



ARRAY & LINKED LIST

MODUL 1

Standar kompetensi:

1. Mahasiswa mengetahui perbedaan array dan linked list.
 2. Mahasiswa dapat membuat dan menggunakan array dan linked list dalam suatu kasus.
 3. Mahasiswa dapat melakukan operasi create, insert, update dan delete data di dalam array dan linked list.
-

Pengenalan Array

Array adalah sebuah variabel yang bisa menyimpan beberapa item dengan tipe data yang sama. Array digunakan karena penggunaan variabel biasa dinilai kurang efisien. Jika kita memiliki 3 variabel yang menyimpan tipe data integer seperti berikut:

```
int angka1;  
int angka2;  
int angka3;  
int angka4;
```

Akan lebih efektif jika semua variabel itu disatukan. Karena untuk proses selanjutnya dapat lebih mudah. Proses selanjutnya ini melingkupi antara lain adalah sorting, searching, dynamic programming, pembuatan tree, dan lain sebagainya.

Dalam java, array akan diberikan perlakuan yang sama seperti sebuah objek hasil bentukan dari class, sehingga cara menginisialisasinya mirip dengan cara menginisialisasikan objek. Array dibuat dengan cara menginstantiasikan isi atau besarnya. Contoh:

```
int kumpulanAngka[];  
kumpulanAngka = new int[20];
```

Atau jika kedua baris di atas disatukan

```
int kumpulanAngka[] = new int[20];
```

Pada contoh diatas, variabel kumpulanAngka akan menampung 20 buah nilai integer. Selain mendeklarasikan array kosong, pendeklarasian array juga bisa langsung memberikan nilai. Berikut adalah cara untuk membuat array yang tidak kosong.

Cara 1 :

```
int kumpulanAngka[] = {1,2,3,4};
```

Cara 2:

```
int kumpulanAngka[];  
int kumpulanAngka = new int[]{1,2,3,4};
```

Array sering memiliki banyak elemen. Untuk itu diberikan kemudahan dalam mengakses setiap elemen dalam array tersebut menggunakan index. Index merupakan nomor urut dari setiap elemen array. Index pada array dimulai dari 0 hingga panjang array dikurangi 1. Berikut adalah contoh cara pengaksesan array pada suatu elemen tertentu:

```
kumpulanAngka[0] = 10;  
kumpulanAngka[1] = 7;
```

Struktur data Array

Sebagai sebuah struktur data, array harus dapat menangani proses yang sering dilakukan ketika bekerja dengan sebuah struktur data. Proses-proses tersebut antara lain adalah:

1. Insert
2. Delete
3. Pencarian
4. Menampilkan isi array

Contoh dari implementasi fungsi-fungsi tersebut dapat dilihat dalam contoh program di bawah ini.

Catatan: program terdiri dari file *HighArray.java* dan *HighArrayApp.java*.

Class berikut merupakan class untuk mengimplementasikan array sebagai suatu struktur data, sehingga dapat mudah digunakan oleh kelas lain yang memerlukan.

➔ HighArray.java

```
public class HighArray{
    private long[] a;           // ref to array a
    private int nElems;         // number of data items
    //-----
    public HighArray(int max){   // constructor
        a = new long[max];      // create the array
        nElems = 0;             // no items yet
    }
    //-----
    public boolean find(long searchKey){ // find specified value
        int j;
```

Class berikut merupakan contoh class program utama yang menggunakan struktur data array di atas

```

        for(j=0; j<nElems; j++)          // for each element,
            if(a[j] == searchKey)        // found item?
                break;                  // exit loop before end
        if(j == nElems)                  // gone to end?
            return false;                // yes, can't find it
        else
            return true;                 // no, found it
    }                                    // end find()
//-----
public void insert(long value){          // put element into array
    a[nElems] = value;                 // insert it
    nElems++;                          // increment size
}
//-----
public boolean delete(long value){
    int j;
    for(j=0; j<nElems; j++)             // look for it
        if( value == a[j] )
            break;
    if(j==nElems)                       // can't find it
        return false;
    else {                             // found it
        for(int k=j; k<nElems; k++)    // move higher ones down
            a[k] = a[k+1];
        nElems--;                     // decrement size
        return true;
    }
}                                       // end delete()
//-----
public void display(){                 // displays array contents
    for(int j=0; j<nElems; j++)        // for each element,
        System.out.print(a[j] + " "); // display it
    System.out.println("");
}
//-----
}                                       // end class HighArray

```

Class berikut merupakan contoh class program utama yang menggunakan struktur data array di atas untuk menyimpan data.

➔ HighArrayApp.java

```

public class HighArrayApp{

    public static void main(String[] args){
        int maxSize = 100;                // array size
    }
}

```

```

        HighArray arr;                                // reference to array
        arr = new HighArray(maxSize);                 // create the array

        arr.insert(77);                                // insert 10 items
        arr.insert(99);
        arr.insert(44);
        arr.insert(55);
        arr.insert(22);
        arr.insert(88);
        arr.insert(11);
        arr.insert(00);
        arr.insert(66);
        arr.insert(33);

        arr.display();                                // display items

        int searchKey = 35;                            // search for item
        if( arr.find(searchKey) )
            System.out.println("Found " + searchKey);
        else
            System.out.println("Can't find " + searchKey);

        arr.delete(00);
        arr.delete(55);
        arr.delete(99);                                // delete 3 items
        arr.display();                                // display items again
    }                                                    // end main()
}                                                        // end class HighArrayApp

```

Linked List

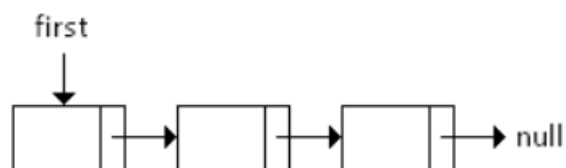
Linked list merupakan struktur data yang mirip dengan array. Salah satu perbedaan antara array dan linked list yang paling signifikan adalah: array memiliki daya tampung yang statis sementara linked list memiliki panjang yang dinamis.

Array memiliki panjang yang sudah ditentukan di awal pembuatan array. Jika data yang akan dimasukkan lebih sedikit dari daya tampung array, maka akan terjadi pemborosan memori. Sementara jika data yang akan dimasukkan lebih banyak dari daya tampung array, maka array tidak dapat menampung keseluruhan data. Jika data dipaksakan untuk masuk, maka akan terjadi error **"array index out of bound"**.

Linked list dapat dianalogikan seperti rantai, yang terdiri atas banyak mata rantai. Panjang rantai dapat disesuaikan dengan cara menambah atau mengurangi mata rantai. Seperti rantai, daya tampung Linked List dapat menyesuaikan dengan jumlah data yang akan dimasukkan. Satu-satunya hal yang membatasi daya tampung array adalah kapasitas memori yang dimiliki oleh komputer yang menjalankan program.

Sebagai sebuah struktur data, Linked List juga harus dapat menangani proses yang sering dilakukan ketika bekerja dengan sebuah struktur data. Proses-proses tersebut antara lain adalah:

1. Insert
2. Delete
3. Pencarian
4. Menampilkan isi array



Gambar 1: Ilustrasi Linked List

Contoh dari implementasi fungsi-fungsi tersebut dapat dilihat dalam contoh program di bawah ini.

Catatan: program terdiri dari file *Link.java*, *LinkedList.java*, dan *LinkedListApp.java*.

Class berikut merupakan class untuk menyimpan satu elemen data dalam Linked List.

➔ Link.java

```
public class Link{
    public int iData;           // data item (key)
    public double dData;       // data item
    public Link next;          // next link in list
    // -----
    public Link(int id, double dd){    // constructor
        iData = id;                 // initialize data
        dData = dd;
                                     // ('next' is automatically
                                     // set to null)
    }
    // -----
    public void displayLink(){        // display ourself
        System.out.print("{ " + iData + ", " + dData + " } ");
    }
}                                     // end class Link
```


Class berikut merupakan implementasi dari struktur data linked list. Class ini memanfaatkan class Link sebagai “mata rantai” untuk dijadikan “rantai” data.

➔ LinkedList.java

```
public class LinkedList{
    private Link first;                // ref to first link on list
    // -----
    public LinkedList(){                // constructor
        first = null;
        // no items on list yet
    }
    // -----
    public boolean isEmpty(){           // true if list is empty
        return (first==null);
    }
    // -----
    // insert at start of list
    public void insertFirst(int id, double dd){
        Link newLink = new Link(id, dd); // make new link
        newLink.next = first;             // newLink --> old first
        first = newLink;                 // first --> newLink
    }
    // -----
    public Link deleteFirst(){          // delete first item
        // (assumes list not empty)
        Link temp = first;               // save reference to link
        first = first.next;              // delete it: first-->old next
        return temp;                    // return deleted link
    }
    // -----
    public void displayList(){
        System.out.print("List (first-->last): ");
        Link current = first;            // start at beginning of list
        while(current != null){           // until end of list,
            current.displayLink();        // print data
            current = current.next;       // move to next link
        }
        System.out.println("");
    }
    // -----
}                                       // end class LinkedList
```

Class berikut merupakan contoh class program utama yang menggunakan struktur data LinkList di atas untuk menyimpan data

→ LinkListApp.java

```
public class LinkListApp{

    public static void main(String[] args){
        LinkList theList = new LinkList();    // make new list

        theList.insertFirst(22,2.99);          // insert four items
        theList.insertFirst(44,4.99);
        theList.insertFirst(66,6.99);
        theList.insertFirst(88,8.99);

        theList.displayList();                 // display list

        while( !theList.isEmpty() ){           // until it's empty,
            Link aLink = theList.deleteFirst(); // delete link
            System.out.print("Deleted ");      // display it
            aLink.displayLink();
            System.out.println("");
        }

        theList.displayList();                 // display list
    }                                           // end main()
}                                              // end class LinkListApp
```

Soal Jurnal :

1. Buatlah sebuah program dengan menggunakan struktur data array yang dapat menyimpan nim dan nama mahasiswa. Penginputan nim dan nama dilakukan saat program berjalan.

STACK & QUEUE

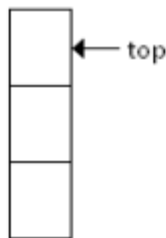
MODUL 2

Standar Kompetensi:

1. Mahasiswa memahami konsep Stack.
 2. Mahasiswa memahami konsep Queue.
 3. Mahasiswa dapat membuat dan menggunakan stack dan queue dalam kasus yang sesuai.
 4. Mahasiswa dapat melakukan operasi create, insert, dan delete data di dalam stack dan queue.
-

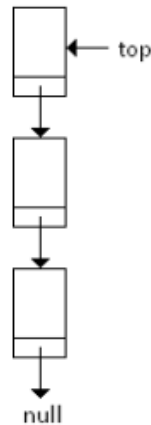
Stack

Stack merupakan suatu struktur data yang memiliki sifat **Last In First Out (LIFO)**. Seperti kata stack yang berarti tumpukan, stack dapat dianalogikan seperti suatu tumpukan. Tumpukan tersusun dari bawah ke atas, dengan setiap elemen yang baru selalu diletakkan di bagian paling atas tumpukan. Analogi dari stack dapat dilihat di gambar berikut:



Gambar 1 : Stack sebagai suatu tumpukan elemen

Untuk mengimplementasikan struktur data stack ke dalam pemrograman, kita dapat menggunakan struktur data yang pernah kita ketahui, seperti array dan linked list. Karena kita mengetahui bahwa array memiliki kekurangan yang cukup signifikan jika dibandingkan dengan linked list, maka kita akan mengimplementasikan stack menggunakan struktur data linked list.



Gambar 2 : Implementasi stack menggunakan linked list

Operasi-operasi yang dapat dilakukan pada stack terbatas pada operasi pemeriksaan apakah stack kosong, insert, dan delete. Proses penampilan data atau pengaksesan data dilakukan di luar struktur data stack.

1. Pemeriksaan status stack, kosong atau berisi

Untuk memeriksa apakah stack memiliki elemen atau tidak, cukup dengan memeriksa apakah penunjuk ke elemen teratas (top) menunjuk ke null atau tidak. Jika null, maka kembalikan status bahwa stack kosong. Jika tidak, maka kembalikan status bahwa stack tidak kosong.

2. Push (insert)

Pada stack, proses insert elemen baru dinamakan push. Proses push selalu meletakkan elemen baru di atas elemen yang paling atas (top).

3. Pop (delete)

Pada stack, proses delete elemen dinamakan pop. Proses pop selalu menghapus elemen yang terletak paling atas (top). Elemen yang dihapus oleh proses pop dikembalikan (return). Agar dapat digunakan oleh baris yang memanggil proses pop.

Contoh dari implementasi fungsi-fungsi tersebut dapat dilihat dalam contoh program di bawah ini.

Catatan: program terdiri dari file *Data.java*, *Stack.java*, dan *StackRunner.java*.

Class berikut merupakan class untuk menyimpan satu elemen data dalam stack, class ini juga dapat digunakan dalam queue.

➔ **Data.java**

```
public class Data{  
    int data;  
    Data next;  
}
```

Class berikut merupakan class untuk mengimplementasikan struktur data stack. Class ini menggunakan class Data sebagai elemen dari stack.

➔ **Stack.java**

```
public class Stack{  
    Data top;  
    void push(Data input){  
        input.next = top;  
        top = input;  
    }  
    Data pop(){  
        Data x = top;  
        top = top.next;  
        return x;  
    }  
    boolean isEmpty(){  
        return (top == null);  
    }  
}
```

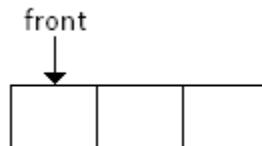
Class berikut merupakan program utama yang menggunakan struktur data stack yang telah dibuat untuk menyimpan dan menampilkan data.

➔ StackRunner.java

```
public class StackRunner {  
    public static void main(String[] args) {  
  
        Stack st = new Stack();  
        Data tmp;  
        tmp = new Data();  
        tmp.data = 12;  
        st.push(tmp);  
        tmp = new Data();  
        tmp.data = 9;  
        st.push(tmp);  
        tmp = new Data();  
        tmp.data = 11;  
        st.push(tmp);  
        tmp = new Data();  
        tmp.data = 2;  
        st.push(tmp);  
        tmp = new Data();  
        tmp.data = 7;  
        st.push(tmp);  
        while (!st.isEmpty()) {  
            tmp = st.pop();  
            System.out.println(tmp.data);  
            tmp = null;  
        }  
    }  
}
```

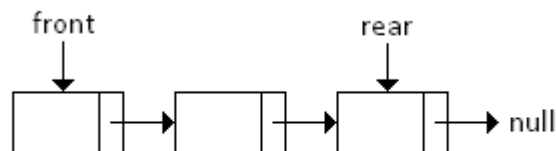
Queue

Queue merupakan struktur data yang memiliki sifat First In First Out (FIFO). Sesuai dengan kata queue yang berarti antrian, maka queue juga dapat dianalogikan dengan suatu antrian, dimana setiap elemen yang masuk lebih awal, juga akan keluar lebih awal. Pada antrian, elemen yang baru akan diletakkan di bagian paling belakang dari antrian, dan elemen yang keluar adalah elemen yang paling depan. Antrian dapat dianalogikan sebagai berikut:



Gambar 3 : Queue sebagai antrian elemen

Seperti halnya pada stack, untuk mengimplementasikan struktur data queue ke dalam pemrograman, kita dapat menggunakan struktur data yang pernah kita ketahui, seperti array dan linked list. Karena kita mengetahui bahwa array memiliki kekurangan yang cukup signifikan jika dibandingkan dengan linked list, maka kita akan mengimplementasikan queue menggunakan struktur data linked list.



Gambar 4 : Implementasi queue menggunakan linked list

Seperti terlihat di gambar 4, queue dapat dimodelkan dengan menggunakan linked list. Perbedaan yang cukup signifikan adalah, pada implementasi queue menggunakan linked list, kita juga perlu mencatat elemen yang paling belakang. Hal ini dilakukan agar setiap kali ada proses penambahan data, kita tidak perlu menelusuri list dari awal sehingga proses *insert* menjadi efisien.

Proses-proses yang terdapat di dalam queue secara umum sama dengan stack. Proses - proses tersebut adalah:

1. Pemeriksaan status queue, kosong atau berisi

Untuk memeriksa apakah queue memiliki elemen atau tidak, cukup dengan memeriksa apakah penunjuk ke elemen terdepan (front) menunjuk ke null atau tidak. Jika null, maka kembalikan status bahwa stack kosong, jika tidak maka kembalikan status bahwa stack tidak kosong.

2. Insert

Pada queue, proses insert selalu meletakkan elemen baru di atas elemen yang paling depan (front).

3. Remove

Pada queue, proses remove selalu menghapus elemen yang terletak paling depan (front). Elemen yang dikeluarkan oleh proses remove akan dikembalikan (return) agar dapat digunakan oleh baris yang memanggil proses remove.

Contoh dari implementasi fungsi-fungsi tersebut dapat dilihat dalam contoh program di bawah ini.

Catatan: program terdiri dari file *Data.java*, *Queue.java*, dan *QueueRunner.java*. Jika pengerjaan queue dilakukan dalam package yang sama dengan stack, maka tidak perlu lagi membuat file *Data.java*.

Class berikut merupakan class untuk menyimpan satu elemen data dalam queue, class ini juga dapat digunakan dalam stack.

➔ Queue.java

```
public class Queue{
    Data front, rear;
    void insert(Data input){
        if (isEmpty())
```



```

front = input;
else
rear.next = input;
rear = input;
}
Data remove(){
Data tmp = front;

front = front.next;
return tmp;
}
boolean isEmpty(){
return (front == null);
}
}

```

Class berikut merupakan program utama yang menggunakan struktur data queue yang telah dibuat untuk menyimpan dan menampilkan data.

➔ QueueRunner.java

```

public class QueueRunner {
public static void main(String[] args) {
Queue q = new Queue();
Data tmp;

tmp = new Data();
tmp.data = 12;
q.insert(tmp);
tmp = new Data();
tmp.data = 9;
q.insert(tmp);
tmp = new Data();
tmp.data = 11;
q.insert(tmp);
tmp = new Data();
tmp.data = 2;
q.insert(tmp);
tmp = new Data();
tmp.data = 7;
q.insert(tmp);
while (!q.isEmpty()){
tmp = q.remove();
System.out.println(tmp.data);
} } }

```

Soal Jurnal

1. Adi membeli 30 kotak makanan yang akan dibagikan kepada beberapa kelompok praktikan. Kotak makanan tersebut disusun tiap 5 tumpuk agar mudah dibawa. Setelah sampai di lab, kotak makanan tersebut dibagikan kepada praktikan yang sudah mengantri. Implementasikanlah cerita tersebut dengan menggunakan struktur data yang sesuai.

DOUBLE LINKED LIST

MODUL 3

Standar kompetensi:

1. Mahasiswa memahami konsep Double Linked List
 2. Mahasiswa dapat membuat dan menggunakan double linked list dalam kasus yang sesuai.
 3. Mahasiswa dapat melakukan operasi create, insert, dan delete data di dalam double linked list.
 4. Mahasiswa dapat menggunakan library struktur data yang sudah disediakan oleh Java.
-

Double Linked List

Double Linked List merupakan pengembangan dari Linked List biasa. Linked list biasa memiliki masalah dalam hal fleksibilitas arah pembacaan karena setiap elemen dalam linked list biasa hanya memiliki pointer untuk menunjukkan elemen berikutnya. Pada saat pembacaan elemen sebelumnya diperlukan, maka linked list biasa perlu melakukan pembacaan ulang dari elemen paling awal. Hal ini tentu sangat tidak efisien mengingat sebenarnya hanya diperlukan satu langkah untuk mencapai data yang diperlukan tersebut



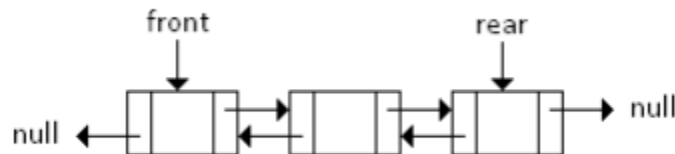
Gambar 2 : Struktur dari sebuah elemen linked list biasa

Untuk mengatasi masalah tersebut maka dilakukanlah modifikasi terhadap model linked list biasa. Dalam modifikasi tersebut kita menambahkan sebuah pointer yang menunjukkan elemen yang terletak sebelum elemen yang bersangkutan. Untuk pointer yang menunjukkan elemen berikutnya akan kita namakan "next" dan pointer yang menunjukkan elemen sebelumnya akan kita namakan "prev".



Gambar 3 : Struktur dari sebuah elemen double linked list

Karena double linked list memiliki kemampuan untuk membaca data dengan cara maju atau mundur, maka double linked list juga memiliki dua buah pointer yang menunjukkan awal dan akhir dari double linked list. Untuk pointer yang menunjukkan awal kita namakan “front”, dan untuk pointer yang menunjukkan akhir akan kita namakan “rear”. Gambar berikut merupakan visualisasi dari sebuah double linked list.



Gambar 4 : Representasi visual dari sebuah Double Linked List

Secara umum, method-method yang dimiliki oleh Double Linked List tidak berbeda dari method yang dimiliki oleh Linked List. Method-method tersebut adalah:

1. Insert: front, middle, rear
2. Remove, front, middle, rear
3. Memeriksa apakah list kosong
4. Menampilkan isi Double Linked List

Contoh dari implementasi fungsi-fungsi tersebut dapat dilihat dalam contoh program di bawah ini.

Catatan: program terdiri dari file *Data.java*, *DoubleLinkedList.java*, dan *DLLApp.java*.

Class berikut merupakan class untuk mengimplementasikan sebuah elemen dari Double Linked List.

➔ Data.java

```
public class Data{  
    int data;  
    Data next;  
    Data prev;  
}
```

Class berikut merupakan implementasi dari Double Linked List. Class ini memanfaatkan class Data yang telah diimplementasikan di atas.

➔ DoubleLinkedList.java

```
public class DoubleLinkedList{  
    Data first;  
    Data last;  
  
    void insertAwal(Data input){  
        input.next = last;  
        input.prev = first;  
  
        first = input;  
        last = input;  
    }  
}
```

```

void insertFront(Data input){
    input.next = first;
    first.prev = input;
    first = input;
}

void insertLast(Data input){
    input.prev = last;
    last.next = input;
    last = input;
}

void insertMiddle(Data input, Data current){
    current.prev.next = input;
    input.prev = current.prev;

    input.next = current;
    current.prev = input;
}

Data deleteFront(){
    Data tmp = first;
    first = first.next;
    first.prev = null;
    tmp.next = null;

    return tmp;
}

Data deleteLast(){
    Data tmp = last;
    last = last.prev;
    last.next = null;
    tmp.prev = null;

    return tmp;
}

void deleteMiddle(Data current){
    current.prev.next = current.next;
    current.next.prev = current.prev;
    current.next = null;
    current.prev = null;
}

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

void display(){
    Data current = first;
    while (current != null) {

```

```

        System.out.println(current.data);
        current = current.next;
    }
}
}

```

Class berikut merupakan class yang berfungsi untuk mensimulasikan penggunaan dari class Double Linked List yang telah dibuat di atas.

➔ DLLApp.java

```

public class DLLApp {

    public static void main(String[] args) {
        DoubleLinkedList dll = new DoubleLinkedList();

        Data tmp;

        tmp = new Data();
        tmp.data = 10;
        dll.insertAwal(tmp);

        tmp = new Data();
        tmp.data = 7;
        dll.insertFront(tmp);

        tmp = new Data();
        tmp.data = 21;
        dll.insertLast(tmp);

        dll.display();
    }
}

```

Library Java untuk struktur data linier

Beberapa bahasa pemrograman yang ada sekarang memiliki library yang dapat digunakan untuk menampung data sesuai dengan konsep struktur data yang telah kita pelajari. Library tersebut dapat mempermudah kita dalam menggunakan struktur data untuk berbagai keperluan kita. Berikut adalah contoh dari penggunaan library struktur data dalam Java:

Berikut merupakan class yang menampung data seorang mahasiswa.

➔ Mahasiswa.java

```
public class Mahasiswa{  
    String nim;  
    String nama;  
    char gender;  
    double tinggi;  
    double berat;  
}
```

Berikut merupakan class contoh penggunaan library java untuk menampung data yang akan digunakan dalam program.

➔ JavaDataStructuresLib.java

```
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Stack;

public class JavaDataStructuresLib {

    public static void main(String args[]){
        /* Deklarasi masing-masing objek
        * Bagian "<Mahasiswa>" merupakan bagian yang menyatakan bahwa
        * object dari class struktur data yang digunakan hanya menampung
        * class Mahasiswa.
        * Jika bagian "<Mahasiswa>" dihilangkan, tidak akan terjadi error
        * dan struktur data dapat menampung object dari class apapun.
        * Namun hal ini akan menambah proses yang diperlukan pada saat
        * pembacaan object, karena dibutuhkan casting sebelum digunakan.
        */
        LinkedList<Mahasiswa> ll = new LinkedList<Mahasiswa>();
        ArrayList<Mahasiswa> al = new ArrayList<Mahasiswa>();
        Stack<Mahasiswa> s = new Stack<Mahasiswa>();
        Queue<Mahasiswa> q = new LinkedList<Mahasiswa>();

        Mahasiswa tmp;

        // inisialisasi objek dan input ke dalam struktur data
        tmp = new Mahasiswa();
        tmp.nim = "120";
        tmp.nama = "John Harbour";
        tmp.gender = 'L';
        tmp.tinggi = 180.3;
        tmp.berat = 80.4;
```

```

ll.add(tmp);
al.add(tmp);
s.push(tmp);
q.add(tmp);

tmp = new Mahasiswa();
tmp.nim = "121";
tmp.nama = "Amelia Kasandra";
tmp.gender = 'P';
tmp.tinggi = 160.2;
tmp.berat = 48.7;
ll.add(tmp);
al.add(tmp);
s.push(tmp);
q.add(tmp);

tmp = new Mahasiswa();
tmp.nim = "122";
tmp.nama = "Mia Cantika";
tmp.gender = 'P';
tmp.tinggi = 168.2;
tmp.berat = 51.8;
ll.add(tmp);
al.add(tmp);
s.push(tmp);
q.add(tmp);

tmp = new Mahasiswa();
tmp.nim = "123";
tmp.nama = "Koko Kusnandar";
tmp.gender = 'L';
tmp.tinggi = 173.0;
tmp.berat = 69.3;
ll.add(tmp);
al.add(tmp);
s.push(tmp);
q.add(tmp);

// menampilkan objek di Linked List
for (int i=0; i<ll.size(); i++){
    tmp = ll.get(i);
    System.out.println("nama: "+tmp.nama);
    System.out.println("nim: "+tmp.nim);
    System.out.println("gender: "+tmp.gender);
    System.out.println("tinggi: "+tmp.tinggi);
    System.out.println("berat: "+tmp.berat+"\n");
}

// menampilkan objek di Array List
for (int i=0; i<al.size(); i++){
    tmp = al.get(i);
    System.out.println("nama: "+tmp.nama);
    System.out.println("nim: "+tmp.nim);

```

```

        System.out.println("gender: "+tmp.gender);
        System.out.println("tinggi: "+tmp.tinggi);
        System.out.println("berat: "+tmp.berat+"\n");
    }

    // menampilkan objek di Stack
    while (!s.isEmpty()){
        tmp = s.pop();
        System.out.println("nama: "+tmp.nama);
        System.out.println("nim: "+tmp.nim);
        System.out.println("gender: "+tmp.gender);
        System.out.println("tinggi: "+tmp.tinggi);
        System.out.println("berat: "+tmp.berat+"\n");
    }

    // menampilkan objek di Queue
    while (!q.isEmpty()){
        tmp = q.remove();
        System.out.println("nama: "+tmp.nama);
        System.out.println("nim: "+tmp.nim);
        System.out.println("gender: "+tmp.gender);
        System.out.println("tinggi: "+tmp.tinggi);
        System.out.println("berat: "+tmp.berat+"\n");
    }
}
}

```

Soal Jurnal

1. Buatlah program untuk mendata anggota UKM meliputi (nim, nama, tanggal masuk) yang memiliki aksi untuk menambah data, menampilkan data dari depan & belakang, serta menghapus data.

SORTING & SEARCHING

MODUL 4

Standar kompetensi:

1. Mahasiswa memahami beberapa algoritma searching.
 2. Mahasiswa memahami beberapa algoritma sorting.
 3. Mahasiswa dapat menggunakan algoritma yang tepat untuk melakukan pencarian data.
 4. Mahasiswa dapat menggunakan salah satu metode sorting untuk mengurutkan data.
-

Pencarian (Searching)

1. Linear Searching

Linear searching merupakan metode pencarian di dalam suatu struktur data linear dengan cara membaca setiap data satu persatu. Metode pencarian seperti ini tidak beda dengan proses untuk menampilkan seluruh data.

2. Binary Search

Binary search merupakan metode pencarian pada sebuah struktur data linear yang sudah terurut. Dalam metode pencarian ini, tidak semua data dibaca, namun cukup dengan membandingkan data yang ada di tengah dari suatu range. Algoritma binary search bisa dianalogikan dengan metode pencarian suatu kata di dalam kamus.

Program berikut mengimplementasikan pencarian data menggunakan Binary Search:

➔ BinarySearch.java

```
public class BinarySearch{
    public static void main(String args[]){
        // inisialisasi data yang sudah terurut
        int data[] = {0,11,22,33,44,55,66,77,88,99};
        int n = data.length;
        int cari = 44;

        // tampilkan isi array
        for (int i=0; i<n; i++){
            System.out.print(data[i] + " ");
        }
        System.out.println("\nData yang dicari: " + cari);

        // Binary search
        int awal = 0;
        int akhir = n-1;
        int tengah;
        // posisi -1 menandakan data belum ketemu
        int posisi = -1;
        while (awal < akhir){
            tengah = (int) ((awal+akhir)/2);
            if (data[tengah] == cari){
                posisi = tengah;
                break;
            } else if (data[tengah] < cari){
                awal++;
            } else {
                akhir--;
            }
        }

        // tampilkan hasil pencarian
        if (posisi == -1){
            System.out.println("data tidak ditemukan");
        } else{
            System.out.println("data    ditemukan    di    index:    "    +
posisi);
        }
    }
}
```

Pengurutan (Sorting)

1. Bubble Sort

Setiap iterasi bubble sort dilakukan dengan cara membandingkan setiap elemen data yang bersebelahan. Jika data yang dibandingkan tidak sesuai dengan sifat yang diinginkan (menaik atau menurun), maka kedua data itu akan ditukar. Program berikut mengimplementasikan pengurutan data menggunakan Bubble Sort

→ BubbleSort.java

```
public class BubbleSort{
    public static void main(String args[]){
        // inisialisasi data
        int data[] = {2,4,0,5,7,9,1,6,8,3};
        int n = data.length;

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

        // Bubble Sort
        for (int i=0; i<(n-1); i++){
            for (int j=0; j<(n-i-1); j++){
                if (data[j]>data[j+1]){
                    // tukar data ke-j dengan data ke-j+1
                    int tmp = data[j];
                    data[j] = data[j+1];
                    data[j+1] = tmp;
                }
            }
        }

        // tampilkan isi array
        for (int i=0; i<n; i++){
            System.out.print(data[i] + " ");
        }
        System.out.println();
    }
}
```

Jika program tersebut dijalankan, maka proses yang terjadi bisa digambarkan sebagai berikut:

Iterasi	Data									
	2	4	0	5	7	9	1	6	8	3
1	2	4	0	5	7	9	1	6	8	3
	2	0	4	5	7	9	1	6	8	3
	2	0	4	5	7	9	1	6	8	3
	2	0	4	5	7	9	1	6	8	3
	2	0	4	5	7	9	1	6	8	3
	2	0	4	5	7	1	9	6	8	3
	2	0	4	5	7	1	6	9	8	3
	2	0	4	5	7	1	6	8	9	3
	2	0	4	5	7	1	6	8	3	9
2	0	2	4	5	7	1	6	8	3	9
	0	2	4	5	7	1	6	8	3	9
	0	2	4	5	7	1	6	8	3	9
	0	2	4	5	7	1	6	8	3	9
	0	2	4	5	1	7	6	8	3	9
	0	2	4	5	1	6	7	8	3	9
	0	2	4	5	1	6	7	8	3	9
	0	2	4	5	1	6	7	3	8	9
3	0	2	4	5	1	6	7	3	8	9
	0	2	4	5	1	6	7	3	8	9
	0	2	4	5	1	6	7	3	8	9
	0	2	4	1	5	6	7	3	8	9
	0	2	4	1	5	6	7	3	8	9
	0	2	4	1	5	6	7	3	8	9
	0	2	4	1	5	6	3	7	8	9
4	0	2	4	1	5	6	3	7	8	9
	0	2	4	1	5	6	3	7	8	9
	0	2	1	4	5	6	3	7	8	9
	0	2	1	4	5	6	3	7	8	9
	0	2	1	4	5	3	6	7	8	9
	0	2	1	4	5	3	6	7	8	9
5	0	2	1	4	5	3	6	7	8	9
	0	1	2	4	5	3	6	7	8	9
	0	1	2	4	5	3	6	7	8	9

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

: tidak ditukar
 : ditukar
 : tidak diperiksa karena sudah terurut

2. Selection Sort

Program berikut mengimplementasikan pengurutan data menggunakan Selection Sort

➔ SelectionSort.java

```

public class SelectionSort{
    public static void main(String args[]){
        // inisialisasi data
        int data[] = {2,4,0,5,7,9,1,6,8,3};
        int n = data.length;

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

        // Selection Sort
        int indexMin;
        for (int i=0; i<n; i++){
            // cari minimum dari data
            indexMin = i;
            for (int j=i; j<n; j++){
                if (data[j]<data[indexMin]){
                    indexMin = j;
                }
            }
            // tukar data ke-i dengan data ke-indexMin
            int tmp = data[i];
            data[i] = data[indexMin];
            data[indexMin] = tmp;
        }

        // tampilkan isi array
        for (int i=0; i<n; i++){

```



```



        System.out.print(data[i] + " ");
    }
    System.out.println();
}
}

```

Hasil tracing Selection Sort

Iterasi	Cari Min	Data										Index Min
		2	4	0	5	7	9	1	6	8	3	
0	0	2	4	0	5	7	9	1	6	8	3	0
	1	2	4	0	5	7	9	1	6	8	3	0
	2	2	4	0	5	7	9	1	6	8	3	2
	3	2	4	0	5	7	9	1	6	8	3	2
	4	2	4	0	5	7	9	1	6	8	3	2
	5	2	4	0	5	7	9	1	6	8	3	2
	6	2	4	0	5	7	9	1	6	8	3	2
	7	2	4	0	5	7	9	1	6	8	3	2
	8	2	4	0	5	7	9	1	6	8	3	2
	9	2	4	0	5	7	9	1	6	8	3	2
	tukar	0	4	2	5	7	9	1	6	8	3	
1	1	0	4	2	5	7	9	1	6	8	3	1
	2	0	4	2	5	7	9	1	6	8	3	2
	3	0	4	2	5	7	9	1	6	8	3	2
	4	0	4	2	5	7	9	1	6	8	3	2
	5	0	4	2	5	7	9	1	6	8	3	2
	6	0	4	2	5	7	9	1	6	8	3	6
	7	0	4	2	5	7	9	1	6	8	3	6
	8	0	4	2	5	7	9	1	6	8	3	6
	9	0	4	2	5	7	9	1	6	8	3	6
	tukar	0	1	2	5	7	9	4	6	8	3	
2	2	0	1	2	5	7	9	4	6	8	3	2
	3	0	1	2	5	7	9	4	6	8	3	2
	4	0	1	2	5	7	9	4	6	8	3	2
	5	0	1	2	5	7	9	4	6	8	3	2
	6	0	1	2	5	7	9	4	6	8	3	2
	7	0	1	2	5	7	9	4	6	8	3	2
	8	0	1	2	5	7	9	4	6	8	3	2
	9	0	1	2	5	7	9	4	6	8	3	2
	tukar	0	1	2	5	7	9	4	6	8	3	
3	3	0	1	2	5	7	9	4	6	8	3	3
	4	0	1	2	5	7	9	4	6	8	3	3
	5	0	1	2	5	7	9	4	6	8	3	3
	6	0	1	2	5	7	9	4	6	8	3	6
	7	0	1	2	5	7	9	4	6	8	3	6
	8	0	1	2	5	7	9	4	6	8	3	6
	9	0	1	2	5	7	9	4	6	8	3	9
	tukar	0	1	2	3	7	9	4	6	8	5	
4	4	0	1	2	3	7	9	4	6	8	5	4
	5	0	1	2	3	7	9	4	6	8	5	4
	6	0	1	2	3	7	9	4	6	8	5	6
	7	0	1	2	3	7	9	4	6	8	5	6
	8	0	1	2	3	7	9	4	6	8	5	6
	9	0	1	2	3	7	9	4	6	8	5	6

	tukar	0	1	2	3	4	9	7	6	8	5	
5	5	0	1	2	3	4	9	7	6	8	5	5
	6	0	1	2	3	4	9	7	6	8	5	6
	7	0	1	2	3	4	9	7	6	8	5	7
	8	0	1	2	3	4	9	7	6	8	5	7
	9	0	1	2	3	4	9	7	6	8	5	9
	tukar	0	1	2	3	4	5	7	6	8	9	
6	6	0	1	2	3	4	5	7	6	8	9	6
	7	0	1	2	3	4	5	7	6	8	9	7
	8	0	1	2	3	4	5	7	6	8	9	7
	9	0	1	2	3	4	5	7	6	8	9	7
	tukar	0	1	2	3	4	5	6	7	8	9	
7	7	0	1	2	3	4	5	6	7	8	9	7
	8	0	1	2	3	4	5	6	7	8	9	7
	9	0	1	2	3	4	5	6	7	8	9	7
	tukar	0	1	2	3	4	5	6	7	8	9	
8	8	0	1	2	3	4	5	6	7	8	9	8
	9	0	1	2	3	4	5	6	7	8	9	8
	tukar	0	1	2	3	4	5	6	7	8	9	

 : minimum
 : tidak diperiksa karena sudah terurut

3. Insertion Sort

Program berikut mengimplementasikan pengurutan data menggunakan Insertion Sort

➔ InsertionSort.java



```
public class InsertionSort{
    public static void main(String args[]){
        // inisialisasi data
        int data[] = {2,4,0,5,7,9,1,6,8,3};
        int n = data.length;

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

        // Insertion Sort
        int tmp;
        for (int i=1; i<n; i++){
            // masukkan data yang akan diinsert ke dalam tmp
            tmp = data[i];
            // cari posisi paling sesuai dg data[i]
            // posisi diinisialisasi dg i, yang merupakan posisi
            // default dari data
            int posisi = i;
            for (int j=0; j<i; j++){
                if (data[j]>tmp){
                    // catat posisi j
                    posisi = j;
                    // geser data, untuk mengosongkan posisi j
                    for (int k=i; k>posisi; k--){
                        data[k] = data[k-1];
                    }
                    // keluar dari loop pencarian posisi
                    break;
                }
            }
            // masukkan data pada posisi yg tepat
            data[posisi] = tmp;
        }

        // tampilkan isi array
        for (int i=0; i<n; i++){
            System.out.print(data[i] + " ");
        }
        System.out.println();
    }
}
```

Iterasi	Aksi	Data										Tmp
		2	4	0	5	7	9	1	6	8	3	
1		2		0	5	7	9	1	6	8	3	4
	Geser	2		0	5	7	9	1	6	8	3	4
	Insert	2	4	0	5	7	9	1	6	8	3	
2		2	4		5	7	9	1	6	8	3	0
	Geser		2	4	5	7	9	1	6	8	3	0
	Insert	0	2	4	5	7	9	1	6	8	3	
3		0	2	4		7	9	1	6	8	3	5
	Geser	0	2	4		7	9	1	6	8	3	5
	Insert	0	2	4	5	7	9	1	6	8	3	
4		0	2	4	5		9	1	6	8	3	7
	Geser	0	2	4	5		9	1	6	8	3	7
	Insert	0	2	4	5	7	9	1	6	8	3	
5		0	2	4	5	7		1	6	8	3	9
	Geser	0	2	4	5	7		1	6	8	3	9
	Insert	0	2	4	5	7	9	1	6	8	3	
6		0	2	4	5	7	9		6	8	3	1
	Geser	0		2	4	5	7	9	6	8	3	1
	Insert	0	1	2	4	5	7	9	6	8	3	
7		0	1	2	4	5	7	9		8	3	6
	Geser	0	1	2	4	5		7	9	8	3	6
	Insert	0	1	2	4	5	6	7	9	8	3	
8		0	1	2	4	5	6	7	9		3	8
	Geser	0	1	2	4	5	6	7		9	3	8
	Insert	0	1	2	4	5	6	7	8	9	3	
9		0	1	2	4	5	6	7	8	9		3
	Geser	0	1	2		4	5	6	7	8	9	3
	Insert	0	1	2	3	4	5	6	7	8	9	

 : posisi yang akan diinsert
 : data yang sudah terurut

Soal Jurnal

1. Buatlah proses pengurutan data secara ascending dengan metode Bubble Sort dengan angka 5 1 4 8 2 3

TREE

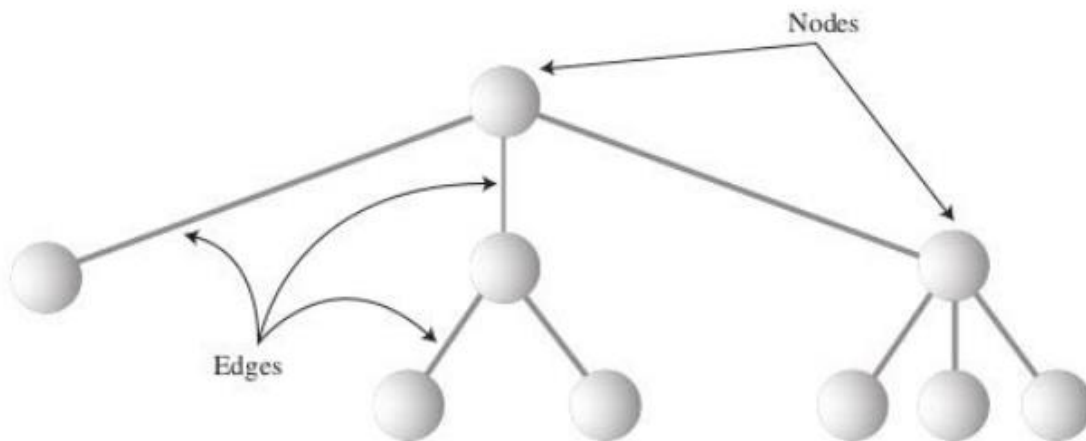
MODUL 5

Standar kompetensi:

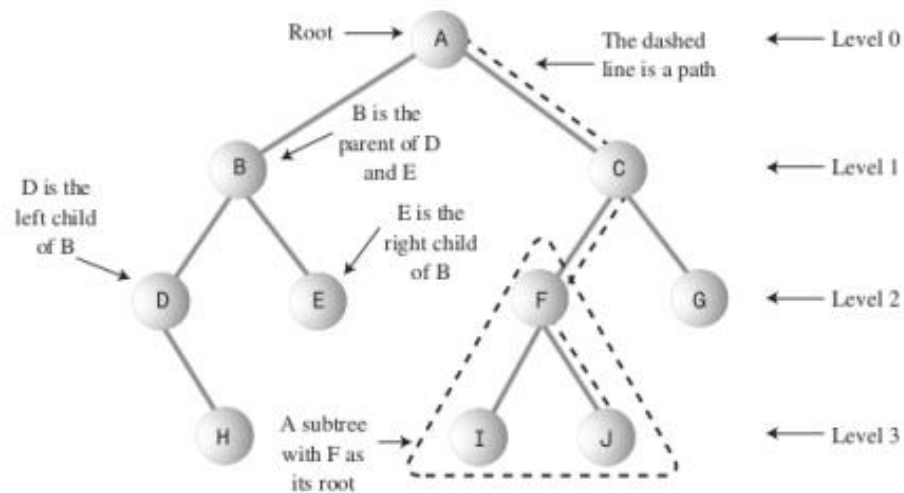
1. Mahasiswa memahami konsep tree.
 2. Mahasiswa memahami konsep binary search tree.
 3. Mahasiswa dapat menerapkan binary tree untuk menyelesaikan suatu kasus.
-

Tree

Tree merupakan sekumpulan node yang terhubung dengan edges. Dalam pemrograman, node mewakili data yang disimpan, dan edges mewakili hubungan yang dimiliki antar data



Dalam tree, dikenal istilah-istilah sebagai berikut:



- Path: jalur yang ditempuh dari suatu node menuju node lain.
- Root: merupakan node yang paling atas. Setiap tree hanya memiliki satu root.
- Parent: setiap node (kecuali root) hanya dapat memiliki satu edges ke node “atas” -nya.
- Node “atas” inilah yang disebut parent.
- Child: setiap node dapat memiliki node yang terhubung di “bawah”-nya. Node “bawah” inilah yang disebut child.
- Leaf: setiap node yang tidak memiliki child.
- Sub-tree: merupakan tree yang menjadi bagian dari tree lain.
- Visiting: merupakan proses untuk mengunjungi suatu node, dengan tujuan untuk melakukan suatu proses terhadap data yang dimiliki oleh node tersebut.
- Traversing: proses untuk mengunjungi (visiting) setiap node yang dimiliki oleh suatu tree.
- Levels: merupakan jarak suatu node terhadap root.
- Keys: merupakan data yang biasa digunakan dalam proses pencarian data yang dimiliki oleh tree.

Binary Tree merupakan bentuk khusus dari tree dimana setiap node hanya dapat memiliki maksimum dua buah node child. Sedangkan **Binary Search Tree** merupakan bentuk khusus dari Binary Tree, dimana setiap node child kiri selalu memiliki nilai key yang lebih kecil dari parent-nya, dan setiap node child kanan selalu memiliki nilai key yang lebih besar dari parent-nya.

Proses-proses yang dimiliki suatu Binary Search Tree antara lain adalah:

1. Insert

Proses insert dilakukan dengan mencari posisi yang paling tepat dari sebuah node baru. Proses pencarian ini dilakukan dengan tetap mengikuti sifat yang dimiliki oleh Binary Search Tree, yaitu node child kiri lebih kecil dari node parent, dan node child kanan lebih besar dari node parent. Node baru diletakkan ketika menemukan posisi node child yang sesuai, dan posisi tersebut kosong (null).

2. Search

Dengan mengikuti sifat dari Binary Search Tree, kita dapat melakukan pencarian terhadap key yang dicari. Ketika kita mengunjungi sebuah node, maka kemungkinan yang terjadi adalah:

- Node tersebut merupakan node yang dicari
- Key yang kita cari lebih kecil dari key node yang sedang dikunjungi
- Kita harus mengunjungi node child sebelah kiri.
- Key yang kita cari lebih besar dari key node yang sedang dikunjungi
- Kita harus mengunjungi node child sebelah kanan.

Ketika kita menemukan node yang bernilai null, maka kita dapat memutuskan bahwa key yang kita cari tidak terdapat di dalam binary tree yang kita periksa.

3. Traverse

Dalam melakukan travers terhadap suatu binary tree, kita dapat memilih satu di antara tiga cara yang ada:

a. Inorder

Inorder dilakukan dengan langkah-langkah berikut:

- 1) Kunjungi child kiri
- 2) Proses current node
- 3) Kunjungi child kanan

b. Preorder

Inorder dilakukan dengan langkah-langkah berikut:

- 1) Proses current node
- 2) Kunjungi child kiri
- 3) Kunjungi child kanan

c. Postorder

Inorder dilakukan dengan langkah-langkah berikut:

- 1) Kunjungi child kiri
- 2) Kunjungi child kanan
- 3) Proses current node

Contoh dari implementasi fungsi-fungsi tersebut dapat dilihat dalam contoh program di bawah ini.

Catatan: program terdiri dari file *Node.java*, *BinarySearchTree.java*, dan *BSTApp.java*.

Class berikut merupakan class untuk mengimplementasikan sebuah node dari Binary Search Tree

➔ **Node.java**

```
public class Node {  
    int data;  
    Node right;  
    Node left;  
}
```


Class berikut merupakan class untuk mengimplementasikan Binary Search Tree.

```
public class BinarySearchTree {

    Node root;

    public void insert(Node input){
        if (isEmpty()){
            root = input;
        } else {
            // cari parent yg sesuai dan (kiri/kanan)
            Node current = root;
            Node parent = null;
            boolean diKiri = true;
            while (current != null){
                parent = current;
                // kalau data yang akan diinputkan lebih besar,
                // bergerak ke kanan
                if (current.data < input.data){
                    current = current.right;
                    diKiri = false;
                } // else gerak ke kiri
                else {
                    current = current.left;
                    diKiri = true;
                }
            }
            // hubungkan ke parent
            if (diKiri)
                parent.left = input;
            else
                parent.right = input;
        }
    }

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

    public void traverseInorder(){
        inorder(root);
    }
}
```

```

private void inorder(Node akar){
    if (akar != null){
        inorder(akar.left);
        System.out.print(akar.data + " ");
        inorder(akar.right);
    }
}

public void traversePreorder(){
    preorder(root);
}

private void preorder(Node akar){
    if (akar != null){
        System.out.print(akar.data + " ");
        preorder(akar.left);
        preorder(akar.right);
    }
}

public void traversePostorder(){
    postorder(root);
}

private void postorder(Node akar){
    if (akar != null){
        postorder(akar.left);
        postorder(akar.right);
        System.out.print(akar.data + " ");
    }
}

public Node search(int key){
    Node node = null;
    Node current = root;

    // lakukan pencarian selama current bukan null
    while (current != null){
        if (current.data == key){
            return node;
        } else {
            if (current.data < key){
                current = current.right;
            } else {
                current = current.left;
            }
        }
    }

    return node;
}
}

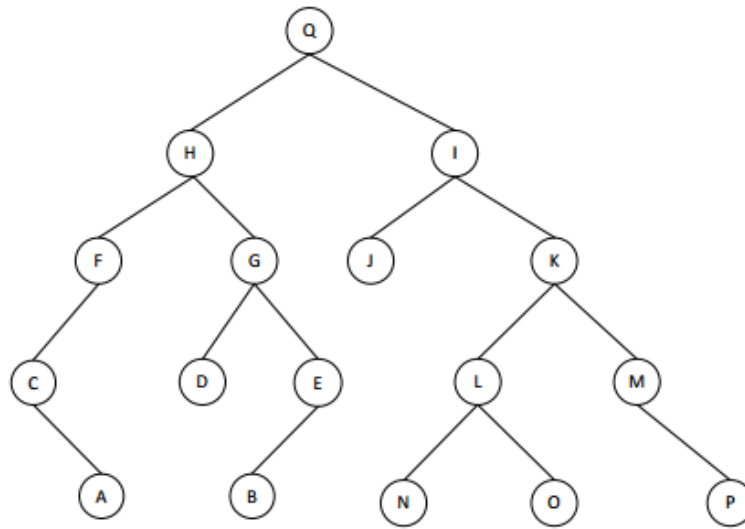
```

Class berikut merupakan class untuk menunjukkan penggunaan Binary Search Tree.

➔ BSTApp.java

```
public class BSTApp {  
  
    public static void main(String[] args) {  
        BinarySearchTree bst = new BinarySearchTree();  
  
        Node tmp;  
  
        tmp = new Node();  
        tmp.data = 7;  
        bst.insert(tmp);  
  
        tmp = new Node();  
        tmp.data = 5;  
        bst.insert(tmp);  
  
        tmp = new Node();  
        tmp.data = 6;  
        bst.insert(tmp);  
  
        tmp = new Node();  
        tmp.data = 4;  
        bst.insert(tmp);  
  
        tmp = new Node();  
        tmp.data = 9;  
        bst.insert(tmp);  
  
        bst.traverseInorder();  
        System.out.println();  
        bst.traversePreorder();  
        System.out.println();  
        bst.traversePostorder();  
    }  
}
```

Soal Jurnal



1. Dari tree di atas, tentukan traverse inorder, preorder, dan postorder, serta penjelasan dari masing-masingnya.

GRAPH

MODUL 6

Standar kompetensi:

1. Mahasiswa memahami konsep tree.
 2. Mahasiswa memahami konsep binary search tree.
 3. Mahasiswa dapat menerapkan binary tree untuk menyelesaikan suatu kasus.
-

Graph

Graph merupakan struktur data yang menyerupai Tree. Jika kita memandang tree dan graph secara matematis, maka kita akan menemukan bahwa tree merupakan bentuk khusus dari graph. Namun karena perbedaan metode implementasi dari graph dan tree, maka kedua kasus ini dipisahkan. Implementasi tree dalam pemrograman lebih menyerupai implementasi linked list, stack, dan queue. Hal terlihat dari penggunaan pointer untuk membentuk struktur dari data yang ada. Implementasi graph dalam pemrograman tidak menggunakan pointer untuk membentuk struktur graph. Graph direpresentasikan menggunakan elemen-elemen berikut:

1. Daftar node

Node yang ada di dalam graph dikumpulkan dalam satu daftar yang memiliki index, agar masing-masing node dapat mudah diakses. Daftar ini dapat diimplementasikan menggunakan array, atau linked list. Contoh:

```
class Node{
    ...
}

class Graph{
    Node daftarNode[];
}
```

2. Kumpulan edge

Setiap edge menggambarkan hubungan antara dua node yang dihubungkan oleh edge tersebut. Dua buah node yang terhubung oleh sebuah edge merupakan node-node yang bertetangga. Ketetanggaan antar node digambarkan di dalam sebuah matriks, yang disebut matriks ketetanggaan (adjacency matrix). Format isi dari matriks ketetanggaan tergantung dari jenis graph yang digambarkan. Secara umum, bentuk dari matriks ketetanggaan adalah sebagai berikut:

		tujuan			
		0	1	...	n
asal	0				
	1				
	...				
	n				

Matriks ketetanggaan menampung bobot hubungan antar node. Bobot ini bisa berupa bilangan, baik bilangan bulat (integer, long) atau bilangan real (float, double). Masing masing index berkorespondensi dengan daftarNode yang telah dibuat, sehingga panjang sisi matriks ketetanggaan sama dengan jumlah node. Contoh deklarasi matriks ketetanggaan:

```
class Graph{
    int jumlahNode = 10;
    Node daftarNode[] = new Node[jumlahNode];
    int matriksKetetanggaan[][] = new int[jumlahNode][jumlahNode];
}
```

Jenis-jenis graph:

1. Berdasarkan arah edge

a. Graph berarah

Matriks ketetanggaan hanya menggambarkan hubungan dari node asal ke node tujuan. Arah sebaliknya (tujuan - asal) belum tentu memiliki nilai yang sama, atau bahkan bisa jadi tidak ada.

b. Graph tidak berarah

Matriks ketetanggaan menggambarkan bahwa edge “asal - tujuan” memiliki nilai yang sama dengan edge “tujuan - arah”.

2. Berdasarkan bobot

a. Graph berbobot

Dalam graph, setiap edge memiliki bobot yang menjadi ukuran dari hubungan antar edge. Contoh kita membuat graph dimana setiap node merupakan kota, dan setiap edge merupakan hubungan antar kota. Bobot untuk masing-masing edge dalam graph ini bisa berupa nilai jarak, waktu tempuh, jumlah kerjasama, dan lain sebagainya.

b. Graph tidak berbobot

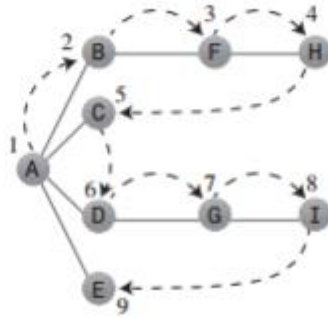
Dalam graph tidak berbobot, matriks ketetanggaan hanya menggambarkan ada atau tidaknya edge (hubungan) antar node. Oleh sebab itu nilai yang disimpan hanyalah nilai biner (0 atau 1).

Fungsi-fungsi yang ada dalam graph antara lain adalah:

1. Insert node: memasukkan sebuah node ke dalam daftar node.
2. Insert edge: memasukkan nilai sebuah edge ke dalam matriks ketetanggaan.
3. Pencarian: apakah sebuah node dapat dikunjungi dari suatu node. Metode pencarian dalam graph dapat berupa salah satu di antara metode berikut:

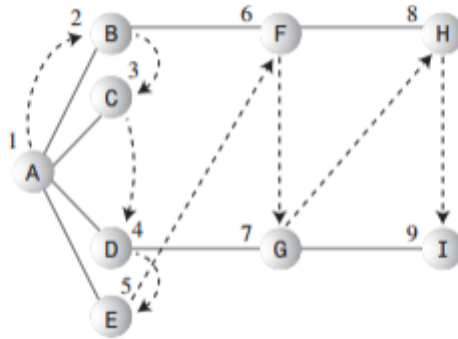
a. Depth First Search (DFS)

Dalam DFS, pencarian berfokus pada salah satu jalur terlebih dahulu, sampai node yang dicari ditemukan atau jalur tersebut tidak dapat ditelusuri lagi. Logika dalam DFS dapat digambarkan sebagai berikut :



b. Breadth First Search (BFS)

Dalam BFS, pencarian dilakukan dengan cara mencari di node yang bertetangga dengan node yang sudah dikunjungi. Pencarian selesai ketika node yang dicari ditemukan, atau tidak ada lagi node yang dapat dikunjungi. Logika dalam BFS dapat digambarkan sebagai berikut :



4. Menampilkan seluruh node yang terhubung dengan suatu node.

Metode BFS dan DFS yang digunakan untuk melakukan pencarian dapat juga digunakan untuk menampilkan semua node yang terhubung (langsung atau tidak langsung) dengan suatu node tertentu. Contoh dari implementasi fungsi-fungsi tersebut dapat dilihat dalam contoh program di bawah ini.

Catatan: program terdiri dari file *Node.java*, *Graph.java*, dan *GraphApp.java*.

Class berikut merupakan class untuk mengimplementasikan sebuah node dalam graph.

➔ Node.java

```
public class Node {  
    String data;  
    boolean sudahDikunjungi;  
  
    public Node(String data){  
        this.data = data;  
        this.sudahDikunjungi = false;  
    }  
}
```

Class berikut merupakan class untuk mengimplementasikan graph.

➔ Graph.java

```
import java.util.PriorityQueue;
import java.util.Stack;

public class Graph {
    Node daftarNode[];
    int matriksKetetanggaan[][];

    int jumlahNode;
    final int jumlahNodeMaksimum = 10;

    public Graph(){
        daftarNode = new Node[jumlahNodeMaksimum];
        matriksKetetanggaan = new int[jumlahNodeMaksimum][jumlahNodeMaksimum];
        jumlahNode = 0;
    }

    public void insertNode(String data){
        Node tmp = new Node(data);
        daftarNode[jumlahNode] = tmp;
        jumlahNode++;
    }

    // CATATAN!
    // metode insert cukup dibuat salah satu saja,
    // sesuai jenis graph yang dibuat

    public void insertEdge(int asal, int tujuan){
        matriksKetetanggaan[asal][tujuan] = 1;
        matriksKetetanggaan[tujuan][asal] = 1;
    }

    public void insertEdgeDirected(int asal, int tujuan){
        matriksKetetanggaan[asal][tujuan] = 1;
    }

    public void insertEdgeWeighted(int asal, int tujuan, int bobot){
        matriksKetetanggaan[asal][tujuan] = bobot;
        matriksKetetanggaan[tujuan][asal] = bobot;
    }
}
```

```

    }

    public void insertEdgeWeightedDirected(int asal, int tujuan, int bobot){
        matriksKetetanggaan[asal][tujuan] = bobot;
    }

    private void resetKunjungan(){
        for (int i=0; i<jumlahNode; i++){
            daftarNode[i].sudahDikunjungi = false;
        }
    }

    public int getAdjacentUnvisitedNode(int v){
        for(int j=0; j<jumlahNode; j++){
            if(matriksKetetanggaan[v][j] == 1 &&
daftarNode[j].sudahDikunjungi==false)
                return j;
        }
        return -1;
    }

    public void displayDFS(int awal){
        Stack<Integer> stack = new Stack<Integer>();

        daftarNode[awal].sudahDikunjungi = true;
        System.out.println(daftarNode[awal].data);
        stack.push(awal);

        // lakukan selama ada index node di dalam stack
        while( !stack.isEmpty() ) {
            int indexTetangga = getAdjacentUnvisitedNode( stack.peek() );
            if(indexTetangga == -1)
                stack.pop();
            else {
                daftarNode[indexTetangga].sudahDikunjungi = true;
                System.out.println(daftarNode[indexTetangga].data);
                stack.push(indexTetangga);
            }
        }
        resetKunjungan();
    }

    public void displayBFS(int awal){
        PriorityQueue<Integer> queue = new PriorityQueue<Integer>();

        daftarNode[awal].sudahDikunjungi = true;
        queue.add(awal);

        while (!queue.isEmpty()) {
            // ambil data dari queue
            int currentIndex = queue.remove();
            System.out.println(daftarNode[currentIndex].data);

```

```

        // ambil semua tetangga, masukkan ke dalam queue
        int indexTetangga =
this.getAdjacentUnvisitedNode(currentIndex);
        while (indexTetangga >= 0) {
            daftarNode[indexTetangga].sudahDikunjungi = true;
            queue.add(indexTetangga);
            indexTetangga =
this.getAdjacentUnvisitedNode(currentIndex);
        }

        resetKunjungan();
    }

    public int DFS(String data, int awal){
        Stack<Integer> stack = new Stack<Integer>();

        daftarNode[awal].sudahDikunjungi = true;
        if (daftarNode[awal].data.equals(data)){
            resetKunjungan();
            return awal;
        }
        stack.push(awal);

        // lakukan selama ada index node di dalam stack
        while( !stack.isEmpty() ) {
            int indexTetangga = getAdjacentUnvisitedNode( stack.peek() );
            if(indexTetangga == -1)
                stack.pop();
            else {
                daftarNode[indexTetangga].sudahDikunjungi = true;
                if (daftarNode[indexTetangga].data.equals(data)){
                    resetKunjungan();
                    return indexTetangga;
                }
                stack.push(indexTetangga);
            }
        }

        resetKunjungan();

        return -1;
    }

    public int BFS(String data, int awal){
        PriorityQueue<Integer> queue = new PriorityQueue<Integer>();

        daftarNode[awal].sudahDikunjungi = true;
        queue.add(awal);

        while (!queue.isEmpty()) {
            // ambil data dari queue, dan periksa apakah data yang dicari

```

```

        int currentIndex = queue.remove();
        if (daftarNode[currentIndex].data.equals(data)) {
            resetKunjungan();
            return currentIndex;
        }

        // ambil semua tetangga, masukkan ke dalam queue
        int indexTetangga =
this.getAdjacentUnvisitedNode(currentIndex);
        while (indexTetangga >= 0) {
            daftarNode[indexTetangga].sudahDikunjungi = true;
            queue.add(indexTetangga);
            indexTetangga =
this.getAdjacentUnvisitedNode(currentIndex);
        }
    }

    return -1; }
}

```

Class berikut merupakan class contoh penggunaan graph.

➔ GraphApp.java

```

public class GraphApp {

    public static void main (String a[]){
        Graph g = new Graph();

        g.insertNode("Jakarta"); // 0
        g.insertNode("Bandung"); // 1
        g.insertNode("Sukabumi"); // 2
        g.insertNode("Cianjur"); // 3
        g.insertNode("Bogor"); // 4

        g.insertEdge(0,2);
        g.insertEdge(0,4);
        g.insertEdge(0,1);
        g.insertEdge(4,3);
        g.insertEdge(4,1);
        g.insertEdge(3,2);
        g.insertEdge(3,1);

        g.displayDFS(0);
        System.out.println("---");
        g.displayBFS(0);
        System.out.println("---");
        String dataDicari = "Surabaya";
        System.out.println("Hasil DFS: " + g.DFS(dataDicari,0));
        System.out.println("---");
        System.out.println("Hasil BFS: " + g.BFS(dataDicari,0));})

```

Tabel jarak kota asal dan kota tujuan (dalam Km)

Jakarta	Cianjur	114
Bandung	Ciamis	115
Tasik	Banjar	42
Nagrek	Tasik	66
Palimanan	Kanci	27
Bandung	Sumedang	13
Sumedang	Palimanan	71
Bandung	Nagrek	32
Cianjur	Bandung	59
Ciamis	Wangon	116
Jakarta	Cikampek (via tol)	73
Jakarta	Bandung (via tol)	146
Tasik	Ciamis	17

1. Gambarkan relasi antar kota di atas.