DSAD Home Assignment III:2022

Name: Prashant Kumar

Roll No.: 222CS2097

Assignment III: DSAD

[1]. Demonstrate the insertion of the keys 5, 28, 19, 15, 20, 33, 12, 17, 10 into a hash table with collisions resolved by chaining. Let the table have 9 slots, and let the hash function be h(k) = k mod 9.

Solution:

The keys will be mod with 9 and then inserted into the hash table. The mod values for each key will be:
5 mod 9 = 5
28 mod 9 = 0
19 mod 9 = 5
15 mod 9 = 1
20 mod 9 = 6
33 mod 9 = 5
12 mod 9 = 5
17 mod 9 = 3
10 mod 9 = 3
Hence, the hash table will be filled as:
0 -
1 - 28 -> 19 ->10
2 - 20
3 -12
4 -
5 - 5
6 - 15 -> 33
7 -
8 - 17

[2]. Show that if |U| > nm, there is a subset of U of size n consisting of keys that all hash to the same slot, so that the worst-case searching time for hashing with chaining is O(n).

Solution:

Since there are **m** slots available and the size of Universe(U) > nm The number of keys in Universe per slot > n

=> U > nm

=> U/m > n

Assume that:

Number of keys (n) = 100

Number of slots in hash table (m) = 10

Number of keys in U that hash to 1 slot = U/m = n = 100

From the given information, we know that **U/m > n** which means that at least **n** keys hash to the same slot of the table.

Since this implementation is achieved via Chaining, so searching in a list of size **n** will cost O(n) running time.

[3]. Suppose a string of r characters is hashed into m slots by treating it as a radix-128 number and then using the division method. The number m is easily represented as a 32-bit computer word, but the string of r characters, treated as a radix-128 number, takes many words. How can we apply the division method to compute the hash value of the character string without using more than a constant number of words of storage outside the string itself?

Solution:

Basics:

This problem requires some basic number theory principles, and more specifically modulo arithmetic. The following 2 will be applied to solve this problem. (The second one holds for integers only)

(a + b) % m = (a % m + b % m) % m

and

(a. b) % m = ((a % m). (b % m)) %m

Solution:

Based on above analysis, the string of r characters can be hashed using division method with constant space.

Any string of length r can be converted to a radix-128 number, say,

$n = c_0.128^0 + c_1.128^1 + c_2 .128^2 + ... + c_{r-1}.128^{r-1}$,

where ci means the i^th character starting from the least significant end with 0 (which can also be viewed as an integer of its ASCII code).

Hence, applying the division hashing function,

$n \% m = (c_0.128^0 + c_1.128^1 + c_2 .128^2 + ... + c_{r-1}.128^{r-1)})\% m$

$= (c_0.128^0) \% m + (c_1.128^1)\% m + (c_2 .128^2)\% m + ... + (c_{r-1}.128^{r-1})\% m)\% m$

However, It looks that we need first compute some large numbers, for example, 128^{r--1} of the last term, If so. we still need many words to store them. But based on the second rule mentioned above, we don't have to store such large numbers. All we need to do is take the modulo operation after each multiplication. In this way, only constant number of words for storage is needed.

[4]. Draw the 11-entry hash table for hashing the keys 12, 44, 13, 88, 23, 94, 11, 39, 20 using the function (2i+5) mod 11, closed hashing, assuming collisions are handled by linear probing.

Solution:

h(x) = (2*i + 5) mod 11

The keys will be calculated using h(x) and then inserted into the hash table with collision resolution technique linear probing. The mod values for each key will be:

2*12 + 5 mod 11 = 5		
2*22 + 5 mod 11 = 5	2*22 + 5 +1 mod 11 = 6	
2*13 + 5 mod 11 = 9		
2 *88 + 5 mod 11 = 5	2 *88 + 5 +1 mod 11 = 6	2 *88 + 5 +2 mod 11 = 7
2*23 + 5 mod 11 = 7	2*23 + 5 + 1 mod 11 = 8	
2*94 + 5 mod 11 = 6	2*94 + 5 + 1mod 11 = 7	2*94 + 5 +2 mod 11 = 8
	2*94 + 5 +3 mod 11 = 9	2*94 + 5 + 4 mod 11 = 10
2*11 + 5 mod 11 = 5	2*11 + 5+ 1 mod 11 = 6	2*11 + 5+ 2 mod 11 = 7
	2*11 + 5+ 3 mod 11 = 8	2*11 + 5+ 4 mod 11 = 9
	2*11 + 5+ 5 mod 11 = 10	2*11 + 5+ 6 mod 11 = 0
2*39 + 5 mod 11 = 6	2*39 + 5+ 1 mod 11 = 7	2*39 + 5+ 2 mod 11 = 8
	2*39 + 5+ 3 mod 11 = 9	2*39 + 5+ 4 mod 11 = 10
	2*39 + 5+ 5 mod 11 = 0	2*39 + 5+ 6 mod 11 = 1
2*20 + 5 mod 11 = 1	2*20 + 5+1 mod 11 = 2	

Index	Keys
0	**11**
1	**39**
2	**20**
3	
4	
5	**12**
6	**22**
7	**88**
8	**23**

9	**13**
10	**94**

[5]. Design a C++ class that implements the Hash table as data structure?

Solution:

In hashing there is a hash function that maps keys to some values. But these hashing function may lead to collision that is two or more keys are mapped to same value. Chain hashing avoids collision. The idea is to make each cell of hash table point to a linked list of records that have same hash function value.
Suppose a hash function, such that our hash table has 'N' number of buckets.
To insert a node into the hash table, we need to find the hash index for the given key. And it could be calculated using the hash function.
Example: hash Index = key % no of Buckets.

Insert: Move to the bucket corresponds to the above calculated hash index and insert the new node at the end of the list.

Delete: To delete a node from hash table, calculate the hash index for the key, move to the bucket corresponds to the calculated hash index, search the list in the current bucket to find and remove the node with the given key.

Code:

```
#include<iostream>
#include <list>
using namespace std;
class Hash
{
int BUCKET;  // No. of buckets

// Pointer to an array containing buckets
list<int> *table;
public:
  Hash(int V);  // Constructor

  // inserts a key into hash table
  void insertItem(int x);
```

```
// deletes a key from hash table
void deleteItem(int key);

// hash function to map values to key
int hashFunction(int x) {
   return (x % BUCKET);
}

void displayHash();
};

Hash::Hash(int b)
{
this->BUCKET = b;
table = new list<int>[BUCKET];
}

void Hash::insertItem(int key)
{
int index = hashFunction(key);
table[index].push_back(key);
}

void Hash::deleteItem(int key)
{
// get the hash index of key
int index = hashFunction(key);

// find the key in (inex)th list
```

```
list <int> :: iterator i;
for (i = table[index].begin();
        i != table[index].end(); i++) {
if (*i == key)
   break;
}

// if key is found in hash table, remove it
if (i != table[index].end())
table[index].erase(i);
}

// function to display hash table
void Hash::displayHash() {
for (int i = 0; i < BUCKET; i++) {
cout << i;
for (auto x : table[i])
   cout << " --> " << x;
cout << endl;
}
}

// Driver program
int main()
{
// array that contains keys to be mapped
int a[] = {15, 11, 27, 8, 19, 16 ,17, 21, 26, 12};
int n = sizeof(a)/sizeof(a[0]);

// insert the keys into the hash table
```

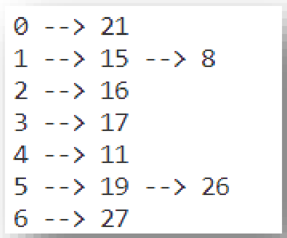
```
Hash h(7);    // 7 is count of buckets in
          // hash table
for (int i = 0; i < n; i++)
h.insertItem(a[i]);

// delete 12 from hash table
h.deleteItem(12);

// display the Hash table
h.displayHash();

return 0;
}
```

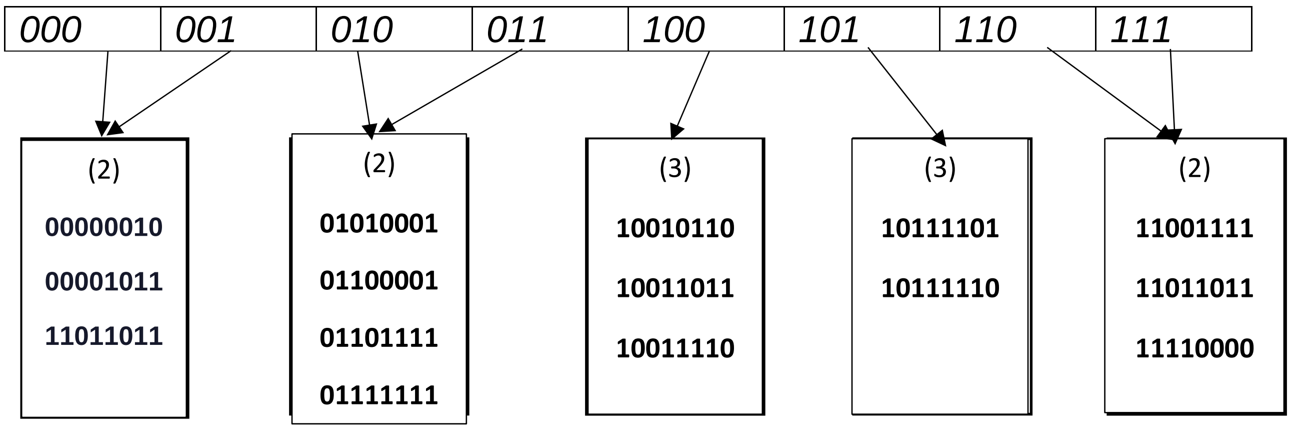
Result:



```
0 --> 21
1 --> 15 --> 8
2 --> 16
3 --> 17
4 --> 11
5 --> 19 --> 26
6 --> 27
```

[6]. Show the result of inserting the keys 10111101, 00000010, 10011011, 10111110, 01111111, 01010001, 10010110, 00001011, 11001111, 10011110, 11011011, 00101011, 01100001, 11110000, 01101111 into an initially empty extendible hashing data structure with M = 4.

Solution:



[7]. Using buckets of size 3 and a hash function of mod (key, 5) and bucket chaining enter the following records (only the key values are shown) into an empty traditional hash file. Create chains of buckets when needed.

42, 57, 16, 52, 66, 77, 12, 25, 21, 33, 32, 14

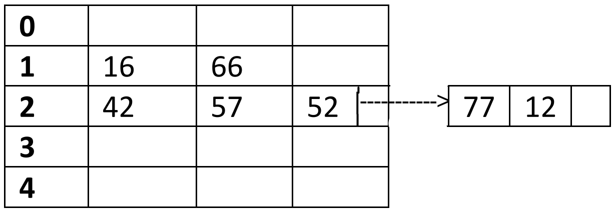
Solution:

Key	Mod 5
42	**2**
57	**2**
16	**1**
52	**2**
66	**1**
77	**2**
12	**2**
25	**0**
21	**1**
33	**3**
32	**2**
14	**4**

After Inserting 42,57,16,52,66:

0			
1	16	66	
2	42	57	52
3			
4			

After Inserting 77 12:



0			
1	16	66	
2	42	57	52
3			
4			

77	12	

After Inserting 25 21 33 32 14:

0	25		
1	16	66	21
2	42	57	52 -------->
3	33		
4	14		

77	12	32

[8]. Define and differentiate between Hash function and cryptographic hash function? What are the applications of randomized hashing?
Solution:

- Every cryptographic hash function is a hash function. But not every hash function is a cryptographic hash.
- A cryptographic hash function aims to guarantee a number of security properties. Most importantly that it's hard to find collisions or pre-images and that the output appears random.
- Non cryptographic hash functions just try to avoid collisions for non-malicious input. Some aim to detect accidental changes in data (CRCs), others try to put objects into different buckets in a hash table with as few collisions as possible.
- In exchange for weaker guarantees, they are typically (much) faster.
- I'd still call MD5 a cryptographic hash function, since it aimed to provide security. But it's broken, and thus no longer usable as a cryptographic hash. On the other hand, when you have a non-cryptographic hash function, you can't really call it "broken", since it never tried to be secure in the first place.