EXPERIMENT - 9

Student Name: Sandeep Kumar **UID: 20BCS4885**

Branch: CSE **Section/Group: 603/A**

Semester: 6th semester **Subject: Competitive Coding**

Aim: Backtracking
Objective:

a) Binary Watch
b) Word Letter II

Problem 1: Binary Watch

Solution code:

```
class Solution {
public:
   vector<string> readBinaryWatch(int turnedOn) {
      vector<string> result;
      vector<int> hour_leds({ 1, 2, 4, 8 });
      vector<int> min_leds({ 1, 2, 4, 8, 16, 32 });
      vector<vector<string>> h_map(5);
      vector<vector<string>> m_map(7);

      for (int i = 0; i < 12; i++) {
         int num = 0;
         int hour = i;
         for (int j = hour_leds.size() - 1; j >= 0; j--) {
            if (hour >= hour_leds[j]) {
               hour -= hour_leds[j];
               ++num;
            }
         }
         h_map[num].push_back(std::to_string(i));
      }
      for (int i = 0; i < 60; i++) {
         int num = 0;
         int min = i;
         for (int j = min_leds.size() - 1; j >= 0; j--) {
            if (min >= min_leds[j]) {
               min -= min_leds[j];
               ++num;
            }
         }
         if (i < 10) {
            m_map[num].push_back("0" + std::to_string(i));
```

```
                }
                else {
                    m_map[num].push_back(std::to_string(i));
                }
            }
            for (int h_num = 0; h_num < h_map.size() && h_num <= turnedOn; h_num++) {
                int min_num = turnedOn - h_num;
                if (min_num < m_map.size()) {
                    for (int i = 0; i < h_map[h_num].size(); i++) {
                        for (int j = 0; j < m_map[min_num].size(); j++) {
                            result.push_back(h_map[h_num][i] + ":" + m_map[min_num][j]);
                        }
                    }
                }
            }
            return result;
        }
    };
```

Approach:

1. Save the hours mapping in vector<vecotr>, say h_map;
   Save the minutes mapping in vector<vector>, say m_map;

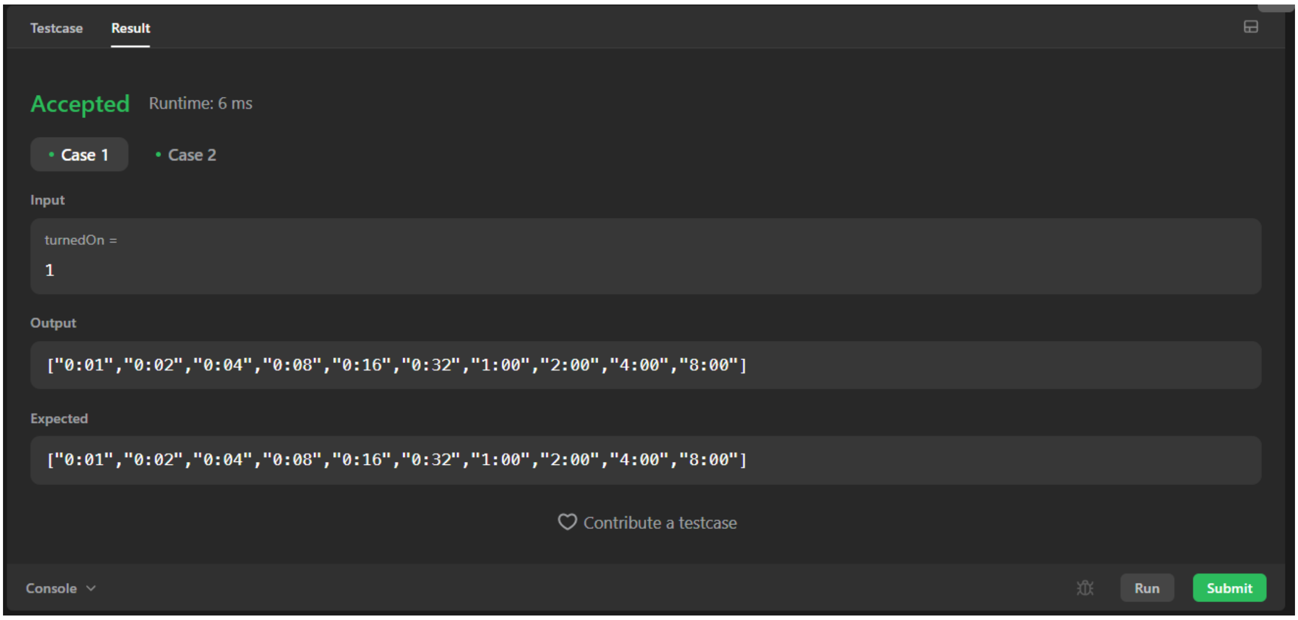
2. Index of h_map and m_map represents the number of LEDs.
   h_map[i] contains all the hours that can be represented by i LEDs.
   m_map[j] contains all the minutes that can be represented by j LEDs.

Complexity:

Time Complexity: O(turnedOn)

Space Complexity: O(411) + O(659);

Output:



Accepted Runtime: 6 ms

Input

turnedOn =
1

Output

```
["0:01","0:02","0:04","0:08","0:16","0:32","1:00","2:00","4:00","8:00"]
```

Expected

```
["0:01","0:02","0:04","0:08","0:16","0:32","1:00","2:00","4:00","8:00"]
```

Problem 2: Word Letter II

Input Code:

```
class Solution
{
    vector<vector<string>> ans;
    unordered_map<string, int> mp;

private:
    void backtrack(string &word, int len, vector<string> &seq)
    {
        if (mp[word] == 0)
        {
            reverse(seq.begin(), seq.end());
            ans.push_back(seq);
            reverse(seq.begin(), seq.end());
        }
        int level = mp[word];
        for (int i = 0; i < len; i++)
        {
            char original = word[i];
            for (int ch = 'a'; ch <= 'z'; ch++)
            {
                word[i] = ch;
                if (mp.find(word) != mp.end() && mp[word] + 1 == level)
                {
                    seq.push_back(word);
                    backtrack(word, len, seq);
                    seq.pop_back();
                }
            }
            word[i] = original;
        }
    }

public:
    vector<vector<string>> findLadders(string beginWord, string endWord, vector<string> &givenList)
    {
        unordered_set<string> wordList(givenList.begin(), givenList.end());
        queue<string> q;

        wordList.erase(beginWord);
        q.push({beginWord});
        mp[beginWord] = 0;

        int len = beginWord.size();

        while (!q.empty())
        {
            string word = q.front();
            q.pop();
            int level = mp[word];

            if (word == endWord)
                break;

            for (int i = 0; i < len; i++)
            {
                char original = word[i];
                for (int ch = 'a'; ch <= 'z'; ch++)
                {
                    word[i] = ch;
                    if (wordList.find(word) != wordList.end())
                    {
                        q.push(word);
                        mp[word] = level + 1;
                        wordList.erase(word);
                    }
                }
                word[i] = original;
            }
        }
        if (mp.find(endWord) != mp.end())
        {
            vector<string> seq;
            seq.push_back(endWord);
            backtrack(endWord, len, seq);
        }
        return ans;
    }
};
```

Approach:

Using Backtracking

We can get the solution by 2 main steps.

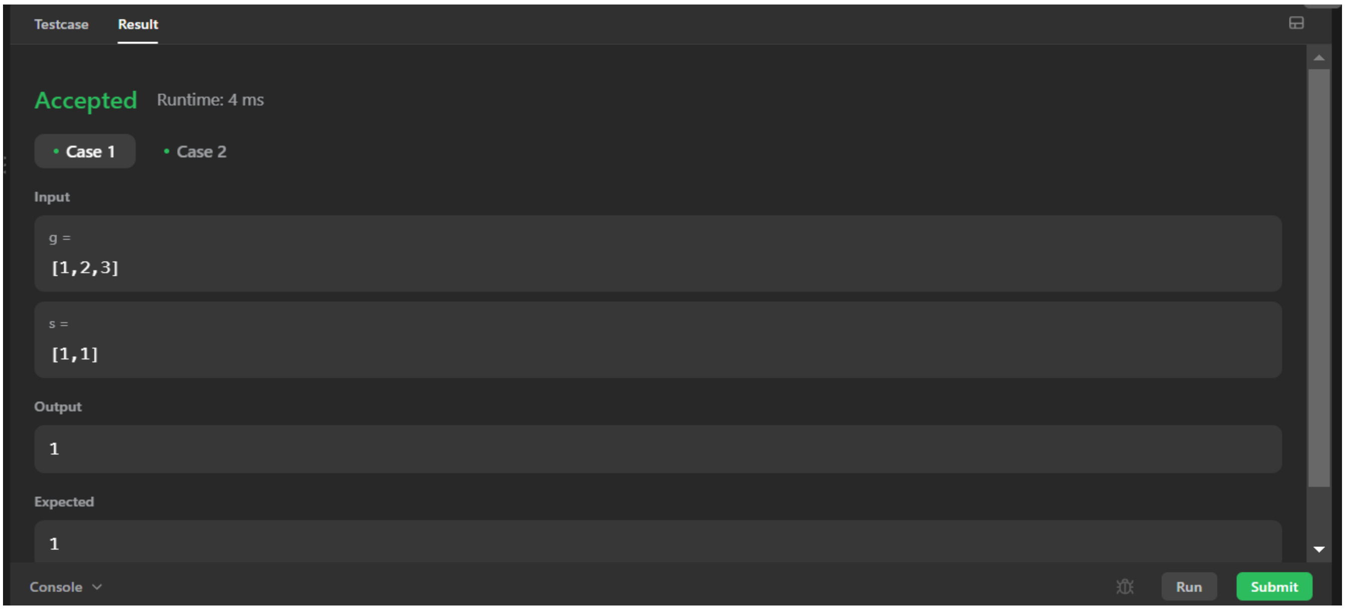
STEP 1: Find the shortest length of the ladder (similar to word-ladder-1 solution) and store each word with its level in the map.
STEP 2: Backtrack the map and get sequences (ladders) for the solution

Complexity:

Time Complexity: O(N^2 * L)

Space Complexity: O(N^2 * L).

Output:



Accepted Runtime: 4 ms

Input

```
g =
[1,2,3]

s =
[1,1]
```

Output

```
1
```

Expected

```
1
```