ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 6

«Динамическое программирование»

Выполнила работу

Луценко Елена

Академическая группа J3112

Принято

Дунаев Максим Владимирович

Санкт-Петербург

2024

**Структура отчёта:**

1. Введение

Цель работы: разработать эффективное решение для задачи “Maximum Score Words Formed by Letters” с использованием методов динамического программирования.

Задачи:

1. Проанализировать условие задачи.
2. Выбрать оптимальный алгоритмический подход для решения задачи.
3. Реализовать выбранный алгоритм на языке C++.
4. Оптимизировать решение с учетом ограничений по памяти и времени выполнения.
5. Провести анализ временной и пространственной сложности полученного алгоритма.
6. Теоретическая подготовка
7. Класс:

Объединяет данные (поля, атрибуты) и функции (методы), работающие с этими данными, в одну структуру

Используемые уровни доступа:

* **public** — члены класса доступны извне (для других частей программы).
* **private** — члены класса доступны только внутри самого класса.

1. Функция:

Блок кода, который выполняет определенную задачу, принимает входные параметры (если они есть), выполняет вычисления и, возможно, возвращает результат; функции помогают организовать код, повышая его читаемость и повторное использование.

1. Векторы:

Использование std::vector для динамического хранения слов, букв, количества каждой буквы, скоров.

1. Типы данных:

int: применяется для представления целых значений элементов, используется для индексов массивов и счётчиков элементов;

char: применяется для представления символов;

bool: применяется для обозначения возможности использования слова.

1. Алгоритм:

Полный перебор комбинаций включения слов для нахождения максимального скора (с отсечением неперспективных случаев при использовании всех букв)

1. Реализация (в заданной структуре класса Solution)
2. Реализация метода maxScoreWords (метод для нахождения максимального скора, использующий findScoreWords)

int maxScoreWords(vector<string> &words, vector<char> &letters, vector<int> &score, int k = 26) {  
 vector<int> count(k, 0);  
 for (char symb: letters) {  
 count[symb - 'a']++;  
 }  
 return findScoreWords(words, score, count, 0);  
}

1. Реализация метода findScoreWords (для нахождения максимального скора)

int findScoreWords(vector<string> words, vector<int> score, vector<int> count, int index) {  
 if (index == words.size()) {  
 return 0;  
 }  
 int maxScore = findScoreWords(words, score, count, index + 1);  
 int wordScore = 0;  
 bool canUse = true;  
 vector<int> newCount = count;  
 for (char symb: words[index]) {  
 if (newCount[symb - 'a'] == 0) {  
 canUse = false;  
 break;  
 }  
 newCount[symb - 'a']--;  
 wordScore += score[symb - 'a'];  
 }  
  
 if (canUse) {  
 int scoreWithWord = wordScore + findScoreWords(words, score, newCount, index + 1);  
 maxScore = max(maxScore, scoreWithWord);  
 }  
 return maxScore;  
}

1. Экспериментальная часть

Временная сложность: O (2^n \* l)

Это связано с тем, что для каждого слова мы рассматриваем два варианта (использовать или нет), что дает 2^n возможных комбинаций, и для каждой комбинации мы проходим по буквам слова (l).

Пространственная сложность: O (2^n \* k + n), где n - кол-во слов в words, l - длина наибольшего слова, k = 26 – кол-во букв (из-за рекурсии и создания копий массива количества букв)

(подробный подсчёт приведён в коде в приложении)

1. Заключение

В ходе выполнения работы был реализован алгоритм для решения задачи максимизации счета слов с использованием ограниченного набора букв. Цель работы была достигнута путем применения принципов динамического программирования.

Ключевые аспекты применения динамического программирования в данном алгоритме:

1. Разбиение задачи на подзадачи.

Каждый рекурсивный вызов представляет собой подзадачу выбора оптимального набора слов из оставшихся при проверке возможности включения слова.

1. Оптимальная подструктура.

Решение для всего набора слов строится из оптимальных решений для подмножеств, что реализуется через выбор максимума между использованием текущего слова и его пропуском.

Эффективность алгоритма достигается за счет:

1. Отсечения неперспективных ветвей поиска, когда слово не может быть составлено из доступных букв.
2. Эффективного управления состоянием через использование временных копий вектора доступных букв.
3. Рекурсивного исследования пространства решений, что позволяет избежать полного перебора всех комбинаций.
4. Приложения

ПРИЛОЖЕНИЕ А

Листинг кода max\_score.cpp

class Solution {  
public:  
 int maxScoreWords(vector<string> &words, vector<char> &letters, vector<int> &score, int k = 26) {// временная сложность - O(2^n \* l), пространственная сложность - O(2^n \* k + n), где n - кол-во слов в words, l - длина наибольшего слова  
 vector<int> count(k, 0); // пространственная сложность - O(k)  
 for (char symb: letters) { // временная сложность - O(p), где p - длина letters  
 count[symb - 'a']++;  
 }  
 return findScoreWords(words, score, count, 0); // временная сложность - O(2^n \* l), пространственная сложность - O(2^n \* k + n), где n - кол-во слов в words, l - длина наибольшего слова  
 }  
  
private:  
 int findScoreWords(vector<string> words, vector<int> score, vector<int> count, int index) { // временная сложность - O(2^(n - index) \* l), пространственная сложность - O(2^(n - index) \* k + n - index), где n - кол-во слов в words, l - длина наибольшего слова  
 if (index == words.size()) {  
 return 0;  
 }  
 int maxScore = findScoreWords(words, score, count, index + 1); // временная сложность - O(2^(n - index - 1) \* l), пространственная сложность - O(2^(n - index - 1) \* k + n - index - 1), где n - кол-во слов в words, l - длина наибольшего слова  
 int wordScore = 0;  
 bool canUse = true;  
 vector<int> newCount = count; // пространственная сложность - O(k)  
 for (char symb: words[index]) { // временная сложность - O(l)  
 if (newCount[symb - 'a'] == 0) {  
 canUse = false;  
 break;  
 }  
 newCount[symb - 'a']--;  
 wordScore += score[symb - 'a'];  
 }  
  
 if (canUse) { // временная сложность - O(2^(n - index - 1) \* l), пространственная сложность - O(2^(n - index - 1) \* k + n - index - 1), где n - кол-во слов в words, l - длина наибольшего слова  
 int scoreWithWord = wordScore + findScoreWords(words, score, newCount, index + 1);  
 maxScore = max(maxScore, scoreWithWord);  
 }  
 return maxScore;  
 }  
};

ПРИЛОЖЕНИЕ B

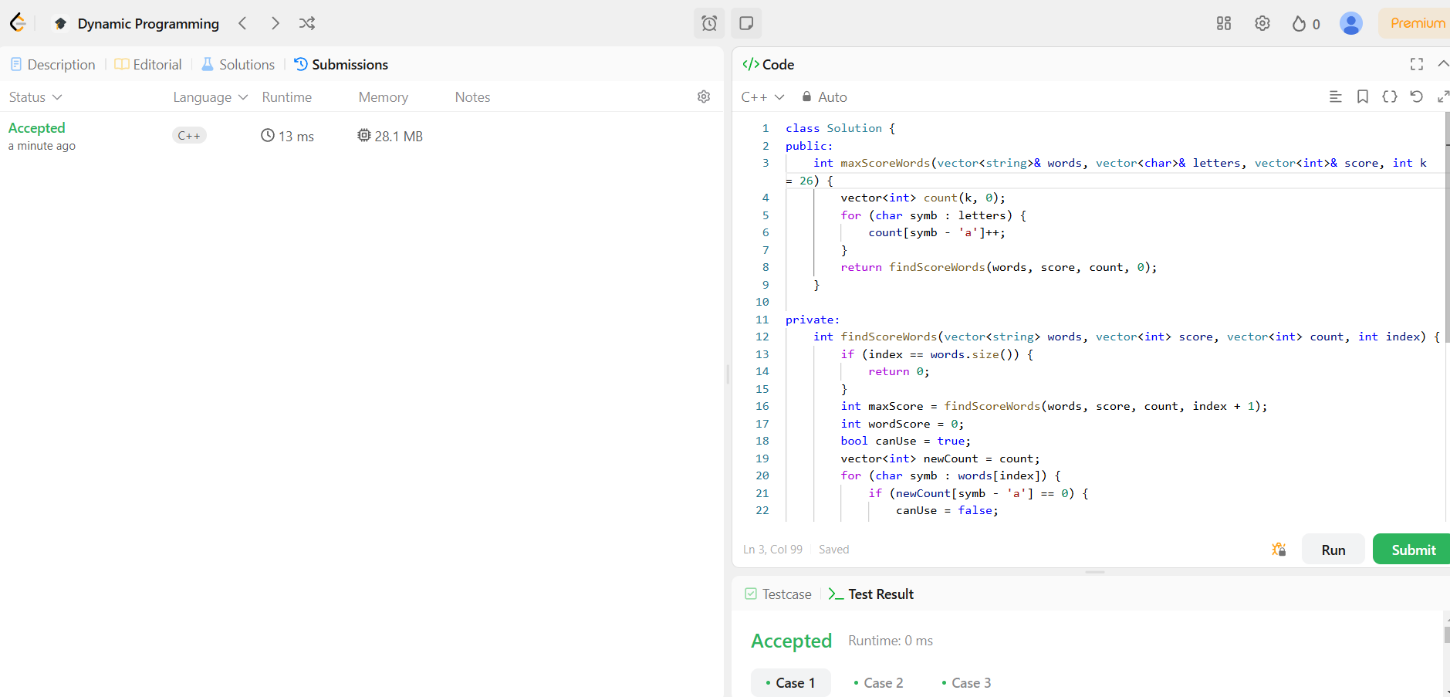


Рис 1. Задача прошла все тесты