中山大学计算机学院

人工智能

本科生实验报告

(2022学年春季学期)

课程名称：Artificial Intelligence

教学班级	20级计科一班	专业（方向）	计算机科学与技术
学号	20337056	姓名	李昊伟

一.实验题目

使用A*和IDA*两种算法解决十五数码问题。

二.实验内容

1. 算法原理（说实话，我用了很久才真正搞懂这一部分）

A*:

A*算法其实主要是利用open和close两个数据结构.

close可以理解为一个哈希表，一个"黑名单"，所有已经被open队列pop出的状态被记录在这个黑名单当中，不可以再进入open队列（道理其实和ucs相似）.

open可以理解为一个优先队列： 启发函数值h加上深度g为f , f 小的优先在堆顶被弹出. 每一个弹出的状态如果是目标状态，搜索结束；如果不是， 就把这个状态的四个方向移动过的子状态压入队列。如果子状态已经在队列中 , 就对比二者的f大小，留下f小的，舍弃f大的，才能保证最优性.

IDA*:

IDA*算法是一种迭代加深的深度优先搜索算法，但是其不再使用深度作为bound，而是采用maxH。

2. 伪代码

A*:

```
* 初始化open_set和close_set；
* 将起点加入open_set中，并设置优先级为0（优先级最高）；
* 如果open_set不为空，则从open_set中选取优先级最高的节点n：
    * 如果节点n为终点，则：
        * 从终点开始逐步追踪parent节点，一直达到起点；
```

```
        * 返回找到的结果路径，算法结束；
    * 如果节点n不是终点，则：
        * 将节点n从open_set中删除，并加入close_set中；
        * 遍历节点n所有的邻近节点：
            * 如果邻近节点m在close_set中，则：
                * 跳过，选取下一个邻近节点
            * 如果邻近节点m也不在open_set中，则：
                * 设置节点m的parent为节点n
                * 计算节点m的优先级
                * 将节点m加入open_set中
```


IDA*:

```
function IDA_search(node, g, bound):
    // node当前节点
    // g当前节点的路径消耗值
    // bound当前搜索的一个界限值
    f = g + h(node)
    if f > bound:
        return f
    if node == B:
        return FOUND
    min = r
    for succ in successors(node) do:
        t = search(succ, g + cost(node, succ), bound)
    if t == FOUND:
        return FOUND
    if t < min:
        min = t
    return min

bound = h(A)
while(True):
    t = IDA_search(root, 0, bound)
    if t == FOUND:
        return bound
    if t = ∞:
        return NOT_FOUND
    bound := t
```


3 .关键代码展示

A*:

```
#扩展函数
def expand(open,open_hashtable,close_hashtable,dic):
    tar = open.get()
    list1 = tar.get_state()
    close_hashtable.add(hash(str(list1)))
    moves = [(1, 0), (-1, 0), (0, 1), (0, -1)]
    zero_position = find_zero(list1)
    x = zero_position[0]
    y = zero_position[1]
    for i, j in moves:
        a, b = x+i, y+j
        if(a<4 and a>-1 and b<4 and b>-1):
            list2 = copy.deepcopy(list1)
            list2[x][y] = list2[a][b]
            list2[a][b] = 0
            depth_now = tar.get_depth()
            newnode = node(list2,depth_now+1,tar)
            if is_goal(list2):
                print("game finished! The process is:")
                print_result(newnode)
                return 1,tar
            if not(newnode.hash_val in close_hashtable):
                if not(newnode.hash_val in open_hashtable):
                    open.put(newnode)
                    dic[newnode.hash_val] = newnode
                    open_hashtable.add(newnode.hash_val)
                else:
                    if dic[newnode.hash_val].f>newnode.f:
                        dic[newnode.hash_val].f = newnode.f
                        dic[newnode.hash_val].parent = tar
                        dic[newnode.hash_val].depth = tar.depth+1
    return 0
```

```
#主函数
def main():
    import time
    start = time.time()
    dic = dict()
    state = [[14,2,8,1], [7,10,4,0], [6,15,11,5], [9,3,13,12]]
    open = PriorityQueue()
    first_node = node(state,0,node([],0,""))
    dic[first_node.hash_val] = first_node
    open.put(first_node)
    flag = 0
    open_hashset = set()
    open_hashset.add(first_node.hash_val)
    close_hashset = set()
    p = 0
    result = 0
    times = 0
    while(result == 0):
        result = expand(open,open_hashset,close_hashset,dic)
        # print(len(open))
        times = times + 1
```

```
        # print(hashset.__len__())
        # print("poped ",times," times!")
    end = time.time()
    print(end-start)
```

IDA*

```
#扩展函数
def frontier(node):
    x, y = 0, 0
    for i in range(4):
        for j in range(4):
            if(node[i][j] == 0):
                x, y = i, j
    frontier = []
    moves = [(1, 0), (-1, 0), (0, 1), (0, -1)]
    for i, j in moves:
        a, b = x+i, y+j
        if(a<4 and a>-1 and b<4 and b>-1):
            temp = [[num for num in col] for col in node]
            temp[x][y] = temp[a][b]
            temp[a][b] = 0
            frontier.append(temp)

    return sorted(frontier, key=lambda x: h(x))
```

```
#深度优先搜索
def dfs(path, g, bound):
    node = path[-1]
    f = g + h(node)
    if(f > bound):
        return f
    if(is_goal(node)):
        return -1
    m = 9999
    for ir in frontier(node):
        if ir not in path:
            path.append(ir)
            t = dfs(path, g+1, bound)
            if(t == -1):
                return -1
            if(t < m):
                m = t
            path.pop()
            # print("pop!")

    return m
```

```
# 主函数
def main():
    build_target()
    state = [[14,2,8,1], [7,10,4,0], [6,15,11,5], [9,3,13,12]]
    import time
    start = time.time()
    limit =  70
```

```
    (path, bound) = main_funct(state,limit)
    step = 0
    for p in path:
        print('step', step)
        step += 1
        for row in p:
            print(row)
    end = time.time()
    print('bound', bound)
    print('time',end-start)
```

4. 创新点&优化

相较于版本1，主要修改以下方面：
1.修改启发函数，删除0的曼哈顿距离，使得启发式函数具有可采纳性。

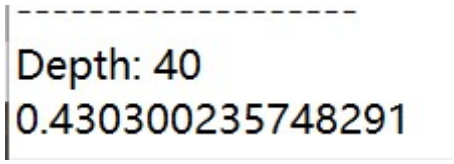
2.采用课上同学分享的方法，对搜索树进行剪枝。

3.比较是否达到搜索目标时将二维数组转化为字符串比较。

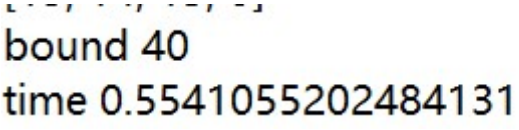
5. 实验结果与分析

example1:

A_star:

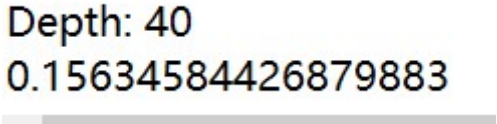


IDA_star:



example2:

A_star:



IDA_star:

example3:

A_star:

```
------------------
Depth: 40
0.067491292953491 21
```

IDA_star:

```
[13, 14, 15, 0]
bound 40
time 0.09345579147338867
```

example4:

A_star:

```
------------------
Depth: 40
1.3467998504638672
```

IDA_star:

```
bound 40
time 5.373098611831665
```

ppt_例子2

A_star:

```
Depth: 49
404597
58.39620566368103
```

ppt_例子4

A_star:

```
------------------
Depth: 48
2964935
369.8874089717865
```

在pop次数较多， A_star表现出在速度上的优势。

6.思考题

quiz1：

元组可以作为字典的键,列表不能。

```
1   dic = dict()
2   dic[(1,2)] = "a"
3   print(dic[(1,2)])
```

```
(base) jianghao@club02:/data3/lhw/aihomework/3$ python3 try.py
a
(base) jianghao@club02:/data3/lhw/aihomework/3$
```

```
dic = dict()
dic[[1,2]] = "a"
print(dic[[1,2]])
```

```
(base) jianghao@club02:/data3/lhw/aihomework/3$ python3 try.py
Traceback (most recent call last):
  File "/data3/lhw/aihomework/3/try.py", line 2, in <module>
    dic[[1,2]] = "a"
TypeError: unhashable type: 'list'
```

quiz2:

可变序列有：列表、字典、集合(修改后在原地址)
不可变序列有：字符串、元组(修改后变了地址)

```python
print("The id of list was ",id(list))
list.append(2)
print("The id of list is now ",id(list))

dic = {1:"a",2:"b"}
print("the id of dictionary was ",id(dic))
dic[2] = "c"
print("The id of dictionary is now ",id(dic))

set = set([1,2,3])
print("The id of set was ",id(set))
set.add(4)
print("The id of set is now ",id(set))

turple = (1,2)
print("The id of turple was ",id(turple))
turple = (2,3)
print("The id of turple is now ",id(turple))

string = "hello"
print("The id of string was ",id(string))
string = "hi"
print("The id of string is now ",id(string))
```

```
(base) jianghao@club02:/data3/lhw/aihomework/3$ python3 try.py
The id of list was  140521318208320
The id of list is now  140521318208320
the id of dictionary was  140521318661504
The id of dictionary is now  140521318661504
The id of set was  140521316867648
The id of set is now  140521316867648
The id of turple was  140521317867456
The id of turple is now  140521317795072
The id of string was  140521317021232
The id of string is now  140521318036016
```