School of Data and Computer Science, Sun Yat-sen University

E2 15-Puzzle Problem (IDA*)

17341137 Zhenpeng Song

September 8, 2019

Contents

1 IDA* Algorithm **2**
1.1 Description . . . 2
1.2 Pseudocode . . . 2
2 Tasks **3**
3 Codes **4**
4 Results **10**

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.

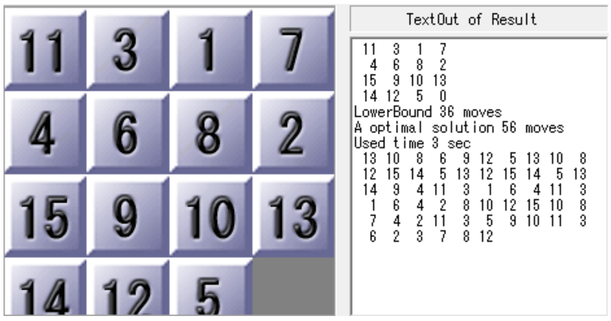
Iterative-deepening-A* works as follows: at each iteration, perform a depth-first search, cutting off a branch when its total cost $f(n) = g(n) + h(n)$ exceeds a given threshold. This threshold starts at the estimate of the cost at the initial state, and increases for each iteration of the algorithm.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

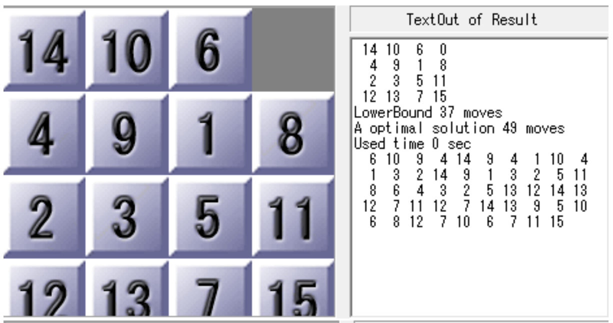
```
path              current search path (acts like a stack)
node              current node (last node in current path)
g                 the cost to reach current node
f                 estimated cost of the cheapest path (root..node..goal)
h(node)           estimated cost of the cheapest path (node..goal)
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

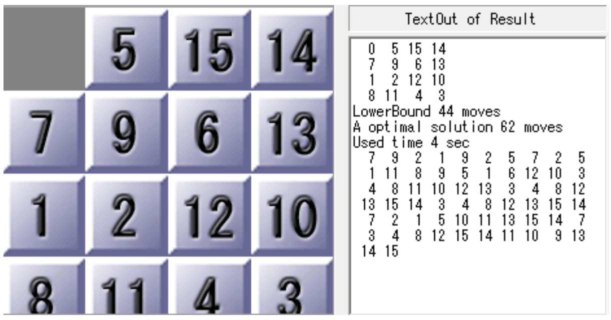
- Please solve 15-Puzzle problem by using IDA* (Python or C++). You can use one of the two commonly used heuristic functions: h1 = the number of misplaced tiles. h2 = the sum of the distances of the tiles from their goal positions.
- Here are 4 test cases for you to verify your algorithm correctness.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



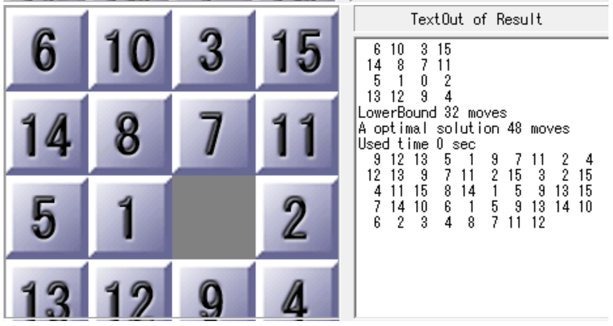
```
 11  3  1  7
  4  6  8  2
 15  9 10 13
 14 12  5  0
LowerBound 36 moves
A optimal solution 56 moves
Used time 3 sec
 13 10  8  6  9 12  5 13 10  8
 12 15 14  5 13 12 15 14  5 13
 14  9  4 11  3  1  6  4 11  3
  1  6  4  2  8 10 12 15 10  8
  7  4  2 11  3  5  9 10 11  3
  6  2  3  7  8 12
```



```
 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
```



```
  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
```



```
  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 LaTeXtemplate.

3 Codes

- Python version

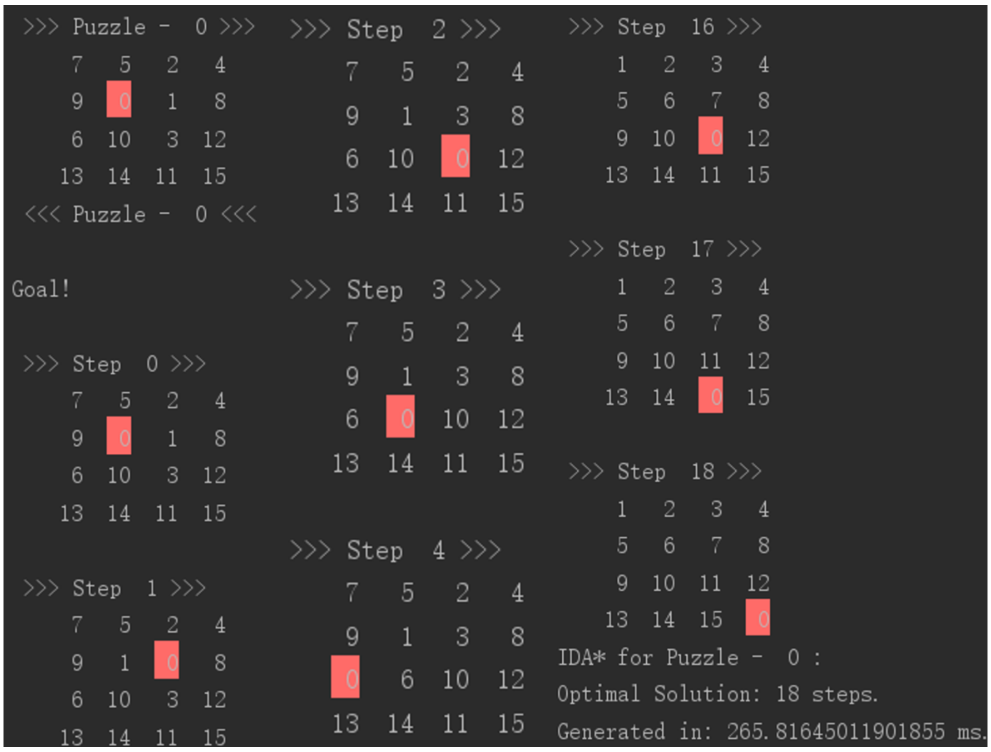
```python
# AI - exp 2 - python - 15Puzzle
# ID: 17341137

import sys
import time


# Estimated cost of the cheapest path (node-to-goal)
def h(node):
    val = 0
    for i in range(len(node)):
        for j in range(len(node[0])):
            val += abs(i - (node[i][j]-1)/4) + abs(j - (node[i][j]-1) % 4)
    return int(val)


# Step cost func.
def cost(node, succ):
    return 1


# Goal test
def is_goal(node):
    t = 1
    for i in node:
        for j in i:
            if j != t:
                break
            else:
                t += 1
    return t == 16


# node expanding func.
def successors(node):
    x, y = 0, 0
    succ = []
    for i in range(len(node)):
        for j in range(len(node[0])):
            if node[i][j] == 0:
                x, y = i, j
    dirs = [[-1, 0], [1, 0], [0, -1], [0, 1]]
    for i, j in dirs:
        tmp = [[r for r in c] for c in node]
        _x, _y = x + i, y + j
        if 0 <= _x < len(node) and 0 <= _y < len(node[0]):
            tmp[x][y] = tmp[_x][_y]
            tmp[_x][_y] = 0
            succ.append(tmp)
    return sorted(succ, key=lambda t: h(t))


# return NOT_FOUND or (best_path, cost)
def ida_star(root):
    bound = h(root)
    path = [root]
    while 1:
        t = search(path, 0, bound)
        if t == "FOUND":
            return path, bound
        if t == sys.maxsize:
            return [], sys.maxsize
        bound = t


# DFS
def search(path, g, bound):
    node = path[-1]
    f = g + h(node)
    if f > bound:
        return f
    elif is_goal(node):
        print("\nGoal!")
        return "FOUND"
    min_dist = sys.maxsize
    for succ in successors(node):
        if succ in path:
            continue
        path.append(succ)
        t = search(path, g + cost(node, succ), bound)
        if t == "FOUND":
            return "FOUND"
        if t < min_dist:
            min_dist = t
        path.pop()
    return min_dist


def load():
    Map = [
        [[7, 5, 2, 4],
         [9, 0, 1, 8],
         [6, 10, 3, 12],
         [13, 14, 11, 15]],
            [[2, 10, 3, 4],
             [1, 9, 6, 7],
             [13, 5, 11, 8],
             [0, 14, 15, 12]],
            [[3, 4, 1, 0],
             [5, 2, 10, 8],
             [9, 7, 6, 11],
             [13, 14, 15, 12]]
        ]
        return Map


    def generate(Map):
        for i in Map:
            for j in i:
                if j == 0:
                    print("\t\033[41m 0\033[0m", end=" ")
                else:
                    print("\t%2d" % j, end=" ")
            print("")


    if __name__ == "__main__":
        # Generate the Map
        MAP = load()
        for i in range(len(MAP)):
            print("\n\n >>> Puzzle - ", i, ">>>")
            generate(MAP[i])
            print(" <<< Puzzle - ", i, "<<<")
            tik = time.time()
            (path, bound) = ida_star(MAP[i])
            tok = time.time()
            for node in range(len(path)):
                print("\n >>> Step ", node, ">>>")
                generate(path[node])
            print("IDA* for Puzzle - ", i, ":")
            print("Optimal Solution:", len(path) - 1, "steps.")
            print("Generated in:", 1000*(tok - tik), "ms.")
```

- Cpp version

```
    #include <iostream>
#include <algorithm>
#include <cstdlib>
#include <vector>
using namespace std;

const int SIZE = 4;
int Map[SIZE][SIZE] = { {9, 1, 12, 15}, {4, 3, 11, 0}, {14, 2, 5, 8}, {10,
// int Map[SIZE][SIZE] = { {7, 5, 2, 4}, {9, 0, 1, 8}, {6, 10, 3, 12},
{13, 14, 11, 15}};
const int fact[10] = {1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880};

int h(int **node) {
    int val = 0;
    for(int i = 0;i<SIZE;i++) {
        for(int j = 0;j<SIZE;j++) {
            val += abs(i - (node[i][j]-1)/SIZE) + abs(j - (node[i][j]-1)%SIZE)
        }
    }
    return (int)val;
}


void generate(int **m) {
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            cout << m[i][j] << " ";
        }
        cout << endl;
    }
}

bool is_goal(int **node) {
    int t = 1;
    for(int i = 0;i<SIZE;i++) {
        for(int j = 0;j<SIZE;j++) {
            if(node[i][j]!=t) break;
            else t++;
        }
    }
    return (t == 16);
}

bool cmp(int **a, int **b) {
    return h(a) < h(b);
}

vector<int **> successor(int **node) {
    int x = 0, y = 0;
    vector<int **> succ;
    for(int i = 0;i<SIZE;i++) {
        for(int j = 0;j<SIZE;j++) {
            if(node[i][j]==0) {
                x = i;
                y = j;
            }
        }
    }
    int dirs[4][2] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
    for(int i = 0;i<4;i++) {
        int **tmp = new int *[SIZE];
        for(int j = 0;j<SIZE;j++) {
            tmp[j] = new int[SIZE];
        }
        for(int m = 0;m<SIZE;m++) {
            for(int n = 0;n<SIZE;n++) {
                tmp[m][n] = node[m][n];
            }
        }
        int _x = x + dirs[i][0];
        int _y = y + dirs[i][1];
        if(_x>=0&&_x<SIZE&&_y>=0&&_y<SIZE) {
            tmp[x][y] = tmp[_x][_y];
            tmp[_x][_y] = 0;
            succ.push_back(tmp);
        }
    }
    sort(succ.begin(), succ.end(), cmp);
    // for(int i = 0;i<succ.size();i++) {
    //     cout<<h(succ[i])<<endl;
    //     generate(succ[i]);
    //     cout<<endl;
    // }
    return succ;
}

int contor(int ** permutation, int n) {
    int num = 0;
    for (int i = 0; i < n; ++i) {
        int cnt = 0; //    i          i
        for (int j = i + 1; j < n; ++j)
            if (permutation[i] > permutation[j]) ++cnt;
        num += cnt * fact[n - i - 1];
    }
    return num + 1;
}

bool in_path(vector<int **> &path, int **node) {
    bool check;
    for(int i = 0;i<path.size();i++) {
        check = true;
        for(int j = 0;j<SIZE;j++) {
            for(int k = 0;k<SIZE;k++) {
                if(node[j][k]!=path[i][j][k]) {
                    check = false;
                    break;
                }
            }
            if(!check) break;
        }
        if(check) break;
    }
    return check;
}

int search(vector<int **> &path, int g, int bound) {
    int **node = path.back();
    // generate(node);
    int f = g + h(node);
    if(f > bound) return f;
    else if(is_goal(node)) {
        generate(node);
        cout<<"Goal!"<<endl;
        return -1;
    }
    int min_dist = INT_MAX;
    vector<int **> successors = successor(node);
    for(int i = 0;i<successors.size();i++) {
        if(in_path(path, successors[i])) continue;
        path.push_back(successors[i]);
        int t = search(path, g + 1, bound);
        if(t==-1) return -1;
        if(t<min_dist) min_dist = t;
        path.pop_back();
    }
    return min_dist;
}

vector<int **> ida_star(int **root) {
    int bound = h(root);
    vector<int **> path;
    path.push_back(root);
    int t = 0;
    while(1) {
        t = search(path, 0, bound);
        cout<<"ida*: "<<t<<endl;
        if(t==-1) return path;
        if(t==INT_MAX) {
            path.clear();
            return path;
        }
        bound = t;
    }
}
```

```
int ** load() {
    int ** ptr = new int *[SIZE];
    for (int i = 0; i < SIZE; i++) {
        ptr[i] = new int[SIZE];
        ptr[i] = (Map[i]);
    }
    return ptr;
}

int main() {
    int **Map = load();
    cout<<"\n\n >>> Puzzle - "<< "1" << ">>>"<<endl;
    generate(Map);
    cout<<" <<< Puzzle - " << "1" << "<<<"<<endl;
    // tik = time.time()
    vector<int **> path = ida_star(Map);
    cout<<path.size()<<endl;
    // tok = time.time()
    for(int i = 0;i<path.size();i++) {
        cout << "\n >>> Step " << i << ">>>"<<endl;
        generate(path[i]);
    }

    cout << "IDA* for Puzzle - " << "1" << ":"<<endl;
    cout << "Optimal Solution:" << path.size() - 1 << "steps."<<endl;
    // cout << "Generated in:" << 1000*(tok - tik) << "ms.";
    system("pause");
    return 0;
}
```

4 Results

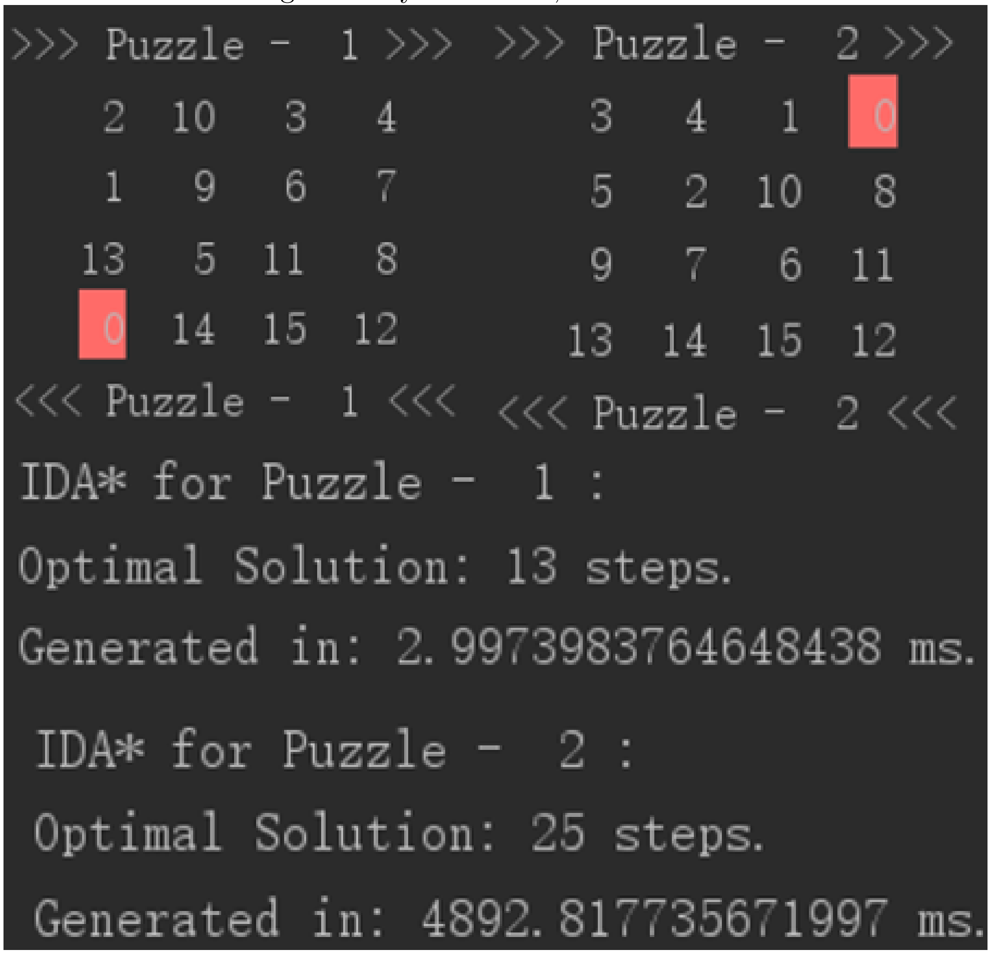


```
>>> Puzzle -  0 >>>
     7   5   2   4
     9   0   1   8
     6  10   3  12
    13  14  11  15
<<< Puzzle -  0 <<<

Goal!

>>> Step  0 >>>
     7   5   2   4
     9   0   1   8
     6  10   3  12
    13  14  11  15

>>> Step  1 >>>
     7   5   2   4
     9   1   0   8
     6  10   3  12
    13  14  11  15
```

```
>>> Step  2 >>>
     7   5   2   4
     9   1   3   8
     6  10   0  12
    13  14  11  15

>>> Step  3 >>>
     7   5   2   4
     9   1   3   8
     6   0  10  12
    13  14  11  15

>>> Step  4 >>>
     7   5   2   4
     9   1   3   8
     0   6  10  12
    13  14  11  15
```

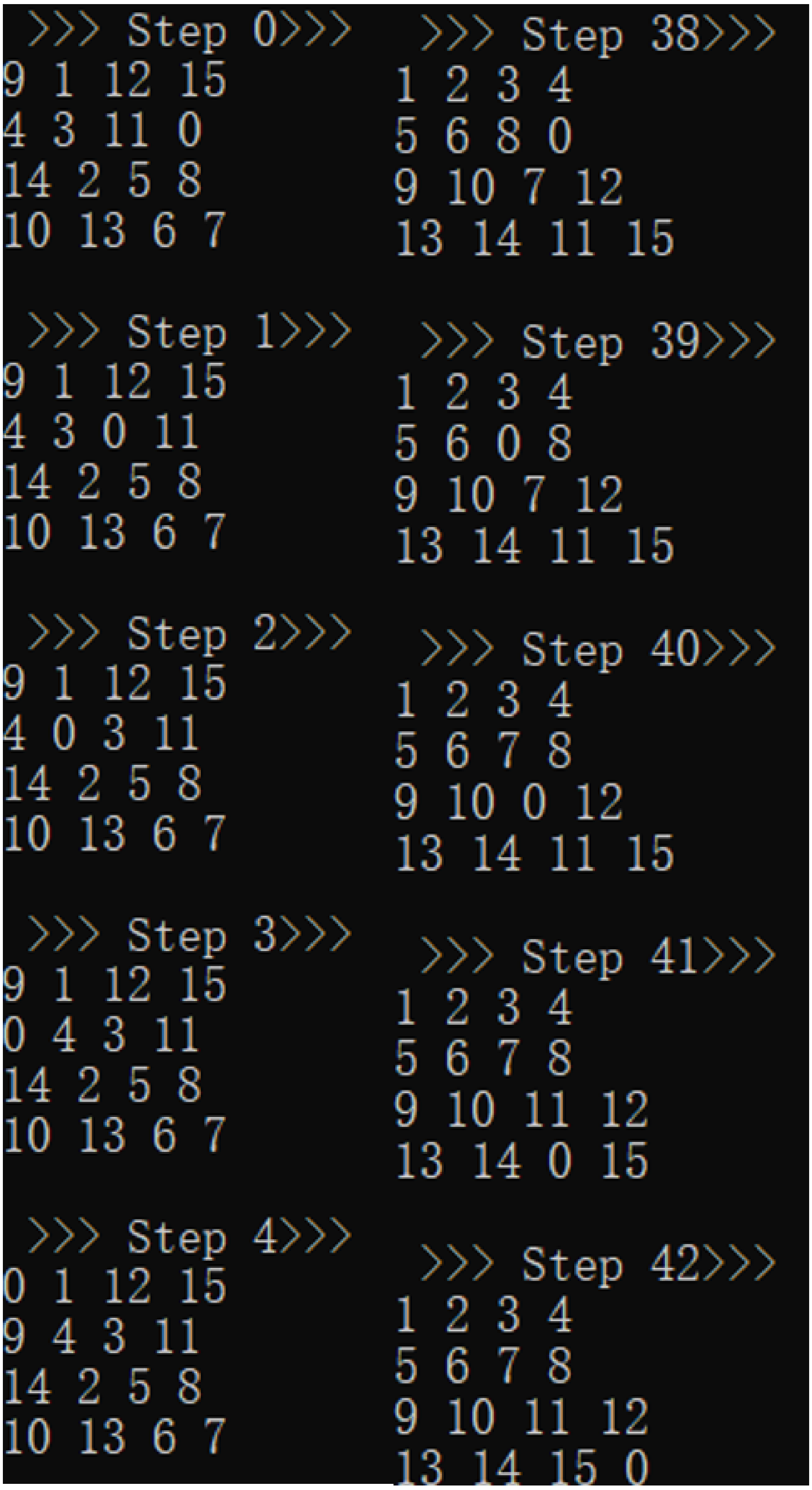
```
>>> Step  16 >>>
     1   2   3   4
     5   6   7   8
     9  10   0  12
    13  14  11  15

>>> Step  17 >>>
     1   2   3   4
     5   6   7   8
     9  10  11  12
    13  14   0  15

>>> Step  18 >>>
     1   2   3   4
     5   6   7   8
     9  10  11  12
    13  14  15   0
IDA* for Puzzle -  0 :
Optimal Solution: 18 steps.
Generated in: 265.81645011901855 ms.
```

Figure 1: Python - res1; red '0': track



```
>>> Puzzle -  1 >>>    >>> Puzzle -  2 >>>
     2  10   3   4          3   4   1   0
     1   9   6   7          5   2  10   8
    13   5  11   8          9   7   6  11
     0  14  15  12         13  14  15  12
<<< Puzzle -  1 <<<    <<< Puzzle -  2 <<<
IDA* for Puzzle -  1 :
Optimal Solution: 13 steps.
Generated in: 2.997398376464844 ms.

IDA* for Puzzle -  2 :
Optimal Solution: 25 steps.
Generated in: 4892.817735671997 ms.
```

Figure 2: Python - res2; red '0': track



```
 >>> Step 0>>>
9 1 12 15
4 3 11 0
14 2 5 8
10 13 6 7

 >>> Step 1>>>
9 1 12 15
4 3 0 11
14 2 5 8
10 13 6 7

 >>> Step 2>>>
9 1 12 15
4 0 3 11
14 2 5 8
10 13 6 7

 >>> Step 3>>>
9 1 12 15
0 4 3 11
14 2 5 8
10 13 6 7

 >>> Step 4>>>
0 1 12 15
9 4 3 11
14 2 5 8
10 13 6 7
```

```
 >>> Step 38>>>
1 2 3 4
5 6 8 0
9 10 7 12
13 14 11 15

 >>> Step 39>>>
1 2 3 4
5 6 0 8
9 10 7 12
13 14 11 15

 >>> Step 40>>>
1 2 3 4
5 6 7 8
9 10 0 12
13 14 11 15

 >>> Step 41>>>
1 2 3 4
5 6 7 8
9 10 11 12
13 14 0 15

 >>> Step 42>>>
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 0
```

Figure 3: Cpp - res3; 42steps; ’0’: track