ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 6

«Динамическое программирование»

Выполнил работу

Баранов Владимир

Академическая группа №J3112

Принято

Дунаев Максим Владимирович

Санкт-Петербург

2024

1. ВВЕДЕНИЕ

Цель работы: разработать и реализовать алгоритм на языке программирования C++ для решения задачи максимизации прибыли при торговле акциями с ограниченным количеством транзакций, используя методы динамического программирования.

Задача заключается в том, чтобы определить максимальную возможную прибыль от покупки и продажи акций, совершив не более k транзакций, где каждая транзакция состоит из одной покупки и одной продажи.

Задачи:

- Изучить условия задачи, определить ограничения и требования;
- Разработать алгоритм на основе динамического алгоритма;
- Проанализировать результаты, оценить эффективность и сделать выводы.

2. ТЕОРЕТИЧЕСКАЯ ПОДГОТОВКА

Задача заключается в следующем: имеется массив цен акций prices, где prices[i] — это цена акции в день i. Необходимо определить максимальную возможную прибыль, которую можно получить, совершив не более k транзакций. Каждая транзакция состоит из одной покупки и одной продажи акции. При этом не допускается одновременное владение более чем одной акцией — следующую акцию можно купить только после продажи текущей.

Ограничения:

- 1 <= k <= 100;
- 1 <= prices.size() <= 1000
- 0 <= pices[i] <= 1000

3. РЕАЛИЗАЦИЯ

Чтобы решить данную задачу, я использовал динамическое программирование для подсчета максимальной прибыли, которую можно получить, совершив не более k операций. Идея состоит в том, чтобы отслеживать максимальную прибыль на каждый день для каждого возможного количества транзакций до k.

Мы создаем два массива, dp_buy и dp_sell, каждый размером k+1. Массив dp_buy[j] будет отслеживать максимальную прибыль после покупки акции с j транзакциями, а dp_sell[j] будет отслеживать максимальную прибыль после продажи акции с j транзакциями. Если k больше или равно половине количества дней, задача упрощается до случая с неограниченным количеством транзакций, потому что мы не можем совершить более n/2 прибыльных транзакций за n дней. В этом случае мы суммируем все положительные разницы между последовательными днями.

Для каждой цены в массиве prices мы обновляем dp_buy и dp_sell для каждого возможного количества транзакций от 11 до k: максимум между текущим dp_buy[j] и прибылью после покупки сегодня, что равно dp_sell[j-1] – price, максимум между текущим dp_sell[j] и прибылью после продажи сегодня, что равно dp_buy[j] + price.

После обработки всех цен максимальная прибыль, достижимая при не более чем k транзакциях, хранится в dp_sell[k].

4. ЭКСПЕРЕМИНТАЛЬНАЯ ЧАСТЬ

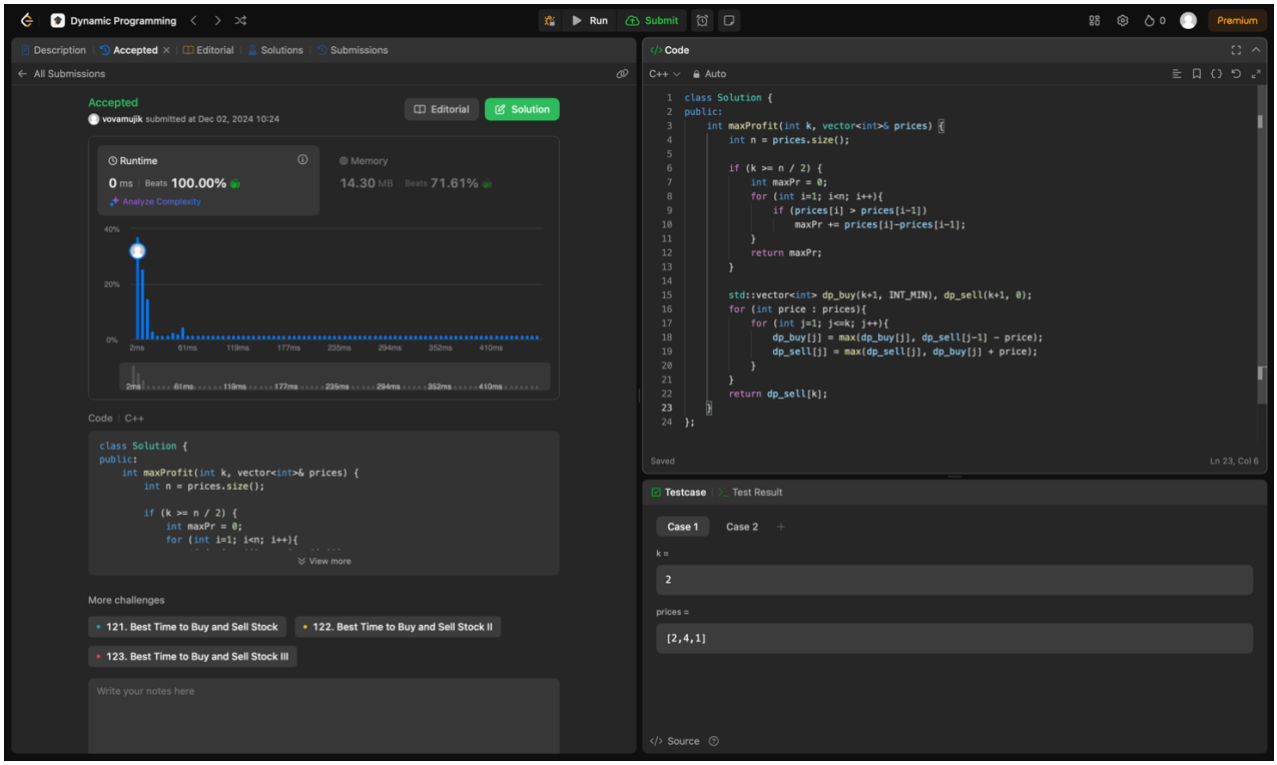
Для анализа сложности получившегося алгоритма, нужно рассмотреть два случая:

1. Случай с неограниченными транзакциями (k >= n/2);
2. Обычный случай (k < n/2).

Для первого случая выполняется только один проход по циклу с прибавлением разницей цены. Итого сложность в данном случае O(N).

Во втором случае, основной цикл по prices (N) имеет вложенный цикл по количеству транзакций k. Внутри вложенного цикла выполняется обновление значений dp_buy[j] и dp_sell[j], каждый за O(1). Итоговая сложность – O(N * k)

По памяти мы создаем только два новых массива dp_buy и dp_sell оба размером k+1. Пространственная сложность получается O(k).



5. ЗАКЛЮЧЕНИЕ

В ходе выполнения работы был разработан и реализован алгоритм для максимизации прибыли от торговли акциями с использованием методов динамического программирования. Цель работы была достигнута: алгоритм успешно решает задачу на тестовых данных. Также алгоритм показывает высокую эффективность со сложностью O(N * k).

ПРИЛОЖЕНИЕ

Листинг кода решения

```
class Solution {
public:
    int maxProfit(int k, vector<int>& prices) {
        int n = prices.size();

        if (k >= n / 2) {
            int maxPr = 0;
            for (int i=1; i<n; i++){
                if (prices[i] > prices[i-1])
                    maxPr += prices[i]-prices[i-1];
            }
            return maxPr;
        }

        std::vector<int> dp_buy(k+1, INT_MIN), dp_sell(k+1, 0);
        for (int price : prices){
            for (int j=1; j<=k; j++){
                dp_buy[j] = max(dp_buy[j], dp_sell[j-1] - price);
                dp_sell[j] = max(dp_sell[j], dp_buy[j] + price);
            }
        }
        return dp_sell[k];
    }
};
```