華中科技大學

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：CS2209

学　　号：U202210755

姓　　名：章宏宇

指导教师：李开

报告日期：2024.05.08

计算机科学与技术学院

目 录

1 课程设计任务

1.1 简介

先简要介绍孔明棋游戏，主要是游戏规则，然后表明本课程设计需要用计算机对孔明棋游戏棋局进行求解。

孔明棋，是在一个 8*8 的棋盘上，把所有格子分成三类：第一类是障碍，棋子不能在障碍内存在；第二类是棋子；第三类是空格，能放棋子但是当前格子内无棋子。规定合法操作为对于在水平或竖直方向上连续的三个格子“223”，可以经过一次操作，变为“332”，即跳过并吃掉一个棋子。目标是让棋盘上只剩一个棋子。

本课程设计需要用计算机对孔明棋游戏棋局进行求解，采用深度优先搜索方法，同时在多个方面进行优化。

1.2设计内容

（1）**任务一**：实现一个函数，能找到一个有解且解唯一的棋局的解。

（2）**任务二**：实现一个函数，能通过递归找到有解但不确定解的棋局的解。

（3）**任务三**：通过哈希表来优化任务二的函数，提升性能。

（4）**任务四**：优化遍历顺序、选择方向等来进一步提升性能。

1.3 设计要求

要求具有以下功能

（1）**设计数据类型**：包括棋盘的表示方式，Hash 表的 key 值类型，move 操作的表示方式，栈元素的类型等。

（2）**算法流程设计**：根据任务提示和代码框架，设计多个函数，分别完成寻找确定解、回溯法寻找不确定解等；同时需要严格遵守题目要求，正确构造栈，方便 verify 函数验证解的正确性。算法的设计可参考文献[1]。

（3）**时间性能的表示**：本设计不以绝对时间和常数大小为评判标准，而是

对于 Hash 表，根据不合法状态的数量来比对不同程序的相对优劣。

（4）程序优化：对深度优先搜索时的遍历顺序、遍历方向和 Hash 表的键值生成函数进行优化设计与实现，提供较明确的性能优化结果。

（5）语法码风：明显的代码约定问题将会导致扣分；也许会根据代码风格评分，特别是无法通过 cc0 的-w 风格检查的代码。

2 系统需求分析与总体设计

2.1 系统需求分析

首先，我们需要对合法操作进行设计，将每一个合法操作压缩成合适的形式。这样做的目的是为了在存储和传输过程中节省空间，并且在进行操作时能更高效地解压和处理。这种压缩形式不仅能够减少空间复杂度，还能在执行过程中提高时间复杂度，使得整体运算更加迅速和高效。

其次，我们需要具备解决一个确定棋局并能够走一步的能力。这意味着，我们必须设计一个算法或程序，能够在给定的棋局状态下，计算出下一步的最佳移动，并执行这个移动。这一过程不仅要求我们能够准确地分析当前的棋局状态，还要具备高效计算下一步行动的能力，从而推进棋局的发展。

然后，我们需要通过深度优先搜索（DFS）来解决一个不确定的棋局。深度优先搜索是一种遍历或搜索树或图的算法，通过尽可能深入到树的每一个分支来寻找解决方案。在这个过程中，当找到解时，递归地返回这个解；如果没有找到解，则返回剩下的最少棋子数。这要求我们的算法不仅能够全面搜索所有可能的棋局状态，还要能够在搜索过程中高效地管理和回溯，以确保找到最优解或最少的剩余棋子数。

最后，我们需要对深度优先搜索进行优化。优化的目标是提高搜索效率，减少不必要的搜索路径，从而加快找到解决方案的速度。优化方法可以包括剪枝技术、启发式搜索、记忆化搜索等。这些优化技术能够显著减少搜索空间和时间，提高算法的整体性能，使得在复杂棋局中的深度优先搜索更加高效和可行。

2.2 系统总体设计

系统主要包括三个部分：库函数定义/实现、Peg Solitaire的递归求解部分、验证解部分。

库函数定义/实现部分CMU基本实现了，只需要对于某些类型进行设计和定义即可。

递归求解包括两个部分：建立Peg初始格局并预处理，深度优先搜索求解Peg。对于第二个部分，应任务要求实现了多个版本以处理不同棋局。

如下为系统的模块结构图：

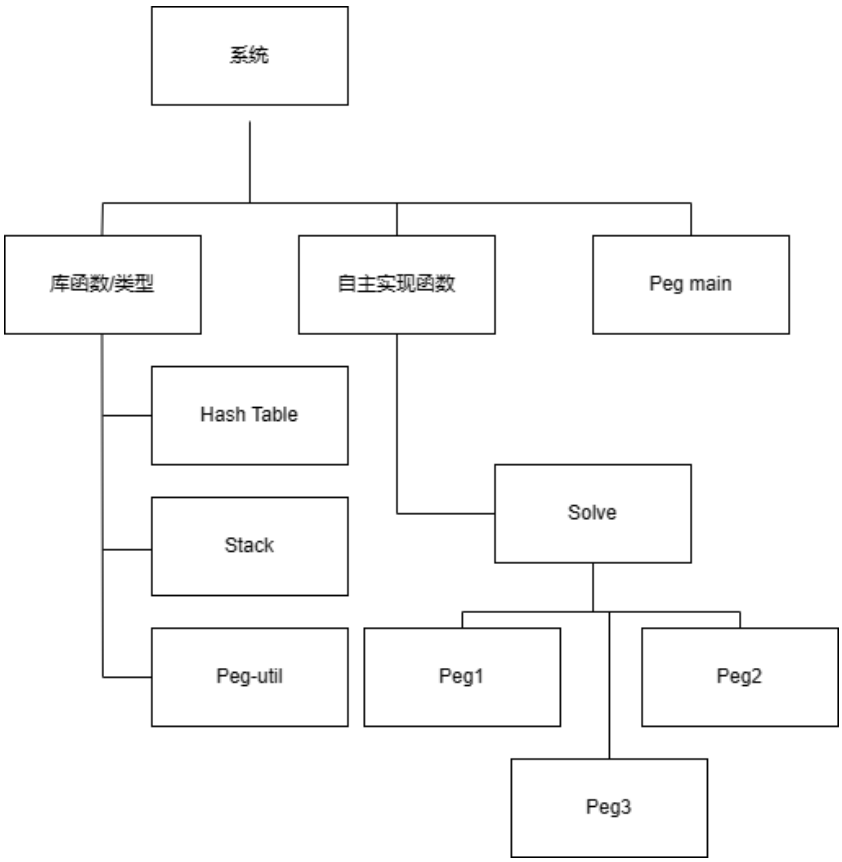

图 2-1 系统模块结构图

3 系统详细设计

3.1 有关数据结构的定义

要处理的数据：基本数据类型，Peg格局；

基本数据类型包括：操作表示，栈元素，Hash表元素，Hash表键值；

Peg格局：Peg棋盘表示，Peg棋盘压缩。

表 3-1 系统数据及数据类型

数据	数据元素	数据项	数据类型
基本数据	操作表示	move	整型
	栈元素	stackelem	整型
	Hash表元素	htelem	two_ints*
	Hash表键值	htkey	two_ints*
Peg格局	Peg棋盘表示	board	整型一维数组
	Peg棋盘压缩	two_ints	结构体

3.2 主要算法设计

首先，设计 move 的类型。这里采用十进制压缩，即 move 的每个十进制位表示一维坐标。即：

$$x_s = m/1000 \ \%\ 10$$

$$y_s = m/100 \ \%\ 10$$

$$x_t = m/10 \ \%\ 10$$

$$y_t = m/1 \ \%\ 10$$

其次，对于一个确定的棋局，我提出了一个简洁又优雅的枚举方式，即先预处理处两个方向数组 dx[]={-1, 0, 0, 1}, dy[]={0, -1, 1, 0}，这样只需要枚举位置和方向下标即可。而且这种方法的扩展性极强。

然后，针对深度优先搜索，按照下图流程图编写程序即可：

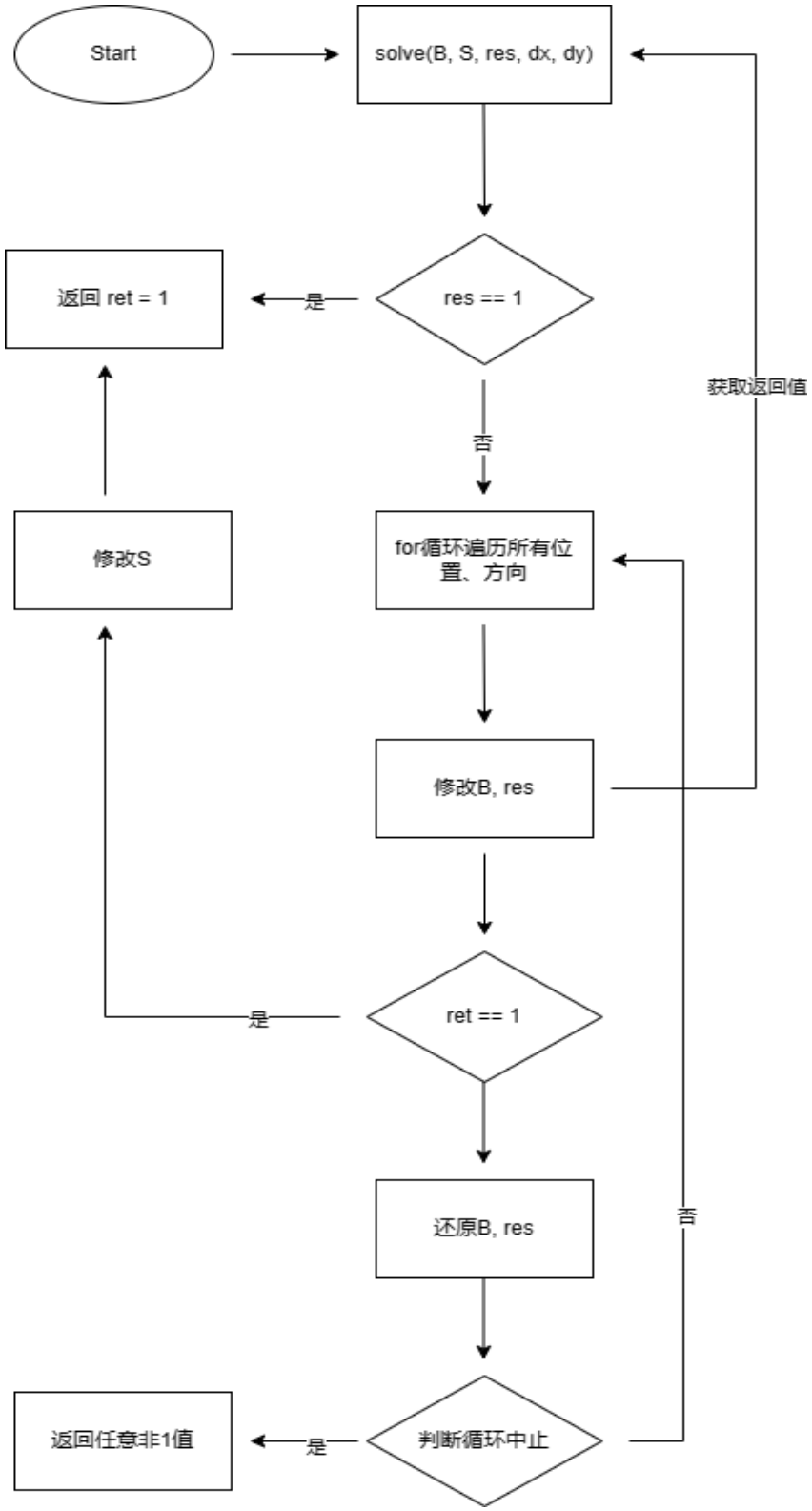

图 3-1 深度优先搜索流程图

最后，对深度优先搜索进行记忆化优化，即将一个棋盘尽可能压缩，把压缩后的值当作key来hash。按照题目提示和代码提示可知，我们需要把一个Peg棋盘

尽可能压缩。

在此我采用了二进制压缩方法，有棋子位置为1，无棋子位置为0，因为int型最大为$2^{32}-1$，所以将前32位和后32位压缩即可，即：

$$i_1 = \sum_{0}^{31} 2^j (B[j] == 1)$$

$$i_2 = \sum_{32}^{63} 2^{j-32} (B[j] == 1)$$

而对于Hash表的hash函数，CMU提供的terrible版本实在不敢恭维，于是我改了很多版本，其中一个版本如下

$$hash_{key} = (i_1 * 1331) + (i_2 * 131)$$

在实际测试过程中，forward-only dfs对于french棋盘表现不佳，在本机上因为一开始遍历方向选择有误，同时WSL内存设置太小，导致french1总是爆内存。在我搜索了大量资料后，找到了一篇关于Peg Solitaire的论文[1]，在论文中发现可以采用forward-backward dfs—— 即双端搜索。于是我在peg4中实现了双端搜索的做法。

之后更是在遍历顺序和方向选择上测试了多种组合，最终找到了各个棋盘对应的最优解。

4 系统实现与测试

4.1 系统实现

这部分可首先叙述一下你的系统实现的软硬件环境；

根据 3.1 的设计，用指定语言定义各种数据类型；

程序代码部分在这里不需要给出来，只需要叙述清楚在系统中包括哪些函数，各函数的说明，如何利用这些函数实现系统各模块的功能，以及函数间的调用关系（可用图表示出来）。程序详见附录。

系统是以 C0 语言为载体，通过数组、Hash 链表、栈等数据结构来实现，一下是系统实现的软硬件环境，数据类型的具体使用和使用到的函数。

4.1.1 软硬件环境

硬件环境：PC 机，AMD Ryzen 7 7735H 八核 CPU 3.20GHz，32G 内存；

系统环境：WSL2；

4.1.2 数据类型

如系统设计所说，采取如下数据类型定义。

typedef int[] board;

typedef int stackelem;

typedef struct two_ints* htelem;

typedef struct two_ints* htkey;

typedef int move;

4.1.3 函数说明

以下为系统实现所使用到的函数，对函数功能的简单阐述以及函数之间的调用关系。

（1）peg1，peg2 求解模块函数说明

int row_start(move m):

解压 m 对应的x_s

int col_start(move m):

解压 m 对应的y_s

int row_end(move m):

解压 m 对应的x_e

int col_end(move m):

解压 m 对应的y_e

move op(int r_s, int c_s, int r_e, int c_e):

压缩操作

int solve(board B, stack S, int res, int[] dx, int[] dy):

递归求解 Peg 棋局

int peg_solve(board B, stack S):

peg 文件入口，预处理之后调用 solve

（2）peg3 求解模块函数说明

ht_lookup(ht H, htkey k):

判断 k 是否存在 hash table 中

ht_insert(ht H, htkey k):

将 k 插入 hash table 中

htelem trans(board B):

压缩棋盘 B 为两个整型

void change(htelem nowhtkey, int ord, int op):

得到改变一位后的压缩数据，优化常数用。

int solve(board B, stack S, ht H, int res, int[] dx, int[] dy):

加入 Hash 表进行记忆化搜索的 solve 函数

其它同 peg1、peg2

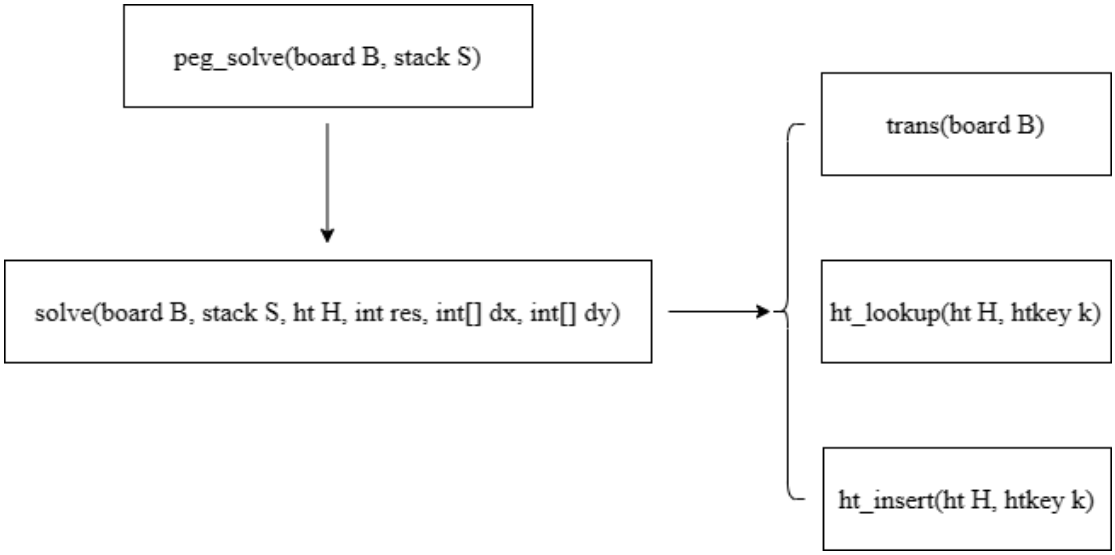

图 4-1 Peg3 求解模块函数调用示意图

（3）peg4 求解模块函数说明

在实现双端搜索时，求解只需要两步：一步 backward，一步 forward。

void pre_solve(board B, ht H, int nums):

从最终态往回走，nums 控制走的步数，H 记录所有走到的状态。

其它函数基本与 peg3 类似。

4.2 系统测试

首先叙述一下常用的软件测试方法，在选择几个主要的功能模块（自行掌握数量，关键要体现你的水平的一些模块）描述测试过程。（1）先明确模块的功能、设计目标等；（2）分析、叙述如何选取测试数据，要求有完整的测试大纲；（3）运行结果（这时可用截图）；（4）分析运行结果、确认程序满足该模块的设计目标。

4.2.1 软件测试方法

软件的测试方法常用的有以下几类：

（1）是否关心内部结构：

白盒测试，黑盒测试，灰盒测试。

（2）是否执行代码：

静态测试，动态测试。

（3）开发过程级别：

单元测试，集成测试，系统测试。

（4）是否需要人工：

手工测试，自动化测试。

（5）测试实施组织：

开发测试，用户测试，第三方测试。

（6）测试环境：

阿尔法测试，贝塔测试。

而在系统测试中，具体需要测试的类型有：功能测试，性能测试，接口测试，人机交互界面测试，强度测试，余量测试，安全性测试，可靠性测试，恢复性测试，边界测试，数据处理测试，安装性测试，容量测试，互操作性测试，敏感性测试，标准符合性测试，兼容性测试，中文本地化测试等。

4.2.2 测试大纲

测试目的：通过对已有数据的运行计算来验证系统是否已经达到设计指标，探索系统优化改进的方向。

测试环境：同系统实现部分，此处不再赘述。

测试方法：选择合适数量的具有代表性的测试用例，通过控制变量法，进行求解时间的测量与比较。进行对测试用例的特征分析，以及对程序运行的结果分析。

测试内容：Peg 格局求解性能测试。

4.2.3 Peg 格局求解性能测试

功能目标：

对所给棋局进行求解，有解的棋局需要给出具体操作，无解的棋局只需给出最少棋子个数即可。

比较 dfs 和记忆化 dfs 和记忆化+双端搜索算法的性能和各自的特点，在实际应用中可以灵活选取三种方法从而高效地求解问题。

测试方法：

执行所给编译运行指令即可

```
% cc0 -d -w -o peg* peg-client.c0 lib/*.c0 peg*.c0 peg-main.c0
```

% ./peg* *******.txt

测试数据：

利用已有数据，测试 english.txt、french1.txt、french2.txt、french3.txt.

运行结果：

首先，先测试 dfs、记忆化搜索 dfs 和双端搜索记忆化在各测试数据上的表现：

表 4-1 各算法所需时间

算法	English.txt	French1.txt	French2.txt	French3.txt
Dfs	1.042s	Unsolved	Unsolved	Unsolved
记忆化	2.424s	5m24.384s	2m16.881s	42. 862s
双端记忆化	7.822s	1m45.311s	2m15.064s	51.491s

表 4-2 各算法 unsolvable 棋盘数

算法	English.txt	French1.txt	French2.txt	French3.txt
dfs	Unknown	Unknown	Unknown	Unknown
记忆化	1034	86774664	20837945	7046060
双端记忆化	188	14920832	18904696	6129986

观察以上结果可得：

（1）最简单的深度优先搜索能解决简单的棋局如 English 版本，但对于复杂的棋局如 French 版本就显得无能为力了。在本机上 dfs 跑 French1.txt 跑了 2h 还未能跑出结果。

（2）记忆化搜索在深度优先搜索的基础上加上了记忆化剪枝，大大缩减了时间和 unsolvable 状态数，在 French*.txt 上能取得较好的成绩，但仍然显得效率低。

（3）双端记忆化搜索在记忆化搜索基础上加上了双端优化，极大程度地缩减了树高，以空间换时间，在（2）的程度上更加缩减了 unsolvable 状态数。是目前的最优解。

改进结果 1：

在测试过程中，注意到记忆化算法在不同的棋盘上时间差别巨大，合理怀疑各棋盘的框架和初态会影响结果，那么能否改变搜索顺序以找到最适合当前棋盘的顺序呢？答案是可以的。首先我们改变方向选择上的遍历：

$$dx[0] = 1;\ dx[1] = 0;\ dx[2] = -1;\ dx[3] = 0;$$
$$dy[0] = 0;\ dy[1] = 1;\ dy[2] = 0;\ dy[3] = -1;$$

我们称呼上述方向为“下右上左”。那么我们考虑修改方向的顺序，为方便控制变量，故只在记忆化搜索上跑：

表 4-3 各方向所需时间

方向	English.txt	French1.txt	French2.txt	French3.txt
下右上左	0.004s	5m21.841s	56.022s	2.806s
下右左上	0.002s	43.814s	56.876s	16.496s
上左右下	0.003s	5m25.728s	58.790s	2.877s

表 4-4 各方向 unsolvable 棋盘数

方向	English.txt	French1.txt	French2.txt	French3.txt
下右上左	10813	81093756	20838004	1637532
下右左上	1034	16639920	20838004	7046060
上左右下	10813	81093756	20837945	1637532

观察结果可以发现，当方向优先顺序不同时，会导致时间和unsolvable棋盘数不同，而且不同的棋盘最适合的方向也不尽相同，所以需要根据棋盘的特征来选择最合适的方向。

因为当前的顺序是$x:0\to 7, y:0\to 7$，是从左上到右下，所以我们优先往右下走在大部分情况都表现更优秀，这也很好地解释了为什么“下右左上”顺序在French.txt表现最好。

改进结果 2：

既然知道方向的优先级会影响结果，那么自然而然地，遍历的顺序也会大幅影响结果。原来的顺序称为normal，是$x:0\to7,y:0\to7$，但分析棋盘的特性可得，理论上应该让棋子尽量在中间，这样棋子之间更加靠近，也会更容易操作。

所以我改进了遍历顺序，改为从外往内。即先遍历离边界近的点，再遍历离边界远的点，我称这种顺序为out-in。相反，从内往外的顺序称为in-out。以下是各顺序和各方向对应的结果：

表 4-5 各顺序、各方向所需时间

顺序	方向	French1.txt	French2.txt	French3.txt
Normal	下右上左	5m21.841s	56.022s	2.806s
	下右左上	43.814s	56.876s	16.496s
	上左右下	5m25.728s	58.790s	2.877s
Out-in	下右上左	2m14.374s	16.401s	2.935s
	下右左上	15.466s	16.457s	3.597s
	上左右下	2m16.480s	16.379s	2.884s
In-out	下右上左	5m26.232s	2m20.177s	43.007s
	下右左上	5m35.357s	2m17.725s	44.299s
	上左右下	5m27.886s	2m17.193s	43.501s

表 4-6 各顺序、各方向 unsolvable 棋盘数

顺序	方向	French1.txt	French2.txt	French3.txt
Normal	下右上左	81093756	20838004	1637532
	下右左上	16639920	20838004	7046060
	上左右下	81093756	20837945	1637532
Out-in	下右上左	47387138	7847221	2076983
	下右左上	8028362	7847221	2372085
	上左右下	47390876	7847261	2076957
In-out	下右上左	86774664	43051576	17570704
	下右左上	86774664	43051576	17570704
	上左右下	86774664	43051577	17570705

分析以上结果可知，当遍历顺序由原来的normal变成out-in时，在各个方向上基本都有较大的提升，甚至在“下右左上”方向上直接把unsolvable状态数缩减至8e7，只需要15s就能跑出结果。

而与之相对的是，当采用in-out顺序时，在各个方向上都比normal性能上大幅下降。In-out 和 out-in 的对比也证明了我的猜测：应该优先削减外围的棋子，再考虑内部棋子。

5 总结

（1）我们完美地完成了 peg1、peg2、peg3 程序的设计和实现，并且在相关数据类型的设计方面也达到了高水平。具体来说，我们设计并实现了诸如 move、stackelem、htelem 等数据类型，确保了程序的健壮性和扩展性。

（2） 然而，在实际测试中，我们发现程序的性能较低。为了提高性能，我们利用搜索引擎查找相关资料，最终找到了一篇相关论文[1]。经过仔细阅读，我们发现论文中介绍的双端搜索优化方法非常适合我们的需求。于是，我们将这种方法应用到程序中，取得了显著的性能提升。

（3）此外，在阅读 performance-debugging.txt 文档后，我们对程序的遍历顺序和 move 方向优先级进行了深入分析和优化。基于对棋盘特性和相关联系的研究，我们猜测最优的遍历顺序和 move 方向分别是 out-in 和下右左上。通过控制变量法，我们进行了多组数据测试，验证了这一猜想。最终，这些优化措施使程序的性能提升了高达 20 倍，显著提高了程序的效率。

6 体会

这次实验起初看起来非常复杂，超长的题目描述、详尽的框架约束，以及复杂又完善的库函数，让人望而生畏。然而，当我仔细阅读题目后，发现其实只需要实现深度优先搜索（DFS）部分即可。

在真正实现 DFS 后，我却发现程序的性能很低，通常需要好几分钟才能得到一个解。优化 DFS 的方法五花八门，这考验了人类的智慧和耐心。面对这一挑战，我需要不断地分析问题特征，寻找合适的优化方法，并进行反复尝试和测试。

优化往往是多个方向的组合，需要尝试许多次才能找到最优解。此外，常见的优化方法并不总是有效，这就需要借鉴前人的经验和智慧。因此，我找了一篇相关论文，花了整个五一假期仔细阅读。在阅读过程中，我摒弃了那些复杂且对这个任务不太有效的优化方法，最终找到了双端搜索这一优雅且高效的优化方案。

在实验过程中，我也得到了老师的大力支持。老师耐心地解答了我提出的各种问题，帮助我更好地理解和应用所学知识。

通过这次实验，我不仅掌握了 DFS 的实现和优化方法，还深刻体会到了科学研究和问题解决的过程。在面对复杂问题时，冷静分析、借鉴前人经验、不断尝试和测试，最终可以找到有效的解决方案。这些经验将对我今后的学习和工作产生深远的影响。

这次实验让我明白了，面对复杂的任务，不能被表面的困难吓倒，而是要通过仔细分析和不懈努力，找到解决问题的最佳途径。希望在未来的学习和实践中，我能不断应用和深化这些经验，进一步提升自己的综合能力。

参考文献

[1] Kiyomi, M., Matsui, T. (2001). Integer Programming Based Algorithms for Peg Solitaire Problems. In: Marsland, T., Frank, I. (eds) Computers and Games. CG 2000. Lecture Notes in Computer Science, vol 2063. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-45579-5_15

附录

Peg2 标准深度优先搜索

```
int row_start(move m){
   int x = m / 1000;
   x %= 10;
   return x;
}

int col_start(move m){
   int y = m / 100;
   y %= 10;
   return y;
}

int row_end(move m){
   int x = m /10;
   x %= 10;
   return x;
}

int col_end(move m){
   int y = m;
   return y%10;
}

int to(int r, int c){
   return r*8 + c;
}

move op(int r_s, int c_s, int r_e, int c_e)
{
   return r_s*1000 + c_s*100 + r_e*10 + c_e;
}

bool chk(board A, board B)
//@requires \length(A) == \length(B);
//@requires \length(A) == 64;
{
   for(int i = 0; i < 64; i++){
      if(A[i] != B[i])return false;
   }
   return true;
}

int solve(board B, stack S, int res, int[] dx, int[] dy)
//@requires is_board(B);
//@requires num_pegs(B) >= 1;
//@requires is_stack(S);
//@ensures is_board(B);
{
   if(res == 1) return 1;
   move now = 0;
   int least = res;
   board A = alloc_array(int, 64);
   for(int i = 0; i < 64; i++)A[i] = B[i];
   for(int x = 0; x < 8; x++){
      for(int y = 0; y < 8; y++){
         if(B[to(x, y)] == 1){
            for(int i = 0; i < 4; i++)
            //@loop_invariant chk(A, B);
            {
               int tx = x + dx[i];
```

```
            int ty = y + dy[i];
            if(0 <= tx && tx < 8 && 0 <= ty && ty < 8 && B[to(tx, ty)] == 1){
               tx += dx[i];    ty += dy[i];
               if(0 <= tx && tx < 8 && 0 <= ty && ty < 8 && B[to(tx, ty)] == 0){
                  now = op(x, y, tx, ty);
                  B[to(x, y)] = 0;
                  B[to(x + dx[i], y + dy[i])] = 0;
                  B[to(tx, ty)] = 1;
                  least = min(least, solve(B, S, res-1, dx, dy));
                  if(least == 1){
                     push(S, now);
                     return 1;
                  }
                  B[to(x, y)] = 1;
                  B[to(x + dx[i], y + dy[i])] = 1;
                  B[to(tx, ty)] = 0;
               }
            }
         }
      }
   }
   return least;
}

int peg_solve(board B, stack S)
//@requires is_board(B);
//@requires num_pegs(B) >= 1;
//@requires stack_empty(S);
//@ensures is_board(B);
//@ensures \result >= 1;
{
   int[] dx = alloc_array(int, 4);
   int[] dy = alloc_array(int, 4);
   dx[0] = -1;    dx[1] = 0;    dx[2] = 0;    dx[3] = 1;
   dy[0] = 0;    dy[1] = -1;    dy[2] = 1;    dy[3] = 0;
   int ans = solve(B, S, num_pegs(B), dx, dy);
   return ans;
}
```

Peg3 记忆化深度优先搜索

```
int row_start(move m){
   int x = m / 1000;
   x %= 10;
   return x;
}

int col_start(move m){
   int y = m / 100;
   y %= 10;
   return y;
}

int row_end(move m){
   int x = m /10;
   x %= 10;
   return x;
}

int col_end(move m){
   int y = m;
   return y%10;
}
```

```
int to(int r, int c){
   return r*8 + c;
}

move op(int r_s, int c_s, int r_e, int c_e)
{
   return r_s*1000 + c_s*100 + r_e*10 + c_e;
}

bool chk(board A, board B)
//@requires \length(A) == \length(B);
//@requires \length(A) == 64;
{
   for(int i = 0; i < 64; i++){
      if(A[i] != B[i])return false;
   }
   return true;
}

htelem trans(board B)
//@requires is_board(B);
{
   htelem now = alloc(struct two_ints);
   now->i1 = 0;
   for(int i = 0; i < 32; i++){
      if(B[i] == 1)now->i1 = now->i1 * 2 + 1;
      else now->i1 = now->i1 * 2;
   }
   now->i2 = 0;
   for(int i = 32; i < 64; i++){
      if(B[i] == 1)now->i2 = now->i2 * 2 + 1;
      else now->i2 = now->i2 * 2;
   }
   now->best_num_pegs = num_pegs(B);
   return now;
}

int solve1(board B, stack S, ht H, int res, int[] dx, int[] dy)
//@requires is_board(B);
//@requires is_ht(H);
//@requires num_pegs(B) >= 1;
//@requires is_stack(S);
//@ensures is_board(B);
{
   if(res == 1) return 1;
   htelem nowhtkey = trans(B);
   htelem tofind = ht_lookup(H, nowhtkey);
   if(tofind != NULL)return tofind->best_num_pegs;
   move now = 0;
   int least = res;
//     board A = alloc_array(int, 64);
//     for(int i = 0; i < 64; i++)A[i] = B[i];
   for(int x = 0; x < 8; x++){
      for(int y = 0; y < 8; y++){
         if(B[to(x, y)] == 1){
            for(int i = 0; i < 4; i++)
            // @loop_invariant chk(A, B);
            {
               int tx = x + dx[i];
               int ty = y + dy[i];
               if(0 <= tx && tx < 8 && 0 <= ty && ty < 8
               && B[to(tx, ty)] == 1){
                  tx += dx[i];    ty += dy[i];
                  if(0 <= tx && tx < 8 && 0 <= ty && ty < 8
                  && B[to(tx, ty)] == 0){
                     now = op(x, y, tx, ty);
```

```
                    B[to(x, y)] = 0;
                    B[to(x + dx[i], y + dy[i])] = 0;
                    B[to(tx, ty)] = 1;
                    least = min(least, solve1(B, S, H, res-1, dx, dy));
                    if(least == 1){
                       push(S, now);
                       return 1;
                    }
                    B[to(x, y)] = 1;
                    B[to(x + dx[i], y + dy[i])] = 1;
                    B[to(tx, ty)] = 0;
                 }
              }
           }
        }
     }
   }
   if(least != 1){
      nowhtkey->best_num_pegs = least;
      ht_insert(H, nowhtkey);
   }
   return least;
}

int solve2(board B, stack S, ht H, int res, int[] dx, int[] dy, int[] order)
//@requires is_board(B);
//@requires is_ht(H);
//@requires num_pegs(B) >= 1;
//@requires is_stack(S);
//@ensures is_board(B);
{
   if(res == 1) return 1;
   htelem nowhtkey = trans(B);
   htelem tofind = ht_lookup(H, nowhtkey);
   if(tofind != NULL)return tofind->best_num_pegs;
   // ht_insert(H, nowhtkey);
   move now = 0;
   int least = res;
//     board A = alloc_array(int, 64);
//     for(int i = 0; i < 64; i++)A[i] = B[i];
   for(int index = 0; index < 64; index++){
      int x = order[index] / 8;
      int y = order[index] % 8;
      if(B[to(x, y)] == 1){
            for(int i = 0; i < 4; i++)
            // @loop_invariant chk(A, B);
            {
                  int tx = x + dx[i];
                  int ty = y + dy[i];
                  if(0 <= tx && tx < 8 && 0 <= ty && ty < 8
                  && B[to(tx, ty)] == 1){
                        tx += dx[i];    ty += dy[i];
                        if(0 <= tx && tx < 8 && 0 <= ty && ty < 8
                        && B[to(tx, ty)] == 0){
                              now = op(x, y, tx, ty);
                              B[to(x, y)] = 0;
                              B[to(x + dx[i], y + dy[i])] = 0;
                              B[to(tx, ty)] = 1;
                              least = min(least, solve2(B, S, H, res-1, dx, dy, order));
                              if(least == 1){
                                 push(S, now);
                                 return 1;
                              }
                              B[to(x, y)] = 1;
                              B[to(x + dx[i], y + dy[i])] = 1;
                              B[to(tx, ty)] = 0;
                        }
```

```
                }
            }
        }
    }
    if(least != 1){
        nowhtkey->best_num_pegs = least;
        ht_insert(H, nowhtkey);
    }
    return least;
}

int getnei(board B, int x, int y, int[] dx, int[] dy){
        int cnt=0;
        for(int i = 0; i < 4; i++){
            int tx = x + dx[i];
            int ty = y + dy[i];
            if(tx >= 0 && tx < 8 && ty >= 0 &&ty < 8 && B[tx*8 + ty] == 1){
                cnt++;
            }
        }
        return cnt;
}

int solve3(board B, stack S, ht H, int res, int[] dx, int[] dy, int[] order)
//@requires is_board(B);
//@requires is_ht(H);
//@requires num_pegs(B) >= 1;
//@requires is_stack(S);
//@ensures is_board(B);
{
    if(res == 1) return 1;
    htelem nowhtkey = trans(B);
    htelem tofind = ht_lookup(H, nowhtkey);
    if(tofind != NULL)return tofind->best_num_pegs;
    move now = 0;
    int least = res;
//      board A = alloc_array(int, 64);
//      for(int i = 0; i < 64; i++)A[i] = B[i];
    for(int len = 4; len > 0; len--){
        for(int index = 0; index < 49; index++){
            int x = order[index] / 8;
            int y = order[index] % 8;
            if(getnei(B, x, y, dx, dy) == len){
                if(B[to(x, y)] == 1){
                    for(int i = 0; i < 4; i++)
                    // @loop_invariant chk(A, B);
                    {
                        int tx = x + dx[i];
                        int ty = y + dy[i];
                        if(0 <= tx && tx < 8 && 0 <= ty && ty < 8
                        && B[to(tx, ty)] == 1){
                            tx += dx[i];    ty += dy[i];
                            if(0 <= tx && tx < 8 && 0 <= ty && ty < 8
                            && B[to(tx, ty)] == 0){
                                now = op(x, y, tx, ty);
                                B[to(x, y)] = 0;
                                B[to(x + dx[i], y + dy[i])] = 0;
                                B[to(tx, ty)] = 1;
                                least = min(least, solve3(B, S, H, res-1, dx, dy, order));
                                if(least == 1){
                                    push(S, now);
                                    return 1;
                                }
                                B[to(x, y)] = 1;
                                B[to(x + dx[i], y + dy[i])] = 1;
                                B[to(tx, ty)] = 0;
                            }
```

```
                    }
                }
            }
        }
    }
    if(least != 1){
        nowhtkey->best_num_pegs = least;
        ht_insert(H, nowhtkey);
    }
    return least;
}

int peg_solve(board B, stack S)
//@requires is_board(B);
//@requires num_pegs(B) >= 1;
//@requires stack_empty(S);
//@ensures is_board(B);
//@ensures \result >= 1;
{
    int[] dx = alloc_array(int, 4);
    int[] dy = alloc_array(int, 4);
    // dx[0] = 1;    dx[1] = 0;    dx[2] = -1;    dx[3] = 0;
    // dy[0] = 0;    dy[1] = 1;    dy[2] = 0;    dy[3] = -1;

    // dx[0] = 0;    dx[1] = 1;    dx[2] = -1;    dx[3] = 0;
    // dy[0] = 1;    dy[1] = 0;    dy[2] = 0;    dy[3] = -1;

    dx[0] = -1;    dx[1] = 1;    dx[2] = 0;    dx[3] = 0;
    dy[0] = 0;    dy[1] = 0;    dy[2] = -1;    dy[3] = 1;
    ht H = ht_new(133331);

    int[] order = alloc_array(int, 8*8);
    int index = 0;
    for(int len = 0; len < 5; len++)
        for(int i = 0; i < 7; i++){
            for(int j = 0; j < 7; j++){
                if(min(min(i, 7-i), min(j, 7-j)) == len){
                    order[index] = i*8 + j;
                    index++;
                }
            }
        }
    // int ans = solve3(B, S, H, num_pegs(B), dx, dy, order);
    int ans = solve1(B, S, H, num_pegs(B), dx, dy);
    // int ans = solve2(B, S, H, num_pegs(B), dx, dy, order);

    printint(ht_size(H));
    println("\n");
    ht_stats(H);
    return ans;
}
```

Peg4 记忆化+优化遍历顺序深度优先搜索

```
int row_start(move m){
    int x = m / 1000;
    x %= 10;
    return x;
}

int col_start(move m){
    int y = m / 100;
    y %= 10;
    return y;
```

```
}

int row_end(move m){
   int x = m /10;
   x %= 10;
   return x;
}

int col_end(move m){
   int y = m;
   return y%10;
}

int to(int r, int c){
   return r*8 + c;
}

move op(int r_s, int c_s, int r_e, int c_e)
{
   return r_s*1000 + c_s*100 + r_e*10 + c_e;
}

bool chk(board A, board B)
//@requires \length(A) == \length(B);
//@requires \length(A) == 64;
{
   for(int i = 0; i < 64; i++){
      if(A[i] != B[i])return false;
   }
   return true;
}

htelem trans(board B)
//@requires is_board(B);
{
   htelem now = alloc(struct two_ints);
   now->i1 = 0;
   for(int i = 0; i < 32; i++){
      if(B[i] == 1)now->i1 = now->i1 * 2 + 1;
      else now->i1 = now->i1 * 2;
   }
   now->i2 = 0;
   for(int i = 32; i < 64; i++){
      if(B[i] == 1)now->i2 = now->i2 * 2 + 1;
      else now->i2 = now->i2 * 2;
   }
   now->best_num_pegs = num_pegs(B);
   return now;
}

void change(htelem nowhtkey, int ord, int op){
   if(ord <= 31){
      nowhtkey->i1 += op*(1 << (31-ord));
   }else{
      nowhtkey->i2 += op*(1 << (63-ord));
   }
}

int solve1(board B, stack S, ht H, int res, int[] dx, int[] dy,
                  htelem Bhtkey)
//@requires is_board(B);
//@requires is_ht(H);
//@requires num_pegs(B) >= 1;
//@requires htkey_equal(Bhtkey, trans(B));
//@requires is_stack(S);
//@ensures is_board(B);
//@ensures htkey_equal(Bhtkey, trans(B));
```

```
{
   if(res == 1) return 1;
   htelem nowhtkey = alloc(struct two_ints);
   nowhtkey->i1 = Bhtkey->i1;
   nowhtkey->i2 = Bhtkey->i2;
   htelem tofind = ht_lookup(H, nowhtkey);
   if(tofind != NULL)return tofind->best_num_pegs;
   move now = 0;
   int least = res;
//     board A = alloc_array(int, 64);
//     for(int i = 0; i < 64; i++)A[i] = B[i];
   for(int x = 0; x < 8; x++){
      for(int y = 0; y < 8; y++){
         if(B[to(x, y)] == 1){
            for(int i = 0; i < 4; i++)
            // @loop_invariant chk(A, B);
            {
               int tx = x + dx[i];
               int ty = y + dy[i];
               if(0 <= tx && tx < 8 && 0 <= ty && ty < 8
               && B[to(tx, ty)] == 1){
                  tx += dx[i];    ty += dy[i];
                  if(0 <= tx && tx < 8 && 0 <= ty && ty < 8
                  && B[to(tx, ty)] == 0){
                     now = op(x, y, tx, ty);
                     B[to(x, y)] = 0;
                     B[to(x + dx[i], y + dy[i])] = 0;
                     B[to(tx, ty)] = 1;
                     change(Bhtkey, x*8 + y, -1);
                     change(Bhtkey, (x+dx[i])*8 + y+dy[i], -1);
                     change(Bhtkey, tx*8 + ty, 1);
                     // int tmp = Bhtkey->best_num_pegs;
                     // Bhtkey->best_num_pegs -= 1;
                     least = min(least, solve1(B, S, H, res-1, dx, dy, Bhtkey));
                     if(least == 1){
                        push(S, now);
                        return 1;
                     }
                     change(Bhtkey, x*8 + y, 1);
                     change(Bhtkey, (x+dx[i])*8 + y+dy[i], 1);
                     change(Bhtkey, tx*8 + ty, -1);
                     // Bhtkey->best_num_pegs = tmp;
                     B[to(x, y)] = 1;
                     B[to(x + dx[i], y + dy[i])] = 1;
                     B[to(tx, ty)] = 0;
                  }
               }
            }
         }
      }
   }
   if(least != 1){
      nowhtkey->best_num_pegs = least;
      ht_insert(H, nowhtkey);
   }
   return least;
}

int solve2(board B, stack S, ht H, int res, int[] dx, int[] dy, int[] order,
                  htelem Bhtkey)
//@requires is_board(B);
//@requires is_ht(H);
//@requires num_pegs(B) >= 1;
//@requires is_stack(S);
//@ensures is_board(B);
{
   if(res == 1) return 1;
```

```
    htelem nowhtkey = alloc(struct two_ints);
    nowhtkey->i1 = Bhtkey->i1;
    nowhtkey->i2 = Bhtkey->i2;
    htelem tofind = ht_lookup(H, nowhtkey);
    if(tofind != NULL)return tofind->best_num_pegs;
    // ht_insert(H, nowhtkey);
    move now = 0;
    int least = res;
//      board A = alloc_array(int, 64);
//      for(int i = 0; i < 64; i++)A[i] = B[i];
    for(int index = 0; index < 64; index++){
        int x = order[index] / 8;
        int y = order[index] % 8;
        if(B[to(x, y)] == 1){
            for(int i = 0; i < 4; i++)
            // @loop_invariant chk(A, B);
            {
                int tx = x + dx[i];
                int ty = y + dy[i];
                if(0 <= tx && tx < 8 && 0 <= ty && ty < 8
                && B[to(tx, ty)] == 1){
                    tx += dx[i];    ty += dy[i];
                    if(0 <= tx && tx < 8 && 0 <= ty && ty < 8
                    && B[to(tx, ty)] == 0){
                        now = op(x, y, tx, ty);
                        B[to(x, y)] = 0;
                        B[to(x + dx[i], y + dy[i])] = 0;
                        B[to(tx, ty)] = 1;
                        change(Bhtkey, x*8 + y, -1);
                        change(Bhtkey, (x+dx[i])*8 + y+dy[i], -1);
                        change(Bhtkey, tx*8 + ty, 1);
                        least = min(least, solve2(B, S, H, res-1, dx, dy, order,
                                                  Bhtkey));
                        if(least == 1){
                            push(S, now);
                            return 1;
                        }
                        change(Bhtkey, x*8 + y, 1);
                        change(Bhtkey, (x+dx[i])*8 + y+dy[i], 1);
                        change(Bhtkey, tx*8 + ty, -1);
                        B[to(x, y)] = 1;
                        B[to(x + dx[i], y + dy[i])] = 1;
                        B[to(tx, ty)] = 0;
                    }
                }
            }
        }
    }
    if(least != 1){
        nowhtkey->best_num_pegs = least;
        ht_insert(H, nowhtkey);
    }
    return least;
}

int getnei(board B, int x, int y, int[] dx, int[] dy){
        int cnt=0;
        for(int i = 0; i < 4; i++){
            int tx = x + dx[i];
            int ty = y + dy[i];
            if(tx >= 0 && tx < 8 && ty >= 0 &&ty < 8 && B[tx*8 + ty] == 1){
                cnt++;
            }
        }
        return cnt;
}
```

```
int solve3(board B, stack S, ht H, int res, int[] dx, int[] dy, int[] order)
//@requires is_board(B);
//@requires is_ht(H);
//@requires num_pegs(B) >= 1;
//@requires is_stack(S);
//@ensures is_board(B);
{
   if(res == 1) return 1;
   htelem nowhtkey = trans(B);
   htelem tofind = ht_lookup(H, nowhtkey);
   if(tofind != NULL)return tofind->best_num_pegs;
   move now = 0;
   int least = res;
//     board A = alloc_array(int, 64);
//     for(int i = 0; i < 64; i++)A[i] = B[i];
   for(int len = 4; len > 0; len--){
      for(int index = 0; index < 49; index++){
            int x = order[index] / 8;
            int y = order[index] % 8;
            if(getnei(B, x, y, dx, dy) == len){
                  if(B[to(x, y)] == 1){
                     for(int i = 0; i < 4; i++)
                     // @loop_invariant chk(A, B);
                     {
                        int tx = x + dx[i];
                        int ty = y + dy[i];
                        if(0 <= tx && tx < 8 && 0 <= ty && ty < 8
                        && B[to(tx, ty)] == 1){
                           tx += dx[i];    ty += dy[i];
                           if(0 <= tx && tx < 8 && 0 <= ty && ty < 8
                           && B[to(tx, ty)] == 0){
                              now = op(x, y, tx, ty);
                              B[to(x, y)] = 0;
                              B[to(x + dx[i], y + dy[i])] = 0;
                              B[to(tx, ty)] = 1;
                              least = min(least, solve3(B, S, H, res-1, dx, dy, order));
                              if(least == 1){
                                 push(S, now);
                                 return 1;
                              }
                              B[to(x, y)] = 1;
                              B[to(x + dx[i], y + dy[i])] = 1;
                              B[to(tx, ty)] = 0;
                           }
                        }
                     }
                  }
            }
      }
   }
   if(least != 1){
      nowhtkey->best_num_pegs = least;
      ht_insert(H, nowhtkey);
   }
   return least;
}

int peg_solve(board B, stack S)
//@requires is_board(B);
//@requires num_pegs(B) >= 1;
//@requires stack_empty(S);
//@ensures is_board(B);
//@ensures \result >= 1;
{
   int[] dx = alloc_array(int, 4);
   int[] dy = alloc_array(int, 4);
   dx[0] = 1;    dx[1] = 0;    dx[2] = 0;    dx[3] = -1;
```

```
    dy[0] = 0;    dy[1] = 1;    dy[2] = -1;    dy[3] = 0;

    // dx[0] = 1;    dx[1] = 0;    dx[2] = -1;    dx[3] = 0;
    // dy[0] = 0;    dy[1] = 1;    dy[2] = 0;    dy[3] = -1;

    // dx[0] = 0;    dx[1] = 1;    dx[2] = -1;    dx[3] = 0;
    // dy[0] = 1;    dy[1] = 0;    dy[2] = 0;    dy[3] = -1;

    // dx[0] = -1;    dx[1] = 1;    dx[2] = 0;    dx[3] = 0;
    // dy[0] = 0;    dy[1] = 0;    dy[2] = -1;    dy[3] = 1;

    // dx[0] = -1;    dx[1] = 0;    dx[2] = 0;    dx[3] = 1;
    // dy[0] = 0;    dy[1] = -1;    dy[2] = 1;    dy[3] = 0;
    ht H = ht_new(133331);

    int[] order = alloc_array(int, 8*8);
    int index = 0;
    for(int len = 5; len >=0; len--)
        for(int i = 0; i < 7; i++){
            for(int j = 0; j < 7; j++){
                if(min(min(i, 7-i), min(j, 7-j)) == len){
                    order[index] = i*8 + j;
                    index++;
                }
            }
        }
    // int ans = solve3(B, S, H, num_pegs(B), dx, dy, order);
    // int ans = solve1(B, S, H, num_pegs(B), dx, dy, trans(B));
    int ans = solve2(B, S, H, num_pegs(B), dx, dy, order, trans(B));

    printint(ht_size(H));
    println("");
    // println("\n");
    // printint(ans);
    // println("\n");
    // ht_stats(H);
    return ans;
}
```

Peg5 记忆化+双端优化深度优先搜索

```
int row_start(move m){
    int x = m / 1000;
    x %= 10;
    return x;
}

int col_start(move m){
    int y = m / 100;
    y %= 10;
    return y;
}

int row_end(move m){
    int x = m /10;
    x %= 10;
    return x;
}

int col_end(move m){
    int y = m;
    return y%10;
}

int to(int r, int c){
```

```
   return r*8 + c;
}

move op(int r_s, int c_s, int r_e, int c_e)
{
   return r_s*1000 + c_s*100 + r_e*10 + c_e;
}

bool chk(board A, board B)
//@requires \length(A) == \length(B);
//@requires \length(A) == 64;
{
   for(int i = 0; i < 64; i++){
      if(A[i] != B[i])return false;
   }
   return true;
}

htelem trans(board B)
//@requires is_board(B);
{
   htelem now = alloc(struct two_ints);
   now->i1 = 0;
   for(int i = 0; i < 32; i++){
      if(B[i] == 1)now->i1 = now->i1 * 2 + 1;
      else now->i1 = now->i1 * 2;
   }
   now->i2 = 0;
   for(int i = 32; i < 64; i++){
      if(B[i] == 1)now->i2 = now->i2 * 2 + 1;
      else now->i2 = now->i2 * 2;
   }
   now->best_num_pegs = num_pegs(B);
   return now;
}

int solve1(board B, stack S, ht H, int res, int[] dx, int[] dy,
                    ht H_B, board fin)
//@requires is_board(B);
//@requires is_ht(H);
//@requires num_pegs(B) >= 1;
//@requires is_stack(S);
//@ensures is_board(B);
{
   htelem nowhtkey = trans(B);
   // if(res < 3)printint(res);
   if(res == 0){
      htelem B_to_find = ht_lookup(H_B, nowhtkey);
      if(B_to_find != NULL){
         // fin = alloc_array(int, 64);
         for(int i = 0; i < 64; i++)fin[i] = B[i];
         // print_board(fin);
         return B_to_find->best_num_pegs;
      }
      return 64;
   }
   htelem tofind = ht_lookup(H, nowhtkey);
   if(tofind != NULL)return tofind->best_num_pegs;
   move now = 0;
   int least = num_pegs(B);
//    board A = alloc_array(int, 64);
//    for(int i = 0; i < 64; i++)A[i] = B[i];
   for(int x = 0; x < 8; x++){
      for(int y = 0; y < 8; y++){
         if(B[to(x, y)] == 1){
            for(int i = 0; i < 4; i++)
            // @loop_invariant chk(A, B);
```

```
            {
               int tx = x + dx[i];
               int ty = y + dy[i];
               if(0 <= tx && tx < 8 && 0 <= ty && ty < 8
               && B[to(tx, ty)] == 1){
                  tx += dx[i];    ty += dy[i];
                  if(0 <= tx && tx < 8 && 0 <= ty && ty < 8
                  && B[to(tx, ty)] == 0){
                     now = op(x, y, tx, ty);
                     B[to(x, y)] = 0;
                     B[to(x + dx[i], y + dy[i])] = 0;
                     B[to(tx, ty)] = 1;
                     least = min(least, solve1(B, S, H, res-1, dx, dy, H_B, fin));
                     if(least == 1){
                        push(S, now);
                        return 1;
                     }
                     B[to(x, y)] = 1;
                     B[to(x + dx[i], y + dy[i])] = 1;
                     B[to(tx, ty)] = 0;
                  }
               }
            }
         }
      }
   }
   if(least != 1){
      nowhtkey->best_num_pegs = least;
      ht_insert(H, nowhtkey);
   }
   return least;
}

void search(board B, ht H, int res, int[] dx, int[] dy)
//@requires is_board(B);
//@requires is_ht(H);
//@requires num_pegs(B) >= 1;
//@ensures is_board(B);
{

   htelem nowhtkey = trans(B);
   htelem tofind = ht_lookup(H, nowhtkey);
   if(tofind != NULL)return ;
   nowhtkey->best_num_pegs = 1;
   ht_insert(H, nowhtkey);
   if(res == 0) return ;
   // move now = 0;
   for(int x = 0; x < 8; x++){
      for(int y = 0; y < 8; y++){
         if(B[to(x, y)] == 1){
            for(int i = 0; i < 4; i++)
            // @loop_invariant chk(A, B);
            {
               int tx = x + dx[i];
               int ty = y + dy[i];
               if(0 <= tx && tx < 8 && 0 <= ty && ty < 8
               && B[to(tx, ty)] == 0){
                  tx += dx[i];    ty += dy[i];
                  if(0 <= tx && tx < 8 && 0 <= ty && ty < 8
                  && B[to(tx, ty)] == 0){
                     // now = op(x, y, tx, ty);
                     B[to(x, y)] = 0;
                     B[to(x + dx[i], y + dy[i])] = 1;
                     B[to(tx, ty)] = 1;
                     // least = min(least, solve1(B, S, H, res-1, dx, dy));
                     // if(least == 1){
                     //      push(S, now);
```

```
                    //      return 1;
                    // }
                    search(B, H, res-1, dx, dy);
                    B[to(x, y)] = 1;
                    B[to(x + dx[i], y + dy[i])] = 0;
                    B[to(tx, ty)] = 0;
                }
            }
        }
    }
}
// if(least != 1){
//      nowhtkey->best_num_pegs = least;
//      ht_insert(H, nowhtkey);
// }
// return least;
return ;
}

int solve2(board B, stack S, ht H, int res, int[] dx, int[] dy)
//@requires is_board(B);
//@requires is_ht(H);
//@requires num_pegs(B) >= 1;
//@requires is_stack(S);
//@ensures is_board(B);
{
    htelem nowhtkey = trans(B);
    if(res == 0){
        return 1;
    }
    htelem tofind = ht_lookup(H, nowhtkey);
    if(tofind != NULL)return tofind->best_num_pegs;
    move now = 0;
    int least = num_pegs(B);
//      board A = alloc_array(int, 64);
//      for(int i = 0; i < 64; i++)A[i] = B[i];
    for(int x = 0; x < 8; x++){
        for(int y = 0; y < 8; y++){
            if(B[to(x, y)] == 1){
                for(int i = 0; i < 4; i++)
                // @loop_invariant chk(A, B);
                {
                    int tx = x + dx[i];
                    int ty = y + dy[i];
                    if(0 <= tx && tx < 8 && 0 <= ty && ty < 8
                    && B[to(tx, ty)] == 1){
                        tx += dx[i];    ty += dy[i];
                        if(0 <= tx && tx < 8 && 0 <= ty && ty < 8
                        && B[to(tx, ty)] == 0){
                            now = op(x, y, tx, ty);
                            B[to(x, y)] = 0;
                            B[to(x + dx[i], y + dy[i])] = 0;
                            B[to(tx, ty)] = 1;
                            least = min(least, solve2(B, S, H, res-1, dx, dy));
                            if(least == 1){
                                push(S, now);
                                return 1;
                            }
                            B[to(x, y)] = 1;
                            B[to(x + dx[i], y + dy[i])] = 1;
                            B[to(tx, ty)] = 0;
                        }
                    }
                }
            }
        }
    }
```

```
    }
    if(least != 1){
        nowhtkey->best_num_pegs = least;
        ht_insert(H, nowhtkey);
    }
    return least;
}

void pre_solve(board B, ht H, int nums)
//@requires is_ht(H);
//@requires is_board(B);
{
    int[] dx = alloc_array(int, 4);
    int[] dy = alloc_array(int, 4);
    dx[0] = -1;    dx[1] = 0;    dx[2] = 0;    dx[3] = 1;
    dy[0] = 0;    dy[1] = -1;    dy[2] = 1;    dy[3] = 0;
    board B_new = alloc_array(int, 8*8);
    for(int i = 0; i < 64; i++){
        for(int j = 0; j < 64; j++){
            if(B[j] == -1){
                B_new[j] = -1;
            }else{
                if(i == j){
                    B_new[j] = 1;
                }else{
                    B_new[j] = 0;
                }
            }
        }
        if(B[i] != -1)search(B_new, H, nums, dx, dy);
    }
}

int peg_solve(board B, stack S)
//@requires is_board(B);
//@requires num_pegs(B) >= 1;
//@requires stack_empty(S);
//@ensures is_board(B);
//@ensures \result >= 1;
{
    int[] dx = alloc_array(int, 4);
    int[] dy = alloc_array(int, 4);
    dx[0] = 1;    dx[1] = 0;    dx[2] = 0;    dx[3] = -1;
    dy[0] = 0;    dy[1] = 1;    dy[2] = -1;    dy[3] = 0;
    ht H = ht_new(133331);
    ht H_B = ht_new(133331);
    int nums = num_pegs(B);
    int pre_nums = 9;
    nums -= 1 + pre_nums;
    pre_solve(B, H_B, pre_nums);
    int[] fin = alloc_array(int, 8*8);
    int[] tmp = alloc_array(int, 8*8);
    int[] init_B = alloc_array(int, 64);
    for(int i = 0; i < 64; i++)init_B[i] = B[i];
    // int ans = solve3(B, S, H, num_pegs(B), dx, dy, order);
    int ans = solve1(B, stack_new(), ht_new(133331), nums, dx, dy, H_B, fin);
    // print_board(fin);
    // printint(ans);
    // solve2(fin, S, H, pre_nums, dx, dy);
    // solve1(init_B, S, H, nums, dx, dy, H_B, fin);


    printint(ht_size(H));
    println("\n");
    ht_stats(H);
    return ans;
}
```