LAPORAN PRAKTIKUM

MODUL V
HASH TABLE

Disusun oleh:
Syafanida Khakiki
NIM: 2311102005

Dosen Pengampu:
Muhammad Afrizal Amrustian, S. Kom., M. Kom

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
PURWOKERTO
2023

BAB I

TUJUAN PRAKTIKUM

1. Mahasiswa mampu menjelaskan definisi dan konsep dari Hash Code
2. Mahasiswa mampu menerapkan Hash Code kedalam pemrograman

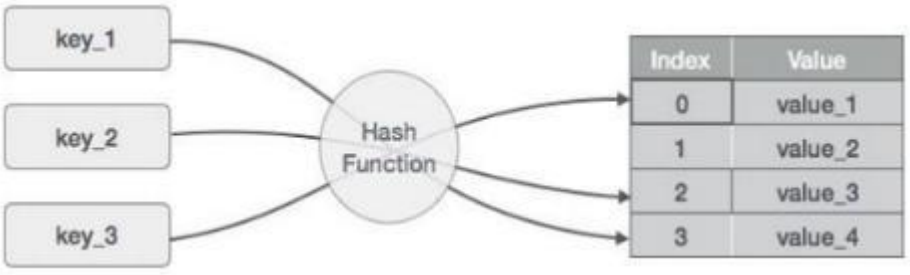
BAB II
DASAR TEORI

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 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.



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.

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:**

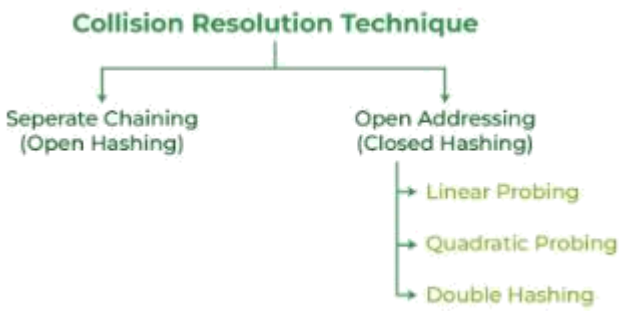
   Memperbarui data dalam hash table dengan mencari data menggunakan fungsi hash, dan kemudian memperbarui data yang ditemukan.

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(NULL) {}
};
// 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 != NULL)
            {
                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 != NULL)
        {
            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 != NULL)
        {
            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 = NULL;
        while (current != NULL)
        {
```

```
            if (current->key == key)
            {
                if (prev == NULL)
                {
                    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 != NULL)
            {
                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;
}
```

Output :

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

Deskripsi program :

Kode di atas menggunakan array dinamis “table” untuk menyimpan bucket dalam hash table. Setiap bucket diwakili oleh sebuah linked list dengan setiap node merepresentasikan satu item data. Fungsi hash sederhana hanya menggunakan modulus untuk memetakan setiap input kunci ke nilai indeks array.

Komponen Program:

- **`#include <iostream>`:** Menyertakan pustaka standar input/output untuk operasi input dan output (cout).
- **`using namespace std;`:** Menggunakan namespace `std` untuk menghindari penulisan `std::` di depan objek standar C++.
- **`const int MAX_SIZE = 10;`:** Menetapkan ukuran maksimum hash table menjadi 10.
- **`int hash_func(int key)`:** Fungsi hash sederhana yang menghitung indeks bucket dalam hash table menggunakan operator modulo (`%`).
- **`struct Node`:** Struktur untuk menyimpan data node, terdiri dari:
    - `key`: Kunci unik untuk mengidentifikasi entri.
    - `value`: Nilai yang terkait dengan kunci.
    - `next`: Pointer ke node berikutnya dalam bucket yang sama (berlaku jika terjadi tabrakan hash).
    - Konstruktor `Node(int key, int value)` untuk menginisialisasi node baru.
- **`class HashTable`:** Kelas untuk mengelola hash table, terdiri dari:
    - **Atribut pribadi:**
        - `table`: Array of pointer (`Node*`) yang mewakili bucket dalam hash table.
    - **Konstruktor:**
        - `HashTable()`: Inisialisasi hash table dengan mengalokasikan memori untuk `MAX_SIZE` pointer `Node*` pada atribut `table`, dan menginisialisasi semua pointer menjadi `NULL`.
    - **Destruktor:**

    - `~HashTable()`: Melakukan dealokasi memori yang dialokasikan untuk node dalam hash table secara iteratif untuk menghindari memory leak.
- **Metode publik:**
    - `insert(int key, int value)`: Memasukkan pasangan key-value baru ke dalam hash table.
        - Menghitung indeks bucket menggunakan fungsi hash.
        - Mencari node dengan key yang sama di bucket tersebut.
            - Jika ditemukan, update nilai (`value`) dari node tersebut.
            - Jika tidak ditemukan, buat node baru dan tambahkan ke awal bucket.
    - `get(int key)`: Mencari nilai yang terkait dengan key tertentu.
        - Menghitung indeks bucket menggunakan fungsi hash.
        - Mencari node dengan key yang sama di bucket tersebut.
            - Jika ditemukan, kembalikan nilai (`value`) dari node tersebut.
            - Jika tidak ditemukan, kembalikan -1 (menandakan key tidak ada).
    - `remove(int key)`: Menghapus node dengan key tertentu dari hash table.
        - Menghitung indeks bucket menggunakan fungsi hash.
        - Mencari node dengan key yang sama di bucket tersebut.
            - Jika ditemukan, hapus node tersebut dengan memperbarui pointer `next` di node sebelumnya (jika ada).
            - Jika tidak ditemukan, tidak ada yang perlu dilakukan.
    - `traverse()`: Menampilkan seluruh isi hash table (key-value) secara iteratif.

- **`int main()`:** Fungsi utama program:

    - Membangun objek `HashTable` bernama `ht`.
    - Memasukkan beberapa pasangan key-value (1, 10), (2, 20), dan (3, 30) ke dalam hash table menggunakan `ht.insert()`.
    - Mencari nilai untuk key 1 dan 4 menggunakan `ht.get()`.
    - Menghapus key 4 (yang tidak ada) menggunakan `ht.remove()`.
    - Menampilkan seluruh isi hash table menggunakan `ht.traverse()`.
    - Mengembalikan 0 untuk menandakan program berhasil dijalankan.

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;
}
```

Output :

```
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 :

Pada program di atas, class HashNode merepresentasikan setiap node dalam hash table, yang terdiri dari nama dan nomor telepon karyawan. Class HashMap digunakan untuk mengimplementasikan struktur hash table dengan menggunakan vector yang menampung pointer ke HashNode. Fungsi hashFunc digunakan untuk menghitung nilai hash dari nama karyawan yang diberikan, dan fungsi insert digunakan untuk menambahkan data baru ke dalam hash table. Fungsi remove digunakan untuk menghapus data dari hash table, dan fungsi searchByName digunakan untuk mencari nomor telepon dari karyawan dengan nama yang diberikan.

Komponen Program:

- **`#include <iostream>`:** Menyertakan pustaka standar input/output untuk operasi input dan output (cout).

- **`#include <string>`:** Menyertakan pustaka untuk menangani string (nama dan nomor telepon).
- **`#include <vector>`:** Menyertakan pustaka untuk menggunakan container vector (digunakan dalam hash table).
- **`using namespace std;`:** Menggunakan namespace `std` untuk menghindari penulisan `std::` di depan objek standar C++.
- **Konstanta:**
  - `const int TABLE_SIZE = 11;`: Menetapkan ukuran maksimum hash table menjadi 11.
  - `string name;`: Variabel global `name` (tidak digunakan dalam program ini).
  - `string phone_number;`: Variabel global `phone_number` (tidak digunakan dalam program ini).
- **`class HashNode`:** Kelas untuk menyimpan data kontak individual, terdiri dari:
  - `string name;`: Nama kontak.
  - `string phone_number;`: Nomor telepon kontak.
  - Konstruktor `HashNode(string name, string phone_number)` untuk menginisialisasi data kontak baru.
- **`class HashMap`:** Kelas untuk mengelola phonebook menggunakan hash table, terdiri dari:
  - **Atribut pribadi:**
    - `vector<HashNode *> table[TABLE_SIZE];`: Array of vector (`vector<HashNode* }`) yang mewakili bucket dalam hash table. Setiap bucket dapat menyimpan beberapa node menggunakan konsep chaining.
  - **Metode publik:**
    - `int hashFunc(string key)`: Fungsi hash sederhana yang menjumlahkan nilai ASCII karakter dalam key (nama kontak) untuk menghasilkan indeks bucket.
    - `void insert(string name, string phone_number)`: Memasukkan kontak baru ke dalam hash table.
      - Menghitung indeks bucket menggunakan fungsi hash.
      - Mencari kontak dengan nama yang sama di bucket tersebut (menangani kemungkinan tabrakan hash).
        - Jika ditemukan, update nomor telepon kontak tersebut.
        - Jika tidak ditemukan, buat node baru dan tambahkan ke bucket.
    - `void remove(string name)`: Menghapus kontak dengan nama tertentu dari hash table.
      - Menghitung indeks bucket menggunakan fungsi hash.
      - Mencari kontak dengan nama yang sama di bucket tersebut.
        - Jika ditemukan, hapus node tersebut dari bucket menggunakan iterator.
        - Jika tidak ditemukan, tidak ada yang perlu dilakukan.
    - `string searchByName(string name)`: Mencari nomor telepon kontak dengan nama tertentu.
      - Menghitung indeks bucket menggunakan fungsi hash.

        - Mencari kontak dengan nama yang sama di bucket tersebut.
            - Jika ditemukan, kembalikan nomor telepon kontak tersebut.
            - Jika tidak ditemukan, kembalikan string kosong.
    - `void print()`: Menampilkan seluruh isi hash table (nama dan nomor telepon) secara iteratif.
- **`int main()`:** Fungsi utama program:
    - Membangun objek `HashMap` bernama `employee_map` untuk menyimpan kontak karyawan.
    - Memasukkan beberapa kontak ke dalam hash table menggunakan `employee_map.insert()`.
    - Mencari dan menampilkan nomor telepon kontak tertentu menggunakan `employee_map.searchByName()`.
    - Menghapus salah satu kontak dari hash table menggunakan `employee_map.remove()`.
    - Mencari dan menampilkan kembali nomor telepon kontak yang dihapus (menunjukkan kontak tersebut sudah tidak ada).
    - Menampilkan seluruh isi hash table menggunakan `employee_map.print()`.
    - Mengembalikan 0 untuk menandakan program berhasil dijalankan.

BAB IV
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 <list>

using namespace std;

class Mahasiswa {
public:
    long long int nim;
    int nilai;

    Mahasiswa(long long int n, int v) : nim(n), nilai(v) {}
};

class HashTable {
private:
    static const int hashSize = 10;
    list<Mahasiswa> table[hashSize];

    int hashFunction(long long int nim) {
        return nim % hashSize;
    }

public:
    void tambahData(long long int nim, int nilai) {
```

```
        int index = hashFunction(nim);
        table[index].emplace_back(nim, nilai);
    }

    void hapusData(long long int 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(long long int nim) {
        int index = hashFunction(nim);
        for (auto it = table[index].begin(); it != table[index].end(); ++it) {
            if (it->nim == nim) {
                cout << "NIM: " << it->nim << ", Nilai: " << it->nilai <<
endl;
                return;
            }
        }
        cout << "Data tidak ditemukan." << endl;
    }


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

int main() {
    HashTable hashTable;
    int choice;

    do {
```

```
        cout << "\nMenu:\n";
        cout << "1. Tambah Data\n";
        cout << "2. Hapus Data\n";
        cout << "3. Cari Data berdasarkan NIM\n";
        cout << "4. Cari Data berdasarkan Rentang Nilai\n";
        cout << "5. Keluar\n";
        cout << "Pilihan Anda: ";
        cin >> choice;

        switch (choice) {
            case 1: {
                long long int nim;
                int nilai;
                cout << "Masukkan NIM: ";
                cin >> nim;
                cout << "Masukkan Nilai: ";
                cin >> nilai;
                hashTable.tambahData(nim, nilai);
                break;
            }
            case 2: {
                long long int nim;
                cout << "Masukkan NIM yang akan dihapus: ";
                cin >> nim;
                hashTable.hapusData(nim);
                break;
            }
            case 3: {
                long long int nim;
                cout << "Masukkan NIM yang akan dicari: ";
                cin >> nim;
                hashTable.cariByNIM(nim);
                break;
            }
            case 4: {
                int minNilai, maxNilai;
                cout << "Masukkan Rentang Nilai (Min Max): ";
                cin >> minNilai >> maxNilai;
                cout << "Mahasiswa dengan nilai antara " << minNilai << " - "
<< maxNilai << ":\n";
                hashTable.cariByRange(minNilai, maxNilai);
                break;
            }

            case 5:
```

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

 return 0;
}
```

Output :

```
Menu:
1. Tambah Data
2. Hapus Data
3. Cari Data berdasarkan NIM
4. Cari Data berdasarkan Rentang Nilai
5. Keluar
Pilihan Anda:
```

```
Menu:
1. Tambah Data
2. Hapus Data
3. Cari Data berdasarkan NIM
4. Cari Data berdasarkan Rentang Nilai
5. Keluar
Pilihan Anda: 1
Masukkan NIM: 2311102017
Masukkan Nilai: 90
```

```
Menu:
1. Tambah Data
2. Hapus Data
3. Cari Data berdasarkan NIM
4. Cari Data berdasarkan Rentang Nilai
5. Keluar
Pilihan Anda: 2
Masukkan NIM yang akan dihapus: 2311102005
```

```
Menu:
1. Tambah Data
2. Hapus Data
3. Cari Data berdasarkan NIM
4. Cari Data berdasarkan Rentang Nilai
5. Keluar
Pilihan Anda: 3
Masukkan NIM yang akan dicari: 2311102016
NIM: 2311102016, Nilai: 86
```

```
Menu:
1. Tambah Data
2. Hapus Data
3. Cari Data berdasarkan NIM
4. Cari Data berdasarkan Rentang Nilai
5. Keluar
Pilihan Anda: 4
Masukkan Rentang Nilai (Min Max): 80 90
Mahasiswa dengan nilai antara 80 - 90:
NIM: 2311102016, Nilai: 86
NIM: 2311102017, Nilai: 90
NIM: 2311102029, Nilai: 83
```

```
Menu:
1. Tambah Data
2. Hapus Data
3. Cari Data berdasarkan NIM
4. Cari Data berdasarkan Rentang Nilai
5. Keluar
Pilihan Anda: 5
Terima kasih!
```

Deskripsi Program :

Program ini mengimplementasikan sistem database mahasiswa sederhana menggunakan struktur data **Hash Table** dalam bahasa C++. Hash table digunakan untuk menyimpan dan mengelola data mahasiswa dengan efisien.

Komponen Program:

- **`#include <iostream>`:** Menyertakan pustaka standar input/output untuk operasi input dan output (cout, cin).
- **`#include <list>`:** Menyertakan pustaka untuk menggunakan container list (digunakan untuk menyimpan data mahasiswa dalam hash table).
- **`using namespace std;`:** Menggunakan namespace `std` untuk menghindari penulisan `std::` di depan objek standar C++.

Kelas:

- **`class Mahasiswa`:** Kelas untuk mewakili data mahasiswa, terdiri dari:
  - `int nim`: NIM (Nomor Induk Mahasiswa) mahasiswa.
  - `int nilai`: Nilai ujian mahasiswa.
  - Konstruktor `Mahasiswa(int n, int v)` untuk menginisialisasi data mahasiswa baru.
- **`class HashTable`:** Kelas untuk mengelola database mahasiswa menggunakan hash table, terdiri dari:
  - **Atribut pribadi:**
    - `static const int hashSize = 10;`: Menetapkan ukuran hash table menjadi 10 (konstanta).
    - `list<Mahasiswa> table[hashSize];`: Array of list (`list<Mahasiswa>`) yang mewakili bucket dalam hash table. Setiap bucket dapat menyimpan beberapa data mahasiswa.

- **Metode publik:**
  - `int hashFunction(int nim)`: Fungsi hash sederhana yang menjumlahkan nilai ASCII karakter dalam NIM untuk menghasilkan indeks bucket.
  - `void tambahData(int nim, int nilai)`: Menambahkan data mahasiswa baru ke dalam hash table.
    - Menghitung indeks bucket menggunakan fungsi hash.
    - Memasukkan data mahasiswa baru (`Mahasiswa(nim, nilai)`) ke dalam bucket yang sesuai.
  - `void hapusData(int nim)`: Menghapus data mahasiswa berdasarkan NIM tertentu.
    - Menghitung indeks bucket menggunakan fungsi hash.
    - Mencari data mahasiswa dengan NIM yang sama dalam bucket tersebut.
      - Jika ditemukan, hapus data mahasiswa tersebut dari bucket.
      - Jika tidak ditemukan, tampilkan pesan error.
  - `void cariByNIM(int nim)`: Mencari data mahasiswa berdasarkan NIM tertentu.
    - Menghitung indeks bucket menggunakan fungsi hash.
    - Mencari data mahasiswa dengan NIM yang sama dalam bucket tersebut.
      - Jika ditemukan, tampilkan data mahasiswa (NIM dan nilai).
      - Jika tidak ditemukan, tampilkan pesan error.
  - `void cariByRange()` : Mencari data mahasiswa berdasarkan rentang nilai (80 - 90).
    - Mengiterasi setiap bucket dalam hash table.
    - Untuk setiap data mahasiswa dalam bucket, periksa apakah nilai berada dalam rentang 80 - 90.
      - Jika ya, tampilkan data mahasiswa (NIM dan nilai).

Fungsi Utama (`main()`):

- Membangun objek `HashTable` bernama `hashTable` untuk menyimpan data mahasiswa.
- Menampilkan menu utama program dengan pilihan:

1. Tambah Data: Memasukkan data mahasiswa baru (NIM dan nilai) ke dalam hash table.
2. Hapus Data: Menghapus data mahasiswa berdasarkan NIM tertentu.
3. Cari Data berdasarkan NIM: Mencari data mahasiswa berdasarkan NIM tertentu dan menampilkannya.

4. Cari Data berdasarkan Rentang Nilai (80 - 90): Mencari data mahasiswa dengan nilai antara 80 dan 90 dan menampilkannya.
5. Keluar: Mengakhiri program.

- Membaca pilihan pengguna dan melakukan operasi yang sesuai menggunakan objek `hashTable`.
- Mengulangi menu utama sampai pengguna memilih opsi 5 (Keluar).
- Mengembalikan 0 untuk menandakan program berhasil dijalankan.

BAB IV

KESIMPULAN

Hash table adalah struktur data yang efisien untuk menyimpan dan mengambil data pasangan key-value. Hash table bekerja dengan cara memetakan key ke indeks bucket dalam array menggunakan fungsi hash. Data kemudian disimpan dalam bucket yang sesuai. Hash table menawarkan beberapa keuntungan dibandingkan struktur data lain seperti array dan linked list:

Keuntungan:

- **Pencarian Cepat:** Hash table memungkinkan pencarian data dengan sangat cepat, rata-rata O(1) operasi. Hal ini karena key digunakan untuk langsung mengakses bucket yang berisi data yang terkait.
- **Penggunaan Memori Efisien:** Hash table dapat menggunakan memori dengan lebih efisien dibandingkan array dan linked list, terutama ketika data tidak terurut.
- **Keserbagunaan:** Hash table dapat digunakan untuk menyimpan berbagai jenis data, seperti string, integer, dan objek.
- **Implementasi Relatif Mudah:** Hash table tergolong struktur data yang mudah dipahami dan diimplementasikan.

Kekurangan:

- **Tabrakan Hash:** Tabrakan hash terjadi ketika dua key dipetakan ke indeks bucket yang sama. Hal ini dapat menyebabkan data yang hilang atau data yang salah. Ada beberapa cara untuk menangani tabrakan hash, seperti chaining dan probing.
- **Ketergantungan pada Fungsi Hash:** Performa hash table sangat bergantung pada fungsi hash yang digunakan. Fungsi hash yang buruk dapat menyebabkan banyak tabrakan hash, yang dapat memperlambat operasi pencarian dan penyisipan.
- **Membutuhkan Memori Tambahan:** Hash table membutuhkan memori tambahan untuk menyimpan array bucket dan data yang terkait.

Kesimpulan:

Hash table adalah struktur data yang sangat berguna untuk menyimpan dan mengambil data pasangan key-value dengan cepat dan efisien. Hash table memiliki beberapa keuntungan dibandingkan struktur data lain, seperti kecepatan pencarian yang cepat dan penggunaan memori yang efisien. Namun, hash table juga memiliki beberapa kekurangan, seperti tabrakan hash dan ketergantungan pada fungsi hash.

Sebelum memilih hash table, penting untuk mempertimbangkan kebutuhan spesifik aplikasi dan menimbang keuntungan dan kekurangannya.

Aplikasi Hash Table:

Hash table digunakan dalam berbagai aplikasi, seperti:

- **Cache:** Hash table digunakan untuk menyimpan data yang sering diakses dengan cepat, seperti hasil query database atau objek game.
- **Compiler:** Hash table digunakan untuk menyimpan simbol tabel dan informasi tentang variabel dan fungsi.
- **Database:** Hash table digunakan untuk menyimpan data indeks untuk mempercepat pencarian data.
- **Internet:** Hash table digunakan untuk menyimpan data routing dan DNS records.

DAFTAR PUSTAKA

Asisten Praktikum (2024). Hash Table. Learning Management System.