A3b. Вломщики!

Демченко Георгий Павлович , БПИ-235

Условия

```
size_t CustomHash(std::string key) {
    const int32_t p = ???;
    size_t h = 0;
    int64_t p_pow = 1;
    for  (size_t i = 0; i < key.length(); ++i) {
        h += (key[i] - 'a' + 1) * p_pow;
        p_pow *= p;
    }

    return h;
}
```

1. Алгоритм поиска двухсимволных строк - нейтральных элементов

Обоснование :

Так как нам необходимо составить алгоритм поиска нейтральных элементов, состоящих всего из двух символов и мы знаем что при первой иттерации p_pow = 1, а при второй p_pow = 2, то мы можем составить уравнение для полученного хэша

$$\Rightarrow Hash = (s_1 - 97 + 1) + (s_2 - 97 + 1) * p = 0$$

$$\Rightarrow s_1 = 96 - (s_2 - 96) * p$$

где

s_i - ascii код i-го символа ключа

97 - ascii код символа 'a'

Тогда, зная второй символ строки (его код) s_2, мы всегда сможем найти такой s_1, чтобы полученный хэш равнялся 0.

Также по условию (char)s_i - строчные/прописные латинские буквы и цифры, что накладывает ограничение на возможные значения s_i

$$\Rightarrow s_i \in S = [48, 57] \cap [65, 90] \cap [97, 122]$$

Тогда алгоритм заключается в нахождения для каждого $s_2 \in S$ соотвествующего s_1 по вышепредставленной формуле за константу и проверки принадлежности $s_1 \in S$, для добавления строки $(char)s_1 + (char)s_2$ в список нейтральных элементов

Временная сложность алгоритма : $O(N)$, где N - мощность множества допустмых символов, в нашем случае N = 62

Реализация :

Реализация представлена в файле **Hack.cpp**

```
// allowedCodes – массив с кодами допустимых символов: [48...57,65...90,97...122] – 62
std::vector<std::string> FindNeutralHashedForP(const std::vector<uint32_t>& allowedCode
    std::vector<std::string> neutralElements;

    for (int32_t i = 0; i < allowedCodes.size(); ++i) {
        uint32_t secondCharCode = allowedCodes[i];
        uint32_t firstCharCode = 96 - (secondCharCode - 96) * p;

        if (IsCharCodeAllowed(firstCharCode)) {
            std::string curNeutralElement(1, firstCharCode);
            curNeutralElement += (char)secondCharCode;
            neutralElements.push_back(curNeutralElement);
        }
    }

    return neutralElements;
}

// Функция проверки принадлежность символа (его кода ascii)
// заданым условием границам – строчные/прописные латинские буквы и цифры
// 48–57 – цифры
// 65–90, 97–122 – прописные и строчные латинские символы
bool IsCharCodeAllowed(uint32_t charCode) {
    return (48 <= charCode && charCode <= 57) || (65 <= charCode && charCode <= 90) ||
           (97 <= charCode && charCode <= 122);
}
```

2. Нахождение нейтральных элементов

Функция для нахождения всех нейтральных элементов для $p \leq p_{max}$ представлена в файле **Hack.cpp**

При исполнении файла **Hack.cpp** в консоль будут выведены все нейтральные элементы для $p \leq 31$.

Результат работы программы представлен в файле **result.txt**

```
void FindAllNeutralElementsForPRange(int32_t maxP) {

    std::vector<uint32_t> allowedCodes;
    for (int32_t i = 48; i < 123; ++i) {
        if (IsCharCodeAllowed(i)) {
            allowedCodes.push_back(i);
        }
    }

    std::string delimiter(30, '-');
    std::vector<std::string> neutralEl;
    for (int32_t p = 0; p <= maxP; ++p) {
        std::cout << delimiter << '\n' << "Neutral hash elements for p = " << p << '\n'

        neutralEl = FindNeutralHashedForP(allowedCodes, p);
        if (neutralEl.empty()) {
            std::cout << "No such elements" << '\n';
        }
        for (int32_t i = 0; i < neutralEl.size(); ++i) {
            std::cout << neutralEl[i] << " => Calculated hash: " << CustomHash(neutralE
        }

    }
}
```