2024.9.14作业

复习题

- 复习题第四题

  机器学习、市场调研、健康数据分析、风险投资分析

- 复习题第五题

  分治：分治的核心思想是分而治之，将一个复杂的问题分成若干个相对独立、相似的子问题，然后各个子问题再继续划分，直到分解成最小、最简单的问题。最终，利用这些子问题的解合并得到整个问题的解。

  递归：递归的核心思想是自我重复。在递归中，函数通过调用自身解决问题。每次调用会使问题逐步简化，直到达到一个可以直接解决的最基本情况

践习题

1.
   1. 由667个3组成的整数数组
   2. 检查是否大于4，然后发现大于4的数都可以已非常相似的方法拆分
   3. ```
      #已知数学有基本不等式，在每个数字接近相等的时候，乘积最大，所以我们要尽可能拆分成2，3；同时多拆分出3会更大
      def max_multi_partition(n):
          if n <= 4:
              return [n]
          result = []
          while n > 4:
              result.append(3)
              n -= 3
          result.append(n)
          return result

      n = int(input())
      result = max_multi_partition(n)
      print(result)
      ```

2. 是的

3. ```
   from collections import deque

   def check_state(state):
       human, wolf, sheep, cabbage = state
       if (wolf == sheep and human != wolf) or (sheep == cabbage and human != sheep):
           return False
       return True
   def find_next_states(state):
       next_states = []
       human, wolf, sheep, cabbage = state
       moves = [
           (1, 0, 0, 0),
           (1, 1, 0, 0),
   ```

```
        (1, 0, 1, 0),
        (1, 0, 0, 1)
    ]

    for move in moves:
        new_state = (
            human ^ move[0],
            wolf ^ move[1],
            sheep ^ move[2],
            cabbage ^ move[3]
        )
        if check_state(new_state):
            next_states.append(new_state)
    return next_states

def bfs():
    start_state = (0, 0, 0, 0)
    goal_state = (1, 1, 1, 1)

    queue = deque([(start_state, [start_state])])
    visited = set([start_state])

    while queue:
        current_state, path = queue.popleft()
        if current_state == goal_state:
            return path

        for next_state in find_next_states(current_state):
            if next_state not in visited:
                visited.add(next_state)
                queue.append((next_state, path + [next_state]))
    return None
solution = bfs()
if solution:
    print("路径如下：")
    for step in solution:
        print(step)
else:
    print("没有找到解决方案！")
```

4.
```
def find_square_root(num, precision=0.0001):
    # 初始猜测值
    guess = 1.0
    g1 = 0.00001  # 步长

    while True:
        if abs(guess * guess - num) < precision:
            return guess

        # 如果平方小于 num，则增加 guess 值
        if guess * guess < num:
            guess += g1
        else:
            # 如果平方超过 num，减少步长的大小并调整 guess
            step /= 10
            guess -= g1
```

```
num = 2
square_root = find_square_root(num)
print(f"√{num} 的近似值是: {square_root}")
```