solution 1

```
class Solution {
public:
    uint32_t reverseBits(uint32_t n) {
        n = (n >> 16) | (n << 16);
        n = ((n & 0xff00ff00) >> 8) | ((n & 0x00ff00ff) << 8);
        n = ((n & 0xf0f0f0f0) >> 4) | ((n & 0x0f0f0f0f) << 4);
        n = ((n & 0xcccccccc) >> 2) | ((n & 0x33333333) << 2);
        n = ((n & 0xaaaaaaaa) >> 1) | ((n & 0x55555555) << 1);
        return n;
    }
};
```

for 8 bit binary number *abcdefgh*, the process is as follow:

abcdefgh -> efghabcd -> ghefcdab -> hgfedcba

written by tworuler original link here

Solution 2

The Java solution is straightforward, just bitwise operation:

```
public int reverseBits(int n) {
    int result = 0;
    for (int i = 0; i < 32; i++) {
        result += n & 1;
        n >>>= 1;   // CATCH: must do unsigned shift
        if (i < 31) // CATCH: for last digit, don't shift!
            result <<= 1;
    }
    return result;
}
```

How to optimize if this function is called multiple times? We can divide an int into 4 bytes, and reverse each byte then combine into an int. For each byte, we can use cache to improve performance.

```
// cache
private final Map<Byte, Integer> cache = new HashMap<Byte, Integer>();
public int reverseBits(int n) {
    byte[] bytes = new byte[4];
    for (int i = 0; i < 4; i++) // convert int into 4 bytes
        bytes[i] = (byte)((n >>> 8*i) & 0xFF);
    int result = 0;
    for (int i = 0; i < 4; i++) {
        result += reverseByte(bytes[i]); // reverse per byte
        if (i < 3)
            result <<= 8;
    }
    return result;
}

private int reverseByte(byte b) {
    Integer value = cache.get(b); // first look up from cache
    if (value != null)
        return value;
    value = 0;
    // reverse by bit
    for (int i = 0; i < 8; i++) {
        value += ((b >>> i) & 1);
        if (i < 7)
            value <<= 1;
    }
    cache.put(b, value);
    return value;
}
```

written by AlexTheGreat original link here

Solution 3

```
uint32_t reverseBits(uint32_t n) {
    uint32_t m = 0;
    for (int i = 0; i < 32; i++, n >>= 1) {
        m <<= 1;
        m |= n & 1;
    }
    return m;
}
```

The process is straightforward, just iterate over all bits.

written by xcv58 original link here