LAPORAN PRAKTIKUM

MODUL 5
HASH TABLE

Disusun Oleh:
Riyon Aryono **: 2211102241**

Dosen

Muhammad Afrizal Amrustian

PROGRAM STUDI S1 TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM
PURWOKERTO

2023

BAB I
TUJUAN PRAKTIKUM

A. Tujuan Praktikum

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

BAB II
DASAR TEORI

1. Pengertian Hash Table

Tabel Hash adalah struktur data yang digunakan untuk menyimpan pasangan kunci/nilai. Hash table biasanya terdiri dari dua komponen utama: array (atau vector) dan fungsi hash. Dengan menggunakan fungsi hash yang baik hashing dapat berjalan dengan baik. 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.

Contoh: Pertimbangkan sebuah array sebagai Peta di mana kuncinya adalah indeks dan nilainya adalah value pada indeks itu. Jadi untuk array A jika

kita memiliki indeks i yang akan diperlakukan sebagai kunci maka kita dapat menemukan nilainya hanya dengan mencari value pada A[i]. Tipe fungsi hash:

a. Devision Method.
b. Mid Squer Method.
c. Folding Method.
d. Multiplication Method.

Untuk mencapai mekanisme hashing yang baik, penting untuk memiliki fungsi hash yang baik dengan persyaratan dasar sebagai berikut:

a. Dapat dihitung secara efesien.
b. Harus mendistribusikan kunci secara seragam (Setiap posisi tabel memiliki kemungkinan yang sama untuk masing-masing.)
c. Harus meminimalkan tabrakan.
d. Harus memiliki faktor muatan rendah (jumlah item dalam tabel dibagi dengan ukuran tabel).

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 Resoltuion

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:

a. 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 dihash. 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.

b. Cosed Hashing

1. 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.

2. Quadratic Probing

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

3. Double Hashing

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

BAB III
LATIHAN & TUGAS

1. Guided

- Demo Hash Table

Source Code

```
#include <iostream>
using namespace std;

const int MAX_SIZE = 10;

// fungsi hash sederhana
int hash_func(int key){
   return key % MAX_SIZE;
}

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 = key;
            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;
        }
```

```
        }

        // Travesal
        void travese(){
            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.travese();
}
```

Output

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

Penjelesan

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.

- Demo 2 Hash Table

  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","12345");
    employee_map.insert("Pastah","5678");
    employee_map.insert("Pastah","5678");
    cout << "Nomor HP Mistah : " <<
   employee_map.searchByName("Mistah") << endl;
    employee_map.print();
}
```

Output

```
Nomor HP Mistah : 12345
0:
1:
2:
3:
4: [Pastah, 5678]
5:
6:
7:
8:
9: [Mistah, 12345]
10:
```

Penjelesan

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.

2. Unguided

- 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.

Source Code

```
#include <iostream>
using namespace std;

const int MAX_SIZE = 10;


struct Node{
    int nim;
    int nilai;
};

// Class hash table
class HashTable{
    private:
     static const int tableSize= 10;
     Node* table[tableSize];
    public:
        HashTable(){
            for (int i = 0; i < tableSize; i++){
                table[i] = nullptr;
            }
        }

        int hashFunc(int nim){
            return nim % tableSize;
        }

        void add(int nim, int nilai){
            int index = hashFunc(nim);
            while(table[index] != nullptr && table[index]->nim != nim){
                index = (index + 1) % tableSize;
            }

            if(table[index] != nullptr){
                delete table[index];
            }

            table[index] = new Node;
            table[index]->nim = nim;
            table[index]->nilai = nilai;
        }
```

```
    void print(){
        for (int i = 0; i < tableSize; i++){
            if(table[i] != nullptr){
                cout << "NIM: " << table[i]->nim << endl;
                cout << "Nilai: " << table[i]->nilai <<endl<<endl;
            }
        }

    }

    void remove(int nim){
        int index = hashFunc(nim);
        while(table[index] != nullptr){
            if(table[index]->nim == nim){
                break;
            }
            index = (index + 1) % tableSize;
        }

        if(table[index] == nullptr){
            cout << nim << " tidak ditemukan." <<endl;
            return;
        }

        delete table[index];
        table[index] = nullptr;
    }

    void searchByNim(int nim) {
        int index = hashFunc(nim);
        while (table[index] != nullptr) {
            if (table[index]->nim == nim) {
                cout << "NIM: " << table[index]->nim << endl;
                cout << "Nilai: " << table[index]->nilai << endl << endl;
                return;
            }
            index = (index + 1) % tableSize;
         }
       cout << "Data dengan NIM " << nim << " tidak ditemukan." <<
endl;
    }

    void searchByRentangNilai(){
        bool found = false;
        for(int i = 0; i < tableSize; i++){
```

```
            if(table[i] != nullptr && table[i]->nilai >=80 && table[i]->nilai
  <= 90){
                cout << "NIM: " << table[i]->nim <<endl;
                cout << "Nilai: " << table[i]->nilai <<endl<<endl;
                found = true;
            }
        }
        if(!found){
            cout << "data dengan rentang nilai 80 - 90 tidak ditemukan"
  <<endl;
        }
    }


};

int main(){
   HashTable dataMhs;
   int choice, nim, nilai;

   do{
      cout << "\n--- Data mahasiswa ---" <<endl;
      cout << "1.Tambah Data" <<endl;
      cout << "2.Hapus Data" <<endl;
      cout << "3.Cari Data Berdasarkan NIM" <<endl;
      cout << "4.Cari Data Berdasarkan Rentang Nilai" <<endl;
      cout << "5.Print Data" <<endl;
      cout << "6.Keluar" <<endl;
      cout << "Pilih Menu [1-6]: ";
      cin >> choice;

      switch (choice){
      case 1:
            cout << "Masukkan NIM: ";
            cin  >> nim;
            cout << "Masukkan Nilai: ";
            cin  >> nilai;
            dataMhs.add(nim,nilai);
         break;
      case 2:
         cout << "Masukkan NIM : ";
         cin  >> nim;
         dataMhs.remove(nim);
         break;
      case 3:
         cout << "Masukkan NIM: ";
```

```
            cin  >> nim;
            dataMhs.searchByNim(nim);
            break;
        case 4:
            dataMhs.searchByRentangNilai();
            break;
        case 5:
            dataMhs.print();
            break;
        case 6:
            cout << "Keluar Program." <<endl;
            break;
        default:
            cout << "Menu salah. Silahkan Coba Lagi" << endl;
            break;
        }

    }while(choice != 6);

}
```

Output

Tambah data

```
--- Data mahasiswa ---
1.Tambah Data
2.Hapus Data
3.Cari Data Berdasarkan NIM
4.Cari Data Berdasarkan Rentang Nilai
5.Print Data
6.Keluar
Pilih Menu [1-6]: 1
Masukkan NIM: 241
Masukkan Nilai: 87
```

```
--- Data mahasiswa ---
1.Tambah Data
2.Hapus Data
3.Cari Data Berdasarkan NIM
4.Cari Data Berdasarkan Rentang Nilai
5.Print Data
6.Keluar
Pilih Menu [1-6]: 5
NIM: 241
Nilai: 87

NIM: 232
Nilai: 96

NIM: 233
Nilai: 78

NIM: 244
Nilai: 90

NIM: 265
Nilai: 85
```

Hapus data

```
--- Data mahasiswa ---
1.Tambah Data
2.Hapus Data
3.Cari Data Berdasarkan NIM
4.Cari Data Berdasarkan Rentang Nilai
5.Print Data
6.Keluar
Pilih Menu [1-6]: 2
Masukkan NIM : 232
```

```
--- Data mahasiswa ---
1.Tambah Data
2.Hapus Data
3.Cari Data Berdasarkan NIM
4.Cari Data Berdasarkan Rentang Nilai
5.Print Data
6.Keluar
Pilih Menu [1-6]: 5
NIM: 241
Nilai: 87

NIM: 233
Nilai: 78

NIM: 244
Nilai: 90

NIM: 265
Nilai: 85
```

Cari data berdasarkan NIM

```
--- Data mahasiswa ---
1.Tambah Data
2.Hapus Data
3.Cari Data Berdasarkan NIM
4.Cari Data Berdasarkan Rentang Nilai
5.Print Data
6.Keluar
Pilih Menu [1-6]: 3
Masukkan NIM: 244
NIM: 244
Nilai: 90
```

Cari data berdaraskan rentang nilai 80 - 90

```
--- Data mahasiswa ---
1.Tambah Data
2.Hapus Data
3.Cari Data Berdasarkan NIM
4.Cari Data Berdasarkan Rentang Nilai
5.Print Data
6.Keluar
Pilih Menu [1-6]: 4
NIM: 241
Nilai: 87

NIM: 244
Nilai: 90

NIM: 265
Nilai: 85
```

Penjelesan

Pertama-tama, kita membuat **struct Node** untuk menyimpan data NIM dan nilai mahasiswa. Selanjutnya, kita membuat **class** HashTable untuk membuat hash table dan menyimpan data mahasiswa di dalamnya. Kita membuat array table untuk menyimpan data mahasiswa dengan ukuran maksimum **TABLE_SIZE**. Untuk membuat **hash function**, kita mengambil digit terakhir dari NIM sebagai index hash. Ini cukup sederhana dan efektif untuk ukuran hash table yang relatif kecil. Fungsi **add** digunakan untuk menambahkan data mahasiswa ke dalam hash table. Fungsi **remove** digunakan untuk menghapus data mahasiswa dari hash table berdasarkan NIM. Fungsi **searchByRentangNilai** digunakan untuk mencari data mahasiswa berdasarkan rentang nilai 80-90. Kita melakukan iterasi pada setiap elemen array dan mencetak data mahasiswa yang memiliki nilai dalam rentang tersebut.

BAB IV
KESIMPULAN

HashTable adalah struktur data yang berguna untuk menyimpan data dalam bentuk key-value pair. HashTable memungkinkan penambahan, pencarian, dan penghapusan data dengan waktu akses konstan, yaitu. O(1), tergantung pada fungsi hash yang digunakan. Dengan implementasi yang baik, waktu akses O(1) dapat dipertahankan bahkan dengan jumlah data yang besar.

HashTables sangat berguna untuk mempercepat akses data, terutama saat kita perlu mencari data dengan cepat. Beberapa penggunaan umum dari HashTables termasuk database, mesin pencari, pengenalan wajah, dan pengolah kata.