Assignment 1:

Fraction 类

```
import math

class Fraction:
    def __init__(self, numerator, denominator):
        if denominator == 0:
            raise ValueError("Denominator cannot be zero")
        # 保证分母为正
        if denominator < 0:
            numerator = -numerator
            denominator = -denominator

        gcd = math.gcd(numerator, denominator)
        self.numerator = numerator // gcd
        self.denominator = denominator // gcd

    def __add__(self, other):
        new_numerator = (
            self.numerator * other.denominator +
            other.numerator * self.denominator
        )
        new_denominator = self.denominator * other.denominator
        return Fraction(new_numerator, new_denominator)

    def __str__(self):
        return f"{self.numerator}/{self.denominator}"


# 读取输入
a, b, c, d = map(int, input().split())

f1 = Fraction(a, b)
f2 = Fraction(c, d)
```

袋子里最少数目的球

```
class Solution:
    def minimumSize(self, nums, maxOperations):
        left, right = 1, max(nums)
        ans = right

        while left <= right:
            mid = (left + right) // 2
            operation = 0
            for num in nums:
                # integer-safe and correct count of splits needed
                operation += (num - 1) // mid

            if operation <= maxOperations:
                ans = mid
                right = mid - 1
            else:
                left = mid + 1

        return ans
```

月度开销

```
def can_split(costs, M, max_month):
    months = 1
    current_sum = 0

    for cost in costs:
        if current_sum + cost <= max_month:
            current_sum += cost
        else:
            months += 1
            current_sum = cost
            if months > M:
                return False
    return True


def main():
    N, M = map(int, input().split())
    costs = [int(input()) for _ in range(N)]

    left = max(costs)          # 至少能装下最大的一天
    right = sum(costs)         # 最多把所有天放一个月
    answer = right

    while left <= right:
        mid = (left + right) // 2
        if can_split(costs, M, mid):
            answer = mid
            right = mid - 1
        else:
            left = mid + 1

    print(answer)
```

note：二分查找

模型整理

```
n = int(input())

models = {}

for _ in range(n):
    line = input().strip()
    name, param = line.split('-')

    # 解析参数量
    if param.endswith('M'):
        value = float(param[:-1])
        value_in_m = value
    else:  # endswith 'B'
        value = float(param[:-1])
        value_in_m = value * 1000

    if name not in models:
        models[name] = []

    models[name].append((value_in_m, param))

# 按模型名排序
for name in sorted(models.keys()):
    # 按参数量排序
    models[name].sort(key=lambda x: x[0])
    params = [p for _, p in models[name]]
    print(f"{name}: {', '.join(params)}")
```

解题思路

1 解析输入

每一行格式：

模型名-参数量

例如：

GPT-1.3B Bert-340M

- 模型名：- 左边
- 参数量：- 右边

2 参数量转成可比较数值

统一转成 百万（M） 作为比较单位：

- xM → x
- yB → y * 1000

注意：只用于排序，输出仍保持原格式

3 数据结构

用字典：

{ "GPT": [(350, "350M"), (1300, "1.3B"), (175000, "175B")], "Bert": [(110, "110M"), (340, "340M")] }

4 排序规则

- 模型名：字典序
- 同一模型下：按数值从小到大

Assigment 2

泰波拿契數

http://cs101.openjudge.cn/practice/20742/

```
n = int(input())

if n == 0:
    print(0)
elif n == 1 or n == 2:
    print(1)
else:
    T = [0] * (n + 1)
    T[0] = 0
    T[1] = 1
    T[2] = 1

    for i in range(3, n + 1):
        T[i] = T[i - 1] + T[i - 2] + T[i - 3]

    print(T[n])
```

一定要确定 range, edge case

Chat Room

http://codeforces.com/problemset/problem/58/A

```
s = input().strip()
target = "hello"

idx = 0
for ch in s:
    if ch == target[idx]:
        idx += 1
        if idx == len(target):
            break

print("YES" if idx == len(target) else "NO")
```

strings cannot be modified

for strings, you can also split into characters for python without explicitly saying so

- for ch in s

strip(): strips of whitespaces

use counters for checking if matching target

an variable can be named as a string, and you can compare characters within the string

String Task

https://codeforces.com/problemset/problem/118/A

```
s = input().strip()
vowels = set("aoyeui")

result = []

for ch in s:
    if ch.lower() not in vowels:
        result.append("." + ch.lower())

print("".join(result))
```

.lower(), .upper()

add to a string: use append

Goldbach Conjecture

http://cs101.openjudge.cn/practice/22359/

```
def is_prime(x):
    if x < 2:
        return False
    for i in range(2, int(x ** 0.5) + 1):
        if x % i == 0:
            return False
    return True


S = int(input())

for A in range(2, S):
    B = S - A
    if is_prime(A) and is_prime(B):
        print(A, B)
        break
```

** is sqrt

% is remainder

多项式时间复杂度

http://cs101.openjudge.cn/pctbook/E23563/

```
s = input().strip()
terms = s.split('+') #split at specific terms

max_power = 0

for term in terms: #every character
    # 处理系数和指数
    if 'n' not in term:
        # 常数项，相当于 n^0
        coef = int(term)
        power = 0
    else:
        # 拆分系数
        if term.startswith('n'): #startswith()
            coef = 1
            rest = term
        else:
            coef_part, rest = term.split('n', 1)
            coef = int(coef_part)

        # 拆分指数
        if '^' in rest:
            power = int(rest.split('^')[1])
        else:
            power = 1  # 兜底处理 n 的情况（虽然题目一般不会给）

    if coef != 0:
        max_power = max(max_power, power)

print(f"n^{max_power}")
```

直播计票

http://cs101.openjudge.cn/practice/24684/

```
votes = list(map(int, input().split()))

count = {}
for v in votes:
    count[v] = count.get(v, 0) + 1

max_votes = max(count.values())

result = [k for k, v in count.items() if v == max_votes]
result.sort()

print(" ".join(map(str, result)))
```

Assignment 3

Bigram 分词

https://leetcode.cn/problems/occurrences-after-bigram/

```
class Solution(object):
    def findOcurrences(self, text, first, second):
        words = text.split(" ")
        result = []
        for i in range(len(words) - 2):
            if words[i] == first and words[i+1] == second:
                result.append(words[i+2]) #append: add to list
        return result

```

移动零

https://leetcode.cn/problems/move-zeroes/

```
class Solution:
    def moveZeroes(self, nums):
        non_zero_pos = 0
        for i in range(len(nums)):
            if nums[i] != 0:
                nums[non_zero_pos] = nums[i]
                nums[i] = nums[non_zero_pos]
                non_zero_pos += 1
        for i in range((non_zero_pos, len(nums))):
            nums[i] = 0

        return nums
```

use a pointer to indicate which position you want to keep track

有效的括号

https://leetcode.cn/problems/valid-parentheses/description/

```
class Solution(object):
    def isValid(self, s):
        stack = []
        pairs = {')': '(', '}': '{', ']': '['} #mapping whihc ones pair together
        for char in s:
            if char in '({[':
                stack.append(char)
            else:
                if not stack or stack[-1] != pairs[char]:
                    return False
                stack.pop()
        return not stack
```

mapping pairs is cruical for this question

stack operations

- stack[]

- stack append()
- stack.pop()

not stack = if it is empty

杨辉三角

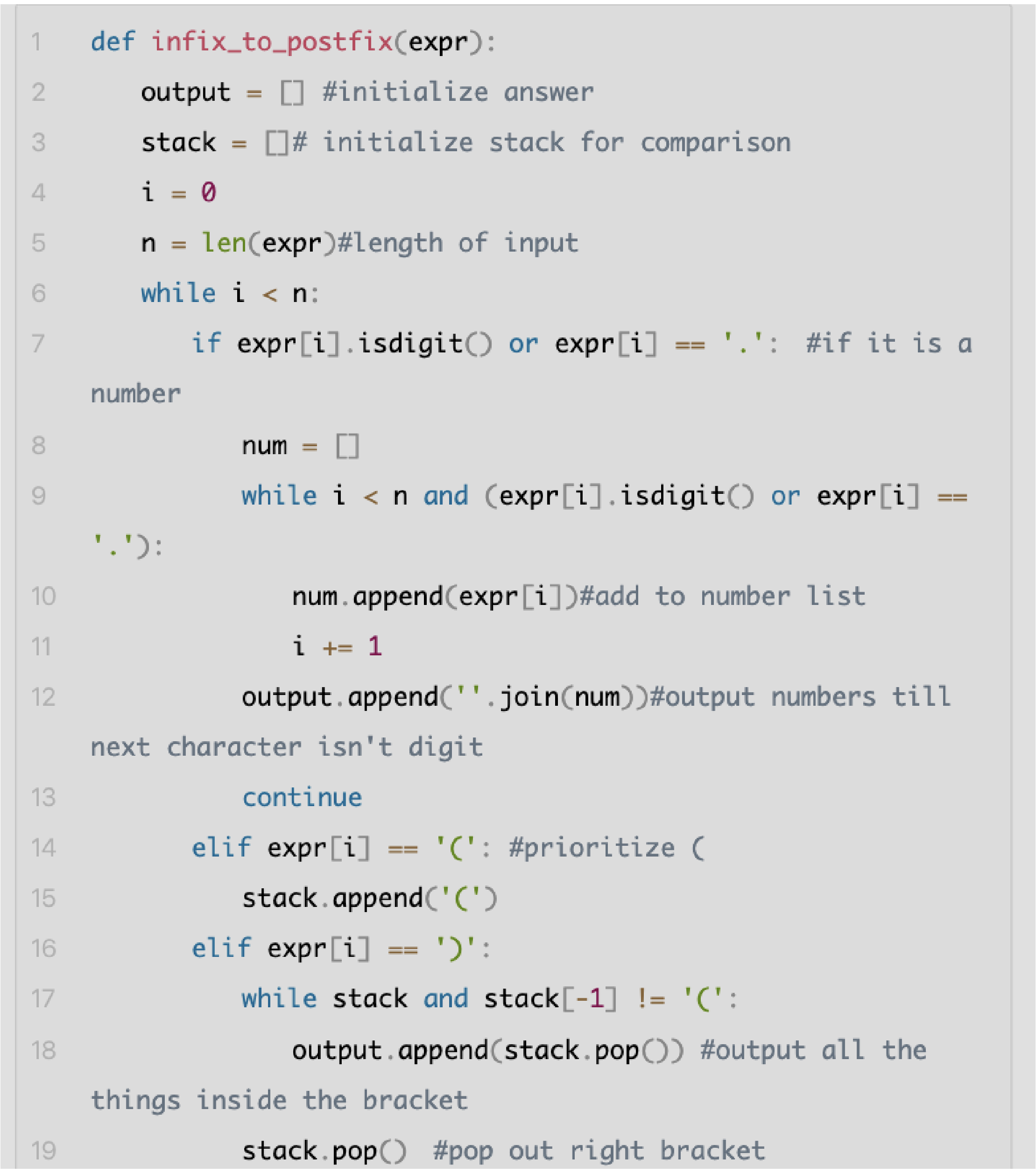
https://leetcode.cn/problems/pascals-triangle/description/

```
class Solution:
    def generate(self, numRows: int):
        triangle = []

        for i in range(numRows):
            row = [1] * (i + 1)  # 初始化每行
            for j in range(1, i):
                row[j] = triangle[i-1][j-1] + triangle[i-1][j]
            triangle.append(row)

        return triangle
```

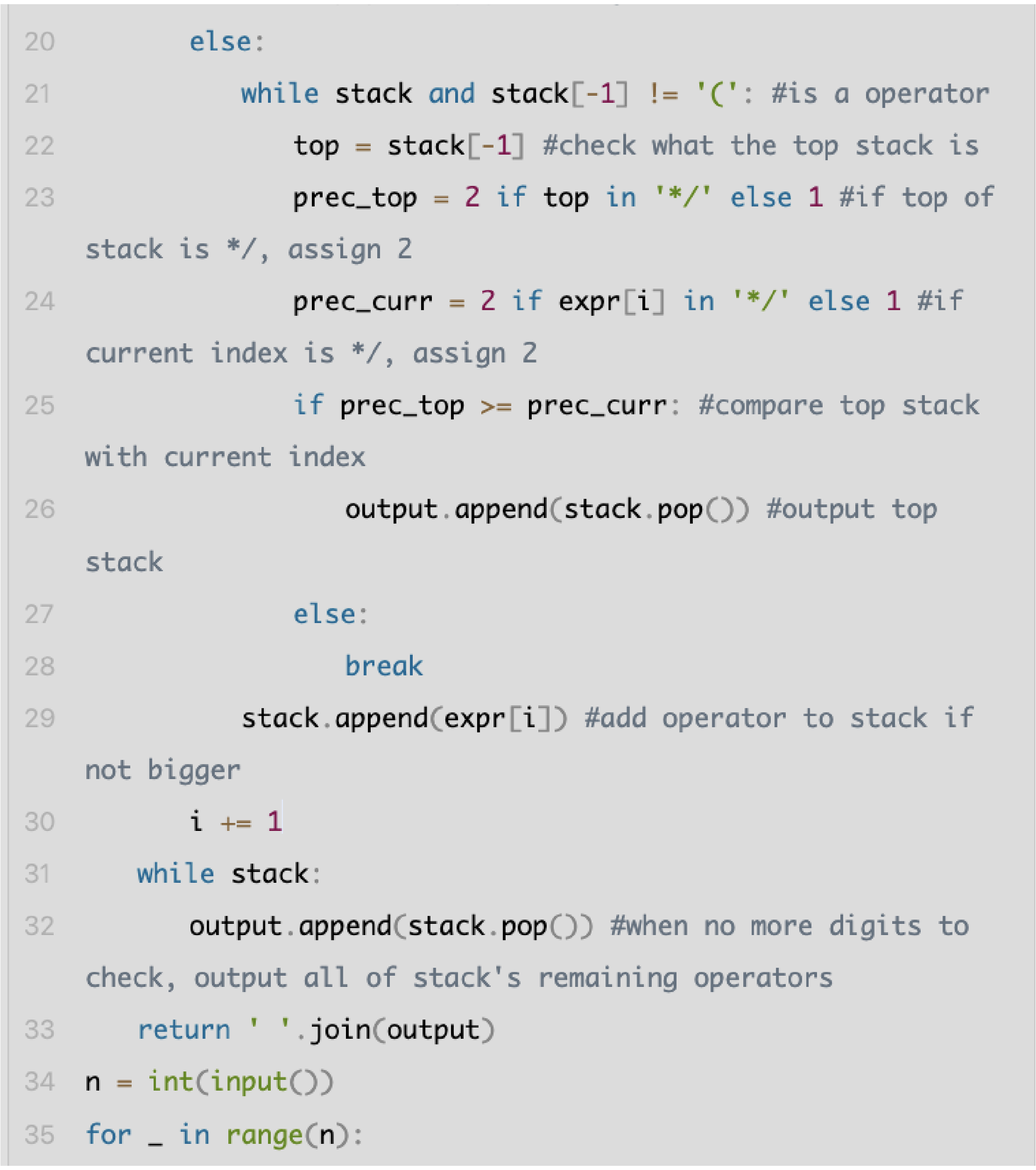
全排列

https://leetcode.cn/problems/permutations/description/

```
from typing import List

class Solution:
    def permute(self, nums: List[int]) -> List[List[int]]:
        res = []
        path = []
        used = [False] * len(nums)

        def backtrack():
            if len(path) == len(nums):
                res.append(path[:])
                return

            for i in range(len(nums)):
                if used[i]:
                    continue

                used[i] = True
                path.append(nums[i])
                backtrack()
                path.pop()
                used[i] = False

        backtrack()
        return res

class Solution:
    def permute(self, nums):
        result = []   # to store all permutations

        def backtrack(path, remaining):
            # Base case: no remaining numbers → a full permutation is formed
```



```python
def infix_to_postfix(expr):
    output = [] #initialize answer
    stack = []# initialize stack for comparison
    i = 0
    n = len(expr)#length of input
    while i < n:
        if expr[i].isdigit() or expr[i] == '.':  #if it is a number
            num = []
            while i < n and (expr[i].isdigit() or expr[i] == '.'):
                num.append(expr[i])#add to number list
                i += 1
            output.append(''.join(num))#output numbers till next character isn't digit
            continue
        elif expr[i] == '(': #prioritize (
            stack.append('(')
        elif expr[i] == ')':
            while stack and stack[-1] != '(':
                output.append(stack.pop()) #output all the things inside the bracket
            stack.pop()  #pop out right bracket
```



```python
        else:
            while stack and stack[-1] != '(': #is a operator
                top = stack[-1] #check what the top stack is
                prec_top = 2 if top in '*/' else 1 #if top of stack is */, assign 2
                prec_curr = 2 if expr[i] in '*/' else 1 #if current index is */, assign 2
                if prec_top >= prec_curr: #compare top stack with current index
                    output.append(stack.pop()) #output top stack
                else:
                    break
            stack.append(expr[i]) #add operator to stack if not bigger
        i += 1
    while stack:
        output.append(stack.pop()) #when no more digits to check, output all of stack's remaining operators
    return ' '.join(output)
n = int(input())
for _ in range(n):
```

子集

https://leetcode.cn/problems/subsets/description/

```
from typing import List

class Solution:
    def subsets(self, nums: List[int]) -> List[List[int]]:
        res = []
        path = []

        def backtrack(start):
            # 当前 path 本身就是一个子集
            res.append(path[:])

            for i in range(start, len(nums)):
                path.append(nums[i])      # 选择 nums[i]
                backtrack(i + 1)          # 递归处理后面的元素
                path.pop()                # 撤销选择

        backtrack(0)
        return res

```

compiler error

wrong answer

correct

Assignment 4

矩阵运算

🔗 http://cs101.openjudge.cn/pctbook/E18161/

```python
def read_matrix():
    r, c = map(int, input().split())
    mat = []
    for _ in range(r):
        mat.append(list(map(int, input().split())))
    return r, c, mat


# 读取三个矩阵
ar, ac, A = read_matrix()
br, bc, B = read_matrix()
cr, cc, C = read_matrix()

# 检查乘法是否合法
if ac != br:
    print("Error!")
    exit()

# 乘法结果维度
mr, mc = ar, bc

# 检查加法是否合法
if mr != cr or mc != cc:
    print("Error!")
    exit()

# 计算 A · B
D = [[0] * mc for _ in range(mr)]
for i in range(mr):
    for j in range(mc):
        for k in range(ac):
            D[i][j] += A[i][k] * B[k][j]
```

the importnat thing is to **compare column and rows, assign correctly.**

draw out the matrix, understand who is mulitplying who

二维矩阵上的卷积运算

http://cs101.openjudge.cn/pctbook/E19942/

```
# 读取输入
m, n, p, q = map(int, input().split())

matrix = [list(map(int, input().split())) for _ in range(m)]
kernel = [list(map(int, input().split())) for _ in range(p)]

# 输出矩阵大小
out_rows = m - p + 1
out_cols = n - q + 1

# 卷积计算
result = [[0] * out_cols for _ in range(out_rows)]

for i in range(out_rows):
    for j in range(out_cols):
        s = 0
        for x in range(p):
            for y in range(q):
                s += matrix[i + x][j + y] * kernel[x][y]
        result[i][j] = s

# 输出结果
for row in result:
    print(" ".join(map(str, row)))
```

倒排索引

http://cs101.openjudge.cn/pctbook/M06640/

```
# Read number of documents
N = int(input())

index = {}   # inverted index: word -> list of doc IDs

for doc_id in range(1, N + 1):
    parts = input().split()
    c = int(parts[0])          # number of words
    words = parts[1:]

    seen = set()   # avoid duplicate words in the same document

    for word in words:
        if word in seen:
            continue
        seen.add(word)

        if word not in index:
            index[word] = []
        index[word].append(doc_id)

# Read number of queries
M = int(input())

for _ in range(M):
    query = input().strip()

    if query in index:
        print(" ".join(map(str, index[query])))
    else:
        print("NOT FOUND")
```

use a dictionary to match values to items use continue to skiip the loop once for this iteration

相交链表

🔗 https://leetcode.cn/problems/intersection-of-two-linked-lists/description/

```
class Solution:
    def getIntersectionNode(self, headA, headB):
        # Step 1: Get lengths of both lists
        def get_length(head):
            length = 0
            while head:
                length += 1
                head = head.next
            return length

        lenA = get_length(headA)
        lenwB = get_length(headB)

        # Step 2: Align both lists by skipping the difference in lengths
        while lenA > lenB:
            headA = headA.next
            lenA -= 1
        while lenB > lenA:
            headB = headB.next
            lenB -= 1

        # Step 3: Traverse both lists and find the intersection
        while headA and headB:
            if headA == headB:
                return headA  # Intersection found
            headA = headA.next
            headB = headB.next

        return None  # No intersection
```

反转链表

https://leetcode.cn/problems/reverse-linked-list/description/

```
class Solution:
    def reverseList(self, head):
        prev = None
        curr = head
        while curr:
            nxt = curr.next
            curr.next = prev
            prev = curr
            curr = nxt
        return prev
```

Knight's journey

http://cs101.openjudge.cn/practice/02488/

```
def solve():
    t = int(input().strip())

    # Knight moves
    moves = [
        (2, 1), (1, 2), (-1, 2), (-2, 1),
        (-2, -1), (-1, -2), (1, -2), (2, -1)
    ]

    for case in range(1, t + 1):
        p, q = map(int, input().split())
        total = p * q

        visited = [[False] * q for _ in range(p)]
        path = []
        found = False

        def dfs(r, c, step):
            nonlocal found
            if found:
                return
            visited[r][c] = True
            path.append((r, c))

            if step == total:
                found = True
                return

            next_moves = []
            for dr, dc in moves:
                nr, nc = r + dr, c + dc
                if 0 <= nr < p and 0 <= nc < q and not visited[nr][nc]:
                    next_moves.append((nr, nc))
```

Mock Exam Assignment 5

咒语序列

http://cs101.openjudge.cn/practice/29952/

```
s = input().strip()

stack = [-1]    # 哨兵
max_len = 0

for i, ch in enumerate(s):
    if ch == '(':
        stack.append(i)
    else:   # ch == ')'
        stack.pop()
        if not stack:
            # 重新设定起点
            stack.append(i)
        else:
            max_len = max(max_len, i - stack[-1])

print(max_len)
```

We iterate through the string using enumerate.

If the character is '(', we push its **index** onto the stack.

If the character is ')', we pop from the stack to try to match it.

- If the stack becomes empty after popping, we push the current index as a new base (this) is invalid).
- Otherwise, we update the maximum length using

i - stack[-1].

radar installation

http://cs101.openjudge.cn/practice/01328/

```
import math

case_num = 1

while True:
    line = input().strip()
    if not line:
        continue
    n, d = map(int, line.split())
    if n == 0 and d == 0:
        break

    intervals = []
    impossible = False

    for _ in range(n):
        x, y = map(int, input().split())
        if y > d:
            impossible = True
        else:
            dx = math.sqrt(d * d - y * y)
            intervals.append((x - dx, x + dx))

    if impossible:
        print(f"Case {case_num}: -1")
        case_num += 1
        continue

    # 按右端点排序
    intervals.sort(key=lambda x: x[1])

    count = 0
    pos = -float('inf')
```

八皇后

http://cs101.openjudge.cn/practice/02754/

```
# 预计算 8 皇后所有解
solutions = []

def dfs(row, cols, diag1, diag2, path):
    if row == 8:
        # 转成字符串
        solutions.append(''.join(str(c + 1) for c in path))
        return
    for col in range(8):
        if col in cols or (row - col) in diag1 or (row + col) in diag2: #check if there are any in the same column, diagonal or other diagonal direction
            continue
        dfs(
            row + 1,
            cols | {col}, #update column
            diag1 | {row - col}, #update diag1
            diag2 | {row + col}, #update diag2
            path + [col] #
        )

dfs(0, set(), set(), set(), [])

# 排序（按整数大小）
solutions.sort()

# 处理输入
n = int(input())
for _ in range(n):
    b = int(input())
    print(solutions[b - 1])
```

USE DFS

- check for conlfict cases, like same column, diagonal
- backtracking
  - trying a choice, continuing if it works, undoinging it if it leads to a dead end

洋葱

🔗 http://cs101.openjudge.cn/practice/25570/

```
n = int(input())
mat = [list(map(int, input().split())) for _ in range(n)]

max_sum = 0
layers = (n + 1) // 2

for k in range(layers):
    layer_sum = 0
    top = k
    bottom = n - 1 - k
    left = k
    right = n - 1 - k

    if top == bottom and left == right:
        # 中心单点
        layer_sum = mat[top][left]
    else:
        # 上边
        for j in range(left, right + 1):
            layer_sum += mat[top][j]
        # 下边
        for j in range(left, right + 1):
            layer_sum += mat[bottom][j]
        # 左边（不含角）
        for i in range(top + 1, bottom):
            layer_sum += mat[i][left]
        # 右边（不含角）
        for i in range(top + 1, bottom):
            layer_sum += mat[i][right]

    max_sum = max(max_sum, layer_sum)

print(max_sum)
```

#NAVIGATE TOP DOWN LEFT RIGHT BY HOLDING ONE OF THEM CONSTANT

find patterns in finding layers

find patterns in finding where each layer is

you just have to write out the indexes and compare it to find a function

逃离紫罗兰监狱

http://cs101.openjudge.cn/practice/29954/

```
from collections import deque

R, C, K = map(int, input().split())
grid = [list(input().strip()) for _ in range(R)]

# 找起点
for i in range(R):
    for j in range(C):
        if grid[i][j] == 'S':
            sr, sc = i, j

# visited[r][c][used_flash]
visited = [[[False] * (K + 1) for _ in range(C)] for _ in range(R)]

queue = deque()
queue.append((sr, sc, 0, 0))  # r, c, used_flash, steps
visited[sr][sc][0] = True

dirs = [(1, 0), (-1, 0), (0, 1), (0, -1)]

while queue:
    r, c, used, steps = queue.popleft()

    if grid[r][c] == 'E':
        print(steps)
        exit(0)

    for dr, dc in dirs:
        nr, nc = r + dr, c + dc
        if 0 <= nr < R and 0 <= nc < C:
            cell = grid[nr][nc]
            if cell == '#':
                if used < K and not visited[nr][nc][used + 1]:
```

当前队列中位数

🔗 http://cs101.openjudge.cn/practice/27256/

```
import heapq
#min-heap, max-heap
from collections import deque
#appendx: add to back
#popleft: remove from the front
from collections import defaultdict

n = int(input())


# FIFO queue to remember insertion order
queue = deque() #create queue

# Heaps for median
small = []     # max-heap (store negative values)
large = []     # min-heap

# Lazy deletion counter
delayed = defaultdict(int)

# Sizes excluding delayed elements
small_size = 0
large_size = 0



def prune(heap):
    """
    Remove elements from heap top that are marked for deletion
    """
    while heap:
        x = -heap[0] if heap is small else heap[0]
        if delayed[x] > 0:
            delayed[x] -= 1
```

Assignment 6：链表，栈，排序

后续表达式求值

http://cs101.openjudge.cn/practice/24588/

```
n = int(input())

for _ in range(n):
    expr = input().strip().split()
    stack = []

    for token in expr:
        if token in {"+", "-", "*", "/"}:
            b = stack.pop()
            a = stack.pop()
            if token == "+":
                stack.append(a + b)
            elif token == "-":
                stack.append(a - b)
            elif token == "*":
                stack.append(a * b)
            elif token == "/":
                stack.append(a / b)
        else:
            # 操作数
            stack.append(float(token))

    result = stack.pop()
    print(f"{result:.2f}")

```

回文链表

https://leetcode.cn/problems/palindrome-linked-list/

有多少种合法的出栈顺序

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def isPalindrome(self, head) -> bool:
        if not head or not head.next:
            return True
        slow = fast = head
        while fast and fast.next:
            slow = slow.next
            fast = fast.next.next

        prev = None
        curr = slow
        while curr:
            next_temp = curr.next
            curr.next = prev
            prev = curr
            curr = next_temp

        left, right = head, prev
        while right:
            if left.val != right.val:
                return False
            left = left.next
            right = right.next
        return True
```

有多少种合法的出栈顺序

🔗 http://cs101.openjudge.cn/practice/27217/

```
n = int(input())

catalan = 1
for i in range(2, n + 1):
    catalan = catalan * (n + i) // i

catalan //= (n + 1)

print(catalan)
```

中序表达式转后续表达式

http://cs101.openjudge.cn/practice/24591/

```
def infix_to_postfix(expr):
    output = [] #initialize answer
    stack = []# initialize stack for comparison

    i = 0
    n = len(expr)#length of input
    while i < n:
        if expr[i].isdigit() or expr[i] == '.':   #if it is a number
            num = []
            while i < n and (expr[i].isdigit() or expr[i] == '.'):
                num.append(expr[i])#add to number list
                i += 1
            output.append(''.join(num))#output numbers till next character isn't digit
            continue
        elif expr[i] == '(': #prioritize (
            stack.append('(')
        elif expr[i] == ')':
            while stack and stack[-1] != '(':
                output.append(stack.pop()) #output all the things inside the bracket
            stack.pop()   #pop out right bracket
        else:
            while stack and stack[-1] != '(': #is a operator
                top = stack[-1] #check what the top stack is
                prec_top = 2 if top in '*/' else 1 #if top of stack is */, assign 2
                prec_curr = 2 if expr[i] in '*/' else 1 #if current index is */, assign 2
                if prec_top >= prec_curr: #compare top stack with current index
                    output.append(stack.pop()) #output top stack
                else:
```

Ultra quicksort

🔗 http://cs101.openjudge.cn/practice/02299/

```
import sys
sys.setrecursionlimit(10**7)

def merge_count(arr):
    n = len(arr)
    if n <= 1:
        return arr, 0

    mid = n // 2
    left, inv_left = merge_count(arr[:mid])
    right, inv_right = merge_count(arr[mid:])

    merged = []
    i = j = 0
    inv_count = inv_left + inv_right

    while i < len(left) and j < len(right):
        if left[i] <= right[j]:
            merged.append(left[i])
            i += 1
        else:
            merged.append(right[j])
            inv_count += len(left) - i
            j += 1

    merged.extend(left[i:])
    merged.extend(right[j:])

    return merged, inv_count


while True:
    n = int(sys.stdin.readline())
```

LRU 缓存

🔗 https://leetcode.cn/problems/lru-cache/

```
import sys
sys.setrecursionlimit(10**7)

def merge_count(arr):
    n = len(arr)
    if n <= 1:
        return arr, 0

    mid = n // 2
    left, inv_left = merge_count(arr[:mid])
    right, inv_right = merge_count(arr[mid:])

    merged = []
    i = j = 0
    inv_count = inv_left + inv_right

    while i < len(left) and j < len(right):
        if left[i] <= right[j]:
            merged.append(left[i])
            i += 1
        else:
            merged.append(right[j])
            inv_count += len(left) - i
            j += 1

    merged.extend(left[i:])
    merged.extend(right[j:])

    return merged, inv_count


while True:
    n = int(sys.stdin.readline())
```

Assignment 7: BFS

节省存储的矩阵乘法

🔗 http://cs101.openjudge.cn/practice/23555

```
import sys
from collections import defaultdict

def main():
    data = sys.stdin.read().strip().split()
    it = iter(data)

    m = int(next(it))
    n = int(next(it))
    k = int(next(it))

    rest = []
    while True:
        try:
            r = int(next(it))
            c = int(next(it))
            v = int(next(it))
            rest.append((r, c, v))
        except StopIteration:
            break

    # A 的非零元素个数
    a_cnt = len(rest) - k

    A = rest[:a_cnt]
    B = rest[a_cnt:]

    # 构造 B 的列映射：B[k][j]
    Bmap = defaultdict(list)
    for r, c, v in B:
        Bmap[r].append((c, v))

    # 结果矩阵
```

二叉树的层序遍历

https://leetcode.cn/problems/binary-tree-level-order-traversal/

```
from collections import deque

class Solution:
    def levelOrder(self, root):
        if not root:
            return []
        res, q = [], deque([root])#create a queue, start from root
        while q:
            level = [] #nodes on this level
            for _ in range(len(q)): #for all the nodes on this level
                node = q.popleft() #pop it
                level.append(node.val) #add it to the result
                if node.left: #add the left children to the queue
                    q.append(node.left)
                if node.right: #add the right children
                    q.append(node.right)
            res.append(level)
        return res
```

For BFS on a tree, you need:

1. **A queue** (usually collections.deque)
2. **An initial element** (the root)
3. **A loop while the queue is not empty**
4. **Level control** (know how many nodes are in the current level)
5. **Add children to the queue**

分割回文串

https://leetcode.cn/problems/palindrome-partitioning/

```
class Solution:
    def partition(self, s):
        res = []
        path = []

        def backtrack(start):
            if start == len(s):
                res.append(path[:])
                return
            for end in range(start + 1, len(s) + 1):
                sub = s[start:end]
                if sub == sub[::-1]:
                    path.append(sub)
                    backtrack(end)
                    path.pop()

        backtrack(0)
        return res
```

岛屿数量

https://leetcode.cn/problems/number-of-islands/

```
class Solution:
    def numIslands(self, grid):
        if not grid:
            return 0
        m, n = len(grid), len(grid[0])

        def dfs(i, j):
            if i < 0 or i >= m or j < 0 or j >= n or grid[i][j] == '0':
                return
            grid[i][j] = '0'
            dfs(i+1, j)
            dfs(i-1, j)
            dfs(i, j+1)
            dfs(i, j-1)

        count = 0
        for i in range(m):
            for j in range(n):
                if grid[i][j] == '1':
                    count += 1
                    dfs(i, j)
        return count
```

DFS is good when:

- You want to fully explore one connected region
- Mark everything visited
- Then move on to find the next region

That’s exactly what an island is.

High-level DFS strategy

1. Traverse every cell in the grid
2. When you see a '1':
    a. You’ve found a new island

    b. Increment island count
    c. Use DFS to flood-fill the entire island (mark it visited)

3. Continue scanning the grid

DFS essentials (mental checklist)

For DFS on a grid, you need:

1. Boundary check
    a. dont go out of bounds
    b. follow the rules/dont revisit nodes
2. Base case (water or already visited)
3. Mark visited
4. Explore neighbors recursively

how to mark “visited”

Two common methods:

Option A (most common): modify the grid

Option B: separate visited set

最深叶节点的最近公共祖先

🔗 https://leetcode.cn/problems/lowest-common-ancestor-of-deepest-leaves/

```python
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right

class Solution:
    def lcaDeepestLeaves(self, root: TreeNode) -> TreeNode:
        def dfs(node):
            if not node:
                return None, 0

            left_lca, left_depth = dfs(node.left)
            right_lca, right_depth = dfs(node.right)

            if left_depth > right_depth:
                return left_lca, left_depth + 1
            elif right_depth > left_depth:
                return right_lca, right_depth + 1
            else:
                return node, left_depth + 1

        lca, _ = dfs(root)
        return lca

```

单词搜索

https://leetcode.cn/problems/word-search/

```
class Solution:
    def exist(self, board, word):
        m, n = len(board), len(board[0])
        visited = [[False]*n for _ in range(m)]

        def dfs(i, j, k):
            if k == len(word):
                return True
            if i < 0 or i >= m or j < 0 or j >= n:
                return False
            if visited[i][j] or board[i][j] != word[k]:
                return False

            visited[i][j] = True
            res = (
                dfs(i+1, j, k+1) or
                dfs(i-1, j, k+1) or
                dfs(i, j+1, k+1) or
                dfs(i, j-1, k+1)
            )
            visited[i][j] = False
            return res

        for i in range(m):
            for j in range(n):
                if dfs(i, j, 0):
                    return True
        return False

```

Assignment 8: Tree

有序组转换为二叉搜索树

https://leetcode.cn/problems/convert-sorted-array-to-binary-search-tree/

```
class Solution:
    def sortedArrayToBST(self, nums):
        if not nums:
            return None
        mid = len(nums) // 2
        root = TreeNode(nums[mid])
        root.left = self.sortedArrayToBST(nums[:mid])
        root.right = self.sortedArrayToBST(nums[mid+1:])
        return root
```

森林的带度数层次序列存储

http://cs101.openjudge.cn/practice/07161/

```
from collections import deque

class Node:
    def __init__(self, val):
        self.val = val
        self.children = []

def build_tree(tokens):
    """
    根据带度数的层次序列恢复一棵树
    tokens: [C, 3, E, 3, F, 0, ...]
    """
    idx = 0
    root = Node(tokens[idx])
    root_degree = int(tokens[idx + 1])
    idx += 2

    queue = deque()
    queue.append((root, root_degree))

    while queue:
        node, degree = queue.popleft()
        for _ in range(degree):
            child_val = tokens[idx]
            child_degree = int(tokens[idx + 1])
            idx += 2

            child = Node(child_val)
            node.children.append(child)
            queue.append((child, child_degree))

    return root
```

遍历树

🔗 http://cs101.openjudge.cn/practice/27928/

```python
def solve():
    n = int(input().strip())

    children = {}
    nodes = set()
    child_nodes = set()

    # 建立树结构
    for _ in range(n):
        parts = list(map(int, input().split()))
        u = parts[0]
        nodes.add(u)
        if len(parts) > 1:
            children[u] = parts[1:]
            for v in parts[1:]:
                child_nodes.add(v)
                nodes.add(v)
        else:
            children[u] = []

    # 找根节点
    root = (nodes - child_nodes).pop()

    result = []

    # DFS 按题目规则遍历
    def dfs(u):
        group = [u] + children[u]
        group.sort()

        for x in group:
            if x == u:
                result.append(u)
```

秋根节点到叶节点数字之和

https://leetcode.cn/problems/sum-root-to-leaf-numbers/

```
class Solution:
    def sumNumbers(self, root):
        def dfs(node, num):
            if not node:
                return 0
            num = num * 10 + node.val
            if not node.left and not node.right:
                return num
            return dfs(node.left, num) + dfs(node.right, num)
        return dfs(root, 0)
```

括号嵌套树

http://cs101.openjudge.cn/practice/24729/

```python
def solve():
    s = input().strip()
    n = len(s)
    i = 0

    preorder = []
    postorder = []

    def parse():
        nonlocal i

        # 当前一定是一个结点
        root = s[i]
        preorder.append(root)
        i += 1

        # 如果有子树
        if i < n and s[i] == '(':
            i += 1  # 跳过 '('
            while True:
                parse()  # 解析子树
                if s[i] == ',':
                    i += 1  # 跳过 ','
                elif s[i] == ')':
                    i += 1  # 跳过 ')'
                    break

        postorder.append(root)

    parse()

    print("".join(preorder))
    print("".join(postorder))
```

文件结构图

http://cs101.openjudge.cn/practice/02775/

```
class Node:
    def __init__(self, name, is_dir):
        self.name = name
        self.is_dir = is_dir
        self.dirs = []      # 子目录
        self.files = []     # 文件


def print_tree(node, depth):
    indent = "|     " * depth

    # 先打印目录
    for d in node.dirs:
        print(f"{indent}{d.name}")
        print_tree(d, depth + 1)

    # 再打印文件（按字典序）
    for f in sorted(node.files):
        print(f"{indent}{f}")


def main():
    data_set = 1

    while True:
        root = Node("ROOT", True)
        stack = [root]

        while True:
            try:
                line = input().strip()
            except EOFError:
                return
```

Assignment 9