PENERAPAN ALGORITHMA POHON UNTUK OPERASI PENGOLAHAN DAN PENYIMPANAN DATA DALAM TEKNIK PEMROGRAMAN (KAJIAN ALGORITHMA POHON PADA TEKNIK PEMROGRAMAN)

Fitri Latifah

Program Studi Komputerisasai Akuntansi AMIK BSI Jakarta
Jl RS Fatmawati Pondok Labu Jakarta Selatan
fitri.flr@bsi.ac.id

Abstract—*The implementation of algorithms in a programming technique is very important, the algorithm is expressive of one of the teaching methods, algorithms can also be used to explain how a formal system of a number of axioms And rules. For some cases the algorithm should be carefully formulated to how each step should be systematically organized case by case this is because the algorithm is a set of steps - the appropriate steps, the sequence of computing. In the computer structure scientific terms of data is a way to prepare and organize the storage of data on the storage medium that can be used effectively. Tree (tree) is a connected graph that contains no circuits. Tree (tree) is a data structure that is not linear and represented with hierarchical relationships between elements.*
Keyword : *Algorithm, Tree data structrue.*

Intisari—Penerapan algorithma dalam teknik pemrograman sangat penting, algorithma merupakan metode ekspresif dari sebuah instruksi, algorithma juga dapat digunakan untuk menjelaskan bagaimana sebuah sistem formal berasal dari sejumlah aksioma dan aturan-aturan. Untuk beberapa kasus, algorithma harus diformulasikan secara teliti dengan cara setiap langkah harus disusun secara sistematis kasus perkasus hal ini dikarenakan algorithma merupakan kumpulan dari langkah – langkah yang tepat, terurut dari komputasi. Dalam ilmu komputer istilah stuktur data merupakan cara penyimpanan penyusunan dan pengaturan data di dalam media penyimpanan sehingga dapat digunakan secara efisien. Pohon (tree) adalah graph terhubung yang tidak mengandung sirkuit. Pohon (tree) merupakan stuktur data yang tidak linier yang digambarkan dengan hubungan yang bersifat hirarkis antar satu elemen.
Kata kunci, algorithma, stuktur data pohon

PENDAHULUAN

Teori pohon pertama kali diperkenalkan sejak tahun 1857, oleh matematikawan Arthur Cayley yang digunakan untuk menghitung jumlah senyawa kimia. Teori pohon adalah teori yang digunakan untuk menyelesaikan permasalahan dengan menggunakan analogi permasalahan ke dalam bentuk pohon yang kemudian dicaarikan solusi pemecahan permasalahannya , selain dari itu teori pohon juga digunakan dalam penerapan konsep graph, dimana pohon dapat didefinisikan sebagai graph tidak berarah terhubung dan tidak mengandung sirkuit. Penerapan struktur data merupakan hal yang sangat penting dalam proses pembuatan progam komputer untuk meningkatkan kinerja program,
Teori pohon merupakan teori yang digunakan dalam stuktur data untuk aplikasi – aplikasi penyimpanan data. Dalam kajian tulisan ini akan dijelaskan beberapa algoritham teori pohon yang digunakan dalam perancangan program komputer
Dalam kajian ini penulis akan membahas tentang bagimana algorithma pohon dapat digunakan dalam penyelesaian masalah dengan bahasa program
Teori pohon adalah teori yang sangat bermanfaat dalam struktur data karena aplikasi – aplikasi teori pohon dapat dijadikan sebagai struktur dalam penyimpanan data yang sangat baik dalam berbagai kasus tertentu.
Berdasarkan uraian diatas maka dapat diuraikan bagaimana struktur data pohon dapat digunakan untuk menyimpan dan mencari data dalam teknik pemrograman serta bagaimana cara struktur data pohon mengelola data dalam mendia penyimpanan, bagaimana struktur data pohon dapat dimanfaatkan dalam teknik pemrograman untuk menyimpan dan mencari data dengan cepat dan efisien dan mengurangi bug dalam program komputer

BAHAN DAN METODE

1. Pengertian Algorithma

Algorithma adalah deskripsi langkah – langkah penyelesaian masalah yang tersusun secara ogis atau ururtan logis pengambilan keputusan untuk pemecahan suatu masalah (Teddy Marcus; 2005). Dalam ilmu computer algorithma merupakan prosedur dan langkah

untuk melakukan perhitungan, algorithma juga digunakan untuk melakukan perhitungan, perosesan data dan penalaran secara otomatis. Algoritma juga merupakan metode ekspresif dari sebuah instruksi yang digunakan untuk menghitung sebuah fungsi yang di awali dengan sebuah kondisi awal atau input awal yang dapat berupa "kosong", dimana instruksi tersebut akan di jalankan sehingga akan menghasilkan hasil akhir berupa keluaran. Konsep dari algorithma juga dapat digunakan untuk mendifiniskan notasi desidabilitas dimana notasi tersebut merupakan pusat untuk menjelaskan bagaimana sebuah system formal berasal dari sejumlah aksioma dan aturan-aturan. Dalam konsep logika waktu dalam algorithma untuk menyelesaikan suatu permasalahan tidak dapat dihitung hal ini dikarenakan tidak berelasi dengan dimensi fisik dari konsep tersebut maka muncul istilah ketidakpastian yang mengkarakteristikan sebuah pekerjaan yang sedang dijalankan sehingga muncul ketidak tersediaan definisi yang konkret pada penggunaan secara abstrak dan istilah dari algorithma. Algirithma sangat diperlukan untuk computer dalam mengolah data, Banyak program computer memberikan instruksi untuk dilakukan computer dalam menjalankan proram tertentu. Sebuah algorithma dapat dianggap sebagai langkah yang dapat disimulasikan oleh sebuah system.

Untuk beberapa kasus, algorithma harus diformulasikan secara teliti dengan cara setiap langkah harus disusun secara sistematis kasus perkasus hal ini dikarenakan algorithma merupakan kumpulan dari langkah – langkah yang tepat, terurut dari komputasi. Algorithma digambarkan dengan banyak notasi seperti Bahasa alamiah, psedocode, diagram alur. Bahasa alamiah jarang digunakan untuk ekspresi algorithma yang kompleks, sedangkan untuk psedocode, diagram alur adalah cara untuk menggambarkan algorithma secara terstruktur. Kebanyakan algorithma digunakan untuk mengimplementasikan program computer, akan tetapi untuk saat ini algorithma juga dapat digunakan untuk mendeskripsikan jaringan syaraf dan sirkuit eketronik (wikypedia)

2. Definsi struktur data

Dalam ilmu komputer istilah stuktur data merupakan cara penyimpanan penyusunan dan pengaturan data di dalam media penyimpanan sehingga dapat digunakan secara efisien. Sedangkan dala istilah pemrograman struktur data berarti tata letak data yang berisi kolom – kolom.

Struktur data adalah model logika yang secara khusus mengorganisasi data (Zakaria & Prijono, 2006)struktur data juga dapat di pahami bagaimana data dapat disimpan. Dalam pemahaman struktur data dikenal dengan 1) struktur data statis yaitu struktur data yang tidak berubah ubah dan yang ke 2) struktur data dinamik yaitu struktur data yang selalu berubah ubah

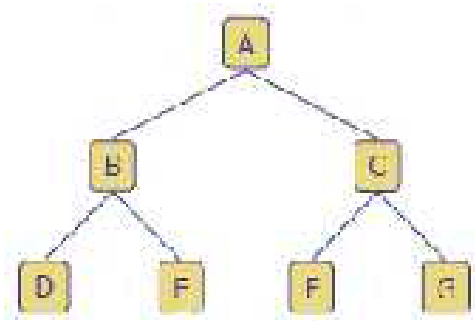
3. Pengertian stuktur data Pohon

Pohon (tree) adalah graph terhubung yang tidak mengandung sirkuit. Pohon (tree) merupakan stuktur data yang tidak linier yang digambarkan dengan hubungan yang bersifat hirarkis antar satu elemen (Teddy Marcus; 2005).

Jenis pohon dalam sturktur data sebagai berikut :

a. Full binary tree

Binary tree yang tiap nodenya memiliki satu root dan dua child dan harus memiliki panjang yang saman

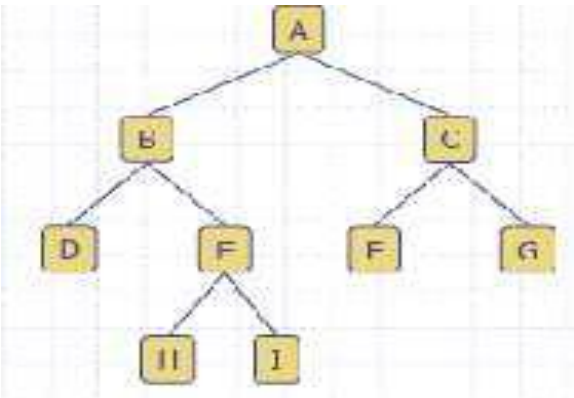


Sumber: (Setyaningsih, 2012)

Gambar 1. *Full Bianry Tree*

b. Complete binary tree

Pohon ini mirip dengan Full Bianry Tree, namun pada tiap sub tree boleh memiliki panjang path yang berbeda node kecuali leaf yang meiliki 0 atau 2 anak

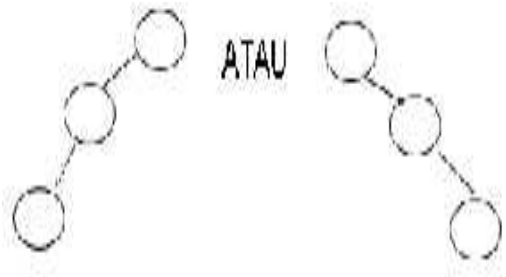


Sumber: (Setyaningsih, 2012)

Gambar 2 *Complete binary Tree*

c. Skewed binary tree

Pohon yang semua nodenya kecuali daun hanya memiliki satu anak (*child*)



Sumber: (Setyaningsih, 2012)
Gambar 3 Complete binary Tree

HASIL DAN PEMBAHASAN

Penerapan Algorithma pohon dalam pemrogaman

Struktur data pohon dapat digunakan dalam operasi dasar dalam teknik pemrograman seperti search, predecessor, succesor, minimum, maksimum, insert dan delete. Untuk penerapan struktur penyimpanan data dalam terori pohon ada 2 hal terbesar yang dapat di implementasikan yaitu penyimpanan secara static dan dinamik

1. Penyimpanan Data Statik

Penyimpanan statik merupakan penyimpanan data dimana memori yang digunakan untuk penyimpanan sudah di persiapkan terlebih dahulu, penyimpanan data ini lebih menekankan kepada stuktur data dengan menggunakan table kontinyu. Penyimpanan data dengan cara statik tentunya memiliki kelebihan dan kekurangan. Kelebihan dari penyimpanan ini data dapat diakses secara langsung dengan cara mengetahui lebih dahulu indeks data yang disimpan sedangkan kekurangan dari teknik statik kita tidak mengetahui apakan memori yang digunakan sudah penuh atau tidak dan batas maksimum indeks, serta jenis data yang di input tidak ada atau table dalam kondisi kosong status memori akan sama dengan keadaan table penuh sehingga penggunaan memori akan lebih boros.

Dengan kondisi seperti ini maka penggunaan penyimpanan data static tidak baik untuk kasus penyimpanan data yang tidak sama tipenya dan tidak diketahui berapa jumlah data yang akan disimpan sehingga akan menyebabkan sulitnya dalam mendeklarasikan table kontigu yang dapat menampung masukan data.

2. Penyimpanan Data Dinamik

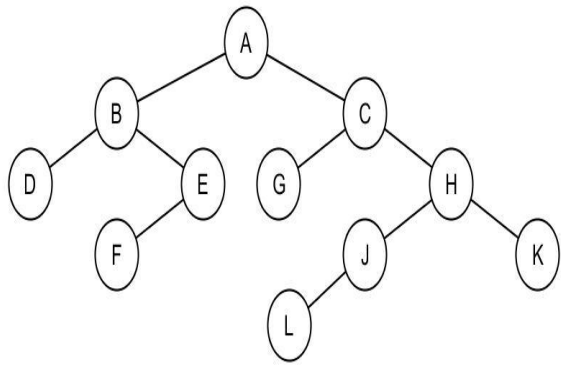
Penyimpanan data secara dinamik diimplementasikan dengan konsep linked list, dimana list itu mempunyai penujuk elemen pertama yang dijadikan sebagai acuan sebuah list yang kosong atau isi, namun implementasi list masih memiliki kekurangan untuk operasi, pencarian, penambahan, dan penghapusan data yang harus ditelusuri per-elemen mulai dari elemen pertama.

3. Stuktur data B-Tree

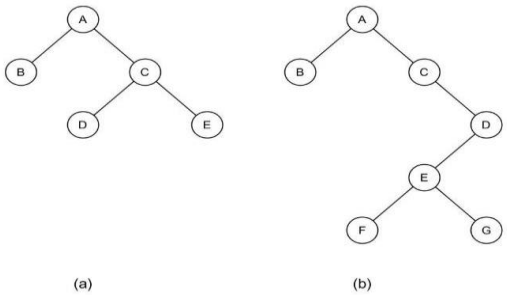
B-tree merupakan pohon cari keseimbangan yang dibuat untuk penyimpanan pada magnitic disk karena operasi yang sangat lambat dibandingkan dengan RAM.

B- tree mulai di perkenalkan pada tahun1960 tujuan dikembangkan metode ini adalah sebagai file system yang disebut dengan metode akses yang digunakan untuk beberapa mesin sperti *Sperry Univac Corporation*. Orang yang pertama kali menciptakan adalah *Rudolf Bayer dan Ed McCreight* tidak menjelaskan huruf B pada B-tree namun yang paling diyakini adalah B merupakan kependekan dari Balance yang artinya seimbang, karena semua simpul daun pada pohon berada pada tingkat atau level yang salam, namun ada sebagian yang mengartikan B merupakan Bayer nama depan dari penciptanya atau Boeing kerena merka bekerja untuk *Boeing Scientific Research Labs*. Balance tree merupakan pohon seimbang dimana tidak ada simpul daun yang lebih panjang dari yang lainnya. B-Tree merupakan sebuah m-ary balanced search tree yang digunakan untuk basis data, hal ini disebabkan strukturnya memungkinkan data yang disimpan dapat dengan mudah untuk disisipi, dihapus dan diambil dengan jaminan proses dengan waktu yang tidak bagus, dimana setiap simpulnya terdiri dari (m/2) samapai m buah simpul anak, dimana m>1 merupakan bilangan bulat positif dimana m merupakan orde. Akar pohon B tree paling sedikit memiliki 2 simpul anak. Balance tree juga dapat dianggap sebagai secarh tree dimana setiap subpohon dari sebuah simpul mempunyai kunci lebih kecil dari subpohon kanan. Sebuah B-tree didesain untuk digunakan pada storage berbentuk disk dimana pembacaan data hanya dpat digunakan pada blok dengan ukuran yang tetap dan besar. B-tree merupakan himpunan simpul yang terdiri dari 2 subpohon yang memiliki derajat keluaranmaksimum = 2 dan bersifat rekursif. Dalam ilmu komputer B-Tree adalah stuktur data pohon yang membuat data diurutkan serta pencarian data dilakukan secara sekuential dan penambahan data serta penghapusan Sebuah B-tree memiliki jumlah minimun anak yang mungkin untuk setiap simpul mungkin yang disebut dengan minimazation factor dengan asumsi jika t adalah faktor minimum dan setiap simpul harus memiliki paling sedikit t-1 kunci, dalam kondisi tertentu simpul diakar

diperbolehkan untuk memiliki lebih kecil dari t-1 kunci.



Sumber : (Suwanty & Pribadi, 2016)
Gambar 4. Struktru pohon binner



Sumber : (Suwanty & Pribadi, 2016)
Gambar 5 truktur pohon binner

4. Proses (operasi) Pada B-Tree

Proses atau operasi pada pohon biner adalah satu rangkaian proses atau fungsi – fungsi yang dibagi menjadi beberapa fungsi sebagai berikut :

a. Inisialisasi
b. Pembuatan simpul
c. Pembuatan simpul akar

a. Proses Inisialisasi

Proses ini merupakan pemberian nilai awal pada suatu variable atau kondisi yang dapat digunakan sebagai suatu ciri dalam satu kondisi. Untuk Instruksi dasar pada tahapan inisialisasi adalah :
Root = NULL;
P = NULL;
dan dapat difungsikan sebagai berikut

```
void Inisialisasi ( )
{
  Root  = NULL;
  P = NULL;
}
```

Fungsi dari inisialisasi ini harus dilaksanakan sebelum operasi yang lain dijalankan, dimana pointer Root pada saat dideklarasikan sudah berisi akan tetapi nilainya masih belum diketahui, pointer Root berisi Null karena hal ini akan dijadikan tanda, berisi Null karena pohon belum ada dan sebaliknya berarti pohon sudah ada dengan ditandai simpul akar yang ditunjuk oleh pointer.

b. Pembuatan Simpul

Proses ini merupakan pembuatan simpul awal di dalam sebuah tree, untuk pembuatan simpul dapat difungsikan sebagai berikut

```
void BuatSimpul (character X0
  { P = (Simpul*) malloc (sizeof (Simpul))
      If (P 1= NULL)
        { P-> INFO = X;
          P-> Leaf  = NULL;
          P-> Right = NULL:
        }
      else
        { printf ("Pembuatan Simpul Gagal");
        }
  }
```

c. Pembuatan Simpul Akar

Sebelum pohon dibuat untuk pertama kali, perlu untuk diperiksa apakan pohon memang belum ada, ciri suatu pohon belum ada adalah root=NULL jika pohon sudah ada maka tidak perlu untuk membuat simpul akar baru kemudian Root akan menujuk pohon baru, jika pohon sudah ada maka akan tercetak pesan pohon sudah ada. Fungsi untuk membuat simpul awal menjadi sebuahh root adalah sebagai berikut

```
void BuatSimpulAkar ( )
{ if (Root  == NULL)
    { if (P 1 = NULL)
        { Root = P ;
            Root -> Leaf = NULL;
            Root -> Right = NULL;
        }
        else
           printf (" \n Simpul Belum Dibuat");
        }
        else
            printf ("Pohon sudah Ada ");
```

5. Implemetasi Pada B-Tree

Implementasi teori pohon dalam tektik pemrograman merupakan penggunaan struktur data terbaik yang ada untuk kasus-kasus tertentu. Implementasi teori pohon yang dapat diterapkan dalam teknik pemrograman adalah :

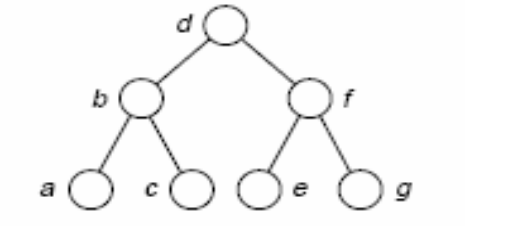
a. Binary Search tree
b. AVL Tree
c. Splay tree

a. Binary Search Tree

1) Model Opeasi Binary Search Tree

Binary Search tree dapat didefinisikan sebagai sebuah pohon yang memiliki properti sebagai berikut:

a) Pada semua elemen pada subpohon kiri memiliki nilai yang lebih kecil atau sama dengan nilai akar
b) Pada semua elemen pada subpohon kanan memiliki nilai yang lebih besar dari nilai akar
c) Pada subpohon kiri dan kanan adalah Binary Search Tree



Gambar 6.. Binary Search Tree
Sumber : Ridhwan akbar

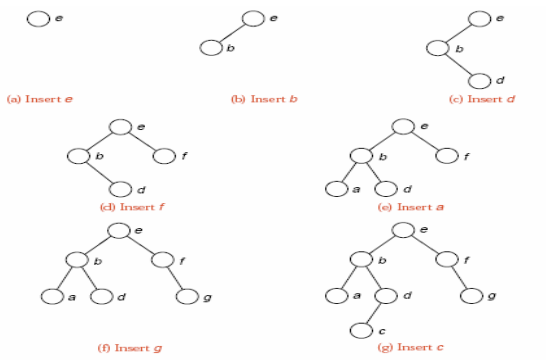
Pada Binary Seach tree terdapat 3 operasi dasar dan sangat penting yaitu : (1) operasi pencarian, (2) operasi penambahan (insert) dan (3) operasi penghapusan (delete)

Proses Pencarian pada binary search tree adalah pertema bandingkan terlebih dahulu kunci data yang ingin dicari dengan kunci di akar jika tidak cocok maka carilah ke subpohon sebelah kiri atau kanan sampai kunci data yang ingin dicari cocok, berikut gambaran dari algorithma operasi pencarian sebagai berikut :

Algorithma Cari Simpul (kunci K, pohon P)

a) Jika pohon P kosong kembali nilai NULL
b) Jika kunci K cocok dengan kunci akar t maka kembalilan simpul P
c) Jika kunci K > kunci akar simpul P maka kemblikan nilai dari Cari Simpul

Proses penambahan (insert) simpul ke dalam pohon adalah proses penempatan atau penempelan simpul baru menjadi subordinat sebuah simpul baik pada simpul kiri atau kanan. Dalam penambahan simpul pada pohon dapat dilakukan dengan dua cara yaitu penambahan nomor simpul atau diistilahkan dengan insert level perlevel dan penambahan nomor simpul tertentu. Pencarian lokasi untuk node yang akan ditambahkan dilakukan dimulai dari simpul akar , jika node yang akan ditambahkan lebih kecil dari akar maka akan dilakukan penambahan pada subpohon sebelah kiri sedangkan jika node yang akan ditambahkan lebih besar dari akan maka akan dilakukan penambahan pada subpohon sebelah kanan, untuk operasi penambahan node dapat di ilustrasikan sebagai beikut :



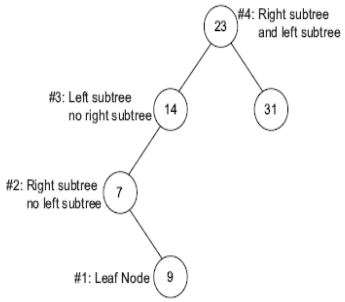
Gambar 7.. Operasi penambahan pada Binary Search tree
Sumber Khoirush Sholih

Secara umum algorithma dari proses penambahan (insert) dapat digambarkan sebagi berikut:

```
B-TREE-SPLIT-CHILD(x, i, y)
z <- ALLOCATE-NODE()
leaf[z] <- leaf[y]
n[z] <- t - 1
for j <- 1 to t - 1
do keyj[z]<- keyj+t[y]
if not leaf [y]
then for j<- 1 to t
do cj[z] <- cj+t[y]
n[y] <- t - 1
for j<- n[x] + 1 downto i + 1
do cj+1[x] <- cj [x]
ci+1[x] <- z
for j<- n[x] downto i
do keyj+1[x] <- keyj[x]
keyi[x] <- keyt[y]
n[x] <- n[x] + 1
DISK-WRITE(y)
DISK-WRITE(z)
DISK-WRITE(x)
```

Proses penghapusan. Penghapusan pada pohon cukup mudah dilakukan dengan empat cara yaitu :

a) Penghapusan elemen dilakukan disebelah kiri node subpohon
b) Penghapusan elemen dilakukan disebelah kanan node, akan tetapi bukan disebelah kiri subpohon
c) Pengahpusan elemen dilakukan disebelah kiri node, akan tetapi bukan disebelah kanan subpohon
d) Penghapusan di sebelah kiri dan kanan dari subpohon



Sumber: (Barnett & Del Tongo, 2008)
Gambar 8. Gambar Penghapusan Pohon

Algorithma penghapusan dapat difungsikan sebagai berikut :

1. Cari simpul yang memiliki kunci K
2. Jika simpul dengan kunci K ditemukan maka
   a. Jika subpohon kiri kosong makan tukar simpul P dengan subpohon kanan P lalu hapus simpul yang sudah ditukar.
   b. Jika subpohon kanan kosong maka tukar simpul P dengan subpohon kiri P lalu hapus simpul yang sudah ditukar.

```
1) algorithm Remove(value)
2)    Pre:  value is the value of the node to remove, root is the root node of the BST
3)          Count is the number of items in the BST
3)    Post: node with value is removed if found in which case yields true, otherwise false
4)    nodeToRemove ← FindNode(value)
5)    if nodeToRemove = ∅
6)       return false // value not in BST
7)    end if
8)    parent ← FindParent(value)
9)    if Count = 1
10)      root ← ∅ // we are removing the only node in the BST
11)   else if nodeToRemove.Left = ∅ and nodeToRemove.Right = null
12)      // case #1
13)      if nodeToRemove.Value < parent.Value
14)         parent.Left ← ∅
15)      else
16)         parent.Right ← ∅
17)      end if
18)   else if nodeToRemove.Left = ∅ and nodeToRemove.Right ≠ ∅
19)      // case # 2
20)      if nodeToRemove.Value < parent.Value
21)         parent.Left ← nodeToRemove.Right
22)      else
23)         parent.Right ← nodeToRemove.Right
24)      end if
25)   else if nodeToRemove.Left ≠ ∅ and nodeToRemove.Right = ∅
26)      // case #3
27)      if nodeToRemove.Value < parent.Value
28)         parent.Left ← nodeToRemove.Left
29)      else
30)         parent.Right ← nodeToRemove.Left
31)      end if
32)   else
33)      // case #4
34)      largestValue ← nodeToRemove.Left
35)      while largestValue.Right ≠ ∅
36)         // find the largest value in the left subtree of nodeToRemove
37)         largestValue ← largestValue.Right
38)      end while
39)      // set the parents' Right pointer of largestValue to ∅
40)      FindParent(largestValue.Value).Right ← ∅
41)      nodeToRemove.Value ← largestValue.Value
42)   end if
43)   Count ← Count −1
44)   return true
45) end Remove
```

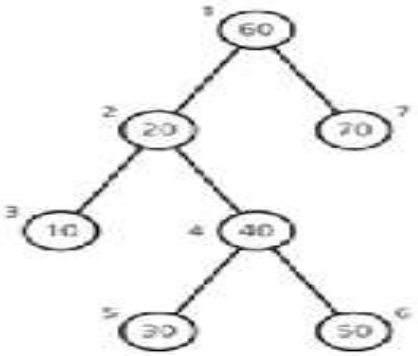
b. Model Kunjungan pada Binary Search Tree

Pada kunjungan binary search tree dikenal dengan istilah LRO (*leaf to Right Oriented*) yang maksudnya adalah kunjungan akan dilakukan pada *leaft child* terlebih dahulu kemudi ke *right child*

a. PreOrder (Depth First Order)

Model operasi pada PreOrder adalah akar-kiri- kanan dan dapat di fungsikan sebagai beikut

```
Prosedur PreOrdeer (P;B-Tree)
If P≠Nil then
   Output (info (P))
    PreOrder (kiri (P))
    PreOrder (kanan(P))
endif
```

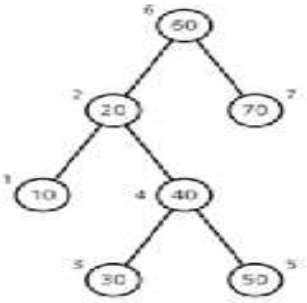


Gambar 9. PreOrder
Sumber : Emy Setyaningsih

b. InOrder (Symantic Order)

Model operasi pada PreOrder adalah kiri-akar-kanan dan dapat di fungsikan sebagai beikut:

```
Prosedur InOrder ( P: BTree)
   If P≠ Nil then
         InOrder (Kiri (P))
         Output (info (P))
         InOrder (kanan(P))
   endif
```



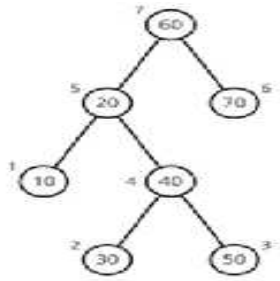
Sumber: (Setyaningsih, 2012)
Gambar 10. InOrder

c. PostOrder

Model operasi pada PreOrder adalah kanan–kiri – akar dan dapat di fungsikan sebagai beikut:

```
Procedur PostOrder (P : BTree)
   If P≠ Nil then
      PostOrder (kiri(P))
      PostOrder (kanan (P))
      Output (info (P)
```

Endif

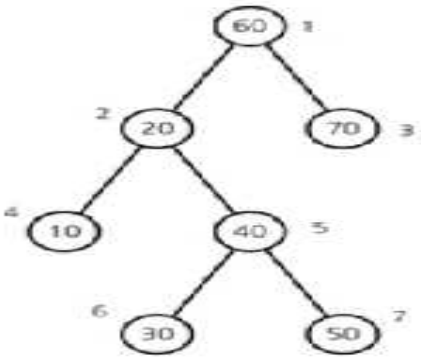


Sumber: (Setyaningsih, 2012)

Gambar 11. PostOrder

d. Level Order

Kunjungan pada node pohon ditingkat level yang sama yang dimulai dari akar sampai ke node yang merupakan daunnya

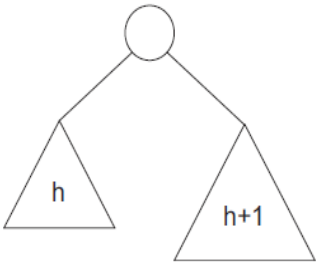


Sumber: (Setyaningsih, 2012)

Gambar 13. LevelOrder

a. AVL Tree

AVL Tree adalah Binary Search Tree yang memiliki keseimbangan yang tetap antara subpohon kiri dan kanan tidak lebih dari 1 untuk setiap simpulnya dan memiliki ketinggian yang sama (Barnett & Del Tongo, 2008)

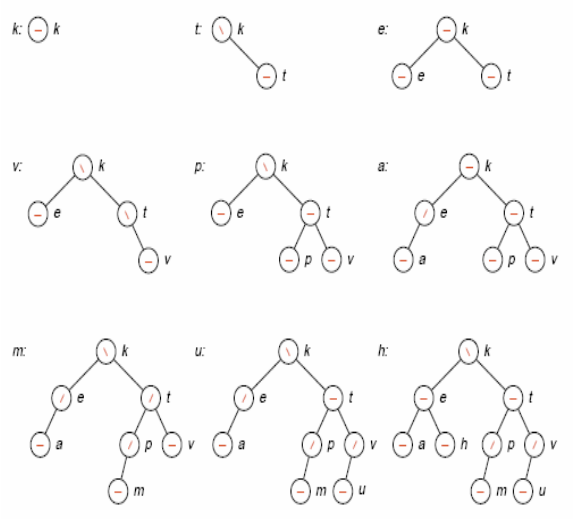


The left and right subtrees of an AVL tree differ in height

Gambar 14. AVL Tree

Sumber: (Barnett & Del Tongo, 2008)

Jadi AVL tree merupakan binary search tree yang subpohon kri dan kananya dari akar tidak berselisih lebh dari 1 serta setiap subpohon dari AVL Tree adalah AVL Tree yang tiap simpulnya memiliki faktor penyeimbang yang nilainya left higher (subpohon kiri > kanan), equal-heigght (subpohon kiri = kanan), right higher (subpohon kiri < kanan) (Khoirush Sholih, 2006). Sama dengan binary tree search tree pada AVL tree untuk operasi terdapat tiga cara yaitu : (1), Penambahan (2) Pencarian dan (3) penghapusan. Operasi Penambahan pada AVL Tree pada dasarnya sama dengan Binary Seacrh Tree. Pada kasus tertentu penambahan node sering menjadikan selisih tinggi subpohon kiri dan kanan lebih dari 1,



Sumber: (Akbar, 2006)

Gambar 15. Penambahan Elemen Pada AVL-Tree yang Sederhana

Saat akan melakukan penambahan node baru kedalam AVL Tree ada dua langkah utama yaitu :

1. Kita akan menelusuri struktur pohon untuk menemukan node yang benar untuk penambahan seperti pada Binary Seach Tree.
2. Kita akan menelusuri node yang akan ditambahkan dan mengecek bahwa keseimbangan node pada tree tetap, jika keseimbangan tidak ada maka akan diakukan penyeimbangan ulang

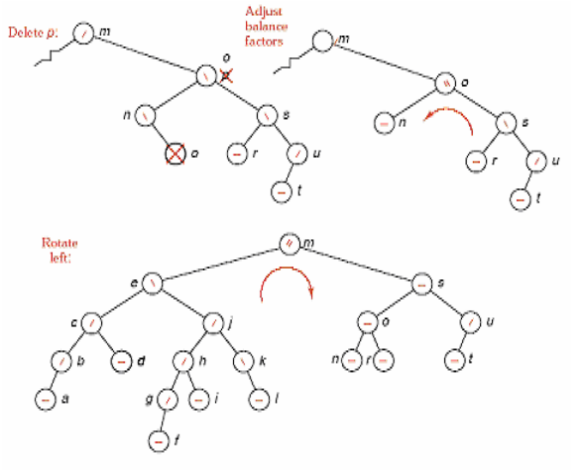
Kedua langkah diatas dapat digamabrkan dengan algorithma penambahan pada AVL Tree sebagai beikut

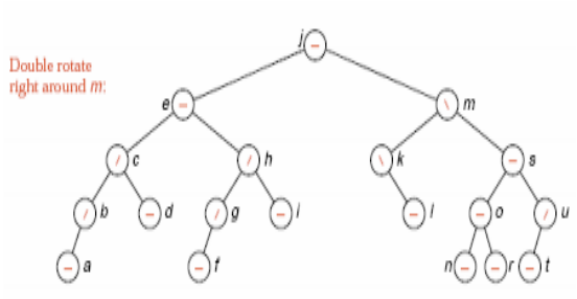
```
1) algorithm Insert(value)
2)    Pre:  value has passed custom type checks for type T
3)    Post: value has been placed in the correct location in the tree
4)    if root  = ∅
5)       root ← node(value)
6)    else
7)       InsertNode(root, value)
8)    end if
9) end Insert
```

```
1) algorithm InsertNode(current, value)
2)    Pre:  current is the node to start from
3)    Post: value has been placed in the correct location in the tree while
4)          preserving tree balance
5)    if value < current.Value
6)       if current.Left = ∅
7)          current.Left ← node(value)
8)       else
9)          InsertNode(current.Left, value)
10)      end if
11)   else
12)      if current.Right = ∅
13)         current.Right ← node(value)
14)      else
15)         InsertNode(current.Right, value)
16)      end if
17)   end if
18)   CheckBalance(current)
19) end InsertNode
```

Pada proses pencarian pada AVL pada dasarnya sama dengan operasi pencarian pada Binary Seacrh Tree, ini dikarenakan AVL Tree juga merupakanBinary Seach Tree dengan menambahkan beberapa properti yang disebut dengan keseibangan pohon

Pada proses penghapusan node AVL Tree pada prinsipnya memiliki kesamaan dengan proses penambahan node pada AVL Tree





Sumber: (Akbar, 2006)

Gambar 16. Contoh Penghapusan Elemen AVL-Tree

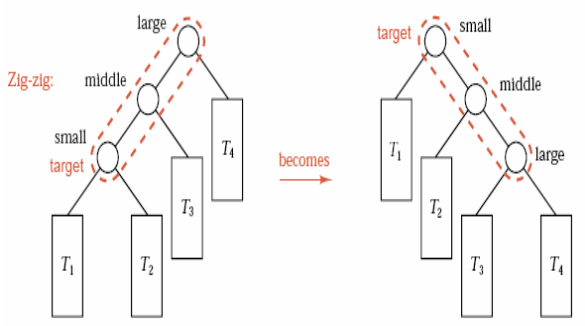
```
1) algorithm Remove(value)
2)    Pre:  value is the value of the node to remove, root is the root node
3)          of the Avl
4)    Post: node with value is removed and tree rebalanced if found in which
5)          case yields true, otherwise false
6)    nodeToRemove ← root
7)    parent ← ∅
8)    Stackpath ← root
9)    while nodeToRemove ≠ ∅ and nodeToRemove.Value ≠ Value
10)         parent = nodeToRemove
11)         if value < nodeToRemove.Value
12)            nodeToRemove ← nodeToRemove.Left
13)         else
14)         nodeToRemove ← nodeToRemove.Right
15)         end if
16)   path.Push(nodeToRemove)
17)   end while
18)   if nodeToRemove = ∅
19)      return false // value not in Avl
20)   end if
21)   parent ← FindParent(value)
22)   if count = 1 // count keeps track of the # of nodes in the Avl
23)      root ← ∅ // we are removing the only node in the Avl
24)   else if nodeToRemove.Left = ∅ and nodeToRemove.Right = null
25)      // case #1
26)      if nodeToRemove.Value < parent.Value
27)         parent.Left ← ∅
28)      else
29)         parent.Right ← ∅
30)      end if
31)   else if nodeToRemove.Left = ∅ and nodeToRemove.Right ≠ ∅
32)      // case # 2
33)      if nodeToRemove.Value < parent.Value
34)         parent.Left ← nodeToRemove.Right
35)      else
36)         parent.Right ← nodeToRemove.Right
37)      end if
38)   else if nodeToRemove.Left ≠ ∅ and nodeToRemove.Right = ∅
39)      // case #3
40)      if nodeToRemove.Value < parent.Value
41)         parent.Left ← nodeToRemove.Left
42)      else
43)         parent.Right ← nodeToRemove.Left
44)      end if
45)   else
46)      // case #4
47)      largestValue ← nodeToRemove.Left
48)      while largestValue.Right ≠ ∅
49)         // find the largest value in the left subtree of nodeToRemove
50)         largestValue ← largestValue.Right
51)      end while
52)      // set the parents' Right pointer of largestValue to ∅
53)      FindParent(largestValue.Value).Right ← ∅
54)      nodeToRemove.Value ← largestValue.Value
55)   end if
56)   while path.Count > 0
57)      CheckBalance(path.Pop()) // we trackback to the root node check balance
58)   end while
59)   count ← count − 1
60)   return true
61) end Remove
```

e. Spay Tree

Splay Tree adalah modifikasi binary search tree dengan tujuan utama untuk memudahkan pencarian dan pengambilan data terutama data yang baru masuk dan yang paling aktif diakses atau dimodifikasi (Khoiru sholihin,2007). Perbedaan antara Splay tree dibandingkan dengan binary search tree ataupun AVL Tree terletak pada data baru atau data yang frekuensi aksesnya

tinggi dan berada dekat dengan akar sehingga untuk mengakses data tersebut tidak membutuhkan waktu yang lama jika dibandingkan dengan binary seach tree maupun AVL Tree. Untuk operasi Splay Tree kita hanya perlu melakukan proses splaying atau proses pelebaran pada binary search tree. Ide utama dari proses splaying adalah dengan cara memindahkan simpul tujuan ke 2 level atas dalam setiap langkahnya. Untuk mengakses data tidak diperlukan waktu lama pada splay tree pada setiap operasi penambahan ataupun pengambilan data maka pohonnya akan dirombak ulang dengan menaikkan data yang berada dibagian bawah pohon atau daun sesuai dengan tingkat frekuensi akses dan keterbaharuannya. Seperti yang telah dijelaskan diatas splay tree merupakan binary search tree yang dimodifikasi untuk membuat splay tree memerlukan splaying atau pelebaran pohon.

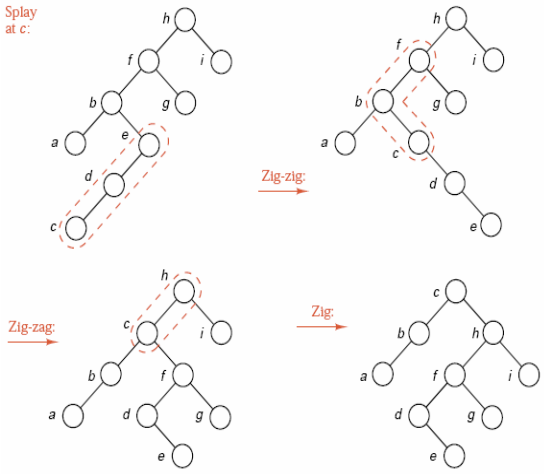
Untuk menjelaskan proses penambahan pada splay tree kita dapat mengilustraikan splay tree sebagai berikut: kita bayangkan lintasan sebuah pohon biner mulai dari akar sampai kesimpul yang akan diakses, untuk setiap kali pergerakan kekiri kita sebut dengan *zig* dan setiap perkegrakan kekanan kita sebut dengan *zag*, pergerakan 2 kali kekiri kita sebut dengan *zig-zig* dan perkerakan 2 kali kekanan kita sebut dengan *zag- zag*, jika pergerakan kekiri dan kekanan disebut dngan *zig – zag*, demikian seterusnya urutan langkah dalam penelusuran splay tree. Beriukutnya contoh proses *zig – zig, zag – zag, zig – zag* serta *zag - zig*



Sumber: (Akbar, 2006)

Gambar 17. Rotasi Pada Splay Tree

Dibwah ini adalah splaying pada binary seacrh tree, dimulai dari splaying pada simpul c, untuk mencapai simpul c dari akar akan menempuh lintasan h,f,b,e,d,c, oleh karena itu dari e ke d ke c akan dilakukan rotasi zig – zig berikutnya dari f ke b ke c dengan menggunakan rotasi zig – zag. Sub pohon d dan e tidak berubah bentuk tapi akan pidah tempat kemudian simpul c tinggal 1 tingkat ke akar dan subpohon f juga hanya akan pindahposisi



Sumber: (Akbar, 2006)

Gambar 18. Proses Splaying pada Binary Search Tre

KESIMPULAN

Berdasarkan uraian diatas dapat ditarik kesimpulan untuk penerapan teori pohon sangat berguna dalam pengolahan dan penempatan data dalam teknik pemrograman dimana dengan algorithma pohon kita mendapatkan struktur penyimpanan data yang relatif lebih baik dan efisien daripada stuktur penyimpanan lainnya.

Penggunaan representasi data dengan binary seacrh tree jauh lebih baik daripada penggunaan struktur tabel atau dengan link list. Penggunaan struktur data pohon sangat flesibel dan konsepnya dapat dikembangkan sesuai dengan permasalahan yang akan ditemui.

REFERENSI

Akbar, K. S. R. (2006). *Penerapan Teori Pohon Dalam Kajian Struktur Data*. Bandubg. Retrieved from http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2006-2007/Makalah/Makalah0607-24.pdf

Barnett, G., & Del Tongo, L. (2008). *Data Structures and Algorithms: Annotated Reference with Examples*. DotNetSlackers.

Setyaningsih, E. (2012). *Struktur Data*. Yogyakarta: AKPRIND Press.

Suwanty, S., & Pribadi, O. (2016). METODE AVL TREE UNTUK PENYEIMBANGAN TINGGI BINARY TREE. *Jurnal TIMES*, *4*(2), 61–65.

Zakaria, T. M., & Prijono, A. (2006). *Konsep dan Implementasi Struktur Data*. Bandung: Informatika.

BIO DATA PENULIS

Fitri Lataifah adalah dosen tetap pada AMIK BSI Jakarta di program studi Komputerisasi Akuntansi, dan juga sebagai pengajar di Prodi Teknik Informatika STMIK Nusamandiri dengan matakuliah stuktur data serta teknik pemrograman