

**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班**

**学 号： U202215072**

**姓 名： 黄子益**

**指导教师： 李开**

**报告日期： 2024年5月19日**

**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 1

**2　系统需求分析与总体设计 2**

2.1系统需求分析 2

2.2系统总体设计 2

**3　系统详细设计 4**

3.1有关数据结构的定义 4

3.2 主要算法设计 4

**4　系统实现与测试 5**

4.1系统实现 5

4.2系统测试 5

**5　总结 6**

**6　体会 7**

**附录 8**

# 1　课程设计任务

## 1.1 简介

孔跳棋是一种单人棋盘游戏，目标是在棋盘上只留下一个棋子，移除其他所有的棋子。游戏从某个初始棋盘开始，其中一些孔上有棋子。

每走一步都是选择一个棋子垂直或水平跳过相邻的一个棋子到一个孔上，并移除跳过的棋子。例如，在标准的英式初始棋盘中，有4种可能的移动，所有移动都以中心为终点。如图1-1所示，棋子从上方跳入中心后会导致右边的局面。

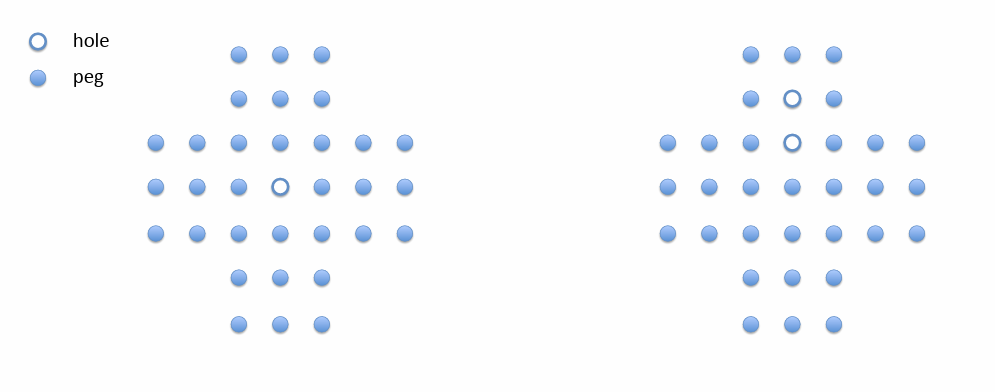


图1-1 孔明棋例子解释图

游戏的目标是最终只剩下一个棋子。在标准的英式棋盘上，我们开始时有32个棋子，因此所有解决方案都需要正好31步，因为每次跳跃都会移除一个棋子。

本课程要求使用C0语言编程对孔明棋棋局求解。

## 1.2 设计内容

棋盘初始状态由文件读入，编写三个不同效率的解孔明棋程序，分别为peg1.c0、peg2.c0、peg3.c0，效率依次提升。

我们采用递归的方式来解出孔明棋。例如在一个有32个棋子的棋盘上玩孔明棋，我们列举出在该棋盘上的所有有效移动（move）。然后，在进行其中的一个移动之后，我们再次解这个孔明棋——只是这次是在一个剩余31个棋子的棋盘上进行。

编写孔明棋求解算法以及要用到的相关数据结构，并对不同方式进行测试。

## 1.3 设计要求

能够快速解出较为简单的孔明棋局，对于较为复杂、难以计算的孔明棋局，可以尽量在1分钟内解出，同时占用空间不能太大。

# 2　系统需求分析与总体设计

## 2.1系统需求分析

通过此系统，我们可以从文件中读入初始棋盘，解出棋盘或告知最少能剩多少棋子。如果存在解，输出一种方案，否则仅输出可以剩下的最少棋子数到命令行中。

在此系统中，主要包括一些数据结构的库端和用户端，求解算法的实现，验证结果，以及一些交互界面输入输出的控制。

## 2.2系统总体设计

在lib/ht.c0、lib/stack.c0、lib/peg-util.c0中，是hash table、stack和读入输出棋盘的库端。

在peg-main.c0中，包含了主函数，用来读入棋盘并调用求解器，最后验证解，并将解输出。

在peg-client.c0中，完成对stack和hash table的用户端定义。

在peg\*.c0中，完成求解过程的具体实现。

系统结构数据图如图2-1所示。

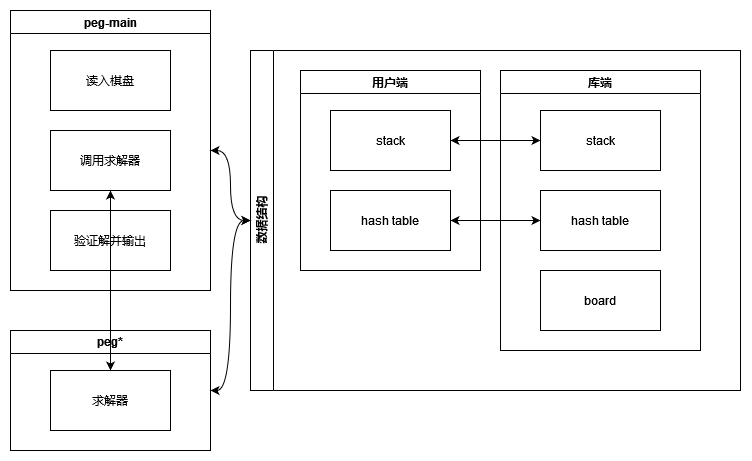


图2-1 系统模块结构图

# 3　系统详细设计

## 3.1有关数据结构的定义

此系统需要处理棋盘数据、行动数据、压缩后的棋盘数据以及栈内元素数据和哈希表中元素数据。

系统数据及数据类型如表3-1所示。

表 3-1 系统数据及数据类型

|  |  |  |
| --- | --- | --- |
| 数据 | 数据项 | 数据类型 |
| 行动 | 起始行 | 1个字节 |
| 起始列 | 1个字节 |
| 终点行 | 1个字节 |
| 终点列 | 1个字节 |
| 棋盘 | 棋盘布局 | 整型数组 |
| 压缩的棋盘 | 棋盘布局，  不包含阻挡格信息 | 2个整型 |
| 栈 | 栈顶 | 栈链表节点指针 |
| 栈底 | 栈链表节点指针 |
| 栈大小 | 整型 |
| 栈链表节点 | 栈元素 | 行动 |
| 链表下一个节点 | 栈链表节点指针 |
| 哈希表 | 大小 | 整型 |
| 容量 | 整型 |
| 表 | 哈希链表节点指针数组 |
| 哈希链表节点 | 哈希表元素 | 压缩的棋盘及对应数据 |
| 链表下一个节点 | 哈希表链表节点指针 |

棋盘数据经过压缩与当前棋盘状态经操作可获得的最少棋子数一起构成一个键值对，存放进入哈希表中。

棋盘相关数据关联图如图3-1所示。

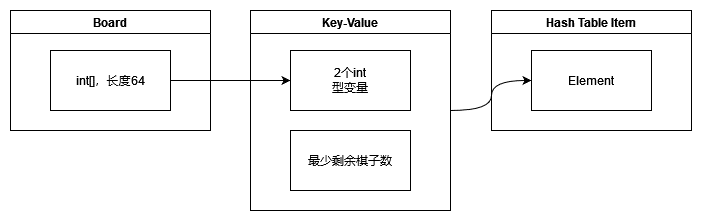


图3-1 棋盘相关数据关联图

每次查找当前棋局合法行动的时候，将所有行动存放进一个栈中，在搜索算法内，逐个取出栈中元素。

行动相关数据关联图如图3-2所示。

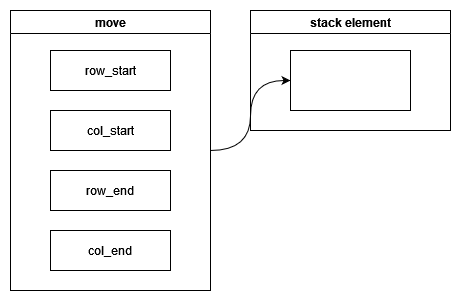


图3-2 行动相关数据关联图

## 3.2 主要算法设计

在peg1.c0中，完成对每次行动move数据结构的定义，并实现从move中获取这一步起始行列、终止行列的函数。实现一个帮助函数stack all\_possible\_moves(board B)，获取当前棋局所有的合法行动，存入一个栈中。实现一个求解函数int solve(board B, stack S, int num\_pegs)，可以每次执行栈顶一步，忽略其他步，直到无法继续执行，并且将每次行动存入栈S中。

peg1求解流程图如图3-3所示。

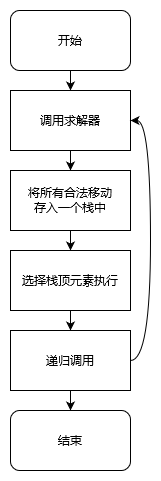


图3-3 peg1求解流程图

在peg2.c0中，在peg1.c0的基础上继续完成，实现回溯方法，如果找到解，那么在回溯过程中将每次行动存入栈S中。

peg2求解流程图如图3-4所示。

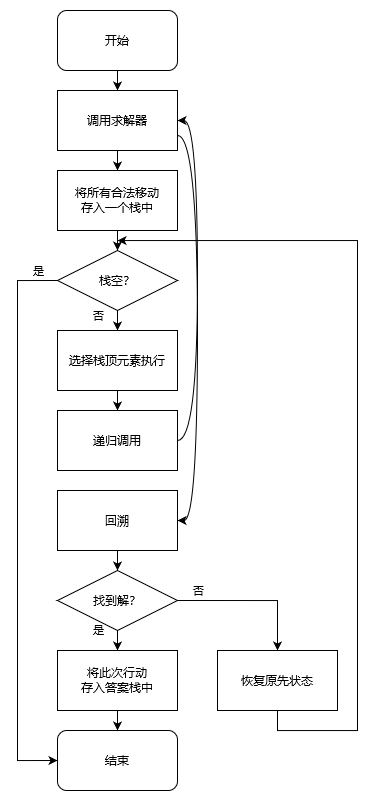


图3-4 peg2求解流程图

在peg3.c0中，在peg2.c0的基础上继续完成，使用hash table实现对相同局面的记录，进行记忆化剪枝，完成地图的存放方式和hash函数的设计。

peg3求解流程图如图3-5所示。

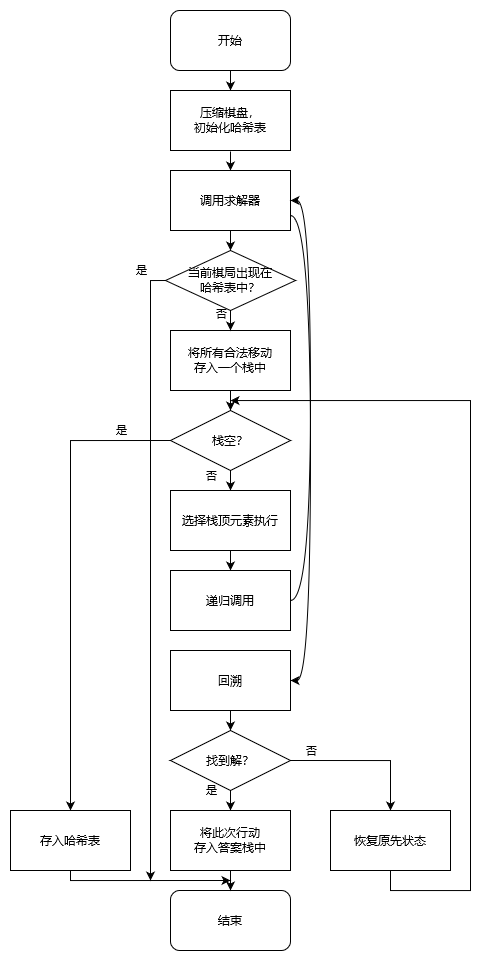


图3-5 peg3求解流程图

# 4　系统实现与测试

## 4.1系统实现

#### 4.1.1 软硬件环境

* **硬件环境：**

如表4-1所示。

表 4-1 硬件环境

|  |  |
| --- | --- |
| 处理器 | 13th Gen Intel(R) Core(TM) i9-13900HX 2.20 GHz |
| 机带 RAM | 16.0 GB (15.7 GB可用) |
| 系统类型 | 1. 位操作系统, 基于x64的处理器 |

* **软件环境：**

Windows 11 家庭中文版，版本 23H2上的WSL 2，Linux发行版本为Ubuntu 22.04.3 LTS 版本。

C0编译器和解释器版本均为20220712。

#### 4.1.2 数据类型实现

棋盘为大小，在C0中无法定义一个二维数组，因此用长度为64的一维数组实现，下标均从开始，访问第行第列元素时，即访问此一维数组中的项。输入文件有8行8列，每格有3种情况，分别为’.’ ’o’ ’\*’，分别代表被挡住的格子、孔、棋子，分别转化为 -1、0、1。

typedef int[] board;

行动数据包括此次行动起始的行列和终止的行列，共4个小于8的数，可以将这4个数压缩成一个int传输，第位为起始的行数，第位为起始的列数，第位为结束的行数，第位为结束的列数。

typedef int move;

栈内存放元素类型与行动定义的数据结构类型一致。

typedef int stackelem;

压缩后的棋盘数据用two\_int结构体存储，棋盘共有位，在压缩时，忽略-1，当成0处理，并保存当前棋局最后能剩下的最少棋子数。

struct two\_ints {

int i1;

int i2;

int best\_num\_pegs;

};

哈希表中元素数据类型与压缩后的棋盘数据类型一致。

typedef struct two\_ints\* htelem;

typedef struct two\_ints\* htkey;

#### 4.1.3 函数说明

* 1. 库端函数接口

1. Stack

bool stack\_empty(stack S); /\* O(1) \*/

判断栈S是否为空，时间复杂度O(1)。

stack stack\_new(); /\* O(1) \*/

新建一个空栈，时间复杂度O(1)。

void push(stack S, stackelem e); /\* O(1) \*/

将一个元素e压入栈S中，时间复杂度O(1)。

stackelem pop(stack S) /\* O(1) \*/

//@requires !stack\_empty(S);

;

将栈顶元素从栈S中弹出并返回，时间复杂度O(1)。

int stack\_size(stack S); /\* O(1) \*/

获取栈S中元素个数，时间复杂度O(1)。

1. Hash Table

ht ht\_new(int capacity)

//@requires capacity > 0;

;

新建一个尺寸为capacity的哈希表。

htelem ht\_lookup(ht H, htkey k); /\* O(1) avg. \*/

通过键值k，在哈希表H中查找对应的元素，均摊时间复杂度O(1)。

void ht\_insert(ht H, htelem e) /\* O(1) avg. \*/

//@requires e != NULL;

;

向哈希表H中插入元素e，均摊时间复杂度O(1)。

int ht\_size(ht H); /\* O(1) \*/

获取哈希表H中元素个数，时间复杂度O(1)。

void ht\_stats(ht H);

输出哈希表调试信息。

1. Board

int num\_pegs(board B);

返回当前棋盘上剩余几个棋子。

void init\_board(int[] B);

初始化为默认棋盘（English棋盘）。

void print\_board(board B);

打印棋盘。

void read\_board(string filename, int[] B, int n);

从文件中读入棋盘。

void array\_copy(int[] dst, int[] src, int n);

从数组src中复制n个元素到数组dst。

* 1. peg-client 中函数说明

int hash(htkey k);

将键值转化为哈希值。

bool htkey\_equal(htkey k1, htkey k2);

判断两键值相等。

htkey htelem\_key(htelem e)

//@requires e != NULL;

;

从元素中获取键值。

* 1. peg1、peg2中函数说明

int row\_start(move m);

获取此次行动的起始行。

int col\_start(move m);

获取此次行动的起始列。

int row\_end(move m);

获取此次行动的终止行。

int col\_end(move m);

获取此次行动的终止列。

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;

;

从棋局B开始解孔明棋，返回最少剩余棋子数，若为1，在栈S中存放行动序列。

int solve(board B, stack S, int num\_pegs);

参数棋局B，剩余棋子数num\_pegs，答案栈S，返回棋局B能达到的最少剩余棋子数。

stack all\_possible\_moves(board B);

在棋局B上有多少合法行动，存入栈中返回。

* 1. peg3中与peg2不同的函数说明

stack all\_possible\_moves(board B, int[] dr, int[] dc);

多传入了方向数组dr、dc。

int solve(board B, stack S, int num\_pegs, int i1, int i2, ht H, int[] dr, int[] dc)

多传入了压缩后棋盘i1、i2，哈希表H，方向数组dr、dc。

* 1. peg-main中函数说明

int main();

程序入口。负责检查运行参数等工作。

int peg\_main(board B\_init);

主程序。调用求解器，调用检验器。

bool verify\_solution(board B, stack S);

检查解是否正确。

bool legal\_move(board B, int row1, int col1, int row2, int col2);

判断此次行动是否合法。

void apply\_move(board B, int row1, int col1, int row2, int col2);

执行行动。

bool is\_coord(int x);

判断坐标是否合法。

void print\_coords(int row, int col);

打印坐标。

peg-main函数调用关系图如图4-1所示。

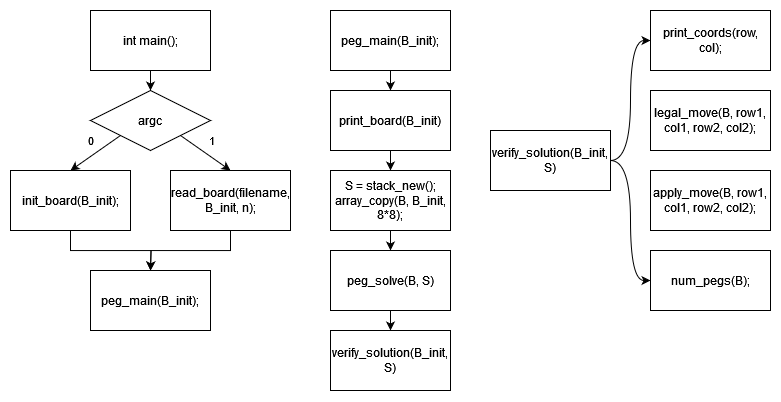


图4-1 peg-main函数调用关系图

peg1和peg2函数调用关系图如图4-2所示。

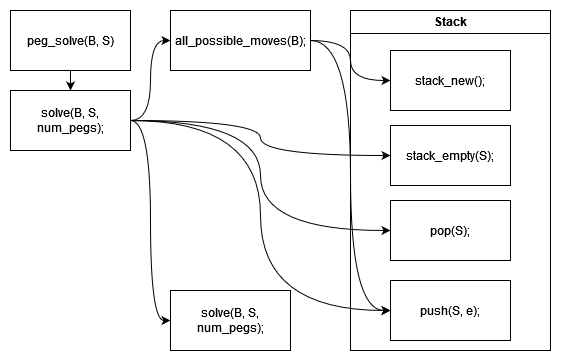


图4-2 peg1和peg2函数调用关系图

peg3函数调用关系图如图4-3所示。

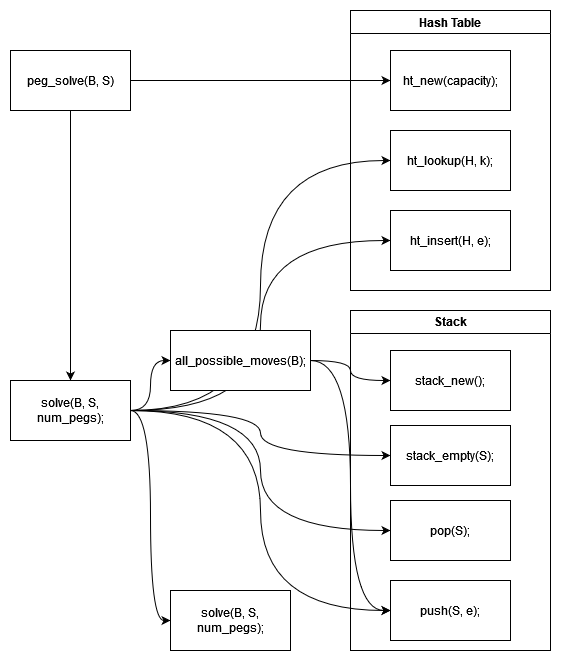


图4-3 peg3函数调用关系图

## 4.2系统测试

#### 4.2.1 基础功能测试

模块功能：孔明棋求解器。

测试目的：检查是否已经达成基础功能目标，可以正确地解出孔明棋，给出正确的答案。

详细说明：peg1.c0仅要求能够解出每步有唯一后继棋局的棋局。peg2.c0和peg3.c0则要求能够解出一般的棋盘。

测试数据：german.txt、english.txt。

测试方法：由于是功能验证，要求打开动态检查。检查运行结果是否符合要求，运行过程中没有发生错误。

测试命令：

% cc0 -d -w -o peg1 peg-client.c0 lib/\*.c0 peg1.c0 peg-main.c0

% ./peg1 german.txt

% cc0 -d -w -o peg2 peg-client.c0 lib/\*.c0 peg2.c0 peg-main.c0

% ./peg2 german.txt

% ./peg2 english.txt

% cc0 -d -w -o peg3 peg-client.c0 lib/\*.c0 peg3.c0 peg-main.c0

% ./peg3 german.txt

% ./peg3 english.txt

测试结果：

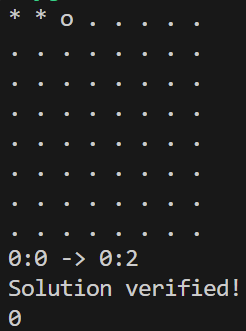


图4-4 peg1功能测试图

peg1、peg2、peg3测试功能图分别如图4-4、图4-5、图4-6所示。

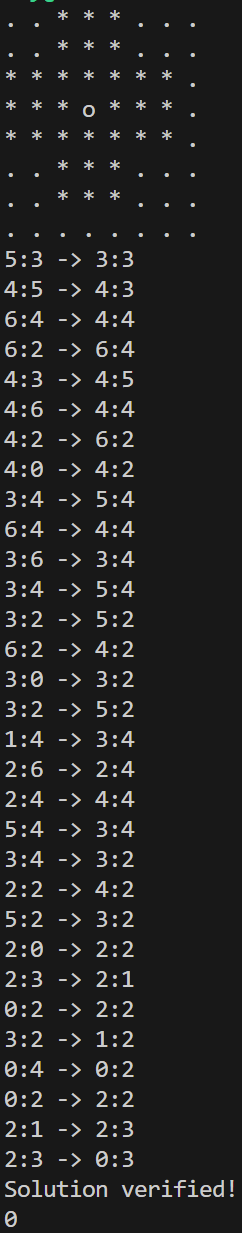
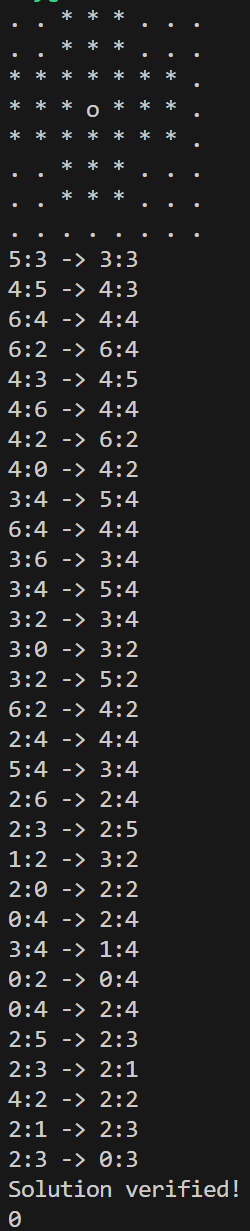
 

图4-5 peg2功能测试图 图4-6 peg3功能测试图

测试结论：所有部分功能均能正常执行，无异常提示。

#### 4.2.2 记忆化性能优化测试

测试目的：比较记忆化与否对性能差异的影响。

详细说明：由于记忆化与否是peg2.c0与peg3.c0的关键不同，因此选用peg2.c0和peg3.c0在English标准棋盘中的表现来进行测试。

测试数据：english.txt。

测试方法：由于是性能测试，要求关闭动态检查，开启O2优化，通过比对运行时间来得出结论。

测试命令：

% cc0 -w -o peg2 -r unsafe -c-O2 peg-client.c0 lib/\*.c0 peg2.c0 peg-main.c0

% time ./peg2 english.txt

% cc0 -w -o peg3 -r unsafe -c-O2 peg-client.c0 lib/\*.c0 peg3.c0 peg-main.c0

% time ./peg3 english.txt

测试结果：如表4-1所示。

表4-1 记忆化性能测试结果图

|  |  |  |
| --- | --- | --- |
| 耗时 | peg2 | peg3 |
| english.txt | 0m0.846s | 0m0.056s |

测试结论：采用记忆化的peg3效率显著高于peg2。

#### 4.2.3 方向选择优化测试

测试目的：测试不同方向的枚举顺序不同对最终性能的影响。

详细说明：由于每次执行行动时，都是从栈顶取出元素，而栈来源于all\_posible\_moves(board B, int[] dr, int[] dc)函数。函数内先枚举整个棋盘，这里有3种等价的方法：1. 每次找到一个棋子，看它能否向各方向跳；2. 每次找到一个棋子，看它能否被从各方向越过；3. 每次找到一个孔，看它是否能被来自各方向的棋子跳入。这里采取了第一种方案，那么需要测试不同方向顺序对入栈顺序的影响，进而对遍历到的无解局面数的影响。

测试数据：english.txt、french1.txt、french2.txt、french3.txt。

测试方法：开启O2优化选项，枚举各个方向顺序，以哈希表中存放的无解状态数作为评判依据。

测试命令：

% cc0 -w -o peg3test -r unsafe -c-O2 peg-client.c0 lib/\*.c0 peg3test.c0 peg-main.c0

% ./peg3test english.txt

% ./peg3test french1.txt

% ./peg3test french2.txt

% ./peg3test french3.txt

测试结果：如表4-2所示。

表4-2 方向选择测试结果表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 方向顺序 | english.txt | french1.txt | french2.txt | french3.txt |
| 上下左右 | 859 | 22730603 | 10991786 | 12022594 |
| 上下右左 | 859 | 22730603 | 10991786 | 12022594 |
| 上左下右 | 859 | 22730603 | 10991786 | 12022594 |
| 上左右下 | 9752 | 22730603 | 10991786 | 12022812 |
| 上右下左 | 9752 | 22730603 | 10991786 | 12022812 |
| 上右左下 | 9752 | 22730603 | 10991786 | 12022812 |
| 下上左右 | 859 | 22730603 | 10991786 | 13362377 |
| 下上右左 | 859 | 22730603 | 10991786 | 13362377 |
| 下左上右 | 859 | 22730603 | 10991786 | 13362377 |
| 下左右上 | 859 | 22730603 | 10991786 | 13362377 |
| 下右上左 | 859 | 22730603 | 10991786 | 13362377 |
| 下右左上 | 859 | 22730603 | 10991786 | 13362377 |
| 左上下右 | 859 | 22730603 | 10991786 | 12022594 |
| 左上右下 | 9752 | 22730603 | 10991786 | 12022812 |
| 左下上右 | 859 | 22730603 | 10991786 | 13362377 |
| 左下右上 | 859 | 22730603 | 10991786 | 13362377 |
| 左右上下 | 9752 | 22730603 | 10991786 | 12022812 |
| 左右下上 | 9752 | 22730603 | 10991786 | 13362465 |
| 右上下左 | 9752 | 22730603 | 10991786 | 12022812 |
| 右上左下 | 9752 | 22730603 | 10991786 | 12022812 |
| 右下上左 | 9752 | 22730603 | 10991786 | 13362465 |
| 右下左上 | 9752 | 22730603 | 10991786 | 13362465 |
| 右左上下 | 9752 | 22730603 | 10991786 | 12022812 |
| 右左下上 | 9752 | 22730603 | 10991786 | 13362465 |

测试结论：方向顺序对某些地图来说没有影响，选择“上下左右”方向顺序可以在这4个测试数据上均取得最好效果。

#### 4.2.4 哈希函数选择优化测试

测试目的：测试不同的哈希函数对性能影响。

详细说明：棋盘已经被压缩成2个整型，现在需要写一个哈希函数将两个整型转化为1个整型返回值。我选取了题目中给的将两数直接相加、Java标准库中写的将两数相异或、以及将前数乘一个系数与后数相加，这里系数我选取了1个常用的系数，13331，一共3个哈希函数进行测试。

测试方法：使用题目中的测试代码进行测试，比较最终哈希表的状态。

测试命令：

$ cc0 peg-client.c0 lib/\*.c0 peg3.c0 pegmark.c0

$ time ./a.out

测试结果：两数相加哈希函数测试图如图4-7所示。

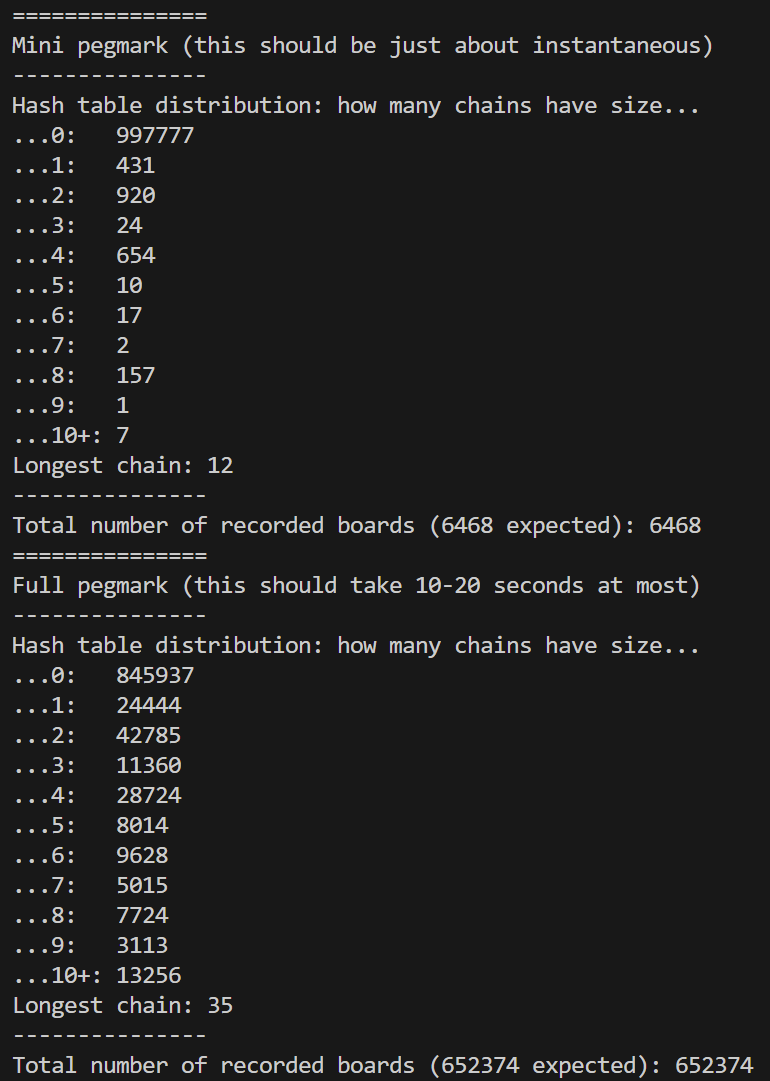


图4-7两数相加哈希函数测试图

两数相异或哈希函数测试图如图4-8所示。

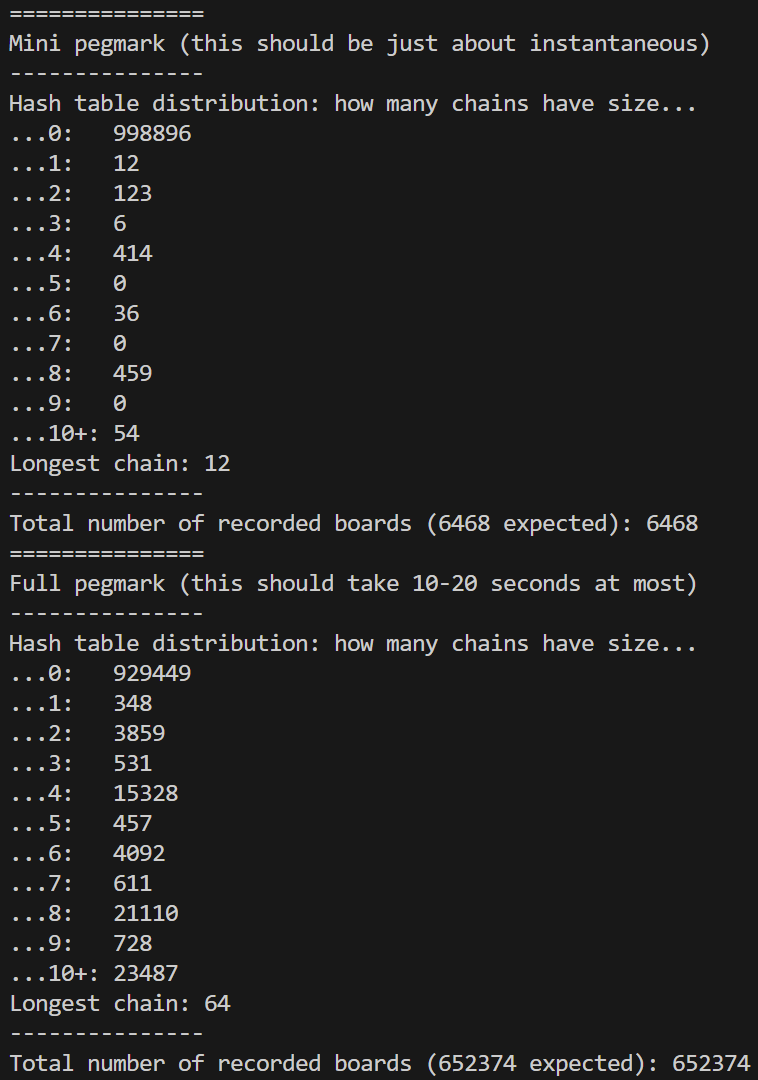


图4-8两数相异或哈希函数测试图

乘系数相加哈希函数测试图如图4-9所示。

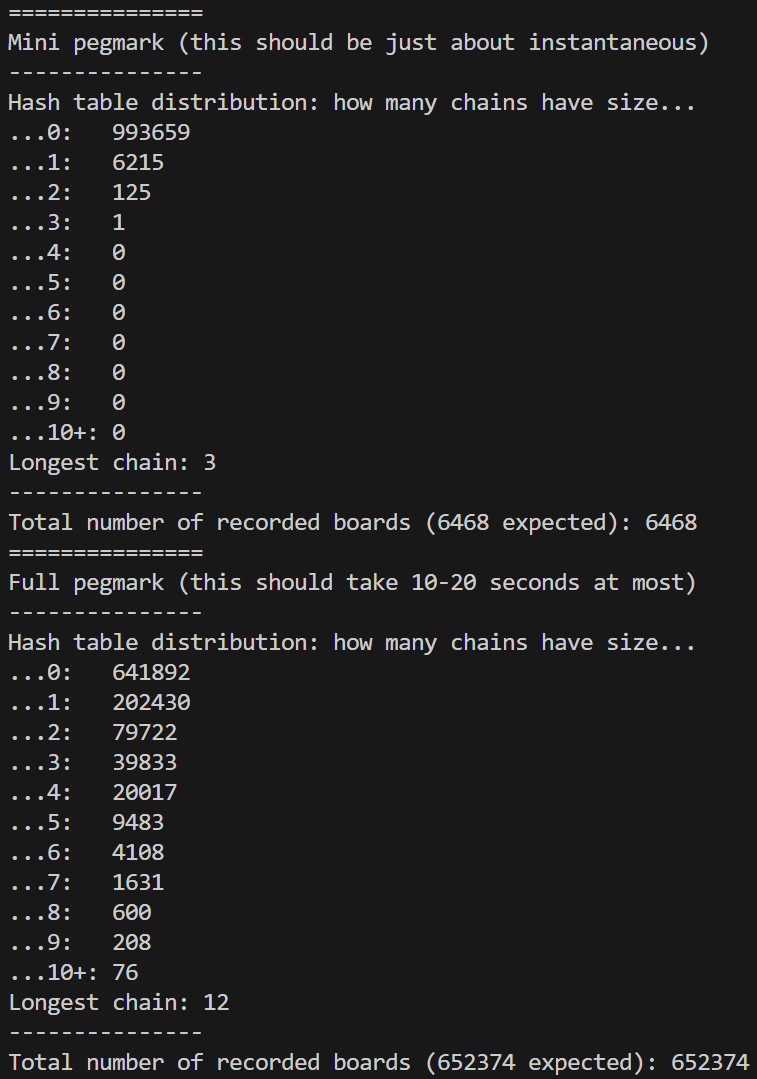


图4-9乘系数相加哈希函数测试图

测试结论：采用i1 \* 13331 + i2哈希函数是三者中最优，最长链最短。而标准库的写法优点在于避免了整型溢出，但是在这个情况下反而表现最差，发生碰撞的概率很高。

#### 其他优化

在以上提及的主要优化之外，我还实现了一些可以减小常数的优化，尽管开启了O2优化选项，但是还有一些部分是编译器无法帮我们自动完成的。

1. **位运算优化**

在求中点时，我们原本需要写l + (r - l) / 2来防止溢出的发生，但是在孔明棋这种环境下完全不用担心溢出发生，而且由于我们知道l + r一定是正数，因此也不用加符号位，可以直接写出(l + r) >> 1。

在进行行动的压缩时，选择一个适当的压缩方式也是值得考虑的。如果按照10进制来存放，尽管在调试过程中可能十分简单，一目了然，但是在压缩和解压时要用到乘除法，没有按照字节使用位运算的方法效率高。我们可以写出这样的式子来构建一个行动：move M = row1 | (col1 << 8) | (row3 << 16) | (col3 << 24); ，在获取各成员时，使用右移和按位与运算即可，运用位运算可以提高程序效率。

由于C0语言没有多维数组的概念，我们只能拿一维数组去模拟二维数组。在本孔明棋求解问题中，地图大小是的，因此我们可以使用row \* 8 + col来将一个行列转化为地图中的下标。要注意这里如果使用了O2优化，编译器会使用地址传输汇编指令优化，如图4-10所示，因此这里没必要用位运算优化成(row << 3) | col。但如果没有使用优化指令，位运算效率更高。

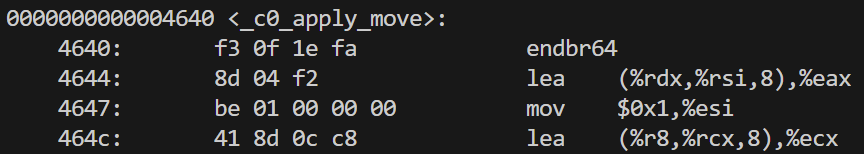


图4-10 O2优化后部分反汇编指令图

1. **减少动态空间分配次数**

由于C0只有分配内存的方法，并没有给我们提供主动释放内存的方法。全都依赖C0语言自己管理的垃圾回收机制。如果触发了垃圾回收，那么程序的运行效率就会突然变得很低了。就算没有触发垃圾回收，在堆区分配空间、数据搬运等也会造成时间上的开销。所以我们可以在申请空间时尽量开大一点容量，或者在尝试过程中慢慢调大，例如在上文提及的测试中，就把哈希表的容量初始化为30000000。同时，连续分配内存也有助于提高cache的命中率，加快速度。

除开哈希表，方向数组也可以避免总是每次重新分配内存。C0语言没有全局变量，因此需要在调用求解函数的时候把方向数组同时传入，这样也能减少内存分配，减少堆区小块数据。

其实还可以直接去掉all\_possible\_moves这个辅助函数，不用存入栈中，每找到一个合法行动就尝试执行，这样也能减少内存分配，但是这样不利于模块化编程，最终没有按照这样实现。

1. **减少多余计算次数**

我们不用每次都重复计算压缩棋盘，只需要在每次修改的时候同步将棋盘和压缩数据一起更新。由于压缩数据也是基于位运算计算出来的，所以更新起来也非常快。

验证行动是否合法时，只检验终点，不用检验中间点是否超出地图。

最终性能测试命令：

% cc0 -w -o peg3 -r unsafe -c-O2 peg-client.c0 lib/\*.c0 peg3.c0 peg-main.c0

% time ./peg3 french1.txt

% time ./peg3 french2.txt

% time ./peg3 french3.txt

测试结果：如表4-3所示。

表4-3 最终性能测试结果图

|  |  |  |  |
| --- | --- | --- | --- |
| 耗时 | french1.txt | french2.txt | french3.txt |
| peg3 | 0m35.792s | 0m12.631s | 0m13.177s |

测试结论：本程序很好地达到了设计时的预期效果。可以稳定在40秒内解出较为复杂的棋盘。

# 5　总结

主要工作如下：

（1）完成了peg-client.c0、peg1.c0、peg2.c0、peg3.c0的实现。具体地说，完成了求解算法的实现，并且设计了行动和压缩棋盘数据结构等。很好地实现了模块化的编程，各个模块有着有低耦合、高内聚的特点，在性能方面相互影响较小。模块间相互调用联系，顺利地实现了目标功能。

（2）完成了性能优化，编写了peg3test.c0，进行了多项性能优化方面的测试。尤其在测试的过程中，对多个相对独立的模块通过控制变量法分别优化，都大大提升了程序效率，在这过程中，也提升了对计算机系统的理解。

孔明棋求解程序应该还有更大的进步空间，比如可能可以给每次行动估计是否较优，或是实现双端搜索来减小搜索树的大小等。但具体实现起来，都有一定的复杂度。如果对每次行动估值然后采用A\*算法，对于估价函数要求满足一致性等条件，设计一个估价函数可能很复杂，计算起来也会比较麻烦。实现双端搜索需要枚举结束位置，具体运行时哈希表需要的空间开销可能较大，最终由于时间和能力有限，没有实现这两种方法。

# 6　体会

此次课程设计初看比较复杂，其实是因为需求文件十分细致，将每一模块需要完成什么，为什么要这样设计都讲得十分清楚，看起来需求很长，实际上是减少了我们的负担，避免我们走弯路。

将整个需求文档了解清楚之后，就可以知道我们需要完成一个搜索，并且在这个基础上优化。文档十分详细地给出了几个优化的方向，比如减小数据结构的大小，记忆化剪枝，调整枚举方向顺序，减少哈希碰撞等方法。我按照这些优化方法逐步实现，从最开始的完全运行不出来，到需要几分钟，到最后的35秒左右，我在这个过程中的喜悦难以言表。

在此次课程设计与实现的过程中，我充分运用了所学知识，不断精益求精，从中收获了许多。

最后感谢李开老师与各位同学在命令式计算原理学习过程中给与我的无私帮助！

# 参考文献

1. Frank Pfenning. C0 Reference. 2021

<https://c0.cs.cmu.edu/docs/c0-reference.pdf>

1. Frank Pfenning. C0 Libraries. 2022

<https://c0.cs.cmu.edu/docs/c0-libraries.pdf>

1. 骆可强.论程序底层优化的一些方法与技巧[C].2009年全国信息学奥林匹克冬令营论文.2009

<https://github.com/bojieli/NOI-papers/blob/master/%E5%9B%BD%E5%AE%B6%E9%9B%86%E8%AE%AD%E9%98%9F2009%E8%AE%BA%E6%96%87%E9%9B%86/4.%E9%AA%86%E5%8F%AF%E5%BC%BA%E3%80%8A%E8%AE%BA%E7%A8%8B%E5%BA%8F%E5%BA%95%E5%B1%82%E4%BC%98%E5%8C%96%E7%9A%84%E4%B8%80%E4%BA%9B%E6%96%B9%E6%B3%95%E4%B8%8E%E6%8A%80%E5%B7%A7%E3%80%8B/%E8%AE%BA%E7%A8%8B%E5%BA%8F%E5%BA%95%E5%B1%82%E4%BC%98%E5%8C%96%E7%9A%84%E4%B8%80%E4%BA%9B%E6%96%B9%E6%B3%95%E4%B8%8E%E6%8A%80%E5%B7%A7.pdf>

# 附录

**peg-client.c0**

/\* Client interface for stacks. \*/

/\* See lib/stacks.c0 \*/

typedef int stackelem;

/\* Client side implementation for hashtables. (Optional) \*/

/\* See lib/ht.c0 \*/

struct two\_ints {

int i1;

int i2;

int best\_num\_pegs;

};

typedef struct two\_ints\* htelem;

typedef struct two\_ints\* htkey;

int hash(htkey k)

//@requires k != NULL;

{

return k->i1 \* 13331 + k->i2;

}

bool htkey\_equal(htkey k1, htkey k2)

//@requires k1 != NULL;

//@requires k2 != NULL;

{

return k1->i1 == k2->i1 && k1->i2 == k2->i2;

}

htkey htelem\_key(htelem e)

//@requires e != NULL;

{

return e;

}

**peg1.c0**

#use <args>

#use <conio>

#use <util>

typedef int move;

int row\_start(move m);

int col\_start(move m);

int row\_end(move m);

int col\_end(move m);

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 row\_start(move m) {

return m&0xFF;

}

int col\_start(move m) {

return (m>>8)&0xFF;

}

int row\_end(move m) {

return (m>>16)&0xFF;

}

int col\_end(move m) {

return (m>>24)&0xFF;

}

stack all\_possible\_moves(board B)

//@requires is\_board(B);

{

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

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

dr[0] = 0; dr[1] = 0; dr[2] = 1; dr[3] = -1;

dc[0] = 1; dc[1] = -1; dc[2] = 0; dc[3] = 0;

stack S = stack\_new();

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

for (int j = 0; j < 8; j++)

if (B[i \* 8 + j] == 1) // (i, j) is a peg

{

for (int d = 0; d < 4; d++) {

int row1 = i;

int col1 = j;

int row2 = i + dr[d];

int col2 = j + dc[d];

int row3 = i + 2\*dr[d];

int col3 = j + 2\*dc[d];

if (0 <= row2 && row2 < 8 && 0 <= col2 && col2 < 8 &&

0 <= row3 && row3 < 8 && 0 <= col3 && col3 < 8 &&

B[row2\*8+col2] == 1 && B[row3\*8+col3] == 0) {

move M = row1 | (col1 << 8) | (row3 << 16) | (col3 << 24);

push(S, M);

}

}

}

return S;

}

int solve(board B, stack S, int num\_pegs)

{

stack moves = all\_possible\_moves(B);

if (stack\_empty(moves)) {

return num\_pegs;

}

move M = pop(moves);

int row1 = row\_start(M);

int col1 = col\_start(M);

int row3 = row\_end(M);

int col3 = col\_end(M);

int row2 = (row1 + row3) >> 1;

int col2 = (col1 + col3) >> 1;

B[row1\*8+col1] = 0;

B[row2\*8+col2] = 0;

B[row3\*8+col3] = 1;

int result = solve(B, S, num\_pegs - 1);

push(S, M);

return result;

}

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;

{

return solve(B, S, num\_pegs(B));

}

**peg2.c0**

#use <args>

#use <conio>

#use <util>

typedef int move;

int row\_start(move m);

int col\_start(move m);

int row\_end(move m);

int col\_end(move m);

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 row\_start(move m) {

return m&0xFF;

}

int col\_start(move m) {

return (m>>8)&0xFF;

}

int row\_end(move m) {

return (m>>16)&0xFF;

}

int col\_end(move m) {

return (m>>24)&0xFF;

}

stack all\_possible\_moves(board B)

//@requires is\_board(B);

{

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

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

dr[0] = 0; dr[1] = 0; dr[2] = 1; dr[3] = -1;

dc[0] = 1; dc[1] = -1; dc[2] = 0; dc[3] = 0;

stack S = stack\_new();

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

for (int j = 0; j < 8; j++)

if (B[i \* 8 + j] == 1) // (i, j) is a peg

{

for (int d = 0; d < 4; d++) {

int row1 = i;

int col1 = j;

int row2 = i + dr[d];

int col2 = j + dc[d];

int row3 = i + 2\*dr[d];

int col3 = j + 2\*dc[d];

if (0 <= row2 && row2 < 8 && 0 <= col2 && col2 < 8 &&

0 <= row3 && row3 < 8 && 0 <= col3 && col3 < 8 &&

B[row2\*8+col2] == 1 && B[row3\*8+col3] == 0) {

move M = row1 | (col1 << 8) | (row3 << 16) | (col3 << 24);

push(S, M);

}

}

}

return S;

}

int solve(board B, stack S, int num\_pegs)

{

stack moves = all\_possible\_moves(B);

if (stack\_empty(moves)) {

return num\_pegs;

}

int best\_result = num\_pegs;

while (!stack\_empty(moves)) {

move M = pop(moves);

int row1 = row\_start(M);

int col1 = col\_start(M);

int row3 = row\_end(M);

int col3 = col\_end(M);

int row2 = (row1 + row3) >> 1;

int col2 = (col1 + col3) >> 1;

B[row1\*8+col1] = 0;

B[row2\*8+col2] = 0;

B[row3\*8+col3] = 1;

int result = solve(B, S, num\_pegs - 1);

best\_result = min(best\_result, result);

if (result == 1) {

push(S, M);

return 1;

}

B[row1\*8+col1] = 1;

B[row2\*8+col2] = 1;

B[row3\*8+col3] = 0;

}

return best\_result;

}

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;

{

return solve(B, S, num\_pegs(B));

}

**peg3.c0**

#use <args>

#use <conio>

#use <util>

typedef int move;

int row\_start(move m);

int col\_start(move m);

int row\_end(move m);

int col\_end(move m);

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 row\_start(move m) {

return m&0xFF;

}

int col\_start(move m) {

return (m>>8)&0xFF;

}

int row\_end(move m) {

return (m>>16)&0xFF;

}

int col\_end(move m) {

return (m>>24)&0xFF;

}

stack all\_possible\_moves(board B, int[] dr, int[] dc)

//@requires is\_board(B);

{

stack S = stack\_new();

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

for (int j = 0; j < 8; j++)

if (B[i \* 8 + j] == 1) // (i, j) is a peg

{

for (int d = 0; d < 4; d++) {

int row1 = i;

int col1 = j;

int row2 = i + dr[d];

int col2 = j + dc[d];

int row3 = i + 2\*dr[d];

int col3 = j + 2\*dc[d];

if (0 <= row3 && row3 < 8 && 0 <= col3 && col3 < 8 &&

B[row2 \* 8 + col2] == 1 && B[row3 \* 8 + col3] == 0) {

move M = row1 | (col1 << 8) | (row3 << 16) | (col3 << 24);

push(S, M);

}

}

}

return S;

}

int solve(board B, stack S, int num\_pegs, int i1, int i2, ht H, int[] dr, int[] dc)

{

struct two\_ints\* current = alloc(struct two\_ints);

current->i1 = i1;

current->i2 = i2;

current->best\_num\_pegs = num\_pegs;

struct two\_ints\* find = ht\_lookup(H, current);

if (find != NULL) {

return find->best\_num\_pegs;

}

stack moves = all\_possible\_moves(B, dr, dc);

if (stack\_empty(moves))

return num\_pegs;

while (!stack\_empty(moves)) {

move M = pop(moves);

int row1 = row\_start(M);

int col1 = col\_start(M);

int row3 = row\_end(M);

int col3 = col\_end(M);

int row2 = (row1 + row3) >> 1;

int col2 = (col1 + col3) >> 1;

int index1 = row1 \* 8 + col1;

int index2 = row2 \* 8 + col2;

int index3 = row3 \* 8 + col3;

int ni1 = i1;

int ni2 = i2;

B[index1] = 0;

B[index2] = 0;

B[index3] = 1;

if (index1<32)

ni1 ^= (1<<index1);

else

ni2 ^= (1<<(index1-32));

if (index2<32)

ni1 ^= (1<<index2);

else

ni2 ^= (1<<(index2-32));

if (index3<32)

ni1 ^= (1<<index3);

else

ni2 ^= (1<<(index3-32));

int result = solve(B, S, num\_pegs - 1, ni1, ni2, H, dr, dc);

current->best\_num\_pegs = min(current->best\_num\_pegs, result);

B[index1] = 1;

B[index2] = 1;

B[index3] = 0;

if (result == 1) {

push(S, M);

return 1;

}

}

ht\_insert(H, current);

return current->best\_num\_pegs;

}

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

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

dr[0] = -1; dr[1] = 1; dr[2] = 0; dr[3] = 0;

dc[0] = 0; dc[1] = 0; dc[2] = -1; dc[3] = 1;

int i1 = 0;

int i2 = 0;

ht H = ht\_new(30000000);

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

i1 |= (max(B[i],0) << i);

}

for (int i = 32; i < 64; i++) {

i2 |= (max(B[i],0) << (i-32));

}

int ans = solve(B, S, num\_pegs(B), i1, i2, H, dr, dc);

return ans;

}

**peg3test.c0**

#use <args>

#use <conio>

#use <util>

typedef int move;

int row\_start(move m);

int col\_start(move m);

int row\_end(move m);

int col\_end(move m);

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 row\_start(move m) {

return m&0xFF;

}

int col\_start(move m) {

return (m>>8)&0xFF;

}

int row\_end(move m) {

return (m>>16)&0xFF;

}

int col\_end(move m) {

return (m>>24)&0xFF;

}

stack all\_possible\_moves(board B, int[] dr, int[] dc)

//@requires is\_board(B);

{

stack S = stack\_new();

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

for (int j = 0; j < 8; j++)

if (B[i \* 8 + j] == 1) // (i, j) is a peg

for (int d = 0; d < 4; d++) {

int row1 = i;

int col1 = j;

int row2 = i + dr[d];

int col2 = j + dc[d];

int row3 = i + 2\*dr[d];

int col3 = j + 2\*dc[d];

if (0 <= row3 && row3 < 8 && 0 <= col3 && col3 < 8 &&

B[(row2<<3)|col2] == 1 && B[(row3<<3)|col3] == 0) {

move M = row1 | (col1 << 8) | (row3 << 16) | (col3 << 24);

push(S, M);

}

}

return S;

}

int solve(board B, stack S, int num\_pegs, int i1, int i2, ht H, int[] dr, int[] dc)

{

struct two\_ints\* current = alloc(struct two\_ints);

current->i1 = i1;

current->i2 = i2;

current->best\_num\_pegs = num\_pegs;

struct two\_ints\* find = ht\_lookup(H, current);

if (find != NULL) {

return find->best\_num\_pegs;

}

stack moves = all\_possible\_moves(B, dr, dc);

if (stack\_empty(moves)) {

if (num\_pegs == 1) {

printf("size = %d\n", ht\_size(H));

}

return num\_pegs;

}

while (!stack\_empty(moves)) {

move M = pop(moves);

int row1 = row\_start(M);

int col1 = col\_start(M);

int row3 = row\_end(M);

int col3 = col\_end(M);

int row2 = (row1 + row3) >> 1;

int col2 = (col1 + col3) >> 1;

int index1 = (row1<<3)|col1;

int index2 = (row2<<3)|col2;

int index3 = (row3<<3)|col3;

int ni1 = i1;

int ni2 = i2;

B[index1] = 0;

B[index2] = 0;

B[index3] = 1;

if (index1<32)

ni1 ^= (1<<index1);

else

ni2 ^= (1<<(index1-32));

if (index2<32)

ni1 ^= (1<<index2);

else

ni2 ^= (1<<(index2-32));

if (index3<32)

ni1 ^= (1<<index3);

else

ni2 ^= (1<<(index3-32));

int result = solve(B, S, num\_pegs - 1, ni1, ni2, H, dr, dc);

current->best\_num\_pegs = min(current->best\_num\_pegs, result);

B[index1] = 1;

B[index2] = 1;

B[index3] = 0;

if (result == 1) {

push(S, M);

return 1;

}

}

ht\_insert(H, current);

return current->best\_num\_pegs;

}

bool next\_permutation(int[] a)

{

int n = 4;

int i = n - 2;

while (i >= 0 && a[i] >= a[i+1]) {

i--;

}

if (i < 0) {

return false;

}

int j = n - 1;

while (a[j] <= a[i]) {

j--;

}

int temp = a[i];

a[i] = a[j];

a[j] = temp;

int l = i + 1;

int r = n - 1;

while (l < r) {

temp = a[l];

a[l] = a[r];

a[r] = temp;

l++;

r--;

}

return true;

}

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

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

DR[0] = -1; DR[1] = 1; DR[2] = 0; DR[3] = 0;

DC[0] = 0; DC[1] = 0; DC[2] = -1; DC[3] = 1;

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

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

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

a[0] = 0; a[1] = 1; a[2] = 2; a[3] = 3;

int i1 = 0;

int i2 = 0;

ht H = ht\_new(40000000);

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

i1 |= (max(B[i],0) << i);

}

for (int i = 32; i < 64; i++) {

i2 |= (max(B[i],0) << (i-32));

}

for (int i = 0; i < 4; i++) {

dr[i] = DR[a[i]];

dc[i] = DC[a[i]];

}

printf("%d %d %d %d: %d\n", a[0], a[1], a[2], a[3], solve(B, S, num\_pegs(B), i1, i2, H, dr, dc));

while(!stack\_empty(S))

pop(S);

while(next\_permutation(a)){

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

i1 |= (max(B[i],0) << i);

}

for (int i = 32; i < 64; i++) {

i2 |= (max(B[i],0) << (i-32));

}

H = ht\_new(40000000);

for (int i = 0; i < 4; i++) {

dr[i] = DR[a[i]];

dc[i] = DC[a[i]];

}

printf("%d %d %d %d: %d\n", a[0], a[1], a[2], a[3], solve(B, S, num\_pegs(B), i1, i2, H, dr, dc));

while(!stack\_empty(S))

pop(S);

}

return solve(B, S, num\_pegs(B), i1, i2, H, dr, dc);

}