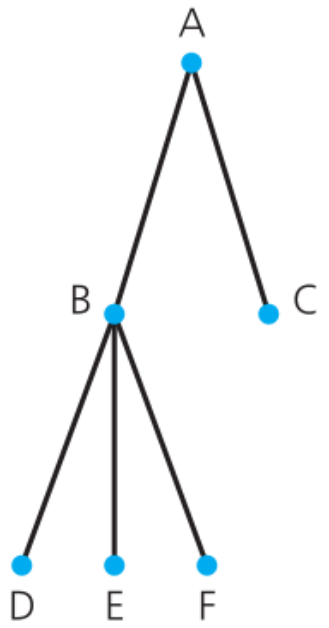
Struktur Data Pohon (2)

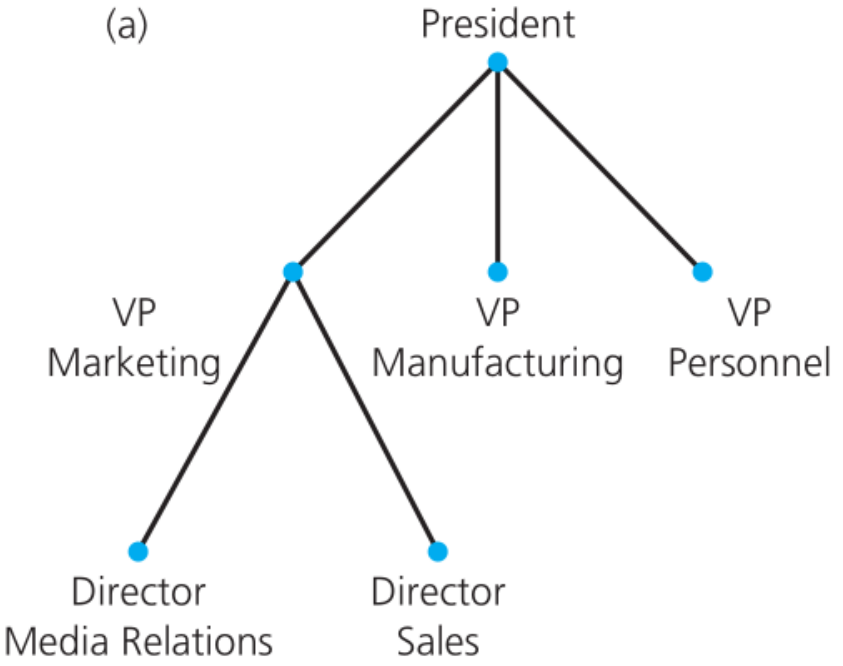
Outline

- Properti (sifat-sifat) Tree
- Binary Tree
- Implementasi
- Aplikasi konsep tree
- Advance Tree

Jenis tree

- **General tree** : ada satu root, jumlah subtree tidak dibatasi

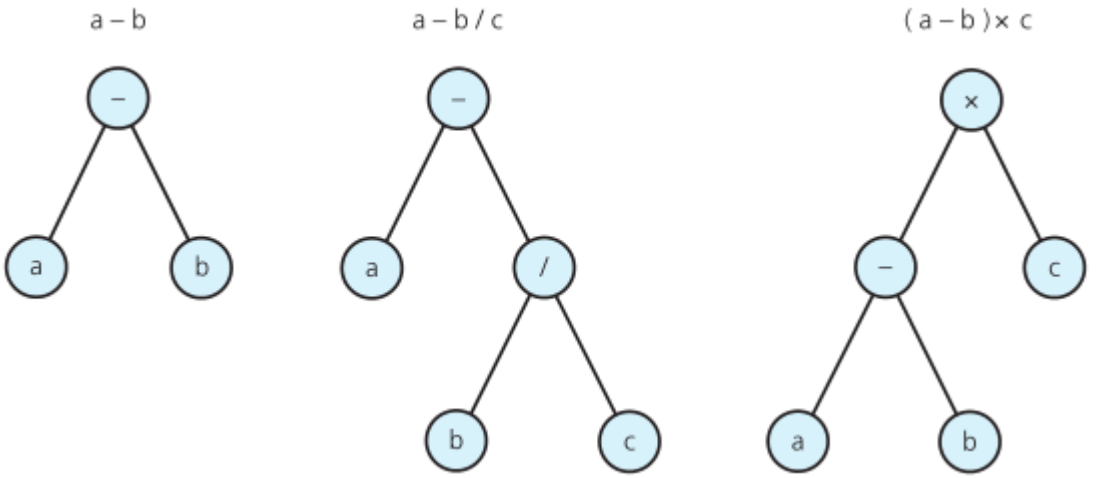




- N-arry tree : tiap node punya batas maksimal anak

Binary Tree

- N-arry tree dengan n=2
- Tiap node (kecuali node paling bawah/ daun) punya **maksimum** jumlah node anak 2, yaitu anak kanan dan anak kiri
- Binary tree banyak diterapkan di informatika, contoh untuk representasi ekspresi matematik

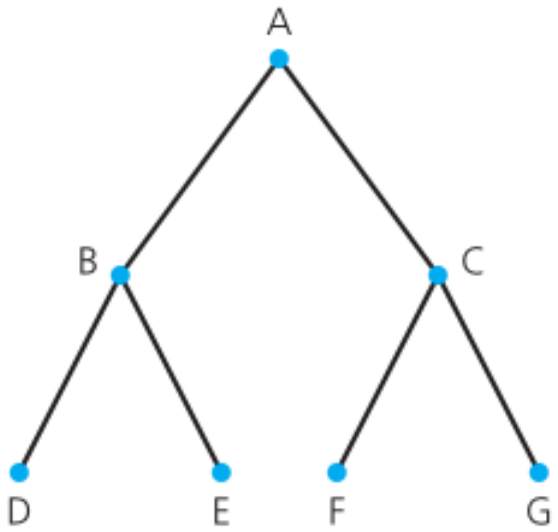


Apa yang ada didalam tiap node?

- Seperti struktur data sebelumnya, node memuat informasi / data
- Data ini bisa disearch, ditambah dihapus, dll
- Seperti pada pemrosesan ekspresi matematik, penafsiran data tergantung pada konteks penggunaan
- “a – b/c” disimpan jadi pohon
- Data array {31,11, 10,11,21,10} → disimpan jadi pohon

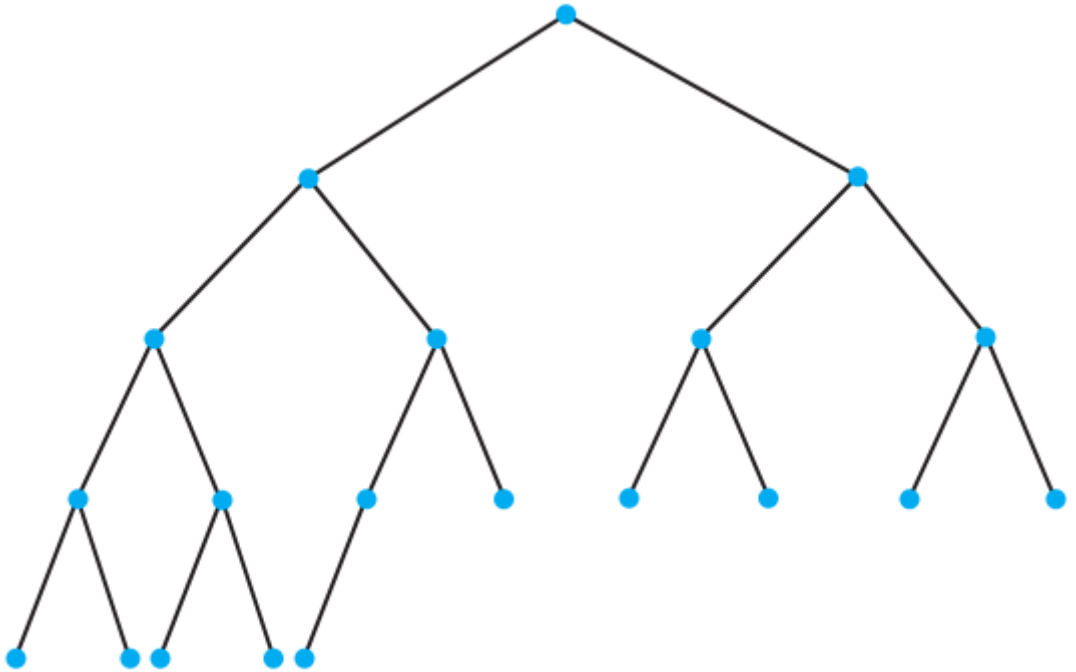
Jenis binary tree

- Full binary tree: tiap node (kecuali daun) memiliki dua node anak



(a)

- Complete binary tree: node daun paling kiri terdalam

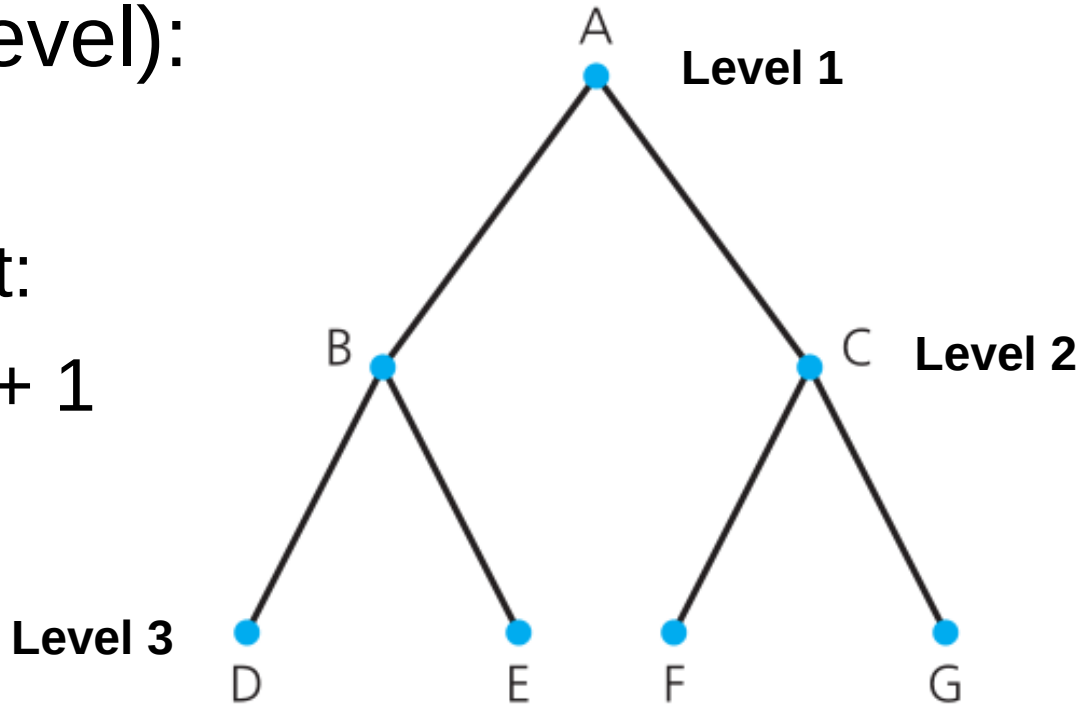


(b)

Tinggi Pohon

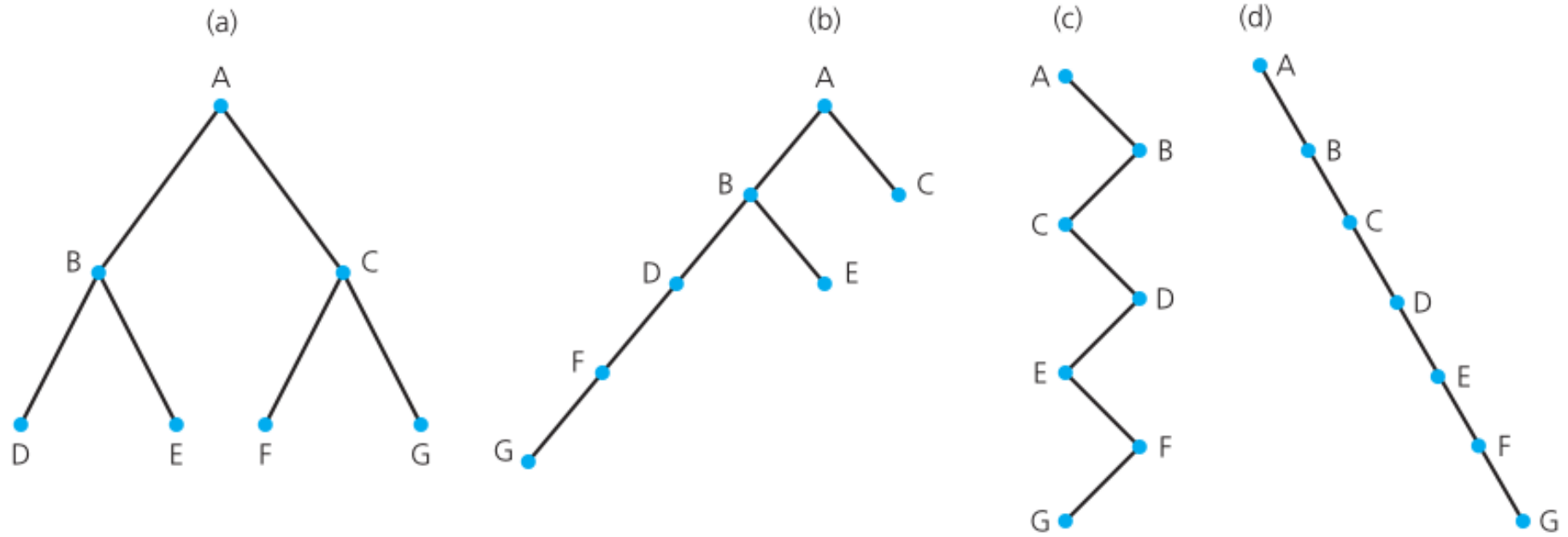
- Height of tree: jumlah node yang dilewati path (jalur) terpanjang dari node daun (leaf) keatas s.d root (akar)
- Menghitung height(level):
  - Node akar: level 1
  - Jika node bukan root:

    Level = level parent + 1



Tinggi (2)

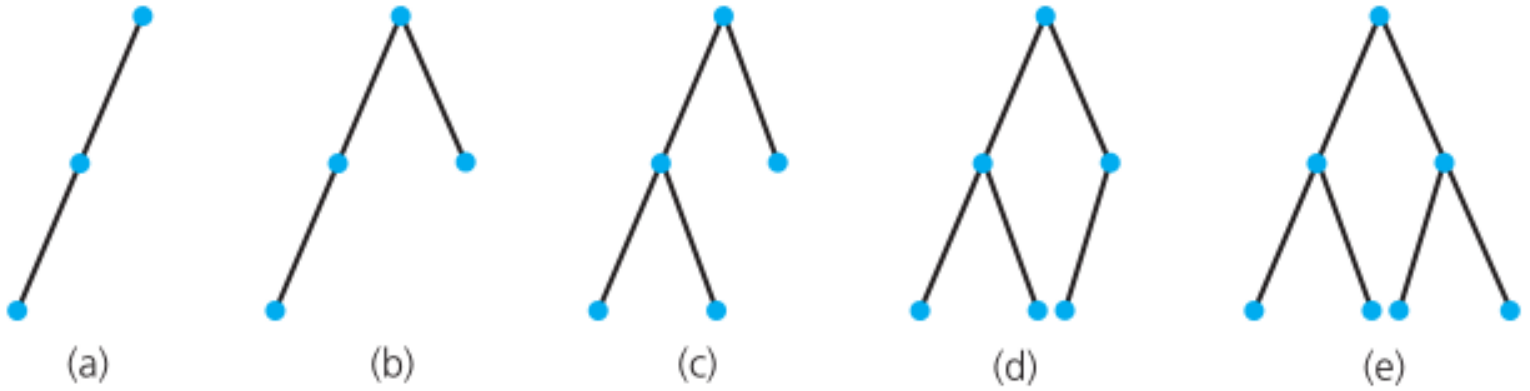
- Meski jumlah node sama, tinggi pohon bisa berbeda-beda tergantung susunan pohon



- Tinggi pohon: jalur paling panjang leaf to root

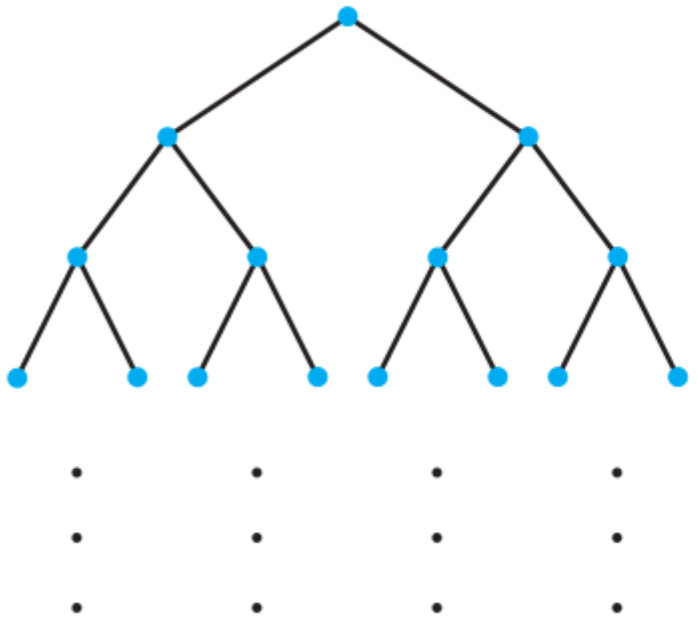
Max and Min height

- Suatu pohon dengan n node akan memiliki tinggi maksimum n level
- Agar tinggi minimum, tiap level pada binary tree harus diisi komplit
- Maka ini akan menjadi full binary tree



Relasi tinggi node & max node

- Suatu full binary tree dengan tinggi h akan memiliki total node: $2^h - 1$



Level	Number of nodes at this level	Total number of nodes at this level and all previous levels
1	$1 = 2^0$	$1 = 2^1 - 1$
2	$2 = 2^1$	$3 = 2^2 - 1$
3	$4 = 2^2$	$7 = 2^3 - 1$
4	$8 = 2^3$	$15 = 2^4 - 1$
.	.	.
.	.	.
.	.	.
h	2^{h-1}	$2^h - 1$

Tinggi minimum jika diketahui jumlah node n

- Jika tree adalah full binary maka $n = 2^h - 1$ untuk suatu h integer
- Jika tidak full, maka $n < 2^h - 1$ dan juga ada batas bawah $(2^h - 1) < n$
- Tinggi minimum pohon untuk jumlah node n adalah bilangan h terkecil yang memenuhi :

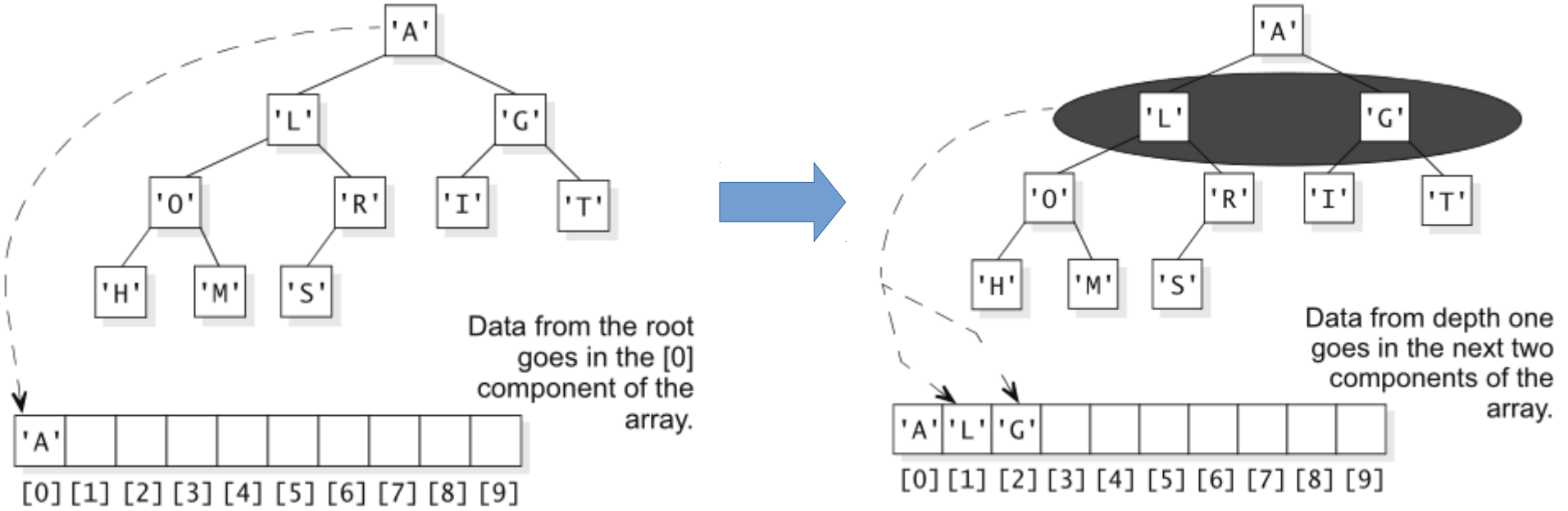
$$2^{h-1} - 1 < n \leq 2^h - 1$$

$$2^{h-1} < n + 1 \leq 2^h$$

$$h - 1 < \log_2(n + 1) \leq h$$

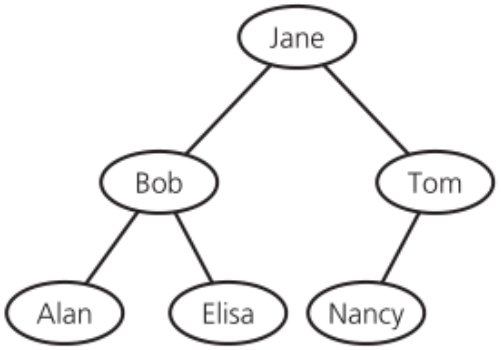
Representasi Tree

- Tree disimpan sebagai array di memori (fisik)
- Tapi secara logis merepresentasi Tree



Tree sebagai Array

- Free: pointer menunjuk lokasi array masih kosong



	item	leftChild	rightChild
0	Jane	1	2
1	Bob	3	4
2	Tom	5	–1
3	Alan	–1	–1
4	Elisa	–1	–1
5	Nancy	–1	–1
6	?	–1	7
7	?	–1	8
8	?	–1	9
.	.	.	.
.	.	.	.
.	.	.	.

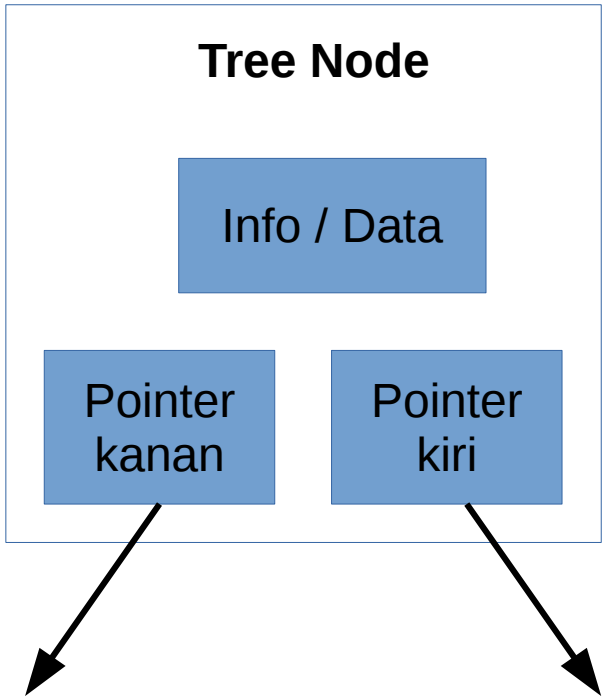
root: 0

free: 6

Free list

Binary tree dengan alokasi dinamis

- Implementasi dengan array memiliki kelemahan yaitu ukuran array tetap (jumlah node fix)

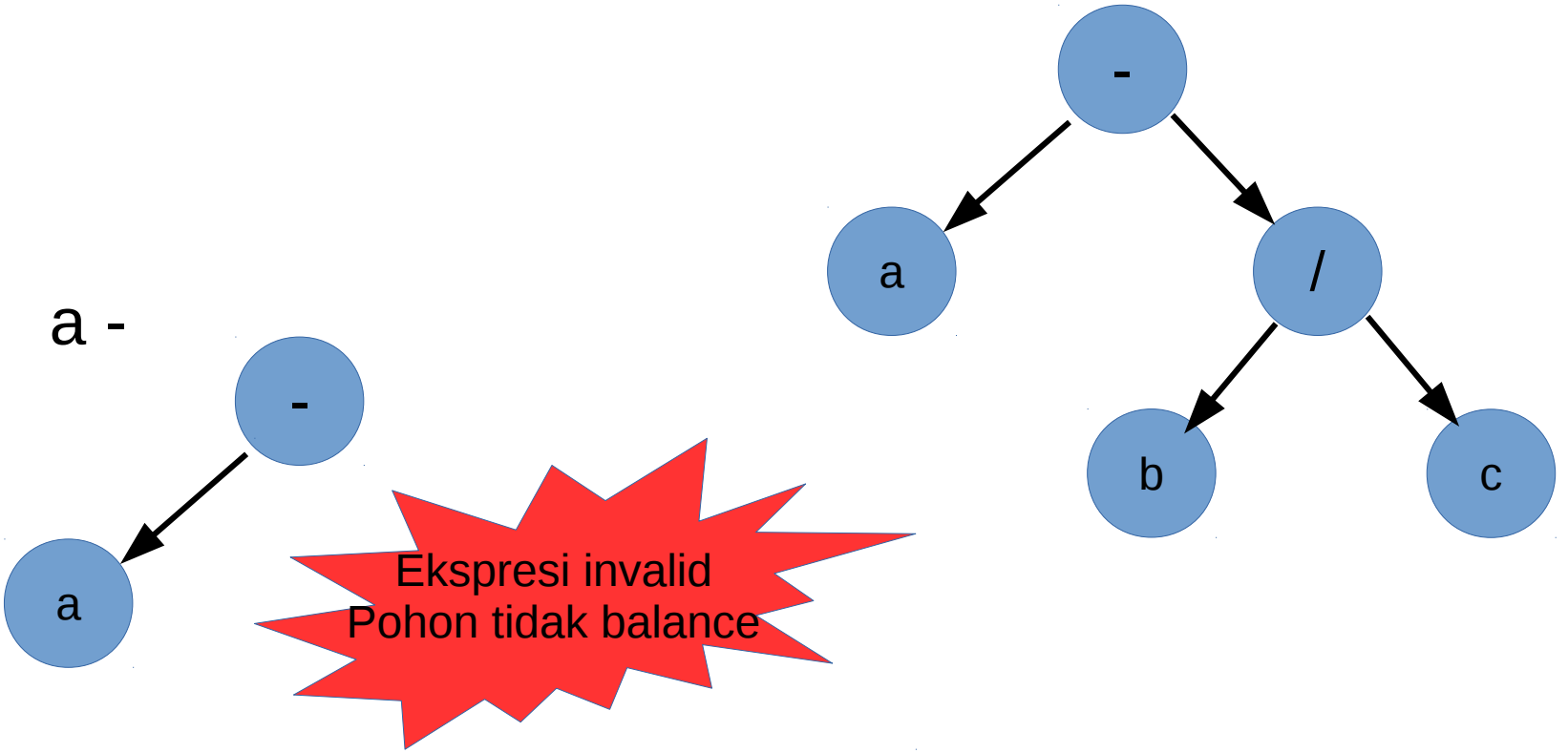



```
class TreeNode
{
public:
  int Data;      //tree menampung data karakter
  TreeNode *Lchild;    //pointer ke node kiri
  TreeNode *Rchild;    //pointer ke node kanan
};

class BinaryTree
{
private:
  TreeNode *Root;
public:
  BinaryTree() { Root=0; } //constructor
  //operasi lainnya
  void InsertNode(TreeNode* n);
};
```

Contoh Penerapan Tree

- Dalam compiler, tree digunakan untuk parsing: yaitu memeriksa ekspresi aritmatik
- Contoh: a – b / c

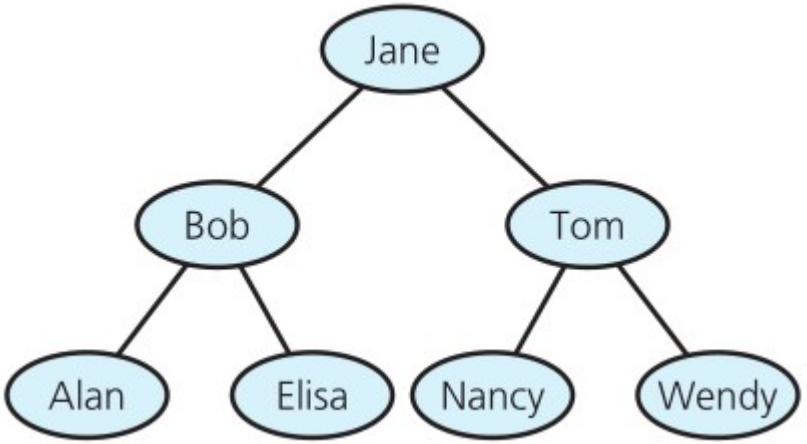


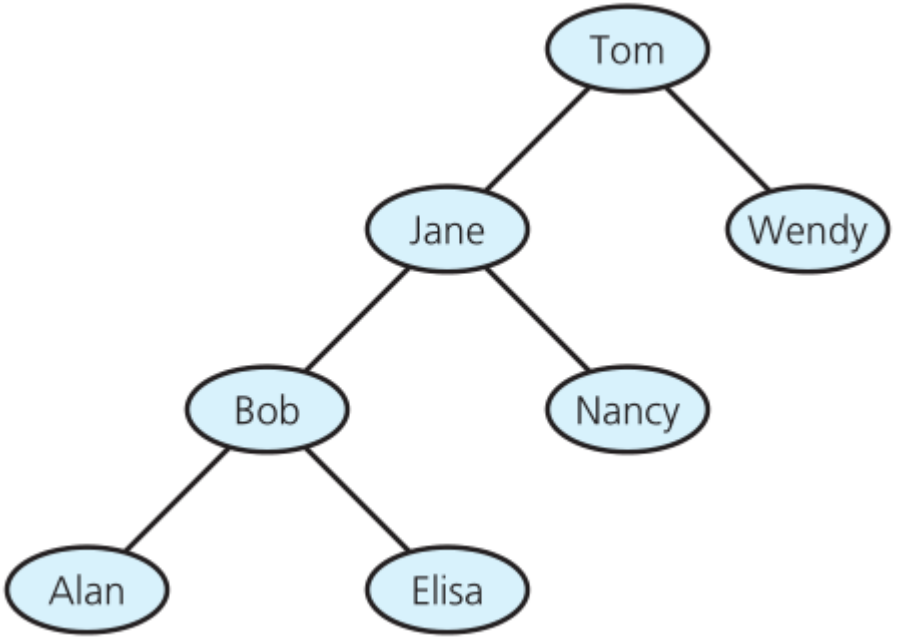
Contoh penerapan

- Tree digunakan untuk binary search
- Binary search adalah algoritma pencarian dengan membagi data dengan syarat kondisi data terurut (dibahas pada materi rekursif)
- Bagaimana tree diterapkan untuk search ?
  - Bandingkan info pada node apakah sama dengan nilai yang dicari
  - Jika nilai < info pada node, telusuri anak kanan
  - Jika nilai > info pada node, telusuri anak kiri

Binary search tree

- Langkah 1 : susun data ke dalam Tree

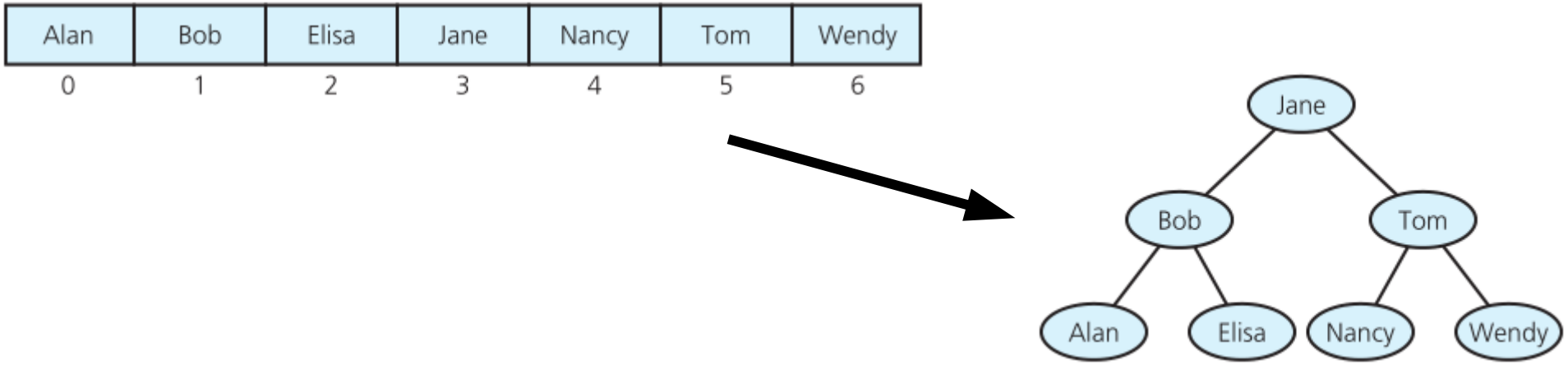




- Ada banyak cara menyusun untuk data yang sama, bagaimana cara menyusun ?

Binary search tree

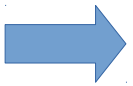
- Bentuk tree mempengaruhi efisiensi proses seach
- Semakin balance tree, proses search lebih efisien

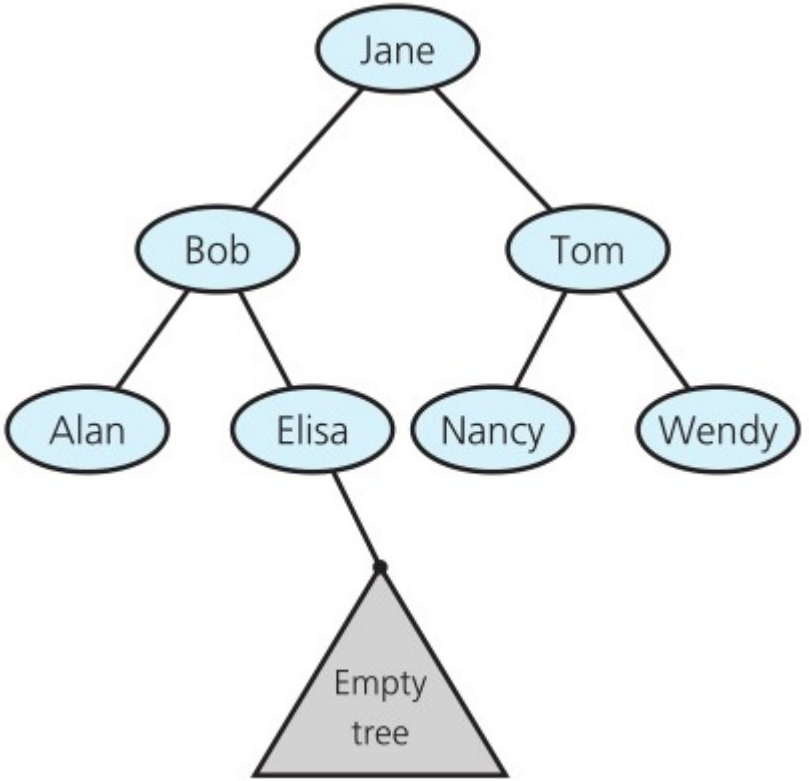


Membangun tree

- Kondisi awal: tree kosong (tidak ada node)
- Tambah data: “Jane” → jadi root
- Tambah: “Bob” → jadi anak kanan (“Bob”<”Jane”)
- dst

```
nameTree = a new, empty binary search tree
nameTree.insert("Jane")
nameTree.insert("Bob")
nameTree.insert("Alan")
nameTree.insert("Elisa")
nameTree.insert("Tom")
nameTree.insert("Nancy")
nameTree.insert("Wendy")
```





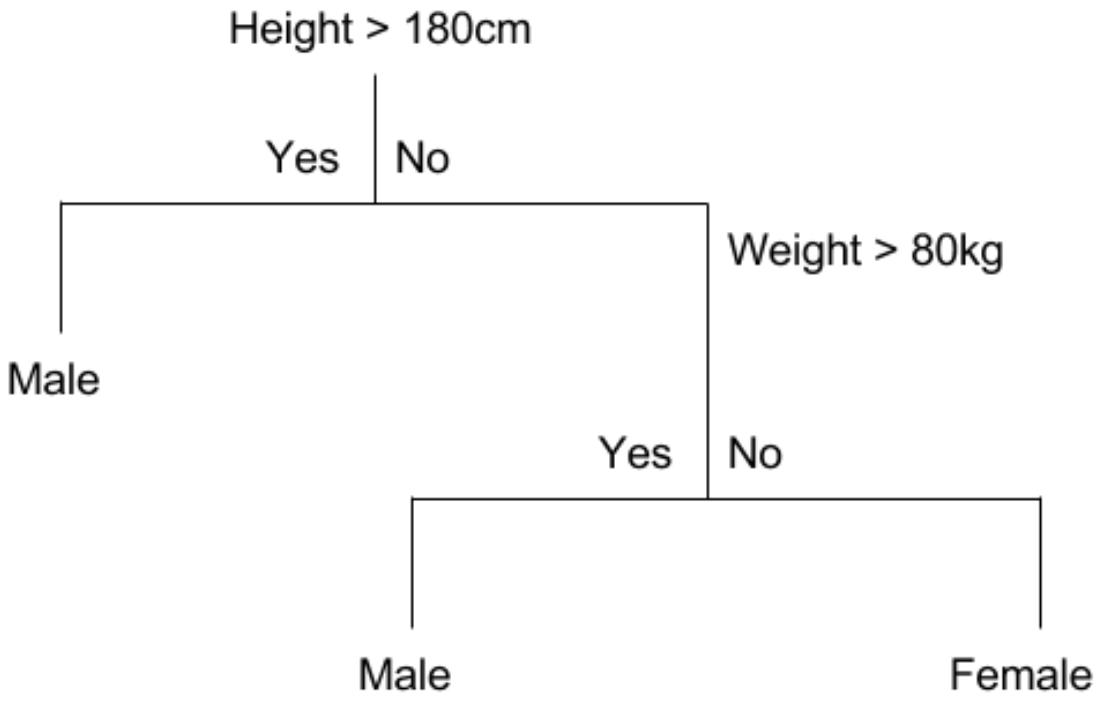
Decission Tree

- Melakukan penelusuran dari root ke leaf dengan menentukan evaluasi : jika Ya ke kanan, tidak ke kiri
- Decission tree banyak digunakan di Artificial Intelligence
- Ada dua jenis :
  - **Classification** tree : data bersifat discrete (fit vs tidak fit, cocok vs tidak cocok) splitting data into partitions, binary recursive partitioning
  - **Regression** tree : target variabel bersifat continuous
- Tujuan decission tree : membagi (partisi) data ke dalam cluster (kelompok) yang lebih sempit/khusus

Contoh

- Classification tree

  perhatikan sifat data binary (yes / no)



Contoh

- Regression tree

Perhatikan pada pohon, dari root semua nilai tinggi (height) < 1.85 jadi bukan boolean (T/F)