БУ ВО Ханты-Мансийского автономного округа – Югры
«Сургутский государственный университет»

Политехнический институт
Кафедра автоматики и компьютерных систем

ОТЧЁТ

по лабораторной работе №7

по дисциплине: «Алгоритмы и структуры данных»

Выполнил: студент группы №609-22,
Бельтюков Михаил Олегович
Принял: старший преподаватель кафедры АиКС
Назаров Евгений Владимирович

Сургут
2024г.

Цель работы

изучить принципов построения хеш-функций, обладающих равномерным распределением, исследовать статистические свойства хеш-функций, закрепить навыки структурного программирования.

Задание

1. Разработать и реализовать функцию, осуществляющую хеширование данных (тип данных определяется вариантом).

2. Разработать и реализовать функцию-генератор, осуществляющую формирование значений ключей в соответствии с заданным типом данных. Генерируемые ключи должны быть уникальны.

3. Исследовать статистические свойства разработанной хеш-функции при заданных размерах хеш-таблицы и количестве ключей.

4. Составить отчет, в котором привести листинг хеш-функции, гистограммы распределений индексов, формируемых хеш-функцией (для двух значений размера хеш-таблицы) и выводы по работе (дать оценку зависимости от размера таблицы и от природы исходных данных – если таковые имеются; оценить качество разработанной хеш-функции).

Таблица 1 — индивидуальный вариант

№	Тип данных	Размеры хеш-таблицы и количество ключей	Примечание
2	struct AutoNumber { char Letter[3]; int Number; };	M1 = 512 M2 = 511 K = 10000	В качестве значений массива используются буквы русского алфавита в нижнем регистре, кроме й, ы, ъ, ь.

Ход работы

```
struct AutoNumber {
    wchar_t Letter[3]; // 1 + 2 * 10 + 3 * 100 + Number
```

```
int Number;
int hash() {
    // 1081 1098 1099 1100
    int deleted = 0;
    for (int i = 0; i < 3; i++) {
        if (Letter[i] > 1081) {
            deleted += 1;
        }
        if (Letter[i] > 1098) {
            deleted += 3;
        }
    }
    return Letter[0] + Letter[1] + Letter[2] + Number - 1072 * 3 - deleted;
    // 1072 1073...1102 1103 - value of rus letters
}
bool equals(AutoNumber a) {
    return ((Letter[0] == a.Letter[0]) && (Letter[1] == a.Letter[1]) && (Letter[2] == a.Letter[2]) && (Number == a.Number));
}
AutoNumber(int * values, int size) {
    bool flag = true;
    while (flag) {
        Number = rand() % (M-84);
        Letter[0] = values[rand() % size];
        Letter[1] = values[rand() % size];
        Letter[2] = values[rand() % size];

        for (int i=0;i<2000;i++) {
            if (not_hash[i].equals(*this)) {
                flag = false;
            }
        }
        flag = !flag;
    }

    // 84 for symbols and M-84 on Number
```

```
    }
    AutoNumber() {
        Number = -1;
        Letter[0] = 0;
        Letter[1] = 0;
        Letter[2] = 0;
    }
};
```

Листинг 1 — Структура, в которой реализованы методы генерации хеша и генерации значений ключа

```
    AutoNumber(int * values, int size) {
        bool flag = true;
        while (flag) {
            Number = rand() % (M-84);
            Letter[0] = values[rand() % size];
            Letter[1] = values[rand() % size];
            Letter[2] = values[rand() % size];

            for (int i=0;i<2000;i++) {
                if (not_hash[i].equals(*this)) {
                    flag = false;
                }
            }
            flag = !flag;
        }

        // 84 for symbols and M-84 on Number

    }
```

Листинг 2 — Функция, генерирующая уникальные значения ключей

```
     int hash() {
        // 1081 1098 1099 1100
        int deleted = 0;
        for (int i = 0; i < 3; i++) {
            if (Letter[i] > 1081) {
```

```
            deleted += 1;
        }
        if (Letter[i] > 1098) {
            deleted += 3;
        }
    }
    return Letter[0] + Letter[1] + Letter[2] + Number - 1072 * 3 - deleted;
    // 1072 1073...1102 1103 - value of rus letters
}
```

Листинг 3 — Функция, генерирующая хеш-индекс по ключу

```
for (int i=0;i<2000;i++) {
        asd = new AutoNumber(values, 28);
        // hashes[i] = (asd->Number + 110300) + asd->Letter[0] * 100 + asd->Letter[1] * 10 + asd->Letter[2] ;
        j = 0;
        while ((hash_table[asd->hash()]+j)->Number != -1){

            j++;
        }
        hash_table[asd->hash()][j] = *asd;
}
```

Листинг 4 — заполнение хеш-таблицы

```
  double sigma = 0;
  int count = 0;
for (int i=0;i<M;i++) {
    count = 0;
    while (hash_table[i][count].Number != -1) {
        count++;
    }
    sigma += pow((count - 10000 / M), 2);
}
cout << sigma * ((double)M/(double)10000);
```

Листинг 5 — вычисление χ2

χ2 = 1979.46

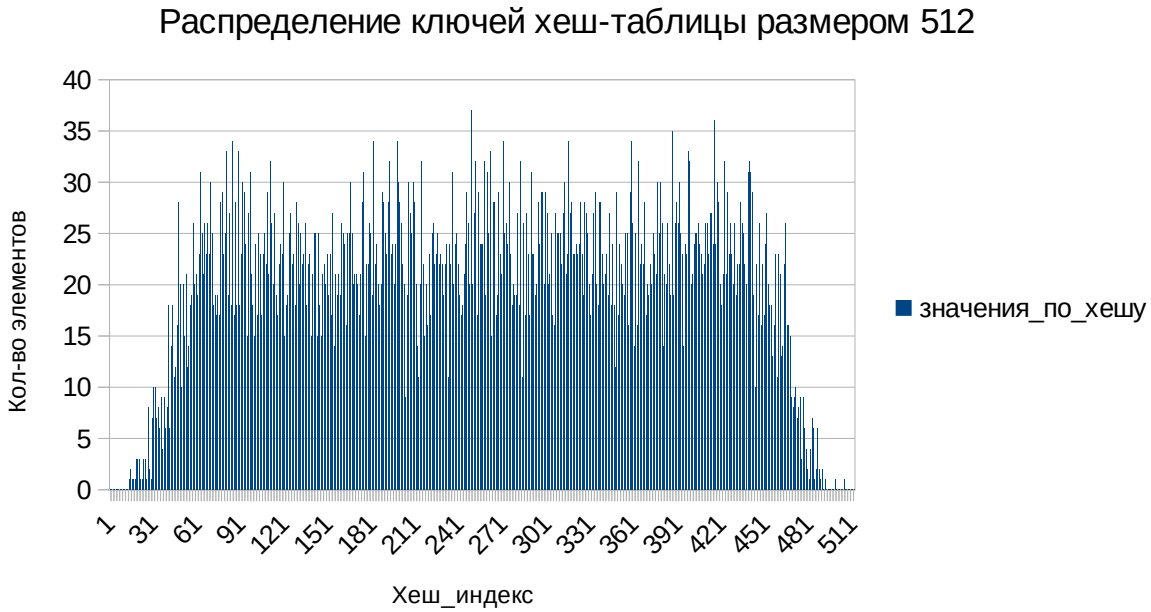

Рисунок 1 — гистограмма распределения ключей хеш-таблицы размером 512

Среднее количество элементов по хеш-индексу равно 19.53125

Рисунок 2 — гистограмма распределения ключей хеш-таблицы размером 511

Среднее количество элементов по хеш-индексу равно 19.5694716

Анализ

Возможных значений ключей гораздо больше чем значений хеш-индексов возникает коллизия — одинаковые хеш-индексы. Для решения этой проблеммы я выбрал способ устранения коллизий «цепочками», в котором хеш-индекс — массив, хранящий в себе несколько ключей с одним хеш-кодом.

Математическое ожидание времени поиска элемента в хеш-таблице составляет $O(1)$. А все операции (поиск, вставка и удаление элементов) в среднем выполняются за время $O(1)$.

По гистограмме видно, что в случае, когда размер хеш-таблицы равен 512 или 511, кол-во уникальных ключей 10000 — максимальная высота цепочки не превышает 40 — это означает, что операции поиска, вставки и удаления не превысит 40 операций сравнения.

Вывод

изучены принципы построения хеш-функций, обладающих равномерным распределением, исследованы статистические свойства хеш-функций.