实验 5: 哈希表

1. 问题描述

1. 输入关键字序列；
2. 用除留余数法构建哈希函数，用线性探测法解决冲突，构建哈希表HT1；
3. 用除留余数法构建哈希函数，用拉链法解决冲突，构建哈希表HT2；
4. 分别对HT1和HT2计算在等概率情况下查找成功和查找失败的ASL；
5. 分别在HT1和HT2中查找给定的关键字，给出比较次数

2. 算法的描述

2.1 数据结构的描述

除留余数法: 对除留余数法，其基本操作是构建一个长度为 m 的哈希表，使用除留余数对将每个数放入哈希表中，若遇到冲突，则以间隔为1向后找到第一个空的位置并将该数放入该位置。

拉链法: 对哈希表中的每个节点，都是一个链表的头，若产生了冲突，则将元素加入该链表的末尾。

2.2 程序结构的描述

`first-hashing.c` 为除留余法，其基本思路如上所述，放入哈希表的代码如下所示：

```
for (i = 0; i < n; i++)
{
    if (h[d[i] % m] == -1)
    {
        h[d[i] % m] = d[i];
    }
    else
    {
        k = 1;
        while (h[(d[i] + k) % m] != -1) k += 1;
        h[(d[i] + k) % m] = d[i];
    }
}
```

`sec-hashing.c` 为拉链法，放入哈希表的代码如下：

```
for(i = 0; i < num_size; ++ i) {
    key = key_saver[i];
    mod = key % mod_set;
    tmp = hash_table[mod];
    if(tmp->firstNode == NULL) {
        p = malloc(sizeof(struct ChainNode));
        p->key = key;
        p->next = NULL;
        tmp->firstNode = p;
```

```
            tmp->chain_len ++;
        } else {
            p = tmp->firstNode;
            while(p->next) {
                p = p->next;
                if(p->key == key) {
                    // duplicated
                    p = NULL;
                    break;
                }
            }

            if(p) {
                new_node = malloc(sizeof(struct ChainNode));
                new_node->key = key;
                new_node->next = NULL;
                p->next = new_node;
                tmp->chain_len ++;
            }
        }
    }
```

3. 调试分析

使用将两个哈希表全部打印出来与手算比较的方法来确认算法的正确性

4. 实验体会和收获

本实验深化了我对哈希表的两种构建方法的理解