LAPORAN PRAKTIKUM
STRUKTUR DATA
MATERI KE – 8 : Hash Table

Oleh :

Ridho Aulia Rahman (220605110044)

Dosen Pengampu :

ASHRI SHABRINA AFRAH,M.T.

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS SAINS DAN TEKNOLOGI

UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM

MALANG

2023

A. Pendahuluan

1. Lengkapi listing program tersebut dengan menanmbahkan class HashTableApp yang memiliki method main. Lakukan penambahan 10 item pada hash table berukuran 15, tampilkan isi data. Tulis output program yang telah anda lengkapi dan jelaskan bagaimana penentuan indeks (sel array) yang digunakan untuk menyimpan tiap item tersebut!

```
public class HashTableApp {
    public static void main(String[] args) {
        HashTable hashTable = new HashTable(15);
        hashTable.insert(43);
        hashTable.insert(23);
        hashTable.insert(42);
        hashTable.insert(44);
        hashTable.insert(56);
        hashTable.insert(78);
        hashTable.insert(79);
        hashTable.insert(10);
        hashTable.insert(21);
        hashTable.insert(21);
        hashTable.displayTable();
    }

}
```

Jika class main tersebut di run maka akan menghasilkan output sebagai berikut :

```
Table:
| 0     | -- |
| 1     | -- |
| 2     | -- |
| 3     | 78
| 4     | 79
| 5     | -- |
| 6     | 21
| 7     | 21
| 8     | 23
| 9     | -- |
| 10    | 10
| 11    | 56
| 12    | 42
| 13    | 43
| 14    | 44
```

Untuk penentuan indeks pada sel array diatas perlunya untuk melihat kembali method dibawah ini

```
public HashTable(int size) {
        this.size = size;
        hashArray = new Data[size];
    }

    public void displayTable() {
        System.out.println("Table: ");
```

```
        for (int i = 0; i < size; i++) {
            if (hashArray[i] != null) {
                System.out.println("| " + i + "\t| " +
hashArray[i].getKey() + " ");
            } else {
                System.out.println("| " + i + "\t| -- |");
            }
        }
        System.out.println();
    }

    public int hashFunc(int key) {
        return key % size;
    }
```

Pada kelas HashTable mempunyai constructor yang akan menginisialisasi ukuran array yang akan dibuat, yang kemudian pada method hashFunc indeks akan ditentukan melalui method ini, jika kita melihat pada data yang kita inputkan yang pertama adalah 43, dan pada method hasfunc akan menentukan indeks dengan cara key yaitu 43 akan dimodulus dengan size array yaitu 15, maka akan hasilnya adalah 13, maka jika kita lihat pada output benar 43 berada pada indeks 13

2. Tambahkan 5 item lain pada hash table tersebut. Tampilkan isi table sebelum penambahan (10 item) dan setelah penambahan (15 item). Dimana posisi 5 item yang baru saja ditambahkan? Jelaskan!

```
public class HashTableApp {
    public static void main(String[] args) {
        HashTable hashTable = new HashTable(15);
        hashTable.insert(43);
        hashTable.insert(23);
        hashTable.insert(42);
        hashTable.insert(44);
        hashTable.insert(56);
        hashTable.insert(78);
        hashTable.insert(79);
        hashTable.insert(10);
        hashTable.insert(21);
        hashTable.insert(21);
        hashTable.displayTable();

        // Menambahkan data baru
        System.out.println("Menambahkan data baru");
        hashTable.insert(19);
        hashTable.insert(13);
        hashTable.insert(14);
        hashTable.insert(16);
        hashTable.insert(18);
        hashTable.displayTable();
```

```
    }

}
```

Maka jika kita running program diatas maka akan menghasilkan output sebagai berikut :

```
Table:
| 0     | -- |
| 1     | -- |
| 2     | -- |
| 3     | 78
| 4     | 79
| 5     | -- |
| 6     | 21
| 7     | 21
| 8     | 23
| 9     | -- |
| 10    | 10
| 11    | 56
| 12    | 42
| 13    | 43
| 14    | 44

Menambahkan data baru
Table:
| 0     | 13
| 1     | 14
| 2     | 16
| 3     | 78
| 4     | 79
| 5     | 19
| 6     | 21
| 7     | 21
| 8     | 23
| 9     | 18
| 10    | 10
| 11    | 56
| 12    | 42
| 13    | 43
| 14    | 44
```

Pada program diatas kita akan mendapati ada angka yang tidak menempati indeks yang seharusnya, yaitu 13 seharusnya menempati indeks 13, maka pada method insert

```
public void insert(int data) {
        Data item = new Data(data);
        int key = item.getKey();
        int hashVal = hashFunc(key);
        while (hashArray[hashVal] != null) {
            ++hashVal;
            hashVal %= size;
        }
        hashArray[hashVal] = item;
    }
```

Kita bisa melihat bahwa perulangan while akan mencari indeks yang masih

kosong untuk diisi value, begitu seterusnya sampai semua indeks terpenuhi

3. Pada method insert terdapat baris code
   while (hashArray[hashVal] != null) {
   ++hashVal; hashVal %= size;
   }
   Apa yang terjadi ketika bari tersebut dihapus dan program dijalankan untuk menambahkan item? Jelaskan!
   Telah kita ketahui bahwa baris kode diatas berfungsi untuk mencari indeks yang kosong ketika terjadi collision jika baris kode diatas dihapus pada program kita akan menghasilkan output sebagai berikut :

```
Menambahkan data baru
Table:
| 0     | -- |
| 1     | 16
| 2     | -- |
| 3     | 18
| 4     | 19
| 5     | -- |
| 6     | 21
| 7     | -- |
| 8     | 23
| 9     | -- |
| 10    | 10
| 11    | 56
| 12    | 42
| 13    | 13
| 14    | 14
```

   Maka data lama akan hilang karena tertumpuk data yang baru ditambahkan

4. Lakukan pencarian berdasarkan key tertentu pada class HashTableApp!, jelaskan bagaimana proses pencarian pada method find?
   Saya akan menambahkan baris kode dibawah pada class HashTableApp

```
Data find = hashTable.find(19);
        if (find == null)
            System.out.println("Data tidak ditemukan");
        else
            System.out.println("Data ditemukan : " +
find.getKey());
```

   Dan jika baris kode diatas di jalanakan maka akan menghasilkan output sebagai berikut :

```
Data ditemukan : 19
```

```
public Data find(int key) {
        int hashVal = hashFunc(key);
        while (hashArray[hashVal] != null) {
            if (hashArray[hashVal].getKey() == key) {
                return hashArray[hashVal];
            }
            ++hashVal;
            hashVal %= size;
        }
        return null;
    }
```

Cara kerja dari method find adalah Pertama, fungsi hashFunc akan mengembalikan nilai hash dari key. Kemudian, fungsi find akan mencari objek Data yang memiliki kunci yang sama dengan key dengan melakukan iterasi pada array hash hashArray mulai dari indeks hashVal yang dihitung menggunakan fungsi hashFunc. Jika objek Data dengan kunci yang sama ditemukan, fungsi find akan mengembalikan objek Data tersebut. Jika tidak ditemukan, fungsi find akan mengembalikan null.

5. Lakukan hapus item berdasarkan key tertentu pada class HashTableApp!, jelaskan bagaimana proses pencarian pada method delete?

Saya menambahkan baris kode dibawah ini untuk menghapus item :

```
Data delete = hashTable.delete(10);
        if (delete == null)
        System.out.println("Data Tidak Ada");
        else
        System.out.println("Data Terhapus : " + delete.getKey());
        hashTable.displayTable();
```

program diatas akan mengahapus angka 10 maka jika kita running programnya maka akan menghasilkan output sebagai berikut :

```
Data Terhapus : 10
Table:
| 0     | 13
| 1     | 14
| 2     | 16
| 3     | 78
| 4     | 79
| 5     | 19
| 6     | 21
| 7     | 21
| 8     | 23
| 9     | 18
| 10    | -- |
| 11    | 56
| 12    | 42
| 13    | 43
| 14    | 44
```

Data 10 berhasil dihapus dari table

Dan cara kerja dari method delete sebagai berikut :

```
public Data delete(int key) {
        int hashVal = hashFunc(key);
        while (hashArray[hashVal] != null) {
            if (hashArray[hashVal].getKey() == key) {
                Data temp = hashArray[hashVal];
                hashArray[hashVal] = null;
                return temp;
            }
            ++hashVal;
            hashVal %= size;
        }
```

```
        return null;
    }
```

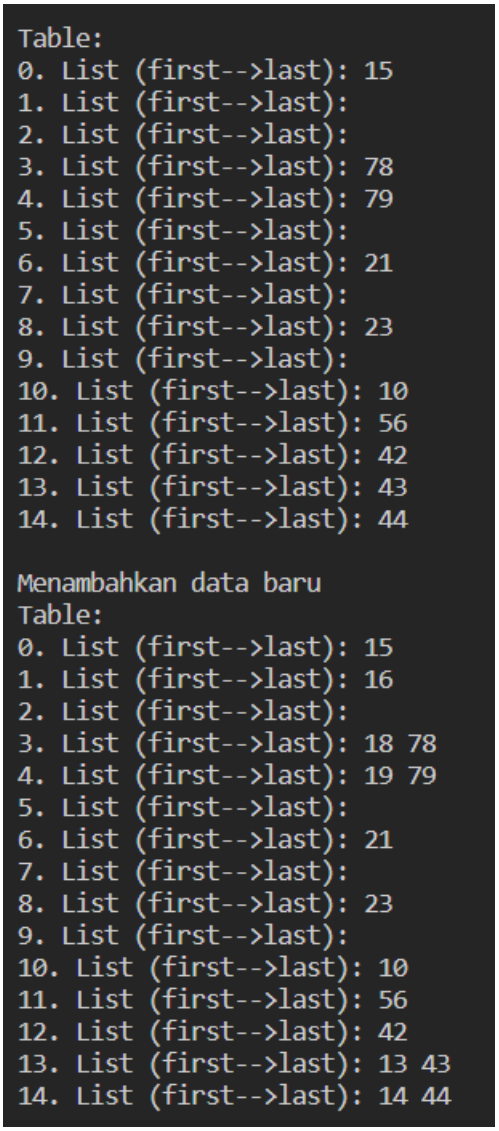
Cara kerjanya adalah Pertama, fungsi hashFunc akan mengembalikan nilai hash dari key. Kemudian, fungsi delete akan mencari objek Data yang memiliki kunci yang sama dengan key dengan melakukan iterasi pada array hash hashArray mulai dari indeks hashVal yang dihitung menggunakan fungsi hashFunc. Jika objek Data dengan kunci yang sama ditemukan, fungsi delete akan menghapus objek Data tersebut dari array hash hashArray dan mengembalikan objek Data tersebut. Jika tidak ditemukan, fungsi delete akan mengembalikan null.

6. Lengkapi listing program tersebut dengan menambahkan class HashChainApp yang berisi method main. lakukan penambahan data sebagaimana soal nomer 1. Jalankan program dan tulis outputnya. jelaskan bagaimana penentuan indeks (sel array) yang digunakan untuk menyimpan tiap item tersebut!

```
public class HashChainApp {
    public static void main(String[] args) {
        HashTable hashTable = new HashTable(15);
        hashTable.insert(43);
        hashTable.insert(23);
        hashTable.insert(42);
        hashTable.insert(44);
        hashTable.insert(56);
        hashTable.insert(78);
        hashTable.insert(79);
        hashTable.insert(10);
        hashTable.insert(21);
        hashTable.insert(15);
        hashTable.displayTable();

        // Menambahkan data baru
        System.out.println("Menambahkan data baru");
        hashTable.insert(19);
        hashTable.insert(13);
        hashTable.insert(14);
        hashTable.insert(16);
        hashTable.insert(18);
        hashTable.displayTable();
    }
}
```

Jika method main tersebut di run maka akan menghasilkan output sebagai berikut :



cara penentuan indeks kurang lebih sama dengan open adrress, namun pada open hashing setiap indeks merupakan linked list, jika terjadi collision maka akan value/ key tetap berada pada indeks yang sesuai yaitu ditambahkan pada linked list sesuai indeks masing masing.

Praktikum

Karena NIM saya adalah genap maka saya akan memakai teknik quadratic probing

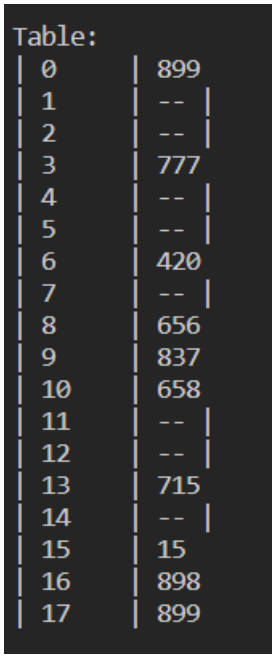
1. Pada quadratic probing, saat terjadi collision dilakukakan increment menggunakan fungsi quadratic: f(i)= i2. Sehingga jarak dari posisi awal adalah x+12, x+22, x+32, x+42, x+52, dan seterusnya hingga ditemukan sel kosong.
   Jawab : Implementasi dengan cara membuat method baru dengan logic sebagai berikut

```
public void insertQuadratic(int data) {
        Data item = new Data(data);
        int key = item.getKey();
        int hashVal = hashFunc(key);
        int i = 0;
        while (hashArray[hashVal] != null) {
            ++i;
            hashVal = (hashFunc(key) + i * i) % size;
        }
        hashArray[hashVal] = item;
    }
```

   Jadi ketika ditemukan indeks ke-n sama dengan null maka ia akan mencari indeks dengan kelipatan kuadrat

```
class quadratic{
    public static void main(String[] args) {
        HashTable hashTable = new HashTable(18);
        hashTable.insertQuadratic(656);
        hashTable.insertQuadratic(715);
        hashTable.insertQuadratic(658);
        hashTable.insertQuadratic(777);
        hashTable.insertQuadratic(837);
        hashTable.insertQuadratic(899);
        hashTable.insertQuadratic(899);
        hashTable.insertQuadratic(15);
        hashTable.insertQuadratic(420);
        hashTable.insertQuadratic(898);
        hashTable.displayTable();
    }
}
```

   Jawaban : dari method main diatas kita menginputkan 10 buah bilangan, maka jika kita running program tersebut akan menghasilkan output sebagai berikut :



Penjelasan : contoh dari data diatas kita memiliki collasion yaitu 899, jika 899 kita moduluskan dengan 18 maka hasilnya adalah 17, maka kita lihat 17 berada pada indeks 17, lalu ada 899 lagi karena 17 sudah ada valuenya, maka akan mencari indeks yang kosong dengan kelipatan 1 kuadrat maka 899 yang kedua berada pada indeks 0. Begitu seterusnya jika terjadi collasion

B. Kesimpulan

1. Tentang Hash Table
   Hash table adalah struktur data yang efisien untuk menyimpan dan mengambil data dengan menggunakan fungsi hash. Fungsi hash ini mengonversi kunci data menjadi indeks dalam tabel, Namun, collision hash dapat terjadi ketika dua kunci menghasilkan indeks yang sama, dan hash table harus mengatasi konflik ini dengan teknik seperti chaining atau open addressing.
2. Tentang Perbedaan Open Addressing dan Separate Chain
   Open Addressing:
   Mengatasi konflik dengan mencoba menemukan slot kosong lain di dalam tabel hash.
   Tidak menggunakan struktur data tambahan untuk menangani konflik.
   Membutuhkan metode probing untuk menentukan slot yang akan diisi.
   Lebih efisien dalam penggunaan memori karena tidak memerlukan struktur data tambahan.
   Dapat mengalami clustering, di mana sekumpulan slot terisi secara berdekatan, memungkinkan konflik lebih lanjut.

   Separate Chaining:
   Mengatasi konflik dengan menggunakan struktur data tambahan, seperti linked list, untuk menyimpan data dengan kunci yang menghasilkan indeks yang sama.
   Setiap slot pada tabel hash berisi referensi ke linked list atau struktur data lainnya.
   Memerlukan alokasi memori tambahan untuk menyimpan struktur data

tambahan.

Lebih tahan terhadap clustering, karena setiap indeks dapat menyimpan banyak elemen dalam linked list terpisah.

Operasi pencarian dan penyisipan dapat lebih lambat daripada Open Addressing karena melibatkan manipulasi linked list.

3. Tentang perbedaan linier probing, quadratic probing, dan double hashing sebagai collision resolution

Linier Probing:

Mengatasi konflik dengan mencari slot berikutnya yang tersedia secara berurutan. Formula untuk menentukan slot berikutnya adalah (hash_value + 1) % table_size.

Mudah diimplementasikan dan membutuhkan sedikit perhitungan.

Rentan terhadap clustering linier, di mana slot-slot berurutan dapat terisi, meningkatkan kemungkinan konflik lebih lanjut.

Cenderung menyebabkan efek clustering karena pencarian slot kosong berurutan.

Quadratic Probing:

Mencari slot berikutnya dengan menggunakan fungsi kuadrat, dengan formula (hash_value + c1 * i + c2 * i^2) % table_size, di mana i adalah iterasi konflik.

Memiliki keunggulan dalam menghindari clustering linier, karena menggunakan fungsi kuadrat untuk menentukan langkah selanjutnya.

Membutuhkan perhitungan lebih kompleks dibandingkan linier probing.

Dapat menyebabkan clustering kuadrat, di mana slot terisi secara periodik.

Pilihan parameter c1 dan c2 mempengaruhi performa metode ini.

Double Hashing:

Mengatasi konflik dengan menggunakan dua fungsi hash, yaitu h1(key) untuk menentukan slot awal dan h2(key) untuk menentukan langkah selanjutnya.

Formula umumnya adalah (h1(key) + i * h2(key)) % table_size.

Menawarkan solusi yang efisien dan menghindari clustering secara signifikan.

Memerlukan implementasi dua fungsi hash yang baik dan independen.

Menyulitkan proses pencarian slot yang kosong karena bergantung pada kedua fungsi hash.