LAPORAN PRAKTIKUM STRUKTUR DATA DAN ALGORITMA

MODUL 5
HASH TABLE

Disusun Oleh :

Muhammad Hamzah Haifan Ma'ruf

231102091

Dosen :

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

PROGRAM STUDI S1 TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO

2024

A. Dasar Teori

Hash table merupakan struktur data yang secara asosiatif menyimpan data. Dalam hal ini, data disimpan dalam format array, di mana setiap nilai data memiliki nilai indeks uniknya sendiri. Akses data akan menjadi sangat cepat jika Anda mengetahui indeks dari data yang diinginkan. Dengan demikian, hash table menjadi struktur data di mana operasi penyisipan dan pencarian data terjadi sangat cepat terlepas dari ukuran data tersebut. Hash table menggunakan array sebagai media penyimpanan dan tekniknya untuk menghasilkan indeks suatu elemen yang dimasukkan atau ditempatkan.

Untuk membuat hash table, sepotong memori perlu diblokir dengan cara yang sama seperti saat membuat array. Anda perlu membuat indeks yang didasarkan pada kunci dengan menggunakan fungsi hash karena indeks yang dihasilkan harus sesuai dengan potongan memori. Ada dua pemeriksaan yang dibutuhkan saat menempatkan data baru pada hash table, yaitu nilai hash dari kunci dan bagaimana nilainya dibandingkan dengan objek lain. Pemeriksaan ini diperlukan saat membuatnya dengan Python karena saat data dimasukkan, kunci akan di-hash dan di-mask agar diubah menjadi larik atau indeks yang efisien.

1. Hashing

   Hashing merupakan sebuah proses mengganti kunci yang diberikan atau string karakter menjadi nilai lain. Penggunaan hashing paling populer adalah pada hash table. Hash table menyimpan pasangan kunci dan nilai dalam daftar yang dapat diakses melalui indeksnya. Karena pasangan kunci dan nilai tidak terbatas, maka fungsinya akan memetakan kunci ke ukuran tabel dan kemudian nilainya menjadi indeks untuk elemen tertentu.

2. Liniar Probing

   Linear probing merupakan skema dalam pemrograman komputer untuk menyelesaikan collision pada hash table. Dalam skema ini, setiap sel dari hash table menyimpan satu pasangan kunci-nilai. Saat fungsi hash menyebabkan collision dengan memetakan kunci baru ke sel hash table yang sudah ditempati oleh kunci lain, maka linear probing akan mencari tabel untuk lokasi bebas terdekat dan menyisipkan kunci baru. Pencarian dilakukan dengan cara yang sama, yaitu dengan mencari tabel secara berurutan, mulai dari posisi yang diberikan oleh fungsi hash, hingga menemukan sel dengan kunci yang cocok atau sel kosong. Hash table adalah struktur data non trivial yang paling umum digunakan. Linear probing dapat memberikan kinerja tinggi karena lokasi referensi yang baik, namun lebih sensitif terhadap kualitas fungsi hash daripada beberapa skema resolusi collision lainnya

B. **Guided**

Guided 1

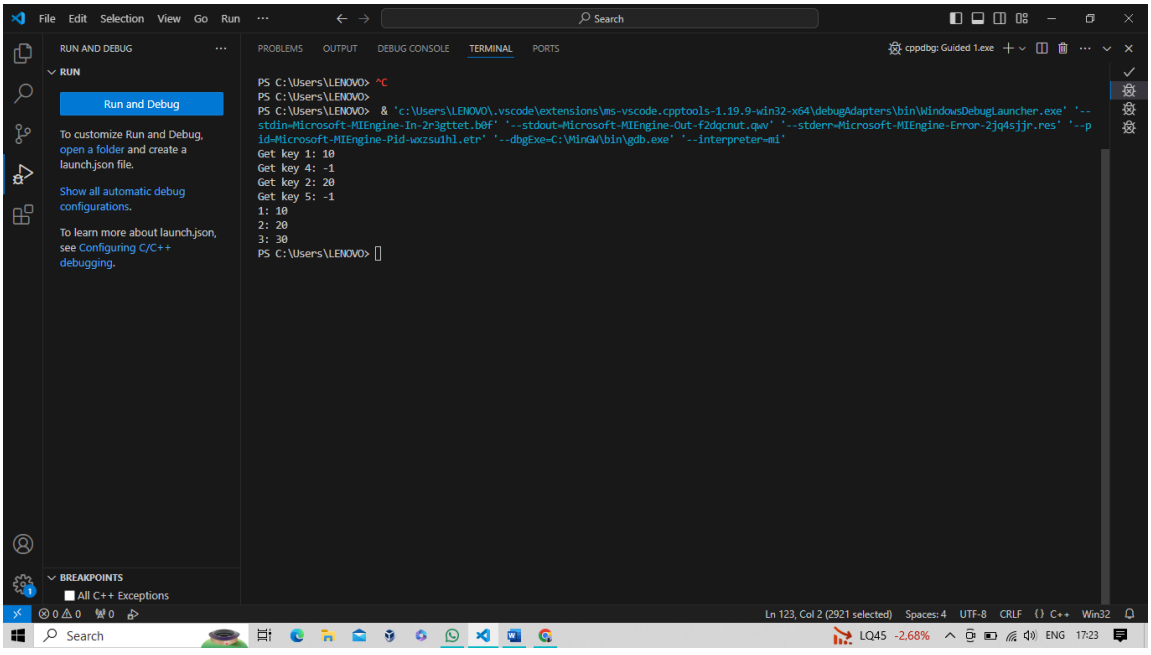
Source Code

```
#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;
    cout << "Get key 2: " << ht.get(2) << endl;
    cout << "Get key 5: " << ht.get(5) << endl;

    // Deletion
    ht.remove(4);

    // Traversal
    ht.traverse();

    return 0;
}
```

Screenshots Output :



Deskripsi :

Program di atas adalah untuk menyimpan pasangan kunci-nilai (key-value pairs) dan memungkinkan operasi-operasi seperti penambahan, pencarian, dan penghapusan dengan waktu yang cepat. Dalam program ini, hash table direpresentasikan sebagai sebuah array dinamis dari pointer-pointer ke node-node. Setiap node menyimpan kunci dan nilai, serta pointer ke node berikutnya untuk menangani tumpang tindih (collision). Program memiliki metode-metode untuk operasi dasar hash table seperti penambahan, pencarian, penghapusan, dan penelusuran. Ini memberikan pemahaman dasar tentang bagaimana hash table bekerja dan bagaimana itu dapat diimplementasikan dalam bahasa pemrograman C++.

Guided 2

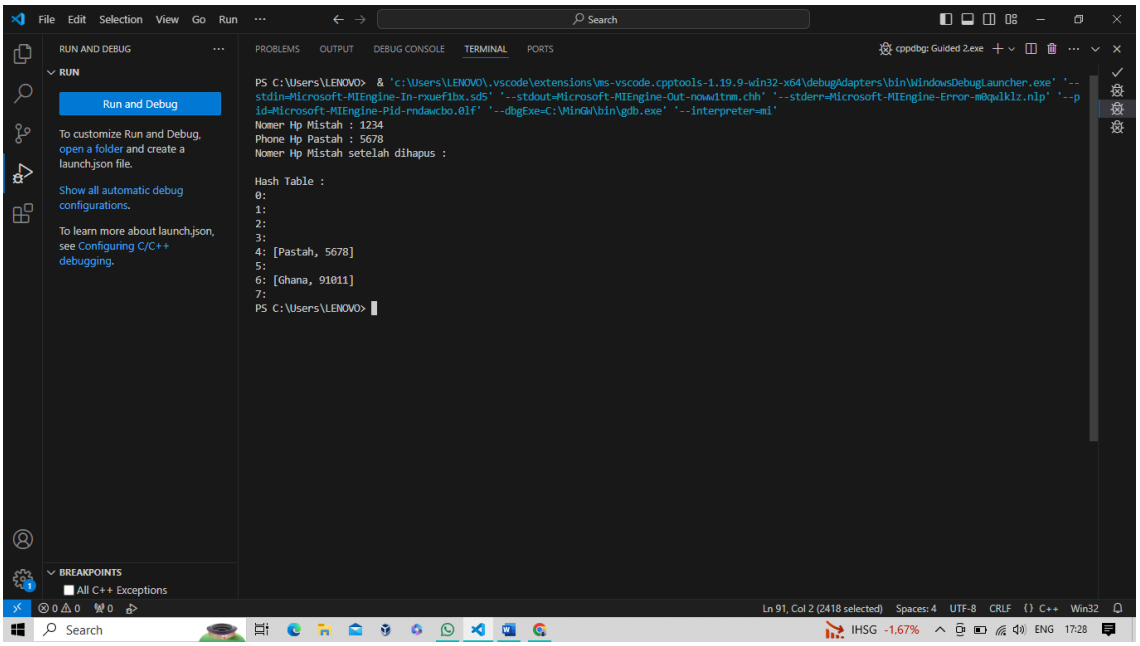
Source Code

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

const int TABLE_SIZE = 11;

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;
}
```

Screenshots Output :



Deskripsi :

Program hash map ini, setiap entri terdiri dari sepasang kunci-nilai, di mana kunci adalah nama dan nilai adalah nomor telepon. Implementasi menggunakan array dari vektor, di mana setiap vektor diindeks oleh fungsi hash yang mengonversi nama menjadi indeks dalam array. Setiap vektor dapat memiliki beberapa node yang menyimpan nama dan nomor telepon. Program ini memiliki metode untuk menyisipkan pasangan kunci-nilai, menghapus berdasarkan kunci, mencari nomor telepon berdasarkan nama, dan mencetak isi hash map. Melalui contoh penggunaan dalam fungsi main, program menunjukkan operasi-operasi dasar seperti penyisipan, pencarian, dan penghapusan, serta mencetak isi hash map setelah beberapa operasi dilakukan.

C. Unguided

Unguided 1

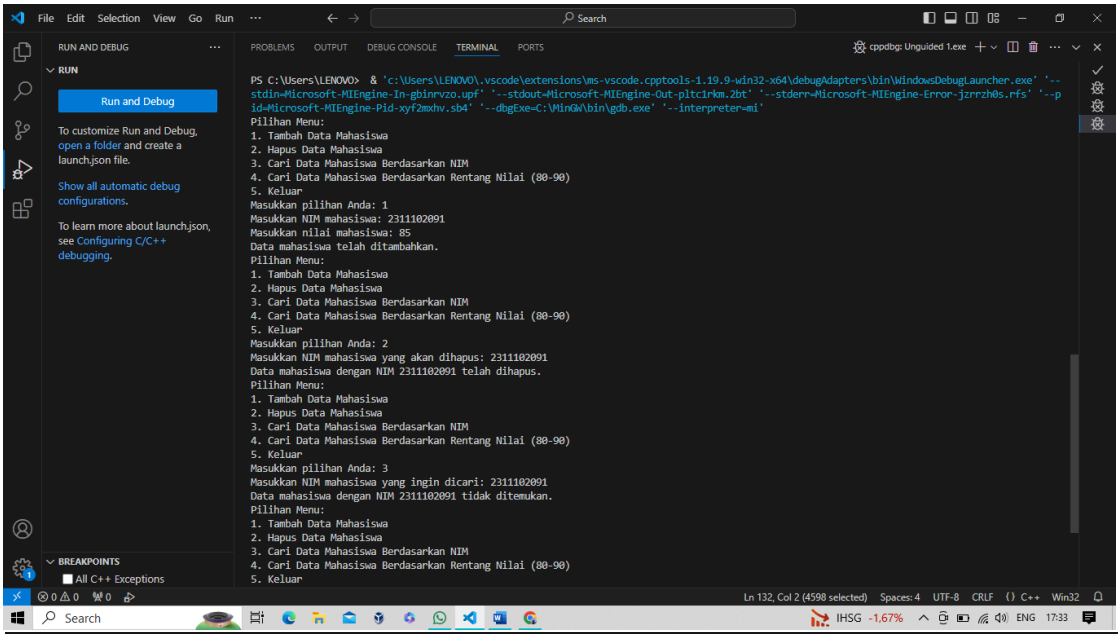
Source Code

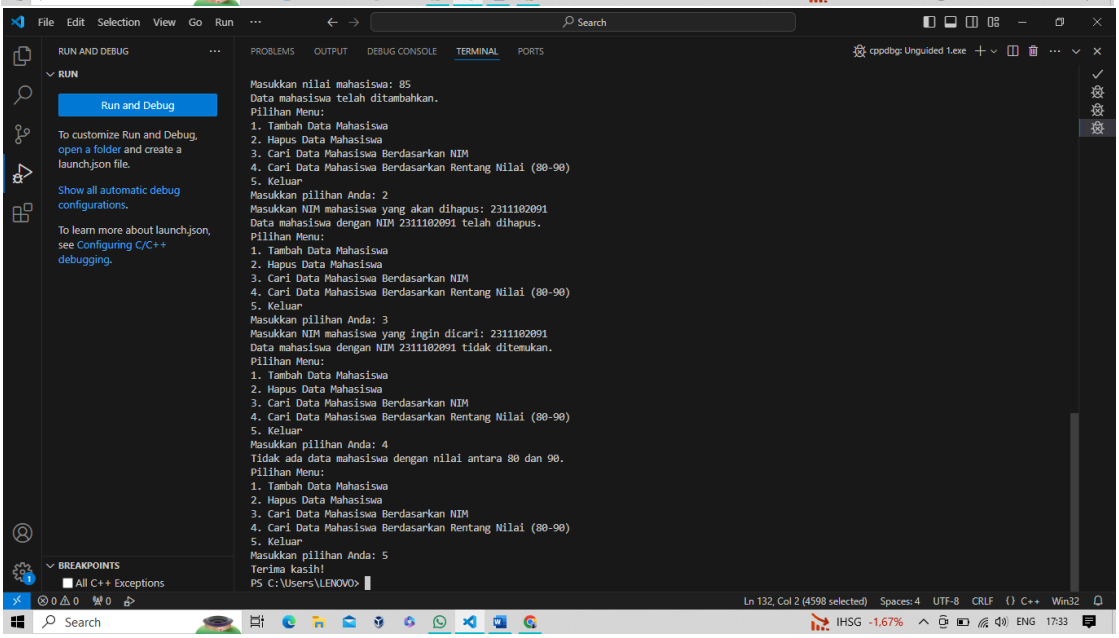
```
#include <iostream>
#include <vector>
#include <string>
using namespace std;

struct Mahasiswa {
    string nim_091;
    float nilai_091;
};

class HashTable {
private:
    static const int table_size = 10; // Ukuran hash table disetel ke 10
untuk contoh
    vector<Mahasiswa> table[table_size];

    int hash_func(string nim_091) {
        int sum = 0;
        for (char c : nim_091) {
            sum += int(c);
        }
        return sum % table_size;
    }

public:
    void tambah_data(Mahasiswa mahasiswa) {
        int index = hash_func(mahasiswa.nim_091);
        table[index].push_back(mahasiswa);
    }

    bool hapus_data(string nim_091) {
        int index = hash_func(nim_091);
        for (int i = 0; i < table[index].size(); ++i) {
            if (table[index][i].nim_091 == nim_091) {
                table[index].erase(table[index].begin() + i);
                return true;
            }
        }
        return false;
    }

    Mahasiswa* cari_berdasarkan_nim(string nim_091) {
        int index = hash_func(nim_091);
        for (Mahasiswa &mahasiswa : table[index]) {
            if (mahasiswa.nim_091 == nim_091) {
```

```
                return &mahasiswa;
            }
        }
        return nullptr;
    }

    vector<Mahasiswa> cari_berdasarkan_rentang_nilai(float nilai_awal,
float nilai_akhir) {
        vector<Mahasiswa> mahasiswa_ditemukan;
        for (int i = 0; i < table_size; ++i) {
            for (Mahasiswa &mahasiswa : table[i]) {
                if (mahasiswa.nilai_091 >= nilai_awal &&
mahasiswa.nilai_091 <= nilai_akhir) {
                    mahasiswa_ditemukan.push_back(mahasiswa);
                }
            }
        }
        return mahasiswa_ditemukan;
    }
};

// Fungsi untuk menampilkan menu
void tampilkan_menu() {
    cout << "Pilihan Menu:" << 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 Rentang Nilai (80-90)" <<
endl;
    cout << "5. Keluar" << endl;
}

int main() {
    HashTable hash_table;

    while (true) {
        tampilkan_menu();
        int pilihan;
        cout << "Masukkan pilihan Anda: ";
        cin >> pilihan;

        if (pilihan == 1) {
            Mahasiswa mahasiswa;
            cout << "Masukkan NIM mahasiswa: ";
            cin >> mahasiswa.nim_091;
            cout << "Masukkan nilai mahasiswa: ";
            cin >> mahasiswa.nilai_091;
            hash_table.tambah_data(mahasiswa);
            cout << "Data mahasiswa telah ditambahkan." << endl;
```

```
        } else if (pilihan == 2) {
            string nim_091;
            cout << "Masukkan NIM mahasiswa yang akan dihapus: ";
            cin >> nim_091;
            if (hash_table.hapus_data(nim_091)) {
                cout << "Data mahasiswa dengan NIM " << nim_091 << "
telah dihapus." << endl;
            } else {
                cout << "Data mahasiswa dengan NIM " << nim_091 << "
tidak ditemukan." << endl;
            }
        } else if (pilihan == 3) {
            string nim_091;
            cout << "Masukkan NIM mahasiswa yang ingin dicari: ";
            cin >> nim_091;
            Mahasiswa* mahasiswa =
hash_table.cari_berdasarkan_nim(nim_091);
            if (mahasiswa != nullptr) {
                cout << "Data mahasiswa ditemukan:" << endl;
                cout << "NIM: " << mahasiswa->nim_091 << endl;
                cout << "Nilai: " << mahasiswa->nilai_091 << endl;
            } else {
                cout << "Data mahasiswa dengan NIM " << nim_091 << "
tidak ditemukan." << endl;
            }
        } else if (pilihan == 4) {
            vector<Mahasiswa> mahasiswa_ditemukan =
hash_table.cari_berdasarkan_rentang_nilai(80, 90);
            if (!mahasiswa_ditemukan.empty()) {
                cout << "Data mahasiswa dengan nilai antara 80 dan 90:"
<< endl;
                for (Mahasiswa &mahasiswa : mahasiswa_ditemukan) {
                    cout << "NIM: " << mahasiswa.nim_091 << endl;
                    cout << "Nilai: " << mahasiswa.nilai_091 << endl;
                }
            } else {
                cout << "Tidak ada data mahasiswa dengan nilai antara 80
dan 90." << endl;
            }
        } else if (pilihan == 5) {
            cout << "Terima kasih!" << endl;
            break;
        } else {
            cout << "Pilihan tidak valid. Silakan pilih lagi." << endl;
        }
    }

    return 0;
}
```

Screenshots Output :





Deskripsi :

Program di atas untuk menyimpan dan mengelola data mahasiswa berdasarkan NIM dan nilai. Kelas HashTable memiliki atribut berupa array dari vektor, di mana setiap elemen vektor menyimpan daftar mahasiswa yang di-hash ke dalam indeks tertentu menggunakan fungsi hashing. Fungsi hash_func mengonversi NIM mahasiswa menjadi indeks dalam hash table dengan menjumlahkan nilai ASCII dari setiap karakter dalam NIM dan melakukan modulo dengan ukuran hash table.

Metode dalam kelas HashTable memungkinkan pengguna untuk menambahkan data mahasiswa, menghapus data berdasarkan NIM, mencari data berdasarkan NIM, dan mencari data berdasarkan rentang nilai tertentu. Dalam fungsi main, program menampilkan menu pilihan operasi kepada pengguna dan menjalankan operasi yang

dipilih. Alur operasi berlanjut hingga pengguna memilih untuk keluar dari program. Dengan struktur yang disediakan, program ini memberikan fungsionalitas yang memadai untuk manajemen data mahasiswa menggunakan hash tabl

D. Kesimpulan

Hash table adalah struktur data yang efisien dan kuat dalam menyimpan dan mengakses data dengan cepat menggunakan kunci tertentu. Ketiga contoh implementasi hash table di atas menunjukkan berbagai pendekatan dalam menggunakan struktur data ini untuk memecahkan masalah yang berbeda. Contoh pertama memberikan dasar operasi hash table seperti penambahan, pencarian, dan penghapusan data dengan teknik chaining. Contoh kedua menunjukkan penggunaan hash map untuk manajemen data yang lebih kompleks, sementara contoh ketiga menggunakan hash table untuk menyimpan dan mencari data mahasiswa berdasarkan NIM dan nilai. Dengan pemahaman yang baik tentang konsep dan implementasi hash table, kita dapat menyelesaikan berbagai masalah pemrograman dengan efisiensi tinggi.

E. Referensi

Bunga (2022) *Apa itu Hash Table dan Bagaimana Penggunaannya?* https://algorit.ma/blog/hash-table-adalah-2022/.

Ram, V. (2023) *How to implement a sample hash table in C/C++*. https://www.digitalocean.com/community/tutorials/hash-table-in-c-plus-plus.

Hash table for C++ (no date). https://gist.github.com/clarkngo/ed6f71f880453baf71481063407fe2e8.