Experiment 3.2

Student Name: Praduman Kumar
Branch: BE CSE
Semester: 6th
Subject Name: CC Lab

UID: 20BSC9446
Section/Group: 714/A
Date of Performance: 05/05/2023
Subject Code: 20CSP-351

---

1. Aim/Overview of the practical:

To implement the concept of backtracking.

Binary Watch

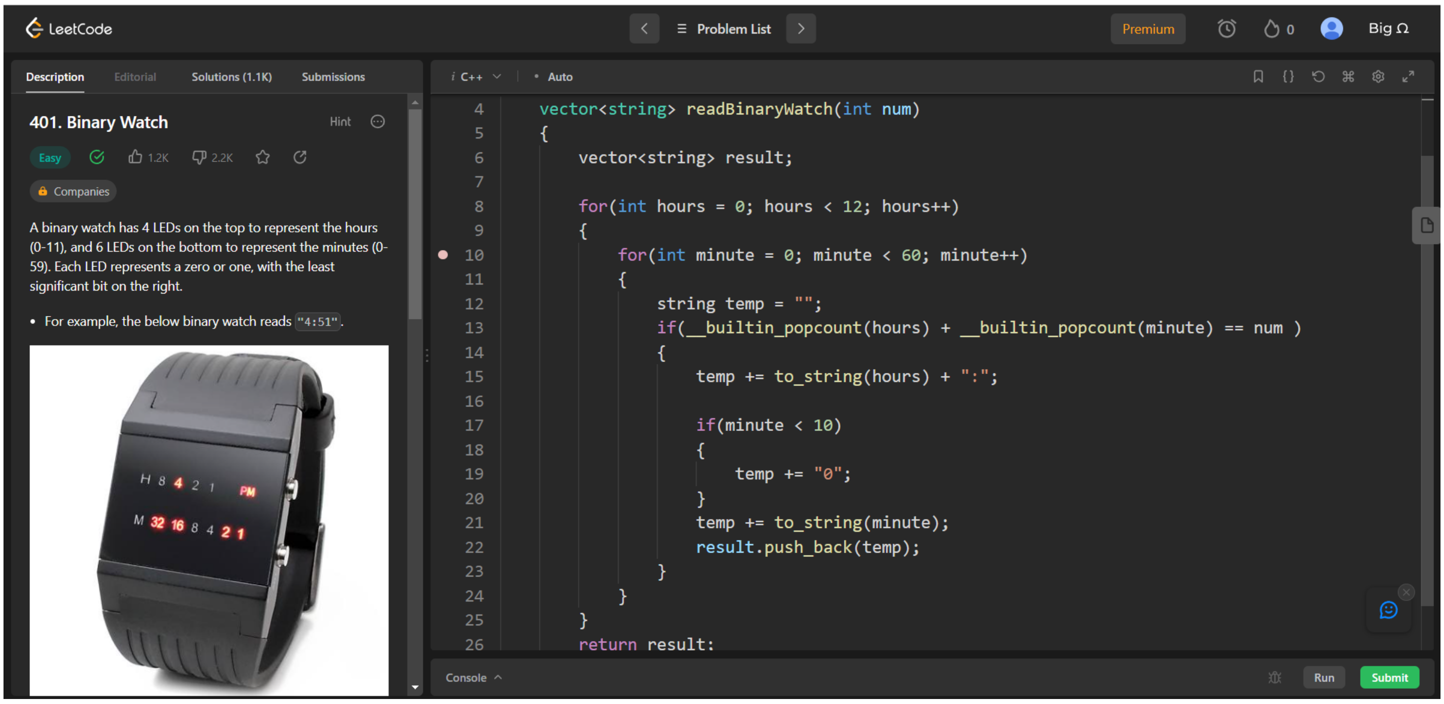
A binary watch has 4 LEDs on the top to represent the hours (0-11), and 6 LEDs on the bottom to represent the minutes (0-59). Each LED represents a zero or one, with the least significant bit on the right.

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

- **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;
    }
}
```

- **Result/Output/Writing Summary:**



401. Binary Watch

Easy

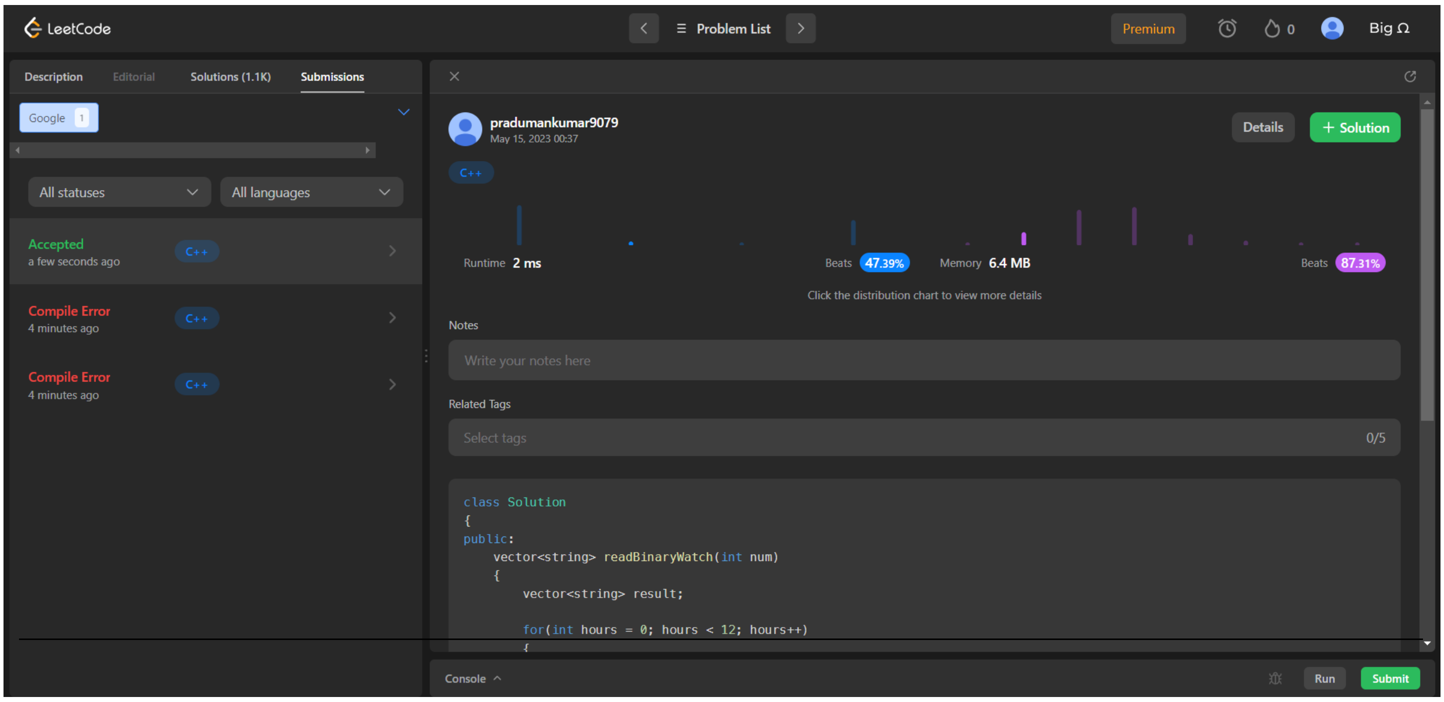
A binary watch has 4 LEDs on the top to represent the hours (0-11), and 6 LEDs on the bottom to represent the minutes (0-59). Each LED represents a zero or one, with the least significant bit on the right.

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

```cpp
    vector<string> readBinaryWatch(int num)
    {
        vector<string> result;

        for(int hours = 0; hours < 12; hours++)
        {
            for(int minute = 0; minute < 60; minute++)
            {
                string temp = "";
                if(__builtin_popcount(hours) + __builtin_popcount(minute) == num )
                {
                    temp += to_string(hours) + ":";

                    if(minute < 10)
                    {
                        temp += "0";
                    }
                    temp += to_string(minute);
                    result.push_back(temp);
                }
            }
        }
        return result;
```



pradumankumar9079
May 15, 2023 00:37

Accepted — a few seconds ago — C++
Compile Error — 4 minutes ago — C++
Compile Error — 4 minutes ago — C++

Runtime 2 ms — Beats 47.39%
Memory 6.4 MB — Beats 87.31%

Click the distribution chart to view more details

```cpp
class Solution
{
public:
    vector<string> readBinaryWatch(int num)
    {
        vector<string> result;

        for(int hours = 0; hours < 12; hours++)
        {
```

2. 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.

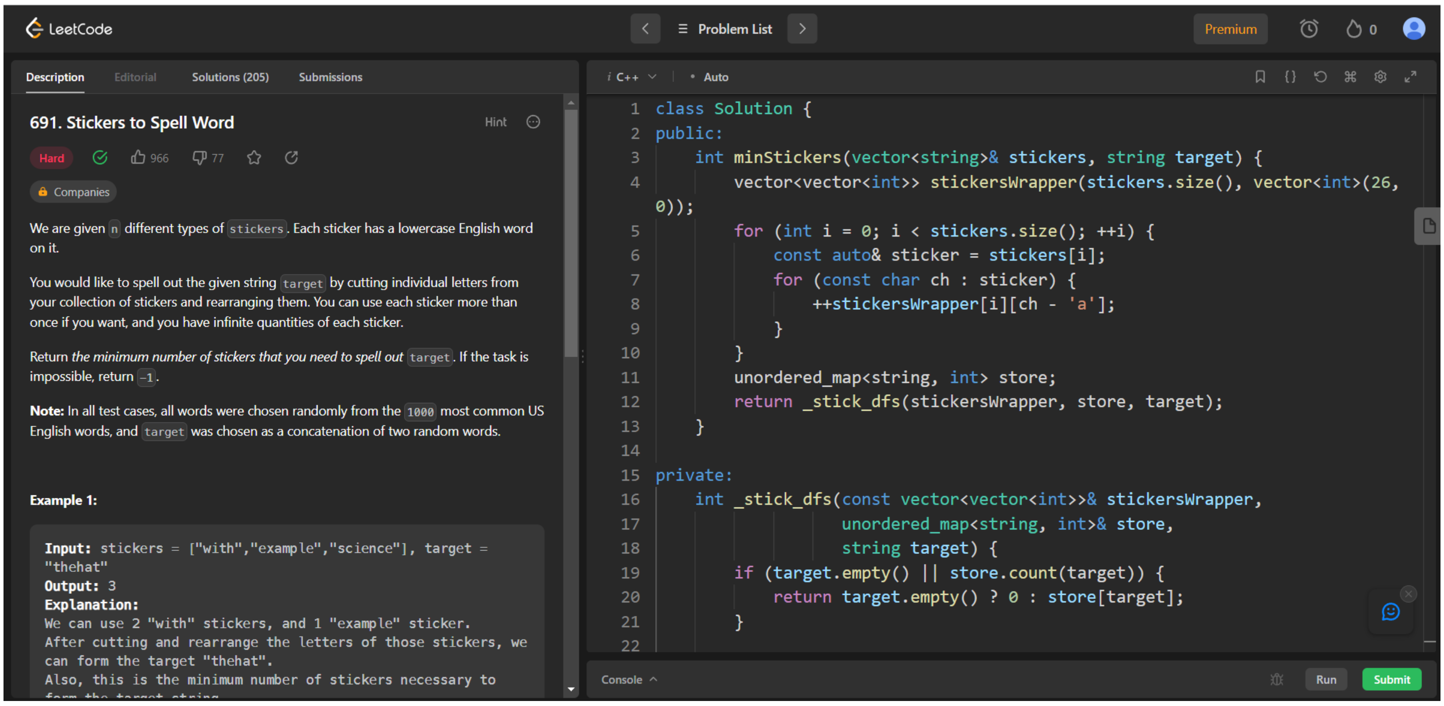
https://leetcode.com/problems/stickers-to-spell-word/

- **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 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];
    }};
```

- **Result/Output/Writing Summary:**



691. 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`.

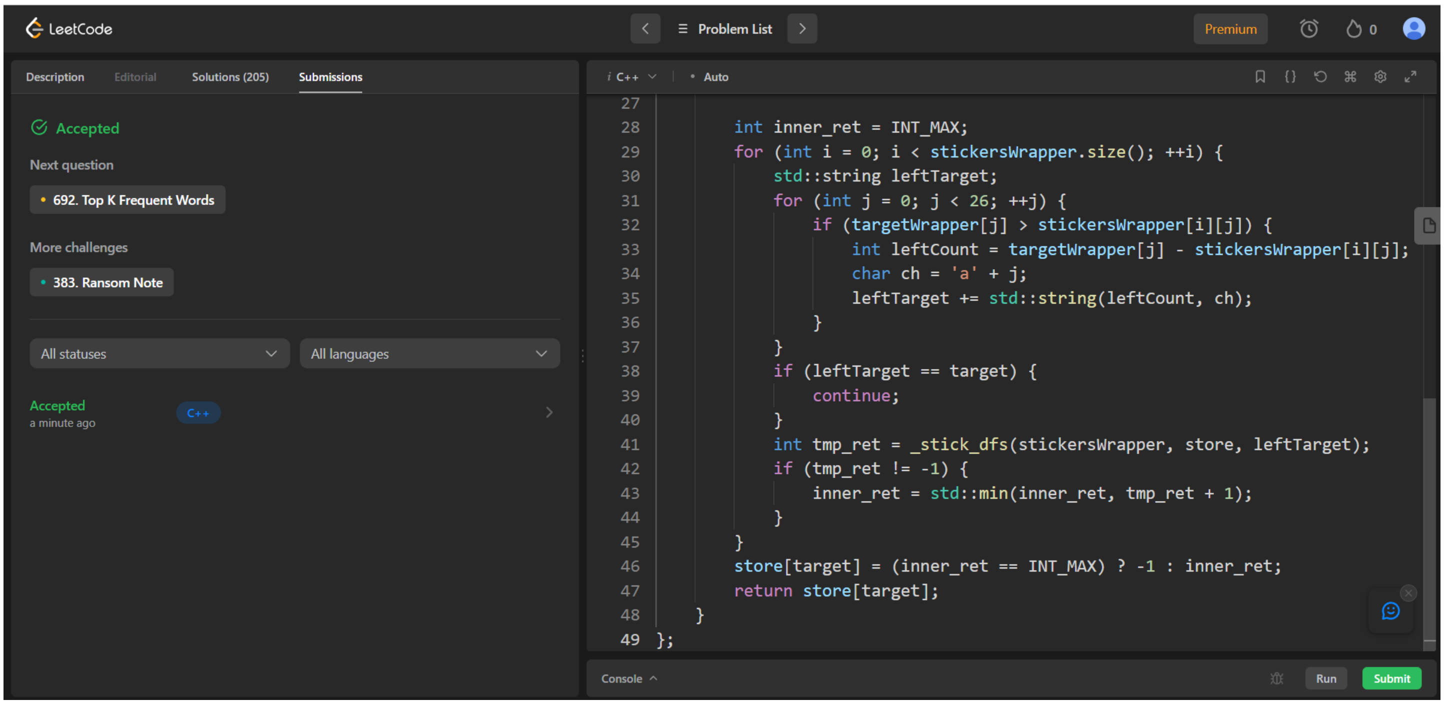
Note: In all test cases, all words were chosen randomly from the `1000` most common US English words, and `target` was chosen as a concatenation of two random words.

Example 1:

```
Input: stickers = ["with","example","science"], target =
"thehat"
Output: 3
Explanation:
We can use 2 "with" stickers, and 1 "example" sticker.
After cutting and rearrange the letters of those stickers, we
can form the target "thehat".
Also, this is the minimum number of stickers necessary to
```

```cpp
class Solution {
public:
    int minStickers(vector<string>& stickers, string target) {
        vector<vector<int>> stickersWrapper(stickers.size(), vector<int>(26,
0));
        for (int i = 0; i < stickers.size(); ++i) {
            const auto& sticker = stickers[i];
            for (const char ch : sticker) {
                ++stickersWrapper[i][ch - 'a'];
            }
        }
        unordered_map<string, int> store;
        return _stick_dfs(stickersWrapper, store, target);
    }

private:
    int _stick_dfs(const vector<vector<int>>& stickersWrapper,
                   unordered_map<string, int>& store,
                   string target) {
        if (target.empty() || store.count(target)) {
            return target.empty() ? 0 : store[target];
        }
```



Accepted

Accepted
a minute ago

```cpp
        int inner_ret = INT_MAX;
        for (int i = 0; i < stickersWrapper.size(); ++i) {
            std::string leftTarget;
            for (int j = 0; j < 26; ++j) {
                if (targetWrapper[j] > stickersWrapper[i][j]) {
                    int leftCount = targetWrapper[j] - stickersWrapper[i][j];
                    char ch = 'a' + j;
                    leftTarget += std::string(leftCount, ch);
                }
            }
            if (leftTarget == target) {
                continue;
            }
            int tmp_ret = _stick_dfs(stickersWrapper, store, leftTarget);
            if (tmp_ret != -1) {
                inner_ret = std::min(inner_ret, tmp_ret + 1);
            }
        }
        store[target] = (inner_ret == INT_MAX) ? -1 : inner_ret;
        return store[target];
    }
};
```

Learning outcomes (What I have learnt):

- Learned the concept of Backtracking.