

LAPORAN PRAKTIKUM

MODUL V HASH TABLE



Disusun oleh:
Anita Nurazizah Agussalim
NIM: 2311102017

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

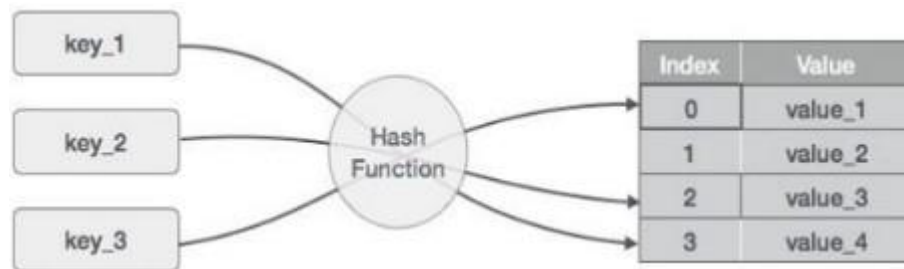
DASAR TEORI

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 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.



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:

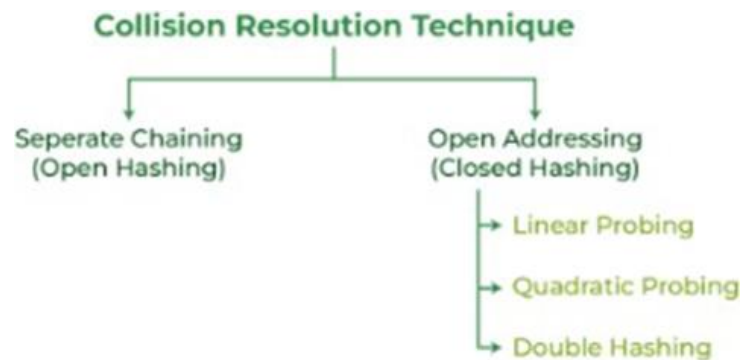
Memperbarui data dalam hash table dengan mencari data menggunakan fungsi hash, dan kemudian memperbarui data yang ditemukan.

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;
        }
    }
}

```



```

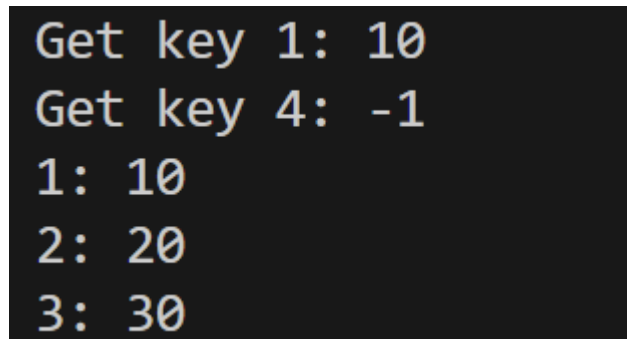
        prev = current;
        current = current->next;

    }
}
// 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);
    // 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;
}

```

```
}    tampil();  
    insertTengah(7, "lima", 2);  
    tampil();  
    hapusTengah(2);  
    tampil();  
    ubahDepan(1, "tujuh");  
    tampil();  
    ubahBelakang(8, "delapan");  
    tampil();  
    ubahTengah(11, "sembilan", 2);  
    tampil();  
    return 0;  
}
```

Screenshot Output



```
Get key 1: 10  
Get key 4: -1  
1: 10  
2: 20  
3: 30
```

Deskripsi program

Program ini mengimplementasikan tabel hash sederhana. Pertama, program ini mendefinisikan fungsi hash untuk memetakan kunci ke lokasi bucket di dalam tabel. Setiap bucket dapat menyimpan daftar tertaut yang berisi pasangan key-value. Program ini memungkinkan Anda untuk memasukkan pasangan key-value, mencari nilai berdasarkan key, menghapus pasangan key-value, dan melintasi seluruh tabel untuk melihat semua data yang tersimpan.

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 "";
    }

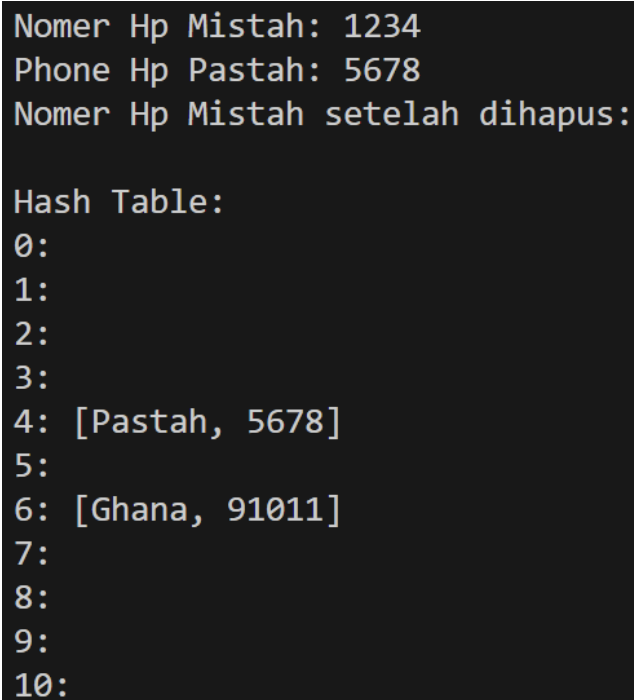
    void print()
    {
        for (int i = 0; i < TABLE_SIZE; i++)
        {
            cout << i << ": ";
            for (auto pair : table[i])
            {
                if (pair != nullptr)
                {
                    cout << "[" << pair->name << ", " << pair-
>phone_number << "];"
                }
            }
            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;
}
```

Screenshot 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:
```

Deskripsi program

Program ini membuat direktori telepon sederhana. Program ini menggunakan tabel hash untuk menyimpan data berupa nama dan nomor telepon. Kita bisa menambahkan data (nama dan nomor telepon) baru, mencari nomor telepon berdasarkan nama, menghapus data, dan melihat seluruh isi direktori.

LATIHAN KELAS - UNGUIDED

1. Implementasikan hash table untuk menyimpan data mahasiswa. Setiap mahasiswa memiliki NIM dan nilai. Implementasikan fungsi untuk menambahkan data baru, menghapus data, mencari data berdasarkan NIM, dan mencari data berdasarkan nilai. Dengan ketentuan :
 - 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).

Source code

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

using namespace std;

class Mahasiswa {
public:
    long long int nim;
    int nilai;

    Mahasiswa(long long int n, int v) : nim(n), nilai(v) {}
};

class HashTable {
private:
    static const int hashSize = 10;
    list<Mahasiswa> table[hashSize];

    int hashFunction(long long int nim) {
```

```

        return nim % hashSize;

    }

public:
    void tambahData(long long int nim, int nilai) {
        int index = hashFunction(nim);
        table[index].emplace_back(nim, nilai);
    }

    void hapusData(long long int nim) {
        int index = hashFunction(nim);
        for (auto it = table[index].begin(); it !=
table[index].end(); ++it) {
            if (it->nim == nim) {
                table[index].erase(it);
                break;
            }
        }
    }

    void cariByNIM(long long int nim) {
        int index = hashFunction(nim);
        for (auto it = table[index].begin(); it !=
table[index].end(); ++it) {
            if (it->nim == nim) {
                cout << "NIM: " << it->nim << ", Nilai: " << it-
>nilai << endl;
                return;
            }
        }
        cout << "Data tidak ditemukan." << endl;
    }

```



```

        void cariByRange(int minNilai, int maxNilai) {
            for (int i = 0; i < hashSize; ++i) {
                for (auto it = table[i].begin(); it !=
table[i].end(); ++it) {
                    if (it->nilai >= minNilai && it->nilai <=
maxNilai) {
                        cout << "NIM: " << it->nim << ", Nilai: " <<
it->nilai << endl;
                    }
                }
            }
        };

    int main() {
        HashTable hashTable;
        int choice;

        do {
            cout << "\nMenu:\n";
            cout << "1. Tambah Data\n";
            cout << "2. Hapus Data\n";
            cout << "3. Cari Data berdasarkan NIM\n";
            cout << "4. Cari Data berdasarkan Rentang Nilai\n";
            cout << "5. Keluar\n";
            cout << "Pilihan Anda: ";
            cin >> choice;

            switch (choice) {
                case 1: {
                    long long int nim;
                    int nilai;
                    cout << "Masukkan NIM: ";

```

```

        cin >> nim;
        cout << "Masukkan Nilai: ";
        cin >> nilai;
        hashTable.tambahData(nim, nilai);
        break;
    }
    case 2: {
        long long int nim;
        cout << "Masukkan NIM yang akan dihapus: ";
        cin >> nim;
        hashTable.hapusData(nim);
        break;
    }
    case 3: {
        long long int nim;
        cout << "Masukkan NIM yang akan dicari: ";
        cin >> nim;
        hashTable.cariByNIM(nim);
        break;
    }
    case 4: {
        int minNilai, maxNilai;
        cout << "Masukkan Rentang Nilai (Min Max): ";
        cin >> minNilai >> maxNilai;
        cout << "Mahasiswa dengan nilai antara " <<
minNilai << " - " << maxNilai << ":\n";
        hashTable.cariByRange(minNilai, maxNilai);
        break;
    }
    case 5:
        cout << "Terima kasih!\n";
        break;
    default:

```

```
        cout << "Pilihan tidak valid. Silakan coba  
lagi.\n";  
    }  
    } while (choice != 5);  
  
    return 0;  
}
```

Screenshot output

```
Menu:  
1. Tambah Data  
2. Hapus Data  
3. Cari Data berdasarkan NIM  
4. Cari Data berdasarkan Rentang Nilai  
5. Keluar  
Pilihan Anda: 
```

1. Tambah Data

```
Menu:  
1. Tambah Data  
2. Hapus Data  
3. Cari Data berdasarkan NIM  
4. Cari Data berdasarkan Rentang Nilai  
5. Keluar  
Pilihan Anda: 1  
Masukkan NIM: 2311102017  
Masukkan Nilai: 90
```

2. Hapus Data

```
Menu:  
1. Tambah Data  
2. Hapus Data  
3. Cari Data berdasarkan NIM  
4. Cari Data berdasarkan Rentang Nilai  
5. Keluar  
Pilihan Anda: 2  
Masukkan NIM yang akan dihapus: 2311102005
```

3. Cari Data Berdasarkan NIM

```
Menu:
1. Tambah Data
2. Hapus Data
3. Cari Data berdasarkan NIM
4. Cari Data berdasarkan Rentang Nilai
5. Keluar
Pilihan Anda: 3
Masukkan NIM yang akan dicari: 2311102016
NIM: 2311102016, Nilai: 86
```

4. Cari Data Berdasarkan rentang nilai (80-90)

```
Menu:
1. Tambah Data
2. Hapus Data
3. Cari Data berdasarkan NIM
4. Cari Data berdasarkan Rentang Nilai
5. Keluar
Pilihan Anda: 4
Masukkan Rentang Nilai (Min Max): 80 90
Mahasiswa dengan nilai antara 80 - 90:
NIM: 2311102016, Nilai: 86
NIM: 2311102017, Nilai: 90
NIM: 2311102029, Nilai: 83
```

5. Keluar

```
Menu:
1. Tambah Data
2. Hapus Data
3. Cari Data berdasarkan NIM
4. Cari Data berdasarkan Rentang Nilai
5. Keluar
Pilihan Anda: 5
Terima kasih!
```

Deskripsi program

Program di atas adalah implementasi dari hash table untuk menyimpan data mahasiswa, dengan setiap mahasiswa memiliki NIM dan nilai. Program menyediakan sebuah menu interaktif yang memungkinkan pengguna untuk menambahkan data baru, menghapus data berdasarkan NIM, mencari data berdasarkan NIM, dan mencari data berdasarkan rentang nilai yang ditentukan oleh pengguna. Setiap data mahasiswa disimpan dalam hash table, di mana kunci hash didasarkan pada NIM untuk memungkinkan akses cepat. Melalui penggunaan struktur data list, program dapat menangani hash collision. Program juga memungkinkan pengguna untuk menentukan rentang nilai yang ingin dicari saat mencari data berdasarkan rentang nilai.

BAB IV

KESIMPULAN

Hash table adalah struktur data yang efisien untuk menyimpan dan mengakses data dengan kecepatan konstan, terutama untuk operasi pencarian, penyisipan, dan penghapusan. Ini terdiri dari dua komponen utama: array (atau vektor) dan fungsi hash. Fungsi hash digunakan untuk memetakan kunci data ke indeks array, sehingga memungkinkan pencarian data dalam waktu yang konstan dalam kasus terbaik. Ada beberapa metode penanganan collision, di antaranya chaining dan closed hashing dengan teknik seperti linear probing, quadratic probing, dan double hashing. Metode chaining memungkinkan item data dengan nilai hash yang sama disimpan dalam linked list yang terkait dengan indeks array, sementara metode closed hashing mengatasi collision dengan mencari posisi kosong terdekat di tabel hash. Dengan demikian, hash table adalah alat yang kuat untuk mengelola dan mengakses data dengan cepat dan efisien.

DAFTAR PUSTAKA

Asisten Praktikum. (2024). MODUL 5: HASH TABLE.

Karumanchi, N. (2016). Data Structures and algorithms made easy: Concepts, problems, Interview Questions. CareerMonk Publications.