# LAPORAN PRAKTIKUM

# MODUL 5
# HASH TABLE

**Disusun oleh:**
**Denny Budiansyach**
**NIM: 2311102022**

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

**PROGRAM STUDI TEKNIK INFORMATIKA**
**FAKULTAS INFORMATIKA**
**INSTITUT TEKNOLOGI TELKOM PURWOKERTO**
**PURWOKERTO**
**2024**

# BAB I
# TUJUAN PRAKTIKUM

1. Praktikan dapat mengetahui dan memahami definisi dan konsep dari Hash Code Praktikan dapat membuat linked list circular dan non circular.
2. Praktikan dapat mengaplikasikan atau menerapkan Hash Code pada program yang dibuat

# BAB II

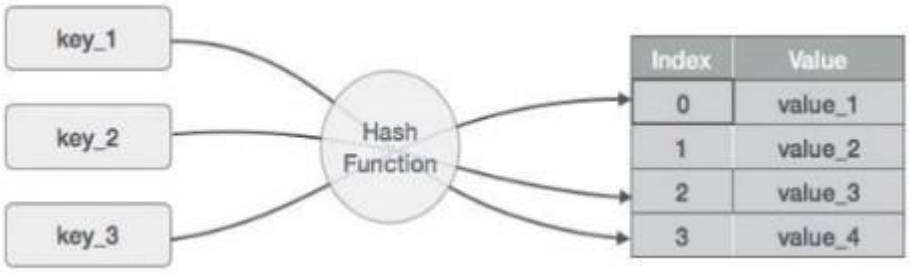
# DASAR TEORI

## 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:

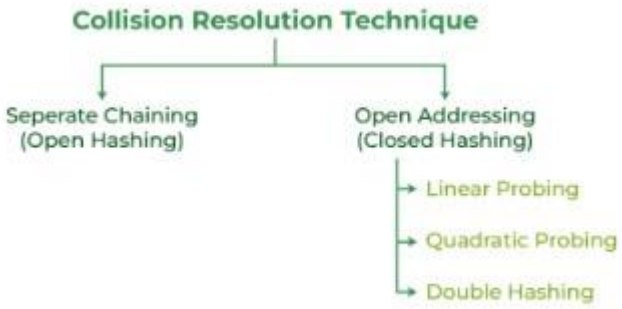
Memperbarui data dalam hash table dengan mencari data menggunakan fungsi hash, dan kemudian memperbarui data yang ditemukan.

**5.** Traversal:

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

**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 :




**1.** 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.

**2.** Closed Hashing

- Linear Probing

  Pada saat terjadi collision, maka akan mencari posisi yang kosong di bawah 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**

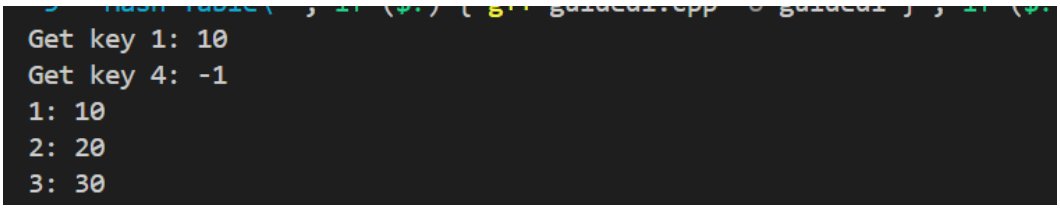
```
#include <iostream>
using namespace std;
const int MAX_SIZE = 10;
// Fungsi hash sederhana
int hash_func(int key)
{
    return key % MAX_SIZE;
}
// Struktur data untuk setiap node
struct Node
{
    int key;
    int value;
    Node *next;
    Node(int key, int value) : key(key), value(value),next(nullptr)
{}
};
// 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 di atas adalah bentuk implementasi dari Hash Table yang menerapkan beberapa operasi dasar seperti penambahan, penghapusan, pengubahan, pencarian, dan pengaksesan/pencetakan data. Setiap node pada program memiliki value dan key nya masing masing. Setiap key node akan melalui proses hashing dengan teknik divison atau modulo(%), dimana hasil key hashing akan menentukan indeks dari data atau node tersebut dalam sebuah hash table. Program tersebut menggunakan beberapa fungsi untuk menjalankan operasi dasar seperti fungsi insert untuk input pasangan key dan value ke dalam hash table.

## GUIDED 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].end();
             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

```
 5 - Hash Table\" ; if ($?) { g++ guided2.cpp -o guided2 } ; if ($?) { .\guided2 }
Nomer Hp Mistah : 1234
Phone Hp Pastah : 5678
Nomer Hp Mistah setelah dihapus :

Hash Table :
0:
1:
2:
3:
4: [Pastah, 5678]
5:
6: [Ghana, 91011]
7:
8:
9:
10:
```

**Deskripsi program**

Program di atas adalah implementasi dari Hash Map yang menggunakan struktur data hash table untuk menyimpan dan mengakses pasangan kunci-nomor telepon. Setiap pasangan kunci dan nomor telepon direpresentasikan oleh sebuah node yang memiliki dua anggota, yaitu nama dan nomor telepon. Program ini menyediakan operasi dasar seperti penambahan, penghapusan, dan pencarian data berdasarkan nama.

Proses hashing dalam program ini menggunakan teknik penjumlahan kode ASCII dari setiap karakter dalam nama sebagai kunci, kemudian hasilnya dimodulo dengan ukuran hash table untuk mendapatkan indeks di mana data akan disimpan. Hasil dari proses hashing ini menentukan lokasi atau indeks dari data dalam hash table.

Operasi penghapusan dilakukan dengan menghapus elemen dari vektor hash table menggunakan iterator. Setiap operasi yang dilakukan diimplementasikan dalam class `HashMap`, yang juga memiliki metode untuk mencetak isi hash table untuk keperluan pemantauan dan debugging. Dengan pendekatan ini, program kedua menggambarkan penggunaan struktur data hash map untuk memetakan nama ke nomor telepon dengan efisien, serta menunjukkan cara mengimplementasikan operasi dasar seperti penambahan, penghapusan, dan pencarian data dalam hash table.

## LATIHAN KELAS - UNGUIDED

1. **Unguided Source code**

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

const int TABLE_SIZE = 11;

class HashNode
{
public:
    string nim;
    int nilai;

    HashNode(string nim, int nilai)
    {
        this->nim = nim;
        this->nilai = nilai;
    }
};

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

    int hashFunc(string key)
    {
        int index = 0;
        for (char c : key)
        {
            index += c;
        }
        return index % TABLE_SIZE;
    }

public:
    void insert(string nim, int nilai)
    {
```

```
        int index = hashFunc(nim);
        for (auto node : table[index])
        {
            if (node->nim == nim)
            {
                node->nilai = nilai;
                return;
            }
        }
        table[index].push_back(new HashNode(nim, nilai));
    }

    void remove(string nim)
    {
        int index = hashFunc(nim);
        for (auto it = table[index].begin(); it !=
table[index].end(); it++)
        {
            if ((*it)->nim == nim)
            {
                table[index].erase(it);
                return;
            }
        }
    }

    int searchByNIM(string nim)
    {
        int index = hashFunc(nim);
        for (auto node : table[index])
        {
            if (node->nim == nim)
            {
                return node->nilai;
            }
        }
        return -1;
    }

    vector<string> searchByRange(int min, int max)
    {
        vector<string> result;
```

```
        for (int i = 0; i < TABLE_SIZE; ++i)
        {
            for (auto node : table[i])
            {
                if (node->nilai >= min && node->nilai <= max)
                {
                    result.push_back(node->nim);
                }
            }
        }
        return result;
    }

    void print()
    {
        for (int i = 0; i < TABLE_SIZE; i++)
        {
            cout << i << ": ";
            for (auto pair : table[i])
            {
                if (pair != nullptr)
                {
                    cout << "[" << pair->nim << ", " << pair->nilai
<< "]";
                }
            }
            cout << endl;
        }
    }
};

int main()
{
    HashMap student_map;
    int choice;
    string nim;
    int nilai;

    do
    {
        cout << "\nMenu:\n"
             << "1. Tambah Data Mahasiswa\n"
```

```
             << "2. Hapus Data Mahasiswa\n"
             << "3. Cari Data Mahasiswa berdasarkan NIM\n"
             << "4. Cari Data Mahasiswa berdasarkan Rentang Nilai
(80-90)\n"
             << "5. Keluar\n"
             << "Pilihan: ";
        cin >> choice;

        switch (choice)
        {
        case 1:
            cout << "Masukkan NIM Mahasiswa: ";
            cin >> nim;
            cout << "Masukkan Nilai Mahasiswa: ";
            cin >> nilai;
            student_map.insert(nim, nilai);
            cout << "Data berhasil ditambahkan.\n";
            break;
        case 2:
            cout << "Masukkan NIM Mahasiswa yang akan dihapus: ";
            cin >> nim;
            student_map.remove(nim);
            cout << "Data berhasil dihapus.\n";
            break;
        case 3:
            cout << "Masukkan NIM Mahasiswa yang ingin dicari: ";
            cin >> nim;
            nilai = student_map.searchByNIM(nim);
            if (nilai != -1)
                cout << "Nilai Mahasiswa dengan NIM " << nim << "
adalah: " << nilai << endl;
            else
                cout << "Mahasiswa dengan NIM " << nim << " tidak
ditemukan.\n";
            break;
        case 4:
            cout << "Mahasiswa dengan Nilai antara 80-90:\n";
            {
                vector<string> result =
student_map.searchByRange(80, 90);
                if (result.empty())
```

```
                    cout << "Tidak ada mahasiswa dengan nilai antara
80-90.\n";
                else
                {
                    for (const auto &nim : result)
                    {
                        cout << "NIM: " << nim << ", Nilai: " <<
student_map.searchByNIM(nim) << endl;
                    }
                }
            }
            break;
        case 5:
            cout << "Keluar dari program.\n";
            break;
        default:
            cout << "Pilihan tidak valid. Silakan coba lagi.\n";
            break;
        }

    } while (choice != 5);

    return 0;
}
```

## Screenshoot program

### a. Tambah Data

```
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
Pilihan: 1
Masukkan NIM Mahasiswa: 2311102022
Masukkan Nilai Mahasiswa: 78
Data 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
Pilihan: 1
Masukkan NIM Mahasiswa: 2311102000
Masukkan Nilai Mahasiswa: 91
Data 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
Pilihan: 1
Masukkan NIM Mahasiswa: 2311102099
Masukkan Nilai Mahasiswa: 89
Data berhasil ditambahkan.
```

### b. Cari Data Berdasarkan NIM

```
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
Pilihan: 3
Masukkan NIM Mahasiswa yang ingin dicari: 2311102099
Nilai Mahasiswa dengan NIM 2311102099 adalah: 89
```

### c. Cari Data Berdasarkan Nilai Rentang 80-90

```
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
Pilihan: 4
Mahasiswa dengan Nilai antara 80-90:
NIM: 2311102099, Nilai: 89
```

### d. Hapus Data

```
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
Pilihan: 2
Masukkan NIM Mahasiswa yang akan dihapus: 2311102099
Data berhasil dihapus.
```

**Deskripsi program**

Program di atas adalah implementasi dari Hash Map yang digunakan untuk mengelola data mahasiswa berdasarkan Nomor Induk Mahasiswa (NIM) dan nilai. Setiap data mahasiswa direpresentasikan oleh sebuah node yang memiliki anggota NIM dan nilai. Program ini menyediakan beberapa operasi dasar seperti penambahan, penghapusan, dan pencarian data berdasarkan NIM, serta pencarian data berdasarkan rentang nilai tertentu.

Proses hashing dalam program ini menggunakan teknik penjumlahan kode ASCII dari setiap karakter dalam NIM sebagai kunci, kemudian hasilnya dimodulo dengan ukuran hash table untuk mendapatkan indeks di mana data akan disimpan. Hasil dari proses hashing ini menentukan lokasi atau indeks dari data dalam hash table.

Operasi penghapusan dilakukan dengan menghapus elemen dari vektor hash table menggunakan iterator. Program juga menyediakan fungsi pencarian data berdasarkan rentang nilai tertentu, yang mengembalikan daftar NIM mahasiswa yang memiliki nilai di antara rentang yang ditentukan.

Interaksi dengan pengguna dilakukan melalui menu yang ditampilkan di layar, di mana pengguna dapat memilih operasi yang ingin dilakukan, seperti menambahkan data mahasiswa, menghapus data, mencari data berdasarkan NIM, mencari data berdasarkan rentang nilai, atau keluar dari program. Setiap operasi diimplementasikan dalam class `HashMap`, yang juga memiliki metode untuk mencetak isi hash table untuk keperluan pemantauan dan debugging.

# BAB IV
# KESIMPULAN

Hash Table adalah struktur data yang digunakan untuk menyimpan dan mengakses data dengan melalui pasangan kunci-nilai. Wadah dalam menampung data dapat berupa array atau vektor yang diisi kedalam slot-slot yang disebut bucket, di mana setiap bucket dapat menampung satu atau beberapa item data. Fungsi hash digunakan untuk memetakan kunci data ke indeks array. Operasi dasar pada Hash Table meliputi penambahan, penghapusan, pencarian, dan pembaruan data. Teknik penyelesaian collision, adalah kondisi di mana dua atau lebih data memiliki nilai hash yang sama. Terdapat 2 teknik dalam menyelesaikan collision, yaitu chaining (open hashing) dan closed hashing (linear probing, quadratic probing, dan double hashing), yang dapat membantu program menagani konflik hashing secara efektif.

Fungsi hash table penting dalam struktur data karena menyediakan cara yang efisien untuk mengakses dan mengelola data. Dengan memanfaatkan fungsi hash yang efektif dan teknik penyelesaian collision yang sesuai, hash table dapat digunakan dalam berbagai aplikasi, seperti basis data, penyimpanan cache, dan pembuatan indeks, untuk meningkatkan kinerja dan efisiensi dalam pengolahan data. Dengan demikian, pemahaman tentang konsep dasar hash table dan operasi yang terkait dengannya penting untuk pengembangan dan pemeliharaan sistem yang mengandalkan manajemen data yang efisien.

# DAFTAR PUSTAKA

Asisten Praktikum. (2024). MODUL 5 – HASH TABLE, Learning Managament System.

Yusuf, A. D. (2021). Collision Resolution Techniques in Hash Table: A. Paper_84-Collision_Resolution_Techniques_in_Hash_Table-libre, 6.