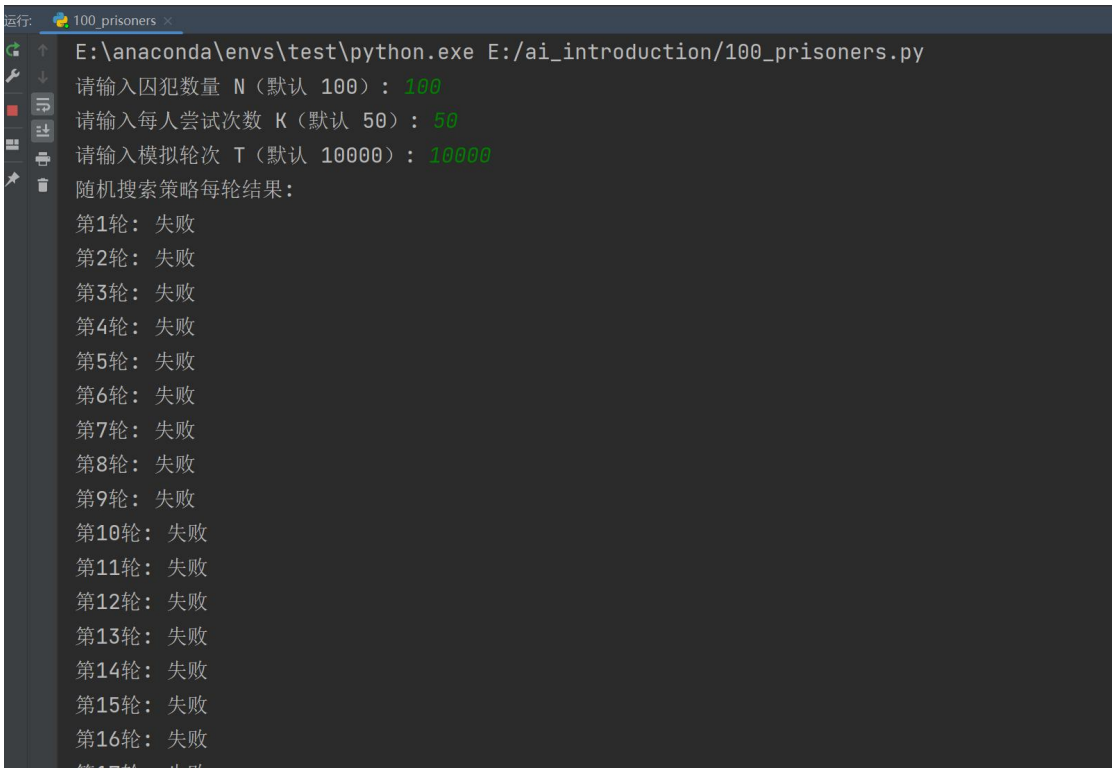
百囚犯问题实验报告

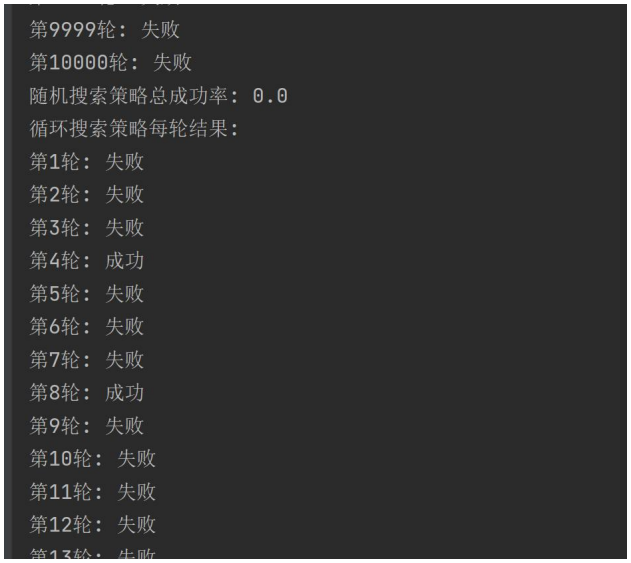
1. 算法说明

随机搜索策略：每个囚犯独立、随机地打开盒子，每次从所有盒子中随机选择一个未尝试过的盒子，直到找到自己的编号或用尽尝试次数。每个囚犯的每次选择都是独立随机事件，不依赖任何结构或顺序。如果所有囚犯均在各自的尝试次数内通过随机选择找到自己的编号，则成功。

循环搜索策略：利用盒子排列的循环结构，每个囚犯从与自己编号相同的盒子开始，按盒内纸条编号依次跳转，形成一条确定的循环路径。若排列中所有循环的长度均不超过尝试次数 K，则所有囚犯均能在 K 次内找到自己的编号，则成功。

2. 实验结果





```
第9994轮： 失败
第9995轮： 失败
第9996轮： 成功
第9997轮： 失败
第9998轮： 失败
第9999轮： 失败
第10000轮： 失败
循环搜索策略总成功率： 0.3148
调整后随机搜索策略成功率（N=50, K=25）: 0.0
调整后循环搜索策略成功率（N=50, K=25）: 0.3114
```

结果分析：在随机搜索策略中，成功概率几乎为 0（在这里我只保留四位小数，所以概率为 0），在循环搜索策略中，成功率为 0.3148，在 N 和 K 同时减半时，概率基本差不多，为 0.3114。

3. 理论分析

随机搜索策略

如果每名囚犯都随机打开 50 个盒子，那么对于一名囚犯没有选到自己号码牌的概率为 $P(\overline{A})=\frac{1}{2}$ 因此囚犯选到自己号码的概率： $P(A)=\frac{1}{2}$。100 名囚犯都需要选到自己的号码： $P(success)=)(\frac{1}{2})^{100}$ 。由此可以看出如果随机的选，这些囚犯几乎必死。

循环搜索策略

每个囚犯选择自己的号码牌开始，下一个选择的盒子为这次盒子中的号码对应的盒子，按照这个策略，经过有限次打开，盒子的操作，肯定能找到自己的号码牌。实际上可以将盒子抽象为顶点，而盒子中的数抽象为边。一个囚徒能在 50 次之内找到处自己的号码，说明它号码所在的环中节点数少于 50。由于盒子中纸条的位置并没有改变，所以图也没有改变。让所有囚徒在 50 次以内找到自己的号码，就需要让最大的有向环中节点数少于 51。环最大节点数小于 51 的可能并不好算，因此我们算大于等于 51 的。

(k−1)!为 k 个数成一个大环的所有可能：第一个数不与自己的位置相同，就有 k-1 可能 ，第二个数位置与自己位置和前一个数占据位置数不同，有 k-2 种可能，之后的数都与前面占据的数加自身位置不同。因此总的可能数为(k-1)!。

假设此时的最大环节点数为 k，可能出现的情况数： $P(\mathrm{k})=\frac{C_{100}^{k}(k-1)(100-k)!}{100!}=\frac{1}{k}$

最大环节点大于 50 的可能： $P(failure)=\sum_{k=51}^{100}\frac{1}{k}$

因此使用该策略能活下去的概率为： $P(success)=1-P(failure)=1-\ln 2$

根据上述计算结果，我们知道即使囚犯数量更多，所有囚犯仍然有 30%左右的成功概率。与实验结果相符，在 N，K 等比例减少的时候，发现概率还是 30%左右。

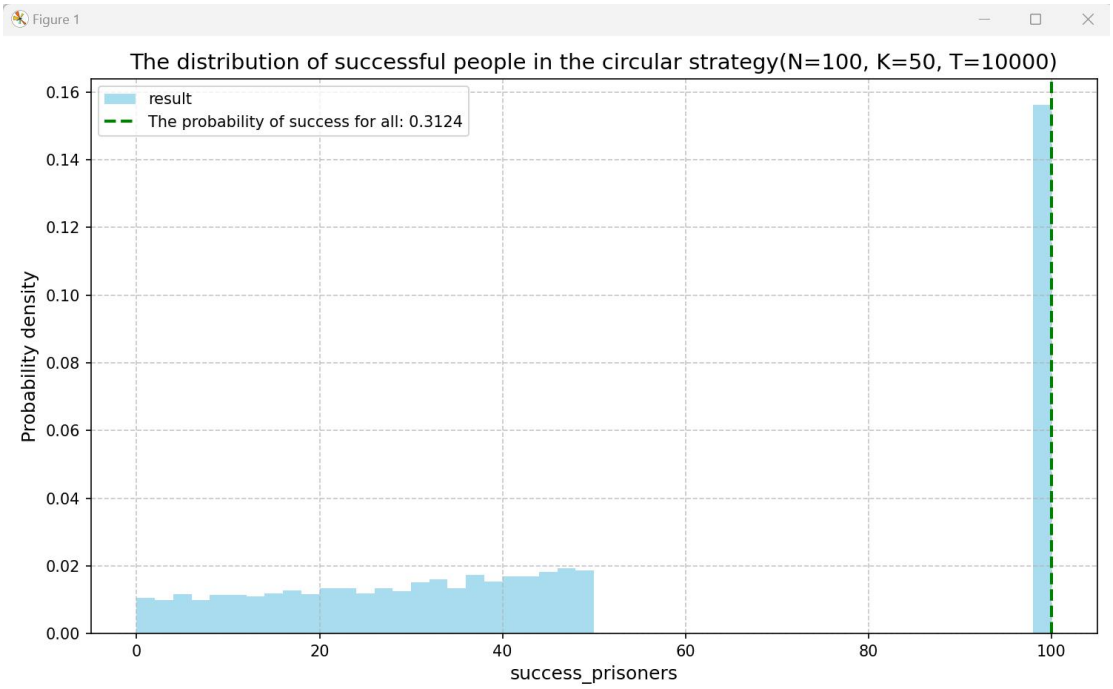
4. 优化思路

随机搜索策略优化：每个囚犯逐个尝试盒子，使用集合记录已尝试的盒子，避免重复选择。一次性生成所有囚犯在所有尝试中的随机选择，避免循环内重复调用随机函数。使用 boxes[current_choices] == prisoners 批量比较所有囚犯当前选择的盒子，利用 NumPy 的并行计算能力。一旦所有囚犯都找到编号，立即返回结果，减少不必要的计算。

循环搜索策略优化：每个囚犯按循环路径逐个跳转盒子，直到找到自己的编号或用尽尝试次数。路径跳转向量化：通过 current_boxes = boxes[current_boxes - 1] 一次性更新所有囚犯的位置，避免逐个处理。使用 boxes[current_boxes - 1] == prisoners 同时检查所有囚犯是否找到目标，充分利用 NumPy 的并行计算。

模拟函数优化：一次性生成所有轮次的盒子排列，避免在循环中重复生成，减少随机数生成的开销。直接调用向量化的搜索策略函数，避免在模拟循环中引入额外的 Python 循环。

5. 图



图像分析：展示了采用循环策略时成功人数的概率分布图，这里不是单纯的密度，而是概率密度分布。可以看出，在人数较少和 100 左右的时候，概率存在，也就是说，概率分布在两端，中间一部分的概率几乎为 0。