ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 6

«Динамическое программирование»

Выполнил работу

Ширкунова Мария

Академическая группа №J3114

Принято

Дунаев Максим Владимирович

Санкт-Петербург

2024

Содержание отчета

1. Введение

Цель работы: решение задачи уровня hard на платформе leetcode, используя метод решения динамическое программирование.

В рамках работы необходимо решить задачу 1255. Maximum Score Words Formed by Letters.

Задачи:

- Реализовать алгоритм.
- Протестировать алгоритм на платформе.
- Оценить сложность алгоритма.
- Оценить использование дополнительной памяти.
- Проанализировать, почему в решении необходимо ДП.

2. Реализация

Импортируем необходимые для решения библиотеки. Iostream – ввод/вывод, vector – хранение элементов в массивах, unordered_map – словарь для подсчета частот и скоров, string – для работы со строками.

```
#include <iostream>
#include <vector>
#include <unordered_map>
#include <string>

using namespace std;
```

Создаем функцию, решающую задачу. На вход принимаем неизменяемые массивы по ссылке: words, letters, score – согласно условию задачи. Создаем словарь частот каждой буквы. Считаем для каждого слова его скор (проходимся по каждой букве и суммируем скор буквы, брав значение из массива score). Проверяем, что количество используемых букв для каждого слова не превосходит их количество в массиве letters. Если превосходит, обнуляем скор. Далее используем ДП для нахождения максимального скора – ответ на задачу. Для этого проходимся по каждой комбинации (каждое слово можно либо взять в нее – 1, либо не взять – 0). Проходимся по всем словам, если оно входит в комбинацию, суммируем в текущий скор, считаем используемые буквы. Проверяем валидность комбинации. Если используемые буквы есть в letters в нужном количестве обновляем ответ. Возвращаем ответ.

```
int maxScoreWords(const vector<string>& words, const vector<char>& letters, const
vector<int>& score) {
    // Step 1: Считаем частоту каждой буквы
    unordered_map<char, int> letterCount;
    for (char letter : letters) {
        letterCount[letter]++;
    }

    // Step 2: Считаем word scores и проверяем валидность
    vector<int> wordScores;
    vector<vector<int>> wordLetterCounts(words.size(), vector<int>(26, 0));

    for (const string& word : words) {
        int currentScore = 0;
        bool canForm = true;

        for (char c : word) {
            currentScore += score[c - 'a'];
            // wordScores.size() - индекс слова (добавляем туда новое слово на каждом
шаге)
```

```
            wordLetterCounts[wordScores.size()][c - 'a']++;
        }

        // Проверяем можем ли мы составить это слово из наших букв
        for (char c : word) {
            if (wordLetterCounts[wordScores.size()][c - 'a'] > letterCount[c]) {
                canForm = false;
                break;
            }
        }

        if (canForm) {
            wordScores.push_back(currentScore);
        }
        else {
            wordScores.push_back(0); // Для невалидных слов ставим скор 0
        }
    }

    // Step 3: Используем ДП чтобы найти максимальный скор
    int maxScore = 0;
    int numWords = wordScores.size();

    // Есть 2^numWords комбинаций слов, 1-слово в комбинации, 0-нет
    for (int mask = 0; mask < (1 << numWords); ++mask) {
        unordered_map<char, int> usedLetters;
        int currentScore = 0;

        for (int i = 0; i < numWords; ++i) {
            if (mask & (1 << i)) { // Если i-ое слово в комбинации
                currentScore += wordScores[i];
                for (char c : words[i]) {
                    usedLetters[c]++;
                }
            }
        }

        // Проверяем что эта комбинация валидна по возможным буквам
        bool validCombination = true;
        for (const auto& entry : usedLetters) {
            if (entry.second > letterCount[entry.first]) {
                validCombination = false;
                break;
            }
        }

        if (validCombination) {
            maxScore = max(maxScore, currentScore);
        }
    }

    return maxScore;
}
```

Проверяем функцию. Берем примеры из теста и запускаем. Сравниваем ответы.

```
int main() {
    vector<string> words1 = { "dog", "cat", "dad", "good" };
    vector<char> letters1 = { 'a', 'a', 'c', 'd', 'd', 'd', 'g', 'o', 'o' };
    vector<int> score1 = { 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 };

    cout << "Maximum Score: " << maxScoreWords(words1, letters1, score1) << endl;

    return 0;
}
```

3. Экспериментальная часть

3.1. Подсчет по памяти.

```
int maxScoreWords(const vector<string>& words, const vector<char>& letters, const
vector<int>& score) {
    unordered_map<char, int> letterCount; // 4*letters.size() байт
    for (char letter : letters) { // 1 байт
        letterCount[letter]++;
    }

    vector<int> wordScores;  // 4*words.size() байт
    vector<vector<int>> wordLetterCounts(words.size(), vector<int>(26, 0)); //
4*26*words.size() байт

    for (const string& word : words) { //1*k байт, где k - максимальная длина слова в
words
        int currentScore = 0;  // 4 байт
        bool canForm = true; // 1 байт

        for (char c : word) { // 1 байт
            currentScore += score[c - 'a'];
            wordLetterCounts[wordScores.size()][c - 'a']++;
        }

        for (char c : word) { // 1 байт
            if (wordLetterCounts[wordScores.size()][c - 'a'] > letterCount[c]) {
                canForm = false;
                break;
            }
        }

        if (canForm) {
            wordScores.push_back(currentScore);
        }
        else {
            wordScores.push_back(0);
        }
    }

    int maxScore = 0; // 4 байт
    int numWords = wordScores.size(); // 4 байт

    for (int mask = 0; mask < (1 << numWords); ++mask) { // 4 байт
        unordered_map<char, int> usedLetters; // 4*26 байт
        int currentScore = 0; // 4 байт

        for (int i = 0; i < numWords; ++i) { // 4 байт
            if (mask & (1 << i)) {
                currentScore += wordScores[i];
                for (char c : words[i]) { // 1 байт
                    usedLetters[c]++;
                }
            }
        }

        bool validCombination = true; // 1 байт
        for (const auto& entry : usedLetters) { // 4 + 1 байт (пара-pair из char и int)
            if (entry.second > letterCount[entry.first]) {
                validCombination = false;
                break;
            }
        }
```

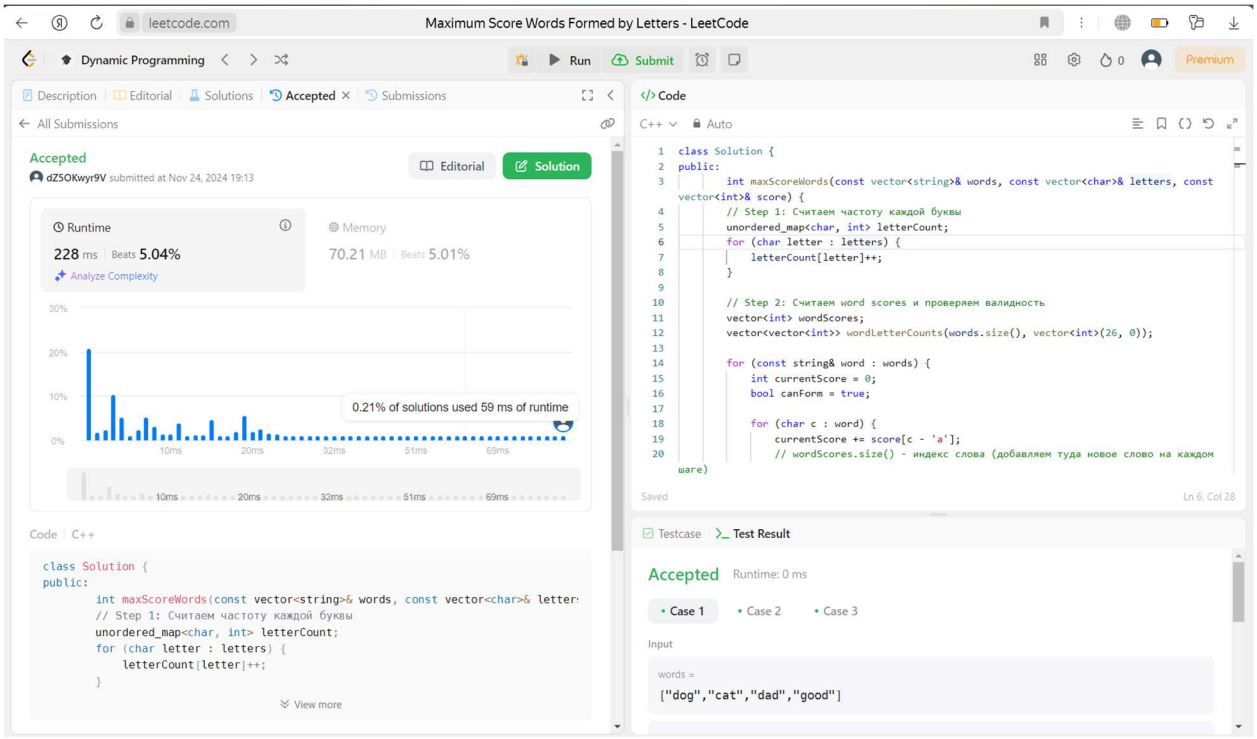
```
        if (validCombination) {
            maxScore = max(maxScore, currentScore);
        }
    }

    return maxScore;

    // Итого: 4*letters.size() + 4*27*words.size() + k + 139 байт, где k - максимальная
длина слова в words
}
```

3.2. Подсчет асимптотики.

```
#include <iostream>
#include <vector>
#include <unordered_map>
#include <string>
#include <algorithm>

using namespace std;

int maxScoreWords(const vector<string>& words, const vector<char>& letters, const
vector<int>& score) {
    // N = words.size(), M = letters.size(), L = score.size(), K - максимальная длина
слова в words
    unordered_map<char, int> letterCount;
    for (char letter : letters) { // O(M)
        letterCount[letter]++;
    }

    vector<int> wordScores;
    vector<vector<int>> wordLetterCounts(words.size(), vector<int>(26, 0));

    for (const string& word : words) { // O(N)
        int currentScore = 0;
        bool canForm = true;

        for (char c : word) { // O(K)
            currentScore += score[c - 'a'];
            wordLetterCounts[wordScores.size()][c - 'a']++;
        }

        for (char c : word) { // O(K)
            if (wordLetterCounts[wordScores.size()][c - 'a'] > letterCount[c]) {
                canForm = false;
                break;
            }
        }

        if (canForm) {
            wordScores.push_back(currentScore);
        }
        else {
            wordScores.push_back(0);
        }
    }

    int maxScore = 0;
    int numWords = wordScores.size();

    for (int mask = 0; mask < (1 << numWords); ++mask) { // O(2^N)
        unordered_map<char, int> usedLetters;
        int currentScore = 0;
```

```
        for (int i = 0; i < numWords; ++i) { // O(N)
            if (mask & (1 << i)) {
                currentScore += wordScores[i];
                for (char c : words[i]) {
                    usedLetters[c]++;
                }
            }
        }

        bool validCombination = true;
        for (const auto& entry : usedLetters) { // O(26)
            if (entry.second > letterCount[entry.first]) {
                validCombination = false;
                break;
            }
        }

        if (validCombination) {
            maxScore = max(maxScore, currentScore);
        }
    }

    return maxScore;

    // Итого: O(M) + O(N*2*K) + O(26*N*2^N) = O(M) + O(N*K) + O(N*2^N) = O(M + N*(K + 2^N)) = O(M + N*2^N) = O(N*2^N)
}
```

3.3. Прохождение тестов на leetcode.



4. Заключение

В ходе выполнения лабораторной работы был реализован алгоритм с использованием ДП для решения задачи на leetcode о нахождении максимального скора слов по буквам.

Задача имеет оптимальную подструктуру, что означает, что оптимальное решение для общей задачи может быть построено на основе оптимальных решений её подзадач. Каждое слово имеет свою оценку, зависящую от букв, которые его составляют. Если мы знаем максимальные оценки для меньших наборов слов и букв, мы можем использовать эти значения для вычисления максимальной оценки для более крупных наборов.

Цель работы была достигнута путем тестирования алгоритма на платформе. Полученные результаты подтвердили теоретические оценки сложности алгоритма и эффективность решения задач динамическим программированием.

5. Приложения

ПРИЛОЖЕНИЕ А

Листинг кода файла lab-6.cpp

```
#include <iostream>
#include <vector>
#include <unordered_map>
#include <string>
#include <algorithm>

using namespace std;

int maxScoreWords(const vector<string>& words, const vector<char>& letters, const
vector<int>& score) {
    // Step 1: Считаем частоту каждой буквы
    unordered_map<char, int> letterCount;
    for (char letter : letters) {
        letterCount[letter]++;
    }

    // Step 2: Считаем word scores и проверяем валидность
    vector<int> wordScores;
    vector<vector<int>> wordLetterCounts(words.size(), vector<int>(26, 0));

    for (const string& word : words) {
        int currentScore = 0;
        bool canForm = true;

        for (char c : word) {
            currentScore += score[c - 'a'];
            // wordScores.size() - индекс слова (добавляем туда новое слово на каждом
шаге)
            wordLetterCounts[wordScores.size()][c - 'a']++;
        }

        // Проверяем можем ли мы составить это слово из наших букв
        for (char c : word) {
            if (wordLetterCounts[wordScores.size()][c - 'a'] > letterCount[c]) {
                canForm = false;
                break;
            }
        }

        if (canForm) {
            wordScores.push_back(currentScore);
        }
        else {
            wordScores.push_back(0); // Для невалидных слов ставим скор 0
        }
    }

    // Step 3: Используем ДП чтобы найти максимальный скор
    int maxScore = 0;
    int numWords = wordScores.size();

    // Есть 2^numWords комбинаций слов, 1-слово в комбинации, 0-нет
    for (int mask = 0; mask < (1 << numWords); ++mask) {
        unordered_map<char, int> usedLetters;
        int currentScore = 0;

        for (int i = 0; i < numWords; ++i) {
```

```
            if (mask & (1 << i)) { // Если i-ое слово в комбинации
                currentScore += wordScores[i];
                for (char c : words[i]) {
                    usedLetters[c]++;
                }
            }
        }

        // Проверяем что эта комбинация валидна по возможным буквам
        bool validCombination = true;
        for (const auto& entry : usedLetters) {
            if (entry.second > letterCount[entry.first]) {
                validCombination = false;
                break;
            }
        }

        if (validCombination) {
            maxScore = max(maxScore, currentScore);
        }
    }

    return maxScore;
}

int main() {
    vector<string> words1 = { "dog", "cat", "dad", "good" };
    vector<char> letters1 = { 'a', 'a', 'c', 'd', 'd', 'd', 'g', 'o', 'o' };
    vector<int> score1 = { 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 };

    cout << "Maximum Score: " << maxScoreWords(words1, letters1, score1) << endl;

    return 0;
}
```