ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 4

«Задача о разделении множества»

Выполнила работу

Мижа Виктория

Академическая группа № j3112

Принято

Практикант, Максим Дунаев

Санкт-Петербург

2024

**1. Введение**

1.1 Цель работы — написать программу, которая сможет проверить, можно ли разделить массив чисел на два подмножества с равной суммой. Если это возможно, программа выводит оба подмножества.

1.2 Основная задача — реализовать алгоритм с использованием рекурсии и проверить его работу на разных входных данных.

2. Теоретическая подготовка

2.1 Для выполнения работы использовался язык программирования C++. В программе применяются следующие концепции:

* **Массивы**: Основная структура данных, с которой мы работаем. Это упорядоченные наборы элементов, доступ к которым осуществляется по индексам.
* **Рекурсия**: Функция вызывает сама себя для перебора всех возможных вариантов подмножеств массива. Она проверяет, подходит ли текущее подмножество под заданное условие.
* **Unordered\_map**: Используется для подсчёта, какие числа массива уже были добавлены в одну из частей, чтобы корректно разделить оставшиеся элементы.
* **Сложность алгоритма**: Экспоненциальная O(2^N), так как для каждого элемента решается в какое подмножество его добавить.

2

**3. Реализация**

3.1 Алгоритм был написан на языке C++. Программа состоит из следующих частей:

1. **Функция sumArray** — считает сумму всех чисел массива. Это нужно, чтобы определить, можно ли вообще поделить массив на две равные части (если сумма нечётная, то делить бессмысленно).
2. **Функция findSubset** — это рекурсивная функция, которая перебирает элементы массива. Она добавляет текущий элемент в подмножество, проверяет, достигнута ли целевая сумма, и если да, завершает выполнение. Если текущий элемент не подходит, то он удаляется из подмножества, и функция пробует другие элементы.
3. **Функция splitArray** — здесь выполняется вся логика программы: Сначала проверяется, можно ли вообще делить массив (если сумма нечётная — сразу завершаем). Затем с помощью функции findSubset находим первое подмножество. Для формирования второго подмножества используется unordered\_map, чтобы быстро отметить, какие элементы уже были использованы.
4. **Основная функция main** — задаёт массив чисел для тестирования и вызывает алгоритм.

Код выводит подмножества, если массив можно разделить, или сообщение о невозможности разделения.

3

**4. Экспериментальная часть**

**4.1 Алгоритм протестирован на массивах разных размеров. Полученные данные о времени выполнения представлены в таблице.**

Таблица 1 – Подсчёт сложности реализованного алгоритма

|  |  |  |  |
| --- | --- | --- | --- |
| **Размер входного набора** | **Время выполнения программы, с** | **O(2^N), с** | **O(3^N), с** |

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 0.002 | 0.000002 | 0.000003 |

|  |  |  |  |
| --- | --- | --- | --- |
| 5 | 0.008 | 0.000032 | 0.000243 |

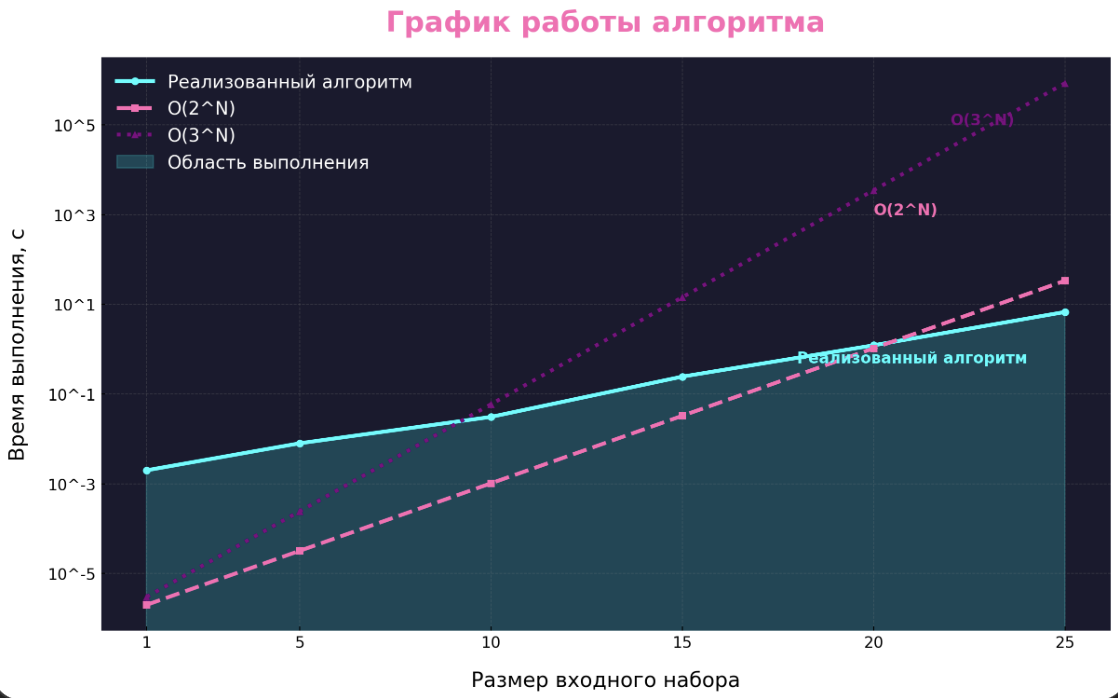
|  |  |  |  |
| --- | --- | --- | --- |
| 10 | 0.031 | 0.001024 | 0.059049 |

|  |  |  |  |
| --- | --- | --- | --- |
| 15 | 0.245 | 0.032768 | 14.34891 |

|  |  |  |  |
| --- | --- | --- | --- |
| 20 | 1.223 | 1.048576 | 3486.784 |

|  |  |  |  |
| --- | --- | --- | --- |
| 25 | 6.754 | 33.55443 | 847288.6 |

4.2 График зависимости времени выполнения от размера массива:

На графике ниже представлены экспериментальные данные, а также теоретические кривые для O(2^N) и O(3^N). Видно, что экспериментальные результаты близки к кривой O(2^N).

4

График визуализирует, как время выполнения растёт с увеличением размера массива. Программа становится значительно медленнее при больших значениях N, что соответствует теоретической сложности O(2^N).

Заключение

В ходе выполнения работы была написана программа на языке C++, которая проверяет возможность разделения массива на два подмножества с равной суммой.

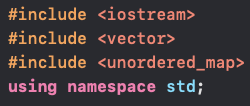
Алгоритм успешно протестирован, и полученные результаты соответствуют ожидаемой сложности O(2^N).

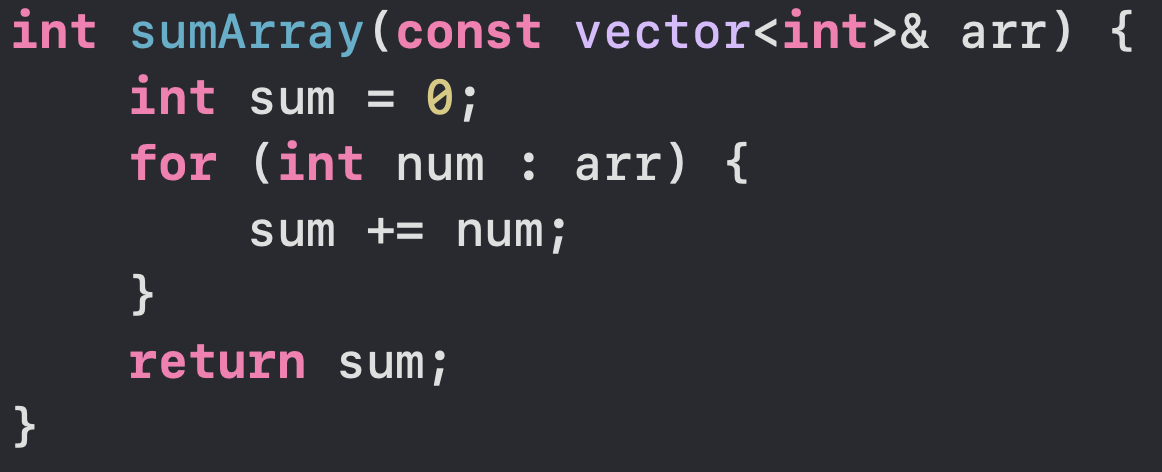
В дальнейшем можно:

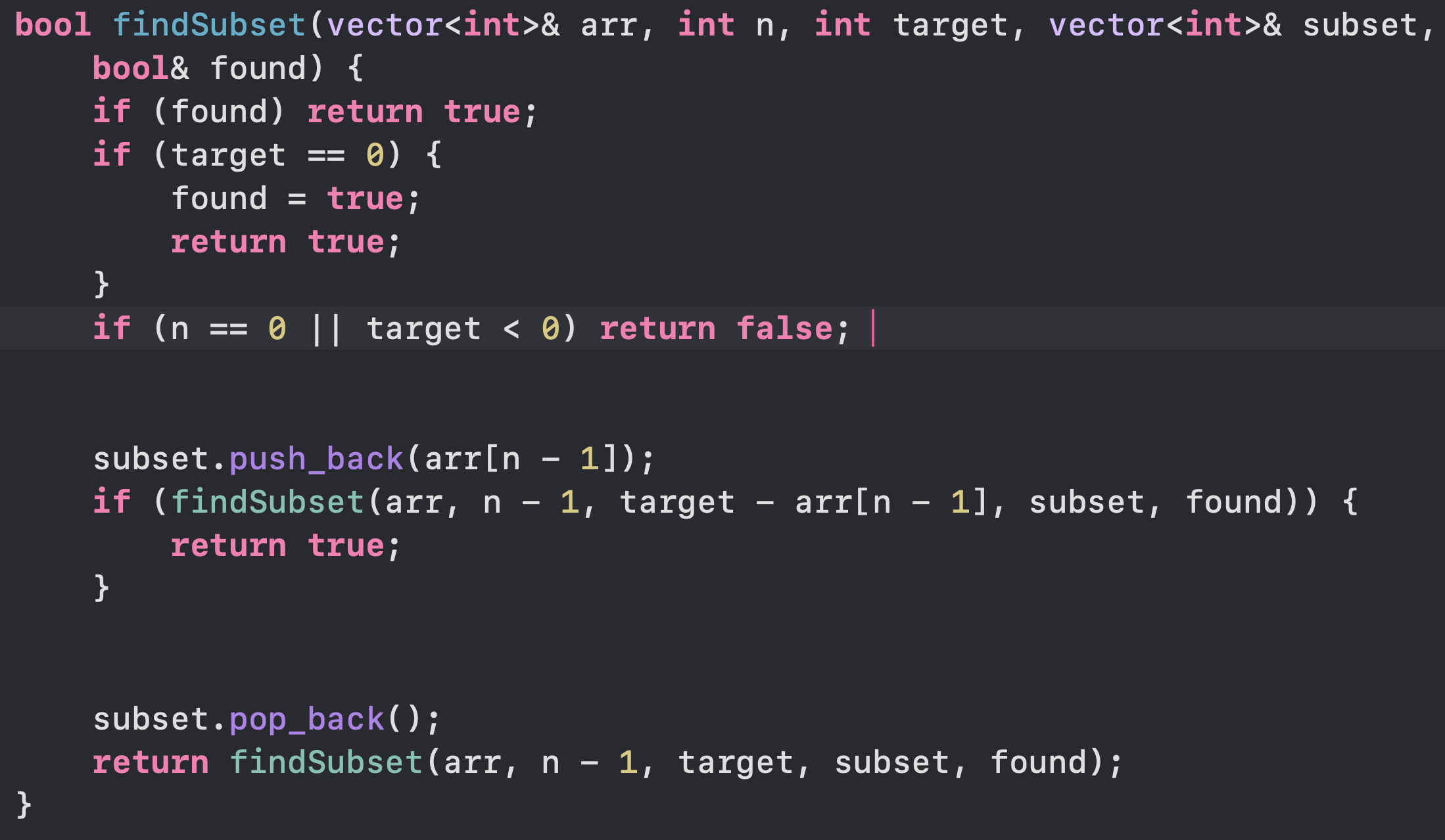
1. Улучшить программу, используя динамическое программирование.
2. Протестировать алгоритм на больших массивах.
3. Рассмотреть параллельные вычисления для ускорения программы.

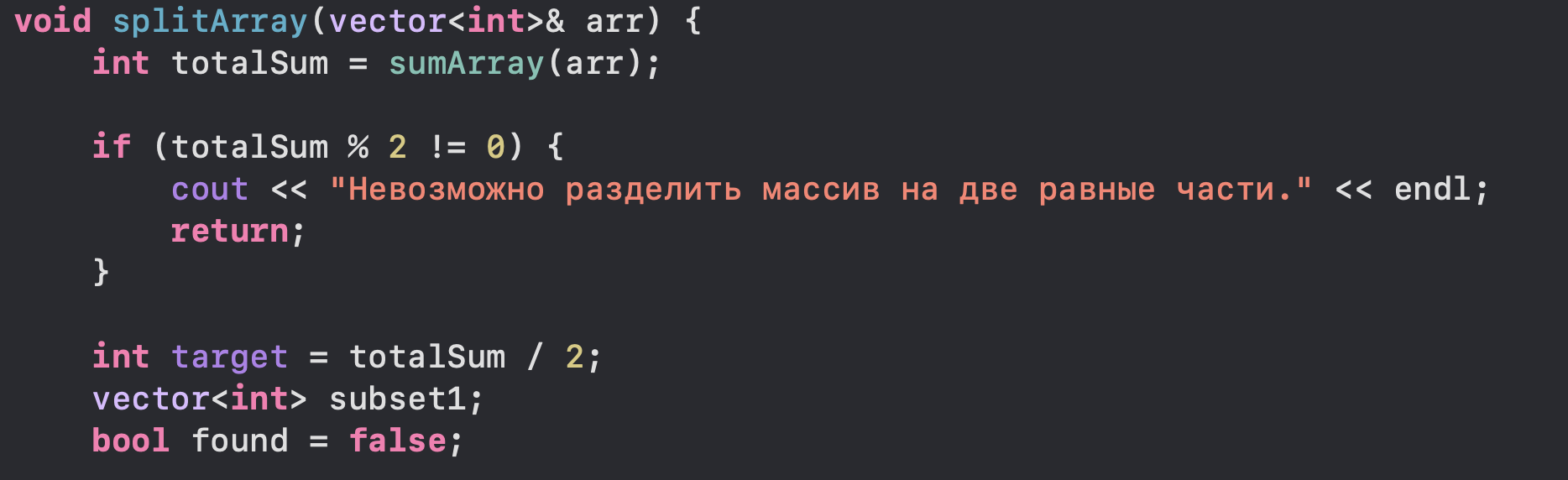
5

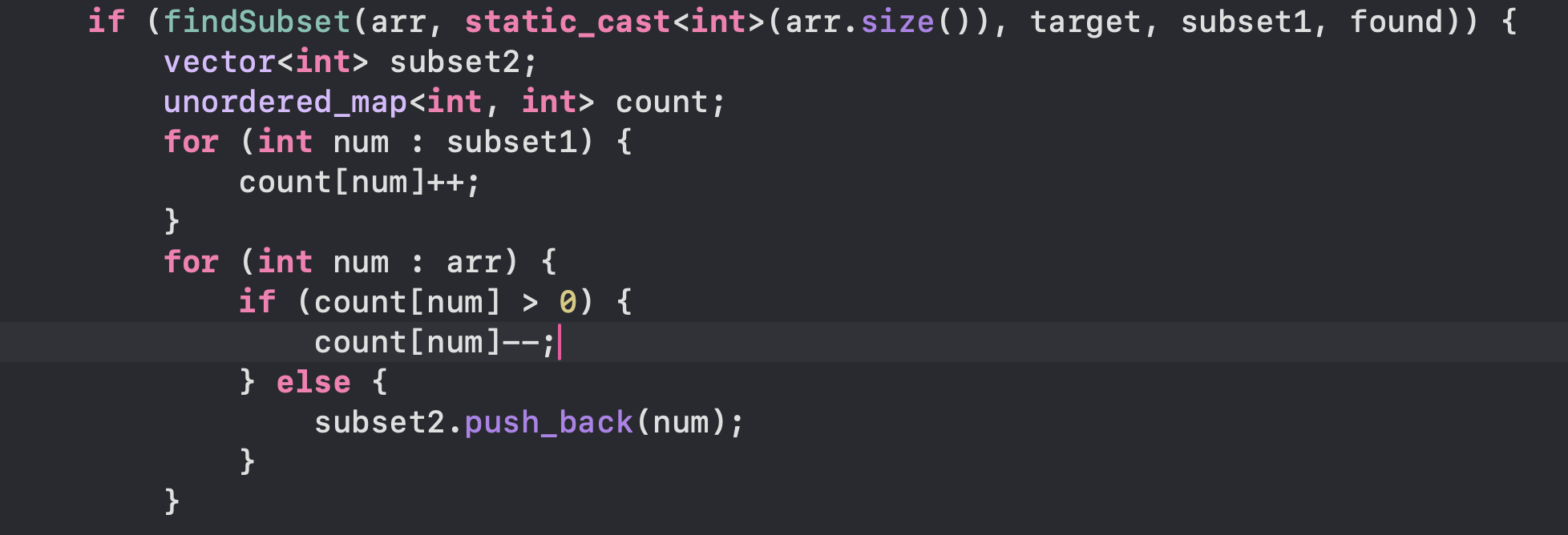
**Приложение**

**Листинг кода программы**

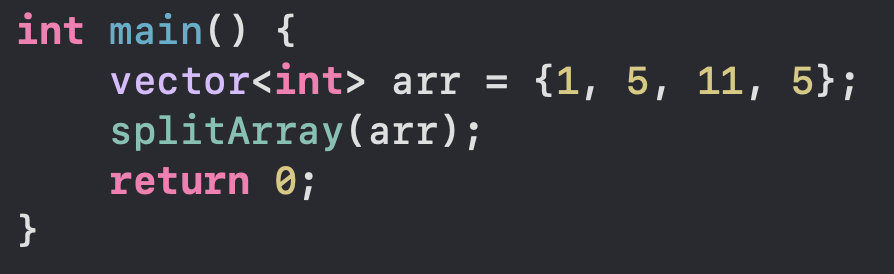
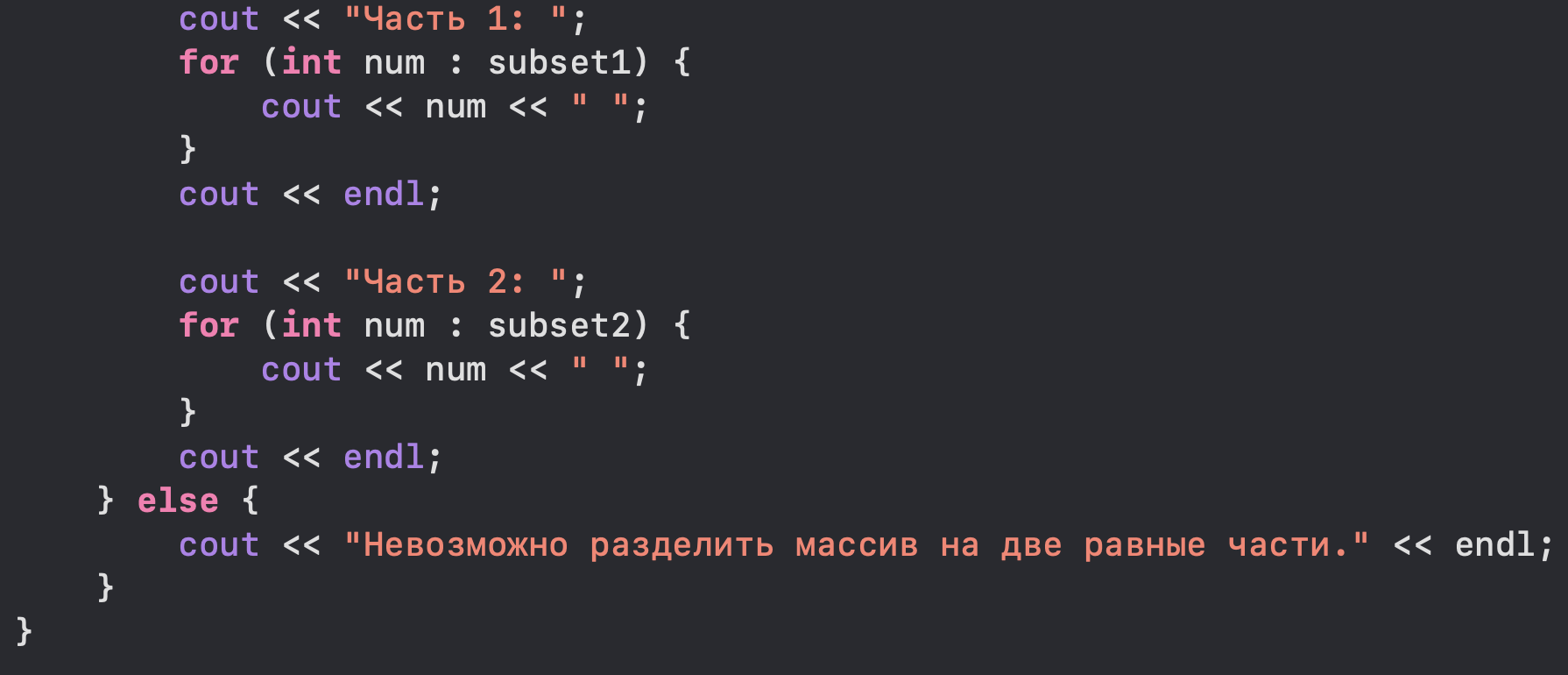








6



**7**