E2 15-Puzzle Problem (IDA*)

学号	姓名	日期
183340215	张天祎	2020.9.13

1、IDA* Algorithm

> 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.

2、Task

- 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.

3、Codes

```
#include <iostream>
#include <fstream>
#include <string>
#define SIZE 4
using namespace std;

enum TURN
{
    L,
    R,
    U,
    D,
    E
};
int mat[SIZE][SIZE];
TURN sta[100];
```

```
int dx[4] = {0, 0, -1, 1};
int dy[4] = {-1, 1, 0, 0};
TURN wr[4] = {R, L, D, U};
bool flag;

void print()
{
    for (int i = 0; i < SIZE; i++)
    {
        for (int j = 0; j < SIZE; j++)
        {
            cout << mat[i][j] << ' ';
        }
        puts("");
    }
}

inline int point_dis(int x, int y)
{
    return abs((mat[x][y] - 1) / SIZE - x) + abs((mat[x][y] - 1) % SIZE - y);
}

int dfs(int g, int h, int depth, int x, int y, TURN T)
{
    if (!h)
    {
        flag = true;
        return 0;
    }
    if (g + h > depth)
        return g + h;
    int rst = INT_MAX;

    for (int i = 0; i < SIZE; i++)
    {
        if (T == wr[i])
            continue;
        int nx = x + dx[i];
        int ny = y + dy[i];
        if (nx >= 0 && ny >= 0 && nx < SIZE && ny < SIZE)
        {
            int hold = h;
            h -= point_dis(nx, ny);
            mat[x][y] = mat[nx][ny];
            mat[nx][ny] = 0;
            h += point_dis(x, y);
            int ret = dfs(g + 1, h, depth, nx, ny, TURN(i));
            if (flag)
            {
                sta[g] = TURN(i);
                return ret;
            }
            h = hold;
            mat[nx][ny] = mat[x][y];
            mat[x][y] = 0;
            rst = ret < rst ? ret : rst;
        }
    }
```

```
    return rst;
}

int main()
{
    ifstream inFile;
    inFile.open("data.txt", ifstream::in);
    int num, x, y;
    inFile >> num;
    for (int i = 0; i < num; i++)
    {
        int h = 0;
        flag = false;
        for (int i = 0; i < SIZE; i++)
        {
            for (int j = 0; j < SIZE; j++)
            {
                inFile >> mat[i][j];
                if (!mat[i][j])
                {
                    x = i;
                    y = j;
                }
                else
                {
                    h += point_dis(i, j);
                }
            }
        }
        print();
        int re = 0;
        for (int i = 0; i < SIZE * SIZE; i++)
        {
            for (int j = 0; j < i; j++)
            {
                if (!mat[i / SIZE][i % SIZE])
                    continue;
                if (mat[i / SIZE][i % SIZE] < mat[j / SIZE][j % SIZE])
                    re++;
            }
        }
        if ((re + SIZE - 1 - x) % 2 != 0)
        {
            cout << "No answer!" << endl;
            continue;
        }
        int depth = h;
        while (1)
        {
            int rst = dfs(0, h, depth, x, y, E);
            if (rst)
            {
                depth = rst;
            }
            else
            {
                cout << "Length:" << depth << endl;
                for (int i = 0; i < depth; i++)
```

```
                {
                    char curr;
                    switch (sta[i])
                    {
                    case 0:
                        curr = 'L';
                        break;
                    case 1:
                        curr = 'R';
                        break;
                    case 2:
                        curr = 'U';
                        break;
                    case 3:
                        curr = 'D';
                        break;
                    default:
                        curr = 'E';
                        break;
                    }
                    cout << curr;
                }
                puts("");
                break;
            }
        }
    }
    inFile.close();
    return 0;
}
```

4、Results

```
11 3 1 7
4 6 8 2
15 9 10 13
14 12 5 0
Length:56
LLURRDLURULLURDLLURDDRRULLDLURRDLLDRRULLDRUURULDRRULDRDD
14 10 6 0
4 9 1 8
2 3 5 11
12 13 7 15
Length:49
LLDLURDRULDDLUURDDRRUULLDDDLURDRULDLUURRRDLLURDDR
0 5 15 14
7 9 6 13
1 2 12 10
8 11 4 3
Length:62
DRDLURULDRDDLUURRDRDLLURURDDLUUURDDDLUUULLDDRDRUUURDDDLULLDRRR
6 10 3 15
14 8 7 11
5 1 0 2
13 12 9 4
Length:48
DLLURRURDDLLURURULDRDLULLDDRRULUULDDDRUUURRDLDRD
```

- L：Left R:Right D:Down U:Up

- 最后的结果和15puzzle.exe的长度结果是吻合的。
- 在4个示例矩阵分别运行时，花费的时间大概为30s左右，小于1s，60+s，1s左右。这和他们的路径长度是符合的，运行时间是以路径长度的指数次增长的。但在15puzzle.exe中似乎实例1和实例3的运行时间都小于10s，不知道它内部是如何实现的。
- 判断问题是否有解的办法是判断0所在的行和最后一行的差与除0元素外的逆序对之和是否为偶。