Sliding Puzzle

solved by	Senan
Platform	LeetCode
difficulty	Hard
# Serial	773
tags	BFS
language	C++
solved on	@25/11/2024
link	https://leetcode.com/problems/sliding-puzzle/description/
Completion	☑

Intuition

The problem involves solving a sliding puzzle, which can be treated as a search problem. The objective is to transform the given starting board configuration into the target configuration using the fewest possible moves. Each move swaps the empty tile (`0`) with one of its valid neighbors. Since the puzzle involves exploring all possible configurations, it is natural to use **Breadth-First Search (BFS)** because BFS explores all states at the same "depth" (or number of moves) before moving deeper. This ensures that the first time we encounter the target state, we do so in the minimum number of steps.

Approach

1. **Representation**:

   The 2x3 board is converted into a single string to simplify state representation. Each cell in the 2D grid corresponds to a position in the string, and swapping tiles is done by swapping characters in the string.

2. **Goal State**:

   The goal is defined as `"123450"`, representing the solved configuration.

3. **Movement Mapping**:

   For each position in the string, we predefine its valid moves (neighbors). For example:

   - Index 0 can move to indices 1 and 3.
   - Index 1 can move to indices 0, 2, and 4.
     This is represented as a 2D vector `moves`.

4. **BFS**:

   - A queue is used to store states in the form of `{current_state, empty_tile_position}`.
   - A set is used to track visited states to avoid re-exploration.
   - For each state, generate all valid next states by swapping the empty tile with its neighbors. If a next state hasn't been visited, add it to the queue.

5. **Termination**:

   If the target configuration is reached, return the current depth (number of moves). If the queue is exhausted and the target isn't reached, return `-1`.

Complexity

Time Complexity:

- The total number of possible states for a 2x3 board is `6! = 720` (factorial of 6), as there are 6 tiles. BFS ensures we visit each state at most once.
- For each state, we perform a constant number of swaps (at most 4 neighbors for any tile).
  Thus, the time complexity is
  O(720 × 4) = O(2880) ≈ O(1) since 720 is a constant.

Space Complexity:

- The queue can store up to 720 states, and each state requires `O(6)` space for the string and position.
- The visited set also stores up to 720 states.
  Thus, the space complexity is
  O(720 × 6) = O(4320) ≈ O(1).

Code

```
class Solution {
public:
    int slidingPuzzle(vector<vector<int>>& board) {
        string goal = "123450";
        vector<vector<int>> moves = {{1, 3}, {0, 2, 4}, {1, 5},
                                     {0, 4}, {1, 3, 5}, {2, 4}};

        string start(6,' ');
        for(int i = 0; i<2; i++){
            for(int j = 0; j<3; j++){
                start[i * 3 + j] = board[i][j] + '0';
            }
        }

        queue<pair<string, int>> q;
        q.push({goal,5});

        unordered_set<string> vis;
        vis.insert(goal);
        int depth = 0;
        while(!q.empty()){
            int size = q.size();

            for(int i = 0; i<size; i++){
                string top = q.front().first;
                int to_swap = q.front().second;
                q.pop();
                if(top == start) return depth;

                for(auto j: moves[to_swap]){
                    swap(top[j], top[to_swap]);
                    if(vis.count(top) == 0){
                        vis.insert(top);
                        q.push({top, j});
                    }
                    swap(top[j], top[to_swap]);
                }
```

```
            }

            depth++;
        }

        return -1;
    }
};
```