752. Open the Lock

Medium 1379 54 Add to List Share

You have a lock in front of you with 4 circular wheels. Each wheel has 10 slots: `'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'`. The wheels can rotate freely and wrap around: for example we can turn `'9'` to be `'0'`, or `'0'` to be `'9'`. Each move consists of turning one wheel one slot.

The lock initially starts at `'0000'`, a string representing the state of the 4 wheels.

You are given a list of `deadends` dead ends, meaning if the lock displays any of these codes, the wheels of the lock will stop turning and you will be unable to open it.

Given a `target` representing the value of the wheels that will unlock the lock, return the minimum total number of turns required to open the lock, or -1 if it is impossible.

Example 1:

```
Input: deadends = ["0201","0101","0102","1212","2002"], target = "0202"
Output: 6
Explanation:
A sequence of valid moves would be "0000" -> "1000" -> "1100" -> "1200" ->
"1201" -> "1202" -> "0202".
Note that a sequence like "0000" -> "0001" -> "0002" -> "0102" -> "0202" would
be invalid,
because the wheels of the lock become stuck after the display becomes the dead
end "0102".
```

Example 2:

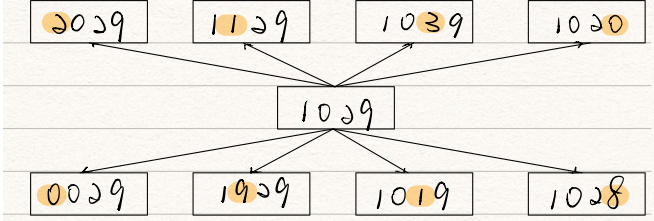
```
Input: deadends = ["8888"], target = "0009"
Output: 1
Explanation:
We can turn the last wheel in reverse to move from "0000" -> "0009".
```

Example 3:

```
Input: deadends = ["8887","8889","8878","8898","8788","8988","7888","9888"],
target = "8888"
Output: -1
Explanation:
We can't reach the target without getting stuck.
```

Example 4:

```
Input: deadends = ["0000"], target = "8888"
Output: -1
```



Idea: shortest path in a undirected unweighted graph -> BFS

Each node has at most 8 neighbors.

Time complexity: O(8 * 10000)
Space complexity: O(10000 + D)

```cpp
class Solution {
public:
    int openLock(vector<string>& deadends, string target) {
        string start = "0000";

        queue<string> q;
        set<string> visited;

        q.push(start);
        visited.insert(start);
        int turn = 0;

        while(q.size() != 0) {
            int sz = (int)q.size();
            for (int i = 0; i < sz; i++) {
                string cur = q.front();
                q.pop();
                if (find(deadends.begin(), deadends.end(), cur) != deadends.end()) {
                    continue;
                }
                if (cur.compare(target) == 0) {
                    return turn;
                }
                for (string adj : get_adj(cur)) {
                    if (visited.find(adj) == visited.end()) {
                        q.push(adj);
                        visited.insert(adj);
                    }
                }
            }
            turn++;
        }
        return -1;
    }

private:
    char get_up(char c) {
        char res;
        if (c == '9') {
            res = '0';
        } else {
            res = c + 1;
        }
        return res;
    }

    char get_down(char c) {
        char res;
        if (c == '0') {
            res = '9';
        } else {
            res = c - 1;
        }
        return res;
    }

    vector<string> get_adj(string str) {
        vector<string> res;

        for (int i = 0; i < str.length(); i++) {
            char c = str.at(i);
            string before = str.substr(0, i);
            string after = str.substr(i + 1, str.length());
            res.push_back(before + get_up(c) + after);
            res.push_back(before + get_down(c) + after);
        }

        return res;
    }
};
```