100 囚犯抽签问题实验报告

2023141461105

刘鑫

一、算法说明

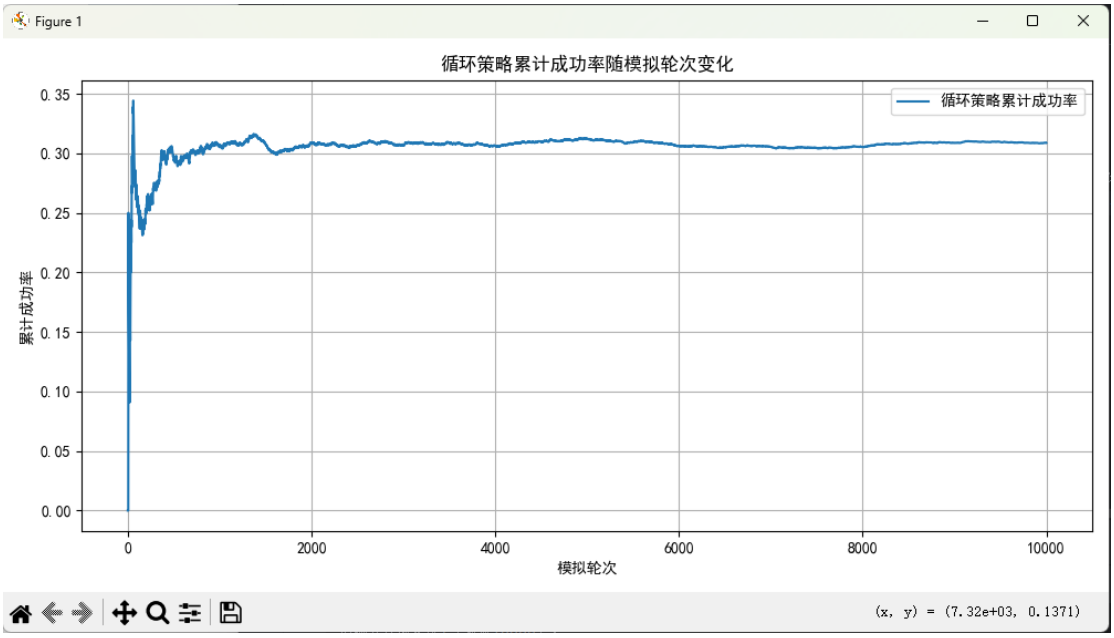
本项目基于经典的“100 囚犯抽签问题”进行仿真模拟，使用 Python 编写，主要实现了两种搜索策略：

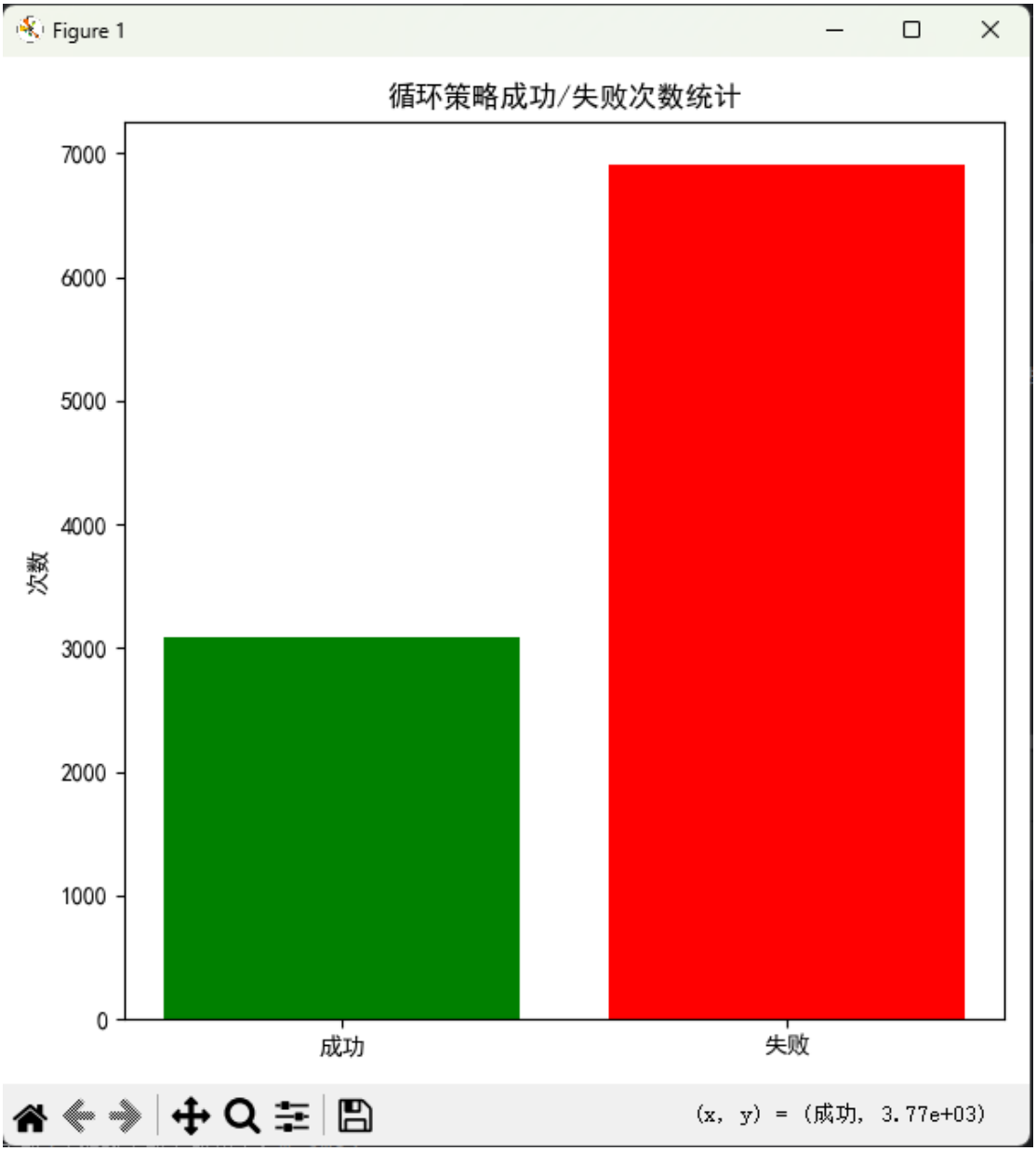
- **策略 1（随机搜索）**
  每个囚犯随机选择 50 个不同的盒子，检查是否包含自己编号。若任一囚犯未找到编号，则本轮失败。
- **策略 2（循环策略）**
  每个囚犯从自己编号对应的盒子开始，打开盒子内的编号作为下一次要打开的盒子编号，最多跳转 50 次。若所有囚犯均找到编号，则本轮成功。

核心实现中，盒子编号使用 list(range(N))，通过 random.shuffle() 随机排列。每轮模拟均随机生成新排列，重复 T=10000 次。

实验程序中利用循环遍历模拟所有囚犯搜索过程，成功与否结果保存在列表中，便于统计和可视化。

二、实验结果

- **模拟参数**：
  - 囚犯数量 N = 100
  - 最大尝试次数 K = 50
  - 模拟轮数 T = 10000
- **实验数据**：
  - 策略 1（随机搜索）成功率约为 0.0000
  - 策略 2（循环策略）成功率约为 0.3098
- **结果可视化**：
  通过 matplotlib 绘制循环策略每轮成功（标记为 1）或失败（标记为 0）折线图，图像保存为 runtime_plot.png。
- 

- 

从图中可以看出循环策略在多次实验中大约有 31% 的成功率，明显优于随机策略的几乎为零。

三、优化思路

- **提前终止**
  在循环策略中，一旦检测到某囚犯搜索超过最大尝试次数即终止当前模拟，避免无谓计算。
- **并行计算**
  由于模拟次数大，考虑使用 Python 多线程或多进程加速实验，提高运行效率。
- **参数灵活调整**
  程序中默认参数可调节，方便观察不同 N 和 K 对成功率的影响。
- **理论对比**
  结合排列循环理论，分析成功率与最大循环长度的关系，为仿真结果提供理论支持。

四、代码关键点说明

- 使用 random.shuffle() 生成盒子随机排列，确保无重复编号。
- 策略 1 采用 random.sample() 选取随机盒子索引模拟随机搜索。
- 策略 2 用循环跳转盒子编号实现链式搜索。
- 统计成功率时，用 success_count / T 计算比例。
- 利用 matplotlib 画出成功/失败序列，直观展示循环策略表现。

五、总结

本实验通过大量模拟验证了经典循环策略的有效性，成功率约 31%，远高于随机策略。该方法利用排列的循环性质，显著提高囚犯整体获释概率。未来工作将包括算法性能优化、参数扩展和更深入的理论分析。