house-robber

```
package algorithm.dp;

/**
 * https://leetcode.com/problems/house-robber/
 *
 You are a professional robber planning to rob houses along a street.
 Each house has a certain amount of money stashed,
 the only constraint stopping you from robbing each of them is that
 adjacent houses have security system connected and it will
 automatically contact the police if two adjacent houses were
 broken into on the same night.

 Given a list of non-negative integers representing
 the amount of money of each house,
 determine the maximum amount of money you can rob tonight
 without alerting the police

 * @author xiaobaoqiu  Date: 16-5-28 Time:  上午12:46
 */
public class HouseRobber {
    public static void main(String[] args) {
        int[] nums = new int[]{1,1,2,1};
//        int[] nums = new int[]{6, 2, 5, 7, 9, 3, 1, 4, 8};
        System.out.println(rob(nums));
    }

    /**
     * DP
     * s(n)表示前n个房子能偷到的最大值
     *
     * s(n) = max(max(s(n-2), s(n-3)) + nums[n], s[n-1])
     *
     * 1 ms
     * Your runtime beats 4.35% of java submissions
     */
    public static int rob(int[] nums) {
        if (nums == null || nums.length == 0) return 0;
        if (nums.length == 1) return nums[0];
        if (nums.length == 2) return Math.max(nums[0], nums[1]);
        if (nums.length == 3) return Math.max(nums[1], nums[0] + nums[2]);

        int[] s = new int[4];
        s[0] = nums[0];
        s[1] = Math.max(nums[0], nums[1]);
        s[2] = Math.max(nums[1], nums[0] + nums[2]);

        int ret = 0;
        for (int i=3; i<nums.length; i++) {
            ret = Math.max(Math.max(s[1], s[0]) + nums[i], s[2]);
            s[0] = s[1];
            s[1] = s[2];
            s[2] = ret;
            //System.out.println(i + " --> " + ret);
        }
        return ret;
    }
}
```

integer-break

```
package algorithm.dp;

/**
 * https://leetcode.com/problems/integer-break/
 * <p/>
 * Given a positive integer n, break it into the sum of at least two posit
 * Return the maximum product you can get.
 * <p/>
 * For example,
 * given n = 2, return 1 (2 = 1 + 1);
 * given n = 10, return 36 (10 = 3 + 3 + 4).
 * <p/>
 * Note: you may assume that n is not less than 2.
 *
 * @author xiaobaoqiu  Date: 16-5-23 Time:  下午11:24
 */
public class IntegerBreak {
    public static void main(String[] args) {
        System.out.println(integerBreak(11));   //54
    }

    /**
     * DP
     * 2 3 4应该是基数
     * f(n) = max(f(n-2)*2, f(n-3)*3, f(n-4)*4)
     *
     * 0 ms
     */
    public static int integerBreak(int n) {
        //0 -> 10, 环形队列
        int[] queue = new int[]{0, 0, 1, 2, 4, 6, 9, 12, 18, 27, 36};
        if (n <= 10) return queue[n];
        int curPos = 0; // 下一个写数据的下标

        int cur = 0;
        for (int i = 11; i <= n; i++) {
            cur = Math.max(Math.max(queue[(i - 2) % 11] * 2, queue[(i - 3)
            queue[curPos] = cur;
            curPos = (curPos + 1) % 11;
        }

        return cur;
    }
}
```

range-sum-query-immutable

```
package algorithm.dp;

/**
 * https://leetcode.com/problems/range-sum-query-immutable/
 *
 Given an integer array nums, find the sum of the elements
 between indices i and j (i ≤ j), inclusive.

 Example:
 Given nums = [-2, 0, 3, -5, 2, -1]

 sumRange(0, 2) -> 1
 sumRange(2, 5) -> -1
 sumRange(0, 5) -> -3
 Note:
 You may assume that the array does not change.
 There are many calls to sumRange function.

 * @author xiaobaoqiu  Date: 16-7-8 Time:  下午10:48
 */
public class RangeSumQueryImmutable {
    public static void main(String[] args) {
        //0, -2, -2, 1, -4, -2, -3
        int[] nums = new int[]{-2, 0, 3, -5, 2, -1};
        NumArray numArray = new NumArray(nums);
        System.out.println(numArray.sumRange(0, 2));    //1
        System.out.println(numArray.sumRange(2, 5));    //-1
        System.out.println(numArray.sumRange(0, 5));    //3
    }

    /**
     * 思路：数组s, s[i] 表示 num[0] + ... + num[i-1]
     *
     * 3 ms
     * Your runtime beats 24.54% of java submissions
     */
    public static class NumArray {
        int s[];

        public NumArray(int[] nums) {
            s = new int[nums.length + 1];
            s[0] = 0;
            for (int i = 1; i <= nums.length; i++) {
                s[i] = s[i - 1] + nums[i - 1];
            }
        }

        public int sumRange(int i, int j) {
            return s[j + 1] - s[i];
        }
    }
}
```

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;
    }
}
```