LAPORAN PRAKTIKUM
MODUL V
HASH TABLE

Disusun oleh:

Annisa Al Jauhar

NIM: 2311102014

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. TUJUAN PRAKTIKUM

1. Mahasiswa mampu menjelaskan definisi dan konsep dari hash table
2. Mahasiswa mampu menerapkan Hash Code kedalam pemograman

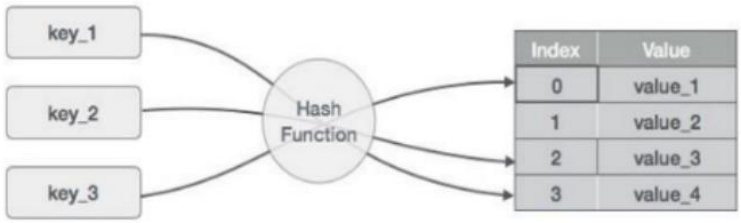
BAB II

DASAR TEORI

B. DASAR TEORI

a) 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. Array menyimpan data dalam slot-slot yang disebut bucket. Setiap bucket dapat menampung satu atau beberapa item data. Fungsi hash digunakan untuk menghasilkan nilai unik dari setiap item data, yang digunakan sebagai indeks array. Dengan cara ini, hash table memungkinkan pencarian data dalam waktu yang konstan (O(1)) dalam kasus terbaik. Sistem hash table bekerja dengan cara mengambil input kunci dan memetakkannya 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.



b) 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

c) Operasi Hash Table

1. Insertion: Memasukkan data baru ke dalam hash table dengan memanggil fungsi hash untuk menentukan posisi bucket yang tepat, dan kemudian menambahkan data ke bucket tersebut.

2. Deletion: Menghapus data dari hash table dengan mencari data menggunakan fungsi hash, dan kemudian menghapusnya dari bucket yang sesuai.

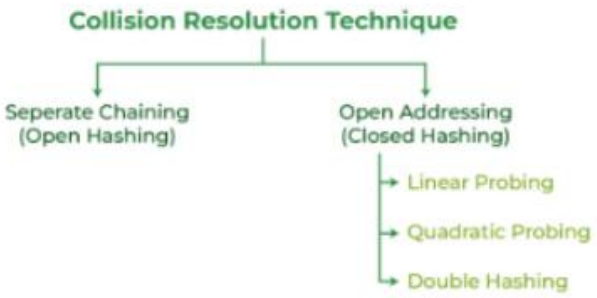
3. Searching: Mencari data dalam hash table dengan memasukkan input kunci ke fungsi hash untuk menentukan posisi bucket, dan kemudian mencari data di dalam bucket yang sesuai.

4. Update: Memperbarui data dalam hash table dengan mencari data menggunakan fungsi hash, dan kemudian memperbarui data yang ditemukan.

5. Traversal: Melalui seluruh hash table untuk memproses semua data yang ada dalam tabel

D) Collision Resolution

Keterbatasan tabel hash adalah jika dua angka dimasukkan ke dalam fungsi hash menghasilkan nilai yang sama. Hal ini disebut dengan collision. Ada dua teknik untuk menyelesaikan masalah ini diantaranya :



1. Open Hashing (Chaining)

Metode chaining mengatasi collision dengan cara menyimpan semua item data dengan nilai indeks yang sama ke dalam sebuah linked list. Setiap node pada linked list merepresentasikan satu item data. Ketika ada pencarian atau penambahan item data, pencarian atau penambahan dilakukan pada linked list yang sesuai dengan indeks yang telah dihitung dari kunci yang di hash. Ketika linked list memiliki banyak node, pencarian atau penambahan item data menjadi lambat, karena harus mencari di seluruh linked list. Namun, chaining dapat mengatasi jumlah item data yang besar dengan efektif, karena keterbatasan array dihindari.

2. Closed Hashing

● Linear Probing Pada saat terjadi collision, maka akan mencari posisi yang kosong di bawah tempat terjadinya collision, jika masih penuh terus ke bawah, hingga ketemu tempat yang kosong. Jika tidak ada tempat yang kosong berarti HashTable sudah penuh.

● Quadratic Probing Penanganannya hampir sama dengan metode linear, hanya lompatannya tidak satu-satu, tetapi quadratic (12, 22, 32, 42, ...)

● Double Hashing Pada saat terjadi collision, terdapat fungsi hash yang kedua untuk menentukan posisinya kembali.

BAB III

GUIDED

1. Guided 1

Source code

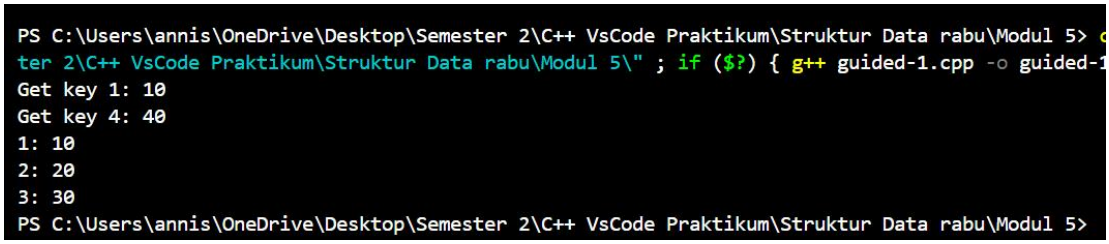
```
#include <iostream>
using namespace std;
const int MAX_SIZE = 10;
// Fungsi hash sederhana
int hash_func(int key)
{
    return key % MAX_SIZE;
}
// Struktur data untuk setiap node
struct Node
{
    int key;
    int value;
    Node *next;
    Node(int key, int value) : key(key), value(value),
                               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);
```

```
    ht.insert(4, 40);
    // 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 yang adalah implementasi sederhana dari struktur data Hash Table dalam. Hash table adalah struktur data yang memungkinkan penyimpanan dan pencarian efisien dari pasangan kunci-nilai (key-value).

Dalam program ini, terdapat definisi fungsi hash sederhana yang menggunakan operasi modulo untuk menentukan indeks dalam hash table. Setiap elemen dalam hash table direpresentasikan sebagai larik pointer ke node. Setiap node menyimpan pasangan kunci-nilai beserta dengan pointer ke node berikutnya untuk menangani tumpang tindih (collision).

Kelas HashTable menyediakan metode untuk operasi dasar pada hash table, termasuk penambahan, pencarian, penghapusan, dan penelusuran. Metode insert() digunakan untuk menambahkan pasangan kunci-nilai baru ke dalam hash table. Metode get() digunakan untuk mencari nilai yang terkait dengan kunci yang diberikan. Metode remove() digunakan untuk menghapus pasangan kunci-nilai yang sesuai dengan kunci yang diberikan. Metode traverse() digunakan untuk menampilkan semua pasangan kunci-nilai yang tersimpan dalam hash table.

Dalam fungsi main(), objek HashTable dibuat dan beberapa operasi dilakukan. Ini termasuk penambahan pasangan kunci-nilai, pencarian nilai yang terkait dengan kunci tertentu, penghapusan pasangan kunci-nilai, dan penelusuran isi hash table.

2. Guided 2

Source code

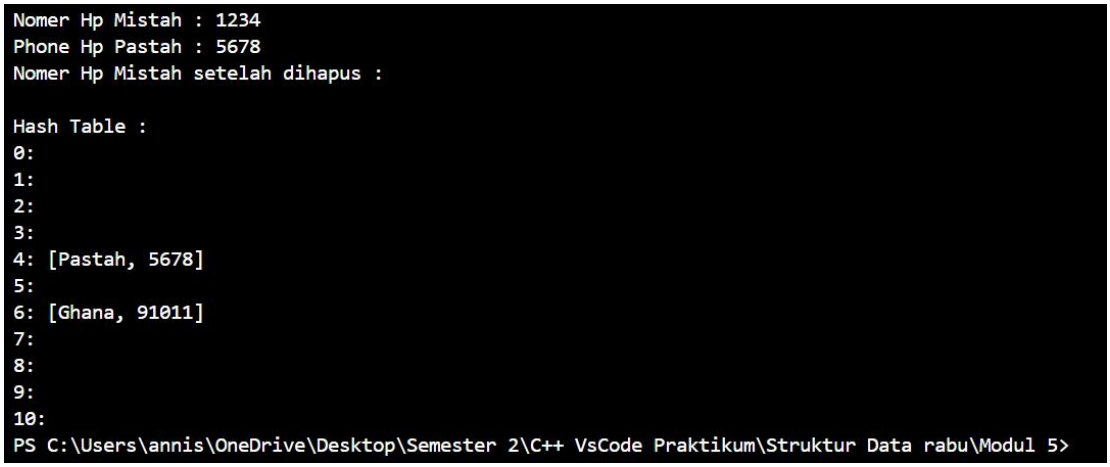
```
#include <iostream>
#include <string>
#include <vector>
using namespace std;
const int TABLE_SIZE = 11;
string name;
string phone_number;
class HashNode
{
public:
    string name;
    string phone_number;
    HashNode(string name, string phone_number)
    {
        this->name = name;
        this->phone_number = 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 dari struktur data hash map. Hash map adalah koleksi pasangan kunci-nilai di mana setiap kunci unik terhubung dengan nilai tertentu. Dalam program ini, terdapat dua kelas utama, yaitu `HashNode` dan `HashMap`.

Kelas `HashNode` digunakan untuk merepresentasikan simpul individu dalam hash map. Setiap `HashNode` memiliki dua atribut, yaitu `name` dan `phone_number`, yang mewakili nama dan nomor telepon karyawan.

Kelas `HashMap` adalah kelas utama yang digunakan untuk menyimpan dan mengelola pasangan kunci-nilai. Ini menggunakan larik vektor `table` yang memiliki ukuran tetap yang ditentukan oleh konstanta `TABLE_SIZE`. Setiap elemen vektor berisi daftar simpul yang diinisialisasi dengan nilai awal null. Metode `hashFunc`

digunakan untuk menghitung nilai hash dari kunci yang diberikan. Ini melakukan penjumlahan nilai ASCII dari setiap karakter kunci dan kemudian mengembalikan hasilnya setelah di-modulus dengan `TABLE_SIZE`.

Metode `insert` digunakan untuk menambahkan pasangan kunci-nilai ke dalam hash map. Jika kunci sudah ada dalam tabel hash, nilai yang terkait dengan kunci tersebut diperbarui; jika tidak, pasangan kunci-nilai baru ditambahkan ke dalam daftar simpul yang sesuai. Metode `remove` digunakan untuk menghapus pasangan kunci-nilai berdasarkan kunci yang diberikan dari hash map. Ini mencari kunci dalam daftar simpul yang sesuai dan menghapusnya jika ditemukan.

Fungsi `main` membuat objek `HashMap`, menambahkan beberapa pasangan kunci-nilai ke dalamnya, melakukan pencarian nilai berdasarkan kunci, menghapus pasangan kunci-nilai, dan mencetak isi hash map setelah operasi yang dilakukan.

LATIHAN KELAS - UNGUIDED

1. Unguided 1

A. Source code

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

using namespace std;

const int TABLE_SIZE = 10;

struct Student {
    string nim;
    int grade;
    Student(string nim, int grade) : nim(nim), grade(grade) {}
};

struct HashNode {
    vector<Student> students;
};

class HashTable {
private:
    vector<HashNode> table;

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

    int hashFunc(string nim) {
        int hash_val = 0;
        for (char c : nim) {
            hash_val += c;
```

```
        }
        return hash_val % TABLE_SIZE;
    }

    void insert(string nim, int grade) {
        int index = hashFunc(nim);
        table[index].students.push_back(Student(nim, grade));
    }

    void remove(string nim) {
        int index = hashFunc(nim);
        vector<Student>& students = table[index].students;
        for (auto it = students.begin(); it != students.end();
++it) {
            if (it->nim == nim) {
                students.erase(it);
                break;
            }
        }
    }

    void searchByNIM(string nim) {
        int index = hashFunc(nim);
        vector<Student>& students = table[index].students;
        for (const auto& student : students) {
            if (student.nim == nim) {
                cout << "Mahasiswa dengan NIM " << nim << "
ditemukan dengan nilai " << student.grade << endl;
                return;
            }
        }
        cout << "Mahasiswa dengan NIM " << nim << " tidak
ditemukan" << endl;
    }
```

```
    void searchByGradeRange() {
        cout << "Mahasiswa dengan nilai antara 80 dan 90:" <<
endl;
        for (const auto& node : table) {
            for (const auto& student : node.students) {
                if (student.grade >= 80 && student.grade <= 90) {
                    cout << "NIM: " << student.nim << ", Nilai: "
<< student.grade << endl;
                }
            }
        }
    }
};

int main() {
    HashTable studentTable;
    int choice;
    string nim;
    int grade;

    do {
        cout << "\nMenu:\n1. Tambah Data Mahasiswa\n2. Hapus Data
Mahasiswa\n3. Cari Data Mahasiswa berdasarkan NIM\n4. Cari Data
Mahasiswa berdasarkan Rentang Nilai (80 - 90)\n5. Keluar\n";
        cout << "Pilih: ";
        cin >> choice;

        switch (choice) {
        case 1:
            cout << "Masukkan NIM mahasiswa: ";
            cin >> nim;
            cout << "Masukkan nilai mahasiswa: ";
            cin >> grade;
```

```
            studentTable.insert(nim, grade);
            break;
        case 2:
            cout << "Masukkan NIM mahasiswa yang akan dihapus: ";
            cin >> nim;
            studentTable.remove(nim);
            break;
        case 3:
            cout << "Masukkan NIM mahasiswa yang ingin dicari: ";
            cin >> nim;
            studentTable.searchByNIM(nim);
            break;
        case 4:
            studentTable.searchByGradeRange();
            break;
        case 5:
            cout << "Program selesai." << endl;
            break;
        default:
            cout << "Pilihan tidak valid. Ulangi." << endl;
            break;
        }
    } while (choice != 5);

    return 0;
}
```

Screenshoot program

```
PS C:\Users\annis\OneDrive\Desktop\Semester 2\C++ VsCode Praktikum\
ter 2\C++ VsCode Praktikum\Struktur Data rabu\Modul 5\" ; if ($?) {
pCodeRunnerFile }

Menu:
1. Tambah Data Mahasiswa
2. Hapus Data Mahasiswa
3. Cari Data Mahasiswa berdasarkan NIM
4. Cari Data Mahasiswa berdasarkan Rentang Nilai (80 - 90)
5. Keluar
Pilih:
```

```
Menu:
1. Tambah Data Mahasiswa
2. Hapus Data Mahasiswa
3. Cari Data Mahasiswa berdasarkan NIM
4. Cari Data Mahasiswa berdasarkan Rentang Nilai (80 - 90)
5. Keluar
Pilih: 1
Masukkan NIM mahasiswa: 2311102014
Masukkan nilai mahasiswa: 90
```

```
Menu:
1. Tambah Data Mahasiswa
2. Hapus Data Mahasiswa
3. Cari Data Mahasiswa berdasarkan NIM
4. Cari Data Mahasiswa berdasarkan Rentang Nilai (80 - 90)
5. Keluar
Pilih: 3
Masukkan NIM mahasiswa yang ingin dicari: 2311102014
Mahasiswa dengan NIM 2311102014 ditemukan dengan nilai 90
```

```
Menu:
1. Tambah Data Mahasiswa
2. Hapus Data Mahasiswa
3. Cari Data Mahasiswa berdasarkan NIM
4. Cari Data Mahasiswa berdasarkan Rentang Nilai (80 - 90)
5. Keluar
Pilih: 4
Mahasiswa dengan nilai antara 80 dan 90:
NIM: 2311102014, Nilai: 90
```

```
Menu:
1. Tambah Data Mahasiswa
2. Hapus Data Mahasiswa
3. Cari Data Mahasiswa berdasarkan NIM
4. Cari Data Mahasiswa berdasarkan Rentang Nilai (80 - 90)
5. Keluar
Pilih: 2
Masukkan NIM mahasiswa yang akan dihapus: 2311102014
```

```
Menu:
1. Tambah Data Mahasiswa
2. Hapus Data Mahasiswa
3. Cari Data Mahasiswa berdasarkan NIM
4. Cari Data Mahasiswa berdasarkan Rentang Nilai (80 - 90)
5. Keluar
Pilih: 5
Program selesai.
PS C:\Users\annis\OneDrive\Desktop\Semester 2\C++ VsCode Praktikum\Struktur Data rabu\Modul 5>
```

Deskripsi program

Program di atas adalah implementasi dari struktur data hash table, yang digunakan untuk menyimpan data mahasiswa berdasarkan nomor induk mahasiswa (NIM) dan nilai. Struktur data hash table ini dirancang untuk menyimpan beberapa mahasiswa di dalam vektor, di mana setiap vektor berisi objek HashNode yang menampung daftar mahasiswa.

Kelas HashTable memiliki metode untuk operasi dasar pada hash table, seperti penyisipan, penghapusan, dan pencarian berdasarkan NIM atau rentang nilai. Metode insert() digunakan untuk menambahkan data mahasiswa baru ke hash table. Metode remove() digunakan untuk menghapus data mahasiswa berdasarkan NIM yang diberikan. Metode searchByNIM() digunakan untuk mencari data mahasiswa berdasarkan NIM yang diberikan. Metode searchByGradeRange() digunakan untuk mencari data mahasiswa berdasarkan rentang nilai tertentu.

Fungsi main() mengatur antarmuka pengguna untuk berinteraksi dengan hash table. Pengguna dapat menambahkan data mahasiswa baru, menghapus data mahasiswa, mencari data mahasiswa berdasarkan NIM, mencari data mahasiswa berdasarkan rentang nilai, atau keluar dari program. Program akan terus berjalan sampai pengguna memilih opsi keluar

BAB IV

KESIMPULAN

Hash Table adalah struktur data efisien yang menggunakan fungsi hash untuk menetapkan elemen ke indeks array. Fungsi hash yang baik memungkinkan distribusi key yang merata dan mengurangi collision. Ada teknik hashing seperti Truncation, Folding, dan Modular. Collision diatasi melalui Closed Hashing (Open Addressing) atau Open Hashing (Separate Chaining). Hash Table memungkinkan operasi penambahan dan pencarian yang cepat, ideal untuk data besar dan efisiensi operasi yang penting.