

LAPORAN PRAKTIKUM

MODUL 5

HASH TABLE



Disusun oleh:

Reza Alvonzo

NIM : 2311102026

Dosen Pengampu:

Wahyu Andi Saputra, S.Pd., M.Eng.

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS INFORMATIKA

INSTITUT TEKNOLOGI TELKOM 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

Apa itu Hash Table?

Hash table adalah struktur data yang digunakan untuk menyimpan pasangan kunci-nilai. Ini memungkinkan pencarian, penyisipan, dan penghapusan elemen dalam waktu konstan ($O(1)$) pada kasus rata-rata, asalkan fungsi hash yang baik digunakan.

Fungsi Hash

Fungsi hash adalah fungsi yang mengonversi kunci menjadi indeks dalam tabel. Tujuannya adalah untuk memetakan kunci ke dalam "wadah" (slot) yang tersedia dalam tabel. Fungsi hash yang baik akan mendistribusikan kunci secara merata di seluruh ruang indeks.

Collision

Collision terjadi ketika dua kunci dihash ke lokasi yang sama dalam tabel. Hal ini dapat diatasi dengan berbagai teknik, salah satunya adalah chaining, di mana setiap slot dalam tabel adalah linked list.

Setiap slot dalam tabel adalah sebuah linked list yang berisi pasangan kunci-nilai. Fungsi hash yang sederhana hanya melakukan modulus dengan ukuran tabel untuk mendapatkan indeks.

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;
        }
    }
}

```

```

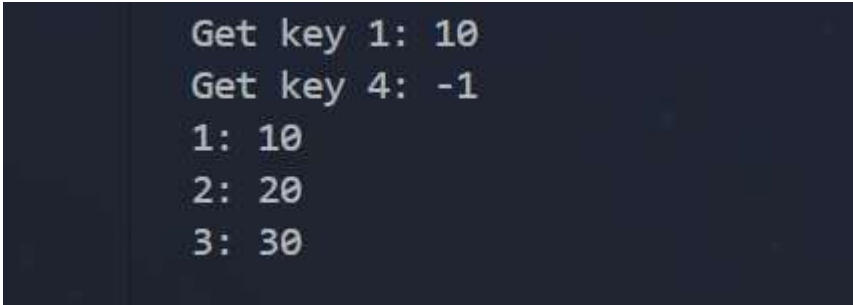
        prev = current;
        current = current->next;

    }
}
// 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

Implementasi ini menggunakan chaining untuk menangani collision, yaitu ketika dua kunci dihash ke lokasi yang sama dalam tabel. Masing-masing bucket dalam tabel adalah linked list yang menampung semua nilai yang dihash ke indeks 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)
```


[illegible]

```

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 ini menampilkan output yang menggambarkan operasi-operasi yang dilakukan pada linked list. Setiap operasi ditampilkan melalui fungsi tampil().

UNGUIDED

1. 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 <list>

using namespace std;

struct Mahasiswa {
    string nim;
    int nilai;
};

const int hashTableSize = 100;

class HashTable {
private:
    vector<list<Mahasiswa>> table;

    int hashFunction(const string& key) {
        int sum = 0;
        for (char c : key) {
```

```

        sum += c;
    }
    return sum % hashTableSize;
}

public:
    HashTable() {
        table.resize(hashTableSize);
    }

    void tambahData(const string& nim, int nilai) {
        Mahasiswa mahasiswa;
        mahasiswa.nim = nim;
        mahasiswa.nilai = nilai;
        int index = hashFunction(nim);
        table[index].push_back(mahasiswa);
    }

    void hapusData(const string& 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(const string& nim) {
        int index = hashFunction(nim);
        for (auto it = table[index].begin(); it !=
table[index].end(); ++it) {
            if ((*it).nim == nim) {

```

```

        cout << "Data ditemukan - NIM: " << (*it).nim <<
        ", Nilai: " << (*it).nilai << endl;
        return;
    }
}

cout << "Data tidak ditemukan" << endl;
}

void cariByRange() {
    for (int i = 0; i < hashTableSize; ++i) {
        for (auto it = table[i].begin(); it !=
table[i].end(); ++it) {
            if ((*it).nilai >= 80 && (*it).nilai <= 90) {
                cout << "NIM: " << (*it).nim << ", Nilai: "
<< (*it).nilai << endl;
            }
        }
    }
}

};

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

    do {
        cout << "\nMenu:\n";
        cout << "1. Menambah Data\n";
        cout << "2. Menghapus Data\n";
        cout << "3. Cari mahasiswa menggunakan NIM\n";
        cout << "4. Cari Nilai (80 - 90)\n";
        cout << "Masukkan Pilihan: ";
        cin >> choice;
    }
}

```

```

switch (choice) {
    case 1:
        cout << "Masukkan NIM: ";
        cin >> nim;
        cout << "Masukkan Nilai: ";
        cin >> nilai;
        hashTable.tambahData(nim, nilai);
        break;
    case 2:
        cout << "Masukkan NIM untuk dihapus: ";
        cin >> nim;
        hashTable.hapusData(nim);
        break;
    case 3:
        cout << "Masukkan NIM yang ingin dicari: ";
        cin >> nim;
        hashTable.cariByNIM(nim);
        break;
    case 4:
        cout << "Mahasiswa dengan nilai antara 80 -
90:\n";

        hashTable.cariByRange();
        break;
    default:
        cout << "Pilihan tidak ada dalam Menu.\n";
}
} while (choice != 4);

return 0;
}

```


SCREENSHOOT PROGRAM

```
Menu:
1. Menambah Data
2. Menghapus Data
3. Cari mahasiswa menggunakan NIM
4. Cari Nilai (80 - 90)
Masukkan Pilihan: 1
Masukkan NIM: 2311102026
Masukkan Nilai: 89

Menu:
1. Menambah Data
2. Menghapus Data
3. Cari mahasiswa menggunakan NIM
4. Cari Nilai (80 - 90)
Masukkan Pilihan: 1
Masukkan NIM: 2311102027
Masukkan Nilai: 91

Menu:
1. Menambah Data
2. Menghapus Data
3. Cari mahasiswa menggunakan NIM
4. Cari Nilai (80 - 90)
Masukkan Pilihan: 2
Masukkan NIM untuk dihapus: 2311102027
```

Program menambahkan data dan menghapus data dari NIM yang dimasukkan

```
Menu:
1. Menambah Data
2. Menghapus Data
3. Cari mahasiswa menggunakan NIM
4. Cari Nilai (80 - 90)
Masukkan Pilihan: 3
Masukkan NIM yang ingin dicari: 2311102026
Data ditemukan - NIM: 2311102026, Nilai: 89
```

```
Menu:
1. Menambah Data
2. Menghapus Data
3. Cari mahasiswa menggunakan NIM
4. Cari Nilai (80 - 90)
Masukkan Pilihan: 4
Mahasiswa dengan nilai antara 80 - 90:
NIM: 2311102026, Nilai: 89
```

Program mencari dari NIM yang dimasukkan dan berdasarkan nilai 80 sampai 90

DESKRIPSI PROGRAM

Program utama menggunakan sebuah loop do-while untuk memungkinkan pengguna memilih opsi dari menu yang tersedia. Opsi-opsi tersebut meliputi menambahkan data, menghapus data, mencari data berdasarkan NIM, dan mencari mahasiswa dengan nilai antara 80-90. Setiap opsi memanggil metode yang sesuai dalam kelas HashTable untuk melakukan tugas yang diperlukan.

BAB IV

KESIMPULAN

Setelah melakukan pembelajaran mengenai tipe data di Bahasa Pemrograman C++ berikut poin utama yang telah dipelajari :

1. Kecepatan Akses:

Hash table memungkinkan pencarian, penyisipan, dan penghapusan elemen dalam waktu konstan ($O(1)$) pada kasus rata-rata, asalkan fungsi hash yang baik digunakan. Ini menjadikannya struktur data yang sangat efisien untuk operasi-operasi tersebut.

2. Penanganan Collision:

Collision terjadi ketika dua atau lebih kunci dihash ke lokasi yang sama dalam tabel. Dalam C++, collision biasanya ditangani dengan teknik chaining, di mana setiap slot dalam tabel adalah linked list yang menyimpan semua elemen yang dihash ke indeks yang sama.

3. Implementasi:

Hash table dapat diimplementasikan menggunakan array dari linked list atau vektor dari linked list. Setiap elemen di tabel hash berisi pasangan kunci-nilai, dan pencarian dilakukan berdasarkan kunci.

DAFTAR PUSTAKA

Philip, Wheeler. (2014, 16 Juni) Laporan Resmi Hash. diakses pada 13 Mei 2024 dari <https://www.scribd.com/doc/229952345/Laporan-resmi-Hash>