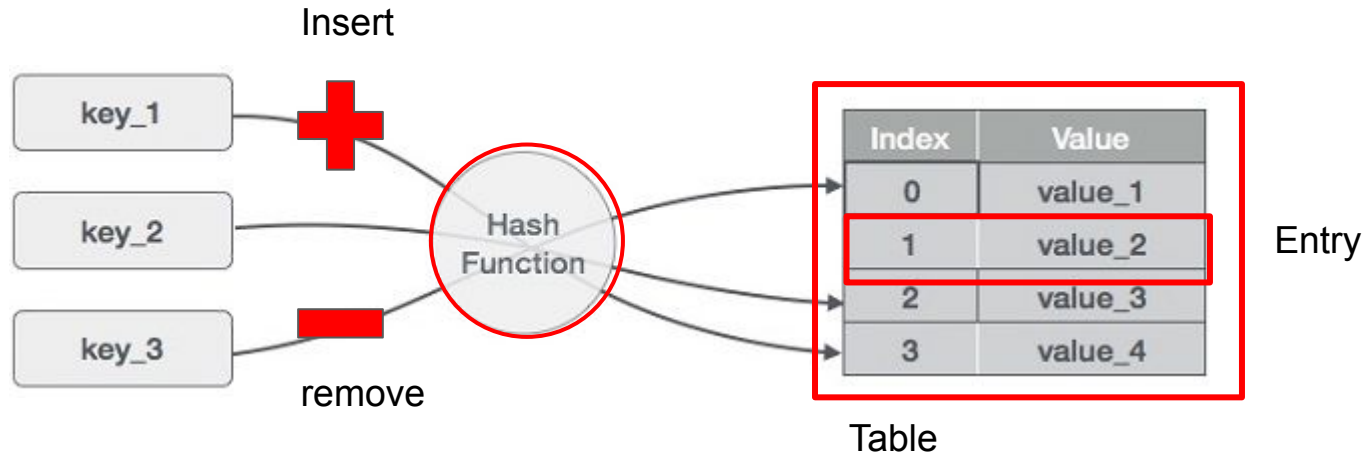


Session 9: Hash Tables

Data Structures and Algorithm 1 - Lab

Yahya Tawil
10 Dec 2021

Hash Table Implementation



Hash Table Implementation: Data Structure

```
class HashTableEntry {  
    public:  
        int k; // key  
        int v; // value  
        HashTableEntry(int k, int v) {  
            this->k = k;  
            this->v = v;  
        }  
};
```

Index	Value
0	value_1
1	value_2
2	value_3
3	value_4

Entry

Table

Hash Table Implementation: Data Structure

```
const int T_S = 200; // Table size
```

```
class HashMapTable {  
    private:  
        HashTableEntry **t;  
    public:  
        HashMapTable() {  
            t = new HashTableEntry * [T_S];  
            for (int i = 0; i < T_S; i++) {  
                t[i] = NULL;  
            }  
        }  
};
```

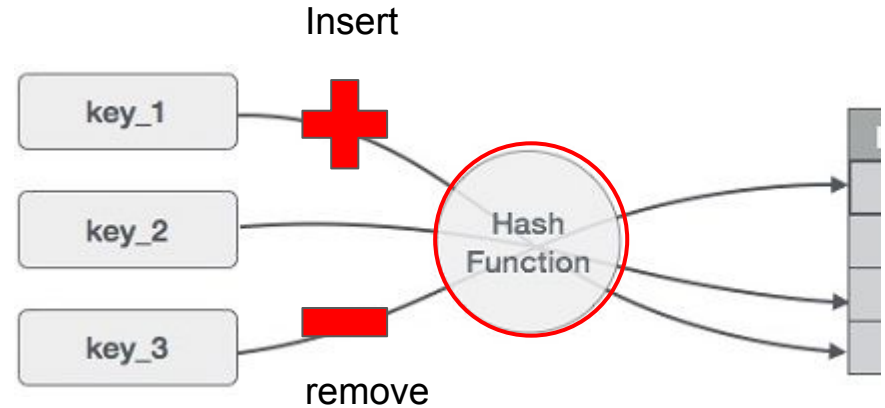
Index	Value
0	value_1
1	value_2
2	value_3
3	value_4

Entry

Table

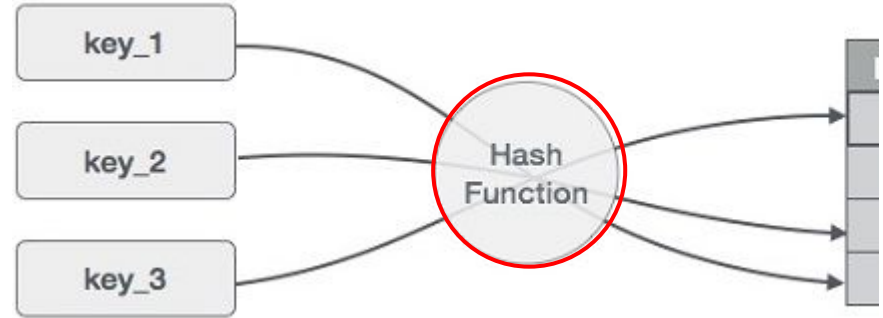
Hash Table Implementation: Data Structure

```
class HashMapTable {  
    private:  
        HashTableEntry **t;  
    public:  
        HashMapTable() {  
            t = new HashTableEntry * [T_S];  
        }  
        int HashFunc(int k);  
        void Insert(int k, int v);  
        int SearchKey(int k);  
        void Remove(int k);  
};
```



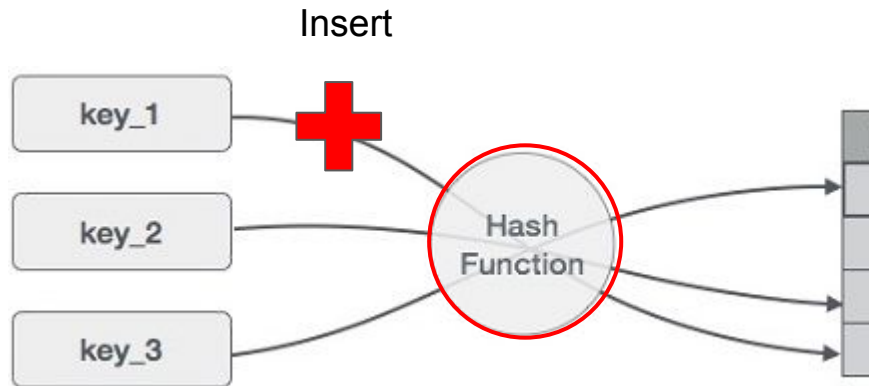
Hash Table Implementation: Hash Function

```
int HashFunc(int k) {  
    return k % T_S; // mod table size  
}
```



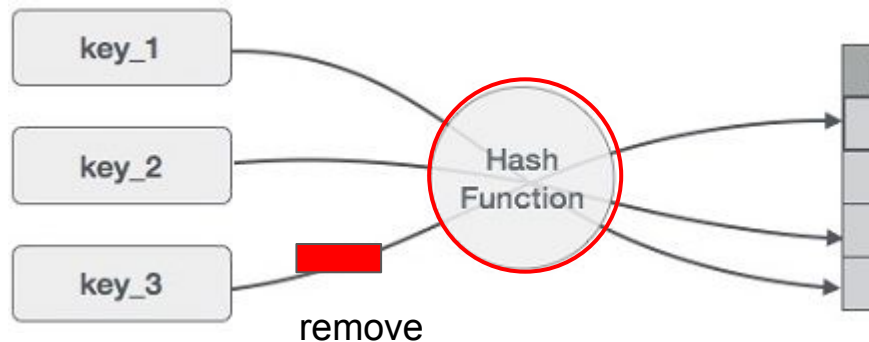
Hash Table Implementation: Insert

```
void Insert(int k, int v) {  
    int h = HashFunc(k);  
  
    while (t[h] != NULL && t[h]->k != k) {  
        h = HashFunc(h + 1);  
    }  
    if (t[h] != NULL)  
        delete t[h];  
    t[h] = new HashTableEntry(k, v);  
}
```



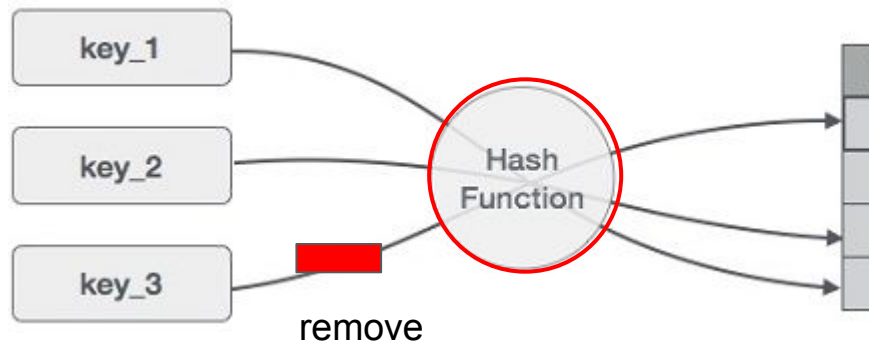
Hash Table Implementation: Remove

```
void Remove(int k) {  
    int h = HashFunc(k);  
    while (t[h] != NULL) {  
        if (t[h]->k == k)  
            break;  
  
        h = HashFunc(h + 1);  
    }  
    if (t[h] == NULL) {  
        return;  
    } else {  
        delete t[h];  
    }  
}
```



Hash Table Implementation: Search

```
int SearchKey(int k) {  
    int h = HashFunc(k);  
  
    while (t[h] != NULL && t[h]->k != k) {  
        h = HashFunc(h + 1);  
    }  
    if (t[h] == NULL)  
        return -1;  
    else  
        return t[h]->v;  
}
```



Hash Table Implementation: Discussions

- How to change the previous implementation to support integer keys and string values.
- How to change the previous implementation to support using user-defined hash function.

Hash Table Implementation: Discussions

- How to change the previous implementation to support integer keys and string values.

Possible solution:

```
class HashTableEntry {
public:
    int k; // key
    int v; // value
    string str_v;
    HashTableEntry(int k, int v) {
        this->k= k;
        this->v = v;
    }
    HashTableEntry(int k, string v) {
        this->k= k;
        this->str_v = v;
    }
};
```

Hash Table Implementation: Discussions

- How to change the previous implementation to support using user-defined hash function.

Possible solution: use Hash function as pointer to function

```
class HashMapTable {
private:
    HashTableEntry **t;
    int (*hf)(int) = nullptr;
}

void setHashFunc(int (*_hf)(int))
{
    hf = _hf;
}
```

```
int HashFunc(int k) {
    if(hf == nullptr) return k % T_S ;
    return hf(k); // mod table size
}

int myHashFunction(int key){
    cout<<"My hash Function"<<endl;
    return key % 10 ;
}
```

Hash Table Implementation: Discussions

- For the string hash functions, the size of the hash table limits the length of the string that can be hashed. (True or False ?)

Lab Project: Third week resources

Fit Coder

Sorted Array To Balanced BST

11, 22, 33, 44, 55, 66, 77, 88

index 0 1 2 3 4 5 6 7

```
graph TD
    44((44)) --> 22((22))
    44 --> 66((66))
    22 --> 11((11))
    22 --> 33((33))
    66 --> 55((55))
    66 --> 77((77))
    77 --> 88((88))
    11 --> NULL1[NULL]
    11 --> NULL2[NULL]
    33 --> NULL3[NULL]
    33 --> NULL4[NULL]
    55 --> NULL5[NULL]
    55 --> NULL6[NULL]
    77 --> NULL7[NULL]
    88 --> NULL8[NULL]
    88 --> NULL9[NULL]
```

Sandeep Thapar

```
TreeNode* arrayToBST (int start, int end)
{
    if (start > end)
        return NULL;
    mid = (start + end) / 2;
    node = Create new node of value arr[mid];
    node->left = arrayToBST (start, mid-1);
    node->right = arrayToBST (mid+1, end);
    return node;
}
```

Handwritten notes on the code: "Base condition." next to the if statement; "7" next to the mid calculation; "7" next to the mid-1 and mid+1 arguments; "8, 7" next to the final return statement.

Smaller form left subtree < Mid point < Greater form Right subtree

<https://www.youtube.com/watch?v=-FfprfvY4IQ>