## [剑指 Offer 14- I. 剪绳子](https://leetcode-cn.com/problems/jian-sheng-zi-lcof/)

给你一根长度为 n 的绳子，请把绳子剪成整数长度的 m 段（m、n都是整数，n>1并且m>1），每段绳子的长度记为 k[0],k[1]...k[m-1] 。请问 k[0]\*k[1]\*...\*k[m-1] 可能的最大乘积是多少？例如，当绳子的长度是8时，我们把它剪成长度分别为2、3、3的三段，此时得到的最大乘积是18。

示例 1：

输入: 2

输出: 1

解释: 2 = 1 + 1, 1 × 1 = 1

示例 2:

输入: 10

输出: 36

解释: 10 = 3 + 3 + 4, 3 × 3 × 4 = 36

提示：

2 <= n <= 58

**解：**

动态规划：

当n=2和n=3时拆分必有1，归入特殊情况，直接返回值。

当n>=4时，采取自底向上的动态规划算法。建立dp数组用来保存结果，dp[0]为n=2，dp[1]为n=3,dp[2]为n=4（n=4时拆分成2\*2，不符合状态转移方程）。

n>4时状态转移方程：dp[n] = max(dp[n], dp[n-i] \* n) (2 ≤ i ≤ n-2)

自底向上计算到输入的值以得到结果。

时间O() 空间O(n)

class Solution {

public:

    int cuttingRope(int n) {

        if(2 == n){

            return 1;

        }

        else if(3 == n){

            return 2;

        }

        else{

            vector<int> dp(n-1,0);

            dp[0] = 1; //n=2

            dp[1] = 3; //n=3

            dp[2] = 4; //n=4

            for(int i = 3; i <= n - 2; ++i){ //由于n从2开始所以n-2

                for(int j = 2; j <= i; ++j){

                    dp[i] = max(dp[i],j \* dp[i-j]);//状态转移方程

                }

            }

            return dp[n-2];

        }

    }

};

数论：

数学证明：定义函数表示将给定的正整数 n 拆分成尽可能多的正数 x 的情况下的最大乘积，则可以将 n 分成 项，此时 ，通过求导可得f(x)在x=e时取最大值，f(3)>f(2)，x=3 时，可以得到最大乘积。

n<4时同上一解法，n≥4时由于2 × 2 > 3 × 1 ，所以当n > 4时，不断从n中分出3，最终只会剩下2或3或4，都满足相乘最大。

时间O(n) 空间O(1)

class Solution {

public:

    int cuttingRope(int n) {

        if(2 == n){

            return 1;

        }

        else if(3 == n){

            return 2;

        }

        else{

            int res = 1;

            while(n > 4){

                res \*= 3;

                n -= 3;

            }

            res \*= n;

            return res;

        }

    }

};

## [剑指 Offer 14- II. 剪绳子 II](https://leetcode-cn.com/problems/jian-sheng-zi-ii-lcof/)

给你一根长度为 n 的绳子，请把绳子剪成整数长度的 m 段（m、n都是整数，n>1并且m>1），每段绳子的长度记为 k[0],k[1]...k[m-1] 。请问 k[0]\*k[1]\*...\*k[m-1] 可能的最大乘积是多少？例如，当绳子的长度是8时，我们把它剪成长度分别为2、3、3的三段，此时得到的最大乘积是18。

答案需要取模 1e9+7（1000000007），如计算初始结果为：1000000008，请返回 1。

示例 1：

输入: 2

输出: 1

解释: 2 = 1 + 1, 1 × 1 = 1

示例 2:

输入: 10

输出: 36

解释: 10 = 3 + 3 + 4, 3 × 3 × 4 = 36

提示：

2 <= n <= 58

**解：**

动态规划方法由于要用到max方法不方便取余，采用数论结合循环取余的方法。

时间O(n) 空间O(1)

class Solution {

public:

    int cuttingRope(int n) {

        if(2 == n){

            return 1;

        }

        else if(3 == n){

            return 2;

        }

        else{

            long res = 1;

            while(n > 4){

                res \*= 3;

                res = res % 1000000007;//循环取余

                n -= 3;

            }

            res \*= n;

            return res % 1000000007;

        }

    }

};

以下附快速幂求余：

long quickpow(int x, int n ,int mod){

    long res = 1;

    long t = x;

    while(n != 0){

        if((n & 1) == 1){//按位与，求奇偶

            res \*= t;

            res %= mod;

        }

        t \*= t;

        t %= mod;

        n /= 2;

    }

    return res;

}

## [剑指 Offer 15. 二进制中1的个数](https://leetcode-cn.com/problems/er-jin-zhi-zhong-1de-ge-shu-lcof/)

请实现一个函数，输入一个整数（以二进制串形式），输出该数二进制表示中 1 的个数。例如，把 9 表示成二进制是 1001，有 2 位是 1。因此，如果输入 9，则该函数输出 2。

示例 1：

输入：00000000000000000000000000001011

输出：3

解释：输入的二进制串 00000000000000000000000000001011 中，共有三位为 '1'。

示例 2：

输入：00000000000000000000000010000000

输出：1

解释：输入的二进制串 00000000000000000000000010000000 中，共有一位为 '1'。

示例 3：

输入：11111111111111111111111111111101

输出：31

解释：输入的二进制串 11111111111111111111111111111101 中，共有 31 位为 '1'。

提示：

输入必须是长度为 32 的 二进制串 。

**解：**

逐位右移与1相与：

时间O(1)（固定32位二进制数） 空间O(1)

class Solution {

public:

    int hammingWeight(uint32\_t n) {

        int res = 0;

        while(n != 0){

            if((n & 1) == 1){

                res++;

            }

            n >>= 1;

        }

        return res;

    }

};

n & (n-1)：

n - 1操作：

无论是借位，还是减1，遇到1就停止；

从低位到高位，一直到1，每一位都发生了反转。

每次n 与 n-1相与可以消除n中的一个1。

时间O(M)（M为n中1的数目） 空间O(1)

class Solution {

public:

    int hammingWeight(uint32\_t n) {

        int res = 0;

        while(n != 0){

            n = n & (n - 1);

            res++;

        }

        return res;

    }

};

## [剑指 Offer 16. 数值的整数次方](https://leetcode-cn.com/problems/shu-zhi-de-zheng-shu-ci-fang-lcof/)

实现函数double Power(double base, int exponent)，求base的exponent次方。不得使用库函数，同时不需要考虑大数问题。

示例 1:

输入: 2.00000, 10

输出: 1024.00000

示例 2:

输入: 2.10000, 3

输出: 9.26100

示例 3:

输入: 2.00000, -2

输出: 0.25000

解释: 2-2 = 1/22 = 1/4 = 0.25

说明:

-100.0 < x < 100.0

n 是 32 位有符号整数，其数值范围是 [−2^31, 2^31 − 1] 。

**解：**

快速幂：

时间O(logn) 空间O(1)

class Solution {

public:

    double myPow(double x, int n) {

        double res = 1;

        long t = n;//当n=-2^31时取反会超出int范围

        if(t < 0){

            x = 1 / x;

            t = -t;

        }

        while(t > 0){

            if(t & 1){

                res \*= x;

            }

            x \*= x;

            t >>= 1;

        }

        return res;

    }

};

## [剑指 Offer 17. 打印从1到最大的n位数](https://leetcode-cn.com/problems/da-yin-cong-1dao-zui-da-de-nwei-shu-lcof/)

输入数字 n，按顺序打印出从 1 到最大的 n 位十进制数。比如输入 3，则打印出 1、2、3 一直到最大的 3 位数 999。

示例 1:

输入: n = 1

输出: [1,2,3,4,5,6,7,8,9]

说明：

用返回一个整数列表来代替打印

n 为正整数

**解：**

针对本题的简单解法：

由于本题不会出现大数，可采取以下解法

时间O() 空间O(1)

class Solution {

public:

    vector<int> printNumbers(int n) {

        vector<int> res;

        int max = pow(10,n);

        for(int i = 1; i < max; ++i){

            res.push\_back(i);

        }

        return res;

    }

};

        else{

            return -1;

        }

    }

    void adjust(){

        int tmp;

        while(!s1.empty()){

            tmp = s1.top();

            s1.pop();

            s2.push(tmp);

        }

    }

};

大数解法：

使用string模拟加法过程。

class Solution {

public:

    vector<int> res;

    vector<int> printNumbers(int n) {

        if (n <= 0) return res;

        //创建一个能容纳最大值的字符数组

        string number(n, '0');

        //初始全部设置为0

        while (!Increment(number))

        {

            saveNumber(number);

        }

        return res;

    }

    bool Increment(string& number) {

        //注意要使用引用传递，否则无法修改number

        bool isOverflow = false;//检测是否越界

        int nTakeOver = 0;//存储进位

        int nLength = number.size();

        for (int i = nLength - 1; i >= 0; i--)

        {

            int nSum = number[i] - '0' + nTakeOver;

            if (i == nLength - 1)

                //如果是第一位，进位

            {

                nSum++;

            }

            if (nSum >= 10)//有进位

            {

                if (i == 0)

                    //如果是最高位有进位，说明超过了给定得到最大值，越界

                {

                    isOverflow = true;

                }

                else

                {

                    nTakeOver = 1;

                    number[i] = nSum - 10 + '0';//对第i位进行设置

                }

            }

            else//没有进位

                //设置第i位数字

                //并直接跳出循环

            {

                number[i] = nSum + '0';

                break;

            }

        }

        return isOverflow;

    }

    void saveNumber(string number)

        //由于此处输出，不需要修改number，因此不需要加引用

    {

        string s = "";

        bool isBegin0 = true;

        string::iterator it = number.begin();

        while (it != number.end())

        {

            if (isBegin0 && \*it != '0') isBegin0 = false;

            if (!isBegin0)

            {

                s += \*it;

            }

            it++;

        }

        int num = stoi(s);

        res.push\_back(num);

    }

};

## [剑指 Offer 18. 删除链表的节点](https://leetcode-cn.com/problems/shan-chu-lian-biao-de-jie-dian-lcof/)

给定单向链表的头指针和一个要删除的节点的值，定义一个函数删除该节点。

返回删除后的链表的头节点。

示例 1:

输入: head = [4,5,1,9], val = 5

输出: [4,1,9]

解释: 给定你链表中值为 5 的第二个节点，那么在调用了你的函数之后，该链表应变为 4 -> 1 -> 9.

示例 2:

输入: head = [4,5,1,9], val = 1

输出: [4,5,9]

解释: 给定你链表中值为 1 的第三个节点，那么在调用了你的函数之后，该链表应变为 4 -> 5 -> 9.

说明：

题目保证链表中节点的值互不相同

若使用 C 或 C++ 语言，你不需要 free 或 delete 被删除的节点

**解：**

双指针：

在头结点前建立节点p指向头结点，建立q节点同p节点。对q->next的值进行遍历，若q->next的值等于val，则q->next = q->next->next，即删除当前q->next对应节点。返回p->next。

时间O(1)（最坏情况下O(n)）空间O(1)

/\*\*

 \* Definition for singly-linked list.

 \* struct ListNode {

 \*     int val;

 \*     ListNode \*next;

 \*     ListNode(int x) : val(x), next(NULL) {}

 \* };

 \*/

class Solution {

public:

    ListNode\* deleteNode(ListNode\* head, int val) {

        ListNode\* p = new ListNode();

        p->next = head;

        ListNode\* q = p;

        while(q->next->val != val){

            q = q->next;

        }

        q->next = q->next->next;

        return p->next;

    }

};

递归：

空间复杂度增加

class Solution {

public:

    ListNode\* deleteNode(ListNode\* head, int val) {

        if(head->next == NULL)

            return NULL;

        if(head->val == val)

            return head->next;

        head->next = deleteNode(head->next,val);

        return head;

    }

};

## [剑指 Offer 19. 正则表达式匹配](https://leetcode-cn.com/problems/zheng-ze-biao-da-shi-pi-pei-lcof/)

题见链接

**解：**

动态规划：

https://leetcode-cn.com/problems/zheng-ze-biao-da-shi-pi-pei-lcof/solution/jian-zhi-offer-19-zheng-ze-biao-da-shi-pi-pei-dong/

class Solution {

public:

    bool isMatch(string s, string p) {

        int m = s.size() + 1, n = p.size() + 1;

        vector<vector<bool>> dp(m, vector<bool>(n, false));

        dp[0][0] = true;

        for(int j = 2; j < n; j += 2)

            dp[0][j] = dp[0][j - 2] && p[j - 1] == '\*';

        for(int i = 1; i < m; i++) {

            for(int j = 1; j < n; j++) {

                dp[i][j] = p[j - 1] == '\*' ?

                    dp[i][j - 1] || dp[i][j - 2] || dp[i - 1][j] && (s[i - 1] == p[j - 2] || p[j - 2] == '.'):

                    dp[i - 1][j - 1] && (p[j - 1] == '.' || s[i - 1] == p[j - 1]);

            }

        }

        return dp[m - 1][n - 1];

    }

};

## [剑指 Offer 20. 表示数值的字符串](https://leetcode-cn.com/problems/biao-shi-shu-zhi-de-zi-fu-chuan-lcof/)

请实现一个函数用来判断字符串是否表示数值（包括整数和小数）。例如，字符串"+100"、"5e2"、"-123"、"3.1416"、"-1E-16"、"0123"都表示数值，但"12e"、"1a3.14"、"1.2.3"、"+-5"及"12e+5.4"都不是。

**解：**

有限状态自动机：

<https://leetcode-cn.com/problems/biao-shi-shu-zhi-de-zi-fu-chuan-lcof/solution/mian-shi-ti-20-biao-shi-shu-zhi-de-zi-fu-chuan-y-2/>

class Solution {

public:

    enum State {

        STATE\_INITIAL,

        STATE\_INT\_SIGN,

        STATE\_INTEGER,

        STATE\_POINT,

        STATE\_POINT\_WITHOUT\_INT,

        STATE\_FRACTION,

        STATE\_EXP,

        STATE\_EXP\_SIGN,

        STATE\_EXP\_NUMBER,

        STATE\_END,

    };

    enum CharType {

        CHAR\_NUMBER,

        CHAR\_EXP,

        CHAR\_POINT,

        CHAR\_SIGN,

        CHAR\_SPACE,

        CHAR\_ILLEGAL,

    };

    CharType toCharType(char ch) {

        if (ch >= '0' && ch <= '9') {

            return CHAR\_NUMBER;

        } else if (ch == 'e' || ch == 'E') {

            return CHAR\_EXP;

        } else if (ch == '.') {

            return CHAR\_POINT;

        } else if (ch == '+' || ch == '-') {

            return CHAR\_SIGN;

        } else if (ch == ' ') {

            return CHAR\_SPACE;

        } else {

            return CHAR\_ILLEGAL;

        }

    }

    bool isNumber(string s) {

        unordered\_map<State, unordered\_map<CharType, State>> transfer{

            {

                STATE\_INITIAL, {

                    {CHAR\_SPACE, STATE\_INITIAL},

                    {CHAR\_NUMBER, STATE\_INTEGER},

                    {CHAR\_POINT, STATE\_POINT\_WITHOUT\_INT},

                    {CHAR\_SIGN, STATE\_INT\_SIGN},

                }

            }, {

                STATE\_INT\_SIGN, {

                    {CHAR\_NUMBER, STATE\_INTEGER},

                    {CHAR\_POINT, STATE\_POINT\_WITHOUT\_INT},

                }

            }, {

                STATE\_INTEGER, {

                    {CHAR\_NUMBER, STATE\_INTEGER},

                    {CHAR\_EXP, STATE\_EXP},

                    {CHAR\_POINT, STATE\_POINT},

                    {CHAR\_SPACE, STATE\_END},

                }

            }, {

                STATE\_POINT, {

                    {CHAR\_NUMBER, STATE\_FRACTION},

                    {CHAR\_EXP, STATE\_EXP},

                    {CHAR\_SPACE, STATE\_END},

                }

            }, {

                STATE\_POINT\_WITHOUT\_INT, {

                    {CHAR\_NUMBER, STATE\_FRACTION},

                }

            }, {

                STATE\_FRACTION,

                {

                    {CHAR\_NUMBER, STATE\_FRACTION},

                    {CHAR\_EXP, STATE\_EXP},

                    {CHAR\_SPACE, STATE\_END},

                }

            }, {

                STATE\_EXP,

                {

                    {CHAR\_NUMBER, STATE\_EXP\_NUMBER},

                    {CHAR\_SIGN, STATE\_EXP\_SIGN},

                }

            }, {

                STATE\_EXP\_SIGN, {

                    {CHAR\_NUMBER, STATE\_EXP\_NUMBER},

                }

            }, {

                STATE\_EXP\_NUMBER, {

                    {CHAR\_NUMBER, STATE\_EXP\_NUMBER},

                    {CHAR\_SPACE, STATE\_END},

                }

            }, {

                STATE\_END, {

                    {CHAR\_SPACE, STATE\_END},

                }

            }

        };

        int len = s.length();

        State st = STATE\_INITIAL;

        for (int i = 0; i < len; i++) {

            CharType typ = toCharType(s[i]);

            if (transfer[st].find(typ) == transfer[st].end()) {

                return false;

            } else {

                st = transfer[st][typ];

            }

        }

        return st == STATE\_INTEGER || st == STATE\_POINT || st == STATE\_FRACTION || st == STATE\_EXP\_NUMBER || st == STATE\_END;

    }

};