

## Kylee Valencia

1. Linear data structure  
↳ Tipe data structure yang memiliki pointer tanpa node bercabang dan dapat ditraverse menggunakan loop

Non Linear data structure  
↳ Tipe data structure yang memiliki <sup>pointer</sup> node & node bercabang seperti pohon yang harus dapat ditraverse di akses agar bisa balik ke parent parents dengan rekursi

2. Base root  
↳

Leaf node yang tidak memiliki anak

Edge ↳ Pointer dari node, <sup>garis</sup> yang menghubungkan antar sibling

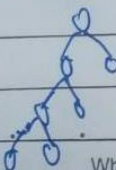
Key ↳ ~~value utama~~ (value yang menjadi faktor penentu posisi ~~struktur~~ node dalam suatu tree)

Siblings ↳ Node Node yang saling berhubungan langsung yang dihubungkan melalui 1 node Parent

Parent ↳ Node yang memiliki memiliki cabang

3. Full Binary Tree  
↳ Tree yang setiap ~~node~~ node harus memiliki 2 anak atau tanpa anak sama sekali

Full B Tree



Where there is a will, there is a way



11

22

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	-----



11



5. Properties in Binary Tree (Perfect B Tree Condition)

- Jumlah Maksimal Node di setoran level  $2^n$
- Jumlah Maksimal Node pada sebuah B tree  $2^{n+1} - 1$
- Tinggi Minimum dari Binary Tree dengan node sebanyak  $a$  adalah  $2 \log a$
- Tinggi Maximum dari B Tree dengan node sebanyak  $a$  adalah  $a - 1$

Tujuan dari ini adalah ~~mencari~~ ~~menjadi~~ B Tree menjadi

6. ~~tidak balance~~

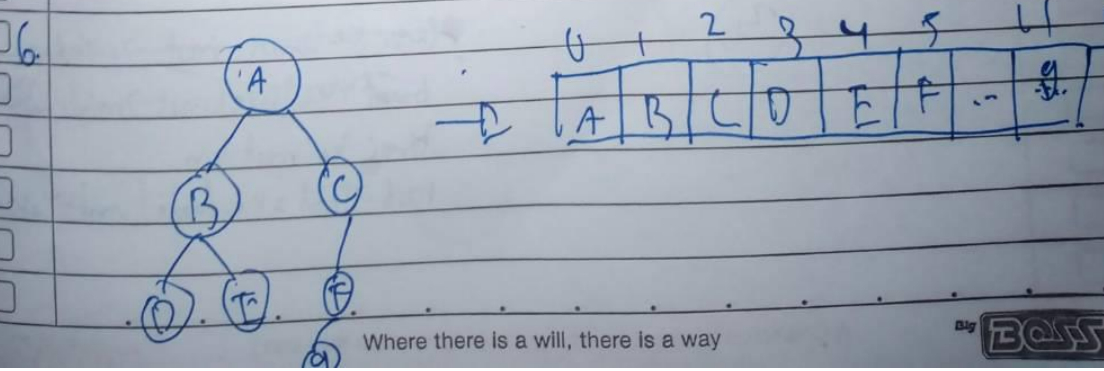


7. Inorder Predecessor

→ Nilai node yang digunakan sebagai pengganti node yang dihapus dimana nilai node pengganti tersebut berada pada subtree kiri dengan node terakhir paling kanan, dan secara nilai merupakan nilai terbesar yang paling dekat dengan node pengganti

Inorder Successor

→ Nilai node yang digunakan sebagai pengganti node yang dihapus dimana nilai node tersebut berada pada subtree kanan dengan node terakhir paling kiri, dan secara nilai merupakan nilai terkecil yang paling dekat dengan node pengganti



8 Insert 80

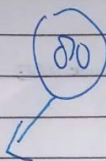
→

Null

(Kosong jadi insert 80)



Insert 30

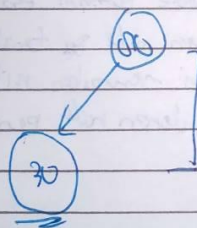


(Ada root dan nilai root lebih besar dari 30 jadi Arus ke root left dengan mengassign

root → left = ~~80~~

recNode()

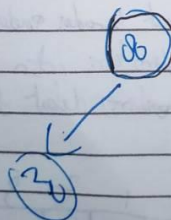
root → left, nilai



1 Balance

Sudah sampai ada left dan rootnya ke nilai kosong assign nilai 30 ke sin dan return ~~80~~ buat node

Insert 60



Ada root dan nilai 80 lebih ~~besar dari~~ 30 lebih dari 60

Pergi ke kanan root → right = ~~80~~  
buat recNode(root → right, ~~80~~)

Pergi ke root kiri

root → left = recNode(root → left, nilai)

Date: \_\_\_\_\_



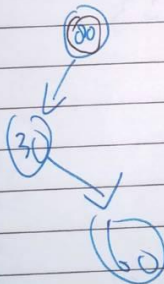
Ada root 30 dan  
60 lebih besar dari 30  
selanjutnya bagi ke kanan  
root  $\rightarrow$  right = ~~new Node~~ (root nilai, nilai)



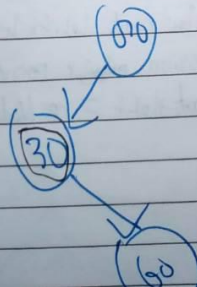
$2 - 0 = 2$   
Balance = 2

sekarang root kosong  
yang berarti assign nilai  
root  
kemudian buat Node (nilai)  
dan pengembalian stacknya

Insert 50

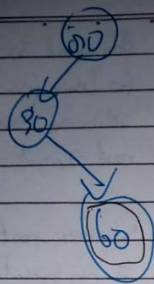


Posisi root sekarang memiliki  
nilai dan ~~root~~ root  $\rightarrow$  value  
yang berarti posisi ke kiri  
root  $\rightarrow$  left = ~~new Node~~ root  $\rightarrow$  left  
nilai di assign root  $\rightarrow$  ~~left~~  
adalah agar ketika rekursi  
balik dapat menaruh nilai  
sebelumnya



Posisi root sekarang ada nilai  
dan nilai root lebih  
kecil root (value)  
jadi assign root  $\rightarrow$  right  
= ~~new Node~~ root  $\rightarrow$  right = ~~new~~





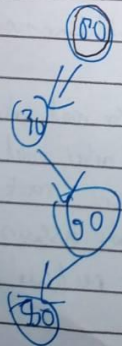
root lebih besar nilainya dari  
dengan nilai yang di insert, sehingga  
refer pointer ke kiri  
root->left = newNode(root->left,  
nilai)



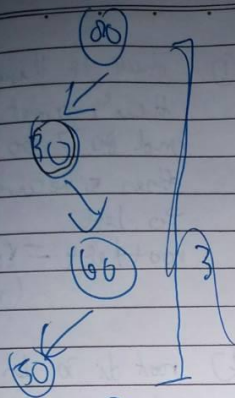
keadaan root sekarang forms  
nilai selanjutnya langsung  
menyisipkan nilai rootnya  
dan return balik rekursi

Balance = 3

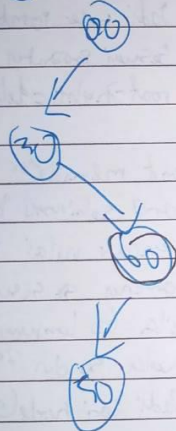
Insert 75



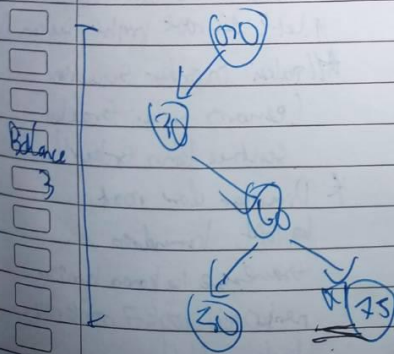
root sekarang memiliki nilai dan  
nilai root lebih besar dari nilai selanjutnya  
assign nilai pointer ke left  
root->left = newNode(root->left, nilai)



root sekarang memiliki nilai yang  
lebih kecil dari seluruh node  
pointer ke kanan  
root  $\rightarrow$  right = nextNode ke root  $\rightarrow$  right  
nilai

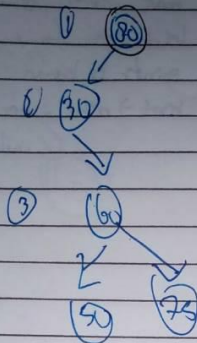


root sekarang memiliki nilai lebih  
kecil, sehingga node pointer ke kanan  
root  $\rightarrow$  right = nextNode root  $\rightarrow$  right, nilai



nilai root adalah kosong (bukan nol)  
selama bisa assign nilai  
insert dan return ke bawah  
sebelumnya

Delete: 60

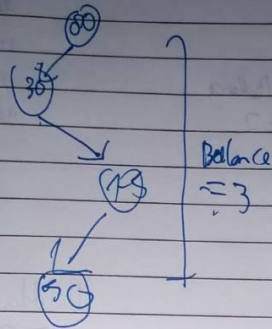


1) check if there's a root  
there's root 80  
and  $80 > 60$   
then a recursion points  
to left  
rootLeft = delete  
(rootLeft, 60)

2) root di 30 dan  $30 < 60$   
jadi kita pindah ke kanan  
sesuai Binary BST  
rootRight = delete(rootRight, 60)

3) root memiliki nilai 80  
sama sehingga bisa lanjut  
mencari nilai pengganti 60  
karena 60 sesuai garis  
jika 60 langsung hilang  
maka 50 dan 75 tidak bisa lanjut  
jadi cari node candidate  
pengganti 60 karena root  
left dan root right bukan Null  
// pakai inorder Successor  
(cari nilai terkecil  
subtree kanan 60)  
\* Dimulai dari root  
ke left kemudian  
traverse ke kanan hingga  
menemukan rootRight yang  
bukan Null

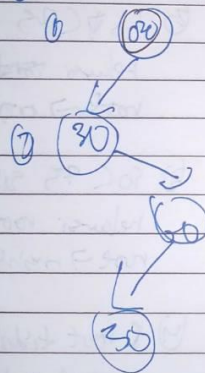




dimana dalam buku  
ini adalah 75  
dan nilai 60  
ditentukan 75  
sekarang node 75  
dikawat index 0  
1

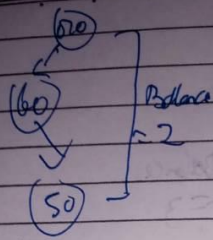
\* 75 di delete menurut  
kan nilai 60  
dikawat dapat mendu  
parent

Delete 30



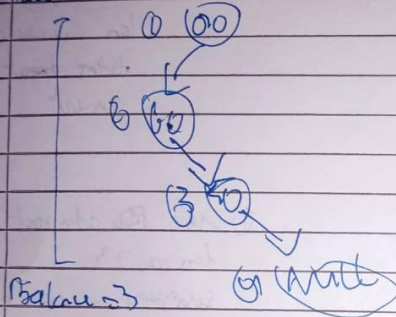
① Check jika ada root  
dan ada 30  
sekarang nilai 30  
letak ke kiri  
root → left = delete root → left  
(30)

② 30 rootnya adalah sama  
sekarang 30 dan 35  
30 masih ada root karena  
dan yang harus kita  
cari root pengganti dari  
dari root yang delete  
inorder successor  
Pengganti dari root jika  
langsung dilakukan jika  
root left atau right  
adalah Null



dan dalam kasus ini  
root left kosong,  
yang berarti yang ada  
adalah root right

Delete 75



1) Ada root dan root 100  
775 yang berarti  
recursi ke pointer left  
root 100

2) Go ke 75 yang berarti  
rekursi ke rekursi  
root → right dari 60

3) SOC 75 selesai  
rekursi root 50 yang berarti  
root → right

4) If root tidak bernilai  
yang berarti ke kembali  
nilai root