LAPORAN PRAKTIKUM

MODUL 5
HASH TABLE

Disusun oleh:
Imelda Fajar Awalina Crisyanti
NIM: 2311102004

Dosen Pengampu:
Wahyu Andi Saputra, S.Pd., M.Eng

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
PURWOKERTO
2023

BAB I
TUJUAN PRAKTIKUM

a. Mahasiswa mampu menjelaskan definisi dan konsep dari Hash Code

b. Mahasiswa mampu menerapkan Hash Code kedalam pemrograman

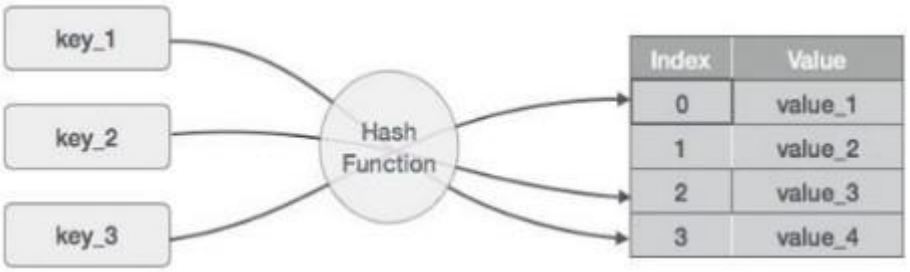
BAB II
DASAR TEORI

a. Pengertian Hash Table

Hash Table adalah struktur data yang mengorganisir data ke dalam pasangan kunci-nilai. Hash table biasanya terdiri dari dua komponen utama: array (atau vektor) dan fungsi hash. Hashing adalah teknik untuk mengubah rentang nilai kunci menjadi rentang indeks array.

Array menyimpan data dalam slot-slot yang disebut bucket. Setiap bucket dapat menampung satu atau beberapa item data. Fungsi hash digunakan untuk menghasilkan nilai unik dari setiap item data, yang digunakan sebagai indeks array. Dengan cara ini, hash table memungkinkan pencarian data dalam waktu yang konstan (O(1)) dalam kasus terbaik.

Sistem hash table bekerja dengan cara mengambil input kunci dan memetakkannya ke nilai indeks array menggunakan fungsi hash. Kemudian, data disimpan pada posisi indeks array yang dihasilkan oleh fungsi hash. Ketika data perlu dicari, input kunci dijadikan sebagai parameter untuk fungsi hash, danposisi indeks array yang dihasilkan digunakan untuk mencari data. Dalam kasus hash collision, di mana dua atau lebih data memiliki nilai hash yang sama, hash table menyimpan data tersebut dalam slot yang sama dengan Teknik yang disebutchaining.



b. Fungsi Hash Table

Fungsi hash membuat pemetaan antara kunci dan nilai, hal ini dilakukan melalui penggunaan rumus matematika yang dikenal sebagai fungsi hash. Hasil dari fungsi hash disebut sebagai nilai hash atau hash. Nilai hash adalah representasi dari string karakter asli tetapi biasanya lebih kecil dari aslinya.

c. Operasi Hash Table

1. **Insertion:**

   Memasukkan data baru ke dalam hash table dengan memanggil fungsi hash untuk menentukan posisi bucket yang tepat, dan kemudian menambahkan data ke bucket tersebut.

2. **Deletion:**

   Menghapus data dari hash table dengan mencari data menggunakan fungsi hash, dan kemudian menghapusnya dari bucket yang sesuai.

3. **Searching:**

   Mencari data dalam hash table dengan memasukkan input kunci ke fungsi hash untuk menentukan posisi bucket, dan kemudian mencari data di dalam bucket yang sesuai.

4. **Update:**

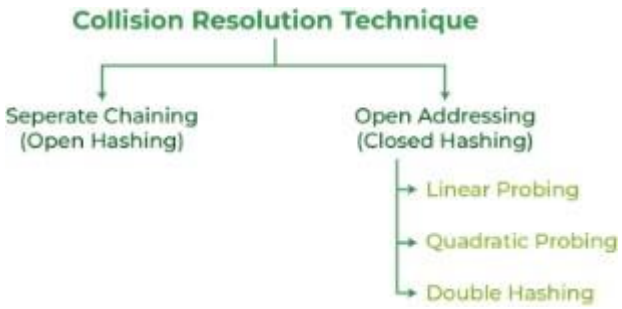
   Memperbarui data dalam hash table dengan mencari data menggunakan fungsi hash, dan kemudian memperbarui data yang ditemukan.

5. **Traversal:**

   Melalui seluruh hash table untuk memproses semua data yang ada dalam tabel.

d. Collision Resolution

Keterbatasan tabel hash adalah jika dua angka dimasukkan ke dalam fungsi hash menghasilkan nilai yang sama. Hal ini disebut dengan collision. Ada dua teknik untuk menyelesaikan masalah ini diantaranya :



6.Open Hashing (Chaining)

Metode chaining mengatasi collision dengan cara menyimpan semua item data dengan nilai indeks yang sama ke dalam sebuah linked list. Setiap node pada linked list merepresentasikan satu item data. Ketika ada pencarian atau penambahan item data, pencarian atau penambahan dilakukan pada linked list yang sesuai dengan indeks yang telah dihitung dari kunci yang di hash. Ketika linked list memiliki banyak node, pencarian atau penambahan item data menjadi lambat, karena harus mencari di seluruh linked list. Namun, chaining dapat mengatasi jumlah item data yang besar dengan efektif, karena keterbatasan array dihindari.

7. Closed Hashing

- **Linear Probing**

  Pada saat terjadi collision, maka akan mencari posisi yang kosong dibawah tempat terjadinya collision, jika masih penuh terus ke bawah, hingga ketemu tempat yang kosong. Jika tidak ada tempat yang kosong berarti HashTable sudah penuh**.**

- **Quadratic Probing**

  Penanganannya hampir sama dengan metode linear, hanya lompatannya tidak satu-satu, tetapi quadratic (12, 22, 32, 42, ...)

- **Double Hashing**

Pada saat terjadi collision, terdapat fungsi hash yang kedua untuk menentukan posisinya kembali.

BAB III

GUIDED

1. Guided 1

Source code

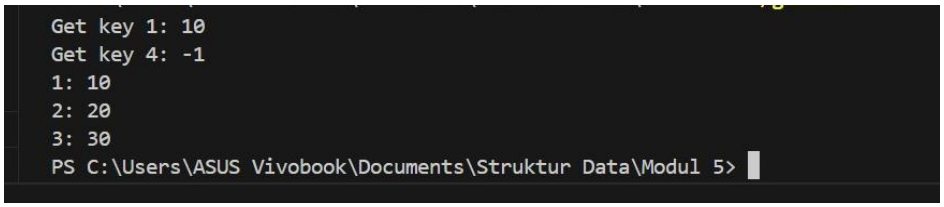
```
#include <iostream>
using namespace std;
const int MAX_SIZE = 10;
// Fungsi hash sederhana
int hash_func(int key)
{
    return key % MAX_SIZE;
}
// Struktur data untuk setiap node
struct Node
{
    int key;
    int value;
    Node *next;
    Node(int key, int value) : key(key), value(value),
                               next(nullptr) {}
};
// Class hash table
class HashTable
{
private:
    Node **table;

public:
    HashTable()
    {
        table = new Node *[MAX_SIZE]();
    }
    ~HashTable()
    {
        for (int i = 0; i < MAX_SIZE; i++)
        {
            Node *current = table[i];
            while (current != nullptr)
```

```
            {
                Node *temp = current;
                current = current->next;
                delete temp;
            }
        }
        delete[] table;
    }
    // Insertion
    void insert(int key, int value)
    {
        int index = hash_func(key);
        Node *current = table[index];
        while (current != nullptr)
        {
            if (current->key == key)
            {
                current->value = value;
                return;
            }
            current = current->next;
        }
        Node *node = new Node(key, value);
        node->next = table[index];
        table[index] = node;
    }
    // Searching
    int get(int key)
    {
        int index = hash_func(key);
        Node *current = table[index];
        while (current != nullptr)
        {
            if (current->key == key)
            {
                return current->value;
            }
            current = current->next;
        }
        return -1;
    }
```

```
    // Deletion
    void remove(int key)
    {
        int index = hash_func(key);
        Node *current = table[index];
        Node *prev = nullptr;
        while (current != nullptr)
        {
            if (current->key == key)
            {
                if (prev == nullptr)
                {
                    table[index] = current->next;
                }
                else
                {
                    prev->next = current->next;
                }
                delete current;
                return;
            }
            prev = current;
            current = current->next;
        }
    }
    // Traversal
    void traverse()
    {
        for (int i = 0; i < MAX_SIZE; i++)
        {
            Node *current = table[i];
            while (current != nullptr)
            {
                cout << current->key << ": " << current->value
                     << endl;
                current = current->next;
            }
        }
    }
};
int main()
```

```
{
    HashTable ht;
    // Insertion
    ht.insert(1, 10);
    ht.insert(2, 20);
    ht.insert(3, 30);
    // Searching
    cout << "Get key 1: " << ht.get(1) << endl;
    cout << "Get key 4: " << ht.get(4) << endl;
    // Deletion
    ht.remove(4);
    // Traversal
    ht.traverse();
    return 0;
}
```

Screenshoot program



Deskripsi program

Program ini mengimplementasikan struktur data bernama Hash Table dalam C++. Hash Table digunakan untuk menyimpan pasangan key-value secara efisien. Key digunakan untuk mencari value terkait dengan cepat.

- **Hash Function:** Fungsi ini mengubah key menjadi sebuah index di dalam Hash Table.
- **Node:** Struktur data yang menyimpan key, value, dan pointer ke node selanjutnya (digunakan untuk menangani tabrakan).
- **HashTable Class:** Kelas yang merepresentasikan Hash Table itu sendiri. Ia memiliki array of Node pointers untuk menyimpan node-node berdasarkan index yang dihasilkan oleh Hash Function.

Program mendefinisikan ukuran Hash Table (MAX_SIZE) dan fungsi Hash sederhana. Struktur Node dibuat untuk menyimpan key, value, dan pointer ke node

selanjutnya. HashTable Class dibuat untuk mengelola Hash Table, termasuk alokasi memori dan operasi seperti insert, get, remove, dan traverse. Fungsi main digunakan untuk membuat Hash Table, melakukan insert beberapa key-value, mencari nilai berdasarkan key, menghapus key tertentu (walaupun tidak ada key 4), dan menampilkan seluruh isi Hash Table.

2. Guided 2

Source code

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;
const int TABLE_SIZE = 11;
string name;
string phone_number;
class HashNode
{
public:
    string name;
    string phone_number;
    HashNode(string name, string phone_number)
    {
        this->name = name;
        this->phone_number = phone_number;
    }
};
class HashMap
{
private:
    vector<HashNode *> table[TABLE_SIZE];

public:
    int hashFunc(string key)
    {
        int hash_val = 0;
        for (char c : key)
        {
            hash_val += c;
        }
```

```
        return hash_val % TABLE_SIZE;
    }
    void insert(string name, string phone_number)
    {
        int hash_val = hashFunc(name);
        for (auto node : table[hash_val])
        {
            if (node->name == name)
            {
                node->phone_number = phone_number;
                return;
            }
        }
        table[hash_val].push_back(new HashNode(name,
                                               phone_number));
    }
    void remove(string name)
    {
        int hash_val = hashFunc(name);
        for (auto it = table[hash_val].begin(); it !=
                                                table[hash_val].e
nd();
             it++)
        {
            if ((*it)->name == name)
            {
                table[hash_val].erase(it);
                return;
            }
        }
    }
    string searchByName(string name)
    {
        int hash_val = hashFunc(name);
        for (auto node : table[hash_val])
        {
            if (node->name == name)
            {
                return node->phone_number;
            }
        }
```

```
        return "";
    }
    void print()
    {
        for (int i = 0; i < TABLE_SIZE; i++)
        {
            cout << i << ": ";
            for (auto pair : table[i])
            {
                if (pair != nullptr)
                {
                    cout << "[" << pair->name << ", " << pair-
>phone_number << "]";
                }
            }
            cout << endl;
        }
    }
};
int main()
{
    HashMap employee_map;
    employee_map.insert("Mistah", "1234");
    employee_map.insert("Pastah", "5678");
    employee_map.insert("Ghana", "91011");
    cout << "Nomer Hp Mistah : "
         << employee_map.searchByName("Mistah") << endl;
    cout << "Phone Hp Pastah : "
         << employee_map.searchByName("Pastah") << endl;
    employee_map.remove("Mistah");
    cout << "Nomer Hp Mistah setelah dihapus : "
         << employee_map.searchByName("Mistah") << endl
         << endl;
    cout << "Hash Table : " << endl;
    employee_map.print();
    return 0;
}
```

Screenshoot program

Deskripsi program

Program ini adalah program sederhana untuk menyimpan dan mencari data kontak berupa nama dan nomor telepon. Program ini menggunakan struktur data yang disebut Hash Table untuk menyimpan data kontak tersebut.

Program ini mendefinisikan sebuah class bernama HashMap yang memiliki fungsi untuk menyimpan, mencari, dan menghapus data kontak. Data kontak disimpan dalam bentuk objek HashNode yang berisi nama dan nomor telepon. Fungsi hashFunc digunakan untuk menentukan posisi penyimpanan data kontak dalam Hash Table berdasarkan nama kontak. Fungsi insert digunakan untuk menyimpan data kontak baru ke dalam Hash Table. Fungsi searchByName digunakan untuk mencari nomor telepon berdasarkan nama kontak. Fungsi remove digunakan untuk menghapus data kontak dari Hash Table. Fungsi print digunakan untuk menampilkan isi dari Hash Table.

Program ini menyimpan 3 data kontak: Mistah (1234), Pastah (5678), dan Ghana (91011). Program kemudian mencari nomor telepon Mistah dan Pastah. Program kemudian menghapus data kontak Mistah.Terakhir, program menampilkan isi dari Hash Table setelah penghapusan.

LATIHAN KELAS - UNGUIDED

1. **Unguided 1**

   Source code

```cpp
#include <iostream>
#include <vector>
using namespace std;

const int MAX_SIZE = 10;

int hash_func(int key)
{
    return key % MAX_SIZE;
}

struct Node
{
    int nim;
    int nilai;
    Node *next;
    Node(int nim, int nilai) : nim(nim), nilai(nilai),
next(nullptr) {}
};

class HashTable
{
private:
    Node **table;

public:
    HashTable()
    {
        table = new Node *[MAX_SIZE]();
    }

    ~HashTable()
    {
        for (int i = 0; i < MAX_SIZE; i++)
        {
```

```
            Node *current = table[i];
            while (current != nullptr)
            {
                Node *temp = current;
                current = current->next;
                delete temp;
            }
        }
        delete[] table;
    }

    void insert(int nim, int nilai)
    {
        int index = hash_func(nim);
        Node *current = table[index];
        while (current != nullptr)
        {
            if (current->nim == nim)
            {
                current->nilai = nilai;
                return;
            }
            current = current->next;
        }
        Node *node = new Node(nim, nilai);
        node->next = table[index];
        table[index] = node;
    }

    int get(int nim)
    {
        int index = hash_func(nim);
        Node *current = table[index];
        while (current != nullptr)
        {
            if (current->nim == nim)
            {
                return current->nilai;
            }
            current = current->next;
        }
```

```
        return -1;
    }

    void remove(int nim)
    {
        int index = hash_func(nim);
        Node *current = table[index];
        Node *prev = nullptr;
        while (current != nullptr)
        {
            if (current->nim == nim)
            {
                if (prev == nullptr)
                {
                    table[index] = current->next;
                }
                else
                {
                    prev->next = current->next;
                }
                delete current;
                return;
            }
            prev = current;
            current = current->next;
        }
    }

    void search_by_value_range(int min_val, int max_val)
    {
        for (int i = 0; i < MAX_SIZE; i++)
        {
            Node *current = table[i];
            while (current != nullptr)
            {
                if (current->nilai >= min_val && current->nilai
<= max_val)
                {
                    cout << "NIM: " << current->nim << ", Nilai:
" << current->nilai << endl;
                }
```

```
                current = current->next;
            }
        }
    }

    void traverse()
    {
        for (int i = 0; i < MAX_SIZE; i++)
        {
            Node *current = table[i];
            while (current != nullptr)
            {
                cout << "NIM: " << current->nim << ", Nilai: " <<
current->nilai << endl;
                current = current->next;
            }
        }
    }
};

void display_menu()
{
    cout << "1. Tambah Data Mahasiswa" << endl;
    cout << "2. Hapus Data Mahasiswa" << endl;
    cout << "3. Cari Data Mahasiswa Berdasarkan NIM" << endl;
    cout << "4. Cari Data Mahasiswa Berdasarkan Rentang Nilai
(80-90)" << endl;
    cout << "5. Tampilkan Semua Data Mahasiswa" << endl;
    cout << "6. Keluar" << endl;
}

int main()
{
    HashTable ht;
    int choice;
    do
    {
        display_menu();
        cout << "Pilihan: ";
        cin >> choice;
        switch (choice)
```

```
        {
        case 1:
        {
            int nim, nilai;
            cout << "Masukkan NIM: ";
            cin >> nim;
            cout << "Masukkan Nilai: ";
            cin >> nilai;
            ht.insert(nim, nilai);
            break;
        }
        case 2:
        {
            int nim;
            cout << "Masukkan NIM yang akan dihapus: ";
            cin >> nim;
            ht.remove(nim);
            break;
        }
        case 3:
        {
            int nim;
            cout << "Masukkan NIM yang akan dicari: ";
            cin >> nim;
            int nilai = ht.get(nim);
            if (nilai != -1)
            {
                cout << "NIM: " << nim << ", Nilai: " << nilai <<
endl;
            }
            else
            {
                cout << "Data tidak ditemukan." << endl;
            }
            break;
        }
        case 4:
        {
            cout << "Mahasiswa dengan nilai antara 80 dan 90:" <<
endl;
            ht.search_by_value_range(80, 90);
```

```
            break;
        }
        case 5:
        {
            cout << "Semua Data Mahasiswa:" << endl;
            ht.traverse();
            break;
        }
        case 6:
        {
            cout << "Keluar." << endl;
            break;
        }
        default:
            cout << "Pilihan tidak valid." << endl;
        }
    } while (choice != 6);

    return 0;
}
```

Screenshoot program

- **Tampilan Menu**

```
1. Tambah Data Mahasiswa
2. Hapus Data Mahasiswa
3. Cari Data Mahasiswa Berdasarkan NIM
4. Cari Data Mahasiswa Berdasarkan Rentang Nilai (80-90)
5. Tampilkan Semua Data Mahasiswa
6. Keluar
Pilihan:
```

- **Tambah Data Mahasiswa**

```
1. Tambah Data Mahasiswa
2. Hapus Data Mahasiswa
3. Cari Data Mahasiswa Berdasarkan NIM
4. Cari Data Mahasiswa Berdasarkan Rentang Nilai (80-90)
5. Tampilkan Semua Data Mahasiswa
6. Keluar
Pilihan: 1
Masukkan NIM: 004
Masukkan Nilai: 85
```

- **Hapus Data Mahasiswa**

```
1. Tambah Data Mahasiswa
2. Hapus Data Mahasiswa
3. Cari Data Mahasiswa Berdasarkan NIM
4. Cari Data Mahasiswa Berdasarkan Rentang Nilai (80-90)
5. Tampilkan Semua Data Mahasiswa
6. Keluar
Pilihan: 2
Masukkan NIM yang akan dihapus: 005
1. Tambah Data Mahasiswa
2. Hapus Data Mahasiswa
3. Cari Data Mahasiswa Berdasarkan NIM
4. Cari Data Mahasiswa Berdasarkan Rentang Nilai (80-90)
5. Tampilkan Semua Data Mahasiswa
6. Keluar
Pilihan: 5
Semua Data Mahasiswa:
NIM: 4, Nilai: 85
NIM: 6, Nilai: 92
NIM: 7, Nilai: 90
```

- **Cari Mahasiswa Berdasarkan NIM**

```
1. Tambah Data Mahasiswa
2. Hapus Data Mahasiswa
3. Cari Data Mahasiswa Berdasarkan NIM
4. Cari Data Mahasiswa Berdasarkan Rentang Nilai (80-90)
5. Tampilkan Semua Data Mahasiswa
6. Keluar
Pilihan: 3
Masukkan NIM yang akan dicari: 004
NIM: 4, Nilai: 85
```

- **Cari Mahasiswa Berdasarkan Rentang Nilai (80-90)**

```
1. Tambah Data Mahasiswa
2. Hapus Data Mahasiswa
3. Cari Data Mahasiswa Berdasarkan NIM
4. Cari Data Mahasiswa Berdasarkan Rentang Nilai (80-90)
5. Tampilkan Semua Data Mahasiswa
6. Keluar
Pilihan: 4
Mahasiswa dengan nilai antara 80 dan 90:
NIM: 4, Nilai: 85
NIM: 7, Nilai: 90
```

- **Keluar**

```
1. Tambah Data Mahasiswa
2. Hapus Data Mahasiswa
3. Cari Data Mahasiswa Berdasarkan NIM
4. Cari Data Mahasiswa Berdasarkan Rentang Nilai (80-90)
5. Tampilkan Semua Data Mahasiswa
6. Keluar
Pilihan: 6
Keluar.
```

Deskripsi program

Program di atas adalah implementasi dari hash table sederhana yang digunakan untuk menyimpan data mahasiswa berdasarkan NIM (Nomor Induk Mahasiswa) dan nilai mereka. Program ini memiliki beberapa fitur utama yang diatur dalam bentuk menu interaktif. Berikut deskripsi singkat dari program tersebut:

Struktur dan Fungsi Utama

Konstanta dan Fungsi Hash

MAX_SIZE: Konstanta yang menentukan ukuran maksimum hash table.

hash_func: Fungsi hash yang menghitung indeks dari NIM menggunakan operasi modulus (%).

Struktur Node

Node: Struktur data yang merepresentasikan setiap entri dalam hash table, menyimpan NIM, nilai, dan pointer ke node berikutnya (untuk mengatasi kolisi menggunakan linked list).

Kelas HashTable

Konstruktor: Menginisialisasi hash table dengan ukuran MAX_SIZE.

Destruktor: Membersihkan memori yang digunakan oleh hash table.

insert: Menambahkan data mahasiswa ke dalam hash table. Jika NIM sudah ada, nilai diperbarui.

get: Mencari nilai berdasarkan NIM yang diberikan.

remove: Menghapus data mahasiswa berdasarkan NIM.

search_by_value_range: Mencari dan menampilkan mahasiswa yang memiliki nilai dalam rentang tertentu (80-90).

traverse: Menampilkan semua data mahasiswa yang tersimpan dalam hash table.

Fungsi Menu

display_menu: Menampilkan opsi menu kepada pengguna.

Fungsi main

Menggunakan do-while loop untuk menampilkan menu dan menerima input dari pengguna.

Menangani pilihan pengguna dan memanggil metode yang sesuai pada objek HashTable.

Alur Program

Tambah Data Mahasiswa:

Pengguna memasukkan NIM dan nilai untuk menambahkan atau memperbarui data mahasiswa.

Hapus Data Mahasiswa:

Pengguna memasukkan NIM untuk menghapus data mahasiswa yang sesuai.

Cari Data Berdasarkan NIM:

Pengguna memasukkan NIM untuk mencari nilai yang sesuai dan menampilkannya.

Cari Data Berdasarkan Rentang Nilai:

Program mencari dan menampilkan mahasiswa yang memiliki nilai dalam rentang 80-90.

Tampilkan Semua Data Mahasiswa:

Menampilkan semua data mahasiswa yang tersimpan dalam hash table.

Keluar:

Mengakhiri program.

Contoh Penggunaan

Pengguna dapat menambahkan, menghapus, mencari, dan menampilkan data mahasiswa dengan memilih opsi yang tersedia di menu.

Program terus berjalan hingga pengguna memilih opsi "Keluar".

Program ini menggunakan teknik dasar hash table dengan penanganan kolisi menggunakan chaining (linked list). Hal ini memungkinkan penyimpanan dan pencarian data yang efisien.

BAB IV

KESIMPULAN

Hash table atau tabel hash adalah struktur data yang memungkinkan pengindeksan dan pencarian data secara efisien. Hash table menggunakan fungsi hash untuk mengonversi kunci data menjadi indeks tabel. Pada C++, implementasi hash table dapat menggunakan struktur data berupa array dan linked list. Pada program di atas, hash table diimplementasikan menggunakan array dari linked list. Hash table sangat berguna dalam pengolahan data yang besar, seperti pada database atau aplikasi yang membutuhkan pencarian data yang cepat dan efisien. Selain itu, di modul kali ini kita juga berkesempatan untuk mengaplikasikan beberapa operasi hash table yang umum digunakan seperti Insert (Operasi ini digunakan untuk menambahkan elemen baru), Get (Operasi ini digunakan untuk mengakses nilai data yang terkait dengan kunci tertentu dalam hash table), Remove (Operasi ini digunakan untuk menghapus elemen dengan kunci tertentu dari hash table), dan Iterate (Operasi ini digunakan untuk mengakses seluruh elemen dalam hash table secara berurutan).