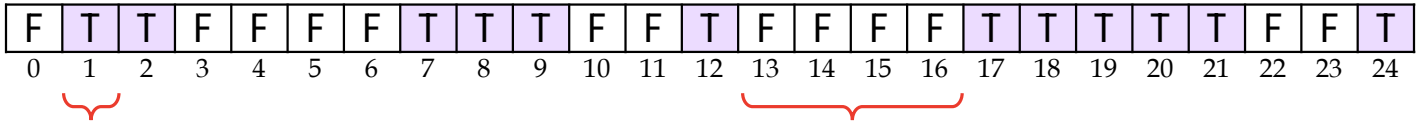
Studi Kasus: Pengelolaan Memori Rep. Berkait – Blok Kosong

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

Memori

Jumlah blok: N_BLOCK = 25



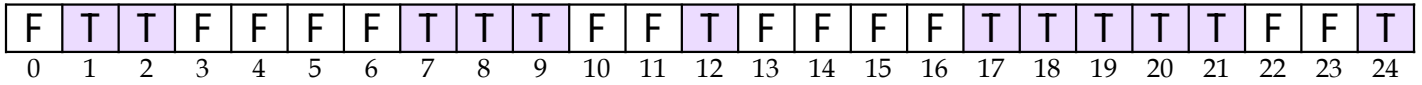
Satu blok
F: kosong
T: isi

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

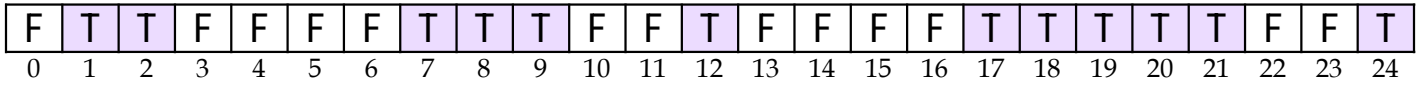
Alokasi dan dealokasi memori menyebabkan perubahan terhadap zone bebas

Realisasikan prosedur-prosedur sebagai berikut:

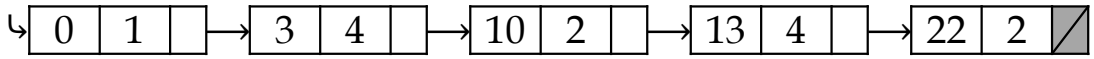
- Prosedur **initMem** mengeset semua blok menjadi blok KOSONG
- Prosedur **allocBlock** melakukan alokasi: membuat blok KOSONG sejumlah x menjadi ISI; menghasilkan indeks awal ($startIdx$) di mana alokasi dilakukan
- Prosedur **deallocBlock** "membebaskan" zone ISI: membuat blok ISI sejumlah x yang 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. }
```

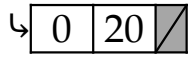
Representasi Berkait Blok Kosong:
Prosedur allocBlock – First Fit (2)

Algoritma: diktat hlm. 177

Sketsa umum algoritma:

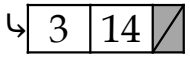
```
sequential search List FirstZB, p sebuah address elemen
kondisi berhenti: semua elemen list diperiksa atau p↑.size ≥ x
if p↑.size ≥ x then { ada yg memenuhi syarat }
  if p↑.size = x then { zone kosong menjadi isi }
    delete elemen beralamat p; dealokasi p
  else { lebih besar: update zone kosong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

Ilustrasi Alokasi: First Fit

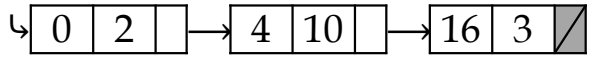
Ⓐ 

(N_BLOCK = 20)

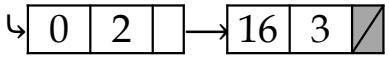
allocBlockFirstFit(3,idx) ⇒



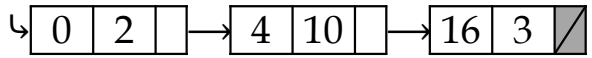
(idx = 0)

Ⓑ 

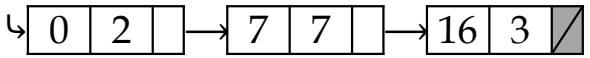
allocBlockFirstFit(10,idx) ⇒



(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

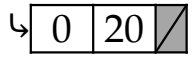
Prosedur allocBlock – Best Fit (2)

Algoritma: diktat hlm. 178

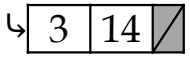
sequential search List `FirstFZ`, `p` *sebuah address elemen*
kondisi berhenti: `p↑.size = x` *atau semua elemen list sudah diperiksa*
(skema search dengan boolean)
untuk setiap elemen list beralamat `p` *yang diperiksa:*

```
  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
```

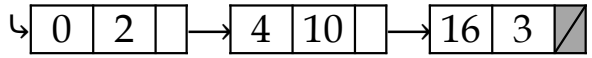
Ilustrasi Alokasi: Best Fit

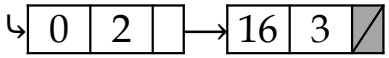
Ⓐ 

(N_BLOCK = 20)

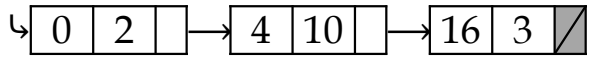
allocBlockBestFit(3,idx) ⇒ 

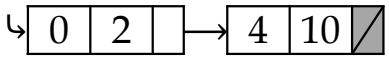
(idx = 0)

Ⓑ 

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). }
```