Informatics II
Exercise 10

Week 11

Hash Table and Hash Function

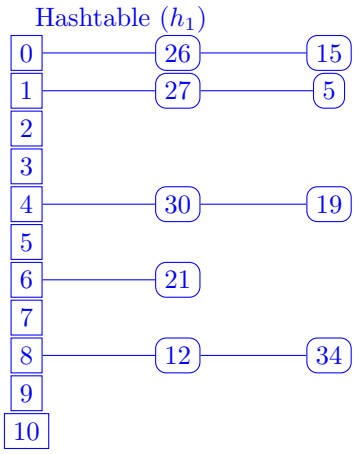
Task 1. Concepts and Basic Calculations Answer the following questions about the hash table.

1. A hash table can be used directly for efficient range queries, i.e., queries that involve a range of keys. For example, find all keys that are smaller than a given key. Justify your answer.

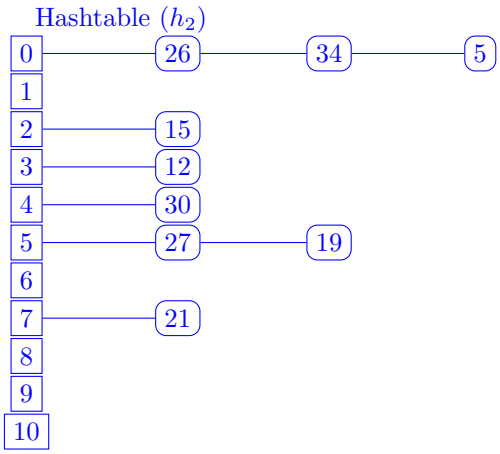
   No. A hash table cannot be used for efficient range queries. Unlike binary search trees, hash tables do not guarantee that the keys are stored in order. Therefore, to find all keys that fall within a given range, we need to iterate over all keys in the hash table, which is not efficient.

2. Consider a hash table `HT` with 11 slots, and it uses chaining to resolve conflicts. For each hash function below, illustrate the hash table `HT` after inserting the following values: 5, 19, 27, 15, 30, 34, 26, 12, and 21.

   (a) $h_1(k) = (k + 7) \bmod 11$ **(division method)**



   (b) $h_2(k) = \lfloor 8(k \cdot 0.618 \bmod 1) \rfloor$ **(multiplication method)**



3. Consider a hash table `HT` with 11 slots using open addressing. For each hash function below, illustrate the hash table `HT` after the keys 5, 19, 27, 15, 30, 34, 26, 12, and 21 have been inserted.

   (a) $h_1(k,i) = (k+i) \bmod 11$ **(linear probing)**

   Linear probing:

Slot	Value
0	
1	34
2	12
3	
4	15
5	5
6	27
7	26
8	19
9	30
10	21

   (b) $h_2(k,i) = (k \bmod 11 + i(k \bmod 7)) \bmod 11$ **(double hashing probing)**

   Double hashing probing:

Slot	Value
0	27
1	34
2	
3	
4	15
5	5
6	12
7	
8	19
9	26
10	30

Task 2. Implementation of Hash Table in C Implement a hash table with m slots, where m is a non-negative integer. You can assume the following:

- $m = 0$ is allowed and should be treated as an empty hash table.
- All keys are positive integers.
- Assume that conflicts are resolved with **double hashing** defined as follows:

$$\begin{aligned} h(k,i) &= (h_1(k) + i * h_2(k)) \ mod \ m \\ h_1(k) &= (k \ mod \ m) + 1 \\ h_2(k) &= m' - (k \ mod \ m') \\ m' &= m - 1 \end{aligned}$$

You can use all mathematical functions directly, as long as they are available in the `math.h` library. Your program should include the following:

- The number `m` is defined as the size of the hash table.

```c
#define m 7
```

- The struct of element `HTElement` in the hash table, including its value and status (occupied, empty or deleted).

```c
#define OCC 0
#define EMP -1
#define DEL -2

struct HTElement {
  int value;
  int status; // 0: occupied, -1: empty, -2: deleted
};
```

- The function `void init(struct HTElement A[])` fills all slots of the hash table `A` with the value 0.

```
1 void init(struct HTElement A[]) {
2   int i;
3   for (i = 0; i < m; i++) {
4     A[i].value = 0;
5     A[i].status = −1;
6   }
7 }
```

- The hashing function `int hash(int k, int i)` that receives the key k and the probe number i and returns the hashed key.

```
1 int hash(int k, int i) {
2   int h1 = (k % m) + 1;
3   int h2 = (m−1)−(k % (m−1));
4   return (int)(h1 + i * h2) % m;
5 }
```

- The function `void insert(struct HTElement[], int key)` inserts the `key` into the hash table `A`.

```
1 void insert(struct HTElement A[], int key) {
2   int counter = 0;
3   int hkey;
4   do {
5     hkey = hash(key, counter);
6   } while(A[hkey].status == OCC && counter++ < m);
7   A[hkey].value = key;
8   A[hkey].status = OCC;
9 }
```

- The function `void delete(struct HTElement A[], int key)` deletes the `key` from the hash table `A`.

```
1 int delete(struct HTElement A[], int key) {
2   int counter = 0;
3   int hkey;
4   do {
5     hkey = hash(key, counter);
6     if (A[hkey].value == key) {
7       A[hkey].status = DEL;
8       return key;
9     }
10  } while (A[hkey].value != key && A[hkey].status == OCC && counter++ < m);
11  return −1;
12 }
```

- The function `int search(struct HTElement A[], int key)` that returns -1 if the *key* was not found in the hash table `A`. Otherwise, it should return the index of the key in the hash table.

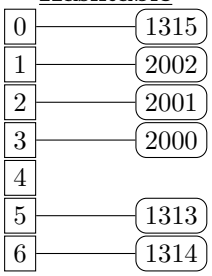
```
1 int search(struct HTElement A[], int key) {
2   int counter = 0;
3   int hkey;
4   do {
5     hkey = hash(key, counter);
```

```
6    } while(A[hkey].value != key && A[hkey].status == OCC && counter++ < m);
7    if (A[hkey].value == key) { return hkey; }
8      else { return −1; }
9  }
```

- The function `void printHashTable(struct HTElement A[])` prints the table size and all non-empty slots of `A`, accompanied by their index and the key.

```
1  void printHashTable(struct HTElement A[]) {
2    int i;
3    printf("Table size: %d\n", m);
4    for (i = 0; i < m; i++) {
5      if (A[i].status == OCC) {
6        printf("i: %d\tkey: %d\n", i, A[i].value);
7      }
8    }
9  }
```

Hashtable



Output printHash

```
Table size: 7
i: 0    key: 1315
i: 1    key: 2002
i: 2    key: 2001
i: 3    key: 2000
i: 5    key: 1313
i: 6    key: 1314
```

Task 3. Birthday Paradox A year has 365 days. Any day is equally likely to be the birthday of one person. More precisely, the probability.

We are interested in two values:

1. If there are $10, 20, 25, 50, 60$ people in the room, what is the probability that there is at least one shared birthday?
2. If there are 50 people in the room, how many people will share a birthday can we expect?

Write a program in C to estimate the above two values.

We create a hash table with 365 slots such that each slot represents a day in a year. We use chaining to resolve collisions, i.e., if two people share a birthday, we store the two people in the linked list of the slot. After we have inserted all people, we can start to count two values according to the task:

1. The number of slots in which there are at least two people. This is the number of at least one shared birthday.
2. The total number of shared birthdays.

We repeat the above process many times, and we can get the probability of at least one shared birthday and the total number of people sharing a birthday that we can expect.

We present the solution source code in three parts:

First, we need to implement a linked list structure, as shown below:

```
1  #include <math.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  #define m 365
7
8  struct Node {
9    int data;
10   struct Node *next;
11 };
12
13 struct Node *HT[m];
14
15 void displayList(struct Node *head) {
16   struct Node *current = head;
17   while (current != NULL) {
18     if (current->next != NULL) {
19       printf(" %d ->", current->data);
20     } else {
21       printf(" %d", current->data);
22       printf("\n");
23     }
24     current = current->next;
25   }
26 }
27
28 int calculateListLength(struct Node *head) {
29   int length = 0;
30   struct Node *current = head;
31   while (current != NULL) {
32     length++;
33     current = current->next;
34   }
35   return length;
36 }
37
38 struct Node *insertList(struct Node *head, int data) {
39   if (head == NULL) {
40     head = malloc(sizeof(struct Node));
41     head->data = data;
42     head->next = NULL;
43     return head;
44   } else {
45     struct Node *newNode = malloc(sizeof(struct Node));
46     struct Node *current = head;
47     newNode->data = data;
48     newNode->next = NULL;
49     while (current->next != NULL) {
50       current = current->next;
51     }
52     current->next = newNode;
53     return head;
54   }
55 }
```

Then we implement the hash table with chaining, with the help of the linked list that we implemented.

```
1  void printHashTable() {
2    int i;
3    printf("Table size: %d\n", m);
4    for (i = 0; i < m; i++) {
5      if (HT[i] != NULL) {
6        printf("i: %d\t, length: %d\t", i, calculateListLength(HT[i]));
7        displayList(HT[i]);
8      }
9    }
10 }
11
12 int hash(int key) { return key % m; }
13
14 void insertHT(int key, int value) {
15   HT[hash(key)] = insertList(HT[hash(key)], value);
16 };
17
18 void clearHT() {
19   int i;
20   for (i = 0; i < m; i++) {
21     HT[i] = NULL;
22   }
23 }
```

Finally, we implement the function for calculating the values we need:

```
1  int find_num_shared_birthday() {
2    // returns the number of shared birthday
3    int i = 0;
4    int length;
5    int num_shared_birthday = 0;
6    for (i = 0; i < m; i++) {
7      if (HT[i] != NULL) {
8        length = calculateListLength(HT[i]);
9        if (length > 1) {
10         num_shared_birthday += 1;
11       }
12     }
13   }
14   return num_shared_birthday;
15 }
16
17 int find_num_people_with_shared_birthday() {
18   // returns the number of people with shared birthday
19   int i = 0;
20   int length;
21   int num_shared_birthday = 0;
22   for (i = 0; i < m; i++) {
23     if (HT[i] != NULL) {
24       length = calculateListLength(HT[i]);
25       if (length > 1) {
26         num_shared_birthday += length;
27       }
28     }
29   }
30   return num_shared_birthday;
```

```
31 }
```

Hint 1: What does a shared birthday mean with respect to a hash table? How would you handle collisions in this case?
Hint 2: You should run your program several times and take the average. You can use the following code to generate random integers:

```
1  #include <time.h>
2  #include <stdlib.h>
3
4  srand(time(NULL)); // Initialization, should only be called once.
5  int r = rand(); // Returns a pseudo-random integer between 0 and RAND_MAX.
```