Cheating Paper

常用输入

1.输入数据：

```
a=int(input())

b=list(map(int,input().split()))

for i in range(#):
c.append(list(map(int,input().split())))

a,b=map(int,input().split())
c=list(input().split())

while 1:
    try:

    except EOFError:
        break
```

2. 矩阵

创建矩阵：

```
result = [[0 for x in range(cols_result)] for x in range(rows_result)]
```

添加保护圈：

一层

```
def edge(A:list,k):
    x=len(A[0]);y=len(A)
    for i in range(y):
        A[i]=[k]+A[i]+[k]
    A.append([k]*(x+2));A.insert(0,[k]*(x+2))
```

二层

```
def edge(A:list,k):
    x=len(A[0]);y=len(A)
```

```
    for i in range(y):
        A[i]=[k]*2+A[i]+[k]*2
    A.append([k]*(x+4));A.insert(0,[k]*(x+4))
```

3.方向：

四向

```
dirtx=[-1,1,0,0]
dirty=[0,0,-1,1]
for i in range(4):
        tx = x + dirtx[i]
        ty = y + dirty[i]
```

八向

```
dirtx=[-1,1,0,0,1,1,-1,-1]
dirty=[0,0,-1,1,1,-1,1,-1]
for i in range(8):
        tx = x + dirtx[i]
        ty = y + dirty[i]
```

3.字符串：

str.find(str, beg=0, end=len(string))——》字符串搜索

格式化字符串：

```
print("{:.2f}".format(3.1415926))
3.1415926  {:.2f} 3.14   保留小数点后两位
3.1415926  {:+.2f}    +3.14  带符号保留小数点后两位
-1 {:-.2f}    -1.00  带符号保留小数点后两位
2.71828    {:.0f} 3  不带小数
5  {:0>2d}    05 数字补零 (填充左边, 宽度为2)
5  {:x<4d}    5xxx   数字补x (填充右边, 宽度为4)
10 {:x<4d}    10xx   数字补x (填充右边, 宽度为4)
1000000    {:,}   1,000,000  以逗号分隔的数字格式
0.25   {:.2%} 25.00% 百分比格式
1000000000 {:.2e} 1.00e+09   指数记法
13 {:>10d}             13 右对齐 (默认, 宽度为10)
13 {:<10d}    13 左对齐 (宽度为10)
13 {:^10d}         13 中间对齐 (宽度为10)
```

```
%2d是将数字按宽度为2，采用右对齐方式输出，若数据位数不到2位，则左边补空格。
```

%02d，和%2d差不多，只不过左边补0
%-2d将数字按宽度为2，采用左对齐方式输出，若数据位数不到2位，则右边补空格
%.2d 输出整形时最少输出2位，如不够前面以0占位。如输出2时变成02，200时只输出200；输出浮点型时（%.2f）小数点后强制2位输出
比如：

```
num = 1
print("%d" % (num))结果为:    （1）
print("%2d" % (num))结果为:   （ 1）
print("%02d" % (num))结果为:  （01）
print("%-2d" % (num))结果为:  （1 ）
print("%.2d" % (num))结果为:  （01）
print("%.2d" % (200))结果为:  （200）
```

字符转化成ascll码：ord()
ascll码转化成字符使用chr()

4.列表：

```
sorted(list) 生成新列表的排序
list.reverse()  反向列表
list.remove(obj) 移除第一个匹配值
list.pop() 默认index为-1
c1.sort(key=lambda x:x[1],reverse=True) 以第二个字符为标准进行排序
c1.sort(key=lambda x:(x[0],-x[1]),reverse=True) 以第一个字符正序，第二个字符逆序进行排序
del list[index] 删除列表中的值
```

5.字典

```
b=dict()
① 当不确定一个值是否在b中有key：
c2=b.get(a,0)
c2+=c1
b[a]=c2

② 解决用列表当键值时的unhashable：使用元组作为键值
b[(x1,y1)]=x
```

6.输出：

print(“”.join(map(str,alist))) 数字列表转化为字符串输出

类

eg：Fraction类

```
from typing import List


def gcd(m, n):
    while m % n != 0:
        m, n = n, m%n
    return n


class Fraction:
    def __init__(self, top, bottom): #定义函数
        self.num = top
        self.den = bottom

    def __str__(self):
        return f"{self.num}/{self.den}"

    def __add__(self, other_fraction):
        new_num = self.num * other_fraction.den + self.den *
other_fraction.num
        new_den = self.den * other_fraction.den
        common_divisor = gcd(new_num, new_den)
        return Fraction(new_num//common_divisor, new_den//common_divisor)


input1 = list(map(int, input().split()))

Fraction1 = Fraction(input1[0], input1[1])
Fraction2 = Fraction(input1[2], input1[3])

print(Fraction1 + Fraction2)
```

eg：矩阵运算

```
class MyMatrix:
    def __init__(self, data):
        self.data = data
        self.rows = len(data)
        self.cols = len(data[0])

    def __matmul__(self, other):
        if len(self.data[0]) != len(other.data):
            raise ValueError('Error!')

        result = [[0 for x in range(len(other.data[0]))] for x in
range(len(self.data))]
        for i in range(self.rows):
            for j in range(self.cols):
```

```
                for k in range(other.cols):
                    result[i][k] += self.data[i][j] * other.data[j][k]
        return MyMatrix(result)

    def __add__(self, other):
        if self.rows != other.rows or self.cols != other.cols:
            raise ValueError('Error!')
        result = []
        for i in range(self.rows):
            row = [self.data[i][j] + other.data[i][j] for j in
range(self.cols)]
            result.append(row)
        return MyMatrix(result)


    def __str__(self):
        return f'{self.data}'

    def __getitem__(self, index):
        return self.data[index]


class Solution:
    def matrix_cal(self, matrix):
        try:
            result = MyMatrix(matrix[0]) @ MyMatrix(matrix[1])
            result = result + MyMatrix(matrix[2])
            return [result, row[0]]
        except ValueError:
            pass
            return 'Error!'


if __name__ == '__main__':
    solution = Solution()
    matrix = [[], [], []]
    row = []
    col = []
    for i in range(3):
        temp = [int(x) for x in input().split()]
        row.append(temp[0])
        col.append(temp[1])
        for j in range(temp[0]):
            matrix[i].append([int(x) for x in input().split()])
    ans = solution.matrix_cal(matrix)
    if ans == 'Error!':
        print(ans)
    else:
```

```
        for i in range(ans[1]):
            print(' '.join([str(x) for x in ans[0][i]]))
```

二分

eg：月度开销

```
class Solution:
    def minicost(self, day, month, cost_list) :
        left = max(cost_list)  #使用二分查找获得合适的最大的cost
        right = sum(cost_list)
        while left <= right:
            mid = (right + left)//2
            month_needed = 1
            cost_per_month = 0
            ans = 0
            for j in range(day):
                if cost_per_month + cost_list[j] <= mid:
                    cost_per_month += cost_list[j]
                else:
                    cost_per_month = cost_list[j]
                    month_needed += 1
                ans = max(cost_per_month, ans)
            if month_needed < month:
                right = mid - 1
            elif month_needed > month:
                left = mid + 1
            else:
                return ans



if __name__ == "__main__":
    solution = Solution()

    day1, month1 = list(map(int, input().split()))
    cost_list1 = []
    for i in range(day1):
        cost_list1.append(int(input()))
    result = solution.minicost(day1, month1, cost_list1)
    print(result)
```

字典：

eg：直播计票

```python
class solution:
    def vote_counting(self, vote):
        vote_list = list(map(int, vote.split()))
        vote_count = dict()
        max_count = 0
        for i in range(len(vote_list)):
            if vote_list[i] not in vote_count.keys():
                vote_count[vote_list[i]] = 1
            else:
                vote_count[vote_list[i]] += 1
        for key in vote_count.keys():
            if vote_count[key] > max_count:
                max_count_index = [key]
                max_count = vote_count[key]
            elif vote_count[key] == max_count:
                max_count_index.append(key)
        return ' '.join([str(x) for x in sorted(max_count_index)])


if __name__ == '__main__':
    vote1 = input()
    Solution = solution()
    print(Solution.vote_counting(vote1))
```

栈

后进：append
先出：pop

eg：逆波兰表达式：

```python
from math import trunc


class Solution:
    def evalRPN(self, tokens) -> int:
        stack = []
        sign = ['+','-','*','/']
        for i in tokens:
            if i in sign:
                b,a = stack.pop(),stack.pop()
                cal_result = trunc(eval(''.join([a,i,b])))
                stack.append(str(cal_result))

            else:
                stack.append(i)
        return int(stack[0])
```

字符串解码：

```python
class Solution:
    def decodeString(self, s: str) -> str:
        stack, res, multi = [], "", 0
        for c in s:
            if c == '[':
                stack.append([multi, res])
                res, multi = "", 0
            elif c == ']':
                cur_multi, last_res = stack.pop()
                res = last_res + cur_multi * res
            elif '0' <= c <= '9':
                multi = multi * 10 + int(c)
            else:
                res += c
        return res
```

作者：Krahets
链接：https://leetcode.cn/problems/decode-string/solutions/19447/decode-string-fu-zhu-zhan-fa-di-gui-fa-by-jyd/
来源：力扣（LeetCode）

双指针：

eg：移动零

```python
class Solution:
    # 双指针
    def moveZeroes(self, nums: list[int]) -> None:
        i = 0
        j = 0
        while i < len(nums):
            if nums[i] == 0:
                while j < len(nums):
                    # 将第二个指针j向右移动至非0处
                    if nums[j] == 0:
                        j += 1
                    elif j < i:
                        j += 1
                    else:
                        nums[i], nums[j] = nums[j], nums[i]
                        break
            i += 1
```

递归：

调用自身。需要注意：

1. 结束条件。在完成的时候需要判断及时结束递归；
2. 在递归结束后，记得复原原本的列表。

eg：全排列

```
class Solution:
    def permute(self, nums: list[int]) -> list[list[int]]:
        n = len(nums)
        permuted_array = []
        def backtrack(index):
            if index == n:
                permuted_array.append(nums[:])
            else:
                for i in range(index, n):
                    nums[index], nums[i] = nums[i], nums[index]
                    backtrack(index + 1)
                    nums[index], nums[i] = nums[i], nums[index]
        backtrack(index=0)
        return permuted_array
```

eg：子集

```
class Solution:
    def subsets(self, nums: list[int]) -> list[list[int]]:
        n = len(nums)
        subset_array = [[]]
        def backtrack(index, current):
            if index != n:
                for i in range(index, n):
                    current.append(nums[i])
                    subset_array.append(current[:])
                    backtrack(i + 1, current)
                    current.pop()
        backtrack(index=0, current=[])
        return subset_array
```

eg：八皇后

```
import string


def backtrack(index_x, index_y, chessboard, count, row_x, row_y, path):
    # 标记骑士走过的位置：
    chessboard[index_x][index_y] = 1
```

```
    # count达标时，将坐标记录至path，结束递归。
    if count == row_x * row_y - 1:
        # 注意要复原棋盘
        chessboard[index_x][index_y] = 0
        path.append([index_y, index_x])
        return path

    # count未达标，则在八个方向上遍历并递归。如果某次递归返回的path不为[]，说明骑士已经完全走遍棋盘。
    # 此时，可以将当前位置append到棋盘内，结束本轮递归
    else:
        for i in range(8):
            temp_index_x = index_x + direct[i][0]
            temp_index_y = index_y + direct[i][1]
            if chessboard[temp_index_x][temp_index_y] == 0:
                backtrack(temp_index_x, temp_index_y, chessboard, count+1, row_x, row_y, path)
                if path:
                    path.append([index_y, index_x])
                    chessboard[index_x][index_y] = 0
                    return path
        chessboard[index_x][index_y] = 0


class Solution:
    def subsets(self, nums):
        all_result = []
        for i in range(nums):
            # 初始化chessboard。往上下、左右额外添加两行/列避免out of index
            chessboard_rows, chessboard_cols = [int(x) for x in input().split()]
            chessboard = [[1 for _ in range(chessboard_cols+4)] for _ in range(chessboard_rows+4)]
            for i in range(2, chessboard_rows+2):
                for j in range(2, chessboard_cols+2):
                    chessboard[i][j] = 0
            # 遍历矩阵作为骑士的初始位置，并开始递归
            for j in range(2, chessboard_cols + 2):
                for i in range(2, chessboard_rows+2):
                    result = backtrack(i, j, chessboard, count=0, row_x=chessboard_rows, row_y=chessboard_cols, path=[])
                    # 这一连串break丑丑的（
                    if result:
                        all_result.append(result)
                        break
                if result:
                    break
            if result:
                continue
            else:
```

```
                all_result.append([])
        return all_result


if __name__ == '__main__':
    solution = Solution()

    # 初始化骑士走的方向。注意骑士的走向需要按字典序排列以满足题目的要求。
    direct = [[-1, -2], [1, -2], [-2, -1], [2, -1], [-2, 1], [2, 1], [-1,
2], [1, 2]]

    # 输入参数、输出答案
    nums = int(input())
    ans = solution.subsets(nums)
    letter = list(string.ascii_uppercase)
    num_in_ans = 0
    for i in ans:
        i.reverse()
        path_str = ''
        if i:
            for j in i:
                path_str += ''.join([letter[j[0]-2], str(j[1]-1)])
        else:
            path_str += 'impossible'
        num_in_ans += 1
        print(f'Scenario #{num_in_ans}:')
        print(path_str)
        if num_in_ans != nums:
            print()
```

链表

回文链表（在本地创建链表并测试）

```
 # Definition for singly-linked list.
class ListNode:
     def __init__(self, val=0, next=None):
        self.val = val
        self.next = next


def create_linked_list(values):
    """从列表构建链表"""
    if not values:
        return None

    head = ListNode(values[0])
    current = head
```

```
    for val in values[1:]:
        current.next = ListNode(val)
        current = current.next

    return head

def linked_list_to_list(head):
    """将链表转换为列表（用于验证）"""
    result = []
    current = head
    while current:
        result.append(current.val)
        current = current.next
    return result

class Solution:
    def isPalindrome(self, head) -> bool:
        chainlist = []
        q = head
        while q:
            chainlist.append(q.val)
            q = q.next
        print(chainlist)
        if chainlist == list(reversed(chainlist)):
            return True
        else:
            return False


def run_tests():
    test_cases = [
        ([], True),  # 空链表
        ([1], True),  # 单节点
        ([1, 2], False),  # 非回文
        ([1, 1], True),  # 回文
        ([1, 2, 1], True),  # 回文
        ([1, 2, 3], False),  # 非回文
        ([1, 2, 2, 1], True),  # 回文
    ]

    for i, (values, expected) in enumerate(test_cases):
        head = create_linked_list(values)
        result = solution.isPalindrome(head)
        status = "√" if result == expected else "X"
        print(f"Test {i + 1}: {status} Input: {values} -> Expected:
{expected}, Got: {result}")


# 运行测试
```

```python
if __name__ == "__main__":
    solution = Solution()
    run_tests()
```