ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 6

«1402. Reducing dishes»

Выполнил работу

Бровкин Аким

J3110

Принято

Ментор, Вершинин Владислав

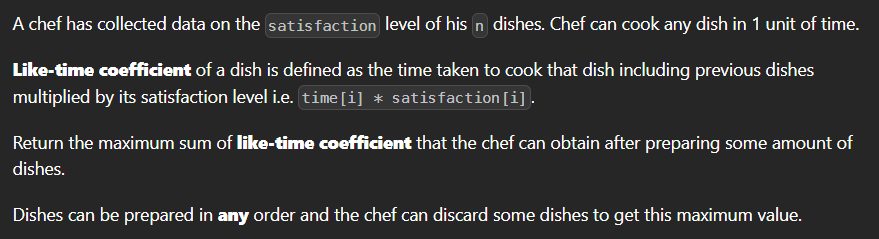
Санкт-Петербург

2024

**Структура отчёта:**

1. Введение

Цель работы: вычислить максимальную сумму коэффициента «нравится - не нравится», которую может получить повар после приготовления некоторого количества блюд.



Задачи:

— Решить задачу используя динамическое программирование.

— Проанализировать затрачиваемую память и время работы программы.

— Объяснить, зачем здесь используется динамическое программирование.

1. Теоретическая подготовка

Использовались следующие типы данныx:

int - базовый целочисленный тип по умолчанию

vector<int> - динамический массив, обеспечивающий быстрое добавление новых элементов в конец и меняющий свой размер при необходимости

Также была использована рекурсия в функции solve, мемоизация, сортировка, жадное распределение времени через произведение

1. Реализация

­- Проанализированы ограничения: массив sat может содержать как положительные, так и отрицательные значения

- Определены ключевые параметры:

Индексы блюд;

Текущее время, влияющее на «вес» каждого блюда;

Возможность отказаться от готовки блюда.

Задачу можно было решить полным перебором, но для ускорения предложена оптимизация через мемоизацию, чтобы хранить результаты вычислений для одинаковых входных данных.

Учитывая, что важность блюда возрастает с увеличением времени, добавлен этап сортировки.

Рекурсия:

int take = sat[index] \* ctime + solve(index + 1, ctime + 1, sat, mem);

int notTake = solve(index + 1, ctime, sat, mem);

Мемоизация:

vector<vector<int>> mem(n, vector<int>(n, -1));

if(mem[index][ctime - 1] != -1)

return mem[index][ctime - 1];

Написан базовый скелет класса Solution.

Реализована рекурсивная функция solve с двумя вариантами выбора: "взять" или "не взять" блюдо.

Добавлена мемоизация для ускорения повторяющихся вычислений.

Перед вызовом рекурсии добавлена сортировка массива.

Сортировка:

sort(sat.begin(), sat.end());

Рекурсивный вызов:

return solve(0, 1, sat, mem);

1. Экспериментальная часть

В данной задаче было использовано динамическое программирование, потому что при больших значениях n без оптимизации потребуется 2^n вариантов перебора, а считаться это будет очень долго. Динамическое программирование позволяет устранить лишние повторяющиеся вычисления за счет хранения промежуточных результатов. Это применяется в следующих местах:

Если текущий индекс блюда и текущее время одинаковы, результат не зависит от предыдущих шагов — достаточно один раз вычислить ответ. (Вычисления для (index = 2, ctime = 3) будут идентичны при любом предыдущем пути).

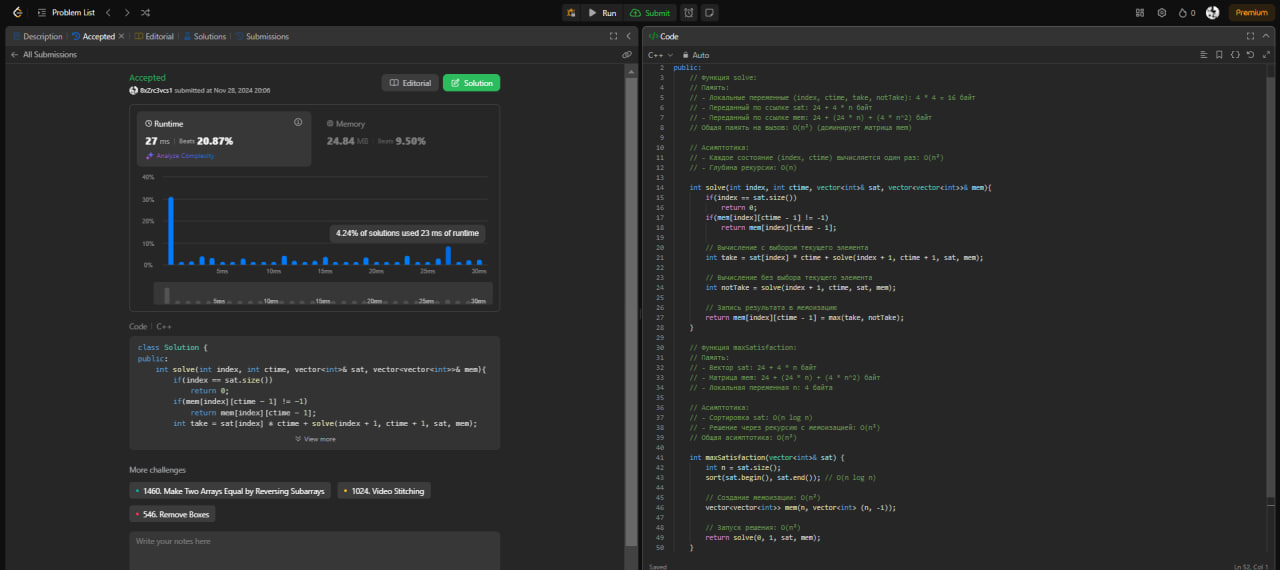
В коде используется таблица mem для хранения промежуточных результатов.

Перед рекурсивным вызовом проверяется, рассчитан ли уже результат для данной подзадачи. Если рассчитан, он просто возвращается:

if (mem[index][ctime - 1] != -1)

return mem[index][ctime - 1];

1. Заключение



1. Приложения

ПРИЛОЖЕНИЕ А

Листинг кода

class Solution {

public:

    // Функция solve:

    // Память:

    // - Локальные переменные (index, ctime, take, notTake): 4 \* 4 = 16 байт

    // - Переданный по ссылке sat: 24 + 4 \* n байт

    // - Переданный по ссылке mem: 24 + (24 \* n) + (4 \* n^2) байт

    // Общая память на вызов: O(n²) (доминирует матрица mem)

    // Асимптотика:

    // - Каждое состояние (index, ctime) вычисляется один раз: O(n²)

    // - Глубина рекурсии: O(n)

    int solve(int index, int ctime, vector<int>& sat, vector<vector<int>>& mem){

        if(index == sat.size())

            return 0;

        if(mem[index][ctime - 1] != -1)

            return mem[index][ctime - 1];

        // Вычисление с выбором текущего элемента

        int take = sat[index] \* ctime + solve(index + 1, ctime + 1, sat, mem);

        // Вычисление без выбора текущего элемента

        int notTake = solve(index + 1, ctime, sat, mem);

        // Запись результата в мемоизацию

        return mem[index][ctime - 1] = max(take, notTake);

    }

    // Функция maxSatisfaction:

    // Память:

    // - Вектор sat: 24 + 4 \* n байт

    // - Матрица mem: 24 + (24 \* n) + (4 \* n^2) байт

    // - Локальная переменная n: 4 байта

    // Асимптотика:

    // - Сортировка sat: O(n log n)

    // - Решение через рекурсию с мемоизацией: O(n²)

    // Общая асимптотика: O(n²)

    int maxSatisfaction(vector<int>& sat) {

        int n = sat.size();

        sort(sat.begin(), sat.end()); // O(n log n)

        // Создание мемоизации: O(n²)

        vector<vector<int>> mem(n, vector<int> (n, -1));

        // Запуск решения: O(n²)

        return solve(0, 1, sat, mem);

    }

};