

**LAPORAN PRAKTIKUM STRUKTUR
DATA DAN ALGORITMA**

**MODUL V
HASH TABLE**



Disusun Oleh :

NAMA : GALIH TRISNA
NIM : 2311102050

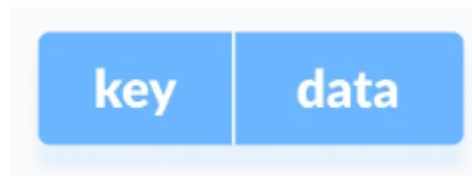
Dosen

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

**PROGRAM STUDI S1 TEKNIK INFORMATIKA
FAKULTAS INFROMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
2024**

A. Dasar Teori

Struktur data Hash Table adalah struktur data yang digunakan untuk menyimpan dan mengelola data dengan cepat dan efisien. Ini beroperasi dengan prinsip kunci-nilai, di mana setiap elemen data memiliki kunci yang unik yang digunakan untuk mengakses atau memanipulasinya. Hash Table (Tabel Hash) adalah struktur data yang mengorganisir data ke dalam pasangan kunci-nilai. Ini menggunakan fungsi hash untuk mengonversi kunci menjadi indeks dalam array. Dengan cara ini, akses ke data menjadi sangat cepat karena kita dapat langsung menghitung indeks tempat data disimpan. Ini cocok untuk pencarian, penyisipan, penghapusan, dan pembaruan data dalam waktu konstan, asalkan tidak ada konflik dalam fungsi hash (collision).



Kegunaan Struktur Data Hash Table adalah sebagai berikut :

1. Pencarian Cepat: Dapat digunakan untuk mencari data dengan cepat berdasarkan key. Ini sangat berguna dalam aplikasi seperti basis data, kamus, dan cache.
2. Penyimpanan Data: Hash table dapat digunakan untuk menyimpan data dengan efisien. Data dapat diambil dan dimasukkan ke dalam tabel dengan waktu konstan, asalkan tidak ada collision yang signifikan.
3. Implementasi Struktur Data Lain: Hash table dapat digunakan untuk mengimplementasikan struktur data lain, seperti set dan map.

Teknik Hash Table adalah cara atau strategi yang digunakan dalam mengimplementasikan fungsi hash dan menangani tabrakan hash dalam struktur data Hash Table. Berikut adalah beberapa teknik Hash Table yang umum digunakan:

1. Chaining : Metode penanganan tabrakan ini melibatkan penggunaan struktur data seperti linked list atau array pada setiap slot tabel hash. Ketika terjadi tabrakan, data baru akan disisipkan dalam linked list atau array yang terkait dengan slot yang bersangkutan. Chaining memungkinkan penyimpanan beberapa nilai dengan indeks yang sama, dan ini adalah salah satu metode penanganan tabrakan yang paling umum dan fleksibel.

2. Closed Hashing

a. 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.

b. Quadratic Probing

Penanganannya hampir sama dengan metode linear, hanya lompatannya tidak satu-satu, tetapi quadratic (12, 22, 32, 42, ...)

c. Double Hashing

Pada saat terjadi collision, terdapat fungsi hash yang kedua untuk menentukan posisinya kembali

B. 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;
}

```

```

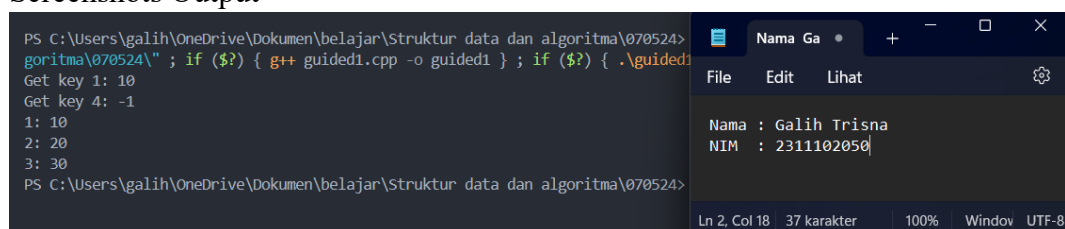
// Deletion
ht.remove(4);

// Traversal
ht.traverse();

return 0;
}

```

Screenshots Output



```

PS C:\Users\galih\OneDrive\Dokumen\belajar\Struktur data dan algoritma\070524> g++ guided1.cpp -o guided1 ; if ($?) { .\guided1 }
Get key 1: 10
Get key 4: -1
1: 10
2: 20
3: 30
PS C:\Users\galih\OneDrive\Dokumen\belajar\Struktur data dan algoritma\070524>

```

Deskripsi: Program di atas menggunakan struktur data hash table yang disimpan dalam single linked list, di mana setiap node dalam list menyimpan key (sebagai indeks), value (data/nilai yang disimpan), dan pointer yang menunjuk ke data selanjutnya. Pointer digunakan untuk menangani bentrokan data (collision) jika terjadi, yang merupakan solusi dalam implementasi hash table. Fungsi hash (hash function) dalam program ini yaitu dengan mendapatkan sisa bagi key dengan ukuran maksimum hash table, yang dalam program ini adalah 10. Program ini terdapat beberapa fungsi untuk memanipulasi data dalam hash table, yaitu insert, get, remove, dan traverse.

1. Insert: Fungsi ini digunakan untuk memasukkan data ke dalam hash table. Jika indeks dari key yang dimasukkan sudah terisi oleh data lain, maka program akan mengubah nilai data tersebut menjadi data yang akan dimasukkan.
2. Get: Fungsi ini digunakan untuk mendapatkan nilai (value) berdasarkan key yang diminta. Jika tidak ada data yang memiliki key yang sama, maka nilai -1 akan dikembalikan, menunjukkan bahwa key yang dicari tidak ditemukan.
3. Remove: Fungsi ini digunakan untuk menghapus data atau node berdasarkan key yang diberikan.
4. Traverse: Fungsi ini digunakan untuk menampilkan isi dari hash table.

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 << "]\n";
            }
        }
    }
}

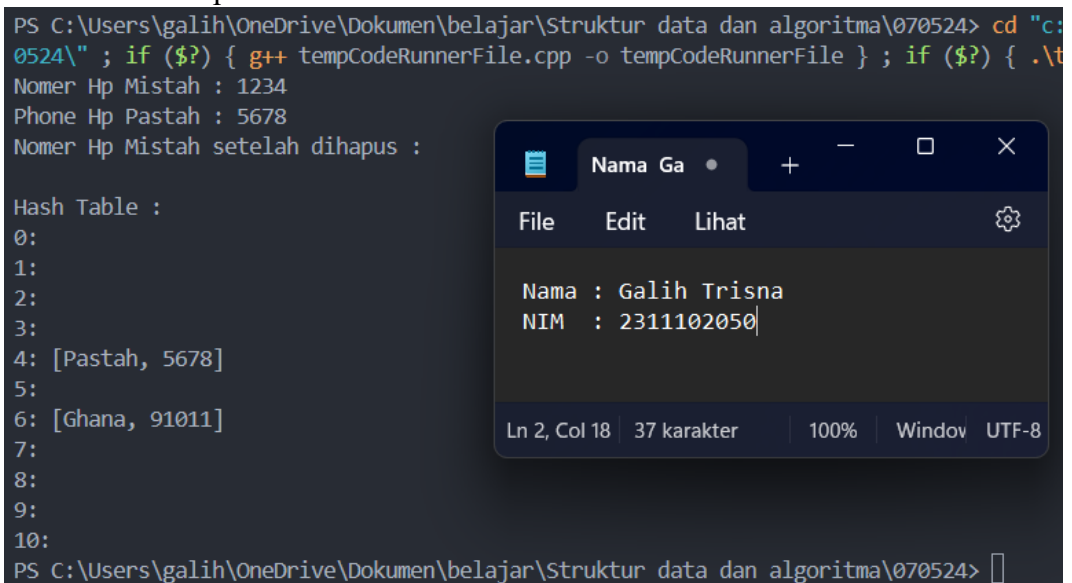
```

```

    }
    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;
}

```

Screenshots Output



PS C:\Users\galih\OneDrive\Dokumen\belajar\Struktur data dan algoritma\070524> cd "c:\070524\" ; if (\$?) { g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile } ; if (\$?) { .\t
 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:
 PS C:\Users\galih\OneDrive\Dokumen\belajar\Struktur data dan algoritma\070524>

Nama : Galih Trisna
 NIM : 2311102050

Deskripsi: Program diatas mengimplementasikan penggunaan Hash Table. Masih sama seperti program sebelumnya, namun pada program ini memiliki

perbedaan pada tempat penyimpanan data. Pada program ini menggunakan vector. Vector hampir sama seperti array, perbedaan nya adalah jika vector bersifat dinamis. Data pada vector berbentuk object dengan nama hash node. Pada program ini has node terdapat 2 data yaitu nama dan nomer handphone. Hash pada program ini dijalankan dengan menjumlahkan nilai dari key lalu dibagi dengan ukuran hash table dan diambil sisa hasil bagianya. Berikut adalah beberapa fungsi yang ada di dalam program :

1. hashFunc: berfungsi untuk menghasilkan nilai hash dari kunci (key).
2. insert: digunakan untuk memasukkan data baru.
3. remove: digunakan untuk menghapus data dengan cara mencari berdasarkan nama.
4. searchByName: berguna untuk mencari nomor handphone berdasarkan nama. Proses pencarian dilakukan dengan menggunakan algoritma sequential search, yaitu dengan mengecek satu per satu data secara berurutan dari awal hingga akhir atau hingga data yang sesuai ditemukan. Jika tidak ditemukan, maka program akan mengembalikan string kosong.
5. print: digunakan untuk menampilkan isi dari hash table.

C. Unguided/Tugas

Unguided 1

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;
const int TABLE_SIZE = 11;
string nim;
int nilai;
class HashNode
{
public:
    string nim;
    int nilai;
    HashNode(string nim, int nilai)
    {
        this->nim = nim;
        this->nilai = nilai;
    }
};
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 nim, int nilai)
    {
        int hash_val = hashFunc(nim);
        for (auto node : table[hash_val])
        {
            if (node->nim == nim)
            {

```

```

        node->nilai = nilai;
        return;
    }
}
table[hash_val].push_back(new HashNode(nim,
                                         nilai));
}
void remove(string nim)
{
    int hash_val = hashFunc(nim);

    for (auto it = table[hash_val].begin(); it !=
table[hash_val].end(); it++)
    {
        if ((*it)->nim == nim)
        {
            table[hash_val].erase(it);
            return;
        }
        else
        {
            cout << "Tidak Ditemukan" << endl;
        }
    }
}

int searchByNim(string nim)
{
    int hash_val = hashFunc(nim);
    for (auto node : table[hash_val])
    {
        if (node->nim == nim)
        {
            // return node->nilai;
            cout << "\nMahasiswa dengan NIM " << node->nim
<< " memiliki nilai " << node->nilai << endl;
        }
    }
    return 0;
}
int searchByNilai(int minNilai, int maxNilai)
{
    for (const auto &bucket : table)
    {

```

```

        for (auto node : bucket)
        {
            if (node->nilai >= minNilai && node->nilai <=
maxNilai)
            {
                cout << "[ NIM : " << node->nim << ", NILAI
: " << node->nilai << " ]" << endl;
            }
        }
    }
    return 0;
}

void print()
{
    for (int i = 0; i < TABLE_SIZE; i++)
    {
        cout << i << ": ";
        for (auto pair : table[i])
        {
            if (pair != nullptr)
            {
                cout << "[ NIM : " << pair->nim << ", NILAI
: " << pair->nilai << " ]";
            }
        }
        cout << endl;
    }
}

};

int main()
{
    HashMap mahasiswa_map;
    bool menu = true;
    int choice;
    do
    {

        cout << "===== # Data Mahasiswa #
===== " << endl;
        mahasiswa_map.print();
        cout << "===== # ===== #
===== " << endl;
        cout << "1. Tambah Data" << endl;
    }
}

```

```

cout << "2. Hapus Data" << endl;
cout << "3. Cari berdasarkan NIM" << endl;
cout << "4. Cari berdasarkan Nilai" << endl;
cout << "0. Keluar" << endl;
cout << "Pilihan Anda: ";
cin >> choice;
switch (choice)
{
case 1:
    cout << "Masukkan NIM: ";
    cin >> nim;
    cout << "Masukkan Nilai: ";
    cin >> nilai;

    mahasiswa_map.insert(nim, nilai);
    break;
case 2:
    cout << "Masukkan NIM: ";
    cin >> nim;
    mahasiswa_map.remove(nim);
    break;
case 3:
    cout << "Masukkan NIM: ";
    cin >> nim;
    mahasiswa_map.searchByNim(nim);

    break;
case 4:
    int maxNilai, minNilai;
    cout << "Masukkan Nilai Tertinggi: ";
    cin >> maxNilai;
    cout << "Masukkan Nilai Terendah: ";
    cin >> minNilai;
    mahasiswa_map.searchByNilai(minNilai, maxNilai);
    break;
case 0:
    menu = false;
    break;

default:
    break;
}

} while (menu == true);

```

```

    return 0;
}

```

Screenshots Output

- Menu

```

PS C:\Users\galih\OneDrive\Dokumen\belajar\Struktur data dan algoritma\070524> cd "c:\Users\galih\OneDrive\Dokumen\070524\" ; if ($?) { g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
===== # Data Mahasiswa # =====
0:
1:
2:
3:
4:
5:
6:
7:
8:
9:
10:
----- # ----- # -----
1. Tambah Data
2. Hapus Data
3. Cari berdasarkan NIM
4. Cari berdasarkan Nilai
0. Keluar
Pilihan Anda: 

```

- Tambah Data

```

===== # ----- # -----
1. Tambah Data
2. Hapus Data
3. Cari berdasarkan NIM
4. Cari berdasarkan Nilai
0. Keluar
Pilihan Anda: 1
Masukkan NIM: 23100000
Masukkan Nilai: 80
===== # Data Mahasiswa # =====
0: [ NIM : 2311102050, NILAI : 98 ]
1:
2:
3:
4:
5: [ NIM : 23100000, NILAI : 80 ]
6:
7:
8:
9:
10:
===== # ----- # -----

```


- Hapus Data

```

===== # ===== # =====
1. Tambah Data
2. Hapus Data
3. Cari berdasarkan NIM
4. Cari berdasarkan Nilai
0. Keluar
Pilihan Anda: 2
Masukkan NIM: 23100000
===== # Data Mahasiswa # =====
0: [ NIM : 2311102050, NILAI : 98 ]
1:
2:
3:
4:
5:
6:
7:
8:
9:
10:
===== # ===== # =====

```

- Mencari data berdasarkan NIM

```

===== # ===== # =====
1. Tambah Data
2. Hapus Data
3. Cari berdasarkan NIM
4. Cari berdasarkan Nilai
0. Keluar
Pilihan Anda: 3
Masukkan NIM: 2311102050

Mahasiswa dengan NIM 2311102050 memiliki nilai 98

```

- Mencari data berdasarkan rentang nilai (90 – 100).

```

===== # ===== # =====
1. Tambah Data
2. Hapus Data
3. Cari berdasarkan NIM
4. Cari berdasarkan Nilai
0. Keluar
Pilihan Anda: 4
Masukkan Nilai Tertinggi: 100
Masukkan Nilai Terendah: 90
[ NIM : 2311102050, NILAI : 98 ]

```

- Deskripsi

Program di atas merupakan sebuah program sederhana untuk menyimpan data nilai mahasiswa berdasarkan NIM. Program ini mengimplementasikan struktur data hash table untuk menyimpan dan mengelola data mahasiswa. Pertama-tama, program mendefinisikan class HashNode. Setiap node memiliki dua atribut, yaitu nim untuk menyimpan NIM mahasiswa (sebagai key) dan nilai untuk menyimpan nilai mahasiswa. Kemudian, terdapat class HashMap, yang merupakan class utama yang merepresentasikan hash table. class ini memiliki array vektor table yang digunakan untuk menyimpan node-node hash table. fungsi hashFunc digunakan untuk menghitung nilai hash dari NIM mahasiswa. Selanjutnya, terdapat fungsi insert untuk menambahkan data mahasiswa ke dalam hash table. Jika terjadi collision, data akan disimpan dalam bentuk chaining di dalam vektor pada indeks yang sama dalam array table. fungsi remove digunakan untuk menghapus

data mahasiswa berdasarkan NIM yang diberikan. fungsi `searchByNim` digunakan untuk mencari dan menampilkan nilai mahasiswa berdasarkan NIM. Terakhir, terdapat fungsi `searchByNilai` yang mencari dan menampilkan data mahasiswa berdasarkan rentang nilai tertentu. Di dalam fungsi `main`, terdapat sebuah loop `do-while` yang menampilkan menu kepada pengguna. Pengguna diberikan pilihan untuk menambah, menghapus, atau mencari data mahasiswa berdasarkan NIM atau rentang nilai. Program akan terus berjalan hingga pengguna memilih opsi untuk keluar.

D. Kesimpulan

Hash table adalah struktur data yang efisien untuk menyimpan dan mengakses data dengan cepat berdasarkan key. Melalui penggunaan fungsi hash, data dapat diindeks ke dalam array dengan cara yang meminimalkan konflik. Teknik chaining dapat digunakan untuk menangani konflik saat dua key menghasilkan nilai hash yang sama, memungkinkan penyimpanan data secara terstruktur dalam bucket yang terpisah. Dengan demikian, hash table merupakan alat yang sangat berguna dalam pemrosesan data yang cepat dan efisien.

E. Referensi

Rahmawati, Rini. Hash Table: Pengertian, Fungsi dan Cara Membuat. Diakses pada 10 Mei 2024, dari <https://dosenit.com/kuliah-it/hash-table>

Asisten Praktikum. (2024). Modul V : Hash Table

Programiz. (n.d.). Hash Table. Diakses pada 10 Mei 2024, dari <https://www.programiz.com/dsa/hash-table>

Annisa. (2023). Struktur Data Hash Table: Pengertian, Cara Kerja, dan Operasi Hash Table. Diakses pada 10 Mei 2024, dari <https://fikti.umsu.ac.id/struktur-data-hash-table-pengertian-cara-kerja-dan-operasi-hash-table/>