例 1

完 美 平 方

1. 问题描述

给定一个正整数 n，找到若干个完全平方数(例如：1,4,9,…)，使得它们的和等于 n，完全平方数的个数最少。

2. 问题示例

给出 $n=12$，返回 3，因为 $12=4+4+4$；给出 $n=13$，返回 2，因为 $13=4+9$。

3. 代码实现

```
#参数 n 是一个正整数
#返回一个整数
class Solution:
    def numSquares(self, n):
        while n % 4 == 0:
            n //= 4
        if n % 8 == 7:
            return 4
        for i in range(n + 1):
            temp = i * i
            if temp <= n:
                if int((n - temp) * * 0.5) * * 2 + temp == n:
                    return 1 + (0 if temp == 0 else 1)
            else:
                break
        return 3
#主函数
if __name__ == '__main__':
```

```
n = 12
print("初始值: ", n)
solution = Solution()
print("结果: ", solution.numSquares(n))
```

4. 运行结果

初始值：12
结果：3

例 2

判断平方数

1. 问题描述

给定一个正整数 *num*，判断是否为完全平方数，要求当 *num* 为完全平方数时返回 True，否则返回 False。

2. 问题示例

输入 *num*＝16，输出 True，*sqrt*(16)＝4；输入 *num*＝15，输出 False，*sqrt*(15)＝3.87。

3. 代码实现

```
#参数 num 是一个正整数
#返回值是一个布尔值,如果 num 是完全平方数就返回 True,否则返回 False
class Solution:
    def isPerfectSquare(self, num):
        l = 0
        r = num
        while (r - l > 1):
            mid = (l + r) / 2
            if (mid * mid <= num):
                l = mid
            else:
                r = mid
        ans = l
        if (l * l < num):
            ans = r
        return ans * ans == num
#主函数
if __name__ == '__main__':
```

```
num = 16
print("初始值: ", num)
solution = Solution()
print("结果: ", solution.isPerfectSquare(num))
```

4. 运行结果

初始值：16
结果：True

例 3

检测 2 的幂次

1. 问题描述

检测一个整数 n 是否为 2 的幂次。

2. 问题示例

$n=4$，返回 True；$n=5$，返回 False。

3. 代码实现

```
#采用 UTF-8 编码格式
#参数 n 是一个整数
#返回 True 或者 False
class Solution:
    def checkPowerOf2(self, n):
        ans = 1
        for i in range(31):
            if ans == n:
                return True
            ans = ans << 1
        return False
if __name__ == '__main__':
    temp = Solution()
    nums1 = 16
    nums2 = 17
    print(("输入: " + str(nums1)))
    print(("输出: " + str(temp.checkPowerOf2(nums1))))
    print(("输入: " + str(nums2)))
    print(("输出: " + str(temp.checkPowerOf2(nums2))))
```

4. 运行结果

输入：16

输出：True

输入：17

输出：False

例 4

求 平 方 根

1. 问题描述

实现 int *sqrt*(int x)函数,计算并返回 x 的平方根。

2. 问题示例

sqrt(3)=1; *sqrt*(4)=2; *sqrt*(5)=2; *sqrt*(10)=3。

3. 代码实现

```
#采用 UTF-8 编码格式
#参数 x 是一个整数
#返回值是 x 的平方根
class Solution:
    def sqrt(self, x):
        l, r = 0, x
        while l + 1 < r:
            m = (r + l) // 2
            if m * m == x:
                return m
            elif m * m > x:
                r = m
            else:
                l = m
        if l * l == x:
            return l
        if r * r == x:
            return r
        return l
if __name__ == '__main__':
    temp = Solution()
```

```
x1 = 5
x2 = 10
print(("输入:" + str(x1)))
print(("输出:" + str(temp.sqrt(x1))))
print(("输入:" + str(x2)))
print(("输出:" + str(temp.sqrt(x2))))
```

4. 运行结果

输入：5
输出：2
输入：10
输出：3

例 5

x 的 n 次幂

1. 问题描述

实现函数 $Pow(x,n)$，计算并返回 x 的 n 次幂。

2. 问题示例

$Pow(2.1, 3)=9.261$；$Pow(0, 1)=0$；$Pow(1, 0)=1$。

3. 代码实现

```
#采用 UTF-8 编码格式
#参数 x 是一个 double 型的底数
#参数 n 是一个整数型的指数
#返回值是一个 double 类型的结果
class Solution:
    def myPow(self, x, n):                    #在 Python3 中整除需使用"//"
        if n < 0 :
            x = 1 // x
            n = -n
        ans = 1
        tmp = x
        while n != 0:
            if n % 2 == 1:
                ans *= tmp
            tmp *= tmp
            n //= 2
        return ans
if __name__ == '__main__':
    temp = Solution()
    num1 = 123
```

```
num2 = 3
print(("输入:" + str(num1) + " " + str(num2)))
print(("输出:" + str(temp.myPow(num1,num2))))
```

4. 运行结果

输入：123　3
输出：1860867

例 6

快　速　幂

1. 问题描述

计算 $a^n \% b$，其中 a、b 和 n 都是 32 位的非负整数。

2. 问题示例

例如：$2^{31} \% 3 = 2$。

3. 代码实现

```
#参数 a、b、n 是 32 位的非负整数
#返回值是一个整数
class Solution:
    def fastPower(self, a, b, n):
        ans = 1
        while n > 0:
            if n % 2 == 1:
                ans = ans * a % b
            a = a * a % b
            n = n / 2
        return ans % b
if __name__ == '__main__':
    a = int(input("请输入 a:"))
    n = int(input("请输入 n:"))
    b = int(input("请输入 b:"))
    solution = Solution()
    print("输出:", solution.fastPower(a, n, b))
```

4. 运行结果

请输入 a：2

请输入 n：31

请输入 b：3

输出：2

例 7

四 数 乘 积

1. 问题描述

给定一个长度为 n 的数组 $\boldsymbol{a}$ 和一个正整数 k，从数组中选择四个数，要求四个数的乘积小于等于 k，求方案总数。

2. 问题示例

给定 $n=5$，$\boldsymbol{a}=[1,1,1,2,2]$，$k=3$，返回 2。

3. 代码实现

```
#参数 n 是数组的长度
#参数 a 是已知的数组
#参数 k 是选择的四个数乘积小于等于 k 的乘积值
#返回方案总数
class Solution:
    def numofplan(self, n, a, k):
        sum = [0] * 1000010
        cnt = [0] * 1000010
        for i in range(n):
            if a[i] > k:
                continue
            cnt[a[i]] += 1
        for i in range(n):
            for j in range(i + 1, n):
                if a[i] * a[j] > k:
                    continue
                sum[a[i] * a[j]] += 1
        for i in range(1, k + 1):
            cnt[i] += cnt[i - 1]
            sum[i] += sum[i - 1]
        ans = 0
        for i in range(n):
```

```
            for j in range(i + 1, n):
                res = a[i] * a[j]
                if res > k:
                    continue
                res = k // res
                ans += sum[res]
                if a[i] <= res:
                    ans -= cnt[res // a[i]]
                    if a[i] <= res // a[i]:
                        ans += 1
                if a[j] <= res:
                    ans -= cnt[res // a[j]]
                    if a[j] <= res // a[j]:
                        ans += 1
                if a[i] * a[j] <= res:
                    ans += 1
        return ans // 6
#主函数
if __name__ == '__main__':
    n = 5
    a = [1,1,1,2,2]
    k = 3
    solution = Solution()
    print("方案总数为:", solution.numofplan(n, a, k))
```

4. 运行结果

方案总数为：2

例 8

将整数 *A* 转换为 *B*

1. 问题描述

给定整数 A 和 B，求出将整数 A 转换为 B，需要改变 bit 的位数。

2. 问题示例

把 31 转换为 14，需要改变 2 个 bit 位，即：(31)10＝(11111)2，(14)10＝(01110)2。

3. 代码实现

```
#采用 UTF-8 编码格式
#参数 a、b 是两个整数
#返回一个整数
class Solution:
    def bitSwapRequired(self, a, b):
        c = a ^ b
        cnt = 0
        for i in range(32):
            if c & (1 << i) != 0:
                cnt += 1
        return cnt
if __name__ == '__main__':
    temp = Solution()
    a1 = 4; b1 = 45
    a2 = 10; b2 = 26
    print(("输入:" + str(a1) + " " + str(b1)))
    print(("输出:" + str(temp.bitSwapRequired(a1,b1))))
    print(("输入:" + str(a2) + " " + str(b2)))
    print(("输出:" + str(temp.bitSwapRequired(a2,b2))))
```

4. 运行结果

输入：4　45

输出：3

输入：10　26

输出：1

例 9

罗马数字转换为整数

1. 问题描述

给定一个罗马数字，将其转换为整数，要求返回结果的取值为 1～3999。

2. 问题示例

Ⅳ→4，Ⅻ→12，XXI→21，XCVI→99。

3. 代码实现

```
#采用 UTF-8 编码格式
#参数 s 是一个字符串
#返回一个整数值
class Solution:
    def romanToInt(self, s):
        ROMAN = {
            'I': 1,
            'V': 5,
            'X': 10,
            'L': 50,
            'C': 100,
            'D': 500,
            'M': 1000
        }
        if s == "":
            return 0

        index = len(s) - 2
        sum = ROMAN[s[-1]]
        while index >= 0:
            if ROMAN[s[index]] < ROMAN[s[index + 1]]:
                sum -= ROMAN[s[index]]
            else:
```

```
                sum += ROMAN[s[index]]
            index -= 1
        return sum
if __name__ == '__main__':
    temp = Solution()
    string1 = "DCXXI"
    string2 = "XX"
    print(("输入:" + string1))
    print(("输出:" + str(temp.romanToInt(string1))))
    print(("输入:" + string2))
    print(("输出:" + str(temp.romanToInt(string2))))
```

4. 运行结果

输入：DCXXI
输出：621
输入：XX
输出：20

例 10

整数转换为罗马数字

1. 问题描述

给定一个整数，将其转换为罗马数字，要求返回结果的取值范围为 1～3999。

2. 问题示例

4→Ⅳ，12→Ⅻ，21→ⅩⅪ，99→XCIX。

3. 代码实现

```
#采用 UTF-8 编码格式
#参数 num 是一个整数
#返回值是一个字符串
class Solution:
    def parse(self, digit, index):
        NUMS = {
            1: 'Ⅰ',
            2: 'Ⅱ',
            3: 'Ⅲ',
            4: 'Ⅳ',
            5: 'Ⅴ',
            6: 'Ⅵ',
            7: 'Ⅶ',
            8: 'Ⅷ',
            9: 'Ⅸ',
        }
        ROMAN = {
            'I': ['I', 'X', 'C', 'M'],
            'V': ['V', 'L', 'D', '?'],
            'X': ['X', 'C', 'M', '?']
        }
        s = NUMS[digit]
        return s.replace('X', ROMAN['X'][index]).replace('I', ROMAN['I'][index]).replace('V'
```

```
, ROMAN['V'][index])
    def intToRoman(self, num):
        s = ''
        index = 0
        while num != 0:
            digit = num % 10
            if digit != 0:
                s = self.parse(digit, index) + s
            num = num // 10
            index += 1
        return s
if __name__ == '__main__':
    temp = Solution()
    int1 = 56
    int2 = 99
    print(("输入:" + str(int1)))
    print(("输出:" + str(temp.intToRoman(int1))))
    print(("输入:" + str(int2)))
    print(("输出:" + str(temp.intToRoman(int2))))
```

4. 运行结果

输入：56

输出：LVI

输入：99

输出：XCIX

例 11

整 数 排 序

1. 问题描述

给出一组整数,将其按照升序排列。

2. 问题示例

给出[3，2，1，4，5],排序后的结果为[1，2，3，4，5]。

3. 代码实现

```
#参数 A 是一个整数数组
#返回一个整数数组
class Solution:
    def sortIntegers2(self, A):
        self.quickSort(A, 0, len(A) - 1)
    def quickSort(self, A, start, end):
        if start >= end:
            return
        left, right = start, end
        pivot = A[int((start + end) / 2)]
        while left <= right:
            while left <= right and A[left] < pivot:
                left += 1
            while left <= right and A[right] > pivot:
                right -= 1
            if left <= right:
                A[left], A[right] = A[right], A[left]
                left += 1
                right -= 1
        self.quickSort(A, start, right)
        self.quickSort(A, left, end)
```

```
# 主函数
if __name__ == '__main__':
    A = [3, 2, 1, 4, 5]
    print('初始数组:', A)
    solution = Solution()
    solution.sortIntegers2(A)
    print('快速排序:', A)
```

4. 运行结果

初始数组:[3,2,1,4,5]
快速排序:[1,2,3,4,5]

例 12

整 数 替 换

1. 问题描述

给定一个正整数 n,如果 n 为偶数,将 n 替换为 $n/2$; 如果 n 为奇数,将 n 替换为 $n+1$ 或 $n-1$,那么将 n 转换为 1,最少的替换次数为多少?

2. 问题示例

输入 8,输出 3,即 8 → 4 → 2→1; 输入 7,输出 4,即 7→8→4→2→1,或者 7→6→3→2→1。

3. 代码实现

```
#采用 UTF-8 编码格式
#参数 n 是一个正整数
#返回值是最少的替换次数
#直接使用 DFS 算法
#这类似于因式分解
class Solution:
    def integerReplacement(self, n):
        memo = {}
        #if n == 1:
        # return 0
        self.dfs(n, memo)
        print(memo[n])
        return len(memo[n]) - 1
    def dfs(self, n, memo):
        temp = []
        if n in memo:
            return memo[n]
        if n == 1:
            temp.append(1)
            memo[1] = temp
            return temp
        if n % 2 == 0:
```

```
            temp.append(n)
            cur = self.dfs(n // 2, memo)
            temp.extend(cur)
            memo[n] = temp
            return temp
            # temp.pop()
        else:
            temp2 = temp.copy()
            n2 = n
            temp.append(n)
            cur = self.dfs((n + 1), memo)
            temp.extend(cur)
            temp2.append(n2)
            cur2 = self.dfs((n2 - 1), memo)
            temp2.extend(cur2)
            if len(temp) < len(temp2):
                memo[n] = temp
                return temp
            else:
                memo[n] = temp2
                return temp2
if __name__ == '__main__':
    temp = Solution()
    nums1 = 8
    nums2 = 18
    print(("输入:" + str(nums1)))
    print(("输出:" + str(temp.integerReplacement(nums1))))
    print(("输入:" + str(nums2)))
    print(("输出:" + str(temp.integerReplacement(nums2))))
```

4. 运行结果

输入：8
[8,4,2,1]
输出：3
输入：18
[18,9,8,4,2,1]
输出：5

例 13

两个整数相除

1. 问题描述

要求不使用乘法、除法和 mod 运算符，实现两个整数相除，如果溢出，返回 2147483647。

2. 问题示例

给定被除数 100，除数 9，返回 11。

3. 代码实现

```
#采用 UTF-8 编码格式
class Solution(object):
    def divide(self, dividend, divisor):
        INT_MAX = 2147483647
        if divisor == 0:
            return INT_MAX
        neg = dividend > 0 and divisor < 0 or dividend < 0 and divisor > 0
        a, b = abs(dividend), abs(divisor)
        ans, shift = 0, 31
        while shift >= 0:
            if a >= b << shift:
                a -= b << shift
                ans += 1 << shift
            shift -= 1
        if neg:
            ans = - ans
        if ans > INT_MAX:
            return INT_MAX
        return ans
if __name__ == '__main__':
```

```
temp = Solution()
x1 = 100
x2 = 10
print(("输入:" + str(x1) + " " + str(x2)))
print(("输出:" + str(temp.divide(x1,x2))))
```

4. 运行结果

输入：100　10
输出：10

例 14

整 数 加 法

1. 问题描述

给定两个整数 a 和 b，求它们的和。

2. 问题示例

输入 $a=1$，$b=2$，输出 3；输入 $a=-1$，$b=1$，输出 0。

3. 代码实现

```
#采用 UTF-8 编码格式
#参数 a 是一个整数
#参数 b 是一个整数
#返回值是 a 和 b 的和
class Solution:
    def aplusb(self, a, b):
        while b!= 0:
            a,b = (a^b)&0xffffffff,(a&b)<<1
        return a
if __name__ == '__main__':
    temp = Solution()
    nums1 = 8
    nums2 = 18
    print(("输入:" + str(nums1) + " " + str(nums2)))
    print(("输出:" + str(temp.aplusb(nums1,nums2))))
```

4. 运行结果

输入：8　18

输出：26

例 15

合 并 数 字

1. 问题描述

给出 n 个数，将这 n 个数合并成一个数，每次只能选择两个数 a、b 合并，合并需要消耗的能量为 $a+b$，输出将 n 个数合并成一个数后消耗的最小能量。

2. 问题示例

给出[1,2,3,4]，返回 19，即选择 1、2 合并，消耗 3 能量；现在为[3,4,3]，选择 3、3 合并，消耗 6；现在为[6,4]，剩下两个数合并，消耗 10，一共消耗 19。给出[2,8,4,1]，返回 25，即选择 1、2 合并，消耗 3 能量；现在为[8,4,3]，选择 3、4 合并，消耗 7，现在为[7,8]，剩下两个数合并，消耗 15，一共消耗 25 能量。

3. 代码实现

```
#采用 UTF-8 编码格式
#参数 numbers 代表了数字数量
#返回值是最小的能量消耗
import heapq
class Solution:
    def mergeNumber(self, numbers):
        Q = []
        ans = 0
        for i in numbers:
            heapq.heappush(Q, i)
        while(len(Q) > 1):
            a = heapq.heappop(Q)
            b = heapq.heappop(Q)
            ans = ans + a + b
            heapq.heappush(Q, a + b)
        return ans
if __name__ == '__main__':
    temp = Solution()
```

```
List1 = [1,2,3,4,5]
List2 = [6,7,8,9,10]
print(("输入:" + str(List1)))
print(("输出:" + str(temp.mergeNumber(List1))))
print(("输入:" + str(List2)))
print(("输出:" + str(temp.mergeNumber(List2))))
```

4. 运行结果

输入：[1,2,3,4,5]
输出：33
输入：[6,7,8,9,10]
输出：93

例 16

数 字 判 断

1. 问题描述

给定一个字符串,验证其是否为数字。

2. 问题示例

"0"判断为 True," 0.1 "判断为 True,"abc"判断为 False,"1 a"判断为 False,"2e10"判断为 True。

3. 代码实现

```
#采用 UTF-8 编码格式
#参数 s 是一个字符串
#返回一个布尔值
#有限的自动化
class Solution:
    def isNumber(self, s):
        INVALID = 0; SPACE = 1; SIGN = 2; DIGIT = 3; DOT = 4; EXPONENT = 5;
        #0 是无效的,1 空格,2 符号,3 数字,4 小数点,5 指数,6 输入的数字
        transitionTable = [[-1, 0, 3, 1, 2, -1],     #状态 0 代表没有输入或者是空格
                           [-1, 8, -1, 1, 4, 5],      #状态 1 输入是数字
                           [-1, -1, -1, 4, -1, -1],   #状态 2 代表前面没有数字只有小数点
                           [-1, -1, -1, 1, 2, -1],    #状态 3 代表符号
                           [-1, 8, -1, 4, -1, 5],     #状态 4 代表数字其前方有小数点
                           [-1, -1, 6, 7, -1, -1],    #状态 5 代表输入是'e'或者'E'
                           [-1, -1, -1, 7, -1, -1],   #状态 6 代表在符号之后输入'e'
                           [-1, 8, -1, 7, -1, -1],    #状态 7 代表在数字之后输入'e'
                           [-1, 8, -1, -1, -1, -1]]   #状态 8 代表在输入有限输入后输入空格
        state = 0; i = 0
        while i < len(s):
            inputtype = INVALID
            if s[i] == ' ': inputtype = SPACE
            elif s[i] == '-' or s[i] == '+': inputtype = SIGN
```

```
            elif s[i] in '0123456789': inputtype = DIGIT
            elif s[i] == '.': inputtype = DOT
            elif s[i] == 'e' or s[i] == 'E': inputtype = EXPONENT
            state = transitionTable[state][inputtype]
            if state == -1: return False
            else: i += 1
        return state == 1 or state == 4 or state == 7 or state == 8
if __name__ == '__main__':
    temp = Solution()
    string1 = "1"
    string2 = "23"
    print(("输入:" + string1))
    print(("输出:" + str(temp.isNumber(string1))))
    print(("输入:" + string2))
    print(("输出:" + str(temp.isNumber(string2))))
```

4. 运行结果

输入：1
输出：True
输入：23
输出：True

例 17

下一个稀疏数

1. 问题描述

如果一个数是稀疏数，则它的二进制表示中没有相邻的 1，例如 5(二进制表示为 101)是稀疏数，但是 6(二进制表示为 110)不是稀疏数，本例将给出一个 n，找出大于或等于 n 的最小稀疏数。

2. 问题示例

给出 $n=6$，返回 8，即下一个稀疏数是 8；给出 $n=4$，返回 4，即下一个稀疏数是 4；给出 $n=38$，返回 40，即下一个稀疏数是 40；给出 $n=44$，返回 64，即下一个稀疏数是 64。

3. 代码实现

```
#采用 UTF-8 编码格式
#参数 x 是一个数字
#返回 x 后面的下一个稀疏数
class Solution:
    def nextSparseNum(self, x):
        b_x = bin(x)[2:]
        pos = self.find_highest_continue_one(b_x)
        while pos != -1:
            if pos == 0:
                b_x = "1" + "0" * len(b_x)
            else:
                b_x = b_x[:pos - 1] + "1" + (len(b_x) - pos) * "0"
            pos = self.find_highest_continue_one(b_x)
        return int(b_x, 2)
    def find_highest_continue_one(self, s):
        n = len(s)
        for i in range(n - 1):
            if s[i] == s[i + 1] == "1":
                return i
```

```
        return -1
if __name__ == '__main__':
    temp = Solution()
    nums1 = 16
    nums2 = 50
    print(("输入:" + str(nums1)))
    print(("输出:" + str(temp.nextSparseNum(nums1))))
    print(("输入:" + str(nums2)))
    print(("输出:" + str(temp.nextSparseNum(nums2))))
```

4. 运行结果

输入：16

输出：16

输入：50

输出：64