Studi Kasus: Multi-List

IF2110/IF2111 – Algoritma dan Struktur Data
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

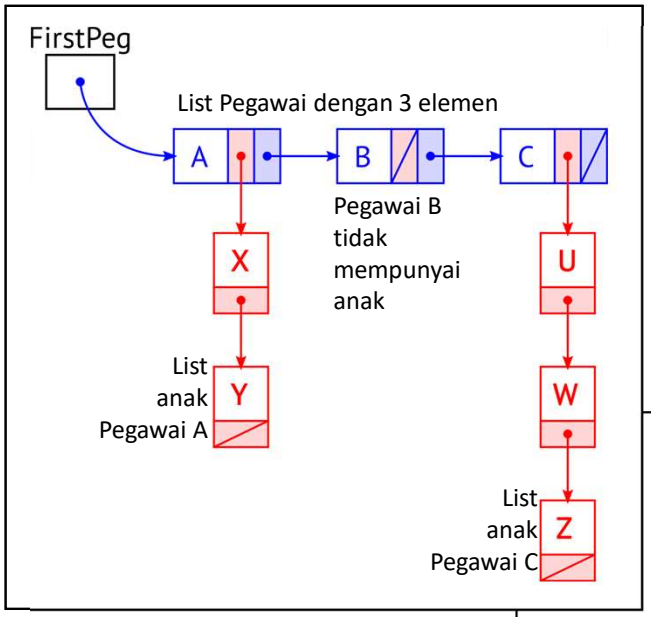
Deskripsi Persoalan

Kita harus mengelola sekumpulan pegawai, dan untuk setiap pegawai selain **informasi mengenai dirinya,** kita juga harus menyimpan **informasi tentang anak-anaknya** (jika ada).

Jika informasi tersebut harus direpresentasikan dalam struktur data internal, maka kita mempunyai list dari pegawai, dan juga list dari anak-anak pegawai.

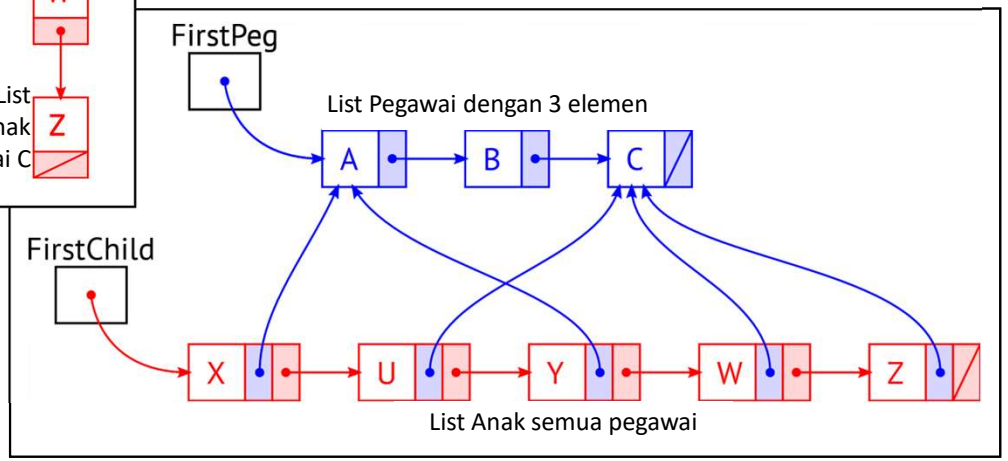
Informasi pegawai: nopeg, nama, jabatan, gaji
Informasi anak: nama, tanggal lahir

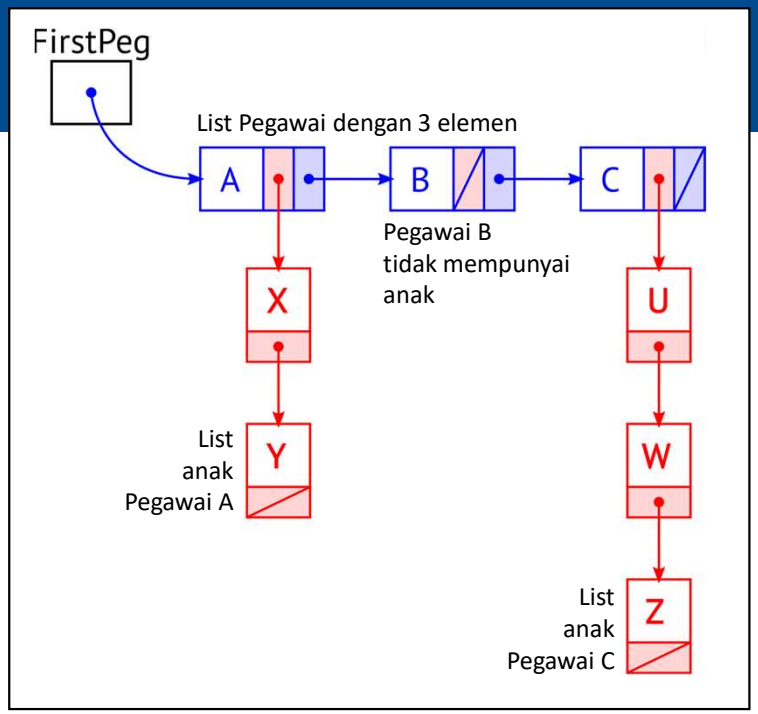
Alternatif Struktur Data



Pegawai: A, B, C, ...
Anak: A: X,Y
B: -
C: U,W,Z

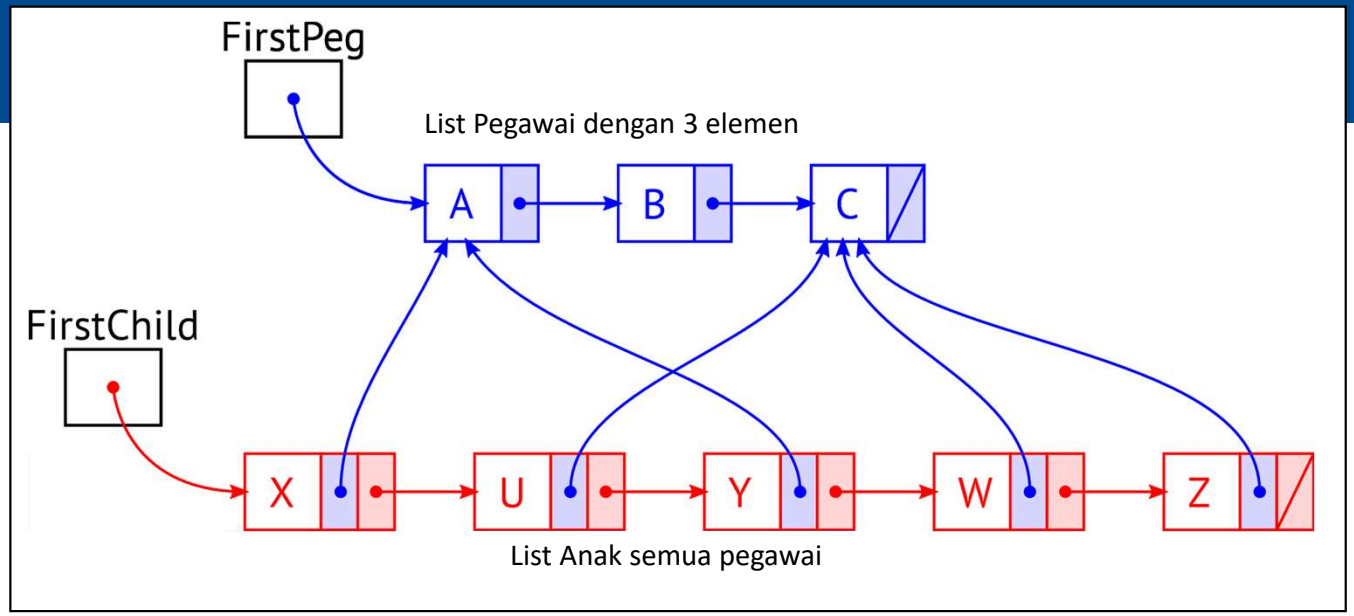
Mana yang lebih baik ?





KAMUS

```
type  AdrPeg: ... { type terdefinisi, alamat sebuah elemen list pegawai }
type AdrAnak: ... { type terdefinisi, alamat sebuah elemen list anak }
type Pegawai: < nip: integer, nama: string, jabatan: string,
                qajiPokok: real, firstAnak: AdrAnak, nextPeg: AdrPeg >
type    Anak: < nama: string, tglLahir: integer, nextAnak: AdrAnak >
type ListPeg: AdrPeg
FirstPeg: ListPeg
```




```
KAMUS
type   AdrPeg: ... { type terdefinisi, alamat sebuah elemen list pegawai }
type  AdrAnak: ... { type terdefinisi, alamat sebuah elemen list anak }
type  Pegawai: < nip: integer, nama: string, jabatan: string,
                 gajiPokok: real, nextPeg: AdrPeg >
type     Anak: < nama: string, tglLahir: integer, nextAnak: AdrAnak,
                 father: AdrPeg>
type  ListPeg: AdrPeg
type ListAnak: AdrAnak
FirstPeg: ListPeg
FirstAnak: ListAnak
```

Fitur Program

1. Daftar pegawai, dan untuk setiap pegawai harus dibuat juga nama anak-anaknya (jika ada).
2. Daftar anak-anak yang umurnya kurang dari 18 tahun (untuk keperluan tunjangan).
3. Daftar pegawai yang anaknya lebih dari 3 (keperluan KB).
4. Diketahui nama seorang anak, harus dituliskan nama orang tuanya.
5. Mendaftarkan seorang anak yang baru lahir ke dalam list anak, jika diberikan tanggal lahir dan nama anaknya, dan NIP orang tuanya.

Alternatif 1: hal 183–187

Alternatif 2: hal 188–191

1. Daftar Nama Pegawai & Anaknya

Alternatif-1:

```
Loop list pegawai
    output nama-pegawai
    Loop list anak dari pegawai
        output nama-anak
```

Alternatif-2:

```
Loop list pegawai
    output nama-pegawai
    Loop list anak
        if father(anak)=pegawai then
            output nama-anak
```

Kedua alternatif:

Perlu penanganan kasus kosong: "List kosong, tidak ada pegawai"

Perlu penanganan kasus kosong: "Pegawai ybs. tidak mempunyai anak"

Alternatif 1 (hal. 184)

```
procedure ListPegLengkap(input FirstPeg: ListPeg)
KAMUS LOKAL
    PtrPeg: AdrPeg   { address untuk traversal, @ sebuah elemen list pegawai }
    PtrAnak: AdrAnak { address untuk traversal, @ sebuah elemen list anak }
ALGORITMA
    { Traversal pegawai:skema pemrosesan sekuensial dg penanganan kasus kosong
      Untuk setiap pegawai, traversal list anak untuk dituliskan namanya }
    if (FirstPeg = NIL) then
        output("List kosong, tidak ada pegawai")
    else { Minimal 1 Pegawai }
        PtrPeg ← FirstPeg                    { First Pegawai }
        repeat
            output(Nama(PtrPeg))
            { Traversal Anak }
            PtrAnak ← FirstAnak(PtrPeg)        { First Anak }
            if (PtrAnak = NIL) then
                output("Pegawai ybs. tidak mempunyai anak")
            else
                repeat
                    output(Nama(PtrAnak))    { Proses anak }
                    PtrAnak ← NextAnak(PtrAnak) { Next Anak }
                until (PtrAnak = NIL)
            PtrPeg ← NextPeg(PtrPeg)         { Next Pegawai }
        until (PtrPeg = NIL)
```

Alternatif 2 (hal. 189)

```
procedure ListPegLengkap(input FirstPeg: ListPeg, input FirstAnak: ListAnak)
KAMUS LOKAL
    PtrPeg: AdrPeg { address untuk traversal, @ sebuah elemen list pegawai }
    PtrAnak: AdrAnak { address untuk traversal, @ sebuah elemen list anak }
    JmlAnak: integer
ALGORITMA
    { Traversal pegawai }
    if (FirstPeg = NIL) then output("List kosong, tidak ada pegawai")
    else
        PtrPeg ← FirstPeg                    { First Pegawai }
        repeat
            output (Nama(PtrPeg))
            { Traversal Anak }
            PtrAnak ← FirstAnak                 { First Anak }
            JmlAnak ← 0
            while (PtrAnak ≠ NIL) do
                if (Father(PtrAnak) = PtrPeg) then { Proses }
                    output(Nama(PtrAnak));
                    JmlAnak ← JmlAnak+1
                PtrAnak ← NextAnak(PtrAnak)     { Next Anak }
            { PtrAnak = NIL }
            if JmlAnak=0 then output ("Pegawai tidak/belum mempunyai anak")
            PtrPeg ← NextPeg(PtrPeg)          { Next Pegawai }
        until (PtrPeg = NIL)
```

Tw1

Slide 9

Tw1 Saya tambah, karena jika tidak ada maka pesan "Pegawai tidak/belum mempunyai anak" akan selalu muncul.
Tricya widagdo; 21/11/2017

2. Daftar Anak < 18 tahun (Tunj.)

Alternatif-1:

```
Loop list pegawai
    output nama-pegawai
    Loop list anak dari pegawai
        if umur(anak)<18 then
            output nama-anak
```

Alternatif-2:

```
Loop list anak
    if umur(anak)<18 then
        output nama-anak, nama-father(anak)
Catatan: tidak perlu traversal list pegawai
```

Kedua alternatif:

Perlu penanganan kasus kosong

Alternatif 1 (hal. 185)

```
procedure ListTunjAnak(input FirstPeg: ListPeg)
{ deklarasi variabel tidak ditulis untuk menghemat tempat }
ALGORITMA
    { Trav.list Pegawai, skema sekuensial dg penanganan kasus kosong.
      Untuk setiap pegawai, traversal anaknya }
    if (FirstPeg = NIL) then output("List kosong, tidak ada pegawai")
    else { Minimal ada satu pegawai }
        PtrPeg ← FirstPeg { First Pegawai }
        repeat
            output(Nama(PtrPeg))
            { Traversal Anak }
            PtrAnak ← FirstAnak(PtrPeg) { First anak }
            if (PtrAnak = NIL) then
                output("Pegawai ybs tidak mempunyai anak")
            else { Minimal ada 1 anak }
                counter ← 0
                repeat
                    UmurAnak ← Umur(TglLahir(PtrAnak)) { Proses }
                    if UmurAnak < 18 then
                        output(Nama(PtrAnak),UmurAnak)
                        counter ← counter + 1
                    PtrAnak ← NextAnak(PtrAnak) { Next Anak }
                until (PtrAnak=NIL)
                if (counter=0) then output("Tidak ada anak pegawai berumur < 18")
            PtrPeg ← NextPeg(PtrPeg) { Next Pegawai }
        until (PtrPeg=NIL)
```

Alternatif 2 (hal. 189)

```
procedure ListTunjAnak (input FirstAnak: ListAnak)
KAMUS LOKAL
    PtrAnak: AdrAnak { address utk traversal,@sbh elemen list anak }
    UmurAnak: integer { umur anak pegawai }
    function Umur (TglLahir: integer) → integer
    { Fungsi yg mengirim umur dgn rumus: tgl hari ini dr sistem
      dikurangi TglLahir }
ALGORITMA
    { Trav.list anak,skema proses sekuensial dg penanganan kasus kosg }
    { Untuk setiap anak periksa umurnya }
    if (FirstAnak = NIL) then
        output("List Anak kosong, tidak ada anak")
    else
        PtrAnak ← FirstAnak { First Anak }
        repeat
            UmurAnak ← Umur(TglLahir(PtrAnak)) { Proses }
            if UmurAnak < 18 then
                output(Nama(Father(PtrAnak)))
                output(Nama(PtrAnak), UmurAnak)
            PtrAnak ← NextAnak(PtrAnak) { Next Anak }
        until (PtrAnak = NIL)
```

3. Daftar pegawai dg anak>3 (KB).

Alternatif-1:

```
Loop list pegawai
    Hitung anak pegawai dgn loop list anaknya
    if jumlah-anak>3 then
        output nama-pegawai, status anak>3
```

Alternatif-2 (pola ListPegLengkap):

```
Loop list pegawai
    Hitung anak pegawai dgn loop list anak
    if jumlah-anak>3 then
        output nama-pegawai, status-anak>3
```

Kedua alternatif:

Perlu penanganan kasus kosong

Alternatif 1 (hal. 186)

```
procedure ListPegNonKB(input FirstPeg: ListPeg)
{ I.S. List FirstPeg terdefinisi, mungkin kosong }
{ F.S. Semua pegawai yg anaknya > 3 orang ditulis informasinya }
KAMUS LOKAL
    PtrPeg: AdrPeg   { address utk traversal, @ elemen list pegawai }
    PtrAnak: AdrAnak { address utk traversal, @ elemen list anak }
    JumlahAnak: integer { banyaknya anak pegawai }
ALGORITMA
    { Traversal pegawai }
    if (FirstPeg = NIL) then
        output("List kosong, tidak ada pegawai")
    else  { minimal ada satu pegawai }
        PtrPeg ← FirstPeg                { First-Pegawai }
        repeat
            { Traversal Anak }
            JumlahAnak ← 0                { Inisialisasi }
            PtrAnak ← FirstAnak(PtrPeg)     { First Anak }
            while (PtrAnak ≠ NIL) do
                JumlahAnak ← JumlahAnak + 1     { Proses }
                PtrAnak ← NextAnak(PtrAnak)  { Next Anak }
            if (JumlahAnak > 3 ) then
                output(Nama(PtrPeg), " mempunyai anak > 3")
            PtrPeg ← NextPeg(PtrPeg)      { Next Pegawai }
        until (PtrPeg=NIL)
```

Alternatif 2 (hal. 190)

```
procedure ListPegNonKB(input FirstPeg: ListPeg, input FirstAnak: ListAnak)
{ I.S. List First Peg terdefinisi, mungkin kosong }
{ F.S. Semua pegawai yang anaknya > 3 orang ditulis informasinya }
KAMUS LOKAL
    PtrPeg: AdrPeg   { address utk traversal, @ elemen list pegawai }
    PtrAnak: AdrAnak { address utk traversal, @ elemen list anak }
    JumlahAnak: integer { banyaknya anak pegawai }
ALGORITMA
{ Traversal list Pegawai: skema sekuensial dg penanganan kasus kosong }
{ Untuk setiap pegawai, traversal list anak utk mencacah jumlah anaknya.
  Jika jumlah anak > tiga maka nama pegawai ditulis }
    { Traversal pegawai }
    if (FirstPeg = NIL) then output("List pegawai kosong")
    else
        PtrPeg ← FirstPeg                          { First Pegawai }
        repeat                                            { Proses }
            JumlahAnak ← 0
            { Traversal Anak }
            PtrAnak ← FirstAnak                      { First Anak }
            while (PtrAnak ≠ NIL) do
                 if (Father(PtrAnak) = PtrPeg) then    { Proses Anak }
                    JumlahAnak ← JumlahAnak + 1
                PtrAnak  ← NextAnak(PtrAnak)           { Next Anak }
            { PtrAnak = NIL) }
            if (JumlahAnak > 3) then output(Nama(PtrPeg), " mempunyai anak lebih dari 3")
            PtrPeg ← NextPeg(PtrPeg)                { Next Pegawai }
        until (PtrPeg = NIL)
    { semua elemen list pegawai selesai diproses }
```

4. Search Nama Orang Tua dari Anak

Alternatif-1:

```
Loop list pegawai
    Search NamaAnak dg loop list anak dr pegawai
        if NamaAnak ketemu then
            output nama-pegawai
            keluar dr loop anak dan loop pegawai
```

Alternatif-2:

```
Search NamaAnak dgn loop list anak
    if NamaAnak ketemu then
        output nama-pegawai
        keluar dr loop list anak
```

Alternatif 1 (hal. 186)

```
procedure OrTuAnak(input FirstPeg: ListPeg, input NamaAnak: string)
{ I.S. List Pegawai terdefinisi }
{ F.S. Jika ada anak yg bernama sesuai dg NamaAnak, nama Pegawai ditulis.
       Jika tidak ada NamaAnak, tidak menuliskan apa-apa }
KAMUS LOKAL
    PtrPeg: AdrPeg   { address utk traversal, @ elemen list pegawai }
    PtrAnak: AdrAnak { address utk traversal, @ elemen list anak }
    Found: boolean   { hasil pencarian orangtua anak }
ALGORITMA
    { Search }
    Found ← false
    PtrPeg ← FirstPeg
    while (PtrPeg ≠ NIL) and (not Found) do
        { Search anak dengan NamaAnak yang diberikan pada list anak }
        PtrAnak ← FirstAnak(PtrPeg)
        while (PtrAnak ≠ NIL) and (not Found) do
            if (Nama(PtrAnak) = NamaAnak) then Found ← true
            else PtrAnak ← NextAnak(PtrAnak)
        { PtrAnak = NIL or Found }
        if (not Found) then { explore pegawai yg berikutnya }
            PtrPeg ← NextPeg(PtrPeg)
    { PtrPeg = NIL or Found }
    if (Found) then output(Nama(PtrPeg))
```

Alternatif 2 (hal. 190)

```
procedure OrTuAnak(input FirstAnak: ListAnak, input NamaAnak: string)
{ Alternatif Kedua }
{ I.S. List Pegawai terdefinisi }
{ F.S. Jika ada anak yg bernama NamaAnak, nama Pegawai ditulis.
       Jika tidak ada NamaAnak, tidak menuliskan apa-apa. }
KAMUS LOKAL
    PtrPeg: AdrPeg   { address utk traversal, @ elemen list pegawai }
    PtrAnak: AdrAnak { address utk traversal, @ elemen list anak }
    Found: boolean   { hasil pencarian orangtua anak }
ALGORITMA
{ Search pada list Anak berdasarkan nama. Jika ketemu, akses Bapaknya }
    PtrAnak ← FirstAnak; Found ← false
    while (PtrAnak ≠ NIL) and (not Found) do
        if (Nama(PtrAnak) = NamaAnak) then
            Found ← true
        else
            PtrAnak ← NextAnak(PtrAnak)
    { PtrAnak = NIL or Found }
    if (Found) then
        output(Nama(Father(PtrAnak)))
```

5. Mendaftarkan anak yang baru lahir

Alternatif-1:

```
Search elemen dgn NIPPeg pd list pegawai
If ketemu then
    Insert first data-anak pd list anaknya
```

Alternatif-2:

```
Search elemen dgn NIPPeg pd list pegawai
If ketemu then
    Insert first data-anak pada list anak
```

Alternatif 1 (hal. 187)

```
procedure AddAnak (input/output FirstPeg: ListPeg, input NIPPeg: string,
                   input NamaAnak: string, input TglLahirAnak: integer)
{ Mendaftar seorang anak yang baru lahir, insert selalu pada awal list }
{ I.S. List Pegawai terdefinisi }
{ F.S. Jika pegawai dgn NIP=NIPPeg ada, alokasi anak.
       Jika berhasil insert seorang anak sebagai elemen pertama list anak.
       Jika alokasi gagal atau NIPPeg tidak ada, hanya menulis pesan }
KAMUS LOKAL
    PtrPeg: AdrPeg    { address utk traversal, @ elemen list pegawai }
    PtrAnak: AdrAnak  { address utk traversal, @ elemen list anak }
    FoundNIP: boolean { hasil pencarian NIP pegawai sebelum insert anak }
ALGORITMA
    { Search Pegawai }
    FoundNIP ← false
    PtrPeg ← FirstPeg
    while (PtrPeg ≠ NIL) and (not FoundNIP) do
        { search Pegawai dengan NIP yang diberikan }
        if (NIP(PtrPeg) = NIPPeg) then
            FoundNIP ← true
        else
            PtrPeg ← NextPeg(PtrPeg)
    ...
```

Alternatif 1 (lanjutan)

```
...
{ Akhir search pegawai: PtrPeg=NIL or FoundNIP }
if (FoundNIP) then      { add anak }
    PtrAnak ← Alokasi(NamaAnak , TglLahirAnak)
    if (PtrAnak ≠ NIL) then
        NextAnak(PtrAnak) ← FirstAnak(PtrPeg)
        FirstAnak(PtrPeg) ← PtrAnak
    else { Alokasi gagal, tidak insert, hanya pesan }
        output("Alokasi gagal")
else  { NIPPeg tidak ada, error }
    output("Pegawai tidak ada dalam list")
```

Alternatif 2 (hal. 191)

```
procedure AddAnak (input/output FirstPeg:ListPeg, input NIPPeg:integer,
                   input/output FirstAnak:ListAnak,
                   input NamaAnak:string, input TglLahirAnak:integer)
{ Mendaftar seorang anak yang baru lahir,
  Cari Pegawai, insert anak pada list anak selalu pada awal list }
{ I.S. List Pegawai terdefinisi }
{ F.S. List FirstPeg terdefinisi.
       Jika pegawai dengan NIP=NIPPeg ada, alokasi anak.
       Jika alokasi berhasil, insert seorg anak sbg elemen pertama list
       anak, tentukan Bapak.
       Jika alokasi gagal atau NIPPeg tidak ada, hanya menulis pesan. }
KAMUS LOKAL
    PtrPeg: AdrPeg    { address utk traversal, @ elemen list pegawai }
    PtrAnak: AdrAnak  { address utk traversal, @ elemen list anak }
    FoundNIP: boolean { hasil pencarian NIP pegawai sblm insert anak }
...
```

Alternatif 2 (lanjutan)

```
ALGORITMA
    { Search Pegawai dgn NIP yg diberikan: skema search dgn boolean }
    FoundNIP ← false
    PtrPeg ← FirstPeg
    while (PtrPeg ≠ NIL) and (not FoundNIP) do
        if (NIP(PtrPeg) = NIPPeg) then
            FoundNIP ← true
        else
            PtrPeg ← NextPeg(PtrPeg)
    { PtrPeg = NIL or FoundNIP }
    { Akhir search pegawai: PtrPeg=NIL or FoundNIP }
    if (FoundNIP) then      { Insert anak }
        PtrAnak ← Alokasi(NamaAnak, TglLahir)
        if (PtrAnak ≠ NIL) then
            Father(PtrAnak) ← Ptrpeg { Tentukan Bapaknya }
            { Insert Anak, kasus kosong ditangani dg cara yg sama dengan
              kasus tidak kosong }
            NextAnak(PtrAnak) ← FirstAnak
            FirstAnak ← PtrAnak
        else { Alokasi gagal:tidak melakukan apa-apa, hanya pesan }
            output("Alokasi gagal")
    else { NIPPeg tidak ada, error }
        output("Pegawai tidak ada dalam list")
```

Studi Kasus: Representasi Relasi M-N

IF2110/IF2111 – Algoritma dan Struktur Data
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

Deskripsi Persoalan

Sistem harus mengelola sekumpulan dosen dan sekumpulan mata kuliah serta data pengajaran (mata kuliah MK diajar oleh dosen D). Seorang **dosen dapat mengajar lebih dari satu mata kuliah** dan **sebuah mata kuliah dapat diajar oleh lebih dari satu dosen**.

$\rightarrow$ relasi antara Dosen dan Mata kuliah adalah M-N

Jika informasi tersebut harus direpresentasikan dalam struktur data internal, akan terdapat list dari dosen dan juga list dari mata kuliah.

Masalahnya adalah bagaimana merepresentasikan daftar pengajaran (relasi antara dosen dan mata kuliah)?

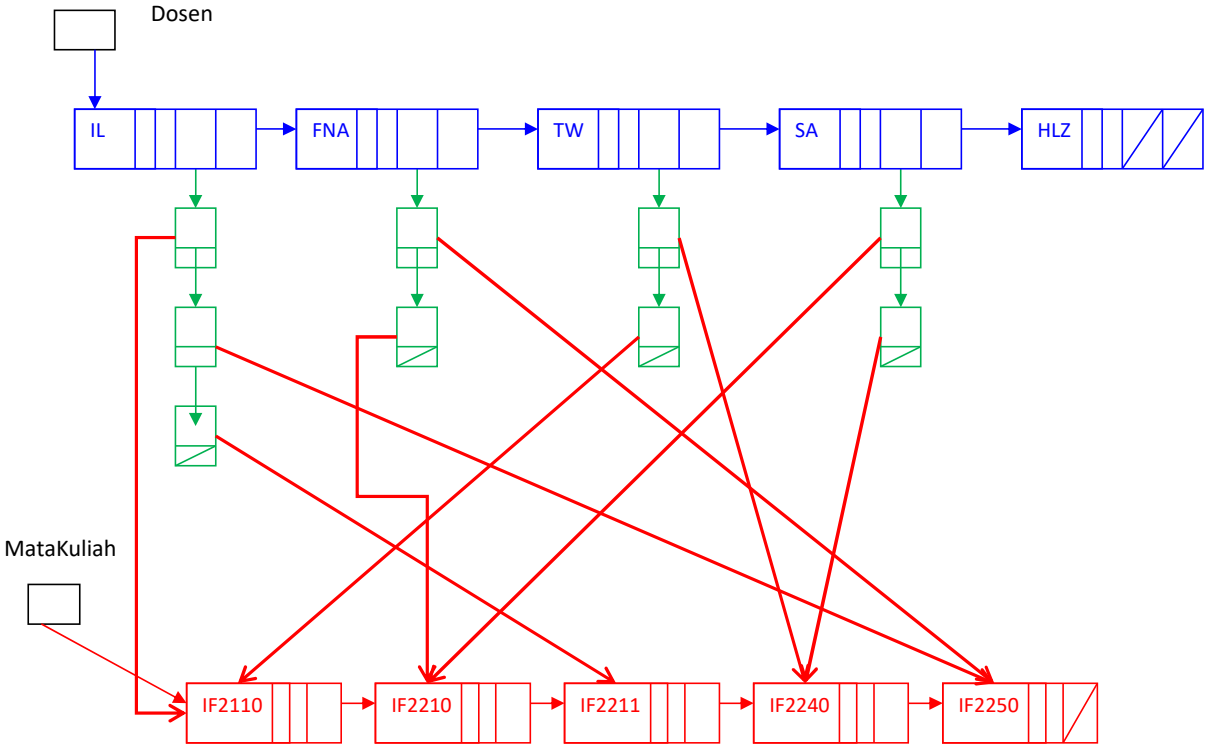
Alternatif Representasi Relasi M-N

1. **Relasi “Mengajar”**
2. **Relasi “Diajar Oleh”**
3. **Relasi hubungan “Dosen_MataKuliah”**

Alternatif 1: Relasi sebagai “Mengajar”

Relasi “Mengajar”

Untuk merepresentasi relasi dari sudut pandang setiap Dosen: setiap Dosen mempunyai list MataKuliah yang diajarnya. Hubungan ini 1-N dipandang dari list Dosen. Dengan representasi ini, setiap elemen list Dosen akan mempunyai list of MataKuliah yang diajarnya.



Alternatif 2: Relasi sebagai “Diajar Oleh”

Relasi “Diajar oleh”

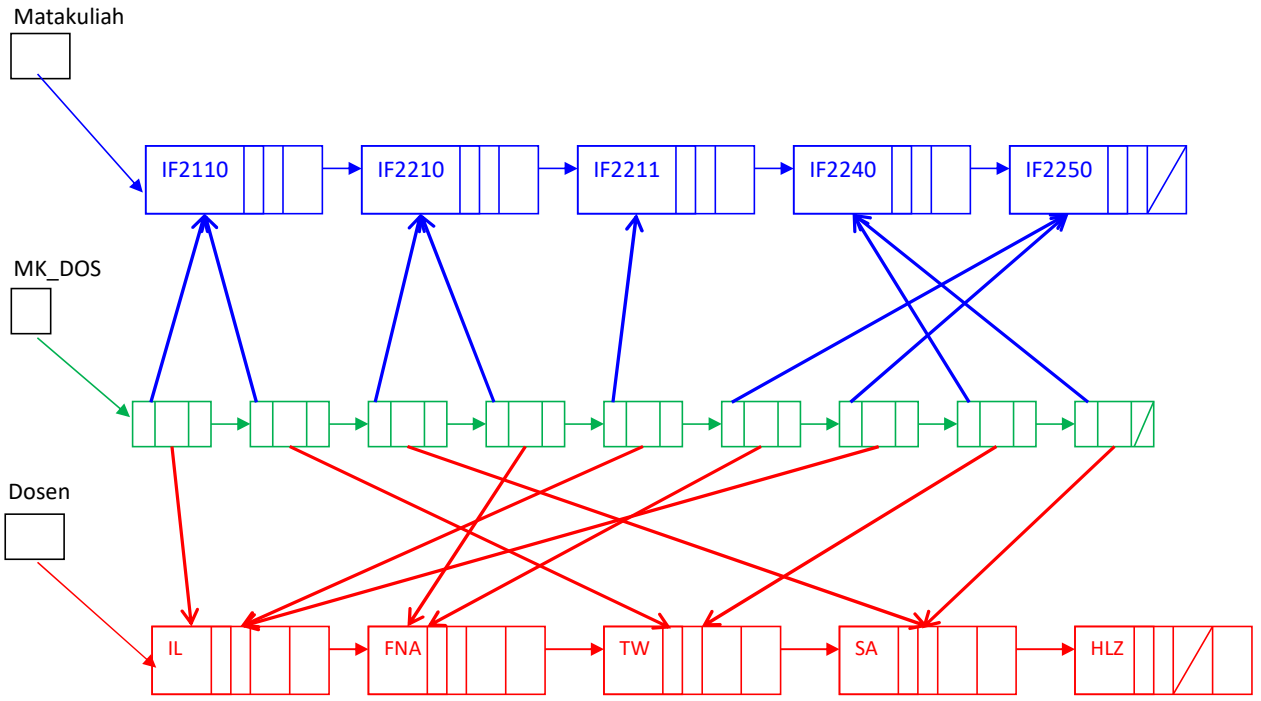
Untuk merepresentasi relasi dari sudut pandang setiap MataKuliah: sebuah MataKuliah diajarkan oleh siapa saja. Hubungan ini adalah hubungan 1-N dipandang dari list MataKuliah. Dengan representasi ini, setiap elemen list MataKuliah akan mempunyai list of Dosen pengajarnya

Menyerupai alternatif 1, hanya saja yang menjadi parent dari relasi adalah relasi Mata kuliah.

Alternatif 3: Relasi sebagai list terpisah

Relasi hubungan "Dosen_MataKuliah"

Yaitu untuk merepresentasi setiap relasi MataKuliah dan Dosen: setiap elemen relasi ⟨Dosen, MataKuliah⟩ adalah unik, maka relasi hubungan ini dikelola sebagai list yang terpisah.



Perbandingan Ketiga Alternatif

Buatlah studi perbandingan terhadap ketiga representasi tersebut terutama dari sisi efisiensi proses, jika misalnya sistem harus mampu menampilkan:

- Daftar mata kuliah yang diajar oleh dosen X
- Daftar dosen yang mengajar mata kuliah Y
- Jumlah mata kuliah yang diajar oleh setiap dosen

Penambahan Relasi Baru

Procedure **AddRel** menerima $\langle D, MK \rangle$ dan menambahkan sebuah relasi Dosen MataKuliah, dengan ketentuan:

- jika D belum ada di list Dosen maka ditambahkan lebih dulu sebagai elemen list. Demikian pula jika MK belum ada, maka sebelum menambahkan relasi, elemen list mata kuliah ditambahkan lebih dulu.
- jika D dan MK sudah ada pada list Dosen dan Matakuliah, maka $\langle D, MK \rangle$ harus belum muncul dalam list relasi (harus unik).

Studi Kasus: Pengembangan Representasi

Coba kembangkan masing-masing alternatif untuk kasus berikut ini:

- Sebuah relasi dapat terjadi antara elemen list yang sama. Misalnya mata kuliah berelasi dengan mata kuliah lain untuk menyatakan daftar mata kuliah yang menjadi prerequisite dari suatu mata kuliah.
- Sebuah himpunan objek dapat terkait dengan lebih dari satu relasi. Misalnya mata kuliah terlibat dalam relasi pengajaran dan prerequisite.

Studi Kasus: Topological Sort

IF2110/IF2111 – Algoritma dan Struktur Data
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

Deskripsi Persoalan

Diberikan urutan partial dari elemen suatu himpunan,

Dikehendaki agar elemen tersebut mempunyai keterurutan linier.

Contoh Persoalan

- Dalam suatu kurikulum, suatu mata pelajaran mempunyai prerequisit mata pelajaran lain.

  → Urutan linier adalah urutan untuk seluruh mata pelajaran dalam kurikulum

- Dalam suatu proyek, suatu pekerjaan harus dikerjakan lebih dulu dari pekerjaan lain

  → misalnya membuat pondasi harus sebelum dinding, membuat dinding harus sebelum pintu. Namun pintu dapat dikerjakan bersamaan dengan jendela. Dan sebagainya.

Contoh Persoalan (2)

- Dalam pembuatan tabel pada basis data, tabel yang di-refer oleh tabel lain harus dideklarasikan terlebih dulu. Jika suatu aplikasi terdiri dari banyak tabel, maka urutan pembuatan tabel harus sesuai dengan definisinya.

- Dalam sebuah program Pascal, pemanggilan prosedur harus sedemikian rupa, sehingga peletakan prosedur pada teks program harus seuai dengan urutan (partial) pemanggilan.

Definisi Permasalahan

Jika $X<Y$ adalah simbol untuk X "sebelum" Y, dan merupakan keterurutan *partial,*

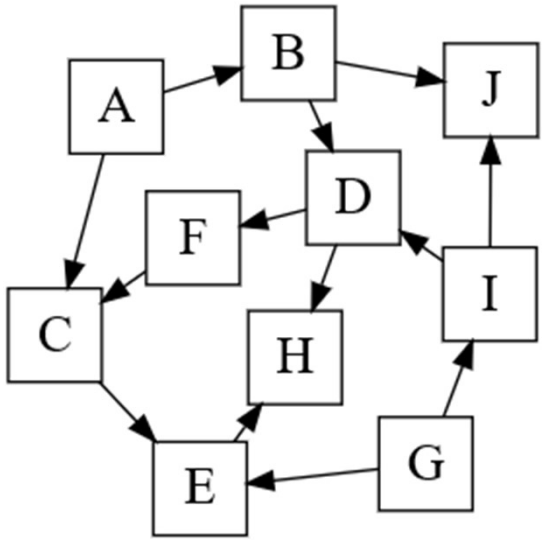
Contoh pada simpul-simpul dalam sebuah graf

```
A<B   B<D   D<F   B<J   D<H
F<C   A<C   C<E   E<H   G<E
G<I   I<D   I<J
```

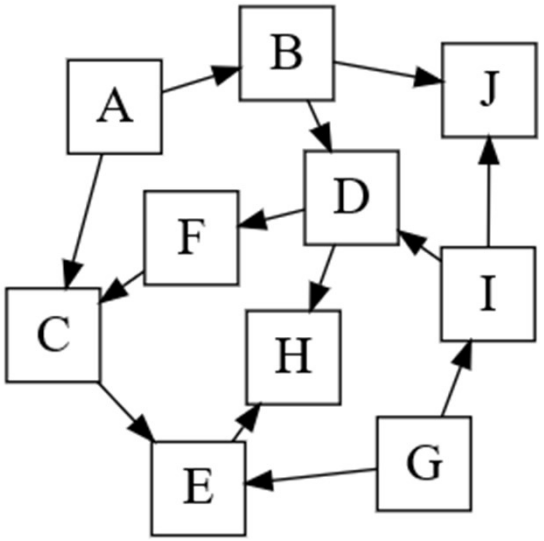
Defini Persoalan

Representasi dalam Graf
(DAG-Directed Acyclic Graph)

```
A<B   B<D   D<F   B<J   D<H
F<C   A<C   C<E   E<H   G<E
G<I   I<D   I<J
```

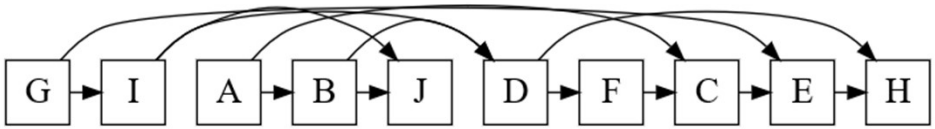


Definisi Persoalan

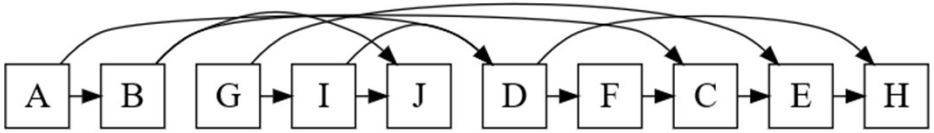


Bagaimana total/linear order? [beberapa solusi]

1. G→I→A→B→J→D→F→C→E→H



2. A→B→G→I→J→D→F→C→E→H



Ide dari Topological Sort

Andaikata item yang mempunyai keterurutan partial adalah anggota himpunan S.

1. **Pilih salah satu item yang tidak mempunyai predesesor**, misalnya X. Minimal ada satu elemen semacam ini. Jika tidak, maka akan *looping*.
2. **Hapus X dari himpunan S** (dan keterhubungan dari X ke item lainnya), dan insert ke dalam list hasil sorting.
3. Sisa himpunan S masih merupakan himpunan terurut partial, maka **proses 1 dan 2** dapat dilakukan lagi terhadap sisa dari S **hingga kosong**.

Solusi 1: MultiList

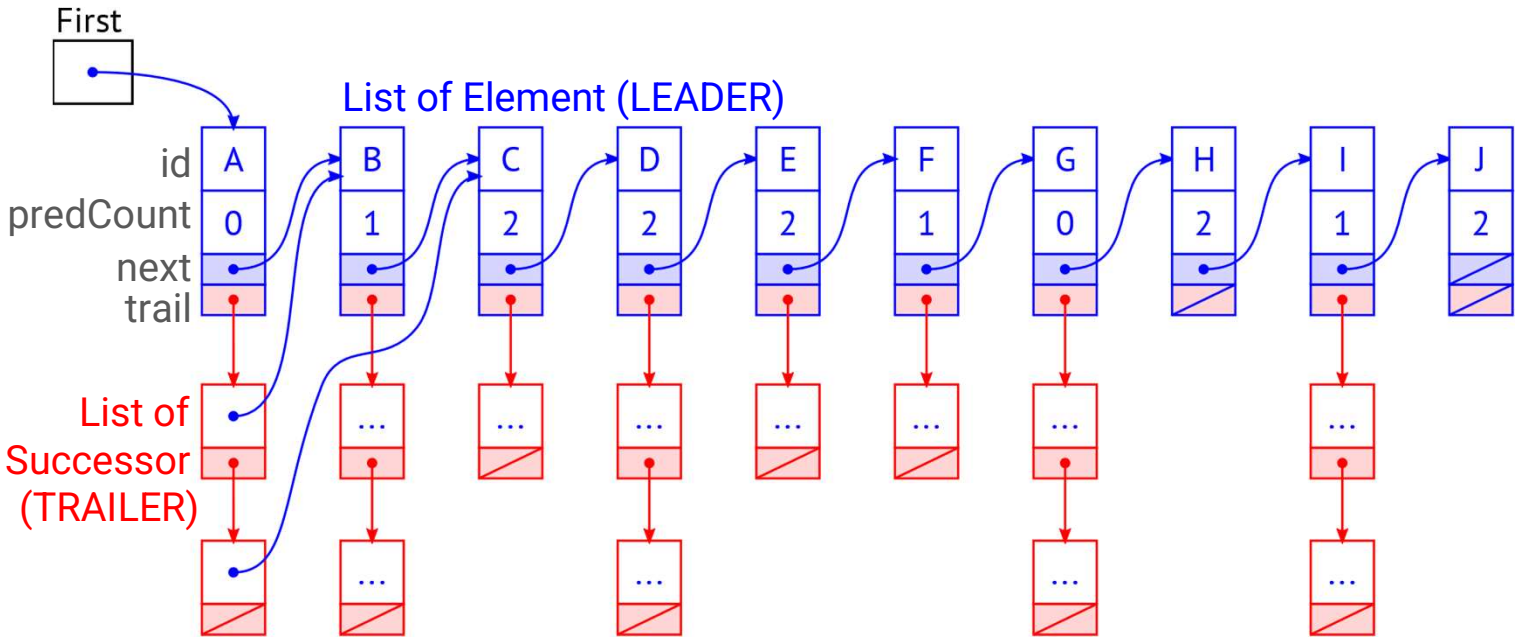
Struktur Data:

Jumlah elemen pada DAG *unbounded* →list berkait dengan representasi dinamis (pointer)

Field pada setiap elemen
- Identitas (id)
- Jumlah Predecessor (predCount)
- List to Successor (trail)

MultiList
- List of Elemen (horizontal)
  - List of Successor dari tiap Elemen (vertikal)

Struktur Data



Algoritma Pembentukan MultiList

Bentuk list Leader dan Trailer

baca pasangan nilai ($X < Y$).

temukan alamat X dan Y pada Leader

jika belum ada sisipkan sebagai elemen baru

InsertFirst alamat Y sebagai Trailer X.

Increment nilai Count (jumlah predesessor) dari Y

Algoritma Topological Sort

Search elemen dari list Leader dengan jumlah predesesor = 0

Delete elemen tersebut dan decrement *Count* dari successor dari elemen tersebut

InsertLast sebagai elemen list linier hasil pengurutan

[Lakukan hingga List Leader kosong]

Solusi 2: Pendekatan Fungsional dengan List Linear Biasa

Struktur Data:

Graf partial dinyatakan sebagai list linier dengan representasi fisik First-Last dengan *dummy*.

Elemen list terdiri dari ⟨prec, succ⟩. Contoh: sebuah elemen bernilai ⟨1,2⟩ artinya 1 adalah predesesor dari 2.

```
type Node: < prec: char,
             succ: char,
             next: address >
type ListTopo: < first: Address,
                 last: Address >
```

Algoritma Topological Sort

1. P adalah elemen pertama (First(L)), misalkan elemennya adalah ⟨X,Y⟩
2. *Search* pada sisa list (list tanpa elemen pertama), **apakah X mempunyai predesesor**.

   Jika ya, maka elemen ini harus dipertahankan sampai saatnya dapat dihapus dari list untuk dioutputkan:

   1. *Delete* P, tapi jangan didealokasi.
   2. *Insert* P sebagai Last(L) yang baru.

   Jika tidak mempunyai predesesor, maka X siap untuk dioutputkan, tetapi Y masih harus dipertanyakan. Maka langkah yang harus dilakukan:

   1. InsertLast X pada list hasil sorting, jika belum ada.
   2. *Search* **apakah Y masih ada pada sisa list**, baik sebagai Prec maupun sebagai Succ.

      Jika ya, maka Y akan dioutputkan nanti. Hapus elemen pertama yang sedang diproses dari list.

      Jika tidak muncul sama sekali, berarti Y tidak mempunyai predesesor, maka InsertLast Y pada list hasil sorting, baru hapus elemen pertama dari list.

[Lakukan hingga list kosong]

Contoh proses topological sort (1)

List input: [⟨A,B⟩, ⟨B,D⟩, ⟨D,F⟩, ⟨B,J⟩, ⟨D,H⟩, ⟨F,C⟩, ⟨A,C⟩, ⟨C,E⟩, ⟨E,H⟩, ⟨G,E⟩, ⟨G,I⟩, ⟨I,D⟩, ⟨I,J⟩]
List output: [ ]

P=⟨A,B⟩, X=A tidak memiliki predesesor, Y=B masih ada di sisa list
List output: [A]
List input: [⟨B,D⟩, ⟨D,F⟩, ⟨B,J⟩, ⟨D,H⟩, ⟨F,C⟩, ⟨A,C⟩, ⟨C,E⟩, ⟨E,H⟩, ⟨G,E⟩, ⟨G,I⟩, ⟨I,D⟩, ⟨I,J⟩]

P=⟨B,D⟩, X=B tidak memiliki predesesor, Y=D masih ada di sisa list
List output: [A,B]
List input: [⟨D,F⟩, ⟨B,J⟩, ⟨D,H⟩, ⟨F,C⟩, ⟨A,C⟩, ⟨C,E⟩, ⟨E,H⟩, ⟨G,E⟩, ⟨G,I⟩, ⟨I,D⟩, ⟨I,J⟩]

Contoh proses topological sort (2)

P=⟨D,F⟩, X=D memiliki predesesor
List output: [A,B]
List input: [⟨B,J⟩, ⟨D,H⟩, ⟨F,C⟩, ⟨A,C⟩, ⟨C,E⟩, ⟨E,H⟩, ⟨G,E⟩, ⟨G,I⟩, ⟨I,D⟩, ⟨I,J⟩, ⟨D,F⟩]

P=⟨B,J⟩, X=B tidak memiliki predesesor, Y=J masih ada di sisa list
List output: [A,B] ← tetap, karena B sudah ada
List input: [⟨D,H⟩, ⟨F,C⟩, ⟨A,C⟩, ⟨C,E⟩, ⟨E,H⟩, ⟨G,E⟩, ⟨G,I⟩, ⟨I,D⟩, ⟨I,J⟩, ⟨D,F⟩]

P=⟨D,H⟩, X=D memiliki predesesor
List output: [A,B]
List input: [⟨F,C⟩, ⟨A,C⟩, ⟨C,E⟩, ⟨E,H⟩, ⟨G,E⟩, ⟨G,I⟩, ⟨I,D⟩, ⟨I,J⟩, ⟨D,F⟩, ⟨D,H⟩]

Contoh proses topological sort (3)

P=⟨F,C⟩, X=F memiliki predesesor
List output: [A,B]
List input: [⟨A,C⟩, ⟨C,E⟩, ⟨E,H⟩, ⟨G,E⟩, ⟨G,I⟩, ⟨I,D⟩, ⟨I,J⟩, ⟨D,F⟩, ⟨D,H⟩, ⟨F,C⟩]

P=⟨A,C⟩, X=A tidak memiliki predesesor, Y=C masih ada di sisa list
List output: [A,B] ← tetap, karena A sudah ada
List input: [⟨C,E⟩, ⟨E,H⟩, ⟨G,E⟩, ⟨G,I⟩, ⟨I,D⟩, ⟨I,J⟩, ⟨D,F⟩, ⟨D,H⟩, ⟨F,C⟩]

P=⟨C,E⟩, X=C memiliki predesesor
List output: [A,B]
List input: [⟨E,H⟩, ⟨G,E⟩, ⟨G,I⟩, ⟨I,D⟩, ⟨I,J⟩, ⟨D,F⟩, ⟨D,H⟩, ⟨F,C⟩, ⟨C,E⟩]

Contoh proses topological sort (4)

P=⟨E,H⟩, X=E memiliki predesesor
List output: [A,B]
List input: [⟨G,E⟩, ⟨G,I⟩, ⟨I,D⟩, ⟨I,J⟩, ⟨D,F⟩, ⟨D,H⟩, ⟨F,C⟩, ⟨C,E⟩, ⟨E,H⟩]

P=⟨G,E⟩, X=G tidak memiliki predesesor, Y = E masih ada di sisa list
List output: [A,B,G]
List input: [⟨G,I⟩, ⟨I,D⟩, ⟨I,J⟩, ⟨D,F⟩, ⟨D,H⟩, ⟨F,C⟩, ⟨C,E⟩, ⟨E,H⟩]

P=⟨G,I⟩, X=G tidak memiliki predesesor, Y = I masih ada di sisa list
List output: [A,B,G] ← tetap, karena G sudah ada
List input: [⟨I,D⟩, ⟨I,J⟩, ⟨D,F⟩, ⟨D,H⟩, ⟨F,C⟩, ⟨C,E⟩, ⟨E,H⟩]

Contoh proses topological sort (5)

P=⟨I,D⟩, X=I tidak memiliki predesesor, Y = D masih ada di sisa list
List output: [A,B,G,I]
List input: [⟨I,J⟩, ⟨D,F⟩, ⟨D,H⟩, ⟨F,C⟩, ⟨C,E⟩, ⟨E,H⟩]

P=⟨I,J⟩, X=I tidak memiliki predesesor, Y = J tidak ada di sisa list
List output: [A,B,G,I,J]
List input: [⟨D,F⟩, ⟨D,H⟩, ⟨F,C⟩, ⟨C,E⟩, ⟨E,H⟩]

P=⟨D,F⟩, X=D tidak memiliki predesesor, Y = F masih ada di sisa list
List output: [A,B,G,I,J,D]
List input: [⟨D,H⟩, ⟨F,C⟩, ⟨C,E⟩, ⟨E,H⟩]

Contoh proses topological sort (6)

P=⟨D,H⟩, X=D tidak memiliki predesesor, Y = H masih ada di sisa list
List output: [A,B,G,I,J,D] ← tetap, karena D sudah ada
List input: [⟨F,C⟩, ⟨C,E⟩, ⟨E,H⟩]

P=⟨F,C⟩, X=F tidak memiliki predesesor, Y = C masih ada di sisa list
List output: [A,B,G,I,J,D,F]
List input: [⟨C,E⟩, ⟨E,H⟩]

P=⟨C,E⟩, X=C tidak memiliki predesesor, Y = E masih ada di sisa list
List output: [A,B,G,I,J,D,F,C]
List input: [⟨E,H⟩]

Contoh proses topological sort (7)

P=⟨E,H⟩, X=E tidak memiliki predesesor, Y = H tidak ada di sisa list
List output: [A,B,G,I,J,D,F,C,E,H]
List input: [ ]

Fungsi Search

Untuk solusi ini diperlukan primitif *search* apakah sebuah nilai X muncul sebagai Prec(P) atau Succ(P) pada "sisa list" (list tanpa elemen pertama). Untuk efisiensi proses, kedua macam *search* digabungkan dengan spesifikasi:

```
function searchTopo (l: ListTopo, x: char) → <boolean, boolean>
{ boolean pertama akan bernilai true jika x muncul sebagai
  prec pada salah satu elemen sisa list l
  boolean kedua akan bernilai true jika x muncul sebagai
  succ pada salah satu elemen sisa list l }
```