ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 7

«Жадные алгоритмы»

Выполнила работу

Шакина Анна

Академическая группа № J3110

Принято

Вершинин Владислав Константинович

Санкт-Петербург

2024

1. Введение

Цель: решить задачу с использованием жадного алгоритма

Задачи:

1. Понять, как и почему в данном случае требуется применение жадного алгоритма.
2. Написать код.
3. Посчитать количество памяти, требуемое для выполнения кода.
4. Посчитать асимптотику.
5. Реализация
   1. Используемые библиотеки

* <iostream> - для использования ввода и вывода
* <vector> - для использования std::vector
  1. Условие задачи

Дан отсортированный массив целых чисел nums и целое число n, вставьте элементы в массив таким образом, чтобы любое число в диапазоне [1, n] включительно могло быть представлено в виде суммы некоторых элементов в массиве. Ответ – минимальное количество необходимых вставок.

* 1. Алгоритм решения задачи

Жадный алгоритм работает, выбирая на каждом шаге наилучший (локально оптимальный) вариант. В данном случае, если текущая сумма sum меньше n, мы пытаемся добавить минимальное число, которое позволит нам расширить диапазон достижимых сумм. Если следующее число в массиве больше sum + 1, это означает, что мы не можем достигнуть sum + 1, и нам нужно вставить это значение, чтобы продолжить.

Жадный алгоритм позволяет эффективно увеличивать сумму, добавляя наименьшие возможные значения (т.е. sum + 1). Это позволяет минимизировать количество вставок, так как каждое добавление нового элемента максимально увеличивает диапазон достижимых сумм. Переборное решение данной задачи будет проверять все возможные комбинации вставок, что приводит к экспоненциальному росту числа вариантов по мере увеличения размера массива и значения n.

1. Нам нужно убедиться, что мы можем составить все суммы от 1 до n, используя заданный массив чисел nums. Если какой-либо суммы не хватает, мы должны "вставить" число, чтобы её достичь.
2. sum – текущая максимальная сумма, которую мы можем составить, ans – количество вставок. Мы проходим по массиву nums, начиная с наименьшей суммы (1) и пытаемся использовать числа из массива для достижения этой суммы. Если текущее число в nums больше sum + 1, это означает, что между sum и этим числом есть промежуток. Мы не можем достичь суммы sum + 1, поэтому мы должны вставить это значение. Увеличиваем ans и sum.
3. Если текущее число в nums меньше или равно sum + 1, мы можем использовать его для увеличения нашей максимальной достижимой суммы. Мы добавляем это число к sum и переходим к следующему числу в массиве.
4. После того как мы обработали все числа в массиве, если sum все еще меньше n, нам нужно продолжать вставлять числа, начиная с sum + 1, пока не достигнем n.
5. В конце возвращаем количество вставок, которое мы сделали.
6. Экспериментальная часть
   1. Подсчёт по памяти

• long long sum: 8 байт (для хранения больших чисел).

• int i: 4 байта (размер целого числа).

• int size: 4 байта (размер вектора).

• int ans: 4 байта (счетчик количества вставок).

Общее использование памяти = 8 + 4 + 4 + 4 = 20 байт

* 1. Подсчёт асимптотики

Цикл while (sum < n && i < size):

В этом цикле мы проходим по элементам массива nums. Если nums[i] > sum + 1, мы добавляем значение sum + 1 и увеличиваем ans. В противном случае, мы просто добавляем текущий элемент массива к sum.

В худшем случае, если все элементы массива могут быть использованы, этот цикл выполняется O(M) раз, где M — размер вектора nums.

Цикл while (sum < N):

Этот цикл добавляет значения до тех пор, пока sum не станет больше или равен N. Каждый раз, когда мы добавляем sum + 1, значение sum удваивается.

Поскольку каждый шаг увеличивает sum значительно, количество итераций этого цикла будет O(log N), так как каждый шаг удваивает текущее значение sum.

Таким образом, общая асимптотическая сложность будет O(M + log N).

1. Заключение

В ходе выполнения работы был реализован алгоритм кластеризации массива на K кластеров посредством полного перебора всех комбинаций значений этого массива. Цель работы была достигнута с помощью перебора битовых масок всех возможных разбиений массива на кластеры. Алгоритм был протестирован на наборах данных различного размера, и его результаты соответствуют теоретической оценке сложности. В качестве дальнейших исследований можно предложить оптимизацию алгоритма, а также исследование зависимости времени выполнения не только от количества входных данных, но и от размера данных, их расположения во входном массиве.

1. Приложения

В ходе работы над задачей была успешно решена проблема минимизации количества вставок в отсортированном массиве целых чисел для представления всех чисел в диапазоне от 1 до n*n* в виде суммы элементов массива. Для решения был применён жадный алгоритм, который на каждом этапе выбирает локально оптимальное решение, обеспечивая тем самым глобальную оптимальность результата.

ПРИЛОЖЕНИЕ А

Листинг кода файла patching\_array.cpp

#include <iostream>

#include <vector>

int **minPatches**(std::**vector**<int> &nums, int n)

{

    long long sum = 0;

    int i = 0, size = nums.**size**(), ans = 0;

    while (sum < n && i < size)

    {

        if (nums**[**i**]** > sum + 1)

        {

            sum += sum + 1;

            ans++;

        }

        else

        {

            sum += nums**[**i**]**;

            i++;

        }

    }

    while (sum < n)

    {

        sum += sum + 1;

        ans++;

    }

    return ans;

}

int **main**()

{

    std::**vector**<int> nums = {1, 5, 10};

    int n = 20;

    int result = **minPatches**(nums, n);

    std::cout **<<** "Минимальное количество: " **<<** result **<<** std::**endl**;

    return 0;

}