LAPORAN PRAKTIKUM STRUKTUR DATA DAN ALGORITMA

MODUL V
HASH TABLE

Disusun Oleh :

NAMA : WISNU RANANTA RADITYA PUTRA

NIM : 2311102013

Dosen :

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

PROGRAM STUDI S1 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

1) 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.

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

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.

4) 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

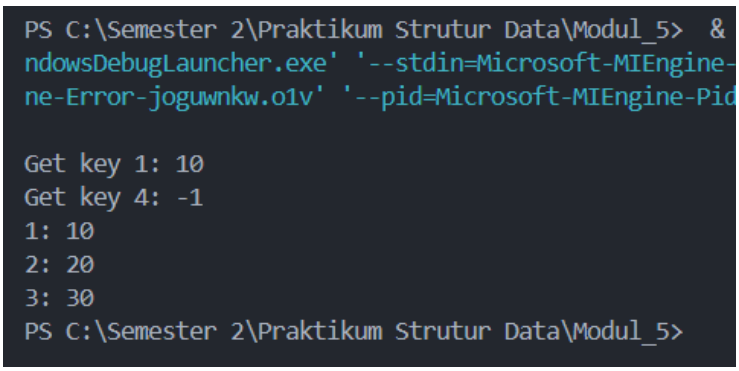
Guided 1

```
#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;
}
```

Screenshots Output



Deskripsi:

Program diatas merupakan implementasi Hash Table. Program ini menggunakan metode chaining untuk mengatasi collision. Program diatas memiliki fungsi dasar seperti insert untuk menambah data baru, get untuk mencari nilai berdasarkan *key*, remove untuk menghapus data berdasarkan *key*, dan traverse untuk menampilkan semua data yang tersimpan dalam hash table. Pada fungsi main digunakan untuk menjalankan semua fungsi diatas.

Guided 2

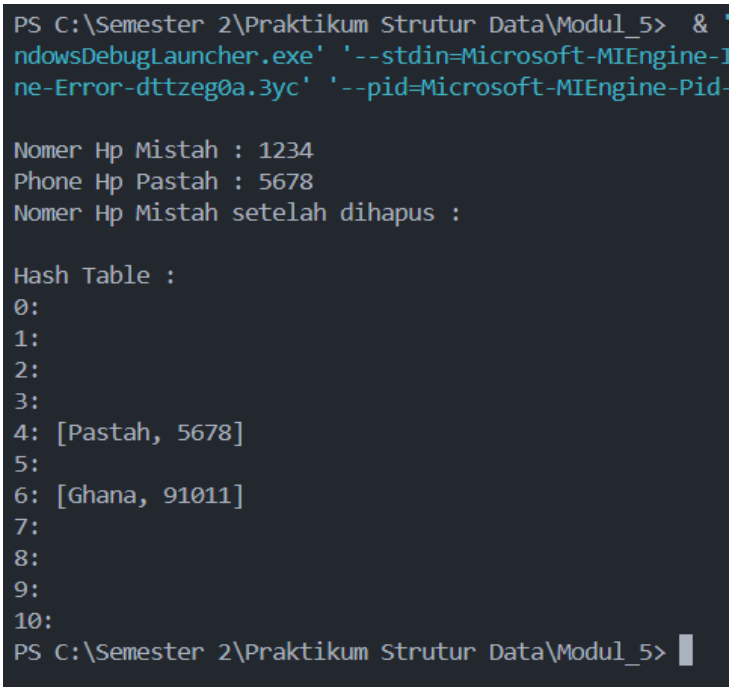
```
#include <iostream>
#include <string>
#include <vector>
using namespace std;
const int TABLE_SIZE = 11;
string name;
string phone_number;
class HashNode
{
public:
    string name;
    string phone_number;
    HashNode(string name, string phone_number)
    {
        this->name = name;
        this->phone_number = 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;
}
```

Screenshots Output



Deskripsi

Program di atas merupakan implementasi Hash table. Program ini juga menggunakan metode chaining untuk mengatasi collision. Program ini digunakan untuk menyimpan nama dan nomor telepon, dengan menggunakan fungsi hash untuk menentukan lokasi penyimpanan data di dalam array. Struktur data yang digunakan adalah vektor dari pointer ke HashNode, di mana setiap HashNode berisi nama dan nomor telepon. Program ini memiliki fungsi insert untuk menambahkan data baru, remove untuk menghapus data berdasarkan nama, dan searchByName untuk mencari nomor telepon berdasarkan nama. Selain itu, terdapat juga fungsi print untuk menampilkan seluruh isi tabel hash. Fungsi main digunakan untuk menjalankan fungsi diatas.

BAB IV
UNGUIDED

Unguided 1

```
#include <iostream>
#include <unordered_map>
#include <vector>

using namespace std;

struct Mahasiswa {
    string nim_2311102013;
    string nama_wrrp;
    int nilai;
};

class HashTable {
private:
    unordered_map<string, Mahasiswa> data;

public:
    void tambahData(const Mahasiswa& mhs) {
        data[mhs.nim_2311102013] = mhs;
    }

    void hapusData(const string& nim_2311102013) {
        data.erase(nim_2311102013);
    }

    Mahasiswa* cariBynim_2311102013(const string&
nim_2311102013) {
        if (data.find(nim_2311102013) != data.end()) {
            return &data[nim_2311102013];
        }
        return nullptr;
    }

    vector<Mahasiswa> cariByNilai() {
        vector<Mahasiswa> result;
        for (auto& pair : data) {
            if (pair.second.nilai >= 80 && pair.second.nilai <=
90) {
                result.push_back(pair.second);
            }
        }
        return result;
    }
};

void tampilkanMenu() {
    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 << "Pilih : ";
}

int main() {
    HashTable hashTable;
    int pilihan;
    string nim_2311102013;
    string nama_wrrp;
    int nilai;

    do {
        tampilkanMenu();
        cin >> pilihan;

        switch (pilihan) {
            case 1: {
                Mahasiswa mhs;
                cout << "Masukkan NIM : ";
                cin >> mhs.nim_2311102013;
                cout << "Masukkan Nama : ";
                cin.ignore();
                getline(cin, mhs.nama_wrrp);
                cout << "Masukkan nilai : ";
                cin >> mhs.nilai;
                hashTable.tambahData(mhs);
                break;
            }
            case 2: {
                cout << "Masukkan NIM mahasiswa yang akan
dihapus : ";
                cin >> nim_2311102013;
                hashTable.hapusData(nim_2311102013);
                break;
            }
            case 3: {
                cout << "Masukkan NIM mahasiswa yang akan
dicari : ";
                cin >> nim_2311102013;
                Mahasiswa* mhs =
hashTable.cariBynim_2311102013(nim_2311102013);
                if (mhs != nullptr) {
                    cout << "nim_2311102013 : " << mhs-
>nim_2311102013 << ", Nama : " << mhs->nama_wrrp << ", Nilai :
" << mhs->nilai << endl;
                } else {
                    cout << "Mahasiswa dengan NIM tersebut
tidak ditemukan!\n";
                }
                break;
            }
            case 4: {
                vector<Mahasiswa> result =
```

```
hashTable.cariByNilai();
                if (result.empty()) {
                    cout << "Tidak ada mahasiswa dengan nilai
di rentang (80 - 90)!\n";
                } else {
                    cout << "Mahasiswa dengan nilai di rentang
(80 - 90) :\n";
                    for (const Mahasiswa& mhs : result) {
                        cout << "NIM : " << mhs.nim_2311102013
<< ", Nama : " << mhs.nama_wrrp << ", Nilai : " << mhs.nilai <<
endl;
                    }
                }
                break;
            }
            case 5:
                cout << "Terima kasih!\n";
                break;
            default:
                cout << "Pilihan tidak valid. Silakan pilih
kembali!\n";
        }

    } while (pilihan != 5);

    return 0;
}
```

Screenshots Output:

a. Setiap Mahasiswa memiliki NIM dan nilai

```
Masukkan NIM : 2311102013
Masukkan Nama : Wisnu Rananta Raditya Putra
Masukkan nilai : 95
```

b. Program memiliki tampilan pilihan menu berisi poin C

```
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
```

c. Implementasikan fungsi untuk menambahkan data baru, menghapus data, mencari data berdasarkan NIM, dan mencari data berdasarkan rentang nilai (80 – 90).

- Menambahkan data

```
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
Pilih : 1
Masukkan NIM : 2311102013
Masukkan Nama : Wisnu Rananta Raditya Putra
Masukkan nilai : 95
Data Berhasil Ditambahkan!
```

- Menghapus data

```
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
Pilih : 2
Masukkan NIM mahasiswa yang akan dihapus : 2311102099
Data Berhasil Dihapus!
```

- Mencari data 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
Pilih : 3
Masukkan NIM mahasiswa yang akan dicari : 2311102022
nim 2311102013 : 2311102022, Nama : Jason, Nilai : 90
```

- Mencari data 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
Pilih : 4
Mahasiswa dengan nilai di rentang (80 - 90) :
NIM : 2311102013, Nama : Wisnu Rananta Raditya Putra, Nilai : 87
NIM : 2311102022, Nama : Jason, Nilai : 90
NIM : 2311102100, Nama : Kendrick Lamar, Nilai : 89
```

Deskripsi:

Program di atas merupakan program implementasi menu mahasiswa menggunakan hash table. Program diatas menggunakan metode chaining untuk menangani collisions. Class Hash table menggunakan ‘unordered_map’ untuk menyimpan data mahasiswa dengan NIM menggunakan key. Fungsi yang tersedia pada program diatas yaitu tambahData untuk menambah data, hapusData untuk menghapus data mahasiswa dengan NIM, cariBynim_2311102013 untuk mencari data mahasiswa dengan NIM, cariByNilai digunakan untuk mencari mahasiswa dengan nilai rentang 80-90, sementara pada fungsi main digunakan untuk menjalankan menu yang didalamnya berisi case fungsi-fungsi di atas.

BAB V

KESIMPULAN

Kesimpulan yang dapat saya ambil, Hash Table adalah struktur data yang menyediakan penyimpanan dan pengambilan data dengan efisiensi tinggi menggunakan teknik hashing. Hash table menggunakan fungsi hash untuk memetakan kunci (key) ke indeks dalam sebuah array. Ini memungkinkan akses data yang sangat cepat, umumnya O(1) dalam waktu rata-rata. Fungsi hash mengubah key menjadi indeks array, dan data disimpan di lokasi array tersebut. Fungsi hash yang baik mendistribusikan key secara merata ke seluruh tabel untuk mengurangi kemungkinan terjadinya collision. Collisions terjadi ketika dua key berbeda di-hash ke indeks yang sama. Teknik yang digunakan untuk menangani collisions adalah teknik chaining.

BAB VI

DAFTAR PUSTAKA

[1] Asisten Praktikum. 2024. Modul 5 “Hash Table”. Diakses 13 Mei 2024, 19:00 WIB. https://lms.ittelkom-pwt.ac.id/