Artificial Intelligence Lab1 Report

王嵘晟　PB1711614

1　数码问题：

(1). 启发式函数

启发式函数使用曼哈顿距离，即根据输入的初始状态和最终状态，计算每个数字从初始状态到最终状态的曼哈顿距离作为 h，对于”7” 型数字，将左上块位置作为定位点。对于每个数字的曼哈顿距离，额外赋一个权值 gWeight，用来表示曼哈顿距离带来的实际代价。在这里我把每个数字的权值都赋值为 10，将各个数字的权值乘以曼哈顿距离并求和，作为这个状态的总 h 值。

(2). 采用策略

采用搜索策略，将”7” 也当做普通的数字快，根据位置做判断来移动，每次移动后得到一个新的状态，赋以一个 Hash 值作为对这个状态的唯一编号，来保证不会重复出现某状态，导致最终死循环。

(3). 伪代码概括算法

A* Search:

这里使用文字描述比较方便，首先将输入作为初始状态 initNode，然后计算它的 f, g, h, 以及 Hash 值 stateKey 用来做唯一标识。这里调用了 C++ 封装好的容器 unordered_set 以及 priority_queue，前者按照 Hash 表存储数据，所以调用插入、查找、删除操作都可以在 O(1) 的时间内完成。后者是优先级队列，将各种状态的 f 值大小作为优先级插入队列中，则队列中队头元素的 f 值为最小值。在搜索中，当队列不为空时，每次取出队头元素可以作为可能的下一状态，先判断 7 能否移动，如果可以则先移动 7，生成新的 node，即调用 MakeNode 函数，生成下一状态。若得到了最终状态，即判断方法为 h=0 则结束算法，否则继续看 7 能

否移动，7 不能移动了开始选择可以移动的其他数字。同时这里把每个新生成的状态的 Hash 值存入 unordered_set，以防止某个状态重复出现发生循环。用这种状态可以转移则转移，转移失败则回溯的方法，最终可以求解问题，输出移动方式。

IDA* Search:

大体逻辑与 A* 相同，区别在于每次取出一个 node 时看它的 f 值，并找是否有其他 node 的 f 值与该结点相同，有则对此迭代，没有则将下一个 f 最小的 node 作为 nextNode。

(4). 复杂度

A* Search

由于对于状态的查找操作都是在 O(1)，而优先队列的维护代价为 O(lgn)，n 为出现的状态空间数，所以使用 A* 搜索的总的时间复杂度为 $O(b^{\epsilon(d+1)})$，b 为层数，d 为深度。空间复杂度为 $O(nb^d)$ 因为空间上只存储了足够多的个状态的数值化表示。

IDA* Search:

时间复杂度同上为 $O(b^{\epsilon(d+1)})$，由于此时只存储了上一状态的 f 值，其余的都删除了，所以空间复杂度为 $O(b^d)$

(5). 运行结果

A* Search

输入 1.txt:

```
PS D:\AI\labs\lab1\digit\src> .\digit_AStar.exe
Please enter the digit test path and name, such as ..\input\1.txt:
..\input\1.txt
Scanning the input file: ..\input\1.txt
Please enter the result state path and name, e.g. ..\input\result.txt:
..\input\result.txt
Scanning the input result file: ..\input\result.txt
initf: 240  initg: 0  inith: 240
init stateKey: p
Find solution!:

 Total steps: 24
(1, u)          (1, u)          (6, u)          (19, l)
(15, d)         (7, l)          (14, d)         (11, d)
(14, d)         (3, r)          (2, r)          (1, u)
(11, d)         (8, r)          (7, u)          (6, u)
(14, l)         (14, l)         (15, u)         (16, l)
(20, l)         (17, u)         (21, l)         (21, l)

Excuting time: 34 ms
```

输入 2.txt:

```
PS D:\AI\labs\lab1\digit\src> .\digit_AStar.exe
Please enter the digit test path and name, such as ..\input\1.txt:
..\input\2.txt
Scanning the input file: ..\input\2.txt
Please enter the result state path and name, e.g. ..\input\result.txt:
..\input\result.txt
Scanning the input result file: ..\input\result.txt
initf: 150  initg: 0  inith: 150
init stateKey: p
Find solution!:

 Total steps: 12
(14, d)         (6, d)          (15, d)         (7, l)
(8, l)          (9, l)          (10, u)         (13, u)
(18, u)         (11, u)         (16, u)         (21, l)

Excuting time: 2231 ms
```

输入 3.txt:

```
PS D:\AI\labs\lab1\digit\src> .\digit_AStar.exe
Please enter the digit test path and name, such as ..\input\1.txt:
..\input\3.txt
Scanning the input file: ..\input\3.txt
Please enter the result state path and name, e.g. ..\input\result.txt:
..\input\result.txt
Scanning the input result file: ..\input\result.txt
initf: 470  initg: 0  inith: 470
init stateKey: pm
Find solution!:

 Total steps: 57
(6, l)          (21, r)         (17, r)         (15, l)
(16, d)         (3, d)          (13, d)         (4, l)
(7, l)          (5, u)          (5, u)          (10, r)
(10, u)         (13, r)         (13, r)         (2, r)
(9, d)          (15, d)         (2, r)          (4, d)
(7, l)          (2, u)          (4, r)          (9, r)
(1, r)          (8, d)          (6, d)          (2, u)
(4, u)          (9, r)          (1, r)          (15, d)
(7, l)          (1, u)          (1, u)          (3, u)
(3, u)          (11, r)         (15, d)         (8, r)
(6, d)          (8, r)          (7, d)          (1, l)
(1, l)          (2, l)          (2, l)          (3, u)
(4, u)          (8, u)          (9, u)          (11, u)
(12, u)         (16, u)         (17, u)         (21, l)
(21, l)
Excuting time: 28894 ms
```

IDA* Search

输入 1.txt

```
PS D:\AI\labs\lab1\digit\src> .\digit_IDAStar.exe
Please enter the digit test path and name, such as ..\input\1.txt:
..\input\1.txt
Scanning the input file: ..\input\1.txt
Please enter the result state path and name, e.g. ..\input\result.txt:
..\input\result.txt
Scanning the input result file: ..\input\result.txt
initf: 370  initg: 0  inith: 370
init stateKey: p
Find solution!:

 Total steps: 24
(1, u)          (1, u)          (6, u)          (19, l)
(15, d)         (7, l)          (14, d)         (11, d)
(14, d)         (3, r)          (2, r)          (1, u)
(11, d)         (8, r)          (7, u)          (14, l)
(16, l)         (17, u)         (6, u)          (14, l)
(15, u)         (20, l)         (21, l)         (21, l)

Excuting time: 44 ms
```

输入 2.txt

```
PS D:\AI\labs\lab1\digit\src> .\digit_IDAStar.exe
Please enter the digit test path and name, such as ..\input\1.txt:
..\input\2.txt
Scanning the input file: ..\input\2.txt
Please enter the result state path and name, e.g. ..\input\result.txt:
..\input\result.txt
Scanning the input result file: ..\input\result.txt
initf: 165  initg: 0  inith: 165
init stateKey: p
Find solution!:

 Total steps: 12
(14, d)         (6, d)          (15, d)         (7, l)
(8, l)          (9, l)          (10, u)         (13, u)
(18, u)         (11, u)         (16, u)         (21, l)

Excuting time: 30 ms
```

输入 3.txt

```
PS D:\AI\labs\lab1\digit\src> .\digit_IDAStar.exe
Please enter the digit test path and name, such as ..\input\1.txt:
..\input\3.txt
Scanning the input file: ..\input\3.txt
Please enter the result state path and name, e.g. ..\input\result.txt:
..\input\result.txt
Scanning the input result file: ..\input\result.txt
initf: 570  initg: 0  inith: 570
init stateKey: p
Find solution!:

 Total steps: 63
(6, l)          (21, r)         (17, r)         (16, d)
(3, d)          (13, d)         (4, l)          (9, u)
(4, l)          (15, d)         (7, l)          (5, u)
(5, u)          (10, r)         (10, u)         (13, r)
(15, d)         (13, r)         (7, d)          (9, r)
(4, u)          (9, r)          (4, r)          (2, u)
(2, u)          (8, r)          (6, d)          (8, d)
(6, r)          (1, u)          (1, u)          (14, u)
(11, l)         (8, d)          (15, l)         (6, l)
(7, l)          (12, u)         (3, r)          (8, r)
(12, u)         (3, u)          (11, r)         (8, r)
(14, d)         (11, r)         (6, d)          (15, d)
(7, l)          (3, l)          (3, u)          (8, u)
(8, l)          (12, d)         (9, d)          (4, r)
(3, u)          (8, u)          (11, u)         (16, u)
(17, u)         (21, l)         (21, l)
Excuting time: 21774 ms
```

X 数独问题：

算法思想

对于输入的初始数独，先做行检查、列检查、粗线宫检查、对角线检查，以确定 CSP 问题的约束条件。如果找不到可以填入的值，要么数独无解，要么回溯。用回溯的思想最终可以确定数独每个格子中的值。

优化：使用 MRV 启发式以及前向检验，选择要填数字的格子的时候选择可选空间最小的格子，前向检验来消除肯定无法填入格子的数字来减少迭代次数与回溯次数，以减少程序运行时间。

实验结果

不优化时运行时间：

```
PS D:\AI\labs\lab1\sudoku\src> .\sudoku.exe
Please enter the sudoku test path and name, such as ..\input\sudoku01.txt:
..\input\sudoku01.txt
Successfully solve Sudoku problem, please check in ..\output\sudoku01.txt
Excuting time:27 ms
请按任意键继续. . .
PS D:\AI\labs\lab1\sudoku\src> .\sudoku.exe
Please enter the sudoku test path and name, such as ..\input\sudoku01.txt:
..\input\sudoku02.txt
Successfully solve Sudoku problem, please check in ..\output\sudoku02.txt
Excuting time:26 ms
请按任意键继续. . .
PS D:\AI\labs\lab1\sudoku\src> .\sudoku.exe
Please enter the sudoku test path and name, such as ..\input\sudoku01.txt:
..\input\sudoku03.txt
Successfully solve Sudoku problem, please check in ..\output\sudoku03.txt
Excuting time:3207 ms
请按任意键继续. . .
```

优化后运行时间:

```
PS D:\AI\labs\lab1\sudoku\src> .\sudoku_Optimized.exe
Please enter the sudoku test path and name, such as ..\input\sudoku01.txt:
..\input\sudoku01.txt
Successfully solve Sudoku problem, please check ..\output\sudoku01.txt
Excuting time:12 ms
请按任意键继续. . .
PS D:\AI\labs\lab1\sudoku\src> .\sudoku_Optimized.exe
Please enter the sudoku test path and name, such as ..\input\sudoku01.txt:
..\input\sudoku02.txt
Successfully solve Sudoku problem, please check ..\output\sudoku02.txt
Excuting time:8 ms
请按任意键继续. . .
PS D:\AI\labs\lab1\sudoku\src> .\sudoku_Optimized.exe
Please enter the sudoku test path and name, such as ..\input\sudoku01.txt:
..\input\sudoku03.txt
Successfully solve Sudoku problem, please check ..\output\sudoku03.txt
Excuting time:16 ms
```

分析：

可见优化后运行时间比不优化时在数独问题复杂的时候大大缩短。不优化时由于不进行前向检验，导致对每个格子，可能填入的值都会被填入一遍作为一个结点，完全没有考虑填入当前格子值后其他格子无法填入的情况。而加入前向检验同时用 MRV 启发式后，可以保证当前填入数字后，其余受其约束的格子仍然有值可以填入。同时首先选择最少剩余可选数字的格子填入，可以保证迭代展开的结点数尽可能的少。

思考题

a)

可以爬山算法来解决。
爬山：把 CSP 的约束条件取相反数作为爬山的值，利用局部贪心使得填入一个数字后剩余的约束条件尽可能地少，如果不行则回溯，由此求解
模拟退火与遗传没有想出来。

b)

使用爬山算法时，没有选好高度值可能会对于特定的可以求解的数独，根据算法求解的时候结果为无解，或者迭代次数过多导致程序执行缓慢，因为爬山算法只关注局部值，无法对于数独问题这种需要兼顾全局的问题做太好的求解。