人工智能基础第一次实验报告

- 焦培淇　PB17151767

实验一：数码问题

程序运行方式

请见src目录的readme文件

启发式函数

算法选择的启发式函数为当前数码图中的元素与其在结果图中的位置的横纵坐标差的绝对值的和（此时对于节点7来讲只考虑一个位置的移动步数）。很明显，该启发式函数是可采纳的，因为在一个没有元素阻碍的数码图中，移动一个元素到其最终位置所需的最小步数就是该元素与其在结果图中的位置的横纵坐标差的绝对值的和，所以在这个问题中，该启发式函数是可采纳的。

同时构造启发式函数2，该函数大体上采用上述曼哈顿距离式启发式，但是对于节点7，考虑其三个位置的移动步数和（即为上述的7节点的数值的3倍），对于该启发式，并不能证明是可采纳的，因此作为探索只用。

代码中对于启发式函数的选择请修改宏定义H_NUM变量，1代表第一种，2代表第二种。

算法伪代码概括

普通A*算法

```
a_start(start_state)
    let list be a priority queue;
    list.push(start_state);
    while(list not empty)
        minNode = list.pop;
        if the first zero in minNode is moveable:
            move first zero and get all the next states
            for state in the next states:
                if it is the end state
                    return success
                if it is not a repeated state
                    push it into the list
        endif
        if the second zero in minNode is moveable:
            move second zero and get all the next states
            for state in the next states:
                if it is the end state
                    return success
                if it is not a repeated state
                    push it into the list
        endif
        if the whole seven block in minNode is moveable:
```

```
            move the whole seven block and get all the next states
            for state in the next states:
                if it is the end state
                    return success
                if it is not a repeated state
                    push it into the list
        endif
    endwhile
    return fail
```

迭代加深的A*算法

```
a_start(start state)
    let the f of start state be bound
    while bound < max_bound
        result = DFS_astar(start state,bound)
        if result is success
            return success
        else
            bound = result
    endwhile
    return out of bound

DFS_astar(state,bound)
    if state.f > bound
        return state.f
    if state == final_state
        return success
    let next be the next non-repeating state of state
    while(state has another next non-repeating state)
        f = DFS_astar(next, bound)
        if f < min
            min = f
        if f == success
            return success
        let next be another next non-repeating state of state
    endwhile
    return min
```

算法时间复杂度分析

对于普通的A*算法

时间复杂度：对于普通A*算法，算法中大部分函数直接对数码图进行遍历，因此时间复杂度都是$O(N^2)$，在这里N=5，对于A*算法的主体，设最后求出的路径长度为m，那么最好情况下的时间复杂度就是$O(mN^2)$，而最坏情况下的时间复杂度为$O(8^m*N^2)$，因为每个节点处存在最多八种可能的移动情况。

空间复杂度：对于普通A*算法，空间主要使用在每个结点的存储上，每个结点的空间复杂度为$O(N^2)$，如果设最后求出的路径长度为m，那么最好情况下空间复杂度就是$O(mN^2)$，最坏情况为$O(8^m*N^2)$。

对于迭代加深的A*算法来讲

时间复杂度：分析同上述的普通A*算法，最好情况下的时间复杂度就是$O(mN^2)$，而最坏情况下的时间复杂度为$O(8^m*N^2)$。

空间复杂度：对于迭代加深来讲，每次仅保存当前的路径节点，因此要节省空间复杂度，最好情况下空间复杂度就是$O(mN^2)$，最坏情况为$O(8m*N^2)$。

不同算法在不同初始状态下的执行比较

首先考虑在第一种可证明可采纳的启发式函数下

普通A*算法的结果

测试样例号	运行时间	解的步数
0	0s(时间小于1ms)	24
1	0s(时间小于1ms)	12
2	N/A	N/A

迭代加深A*算法

测试样例号	运行时间	解的步数
0	0.034s	24
1	0.01s	12
2	N/A	N/A

N/A表示算法需要内存空间过大，无法得出结果。

此时考虑进行优化，采用设计的第二种启发式函数，该函数没能证明为可采纳，但是可以大大降低搜索空间的大小，结果如下：

普通A*算法的结果

测试样例号	运行时间	解的步数
0	0s(时间小于1ms)	26
1	0s(时间小于1ms)	12
2	144.682s	57

迭代加深A*算法

测试样例号	运行时间	解的步数

测试样例号	运行时间	解的步数
0	0.034s	40
1	0.01s	14
2	77.998s	79

在第二种启发式函数下，整个算法可以运行出结果，但是并不是最优解。

从这里我们可以看出：

- 曼哈顿距离作为可采纳的启发式在A*和标准IDA*都可以得出最优解，但对于一些搜索空 间很大的问题，不能有效剪枝缩减搜索空间，导致资源占用高，耗时长。
- 不可采纳的启发式往往能够有效缩减搜索空间，减少算法资源占用和所花时间，但不一 定能得出全局最优解。

实验二：数独问题

程序运行方式

请见src目录的readme文件

算法思想

普通的回溯型搜索算法，采用对于问题中每个没有填入值的位置，尝试填入不同的符合约束的值，递归调用算法，直到全部位置都被填入符合约束的值，算法结束，输出结果。

优化：对于普通的回溯算法，采用MRV+前向检验的方式进行优化，MRV用于决定选择变量的顺序，前向检验来减少搜索空间的大小。具体实施如下：

首先对于所有没有填入值的位置，计算其所有合法值并统计个数，将上述信息存储在数组A中，并按照合法值的个数从大到小排列。每次循环，取有最小合法值个数的节点，对于其拥有的合法值进行尝试赋值，再次刷新数组A并排序，检验有最小合法值个数的节点的合法值个数是否为0，若不为0，递归调用算法，直到全部节点都被填入合法值，算法结束。由于每次回溯都需要不断的调用排序算法，这里采用每次只选择有最小剩余值得办法，避免得过多得排序算法。

结果分析

- 简单回溯算法

测试样例号	运行时间	访问结点数
01	0s(时间小于1ms)	111
02	0.002s	14853
03	0.393s	4233934

- 优化后算法

测试样例号	运行时间	访问结点数

测试样例号	运行时间	访问结点数
01	0s(时间小于1ms)	47
02	0s(时间小于1ms)	79
03	0.016s	532

通过对比可见，采用优化后的算法，搜索空间相比简单的回溯算法大大降低，运行时间也大幅下降，可见优化算法确实起到了很好的效果

思考题

- a

可以使用爬山算法、模拟退火算法和遗传算法来求解这个问题。

对于爬山算法：可以先对问题设置一个初始解作为当前解，通过一个评价函数来评估当前解的矛盾程度，该评估函数值越小，说明当前解越接近最终解。通过每一个位置的数字展开搜索，选择该位置的最终取值就是使得评估函数值最小的那个值，当评估函数值无法再减小时，算法结束。

模拟退火算法：首先找出一个初始解作为该问题的当前解，设计一个评价函数来评估当前解的矛盾程度，如果该评估函数值越小，说明当前解越接近最终解。对数独内的每一个位置展开搜索，刚开始搜索时，除了尝试填入该位置的最优值，还会去尝试使用一些次优值作为该位置的值，但随着次数的增加，选择那些并不最优的值的概率就越低，最终概率会趋于0，当评估函数值无法再减小时，算法结束并得到最优解。

遗传算法：首先找出一个初始解，设计一个评价函数来评估当前解的矛盾程度，该评价函数值越小，说明当前解越接近最终解。对每一个位置展开搜索，每次选取几个较优的值进行下一次的搜索，同时考虑让那些优秀后代（即选取的几个最优的值）之间进行合并和变异，目的就是使得评估函数的值越来越小，在这个过程中，及时的将那些评估函数值过大的后代淘汰，经过不断的迭代变异进化，最终种群达到最优时，得到的后代就是该算法的最终解。

- b

可能出现的问题有：

爬山算法：可能只求到局部的最优解。

模拟退火算法：该算法问题在于如何控制搜索非最优解的概率以及搜索哪些非最优解，如果搜索非最优解太过于频繁，算法性能必然很低，如果搜索非最优解概率过低，就会退化为爬山算法的情况，从而只能得到局部最优解。

遗传算法：该算法的问题是如何有效的对几个较优的解进行合并和变异才能使得新产生的后代的评价函数值进一步减小。