MODUL PRAKTIKUM
STRUKTUR DATA

Semester Ganjil Tahun Ajaran 2022/2023

Dosen:

Khadijah Fahmi Hayati Holle, M.Kom

NIP. 19900626 202203 2 002

IDENTITAS PRAKTIKAN

NIM : _____

Nama Lengkap : _____

Kelas/ Hari/ Jam : _____ / _____ / _____

Nomor komputer : _____

ASISTEN

NAMA _____ (NIM. _____)

NAMA _____ (NIM. _____)

JURUSAN TEKNIK INFORMATIKA

FAKULTAS SAINS DAN TEKNOLOGI

UIN MAULANA MALIK IBRAHIM MALANG

OUTLINE MATERI
PRAKTIKUM STRUKTUR DATA

Praktikum 1	**Arrays**
Praktikum 2	**Simple Sorting**
Praktikum 3	**Stack and Queue**
Praktikum 4	**Linked List**
Praktikum 5	**Recursion**
Praktikum 6	**Advance Sorting**
Middle Test	
Praktikum 7	**Binary Tree**
Praktikum 8	**Hash Table**
Praktikum 9	**Heap**
Praktikum 10	**Graph**
Progres	
Progres	
Final Test	

PRAKTIKUM 1 - STRUKTUR DATA

ARRAYS

Learning outcomes:

1. Mampu menjelaskan konsep dan implementasi array pada program
2. Mampu melakukan manipulasi data array: menambahkan item, melakukan pencarian, dan menghapus item pada array
3. Mampu mengimplementasikan *ordered array* pada program.
4. Mampu mengimplementasikan *binary search* pada *ordered array*.
5. Mampu menyimpan dan manipulasi objek pada array.

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

A. PENDAHULUAN

1. Format penulisan code yang digunakan untuk mendeklarasikan sebuah array adalah:

```
TipeData namaVariable[ ] = new TipeData[panjang/ukuranArray];
```

Sedangkan format penulisan code untuk menambahkan item pada array yang telah dideklarasikan adalah:

```
namaVariable[index]= value;
```

Tulis dan jalankan listing program berikut:

```
public class classArray {

    public static void main(String[] args) {
        int[] array = new int[10];
        array[0] = 10;
        array[1] = 20;
        array[2] = 30;
        array[3] = 40;
        array[4] = 50;

        for (int i = 0; i < array.length; i++) {
            System.out.print(array[i] + " ");
        }
        System.out.println("");

    }
}
```

Pada listing program tersebut, ukuran array yang dideklarasikan adalah 10. *Insert* item dilakukan hingga index ke-4, artinya hanya terdapat 5 item.
Tuliskan output program tersebut dan jelaskan kenapa demikian!

jawaban

2. Tambahkan baris code berikut ini pada listing program nomer 1

```
...

array = new int[20];

for (int i = 0; i < array.length; i++) {
    System.out.print(array[i] + " ");
}
System.out.println("");

} //akhir method main
}//akhir class
```

Jalankan program tersebut, apa output yang dihasilkan?

jawaban

Apakah item yang awal dimasukkan (pada listing no 1) masih tersimpan didalam array? Jelaskan kenapa demikian?

jawaban

Hingga tahap ini, yang dapat disimpulkan adalah:

Ukuran array bersifat *fixed* / *not fixed* *)

**)coret salah satu*

3. Lakukan *experiment* menggunakan listing nomer 1 untuk menjawab pertanyaan berikut (beri keterangan benar/salah untuk soal berupa statemen dan tulis jawaban untuk soal isian)

Insert item pada array hanya bisa dilakukan secara berurutan mulai index ke-0.	
Insert item pada array hanya bisa dilakukan hingga ukuran array – 1.	
Cell array untuk semua tipe data *primitive* yang belum diberi value secara *default* bernilai 0.	
Cell array untuk tipe data *String* yang belum diberi value secara *default* bernilai *null*.	
Keterangan yang muncul jika memasukkan item melebihi ukuran array adalah ……	

4. Lengkapi listing berikut:

```
run:
30 20 60 70 50 10
BUILD SUCCESSFUL (total time: 1 second)
```

```
public class ClassArray {

    public static void main(String[] args) {
        int[] array = new int[100];
        int nElemen = 0;
        array[0] = 30;
        array[1] = 20;
        array[2] = 60;
        array[3] = 70;
        array[4] = 50;
        array[5] = 10;
        [                    ]

        for (int i = 0; i < [          ]; i++) {
            System.out.print(array[i] + " ");
        }
        System.out.println("");

    }
}
```

Jalankan dan tuliskan penjelasan dari listing yang telah Anda lengkapi!

jawaban

5. Berikut ini adalah listing program array yang dituliskan dalam bentuk *object oriented programming*. Class *HighArray* memiliki method untuk manipulasi array, yaitu *insert*, *find/search*, dan *delete* serta method *display* untuk menampilkan isi array. Method dalam class HighArray tersebut dipanggil dan dijalankan pada class HighArrayApp.
   Pahami listing berikut dengan menulis dan menjalankannya, kemudian tuliskan penjelasan tiap barisnya!

`class HighArray {`	*Awal sebuah kelas bernama HighArray*
`private int[] arr;`	*Deklarasi variable integer bertipe array bernama "arr" dengan akses private*
`private int nElemen;`	
`public HighArray(int max) {`	
`arr = new int[max];`	
`nElemen = 0;`	
`}`	
`public void insert(int value) {`	
`arr[nElemen] = value;`	
`nElemen++;`	
`}`	
`public boolean find(int key) {`	
`int i;`	
`for (i = 0; i < nElemen; i++) {`	
`if (arr[i] == key) {`	
`break;`	
`}`	
`}`	
`if (i == nElemen) {`	
`return false;`	
`} else {`	
`return true;`	
`}`	
`}`	
`public boolean delete(int value) {`	

```
        int i;
        for (i = 0; i < nElemen; i++) {
            if (value == arr[i]) {
                break;
            }
        }
        if (i == nElemen) {
            return false;
        } else {
          for (int j = i; j < nElemen; j++) {
                arr[j] = arr[j + 1];
            }
            nElemen--;
            return true;
        }
    }

    public void display() {
        for (int i = 0; i < nElemen; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println("");
    }
}

public class HighArrayApp {

    public static void main(String[] args) {
        int maxSize = 100;
        HighArray arr;
        arr = new HighArray(maxSize);

        arr.insert(70);
        arr.insert(80);
        arr.insert(75);
        arr.insert(55);
        arr.insert(85);
        arr.insert(25);
        arr.insert(30);
        arr.insert(00);
        arr.insert(90);
        arr.insert(40)

        arr.display();
```

```
        int key = 25;
        if (arr.find(key)) {
            System.out.println(key +
                    " ditemukan");
        } else {
            System.out.println(key +
                    " tidak ditemukan");
        }

        arr.delete(00);
        arr.delete(80);
        arr.delete(55);

        arr.display();
    }
}
```

Output program tersebut adalah….

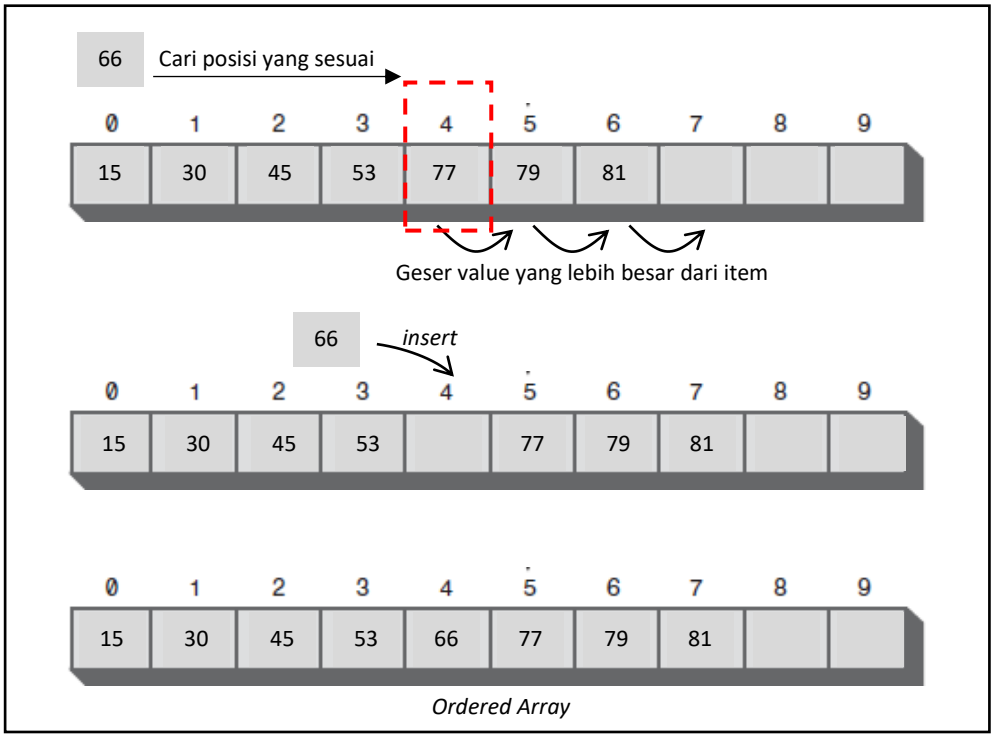
jawaban

6. Tambahkan sebuah method *size* pada class *HighArray* yang mempu mengembalikan nilai jumlah elemen array. Panggil method tersebut pada class *HighArrayApp* untuk menampilkan jumlah elemen.
   Tulis code dan penjelasannya!

jawaban

B. PRAKTIKUM

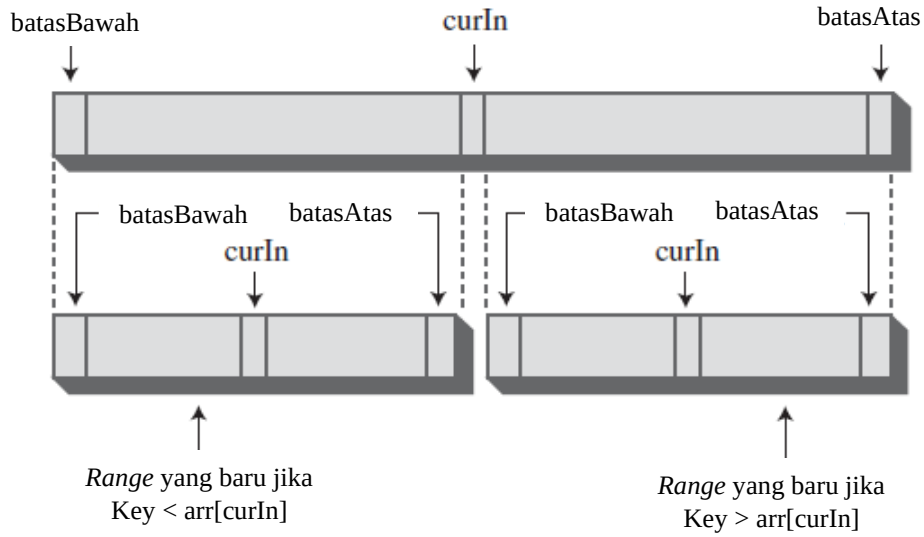
1. Pada listing nomer 5 (tugas pendahuluan), method *insert* digunakan untuk menambahkan item pada *cell* yang belum terisi tanpa memberhatikan *value* item yang ditambahkan sehingga elemen pada array disimpan secara tidak berurutan (*unordered*).
   Agar item dapat disimpan pada urutan yang sesuai dengan *value*-nya maka perlu dilakukan pencarian posisi *cell* yang tepat bagi item yang akan dimasukkan dengan cara membandingkan tiap item pada *cell* dengan item yang akan dimasukkan, yaitu pencarian secara *linier*. Setelah *cell* tepat ditemukan, langkah selanjutnya adalah menyiapkan *cell* tersebut untuk diisi jika sudah ada item yang tersimpan pada *cell* tersebut. Hal ini bisa dilakukan dengan cara menggeser item yang memiliki value lebih besar dari item yang akan dimasukkan, dengan demikian terdapat *cell* kosong untuk diisi dengan item baru. Langkah-langkah insert item pada ordered array ditunjukkan pada Gambar 1.1 berikut ini.



Gambar 1.1 Langkah *insert* item pada *ordered array*

Tuliskan listing untuk method *insert* untuk menyimpan elemen array secara berurutan (*ordered*)!

2. Pencarian pada method *find* listing nomer 5 (tugas pendahuluan) menggunakan *linier search*, artinya terhadap key yang dicari, program akan melakukan pencarian pada array secara berurutan mulai dari elemen pertama hingga elemen terakhir. Hal ini tidak efisien. Pada *ordered array*, dapat dilakukan pencarian menggunakan *binary serach* yang lebih efisien dibandingkan dengan *linier search*.
   Pada *binary search*, *range* elemen array dibagi dua secara berulang-ulang. Hal ini menjadikan *range* pencarian semakin kecil dan terpusat pada item yang memiliki *value* mendekati *key* pencarian. Pembagian *range* pencarian pada *binary search* ditunjukkan pada Gambar 1.2 berikut ini.



Gambar 1.2 Pembagian *range* pencarian pada *binary Search*

Tuliskan listing untuk method *find* yang mengimplementasikan *binary search*!

3. *Storing object*

Item data pada *real world* tidak direpresentasikan dalam bentuk data *primitive* tapi berupa *record* yang merupakan kombinasi dari beberapa *field*. Misalkan untuk *record* personal, kita dapat menyimpan nama, tempat tanggal lahir, nomer telpon, email, dsb. Untuk data mahasiswa, kita dapat menyimpan nim, nama, jurusan, asal, dsb. Dalam java, *record* data biasanya direpresentasikan dengan sebuah *class object*.

Berikut ini listing yang menunjukkan implementasi *storing object*. Terdapat tiga class, yaitu class "Mahasiswa", "DataArray", dan "DataArrayApp". *Record* yang disimpan adalah data mahasiswa yang terdiri dari *field* nim, nama, dan asal. *Record* mahasiswa ini direpresentasikan dalam sebuah *class object* dengan nama "Mahasiswa".

Tulis dan pahamilah listing program untuk menyimpan object berikut ini.

```
class Mahasiswa {

    private long nim;
    private String nama;
    private String asal;

    public Mahasiswa(long nim, String nama, String asal) {
        this.nim = nim;
        this.nama = nama;
        this.asal = asal;
    }

    public void displayMhs() {
        System.out.print("\tNIM: " + nim);
        System.out.print(", Nama: " + nama);
        System.out.println(", Asal: " + asal);
    }

    public long getNim() {
        return nim;
    }
}
```

```
public class DataArray {

    private Mahasiswa[] mhs;
    private int nElemen;

    public DataArray(int max) {
        mhs = new Mahasiswa[max];
        nElemen = 0;
    }

    public Mahasiswa find(long searchNim) {
        int i;
        for (i = 0; i < nElemen; i++) {
            if (mhs[i].getNim()==searchNim) {
                break;
            }
        }
        if (i == nElemen) {
            return null;
        } else {
            return mhs[i];
        }
    }

    public void insert(long nim, String nama, String asal) {
        mhs[nElemen] = new Mahasiswa(nim, nama, asal);
        nElemen++;
    }

    public boolean delete(long searchNim) {
        int i;
        for (i = 0; i < nElemen; i++) {
            if (mhs[i].getNim()==searchNim) {
                break;
            }
        }
        if (i == nElemen) {
            return false;
        } else {
            for (int j = i; j < nElemen; j++) {
                mhs[j] = mhs[j + 1];
            }
            nElemen--;
            return true;
        }
    }
    public void displayArray(){
        for (int i = 0; i < nElemen; i++) {
            mhs[i].displayMhs();
        }
    }
}
```

Objek mahasiswa disimpan dalam array. Class “DataArray” berisi method-method untuk manipulasi object mahasiswa, yaitu *insert*, *find*, dan *delete*, serta method untuk menampilkan array berisi objek mahasiswa, yaitu *displayArray*.
Class yang digunakan untuk menjalankan program adalah class “DataArrayApp”. Class ini memiliki method main yang didalamnya terdapat listing untuk memanggil dan menjalankan fungsi-fungsi pada class DataArray yang telah dibuat.

```
public class DataArrayApp {

    public static void main(String[] args) {
        int maxSize = 100;
        DataArray arr;
        arr = new DataArray(maxSize);
        arr.insert(16650200, "Jundi", "Malang");
        arr.insert(16650210, "Ahmad", "Sidoarjo");
        arr.insert(16650220, "Ismail", "Banyuwangi");
        arr.insert(16650230, "Sofi", "Semarang");
        arr.insert(16650240, "Dinda", "Bandung");
        arr.insert(16650250, "Rais", "Ambon");
        arr.insert(16650260, "Helmi", "Madura");
        arr.insert(16650270, "Agung", "Madiun");
        arr.insert(16650280, "Arina", "Malang");

        arr.displayArray();

        long searchKey = 16650270;
        Mahasiswa mhs = arr.find(searchKey);
        if (mhs != null) {
            System.out.print("\nketemu");
            mhs.displayMhs();
        } else {
            System.out.println("ga ketemu " + searchKey);
        }

        searchKey=16650240;
        System.out.println("\nhapus nim: "+searchKey);
        arr.delete(searchKey);

        arr.displayArray();
    }
}
```

Output:

```
run:
        NIM: 16650200, Nama: Jundi, Asal: Malang
        NIM: 16650210, Nama: Ahmad, Asal: Sidoarjo
        NIM: 16650220, Nama: Ismail, Asal: Banyuwangi
        NIM: 16650230, Nama: Sofi, Asal: Semarang
        NIM: 16650240, Nama: Dinda, Asal: Bandung
        NIM: 16650250, Nama: Rais, Asal: Ambon
        NIM: 16650260, Nama: Helmi, Asal: Madura
        NIM: 16650270, Nama: Agung, Asal: Madiun
        NIM: 16650280, Nama: Arina, Asal: Malang

ketemu  NIM: 16650270, Nama: Agung, Asal: Madiun

hapus nim: 16650240
        NIM: 16650200, Nama: Jundi, Asal: Malang
        NIM: 16650210, Nama: Ahmad, Asal: Sidoarjo
        NIM: 16650220, Nama: Ismail, Asal: Banyuwangi
        NIM: 16650230, Nama: Sofi, Asal: Semarang
        NIM: 16650250, Nama: Rais, Asal: Ambon
        NIM: 16650260, Nama: Helmi, Asal: Madura
        NIM: 16650270, Nama: Agung, Asal: Madiun
        NIM: 16650280, Nama: Arina, Asal: Malang
BUILD SUCCESSFUL (total time: 1 second)
```

C. KESIMPULAN

Kesimpulan yang diperoleh dari pembahasan praktikum kali ini adalah:

1. Tentang *unordered arrays* dan *ordered arrays*

2. Tentang *linier search* dan *binary search*

3. Tentang menyimpan object (*storing object*)

PRAKTIKUM 2 - STRUKTUR DATA

SIMPLE SORTING

Learning outcomes:

1. Mampu menjelaskan langkah-langkah *sorting* algoritma *bubble sort*, *insertion sort*, dan *selection sort*.
2. Mampu mengimplementasikan algoritma *bubble sort*, *insertion sort*, dan *selection sort* pada program.
3. Mampu mengimplementasikan *sorting object* menggunakan algoritma *bubble sort*, *insertion sort*, dan *selection sort.*
4. Mampu menjelelaskan perbandingan *sorting* algoritma: *bubble sort*, *insertion sort*, dan *selection sort*.

Tiga algoritma *sorting* sederhana yang dibahas pada modul ini adalah *bubble sort*, *selection sort*, dan *insertion sort*. Ketiga algoritma tersebut menggunakan dua langkah/tindakan yang terus dilakukan secara berulang hingga data berurutan. Dua tindakan tersebut adalah:

1. membandingkan (*compare*) dua item
2. menukar (*swap*) dua item, atau menyalin (*copy*) satu item

A. PENDAHULUAN

1. Algoritma *sorting* yang paling sederhana adalah ***bubble sort***. Langkah *sorting* secara *ascending* menggunakan *bubble sort* adalah sebagai berikut:
   a. Bandingkan dua item
   b. Jika item pertama (sebelah kiri) lebih besar daripada item kedua (sebelah kanan), maka tukar kedua item tersebut
   c. Pindah ke posisi kanan. Lakukan langkah a dan b hingga posisi sampai di batas akhir
   d. Ketika satu item telah berada sesuai urutan, ulangi langkah a-c hingga semua item sesuai urutan.

```
...
public void BubbleSort() {
int batas, i;
for (batas = nElemen-1; batas>0; batas--) {
    for (i = 0; i < batas; i++) {
        if (arr[i] > arr[i + 1]) {
            swap(i, i + 1);
        }
    }
}
}


public void swap(int one, int two) {
    int temp = arr[one];
    arr[one] = arr[two];
    arr[two] = temp;
}
...
```

Listing diatas adalah baris kode yang mengimplementasikan tiap langkah *bubble sort*. Implementasikan baris kode tersebut ke dalam sebuah program sehingga dapat digunakan untuk mengurutkan elemen array. Anda dapat menambahkan listing tersebut pada class *HighArray* (listing 5 praktikum 1) yang dipanggil pada class

HighArrayApp. Atau anda juga dapat menuliskan listing ini pada satu class secara terstruktur.
Insert 6 item pada array program tersebut. Tampilkan isi array sebelum dilakukan pengurutan. Kemudian urutkan dan tampilkan isi array setelah pengurutan.
Jalankan program yang telah Anda buat dan tuliskan output program tersebut.

jawaban

2. Untuk sorting array yang memiliki 6 item, berapa jumlah perbandingan item yang dilakukan hingga semua item sesuai urutan?

jawaban

3. Pada program nomer 1, tambahkan kode untuk menampilkan isi array setelah baris kode pertukaran item (`swap(i, i + 1);`). Jalankan kembali dan amati bagaimana proses pengurutan pada tiap iterasi. Tuliskan isi array pada 10 perulangan pertama. Jelaskan!

jawaban

4. Jika Anda ingin melakukan *sorting* menggunakan algoritma *bubble sort* secara *descending*, maka pada program nomer 1 bagian manakah yang harus diganti. Tuliskan code-nya dan jelaskan!

jawaban

5. Algoritma *sorting* yang lain adalah ***selection sort***. Langkah *selection sort* untuk pengurutan secara *ascending* yaitu:
   a. Cari item terkecil pada array
   b. Letakkan item terkecil sesuai urutannya dengan cara menukar item terkecil dengan item pada index awal
   c. Geser posisi awal pencarian ke kanan.
   d. Lakukan langkah a, b, dan c hingga semua item terurut.

   Berikut ini listing untuk algoritma *selection sort*. Lengkapi sebagaimana soal nomor 1, gunakan data array yang sama, jalankan dan jelaskan tiap baris code pengurutan berikut ini!

```
...
public void SelectionSort() {
   int awal, i, min;


   for (awal=0; awal< nElemen-1; awal++) {
      min = awal;
      for (i = awal + 1; i < nElemen; i++) {
           if (arr[i] < arr[min]) {
                min = i;
           }
      }
      swap(awal, min);
   }
}
...
```

6. Pada listing nomor 5, tambahkan kode untuk menampilkan isi elemen pada array setelah kode pertukaran (`swap(awal, min);`). Jalankan program, amati dan tulis outputnya, kemudian jelaskan!

jawaban

7. Sedikit berbeda dengan dua algoritma sebelumnya, pada ***insertion sort***, aksi yang dilakukan adalah membandingkan dan meng-*copy* item. Berikut ini langkah *insertion sort* untuk pengurutan secara *ascending*:
   a. Tandai sebuah item sebagai batas antara *partially sorted* dan *unsorted items*.
   b. Geser item pada *partially sorted* yang bernilai lebih besar dari pada item yang ditandai pada langkah a.
   c. Sisipkan item tersebut pada posisi yang sesuai di bagian *partially sorted*.
   d. Ulangi langkah a-c hingga semua *unsorted items* telah disisipkan (*insert*) ke *sorted group*.

   Berikut ini listing untuk algoritma *insertion sort*. Lengkapi sebagaimana soal nomor 1, gunakan data array yang sama, jalankan dan jelaskan tiap baris code pengurutan berikut ini!

`...`	
`public void InsertionSort() {`	
`  int i, curIn;`	
`  for (curIn= 1; curIn < nElemen; curIn++) {`	
`      int temp = arr[curIn];`	
`      i = curIn;`	

```
        while (i > 0 && arr[i - 1] >= temp) {
                arr[i] = arr[i - 1];
                i--;
        }
        arr[i] = temp;
    }
}
...
```

8. Pada listing nomor 7, tambahkan kode untuk menampilkan isi elemen pada array setelah kode pergeseran item (`arr[i] = arr[i - 1];`) dan setelah kode insert item (`arr[i] = temp;`). Jalankan program, amati dan tulis outputnya hingga 4 kali tahap penyisipan, kemudian jelaskan!

jawaban

B. PRAKTIKUM

1. *Sorting Object*

   Pada praktikum 1 (Arrays), Anda telah mengiimplementasikan *storing object* "mahasiswa". Implementasikan algoritma *sorting* untuk *sorting object* "Mahasiswa" berdasarkan NIM. Yaitu dengan cara menambahkan method *BubbleSort( ), SelectionSort( ),* dan *InsertionSort( )* sebagaimana yang dituliskan pada tugas pendahuluan modul ini (dengan sedikit penyesuaian) pada class *DataArray* (listing nomer 3 parktikum 1). Method tersebut dipanggil pada class *DataArrayApp*.

   Keterangan:
   - Praktikan dengan **NIM genap** mengerjakan *BubbleSort( )* dan *SelectionSort( )*
   - Praktikan dengan **NIM ganjil** mengerjakan *BubbleSort( )* dan *InsertionSort( )*

2. *Lexicographical Comparisons*

   NIM pada object mahasiswa adalah variable bertipe long. Anda dapat membandingkan item dengan tipe long dengan menggunakan operator *equality relational*. Pada *sorting object*, kita perlu mengetahui cara membandingkan item dengan tipe String (object). Misal, adakalanya pencarian *record* mahasiswa dapat menggunakan *field* "nama" sebagai *key*. Untuk kondisi ini, kita dapat menggunakan method **`compareTo()`** yang ada pada class String.

   Method `compareTo()` membandingkan dua string secara leksikograf (alfabetis). Perbandingan ini didasarkan pada nilai Unicode dari masing-masing karakter dalam String. Deretan karakter objek String ini dibandingkan secara leksikograf dengan deretan karakter string argumen.

   Value yang dikembalikan dalam pemanggilan method ini adalah nilai 0 jika objek string sama dengan string argumen; nilai kurang dari 0 jika objek string secara leksikografis kurang dari string argumen; dan nilai lebih besar dari 0 jika objek string secara leksikograf lebih besar dari string argumen. (lihat Tabel 2.1)

   Tabel 2.1 Operasi pada method `compareTo()`

`StringObjek.compareTo(StringArgumen)`	**Return value**
`StringArgumen` < `StringObjek`	< 0
`StringArgumen` equal `StringObjek`	0
`StringArgumen` > `StringObjek`	> 0

   Variasi lain dari method `compareTo()` adalah `compareToIgnoreCase()`. Method ini memiliki fungsi yang sama dengan method `compareTo()` namun dengan mengabaikan besar/kecilnya huruf.

   Gunakan method `compareTo()` atau `compareToIgnoreCase()` pada tahap perbandingan item untuk melakukan *sorting object* "mahasiswa" berdasarkan *field* "nama"!

   Keterangan:
   - Praktikan dengan **NIM genap** mengerjakan *InsertionSortbyName( )*
   - Praktikan dengan **NIM ganjil** mengerjakan *SelectionSortbyName( )*

```
run:
Data mahasiswa sebelum diurutkan
        NIM: 16650270, Nama: Agung, Asal: Madiun
        NIM: 16650230, Nama: Sofi, Asal: Semarang
        NIM: 16650280, Nama: Arina, Asal: Malang
        NIM: 16650260, Nama: Helmi, Asal: Madura
        NIM: 16650220, Nama: Ismail, Asal: Banyuwangi
        NIM: 16650240, Nama: Dinda, Asal: Bandung
        NIM: 16650250, Nama: Rais, Asal: Ambon
        NIM: 16650200, Nama: Jundi, Asal: Malang
        NIM: 16650210, Nama: Ahmad, Asal: Sidoarjo

1. Sorting Mahasiswa by NIM
        NIM: 16650200, Nama: Jundi, Asal: Malang
        NIM: 16650210, Nama: Ahmad, Asal: Sidoarjo
        NIM: 16650220, Nama: Ismail, Asal: Banyuwangi
        NIM: 16650230, Nama: Sofi, Asal: Semarang
        NIM: 16650240, Nama: Dinda, Asal: Bandung
        NIM: 16650250, Nama: Rais, Asal: Ambon
        NIM: 16650260, Nama: Helmi, Asal: Madura
        NIM: 16650270, Nama: Agung, Asal: Madiun
        NIM: 16650280, Nama: Arina, Asal: Malang

2. Sorting Mahasiswa by NAME
        NIM: 16650270, Nama: Agung, Asal: Madiun
        NIM: 16650210, Nama: Ahmad, Asal: Sidoarjo
        NIM: 16650280, Nama: Arina, Asal: Malang
        NIM: 16650240, Nama: Dinda, Asal: Bandung
        NIM: 16650260, Nama: Helmi, Asal: Madura
        NIM: 16650220, Nama: Ismail, Asal: Banyuwangi
        NIM: 16650200, Nama: Jundi, Asal: Malang
        NIM: 16650250, Nama: Rais, Asal: Ambon
        NIM: 16650230, Nama: Sofi, Asal: Semarang
BUILD SUCCESSFUL (total time: 1 second)
```

Gambar 2.1 Contoh output *sorting object* mahasiswa berdasarkan *field* NIM dan nama

C. KESIMPULAN

Kesimpulan yang diperoleh dari pembahasan praktikum kali ini adalah:

1. Tentang perbandingan algoritma sorting *bubble*, *selection*, dan *insertion*

2. Tentang *sorting object*

PRAKTIKUM 3 - STRUKTUR DATA

STACKS AND QUEUES

Learning outcomes:

1. Mampu menjelaskan konsep LIFO pada Stacks dan FIFO pada Queues
2. Mampu mengimplementasikan operasi dasar Stack dan Queue (*push*, *pop*, *peek*) dengan struktur data Array.
3. Mampu menjelaskan implementasi Stack pada *parsing arithmetic expressions.*
4. Mampu menjelaskan dan mengimplementasikan *circular* Queue.
5. Mampu mengimplementasikan struktur data Stacks dan Queues pada program.

Struktur data yang dibahas pada praktikum kali ini adalah Stacks dan Queues. Selain itu, ada pula struktur data yang sejenis yaitu priority queue (tidak dibahas pada praktikum ini). Fungsi ketiga struktur data ini lebih sering digunakan sebagai *programmer's tool*. Yaitu, digunakan sebagai alat bantu konseptual penyimpanan data, bukan sebagai penyimpanan data itu sendiri.

Stack, queue, dan priority queue lebih abstrak dibandingkan dengan struktur penyimpanan data seperti arrays dan beberapa yang lainnya. Mekanisme mendasar untuk mengimplementasikan stack, queue, dan priority queue dapat berupa Arrays sebagaimana yang ditunjukkan pada pembahasan modul ini. Selain itu, dapat juga menggunakan "Linked list". Mekanisme mendasar untuk Priority queue dapat juga berupa salah satu jenis khusus dari struktur data *tree* yang disebut dengan *heap*.

Pada struktur data Arrays, semua item dapat diakses, sedangkan pada stack, queue, dan priority queue akses tersebut dibatasi, yaitu hanya satu item yang dapat diakses untuk dibaca atau dihapus.

A. PENDAHULUAN

1. Stacks (tumpukan) merupakan suatu susunan koleksi data dimana data yang dapat ditambahkan dan dihapus selalu dilakukan pada bagian akhir data, yang disebut dengan *top of stack*. Dengan kata lain, stack hanya mengijinkan akses pada item yang terakhir dimasukkan.
   Stacks bersifat LIFO (*Last In First Out*). Jelaskan sifat LIFO pada stacks dan gambarkan skema lengkap dari LIFO!

jawaban

2. Operasi utama pada Stacks yaitu *push* dan *pop*. Selain dua operasi tersebut, juga terdapat operasi *peek* pada Stacks. Jelaskan masing-masing dari tiga operasi tersebut!

jawaban

3. Tulislah listing program berikut ini dan jelaskan tiap barisnya!

```
class Stack {


    private int maxSize;
    private long[] stackArray;
    private int top;


    public Stack(int size) {
        maxSize = size;
        stackArray = new long[maxSize];
        top = -1;
    }


    public void push(long item) {
        stackArray[++top] = item;
    }


    public long pop() {
        return stackArray[top--];
    }

```

```
    public long peek() {
        return stackArray[top];
    }

    public boolean isEmpty() {
        return (top == -1);
    }

    public boolean isFull() {
        return (top == maxSize - 1);
    }
}

public class StackApp {

    public static void main(String[] args) {
        Stack theStack = new Stack(10);
        System.out.println(">> push some
                           items");
        theStack.push(20);
        theStack.push(40);
        theStack.push(60);
        theStack.push(80);

        System.out.println("\n>> pop items
                           in the stack");
        while (!theStack.isEmpty()) {
            long value = theStack.pop();
            System.out.print(value + " ");
        }
    }
}
```

Apa output dari listing program tersebut? Jelaskan!

jawaban

4. Setelah anda memahami tiap baris listing nomer 3. Tuliskan kesimpulan logika untuk setiap method pada class Stack: method `push()`, `pop()`, `peek()`, `isEmpty()`, dan `isFull()`!

jawaban

5. Salah satu aplikasi yang menggunakan struktur penyimpanan stack adalah parsing ekspresi aritmatika. Tuliskan langkah manual untuk merubah notasi *infix*:

   U * (I + N) / M ^ L – G

   menjadi notasi *postfix* menggunakan teknik stack!

jawaban

6. Queues (antrian) adalah struktur data yang hampir mirip dengan stack. Perbedaannya adalah pada queues, akses item bagi yang pertama dimasukkan. Queues bersifat FIFO (*First In First Out*). Jelaskan sifat FIFO pada queues dan gambarkan skema lengkap dari FIFO!

jawaban

7. Berikut ini listing Queue dengan Array. Tulis dan jelaskan!

```
class Queue {
    private int maxSize;
    private long[] queArray;
    private int front;
    private int rear;
    private int nItems;

    public Queue(int size) {
        this.maxSize = size;
        queArray = new long[maxSize];
        front = 0;
        rear = -1;
        nItems = 0;
    }

    public void insert(long value) {
        if (rear == maxSize - 1) {
            rear = -1;
        }
        queArray[++rear] = value;
        nItems++;
    }

    public long remove() {
        long temp = queArray[front++];
        if (front == maxSize) {
            front = 0;
        }
        nItems--;
        return temp;
    }
```

```
    public long peekFront() {
        return queArray[front];
    }

    public boolean isEmpty() {
        return (nItems == 0);
    }

    public boolean isFull() {
        return (nItems == maxSize);
    }

    public int size() {
        return nItems;
    }
}

public class QueueApp {
    public static void main(String[] args) {
        Queue theQueue = new Queue(5);
        theQueue.insert(10);
        theQueue.insert(20);
        theQueue.insert(30);
        theQueue.insert(40);
        theQueue.remove();
        theQueue.remove();
        theQueue.remove();
        theQueue.insert(50);
        theQueue.insert(60);
        theQueue.insert(70);
        theQueue.insert(80);

        while (!theQueue.isEmpty()){
```

```
            long n = theQueue.remove();
            System.out.print(n);
            System.out.print(" ");
        }
        System.out.println("");
    }
}
```

Tuliskan output dari listing tersebut dan jelaskan!

jawaban

8. Setelah anda memahami tiap baris listing nomer 7. Tuliskan kesimpulan logika untuk setiap method pada class Queue: method `insert()`, `remove()`, `peek()`, `isEmpty()`, dan `isFull()`!

jawaban

9. Apa jenis queue (*linier/circular*) yang diimplementasikan pada listing tersebut? Beri alasan/bukti dari jawaban anda!

jawaban

B. PRAKTIKUM

1. Implementasi Stack

   Salah satu contoh program sederhana yang mengimplementasikan stack adalah program pembalik kata. Stack digunakan untuk membalik huruf. Langkah pertama, tiap katakter pada String input diekstrak dan dimasukkan kedalam stack. Kemudian tiap karakter tersebut dikeluarkan dan ditampilkan sebagai output. Karena memiliki sifat LIFO, maka keluaran stack adalah karakter-karakter dengan urutan yang berkebalikan dengan input.

   Buatlah program pembalik kata tersebut dengan membuat 3 class, yaitu:

   a. class "stack". Class ini digunakan untuk menyimpan setiap karakter input pada stack array. Berisi constructor dan method-method operasi stack.
   b. class "pembalik". Class ini digunakan untuk membaca setiap karakter input, menyimpan karakter dengan memanggil method `push()` pada class "stack" (point a), dan membalik input dengan memanggil method `pop()` pada class "stack" (point a). Tiap karakter keluaran dari stack tersebut disimpan/ditambahkan (*append*) pada String output sebagai keluaran yang akan ditampilkan.
   c. class "AppPembalik". Class ini berisi method main. Digunakan untuk deklarasi dan inisialisasi input, memanggil class "pembalik" untuk membalik input dan mendapatkan output kata yang telah dibalik, serta menampilkan output pada console.

   Untuk membaca tiap karakter String, anda dapat mengunakan method **`charAt()`** yang terdapat pada class String, misal `StringInput.charAt(index)`.

   Contoh program pembalik kata ditunjukkan pada Gambar 3.1. Anda dapat mengerjakan sebagaimana Gambar 3.1 (a) dan point tambahan akan diberikan jika Anda mengerjakan sebagaimana Gambar 3.1 (b).

```
run:
>> katanya...
        kasur
>> dibalik jadi...
        rusak
BUILD SUCCESSFUL (total time: 2 seconds)
```

(a)

```
run:
Masukkan kata: bakso
kebalikan: oskab
Masukkan kata: soto
kebalikan: otos
Masukkan kata:
```

(b)

Gambar 3.1 Contoh output program pembalik kata. (a) String input diinisialisasi secara langsung pada listing program (*hardcode*). (b) String input didapat dari masukan dengan keyboard, program dapat membaca dan membalikkan input secara berulang-ulang

2. Implementasi Queue

   Implementasi queue banyak didunia nyata, sebagaimana antrian. Buatlah program simulasi antrian dengan mengimplementasikan Queue. Simulasi antrian menunjukkan (lihat Gambar 3.2):

   - penambahan objek pada daftar antrian. Lakukan beberapa kali hingga antrian penuh. Ketika objek ditambahkan pada antrian yang penuh maka program akan menampilkan keterangan antrian penuh.
   - Menampilkan isi antrian
   - Satu persatu objek keluar antrian hingga antrian kosong

```
run:
>> beberapa mulai mengantri
Andi masuk antrian
Ahmad masuk antrian
Satrio masuk antrian
Afrizal masuk antrian
Maaf sukran, antrian masih penuh
Maaf Mahmud, antrian masih penuh

>> isi antrian
Andi,Ahmad,Satrio,Afrizal,

>> satu persatu keluar antrian
Andi Keluar antrian
Ahmad,Satrio,Afrizal,Kosong,

Ahmad Keluar antrian
Satrio,Afrizal,Kosong,Kosong,

Satrio Keluar antrian
Afrizal,Kosong,Kosong,Kosong,

Afrizal Keluar antrian
Kosong,Kosong,Kosong,Kosong,

antrian kosong
0 Person
Kosong,Kosong,Kosong,Kosong,
BUILD SUCCESSFUL (total time: 2 seconds)
```

Gambar 3.2 Output program simulasi antrian

C. KESIMPULAN

Kesimpulan yang diperoleh dari pembahasan praktikum kali ini adalah:

1. Tentang konsep dan implementasi Stack

2. Tentang konsep dan implementasi Queue

PRAKTIKUM 4 - STRUKTUR DATA

LINKED LIST

Learning outcomes:

1. Mampu menjelaskan empat macam linked list yaitu *linier singly-linked list, linier doubly-linked list, circular singly-linked list,* dan *circular doubly-linked list*.
2. Mampu mengimplementasikan singly-linked list
3. Mampu mengimplementasikan doubly-linked list
4. Mampu mengimplementasikan Stacks pada struktur data linked list
5. Mampu mengimplementasikan Queues pada struktur data linked list

Linked List adalah salah satu bentuk struktur data, berisi kumpulan data (*node*) yang tersusun secara sekuensial, saling sambung-menyambung, dinamis dan tidak terbatas. Pada Linked List, setiap item berada di dalam sebuah `node` atau object `link`. Setiap objek `Link` memuat sebuah referensi (biasa disebut `next`) menuju link setelahnya pada list. Pada bahasa pemrograman C, referensi disini biasa disebut dengan pointer.

A. PENDAHULUAN

1. Dalam pengaplikasiannya, secara struktur link, linked list dibagi menjadi 2 macam, yaitu :
   a. Singly-Linked List, merupakan list dengan penghubung 1 referensi antar data yang menunjuk data selanjutnya (`next`).
   b. Doubly-Linked List, merupakan list dengan penghubung 2 referensi antar data yang menunjuk data sebelum (`prev`) dan setelahnya (`next`).

   Dan dari 2 jenis di atas, secara bentuk list yang terbentuk, masing-masing dapat diaplikasikan dengan 2 bentuk, yaitu :
   a. Linear: List yang berbentuk linear dengan *head/first* sebagai penunjuk awal list dan akan diteruskan dengan obyek-obyek yang saling berhubungan, diakhiri dengan *tail/ last* sebagai penunjuk akhir list, yang mana data terakhir ini akan memiliki referensi null sebagai indikator akhir list.
   b. Circular: List yang berbentuk circular dengan *head/first* sebagai penunjuk awal list dan akan diteruskan dengan obyek-obyek yang saling berhubungan, diakhiri dengan *tail/ last* sebagai penunjuk akhir list. Bedanya akhir list akan memiliki referensi yang akan menunjuk kembali ke awal list (*head/first*) dan atau *head/first* pun akan memiliki referensi yang menunjuk bahwa data sebelumnya adalah *tail/last* list.

   Sehingga, dapat dikatakan terdapat 4 macam struktur linked list, yaitu linier singly-linked list, linier doubly-linked list, circular singly-linked list, dan circular-doubly linked list. Dari ke-empat macam struktur linked list tersebut, gambarkan ilustrasi masing-masing struktur linked list tersebut sehingga tampak isi objek `link` dan hubungan tiap `link`!

jawaban

2. Berikut ini listing program yang menunjukkan implementasi linier singly-linked list.

```
class Link {

    public int Data;
    public Link next;

    public Link(int Data) {
        this.Data = Data;
    }

    public void displayLink() {
        System.out.print(Data +" ");
    }
}

class LinkedList {

    private Link first;
    public LinkedList() {
        first = null;
    }

    public boolean isEmpty() {
        return (first == null);
    }

    public void insertFirst(int Data) {
        Link newLink = new Link(Data);
        newLink.next = first;
        first = newLink;
    }
```

```
public Link deleteFirst() {
    Link temp = first;
    first = first.next;
    return temp;
}


public Link find(int key) {
    Link current = first;
    while (current.Data != key) {
        if (current.next == null) {
            return null;
        } else {
            current = current.next;
        }
    }
    return current;
}


public Link delete(int key) {
    Link current = first;
    Link previous = first;
    while (current.Data != key) {
        if (current.next == null) {
            return null;
        } else {
            previous = current;
            current = current.next;
        }
  }
    if (current == first) {
        first = first.next;
    } else {
```

```
            previous.next = current.next;
        }
        return current;
    }


    public void displayList() {
        System.out.println("List(first
                           -->last):");
        Link current = first;
        while (current != null) {
            current.displayLink();
            current = current.next;
        }
        System.out.println("");
    }
}


public class LinkedListApp {
    public static void main(String[] args) {
        LinkedList theList = new LinkedList();


        theList.insertFirst(22);
        theList.insertFirst(44);
        theList.insertFirst(66);
        theList.insertFirst(88);
        theList.displayList();


        while (!theList.isEmpty()) {
            Link aLink =
                   theList.deleteFirst();

            System.out.print("Deleted ");
            aLink.displayLink();
```

```
            System.out.println("");
        }
        theList.displayList();


        theList.insertFirst(33);
        theList.insertFirst(55);
        theList.insertFirst(77);
        theList.insertFirst(88);
        theList.displayList();

        Link f = theList.find(77);
        if (f != null) {
            System.out.println("ketemu..."
                               + f.Data);
        } else {
            System.out.println("link tidak
                                ditemukan");
        }

        Link d = theList.delete(88);
        if (d != null) {
            System.out.println("hapus link
                    dengan key " + d.Data);
        } else {
            System.out.println("link tidak
                                ditemukan");
        }
        theList.displayList();
    }
}
```

Tuliskan output dari listing program tersebut!

jawaban

3. Pada class `Link` (listing nomor 2) terdapat deklarasi variable "`next`" yang bertipe objek `Link` (sama dengan nama class tersebut). Variable tersebut tidak di-inisialisasi dengan value tertentu. Jelaskan maksud dan fungsi dari variable `next` pada baris `public Link next;`

jawaban

4. Jelaskan fungsi dan logika tiap method `insertFirst()`, `deleteFirst()`, `find()`, dan `delete()` yang terdapat pada listing nomer 2!

jawaban

5. Jelaskan langkah/logika untuk menambahkan operasi “insert Last” yang berfungsi untuk menambahkan data pada akhir list. Tuliskan listing program yang perlu ditambahkan pada program nomor 2!

jawaban

6. Jelaskan langkah/logika untuk menambahkan operasi "delete Last" yang berfungsi untuk menghapus data pada akhir list. Tuliskan listing program yang perlu ditambahkan pada program nomor 2!

jawaban

7. Berikut ini listing program yang menunjukkan implementasi doubly-linked list.

```
class Link {

    public int Data;

    public Link next;

    public Link previous;


    public Link(int Data) {

        this.Data = Data;

    }

    public void displayLink() {

        System.out.print(Data + " ");

    }

}
```

```
class DoublyLinkedList {
    private Link first;
    private Link last;


    public DoublyLinkedList() {
        first = null;
        last = null;
    }
```

```
    public boolean isEmpty() {
        return first == null;
    }
```

```
    public void insertFirst(int Data) {
        Link newLink = new Link(Data);
        if (isEmpty()) {
            last = newLink;
        } else {
            first.previous = newLink;
        }
        newLink.next = first;
        first = newLink;
    }
```

```
    public void insertLast(int Data) {
        Link newLink = new Link(Data);
        if (isEmpty()) {
            first = newLink;
        } else {
            last.next = newLink;
            newLink.previous = last;
        }
        last = newLink;
    }
```

```
    public Link deleteFirst() {
        Link temp = first;
        if (first.next == null) {
            last = null;
        } else {
            first.next.previous = null;
        }
        first = first.next;
        return temp;
    }
```

```
    public Link deleteLast() {
        Link temp = last;
        if (first.next == null) {
            first = null;
        } else {
            last.previous.next = null;
        }
        last = last.previous;
        return temp;
    }
```

```
   public boolean insertAfter(int key,
                              int Data) {
        Link current = first;
        while (current.Data != key) {
            current = current.next;
            if (current == null) {
                return false;
            }
        }
        Link newLink = new Link(Data);
        if (current == last) {
```

```
            newLink.next = null;
            last = newLink;
        } else {
            newLink.next = current.next;
            current.next.previous = newLink;
        }
        newLink.previous = current;
        current.next = newLink;
        return true;
    }
```

```
    public Link deleteKey(int key) {
        Link current = first;
        while (current.Data != key) {
            current = current.next;
            if (current == null) {
                return null;
            }
        }
        if (current == first) {
            first = current.next;
        } else {
            current.previous.next =
                                current.next;
        }
        if (current == last) {
            last = current.previous;
        } else {
            current.next.previous =
                            current.previous;
        }
        return current;
    }
```

```
    public void displayForward() {
        System.out.print("List ”
```

```
                        + "(first-->last): ");
        Link current = first;
        while (current != null) {
            current.displayLink();
            current = current.next;
        }
        System.out.println("");
    }
```

```
    public void displayBackward() {
        System.out.print("List "
                        + "(last-->first): ");
        Link current = last;
        while (current != null) {
            current.displayLink();
            current = current.previous;
        }
        System.out.println("");
    }
} // akhir class
```

```
public class DoublyLinkedListApp {
    public static void main(String[] args) {
        DoublyLinkedList theList =
                    new DoublyLinkedList();
        theList.insertFirst(22);
        theList.insertFirst(44);
        theList.insertFirst(66);
        theList.displayForward();
        theList.insertLast(11);
        theList.insertLast(33);
        theList.insertLast(55);
        theList.displayForward();
        theList.displayBackward();
        theList.deleteFirst();
```

Tuliskan output program ini:

```
        theList.displayForward();

        theList.deleteLast();

        theList.displayForward();

        theList.deleteKey(11);

        theList.displayForward();

        theList.insertAfter(22, 77);

        theList.insertAfter(33, 88);

        theList.displayForward();

    }

}
```

8. Dari penjelasan tiap bagian yang Anda tuliskan pada nomor 7, tuliskan kesimpulan logika yang digunakan pada tiap method: `insertFirst()`, `insertLast()`, `insertAfter()`, `deleteFirst()`, `deleteLast()`, `deleteKey()`!

jawaban

B. PRAKTIKUM

1. Implementasi Stack pada Linked List
   Pada modul 3, Stacks telah dibahas beserta implementasinya pada struktur data Array. Pada praktikum kali ini, buatlah program yang mengimplementasikan Stack menggunakan struktur data Linked list dengan ketentuan:
   - Data yang disimpan adalah record barang elektronik yang memiliki field id bertipe integer dan nama barang bertipe String.
   - Lakukan push, pop, dan display stack pada program

```
run:
Stack(top-- > bottom):
 {2, TV}
 {1, VCD}

Stack(top-- > bottom):
 {6, dispenser}
 {5, rice cooker}
 {4, PC}
 {3, kulkas}
 {2, TV}
 {1, VCD}

Stack(top-- > bottom):
 {4, PC}
 {3, kulkas}
 {2, TV}
 {1, VCD}

BUILD SUCCESSFUL (total time: 1 second)
```

Gambar 4.1 Contoh output program Stack pada Linked list

2. Implementasi Queue pada Linked List
   Sebagaimana soal praktikum nomer 1, buatlah program yang mengimplementasikan Queue menggunakan struktur data Linked list dengan ketentuan:
   - data yang disimpan adalah data mahasiswa (nim dan nama) yang melakukan antrian
   - Lakukan insert, remove, dan display queue pada program!

```
run:
Queue (front-->rear):
        1665100 Dee
        1665200 Deaja

Queue (front-->rear):
        1665100 Dee
        1665200 Deaja
        1665300 Ami
        1665400 Haya
        1665500 Yati

Queue (front-->rear):
        1665300 Ami
        1665400 Haya
        1665500 Yati

BUILD SUCCESSFUL (total time: 2 seconds)
```

Gambar 4.2 Contoh output program Queue pada Linked list

C. KESIMPULAN

Kesimpulan yang diperoleh dari pembahasan praktikum kali ini adalah:

1. Tentang perbandingan 4 macam struktur linked list, yaitu linier singly-linked list, linier doubly-linked list, circular singly-linked list, dan circular-doubly linked list.

2. Tentang implementasi Stack pada Linked list

3. Tentang implementasi Queue pada Linked list

PRAKTIKUM 5 - STRUKTUR DATA

RECURSION

Learning outcomes:

1. Mampu menjelaskan konsep rekursi serta menyebutkan contoh program yang dapat diimplementasikan menggunakan teknik rekursi.
2. Mampu mengimplementasikan teknik rekursi pada perhitungan *triangular number*, *factorial*, dan *raising a number to a power*.
3. Mampu menyelesaikan puzzle Menara Hanoi dan mengimplementasikannya dalam program.
4. Mampu mengimplementasikan algoritma merge sort pada array.

Rekursi adalah teknik dalam pemrograman dimana suatu method/fungsi memanggil dirinya sendiri. Fungsi ini akan terus berjalan sampai kondisi berhenti terpenuhi, oleh karena itu dalam sebuah fungsi rekursi perlu terdapat 2 blok penting, yaitu blok yang menjadi titik berhenti dari sebuah proses rekursi dan blok yang memanggil dirinya sendiri.

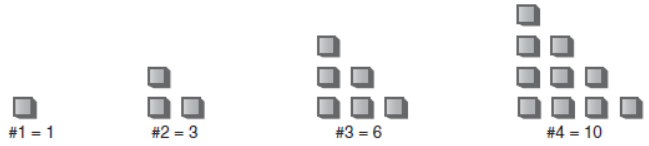
Konsep ini dapat digunakan untuk merumuskan solusi sederhana dalam sebuah permasalahan yang sulit untuk diselesaikan secara iteratif dengan menggunakan loop for, while, do-while. Pada saat tertentu konsep ini dapat digunakan untuk mendefinisikan permasalahan dengan konsisten dan sederhana.

A. PENDAHULUAN

1. Rekursi merupakan teknik dimana method/fungsi memanggil dirinya sendiri. Sebutkan minimal tiga contoh program yang dapat diimplementasikan menggunakan teknik rekursi! Untuk masing-masing contoh tersebut, jelaskan bagaimana penggunaan rekursi dalam menyelesaikan masalah pada contoh yang Anda sebutkan!

jawaban

2. Menghitung bilangan segitiga (*triangular number*)



Gambar 5.1 Ilustrasi bilangan segitiga

Rekursi dapat digunakan untuk menghitung bilangan segitiga tanpa menggunakan iterasi (for, while, atau do-while). Berikut ini listing program method perhitungan bilangan segitiga.

Iterasi	Rekursi

`public int triangleIter(int n) {`
`  int result = 0;`
`  for (int i = n; i > 0; i--) {`
`  result += i;`
`  }`
`  return result;`
`}`
	`public int triangleRecur (int n)`
	`{`
	`  if (n == 1) {`
	`  return 1;`
	`  } else {`
	`  return n + triangleRecur`
	`  (n - 1);`
	`  }`
	`}`

Lengkapi listing program tersebut dalam sebuah class dan tambahkan method main untuk memanggil masing-masing method tersebut. Jalankan program untuk menghitung bilangan segitiga dari 5. Tuliskan output program yang telah Anda lengkapi!

jawaban

3. Pahami dan bandingkan langkah perhitungan bilangan segitiga antara menggunakan teknik perulangan dan dengan teknik rekursi. Tuliskan langkah perhitungan sesuai program nomor 3 yang telah Anda jalankan! Jelaskan!

jawaban

Berdasarkan tahapan perhitungan tersebut, maka dapat dikatakan bahwa teknik rekursi menyelesaikan problem mulai dari yang **lebih kecil / lebih besar** (*coret salah satu*)

4. Pada method rekursi nomor 3 (`triangleRecur`) terdapat baris kode (`if(n==1)`, `return 1;`, dan `else`) yang menunjukkan *base case* untuk masalah perhitungan bilangan segitiga. Pada kondisi tersebut, tidak dilakukan pemanggilan terhadap method itu sendiri.

Lakukan percobaan dengan menghilangkan baris kode tersebut. Jalankan program dan lihat apa yang terjadi! Jelaskan!

jawaban

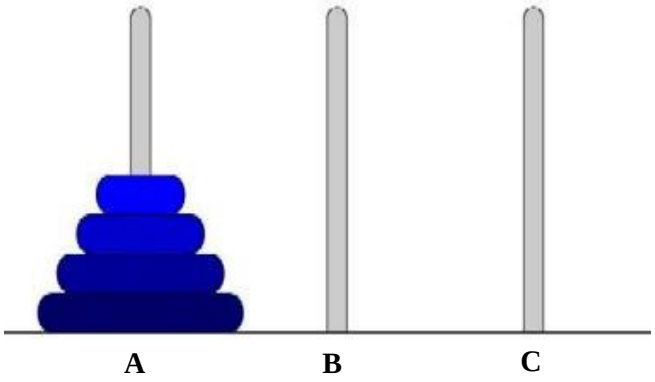
5. Dengan algoritma yang semisal dengan perhitungan bilangan segitiga, tuliskan listing program untuk menghitung nilai faktorial, baik menggunakan iterasi maupun menggunakan teknik rekursi! Jelaskan!

jawaban

6. Menara Hanoi

   Menara Hanoi ini adalah sebuah permainan matematis atau teka-teki. Permainan ini terdiri dari tiga tiang dan sejumlah cakram dengan ukuran berbeda-beda yang bisa dimasukkan ke tiang mana saja. Permainan dimulai dengan cakram-cakram yang tertumpuk rapi berurutan berdasarkan ukurannya dalam salah satu tiang, cakram terkecil diletakkan teratas, sehingga membentuk kerucut.

   Tujuan dari teka-teki ini adalah untuk memindahkan seluruh tumpukan ke tiang yang lain, mengikuti aturan berikut:
   - Hanya satu cakram yang boleh dipindahkan dalam satu waktu.
   - Setiap perpindahan berupa pengambilan cakram teratas dari satu tiang dan memasukkannya ke tiang lain, di atas cakram lain yang mungkin sudah ada di tiang tersebut.
   - Tidak boleh meletakkan cakram di atas cakram lain yang lebih kecil.

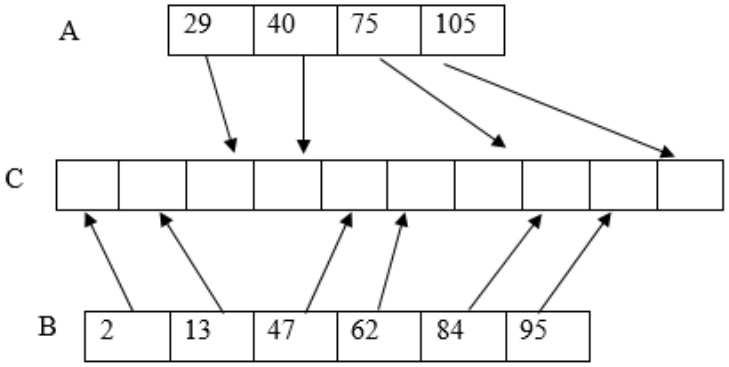


Gambar 5.2 Menara Hanoi

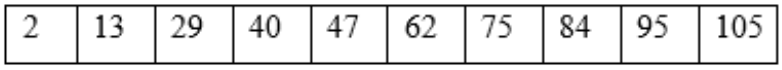
Hitung jumlah perpindahan dan tuliskan langkah untuk memindahkan 3, 4 dan 5 buah cakram ke tiang C (tujuan)!

jawaban

7. Menggabungkan dua *ordered array*
   Penggabungan (Merge) adalah teknik yang menjadi konsep algoritma pengurutan "Merge Sort". Berikut ini merupakan ilustrasi merge (penggabungan) dua array, yaitu array A dan array B yang masing-masing sudah terurut. Array A dan B digabungkan dan disimpan dalam Array C.



(a) Array sebelum digabungkan



(b) Array setelah digabungkan

Gambar 5.3 Penggabungan dua *ordered array*

Tuliskan langkah-langkah penggabungan dua Array sesuai dengan ilustrasi diatas!

jawaban

8. Berikut ini listing program untuk sorting menggunakan algoritma MergeSort

```
class Arrays {

    private int arr[];
    private int nElemen;

    public Arrays(int size) {
        arr = new int[size];
        nElemen = 0;
    }

    public void insert(int value) {
        arr[nElemen] = value;
        nElemen++;
    }
```

```
    public void display() {
        for (int i = 0; i < nElemen; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println("");
    }

    public void mergeSort() {
        int[] workSpace = new int[nElemen];
        recMergeSort(workSpace, 0, nElemen - 1);
    }
```

```
public void recMergeSort(int[] workSpace,
        int lowerBound, int upperBound) {



  if (lowerBound == upperBound) {
      return;
  } else {
      int mid = (lowerBound + upperBound)
                / 2;
      recMergeSort(workSpace, lowerBound,
                   mid);
      recMergeSort(workSpace, mid + 1,
                   upperBound);
      merge(workSpace, lowerBound,
            mid + 1, upperBound);
  }
}
```

```
public void merge(int[] workSpace,
        int lowIndex, int highIndex,
        int upperBound) {



  int j = 0;
  int lowerBound = lowIndex;
  int mid = highIndex - 1;
  int nItem = upperBound - lowerBound + 1;



  while (lowIndex <= mid
         && highIndex <= upperBound) {
```

```
    if (arr[lowIndex] < arr[highIndex]) {
      workSpace[j++] = arr[lowIndex++];
    } else {
      workSpace[j++] = arr[highIndex++];
    }
  }

  while (lowIndex <= mid) {
    workSpace[j++] = arr[lowIndex++];
  }

  while (highIndex <= upperBound) {
    workSpace[j++] = arr[highIndex++];
  }
  for (j = 0; j < nItem; j++) {
    arr[lowerBound + j] = workSpace[j];
  }
}
}//akhir class Arrays
```

Lengkapi listing program diatas dengan sebuah class yang berisi method main. lakukan penambahan 10 item yang tidak berurutan. Tampilkan data sebelum diurutkan kemudian lakukan pengurutan menggunakan mergeSort dan tampilkan data setelah diurutkan!
Jalankan program yang telah Anda lengkapi dan tulis outputnya!

jawaban

9. Berdasarkan data yang Anda gunakan pada soal nomor 8, tuliskan tiap tahap pengurutan menggunakan mergeSort! Jelaskan!

jawaban

B. PRAKTIKUM

1. Buatlah sebuah program untuk menghitung nilai pangkat dari suatu bilangan yang mengimplementasikan method rekursi dengan dua parameter (*base* dan *exponent*). Anda dapat menggunakan aturan eksponen berikut ini!
   - $base^0 = 1$
   - $base^{exp} = base * base^{exp-1}$
   - $base^{2*exp} = (base^2)^{exp}$

```
run:
2^10 = 1024
3^ 5 = 243
3^16 = 43046721
BUILD SUCCESSFUL (total time: 2 seconds)
```

2. Tuliskan listing program untuk menyelesaikan permainan Menara Hanoi menggunakan teknik rekursi! Terdapat method main() yang berisi pemanggilan fungsi rekursif dari method doMenara(). Method ini memanggil dirinya sendiri sampai puzzle menara Hanoi terselesaikan.

C. KESIMPULAN

Kesimpulan yang diperoleh dari pembahasan praktikum kali ini adalah:

1. Tentang konsep rekursi

2. Tentang implementasi rekursi untuk meyelesaikan teka-teki Menara Hanoi

3. Tentang Merge sort

PRAKTIKUM 6 - STRUKTUR DATA

ADVANCED SORT

Learning outcomes:

1. Mampu menjelaskan langkah-langkah *sorting* algoritma *Shell sort.*
2. Mampu menjelaskan langkah-langkah *partitioning* yang menjadi konsep algoritma *Quick sort*.
3. Mampu menjelaskan langkah-langkah sorting algoritma *Quick sort*.
4. Mampu mengimplementasikan *sorting* algoritma *Shell sort* dan *Quick sort* pada program.
5. Mampu mengimplementasikan *sorting* algoritma *Merge sort*, *Shell sort*, dan *Quick sort* untuk *sorting object*.

Pada modul 2, *simple sorting* telah dibahas, yaitu meliputi *bubble sort*, *selection sort*, dan *insertion sort*. Ketiga algoritma tersebut adalah algoritma *sorting* sederhana yang mudah diimplementasikan namun tergolong lambat. Algoritma *sorting* yang lain yaitu *merge sort*, lebih cepat daripada *simple sorting* namun membutuhkan ruang dua kali lipat dari ukuran array yang diurutkan. Kali ini akan dibahas dua algoritma *advanced sorting*, yaitu *shell sort* dan *quick sort*. Keduanya lebih cepat daripada algoritma *simple sorting* dan tidak membutuhkan tambahan ruang sebagaimana *merge sort*.

A. PENDAHULUAN

1. *Shell sort*

Shell sort menggunakan konsep yang berdasarkan pada *insertion sort*. Pada *insertion sort*, setiap item ditandai dengan menyimpannya pada variable temporary kemudian item tersebut dibandingkan dan disisipkan pada *sorted group* sesuai urutannya. Penyisipan dilakukan dengan menggeser item yang memiliki nilai lebih besar dari *marked item* sehingga terdapat ruang untuk ditempati oleh *marked item*.

Masalah pada *insertion sort* tampak ketika nilai yang paling becil berada pada posisi paling kanan dan harus ditempatkan pada posisi paling kiri di *sorted group*. Pada kondisi tesebut penyisipan tetap dilakukan dengan menggeser sejumlah item satu ruang kekanan.

Performa pada *insertion sort* tesebut dapat ditingkatkan dengan cara melakukan pergeseran pada jarak tertentu. Hal inilah yang dilakukan pada *shell sort*. Jarak antar elemen disebut dengan *increment value*. Misal, pada proses pengurutan 10 elemen array dengan nilai *increment* 4 (disebut *4-sort*), maka tiap subarray yang dibandingkan adalah pada index (0, 4, 8), (1, 5, 9), (2, 6), dan (3, 7).

Proses 4-sort menghasilkan array yang hampir terurut. Untuk array yang lebih panjang, dibutuhkan jarak yang lebih besar kemudian secara berulang berkurang hingga bernilai 1 sehingga elemen dapat terurut secara sempurna. Misalkan array dengan 1000 item dapat diurutkan secara bertahap mulai 364-sorted, kemudian 121-sorted, 40-sorted, 13-sorted, 4-sorted, hingga 1-sorted. Deret bilangan yang digunakan untuk membentuk interval ini (pada contoh ini: 364, 121, 40, 13, 4, 1) disebut dengan *interval sequence* atau *gab sequence*.

Original *shell sort* menggunakan N/2 untuk menentukan *interval sequence*, dimana N dimulai dari jumlah elemen dan secara berulang dibagi dengan 2 hingga bernilai 1.

Berikut ini listing program untuk *shell sort* dengan N/2 sequence.

```
public void ShellSort () {
  int in, out;
  int temp;
  int h = nElemen / 2;

  while (h > 0) {
    for (out = h; out < nElemen; out++) {
       temp = arr[out];
       in = out;
```

```
        while (in > h - 1 &&
                arr[in - h] >= temp) {
            arr[in] = arr[in - h];
            in -= h;
        }
        arr[in] = temp;
    }
    h /= 2;
 }
}
```

Lengkapi listing program tersebut sebagaimana yang pernah Anda kerjakan pada modul 2. Lakukan pengurutan terhadap 8 elemen array. Tambahkan code untuk menampilkan isi array (`display();`) setelah baris code `arr[in] = temp`;.
Jalankan dan tulis output program tersebut! Jelaskan!

jawaban

2. N/2 *interval sequence* dianggap tidak efektif. Knuth telah mengenalkan salah satu bentuk *interval sequence* yang kini umum digunakan. Knuth's interval sequence menggunakan bentuk kebalikan yang dimulai dari 1 menggunakan persamaan:

$$h = 3 * h + 1$$

Tabel 6.1 Knuth's interval sequence

H	**3 * h + 1**	**(h - 1) / 3**
1	4	
4	13	1
13	40	4
40	121	13
121	364	40
364	1093	121
1093	3280	364

Untuk mengimplementasikan Knuth's interval sequence pada shell sort (listing nomor 1), tunjukkan baris code yang harus dirubah dan tuliskan code-nya! Jelaskan!

jawaban

3. Lakukan pengurutan terhadap 8 elemen array sesuai data yang anda gunakan pada soal nomor 1 menggunakan algoritma shell sort dengan interval/ jarak yang dirumuskan oleh Knuth. Jalankan program, bagaimana proses pengurutannya? Bandingkan dengan output program nomor 1. Jelaskan!

jawaban

4. Partisi

   Partisi adalah mekanisme yang digunakan pada quick sort. Melakukan partisi data berarti membagi data tersebut menjadi dua bagian berdasarkan nilai batas tertentu. Item yang memiliki nilai lebih kecil dari nilai batas menjadi satu kelompok sedangkan item yang memiliki nilai lebih besar dari nilai batas menjadi satu kelompok yang lain. Batas yang menjadi penentu kelompok data tersebut dikenal sebagai *pivot value*.

   Jika terdapat 10 elemen array berikut ini:

60	5	15	45	35	20	25	10	50	30
0	1	2	3	4	5	6	7	8	9

   dan diketahui nilai pivot adalah 30.

   Tuliskan langkah-langkah partisi array tersebut hingga diperoleh array hasil partisi sebagai berikut:

30	5	15	10	25	20	35	45	50	60
0	1	2	3	4	5	6	7	8	9

   Jelaskan!

jawaban

Berapa kali pertukaran item pada proses partisi array tersebut?

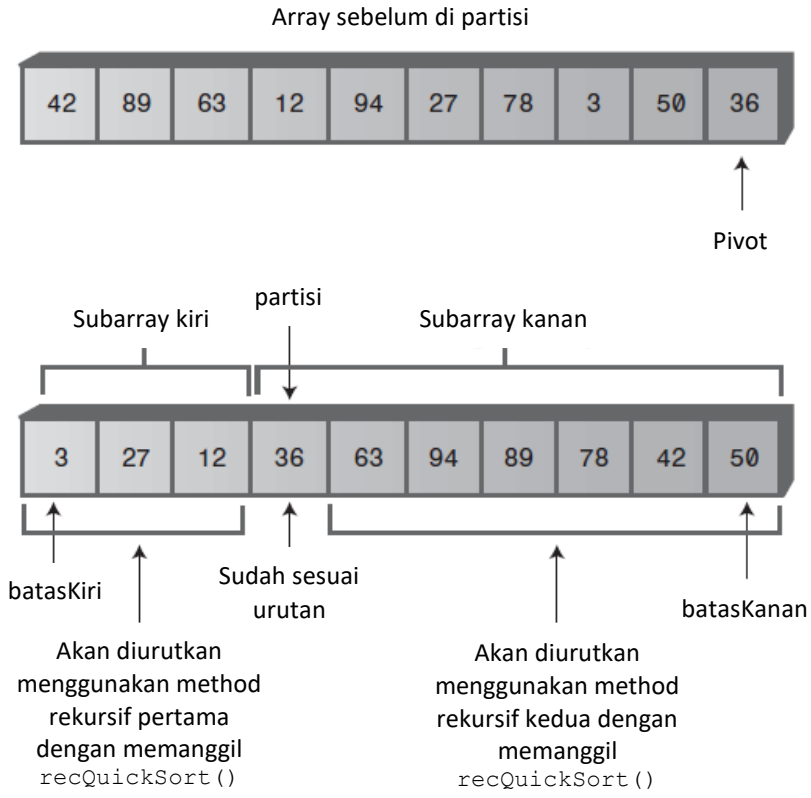
Dimana posisi index partisi yang menjadi batas bagian kiri dan kanan?

5. ***Quick Sort***

Quick sort berkerja dengan melakukan patisi sebuah array menjadi dua subarray kemudian memanggil fungsi itu sendiri secara rekursif untuk melakukan *quick-sort* terhadap tiap subarray tersebut.

Berikut ini tiga tahapan dasar dari quick sort:

a. Partisi array atau subarray menjadi bagian kiri (kumpulan item yang memiliki nilai lebih kecil) dan bagian kanan (kumpulan item yang memiliki nilai lebih besar).
b. Panggil method partisi itu sendiri untuk mengurutkan bagian kiri.
c. Panggil method partisi itu sendiri untuk mengurutkan bagian kanan.



Gambar 6.1 Pengurutan Subarray secara rekursif

Untuk melakukan partisi maka perlu menentukan nilai pivot. Berikut ini beberapa catatan tentang pivot pada *quick sort*:

- Pivot dipilih dari nilai item yang eksis berada pada array.
- Nilai item sebagai pivot dapat dipilih secara acak. Untuk memudahkan, kita dapat memilih item di batas kanan dari subarray yang akan di partisi
- Setelah melakukan partisi, jika pivot berada diantara batas kiri dan batas kanan dari subarray, maka kondisi tersebut menunjukkan array telah diurutkan.

Berikut ini listing program untuk *Quick Sort*:

```
public void QuickSort() {
   recQuickSort(0, nElemen - 1);
}
```

```
public void recQuickSort(int batasKiri,
                         int batasKanan) {
   if (batasKanan - batasKiri <= 0) {
       return ;
   } else {
       int pivot = arr[batasKanan];
       int partisi = partitionIt(batasKiri,
                        batasKanan, pivot);
       recQuickSort(batasKiri, partisi - 1);
       recQuickSort(partisi + 1, batasKanan);
   }
}
```

```
public int partitionIt(int batasKiri,
                int batasKanan, int pivot) {
   int indexKiri = batasKiri - 1;
   int indexKanan = batasKanan + 1;
   while (true) {
       while (indexKiri < batasKanan &&
              arr[++indexKiri] < pivot) ;
       while (indexKanan > batasKiri &&
              arr[--indexKanan] > pivot) ;
       if (indexKiri >= indexKanan) {
            break;
       } else {
           swap(indexKiri, indexKanan);
       }
   }
   return indexKiri;
}
```

Lengkapi listing program tersebut. Lakukan pengurutan terhadap 8 elemen array yang sama dengan data yang Anda gunakan pada soal nomor 1 modul ini. Jalankan program yang telah Anda lengkapi. Tulis dan jelaskan langkah-langkah pengurutan menggunakan *quick sort* terhadap data array Anda!

jawaban

B. PRAKTIKUM

Pada modul 2 (*simple sorting*) Anda telah melakukan *sorting object* "Mahasiswa" menggunakan algoritma *sorting* sederhana. Pada praktikum kali ini, lakukan *sorting object* "Mahasiswa" menggunakan algoritma *merge sort*, *shell sort*, dan *quick sort* dengan ketentuan sebagai berikut:

a. NIM ganjil mengimplementasikan algoritma:
   - *Merge Sort* untuk pengurutan berdasarkan NIM
   - *Shell Sort* untuk pengurutan berdasarkan Nama
   - *Quick Sort* untuk pengurutan berdasarkan NIM

b. NIM genap mengimplementasikan algoritma:
   - *Merge Sort* untuk pengurutan berdasarkan Nama
   - *Shell Sort* untuk pengurutan berdasarkan NIM
   - *Quick Sort* untuk pengurutan berdasarkan Nama

C. KESIMPULAN

Kesimpulan yang diperoleh dari pembahasan praktikum kali ini adalah:

1. Tentang algoritma *shell sort*

2. Tentang konsep *partitioning*

3. Tentang algoritma *quick sort*

PRAKTIKUM 7 - STRUKTUR DATA

BINARY TREES

Learning outcomes:

1. Mahasiswa mampu menjelaskan terminology tree.
2. Mahasiswa mampu menjelaskan dan mengimpementasikan operasi *find*, *insert*, dan *delete* node pada program binary tree.
3. Mahasiswa mampu menjelaskan tiga jenis operasi *traverse* (preorder, inorder, postorder) dan mengimpementasikan operasi tersebut pada program binary tree.
4. Mahasiswa mampu menjelaskan dan mengimpementasikan operasi *find* value maksimum dan minimum pada program binary tree.
5. Mahasiswa mampu memodifikasi method insert pada program binary tree untuk masalah *duplicate key*.

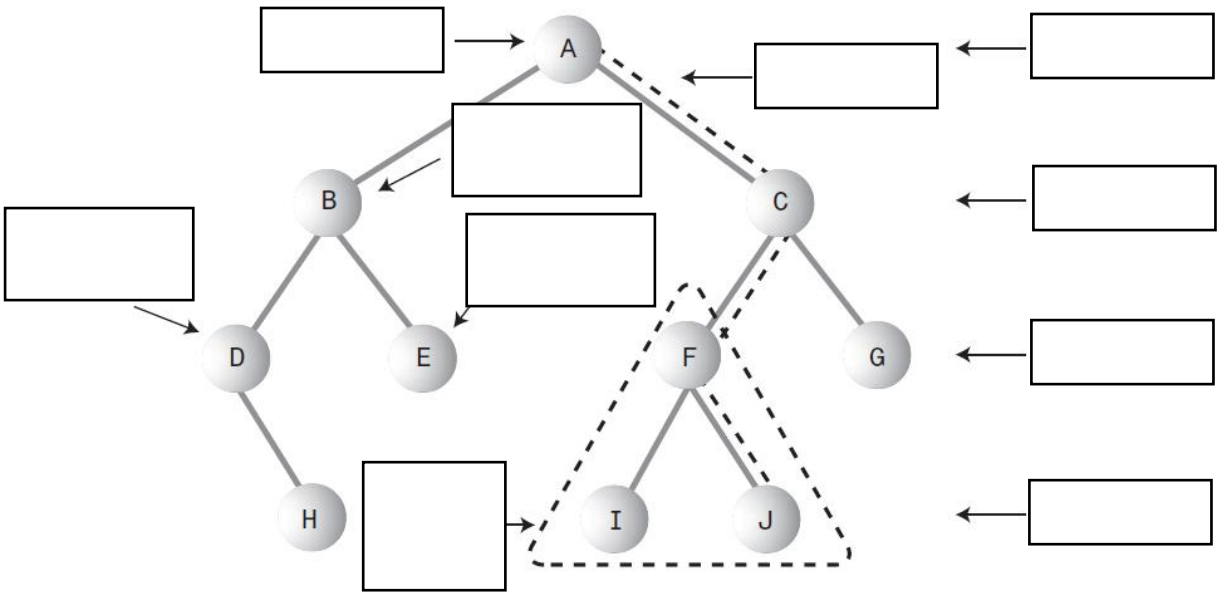
Struktur data yang dibahas pada modul ini adalah binary trees. Binary tree mengkombinasikan keuntungan pada dua struktur data lain, yaitu ordered array dan linked list. Search pada binary tree, cepat sebagaimana pada ordered array. Sedangkan insert dan delete pada binary tree juga cepat sebagaimana pada linked list.

Tree terdiri dari beberapa node yang terhubung dengan edge. Pada program, node sering kali merepresentasikan suatu entiti seperti orang, barang, dan lain sebagainya. Sedangkan edge antar node merepresentasikan hubungan antar node tersebut.

Ada beberapa jenis tree, salah satunya adalah binary tree. Tiap node pada binary tree memiliki maksimal 2 children.

A. PENDAHULUAN

1. Untuk mengenali istilah-istilah yang ada pada tree, isilah tiap kotak pada gambar berikut dengan istilah yang sesuai.



H, E, I, J, dan G adalah

2. Jelaskan apa yang dimaksud dengan path, root, parent, child, leaf, subtree, visiting, traversing, level, dan key pada tree.

Path :	
Root :	
Parent :	
Child :	

Leaf :	
Subtree :	
Visiting :	
Traversing :	
Level :	
Key :	

3. Implementasi Binary tree pada Java
   Tree dapat diimplementasikan dengan menyimpan node pada memori sebagai sebuah array. Atau, tree juga dapat diimplementasikan dengan menyimpan node pada memori dan menghubungkannya menggunakan references pada tiap node yang menunjuk pada children node tersebut. Berikut ini beberapa class yang perlu implementasikan dalam membuat program binary tree.

   a. Class Node
      Class Node sebagai objek yang merepresentasikan data objek yang akan disimpan dan juga memuat reference ke masing-masing dua children
   b. Class Tree
      Class Tree merepresentasikan objek tree itu sendiri, yaitu sebuah objek yang menangani semua node. Pada class Tree, hanya ada 1 field, yaitu variable node dengan nama root. Node lain diakses melalui root sehingga tidak diperlukan adanya field lain.
      Terdapat beberapa method pada class Tree, yaitu method untuk pencarian (find), menambahkan node (insert), menghapus node (delete), serta method yang menangani berbagai macam travers, dan untuk menampilkan tree (display).
   c. Class TreeApp
      Class TreeApp berisi method main untuk menjalankan program. Pada listing program berikut, method main menyediakan sebuah *primitive user interface* sehingga user dapat menentukan aksi dari input keyboard, apakah ingin insert, find, delete, show, atau travers.

   Tuliskan listing program berikut ini. Lengkapi dengan class Stack yang digunakan pada method display di class Tree.

```
public class Node {

    public int id;
    public String data;
    public Node leftChild;
    public Node rightChild;

    public void displayNode() {
        System.out.print("{ " + id + ", " + data + " } ");
    }
}
```

```
public class Tree {

    private Node root;

    public Tree() {
        root = null;
    }

    public Node find(int key) {
        Node current = root;
        while (current.id != key) {
            if (key < current.id) {
                current = current.leftChild;
            } else {
                current = current.rightChild;
            }
            if (current == null) {
                return null;
            }
        }
        return current;
    }

    public void insert(int id, String data) {
        Node newNode = new Node();
        newNode.id = id;
        newNode.data = data;
        if (root == null) {
            root = newNode;
        } else {
            Node current = root;
            Node parent;
            while (true) {
                parent = current;
                if (id < current.id) {
                    current = current.leftChild;
                    if (current == null) {
                        parent.leftChild = newNode;
                        return;
                    }
                } else {
                    current = current.rightChild;
                    if (current == null) {
                        parent.rightChild = newNode;
                        return;
                    }
                }
            }
        }
    }

    public boolean delete(int key) {
        Node current = root;
        Node parent = root;
        boolean isLeftChild = true;
```

```
            while (current.id != key) {
                parent = current;
                if (key < current.id) {
                    isLeftChild = true;
                    current = current.leftChild;
                } else {
                    isLeftChild = false;
                    current = current.rightChild;
                }
                if (current == null) {
                    return false;
                }
            }
            if (current.leftChild == null
                    && current.rightChild == null) {
                if (current == root) {
                    root = null;
                } else if (isLeftChild) {
                    parent.leftChild = current.leftChild;
                } else {
                    parent.rightChild = current.leftChild;
                }
            } else if (current.rightChild == null) {
                if (current == root) {
                    root = current.leftChild;

                } else if (isLeftChild) {
                    parent.leftChild = current.leftChild;
                } else {
                    parent.rightChild = current.leftChild;
                }
            } else if (current.leftChild == null) {
                if (current == root) {
                    root = current.rightChild;
                } else if (isLeftChild) {
                    parent.leftChild = current.rightChild;
                } else {
                    parent.rightChild = current.rightChild;
                }
            } else {
                Node successor = getSuccessor(current);
                if (current == root) {
                    root = successor;
                } else if (isLeftChild) {
                    parent.leftChild = successor;
                } else {
                    parent.rightChild = successor;
                }
                successor.leftChild = current.leftChild;
            }
            return true;
        }

        private Node getSuccessor(Node delNode) {
            Node successorParent = delNode;
            Node successor = delNode;
            Node current = delNode.rightChild;
            while (current != null) {
                successorParent = successor;
                successor = current;
                current = current.leftChild;
            }
            if (successor != delNode.rightChild) {
                successorParent.leftChild = successor.rightChild;
                successor.rightChild = delNode.rightChild;
            }
            return successor;
        }
```

```
public void traverse(int traverseType) {
    switch (traverseType) {
        case 1:
            System.out.print("Preorder traversal: ");
            preOrder(root);
            break;
        case 2:
            System.out.print("Inorder traversal: ");
            inOrder(root);
            break;
        case 3:
            System.out.print("Postorder traversal: ");
            postOrder(root);
            break;
    }
    System.out.println();
}

private void preOrder(Node localRoot) {
    if (localRoot != null) {
        System.out.print(localRoot.id + " ");
        preOrder(localRoot.leftChild);
        preOrder(localRoot.rightChild);
    }
}

private void inOrder(Node localRoot) {
    if (localRoot != null) {
        inOrder(localRoot.leftChild);
        System.out.print(localRoot.id + " ");
        inOrder(localRoot.rightChild);
    }
}

private void postOrder(Node localRoot) {
    if (localRoot != null) {
        postOrder(localRoot.leftChild);
        postOrder(localRoot.rightChild);
        System.out.print(localRoot.id + " ");
    }
}

public void displayTree() {
    Stack globalStack = new Stack();
    globalStack.push(root);
    int nBlanks = 32;
    boolean isRowEmpty = false;
    System.out.println(
            ".........................................");
    while (isRowEmpty == false) {
        Stack localStack = new Stack();
        isRowEmpty = true;
        for (int j = 0; j < nBlanks; j++) {
            System.out.print(' ');
        }
        while (globalStack.isEmpty() == false) {
            Node temp = (Node) globalStack.pop();
            if (temp != null) {
                System.out.print(temp.id);
                localStack.push(temp.leftChild);
                localStack.push(temp.rightChild);
                if (temp.leftChild != null
                        || temp.rightChild != null) {
                    isRowEmpty = false;
                }
            } else {
```

```
                    System.out.print("--");
                    localStack.push(null);
                    localStack.push(null);
                }
                for (int j = 0; j < nBlanks * 2 - 2; j++) {
                    System.out.print(' ');
                }
            }
            System.out.println();
            nBlanks /= 2;
            while (localStack.isEmpty() == false) {
                globalStack.push(localStack.pop());
            }
        }
        System.out.println(
                "........................................");
    }
}
```

```
public class TreeApp {

    public static void main(String[] args) throws IOException {
        int value;
        String data;
        Tree theTree = new Tree();
        theTree.insert(50, "Ahmad");
        theTree.insert(25, "Rosa");
        theTree.insert(75, "Raisa");
        theTree.insert(12, "Naya");
        theTree.insert(37, "Gagas");
        theTree.insert(43, "Ainun");
        theTree.insert(30, "Beri");
        theTree.insert(33, "Vivid");
        theTree.insert(87, "Orin");
        theTree.insert(93, "Wiwid");
        theTree.insert(97, "Sasa");

        while (true) {
            System.out.print("Enter first letter of show, "
                    + "insert, find, delete, or traverse: ");
            int choice = getChar();
            switch (choice) {
                case 's':
                    theTree.displayTree();
                    break;
                case 'i':
                    System.out.print("Enter value and data to"
                            + " insert: ");
                    value = getInt();
                    data = getString();
                    theTree.insert(value, data);
                    break;
                case 'f':
                    System.out.print("Enter value to find: ");
                    value = getInt();
                    Node found = theTree.find(value);
                    if (found != null) {
                        System.out.print("Found: ");
                        found.displayNode();
                        System.out.print("\n");
                    } else {
                        System.out.println("Could not find "
                                + value);
                    }
                    break;
```

```
                    case 'd':
                        System.out.print("Enter value to delete: ");
                        value = getInt();
                        boolean didDelete = theTree.delete(value);
                        if (didDelete) {
                            System.out.println("Deleted " + value );
                        } else {
                            System.out.println("Could not delete "
                                               + value);
                        }
                        break;
                    case 't':
                        System.out.print("Enter type 1, 2 or 3: ");
                        value = getInt();
                        theTree.traverse(value);
                        break;
                    default:
                        System.out.println("Invalid entry ");
                }
            }
        }

        public static String getString() throws IOException {
            InputStreamReader isr = new InputStreamReader(System.in);
            BufferedReader br = new BufferedReader(isr);
            String s = br.readLine();
            return s;
        }

        public static char getChar() throws IOException {
            String s = getString();
            return s.charAt(0);
        }

        public static int getInt() throws IOException {
            String s = getString();
            return Integer.parseInt(s);
        }
    }
```

4. Tuliskan output program pada listing nomor 3 dalam bentuk tree ketika memilih menu Show

jawaban

5. Tambahkan satu node dengan value tertentu! Tampilkan tree, tunjukkan dimana posisi node yang baru ditambahkan, dan jelaskan langkah penambahan node yang diimplementasikan pada method insert!

jawaban

6. Lakukan percarian salah satu leaf pada tree tersebut! Tuliskan outputnya dan penjelasan langkah pencarian yang diimplementasikan pada method find! Anda dapat menggambarkan ilustrasi pencarian node tersebut pada tree sebagai visualisasi langkah pencarian.

jawaban

7. Proses hapus node pada tree cukup kompleks dibandingkan dengan operasi lain. Langkah pertama yang dilakukan adalah mencari node yang akan dihapus. Terdapat 3 kemungkinan kondisi node yang akan dihapus, yaitu:
   a. Node yang akan dihapus adalah leaf, yang berarti node tersebut tidak memiliki children. Untuk kondisi ini, langkah penghapusan cukup sederhana
   b. Node yang akan dihapus memiliki satu child. Penghapusan node pada kondisi ini tidak semudah kondisi pertama
   c. Node yang akan dihapus memiliki dua children. Penghapusan node kondisi ini cukup rumit.

   Pada program nomor 3, hapuslah 3 node yang mewakili masing-masing kemungkinan kondisi node diatas. Gambarkan ilustrasi proses hapus masing-masing node tersebut berdasarkan langkah yang diimplentasikan pada program nomor 3.

jawaban

8. Terdapat tiga jenis operasi *traverse*, yaitu preorder, inorder, dan postorder. pada program nomor 3, jalankan masing-masing operasi *traverse* tersebut, tulis outputnya dan jelaskan perbedaan ketiga operasi tersebut!

jawaban

B. PRAKTIKUM

1. Tambahkan method pada class tree untuk mendapatkan *value* maksimum dan minimum pada tree tersebut!

2. Modifikasi method *insert* pada class tree sehingga tidak memungkinkan adanya *duplicate key* pada tree!

C. KESIMPULAN

Kesimpulan yang diperoleh dari pembahasan praktikum kali ini adalah:

1. Tentang proses *insert* pada binary tree

2. Tentang proses *find* pada binary tree

3. Tentang proses *delete* pada binary tree

4. Tentang operasi *traverse*

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*.

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:

a. Truncation. Mengambil beberapa digit dari key sebagai indeks
b. Folding. Menjumlahkan beberapa digit dari key, hasil penjumlahan sebagai indeks
c. 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.

A. 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:

a. Linier probing
b. Quadratic probing
c. 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

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!

jawaban

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!

jawaban

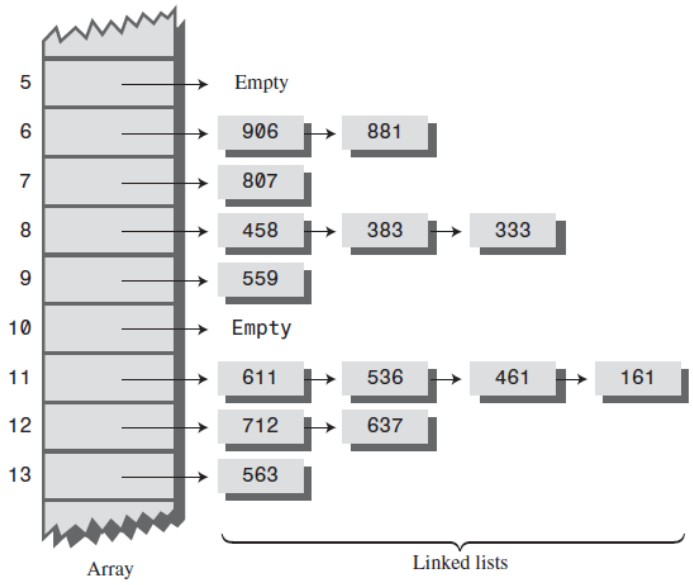
4. Lakukan pencarian berdasarkan key tertentu pada class HashTableApp!, jelaskan bagaimana proses pencarian pada method find?

jawaban

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

jawaban

6. **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;
    }
}
```

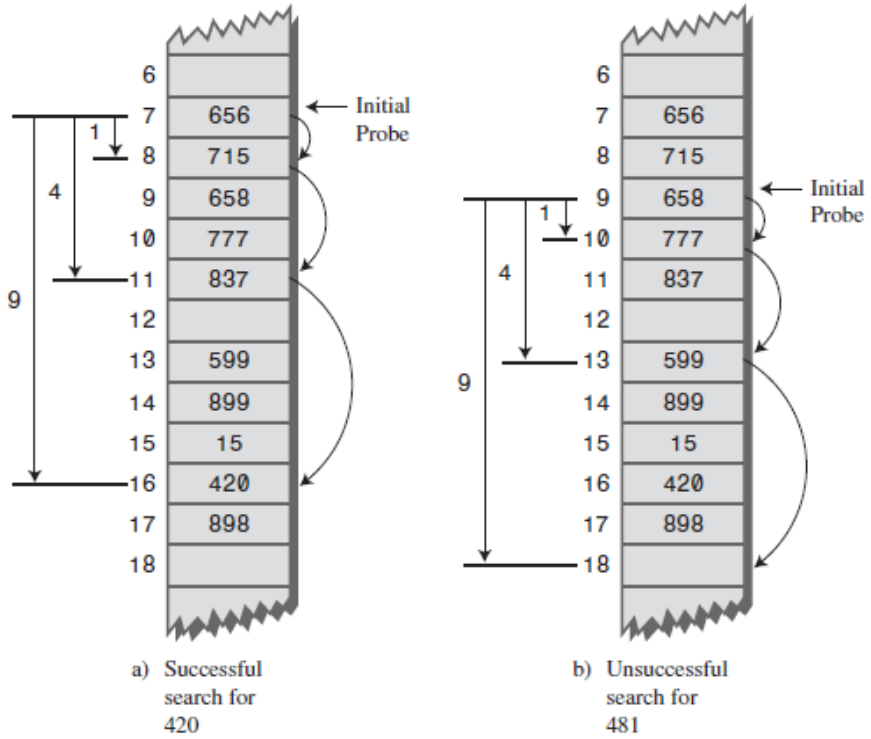
Lengkapi listing program tersebut dengan menambahkan class HashChainApp yang berisi method main. lakukan penambahan data sebagaimana soal nomer 1. Jalankan program dan tulis outputnya. jelaskan bagaimana penentuan indeks (sel array) yang digunakan untuk menyimpan tiap item tersebut!

jawaban

B. 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)= i^2. Sehingga jarak dari posisi awal adalah x+1^2, x+2^2, x+3^2, x+4^2, x+5^2, dan seterusnya hingga ditemukan sel kosong.



Gambar 8.2 Ilustrasi quadratic probe

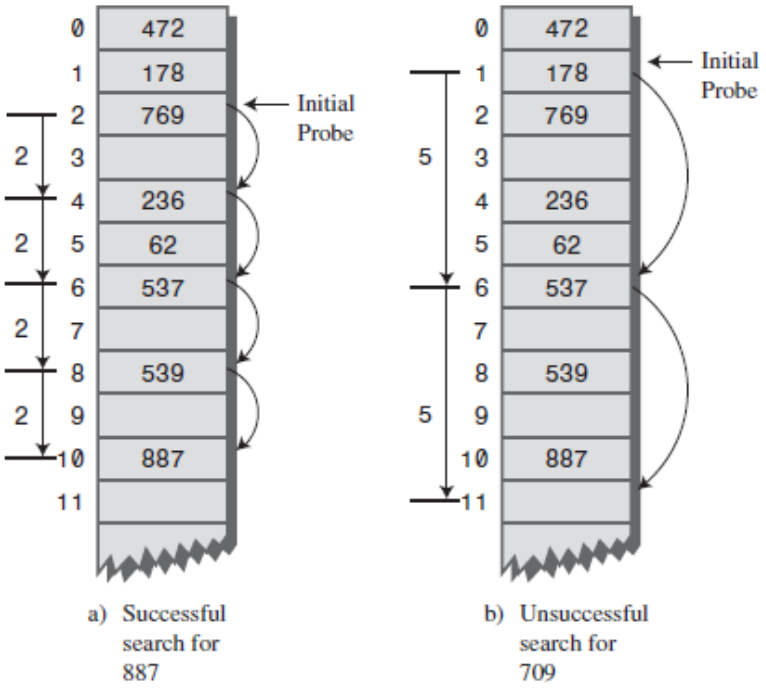
2. NIM ganjil: mengimplementasikan teknik double hashing
   Collision resolution dilakukan dengan melakukan hash terhadap key menggunakan fungsi hash yang lain dan menggunakan hasilnya sebagai jarak penentuan sel. Berikut ini fungsi yang dapat digunakan sebagai fungsi hash kedua:

   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

C. 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

PRAKTIKUM 9 - STRUKTUR DATA

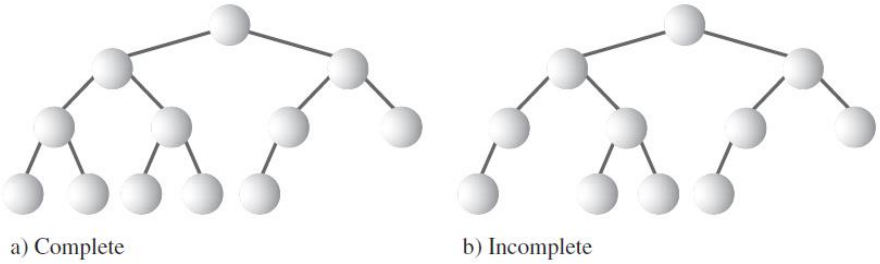
HEAPS

Learning outcomes:

1. Mahasiswa mampu menjelaskan struktur Heap
2. Mahasiswa mampu mengimplementasikan Heap MAX dalam program
3. Mahasiswa mampu mengimplementasikan manipulasi data pada struktur data heaps, yaitu: *insert item*, *removal*, dan *change priority*
4. Mahasiswa mampu mengimplementasikan Heap untuk program sorting menggunakan algoritma *Heapsort*

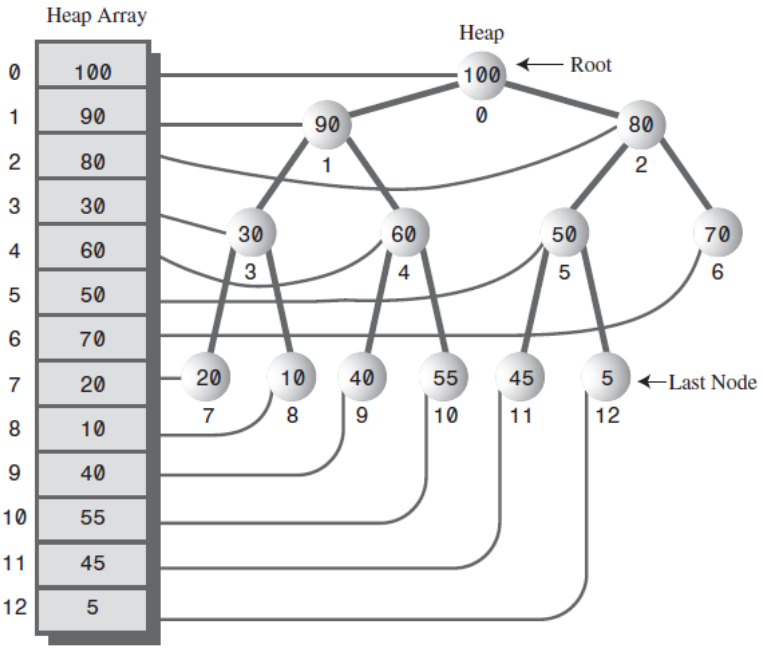
Heap adalah sebuah binary tree yang memiliki karateristik berikut ini:

1. Binary tree pada heap adalah tree yang complete, yaitu node pada tiap levelnya penuh dari kiri hingga ke kanan, hanya pada level terakhir yang dibolehkan tidak penuh.



Gambar 9.1 Binary tree yang komplit dan tidak komplit

2. Sebuah heap biasanya diimplementasikan sebagai sebuah array daripada menggunakan reference untuk menghubungkan tiap node
3. Setiap node pada sebuah heap memenuhi kondisi/syarat heap, yaitu:
   - Setiap key dari parent node lebih besar atau sama dengan key dari children node tersebut (Parent(key) >= Children(key)) untuk Heap MAX
   - Setiap key dari parent node lebih kecil atau sama dengan key dari children node tersebut (Parent(key) <= Children(key)) untuk Heap MIN



Gambar 9.2 Contoh sebuah heap dan penyimpanan heap pada Array

Tampak pada gambar 9.2, sebuah heap sebagai bentuk binary tree yang complete ketika direpresentasikan pada sebuah array, maka tiap cell array tersebut akan terisi penuh mulai indeks ke-0 hingga N-1.

Struktur data heap dapat digunakan untuk mengimplementasikan ADT priority queues, yaitu bentuk queue yang tersusun berdasarkan prioritas tiap item. Tingkat prioritas tersebut dinyatakan dengan key setiap item. Priority queue yang diimplementasikan menggunakan sebuah Heap MAX, dimana nilai key maksimum berada pada root, disebut dengan descending-priority queues. Sebaliknya, ketika diimplementasikan pada Heap MIN (root berisi nilai key minimal), maka disebut ascending-priority queues.

A. PENDAHULUAN

1. Listing program java untuk Heap

   Berikut ini listing program untuk Heap. Tulis dan pelajari listing ini

```
public class Node {

    private int data;

    public Node(int key) {
        data = key;
    }

    public int getKey() {
        return data;
    }

    public void setKey(int id) {
        data = id;
    }
}
```

```
public class Heap {

    private Node[] heapArray;
    private int maxSize;
    private int currentSize;

    public Heap(int size) {
        maxSize = size;
        currentSize = 0;
        heapArray = new Node[size];
    }

    public boolean isEmpty() {
        return currentSize == 0;
    }
```

```
    public boolean insert(int key) {
        if (currentSize == maxSize) {
            return false;
        }
        Node newNode = new Node(key);
        heapArray[currentSize] = newNode;
        trickleUp(currentSize++);
        return true;
    }
```

```
    public void trickleUp(int index) {
        int parent = (index - 1) / 2;
        Node bottom = heapArray[index];

        while (index > 0 &&
            heapArray[parent].getKey() <
            bottom.getKey()) {

            heapArray[index] =
                    heapArray[parent];
            index = parent;
            parent = (parent - 1) / 2;
        }
        heapArray[index] = bottom;
    }
```

```
public Node remove() {
    Node root = heapArray[0];
    heapArray[0] =
           heapArray[--currentSize];
    trickleDown(0);
    return root;
}
```

```
public void trickleDown(int index) {
    int largerChild;
    Node top = heapArray[index];
    while (index < currentSize / 2) {
       int leftChild = 2 * index + 1;
       int rightChild = leftChild + 1;

       if (rightChild < currentSize &&
        heapArray[leftChild].getKey()<
        heapArray[rightChild].getKey())
        {
            largerChild = rightChild;
        } else {
            largerChild = leftChild;
        }

       if (top.getKey() >=
        heapArray[largerChild].getKey()
        {
            break;
        }

       heapArray[index] =
            heapArray[largerChild];
       index = largerChild;
    }

    heapArray[index] = top;
}
```

```
public void displayHeap() {
    System.out.println("Heap Array: ");
    for (int i = 0; i < currentSize; i++) {
        if (heapArray[i] != null) {
            System.out.print(heapArray[i].getKey() + " ");
        } else {
            System.out.println("--");
        }
    }
    System.out.println("");
    int nBlanks = 32;
    int itemsPerRow = 1;
    int column = 0;
    int j = 0;
    String dots = "...............................";
    System.out.println(dots + dots);
    while (currentSize > 0) {
        if (column == 0) {
            for (int k = 0; k < nBlanks; k++) {
                System.out.print(' ');
            }
        }
        System.out.print(heapArray[j].getKey());
        if (++j == currentSize) {
            break;
        }
        if (++column == itemsPerRow) {
```

```
                nBlanks /= 2;
                itemsPerRow *= 2;
                column = 0;
                System.out.println();
            } else {
                for (int k = 0; k < nBlanks * 2 - 2; k++) {
                    System.out.print(' ');
                }
            }
        }
        System.out.println("\n" + dots + dots);
    }

    public void displayArray() {
        for (int j = 0; j < maxSize; j++) {
            System.out.print(heapArray[j].getKey() + " ");
        }
        System.out.println("");
    }

} //akhir class Heap
```

Lengkapi listing tersebut dengan sebuah class HeapApp berisi method main. deklarasikan sebuah heap dengan ukuran 35, lakukan penambahan 12 item, tampilkan heap tersebut. Jalankan program, bagaimana output program yang telah anda lengkapi?

jawaban

Dari output program tersebut, tampak bahwa heap yang diimplementasikan adalah

Heap MAX / MIN *(coret salah satu)*

karena

_____

_____

2. Heap Array

   Perlu diingat bahwa heap adalah sebuah complete binary tree. Dalam bentuk heap (tree) maka akan jelas terlihat parent dan child suatu node. Sedangkan, jika heap tersebut direpresentasikan pada array (sehingga disebut Heap Array), maka visualisasi struktur tree tidak tampak, sehingga bagaimana mengetahui parent dan child suatu node? Pada heap array, tiap node disimpan pada tiap cell array, oleh karena itu, parent dan children suatu node dapat ditelusuri dari posisi indeks node tersebut.

   Pada program nomor 1, tuliskan kembali tampilan heap dan heap array sesuai data yang telah anda masukkan! Tuliskan indeks tiap item heap array, begitu pula tuliskan indeks item tersebut pada heap. Amati dimana posisi parent dan children suatu node pada heap array! Pada heap Array, beri garis sebagai tanda hubung antara root node dengan left child dan right child dari root. Begitu pula seterusnya, jika right child dari root adalah parent node, hubungkan dengan left child dan right child node tersebut!

   jawaban

   Dari susunan array tersebut diketahui untuk sebuah node pada indeks x, maka:

Indeks Parent	**Indeks Children**	
x	*Left child*	……
	Right child	……
……	x	

3. Menambahkan sebuah item pada Heap

   Langkah untuk menambahkan item pada heap, yaitu:

   a. Tambahkan node baru yang merepresentasikan sebuah item dengan key tertentu pada bagian akhir heap, yaitu sebagi last node

b. Lakukan trickle up untuk menempatkan node baru tersebut sehingga berada di posisi yang tepat pada struktur heap

Trickle, bisa juga disebut dengan istilah bubble atau percolate, yaitu proses untuk memindahkan node baik keatas (trickle up) ataupun kebawah (trickle down) pada path node tersebut secara bertahap dengan cara membandingkan node di tiap tahap apakah node tersebut sudah berapa pada posisi yang sesuai atau belum. Jika posisi tidak sesuai maka dilakukan pertukaran node tersebut dengan node yang dibandingkan hingga menjadi struktur heap yang sesuai.

Pada program nomor 1, dari 12 item yang telah ada, lakukan penambahan sebuah item. Tampilkan heap sebelum dan setelah penambahan item. Gambarkan langkah-langkah penambahan item tersebut mulai dari penambahan node baru sebagai last node hingga node tersebut berada pada posisi yang tepat sesuai struktur heap! Beri penjelasan pada tiap langkah tersebut!

jawaban

4. Menghapus sebuah item pada Heap
   Perlu ingat bahwa removal pada heap berarti menghapus node yang memiliki nilai key maksimum (untuk heap MAX) atau menghapus node yang memiliki nilai key minimum (untuk heap MIN). Sehingga node yang dihapus adalah root node.

Berikut ini langkah-langkah removal pada heap:

a. Hapus root node
b. Pindahkan last node pada root
c. Lakukan trickle down untuk menempatkan node tersebut pada posisi yang tepat sesuai struktur heap

Pada program nomor 1, panggil method remove() pada class HeapApp. Tampilkan heap sebelum dan setelah penghapusan. Gambarkan langkah-langkah removal/penghapusan tersebut! Jelaskan tiap tahapnya!

jawaban

5. Merubah key (priority)

   Dengan adanya method trickleUp() dan trickleDown(), maka algoritma untuk merubah suatu key (representasi dari nilai prioritas pada priority queue) sebuah node dapat mudah diimplementasikan. Secara garis besar, berikut ini langkah untuk merubah key suatu node pada heap:

   a. Ubah nilai key pada node tertentu (ditunjukkan dengan indeks suatu cell pada heap array yang ingin dirubah) dengan nilai yang baru.
   b. Jika nilai key yang lama kurang dari nilai yang baru, maka lakukan trickle up

c. Jika tidak, maka lakukan trickle down.

Tuliskan sebuah method *change* dengan parameter indeks dan nilai yang baru serta implmentasikan algoritma merubah key pada method tersebut! Panggil method tersebut pada class HeapApp, tampilkan heap sebelum perubahan key dan stelah key dirubah.

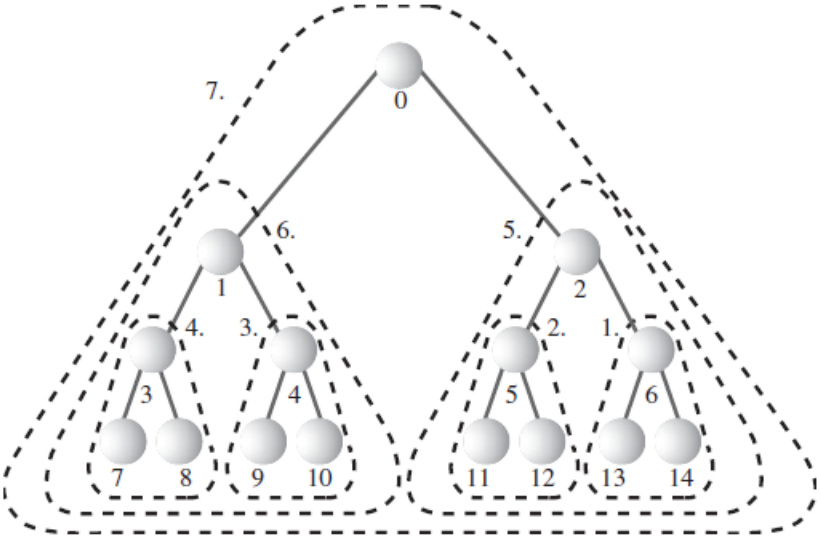
jawaban

B. PRAKTIKUM

Implementasi lain dari struktur heap adalah digunakan untuk sorting yang dikenal sebagai ***Heapsort***.
Buatlah program sorting menggunakan heapsort.

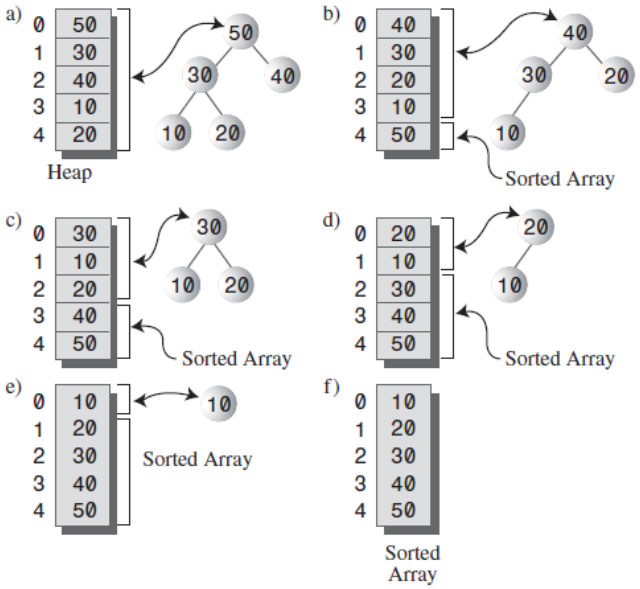
Anda dapat mengimplementasikannya berdasarkan langkah-langkah berikut ini:

a. Tentukan array dengan data random
b. Ubah posisi indeks tiap item pada array tersebut menjadi susunan struktur heap dengan cara melakukan trickle down item pada indeks ke-(N/2-1) hingga indeks ke-0.



Gambar 9.3 urutan penerapan trickleDown

c. Remove heap dan simpan dalam heapArray mulai indeks terakhir



Gambar 9.4 *heap removal* dan penyimpanan *removed node* pada heapArray

Langkah tersebut dapat diimplementasikan dalam 3 class, yaitu:

a. Class Node, sebagai objek dari item
b. Class Heap, secara umum, class ini sama dengan class Heap pada listing nomor 1 tugas pendahuluan, dengan beberapa penyesuaian sehingga dapat digunakan untuk melakukan pengurutan. Berikut ini method yang diperlukan:
   - Method Heap sebagai constructor dengan parameter size. Di dalamnya dilakukan inisialisasi variable maxSize, currentSize, dan heapArray
   - Method insertAt(indeks, value), berisi code untuk melakukan penambahan item pada heapArray pada indeks sesusai parameter.
   - Method trickleDown(indeks). Sama dengan listing nomor 1 tugas pendahuluan
   - Method displayArray(), untuk menampilkan isi array. Sama dengan listing nomor 1 tugas pendahuluan
   - Method HeapSort(), berisi code untuk trickle down item pada indeks ke-(N/2-1) hingga indeks ke-0. Setelah itu, dilakukan removal secara berulang. Item yang dihapus (node dengan nilai maksimum) disimpan dalam heapArray mulai indeks yang terakhir hingga indeks pertama.
c. Class HeapSortApp, berisi method main yang didalamnya dilakukan:
   - pemanggilan method insertAt(indeks, value) pada class heap untuk menambahkan item pada HeapArray. Lakukan pemanggilan method ini dalam perulangan dengan batas indeks tertentu. Value yang ditambahkan adalah nilai random.
   - menampilkan array sebelum diurutkan
   - memanggil method HeapSort() pada class Heap untuk menjalankan proses sorting
   - menampilkan array setelah diurutkan

Pengayaan

Modul ini memberikan gambaran implementasi Heap MAX. Sesuaikan listing program pada tugas pendahuluan modul ini sehingga merepresentasikan implementasi dari Heap MIN.

C. KESIMPULAN

Kesimpulan yang diperoleh dari pembahasan praktikum kali ini adalah:

1. Tentang struktur data Heap

2. Tentang manipulasi data pada heap: insert item, removal, change priority

3. Tentang Heapsort

PRAKTIKUM 10 - STRUKTUR DATA

GRAPHS

Learning outcomes:

1. Mahasiswa mampu menjelaskan struktur data Graph dan jenis-jenisnya
2. Mahasiswa mampu merepresentasikan Graph dalam bentuk *adjacency matrix* dan *adjacency list*
3. Mahasiswa mampu mengimplementasikan *directed graph* dan *undirected graph* pada program
4. Mahasiswa mampu mengimplementasikan algoritma *depth-first search* dan *breath-first search* pada *undirected graph*
5. Mahasiswa mampu mengimplementasikan *minimum spanning tree* pada *undirected graph*
6. Mahasiswa mampu mengimplementasikan *Topological sorting* dengan *directed graph*.

Graph adalah struktur data yang hampir menyerupai tree. Graph terdiri dari simpul dan busur yang secara matematis dinyatakan sebagai:

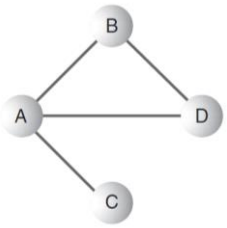
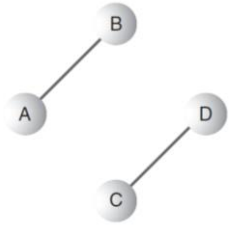
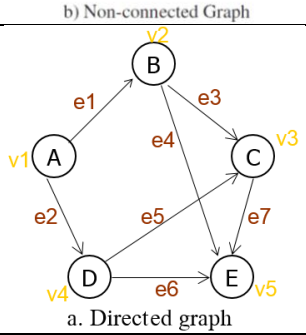
$$G = (V, E)$$

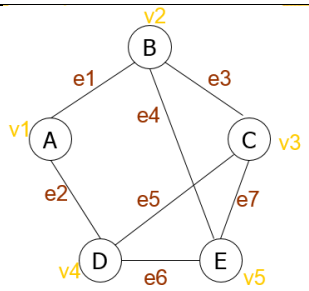
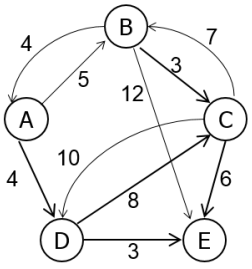
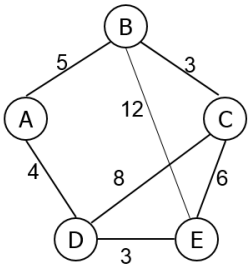
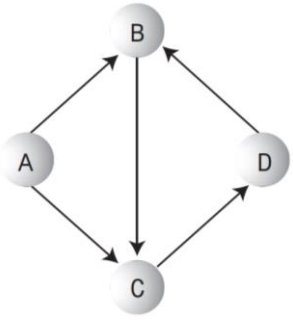
dimana G adalah Graph, V adalah simpul atau vertex (jamaknya vertices) atau node atau titik, dan E adalah busur atau edge atau arc.

Graph dapat digunakan untuk merepresentasikan jaringan, peta (mencari jalur terpendek), penjadwalan (perencanaan proyek), dll.

A. PENDAHULUAN

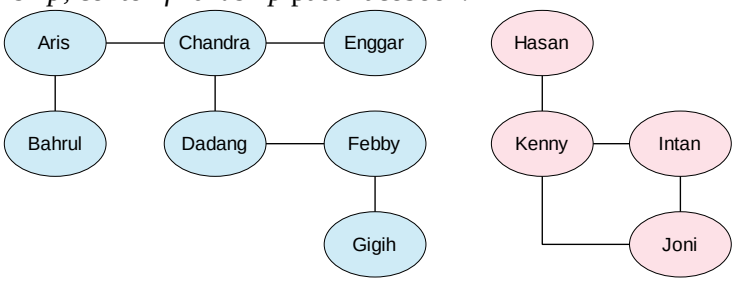
1. Beberapa istilah berkaitan dengan struktur data Graph

Istilah	Ilustrasi
Adjacency :	
Path :	
Successor dan Predecessor :	
Connected graph dan non-connected graph :	 a) Connected Graph  b) Non-connected Graph
Directed graph dan undirected graph :	 a. Directed graph

	 b. Undirected graph
Weighted graph :	 a. Directed graph  b. Undirected graph
Degree (of vertices) Pada directed graph, dikategorikan menjadi *indegree* dan *outdegree* :	
Cycle dan tree :	 Graph dengan sebuah cycle (B-C-D-B)

Directed acyclic graph (DAG) :	

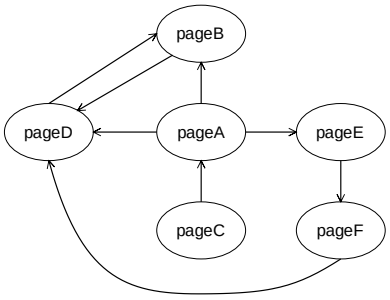
2. Secara umum, Graph dapat dikategorikan menjadi:
   a. Undirected graph
      Tipe graph ini dapat digunakan untuk merepresentasikan *symmetric relationship*, contoh *friendship* pada facebook.



Gambar 10.1 Contoh undirected graph: friendship

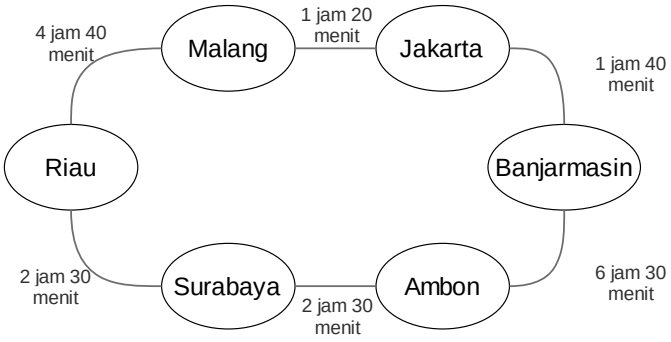
Pada contoh tersebut, tiap orang direpresentasikan sebagai vertex dan hubungan pertemanan ditunjukkan dengan adanya edge yang menghubungkan vertices. Disana berlaku symmetric relationship, Aris adalah teman Bahrul, begitu pula sebaliknya, Bahrul adalah teman Aris. Sebagian orang bisa juga tidak mengenali (tidak berhubungan) dengan sebagian yang lain sehingga terdapat path yang tidak terhubung pada contoh tersebut (unconnected graph).

   b. Directed graph
      Berbeda dengan undirected graph, directed graph menunjukkan asymmetric relationship, contoh *page link* pada website. Satu link dapat terhubung dengan link lain, namun link tersebut belum tentu langsung terhubung dengan link awal.



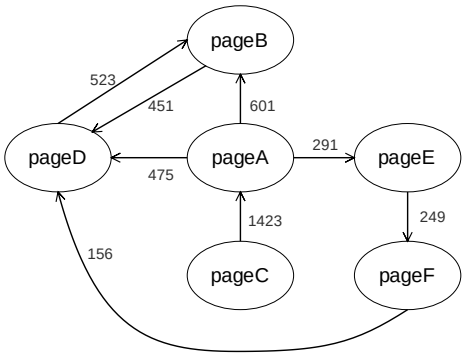
Gambar 10.2 Contoh Directed graph: page link

   c. Weighted Undirected graph
      Bobot pada graph dapat merepresentasikan angka nyata, misal jarak atau waktu penerbangan yang ditempuh dari satu wilayah ke wilayah yang lain.



Gambar 10.3 Contoh Weighted undirected graph waktu penerbangan antar daerah

d. Weighted Directed graph
Pada contoh directed graph (gambar 10.2), tiap edge/simpul dapat diberi bobot berupa waktu bagi link yang dituju untuk merespon dan menampilkan data.



Gambar 10.4 Contoh Weighted Directed graph: pagelink dengan bobot waktu manampilkan data

Sebutkan contoh lain untuk setiap 4 tipe graph tersebut, jelaskan dan beri ilustrasi!

jawaban

3. Representasi graph: adjacency matrix, adjacency list
   Implementasi graph pada program dilakukan dengan menyimpan semua vertex dan edge. Teknik yang dapat digunakan yaitu: adjacency matrix dan adjacency linked list. Pada teknik adjacency matrix, nama vertex disimpan pada array (atau bisa juga dalam hash table) dan hubungan antar vertex (adjacent) ditandai pada sebuah matriks (array 2 dimensi). Keberadaan edge yang menghubungkan dua vertices dapat dinyatakan dengan nilai 1 sedangkan nilai 0 jika tidak terdapat edge, atau dapat pula menggunakan Boolean true – false. Pada weighted graph, nilai yang disimpan pada matriks atau linked list adalah bobot dari edge dari satu vertex ke vertex tertentu.

   Lengkapi matriks berikut ini sesuai graph pada Gambar 10.1

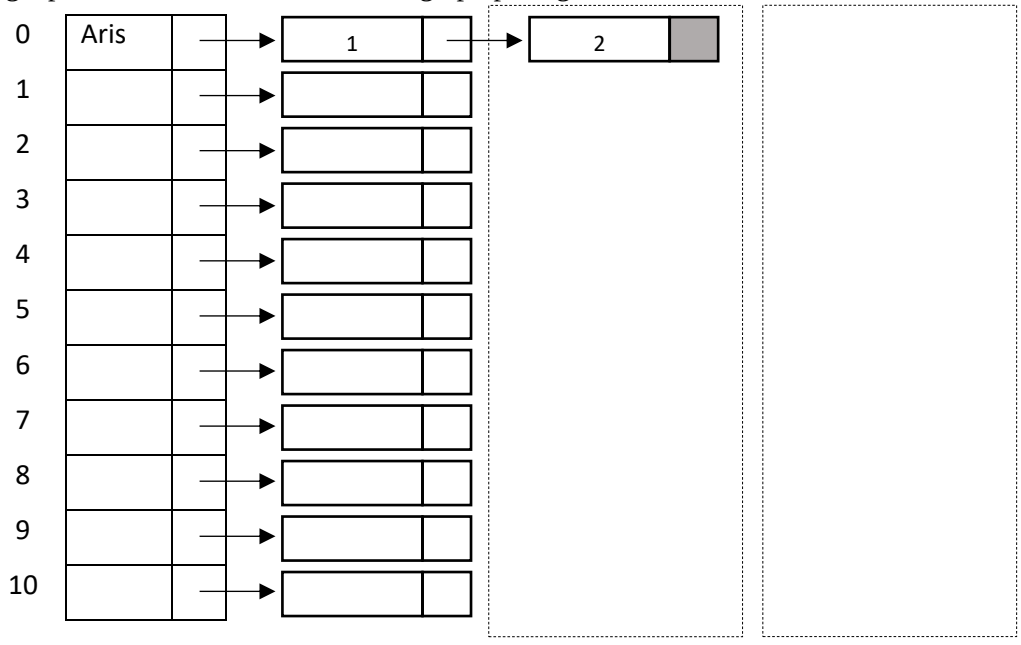
0	Aris
1	Bahrul
2	Chandra
3	Dadang
4	Enggar
5	Febby
6	Gigih
7	Hasan
8	Intan
9	Joni
10	Kenny

Array untuk nama vertex

	0	1	2	3	4	5	6	7	8	9	10
0		T	T								
1	T										
2	T										
3											
4											
5											
6											
7											T
8											
9											
10								T			

Adjacency matrix

   Cara lain untuk merepresentasikan graph adalah menggunakan linked list. Teknik ini dikenal dengan adjacency list (adjacency linked list).
   Lengkapi Linked list berikut ini sesuai graph pada gambar 10.1



Adjacency linked list

4. Bentuk adjacency matrix dan adjacency list pada nomer 3 merepresentasikan undirected graph. Tuliskan adjacency matrix dan adjacency list untuk directed graph sesuai contoh yang anda tuliskan pada nomor 2!

jawaban

5. Implementasi graph pada program java

```
public class Vertex {

    public char label;
    public boolean wasVisited;

    public Vertex(char label) {
        this.label = label;
        wasVisited = false;
    }
}
```

```
public class Graph {

    private final int MAX_VERTS = 20;
    private Vertex vertexList[];
    private int adjMat[][];
    private int nVerts;

    public Graph() {
        vertexList = new Vertex[MAX_VERTS];
        adjMat = new int[MAX_VERTS][MAX_VERTS];
        nVerts = 0;
        for (int i = 0; i < MAX_VERTS; i++) {
          for (int j = 0; j < MAX_VERTS; j++) {
                adjMat[i][j] = 0;
          }
        }
    }
```

```
    public void addVertex(char label) {
        vertexList[nVerts++] =
                            new Vertex(label);
    }
```

```
    public void addEdge(int start, int end) {
        adjMat[start][end] = 1;
        adjMat[end][start] = 1;
    }
```

```
    public void display() {
        System.out.println("Adjecency:");
        for (int row = 0; row < nVerts; row++)
        {
            for (int col = 0; col < row;
                 col++) {
                if (adjMat[row][col] == 1) {
                   System.out.println(
                      vertexList[row].label +
                      " -- " +
                      vertexList[col].label);
                }
            }
        }
        System.out.println("");
    }
}//end class Graph
```

```
public class GraphApp {

    public static void main(String[] args) {

        Graph theGraph = new Graph();
        theGraph.addVertex('A'); // 0
        theGraph.addVertex('B'); // 1
        theGraph.addVertex('C'); // 2
        theGraph.addVertex('D'); // 3
        theGraph.addVertex('E'); // 4

        theGraph.addEdge(0, 1); // AB
        theGraph.addEdge(1, 2); // BC
        theGraph.addEdge(0, 3); // AD
        theGraph.addEdge(3, 4); // DE
        theGraph.display();
    }
}
```

Output:

Ilustrasi Graph:

Graph yang dibentuk pada program tersebut adalah:

Directed Graph / Undirected Graph *(coret salah satu)*

karena

_____

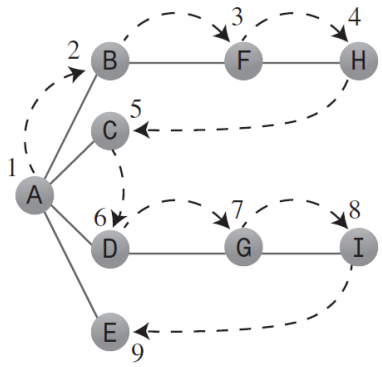
_____

hal tersebut dinyatakan dalam baris program:

_____

_____

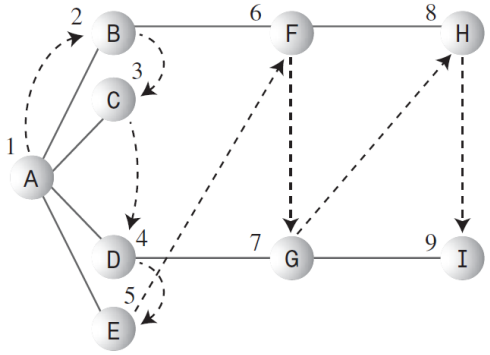
6. Searches: depth-first search, breadth-first search
   Operasi mendasar yang ada pada Graph adalah mencari vertices yang dapat dicapai dari vetex tertentu. Misalkan pada jalur penerbangan, dari suatu wilayah dapat menuju wilayah yang lain, namun terdapat wilayah yang tidak dapat dituju karena tidak masuk jalur penerbangan yang dituju.
   Terdapat dua metode untuk pencarian graph, yaitu:
   a. Depth-first search (DFS)
      - DFS diimplementasikan menggunakan stack. Stack digunakan untuk mengingat vertex yang dituju ketika ketika sudah mencapai vertex terdalam.
      - Contoh:



Gambar 10.5 Contoh urutan DFS pada graph

      - Rules:
        1. Selama memungkinkan, kunjungi vertex terhubung yang belum dikunjungi, beri tanda vertex tersebut, dan push pada stack
        2. Jika aturan 1 tidak dapat dipenuhi, maka, selama memungkinkan, pop sebuah vertex keluar dari stack
        3. Jika aturan 1 dan 2 tidak dapat dipenuhi, maka proses DFS telah selesai.

   b. Breadth-first search (BFS)
      - BFS diimplementasikan menggunakan queue
      - Contoh:



Gambar 10.6 Contoh urutan BFS pada graph

      - Rules:
        1. Kunjungi vertex selenjutnya yang berhubungan (jika ada), tandai vertex tersebut, dan insert pada queue
        2. Jika tidak dapat melakukan rule 1 karena tidak ada vertex terhubung yang belum dikunjungi, maka remove vertex dari queue (jika ada), dan jadikan vertex tersebut sebagai current vertex.
        3. Jika tidak dapat melakukan rule 2 karena queue telah kosong, maka proses BFS telah selesai

Implementasi DFS pada program

```
    public void dfs() {
        System.out.println("Visit by using" +
                    " DFS algorithm: ");
        vertexList[0].wasVisited = true;
        displayVertex(0);
        theStack.push(0);
        while (!theStack.isEmpty()) {
          int v = getAdjUnvisitedVertex(
                         theStack.peek());
          if (v == -1) {
            theStack.pop();
          } else {
            vertexList[v].wasVisited = true;
            displayVertex(v);
            theStack.push(v);
          }
        }
        System.out.println("");
        resetFlags();
    }
```

```
public void displayVertex(int v) {
   System.out.print(vertexList[v].label+" ");
}
```

```
public int getAdjUnvisitedVertex(int v) {
    for (int i = 0; i < nVerts; i++) {
        if (adjMat[v][i] == 1 &&
            vertexList[i].wasVisited ==
            false) {
            return i;
        }
    }
    return -1;
}
```

```
private void resetFlags() {
    for (int i = 0; i < nVerts; i++) {
        vertexList[i].wasVisited = false;
    }
}
```

Tambahkan listing tersebut pada class Graph di listing nomor 5. Lengkapi dengan class Stack dan inisialisasi pemanggilan class Stack pada constructor class Graph, misal sebagaimana berikut:

```
public class Graph{
    ...
    private Stack theStack;

    public Graph(){
        ...
        theStack = new Stack(MAX_VERTS);
    }
    ...
```

Panggil method dfs() pada class GraphApp. Jalankan program, Bagaimana outpunya? Jelaskan!

jawaban

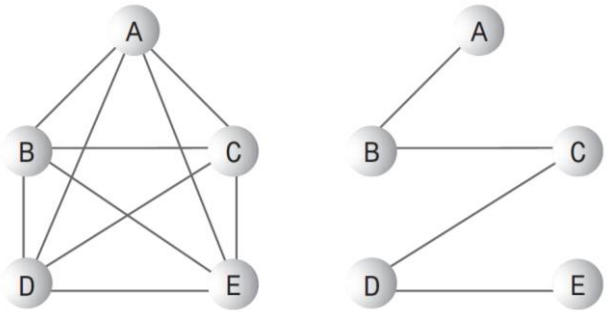
7. Implementasi BFS pada program

```
public void bfs() {
    System.out.println("Visit by using"+
                    " BFS algorithm: ");
    vertexList[0].wasVisited = true;
    displayVertex(0);
    theQueue.insert(0);
    int v2;
    while (!theQueue.isEmpty()) {
        int v1 = theQueue.remove();
        while ((v2 =
              getAdjUnvisitedVertex(v1))
              != -1) {
            vertexList[v2].wasVisited =
              true;
            displayVertex(v2);
            theQueue.insert(v2);
        }
    }
    System.out.println("");
    resetFlags();
}
```

Tambahkan method tersebut pada class Graph. Tambahkan pula class queue, kemudian sebagaimana class stack sebelumnya, lakukan inisialisasi dan deklarasi pemanggilan class queue pada class Graph.
Panggil method bfs() pada class GraphApp. Jalankan program, Bagaimana output yang dihasilkan? Jelaskan!

jawaban

8. Minimum Spanning Tree
Minimum spanning tree (MST) merupakan algoritma yang digunakan untuk menghubungkan setiap vertices pada graph dengan jumlah edge minimum sehingga edge yang berlebih akan dihapus. Hasil dari penerapan MST adalah sebuah graph dengan jumlah edge minimum yang dibutuhkan untuk meghubungkan vertices. Jumlah edge pada suatu MST selalu kurang dari 1 dari jumlah vertices (E = V - 1)



(a) (b)

Gambar 10.7 (a) graph dengan jumlah edge lebih; (b) graph dengan jumlah edge minimum

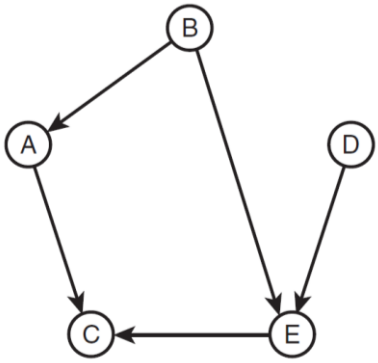
Berikut ini adalah implementasi MST pada program. Tambahkan method mst() berikut pada class Graph. Buatlah Graph yang sesuai dengan ilustrasi pada Gambar 10.7 (a) kemudian panggil method mst(). Jalankan program, jelaskan bagaimana hasilnya!

```
public void mst() {
    System.out.println("Minimum spanning" +
                       " tree: ");
    vertexList[0].wasVisited = true;
    theStack.push(0);
    while (!theStack.isEmpty()) {
        int currentVertex = theStack.peek();
        int v = getAdjUnvisitedVertex(
                              currentVertex);
        if (v == -1) {
            theStack.pop();
        } else {
            vertexList[v].wasVisited = true;
            theStack.push(v);

            displayVertex(currentVertex);
            System.out.print(" -- ");
            displayVertex(v);
            System.out.println("");
       }
    }
    resetFlags();
}
```

jawaban

B. PRAKTIKUM

1. Implementasikan directed graph
   Tuliskan listing program yang mengimplementasikan directed graph sesuai gambar 10.8.
   Anda dapat melakukan modifikasi pada listing program nomor 5 tugas pendahuluan sehingga merepresentasikan directed graph.



Gambar 10.8 Directed Graph

2. Cennectivity pada directed graph
   Pada directed graph, adakalanya tidak semua vertex dapat dituju dari titik tertentu. Berdasarkan gambar 10.8, terlihat bahwa jika start pada titik A, maka kita dapat menuju titik C tapi tidak dapat menuju titik lain. Sedangkan jika memulai pada titik C, maka tidak ada vertex yang dapat dituju.
   Modifikasi method dfs() sehingga menampilkan tabel koneksi (connectivity table)

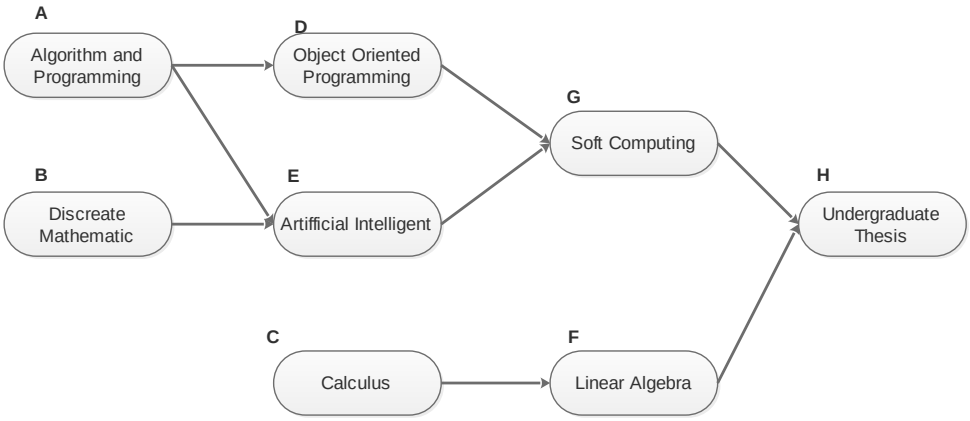
```
run:
Adjecency:
A --> C
B --> A
B --> E
D --> E
E --> C

Connectivity Table:
A C
B A C E
C
D E C
E C
BUILD SUCCESSFUL (total time: 1 second)
```

Gambar 10.9 Output tabel koneksi

3. Topological Sorting dengan directed graph
   Topological sorting adalah operasi lain yang dapat dimodelkan dengan graph. Operasi ini diperlukan pada situasi dimana item atau kejadian perlu disusun berdasarkan urutan tertentu. Misalkan dalam memilih matakuliah, terdapat matkuliah yang harus diambil lebih dahulu sebelum matakuliah lain. Contoh, berdasarkan gambar 10.10 maka sebelum mengambil skripsi maka mahasiswa perlu mengambil matakuliah berdasarkan urutan berikut:

   B A E D G C F H atau **C F B A E D G H**

   Dengan penyusunan seperti kondisi tersebut, suatu graph dikatakan “topologically sorted”



Gambar 10.10 Matakuliah prasyarat

Selain pada persyaratan matakuliah, topological sorting dalam memodelkan *job scheduling*. Misal, untuk merakit mobil, maka kerangka mobil harus dipasang sebelum ban, dan mesin diatur lebih dahulu sebelum dikunci pada kerangka mobil. Perakitan mobil dapat dimodelkan dalam graph untuk memodelkan ribuan operasi proses perakitan mobil yang dilakukan berdasarkan urutan yang tepat.

Dua tahapan pada topological sorting, yaitu:

a. Cari sebuah vertex yang tidak memiliki successor
b. Hapus vertex tersebut dari graph dan tambahkan label vertex tersebut pada bagian awal list.

Kedua tahap tersebut dilakukan secara berulang hingga sebuat vertex tidak ada.

Perlu diingat bahwa topological sort tidak dapat dilakukan pada graph yang memiliki cycle. Topological sort dapat dilakukan pada *directed acyclic graph* (DAG).

Implementasikan topological sort pada program. Berikut ini method yang merepresentasikan topological sort. Tuliskan method berikut pada class Graph. Lengkapi dan gunakan untuk melakukan topological sort sesuai graph pada gambar 10.10!

```
public void topo() {
    int orig_nVerts = nVerts;
    while (nVerts > 0) {
        int currentVertex = noSuccessors();
        if (currentVertex == -1) {
            System.out.println("ERROR: Graph has cycles");
            return;
        }
        sortedArray[nVerts - 1] = vertexList[currentVertex].label;
        deleteVertex(currentVertex);
    }
    System.out.println("Topologically sorted order: ");
    for (int i = 0; i < orig_nVerts; i++) {
        System.out.print(sortedArray[i] + " ");
    }
    System.out.println("");
}
```

```
    private int noSuccessors() {
        boolean isEdge;
        for (int row = 0; row < nVerts; row++) {
            isEdge = false;
            for (int col = 0; col < nVerts; col++) {
                if (adjMat[row][col] > 0) {
                    isEdge = true;
                    break;
                }
            }
            if (!isEdge) {
                return row;
            }
        }
        return -1;
    }

    private void deleteVertex(int delVert) {
        if (delVert != nVerts - 1) {
            for (int j = delVert; j < nVerts - 1; j++) {
                vertexList[j] = vertexList[j + 1];
            }
            for (int row = delVert; row < nVerts - 1; row++) {
                moveRowUp(row, nVerts);
            }
            for (int col = delVert; col < nVerts - 1; col++) {
                moveColLeft(col, nVerts - 1);
            }
        }
        nVerts--;
    }

    private void moveRowUp(int row, int length) {
        for (int col = 0; col < length; col++) {
            adjMat[row][col] = adjMat[row + 1][col];
        }
    }

    private void moveColLeft(int col, int length) {
        for (int row = 0; row < length; row++) {
            adjMat[row][col] = adjMat[row][col + 1];
        }
    }
}
```

Pengayaan:

Pada modul ini, pendekatan yang digunakan untuk merepresentasikan graph pada program adalah dengan *adjacency matrix*. Cobalah mengimplementasikan graph menggunakan *adjacency linked list*.

C. KESIMPULAN

Kesimpulan yang diperoleh dari pembahasan praktikum kali ini adalah:

1. Tentang struktur data Graph

2. Tentang representasi graph: adjacency matrix dan adjacency list

3. Tentang Depth-first search dan Breath-first search

4. Tentang Topological Sort