

数据结构与算法B 作业1

1. 题目

E27653: Fraction类

[<http://cs101.openjudge.cn/pctbook/E27653/>]

请练习用OOP方式实现。

思路：作为定义类的初学者，几乎是照搬了参考学习资料的网站上的内容和代码。十分惭愧。

代码：

```
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)
```

#50012118提交状态

[查看](#) [提交](#) [统计](#) [提问](#)

状态: Accepted

源代码

```
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)
```

基本信息

#: 50012118
题目: E27653
提交人: 22n2200011816(略约横溪)
内存: 4088kB
时间: 29ms
语言: Python3
提交时间: 2025-09-16 16:50:10

M1760.袋子里最少数目的球

binary search, <https://leetcode.cn/problems/minimum-limit-of-balls-in-a-bag/>

思路：想到了“操作多少次，就是多出了多少个袋子”，于是用二分查找的方式（当然看了提示）找到每个袋子可以有的最大球数来match袋子数量，最后debug（使用AI帮忙找了些测试样例）后也是AC了（当然代码十分不简洁，用的时间和空间也令人汗颜QWQ）。

代码：

```
from math import ceil

class Solution:
    def minimumSize(self, nums: List[int], maxOperations: int) -> int:
        max_num_of_bag = len(nums) + maxOperations
        left = 0 #使用二分查找获得合适的每个小袋的最高球数
        right = max(nums)
        while left <= right:
            mid = left + ceil((right - left)/2)
```

```

print(mid)
sum_num_of_bag = 0
num_of_bag = []
for i in range(len(nums)):
    num_of_bag.append(ceil(nums[i]/mid))
    sum_num_of_bag += ceil(nums[i]/mid)
if sum_num_of_bag > max_num_of_bag:
    left = mid + 1
elif sum_num_of_bag < max_num_of_bag:
    right = mid - 1
    if right == 0:
        return max([ceil(a / b) for a, b in zip(nums,
num_of_bag)])
    else:
        return max([ceil(a / b) for a, b in zip(nums, num_of_bag)])
return left

```

通过 61 / 61 个通过的测试用例

Happy l3sefod 提交于 2025.09.16 20:56

官方题解

写题解



开学限时福利

教育优惠 7 折获 365 天 Plus 会员，参与活动额外加送 180 天。

执行用时分布

1415 ms | 击败 5.04%

复杂度分析

消耗内存分布

32.34 MB | 击败 5.04%



```

1 from math import ceil
2
3
4 class Solution:
5     def minimumSize(self, nums: list[int], maxOperations: int) -> int:
6         max_num_of_bag = len(nums) + maxOperations
7         left = 0
8         right = max(nums)
9         while left <= right:
10             mid = left + ceil((right - left)/2)
11             print(mid)
12             sum_num_of_bag = 0
13             num_of_bag = []
14             for i in range(len(nums)):
15                 num_of_bag.append(ceil(nums[i]/mid))
16                 sum_num_of_bag += ceil(nums[i]/mid)
17             if sum_num_of_bag > max_num_of_bag:
18                 left = mid + 1
19             elif sum_num_of_bag < max_num_of_bag:
20                 right = mid - 1
21             if right == 0:
22                 return max([ceil(a / b) for a, b in zip(nums, num_of_bag)])
23             else:
24                 return max([ceil(a / b) for a, b in zip(nums, num_of_bag)])
25         return left
26

```

M04135: 月度开销

binary search, <http://cs101.openjudge.cn/pctbook/M04135/>

思路：考虑了使用二分查找的方法，设定上界和下界，使用mid作为cost，检查这个cost所需的月份数，最后查找到最小的cost，查找终止。

但是，在作答的过程中，一开始是将遍历的过程中记录的每个Month的cost的最大值作为答案输出，出现了答案符合月份要求，但是大于最优答案的情况，导致WA。后续在AI的帮助下成功地改正代码，将查找后的下界作为答案输出，最后才成功AC。

代码：

```

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)

```

状态: Accepted

源代码

```
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
            if month_needed <= month:
                right = mid - 1
            elif month_needed > month:
                left = mid + 1
        return left

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)
```

基本信息

#: 50027589
题目: M04135
提交人: 22n2200011816(略衍横溪)
内存: 7972kB
时间: 474ms
语言: Python3
提交时间: 2025-09-17 17:32:31

M27300: 模型整理

sortings, AI, <http://cs101.openjudge.cn/pctbook/M27300/>

思路：一道分类题目。首先对输入进行处理，将输入的文本分成名称 大小 单位 三部分。随后创建字典套字典，第一个键为模型名称，第二个键为单位，预留模型大小的列表，将大小添加至列表并进行排序，最后按要求输出即可。

代码：

```
class Solution:
    def model_sort(self, nums, model_list):
        model_list.sort(key = lambda name: name[0])
        model_dict = dict()
        # 创建字典套字典，第一个键为模型名称，第二个键为单位，预留模型大小的列表
        for j in range(nums):
            if model_list[j][0] not in model_dict.keys():
                model_dict[model_list[j][0]] = {'M': [], 'B': []}
            size = 0
            # 将模型大小归并到对应的模型和单位的字典内，储存为列表
            if float(model_list[j][1]) % 1 == 0:
```

```

        size = int(model_list[j][1])
    else:
        size = float(model_list[j][1])
    model_dict[model_list[j][0]][model_list[j][-1]].append(size)
# 按照要求输出文本
for key1 in model_dict.keys():
    str_result = ''
    for key2 in model_dict[key1].keys():
        model_dict[key1][key2].sort()
        for k in range(len(model_dict[key1][key2])):
            str_result += f'{model_dict[key1][key2][k]}{key2}, '

    print(f'{key1}: ' + str_result[:-2])

if __name__ == "__main__":
    solution = Solution()
    nums1 = int(input())
    model_list1 = []
    for i in range(nums1): # 将输入的模型进行处理，分成名称 大小 单位 三部分
        model = input().split('-')
        model.append(model[-1][-1])
        model[-2] = model[-2][:1]
        model_list1.append(model)
    solution.model_sort(nums1, model_list1)

```

状态: Accepted

源代码

```
class Solution:
    def model_sort(self, nums, model_list):
        model_list.sort(key = lambda name: name[0])
        model_dict = dict()
        for j in range(nums):
            if model_list[j][0] not in model_dict.keys():
                model_dict[model_list[j][0]] = {'M': [], 'B': []}
                size = 0
            if float(model_list[j][1]) % 1 == 0:
                size = int(model_list[j][1])
            else:
                size = float(model_list[j][1])
            model_dict[model_list[j][0]][model_list[j][-1]].append(size)
        for key1 in model_dict.keys():
            str_result = ''
            for key2 in model_dict[key1].keys():
                model_dict[key1][key2].sort()
                for k in range(len(model_dict[key1][key2])):
                    str_result += f'{model_dict[key1][key2][k]} {key2}, '
            print(f'{key1}: ' + str_result[:-2])

if __name__ == "__main__":
    solution = Solution()
    nums1 = int(input())
    model_list1 = []
    for i in range(nums1):
        model = input().split('-')
        model.append(model[-1][-1])
        model[-2] = model[-2][:-1]
        model_list1.append(model)
    solution.model_sort(nums1, model_list1)
```

基本信息

#: 50030491
题目: M27300
提交人: 22n2200011816(略约横溪)
内存: 3696kB
时间: 22ms
语言: Python3
提交时间: 2025-09-17 19:49:59

Q5. 熟悉云虚拟机Linux环境与大语言模型（LLM）本地部署

本项目包括两个任务：

1) 通过云虚拟机（如 <https://clab.pku.edu.cn/> 提供的资源）熟悉Linux系统操作环境：

项目进展：目前成功在 <https://clab.pku.edu.cn/>部署云虚拟机，并可以通过本地电脑顺利访问：

安装最新的 PowerShell，了解新功能和改进！ <https://aka.ms/PSWindows>

```
PS C:\Users\48144> ssh rocky@10.129.245.12
Last login: Wed Sep 17 12:57:18 2025 from 10.129.215.184
[rocky@xiweil ~]$ |
```

目前，项目停留在week1中教的在云虚拟机上访问校内网关：

```
[rocky@xiweil ~]$ python3 login.py
请输入用户名: 2200011816
请输入密码:
{"succ":"","ver":"1.1","FIXRATE":"YES","FR_TYPE":"","FR_DESC_CN":"不限时间","FR_DESC_EN":"unlimited","SCOPE":"international","DEFICIT":"","FR_TIME_CN":"","FR_TIME_EN":"Unlimited","CONNECTIONS":"4","BALANCE_CN":"10.0","BALANCE_EN":"10.0","IP":"10.129.245.12"}
[rocky@xiweil ~]$ |
```

以及从硬盘上划分了6个G的硬盘空间用于模拟内存（但是当时没有截图）。后续使用虚拟机来作为一个offlinejudge，用于数据的io。

2) 完成大语言模型（LLM）的本地部署与功能测试：

在这一部分，我使用LM Studio，在本地部署了LLM模型。由于我的显卡是3050Laptop，只有可怜的4个G的显存，Qwen 34B的模型可以流畅地运行，生成很快；8B的模型也凑合（每秒8-9字左右）；而14 B的模型几乎不可用（经过测试，每秒输出3.5个字，几乎不可用）。

因此，接下来的题目我都使用Qwen3 4B测试。结果显示，Qwen3 4B对相对简单的题目都轻松AC了，

1. 04123马走日：


Note

非常好！这是一个经典的“**骑士遍历棋盘**”问题，属于 **回溯算法** 的典型应用。

问题理解

我们要求：

给定一个 $n \times m$ 的棋盘，马从初始位置 (x, y) 出发，按照中国象棋“马”的走法（日字形）移动，**不能重复访问同一个格子**，问：马能走完所有格子（即访问所有 $n \times m$ 个格子）的**不同路径总数**是多少？


 注意：

- “马”走“日”字：8 种可能移动方向（如：上下左右 + 对角）
- 不能重复访问格子
- 走完所有格子才算一次“遍历”
- 每条路径是**从起点开始**，走完所有格子，路径不同就算不同（顺序不同算不同）

举例说明

比如 $n=3, m=3, x=0, y=0$

这是一个 3×3 棋盘，马从左上角出发，问有多少种方式走完所有格子。

 注意：在小棋盘上，可能根本无法走完所有格子，结果为 0。

✓ 解法：深度优先搜索（DFS） + 回溯

我们使用 **DFS + 访问标记数组** 来枚举所有可能的路径。

步骤：

1. 定义马的 8 种移动方向 ($\Delta x, \Delta y$)
2. 从起点开始 DFS，每次尝试走一个合法方向
3. 访问过的格子用一个布尔数组记录
4. 当访问了 $n \times m$ 个格子时，计数加 1
5. 回溯恢复状态，继续尝试其他路径

✓ Python 代码实现

```
def solve():
    # 马的8个移动方向（日字形）
    directions = [
        (-2, -1), (-2, 1), (-1, -2), (-1, 2),
        (1, -2), (1, 2), (2, -1), (2, 1)
    ]

    T = int(input())

    for _ in range(T):
        n, m, x, y = map(int, input().split())

        # 总格子数
        total_cells = n * m

        # 用于记录是否访问过某个格子
        visited = [[False] * m for _ in range(n)]

        # 计算路径总数
        count = 0

        def dfs(row, col, steps):
            nonlocal count
            # 如果已经走完所有格子
            if steps == total_cells:
                count += 1
                return

            for dx, dy in directions:
                nr, nc = row + dx, col + dy
                if 0 <= nr < n and 0 <= nc < m and not visited[nr][nc]:
                    visited[nr][nc] = True
                    dfs(nr, nc, steps + 1)
                    visited[nr][nc] = False
```

```

        # 尝试8个方向
        for dx, dy in directions:
            new_row = row + dx
            new_col = col + dy

            # 判断是否在棋盘范围内
            if 0 <= new_row < n and 0 <= new_col < m:
                # 如果没访问过
                if not visited[new_row][new_col]:
                    visited[new_row][new_col] = True
                    dfs(new_row, new_col, steps + 1)
                    visited[new_row][new_col] = False # 回溯

        # 从起点开始
        visited[x][y] = True
        dfs(x, y, 1)

    print(count)

solve()

```

✓ 输入输出示例

输入：

```

2
3 3 0 0
3 3 1 1

```

输出：

```

0
0

```

解释：3×3 棋盘上，从任意位置出发，马无法走完所有格子，所以路径数为 0。

✓ 事实上，在 3×3 棋盘上，马无法走完所有格子，这是已知结论。



特别说明

- 本题中，“**路径**”是指从起点开始，经过每个格子一次，最终走完所有格子的路径。
- 该问题类似于“骑士遍历”或“哈密顿路径”问题，属于 **NP-hard** 问题，在**小棋盘上** ($n < 10, m < 10$) 可用 DFS 回溯解决。
- 由于棋盘小，时间复杂度是可接受的（最多 100 个格子，最多 100! 种路径，但实际通过剪枝极大减少）。



注意事项

- 题目要求不能重复访问格子 → 使用 `visited` 数组
- 所有路径必须从 `(x,y)` 开始
- 每个路径的访问顺序不同，就算不同路径（题目要求是“途径总数”）



扩展建议（可选）

如果你想支持更复杂的判断（比如路径是否合法、是否重复），或者想优化性能，可以：

- 添加剪枝（比如：如果当前步数超过总数，直接返回）
- 使用位运算记录状态（适用于小棋盘）
- 改为状态压缩（但本题 $n, m < 10$ ，直接 DFS 更简单）



总结

项目	说明
算法	DFS + 回溯
时间复杂度	$O(8^{(n \times m)})$ ，但实际受剪枝限制
空间复杂度	$O(n \times m)$
适用范围	$n, m < 10$



你只需要把上面的代码复制到你的程序中，即可运行！

如果你希望我再补充一个 **测试用例** 或者 **解释某个具体路径**，也可以告诉我！ 😊

需要我帮你写一个更优化的版本（比如带剪枝、预处理）吗？欢迎继续提问！

结果：也是顺利AC了：

#50210804提交状态

状态: Accepted

源代码

```
def solve():
    # 马的8个移动方向（日字形）
    directions = [
        (-2, -1), (-2, 1), (-1, -2), (-1, 2),
        (1, -2), (1, 2), (2, -1), (2, 1)
    ]

    T = int(input())
```

2.反转链表：

[206. 反转链表 - 力扣 \(LeetCode\)](#)

Note

要反转一个单链表，可以使用迭代的方式，通过维护三个指针（`prev`、`current` 和 `next_node`）来实现。以下是详细的思路和代码实现：

思路

1. **初始化指针：** `prev` 初始化为 `None`，`current` 初始化为链表的头节点 `head`。
2. **遍历链表：** 在循环中，将 `current` 的 `next` 指针指向 `prev`，然后将 `prev` 指向 `current`，最后将 `current` 移动到 `next_node`。
3. **终止条件：** 当 `current` 为 `None` 时，循环结束，此时 `prev` 即为反转后链表的头节点。

代码实现

```
class Solution:
    def reverseList(self, head: ListNode) -> ListNode:
        prev = None
```

```
current = head
while current:
    next_node = current.next
    current.next = prev
    prev = current
    current = next_node
return prev
```

解释

- `prev`：用于保存当前节点的前一个节点，初始为 `None`。
- `current`：用于遍历链表的当前节点，初始为链表头节点。
- `next_node`：保存当前节点的下一个节点，以便在反转后继续遍历。
- **循环过程**：每次将当前节点的 `next` 指向 `prev`（即反转链表），然后将 `prev` 更新为当前节点，`current` 移动到下一个节点。
- **返回结果**：当 `current` 为 `None` 时，`prev` 指向新的链表头节点，即反转后的链表头。

示例验证

- **示例 1**：输入 `[1, 2, 3, 4, 5]` → 输出 `[5, 4, 3, 2, 1]`
- **示例 2**：输入 `[1, 2]` → 输出 `[2, 1]`
- **示例 3**：输入 `[]` → 输出 `[]`

此方法时间复杂度为 $O(n)$ ，其中 n 为链表节点数；空间复杂度为 $O(1)$ ，仅使用了常数级额外空间。

最终答案

```
class Solution:
    def reverseList(self, head: ListNode) -> ListNode:
        prev = None
        current = head
        while current:
            next_node = current.next
            current.next = prev
            prev = current
            current = next_node
        return prev
```

← 全部提交记录



通过 28 / 28 个通过的测试用例 用时: 2 hrs 42 m 45 s

Happy l3osefod 提交于 2025.10.02 16:46

官方题解

写题解



面向在校学生的专享特惠

完成认证享 7 折 Plus 会员，享受更多学业及职业成长帮助



执行用时分布



0 ms | 击败 100.00%

复杂度分析

消耗内存分布

18.24 MB | 击败 86.80%

100%

90.62% 的用户使用了类似解法 Runtime: 0 ms

50%

0%

1ms

2ms

3ms

4ms

1ms

2ms

3ms

4ms

3.有向图中的最大颜色值：

<https://leetcode.cn/problems/largest-color-value-in-a-directed-graph/>

在完成这道题目的时候，一开始我用了和上面反转链表的题目的同一个窗口，结果思考了半天，又回答了反转链表的答案，显示这个参数量的模型对上下文理解可能还有欠缺：

多 颜色的节点数目。

请你返回给定图中有效路径里面的 最大颜色值。如果图中含有环，请返回 -1。

示例 1:

输入: colors = "abaca", edges = [[0,1],[0,2],[2,3],[3,4]]

qwen/qwen3-4b-thinking-2507

> Thought for 2 minutes 20 seconds

要反转单链表，可以使用迭代法。以下是详细的思路 and 代码：

思路

1. **初始化指针**：使用两个指针 `prev`（前一个节点）和 `curr`（当前节点）。
2. **遍历链表**：在遍历链表的过程中，将当前节点的 `next` 指针指向前一个节点（`prev`），然后将 `prev` 指向当前节点（`curr`），并将 `curr` 移动到下一个节点。
3. **返回反转后的头节点**：当链表遍历结束时，`prev` 指向新的头节点（原链表的最后一个节点）。

代码实现


python

Copy

后来重新开了个窗口，又生成了一遍，在思考的过程中出现了大量的英文，同时输出的结果不出所料地回答错误了：

```
AttributeError: 'Solution' object has no attribute '
largestPathValue'
~~~~~
    ret = Solution().largestPathValue(param_1, param
_2)
Line 72 in _driver (Solution.py)
    _driver()
Line 83 in <module> (Solution.py)
```

最后执行的输入

添加到测试用例 

```
colors =
"abaca"
```

```
edges =
[[0,1],[0,2],[2,3],[3,4]]
```

代码 | Python3

 复杂度分析

```
from typing import List
import collections

class Solution:
    def maxColorValue(self, colors: str, edges:
        n = len(colors)
        adj = [[] for _ in range(n)]
        in_degree = [0] * n
```

 查看更多

这个参数的AI在解决HARD问题上确实不大行。

4.18250:冰阔落 I

[OpenJudge - 18250:冰阔落 I](#)

做这道题目时，AI也是轻松AC了，但是很神奇地思考到一半变成了英文输出emm：

Note

To solve this problem, we need to simulate a series of operations where we merge cups of ice tea based on given instructions. The goal is to determine for each operation whether the specified cups are already in the same cup or if we need to merge them, and finally, count how many cups still contain ice tea after all operations.

Approach

1. **Problem Analysis:** The problem involves processing multiple test cases where each test case starts with `n` cups of ice tea and `m` operations. Each operation specifies two cups (by their initial numbers) and requires checking if they are already in the same cup. If not, we merge all cups from the second cup into the first cup.
2. **Union-Find Data Structure:** We use the Union-Find (Disjoint Set Union, DSU) data structure with path compression to efficiently manage and merge the cups. This helps in quickly determining if two cups are in the same set and merging them in near-constant time per operation.
3. **Processing Operations:** For each operation:
 - Find the root (representative) of both cups.
 - If the roots are the same, output "Yes".
 - Otherwise, merge the root of the second cup into the root of the first cup and output "No".
4. **Result Compilation:** After processing all operations, count the number of cups that remain as roots (i.e., cups that have not been merged into any other cup) and output them in increasing order.

Solution Code

```
import sys

def main():
    data = sys.stdin.read().split()
    if not data:
        return
    index = 0
    output_lines = []
    while index < len(data):
        n = int(data[index])
        m = int(data[index + 1])
        index += 2
        parent = list(range(n + 1))

        def find(x):
            root = x
            while root != parent[root]:
                root = parent[root]
            while x != root:
                next_node = parent[x]
                parent[x] = root
                x = next_node
            return root
```

```

        x = next_node
    return root

for _ in range(m):
    x = int(data[index])
    y = int(data[index + 1])
    index += 2
    rx = find(x)
    ry = find(y)
    if rx == ry:
        output_lines.append("Yes")
    else:
        parent[ry] = rx
        output_lines.append("No")

roots = []
for i in range(1, n + 1):
    if parent[i] == i:
        roots.append(i)
output_lines.append(str(len(roots)))
output_lines.append(" ".join(map(str, roots)))

print("\n".join(output_lines))

if __name__ == '__main__':
    main()

```

Explanation

1. **Reading Input:** The input is read all at once for efficiency, especially given the constraints (up to 50,000 operations per test case).
2. **Union-Find Initialization:** For each test case, we initialize the parent array where each cup initially points to itself.
3. **Find Function:** The `find` function uses path compression to flatten the structure of the tree, ensuring future queries are faster.
4. **Processing Operations:** For each operation:
 - The roots of the two specified cups are found.
 - If they are the same, "Yes" is added to the output; otherwise, the root of the second cup is merged into the root of the first cup, and "No" is added to the output.
5. **Result Compilation:** After all operations, we count the cups that remain as roots (i.e., cups that have not been merged) and output the count followed by the sorted list of these cup numbers.

6. **Output:** The results for all test cases are printed in the required format, with each test case's results spanning `m` lines of "Yes"/"No", followed by two additional lines.

This approach efficiently handles the problem using the Union-Find data structure with path compression, ensuring optimal performance even for large inputs. The solution processes each test case in near-linear time relative to the number of operations.

#50211465提交状态

状态: Accepted

源代码

```
import sys

def main():
    data = sys.stdin.read().split()
    if not data:
        return
    index = 0
    output_lines = []
    while index < len(data):
        n = int(data[index])
        m = int(data[index + 1])
        index += 2
        parent = list(range(n + 1))

        def find(x):
            root = x
            while root != parent[root]:
                root = parent[root]
            while x != root:
                next_node = parent[x]
                parent[x] = root
                x = next_node
            return root

        for _ in range(m):
            u = int(data[index])
            v = int(data[index + 1])
            index += 2
            ru = find(u)
            rv = find(v)
            if ru != rv:
                parent[ru] = rv

    for _ in range(len(output_lines)):
        print("Yes")
    for _ in range(len(output_lines)):
        print("No")
```

Q6. 阅读《Build a Large Language Model (From Scratch)》第一章

日期: /
Large Language model

LLM: 利用神经网络对大量文本数据进行预训练

1.1 什么是 LLM? → 对文本的上下文进行深入理解 { 翻译 → 应用推广
分析
问答

Large { 参数量
训练集大小
用自然语言顺序理解上下文并生成下一段
Transformer Architecture (Transformer 架构)

特点: 通过数据集进行判断而
非固定程序 大量标注

训练数据集: 需要人去标注对数据集进行标注
{ 预训练: 无需人工进行特征提取

1.2. LLM 之应用: 分析 文本 { 翻译、生成文本
聊天机器人
筛选、总结大量文献

1.3. 构建/使用 LLM:

理由: ① 理解 LLM, 微调 LLM

② 专用 LLM > 通用 LLM

③ 隐私、自主权

日期: /

方法: 预训练: 多任务数据构建基本模型

“预训练” → 无监督 LLM

获得自学习能力, 后续仅需少量数据来定
成任务

微调: 使用专项数据对模型进行
训练

指令微调 数据集要有指令
与答案对 *Instruction Pairs*
Answer Pairs

监督微调 数据集要有标签标注

1.4. Transformer Architecture

两部分:

Encoder

文本输入

Self-attention Mechanism

Decoder

输出文本

→ 关键 自注意力机制 → 保证模型可以注意文本间各部分联系

→ 不可忽略: BERT → 用于对掩码/随机词的输出

GPT → 用于对文本的生成

* Transformer Architecture ≠ LLM

可用于计算机视觉

有基于层积/递归架构的 LLM

日期:

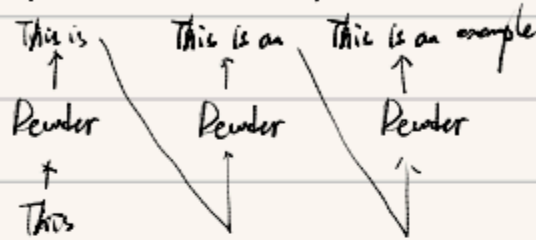
1.7.11号)

★ GPT Architecture: 自我学习不需要有数据收集标签, 而是将下一个生成词作为标签

人类输入模型:

补下一个词

只要有前缀



将某次模型的输出作为下一次的输入

★ 没有标签也下的翻译器 → 多语言训练中的涌现

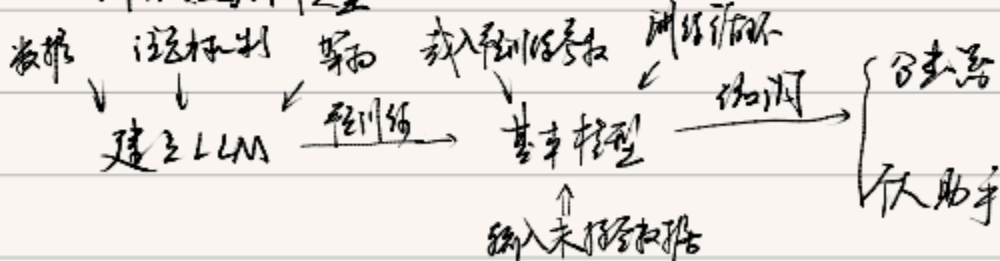
1.5. 模型数据集

训练 BERT/GPT 的数据集有上亿词 → 小数据集

用于生成大量文本工作

下的生成模型工作

1.6. 训练 LLM 模型:



2. 学习总结和个人收获:

前面的四道题目作为热身题, 个人确实感觉有点困难, 做起来花了不少的时间。同时也学了一些类的写法, 重温了二分查找。后面在配置云虚拟机的时候, 由于RSA密钥在导出的时候出现了一些小问题, 用了一些时间才在本体连接上虚拟机, 总体也不容易。AI由于有图形化的软件, 配置蒸馏模型都是傻瓜式操作, 没有遇上什么困难, 总体玩AI的过程还是很好玩的, 但是由于电脑配

置不是很好，只能用4B和8B的模型了QWQ，有钱真好。《Build a Large Language Model》这本书也认真读了第一章，目前感觉讲得挺清晰，理解上没有困难。