LAPORAN PRAKTIKUM

MODUL V
HASH TABLE

Disusun oleh:
Geranada Saputra Priambudi
NIM: 2311102008

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

a. Mahasiswa mampu menjelaskan definisi dan konsep dari Hash Code.

b. Mahasiswa mampu menerapkan Hash Code kedalam pemrograman.

BAB II

DASAR TEORI

Hash Table adalah struktur data yang digunakan untuk menyimpan dan mengelola Kumpulan data, Dimana setiap elemen dalam Kumpulan data memiliki kunci (key) yang unik nilainya (value) terkait. Konsep utama di balik Hash Table adalah penggunaan fungsi hash untuk mengonversi kunci menjadi Alamat atau indeks di dalam tabel, sehingga memungkinkan pencarian dan pengambilan data dengan efisien

Fungsi utama Hash Table adalah untuk menyediakan struktur data yang memungkinkan pencarian , penyisipan dan penghapusan data dengan efisien. Berikut adalah beberapa fungsi utama dari Hash Table :

1. Pencarian (Search)
Hash Table memungkinkan pencarian data berdasarkan kunci (key) dengan cepat. Saat kita ingin mencari nilai berdasarkan kunci, fungsi hash akan mengonversi kunci menjadi indeks, dan nilai yang sesuai dapat diakses langsung dari indeks tersebut, menghasilkan waktu akses yang konstan (O(1)).

2. Penyisipan (Insertion)
Hash Table memungkinkan penambahan data baru dengan kunci dan nilainya. Saat kita ingin menyisipkan data baru, fungsi hash akan mengonversi kunci menjadi indeks, dan nilai tersebut akan disimpan pada indeks yang sesuai. Proses ini dapat dilakukan dengan waktu yang konstan (O(1)) pada kebanyakan kasus, membuatnya sangat efisien.

3. Penghapusan (Deletion)
Hash Table memungkinkan penghapusan data berdasarkan kunci. Saat kita ingin menghapus data, hash table akan menggunakan fungsi hash untuk menemukan indeks data yang sesuai dengan kunci dan menghapusnya dari tabel. Seperti pencarian dan penyisipan, operasi penghapusan juga berjalan dengan waktu yang konstan (O(1)) dalam kebanyakan kasus.

4. Asosiasi Kunci-Nilai (Key-Value Association)
Hash Table menyimpan data dalam bentuk pasangan kunci-nilai (key-value). Ini memungkinkan kita untuk mengaitkan kunci dengan nilai tertentu sehingga kita dapat dengan mudah mengakses nilai tersebut ketika diberikan kunci.

5. Kecepatan Akses
Salah satu keunggulan utama dari Hash Table adalah kecepatan aksesnya. Dengan menggunakan fungsi hash, proses mencari dan mengakses data menjadi sangat cepat karena kita dapat langsung menuju lokasi data tanpa perlu mencari secara berurutan.

BAB III

GUIDED

1. 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:
```

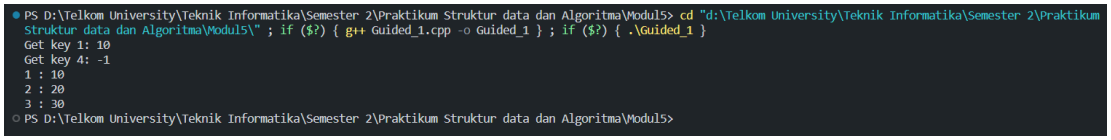
```
    HashTable()
    {
        table = new Node *[MAX_SIZE]();
    }
    ~HashTable()
    {
        for (int i = 0; i < MAX_SIZE; i++)
        {
            Node *current = table[i];
            while (current != nullptr)
            {
                Node *temp = current;
                current = 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



Deskripsi program

Program diatas menyediakan implementasi dasar dari Hash Table dengan Teknik chaining untuk menangani collison. Ini mencakup operasi dasar seperti penyisipan (‘insert’), pencarian (‘get’), penghapusan (‘remove’) dan penelusuran (‘traverse’). Namun, perlu diperhatikan bahwa program ini masih sederhana dasn dapat ditingkatkan untuk menangani kasus khusus, seperti menangani collison dengan Teknik open addressing, menambahkan mekanisme resize hash table, atau menambahkan fungsi lainnya sesuai kebutuhan aplikasi.

2. Guided 2

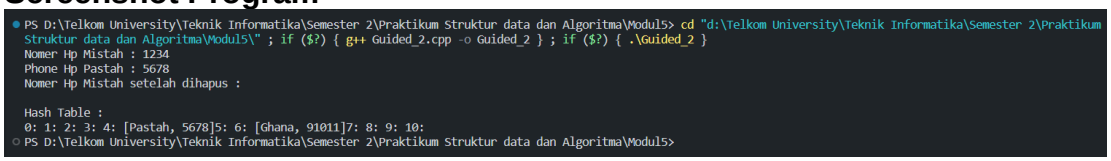
Source Code

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

using namespace std;

// ukuran tabel hash
const int TABLE_SIZE = 11;

string name; //deklarasi variabel string name
string phone_number; //deklarasi variabel string phone_number

// Struktur Data Untuk Setiap Node
class HashNode
{
//deklarasi variabel name dan phone_number
public:
    string name;
    string phone_number;

    HashNode(string name, string phone_number)
    {
        this->name = name;
        this->phone_number = phone_number;
    }
};

// Class HashMap
class HashMap
{
private:
    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;
    }

    // Tambah data
    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));
    }

    // Hapus data
    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;
            }
        }
    }

    // Cari data berdasarkan nama
    string searchByName(string name)
    {
        int hash_val = hashFunc(name);
        for (auto node : table[hash_val])
        {
            if (node->name == name)
            {
                return node->phone_number;
            }
        }
        return "";
    }

    // Cetak data
    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 << "]";
                }
            }
        }
    }
};

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;
}
```

Screenshot Program



Deskripsi program

Program ini menyediakan implementasi sederhana dari hash map dengan teknik chaining untuk menyimpan dan mengelola data pegawai berdasarkan nama dan nomor telepon. Operasi dasar seperti penyisipan data baru, pencarian berdasarkan nama, dan penghapusan data berdasarkan nama dapat dilakukan dengan menggunakan hash map ini. Namun, perlu diingat bahwa program ini hanya mencakup fungsi dasar dan dapat ditingkatkan atau disesuaikan lebih lanjut sesuai dengan kebutuhan aplikasi yang lebih kompleks.

LATIHAN KELAS - UNGUIDED

1. Unguided 1

Source code

```
#include <iostream>
#include <list>

using namespace std;

// Struktur data untuk mahasiswa
struct Mahasiswa {
    int nim;
    int nilai;
};

// Ukuran tabel hash
const int TABLE_SIZE = 10;

// Class HashTable untuk tabel hash
class HashTable {
private:
    list<Mahasiswa> table[TABLE_SIZE]; // Array dari list untuk
chaining

    // Fungsi hash sederhana
    int hashFunction(int nim) {
        return nim % TABLE_SIZE;
    }

public:
    // Menambahkan data mahasiswa ke tabel hash
    void insertData(int nim, int nilai) {
        int index = hashFunction(nim);
```

```
        Mahasiswa mhs = {nim, nilai};
        table[index].push_back(mhs);
    }

    // Menghapus data mahasiswa dari tabel hash berdasarkan NIM
    void deleteData(int nim) {
        int index = hashFunction(nim);
        for (auto it = table[index].begin(); it != table[index].end(); ++it) {
            if (it->nim == nim) {
                table[index].erase(it);
                cout << "Data mahasiswa dengan NIM " << nim << " telah dihapus.\n";
                return;
            }
        }
        cout << "Data mahasiswa dengan NIM " << nim << " tidak ditemukan.\n";
    }

    // Mencari data mahasiswa berdasarkan NIM
    void searchDataByNIM(int nim) {
        int index = hashFunction(nim);
        for (const auto& mhs : table[index]) {
            if (mhs.nim == nim) {
                cout << "Data mahasiswa dengan NIM " << nim << " ditemukan. Nilai: " << mhs.nilai << "\n";
                return;
            }
        }
        cout << "Data mahasiswa dengan NIM " << nim << " tidak ditemukan.\n";
    }
```

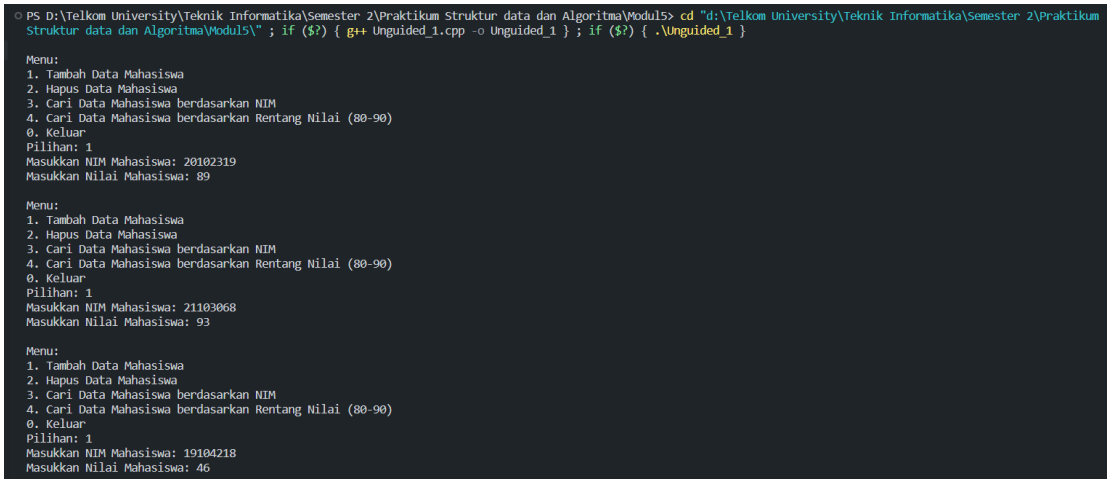
```
    // Mencari dan menampilkan data mahasiswa berdasarkan rentang
nilai (80-90)
    void searchByScoreRange(int minScore, int maxScore) {
        for (int i = 0; i < TABLE_SIZE; ++i) {
            for (const auto& mhs : table[i]) {
                if (mhs.nilai >= minScore && mhs.nilai <=
maxScore) {
                    cout << "NIM: " << mhs.nim << ", Nilai: " <<
mhs.nilai << "\n";
                }
            }
        }
    }
};

// Fungsi main untuk penggunaan hash table
int main() {
    HashTable hashTable;
    int choice;

    do {
        cout << "\nMenu:\n";
        cout << "1. Tambah Data Mahasiswa\n";
        cout << "2. Hapus Data Mahasiswa\n";
        cout << "3. Cari Data Mahasiswa berdasarkan NIM\n";
        cout << "4. Cari Data Mahasiswa berdasarkan Rentang Nilai
(80-90)\n";
        cout << "0. Keluar\n";
        cout << "Pilihan: ";
        cin >> choice;

        switch (choice) {
            case 1: {
                int nim, nilai;
```

```
                cout << "Masukkan NIM Mahasiswa: ";
                cin >> nim;
                cout << "Masukkan Nilai Mahasiswa: ";
                cin >> nilai;
                hashTable.insertData(nim, nilai);
                break;
            }
            case 2: {
                int nim;
                cout << "Masukkan NIM Mahasiswa yang akan
dihapus: ";
                cin >> nim;
                hashTable.deleteData(nim);
                break;
            }
            case 3: {
                int nim;
                cout << "Masukkan NIM Mahasiswa yang ingin
dicari: ";
                cin >> nim;
                hashTable.searchDataByNIM(nim);
                break;
            }
            case 4: {
                cout << "Mahasiswa dengan nilai antara 80 dan
90:\n";
                hashTable.searchByScoreRange(80, 90);
                break;
            }
            case 0:
                cout << "Terima kasih!\n";
                break;
            default:
```

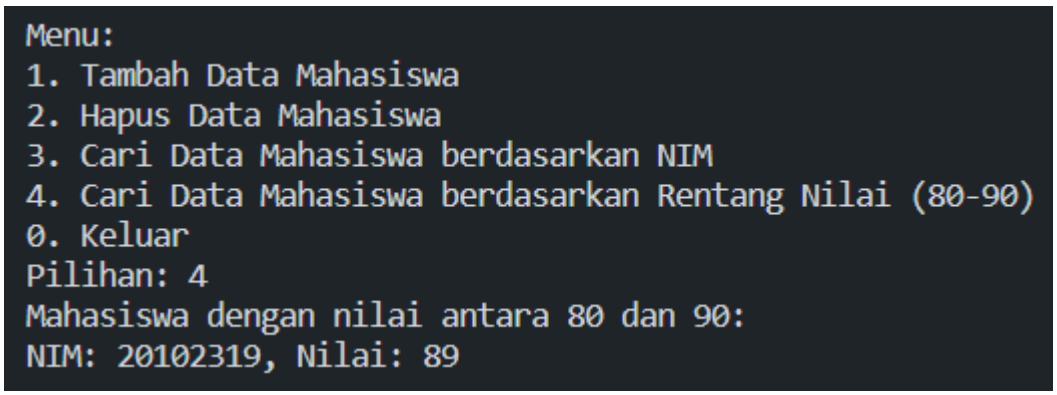
```
                cout << "Pilihan tidak valid. Silakan coba
lagi.\n";
            }
        } while (choice != 0);

        return 0;
}
```

Screenshoot program

Menu tambah data mahasiswa :



Menu cari data mahasiswa berdasarkan rentang nilai (80-90) :



Menu cari data mahasiswa berdasarkan NIM :

```
Menu:
1. Tambah Data Mahasiswa
2. Hapus Data Mahasiswa
3. Cari Data Mahasiswa berdasarkan NIM
4. Cari Data Mahasiswa berdasarkan Rentang Nilai (80-90)
0. Keluar
Pilihan: 3
Masukkan NIM Mahasiswa yang ingin dicari: 19104218
Data mahasiswa dengan NIM 19104218 ditemukan. Nilai: 46

Menu:
1. Tambah Data Mahasiswa
2. Hapus Data Mahasiswa
3. Cari Data Mahasiswa berdasarkan NIM
4. Cari Data Mahasiswa berdasarkan Rentang Nilai (80-90)
0. Keluar
Pilihan: 3
Masukkan NIM Mahasiswa yang ingin dicari: 21103068
Data mahasiswa dengan NIM 21103068 ditemukan. Nilai: 93
```

Menu keluar :

```
Menu:
1. Tambah Data Mahasiswa
2. Hapus Data Mahasiswa
3. Cari Data Mahasiswa berdasarkan NIM
4. Cari Data Mahasiswa berdasarkan Rentang Nilai (80-90)
0. Keluar
Pilihan: 0
Terima kasih!
PS D:\Telkom University\Teknik Informatika\Semester 2\Praktikum Struktur data dan Algoritma\Modul5>
```

Deskripsi program

Program ini menyediakan implementasi dasar dari hash table untuk manajemen data mahasiswa dengan menggunakan teknik chaining. Fitur utama yang disediakan antara lain: penambahan data mahasiswa, penghapusan data mahasiswa berdasarkan NIM, pencarian data mahasiswa berdasarkan NIM, dan pencarian data mahasiswa berdasarkan rentang nilai. Meskipun program ini sederhana, ini memberikan dasar yang baik untuk memahami konsep dasar penggunaan hash table dalam pemrograman.

BAB IV

KESIMPULAN

Hash table adalah struktur data yang digunakan untuk menyimpan elemen-elemen dengan kunci unik dan nilai terkait. Hash table menggunakan fungsi hash untuk mengonversi kunci menjadi indeks dalam array, sehingga memungkinkan akses cepat ke data. Salah satu tantangan dalam hash table adalah penanganan collision, yaitu ketika dua kunci berbeda menghasilkan indeks yang sama setelah di-hash. Teknik penanganan collision yang umum digunakan adalah chaining (menggunakan linked list di setiap slot hash) atau open addressing (menggunakan strategi probe untuk mencari slot kosong).

Implementasi hash table melibatkan beberapa komponen:

- Fungsi hash: Fungsi yang mengonversi kunci menjadi indeks dalam array.

- Array atau struktur data lain untuk menyimpan elemen-elemen, seperti linked list untuk chaining.

- Operasi dasar: Penyisipan data (insert), pencarian data (get/search), penghapusan data (remove), dan lainnya sesuai kebutuhan aplikasi.

Keuntungan Hash Table yaitu Akses cepat Dengan menggunakan fungsi hash yang baik, pencarian, penyisipan, dan penghapusan data dapat dilakukan dalam waktu konstan (O(1)), Efisien untuk jumlah data besar Meskipun kompleksitas waktu terburuknya adalah O(n) dalam kasus terburuk (semua kunci hash ke slot yang sama), hash table tetap efisien untuk jumlah data yang besar. Penerapan dan Kasus Penggunaan, Hash table banyak digunakan dalam implementasi basis data, kamus (dictionary), caching, dan lainnya. Contoh penerapan hash table dalam kehidupan sehari-hari termasuk penyimpanan data pengguna pada aplikasi web, pengindeksan data pada sistem pencarian, dan lain sebagainya. Optimalisasi Hash Table: Pemilihan ukuran tabel (TABLE_SIZE) dan fungsi hash yang efektif dapat mempengaruhi kinerja hash table. Terkadang, hash table perlu diresize atau teknik-teknik lain diterapkan untuk menjaga kinerja saat data bertambah. Laporan praktikum tentang hash table memberikan wawasan yang baik tentang konsep dasar, implementasi, dan penerapan hash table dalam pengembangan perangkat lunak. Hal ini penting untuk memahami bagaimana struktur data ini dapat digunakan untuk memecahkan berbagai masalah pemrograman dan perangkat lunak.