`LAPORAN PRAKTIKUM

MODUL V
HASH TABLE

Disusun oleh:
Brian Farrel Evandhika
NIM: 2311102037

Dosen Pengampu:
Wahyu Andi Saputra, S.Pd., M.Eng.

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
PURWOKERTO
2023

BAB I

TUJUAN PRAKTIKUM

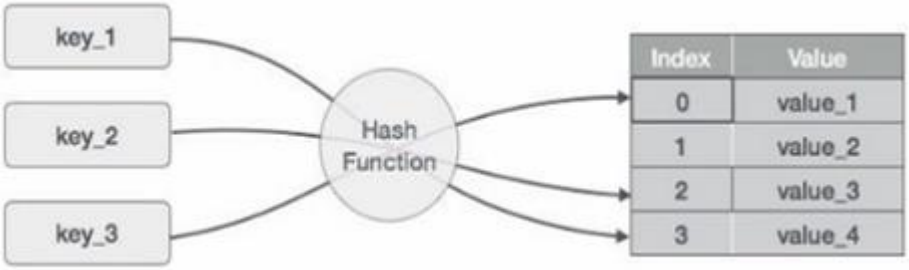
A. TUJUAN PRAKTIKUM

a. Mahasiswa mampu menjelaskan definisi dan konsep dari Hash Code

b. Mahasiswa mampu menerapkan Hash Code kedalam pemrograman

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

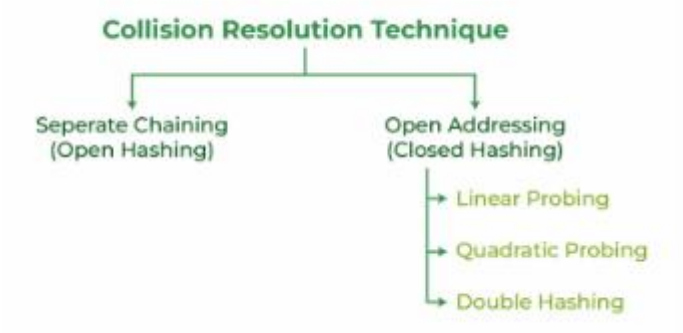


B. Fungsi Hash Table Fungsi hash membuat pemetaan antara kunci dan nilai, hal ini dilakukan melalui penggunaan rumus matematika yang dikenal sebagai fungsi hash. Hasil dari fungsi hash disebut sebagai nilai hash atau hash. Nilai hash adalah representasi dari string karakter asli tetapi biasanya lebih kecil dari aslinya.

C. 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. 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. OpenHashing (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, ...)
● DoubleHashing Pada saat terjadi collision, terdapat fungsi hash yang kedua untuk menentukan posisinya kembali

Guided

Guided 1

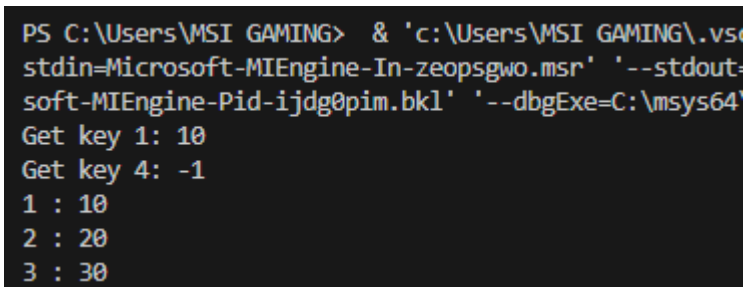
```cpp
#include <iostream>

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 Program



Deskripsi Porgram

Program tersebut mendefinisikan sebuah tabel hash sederhana menggunakan chaining untuk menangani tabrakan (collisions). Dalam program ini, ukuran maksimum dari tabel hash ditetapkan sebesar 10, dan fungsi hash yang digunakan adalah modulus dari ukuran tersebut. Program terdiri dari struktur Node yang menyimpan pasangan kunci-nilai serta pointer ke node berikutnya, dan kelas HashTable yang mengelola array dari pointer ke node. Kelas ini menyediakan metode untuk menyisipkan pasangan kunci-nilai (insert), mencari nilai berdasarkan kunci (get), menghapus node berdasarkan kunci (remove), dan menampilkan seluruh isi tabel hash (traverse). Di dalam fungsi main, beberapa pasangan kunci-nilai disisipkan, beberapa pencarian dilakukan, satu penghapusan dicoba (walaupun untuk kunci yang tidak ada), dan kemudian seluruh isi tabel hash ditampilkan.

Guided 2

```
#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

```
PS C:\Users\MSI GAMING>  & 'c:\Users\MSI GAMING\.vscode\extensi
stdin=Microsoft-MIEngine-In-ft1jef0f.gtl' '--stdout=Microsoft-M
soft-MIEngine-Pid-jughajli.guj' '--dbgExe=C:\msys64\ucrt64\bin\
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:
PS C:\Users\MSI GAMING>
```

Deskripsi Program

Program tersebut mendefinisikan sebuah tabel hash (HashMap) yang digunakan untuk menyimpan dan mengelola data pasangan nama dan nomor telepon menggunakan chaining untuk menangani tabrakan (collisions). Tabel hash ini berukuran 11, dan fungsi hash sederhana dijalankan dengan menjumlahkan nilai ASCII dari karakter dalam string nama, kemudian mengambil sisa bagi dengan ukuran tabel. Struktur data HashNode menyimpan pasangan nama dan nomor telepon, dan kelas HashMap mengelola vektor dari pointer ke HashNode. Program ini menyediakan metode untuk menambahkan data (insert), menghapus data berdasarkan nama (remove), mencari nomor telepon berdasarkan nama (searchByName), dan mencetak seluruh isi tabel hash (print). Dalam fungsi main, beberapa data karyawan disisipkan, nomor telepon dicari berdasarkan nama, satu entri dihapus, dan akhirnya seluruh isi tabel hash dicetak ke layar.

Unguided

Unguided 1

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

using namespace std;

struct MahasiswaITTP {
    string nim;
    int nilai;
    MahasiswaITTP(string nim, int nilai) : nim(nim), nilai(nilai) {}
};

struct HashNode {
    MahasiswaITTP *data;
    HashNode(MahasiswaITTP *data) : data(data) {}
    ~HashNode() {
        delete data;
    }
};

class HashTable {
private:
    const int TABLE_SIZE = 10;
    vector<HashNode *> table[10];

    int hashFunc(string key) {
        int hash_val = 0;
        for (char c : key) {
            hash_val += c;
        }
        return hash_val % TABLE_SIZE;
    }

public:
    ~HashTable() {
        for (int i = 0; i < TABLE_SIZE; i++) {
            for (auto node : table[i]) {
                delete node;
            }
        }
    }

    void tambahMahasiswa(string nim, int nilai) {
        int index = hashFunc(nim);
        table[index].push_back(new HashNode(new MahasiswaITTP(nim, nilai)));
```

```
    }

    void hapusMahasiswa(string nim) {
        int index = hashFunc(nim);
        for (auto it = table[index].begin(); it != table[index].end(); it++) {
            if ((*it)->data->nim == nim) {
                delete *it;
                table[index].erase(it);
                return;
            }
        }
    }

    MahasiswaITTP *cariMahasiswaByNIM(string nim) {
        int index = hashFunc(nim);
        for (auto node : table[index]) {
            if (node->data->nim == nim) {
                return node->data;
            }
        }
        return nullptr;
    }

    vector<MahasiswaITTP *> cariMahasiswaByNilai() {
        vector<MahasiswaITTP *> result;
        for (int i = 0; i < TABLE_SIZE; i++) {
            for (auto node : table[i]) {
                if (node->data->nilai >= 80 && node->data->nilai <= 90) {
                    result.push_back(node->data);
                }
            }
        }
        return result;
    }
};

int main() {
    HashTable hashTable;
    int choice;
    string nim;
    int nilai;

    do {
        cout << "\nMenu:\n";
        cout << "1. Tambah Mahasiswa\n";
        cout << "2. Hapus Mahasiswa\n";
        cout << "3. Cari Mahasiswa berdasarkan NIM\n";
        cout << "4. Cari Mahasiswa berdasarkan Nilai (80 - 90)\n";
```

```
        cout << "5. Keluar\n";
        cout << "Pilih: ";
        cin >> choice;

        switch (choice) {
        case 1:
            cout << "Masukkan NIM: ";
            cin >> nim;
            cout << "Masukkan Nilai: ";
            cin >> nilai;
            hashTable.tambahMahasiswa(nim, nilai);
            break;
        case 2:
            cout << "Masukkan NIM mahasiswa yang ingin dihapus: ";
            cin >> nim;
            hashTable.hapusMahasiswa(nim);
            break;
        case 3:
            cout << "Masukkan NIM mahasiswa yang ingin dicari: ";
            cin >> nim;
            {
                MahasiswaITTP *mahasiswa = hashTable.cariMahasiswaByNIM(nim);
                if (mahasiswa != nullptr) {
                    cout << "Mahasiswa ditemukan: NIM = " << mahasiswa->nim <<
", Nilai = " << mahasiswa->nilai << endl;
                } else {
                    cout << "Mahasiswa dengan NIM " << nim << " tidak
ditemukan." << endl;
                }
            }
            break;
        case 4:
        {
            vector<MahasiswaITTP *> mahasiswas =
hashTable.cariMahasiswaByNilai();
            if (mahasiswas.empty()) {
                cout << "Tidak ada mahasiswa dengan nilai antara 80 - 90." <<
endl;
            } else {
                cout << "Daftar mahasiswa dengan nilai antara 80 - 90:" <<
endl;
                for (MahasiswaITTP *mahasiswa : mahasiswas) {
                    cout << "NIM = " << mahasiswa->nim << ", Nilai = " <<
mahasiswa->nilai << endl;
                }
            }
        }
        break;
```

```
        case 5:
            cout << "Program selesai.\n";
            break;
        default:
            cout << "Pilihan tidak valid. Silakan pilih lagi.\n";
            break;
        }
    } while (choice != 5);
    return 0;
}
```

Screenshot Program

```
Menu:
1. Tambah Mahasiswa
2. Hapus Mahasiswa
3. Cari Mahasiswa berdasarkan NIM
4. Cari Mahasiswa berdasarkan Nilai (80 - 90)
5. Keluar
Pilih: 1
Masukkan NIM: 2311102037
Masukkan Nilai: 100

Menu:
1. Tambah Mahasiswa
2. Hapus Mahasiswa
3. Cari Mahasiswa berdasarkan NIM
4. Cari Mahasiswa berdasarkan Nilai (80 - 90)
5. Keluar
Pilih: 3
Masukkan NIM mahasiswa yang ingin dicari: 231110203
Mahasiswa ditemukan: NIM = 2311102037, Nilai = 100

Menu:
1. Tambah Mahasiswa
2. Hapus Mahasiswa
3. Cari Mahasiswa berdasarkan NIM
4. Cari Mahasiswa berdasarkan Nilai (80 - 90)
5. Keluar
Pilih: 4
Tidak ada mahasiswa dengan nilai antara 80 - 90.

Menu:
1. Tambah Mahasiswa
2. Hapus Mahasiswa
3. Cari Mahasiswa berdasarkan NIM
4. Cari Mahasiswa berdasarkan Nilai (80 - 90)
5. Keluar
Pilih: 1
Masukkan NIM: 2311102030
Masukkan Nilai: 85

Menu:
1. Tambah Mahasiswa
2. Hapus Mahasiswa
3. Cari Mahasiswa berdasarkan NIM
4. Cari Mahasiswa berdasarkan Nilai (80 - 90)
5. Keluar
Pilih: 4
Daftar mahasiswa dengan nilai antara 80 - 90:
NIM = 2311102030, Nilai = 85

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

Deskripsi Program

Program tersebut adalah implementasi dari sebuah tabel hash (HashTable) untuk menyimpan dan mengelola data mahasiswa, di mana setiap mahasiswa memiliki NIM (Nomor Induk Mahasiswa) dan nilai. Tabel hash ini menggunakan chaining untuk menangani tabrakan, dengan ukuran tabel sebesar 10 dan fungsi hash yang menghitung nilai hash berdasarkan jumlah karakter ASCII dalam NIM. Program ini memungkinkan pengguna untuk menambahkan mahasiswa (tambahMahasiswa), menghapus mahasiswa berdasarkan NIM (hapusMahasiswa), mencari mahasiswa berdasarkan NIM (cariMahasiswaByNIM), dan mencari mahasiswa dengan nilai antara 80 hingga 90 (cariMahasiswaByNilai). Dalam fungsi main, terdapat menu interaktif yang memungkinkan pengguna memilih operasi yang diinginkan, seperti menambah, menghapus, atau mencari data mahasiswa, serta keluar dari program. Tabel hash secara otomatis menangani alokasi dan dealokasi memori untuk objek mahasiswa yang disimpan di dalamnya.

KESIMPULAN

Dalam keseluruhan praktikum ini, kami mempelajari implementasi hash table dalam bahasa pemrograman C++. Hash table adalah struktur data yang efisien untuk menyimpan data secara asosiatif, di mana setiap entri data diakses melalui kunci uniknya sendiri. Kami memahami bahwa hash table menggunakan fungsi hash untuk mengonversi kunci menjadi indeks dalam array, memungkinkan akses data yang cepat. Dalam praktikum ini, kami telah membangun sebuah hash table untuk menyimpan data mahasiswa. Setiap mahasiswa direpresentasikan oleh struktur `MahasiswaITTP` yang memiliki atribut NIM dan nilai. Kami mengimplementasikan fungsi-fungsi dasar seperti menambahkan data mahasiswa baru, menghapus data berdasarkan NIM, dan mencari data berdasarkan NIM atau rentang nilai. Selain itu, kami juga telah membuat sebuah antarmuka pengguna sederhana menggunakan konsol, di mana pengguna dapat memilih berbagai opsi seperti menambah, menghapus, atau mencari data mahasiswa. Praktikum ini memberikan pemahaman yang baik tentang bagaimana menggunakan hash table dalam konteks aplikasi nyata dan bagaimana memanfaatkannya untuk menyimpan dan mengelola data dengan efisien.

Daftar Pustaka

[1] https://fikti.umsu.ac.id/struktur-data-hash-table-%20pengertian-cara-kerja-dan-operasi-hash-table/

[2] https://www-educba-com.translate.goog/c-plus-plus-hashtable/?_x_tr_sl=en&_x_tr_tl=id&_x_tr_hl=id&_x_tr_pto=tc