**囚徒问题实验报告**

1. 算法说明

import random

from typing import List

def generate\_random\_boxes(n: int) -> List[int]:

    """生成随机排列的盒子内容"""

    numbers = list(range(1, n+1))

    random.shuffle(numbers)

    return numbers

def strategy\_random(boxes: List[int], prisoner\_num: int, max\_attempts: int) -> bool:

    """随机策略：随机选择盒子"""

    attempts = random.sample(range(len(boxes)), max\_attempts)

    for attempt in attempts:

        if boxes[attempt] == prisoner\_num:

            return True

    return False

def strategy\_cycle(boxes: List[int], prisoner\_num: int, max\_attempts: int) -> bool:

    """循环策略：跟随盒子中的数字跳转"""

    current\_box = prisoner\_num - 1  # 转换为0-based索引

    for \_ in range(max\_attempts):

        if boxes[current\_box] == prisoner\_num:

            return True

        current\_box = boxes[current\_box] - 1  # 跳转到下一个盒子

    return False

def run\_simulation(n: int, k: int, strategy) -> bool:

    """运行一次模拟"""

    boxes = generate\_random\_boxes(n)

    for prisoner\_num in range(1, n+1):

        if not strategy(boxes, prisoner\_num, k):

            return False

    return True

def compare\_strategies(n: int = 100, k: int = 50, t: int = 10000):

    """比较两种策略的成功率"""

    random\_success = 0

    cycle\_success = 0

    for \_ in range(t):

        # 对两种策略使用相同的盒子排列

        boxes = generate\_random\_boxes(n)

        # 测试随机策略

        random\_failed = False

        for prisoner\_num in range(1, n+1):

            if not strategy\_random(boxes, prisoner\_num, k):

                random\_failed = True

                break

        if not random\_failed:

            random\_success += 1

        # 测试循环策略

        cycle\_failed = False

        for prisoner\_num in range(1, n+1):

            if not strategy\_cycle(boxes, prisoner\_num, k):

                cycle\_failed = True

                break

        if not cycle\_failed:

            cycle\_success += 1

    print(f"set: {n} prisoners, {k} tries per person, {t} simulation")

    print(f"random strategy succeed probability: {random\_success/t:.6f} ({random\_success}/{t})")

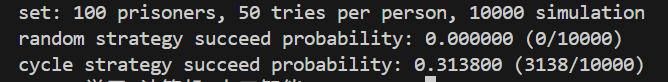
    print(f"c: {cycle\_success/t:.6f} ({cycle\_success}/{t})")

if \_\_name\_\_ == "\_\_main\_\_":

    # 默认参数: N=100, K=50, T=10000

    compare\_strategies(n=100, k=50, t=10000)

1. 实验结果



理论计算

对于n=100，k=50：

Psuccess≈1−≈1−(ln(100)−ln(50))≈1−ln2≈0.30685

约 30.685% 的成功率，远高于随机策略。

1. 优化思路

**向量化优化**：

1. **使用NumPy替代列表和循环**
2. **并行模拟**
3. **批量模拟囚犯行为**：在策略实现中尽量减少显式的 for 循环，借助布尔数组、索引数组等技巧一次性模拟多个囚犯的行为。
4. 图

图表, 条形图

AI 生成的内容可能不正确。

切换参数（n=50, K=25）后输出结果：

