

Assignment 1:

Fraction 类

```
1 import math
2
3 class Fraction:
4     def __init__(self, numerator, denominator):
5         if denominator == 0:
6             raise ValueError("Denominator cannot be zero")
7         # 保证分母为正
8         if denominator < 0:
9             numerator = -numerator
10            denominator = -denominator
11
12         gcd = math.gcd(numerator, denominator)
13         self.numerator = numerator // gcd
14         self.denominator = denominator // gcd
15
16     def __add__(self, other):
17         new_numerator = (
18             self.numerator * other.denominator +
19             other.numerator * self.denominator
20         )
21         new_denominator = self.denominator * other.denominator
22         return Fraction(new_numerator, new_denominator)
23
24     def __str__(self):
25         return f"{self.numerator}/{self.denominator}"
26
27
28     # 读取输入
29     a, b, c, d = map(int, input().split())
30
31     f1 = Fraction(a, b)
32     f2 = Fraction(c, d)
33
```

袋子里最少数目的球

```
1  class Solution:
2      def minimumSize(self, nums, maxOperations):
3          left, right = 1, max(nums)
4          ans = right
5
6          while left <= right:
7              mid = (left + right) // 2
8              operation = 0
9              for num in nums:
10                  # integer-safe and correct count of splits needed
11                  operation += (num - 1) // mid
12
13          if operation <= maxOperations:
14              ans = mid
15              right = mid - 1
16          else:
17              left = mid + 1
18
19      return ans
```

月度开销

```
1  def can_split(costs, M, max_month):
2      months = 1
3      current_sum = 0
4
5      for cost in costs:
6          if current_sum + cost <= max_month:
7              current_sum += cost
8          else:
9              months += 1
10             current_sum = cost
11             if months > M:
12                 return False
13
14
15
16 def main():
17     N, M = map(int, input().split())
18     costs = [int(input()) for _ in range(N)]
19
20     left = max(costs)      # 至少能装下最大的一天
21     right = sum(costs)     # 最多把所有天放一个月
22     answer = right
23
24     while left <= right:
25         mid = (left + right) // 2
26         if can_split(costs, M, mid):
27             answer = mid
28             right = mid - 1
29         else:
30             left = mid + 1
31
32     print(answer)
33
```

note : 二分查找

模型整理

```
1  n = int(input())
2
3  models = {}
4
5  for _ in range(n):
6      line = input().strip()
7      name, param = line.split('-')
8
9      # 解析参数量
10     if param.endswith('M'):
11         value = float(param[:-1])
12         value_in_m = value
13     else: # endswith 'B'
14         value = float(param[:-1])
15         value_in_m = value * 1000
16
17     if name not in models:
18         models[name] = []
19
20     models[name].append((value_in_m, param))
21
22 # 按模型名排序
23 for name in sorted(models.keys()):
24     # 按参数量排序
25     models[name].sort(key=lambda x: x[0])
26     params = [p for _, p in models[name]]
27     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：

- $xM \rightarrow x$
- $yB \rightarrow y * 1000$

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

Assignment 2

泰波拿契數

🔗 <http://cs101.openjudge.cn/practice/20742/>

```
1 n = int(input())
2
3 if n == 0:
4     print(0)
5 elif n == 1 or n == 2:
6     print(1)
7 else:
8     T = [0] * (n + 1)
9     T[0] = 0
10    T[1] = 1
11    T[2] = 1
12
13    for i in range(3, n + 1):
14        T[i] = T[i - 1] + T[i - 2] + T[i - 3]
15
16    print(T[n])
```

一定要确定 range, edge case

Chat Room

🔗 <http://codeforces.com/problemset/problem/58/A>

```
1 s = input().strip()
2 target = "hello"
3
4 idx = 0
5 for ch in s:
6     if ch == target[idx]:
7         idx += 1
8         if idx == len(target):
9             break
10
11 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>

```
1 s = input().strip()
2 vowels = set("aoyeui")
3
4 result = []
5
6 for ch in s:
7     if ch.lower() not in vowels:
8         result.append("." + ch.lower())
9
10 print"".join(result))
```

.lower(), .upper()

add to a string: use append

Goldbach Conjecture

🔗 <http://cs101.openjudge.cn/practice/22359/>

```
1 def is_prime(x):
2     if x < 2:
3         return False
4     for i in range(2, int(x ** 0.5) + 1):
5         if x % i == 0:
6             return False
7     return True
8
9
10 S = int(input())
11
12 for A in range(2, S):
13     B = S - A
14     if is_prime(A) and is_prime(B):
15         print(A, B)
16         break
```

** is sqrt

% is remainder

多项式时间复杂度

🔗 <http://cs101.openjudge.cn/pctbook/E23563/>

```
1 s = input().strip()
2 terms = s.split('+') #split at specific terms
3
4 max_power = 0
5
6 for term in terms: #every character
7     # 处理系数和指数
8     if 'n' not in term:
9         # 常数项，相当于 n^0
10        coef = int(term)
11        power = 0
12    else:
13        # 拆分系数
14        if term.startswith('n'): #startswith()
15            coef = 1
16            rest = term
17        else:
18            coef_part, rest = term.split('n', 1)
19            coef = int(coef_part)
20
21        # 拆分指数
22        if '^' in rest:
23            power = int(rest.split('^')[1])
24        else:
25            power = 1 # 底处理 n 的情况（虽然题目一般不会给）
26
27        if coef != 0:
28            max_power = max(max_power, power)
29
30 print(f"n^{max_power}")
```

直播计票

🔗 <http://cs101.openjudge.cn/practice/24684/>

```
1  votes = list(map(int, input().split()))
2
3  count = {}
4  for v in votes:
5      count[v] = count.get(v, 0) + 1
6
7  max_votes = max(count.values())
8
9  result = [k for k, v in count.items() if v == max_votes]
10 result.sort()
11
12 print(" ".join(map(str, result)))
```

Assignment 3

Bigram 分词

🔗 <https://leetcode.cn/problems/occurrences-after-bigram/>

```
1  class Solution(object):
2      def findOcurrences(self, text, first, second):
3          words = text.split(" ")
4          result = []
5          for i in range(len(words) - 2):
6              if words[i] == first and words[i+1] == second:
7                  result.append(words[i+2]) #append: add to list
8
9      return result
```

移动零

🔗 <https://leetcode.cn/problems/move-zeroes/>

```

1  class Solution:
2      def moveZeroes(self, nums):
3          non_zero_pos = 0
4          for i in range(len(nums)):
5              if nums[i] != 0:
6                  nums[non_zero_pos] = nums[i]
7                  nums[i] = nums[non_zero_pos]
8                  non_zero_pos += 1
9          for i in range(non_zero_pos, len(nums)):
10             nums[i] = 0
11
12     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/>

```

1  class Solution(object):
2      def isValid(self, s):
3          stack = []
4          pairs = {')': '(', '}': '{', ']': '['} #mapping whihc ones pair together
5          for char in s:
6              if char in '({[':
7                  stack.append(char)
8              else:
9                  if not stack or stack[-1] != pairs[char]:
10                     return False
11                  stack.pop()
12
13      return not stack

```

mapping pairs is crucial 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/>

```
1 class Solution:
2     def generate(self, numRows: int):
3         triangle = []
4
5         for i in range(numRows):
6             row = [1] * (i + 1) # 初始化每行
7             for j in range(1, i):
8                 row[j] = triangle[i-1][j-1] + triangle[i-1][j]
9             triangle.append(row)
10
11         return triangle
```

全排列

🔗 <https://leetcode.cn/problems/permutations/description/>

```
1  from typing import List
2
3  class Solution:
4      def permute(self, nums: List[int]) -> List[List[int]]:
5          res = []
6          path = []
7          used = [False] * len(nums)
8
9          def backtrack():
10             if len(path) == len(nums):
11                 res.append(path[:])
12                 return
13
14             for i in range(len(nums)):
15                 if used[i]:
16                     continue
17
18                 used[i] = True
19                 path.append(nums[i])
20                 backtrack()
21                 path.pop()
22                 used[i] = False
23
24             backtrack()
25         return res
26
27 class Solution:
28     def permute(self, nums):
29         result = [] # to store all permutations
30
31         def backtrack(path, remaining):
32             # Base case: no remaining numbers → a full permutation is
33             formed
```

```
1 def infix_to_postfix(expr):
2     output = [] #initialize answer
3     stack = []# initialize stack for comparison
4     i = 0
5     n = len(expr)#length of input
6     while i < n:
7         if expr[i].isdigit() or expr[i] == '.': #if it is a
number
8             num = []
9             while i < n and (expr[i].isdigit() or expr[i] ==
'.'):
10                 num.append(expr[i])#add to number list
11                 i += 1
12             output.append(''.join(num))#output numbers till
next character isn't digit
13             continue
14         elif expr[i] == '(': #prioritize (
15             stack.append('(')
16         elif expr[i] == ')':
17             while stack and stack[-1] != '(':
18                 output.append(stack.pop()) #output all the
things inside the bracket
19             stack.pop() #pop out right bracket
```

```
20     else:
21         while stack and stack[-1] != ')': #is a operator
22             top = stack[-1] #check what the top stack is
23             prec_top = 2 if top in '*/' else 1 #if top of
24                 stack is */, assign 2
25             prec_curr = 2 if expr[i] in '*/' else 1 #if
26                 current index is */, assign 2
27             if prec_top >= prec_curr: #compare top stack
28                 with current index
29                 output.append(stack.pop()) #output top
30                 stack
31             else:
32                 break
33             stack.append(expr[i]) #add operator to stack if
34                 not bigger
35             i += 1
36     while stack:
37         output.append(stack.pop()) #when no more digits to
38             check, output all of stack's remaining operators
39     return ' '.join(output)
40 n = int(input())
41 for _ in range(n):
```

子集

🔗 <https://leetcode.cn/problems/subsets/>

```
1  from typing import List
2
3  class Solution:
4      def subsets(self, nums: List[int]) -> List[List[int]]:
5          res = []
6          path = []
7
8          def backtrack(start):
9              # 当前 path 本身就是一个子集
10             res.append(path[:])
11
12             for i in range(start, len(nums)):
13                 path.append(nums[i])      # 选择 nums[i]
14                 backtrack(i + 1)        # 递归处理后面的元素
15                 path.pop()            # 撤销选择
16
17             backtrack(0)
18
19         return res
```

compiler error

wrong answer

correct

Assignment 4

矩阵运算

↙ <http://cs101.openjudge.cn/pctbook/E18161/>

```
1 def read_matrix():
2     r, c = map(int, input().split())
3     mat = []
4     for _ in range(r):
5         mat.append(list(map(int, input().split())))
6     return r, c, mat
7
8
9 # 读取三个矩阵
10 ar, ac, A = read_matrix()
11 br, bc, B = read_matrix()
12 cr, cc, C = read_matrix()
13
14 # 检查乘法是否合法
15 if ac != br:
16     print("Error!")
17     exit()
18
19 # 乘法结果维度
20 mr, mc = ar, bc
21
22 # 检查加法是否合法
23 if mr != cr or mc != cc:
24     print("Error!")
25     exit()
26
27 # 计算 A · B
28 D = [[0] * mc for _ in range(mr)]
29 for i in range(mr):
30     for j in range(mc):
31         for k in range(ac):
32             D[i][j] += A[i][k] * B[k][j]
33
```

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/>

```
1 # 读取输入
2 m, n, p, q = map(int, input().split())
3
4 matrix = [list(map(int, input().split())) for _ in range(m)]
5 kernel = [list(map(int, input().split())) for _ in range(p)]
6
7 # 输出矩阵大小
8 out_rows = m - p + 1
9 out_cols = n - q + 1
10
11 # 卷积计算
12 result = [[0] * out_cols for _ in range(out_rows)]
13
14 for i in range(out_rows):
15     for j in range(out_cols):
16         s = 0
17         for x in range(p):
18             for y in range(q):
19                 s += matrix[i + x][j + y] * kernel[x][y]
20         result[i][j] = s
21
22 # 输出结果
23 for row in result:
24     print(" ".join(map(str, row)))
```

倒排索引

🔗 <http://cs101.openjudge.cn/pctbook/M06640/>

```

1 # Read number of documents
2 N = int(input())
3
4 index = {} # inverted index: word -> list of doc IDs
5
6 for doc_id in range(1, N + 1):
7     parts = input().split()
8     c = int(parts[0])      # number of words
9     words = parts[1:]
10
11    seen = set() # avoid duplicate words in the same document
12
13    for word in words:
14        if word in seen:
15            continue
16        seen.add(word)
17
18        if word not in index:
19            index[word] = []
20            index[word].append(doc_id)
21
22 # Read number of queries
23 M = int(input())
24
25 for _ in range(M):
26     query = input().strip()
27
28     if query in index:
29         print(" ".join(map(str, index[query])))
30     else:
31         print("NOT FOUND")

```

use a dictionary to match values to items

use continue to skip the loop once for this iteration

相交链表

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

```
1  class Solution:
2      def getIntersectionNode(self, headA, headB):
3          # Step 1: Get lengths of both lists
4          def get_length(head):
5              length = 0
6              while head:
7                  length += 1
8                  head = head.next
9              return length
10
11         lenA = get_length(headA)
12         lenB = get_length(headB)
13
14         # Step 2: Align both lists by skipping the difference in lengths
15         while lenA > lenB:
16             headA = headA.next
17             lenA -= 1
18         while lenB > lenA:
19             headB = headB.next
20             lenB -= 1
21
22         # Step 3: Traverse both lists and find the intersection
23         while headA and headB:
24             if headA == headB:
25                 return headA # Intersection found
26             headA = headA.next
27             headB = headB.next
28
29         return None # No intersection
```

反转链表

🔗 <https://leetcode.cn/problems/reverse-linked-list/>

```
1 class Solution:
2     def reverseList(self, head):
3         prev = None
4         curr = head
5         while curr:
6             nxt = curr.next
7             curr.next = prev
8             prev = curr
9             curr = nxt
10        return prev
```

Knight's journey

↙ <http://cs101.openjudge.cn/practice/02488/>

```

1  def solve():
2      t = int(input().strip())
3
4      # Knight moves
5      moves = [
6          (2, 1), (1, 2), (-1, 2), (-2, 1),
7          (-2, -1), (-1, -2), (1, -2), (2, -1)
8      ]
9
10     for case in range(1, t + 1):
11         p, q = map(int, input().split())
12         total = p * q
13
14         visited = [[False] * q for _ in range(p)]
15         path = []
16         found = False
17
18         def dfs(r, c, step):
19             nonlocal found
20             if found:
21                 return
22             visited[r][c] = True
23             path.append((r, c))
24
25             if step == total:
26                 found = True
27                 return
28
29             next_moves = []
30             for dr, dc in moves:
31                 nr, nc = r + dr, c + dc
32                 if 0 <= nr < p and 0 <= nc < q and not visited[nr][nc]:
33                     next_moves.append((nr, nc))

```

Mock Exam Assignment 5

咒语序列

🔗 <http://cs101.openjudge.cn/practice/29952/>

```
1 s = input().strip()
2
3 stack = [-1]    # 哨兵
4 max_len = 0
5
6 for i, ch in enumerate(s):
7     if ch == '(':
8         stack.append(i)
9     else:    # ch == ')'
10        stack.pop()
11    if not stack:
12        # 重新设定起点
13        stack.append(i)
14    else:
15        max_len = max(max_len, i - stack[-1])
16
17 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/>

```
1 import math
2
3 case_num = 1
4
5 while True:
6     line = input().strip()
7     if not line:
8         continue
9     n, d = map(int, line.split())
10    if n == 0 and d == 0:
11        break
12
13    intervals = []
14    impossible = False
15
16    for _ in range(n):
17        x, y = map(int, input().split())
18        if y > d:
19            impossible = True
20        else:
21            dx = math.sqrt(d * d - y * y)
22            intervals.append((x - dx, x + dx))
23
24    if impossible:
25        print(f"Case {case_num}: -1")
26        case_num += 1
27        continue
28
29    # 按右端点排序
30    intervals.sort(key=lambda x: x[1])
31
32    count = 0
33    pos = -float('inf')
```

八皇后

🔗 <http://cs101.openjudge.cn/practice/02754/>

```
1 # 预计算 8 皇后所有解
2 solutions = []
3
4 def dfs(row, cols, diag1, diag2, path):
5     if row == 8:
6         # 转成字符串
7         solutions.append("".join(str(c + 1) for c in path))
8         return
9     for col in range(8):
10         if col in cols or (row - col) in diag1 or (row + col) in diag2: #check
11             if there are any in the same column, diagonal or other diagonal direction
12                 continue
13         dfs(
14             row + 1,
15             cols | {col}, #update column
16             diag1 | {row - col}, #update diag1
17             diag2 | {row + col}, #update diag2
18             path + [col] #
19         )
20
21
22 # 排序 ( 按整数大小 )
23 solutions.sort()
24
25 # 处理输入
26 n = int(input())
27 for _ in range(n):
28     b = int(input())
29     print(solutions[b - 1])
```

USE DFS

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

洋葱

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

```
1 n = int(input())
2 mat = [list(map(int, input().split())) for _ in range(n)]
3
4 max_sum = 0
5 layers = (n + 1) // 2
6
7 for k in range(layers):
8     layer_sum = 0
9     top = k
10    bottom = n - 1 - k
11    left = k
12    right = n - 1 - k
13
14    if top == bottom and left == right:
15        # 中心单点
16        layer_sum = mat[top][left]
17    else:
18        # 上边
19        for j in range(left, right + 1):
20            layer_sum += mat[top][j]
21        # 下边
22        for j in range(left, right + 1):
23            layer_sum += mat[bottom][j]
24        # 左边 (不含角)
25        for i in range(top + 1, bottom):
26            layer_sum += mat[i][left]
27        # 右边 (不含角)
28        for i in range(top + 1, bottom):
29            layer_sum += mat[i][right]
30
31    max_sum = max(max_sum, layer_sum)
32
33 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/>

```
1  from collections import deque
2
3  R, C, K = map(int, input().split())
4  grid = [list(input().strip()) for _ in range(R)]
5
6  # 找起点
7  for i in range(R):
8      for j in range(C):
9          if grid[i][j] == 'S':
10             sr, sc = i, j
11
12 # visited[r][c][used_flash]
13 visited = [[[False] * (K + 1) for _ in range(C)] for _ in range(R)]
14
15 queue = deque()
16 queue.append((sr, sc, 0, 0))  # r, c, used_flash, steps
17 visited[sr][sc][0] = True
18
19 dirs = [(1, 0), (-1, 0), (0, 1), (0, -1)]
20
21 while queue:
22     r, c, used, steps = queue.popleft()
23
24     if grid[r][c] == 'E':
25         print(steps)
26         exit(0)
27
28     for dr, dc in dirs:
29         nr, nc = r + dr, c + dc
30         if 0 <= nr < R and 0 <= nc < C:
31             cell = grid[nr][nc]
32             if cell == '#':
33                 if used < K and not visited[nr][nc][used + 1]:
```

当前队列中位数

↙ <http://cs101.openjudge.cn/practice/27256/>

```
1 import heapq
2 #min-heap, max-heap
3 from collections import deque
4 #appendx: add to back
5 #popleft: remove from the front
6 from collections import defaultdict
7
8 n = int(input())
9
10
11 # FIFO queue to remember insertion order
12 queue = deque() #create queue
13
14 # Heaps for median
15 small = [] # max-heap (store negative values)
16 large = [] # min-heap
17
18 # Lazy deletion counter
19 delayed = defaultdict(int)
20
21 # Sizes excluding delayed elements
22 small_size = 0
23 large_size = 0
24
25
26 def prune(heap):
27     """
28         Remove elements from heap top that are marked for deletion
29     """
30     while heap:
31         x = -heap[0] if heap is small else heap[0]
32         if delayed[x] > 0:
33             delayed[x] -= 1
```

Assignment 6 : 链表，栈，排序

后续表达式求值

🔗 <http://cs101.openjudge.cn/practice/24588/>

```
1  n = int(input())
2
3  for _ in range(n):
4      expr = input().strip().split()
5      stack = []
6
7      for token in expr:
8          if token in {"+", "-", "*", "/"}:
9              b = stack.pop()
10             a = stack.pop()
11             if token == "+":
12                 stack.append(a + b)
13             elif token == "-":
14                 stack.append(a - b)
15             elif token == "*":
16                 stack.append(a * b)
17             elif token == "/":
18                 stack.append(a / b)
19         else:
20             # 操作数
21             stack.append(float(token))
22
23     result = stack.pop()
24     print(f"{result:.2f}")
25
```

回文链表

🔗 <https://leetcode.cn/problems/palindrome-linked-list/>

```
1 # Definition for singly-linked list.
2
3 # class ListNode:
4 #     def __init__(self, val=0, next=None):
5 #         self.val = val
6 #         self.next = next
7
8 class Solution:
9
10    def isPalindrome(self, head) -> bool:
11        if not head or not head.next:
12            return True
13
14        slow = fast = head
15
16        while fast and fast.next:
17            slow = slow.next
18            fast = fast.next.next
19
20
21        prev = None
22        curr = slow
23
24        while curr:
25            next_temp = curr.next
26            curr.next = prev
27            prev = curr
28            curr = next_temp
29
30
31        left, right = head, prev
32
33        while right:
34            if left.val != right.val:
35                return False
36            left = left.next
37            right = right.next
38
39        return True
```

有多少种合法的出栈顺序

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

```
1 n = int(input())
2
3 catalan = 1
4 for i in range(2, n + 1):
5     catalan = catalan * (n + i) // i
6
7 catalan //= (n + 1)
8
9 print(catalan)
```

中序表达式转后续表达式

🔗 <http://cs101.openjudge.cn/practice/24591/>

```

1  def infix_to_postfix(expr):
2      output = [] #initialize answer
3      stack = []# initialize stack for comparison
4
5      i = 0
6      n = len(expr)#length of input
7      while i < n:
8          if expr[i].isdigit() or expr[i] == '.': #if it is a number
9              num = []
10             while i < n and (expr[i].isdigit() or expr[i] == '.'):
11                 num.append(expr[i])#add to number list
12                 i += 1
13             output.append(''.join(num))#output numbers till next
14             character isn't digit
15             continue
16         elif expr[i] == '(': #prioritize (
17             stack.append('(')
18         elif expr[i] == ')':
19             while stack and stack[-1] != '(':
20                 output.append(stack.pop()) #output all the things inside
21                 the bracket
22                 stack.pop() #pop out right bracket
23             else:
24                 while stack and stack[-1] != '(': #is a operator
25                     top = stack[-1] #check what the top stack is
26                     prec_top = 2 if top in '*/' else 1 #if top of stack is */,
27                     assign 2
28                     prec_curr = 2 if expr[i] in '*/' else 1 #if current index is */,
29                     assign 2
30                     if prec_top >= prec_curr: #compare top stack with current
31                         index
32                         output.append(stack.pop()) #output top stack
33                     else:

```

Ultra quicksort

↙ <http://cs101.openjudge.cn/practice/02299/>

```
1 import sys
2 sys.setrecursionlimit(10**7)
3
4 def merge_count(arr):
5     n = len(arr)
6     if n <= 1:
7         return arr, 0
8
9     mid = n // 2
10    left, inv_left = merge_count(arr[:mid])
11    right, inv_right = merge_count(arr[mid:])
12
13    merged = []
14    i = j = 0
15    inv_count = inv_left + inv_right
16
17    while i < len(left) and j < len(right):
18        if left[i] <= right[j]:
19            merged.append(left[i])
20            i += 1
21        else:
22            merged.append(right[j])
23            inv_count += len(left) - i
24            j += 1
25
26    merged.extend(left[i:])
27    merged.extend(right[j:])
28
29    return merged, inv_count
30
31
32 while True:
33     n = int(sys.stdin.readline())
```

LRU 缓存

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

```
1 import sys
2 sys.setrecursionlimit(10**7)
3
4 def merge_count(arr):
5     n = len(arr)
6     if n <= 1:
7         return arr, 0
8
9     mid = n // 2
10    left, inv_left = merge_count(arr[:mid])
11    right, inv_right = merge_count(arr[mid:])
12
13    merged = []
14    i = j = 0
15    inv_count = inv_left + inv_right
16
17    while i < len(left) and j < len(right):
18        if left[i] <= right[j]:
19            merged.append(left[i])
20            i += 1
21        else:
22            merged.append(right[j])
23            inv_count += len(left) - i
24            j += 1
25
26    merged.extend(left[i:])
27    merged.extend(right[j:])
28
29    return merged, inv_count
30
31
32 while True:
33     n = int(sys.stdin.readline())
```

Assignment 7: BFS

节省存储的矩阵乘法

↙ <http://cs101.openjudge.cn/practice/23555>

```
1 import sys
2 from collections import defaultdict
3
4 def main():
5     data = sys.stdin.read().strip().split()
6     it = iter(data)
7
8     m = int(next(it))
9     n = int(next(it))
10    k = int(next(it))
11
12    rest = []
13    while True:
14        try:
15            r = int(next(it))
16            c = int(next(it))
17            v = int(next(it))
18            rest.append((r, c, v))
19        except StopIteration:
20            break
21
22    # A 的非零元素个数
23    a_cnt = len(rest) - k
24
25    A = rest[:a_cnt]
26    B = rest[a_cnt:]
27
28    # 构造 B 的列映射 : B[k][j]
29    Bmap = defaultdict(list)
30    for r, c, v in B:
31        Bmap[r].append((c, v))
32
33    # 填充矩阵
```

二叉树的层序遍历

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

```
1  from collections import deque
2
3  class Solution:
4      def levelOrder(self, root):
5          if not root:
6              return []
7          res, q = [], deque([root])#create a queue, start from root
8          while q:
9              level = [] #nodes on this level
10             for _ in range(len(q)): #for all the nodes on this level
11                 node = q.popleft() #pop it
12                 level.append(node.val) #add it to the result
13                 if node.left: #add the left children to the queue
14                     q.append(node.left)
15                 if node.right: #add the right children
16                     q.append(node.right)
17             res.append(level)
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
159
160
161
162
163
164
165
166
167
168
169
169
170
171
172
173
174
175
176
177
178
179
179
180
181
182
183
184
185
186
187
188
189
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
209
210
211
212
213
214
215
216
217
218
219
219
220
221
222
223
224
225
226
227
228
229
229
230
231
232
233
234
235
236
237
238
239
239
240
241
242
243
244
245
246
247
248
249
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
```

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/>

```
1 class Solution:
2     def partition(self, s):
3         res = []
4         path = []
5
6         def backtrack(start):
7             if start == len(s):
8                 res.append(path[:])
9                 return
10            for end in range(start + 1, len(s) + 1):
11                sub = s[start:end]
12                if sub == sub[::-1]:
13                    path.append(sub)
14                    backtrack(end)
15                    path.pop()
16
17            backtrack(0)
18        return res
```

岛屿数量

🔗 <https://leetcode.cn/problems/number-of-islands/>

```

1  class Solution:
2      def numIslands(self, grid):
3          if not grid:
4              return 0
5          m, n = len(grid), len(grid[0])
6
7          def dfs(i, j):
8              if i < 0 or i >= m or j < 0 or j >= n or grid[i][j] == '0':
9                  return
10             grid[i][j] = '0'
11             dfs(i+1, j)
12             dfs(i-1, j)
13             dfs(i, j+1)
14             dfs(i, j-1)
15
16             count = 0
17             for i in range(m):
18                 for j in range(n):
19                     if grid[i][j] == '1':
20                         count += 1
21                         dfs(i, j)
22             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/>

```
1 # Definition for a binary tree node.
2 # class TreeNode:
3 #     def __init__(self, val=0, left=None, right=None):
4 #         self.val = val
5 #         self.left = left
6 #         self.right = right
7
8 class Solution:
9     def lcaDeepestLeaves(self, root: TreeNode) -> TreeNode:
10         def dfs(node):
11             if not node:
12                 return None, 0
13
14             left_lca, left_depth = dfs(node.left)
15             right_lca, right_depth = dfs(node.right)
16
17             if left_depth > right_depth:
18                 return left_lca, left_depth + 1
19             elif right_depth > left_depth:
20                 return right_lca, right_depth + 1
21             else:
22                 return node, left_depth + 1
23
24         lca, _ = dfs(root)
25         return lca
```

单词搜索

🔗 <https://leetcode.cn/problems/word-search/>

```

1  class Solution:
2      def exist(self, board, word):
3          m, n = len(board), len(board[0])
4          visited = [[False]*n for _ in range(m)]
5
6          def dfs(i, j, k):
7              if k == len(word):
8                  return True
9              if i < 0 or i >= m or j < 0 or j >= n:
10                 return False
11              if visited[i][j] or board[i][j] != word[k]:
12                  return False
13
14              visited[i][j] = True
15              res = (
16                  dfs(i+1, j, k+1) or
17                  dfs(i-1, j, k+1) or
18                  dfs(i, j+1, k+1) or
19                  dfs(i, j-1, k+1)
20              )
21              visited[i][j] = False
22              return res
23
24          for i in range(m):
25              for j in range(n):
26                  if dfs(i, j, 0):
27                      return True
28          return False
29

```

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/>

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

森林的带度数层次序列存储

🔗 <http://cs101.openjudge.cn/practice/07161/>

```
1  from collections import deque
2
3  class Node:
4      def __init__(self, val):
5          self.val = val
6          self.children = []
7
8  def build_tree(tokens):
9      """
10     根据带度数的层次序列恢复一棵树
11     tokens: [C, 3, E, 3, F, 0, ...]
12     """
13     idx = 0
14     root = Node(tokens[idx])
15     root_degree = int(tokens[idx + 1])
16     idx += 2
17
18     queue = deque()
19     queue.append((root, root_degree))
20
21     while queue:
22         node, degree = queue.popleft()
23         for _ in range(degree):
24             child_val = tokens[idx]
25             child_degree = int(tokens[idx + 1])
26             idx += 2
27
28             child = Node(child_val)
29             node.children.append(child)
30             queue.append((child, child_degree))
31
32     return root
33
```

遍历树

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

```
1 def solve():
2     n = int(input().strip())
3
4     children = {}
5     nodes = set()
6     child_nodes = set()
7
8     # 建立树结构
9     for _ in range(n):
10         parts = list(map(int, input().split()))
11         u = parts[0]
12         nodes.add(u)
13         if len(parts) > 1:
14             children[u] = parts[1:]
15             for v in parts[1:]:
16                 child_nodes.add(v)
17                 nodes.add(v)
18         else:
19             children[u] = []
20
21     # 找根节点
22     root = (nodes - child_nodes).pop()
23
24     result = []
25
26     # DFS 按题目规则遍历
27     def dfs(u):
28         group = [u] + children[u]
29         group.sort()
30
31         for x in group:
32             if x == u:
33                 result.append(u)
```

从根节点到叶节点数字之和

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

```
1 class Solution:
2     def sumNumbers(self, root):
3         def dfs(node, num):
4             if not node:
5                 return 0
6             num = num * 10 + node.val
7             if not node.left and not node.right:
8                 return num
9             return dfs(node.left, num) + dfs(node.right, num)
10        return dfs(root, 0)
```

括号嵌套树

🔗 <http://cs101.openjudge.cn/practice/24729/>

```
1  def solve():
2      s = input().strip()
3      n = len(s)
4      i = 0
5
6      preorder = []
7      postorder = []
8
9      def parse():
10         nonlocal i
11
12         # 当前一定是一个结点
13         root = s[i]
14         preorder.append(root)
15         i += 1
16
17         # 如果有子树
18         if i < n and s[i] == '(':
19             i += 1    # 跳过 '('
20             while True:
21                 parse()  # 解析子树
22                 if s[i] == ',':
23                     i += 1    # 跳过 ','
24                 elif s[i] == ')':
25                     i += 1    # 跳过 ')'
26                     break
27
28         postorder.append(root)
29
30     parse()
31
32     print("".join(preorder))
33     print("".join(postorder))
```

文件结构图

[🔗 http://cs101.openjudge.cn/practice/02775/](http://cs101.openjudge.cn/practice/02775/)

```
1  class Node:
2      def __init__(self, name, is_dir):
3          self.name = name
4          self.is_dir = is_dir
5          self.dirs = []      # 子目录
6          self.files = []     # 文件
7
8
9  def print_tree(node, depth):
10     indent = "|"      * depth
11
12     # 先打印目录
13     for d in node.dirs:
14         print(f"{indent}{d.name}")
15         print_tree(d, depth + 1)
16
17     # 再打印文件 (按字典序)
18     for f in sorted(node.files):
19         print(f"{indent}{f}")
20
21
22 def main():
23     data_set = 1
24
25     while True:
26         root = Node("ROOT", True)
27         stack = [root]
28
29         while True:
30             try:
31                 line = input().strip()
32             except EOFError:
33                 return
```

Assignment 9