LAPORAN PRAKTIKUM
MODUL 5
HASH TABLE

Disusun oleh :

Diva Octaviani

NIM : 2311102006

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.

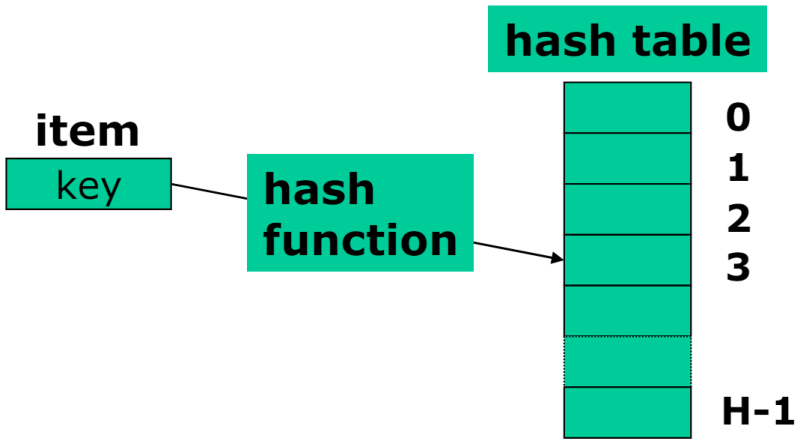
BAB II

DASAR TEORI

1. Pengertian Hash Table

Hash Table adalah sebuah array dengan sel-sel yang ukurannya telah ditentukan dan dapat berisi data atau *key* yang bersesuaian dengan data. Hashing adalah teknik untuk melakukan penambahan, penghapusan, dan pencarian dengan *constant average time*. Hashing digunakan untuk menyimpan data yang cukup besar pada ADT yang disebut Hash Table.

Ukuran Hash Table (H-size), biasanya lebih besar dari jumlah data yang hendak disimpan. Untuk setiap *key*, digunakan fungsi hash untuk memetakan *key* pada bilangan dalam rentang 0 hingga H-size. Fungsi hash memetakan elemen pada indeks dari Hash Table. Untuk menambahkan data atau pencarian, ditentukan *key* dari data tersebut dan digunakan sebuah fungsi hash untuk menetapkan lokasi untuk *ke*y tersebut.



2. 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.

3. Operasi Hash Table

a) **Insertion** : memasukkan data baru ke dalam Hash Table dengan memanggil fungsi hash untuk menentukan posisi bucket yang tepat, kemudian menambahkan data ke *bucket* tersebut.

b) **Deletion** : menghapus data dari Hash Table dengan mencari data menggunakan fungsi hash, kemudian menghapusnya dari *bucket* yang sesuai.

c) **Searching** : mencari data dalam Hash Table dengan memasukkan input kunci ke fungsi hash untuk menentukan posisi *bucket*, kemudian mencari data di dalam *bucket* yang sesuai.

d) **Update** : memperbarui data dalam Hash Table dengan mencari data menggunakan fungsi hash, kemudian memperbarui data yang ditemukan.

e) **Traversal** : melalui seluruh Hash Table untuk memproses semua data yang ada dalam tabel.

4. Collision Resolution

Collision Resolution merupakan penyelesaian masalah bila terjadi *collision* (tabrakan). Tabrakan terjadi jika dua buah *keys* dipetakan pada sebuah sel. *Collision* bisa terjadi saat melakukan *insertion*. Dibutuhkan prosedur tambahan untuk mengatasi terjadinya tabrakan. Ada dua strategi umum, yaitu Open Hashing (chaining) dan Closed Hashing (open addressing).

a) 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.

b) Closed Hashing (Open Addressing).

- **Linear Probing**

  Bila terjadi *collision,* cari posisi pertama pada tabel yang terdekat dengan posisi yang seharusnya. Linear Probing hanya disarankan untuk ukuran Hash Table yang ukurannya lebih besar dua kali dari jumlah data.

- **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;
```

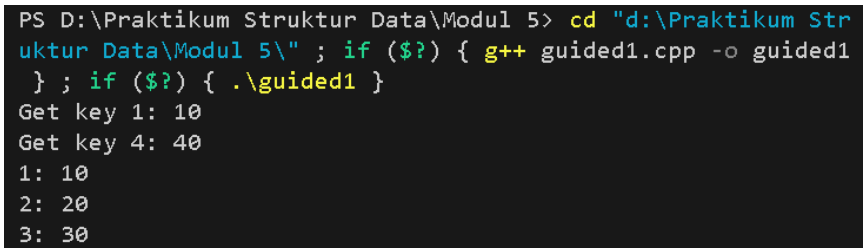
```
                }
                else
                {
                    prev->next = current->next;
                }
                delete current;
                return;
            }
            prev = current;
            current = current->next;
        }
    }

    // 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

Hash table direpresentasikan sebagai array dengan ukuran MAX_SIZE yang elemennya menunjuk ke linked list. Fungsi hash_func digunakan untuk memetakan kunci ke indeks dalam array. Setiap node dalam linked list menyimpan kunci (*key*) dan nilai (*value*) serta pointer ke node berikutnya. Dalam fungsi main, program ini menggunakan operasi untuk menyisipkan beberapa elemen, mencari nilai dengan kunci tertentu, menghapus elemen, dan menampilkan semua elemen yang tersimpan. Implementasi ini menggunakan teknik chaining untuk menangani kolisi, yaitu menyimpan elemen dengan indeks yang sama dalam linked list yang sama.

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;
            }
```

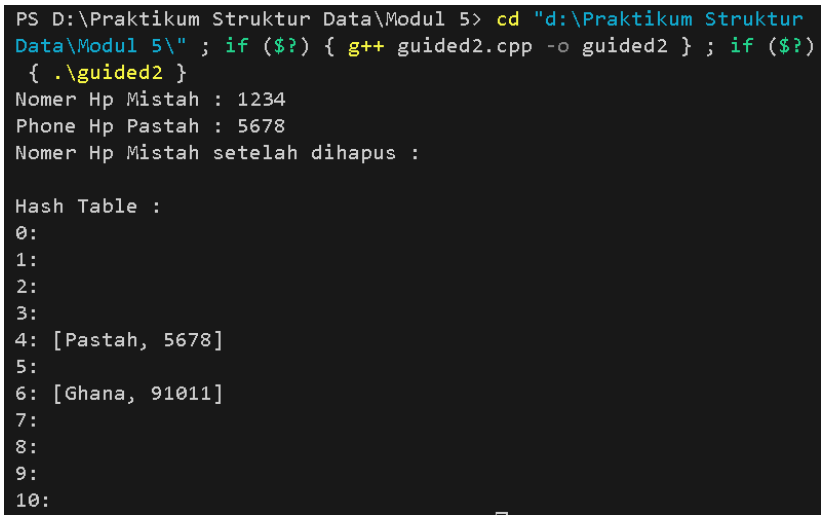
```
        }
    }

    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

Hash table direpresentasikan sebagai array dengan ukuran 11 yang elemennya berupa vektor dari objek HashNode. Fungsi hashFunc mengimplementasikan hash sederhana dengan menjumlahkan nilai ASCII karakter dalam string nama. Operasi insert menyisipkan atau memperbarui elemen, remove menghapus elemen berdasarkan nama, searchByName mencari nomor telepon berdasarkan nama, dan print menampilkan semua elemen dalam tabel. Dalam fungsi main, contoh penggunaan operasi-operasi tersebut ditunjukkan dengan menyisipkan, mencari, menghapus, dan menampilkan elemen dalam hash table.

LATIHAN KELAS – UNGUIDED

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 <string>
#include <vector>
using namespace std;
const int TABLE_SIZE = 11;

struct Student
{
    string nim;
    int nilai;
};

class HashNode
{
public:
    Student data;
```

```
    HashNode *next;

    HashNode(const Student &data) : data(data), next(nullptr) {}

    ~HashNode()
    {
        // Add cleanup code if necessary
    }
};

class HashTable
{
private:
    vector<HashNode *> table;

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

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

    ~HashTable()
    {
        for (int i = 0; i < TABLE_SIZE; ++i)
        {
            HashNode *entry = table[i];
            while (entry != nullptr)
```

```
            {
                HashNode *prev = entry;
                entry = entry->next;
                delete prev;
            }
        }
    }

    void insert(const Student &student)
    {
        int hashVal = hashFunc(student.nim);
        HashNode *prev = nullptr;
        HashNode *entry = table[hashVal];

        while (entry != nullptr && entry->data.nim != student.nim)
        {
            prev = entry;
            entry = entry->next;
        }

        if (entry == nullptr)
        {
            entry = new HashNode(student);

            if (prev == nullptr)
            {
                table[hashVal] = entry;
            }
            else
            {
                prev->next = entry;
            }
        }
```

```
        else
        {
            entry->data = student;
        }
    }

    void remove(const string &nim)
    {
        int hashVal = hashFunc(nim);
        HashNode *entry = table[hashVal];
        HashNode *prev = nullptr;

        while (entry != nullptr && entry->data.nim != nim)
        {
            prev = entry;
            entry = entry->next;
        }

        if (entry == nullptr)
        {
            cout << "Tidak ditemukan mahasiswa dengan NIM " <<
nim << endl;
        }
        else
        {
            if (prev == nullptr)
            {
                table[hashVal] = entry->next;
            }
            else
            {
                prev->next = entry->next;
            }
            delete entry;
```

```
            cout << "Mahasiswa dengan NIM " << nim << " telah
dihapus." << endl;
        }
    }

    Student *searchByNIM(const string &nim)
    {
        int hashVal = hashFunc(nim);
        HashNode *entry = table[hashVal];

        while (entry != nullptr)
        {
            if (entry->data.nim == nim)
            {
                return &(entry->data);
            }
            entry = entry->next;
        }
        return nullptr;
    }

    vector<Student *> searchByValue(int minValue, int maxValue)
    {
        vector<Student *> results;
        for (int i = 0; i < TABLE_SIZE; i++)
        {
            HashNode *entry = table[i];
            while (entry != nullptr)
            {
                if  (entry->data.nilai  >=  minValue  &&  entry-
>data.nilai <= maxValue)
                {
                    results.push_back(&(entry->data));
                }
```

```
                entry = entry->next;
            }
        }
        return results;
    }
};

void displayMenu()
{
    cout << "Menu:\n";
    cout << "1. Tambah data mahasiswa\n";
    cout << "2. Hapus data mahasiswa\n";
    cout << "3. Cari data mahasiswa berdasarkan NIM\n";
    cout << "4. Cari data mahasiswa berdasarkan rentang nilai
(80-90)\n";
    cout << "5. Keluar\n";
    cout << "Pilihan: ";
}

int main()
{
    HashTable hashTable;
    int choice;

    do
    {
        displayMenu();
        cin >> choice;

        switch (choice)
        {
        case 1:
        {
            string nim;
```

```
            int nilai;
            cout << "Masukkan NIM: ";
            cin >> nim;
            cout << "Masukkan nilai: ";
            cin >> nilai;
            Student student = {nim, nilai};
            hashTable.insert(student);
            break;
        }
        case 2:
        {
            string nim;
            cout << "Masukkan NIM mahasiswa yang akan dihapus:
";
            cin >> nim;
            hashTable.remove(nim);
            break;
        }
        case 3:
        {
            string nim;
            cout << "Masukkan NIM mahasiswa yang akan dicari: ";
            cin >> nim;
            Student *student = hashTable.searchByNIM(nim);
            if (student)
            {
                cout << "NIM: " << student->nim << ", Nilai: "
<< student->nilai << endl;
            }
            else
            {
                cout << "Mahasiswa tidak ditemukan." << endl;
            }
            break;
```

```
        }
        case 4:
        {
            int minValue = 80, maxValue = 90;
            vector<Student         *>         students         =
hashTable.searchByValue(minValue, maxValue);
            if (!students.empty())
            {
                cout  <<  "Mahasiswa  dengan  nilai  antara  "  <<
minValue << " dan " << maxValue << ":\n";
                for (Student *student : students)
                {
                    cout << "NIM: " << student->nim << ", Nilai:
" << student->nilai << endl;
                }
            }
            else
            {
                cout  <<  "Tidak  ada  mahasiswa  dengan  nilai
tersebut." << endl;
            }
            break;
        }
        case 5:
            cout << "Terima kasih telah menggunakan program ini."
<< endl;
            break;
        default:
            cout << "Pilihan tidak valid." << endl;
        }
        cout << endl;
    } while (choice != 5);
    return 0;
}
```

Screenshoot program

1. Tampilan menu

```
PS D:\Praktikum Struktur Data\Modul 5> cd "d:\Praktikum
Struktur Data\Modul 5\" ; if ($?) { g++ unguided.cpp -o
unguided } ; if ($?) { .\unguided }
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
Pilihan:
```

2. Menambah data mahasiswa

```
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
Pilihan: 1
Masukkan NIM: 2311102006
Masukkan nilai: 90
```

3. Mencari data mahasiswa berdasarkan NIM

```
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
Pilihan: 3
Masukkan NIM mahasiswa yang akan dicari: 2311102006
NIM: 2311102006, Nilai: 90
```

4. Mencari data mahasiswa berdasarkan rentang nilai (80-90)

```
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
Pilihan: 4
Mahasiswa dengan nilai antara 80 dan 90:
NIM: 2311102006, Nilai: 90
```

5. Menghapus data mahasiswa

```
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
Pilihan: 2
Masukkan NIM mahasiswa yang akan dihapus: 2311102006
Mahasiswa dengan NIM 2311102006 telah dihapus.
```

6. Keluar dari program

```
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
Pilihan: 5
Terima kasih telah menggunakan program ini.
```

Deskripsi program

Program ini mengimplementasikan struktur data hash table untuk menyimpan data mahasiswa yang terdiri dari NIM dan nilai. Hash table direpresentasikan sebagai vektor dari linked list. Program menyediakan operasi insert untuk menambah data baru, remove untuk menghapus data berdasarkan NIM, searchByNIM untuk mencari data berdasarkan NIM, dan searchByValue untuk mencari data berdasarkan rentang nilai (80-90). Fungsi hash digunakan untuk memetakan NIM ke indeks bucket dalam vektor, dan teknik chaining diterapkan untuk menangani kolisi. Program menampilkan menu pilihan yang memungkinkan pengguna berinteraksi dengan operasi-operasi tersebut, seperti menambah, menghapus, mencari data mahasiswa, atau keluar dari program.

BAB IV

KESIMPULAN

Hash Table merupakan sebuah struktur data yang efisien untuk menyimpan dan mengakses data dengan operasi seperti insertion, deletion, searching, updating, dan traversal. Fungsi hash berperan dalam memetakan kunci ke dalam rentang indeks array Hash Table. Setiap kunci memiliki lokasi tersendiri dalam Hash Table yang ditentukan oleh fungsi hash. Namun, terdapat kemungkinan terjadinya *collision* (tabrakan) ketika dua atau lebih kunci terpetakan ke indeks yang sama. Untuk mengatasinya, terdapat dua strategi utama yaitu open hashing (chaining) yang menyimpan data dengan indeks sama ke dalam linked list, dan closed hashing (open addressing) yang mencari lokasi alternatif dalam tabel dengan teknik seperti linear probing, quadratic probing, atau double hashing. Pemilihan strategi penanganan kolisi yang tepat memengaruhi kinerja dan efisiensi Hash Table dalam menyimpan dan mengakses data.

DAFTAR PUSTAKA

Asisten Praktikum, "*Modul 5 Hash Table*", Learning Management System, 2024.

Manurung, Ruli dan Ade Azurat. (2008). IKI 20100: Struktur Data & Algoritma Hash Table. Fasilkom UI.