

LAPORAN PRAKTIKUM

MODUL 5 HASH TABLE



Disusun oleh:
Nabila Shasya Sabrina
NIM: 2311102039

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. Mahasiswa mampu menjelaskan definisi dan konsep dari Hash Code
- b. Mahasiswa mampu menerapkan Hash Code kedalam pemrograman

BAB II

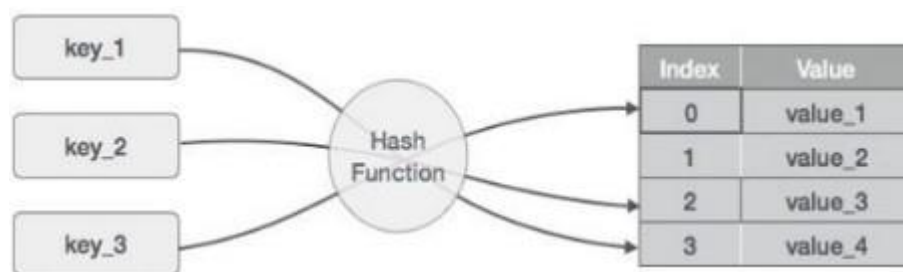
DASAR TEORI

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.

Linked list circular dapat digunakan untuk menyimpan data yang perlu diakses secara berulang, seperti daftar putar lagu, daftar pesan dalam antrian, atau penggunaan memori berulang dalam suatu aplikasi.

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:

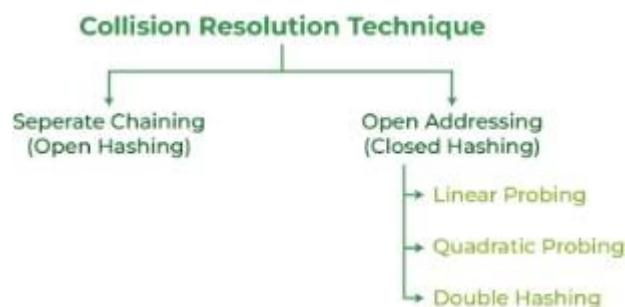
Memperbarui data dalam hash table dengan mencari data menggunakan fungsi hash, dan kemudian memperbarui data yang ditemukan.

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()
{

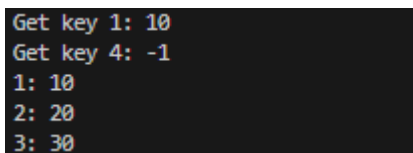
```

```

        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;
}

```

Screenshoot Program:



```

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

```

Deskripsi program:

Program tersebut merupakan implementasi dari *hash table*. *Hash table* digunakan untuk menyimpan pasangan kunci-nilai dengan memanfaatkan fungsi hash untuk menentukan lokasi penyimpanan.

1. Program ini mendefinisikan sebuah kelas `HashTable` yang memiliki fungsi untuk memasukkan data (`insert`), mencari data (`get`), menghapus data (`remove`), dan menampilkan seluruh data (`traverse`).
2. Fungsi hash sederhana (`hash_func`) digunakan untuk mengonversi kunci menjadi indeks dalam array.
3. Setiap node dalam tabel hash direpresentasikan oleh struktur `Node` yang menyimpan pasangan kunci-nilai dan referensi ke node berikutnya.
4. Implementasi `insert`, `get`, `remove`, dan `traverse` dilakukan di dalam kelas `HashTable` dengan menyesuaikan indeks sesuai dengan hasil fungsi hash.

5. Pada main program, beberapa operasi seperti memasukkan data, mencari data, menghapus data, dan menampilkan seluruh data dilakukan untuk menguji fungsionalitas dari tabel hash yang dibuat.

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 << "]\n";
            }
        }
        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:

```
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 tersebut adalah implementasi sederhana dari *hash table*. *Hash table* digunakan untuk menyimpan data dalam bentuk pasangan kunci-nilai, di mana kunci adalah nama dan nilainya adalah nomor telepon.

1. Program ini mendefinisikan dua kelas, yaitu `HashNode` yang merepresentasikan node dalam tabel hash, dan `HashMap` yang merupakan tabel hash itu sendiri.
2. Tabel hash direpresentasikan sebagai sebuah array dari vektor yang berisi pointer ke objek `HashNode`.
3. Terdapat fungsi hash (`hashFunc`) yang menghasilkan indeks dalam tabel hash berdasarkan kunci (nama).
4. Fungsi-fungsi seperti `insert`, `remove`, dan `searchByName` digunakan untuk menambah, menghapus, dan mencari data dalam tabel hash.
5. `print` digunakan untuk mencetak isi dari tabel hash.
6. Pada `main` program, beberapa operasi seperti memasukkan data, mencari data, menghapus data, dan mencetak isi tabel hash dilakukan untuk menguji fungsionalitas dari tabel hash yang dibuat.

BAB IV

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 <vector>
#include <string>

using namespace std;

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

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

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

public:
    void insert(string nim, int nilai)
    {
        int index = hashFunc(nim);
        table[index].push_back(new Mahasiswa(nim, nilai));
    }
};
```

```

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

Mahasiswa *searchByNIM(string nim)
{
    int index = hashFunc(nim);
    for (Mahasiswa *mhs : table[index])
    {
        if (mhs->NIM == nim)
        {
            return mhs;
        }
    }
    return nullptr;
}

vector<Mahasiswa *> searchByScoreRange(int minScore, int
maxScore)
{
    vector<Mahasiswa *> result;
    for (int i = 0; i < TABLE_SIZE; ++i)
    {
        for (Mahasiswa *mhs : table[i])
        {
            if (mhs->nilai >= minScore && mhs->nilai <= maxScore)
            {
                result.push_back(mhs);
            }
        }
    }
    return result;
}

};

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

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

```

```

cout << "Pilih: ";
cin >> choice;

switch (choice)
{
case 1:
    cout << "\nMasukkan NIM mahasiswa: ";
    cin >> nim;
    cout << "Masukkan nilai mahasiswa: ";
    cin >> nilai;
    hashTable.insert(nim, nilai);
    cout << "Data mahasiswa berhasil ditambahkan.\n";
    break;

case 2:
    cout << "\nMasukkan NIM mahasiswa yang ingin dihapus: ";
    cin >> nim;
    hashTable.remove(nim);
    cout << "Data mahasiswa dengan NIM " << nim << " berhasil
dihapus.\n";
    break;

case 3:
    cout << "\nMasukkan NIM mahasiswa yang ingin dicari: ";
    cin >> nim;
    {
        Mahasiswa *mhs = hashTable.searchByNIM(nim);
        if (mhs != nullptr)
        {
            cout << "Mahasiswa dengan NIM " << nim << "
ditemukan. Nilainya: " << mhs->nilai << endl;
        }
        else
        {
            cout << "Mahasiswa dengan NIM " << nim << " tidak
ditemukan.\n";
        }
    }
    break;

case 4:
    {
        int minScore = 80;
        int maxScore = 90;
        vector<Mahasiswa *> result =
hashTable.searchByScoreRange(minScore, maxScore);
        if (!result.empty())
        {
            cout << "\nMahasiswa dengan nilai antara 80 - 90:\n";
            for (Mahasiswa *mhs : result)
            {
                cout << "NIM: " << mhs->NIM << ", Nilai: " <<
mhs->nilai << endl;
            }
        }
        else
        {
            cout << "\nTidak ada mahasiswa dengan nilai antara
80 - 90.\n";
        }
    }
}

```

```

        break;

        case 5:
            cout << "Program selesai.\n";
            break;

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

return 0;
}

```

Screenshot Output:

```

Menu:
1. Tambah data mahasiswa
2. Hapus data mahasiswa
3. Cari mahasiswa berdasarkan NIM
4. Cari mahasiswa berdasarkan rentang nilai (80 - 90)
5. Keluar
Pilih: 1

```

```

Masukkan NIM mahasiswa: 2311102039
Masukkan nilai mahasiswa: 90
Data mahasiswa berhasil ditambahkan.

```

```

Menu:
1. Tambah data mahasiswa
2. Hapus data mahasiswa
3. Cari mahasiswa berdasarkan NIM
4. Cari mahasiswa berdasarkan rentang nilai (80 - 90)
5. Keluar
Pilih: 1

```

```

Masukkan NIM mahasiswa: 2311102055
Masukkan nilai mahasiswa: 88
Data mahasiswa berhasil ditambahkan.

```

```

Menu:
1. Tambah data mahasiswa
2. Hapus data mahasiswa
3. Cari mahasiswa berdasarkan NIM
4. Cari mahasiswa berdasarkan rentang nilai (80 - 90)
5. Keluar
Pilih: 1

```

```

Masukkan NIM mahasiswa: 2311102040
Masukkan nilai mahasiswa: 85
Data mahasiswa berhasil ditambahkan.

```

```

Pilih: 2

```

```

Masukkan NIM mahasiswa yang ingin dihapus: 2311102055
Data mahasiswa dengan NIM 2311102055 berhasil dihapus.

```

```

Menu:
1. Tambah data mahasiswa
2. Hapus data mahasiswa
3. Cari mahasiswa berdasarkan NIM
4. Cari mahasiswa berdasarkan rentang nilai (80 - 90)
5. Keluar
Pilih: 3

Masukkan NIM mahasiswa yang ingin dicari: 2311102039
Mahasiswa dengan NIM 2311102039 ditemukan. Nilainya: 90

Menu:
1. Tambah data mahasiswa
2. Hapus data mahasiswa
3. Cari mahasiswa berdasarkan NIM
4. Cari mahasiswa berdasarkan rentang nilai (80 - 90)
5. Keluar
Pilih: 3

Masukkan NIM mahasiswa yang ingin dicari: 2311101020
Mahasiswa dengan NIM 2311101020 tidak ditemukan.

```

```

Pilih: 4

Mahasiswa dengan nilai antara 80 - 90:
NIM: 2311102039, Nilai: 90
NIM: 2311102040, Nilai: 85

```

```

Pilih: 5
Program selesai.

```

Deskripsi Program:

Program di atas adalah implementasi dari hash table untuk menyimpan data mahasiswa. Setiap mahasiswa memiliki NIM dan nilai. Program ini menyediakan pilihan menu untuk menambahkan data baru, menghapus data, mencari data berdasarkan NIM, dan mencari data berdasarkan rentang nilai (80 - 90).

1. Struktur Mahasiswa memiliki atribut NIM dan nilai. Konstruktor Mahasiswa digunakan untuk inisialisasi nilai tersebut.
2. Kelas HashTable memiliki vektor sebagai array untuk menyimpan data mahasiswa. Setiap slot dalam vektor mewakili slot hash dalam tabel hash.
3. Fungsi hash (hashFunc) menghitung nilai hash dari sebuah NIM.
4. Fungsi insert digunakan untuk menambahkan data mahasiswa ke dalam tabel hash.
5. Fungsi remove digunakan untuk menghapus data mahasiswa dari tabel hash berdasarkan NIM.
6. Fungsi searchByNIM digunakan untuk mencari data mahasiswa berdasarkan NIM.
7. Fungsi searchByScoreRange digunakan untuk mencari data mahasiswa berdasarkan rentang nilai.
8. Di dalam main program, pengguna diberi pilihan menu untuk berbagai

operasi. Input pengguna digunakan untuk memilih operasi yang diinginkan, seperti menambahkan data, menghapus data, mencari data berdasarkan NIM, mencari data berdasarkan rentang nilai, atau keluar dari program.

BAB V

KESIMPULAN

Hash table adalah struktur data yang digunakan untuk menyimpan data dalam bentuk pasangan kunci-nilai, di mana setiap data memiliki kunci unik yang digunakan untuk mengakses nilainya. Hash table menggunakan fungsi hash untuk mengonversi kunci menjadi indeks dalam array, di mana data akan disimpan.

Proses penyimpanan data dalam hash table sangat efisien karena pencarian, penambahan, dan penghapusan data memiliki kompleksitas waktu yang konstan, asalkan tidak terjadi tabrakan (collision). Tabrakan terjadi ketika dua atau lebih kunci menghasilkan indeks yang sama. Untuk menangani tabrakan, hash table dapat menggunakan metode chaining atau open addressing.

Dalam chaining, setiap slot dalam array hash table memiliki daftar terkait (linked list) yang menampung semua data yang di-hash ke slot tersebut. Sedangkan dalam open addressing, jika terjadi tabrakan, maka pencarian dilakukan untuk menemukan slot kosong terdekat untuk menyimpan data.

Ukuran hash table harus dipilih dengan hati-hati. Ukuran yang terlalu kecil dapat meningkatkan kemungkinan terjadinya tabrakan, sedangkan ukuran yang terlalu besar dapat mengakibatkan pemborosan memori.

Hash table banyak digunakan dalam berbagai aplikasi seperti penyimpanan data pengguna, implementasi database, pencarian cepat, tabel hash, dan lain-lain. Dengan memahami konsep dan prinsip kerja hash table, kita dapat memilih dan mengimplementasikan struktur data ini secara efektif sesuai dengan kebutuhan aplikasi yang bersangkutan.

DAFTAR PUSTAKA

Asisten Praktikum. 2024. "*MODUL 5 HASH TABLE*". Learning Management System.