773. Sliding Puzzle

On a 2x3 `board`, there are 5 tiles represented by the integers 1 through 5, and an empty square represented by 0.

A move consists of choosing `0` and a 4-directionally adjacent number and swapping it.

The state of the board is *solved* if and only if the `board` is `[[1,2,3],[4,5,0]].`

Given a puzzle board, return the least number of moves required so that the state of the board is solved. If it is impossible for the state of the board to be solved, return -1.

```
class Solution {
    public int slidingPuzzle(int[][] board) {

        String target = "123450";
        String start = "";

        for (int i = 0; i < board.length; i++) {
            for (int j = 0; j < board[0].length; j++) {
                start += board[i][j];
            }
        }

        HashSet<String> visited = new HashSet<>();

        // all the positions 0 can be swapped to
        int[][] dirs = new int[][] { { 1, 3 }, { 0, 2, 4 },
                { 1, 5 }, { 0, 4 }, { 1, 3, 5 }, { 2, 4 } };

        Queue<String> queue = new LinkedList<>();
        queue.offer(start);
        visited.add(start);
        int res = 0;

        while (!queue.isEmpty()) {
            int size = queue.size();

            for (int i = 0; i < size; i++) {
                String cur = queue.poll();
                if (cur.equals(target))
                    return res;

                int zero = cur.indexOf('0');

                for (int dir : dirs[zero]) {
                    String next = swap(cur, zero, dir);
                    if (visited.contains(next))
                        continue;

                    visited.add(next);
                    queue.offer(next);
                }
            }
```

```java
            res++;
        }

        return -1;
    }

    private String swap(String str, int i, int j) {
        StringBuilder sb = new StringBuilder(str);
        sb.setCharAt(i, str.charAt(j));
        sb.setCharAt(j, str.charAt(i));
        return sb.toString();
    }
}
```