BAB 9

Pohon

Jika berencana untuk satu tahun, tanamlah padi
Jika berencana untuk sepuluh tahun, tanamlah pohon
Namun jika berencana untuk seratus tahun, didiklah generasi penerus
(Confusius)

Graf terhubung yang tidak mengandung sirkuit disebut pohon. Di antara sekian banyak konsep dalam teori graf, konsep pohon (*tree*) mungkin merupakan konsep yang paling penting, karena terapannya yang luas dalam berbagai bidang ilmu. Banyak terapan, baik dalam bidang ilmu komputer maupun di luar bidang ilmu komputer, yang telah mengkaji pohon secara intensif sebagai objek matematika. Dalam kehidupan sehari-hari, orang telah lama menggunakan pohon untuk menggambarkan hirarkhi. Misalnya, pohon silsilah keluarga, struktur organisasi, organisasi pertandingan, dan lain-lain. Para ahli bahasa menggunakan pohon untuk menguraikan kalimat, yang disebut pohon parsing (*parse tree*).

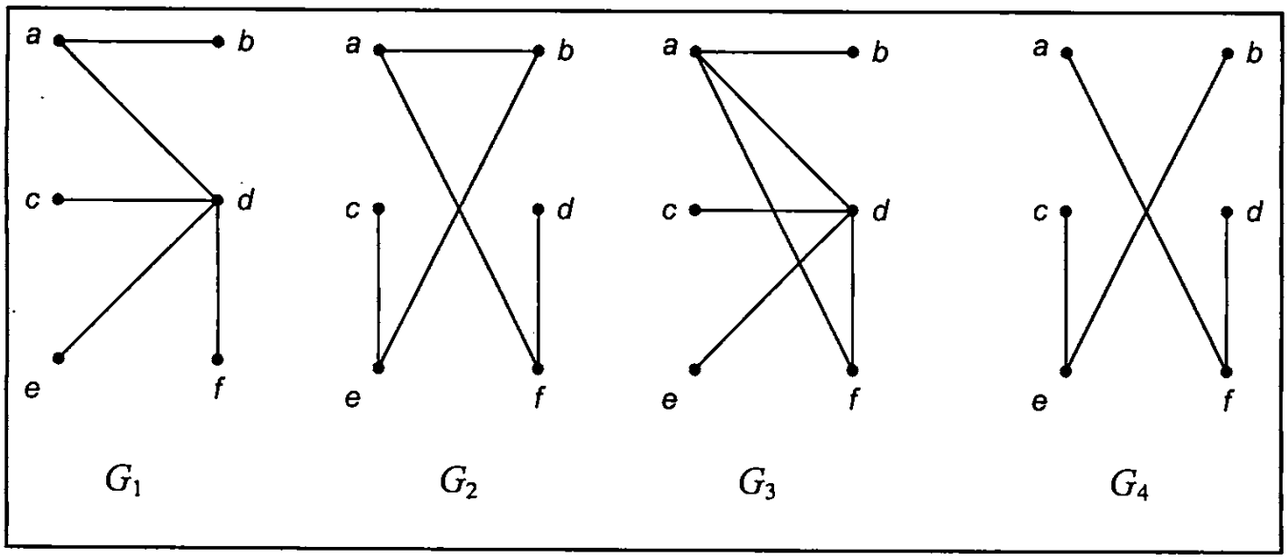
Pohon sudah lama digunakan sejak tahun 1857, ketika matematikawan Inggris Arthur Cayley menggunakan pohon untuk menghitung jumlah senyawa kimia. Bab ini membahas pohon dari sudut pandang teori graf. Pohon sebagai struktur data rekursif merupakan bagian dari perkuliahan Struktur Data.

9.1 Definisi Pohon

Pohon adalah graf yang khusus. Definisi pohon adalah sebagai berikut:

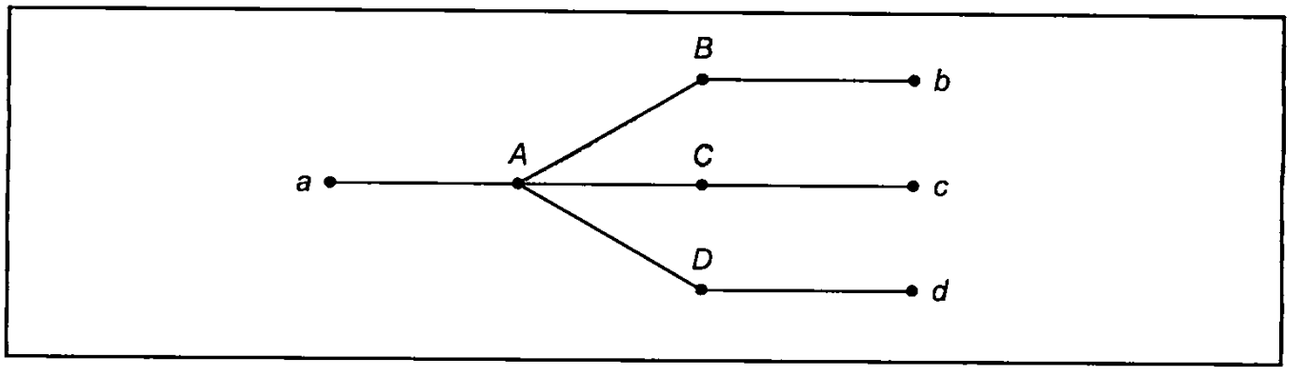
DEFINISI 9.1. Pohon adalah graf tak-berarah terhubung yang tidak mengandung sirkuit.

Menurut definisi 9.1 di atas, ada dua sifat penting pada pohon: terhubung dan tidak mengandung sirkuit. Pada Gambar 9.1, hanya G_1 dan G_2 yang pohon, sedangkan G_3 dan G_4 bukan pohon. G_3 bukan pohon karena ia mengandung sirkuit a, d, f, a, sedangkan G_4 bukan pohon karena ia tidak terhubung (anda jangan tertipu dengan persilangan dua buah sisi –dalam hal ini sisi (a, f) dan sisi (b, e)– karena titik silangnya bukan menyatakan simpul).



Gambar 9.1 G_1 dan G_2 adalah pohon, sedangkan G_3 dan G_4 bukan pohon

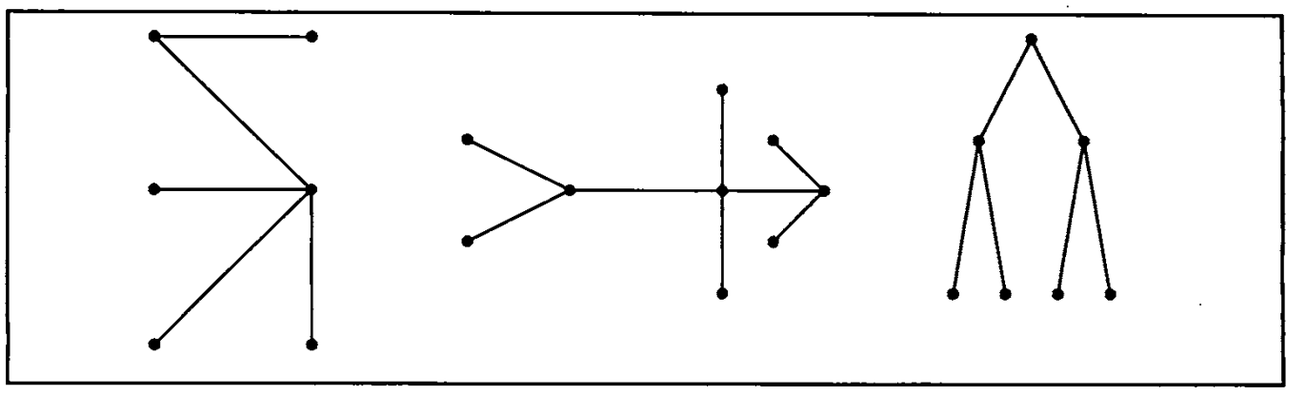
Berikut contoh pohon lainnya, diambil dari [LIU85], misalkan himpunan $V = \{a, A, b, B, c, C, d, D\}$ adalah empat pasangan suami-istri tukang gosip, dengan a, b, c, dan d para suami, dan A, B, C, D para istri. Misalkan a menceritakan gosip lewat telpon kepada istrinya A, yang kemudian A menelepon para istri lainnya untuk menyebarkan gosip itu, dan setiap istri itu menelpon dan menceritakan gosip kepada suami masing-masing. Pohon pada Gambar 9.2 menunjukkan bagaimana gosip tersebut tersebar, dengan simpul menyatakan istri/suami dan sisi menyatakan panggilan telpon.



Gambar 9.2 Pohon penyebaran gosip

Karena definisi pohon diacu dari teori graf, maka sebuah pohon dapat mempunyai hanya sebuah simpul tanpa sebuah sisipun. Dengan kata lain, jika $G = (V, E)$ adalah pohon, maka V tidak boleh berupa himpunan kosong, namun E boleh kosong. Pada sebagian literatur, pohon yang dimaksudkan oleh Definisi 9.1 sering disebut juga **pohon bebas** (*free tree*) untuk membedakannya dengan **pohon berakar** (*rooted tree*). Pohon berakar akan kita bahas lebih lanjut di dalam upabab 9.4. Untuk selanjutnya, jika disebut pohon, maka yang dimaksudkan adalah pohon bebas.

Beberapa pohon dapat membentuk hutan. **Hutan** (*forest*) adalah kumpulan pohon yang saling lepas. Kita dapat juga menyatakan bahwa hutan adalah graf tak-terhubung yang tidak mengandung sirkuit, yang dalam hal ini setiap komponen di dalam graf terhubung tersebut adalah pohon. Gambar 9.3 berikut adalah hutan yang terdiri atas 3 buah pohon.



Gambar 9.3 Hutan yang terdiri dari tiga buah pohon

Pohon juga seringkali didefinisikan sebagai graf tak-berarah dengan sifat bahwa hanya terdapat sebuah lintasan unik antara setiap pasang simpul. Tinjau kembali graf G_1 pada Gambar 9.1. Setiap simpul di G_1 terhubung dengan lintasan tunggal. Sebagai contoh, dari b ke f hanya ada satu lintasan, yaitu b, a, d, f. Dari c ke a hanya ada satu lintasan, yaitu c, d, a. Demikian juga untuk setiap pasang simpul manapun di G_1.

Selain itu, kita juga melihat bahwa di dalam pohon, jumlah sisinya adalah jumlah simpul dikurangi satu. Tinjau G_1 pada Gambar 9.1, jumlah simpul = 6 dan jumlah sisi = 5 (yaitu 6 – 1). Sifat-sifat pohon lain yang dirangkum di dalam Teorema 9.1 pada upabab 9.2 di bawah ini.

9.2 Sifat-sifat Pohon

Sifat-sifat (*properties*) pohon dinyatakan dengan Teorema 9.1 di bawah ini.

TEOREMA 9.1. Misalkan $G = (V, E)$ adalah graf tak-berarah sederhana dan jumlah simpulnya n. Maka, semua pernyataan di bawah ini adalah ekivalen:

1. G adalah pohon.
2. Setiap pasang simpul di dalam G terhubung dengan lintasan tunggal.
3. G terhubung dan memiliki $m = n - 1$ buah sisi.
4. G tidak mengandung sirkuit dan memiliki $m = n - 1$ buah sisi.
5. G tidak mengandung sirkuit dan penambahan satu sisi pada graf akan membuat hanya satu sirkuit.
6. G terhubung dan semua sisinya adalah jembatan (jembatan adalah sisi yang bila dihapus menyebabkan graf terpecah menjadi dua komponen).

Semua butir pernyataan di atas juga dapat dianggap sebagai definisi lain dari pohon. Kita juga dapat membuktikan bahwa hutan F dengan k komponen mempunyai $m = n - k$ buah sisi.

Contoh 9.1

Sebuah pohon mempunyai $2n$ buah simpul berderajat 1, $3n$ buah simpul berderajat 2, dan n buah simpul berderajat 3. Tentukan banyaknya simpul dan sisi di dalam pohon itu.

Penyelesaian:
Menurut Lemma Jabat tangan, jumlah derajat semua simpul di dalam graf adalah 2 kali jumlah sisi di dalam graf tersebut. Jadi,

$$(2n \times 1) + (3n \times 2) + (n \times 3) = 2|E| \Leftrightarrow 11n = 2|E|$$

Menurut Teorema 9.1, jumlah sisi pada sebuah pohon adalah jumlah simpul minus satu, jadi

$$|E| = (2n + 3n + n) - 1 = 6n - 1$$

Dengan menyulihkan persamaan terakhir ke persamaan pertama,

$$11n = 2(6n - 1) = 12n - 2 \Leftrightarrow n = 2$$

Jadi, jumlah simpul pada pohon = $6n = 6 \times 2 = 12$ dan jumlah sisi = $6n - 1 = 11$. ■

9.3 Pewarnaan Pohon

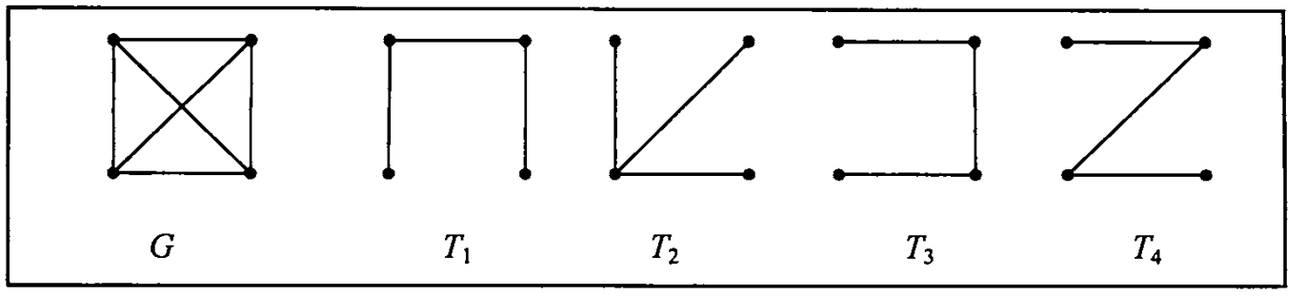
Ditinjau dari teori pewarnaan graf, maka pohon mempunyai bilangan kromatik 2. Dengan kata lain, dua buah warna sudah cukup mewarnai simpul-simpul di pohon sedemikian sehingga tidak ada dua buah simpul bertetangga mempunyai warna sama.

Pewarnaan pada pohon T dilakukan dengan cara berikut: petakan warna pertama pada sembarang sebuah simpul. Kemudian, petakan warna kedua pada simpul-simpul yang bertetangga dengan simpul pertama tadi. Selanjutnya, petakan warna pertama ke semua simpul yang bertetangga dengan simpul-simpul yang telah diberi warna kedua. Ulangi proses ini sampai semua simpul telah diwarnai. Sebagai contoh, tinjau G_1 pada Gambar 9.1. Simpul-simpul pada G_1 akan diwarnai dengan warna kuning dan biru. Simpul a dipilih pertama kali untuk diberi warna kuning. Kemudian simpul-simpul tetangga a, yaitu b dan d, diberi warna biru. Selanjutnya simpul-simpul yang beretangga dengan d, yaitu c, e, dan f, diberi warna kuning.

Kuning: a, c, e, f Biru: b, d

9.4 Pohon Merentang

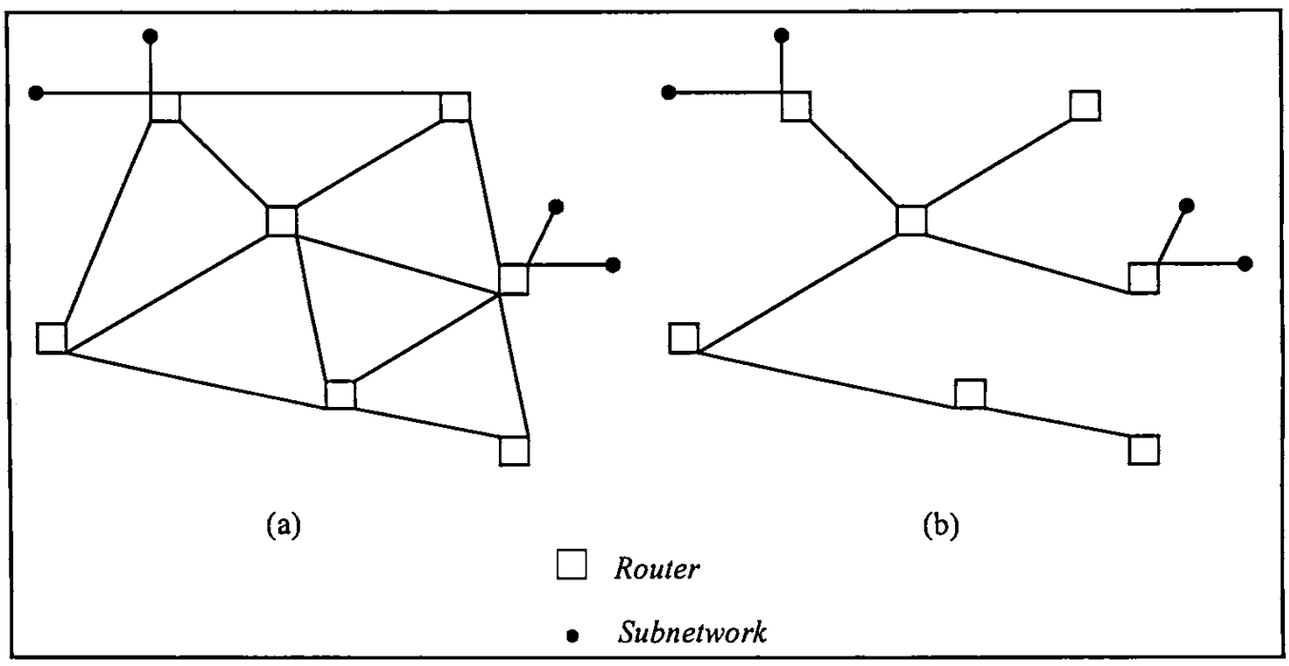
Misalkan $G = (V, E)$ adalah graf tak-berarah terhubung yang bukan pohon, yang berarti di G terdapat beberapa sirkuit. G dapat diubah menjadi pohon $T = (V_1, E_1)$ dengan cara memutuskan sirkuit-sirkuit yang ada. Caranya, mula-mula dipilih sebuah sirkuit, lalu hapus satu buah sisi dari sirkuit ini. G akan tetap terhubung dan jumlah sirkuitnya berkurang satu. Bila proses ini dilakukan berulang-ulang sampai semua sirkuit di G hilang, maka G menjadi sebuah pohon T, yang dinamakan **pohon merentang** (*spanning tree*). Disebut pohon merentang karena semua simpul pada pohon T sama dengan semua simpul pada graf G, dan sisi-sisi pada pohon $T \subseteq$ sisi-sisi pada graf G. Dengan kata lain, $V_1 = V$ dan $E_1 \subseteq E$ (Bandingkan dengan definisi upagraf rentang di Bab 8. Dengan kata lain, jika upagraf dari graf terhubung berbentuk pohon, maka upagraf merentang tersebut dinamakan pohon merentang). Gambar 9.4 adalah graf lengkap dengan empat buah pohon merentangnya (coba anda temukan seluruh pohon merentang lainnya).



Gambar 9.4 Graf lengkap G dan empat buah pohon merentangnya, T_1, T_2, T_3, dan T_4

Aplikasi pohon merentang misalnya pada pemeliharaan jalan raya. Misalkan graf G pada Gambar 9.4 adalah peta jaringan jalan raya yang menghubungkan empat buah kota. Karena dana pemeliharaan yang terbatas, pemerintah daerah mempertimbangkan hanya memelihara jalan-jalan sesedikit mungkin sedemikian sehingga keempat kota masih tetap terhubung satu sama lain. Masalah ini dapat dipecahkan dengan membuat upagraf yang mengandung jumlah sisi minimum dan mengandung semua simpul di dalam graf. Graf semacam itu haruslah pohon merentang.

Pohon merentang juga memainkan peranan penting dalam jaringan komputer. Jaringan komputer dapat dimodelkan sebagai sebuah graf. Simpul pada graf dapat menyatakan suatu terminal komputer (*work station*) atau suatu *router* (*router* adalah komputer yang difungsikan untuk meneruskan data dari suatu simpul komunikasi ke simpul komunikasi lain). Jika sebuah komputer mengirim pesan (atau data) ke komputer lain (melalui *router*), maka komputer tersebut mengirimkannya ke seluruh simpul-simpul di jaringan. Setiap pesan yang sampai ke suatu *router* antara akan diteruskan ke satu atau lebih *router* lainnya. Dengan cara seperti ini, maka pesan akan sampai ke komputer penerima. Pesan yang telah sampai ke suatu router diharapkan tidak pernah kembali diterima oleh router tersebut. Tetapi karena *router-router* pada jaringan umumnya membentuk sirkuit (atau *cycle* atau *loop*), maka menerima pesan yang sama lebih dari sekali pasti terjadi. Untuk mengatasi hal ini, maka algoritma jaringan membentuk pohon merentang di dalam graf sehingga antara sepasang simpul *router* hanya ada satu lintasan tunggal dan simpul-simpul router tidak pernah menerima pesan yang sama lebih dari sekali. Metode penyebaran pesan (*routing*) seperti ini dinamakan *IP Multicasting*. Gambar 9.5 memperlihatkan contoh sebuah jaringan komputer dan *router-router* yang membentuk pohon merentang.



Gambar 9.5 (a) Jaringan komputer, (b) Pohon merentang *multicast*

Harus diingat bahwa pohon merentang didefinisikan hanya untuk graf terhubung, karena pohon selalu terhubung. Pada graf tak-terhubung dengan n buah simpul kita tidak dapat menemukan upagraf terhubung dengan n buah simpul. Tiap komponen dari graf tak-terhubung mempunyai satu buah pohon merentang. Dengan demikian, graf tak-terhubung dengan k komponen mempunyai **hutan merentang** (*spanning forest*) yang terdiri dari k buah pohon merentang.

Pada graf terhubung terdapat setidaknya satu buah pohon merentang. Sifat ini dinyatakan dengan Teorema 9.2 berikut ini.

TEOREMA 9.2. Setiap graf terhubung mempunyai paling sedikit satu buah pohon merentang.

Teorema 9.2 menyatakan bahwa graf yang tidak mengandung sirkuit adalah pohon merentang itu sendiri. Pada graf yang mengandung sirkuit, pohon merentangnya diperoleh dengan cara memutuskan sirkuit yang ada.

Sisi pada pohon merentang – disebut **cabang** (*branch*) – adalah sisi dari graf semula, sedangkan **tali-hubung** (*chord* atau *link*) dari pohon adalah sisi dari graf yang tidak terdapat di dalam pohon merentang. Pada graf terhubung dengan m buah sisi dan n buah simpul terdapat $n - 1$ buah cabang dan $m - n + 1$ buah tali. Himpunan tali-hubung beserta simpul yang bersisian dengannya disebut **komplemen pohon**. Untuk graf terhubung G dengan n buah simpul dan m buah sisi, kita dapat menghitung jumlah cabang dan tali-hubung dengan rumus

$$\text{jumlah cabang} = n - 1$$
$$\text{jumlah tali-hubung} = m - n + 1$$

dan pada graf tidak terhubung dengan k komponen, m buah sisi dan n buah simpul,

$$\text{jumlah cabang} = n - k$$
$$\text{jumlah tali-hubung} = m - n + k$$

Jumlah cabang pada pohon merentang dari sebuah graf G disebut *rank* graf G, dan jumlah tali-hubung pada graf G disebut *nullity* graf G. Dapat dilihat bahwa

$$rank + nullity = \text{jumlah sisi graf } G$$

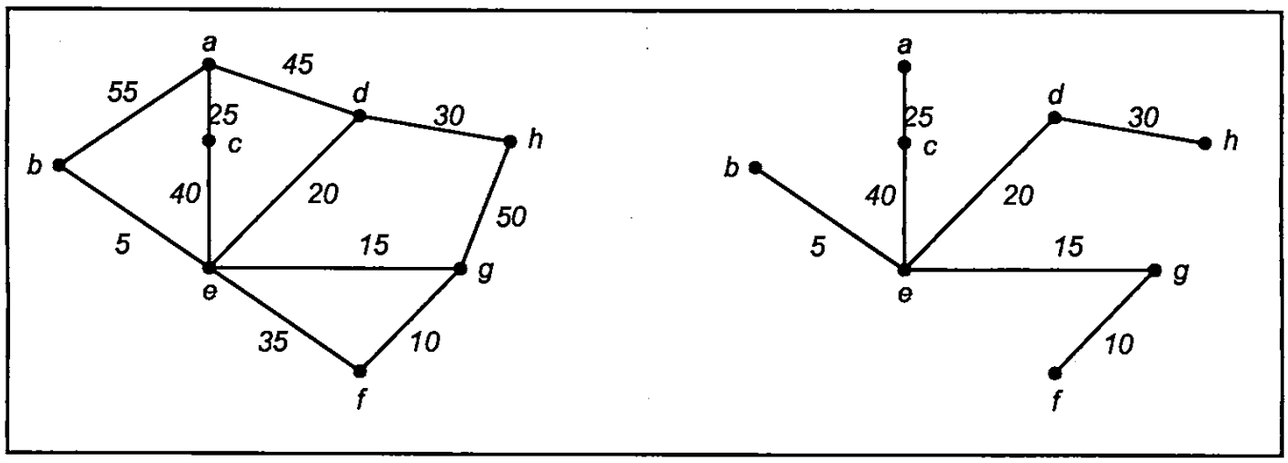
Nullity graf sering diacu sebagai **bilangan siklomatik**, atau **bilangan Betti pertama** [DEO74].

Ingatlah kembali bahwa jika kita menambahkan sebuah sisi antara dua buah simpul pada pohon maka akan terbentuk sirkuit. Tinjau kembali pohon merentang T pada graf terhubung G. Dengan menambahkan sebuah tali-hubung pada T akan terbentuk sirkuit. Sirkuit yang terbentuk dengan penambahan sebuah tali-hubung

pada pohon merentang disebut **sirkuit fundamental** (*fundamental circuit*). Berapa jumlah sirkuit fundamental pada sebuah graf? Tentu saja sebanyak jumlah tali-hubung.

Pohon Merentang Minimum

Jika G adalah graf berbobot, maka maka bobot pohon merentang T dari G didefinisikan sebagai jumlah bobot semua sisi di T. Pohon merentang yang berbeda mempunyai bobot yang berbeda pula. Di antara semua pohon merentang di G, pohon merentang yang berbobot minimum – dinamakan **pohon merentang minimum** (*minimum spanning tree*) – merupakan pohon merentang yang paling penting. Pohon merentang minimum mempunyai terapan yang luas dalam praktek. Misalkan Pemerintah akan membangun jalur rel kereta api yang menghubungkan sejumlah kota seperti yang digambarkan oleh graf pada Gambar 9.6. Membangun jalur rel kereta api biayanya mahal, karena itu pembangunan jalur ini tidak perlu menghubungkan langsung dua buah kota; tetapi cukup membangun jalur kereta seperti pohon merentang. Karena di dalam sebuah graf mungkin saja terdapat lebih dari satu pohon merentang, harus dicari pohon merentang yang mempunyai jumlah jarak terpendek, dengon kata lain harus dicari pohon merentang minimum.



Gambar 9.6 (a) Graf yang menyatakan jaringan jalur rel kereta api. Bobot pada tiap sisi menyatakan panjang rel kereta api (× 100 km)

(b) Pohon merentang yang mempunyai jumlah jarak minimum

Terdapat dua buah algoritma membangun pohon merentang minimum. Yang pertama adalah algoritma Prim, dan yang kedua adalah algoritma Kruskal

Algoritma Prim

Misalkan T adalah pohon merentang yang sisi-sisinya diambil dari graf G. Algoritma Prim membentuk pohon merentang minimum langkah per langkah. Pada setiap langkah kita mengambil sisi e dari graf G yang mempunyai bobot

minimum dan bersisian dengan simpul-simpul di dalam T tetapi e tidak membentuk sirkuit di dalam T.

ALGORITMA PRIM

1. Ambil sisi dari graf G yang berbobot minimum, masukkan ke dalam T.
2. Pilih sisi e yang mempunyai bobot minimum dan bersisian dengan simpul di T, tetapi e tidak membentuk sirkuit di T. Masukkan e ke dalam T.
3. Ulangi 2 sebanyak $n - 2$ kali.

Jumlah langkah seluruhnya di dalam algoritma Prim adalah $1 + (n - 2) = n - 1$, yaitu sebanyak jumlah sisi di dalam pohon merentang dengan n buah simpul.

Dalam notasi *pseudo-code*, algoritma Prim kita tuliskan sebagai berikut:

```
procedure Prim(input G : graf, output T : pohon)
{ Membentuk pohon merentang minimum T dari graf terhubung G.
  Masukan: graf-berbobot terhubung G = (V, E), yang mana |V|= n
  Keluaran: pohon merentang minimum T = (V, E')
}

Deklarasi
  e : sisi

Algoritma
  T ← sisi e yang mempunyai bobot minimum di dalam E
  E ← E - {e}  { e sudah dipilih, jadi buang e dari E }
  for i ← 1 to n - 2 do
    e ← sisi yang mempunyai bobot terkecil di dalam E dan bersisian
        dengan simpul di T
    T ← T ∪ {e}  { masukkan e ke dalam T yang sudah terbentuk }
    E ← E - {e}  { e sudah dipilih, jadi buang e dari E }
  endfor
```

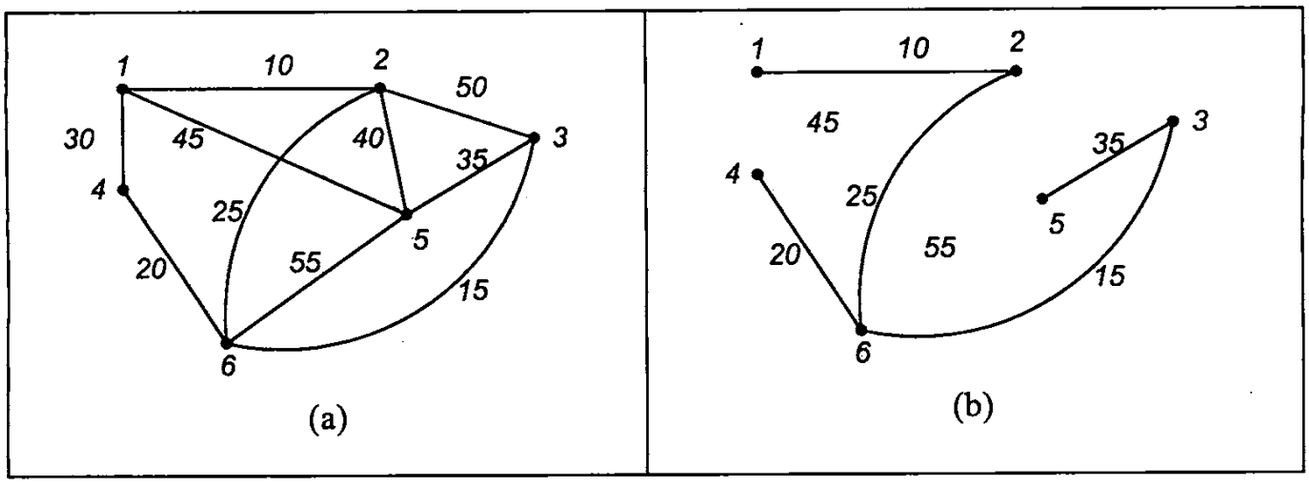
Algoritma 9.1 Algoritma Prim untuk membentuk pohon merentang minimum

Contoh 9.2

Kita akan mencari pohon merentang minimum pada graf yang ditunjukkan pada Gambar 9.7 dengan algoritma Prim.

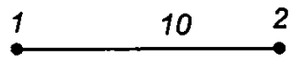
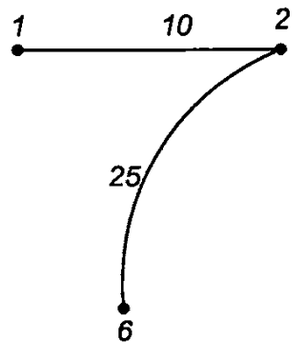
Penyelesaian:
Langkah-langkah pembentukan pohon merentang minimum diperlihatkan pada Tabel 9.1. Pohon merentang minimum dari graf pada Gambar 9.7(a) adalah seperti yang ditunjukkan oleh Gambar 9.7(b). Bobot pohon merentang minimum ini adalah

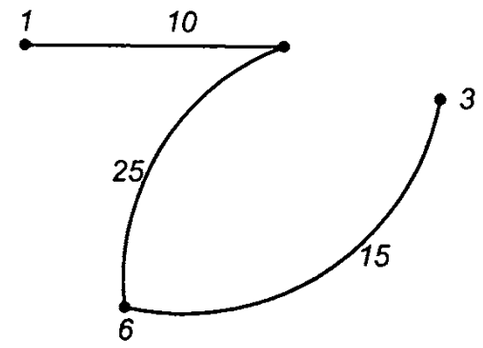
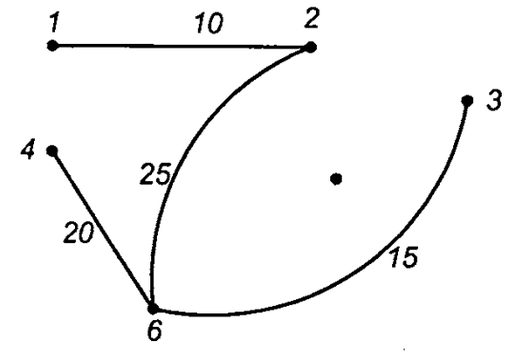
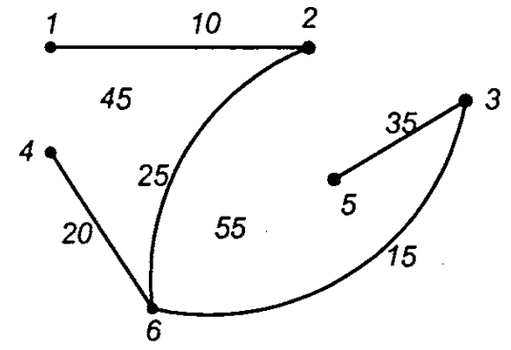
$$10 + 25 + 15 + 20 + 35 = 105$$ ■



Gambar 9.7 (a) Contoh graf untuk algoritma Prim dan Kruskal
(b) Pohon merentang minimum dari graf pada Gambar

Tabel 9.1 Tabel pembentukan pohon merentang minimum dengan algoritma Prim

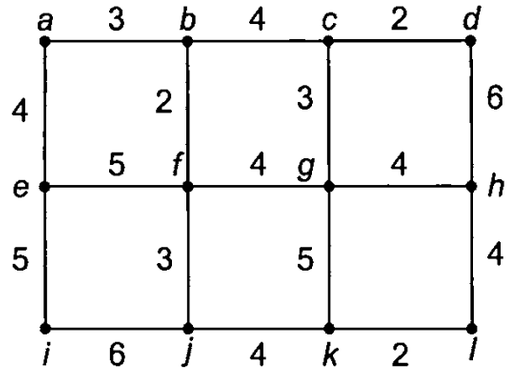
Langkah	Sisi	Bobot	Pohon merentang
1	(1, 2)	10	
2	(2, 6)	25	

3	(3, 6)	15	
4	(4, 6)	20	
5	(3, 5)	35	

Perhatikanlah bahwa algoritma Prim tidak menentukan sisi mana yang dipilih jika terdapat lebih dari satu buah sisi yang berbobot sama. Satu cara untuk mengatasi hal ini adalah dengan mengurutkan sisi-sisi itu berdasarkan bobotnya dari kecil ke besar. Lagi pula, pohon merentang minimum yang dihasilkan tidak selalu unik. Graf sederhana terhubung dan berbobot dapat memiliki lebih dari satu buah pohon merentang minimum yang berbeda tetapi jumlah bobot minimumnya tetap sama. Hal ini ditunjukkan pada Contoh 9.3 berikut.

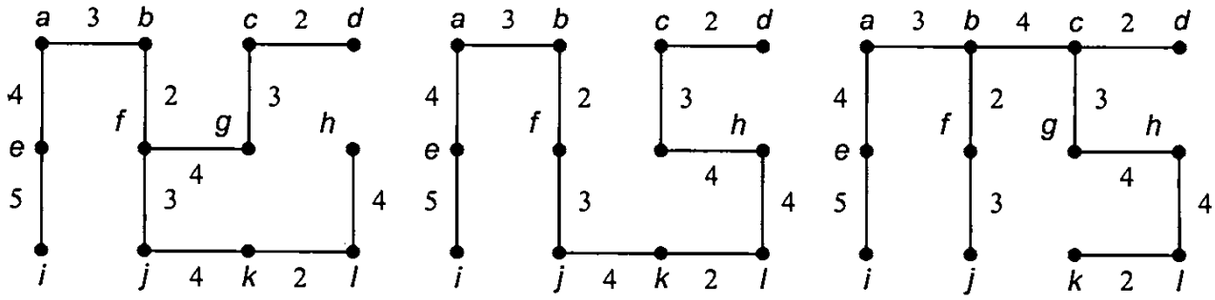
Contoh 9.3

Gambarkan 3 buah pohon merentang minimum yang berbeda beserta bobotnya untuk graf pada Gambar 9.8 dengan menggunakan algoritma Prim.



Gambar 9.8 Graf untuk persoalan pada Contoh 9.3

Penyelesaian:
Graf di atas memiliki beberapa sisi yang berbobot sama, maka ada kemungkinan pohon merentang minimumnya lebih dari satu. Tiga buah di antaranya di adalah seperti di bawah ini:



Ketiga buah pohon merentang minimum di atas sama hanya bentuknya yang berbeda, namun jumlah bobot seluruh sisinyanya tetap sama, yaitu 36. ■

Algoritma Kruskal

Pada algoritma Kruskal, sisi-sisi di dalam graf diurut terlebih dahulu berdasarkan bobotnya dari kecil ke besar. Sisi yang dimasukkan ke dalam himpunan T adalah sisi graf G sedemikian sehingga T adalah pohon. Pada keadaan awal, sisi-sisi sudah diurut berdasarkan bobot membentuk hutan (*forest*), masing-masing pohon di hutan hanya berupa satu buah simpul. Hutan tersebut dinamakan **hutan merentang** (*spanning forest*). Sisi dari graf G ditambahkan ke T jika ia tidak membentuk siklus di T.

Perbedaan prinsip antara algoritma Prim dan Kruskal adalah: jika pada algoritma Prim sisi yang dimasukkan ke dalam T harus bersisian dengan sebuah simpul di T, maka pada algoritma Kruskal sisi yang dipilih tidak perlu bersisian dengan sebuah simpul di T asalkan penambahan sisi tersebut tidak membentuk sirkuit (siklus).

ALGORITMA KRUSKAL
(Asumsi: sisi-sisi dari graf sudah diurut menaik berdasarkan bobotnya)
1. T masih kosong
2. Pilih sisi e dengan bobot minimum yang tidak membentuk sirkuit di T. Masukkan e ke dalam T.
3. Ulangi langkah 2 sebanyak $n - 1$ kali.

Dalam notasi *pseudo-code*, algoritma Prim kita tuliskan sebagai berikut:

```
procedure Kruskal(input G : graf, output T : pohon)
{ Membentuk pohon merentang minimum T dari graf terhubung G.
  Masukan: graf-berbobot terhubung G = (V, E), yang mana |V|= n
  Keluaran: pohon rentang minimum T = (V, E')
}

Deklarasi
  i, p, q, u, v : integer

Algoritma
  ( Asumsi: sisi-sisi dari graf sudah diurut menaik berdasarkan bobotnya)
  T ← {}
  while jumlah sisi T < n - 1 do
    e ← sisi di dalam E yang bobotnya terkecil
    E ← E - {e}        { e sudah dipilih, jadi buang e dari E }
    if e tidak membentuk siklus di T then
       T ← T ∪ {e}     { masukkan e ke dalam T yang sudah terbentuk }
    endif
  endwhile
```

Algoritma 9.2 Algoritma Kruskal untuk membentuk pohon merentang minimum

---

Contoh 9.4

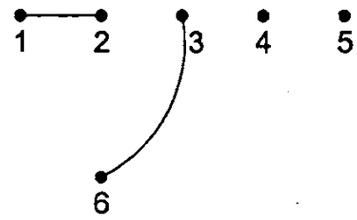
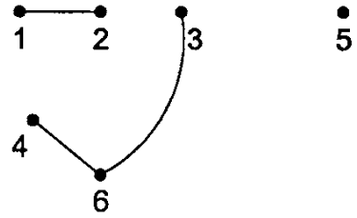
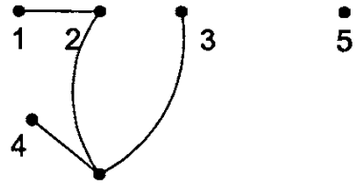
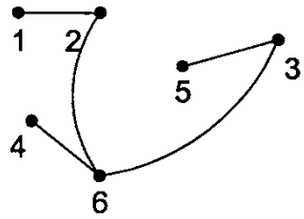
Tinjau kembali graf pada Gambar 9.7(a) di atas. Carilah pohon merentang minimumnya dengan algoritam Kruskal.

Penyelesaian:
Sisi-sisi graf diurut menaik menaik berdasarkan bobotnya:

Sisi	(1,2)	(3,6)	(4,6)	(2,6)	(1,4)	(3,5)	(2,5)	(1,5)	(2,3)	(5,6)
Bobot	10	15	20	25	30	35	40	45	50	55

Langkah-langkah pembentukan pohon merentang minimum diperlihatkan pada Tabel 9.2. Pohon merentang minimum dari graf pada Gambar 9.7(a) adalah seperti yang ditunjukkan oleh Gambar 9.7(b). Bobot pohon merentang minimum ini adalah 10 + 25 + 15 + 20 + 35 = 105. ■

Tabel 9.2 Tabel pembentukan pohon merentang minimum dengan algoritma Kuskal

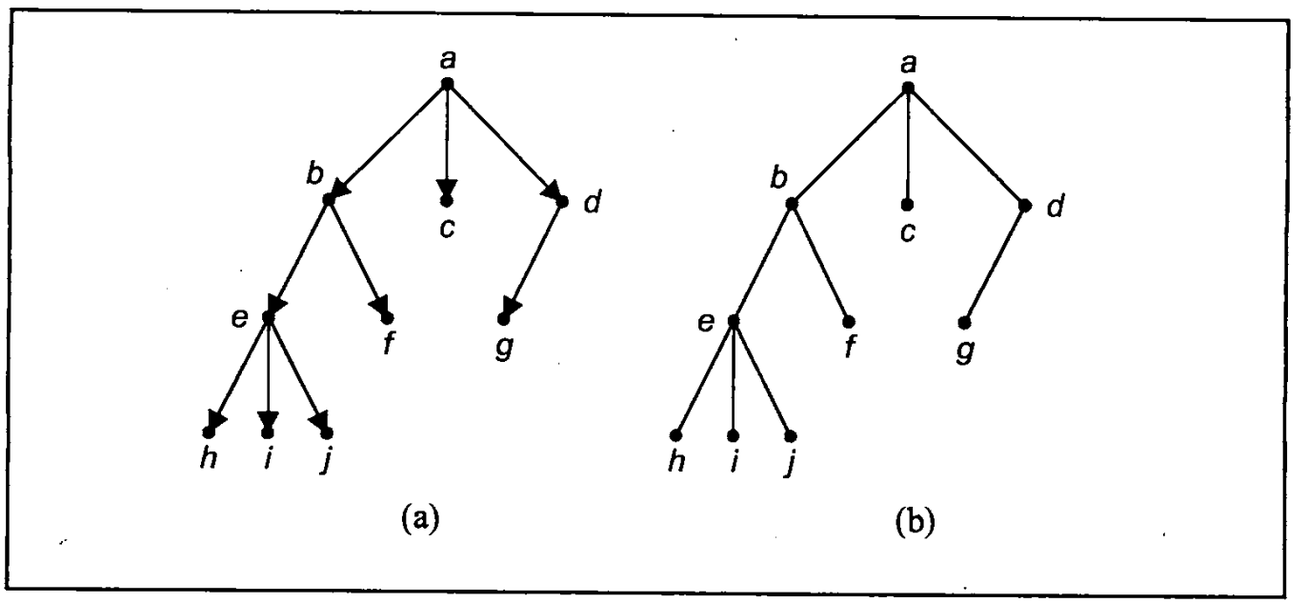
Langkah	Sisi	Bobot	Hutan merentang
0			
1	(1, 2)	10	
2	(3, 6)	15	
3	(4, 6)	20	
4	(2, 6)	25	
5	(1, 4)	30	ditolak
6	(3, 5)	35	

9.5 Pohon Berakar

Pada kebanyakan aplikasi pohon, simpul tertentu diperlakukan sebagai akar (*root*). Sekali sebuah simpul ditetapkan sebagai akar, maka simpul-simpul lainnya dapat dicapai dari akar dengan memberi arah pada sisi-sisi pohon yang mengikutinya.

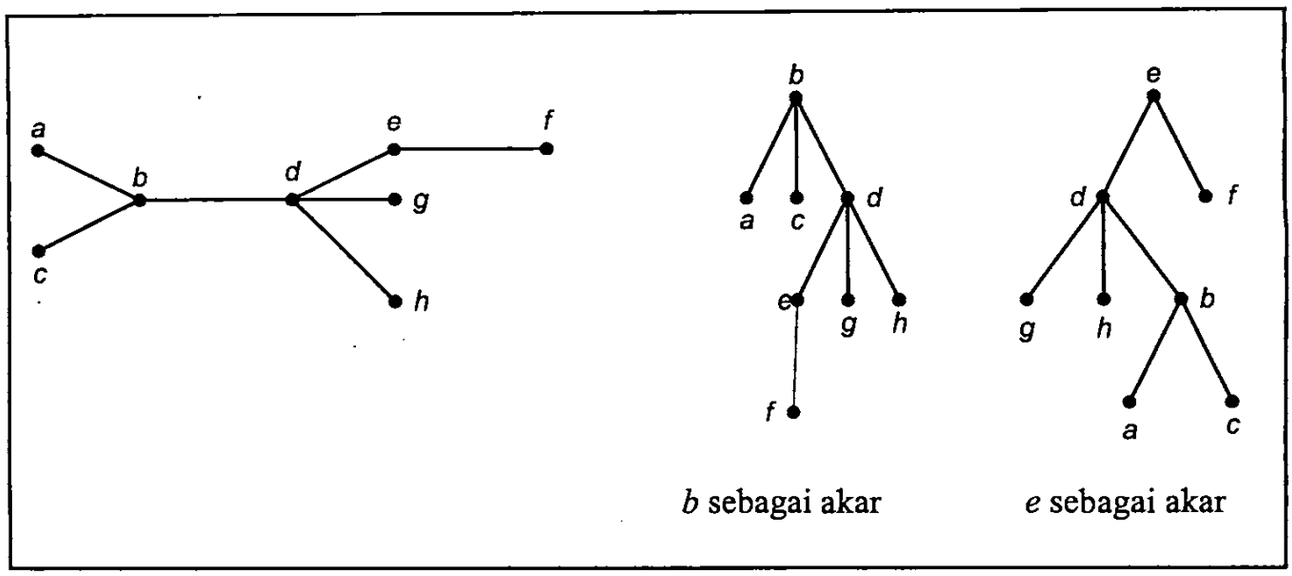
DEFINISI 9.2. Pohon yang sebuah simpulnya diperlakukan sebagai akar dan sisi-sisinya diberi arah menjauh dari akar dinamakan **pohon berakar** (*rooted tree*).

Akar mempunyai derajat-masuk sama dengan nol dan simpul-simpul lainnya berderajat-masuk sama dengan satu. Simpul yang mempunyai derajat-keluar sama dengan nol disebut **daun** atau simpul terminal. Simpul yang mempunyai derajat-keluar tidak sama dengan nol disebut simpul dalam atau simpul cabang. Setiap simpul di pohon dapat dicapai dari akar dengan sebuah lintasan tunggal (unik). Gambar 9.9(a) adalah contoh pohon berakar dengan *a* adalah simpul akarnya. Sebagai konvensi, arah sisi di dalam pohon tidak perlu digambar, karena setiap simpul di pohon harus dicapai dari akar, maka lintasan di dalam pohon berakar selalu dari "atas" ke "bawah". Gambar 9.9(b) menunjukkan hal ini.



Gambar 9.9 (a) Pohon berakar, (b) sebagai konvensi, arah panah pada sisi dapat dibuang

Sembarang pohon tak-berakar dapat diubah menjadi pohon berakar dengan memilih sebuah simpul sebagai akar. Pemilihan simpul yang berbeda menjadi akar menghasilkan pohon berakar yang berbeda pula. Gambar 9.10 memperlihatkan dua pohon akar yang berbeda dari pengubahan sebuah pohon tak-berakar. Pohon berakar pertama memilih *b* sebagai akar sedangkan pohon akar kedua memilih *e* sebagai akar.



Gambar 9.10 (kiri) Pohon dan (kanan) dua buah pohon berakar yang dihasilkan dari pemilihan dua simpul berbeda sebagai akar

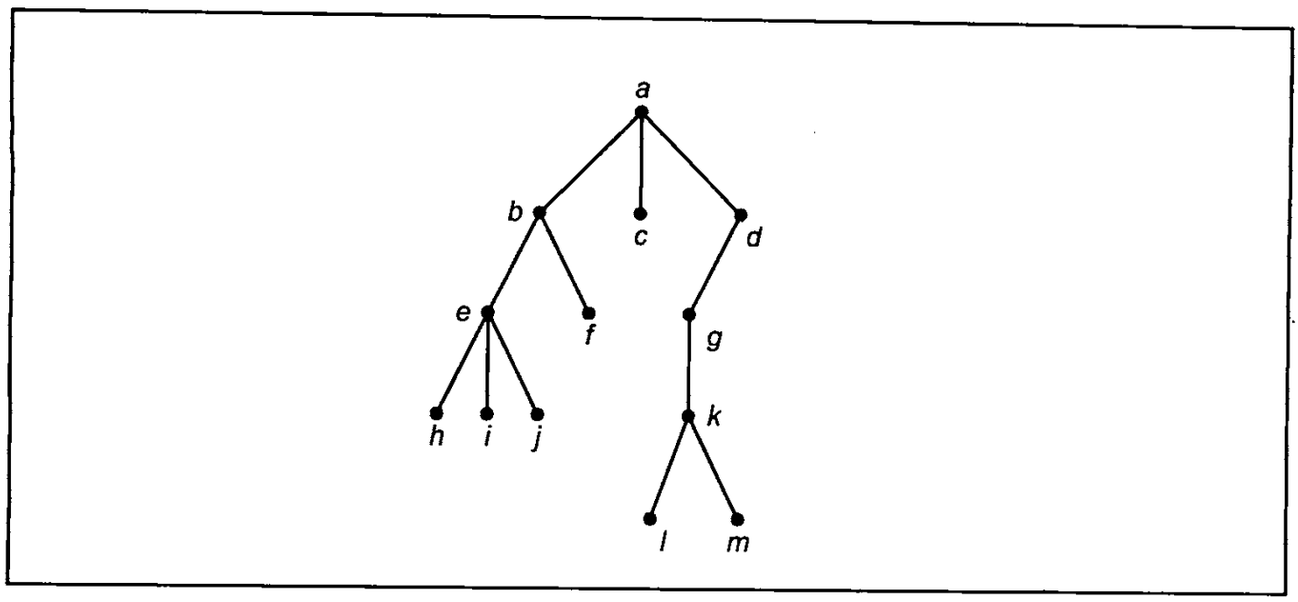
9.6 Terminologi pada Pohon Berakar

Keseluruhan sisa bab ini akan menggunakan istilah "pohon berakar" untuk membedakannya dengan "pohon" tidak berakar (*free tree*) (sebagaimana yang telah disinggung di dalam upabab 9.1).

Sebagaimana pada graf, kita akan sering menggunakan terminologi yang berhubungan dengan pohon. Di bawah ini didaftarkan beberapa terminologi yang penting pada pohon berakar. Untuk ilustrasi, pohon pada Gambar 9.12 dipakai sebagai contoh untuk menjelaskan terminologi yang dimaksudkan. Simpul-simpul pada pohon diberi label untuk mengacu simpul mana yang dimaksudkan. Kebanyakan terminologi pohon yang ditulis di bawah ini diadopsi dari terminologi botani dan silsilah keluarga.

Anak (*child* atau *children*) dan Orangtua (*parent*)

Misalkan *Xx* adalah sebuah simpul di dalam pohon berakar. Simpul *y* dikatakan **anak** simpul *x* jika ada sisi dari simpul *x* ke *y*. Dalam hal demikian, *x* disebut **orangtua** (*parent*) *y*. Pada Gambar 9.11, *b*, *c*, dan *d* adalah anak-anak simpul *a*, dan *a* adalah orangtua dari anak-anak itu. *e* dan *f* adakah anak-anak simpul *b*, dan *b* adalah orangtua dari *e* dan *f*. *g* adalah anak simpul *d*, dan *d* adalah orangtua *g*. Simpul *h*, *i*, *j*, *l*, dan *m* tidak mempunyai anak.



Gambar 9.11 Pohon berakar yang digunakan untuk menjelaskan terminologi pohon

Lintasan (*path*)

Lintasan dari simpul v_1 ke simpul v_k adalah runtunan simpul-simpul v_1, v_2,..., v_k sedemikian sehingga v_i adalah orangtua dari v_{i+1} untuk $1 \leq i < k$. Dari pohon pada Gambar 9.11, lintasan dari *a* ke *j* adalah *a*, *b*, *e*, *j*. **Panjang lintasan** adalah jumlah sisi yang dilalui dalam suatu lintasan, yaitu $k - 1$. Panjang lintasan dari *a* ke *j* adalah 3.

Keturunanan (*descendant*) dan Leluhur (*ancestor*)

Jika terdapat lintasan dari simpul *x* ke simpul *y* di dalam pohon, maka *x* adalah **leluhur** dari simpul *y*, dan *y* adalah adalah **keturunan** simpul *x*. Pada Gambar 9.11, *b* adalah leluhur simpul *h*, dan dengan demikian *h* adalah keturunan *b*.

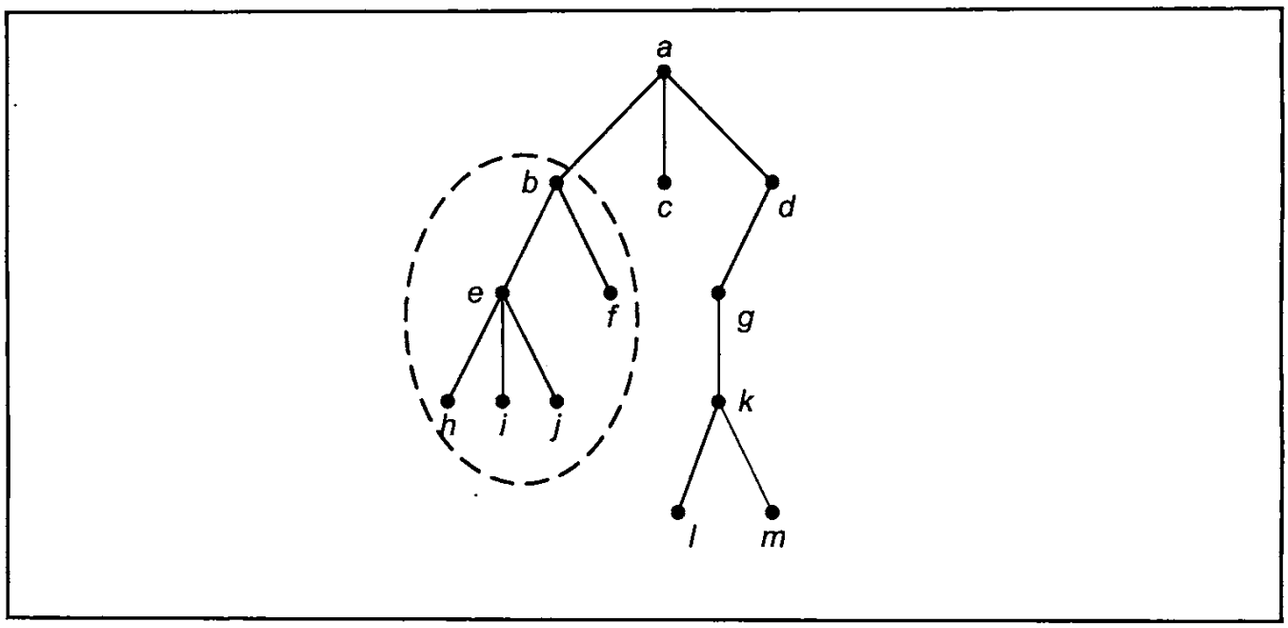
Saudara kandung (*sibling*)

Simpul yang berorangtua sama adalah **saudara kandung** satu sama lain. Pada Gambar 9.11, *f* adalah saudara kandung *e*. Tetapi, *g* bukan saudara kandung *e*, karena orangtua mereka berbeda.

Upapohon (*subtree*)

Misalkan *x* adalah simpul di dalam pohon *T*. Yang dimaksud dengan **upapohon** dengan *x* sebagai akarnya ialah upagraf $T' = (V', E')$ sedemikian sehingga V' mengandung *x* dan semua keturunannya dan E' mengandung sisi-sisi dalam semua lintasan yang berasal dari *x*. Sebagai contoh, $T' = (V', E')$ adalah upapohon dari pohon pada Gambar 9.12 dengan $V' = \{ b, e, f, h, i, j\}$ dan $E' = \{$

(b, e), (b, f), (e, h), (e, i), (e, j) } dan b adalah simpul akarnya. Terdapat banyak upapohon di dalam pohon T. Dengan pengertian di atas, jika x adalah simpul, maka akar tiap-tiap upapohon dari x disebut anak, dan x adalah orangtua setiap akar upapohon.



Gambar 9.12 Upapohon $T' = (V, E)$ dengan b sebagai akarnya

Derajat (*degree*)

Ada perbedaan definisi derajat pada pohon berakar dengan definisi derajat pada graf (termasuk pohon tidak berakar). **Derajat** sebuah simpul pada pohon berakar adalah jumlah upapohon (atau jumlah anak) pada simpul tersebut. Pada Gambar 9.11, derajat a adalah 3, derajat b adalah 2, derajat d adalah satu dan derajat c adalah 0. Jadi, derajat yang dimaksudkan di sini adalah derajat-keluar.

Derajat maksimum dari semua simpul merupakan derajat pohon itu sendiri. Pohon pada Gambar 9.11 berderajat 3, karena derajat tertinggi dari seluruh simpulnya adalah 3.

Daun (*leaf*)

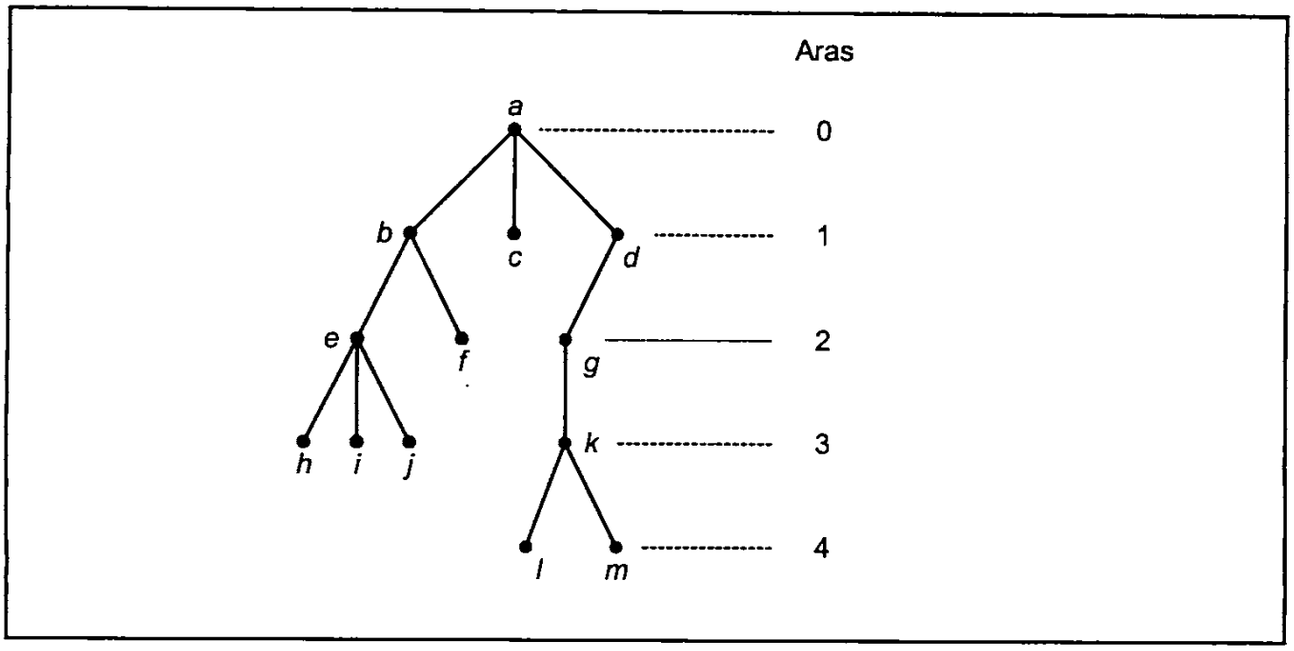
Simpul yang berderajat nol (atau tidak mempunyai anak) disebut **daun**. Simpul h, i, j, f, c, l, dan m adalah daun.

Simpul Dalam *(internal nodes)*

Simpul yang mempunyai anak disebut **simpul dalam**. Simpul d, e, g, dan k paad Gambar 9.11 adalah simpul dalam.

Aras (*level*) atau Tingkat

Akar mempunyai aras = 0, sedangkan aras simpul lainnya = 1 + panjang lintasan dari akar ke simpul tersebut. Beberapa literatur memulai nomor aras dari 0, literatur lainnya dari 1. Sebagai konvensi, kita memulai penomoran aras dari 0. Lihat Gambar 9.13.



Gambar 9.13 Pendefinisian aras tiap simpul

Tinggi (*height*) atau Kedalaman (*depth*)

Aras maksimum dari suatu pohon disebut **tinggi** atau **kedalaman** pohon tersebut. Atau, dapat juga dikatakan, tinggi pohon adalah panjang maksimum lintasan dari akar ke daun. Pohon pada Gambar 9.13 mempunyai tinggi 4.

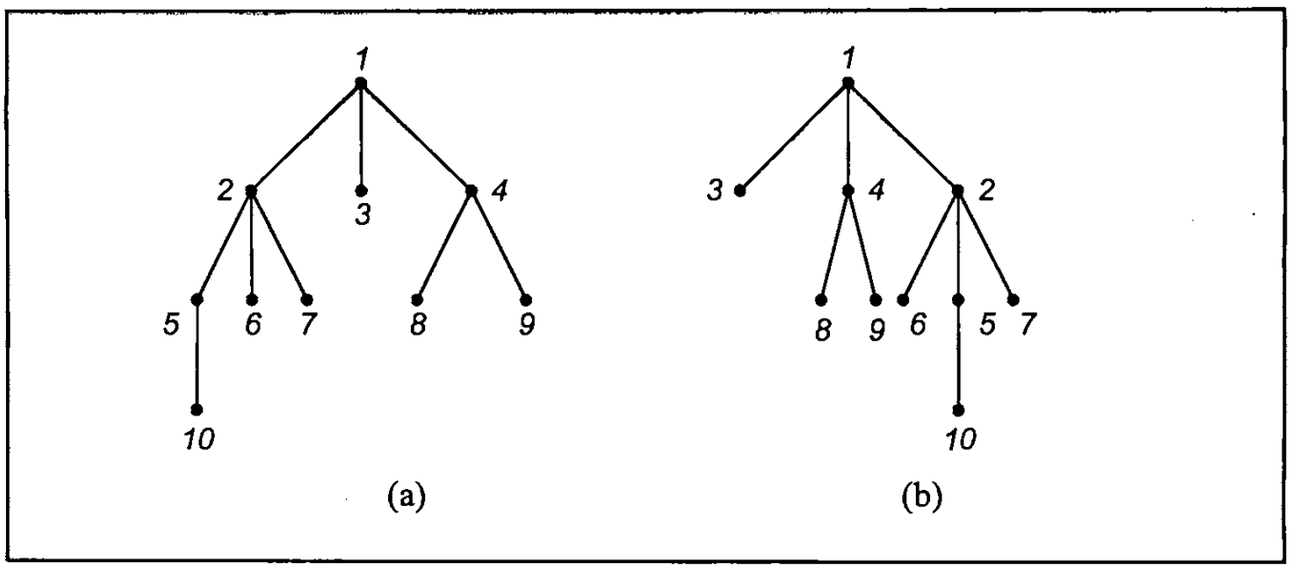
9.7 Pohon Berakar Terurut

DEFINISI 9.3. Pohon berakar yang urutan anak-anaknya penting disebut **pohon terurut** (*ordered tree*).

Pada pohon terurut, urutan anak-anak dari simpul dalam dispesifikasikan dari kiri ke kanan. Sebagai contoh, dua buah pohon pada Gambar 9.14 adalah pohon berakar yang sama, tetapi sebagai pohon terurut, keduanya berbeda. Misalnya urutan anak-anak dari simpul 1 pada Gambar 9.14(a) adalah 2, 3, 4, sedangkan urutan anak-anak dari simpul 1 pada Gambar 9.14(b) adalah 3, 4, 2.

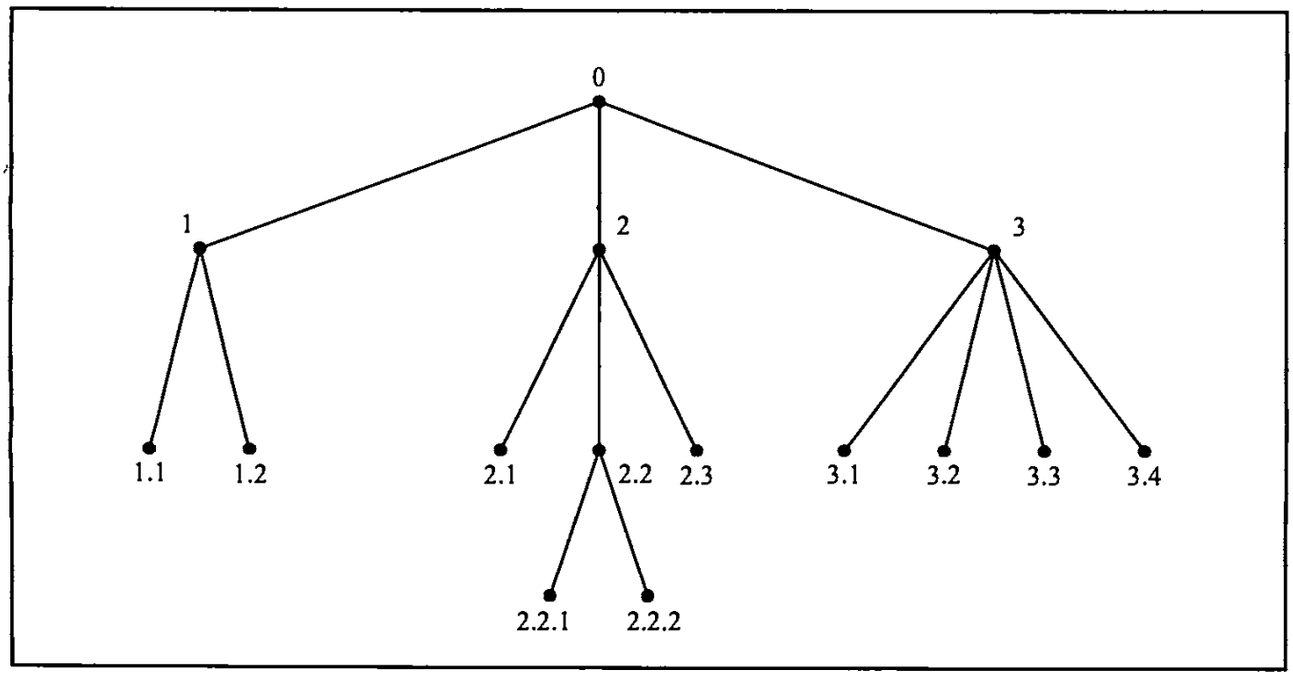
Perbedaan ini menjadi penting bila kita merepresentasikan pohon di dalam komputer, karena penelusuran dua buah pohon terurut yang berbeda akan menghasilkan urutan simpul yang berbeda pula. Jika pohon berakar terurut pada

simpul x mempunyai p buah upapohon, kita akan mengacunya sebagai upapohon pertama, upapohon kedua, ..., upapohon ke-p.



Gambar 9.14 Dua buah pohon terurut yang berbeda

Sistem yang universal dalam pengalamatan simpul-simpul pada pohon terurut adalah dengan memberi nomor setiap simpulnya seperti penomoran bab (beserta upababnya) di dalam sebuah buku. Simpul akar diberi nomor 0, simpul lain yang segera mengikuti akar diberi nomor 1, 2, 3, dan seterusnya. Anak-anak simpul 1 diberi nomor 1.1, 1.2, ...; anak-anak simpul 2 diberi nomor 2.1, 2.2, ...; demikian seterusnya (Gambar 9.15). Semua penomoran dimulai dengan aturan dari kiri ke kanan.



Gambar 9.15 Sistem pengalamatan universal pada pohon terurut.

9.8 Pohon *m-ary*

DEFINISI 9.5. Pohon berakar yang setiap simpul cabangnya mempunyai paling banyak *n* buah anak disebut **pohon *m-ary***.

Jika $m = 2$, pohon disebut **pohon biner** (*binary tree*). Pohon pada Gambar 9.15 adalah pohon 3-*ary*. Pohon *m-ary* dikatakan **pohon penuh** (*full*) atau **pohon teratur** jika setiap simpul cabangnya mempunyai tepat *m* buah anak.

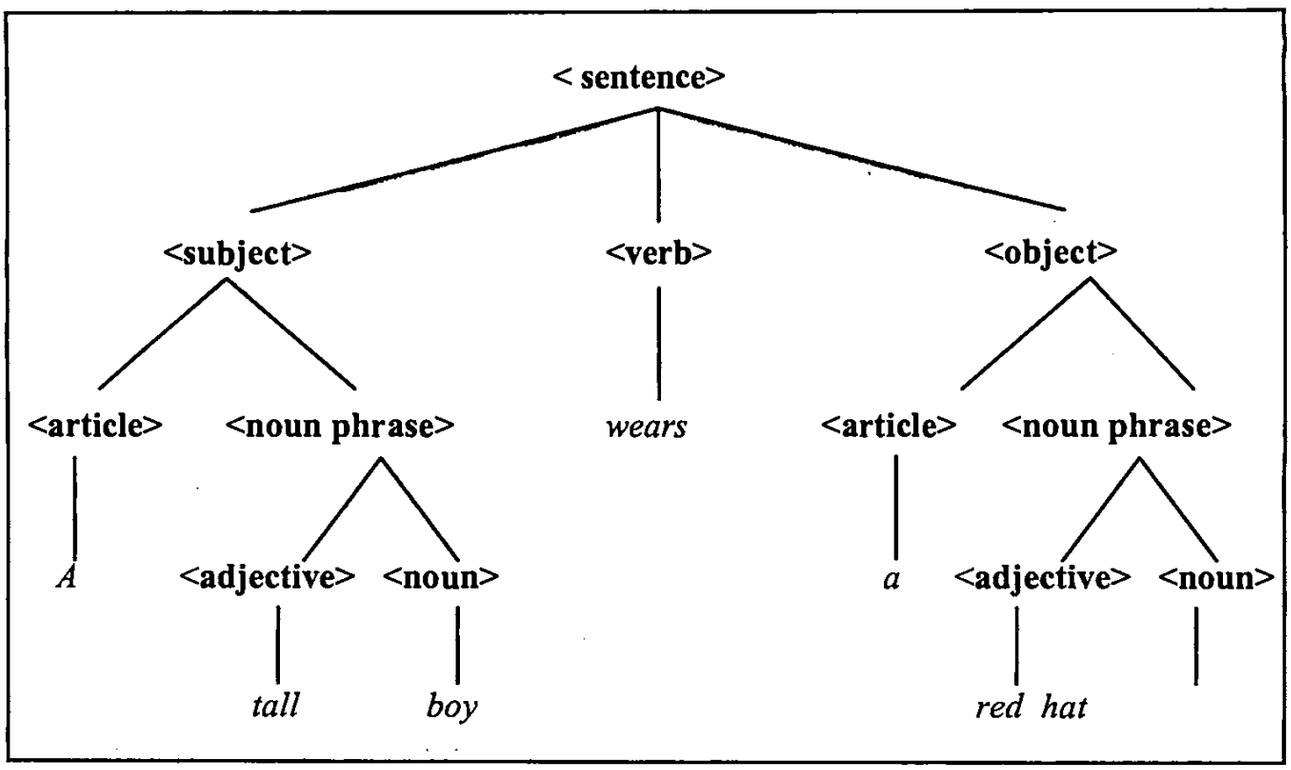
Pohon *m-ary* banyak digunakan di berbagai bidang ilmu maupun dalam kehidupan sehari-hari. Dalam terapannya, pohon *m-ary* digunakan sebagai model yang merepresentasikan suatu struktur. Dua contoh penggunaan pohon *m-ary* dikemukakan di bawah ini, yaitu penurunan kalimat (dalam bidang bahasa) dan direktori arsip di dalam komputer. Contoh penggunaan pohon *m-ary* lainnya adalah struktur organisasi, silsilah keluarga (dalam bidang genetika), struktur bab atau daftar isi di dalam buku, bagan pertandingan antara beberapa tim sepakbola, dan sebagainya.

Contoh 9.5

Pohon *m-ary* digunakan untuk merepresentasikan struktur kalimat dalam bahasa alami (*natural language*) maupun dalam bahasa pemrograman. Pohon penurunan kalimat itu disebut pohon *parsing* (*parse tree*). Gambar 9.16 memperlihatkan cara penurunan kalimat dalam Bahasa Inggris yang berbunyi

A tall boy wears a red hat

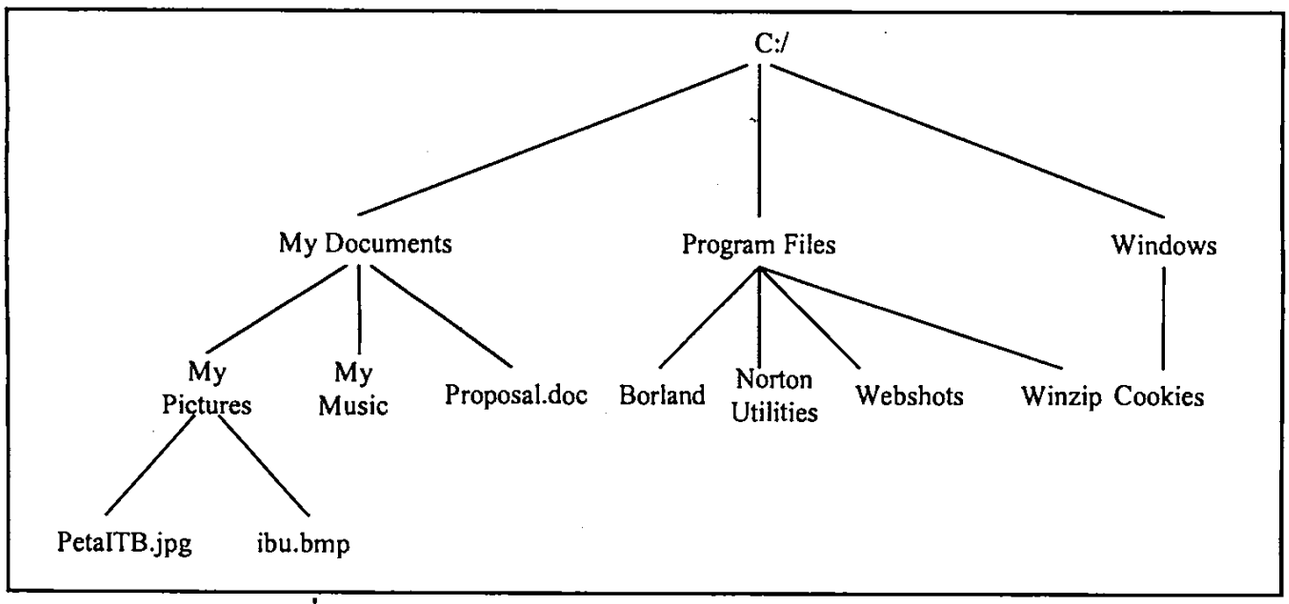
Akar menyatakan kalimat (*sentence*), daun menyatakan setiap kata-kata yang menyusun kalimat, sedangkan simpul dalam menyatakan cara pembagian kalimat menjadi unsur-unsur pembentuknya. Sebuah kalimat (*sentence*) dalam Bahasa Inggris disusun oleh *subject*, *verb* dan *object*. *Subject* dapat terdiri dari sebuah *article* dan *noun phrase*. Sebuah *noun phrase* dapat terdiri atas *adjective* dan *noun*. Begitu juga *object* dapat terdiri atas sebuah *article* dan *noun phrase*. ■



Gambar 9.16 Pohon parsing dari kalimat *A tall boy wears a red hat*

Contoh 9.6

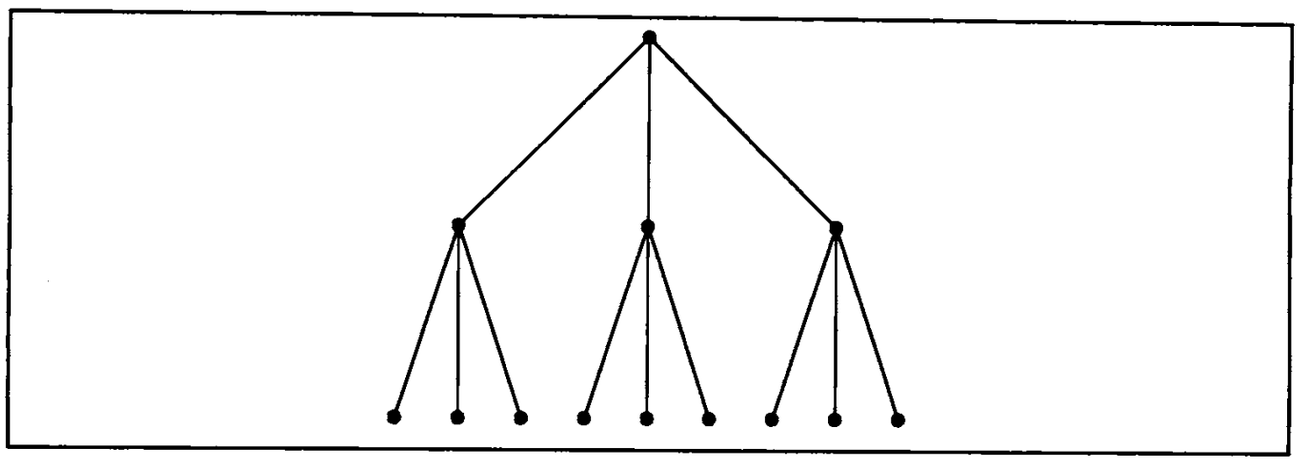
Dalam sistem pengarsipan komputer (*computer file system*), arsip-arsip di dalam media penyimpanan sekunder (seperti *floppy disk*, *compact disk*, *hard disk*), disusun dalam bentuk direktori. Suatu direktori dapat mengandung beberapa upa-direktori (*sub-directory*). Struktur direktori dimodelkan dalam pohon *m-ary*. Di sini akar menyatakan sistem arsip keseluruhan, simpul dalam menyatakan upa-direktori, dan daun menyatakan arsip atau direktori kosong. Gambar 9.17 memperlihatkan struktur direktori dalam *platform* sistem operasi *Windows*. ■



Gambar 9.17 Struktur direktori arsip di dalam sistem operasi *Windows*

Jumlah Daun pada Pohon *m-ary* Penuh

Pohon *m-ary* penuh adalah pohon yang setiap simpulnya tepat mempunyai m buah anak. Pada pohon *m-ary* penuh dengan tinggi h, jumlah daun adalah m^h. Perhatikan bahwa jika T bukan pohon *m-ary* penuh, maka jumlah daun $\leq m^h$. Gambar 9.18 adalah pohon 3-*ary* penuh dengan jumlah daun = $3^2 = 9$.



Gambar 9.18 Pohon 3-*ary* penuh dengan jumlah daun = $3^2 = 9$.

Jumlah Seluruh Simpul pada Pohon *m-ary* Penuh

Pada pohon *m-ary* penuh dengan tinggi h,

aras 0 → jumlah simpul = $m^0 = 1$
aras 1 → jumlah simpul = m^1
aras 2 → jumlah simpul = m^2
...
aras h → jumlah simpul = m^h

maka jumlah seluruh simpul adalah

$$S = m^0 + m^1 + m^2 + \ldots + m^h = \frac{m^{h+1} - 1}{m - 1} \qquad (9.1)$$

Perhatikan bahwa jika T bukan pohon *m-ary* penuh, maka $S \leq \frac{m^{h+1} - 1}{m - 1}$.

Hubungan Jumlah Daun dan Simpul Dalam pada Pohon *m-ary* Penuh

Contoh 9.7

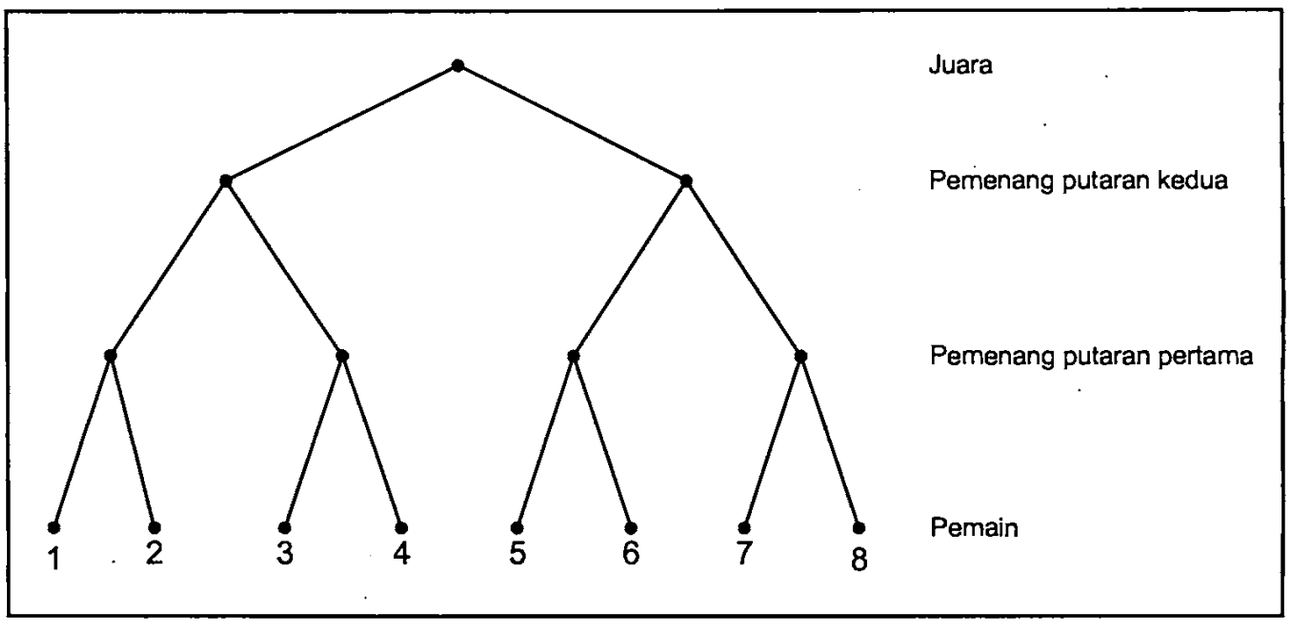
[LIU85] Misalkan sebuah turnamen tenis diikuti oleh t orang tim dengan sistem pertandingan gugur (pemain yang kalah langsung tersingkir, pemain yang menang melawan pemenang pertandingan lainnya). Berapa banyak pertandingan yang terjadi?

Penyelesaian:
Persoalan ini dapat dimodelkan dengan pohon biner penuh. Daun menyatakan pemain, simpul dalam (termasuk akar) menyatakan para pemenang pertandingan (jumlah seluruh simpul dalam berarti juga total banyak pertandingan yang dilakukan sampai mencapai juara).

Misalkan i adalah banyaknya simpul dalam dan t adalah banyaknya simpul daun di dalam pohon biner penuh. Karena di dalam sistem gugur setiap pertandingan menggugurkan seorang pemain, dan pada akhir pertandingan yang dimainkan semua peserta telah gugur kecuali sang juara, maka banyaknya pertandingan yang terjadi satu lebih sedikit daripada jumlah pemain. Jadi, diperoleh hubungan:

$$i = t - 1 \qquad (9.2)$$

Gambar 9.19 adalah pohon biner teratur yang memodelkan sistem gugur untuk $t = 8$ orang pemain. Banyaknya pertandingan yang terjadi adalah $i = 8 - 1 = 7$ kali. ■



Gambar 9.19 Pohon pertandingan turnamen tenis dengan sistem gugur

Persamaan di atas dapat dirampatkan untuk pohon *m-ary* penuh. Misalkan setiap pertandingan diikuti oleh m orang pemain dan hanya satu yang keluar sebagai pemenang. Jadi, setiap pertandingan menggugurkan $m - 1$ orang pemain. Kita peroleh hubungan:

$$(m - 1)i = t - 1 \qquad (9.3)$$

Contoh 9.6

[LIU85] Misalkan kita akan menyambungkan 19 buah lampu pada satu stop kontak dengan menggunakan sejumlah kabel ekstensi yang masing-masing mempunyai empat

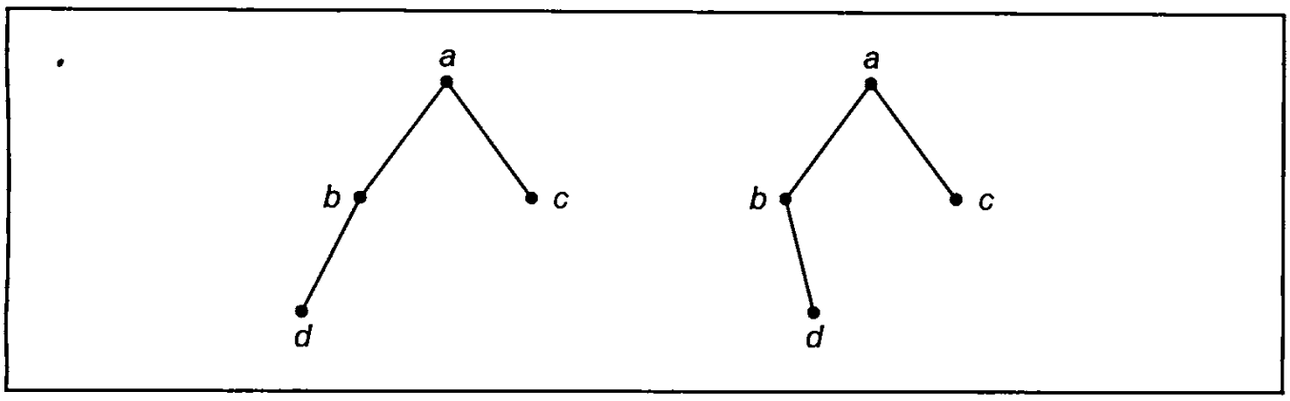
outlet. Karena penyambungan semacam itu merupakan pohon 4-*ary* dengan stop kontak sebagai akar pohon, maka

$$(4 - 1)\, i = 19 - 1$$

diperoleh $i = 6$. Jadi, dibutuhkan enam buah kabel ekstensi. ■

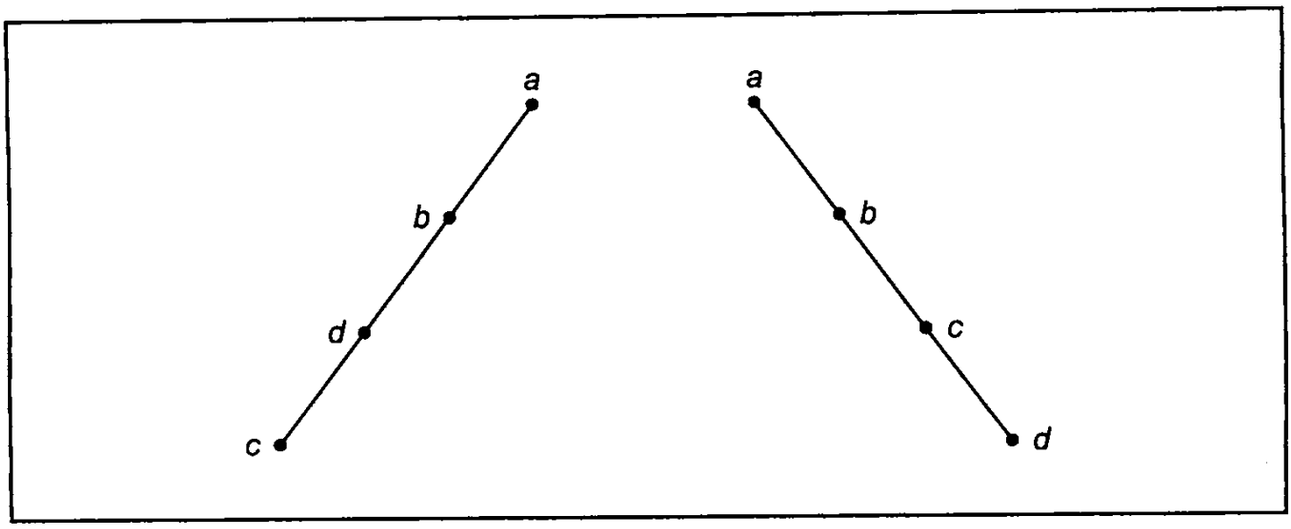
9.9 Pohon Biner

Pohon *m-ary* yang paling penting adalah pohon biner (*binary tree*). Pohon biner merupakan pohon *m-ary* jika $m = 2$. Pohon biner adalah pohon yang setiap simpul cabangnya mempunyai paling banyak dua buah anak. Alih-alih menyebutnya anak pertama dan anak kedua dari suatu simpul dalam, kita menyebutnya **anak kiri** (*left child*) dan **anak kanan** (*right child*). Pohon yang akarnya adalah anak kiri disebut **upapohon kiri** (*left subtree*), sedangkan pohon yang akarnya adalah anak kanan disebut **upapohon kanan** (*right subtree*). Karena adanya perbedaan anak/upapohon kiri dan anak/upapohon kanan, maka pohon biner adalah pohon terurut. Dua buah pohon pada Gambar 9.20 adalah dua buah pohon biner berbeda.



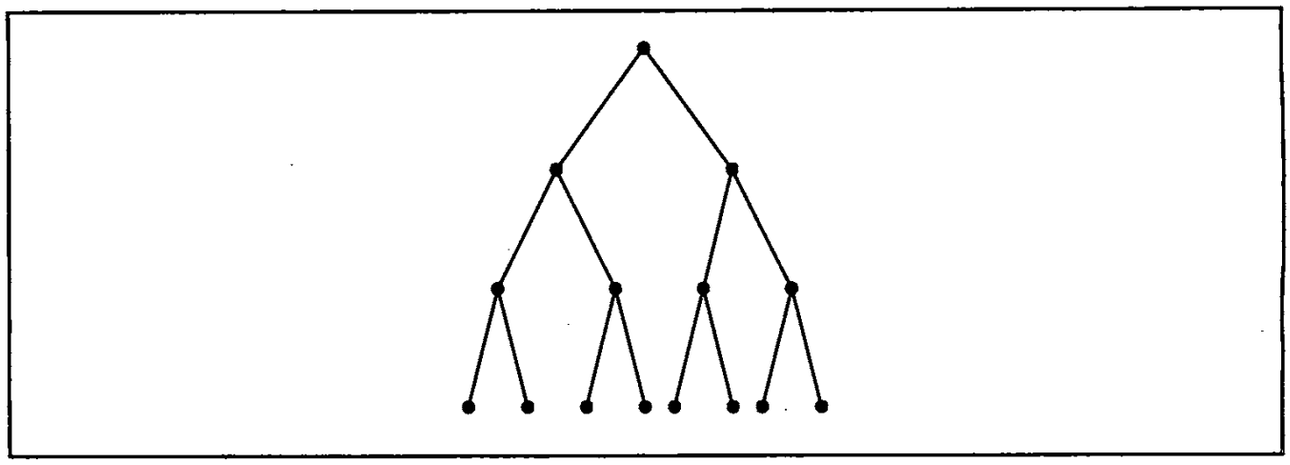
Gambar 9.20 Dua buah pohon biner yang berbeda

Pohon yang semua simpulnya terletak di bagian kiri saja atau di bagian kanan saja disebut **pohon condong** (*skewed tree*). Pohon yang condong ke kiri disebut **pohon condong-kiri** (*skew left*), pohon yang condong ke kanan disebut **pohon condong-kanan** (*skew right*). Lihat Gambar 9.21.



Gambar 9.21 (a) Pohon condong-kiri, dan (b) pohon condong kanan

Pohon biner penuh (*full binary tree*) adalah pohon biner yang setiap simpulnya mempunyai tepat dua buah anak, kiri dan kanan, kecuali simpul pada aras bawah (Gambar 9.22).



Gambar 9.22 Pohon biner penuh

Pohon biner penuh dengan tinggi h memiliki jumlah daun sebanyak 2^h, sedangkan jumlah seluruh simpulnya adalah:

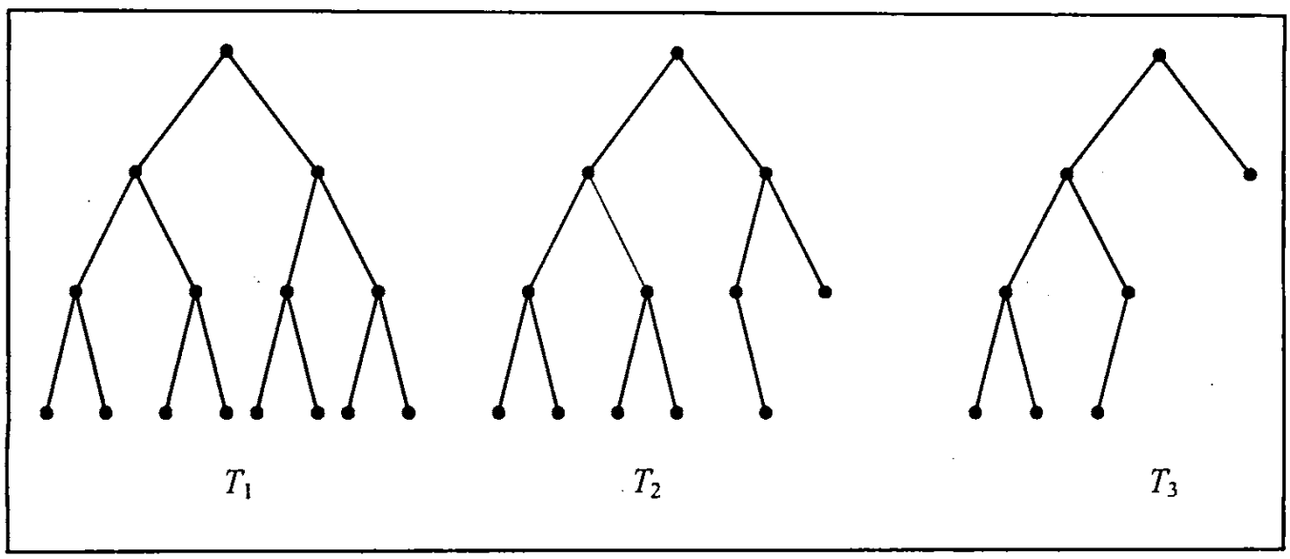
$$S = 2^0 + 2^1 + 2^2 + \ldots + 2^h = 2^{h+1} - 1 \qquad (9.4)$$

Pohon Biner Seimbang

Pohon biner seimbang (*balanced binary tree*) adalah pohon biner yang perbedaan tinggi antara upapohon kiri dan upapohon maksimal 1. Pada pohon biner seimbang dengan tinggi h, semua daun berada pada aras h atau $h - 1$. Untuk membuat pohon seimbang, tinggi pohon secara keseluruhan harus dibuat

seminimal mungkin. Untuk memperoleh tinggi minimum, setiap aras harus mengandung jumlah simpul sebanyak mungkin. Hal ini dapat dibuat dengan menyebarkan setengah dari jumlah simpul di upapohon kiri dan setengah dari jumlah simpul di upapohon kanan.

Pohon T_1 dan T_2 pada Gambar 9.23 adalah pohon seimbang, sedangkan T_3 bukan pohon seimbang karena perbedaan upapohon kiri dan upapohon kanan tidak maksimal satu (periksa!).



Gambar 9.23 T_1 dan T_2 adalah pohon seimbang, sedangkan T_3 bukan pohon seimbang.

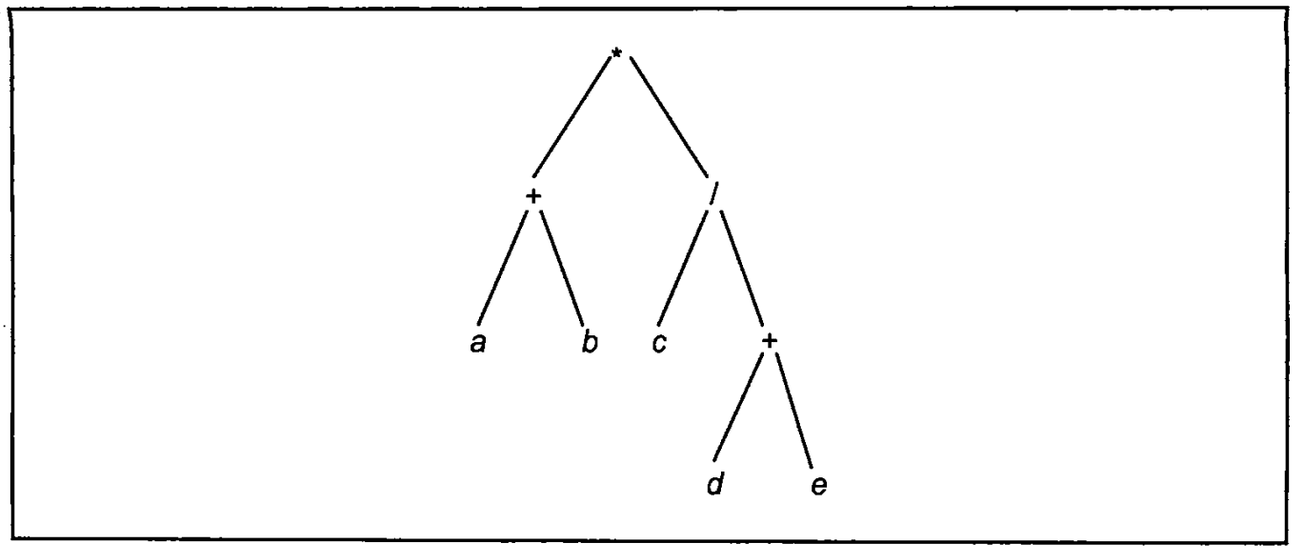
Pohon biner merupakan struktur yang penting dalam ilmu komputer. Terapan pohon biner di dalam ilmu komputer sangat banyak, di antaranya yang disebutkan di sini adalah pohon ekspresi (*expression tree*), pohon keputusan (*decision tree*), kode prefiks (*prefix code*), kode Huffman (*Huffman code*), dan pohon pencarian biner (*binary search tree*). Masing-masing terapan pohon biner di atas diuraikan pada masing-masing upabab di bawah ini.

9.10 Pohon Ekspresi

1. Pohon Ekspresi

Pohon ekspresi ialah pohon biner dengan daun menyatakan *operand* dan simpul dalam (termasuk akar) menyatakan *operator*. Perhatikan bahwa tanda kurung tidak lagi diperlukan bila suatu ekspresi aritmetik direpresentasikan sebagai pohon biner.

Sebagai contoh, ekspresi $(a + b)*(c/(d + e))$ dinyatakan dalam pohon biner pada Gambar 9.24. Daun menyatakan *operand* a, b, c, d, dan e, sedangkan simpul dalam –termasuk akar– menyatakan *operator* +, *, dan /.



Gambar 9.24 Pohon ekspresi dari $(a + b)*(c/(d + e))$

Pohon ekspresi digunakan oleh *compiler* bahasa tingkat tinggi untuk mengevaluasi ekspresi yang ditulis dalam notasi *infix*, *prefix* (*polish notation*), dan *prefix* (*inverse Polish notation*).

Dalam notasi *infix*, operator berada di antara dua buah *operand*, pada notasi *prefix*, operator mendahului dua buah *operand*-nya, dan pada notasi *postfix* kedua *operand* mendahului operatornya.

Ekspresi $(a + b)*(c/(d + e))$ adalah dalam bentuk *infix*, sedangkan ekspresi *prefix*-nya adalah

$$* + a\ b\ /\ c + d\ e$$

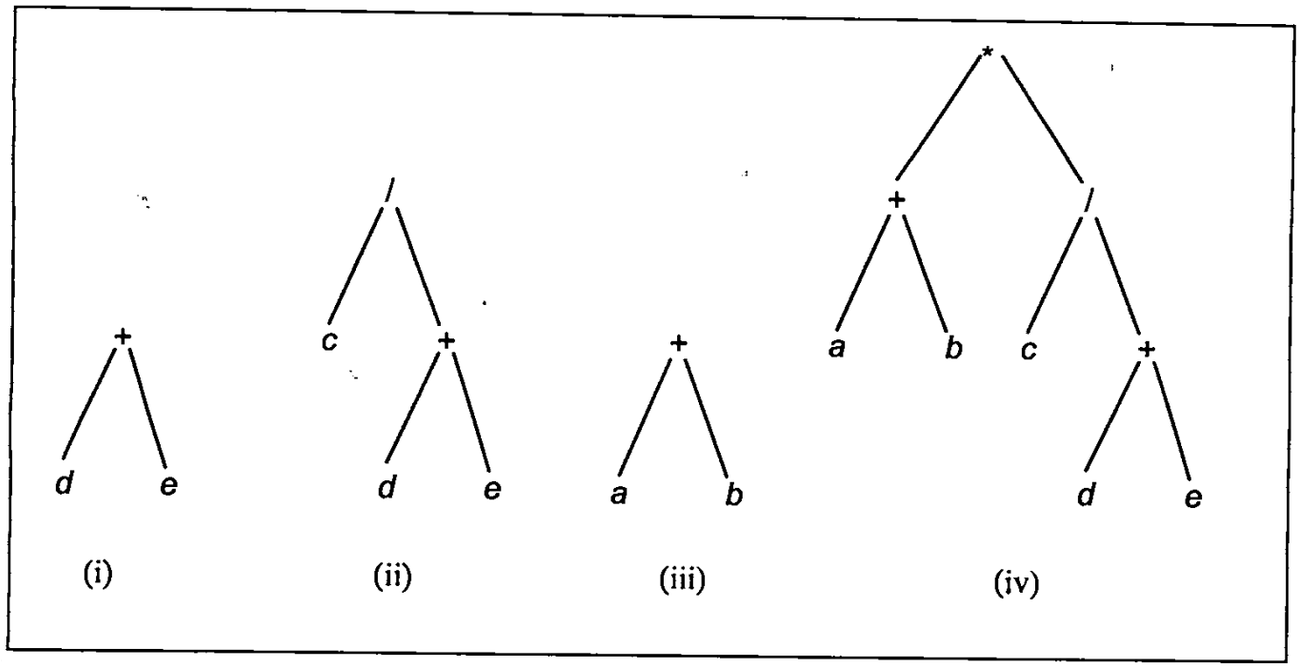
dan ekspresi *postfix*-nya adalah

$$a\ b + c\ d\ e + /\ *$$

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 $(a + b)*(c/(d + e))$

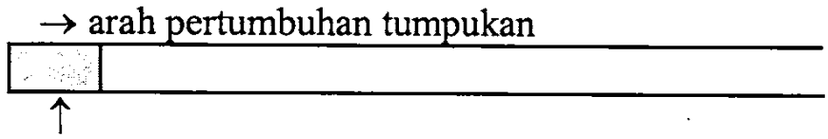
Pembentukan Pohon Ekspresi dari Notasi *Postfix*

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").



```
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*

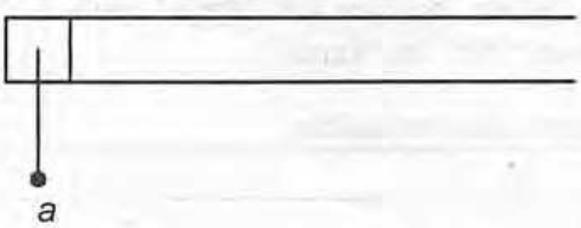
Algoritma pembentukan pohon ekspresi dari notasi *prefix* relatif sukar dipahami sehingga tidak dibicarakan di dalam bab ini.

Contoh 9.8

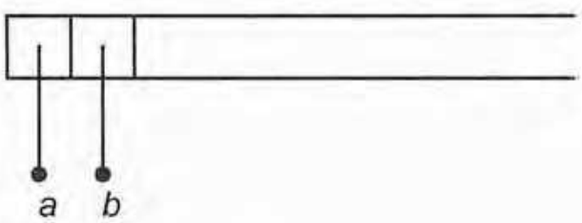
Terapkan algoritma `BangunPohonEkspresiDariPostfix` untuk membangun pohon ekspresi dari notasi *postfix* $a\ b + c\ d\ e + / *$

Penyelesaian:

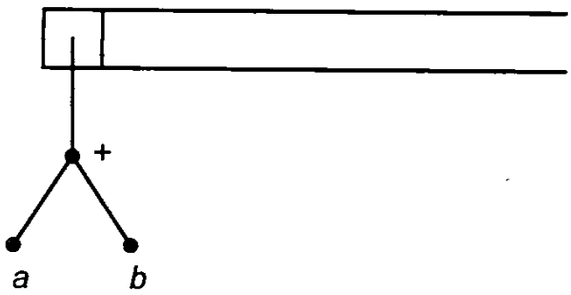
(i) Mulai dari elemen *potfix* pertama, P_1. Karena P_1 = '*a*' = *operand*, buat simpul untuk P_1, *push pointer*-nya ke dalam tumpukan *S*.



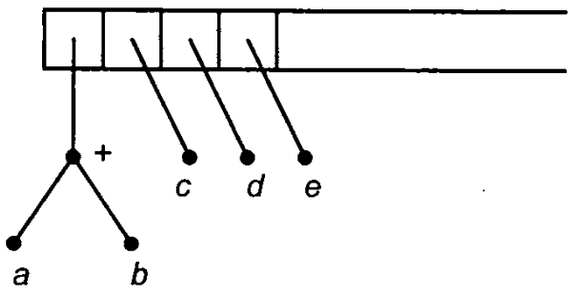
(ii) Baca P_2. Karena P_2 = '*b*' = *operand*, buat simpul untuk P_2, *push* pointer-nya ke tumpukan *S*.



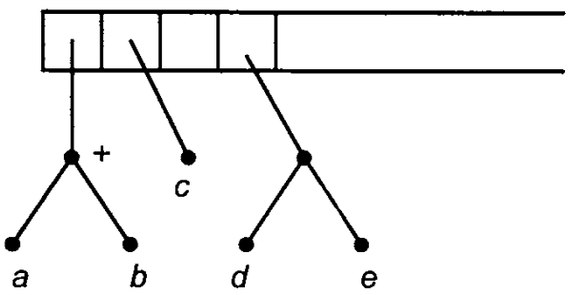
(iii) Baca P_3. Karena P_3 = '+' = *operator*, buat pohon T dengan 'a' dan 'b' sebagai anak.



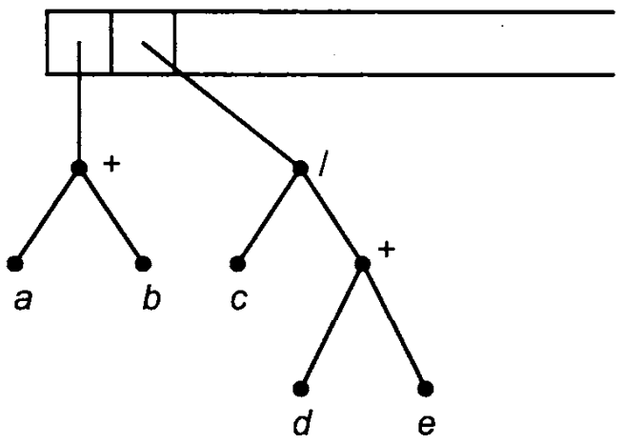
(iv) Baca P_4, P_5, P_6. Karena P_4, P_5, P_6 = *operand*, buat simpul untuk P_4, P_5, dan P_6. *Push pointer*-nya ke dalam tumpukan S.



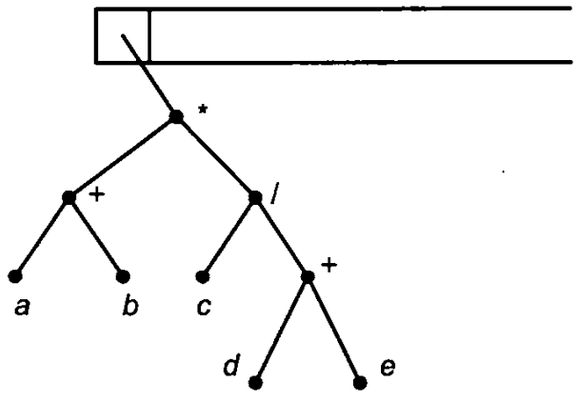
(v) Baca P_7. Karena P_7 = '+' = *operator*, buat pohon T dengan 'd' dan 'e' sebagai anak.



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



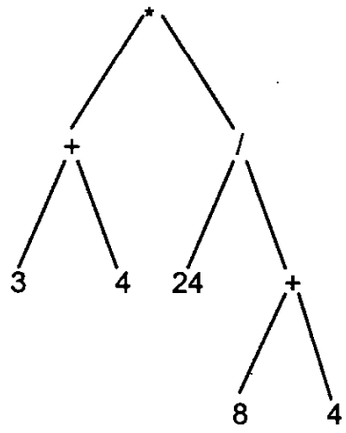
(vii) Baca P_9. Karena P_9 = '*' = operator, buat pohon T dengan '+' dan '*' sebagai anak.



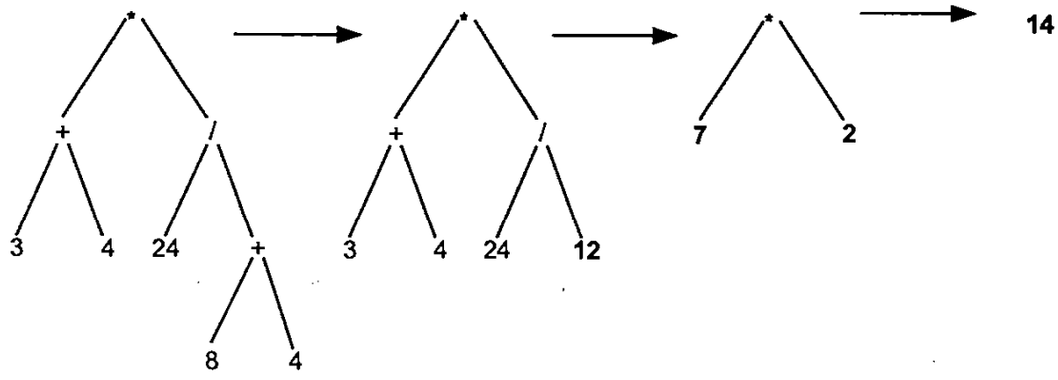
Karena semua elemen tabel P sudah habis dibaca, maka tumpukan S akan berisi *pointer* yang menunjuk ke akar pohon ekspresi. ■

Contoh 9.9

Evaluasi pohon ekspresi berikut:



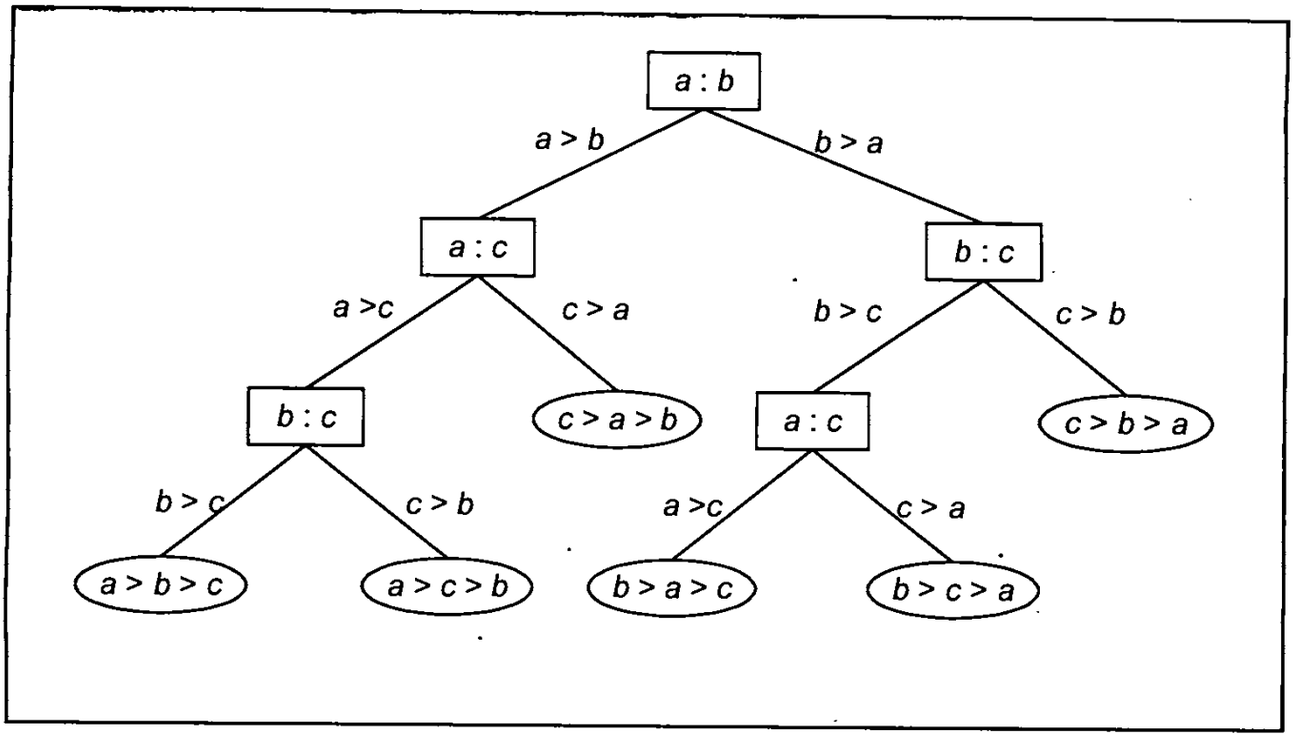
Penyelesaian:
Pohon ekspresi dievaluasi mulai dari bawah ke atas. Dua buah daun (yang merepresentasikan *operand*) diperasikan dengan operator yang menjadi orangtuanya. Nilai evaluasi sementara (dicetak tebal) disimpan pada simpul orangtua tadi, dan kedua *operand* yang dioperasikan dihapus. Pada akhir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



■

9.11 Pohon Keputusan

Pohon keputusan digunakan untuk memodelkan persoalan yang teriri dari serangkaian keputusan yang mengarah ke solusi. Tiap simpul dalam menyatakan keputusan, sedangkan daun menyatakan solusi. Sebagai contoh, kita ingin mengurutkan tiga buah bilangan, *a*, *b*, dan *c*. Pohon keputusan untuk persoalan ini ditunjukkan pada Gambar 9.26.

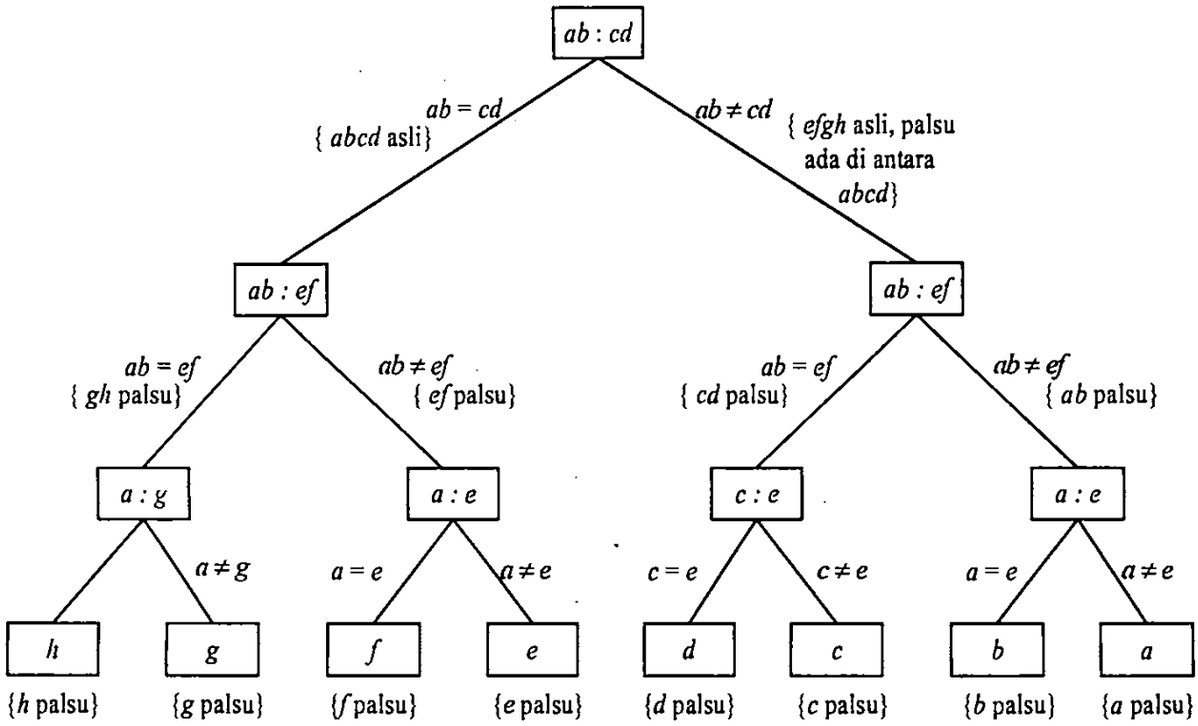


Gambar 9.26 Pohon keputusan untuk mengurutkan 3 buah elemen

Contoh 9.10

Diketahui 8 buah koin uang logam. Satu dari delapan koin itu ternyata palsu. Koin yang palsu mungkin lebih ringan atau lebih berat daripada koin yang asli. Misalkan tersedia sebuah timbangan neraca yang sangat teliti. Buatlah pohon keputusan untuk mencari uang palsu dengan cara menimbang paling banyak hanya 3 kali saja.

Penyelesaian:
Misalkan 8 koin itu dinamai *a*, *b*, *c*, *d*, *e*, *f*, *g*, *h*. Pohon keputusan untuk mencari koin yang palsu ditunjukkan di bawah ini. Daun menyatakan koin yang palsu.



■

9.12 Kode Awalan

Kode awalan (*prefix code*) adalah himpunan kode, misalnya kode biner, sedemikian sehingga tidak ada anggota kumpulan yang merupakan awalan dari anggota yang lain. Contohnya, himpunan

{ 000, 001, 01, 10, 11 }

adalah kode awalan, tetapi

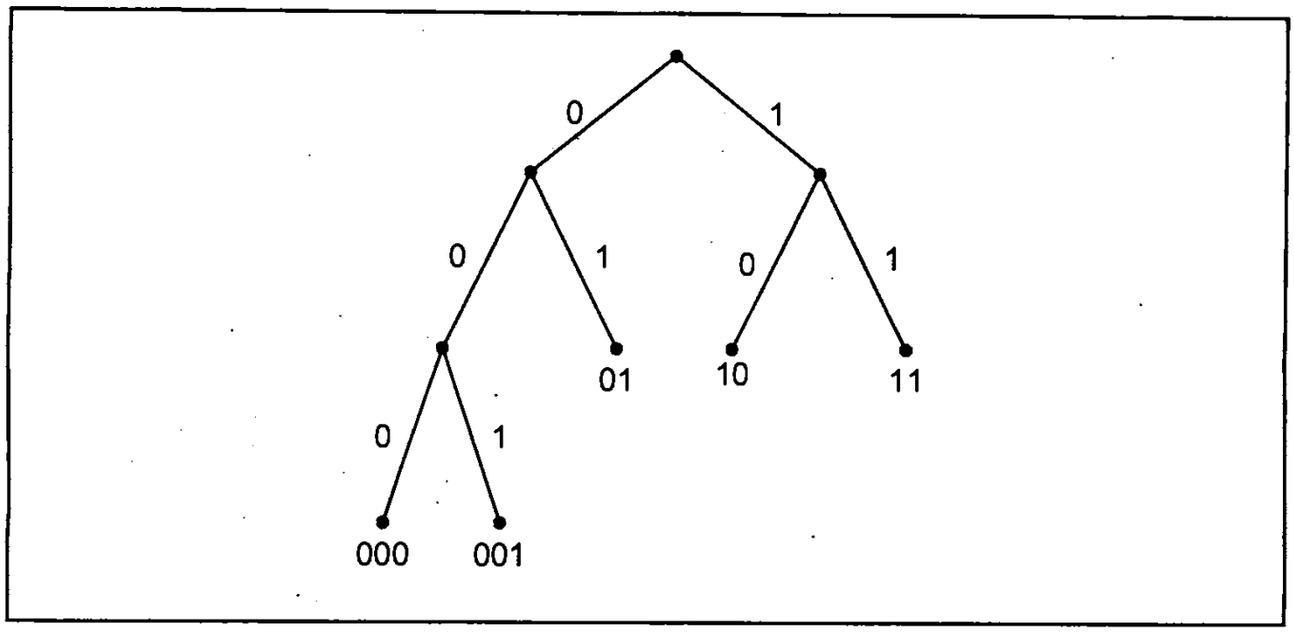
{ 1, 00, 01, 000, 0001 }

bukan kode awalan, sebab 00 adalah prefiks dari 0001.

Kode awalan mempunyai pohon biner yang bersesuaian. Sisi diberi label 0 atau 1. Pelabelan sisi harus taat-asas, yakini semua sisi kiri dilabeli 0 saja (atau 1 saja), sedangkan sisi kanan dilabeli 1 saja (atau 0 saja). Barisan sisi-sisi yang dilalui oleh lintasan dari akar ke daun menyatakan kode awalan. Kode awalan ini ditulis pada daun. Contoh himpunan yang pertama mempunyai pohon biner pada Gambar 9.27.

Kegunaan kode awalan adalah untuk mengirim pesan pada komunikasi data. Setiap karakter di dalam pesan direpresentasikan dengan barisan angka 0 dan 1. Untuk mengirim pesan, kita cukup mengirimkan *string* angka 0 dan 1 yang merepresentasikan karakter di dalam pesan. Oleh pihak penerima, *string* angka 0

dan 1 ini dikembalikan lagi ke karakter penyusun pesan semula. Agar tidak timbul ambigu dalam mengkonversikan kembali *string* 0 dan 1 menjadi karakter semula, maka setiap karakter tidak boleh mempunyai kode yang merupakan awalan bagi kode yang lain.



Gambar 9.27 Pohon biner dari kode prefiks { 000, 001, 01, 10, 11}

Terapan lain dari kode awalan adalah untuk pembentukan kode Huffman dalam pemampatan data (*data compression*). Kode Huffman dijelaskan di bawah ini.

9.13 Kode Huffman

Dalam komunikasi data, pesan (*message*) yang dikirim seringkali ukurannya sangat besar sehingga waktu pengirimannya lama. Begitu juga dalam penyimpanan data, arsip (*file*) yang berukuran besar memakan ruang penyimpanan yang besar. Kedua masalah ini dapat diatasi dengan mengkodekan pesan atau isi arsip sesingkat mungkin, sehingga waktu pengiriman pesan juga relatif cepat, dan ruang penyimpanan yang dibutuhkan juga sedikit. Cara pengkodean seperti ini disebut pemampatan (*compression*) data. Pemampatan data dilakukan dengan mengkodekan setiap karakter di dalam pesan atau di dalam arsip dikodekan dengan kode yang lebih pendek.Sistem kode yang banyak digunakan adalah kode ASCII. Dengan kode ASCII, setiap karakter dikodekan dalam 8 bit biner. Tabel 9.3 berikut adalah contoh kode ASCII untuk beberapa karakter:

Tabel 9.3 Kode ASCII

Simbol	Kode ASCII
A	01000001
B	01000010
C	01000011
D	01000100

Dengan mengikuti ketentuan pengkodean di atas, *string 'ABACCDA'* direpresentasikan menjadi rangkaian bit:

01000001010000010010000010100000110100000110100010001000001

Jadi, dengan sistem pengkodean ASCII, representasi 7 huruf membutuhkan 7 × 8 = 56 bit (7 *byte*). Untuk meminimumkan jumlah bit yang dibutuhkan, panjang kode untuk setiap karakter sedapat mungkin diperpendek, terutama untuk karakter yang kekerapan (*frequency*) kemunculannya besar. Pemikiran seperti inilah yang mendasari munculnya kode Huffman. Misalnya pada pesan *'ABACCDA'*, kekerapan kemunculan *A* adalah 3, kekerapan *B* adalah 1, kekerapan *C* adalah 2, dan kekerapan *D* adalah 1, sehingga dapat dibuat Tabel 9.4 berikut:

Tabel 9.4 Tabel kekerapan 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 menggunakan kode Huffman di dalam Tabel 9.4, pesan "*ABACCDA*" direpresentasikan menjadi rangkaian bit:

0110010101110

Jadi, dengan menggunakan kode Huffman ini, jumlah bit yang dibutuhkan untuk *string* '*ABACCDA*' hanya 13 bit. Simbol-simbol yang sering muncul di dalam *string* direpresentasikan dengan kode yang lebih pendek daripada kode untuk simbol yang jarang muncul. Kode untuk setiap simbol tidak boleh merupakan awalan dari kode yang lain sebab akan menimbulkan keraguan (*ambigou*) dalam proses pemulihannya (*decoding* - yaitu mengubah kembali kode biner ke simbol asalnya). Oleh karena itu, kode Huffman adalah kode prefiks.

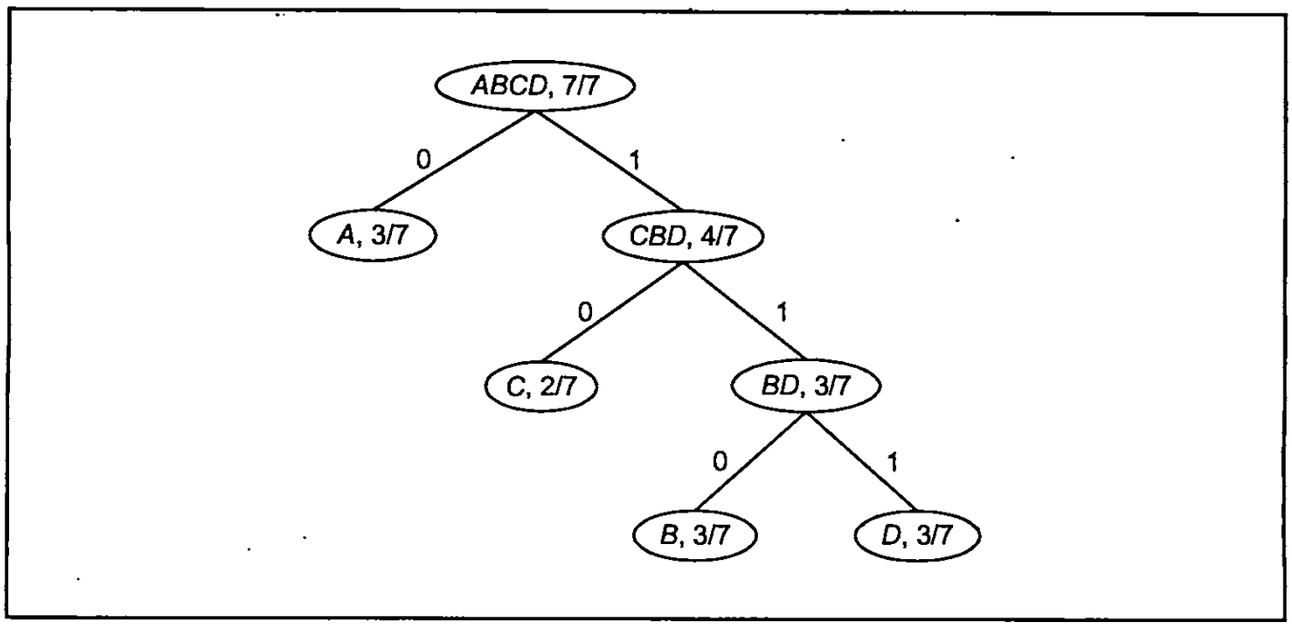
Bagaimana cara mendapatkan kode Huffman?

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.



Gambar 9.28 Pohon Huffman untuk pesan '*ABACCDA*'

Daun pada pohon Huffman menyatakan simbol yang digunakan dalam teks atau pesan. Kode setiap simbol dengan memberi label 0 pada setiap cabang (sisi) kiri dan label 1 untuk setiapcangan kanan. Pohon Huffman untuk *string 'ABACCDA'* ditunjukkan pada Gambar 9.28.

Dengan membuat lintasan dari akar ke daun, akan dihasilkan kode untuk setiap simbol. Dari Gambar 9.28 diperoleh kode untuk masing-masing simbol asal sebagai berikut:

$$A = 0, \qquad B = 110, \qquad C = 10, \qquad D = 111$$

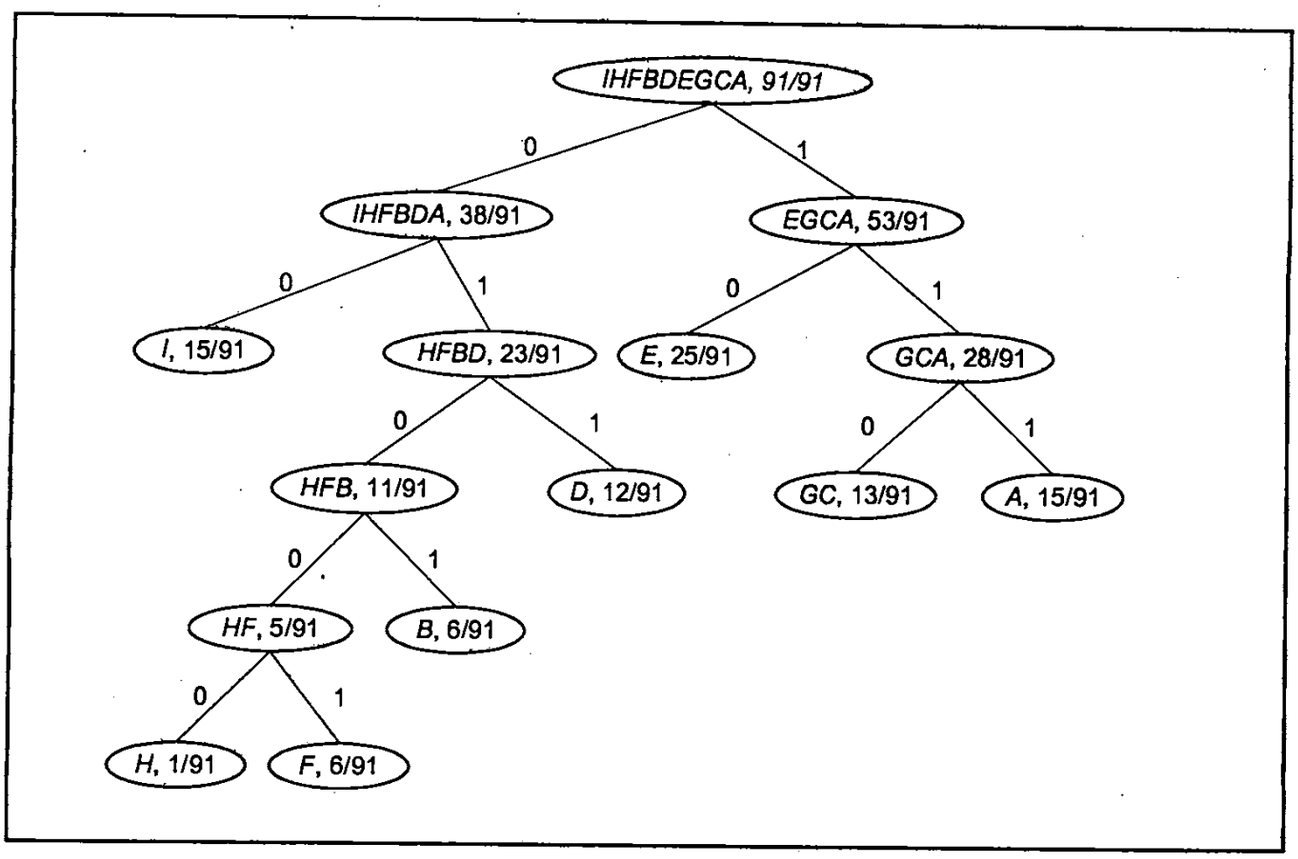
Perhatikan bahwa simbol yang paling sering muncul memiliki kode dengan jumlah bit paling sedikit. Perhatikan pula bahwa tidak ada simbol yang kodenya merupakan kode awalan untuk simbol yang lain. Dengan cara ini, kode yang diawali 0 dapat dipastikan adalah A, sedangkan kode yang diawali 1 mungkin B, C, atau D, yang harus diperiksa lagi dari bit-bit selanjutnya.

Perhatikanlah bahwa kode Huffman tidak bersifat unik, artinya kode untuk setiap karakter berbeda-beda pada setiap pesan bergantung pada kekerapan kemunculan karakter tersebut di dalam pesan. Selain itu, keputusan apakah suatu simpul pada pohon Huffman diletakkan di kiri atau di kanan juga menentukan kode yang dihasilkan (tetapi tidak mempengaruhi panjang kodenya)

Contoh 9.11

Misalkan dari sebuah pesan sudah dihitung kekerapan kemunculan setiap simbol karakter seperti diperlihatkan pada Tabel 9.5. Pohon Huffman-nya ditunjukkan pada Gambar 9.29.

Simbol	Kekerapan	Peluang	Kode Huffman
A	15	15/91	111
B	6	6/91	0101
C	7	7/91	1101
D	12	12/91	011
E	25	25/91	10
F	4	4/91	01001
G	6	6/91	1100
H	1	1/91	01000
I	15	15/91	00



Gambar 9.29 Pohon Huffman untuk Tabel 9.5

9.14 Pohon Pencarian

Pohon pencarian biner (*binary search tree* - BST) mungkin adalah pohon biner yang paling penting, khususnya pada persoalan yang banyak melakukan operasi pencarian, penyisipan, dan penghapusan elemen. Untuk operasi semacam itu, pohon pencarian biner memiliki kinerja yang lebih baik daripada struktur data lain, yang dalam hal ini waktu pencarian.

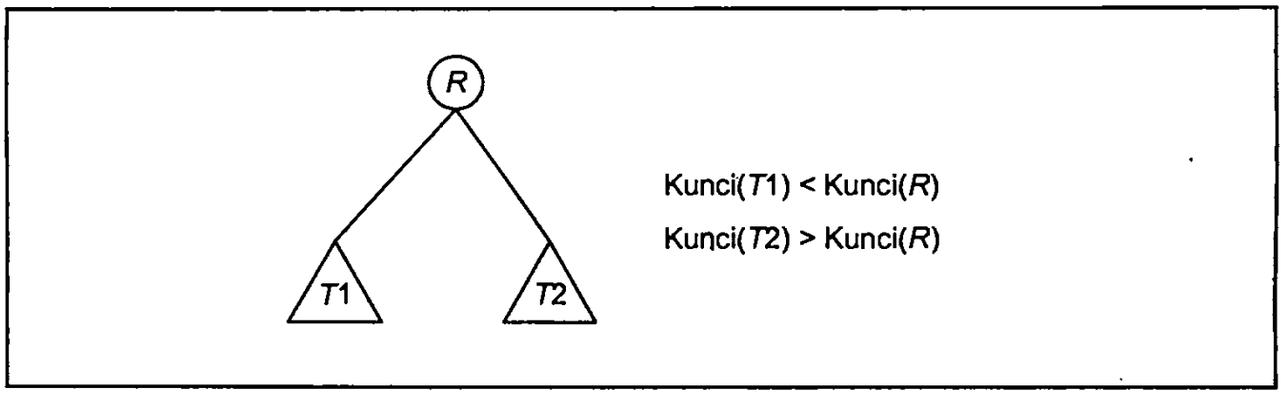
Simpul pada pohon pencarian dapat berupa *field* kunci (*kunci*) pada data *record*, atau data itu sendiri (dengan catatan setiap data adalah unik). Kunci adalah nilai yang membedakan setiap simpul dengan simpul yang lainnya. Kunci harus unik, karena itu tidak ada dua buah simpul atau lebih yang mempunyai kunci yang sama.

Pohon pencarian biner adalah pohon biner yang setiap kuncinya diatur dalam suatu urutan tertentu. Ketentuan pengaturan kunci adalah sebagai berikut:

Jika *R* adalah akar, dan semua kunci yang tersimpan pada setiap simpul tidak ada yang sama, maka (lihat Gambar 9.30):

(a) semua simpul pada upapohon kiri mempunyai kunci lebih kecil dari Kunci(*R*)

(b) semua simpul di upapohon kanan mempunyai kunci nilai lebih besar dari Kunci(R)

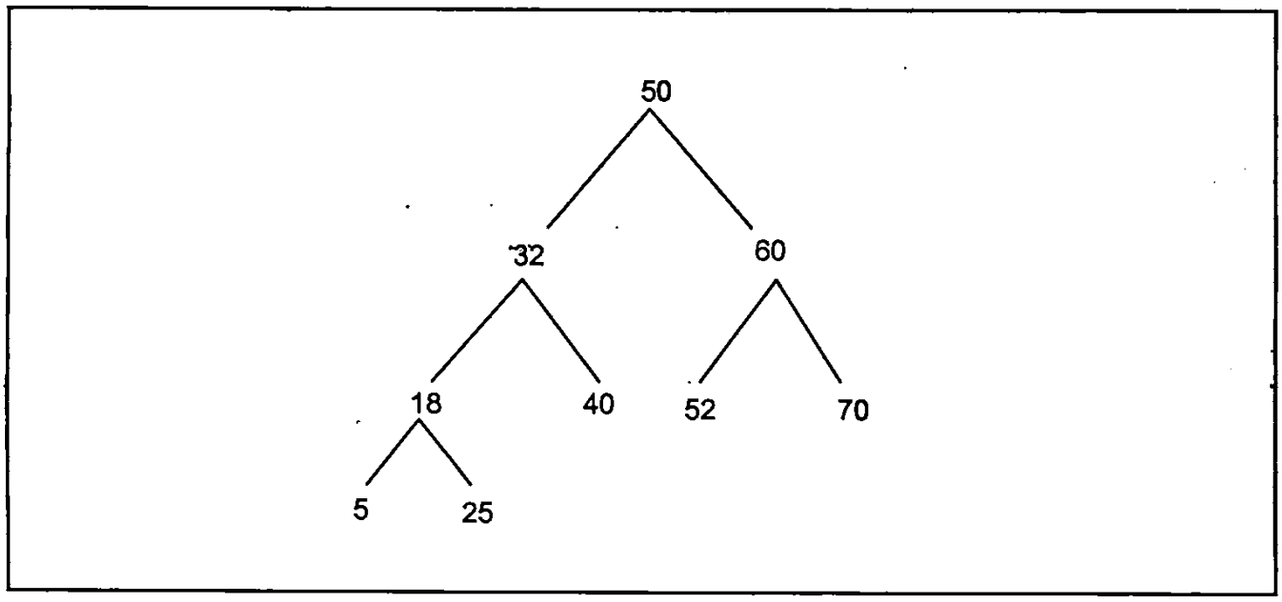


Gambar 9.30 Skema pohon pencarian

Contoh 9.12

Gambar 9.31 adalah sebuah pohon pencarian untuk data masukan dengan urutan sebagai berikut:

50, 32, 18, 40, 60, 52, 5, 25, 70. ■



Gambar 9.31 Contoh pohon pencarian

Perhatikanlah dari Gambar 9.31 di atas, bahwa simpul di upapohon kiri 50 mempunyai kunci lebih kecil dari 50, dan simpul di upapohon kanan 50 mempunyai kunci lebih besar dari 50. Secara rekursif, simpul lainnya juga mempunyai pola demikian.

Sesuai namanya, struktur pohon pencarian dimaksudkan untuk memberikan pengaksesan yang cepat terhadap data di simpul. Pencarian data di pohon pencarian dilakukan melalui kunci. Pencarian selalu dimulai dari simpul akar. Kunci di simpul akar dibandingkan dengan nilai yang dicari (x). Jika kunci di simpul akar tidak sama dengan x, pencarian dilanjutkan di upapohon kiri atau di upapohon kanan, bergantung pada apakah x lebih kecil dari kunci di akar atau x lebih besar daripada kunci di akar. Pembandingan demikian diteruskan sampai x sama dengan nilai suatu kunci atau tercapai sebuah daun.

9.15 Traversal Pohon Biner

Operasi dasar yang sering dilakukan pada pohon biner ialah mengunjungi (*traversal*) setiap simpul tepat satu kali. Misalkan T adalah pohon biner, akarnya R, upapohon kiri T_1 dan upapohon kanan T_2 (Gambar 9.32).

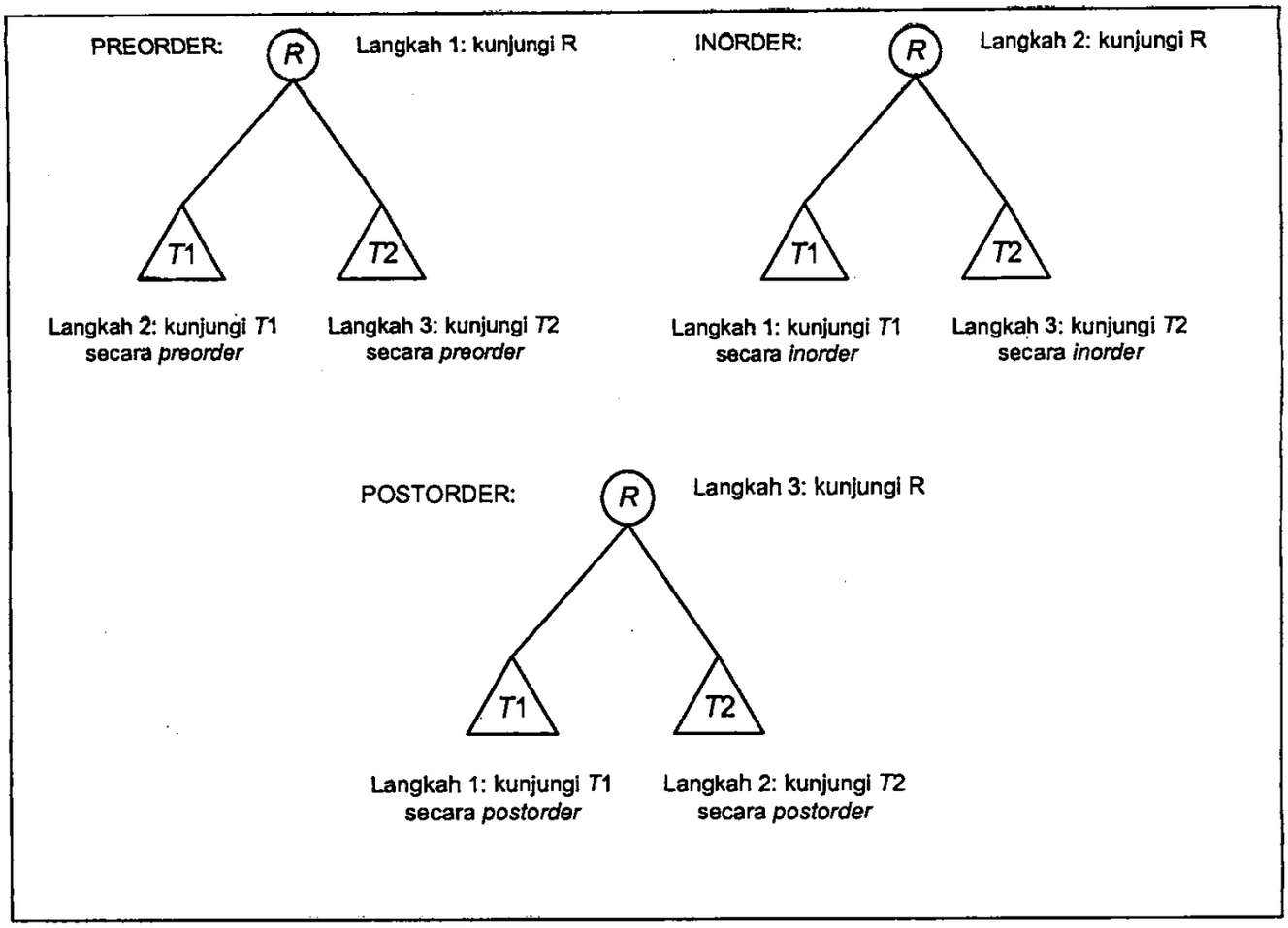
Ada tiga macam skema mengunjungi simpul-simpul di dalam pohon biner T:

1. *Preorder*
   (i) Kunjungi R (sekaligus memproses simpul R)
   (ii) Kunjungi T_1 secara *preorder*
   (iii) Kunjungi T_2 secara *preorder*

2. *Inorder*
   (i) Kunjungi T_1 secara *inorder*
   (ii) Kunjungi R (sekaligus memproses simpul R)
   (iii) Kunjungi T_2 secara *inorder*

3. *Postorder*
   (i) Kunjungi T_1 secara *postorder*
   (ii) Kunjungi T_2 secara *postorder*
   (iii) Kunjungi R (sekaligus memproses simpul R)

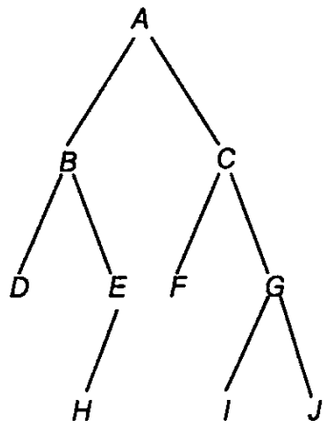
Proses yang dilakukan terhadap simpul yang dikunjungi misalnya mencetak informasi yang disimpan di dalam simpul, memanipulasi nilai, dan sebagainya.



Gambar 9.32 Skema mengunjungi pohon biner

Contoh 9.13

Tinjau pohon biner T di bawah ini:

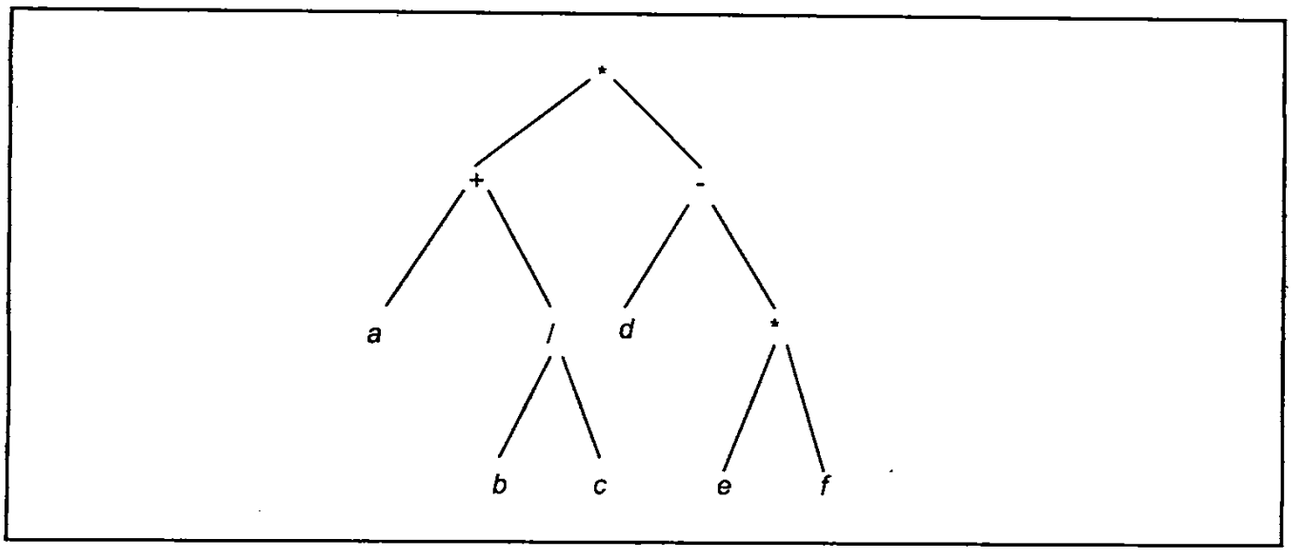


Lintasan *preorder*, *inorder*, dan *postorder* dari T adalah:

preorder: $A, B, D, E, F, C, F, G, I, J$
inorder: $D, B, H, E, A, F, C, I, G, J$
potsorder: $D, H, E, B, F, I, J, G, C, A$ ■

Ada korespondensi antara ketiga cara mengunjungi pohon biner dengan notasi *prefix*, *infix*, dan *postfix*. Misalkan pohon yang dikunjungi adalah pohon ekspresi pada Gambar 9.33. Penelusuran pohon secara *preorder*, *inorder*, dan *postorder* masing-masing menghasilkan ekspresi dalam notasi *prefix*, *infix*, dan *postfix* sebagai berikut:

preorder	: * + a / b c - d * e f	(*prefix*)
inorder	: a + b / c * d - e * f	(*infix*)
postorder	: a b c / + d e f * - *	(*postfix*)



Gambar 9.33 Pohon ekspresi

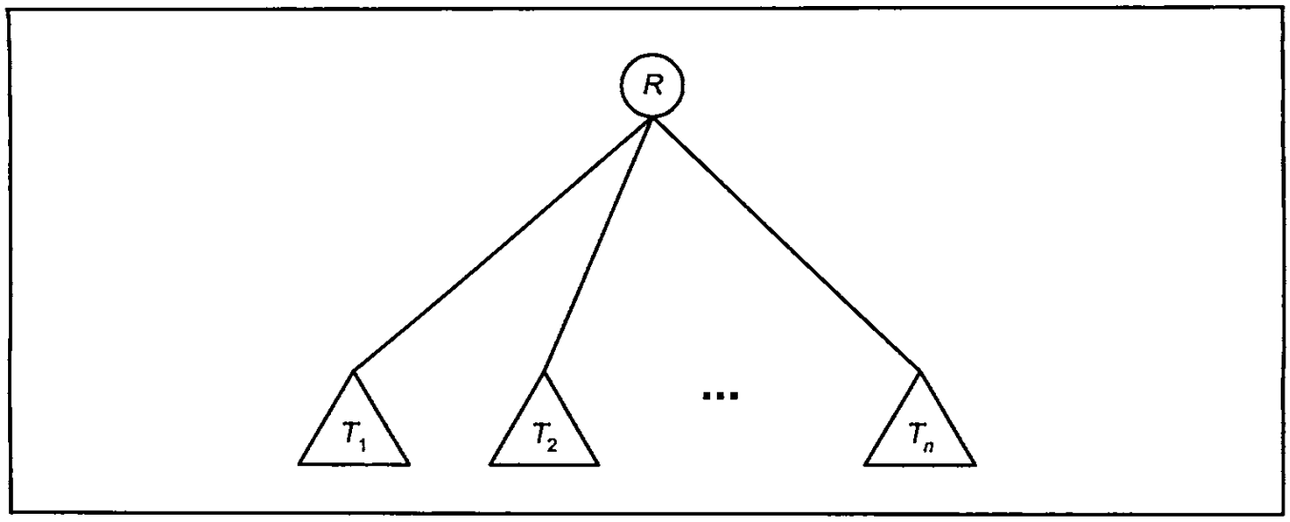
Preorder, *Inorder*, dan *Postorder* pada Pohon *m-ary*

Skema *preorder*, *postorder*, dan *inorder* pada pohon biner dapat dirampatkan sehingga berlaku untuk sembarang pohon *m-ary*. Misalkan T adalah pohon *m-ary*, akarnya R, dan upapohonnya adalah T_1, T_2, ..., T_n (Gambar 9.34). Traversal *preorder*, *inorder* dan *postorder* pada pohon *m-ary* didefinisikan sebagai berikut:

1. *Preorder*
   (i) Kunjungi R
   (ii) Kunjungi T_1, T_2, ..., T_n secara *preorder*

2. *Inorder*
   (i) Kunjungi T_1 secara *inorder*
   (ii) Kunjungi R
   (iii) Kunjungi T_2, T_3, ..., T_n secara *inorder*

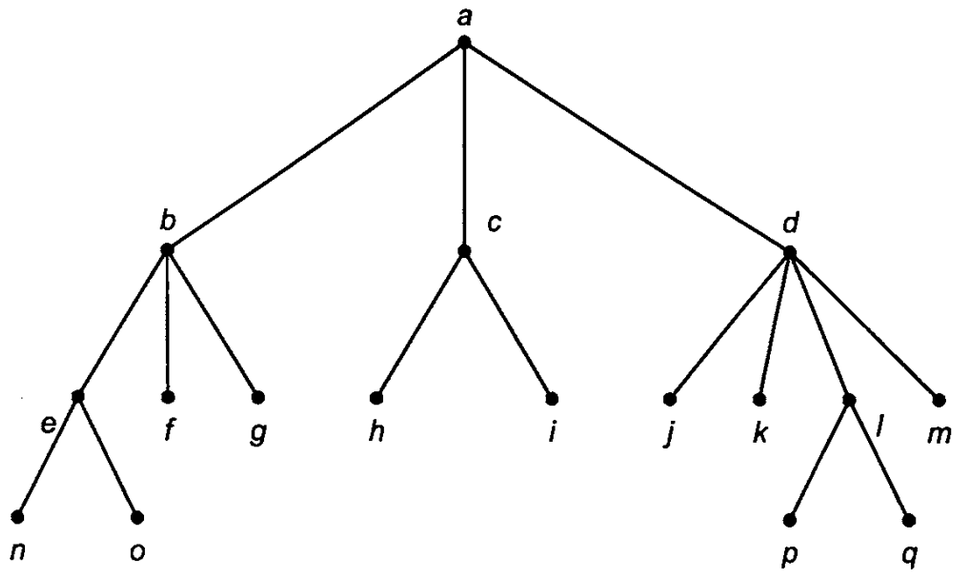
3. *Postorder*
   (i) Kunjungi T_1, T_2, ..., T_n secara *postorder*
   (ii) Kunjungi R



Gambar 9.34 Skema pohon *m-ary*

Contoh 9.14

Tentukan hasil kunjungan *preorder*, *inorder*, dan *postorder* pada pohon 4-*ary* berikut ini:



Penyelesaian:
Lintasan hasil traversal:

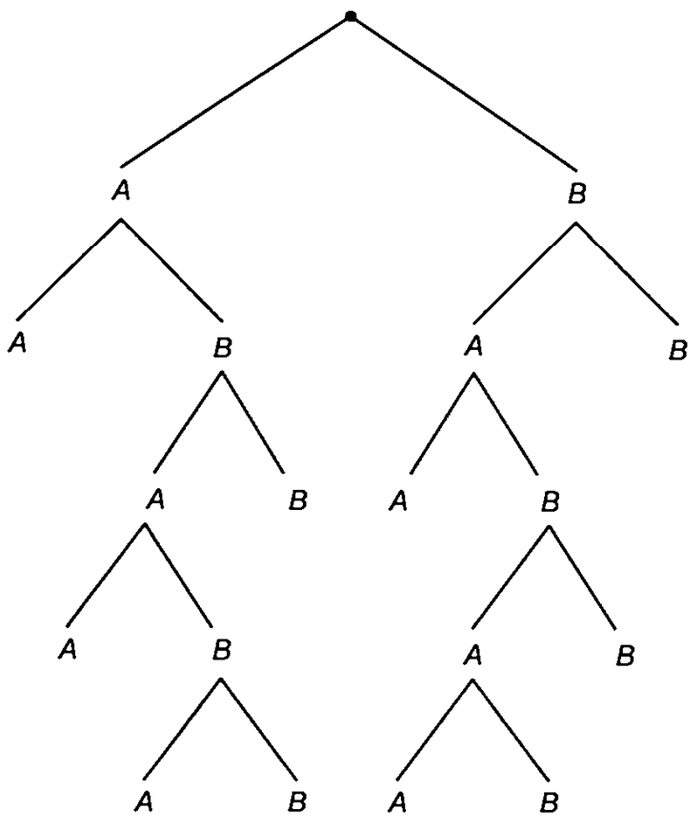
preorder: *a, b, e, n, o, f, g, c, h, i, d, j, k, l, p, q, m*
inorder: *n, e, o, b, f, g, a, h, c, i, j, d, k, p, l, q, m*
postorder: *n, o, e, f, g, b, h, i, c, j, k, p, q, l, m, d, a* ■

9.16 Ragam Soal dan Pembahasan

Contoh 9.15

[LIP92] Gunakan pohon berakar untuk menggambarkan semua kemungkinan hasil dari pertandingan tenis antara dua orang pemain, Anton dan Budi, yang dalam hal ini pemenangnya adalah pemain yang pertama memenangkan dua set berturut-turut atau pemain yang pertama memenangkan total tiga set.

Penyelesaian:
Pohon berakar yang menggambarkan semua kemungkinan hasil pertandingan tenis untuk mencari pemenang pertandingan tenis ditunjukkan di bawah ini (A = Anton, B = Budi):

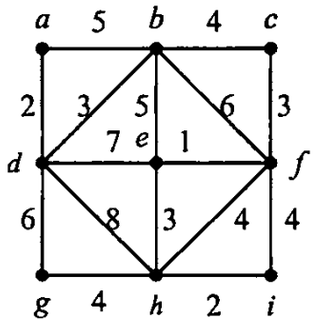


Setiap simpul, kecuali akar, menyatakan pemenang pertandingan. Setiap lintasan dari akar ke daun menyatakan kejadian (yaitu, siapa yang memenangkan set) yang menuju ke suatu hasil pertandingan. Ada 10 daun dan 10 lintasan yang menyatakan kemungkinan hasil pertandingan, yaitu

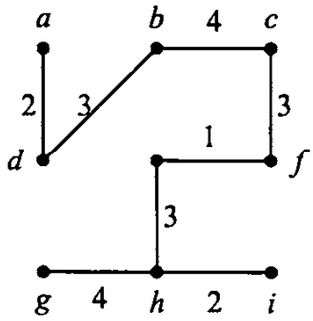
AA, ABAA, ABB, ABABA, ABABB, BAA, BABAA, BABAB, BABB, BB ■

Contoh 9.16

Tentukan dan gambarkan pohon merentang minimum dari graf di bawah ini (tahapan pembentukannya tidak perlu ditulis).



Penyelesaian:
Pohon merentang minimumnya adalah seperti di bawah ini:



Bobot pohon merentang minimum: 1 + 3 + 3 + 2 + 4 + 4 + 3 + 2 = 22. ■

Contoh 9.17

Sebuah pohon 4-*ary* tingginya 5. Tanpa menggambarkan pohonnya, hitung jumlah maksimum seluruh simpul di dalam pohon tersebut. Berapa jumlah maksimum seluruh daunnya?

Penyelesaian:
Pohon 4-ary dengan tinggi 5 berarti $m = 4$ dan $h = 5$. Jumlah maksimum seluruh simpul

$$S \leq \frac{m^{h+1}-1}{m-1} = \frac{4^{5+1}-1}{4-1} = 1365 \text{ simpul}$$

sedangkan jumlah maksimum seluruh daun $\leq m^h = 4^5 = 1024$ daun. ■

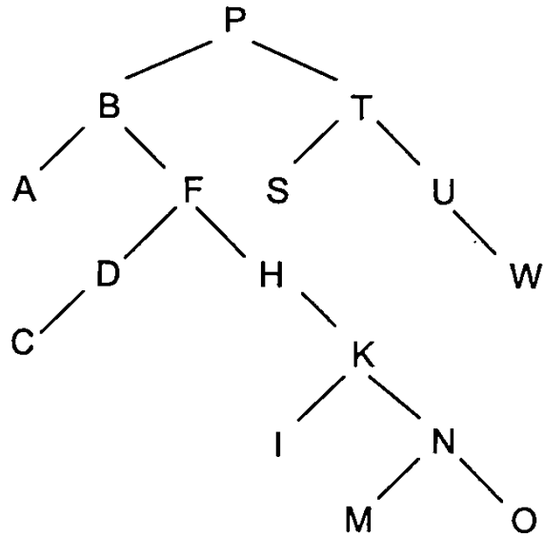
Contoh 9.18

Diberikan masukan berupa rangkaian karakter dengan urutan sebagai berikut:

P, T, B, F, H, K, N, S, A, U, M, I, D, C, W, O

(a) Gambarkan pohon pencarian (*search tree*) yang terbentuk.
(b) Tentukan hasil penelusuran *preorder*, *inorder*, dan *postorder*, dari pohon jawaban (a) di atas.

Penyelesaian:
Pohon pencarian biner yang terbentuk adalah:



Preorder : *P, B, A, F, D, C, H, K, I, N, M, O, T, S, U, W*
Inorder : *A, B, C, D, F, H, I, K, M, N, O, P, S, T, U, W*
Postorder: *A, C, D, I, M, O, N, K, H, F, B, S, W, U, T, P* ■

Contoh 9.19

Sebuah turnamen catur diikuti oleh 5000 orang peserta. Berapa banyak pertandingan yang harus diadakan sampai ditemukan seorang juara jika turnamen menggunakan sistem gugur, yaitu peserta yang kalah tidak pernah bertanding lagi, dan peserta yang menang akan melawan pemenang pertandingan lainnya? Graf apa yang terbentuk?

Penyelesaian:
Persoalan dapat dimodelkan dengan pohon biner penuh. Daun menyatakan pemain, simpul dalam (termasuk akar) menyatakan pemenang pertandingan. Jumlah simpul dalam berarti total banyak pertandingan yang dilakukan sampai menjadi juara.

i = banyaknya simpul dalam
t = banyaknya simpul daun

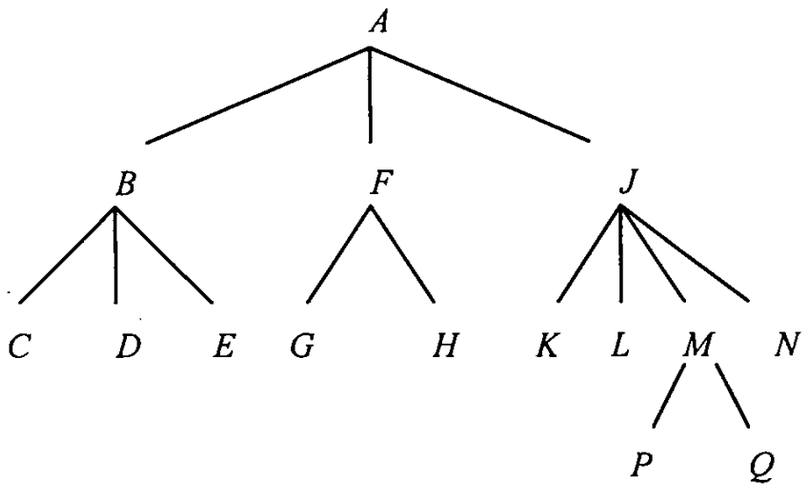
Setiap pertandingan menggugurkan seorang pemain, dan pada akhirnya hanya ada satu pemenang. Maka berlaku hubungan:

$$i = t - 1$$

Jadi banyaknya pertandingan adalah 5000 – 1 = 4999 pertandingan. Graf yang terbentuk adalah pohon biner penuh. ■

Contoh 9.20

Tentukan hasil penelusuran *preorder* dan *postorder* dari pohon 4-*ary* berikut ini:



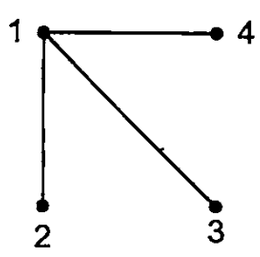
Penyelesaian:

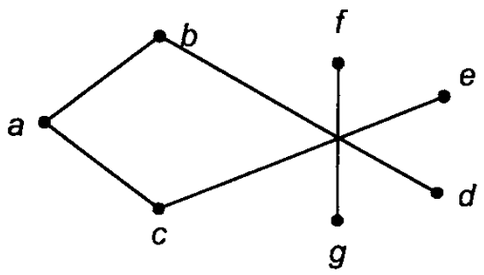
Preorder : *A, B, C, D, E, F, G, H, J, K, L, M, P, Q, N*

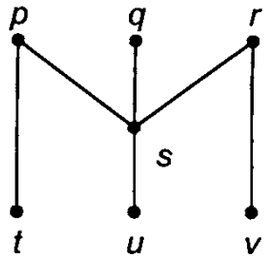
Postorder : *C, D, E, B, G, H, F, K, L, P, Q, M, N, J, A* ■

Soal Latihan

1. Manakah di antara ketiga buah graf di bawah ini yang merupakan pohon?







2. Misalkan T sebuah pohon dengan 50 buah sisi. Pembuangan suatu sisi tertentu dari T menghasilkan dua pohon T_1 dan T_2 yang terpisah satu sama lain. Jika banyaknya simpul di dalam T_1 sama banyak dengan banyaknya simpul di dalam T_2, tentukan jumlah simpul dan sisi di dalam T_1 dan T_2.

3. Tunjukkan bahwa jumlah derajat semua simpul di dalam pohon dengan n simpul adalah $2n - 2$.

4. Tunjukkan bahwa sebuah pohon biner teratur mempunyai sejumlah ganjil simpul.

5. Gambarkan semua pohon rentang dari graf lengkap dengan 4 buah simpul.

6. Berapa banyak sisi harus dibuang dari graf terhubung dengan n buah simpul dan m buah untuk menghasilkan pohon rentang?

7. Apakah lintasan Hamilton di dalam sebuah graf terhubung merupakan pohon merentang? Jelaskan jawaban anda.

8. Apakah semua pohon merentang T dari graf terhubung G harus mengandung jumlah sisi yang sama? Jelaskan alasannya (bukan dengan contoh).

9. Apakah semua pohon planar? Jika ya, jelaskan mengapa? Jika tidak, dapatkah anda temukan pohon yang tidak planar?

10. 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.

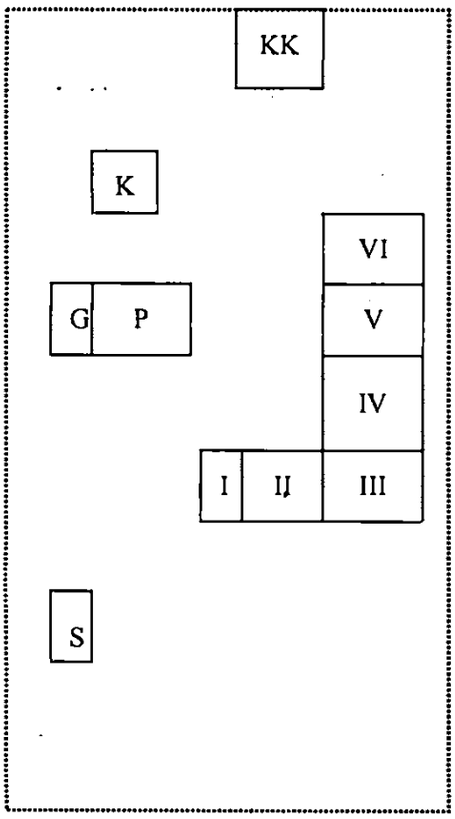
11. Bangunlah pohon Huffman untuk string '*makan malam di mana*' (tidak termasuk tanda petik) dengan ketentuan: simbol dengan peluang lebih kecil sebagai anak kiri dan simbol dengan peluang lebih besar sebagai anak kanan, sisi kiri

dilabeli dengan 0 dan sisi kanan dengan 1. Tentukan kode Huffman untuk setiap karakter, dan tentukan representasi biner dari string yang telah dimampatkan itu.

12. Gambarkan pohon ekspresi untuk (a) $b^2 - 4ac$ dan (b) $((a_3x + a_2)x + a_1)x + a_0$

13. (a) Gambarkan sebuah pohon biner dengan tujuh simpul dan hanya satu buah daun.
    (b) Gambarkan sebuah pohon biner dengan tujuh simpul

14. Pada alam hari harus dipasang lampu-lampu khusus di gedung-gedung utama sebuah sekolah dasar. Lampu-lampu itu dinyalakan dan dimatikan dari gardu Satpam sekolah. Diagram di sebelah kiri menunjukkan tata letak sekolah itu. Digunakan simbol-simbol I, II, III, IV, V, dan VI untuk kelas 1 sampai kelas 6, KK untuk kamar kecil, P untuk perpustakaan, G untuk ruang guru, K untuk kantin, dan S untuk gardu Satpam. Tabel di sebelah kanan menunjukkan panjang kabel listrik dalam satuan meter antar setiap lampu yang dipasang.
    (a) Gambarkan graf yang memodelkan persoalan.
    (b) Bagaimana cara menghubungkan lampu-lampu itu satu dengan yang lain dengan menggunakan kabel listrik yang panjangnya *minimum*? Berapa panjang minimum kabel yang diperlukan? (dalam anda menjawab soal ini, jelaskan pendekatan/metode yang anda gunakan).



Panjang kabel listrik antar ruangan

	I	II	III	IV	V	VI	G	P	K	KK	S
I	-	25	-	-	-	-	80	60	-	-	50
II	25	-	45	-	-	-	-	65	-	-	-
III	-	45	-	20	-	-	-	55	-	-	-
IV	-	-	20	-	47	-	-	60	-	-	-
V	-	-	-	47	-	40	-	45	-	-	-
VI	-	-	-	-	40	-	-	-	30	60	-
G	80	-	-	-	-	-	-	55	90	-	85
P	60	65	55	60	45	-	55	-	60	-	-
K	-	-	-	-	-	30	90	60	-	90	-
KK	-	-	-	-	-	60	-	-	90	-	-
S	50	-	-	-	-	-	85	-	-	-	-

15. Diberikan kode Huffman sebagai berikut: *a*:001, *b*:0001, *e*:1, *r*:0000, *s*:0100, *t*:011, *x*:01010.
    (a) Gambarkan pohon Huffman yang merepresentasikan kode tersebut.
    (b) *Decode* rangkaian bit berikut: 011110010100110001001

16. Sebuah surat berantai dimulai ketika seseorang mengirim sebuah surat kepada 5 orang lainnya. Tiap orang yang menerima surat mengirimkan surat tersebut kepada 5 orang lain yang belum pernah menerima surat tersebut atau kepada orang belum pernah mengirimkannya ke orang lain. Misalkan ada 10.000 orang yang mengirimkan surat tersebut sebelum rantai berakhir. Berapa banyak orang yang menerima surat tersebut dan berapa banyak yang tidak mengirimkannya?

17. Perhatikan situasi bisnis khusus ini. Sebuah perusahaan yang menjual produk-produknya di dua wilayah geografis utama merencanakan untuk memperkenalkan sebuah produk baru. Prosedur normal untuk pengenalan produk adalah sebagai berikut: Pertama, produk diperkanalkan dalam sebuah uji coba (pasar kecil) di wilayah I. Jika produk gagal, ini tidak dilanjutkan; jika ini sukses maka produk diperkenalkan ke seluruh wilayah I. Jika produk sukses di wilayah I, produk diperkenalkan ke seluruh wilayah II; jika tidak, produk diperkenalkan dalam sebuah uji coba (pasar kecil) di wilayah II. Lagi, jika ini berjalan suskes, produk akan diperkenalkan ke seluruh wilayah. Gunakan pohon berakar untuk menentukan semua kemungkinan hasil dari prosedur pengenalan produk.

Esensi dari matematika adalah kebebasannya.
(George Cantor)