



EAST WEST UNIVERSITY

Department of Computer Science and Engineering

Report on

Double Hash Table

COURSE TITLE : Data Structure
COURSE CODE : CSE207
SEMESTER : Summer 2023
SECTION : 01

Submitted To

Name : Dr. Maheen Islam
Designation : Associate Professor
Department of Computer Science and Engineering

Submitted By

Name : Shishir Khondoker
ID NO : 2021-2-60-080
Name : Sakib Ibna
Sorowar Abir
ID NO: 2021-2-60-001
Department of Computer Science and Engineering

Submitted on 17th September 2023

Introduction





Hashing is used to map keys, values into the hash table by using a hash function. It is done for faster access to elements. A hash function maps a big value or string to a small integer that can be used as index in hash table. Since a hash function generates a small integer, there is possibility that two values result in same integer. This creates collision. To avoid collision, double hashing can be used. Double hashing is a collision resolving technique in Open Addressed Hash tables. Double hashing uses the idea of applying a second hash function when collision occurs. In double hashing, the both hash codes are used to find a free space by probing through array elements. Time complexity of Double Hash Table in both average and best case is $O(1)$ and in worst-case is $O(n)$.

Application of Double Hash table

- **Address Resolution Protocol (ARP) Cache** : ARP is a protocol used in computer networking to map an IP address to physical Media Access Control address on a local network . ARP caches are used to store these mapping temporarily to speed up future network communications within the same subnet.
- **Distributed Databases:** In distributed databases, data is stored across multiple servers or nodes. Double hashing can be used to distribute data evenly among these nodes, ensuring a balanced distribution of keys and efficient data retrieval.
- **Load Balancing:** Load balancers in network infrastructure use double hashing to evenly distribute incoming requests to multiple servers. This ensures that no single server becomes overwhelmed with traffic, improving system performance and reliability.
- **Cache Management:** In caching systems, double hashing can be used to manage the cache eviction policy. When a cache is full and needs to replace an item with a new one, double hashing can help determine which item to evict, optimizing cache performance.
- **File Systems:** Double hashing can be applied in file systems to handle file indexing and data block allocation efficiently. It helps reduce the likelihood of file data blocks colliding in the same storage location.
- **Database Indexing:** Double hashing can be used in database indexing structures, such as hash indexes, to optimize data retrieval. It helps reduce the occurrence of index collisions, leading to faster query execution.

- **Network Routing:** In computer networks, double hashing can be applied to determine the routing path for packets. This ensures that network traffic is evenly distributed across available routes, improving network efficiency and fault tolerance.
- **Task Scheduling:** Double hashing can be used in task scheduling algorithms to allocate resources or time slots to tasks or processes. It helps prevent resource contention and ensures efficient execution of tasks.
- **Peer-to-Peer (P2P) Networks:** P2P file-sharing networks can employ double hashing to distribute files across nodes in the network. This helps in efficient file retrieval and load balancing among participating peers.
- **Data Deduplication:** Double hashing can be used in data deduplication systems to identify and store unique data blocks efficiently. It ensures that duplicate data is recognized and eliminated, saving storage space.
- **Password Hashing:** In password security, double hashing can be applied to enhance password storage. It provides an additional layer of security by applying a second hashing function to the hash value of a password.
- **Online Gaming:** Online gaming platforms can use double hashing to allocate players to game servers. This helps distribute players evenly and reduces server overload during peak gaming times.
- **Distributed Hash Tables (DHTs):** DHTs are used in peer-to-peer networks for distributed data storage. Double hashing can be applied to improve the distribution of data across network nodes, ensuring balanced data access.

Properties:

-  Insert
-  Delete
-  Search
-  Display

Discussion:

A doubly hashed table, a variant of the traditional hash table, is designed to handle collisions more effectively by utilizing two separate hash functions. In this data structure, when multiple keys map to the same index during the initial hashing process, the second hash function is applied to determine an

alternative index. This process continues until an empty slot is found.

One significant advantage of a doubly hashed table is its ability to distribute keys more evenly, reducing the risk of clustering and maintaining efficient data retrieval operations. The choice of two different hash functions helps minimize collisions, resulting in a more balanced load distribution within the table.

This data structure's performance largely depends on the quality of the hash functions selected and the appropriate management of load factors. When implemented correctly, a doubly hashed table can achieve an average-case constant-time complexity ($O(1)$) for insertions, deletions, and retrievals. As such, it finds application in scenarios where fast and reliable data access is paramount, such as network routing, database systems, and distributed data storage.

Algorithm

In this program there is insert, delete, search, and display operations.

Insert:

```
insert(int key, int value)
{
    int hash1 = hashfunction1(key);
    int hash2 = hashfunction2(key);
    int index = hash1;
    struct data *new_item = (struct data*) malloc(sizeof(struct data));
    new_item->key = key; new_item->value = value;
    if (size == max) {
        printf("\n Hash Table is full, cannot insert more items \n");
        return;
    }
    while (array[index].flag == 1) {
        index = (index + hash2) % max;
        printf("\n probing \n");
    }
    array[index].item = new_item;
    array[index].flag = 1;
    size++;
    printf("\n Key (%d) has been inserted \n", key);
}
```

Delete:

```
Delete (int key)
{
    int hash1 = hashfunction1(key);
    int hash2 = hashfunction2(key);
    int index = hash1;
    if (size == 0)
    {
        printf("\n Hash Table is empty \n");
    }
}
```

```

        return;
    }
while (array[index].flag != 0)
{
    if (array[index].flag == 1 && array[index].item->key == key)
    {
        array[index].item = NULL;
        array[index].flag = 2;
        size--;
        printf("\n Key (%d) has been removed \n", key);
        return
    }
    index = (index + hash2) % max;
    printf("\n Key (%d) does not exist \n", key);
}

```

Search:

```

search(int key)
{
    int hash1 = hashfunction1(key);
    int hash2 = hashfunction2(key);
    int index = hash1;
    if (size == 0)
    {
        printf("\n Hash Table is empty \n");
        return;
    }
while (array[index].flag != 0)
{
    if (array[index].item->key == key)
    {
        printf("\n Array[%d] has elements \n Key (%d) and Value (%d) \n", index,
array[index].item->key, array[index].item->value);
        return;
    }
    index = (index + hash2) % max
    printf("\n Key (%d) does not exist \n", key);
}

```

Display:

```

display()
{
    int i;
    for (i = 0; i < max; i++)
    {
        if (array[i].flag != 1)
        {
            printf("\n Array[%d] has no elements \n", i);
        }
        else
        {
            printf("\n Array[%d] has elements \n Key (%d) and Value (%d) \n", i, array[i].item->key,
array[i].item->value);
        }
    }
}

```

Output:

```
Select C:\Users\Admin\Downloads\Project_DoubleHashTable.exe
Enter size of Array (must be >= 3) : 3
Largest prime number less than array size : 2

MENU

1.Inserting item in the Hash Table
2. Delete item from the Hash Table
3.Check the size of Hash Table
4.Display Hash Table
5.Search key in Hash Table
6. Exit

Enter your choice :1

Inserting element in Hash Table
Enter key and value : 10 20

Key (10) has been inserted

MENU

1.Inserting item in the Hash Table
2. Delete item from the Hash Table
3.Check the size of Hash Table
4.Display Hash Table
5.Search key in Hash Table
6. Exit

Enter your choice :1

Inserting element in Hash Table
Enter key and value : 30 40

Key (30) has been inserted

MENU

1.Inserting item in the Hash Table
2. Delete item from the Hash Table
3.Check the size of Hash Table
4.Display Hash Table
5.Search key in Hash Table
6. Exit

Enter your choice :1

Inserting element in Hash Table
Enter key and value : 50 60

Key (50) has been inserted
```

MENU

- 1.Inserting item in the Hash Table
2. Delete item from the Hash Table
- 3.Check the size of Hash Table
- 4.Display Hash Table
- 5.Search key in Hash Table
6. Exit

Enter your choice :4

Array[0] has elements
Key (30) and Value (40)

Array[1] has elements
Key (10) and Value (20)

Array[2] has elements
Key (50) and Value (60)

MENU

- 1.Inserting item in the Hash Table
2. Delete item from the Hash Table
- 3.Check the size of Hash Table
- 4.Display Hash Table
- 5.Search key in Hash Table
6. Exit

Enter your choice :3

Size of Hash Table is :3

MENU

- 1.Inserting item in the Hash Table
2. Delete item from the Hash Table
- 3.Check the size of Hash Table
- 4.Display Hash Table
- 5.Search key in Hash Table
6. Exit

Enter your choice :5

Searching in Hash Table
Enter the key to search :50

Array[2] has elements
Key (50) and Value (60)

MENU

- 1.Inserting item in the Hash Table
2. Delete item from the Hash Table
- 3.Check the size of Hash Table
- 4.Display Hash Table
- 5.Search key in Hash Table
6. Exit

Enter your choice :5

Searching in Hash Table
Enter the key to search :100

Key (100) does not exist

MENU

- 1.Inserting item in the Hash Table
2. Delete item from the Hash Table
- 3.Check the size of Hash Table
- 4.Display Hash Table
- 5.Search key in Hash Table
6. Exit

Enter your choice :2

Deleting in Hash Table

Enter the key to delete :10

Key (10) has been removed

MENU

- 1.Inserting item in the Hash Table
2. Delete item from the Hash Table
- 3.Check the size of Hash Table
- 4.Display Hash Table
- 5.Search key in Hash Table
6. Exit

Enter your choice :4

Array[0] has elements
Key (30) and Value (40)

Array[1] has no elements

Array[2] has elements
Key (50) and Value (60)

MENU

- 1.Inserting item in the Hash Table
2. Delete item from the Hash Table
- 3.Check the size of Hash Table
- 4.Display Hash Table
- 5.Search key in Hash Table
6. Exit

Enter your choice :5

Searching in Hash Table

Enter the key to search :10

Key (10) does not exist

MENU

- 1.Inserting item in the Hash Table
2. Delete item from the Hash Table
- 3.Check the size of Hash Table
- 4.Display Hash Table
- 5.Search key in Hash Table
6. Exit

Enter your choice :6

Process returned 0 (0x0) execution time : 103.777 s
Press any key to continue.

Conclusion

In our program, we take input the size of the array from user. But we cannot update the size of the array in one run time.

However, when it comes to the selection of a data structure for a specific use case, there are many factors to consider. Double hashing is useful if an application requires a smaller hash table since it effectively finds a free slot.

Whenever a collision occurs, choose another spot in table to put the value. The difference here is that instead of choosing next opening, a second hash function is used to determine the location of the next spot.