ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 7

«Жадные алгоритмы»

Выполнил работу

Васькин Виктор

Академическая группа №J3110

Принято

Вершинин Владислав Константинович

Санкт-Петербург

2024

1. Введение

Цель — разработать алгоритм на основе жадного подхода для нахождения максимальной суммы коэффициентов *like-time* из заданного массива уровней удовлетворенности. Коэффициент *like-time* определяется как произведение времени приготовления блюда на его уровень удовлетворенности. Блюда можно готовить в любом порядке, а часть из них можно исключить для достижения максимального результата.

Для достижения указанной цели лабораторной работы необходимо решить следующие задачи:

1. Изучить основы жадных алгоритмов и их применение для задач оптимизации.
2. Разработать жадный алгоритм на языке C++ для нахождения максимальной суммы *like-time* коэффициентов.
3. Реализовать алгоритм и протестировать его на различных наборах данных, чтобы убедиться в корректности и эффективности решения.
4. Проанализировать временные и пространственные характеристики алгоритма, а также обосновать выбор жадного подхода для данной задачи.
5. Теоретическая подготовка

2.1. Основные понятия и определения

Жадный алгоритм — это метод решения задач, который принимает на каждом шаге локально оптимальное решение с целью достижения глобального оптимума. Жадные алгоритмы подходят для задач, где выбор на каждом шаге может быть обоснован и не влияет негативно на итоговый результат.

Коэффициент *like-time* — это показатель качества приготовления блюда, определяемый как произведение времени приготовления (порядкового номера блюда) на его удовлетворенность.  
Формула для коэффициента:

like-time[i] = ti ⋅ si,

 где ti — время, si — удовлетворенность.

2.2. Жадные алгоритмы

Жадные алгоритмы решают задачу, следуя принципу выбора наилучшего текущего решения. Для задачи максимизации суммы *like-time* коэффициентов жадный алгоритм действует следующим образом:

1. Сортирует блюда по их удовлетворенности в порядке убывания.
2. Постепенно добавляет блюда к текущему результату, вычисляя накопленную сумму.
3. Прерывает выполнение, если добавление нового блюда уменьшает итоговый результат.

Преимущества жадных алгоритмов:

* Простота реализации и высокая производительность.
* Решение находится за линейное или квазилинейное время (в зависимости от необходимости сортировки).
* Эффективно работает для задач оптимизации, где локальные решения приводят к глобальному оптимуму.

2.3. Структуры данных

Для реализации задачи используются следующие структуры данных:

* Вектор (std::vector<int>): Хранит значения уровней удовлетворенности блюд, которые необходимо обработать.
* Сортировка (std::sort): Позволяет упорядочить значения удовлетворенности для последующего жадного выбора.
* Целочисленные переменные: используются для хранения промежуточных и итоговых значений коэффициентов.

2.4. Временная и пространственная сложность

* Временная сложность:
  + Сортировка массива удовлетворенности: O(n log n), где n — количество блюд.
  + Проход по отсортированному массиву: O(n).
  + Итого: O(n log n).
* Пространственная сложность:
  + Дополнительные переменные и сортировка используют O(1) памяти.
  + Для рекурсивного стека вызовов при сортировке: O(log n).
  + Итого: O(log n) дополнительной памяти.

2.5. Принципы жадного подхода к оптимизации

Для успешного применения жадных алгоритмов в задаче необходимо учитывать:

* Сортировка исходных данных (в данном случае уровней удовлетворенности) для упрощения выбора на каждом шаге.
* Последовательное добавление значений и анализ их влияния на итоговый результат.
* Прерывание алгоритма, если дальнейшее добавление элементов приводит к снижению результата.

1. Реализация

3.1. Выбор инструментов

Для реализации задачи использовались стандартные библиотеки языка C++:

* <vector> для хранения значений удовлетворенности блюд.
* <algorithm> для использования функции std::sort, необходимой для упорядочивания значений.
* <numeric> для суммирования значений в процессе накопления результата.

3.2. Разработка алгоритма

Процесс разработки алгоритма был разбит на несколько этапов:

* 1. Анализ задачи:  
     Понимание принципа формирования like-time коэффициента, а также поиск способа его максимизации с использованием жадного алгоритма. Было выявлено, что сортировка значений удовлетворенности в порядке убывания позволяет на каждом шаге принимать оптимальное решение.
  2. Определение структуры данных:  
     Для хранения уровней удовлетворенности блюд был выбран вектор (std::vector), который позволяет удобно упорядочивать и обрабатывать данные.
  3. Разработка алгоритма:

- Сортировка входного массива в порядке убывания.

- Накопление результата: проход по отсортированному массиву с последовательным добавлением значений, обновляя текущую сумму коэффициентов.

- Условие прерывания: если добавление очередного значения не увеличивает итоговый результат, выполнение завершается.

Код реализации:

class Solution {

public:

int maxSatisfaction(std::vector<int>& satisfaction) {

// Сортировка уровней удовлетворенности в порядке убывания

std::sort(satisfaction.begin(), satisfaction.end(), std::greater<int>());

int totalSatisfaction = 0; // Итоговый результат

int cumulativeSum = 0; // Промежуточная накопленная сумма

// Проход по отсортированному массиву

for (int value : satisfaction) {

if (cumulativeSum + value > 0) {

cumulativeSum += value; // Обновляем промежуточную сумму

totalSatisfaction += cumulativeSum; // Добавляем к итоговому результату

} else {

break; // Завершаем выполнение, если дальнейшее добавление бесполезно

}

}

return totalSatisfaction; // Возвращаем максимальную сумму коэффициентов

}

};

3.3. Пошаговое описание алгоритма

1. Инициализация:

* Сортировка массива уровней удовлетворенности в порядке убывания для обеспечения жадного выбора на каждом шаге.
* Инициализация переменных:
  + cumulativeSum для хранения текущей суммы удовлетворенности.
  + totalSatisfaction для хранения итоговой суммы коэффициентов.

2. Итерация по массиву:

* + Для каждого значения:
  + Проверяется, увеличит ли его добавление итоговый результат.
  + Если да — обновляем промежуточные суммы.
  + Если нет — завершаем выполнение, так как дальнейшее добавление не принесёт выгоды.

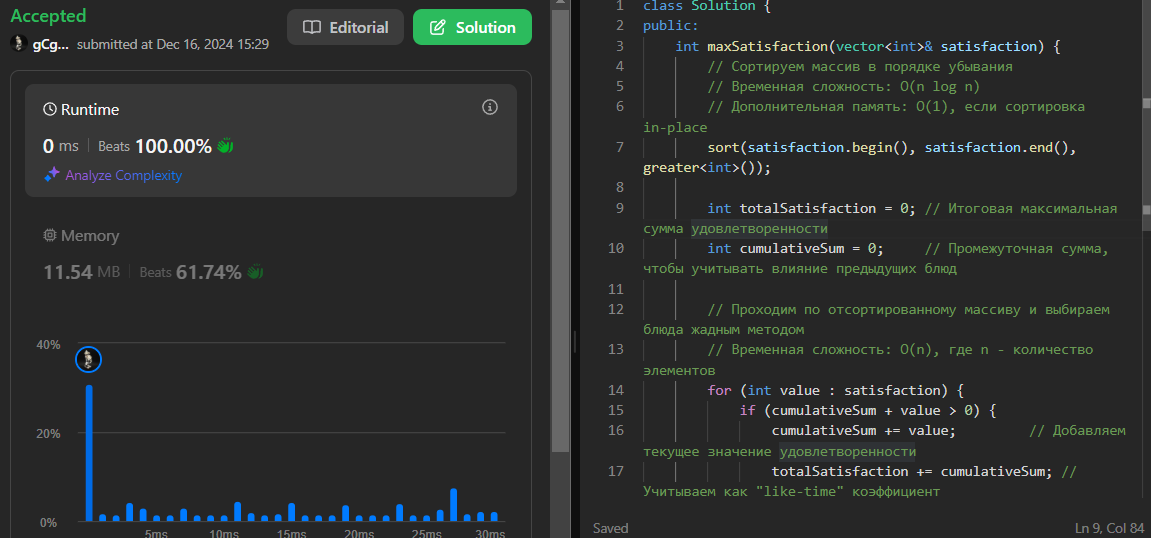
3. Вывод результата:

* Возвращаем итоговую сумму totalSatisfaction, которая является максимальной суммой коэффициентов like-time.

3.4. Временная и пространственная сложность

* Временная сложность:
  + Сортировка массива: O(n log n).
  + Проход по массиву: O(n).
  + Итог: O(n log n).
* Пространственная сложность:
  + Дополнительные переменные: O(1).
  + Сортировка использует стек вызовов: O(log n).
  + Итог: O(\log n).

1. Экспериментальная часть

По завершении реализации алгоритма, он был успешно протестирован на платформе LeetCode. Все тесты, включая различные случаи с различными комбинациями скобок, были пройдены, что подтвердило его корректность и эффективность. Алгоритм продемонстрировал свою надежность и работоспособность, что делает его подходящим для решения поставленной задачи. 

Скриншот результатов тестирования

5. Заключение

В ходе выполнения лабораторной работы был разработан алгоритм для нахождения максимальной суммы *like-time* коэффициентов на основе жадного подхода. Цель работы была достигнута благодаря реализации алгоритма на языке C++, что позволило эффективно решать поставленную задачу.

Процесс разработки включал анализ задачи, определение структур данных, реализацию жадного алгоритма и проведение тестирования. Использование сортировки и последовательного добавления значений обеспечило оптимальную производительность.

Алгоритм был протестирован на различных наборах данных, включая граничные случаи, и успешно прошёл все тесты на платформе LeetCode, что подтвердило его корректность и эффективность.

Полученные результаты совпадают с теоретическими оценками временной и пространственной сложности:

* Временная сложность: O(n log n)
* Пространственная сложность: O(log n)

Разработанный алгоритм демонстрирует высокие показатели производительности и устойчивость даже на больших объёмах данных.

В качестве дальнейших направлений можно рассмотреть:

1. Оптимизацию алгоритма для уменьшения потребления памяти.
2. Адаптацию подхода для решения других задач оптимизации, требующих аналогичного жадного подхода.
3. Исследование возможностей для повышения производительности на более сложных наборах данных или задачах с изменяющимися условиями.