DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING
Discover. Learn. Empower.

Experiment-9

Student Name: Nikhil Kumar	**UID: 20BCS1817**
Branch: BE-CSE	**Section/Group:716/B**
Semester: 6th	**Date of Performance: 09/05/2023**
Subject Name: CC-II Lab	**Subject Code: 20CSP-351**

PROBLEM-1 :- Binary Watch

A binary watch has 4 LEDs on the top to represent the hours (0-11), and 6 LEDs on the bottom to represent the minutes (0-59). Each LED represents a zero or one, with the least significant bit on the right.

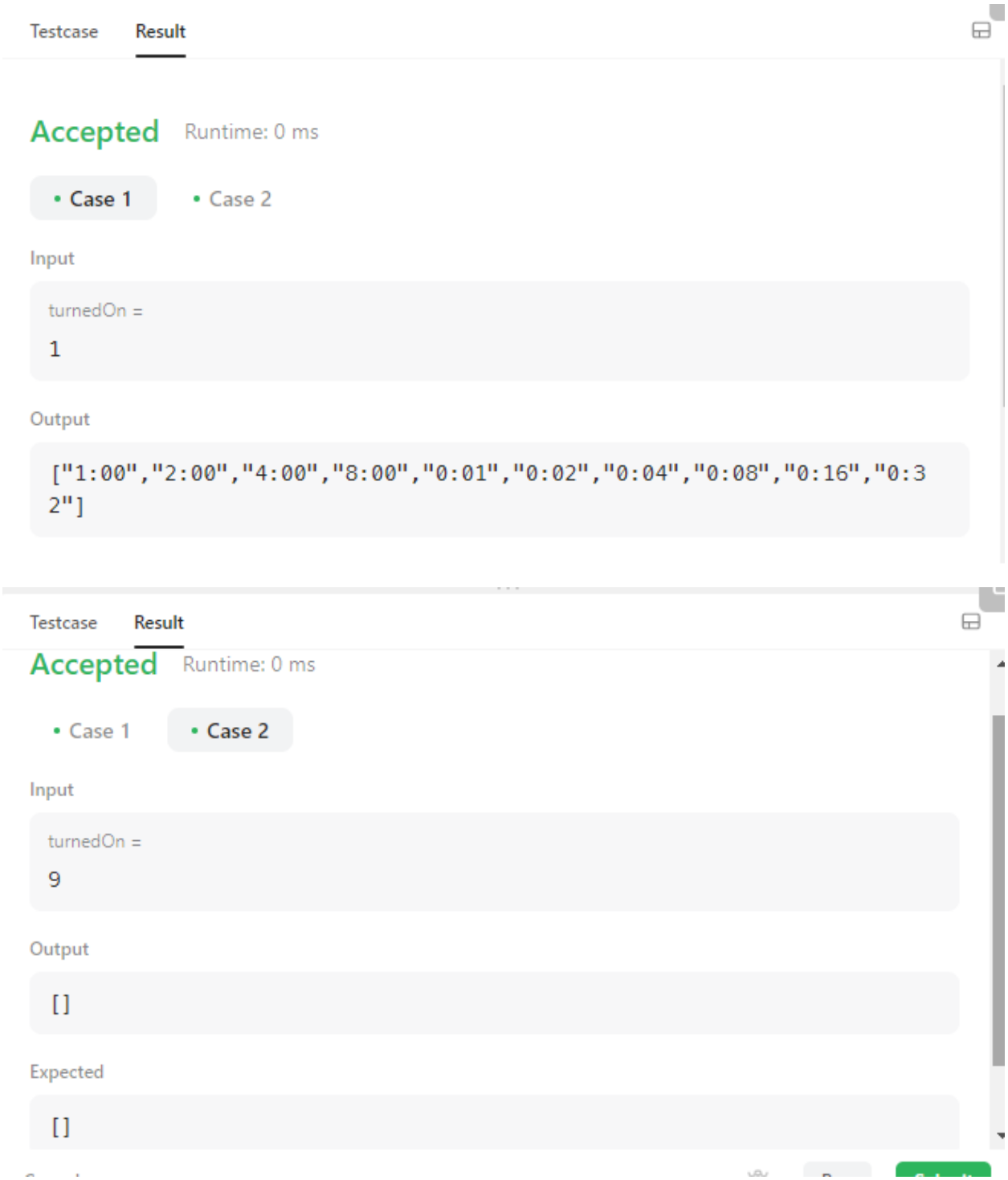
- For example, the below binary watch reads `"4:51"`.

CODE:-

```
class Solution {
public:
    vector<string> readBinaryWatch(int num) {
        union {
            struct {
                unsigned hours:4;
                unsigned minutes:6;
            };
            unsigned all;
        } time {0};
```

```
        vector<string> result;
        function<void(int, int)> place = [&](int n, int ifrom) {
          if (n == 0) {
            if (time.hours < 12 and time.minutes < 60) {
              char buf[20];
              sprintf(buf, "%d:%02d", time.hours, time.minutes);
              result.push_back(string(buf));
            }
          } else {
            for (int i = ifrom; i < 10; ++i) {
              if (!(time.all >> i & 1)) {
                time.all |= 1 << i;
                place(n - 1, i);
                time.all &= ~(1 << i);
              }
            }
          }
        };
    place(num, 0);
        return result;
    };
```

OUTPUT SCREENSHOT:-



PROBLEM-2: Word Ladder II

CODE:-

```
class Solution {
 public:
  vector<vector<string>> findLadders(string beginWord, string endWord,
                                     vector<string>& wordList) {
    unordered_set<string> wordSet{begin(wordList), end(wordList)};
    if (!wordSet.count(endWord))
      return {};
    // {"hit": ["hot"], "hot": ["dot", "lot"], ...}
    unordered_map<string, vector<string>> graph;
// Build graph from beginWord -> endWord
    if (!bfs(beginWord, endWord, wordSet, graph))
      return {};
    vector<vector<string>> ans;
    dfs(graph, beginWord, endWord, {beginWord}, ans);
    return ans;
  }
 private:
  bool bfs(const string& beginWord, const string& endWord,
           unordered_set<string>& wordSet,
           unordered_map<string, vector<string>>& graph) {
    unordered_set<string> currentLevelWords{beginWord};
    while (!currentLevelWords.empty()) {
 for (const string& word : currentLevelWords)
```

```
        wordSet.erase(word);
      unordered_set<string> nextLevelWords;
      bool reachEndWord = false;
      for (const string& parent : currentLevelWords) {
        vector<string> children;
        getChildren(parent, wordSet, children);
        for (const string& child : children) {
          if (wordSet.count(child)) {
            nextLevelWords.insert(child);
            graph[parent].push_back(child);
          }
          if (child == endWord)
            reachEndWord = true;
        }
      }
      if (reachEndWord)
        return true;
      currentLevelWords = move(nextLevelWords);
    }
  return false;
  }
void getChildren(const string& parent, const unordered_set<string>& wordSet,
                 vector<string>& children) {
  string s(parent);
  for (int i = 0; i < s.length(); ++i) {
      const char cache = s[i];
```

```
        for (char c = 'a'; c <= 'z'; ++c) {
          if (c == cache)
            continue;
          s[i] = c;  // Now is `child`
          if (wordSet.count(s))
            children.push_back(s);
        }
        s[i] = cache;
      }
    }
void dfs(const unordered_map<string, vector<string>>& graph,
             const string& word, const string& endWord, vector<string>&& path,
             vector<vector<string>>& ans) {
      if (word == endWord) {
        ans.push_back(path);
        return;
      }
      if (!graph.count(word))
        return;
      for (const string& child : graph.at(word)) {
        path.push_back(child);
  path.pop_back();
      }
    }
};
```

OUTPUT SCREENSHOT:-

Testcase Result

Accepted Runtime: 2 ms

• Case 1 • Case 2

Input

beginWord =
"hit"

endWord =
"cog"

wordList =
["hot","dot","dog","lot","log","cog"]

Output

[["hit","hot","dot","dog","cog"],["hit","hot","lot","log","cog"]]

Expected

[["hit","hot","dot","dog","cog"],["hit","hot","lot","log","cog"]]

Testcase Result

Accepted Runtime: 2 ms

• Case 1 • Case 2

Input

beginWord =
"hit"

endWord =
"cog"

wordList =
["hot","dot","dog","lot","log"]

Output

[]

Expected

[]