

# **LAPORAN PRAKTIKUM**

## **MODUL V HASH TABLE**



**Disusun oleh:**  
**Rizal Dwi Anggoro**  
**NIM: 2311102034**

**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**

### **A. Tujuan Praktikum**

Adapun tujuan dari laporan praktikum ini sebagai berikut :

1. Mahasiswa mampu menjelaskan definisi dan konsep dari Hash Code.
2. Mahasiswa mampu menerapkan Hash Code kedalam pemrograman

## BAB II

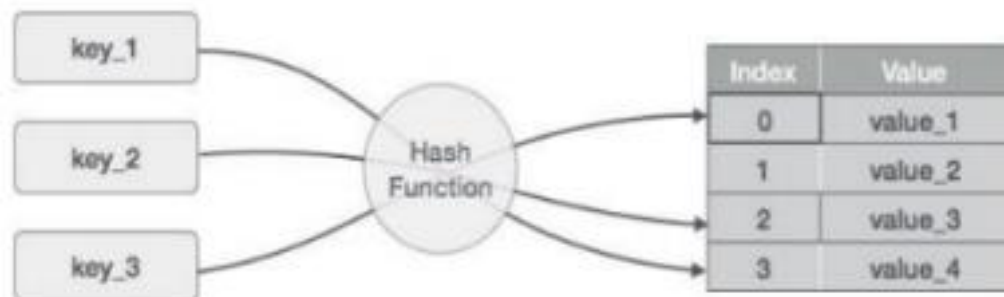
### DASAR TEORI

#### 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 memetakannya 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:

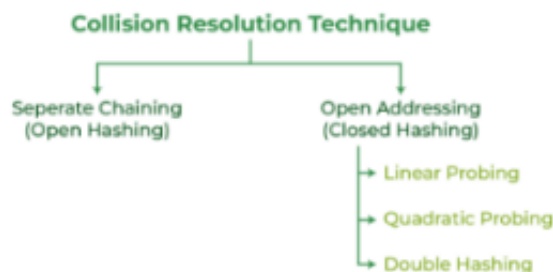
Memperbarui data dalam hash table dengan mencari data menggunakan fungsi hash, dan kemudian memperbarui data yang ditemukan.

### 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 ( 1, 4, 9, 16, 25, ... )

- **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);

```

```

    ht.insert(4, 40);

    // 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

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\File D\ITTP\Tugas ITTP\Semester 2\Praktikum Struktur Data dan Algoritma\Modul 5> g++ guided1.cpp -o guided1
Get key 1: 10
Get key 4: 40
1: 10
2: 20
3: 30
PS D:\File D\ITTP\Tugas ITTP\Semester 2\Praktikum Struktur Data dan Algoritma\Modul 5>

```

### Deskripsi program

Program ini menggunakan array dinamis “table” untuk menentukan indeks penyimpanan untuk setiap kunci yang dimasukkan. Struktur data Node digunakan untuk menyimpan kunci dan nilai yang terkait. Setiap entri dalam tabel hash adalah pointer ke linked list Node.

Implementasi ini mencakup operasi dasar seperti insertion (penyisipan), searching (pencarian), removal (penghapusan), dan traversal

(penelusuran). Semua operasi ini mengikuti prinsip dasar struktur data hash table.

Dalam fungsi main(), beberapa kunci dan nilai dimasukkan ke dalam hash table menggunakan metode insert(). Kemudian, beberapa kunci dicari menggunakan metode get(). Salah satu entri dihapus menggunakan metode remove(). Akhirnya, seluruh isi hash table ditampilkan menggunakan metode traverse()

## 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 << "];"
            }
        }
    }
}

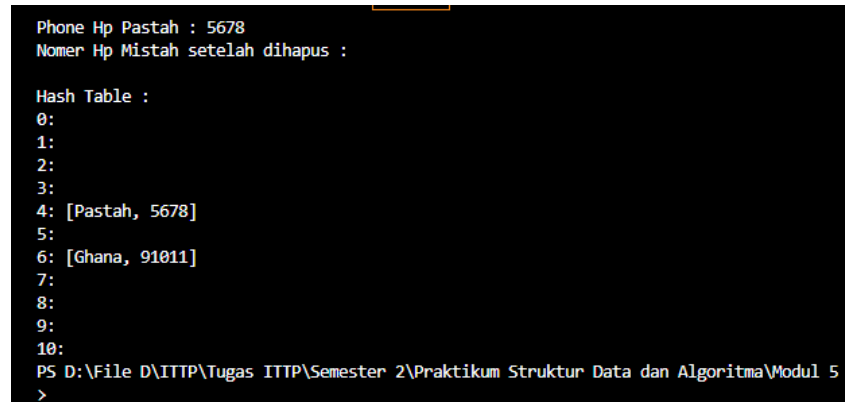
```

```

    }
    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



```

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:
PS D:\File D\ITTP\Tugas ITTP\Semester 2\Praktikum Struktur Data dan Algoritma\Modul 5
>

```

## Deskripsi Program

Kode di atas digunakan untuk menyimpan dan mengelola data menggunakan hash table. Pada program tersebut terdapat struktur data HashNode yang berfungsi untuk menyimpan data berupa nama dan no telp. Class HashMap yang memiliki beberapa metode seperti hashFunc yang berfungsi untuk menghasilkan nilai dari kunci(name), Insert untuk menambahkan data, remove untuk menghapus data, searchByName untuk mencari no telp berdasarkan nama, print untuk mencetak/menampilkan seluruh data. Pada program tersebut user dapat menginput, menghapus, mencari, dan menampilkan data.

## LATIHAN KELAS - UNGUIDED

### 1. Unguided 1

#### Source code

```
#include <iostream>
#include <string>
using namespace std;
const int MAX_SIZE = 10;

// 2311102034_Rizal Dwi Anggoro

struct Node
{
    string NIM;
    string nama;
    int nilai;
    Node *next;
    Node(string NIM, string nama, int nilai) : NIM(NIM),
nama(nama), 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;
}

int hash(string NIM)
{
    int sum = 0;
    for (char c : NIM)
    {
        sum += c;
    }
    return sum % MAX_SIZE;
}

void insert(string NIM, string nama, int nilai)
{
    int index = hash(NIM);
    Node *newNode = new Node(NIM, nama, nilai);
    if (table[index] == nullptr)
    {
        table[index] = newNode;
    }
    else
    {
        Node *current = table[index];

```

```

        while (current->next != nullptr)
        {
            current = current->next;
        }
        current->next = newNode;
    }
}

void remove(string NIM)
{
    int index = hash(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;
        }
    }
    cout << "NIM " << NIM << " Gak Ada\n";
}

```



```

void searchNIM(string NIM)
{
    int index = hash(NIM);
    Node *current = table[index];
    while (current != nullptr)
    {
        if (current->NIM == NIM)
        {
            cout << "NIM : " << current->NIM << endl;
            cout << "Nama: " << current->nama << endl;
            cout << "Nilai: " << current->nilai << endl;
            return;
        }
        current = current->next;
    }
    cout << "NIM : " << NIM << " TIDAK ADA" << endl;
}
// Searching by range of values (80-90)
void searchRange()
{
    for (int i = 0; i < MAX_SIZE; i++)
    {
        Node *current = table[i];
        while (current != nullptr)
        {
            if (current->nilai >= 80 && current->nilai <=
90)
            {
                cout << "NIM: " << current->NIM << ", NAMA:
" << current->nama << ", Nilai: " << current->nilai << endl;
            }
            current = current->next;
        }
    }
}

```

```

    }

    void display()
    {
        cout << "DATA MAHASISWA:\n";
        for (int i = 0; i < MAX_SIZE; i++)
        {
            Node *current = table[i];
            while (current != nullptr)
            {
                cout << "NIM: " << current->NIM << ", NAMA: " <<
current->nama << ", Nilai: " << current->nilai << endl;
                current = current->next;
            }
        }
    };

    int main()
    {
        HashTable hashTable;
        int menu;
        string NIM, nama;
        int nilai;
        do
        {
            cout << "====Menu====\n";
            cout << "1. TAMBAH DATA\n";
            cout << "2. HAPUS DATA\n";
            cout << "3. CARI NIM\n";
            cout << "4. CARI RENTANG NILAI (80-90)\n";
            cout << "5. MENAMPILKAN SEMUA DATA\n";
            cout << "6. EXIT\n";
            cout << "PILIH : ";
            cin >> menu;

```

```
cout << "=====\n";
switch (menu)
{
case 1:
    cout << "Input NIM: ";
    cin >> NIM;
    cout << "Input Nama: ";
    cin >> nama;
    cout << "Input Nilai: ";
    cin >> nilai;
    hashTable.insert(NIM, nama, nilai);
    break;
case 2:
    cout << "Input NIM : ";
    cin >> NIM;
    hashTable.remove(NIM);
    break;
case 3:
    cout << "Input NIM : ";
    cin >> NIM;
    hashTable.searchNIM(NIM);
    break;
case 4:
    cout << "Data Nilai 80-90: ";
    hashTable.searchRange();
    break;
case 5:
    hashTable.display();
    break;
case 6:
    cout << "TY.\n";
    break;
default:
    cout << "Menu Tidak Ada.\n";
```

```
    }  
    } while (menu != 6);  
    return 0;  
}
```

## Screenshoot program

### 1. Tambah Data

```
=====Menu=====  
1. TAMBAH DATA  
2. HAPUS DATAA  
3. CARI NIM  
4. CARI RENTANG NILAI (80-90)  
5. MENAMPILKAN SEMUA DATA  
6. EXIT  
PILIH : 1  
=====
```

```
Input NIM: 2311102034  
Input Nama: Rizal  
Input Nilai: 89
```

```
DATA MAHASISWA:  
NIM: 2311102001, NAMA: Kasrun, Nilai: 96  
NIM: 2311102040, NAMA: Raihan, Nilai: 87  
NIM: 2311102024, NAMA: Rafael, Nilai: 90  
NIM: 2311102034, NAMA: Rizal, Nilai: 89
```

### 2. Hapus Data

```
=====Menu=====  
1. TAMBAH DATA  
2. HAPUS DATAA  
3. CARI NIM  
4. CARI RENTANG NILAI (80-90)  
5. MENAMPILKAN SEMUA DATA  
6. EXIT  
PILIH : 2  
=====
```

```
Input NIM : 2311102040
```

```
DATA MAHASISWA:
NIM: 2311102001, NAMA: Kasrun, Nilai: 96
NIM: 2311102024, NAMA: Rafael, Nilai: 90
NIM: 2311102034, NAMA: Rizal, Nilai: 89
```

### 3. Cari NIM

```
=====Menu=====
1. TAMBAH DATA
2. HAPUS DATAA
3. CARI NIM
4. CARI RENTANG NILAI (80-90)
5. MENAMPILKAN SEMUA DATA
6. EXIT
PILIH : 3

=====
Input NIM : 2311102034
NIM : 2311102034
Nama: Rizal
Nilai: 89
```

### 4. Cari rentang Nilai (80-90)

```
=====Menu=====
1. TAMBAH DATA
2. HAPUS DATAA
3. CARI NIM
4. CARI RENTANG NILAI (80-90)
5. MENAMPILKAN SEMUA DATA
6. EXIT
PILIH : 4

=====
Data Nilai 80-90: NIM: 2311102024, NAMA: Rafael, Nilai: 90
NIM: 2311102034, NAMA: Rizal, Nilai: 89
```

### 5. Menampilkan Semua Data

```
=====Menu=====
1. TAMBAH DATA
2. HAPUS DATAA
3. CARI NIM
4. CARI RENTANG NILAI (80-90)
5. MENAMPILKAN SEMUA DATA
6. EXIT
PILIH : 5

=====
DATA MAHASISWA:
NIM: 2311102001, NAMA: Kasrun, Nilai: 96
NIM: 2311102024, NAMA: Rafael, Nilai: 90
NIM: 2311102034, NAMA: Rizal, Nilai: 89
```

## **Deskripsi program**

Program ini adalah implementasi sederhana dari struktur data tabel hash dalam C++. Program ini menggunakan hash table untuk menyimpan data mahasiswa berdasarkan NIM inputan user. Setiap item hash table berisi terkait dari struktur data node, di mana setiap node menyimpan NIM, nama, dan nilai mahasiswa.

Struktur Data Node, Berisi informasi tentang seorang mahasiswa, seperti NIM, nama, dan nilai. Node ini juga memiliki pointer ke node berikutnya dalam kasus terjadi tabrakan hash. Class Hash Table, Mewakili struktur data tabel hash. Dengan ini dapat untuk menambahkan, menghapus, mencari, menampilkan, dan mencari rentang nilai mahasiswa.

Fungsi hash(), Digunakan untuk menghitung fungsi hash dari NIM mahasiswa untuk menentukan indeks di mana mereka akan disimpan dalam hash table. Fungsi main(), program memiliki fungsi main() untuk melakukan operasi-operasi dasar pada tabel hash, seperti menambah, menghapus, mencari, menampilkan, dan keluar dari program.

## **BAB IV**

### **KESIMPULAN**

Praktikum Hash Table memberikan pemahaman saya tentang cara menyimpan data dengan cepat menggunakan fungsi hash, yang sangat berguna untuk pencarian dan manipulasi data dalam aplikasi komputer seperti database, kamus, dan lainnya.

## **DAFTAR PUSTAKA**

Assisten Praktikum, "Modul V Hash Table", Group WhatsApp, 2024.