counting-bits

```
package algorithm.bit;

/**
 * https://leetcode.com/problems/counting-bits/

 Given a non negative integer number num. For every numbers i in the range

 Example:
 For num = 5 you should return [0,1,1,2,1,2].

 Follow up:

 It is very easy to come up with a solution with run time O(n*sizeof(intege
 Space complexity should be O(n).
 Can you do it like a boss? Do it without using any builtin function like _

 * @author xiaobaoqiu  Date: 16-5-17 Time:  下午9:10
 */
public class CountingBits {
    public static void main(String[] args) {
        int n = 5;
        int[] ret = countBits(n);
        for (int v : ret) {
            System.out.println(v);
        }
    }

    /**
     * 5 ms
     */
    public static int[] countBits(int num) {
        int[] ret = new int[num + 1];
        for (int i = 0; i <= num; i++) {
            int b = 0, value = i;
            while(value != 0) {
                if((value & 1) == 1) b += 1;
                value >>= 1;
            }
            ret[i] = b;
        }

        return ret;
    }
}
```

majority-element

```
package algorithm.bit;

import java.util.HashMap;
import java.util.Map;

/**
 * https://leetcode.com/problems/majority-element/
 * <p/>
 * Given an array of size n, find the majority element. The majority elemer
 * <p/>
 * You may assume that the array is non-empty and the majority element alwa
 *
 * @author xiaobaoqiu  Date: 16-5-22 Time:  上午11:24
 */
public class MajorityElement {
    public static void main(String[] args) {
        int[] nums = new int[]{0};
//        int[] nums = new int[]{1, 2, 3, 1, 2, 1, 4, 1, 1};
//        int[] nums = new int[]{3, 3, 4};
//        int[] nums = new int[]{6, 6, 6, 7, 7};
        System.out.println(majorityElement(nums));
//        System.out.println(majorityElement_1(nums));
    }

    /**
     * 异或
     *
     * 3 ms
     * Your runtime beats 38.60% of java submissions
     */
    public static int majorityElement(int[] nums) {
        int candidate = nums[0], count = 1;
        for (int i = 1; i < nums.length; i++) {
            if ((nums[i] ^ candidate) == 0) count += 1;
            else {
                count -= 1;
                if (count == 0) {
                    candidate = nums[i];
                    count = 1;
                }
            }
        }

        return candidate;
    }

    /**
     * Hashmap
     *
     * 40 ms
     * Your runtime beats 4.62% of java submissions.
     */
    public static int majorityElement_1(int[] nums) {
        Map<Integer, Integer> counter = new HashMap<Integer, Integer>();
```

```
        int candidate = 0, max = 0;
        for (int v : nums) {
            if (counter.containsKey(v)) counter.put(v, counter.get(v) + 1);
            else counter.put(v, 1);

            int count = counter.get(v);
            if (count > max) {candidate = v; max = count;}
        }

        return candidate;
    }
}
```

number-of-1-bits

```
package algorithm.bit;

/**
 * https://leetcode.com/problems/number-of-1-bits/
 *
 Write a function that takes an unsigned integer and returns the number of
 (also known as the Hamming weight).

 For example, the 32-bit integer ’11' has binary representation 00000000000
 so the function should return 3.
 * @author xiaobaoqiu  Date: 16-5-25 Time:  下午11:28
 */
public class NumberOf1Bits {
    public static void main(String[] args) {
//        int n = 11;
//        int n = -2147483648;    //1
//        int n = (int)4294967295L; //32
        int n = (int)0; //0
        System.out.println(hammingWeight(n));
    }

    /**
     * 2 ms
     * Your runtime beats 12.55% of java submissions.
     */
    public static int hammingWeight(int n) {
        long x = n;
        if (n < 0) {
            x = n & Integer.MAX_VALUE;
            x |= 0x80000000L;
        }
        int count = 0;
        while (x > 0) {
            if ((x ^ (x-1)) == 1) count++;
            x >>= 1;
        }
        return count;
    }
}
```

power-of-four

```
package algorithm.bit;

/**
 * https://leetcode.com/problems/power-of-four/
 *
 Given an integer (signed 32 bits), write a function to check whether it is

 Example:
 Given num = 16, return true. Given num = 5, return false.

 Follow up: Could you solve it without loops/recursion?

 *
 * @author xiaobaoqiu  Date: 16-6-2 Time:  下午9:59
 */
public class PowerOfFour {
    public static void main(String[] args) {
        int num = 16;
        System.out.println(isPowerOfFour_1(num));
    }

    /**
     * 思路: log(x)/log(4) 为整数
     *
     *  2 ms
     *  Your runtime beats 22.59% of java submissions
     */
    public static boolean isPowerOfFour(int num) {
        if (num <= 0) return false;

        double value = Math.log(num)/Math.log(4);
        return value - (int)value < 1e-10;
    }

    /**
     * 二进制的奇数位为 1
     *
     * 2 ms
     * Your runtime beats 22.59% of java submissions
     */
    public static boolean isPowerOfFour_1(int num) {
        return num > 0 && ((num & (num - 1)) == 0) && (0x55555555 & num) !=
    }
}
```

power-of-two

```
package algorithm.bit;

/**
 * https://leetcode.com/problems/power-of-two/
 *
 * Given an integer, write a function to determine if it is a power of two
 *
 * @author xiaobaoqiu  Date: 16-5-27 Time:  下午10:23
 */
public class PowerOfTwo {
    public static void main(String[] args) {
        int n = 1;
        System.out.println(isPowerOfTwo(n));
    }

    /**
     * 2 ms
     * Your runtime beats 19.36% of java submissions
     */
    public static boolean isPowerOfTwo(int n) {
//        return n > 0 && (n & (n-1)) == 0;
        if (n < 1) return false;
        return (n & (n-1)) == 0;
    }
}
```

single-number

```
package algorithm.bit;

/**
 * https://leetcode.com/problems/single-number/
 *
 Given an array of integers, every element appears twice except for one. Fi

 Note:
 Your algorithm should have a linear runtime complexity. Could you implemer

 * @author xiaobaoqiu  Date: 16-5-19 Time:  下午10:00
 */
public class SingleNumber {

    public static void main(String[] args) {
        int[] nums = new int[] {1, 2, 3, 2, 1, 4, 4};
        System.out.println(singleNumber(nums));
    }

    /**
     * 2 ms
     * Your runtime beats 31.74% of java submissions
     */
    public static int singleNumber(int[] nums) {
        int ret = 0;
        for (int v : nums) ret ^= v;
        return ret;
    }
}
```

single-number-iii

```
package algorithm.bit;

import java.util.HashSet;
import java.util.Set;

/**
 * https://leetcode.com/problems/single-number-iii/

 Given an array of numbers nums, in which exactly two elements appear only

 For example:

 Given nums = [1, 2, 1, 3, 2, 5], return [3, 5].

 Note:
 The order of the result is not important. So in the above example, [5, 3]
 Your algorithm should run in linear runtime complexity. Could you implemer

 * @author xiaobaoqiu  Date: 16-5-17 Time:  下午9:57
 */
public class SingleNumberIII {
    public static void main(String[] args) {
        int[] nums = new int[]{1, 2, 1, 3, 2, 5};
        for (int v : nums) System.out.print(v + " , ");
        System.out.println();

//        nums = singleNumber(nums);
        nums = singleNumber_1(nums);
        for (int v : nums) System.out.print(v + " , ");
        System.out.println();
    }

    private static String toFullBinaryString(int x) {
        int[] buffer = new int[Integer.SIZE];
        for (int i = (Integer.SIZE - 1); i >= 0; i--) {
            buffer[i] = x >> i & 1;
        }
        String s = "";
        for (int j = (Integer.SIZE - 1); j >= 0; j--) {
            s = s + buffer[j];
        }
        return s;
    }

    /**
     * 思路：使用Set,不是常量的空间，最多会是 （n-2）/2 的空间
     *
     * 12 ms
     * Your runtime beats 14.33% of java submissions.
     */
    public static int[] singleNumber(int[] nums) {
        Set<Integer> set = new HashSet<Integer>();
        for (int v : nums) {
            if (set.contains(v)) set.remove(v);
```

```
            else set.add(v);
        }

        int[] ret = new int[set.size()];
        int i = 0;
        for (Integer v : set) ret[i++] = v;
        return ret;
    }

    /**
     * 思路：位运算
     * 2 ms
     * Your runtime beats 37.32% of java submissions
     *
     * 记 aXorB = a ^ b，则有 aXorB ^ a == b，而 aXorB ^ b == a
     * 所有，如果我们能将原始数据分成两组，其中一组包含 a（记为数组A），另外一组包含 b
     * 则将 aXorB 异或 数组A中所有数据的结果 = aXorB 异或 a，即b
     * 同理，将 aXorB 异或 数组B中所有数据的结果 = aXorB 异或 b，即a
     *
     * 如何分成两组： aXorB的某一位为1，说明这个bit上 a 和 b是不相同的，通过这个 bi
     */
    public static int[] singleNumber_1(int[] nums) {
        int aXorB = 0;  //a,b 为最终的两位数，aXorB为 a异或b的结果
        for (int v : nums) aXorB ^= v;

        //aXorB 最高位的1，说明这个bit上，a和b是不同的，因此通过这个就可以区分出两个
        int lowestOneBit = Integer.lowestOneBit(aXorB);

        int[] ret = new int[]{aXorB, aXorB};
        for (int v : nums) {
            if ((v & lowestOneBit) == 0) {  // 只能区分 0 和 非0
                ret[0] ^= v;
            } else {
                ret[1] ^= v;
            }
        }
        return ret;
    }
}
```

sum-of-two-integers

```
package algorithm.bit;

/**
 * https://leetcode.com/problems/sum-of-two-integers/
 *
 Calculate the sum of two integers a and b,
 but you are not allowed to use the operator + and -.

 Example:
 Given a = 1 and b = 2, return 3.

 * @author xiaobaoqiu  Date: 16-7-12 Time:  下午10:13
 */
public class SumOfTwoIntegers {
    public static void main(String[] args) {
        getSum(1, 3);
    }

    /**
     * 思路：位运算
     *
     * 0 ms
     * Your runtime beats 7.29% of java submissions.
     */
    public static int getSum(int a, int b) {
        int carry;
        while(true) {
//            System.out.print(a + " + " + b + " --> ");
            carry = a & b;
            a = a ^ b;
//            System.out.println(a + ", carry = " + carry);
            b = carry << 1;
            if (b == 0) break;
        }
        return a;
    }
}
```