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.

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)**

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

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:

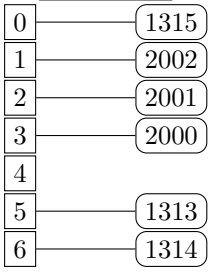
- $m = 0$ is allowed and should be treated as an empty hash table.
- All keys are positive integers.
- Assume that conflicts are resolved with **double hashing** defined as follows:

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

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.
- The struct of element `HTElement` in the hash table, including its value and status (occupied, empty or deleted).
- The function `void init(struct HTElement A[])` fills all slots of the hash table `A` with the value 0.
- The hashing function `int hash(int k, int i)` that receives the key k and the probe number i and returns the hashed key.
- The function `void insert(struct HTElement[], int key)` inserts the `key` into the hash table `A`.
- The function `void delete(struct HTElement A[], int key)` deletes the `key` from the hash table `A`.
- 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.
- 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.

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. We assume that a person has a random birthday.

We are interested in two values:

1. If there are N people in the room, what is the probability that there is at least one shared birthday? Try out N from the following numbers: $10, 20, 25, 50, 60$,
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 people have the same birthday. After we have inserted birthdays of all people, we can start to count two values according to the task:

1. The number of slots that contain more than 1 elements, *i.e.*, more than one person have the same 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.

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.
```