TREE LAJUTAN

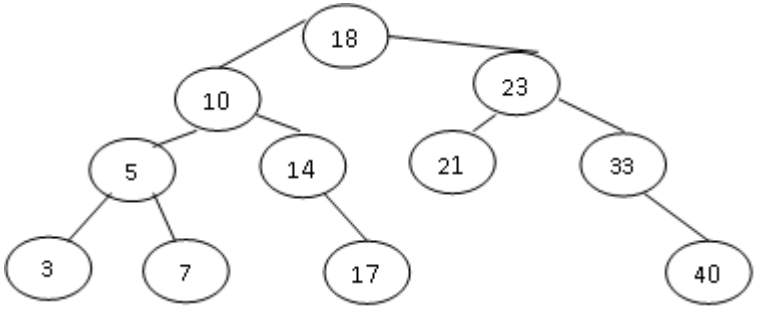
Struktur Data

JENIS TREE

- 2. Binary Search Tree

  Binary Search Tree Adalah Binary Tree dengan sifat bahwa semua left child harus lebih kecil daripada right child dan parentnya. Juga semua right child harus lebih besar dari left child serta parentnya. Binary seach tree dibuat untuk mengatasi kelemahan pada binary tree biasa, yaitu kesulitan dalam searching / pencarian node tertentu dalam binary tree.

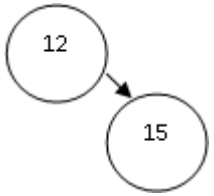
- Contoh binary search tree umum :



- Pada dasarnya operasi dalam binary search tree sama dengan Binary tree biasa, kecuali pada operasi insert, update, dan delete.
- 1. Insert : Pada Binary Search Tree, insert dilakukan setelah ditemukan lokasi yang tepat. (Lokasi tidak ditentukan oleh user sendiri).
- **Insert (12)**

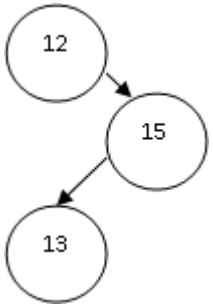
Memasukkan sebuah node yang berisi angka 12. Karena tree masih kosong, maka secara otomatis node tersebut menjadi root.

- **Insert (15)**



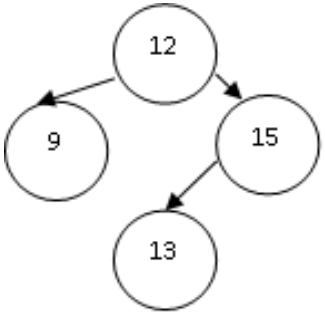
Karena 15 lebih besar dari 12, maka sesuai dengan peraturan harus berada disebelah kanan parent (right child)

- **Insert (13)**



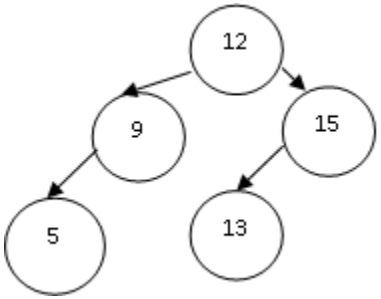
Karena 13 lebih besar dari 12, maka sesuai dengan peraturan harus berada di sebelah kanan parent (12), lalu bandingkan lagi dengan 15. karena lebih kecil maka 13 menempati left child dari 15.

- **Insert (9)**

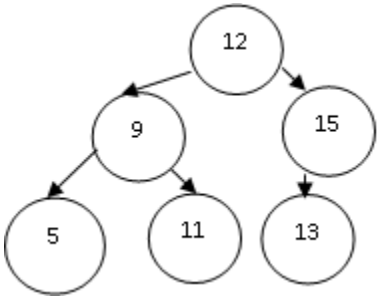


Proses yang sama seperti sebelumnya berlaku untuk selanjutnya

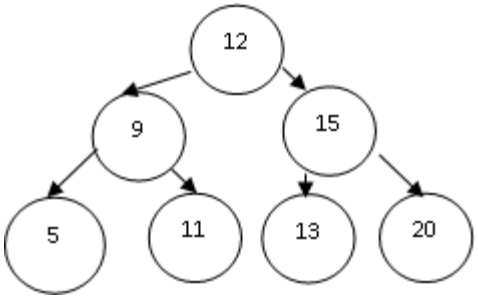
- **Insert (5)**



Insert (11)



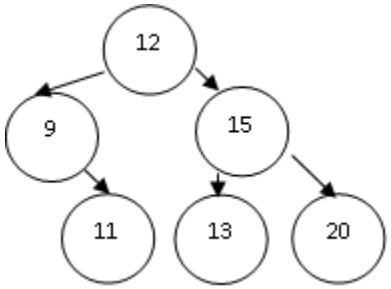
- **Insert (20)**



- 2. Update : Seperti pada Binary Tree biasa, namun disini uapte akan berpengaruh pada posisi node tersebut selanjutnya. Bila setelah diupdate mengakibatkan tree tersebut bukan Binary Search Tree lagi, maka harus dilakukan perubahan pada tree dengan melakukan perubahan pada tree dengan melakukan rotasi supaya tetap menjadi Binary Search Tree.

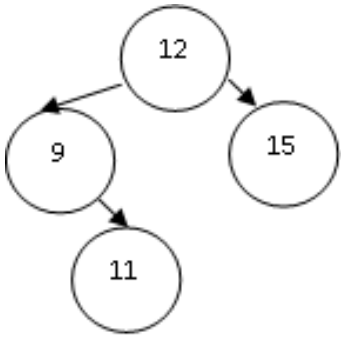
  Delete: Seperti halnya update, delete dalam Binary Search Tree juga turut mempengaruhi struktur dari tree tersebut

- **Delete (5)**



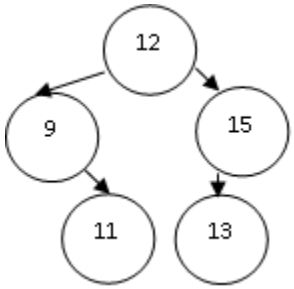
Karena node yang berisi 5 adalah leaf, maka ia dapat langsung dihapus.

- **Delete (20)**



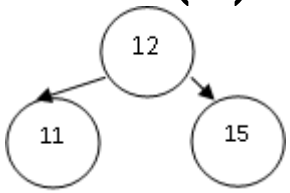
Karena node yang berisi 20 adalah leaf, maka ia dapat langsung dihapus.

- **Delete (13)**



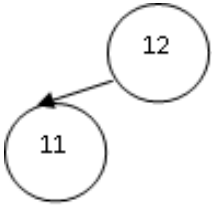
Pada operasi di samping, delete dilakukan terhadap Node dengan 2 child. Maka untuk menggantikannya, diambil node paling kiri dari Right SubTree yaitu 13

- **Delete (9)**



Pada operasi di samping, delete dilakukan terhadap node dengan 1 child. Maka child tersebut (11) akan menggantikan posisi dari node yang didelete (9).

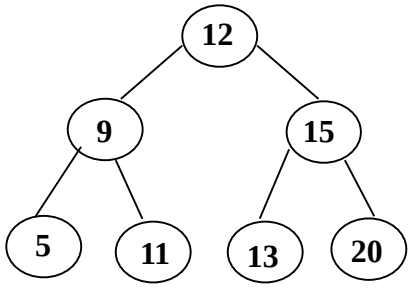
- **Delete (15)**



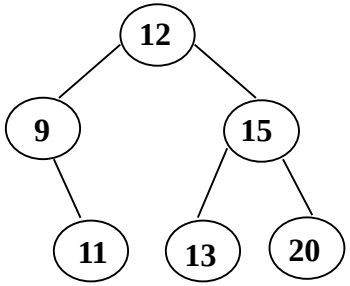
OPERASI DELETE BST

- Operasi delete dapat mengakibatkan BST tidak urut lagi, sehingga perlu dilakukan rotasi terhadap BST tersebut agar menjadi urut kecuali, beberapa posisi node pada saat dilakukan operasi delete adalah :
  1. Node yang dihapus adalah **LEAF**, sehingga penghapusan akan tetap membuat BST terurut. Bila yang terjadi adalah ini, maka operasi penghapusan dapat langsung dilakukan.
  2. Node yang dihapus adalah node yang memiliki node **1 CHILD**, sehingga child yang bersangkutan dapat langsung dipindahkan untuk menggantikan posisi node yang dihapus.
  3. Node yang akan dihapus memiliki **2 CHILDREN** (2 SUBTREE), maka node yang diambil untuk menggantikan posisi node yang dihapus adalah :
     a. Bisa berasal dari **LEFT SUBTREE**, dimana node yang diambil adalah node yang mempunyai nilai yang **PALING BESAR** (yang berada pada posisi paling kanan).
     b. Bisa berasal dari **RIHT SUBTREE**, dimana node yang diambil adalah node yang mempunyai nilai yang PALING KECIL (yang berada pada posisi paling kiri).

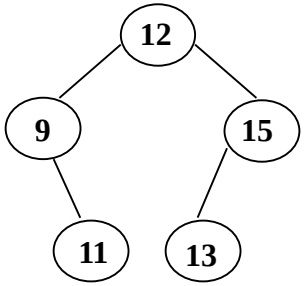
- Contoh :



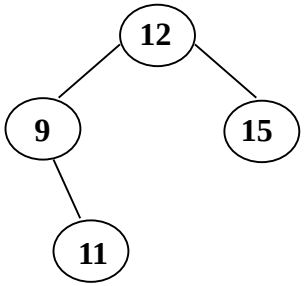
Delete (5)



Delete (20)

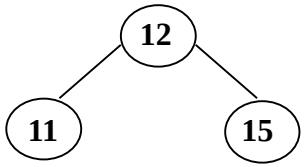


Delete (13)



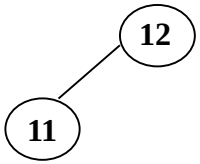
- Syarat 1

Delete (9)



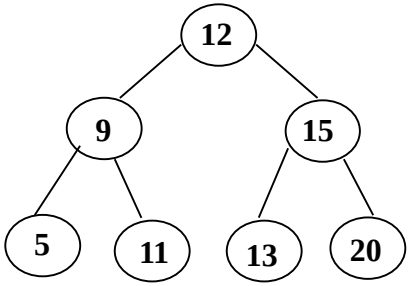
- Syarat 2

Delete (15)

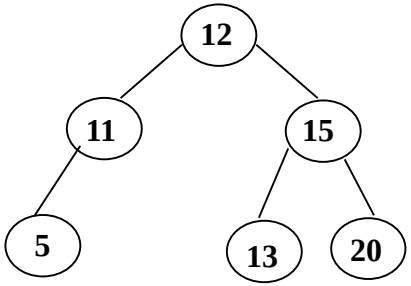


Syarat 1

- Contoh :

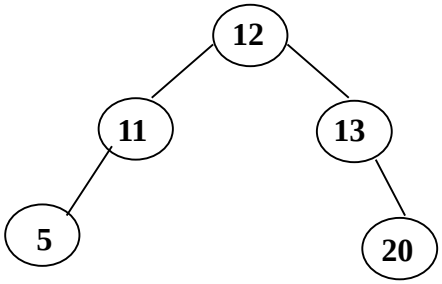


Delete (9)



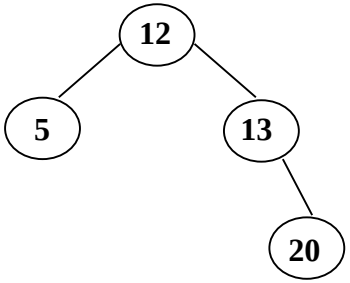
- Syarat 3A

Delete (15)



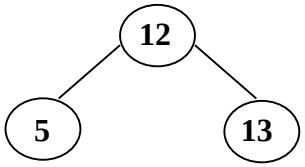
- Syarat 3B

Delete (11)



Syarat 2

Delete (20)



- Syarat 1

- Latihan

  Diketahui data : 20, 10, 28, 6, 15, 35, 8, 2, 7, 40, 12, 9, 25

  Dari data tersebut :

- 1. Susunlah menjadi BINARY SEARCH TREE, kemudian

  2. Lakukan Traversal Preorder, Inorder, dan Postorder

  3. Lakukan penghapusan terpisah untuk node-node 8, 6, 15, 10, 28