EAST WEST UNIVERSITY

Mini Project Report

Topic: Linear Probing Hash Table

Course Code: CSE207

Course Title: Data Structures

Submitted By:

Name : Akmal Hossain
ID : 2019-3-60-123
Section : 02

Submitted To:

Dr. Maheen Islam
Assosiate Proffessor
Department of Computer Science & Engineering
East West University

Introduction

Name	Akmal Hossain
ID	2019-3-60-123
Section	2
Department	Computer Science & Engineering
Topic	Linear Probing Hash Table

Problem Description

A hash table is a data structure used to implement an associative array, a structure that can map keys to values. A hash table uses a **hash function** to compute an index into an array of buckets or slots. Due to **collision of keys** while inserting elements into the hash table, idea of **Linear Probing** is used to probe the through the subsequent elements (looping back) of array **starting from hash code** value (index of the key) where key collision occurs.

Solution Analysis

Data Structures

For an open addressing solution, we can use an **array** as data structure where **linear probing** will avoid collisions and data might "**leak out**" of its preferred position.

Insertion

To insert an element x at a certain key, compute `hash(key)` and try to place x there. If that position is full, keep moving through the array, wrapping around at the end, until a free spot is found and place x there.

Searching

To search an element with key, compute `index = hash(key)` and start looking there. Move through the array until element is found or a blank spot is detected. A blank spot will indicate that key doesnt exist in hashtable.

Deletion

Deletions are a bit tricky because, we cannot just do search and remove, as that will result in blank spots in the middle of the array disjointing the hashtable elements.
When removing an element, we have to mark the index as previously occupied. This is known as a tombstone. So that when we lookup/search a key, we dont stop at a tombstone. Instead we keep the search going until an actual blank spot is found.
While inserting data, we can replace any tombstone we find as it is a free space.

Linear Probing in Practice

Advantages

- **Low memory usage:** Since we use an array and a hash function, this method has low memory usage.
- **Locality:** When collisions occurs, we only lookup in the adjacent locations. Therefore locality of the key/value pairs are better.

Disadvantages

- **Performance:** Linear probing has severe performance degradations when load factor gets high.
- **Primary clustering:** The number of collisions grow as a function of the number of existing collisions.

Algorithms

Insertion

```
function Insert(key, value):
    index = hash(key)
    while(array[index] is not empty):
        if(array[index] == key):
            update (key, value) pair
        else
            index = hash(index + 1)
        end if
    end loop
    insert (key, value) at array[index]
end function
```

Deletion

```
function Delete(key):
    index = hash(key)
    while(array[index] is full or tombstone):
        if(array[index] is full and key is equal):
            remove (key, value) pair
        else
            index = hash(index + 1)
        end if
    end loop
end function
```

Searching

```
function Search(key):
    index = hash(key)
    while(array[index] is full or tombstone):
        if(array[index] is full and key is equal):
            return (key, value) pair
        else
            index = hash(index + 1)
        end if
    end loop
end function
```

Conclusion

Linear probing hash table is a lightning fast hash table implementation. While this implementation of hash tables is very basic and impractical, it teaches us some fundamental theories of hashing methods and shows us another approach at implementation of a hash table data structure.