MODUL PRAKTIKUM
ALGORITMA DAN PEMROGRAMAN II

Tim Penyusun :

Dosen dan Asisten Praktikum Algoritma Pemrograman II

JURUSAN TEKNIK INFORMATIKA

FAKULTAS TEKNIK

UNIVERSITAS PALANGKA RAYA

2021

TATA TERTIB DAN TATA LAKSANA PRAKTIKUM
JURUSAN/PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNIK UPR

TATA TERTIB

1. Praktikan **WAJIB** mengikuti semua modul praktikum.
2. Praktikan hanya boleh tidak mengikuti praktikum **1 (satu) kali DENGAN ATAU TANPA surat izin** dari jumlah pertemuan praktikum.
3. Praktikan yang berhalangan mengikuti praktikum, diwajibkan melaporkan ke asisten praktikum untuk menentukan jadwal praktikum sebagai pengganti jadwal yang berhalangan.
4. Praktikan yang lebih dari 1 (satu) kali tidak mengikuti praktikum, tidak diperbolehkan untuk mengikuti praktikum untuk modul-modul praktikum selanjutnya dan **NILAI AKHIR PRAKTIKUM** adalah **NOL**.
5. Praktikan wajib hadir 15 menit sebelum praktikum dimulai.
6. Tidak diperbolehkan saling bekerjasama.
7. Dilarang menggunakan kaos oblong dan sandal jepit selama praktikum. Bagi yang melanggarpoin ini, tidak diperbolehkan mengikuti praktikum.

TATA LAKSANA

1. Sebelum praktikum dimulai, setiap praktikan wajib mengumpulkan **LAPORAN RENCANA PRAKTIKUM (Tulis Tangan)** dan **LAPORAN HASIL PRAKTIKUM** modul sebelumnya.
2. Jika praktikan tidak melaksanakan tata laksana poin 1, maka **TIDAK DIPERBOLEHKAN MENGIKUTI PRAKTIKUM**.
3. Setiap modul praktikum, akan dilakukan ***Pre-test***.
4. Pada saat praktikum hanya membawa alat tulis, modul dan Laporan. HP, Laptop & USB di simpan dan tas letakan didepan.
5. Pada saat praktikum tidak diijinkan mengakses Internet.
6. Format laporan meliputi:

   Laporan Rencana Praktikum :
   - Format A4
   - Tujuan Praktikum dari modul yang akan dilaksanakan
   - Hal-hal yang akan dilakukan selama praktikum.

   Laporan Hasil Praktikum :
   - Halaman Depan
   - Bab I. Tujuan dan Landasan Teori
   - Bab II. Pembahasan
   - Bab III. Kesimpulan
   - Daftar Pustaka
   - Lampiran (disertai laporan rencana praktikum)

7. Format Penulisan Laporan Hasil Praktikum
   - Spasi : 1,5
   - Font : Times New Roman
   - Font Size : 12
   - Margins : Top 3 cm, Left 4 cm, Right 3 cm, Bottom 4 cm
   - Kertas : A4

8. Penilaian Laporan Hasil Praktikum :
   - Bab I. Tujuan dan LandasanTeori Nilai 10
   - Bab II. Pembahasan Nilai 60
   - Bab III. Kesimpulan Nilai 20
   - Daftar Pustaka Nilai 5
   - Lampiran Nilai 5

   Total **100**

9. Praktikan yang mengabaikan format penulisan poin 5, nilai akan dikurangi 5 setiap kesalahan.

10. PenilaianAkhirPraktikum :
    - *Pre-Test* : 10%
    - Praktikum : 50%
    - Laporan Praktikum : 15%
    - Responsi : 25%

    Total **100%**

11. PenilaianAkhir Mata Kuliah:
    - Kuliah, meliputi:

      Tugas : 20%
      UTS : 30%
      Praktikum : 50%
      Total **: 50%**
    - UAS : 50%

      Total **100%**

MODUL I

PENGENALAN JAVA

I. Tujuan Praktikum

1. Mahasiswa dapat mengenal apa itu Java dan struktur awal beserta aturan penulisan sintaks pada Java
2. Mahasiswa dapat mengenal variable, tipe data, operasi dasar dan operasi perhitungan pada Java.
3. Mahasiswa memahami percabangan dan pengulangan pada bahasa pemrograman Java
4. Mahasiswa dapat mengimplementasikan percabangan dan pengulangan pada Java.

II. Landasan Teori

1. Deklarasi Package

Package merupakan sebuah folder yang berisi sekumpulan program Java. Deklarasi package biasanya dilakukan saat membuat program atau aplikasi besar. Contoh deklarasi package:

```
Package Belajar ;
```

Biasanya nama package mengikuti nama program yang sudah kita buat ataupun bisa kita buat sesuai dengan kebutuhan kita. *Belajar* Disini adalah Nama Program nya.

2. *Class*

Java merupakan bahasa pemrograman yang menggunakan paradigma OOP *(Object Oriented Programming)*. Setiap program harus dibungkus di dalam class agar nanti bisa dibuat menjadi objek.

Contoh Deklarasi Class:

```
class Belajar {
   public static void main(String[] args)  {
      System.out.println("Hello World");
   }
}
```

3. *Method Main*

Method main( ) atau fungsi main( ) merupakan blok program yang akan dieksekusi pertama kali. Ini adalah inti dari program. Method main( ) wajib kita buat. Kalau tidak, maka programnya tidak akan bisa dieksekusi atau dijalankan.

Contoh method main( ) :

```
public static void main(String[] args)  {
   System.out.println("Hello World");
}
```

4. *Variabel dan Tipe data*

Berikut ini macam-macam tipe data pada Java:

- ***char***: Tipe data karakter
- ***int***: angka atau bilangan bulat
- ***float***: bilangan decimal
- ***double***: bilangan desimal juga, tapi lebih besar kapasistanya
- ***String***: kumpulan dari karakter yang membentuk teks
- ***boolean***: tipe data yang hanya bernilai true dan false

Membuat Variabel :
Formatnya seperti ini:

```
<tipe data> namaVariabel;
```

Contoh:

Membuat variabel kosong bertipe integer:

```
int namaVariabel;
```

Membuat variabel bertipe integer dan langsung diisi nilai:

```
int namaVariabel = 200;
```

Membuat sekumpulan variabel yang tipe datanya sama:

```
int x, y, z;
```

5. *Operator Dasar dan Operator Perhitungan*

- ***Operator Aritmatika***

Operator aritmatika ini adalah operator yang sering kita gunakan untuk hitung menghitung seperti kali bagi tambah kurang dam lainnya . Untuk Jelasnya, bisa dilihat dari tabel di bawah ini :

Tabel 1.1. Operator Aritmatika

Operator	Keterangan
+	penjumlahan
–	Pengurangan
*	Perkalian
/	Pembagian
%	Sisa Pembagian

- ***Operator Penugasan/Assignment Operator***

Operator Penugasan atau Assignment Operator digunakan untuk memberi tugas suatu variable untuk melakukan suatu proses. Operator ini sering digunakan dalam pemrograman untuk mengulangi suatu perintah, salah satunya adalah increment/decrement.

Tabel 1.2. Operator Penugasan

Operator	Keterangan
=	Pemberian Nilai
+=	Penambahan Bilangan
-=	Pengurangan Bilangan
/=	Pembagian Bilangan
%	PerolehSisa Pembagian

- ***Operator Pembanding***

Operator Pembanding merupakan suatu operator yang digunakan untuk membandingkan antara dua buah nilai . Hasil perbandingan dari operator ini adalah TRUE dan FALSE (Tipe data BOOLEAN).

Tabel 1.3. Operator Pembanding

Operator	Keterangan
==	Sama Dengan
!=	Tidak sama Dengan
>	Lebih Besar dari
<	Kurang Dari
>=	Lebih besar sama dengan
<=	Kurang dari sama dengan

- ***Operator Logika***

Operator logika adalah suatu operator yang membandingkan dua bukan KONDISI , mirip seperti dengan operator pembanding. Operator Logika ini menghasilkan nilai bertipe BOOLEAN , yaitu TRUE dan FALSE.

Tabel 1.4. Operator Logika

Operator	Keterangan
&&	Dan
||	Atau

6. Statement Kontrol

Statement kontrol digunakan untuk mengatur jalannya alur program sesuai dengan yang kita inginkan. Statement-statement ini dikategorikan kedalam tiga jenis yaitu: pemilihan, pengulangan, dan statement peloncatan. Pemilihan adalah suatu keadaan dimana pernyataan dapat dieksekusi apabila sesuatu kondisi memenuhi syarat untuk mengerjakan pernyataan tersebut. Pengulangan digunakan untuk melakukan eksekusi terhadap statement secara berulang sesuai dengan kondisi yang ditentukan. Adapun statement peloncatan digunakan untuk memindahkan proses eksekusi ke bagian kode program yang kita inginkan.

7. Pemilihan

Terdapat dua buah statement untuk proses pemilihan, yaitu `if` dan `switch`. Kedua buah statement tersebut digunakan untuk mengontrol eksekusi statement tergantung pada kondisi yang ditentukan sebelumnya. Statement `if` dapat digunakan untuk menangani pemilihan

statement yang didasarkan atas : satu, dua, atau lebih dari dua kondisi. Statement `switch` digunakan untuk menyederhanakan kompleksitas statement `if` yang banyak mengandung kondisi.

- Bentuk umum penulisan statement `if` untuk satu kondisi

```
If(kondisi) {
   Statement1;
   Statement2;
   ……
}
```

- Bentuk umum penulisan statement `if` untuk dua kondisi

```
If(kondisi) {
//statement yang akan dilakukan jika kondisi benar
   Statement1;
   Statement2;
   ……
 } else {
//statement yang akan dilakukan jika kondisi salah
   Statement1;
   Statement2;
   ……
 }
```

- Bentuk umum penulisan statement `if` untuk tiga kondisi atau lebih

```
If(kondisi1) {
//statement yang akan dilakukan jika kondisi1 benar
   Statement1;
   Statement2;
   ……
 } else if(kondisi2) {
//statement yang akan dilakukan jika kondisi2 benar
   Statement1;
   Statement2;
   ……
 } else {
//statement yang akan dilakukan jika kondisi1 dan kondisi2 salah
   Statement1;
   Statement2;
   ……
 }
```

- Bentuk umum penulisan statement `switch`

```
switch(ekspresi)  {
   case nilai1:
   //statement yang akan dilakukan bila ekspresi sama dengan nilai1
   break;
```

```
  case nilai2:
  //statement yang akan dilakukan bila ekspresi sama dengan nilai2
  break;
  ……
  case nilaiN:
  //statement yang akan dilakukan bila ekspresi sama dengan nilaiN
  break;
  default:
}
```

8. Pengulangan

Terdapat tiga buah jenis struktur pengulangan, yaitu: `for, while, dan do-while`.

- Struktur `for`

  Digunakan untuk melakukan pengulangan yang banyaknya sudah pasti atau sudah diketahui. Proses pengulangan akan terus dilakukan selama kondisi menghasilkan nilai true.

  Bentuk umum struktur `for`:

```
for(inisialisasi;kondisi;iterasi){
  //statement yang akan diulang
}
```

  Inisialisasi = berupa tipe data dan nilainya.

  Kondisi = kondisi akan dibandingkan dengan inisialisasi

  Iterasi = varibel pengontrol untuk melakukan proses increment atau decrement

- Struktur `while`

  Jenis pengulangan yang mendefinisikan kondisi di awal blok. Apabila kondisi tidak terpenuhi (bernilai false) maka proses pengulangan tidak akan pernah dilakukan. Bentuk umum struktur `while` :

```
inisialisasi
while(kondisi){
  //statement yang akan diulang
  ……
  iterasi
}
```

- Struktur `do-while`

  Penempatan kondisi berada di akhir blok, sehingga proses pengulangan akan dilakukan minimal sekali meskipun ternyata kondisinya tidak terpenuhi (bernilai false). Bentuk umum struktur `do-while` :

```
inisialisasi
do{
   //statement yang akan diulang
   ……
   iterasi
} while(kondisi);
```

9. Statement Peloncatan

Statement peloncatan digunakan untuk mengontrol jalannya program. Lebih tepatnya, untuk memindahkan eksekusi program ke baris kode yang dikehendaki. Dalam Java terdapat tiga buah statement peloncatan, yaitu: `break`, `continue`, dan `return`.

III. Praktikum

a. Ketiklah dan analisislah program berikut :

```
Package belajar;
import java.util.*;

public class Belajar {
    public static void main(String[] args) {
            Scanner input = new Scanner (System.in);
            int a,b,tambah,kurang;
            System.out.print(" Bilangan Pertama : ");
            a = input.nextInt();
            System.out.print(" Bilangan Kedua : ");
            b = input.nextInt();
            tambah = a+b;
            kurang = a-b;
            System.out.println(" Hasil Penjumlahan :"+tambah);
            System.out.println(" Hasil Pengurangan :"+kurang);
    }
}
```

b. Sempurnakan serta pahami program sederhana berikut

```
public class Percabangan {
    public static void main(String[] args) {
        int b=2;
        if(b==1){
            System.out.println("Nilai Variabel b adala 1r");
        }else if(b==2){
            System.out.println("Nilai Variabel b adalah 2");
        }
    }
}
```

```
public class pengulangan {
   public static void main(String args[]){
      for(x=1;x<10;x++){
         System.out.println("Ini Pengulangan ke-"+x);
      }
   }
}
```

```
package pengulangan;
public class pengulangan {
   public static void main(String args[]){
      while (x<10) {
      System.out.println("Ini Pengulangan ke-+x)
      x++;
    }
}
```

```
public class pengulangan {
   public static void main(String args[]){
      int x=1;
      do{
         System.out.println("Ini pengulangan ke-"+x);
         x++;
      }while (x<0);
   }
}
```

IV. Tugas

1. Buatlah program untuk menghitung nilai rata-rata dari angka yang dimasukkan oleh pengguna.

   Input :

   - jumlah deret angka yang akan dihitung rata-ratanya
   - angka-angka yang akan dihitung rata-ratanya

   Proses : menghitung rata-rata

   Output : hasil rata-rata
2. Buatlah program untuk menghitung gaji bersih setelah dipotong pajak 5%.
3. Buatlah program pengulangan sederhana dengan output angka 1-15 dengan menggunakan pengulangan while, do-while dan for. Kemudian tampilkan total 15 angka dari pengulangan tersebut.

MODUL II

MENERAPKAN ALGORITMA DIVIDE AND CONQUER

I. Tujuan Praktikum

1. Mahasiswa dapat memahami dan menerapkan konsep array dengan menggunakan bahasa pemrograman Java.
2. Mahasiswa dapat memahami dan menerapkan konsep function dengan menggunakan bahasa pemrograman Java.
3. Mahasiswa dapat menerapkan algoritma divide and conquer dalam pemrograman Java.

II. Landasan Teori

a. ARRAY

Array adalah sebuah variabel yang bisa menyimpan banyak data dalam satu variabel. Array menggunakan indeks untuk memudahkan akses terhadap data yang disimpannya. Untuk mendeklarasikan sebuah array dapat dituliskan seperti berikut ini:

```
//cara pertama
String[] nama;

// cara kedua
String nama[];

// cara ketiga dengan kata kunci new
String[] nama = new String[5];
```

- Kurung siku [] digunakan untuk membuat array
- Kurung siku bisa diletakkan setelah tipe data atau nama array
- Angka 5 dalam kurung artinya batas atau ukuran array-nya

Cara yang dapat digunakan untuk mengambil data dari sebuah array antara lain seperti dibawah ini:

```
// membuat array
String[] nama = {"Linda", "Santi", "Susan", "Mila",
"Ayu"};
// mengambil data array
System.out.println(teman[2]);
```

Untuk dapat menampilkan seluruh data array dapat dilakukan dengan menggunakan atribut length yang digunakan untuk mengambil banyak nya data yang ada pada suatu array. Contoh program untuk mengambil seluruh data array antara lain sebagai berikut.

```
public class Cobaprak {
public static void main(String[] args) {
    String [] nama = {"Linda", "Santi", "Susan", "Mila", "Ayu"};
    for (int i= 0; i<nama.length; i++){ System.out.println("nama ke-"+i+": "+nama[i]);
    }
}
```

Multidimensi Array

Array multidimensi adalah array yang berisi satu atau lebih array. Untuk membuat array dua dimensi, tambahkan setiap array dalam set kurungnya sendiri:

```
Tipedata [][] namaarray = new tipedata [n][m];
```

Dimana n adalah jumlah elemen untuk baris dan m adalah jumlah elemen kolom.

Array List

Array list merupakan sebuah class yang memungkinkan kita membuat sebuah objek untuk menampung apapun.

Contoh Program dengan Array List

```
import java.util.ArrayList;
public class Doraemon {
public static void main(String[] args) {

// membuat objek array list
ArrayList kantongAjaib = new ArrayList();

// Mengisi kantong ajaib dengan 5 benda kantongAjaib.add("Senter Pembesar");
kantongAjaib.add(532); kantongAjaib.add("tikus");
kantongAjaib.add(1231234.132); kantongAjaib.add(true);

// menghapus tikus dari kantong ajaib kantongAjaib.remove("tikus");
// Menampilkan isi kantong ajaib System.out.println(kantongAjaib);
// menampilkan banyak isi kantong ajaib
System.out.println("Kantong ajaib berisi "+ kantongAjaib.size() +" item");
}
}
```

b. FUNCTION

Prosedur/fungsi dapat memecah program menjadi sub-sub program, sehingga dapat membuat program lebih efisien. Penggunaan prosedur/fungsi dapat mengurangi pengetikan kode yang berulang-ulang.

Prosedur adalah sebutan untuk fungsi yang tidak mengembalikan nilai. Fungsi ini biasanya ditandai dengan kata kunci void. **Fungsi** adalah sebutan untuk fungsi yang mengembalikan nilai. **Method** adalah fungsi yang berada di dalam Class. Sebutan ini, biasanya digunakan pada OOP. Fungsi harus dibuat atau ditulis di dalam class. Struktur dasarnya seperti ini:

```
static TypeDataKembalian
namaFungsi(){
        // statement atau kode fungsi
}
```

Penjelasan:

1. Kata kunci static, artinya membuat fungsi yang dapat dipanggil tanpa harus membuat instansiasi objek.
2. TypeDataKembalian adalah tipe data dari nilai yang dikembalikan setelah fungsi dieksekusi.
3. namaFungsi() adalah nama fungsinya. Biasanya ditulis dengan huruf kecil di awalnya. Lalu, kalau terdapat lebih dari satu suku kata, huruf awal di kata kedua ditulis kapital.
4. Tipe data void artinya kosong, fungsi tersebut tidak mengebalikan nilai apa-apa.

Contoh pemanggilan fungsi dalam dalam funsgi main:

```
public static void main(String[] args){
   ucapSalam();
}
```

Fungsi dengan Parameter

Parameter adalah variabel yang menampung nilai untuk diproses di dalam fungsi. Parameter berperan sebagai input untuk fungsi.

Struktur dasarnya seperti ini:

```
static TipeData namaFungsi(TipeData namaParameter. TipeData namaParameterLain){
        //kode fungsi
}
```

Penjelasan:

1. Parameter ditulis di antara tanda kurung (...);
2. Parameter harus diberikan tipe data;
3. Bila terdapat lebih dari satu parameter, maka dipisah dengan tanda koma.

Contoh fungsi yang memiliki parameter:

```
static void ucapin(String ucapan){
System.out.println(ucapan);
}
```

Fungsi yang Mengembalikan Nilai

Setelah fungsi memproses data yang diinputkan melalui parameter, selanjutnya fungsi harus mengembalikan nilai agar dapat diolah pada proses berikutnya. Pengembalian nilai pada fungsi menggunakan kata kunci **return**.

Contoh:

```
static int luasPersegi(int sisi){
   int luas = sisi * sisi;
   return luas;
}
```

c. ALGORITMA DIVIDE AND CONQUER

Algoritma Divide and Conquer merupakan algoritma yang sangat populer di dunia Ilmu Komputer. Divide and Conquer merupakan algoritma yang berprinsip memecah permasalahan yang terlalu besar menjadi beberapa bagian kecil sehingga lebih mudah untuk diselesaikan. Langkah-langkah umum algoritma Divide and Conquer :

1. Divide : Membagi masalah menjadi beberapa upa-masalah yang memiliki kemiripan

dengan masalah semula namun berukuran lebih kecil (idealnya berukuran hampir sama).

2. Conquer : Memecahkan (menyelesaikan) masing-masing upa-masalah (secara rekursif).
3. Combine : Menggabungkan solusi masing-masing upa-masalah sehingga membentuk solusi masalah semula.

III. Langkah Praktikum

Project pertama (program tempat duduk)

1. Bukalah aplikasi netbeans, pilih menu new project → java application, kemudian beri nama project dengan TempatDuduk. Lalu klik finish dan netbeans akan membuat suatu file java dengan nama TempatDuduk.
2. Setelah form terbuka, ketikan kode program berikut ini. import java.util.Scanner;

```
import java.util.Scanner;
public class TempatDuduk {
    public static void main(String[] args) {
        String[][] meja = new String[2][3];
        Scanner scan = new Scanner(System.in);

        for(int bar = 0; bar < meja.length; bar++){
            for(int kol = 0; kol < meja[bar].length; kol++){
                System.out.format("Siapa yang akan duduk di meja (%d,%d): ", bar, kol);
                meja[bar][kol] = scan.nextLine();
            }
        }
        System.out.println("-------------------------");
        for(int bar = 0; bar < meja.length; bar++){
            for(int kol = 0; kol < meja[bar].length; kol++){
                System.out.format("| %s | \t", meja[bar][kol]);
            }
            System.out.println("");
        }
        System.out.println("-------------------------");
    }
}
```

3. Jalankan program dengan menekan tombol F6 lalu akan muncul hasil seperti berikut

```
run:
Siapa yang akan duduk di meja (0,0): Lina
Siapa yang akan duduk di meja (0,1): Putri
Siapa yang akan duduk di meja (0,2): Dio
Siapa yang akan duduk di meja (1,0): David
Siapa yang akan duduk di meja (1,1): Kara
Siapa yang akan duduk di meja (1,2): Nina
-------------------------
| Lina |        | Putri |       | Dio |
| David |       | Kara |        | Nina |
-------------------------
```

Project kedua

1. Bukalah aplikasi netbeans, pilih menu new project → java application, kemudian beri nama project dengan FungsiProsedur. Lalu klik finish dan netbeans akan membuat suatu file java dengan nama FungsiProsedur.
2. Setelah form terbuka, ketikan kode program berikut ini. import java.io.BufferedReader;

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;

public class FungsiProsedur {

    static ArrayList listBuah = new ArrayList(); static boolean isRunning = true;
    static InputStreamReader inputStreamReader = new InputStreamReader(System.in);
    static BufferedReader input = new BufferedReader(inputStreamReader);
    static void showMenu() throws IOException{

        System.out.println("========= MENU ========");
        System.out.println("[1] Show All Buah");
        System.out.println("[2] Insert Buah");
        System.out.println("[3] Edit Buah");
        System.out.println("[4] Delete Buah");
        System.out.println("[5] Exit");
        System.out.print("PILIH MENU> ");
        int selectedMenu = Integer.valueOf(input.readLine());
        switch(selectedMenu){
          case 1:
             showAllBuah();
             break;
          case 2:
             insertBuah();
             break;
```

```
            case 3:
                editBuah();
                break;
            case 4:
                deleteBuah();
                break;
            case 5:
            System.exit(0);
            break;
          default:
            System.out.println("Pilihan salah!");
          }
    }
    static void showAllBuah(){
        if(listBuah.isEmpty()){
            System.out.println("Belum ada data");
        } else {
            // tampilkan semua buah for(int i = 0; i < listBuah; i++){
                  System.out.println(String.format("[%d] %s",i, listBuah.get(i)));
          }
        }
    }
    static void insertBuah() throws IOException{
        System.out.print("Nama buah: ");
        String namaBuah = input.readLine();
        listBuah.add();
    }
    static void editBuah() throws IOException{
        showAllBuah();
        System.out.print("Pilih nomer buah: ");
        int indexBuah = Integer.valueOf(input.readLine());
        System.out.print("Nama Baru: "); String namaBaru = input.readLine();
        // ubah nama buah listBuah.set(indexBuah, namaBaru);
    }
    static void deleteBuah() throws IOException{
        showAllBuah();
        System.out.print("Pilih nomer buah: ");
        int indexBuah = Integer.valueOf(input.readLine());
        // hapus buah listBuah.remove();
    }
    public static void main(String[] args) throws IOException {
        do {
          showMenu();
        } while (isRunning);
    }
}
```

IV. Soal Latihan

1. Buatlah program java untuk menjumlahkan 2 buah matriks A dan B dengan ukuran masing masing 2 x 3. Pada program harus menggunakan konsep Array.
2. Perbaikilah kode program dibawah ini sehingga program dapat dijalankan .

Class 'myInput'

```
Package arraykaryawan;
Import java.io.*;

   Public class myInput {
     Public string bacaString () {
       BufferedReader bfr = new BufferedReader
         (newInputStreamReader (System.in), 1);
       String string = " ";
       Try
       {
         String = bfr.readLine();
       }
       Catch (IOException ex)
       {
         System.out.println(ex);
       }
       Return string;
     }
     Public int bacaInt() {
       Return Interger.parseInt(bacaString());
     }
   }
```

Class 'dataKaryawan'

```
Package arraykaryawan;

Public class dataKaryawan {
   Private string nik = new string[100];
   Private string nama = new string[100];
   Private int gapok = new int[100];
   Public int n;

   Public void setNIK (int i, string x)
   {
      This.nik = x;
   }
```

```
   Public void setNama (int i, string x)
   {
      This.nama = x;
   }

   Public void setGapok (int i, int x)
   {
      This.gapok = x;
   }

   Public float rerataGapok (int i, int x)
   {
      Int sum = 0, I;
      For (i=0; i<this.n; i++)
        {
          Sum += this.gapok[i];
        }
      Return sum/this.n;
   }

   Public void setJmlKaryawan(int x)
   {
      This.n = x;
   }

   Public void tampilData ()
   {
      Int i;
      System.out.println ("=====================================")
      System.out.println ("NIK           NAMA             GAJI")
      System.out.println ("=====================================")
      For (i=0; I<this.n; i++)
      {
        System.out.println (string.format("%-5s %-35s Rp %10d",
      nik[i], nama[i], gapok[i]));
         }
        System.out.println("=====================================")
        System.out.println("Rata-rata gaji pokok dari
 "+this.n+"karyawan adalah Rp " + this.rerataGapok());
   }
}
```

Bagian main class Package arraykaryawan;

```
Package arraykaryawan;
Public class dataKaryawan {
   Public static void main(String[] args){
```

```
        Int I;
        dataKaryawan dataKar = new dataKaryawan();
        myInput input1 = myInput();
        System.out.println("Berapa jumlah karyawan: ");
        dataKar.setJmlKaryawan(input1.bacaInt());

        for (i-0; I<dataKar.n; i++)
        {
           System.out.println("Karyawan ke- "+(i+1));
           System.out.println("Masukan NIK    : ");
           dataKar.setNIK(I, bacaString());
           System.out.println("Masukan Nama   : ");
           dataKar.setNama(I, bacaInt());
           System.out.println("Masukan Gaji Pokok   : ");
           dataKar.setGapok(I, input1.bacaInt());
        }
        dataKaryawan.tampilData();
    }
}
```

3. Buatlah program untuk mengurutkan bilangan yang diinput oleh user menggunakan algoritma Divide and Conquer.

MODUL III

SEARCHING

A. Tujuan

1. Mahasiswa dapat memahami Algoritma pencarian Beruntun dan Binary.
2. Mahasiswa dapat mengimplementasikan algoritma pencarian untuk menyelesaikan permasalahan.

B. Pembahasan

Searching merupakan proses untuk menemukan suatu data atau informasi dari sekumpulan data/informasi yang ada. Algoritma pencarian/searching algorithm merupakan algoritma yang menerima suatu kata kunci sebagai kriteria pencarian, dan dengan langkah-langkah tertentu akan mencari rekaman yang sesuai dengan kata kunci tersebut. Di dalam modul ini akandiperken alkan 2 teknik pencarian data yaitu pencarian beruntun/sequential searching dan pencarian biner/binary search

a. Pencarian Beruntun/Sequential Search

Sequential Search adalah teknik pencarian data dimana data dicari secara urut dari depan ke belakang atau dari awal sampai akhir. Kelebihan dari proses pencarian secara sequential ini jika data yang dicari terletak didepan, maka data akan ditemukan dengan cepat. Tetapi dibalik kelebihannya ini, teknik ini juga memiliki kekurangan. Pertama, jika data yang dicari terletak dibelakang atau paling akhir, maka akan membutuhkan waktu yang lama dalam proses pencariannya. Kedua, beban komputer akan semakin bertambah jika jumlah data dalam array sangat banyak.

Teknik pencarian data dari array yang paling mudah adalah dengan cara sequential search, dimana data dalam array dibaca 1 demi satu, diurutkan dari index terkecil ke index terbesar, maupun sebaliknya.

Contoh :

Array :

int a[5] = {0,3,6,10,1} (index array pada bahasa C dimulai dari index ke 0) jika kita ingin mencari bilangan 6 dalam array tersebut, maka proses yang terjadi kita mencari

1. dari array index ke-0, yaitu 0, dicocokan dengan bilangan yang akan dicari, jika tidak sama, maka mencari ke index berikutnya
2. pada array index ke-1, juga bukan bilangan yang dicari, maka kita mencari lagi pada index berikutnya
3. pada array index ke-2, ternyata bilangan yang kita cari ada ditemukan, maka kita keluar dari looping pencarian.

b. Binary Search

Metode pencarian yang kedua adalah binary search, pada metode pencarian ini, bila data belum terurut maka data harus diurutkan terlebih dahulu. Pencarian Biner (Binary Search) dilakukan untuk :

1. Memperkecil jumlah operasi pembandingan yang harus dilakukan antara data yang dicari dengan data yang ada khususnya untuk jumlah data yang sangat besar ukurannya.
2. Prinsip dasarnya adalah melakukan proses pembagian ruang pencarian secara berulang-ulang sampai data ditemukan atau sampai ruang pencarian tidak dapat dibagi lagi (berarti ada kemungkinan data tidak ditemukan).
3. Syarat utama untuk pencarian biner adalah data harus sudah terurut, misalkan terurut menaik.

Contoh :

Diketahui Data sebagai berikut, dan data yang dicari adalah 15

A	2	5	8	12	15	25	37	57
	1	2	3	3	5	6	7	8

Loop pertama :

Tengah=(BatasAtas + BatasBawah) div 2=(1 + 8) div 2=4

A [Tengah] = A [4] = 12 < cari = 15,

berarti BatasAtas = Tengah + 1 = 4 + 1 = 5

Loop kedua :

Tengah=(BatasAtas + BatasBawah) div 2=(5 + 8) div 2=6

A [Tengah] = A [6] = 25 > cari = 15,

berarti BatasBawah = Tengah – 1 = 6 – 1 = 5

Loop ketiga :

Tengah=(BatasAtas + BatasBawah) div 2=(5 + 5) div 2=5

A [Tengah] = A [5] = 15,

berarti setelah loop ketiga, data ditemukan

C. Langkah Kerja

a. Sequential Search

```
public class Sequential {

        public static void main(String[] args) {

            int[] data = {99,20,17,8,27,5,21,10,41,11};

            int cari=8;

            int i;

            boolean ditemukan = false;

        for(i=0; i < data.length; i++){

            if (data[i] == cari){

                ditemukan=true;

                break;

            }

        }

        if (ditemukan) {

            System.out.println("Data : " + cari + " Ditemukan Pada
Index :"+ i +".");

        }

        else{

            System.out.println("Data Tidak di temukan");

        }    }

}
```

b. Binary Search

```
public class Binary {



    public static void main(String[] args) {

        int N = 8;

        int A [] = {5,2,9,7,1,6,8,3};

        int BatasAtas, BatasBawah, Tengah;

        int cari = 2;



        BatasAtas = 0;

        BatasBawah = N - 1;

        Tengah = 0;

        boolean ketemu;

        ketemu = false;



        while((BatasAtas <= BatasBawah) && (ketemu == false)){

            Tengah = (BatasAtas + BatasBawah) / 2;

            if (A[Tengah] == cari){

                ketemu = true;

            } else

                if (A[Tengah] <= cari){

                BatasAtas = Tengah + 1;

            } else {

                BatasBawah = Tengah - 1;

            }
```

```
        }
        if (ketemu) {
            System.out.println ("Angka : "+ cari + " Data berada di index
       nomor " + Tengah);
        }
        else {
            System.out.println ("Angka :" + cari + "Data tidak ditemukan");
        }
    }
}
```

D. Tugas Praktikum

Tabel Produk

No	Produk	Harga
1	Jaguar	1.340.000.000
2	Lamborgini	34.500.000.000
3	Honda	350.000.000
4	Audi	2.000.000.000
5	Suzuki	245.000.000
6	Mazda	500.000.000
7	Daihatsu	169.000.000
8	Ford	789.000.000
9	Hyundai	122.900.000
10	Mitsubishi	278.100.000

Gunakan tabel Produk untuk mengerjakan tugas-tugas berikut.

1. Buatlah 2 array untuk menyimpan data Produk dan Harga.
2. Tampilkan produk bernama “Audi” serta lokasinya dalam struktur data Array.
3. Urutkanlah data harga dari nilai termurah hingga termahal.
4. Tampilkan harga produk termurah nomor ke-2.

MODUL IV
MINIMUM SPANNING TREE DAN SHORTEST PATH

I. Tujuan

1. Mengembangkan algoritma menggunakan *Minimum Spanning Tree*
2. Mengembangkan algoritma menggunakan metode *Single-Source Shortest Path*.

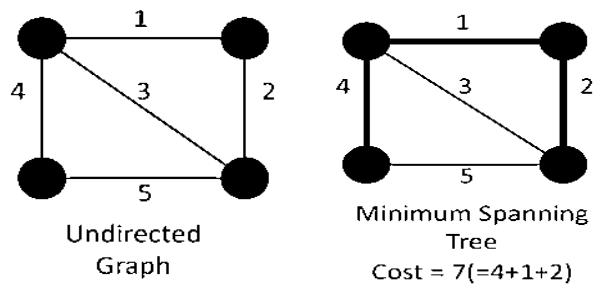
II. Pembahasan

A. Minimum Spanning Tree

Minimum Spanning Tree atau Pohon Merentang Minimum adalah teknik mencari jalur terdekat atau biaya minimum untuk menghubungkan semua titik atau simpul dalam suatu jaringan (graf) secara bersamaan sehingga membentuk pohon dan diperoleh jarak minimum. Pada Pohon Merentang Minimum semua simpul yang ada harus saling terhubung dengan menghasilkan jarak terdekat maksimal pada suatu graf, tetapi tidak boleh membentuk sirkuit. Minimum Spanning Tree banyak diaplikasikan di berbagai bidang sains dan teknologi seperti analisis Cluster pengenalan tulisan tangan, segmentasi gambar, pemodelan jaringan listrik dengan bobot (panjang kabel) minimum, pemodelan jaringan pipa PDAM, pemodelan gardu sinyal (Tower) pada perusahaan telekomunikasi, pemodelan pembangunan jalan raya, digunakan untuk memilih jalur dengan bobot pertimbangan, untuk memperbaiki biaya pembangunan jalan, dll.

Pencarian biaya minimum dari suatu graf sehingga membentuk minimum spanning tree memiliki syarat sebagai berikut:

a. Graf harus terhubung
b. Ruasnya punya bobot
c. Graf tidak berarah
d. Tidak membentuk siklus/sirkel/sirkuit



Gambar 1 Minimum Spanning Tree

Untuk menemukan pohon merentang minimum terdapat dua algoritma yang umum digunakan. Kedua algoritma tersebut adalah Algoritma Prim dan Algoritma Kruskal. Perbedaan dari kedua metode tersebut hanyalah pada langkah-langkah pencarian Pohon Merentang minimum saja pada suatu graf.

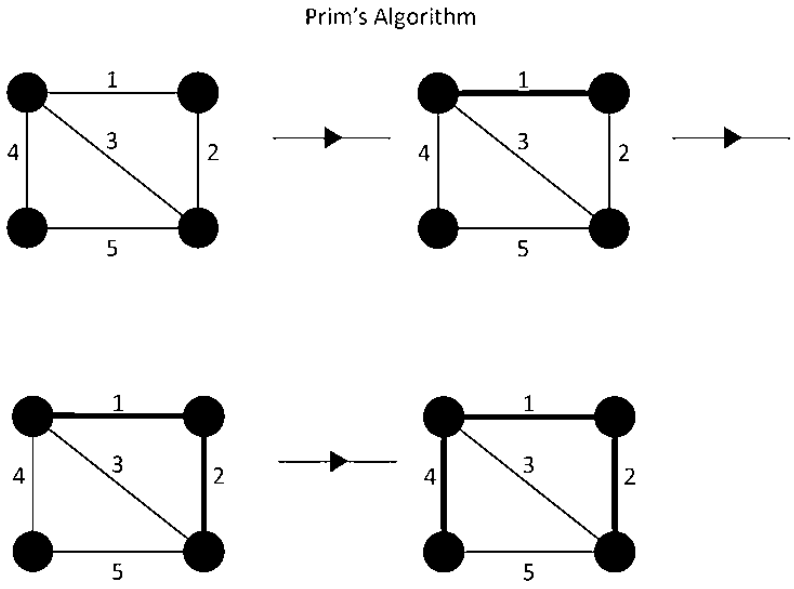
1. **Algoritma Prim**

Algoritma Prim digunakan untuk mencari pohon merentang minimum dari graf terhubung berbobot dengan cara mengambil sisi/ruas garis yang memiliki bobot terkecil dari graf, dimana ruas garis tersebut bersisian dengan pohon merentang yang telah dibuat dan yang tidak membentuk siklus.

Langkah-langkah dalam algoritma prim adalah sebagai berikut:

1) Ambil sisi graf G yang berbobot minimum, masukan kedalam T.
2) Pilih sisi (u,v) yang memiliki bobot minimum dan bersisian dengan simpul di T. Tetapi (u,v) tidak membentuk sirkuit di T. Tambahkan (u,v) kedalam T.
3) Ulangi langkah ke-2 sebanyak (n-2) kali.

Contoh:



2. **Algoritma Kruskal**

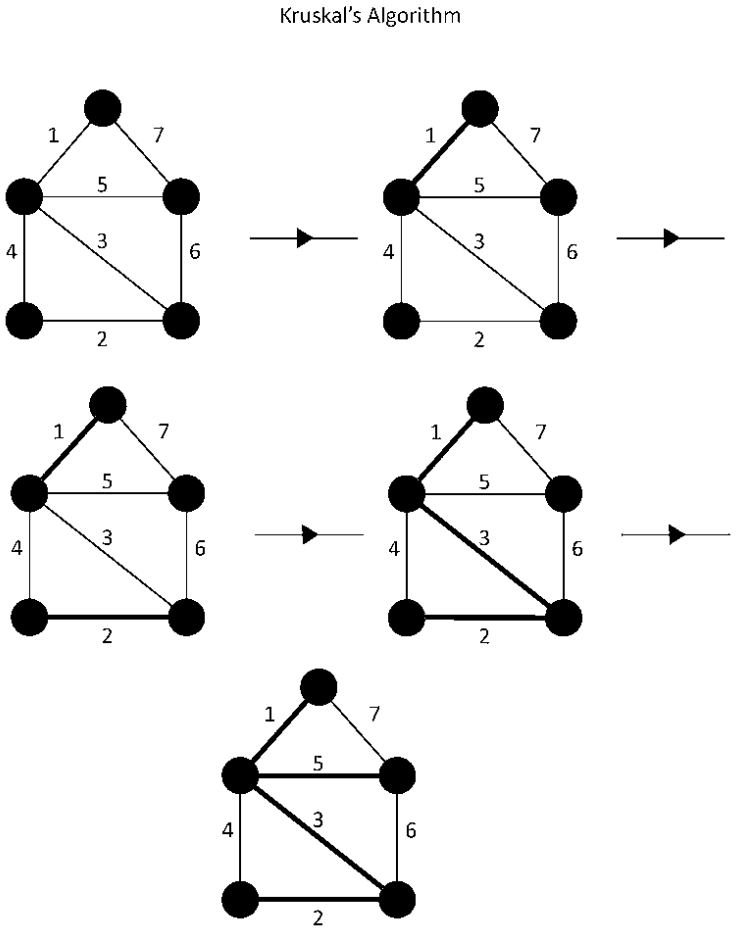
Algoritma kruskal adalah sebuah algoritma dalam teori graf yang mencari sebuah minimum spanning tree untuk sebuah graf berbobot yang terhubung. Langkah awal algoritma Kruskal adalah sisi-sisi di dalam graf diurutkan terlebih dulu berdasarkan bobot

dari yang kecil ke besar. Sisi-sisi yang sudah urut berdasarkan bobot akan membentuk hutan dengan masing-masing pohon di hutan hanya berupa satu buah simpul, disebut dengan hutan rentang (*spanning forest*). Misalkan T adalah pohon rentang yang sisi-sisinya diambil dari graf G. Sisi yang dimasukkan ke dalam himpunan T adalah sisi graf G sedemikian sehingga T adalah pohon. Sisi dari graf G ditambahkan ke T jika ia tidak membentuk sirkuit atau siklus di T.

Langkah-langkah dalam algoritma Kruskal adalah sebagai berikut:

1) Urutkan semua setiap sisi di graf berdasarkan bobotnya mulai dari sisi dengan bobot terkecil sampai terbesar.
2) Pilih sisi yang mempunyai bobot minimum yang tidak membentuk sirkuit di T. Tambahkan sisi tersebut ke dalam T.
3) Ulangi langkah 2 sampai pohon merentang minimum terbentuk, yaitu ketika sisi di dalam pohon merentang minimum berjumlah n-1 (n adalah jumlah simpul di graf).

Contoh:



B. Shortest Path

Shortest path adalah pencarian rute atau path terpendek antara node yang ada pada graf, biaya (*cost*) yang dihasilkan adalah minimum. Graf yang digunakan dalam pencarian lintasan terpendek adalah graf berbobot (*weighted graph*) yaitu graf yang setiap sisinya diberikan suatu nilai atau bobot. Bobot pada sisi graf dapat menyatakan jarak antar kota, waktu pengiriman pesan, ongkos pembangunan dan sebagainya. Secara umum "terpendek" berarti meminimalisasi bobot pada suatu lintasan didalam graf, sehingga dalam pencarian lintasan terpendek masalah yang dihadapi adalah mencari lintasan mana yang akan dilalui sehingga didapat lintasan yang paling pendek dari satu verteks ke verteks yang lain.

Shortest path dapat dibedakan menjadi :

- *Single Source Shortest Path*
  Menentukan shortest path dari verteks sumber s ∈ V ke setiap verteks v ∈ V
- *Single Destination Shortest Path*
  Menentukan shortest path ke suatu tempat t dari tiap verteks v
- *Single Pair shortest path*
  Menentukan shortest path dari u ke v jika diketahui pasangan u dan v
- *All pair shortest path*
  Untuk semua pasangan (u,v) ditentukan kemungkinan shortest pathnya.

Dalam pembahasan ini kita akan fokus pada *Single Source Shortest Path*. Ada 2 macam algoritma yang digunakan dalam memecahkan masalah *single source shortest path*, yaitu Algoritma Djikstra dan algoritma Bellman Ford.

1. Algoritma Djikstra

Algoritma Dijkstra ditemukan oleh Edger Wybe Dijkstra yang merupakan salah satu jenis bentuk algoritma populer dalam pemecahan persoalan yang terkait dengan masalah optimasi dan bersifat sederhana. Algoritma ini dipakai dalam memecahkan permasalahan jarak terpendek (*shortest path*) untuk sebuah graf berarah (*directed graph*). Pada algoritma dijkstra pemecahan masalah diperuntukkan untuk sebuah Graf G = (V, E) yang berbobot non negatif . Diasumsikan w (u,v) ≥ 0 untuk masing-masing edge (u, v) ∈ E.

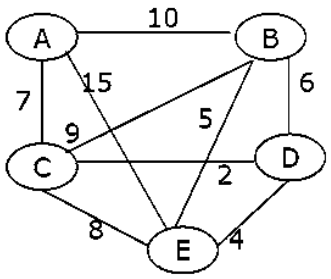
Metode algoritma Djikstra:

1) Inisialisasi s (sumber)
   - Pilih salah satu verteks sebagai sumber
   - Maka d(s) = 0
   - Beri label 0 pada verteks s

2. Untuk masing-masing edge → e ∈ E
   - Jika u adalah endpoint dari e yang sudah diberi label dan v adalah endpoint yang belum diberi label maka p (u,v) adalah = d (u) + w(u,v)
3. e adalah edge untuk T yang mempunyai nilai P terkecil
   - Jika u adalah endpoint dari e yang sudah diberi label dan v adalah endpoint yang belum diberi label maka tambahkan e dan verteks v ke pohon T
   - d(v) = P(uv)
   - Beri label d(v) pada vertek v
4. Kembali ke nomor 2

Contoh:

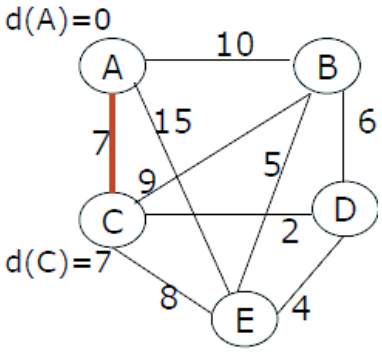
Tentukan shortest path dari A ke setiap v pada graf G berikut:



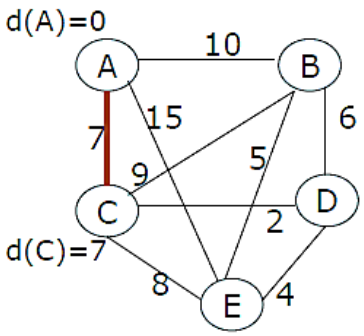
Penyelesaian:

Spanning tree T kosong

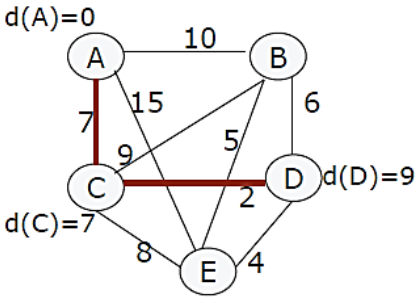
1) Inisialisasi s (sumber)
   - pilih verteks A sebagai sumber.
   - S = A, maka d(A) = 0. beri label 0 pada A
2) Untuk masing-masing edge ∈ E,
   - u adalah endpoint yg sudah diberi label, u = A
   - v adalah endpoint yg belum diberi label, v = B,C,D,E
   - P(AB) = 10, P(AC) = 7, P(AE) = 15



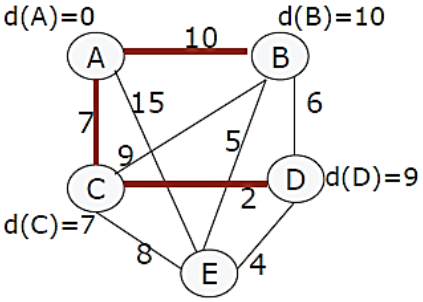
3) AC yang mempunyai nilai P terkecil sehingga C ditambahkan ke spanning tree T
   - d(C)=P(AC)=7
   - Beri label d(C) pada verteks c



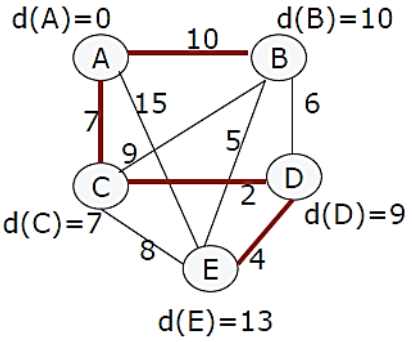
4) Kembali ke no 2
   - P(AB)=10,P(AE)=15, P(CB)=22,P(CD)=9,P(CE)=15
   - CD yg mempunyai nilai P terkecil, sehingga D ditambahkan ke T
   - Beri label d(D)=9



5) Kembali ke no 2
   - P(AB)=10,P(AE)=15, P(CB)=22,P(CE)=15,P(DB)=15,P(DE)=13
   - AB yg mempunyai nilai terkecil,sehingga B ditambahkan ke T
   - Beri label d(B) =10A



6) Kembali ke no 2
   - P(AE)=15, P(CB)=22, P(CE)=15,P(DB)=15,
   - P(DE)=13, P(BE)=18
   - DE yg mempunyai nilai terkecil,sehingga E ditambahkan ke T
   - Beri label d(E) =13



7) Semua verteks sudah diberi label
8) Selesai

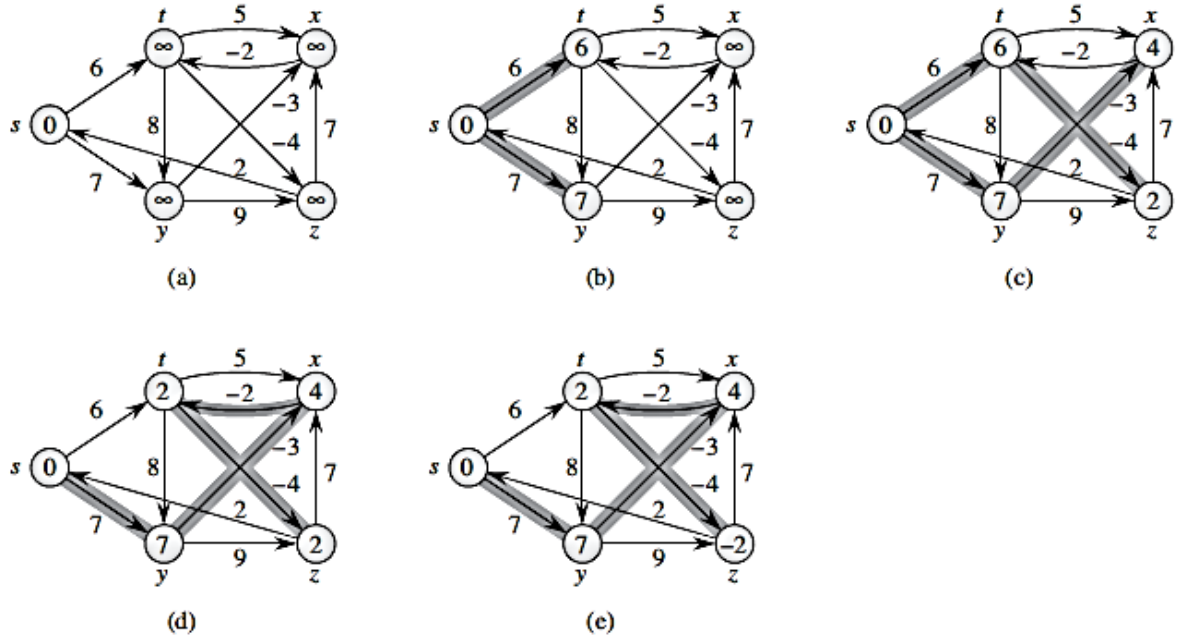
2. Algoritma Bellman Ford

Algoritma Bellman-Ford adalah salah satu algoritma untuk shortest path di mana algoritma ini dapat menentukan jalur terpendek dari sebuah simpul/verteks ke semua simpul lain dari sebuah graf berbobot. Algoritma ini mirip dengan algoritma Dijkstra tetapi bedanya algoritma Bellman-Ford dapat bekerja dengan graf yang sisi/edge dapat memiliki bobot negatif. Sisi/edge berbobot negatif menjelaskan banyak fenomena seperti arus kas (cashflow), panas yang dilepaskan / diserap dalam reaksi kimia, dll. Misalnya, untuk menjangkau dari satu bahan kimia A ke bahan kimia B, setiap metode akan memiliki sub-reaksi yang melibatkan pelepasan/dispasi dan penyerapan/ absorsi panas. Jika kita ingin mencari himpunan reaksi yang memerlukan energi minimum, kita harus dapat memfaktorkan penyerapan panas sebagai bobot negatif dan disipasi panas sebagai bobot positif.

Algoritma Bellman-Ford :

```
Bellman-Ford(G,w,s)
            Inisialisasi single source(G,s)
    for i=1 to |V[G]|-1
            do for each edge (u,v)∈E[G]
                    do RELAX(u,v)
for each edge (u,v) ∈E[G]                ;
untuk mencek apakah ada atau tidak cycle dgn bobot negatif
            do if d[v] > d[u] +w ((u,v))          ;
jika hasil algoritma yang diinginkan belum didapat
            then return FALSE
return TRUE;
```

Contoh:



III. Langkah Kerja

A. Minimum Spanning Tree

Algoritma Prim

```
import java.util.Arrays;
class PGraph {
  public void Prim(int G[][], int V) {
      int INF = 9999999;
      int no_edge;
      boolean[] selected = new boolean[V];
      Arrays.fill(selected, false);
      no_edge = 0;
      selected[0] = true;
      System.out.println("Edge : Weight");

      while (no_edge < V - 1) {
      int min = INF;
      int x = 0; // row number
      int y = 0; // col number

      for (int i = 0; i < V; i++) {
        if (selected[i] == true) {
          for (int j = 0; j < V; j++) {
             if (!selected[j] && G[i][j] != 0) {
                if (min > G[i][j]) {
                   min = G[i][j];
                   x = i;
                   y = j;
          } } } }
       }
      System.out.println(x + " - " + y + " :  " + G[x][y]);
      selected[y] = true;
      no_edge++;
     }
     }
```

```
    public static void main(String[] args) {
        PGraph g = new PGraph();
            int V = 4;
            int[][] G = { { 0, 1, 4, 3 }, { 1, 0, 0, 2 }, { 4, 0, 0, 5}, { 3, 2, 5, 0}};
            g.Prim(G, V);
    }
}
```

Algoritma kruskal

```
import java.util.Arrays;
class Graph {
  class Edge implements Comparable<Edge> {
    int src, dest, weight;
    public int compareTo(Edge compareEdge) {
      return this.weight - compareEdge.weight;
    } }

  class subset {
    int parent, rank;
  }
  int vertices, edges;
  Edge edge[ ];
  Graph(int v, int e) {
    vertices = v;
    edges = e;
    edge = new Edge[edges];
    for (int i = 0; i < e; ++i)
      edge[i] = new Edge();
  }

  int find(subset subsets[ ], int i) {
    if (subsets[i].parent != i)
      subsets[i].parent = find(subsets, subsets[i].parent);
```

```
  return subsets[i].parent;
}

void Union(subset subsets[ ], int x, int y) {
  int xroot = find(subsets, x);
  int yroot = find(subsets, y);

  if (subsets[xroot].rank < subsets[yroot].rank)
    subsets[xroot].parent = yroot;
  else if (subsets[xroot].rank > subsets[yroot].rank)
    subsets[yroot].parent = xroot;
  else {
    subsets[yroot].parent = xroot;
    subsets[xroot].rank++;
  } }

void KruskalAlgo() {
  Edge result[ ] = new Edge[vertices];
  int e = 0;
  int i = 0;
  for (i = 0; i < vertices; ++i)
    result[i] = new Edge();

  Arrays.sort(edge);
  subset subsets[ ] = new subset[vertices];
  for (i = 0; i < vertices; ++i)
    subsets[i] = new subset();

  for (int v = 0; v < vertices; ++v) {
    subsets[v].parent = v;
    subsets[v].rank = 0;
  }
  i = 0;
  while (e < vertices - 1) {
    Edge next_edge = new Edge();
```

```
    next_edge = edge[i++];
    int x = find(subsets, next_edge.src);
    int y = find(subsets, next_edge.dest);
    if (x != y) {
      result[e++] = next_edge;
      Union(subsets, x, y);
    }
  }
  for (i = 0; i < e; ++i)
    System.out.println(result[i].src + " - " + result[i].dest + ": " + result[i].weight);
}

public static void main(String[ ] args) {
  int vertices = 5; // Number of vertices
  int edges = 7; // Number of edges
  Graph G = new Graph(vertices, edges);

  G.edge[0].src = 0;
  G.edge[0].dest = 1;
  G.edge[0].weight = 1;

  G.edge[1].src = 0;
  G.edge[1].dest = 2;
  G.edge[1].weight = 7;

  G.edge[2].src = 1;
  G.edge[2].dest = 2;
  G.edge[2].weight = 5;

  G.edge[3].src = 1;
  G.edge[3].dest = 3;
  G.edge[3].weight = 4;

  G.edge[4].src = 1;
  G.edge[4].dest = 4;
  G.edge[4].weight = 3;
```

```
    G.edge[5].src = 2;
    G.edge[5].dest = 4;
    G.edge[5].weight = 6;

    G.edge[6].src = 3;
    G.edge[6].dest = 4;
    G.edge[6].weight = 2;
    G.KruskalAlgo();
  }
}
```

B. Shortest Path

Algoritma Djikstra

```
public class DijkstraAlgorithm {
  public static void dijkstraAlgorithm(int[ ][ ] graph, int source) {
    int count = graph.length;
    boolean[ ] visitedVertex = new boolean[count];
    int[ ] distance = new int[count];
    for (int i = 0; i < count; i++) {
      visitedVertex[i] = false;
      distance[i] = Integer.MAX_VALUE;
    }

    distance[source] = 0;
    for (int i = 0; i < count; i++) {
      int u = findMinDistance(distance, visitedVertex);
      visitedVertex[u] = true;

      for (int v = 0; v < count; v++) {
        if (!visitedVertex[v] && graph[u][v] != 0 && (distance[u] + graph[u][v] <
            distance[v])) {
          distance[v] = distance[u] + graph[u][v];
        } }
```

```
  }
  for (int i = 0; i < distance.length; i++) {
   System.out.println(String.format("Distance from %s to %s is %s", source, i,
        distance[i]));
  }
 }

 private static int findMinDistance(int[] distance, boolean[] visitedVertex) {
  int minDistance = Integer.MAX_VALUE;
  int minDistanceVertex = -1;
  for (int i = 0; i < distance.length; i++) {
   if (!visitedVertex[i] && distance[i] < minDistance) {
    minDistance = distance[i];
    minDistanceVertex = i;
   }
  }
  return minDistanceVertex;
 }

 public static void main(String[ ] args) {
  int graph[ ][ ] = new int[ ][ ] { { 0, 10, 7, 0, 15}, { 10, 0, 9, 6, 5}, { 7, 9, 0, 2, 8},
                                    { 0, 6, 2, 0, 4}, { 15, 5, 8, 4, 0} };
  DijkstraAlgorithm T = new DijkstraAlgorithm();
  T.dijkstraAlgorithm(graph, 0);
 }
}
```

Algoritma Bellman Ford

```
class Graph {
    class Edge {
        int src, dest, weight;
        Edge( ) {
                src = dest = weight = 0;
        } }
```

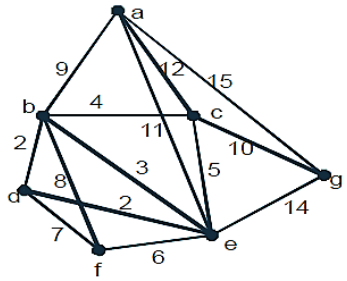
```
int V, E;
Edge edge[ ];

Graph(int v, int e) {
    V = v;
    E = e;
    edge = new Edge[e];
    for (int i = 0; i < e; ++i)
            edge[i] = new Edge();
}

void BellmanFord(Graph graph, int src) {
    int V = graph.V, E = graph.E;
    int dist[ ] = new int[V];

    for (int i = 0; i < V; ++i)
            dist[i] = Integer.MAX_VALUE;
    dist[src] = 0;

    for (int i = 1; i < V; ++i) {
            for (int j = 0; j < E; ++j) {
                    int u = graph.edge[j].src;
                    int v = graph.edge[j].dest;
                    int weight = graph.edge[j].weight;
                    if (dist[u] != Integer.MAX_VALUE && dist[u] + weight < dist[v])
                            dist[v] = dist[u] + weight;
            }
    }

    for (int j = 0; j < E; ++j) {
            int u = graph.edge[j].src;
            int v = graph.edge[j].dest;
            int weight = graph.edge[j].weight;
            if (dist[u] != Integer.MAX_VALUE && dist[u] + weight < dist[v]) {
```

```
            System.out.println("Graph contains negative weight cycle");
                return;
            }
    }
    printArr(dist, V);
}

void printArr(int dist[ ], int V) {
    System.out.println("Vertex Distance from Source");
    for (int i = 0; i < V; ++i)
            System.out.println(i + "\t\t" + dist[i]);
}

public static void main(String[ ] args) {
    int V = 5;
    int E = 10;

    Graph graph = new Graph(V, E);
    graph.edge[0].src = 0;
    graph.edge[0].dest = 1;
    graph.edge[0].weight = 6;
    graph.edge[1].src = 0;
    graph.edge[1].dest = 2;
    graph.edge[1].weight = 7;
    graph.edge[2].src = 1;
    graph.edge[2].dest = 2;
    graph.edge[2].weight = 8;
    graph.edge[3].src = 1;
    graph.edge[3].dest = 3;
    graph.edge[3].weight = 5;
    graph.edge[4].src = 1;
    graph.edge[4].dest = 4;
    graph.edge[4].weight = -4;
    graph.edge[5].src = 2;
```

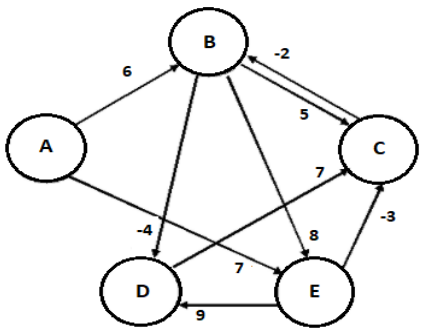
```
        graph.edge[5].dest = 3;
        graph.edge[5].weight = -3;
        graph.edge[6].src = 2;
        graph.edge[6].dest = 4;
        graph.edge[6].weight = 9;
        graph.edge[7].src = 3;
        graph.edge[7].dest = 1;
        graph.edge[7].weight = -2;
        graph.edge[8].src = 4;
        graph.edge[8].dest = 0;
        graph.edge[8].weight = 2;
        graph.edge[9].src = 4;
        graph.edge[9].dest = 3;
        graph.edge[9].weight = 7;
        graph.BellmanFord(graph, 0);
    }
}
```

C. Tugas Praktikum

1. Buatlah program untuk menentukan minimum spanning tree dari graf di bawah ini dengan menggunakan algoritma prim atau kruskal



2. Buatlah program untuk mencari jalur terpendek (*shortest path*) dari graf dibawah ini.



MODUL 5

STRING MATCHING

I. Tujuan:

1. Mengenal algoritma-algoritma *String Matchimg*
2. Menerapkan algoritma-algoritma *String Matching*

II. Pembahasan

Program pengeditan teks sering kali perlu menemukan kemunculan suatu pola dalam teks. Biasanya, teks adalah dokumen yang sedang diedit, dan pola yang dicari adalah kata tertentu yang dicari oleh pengguna. Algoritme yang efisien untuk masalah ini disebut *String Matching* atau pencocokan string.

Di antara banyak aplikasi lainnya, algoritma pencocokan string juga diterapkan untuk mencari pola tertentu dalam urutan DNA. Mesin pencari Internet juga menggunakannya untuk menemukan halaman Web yang relevan dengan kueri. Berikut ini adalah pembahasan beberapa algoritma *String Matching.*

A Naive

Metode ini memeriksa semua karakter dalam string utama dibandingkan dengan substring. Jika huruf pertama dari sub string cocok dengan string utama maka iterasi dilakukan pada *inner loop* dalam dan memeriksa elemen selanjutnya. Jika semua elemen dalam substring cocok dengan string utama maka akan dikembalikan indeks awal dari string utama dan sekali lagi akan diperiksa kemunculan berikutnya. Algoritma Naive adalah pencarian brute-force yang berguna optimal untuk string berukuran pendek karena untuk string yang panjang akan menghabiskan terlalu banyak waktu.

Implementasi Naive Brute-Force pada Java.

```
public class Naive {
    public static void search(String str, String pattern)
  {
     int n = str.length();
     int m = pattern.length();

     for (int s = 0; s <= n - m; s++) {
        int j;
```

```
        for (j = 0; j < m; j++)
            if (str.charAt(s + j) != pattern.charAt(j))
                break;
        if (j == m)
            System.out.println("Pola muncul pada indeks ke- " + s);
    }
  }

  public static void main(String[] args)
  {
    String txt = "AABAACAADAABAAABAA";
    String pat = "AABA";
    search(txt, pat);
  }
}
```

B. Rabin Karp

Rabin Karp ada algoritma pencarian yang memanfaatkan fungsi HASH. Rabin Karp bekerja seperti berikut:

1. Asumsikan teks adalah string t yang panjangnya n dan pattern (string yang akan dicari) P panjangnya m.
2. Assumsikan Si menyatakan sebuah substring dengan panjang m, yang berkelanjutan pada awal teks- misalkan S_0 adalah substring dengan panjang m diawal t
3. Ide utama dari algoritma ini adalah memanfaatka fungsi hash untuk memetakan setiap Si kedalam himpunan.

Fungsi hash digunakan untuk menempatkan suatu record yang mempunyai nilai kunci k. Fungsi hash yang paling umum berbentuk ; H(k) = k mod n. Di dalam algoritma ini k dapat berupa string P atau String Si

Algoritma Rabin Krap:

a. Pertama kita menghitung nilai fungsi hash dari string P.
b. Kemudian untuk masing-masing Si kita menhitung fungsi hashnya.
c. Lakukan penelusuran terhadap Si, jika h(Si) = h(P), maka lakukan pencocokan antara String Si dengan string P secara brute force.
d. Jika $h(Si) \neq h(P)$ kita tidak perlu melakukan pencocokan string, penelusuran dilanjutkan kembali terhadap Si yang berikutnya sampai ditemukan atau sampai string t berakhir.

Implementasi Naive Brute-Force pada Java.

```
public class RabinKarp {
  public final static int d = 10;

  static void search(String pattern, String txt, int q) {
            int m = pattern.length();
            int n = txt.length();
            int i, j;
            int p = 0;
            int t = 0;
            int h = 1;

            for (i = 0; i < m - 1; i++)
              h = (h * d) % q;

          // hitung nilai  hash untuk pattern and text
            for (i = 0; i < m; i++) {
              p = (d * p + pattern.charAt(i)) % q;
              t = (d * t + txt.charAt(i)) % q;
            }

            // Cari kecocokan
            for (i = 0; i <= n - m; i++) {
              if (p == t) {
                        for (j = 0; j < m; j++) {
                          if (txt.charAt(i + j) != pattern.charAt(j))
                            break;
                        }
                        if (j == m)
                          System.out.println("Pattern ditemukan pada posisi: " + (i + 1));
              }
              if (i < n - m) {
                t = (d * (t - txt.charAt(i) * h) + txt.charAt(i + m)) % q;
                if (t < 0)
                  t = (t + q);
              }
            }
  }

  public static void main(String[] args) {
            String txt = "ABCCDDAEFG";
            String pattern = "CDD";
            int q = 13;
            search(pattern, txt, q);
```

```
 }
}
```

C. Finite State Automata

Pada finite state automata pencarian akan dibagi dalam state-state. Ada state yang menunjukkan bahwa string diterima. Jadi bila dalam pembacaan string, masuk ke state tersebut, maka pola string ditemukan. Sebuah finite automaton M adalah sebuah 5-tuple ($Q,q_0,A, \sum, \delta$) dimana,

- Q adalah himpunan state
- q_0 adalah start state
- A adalah himpunan state yang diterima
- $\sum$ adalah himpunan alfabet masukan
- δ adalah fungsi transisi dari M

Finite state dimulai pada state q0 dan membaca karakter string inputnya satu per satu. Jika automaton dalam state q dan membaca karakter input a, maka automaton berpindah dari keadaan q ke keadaan δ (q, a). Kapan pun status q adalah anggota A, maka mesin M telah menerima pembacaan string. Masukan yang tidak diperbolehkan akan ditolak.

Sebuah finite automaton M menginduksi suatu fungsi $\emptyset$ yang disebut fungsi keadaan akhir (final-state function) , dari $\sum$ * ke Q sedemikian rupa sehingga $\emptyset$ (w) adalah keadaan M yang berakhir setelah memindai string w. Jadi, M menerima string w jika dan hanya jika $\emptyset$ (w) $\in$ A.

Implementasi Finite State pada Java

```
public class GFG {
   static int NO_OF_CHARS = 256;
   static int getNextState(char[] pat, int M,  int state, int x)
   {
         //Jika karakter c sama dengan karakter berikutnya di dalam pattern maka increment state
               if(state < M && x == pat[state])
                  return state + 1;

               // ns  adalah variabel yang menyimpan hasil state berikutnya (next state)
               int ns, i;
```

```
                // berikutnya ns berisi awalan terpanjang
                // yang juga merupakan akhiran dalam "pat [0..state-1] c"
                 // Mulai dari kemungkinan nilai terbesar
                // dan berhenti ketika Anda menemukan awalan yang akhiran

                    for (ns = state; ns > 0; ns--)
                    {
                      if (pat[ns-1] == x)
                      {
                        for (i = 0; i < ns-1; i++)
                          if (pat[i] != pat[state-ns+1+i])
                            break;
                          if (i == ns-1)
                            return ns;
                      }
                    }
        return 0;
   }

   /* membangun tabel untuk merepresentasikikan fungsi automata dari suatu pola */
   static void computeTF(char[] pat, int M, int TF[][])
   {
      int state, x;
      for (state = 0; state <= M; ++state)
         for (x = 0; x < NO_OF_CHARS; ++x)
            TF[state][x] = getNextState(pat, M, state, x);
   }

/*Mencetak kemunculan pola dalam text*/
   static void search(char[] pat, char[] txt)
   {
      int M = pat.length;
      int N = txt.length;

      int[][] TF = new int[M+1][NO_OF_CHARS];

      computeTF(pat, M, TF);

      // process pencocokan teks ke pola.
      int i, state = 0;
      for (i = 0; i < N; i++)
      {
         state = TF[state][txt[i]];
         if (state == M)
            System.out.println("Pattern found "
```

```
                + "at index " + (i-M+1));
      }
   }

   // Fungsi Main
   public static void main(String[] args)
   {
      char[] pat = "Saya mau makan".toCharArray();
      char[] txt = "ma".toCharArray();
      search(txt,pat);
   }
}
```

D. Knuth Morris Pratt (KMP)

Berikut ini adalah langkah-langkah yang dilakukan algoritma Knuth-Morris-Pratt pada saat mencocokkan string:

1. Algoritma Knuth-Morris-Pratt mulai mencocokkan pattern pada awal teks.
2. Dari kiri ke kanan, algoritma ini akan mencocokkan karakter per karakter pattern dengan karakter di teks yang bersesuaian, sampai salah satu kondisi berikut dipenuhi:
    a. Karakter dipattern dan di teks yang dibandingkan tidak cocok (mismatch).
    b. Semua karakter di pattern cocok. Kemudian algoritma akan memberitahukan penemuan di posisi ini.
3. Algoritma kemudian menggeser pattern berdasarkan tabel next, lalu mengulangi langkah 2 sampai pattern berada di ujung teks.

Implementasi KMP pada Java

```
public class KMP_String_Matching {
   void KMPSearch(String pat, String txt)
   {
      int M = pat.length();
      int N = txt.length();

      // buat variabel lps[] untuk menangani
      // nilai awalan dan akhiran terpanjang untuk pattern/pola
      int lps[] = new int[M];
      int j = 0;

            // pra pemrosesan pattern (kalkulasi array lps[])
      computeLPSArray(pat, M, lps);
```

```
    int i = 0; // index for txt[]
    while (i < N) {
        if (pat.charAt(j) == txt.charAt(i)) {
            j++;
            i++;
        }
        if (j == M) {
            System.out.println("Pattern ditemukan pada "
                               + "at index " + (i - j));
            j = lps[j - 1];
        }

        // ketidakcocokan pada pencocokan indeks J pattern
        else if (i < N && pat.charAt(j) != txt.charAt(i)) {
            if (j != 0)
                j = lps[j - 1];
            else
                i = i + 1;
        }
    }
}

void computeLPSArray(String pat, int M, int lps[])
{

        // panjang dari awalan(prefix) dan akhiran (suffix)
        //terpanjang sebelumnya
    int len = 0;

    int i = 1;
    lps[0] = 0;


        // Looping untuk kalkulasi lps[i] = 1 sampai M-1
    while (i < M) {
        if (pat.charAt(i) == pat.charAt(len)) {
            len++;
            lps[i] = len;
            i++;
        }
        else
        {

            if (len != 0) {
                len = lps[len - 1];

            }
            else             {
                lps[i] = len;
                i++;
            }
```

```
        }
      }
    }

    // Fungsi Main
    public static void main(String args[])
    {
      String txt = "ABABDABACDABABCABAB";
      String pat = "ABABCABAB";
      new KMP_String_Matching().KMPSearch(pat, txt);
    }
}
```

III. Tugas

Buatlah aplikasi pencariang dengan Tampilan seperti berikut:

Aplikasi Pencarian String

```
Masukan Teks : Saya mau makan
Masukan Teks yang dicari : ma
Pilih metode Pencarian:
 1. Naive
 2. Rabin Karp
 3. Infinite State Automata
 4. Knuth Morris Pratt
Metode yang dipilih : 4
```

```
Hasil Pencarian:

Teks ditemukan pada indeks ke-5
Teks ditemukan pada indeks ke-9

 Waktu Eksekusi: 430745.0 milidetik
```