“**Experiment 3.2”**

**Aim:**

To demonstrate the concept of Backtracking.

**Objective:**

• The objective is to build problem solving capability and to learn the basic concepts of data structures.

• The implementation of ‘Binary Watch’ problem brushes up the concept of greedy approach.

• The implementation of ‘Word Ladder II’ problem brushes up the concept of greedy approach.

**Problem 1: “Binary Watch”**

<https://leetcode.com/problems/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".

Given an integer turnedOn which represents the number of LEDs that are currently on (ignoring the PM), return all possible times the watch could represent. You may return the answer in any order.

**Code:**

class Solution {

public:

    vector<string> readBinaryWatch(int turnedOn) {

        vector<string> rs;

        for (int h = 0; h < 12; h++){

            for (int m = 0; m < 60; m++){

                if (bitset<10>(h << 6 | m).count() == turnedOn)

                    rs.emplace\_back(to\_string(h) + (m < 10 ? ":0" : ":") + to\_string(m));

            }

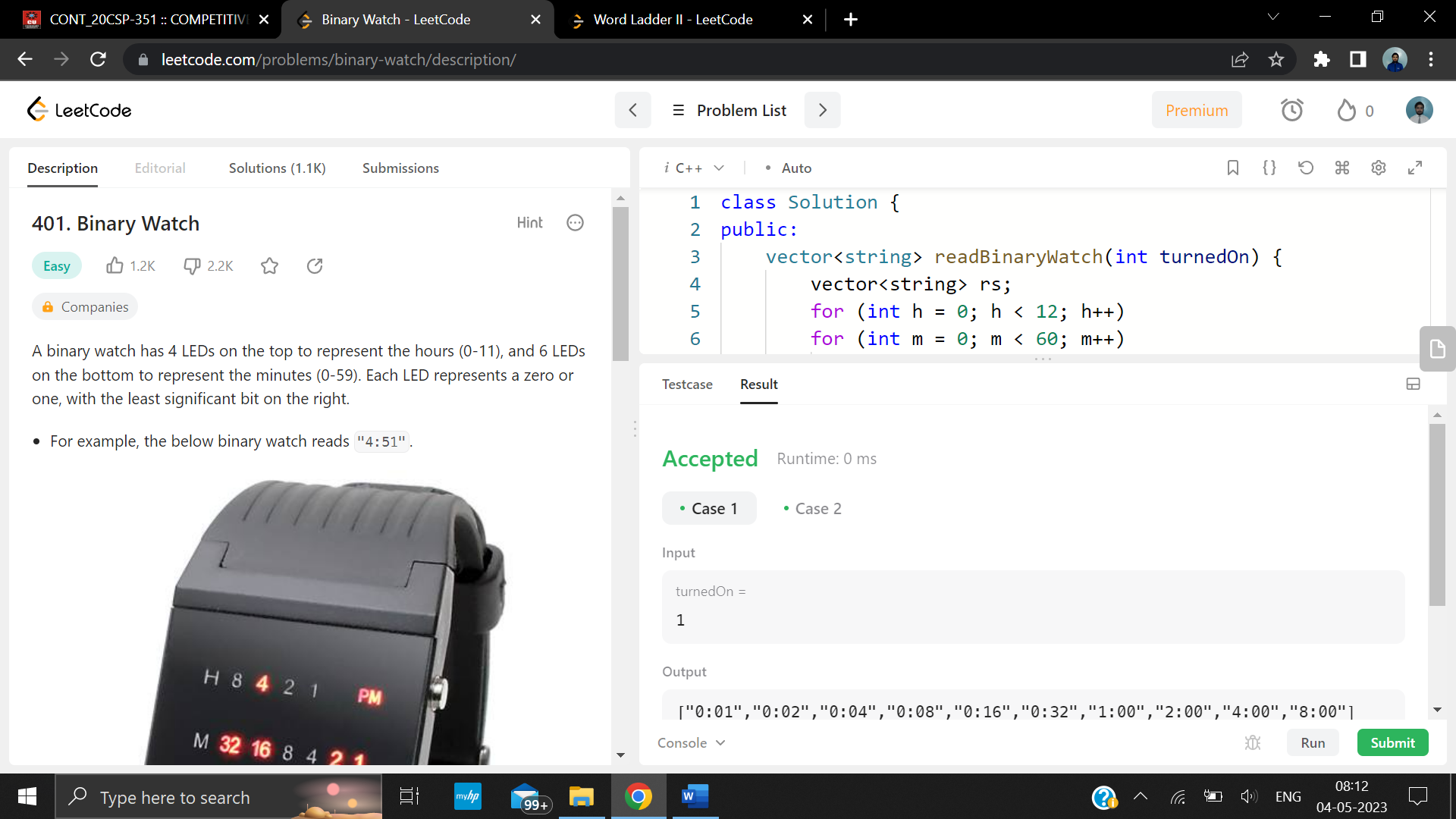
        }

        return rs;

    }

};

**Output:**



**Problem 2: “Word Ladder II”**

<https://leetcode.com/problems/word-ladder-ii/>

A transformation sequence from word beginWord to word endWord using a dictionary wordList is a sequence of words beginWord -> s1 -> s2 -> ... -> sk such that:

* Every adjacent pair of words differs by a single letter.
* Every Si for 1 <= i <= k is in wordList. Note that beginWord does not need to be in wordlist
* Sk == endWord

Given two words, beginWord and endWord, and a dictionary wordList, return all the shortest transformation sequences from beginWord to endWord, or an empty list if no such sequence exists. Each sequence should be returned as a list of the words [beginWord, S1, S2, … , Sk].

**Code:**

class Solution {

private:

    void dfs(string word,string beginWord,vector<string> &sequence,vector<vector<string>> &ans,unordered\_map<string,int> &mp){

        if(word==beginWord){

            ans.push\_back(sequence);

            return;

        }

        int steps=mp[word];

        int wordSize=beginWord.size();

        for(int i=0;i<wordSize;i++){

            char origChar=word[i];

            for(char c='a';c<='z';c++){

                word[i]=c;

                if(mp[word]+1==steps){

                    sequence.push\_back(word);

                    dfs(word,beginWord,sequence,ans,mp);

                    sequence.pop\_back();

                }

            }

            word[i]=origChar;

        }

    }

public:

    vector<vector<string>> findLadders(string beginWord, string endWord, vector<string>& wordList) {

        unordered\_set<string> st(wordList.begin(),wordList.end());

        unordered\_map<string,int> mp;

        mp[beginWord]=1;

        queue<string> q;

        q.push(beginWord);

        st.erase(beginWord);

        vector<vector<string>> ans;

        int wordSize=beginWord.size();

        //doing bfs

        while(!q.empty()){

            string word=q.front();

            int steps=mp[word];

            q.pop();

            if(word==endWord)

                break;

            // traverse

            for(int i=0;i<wordSize;i++){

                char origChar=word[i];

                for(char c='a';c<='z';c++){

                    word[i]=c;

                    if(st.find(word)!=st.end()){

                        st.erase(word);

                        mp[word]=steps+1;

                        q.push(word);

                    }

                }

                word[i]=origChar;

            }

        }

        if(mp[endWord]!=0){

            vector<string> sequence;

            sequence.push\_back(endWord);

            dfs(endWord,beginWord,sequence,ans,mp);

            int ansSize=ans.size();

            for(int i=0;i<ansSize;i++){

                reverse(ans[i].begin(),ans[i].end());

            }

        }

        return ans;

    }

};

**Output:**

