

## E2 15-Puzzle Problem (IDA\*)

---

17341189 姚森舰

2019 年 9 月 7 日

### 目录

<b>1 IDA* Algorithm</b>	<b>2</b>
1.1 Description . . . . .	2
1.2 Pseudocode . . . . .	2
<b>2 Tasks</b>	<b>2</b>
<b>3 Codes</b>	<b>4</b>
<b>4 Results</b>	<b>8</b>

# 1 IDA\* Algorithm

## 1.1 Description

Iterative deepening A\* (IDA\*) was first described by Richard Korf in 1985, which is a graph traversal and path search algorithm that can find the shortest path between a designated start node and any member of a set of goal nodes in a weighted graph.

It is a variant of **iterative deepening depth-first search** that borrows the idea to use a heuristic function to evaluate the remaining cost to get to the goal from the **A\* search algorithm**.

Since it is a depth-first search algorithm, its memory usage is lower than in A\*, but unlike ordinary iterative deepening search, it concentrates on exploring the most promising nodes and thus does not go to the same depth everywhere in the search tree.

**Iterative-deepening-A\* works as follows:** at each iteration, perform a depth-first search, cutting off a branch when its total cost  $f(n) = g(n) + h(n)$  exceeds a given threshold. This threshold starts at the estimate of the cost at the initial state, and increases for each iteration of the algorithm. At each iteration, the threshold used for the next iteration is the minimum cost of all values that exceeded the current threshold.

## 1.2 Pseudocode

## 2 Tasks

- Please solve 15-Puzzle problem by using IDA\* (Python or C++). You can use one of the two commonly used heuristic functions:  $h1$  = the number of misplaced tiles.  $h2$  = the sum of the distances of the tiles from their goal positions.
- Here are 4 test cases for you to verify your algorithm correctness. You can also play this game (15puzzle.zip) for more information.

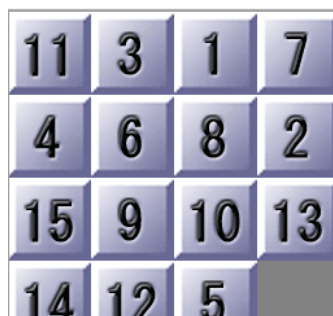
```

path          current search path (acts like a stack)
node          current node (last node in current path)
g            the cost to reach current node
f            estimated cost of the cheapest path (root..node..goal)
h(node)      estimated cost of the cheapest path (node..goal)
cost(node, succ) step cost function
is_goal(node) goal test
successors(node) node expanding function, expand nodes ordered by g + h(node)
ida_star(root) return either NOT_FOUND or a pair with the best path and its cost

procedure ida_star(root)
  bound := h(root)
  path := [root]
  loop
    t := search(path, 0, bound)
    if t = FOUND then return (path, bound)
    if t = ∞ then return NOT_FOUND
    bound := t
  end loop
end procedure

function search(path, g, bound)
  node := path.last
  f := g + h(node)
  if f > bound then return f
  if is_goal(node) then return FOUND
  min := ∞
  for succ in successors(node) do
    if succ not in path then
      path.push(succ)
      t := search(path, g + cost(node, succ), bound)
      if t = FOUND then return FOUND
      if t < min then min := t
      path.pop()
    end if
  end for
  return min
end function

```



TextOut of Result

```

11 3 1 7
4 6 8 2
15 9 10 13
14 12 5 0
LowerBound 36 moves
A optimal solution 56 moves
Used time 3 sec
13 10 8 6 9 12 5 13 10 8
12 15 14 5 13 12 15 14 5 13
14 9 4 11 3 1 6 4 11 3
1 6 4 2 8 10 12 15 10 8
7 4 2 11 3 5 9 10 11 3
6 2 3 7 8 12

```



TextOut of Result

```

0 5 15 14
7 9 6 13
1 2 12 10
8 11 4 3
LowerBound 44 moves
A optimal solution 62 moves
Used time 4 sec
7 9 2 1 9 2 5 7 2 5
1 11 8 9 5 1 6 12 10 3
4 8 11 10 12 13 3 4 8 12
13 15 14 3 4 8 12 13 15 14
7 2 1 5 10 11 13 15 14 7
3 4 8 12 15 14 11 10 9 13
14 15

```



TextOut of Result

```

14 10 6 0
4 9 1 8
2 3 5 11
12 13 7 15
LowerBound 37 moves
A optimal solution 49 moves
Used time 0 sec
8 10 9 4 14 9 4 1 10 4
1 3 2 14 9 1 3 2 5 11
8 6 4 3 2 5 13 12 14 13
12 7 11 12 7 14 13 9 5 10
6 8 12 7 10 6 7 11 15

```



TextOut of Result

```

6 10 3 15
14 8 7 11
5 1 0 2
13 12 9 4
LowerBound 32 moves
A optimal solution 48 moves
Used time 0 sec
9 12 13 5 1 9 7 11 2 4
12 13 9 7 11 2 15 3 2 15
4 11 15 8 14 1 5 9 13 15
7 14 10 6 1 5 9 13 14 10
6 2 3 4 8 7 11 12

```

- Please send E02\_YourNumber.pdf to ai\_201901@foxmail.com, you can certainly use E02\_15puzzle.tex as the L<sup>A</sup>T<sub>E</sub>X template.

### 3 Codes

```

1 # -*- coding: utf-8 -*-
  import sys
3 import copy

5
def h(node):
7     """启发函数，估计到目标状态的距离

9     :param node: 4x4二维列表，表示一种状态
    :return: 与目标状态的距离，采用曼哈顿距离
11     """

    cost = 0
13    # 遍历每个格子，计算与目标状态的距离
    for i in range(0, 16):
15        cur_row = int(i / 4)
        cur_col = int(i % 4)
17        tile = node[cur_row][cur_col]      # 格子里面的值，[0,15]
        # 根据格子里面的值计算在目标状态的坐标
19        if tile == 0:
            goal_row = 3
21            goal_col = 3
        else:
23            goal_row = int((tile - 1) / 4)
            goal_col = int((tile - 1) % 4)
25        # 计算曼哈顿距离
        dis_x = abs(cur_row - goal_row)
27        dis_y = abs(cur_col - goal_col)
        cost += dis_x
29        cost += dis_y
        # print(tile, goal_row, goal_col, cost)
31    return cost
# test = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 0, 15]]
33 # print(h(test))

35
def is_goal(node):
37     """判断是否为目标状态

39     :param node: 4x4二维列表，表示一种状态
    :return: 达到目标状态返回True，否则返回False
41     """

```

```

    for i in range(4):
        for j in range(4):
            if node[i][j] != goal[i][j]:
                return False
        return True
# test = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 0]]
# print(is_goal(test))

def successors(node):
    """计算此状态可能的下一状态

    :param node: 4x4二维列表, 表示一种状态
    :return: 返回可能达到的状态的列表, 三维列表
    """
    row, col = 0, 0
    # 找到空白块的位置
    for i in range(4):
        for j in range(4):
            if node[i][j] == 0:
                row, col = i, j

    # 上右下左
    moves = [[-1, 0], [0, 1], [1, 0], [0, -1]]
    succs = []
    for move in moves:
        s_row = row + move[0]
        s_col = col + move[1]
        if 0 <= s_row < 4 and 0 <= s_col < 4:
            temp_node = copy.deepcopy(node)
            temp_node[row][col], temp_node[s_row][s_col] = temp_node[s_row][s_col],
                temp_node[row][col]
            succs.append(temp_node)

    return sorted(succs, key=lambda x: h(x))
# test = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 0]]
# test = [[1, 2, 3, 4], [5, 0, 7, 8], [9, 10, 11, 12], [13, 14, 15, 6]]
# print(successors(test))

def search(path, g, bound):
    """向下探索路径

    :param path: 已走过的路径
    :param g: 走过的路径
    :param bound: 总路线长度估计值的下限
    :return: 如果到达目标状态就返回路径和花费, 没有找到则继续探索其后继状态
    """

```

```

node = path[-1]
89 f = g + h(node)
    if f > bound:
91         return f
    if is_goal(node):
93         return (path, f)
# 初始化为int的最大值
95 min_cost = sys.maxsize
for succ in successors(node):
97     if succ not in path:
        path.append(succ)
99         t = search(path, g+1, bound)
        if type(t) == tuple:
101             return t
        if t < min_cost:
103             min_cost = t
        del(path[-1])
105 return min_cost

107
def ida_star(root):
109     """使用IDA*算法解决15puzzle问题

111     :param root: 初始状态, 4x4二维列表
112     :return: 达到目标状态的每一步路径
113     """
    bound = h(root)
115     path = [root]

117     while True:
        t = search(path, 0, bound)
119         if type(t) == tuple:
            return t
        if t > 60:
121             return ([], bound)
        bound = t
123

125
def is_possible(root):
127     """用逆序对判断是否有解

129     :param root: 初始状态, 二维列表
130     :return: 如果有解返回True, 否则返回False
131     """
    temp = sum(root, [])    # 转化为一维列表, 方便判断
133     n = 0
    # print(temp)

```

```

135     for i in range(len(temp)):
136         if temp[i] == 0:
137             cur_row = int(i / 4)
138             dis_x = abs(cur_row - 3)
139             n += dis_x
140             continue
141         for j in range(i):
142             if temp[j] > temp[i]:
143                 n += 1
144 # 逆序数 + 空白格到目标状态的行数为偶数则有解
145 if n % 2 == 0:
146     return True
147 else:
148     return False
149
151 if __name__ == "__main__":
152     # 目标状态
153     goal = []
154     tile = 1
155     for i in range(4):
156         temp = []
157         for j in range(4):
158             temp.append(tile)
159             tile += 1
160         goal.append(temp)
161     # print(temp)
162     goal[3][3] = 0
163     # print(goal)
164     # root = [[5, 1, 3, 4], [2, 7, 8, 12], [9, 6, 11, 15], [0, 13, 10, 14]] # 15
165     # root = [[4, 11, 10, 1], [13, 14, 8, 2], [7, 5, 12, 6], [9, 3, 15, 0]] # 54
166     # root = [[1, 2, 3, 4], [12, 9, 14, 11], [7, 8, 6, 15], [0, 5, 13, 10]] # 31
167     # root = [[1, 3, 2, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 0]] # 无解
168     # root = [[6, 10, 3, 15], [14, 8, 0, 11], [1, 9, 7, 2], [5, 13, 12, 4]] # 41
169     # root = [[6, 10, 3, 15], [14, 8, 7, 11], [5, 1, 0, 2], [13, 12, 9, 4]] # 48
170     # root = [[14, 10, 6, 0], [4, 9, 1, 8], [2, 3, 5, 11], [12, 13, 7, 15]] # 49
171
172     for row in root:
173         print(row)
174
175     if is_possible(root):
176         (path, bound) = ida_star(root)
177         move = []
178         for node in path:
179             for i in range(4):
180                 for j in range(4):
181                     if node[i][j] == 0:

```

```

183         move.append([i, j])
184     del (move[0])
185     print('The solution need: ', len(move), 'moves.')
186     print('The path is:')
187     for i in range(len(move)):
188         x = move[i][0]
189         y = move[i][1]
190         print(path[i][x][y], end=' ')
191 else:
    print("Impossible!!")

```

## 4 Results

下面是一些测试样例：

```

[5, 1, 3, 4]
[2, 7, 8, 12]
[9, 6, 11, 15]
[0, 13, 10, 14]
The solution need: 15 moves.
The path is:
13 10 14 15 12 8 7 2 5 1 2 6 10 14 15

[1, 2, 3, 4]
[12, 9, 14, 11]
[7, 8, 6, 15]
[0, 5, 13, 10]
The solution need: 31 moves.
The path is:
5 13 6 8 9 12 7 5 13 6 10 15 8 14 12 7 5 9 6 10 14 12 11 8 12 11 7 6 10 14 15

[1, 3, 2, 4]
[5, 6, 7, 8]
[9, 10, 11, 12]
[13, 14, 15, 0]
Impossible!!

[6, 10, 3, 15]
[14, 8, 0, 11]
[1, 9, 7, 2]
[5, 13, 12, 4]
The solution need: 41 moves.
The path is:
11 2 4 12 13 9 7 11 2 15 3 2 15 4 11 15 8 14 1 5 9 13 15 7 14 10 6 1 5 9 13 14 10 6 2 3 4 8 7 11 12

[6, 10, 3, 15]
[14, 8, 7, 11]
[5, 1, 0, 2]
[13, 12, 9, 4]
The solution need: 50 moves.
The path is:
2 11 7 2 9 4 11 7 15 3 2 8 1 9 4 12 13 5 14 1 9 14 5 13 14 9 10 6 1 5 9 10 6 2 8 4 7 15 4 8 3 4 8 7 15 11 12 15 11 12

[14, 10, 6, 0]
[4, 9, 1, 8]
[2, 3, 5, 11]
[12, 13, 7, 15]
The solution need: 49 moves.
The path is:
6 10 9 4 14 9 4 1 10 4 1 3 2 14 9 1 3 2 5 11 8 6 4 3 2 5 13 12 14 13 12 7 11 12 7 14 13 9 5 10 6 8 12 7 10 6 7 11 15

```

本次实验的主要思路在文档中已经给出，其他要考虑的问题是如何存储，h 函数的选择，以及如何判断问题有解。我选择了曼哈顿距离作为 h 函数，但是这样做的运行速度还是有点慢，40 步以下




的都很快，当需要的步数多于 45 时，就需要较多的时间，测试了一个 58 步的样例，几个小时都没能出结果，就放弃了。

另外，对于是否有解的问题，查找资料发现：解决  $N \times N-1$  数码问题时，当  $N$  为奇数时，初始状态与指定状态逆序数奇偶性相同即有解； $N$  为偶数时，先计算出从初始状态到指定状态，空位要移动的行数  $m$ ，如果初始状态的逆序数加上  $m$  与指定状态的逆序数奇偶性相同，则有解。在本次实验中，因为  $N=4$ ，目标状态的逆序数为 0，即为偶数，简化一下就是，初始状态的逆序数加上空位要移动的行数  $m$  之和为偶数时有解。

另外一个要注意的问题还是关于列表的拷贝，二维数组的拷贝最好是用深拷贝。

其实还有一个问题没有解决。对于下面这个例子，也就是上面的第 5 个测试样例：



TextOut of Result															
6	10	3	15												
14	8	7	11												
5	1	0	2												
13	12	9	4												
LowerBound 32 moves															
A optimal solution 48 moves															
Used time 0 sec															
9	12	13	5	1	9	7	11	2	4						
12	13	9	7	11	2	15	3	2	15						
4	11	15	8	14	1	5	9	13	15						
7	14	10	6	1	5	9	13	14	10						
6	2	3	4	8	7	11	12								

最优的路径应该是 48 步，但是我的程序跑出来却是 50 步，问题就出在第一次移动方块的选择上。初始状态下，移动 2 和 9 的  $h$  函数值是一样的，而我在实现时，检查空格的后继的顺序是上右下左，也就是会选择 2，而最优选择应该是 9。思考后发现，即使我修改了检查的顺序，总有某个时刻可能会出错，因为使用曼哈顿距离作为  $h$  函数的话，有可能出现  $h$  函数值相同的情况，这时候有可能选的不是最优路径。所以使用曼哈顿距离作为  $h$  函数有两个问题，一是速度慢，二是找到的路径不一定是最优路径，但是这种情况比较少。我也没有想到更好的  $h$  函数，还需要再思考思考。