

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

---

18308045 Zhengyang Gu

September 11, 2020

### Contents

<b>1</b>	<b>IDA* Algorithm</b>	<b>2</b>
1.1	Description . . . . .	2
1.2	Pseudocode . . . . .	2
<b>2</b>	<b>Tasks</b>	<b>3</b>
<b>3</b>	<b>Codes</b>	<b>3</b>
<b>4</b>	<b>Results</b>	<b>6</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

```
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
```

## 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.exe) for more information.

	<p>TextOut of Result</p> <pre> 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 </pre>		<p>TextOut of Result</p> <pre> 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 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 </pre>
	<p>TextOut of Result</p> <pre> 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 </pre>		<p>TextOut of Result</p> <pre> 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 </pre>

- Please send E02\_YourNumber.pdf to ai\_2020@foxmail.com, you can certainly use E02\_15puzzle.tex as the L<sup>A</sup>T<sub>E</sub>X template.

## 3 Codes

```

1 import time
2 path = list()
3 # 通过值查目标位置
4 goal_reverse = ((3, 3), (0, 0), (0, 1), (0, 2),
5                 (0, 3), (1, 0), (1, 1), (1, 2),
6                 (1, 3), (2, 0), (2, 1), (2, 2),
7                 (2, 3), (3, 0), (3, 1), (3, 2))
8 row = 4
9 col = 4
10 range_row = range(row)
11 range_col = range(col)
12 puzzle = list()
13 # 曼哈顿距离, 只调用一次, 后续动态修改f值
14 def h():
15     ans = 0
16     for i in range_row:
17         for j in range_col:

```

```

18         if puzzle[i][j] is 0:
19             continue
20             goal_pos = goal_reverse[puzzle[i][j]]
21             ans += abs(i - goal_pos[0]) + abs(j - goal_pos[1])
22     return ans
23 cost = 1
24 # 通过位置查目标值
25 goal = ((1, 2, 3, 4),
26          (5, 6, 7, 8),
27          (9, 10, 11, 12),
28          (13, 14, 15, 0))
29 def is_goal():
30     for i in range_row:
31         for j in range_col:
32             if puzzle[i][j] is 0:
33                 continue
34             if puzzle[i][j] is not goal[i][j]:
35                 return False
36     return True
37 # 记录0所在的位置, 方便更新
38 zero_pos
39 directions = ((-1, 0), (1, 0), (0, -1), (0, 1))
40 def successors():
41     global zero_pos
42     ans = list()
43     for direction in directions:
44         succ = (zero_pos,
45                 (zero_pos[0] + direction[0],
46                  zero_pos[1] + direction[1]))
47         if 0 <= succ[1][0] \
48             and succ[1][0] < row \
49             and 0 <= succ[1][1] \
50             and succ[1][1] < col:
51             ans.append(succ)
52     return ans
53 # 当前图的hash值, 可以放入set中用于路径检测
54 # list类型的puzzle无法放入set中
55 cur_hash = 0
56 # 初始化cur_hash, 后续只对变化元素修改cur_hash
57 def hash():
58     global cur_hash
59     base = 1
60     for i in range_row:
61         for j in range_col:
62             cur_hash += base * puzzle[i][j]
63             base *= 16
64 inf = float("inf")
65 vis = set()
66 def ida_star():

```

```

67     global zero_pos
68     stt = h()
69     bound = stt
70     vis.add(hash())
71     for i in range_row:
72         for j in range_col:
73             if puzzle[i][j] is 0:
74                 zero_pos = (i, j)
75                 break
76         else:
77             continue
78     break
79     while 1:
80         t = search(path, stt, bound)
81         if t is True:
82             return (path, bound)
83         if t == inf:
84             return False
85         bound = t
86 def search(path, f, bound):
87     global zero_pos
88     global cur_hash
89     if f > bound:
90         return f
91     if is_goal():
92         return True
93     min = inf
94     for succ in successors():
95         succ_value = puzzle[succ[1][0]][succ[1][1]]
96         last_hash = cur_hash
97         cur_hash += succ_value
98         * (16 ** (succ[0][0] * col + succ[0][1]))
99         - 16 ** (succ[1][0] * col + succ[1][1]))
100        puzzle[succ[0][0]][succ[0][1]] = succ_value
101        puzzle[succ[1][0]][succ[1][1]] = 0
102        zero_pos = succ[1]
103        if cur_hash not in vis:
104            path.append(succ)
105            vis.add(cur_hash)
106            goal_pos = goal_reverse[succ_value]
107            t = search(path,
108                        f + abs(succ[0][0] - goal_pos[0]) +
109                        abs(succ[0][1] - goal_pos[1])
110                        - (abs(succ[1][0] - goal_pos[0]) +
111                          abs(succ[1][1] - goal_pos[1])) + cost,
112                        bound)
113            if t is True:
114                return True
115            if t < min:

```

```

116         min = t
117         vis.remove(cur_hash)
118         path.pop()
119         zero_pos = succ[0]
120         puzzle[succ[1][0]][succ[1][1]] = succ_value
121         puzzle[succ[0][0]][succ[0][1]] = 0
122         cur_hash = last_hash
123     return min
124 def read_15puzzle(file_name):
125     with open(file_name) as f:
126         for i in range_row:
127             line = f.readline()
128             if not line:
129                 break
130             puzzle.append([int(i) for i in line.split()])
131 read_15puzzle("test4.txt")
132 for i in puzzle:
133     for j in i:
134         print(j, end = '\t')
135     print()
136 start = time.perf_counter()
137 ans = ida_star()
138 end = time.perf_counter()
139 print("Time_used:", end - start)
140 print(ans)

```

## 4 Results

```

14      10      6      0
4       9      1      8
2       3      5     11
12      13      7     15
Time used: 81.6847216
([((0, 3), (0, 2)), ((0, 2), (0, 1)), ((0, 1), (1, 1)), ((1, 1), (1, 0)), ((1, 0), (0, 0)), ((0, 0),
(0, 1)), ((0, 1), (1, 1)), ((1, 1), (1, 2)), ((1, 2), (0, 2)), ((0, 2), (0, 1)), ((0, 1), (1, 1)),
((1, 1), (2, 1)), ((2, 1), (2, 0)), ((2, 0), (1, 0)), ((1, 0), (0, 0)), ((0, 0), (0, 1)), ((0, 1), (1,
1)), ((1, 1), (2, 1)), ((2, 1), (3, 1)), ((3, 1), (3, 0)), ((3, 0), (2, 0)), ((2, 0), (2, 1)), ((2,
1), (2, 2)), ((2, 2), (2, 3)), ((2, 3), (1, 3)), ((1, 3), (0, 3)), ((0, 3), (0, 2)), ((0, 2), (0,
1)), ((0, 1), (1, 1)), ((1, 1), (2, 1)), ((2, 1), (3, 1)), ((3, 1), (3, 2)), ((3, 2), (2, 2)), ((2,
2), (2, 1)), ((2, 1), (3, 1)), ((3, 1), (3, 0)), ((3, 0), (2, 0)), ((2, 0), (1, 0)), ((1, 0), (1, 1)),
((1, 1), (1, 2)), ((1, 2), (1, 3)), ((1, 3), (2, 3)), ((2, 3), (2, 2)), ((2, 2), (2, 1)), ((2, 1),
(1, 1)), ((1, 1), (1, 2)), ((1, 2), (2, 2)), ((2, 2), (3, 2)), ((3, 2), (3, 3))], 49)

```

Figure 1: result2

```

6       10      3     15
14      8      7     11
5       1      0      2
13      12      9      4
Time used: 223.58929429999998
([((2, 2), (3, 2)), ((3, 2), (3, 1)), ((3, 1), (3, 0)), ((3, 0), (2, 0)), ((2, 0), (2, 1)), ((2, 1),
(2, 2)), ((2, 2), (1, 2)), ((1, 2), (1, 3)), ((1, 3), (2, 3)), ((2, 3), (3, 3)), ((3, 3), (3, 2)),
((3, 2), (3, 1)), ((3, 1), (2, 1)), ((2, 1), (2, 2)), ((2, 2), (1, 2)), ((1, 2), (1, 3)), ((1, 3), (0,
3)), ((0, 3), (0, 2)), ((0, 2), (1, 2)), ((1, 2), (1, 3)), ((1, 3), (2, 3)), ((2, 3), (2, 2)), ((2,
2), (1, 2)), ((1, 2), (1, 1)), ((1, 1), (1, 0)), ((1, 0), (2, 0)), ((2, 0), (3, 0)), ((3, 0), (3,
1)), ((3, 1), (3, 2)), ((3, 2), (2, 2)), ((2, 2), (2, 1)), ((2, 1), (1, 1)), ((1, 1), (0, 1)), ((0,
1), (0, 0)), ((0, 0), (1, 0)), ((1, 0), (2, 0)), ((2, 0), (3, 0)), ((3, 0), (3, 1)), ((3, 1), (2, 1)),
((2, 1), (1, 1)), ((1, 1), (0, 1)), ((0, 1), (0, 2)), ((0, 2), (0, 3)), ((0, 3), (1, 3)), ((1, 3),
(1, 2)), ((1, 2), (2, 2)), ((2, 2), (2, 3)), ((2, 3), (3, 3))], 48)

```

Figure 2: result4