**Problem Name:** Maximum Score Words Formed by Letters

**Topics:** String, Backtracking

**Companies:** Google, Amazon

**Level:** Hard

**Language:** C++

**Problem Statement:** Given a list of words, list of single letters (might be repeating) and score of every character.

Return the maximum score of **any** valid set of words formed by using the given letters (words[i] cannot be used two or more times).

It is not necessary to use all characters in letters and each letter can only be used once. Score of letters 'a', 'b', 'c', ... ,'z' is given by score[0], score[1], ... , score[25] respectively.

**Input Format:**

The first line contains integer n (no of words).

The Second line contains n different words.

The Third line contains integer m (no of letters).

The fourth line contain m letters.

Last line contains 26 integer value representing score of alphabets.

Ex.

4

dog cat dad good

9

a a c d d d g o o

1 0 9 5 0 0 3 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0

**Output Format:**

Print maximum possible score.

Ex.

23

**Constraints:**

* 1 <= words.length <= 14
* 1 <= words[i].length <= 15
* 1 <= letters.length <= 100
* letters[i].length == 1
* score.length == 26
* 0 <= score[i] <= 10
* words[i], letters[i] contains only lower-case English letters.

**Examples:**

**Input:** words = ["dog","cat","dad","good"], letters = ["a","a","c","d","d","d","g","o","o"], score = [1,0,9,5,0,0,3,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0]

**Output:** 23

**Explanation:**

Score a=1, c=9, d=5, g=3, o=2

Given letters, we can form the words "dad" (5+1+5) and "good" (3+2+2+5) with a score of 23.

Words "dad" and "dog" only get a score of 21.

**Brute force Solution:**

**Explanation:** Goal is to make subset using given words such that all words has a score, we need to find subset with maximum sum of these scores, but we have to follow few conditions:  
-- letter[] tells how many time chars can be used in subset  
-- for example : 'd' frequency is 3 in letters[],  
-- so that we can use only those words having in subset such freq. of d in the subset is also three  
-- example: [dog, dad] <- one 'd' in dog and two 'd's in dad which is equal to three 'd's  
-- we cannot exceed frequency of the char in our subset, it should be <= frequency(ch);

every word has two options either to contribute in the current call or do not contribute to access each we will need a variable idx = 0; idx+1 on each call untill all words are consumed we will take current word and we will keep decreasing frequencies of each char in that word in case if any of the char has frequency <= 0 this mean we cannot use that word; because to make any word frequency of any char >= 1 for each chars of the current word, we have some scores, we keep adding this scores

**Code:**

#include <bits/stdc++.h>

using namespace std;

int helper(vector<string>& words, int\* fmap, vector<int>& score, int idx){

    if(idx == words.size()){

        return 0;

    }

    int noContribScore = helper(words, fmap, score, idx+1);

    string currentWord = words[idx];

    int currentWordScore = 0;

    bool can\_we\_use\_currentWord = true;

    for(auto x : currentWord){

        char currentChar = x;

        if(fmap[currentChar - 'a'] <= 0){

            can\_we\_use\_currentWord = false;

        }

        fmap[currentChar - 'a'] -=1;

        currentWordScore += score[currentChar-'a'];

    }

    int contribScore = 0;

    if(can\_we\_use\_currentWord == true){

        contribScore = currentWordScore + helper(words, fmap, score, idx+1);

    }

    for(auto x : currentWord){

        fmap[x - 'a']+=1;

    }

    return max(contribScore, noContribScore);

}

int maxScoreWords(vector<string>& words, vector<char>& letters, vector<int>& score) {

    int fmap[26] = {0};

    for(auto x : letters){

        fmap[x-'a']++;

    }

    return helper(words, fmap, score, 0);

}

int main(){

    int n,m;

    cin>>n;

    vector<string> words(n);

    vector<int> score(26);

    for(int i=0; i<n; i++){

        cin>>words[i];

    }

    cin>>m;

    vector<char> letter(m);

    for(int i=0; i<m; i++){

        cin>>letter[i];

    }

    for(int i=0; i<26; i++){

        cin>>score[i];

    }

    cout<<maxScoreWords(words, letter, score);

}

**Time Complexity**: O(2n)

**Space Complexity:** O(N)

**Optimized Solution:**

**Explanation:** There are 2^n possible subsets of words that can be selected because a word may or may not be present in the ans, so for every word there are 2 options.  
so, to simulate the above procedure a word of being or not being in the final ans,

int lim=(1<<n); is using that int lim.

**Code:**

#include <bits/stdc++.h>

using namespace std;

int maxScoreWords(vector<string>& words, vector<char>& letters, vector<int>& score) {

    int ans=0;

    vector<int> cnt(26);

    fill(cnt.begin(),cnt.end(),0);

    for(auto x:letters)cnt[x-'a']++;

    int n=words.size();

    int lim=(1<<n);

    for(int mask=0;mask<lim;mask++){

        int res=0;

        vector<int> tcnt=cnt;

        int flag=0;

        for(int pos=n-1;pos>=0;pos--){

            if((mask>>pos)&1){

                string s=words[pos];

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

                    tcnt[s[i]-'a']--;

                    if(tcnt[s[i]-'a']<0){

                        flag=1;

                        break;

                    }

                    else res+=score[s[i]-'a'];

                }

            }

            if(flag==1)break;

        }

        if(flag==0)ans=max(ans,res);

    }

    return ans;

}

int main(){

    int n,m;

    cin>>n;

    vector<string> words(n);

    vector<int> score(26);

    for(int i=0; i<n; i++){

        cin>>words[i];

    }

    cin>>m;

    vector<char> letter(m);

    for(int i=0; i<m; i++){

        cin>>letter[i];

    }

    for(int i=0; i<26; i++){

        cin>>score[i];

    }

    cout<<maxScoreWords(words, letter, score);

}

**Time Complexity**: O(2n)

**Space Complexity:** O(N)