**PRAKTIKUM 8 - STRUKTUR DATA**

HASH TABLES

*Learning outcomes*:

1. Mahasiswa mampu mengimplementasikan struktur data Hash Table pada sebuah program
2. Mahasiswa mampu mengimplementasikan teknik *Open Addressing* (*linier probing*, *quadratic probing*, dan *double hashing*) pada hash table.
3. Mahasiswa mampu mengimplementasikan teknik *Separate Chaining* pada Hash Table
4. Mahasiswa mampu menjelaskan perbedaan teknik *Open Addressing* dan *Separate Chaining*.

# IDENTITAS PRAKTIKAN

NIM :

Nama Lengkap : Kelas/ Hari/ Jam : / / Nomor komputer : Nama Asisten :

|  |  |
| --- | --- |
| **Tugas** | **Praktikum** |
| Telah diperiksa pada tanggal  *(nilai dan paraf asisten)* | Telah diperiksa pada tanggal  *(nilai dan paraf asisten)* |

Hash Table adalah sebuah struktur data yang sangat cepat dalam insertion dan searching. Hash table diimplementasikan menggunakan array. Penambahan dan pencarian sebuah key pada hash table berdasarkan fungsi hash yang digunakan. Fungsi hash memetakkan elemen pada indek hash table.

Fungsi hash yang baik memiliki sifat berikut: mudah dihitung, cukup mampu mendistribusikan key, meminimalkan jumlah collision (tabrakan) yang terjadi. Fungsi hash dapat menggunakan beberapa teknik, diantaranya:

1. Truncation. Mengambil beberapa digit dari key sebagai indeks
2. Folding. Menjumlahkan beberapa digit dari key, hasil penjumlahan sebagai indeks
3. Modular. Menggunakan sisa hasil bagi dari key (bilangan bulat) dengan ukuran hash table.

Berdasarkan teknik untuk menangani collision, hash table dapat dikategorikan menjadi: Closed Hashing (Open Addressing) dan Open hashing (Separate Chaining). Collision sendiri terjadi ketika terdapat dua key/lebih yang dipetakkan pada sebuah sel array yang sama.

# PENDAHULUAN

* 1. **Open Addressing**

Ide dari teknik ini adalah mencari alternative sel lain pada table ketika terjadi collision. Pada proses insertion, ketika indeks yang telah ditentukan dari Fungsi hash yang digunakan sudah berisi suatu item, maka akan mencari sel lain sesuai urutan menggunakan fungsi pencarian urutan. Beberapa startegi untuk menentukan fungsi pencarian urutan, yaitu:

* + 1. Linier probing
    2. Quadratic probing
    3. Double hashing

Berikut ini listing program implementasi dari struktur data Hash Table dengan fungsi hash modulo key dengan table-size, dan menggunakan teknik linier probing.

Pada linier probing, ketika terjadi collision maka akan mencari posisi terdekat dari posisi seharusnya pada table. Teknik ini hanya disarankan jika ukuran table dua kali lipat dari jumlah data.

|  |
| --- |
| public class Data { private int data;  public Data(int data) { this.data = data;  }  public int getKey() { return data;  }  } |
| public class HashTable {  private Data[] hashArray; private int size; |

|  |  |
| --- | --- |
| public HashTable(int size) { this.size = size;  hashArray = new Data[size];  }  public void displayTable() { System.out.println("Table: "); for (int j = 0; j < size; j++) {  if (hashArray[j] != null) { System.out.println(" | "+j+"\t | "  +hashArray[j].getKey() + " |");  } else {  System.out.println(" | "+j+"\t | -- |");  }  }  System.out.println("");  } | |
| public int hashFunc(int key) { return key % size;  } |  |
| 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;  } |  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| 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;  } |  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| 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;  }  } //akhir class HashTable |  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

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!



***jawaban***

* 1. 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!



***jawaban***

* 1. 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!



***jawaban***

* 1. Lakukan pencarian berdasarkan key tertentu pada class HashTableApp!, jelaskan bagaimana proses pencarian pada method find?



***jawaban***

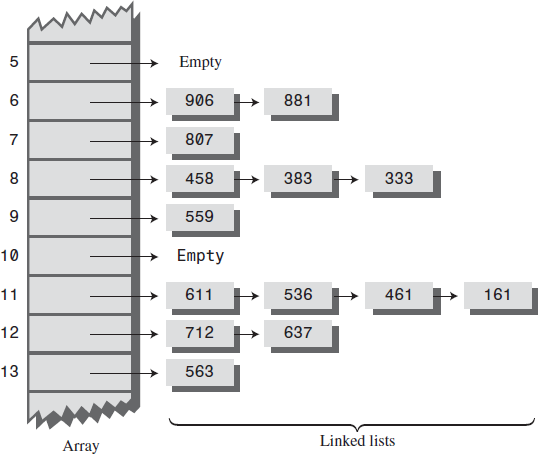
* 1. Lakukan hapus item berdasarkan key tertentu pada class HashTableApp!, jelaskan bagaimana proses pencarian pada method delete?



***jawaban***

# Separate Chaining / Open Hashing

Pada teknik ini, permasalahan collision diselesaikan dengan menambah seluruh elemen yang memiliki nilai sama pada sebuah sel. Hal ini dilakukan dengan cara menyediakan sebuah linked list untuk setiap elemen yang memiliki nilai hash yang sama. Tiap sel pada hash table memiliki sebuah lisked list yang berisi data/elemen.



Gambar 8.1 Contoh Separate Chaining

Berikut ini listing program Hash Table dengan teknik Separate Chaining

|  |
| --- |
| public class Link {  private int iData; public Link next;  public Link(int iData) { this.iData = iData;  }  public int getKey() { return iData;  }  public void displayLink() { System.out.print(iData + " ");  }  } |
| public class SortedLinkList { private Link first;  public SortedLinkList() { first = null;  }  public void insert(Link theLink) { int key = theLink.getKey(); Link previous = null;  Link current = first;  while (current != null && key > current.getKey()) { previous = current;  current = current.next;  }  if (previous == null) { first = theLink;  } else {  previous.next = theLink;  }  theLink.next = current;  } |

|  |  |
| --- | --- |
| public void delete(int key) { Link previous = null; Link current = first;  while (current != null && key != current.getKey()) { previous = current;  current = current.next;  }  if (previous == null) { first = first.next;  } else {  previous.next = current.next;  }  }  public Link find(int key) { Link current = first;  while (current != null && current.getKey() <= key) { if (current.getKey() == key) {  return current;  }  current = current.next;  }  return null;  }  public void displayList() { System.out.print("List (first-->last): "); Link current = first;  while (current != null) { current.displayLink(); current = current.next;  }  System.out.println("");  }  } | |
| public class HashTable {  private SortedLinkList[] hashArray; private int size;  public HashTable(int size) { this.size = size;  hashArray = new SortedLinkList[size]; for (int i = 0; i < size; i++) {  hashArray[i] = new SortedLinkList();  }  }  public void displayTable() { System.out.println("Table: "); for (int j = 0; j < size; j++) {  System.out.print(" "+j + ". "); hashArray[j].displayList();  }  } | |
| public int hashFunc(int key) { return key % size;  } |  |

|  |  |
| --- | --- |
| public void insert(int data) { Link theLink=new Link(data); int key = theLink.getKey(); int hashVal = hashFunc(key);  hashArray[hashVal].insert(theLink);  } |  |
| public void delete(int key) { int hashVal = hashFunc(key);  hashArray[hashVal].delete(key);  } |  |
| public Link find(int key) {  int hashVal = hashFunc(key); Link theLink =  hashArray[hashVal].find(key); return theLink;  }  } |  |

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!



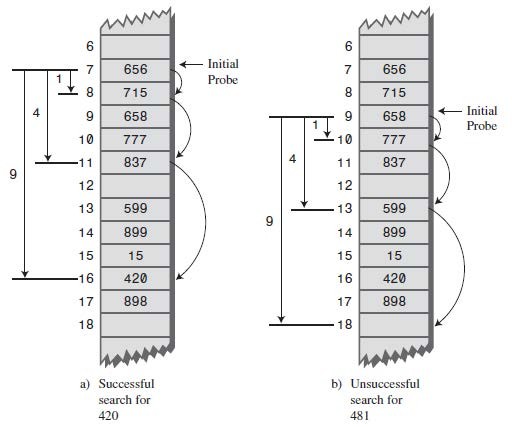
***jawaban***

# PRAKTIKUM

Pada open addressing, untuk menangani collision selain linier probing, terdapat pula teknik quadratic probing dan double hashing.

* 1. NIM genap: mengimplementasikan teknik quadratic probing

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.



Gambar 8.2 Ilustrasi quadratic probe

* 1. NIM ganjil: mengimplementasikan teknik double hashing

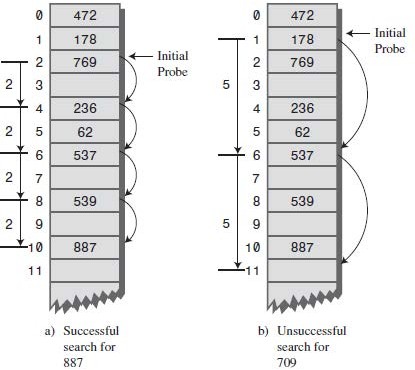
Collision resolution dilakukan dengan melakukan hash terhadap key menggunakan fungsi hash yang lain dan menggunakan hasilnya sebagai jarak penentuan sel. Berikut ini fungsi yang dapat digunakan sebagai fungsi hash kedua:

hashFunc2(key) = c – (key % c)

dimana c adalah konstanta bilangan prima yang lebih kecil daripada ukuran table, misalnya:

hashFunc2(key) = 5 – (key% 5)

hasil dari fungsi hast tersebut sebagai jarak penentuan sel, sehingga hashValue = hashFunc1(key) + hashFunc2(key)



Gambar 8.3 ilustrasi double hashing

# KESIMPULAN

Kesimpulan yang diperoleh dari pembahasan praktikum kali ini adalah:

* 1. Tentang Hash Table
  2. Tentang Perbedaan Open Addressing dan Separate Chain
  3. Tentang perbedaan linier probing, quadratic probing, dan double hashing sebagai collision resolution