Assignment-2

Backtracking: 15-Puzzle

15-Puzzle

- Write a C/C++ program for solving the 15 puzzle using a backtracking algorithm.
- You start from an arbitrary arrangement of the tiles 1, 2, …, 15 in a 4 × 4 board.
- This leaves exactly one blank square.
- Task: To find a sequence of moves that restore the board to the following configuration, called the final configuration.

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

Example

1	2	3	4
5	6	8	→
9	10	7	12
13	14	11	15

1	2	3	4
5	6	↑	8
9	10	7	12
13	14	11	15

1	2	3	4
5	6	7	8
9	10	↑	12
13	14	11	15

1	2	3	4
5	6	7	8
9	10	11	12
13	14	←	15

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

Part 1

- From exactly half of the initial configurations, it is possible to reach the final configuration.
- These solvable initial configurations are characterized as follows.
- First, write the initial configuration in the row-major order, while neglecting the blank square.
- Let m denote the number of inversions in this array (that is, the number of pairs (i, j) such that i > j, but i appears before j in the array).
- Let n denote the width (or height) of the grid.
- Let r be the index of the row in the 2-d grid (indexing starts from 0 at the top), containing the blank square.
- The initial configuration is solvable if and only if one of the following is true.
    - If n is odd, then m is even.
    - If n is even, then m+ r is odd.
- Given an initial configuration of a 4 × 4 board, write a function that determines whether that configuration is solvable.

Example

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

- The initial configuration is written as:

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

m = 6 [Inversions are (8, 7), (9, 7), (10, 7), (12, 11), (13, 11) and (14, 11)]
n = 4 [Grid width (or height)]
r = 1. [Row index of the blank square] is r = 1.
Since n = 4 is even, we use the second case.
But m + r = 7 is odd, so the corresponding initial configuration is solvable.

Part 2

- Write a non-recursive function based upon backtracking in order to compute a sequence of moves that change the board from a given initial configuration to the final configuration (provided that the initial configuration is solvable).
- In order to avoid infinite loops in the search, you need to adopt two measures.
    1. No configuration is repeated on the search path.
    2. Possibilities of the movement of the blank tile are explored in a particular order.
- You may implement a depth-restricted version of the backtracking procedure, and gradually increase the depth until a solution is reached.
- It is known that you may require as many as 31 (or 80) moves for n = 3 (or n = 4).

Submission

- Last date: 18-AUG-2024 (till 11:59 P.M.) (Sunday)
- Programming language: C/C++
- Single File: 24CS06001_A2.c/.cpp or 24AI06001_A2.c/.cpp
- Subject Line: 24CS06001_A2 or 24AI06001_A2
- Email to: pds2016autumn@gmail.com