Organisasi Berkas Sekuensial Berindeks

Tata Sumitra, M.Kom
www.tatasumitra.com

Definisi

- Organisasi Berkas ini mirip dengan Organisasi Berkas Sekuensial dimana setiap rekaman disusun secara beruntun di dalam file, hanya saja ada tambahan indeks yang digunakan untuk mencatat posisi atau alamat dari suatu kunci rekaman di dalam file
- Indeks memiliki dua bagian, yaitu :
  - Kunci
  - Alamat
- Indeks digunakan untuk melakukan *lookup* dari kunci yang ada ke alamat penyimpanan rekaman
- Untuk alasan performa, indeks harus selalu terurut berdasarkan kunci

Single-Level Indexing

- Pengindeksan dengan hanya menggunakan sebuah indeks

indeks

ADE	102
BCA	105
CNN	101
NFL	104
SDN	103

berkas

	…
101	CNN
102	ADE
103	SDN
104	NFL
105	BCA
106	
	…

Multi-Level Indexing

- Pengindeksan dengan menggunakan beberapa indeks yang saling terhubung
- Indeks yang berlevel tinggi merupakan detail dari indeks yang berlevel lebih rendah

level 1

76100	0
76200	4
76300	6
…	…

level 2

0	76100	0
1	76110	5
2	76120	10
3	76130	16
4	76200	21
5	76210	23
6	76300	25
	…	…

level 3

0	76100	102
1	76101	105
2	76102	101
3	76103	104
4	76104	106
5	76110	103
6	76110	107
	…	…

berkas

	…
101	76102
102	76100
103	76110
104	76103
105	76101
	…

Implementasi Indeks

- Dengan Tabel
- Dengan AVL-Tree
- Dengan B-Tree

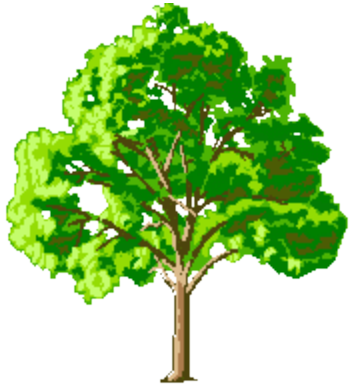
Indexing dengan Tabel

- Indeks dibentuk dari sebuah tabel dengan dua kolom, yaitu kolom kunci dan kolom alamat
- Merupakan implementasi indeks yang paling sederhana, namun dibutuhkan kerja ekstra untuk pemeliharaannya, yaitu menjaga agar tabel tetap dalam keadaan terurut

Indexing dengan AVL-Tree

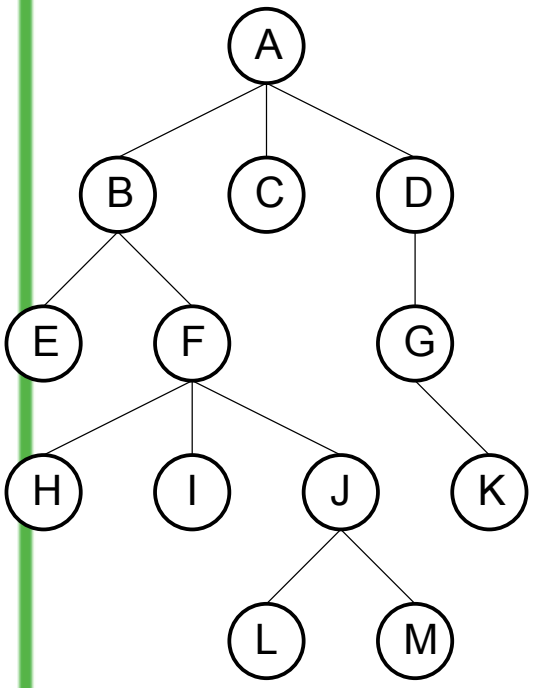
- Indeks dibentuk menggunakan struktur *Binary Search Tree* (BST)
- Dengan cara ini, *maintenance* indeks lebih mudah karena AVL-Tree selalu menjaga agar selalu dalam keadaan terurut

Definisi Tree



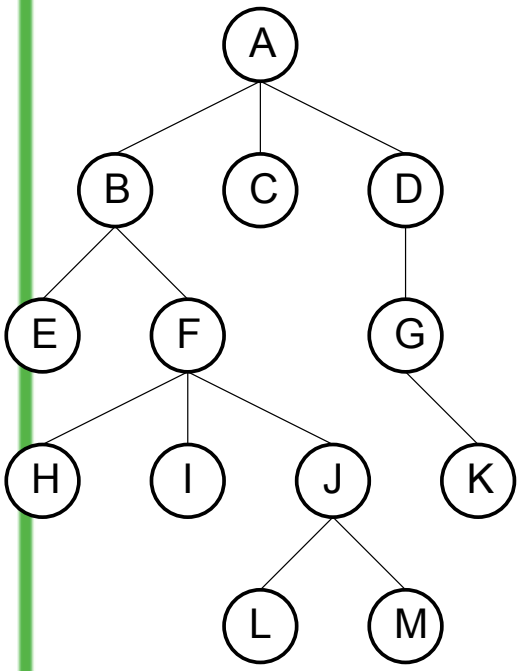
- Dalam ilmu komputer, *Tree* adalah suatu struktur data yang banyak dipakai untuk mengemulasikan suatu struktur pohon yang terdiri dari serangkaian simpul data yang saling terhubung
- *Tree* merupakan bentuk khusus dari *graph*, dimana dalam tree tidak ada hubungan antar simpul yang tertutup (sirkular)

Terminologi–Terminologi Tree



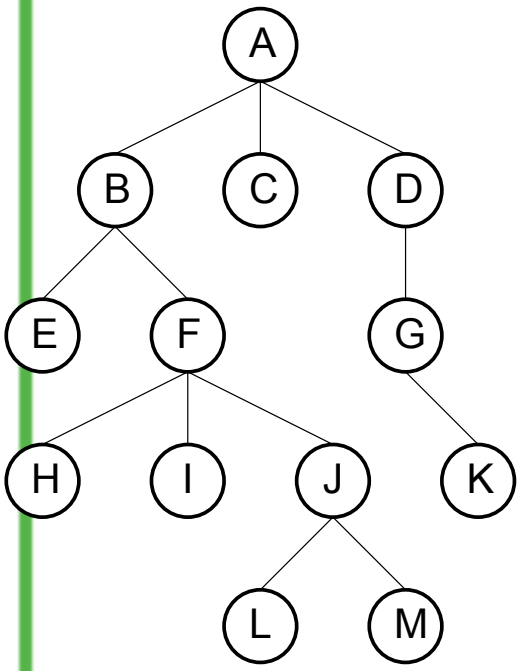
- Setiap data disimpan di dalam bagian yg dinamakan simpul (*node*)
- Dalam contoh di samping, node adalah setiap lingkaran yang berisi huruf
- Setiap node dihubungkan dengan suatu sisi (*edge*) atau cabang
- Pada contoh di samping, sisi atau cabang adalah setiap garis yang menghubungkan setiap lingkaran
- *Level* dari suatu node dalam tree didefinisikan sebagai jarak suatu node dari node teratas dalam tree tersebut
- Dalam contoh, node A memiliki level 0, sementara pada level 2 terdapat node-node E, F dan G

Hubungan Antar Node



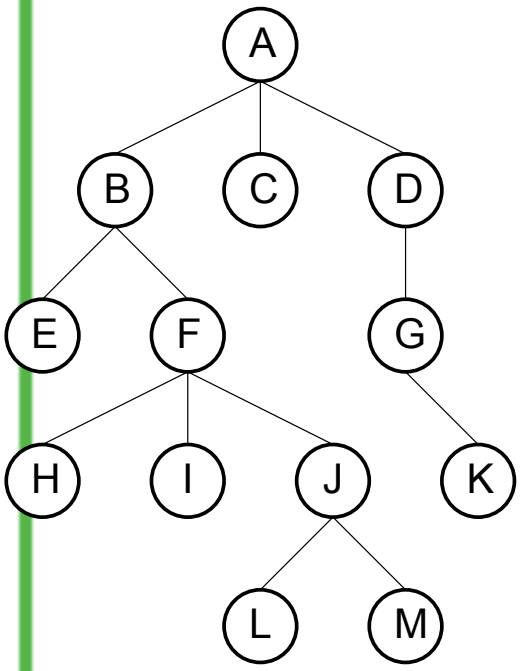
- Node yang berada di atas node lain dan terhubung oleh sebuah cabang dinamakan orang tua (*parent*) dari node di bawahnya
- Dalam contoh, node E dan node F memiliki *parent* node B, sementara node D adalah *parent* dari node G
- Node yang berada di bawah merupakan anak (*child*) dari node di atasnya yang terhubung cabang
- Dalam contoh, node K adalah *child* dari node G, sementara node J memiliki dua buah *child*, yaitu node L dan M

Hubungan Antar Node



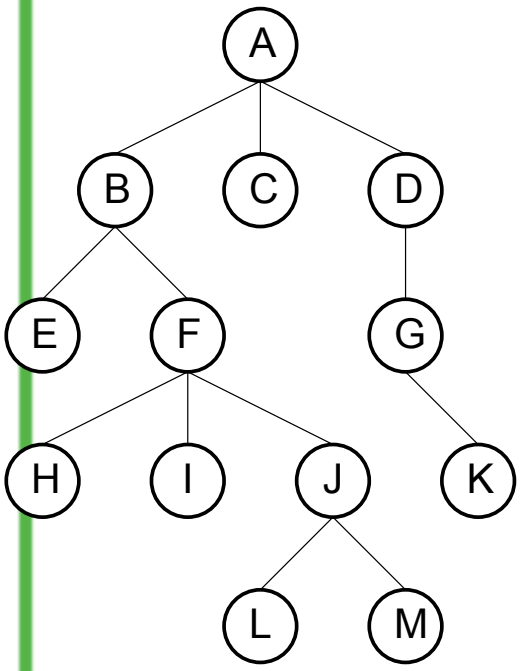
- Seluruh node yang berada di bawah hirarki suatu node X dinamakan turunan (*descendant*) dari node X
- Dalam contoh, semua node dari B hingga M merupakan *descendant* dari node A, sementara node D memiliki *descendant* node G dan K
- Node-node yang memiliki *descendant* node X dinamakan *ancestor* dari node X
- Dalam contoh, node A merupakan *ancestor* dari semua node lainnya, sementara node H memiliki *ancestor* F, B dan A

Hubungan Antar Node



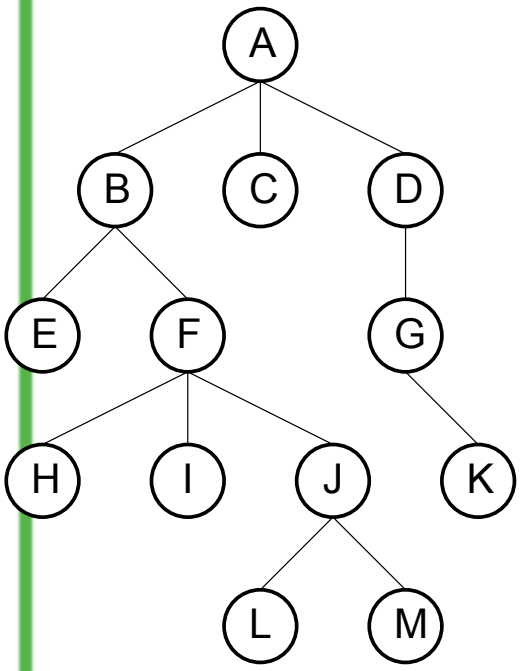
- Node-node yang memiliki *parent* yang sama dinamakan saudara (*siblings*)
- Dalam contoh, node L dan M merupakan *siblings*, karena sama-sama memiliki *parent* yang sama yaitu node J

Node-Node Khusus



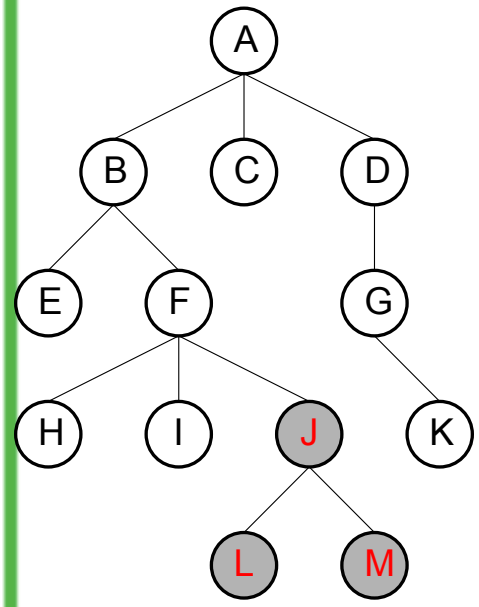
- Node yang paling atas dinamakan akar (*root*), yaitu satu-satunya node dalam *tree* yang tidak memiliki *parent*
- Pada contoh, node A adalah *root*
- Node yang tidak memiliki anak dinamakan daun (*leaf*)
- Pada contoh, node yang merupakan daun adalah node-node C, E, H, I, K, L dan M
- Node selain daun dinamakan *Internal Nodes*

Tinggi Suatu Tree



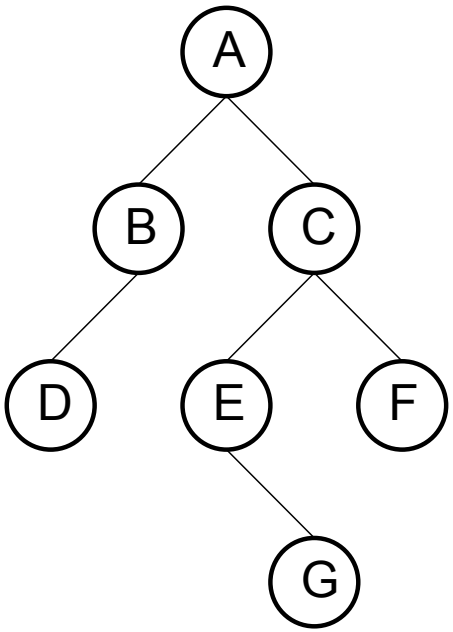
- Tinggi atau *height* atau *order* dari suatu *tree* adalah jarak dari *root* ke daun yang paling jauh
- Tree yang hanya memiliki satu node saja (*root* sekaligus daun) dikatakan memiliki tinggi 0
- Tree pada contoh di samping memiliki tinggi 4, yang merupakan jarak (banyaknya sisi) dari *root* (node A) ke daun terjauh (node L atau M)

Sub-Tree



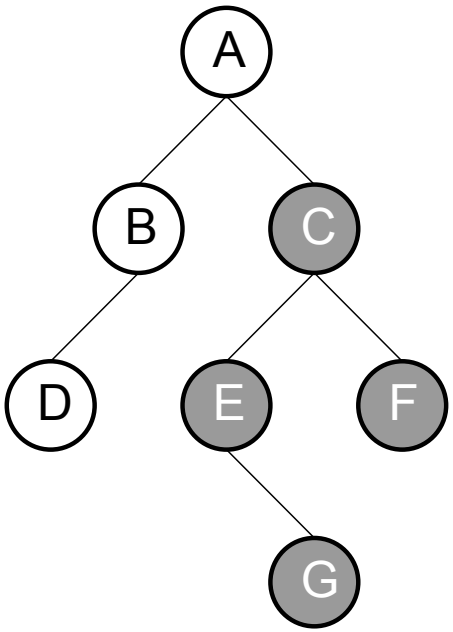
- Sub-tree merupakan suatu struktur tree yang berada di dalam struktur tree lainnya
- Sub-tree dari suatu node adalah sub-tree yang *root*-nya adalah anak node tersebut
- Pada contoh, node A memiliki 3 sub-tree, yaitu sub-tree yang memiliki *root* node B, sub-tree yang hanya terdiri dari satu node C dan sub-tree yang memiliki *root* node D
- Sub-tree yang dibentuk dari node-node berwarna abu-abu pada contoh merupakan sub-tree dari node F
- Daun pasti tidak memiliki sub-tree

Binary Tree



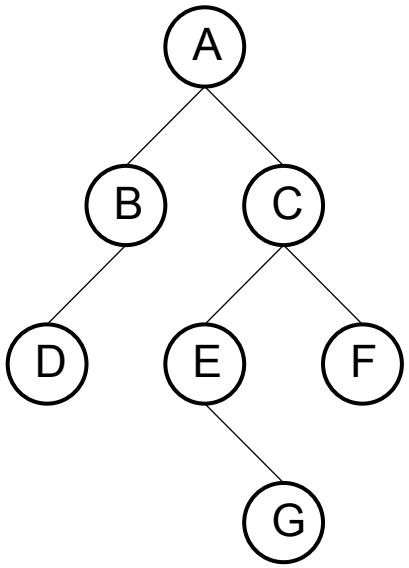
- *Binary Tree* atau Pohon Biner adalah sebuah tree yang setiap nodenya **maksimal** hanya memiliki dua anak
- Pada contoh,
  - Node A memiliki 2 anak
  - Node B memiliki 1 anak
  - Node C memiliki 2 anak
  - Node D memiliki 0 anak
  - Node E memiliki 1 anak
  - Node F memiliki 0 anak
  - Node G memiliki 0 anak

Sub-Tree Kiri & Sub-Tree Kanan



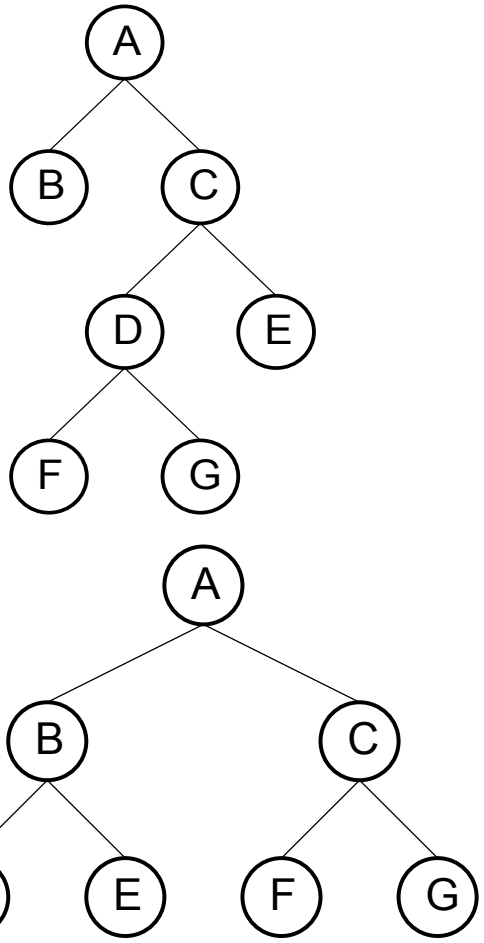
- Setiap anak pada suatu node di *binary tree* dinamakan sebagai anak kiri (*left child*) dan anak kanan (*right child*)
- Pada contoh,
  - **Node A memiliki anak kiri node B dan anak kanan node C**
  - **Node D merupakan anak kiri dari node B**
  - **Node E tidak memiliki anak kiri**
- Sub-tree yang dibentuk dari anak kiri suatu node dinamakan sub-tree kiri dan sub-tree kanan adalah sub-tree yang dibentuk dari anak kanan
- Dari contoh, sub-tree yang dibentuk dari node-node berwarna abu-abu merupakan sub-tree kanan dari node A

Balance Factor



- Faktor Keseimbangan (*Balance Factor* / BF) suatu node pada *binary tree* adalah tinggi sub-tree kanan dikurangi tinggi sub-tree kiri
- Biasanya BF ditulis menggunakan tanda (negatif atau positif) kecuali untuk nilai 0 (nol)
- Jika suatu node tidak memiliki sub-tree pada suatu sisinya maka dianggap tinggi sub-tree node pada sisi tersebut adalah -1
- Pada contoh,
  - **node A memiliki BF = +1 yang didapat dari 2 dikurangi 1**
  - **Node B memiliki BF = -1 yang didapat dari -1 dikurangi 0**
- Suatu daun pasti memiliki BF = 0

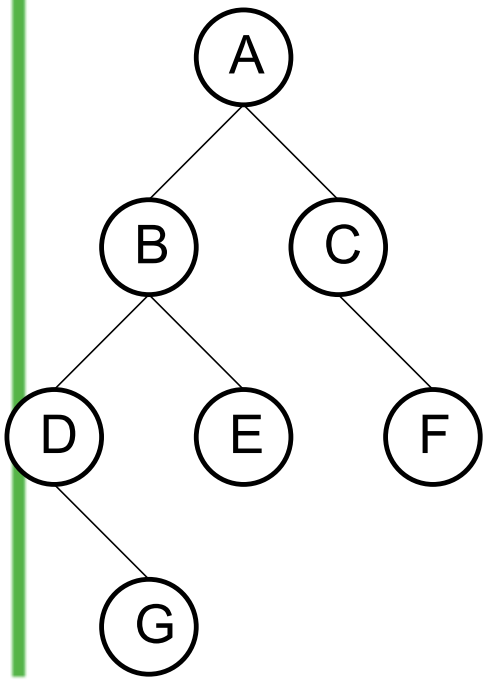
Pohon Penuh & Pohon Sempurna



- Pohon Penuh (*Full Binary Tree*) adalah sebuah *binary tree* yang setiap node internalnya pasti memiliki dua anak

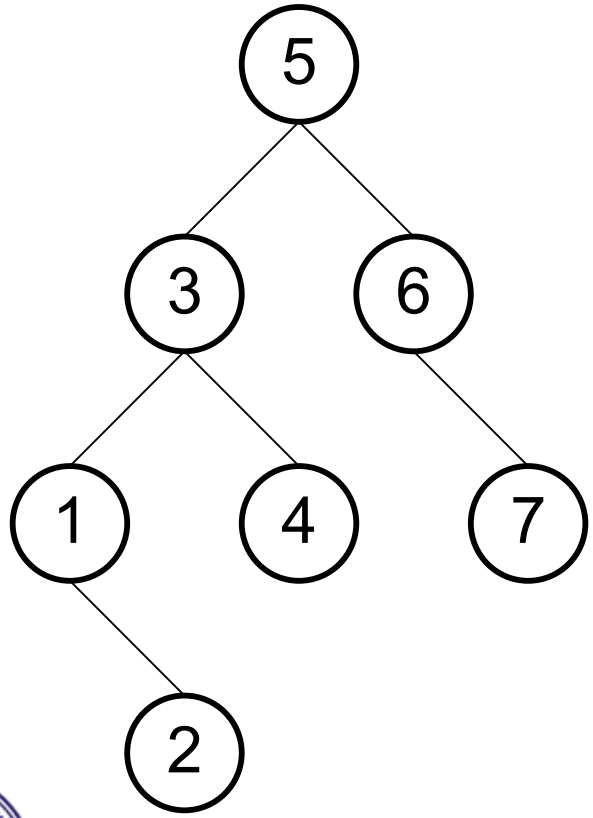
- Pohon Sempurna (*Perfect Binary Tree*) adalah pohon penuh yang semua daunnya berada pada level yang sama, sehingga semua node memiliki BF = 0

Pohon Seimbang



- Pohon Seimbang (*Balanced Tree*) adalah pohon biner yang semua nodenya hanya memiliki BF antara -1, 0, atau +1 saja
- Pada contoh,
  - Node A dan B memiliki BF = -1
  - Node C dan D memiliki BF = +1
  - Node E, F dan G memiliki BF = 0

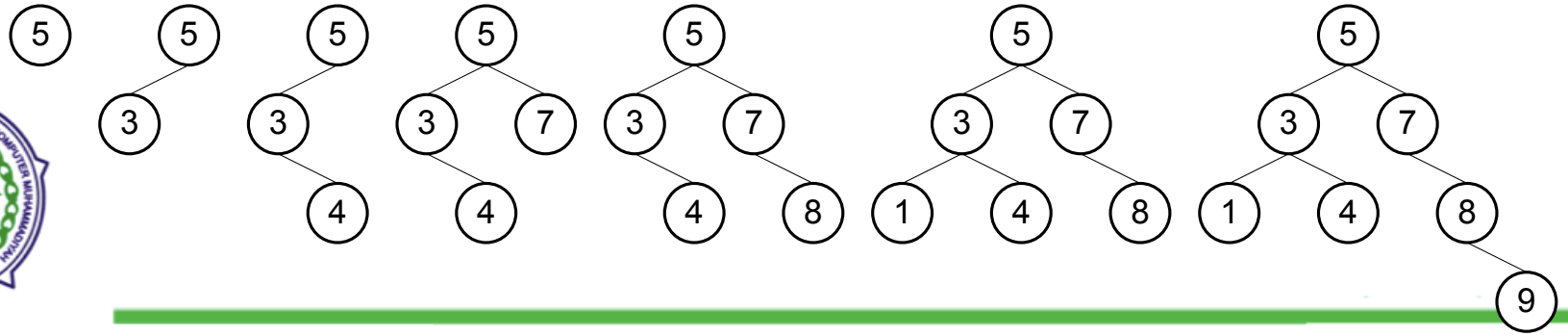
Pohon Terurut



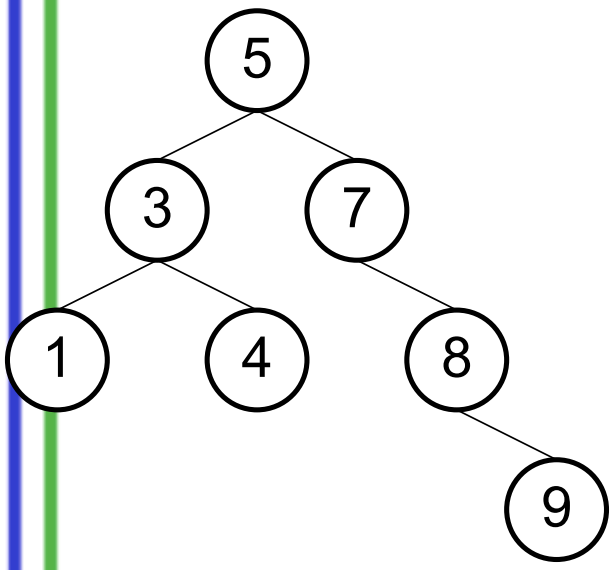
- Pohon Terurut (*Ordered Tree*) adalah sebuah pohon biner yang setiap node pada sub-tree kiri nilainya lebih kecil dari nilai *parent*-nya dan setiap node pada sub-tree kanan memiliki nilai lebih besar dari *parent*-nya

Binary Search Tree (BST)

- BST adalah jenis pohon terurut yang digunakan untuk menyimpan data sehingga memudahkan pencarian kembali data tersebut
- Gambar di bawah adalah ilustrasi penyisipan setiap node pada BST jika data yang masuk berturut-turut adalah 5, 3, 4, 7, 8, 1, dan 9
- Pencarian data pada BST dimulai dari root ke bawah, dengan ketentuan jika nilai yang dicari lebih kecil dari nilai pada node maka pencarian dilanjutkan ke sub-tree kiri dan jika nilainya lebih besar dari nilai node maka pencarian berlanjut ke sub-tree kanan



Keseimbangan BST

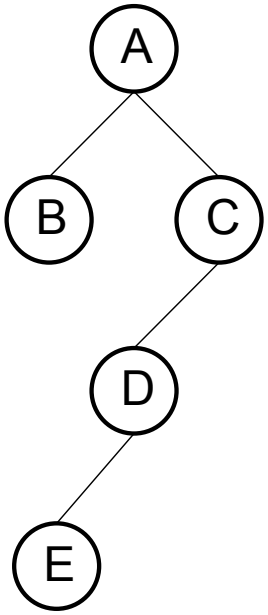


- BST yang umum tidak memperhatikan masalah keseimbangan pohon
- Contohnya BST di samping bukanlah pohon seimbang, karena ada node (dengan nilai 7) yang memiliki BF = +2
- Hal ini membuat pencarian suatu nilai pada BST menjadi tidak merata
- Supaya pencarian menjadi lebih merata sebaiknya sebuah BST dijadikan pohon yang seimbang

AVL-Tree

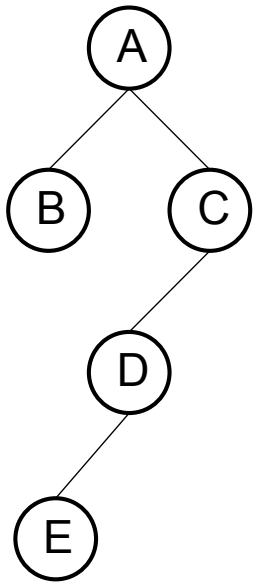
- AVL-Tree dinamakan berdasarkan dua orang penemunya, *G.M. Adelson-Velsky* dan *E.M. Lendis*
- AVL-Tree adalah sebuah *self-balancing binary search tree*, atau BST yang dengan sendirinya selalu seimbang
- Setiap ada penambahan nilai baru maka AVL-Tree akan secara otomatis melakukan pemeriksaan apakah masih seimbang atau tidak
- Jika tidak seimbang, untuk menyeimbangkan kembali dilakukan **rotasi node**

Arah Keseimbangan Tree



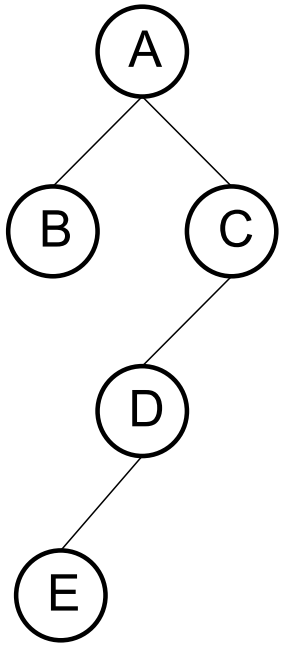
- Suatu *tree* atau sub-tree disebut berat ke kiri jika *root*-nya memiliki BF < -1
- Suatu *tree* atau sub-tree disebut berat ke kanan jika *root*-nya memiliki BF > +1
- Pemberat dari suatu tree atau sub-tree adalah daun yang memiliki jarak terjauh
- Dalam contoh,
  - tree tersebut dikatakan berat ke kanan dengan pemberat node E
  - Sub-tree dengan *root* C dikatakan berat ke kiri dengan pemberat node E

Arah Keseimbangan Tree



- Cabang pemberat dari suatu tree – jika ada – adalah cabang kedua yang menghubungkan *root* dengan pemberatnya
- Dalam contoh,
  - Cabang pemberat tree adalah cabang yang menghubungkan node C dan node D
  - Cabang pemberat sub-tree dengan *root* C adalah cabang yang menghubungkan node D dan node E

Arah Keseimbangan Tree



- Suatu tree disebut berat masuk jika cabang pemberatnya berbeda arah dengan cabang di atasnya
- Suatu tree disebut berat keluar jika cabang pemberatnya sama arahnya dengan cabang di atasnya
- Pada contoh,
  - Tree tersebut dikatakan berat masuk
  - Sub-tree dengan *root* C dikatakan berat keluar

Aturan I Rotasi Node

- *Root* akan diganti oleh salah satu node pada jalur *root* → node pemberat, dengan aturan :
  - Jika tree **berat keluar**, node **kedua** (anak dari *root*) akan naik menggantikan *root*
  - Jika tree **berat masuk**, maka node **ketiga** (cucu *root*) naik menggantikan *root*, sub-tree *root* lama pada arah pemberat menjadi sub-tree dari *root* baru

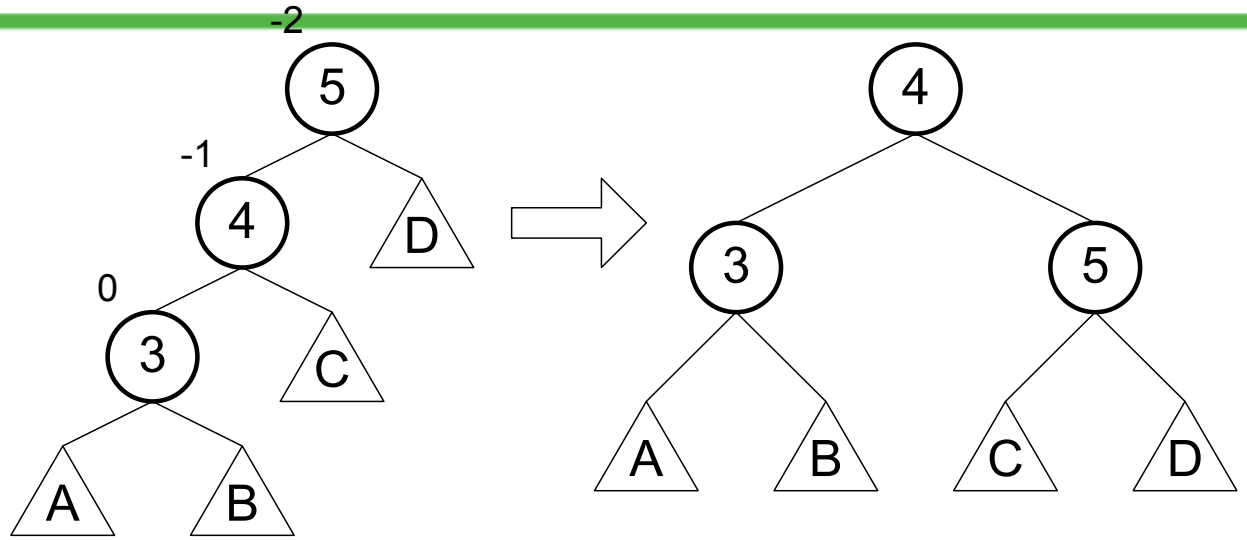
Aturan II Rotasi Node

- *Root* yang lama akan menjadi anak dari *root* yang baru pada sisi yang berlawanan dari arah keseimbangan (pemberat) tree
  - Jika tree berat ke kiri, maka root lama menjadi anak kanan dari root yang baru
  - Jika tree berat ke kanan, maka root lama menjadi anak kiri dari root baru

Aturan III Rotasi Node

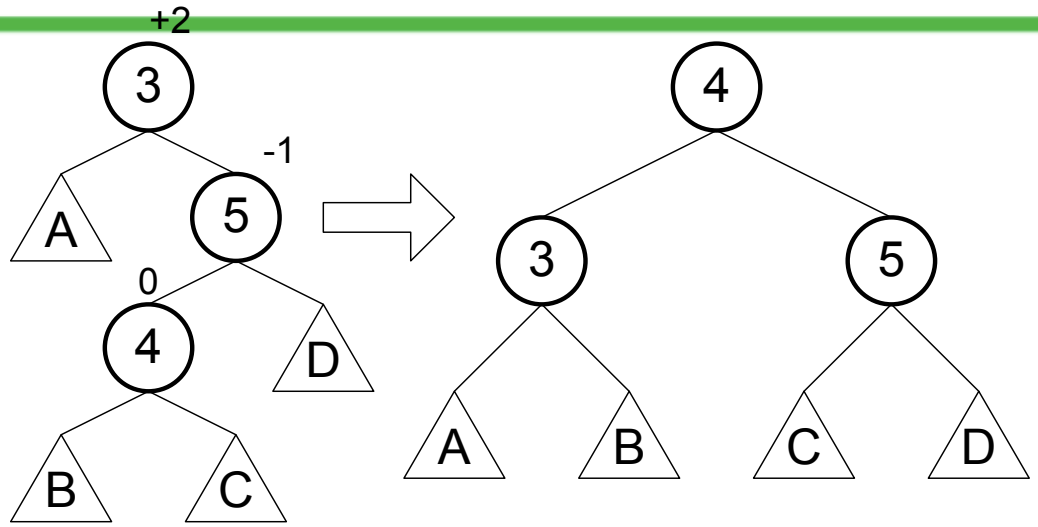
- Jika *root* yang baru jadi memiliki tiga sub-tree, pindahkan sub-tree tengah ke salah satu anaknya, dengan ketentuan :
  - Jika tadinya merupakan sub-tree kiri maka menjadi sub-tree kanan dari anak kiri
  - Jika tadinya merupakan sub-tree kanan maka menjadi sub-tree kiri dari anak kanan

Berat Keluar



- Node 5 dikatakan berat ke kiri karena BF-nya negatif
- Node 5 dikatakan berat keluar karena cabang 3-4 sama arahnya dengan cabang 4-5 yang ada di atasnya
- **Aturan I :** karena berat keluar maka node kedua (node 4) naik menggantikan node 5
- **Aturan II :** karena berat ke kiri maka node 5 menjadi anak kanan dari penggantinya (node 4)
- Sub-tree A, B dan D posisinya tetap
- **Aturan III :** sub-tree C yang tadinya adalah sub-tree kanan dari node 4 berpindah menjadi sub-tree kiri dari node 5

Berat Masuk



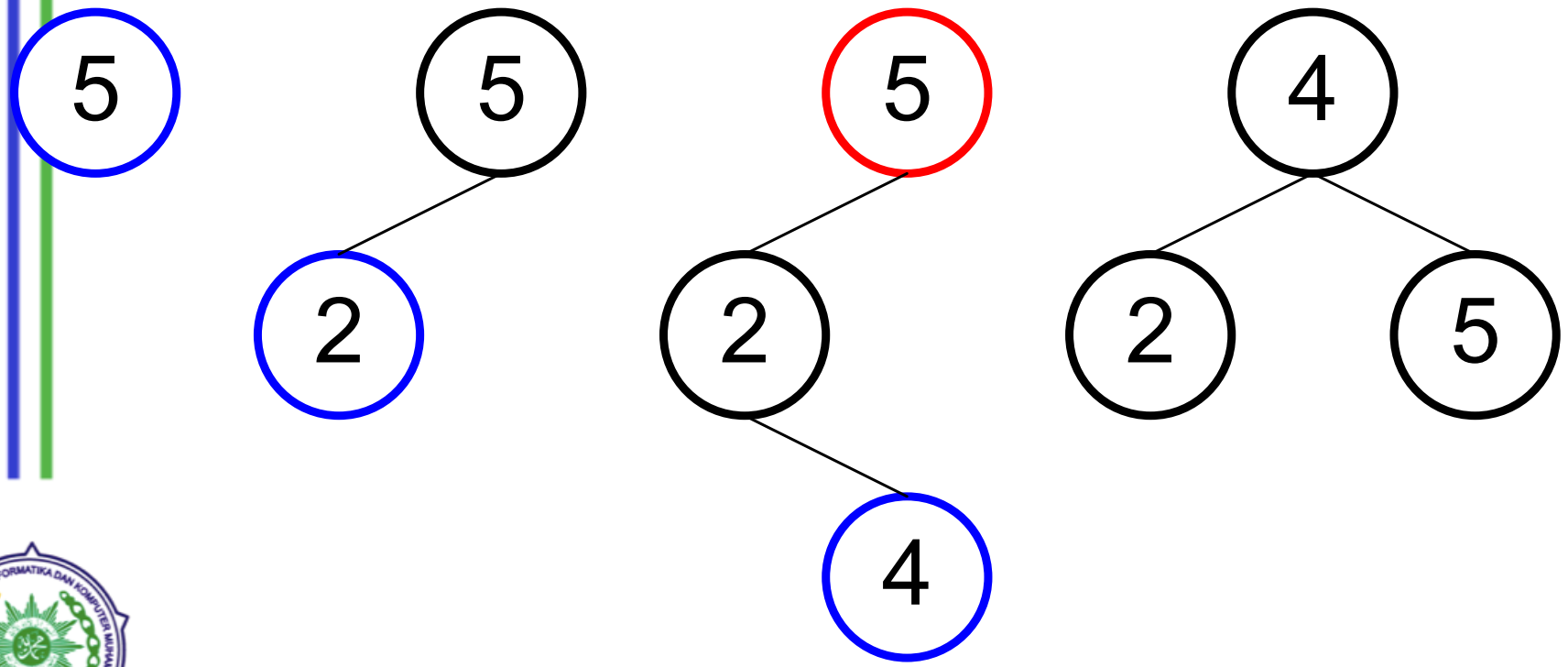
- Node 3 dikatakan berat ke kanan karena BF-nya positif
- Node 5 dikatakan berat masuk karena cabang 4-5 berbeda arahnya dengan cabang di atasnya (cabang 5-3)
- **Aturan I :** karena berat masuk maka node ketiga (node 4) naik menggantikan node 3, dan node 5 menjadi anak kanannya
- **Aturan II :** karena berat ke kanan maka node 3 menjadi anak kiri dari penggantinya (node 4)
- Sub-tree A dan D posisinya tetap
- **Aturan III :** sub-tree B yang tadinya adalah sub-tree kiri dari node 4 berpindah menjadi sub-tree kanan dari node 3, sementara sub-tree C yang tadinya adalah sub-tree kanan dari node 4 berpindah menjadi sub-tree kiri dari node 5

Langkah-Langkah Penambahan Kunci Baru pada AVL-Tree

- Penambahan kunci baru menganut aturan seperti pada BST, setiap kunci baru akan membentuk sebuah daun baru
- Dari daun yang baru terbentuk tersebut, telusuri dan hitung BF node-node ancestor-nya sampai menemui satu di antara tiga kondisi berikut :
  - Mencapai root
  - Menemukan node yang BF-nya nol
  - Menemukan node yang tidak seimbang
- Jika menemukan node yang tidak seimbang, lakukan rotasi node pada node tersebut

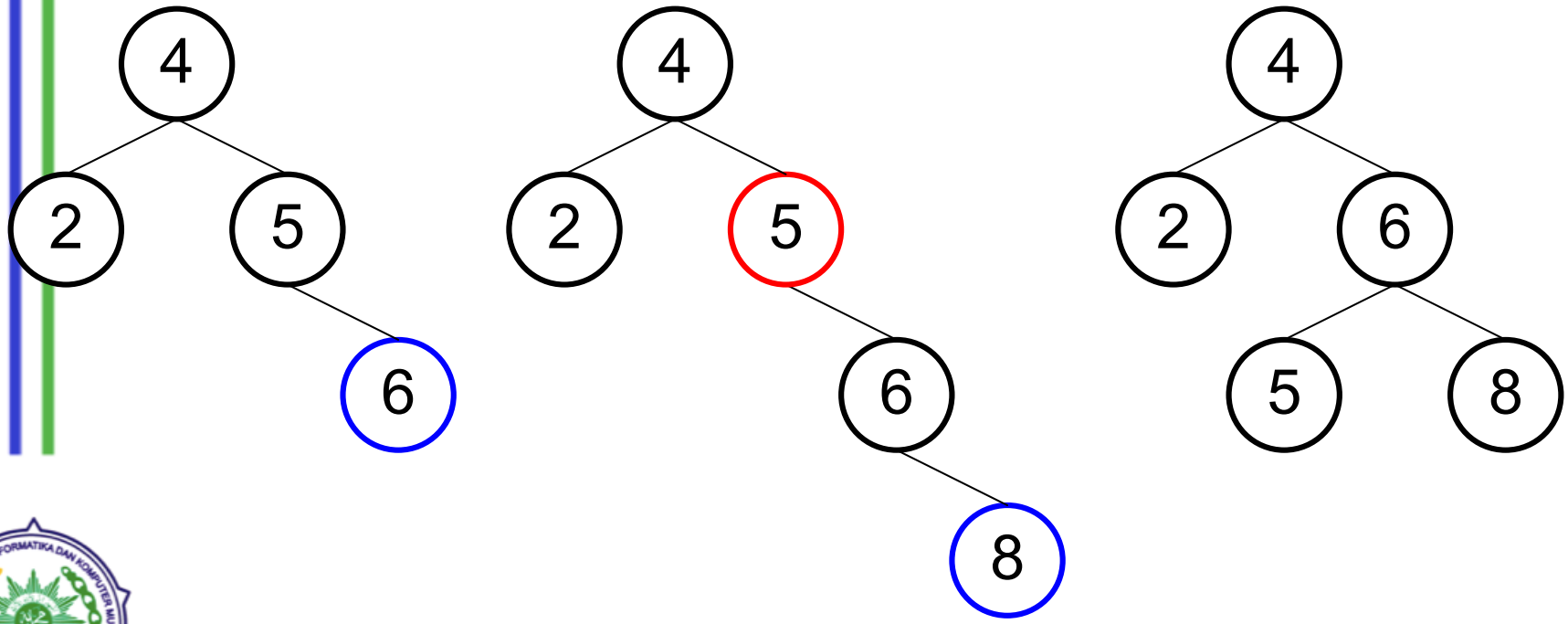
Contoh Pembentukan AVL-Tree

Keys : 5, 2, 4



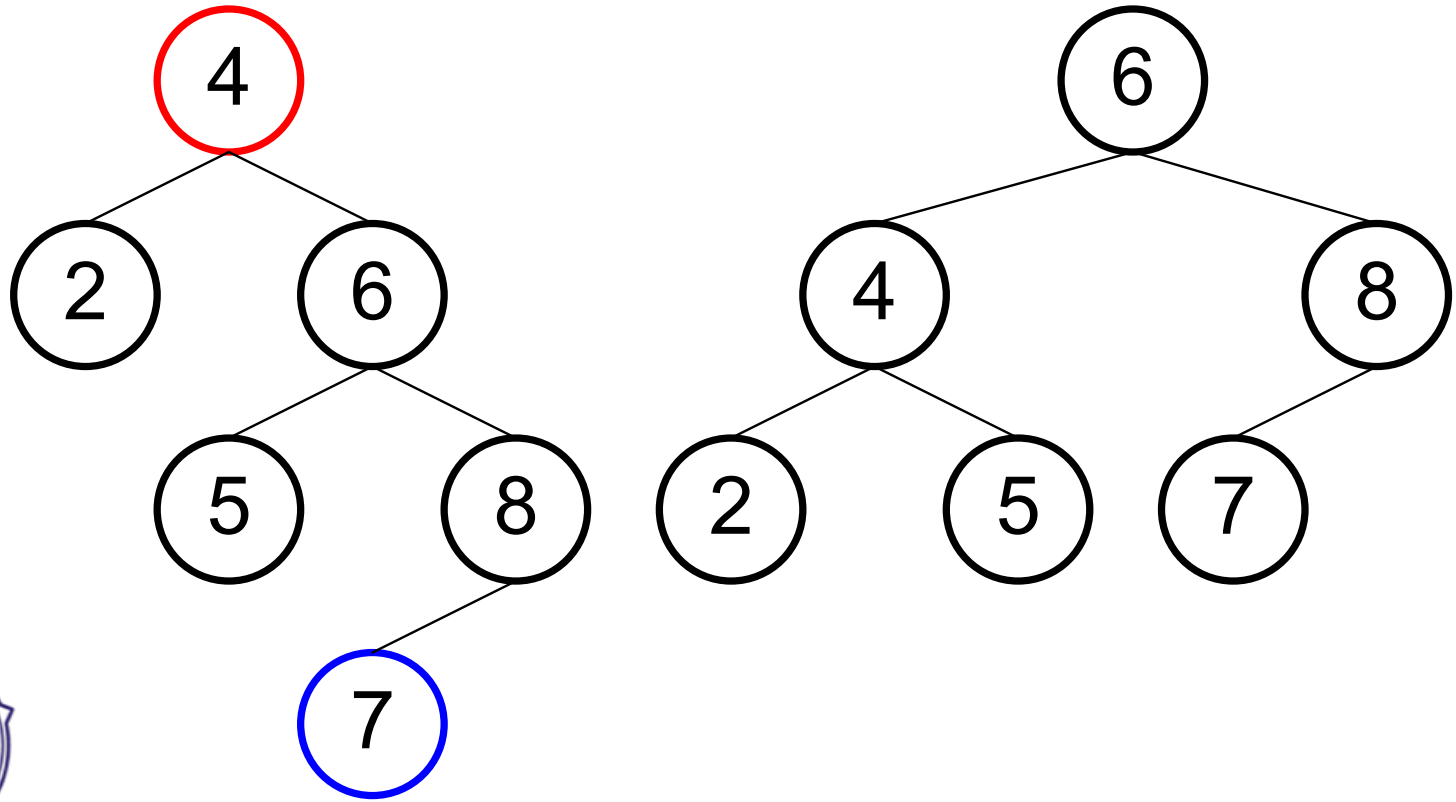
Lanjutan Pembentukan AVL-Tree

Keys : 6, 8



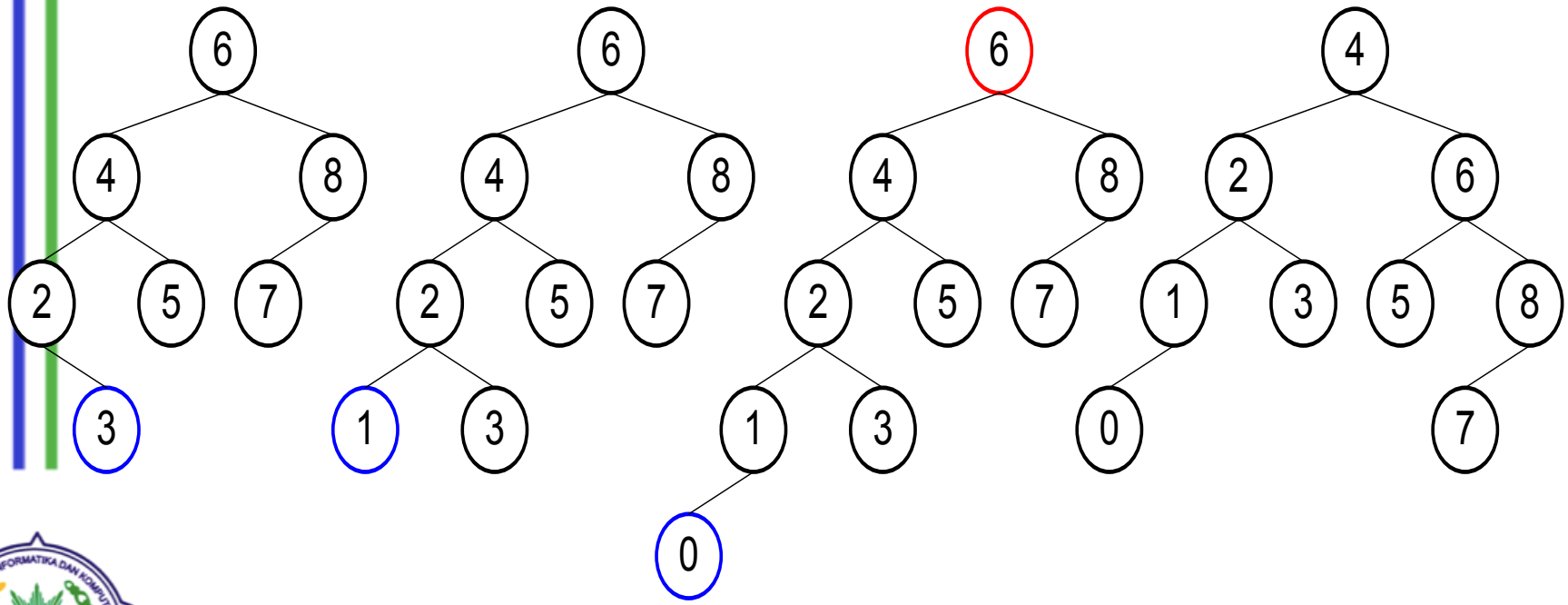
Lanjutan Pembentukan AVL-Tree

Keys : 7



Lanjutan Pembentukan AVL-Tree

Keys : 3, 1, 0



B-Tree

- B-Tree diciptakan oleh Rudolf Bayer dan Ed McCreight
- Penciptanya tidak menjelaskan arti dari "B" pada B-Tree, bisa jadi singkatan dari *Bayer* atau *Boeing*, tempat Bayer bekerja
- Ada yang mengartikan "B" pada B-Tree sebagai *Balanced*
- Saat ini, B-Tree merupakan implementasi yang paling banyak digunakan

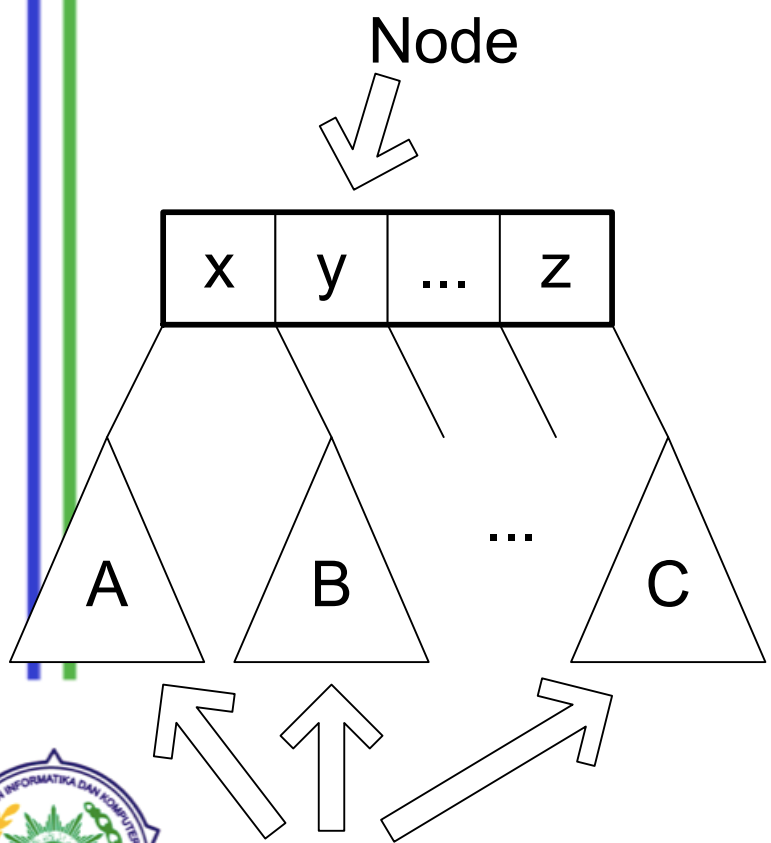
Definisi B-Tree

- B-Tree adalah sebuah struktur pohon multi-cabang yang dapat menyimpan lebih dari satu nilai pada setiap nodenya dan setiap daunnya berada pada level yang sama (seimbang)
- Setiap node pada B-Tree dapat menyimpan lebih dari satu nilai, namun ada kapasitas maksimalnya
- Kapasitas maksimal nilai yang dapat disimpan di tiap node inilah yang menentukan ordo atau *order* dari B-Tree

Order B-Tree

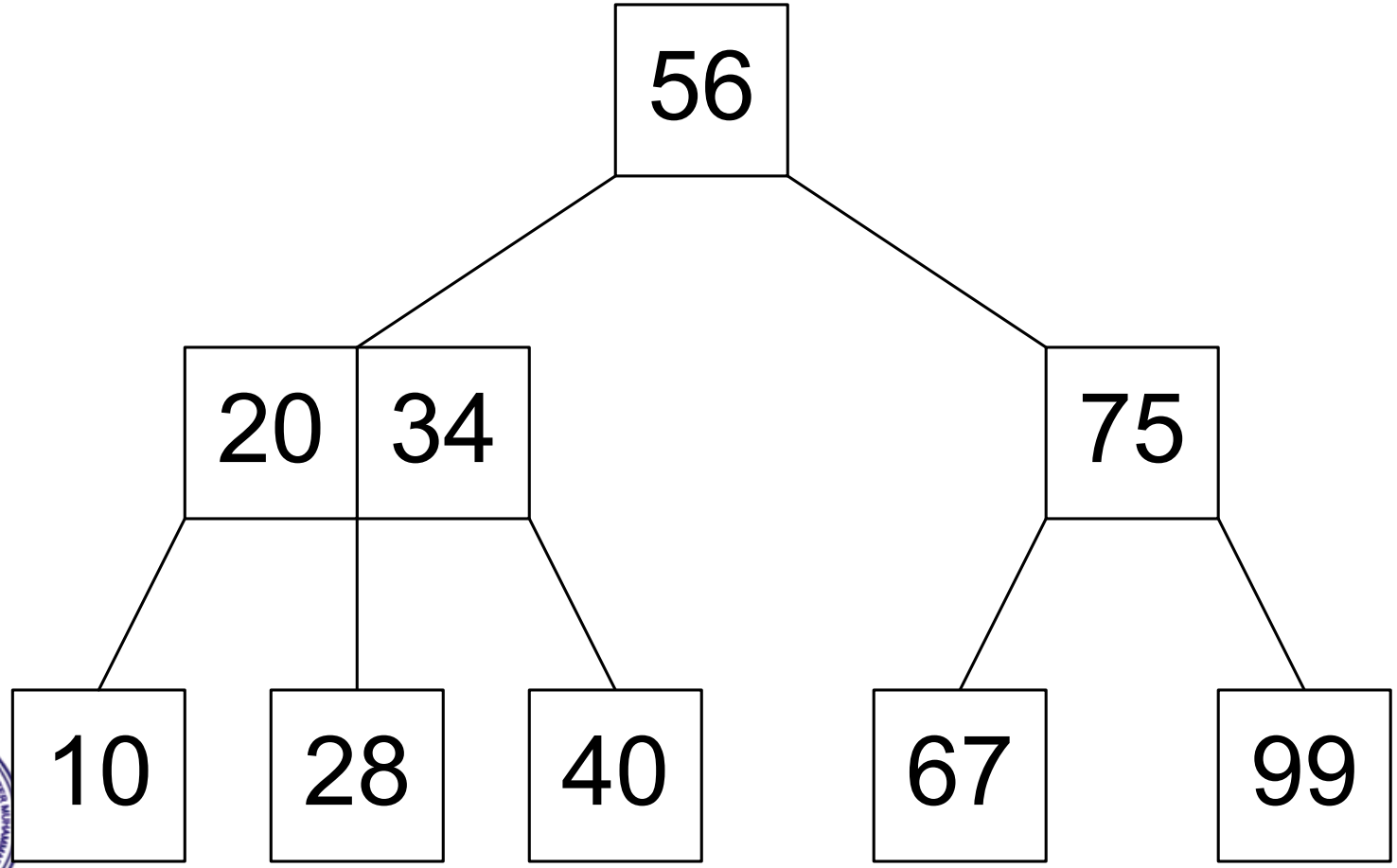
- Definisi dari Order B-Tree tidak jelas dan bervariasi dalam berbagai sumber
- http://www.semaphorecorp.com/btp/algo.html menyebutkan bahwa B-Tree order *n* adalah B-Tree yang tiap nodenya dapat menyimpan nilai sebanyak *n* sampai *2n*
- http://cis.stvincent.edu/carlsond/swdesign/btree/btree.html menyebutkan bahwa B-Tree order *n* adalah B-Tree yang setiap nodenya maksimal memiliki *n* anak atau dengan kata lain setiap nodenya maksimal dapat menyimpan *n-1* nilai
- http://searchsqlserver.techtarget.com/sDefinition/0,,sid87_gci508442,00.html juga menyatakan bahwa order menentukan banyaknya anak maksimal di tiap node
- Sumber lain ada yang menyebutkan B-Tree order *n* adalah B-Tree yang setiap nodenya maksimal menyimpan *n* nilai atau ada juga yang menyebutkan *2n* nilai

Bentuk Umum B-Tree



- Sebuah B-Tree yang tidak kosong paling tidak memiliki sebuah node (yaitu *root*) yang minimal menyimpan satu nilai
- Nilai-nilai pada node B-Tree tersusun urut secara *ascending* atau menaik, pada contoh, $x < y < \ldots < z$
- Setiap nilai pada node internal pasti memiliki anak di sebelah kiri dan kanannya, pada contoh sub-tree *A* adalah anak kiri dari *x* dan sub-tree *B* adalah anak kanan *x* sekaligus anak kiri dari *y*
- Nilai-nilai pada sub-tree kiri harus lebih kecil dari nilai parent dan nilai-nilai pada sub-tree kanan lebih besar dari nilai parent, pada contoh, $A < x < B < y < \ldots < z < C$

Contoh Bentuk B-Tree



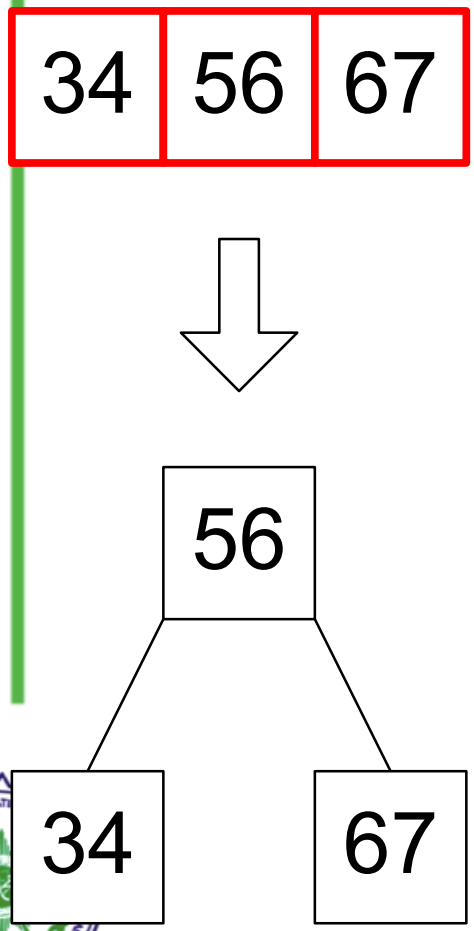
Langkah-Langkah Menambahkan Kunci Baru pada B-Tree

1. Jika B-Tree masih kosong, akan dibuat node baru sebagai *root* dan kunci baru dimasukkan pada *root* tersebut
2. Telusuri Node-node pada B-Tree dengan mencocokkan nilai pada node dengan kunci baru sampai ditemukan daun
3. Sisipkan kunci baru pada daun tersebut pada posisi yang tepat
4. Jika ada node yang menyimpan nilai lebih banyak dari kapasitas yang diperbolehkan maka terjadi *illegal state*
5. Jika terjadi *illegal state* maka lakukan operasi *split* sampai tidak ada *illegal state* lagi

Operasi Split pada Node B-Tree

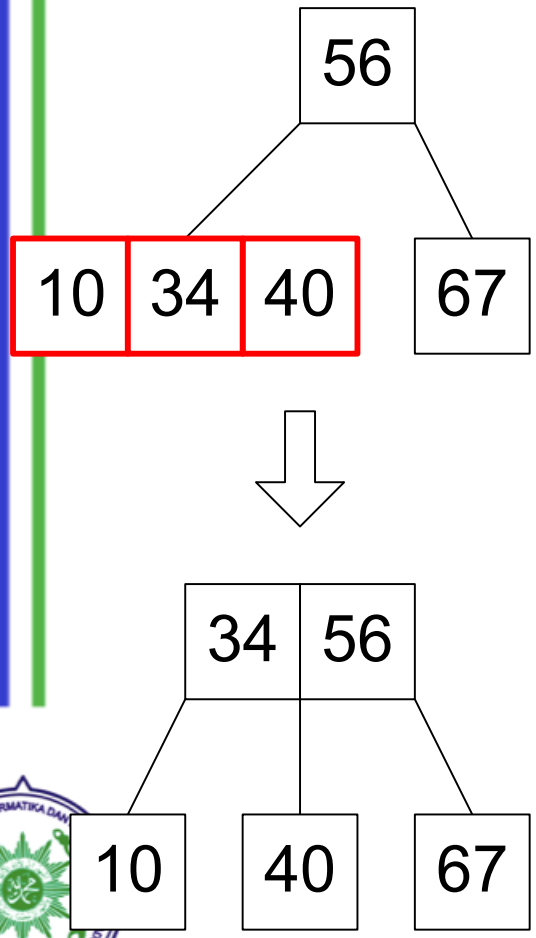
- Operasi Split dilakukan pada suatu node yang mengalami *illegal state*
- Nilai yang berada di tengah (sebut saja *x*) naik ke node di atasnya (jika tidak ada, maka membuat node baru)
- Nilai-nilai di sebelah kiri dan kanan memisahkan diri menjadi node-node terpisah, yang merupakan anak kiri dan anak kanan dari *x*

Ilustrasi Operasi Split (1)



- Semula node hanya terdiri dari dua nilai saja, yaitu 56 dan 67
- Ketika nilai baru (34) muncul maka terjadi *illegal state*
- Nilai yang tengah (56) naik ke atas (membuat node baru)
- Kedua nilai lainnya (34 dan 67) memisahkan diri ke node-node di kiri dan kanan

Ilustrasi Operasi Split (2)



- Dari contoh sebelumnya, ditambahkan nilai-nilai 40 dan 10, sehingga terjadi *illegal state* pada daun di kiri
- Nilai yang tengah (34) naik ke node atasnya, bergabung dengan nilai 56
- Nilai 10 membentuk node sendiri yang merupakan anak kiri dari 34, sementara nilai 40 membentuk node yang menjadi anak kanan 34 sekaligus anak kiri 56

Ilustrasi Operasi Split (3)

- Dari contoh sebelumnya, ditambahkan nilai-nilai 75 dan 99 sehingga terjadi *illegal state*
- Ketika nilai yang di tengah (75) naik ke node atasnya maka menjadikan node tersebut mengalami *illegal state* juga
- Operasi split dilakukan lagi pada node tersebut sehingga nilai yang di tengah (56) naik membentuk node baru dan seterusnya…