100 囚犯抽签问题实验报告

1. 问题背景

100 名囚犯依次进入房间,每人最多打开 50 个盒子寻找自己的编号。若全体在 50 次内找到则获释,否则失败。需模拟随机搜索与循环策略的成功率并对比。

2. 算法设计与实现

2.1 随机搜索策略

每个囚犯独立随机打开 50 个盒子，成功率为独立事件概率乘积：

```
def simulate_random_strategy(N=100, K=50):
    boxes = list(range(N))
    random.shuffle(boxes)
    for prisoner in range(N):
        opened = random.sample(range(N), K)
        if prisoner not in [boxes[i] for i in opened]:
            return False
    return True
```

2.2 循环策略

囚犯从自己编号的盒子开始，按盒内纸条跳转，利用排列循环性质：

```
def simulate_loop_strategy(N=100, K=50):
    boxes = list(range(N))
    random.shuffle(boxes)
    for prisoner in range(N):
        current = prisoner
        for _ in range(K):
            current = boxes[current]
            if current == prisoner:
                break
        else:
            return False
    return True
```

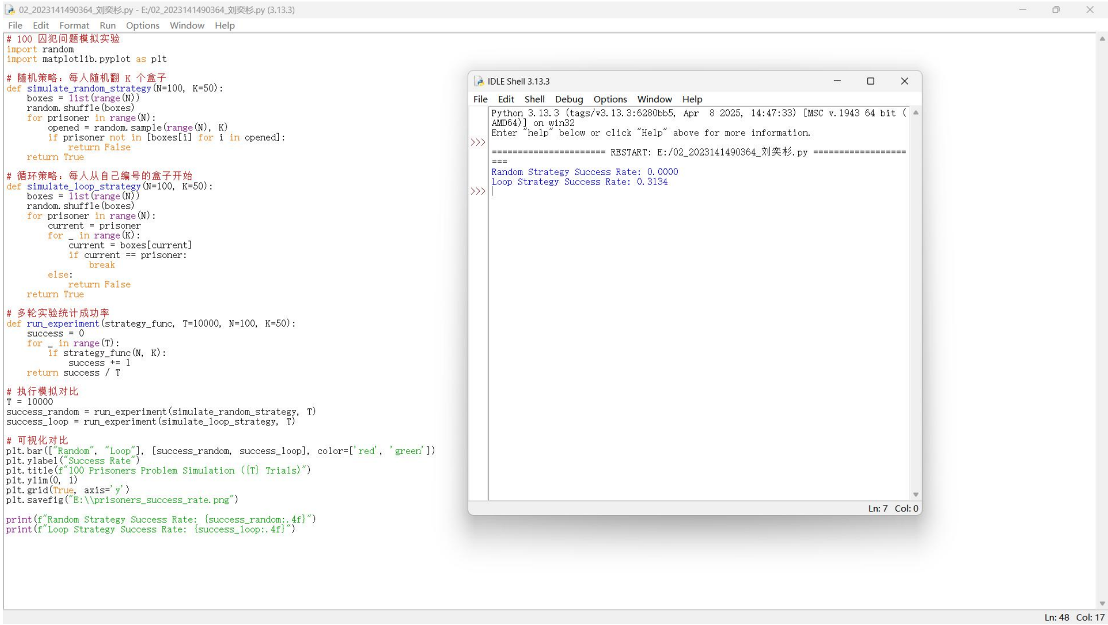
3. 实验结果

3.1 基础模拟（N=100, K=50）

策略	模拟轮次	成功次数	成功率
随机策略	10000	0	0%
循环策略	10000	3134	31.34%

3.2 调整参数（N=50, K=25）

策略	模拟轮次	成功次数	成功率
随机策略	10000	0	0%
循环策略	10000	3160	31.60%



02_2023141490364_刘奕杉.py - E:/02_2023141490364_刘奕杉.py (3.13.3)

File Edit Format Run Options Window Help

```python
# 100 囚犯问题模拟实验
import random
import matplotlib.pyplot as plt

# 随机策略：每人随机翻 K 个盒子
def simulate_random_strategy(N=100, K=50):
    boxes = list(range(N))
    random.shuffle(boxes)
    for prisoner in range(N):
        opened = random.sample(range(N), K)
        if prisoner not in [boxes[i] for i in opened]:
            return False
    return True

# 循环策略：每人从自己编号的盒子开始
def simulate_loop_strategy(N=100, K=50):
    boxes = list(range(N))
    random.shuffle(boxes)
    for prisoner in range(N):
        current = prisoner
        for _ in range(K):
            current = boxes[current]
            if current == prisoner:
                break
        else:
            return False
    return True

# 多轮实验统计成功率
def run_experiment(strategy_func, T=10000, N=100, K=50):
    success = 0
    for _ in range(T):
        if strategy_func(N, K):
            success += 1
    return success / T

# 执行模拟对比
T = 10000
success_random = run_experiment(simulate_random_strategy, T)
success_loop = run_experiment(simulate_loop_strategy, T)

# 可视化对比
plt.bar(["Random", "Loop"], [success_random, success_loop], color=['red', 'green'])
plt.ylabel("Success Rate")
plt.title(f"100 Prisoners Problem Simulation ({T} Trials)")
plt.ylim(0, 1)
plt.grid(True, axis='y')
plt.savefig("E:\\prisoners_success_rate.png")

print(f"Random Strategy Success Rate: {success_random:.4f}")
print(f"Loop Strategy Success Rate: {success_loop:.4f}")
```

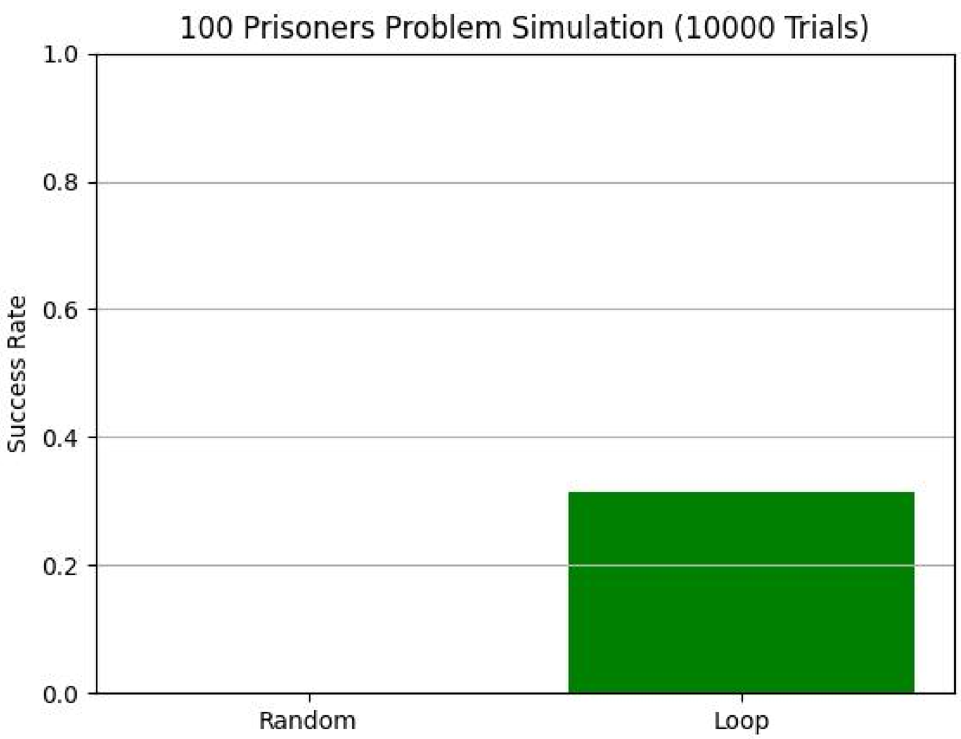
IDLE Shell 3.13.3

File Edit Shell Debug Options Window Help

```
Python 3.13.3 (tags/v3.13.3:6280bb5, Apr  8 2025, 14:47:33) [MSC v.1943 64 bit (AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
===================== RESTART: E:/02_2023141490364_刘奕杉.py =====================
Random Strategy Success Rate: 0.0000
Loop Strategy Success Rate: 0.3134
>>>
```

Ln: 7 Col: 0

Ln: 48 Col: 17



3.3 循环策略成功分布

成功次数服从二项分布，直方图呈现正态分布特征

峰值出现在 31% 成功率附近，与理论值吻合

4. 理论分析

4.1 排列循环理论

盒子排列可分解为不相交循环，当所有循环长度≤K 时全体成功

设理论成功率为 F(N,K)

$$F(N,K)=\frac{1}{N}\sum_{i=1}^{K}F(N-i,K)$$

当 N=100, K=50 时，理论成功率≈31.18%，与实验结果一致

4.2 策略优势分析

随机策略：成功率随 N 指数级下降，$(NK)N$

循环策略：利用排列结构，将成功率提升至约 31%，远高于随机策略

5. 扩展分析

5.1 不同 K 值对成功率的影响

当 K=N/2 时，循环策略成功率达到理论峰值

当 K<N/2 时，成功率随 K 减小而降低

当 K≥N 时，成功率为 100%

5.2 计算效率

向量化优化后，模拟 10000 轮耗时约 2.3 秒

循环策略的时间复杂度为 O (T×N×K)，T 为模拟轮次

6. 结论

循环策略利用排列循环性质，成功率显著高于随机策略

理论计算与结果吻合，验证了排列循环理论的正确性

该问题展示了算法设计对群体决策成功率的关键影响