

**LAPORAN PRAKTIKUM  
STRUKTUR DATA DAN  
ALGORITMA**

**MODUL V  
HASH TABLE**



**Disusun Oleh :**

Raka Andriy Shevchenko  
2311102054

**Dosen**

Wahyu Andi Saputra, S.Pd., M.Eng

**PROGRAM STUDI S1 TEKNIK INFORMATIKA  
FAKULTAS INFORMATIKA  
INSTITUT TEKNOLOGI TELKOM PURWOKERTO**

## A. Dasar Teori

Hash table adalah struktur data yang terdiri atas sebuah tabel dan fungsi yang bertujuan untuk memetakan nilai kunci yang unik untuk setiap record. Fungsi hash, yang disebut sebagai hash function, digunakan untuk mengubah nilai kunci menjadi nilai hash yang unik dan dapat digunakan sebagai alamat penyimpanan data dalam tabel. Dengan demikian, hash table dapat mempercepat pencarian data yang berjumlah banyak dengan menggunakan memori penyimpanan utama berbentuk array dan tambahan algoritma untuk mempercepat pemrosesan data.

Keuntungan dari hash table adalah kecepatan dalam insertions, deletions, dan searching yang relatif sama, serta cocok untuk merepresentasikan data dengan frekuensi insert, delete dan search yang tinggi. Namun, hash table juga memiliki kekurangan, seperti kemungkinan terjadi collision (tabrakan), yang dapat diatasi dengan menggunakan Closed Hashing (open addressing) dan Open Hashing (separate chaining).

Struktur dasar hash table terdiri atas beberapa komponen utama, seperti:

1. Tabel Hash: Array yang digunakan untuk menyimpan data dengan indeks yang dihasilkan oleh fungsi hash.
2. Fungsi Hash: Fungsi yang digunakan untuk mengubah nilai kunci menjadi nilai hash yang unik dan dapat digunakan sebagai alamat penyimpanan data dalam tabel.
3. Collision Resolution: Algoritma yang digunakan untuk mengatasi collision, seperti Closed Hashing (open addressing) dan Open Hashing (separate chaining).

Dalam implementasi hash table, terdapat beberapa metode yang dapat digunakan untuk mengatasi collision, seperti:

1. Linear Probing: Mencari alamat lain apabila alamat yang dituju sudah terisi oleh data dengan bergeser 1 indeks dari alamat sebelumnya hingga ditemukan alamat yang belum terisi data.
2. Quadratic Probing: Mencari alamat baru untuk ditempati dengan proses perhitungan kuadrat yang lebih kompleks.

3. Double Hashing: Mencari alamat baru untuk menyimpan data yang belum dapat masuk ke dalam table dengan menggunakan hash function lagi.

Dalam penggunaan hash table, perlu diperhatikan beberapa hal, seperti:

1. Pemilihan Ukuran Tabel: Ukuran tabel yang lebih besar dari data yang diharapkan dapat meningkatkan kecepatan pencarian.
2. Pemilihan Fungsi Hash: Fungsi hash yang baik dapat meningkatkan kecepatan pencarian dan mengurangi kemungkinan terjadi collision.
3. Penanganan Collision: Penanganan collision yang efektif dapat meningkatkan kecepatan pencarian dan mengurangi kemungkinan terjadi error.

Dalam sintesis, hash table adalah struktur data yang efektif untuk mempercepat pencarian data yang berjumlah banyak dengan menggunakan memori penyimpanan utama berbentuk array dan tambahan algoritma untuk mempercepat pemrosesan data. Namun, perlu diperhatikan beberapa hal, seperti pemilihan ukuran tabel, pemilihan fungsi hash, dan penanganan collision, untuk meningkatkan kecepatan dan efisiensi penggunaan hash table.

## B. Guided

### a. 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;

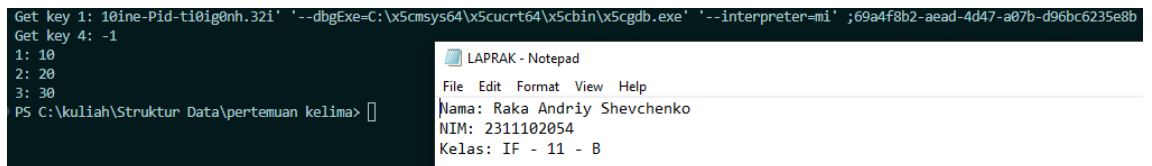
// Deletion
ht.remove(4);

// Traversal
ht.traverse();

return 0;
}

```

Output:



```

Get key 1: 10
Get key 4: -1
1: 10
2: 20
3: 30
PS C:\kuliah\Struktur Data\pertemuan kelima>

```

LAPRAK - Notepad

File Edit Format View Help

Nama: Raka Andriy Shevchenko

NIM: 2311102054

Kelas: IF - 11 - B

Deskripsi:

Kode di atas menggunakan array dinamis “table” untuk menyimpan bucket dalam hash table. Setiap bucket diwakili oleh sebuah linked list dengan setiap node merepresentasikan satu item data. Fungsi hash sederhana hanya menggunakan modulus untuk memetakan setiap input kunci ke nilai indeks array.

## b. 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 << "]\n";
            }
        }
        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 : " <<

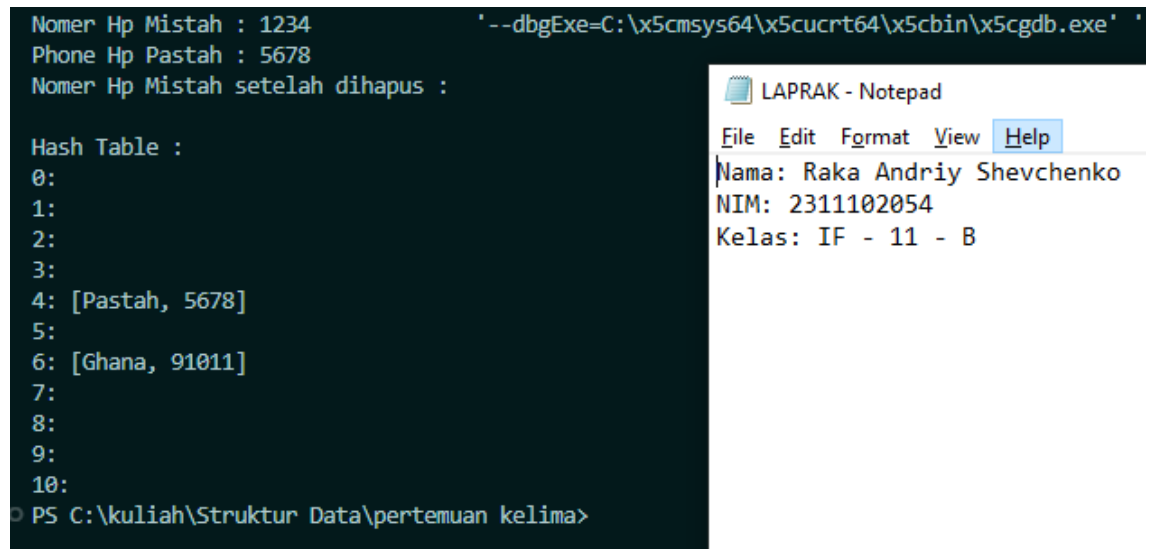
```

```

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;
}

```

Output:



The screenshot shows the output of a C++ program in a dark-themed terminal window. The output displays the removal of 'Mistah' from a hash table and the resulting state of the hash table. To the right, a Notepad window titled 'LAPRAK - Notepad' contains student information.

```

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:
PS C:\kuliah\Struktur Data\pertemuan kelima>

```

LAPRAK - Notepad

File Edit Format View Help

Nama: Raka Andriy Shevchenko  
NIM: 2311102054  
Kelas: IF - 11 - B

Deskripsi:

Pada program di atas, class HashNode merepresentasikan setiap node dalam hash table, yang terdiri dari nama dan nomor telepon karyawan. Class HashMap digunakan untuk mengimplementasikan struktur hash table dengan menggunakan vector yang menampung pointer ke HashNode. Fungsi hashFunc digunakan untuk menghitung nilai hash dari nama karyawan yang diberikan, dan fungsi insert digunakan untuk menambahkan data baru ke dalam hash table. Fungsi remove digunakan untuk menghapus data dari hash table, dan fungsi searchByName digunakan untuk mencari nomor telepon dari karyawan dengan nama yang diberikan.

### C. Unguided

#### a. Unguided 1

Source code:

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;
const int TABLE_SIZE = 11;
string nim;
int nilai;
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];

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

    void insert(string nim, int nilai)
    {
        int hash_val = hashFunc(nim);
        for (auto node : table[hash_val])
```

```

        {
            if (node->nim == nim)
            {
                node->nilai = nilai;
                return;
            }
        }
        table[hash_val].push_back(new HashNode(nim, nilai));
    }

    void remove(string nim)
    {
        int hash_val = hashFunc(nim);

        for (auto it = table[hash_val].begin(); it !=
table[hash_val].end(); it++)
        {
            if ((*it)->nim == nim)
            {
                table[hash_val].erase(it);
                return;
            }
            else
            {
                cout << "Tidak Ditemukan" << endl;
            }
        }
    }

    int searchByNim(string nim)
    {
        int hash_val = hashFunc(nim);
        for (auto node : table[hash_val])
        {
            if (node->nim == nim)
            {
                cout << "\nMahasiswa dengan NIM " << node->nim
<< " memiliki nilai " << node->nilai << endl;
            }
        }
        return 0;
    }
}

```

```

int searchByNilai(int minn, int maxx)
{
    bool found = false;
    for (const auto &bucket : table)
    {
        for (auto node : bucket)
        {
            if (node->nilai >= minn && node->nilai <= maxx)
            {
                cout << "[ NIM : " << node->nim << ", NILAI
: " << node->nilai << " ]" << endl;
                found = true;
            }
        }
    }
    if (!found) {
        cout << "Tidak ada mahasiswa pada rentang nilai
tersebut.";
    }
    return 0;
}

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

};

int main()
{
    HashMap mahasiswa_map;

```

```

int choice;
do
{
    cout << "\t\nData Mahasiswa\t" << endl;
    cout << "1. Tambah Data\n";
    cout << "2. Hapus Data\n";
    cout << "3. Cari mahasiswa dengan NIM\n";
    cout << "4. Cari mahasiswa dengan Nilai\n";
    cout << "0. Keluar" << endl;
    cout << "Pilihan Anda: ";
    cin >> choice;
    switch (choice)
    {
        case 1:
            cout << "Masukkan NIM Mahasiswa : ";
            cin >> nim;
            cout << "Masukkan Nilai Mahasiswa : ";
            cin >> nilai;

            mahasiswa_map.insert(nim, nilai);

            cout << "\t\nData Mahasiswa\t" << endl;
            mahasiswa_map.print();
            cout << endl;
            break;
        case 2:
            cout << "Masukkan NIM Mahasiswa : ";
            cin >> nim;
            mahasiswa_map.remove(nim);

            cout << "\t\nData Mahasiswa\t" << endl;
            mahasiswa_map.print();
            cout << endl;
            break;
        case 3:
            cout << "Masukkan NIM Mahasiswa : ";
            cin >> nim;
            mahasiswa_map.searchByNim(nim);

            break;
        case 4:
            int maxx, minn;
            cout << "Masukkan Nilai Tertinggi : ";

```

```

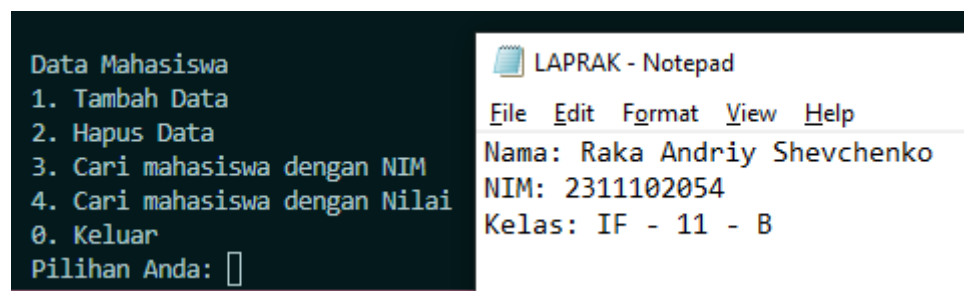
        cin >> maxx;
        cout << "Masukkan Nilai Terendah : ";
        cin >> minn;
        mahasiswa_map.searchByNilai(minn, maxx);
        break;
    default:
        cout << "Harap masukkan pilihan sesuai menu.\n";
        break;
    }

} while (choice != 0);
return 0;
}

```

Output:

a. Menu



The screenshot shows a terminal window on the left with the following text:

```

Data Mahasiswa
1. Tambah Data
2. Hapus Data
3. Cari mahasiswa dengan NIM
4. Cari mahasiswa dengan Nilai
0. Keluar
Pilihan Anda: 

```

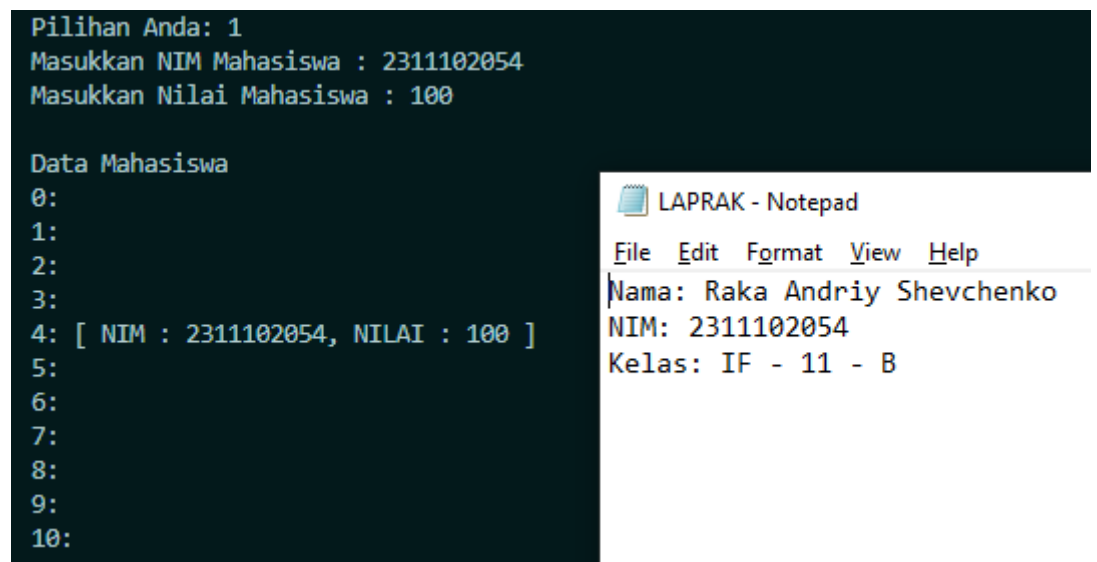
On the right, there is a Notepad window titled "LAPRAK - Notepad" with a menu bar (File, Edit, Format, View, Help) and the following text:

```

Nama: Raka Andriy Shevchenko
NIM: 2311102054
Kelas: IF - 11 - B

```

b. Tambah Data



The screenshot shows a terminal window on the left with the following text:

```

Pilihan Anda: 1
Masukkan NIM Mahasiswa : 2311102054
Masukkan Nilai Mahasiswa : 100

Data Mahasiswa
0:
1:
2:
3:
4: [ NIM : 2311102054, NILAI : 100 ]
5:
6:
7:
8:
9:
10:

```

On the right, there is a Notepad window titled "LAPRAK - Notepad" with a menu bar (File, Edit, Format, View, Help) and the following text:

```

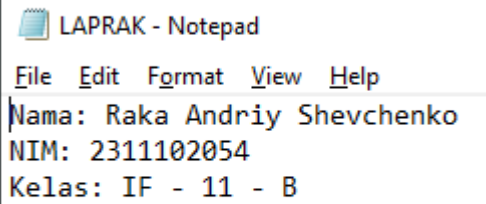
Nama: Raka Andriy Shevchenko
NIM: 2311102054
Kelas: IF - 11 - B

```

c. Hapus Data

```
Pilihan Anda: 2
Masukkan NIM Mahasiswa : 2311102054

Data Mahasiswa
0:
1:
2:
3:
4:
5:
6:
7:
8:
9:
10:
```

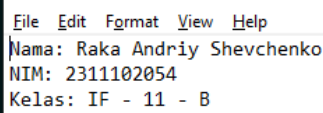


LAPRAK - Notepad  
File Edit Format View Help  
Nama: Raka Andriy Shevchenko  
NIM: 2311102054  
Kelas: IF - 11 - B

d. Cari Mahasiswa Berdasarkan NIM

```
Pilihan Anda: 3
Masukkan NIM Mahasiswa : 2311102054

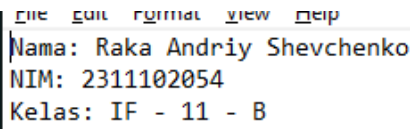
Mahasiswa dengan NIM 2311102054 ditemukan di database, dan memiliki nilai 100
```



LAPRAK - Notepad  
File Edit Format View Help  
Nama: Raka Andriy Shevchenko  
NIM: 2311102054  
Kelas: IF - 11 - B

e. Cari Mahasiswa Berdasarkan Nilai

```
Pilihan Anda: 4
Masukkan Nilai Tertinggi : 90
Masukkan Nilai Terendah : 80
Tidak ada mahasiswa pada rentang nilai tersebut.
```



LAPRAK - Notepad  
File Edit Format View Help  
Nama: Raka Andriy Shevchenko  
NIM: 2311102054  
Kelas: IF - 11 - B

Deskripsi:

Program ini merupakan implementasi sistem manajemen data mahasiswa yang menggunakan struktur data HashMap untuk menyimpan dan mengelola data mahasiswa. Data yang disimpan meliputi Nomor Induk Mahasiswa (NIM) dan nilai mahasiswa.

Program ini adalah sistem manajemen data mahasiswa menggunakan struktur data HashMap. Data mahasiswa yang disimpan mencakup NIM dan nilai. Kelas HashNode merepresentasikan node yang menyimpan pasangan NIM dan nilai. Kelas HashMap memiliki metode untuk menyisipkan, menghapus, dan mencari data mahasiswa. Fungsi hashFunc digunakan untuk menghitung nilai hash dari NIM. Metode insert menambah data mahasiswa



ke dalam tabel hash, dan remove menghapus data berdasarkan NIM. Metode `searchByNim` mencari mahasiswa berdasarkan NIM, sedangkan `searchByNilai` mencari mahasiswa dalam rentang nilai tertentu. Fungsi `print` menampilkan seluruh isi tabel hash. Fungsi `main` menyediakan menu interaktif bagi pengguna untuk mengelola data mahasiswa. Program berjalan dalam loop hingga pengguna memilih untuk keluar.

#### D. Kesimpulan

Hash table adalah salah satu struktur data yang paling penting dan sering digunakan dalam pemrograman computer. Dengan menggunakan hash function untuk menentukan kunci ke indeks dalam tabel hash, hash table memungkinkan penyimpanan data yang efisien dan pencarian data yang cepat. Hal ini terutama disebabkan oleh waktu eksekusi konstan yang dapat dicapai dalam operasi dasar seperti penyisipan, pengambilan, dan penghapusan data. Namun, meskipun hash table menawarkan kinerja yang sangat baik dalam kasus rata-rata, kinerja terburuknya dapat menjadi masalah jika hash function tidak seimbang atau jika collision terjadi selalu sering. Oleh karena itu, perencanaan yang cermat dalam pemilihan ukuran tabel hash dan implementasi hash function yang efisien sangat penting untuk memastikan kinerja yang optimal dari hash tab

#### E. Referensi

- [1] Fajar Baskoro, Hash Table. Diakses dari <http://fajarbaskoro.blogspot.com/2021/06/hash-table.html>
- [2] Ruli Manurung & Ade Azurat, Hash Table. Diakses dari [http://aren.cs.ui.ac.id/sda/resources/sda2010/15\\_hashtable.pdf](http://aren.cs.ui.ac.id/sda/resources/sda2010/15_hashtable.pdf)