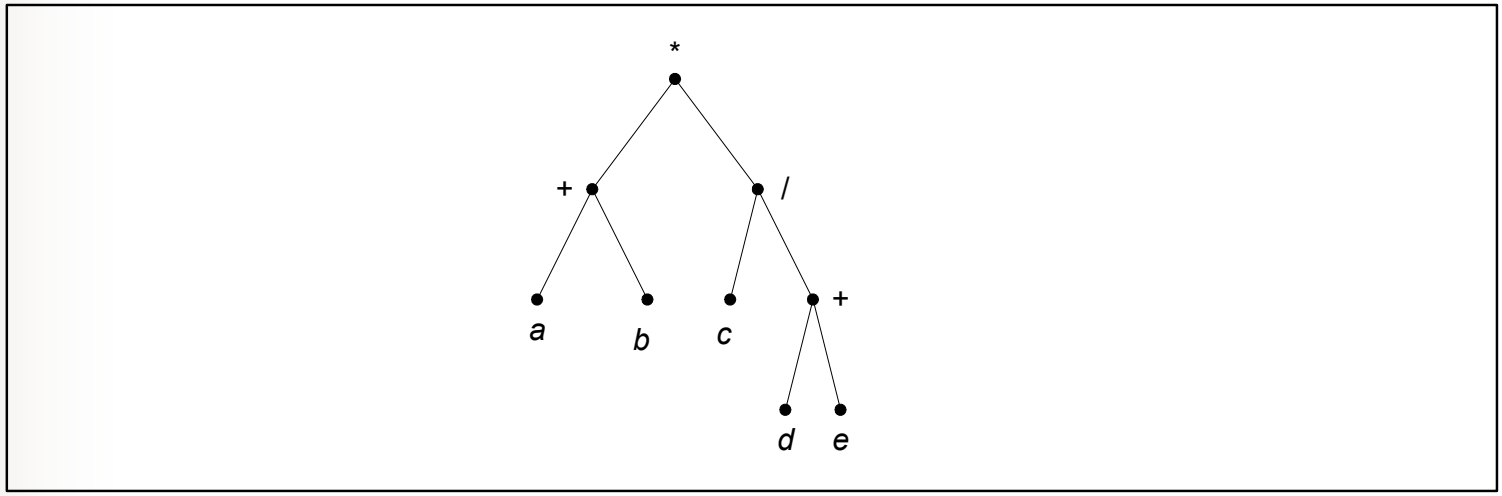
Terapan Pohon Biner

1. Pohon Ekspresi



Gambar Pohon ekspresi dari $(a + b)*(c/(d + e))$

daun → *operand*
simpul dalam → operator

Pohon Ekspresi

Pohon ekspresi digunakan untuk mengevaluasi ekspresi yang ditulis dalam notasi:

- Prefix : Operator mendahului dua buah operand-nya. Contoh: *+ab/c+de
- Infix : Operator berada diantara dua buah operand. Contoh: (a+b)*(c/(d+e))
- Postfix: Kedua operand mendahului operatornya. Contoh: ab+cde+/*

```
procedure BangunPohonEkspresiDariPostfix(input P1, P2, ..., Pn : elemen
                                         postfix, output  T : pohon)
{ Membangun pohon ekspresi dari notasi postfix
  Masukan: notasi postfix, setiap elemennya di simpan di dalam tabel P
  Keluaran: pohon ekspresi T
}
Deklarasi
   i : integer
   S : stack
   T1, T2 : pohon

Algoritma
   for i ← 1 to n do
      if Pi = operand then
         Buat (create) satu buah simpul untuk Pi
         Masukkan (push) pointer-nya ke dalam tumpukan S
      else  { Pi = operator }
         Ambil (pop) pointer dua upapohon T1 dan T2 dari puncak tumpukan S
         Buat pohon T yang akarnya adalah operator dan upapohon kiri
        dan upapohon  kanannya masing-masing T1 dan T2
      endif
   endfor
```

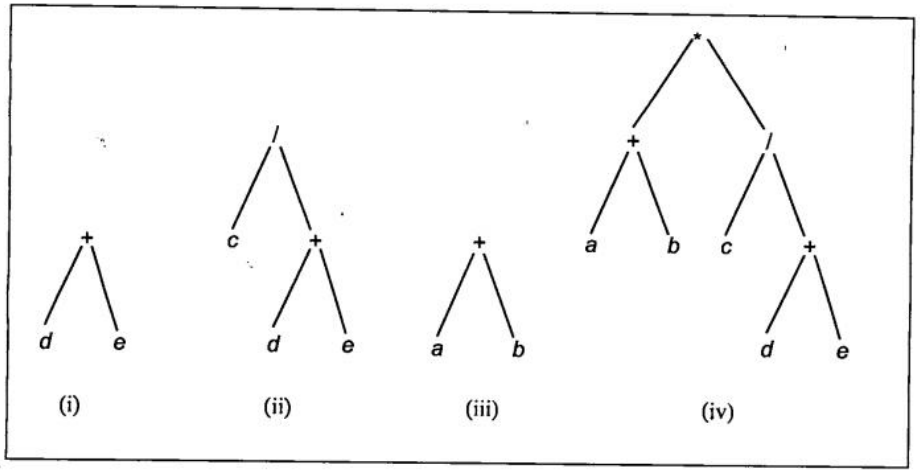
Algoritma 9.3 Pembentukan pohon ekspresi dari notasi *postfix*

Contoh 9.7

Gambarkan pembentukan pohon ekspresi dari ekspresi $(a + b)*(c/(d + e))$ di atas.

Penyelesaian:

Pohon ekspresi dari notasi *infix* dibangun dari bawah ke atas dengan memperhatinkan urutan prioritas pengerjaan operator. Operator / dan * mempunyai prioritas lebih tinggi daripada opaertor + dan -. Mula-mula dibentuk upapohon untuk $(d + e)$, kemudian upapohon untuk $c/(d+e)$, upapohon untuk $(a + b)$ dan akhirnya penggabungan upapohon $(a + b)$ dengan upapohon $d/(d+e)$. Pohon ekspresi diperlihtakan pada Gambar 9.25. ■



Gambar 9.25 Pembentukan pohon ekspresi dari $(a + b)*(d(d + e))$

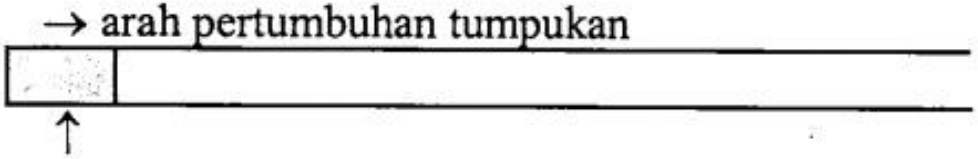
Pembentukan Pohon Ekspresi dari Notasi *Postfix*

Jika diberikan ekspresi dalam notasi *postfix*, kita dapat membangun pohon ekspresinya dengan algoritma di bawah ini. Untuk itu, kita membutuhkan sebuah tabel dan sebuah tumpukan (*stack*).

1. Setiap elemen (*operand* dan operator) dari notasi *postfix* yang panjangnya n disimpan di dalam tabel sebagai elemen $P_1, P_2, \ldots, P_n$.

1	2	3	4	5	6	7	8	$n = 9$
a	*b*	+	*c*	*d*	*e*	+	/	*

2. Tumpukan S menyimpan *pointer* ke simpul pohon biner (bayangkanlah tumpukan tumbuh dari "kiri" ke "kanan").

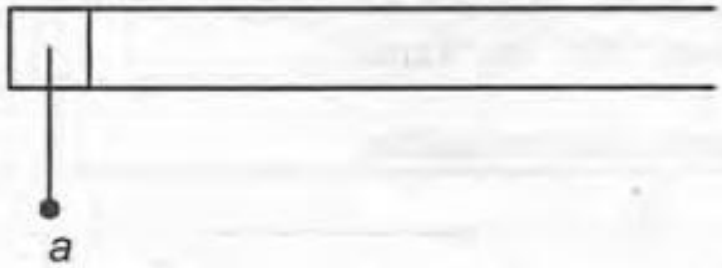


Contoh 9.8

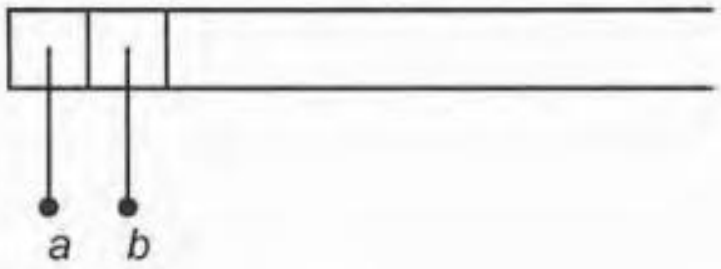
Terapkan algoritma `BangunPohonEkspresiDariPostfix` untuk membangun pohon ekspresi dari notasi *postfix* *a b + c d e + / **

Penyelesaian:

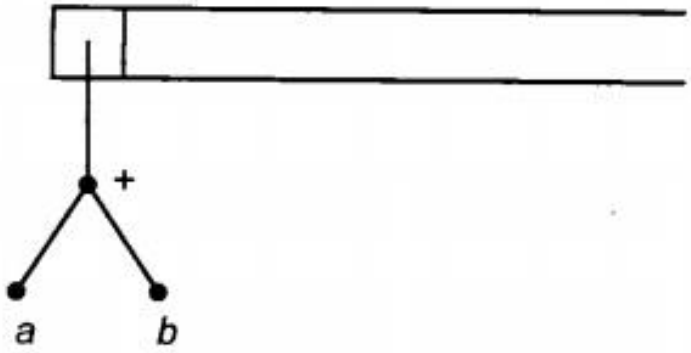
(i) Mulai dari elemen *potfix* pertama, P_1. Karena P_1 = '*a*' = *operand*, buat simpul untuk P_1, *push pointer*-nya ke dalam tumpukan *S*.



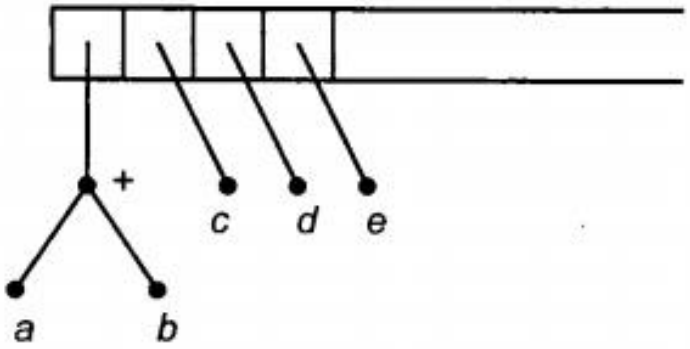
(ii) Baca P_2. Karena P_2 = '*b*' = *operand*, buat simpul untuk P_2, *push* pointer-nya ke tumpukan *S*.



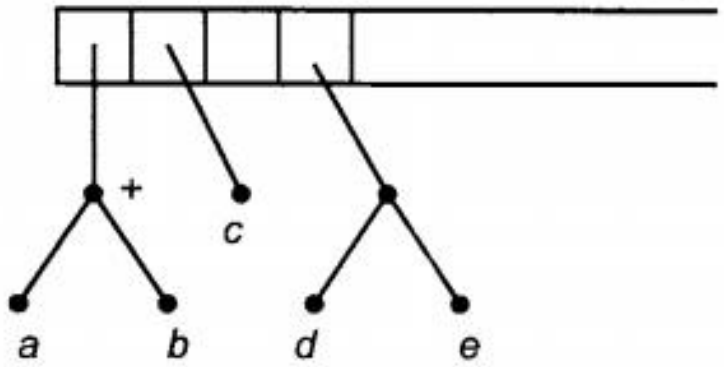
(iii) Baca P_3. Karena P_3 = '+' = *operator*, buat pohon T dengan 'a' dan 'b' sebagai anak.



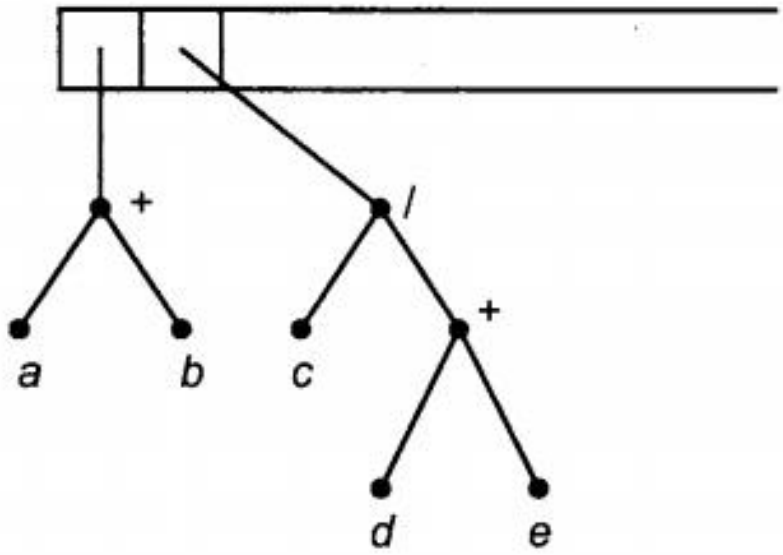
(iv) Baca P_4, P_5, P_6. Karena P_4, P_5, P_6 = *operand*, buat simpul untuk P_4, P_5, dan P_6. *Push pointer*-nya ke dalam tumpukan S.



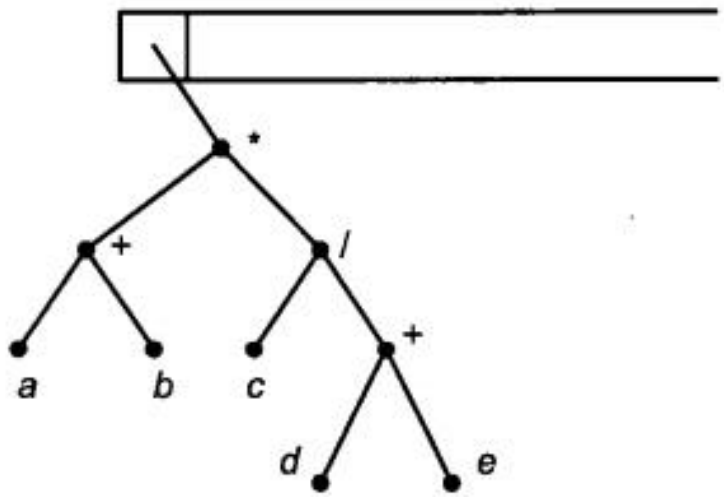
(v) Baca P_7. Karena P_7 = '+' = *operator*, buat pohon T dengan 'd' dan 'e' sebagai anak.



(vi) Baca P_8. Karena P_8 = '/' = *operator*, buat pohon T dengan 'c' dan '+' sebagai anak.



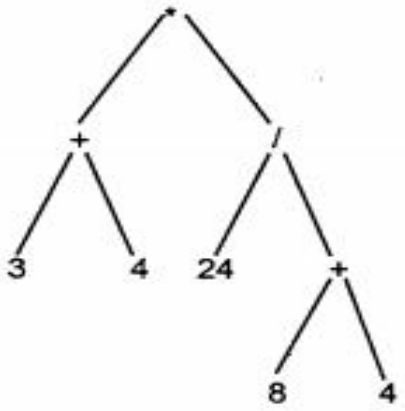
(vii) Baca P_9. Karena P_9 = '*' = operator, buat pohon T dengan '+' dan '*' sebagai anak.



Karena semua elemen tabel P sudah habis dibaca, maka tumpukan S akan berisi *pointer* yang menunjuk ke akar pohon ekspresi. ■

Contoh 9.9

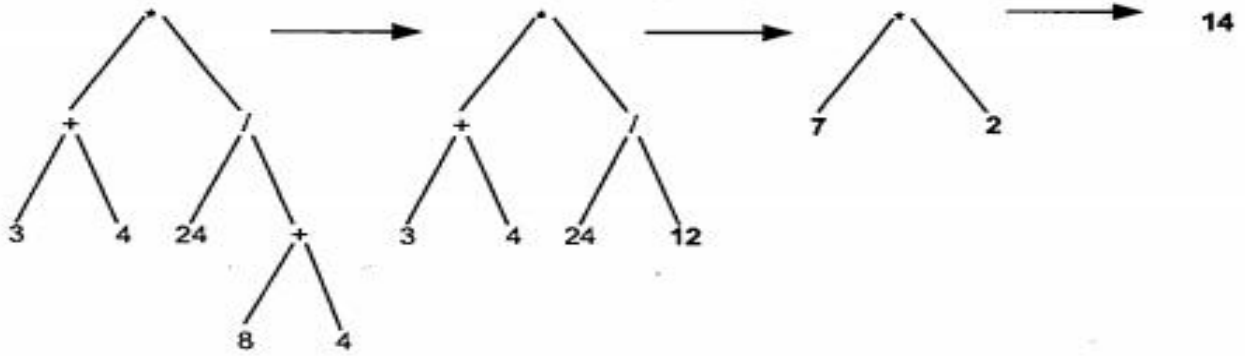
Evaluasi pohon ekspresi berikut:



Penyelesaian:

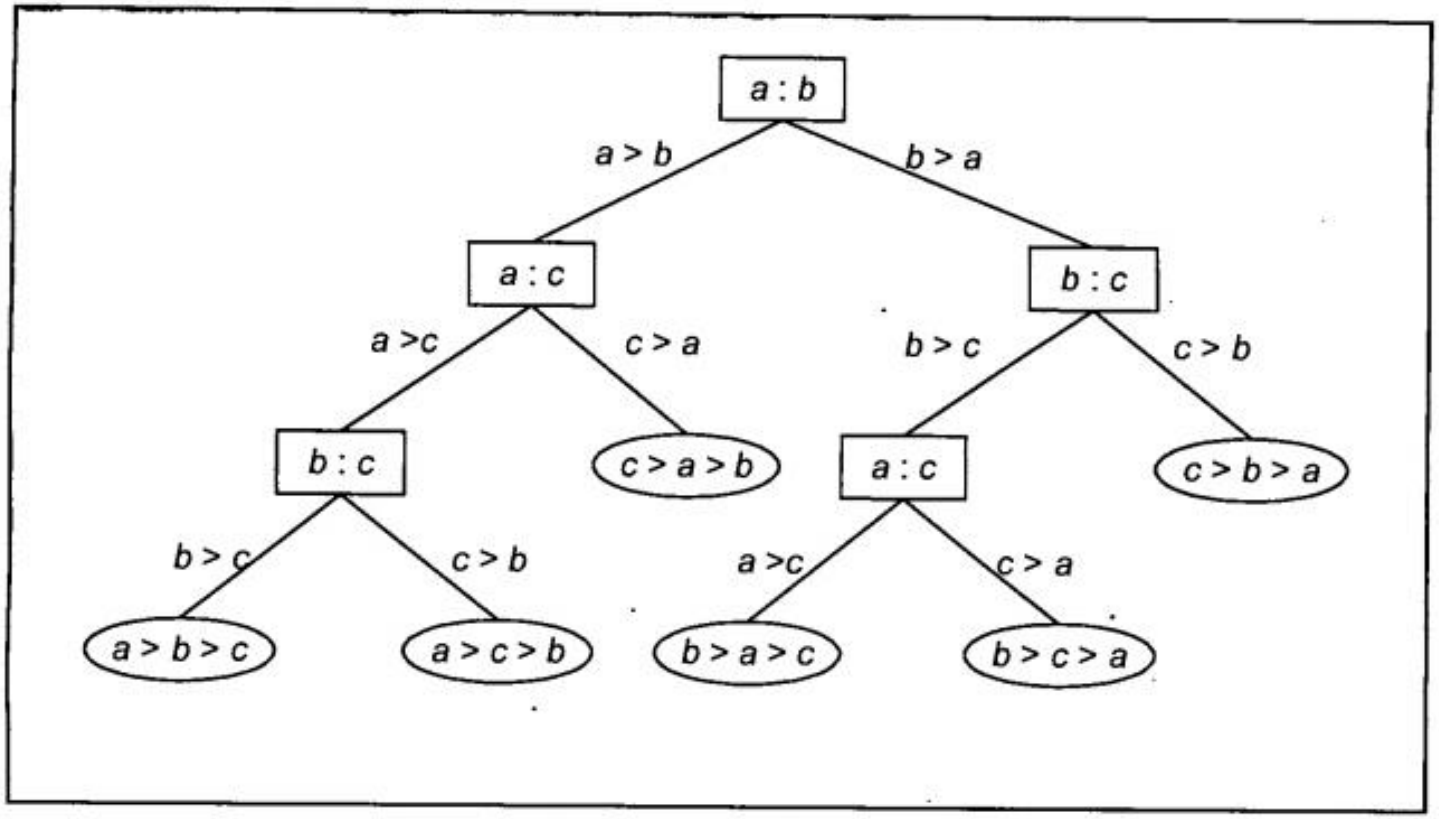
Pohon ekspresi dievaluasi mulai dari bawah ke atas. Dua buah daun (yang merepresentasikan *operand*) diperasikan dengan operator yang menjadi orangtuanya. Nilai evaluasi sementara (dicetak tebal) disimpan pada simpul orangtua tadi, dan kedua *operand* yang dioperasikan dihapus. Pada akhir evaluasi, simpul akar merepresentasikan nilai ekspresi keseluruhan (dalam hal ini 14).

Tahapan evaluasi pohon ekspresi diperlihatkan pada gambar berikut:



Pohon Keputusan

- Digunakan untuk memodelkan persoalan yang terdiri dari rangkaian keputusan yang mengarah ke solusi.
- Tiap simpul menyatakan keputusan, sedangkan daun menyatakan solusi.



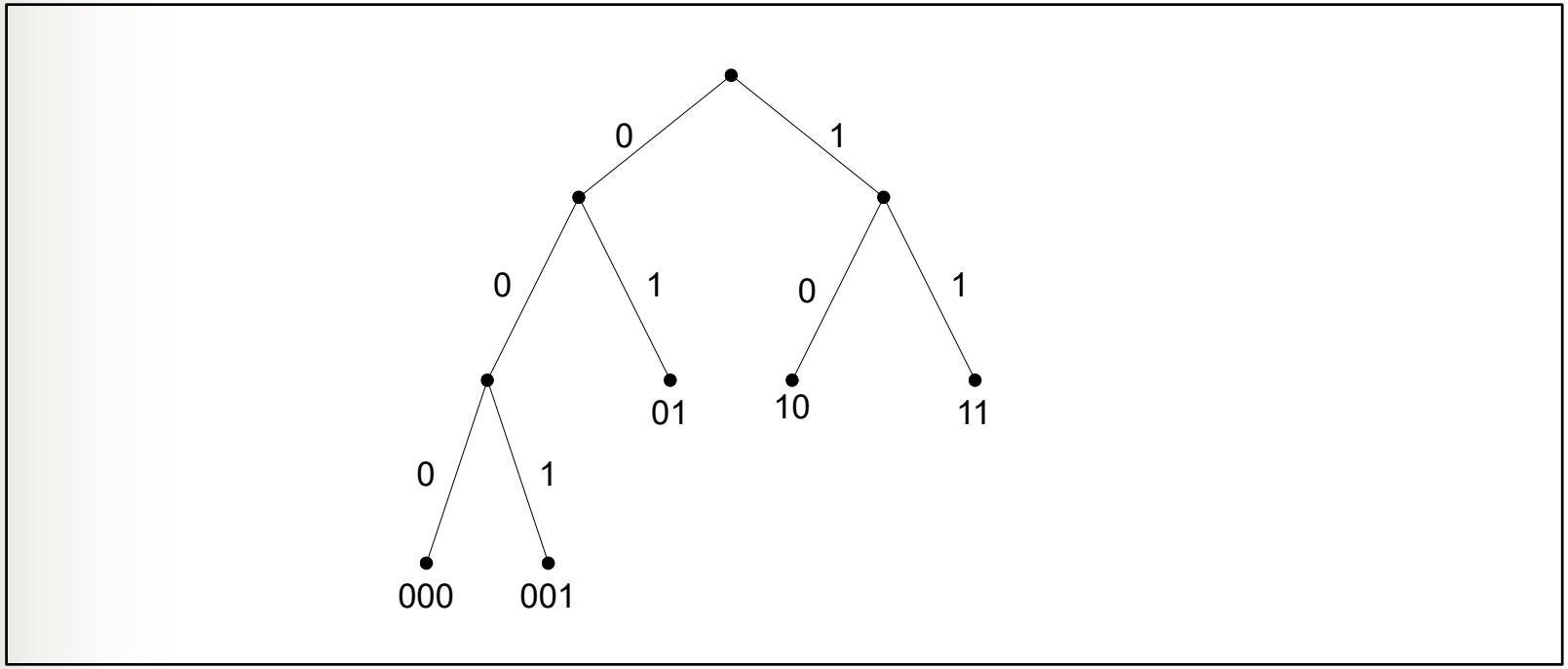
Gambar 9.26 Pohon keputusan untuk mengurutkan 3 buah elemen

Kode Awalan (prefix code)

- Merupakan himpunan kode (ex: kode biner) sedemikian sehingga tidak ada anggota kumpulan yang merupakan awalan dari anggota yang lain.
- Mempunyai pohon biner yang bersesuaian dan setiap sisi diberi label 0 atau 1 (harus taat-asas).
- Semua sisi kiri dilabeli 0 saja, sedangkan sisi kanan dilabeli 1 saja, atau sebaliknya.
- Dinyatakan oleh barisan sisi-sisi yang dilalui oleh lintasan dari akar ke daun.
- Kegunaan:

a. untuk mengirim pesan pada komunikasi data.

b. untuk pembentukan kode Huffman dalam pemampatan data (data compression).



Gambar Pohon biner dari kode prefiks { 000, 001, 01, 10, 11}

- Contoh:

Himpunan

{000, 001, 01, 10, 11}

adalah Kode Awalan.

Tetapi, himpunan

{1, 00, 01, 000, 0001}

bukan kode awalan, karena 00 adalah prefiks dari 0001.

Setiap karakter tidak boleh mempunyai kode yang merupakan awalan bagi kode yang lain agar tidak timbul ambigu dalam proses konversi data string.

Kode Huffman

Latar belakang:

- Di dalam komunikasi data, pesan (message) yang dikirim seringkali ukuran nya sangat besar sehingga waktu pengiriman nya lama.
- Arsip (file) yang berukuran besar akan membutuhkan ruang penyimpanan yang besar.

Solusi:

- Mengkodekan pesan atau isi arsip sesingkat mungkin, sehingga waktu yang dibutuhkan sedikit =>

Pemampatan (compression) Data : mengkodekan setiap karakter di dalam pesan atau di dalam arsip, dengan kode yang lebih pendek.

Sistem kode yang banyak digunakan adalah kode ASCII.

Tabel Kode ASCII

Simbol	Kode ASCII
A	01000001
B	01000010
C	01000011
D	01000100

Kode ASCII:
setiap karakter dikodekan dalam 8 bit biner

rangkaian bit untuk string '*ABACCDA*':

01000001010000010010000010100000110100000110100010001000001

atau 7 × 8 = 56 bit (7 *byte*).

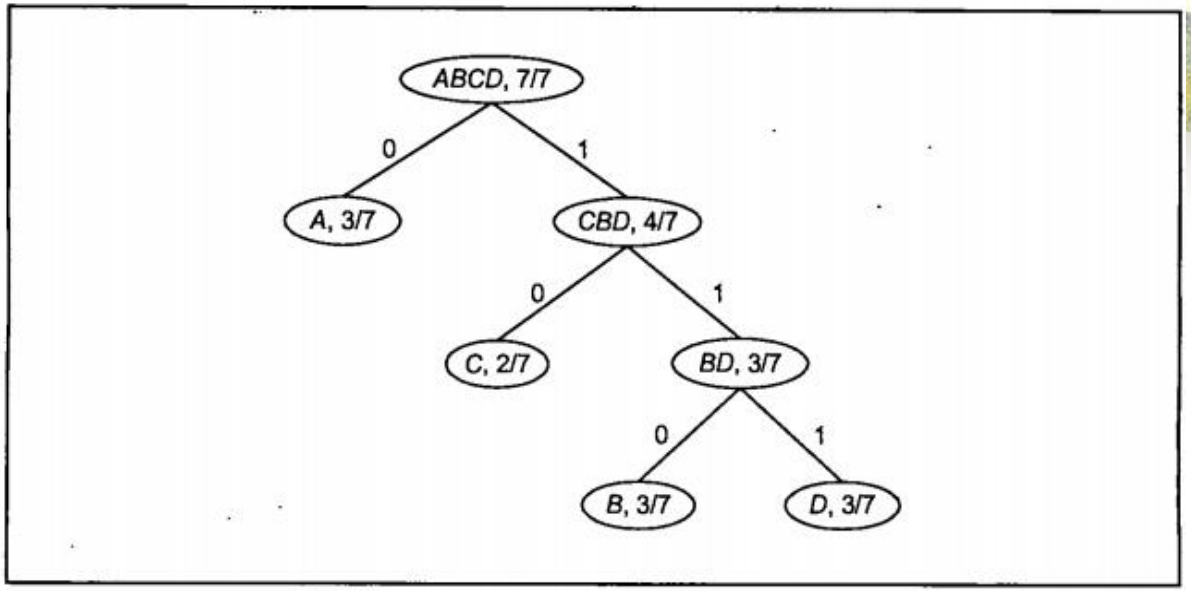
Tabel Tabel kekerapan (frekuensi) dan kode Huffman untuk *string* *ABACCDA*

Simbol	Kekerapan	Peluang	Kode Huffman
A	3	3/7	0
B	1	1/7	110
C	2	2/7	10
D	1	1/7	111

Dengan kode Hufman, rangkaian bit untuk *'ABACCDA'*:

0110010101110

hanya 13 bit!



Gambar 9.28 Pohon Huffman untuk pesan '*ABACCDA*'

Untuk mendapatkan kode Huffman, mula-mula kita harus menghitung dulu kekerapan kemunculan tiap simbol di dalam teks. Cara pembentukan kode Huffman adalah dengan membentuk pohon biner, yang dinamakan pohon Huffman, sebagai berikut:

a) Pilih dua simbol dengan peluang (*probability*) paling kecil (pada contoh di atas simbol *B* dan *D*). Kedua simbol tadi dikombinasikan sebagai simpul orangtua dari simbol *B* dan *D* sehingga menjadi simbol *BD* dengan peluang 1/7 + 1/7 = 2/7, yaitu jumlah peluang kedua anaknya. Simbol baru ini diperlakukan sebagai simpul baru dan diperhitungkan dalam mencari simbol selanjutnya yang memiliki peluang paling kecil.

b) Selanjutnya, pilih dua simbol berikutnya, termasuk simbol baru, yang mempunyai peluang terkecil. Pada contoh ini, dua simbol tersebut adalah *C* (peluang = 2/7) dan *BD* (peluang = 2/7). Lakukan hal yang sama seperti langkah sebelumnya sehingga dihasilkan simbol baru *CBD* dengan kekerapan 2/7 + 2/7 = 4/7.

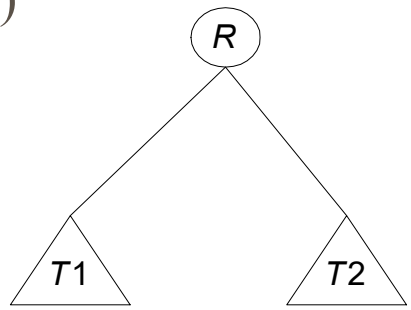
c) Prosedur yang sama dilakukan pada dua simbol berikutnya yang mempunyai peluang terkecil, yaitu *A* (peluang = 3/7) dan *CBD* (peluang = 4/7) sehingga menghasilkan simpul *ACBD*, yang merupakan akar pohon Huffman dengan peluang 3/7 + 4/7 = 7/7.

Pohon pencarian biner (Binary Search Tree - BST)

- Sangat berguna untuk menyelesaian persoalan operasi pencarian, penyisipan, dan penghapusan elemen.
- Simpul pada pohon pencarian adalah field kunci pada data record (unik).
- Merupakan pohon biner yang setiap kuncinya diatur dalam suatu urutan tertentu.
- Jika R adalah akar, dan semua kunci adalah unik, maka:

a. Semua simpul pada upapohon kiri mempunyai kunci lebih kecil dari Kunci (R)

b. Semua simpul di upapohon kanan mempunyai kunci lebih besar dari Kunci (R)

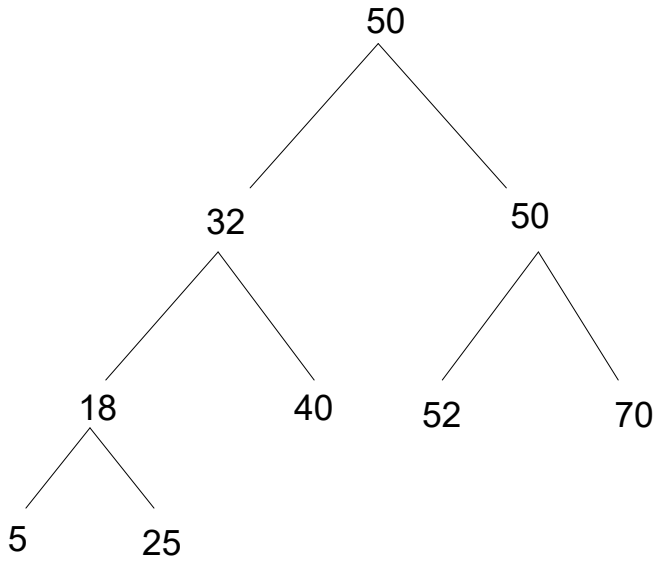


Kunci(*T*1) < Kunci(*R*)

Kunci(*T*2) > Kunci(*R*)

- Contoh: Pohon pencarian untuk data masukan dengan urutan tertentu.

Data: 50, 32, 18, 40, 60, 52, 5, 25, 70



Traversal Pohon Biner

- Operasi dasar yang sering dilakukan pada pohon biner adalah mengunjungi (traversal) setiap simpul tepat satu kali; mencetak informasi yang disimpan di dalam simpul, memanipulasi nilai, dsb.
- Misal akar (R), upapohon kiri (T1), dan upapohon kanan (T2), akan ada tiga skema traversal pohon biner, yaitu:

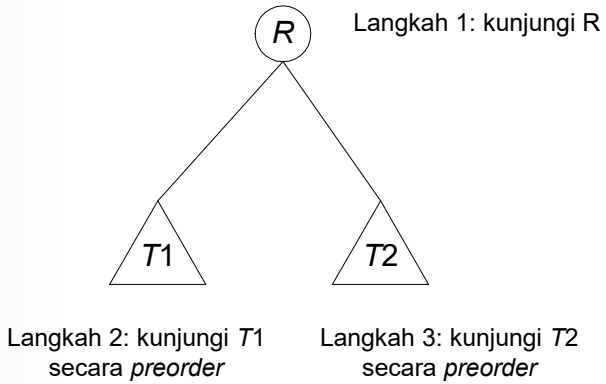
a. Preorder
b. Inorder
c. Postorder

Penelusuran (traversal) Pohon Biner

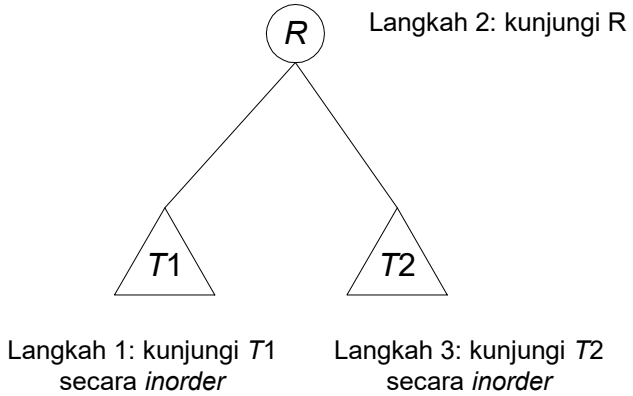
1. *Preorder* : R, T_1, T_2
   - kunjungi R
   - kunjungi T_1 secara *preorder*
   - kunjungi T_2 secara *preorder*

2. *Inorder* : T_1 , R, T_2
   - kunjungi T_1 secara *inorder*
   - kunjungi R
   - kunjungi T_2 secara *inorder*

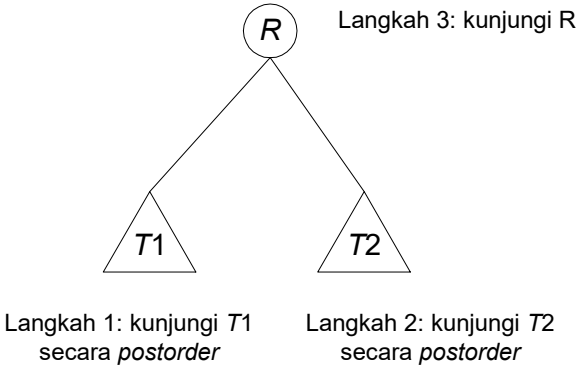
3. *Postorder* : T_1, T_2 , R
   - kunjungi T_1 secara *postorder*
   - kunjungi T_2 secara *postorder*
   - kunjungi R



(a) *preorder*



(b) *inorder*



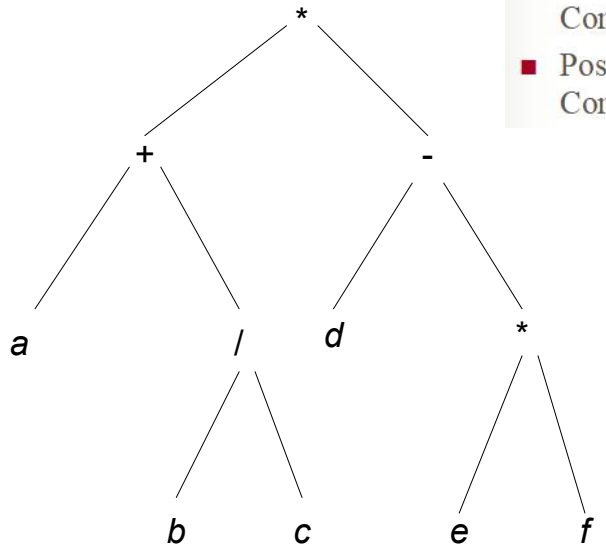
(c) *postorder*

preorder : * + a / b c - d * e f (*prefix*)

inorder : a + b / c * d - e * f (*infix*)

postorder : a b c / + d e f * - * (*postfix*)

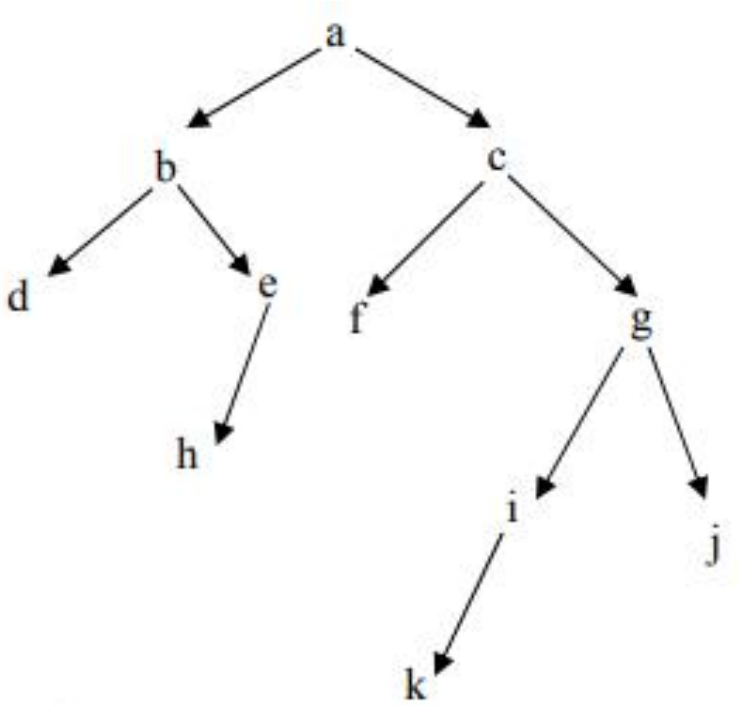
- Prefix : Operator mendahului dua buah operand-nya. Contoh: *+ab/c+de
- Infix : Operator berada diantara dua buah operand. Contoh: (a+b)*(c/(d+e))
- Postfix: Kedua operand mendahului operatornya. Contoh: ab+cde+/*



Tugas 2

1. Buatlah pohon Huffman untuk string ‘abaaccdeba’ dengan ketentuan: Simbol dengan peluang lebih kecil sebagai anak kiri dan simbol dengan peluang lebih besar sebagai anak kanan, sisi kiri dilabeli dengan 0 dan sisi kanan dengan 1. Tuliskan kode Huffman untuk setiap simbol pembentuk string, selanjutnya tuliskan rangkaian bit yang merepresentasikan string tersebut dengan kode Huffman.

2. Telusuri pohon biner berikut secara preorder, inorder, postorder (tuliskan tahapan penelusurannya)!



TERIMAKASIH

EXAMPLE 1 Form a binary search tree for the words *mathematics, physics, geography, zoology, meteorology, geology, psychology*, and *chemistry* (using alphabetical order).

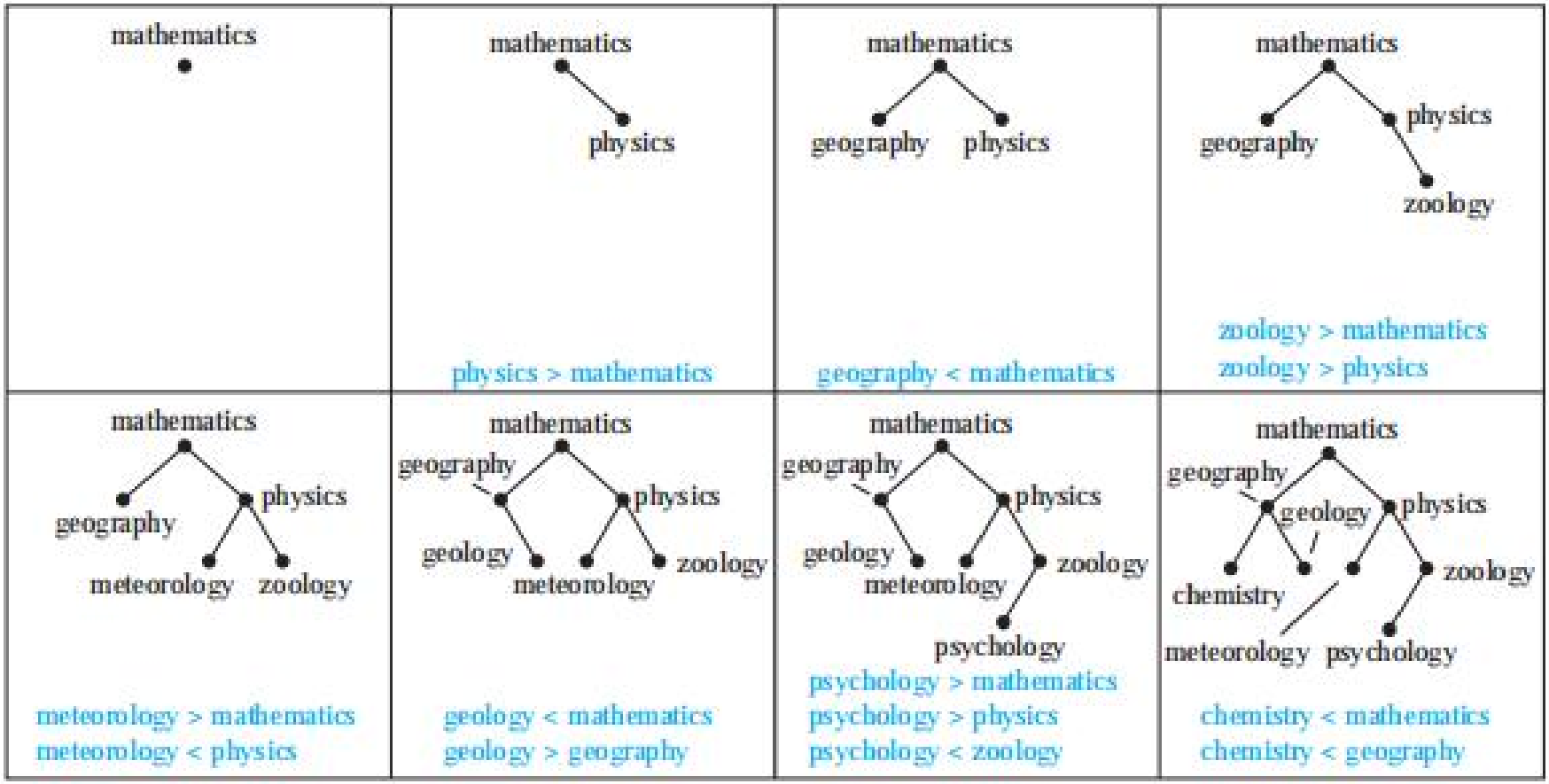


FIGURE 1 **Constructing a Binary Search Tree.**