在一个 2 x 3 的板上（board）有 5 块砖瓦，用数字 1~5 来表示，以及一块空缺用 0 来表示.
一次移动定义为选择 0 与一个相邻的数字（上下左右）进行交换.
最终当板 board 的结果是 [[1,2,3],[4,5,0]] 谜板被解开。
给出一个谜板的初始状态，返回最少可以通过多少次移动解开谜板，如果不能解开谜板，则返回 -1 。

对于这种计算最小步数的问题，要敏感地想到 BFS 算法。 分析：

1. 每次先找到数字 0，然后和周围的数字进行交换，形成新的局面加入队列..... 当第一次到达target时，就得到了赢得游戏的最少步数。 对于第二个问题，我们这里的board仅仅是 2x3 的二维数组，所以可以压缩成一个一维字符串。 其中比较有技巧性的点在于，二维数组有「上下左右」的概念，压缩成一维后，如何得到某一个索引上下左右的索引？ 手动写出来这个映射就行了：

```
vector<vector<int>> neighbor = {
    { 1, 3 },
    { 0, 4, 2 },
    { 1, 5 },
    { 0, 4 },
    { 3, 1, 5 },
    { 4, 2 }
};
```

在一维字符串中，索引i在二维数组中的的相邻索引为neighbor[i]。

```
class Solution {
public:
  int slidingPuzzle(std::vector<std::vector<int>>& board) {
    int         row = 2, col = 3;
    std::string start = "", target = "123450";
    // 将2*3矩阵转换为字符串
    for (int i = 0; i < row; i++) {
      for (int j = 0; j < col; j++) {
        start.push_back(board[i][j] + '0');
      }
    }

    // 记录一维数组与二维数组的映射
    std::vector<std::vector<int>> neighbor =
        {{1, 3}, {0, 4, 2}, {1, 5}, {0, 4}, {3, 1, 5}, {4, 2}};

    // BFS框架
    std::queue<std::string>         que;
    std::unordered_set<std::string> visited;
    que.push(start);
    visited.insert(start);

    int step = 0;
```

```
    while (!que.empty()) {
      int sz = que.size();

      for (int i = 0; i < sz; i++) {  // 保证处理完成一批可能的转换后替换
        std::string cur = que.front();
        que.pop();

        if (cur == target) {
          // 是否达到目标
          return step;
        }

        int idx = 0;  // 查找数字0所在位置
        for (; cur[idx] != '0'; idx++)
          ;

        // 将0和相邻数字交换
        for (int adj : neighbor[idx]) {
          std::string new_string = cur;
          std::swap(new_string[adj], new_string[idx]);

          // 防止重复
          if (!visited.count(new_string)) {
            que.push(new_string);
            visited.insert(new_string);
          }
        }
      }
      step++;
    }

    return -1;
  }
};
```