

LAPORAN PRAKTIKUM STRUKTUR DATA DAN ALGORITMA

MODUL V HASH TABLE



Disusun Oleh :

Muhammad Rusdiyanto Asatman
2311102053

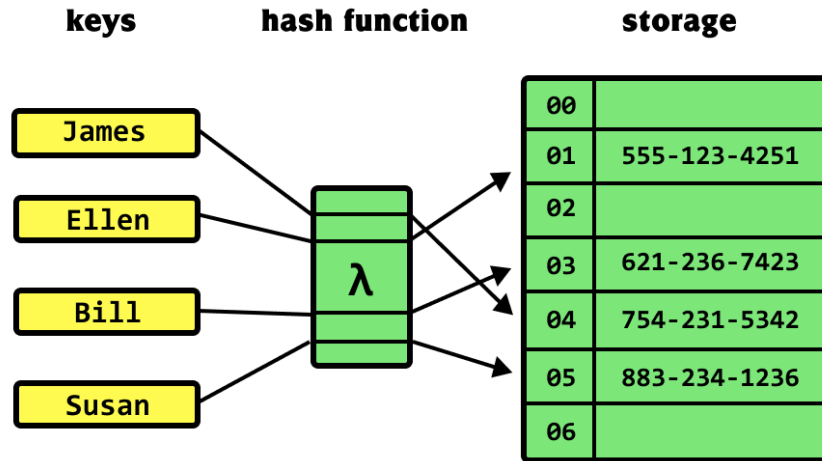
Dosen :

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

**PROGRAM STUDI S1 TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
2024**

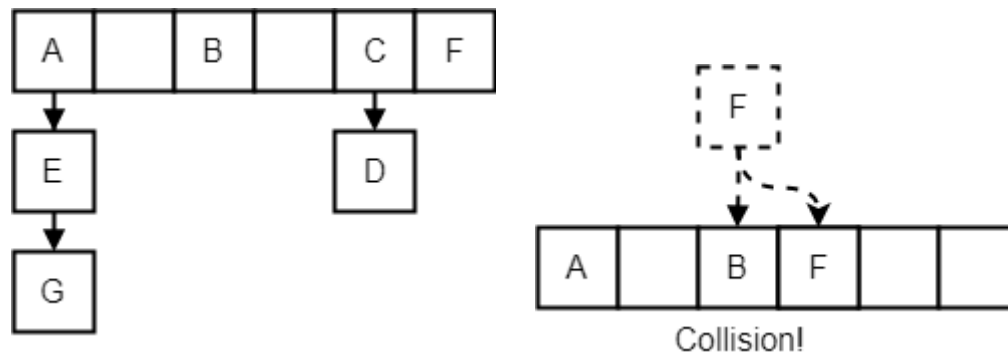
A. Dasar Teori

1. Pengertian



Gambar 1.0 Ilustrasi Hash Table

Hash Table (Tabel Hash) adalah struktur data yang digunakan untuk menyimpan dan mengelola kumpulan data. Setiap elemen dalam kumpulan data memiliki kunci (key) yang unik dan nilainya (value) terkait. Konsep utama dibalik tabel hash adalah penggunaan fungsi hash untuk mengonversi kunci menjadi alamat atau indeks di dalam tabel. Hal tersebut memungkinkan pencarian dan pengambilan data dengan efisien.



Gambar 1.1 & 1.2 Ilustrasi metode closed addressing dan open addressing

Pada dasarnya, tabel hash berfungsi sebagai penyimpanan asosiatif, dimana data disimpan dalam bentuk pasangan kunci-nilai. Proses ini melibatkan penggunaan fungsi hash untuk menghasilkan indeks unik dari kunci, dan nilai yang sesuai akan disimpan pada indeks tersebut. Salah satu keuntungan utama dari tabel hash adalah kecepatan akses dan pencarian data yang sangat efisien. Ketika akan mencari nilai berdasarkan kunci, program hanya perlu melakukan proses hashing pada kunci untuk mengetahui

lokasi penyimpanan datanya, sehingga tidak perlu mencari secara berurutan seperti pada struktur data lainnya seperti array atau list. Namun terkadang bisa saja terjadi bentrokan data (data collision), yaitu ketika dua kunci berbeda menghasilkan indeks yang sama. Untuk menangani tabrakan ini, metode penyelesaian tabrakan (collision resolution) digunakan, seperti linear probing, chaining, atau double hashing.

Tentu saja ada beberapa **kekurangan** yang perlu dipertimbangkan ketika menggunakan tabel hash:

1. **Tabrakan Hash (Hash Collision):** Seperti yang dijelaskan tadi, meskipun metode penyelesaian tabrakan (collision resolution) tersedia digunakan untuk mengatasi masalah ini, tabrakan hash dapat memperlambat akses dan pencarian data.
2. **Penggunaan Memori:** Tabel hash memerlukan alokasi memori yang cukup besar, terutama jika jumlah elemen yang disimpan sangat besar.
3. **Fungsi Hash yang Buruk:** Kinerja tabel hash sangat tergantung pada fungsi hash yang digunakan. Jika fungsi hash tidak efisien atau menghasilkan banyak tabrakan, maka kinerja tabel hash akan menurun.
4. **Tidak Terurut:** Tabel hash tidak mempertahankan urutan data. Jika memerlukan data yang terurut berdasarkan kunci, tabel hash mungkin bukan pilihan yang tepat.
5. **Ketergantungan pada Fungsi Hash yang Unik:** Tabel hash memerlukan fungsi hash yang menghasilkan indeks unik untuk setiap kunci. Jika fungsi hash tidak unik, maka tabrakan hash akan sering terjadi.

Meskipun memiliki kekurangan, tabel hash tetap menjadi struktur data yang sangat berguna dalam banyak aplikasi karena kecepatan akses dan pencarian data yang tinggi.

B. Guided

Guided 1

```
#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;
}

```

Screenshots Output

```

[Running] cd "c:\MyFiles\Visual Studio Projects HERE\strukdat\praktikum\day\" && g++ guided1.cpp -o guided1 && "c:\MyFiles\Visual
Studio Projects HERE\strukdat\praktikum\day\"guided1
Get key 1: 10
Get key 4: -1
1: 10
2: 20
3: 30
[Done] exited with code=0 in 1.099 seconds

```

Deskripsi:

Program di atas adalah program yang mendemonstrasikan penggunaan dari Hash Table. Data hash table disimpan di dalam single linked list dimana setiap node dalam list berisikan key (semacam index), value (data/nilai yang disimpan), dan pointer yang menunjuk ke data selanjutnya. Pointer disini berguna sebagai solusi jikalau ada bentrokan data (data collision) selama program berjalan. Untuk fungsi hash (hash function) itu sendiri tidak terlalu kompleks, karena key didapatkan dengan mendapatkan sisa bagi suatu nilai ket dengan ukuran maksimum hash table, dimana dalam program ini ukuran maksimumnya adalah 10. Program di atas juga memiliki beberapa fungsi untuk mengolah data dalam hash table, yaitu *insert*, *get*, *remove*, *traverse*. Berikut adalah rinciannya :

1. *Insert*, digunakan untuk memasukan data ke dalam hash table. Jika index dari key yang dimasukkan telah terisi oleh data lain, maka program akan mengubah nilai data tersebut menjadi data yang akan dimasukkan (seperti fungsi update).
2. *Get*, digunakan untuk mendapatkan nilai (value) berdasarkan key yang diminta. Jika tidak ada data yang memiliki key yang sama, maka nilai -1 akan dikembalikan, seperti pada screenshot output, dimana saat program ingin mencari value index dari key 4, program mengembalikan nilai -1 karena index dari key 4 masih kosong.

3. *Remove* adalah fungsi yang digunakan untuk menghapus data atau node berdasarkan key.
4. *Traverse* merupakan fungsi yang digunakan untuk menampilkan isi dari hash table. Di dalam fungsi ini juga diterapkan pengecekan data tambahan (dari collision) dalam setiap index. Jika iya (artinya ada lebih dari satu data dalam index itu), maka program akan menampilkan semua data pada index itu terlebih dahulu. Jika tidak, maka program akan lanjut ke index selanjutnya hingga index ke - 10.

Fungsi - fungsi tersebut terletak di dalam class Hash Table. Di dalam class tersebut juga terdapat sebuah *constructor* (fungsi yang berjalan ketika class dipanggil) dan *destructor* (fungsi yang berjalan ketika class dihapus).

Guided 2

```
#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;
}

```

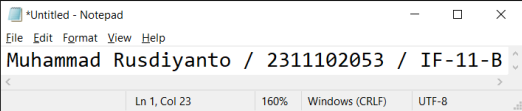


Screenshots Output

```
[Running] cd "c:\MyFiles\Visual Studio Projects HERE\strukdat\praktikum\day\" && g++ guided2.cpp -o guided2 && "c:\MyFiles\Visual Studio Projects HERE\strukdat\praktikum\day\"guided2
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:

[Done] exited with code=0 in 0.608 seconds
```



Deskripsi:

Program di atas adalah program yang mendemonstrasikan penggunaan Hash Table. Perbedaan program ini dengan program sebelumnya terletak pada tempat penyimpanan data table, dimana program ini menggunakan *vector* (sama seperti array, namun dinamis). Data yang disimpan dalam *vector* berbentuk objek dengan nama hash node. Tiap hash node berisikan 2 data berupa nama dan nomor handphone. Untuk hash function dari program ini cenderung lebih kompleks daripada program sebelumnya, proses hash dilakukan dengan menjumlahkan nilai karakter dari key (key disini berupa string), lalu dibagi dengan ukuran hash table dan diambil sisa hasil baginya. Program ini memiliki beberapa fungsi yang terdiri dari :

1. *hashFunc*, berfungsi untuk hashing key dengan proses seperti yang telah dijelaskan sebelumnya.
2. *insert* adalah fungsi yang digunakan untuk memasukkan data baru.
3. *remove*, digunakan untuk menghapus data dengan cara mencari berdasarkan nama.
4. *searchByName* merupakan fungsi yang berguna untuk mencari nomor handphone berdasarkan nama. Untuk proses pencarian sendiri dilakukan dengan algoritma sequential search, jadi program mengecek satu persatu data secara urut dari awal hingga akhir atau hingga data yang sesuai ditemukan. Jika tidak ditemukan, maka program akan mengembalikan string yang kosong.
5. *print* adalah fungsi yang digunakan untuk menampilkan isi dari hash table.

C. Unguided

Unguided 1

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

const int TABLE_SIZE = 10;
string NIM;
int score;

class HashNode {
public:
    string NIM;
    int score;
    HashNode(string NIM, int score) {
        this->NIM = NIM;
        this->score = score;
    }
};

class HashTable {
private:
    vector<HashNode *> table[TABLE_SIZE];
public:
    int hash(string key) {
        int hashVal = 0;
        for (char c : key)
        {
            hashVal += c;
        }
        return hashVal % TABLE_SIZE;
    }

    void insert(string NIM, int score) {
        int hashVal = hash(NIM);
        for (auto node : table[hashVal]) {
            if (node->NIM == NIM) {
                node->NIM = NIM;
                return;
            }
        }
    }
};
```

```

    }
    table[hashVal].push_back(new HashNode(NIM, score));
}

void remove(string NIM) {
    int hashVal = hash(NIM);
    for (auto it = table[hashVal].begin(); it !=
table[hashVal].end(); it++) {
        if ((*it)->NIM == NIM) {
            table[hashVal].erase(it);
            return;
        }
    }
}

int searchByNIM(string NIM) {
    int hashVal = hash(NIM);
    for (auto node : table[hashVal])
    {
        if (node->NIM == NIM)
        {
            return node->score;
        }
    }
    return -1;
}

void searchByScore(int scores[2]) {
    if (scores[0] > scores[1]) {
        int temp = scores[0];
        scores[0] = scores[1];
        scores[1] = temp;
    }
    for (int i = 0; i < TABLE_SIZE; i++)
    {
        cout << i << ": ";
        for (auto pair : table[i])
        {
            if (pair != nullptr && pair->score >= scores[0] &&
pair->score <= scores[1])
            {
                cout << "[" << pair->NIM << ", " <<
pair->score << "];"

```

```

        }
    }
    cout << endl;
}

void print()
{
    for (int i = 0; i < TABLE_SIZE; i++)
    {
        cout << i + 1 << ": ";
        for (auto pair : table[i])
        {
            if (pair != nullptr)
            {
                cout << "[" << pair->NIM << ", " <<
pair->score << "];"
            }
        }
        cout << endl;
    }
};

string inNIM() {
    string NIM;
    cout << "Masukkan NIM mahasiswa -> ";
    cin >> NIM;
    return NIM;
}

int inScore(int useBoundary = 0) {
    int score;
    if (useBoundary <= 0) {
        cout << "Masukkan nilai mahasiswa -> ";
    }
    else {
        cout << "Masukkan batas ke-" << useBoundary << " -> ";
    }
    cin >> score;
    return score;
}

```

```

string line = "===== ";

int main() {
    HashTable hashTable;
    int choice;
    bool run = true;

    while(run) {
        cout << line << "[ Data Mahasiswa ]" << line << endl;
        hashTable.print();
        cout << line << "[ Menu ]" << line << endl;
        cout << "1. Tambah mahasiswa\n";
        cout << "2. Hapus mahasiswa\n";
        cout << "3. Cari mahasiswa dengan NIM\n";
        cout << "4. Cari mahasiswa dengan nilai\n";
        cout << "0. Keluar";
        cout << "\nMasukkan pilihan -> ";
        cin >> choice;

        switch(choice) {
            case 0: return 0; break;
            case 1: {
                string NIM = inNIM();
                int score = inScore();
                hashTable.insert(NIM, score);
                break;
            }
            case 2: {
                string NIM = inNIM();
                hashTable.remove(NIM);
                break;
            }
            case 3: {
                string NIM = inNIM();
                int score = hashTable.searchByNIM(NIM);
                cout << "\nMahasiswa dengan NIM " << NIM << "
memiliki nilai " << score << endl;
                break;
            }
            case 4: {
                int scores[] = {inScore(1), inScore(2)};
                hashTable.searchByScore(scores);
                break;
            }
        }
    }
}

```

```

    }
    default: cout << "Mohon masukkan pilihan yang
benar\n";
    }
}
return 0;
}

```

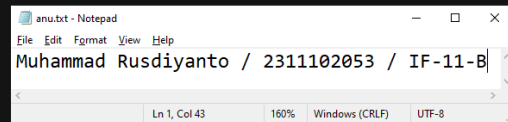
Screenshots Output

- Menu

```

===== [ Data Mahasiswa ] =====
1:
2:
3:
4:
5:
6:
7:
8:
9:
10:
===== [ Menu ] =====
1. Tambah mahasiswa
2. Hapus mahasiswa
3. Cari mahasiswa dengan NIM
4. Cari mahasiswa dengan nilai
0. Keluar

```

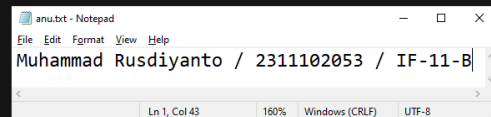


- Tambah Data

```

Masukkan pilihan -> 1
Masukkan NIM mahasiswa -> 2311102001
Masukkan nilai mahasiswa -> 91
===== [ Data Mahasiswa ] =====
1:
2: [2311102001, 91]
3:
4:
5:
6:
7:
8:
9:
10:

```

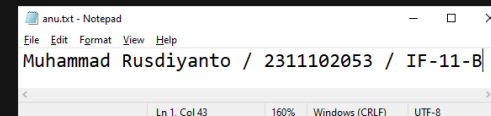


- Hapus Data

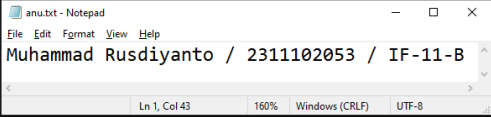
```

===== [ Data Mahasiswa ] =====
1:
2: [2311102001, 91][2311102047, 85]
3: [2311102020, 87][2311102101, 80]
4:
5:
6: [2311102023, 78]
7:
8:
9:
10:

```

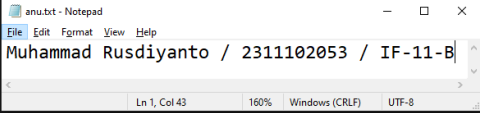



```
===== [ Menu ] =====
1. Tambah mahasiswa
2. Hapus mahasiswa
3. Cari mahasiswa dengan NIM
4. Cari mahasiswa dengan nilai
0. Keluar
Masukkan pilihan -> 2
Masukkan NIM mahasiswa -> 2311102023
===== [ Data Mahasiswa ] =====
1:
2: [2311102001, 91][2311102047, 85]
3: [2311102020, 87][2311102101, 80]
4:
5:
6:
7:
8:
9:
10:
```



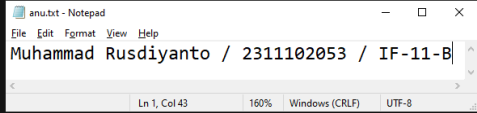
- Tampil berdasarkan NIM

```
===== [ Data Mahasiswa ] =====
1:
2: [2311102001, 91][2311102047, 85]
3: [2311102020, 87][2311102101, 80]
4:
5:
6:
7:
8:
9:
10:
===== [ Menu ] =====
1. Tambah mahasiswa
2. Hapus mahasiswa
3. Cari mahasiswa dengan NIM
4. Cari mahasiswa dengan nilai
0. Keluar
Masukkan pilihan -> 3
Masukkan NIM mahasiswa -> 2311102101
Mahasiswa dengan NIM 2311102101 memiliki nilai 80
```

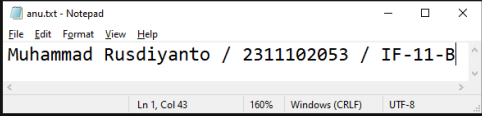


- Tampil berdasarkan nilai

```
===== [ Data Mahasiswa ] =====
1: [126735167, 92]
2: [2311102001, 91][2311102047, 85]
3: [2311102020, 87][2311102101, 80]
4:
5:
6: [2311102023, 79]
7: [176531821, 100]
8:
9:
10:
```



```
===== [ Menu ] =====
1. Tambah mahasiswa
2. Hapus mahasiswa
3. Cari mahasiswa dengan NIM
4. Cari mahasiswa dengan nilai
0. Keluar
Masukkan pilihan -> 4
Masukkan batas ke-1 -> 80
Masukkan batas ke-2 -> 90
0:
1: [2311102047, 85]
2: [2311102020, 87][2311102101, 80]
3:
4:
5:
6:
7:
8:
9:
===== [ Data Mahasiswa ] =====
1: [126735167, 92]
2: [2311102001, 91][2311102047, 85]
3: [2311102020, 87][2311102101, 80]
4:
5:
6: [2311102023, 79]
7: [176531821, 100]
8:
9:
10:
```



Deskripsi:

Program di atas adalah sebuah program pendataan nilai mahasiswa berdasarkan NIM yang merupakan hasil modifikasi dari program Guided 2. Layaknya program Guided 2, program ini menggunakan vector dan hash function yang sama, hanya saja program ini menyimpan dua data yaitu NIM (string) serta nilai (int) dari mahasiswa. Cara kerja kode kurang lebih sama untuk mayoritas dari program. Ada pula modifikasi lain berupa menu dan input pengguna dalam program. Untuk input pengguna dibuat menggunakan fungsi supaya bisa digunakan berulang kali, sehingga menghemat kode. Ada juga fungsi tambahan yaitu *searchByScore* yang berfungsi untuk mencari dan menampilkan mahasiswa - mahasiswa dalam rentang nilai tertentu. Hal ini dilakukan dengan cara meminta input pengguna berupa batas 1 dan batas 2. Nantinya kedua batas tersebut akan dibandingkan mana yang lebih kecil. Hal ini dilakukan supaya tidak terjadi kekeliruan dalam output. Kedua batas tersebut akan disimpan dalam satu array, dimana indeks 0 untuk batas awal (minimum) dan indeks 1 untuk batas akhir (maksimum). Setelah itu, program baru akan mengecek satu per satu data sesuai dengan kriteria nilai yang telah diberikan dan menampilkannya.

D. Kesimpulan

Hash Table merupakan struktur data yang digunakan untuk menyimpan data dengan menggunakan key dan fungsi hash. Hash Table sangat baik dalam mencari, menghapus, dan menambahkan data, Namun dibalik itu, bentrokan data (data collision) adalah masalah yang mungkin terjadi ketika menerapkan struktur data yang satu ini. Hal ini dapat dicegah dengan menerapkan beberapa metode seperti closed addressing dan open addressing.

E. Referensi

Asisten Praktikum, "Modul 5 Hash Table"

Annisa. (2023). "Struktur Data Hash Table: Pengertian, Cara Kerja, dan Operasi Hash Table." Diakses pada 9 Mei 2024, dari <https://fikti.umsu.ac.id/struktur-data-hash-table-pengertian-cara-kerja-dan-operasi-hash-table/>.

Rahmawati, Rini. "Hash Table : Pengertian, Fungsi, dan Cara Membuat." Diakses pada 9 Mei 2024, dari <https://dosenit.com/kuliah-it/hash-table>.

(2019). "Hash Tables." Diakses pada 9 Mei 2024, dari <https://www.data-structures-in-practice.com/hash-tables/>