# ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 7

«Жадные алгоритмы»

Выполнил работу Мельников Олег Академическая группа №J3112

Принято Дунаев Максим Владимирович

Санкт-Петербург 2024

# ВВЕДЕНИЕ

**Цель работы**: разработать и реализовать жадный алгоритм на языке программирования C++ для решения задачи минимизации отклонения в массиве.

**Задачи:**

* Изучить условия задачи;
* Разработать алгоритм на основе динамического алгоритма;
* Проанализировать результаты, оценить эффективность.

# ТЕОРЕТИЧЕСКАЯ ПОДГОТОВКА

1. **Локально оптимальное решение**:
   * На каждом шаге мы стараемся уменьшить максимальное значение в массиве. Уменьшение максимального числа — это естественный способ снизить отклонение от минимума.
   * Этот подход жадного алгоритма локально оптимален, так как всегда приводит к уменьшению разницы между максимальным и минимальным значением.
2. **Работа с приоритетами**:
   * Мы используем максимальную кучу (maxHeap), чтобы всегда иметь возможность быстро извлекать текущее максимальное число. Это позволяет эффективно управлять максимальными значениями массива на каждом шаге.
   * Жадный алгоритм применяет это приоритетное извлечение для того, чтобы сосредоточиться на основных (максимальных) значениях, что и позволяет уменьшить отклонение.
3. **Эффективная реализация**:
   * Вместо полного перебора возможных комбинаций элементов (что было бы вычислительно дорого), на каждом шаге алгоритм выбирает текущее максимальное значение и операциями над ним стремится минимизировать отклонение.
4. **Прекращение обработки**:
   * Если на очередном шаге оказались максимальные нечетные числа, которые нельзя уменьшить, алгоритм завершает свою работу, так как дальнейшее снижение отклонения невозможно. Это также характеристика жадного подхода.

# РЕАЛИЗАЦИЯ

**Преобразование массива**:

Каждый элемент массива проверяется на четность. Нечетные числа умножаются на 2 с целью преобразования их в четные. Это позволяет обеспечить, что с каждым числом могут выполняться операции деления на 2, что будет способствовать уменьшению максимального элемента.

Таким образом, мы начинаем с массива, содержащего только четные числа или преобразованные нечетные числа.

**Использование приоритетной очереди (maxHeap)**:

В алгоритме используется максимальная куча для хранения всех преобразованных чисел. Это позволяет быстро получать и обрабатывать максимальное значение из массива в каждой итерации.

Каждый раз, когда мы извлекаем максимальное число, мы пытаемся уменьшить его, деля на 2. Это жадное решение, так как мы находимся на максимальном значении, и, чтобы минимизировать отклонение, нам необходимо работать с ним в первую очередь.

**Поддержка минимального значения**:

Во время обработки максимального элемента также отслеживается минимальное значение. На каждой итерации мы обновляем минимальное отклонение между текущим максимальным и минимальным значениями.

Если максимальное значение нечетное, мы прекращаем дальнейшие операции, поскольку уже не можем больше его уменьшать. Этот шаг также соответствует жадному подходу. Мы идем по наиболее важному пути — уменьшаем максимальное значение, пока это возможно.

Таким образом, жадный алгоритм в данной задаче использует стратегию брать самое большое (максимальное число), чтобы достигать минимума разности (отклонения) с текущим минимальным значением.

# ЭКСПЕРИМЕНТАЛЬНАЯ ЧАСТЬ

Подсчет по памяти:

1. Память для входного массива: (O(n))
2. Память для maxHeap: (O(n))
3. Временные переменные: (O(1))

В итоге, общее использование памяти алгоритма будет равно: O(n) + O(n) + O(1) = O(n)

Подсчет сложности алгоритма:

1. **Инициализация maxHeap**:
   * При первом проходе по массиву nums, каждый элемент проверяется, и нечетные числа преобразуются в четные. Затем все числа помещаются в максимальную кучу (maxHeap). Временная сложность этой операции составляет: O(n log n)
   * Ограничение (O(n)) связано с проходом по всем (n) элементам, а (O(log n)) — со вставкой в кучу.
2. **Основной цикл**:
   * В основном цикле происходит извлечение максимального элемента из кучи и обновление минимального значения. Этот цикл может выполняться максимум (n) раз в том случае, если все числа четные и регулируются работой алгоритма. Поэтому, временная сложность составит: O(n log n)
   * Поскольку в каждой итерации максимальное число удаляется и обратно добавляется уже уменьшенное значение, операции добавления и извлечения из maxHeap все еще требуют (O(log n)).

Итоговая сложность

Оценив обе части, мы можем обобщить сложность: O(n log n) + O(n log n) = O(n log n)

# ЗАКЛЮЧЕНИЕ

В ходе выполнения работы была достигнута поставленная цель – разработан и реализован жадный алгоритм. Полученное решение соответствует теоретическим оценкам по времени (O(n log n)) и памяти (O(n)) и успешно прошло проверку на тестовых примерах, продемонстрировав корректность и эффективность.

# ПРИЛОЖЕНИЕ

Листинг кода

#include <iostream>

#include <vector>

#include <queue>

#include <algorithm>

using namespace std;

class Solution {

public:

    int minimumDeviation(vector<int>& nums) {

        priority\_queue<int> maxHeap;

        int minNum = INT\_MAX;

        for (int num : nums) {

            if (num % 2 == 1) {

                num \*= 2; // Преобразуем нечетные числа в четные

            }

            maxHeap.push(num);

            minNum = min(minNum, num);

        }

        int minDeviation = INT\_MAX; // Инициализируем для хранения минимального отклонения

        // Продолжаем процесс, пока можем уменьшить максимальный элемент

        while (!maxHeap.empty()) {

            int maxNum = maxHeap.top();

            maxHeap.pop();

            // Рассчитываем текущее отклонение

            minDeviation = min(minDeviation, maxNum - minNum);

            if (maxNum % 2 == 0) {

                maxNum /= 2;

                maxHeap.push(maxNum); // Возвращаем уменьшенное число обратно в кучу

                minNum = min(minNum, maxNum);

            } else {

                // Если текущий maximum нечетный, останавливаем обработку

                break;

            }

        }

        return minDeviation;

    }

};