Informatics II
Exercise 10

Published date: March 3, 2021

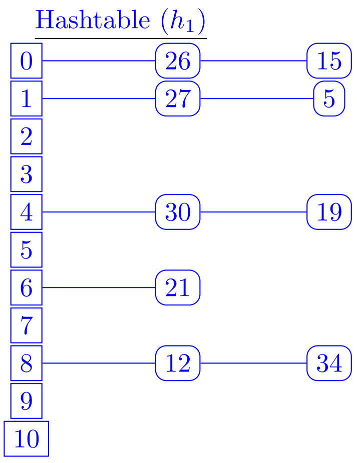
Goals:

- Practise collision resolutions
- Implement hash tables and hash functions in C

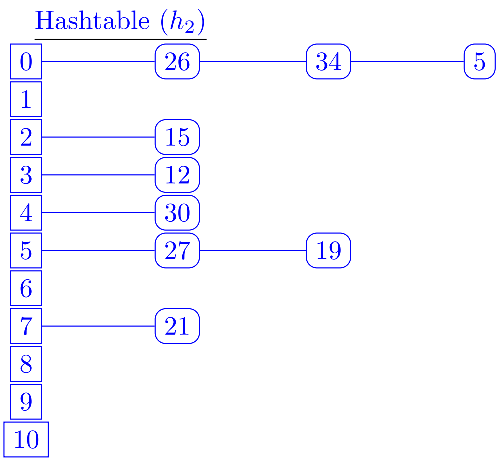
Hash Tables, Hash Functions

Task 1. Consider a hash table HT with 11 slots, and use chaining to resolve conflicts. For each hash function below, illustrate the hash table HT after the following values: 5, 19, 27, 15, 30, 34, 26, 12, and 21 have been inserted (in that order).

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)**



Task 2. 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 (in that order).

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 3. Implement a hash table with m slots, where m is a positive integer. Note that $m = 0$ indicates an empty hash table. All keys are positive integers. Assume that conflicts are resolved with **double hashing** defined as follows:

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

Your program should include the following:

- The number m corresponding to the size of the hash table.

```
#define m 7
```

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

```
void init(int A[]) {
  int i;
  for (i = 0; i < m; i++) {
    A[i] = 0;
  }
}
```

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

```
int h(int k, int i) {
  int h1 = (k % m) + 1;
  int h2 = (m-1)-(k % (m-1));
  return (int)(h1 + i * h2) % m;
}
```

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

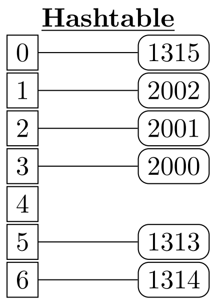
```
void insert(int A[], int key) {
  int counter = 0;
  int hkey;
  do {
    hkey = h(key, counter);
  } while(A[hkey] != 0 && counter++ < m);
  A[hkey] = key;
}
```

- The function *int search(int 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.

```
int search(int A[], int key) {
  int counter = 0;
  int hkey;
  do {
    hkey = h(key, counter);
  } while(A[hkey] != key && A[hkey] != 0 && counter++ < m);
  if (A[hkey] == key) { return hkey; }
    else { return −1; }
}
```

- The function *void printHash(int A[])* prints the table size and all non-empty slots of the hash table *A* accompanied with their index and the key.

```
void printHash(int A[]) {
  int i;
  printf("Table size: %d\n", m);
  for (i = 0; i < m; i++) {
    if (A[i] != 0) {
      printf("i: %d\tkey: %d\n", i, A[i]);
    }
  }
}
```



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
```

Mathematical functions available in the library math.h can be used and don't need to be redefined. For testing purposes, use a table size of 7, add the values 1313, 1314, 1315, 2000, 2001 and 2002 and print your hashtable. Search for the values 2000, 10, 1314, 1313 and 337 and print the results.