**Experiment 9.1**

**Student Name:** Sahil Kaundal **UID:** 21BCS8197

**Branch:** BE CSE (Lateral Entry) **Section/Group:** 616/A

**Semester:** 6th **Date of Performance:** 05/05/2023

**Subject Name:** CC-2 Lab **Subject Code:** 20CSP-351

1. **Aim/Overview of the practical:**

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.

<https://leetcode.com/problems/binary-watch/>

1. **Apparatus / Simulator Used:**

* Windows 7 or above
* Google Chrome

1. **Objective:**

* To understand the concept of Backtracking.

**Backtracking** is an algorithmic technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point in time (by time, here, is referred to the time elapsed till reaching any level of the search tree).  Backtracking can also be said as an improvement to the brute force approach. So basically, the idea behind the backtracking technique is that it searches for a solution to a problem among all the available options.  Initially, we start the backtracking from one possible option and if the problem is solved with that selected option then we return the solution else we backtrack and select another option from the remaining available options. There also might be a case where none of the options will give you the solution and hence we understand that backtracking won’t give any solution to that particular problem. We can also say that backtracking is a form of recursion. This is because the process of finding the solution from the various option available is repeated recursively until we don’t find the solution or we reach the final state. So we can conclude that backtracking at every step eliminates those choices that cannot give us the solution and proceeds to those choices that have the potential of taking us to the solution.

1. **Code:**

class Solution {

    public List<String> readBinaryWatch(int num) {

        List<String> result = new ArrayList<>();

        for (int hh = 0; hh < 12; hh++)

            for (int mn = 0; mn < 60; mn++)

                if (aux(hh, mn, num))

                    if (mn < 10)

                        result.add(String.format("%d:0%d", hh, mn));

                    else

                        result.add(String.format("%d:%d", hh, mn));

        return result;

    }

    private boolean aux(int hh, int mn, int num){

        int temp = 0;

        while(hh != 0 || mn != 0){

            if (hh !=0 ){

                temp += hh % 2;

                hh /=2;

            }

            if (mn != 0){

                temp += mn % 2;

                mn /= 2;

            }

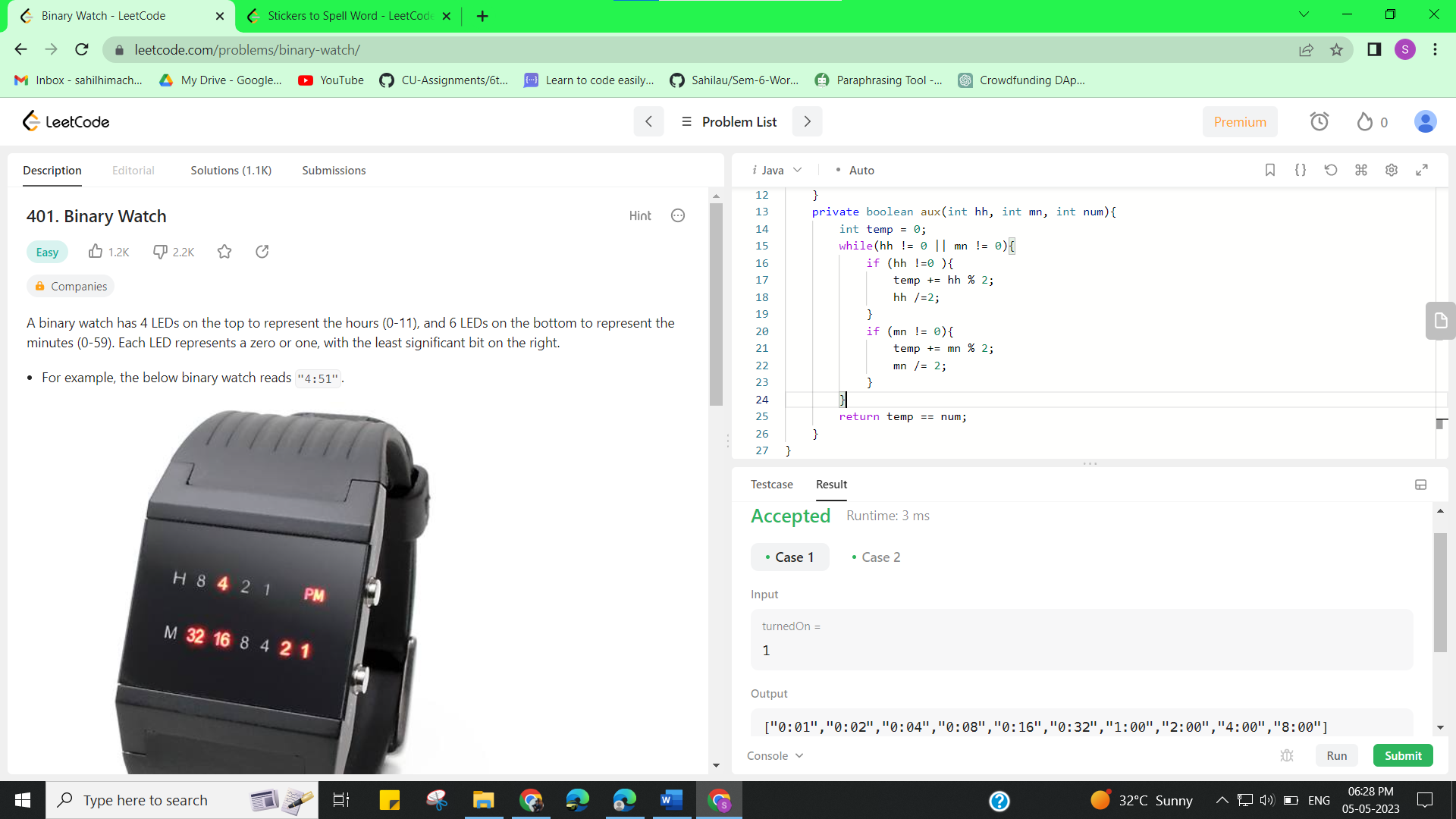
        }

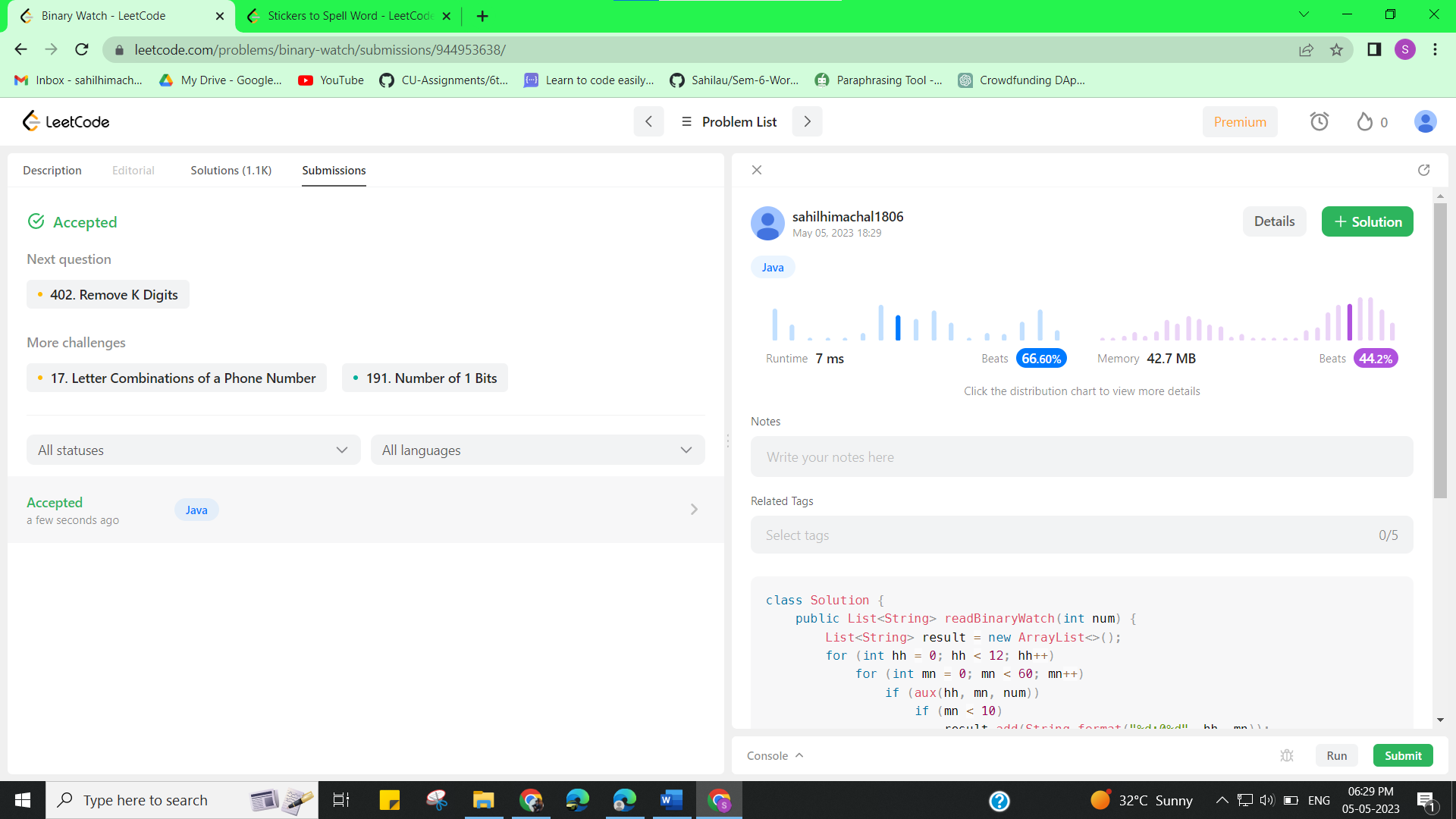
        return temp == num;

    }

}

**4. Result/Output/Writing Summary:**





**Experiment 9.2**

1. **Aim/Overview of the practical:**

Stickers to Spell Word

We are given n different types of stickers. Each sticker has a lowercase English word on it.

You would like to spell out the given string target by cutting individual letters from your collection of stickers and rearranging them. You can use each sticker more than once if you want, and you have infinite quantities of each sticker.

Return the minimum number of stickers that you need to spell out target. If the task is impossible, return -1.

<https://leetcode.com/problems/stickers-to-spell-word/>

1. **Apparatus / Simulator Used:**

* Windows 7 or above
* Google Chrome

1. **Objective:**

* To understand the concept of Backtracking.

**Backtracking** is an algorithmic technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point in time (by time, here, is referred to the time elapsed till reaching any level of the search tree).  Backtracking can also be said as an improvement to the brute force approach. So basically, the idea behind the backtracking technique is that it searches for a solution to a problem among all the available options.  Initially, we start the backtracking from one possible option and if the problem is solved with that selected option then we return the solution else we backtrack and select another option from the remaining available options. There also might be a case where none of the options will give you the solution and hence we understand that backtracking won’t give any solution to that particular problem. We can also say that backtracking is a form of recursion. This is because the process of finding the solution from the various option available is repeated recursively until we don’t find the solution or we reach the final state. So we can conclude that backtracking at every step eliminates those choices that cannot give us the solution and proceeds to those choices that have the potential of taking us to the solution.

1. **Code:**

class Solution {

public:

    int minStickers(vector<string>& stickers, string target) {

        vector<vector<int>> sticker\_counts(stickers.size(), vector<int>(26));

        unordered\_map<string, int> dp;

        for (int i = 0; i < stickers.size(); ++i) {

            for (const auto& c : stickers[i]) {

                ++sticker\_counts[i][c - 'a'];

            }

        }

        dp[""] = 0;

        return minStickersHelper(sticker\_counts, target, &dp);

    }

private:

    int minStickersHelper(const vector<vector<int>>& sticker\_counts, const string& target,

                          unordered\_map<string, int> \*dp) {

        if (dp->count(target)) {

            return (\*dp)[target];

        }

        int result = numeric\_limits<int>::max();

        vector<int> target\_count(26);

        for (const auto& c : target) {

            ++target\_count[c - 'a'];

        }

        for (const auto& sticker\_count : sticker\_counts) {

            if (sticker\_count[target[0] - 'a'] == 0) {

                continue;

            }

            string new\_target;

            for (int i = 0; i < target\_count.size(); ++i) {

                if (target\_count[i] - sticker\_count[i] > 0) {

                    new\_target += string(target\_count[i] - sticker\_count[i], 'a' + i);

                }

            }

            if (new\_target.length() != target.length()) {

                int num = minStickersHelper(sticker\_counts, new\_target, dp);

                if (num != -1) {

                    result = min(result, 1 + num);

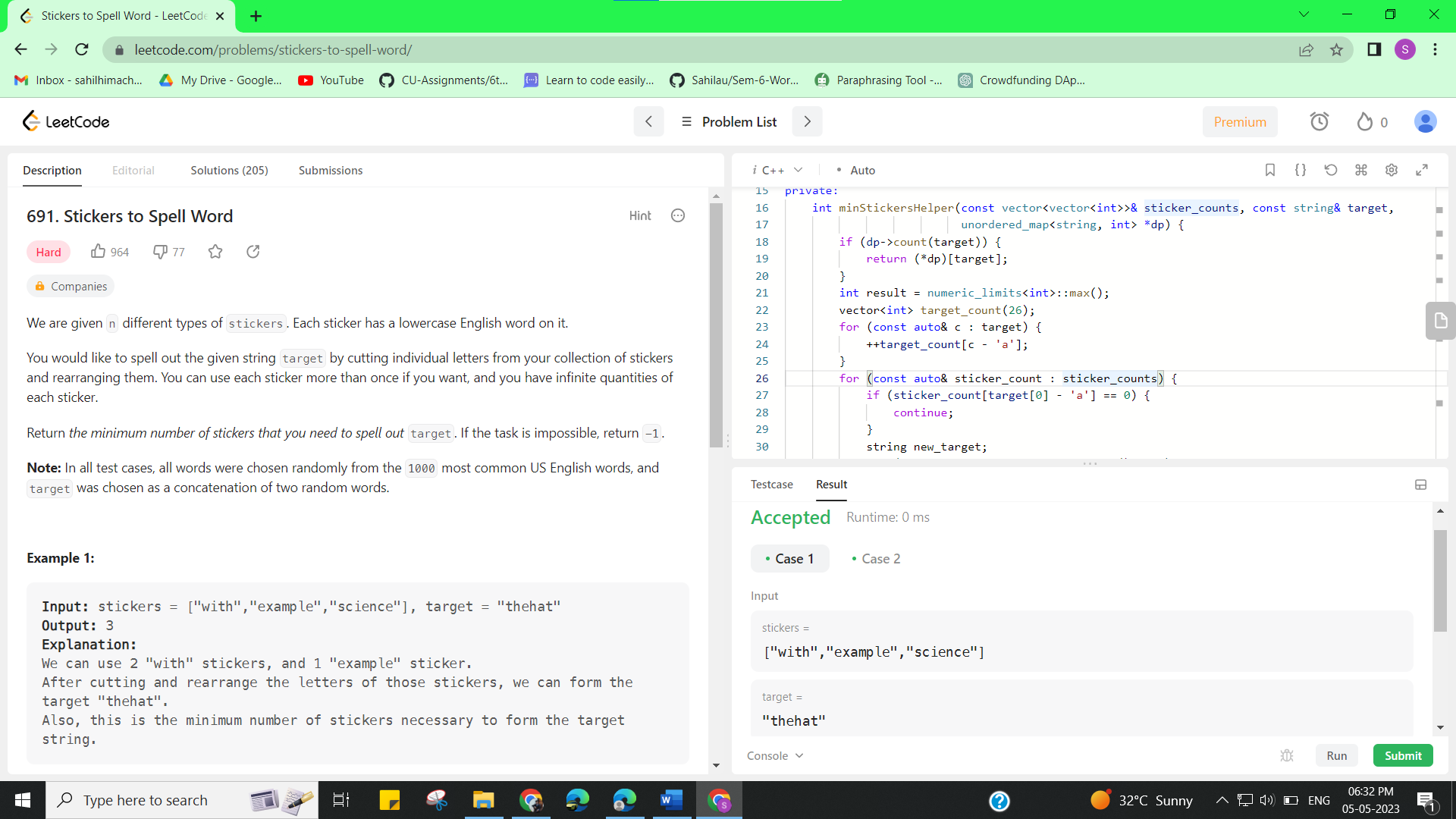
                }            }        }

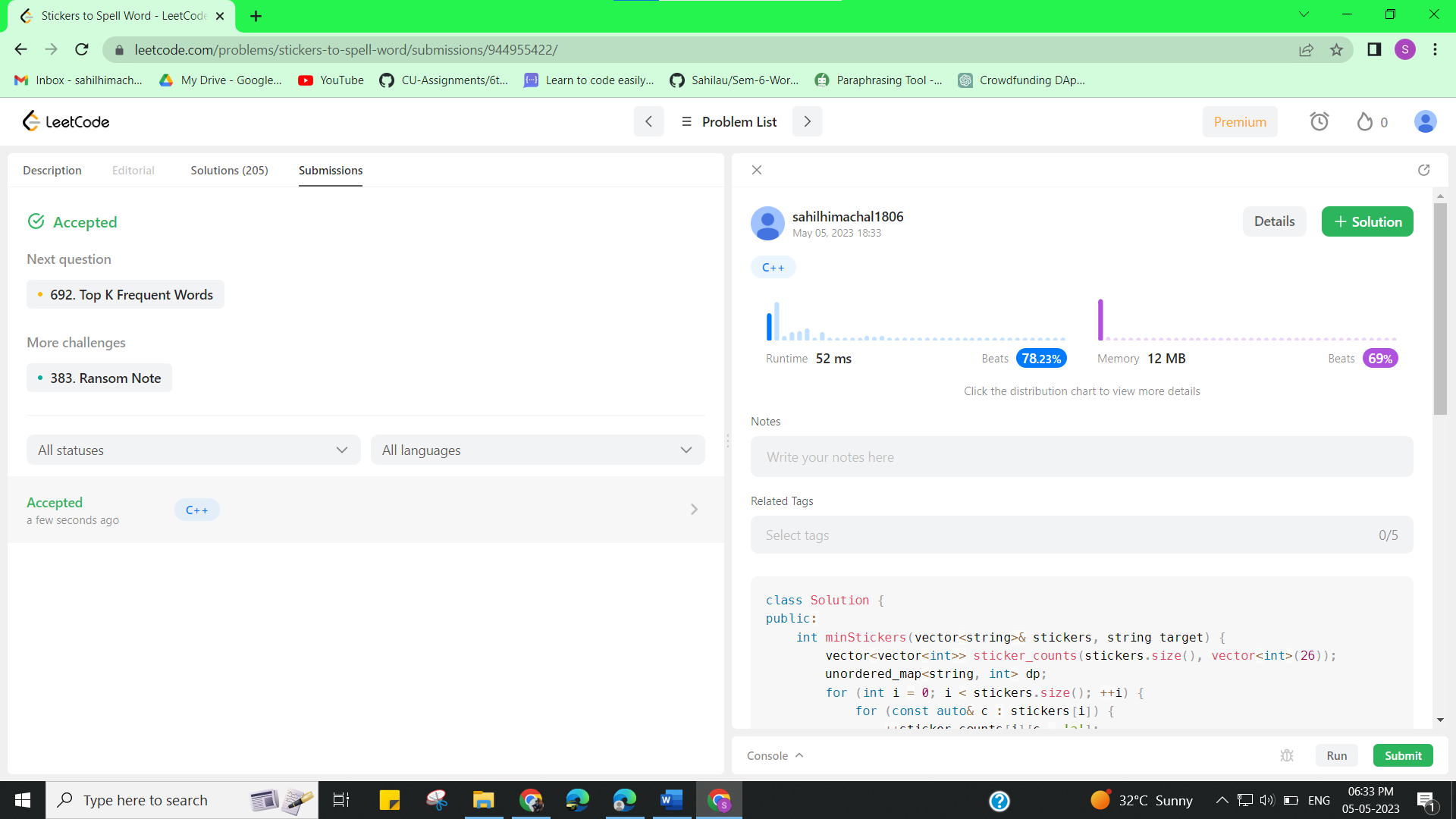
        (\*dp)[target] = (result == numeric\_limits<int>::max()) ? -1 : result;

        return (\*dp)[target];

    }};

1. **Result/Output/Writing Summary:**

****

****

**Learning outcomes (What I have learnt):**

* Learned the concept of Backtracking.