**软件体系结构实验报告**

**(2023-2024学年第一学期)**

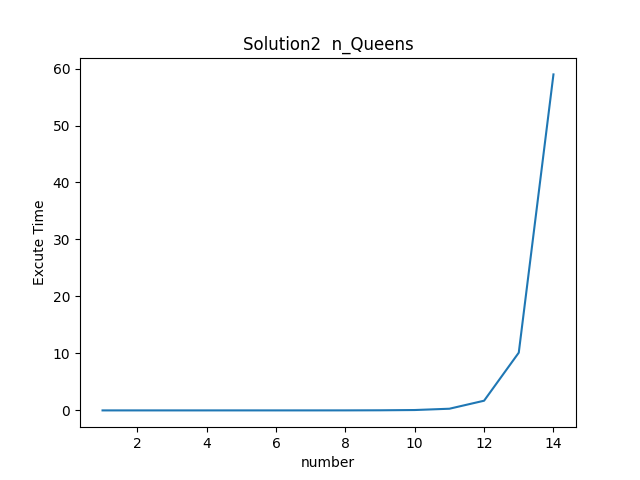
# 实验：n-Queen 问题

1、软件**[架构描述](" \l "%E4%B8%89%E3%80%81%E6%9E%B6%E6%9E%84%E6%8F%8F%E8%BF%B0)，[模块描述](" \l "%E5%9B%9B%E3%80%81%E6%A8%A1%E5%9D%97%E6%8F%8F%E8%BF%B0)，[使用算法](" \l "%E4%BA%94%E3%80%81%E4%BD%BF%E7%94%A8%E7%AE%97%E6%B3%95)，[设计模式](" \l "%E5%85%AD%E3%80%81%E8%AE%BE%E8%AE%A1%E6%A8%A1%E5%BC%8F)等**，详见源码中**readme.md**



1. 本报告仅展示OOP Pattern，Pipe and Filter Pattern，Shared-Data Pattern三种设计模式的**可修改性、性能，安全性**的对比（均基于Solution2.py开发）
2. **基本模式**

|  |  |
| --- | --- |
| 次数 | 运行时间 |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 0 |
| 5 | 0.001011133 |
| 6 | 0 |
| 7 | 0.000999451 |
| 8 | 0.004000902 |
| 9 | 0.019004822 |
| 10 | 0.065004587 |
| 11 | 0.293065786 |
| 12 | 1.700381756 |
| 13 | 10.13527417 |
| 14 | 58.97023964 |

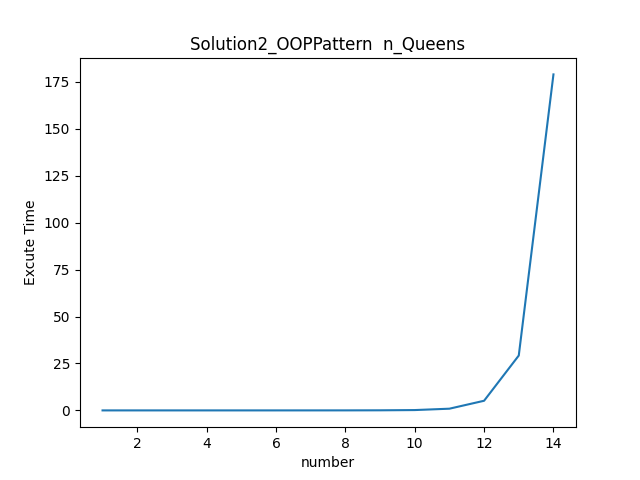
****

第14次时运行时间为58s

1. **OOP设计模式**

**性能**

|  |  |
| --- | --- |
| 次数 | 运行时间 |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 0 |
| 5 | 0.000999689 |
| 6 | 0 |
| 7 | 0.004000902 |
| 8 | 0.011993408 |
| 9 | 0.049011707 |
| 10 | 0.176039219 |
| 11 | 0.908203602 |
| 12 | 5.134152174 |
| 13 | 29.22655821 |
| 14 | 178.979037 |

****

可以看到第14次的时候运行时间已经达到了178s，相较于基本模式多了三倍的时间，性能更低

原因是代码中使用过多的封装性，如将皇后对象，棋盘对象，算法对象封装在了不同的类中，并且返回结果时需要将对象中封装的结果解码，增加了运行时间，性能下降较多

**可修改性**

采用封装性，具有良好的可修改性，因为它们提倡封装和松散耦合。通过良好定义的接口和类之间清晰的关系，可以相对容易地进行修改和扩展。

如在这种设计模式下，棋盘类或者解法类发生改变时，只要维护接口一致性，可以保持其他类不受影响，可修改性较强。

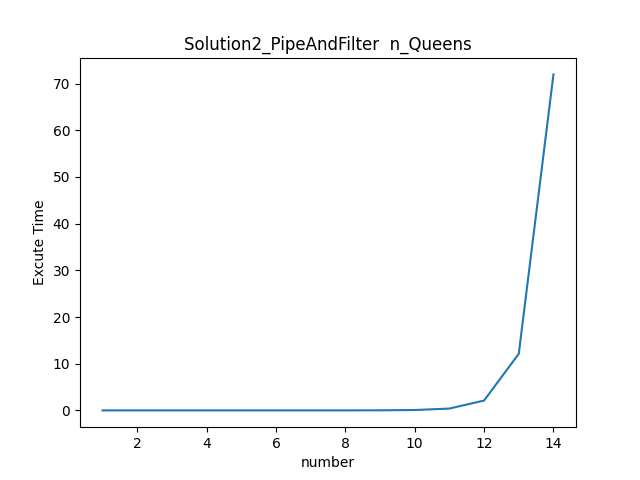
**安全性**

安全性方面，OOP模式通常提供了一定程度的安全性，因为封装允许隐藏实现的细节，而只是展示外部的接口。这样可以防止内部成员被修改，安全性较高。

1. **Pipe and Filter Pattern设计模式**

**性能**

|  |  |
| --- | --- |
| 次数 | 运行时间 |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 0.001999855 |
| 8 | 0.004000425 |
| 9 | 0.021005154 |
| 10 | 0.080017805 |
| 11 | 0.393088102 |
| 12 | 2.11147356 |
| 13 | 12.12422442 |
| 14 | 71.96214819 |

****

第14次的运行时间是71s，性能下降相对较低，性能方面更好些

Pipe and Filter模式通常具有良好的性能，因为它将系统划分为一系列相互独立的过滤器，每个过滤器负责一个特定的任务。这种模式可以实现并行处理，从而提高性能。

本模式的包括棋盘数据源，解法过滤器，结果存储器三种过滤器，解法过滤器从数据源获取数据，并且将结果以编码形式传给结果存储器进行解码，这样可以让每个任务划分给独立的模块处理，性能更好些

**可修改性**

可修改性方面，Pipe and Filter模式也表现良好。通过添加、删除或替换过滤器，系统的功能可以相对容易地进行修改和扩展。

如在相关文件中，替换解法过滤器中算法为另一种，或者更改数据源的编码模式，只要接口保持一致，便可以修改，有较强的可修改性

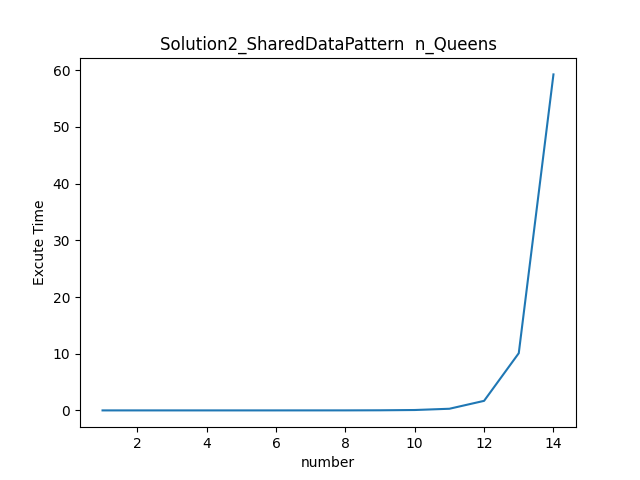
**安全性**

安全性方面，Pipe and Filter模式通过明确定义的接口和独立的过滤器提供了一定程度的安全性。每个过滤器可以独立测试和验证，从而减小引入错误的风险。数据隐藏在过滤器内部，并且通过过滤器之间的管道传递，不被外界暴露，安全性更好。

1. **Shared-Data Pattern设计模式**

**性能**

|  |  |
| --- | --- |
| 次数 | 运行时间 |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 0.002000093 |
| 8 | 0.004000902 |
| 9 | 0.019004345 |
| 10 | 0.068015814 |
| 11 | 0.29006505 |
| 12 | 1.684377909 |
| 13 | 10.09526539 |
| 14 | 59.27603054 |

****

第14次的时候运行时间为59s，性能跟基本模式几乎相同，性能不受影响，性能效果很好

**可修改性**

Shared-Data模式在可修改性方面可能较差。因为多个部分共享同一数据，修改一个部分可能会影响其他部分，增加了系统的复杂性和耦合度。

如在文件中，结果result，棋盘board，记录set，大小size被多个函数共享使用，不易修改，并且维护代价高，耦合度高，不适于编写程序

**安全性**

Shared-Data模式的安全性受到共享数据影响，每个变量均暴露在外，容易被修改，影响整程序安全，并且需要不断去维护，安全性非常差

1. **总结：**

**（1）性能方面：**

Pipe and Filter 模式和Shared-Data模式性能较高，前者是因为每个模块职责明确，性能更好，Shared-Data模式的变量被全局共享，直接调用代价较低，性能较高

而OOP模式因为有过强的封装性，导致每个成员数据的访问，编码，解码都比较困难，增加了性能消耗，性能较低

**（2）可修改性：**

OOP模式和Pipe and Filter模式的可修改性较强，前者因为有较好的封装性，只要维护接口的一致性，便可以对模块内部进行修改而不影响到外部调用，后者是将数据与实现均放在每个过滤器内部，可以对过滤器进行修改而不影响其他模块

而Shared-Data模式修改性很差，因为每个数据变量都是被文件内函数共享的，耦合性过高，想要修改其中一个模块就必然会牵扯到其他模块，不易变更。

**（3）安全性：**

OOP模式和Pipe and Filter模式的安全性较高，因为两者均会对数据与实现隐藏，只展示外部的接口，OOP模式的数据封装在类中，不会被轻易访问改动，Pipe and Filter模式的数据流在管道内传递，有较好的独立性，降低风险。

而Shared-Data模式的安全性很差，每个数据都暴露在外，极其容易被修改，影响程序运行稳定性。