

LAPORAN PRAKTIKUM

MODUL V

HASH TABLE



Disusun oleh:

Reli Gita Nurhidayati

NIM: 2311102025

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

Praktikum ini memiliki tujuan, yaitu:

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

BAB II

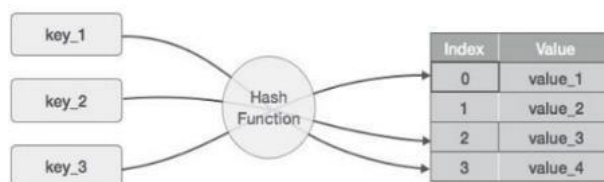
DASAR TEORI

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:

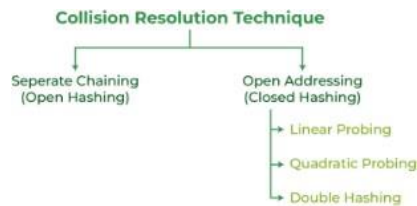
Memperbarui data dalam hash table dengan mencari data menggunakan fungsi hash, dan kemudian memperbarui data yang ditemukan.

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 (12, 22, 32, 42, ...)

- **Double Hashing**

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

BAB III

GUIDED

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)
            {

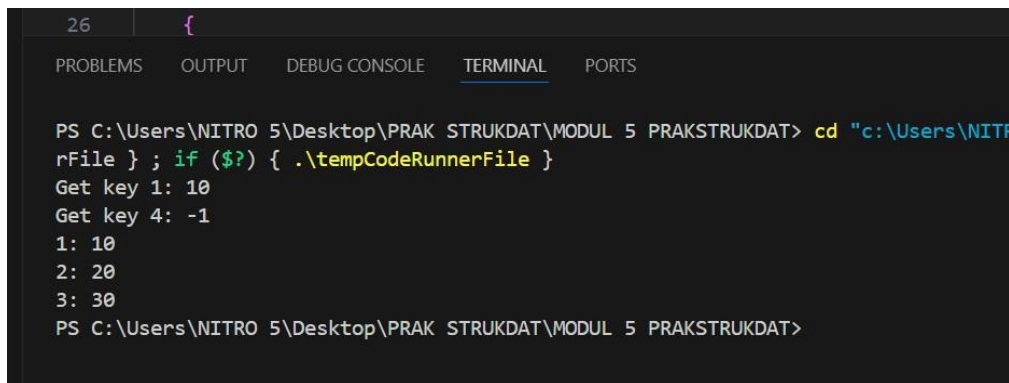
```

```

        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



```
26 {  
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS  
  
PS C:\Users\NITRO 5\Desktop\PRAK STRUKDAT\MODUL 5 PRAKSTRUKDAT> cd "c:\Users\NITRO 5\Desktop\PRAK STRUKDAT" & g++ rFile.cpp & .\tempCodeRunnerFile  
Get key 1: 10  
Get key 4: -1  
1: 10  
2: 20  
3: 30  
PS C:\Users\NITRO 5\Desktop\PRAK STRUKDAT\MODUL 5 PRAKSTRUKDAT>
```

Deskripsi Program

Program ini mendemonstrasikan implementasi sederhana Hash Table dengan separate chaining. Teknik ini mengatasi masalah tabrakan hash (beberapa kunci dipetakan ke indeks yang sama) dengan menyimpan node-node dengan indeks hash yang sama dalam sebuah linked list. Pendekatan ini memungkinkan penyimpanan data yang efisien meskipun terjadi tabrakan hash.

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 "";
}

```

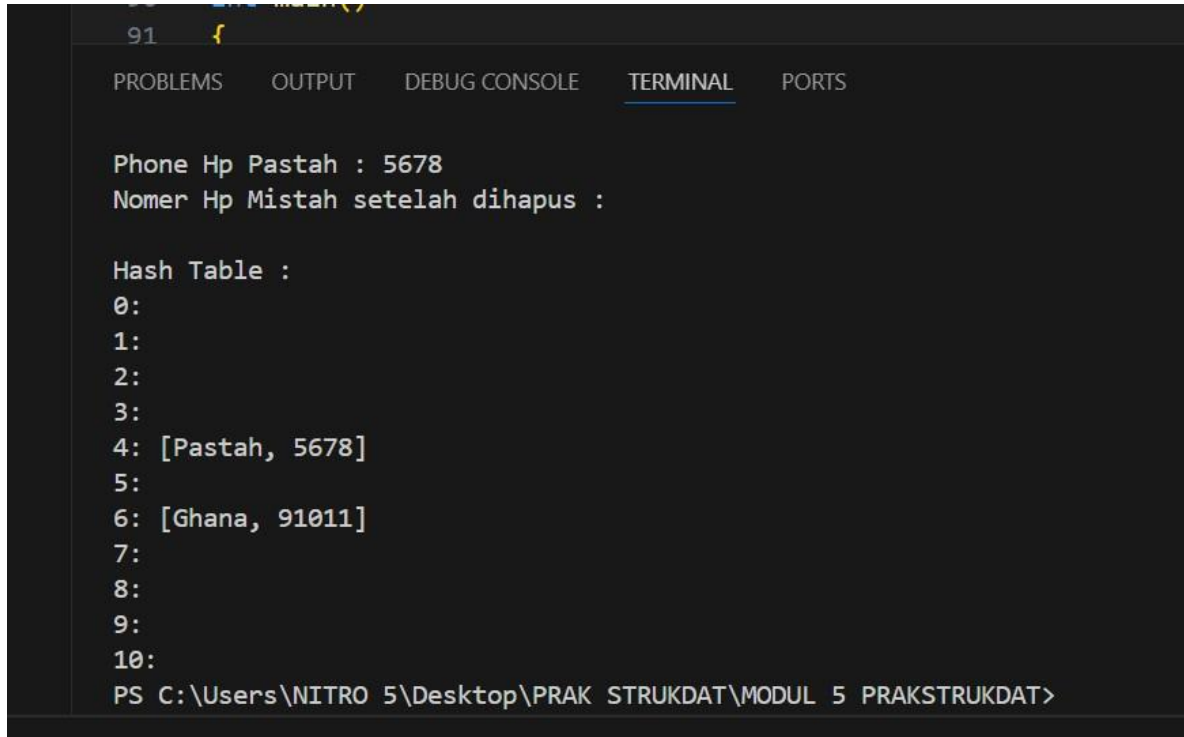
```

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



```
91 {  
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS  
  
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:\Users\NITRO 5\Desktop\PRAK STRUKDAT\MODUL 5 PRAKSTRUKDAT>
```

Deskripsi Program

Program C++ ini mengimplementasikan sebuah phonebook sederhana menggunakan struktur data Hash Table. Program ini memungkinkan pengguna untuk menyimpan, mencari, dan menghapus kontak dari phonebook.

Program ini mendemonstrasikan penggunaan Hash Table untuk membangun sebuah phonebook sederhana. Hash Table memungkinkan penyimpanan dan pencarian kontak yang efisien berdasarkan nama. Program ini dapat dimodifikasi untuk menambahkan fitur-fitur lain, seperti sorting kontak berdasarkan nama, validasi input pengguna, dan penanganan ukuran tabel hash yang penuh.

BAB V

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 <string>
#include <vector>
#include <unordered_map>

using namespace std;

struct Mahasiswa {
    string nim;
    int nilai;
};

void addMahasiswa(unordered_map<string, Mahasiswa>& dataMahasiswa) {
    string nim, nilaiStr;
    cout << "Masukkan NIM: ";
    cin >> nim;
    cout << "Masukkan nilai: ";
    cin >> nilaiStr;

    int nilai = stoi(nilaiStr);

    if (dataMahasiswa.find(nim) != dataMahasiswa.end()) {
        cout << "NIM sudah ada. Silakan masukkan NIM lain." << endl;
        return;
    }

    dataMahasiswa[nim] = {nim, nilai};
    cout << "Data mahasiswa berhasil ditambahkan." << endl;
}
```

```

void deleteMahasiswa(unordered_map<string, Mahasiswa>& dataMahasiswa) {
    string nim;
    cout << "Masukkan NIM: ";
    cin >> nim;

    if (dataMahasiswa.find(nim) == dataMahasiswa.end()) {
        cout << "NIM tidak ditemukan." << endl;
        return;
    }

    dataMahasiswa.erase(nim);
    cout << "Data mahasiswa dengan NIM " << nim << " berhasil dihapus." << endl;
}

void findMahasiswaByNIM(const unordered_map<string, Mahasiswa>& dataMahasiswa)
{
    string nim;
    cout << "Masukkan NIM: ";
    cin >> nim;

    if (dataMahasiswa.find(nim) == dataMahasiswa.end()) {
        cout << "NIM tidak ditemukan." << endl;
        return;
    }

    const Mahasiswa& mahasiswa = dataMahasiswa.at(nim);
    cout << "NIM: " << mahasiswa.nim << endl;
    cout << "Nilai: " << mahasiswa.nilai << endl;
}

void findMahasiswaByNilai(const unordered_map<string, Mahasiswa>&
dataMahasiswa) {
    cout << "Mencari data mahasiswa dengan nilai antara 80 dan 90..." << endl;

    for (const auto& pair : dataMahasiswa) {
        const Mahasiswa& mahasiswa = pair.second;
        if (mahasiswa.nilai >= 80 && mahasiswa.nilai <= 90) {
            cout << "NIM: " << mahasiswa.nim << endl;
            cout << "Nilai: " << mahasiswa.nilai << endl;
        }
    }
}

void showMenu() {

```

```

    cout << "\nMenu:" << endl;
    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 Nilai (80 - 90)" << endl;
    cout << "5. Keluar" << endl;
    cout << "Pilih menu: ";
}

int main() {
    unordered_map<string, Mahasiswa> dataMahasiswa;

    int pilihan;
    do {
        showMenu();
        cin >> pilihan;

        switch (pilihan) {
            case 1:
                addMahasiswa(dataMahasiswa);
                break;
            case 2:
                deleteMahasiswa(dataMahasiswa);
                break;
            case 3:
                findMahasiswaByNIM(dataMahasiswa);
                break;
            case 4:
                findMahasiswaByNilai(dataMahasiswa);
                break;
            case 5:
                cout << "Keluar dari program." << endl;
                break;
            default:
                cout << "Pilihan tidak valid. Silakan masukkan pilihan yang benar." <<
endl;
        }
    } while (pilihan != 5);

    return 0;
}

```

Screenshoot Program

```
PS C:\Users\NITRO 5\Desktop\PRAK STRUKDAT\MODUL 5 PRAKSTRUKDAT> g++ rFile.cpp -o rFile.exe && .\rFile.exe  
rFile } ; if ($?) { .\tempCodeRunnerFile }
```

Menu:

1. Tambah Data Mahasiswa
2. Hapus Data Mahasiswa
3. Cari Data Mahasiswa Berdasarkan NIM
4. Cari Data Mahasiswa Berdasarkan Nilai (80 - 90)
5. Keluar

Pilih menu: █

Opsi Tambah Data

```
26 cout << "NIM sudah ada. Silakan masukkan NIM lain!\n";  
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS  
Menu:  
1. Tambah Data Mahasiswa  
2. Hapus Data Mahasiswa  
3. Cari Data Mahasiswa Berdasarkan NIM  
4. Cari Data Mahasiswa Berdasarkan Nilai (80 - 90)  
5. Keluar  
Pilih menu: 1  
Masukkan NIM: 2311102025  
Masukkan nilai: 90  
Data mahasiswa berhasil ditambahkan.
```

Opsi Hapus Data

```
Menu:  
1. Tambah Data Mahasiswa  
2. Hapus Data Mahasiswa  
3. Cari Data Mahasiswa Berdasarkan NIM  
4. Cari Data Mahasiswa Berdasarkan Nilai (80 - 90)  
5. Keluar  
Pilih menu: 2  
Masukkan NIM: 2311102023  
Data mahasiswa dengan NIM 2311102023 berhasil dihapus.
```

Opsi Mencari Data berdasarkan NIM

```
26 | cout << "NIM sudah ada. Silakan masukkan NIM lain." << endl;
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Menu:
1. Tambah Data Mahasiswa
2. Hapus Data Mahasiswa
3. Cari Data Mahasiswa Berdasarkan NIM
4. Cari Data Mahasiswa Berdasarkan Nilai (80 - 90)
5. Keluar
Pilih menu: 2
Masukkan NIM: 2311102023
Data mahasiswa dengan NIM 2311102023 berhasil dihapus.
```

Opsi Mencari Data berdasarkan rentang nilai (80-90)

```
PROBLEMS OUTPUT TERMINAL ... Code + - [] .

Menu:
1. Tambah Data Mahasiswa
2. Hapus Data Mahasiswa
3. Cari Data Mahasiswa Berdasarkan NIM
4. Cari Data Mahasiswa Berdasarkan Nilai (80 - 90)
5. Keluar
Pilih menu: 4
Mencari data mahasiswa dengan nilai antara 80 dan 90...
NIM: 2311102024
Nilai: 84
NIM: 2311102020
Nilai: 80
NIM: 2311102021
Nilai: 82
NIM: 2311102025
Nilai: 90
NIM: 2311102022
Nilai: 87
```

Deskripsi Program

Program ini adalah aplikasi sederhana yang digunakan untuk mengelola data mahasiswa. Program ini memungkinkan pengguna untuk menambahkan, menghapus, dan mencari data mahasiswa berdasarkan rentang nilai atau NIM. Data mahasiswa disimpan dalam sebuah struktur data yang disebut **unordered_map**

Program ini menyediakan fungsionalitas dasar untuk mengelola data mahasiswa, termasuk menambahkan, menghapus, mencari, dan menampilkan informasi mahasiswa. Penggunaan struktur data **unordered_map** memungkinkan penyimpanan dan pencarian data yang efisien. Program ini dapat dimodifikasi lebih lanjut untuk menambahkan fitur-fitur lain, seperti mengubah nilai mahasiswa, menampilkan statistik data, dan sebagainya.

BAB V

KESIMPULAN

Pada praktikum ini, kami berkesempatan untuk mempelajari dan memperdalam pemahaman tentang Hash Table, salah satu struktur data yang penting dalam ilmu komputer. Hash Table adalah metode penyimpanan dan pengambilan data yang efisien, dengan kompleksitas waktu akses rata-rata $O(1)$, membuatnya sangat berguna dalam banyak aplikasi.

Selama praktikum, kami diajarkan tentang prinsip kerja hash table, termasuk peran fungsi hash dalam mengubah kunci (key) menjadi indeks dalam array. Kami juga mempelajari tentang isu-isu seperti kolisi dan bagaimana penanganannya, baik melalui metode chaining maupun open addressing.

Dalam implementasi praktik, kami menggunakan bahasa pemrograman C++ untuk membangun hash table sederhana dan menerapkan konsep-konsep yang telah dipelajari. Kami menguji hash table kami dengan melakukan operasi penambahan, penghapusan, dan pencarian data, serta menganalisis bagaimana hash table menangani masing-masing operasi tersebut.

Praktikum ini juga membantu kami memahami keterbatasan hash table dan kondisi-kondisi tertentu yang dapat memengaruhi kinerjanya, seperti pemilihan fungsi hash yang tidak efisien atau ketika terjadi banyak kolisi. Kami belajar bahwa desain yang baik dan pemilihan fungsi hash yang tepat sangat penting untuk optimalisasi kinerja hash table.

Secara keseluruhan, praktikum ini telah memberikan wawasan yang berharga tentang hash table dan bagaimana mereka digunakan dalam pemrograman. Kami telah memperoleh pemahaman tentang konsep dasar, implementasi, dan aplikasi hash table dalam konteks yang lebih luas. Kami berharap bahwa pengetahuan dan keterampilan yang diperoleh selama praktikum ini akan dapat diterapkan dalam proyek-proyek pemrograman di masa depan, serta dalam pemecahan masalah yang melibatkan pengolahan data secara efisien.

DAFTAR PUSTAKA

- [1] Karumanchi, N. 2016. Data Structures and algorithms made easy: Concepts, problems, Interview Questions. CareerMonk Publications.
- [2] Fsyeka Gyanmar.2018.Analisis Performansi QoS menggunakan Algoritma Distribution Hash Table. Diakses 12 Mei 2024, dari <https://openlibrary.telkomuniversity.ac.id/pustaka/144582/analisis-peformansi-qos-pada-easyrtc-menggunakan-algoritma-distribution-hash-table.html>