MODUL IV
Struktur Data

Judul	METODA PENCARIAN	
Penyusun	**Distribusi**	**Perkuliahan**
Nixon Erzed	Teknik Informatika Universitas Esa Unggul	Pertemuan – IV online

Tujuan :

Setelah mengikuti kuliah ini, mahasiswa dapat memahami berbagai jenis algoritma searching, dan mahasiswa dapat mengimplementasikannya algoritma searching dalam kasus sederhana

Materi :

1. Pengertian Umum
2. Pencarian Dasar
3. Sequential Searching (pencarian squens)
   a. Tanpa Boolean
      - Tanpa sentinel
      - Dengan sentinel

   b. Menggunakan boolean

2. Binary Searching

   Pohon Pencarian Biner

PENCARIAN (SEARCHING)

Pengertian umum

Saat ini pencarian sering diinterpretasikan sebagai pencarian suatu informasi di cyber space. Mesin pencari menjadi aplikasi yang popular. Mesin pencari adalah sebuah fasilitas cloud (web) yang bisa mencari links dari suatu situs yang menyediakan informasi yang diinginkan. Terdapat banyak mesin pencari (search engine) yang tersedia di cyber space, yang paling popular adalah Google, sehingga pencarian di cyber space sudah mendapatkan sebutan popular yaitu Googling. Selain google, ada berbagai macam search engine yang bisa kita gunakan dalam searcing, yaitu ; yahoo, bing, altavista, lycos, astaga, msn, dan lain sebagainya.

Pencarian (searhing) merupakan proses yang sangat penting dalam pengolahan data. Proses pencarian dasar adalah menemukan nilai(data) tertentu didalam sekumpulan data yang bertipe sama. Himpunan data-data bertype sama tersebut dikenal sebagai ruang pencarian. Secara fisik ruang pencarian sederhana memanfaatkan array. Pencarian data sering juga disebut table look-up atau storage and retrieval information adalah suatu proses untuk mengumpulkan sejumlah informasi di dalam pengingat komputer dan kemudian mencari kembali informasi yang diperlukan secepat mungkin.

Untuk pengertian yang lebih luas, searching adalah proses mendapatkan (retrieve) informasi berdasarkan kunci tertentu dari sejumlah informasi yang telah disimpan. Ruang penyimpanan informasi tersebut juga adalah ruang pencarian, yang memerlukan ruang data yang lebih kompleks yang memungkinkan pencarian informasi dengan metoda tertentu dapat dilakukan.

Algoritma pencarian (searching algorithm) adalah algoritma yang menerima sebuah argumen kunci dan dengan langkah-langkah tertentu akan mencari rekaman dengan kunci tersebut. Setelah proses pencarian dilaksanakan, akan diperoleh salah satu dari dua kemungkinan, yaitu data yang dicari ditemukan (successful) atau tidak ditemukan (unsuccessful).

Metode pencarian data dapat dilakukan dengan dua cara yaitu pencarian internal (internal searching) dan pencarian eksternal (external searching). Pada pencarian internal, semua rekaman yang diketahui berada dalam media penyimpanan internal komputer. Sedangkan pada pencarian eksternal, tidak semua rekaman yang diketahui berada dalam pengingat komputer, tetapi ada sejumlah rekaman yang tersimpan dalam penyimpan luar misalnya media penyimpanan yang berada dalam sistem computer lain yang tersambung dalam suatu sistem jaringan.

Selain itu metode pencarian data juga dapat dikelompokkan menjadi pencarian statis (static searching) dan pencarian dinamis (dynamic searching). Pada pencarian statis, banyaknya ruang data yang diketahui dianggap tetap, pada

pencarian dinamis, banyaknya ruang data yang diketahui bisa berubah-ubah yang disebabkan oleh penambahan atau penghapusan suatu ruang data.

Silahkan anda cari tahu, bagaimana ruang pencarian dibentuk oleh mesin pencari google.

Contoh pencarian

a. Siapa nama mahasiswa dengan NIM 20170801056
b. Siapa saja yang mendapat nilai matematika diskrit ≥ 85
c. Apakah ada mahasiswa yang mendapat nilai 100
d. Ada berapa banyak kata “algoritma” dalam sebuah artikel
e. Saya ingin menuju halaman buku yang membahas tentang algoritma pencarian
f. Mencari cara memasak ikan hiu di cyber space (internet)
g. Menemukan nomor telepon seseorang pada buku telepon
h. Mencari istilah dalam kamus
i. Mencari informasi suatu kata dalam kamus digital

Dalam proses pemrograman serching/pencarian biasa digunakan untuk :

a. proses update atau penghapusan data sebelumnya melakukan proses pencarian data.
b. Penyisipan data pada sekumpulan data, jika data sudah ada maka proses penyisipan tidak diperkenankan. Jika data tidak ada maka proses penyisipan dilakukan tujuan digunakan agar tidak terjadi duplikasi data.

PENCARIAN DASAR

Pencarian dasar diperlukan untuk mencari informasi khusus dari tabel pada saat lokasi yang pasti dari informasi tersebut sebelumnya tidak diketahui.

- Data pada tabel biasanya disimpan dengan menggunakan tipe data Array
- Array memungkinkan untuk menyimpan nilai yang bertipe sama.
- Operasi yang umum dalam array adalah Sequential Search dan Binary search.
- Perbedaan dari kedua teknik ini terletak pada keadaan data.
- Hasil pencarian dapat bermacam-macam, bergantung pada spesifikasi rinci dari persoalan.Misalnya keluaran yang diinginkan adalah pesan (message) bahwa x ditemukan atau tidak ditemukan di dalam larik.

  Contoh:

  Write (‘ditemukan’) atau

  Write (‘tidak ditemukan’)

- Hasil pencarian adalah indeks elemen larik. Jika x ditemukan, maka indeks elemen larik tempat x berada diisikan ke dalam suatu identifier, misalnya idx. Jika x tidak terdapat dalam larik, maka idx diisi dengan harga khusus, misalnya idx = -1.
- Hasil pencarian adalah sebuah nilai boolean yang menyatakan status hasil pencarian. Misalnya “ketemu” diisi dengan nilai true dan sebaliknya.
- Apabila yang dicari terdapat lebih dari satu banyaknya di dalam larik (duplikasi data), maka hanya data yang pertama kali ditemukan yang diacu algoritma pencarian selesai.ang dimaksud sebagai pencarian di array adalah proses pencarian suatu elemen array dengan ketentuan/nilai tertentu.

Bedasarkan lingkup ruang data, yang menjadi area pencarian, operasi dasar pencarian dapat menerapkan : pemrosesan traversal, dan non traversal.

Pemrosesan Traversal (Penelusuran)

Operasi penelusuran adalah operasi dasar dalam pencarian. Prinsip traversal adalah mengunjungi seluruh ruang pencarian dari ruang pertama hingga ruang terakhir. Karena traversal akan mengunjungi seluruh elemen ruang, maka penelusuran secara sistematik dapat dilakukan dengan menggunakan algoritma repetitive sederhana, dengan indeks ruang sebagai referensi dan statement increment/decrement sebagai mesin pencarian.

Contoh pencarian yang memanfaatkan proses traversal, antara lain:

a. Pencarian nilai ekstrim (maksimum atau minimum) dari sekumpulan data tidak terurut.
b. Pencarian nilai tertentu dari sekumpulan data tidak terurut

Contoh

Algoritma pencarian nilai minimum dari sebuah X[i]

```
Prosedur NilaiMin( n : jumlah elemen )
begin
      min :=  X [ 1 ]
      k := 1
      For  i :=  2  to  n  do
        begin
          if   X [ i ] <  min  then
               min  :=  X [ i ]
               k := i
          end-if
      end-for
end-prosedur
```

Jika dimiliki data array X[i] dengan data sebagai berikut :

25	*75*	*56*	*65*	*18*	*78*	*5*	*23*	*12*	*31*
i = 1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*

inisialisasi min = 25 k = 1

I	2	3	4	5	6	7	8	9	10
If X [i] < min	25>75 False	25 > 56 False	25 > 65 False	25 > 18 True	18 > 78 False	18 > 8 True	8 > 23 False	8 > 12 False	8 > 31 False
min ⇐ X [i]	25	25	25	18	18	5	5	5	5
k ⇐ i	1	1	1	5	5	7	7	7	7
Next i	3	4	5	6	7	8	9	10	selesai

Hasil pencarian adalah nilai minimum min = 5 di k = 7

Jika diimplementasikan dalam struktur while..do akan dihasilkan algoritma sbb. :

```
Prosedur NilaiMin( n : jumlah elemen )
 begin
    min := X [ 1 ]
    k := 1
    i  :=  2
    While  i <=  n  do
       begin
          if   X [ i ] <  min  then
                min  :=  X [ i ]
                k := i
          end-if
          i :=  i + 1
    end-while
 end-prosedur
```

Pengelompokkan Metoda Pencarian

Algoritma Pencarian yang akan dibahas adalah:

1. Algoritma Pencarian Squens/Linear/Beruntun (sequential search)
   a. Tanpa Boolean
      - Tanpa sentinel
      - Dengan sentinel
   b. Menggunakan boolean
2. Algoritma Pencarian bagi dua (Binary search)

Sequential Searching (pencarian squens)

Pencarian dilakukan secara berurutan mulai dari elemen pertama sampai ditemukan elemen yang dicari atau menemui elemen terakhir. Untuk terminasi pencarian terdapat beberapa kemungkinan :

a. Menghentikan pencarian setelah dicapai elemen terakhir

   Untuk type ini, digunakan jumlah (maksimum) elemen array sebagai terminasi akhir. Artinya pencarian akan diakhiri setelah penelusuran mencapai elemen terakhir. Implementasinya dikenal sebagai pencarian sequens tanpa Boolean dan tanpa sentinel.

b. Menghentikan pencarian begitu ditemukan elemen yang dicari

   Untuk type ini, digunakan jumlah (maksimum) elemen array atau kondisi data ditemukan bernilai benar, sebagai terminasi akhir. Artinya pencarian akan diakhiri jika data yang dicari ditemukan atau setelah penelusuran mencapai elemen terakhir.

   Implementasinya dikenal sebagai pencarian sequens dengan Boolean

c. Menghentikan pencarian begitu ditemukan elemen yang dicari dan array diberi penanda akhir dengan data yang dicari (sentinel)

   Untuk type ini, array ditambahi satu elemen akhir yang bernilai sama dengan data yang dicari, sehingga cukup digunakan satu kondisi yaitu data ditemukan bernilai benar, sebagai terminasi akhir. Setelah loop pencarian berakhir, elemen yang ditemukan diperiksa. Jika elemen tersebut bukan elemen akhir maka pencarian sukses. Implementasinya dikenal sebagai pencarian sequens tanpa Boolean dengan sentinel

d. Pencarian Sequential dapat digunakan pada data dalam keadaan acak atau tidak terurut maupun yang sudah terurut

Contoh

Terhadap array X [i] akan dilakukan operasi pencarian apakah terdapat data 59

X	21	3	54	17	121	11	32	51
	@1	@2	@3	@4	@5	@6	@7	@8

Sebelum dilakukan pencarian, dilakukan penambahan elemen pada array X [i] yaitu X [8+1] = 59. Dalam hal ini data 59 dijadikan sentinel.

Sehingga dihasilkan array sebagai berikut :

X	21	3	54	17	121	11	32	51	59
	@1	@2	.	.	.			@8	@9

Pencarian adalah sukses jika indeks data ketemu dan x[i] ≠ 59

Algoritma Pencarian

- Menghentikan pencarian setelah dicapai elemen atau data terakhir (bersifat traversal)
- Digunakan jumlah (maksimum) elemen array sebagai terminasi akhir.
  → pencarian diakhiri setelah penelusuran mencapai elemen terakhir.
  - tanpa senitinel
  - dengan sentinel

  → disebut juga pencarian sequens tanpa Boolean

Kasus 1 **data tanpa sentinel.**

```
Function  Qty_x ( x : <type data>  ) : integer;
→  x  adalah data dicari
     begin
    Qty_x := 0
    i  :=  1
         While  i ≤ n  do
          begin
              if   A [ i ] = x  then
                    Qty_x  :=  Qty_x + 1
              end-if
                 i :=  i + 1
          end-while
     end-prosedur
```

Kasus 2 data dengan sentinel

misalnya untuk array A[i] yang bertipe integer, dan sentinel adalah 99999 :

```
Function  Qty_x ( x : <type data>  ) : integer;
→ x  adalah data dicari
    begin
     Qty_x := 0
     i  :=  1
        While  A[ i ]<> 99999  do
         begin
            if   A [ i ] = x  then
                  Qty_x  :=  Qty_x + 1
            end-if
                 i :=  i + 1
        end-while
    end-prosedur
```

Kasus 3 Menghentikan pencarian begitu ditemukan elemen yang dicari

→ *sequens search dengan Boolean*

Terminasi akhir :
Ditemukan jumlah (maksimum) elemen array, atau
kondisi data bernilai benar.

→ pencarian akan diakhiri jika data yang dicari ditemukan atau setelah penelusuran mencapai elemen terakhir.
→ pencarian sequens dengan Boolean

```
Procedure Ada_x ( x : <type data> )
 Var    ada : Boolean
 begin
   | Ada := false
   | i  :=  1
   | While  ( i <=  n)  and  not ada  do
   |  begin
   |    | if   A [ i ] = x  then
   |    |       ada := true
   |    |  end-if
   |    | i :=  i + 1
   |  end-while
 end-prosedur
```

→ *x adalah data dicari*

Untuk mencari apakah ada data bernilai x pada array A[i]

Jika ditemukan data x maka loop diakhiri (data x yang pertama jika mungkin terdapat lebih dari satu x

Kasus 4

Mencari ada beberapa banyak data bernilai X dalam array A[i] yang diberi penanda akhir dengan data yang unik atau dapat juga data yang dicari (sentinel)

→ pencarian sequens dengan sentinel

Array ditambahi satu elemen akhir yang bernilai sama dengan data yang dicari,

→ cukup digunakan satu kondisi yaitu data ditemukan bernilai benar, sebagai terminasi akhir.

→ asumsi : jumlah elemen array > jumlah data yang ada dalam array

→ harus diketahui pointer elemen terakhir

Setelah loop pencarian berakhir, elemen yang ditemukan diperiksa. Jika elemen tersebut bukan elemen akhir maka pencarian sukses.

```
Function  Qty_x ( x : <type data>  ) : integer;        → x  adalah data dicari yaitu 22
 begin
   Qty_x := 0
   i  :=  1
   While  A [ i ] <> 99999  do
    begin
          if   A [ i ] = x  then
               Qty_x  :=  Qty_x + 1
          end-if
              i :=  i + 1
   end-while
   tulis(Qty_x)
 end-prosedur
```

Binary Searching (pencarian biner)

Pencarian dilakukan dengan cara membagi array menjadi dua bagian, data yang dicari dibandingkan dengan elemen tengah : jika sama berarti ketemu, jika yang dicari lebih besar maka pencarian dilanjutkan pada bagian/partisi atas, jika yang dicari lebih kecil maka pencarian dilanjutkan pada bagian/partisi bawah. Syarat penerapan pencarian ini adalah data mestilah dalam keadaan terurut. Kondisi diatas adalah untuk data yang terurut dari kecil ke besar (menaik).

Contoh

Terhadap array X [i] dengan jumlah elemen n = 8 akan dilakukan operasi pencarian apakah terdapat data 29

X	3	8	14	17	21	30	32	51
	@1	@2	@3	@4	@5	@6	@7	@8

Array X [i] dibagi menjadi 2 bagian

t $\Leftarrow$ (indeks bawah + indeks atas) div 2

$\Leftarrow$ (1 + 8) div 2

$\Leftarrow$ 4

sehingga dihasilkan 2 partisi yaitu : - partisi bawah X [1 .. t] dan

- partisi atas X [t+1 .. n]

Periksa X [t =4], karena (X [4] = 17) lebih kecil dari data cari = 29 maka pilih partisi atas atau X[5 .. 8]

Ulangi lagi selama indeks atas lebih besar dari indeks bawah. Secara lebih detil pencarian biner akan dijelaskan dalam Pohon Pencarian Biner

BINARY SEARCH TREE

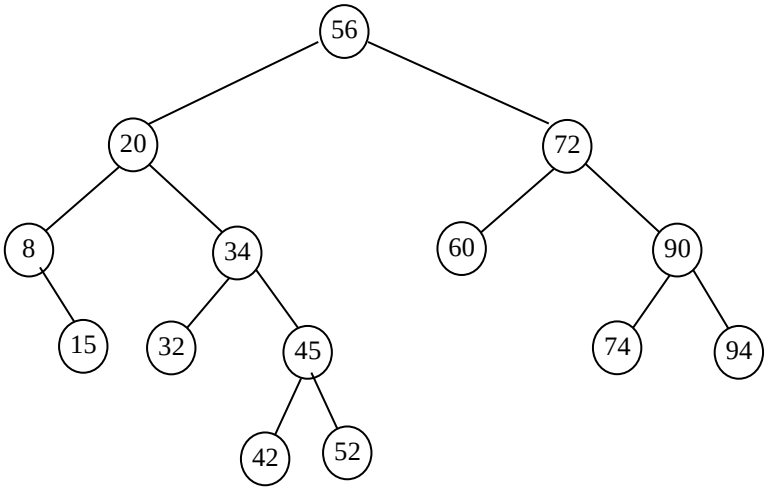
Binary Search Tree (Pohon Pencarian Biner / BST) adalah Binary Tree dengan sifat khusus, yaitu semua Left Child selalu lebih kecil dari Parent dan semua Right Child selalu lebih besar dari Parent, atau LC < Parent < RC.

Binary search tree/BST dibuat untuk mengatasi kelemahan pada binary tree biasa, yaitu kesulitan dalam searching /pencarian node terntentu dalam suatu binary tree.

Jika dimiliki data sebagai berikut :

56 20 72 34 90 74 32 45 42 94 8 52 15 60

dapat dibentuk BST sesuai dengan urutan kedatangan data sebagai berikut :

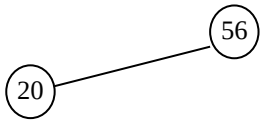


Untuk membangun BST, secara bertahap dilakukan operasi insert sesuai dengan kedatangan data dengan memperhatikan sifat khas BST yaitu : LC < Parent < RC.

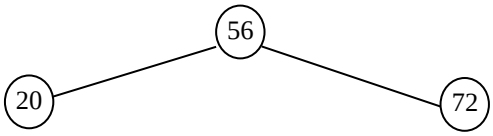
Langkah-langkah pembangunan BST untuk data pada contoh diatas adalah sebagai berikut :

Insert (56) sebagai data pertama --› root

Insert (20) : 20 < 56 --› 20 menjadi LC dari 56

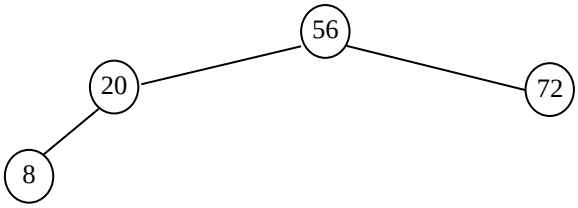


Insert (72) : 72 > 56 --› 72 menjadi RC dari 56



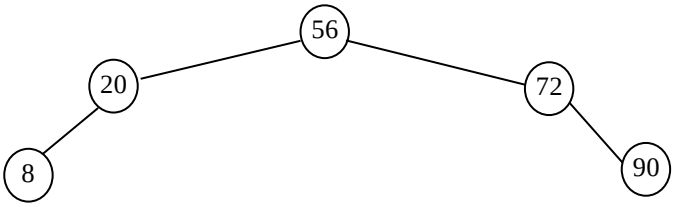
Insert (34) : 34 < 56 --› 34 menjadi LC dari 56

LC sudah berisi 20 : 34 > 20 --› 34 menjadi RC dari 20



Insert (90) : 90 > 56 --› 90 menjadi RC dari 56

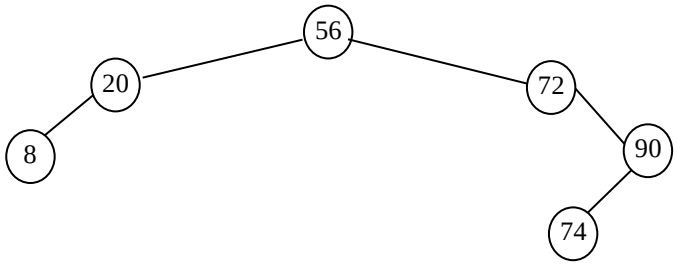
RC sudah berisi 72 : 90 > 72 --› 90 menjadi RC dari 72



Insert (74) : 74 > 56 --› 74 menjadi RC dari 56

RC sudah berisi 72 : 74 > 72 --› 74 menjadi RC dari 72

RC sudah berisi 90 : 74 < 90 --› 74 menjadi LC dari 90

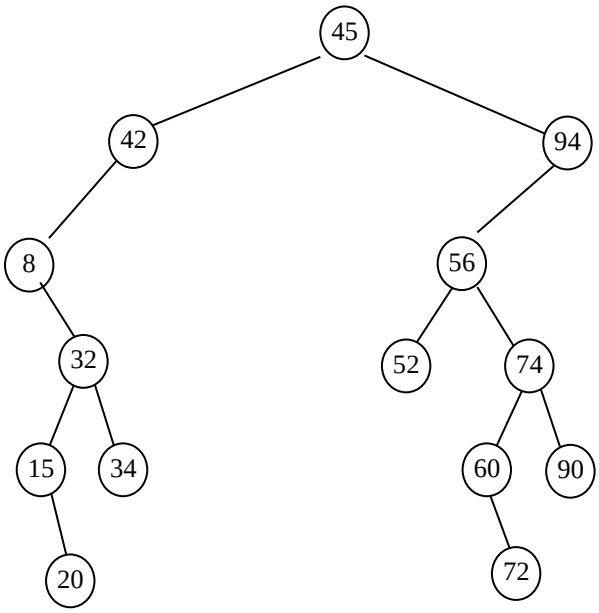


Demikian seterusnya secara berturut-turut akan diinsert 32, 45, 42, 94, 8, 52, 15, 60 sehingga dihasilkan BST.

Bagaimana jika urutan kedatangan data-data tersebut berbeda (datanya sama) ? misalnya sebagai berikut :

45 94 42 8 56 74 32 52 15 34 90 60 20 72

Akan dihasilkan BST sebagai berikut :



Dari contoh tersebut, diketahui bahwa BST yang dihasilkan juga dipengaruhi oleh urutan kedatangan data.

Deklarasi Array untuk implementasi BST

```
type

    Node   =  record
              Data    :  typedata;
              Kiri, Kanan :  integer;
          end;
var
    Pohon, : array [1.. n] of node
     Root : integer
```

Pencarian data dalam BST.

Operasi pencarian data pada BST mengikuti pola/aturan penempatan data pada tree tersebut.

Misalkan ingin dicari apakah pada BST terdapat data X?

Baca Node(root)

If X = Node(root) then data ditemukan

else If X < Node(root) then cari ke Sub BST Kiri

else cari ke Sub BST Kanan

Untuk menghindari kerumitan algoritma, implementasi Pencarian/Searching pada BST menggunakan algoritma rekursif pada halaman berikut :

```
Procedure  Search (var  Scurrent : node ;   cari : typedata );
 begin
     if   cari  <  Scurrent.Data
       then
         if Scurrent.kiri <>  0
            then    Search ( Scurrent.kiri , cari )
            else    write  ('data tidak ditemukan')
         { end-kiri }
       else
         if  cari  > Scurren.Data
           then
              if  Scurrent.kanan <>  nil
                 then    Search ( Scurrent.kanan , cari )
                 else    write  ('data tidak ditemukan')
              { end-kanan }
           else
                write  ('Data ditemukan')
       {end}
   {end}
 end;
```

Insert

BST memiliki sifat khas yaitu : LC < Parent < RC. Sehingga dalam melakukan operasi Insert, Update, dan Delete harus diperhatikan agar hasil akhir operasi tetap merupakan sebuah BST.

Pada operasi Insert, maka mesti ditemukan terlebih dahulu posisi yang sesuai bagi data baru. Sedangkan operasi Update dan Delete, akan memepengaruhi struktur pohon. Karena dapat menyebabkan pohon tersebut bukan lagi BST, sehingga diperlukan perubahan struktur pohon.

Seperti sudah digambarkan pada langkah-langkah pembangunan BST, penyisipan sebuah node baru, didahului dengan operasi pencarian posisi yang sesuai. Dalam hal ini node baru tersebut akan menjadi daun/leaf.

```
Procedure  Insert (databaru : typedata );
 begin
    Btree = <<next index array kosong>>
    B Tree.data    :=  databaru;
    B Tree.kiri    :=  nil
    B Tree.kanan   :=  nil

    If  root  =  0
     then
          root :=  BTree
     else
      begin
           Current  := root;
           Cariposisi (Current, databaru)
           If  databaru  <=  current.data
              then
                   current.kiri  :=   BTree
             else
                  current.kanan  :=  BTree
       end;
 end;

Procedure  Cariposisi (var  Pcurrent : BTree ;   dtBaru : tipedata );
 begin
     if   dtBaru  <=  Pcurrent^.Data
      then
        if  Pcurrent^.kiri <>  nil
            then    Cariposisi (Pcurrent^.kiri , dtBaru )
       { end-kiri }
       else
         if   dtBaru  > Pcurrent^.Data
          then
               if  Pcurrent^.kanan <>  nil   then  Cariposisi (Pcurrent^.kanan , dtBaru )
      { end-kanan }
   {end}
end;
```