Studi Kasus: Polinom Representasi Kontigu

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

Deskripsi Persoalan

Sebuah polinom berderajat n didefinisikan sebagai fungsi $P(x)$ berikut:

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + ... + a_2 x^2 + a_1 x^1 + a_0$$

Perhatikan bahwa untuk mempermudah pemrosesan, definisi $P(x)$ tersebut berbeda dengan definisi polinom pada matematika, yang biasanya diberikan sebagai berikut:

$$P(x) = a_0 x^n + a_1 x^{n-1} + a_2 x^{n-2} + ... + a_{n-2} x^2 + a_{n-1} x^1 + a_n$$

Contoh Polinom:

$$P1(x) = 4x^5 + 2x^4 + 7x^2 + 10$$

$$P2(x) = 23x^{100} + 9x^9 + 2x^7 + 4x^5 + 9x^9 + 2x^4 + 3x^2 + 1$$

$$P3(x) = 10$$

$$P4(x) = 3x^2 + 2x + 8$$

$$P5(x) = x^{1000}$$

Proses Dasar Polinom (1)

1. Membentuk sebuah polinom *P* dari pasangan harga yang dibaca dari masukan, data yang dibaca adalah pasangan harga:

   (*) ⟨ degree: integer, coefficient: integer ⟩

   (1) ⟨ −999, 0 ⟩

2. Menuliskan sebuah polinom *P* terurut mulai dari suku terbesar sampai terkecil

3. Menjumlahkan dua buah polinom *P1* dan *P2* dan menyimpan hasilnya pada *P3*, $P3 \neq P1$ dan $P3 \neq P2$; pada akhir proses $P3 = P1 + P2$.

Proses Dasar Polinom (2)

4. Mengurangi dua buah polinom *P1* dan *P2* dan menyimpan hasilnya pada *P3*, *P3* ≠ *P1* dan *P3* ≠ *P2*, pada akhir proses *P3* = *P1* – *P2*.

5. Membuat turunan dari sebuah polinom *P* dan menyimpan hasilnya pada *P'*, *P'* ≠ *P1*, pada akhir proses *P'* adalah turunan *P1*.

Pemilihan Struktur Data

Secara lojik sebuah suku polinom direpresentasi oleh sepasang harga integer:

```
< degree: integer, coefficient: integer >
```

P(*x*) adalah kumpulan pasangan harga tersebut yang diidentifikasi oleh namanya dan merupakan sebuah list linier dengan elemen bertipe **Suku**, yaitu pasangan harga `< degree: integer, coefficient: integer >`, dengan harga **degree** yang maksimum sebagai derajat polinom.

P(*x*) dapat direpresentasi secara **KONTIGU** atau **BERKAIT**.

Representasi Kontigu

Polinom direpresentasi dalam sebuah tabel P

- setiap elemennya berisi koefisien dari polinom
- **Derajat polinom secara implisit adalah indeks dari tabel**

```
KAMUS
{ Definisi sebuah polinom p adalah }
constant nMax: integer = 100
type  Polinom: < degree: integer≥0,
                 arrSuku: array [0..nMax] of integer >
p1, p2, p3: Polinom
{ p.arrSuku[i] adalah koefisien dari suku ke-i dari sebuah polinom }
```

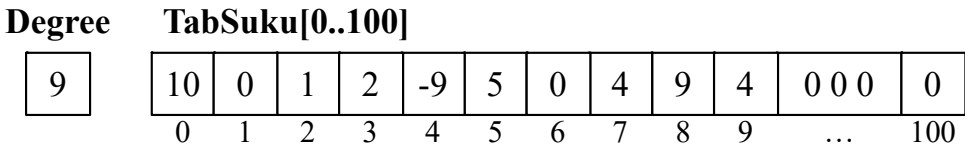
Membentuk sebuah polinom

Membentuk sebuah polinom dari pasangan harga yang dibaca dari alat masukan:

- Setiap kali memasukkan data, yang dimasukkan adalah pasangan
  (*) ⟨ degree: integer, coefficient: integer ⟩
- Akhir pemasukan adalah pasangan harga yang diketikkan bernilai ⟨ −999, 0 ⟩

Prosesnya adalah **proses sekuensial dengan mark** untuk membaca dari masukan dan menyimpannya dalam tabel polinom.

Contoh: Jika dibaca P1: ⟨ 0,10 ⟩, ⟨ 4,−9 ⟩, ⟨ 5,5 ⟩, ⟨ 8,9 ⟩, ⟨ 2,1 ⟩, ⟨ 3,2 ⟩, ⟨ 9,4 ⟩, ⟨ 7,4 ⟩, ⟨ −999,0 ⟩ maka tabel polinom yang dibentuk adalah polinom berderajat 9.



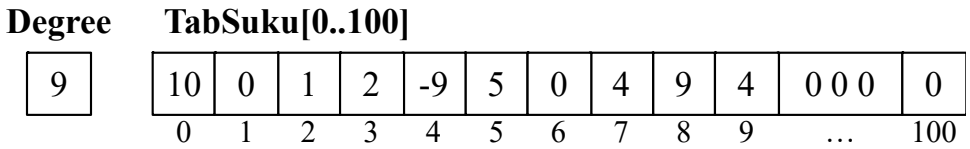
Polinom “kosong”, yaitu polinom dengan Degree = −999

Menuliskan sebuah polinom

Prosesnya adalah **proses sekuensial tanpa mark**, traversal untuk *i* [Degree..0].

Harga suku dituliskan hanya jika koefisiennya tidak nol.

Contoh:



I	P(I)
9	4
8	9
7	4
5	5
4	-9
3	2
2	1
0	10

Menjumlahkan dua buah polinom

Menjumlahkan dua buah polinom P1 dan P2 dan menyimpan hasilnya pada P3, P3 ≠ P1 dan P3 ≠ P2:

- menjumlahkan suku P1 dan P2 yang berderajat sama, menjadi suku berderajat tersebut pada P3
- prosesnya adalah mencari derajat tertinggi dari P1 dan P2, kemudian pemrosesan sekuensial untuk setiap pasangan suku P1 dan P2

Penjumlahan memungkinkan hasil berupa polinom kosong atau polinom yang derajatnya “turun”

Menjumlahkan dua buah polinom (2)

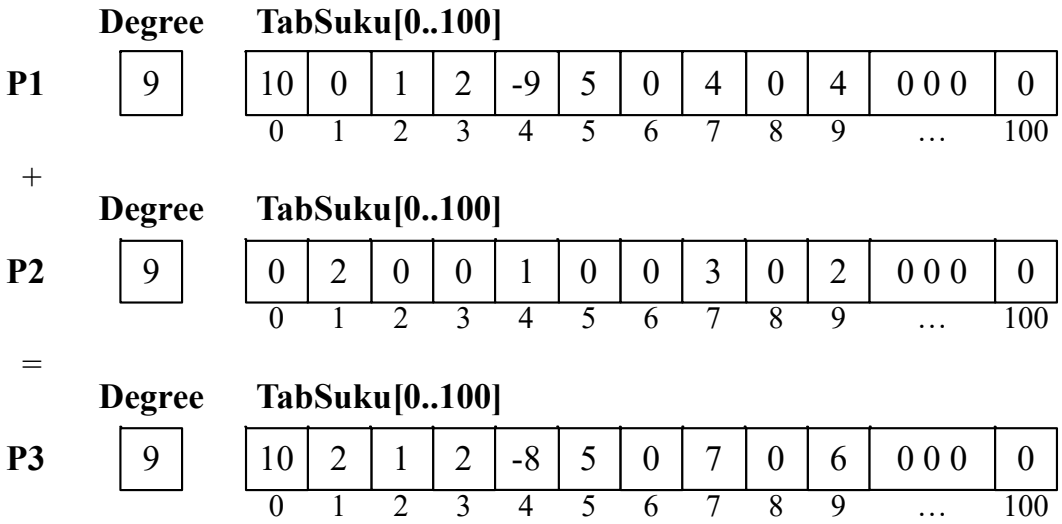
Contoh:

P1: ⟨ 9,4 ⟩, ⟨ 7,4 ⟩, ⟨ 5,5 ⟩, ⟨ 4,−9 ⟩, ⟨ 3,2 ⟩, ⟨ 2,1 ⟩, ⟨ 0,10 ⟩

P2: ⟨ 9,2 ⟩, ⟨ 7,3 ⟩, ⟨ 4,1 ⟩, ⟨ 1,2 ⟩

maka P3 = P1+P2 akan mempunyai harga:

P3: ⟨ 9,6 ⟩, ⟨ 7,7 ⟩, ⟨ 5,5 ⟩, ⟨ 4,−8 ⟩, ⟨ 3,2 ⟩, ⟨ 2,1 ⟩, ⟨ 1,2 ⟩, ⟨ 0,10 ⟩



Mengurangi dua buah polinom

Mengurangi dua buah polinom P1 dan P2 dan menyimpan hasilnya pada P3, P3 $\neq$ P1 dan P3 $\neq$ P2:

- Prosesnya sama dengan menjumlahkan, hanya operasi penjumlahan diganti operasi pengurangan.

Pengurangan juga memungkinkan hasil berupa polinom kosong atau polinom yang derajatnya “turun”.

Mengurangi dua buah polinom (2)

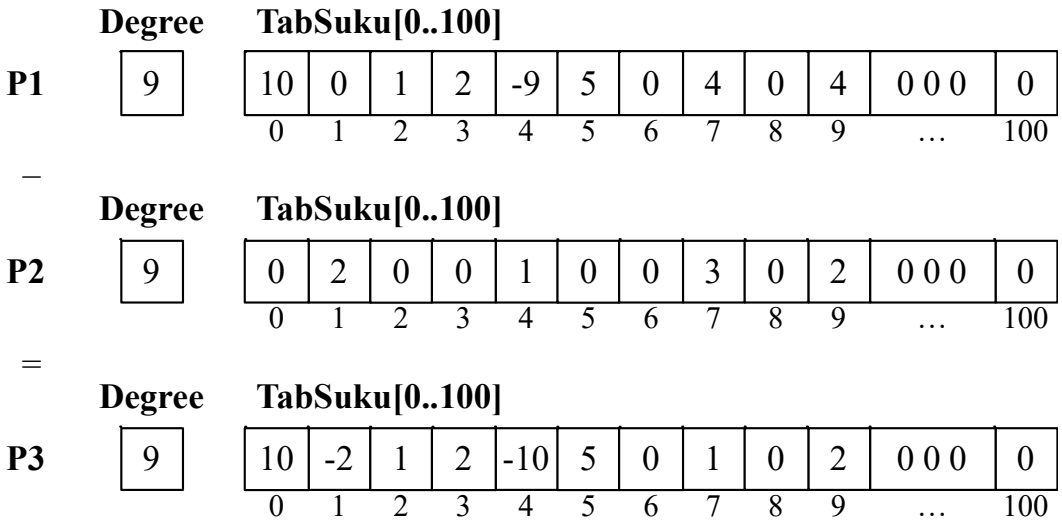
Contoh:

P1: ⟨ 9,4 ⟩, ⟨ 7,4 ⟩, ⟨ 5,5 ⟩, ⟨ 4,−9 ⟩, ⟨ 3,2 ⟩, ⟨ 2,1 ⟩, ⟨ 0,10 ⟩

P2: ⟨ 9,2 ⟩, ⟨ 7,3 ⟩, ⟨ 4,1 ⟩, ⟨ 1,2 ⟩

maka P3 = P1−P2 akan mempunyai harga:

P3: ⟨ 9,2 ⟩, ⟨ 7,1 ⟩, ⟨ 5,5 ⟩, ⟨ 4,−10 ⟩, ⟨ 3,2 ⟩, ⟨ 2,1 ⟩, ⟨ 1,−2 ⟩, ⟨ 0,10 ⟩



Membuat turunan polinom

Membuat turunan P1 dari sebuah polinom P, P1 $\neq$ P:

- Prosesnya adalah proses sekuensial, untuk setiap suku ke-i, $i>0$, yaitu $a_i x^i$ pada polinom P, dihitung $i * a_i$ dan disimpan pada indeks ke-$[i-1]$ pada tabel untuk polinom P1

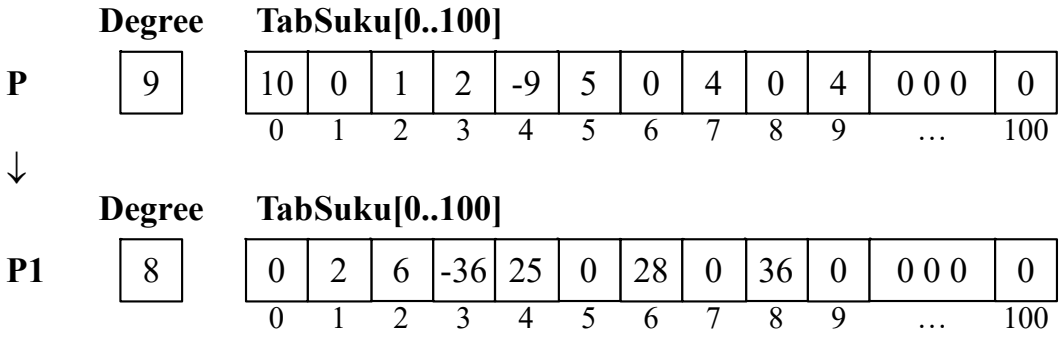
Membuat turunan polinom (2)

Contoh jika diberikan:

P: ⟨ 9,4 ⟩, ⟨ 7,4 ⟩, ⟨ 5,5 ⟩, ⟨ 4,−9 ⟩, ⟨ 3,2 ⟩, ⟨ 2,1 ⟩, ⟨ 0,10 ⟩

maka P1 = P' akan mempunyai harga:

P1: ⟨ 8,36 ⟩, ⟨ 6,28 ⟩, ⟨ 4,25 ⟩, ⟨ 3,−36 ⟩, ⟨ 2,6 ⟩, ⟨ 1,2 ⟩



```
Program POLINOMIAL                                    { Representasi KONTIGU }
KAMUS
{ Struktur data untuk representasi polinom secara kontigu}
    constant nMax: integer = 100 { Derajat tertinggi polinom yang diproses }
    type Polinom: ⟨  degree: integer,
                    arrSuku: array [0..nMax] of integer  ⟩
    p1,p2: Polinom {Operan}
    p3: Polinom { Hasil }
{ Struktur data untuk interaksi }
    finish: boolean { Mengakhiri proses }
    pilihan: integer [0..5] { Nomor tawaran }
{ Primitif operasi terhadap polinom yang dibutuhkan untuk proses }
    procedure CreatePolinom (output p: Polinom)
    { Membuat polinom p yang kosong }
    procedure adjustDegree (input/output p: polinom)
    { Melakukan adjustment terhadap Degree. Diaktifkan jika akibat suatu
      operasi, derajat polinom hasil berubah.
{ Primitif operasi terhadap polinom yang disediakan untuk pemakai }
    procedure populatePol (output p: polinom) { Mengisi polinom p }
    procedure displayPol (input p: polinom)   { Menulis polinom p }
    procedure addPol (input p1, p2: polinom, output p3: polinom)
    { Menjumlahkan p1 + p2 dan menyimpan hasilnya di p3, p3 ≠ p1 dan p3 ≠ p2 }
    procedure subPol (input p1, p2: polinom, output p3: polinom)
    { Mengurangkan p1 - p2 dan menyimpan hasilnya di p3, p3 ≠ p1 dan p3≠ p2 }
    procedure derivPol (input p: polinom, output p1: polinom)
    { Membuat turunan p dan menyimpan hasilnya di p1, p1 ≠ p }
```

```
ALGORITMA
    finish ← false
    repeat
        iterate
            output ("Ketik nomor di bawah [0..5] untuk memilih operasi")
            output ("1. Membentuk dua buah polinom P1 dan P2")
            output ("2. Menuliskan polinom P1,P2 dan P3")
            output ("3. Menjumlahkan polinom P1 dan P2 menjadi P3")
            output ("4. Mengurangkan P1 dan P2 menjadi P3")
            output ("5. Membentuk turunan polinom P1 yaitu P3")
            output ("0. Akhir proses")
            input (pilihan)
        stop pilihan ∈ [0..5]
            output ("Ulangi, pilihan di luar harga yang ditawarkan")
        { Pilihan sesuai, maka lakukan proses sesuai dengan Pilihan }
        depend on pilihan
            pilihan = 1: populatePol(p1); populatePol(p2)
            pilihan = 2: displayPol(p1); displayPol(p2); displayPol(p3)
            pilihan = 3: addPol(p1,p2,p3)
            pilihan = 4: subPol(p1,p2,p3)
            pilihan = 5: derivPol(p1,p3)
            pilihan = 0: finish ← true
    until finish
```

Studi Kasus: Polinom Representasi Berkait

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

Deskripsi Persoalan

Sebuah polinom berderajat n didefinisikan sebagai fungsi $P(x)$ berikut:

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + ... + a_2 x^2 + a_1 x^1 + a_0$$

Perhatikan bahwa untuk mempermudah pemrosesan, definisi $P(x)$ tersebut berbeda dengan definisi polinom pada matematik, yang biasanya diberikan sebagai berikut:

$$P(x) = a_0 x^n + a_1 x^{n-1} + a_2 x^{n-2} + ... + a_{n-2} x^2 + a_{n-1} x^1 + a_n$$

Contoh Polinom:

$$P1(x) = 4x^5 + 2x^4 + 7x^2 + 10$$

$$P2(x) = 23x^{100} + 9x^9 + 2x^7 + 4x^5 + 9x^9 + 2x^4 + 3x^2 + 1$$

$$P3(x) = 10$$

$$P4(x) = 3x^2 + 2x + 8$$

$$P5(x) = x^{1000}$$

Proses Dasar Polinom (1)

1. Membentuk sebuah polinom *P* dari pasangan harga yang dibaca dari masukan, data yang dibaca adalah pasangan harga:

   (*) ⟨ Degree: integer, Coefficient: integer ⟩

   (1) ⟨ −999, 0 ⟩

2. Menuliskan sebuah polinom *P* terurut mulai dari suku terbesar sampai terkecil

3. Menjumlahkan dua buah polinom *P1* dan *P2* dan menyimpan hasilnya pada *P3*,

   $P3 \neq P1$ dan $P3 \neq P2$; pada akhir proses $P3 = P1 + P2$.

Proses Dasar Polinom (2)

4. Mengurangi dua buah polinom *P1* dan *P2* dan menyimpan hasilnya pada *P3*, *P3* ≠ *P1* dan *P3* ≠ *P2*, pada akhir proses *P3* = *P1* − *P2*.
5. Membuat turunan dari sebuah polinom *P* dan menyimpan hasilnya pada *P'*, *P'* ≠ *P1*, pada akhir proses *P'* adalah turunan *P1*.

Pemilihan Struktur Data

Secara lojik sebuah suku polinom direpresentasi oleh sepasang harga integer:

```
⟨ Degree: integer, Coefficient: integer ⟩
```

P(*x*) adalah kumpulan pasangan harga tersebut yang diidentifikasi oleh namanya dan merupakan sebuah list linier dengan elemen Suku, yaitu pasangan harga `⟨ Degree: integer, Coefficient: integer ⟩`, dengan harga Degree yang maksimum sebagai derajat polinom.

P(*x*) dapat direpresentasi secara **KONTIGU** atau **BERKAIT**.

Representasi Berkait (1)

- Hanya suku yang muncul saja yang disimpan datanya.
- Degree setiap suku harus disimpan secara eksplisit.
- Operasi akan lebih efisien jika suku yang muncul diurut mulai dari derajat tertinggi sampai derajat terendah

Maka:

- terjadi penghematan memori kalau suku-suku [0..N] banyak yang tidak muncul. Kalau banyak yang muncul?
- polinom kosong menjadi sangat “natural”

Representasi Berkait (2)

```
KAMUS
{ Definisi sebuah polinom P adalah }
type Address: ... { terdefinisi alamat sebuah suku }
type Polinom: Address { alamat elemen pertama list }
type Suku: < degree: integer,
             coef: integer,
             next: Address  >
{ Polinom adalah List Suku, dengan elemen terurut menurun  menurut
  Degree }
p1, p2, p3: Polinom
```

Membentuk sebuah polinom

Membentuk sebuah polinom dari pasangan harga yang dibaca dari alat masukan:

Setiap kali memasukkan data, yang dimasukkan adalah pasangan

(*) ⟨ Degree: integer, Coefficient: integer ⟩

Akhir pemasukan adalah pasangan harga yang diketikkan bernilai ⟨ −999, 0 ⟩

Proses pembentukan polinom adalah proses sekuensial untuk membaca dari masukan dan melakukan penyisipan dalam list Suku polinom yang selalu terurut menurun menurut Degree.

Membentuk sebuah polinom

Contoh: Jika dibaca P1:

⟨ 1,4 ⟩, ⟨ 2,5 ⟩, ⟨ 5,7 ⟩, ⟨ 8,9 ⟩, ⟨ 3,4 ⟩, ⟨ −999,0 ⟩

maka list polinom yang dibentuk adalah polinom berderajat 8 dengan urutan penyisipan:

⟨ 1,4 ⟩
⟨ 2,5 ⟩, ⟨ 1,4 ⟩
⟨ 5,7 ⟩, ⟨ 2,5 ⟩, ⟨ 1,4 ⟩
⟨ 8,9 ⟩, ⟨ 5,7 ⟩, ⟨ 2,5 ⟩, ⟨ 1,4 ⟩
⟨ 8,9 ⟩, ⟨ 5,7 ⟩, **⟨ 3,4 ⟩**, ⟨ 2,5 ⟩, ⟨ 1,4 ⟩

Polinom “kosong”, yaitu polinom (P) = nil

Menuliskan sebuah polinom

Prosesnya menggunakan skema traversal dasar terhadap sebuah list polinom.

Harga suku yang dituliskan otomatis hanya yang koefisiennya tidak nol.

Contoh:

```
I   P(I)
8   9
5   7
3   4
2   5
1   4
```

Menjumlahkan dua buah polinom (1)

Menjumlahkan dua buah polinom P1 dan P2 dan menyimpan hasilnya pada P3

Prosesnya adalah "merging" dua buah list linier yang terurut dan setiap suku P1 dan P2

Analisis kasus:

- derajat P1 sama dengan derajat P2, jumlahkan dan sisipkan pada P3 (jika hasil penjumlahan tidak 0), maju ke suku P1 dan P2 yang berikutnya
- derajat P1 > derajat P2: sisipkan suku P1 pada P3, maju ke suku P1 yang berikutnya
- derajat P1 < derajat P2: sisipkan suku P2 pada P3, maju ke suku P2 yang berikutnya

Menjumlahkan dua buah polinom (2)

Contoh, jika diberikan:

P1: ⟨ 9,4 ⟩, ⟨ 7,4 ⟩, ⟨ 5,5 ⟩, ⟨ 4,−9 ⟩, ⟨ 3,2 ⟩, ⟨ 2,1 ⟩, ⟨ 0,10 ⟩
P2: ⟨ 9,2 ⟩, ⟨ 7,3 ⟩, ⟨ 4,1 ⟩, ⟨ 1,2 ⟩

Maka urutan pembentukan P3 adalah:

⟨ 9,6 ⟩	InsertLast (P3,P1+P2), Next(P1), Next(P2)
⟨ 7,7 ⟩	InsertLast (P3,P1+P2), Next(P1), Next(P2)
⟨ 5,5 ⟩	InsertLast (P3,P1), Next(P1)
⟨ 4,−8 ⟩	InsertLast (P3,P1+P2), Next(P1), Next(P2)
⟨ 3,2 ⟩	InsertLast (P3,P1), Next(P1)
⟨ 2,1 ⟩	InsertLast (P3,P1), Next(P1)
⟨ 1,2 ⟩	InsertLast (P3,P2), Next(P2)
⟨ 0,10 ⟩	InsertLast (P3,P1), Next(P1)

Keadaan akhir P3: ⟨ 9,6 ⟩, ⟨ 7,7 ⟩, ⟨ 5,5 ⟩, ⟨ 4,−8 ⟩, ⟨ 3,2 ⟩, ⟨ 2,1 ⟩, ⟨ 1,2 ⟩, ⟨ 0,10 ⟩

Mengurangi dua buah polinom

Sama dengan menjumlahkan, hanya operasi penjumlahan diganti operasi pengurangan. Contoh, jika diberikan:

P1: ⟨ 9,4 ⟩, ⟨ 7,4 ⟩, ⟨ 5,5 ⟩, ⟨ 4,−9 ⟩
P2: ⟨ 9,2 ⟩, ⟨ 7,3 ⟩, ⟨ 4,1 ⟩, ⟨ 3,2 ⟩, ⟨ 2,1 ⟩, ⟨ 1,2 ⟩, ⟨ 0,10 ⟩

maka urutan pembentukan P3 adalah sebagai berikut:

⟨ 9,2 ⟩	InsertLast(P3,P1−P2), Next(P1), Next(P2)
⟨ 7,1 ⟩	InsertLast(P3,P1−P2), Next(P1), Next(P2)
⟨ 5,5 ⟩	InsertLast(P3,P1), Next(P1)
⟨ 4,−10 ⟩	InsertLast(P3,P1−P2), Next(P1), Next(P2)
⟨ 3,−2 ⟩	InsertLast(P3,P2), Next(P2)
⟨ 2,−1 ⟩	InsertLast(P3,P2), Next(P2)
⟨ 1,−2 ⟩	InsertLast(P3,P2), Next(P2)
⟨ 0,−10 ⟩	InsertLast(P3,P2), Next(P2)

Dan keadaan akhir P3 adalah: ⟨ 9,2 ⟩, ⟨ 7,1 ⟩, ⟨ 5,5 ⟩, ⟨ 4,−10 ⟩, ⟨ 3,−2 ⟩, ⟨ 2,−1 ⟩, ⟨ 1,−2 ⟩, ⟨ 0,−10 ⟩

Membuat turunan polinom

Prosesnya adalah proses sekuensial, traversal list P, untuk setiap suku ke-i, $i>0$: yaitu $a_i x^i$ pada polinom P, dihitung $i * a_i$ dan disisipkan P1 sebagai suku ke-$(i-1)$.

Seperti pada proses penjumlahan, list P1 dibentuk dengan penyisipan elemen terakhir, dan P1 diinisialisasi sebagai list kosong.

> Karena penyisipan selalu pada akhir list, maka alamat elemen terakhir list P1 selama proses berlangsung layak untuk disimpan.

Contoh: Jika diberikan:

> P: ⟨ 9,4 ⟩, ⟨ 7,4 ⟩, ⟨ 5,5 ⟩, ⟨ 4,−9 ⟩, ⟨ 3,2 ⟩, ⟨ 2,1 ⟩, ⟨ 0,10 ⟩

maka P1 akan mempunyai harga:
⟨ 8,36 ⟩, ⟨ 6,28 ⟩, ⟨ 4,25 ⟩, ⟨ 3,−36 ⟩, ⟨ 2,6 ⟩, ⟨ 1,2 ⟩

Representasi lojik berkait	Representasi fisik berkait dengan Pointer	Representasi fisik berkait dengan Tabel
KAMUS UMUM **type** Address:... **type** Suku: < degree: **integer**, coef: **integer**, next: Address > **type** Polinom: Address p : Polinom pt: Address	KAMUS UMUM **type** Address: **pointer to** Suku **type** Suku: < degree: **integer**, coef: **integer** next: Address > **type** Polinom: Address p : Polinom pt: Address	KAMUS UMUM **constant** NMAX: **integer**=100 **type** Address: **integer**[0..NMAX] **type** Suku: < degree: **integer**, coef: **integer**, next: Address > **type** Polinom: Address arrSuku: **array**[0..NMAX] **of** Suku firstAvail: Address p : Polinom pt: Address
AKSES: FIRST(p) NEXT(pt) DEGREE(pt) COEF(pt)	AKSES: p pt↑.next pt↑.degree pt↑.coef	AKSES: p arrSuku[pt].next arrSuku[pt].degree arrSuku[pt].coef
PRIMITIF ALOKASI/DEALOKASI: { tergantung rep. fisik }	PRIMITIF ALOKASI/DEALOKASI: { sistem, e.g., malloc/free } pt ← newSuku(...) deallocSuku(pt)	PRIMITIF ALOKASI/DEALOKASI: { Harus direalisasi } initialize(arrSuku) pt ← newSuku(...) deallocSuku(pt)

```
Program POLINOMIAL1
{ Representasi BERKAIT, dengan notasi LOJIK }

KAMUS
{ Struktur data untuk representasi polinom }
    type Address: ... { type terdefinisi }
    type Suku: < degree: integer,
                 coef: integer,
                 next: Address  >
    type Polinom: Address
    constant NIL: Address = ... { untuk address tidak terdefinisi }
    p1, p2: Polinom { operan }
    p3: Polinom     { hasil }
{ Untuk interaksi: }
    finish: boolean          { mengakhiri proses }
    pilihan: integer [0..5] { nomor tawaran }
{ Primitif operasi polinom untuk operasi internal }
    function newSuku () → Address  { Alokasi sebuah suku }
    procedure deallocSuku (input/output pt: Address) { Dealokasi sebuah suku }
    procedure CreatePolinom (output p: polinom) { Membuat polinom kosong P }
    procedure insertLast (input pt: Address, input/output p: Polinom,
                          input/output last: Address) { Insert pt sesudah
                          elemen terakhir p dengan address elemen terakhir
                          = last }
```

```
{ Primitif operasi polinom yang ditawarkan ke pengguna }
    procedure populatePol (output p1: polinom) { Mengisi polinom p1 }
    procedure displayPol (input p: polinom)  { Menulis polinom p }
    procedure addPol (input p1, p2: polinom, output p3: polinom)
    { Menjumlahkan p1 + p2 dan menyimpan hasilnya di p3, p3 ≠ p1 dan
      p3 ≠ p2 }
    procedure subPol (input p1, p2: polinom, output p3: polinom)
    { Mengurangkan p1 - p2 dan menyimpan hasilnya di p3, p3 ≠ p1 dan
      p3 ≠ p2 }
    procedure derivPol (input p: polinom, output p1: polinom)
    { Membuat turunan p dan menyimpan hasilnya di p1, p1 ≠ p }

ALGORITMA
{ Sama dengan untuk representasi kontigu }
```

Studi Kasus: Polinom
Diskusi

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

Studi Kasus:
Suku-Suku Polinom Hasil Berasal dari Operan (1)

Persoalannya adalah jika pada operasi penjumlahan, pengurangan, dan derivasi pada paket polinom tersebut suku dari polinom hasil berasal dari polinom operan, dan hasilnya disimpan pada salah satu operan. Contoh: Jika P1 dan P2 adalah polinom, maka hendak dilakukan operasi:

P1 = P1 + P2

P1 = P1 – P2

P1 = P1'

Pada persoalan ini, polinom P1 “tidak ada lagi”, karena “ditimpa” oleh hasil penjumlahan

Implikasi terhadap perubahan spesifikasi tidak sama untuk representasi kontigu dan representasi berkait.

Studi Kasus:
Suku-Suku Polinom Hasil Berasal dari Operan (1)

Representasi KONTIGU

- Untuk representasi kontigu, algoritma penjumlahan dua buah polinom tidak berubah, hanya dengan menggantikan penulisan P3 menjadi P1.

Studi Kasus:
Suku-Suku Polinom Hasil Berasal dari Operan (3)

Representasi BERKAIT

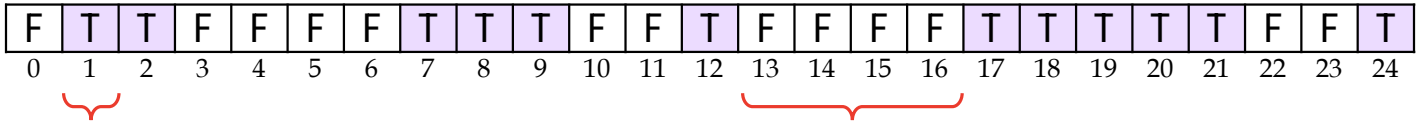
- Perubahan spesifikasi hasil operasi tersebut membawa implikasi cukup banyak pada representasi berkait.
- Prosesnya adalah traversal kedua list, setiap saat kita mengelola dua buah pointer Pt1 untuk traversal Polinom P1 dan Pt2 untuk traversal Polinom P2. Ada empat kemungkinan:
  - Degree(Pt1) > Degree(Pt2): **Tidak ada perubahan terhadap elemen list**, hanya pointer Pt1 yang maju, karena elemen Pt1 tetap menjadi elemen Polinom hasil.
  - Degree(Pt1) = Degree(Pt2)
    - Hasil penjumlahan koefisien tidak sama dengan nol: **Pengubahan nilai koefisien pada Pt1**. Pt1 dan Pt2 maju.
    - Hasil penjumlahan koefisien sama dengan nol: **Penghapusan elemen Pt1** karena hasil penjumlahan adalah nol. Pt2 maju.
  - Degree(Pt1) < Degree(Pt2): **Penambahan elemen baru** dengan degree dan koefisien Pt2 ke polinom P1. Pt2 maju.

Studi Kasus: Pengelolaan Memori Representasi Kontigu

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

Memori

Jumlah blok: N_BLOCK = 25



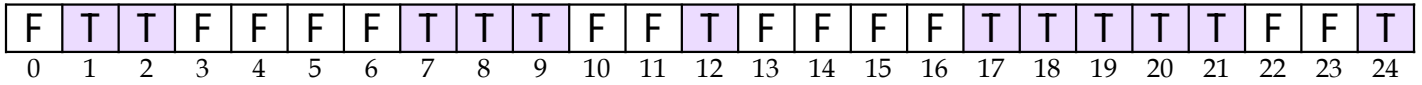
Satu blok
F: kosong
T: isi

Satu zone
⟨13,4⟩: zone bebas/kosong
⟨17,5⟩: zone isi
(zone ⟨i,n⟩: dimulai dari indeks i
sebanyak n blok)

Deskripsi Persoalan

Memori dinyatakan sebagai N_BLOCK buah blok kontigu

- F: KOSONG
- T: ISI



Zone Bebas: blok-blok berurutan yang berstatus F (KOSONG)

Dinyatakan oleh indeks blok KOSONG pertama dan ukurannya

- Zone bebas I: ⟨0,1⟩
- Zone bebas II: ⟨3,4⟩
- Zone bebas III: ⟨10,2⟩
- Zone bebas IV: ⟨13,4⟩
- Zone bebas V: ⟨22,2⟩

Deskripsi Persoalan

Alokasi dan dealokasi memori menyebabkan perubahan terhadap zone bebas

Realisasikan prosedur-prosedur sebagai berikut:

- Prosedur **initMem** mengeset semua blok menjadi blok KOSONG
- Prosedur **allocBlock** melakukan alokasi: membuat blok KOSONG sejumlah x menjadi ISI; menghasilkan indeks awal ($startIdx$) di mana alokasi dilakukan
- Prosedur **deallocBlock** "membebaskan" zone ISI: membuat blok ISI sejumlah x yang berawal di indeks $startIdx$ menjadi KOSONG
- Prosedur **compaction** *(memory compaction)* memampatkan memori sehingga semua blok KOSONG berada di bagian kiri memori dan semua blok ISI berada di kanan memori

Representasi secara kontigu dengan array

Representasi dengan Array: Struktur Data

```
KAMUS
  constant UNDEF: integer = -1
  constant N_BLOCK: integer = 100
  STATMEM: array [0..N_BLOCK-1] of boolean
  { tabel status memori: true jika ISI, false jika kosong }
```

Representasi dengan Array: Prosedur initMem

Secara umum: set seluruh elemen STATMEM menjadi bernilai false

Algoritma: Diktat hlm. 167

```
procedure initMem
{ I.S.: Sembarang }
{ F.S.: Semua blok memori dinyatakan KOSONG }
{ Proses: Semua status blok dengan indeks [0..N_BLOCK-1] dijadikan
  KOSONG dengan traversal blok [0..N_BLOCK-1].
  Proses sekuensial tanpa penanganan kasus kosong (N_BLOCK ≥ 0) }
```

Representasi dengan Array: Prosedur allocBlock (1)

```
procedure allocBlock (input x: integer, output startIdx: integer)
{ I.S.: Sembarang.
        x adalah banyaknya blok yang diminta untuk dialokasi,
        yaitu dijadikan ISI.
{ F.S.: Jika ada zone yang memenuhi syarat (ada zone dengan jumlah
        blok kontigu sebanyak x atau lebih yang berstatus kosong),
        maka startIdx akan berisi indeks blok bebas pertama pada zone
        yang dipilih untuk dialokasi;
        kemudian status pemakaian memori zone tsb. dimutakhirkan.
        Jika tidak ada lagi zone yang memenuhi syarat (tidak ada
        zone dengan jumlah blok sebanyak x) maka startIdx bernilai UNDEF
        dan status blok tetap seperti pada I.S. }
```

Representasi dengan Array: Prosedur allocBlock (2)

Strategi First Fit: Alokasi dilakukan pada zone yang memenuhi syarat yang **pertama kali ditemukan**.

Diktat hlm. 168

Sketsa umum algoritma:

inisialisasi
repeat
  *cari blok kosong pertama (skema **search**)*
  if *ketemu blok kosong* **then**
    catat indeks sebagai `NAwal`*, yaitu blok awal zone kosong*
    hitung `NKosong`*, yaitu banyaknya blok dlm zone kosong tersebut*
until *semua blok sudah diperiksa* `or NKosong ≥ x`
terminasi
if *ada zone memenuhi syarat* **then**
  ubah status semua blok pada zone tersebut [startIdx..startIdx+x−1] menjadi ISI

Representasi dengan Array: Prosedur allocBlock (3)

Strategi Best Fit: Alokasi dilakukan terhadap zone yang memenuhi syarat dan berukuran sama dengan x, atau jika tidak ada zone berukuran x, maka diambil yang ukurannya minimal. Keuntungan: blok tidak terpartisi kecil-kecil

Diktat Hlm. 169

```
inisialisasi
repeat
    cari zone kosong pertama (skema search)
    if ketemu blok kosong then
        hitung banyaknya blok dlm zone kosong tsb.
    if ada zone kosong dan memenuhi syarat then
        if zone kosong pertama then
            inisialisasi
            ukuran zone Minimum: NBMin dan Posisi Awal NAwal
        else cek apakah lebih baik, jika ya, update NBMin dan startIdx
until semua blok diperiksa or blok berukuran = x
terminasi
if ada zone memenuhi syarat then
    ubah status blok pada zone tersebut [startIdx..startIdx+x−1] menjadi ISI
```

Representasi dengan Array: Prosedur deallocBlock

Secara umum: Pemrosesan sekuensial (traversal) untuk membuat status blok [startIdx..startIdx+x-1] KOSONG

Diktat hlm. 170

```
procedure deallocBlock (input x, startIdx: integer)
{ I.S.: x adalah ukuran zone, bilangan positif dan startIdx adalah
        alamat blok awal zone tersebut, dengan startIdx ∈ [0..N_BLOCK-x],
        Blok dengan indeks startIdx s.d. startIdx+x-1 pasti berstatus ISI. }
{ F.S.: Tabel status memori dengan indeks blok startIdx..startIdx+x-1
        menjadi KOSONG }
{ Proses: Sebuah zone berukuran x dan berawal pada blok startIdx
          di-dealokasi (statusnya dijadikan kosong) }
```

Representasi dengan Array: Prosedur compaction (1)

```
procedure compaction
{ I.S.: Sembarang }
{ F.S.: Tabel status memori menjadi dua bagian:
        - zone KOSONG di belahan "kiri" (indeks kecil),
        - zone ISI di belahan "kanan".
        Jika k adalah integer 0..N_BLOCK yang merupakan indeks blok
        ISI pertama pada zone ISI, maka ada 3 kemungkinan
        F.S.:
        1. Jika k = 0  maka semua blok adalah ISI, tidak ada
           zone kosong. }
        2. Jika 0 < k < N_BLOCK, maka blok dengan indeks[0..k-1]
           adalah zone KOSONG, bagian dengan indeks [k..N_BLOCK-1]
           adalah zone ISI
        3. Jika k = N_BLOCK maka semua blok adalah
           KOSONG, memori terdiri dari sebuah zone KOSONG.
```

Representasi dengan Array: Prosedur compaction (2)

Proses: “menggeser” elemen tabel yang berstatus KOSONG ke kiri

Dua pass (diktat hlm. 170):

- Traversal untuk mencacah banyaknya blok kosong, misalnya NKosong
- Traversal untuk memberi status [0..NKosong−1] dengan KOSONG dan [NKosong..N_BLOCK−1] dengan ISI

Satu pass:

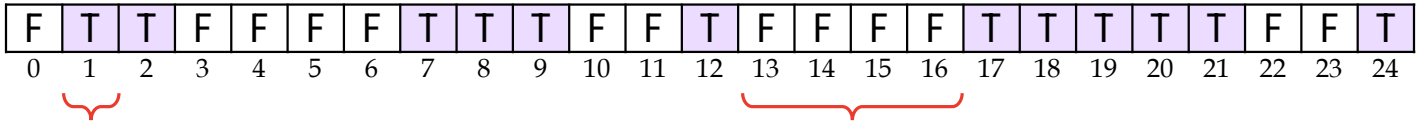
- Traversal untuk mengelompokkan KOSONG di kiri dan ISI di kanan dengan menukarkan dua elemen.

Studi Kasus: Pengelolaan Memori Rep. Berkait – Blok Kosong

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

Memori

Jumlah blok: N_BLOCK = 25



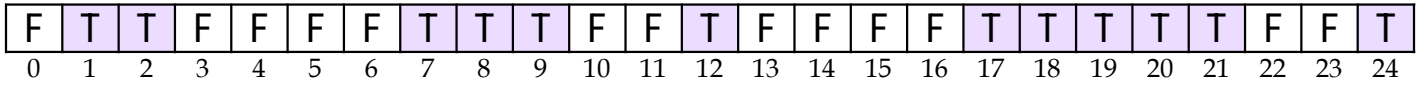
Satu blok
F: kosong
T: isi

Satu zone
⟨13,4⟩: zone bebas/kosong
⟨17,5⟩: zone isi
(zone ⟨i,n⟩: dimulai dari indeks i
sebanyak n blok)

Deskripsi Persoalan

Memori dinyatakan sebagai N_BLOCK buah blok kontigu

- F: KOSONG
- T: ISI



Zone Bebas: blok-blok berurutan yang berstatus F (KOSONG)

Dinyatakan oleh indeks blok KOSONG pertama dan ukurannya

- Zone bebas I: ⟨0,1⟩
- Zone bebas II: ⟨3,4⟩
- Zone bebas III: ⟨10,2⟩
- Zone bebas IV: ⟨13,4⟩
- Zone bebas V: ⟨22,2⟩

Deskripsi Persoalan

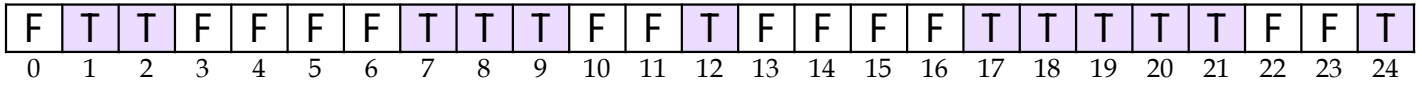
Alokasi dan dealokasi memori menyebabkan perubahan terhadap zone bebas

Realisasikan prosedur-prosedur sebagai berikut:

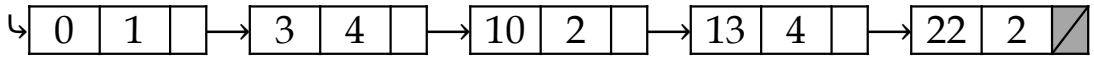
- Prosedur **initMem** mengeset semua blok menjadi blok KOSONG
- Prosedur **allocBlock** melakukan alokasi: membuat blok KOSONG sejumlah x menjadi ISI; menghasilkan indeks awal ($startIdx$) di mana alokasi dilakukan
- Prosedur **deallocBlock** "membebaskan" zone ISI: membuat blok ISI sejumlah x yang berawal di indeks $startIdx$ menjadi KOSONG
- Prosedur **compaction** (*memory compaction*) memampatkan memori sehingga semua blok KOSONG berada di bagian kiri memori dan semua blok ISI berada di kanan memori

Representasi Berkait Blok Kosong

Representasi Berkait Blok Kosong: Ilustrasi



List zone kosong:
{ ⟨0,1⟩, ⟨3,4⟩, ⟨10,2⟩, ⟨13,4⟩, ⟨22,2⟩ }



Representasi Berkait Blok Kosong: Struktur Data

```
KAMUS
    type: Address     { type terdefinisi }
    type FreeZone: <  idx: integer, { indeks awal zone kontigu kosong }
                     size: integer, { banyaknya blok zone kontigu kosong }
                     next: Address >
    FIRST_FZ: Address  { address zone kosong pertama }
    { Elemen list terurut menurut startIdx }
    { Fungsi akses untuk penulisan algoritma secara lojik }
    { Jika p adalah sebuah address, maka dituliskan:
      - next dari p adalah address elemen list sesudah elemen beralamat p
      - idx dari p adalah indeks blok kosong pertama pada sebuah elemen list
        beralamat p yang mewakili sebuah zone kontigu
      - size dari p adalah ukuran blok sebuah zone kosong yang disimpan informasinya
        dalam elemen list beralamat p
      - alloc(p) adalah prosedur utk melakukan alokasi sebuah Zona Bebas dengan
        alamat p, p tidak mungkin sama dengan NIL (alokasi selalu berhasil)
      - dealloc(p) adalah prosedur utk mendealokasi sebuah alamat p }
```

Representasi Berkait Blok Kosong: Prosedur initMem

Algoritma: diktat hlm. 176

```
procedure initMem
{ I.S.: Sembarang }
{ F.S.: Semua blok memori dinyatakan KOSONG }
{ Proses: Diinisialisasi satu zone kosong <1, N_BLOCK>
          jika N_BLOCK adalah jumlah maksimum blok. }
{ Solusi umum:
  1. Create list kosong
  2. Allocate sebuah elemen baru p dengan idx = 1 dan size = N_BLOCK
  3. Insert elemen p ke dalam List }
```

Representasi Berkait Blok Kosong:
Prosedur allocBlock – First Fit (1)

```
procedure allocBlockFirstFit (input x: integer, output startIdx: integer)
{ I.S.: Sembarang. x adalah banyaknya blok yang diminta untuk
        dialokasi, yaitu dijadikan ISI }
{ F.S.: Tergantung kepada proses }
{ Proses: Sequential search sebuah elemen list dengan properti
          jumlah blok kontigunya lebih besar atau sama dengan x.
          Pencarian segera dihentikan jika diketemukan elemen
          list yang memenuhi persyaratan tersebut. }
{ Hasil pencarian menentukan F.S.:
  1. Jika ada, maka ada dua kemungkinan:
     a. jika jumlah blok kontigu sama dengan x, hapus elemen
        list kosong tersebut,
     b. jika jumlah blok kontigu lebih besar dari x,
        update elemen list kosong tersebut.
  2. Jika tidak ada elemen list yang memenuhi syarat:
     keadaan list tetap dan startIdx diberi nilai UNDEF. }
```

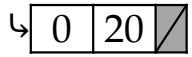
Representasi Berkait Blok Kosong:
Prosedur allocBlock – First Fit (2)

Algoritma: diktat hlm. 177

Sketsa umum algoritma:

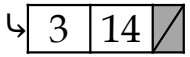
```
sequential search List FirstZB, p sebuah address elemen
kondisi berhenti: semua elemen list diperiksa atau p↑.size ≥ x
if p↑.size ≥ x then { ada yg memenuhi syarat }
  if p↑.size = x then { zone kosong menjadi isi }
    delete elemen beralamat p; dealokasi p
  else { lebih besar: update zone kosong }
    update p↑.size dan p↑.idx
  startIdx ← p↑.idx
else
  startIdx ← UNDEF
```

Ilustrasi Alokasi: First Fit

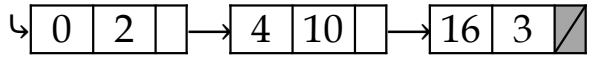
Ⓐ 

(N_BLOCK = 20)

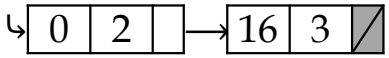
allocBlockFirstFit(3,idx) ⇒



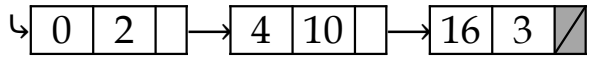
(idx = 0)

Ⓑ 

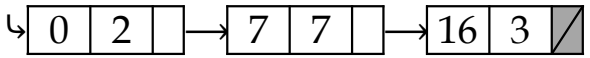
allocBlockFirstFit(10,idx) ⇒



(idx = 4)

Ⓒ 

allocBlockFirstFit(3,idx) ⇒



(idx = 4)

Representasi Berkait Blok Kosong:
Prosedur allocBlock – Best Fit (1)

```
procedure allocBlockBestFit (input x: integer, output startIdx: integer)
{ I.S.: Sembarang; x adalah banyaknya blok yang diminta untuk
        dialokasi, yaitu status memorinya dijadikan ISI }
{ F.S.: Tergantung kepada proses }
{ Proses: Periksa semua elemen list, elemen list dengan properti
          jumlah blok kontigunya lebih besar atau sama dengan x
          ditandai yang minimum. }
{ Setelah semua elemen diperiksa, ada dua kemungkinan:
  1. Jika alokasi dapat dilakukan, ada blok yang memenuhi syarat,
     masih ada dua kemungkinan:
     a. jika jumlah blok kontigu sama dengan x, hapus elemen list
        kosong tersebut,
     b. jika jumlah blok kontigu lebih besar dari x, update elemen
        list kosong tersebut.
  2. Jika alokasi tidak dapat dilakukan (tidak ada blok yang
     memenuhi syarat), maka list tetap keadaannya dan startIdx diberi
     nilai UNDEF }
```

Representasi Berkait Blok Kosong

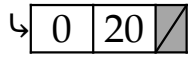
Prosedur allocBlock – Best Fit (2)

Algoritma: diktat hlm. 178

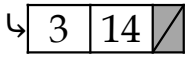
sequential search List `FirstFZ`, `p` *sebuah address elemen*
kondisi berhenti: `p↑.size = x` *atau semua elemen list sudah diperiksa*
(skema search dengan boolean)
untuk setiap elemen list beralamat `p` *yang diperiksa:*

```
  if elemen pertama then
    inisialisasi NBMin
  else { bukan elemen pertama }
    cek apakah p↑.size < NBMin, jika ya update NBMin
if (p↑.size ≥ x) then { ada yang memenuhi syarat }
  if (p↑.size = x) then { zone kosong menjadi isi }
    delete elemen beralamat p; dealokasi p
  else { lebih besar: update zone kosong }
    update p↑.size dan p↑.idx
  startIdx ← p↑.idx
else startIdx ← UNDEF
```

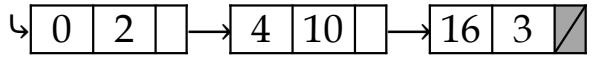
Ilustrasi Alokasi: Best Fit

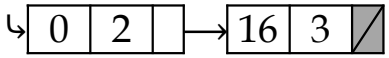
Ⓐ 

(N_BLOCK = 20)

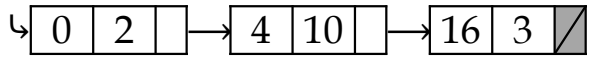
allocBlockBestFit(3,idx) ⇒ 

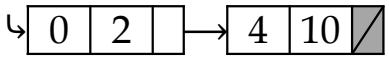
(idx = 0)

Ⓑ 

allocBlockBestFit(10,idx) ⇒ 

(idx = 4)

Ⓒ 

allocBlockBestFit(3,idx) ⇒ 

(idx = 16)

Representasi Berkait Blok Kosong:
Prosedur deallocBlock (1)

Algoritma: diktat hlm. 179

```
procedure DeAlokBlokB (input startIdx: integer, input x: integer)
{ I.S.: x adalah ukuran zone, bilangan positif dan startIdx adalah
        alamat blok awal zone tersebut, dengan startIdx ∈ [0..N_BLOCK-x],
        blok dengan indeks startIdx s.d. startIdx+x-1 pasti berstatus ISI. }
{ F.S.: Tabel status memori dengan indeks blok startIdx..startIdx+x-1
        menjadi KOSONG. }
{ Proses: Sebuah zone berukuran x dan berawal pada blok startIdx
          didealokasi (statusnya dijadikan KOSONG). }
```

Representasi Berkait Blok Kosong:
Prosedur deallocBlock (2)

Kasus-kasus pada proses dealokasi:

- Zone yang dibebaskan mengubah elemen pertama list:
  - Hanya mengubah elemen pertama list
  - Insert first → menambah zone bebas di awal list
- Zone yang dibebaskan mengubah elemen terakhir list:
  - Hanya mengubah elemen terakhir list
  - Insert last → menambah zone bebas di akhir list
- Zone yang dibebaskan berada di tengah list, di antara elemen KIRI dan KANAN:
  - KIRI dan KANAN digabung → salah satu di-delete, lainnya di-update
  - Di tengah KIRI dan KANAN, tapi tidak bersambung → insert elemen baru di antara KIRI dan KANAN
  - Terletak sesudah elemen KIRI → update KIRI
  - Terletak sebelum elemen KANAN → update KANAN

Jauh lebih rumit daripada representasi secara kontigu!

Representasi Berkait Blok Kosong:
Prosedur compaction

Algoritma: Diktat hlm. 180

```
procedure compaction
{ I.S.: Sembarang }
{ F.S.: Semua blok kosong ada di kiri dan blok isi di kanan:
        Jika ada zone kosong, hanya ada satu elemen list,
        karena dijadikan satu zone kosong.
        Jika tidak list zone kosong tidak ada elemennya (kosong),
        maka tidak dilakukan apa-apa. }
{ Proses: Jika list zone kosong tidak kosong, jadikan sebuah
          list dengan elemen tunggal beralamat p,
          p↑.idx bernilai 0 dan p↑.size bernilai jumlah total blok kosong
          (dihitung melalui iterasi list zone kosong yang lama). }
```