人工智能实验报告 Lab1

廖洲洲 PB17081504

P1:数码问题

1.启发式函数

使用`Manhattan距离`作为启发式，这是因为一个松弛问题的最优解的耗散是原问题的一个可采纳的启发式，而本题对应得松弛问题为：只要A和B相邻，棋子可以从A移动到方格B。由该松弛问题，可以得到`Manhattan距离`，故其`Manhattan距离`可以作为启发式。

2.算法思想与伪代码

- A*算法

  > A*搜索方法,是许多问题的常用启发式算法
  > 公式表示为: f(n)=g(n)+h(n), 其中, f(n) 是从初始状态经由状态n到目标状态的代价估计, g(n) 是在状态空间中从初始状态到状态n的实际代价, h(n) 是从状态n到目标状态的最佳路径的估计代价。

  其时间复杂度为$O(b^d)$,空间复杂度为$O(b^d)$，d为目标节点的深度

```
function Astar() return a solution,or a failure
    node=Initial-State
    node.pathcost=0
    initial openList and closedSet
    openList.push(node)
    while(!openList.empty){
        node=openList.pop()
        if(isEqual(node.state,destState))
            return solution(node)
        add node.state to closedSet
        ChildNodes=getChildNode(node)
        for each childNode in childNodes do
            if childNode is not in closedSet or openList then
                openList.push(childNode)
            else if childNode.state is in openList with higner path-cost
then
                replace the openList node with childNode
    }
```

- IDA*算法

> IDA*算法在是A*与ID算法的结合物，用了A*算法的预估方法和预估值(f=g+h)，用了ID算法的迭代深入(最初从Manhatton距离开始)，较之A*算法，IDA*算法不需要Open表和Closed表，大大节省了内存空间，而且IDA*算法可以不采用递归式程序设计方法，这样也可以节约堆栈空间。
>
> 在进行搜索时，使用耗散值替代ID中的深度值（f=g+h）,也就是说，搜索的范围在那些不超过给定值的节点中进行深度优先搜索。如果搜索不成功，那么返回头节点，并且使限定的耗散值变大（具体为所有超过上次限定值节点中的最小耗散）。
>
> 每次循环都进行深度优先搜索，当f超过一给定的阈值时，去掉相应的分支并进行回溯，该阈值的初始值为对初始节点路径费用的估计fs，且该阈值随着算法的每一次循环而不断增加．在每一次循环中计算用于下一次循环的阈值，其计算方法为取本次循环中路径费用超过当前阈值的那些费用中的最小值作为下一次循环的阈值。

时间复杂度为$O(b^d)$,d为目标节点的深度,IDA*算法使用递归形式

```
int dfs(int g, int bound) {
    auto node = path.back();
    //print(node.state);
    int f = node.pathCost + node.manhattanCost;
    if (f > bound)
        return f;
    if (node.manhattanCost == 0)
        return -1;
    int next_bound = INT32_MAX;
    auto childNodes = getChildNode(&node, node.manhattanCost);
    int t;
    for (auto &s:childNodes) {
        path.push_back(s.second);
        t = dfs(g + 1, bound);
        if (t < 0)
            return -1;
        if (t < next_bound)
            next_bound = t;
        path.pop_back();
    }
    return next_bound;
}
```

3.程序优化

对A*算法，刚开始时，连第二个初始状态都需要5s，故对A*算法进行优化

- 编译时开启-Ofast -march=native选项，时间减少
- `Manhattan`距离根据不同数字在不同位置保存其值，之后可以直接在表中查找，减少计算
- 刚开始使用了unordered_set作为closedSet的数据结构，因为认为hash操作能提高查找效率，但其存在扩容、缩容操作，每次扩容缩容需要对表中数据重新计算，因此复杂度或许更高
- 和目标状态的比较不一个一个节点比较，而是使用曼哈顿距离为0
- 尝试使用自己写的红黑树作为closedSet的数据结构，结果更慢了

5.实验结果

A*

初始状态	步数	时间
input1	内存不足	
input2	12	2s
input3	内存不足	

IDA*

初始状态	步数	时间
input1	24	0.001000s
input2	12	0.000000s
input3	内存不足	

可以看出IDA*算法效率远远好于A*算法，这是因为其不用像A*那样消耗大量内存

P2:X数独问题

1.算法思想及优化

- 回溯搜索算法：用于深度优先搜索中，它每次为一变量选择一个赋值，当没有合法的值可以赋给变量时就回溯。
- 最少剩余值启发式(MRV启发式)：也称为最受约束变量启发式，它选择了最可能很快导致失败的变量，从而对搜索树剪枝。如果变量X没有可选的合法取值，那么MRV启发式将选择X并马上检测到失败，避免无意义的搜索。即选择变量值域最小的变量。
- 度启发式：选择与其它未赋值变量约束最多的变量来试图降低未来的分支因子。即每次选择具有最多约束的变量，在数独中也就是行、列、宫格和对角线上空位最少的变量。
- 前向检验：每次进行尝试赋值后，减少当前所有变量的值域
- 前向检验+MRV思路：
    1. 使用MRV启发式选择一个变量，即优先选择值域空间小的变量
    2. 遍历该变量值域，根据变量的赋值，调整相关变量的值域空间，若不会导致有变量值域为空，则向下递归
    3. 若有解则结束，否则恢复相关变量的值域空间，回溯

2.结果分析

- 时间

算法	sudoku01	sudoku02	sudoku03
backtracking	0.000s	0.015s	1.281s
度启发式	0.011s	0.013s	0.196s
MRV	0.001s	0.001s	0.303s
前向检验	0.001s	0.008s	0.891s
MRV+前向检验	0.000s	0.000s	0.000s

- 遍历的节点数

算法	sudoku01	sudoku02	sudoku03
backtracking	858	137341	38255815
度启发式	261	4948	1330405
MRV	214	5688	4004380
前向检验	167	18734	4384596
MRV+前向检验	46	55	64

由此可以看出，使用优化对算法效率有很大提升，当单独使用某一种优化时，遍历的节点数会降低一个数量级，而当使用MRV和前向检验结合时，算法访问的时间和遍历节点数大大减少，尤其时遍历的节点数降至一百以下。这是因为通过变量的选择， 使得算法能在早期进行有效剪枝，从而有助于最小化搜索树中的节点数，而通过前向检验，又使得节点的值域更小了，有利于节点数的减少。

3.思考题

可以

对模拟退火和爬山算法：给数独问题赋予能量，设状态的能量为行列出现的数字冲突数量总和，即对一个九宫格，同一行、同一列如果有任何两个相同的数字则能量加一。于是我们将数独问题转化成了寻找状态最低能量的问题，然后使用爬山算法或退火算法求解即可。

遗传算法：把两个父状态结合来生成后继，而不是通过修改单一状态进行。每个状态使用一个有限长度的字符串表示。

1. 初始化化种群
   首先需要产生较优的初始种群，以减少进化代数，如果没有较优的初始种群会加大后面运算压力。为了得到较优的初始种群，设置一下规则：每个方格的数字不重复、尽可能使填入的数字与所在行或列的数字不重复。根据以上规则得到一定数量的初始九宫格，然后将每个方格缺的数字按从上到下、从左到右的顺序连在一次作为染色体。
2. 交叉
   将染色体随机两两组合，随机取两个染色体中间相同的位置进行交换，交叉完后，将未交叉的重复元素用另一个染色体的重复的元素交换（因为该染色体重复的元素就是另一个染色体缺少的元素，元素守恒）。
3. 变异
   按变异率在种群中随机选择一定数量的个体，随机产生一个变异节点（一个九宫格的方格作为一个节点），将该节点左右翻转。
4. 选择
   将父代、子代、变异代三部分染色体合在一起，计算每个染色体还原到九宫格中行和列重复数字的个数，初始分为8*（9+9）=144，每重复一次减去一分。选出分数最高一部分作为下一轮进化的父代。进化到一定程度，出现分数等于144时，退出进化。