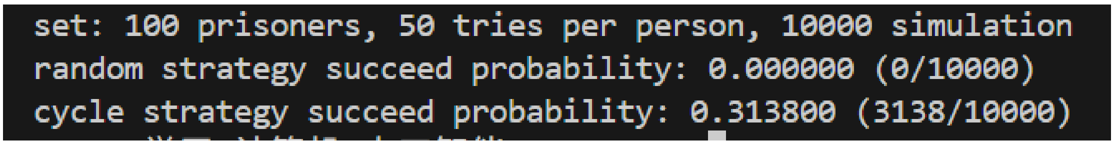
囚徒问题实验报告

一、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)
```

二、实验结果



```
set: 100 prisoners, 50 tries per person, 10000 simulation
random strategy succeed probability: 0.000000 (0/10000)
cycle strategy succeed probability: 0.313800 (3138/10000)
```

理论计算

对于 n=100，k=50：

$$P_{success} \approx 1-\sum_{l=51}^{100}\frac{1}{l} \approx 1-(\ln(100)-\ln(50)) \approx 1-\ln 2 \approx 0.30685$$

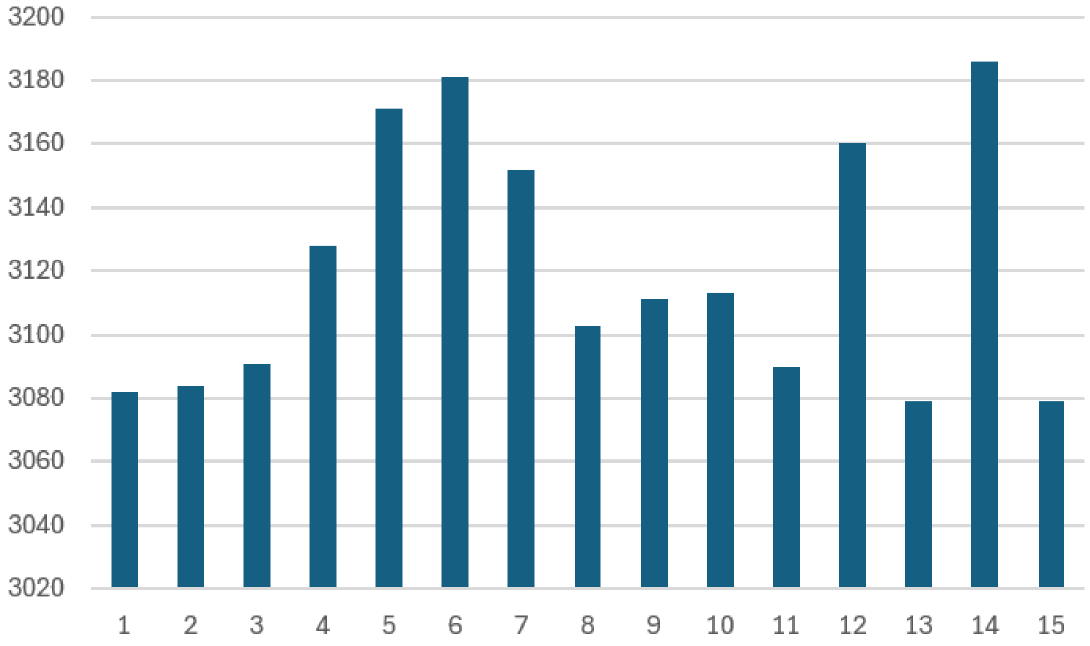
约 30.685% 的成功率，远高于随机策略。

三、优化思路

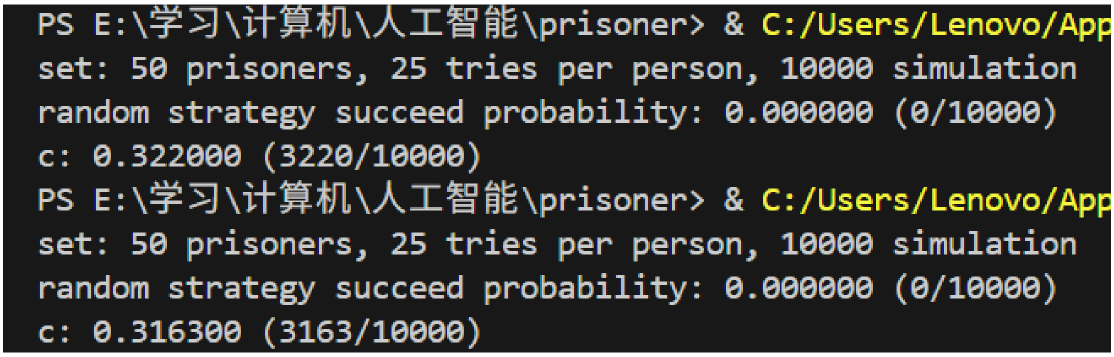
向量化优化：

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。

四、图



切换参数（n=50, K=25）后输出结果：



```
PS E:\学习\计算机\人工智能\prisoner> & C:/Users/Lenovo/App
set: 50 prisoners, 25 tries per person, 10000 simulation
random strategy succeed probability: 0.000000 (0/10000)
c: 0.322000 (3220/10000)
PS E:\学习\计算机\人工智能\prisoner> & C:/Users/Lenovo/App
set: 50 prisoners, 25 tries per person, 10000 simulation
random strategy succeed probability: 0.000000 (0/10000)
c: 0.316300 (3163/10000)
```