

**LAPORAN PRAKTIKUM STRUKTUR
DATA DAN ALGORITMA**

**MODUL IV
LINKED LIST CIRCULAR DAN NON CIRCULAR**



Disusun Oleh :

NAMA : Aulia Radix Putra Winarko

NIM : 2311102056

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 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.

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.

2. 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.

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);
}

```

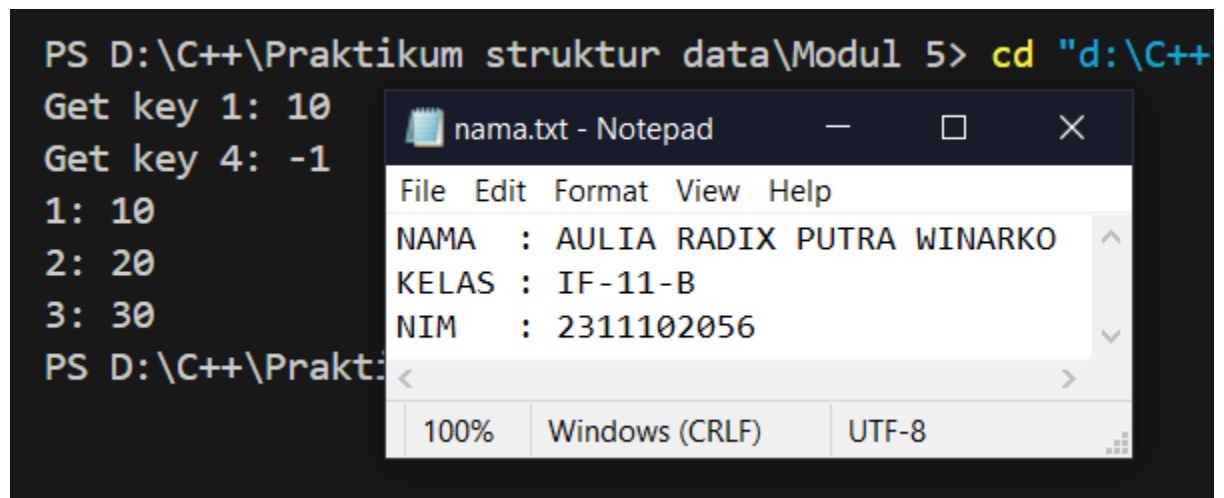
```

    // Traversal
    ht.traverse();

    return 0;
}

```

Screenshots Output



The screenshot shows a terminal window with the following output:

```

PS D:\C++\Praktikum struktur data\Modul 5> cd "d:\C++
Get key 1: 10
Get key 4: -1
1: 10
2: 20
3: 30
PS D:\C++\Prakt:

```

Overlaid on the terminal is a Notepad window titled "nama.txt - Notepad". The text inside the Notepad window is:

```

File Edit Format View Help
NAMA : AULIA RADIX PUTRA WINARKO
KELAS : IF-11-B
NIM : 2311102056

```

The Notepad window also shows a status bar at the bottom with "100%", "Windows (CRLF)", and "UTF-8".

Deskripsi:

Kode di atas mendemonstrasikan implementasi hash table menggunakan teknik chaining untuk mengatasi kolisi, dengan operasi dasar seperti penambahan (insert), pencarian (get), penghapusan (remove), dan penelusuran (traverse).

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 << " ]";
            }
        }
        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

```

Nomer Hp Mistah : 1234
Phone Hp Pastah : 5678
Nomer Hp Mistah setelah dihapus :

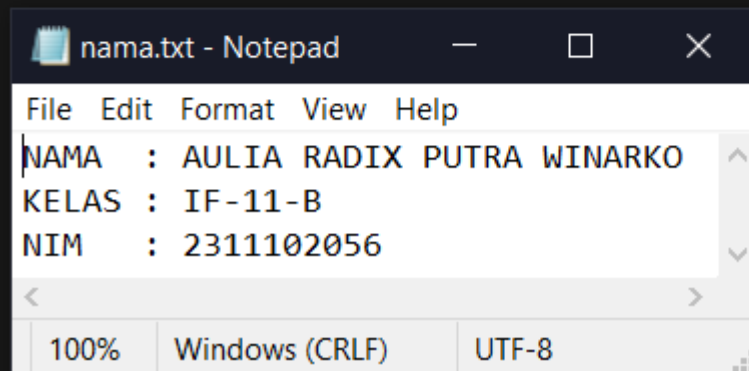
```

Hash Table :

```

0:
1:
2:
3:
4: [Pastah, 5678]
5:
6: [Ghana, 91011]
7:
8:
9:
10:

```



Modul 5>

Deskripsi:

Kode ini mengimplementasikan hash table menggunakan chaining dengan vektor untuk menangani kolisi, serta menyediakan fungsi dasar seperti penambahan (insert), penghapusan (remove), pencarian (searchByName), dan pencetakan isi hash table (print).

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

Unguided 1

```
#include <iostream>
#include <string>
#include <vector>
#include <iomanip>
using namespace std;
class Mahasiswa
{
public:
    string nama;
    long long int NIM;
    int nilai;
    Mahasiswa(string nama, long long int NIM, int nilai) : nama(nama),
                                                            NIM(NIM),
                                                            nilai(nilai) {}
};
class HashNode
{
public:
    vector<Mahasiswa> mahasiswa;
    void insert(Mahasiswa mahasiswa)
    {
        mahasiswa.push_back(mahasiswa);
    }
    void remove(long long int NIM)
    {
        for (auto it = mahasiswa.begin(); it != mahasiswa.end(); ++it)
        {
            if (it->NIM == NIM)
            {
                mahasiswa.erase(it);
                return;
            }
        }
    }
}
```

```

    }
    Mahasiswa *pencarianmhs(long long int NIM)
    {
        for (auto &mahasiswa : mahasiswa)
        {
            if (mahasiswa.NIM == NIM)
            {
                return &mahasiswa;
            }
        }
        return nullptr;
    }
    vector<Mahasiswa *> rentanNilai(int minNilai, int maxNilai)
    {
        vector<Mahasiswa *> result;
        for (auto &mahasiswa : mahasiswa)
        {
            if (mahasiswa.nilai >= minNilai && mahasiswa.nilai <=
maxNilai)
            {
                result.push_back(&mahasiswa);
            }
        }
        return result;
    }
};

class HashMap
{
private:
    static const int TABLE_SIZE = 20;
    HashNode *table[TABLE_SIZE];

public:
    HashMap()
    {
        for (int i = 0; i < TABLE_SIZE; ++i)
        {
            table[i] = new HashNode();
        }
    }
    int hashFunc(long long int NIM)
    {
        return NIM % TABLE_SIZE;
    }
}

```

```

void insert(Mahasiswa mahasiswa)
{
    int hash_val = hashFunc(mahasiswa.NIM);
    table[hash_val]->insert(mahasiswa);
}
void remove(long long int NIM)
{
    int hash_val = hashFunc(NIM);
    table[hash_val]->remove(NIM);
}
Mahasiswa *pencarianmhs(long long int NIM)
{
    int hash_val = hashFunc(NIM);
    return table[hash_val]->pencarianmhs(NIM);
}
vector<Mahasiswa *> rentanNilai(int minNilai, int maxNilai)
{
    vector<Mahasiswa *> result;
    for (int i = 0; i < TABLE_SIZE; ++i)
    {
        vector<Mahasiswa *> temp = table[i]->rentanNilai(minNilai,
                                                         maxNilai);
        result.insert(result.end(), temp.begin(), temp.end());
    }
    return result;
}
void display()
{
    cout << " Data Mahasiswa " << endl;
    for (int i = 0; i < TABLE_SIZE; ++i)
    {
        if (!table[i]->mahasiswas.empty())
        {
            cout << "Slot " << setw(3) << i << " | ";
            for (auto &mahasiswa : table[i]->mahasiswas)
            {
                cout << "Nama: " << setw(15) << mahasiswa.nama << " | "
                << "NIM: " << setw(10) << mahasiswa.NIM << " | "
                << "Nilai: " << setw(3) << mahasiswa.nilai << "
                | ";
            }
            cout << endl;
        }
    }
}

```

```

    }
}
};
int main()
{
    HashMap hashTable;
    int pilih;
    while (true)
    {
        cout << "\nMenu Pilihan \n\n";
        cout << "1. Tambah Data Mahasiswa \n";
        cout << "2. Menghapus Data Mahasiswa \n";
        cout << "3. Mencari Mahasiswa \n";
        cout << "4. Mencari Rentan Nilai \n";
        cout << "5. Display \n";
        cout << "0. Exit \n";
        cout << "Pilih: ";
        cin >> pilih;
        switch (pilih)
        {
            case 1:
            {
                string nama;
                long long int NIM;
                int nilai;
                cout << "Masukkan Nama Mahasiswa: ";
                cin.ignore();
                getline(cin, nama);
                cout << "Masukkan NIM Mahasiswa: ";
                cin >> NIM;
                cout << "Masukkan Nilai Mahasiswa: ";
                cin >> nilai;
                Mahasiswa mahasiswa(nama, NIM, nilai);
                hashTable.insert(mahasiswa);
                cout << "Mahasiswa telah ditambahkan! " << endl;
                break;
            }
            case 2:
            {
                long long int NIM;
                cout << "Masukan NIM mahasiswa yang ingin dihapus: ";
                cin >> NIM;
                hashTable.remove(NIM);
                cout << "Data Mahasiswa Telah Dihapus! " << endl;
            }
        }
    }
}

```

```

        break;
    }
    case 3:
    {
        long long int NIM;
        cout << "Masukkan NIM Mahasiswa: ";
        cin >> NIM;
        Mahasiswa *mahasiswa = hashTable.pencarianmhs(NIM);
        if (mahasiswa != nullptr)
        {
            cout << "Data Mahasiswa:" << endl;
            cout << "Nama: " << mahasiswa->nama << endl;
            cout << "NIM: " << mahasiswa->NIM << endl;
            cout << "Nilai: " << mahasiswa->nilai << endl;
        }
        else
        {
            cout << "Mahasiswa tidak ditemukan." << endl;
        }
        break;
    }
    case 4:
    {
        int minNilai, maxNilai;
        cout << "Masukkan rentang nilai min: ";
        cin >> minNilai;
        cout << "Masukan rentang nilai max: ";
        cin >> maxNilai;
        vector<Mahasiswa *> mahasiswaas =
hashTable.rentanNilai(minNilai,
                                                                maxNil
ai);
        if (mahasiswaas.empty())
        {
            cout << "Tidak ada mahasiswa dengan nilai dalam rentan
tersebut." << endl;
        }
        else
        {
            cout << "Mahasiswa dengan nilai dalam rentang " <<
minNilai << " - " << maxNilai << " adalah:" << endl;
            for (auto mahasiswa : mahasiswaas)
            {

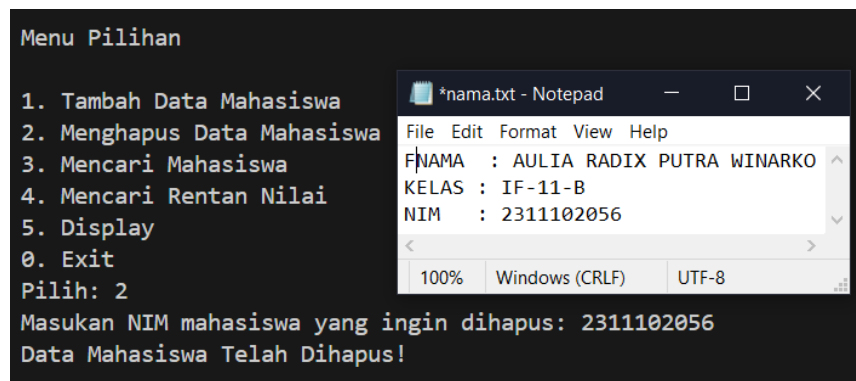
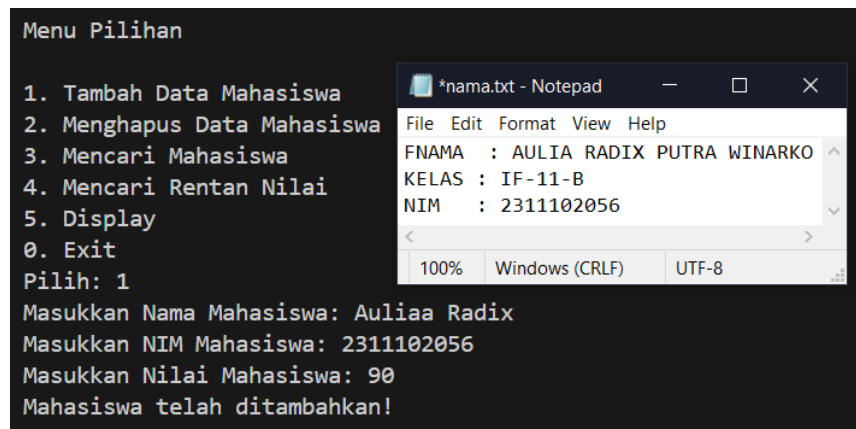
```

```

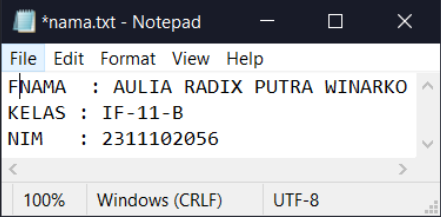
        cout << "Nama: " << mahasiswa->nama << ", NIM: " <<
mahasiswa->NIM << ", Nilai: " << mahasiswa->nilai << endl;
    }
}
break;
}
case 5:
    hashTable.display();
    break;
case 0:
    return 0;
default:
    cout << "Pilihan tidak valid." << endl;
    break;
}
}
return 0;
}

```

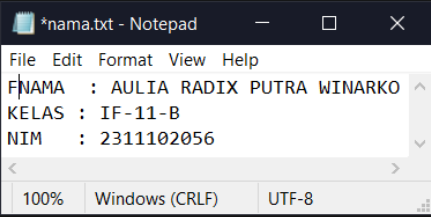
Screenshots Output



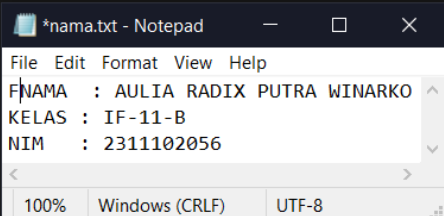
```
Menu Pilihan
1. Tambah Data Mahasiswa
2. Menghapus Data Mahasiswa
3. Mencari Mahasiswa
4. Mencari Rentan Nilai
5. Display
0. Exit
Pilih: 3
Masukkan NIM Mahasiswa: 911
Data Mahasiswa:
Nama: Sigit
NIM: 911
Nilai: 86
```



```
Menu Pilihan
1. Tambah Data Mahasiswa
2. Menghapus Data Mahasiswa
3. Mencari Mahasiswa
4. Mencari Rentan Nilai
5. Display
0. Exit
Pilih: 4
Masukkan rentang nilai min: 80
Masukan rentang nilai max: 90
Mahasiswa dengan nilai dalam rentang 80 - 90 adalah:
Nama: Sigit, NIM: 911, Nilai: 86
Nama: Roesdi, NIM: 119, Nilai: 89
```



```
Menu Pilihan
1. Tambah Data Mahasiswa
2. Menghapus Data Mahasiswa
3. Mencari Mahasiswa
4. Mencari Rentan Nilai
5. Display
0. Exit
Pilih: 5
Data Mahasiswa
Slot 11 | Nama: Sigit | NIM: 911 | Nilai: 86 |
Slot 19 | Nama: Roesdi | NIM: 119 | Nilai: 89 |
```



Deskripsi:

Kode ini menyediakan implementasi hash table yang efisien untuk menyimpan dan mengelola data mahasiswa. Dengan menggunakan metode hashing, data dapat disimpan, dihapus, dan dicari dengan cepat berdasarkan NIM. Selain itu, metode pencarian berdasarkan rentang nilai dan tampilan semua data juga disediakan untuk kemudahan penggunaan.

D. Kesimpulan

Hash Table adalah struktur data yang dirancang menggunakan fungsi khusus yang disebut fungsi hash. Fungsi hash digunakan untuk memetakan nilai yang diberikan dengan kunci tertentu untuk akses elemen yang lebih cepat. Efisiensi pemetaan tergantung pada efisiensi fungsi hash yang digunakan. Hash table menggunakan memori penyimpanan utama berbentuk array dengan tambahan algoritma untuk mempercepat pemrosesan data. Pada intinya hash table merupakan cara penyimpanan data menggunakan key value yang didapat dari nilai data itu sendiri. Dengan key value tersebut didapat hash value. Jadi hash function merupakan suatu fungsi sederhana untuk mendapatkan hash value dari key value suatu data.

Referensi

Asisten Praktikum. "Modul 5 Hash Tabel". Learning Management System. 2024