ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 7

«Выполнения задания «[Candy](https://leetcode.com/problems/tallest-billboard/)» с помощью динамического программирования»

Выполнила работу

Зарайская Яна

Академическая группа №J3110

Принято

Вершинин Владислав Константинович

Санкт-Петербург

2024

1. Введение

Цель работы: вычислить минимальное количество конфет, необходимое, чтобы раздать их некоторому количеству детей.

Задачи:

* Реализовать алгоритм вычисления минимального количества конфет с помощью жадного алгоритма
* Оценить пространственную и временную сложность алгоритма

1. Теоретическая подготовка
   1. Жадный алгоритм:

* Алгоритм, заключающийся в принятии локально [оптимальных решений](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D1%82%D0%B8%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%80%D0%B5%D1%88%D0%B5%D0%BD%D0%B8%D0%B5) на каждом этапе, допуская, что конечное решение также окажется оптимальным.
  1. Векторы:
* Использование std::vector для динамического хранения элементов массива.
  1. Числовые данные:
* int: используется для индексов массивов и счётчиков элементов, что позволяет эффективно управлять позициями, а также для представления значений элементов

1. Реализация
   1. Реализованная функция

int candy(vector<int>& ratings) {

int n = ratings.size(); // O(1)

int totalCandies = n; // O(1)

int i = 1; // O(1)

//Проходит по всем элементам вектора ratings

while (i < n) { // O(n)

//Если текущий рейтинг равен предыдущему, просто переходим к следующему элементу

if (ratings[i] == ratings[i - 1]) { // O(1)

i++; // O(1)

continue; // O(1)

}

//Если текущий рейтинг больше предыдущего, начинаем обработку "пика". Увеличиваем currentPeak и добавляем его к totalCandies

int currentPeak = 0; // O(1)

while (i < n && ratings[i] > ratings[i - 1]) { // O(n)

currentPeak++; // O(1)

totalCandies += currentPeak; // O(1)

i++; // O(1)

}

//Если достигли конца вектора, возвращаем totalCandies

if (i == n) { // O(1)

return totalCandies; // O(1)

}

//Если текущий рейтинг меньше предыдущего, начинаем обработку "долины". Увеличиваем currentValley и добавляем его к totalCandies

int currentValley = 0; // O(1)

while (i < n && ratings[i] < ratings[i - 1]) { // O(n)

currentValley++; // O(1)

totalCandies += currentValley; // O(1)

i++; // O(1)

}

//Вычитаем минимальное значение между currentPeak и currentValley из totalCandies, чтобы минимизировать общее количество конфет

totalCandies -= min(currentPeak, currentValley); // O(1)

}

return totalCandies; // O(1)

}

Функция для вычисления минимального количества конфет, необходимого, чтобы раздать их некоторому количеству детей

* 1. Основные действия:
* **Проход по вектору**:

Начинаем цикл, который проходит по всем элементам вектора ratings, начиная с индекса 1.

* Обработка равных рейтингов

Если текущий рейтинг равен предыдущему, просто переходим к следующему элементу.

* Обработка "пика"

Если текущий рейтинг больше предыдущего, начинаем обработку "пика".

Увеличиваем currentPeak и добавляем его к totalCandies для каждого элемента в возрастающей последовательности.

* Проверка на конец вектора

Если достигли конца вектора, возвращаем totalCandies.

* Обработка "долины"

Если текущий рейтинг меньше предыдущего, начинаем обработку "долины".

Увеличиваем currentValley и добавляем его к totalCandies для каждого элемента в убывающей последовательности.

* Корректировка конфет

Вычитаем минимальное значение между currentPeak и currentValley из totalCandies, чтобы минимизировать общее количество конфет.

* Возврат результата

Возвращаем общее количество конфет после завершения всех итераций.

* 1. Используемые библиотеки:
* <iostream>: Для ввода и вывода.
* <vector>: Для использования динамических массивов (векторов).

1. Подсчет пространственной и временной сложности алгоритма
   1. Подсчет временной сложности алгоритма:

* Основной цикл проходит по всем элементам вектора **ratings**, что дает временную сложность O(n).
* Внутри основного цикла есть два вложенных цикла, которые также проходят по элементам вектора, но каждый элемент обрабатывается только один раз. Таким образом, общая временная сложность остается O(n).
  1. Подсчет пространственной сложности алгоритма:
* Алгоритм использует несколько переменных (n, totalCandies, i, currentPeak, currentValley), которые занимают постоянное количество памяти.
* Таким образом, пространственная сложность алгоритма составляет O(1).

1. Пояснения к задаче

В данной задаче используется жадным алгоритм для эффективного вычисления минимального количества конфет.

* Локальные решения:

Алгоритм проходит по массиву рейтингов и принимает решения на основе текущего состояния. Он ищет "пики" (возрастающие последовательности) и "долины" (убывающие последовательности) и принимает решения о количестве конфет для каждого ребенка на основе этих локальных структур.

* **Однопроходный алгоритм**:

Этот алгоритм проходит по массиву один раз. Это характерно для жадных алгоритмов, которые строят решение по частям, принимая локально оптимальные решения.

* **Минимизация конфет**:

Алгоритм минимизирует количество конфет, необходимых для удовлетворения условий задачи, путем вычитания минимального значения между "пиком" и "долиной". Это локально оптимальное решение, которое помогает минимизировать общее количество конфет.

ПРИЛОЖЕНИЕ А

Листинг кода

#include <iostream>

#include <vector>

using namespace std;

int candy(vector<int>& ratings) {

int n = ratings.size(); // O(1)

int totalCandies = n; // O(1)

int i = 1; // O(1)

//Проходит по всем элементам вектора ratings

while (i < n) { // O(n)

//Если текущий рейтинг равен предыдущему, просто переходим к следующему элементу

if (ratings[i] == ratings[i - 1]) { // O(1)

i++; // O(1)

continue; // O(1)

}

//Если текущий рейтинг больше предыдущего, начинаем обработку "пика". Увеличиваем currentPeak и добавляем его к totalCandies

int currentPeak = 0; // O(1)

while (i < n && ratings[i] > ratings[i - 1]) { // O(n)

currentPeak++; // O(1)

totalCandies += currentPeak; // O(1)

i++; // O(1)

}

//Если достигли конца вектора, возвращаем totalCandies

if (i == n) { // O(1)

return totalCandies; // O(1)

}

//Если текущий рейтинг меньше предыдущего, начинаем обработку "долины". Увеличиваем currentValley и добавляем его к totalCandies

int currentValley = 0; // O(1)

while (i < n && ratings[i] < ratings[i - 1]) { // O(n)

currentValley++; // O(1)

totalCandies += currentValley; // O(1)

i++; // O(1)

}

//Вычитаем минимальное значение между currentPeak и currentValley из totalCandies, чтобы минимизировать общее количество конфет

totalCandies -= min(currentPeak, currentValley); // O(1)

}

return totalCandies; // O(1)

}

int main() {

vector<int> test1 = { 1,2,2 };

cout << candy(test1);

}

ПРИЛОЖЕНИЕ Б

Подтверждение того, что задача прошла все тесты

