人工智能实验报告 第5周

姓名:刘卓逸 学号:21307303

一.实验题目

hw4 启发式搜索算法

二.实验内容

1.算法原理

```
将初始状态加入openlist
while (openlist不为空且目标状态未进入closedlist) do
    在openlist中找到估值函数最小的状态x
    将x从openlist中移除并加入到closedlist
    从x扩展出下一步的状态
    若下一步的状态不在closedlist中
        将这个状态加入到openlist中
        并且尝试更新新状态的估值函数
            更新成功则更新标记
判断目标状态在closedlist中
```

```
    若存在，则找到了解，返回解
    否则无解
```

2.实验过程与关键代码展示

(1)哈希

二维列表似乎因为无法被自动哈希而无法直接作为字典的键，再加上二维列表消耗空间大，拷贝麻烦且慢，于是考虑写一个哈希函数

每一个数只可能是0~15,用4位即可表示，每个状态共16个数，故可以用一个64位无符号整数表示一个状态

具体哈希函数如下:

```
def encode(puzzle): #编码
    ans=0
    for i in range(4):
        for j in range(4):
            ans+=puzzle[i][j]*(1<<(4*(i*4+j)))
    return ans
```

显然这个哈希函数是可逆的，可以通过将哈希值转换回二维列表，方便状态转移

```
def decode(x): #解码
    puzzle=[ [0 for j in range(4)] for i in range(4)]
    for i in range(4):
        for j in range(4):
            puzzle[i][j]=x&15
            x>>=4
    return puzzle
```

(2)估值函数

定义估值函数f(x)=g(x)+h(x)

g(x)为初始状态到目前状态所走的步数

h(x)为每一个数离它的目标位置的曼哈顿距离之和

```
def miraishi(puzzle:list): #未来视 估值函数
    ans=0
    for i in range(4):
        for j in range(4):
            ii=((puzzle[i][j]+15)%16)/4
            jj=(puzzle[i][j]+3)%4
            ans+=abs(i-ii)+abs(j-jj)
    return ans
```

(3)辅助的函数

找出0(空位)在哪的函数

```
def findzero(puzzle:list):
    for i in range(4):
        for j in range(4):
            if puzzle[i][j]==0:
                return (i,j)
```

产生下一个状态的函数

```
def move(nw:list,xz,yz,xi,yi): #生成新的状态,并且[xz,yz]与[xi,yi]交换
    nxt=[[nw[i][j] for j in range(4)] for i in range(4)]
    nxt[xz][yz],nxt[xi][yi]=nxt[xi][yi],nxt[xz][yz]
    return nxt
```

(4)测试用代码

```
if __name__ == '__main__':
    import time
    print("go task0")
    puzzle = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [0, 13, 14, 15]]
    start=time.time()
    sol = A_star(puzzle)
    end=time.time()
    print("time:",end-start)
    print(len(sol),sol)

    print("go task1")
    puzzle = [[1, 2, 4, 8], [5, 7, 11, 10], [13, 15, 0, 3], [14, 6, 9, 12]]
    start=time.time()
    sol = A_star(puzzle)
    end=time.time()
    print("time:",end-start)
    print(len(sol),sol)

    print("go task2")
    puzzle = [[5, 1, 3, 4], [2, 7, 8, 12], [9, 6, 11, 15], [0, 13, 10, 14]]
    start=time.time()
    sol = A_star(puzzle)
    end=time.time()
    print("time:",end-start)
    print(len(sol),sol)

    print("go task3")
    puzzle = [[14, 10, 6, 0],[4, 9 ,1 ,8],[2, 3, 5 ,11],[12, 13, 7 ,15]]
    start=time.time()
    sol = A_star(puzzle)
    end=time.time()
    print("time:",end-start)
    print(len(sol),sol)

    print("go task4")
    puzzle = [[6, 10, 3, 15],[14, 8, 7, 11], [5, 1, 0, 2],[13, 12, 9, 4]]
    start=time.time()
    sol = A_star(puzzle)
    end=time.time()
    print("time:",end-start)
    print(len(sol),sol)

    print("go task5")
    puzzle = [[11, 3, 1, 7],[4, 6, 8, 2], [15, 9, 10, 13],[14, 12, 5, 0]]
    start=time.time()
    sol = A_star(puzzle)
    end=time.time()
    print("time:",end-start)
    print(len(sol),sol)

    print("go task6")
    puzzle = [[0, 5, 15, 14],[7, 9, 6, 13], [1, 2, 12, 10],[8, 11, 4, 3]]
    start=time.time()
    sol = A_star(puzzle)
    end=time.time()
    print("time:",end-start)
    print(len(sol),sol)
```

(5)核心代码(初版)

在找出估值函数最小的状态时，用的方法是是暴力遍历整个openlist

```
def A_star(puzzle):
    fa=[(-1,0),(1,0),(0,1),(0,-1)]
    ans=[]
    origin=encode(puzzle)
    finall=encode([[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,0]])
    trap=encode([[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,15,14,0]])
    fr={origin:[0,0,0]} #from来自信息哈希值:[动了几步，动了什么，上一步的hash]
    op={origin:miraishi(puzzle)} #openinglist{哈希值:估值函数值}
    cl=set() #closedlist
    while len(op)>0 and (not (finall in cl)):
        nwh=0
        nwv=1145141919 #INf
        for (oph,opv) in op.items(): #寻找估值函数最小的
            if nwv>opv:
                nwh=oph
                nwv=opv
        cl.add(nwh) #加入closed
        del op[nwh] #从open中删除
        nw=decode(nwh) #解码
        (xz,yz)=findzero(nw) #找到0在哪
        for i in range(4):
            (xi,yi)=(xz+fa[i][0],yz+fa[i][1])
            if (xi>=0 and xi<=3 and yi>=0 and yi<=3): #开始扩展
                nxt=move(nw,xz,yz,xi,yi)
                nxth=encode(nxt)
                #print("go",nxt)
                if (nxth in cl):
                    continue
                if (nxth==trap):
                    print("fuck this")
                    return []
                if nxth in op.values():
                    if fr[nxth][0]>fr[nwh][0]+1:
                        fr[nxth]=[fr[nwh][0]+1,nw[xi][yi],nwh]
                        op[nxth]=fr[nxth][0]+miraishi(nxt)
                else:
                    fr[nxth]=[fr[nwh][0]+1,nw[xi][yi],nwh]
                    op[nxth]=fr[nxth][0]+miraishi(nxt)
    if not(finall in cl):
        print("No answer")
        return []
    i=finall
    while (i!=origin):
        ans.append(fr[i][1])
        i=fr[i][2]
    ans.reverse()
    return ans
```

结果是跑task3就卡死了，一分多钟没有出结果，所以必须得优化

(6)优化与重构

优化思路:

1.将标记fr的值设为结构体增强可读性

```
class Node:
    def __init__(self,gx,hx,action,parent):
        self.gx=gx
        self.hx=hx
        self.action=action
        self.parent=parent
    def fx(self):
        return int(self.hx+self.gx)
    def __lt__(self,other):
        return self.fx()<other.fx()
```

2.编码等地方用位运算代替取模与除法

```
def encode(puzzle): #编码
    ans=0
    for i in range(4):
        for j in range(4):
            ans+=puzzle[i][j]*(1<<(((i<<2)+j)<<2))
    return ans
```

3.估值函数

直接曼哈顿距离相加过于简单，进行如下考虑：

当一个数离目标位置的距离为1时，则期望用一步就能移动过去，但当距离为2时，不可能只用2步就移动过去。考虑给每种曼哈顿距离赋不同的权重，如曼哈顿距离为1时，期望为1,；曼哈顿距离为2时，期望为3等等。虽然这样无法保证得到最优解，但是能快速得到较优解。

经过手动调整参数，平衡了解的优越性与时间，得到以下估值函数

```
def heuristic(puzzle): #启发式函数
    miracle=[0,1,4,6,9,11,14]
    ans=0
    for i in range(4):
        for j in range(4):
            if (puzzle[i][j]==0):
                continue
            ii=((puzzle[i][j]+15)&15)>>2
            jj=(puzzle[i][j]+3)&3
            ans+=miracle[abs(i-ii)+abs(j-jj)]
    return int(ans)
```

4.主函数：一个状态的估值hx部分只需算一遍即可，生成结构体时直接储存在下来不再修改; 考虑最终步数应该不会特别特别大，所以考虑用桶以估值函数为键来装openlist; 由于[[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,15,14,0]]是无解的，增加对无解情况的检测

```
def A_star(puzzle):
    fa=[(-1,0),(0,-1),(1,0),(0,1)] #方向数组
    maxstep=512
    origin=encode(puzzle)
    finall=encode([[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,0]]) #最终状态
    if (origin==finall):
        return []
    trap=encode([[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,15,14,0]]) #死胡同状态
    if (origin==trap):
        print("No answer")
        return []

    infos={ origin: Node(0,heuristic(puzzle),0,0) } #信息，gx已走步数、hx估值函数、action上一步走了什么、parent从什么状态转移来
    cl=set() #closedlist
    op=[ [] for i in range(maxstep) ] #openlist 桶排序,op[x]为所有估值函数为x为状态组成的列表
    op[infos[origin].fx()].append(origin)
    ops=infos[origin].fx()
    opset={origin} #是否在op中
    while (not (finall in cl)):
        while (ops<maxstep and len(op[ops])==0):
            ops+=1
        if (ops==maxstep):
            print("not found")
            return []
        nwh=op[ops][len(op[ops])-1]
        op[ops].pop() #从op中删除
        if (nwh in cl):
            continue
        opset.remove(nwh)
        cl.add(nwh) #加入closed
        nw=decode(nwh) #解码
        (xz,yz)=findzero(nw) #找到0在哪
        for i in range(4):
            (xi,yi)=(xz+fa[i][0],yz+fa[i][1])
            if (xi>=0 and xi<=3 and yi>=0 and yi<=3): #开始扩展
                nxt=move(nw,xz,yz,xi,yi)
                nxth=encode(nxt)
                if (nxth in cl):
                    continue
                if (nxth==trap):
                    print("No answer")
                    return []
                if nxth in opset:
                    if infos[nxth].gx>infos[nwh].gx+1:
                        infos[nxth].gx=infos[nwh].gx+1
                        op[infos[nxth].fx()].append(nxth)
                        if infos[nxth].fx()<ops:
                            ops=infos[nxth].fx()
                else:
                    infos[nxth]=Node(infos[nwh].gx+1,heuristic(nxt),nw[xi][yi],nwh)
                    op[infos[nxth].fx()].append(nxth)
                    opset.add(nxth)
                    if infos[nxth].fx()<ops:
                        ops=infos[nxth].fx()
    i=finall
    ans=[]
    while (i!=origin):
        ans.append(infos[i].action)
        i=infos[i].parent
    return ans[::-1]
```

经过改进后，速度大幅提升，原本卡死的样例能快速通过

三.实验结果及分析

实验结果展示示例

1.实验结果展示示例

（1）初版代码:

(直接卡死，等不出结果)

（2）改进后的代码(启发式函数参数为[0,1,4,5,9,11,14]):

```
go task0
time: 0.0
3 [13, 14, 15]
go task1
time: 0.003981590270996094
22 [15, 6, 9, 15, 11, 10, 3, 11, 10, 3, 8, 4, 3, 7, 6, 9, 14, 13, 9, 10, 11, 12]
go task2
time: 0.0
15 [13, 10, 14, 15, 12, 8, 7, 2, 5, 1, 2, 6, 10, 14, 15]
go task3
time: 0.3488471508026123
63 [6, 1, 5, 3, 9, 10, 14, 4, 2, 9, 10, 14, 4, 2, 9, 12, 13, 10, 14, 5, 1, 4, 5, 1,
3, 7, 10, 14, 12, 9, 1, 5, 2, 1, 5, 12, 14, 10, 7, 14, 10, 7, 14, 11, 8, 6, 4, 3,
12, 10, 7, 14, 11, 12, 6, 8, 12, 7, 10, 6, 7, 11, 15]
go task4
time: 0.9540331363677979
56 [2, 11, 15, 3, 7, 8, 14, 5, 1, 14, 8, 2, 9, 12, 14, 9, 11, 15, 2, 8, 5, 1, 9,
11, 15, 4, 12, 15, 8, 2, 4, 8, 11, 5, 2, 7, 10, 2, 5, 11, 7, 10, 2, 6, 1, 5, 10, 7,
11, 10, 6, 2, 3, 4, 8, 12]
go task5
time: 0.3412020206451416
70 [13, 10, 5, 12, 14, 15, 9, 5, 10, 13, 12, 14, 15, 9, 5, 6, 4, 11, 3, 1, 7, 2, 8,
4, 11, 5, 6, 10, 13, 8, 4, 11, 10, 13, 11, 7, 2, 4, 7, 11, 14, 15, 13, 14, 11, 7,
4, 2, 1, 3, 5, 6, 9, 13, 14, 10, 3, 1, 2, 4, 7, 3, 6, 5, 1, 2, 3, 7, 8, 12]
go task6
time: 0.129746675491333
82 [5, 9, 2, 11, 8, 1, 7, 5, 9, 2, 6, 13, 10, 3, 4, 8, 11, 6, 13, 10, 3, 4, 8, 11,
6, 13, 10, 15, 14, 3, 4, 12, 11, 6, 13, 7, 1, 13, 6, 11, 15, 14, 2, 10, 7, 6, 11,
15, 14, 7, 6, 14, 12, 8, 15, 11, 14, 6, 10, 9, 5, 1, 6, 10, 9, 5, 1, 9, 5, 2, 3, 4,
8, 12, 10, 6, 9, 5, 6, 10, 11, 15]
```

（3）若将启发式函数中的参数调整成[0,1,4,5,9,11,14]：

```
go task0
time: 0.0
3 [13, 14, 15]
go task1
time: 0.008007049560546875
22 [15, 6, 9, 15, 11, 10, 3, 11, 10, 3, 8, 4, 3, 7, 6, 9, 14, 13, 9, 10, 11, 12]
go task2
time: 0.0
15 [13, 10, 14, 15, 12, 8, 7, 2, 5, 1, 2, 6, 10, 14, 15]
go task3
time: 0.037346601486206055
57 [6, 1, 5, 3, 9, 4, 14, 10, 1, 5, 4, 14, 2, 9, 14, 2, 10, 1, 2, 10, 9, 12, 13,
14, 12, 9, 1, 2, 5, 4, 3, 11, 8, 6, 4, 3, 6, 8, 11, 12, 10, 5, 2, 1, 5, 6, 8, 11,
12, 7, 15, 12, 11, 8, 7, 11, 12]
go task4
time: 1.38558030128479
56 [2, 11, 15, 3, 7, 8, 14, 5, 1, 14, 8, 2, 9, 12, 14, 9, 11, 15, 2, 8, 5, 1, 9,
11, 15, 4, 12, 15, 8, 7, 10, 6, 1, 5, 7, 10, 3, 2, 4, 8, 11, 7, 10, 3, 2, 4, 8, 11,
7, 10, 6, 2, 3, 7, 11, 12]
go task5
time: 9.928896427154541
84 [5, 12, 14, 15, 9, 10, 13, 5, 12, 13, 5, 2, 8, 1, 3, 11, 4, 6, 1, 5, 2, 8, 5, 2,
8, 5, 7, 3, 2, 8, 5, 7, 8, 5, 7, 8, 5, 2, 11, 4, 6, 1, 2, 5, 8, 7, 5, 11, 4, 2, 10,
5, 13, 14, 15, 9, 5, 13, 14, 15, 13, 14, 11, 4, 2, 6, 1, 5, 9, 13, 14, 11, 7, 8, 4,
7, 11, 10, 6, 2, 3, 4, 8, 12]
go task6
time: 6.183424711227417
102 [5, 9, 2, 11, 4, 3, 10, 12, 3, 4, 8, 1, 7, 5, 9, 2, 6, 3, 4, 10, 12, 4, 10, 8,
11, 10, 8, 11, 10, 8, 4, 13, 14, 15, 3, 14, 15, 3, 2, 6, 14, 4, 13, 15, 4, 14, 6,
2, 14, 6, 8, 7, 5, 9, 2, 14, 6, 8, 7, 13, 8, 7, 14, 6, 3, 4, 15, 8, 11, 10, 13, 14,
7, 15, 8, 11, 14, 5, 1, 13, 10, 14, 15, 7, 5, 1, 9, 5, 1, 9, 5, 1, 6, 2, 1, 5, 9,
10, 14, 15, 11, 12]
```

（4）若将启发式函数参数调整成[0,1,3,5,9,11,14]：

```
go task0
time: 0.003996610641479492
3 [13, 14, 15]
go task1
time: 0.0
22 [15, 6, 9, 15, 11, 10, 3, 11, 10, 3, 8, 4, 3, 7, 6, 9, 14, 13, 9, 10, 11, 12]
go task2
time: 0.0
15 [13, 10, 14, 15, 12, 8, 7, 2, 5, 1, 2, 6, 10, 14, 15]
go task3
time: 2.0419180393218994
59 [6, 1, 5, 3, 9, 4, 14, 10, 4, 14, 2, 9, 14, 2, 9, 12, 13, 14, 12, 9, 2, 5, 1, 4,
10, 2, 5, 1, 3, 11, 8, 6, 4, 3, 6, 8, 11, 12, 9, 5, 1, 10, 2, 1, 5, 9, 10, 6, 8,
11, 12, 7, 15, 12, 11, 8, 7, 11, 12]
go task4
```

```
time: 2.2302136421203613
54 [2, 11, 15, 3, 7, 8, 14, 5, 1, 14, 10, 7, 8, 2, 9, 12, 14, 10, 2, 15, 11, 4, 12,
14, 10, 9, 15, 8, 7, 2, 5, 1, 9, 10, 14, 15, 4, 11, 8, 4, 11, 8, 4, 7, 2, 6, 1, 5,
6, 2, 3, 4, 8, 12]
go task5
time: 7.842634201049805
62 [13, 10, 5, 12, 14, 15, 9, 5, 8, 1, 3, 11, 4, 6, 1, 2, 10, 8, 12, 14, 15, 9, 6,
1, 11, 4, 1, 6, 5, 11, 2, 10, 8, 12, 11, 15, 14, 13, 12, 11, 15, 14, 13, 15, 10, 3,
4, 2, 6, 5, 9, 13, 14, 10, 11, 8, 7, 4, 3, 7, 8, 12]
go task6
time: 1.1390786170959473
86 [7, 9, 2, 11, 8, 1, 9, 7, 5, 2, 6, 13, 10, 3, 4, 8, 11, 6, 13, 10, 3, 4, 8, 11,
6, 13, 10, 15, 14, 3, 4, 8, 11, 12, 15, 14, 3, 4, 8, 11, 12, 15, 14, 10, 7, 9, 1,
6, 13, 14, 10, 7, 9, 1, 6, 13, 14, 6, 1, 9, 7, 10, 11, 12, 15, 11, 6, 1, 9, 7, 1,
6, 10, 3, 2, 1, 7, 5, 1, 2, 3, 7, 6, 10, 11, 15]
```

2.评测指标展示及分析

不难看出，启发式函数对解的优越性与时间有至关重要的影响。在以上例子中可以看出，在无法保证h(x)<h*(x)的前提下，修改h(x)会导致部分样例解更优但部分样例解更差，耗费时间上也有相当大的差别。