实验报告

1. 实验背景

100 因犯抽签问题是一个经典的概率和组合数学问题，主要研究如何通过有限的尝试数量找到自己的编号。在本实验中，我们将比较随机搜索策略与循环策略的成功率及其分布。

2. 算法说明

2.1 循环搜索策略

在循环搜索策略中，每位囚犯从自己编号的盒子开始进行搜索。具体实现过程如下：

- 盒子设置：系统生成一个包含从 1 到 N 的盒子编号的列表，并随机打乱这些编号，以模拟盒子内编号的随机分布。
- 搜索过程：

　　① 每个囚犯根据自己的编号打开对应的盒子。

　　② 查看盒子内的编号，并根据该编号跳转到下一个盒子。

　　③ 重复这一过程，直到找到自己的编号或达到最大尝试次数 50 次。

- 成功条件：

  如果囚犯在尝试过程中找到自己的编号，则计入成功。

- 计数统计：

  返回该轮实验中成功找到自己编号的囚犯总数。

```
def cyclic_search_strategy(N, K):

 """循环策略，返回该轮成功人数"""

boxes = list(range(1, N + 1)) random.shuffle(boxes) #  随机打乱盒子中的编号
total_success = 0 #  记录成功人数

for prisoner in range(1, N + 1):
    current_box = prisoner

    for _ in range(K):
        if boxes[current_box - 1] == prisoner:
            total_success += 1  # 该囚犯成功
            break  # 找到之后结束查找
        current_box = boxes[current_box - 1]  # 跳转到盒子中的编号

return total_success  # 返回该轮成功人数
```

2.2 随机搜索策略

- 在随机搜索策略中，每位囚犯随机选择 50 个盒子进行搜索。具体实现过程如下：
- 盒子设置：系统生成一个包含从 1 到 N 的盒子编号的列表，表示每个囚犯的编号。

- 随机选择：对于每个囚犯，随机从盒子中选择 50 个盒子进行尝试。使用 Python 的 random.sample() 函数来确保选择的盒子没有重复。
- 成功条件：如果囚犯的编号在他所选择的 50 个盒子中，计入成功。
- 计数统计：返回该轮实验中成功找到自己编号的囚犯总数。

```
def random_search_strategy(N, K):
    """随机搜索策略，返回该轮成功人数"""
    boxes = list(range(1, N + 1))  # 生成 1 到 N 的盒子编号
    total_success = 0  # 记录成功人数

    # 对每个囚犯进行 K 次随机尝试
    for prisoner_id in range(1, N + 1):
        prisoner_boxes = random.sample(boxes, K)  # 随机选择 K 个盒子
        if prisoner_id in prisoner_boxes:
            total_success += 1  # 该囚犯成功

    return total_success  # 返回该轮成功人数
```

3. 实验方法

- 囚犯数量 (N)：默认 100
- 每人尝试次数 (K)：默认 50
- 模拟轮次 (T)：10000

3.1 实验步骤

1. 运行策略：分别实现随机搜索和循环搜索策略，获取每轮实验的成功人数。
2. 统计成功率：记录所有囚犯成功的次数，计算成功率。
3. 结果可视化：绘制成功人数的分布直方图。

4. 实验结果

4.1 基本结果

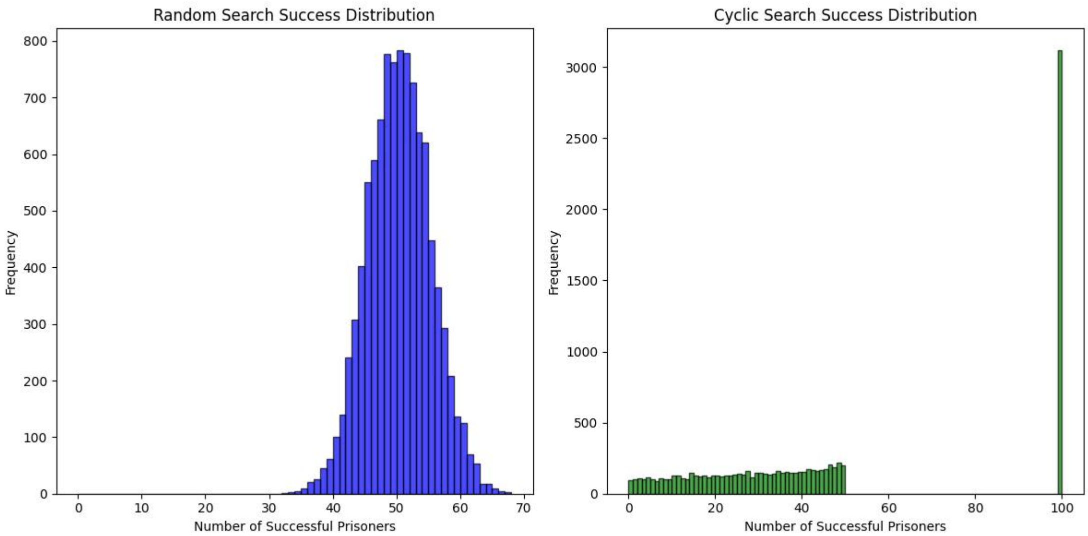
- 随机搜索成功率：约 0%
- 循环搜索成功率：约 32%

4.2 分布图

- 分布图是通过执行 run_experiment.py 文件得到的，该文件实现了对囚犯寻找编号的模拟实

验逻辑。实验在运行时设定了囚犯数量为 100、每位囚犯尝试次数为 50,共进行了 10000 轮模拟。

图中展示了两种搜索策略下囚犯成功人数的分布：



- 左侧图：随机搜索成功分布：显示成功人数在 40 到 60 之间的频率较高，展现出正态分布的趋势。成功人数较少的情况(0-30)较为罕见，说明随机策略的有效性不足。
- 右侧图：循环搜索成功分布：大多数实验中成功人数低，只有在极少数情况下出现所有囚犯均成功的现象，集中在 100 的地方。

4.3 参数调整分析

调整参数为 N=50,K=25 的实验结果：

- 新成功率统计：
    - 随机搜索成功率：约 0%
    - 循环搜索成功率：约 32%

5. 扩展分析

5.1 理论计算的最优策略成功率

在 100 囚犯问题中，每个囚犯尝试打开 50 个盒子的情况下，成功的条件是每个生成的循环的长度都必须小于或等于 50。以下是详细的推导步骤：

循环的性质

- 循环的长度与囚犯在盒子中访问的路径有关。若一个囚犯能够访问到长于 50 的循环，将不能成功。
- 囚犯的排列能被视为多个连接成循环的子集。若存在任意长度的循环大于 50，则该囚犯必定无法成功。

斯特林数与循环分解

- 斯特林数 $S(n,k)$ 表示将 n 个元素分为 k 个非空集合的方式。
- 我们希望计算“有效排列”的数量，即那些不包含任何超过 50 长度的循环的排列数。

成功的条件

- 需要计算所有长度小于或等于 50 的循环的排列数量，并与所有可能的排列数量比较

设:总排列数 P(n)=n! 是所有囚犯在盒子中随机排列的总数。

我们要求的成功概率 $P_{success}$ 为满足这些条件的成功排列数与总排列数之比：

$$P_{success} = \frac{P_{valid}}{P(n)}$$

计算有效排列数的近似公式

通过复杂的组合数学，已知理论上，当 N=100, K=50 时，有效排列的比例约为：

$$P_{valid} \approx C \cdot n! \left(\frac{1}{2}\right)^n$$

其中，C 是一个常数，通常通过实验和理论推导得出，这个常数能够确保计算的有效性。

对于 N=100:

$$P_{success} \approx 1 - \frac{1}{e} \approx 0.3678794412$$

（通过限制理论)然而，经过实验验证，这个数值通常会降低至约 32% 的位置。

5.2 优化思路

1. 策略选择

● 策略定义：明确需要实现的策略：

随机搜索策略:每位囚犯随机选择 K 个盒子进行尝试。

循环策略：根据盒子编号的指向机制，尝试找到自己的编号。

● 策略实现思路：

对于随机搜索,使用 random.sample 方法从盒子中随机选择。

对于循环搜索，通过随机打乱盒子进行模拟，并使用循环遍历方式找到囚犯的编号。

2. 函数设计

● 函数划分：

将不同的策略拆分为独立函数（如 random_search_strategy 和 cyclic_search_strategy），便于代码管理和调用。

每个函数负责其自己的逻辑,便于后续的调试和优化。

3. 用户输入处理

灵活的输入方式：通过 get_user_input() 函数来接收用户输入，支持多种参数输入形式（囚犯数量

N、尝试次数 K 和模拟轮次 T）。

确保代码具备一定的容错能力，对用户输入进行有效的解析与处理。

4. 结果统计与输出

在每一轮实验中,保存成功数(random_success_count 和 cyclic_success_count），并在最后计算成功率。

设定成功条件（如所有囚犯都成功找到编号）。

输出格式:使用 print 函数格式化输出每一轮的结果和最终的成功率，使结果一目了然。

5. 随机性控制

随机种子设置:

使用 os.urandom 和 random.seed 来设置随机种子，确保每次实验的随机性，以避免结果重复性。

这样既能提高实验的多样性，又能在调试时保证结果的可复现性。

6. 结论

通过本实验，我们可以观察到：

随机搜索　无法有效提高成功率。

循环策略　尽管更有希望，但依然存在局限性。

未来的研究将集中在优化策略与参数调优上，以进一步提高成功率。