Санкт-Петербургский политехнический университет Петра Великого
Институт машиностроения, материалов и транспорта
Высшая школа автоматизации и робототехники

Отчёт
по курсовой работе по теме
«Дерево Меркла»

Дисциплина: «Объектно-ориентированное программирование»

Студент гр. 3331506/20102 Майоров Е. Д.

Преподаватель Ананьевский М. С.

Санкт-Петербург
2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Дерево Меркла (Merkle Tree) — это криптографическая структура данных, которая используется для эффективной и безопасной проверки целостности больших объемов данных. Оно представляет собой бинарное дерево, в котором каждый листовой узел содержит хеш некоторого блока данных, а каждый внутренний узел содержит хеш объединения хешей своих дочерних узлов. Дерево Меркла получило широкое распространение в современных технологиях, таких как блокчейн, системы контроля версий и распределенные файловые системы, благодаря своей способности обеспечивать целостность данных с минимальными вычислительными затратами. Эта структура данных позволяет проверять целостность информации, используя лишь корневой хеш и небольшое количество дополнительных данных, что делает её особенно ценной в условиях роста объемов информации.

АЛГОРИТМИЧЕСКАЯ ЧАСТЬ

Постановка задачи

В современном мире объемы данных стремительно увеличиваются, что делает задачу обеспечения их целостности и безопасности всё более актуальной. Дерево Меркла предоставляет эффективное решение этой проблемы, позволяя быстро и надежно проверять, были ли данные изменены, без необходимости передачи или хранения полного набора информации. Это свойство делает его востребованным в таких областях, как блокчейн-технологии, распределенные системы и системы контроля версий, где важны надежность и производительность.

В качестве задачи данной курсовой работы было решено реализовать алгоритм Меркла на языке C++ и включить в код автоматическую проверку целостности данных.

Области применения

Блокчейн-технологии: Используется для проверки целостности транзакций в блоках (например, в Bitcoin и Ethereum).

Системы контроля версий: Применяется в Git для обеспечения целостности коммитов и файлов.

Распределенные файловые системы: Позволяет проверять целостность файлов и их фрагментов.

Сетевые протоколы: Используется для эффективной передачи и верификации данных.

Теоретические сведения

Определение

Дерево Меркла — это бинарное дерево, в котором:

- Листовые узлы содержат хеши отдельных блоков данных.

- Внутренние узлы содержат хеши, вычисленные как результат объединения хешей их дочерних узлов.

Принцип работы

Дерево Меркла строится снизу вверх:

1. Для каждого блока данных вычисляется хеш (например, с использованием алгоритма SHA-256).
2. Хеши листовых узлов объединяются попарно, и для каждой пары вычисляется новый хеш.
3. Процесс повторяется, пока не будет получен единый корневой хеш.

Корневой хеш служит "отпечатком" всего набора данных. Если данные изменяются, корневой хеш также изменится, что позволяет обнаружить любые модификации.

Пример работы

Рассмотрим пример с четырьмя блоками данных: D1, D2, D3, D4.

1. **Вычисляем хеши листовых узлов:**
    - H1 = hash(D1)
    - H2 = hash(D2)
    - H3 = hash(D3)
    - H4 = hash(D4)
2. **Вычисляем хеши внутренних узлов:**
    - H12 = hash(H1 + H2)
    - H34 = hash(H3 + H4)
3. **Вычисляем корневой хеш:**
    - H1234 = hash(H12 + H34)

На рисунке 1 представлена схема полученного дерева.

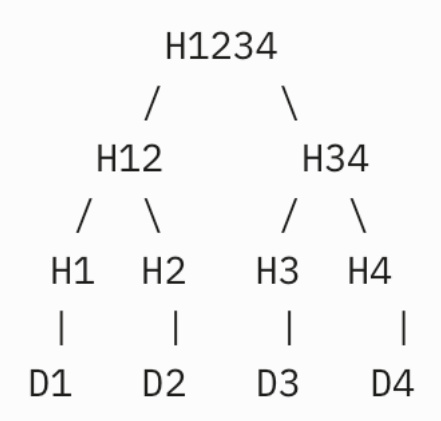

Рисунок 1 – Дерево Меркла

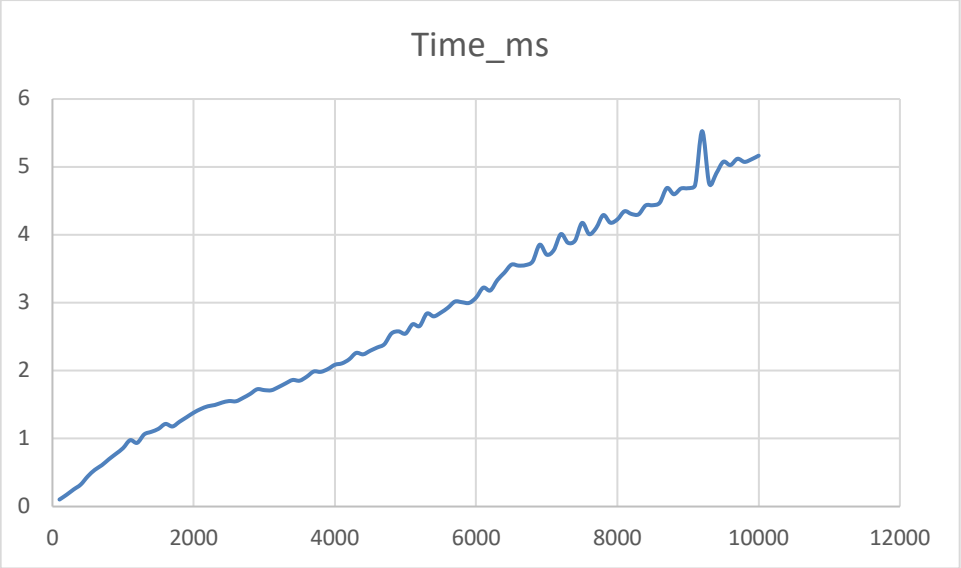

Рисунок 2 – График зависимости времени построения дерева от количества узлов

Комментарии к программному коду

Реализованный код представлен в приложении. Он имеет структуру, соответствующую базовым принципам ООП. Также код содержит все необходимые комментарии. Описание структуры алгоритма и основных реализованных методов представлено далее. Для вычисления хешей используется библиотека OpenSSL.

Класс «MerkleNode» представляет узел дерева Меркла. Листовые узлы содержат хеши блоков данных, а внутренние — хеши дочерних узлов.

Функция «computeHash» вычисляет SHA-256 хеш для заданной строки. Используется как внутри класса, так и отдельно.

Функция «buildMerkleTree» строит дерево Меркла из набора блоков данных, объединяя хеши попарно до получения корневого хеша.

Функция «verifyIntegrity» проверяет целостность данных, сравнивая вычисленный корневой хеш с ожидаемым.

Пример в «main» демонстрирует построение дерева для четырех блоков данных, проверку их целостности и результат после изменения одного из блоков.

ЗАКЛЮЧЕНИЕ

В ходе курсовой работы были реализованы базовые классы для растрового типа данных и слоя многоканального спутникового снимка. Описаны все необходимые методы для расчётов и анализа растра.

Дерево Меркла — это мощный инструмент для обеспечения целостности данных, который сочетает в себе эффективность и криптографическую безопасность. Реализация на языке C++ позволяет глубже понять его принципы работы и демонстрирует практическое применение алгоритма. Изучение и программирование дерева Меркла открывает широкие возможности для применения в современных информационных системах.

СПИСОК ЛИТЕРАТУРЫ

1. Меркл, Р. А. "Метод цифровой подписи на основе обычной криптосистемы" // Труды конференции по компьютерной и коммуникационной безопасности (ACM CCS). – 1987. – С. 369–378.
2. Шнайер, Б. "Прикладная криптография" / Пер. с англ. – М.: Диалектика, 2019. – 784 с. (Классический труд по криптографии, где обсуждаются хеш-функции и их использование в структурах данных.)
3. Онлайн-ресурс: Bitcoin Wiki. "Merkle Tree" [Электронный ресурс]. – URL: https://bitcoinwiki.org/wiki/Merkle_Tree (дата обращения: 21.05.2025).
4. Онлайн-ресурс: Git Documentation. "Git Internals" [Электронный ресурс]. – URL: https://git-scm.com/book/en/v2/Git-Internals-Git-Objects (дата обращения: 21.05.2025).
5. Бутерин, В. "Основы блокчейна: введение в технологию распределенных реестров" / Пер. с англ. – М.: Альпина Паблишер, 2021. – 256 с.

ПРИЛОЖЕНИЕ

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <functional>

// Класс для узла дерева Меркла
class MerkleNode {
public:
    std::string hash;
    MerkleNode* left;
    MerkleNode* right;

    // Конструктор для листового узла
    MerkleNode(const std::string& data) {
        hash = computeHash(data);
        left = nullptr;
        right = nullptr;
    }

    // Конструктор для внутреннего узла
    MerkleNode(MerkleNode* l, MerkleNode* r) {
        left = l;
        right = r;
        hash = computeHash(l->hash + r->hash);
    }

private:
    // Простая хеш-функция на основе std::hash (не особо безопасная)
    std::string computeHash(const std::string& data) {
        std::hash<std::string> hasher;
        size_t hashValue = hasher(data);
        return std::to_string(hashValue);
    }
};

// Функция для построения дерева Меркла из набора блоков данных
MerkleNode* buildMerkleTree(const std::vector<std::string>& dataBlocks) {
    if (dataBlocks.empty()) return nullptr;

    std::vector<MerkleNode*> nodes;
    // Создаем листовые узлы
    for (const auto& block : dataBlocks) {
        nodes.push_back(new MerkleNode(block));
    }

    // Строим дерево снизу вверх
    while (nodes.size() > 1) {
        std::vector<MerkleNode*> newLevel;
        for (size_t i = 0; i < nodes.size(); i += 2) {
            MerkleNode* left = nodes[i];
```

```
            // Если узлов нечетное количество, дублируем последний узел
            MerkleNode* right = (i + 1 < nodes.size()) ? nodes[i + 1] : left;
            MerkleNode* parent = new MerkleNode(left, right);
            newLevel.push_back(parent);
        }
        nodes = newLevel;
    }
    return nodes[0];
}

// Функция для получения корневого хеша
std::string getRootHash(MerkleNode* root) {
    if (root) {
        return root->hash;
    }
    return "";
}

// Функция для проверки целостности данных
bool verifyIntegrity(const std::vector<std::string>& dataBlocks, const std::string&
expectedRootHash) {
    MerkleNode* root = buildMerkleTree(dataBlocks);
    if (!root) return false;
    return root->hash == expectedRootHash;
}

// Пример работы алгоритма
int main() {
    // Набор данных
    std::vector<std::string> data = {"data1", "data2", "data3", "data4"};

    // Построение дерева и получение корневого хеша
    MerkleNode* root = buildMerkleTree(data);
    std::string rootHash = getRootHash(root);
    std::cout << "Корневой хеш: " << rootHash << std::endl;

    // Проверка целостности исходных данных
    bool isValid = verifyIntegrity(data, rootHash);
    std::cout << "Целостность данных: " << (isValid ? "Подтверждена" : "Нарушена") <<
std::endl;

    // Изменение данных
    std::vector<std::string> modifiedData = data;
    modifiedData[2] = "data3_modified";

    // Проверка целостности после изменения
    isValid = verifyIntegrity(modifiedData, rootHash);
    std::cout << "Целостность после изменения: " << (isValid ? "Подтверждена" :
"Нарушена") << std::endl;

    return 0;
```

```
}
```