LAPORAN PRAKTIKUM

MODUL V
HASH TABLE

Disusun oleh:
Nadhif Atha Zaki
NIM: 2311102007

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

1. Mahasiswa mampu menjelaskan definisi dan konsep dari Hash Code2. Praktikan dapat membuat linked list circular dan non circular.

2. Mahasiswa mampu menerapkan Hash Code kedalam pemrograman

BAB II

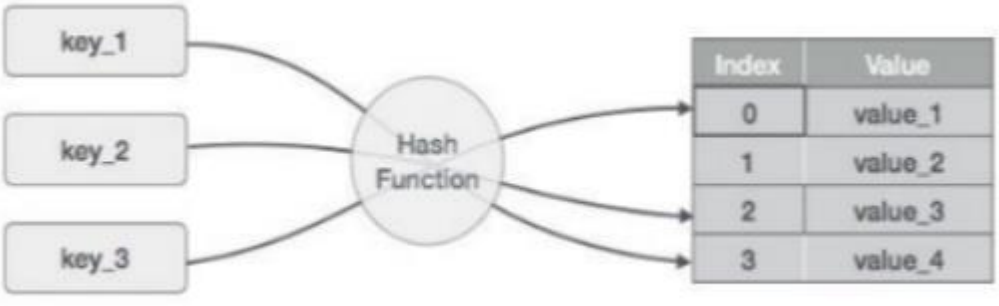
DASAR TEORI

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, dan posisi 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 disebut chaining.



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.

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:

```
PS D:\Nathhh\matkul\smst 2\praktikum struktur data\modul 5> cd
Get key 1: 10
Get key 4: -1
1: 10
2: 20
3: 30
PS D:\Nathhh\matkul\smst 2\praktikum struktur data\modul 5>
```

Deskripsi program:

Program ini merupakan implementasi sederhana dari struktur data hash table menggunakan bahasa pemrograman C++.

Hash table digunakan untuk menyimpan data dengan kunci dan nilai dalam larik yang diindeks oleh fungsi hash sederhana.

Setiap node dalam hash table memiliki kunci dan nilai, dan jika terjadi tabrakan hash (collision), node akan disisipkan dalam bentuk linked list.

Program dapat melakukan operasi penyisipan, pencarian, dan penghapusan data dari hash table.

Penyisipan data dilakukan dengan memasukkan pasangan kunci-nilai ke dalam hash table, diindeks berdasarkan fungsi hash dari kunci.

Pencarian data dilakukan dengan mencari kunci tertentu dalam hash table dan mengembalikan nilai yang sesuai jika ditemukan.

Penghapusan data dilakukan dengan menghapus node yang sesuai dengan kunci yang diberikan dari hash table.

Program juga memiliki fungsi untuk menelusuri semua data yang disimpan dalam hash table.

Dalam contoh ini, hash table digunakan untuk menyimpan pasangan kunci-nilai integer, namun dapat disesuaikan untuk tipe data yang lain.

Program ini memberikan contoh penggunaan hash table untuk operasi dasar seperti penyisipan, pencarian, dan penghapusan data.

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;
```

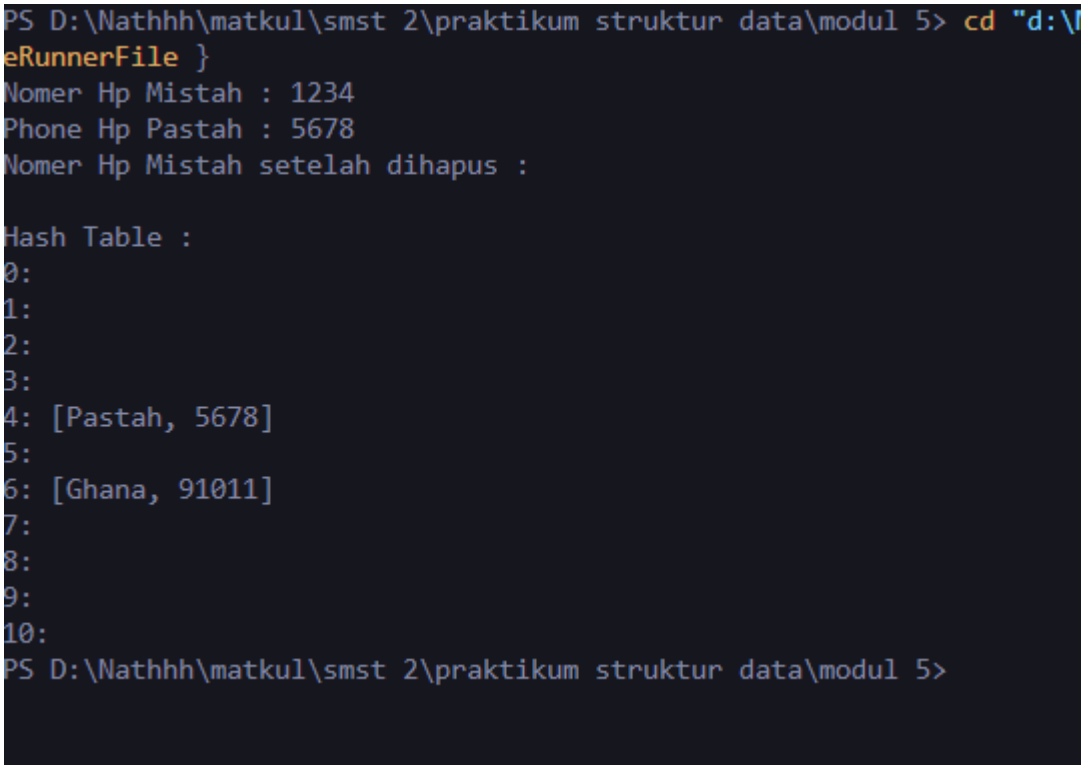
```
    }
};
class HashMap
{
private:
    vector<HashNode *> table[TABLE_SIZE];

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 merupakan implementasi sederhana dari hash map yang digunakan untuk menyimpan dan mengelola pasangan nama dan nomor telepon karyawan.
Setiap pasangan nama dan nomor telepon disimpan dalam struktur data hash node yang terindeks berdasarkan fungsi hash dari nama.
Program menyediakan fungsi untuk menyisipkan data baru, menghapus data berdasarkan nama, dan mencari nomor telepon berdasarkan nama karyawan.
Operasi penyisipan dilakukan dengan memasukkan pasangan nama dan nomor telepon ke dalam tabel hash.
Operasi pencarian dilakukan dengan mencari nama karyawan tertentu dalam tabel hash dan mengembalikan nomor telepon yang sesuai.
Operasi penghapusan dilakukan dengan menghapus node yang sesuai dengan nama karyawan dari tabel hash.
Program juga menyediakan fungsi untuk mencetak isi tabel hash, menampilkan pasangan nama dan nomor telepon yang disimpan.
Dalam contoh ini, hash map digunakan untuk menyimpan data karyawan, namun dapat disesuaikan untuk keperluan lain.
Program ini memberikan contoh penggunaan hash map untuk operasi dasar seperti penyisipan, pencarian, dan penghapusan data.

LATIHAN KELAS - UNGUIDED

1.

Implementasikan hash table untuk menyimpan data mahasiswa. Setiap mahasiswa memiliki NIM dan nilai. Implementasikan fungsi untuk menambahkan data baru, menghapus data, mencari data berdasarkan NIM, dan mencari data berdasarkan nilai. Dengan ketentuan :

a. Setiap mahasiswa memiliki NIM dan nilai.
b. Program memiliki tampilan pilihan menu berisi poin C.
c. Implementasikan fungsi untuk menambahkan data baru, menghapus data, mencari data berdasarkan NIM, dan mencari data berdasarkan rentang nilai (80 – 90).

Source code

```
#include <iostream>
#include <vector>

using namespace std;

// Struktur untuk menyimpan data mahasiswa
struct Mahasiswa
{
    long long nim;
    string nama;
    int nilai;
    Mahasiswa(long long n, string nm, int v) : nim(n), nama(nm),
nilai(v) {}
};

// Kelas HashTable untuk implementasi hash table
class HashTable
{
private:
    static const int TABLE_SIZE = 100;
    vector<Mahasiswa *> table[TABLE_SIZE];
```

```
    // Fungsi hash sederhana
    int hashFunction(long long key)
    {
        return key % TABLE_SIZE;
    }

public:
    // Fungsi untuk menambahkan data mahasiswa ke hash table
    void insert(long long nim, string nama, int nilai)
    {
        int index = hashFunction(nim);
        Mahasiswa *mahasiswa = new Mahasiswa(nim, nama, nilai);
        table[index].push_back(mahasiswa);
    }

    // Fungsi untuk mencari data mahasiswa berdasarkan NIM
    Mahasiswa *searchByNIM(long long nim)
    {
        int index = hashFunction(nim);
        for (Mahasiswa *m : table[index])
        {
            if (m->nim == nim)
                return m;
        }
        return nullptr;
    }

    // Fungsi untuk menghapus data mahasiswa berdasarkan NIM
    void remove(long long nim)
    {
        int index = hashFunction(nim);
        for (auto it = table[index].begin(); it !=
table[index].end(); ++it)
        {
            if ((*it)->nim == nim)
            {
                delete *it;
                table[index].erase(it);
                break;
            }
        }
```

```
    }

    // Fungsi untuk mencari data mahasiswa berdasarkan rentang
nilai (80-90)
    vector<Mahasiswa *> searchByRange()
    {
        vector<Mahasiswa *> result;
        for (int i = 0; i < TABLE_SIZE; ++i)
        {
            for (Mahasiswa *m : table[i])
            {
                if (m->nilai >= 80 && m->nilai <= 90)
                    result.push_back(m);
            }
        }
        return result;
    }
};

int main()
{
    HashTable hashTable;
    int choice;

    do
    {
        cout << "\nMenu:\n";
        cout << "1. Tambah Data Mahasiswa\n";
        cout << "2. Hapus Data Mahasiswa\n";
        cout << "3. Cari Data Mahasiswa berdasarkan NIM\n";
        cout << "4. Cari Data Mahasiswa berdasarkan Rentang Nilai
(80-90)\n";
        cout << "5. Keluar\n";
        cout << "Pilih menu: ";
        cin >> choice;

        switch (choice)
        {
        case 1:
        {
            long long nim;
```

```
            int nilai;
            string nama;
            cout << "Masukkan NIM (10 digit): ";
            cin >> nim;
            cout << "Masukkan Nama: ";
            cin.ignore();
            getline(cin, nama);
            cout << "Masukkan Nilai: ";
            cin >> nilai;
            hashTable.insert(nim, nama, nilai);
            cout << "Data mahasiswa berhasil ditambahkan.\n";
            break;
        }
        case 2:
        {
            long long nim;
            cout << "Masukkan NIM mahasiswa yang akan dihapus: ";
            cin >> nim;
            hashTable.remove(nim);
            cout << "Data mahasiswa berhasil dihapus.\n";
            break;
        }
        case 3:
        {
            long long nim;
            cout << "Masukkan NIM mahasiswa yang ingin dicari: ";
            cin >> nim;
            Mahasiswa *m = hashTable.searchByNIM(nim);
            if (m != nullptr)
                cout << "Data ditemukan - NIM: " << m->nim << ",
Nama: " << m->nama << ", Nilai: " << m->nilai << endl;
            else
                cout << "Data tidak ditemukan.\n";
            break;
        }
        case 4:
        {
            vector<Mahasiswa *> result =
hashTable.searchByRange();
            if (result.empty())
            {
```

```
                cout << "Tidak ada mahasiswa dengan nilai dalam
rentang 80-90.\n";
            }
            else
            {
                cout << "Mahasiswa dengan nilai dalam rentang 80-
90:\n";
                for (Mahasiswa *m : result)
                {
                    cout << "NIM: " << m->nim << ", Nama: " << m-
>nama << ", Nilai: " << m->nilai << endl;
                }
            }
            break;
        }
        case 5:
        {
            cout << "Terima kasih!\n";
            break;
        }
        default:
            cout << "Pilihan tidak valid. Silakan pilih
kembali.\n";
        }
    } while (choice != 5);

    return 0;
}
```

Screenshoot program

```
PS D:\Nathhh\matkul\smst 2\praktikum struktur data\modul 5> cd "

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. Keluar
Pilih menu: 1
Masukkan NIM (10 digit): 2311102007
Masukkan Nama: nadhif
Masukkan Nilai: 90
Data mahasiswa berhasil ditambahkan.

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. Keluar
Pilih menu: 1
Masukkan NIM (10 digit): 2311102007
Masukkan Nama: atha
Masukkan Nilai: 85
Data mahasiswa berhasil ditambahkan.

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. Keluar
Pilih menu: 3
Masukkan NIM mahasiswa yang ingin dicari: 2311102007
Data ditemukan - NIM: 2311102007, Nama: nadhif, Nilai: 90

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. Keluar
Pilih menu: 4
Mahasiswa dengan nilai dalam rentang 80-90:
NIM: 2311102007, Nama: nadhif, Nilai: 90
NIM: 2311102007, Nama: atha, Nilai: 85
```

Deskripsi dan Penjelasan Program:

Program di atas adalah implementasi dari hash table dalam bahasa pemrograman C++ untuk menyimpan data mahasiswa. Hash table digunakan untuk mempercepat pencarian dan penyisipan data dengan menggunakan fungsi hash.

Struktur data `Mahasiswa` digunakan untuk menyimpan informasi mahasiswa, yaitu NIM, nama, dan nilai. Ini memberikan fleksibilitas dalam menyimpan data mahasiswa.

Kelas `HashTable` merupakan implementasi hash table yang menggunakan vektor dari pointer ke objek `Mahasiswa`. Ini memungkinkan untuk menyimpan banyak mahasiswa dalam satu slot hash.

Setiap objek `Mahasiswa` disisipkan ke dalam tabel hash berdasarkan fungsi hash dari NIM-nya. Ini memastikan akses data yang efisien.

Program menyediakan pilihan menu interaktif untuk melakukan operasi seperti menambahkan data mahasiswa baru, menghapus data berdasarkan NIM, mencari data berdasarkan NIM, dan mencari data berdasarkan rentang nilai (80-90).

Untuk memasukkan data mahasiswa baru, pengguna diminta untuk memasukkan NIM (dengan panjang 10 digit), nama, dan nilai. Ini memastikan validitas data yang dimasukkan.

Operasi pencarian dan penghapusan dilakukan berdasarkan NIM, yang memberikan efisiensi tinggi dalam pencarian dan penghapusan data.

Program mencetak hasil operasi dengan jelas, memberikan informasi tentang keberhasilan operasi atau hasil pencarian.

Selain itu, program memiliki fitur untuk mencari data mahasiswa berdasarkan rentang nilai, yaitu nilai antara 80 hingga 90.

Seluruh program dirancang untuk memberikan pengalaman pengguna yang mudah dan intuitif dalam mengelola data mahasiswa menggunakan hash table.

BAB IV

KESIMPULAN

Hash table adalah salah satu struktur data yang penting dan sering digunakan dalam pemrograman komputer. Berikut adalah beberapa kesimpulan keseluruhan dari materi hash table:

1. **Efisiensi Pencarian**: Hash table menyediakan pencarian data yang sangat cepat dengan kompleksitas waktu rata-rata O(1) untuk operasi pencarian, penghapusan, dan penyisipan dalam kasus terbaik. Ini membuatnya menjadi pilihan yang baik untuk aplikasi yang membutuhkan kinerja tinggi dalam pencarian data.

2. **Fungsi Hash**: Fungsi hash adalah inti dari hash table. Ini memetakan kunci (seperti NIM dalam kasus mahasiswa) ke indeks di dalam tabel hash. Fungsi hash yang baik akan mendistribusikan data secara merata di seluruh tabel hash, mengurangi kemungkinan tabrakan hash.

3. **Tabrakan Hash**: Ketika dua kunci memetakan ke indeks yang sama dalam tabel hash, terjadi tabrakan hash. Ini dapat diatasi dengan menggunakan teknik seperti chaining (menggunakan linked list untuk menyimpan elemen yang berbagi indeks) atau probing (mencari slot kosong yang tersedia di sekitar indeks yang bertabrakan).

4. **Penanganan Tabrakan**: Cara penanganan tabrakan dapat memengaruhi kinerja hash table. Metode chaining dapat mengatasi tabrakan dengan efisien, namun membutuhkan alokasi memori tambahan. Metode probing dapat lebih efisien dalam penggunaan memori, tetapi dapat menghasilkan masalah clustering yang mempengaruhi kinerja pencarian.

5. **Aplikasi**: Hash table banyak digunakan dalam berbagai aplikasi, seperti basis data, bahasa pemrograman, kriptografi, dan banyak lagi. Mereka memungkinkan pengindeksan cepat dan pencarian data dalam jumlah besar dengan efisiensi tinggi.

6. **Perlunya Fungsi Hash yang Baik**: Kualitas fungsi hash sangat penting dalam kinerja hash table. Fungsi hash yang buruk dapat menyebabkan banyak tabrakan, mengurangi efisiensi hash table secara keseluruhan. Oleh karena itu, pemilihan atau desain fungsi hash yang baik adalah langkah penting dalam penggunaan hash table.

Dengan memahami konsep dan prinsip dasar hash table serta penerapan yang tepat dari fungsi hash dan teknik penanganan tabrakan, kita dapat memanfaatkan keuntungan hash table untuk mengoptimalkan kinerja aplikasi kita.