

**课程设计报告**

**题目：孔明棋游戏求解程序设计**

**课程名称： 命令式计算原理**

**专业班级： 计科本硕博2201**

**学 号： U202215699**

**姓 名： 温晨阳**

**指导教师： 李开**

**报告日期： 2024/5/1**

**计算机科学与技术学院**

**目 录**

**1　课程设计任务 1**

1.1 简介 1

1.2 设计内容 1

1.3 设计要求 2

**2　系统需求分析与总体设计 3**

2.1系统需求分析 3

2.2系统总体设计 3

**3　系统详细设计 5**

3.1有关数据结构的定义 5

3.2 主要算法设计 5

**4　系统实现与测试 7**

4.1系统实现 7

4.2系统测试 7

**5　总结 12**

**6　体会 13**

**附录 14**

# 1　课程设计任务

## 1.1 简介

孔明棋是一种单人桌游，它的目标是只留下一颗棋子在棋盘上，而其他的都移除。游戏使用一个特殊的棋盘，通常是一个由许多孔组成的正方形网格，每个孔可以放置一颗棋子或者为空。

游戏开始时，棋盘上的一些孔被填充了棋子，而其他的孔是空的。玩家每次可以选择一个棋子，然后跳过它的邻居（前、后、左、右的方向，不可以对角跳）并移除被跳过的棋子。玩家的目标是通过一系列的跳跃操作，最终只留下一个棋子在棋盘上。

课程设计，要求编写计算机程序来解决孔明棋游戏棋局。这个程序将会模拟游戏中的移动和操作，并尝试找到一种解法，即通过最少的步骤将棋盘上的棋子减少到只剩下一颗。

设计这个程序时，需要考虑如何表示棋盘状态、如何生成合法的移动、如何评估每一步的好坏等问题。通过合理设计算法和数据结构，可以实现一个高效且准确的孔明棋求解器。

## 1.2 设计内容

解决确定的孔明棋局： 编写函数 int peg\_solve(board B, stack S)，该函数接受一个棋盘和一个栈作为输入，返回一个整数表示解决棋盘时剩下的棋子数量。如果返回1，表示棋盘解决成功，并且栈S中包含了解决棋盘的一系列合法操作。如果返回大于1的数，表示棋盘解决失败，棋盘上还剩下的棋子数量。此时栈S可以是任意的。

解决不确定的孔明棋局： 扩展任务1的代码，使其能够处理需要回溯法的棋盘情况。在回溯搜索过程中，从当前状态开始，尝试每个可能的第一步操作，并递归地解决下一个状态，直到找到解或者无解。

附加题：记忆化。使用哈希表来记录已经访问过的棋盘状态，以避免重复搜索。在解决任务2中需要回溯的情况时，首先检查当前状态是否已经在哈希表中记录，如果是，则直接返回哈希表中存储的结果；否则，继续搜索并更新哈希表。

在解决这些任务时，需要合理选择数据结构、算法和哈希函数，以实现高效的解决方案。任务的完成应该能够通过给定的测试数据，并且在给定的时间限制内完成。

## 1.3 设计要求

要使用合适的数据结构和算法来实现任务要求。代码清晰易读，使用适当的命名和注释，方便他人理解和维护。确保代码的正确性和稳定性，通过给定的测试数据，并在给定的时间限制内完成任务。也要考虑代码的效率，尽可能优化算法和数据结构，以提高程序性能。

最后使用不同的棋盘情况和复杂度来测试程序，验证程序的正确性和稳定性。考虑算法的效率和内存占用，以确保程序能够在合理的时间内完成计算，并且不会占用过多的系统资源。

# 2　系统需求分析与总体设计

## 2.1系统需求分析

1. 能够有效地解决给定的孔明棋棋局，将棋盘上的棋子减少到只剩下一颗。
2. 能够处理不确定的孔明棋棋局，采用回溯法解决需要多步操作的情况。
3. 支持记忆化功能，使用哈希表记录已经访问过的棋盘状态，以避免重复搜索，提高解决效率。

## 2.2系统总体设计

1. 游戏状态表示模块：

包括定义游戏棋盘的数据结构和操作函数，以及定义棋盘状态的相关属性和操作函数。

1. 移动规则检查模块：

包括检查给定移动是否符合游戏规则的函数，例如检查是否可以进行跳跃等。

1. 移动执行与撤销模块：

包括执行和撤销游戏棋子移动的函数，确保游戏状态可以正确地更新。

1. 可行移动生成模块：

根据当前棋盘状态生成所有可能的合法移动，以供后续的搜索和决策使用。

1. 解法搜索模块：

包括递归地搜索可能的游戏解法的函数，以找到最优解或者任意解。

1. 哈希表模块：

用于存储已经搜索过的棋盘状态，避免重复搜索，提高搜索效率。

1. 主控制模块：

整合以上模块，控制整个游戏求解流程，包括初始化游戏状态、执行解法搜索、输出结果等。

# 3　系统详细设计

## 3.1有关数据结构的定义

游戏棋盘状态（board）用于表示游戏当前的状态，包含在哈希表（ht）中。

移动序列栈（stack）用于存储移动序列，其中包含的移动（move）通过哈希表元素（htelem）关联到游戏棋盘状态（board）。

哈希表（ht）用于存储已经搜索过的棋盘状态，其元素是哈希表元素（htelem），这些元素包含了游戏棋盘状态（board）和与之相关的其他信息。

|  |  |  |
| --- | --- | --- |
| 游戏棋盘状态(board) | B | int[] |
| 移动序列栈(stack) | S | stack |
| 哈希表(ht) | map | ht |
| 移动(move) | m | int[] |
| 哈希表元素(htelem) | node | struct |

## 3.2 主要算法设计

（1）棋盘状态表示模块：

游戏棋盘的数据结构是一个一维数组，用于表示棋盘的状态。每个元素代表一个棋盘位置，其值为1表示该位置有棋子，值为0表示该位置为空。

（2）移动规则检查模块：

检查给定的移动是否符合游戏规则。如：移动是否在棋盘范围内、起始位置和目标位置是否为合法位置、起始位置和目标位置相隔一个棋子。

（3）移动执行与撤销模块：

执行移动时，将起始位置和目标位置的棋子状态更新。撤销移动时，将目标位置和跳跃位置的棋子状态恢复为原始状态。

（4）可行移动生成模块：

遍历棋盘上的每个位置，检查是否有可跳跃的相邻棋子。如果有，则生成对应的移动序列，并将其存储起来。最后将所有的可移动序列返回。

（5）解法搜索模块：

首先检查当前棋盘状态是否已经是解决状态，如果是则返回解的长度为1。否则，生成所有可能的移动序列，并对每个移动序列递归调用解法搜索函数。在递归调用过程中，记录最短解的长度，并在找到解时更新解的路径。

（6）哈希表模块：

存储已经搜索过的棋盘状态，以避免重复搜索。使用哈希表存储棋盘状态，每个键值对包含棋盘状态和与之相关的其他信息。在搜索过程中，检查当前棋盘状态是否已经存在于哈希表中，如果是则跳过当前状态的搜索。

流程之间的过程如图3-1所示。

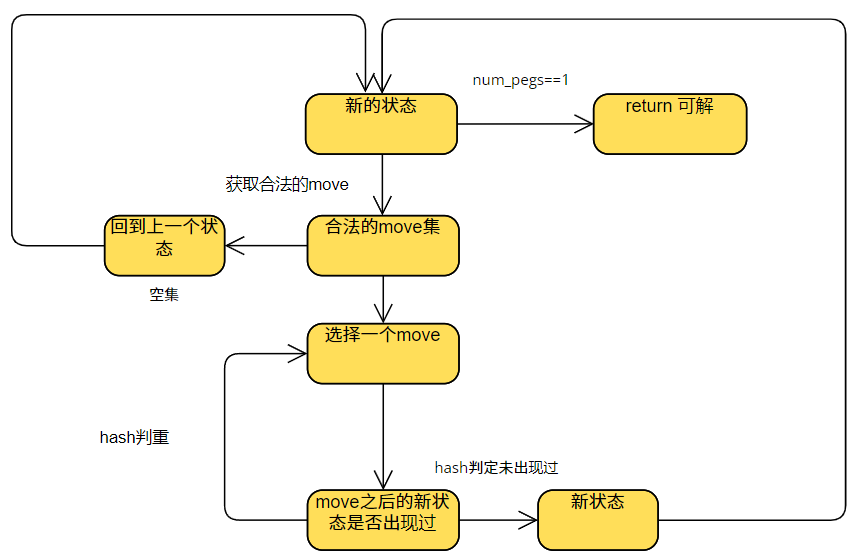


图3-1 系统总设计流程图

# 4　系统实现与测试

## 4.1系统实现

在win11内的虚拟机中构建Ubuntu22.04环境中运行。部分重要函数功能介绍如下：

bool is\_valid\_move(board B, move m):检查给定的移动是否在当前的棋盘状态下是有效的。

void find\_valid\_moves(board B, stack S): 这个函数用于找到当前棋盘状态下所有有效的移动，并将它们压入一个栈中。

htelem create\_node(board B): 这个函数用于创建一个哈希表节点，以便在解决过程中跟踪已经访问过的棋盘状态。

int solve(board B, stack S, int pegs, ht map): 这个函数用于解决孔明棋游戏，它使用回溯算法递归地找到最优解。

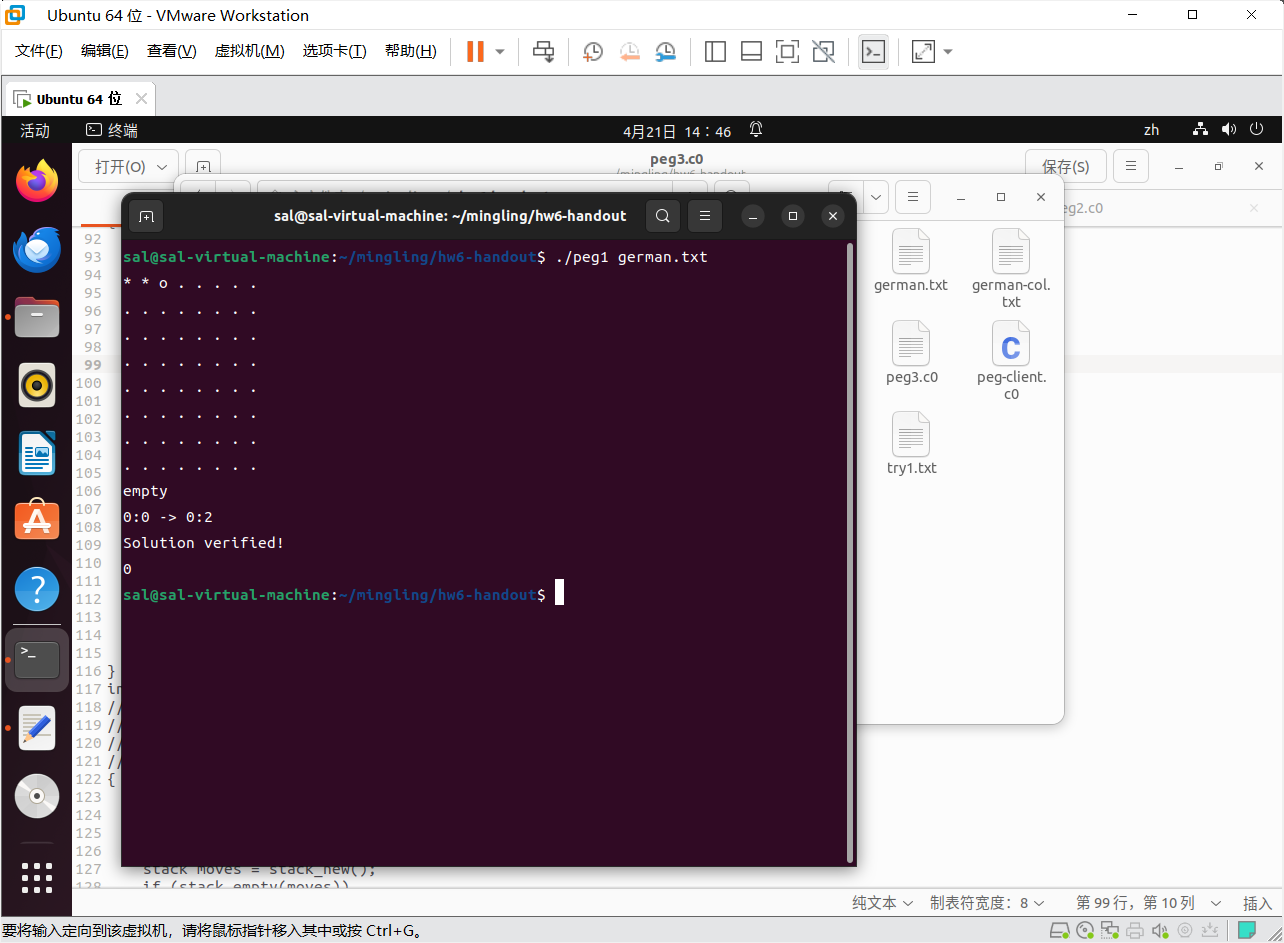
## 4.2系统测试

1. 任务1

对于棋盘B，在find\_valid\_moves（B，S）种穷举所有棋子，并通过is\_valid\_move(board B, move m)，来找到当前棋盘状态下可能的移动方式，并将移动方式压入栈S并返回。采用此移动方式得到新的棋盘B0，然后递归调用solve（B0，S）；当棋盘上棋子数为1时，将此时的移动方式压栈，并返回。

测试代码：./peg1 german.txt

测试效果如图：



1. 任务2

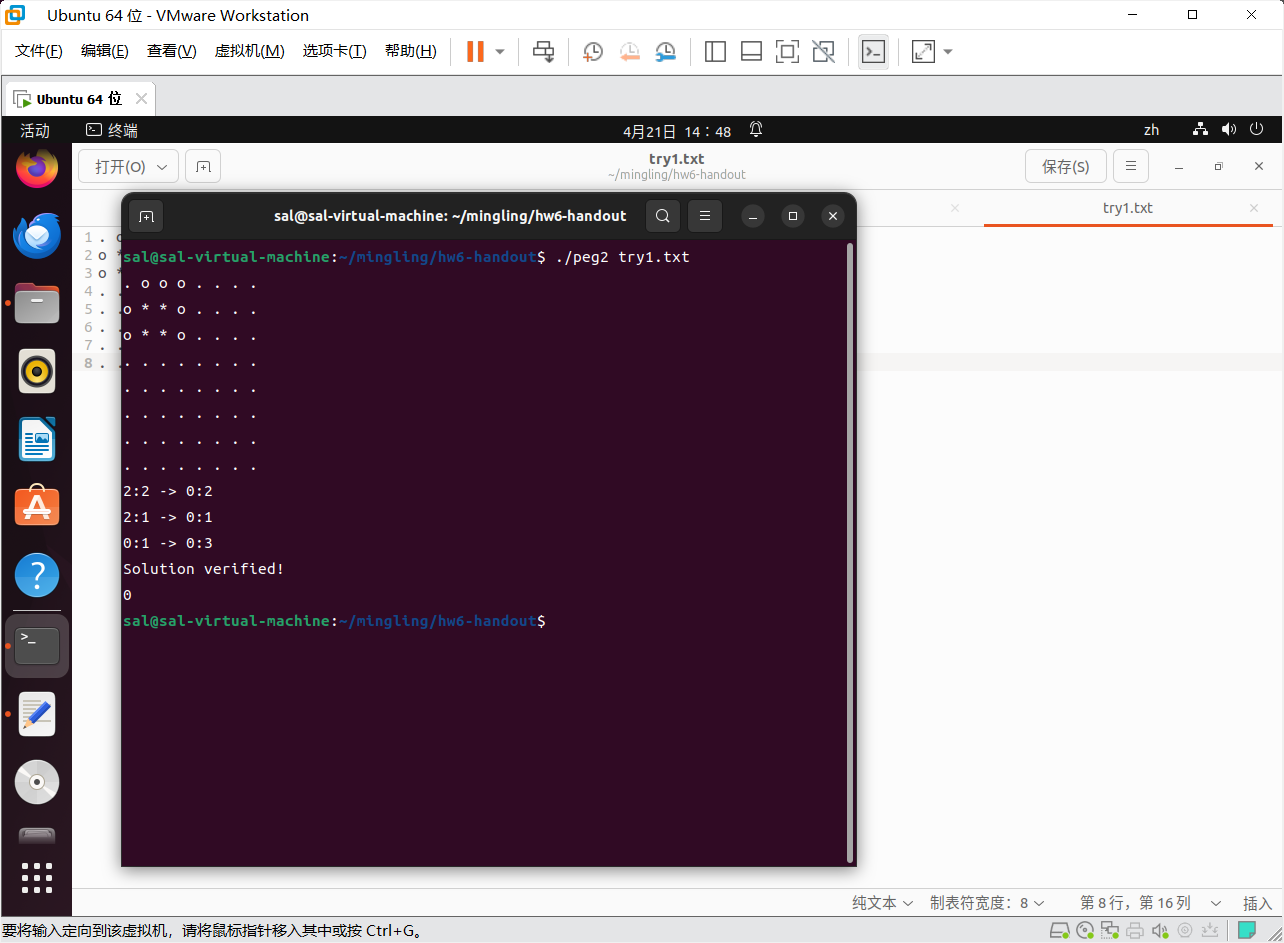
由于棋盘的移动状态并不唯一，则使用一个栈来保存所有的移动方式，每次使用栈顶的移动方式进入递归，并出栈。如果失败，则逆向使用此移动方式还原棋盘。然后继续调用栈顶的移动方式直至栈空为止。

对于一个简单案例：

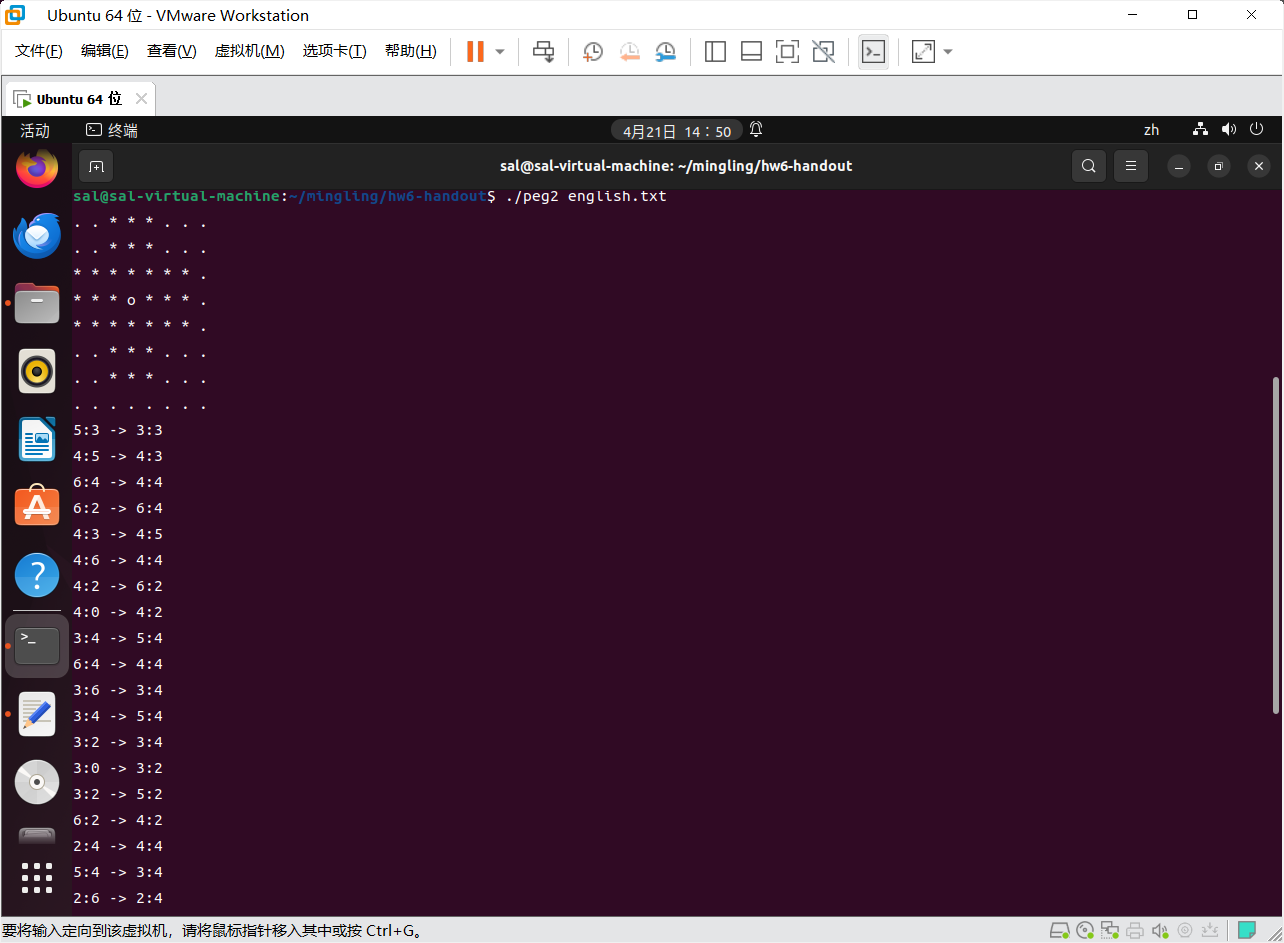


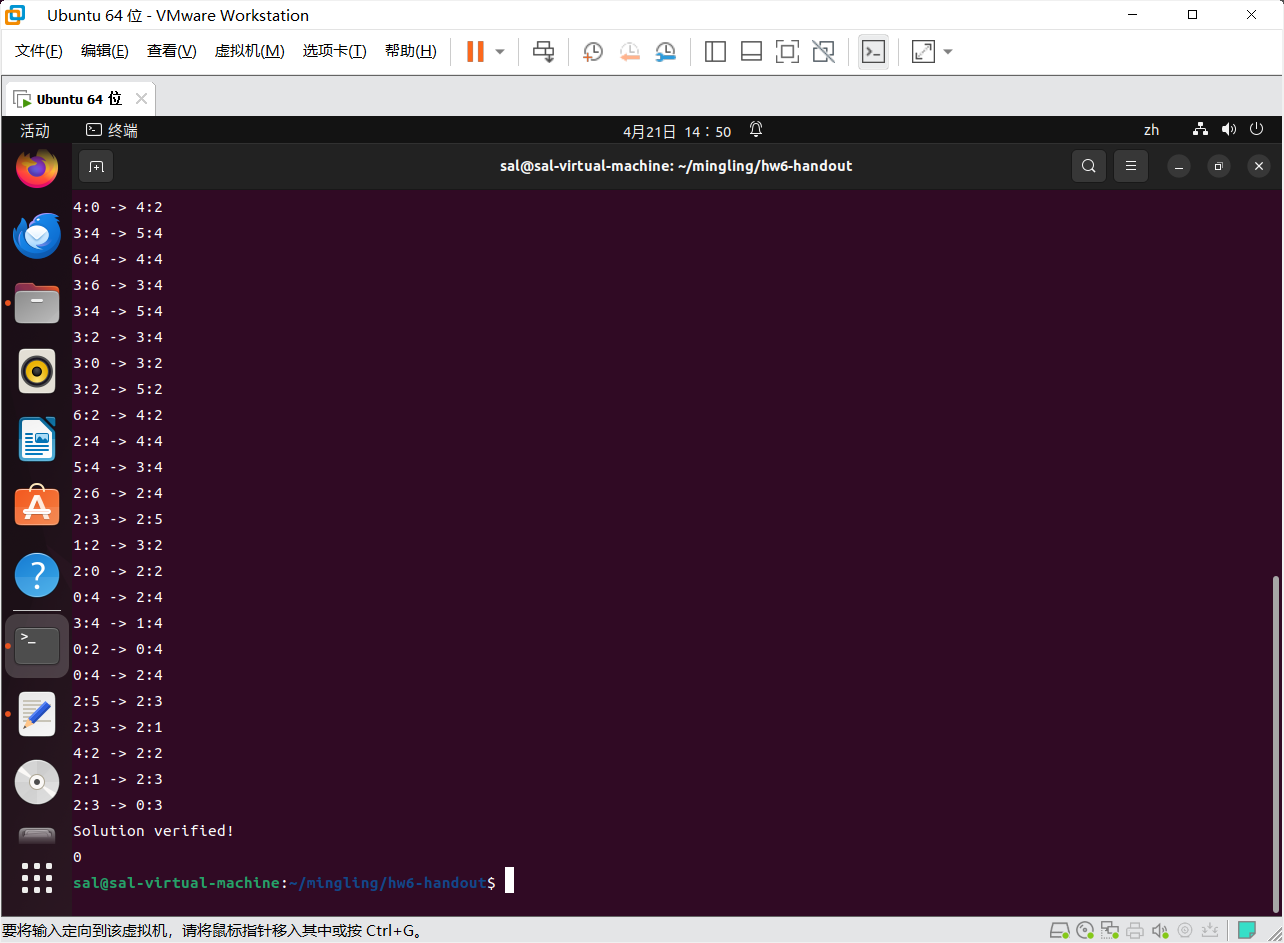
./peg2 try1.txt

可以看到正确的给出了解。



对于测试案例./peg2 english.txt



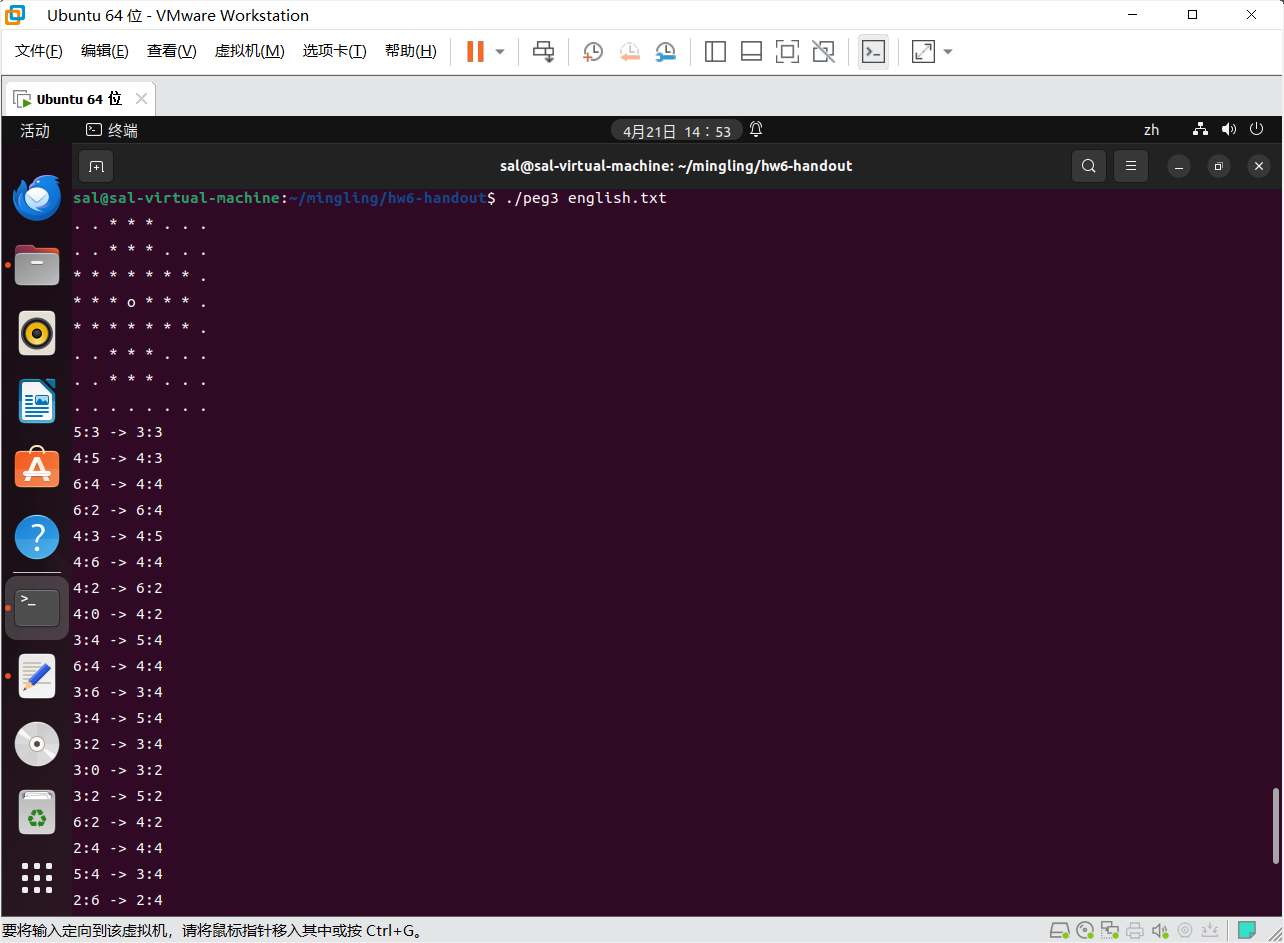


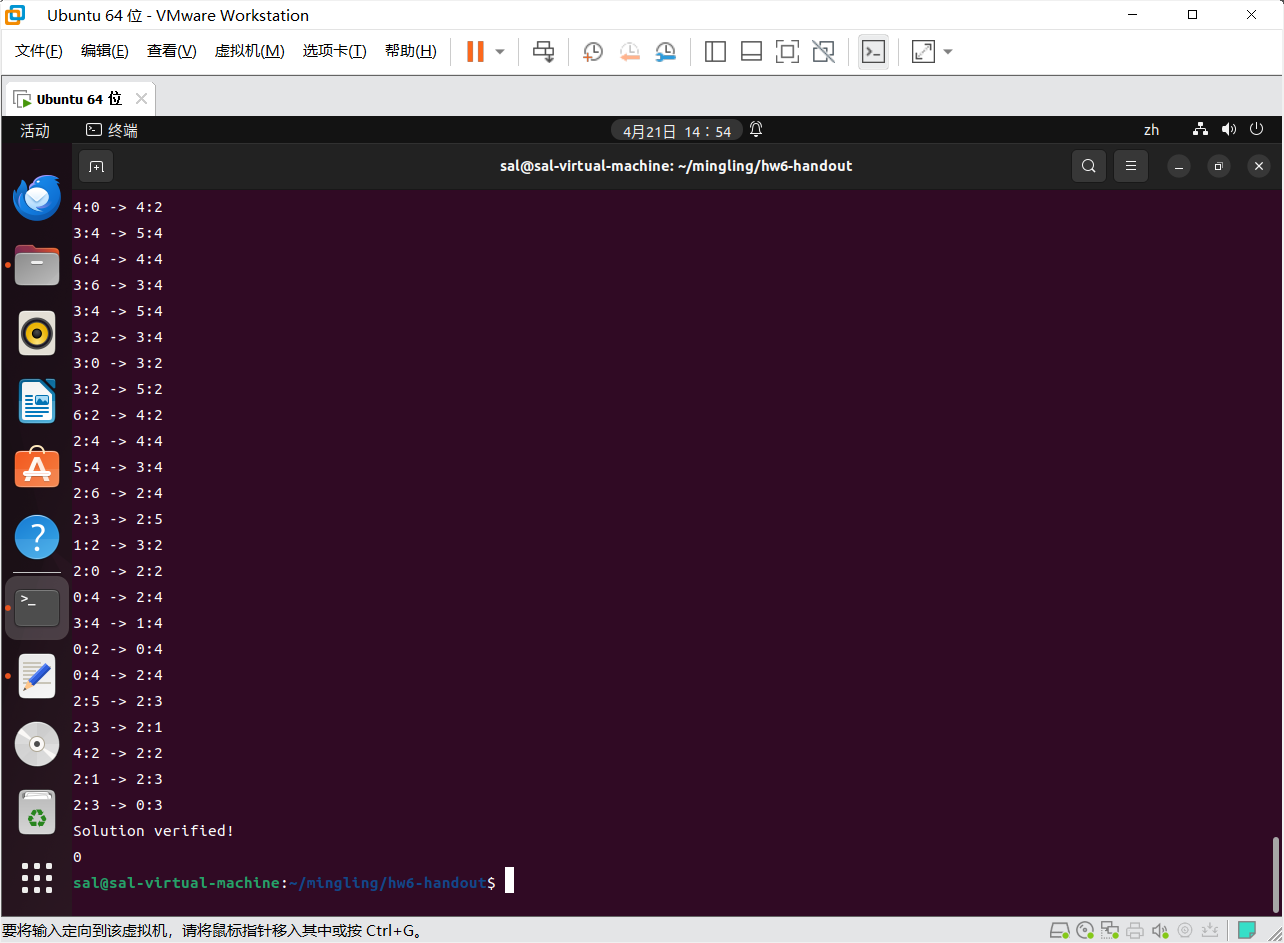
可以看出，虽然正确解出答案，但是时间比较慢，经测试大概需要9s

1. 任务3

每次将棋盘状态都保存在hash表里面，每次构成新的棋盘就在hash表寻找，如果找到了就说明已经考虑过这个状态，就不需要再次考虑。

./peg3 english.txt





经过测试，不到1s就找到了解。

# 5　总结

深入研究孔明棋的规则和游戏机制，并思考如何将这些规则转化为计算机可以理解和处理的形式，是理解问题的关键。

通过设计适当的数据结构来表示棋盘和操作，编写算法来生成合法的操作序列，并递归地尝试这些操作，在尝试每一种可能的操作后，检查是否有解，如果没有则回溯到上一步并尝试其他操作。在递归过程中，需要考虑到棋盘状态的保存和恢复，以及合法操作的生成等问题。

最后一定要进行代码的测试和调试，检查代码在各种情况下的正确性和健壮性。

# 6　体会

编写清晰、可读性高、模块化的代码对于项目的可维护性和扩展性至关重要。Coding的过程中需要遵循良好的编码规范，使用有意义的变量名和函数名，并尽量减少代码的重复和冗余。在代码框架完成后，充分的测试和调试是保证代码质量的关键。在项目中遇到困难和挑战时，也可以通过查阅文档和资料，与同学交流等方法来解决问题。

大作业设计与实现是我在这门课学习过程中的重要组成部分，不仅帮助我提升了编程技能，也培养了问题解决能力。

# 附录

//判断是否是合法移动

bool is\_valid\_move(board B, move m)

//@requires is\_board(B);

{

if (B[m[0]] == 1 && B[m[1]] == 1 && B[m[2]] == 0 &&

((row\_start(m) == row\_end(m) && abs(col\_start(m) - col\_end(m)) == 2) ||

(col\_start(m) == col\_end(m) && abs(row\_start(m) - row\_end(m)) == 2)))

{

return true;

}

else

{

return false;

}

}

//寻找所有合法移动

void find\_valid\_moves(board B, stack S)

//@requires is\_board(B);

//@requires stack\_empty(S);

{

for (int i = 0; i < 8 \* 8; i++)

{

int[] arr = alloc\_array(int, 4);

arr[0] = i - 16;

arr[1] = i + 16;

arr[2] = i - 2;

arr[3] = i + 2;

for (int j = 0; j < 4; j++)

{

if (arr[j] >= 0)

{

move m = alloc\_array(int, 3);

m[0] = i;

m[1] = (i + arr[j]) / 2;

m[2] = arr[j];

if (is\_valid\_move(B, m))

{

push(S, m);

}

}

}

}

}

//构建哈希值

htelem createnode(board B)

//@requires is\_board(B);

{

htelem new = alloc(struct two\_ints);

int first = 0;

int second = 0;

for (int i = 0; i < 32; i++)

{

if (B[i] == 1)

{

first = first | 0x01;

}

first = first << 1;

}

for (int j = 32; j < 64; j++)

{

if (B[j] == 1)

{

second = second | 0x01;

}

second = second << 1;

}

new->i1 = first;

new->i2 = second;

return new;

}

递归求解

int solve(board B, stack S, int pegs, ht map)

//@requires is\_board(B);

//@requires num\_pegs(B) >= 1;

//@ensures is\_board(B);

//@ensures \result >= 1;

{

if (num\_pegs(B) == 1)

{

return 1;

}

stack moves = stack\_new();

find\_valid\_moves(B, moves);

if (stack\_empty(moves))

{

return num\_pegs(B);

}

int min\_num = pegs;

while (!stack\_empty(moves))

{

move m = pop(moves);

excute\_move(B, m);

pegs -= 1;

htelem node = createnode(B);

if (ht\_lookup(map, node) == NULL)

{

// print("\n");

// print\_board(B);

// recursively solve

int next\_solve = solve(B, S, pegs, map);

if (next\_solve < min\_num)

{

min\_num = next\_solve;

}

if (next\_solve == 1)

{

push(S, m);

return 1;

}

undo\_move(B, m);

pegs += 1;

ht\_insert(map, node);

}

else

{

undo\_move(B, m);

pegs += 1;

// print\_board(B);

}

}

return min\_num;

}