ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ УФИМСКИЙ УНИВЕРСИТЕТ НАУКИ И ТЕХНОЛОГИЙ

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1**

по предмету «**Алгоритмы и структуры данных**»

Вариант 33

Выполнил: ст. гр. ПРО-428Б

Мохаммед А.А.

Проверил:

Рипатти А.В.

к.т.н доцент

**Уфа 2024 г.**

**Цель работы**: построение хеш-таблицы, содержащей заданную последовательность элементов (ключей).

**Задачи**:

1. Построить хеш-таблицу, содержащую заданную последовательность элементов (ключей).
2. Построить хеш – таблицу : Сгенерировать 48 пятизначных неповторяющихся чисел (элементов).

Вывести их на экран.

Построить хеш-таблицу, используя:

хеш-функцию – сумма всех цифр элемента,

метод устранения коллизий – метод цепочек.

**Задание**

**Входные данные**

int m = 48 – длина входящей последовательности элементов;

a[m] – сгенерированная последовательность, где m – количество элементов;

**Выходные данные**

b[m] – последовательность полученных индексов, где m – количество индексов;

ht[t] – хеш-таблица, где t – размер хеш-таблицы;

full – количество заполненных ячеек;

full/t – коэффициент заполнения таблицы;

k – всего шагов по всей таблице;

av\_steps (k/m) – среднее количество шагов.

**Порядок построения хеш-таблицы**

Хеш-функция: сумма всех цифр элемента.  
 Метод устранения коллизий: метод цепочки.

1. Генерирование случайного набора чисел с помощью генератора случайных чисел. Проверка набора на уникальность: в случае повторяющихся элементов, набор модифицируется путём замены одного из повторяющихся элементов.

// Генерирование случайного набора чисел с помощью генератора случайных чисел:

let num = Math.floor(Math.random() \* 90000) + 10000;

if (!a.includes(num)) {

insertElementHelper(num);

}

1. Нахождение хеш-адресов каждого числа:  
    for (let i = 0; i < a.length; i++) {

let h\_function = hashFunction(a[i]);

output += h\_function + ((i + 1) % 9 === 0 ? "\n" : " ");

}

1. Внесение элемента в ячейку хеш-таблицы: если ячейка свободна, то элемент вносится в таблицу под полученным хеш-адресом, в противном случае применяется метод устранения коллизий (метод цепочек), и к уже находящемуся в списке элементу добавляется еще один. Процесс идёт до того момента, пока все элементы не будут распределены по хеш-таблице.
2. Нахождение коэффициента заполнения хеш-таблицы и вычисление среднего количества шагов для размещения каждого элемента.

**Описание шагов по построению хеш-таблицы**

1. Генерация неповторяющихся элементов реализована с использованием библиотеки <cstdlib>. На рисунке представлен генератор, который заполняет массив случайными числами. Затем все числа в массиве проходят проверку на уникальность, в случае ее непрохождения повторяющиеся элементы заменяются, на те, которых нет в массиве.

1. Хеширование элементов и создание хеш-таблицы реализованы следующим образом: создается двумерный массив ht размером t, затем через цикл проходим по всем элементам массива сгенерированной последовательности и через арифметическую операцию h\_function = a[i]%t получаем хеш-адрес для каждого элемента. После всего, ключ заносится в двумерный массив, где попадает в одинарный массив с индексом найденного для каждого элемента h\_function. Таким образом получается хеш-таблица в виде двумерного массива ht.

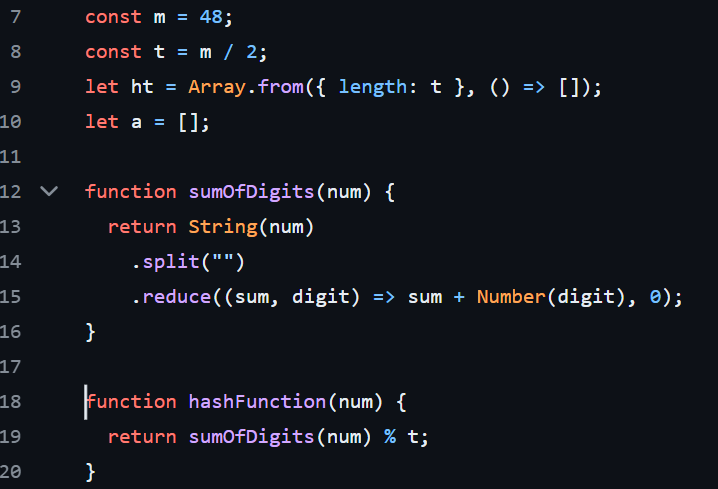


Рис. 1. Хеширование элементов и создание хеш-таблицы

**Формулы расчета требуемых параметров хеш-таблицы**

1. Размер таблицы:

t = int(0.5\*len(a));

где len(a) – размер массива сгенерированных элементов (ключей), по условию 48;

1. Нахождение хеш-адреса элемента:

h\_function = a[i] % t;

1. Коэффициент заполнения таблицы:

a = float(full/t);

где full – количество заполненных ячеек; t – размер таблицы

1. Среднее количество шагов:

k/m;

где k – всего шагов по всей таблице; m – количество элементов, он же размер массива сгенерированных элементов;

**Пример работы программы**

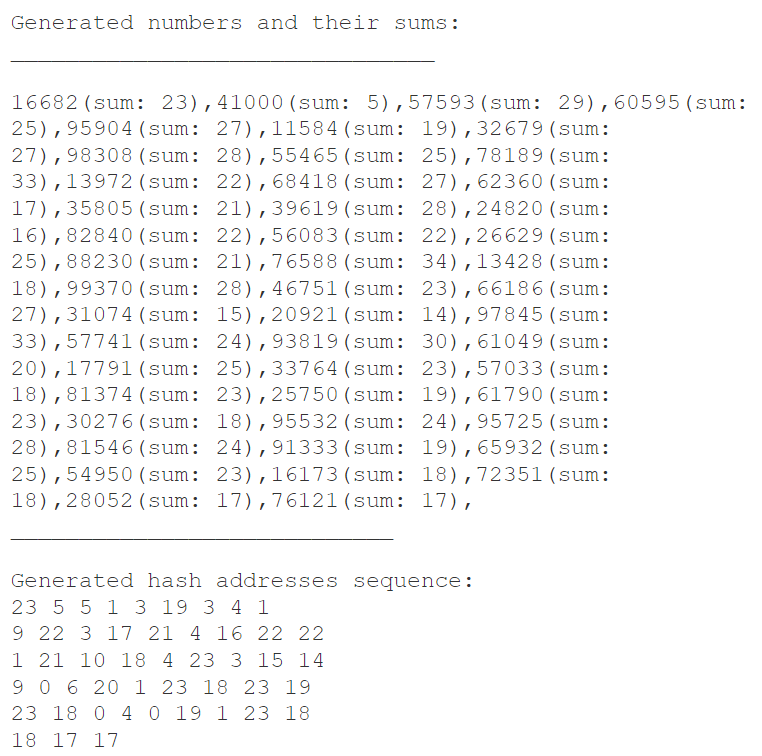


Рис. 2. Генерация случайной последовательности сгенерированных элементов со суммой всех цифр элемент и хеш-адресов (индексов).

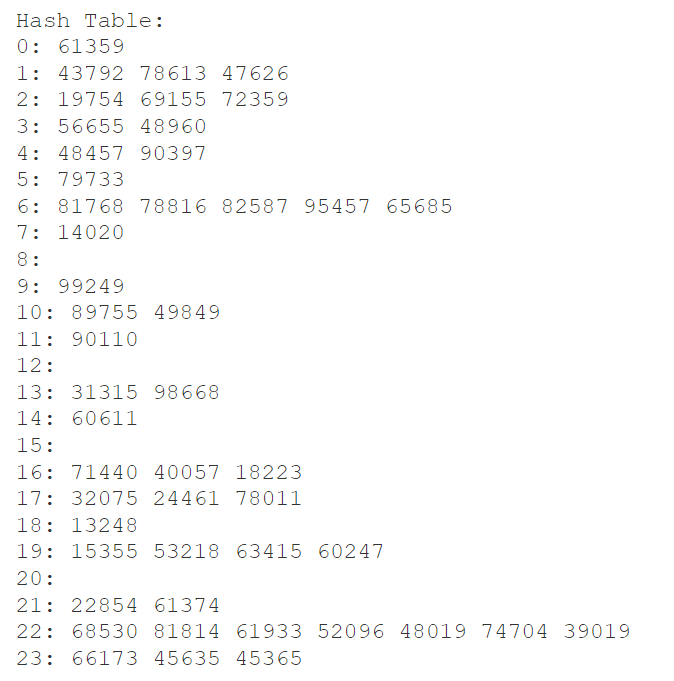


Рис. 3. Вывод хеш-таблицы, метод устранения коллизий – метод цепочек.

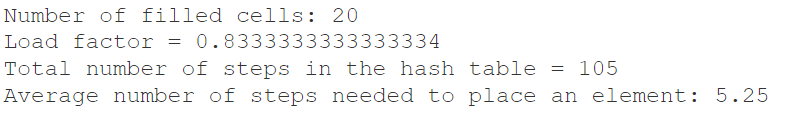


Рис. 4. Вывод показателей.

**Вывод** **по лабораторной работе**

В ходе лабораторной работы была построена хеш-таблица, содержащая заданную последовательность элементов (ключей).

**Приложение 1**

const m = 48;

const t = m / 2;

let ht = Array.from({ length: t }, () => []);

let a = [];

function sumOfDigits(num) {

return String(num)

.split("")

.reduce((sum, digit) => sum + Number(digit), 0);

}

function hashFunction(num) {

return sumOfDigits(num) % t;

}

function printHashTable() {

let output =

"Generated numbers and their sums:\n\_\_\_\_\_\_\_\_\_\_\_\n\n";

for (let i = 0; i < a.length; i++) {

let sum = sumOfDigits(a[i]);

output += ${a[i]}(sum: ${sum}),;

}

output +=

"\n\_\_\_\_\_\_\_\_\_\_\n\nGenerated hash addresses sequence:\n";

for (let i = 0; i < a.length; i++) {

let h\_function = hashFunction(a[i]);

output += h\_function + ((i + 1) % 9 === 0 ? "\n" : " ");

}

output += "\n\nHash Table:\n";

for (let i = 0; i < ht.length; i++) {

output += ${i}: ${ht[i].join(" ")}\n;

}

let filled = ht.filter((bucket) => bucket.length > 0).length;

let load\_factor = filled / ht.length;

output += \nNumber of filled cells: ${filled}\n;

output += Load factor = ${load\_factor}\n;

let totalSteps = ht.reduce(

(sum, bucket) => sum + (bucket.length \* (bucket.length + 1)) / 2,

0

);

let averageSteps = filled ? totalSteps / filled : 0;

output += Total number of steps in the hash table = ${totalSteps}\n;

output += Average number of steps needed to place an element: ${averageSteps}\n;

displayMessage(output);

}

function displayMessage(message) {

document.getElementById("output").innerText = message;

}

function insertElementHelper(num) {

let h\_index = hashFunction(num);

if (!ht[h\_index].includes(num)) {

let position = ht[h\_index].length;

ht[h\_index].push(num);

a.push(num);

return { index: h\_index, position: position };

}

return null;

}

function initializeHashTable() {

while (a.length < m) {

let num = Math.floor(Math.random() \* 90000) + 10000;

if (!a.includes(num)) {

insertElementHelper(num);

}

}

printHashTable();

}

window.onload = initializeHashTable;