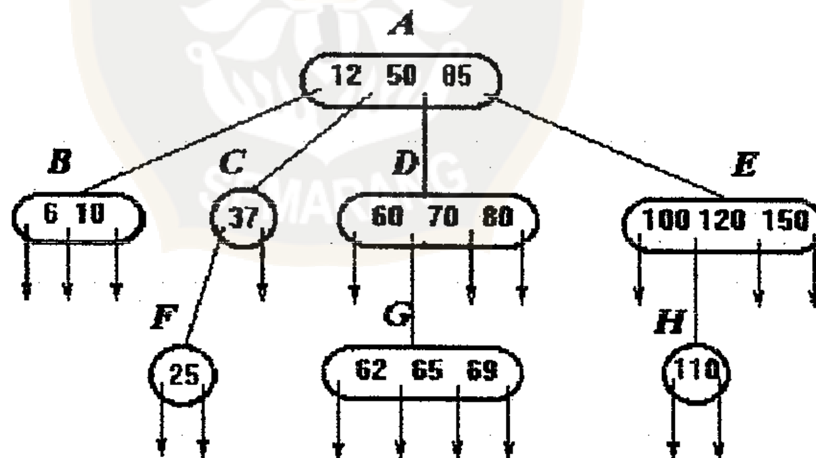
BAB III

POHON TELUSUR BANYAK CABANG

(*MULTIWAY SEARCH TREE*)

3.1 Karakteristik Pohon Telusur *m* - Cabang

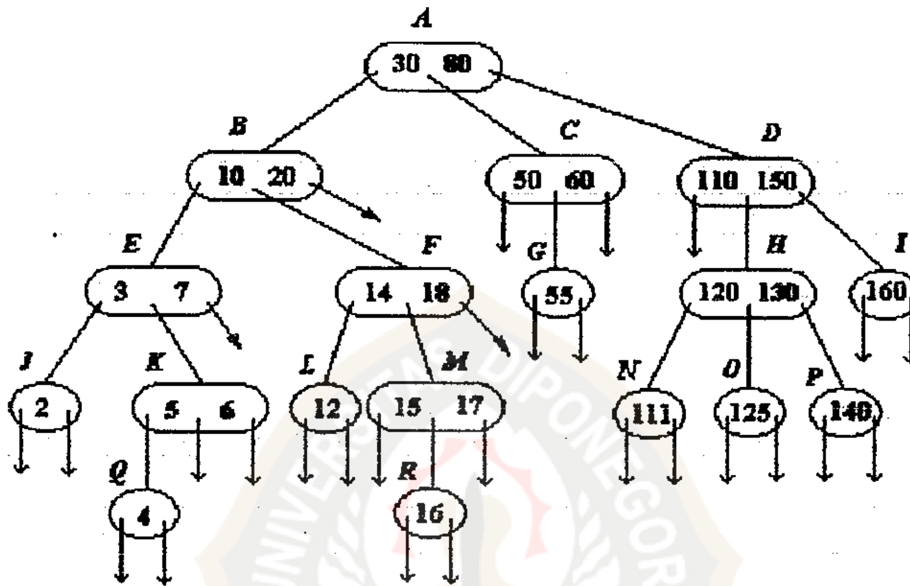
Pohon telusur banyak cabang orde m (*multiway search tree of order **m***) adalah pohon umum yang setiap simpulnya memiliki paling banyak *m* buah subpohon (cabang) atau kurang dan memuat $m - 1$ kunci dari banyaknya subpohon.



Gambar 5. Pohon telusur banyak cabang orde 4

Gambar 5. di atas adalah contoh **pohon telusur banyak cabang orde 4.** Simpul *A, D, E, G* disebut **simpul penuh** (***full nodes***), karena memuat maksimum banyaknya kunci .

This document is Undip Institutional Repository Collection. The author(s) or copyright owner(s) agree that UNDIP-IR may, without changing the content, translate the submission to any medium or format for the purpose of preservation. The author(s) or copyright owner(s) also agree that UNDIP-IR may keep more than one copy of this submission for purpose of security, back-up and preservation:
(http://eprints.undip.ac.id)



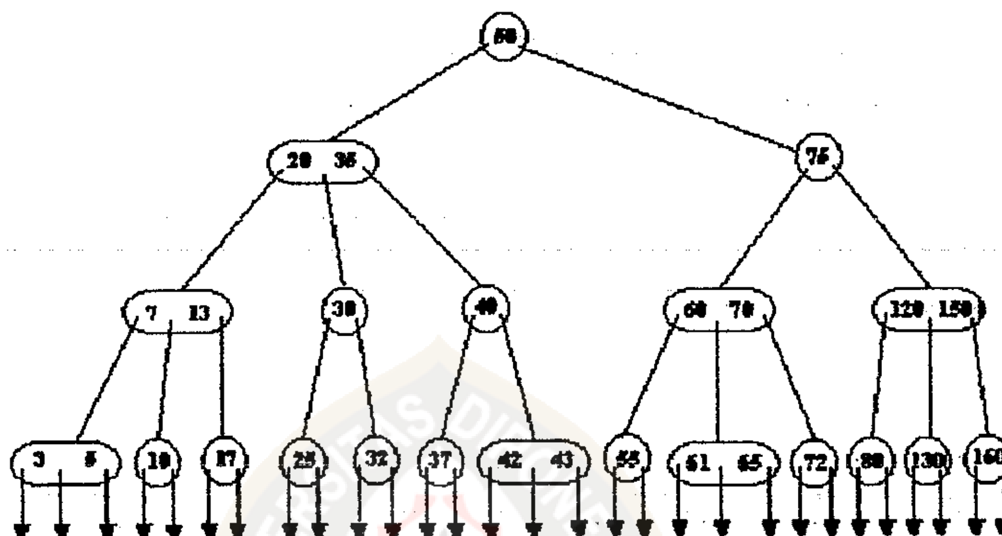
Gambar 6. Pohon telusur banyak cabang atas-bawah orde 3

Gambar 6. di atas adalah contoh **pohon telusur banyak cabang atas-bawah** (*top down multiway search tree*). Pohon demikian memiliki karakteristik yaitu :

a) sebarang **simpul tak penuh** (*nonfull node*) merupakan daun.

b) **semi-daun** (*semileaf*) merupakan simpul penuh, atau jika bukan simpul penuh maka haruslah merupakan daun.

This document is Undip Institutional Repository Collection. The author(s) or copyright owner(s) agree that UNDIP-IR may, without changing the content, translate the submission to any medium or format for the purpose of preservation. The author(s) or copyright owner(s) also agree that UNDIP-IR may keep more than one copy of this submission for purpose of security, back-up and preservation:
(http://eprints.undip.ac.id)



Gambar 7. Pohon telusur banyak cabang seimbang orde 3 (**Pohon-B**)

Gambar 7. merupakan contoh lain dari pohon telusur banyak cabang yang berorde 3. Pohon di atas bukan merupakan pohon telusur banyak cabang atas-bawah, karena terdapat empat simpul yang memiliki satu kunci dan subpohon yang tak kosong. Namun demikian, pohon di atas memiliki karakteristik khusus yang disebut seimbang (*balanced*), yaitu semua semi-daun berada dalam level yang sama (level 3).

This document is Undip Institutional Repository Collection. The author(s) or copyright owner(s) agree that UNDIP-IR may, without changing the content, translate the submission to any medium or format for the purpose of preservation. The author(s) or copyright owner(s) also agree that UNDIP-IR may keep more than one copy of this submission for purpose of security, back-up and preservation: (http://eprints.undip.ac.id)

Definisi **3.1.1 :**

Suatu pohon telusur *m* - cabang (*m-way search tree*), memenuhi sifat-sifat sebagai berikut :

1). Akar pohon paling banyak memiliki m subpohon (cabang) dan memiliki struktur berikut ini : m, (A_1, K_1) ,(A_2, K_2), (A_3, K_3), . . . ,(A_{m-1}, K_{m-1}), A_m dengan A_i, $1 \leq i \leq m$, merupakan pointer yang menunjuk ke subpohon, dan K_i , $1 \leq i < m$, merupakan nilai kunci-kuncinya.

2). $K_i < K_{i+1}$, $1 \leq i < m-1$.

3). Nilai setiap kunci yang berada dalam subpohon A_i, $1 \leq i \leq m$, lebih kecil dari K_i, $1 \leq i < m$.

4). Nilai setiap kunci yang berada dalam subpohon A_m lebih besar dari K_{m-1}.

5). Subpohon A_i , $1 \leq i \leq m$, juga merupakan pohon telusur m-cabang.

3.2 Pencarian Pohon Telusur Banyak Cabang

Diasumsikan simpul *node*(p) memuat medan-medan (*fields*) *numtrees*(p) (yang nilai bilangan bulatnya lebih kecil atau sama dengan orde (*order*) pohon, m, dan sama dengan banyaknya subpohon dari *node*(p)), *son*(p,1) sampai dengan *son*(p, *numtrees*(p)) (yang nilainya adalah pointer-pointer yang menunjuk ke subpohon-subpohon dari *node*(p)) dan $k(p,1)$ sampai dengan $k(p$, *numtrees*$(p) - 1)$ (yang nilainya adalah kunci-kunci yang termuat dalam *node*(p) berdasarkan urutan naik). Subpohon yang ditunjuk oleh *son*(p, i) (untuk i antara 2 sampai dengan

This document is Undip Institutional Repository Collection. The author(s) or copyright owner(s) agree that UNDIP-IR may, without changing the content, translate the submission to any medium or format for the purpose of preservation. The author(s) or copyright owner(s) also agree that UNDIP-IR may keep more than one copy of this submission for purpose of security, back-up and preservation:
(http://eprints.undip.ac.id)

numtrees(p) – 1) memuat seluruh kunci dalam pohon antara $k(p, i-1)$ dan $k(p, i)$. ***son***(p,1) menunjuk ke subpohon yang hanya memuat kunci-kunci yang lebih kecil dari $k(p,1)$, dan ***son***(p, ***numtrees***(p) – 1) menunjuk ke subpohon yang hanya memuat kunci-kunci yang lebih besar $k(p$, ***numtrees***(p) – 1).

Diasumsikan pula suatu fungsi ***nodesearch***(p, ***key***) yang mengembalikan baik bilangan bulat terkecil j sedemikian sehingga ***key*** $\leq k(p, j)$ atau ***numtrees***(p) jika ***key*** lebih besar dari setiap kunci dalam ***node***(p). Berikut adalah algoritma rekursif untuk fungsi ***search***(***tree***, ***key***) yang mengembalikan pointer yang menunjuk ke simpul yang memuat ***key*** (atau *Nil* jika tidak diperoleh simpul dalam pohon) dan men-*set* variabel global ***position*** untuk posisi kunci dalam simpul tersebut:

```
p := tree
if p = nil
   then begin
         search  := nil
         position  := 0
      end { then begin }
   else begin
         i  := nodesearch(p,key)
         if i = numtrees(p) then
            search:=search(son(p,numtrees(p)),key)
         else begin
              if key = k(p,i)
              then begin
                    search  := p
                    position := i
                 end { then begin }
              else search := search(son(p,i), key)
            end { else begin }
      end { else begin }
```


This document is Undip Institutional Repository Collection. The author(s) or copyright owner(s) agree that UNDIP-IR may, without changing the content, translate the submission to any medium or format for the purpose of preservation. The author(s) or copyright owner(s) also agree that UNDIP-IR may keep more than one copy of this submission for purpose of security, back-up and preservation:
(http://eprints.undip.ac.id)

Berikut ini adalah algoritma versi *non-rekursif* :

```
p := tree
found := false
 while ( p <> nil ) and ( not found )
        do begin
               i := nodesearch(p,key)
               if i = numtrees(p) then
                   p:= son(p,numtrees(p))
               else begin
                       if key  = k(p,i)
                             then found := true
                       else p := son(p,i)
                     end { else begin }
           end { while ... do begin }
 if found
     then begin
              search  := p
              position  := i
           end { then begin }
     else begin
              search  := nil
              position :=0
           end { else begin }
```

Fungsi ***nodesearch*** berperan untuk mengalokasikan kunci terkecil dalam simpul yang lebih besar atau sama dengan argumen yang dicari.

3.3 Penerapan Pohon Telusur Banyak Cabang

Jika pohon telusur banyak cabang berada dalam penyimpan luar, pointer yang menunjuk ke suatu simpul merupakan alamat penyimpan luar yang menentukan titik awal dari blok penyimpan. Blok penyimpan yang membentuk simpul harus dibaca ke penyimpan dalam (misal memori) sebelum medan-medan ***numtrees***, ***k*** ,

This document is Undip Institutional Repository Collection. The author(s) or copyright owner(s) agree that UNDIP-IR may, without changing the content, translate the submission to any medium or format for the purpose of preservation. The author(s) or copyright owner(s) also agree that UNDIP-IR may keep more than one copy of this submission for purpose of security, back-up and preservation:
(http://eprints.undip.ac.id)

atau *son* dapat diakses. Diasumsikan rutin (*routine*) *directread(p,block)* membaca suatu simpul pada alamat penyimpan luar *p* ke dalam buffer penyimpan dalam *block*. Diasumsikan pula medan-medan *numtrees*, *k*, dan *son* dalam buffer diakses berturut-turut sebagai berikut, yaitu *block.numtrees*, *block.k*, dan *block.son*, serta fungsi *nodesearch* dimodifikasi untuk menerima blok penyimpan dalam . Berikut adalah algoritma non-rekursif untuk pohon telusur banyak yang tersimpan di luar :

```
p := tree
found := false
 while ( p <> nil ) and ( not found )
        do begin
               directread(p,block)
               i := nodesearch(block,key)
               if i = block.numtrees then
                    p:= block.son(block.numtrees)
               else begin
                      if key  =  block.k(i)
                           then found := true
                      else p := block.son(i)
                    end { else begin }
           end { while ... do begin }
 if found
     then begin
              search  := p
              position  := i
           end { then begin }
     else begin
               search  := nil
               position :=0
           end { else begin }
```

This document is Undip Institutional Repository Collection. The author(s) or copyright owner(s) agree that UNDIP-IR may, without changing the content, translate the submission to any medium or format for the purpose of preservation. The author(s) or copyright owner(s) also agree that UNDIP-IR may keep more than one copy of this submission for purpose of security, back-up and preservation:
(http://eprints.undip.ac.id)

3.4 Kunjungan Pohon Telusur Banyak Cabang

Salah satu bentuk operasi umum dalam struktur data adalah mengakses setiap elemen dari (dalam) struktur yang bersangkutan menurut suatu urutan yang tetap. Berikut ini adalah algoritma rekursif *traverse*(*tree*) yang berfungsi untuk mengunjungi pohon telusur banyak cabang dan mencetak kunci-kuncinya berdasarkan urutan naik.

```
if tree <> nil
   then begin
            nt := numtrees(tree)
            for i := 1 to nt – 1
                do begin
                        traverse(son(tree,i))
                        writeln(k(tree,i))
                    end { for . . . do begin }
             traverse(son(tree,nt))
         end {then begin}
```

Jika setiap simpul adalah suatu blok penyimpan luar dan *tree* adalah alamat penyimpan luar simpul akar, maka suatu simpul harus dibaca ke dalam *internal memory* sebelum medan *son* atau medan *k*-nya dapat diakses. Berikut adalah modifikasi algoritmanya.

```
if tree <> nil
then begin
          directread(tree,block)
          nt := block.numtrees
          for i := 1 to nt – 1
              do begin
                      traverse(block.son(i))
                      writeln(block.k(i))
                  end { for . . . do begin }
          traverse(block.son(nt))
      end {then begin}
```

This document is Undip Institutional Repository Collection. The author(s) or copyright owner(s) agree that UNDIP-IR may, without changing the content, translate the submission to any medium or format for the purpose of preservation. The author(s) or copyright owner(s) also agree that UNDIP-IR may keep more than one copy of this submission for purpose of security, back-up and preservation:
(http://eprints.undip.ac.id)

Operasi yang biasa dilakukan pada pohon telusur banyak lainnya dan masih sangat berhubungan dengan kunjungan (*traversal*), adalah *direct sequential access*. Hal ini berhubungan dengan cara mengakses kunci dengan cepat berdasarkan suatu kunci yang lokasinya di dalam pohon diketahui. Diasumsikan, telah diketahui lokasi suatu kunci *k*1 dengan mencari (menelusuri) pohon dan kunci tersebut berada pada posisi *k*(*m*1, *i*1). suksesor (*successor*) *k*1 dapat ditemukan dengan menjalankan fungsi berikut yaitu *next*(*m*1, *i*1) (*nilkey* adalah nilai khusus yang mengindikasikan bahwa kunci yang dicari tidak ditemukan).

```
p := son(m1, i1+1)
q := nil
while p<> nil
        do begin
                q := p
                p := son(p, 1)
            end { while . . . do begin }
if q  <> nil
        then next := k(q, 1)
        else if i1 < numtrees(m1) – 1
                then next := k(m1, i1+1)
             else next := nilkey
```

Algoritma di atas didasarkan pada kenyataan bahwa suksesor *k*1 adalah kunci pertama dalam subpohon yang mengikuti *k*1 dalam simpul *node*(*m*1), atau jika subpohon tersebut kosong [*son*(*m*1, *i*1+1) sama dengan nil] dan jika *k*1 bukan merupakan kunci terakhir dalam simpulnya [*i*1 < *numtrees*(*m*1) – 1], maka suksesor adalah kunci setelah *k*1 dalam simpul *node*(*m*1).

This document is Undip Institutional Repository Collection. The author(s) or copyright owner(s) agree that UNDIP-IR may, without changing the content, translate the submission to any medium or format for the purpose of preservation. The author(s) or copyright owner(s) also agree that UNDIP-IR may keep more than one copy of this submission for purpose of security, back-up and preservation:
(http://eprints.undip.ac.id)

Namun demikian, jika *k*1 adalah kunci terakhir dalam simpulnya dan jika subpohon berikutnya kosong, maka suksesornya hanya dapat ditemukan dengan cara menelusuri pohon (naik) ke atas (***backing up the tree***). Diasumsikan setiap simpul memuat dua medan tambahan : ***medan ayah*** (***father field***) yang merupakan pointer menunjuk ke simpul ayah dan ***medan indeks*** (***index field***) yang menentukan simpul berada dalam cabang yang keberapa dari simpul ayah. Berikut adalah algoritma ***successor***(*m*1, *i*1) untuk menemukan suksesor kunci yang dicari dalam posisi *i*1 dari simpul yang ditunjuk oleh *m*1 :

```
p := son(m1, i1+1)
if ( p <> nil ) or ( i1 < numtrees(m1) - 1)
   then successor := next(m1, i1)
   else begin
          f := father(m1)
          i := index(m1)
          while ( i = numtrees(f) ) and (f <> nil )
                 do begin
                        i := index(f)
                        f :=father(f)
                    end { while . . . do begin }
          if f = nil
            then successor := nilkey
          else successor := k(f, i)
       end { else begin }
```


This document is Undip Institutional Repository Collection. The author(s) or copyright owner(s) agree that UNDIP-IR may, without changing the content, translate the submission to any medium or format for the purpose of preservation. The author(s) or copyright owner(s) also agree that UNDIP-IR may keep more than one copy of this submission for purpose of security, back-up and preservation:
(http://eprints.undip.ac.id)

3.5 Penyisipan Pohon Telusur Banyak Cabang

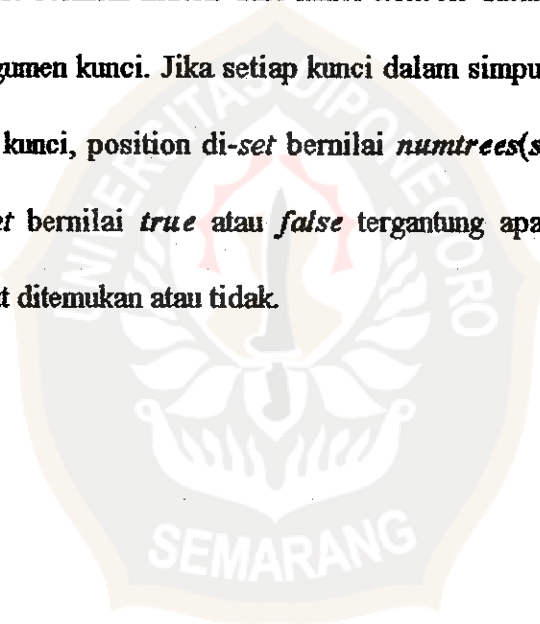
Ada dua teknik penyisipan untuk pohon telusur banyak cabang. Yang pertama analog dengan penyisipan pada pohon telusur biner dan menghasilkan pohon telusur banyak cabang atas - bawah (*top-down multiway search tree*). Teknik yang kedua adalah teknik baru yang menghasilkan suatu jenis khusus yaitu pohon telusur banyak cabang seimbang.

Langkah pertama untuk kedua teknik di atas adalah proses pencarian untuk argumen kunci. Berikut adalah prosedur *find* untuk proses pencarian argumen kunci:

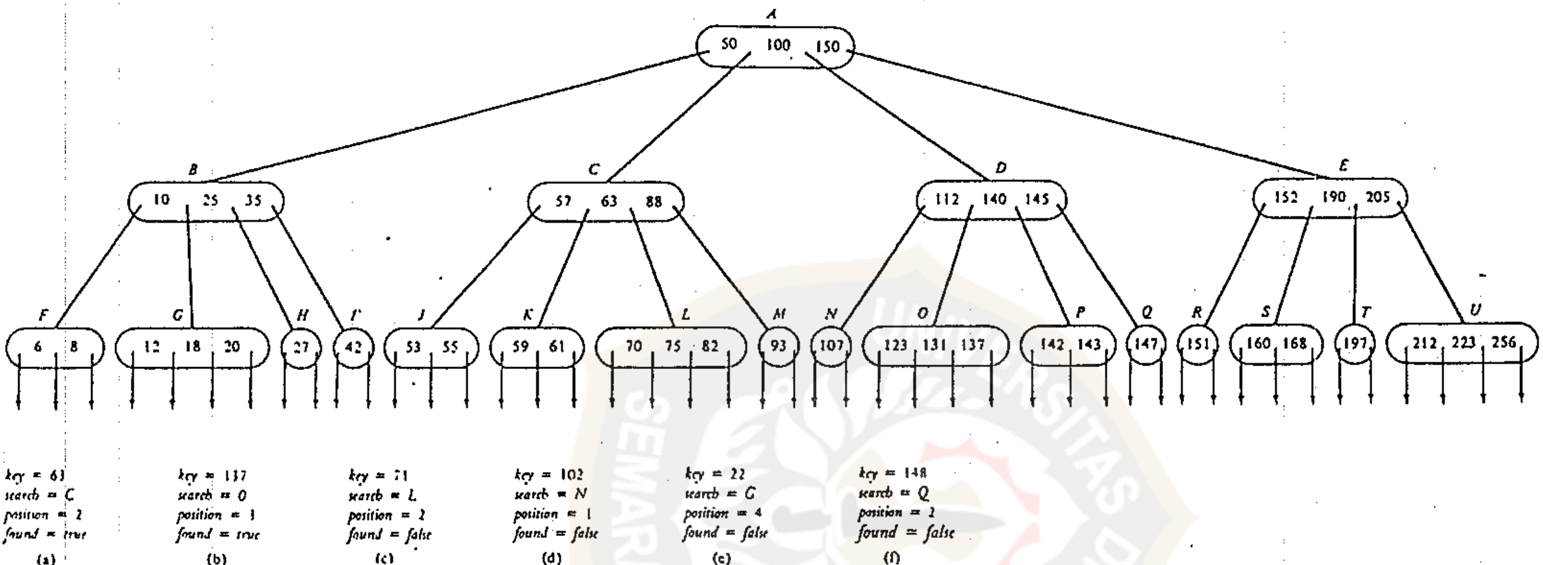
```
procedure find ( tree : nodeptr; key : keytype; var search : nodeptr;
                 var position : integer; var found : boolean )
q := nil
p := tree
found:=false
while  (p<>nil ) and ( not found )
     do begin
          i  := nodesearch(p, key)
          q := p
          if i = numtrees(p) then
               p:= son(p,numtrees(p))
          else begin
                    if  key = k(p,i)
                         then found := true
                    else p := son(p,i)
               end { else begin }
          end { while . . . do begin }
if found
     then search := p { kunci ditemukan dalam simpul node(p)}
else search :=q   { p bernilai nil; q menunjuk semileaf }
position:= i
```

This document is Undip Institutional Repository Collection. The author(s) or copyright owner(s) agree that UNDIP-IR may, without changing the content, translate the submission to any medium or format for the purpose of preservation. The author(s) or copyright owner(s) also agree that UNDIP-IR may keep more than one copy of this submission for purpose of security, back-up and preservation:
(http://eprints.undip.ac.id)

Dalam prosedur di atas, jika argumen kunci ditemukan dalam pohon, maka ***search*** di-*set* untuk menunjuk ke simpul yang memuat kunci tersebut, dan ***position*** bernilai bilangan bulat yang memberikan informasi posisi kunci tersebut. Sebaliknya, jika argumen kunci tidak ditemukan dalam pohon, search di-*set* untuk menunjuk ke semi-daun yang memuat kunci tersebut seandainya kunci itu ada, dan ***position*** di-*set* bernilai indeks dari kunci terkecil dalam ***node***(***search***) yang lebih besar dari argumen kunci. Jika setiap kunci dalam simpul ***node***(***search***) lebih kecil dari argumen kunci, position di-*set* bernilai *numtrees*(***search***). Variabel *boolean*, ***found*** , di-*set* bernilai ***true*** atau ***false*** tergantung apakah argumen kunci dalam pohon tersebut ditemukan atau tidak.



This document is Undip Institutional Repository Collection. The author(s) or copyright owner(s) agree that UNDIP-IR may, without changing the content, translate the submission to any medium or format for the purpose of preservation. The author(s) or copyright owner(s) also agree that UNDIP-IR may keep more than one copy of this submission for purpose of security, back-up and preservation:
(http://eprints.undip.ac.id)



Gambar 8. Hasil proses pencarian prosedur ***find*** pada pohon telusur banyak cabang atas-bawah

This document is Undip Institutional Repository Collection. The author(s) or copyright owner(s) agree that UNDIP-IR may, without changing the content, translate the submission to any medium or format for the purpose of preservation. The author(s) or copyright owner(s) also agree that UNDIP-IR may keep more than one copy of this submission for purpose of security, back-up and preservation:
(http://eprints.undip.ac.id)

Langkah kedua dari prosedur penyisipan berlaku jika kunci yang dicari tidak ditemukan dan jika *node*(*search*) tidak penuh -- yaitu jika *numtrees*(*search*) < *m*, dimana *m* adalah orde dari pohon. Langkah kedua terdiri dari proses penyisipan kunci baru (dan rekaman) ke dalam *node*(*search*).

Misalkan *insrec*(*p*, *i*, *rec*) adalah sebuah rutin untuk menyisipkan rekaman *rec* pada posisi *i* dari simpul *node*(*p*) yang sesuai. Berikut adalah prosedur *insleaf*(*search*, *position*, *key*, *rec*) yang merupakan langkah kedua untuk proses penyisipan :

```
nt := numtrees(search)
numtrees(search) := nt + 1
for i:= nt downto position + 1
        do k(search,i):= k(search, i – 1)
k(search,position):= key
insrec(search,position,rec)
```

Gambar 9.a mengilustrasikan simpul-simpul yang ditemukan oleh prosedur *find* dalam gambar 8.d dan f dengan kunci-kunci baru yang disisipkan.

Jika langkah 2 cocok (yaitu jika simpul daun tak penuh dimana kunci yang dapat disisipkan telah ditemukan), maka kedua prosedur penyisipan tersebut berakhir. Kedua teknik tersebut berbeda pada langkah ketiga yang diperlukan pada saat prosedur *find* menempati semi-daun (yang) penuh.

Teknik penyisipan yang pertama, yang menghasilkan pohon telusur banyak cabang atas-bawah, memiliki tahapan yang sama dengan algoritma penyisipan pada pohon telusur biner. Pertama kali, dilakukan tahap pengalokasian simpul baru,

This document is Undip Institutional Repository Collection. The author(s) or copyright owner(s) agree that UNDIP-IR may, without changing the content, translate the submission to any medium or format for the purpose of preservation. The author(s) or copyright owner(s) also agree that UNDIP-IR may keep more than one copy of this submission for purpose of security, back-up and preservation:
(http://eprints.undip.ac.id)

kemudian dilanjutkan penyisipan kunci dan rekaman ke dalam simpul baru, dan diakhiri dengan penempatan simpul baru dengan tepat sebagai anak dari ***node(search)***. Teknik ini memerlukan fungsi ***maketree(key, rec)*** untuk mengalokasikan simpul dan men-***set m*** pointer dalam simpul tersebut menjadi ***nil***, medan ***numtrees*** bernilai 2, dan medan kunci pertamanya menjadi sama dengan ***key.maketree***. Terakhir, teknik ini menggunakan rutin ***insrec*** untuk menyisipkan rekaman secara tepat dan akhirnya mengembalikan sebuah pointer ke simpul baru yang dialokasikan. Berikut adalah rutin ***insfull*** untuk menyisipkan kunci pada saat semi-daun yang cocok penuh:

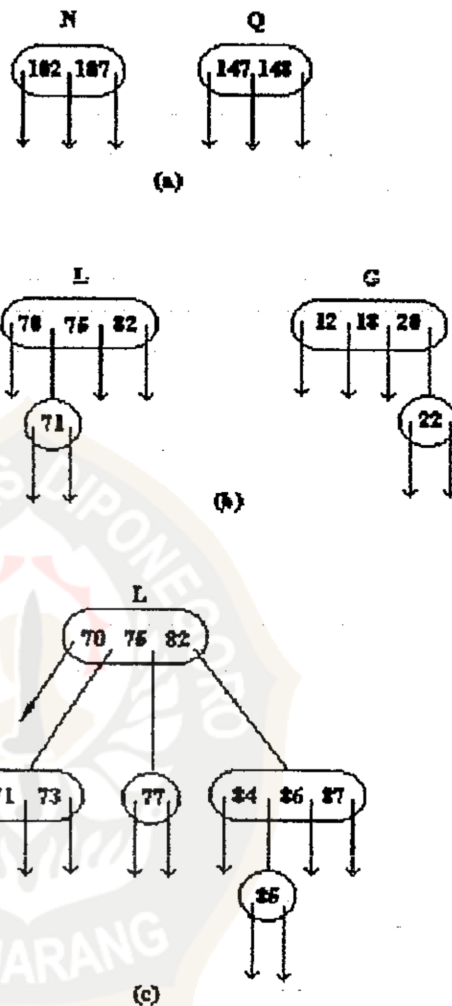
$$p := maketree(key, rec)$$
$$son(search, position) := p$$

Jika medan *father* dan *index* dipasang dalam setiap simpul, operasi berikut ini diperlukan :

$$father(p) := search$$
$$index(p) := position$$

Gambar 9.b mengilustrasikan hasil dari penyisipan kunci-kunci berturut-turut 71 dan 22, ke dalam simpul-simpul yang ditemukan oleh prosedur ***find*** dalam gambar 8.c dan e. Gambar 9.c mengilustrasikan penyisipan berikutnya dari kunci-kunci 86, 77, 87, 84, 85, dan 73 dalam urutan demikian.

This document is Undip Institutional Repository Collection. The author(s) or copyright owner(s) agree that UNDIP-IR may, without changing the content, translate the submission to any medium or format for the purpose of preservation. The author(s) or copyright owner(s) also agree that UNDIP-IR may keep more than one copy of this submission for purpose of security, back-up and preservation:
(http://eprints.undip.ac.id)



Gambar 9. Penyisipan kunci pada pohon telusur banyak cabang atas-bawah

Berikut adalah keseluruhan langkah algoritma tahap pencarian dan penyisipan yang menghasilkan pohon telusur banyak cabang untuk teknik pertama :

```
if tree = nil
   then begin
           tree := maketree(key, rec)
           search := tree
           position := 1
        end { then begin }
```

This document is Undip Institutional Repository Collection. The author(s) or copyright owner(s) agree that UNDIP-IR may, without changing the content, translate the submission to any medium or format for the purpose of preservation. The author(s) or copyright owner(s) also agree that UNDIP-IR may keep more than one copy of this submission for purpose of security, back-up and preservation:
(http://eprints.undip.ac.id)

```
else begin
        find(tree, key, search, position, found)
        if not found
          then if numtrees(search) < m
                then insleaf(search, position, key, rec)
                else begin
                        p := maketree(key, rec)
                        son(search, position) := p
                        search := p
                        position :=1
                end { else begin }
    end { else begin }
```

3.6 Analisis Pohon Telusur Banyak Cabang

Faktor cabang adalah banyaknya cabang yang berasal dari simpul. Faktor cabang biasanya sama dengan banyaknya kunci yang dapat ditempatkan dalam sebuah simpul atau satu lebih banyak dari banyaknya kunci. Untuk kasus pohon-B, digunakan yang terakhir.

Untuk analisis berikut, diasumsikan, setiap simpul memiliki faktor cabang $s + 1$, dan banyaknya kunci per simpul s. Banyaknya kunci yang dapat ditempatkan dalam sebuah pohon pada ketinggian h adalah

$$s + s(s+1) + s(s+1)^2 + \ldots + s(s+1)^h. \qquad \text{(a)}$$

Jika faktor cabang $s + 1$ dibuat sama dengan m, tinggi h dapat ditentukan dengan menyamakan persamaan (a) sama dengan n dan menyelesaikannya untuk tinggi h, yaitu

$$n = s + sm + sm^2 + \ldots + sm^h \qquad \text{(b)}$$

This document is Undip Institutional Repository Collection. The author(s) or copyright owner(s) agree that UNDIP-IR may, without changing the content, translate the submission to any medium or format for the purpose of preservation. The author(s) or copyright owner(s) also agree that UNDIP-IR may keep more than one copy of this submission for purpose of security, back-up and preservation:
(http://eprints.undip.ac.id)

Jika rumus untuk jumlahan deret geometrik digunakan, (b) dapat ditulis kembali menjadi

$$n = s\left(\frac{m^{h+1}-1}{m-1}\right) = \frac{m-1}{m-1}\left(m^{h+1}-1\right) = m^{h+1}-1 \qquad \text{(c)}$$

(c) dapat ditulis kembali menjadi $n+1 = m^{h+1}$ (d)

Tinggi h, dapat dinyatakan sebagai fungsi dari n dan m dengan mengambil logaritma untuk kedua sisi persamaan (d) terhadap m, yaitu,

$$\log_m(n+1) = h+1 \qquad \text{(e)}$$

(e) dapat dinyatakan kembali sebagai

$$h \cong \log_m(n+1) - 1 \qquad \text{(f)}$$

atau $O(\log_m n)$.

This document is Undip Institutional Repository Collection. The author(s) or copyright owner(s) agree that UNDIP-IR may, without changing the content, translate the submission to any medium or format for the purpose of preservation. The author(s) or copyright owner(s) also agree that UNDIP-IR may keep more than one copy of this submission for purpose of security, back-up and preservation:
(http://eprints.undip.ac.id)