

**LAPORAN PRAKTIKUM  
STRUKTUR DATA DAN ALGORITMA**

**MODUL V  
HASH TABLE**



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

**Disusun oleh:**

**MUHAMMAD AULIA MUZZAKI NUGRAHA**

**(2311102051)**

**IF-11-B**

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

**2024**

# BAB I

## DASAR TEORI

### A. Dasar Teori

Hash table adalah struktur data yang digunakan untuk menyimpan data dalam bentuk pasangan kunci-nilai. Ini memungkinkan penyimpanan dan pencarian data yang efisien dengan menggunakan teknik hash function. Hash table sering digunakan dalam berbagai aplikasi, seperti database, kamus, dan pelacakan data.

- **Komponen Utama:**

1. **Hash Function:** Ini adalah fungsi yang mengonversi kunci (misalnya, string atau bilangan bulat) menjadi indeks dalam tabel hash. Fungsi ini harus menghasilkan nilai yang unik atau minimal menyebar secara merata di seluruh jangkauan indeks tabel hash.
2. **Array atau Tabel Hash:** Ini adalah struktur data dasar yang berfungsi sebagai wadah untuk menyimpan pasangan kunci-nilai. Setiap entri dalam tabel hash dapat berupa array, linked list, atau struktur data lainnya untuk menangani kasus bentrok.
3. **Collision Resolution:** Collision terjadi ketika dua kunci berbeda di-mapped ke indeks yang sama dalam tabel hash. Teknik resolusi bentrok digunakan untuk menangani situasi ini. Beberapa teknik resolusi bentrok yang umum termasuk chaining (menggunakan linked list di setiap sel hash) dan probing (mencari sel kosong terdekat dalam tabel hash).

- **Operasi Utama:**

1. **Insertion (Penyisipan):** Operasi ini digunakan untuk menambahkan pasangan kunci-nilai ke dalam hash table. Pertama, kunci diubah menjadi indeks menggunakan hash function. Jika terjadi collision, teknik resolusi bentrok digunakan untuk menangani kasus ini.
2. **Retrieval (Pengambilan):** Operasi ini digunakan untuk mencari nilai yang terkait dengan kunci yang diberikan dalam hash table. Kunci diubah

menjadi indeks menggunakan hash function, dan nilai yang terkait dengan indeks tersebut dikembalikan.

3. **Deletion (Penghapusan):** Operasi ini digunakan untuk menghapus pasangan kunci-nilai dari hash table. Kunci yang diberikan diubah menjadi indeks menggunakan hash function, dan entri yang terkait dihapus dari tabel hash.

- **Keuntungan:**

1. **Kecepatan Akses Tinggi:** Operasi seperti pencarian, penyisipan, dan penghapusan dapat dilakukan dalam waktu konstan ( $O(1)$ ) dalam kasus rata-rata, membuat hash table sangat efisien untuk digunakan.
2. **Penggunaan Memori Efisien:** Hash table hanya menggunakan memori sesuai dengan jumlah item yang disimpan, yang membuatnya efisien dalam penggunaan memori dibandingkan dengan struktur data lain yang memerlukan alokasi memori statis.

- **Keterbatasan:**

1. **Kinerja Terburuk yang Dapat Diterima (Worst-Case Performance):** Dalam beberapa kasus, hash table dapat memiliki kinerja yang buruk jika hash function tidak seimbang atau jika collision terjadi terlalu sering. Ini dapat mengakibatkan waktu eksekusi meningkat menjadi  $O(n)$ , di mana  $n$  adalah jumlah item dalam hash table.
2. **Keterbatasan Kapasitas:** Ukuran tabel hash harus dipilih dengan hati-hati. Jika terlalu kecil, ini dapat menyebabkan collision yang berlebihan. Jika terlalu besar, ini dapat menyebabkan pemborosan memori.

## BAB II

### GUIDED

#### LATIHAN – GUIDED

##### 1. Guided 1

Guided (berisi screenshot source code & output program disertai penjelasannya)

##### 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;
        }
    }

```

```

        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**

```
PS F:\Semester u Andi Saputra,
Get key 1: 10
Get key 4: -1
1: 10
2: 20
3: 30
PS F:\Semester
```

File Edit Lihat

Nama : Muhammad Aulia Muzzaki Nugraha  
NIM : 2311102051  
Kelas : IF11B

Ln 3, Col 14 69 karakter 100% Window UTF-8

### Deskripsi program

Hash table digunakan untuk menyimpan pasangan kunci-nilai (key-value pairs). Setiap elemen dalam hash table diakses menggunakan kunci yang ditentukan. Kode ini menyediakan fungsi dasar untuk menyisipkan (insert), mencari (get), menghapus (remove), dan melakukan traversal terhadap elemen-elemen dalam hash table. Fungsi hash yang digunakan adalah hash\_func, yang melakukan operasi modulo pada kunci untuk menentukan indeks di mana elemen tersebut akan disimpan dalam array yang mewakili hash table. Setiap elemen hash table direpresentasikan oleh struktur data Node, yang berisi kunci, nilai, dan pointer ke node berikutnya dalam kasus terjadi tumpang tindih (collision).

## 2. Guided 2

### Source code

```
#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();

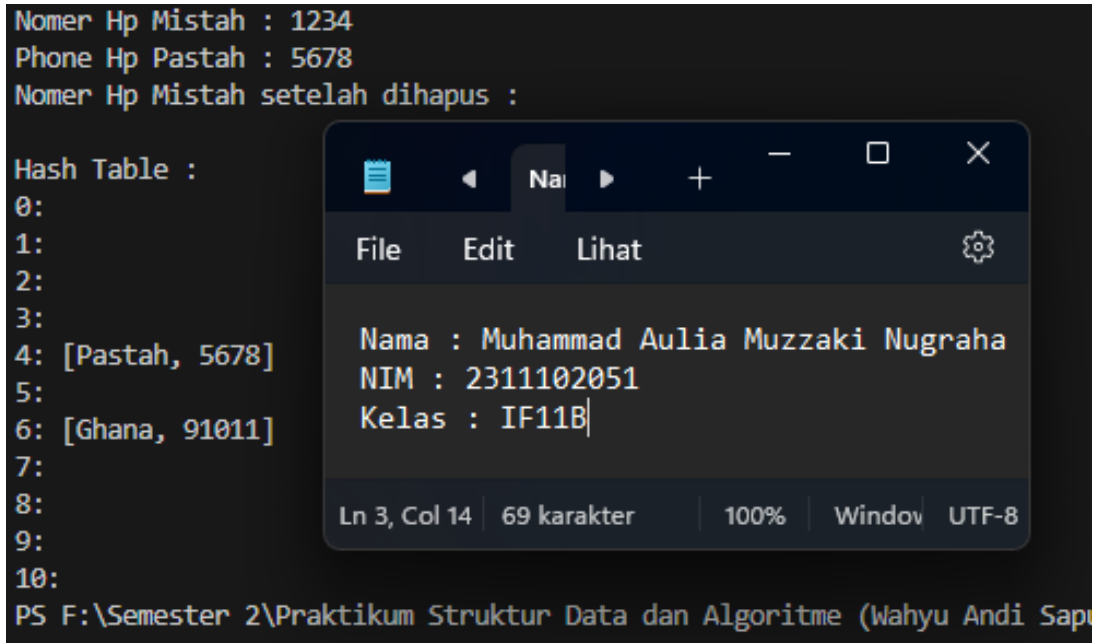
    return 0;
}

```

**Screenshoot program**

```
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 F:\Semester 2\Praktikum Struktur Data dan Algoritme (Wahyu Andi Sapt...
```

The image shows a terminal window with a dark background. The terminal output displays a hash table with 11 slots (0-10). Slots 4 and 6 contain entries: [Pastah, 5678] and [Ghana, 91011] respectively. Other slots are empty. A text editor window is overlaid on the terminal, showing a file named 'Nama'. The editor contains the text: 'Nama : Muhammad Aulia Muzzaki Nugraha', 'NIM : 2311102051', and 'Kelas : IF11B'. The editor's status bar at the bottom indicates 'Ln 3, Col 14', '69 karakter', '100%', 'Window', and 'UTF-8'.

### Deskripsi program

Dalam hash map ini, setiap entri terdiri dari sepasang kunci-nilai, di mana kunci berupa string nama dan nilai berupa nomor telepon. Implementasi menggunakan array dari vektor, di mana setiap vektor menyimpan pointer ke objek HashNode yang mengandung pasangan kunci-nilai. Fungsi hash digunakan untuk menentukan indeks di mana entri akan disimpan dalam array. Fungsi-fungsi seperti insert, remove, dan searchByName digunakan untuk operasi penyisipan, penghapusan, dan pencarian berdasarkan nama. Metode print digunakan untuk mencetak seluruh isi hash map, menunjukkan pasangan kunci-nilai yang tersimpan dalam setiap slot tabel hash.

## BAB III

### UNGUIDED

#### TUGAS – UNGUIDED

##### 1. Unguided 1

Unguided/Tugas (berisi screenshot source code & output program disertai penjelasannya)

Buatlah program menu Linked List Non Circular untuk menyimpan Nama dan NIM mahasiswa, dengan menggunakan input dari user

#### Source Code

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

// Struktur data untuk setiap mahasiswa
struct Mahasiswa {
    string nim;
    string nama;
    int nilai;
    Mahasiswa(string nim, string nama, int nilai) : nim(nim),
nama(nama), nilai(nilai) {}
};

// Class HashNode untuk setiap node pada hash table
class HashNode {
public:
    string nim;
    string nama;
    int nilai;
    HashNode(string nim, string nama, int nilai) : nim(nim),
nama(nama), nilai(nilai) {}
};

// Class HashMap untuk hash table
class HashMap {
private:
    static const int TABLE_SIZE = 10;
    vector<HashNode*> table[TABLE_SIZE];

public:
    // Fungsi hash sederhana
    int hashFunc(string key) {
        int hash_val = 0;
```

```

        for (char c : key) {
            hash_val += c;
        }
        return hash_val % TABLE_SIZE;
    }

    // Menambahkan data mahasiswa baru
    void insert(string nim, string nama, int nilai) {
        int hash_val = hashFunc(nim);
        table[hash_val].push_back(new HashNode(nim, nama, nilai));
    }

    // Menghapus data mahasiswa berdasarkan NIM
    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;
            }
        }
    }

    // Mencari data mahasiswa berdasarkan NIM
    int searchByNIM(string nim) {
        int hash_val = hashFunc(nim);
        for (auto node : table[hash_val]) {
            if (node->nim == nim) {
                return node->nilai;
            }
        }
        return -1; // Mahasiswa tidak ditemukan
    }

    // Mencari data mahasiswa berdasarkan rentang nilai (80 - 90)
    vector<string> searchByRange(int minNilai, int maxNilai) {
        vector<string> result;
        for (int i = 0; i < TABLE_SIZE; i++) {
            for (auto node : table[i]) {
                if (node->nilai >= minNilai && node->nilai <=
maxNilai) {
                    result.push_back(node->nama);
                }
            }
        }
        return result;
    }
}

```

```

};

int main() {
    HashMap mahasiswaMap;
    int choice;
    do {
        cout << "Menu:" << endl;
        cout << 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;
        cout << "Pilih: ";
        cin >> choice;

        cout << endl;

        switch (choice) {
            case 1: {
                string nim, nama;
                int nilai;
                cout << "Masukkan NIM: ";
                cin >> nim;
                cout << "Masukkan Nama: ";
                cin.ignore();
                getline(cin, nama);
                cout << "Masukkan nilai: ";
                cin >> nilai;
                mahasiswaMap.insert(nim, nama, nilai);
                cout << "Data mahasiswa ditambahkan." << endl;
                cout << endl ;
                break;
            }
            case 2: {
                string nim;
                cout << "Masukkan NIM yang ingin dihapus: ";
                cin >> nim;
                mahasiswaMap.remove(nim);
                cout << "Data mahasiswa dihapus." << endl;
                cout << endl ;
                break;
            }
            case 3: {
                string nim;
                cout << "Masukkan NIM yang ingin dicari: ";
                cin >> nim;

```

```

        int nilai = mahasiswaMap.searchByNIM(nim);
        if (nilai != -1) {
            cout << "Nilai mahasiswa dengan NIM " << nim <<
" adalah: " << nilai << endl;
        } else {
            cout << "Mahasiswa dengan NIM " << nim << "
tidak ditemukan." << endl;
        }
        cout << endl ;
        break;
    }
    case 4: {
        vector<string> mahasiswa80_90 =
mahasiswaMap.searchByRange(80, 90);
        if (mahasiswa80_90.size() > 0) {
            cout << "Mahasiswa dengan nilai antara 80 dan 90
adalah: ";

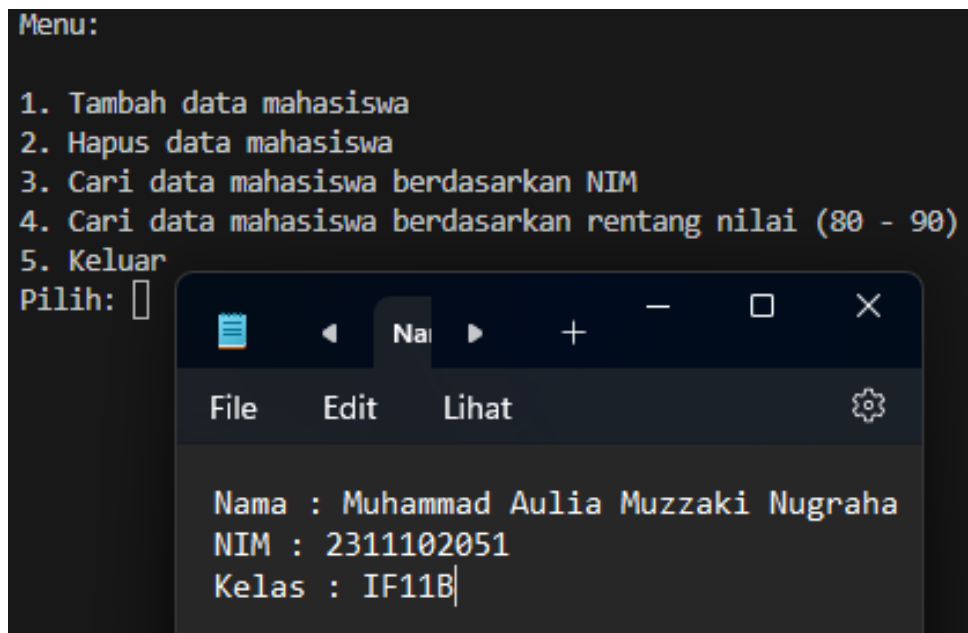
            for (string nama : mahasiswa80_90) {
                cout << nama << " ";
            }
            cout << endl;
        } else {
            cout << "Tidak ada mahasiswa dengan nilai antara
80 dan 90." << endl;
        }
        cout << endl ;
        break;
    }
    case 5: {
        cout << "Program selesai." << endl;
        cout << endl ;
        break;
    }
    default: {
        cout << "Pilihan tidak valid. Silakan pilih
kembali." << endl;
        cout << endl ;
        break;
    }
}
} while (choice != 5);

return 1;
}

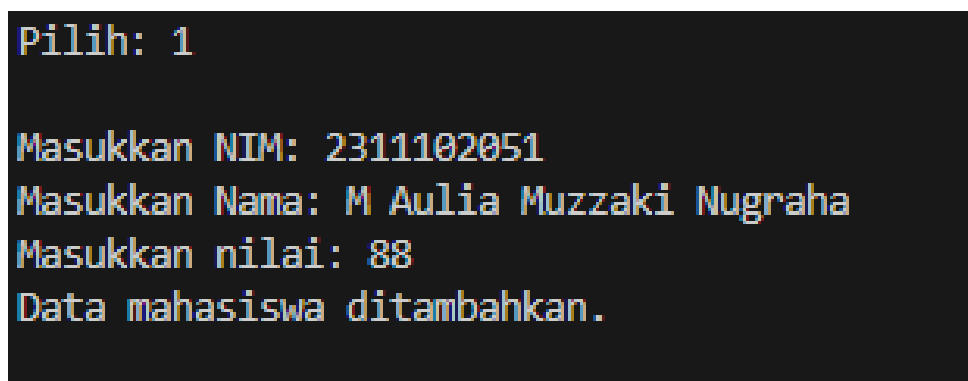
```

## Screenshot Program

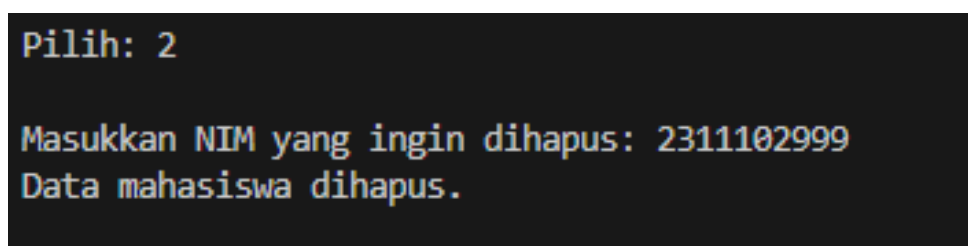
- Tampilan Menu



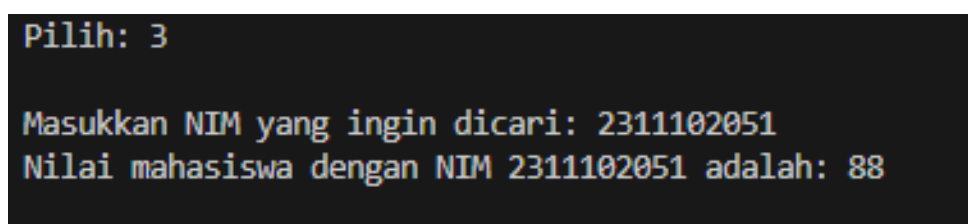
- Tambah Data



- Menghapus Data



- Cari Berdasarkan NIM



- **Cari Berdasarkan rentan nilai**

```
Pilih: 4
```

```
Mahasiswa dengan nilai antara 80 dan 90 adalah: Yusuf Daa M Aulia Muzzaki Nugraha
```

### **Deskripsi program**

Program hash map untuk menyimpan data mahasiswa berdasarkan Nomor Induk Mahasiswa (NIM) sebagai kunci dan informasi mahasiswa lainnya seperti nama dan nilai. Class **HashMap** memiliki beberapa method seperti **insert** untuk menambahkan data mahasiswa baru, **remove** untuk menghapus data mahasiswa berdasarkan NIM, **searchByNIM** untuk mencari data mahasiswa berdasarkan NIM, dan **searchByRange** untuk mencari data mahasiswa berdasarkan rentang nilai. Program tersebut menampilkan menu dengan beberapa pilihan aksi untuk melakukan manipulasi data mahasiswa seperti penambahan, penghapusan, dan pencarian, serta menampilkan hasil operasi yang dilakukan kepada pengguna.



## **BAB IV**

### **KESIMPULAN**

Hash table adalah salah satu struktur data yang paling penting dan sering digunakan dalam pemrograman komputer. Dengan menggunakan hash function untuk memetakan 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 ( $O(1)$ ) 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 (worst-case performance) dapat menjadi masalah jika hash function tidak seimbang atau jika collision terjadi terlalu 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 table.

Selain itu, hash table juga memiliki keuntungan dalam penggunaan memori yang efisien karena hanya menggunakan memori sesuai dengan jumlah item yang disimpan. Ini membuatnya menjadi pilihan yang baik dalam situasi di mana efisiensi memori menjadi perhatian utama. Selain digunakan dalam berbagai aplikasi seperti database, implementasi kamus, dan pemantauan penggunaan memori dalam sistem operasi, hash table juga memiliki penerapan luas dalam pemrograman umum. Namun, penting untuk memahami keterbatasan dan tantangan yang terkait dengan penggunaan hash table, seperti keterbatasan kapasitas dan potensi untuk kinerja terburuk, sehingga dapat mengambil langkah-langkah yang sesuai untuk mengatasi masalah tersebut dan memastikan kinerja yang optimal dari struktur data ini.

## DAFTAR PUSTAKA

- [1] Meidyan Permata Putri, Guntoro Barovih, Rezania Agramanisti Azdy, Yuniansyah, Andri Saputra, Yesi Sriyeni, Arsia Rini, Fadhila Tangguh Admojo. Algoritma dan Struktur Data. Widina Bhakti Persada, 2022. ISBN: 978-623-459-182-8
- [2] Rassokhin, D. (2020). *The C++ programming language in cheminformatics and computational chemistry. Journal of Cheminformatics, 12(10). Retrieved from Journal of Cheminformatics.*
- [3] Bender, M. A., Conway, A., Farach-Colton, M., Kuszmaul, W., & Tagliavini, G. (2021). *Iceberg Hashing: Optimizing Many Hash-Table Criteria at Once. Computer Science > Data Structures and Algorithms. arXiv preprint arXiv:2109.04548*