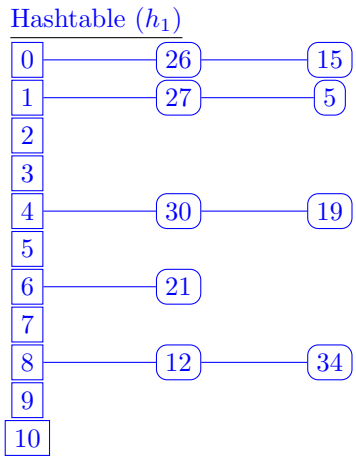
Informatics II
Exercise 10 / **Solution**

May 04, 2020

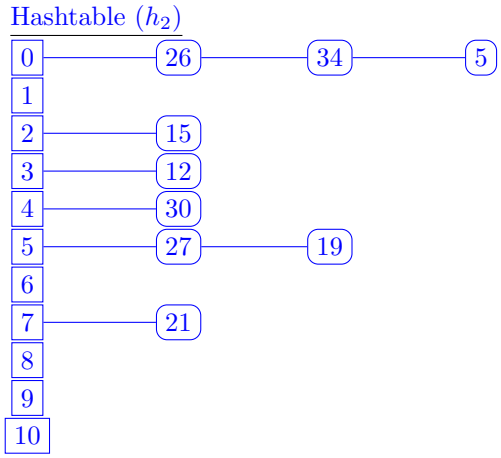
Hash Tables, Hash Functions

Task 1. The hash table $T[0 \ldots 10]$ stores integers and uses chaining to resolve conflicts. Illustrate the hash tables after the values 5, 19, 27, 15, 30, 34, 26, 12, and 21 have been inserted (in that order) using the following hash functions:

1. $h_1(k) = (k + 7) \bmod 11$ **(Division Method)**



2. $h_2(k) = \lfloor 8(k \cdot 0.618 \bmod 1) \rfloor$ **(Multiplication Method)**



Task 2. The hash table T[0...10] may contain up to 11 elements and uses open addressing. Draw an image of the hash table after the keys 5, 19, 27, 15, 30, 34, 26, 12, and 21 have been inserted (in that order) using the following hash functions: For each table indicate the longest probe sequence with the corresponding key.

1. $h_1(k, i) = (k + i) \bmod 11$

   Linear probing: largest probe is 4 for key(26)

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

2. $h_2(k, i) = (k \bmod 11 + i(k \bmod 7)) \bmod 11$

   Double hashing probing: largest successful probe is 1 for insert of keys (27, 30, 26, 12)

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

Task 3. Write a C-program where a hash table consisting of m slots of positive integers is implemented. **Double hashing** should be used to resolve conflicts. The hash-function to be used is:

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

Your program should include the following:

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

```
1 #define m 7
```

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

```
1 void init(int A[]) {
2     int i;
3     for(i = 0; i < m; i++) {
4         A[i] = 0;
5     }
6 }
```

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

```
1  int h(int k, int i) {
2      int h1 = (k % m) + 1;
3      int h2 = (m−1)−(k % (m−1));
4
5      return (int)(h1 + i * h2) % m;
6  }
```

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

```
1  void insert(int A[], int key) {
2      int counter = 0;
3      int hkey;
4      do {
5          hkey = h(key, counter);
6      } while(A[hkey] != 0 && counter++ < m);
7      A[hkey] = key;
8  }
```

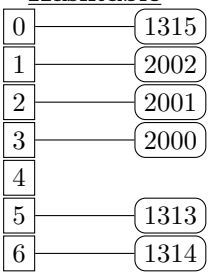
- The function *int search(int A[], int key)* that returns -1 if the requested *key* was not found in the hash table A. Otherwise, it should return the index of the requested hash key in the table.

```
1  int search(int A[], int key) {
2      int counter = 0;
3      int hkey;
4      do {
5          hkey = h(key, counter);
6      } while(A[hkey] != key && A[hkey] != 0 && counter++ < m);
7      if(A[hkey] == key) {
8          return hkey;
9      } else {
10         return −1;
11     }
12 }
```

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

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

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.

```
1   int A[m];
2   init(A);
3
4   insert(A, 1313);
5   insert(A, 1314);
6   insert(A, 1315);
7
8   insert(A, 2000);
9   insert(A, 2001);
10  insert(A, 2002);
11
12  printHash(A);
13
14  int searchValues[] = {2000, 10, 1314, 1313, 337};
15  int i;
16  for(i = 0; i < 5; i++) {
17      if (search(A, searchValues[i]) == -1) {
18          printf("Searching for %d, not found\n", searchValues[i]);
19      }
20      else {
21          printf("Searching for %d, found %d\n", searchValues[i], search(A, searchValues[i]));
22      }
23  }
```

Task 4. Write a C-program where a hash table consisting of m slots of positive integers is implemented. **Chaining** should be used for collision resolution. The hash function to be used is $h(k) = \lfloor m(kA \bmod 1) \rfloor$ where $A = (\sqrt{7} - 1)/2$ and kA mod 1 returns the fractional part of kA. i.e. kA - $\lfloor kA \rfloor$. Each slot is implemented as a linked list to resolve conflicts. A node of a hash table slot is defined as follows:

```
1 struct element {
2   int val;
3   struct element *next;
4 };
```

The global hash table `H` is defined, as follows:

```
1 struct element* H[m];
```

Implement the following functions:

- *void init()*, which initializes hash table H.

```
1 void init() {
2   int i;
3   for (i = 0; i < m; i++)
4     H[i] = NULL;
5 }
```

- *int h(int val)*, which gets a value *val* as input and returns the hashed key.

```
1 int h(int val) {
2   float A=(sqrt(7)-1)/2;
3   return m*(A*val - (int)(A*val));
4 }
```

- *struct element* search(int val)*, which returns NULL if the requested *val* is not found in hash table H. Otherwise, it returns a pointer to the node with the requested *val*.

```
1 struct element* search(int val) {
2   int hkey = h(val);
3   struct element *e = H[hkey];
4   while (e != NULL) {
5     if (e->val == val) { return e; }
6     else { e = e->next; }
7   }
8
9   return NULL;
10 }
```

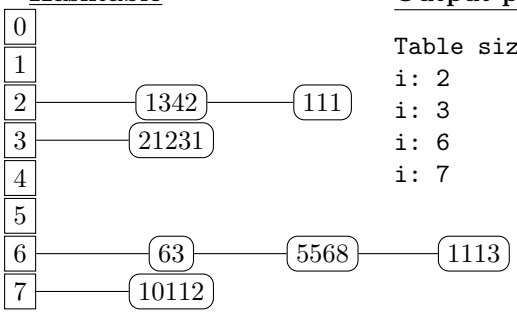
- *void insert(int val)*, which inserts value *val* into hash table H, if it does not already exist in the table (no duplicates should exist).

```
1 void insert(int val) {
2   int i = h(val);
3   struct element* e = malloc(sizeof(struct element));
4
5   e->val = val;
6   e->next = H[i];
7   H[i] = e;
8 }
```

- *void printHash()*, which prints the values of all non-empty slots of the hash table H.

```
1 void printHash() {
2   struct element *e;
3   int i;
4
5   printf("Table size: %d\n", m);
6   for (i = 0; i < m; i++) {
7     if (H[i] != NULL) {
8       printf("i: %d\t val:", i);
9       e = H[i];
10      while (e != NULL) {
11        printf(" -> %d", e->val);
12        e = e->next;
13      }
14      printf("\n");
15    }
16  }
17 }
```

Hashtable



Output printHash

```
Table size: 8
i: 2     val: -> 1342 -> 111
i: 3     val: -> 21231
i: 6     val: -> 63 -> 5568 -> 1113
i: 7     val: -> 10112
```

Mathematical functions that are available via library math.h can be used and don't need to be redefined. For testing purposes, use a hash table of size 8, add the values 111, 10112, 1113, 5568, 63, 1342 and 21231, and print your hash table. Search for values 1, 10112, 1113, 5568, 337 and print the results.

```
1  struct element* tmp;
2  int i;
3
4  init();
5  insert(111);
6  insert(10112);
7  insert(1113);
```

```
8   insert(5568);
9   insert(63);
10  insert(1342);
11  insert(21231);
12
13  printHash();
14
15  int searchValues[] = {1, 10112, 1113, 5568, 337};
16  for (i = 0; i < 5; i++) {
17    tmp = search(searchValues[i]);
18    if (tmp==NULL) printf("Searching for %d, not found\n", searchValues[i]);
19    else printf("Searching for %d, found %d\n", searchValues[i], tmp->val);
20  }
```