LAPORAN PRAKTIKUM STRUKTUR DATA DAN ALGORITMA MODUL 5

“HASH TABLE”

DISUSUN OLEH :

Mohammad Harits Tantowi

2311102016

IF-11-A

DOSEN:

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

PROGRAM STUDI S1 INFORMATIKA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
PURWOKERTO

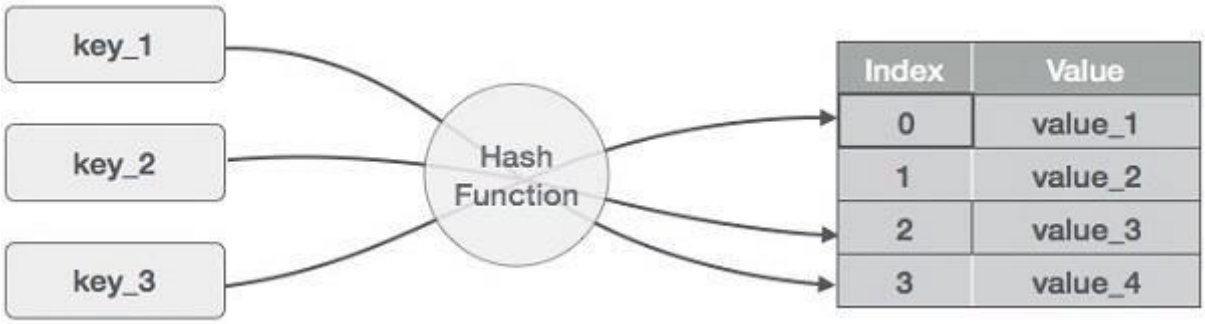
2024

A. Dasar Teori

a. Pengertian Hash Tabel

Hash Table adalah struktur data yang berfungsi untuk menyimpan dan mengakses dengan cara kerjanya mengonversi kunci untuk dapat mengakses dari dalam tabel hashnya itu sendiri. Hash tabel itu terdiri dari dua komponen utama array ,vektor dan fungsi hash. Array menyimpan data dalam slot yang biasa disebut dengan bucket. Dari setiap bucket ini dapat menampung beberapa item data, ini sama halnya seperti array biasa tetapi disini digunakan untuk lebih kompleks lagi dan jauh lebih dinamis dibandingkan array biasa.

Sistem hash tabel ini bekerja dengan cara mengambil dari kunci tadi dan dimasukan kedalam indeks array menggunakan fungsi hash. Lalu,ketika data akan dicari, kuncinya itu sebagai indikasi parameter untuk menjadi fungsi hashnya, dan posisi indeks array untuk mencari sebuah data tersebut. Didalam kasus hash collision, yang dimana dua atau lebih dari data yang dimiliki nilai hash yang sama, hash tabel itu menyimpan data tersebut dalam slot yang sama atau biasa disebut chainning. Sebagai berikut gambaran dari Hashing.



b. Operasi Basic Hash Tabel

Sebagai berikut operasi dari hash tabel:

1. Search – Berfungsi untuk mencari dari data atau elemen dari tabel hash tersebut.
2. Insert – ini berfungsi menambahkan data kedalam tabel hash tersebut dan dapat diakses menggunakan search yang telah dibuat.
3. Delete – berfungsi untuk menghapus data dalam setiap tabelnya.
4. Update – berfungsi untuk memperbarui data dari hash tabel itu sendiri.

c. Resolusi Collision

Resolusi Collision ini adalah tabrakan dalam tabel hash tabrakan terjadi ini ketika dua kunci atau lebih dari nilai hash yang sama. Contohnya ketika kita memiliki tabel hash memiliki indeks 0 – 6 , lalu kita masukan nilai: 3, 5, 6 dan 10 di tabelnya. Kemudian fungsi hash akan memetakan nilai 6 dan 10 ke indeks yang sama sehingga menyebabkan tabrakan. Tabrakan ini juga terjadi karena setiap tabel ini memiliki batas tertentu ketika kita membuat data lebih dari batas tersebut maka akan terjadi tabrakan dan penggunaan fungsi hash yang buruk.

Guided

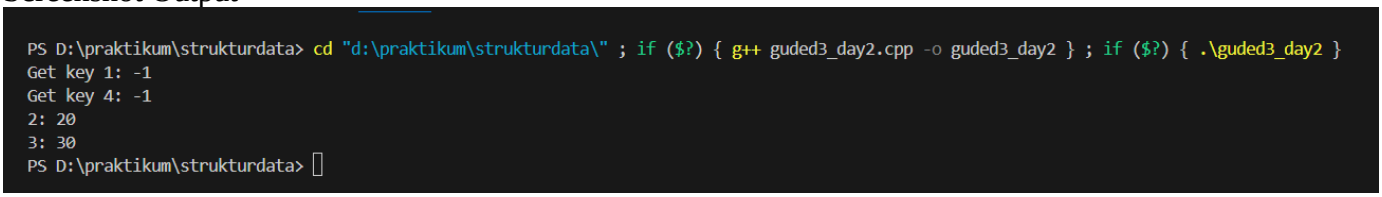
Guided 1

```
#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;
}
```

Screenshot Output



Deskripsi Program

Kode ini mengimplementasikan Hash Table dengan ukuran maksimum 10 elemen. Fungsi hash_func digunakan untuk menghitung index penyimpanan data. Struktur data Node menyimpan key dan value. Class HashTable memiliki array table untuk menyimpan pointer ke Node. Fungsi insert memasukkan data, get mencari data, remove menghapus data, dan traverse menampilkan isi Hash Table. Kode ini masih perlu implementasi penanganan tabrakan yang lebih kompleks

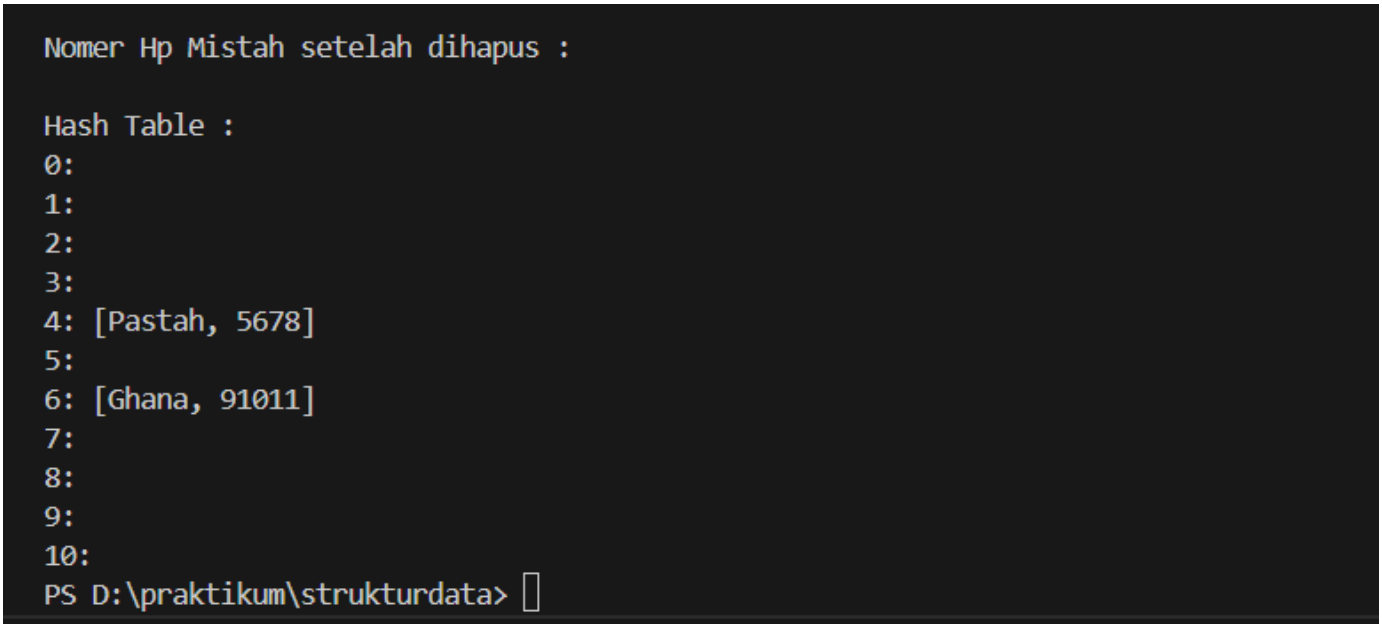
Guided 2

```cpp
#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;
}
```

Screenshot Output



Deskripsi Program

Kode ini mengimplementasikan aplikasi buku telepon sederhana menggunakan tabel hash. Tabel hash adalah struktur data yang memungkinkan penyisipan, penghapusan, dan pencarian pasangan key-value dengan cepat. Dalam hal ini, key adalah nama karyawan dan value adalah nomor telepon mereka.

Pertama-tama kode mendefinisikan kelas `HashNode` untuk menyimpan nama dan nomor telepon karyawan. Kemudian, kode tersebut mendefinisikan kelas `HashMap` yang mengimplementasikan fungsionalitas tabel hash. Tabel hash menggunakan array vector untuk menyimpan data karyawan. Setiap vector dalam array menyimpan karyawan yang memiliki nilai hash yang sama. Fungsi `hashFunc` digunakan untuk menghitung nilai hash dari nama.

Fungsi `insert` memasukkan karyawan baru ke dalam tabel hash. Pertama-tama fungsi ini menghitung nilai hash dari nama karyawan dan kemudian menambahkan karyawan tersebut ke vector yang sesuai dalam tabel hash. Fungsi `remove` menghapus karyawan dari tabel hash dengan mencari nama mereka dan kemudian menghapus data mereka dari tabel hash. Fungsi `searchByName` mencari karyawan berdasarkan nama dan mengembalikan nomor telepon mereka jika mereka ditemukan dalam tabel hash.

Fungsi `main` mendemonstrasikan bagaimana menggunakan kelas `HashMap`. Fungsi ini membuat objek hash table dan kemudian memasukkan beberapa karyawan ke dalam tabel hash. Kemudian fungsi ini mencari beberapa karyawan dan menghapus satu karyawan. Terakhir, fungsi ini mencetak isi dari tabel hash.

B. Tugas

Unguided 1

```cpp
#include <iostream>
#include <unordered_map>
#include <vector>

using namespace std;

struct Mahasiswa {
    string NIM;
    int nilai;
};

class HashTable {
private:
    unordered_map<string, Mahasiswa> tabel;
public:
    void tambahData(Mahasiswa mahasiswa) {
        tabel[mahasiswa.NIM] = mahasiswa;
    }
    void hapusData(string NIM) {
        tabel.erase(NIM);
    }
    Mahasiswa cariDataByNIM(string NIM) {
        if (tabel.find(NIM) != tabel.end()) {
            return tabel[NIM];
        } else {
            throw "NIM tidak ditemukan";
        }
    }
    vector<Mahasiswa> cariDataByScoreRange(int nilaiMin, int nilaiMax) {
        vector<Mahasiswa> hasil;
        for (auto it = tabel.begin(); it != tabel.end(); it++) {
            if (it->second.nilai >= nilaiMin && it->second.nilai <= nilaiMax) {
                hasil.push_back(it->second);
            }
        }
        return hasil;
    }
};

int main() {
    HashTable hashTable;
    int pilihan;
    do {
        cout << "Menu:" << endl;
        cout << "1. Tambah data baru" << endl;
```

```
        cout << "2. Hapus data" << endl;
        cout << "3. Cari data berdasarkan NIM" << endl;
        cout << "4. Cari data berdasarkan rentang nilai (80-90)" << endl;
        cout << "5. Keluar" << endl;
        cout << "Masukkan pilihan: ";
        cin >> pilihan;
        if (pilihan == 1) {
            Mahasiswa mahasiswa;
            cout << "Masukkan NIM: ";
            cin >> mahasiswa.NIM;
            cout << "Masukkan nilai: ";
            cin >> mahasiswa.nilai;
            hashTable.tambahData(mahasiswa);
            cout << "Data berhasil ditambahkan" << endl;
            cout << endl;
        } else if (pilihan == 2) {
            string NIM;
            cout << "Masukkan NIM yang ingin dihapus: ";
            cin >> NIM;
            hashTable.hapusData(NIM);
            cout << "Data berhasil dihapus" << endl;
            cout << endl;
        } else if (pilihan == 3) {
            string NIM;
            cout << "Masukkan NIM yang ingin dicari: ";
            cin >> NIM;
            try {
                Mahasiswa mahasiswa = hashTable.cariDataByNIM(NIM);
                cout << "NIM: " << mahasiswa.NIM << ", Nilai: " << mahasiswa.nilai <<
endl;
            } catch (const char *msg) {
                cerr << msg << endl;
            }
            cout << endl;
        } else if (pilihan == 4) {
            int nilaiMin, nilaiMax;
            cout << "Masukkan rentang nilai yang ingin dicari (contoh: 80 90): ";
            cin >> nilaiMin >> nilaiMax;
            vector<Mahasiswa> hasil = hashTable.cariDataByScoreRange(nilaiMin, nilaiMax);
            for (Mahasiswa mahasiswa : hasil) {
                cout << "NIM: " << mahasiswa.NIM << ", Nilai: " << mahasiswa.nilai <<
endl;
            }
            cout << endl;
        }
    } while (pilihan != 5);

    return 0;
}
```

Screenshot Output
Soal No. 1 Membuat menu untuk menambahkan ,mengubah dan melihat nama dan NIM mahasiswa,sebagai berikut outputnya :

`

- Tampilan Operasi Insert

```
1. Tambah data baru
2. Hapus data
3. Cari data berdasarkan NIM
4. Cari data berdasarkan rentang nilai (80-90)
5. Keluar
Masukkan pilihan: 1
Masukkan NIM: 2311102016
Masukkan nilai: 90
Data berhasil ditambahkan
```

- Tampilan Operasi Remove

```
Menu:
1. Tambah data baru
2. Hapus data
3. Cari data berdasarkan NIM
4. Cari data berdasarkan rentang nilai (80-90)
5. Keluar
Masukkan pilihan: 2
Masukkan NIM yang ingin dihapus: 2311102111
Data berhasil dihapus
```

- Tampilan Pencarian Mahasiswa

```
Menu:
1. Tambah data baru
2. Hapus data
3. Cari data berdasarkan NIM
4. Cari data berdasarkan rentang nilai (80-90)
5. Keluar
Masukkan pilihan: 3
Masukkan NIM yang ingin dicari: 2311102016
NIM: 2311102016, Nilai: 90
```

- Tampilan Operasi Rentang Nilai

```
Menu:
1. Tambah data baru
2. Hapus data
3. Cari data berdasarkan NIM
4. Cari data berdasarkan rentang nilai (80-90)
5. Keluar
Masukkan pilihan: 4
Masukkan rentang nilai yang ingin dicari (contoh: 80 90): 75 95
NIM: 2311102016, Nilai: 90
```

Deskripsi Program

Kode ini menggambarkan sistem manajemen data mahasiswa menggunakan tabel hash. Tabel hash adalah struktur data yang efisien untuk menyimpan dan mengambil data berdasarkan pasangan key-value. Dalam hal ini, key adalah NIM (Nomor Induk Mahasiswa) dan value adalah struct `Mahasiswa` yang menyimpan nama dan nilai mahasiswa.

Kode ini mendefinisikan struct `Mahasiswa` untuk mewakili seorang mahasiswa dengan atribut `NIM` dan `nilai`. Kemudian, kode ini mendefinisikan class `HashTable` yang mengimplementasikan fungsi tabel hash. Tabel hash menggunakan `unordered_map` untuk menyimpan data mahasiswa. `Unordered_map` adalah struktur data bawaan C++ yang menyediakan operasi penyisipan, penghapusan, dan pencarian berbasis hash yang efisien.

Class `HashTable` menyediakan empat fungsi utama:

1. `tambahData(Mahasiswa mahasiswa)`: Fungsi ini menambahkan mahasiswa baru ke dalam tabel hash. Fungsi ini menerima struct `Mahasiswa` sebagai input dan memasukkannya ke dalam tabel hash menggunakan NIM mahasiswa sebagai key.
2. `hapusData(string NIM)`: Fungsi ini menghapus mahasiswa dari tabel hash berdasarkan NIM mereka. Fungsi ini menerima NIM mahasiswa sebagai input dan menggunakan fungsi `erase` dari `unordered_map` untuk menghapus entri yang sesuai.
3. `cariDataByNIM(string NIM)`: Fungsi ini mencari mahasiswa di dalam tabel hash berdasarkan NIM mereka. Fungsi ini menerima NIM mahasiswa sebagai input dan menggunakan fungsi `find` dari `unordered_map` untuk menemukan entri yang sesuai. Jika ditemukan, fungsi ini mengembalikan struct `Mahasiswa`; jika tidak ditemukan, fungsi ini akan mengeluarkan exception yang menunjukkan bahwa mahasiswa tidak ditemukan.
4. `cariDataByScoreRange(int nilaiMin, int nilaiMax)`: Fungsi ini mencari mahasiswa di dalam rentang nilai tertentu. Fungsi ini menerima nilai minimum dan maksimum sebagai input dan beriterasi melalui tabel hash, memeriksa nilai setiap mahasiswa dan menambahkan mereka yang berada dalam rentang tersebut ke dalam vektor hasil. Vektor hasil kemudian dikembalikan.

Fungsi `main` membuat objek `HashTable` dan menampilkan menu kepada pengguna, memungkinkan mereka untuk menambah mahasiswa baru, menghapus mahasiswa, mencari mahasiswa berdasarkan NIM, atau mencari mahasiswa dalam rentang nilai tertentu. Pengguna memilih opsi, dan fungsi terkait dari class `HashTable` dipanggil untuk melakukan operasi yang diinginkan. Looping berlanjut sampai pengguna memilih untuk keluar dari program.