Tutorial 03: Hashing

CSCI2520 - DATA STRUCTURES AND APPLICATIONS

TUTOR: ZHANG KAI

Outlines

1. Info about Assignment 1

2. Hashtable

3. Hash function

4. Collision in Hashing

Info about Assignment 1

Assignment 1 will be released after the tutorial.

It is mainly about stack, queue, hashing and linked list.

There are 4 questions in this assignment, of which 2 for programming.

Due date: Mar. 16, 2020.

Hashtable

- Hashtable is a data structure for implementing **dictionary** operations: insert, search, and delete.
- A hashtable is a symbol table that applies a **hash function** on a given **key** to look up the corresponding **value** in the table.
- Fundamental operations include:
  - **Enter**: insert a particular value for a specified key
  - **Look up**: retrieve the corresponding value for a specified key

Hashtable

Definition:

```
typedef struct cellT {
        char *key; void *value;
        struct cellT *next;
} cellT;

struct hashtableCDT {
        cellT *buckets[101];
};
typedef struct hashtableCDT *hashtableADT;

hashtableADT EmptyHashtable();
void Enter(hashtableADT table, char *key, void *value);
void *Lookup(hashtableADT table, char *key);
```

Bucket	Key	Value
0	“s061234”	49
1	“s067890”	58
2	“s051357”	69
3		
4	“s052468”	34
5		
6		
7		

Hashtable

According to the definition of hashtable in last slide, we can find that the entries in this ADT are elements in a cellT array.

So, we need a method to get the index of array for each key-value pair.

Use **Hash Function** to get the index.

Hash Function

The hash function **transforms the key into the index** (hash values, hash codes) of an element where the corresponding value is to be sought

Properties of good hash function:

- reduce the number of collisions
- evenly distributes the keys into buckets
- quick to compute

Hash Function - Exercise 1

Discuss the pros and cons of the hash functions:

```
int hash1(char* key, int H_SIZE) {
    int hash_val = 0;
    for (int i = 0; i < strlen(key); i++)
        hash_val += key[i];
    return(hash_val % H_SIZE);
}
```

```
int hash2(char* key, int H_SIZE) {
    return ((key[0] + 27 * key[1] +
        729 * key[2]) % H_SIZE);
}
```

Hash Function - Exercise 2

Calculate hash values for keys with given hash function:

- {123, 6, 23, 908, 111111, 284}, H_SIZE = 10;

```
int hash1(char* key, int H_SIZE) {
    int hash_val = 0;
    for (int i = 0; i < strlen(key); i++)
        hash_val += char2int(key[i]);
    return(hash_val % H_SIZE);
}
```

- Result: {123: 6, 6: 6, 23: 5, 908: 7, 111111: 6, 284: 4}

Collision in Hashing

Collision: two keys may hash to the same slot:

h(“s10053344”) = 36
h(“s10069999”) = 36

Handle Collision in Hashing

- Open Addressing
- Chaining

Collision in Hashing

Open Addressing

- Linear Probing
- Quadratic Probing
- Double Hashing

Collision in Hashing - Exercise 3

Calculate hash values for keys with given hash function:

- {123, 6, 23, 908, 111111, 284}, H_SIZE = 10;
- Calculate the result with linear probing & quadratic probing respectively

```
int hash1(char* key, int H_SIZE) {
    int hash_val = 0;
    for (int i = 0; i < strlen(key); i++)
        hash_val += char2int(key[i]);
    return(hash_val % H_SIZE);
}
```

Collision in Hashing - Exercise 3 solution

- {123, 6, 23, 908, 111111, 284}, H_SIZE = 10;
- Linear:

Bucket	Value
0	
1	
2	
3	
4	284
5	23
6	123
7	6
8	908
9	111111

Quadratic:

Bucket	Value
0	111111
1	
2	
3	
4	284
5	23
6	123
7	6
8	908
9	

Collision in Hashing

Double Hashing:

- Double hashing uses another hash function *$Hash_2$* to handle collision
- h0 = Hash(key, Nbuckets)
- h1 = (h0 + 1 * Hash2(key, Nbuckets)) % Nbuckets
- h2 = (h0 + 2 * Hash2(key, Nbuckets)) % Nbuckets
- h3 = (h0 + 3 * Hash2(key, Nbuckets)) % Nbuckets

Collision in Hashing - Exercise 4

Given a hash table of size 10, indexed with 0, 1, …, 9.

The hash function is h(i) = i % 10. A list of entries, **{89, 18, 49, 58, 69}**, enter the table.

Using **double hashing** with hash function h’(i) = 7 – (i % 7), to handle collision.

What is the result of the hash table?

Collision in Hashing - Exercise 4 solution

A list of entries, **{89, 18, 49, 58, 69}**, enter the table.

h(i) = i % 10

h’(i) = 7 – (i % 7),

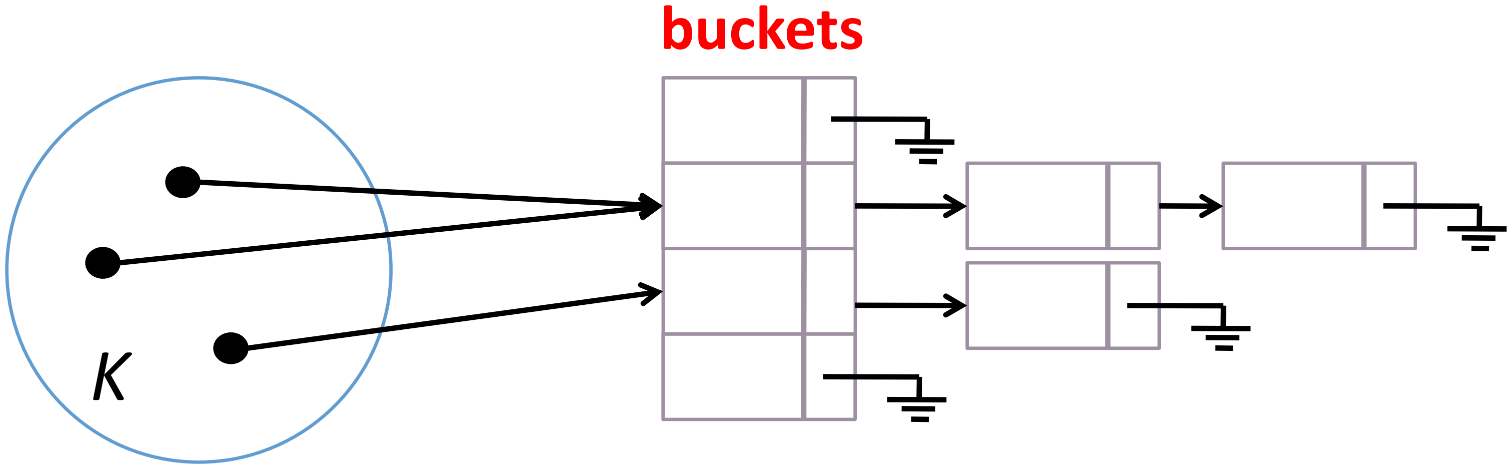
Bucket	Value
0	69
1	
2	
3	58
4	
5	
6	49
7	
8	18
9	89

Collision in Hashing

Chaining

In chaining, we put all elements that hash to the same slot in a **linked list.**

Slot *j* contains a header node of the list that stores all elements that are hashed to *j*.



Collision in Hashing - Exercise 5

Calculate hash values for keys with given hash function:

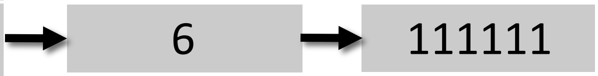
- {123, 6, 23, 908, 111111, 284}, H_SIZE = 10;
- Calculate the result with chaining to handle the collision

```
int hash1(char* key, int H_SIZE) {
    int hash_val = 0;
    for (int i = 0; i < strlen(key); i++)
        hash_val += char2int(key[i]);
    return(hash_val % H_SIZE);
}
```

Collision in Hashing - Exercise 5 solution

A list of entries {123, 6, 23, 908, 111111, 284}, enter the table.

Bucket	Value
0	
1	
2	
3	
4	284
5	23
6	123
7	908
8	
9	

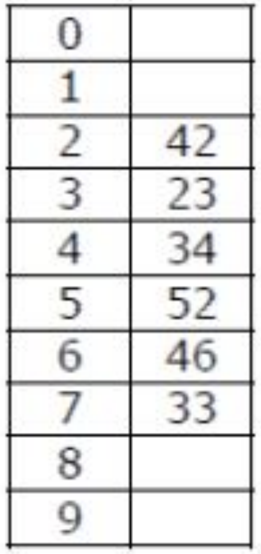


6

111111

Collision in Hashing - Exercise 6

A hash table of length 10 uses open addressing with hash function h(k)=k mod 10, and linear probing. After inserting 6 values into an empty hash table, the table is as shown below.



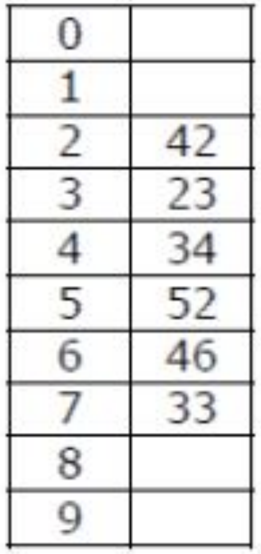
0	
1	
2	42
3	23
4	34
5	52
6	46
7	33
8	
9	

Which one of the following choices gives a possible order in which the key values could have been inserted in the table?

(A) 46, 42, 34, 52, 23, 33
(B) 34, 42, 23, 52, 33, 46
(C) 46, 34, 42, 23, 52, 33
(D) 42, 46, 33, 23, 34, 52

Collision in Hashing - Exercise 6 solution

A hash table of length 10 uses open addressing with hash function h(k)=k mod 10, and linear probing. After inserting 6 values into an empty hash table, the table is as shown below.



0	
1	
2	42
3	23
4	34
5	52
6	46
7	33
8	
9	

Which one of the following choices gives a possible order in which the key values could have been inserted in the table?

(A) 46, 42, 34, 52, 23, 33
(B) 34, 42, 23, 52, 33, 46
(C) 46, 34, 42, 23, 52, 33
(D) 42, 46, 33, 23, 34, 52