Open the Lock

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
```

Note:

1. The length of `deadends` will be in the range `[1, 500]`.
2. `target` will not be in the list `deadends`.
3. Every string in `deadends` and the string `target` will be a string of 4 digits from the 10,000 possibilities `'0000'` to `'9999'`.

Solution 1

The expected output is the same as my output, but it tells me wrong answer. Why??

written by ryx original link here

Solution 2

Every node has 8 edges. The nodes in dead ends cannot be visited. Find the shortest path from the initial node to the target.

```
class Solution {
public:
    int openLock(vector<string>& deadends, string target) {
        unordered_set<string> dds(deadends.begin(), deadends.end());
        unordered_set<string> visited;
        queue<string> bfs;
        string init = "0000";
        if (dds.find(init) != dds.end()) return -1;
        visited.insert("0000");
        bfs.push("0000");
        int res = 0;
        while (!bfs.empty()) {
            int sz = bfs.size();
            for (int i = 0; i < sz; i++) {
                string t = bfs.front(); bfs.pop();
                vector<string> nbrs = move(nbrStrs(t));
                for (auto s : nbrs) {
                    if (s == target) return ++res;
                    if (visited.find(s) != visited.end()) continue;
                    if (dds.find(s) == dds.end()) {
                        bfs.push(s);
                        visited.insert(s);
                    }
                }
            }
            ++res;
        }
        return -1;
    }


    vector<string> nbrStrs(string key) {
        vector<string> res;
        for (int i = 0 ; i < 4; i++) {
            string tmp = key;
            tmp[i] = (key[i] - '0' + 1) % 10 + '0';
            res.push_back(tmp);
            tmp[i] = (key[i] - '0' - 1 + 10) % 10 + '0';
            res.push_back(tmp);
         }
        return res;
    }
};
```

Bidirectional BFS improves the efficiency

```
    int openLock(vector<string>& deadends, string target) {
        unordered_set<string> dds(deadends.begin(), deadends.end());
        unordered_set<string> q1, q2, pass, visited;
        string init = "0000";
        if (dds.find(init) != dds.end() || dds.find(target) != dds.end()) return -1;
        visited.insert("0000");
        q1.insert("0000"), q2.insert(target);
        int res = 0;
        while (!q1.empty() && !q2.empty()) {
            if (q1.size() > q2.size()) swap(q1, q2);
            pass.clear();
            for (auto ss : q1) {
                vector<string> nbrs = nbrStrs(ss);
                for (auto s : nbrs) {
                    if (q2.find(s) != q2.end()) return res + 1;
                    if (visited.find(s) != visited.end()) continue;
                    if (dds.find(s) == dds.end()) {
                        pass.insert(s);
                        visited.insert(s);
                    }
                }
            }
            swap(q1, pass);
            res++;
        }
        return -1;
    }
```

written by IsingModel original link here

Solution 3

Shortest path finding, when the weights are constant, as in this case = 1, BFS is the best way to go.
Best way to avoid TLE is by using deque and popleft() .
[Using list() and pop(0) is a linear operation in Python, resulting in TLE]

Python:

```python
    def openLock(self, deadends, target):
        marker, depth = 'x', 0
        visited, q, deadends = set(), deque(['0000', marker]), set(deadends)

        while q:
            node = q.popleft()
            if node == target:
                return depth
            if node in visited or node in deadends:
                continue
            if node == marker and not q:
                return -1
            if node == marker:
                q.append(marker)
                depth += 1
            else:
                visited.add(node)
                q.extend(self.successors(node))
        return -1

    def successors(self, src):
        res = []
        for i, ch in enumerate(src):
            num = int(ch)
            res.append(src[:i] + str((num - 1) % 10) + src[i+1:])
            res.append(src[:i] + str((num + 1) % 10) + src[i+1:])
        return res
```

Java:

```
public static int openLock(String[] deadends, String target) {
        Queue<String> q = new LinkedList<>();
        Set<String> deads = new HashSet<>(Arrays.asList(deadends));
        Set<String> visited = new HashSet<>();

        int depth = 0;
        String marker = "*";
        q.addAll(Arrays.asList("0000", "*"));
        while(!q.isEmpty()) {
            String node = q.poll();
            if(node.equals(target))
                return depth;
            if(visited.contains(node) || deads.contains(node))
                continue;
            if(node.equals(marker) && q.isEmpty())
                return -1;
            if(node.equals(marker)) {
                q.add(marker);
                depth += 1;
            } else {
                visited.add(node);
                q.addAll(getSuccessors(node));
            }
        }
        return depth;
    }

    private static List<String> getSuccessors(String str) {
        List<String> res = new LinkedList<>();
        for (int i = 0; i < str.length(); i++) {
            res.add(str.substring(0, i) + (str.charAt(i) == '0' ? 9 :  str.charAt(i
) - '0' - 1) + str.substring(i+1));
            res.add(str.substring(0, i) + (str.charAt(i) == '9' ? 0 :  str.charAt(i
) - '0' + 1) + str.substring(i+1));
        }
        return res;
    }
```

written by johnyrufus16 original link here

From Leetcoder.