МЕЖРЕГИОНАЛЬНАЯ ОЛИМПИАДА ШКОЛЬНИКОВ ИМЕНИ И.Я. ВЕРЧЕНКО

Профиль:
Информатика и компьютерная безопасность

2022-2023 учебный год

ЗАДАЧИ С РЕШЕНИЯМИ (11 КЛАСС)

Задача 1. Система обнаружения вторжений 2
Задача 2. Беспилотник 4
Задача 3. Сетевая стеганография 7
Задача 4. Мессенджер 8
Задача 5. Blockchain 10

Задача 1. Система обнаружения вторжений

За несколько месяцев эксплуатации была сформирована статистика работы системы обнаружения вторжений (СОВ), приведенная на рисунке 1.

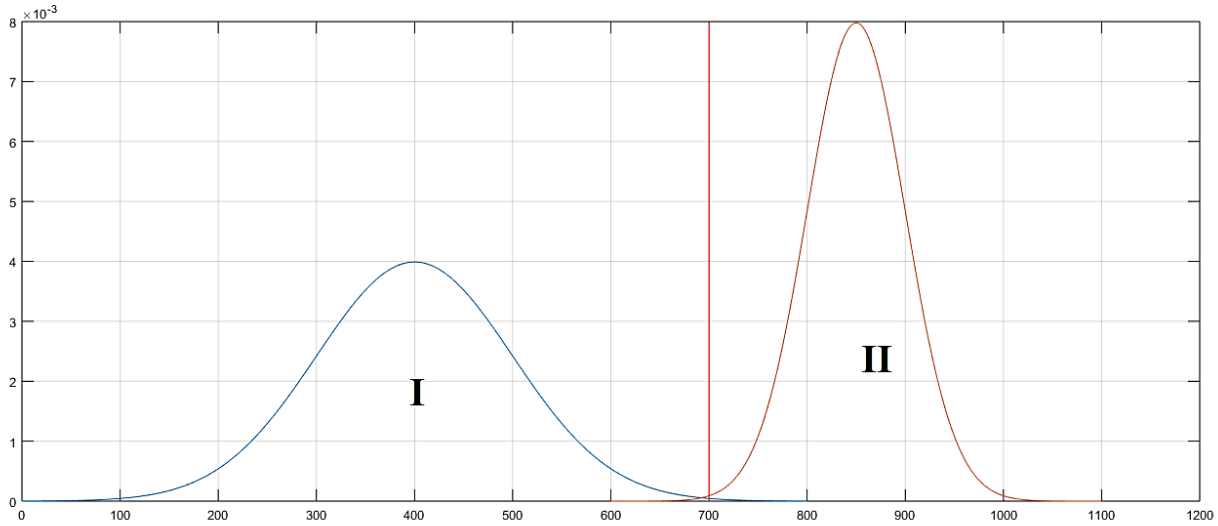


Рисунок 1 – Нормированное распределение трафика по частоте в зависимости от его интенсивности:
(I) – распределение нормального трафика,
(II) – распределение вредоносного трафика

Распределения трафиков представляют собой нормальные распределения со следующими параметрами:

– нормальный трафик (I): математическое ожидание **μ = 400**, дисперсия **σ = 100**,

– вредоносный трафик (II): математическое ожидание **μ = 850**, дисперсия **σ = 50**.

В настоящее время порог принятия решений для СОВ (показан красной линией на рисунке 1) задан так, что объем неправильно обнаруженного нормального трафика (O_I) и объем неправильно обнаруженного вредоносного трафика (O_{II}) равны **0,1%**.

График распределения объема выборки (площади под графиком) в зависимости от положения порога относительно математического ожидания показан на рисунке 2.

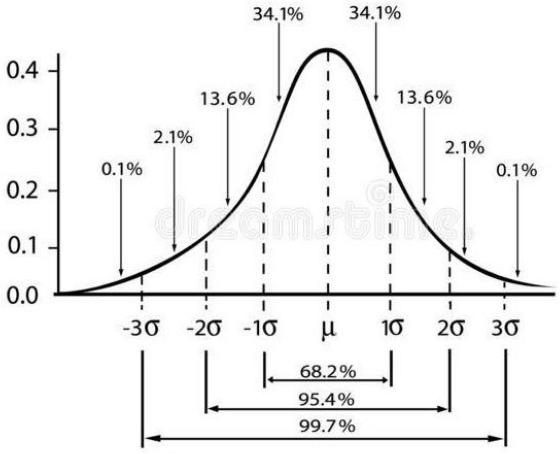


Рисунок 2 – Нормальное распределение трафика

В последнее неделю в сети появился аномальный трафик третьего вида (III), распределение которого показано на рисунке 3.

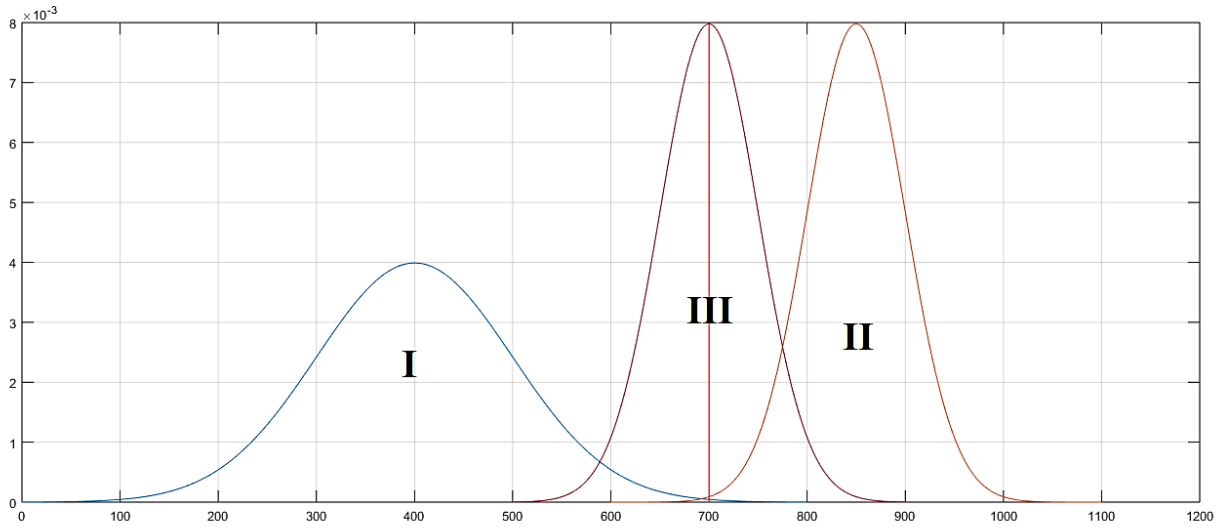


Рисунок 3 – Нормированное распределение аномального трафика по частоте в зависимости от его интенсивности (III)

Аномальный трафик представляет собой нормальное распределение с параметрами: математическое ожидание **μ = 700**, дисперсия **σ = 50**.

При текущих настройках порога СОВ объем ошибочного обнаружения трафика нового вида ($О_{Ш}$) = 50%.

В какую точку необходимо перенести порог принятия решения, чтобы суммарное значение ошибок всех видов трафика было минимальное:

$$O_{сумм} = O_I + O_{II} + O_{III} \rightarrow min?$$

Минимальный шаг изменения порога – 10.

В ответе укажите значение интенсивности, на которой должен быть установлен новый порог принятия решения, а также значения объемов ошибочного обнаружения для всех трех типов трафика.

Решение

Нормальный трафик можно описать следующими количественными параметрами:

- Диапазон значений 0:800
- Мат ожидание 400
- Дисперсия 100
- -3сигма 100
- -2сигма 200
- -1сигма 300
- +1сигма 500
- +2сигма 600
- +3сигма 700

Аномальный трафик 2:

- Диапазон значений 600:1100
- Мат ожидание 850
- Дисперсия 50
- -3сигма 700
- -2сигма 750
- -1сигма 800
- +1сигма 900
- +2сигма 950
- +3сигма 1000

Аномальный трафик 3
- Диапазон значений 500:900
- Мат ожидание 700
- Дисперсия 50
- -3сигма 550
- -2сигма 600
- -1сигма 650
- +1сигма 750
- +2сигма 800
- +3сигма 850

Для нахождения ответа необходимо двигать порог с шагом дисперсии, вычислять вероятность ошибок и искать минимальное значение.

При пороге в 700 график 3 50%, О=0.01+0.1+50.

При пороге в 600 О=3.2*3.2+0+3.2=13,44.

При пороге в 500 О=16,2*16,2+0+0=262,44.

В результате минимальное значение будет найдено при установке порога на значение 600.

Ответ: **600, 3,2% , 0% , 3,2%.**

Задача 2. Беспилотник

Ивану на Новый Год подарили беспилотник с управлением через специальное приложение на смартфоне, которое ведет журнал отправленных беспилотнику команд.

Для проверки корректности работы беспилотника в инструкции предусмотрен специальный тестовый маршрут, по которому необходимо пролететь с использованием приложения на смартфоне. Иван выполнил все команды управления, пролетел маршрут и вернул беспилотник в исходную точку (см. рисунок).

Маршрут из инструкции:

A→B→C→D→E→F→G→H→F→E→B→A

Координаты точек маршрута из инструкции (X;Y;Z):

A = (0; 0; 0) – исходная точка,
B = (0; 0; 12),
C = (12; 0; 12),
D = (12; 6; 12),
E = (0; 6; 12),
F = (0; 6; 24),
G = (0; 2; 24),
H = (-3; 2; 24).

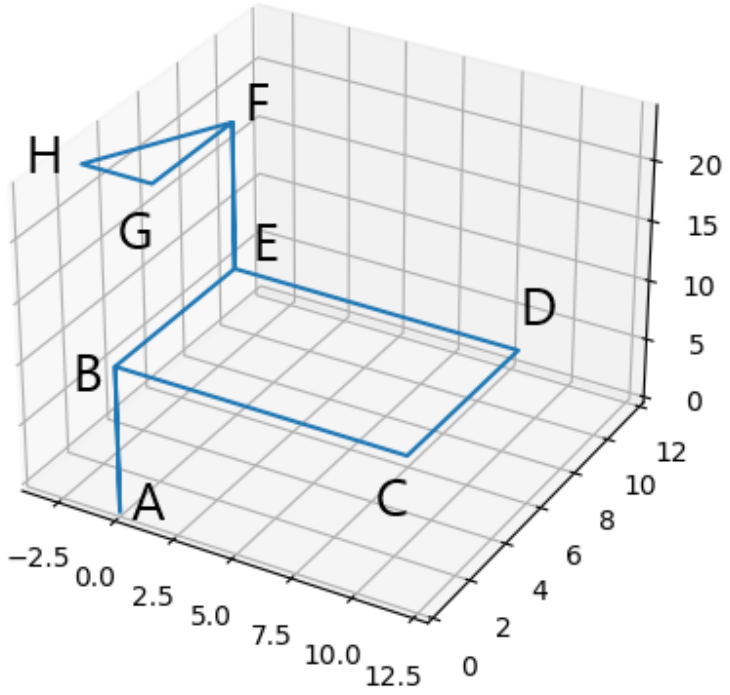


Рисунок – Тестовый маршрут беспилотника (из инструкции).
Единицы изменения шкал – метры

После этого Иван решил самостоятельно управлять беспилотником с использованием приложения. Через какое-то время беспилотник улетел так далеко, что пропал из виду.

На основании журнала отправленных команд помогите Ивану вернуть беспилотник с использованием минимального числа команд. В ответе укажите минимальную последовательность команд, которые необходимо отправить беспилотнику для его возвращения в исходную точку с координатами (0; 0; 0).

Считать, что беспилотник передвигается только по целочисленным координатам, то есть, если после выполнения команды беспилотник должен оказаться в точке с координатами (12,3; 7,8; 5), то он окажется в точке с координатами (12; 8; 5).

К задаче прилагается:

«drone_test_v1.log» – журнал с командами тестового маршрута из инструкции;
«drone_v1.log» – журнал с командами, которые отправлял Иван.

Решение

Требуется понять формат команд, а именно:

id – commandCode – commandParam

Все части команды записаны в шестнадцатеричном формате.

id – во всех командах одинаковый и равен

вариант 1 – `01DF`

вариант 2 – `02FA`

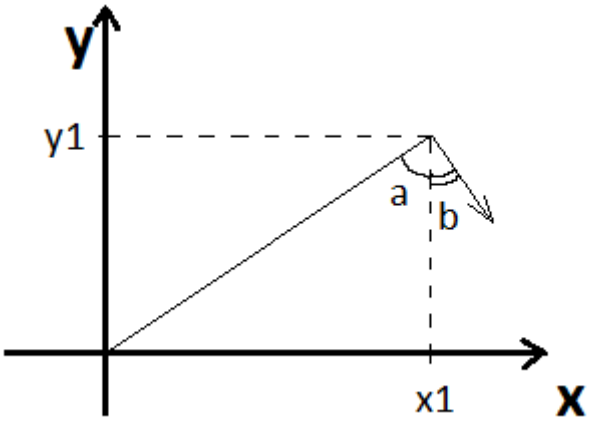
Сопоставляя маршрут движения и отправленные команды из журнала, можно выделить следующие команды:

Команда	Вариант 1	Вариант 2
Установить скорость	A2	D2
Повернуть налево	17	3E
Повернуть направо	E8	C1
Двигаться вперед	F0	A6
Двигаться вверх	B5	B5
Двигаться вниз	4A	4A

Проанализируем журнал команд тестового маршрута. Полный тестовый путь беспилотника был следующий:

00: `01DF A2 0006` (6_{10}) – *УСТАНОВИТЬ СКОРОСТЬ на 6 м/с* (A)
01: `01DF B5 0002` (2_{10}) – *ВВЕРХ 2 с (на 12 м)* (B)
03: `01DF F0 0002` (2_{10}) – *ВПЕРЕД 2 с (на 12 м)* (C)
05: `01DF 17 005A` (90_{10}) – *ПОВОРОТ на 90 градусов* (C)
06: `01DF F0 0001` (1_{10}) – *ВПЕРЕД 1 с (на 6 м)* (D)
07: `01DF 17 005A` (90_{10}) – *ПОВОРОТ на 90 градусов* (D)
08: `01DF F0 0002` (2_{10}) – *ВПЕРЕД 2 с (на 12 м)* (E)
10: `01DF B5 0002` (2_{10}) – *ВВЕРХ 2 с (на 12 м)* (F)
12: `01DF A2 0001` (1_{10}) – *УСТАНОВИТЬ СКОРОСТЬ на 1 м/с* (F)
13: `01DF E8 010E` (270_{10})– *ПОВОРОТ* на 270 градусов* (F)
14: `01DF F0 0004` (4_{10}) – *ВПЕРЕД 4 с (на 4 м)* (G)
18: `01DF E8 005A` (90_{10}) – *ПОВОРОТ* на 90 градусов* (G)
19: `01DF F0 0003` (3_{10}) – *ВПЕРЕД 3 с (на 3 м)* (H)
22: `01DF E8 007E` (126_{10})– *ПОВОРОТ* на 126 градусов* (H)
23: `01DF F0 0005` (5_{10}) – *ВПЕРЕД 5 с (на 4 м)* (F)
28: `01DF A2 0002` (2_{10}) – *УСТАНОВИТЬ СКОРОСТЬ на 2 м/с* (F)
29: `01DF 4A 0006` (6_{10}) – *ВНИЗ 6 с (на 12 м)* (E)
35: `01DF E8 0090` (144_{10})– *ПОВОРОТ* на 144 градусов* (E)
36: `01DF F0 0003` (3_{10}) – *ВПЕРЕД 3 с (на 6 м)* (B)
39: `01DF 4A 0006` (6_{10}) – *ВНИЗ 6 с (на 12 м)* (A)
45: `01DF A2 0000` (0_{10}) – *УСТАНОВИТЬ СКОРОСТЬ на 0 м/с* (A)

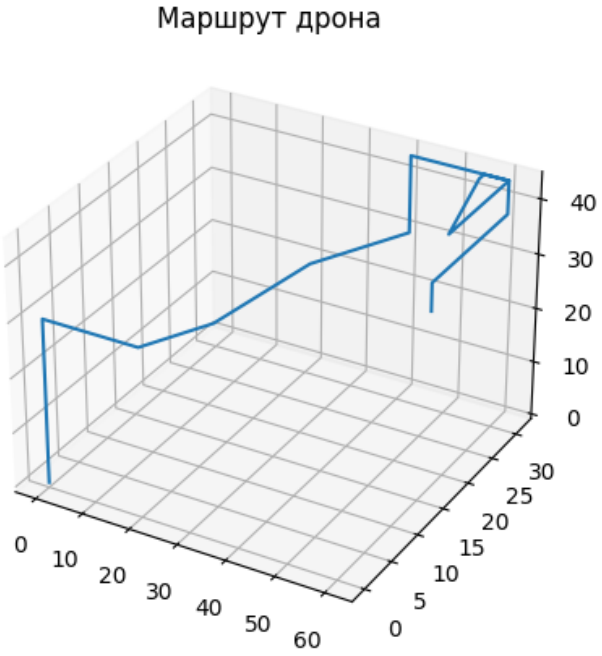
Теперь необходимо вычислить координаты, в которых будет находиться беспилотник после каждой команды и угол его поворота, поскольку движение вперед зависит от угла поворота беспилотника относительно своей оси. Допустим, после последней команды беспилотник оказался в координатах (x1, y1, z1) и «смотрит» под углом b. Тогда ему требуется развернуться на угол a + b:



где угол a = arctg(y1/x1).

Угол b изначально равен 90 градусам (поскольку при первом же движении беспилотника прямо, увеличивается только координата x) и далее вычисляется по ходу приема команд.

Итоговый маршрут беспилотника был следующий:



Для простоты вычисления обратного пути можно установить скорость в 1 м/с. Таким образом ответ будет состоять из команд:

1. Установить скорость в 1 (`A2 01`).
2. Повернуться направо на 75 градусов (`E8 4B`).
3. Двигаться прямо 63 с (`F0 3F`).
4. Двигаться вниз 33 с (`4A 21`).
5. Установить скорость в 0 (`A2 00`).

П.5 может быть опущен в ответе, а п.1 и п.2 могут меняться местами.

Ответ:

1. **`01DFA201` – установить скорость в 1.**
2. **`01DFE84B` – повернуться направо на 4B (75) градусов.**
3. **`01DFF03F` – двигаться прямо 3F (63) с.**
4. **`01DF4A21` – двигаться вниз 21 (33) с.**
5. **`01DFA200` – установить скорость в 0.**

Задача 3. Сетевая стеганография

Агенты одной спецслужбы общаются по открытому каналу связи с помощью сетевых пакетов. Известно, что для передачи текстовых сообщений они используют значения некоторых полей заголовков пакетов.

Администратору удалось перехватить фрагмент передаваемых данных. Изучите его и извлеките передаваемое сообщение. Ответ обоснуйте.

К задаче прилагается:

«packets_v1.cap» – дамп сетевого трафика со скрытым текстовым сообщением.

Решение

При анализе пакетов можно заметить, что их можно отнести к трем протоколам: TCP, IP и UDP. Если фильтровать пакеты по протоколам и анализировать их, то можно заметить следующее.

1) В пакетах, передаваемых по протоколу TCP, пакеты можно поделить на две категории в зависимости от значения поля Source Port. Если предположить, что 445 – 0, а 443 – 1, то можно

получить слово *THIS*.

2) В пакетах, передаваемых по протоколу UDP, есть пакеты с правильной и неправильной контрольной суммой. Если неправильная сумма – 0, а правильная сумма – 1, то можно получить слово *IS*.

3) В пакетах, передаваемых по протоколу IP, различия есть только в одном поле заголовка IPv4 – Time to live (TTL). Он может принимать два различных значения – 64 или 128. Если предположить, что сообщение было закодировано с помощью таблицы ASCII, то можно понять, что каждое значения TTL в пакете соответствует 0 или 1. Рассмотрев два возможных варианта, можно получить секретное слово – *ANSWER*.

Ответ: **THIS IS ANSWER.**

Задача 4. Мессенджер

В компании для общения между сотрудниками используется мессенджер собственной разработки, который передает сообщения в зашифрованном виде. Шифрование производится с использованием метода «двоичного гаммирования» или путем выполнения операции «побитового исключающего ИЛИ» между байтами сообщения и ключа. Для каждого сотрудника ежедневно генерируется новый ключ по следующей формуле

$$K = (Ф_1 * И_1 + Ф_2 И_2 + \ldots + Ф_N И_N) \text{ div } C,$$

где $Ф_1, Ф_2, \ldots, Ф_N$ – код букв дополненной фамилии в соответствии с таблицей ASCII (регистр учитывается),

$И_1, И_2, \ldots, И_N$ – код букв дополненного имени в соответствии с таблицей ASCII (регистр учитывается),

N – максимум из длин фамилии и имени сотрудника,

C – сумма всех цифр текущей даты в формате ДД-ММ-ГГГГ,

div – операция целочисленного деления (целая часть от деления).

Если длина фамилии меньше длины имени, то фамилия дополняется путем дозаписи в конец циклического повторения букв фамилии, пока её длина не сравняется с длиной имени. Аналогично с именем, если его длина меньше длины фамилии.

Например, для сотрудника с ФИО 'Ivanov Petr' 5 марта 2023 ключ будет вычисляться следующим образом:

1. Выравнивание длин имени и фамилии – имя дополняется двумя дополнительными символами:
   Ivanov – 73 118 97 110 111 118
   PetrPe – 80 101 116 114 80 101
2. Вычисление суммы цифр даты:
   C = 0 + 5 + 0 + 3 + 2 + 0 + 2 +3 = 15
3. Вычисление ключа:
   K = (73*80+118*101+97*116+110*114+111*80+118*101) div 15 =
   = 4156_{10} = $103C_{16}$ = 0001 0000 0011 1100_2.

Далее байты текстового сообщения складывается по модулю 2 с байтами, полученными циклическим повторением последовательности байтов вычисленного ключа.

Руководитель отдела разработки дал поручения своим сотрудникам в течение дня 21 февраля 2023 года написать в чат название аэропорта, откуда им удобнее вылетать в командировку: VKO или DME.

Проанализируйте полученный им зашифрованный поток сообщений из мессенджера и определите:

1) кто не выполнил поручение руководителя?
2) за какой аэропорт проголосовало большинство сотрудников?

К задаче прилагается:

«list_v1.txt» – список сотрудников;
«cypher_v1.txt» – зашифрованный текст переписки в мессенджере.

Решение

На первом шаге необходимо вычислить ключи всех сотрудников 21 февраля.

$$C = 2 + 1 + 0 + 2 + 2 + 0 + 2 + 3 = 12$$

Ключи сотрудников равны:

Korovin Aleksandr = 8021_{10} = $1F55_{16}$ = $0001\ 1111\ 0101\ 0101_2$
Lepchenko Nikita = 7705_{10} = $1E19_{16}$ = $0001\ 1110\ 0001\ 1001_2$
Panfilova Alla = 6614_{10} = $19D6_{16}$ = $0001\ 1001\ 1101\ 0110_2$
Golubev Sergej = 5988_{10} = 1764_{16} = $0001\ 0110\ 0110\ 0100_2$
Efratova Anna = 6649_{10} = $19F9_{16}$ = $0001\ 1001\ 1111\ 1001_2$

Переводим возможные тексты сообщений в двоичный вид

VKO = 86 75 79 = 0101 0110 0100 1011 0100 1111
DME = 68 77 69 = 0100 0100 0100 1101 0100 0101

Ключи дополняются до 24 символов, чтобы соответствовать длине сообщения

Korovin Aleksandr – 0001 1111 0101 0101 0001 1111
Lepchenko Nikita – 0001 1110 0001 1001 0001 1110
Panfilova Alla – 0001 1001 1101 0110 0001 1001
Golubev Sergej – 0001 0110 0110 0100 0001 0110
Efratova Anna – 0001 1001 1111 1001 0001 1001

Вычислим возможные шифротексты:

Korovin Aleksandr
VKO – 0100 1001 0001 1110 0101 0000
DME – 0101 1011 0001 1000 0101 1010

Lepchenko Nikita
VKO – 0100 1000 0101 0010 0101 0001
DME – 0101 1010 0101 0100 0101 1011

Panfilova Alla
VKO – 0100 1111 1001 1101 0101 0110
DME – 0101 1101 1001 1011 0101 1100

Golubev Sergej
VKO – 0100 0000 0010 1111 0101 1001
DME – 0101 0010 0010 1001 0101 0011

Efratova Anna
VKO – 0100 1111 1011 0010 0101 0110
DME – 0101 1101 1011 0100 0101 1100

Разделим шифрованный поток сообщений на группы по 24 символа и сравним с возможными шифротекстами:

0101 1010 0101 0100 0101 1011 – Lepchenko DME
0100 1111 1011 0010 0101 0110 – Efratova VKO
0100 1001 0001 1110 0101 0000 – Korovin VKO
0101 0010 0010 1001 0101 0011 – Golubev DME

В результате не предоставила информацию сотрудница *Panfilova Alla*, а в сообщениях остальных сотрудников голоса распределились поровну: 2 за DME и 2 за VKO.

Ответ:

1. **Сотрудник Panfilova Alla не предоставила информацию.**
2. **Сотрудники проголосовали поровну: 2 за VKO, 2 за DME.**

Задача 5. Blockchain

Существует система хранения документов, построенная на основе связного списка блоков (блокчейн). Блоки состоят из транзакций и информации о предыдущем блоке цепочки. Структура блока описана в формате JSON и содержит следующую информацию:

– номер блока (`_id`),

– дата создания блока в формате “*YYYY-MM-DD*” (`date`),

– список транзакций, входящих в состав блока (`storage`),

– хеш-значение предыдущего блока (`hash_prev`),

– контрольная строка (`nonce`).

Для добавления блока в связный список необходимо вычислить значение контрольной строки (NONCE) такое, чтобы хеш-функция этого блока возвращала строку, начинающуюся с символов ‘00’. Хеш-функция блока вычисляется на основе значения хеш-функции предыдущего блока, значения хеш-функции от списка идентификаторов транзакций блока и значения контрольной строки NONCE (см. листинги 1, 2 на языке C++).

Листинг 1 – Функция получения хеш-строки по массиву идентификаторов транзакций

Язык C++

```
// алфавит хеш-строки
const std::string ALPHABET =
"0123456789@<=>ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";

// получение символа из алфавита хеш-строки по его индексу (по модулю длины алфавита)
char getHashChar(unsigned long long index) {
    return ALPHABET[index % ALPHABET.length()];
}


// Получение хеш-строки по массиву из ID транзакций
// STORAGE - массив (вектор) чисел-идентификаторов транзакций
// SIZE    - длина получаемой хеш-строки
// RETURN
//   вычисленную хеш-строку длины SIZE
std::string StorageHash(std::vector<int> storage, int size) {
    // строка - результат (резервируем необходимое количество символов)
    std::string res(size, 0);

    // необходимые константы
    unsigned long long intHash;
    unsigned long long sum = 42;
    unsigned long long mul = 37;

    // основной цикл
    for (int i = 0; i < size; i += 2) {
        // вычисление суммы с предыдущим вычисленным значением hash
        intHash = sum + i + (i > 0 ? res[i - 1] : 0);
        // нормирование - получение символа хеш-строки
        res[i] = getHashChar(intHash);
        // вычисление произведения на основе суммы
        intHash = (mul + i) * intHash;
        // нормирование - получение символа хеш-строки
        res[i + 1] = getHashChar(intHash);
    }
```

```
    // замешивание идентификаторов транзакций
    for (int i = 0; i < storage.size(); i++) {
        // сложение
        intHash = res[i % size] + storage[i];
        // нормирование - получение символа хеш-строки
        res[i % size] = getHashChar(intHash);
        // умножение
        intHash = res[(i + 1) % size] * storage[i];
        // нормирование - получение символа хеш-строки
        res[(i + 1) % size] = getHashChar(intHash);
    }
    // возвращение результата
    return res;
}
```

Листинг 2 – Функция получения хеш-строки по блоку в цепочке блокчейн

Язык C++

```
// алфавит хеш-строки
const std::string ALPHABET =
"0123456789@<=>ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";

// получение символа из алфавита хеш-строки по его индексу (по модулю длины алфавита)
char getHashChar(unsigned long long index) {
    return ALPHABET[index % ALPHABET.length()];
}

// Получение хеш-строки на блок
// PREVHASH    - хеш-строка предыдущего блока блокчейна
// STORAGEHASH - хеш-строка по массиву ID транзакций текущего блока
// NONCE       - контрольная строка
// SIZE        - длина получаемой хеш-строки
// RETURN
//   вычисленную хеш-строку
std::string BlockHash(std::string prevHash, std::string storageHash, std::string nonce,
int size) {
    // строка - результат (резервируем необходимое количество символов)
    std::string res(size, 0);
    // необходимые константы
    unsigned long long intHash;
    unsigned long long hash = 2139062143;
    unsigned long long sum = 42;
    unsigned long long mul = 37;

    // склеивание prevHash и storageHash
    for (int i = 0; i < size; i += 2) {
        res[i] = unsigned char(prevHash[i] + storageHash[i]);
        res[i + 1] = unsigned char(prevHash[i + 1] * storageHash[i + 1]);
    }
    // основной цикл - склеивание с NONCE
    for (int i = 0; i < size; i += 2) {
        // сложение с очередным символом NONCE
        intHash = sum + i + res[i] + nonce[i];
        // нормирование - получение символа хеш-строки
        res[i] = getHashChar(intHash);
        // сложение и умножение с другим символом NONCE
        intHash = (mul + i + 1) * hash + res[i + 1] + nonce[size/2 + i/2];
        // нормирование - получение символа хеш-строки
        res[i + 1] = getHashChar(intHash);
    }
    // возвращение результата
    return res;
}
```

Один сотрудник решил добавить только что созданный документ (транзакция с идентификатором **ID=33**) в блок, сформированный в предыдущем месяце. Укажите, в какой блок необходимо добавить транзакцию и какое новое значение контрольной строки NONCE в этом блоке нужно задать, чтобы значение хеш-функции блока не изменилось, и модификация блока не была обнаружена в блокчейне?

К задаче прилагается:

«blockchain_v1.json» – содержимое блокчейна в формате JSON.

Решение

Значение хеш-функции блока вычисляется на основе трех значений: хеш-значение предыдущего блока, хеш-значение транзакций блока и значение строки NONCE. Хеш-значение предыдущего блока поменять нельзя. Значение NONCE состоит из 14-ти символов, каждый из которых берется из алфавита, состоящем из 66-ти символов. Таким образом полный перебор значения нецелесообразен. Необходимо детально разобраться как формируется хеш-значение транзакций блока.

Рассмотрим тело функции `StorageHash()` и разобьем его на части :

```
std::string StorageHash(std::vector<int> storage, int size) {
    // ЧАСТЬ 1
    std::string res(size, 0);
    unsigned long long intHash;
    unsigned long long sum = 42;
    unsigned long long mul = 37;

    // ЧАСТЬ 2
    for (int i = 0; i < size; i += 2) {
     intHash = sum + i + (i > 0 ? res[i - 1] : 0);
     res[i] = getHashChar(intHash);
     intHash = (mul + i) * intHash;
     res[i + 1] = getHashChar(intHash);
    }

    // ЧАСТЬ 3 - замешивание идентификаторов транзакций
    for (int i = 0; i < storage.size(); i++) {
     intHash = res[i % size] + storage[i];
     res[i % size] = getHashChar(intHash);
     intHash = res[(i + 1) % size] * storage[i];
     res[(i + 1) % size] = getHashChar(intHash);
    }
    // возвращение результата
    return res;
}
```

Часть 1 тела функции содержит объявления переменных и константы. Эта часть всегда одинаковая для всех блоков.

Часть 2 содержит цикл, заполняющий строку-результат `res`. В этом цикле используются только константы, поэтому в результате выполнения этого цикла всегда будет одно и то же значение для любых блоков.

Часть 3 содержит цикл, в котором в значение строки-результата `res` из части 2 подмешиваются идентификаторы транзакций блока. Эта часть зависит от содержания блока и для каждого блока будет вычислено свое значение. Более того, при изменении транзакций блока именно в этой части функции проявятся изменения хеш-значения блока. В этой части осуществляется сложение и умножение символов строки res на идентификаторы транзакций, входящих в состав блока.

Рассмотрим цикл части 3 подробнее:

Номер	Строка
1.	`for (int i = 0; i < storage.size(); i++) {`
2.	`intHash = res[i % size] + storage[i];`
3.	`res[i % size] = getHashChar(intHash);`
4.	`intHash = res[(i + 1) % size] * storage[i];`
5.	`res[(i + 1) % size] = getHashChar(intHash);`
6.	`}`

Исходя из числа транзакций в блоке (`storage.size()`) будут заполнены соответствующие символы строки res. При этом, первый идентификатор транзакции из блока (`storage[0]`) повлияет на значение первого символа строки-результата (`res[0]`) и второго символа строки-результата (`res[1]`) (строки 3 и 5 листинга).

Второй идентификатор транзакции из блока (`storage[1]`) повлияет на значение второго (`res[1]`) и третьего символа строки-результата (`res[2]`). И так далее. Всего в строке-результате 14 символов.

Таким образом, при добавлении новой транзакции в блок необходимо понять, какие символы строки-результата изменятся. В файле с информацией о блоках в каждом блоке содержится по 5 транзакций. При добавлении 6-й транзакции изменятся только символы `res[5]` и `res[6]` строки-результата.

Теперь рассмотрим тело функции `BlockHash()` и разобьем его на части.

```
std::string BlockHash(std::string prevHash, std::string storageHash,
std::string nonce, int size) {
	// ЧАСТЬ 1
	std::string res(size, 0);
	unsigned long long intHash;
	unsigned long long hash = 2139062143;
	unsigned long long sum = 42;
	unsigned long long mul = 37;

	// ЧАСТЬ 2 - склеивание prevHash и storageHash
	for (int i = 0; i < size; i += 2) {
		res[i] = unsigned char(prevHash[i] + storageHash[i]);
		res[i + 1] = unsigned char(prevHash[i + 1] * storageHash[i + 1]);
	}
	// ЧАСТЬ 3 - склеивание с NONCE
	for (int i = 0; i < size; i += 2) {
		intHash = sum + i + res[i] + nonce[i];
		res[i] = getHashChar(intHash);
		intHash = (mul + i + 1) * hash + res[i + 1] + nonce[size/2 + i/2];
		res[i + 1] = getHashChar(intHash);
	}
	// возвращение результата
	return res;
}
```

В качестве параметров функции `BlockHash()` передаются хеш-значение предыдущего блока (`prevHash`), хеш-строка (`storageHash`), вычисленная функцией `StorageHash()` и некоторая строка `NONCE`. Размер всех строк одинаковый и равен 14 символам.

Часть 1 тела функции содержит объявления переменных и константы. Эта часть всегда одинаковая для всех блоков.

Часть 2 содержит цикл, заполняющий строку-результат `res` на основе значений `prevHash` и `storageHash`.

Рассмотрим цикл из части 2 подробнее.

Номер	Строка
1.	`for (int i = 0; i < size; i += 2) {`
2.	`  res[i] = unsigned char(prevHash[i] + storageHash[i]);`
3.	`  res[i + 1] = unsigned char(prevHash[i + 1] * storageHash[i + 1]);`
4.	`}`

Первый символ строки-результата (`res[0]`) формируется на основе значений первых символов строк (`prevHash[0]+storageHash[0]`) (строка 2 листинга).

Второй символ строки-результата формируется на основе значений вторых символов строк (`prevHash[1]*storageHash[1]`) (строка 3 листинга). И так далее. Длины всех строк одинаковые, поэтому тут зацикливаний не будет.

Можно сделать вывод, что `i`-е символы строки-результата зависят только от `i`-х символов строк `prevHash` и `storageHash`.

Таким образом, при добавлении 6-й транзакции в блок, в строке storageHash изменятся символы `storageHash[5]` и `storageHash[6]`, а значит и изменятся только символы `res[5]` и `res[6]` строки-результата. Остальные символы останутся прежними.

Рассмотрим цикл и части 3, в котором осуществляется формирование итогового хеш-значения блока на основании строки-результата из части 2 и строки NONCE:

Номер	Строка
1.	`for (int i = 0; i < size; i += 2) {`
2.	`  intHash = sum + i + res[i] + nonce[i];`
3.	`  res[i] = getHashChar(intHash);`
4.	`  intHash = (mul + i + 1) * hash + res[i + 1] + nonce[size/2 + i/2];`
5.	`  res[i + 1] = getHashChar(intHash);`
6.	`}`

К первому символу строки-результата (`res[0]`) добавляется первый символ строки NONCE (`nonce[0]`) (строки 2-3 листинга).

Ко второму символу строки-результата (`res[1]`) подмешивается символ с середины строки NONCE (`nonce[7]`) (строки 4-5 листинга).

К третьему символу строки-результата (`res[2]`) добавляется третий символ строки NONCE (`nonce[2]`).

К четвертому символу строки-результата (`res[3]`) подмешивается очередной символ после середины строки NONCE (`nonce[8]`).

И так далее:

`res[4]` – зависит от `nonce[4]`, `res[5]` – зависит от `nonce[9]`,
`res[6]` – зависит от `nonce[6]`, `res[7]` – зависит от `nonce[10]`,
`res[8]` – зависит от `nonce[8]`, `res[9]` – зависит от `nonce[11]`,
`res[10]` – зависит от `nonce[10]`, `res[11]` – зависит от `nonce[12]`,
`res[12]` – зависит от `nonce[12]`, `res[13]` – зависит от `nonce[13]`.

Таким образом, если в строке res в цикле из части 2 изменились символы `res[5]` и `res[6]`, то для того, чтобы итоговое хеш-значение строки не изменилось необходимо поменять соответствующие символы строки NONCE: `nonce[9]` и `nonce[6]`.

Можно написать цикл, который перебирает все символы алфавита в значении строки NONCE на месте символов `nonce[9]` и `nonce[6]` так, что итоговое хеш-значение блока не изменится при добавлении к нему новой транзакции.

В условии сказано, что транзакцию необходимо добавить в блок предыдущего месяца (февраль).

Для проверки решения возьмём февральский блок:

```
{
  "_id": 5,
  "date": "2023-02-18",
```

```
    "hash_prev": "007Ctig2m87HRw",
    "nonce": "pZ0uf@YZa0zR23",
    "storage": [ 30, 27, 25, 7, 29 ]
  }
```

В этот блок необходимо добавить транзакцию с ID 33.

Значение storageHash блока: `xi@OL`**`yS`**`qR9dbGq`

Новое значение storageHash с добавленной транзакцией: `xi@OL`**`IT`**`qR9dbGq`

Как видно, отличаются только символы `storageHash[5]` и `storageHash[6]`.

Старое хеш-значение блока: `00>0E`**`11`**`mCe7a1O`

Новое хеш-значение блока: `00>0E`**`B2`**`mCe7a1O`

Для того, чтобы хеш-значение не изменилось, необходимо подобрать новые значения строки NONCE: `nonce[9]` и `nonce[6]`.

Старое значение NONCE: `pZ0uf@`**`Y`**`Za`**`0`**`zR23`

Новое значение NONCE: `pZ0uf@`**`X`**`Za`**`d`**`zR23`

С новым значением NONCE хеш-значение блока не изменится после добавления в него транзакции с ID 33.

Ответ: **добавление транзакции в блок с _id=5,**
новый NONCE = pZ0uf@XZadzR23,
hash-строка блока = 00>0E11mCe7a1O.
