PRAKTIKUM 8

MEMILIH STRUKTUR DATA

1. MINGGU KE : 10

2. PERALATAN : LCD, E-LEARNING

3. SOFTWARE : MAPLE

4. TUJUAN

Mahasiswa dapat memilih struktur data yang tepat sesuai dengan kebutuhannya.

5. TEORI PENGANTAR DAN LANGKAH KERJA

Ketika menulis program kita harus memutuskan bagaimana menyatakan data. Kadang dapat langsung dinyatakan, kadang harus direncanakan terlebih dulu. Pemilihan struktur data dapat membuat prosedur lebih efisien.

Misalkan ada sejumlah kota yang dihubungkan oleh jalan. Buat suatu prosedur untuk menentukan apakah kita dapat bepergian di antara dua kota tertentu. Masalah ini diselesaikan dengan teori graf.

```
> with(networks):
```

Buat graf baru G dan tambahkan beberapa kota (atau puncak dalam istilah teori graf).

```
> new(G):
> cities:={Zurich,Roma, Paris, Berlin, Vienna};
```

cities := { *Roma, Vienna, Berlin, Zurich, Paris* }

```
> addvertex(cities,G);
```

Roma, Vienna, Berlin, Zurich, Paris

Tambahkan jalan yang menghubungkan Zurich dengan Paris, Berlin, dan Vienna, masing-masing. Jalannya dinamakan e1, e2, dan e 3.

```
> connect({Zurich},{Paris,Berlin,Vienna},G);
```

e1, e2, e3

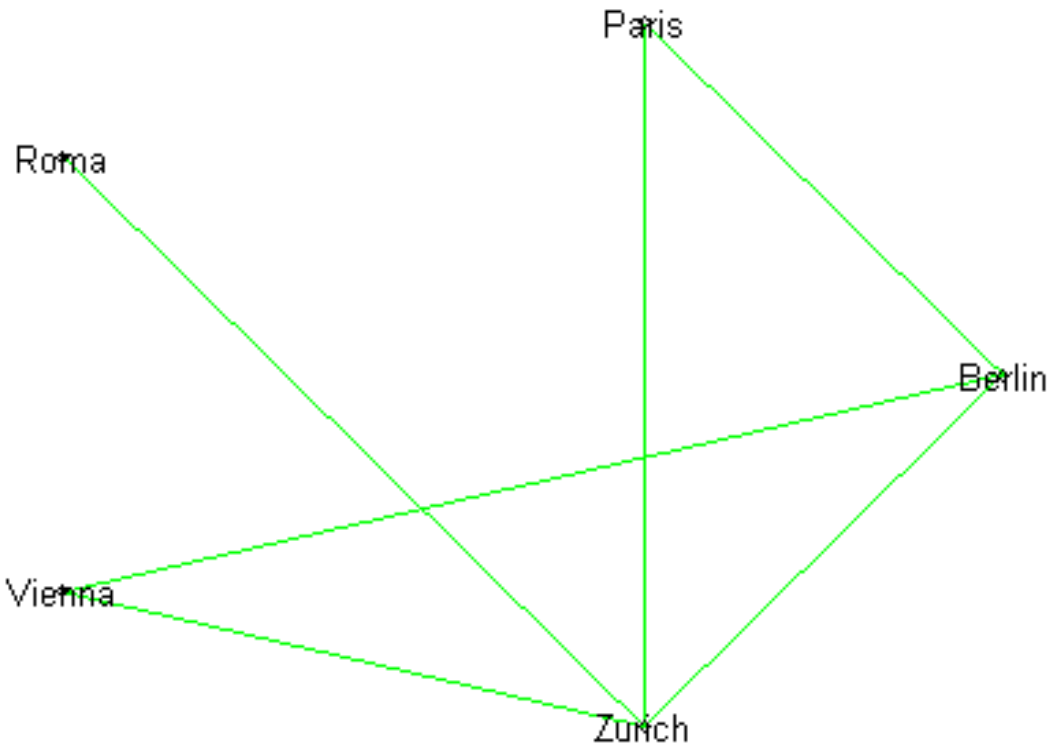
Tambahkan jalan yang menghubungkan Roma dengan Zurich dan Berlin dengan Paris dan Vienna.

```
> connect({Roma},{Zurich},G);
```
e4

```
> connect({Berlin},{Vienna,Paris},G);
```
e5, e6

Lalu gambarkan graf G.

```
> draw(G);
```



Dari gambar ini dapat dilihat jalur yang menghubungkan dua kota. Selain dengan bantuan gambar dapat juga dengan bantuan perintah connectivity.

```
> evalb(connectivity(G)>0);
```
true

Struktur data apa yang dapat menggambarkan kota dan jalan? Karena kota mempunya nama yang berbeda, maka dapat dinyatakan sebagai himpunan nama.

```
> vertices(G);
```

{ *Roma, Vienna, Berlin, Zurich, Paris* }

Paket network juga memberikan nama jalan yang berbeda, maka jalan juga dapat dinyatakan sebagai himpunan nama.

```
> edges(G);
```

{ *e1, e2, e3, e4, e5, e6* }

Jalan juga dapat dinyatakan sebagai himpunan yang terdiri dari dua kota yang terhubung.

```
> ends(e2,G);
```

{ *Berlin, Zurich* }

Jadi jalan dinyatakan sebagai himpunan dari himpunan.

```
> roads:=map(ends,edges(G),G);
```

roads := { { *Roma, Zurich* }, { *Vienna, Zurich* }, { *Berlin, Zurich* }, { *Zurich, Paris* }, { *Vienna, Berlin* }, { *Berlin, Paris* } }

Tetapi jika ingin tahu kota mana yang terhubung langsung ke Roma, misalnya, kita harus mencarinya melalui semua himpunan jalan. Jadi menyatakan data sebagai himpunan kota dan himpunan jalan adalah tidak efisien untuk mengetahui apakah kita dapat melakukan perjalanan melalui dua kota tertentu.

Kita juga dapat menyatakan data sebagai matriks adjasensi, matriks persegi dengan satu baris untuk setiap kota.

```
> adjacency(G);
```

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

Matriks ajasensi juga kurang efisien. Sekalipun setiap baris dalam matriks berkorespondensi dengan satu kota, tapi kita tidak tahu baris mana yang menghubungkan ke kota mana.

Berikut cara lain untuk menyatakan kota dan jalan.

```
> neighbors(Paris,G);
```

{ *Berlin, Zurich* }

Data dapat dinyatakan sebagai tabel dari neighbors: satu entri dalam tabel untuk setiap kota.

```
> T:=table(map(v->(v)=neighbors(v,G),cities));
```

T := table([*Roma* = { *Zurich* }, *Paris* = { *Berlin, Zurich* },
Zurich = { *Roma, Vienna, Berlin, Paris* }, *Vienna* = { *Berlin, Zurich* },
Berlin = { *Vienna, Zurich, Paris* }
]))

Sistem pernyataan kota dan jalan sebagai tabel dari neighbors cocok untuk menjawab pertanyaan apakah mungkin melakukan perjalanan antara dua kota tertentu. Mulai dari satu kota. Tabel akan mencarikan kota yang bertetangga. Demikian pula untuk mencari tetangga dari tetangga. Jadi dengan cepat dapat ditentukan seberapa jauh perjalanan dapat dilakukan.

Prosedur connected menentukan apakah kita dapat melakukan setiap dua kota. Perintah indices untuk mengekstrak himpunan kota dari tabel.

```
> indices(T);
```

[*Roma*], [*Paris*], [*Zurich*], [*Vienna*], [*Berlin*]

Karena perintah indices mengembalikan sequence dari list, kita harus menggunakan perintah op dan map untuk membangun himpunan.

```
> map(op,{%});
```

$$\{Roma, Vienna, Berlin, Zurich, Paris\}$$

Prosedur connected mengawali kunjungan kota pertama v. Lalu connected menambahkan v ke himpunan kota yang baru saja dikunjungi dan tetangga v dari himpunan kota yang dapat dikunjungi. Selama connected masih dapat melalui suatu kota dia akan melakukannya. Jika connected tidak menemukan lagi kota yang dapat dikunjungi, dia akan menentukan apakah semua kota sudah dilalui.

```
> connected:=proc(T::table)
>   local canvisit,seen,v,V;
>   V:=map(op,{indices(T)});
>   seen:={};
>   canvisit:={V[1]};
>   while canvisit<>{} do
>     v:=canvisit[1];
>     seen:=seen union {v};
>     canvisit:=(canvisit union T[v]) minus seen;
>   end do;
>   evalb(seen=V);
> end proc:
```

Jalankan prosedurnya.

```
> connected(T);
```

$$true$$

Kita dapat menambahkan kota Montreal, Toronto, dan Waterloo, dan jalan yang menghubungkan di antaranya.

```
> T[Waterloo]:={Toronto};
```

$$T_{Waterloo} := \{Toronto\}$$

```
> T[Toronto]:={Waterloo,Montreal};
```

$$T_{Toronto} := \{ Waterloo, Montreal \}$$

```
> T[Montreal]:={Toronto};
```

$$T_{Montreal} := \{ Toronto \}$$

Sekarang kita tidak bisa lagi melakukan perjalanan dari Paris ke Waterloo, misalnya.

```
> connected(T);
```

$$false$$