LAPORAN PRAKTIKUM

MODUL 5
HASH TABLE

Disusun oleh:
Rakha Yudhistira
NIM: 2311102010

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

1. Mahasiswa mampu menjelaskan definisi dan konsep dari Hash Code
2. Mahasiswa mampu menerapkan Hash Code kedalam pemrograman.

BAB II

DASAR TEORI

1. Pengertian

   Hashing adalah suatu teknik yang digunakan dalam pemrograman untuk melakukan penambahan, penghapusan, dan pencarian data dengan waktu rata-rata konstan. Proses ini melibatkan penggunaan fungsi hash untuk menentukan lokasi atau indeks penyimpanan data dalam struktur data yang disebut hash tables. Hash tables sendiri merupakan array dengan sel-sel yang telah ditentukan ukurannya, yang dapat menyimpan data atau kunci yang terkait dengan data. Setiap kunci memiliki fungsi hash yang digunakan untuk memetakan kunci tersebut ke bilangan dalam rentang tertentu, biasanya dari 0 hingga ukuran hash table dikurangi satu. Dengan menggunakan teknik hashing, operasi penambahan, penghapusan, dan pencarian data dapat dilakukan dengan efisiensi yang tinggi dan waktu eksekusi yang konsisten, terutama dalam kasus waktu rata-rata konstan.

2. Fungsi

   Fungsi hash harus memiliki sifat berikut:

   - Mudah dihitung.
   - Memastikan bahwa dua kunci yang berbeda akan dipetakan pada dua sel yang berbeda pada array. Meskipun pada umumnya hal ini tidak selalu bisa diterapkan karena kemungkinan terjadinya collision, di mana dua kunci yang berbeda dipetakan pada sel yang sama. Collision bisa terjadi karena jumlah kemungkinan kunci yang mungkin lebih besar dari jumlah sel dalam array.
   - Dapat dicapai dengan menggunakan direct-address table, terutama ketika semesta dari key relatif kecil.
   - Memiliki kemampuan untuk membagi kunci secara merata pada seluruh sel dalam array.
   - Menggunakan fungsi hash sederhana seperti fungsi mod (sisa bagi) dengan bilangan prima.
   - Dapat menggunakan manipulasi digit dengan kompleksitas rendah dan distribusi kunci yang merata untuk memastikan hasil hash yang efisien dan merata

3. Operasi-operasi
    a) Pencarian (Search): Digunakan untuk mencari elemen/data dalam Hash Table berdasarkan kunci atau indeksnya. Pencarian dilakukan dengan menggunakan fungsi hash untuk menghasilkan indeks elemen yang dicari.
    b) Penyisipan (Insertion): Operasi ini digunakan untuk menyisipkan elemen/data baru ke dalam Hash Table. Elemen baru akan ditempatkan pada indeks yang dihasilkan oleh fungsi hash.
    c) Penghapusan (Deletion): Digunakan untuk menghapus elemen/data dari Hash Table berdasarkan kunci atau indeksnya. Elemen yang dihapus akan dihapus dari indeks yang dihasilkan oleh fungsi hash.
    d) Update: Operasi ini digunakan untuk mengubah nilai elemen/data yang sudah ada dalam Hash Table. Nilai elemen dapat diubah berdasarkan kunci atau indeksnya.
    e) Collision Handling: Collision terjadi ketika dua atau lebih elemen memiliki indeks yang sama setelah melalui fungsi hash. Operasi ini digunakan untuk menangani collision dan memastikan bahwa elemen-elemen dengan indeks yang sama dapat disimpan dan diakses dengan benar.
    f) Resize: Operasi ini digunakan untuk mengubah ukuran Hash Table jika jumlah elemen/data yang disimpan melebihi kapasitas yang ditentukan. Resize dilakukan untuk menjaga efisiensi dan kinerja Hash Table.
    g) Iterasi: Operasi yang digunakan untuk mengakses dan memproses semua elemen/data yang ada dalam Hash Table secara berurutan. Ini memungkinkan untuk melakukan operasi seperti pencarian, penyisipan, dan penghapusan secara teratur pada setiap elemen dalam Hash Table.

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:
```

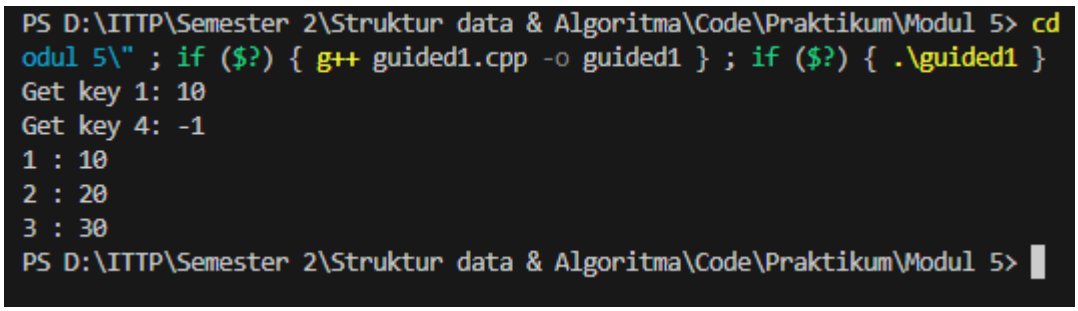
```
    HashTable()
    {
        table = new Node *[MAX_SIZE]();
    }
    ~HashTable()
    {
        for (int i = 0; i < MAX_SIZE; i++)
        {
            Node *current = table[i];
            while (current != nullptr)
            {
                Node *temp = current;
                current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

Kode tersebut merupakan implementasi hash table dengan teknik chaining untuk menangani collision. Ini menggunakan struktur data Node untuk merepresentasikan setiap simpul dalam linked list. Setiap simpul menyimpan sebuah pasangan kunci-nilai. Kelas hash_table memiliki metode untuk menyisipkan, mencari, dan menghapus elemen berdasarkan kunci, serta untuk menelusuri seluruh isi hash table. Metode hashing sederhana digunakan untuk memetakan kunci ke indeks hash table. Contoh penggunaan hash table juga diberikan di dalam fungsi main() untuk demonstrasi.

2. Guided 2

Source code

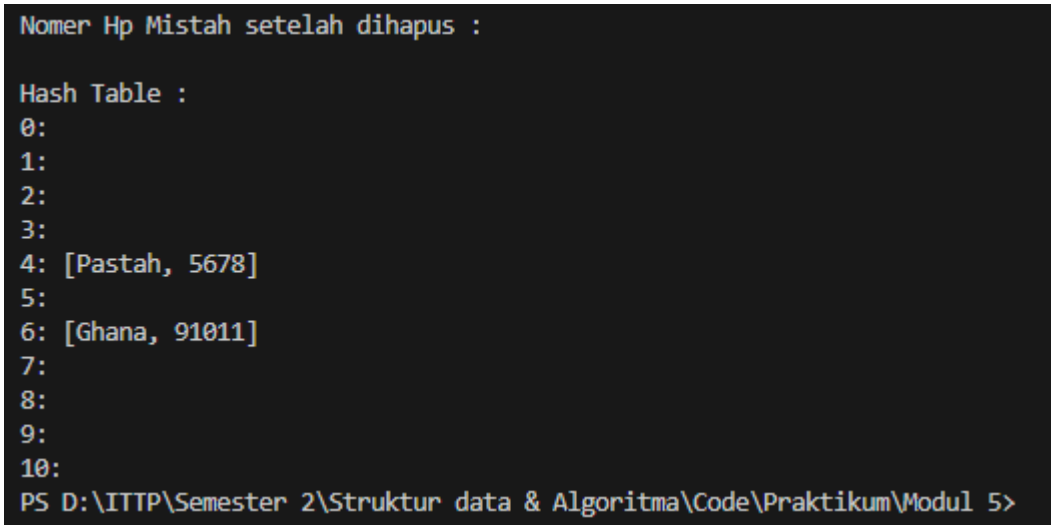
```
#include <iostream>
#include <string>
#include <vector>
using namespace std;
const int TABLE_SIZE = 11;
string name;
string phone_number;
class HashNode
{
public:
    string name;
    string phone_number;
    HashNode(string name, string phone_number)
    {
        this->name = name;
        this->phone_number = 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;
}
```

Screenshoot program



Deskripsi Program

Program tersebut adalah implementasi sederhana dari hash map menggunakan teknik chaining untuk menangani collision. Hash map digunakan untuk memetakan nama ke nomor telepon terkait. Setiap entri dalam hash map direpresentasikan oleh kelas HashNode, yang menyimpan nama dan nomor telepon. Kelas HashMap memiliki fungsi untuk menyisipkan, menghapus, mencari berdasarkan nama, dan mencetak isi seluruh hash map. Teknik hashing yang digunakan adalah menjumlahkan nilai ASCII dari karakter-karakter dalam nama dan mengambil modulo dengan ukuran hash table. Contoh penggunaan dari hash map juga diberikan di dalam fungsi main() untuk demonstrasi.

LATIHAN KELAS - UNGUIDED

1. **Unguided 1**

   Source code

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

using namespace std;

// Struktur data untuk mahasiswa
struct Mahasiswa {
    long long NIM; // Menggunakan long long untuk NIM
    int nilai;
};

// Class untuk hash table
class HashTable {
private:
    static const int hashGroup = 10; // Jumlah grup hash
    vector<list<Mahasiswa>> table; // Array dari linked list
untuk menyimpan data

public:
    HashTable() : table(hashGroup) {}

    // Fungsi hashing sederhana
    int hashFunction(long long NIM) {
        return NIM % hashGroup;
    }

    // Menambahkan data mahasiswa ke hash table
    void tambahData(long long NIM, int nilai) {
```

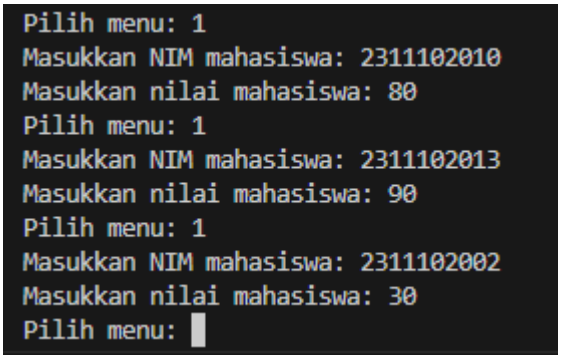
```
        Mahasiswa mhs;
        mhs.NIM = NIM;
        mhs.nilai = nilai;
        int hashKey = hashFunction(NIM);
        table[hashKey].push_back(mhs);
    }

    // Menghapus data mahasiswa dari hash table berdasarkan NIM
    void hapusData(long long NIM) {
        int hashKey = hashFunction(NIM);
        for (auto it = table[hashKey].begin(); it != table[hashKey].end(); ++it) {
            if ((*it).NIM == NIM) {
                table[hashKey].erase(it);
                break;
            }
        }
    }

    // Mencari data mahasiswa berdasarkan NIM
    void cariByNIM(long long NIM) {
        int hashKey = hashFunction(NIM);
        for (auto mhs : table[hashKey]) {
            if (mhs.NIM == NIM) {
                cout << "NIM: " << mhs.NIM << ", Nilai: " << mhs.nilai << endl;
                return;
            }
        }
        cout << "Data mahasiswa dengan NIM " << NIM << " tidak ditemukan." << endl;
    }
```

```
    // Mencari data mahasiswa berdasarkan rentang nilai (80 -
90)
    void cariByRange() {
        for (const auto& group : table) {
            for (const auto& mhs : group) {
                if (mhs.nilai >= 80 && mhs.nilai <= 90) {
                    cout << "NIM: " << mhs.NIM << ", Nilai: " <<
mhs.nilai << endl;
                }
            }
        }
    }
};

int main() {
    HashTable hashTable;

    // Menu
    cout << "Menu:" << 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;

    int choice;
    do {
        cout << "Pilih menu: ";
        cin >> choice;
        switch (choice) {
            case 1:
                long long NIM; // Menggunakan long long untuk
NIM
```

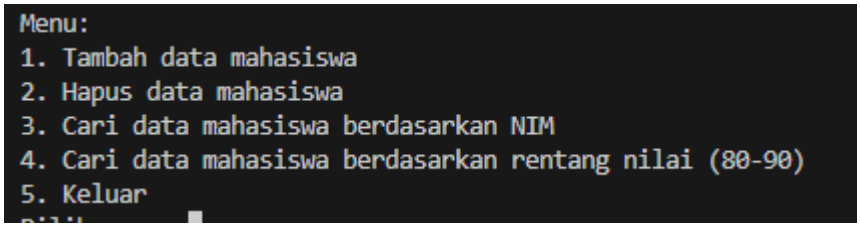
```
                int nilai;
                cout << "Masukkan NIM mahasiswa: ";
                cin >> NIM;
                cout << "Masukkan nilai mahasiswa: ";
                cin >> nilai;
                hashTable.tambahData(NIM, nilai);
                break;
            case 2:
                cout << "Masukkan NIM mahasiswa yang ingin
dihapus: ";
                cin >> NIM;
                hashTable.hapusData(NIM);
                break;
            case 3:
                cout << "Masukkan NIM mahasiswa yang ingin
dicari: ";
                cin >> NIM;
                hashTable.cariByNIM(NIM);
                break;
            case 4:
                cout << "Mahasiswa dengan nilai antara 80 dan
90:" << endl;
                hashTable.cariByRange();
                break;
            case 5:
                cout << "Keluar dari program." << endl;
                break;
            default:
                cout << "Pilihan tidak valid. Silakan pilih
lagi." << endl;
        }
    } while (choice != 5);
    return 0;
}
```

Screenshoot program

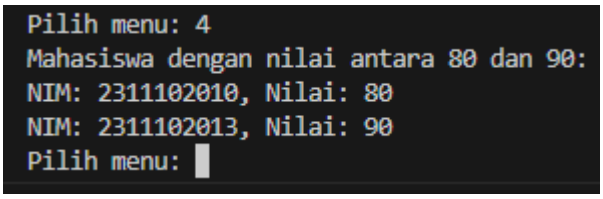
a. Setiap mahasiswa memiliki NIM dan nilai



b. Program memiliki tampilan pilihan menu berisi poin C



c. Implementasikan fungsi untuk menambahkan data baru, menghapus data, mencari data berdasarkan NIM, dan mencari data berdasarkan rentang nilai (80 – 90)



Deskripsi program

Program ini menggunakan hash table yang diimplementasikan dengan struktur data linked list untuk menangani bentrokan (collisions). Program ini dibuat untuk menyimpan dan mengelola data mahasiswa yang terdiri dari NIM (Nomor Induk Mahasiswa) dan nilai. Dalam program ini terdapat beberapa operasi, yaitu menambahkan data mahasiswa, menghapus data mahasiswa, mencari data mahasiswa berdasarkan NIM, dan mencari mahasiswa berdasarkan rentang nilai. Hash table memungkinkan operasi pencarian, penambahan, dan penghapusan data dilakukan dengan efisien menggunakan fungsi hashing sederhana untuk menentukan posisi penyimpanan data dalam array linked list.

BAB IV

KESIMPULAN

Hash table adalah struktur data yang memungkinkan penyimpanan dan pencarian data dengan efisien menggunakan kunci. Fungsi hash mengubah kunci menjadi indeks dalam array, sehingga memungkinkan akses data dengan waktu konstan. Namun, perlu ada penanganan untuk tabrakan hash, yaitu ketika dua kunci menghasilkan indeks yang sama. Walaupun hash table menawarkan kecepatan akses data yang tinggi, ada kelemahan seperti overhead penyimpanan dan kompleksitas analisis yang bergantung pada kualitas fungsi hash. Akan tetapi, hash table tetap menjadi pilihan yang efektif untuk aplikasi yang membutuhkan operasi pencarian, penambahan, dan penghapusan data dengan cepat.

DAFTAR PUSTAKA

Asisten Praktikum. (2024). MODUL 5 HASH TABLE. Learning Managament System

Vijaykrishna Ram, Bradley Kouchi. (13 Jan 2023). How To Implement a Sample Hash Table in C/C++. Diakses pada 14 Mei 2024 dari https://www.digitalocean.com/community/tutorials/hash-table-in-c-plus-plus