BAB III

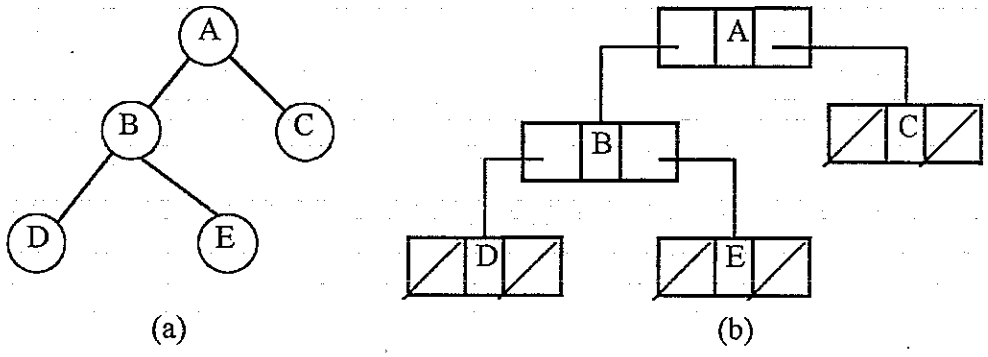
KUNJUNGAN PADA POHON BINER

3.1 Representasi Pohon Biner

Implementasi pohon biner dalam pengingat komputer dapat dilakukan dengan beberapa cara, salah satunya dengan menggunakan senarai berantai. Dalam hal ini setiap simpul pada pohon biner berisi dua buah field pointer yang menunjuk ke cabang kiri dan cabang kanan, dan sebuah field yang berisi informasi yang disimpan dalam simpul tersebut. Sehingga simpul pohon biner dapat disajikan sebagai :

Gambar 3.1

Dengan menggunakan simpul seperti pada gambar 3.1, berikut ini adalah contoh penggambaran pohon biner secara lengkap.



Gambar 3.2

Dalam hal ini field pointer (PKANAN dan PKIRI) yang tidak menunjuk ke cabang atau subpohon kiri atau kanan diberi nilai NIL.

3.2 Kunjungan pada Pohon Biner

Pada sebuah pohon biner dapat dilakukan sejumlah operasi. salah satu operasi yang dapat dilakukan adalah melakukan kunjungan pada pohon biner (binary traversal).

Definisi 3.2.1 :

Yang dimaksud dengan kunjungan pada pohon biner adalah proses mendatangi setiap simpul dari pohon secara sistematik masing-masing satu kali.

Pada saat setiap simpul didatangi dilakukan suatu proses pengolahan data yang tersimpan dalam simpul tersebut. Misalnya dilakukan proses pencetakan informasi yang ada pada simpul tersebut. Jika :

L : bergerak ke subpohon bagian kiri

V : proses mengunjungi simpul

R : bergerak ke subpohon kanan

maka kunjungan pada pohon biner mempunyai 6 kemungkinan yaitu :

1. Bergerak dari kiri ke kanan : LVR, LRV, VLR
2. Bergerak dari kanan ke kiri : RVL, RLV, VRL

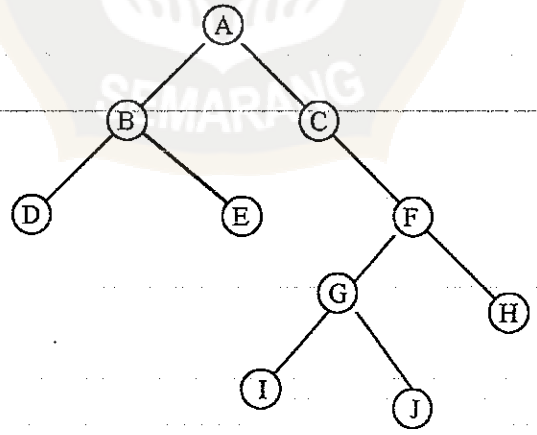
Jika proses kunjungan yang diperhatikan hanya proses dari arah kiri ke kanan, maka hanya terdapat tiga cara kunjungan yaitu kunjungan preorder, inorder dan postorder.

Definisi 3.2.2 :

Kunjungan preorder adalah kunjungan pada setiap simpul pohon biner dengan mendahulukan kunjungan terhadap simpul orangtua sebelum simpul anaknya dan mendahulukan kunjungan terhadap simpul anak kiri sebelum simpul anak kanan.

Contoh 3.2.1 :

Jika diberikan pohon biner berikut (gambar 3.3) maka didapatkan daftar simpul yang dikunjungi secara berurutan : ABDECFGIJH



Gambar 3.3

Definisi 3.2.3 :

Kunjungan inorder adalah kunjungan pada setiap simpul pohon biner dengan mendahulukan kunjungan terhadap simpul anak kiri sebelum simpul orangtua dan mengunjungi simpul anak kanan setelah mengunjungi simpul orangtua.

Contoh 3.2.2 :

Kunjungan inorder pada pohon biner gambar 3.3 menghasilkan daftar simpul yang dikunjungi secara berurutan : DBEACIGJFH

Definisi 3.2.4 :

Kunjungan postorder adalah kunjungan pada setiap simpul pohon biner dengan mendahulukan kunjungan terhadap simpul anak sebelum simpul orang tua dan mendahulukan kunjungan terhadap simpul anak kiri sebelum simpul anak kanan.

Contoh 3.2.3 :

Kunjungan postorder pada pohon biner gambar 3.3 menghasilkan daftar simpul yang dikunjungi secara berurutan : DEBIJGHFCA.

Definisi 3.2.5 :

Simpul Y disebut penerus (successor) dari simpul X, jika dalam suatu kunjungan simpul Y dikunjungi setelah mengunjungi simpul X.

Contoh 3.2.4 :

Pada contoh 3.2.3, simpul E adalah penerus (successor) secara postosder dari simpul D.

3.3. Kunjungan Menggunakan Thread

Ada beberapa cara dalam mengimplementasikan ketiga aturan kunjungan di atas. Salah satunya yaitu dengan menggunakan thread. Sebelum membahas tentang kunjungan menggunakan thread akan dibahas terlebih dahulu mengenai struktur data pohon biner thread.

3.3.1 Struktur Data Pohon Biner Thread

Pada struktur data pohon biner, banyak terdapat field pointer kiri / kanan (PKIRI / PKANAN) yang tidak terpakai karena bernilai nil, yaitu tidak ada percabangannya. Hal ini terjadi terutama pada simpul daun dari pohon. Pointer kiri dan kanan dari setiap daun adalah nil. Misalnya pada pohon biner pada gambar 3.2 (a) terdapat 6 pointer yang NIL.

Definisi 3.3.1 :

Pohon biner thread adalah pohon biner yang mempunyai ciri dan struktur data sebagai berikut :

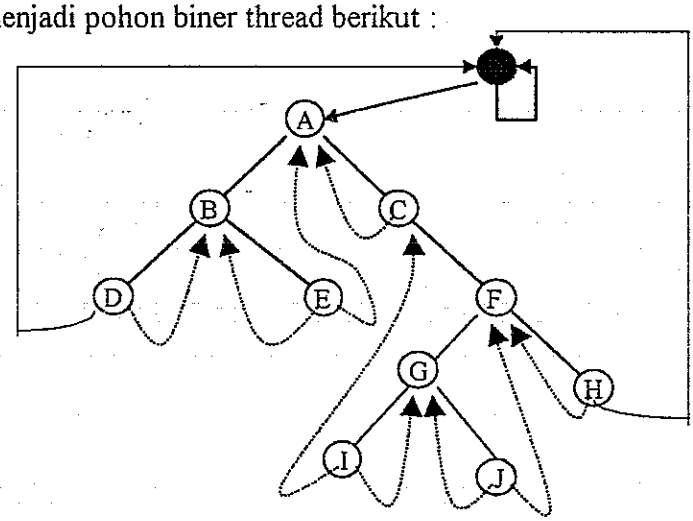
- Setiap simpulnya tidak mempunyai pointer yang bernilai nil, karena pointer yang sebelumnya bernilai nil diisi dengan alamat simpul orangtuanya.
- Terdapat sebuah simpul header yang kosong sedemikian sehingga pointer kiri dari header menunjuk ke simpul akar pohon biner dan pointer kanan dari header menunjuk ke dirinya sendiri.

Jika p adalah sembarang simpul pada pohon biner dan p mempunyai pointer nil, maka aturan pengisian pointer nil tersebut agar terbentuk pohon biner thread adalah sebagai berikut :

- Jika pointer kanan p (p^.pkanan) bernilai nil, maka isilah dengan alamat simpul yang akan dikunjungi secara inorder (inorder successor).
- Jika pointer kiri p (p^.pkiri) bernilai nil, maka isilah dengan alamat simpul yang dikunjungi tepat sebelumnya secara inorder (inorder predecessor).
- Pointer kiri yang nil dari simpul daun terkiri dari akar, diisi dengan alamat simpul header.
- Pointer kanan yang nil dari simpul daun terkanan dari akar, diisi dengan alamat simpul header.

Contoh 3.3.1 :

Dengan menggunakan definisi 3.3.1 , pohon biner pada gambar 3.3 dapat dibentuk menjadi pohon biner thread berikut :



Gambar 3.4

Definisi 3.3.2 :

Misalkan p adalah sembarang simpul pada pohon biner thread. Pointer thread dari p didefinisikan sebagai pointer yang menunjuk ke suatu simpul, di mana simpul yang ditunjuk tersebut bukan simpul anak kiri / kanan dari p.

Contoh 3.3.2 :

Pada gambar 3.4 pointer yang digambarkan dengan garis putus-putus adalah pointer thread.

Definisi 3.3.3 :

Simpul thread adalah simpul yang ditunjuk oleh pointer thread.

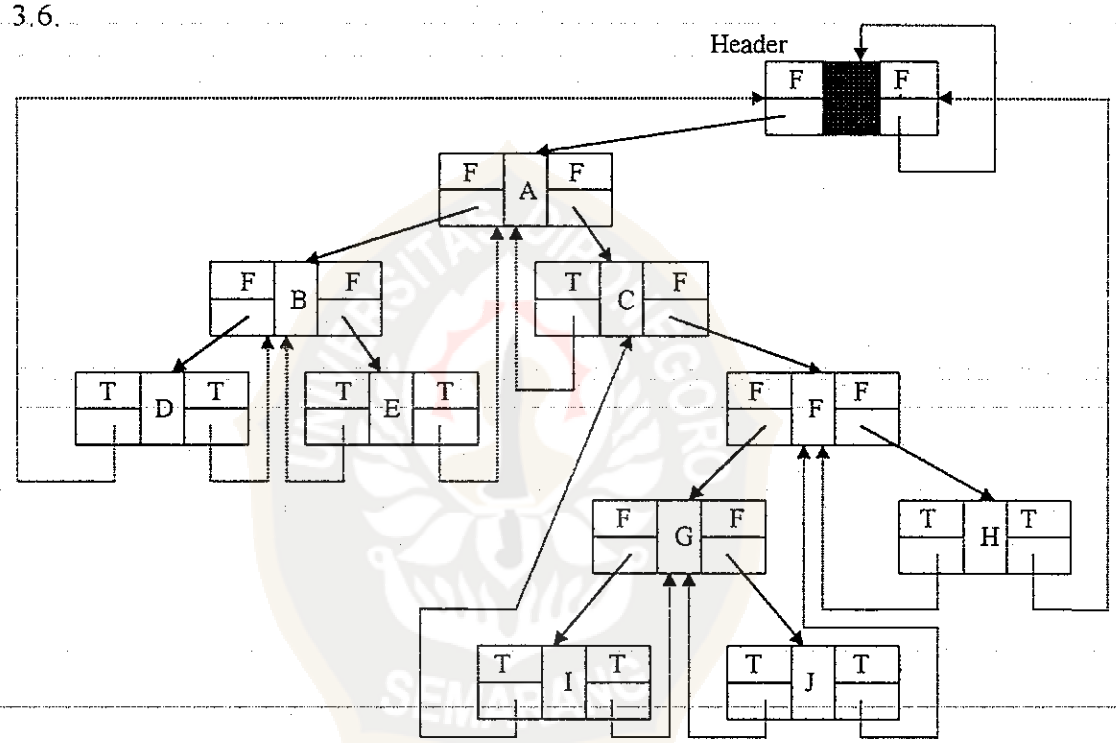
Contoh 3.3.3 :

Pada gambar 3.4, simpul B adalah simpul thread bagi simpul D dan E, karena ditunjuk oleh pointer thread yang terdapat pada simpul D dan E.

Untuk membedakan antara pointer biasa, yaitu pointer yang menunjuk ke cabang-cabang simpul, dengan pointer thread, maka struktur data pohon biner yang telah disajikan di muka (gambar 3.1) harus dirubah yaitu dengan menambahkan 2 field yang bertipe boolean (THKIRI dan THKANAN). Kedua field ini bernilai benar jika field kiri / kanan adalah pointer thread. Bernilai salah jika pointer kiri / kanan adalah bukan pointer thread. Sehingga struktur data suatu simpul pada pohon biner thread dapat disajikan sebagai berikut :

Gambar 3.5

dan pohon biner thread pada gambar 3.4 dapat disajikan kembali seperti pada gambar 3.6.



Gambar 3.6

3.3.2 Algoritma Kunjungan pada Pohon Biner Thread

Algoritma kunjungan pada pohon biner thread disusun dengan menggunakan ketiga cara kunjungan (preorder, inorder dan postorder) dan dengan memperhatikan konsep pohon biner thread. Dengan mengasumsikan bahwa suatu pohon biner merupakan cabang atau subpohon kiri dari simpul header, maka kunjungan pada pohon biner thread dimulai dari simpul header.

Berikut diberikan algoritma kunjungan pada pohon biner thread :

- Input : Pohon biner thread.
- Output : Daftar urutan simpul yang dikunjungi.
- Langkah :

(1) Definisikan pointer temp yang menunjuk ke simpul header (temp:=header)

(2) Tentukan penerus dari temp.

(3) Pindahkan pointer temp ke penerus temp (temp=penerus(temp))

(4) Jika temp ≠ header, cetak informasi yang ada pada temp. Kembali ke (2).

Sebaliknya, jika temp = header, kunjungan selesai.

Contoh 3.3.4 :

A. Kunjungan Inorder terhadap pohon biner thread pada gambar 3.6.

(1) temp:=header

(2) penerus temp secara inorder adalah D

(3) temp:=D

(4) karena temp≠header, cetak temp^.info. Kembali ke (2)

(2) penerus D secara inorder adalah B

(3) temp:=B

(4) karena temp≠header, cetak temp^.info. Kembali ke (2)

Dan seterusnya sehingga tercetak urutan info DBEACIGJFH.

B. Kunjungan Preorder terhadap pohon biner thread pada gambar 3.6

(1) temp := header

(2) penerus temp secara preorder adalah A

(3) temp:=A

(4) Karena temp≠header, cetak temp^.info. Kembali ke (2)

(2) penerus A secara preorder adalah B

(3) temp:=B

(4) Karena temp≠header, cetak temp^.info. Kembali ke (2)

Dan seterusnya sehingga tercetak urutan info ABDECFGIJH.

C. Kunjungan Postorder terhadap pohon biner thread pada gambar 3.6.

(1) temp:=header

(2) penerus temp secara postorder adalah D

(3) temp:=D

(4) karena temp≠header, cetak temp^.info. Kembali ke (2)

(2) penerus D secara postorder adalah E

(3) temp:=E

(4) karena temp≠header, cetak temp^.info. Kembali ke (2)

Dan seterusnya sehingga tercetak urutan info DEBIJGHFCA.

Algoritma tersebut dapat ditulis kembali ke dalam sebuah prosedur berikut :

```
(1) temp := header;
(2) repeat
(3)     temp := penerus(temp);
(4)     if temp <> header
```

(5) then write(temp^.info)

(6) until temp=header;

Prosedur 3.1

Dari contoh terakhir di atas dapat disimpulkan bahwa perbedaan dari ketiga algoritma kunjungan secara preorder, inorder maupun postorder terletak pada langkah 3 dari prosedur 3.1, yaitu cara menentukan simpul penerus suatu simpul.

Berikut ini diberikan algoritma untuk menentukan penerus dari suatu simpul secara inorder, preorder dan postorder.

3.3.2.1 Algoritma menentukan penerus simpul secara inorder

- Input : p, yaitu simpul yang akan dicari penerusnya.
- Output : q, yaitu simpul penerus p.
- Langkah :
  (1) Definisikan q sebagai simpul yang ditunjuk oleh pointer kanan p (q:=p^.pkanan)
  (2) Jika q adalah simpul cabang kanan dari p maka kerjakan langkah berikut. Jika tidak teruskan ke langkah (3).
    (2a) Jika q^.pkiri adalah simpul cabang maka definisikan kembali q sebagai simpul cabang tersebut, q:=q^.pkiri. Sebaliknya, jika q^.pkiri adalah simpul thread teruskan ke langkah (3).
    (2b) Ulangi langkah (2a)
  (3) q adalah penerus p.

Contoh 3.3.5 :

Menentukan penerus inorder dari header (gambar 3.6)

- Input : header
- Inisialisasi : p=header
- Output : q
- Langkah :

(1) q := p^.pkanan := header

(2) Karena header merupakan subpohon kanan dari header maka teruskan ke langkah (2a) dan (2b).

(2a) q:= header mempunyai subpohon kiri yaitu A, maka q:=A

(2b)(2a) q:=A mempunyai subpohon kiri yaitu B maka q:=B

(2b)(2a) q:=B mempunyai subpohon kiri yaitu D maka q:=D

(2b)(2a) q:=D tidak mempunyai subpohon kiri, teruskan ke langkah (3).

(3) q:=D adalah penerus header.

Algoritma menentukan penerus simpul secara inorder dapat dituliskan kembali dalam prosedur berikut :

```
procedure insucc(p);
(1) q:=p^.pkanan
(2) if p^.thkanan=FALSE then
(3)     while q^.thkiri=FALSE do q:=q^.pkiri
(4) return(q)
```

Prosedur 3.2

3.3.2.2 Algoritma menentukan penerus simpul secara Preorder

- Input : p, simpul yang akan dicari penerusnya.
- Output : simpul penerus p.
- Langkah :

(1) Jika p^.pkiri adalah simpul anak kiri, maka simpul anak tersebut adalah penerus p. Langkah selesai. Sebaliknya, jika p^.pkiri adalah simpul thread, teruskan ke langkah (2).

(2) Jika p^.pkanan adalah simpul thread, maka definisikan kembali p sebagai simpul thread tersebut, p:=p^.pkanan. Sebaliknya, jika p^.pkanan adalah simpul cabang teruskan ke langkah (4).

(3) Kerjakan kembali langkah (2).

(4) Penerus p adalah simpul yang ditunjuk oleh pointer kanan p (p^.pkanan).

Contoh 3.3.6 :

A. Menentukan penerus inorder dari header pada gambar 3.6 :

- Input : simpul header
- Output : simpul penerus header
- Inisialisasi : p:=header
- Langkah :

(1) p^.pkiri adalah simpul anak kiri dari p, yaitu simpul A. Maka A adalah penerus p. Langkah selesai.

B. Menentukan penerus inorder dari simpul A pada gambar 3.6 :

- Input : simpul A
- Output : simpul penerus A
- Inisialisasi : p:=A
- Langkah :

  (1) p^.pkiri adalah simpul anak kiri dari p, yaitu simpul B. Maka B adalah penerus p. Langkah selesai.

C. Dengan cara yang sama didapatkan penerus simpul B adalah simpul D.

D. Menentukan penerus inorder dari simpul D pada gambar 3.6 :

- Input : simpul D
- Output : simpul penerus D
- Inisialisasi : p:=D
- Langkah :

  (1) p^.pkiri adalah simpul thread maka teruskan ke langkah (2).

  (2) p^.pkanan = B adalah simpul thread, maka p:=B.

  (3) (2) Karena p^.pkanan = E adalah simpul anak teruskan ke (4)

  (4) p^.pkanan = E adalah penerus D.

Algoritma menentukan penerus simpul secara preorder dapat dituliskan kembali dalam prosedur berikut :

```
prosedure presucc(p);
if p^.thkiri=FALSE then return (p^.pkiri);
while p^.thkanan=TRUE do p:=p^.pkanan;
return(p^.pkanan);
```

Prosedur 3.3

3.3.2.3 Algoritma menentukan penerus simpul secara Postorder

- Input : p, simpul yang akan dicari penerusnya.
- Output : simpul penerus p.
- Langkah :

(1) Definisikan pointer q yang menunjuk ke p (q:=p).

(2) Jika p^.pkanan adalah simpul cabang kanan, pindahkan pointer p ke simpul cabang kanan tersebut. Sebaliknya, jika p^.pkanan adalah simpul thread, teruskan ke (4).

(3) Kerjakan kembali langkah (2).

(4) Pindahkan pointer p ke simpul yang ditunjuk oleh p^.pkanan.

(5) Jika simpul yang ditunjuk oleh pointer kiri p adalah simpul q (p^.pkiri:=q) kerjakan langkah berikut. Jika tidak, teruskan ke (6).

(5a) Jika p^.pkanan adalah simpul cabang kerjakan langkah berikut. Jika tidak teruskan ke (9).

(5a-1) Pindahkan pointer p ke simpul yang ditunjuk oleh pointer kanan (p:=p^pkanan).

(5a-2) Jika p^.pkiri adalah simpul cabang kerjakan langkah berikut. Jika tidak teruskan ke (5b).

(5a-3) Pindahkan pointer p ke simpul yang ditunjuk oleh pointer kiri (p:=p^pkiri).

(5a-4) Kerjakan kembali (5a-2).

(5b) Kerjakan kembali (5a).

(6) Pindahkan pointer p ke simpul yang ditunjuk oleh pointer kiri (p:=p^pkiri).

(7) Jika simpul yang ditunjuk oleh pointer kanan p (p^.pkanan) adalah bukan simpul q, maka pindahkan pointer p ke simpul yang ditunjuk oleh pointer kanan (p:=p^.pkanan). Sebaliknya, jika tidak teruskan ke (9).

(8) Kerjakan kembali (7).

(9) Simpul penerus yang dicari adalah simpul p.

Contoh 3.3.7 :

A. Menentukan penerus postorder dari simpul D pada gambar 3.6 :

- Input : simpul D
- Output : simpul penerus D
- Inisialisasi : p:=D
- Langkah :

(1) q:=D

(2) p^.pkanan bukan simpul cabang, maka teruskan ke langkah (4).

(4) p:= D^.pkanan :=B

(5) Simpul yang ditunjuk oleh pointer kiri B adalah q maka teruskan ke langkah berikutnya.

(5a) B^.pkanan adalah simpul cabang maka kerjakan langkah berikutnya.

(5a-1) p:=B^.pkanan:=E.

(5a-2) Karena E^.pkiri bukan simpul cabang, teruskan ke (9).

(9) simpul penerus yang dicari adalah simpul E.

B. Menentukan penerus postorder dari simpul E pada gambar 3.6 :

- Input : simpul E
- Output : simpul penerus E
- Inisialisasi : p:=E
- Langkah :

(1) q:=E

(2) p^.pkanan adalah simpul thread, maka teruskan ke (4)

(4) p:=A.

(5) A^.pkiri := B bukan q, maka teruskan ke (6)

(6) p:=B.

(7) B^.pkanan = q, maka teruskan ke (9).

(9) Simpul penerus yang dicari adalah simpul B.

C. Menentukan penerus postorder dari simpul B pada gambar 3.6 :

- Input : simpul B

- Output : simpul penerus B
- Inisialisasi : p:=B
- Langkah :

(1) q:=B

(2) p^.pkanan adalah E yang merupakan simpul cabang, maka p:=E.

(3) (2) p^.pkanan = E^.pkanan bukan simpul cabang, maka teruskan ke (4).

(4) p:=A.

(5) A^.pkiri = q maka kerjakan langkah berikutnya.

(5a) A^.pkanan adalah simpul cabang. Kerjakan langkah berikutnya.

(5a-1) p:=p^.pkanan:=C

(5a-2) C^.pkiri bukan simpul cabang, maka teruskan ke (5b)..

(5b) (5a) C^.pkanan adalah simpul cabang, maka teruskan langkah berikutnya.

(5a-1) p:=p^.pkanan:=F.

(5a-2) F^:pkiri adalah simpul cabang, maka teruskan ke (5a-3).

(5a-3) p:=p^.pkiri := G.

(5a-4) (5a-2) G^.pkiri adalah simpul cabang, maka teruskan ke (5a-3).

(5a-3) p:=p^.pkiri = I.

(5a-4) (5a-2) I^.pkiri bukan simpul cabang maka teruskan ke (5b).

(5b) I^.pkanan bukan simpul cabang, teruskan ke (9).

(9) Simpul penerus yang dicari adalah simpul I.

Algoritma menentukan penerus simpul secara postorder dapat dituliskan kembali dalam prosedur berikut :

```
procedure postsucc(p);
(1)  q:=p;
(2)  while p^.thkanan = FALSE do p:=p^.pkanan ;
(3)  p:=p^.pkanan ;
(4)  if p^.pkiri = q
(5)       then
(6)              while p^.thkanan = FALSE do
(7)               { p:=p^.pkanan;
(8)                 while p^.thkiri = FALSE do p:=p^.pkiri}
(9)       else
(10)             {p:=p^.pkiri;
(11)              while p^.pkanan <> q do p:=p^.pkanan}
(12) return(p);
```

Prosedur 3.4